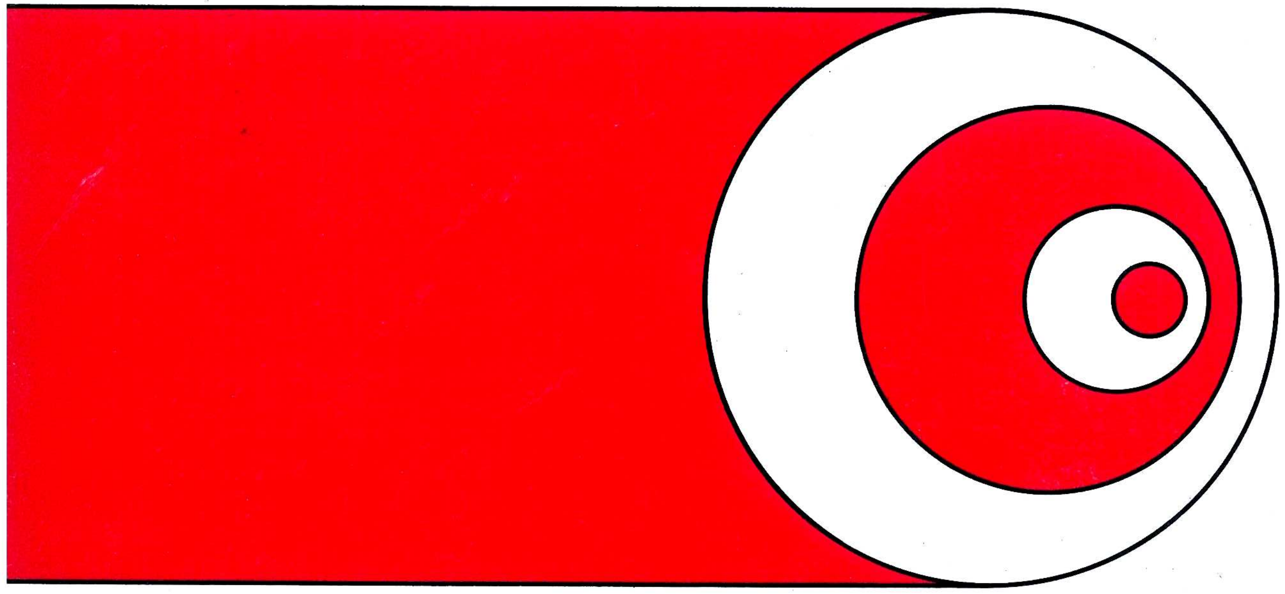
SIGNETICS
2650 MICROPROCESSOR

HARDWARE SPECIFICATIONS



2650
HARDWARE SPECIFICATIONS MANUAL

PREFACE

This manual contains the complete specifications for the Signetics 2650 processor. It describes the instruction set, interface signals, the internal organization, and the electrical characteristics. Examples of memory and I/O system organizations that may be used with the processor are discussed.

CONTENTS

CHAPTER I

INTRODUCTION

FEATURES

GENERAL PURPOSE PROCESSOR
SINGLE CHIP
FIXED INSTRUCTION SET
PARALLEL 8-BIT BINARY OPERATIONS
40 PIN DUAL IN-LINE PACKAGE

N-CHANNEL SILICON GATE MOS TECHNOLOGY
TTL COMPATIBLE INPUTS AND OUTPUTS
SINGLE POWER SUPPLY OF +5 VOLTS
SEVEN GENERAL PURPOSE REGISTERS
RETURN ADDRESS STACK, 8 DEEP, ON CHIP

32K BYTE ADDRESSING RANGE
SEPARATE ADDRESS AND DATA LINES
VARIABLE LENGTH INSTRUCTIONS OF 1, 2, OR 3 BYTES
75 INSTRUCTIONS
MACHINE CYCLE TIME OF 2.4μsec
AT CLOCK FREQUENCY OF 1.25 MHz

DIRECT INSTRUCTIONS TAKE 2, 3 or 4 CYCLES
SINGLE PHASE TTL LEVEL CLOCK INPUT
STATIC LOGIC
TRI-STATE OUTPUT BUSSES
REGISTER, IMMEDIATE, RELATIVE, ABSOLUTE
INDIRECT, AND INDEXED ADDRESSING MODES
VECTOR INTERRUPT FORMAT

GENERAL FEATURES

The 2650 processor is a general purpose, single chip, fixed instruction set, parallel 8-bit binary processor. A general purpose processor can perform any data manipulations through execution of a stored sequence of machine instructions. The processor has been designed to closely resemble conventional binary computers, but executes variable length instructions of one to three bytes in length. BCD Arithmetic is made possible through use of a special "DAR" machine instruction.

The 2650 is manufactured using Signetics' N-channel silicon gate MOS technology. N-channel provides high carrier mobility for increased speed and also allows the use of a single 5 volt power supply. Silicon gate provides for better density and speed. Standard 40 pin dual in-line packages are used for the processor.

The 2650 contains a total of seven general purpose registers, each eight bits long. They may be used as source or destination for arithmetic operations, as index registers, and for I/O transfers.

The processor can address up to 32,768 bytes of memory in four pages of 8,192 bytes each. The processor instructions are one, two, or three bytes long, depending on the instruction. Variable length instructions tend to conserve memory space since a one-or two-byte instruction may often be used rather than a three byte instruction. The first byte of each instruction always specifies the operation to be performed and the addressing mode to be used. Most instructions use six of the first eight bits for this purpose, with the remaining two bits forming the register field. Some instructions use the full eight bits as an operation code.

The most complex direct instruction is three bytes long and takes 9.6 microseconds to execute. This figure assumes that the processor is running at its maximum clock rate, and has an associated memory with cycle and access times of one microsecond or less. The fastest instruction executes in 4.8 microseconds.

The clock input to the processor is a single phase pulse train and uses only one interface pin. It requires a normal TTL voltage swing, so no special clock driver is required.

The Data Bus and Address signals are tri-state to provide convenience in system design. Memory and I/O interface signals are asynchronous so that Direct Memory Access (DMA) and multiprocessor operations are easy to implement.

The 2650 has a versatile set of addressing modes used for locating operands for operations. They are described in detail in the INSTRUCTIONS section of this manual.

The interrupt mechanism is implemented as a single level, address vectoring type. Address vectoring means that an interrupting device can force the processor to execute code at a device determined location in memory. The interrupt mechanism is described in detail in the FEATURES section of this manual.

APPLICATIONS

The ability of the semi-conductor industry to manufacture complete general purpose processors on single chips represents a significant technological advance which should prove to be of great benefit to digital systems manufacturers. In terms of chip size and density of transistors, the processors are simply extensions of the continually evolving MOS technology. But in terms of function provided, a significant threshold has been crossed.

By allowing designers to convert from hardware logic to programmed logic, the integrated processor provides several important advantages.

1. Logic functions may be implemented in memory bits instead of logic gates. The user then has greater access to the advantages of memory circuits. Memories use patterned circuitry and thus provide greater density and therefore greater economy.
2. Random logic implementations of complex functions are highly specialized and cannot be used in other applications. They are not often used in large volume. Programmed logic, on the other hand, relies on general purpose processor and memory circuits that are used in many applications. Thus, economies of volume are available for both the user and the manufacturer.
3. Because the functional specialization resides in the user's program rather than the hardware logic, changes, corrections and additions can be much easier to make and can be accomplished in a much shorter time.
4. With the programmed logic approach it is often possible to add new features and create new products simply by writing new programs.
5. The design cycle of a system using programmed logic can be significantly shorter than a similar system that attempts to use custom random logic. The debugging cycle is also greatly compressed.

A general purpose processor designed to implement programmed logic has many characteristics that allow it to do conventional computer operations as well. Many applications will specialize in programmed logic or in data processing, but some will take advantage of both areas. In a line printer application, for example, a processor can act primarily as a controller handling the housekeeping duties, control sequencing and data interfacing for the printer. It also might buffer the data or do some code conversions, but that is not its primary duty. On the other hand, in a text editing intelligent terminal, the processor is mainly concerned with data manipulation since it handles code translations, display paging, insertions, deletions, line justification, hyphenation, etc.

A point-of-sale type of terminal represents an application that combines both control and data processing activities for the processor. Coordinating the activities of the various devices and displays that make up the terminal is an important part of the job, as are the calculations that are essential to the operation of the machine.

CHAPTER II

INTERNAL ORGANIZATION

INTERNAL REGISTERS

The block diagram for the 2650 shows the major internal components and the data paths that interconnect them. In order for the processor to execute an instruction, it performs the following general steps:

1. The Instruction Address Register provides an address for memory.
2. The first byte of an instruction is fetched from memory and stored in the Instruction Register.
3. The Instruction Register is decoded to determine the type of instruction and the addressing mode.
4. If an operand from memory is required, the operand address is resolved and loaded into the Operand Address Register.
5. The operand is fetched from memory and the operation is executed.
6. The first byte of the next instruction is fetched.

The Instruction Register (IR) holds the first byte of each instruction and directs the subsequent operations required to execute each instruction. The IR contents are decoded and used in conjunction with the timing information to control the activation and sequencing of all the other elements on the chip. The Holding Register (HR) is used in some multiple-byte instructions to contain further instruction information and partial absolute addresses.

The Arithmetic Logic Unit (ALU) is used to perform all of the data manipulation operations, including Load, Store, Add, Subtract, And, Inclusive Or, Exclusive Or, Compare, Rotate, Increment and Decrement. It contains and controls the Carry bit, the Overflow bit, the Interdigit Carry and the Condition Code Register.

The Register Stack contains six registers that are organized into two banks of three registers each. The Register Select bit (RS) picks one of the two banks to be accessed by instructions. In order to accomodate the register-to-register instructions, register zero (RO) is outside the array. Thus, register zero is always available along with one set of three registers.

The Address Adder (AA) is used to increment the instruction address and to calculate relative and indexed addresses.

The Instruction Address Register (IAR) holds the address of the next instruction byte to be accessed. The Operand Address Register (OAR) stores operand addresses and sometimes contains intermediate results during effective address calculations.

The Return Address Stack (RAS) is an eight level, Last In, First Out (LIFO) storage which receives the return address whenever a Branch-to-Subroutine instruction is executed. When a Return instruction is executed, the RAS provides the last return address for the processor's IAR. The stack contains eight levels of storage so that subroutines may be nested up to eight levels deep. The Stack Pointer (SP) is a three bit wraparound counter that indicates the next available level in the stack. It always points to the current address.

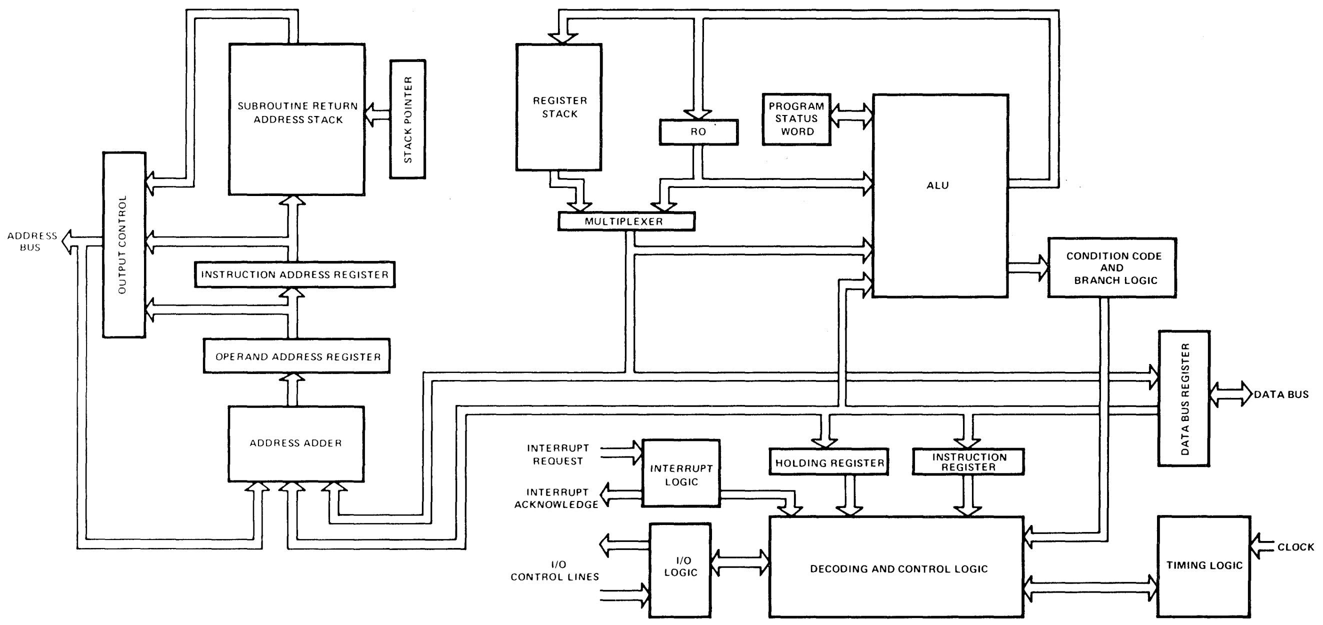


Figure II-1 SIGNETICS 2650 BLOCK DIAGRAM

PROGRAM STATUS WORD

The Program Status Word (PSW) is a special purpose register within the processor that contains status and control bits. It is 16 bits long and is divided into two bytes called the Program Status Upper (PSU) and the Program Status Lower (PSL).

The PSW bits may be tested, loaded, stored, preset or cleared using the instructions which effect the PSW. The sense bit, however, cannot be set or cleared because it is directly connected to pin #1.

PSU

7	6	5	4	3	2	1	0
S	F	II	Not Used	Not Used	SP2	SP1	SP0

S	Sense
F	Flag
II	Interrupt Inhibit
SP2	Stack Pointer Two
SP1	Stack Pointer One
SP0	Stack Pointer Zero

PSL

7	6	5	4	3	2	1	0
CC1	CC0	IDC	RS	WC	OVF	COM	C

CC1	Condition Code One
CC0	Condition Code Zero
IDC	Interdigit Carry
RS	Register Bank Select
WC	With/Without Carry
OVF	Overflow
COM	Logical/Arithmetic Compare
C	Carry/Borrow

SENSE (S)

The Sense bit in the PSU reflects the logic state of the sense input to the processor at pin #1. The sense bit is not affected by the LPSU, PPSU, or CPSU instructions. When the PSU is tested (TPSU) or stored into register zero (SPSU), bit #7 reflects the state of the sense pin at the time of the instruction execution.

FLAG (F)

The Flag bit is a simple latch that drives the Flag output (pin #40) on the processor.

INTERRUPT INHIBIT (II)

When the Interrupt Inhibit (II) bit is set, the processor will not recognize an incoming interrupt. When interrupts are enabled (II=0), and an interrupt signal occurs, the inhibit bit in the PSU is then automatically set. When a Return-and-Enable instruction is executed, the inhibit bit is automatically cleared.

STACK POINTER (SP)

The three Stack Pointer bits are used to address locations in the Return Address Stack (RAS). The SP designates the stack level which contains the current return address. The three SP bits are organized as a binary counter which is automatically incremented with execution of Branch-to-Subroutine instructions, and decremented with execution of Return instructions.

CONDITION CODE (CC)

The Condition Code is a two bit register which is set by the processor whenever a general purpose register is loaded or modified by the execution of an instruction. Additionally, the CC is set to reflect the relative value of two bytes whenever a compare instruction is executed.

The following table indicates the setting of the Condition Code whenever data is set into a general purpose register. The data byte is interpreted as an 8-bit, two's complement number.

Register Contents	CC1	CC0
Positive	0	1
Zero	0	0
Negative	1	0

For compare instructions the following table summarizes the setting of the CC. The data is compared as two 8-bit absolute numbers if bit #1, the COM bit, of the Program Status Lower byte is set to indicate "logical" compare (COM=1). If the COM bit indicates "arithmetic" (COM=0), the comparison instructions interpret the data bytes as two 8-bit two's complement binary numbers.

Register to Storage Compare Instruction	Register to Register Compare Instruction	CC1	CC0
Reg X Greater Than Storage	Reg 0 Greater Than Reg X	0	1
Reg X Equal to Storage	Reg 0 Equal to Reg X	0	0
Reg X Less Than Storage	Reg 0 Less Than Reg X	1	0

The CC is never set to 11 by normal processor operations, but it may be explicitly set to 11 through LPSL or PPSL instruction execution.

INTERDIGIT CARRY (DC)

For BCD arithmetic operations it is sometimes essential to know if there was a carry from bit #3 to bit #4 during the execution of an arithmetic instruction.

The IDC reflects the value of the Interdigit Carry from the previous add or subtract instruction. After any add or subtract instruction execution, the IDC contains the carry or borrow out of bit #3.

The IDC is also set upon execution of Rotate instructions when the WC bit in the PSW is set. The IDC will reflect the same information as bit #5 of the operand register after the rotate is executed. See figure II-2.

REGISTER SELECT (RS)

There are two banks of general purpose registers with three registers in each bank. The register select bit is used to specify which set of three general purpose registers will be currently used. Register zero is common and is always available to the program. An individual instruction may address only four registers, but the bank select feature effectively expands the available on-chip registers to seven. When the Register Select Bit is "0", registers 1, 2, & 3 in register bank #0 will be accessable, and when the bit is "1", registers 1, 2, & 3 in register bank #1 will be accessable.

WITH/WITHOUT CARRY (WC)

This bit controls the execution of the add, the subtract and the rotate instructions.

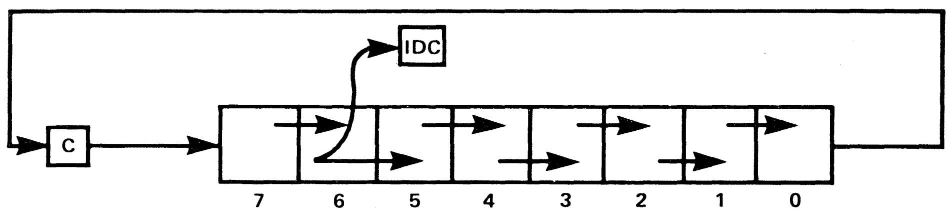
Whenever an add or a subtract instruction executes, the following bits are either set or cleared: Carry/Borrow (C), Overflow (OVF), and Interdigit Carry (IDC). These bits are set or reset without regard to the value of the WC bit. However, when WC=1, the final value of the carry bit affects the result of an add or a subtract instruction, i.e., the carry bit is either added (add instruction) or subtracted (subtract instruction) from the ALU.

Whenever a rotate instruction executes with WC=0, only the eight bits of the rotated register are affected. However, when WC=1, the following bits are also affected: Carry/Borrow (C), Overflow (OVF) and Interdigit Carry (IDC). The carry/borrow bit is combined with the 8-bit register to make a nine-bit rotate (see Figure II-2). The overflow bit is set whenever the sign bit (bit 7) of the rotated register changes its value, i.e., from a zero (0) to a one (1) or from a one (1) to a zero (0). The interdigit carry bit is set to the new value of bit 5 of the rotated register.

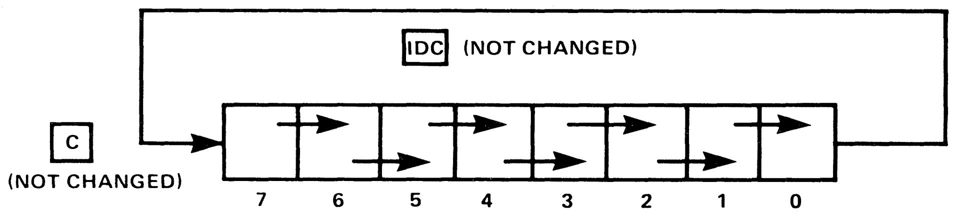
OVERFLOW (OVF)

The overflow bit is set during add or subtract instruction executions whenever the two initial operands have the same sign but the result has a different sign. Operands with different signs cannot cause overflow. Example: A binary +124 (01111100) added to a binary +64 (01000000) produces a result of (10111100) which is interpreted in two's complement form as a −68. The true answer would be 188, but that answer cannot be contained in the set of 8-bit, two's complement numbers used by the processor, so the OVF bit is set.

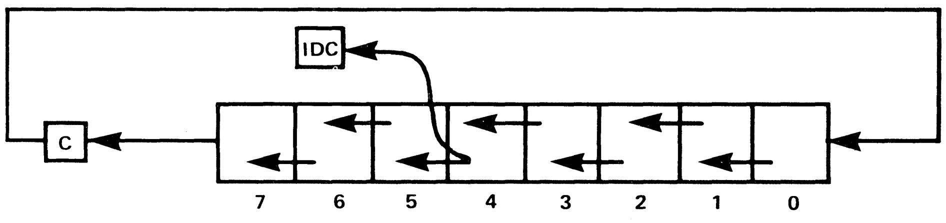
Rotate instructions also cause OVF to be set whenever the sign of the rotated byte changes.



ROTATE REGISTER RIGHT WITH CARRY



ROTATE REGISTER RIGHT WITHOUT CARRY



ROTATE REGISTER LEFT WITH CARRY

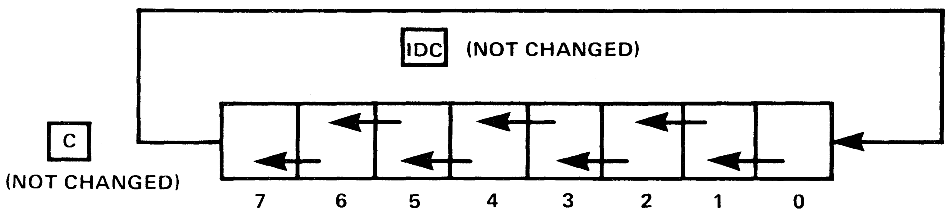


Figure II-2 ROTATE REGISTER LEFT WITHOUT CARRY

COMPARE (COM)

The compare control bit determines the type of comparison that is executed with the Compare instructions. Either logical or arithmetic comparisons may be made. The arithmetic compare assumes that the comparison is between 8-bit, two's complement numbers. The logical compare assumes that the comparison is between 8-bit positive binary numbers. When COM is set to 1, the comparisons will be logical, and when COM is set to 0, the comparisons will be arithmetic. See Condition Code (CC).

CARRY (C)

The Carry bit is set by the execution of any add or subtract instruction that results in a carry or borrow out of the high order bit of the ALU. The carry bit is set to 1 by an add instruction that generates a carry, and a subtract instruction that does *not* generate a borrow. Inversely, an add that does not generate a carry causes the C bit to be cleared, and a subtract instruction that generates a borrow also clears the carry bit.

Even though a borrow is indicated by a zero in the Carry bit, the processor will correctly interpret the zero during subtract with borrow operations as in the following table.

Low Order bit Minuend	Low Order bit Subtrahend	Carry bit Borrow bit	Low Order Bit Result
0	0	0	1
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	0

The carry bit may also be set or cleared by rotate instructions as described earlier under "With/Without Carry".

To perform an Add with Carry or a Subtract with Borrow, the WC bit must be set.

MEMORY ORGANIZATION

The 2650 has a maximum memory addressing capability of 0_{10} —$32{,}767_{10}$ locations. As may be seen in the INSTRUCTIONS section of this manual, most direct addressing instructions have thirteen bits allocated for the direct address. Since thirteen bits can only address locations 0_{10} —$8{,}191_{10}$, a paging system was implemented to accomodate the entire address range.

The memory may be thought of as being divided into four pages of 8,192 bytes each. The addresses in each page range as in the following chart:

	START ADDRESS	END ADDRESS	
page 0	000000000000000	001111111111111	0_{10}—8191_{10}
page 1	010000000000000	011111111111111	8192_{10}—$16{,}383_{10}$
page 2	100000000000000	101111111111111	$16{,}384_{10}$—$24{,}575_{10}$
page 3	110000000000000	111111111111111	$24{,}576_{10}$—$32{,}767_{10}$

The low order 13-bits in every page range through the same set of numbers. These 13-bits are the same 13-bits addressed by non-branch instructions and are also the same 13-bits which are brought out of the 2650 on the address lines ADR0 — ADR12.

The high order two bits of the 15-bit address are known as the page bits. The page bits when examined by themselves also represent, in binary, the number of the memory page. Thus, the address $\underline{01}0000001101101$ is known as address location 109_{10} in page 1. The page bits, corresponding to ADR13 and ADR14 are brought out of the 2650 on pins 19 & 18. These bits may be used for memory access when more than 8,192 bytes of memory are connected.

There are no instructions to explicitly set the page bits. They may be set through execution of direct or indirect, branch or branch-to-subroutine instructions. It may be seen that these instructions (see INSTRUCTION Section) have 15-bits allocated for address and when such an instruction is executed, the two high order address bits are set into the page bit latches in the 2650 processor and will appear on ADR13 and ADR14 during memory accesses until they are specifically changed.

For memory access from non-branch instructions, the 13-bit direct address will address the corresponding location within the current page only. However, the non-branch memory access instruction may access any byte in any page through indirect addressing which provides the full 15-bit address. In the case of non-branch instructions, the page bits are only temporarily changed to correspond to the high order two bits of the 15-bit indirect address used to fetch the argument byte. Immediately after the memory access, ADR13 & ADR14 will revert to their previous value.

The consequences of this page address system may be summarized by the following statements.

1. The RESET signal clears both page latches, i.e., ADR13 & ADR14 are cleared to zero.
2. All non-branch, direct memory access instructions address memory within the current page.
3. All non-branch, memory access instructions may access any byte of addressable memory through use of indirect addressing which temporarily changes the page bits for the argument access, but which revert back to their previous state immediately following instruction execution.
4. All direct and indirect addressing branch instructions set the page bits to correspond to the high order two bits of the 15 bit address.
5. Programs may *not* flow across page boundaries, they must branch to set the page bits.
6. Interrupts always drive the processor to page zero.

CHAPTER III

INTERFACE

SIGNALS

RESET

The RESET signal is used to cause the 2650 to begin processing from a known state. RESET will normally be used to initialize the processor after power-up or to restart a program. RESET clears the Interrupt Inhibit control bit, clears the internal interrupt-waiting signal, and initializes the IAR to zero. RESET is normally low during program execution, and must be driven high to activate the RESET function. The leading and trailing edges may be asynchronous with respect to the clock. The RESET signal must be at least three clock periods long. If RESET alone is used to initiate processing, the first instruction will be fetched from memory location page zero byte zero after the RESET signal is removed. Any instruction may be programmed for this location including a Branch to some program located elsewhere.

Processing can also be initiated by combining an interrupt with a reset. In this case, the first instruction to be executed will be at the interrupt address.

CLOCK

The clock signal is a positive-going pulse train that determines the instruction execution rate. Three clock periods comprise a processor cycle. Direct instructions are 2, 3, or 4 processor cycles long, depending on the specific type of instruction. Indirect addressing adds two processor cycles to the direct instruction times.

$\overline{\text{PAUSE}}$

The $\overline{\text{PAUSE}}$ input provides a means for temporarily stopping the execution of a program. When $\overline{\text{PAUSE}}$ is driven low, the 2650 finishes the instruction in progress and then enters the WAIT state. When $\overline{\text{PAUSE}}$ goes high, program execution continues with the next instruction. If $\overline{\text{PAUSE}}$ is turned on then off again before the last cycle of the current instruction begins, program execution continues without pause. If both $\overline{\text{PAUSE}}$ and $\overline{\text{INTREQ}}$ occur prior to the last cycle of the current instruction, the interrupt will be recognized, and an INTACK will be generated immediately following release of the $\overline{\text{PAUSE}}$. The next instruction to be executed will be a ZBSR to service the interrupt.

If an $\overline{\text{INTREQ}}$ occurs while the 2650 is in a WAIT state due to a $\overline{\text{PAUSE}}$, the interrupt will be acknowledged and serviced after the execution of the next normal instruction following release of the $\overline{\text{PAUSE}}$.

$\overline{\text{INTREQ}}$

The Interrupt Request input (normally high) is a means for external devices to change the flow of program execution. When the processor recognizes an $\overline{\text{INTREQ}}$, i.e., $\overline{\text{INTREQ}}$ is driven low, it finishes the instruction in progress, inserts a ZBSR instruction into the IR, turns on the Interrupt Inhibit bit in the PSU, and then responds with INTACK and OPREQ signals. Upon receipt of INTACK, the interrupting device may raise the $\overline{\text{INTREQ}}$ line and present a data byte to the processor on the DBUS. The required byte takes the same form as the second byte of a ZBSR instruction. Thus, the interrupt initiated Branch-to-Subroutine instruction may have a relative target address anywhere within the first or last 64 bytes of memory page 0. If indirect addressing is specified, a branch to any location in addressable memory is possible.

For devices that do not need the flexibility of the multiple target addresses, a byte of eight zeroes may be presented and will cause a direct subroutine branch to memory location zero in page zero. The relative address presented by the interrupting device is handled with a normal I/O read sequence using the usual interface control signals. The addition of the INTACK signal distinguishes the interrupt address operation from other operations that may take place as part of the execution of the interrupted instruction. At the same time that it acknowledges the $\overline{\text{INTREQ}}$, the processor automatically sets the bit that inhibits recognition of further interrupts. The Interrupt Inhibit bit may be cleared anytime during the interrupt service routine, or a Return-and-Enable instruction may be used to enable interrupts upon leaving the routine. If an $\overline{\text{INTREQ}}$ is waiting when the Interrupt Inhibit bit is cleared, it will be recognized and processed immediately without the execution of an intervening instruction.

$\overline{\text{OPACK}}$

The Operation Acknowledge signal is a reply from external memory or I/O devices as a response to the Operation Request signal from the processor. OPREQ is used to initiate an external operation. The affected external device indicates to the processor that the operation is complete by turning on the $\overline{\text{OPACK}}$ signal. This procedure allows asynchronous functioning of external devices.

If a Memory operation is initiated by the processor, the memory system will provide an $\overline{\text{OPACK}}$ when the requested memory data is valid on the Data Bus. If an I/O operation is initiated by the processor, the addressed I/0 device may respond with an $\overline{\text{OPACK}}$ as soon as the write data is accepted from the Data Bus, or after the read operation is completed. However, in order to avoid slowing down the processor when using memories or I/0 devices that are just fast enough to keep the processor operating at full speed the $\overline{\text{OPACK}}$ signal must be returned before the external operation is completed. Any $\overline{\text{OPACK}}$ that is returned within 600 nsec. following an OPREQ will not delay the processor. Data from a read operation can return up to 1000 nsec. after an OPREQ is sent and still be accepted by the processor without causing delays. If all devices will always respond within these time limits, the $\overline{\text{OPACK}}$ line may be permanently connected in the ON (low) state. Whenever an $\overline{\text{OPACK}}$ is not available within that time, the processor will delay instruction execution until the first clock following receipt of the $\overline{\text{OPACK}}$. All output line conditions remain unchanged during the delay and the processor does not enter the WAIT state. $\overline{\text{OPACK}}$ is true in the low state and false in the high state.

SENSE

The SENSE line provides an input line to the 2650 that is independent of the normal I/0 Bus structures. The SENSE signal is connected directly to one of the bits in the Program Status Word. It may be stored or tested by an executing program. When a store (SPSU) or test (TPSU) instruction is executed, the SENSE line is sampled during the last cycle of the instruction.

Through proper programming techniques the SENSE signal may be used to implement a direct serial data input channel, or it may be used to present any bit of information that the designer chooses.

The SENSE input and FLAG output facilities provide the simplest method of communicating data in or out of the 2650 Processor as neither address decoding nor synchronization with other processor signals is necessary.

$\overline{\text{ADREN}}$

The Address Enable signal allows external control of the tri-state address outputs (ADR0-ADR12). When $\overline{\text{ADREN}}$ is driven high, the address lines are switched to their third state and show a high output impedance. This feature allows wired-OR connections with other signals. The ADR13 and ADR14 lines which are multiplexed with other signals are not affected by this signal.

When a system is not designed to utilize the feature, the $\overline{\text{ADREN}}$ input may be connected permanently to a low signal source.

$\overline{\text{DBUSEN}}$

The Data Bus Enable signal allows external control of the tri-state Data Bus output drivers. When $\overline{\text{DBUSEN}}$ is driven high, the Data Bus will exhibit a high output impedance. This allows wired-OR connection with other signals.

When a system is not designed to utilize this feature, the $\overline{\text{DBUSEN}}$ input may be permanently connected to a low signal source.

DBUS

The Data Bus signals form an 8-bit bi-directional data path in and out of the processor. Memory and I/0 operations use the Data Bus to transfer the write or read data to or from memory.

The direction of the data flow on the Data Bus is indicated by the state of the $\overline{\text{R}}$/W line. For Write operations, the output buffers in the processor output data to the bus for use by memory or by external devices. For Read operations, the buffers are disabled and the data condition of the bus is sensed by the processor. The output buffers may also be disabled by the $\overline{\text{DBUSEN}}$ signal.

The signals on the data bus are true signals, i.e., a one is a high level and a zero is low.

ADR

The Address signals form a 15 bit path out of the processor, and are used primarily to supply memory addresses during memory operations. The addresses remain valid as long as OPREQ is on so that no external address register is required. For extended I/O operations, the low order eight bits of the ADR lines are used to output the immediate byte of the instruction which typically is interpreted as a device address.

The 13 low order lines of the address are used only for address information. The two high order address lines are multiplexed with I/O control information. During memory operations, the lines serve as memory addresses. During I/O operations they serve as the D/$\overline{\text{C}}$ and E/$\overline{\text{NE}}$ control lines. Demultiplexing is accomplished through use of the Memory/$\overline{\text{IO}}$ Control line.

The line ADR0 carries the low order address bit, and ADR12 carries the high order address bit. The output drivers may be disabled by the $\overline{\text{ADREN}}$ signal.

The signals on the address bus are true, i.e., a one is a high level and a zero is low.

OPREQ

The Operation Request output is the coordinating signal for all external operations. The M/$\overline{\text{IO}}$, $\overline{\text{R}}$/W, E/$\overline{\text{NE}}$, D/$\overline{\text{C}}$ and INTACK lines are operation control signals that describe the nature of the external operation when the OPREQ line is true. The DBUS and ADR bus also should not be considered valid except when OPREQ is in the high, or on state.

No output signals from the processor will change as long as OPREQ is on, with the exception of WRP. OPREQ will stay on until the external operation is complete, as indicated by the $\overline{\text{OPACK}}$ input. The processor delays all internal activity following an OPREQ until the $\overline{\text{OPACK}}$ signal is received.

INTACK

The Interrupt Acknowledge signal is used by the processor to respond to an external interrupt. When an $\overline{\text{INTREQ}}$ is received, the current instruction is completed before the interrupt is serviced. When the processor is ready to accept the interrupt it sets the INTACK to the high, or on, state along with OPREQ. The interrupting device then presents a relative address byte to the DBUS and responds with an $\overline{\text{OPACK}}$ signal. $\overline{\text{INTREQ}}$ may be turned off anytime following INTACK. INTACK will fall after the processor receives the $\overline{\text{OPACK}}$ signal.

M/$\overline{\text{IO}}$

The Memory/$\overline{\text{IO}}$ output is one of the operation control signals that defines external operations. M/$\overline{\text{IO}}$ indicates whether an operation is memory or I/O and should be used to gate Read or Write signals between memory or I/O devices.

The state of M/$\overline{\text{IO}}$ will not change while OPREQ is high.

The high state corresponds to Memory operation, and the low state corresponds to an I/O operation.

$\overline{\text{R}}$/W

The Read/Write output is one of the operation control signals that defines external operations. $\overline{\text{R}}$/W indicates whether an operation is *Read* or *Write.* It controls the nature of the external operation and indicates in which direction the DBUS is pointing. $\overline{\text{R}}$/W should not be considered valid until OPREQ is on and the state of the $\overline{\text{R}}$/W line does not change as long as OPREQ is on.

The high state corresponds to the Write operation, and the low state corresponds to the Read operation.

D/$\overline{\text{C}}$

The Data/Control Output is an I/O signal which is used to discriminate between the execution of the two types of one byte I/O instructions. There are four one byte I/O instructions; WRTC, WRTD, REDC, REDD. When Read Control or Write Control is executed, the D/$\overline{\text{C}}$ line takes on the low state which indicates Control ($\overline{\text{C}}$). When Read Data or Write Data is executed, the D/$\overline{\text{C}}$ line takes on the high state, indicating Data (D).

D/$\overline{\text{C}}$ should not be considered valid until (a) OPREQ is on and (b) M/$\overline{\text{IO}}$ indicates an I/O operation and (c) E/$\overline{\text{NE}}$ indicates a non-extended (one byte) operation. D/$\overline{\text{C}}$ is multiplexed with a high order address line. When the M/$\overline{\text{IO}}$ line is in the I/O state, the ADR14-D/$\overline{\text{C}}$ line should be interpreted as "D/$\overline{\text{C}}$". (When the M/$\overline{\text{IO}}$ line is in the M state, the ADR14-D/$\overline{\text{C}}$ line should be interpreted as memory address line #14.)

E/$\overline{\text{NE}}$

The Extended/Non-Extended output is the operation control signal that is used to discriminate between two byte and one byte I/O operations. Thus, E/$\overline{\text{NE}}$ indicates the presence or absence of valid information on the eight low order address lines during I/O operations.

E/$\overline{\text{NE}}$ should not be considered valid until (a) OPREQ is on and (b) M/$\overline{\text{IO}}$ indicates an I/O operation. E/$\overline{\text{NE}}$ is multiplexed with a high order address line. When the M/$\overline{\text{IO}}$ line is in the I/O state, the ADR13-E/$\overline{\text{NE}}$ line should be interpreted as "E/$\overline{\text{NE}}$". (When the M/$\overline{\text{IO}}$ line is in the M state, the ADR13-E/$\overline{\text{NE}}$ line should be interpreted as memory address bit #13.)

There are six I/O instructions; REDE, WRTE, REDC, REDD, WRTC, WRTD. When either of the two byte I/O instructions is executed (REDE, WRTE), the E/$\overline{\text{NE}}$ line takes on the high state or "Extended" indication. When any of the one byte I/O instructions is executed, the line takes on the low state or "non-extended" indication.

RUN/$\overline{\text{WAIT}}$

The RUN/$\overline{\text{WAIT}}$ output signal indicates the Run/Wait Status of the processor. The WAIT state may be entered by executing a HALT instruction or by turning on the $\overline{\text{PAUSE}}$ input. At any other time the processor will be in a RUN state.

When the processor is executing instructions, the line is in the high or RUN state; when in the WAIT state, the line is held low.

The HALT initiated WAIT condition can be changed to RUN by a RESET or an interrupt. The $\overline{\text{PAUSE}}$ initiated WAIT condition can be changed to RUN by removing the $\overline{\text{PAUSE}}$ input.

If a RESET occurs during a $\overline{\text{PAUSE}}$ initiated WAIT state and the $\overline{\text{PAUSE}}$ remains low; the processor will be reset, fetch one instruction from page zero byte zero and return to the WAIT state. When the $\overline{\text{PAUSE}}$ is eventually removed, the previously fetched instruction will be executed.

FLAG

The FLAG output indicates the state of the Flag bit in the PSW. Any change in the Flag bit is reflected by a change in the FLAG output. A one bit in the Flag will give a high level on the FLAG output pin. The LPSU, PPSU, and CPSU instructions can change the state of the Flag bit.The FLAG output is always a valid indication of the state of the Flag bit without regard for the status of the processor or control signals. Changes in the Flag bit are synchronized with the last cycle of the changing instruction.

WRP

The Write Pulse output is a timing signal from the processor that provides a positive-going pulse in the middle of each requested write operation (memory or I/O) and a high level during read operations. The WRP is designed to be used with Signetics 2606 R/W memory circuits to provide a timed Chip Enable signal. For use with memory, it may be gated with the M/$\overline{\text{IO}}$ signal to generate a Memory Write Pulse.

Because the WRP pulse occurs during any write operation, it may also be used with I/O write operations where convenient.

SIGNAL TIMING

The Clock input to the 2650 provides the basic timing information that the processor uses for all its internal and external operations. The clock rate determines the instruction execution rate, except to the extent that external memories and devices slow down the processor. Each internal processor cycle is composed of three clock periods as shown in Figure III-3, GENERAL TIMING.

OPREQ is the master control signal that coordinates all operations external to the processor. Many of the other signal interactions are related to OPREQ. The timing diagram assumes that the clock periods are constant and that $\overline{\text{OPACK}}$ is returned in time to avoid delaying instruction execution. In that case, OPREQ will be high for 1.5 clock periods (1/2 of t_{pc}) and then will be low for another 1.5 clock periods.

The interface control signals have been designed to implement asynchronous interfaces for both memory and input/output devices. The control signals are relatively simple and provide the following advantages: no external synchronizing is necessary, external devices may run at any data rate up to the processor's maximum I/O data rate, and because data signals are furnished with guard signals the external devices are often relieved of the necessity of latching information such as memory address.

MEMORY READ TIMING

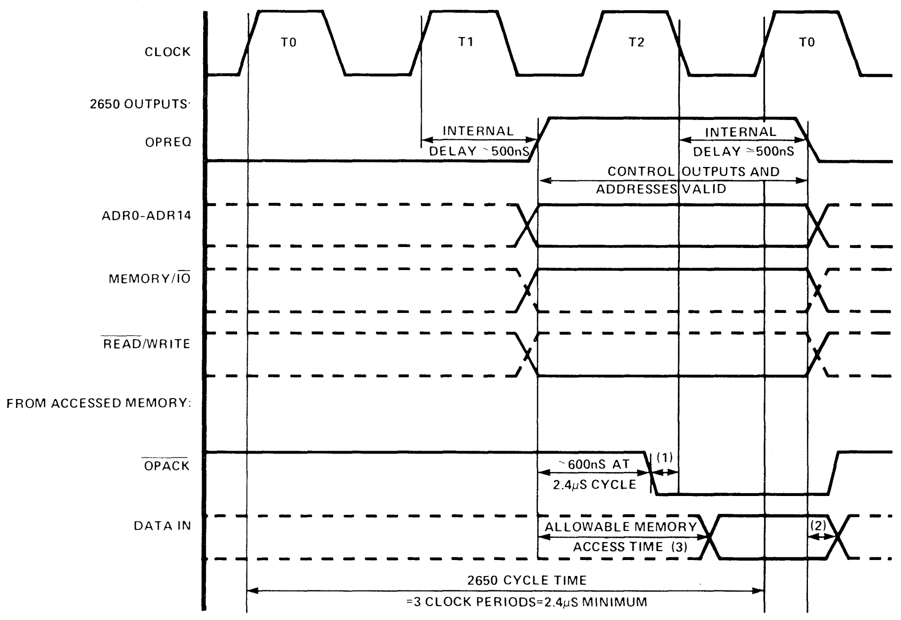
The following signals are involved in the processor's memory read sequence, as shown in Figure III-1.

OPREQ	= Operation Request
DBUS0-DBUS7*	= Data Bus
ADR0-ADR12	= Address Bus
ADR13	= Address bit 13
ADR14	= Address bit 14
$\text{M}/\overline{\text{IO}}$	= Memory/Input-Output
$\overline{\text{R}}/\text{W}$	= Read/Write
$\overline{\text{OPACK}}$*	= Operation Acknowledge

The signals marked with an asterisk are sent from the memory device to the processor. The other signals are developed by the processor.

OPREQ is a guard signal which must be valid (high) for the other signals to have meaning. When reading main memory the 2650 simultaneously switches OPREQ to a high state, $\text{M}/\overline{\text{IO}}$ to M (memory), $\overline{\text{R}}/\text{W}$ to $\overline{\text{R}}$ (Read), and places the memory address on lines ADR0-ADR14. Remember that ADR13 & ADR14 are multiplexed with other signals and must be logically ANDed with OPREQ and M to be interpreted. Of course, ADR13 & ADR14 may be ignored if only page zero (8,192 bytes) is used.

Once the memory logic has determined the simultaneous existance of the signals mentioned above, it places the true data corresponding to the given address location on the data bus (DBUS0 to DBUS7), and returns an $\overline{\text{OPACK}}$ signal to the processor. The processor, recognizing the $\overline{\text{OPACK}}$, strobes the data into the receiving register and lowers the OPREQ. This completes the memory read sequence.



NOTES: (1) OPACK must go low at least 100 nS before the trailing edge of T2 in order not to slow down the 2650.
(2) DATA IN signals must be valid for 50nS after the trailing edge of OPREQ.
(3) Allowable memory access time is 1μs with 2.4μs cycle time.

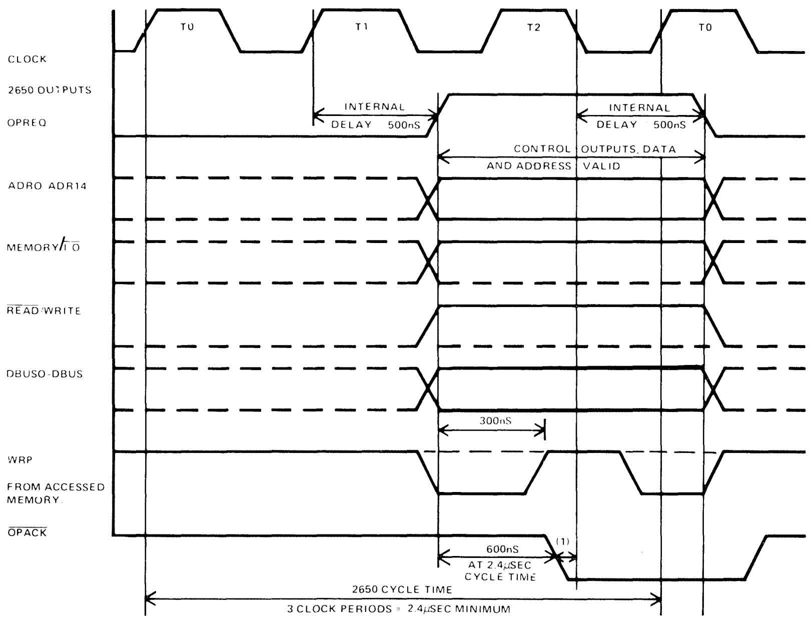
Figure III-1 MEMORY READ SEQUENCE

If the $\overline{\text{OPACK}}$ signal is delayed by the memory device, the processor waits until it is received. OPREQ is lowered only after the receipt of $\overline{\text{OPACK}}$. The memory device should raise $\overline{\text{OPACK}}$ after OPREQ falls.

MEMORY WRITE TIMING

The signals involved with the processor's memory write sequence are similar to those used in the memory read sequence with the following exceptions: 1) the $\overline{\text{R}}$/W signal is in the W state and, 2) the WRP signal provides a positive going pulse during the write sequence which may be used as a chip enable, write pulse, etc.

Figure III-2 demonstrates the signals that occur during a memory write.



NOTES: (1) $\overline{\text{OPACK}}$ must go low at least 100nS before the trailing edge of T2 in order not to slow down the 2650.

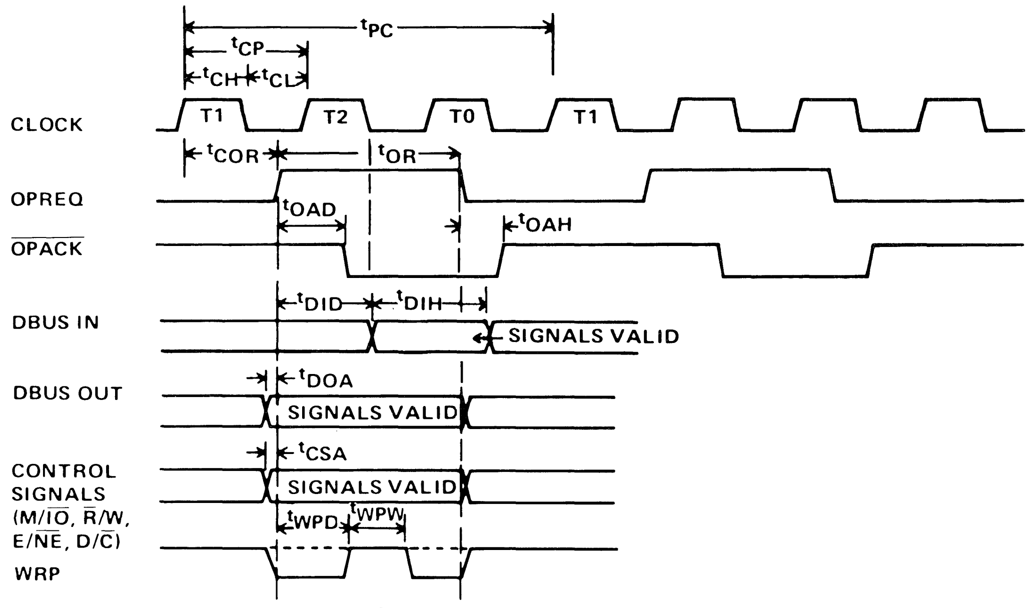
Figure III-2 MEMORY WRITE SEQUENCE

INPUT/OUTPUT TIMING

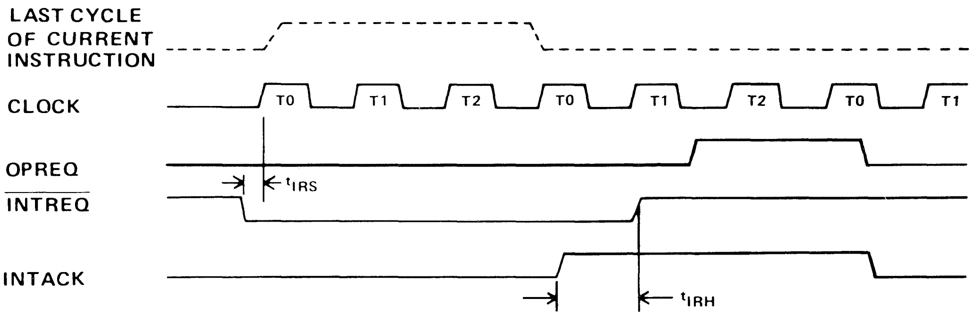
The signal exchanges for I/O with external devices is very similar to the signaling for memory read/write. See the Features Section, INPUT/OUT PUT FACILITIES.

CRITICAL TIMES

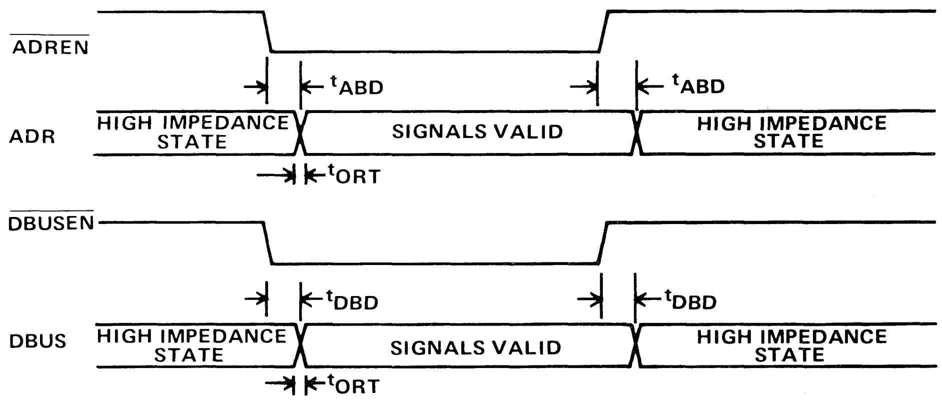
The following timing diagram describes the timing relationship between the various interface signals. The critical times are labeled and defined in the table of AC characteristics.



GENERAL TIMING



INTERRUPT TIMING



TRI-STATE BUS TIMING

Figure III-3 2650 TIMING DIAGRAMS

PRELIMINARY AC CHARACTERISTICS

T_A=0°C to 70°C V_{CC}=5V±5% unless otherwise specified, see notes 1,2,3 & 4.

SYMBOL	PARAMETER	LIMITS		UNITS
		MIN	MAX	
t_{CH}	Clock High Phase	400	10,000	nsec
t_{CL}	Clock Low Phase	400	∞	nsec
t_{CP}	Clock Period	800	∞	nsec
t_{PC}[6]	Processor Cycle Time	2,400	∞	nsec
t_{OR}	OPREQ Pulse Width	$2t_{CH} + t_{CL} - 100$	∞	nsec
t_{COR}	Clock to OPREQ Time	100	700	nsec
t_{OAD}[7]	$\overline{\text{OPACK}}$ Delay Time	0	∞	nsec
t_{OAH}	$\overline{\text{OPACK}}$ Hold Time	0	∞	nsec
t_{CSA}	Control Signal Available	50		nsec
t_{DOA}	Data Out Available	50		nsec
t_{DID}[8]	Data in Delay	0	1000(8)	nsec
t_{DIH}[9]	Data in Hold	150		nsec
t_{WPD}	Write Pulse Delay	$t_{CL}-100$	$t_{CL}-50$	nsec
t_{WPW}	Write Pulse Width	t_{CL}	t_{CL}	nsec
t_{ABD}	Address Bus Delay		80	nsec
t_{DBD}	Data Bus Delay		120	nsec
t_{IRS}[10]	$\overline{\text{INTREQ}}$ Set up Time	0		nsec
t_{IRH}[10]	$\overline{\text{INTREQ}}$ Hold Time	0		nsec
t_{ORT}[5]	Output Buffer Rise Time		150	nsec

NOTES ON AC CHARACTERISTICS
1. See preceding timing diagrams for definition of timing terms.
2. Input levels swing between 0.65 volt and 2.2 volts.
3. Input signal transition times are 20ns.
4. Timing reference level is 1.5 volts.
5. Load is -100μA at 20pF.
6. A Processor Cycle time consists of three clock periods.
7. In order to avoid slowing down the processor, $\overline{\text{OPACK}}$ must be lowered 100ns before the trailing edge of T2 clock, if $\overline{\text{OPACK}}$ is delayed past this point, the processor will wait in the T2 state and sample $\overline{\text{OPACK}}$ on each subsequent negative clock edge until $\overline{\text{OPACK}}$ is lowered.
8. In order to avoid slowing the processor down, input data must be returned to the processor in 1μs or less time from the OPREQ edge, at a cycle time of 2.4μs.
9. Input data must be held until 50ns after OPREQ falls.
10. In order to interrupt the current instruction, $\overline{\text{INTREQ}}$ must fall prior to the first clock of the last cycle of the current instruction. $\overline{\text{INTREQ}}$ must remain low until INTACK goes high.

ELECTRICAL CHARACTERISTICS

MAXIMUM GUARANTEED RATINGS[1]

Operating Ambient Temperature	0^oC to $+70^oC$
Storage Temperature	-65^oC to $+150^oC$
All Input, Output, and Supply Voltages with respect to ground pin[3]	-0.5V to +6V
Package Power Dissipation[2]=IWPkg.	1.6W

PRELIMINARY 2650 DC ELECTRICAL CHARACTERISTICS

			LIMITS		
SYMBOL	PARAMETER	TEST CONDITIONS	MIN	MAX	UNIT
I_{LI}	Input Load Current	V_{IN} = 0 to 5.25V		10	μA
I_{LOH}	Output Leakage Current	ADREN, DBUSEN = 2.2V, V_{OUT} = 4V		10	μA
I_{LOL}	Output Leakage Current	ADREN, DBUSEN = 2.2V, V_{OUT} = 0.45V		10	μA
I_{CC}	Power Supply Current	V_{CC} = 5.25V, T_A = $0^\circ C$		100	mA
V_{IL}	Input Low		-0.6	0.8	V
V_{IH}	Input High		2.2	V_{CC}	V
V_{OL}	Output Low	I_{OL} = 1.6 mA	0.0	0.45	V
V_{OH}	Output High	I_{OH} = -100 μA	2.4	V_{CC}-0.5	V
C_{IN}	Input Capacitance	V_{IN} = 0V		10	pF
C_{OUT}	Output Capacitance	V_{OUT} = 0V		10	pF

Conditions: T_A = $0^\circ C$ to $70^\circ C$, V_{CC} = 5V ±5%

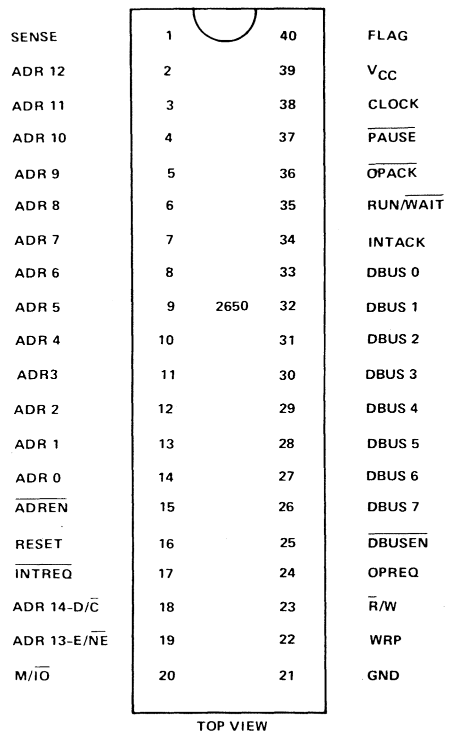
NOTES:
1. Stresses above those listed under "Maximum Guaranteed Ratings" may cause permanent damage to the device. This is a stress rating only and functional operation of the device at these or at any other condition above those indicated in the operation sections of this specification is not implied.
2. For operating at elevated temperatures the device must be derated based on a $+150^oC$ maximum junction temperature and a thermal resistance of $50^oC/W$ junction to ambient (40 pin IW package).
3. This product includes circuitry specifically designed for the protection of its internal devices from the damaging effects of excessive static charge. Nonetheless, it is suggested that conventional precautions be taken to avoid applying any voltages larger than the rated maxima.
4. Parameter valid over operating temperature range unless otherwise specified.
5. All voltage measurements are referenced to ground.
6. Manufacturer reserves the right to make design and process changes and improvements.
7. Typical values are at $+25^oC$, nominal supply voltages, and nominal processing parameters.

SIGNETICS 2650 PROCESSOR

INTERFACE SIGNALS

TYPE	PINS	ABBREVIATION	FUNCTION	SIGNAL SENSE
INPUT	1	GND	Ground	GND=0
INPUT	1	V_{CC}	+5 Volts ±5%	V_{CC}=1
INPUT	1	RESET	Chip Reset	RESET=1 (pulse), causes reset
INPUT	1	CLOCK	Chip Clock	
INPUT	1	$\overline{PAUSE}$	Temp. Halt execution	$\overline{PAUSE}$=0, temporarily halts execution
INPUT	1	$\overline{INTREQ}$	Interrupt Request	$\overline{INTREQ}$=0, requests interrupt
INPUT	1	$\overline{OPACK}$	Operation Acknowledge	$\overline{OPACK}$=0, acknowledges operation
INPUT	1	SENSE	Sense	SENSE=0 (low) or SENSE=1 (high)
INPUT	1	$\overline{ADREN}$	Address Enable	$\overline{ADREN}$=1 drives into third state
INPUT	1	$\overline{DBUSEN}$	Data Bus Enable	$\overline{DBUSEN}$=1 drives into third state
IN/OUT	8	DBUS0-DBUS7	Data Bus	DBUSn=0 (low), DBUSn=1 (high)
OUTPUT	13	ADR0-ADR12	Address 0 through 12	ADRn=0 (low), ADRn=1 (high)
OUTPUT	1	ADR13 or E/$\overline{NE}$	Address 13 or Extended/Non-Extended	Non-Extended=0, Extended=1
OUTPUT	1	ADR14 or D/$\overline{C}$	Address 14 or Data Control	Control=0, Data 1
OUTPUT	1	OPREQ	Operation Request	OPREQ=1, requests operation
OUTPUT	1	M/$\overline{IO}$	Memory/IO	IO=0, M=1
OUTPUT	1	$\overline{R}$/W	Read/Write	R=0, W=1
OUTPUT	1	FLAG	Flag Output	FLAG=1 (high), FLAG=0 (low)
OUTPUT	1	INTACK	Interrupt Acknowledge	INTACK=1, acknowledges interrupt
OUTPUT	1	RUN/$\overline{WAIT}$	Run/Wait Indicator	RUN=1, WAIT=0
OUTPUT	1	WRP	Write Pulse	WRP=1 (pulse), causes writing

PIN CONFIGURATION



CHAPTER IV

FEATURES

INPUT/OUTPUT FACILITIES

The 2650 processor provides several mechanisms for performing input/output functions. They are flag and sense, non-extended I/O instructions, extended I/O instructions, and memory I/O. These four facilities are described below.

FLAG & SENSE I/O

The 2650 has the ability to directly output one bit of data without additional address decoding or synchronizing signals.

The bit labeled "Flag" in the Program Status Word is connected through a TTL compatible driver to the chip output at pin #40. The Flag output always reflects the value in the Flag bit.

When a program changes the Flag bit through execution of an LPSU, PPSU, or CPSU, the bit will be set or cleared during the last cycle of the instruction that changes it.

The Flag bit may be used conveniently for many different purposes. The following is a list of some possible uses:

1. A serial output channel
2. An additional address bit to increase addressing range.
3. A switch or toggle output to control external logic.
4. The origin of a pulse for polling chains of devices.

The Sense bit performs the complementary function of the Flag and is a single bit direct input to the 2650. The Sense input, pin #1 is connected to a TTL compatible receiver and is then routed directly to a bit position in the Program Status Word. The bit in the PSW always represents the value of the external signal. It may be sampled anytime through use of the TPSU or SPSU instructions.

This simple input to the processor may be used in many ways. The following is a list of some possible uses:

1. A serial input channel
2. A sense switch input
3. A break signal to a processing program
4. An input for yes/no signaling from external devices.

NON-EXTENDED I/O

There are four one byte I/O instructions; REDC, REDD, WRTC, and WRTD. They are all referred to as non-extended because they can communicate only one byte of data, either into or out of the 2650.

REDC and REDD causes the input transfer of one byte of data. They are identical except for the fact that the D/$\overline{\text{C}}$ Signal is in the D state for REDD and in the $\overline{\text{C}}$ state for REDC. Similarly, the instructions WRTC and WRTD cause an output transfer of one byte of data. The D/$\overline{\text{C}}$ line discriminates between the two pairs of input/output instructions. The D/$\overline{\text{C}}$ line can be used as a 1-bit device address in simple systems.

The read and write timing sequences for the one byte I/O instructions are the same as the memory read and write sequences with the following exceptions: the M/$\overline{\text{IO}}$ signal is switched to $\overline{\text{IO}}$, the D/$\overline{\text{C}}$ line becomes valid, E/$\overline{\text{NE}}$ is switched to $\overline{\text{NE}}$ (non-extended), and the Address bus contains no valid information.

The $\overline{NE}$ signal informs the devices outside the 2650 that a one byte I/O instruction is being executed. The D/$\overline{C}$ line indicates which pair of the one byte I/O instructions are being executed; D implies either WRTD or REDD, and $\overline{C}$ implies either WRTC or REDC. Finally, to determine whether it is a read or a write, examine the $\overline{R}$/W signal level.

Table IV-1 illustrates the sense of the interface signals. The "Signal Timing" section should be referenced for the exact timing relationships. It should be remembered that the control signals are not to be considered valid except when the OPREQ signal is valid.

Table IV-1 I/O INTERFACE SIGNALS

	OPREQ	M/$\overline{IO}$	$\overline{R}$/W	ADR13-E/$\overline{NE}$	ADR14-D/$\overline{C}$
MEMORY READ	T	M	$\overline{R}$	ADR13	ADR14
MEMORY WRITE	T	M	W	ADR13	ADR14
2 BYTE READ	T	$\overline{IO}$	$\overline{R}$	E	Don't Care
2 BYTE WRITE	T	$\overline{IO}$	W	E	Don't Care
1 BYTE CONTROL READ	T	$\overline{IO}$	$\overline{R}$	$\overline{NE}$	$\overline{C}$
1 BYTE CONTROL WRITE	T	$\overline{IO}$	W	$\overline{NE}$	$\overline{C}$
1 BYTE DATA READ	T	$\overline{IO}$	$\overline{R}$	$\overline{NE}$	D
1 BYTE DATA READ	T	$\overline{IO}$	W	$\overline{NE}$	D

EXTENDED I/O

There are two, two byte I/O instructions; REDE and WRTE. They are referred to as extended because they can communicate two bytes of data when they are executed. The REDE causes the second byte of the instruction to be output on the low order address lines, ADR0-ADR7, which is intended to be used as a device address while the byte of data then on the Data Bus will be strobed into the register specified in the instruction. The WRTE also presents the second byte of the instruction on the Address Bus, but a byte of data from the register specified in the instruction is simultaneously output on the Data Bus.

The two byte I/O instructions are similar to the one byte I/O instructions except: the D/$\overline{C}$ line is not considered, and the data from the second byte of the I/O instruction appears on the Address Bus all during the time that OPREQ is valid. The data on the Address Bus is intended to convey a device address, but may be utilized for any purpose.

Table IV-1 illustrates the sense of the interface signals for extended I/O instructions. Refer to "Signal Timing" section for exact timing relationships.

MEMORY I/O

The 2650 user may choose to transfer data into or out of the processor using the memory control signals. The advantage to this technique is that the data can be read or written by the program through ordinary instruction execution and data may be directly operated upon with the arithmetic instructions.

To make use of this technique, the designer has to assign memory addresses to devices and design the device interfaces to generate the same signals as memory.

A disadvantage to this method is that it may be necessary to decode more address lines to determine the device address than with other I/O facilities.

INTERRUPT MECHANISM

The 2650 has been implemented with a conventional, single level, address vectoring interrupt mechanism. There is one interrupt input pin. When an external device generates an interrupt signal ($\overline{\text{INTREQ}}$), the processor is forced to transfer control to any of 128 possible memory locations as determined by an 8-bit vector supplied by the interrupting device.

Of special interest is that the device may return a relative indirect address signal which causes the processor to enter an indirect addressing sequence upon receipt of an interrupt. This enables a device to direct the processor to execute code anywhere within addressable memory.

Upon recognizing the interrupt signal, the processor automatically sets the Interrupt Inhibit bit in the Program Status Word. This inhibits further interrupts from being recognized until the interrupt routine is finished executing and a Return-and-Enable instruction is executed or the inhibit bit is explicitly cleared.

When the inhibit bit in the PSW is set, the processor will not recognize an interrupt input. The Interrupt Inhibit bit may be set under program control (LPSU, PPSU) and is automatically set whenever the processor accepts an interrupt. The inhibit bit may be cleared in three ways:

1. By a RESET operation
2. By execution of an appropriate clear or load PSU instruction; (CPSU, LPSU)
3. By execution of a Return-and-Enable instruction.

The sequence of events for a normal interrupt operation is as follows:

1. An executing program enables interrupts.
2. External device initiates interrupt with the $\overline{\text{INTREQ}}$ line.
3. Processor finishes executing current instruction.
4. Processor sets inhibit bit.
5. Processor inserts the first byte of ZBSR (Zero Branch-to-Subroutine, Relative) instruction into the instruction register instead of what would have been the next sequential instruction.
6. Processor accesses the data bus to fetch the second byte of the ZBSR instruction.
7. Interrupting device responds to the Processor generated INTACK (Interrupt Acknowledge) by supplying the requested second byte.
8. The processor executes the Zero Branch-to-Subroutine instruction, saving the address of the instruction following the interrupted instruction in the RAS, and proceeds to execute the instruction at page 0, byte 0, or the address relative to page 0, byte 0 as given by the interrupting device.
9. When the interrupt routine is complete, a return instruction (RETC, RETE) pulls the address from the RAS and execution of the interrupted program resumes.

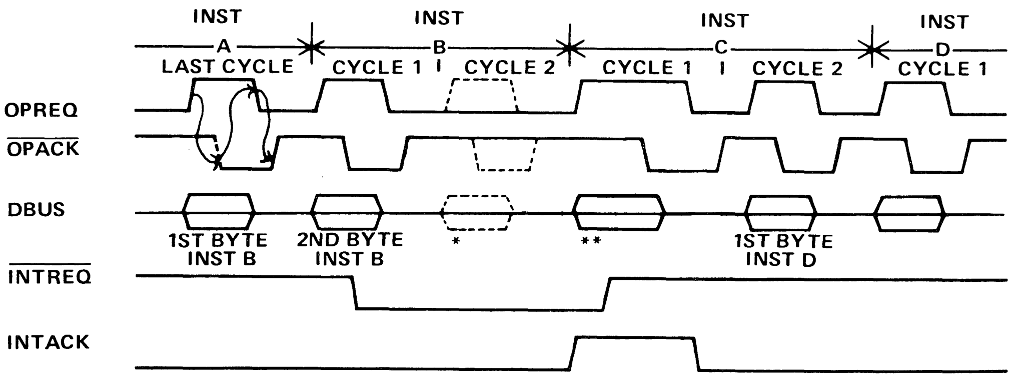
Since the interrupting device specifies the interrupt subroutine address in the standard relative address format, it has considerable flexibility with regard to the interrupt procedure. It can point to any location that is within +63 or −64 bytes of page zero, byte zero of memory. (Negative relative addresses wrap around the memory, modulo $8{,}192_{10}$ bytes.) The interrupting device also may specify whether the subroutine address is direct or indirect by providing a zero or one to DBUS #7 (pin #26). If the external device is not complex enough to exercise these options, it may respond to the INTACK operation with a byte of all zeroes. In such a case, the processor will execute a direct Branch-to-Subroutine to page zero, byte zero of memory.

The timing diagram in Figure IV-2 will help explain how the interrupt system works in the processor. The execution of the instruction labeled "A" has been proceeding before the start of this diagram. The last cycle of instruction "A" is shown. Notice that, as in all external operations, the OPREQ output eventually causes an $\overline{\text{OPACK}}$ input, which in turn allows OPREQ to be turned off. The arrows show this sequence of events. The last cycle of instruction "A" fetches the first byte of instruction "B" from Memory and inserts it into the Instruction Register.

Assume that instruction "B" is a two cycle, two byte instruction with no operand fetch (e.g., ADDI). Since the first byte has already been fetched by instruction "A", the first cycle of instruction "B" is used to fetch the second byte of instructiön "B". Had instruction "B" not been interrupted, it would have fetched the first byte of the next sequential instruction during its second (last) cycle. The dotted lines indicate that operation.

Since instruction "B" is interrupted, however, the last cycle of "B" is used to insert the interrupt instruction (ZBSR) into the instruction register. Notice that the $\overline{\text{INTREQ}}$ input can arrive at any time. Instruction B is interrupted since $\overline{\text{INTREQ}}$ occured prior to the last (2nd) cycle of execution.

Instead of being the next sequential instruction following "B", instruction "C" is the completion of the interrupt. The first cycle of "C" is used to fetch the second byte of the ZBSR instruction from the DBUS as provided by the interrupting device. This fact is indicated by the presence of the INTACK control signal. The $\overline{\text{INTREQ}}$ may then be removed. When the device responds with the requested byte, it uses a standard operation acknowledge procedure ($\overline{\text{OPACK}}$) to so indicate to the processor. During the second cycle of instruction "C" the processor executes the ZBSR instruction, and fetches the first byte of instruction "D" which is located at the subroutine address.



* PROCESSOR INSERTS 1ST BYTE OF ZBSR INSTRUCTION. ADDRESS OF 1ST BYTE OF INSTC IS PUSHED INTO RETURN ADDRESS STACK.

** 2ND BYTE OF ZBSR (INTERRUPT VECTOR)

Figure IV - 2 INTERRUPT TIMING

SUBROUTINE LINKAGE

The on-chip stack, along with the Branch-to-Subroutine and Return instructions provide the facility to transfer control to a subroutine. The subroutine can return control to the program that branched to it via a Return instruction.

The stack is eight levels deep which means that a routine may branch to a subroutine, which may branch to another subroutine, etc., eight times before any Return instructions are executed.

When designing a system that utilizes interrupts, it should be remembered that the processor jams a ZBSR into the IR and then executes it. This will cause an entry to be pushed into the on-chip stack like any other Branch-to-Subroutine instruction and may limit the stack depth available in certain programs.

When branching to a subroutine, the following sequence of events occurs:

1. The address in the IAR is used to fetch the Branch-to-Subroutine instruction and is then incremented in the Address Adder so that it points to the instruction following the subroutine branch.
2. The Stack Pointer is incremented by one so that it points to the next Return Address Stack location.
3. The contents of the IAR are stored in the stack at the location designated by the Stack Pointer.
4. The operand address contained in the Branch-to-Subroutine instruction (the address of the first instruction of the subroutine) is inserted into the IAR.

When returning from a subroutine, this sequence of events occurs:

1. The address in the IAR is used to fetch the return (RETC, RETE) instruction from memory.
2. When the return instruction is recognized by the processor, the contents of the stack entry pointed to by the Stack Pointer is placed into the IAR.
3. The Stack Pointer is decremented by one.
4. Instruction execution continues at the address now in the IAR.

CONDITION CODE USAGE

The two-bit register, called the Condition Code, is incorporated in the Program Status Word. It may be seen in the description of the 2650 instructions, that the Condition Code (CC) is specifically set by every instruction that causes data to be transferred into a general purpose register and it is also set by compare instructions.

The reason for this design feature is that after an instruction executes, the CC contains a modest amount of information about the byte of data which has just been manipulated. For example, a program loads register one with a byte of unknown data and the Condition Code setting indicates that the byte is positive, negative or zero. The negative indication implies that bit #7 is set to one.

Consequently, a data manipulation operation when followed by a conditional branch is often sufficient to determine desired information without resorting to a specific test, thus saving instructions and memory space.

In the following example, the Condition Code is used to test the parity of a byte of data which is stored at symbolic memory location CHAR.

```
EQ      EQU        0       THE EQUAL CONDITION CODE
CHAR    DATA       2       UNKNOWN DATA BYTE
WC      EQU        H'04'   THE WITH CARRY BIT
NEG     EQU        2       CC MASK
        CPSL       WC      CLEAR CARRY BIT
        LODI,R2    -8      SET UP COUNTER
        SUBZ       R0      CLEAR REG 0
        LODR,R1    CHAR    GET THE CHARACTER (cc is set)
LOOP    BCFR,NEG   G01     IF NOT SET, DON'T COUNT (cc is
                             tested)
        ADDI,R0    +1      COUNT THE BIT
G01     RRL,R1             MOVE BITS LEFT (cc is set)
        BIRR,R2    LOOP    LOOP TILL DONE

*       FINISHED,TEST IF REG 0 HAS A ONE IN LOW ORDER
*       IF BIT #0 = 1, ODD PARITY. IF BIT #0 = 0, THEN EVEN.

        TMI,R0     H'01'
        BCTR,EQ    ODD
EVEN    HALT
ODD     HALT
```

START-UP PROCEDURE

The 2650 processor, having no internal start-up procedure must be started in an orderly fashion to assure that the internal control logic begins in a known state.

Assuming power is applied to the chip and the clock input is running, the easiest way to start is to apply a Reset signal for at least three clock periods. When the RESET signal is removed the processor will fetch the instruction at page 0, byte 0 and commence ordinary instruction execution.

To start processing at a specific address, a more complex start-up procedure may be employed. If an Interrupt signal is applied initially along with the Reset, processing will commence at the address provided by the interrupting device. Recall that the address provided may include a bit to specify indirect addressing and therefore the first instruction executed may be anywhere within addressable memory. The Reset and Interrupt signal may be applied simultaneously and when the Reset is removed, the processor will execute the usual interrupt signal sequence as described in INTERRUPT MECHANISM. There is an example of a start-up technique in the System Application Notes.

CHAPTER V

INSTRUCTIONS

ADDRESSING MODES

An addressing mode is a method the processor uses for developing argument addresses for machine instructions.

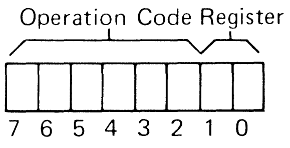
The 2650 processor can develop addresses in eight ways:

- Register addressing
- Immediate addressing
- Relative addressing
- Relative, indirect addressing
- Absolute addressing
- Absolute, indirect addressing
- Absolute, indexed addressing
- Absolute, indirect, indexed addressing

However, of these eight addressing modes, only four of them are basic. The others are variations due to indexing and indirection. The basic addressing mode of each instruction is indicated in parentheses in the first line of each detailed instruction description. The following text describes how effective addresses are developed by the processor.

REGISTER ADDRESSING

All register-to-register instructions are one byte in length. Instructions utilizing this addressing mode appear in this general format.

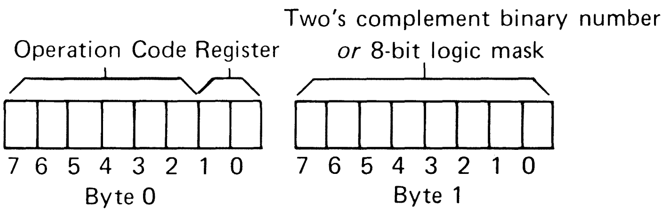


Since there are only two bits designated to specify a register, register zero always contains one of the operands while the other operand is in one of the three registers in the currently selected bank. Register zero may also be specified as the explicit operand giving instructions such as: LODZ R0.

In one byte register addressing instructions which have just one operand, any of the currently selected general purpose registers or register zero may be specified, e.g., RRL,R0.

IMMEDIATE ADDRESSING

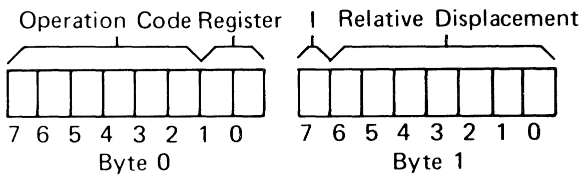
All immediate addressing instructions are two bytes in length. The first byte contains the operation code and register designation, while the second byte contains data used as the argument during instruction execution.



The second byte, the data byte, may contain a binary number or a logic mask depending on the particular instruction being executed. Any register may be designated in the first byte.

RELATIVE ADDRESSING

Relative addressing instructions are all two bytes in length and are memory reference instructions.One argument of the instruction is a register and the other argument is the contents of a memory location. The format of relative addressing instructions is:



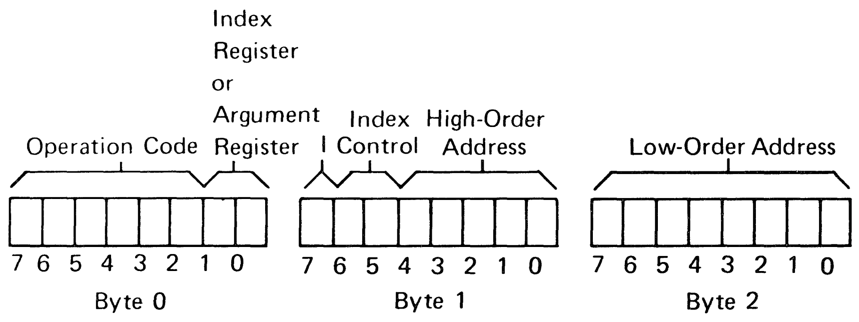
The first byte contains the operation code and register designation, while the second byte contains the relative address. Bits 0—6, byte 1, contain a 7-bit two's complement binary number which can range from −64 to +63. This number is used by the processor to calculate the effective address. The effective address is calculated by adding the address of the first byte following a relative addressing instruction to the relative displacement in the second byte of the instruction.

If bit 7, byte 1 is set to "1", the processor will enter an indirect addressing cycle, where the actual operand address will be accessed from the effective address location. See Indirect Addressing.

Two of the branch instructions (ZBSR, ZBRR) allow addressing relative to page zero, byte 0 of memory. In this case, values up to +63 reference the first 63 bytes of page zero and values up to -64 reference the last 64 bytes of page zero.

ABSOLUTE ADDRESSING FOR NON-BRANCH INSTRUCTIONS

Absolute addressing instructions are all three bytes in length and are memory reference instructions. One argument of the instruction is a register, designated in bits 1 and 0, byte 0; the other argument is the contents of a memory location. The format of absolute addressing instructions is:



Bits 4—0, byte 1 and 7—0, byte 2 contain the absolute address and can address any byte within the same page that the instruction appears.

The index control bits, bits #6 and #5, byte 1 determine how the effective address will be calculated and possibly which register will be the argument during instruction execution. The index control bits have the following interpretation:

Index Control Bit 6	Bit 5	Meaning
0	0	Non-indexed address
0	1	Indexed with auto-increment
1	0	Indexed with auto-decrement
1	1	Indexed only

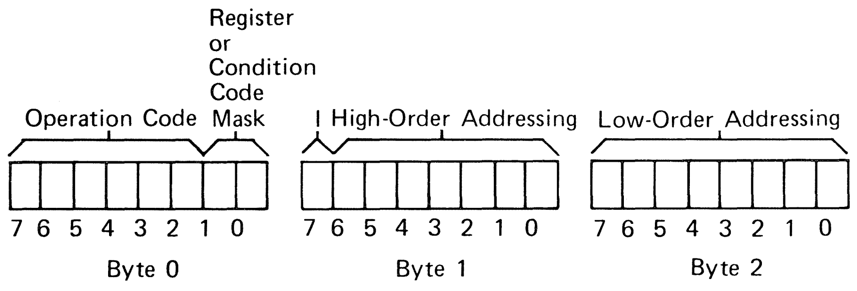
When the index control bits are 0 & 0, bits #1 and #0 in byte 0 contain the argument register designation and bits 0 to 4, byte 1 and bits 0 to 7, byte 2 contain the effective address. Indirect addressing may be specified by setting bit #7, byte 1 to a one.

When the index control bits are 1 & 1, bits #1 and #0 in byte 0 designate the index register and the argument register implicitly becomes register zero. The effective address is calculated by adding the contents of the index register (8-bit absolute integer) to the address field. If indirect addressing is specified, the indirect address is accessed and then the value in the index register is added to the indirect address. This is commonly called post indexing.

When the index control bits contain 0 & 1, the address is calculated by the processor exactly as when the control bits contain 1 & 1 *except* a binary 1 is added to the contents of the selected index register *before* the calculation of the effective address proceeds. Similarly, when the index control bits contain 1 & 0, a binary 1 is subtracted from the contents of the selected index register *before* the effective address is calculated.

ABSOLUTE ADDRESSING FOR BRANCH INSTRUCTIONS

The three byte, absolute addressing, branch instructions deviate slightly in format from ordinary absolute addressing instructions as shown below:



The notable difference is that bits 6 and 5, byte 1, are no longer interpreted as Index Control bits, but instead are interpreted as the high order bits of the address field. This means that there is no indexing allowed on most absolute addressing branch instructions, but indexed branches are possible through use of the BXA and BSXA instructions. The bits #6 and #5, byte 1, are used to set the current page register, thus enabling programs to directly transfer control to another page.

See the MEMORY ORGANIZATION, BXA and BSXA instructions, and INDIRECT ADDRESSING.

INDIRECT ADDRESSING

Indirect addressing means that the argument address of an instruction is not specified by the instruction itself, but rather the argument address will be found in the two bytes pointed to by the address field or relative address field, of absolute or relative addressing instructions. In the case of absolute addressing, the value of the index register is added to the indirect address *not* to the value in the address field of the instruction. In both cases, the processor will enter the indirect addressing state when the bit designated "I" is set to one. Entering the indirect addressing sequence adds two cycles (6 clock periods) to the execution time of an instruction.

Indirect addresses are 15-bit addresses stored right justified in two contiguous bytes of memory. As such, an indirect address may specify any location in addressable memory (0—32,767). The high order bit of the two byte indirect address is not used by the processor.

Only single level indirect addressing is implemented. The following examples demonstrate indirect addressing.

Example 1.

	0 0 0 0 1 1 1 0	1 0 0 0 0 0 0 0	0 1 0 1 0 0 0 1	LODA,R2	*H'51'
Address	10_{16}	11_{16}	12_{16}		
		0 0 0 0 0 0 0 1	0 0 1 0 1 0 0 0	ACON	H'128'
Address		51_{16}	52_{16}		
		0 1 1 0 0 1 1 1		DATA	H'67'
Address		128_{16}			

The LODA instruction in memory locations 10, 11, and 12 specifies indirect addressing (bit 7, byte 1, is set). Therefore, when the instruction is executed, the processor takes the address field value, H' 51', and uses it to access the two byte indirect address at 51 and 52. Then using the contents of 51 and 52 as the effective address, the data byte containing H' 67' is loaded into register 2.

Example 2.

	0 0 0 0 1 0 1 0	1 0 0 0 0 1 0 1		LODR,R2	*H'17'
Address	10_{16}	11_{16}			
		0 0 0 0 0 0 0 1	0 0 1 0 1 0 0 0	ACON	H'128'
Address		17_{16}	18_{16}		
		0 1 1 0 0 1 1 1		DATA	H'67'
Address		128_{16}			

In a fashion similar to the previous example, the relative address is used to access the indirect address which points to the data byte. When the LODR instruction is executed, the data byte contents, H' 67', will be loaded into register 2.

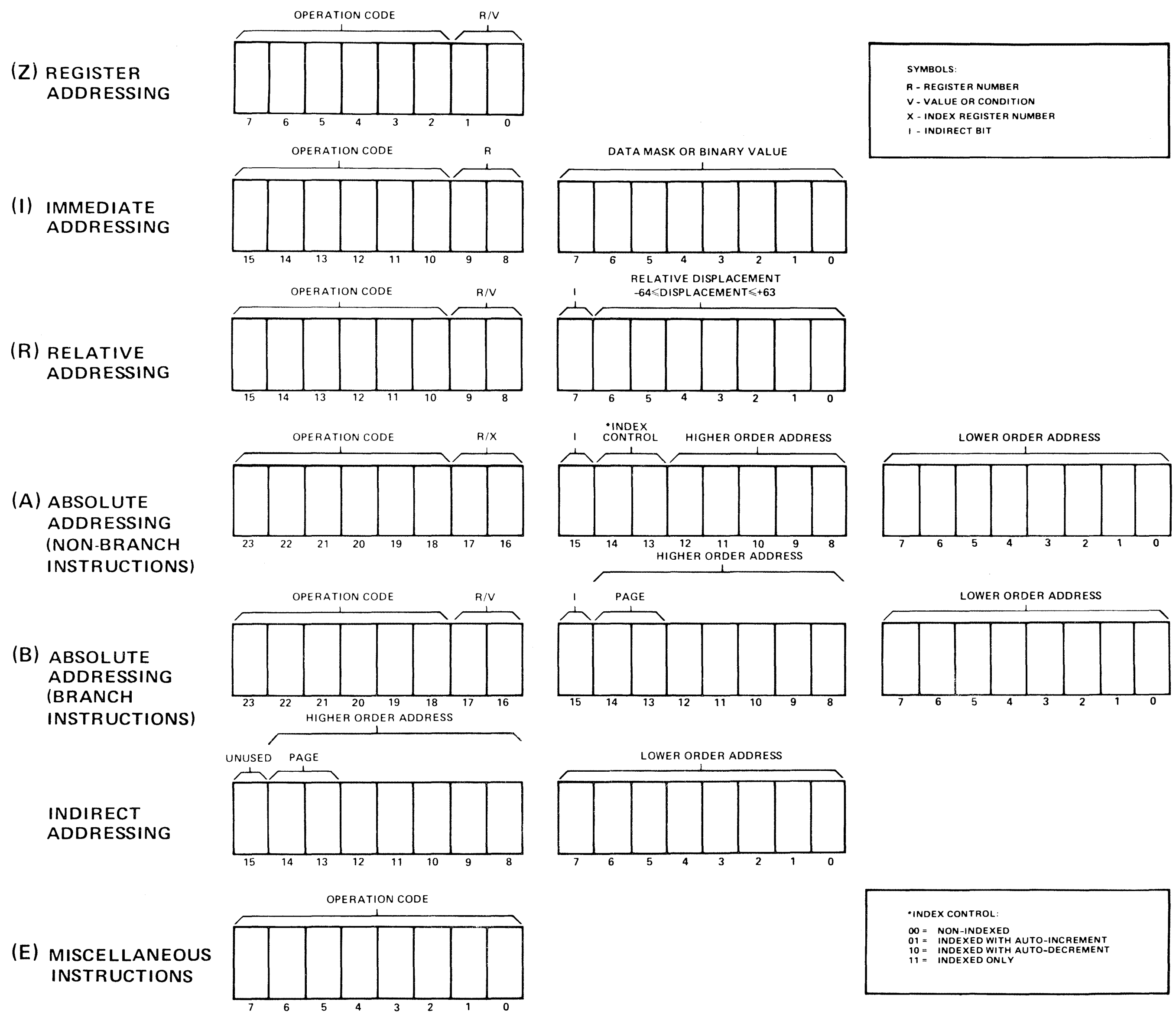
INSTRUCTION FORMAT EXCEPTIONS

There are several instructions which are detected by decoding the entire 8 bits of the first byte of the instruction. These instructions are unique and may be noticed in the instruction descriptions. Examples are: HALT, CPSU, CPSL.

Of this type of instruction, two operation codes were taken from otherwise complete sets thus eliminating certain possible operations. The cases are as follows:

(NOT OKAY)	STRZ 0	Storing register zero into register zero is not implemented, the operation code is used for NOP (no operation).
(OKAY)	NOP	
(NOT OKAY)	ANDZ 0	AND of register zero with register zero is not implemented, the operation code is used for HALT.
(OKAY)	HALT	

INSTRUCTION FORMATS



LOAD REGISTER ZERO

(Register Addressing)

Mnemonic LODZ r

Binary Coding

0	0	0	0	0	0	r	
7	6	5	4	3	2	1	0

Execution Time 2 cycles (6 clock periods)

Description

This one-byte instruction transfers the contents of the specified register, r, into register zero. The previous contents of register zero are lost. The contents of register r remain unchanged.

When the specified register, r, equals 0, the operation code is changed to 60_{16} by the assembler. The instruction, 00000000, yields indeterminate results.

Processor Registers Affected CC

Condition Code Setting

Register Zero	CC1	CC0
Positive	0	1
Zero	0	0
Negative	1	0

LOAD IMMEDIATE

(Immediate Addressing)

Mnemonic LODI,r v

Binary Coding

0	0	0	0	0	1	r	
7	6	5	4	3	2	1	0

			v				
7	6	5	4	3	2	1	0

Execution Time 2 cycles (6 clock periods)

Description

This two-byte instruction transfers the second byte of the instruction, v, into the specified register, r. The previous contents of r are lost.

Processor Registers Affected CC

Condition Code Setting

Register r	CC1	CC0
Positive	0	1
Zero	0	0
Negative	1	0

LOAD RELATIVE

(Relative Addressing)

Mnemonic LODR,r (*)a

Binary Coding

7	6	5	4	3	2	1	0
0	0	0	0	1	0	r	

7	6	5	4	3	2	1	0
I	a						

Execution Time 3 cycles (9 clock periods)

Description

This two-byte instruction transfers a byte of data from memory into the specified register, r. The data byte is found at the effective address formed by the addition of the a field and the address of the byte following this instruction. The previous contents of register r are lost. Indirect addressing may be specified.

Processor Registers Affected CC

Condition Code Setting

Register r	CC1	CC0
Positive	0	1
Zero	0	0
Negative	1	0

LOAD ABSOLUTE

(Absolute Addressing)

Mnemonic LODA,r (*)a(,X)

Binary Coding

0	0	0	0	1	1	r or X	
7	6	5	4	3	2	1	0

I	IC		a high order				
7	6	5	4	3	2	1	0

a low order							
7	6	5	4	3	2	1	0

Execution Time 4 cycles (12 clock periods)

Description

This three-byte instruction transfers a byte of data from memory into the specified register, r. The data byte is found at the effective address. If indexing is specified, bits 1 and 0, byte 0, indicate the index register and the destination of the operation implicitly becomes register zero. The previous contents of register r are lost.

Indirect addressing and/or indexing may be specified.

Processor Registers Affected CC

Condition Code Setting

Register r	CC1	CC0
Positive	0	1
Zero	0	0
Negative	1	0

STORE REGISTER ZERO

(Register Addressing)

Mnemonic STRZ r

Binary Code

1	1	0	0	0	0	r	
7	6	5	4	3	2	1	0

Execution Time 2 cycles (6 clock periods)

Description

This one-byte instruction transfers the contents of register zero into the specified register r. The previous contents of register r are lost. The contents of register zero remain unchanged.

Note: Register r may not be specified as zero. This operation code, ‘11000000’, is reserved for NOP.

Processor Registers Affected CC

Condition Code Setting

Register r	CC1	CC0
Positive	0	1
Zero	0	0
Negative	1	0

STORE RELATIVE

(Relative Addressing)

Mnemonic STRR,r (*)a

Binary Code

1	1	0	0	1	0	r		I	a						
7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0

Execution Time 3 cycles (9 clock periods)

Description

This two-byte instruction transfers a byte of data from the specified register, r, into the byte of memory pointed to by the effective address. The contents of register r remain unchanged and the contents of the memory byte are replaced.

Indirect addressing may be specified.

Processor Registers Affected None

Condition Code Setting N/A

STORE ABSOLUTE

(Absolute Addressing)

Mnemonic STRA,r (*)a(,X)

Binary Code

1	1	0	0	1	1	r	
7	6	5	4	3	2	1	0

I	IC		a high order				
7	6	5	4	3	2	1	0

a low order							
7	6	5	4	3	2	1	0

Execution Time 4 cycles (12 clock periods)

Description

This three-byte instruction transfers a byte of data from the specified register, r, into the byte of memory pointed to by the effective address. The contents of register r remain unchanged and the contents of the memory byte are replaced.

Indirect addressing and/or indexing may be specified. If indexing is specified, bits 1 and 0, byte 0, indicate the index register and the destination of the operation implicitly becomes register zero.

Processor Registers Affected	None
Condition Code Setting	N/A

ADD TO REGISTER ZERO

(Register Addressing)

Mnemonic ADDZ r

Binary Code

1	0	0	0	0	0	r	
7	6	5	4	3	2	1	0

Execution Time 2 cycles (6 clock periods)

Description

This one-byte instruction causes the contents of the specified register, r, and the contents of register zero to be added together in a true binary adder. The 8-bit sum of the addition replaces the contents of register zero. The contents of register r remain unchanged.

Note: Add with Carry may be effected. See Carry bit.

Processor Registers Affected C, CC, IDC, OVF

Condition Code Setting

Register Zero	CC1	CC0
Positive	0	1
Zero	0	0
Negative	1	0

ADD IMMEDIATE

(Immediate Addressing)

Mnemonic ADDI,r v

Binary Coding

7	6	5	4	3	2	1	0
1	0	0	0	0	1	r	

7	6	5	4	3	2	1	0
			v				

Execution Time 2 cycles (6 clock periods)

Description

This two-byte instruction causes the contents of register r and the contents of the second byte of this instruction to be added together in a true binary adder. The eight-bit sum replaces the contents of register r.

Note: Add with Carry may be effected. See Carry bit.

Processor Registers Affected C, CC, IDC, OVF

Condition Code Setting

Register r	CC1	CC0
Positive	0	1
Zero	0	0
Negative	1	0

ADD RELATIVE

(Relative Addressing)

Mnemonic ADDR,r (*)a

Binary Coding

1	0	0	0	1	0	r	
7	6	5	4	3	2	1	0

I	a						
7	6	5	4	3	2	1	0

Execution Time 3 cycles (9 clock periods)

Description

This two-byte instruction causes the contents of register r and the contents of the byte of memory pointed to by the effective address to be added together in a true binary adder. The eight-bit sum replaces the contents of register r.

Indirect addressing may be specified.

Note: Add with Carry may be effected. See Carry bit.

Processor Registers Affected C, CC, IDC, OVF

Condition Code Setting

Register r	CC1	CC0
Positive	0	1
Zero	0	0
Negative	1	0

ADD ABSOLUTE

(Absolute Addressing)

Mnemonic ADDA,r (*)a(,X)

Binary Coding

7	6	5	4	3	2	1	0
1	0	0	0	1	1	r	

7	6	5	4	3	2	1	0
I	IC		a high order				

7	6	5	4	3	2	1	0
a low order							

Execution Time 4 cycles (12 clock periods)

Description

This three-byte instruction causes the contents of register r and the contents of the byte of memory pointed to by the effective address to be added together in a true binary adder. The eight-bit sum replaces the contents of register r.

Indirect addressing and/or indexing may be specified. If indexing is specified, bits 1 and 0, byte 0, indicate the index register and the destination of the operation implicitly becomes register zero.

Note: Add with Carry may be effected. See Carry bit.

Processor Registers Affected C, CC, IDC, OVF

Condition Code Setting

Register r	CC1	CC0
Positive	0	1
Zero	0	0
Negative	1	0

SUBTRACT FROM REGISTER ZERO (Register Addressing)

Mnemonic SUBZ r

Binary Coding

1	0	1	0	0	0	r	
7	6	5	4	3	2	1	0

Execution Time 2 cycles (6 clock periods)

Description

This one-byte instruction causes the contents of the specified register r to be subtracted from the contents of register zero. The result of the subtraction replaces the contents of register zero.

The subtraction is performed by taking the binary two's complement of the contents of register r and adding that result to the contents of register zero. The contents of register r remain unchanged.

Note: Subtract with Borrow may be effected. See Carry bit.

Processor Registers Affected C, CC, IDC, OVF

Condition Code Setting

Register Zero	CC1	CC0
Positive	0	1
Zero	0	0
Negative	1	0

SUBTRACT IMMEDIATE

(Immediate Addressing)

Mnemonic SUBI,r v

Binary Code

1	0	1	0	0	1	r	
7	6	5	4	3	2	1	0

v							
7	6	5	4	3	2	1	0

Execution Time 2 cycles (6 clock periods)

Description

This two-byte instruction causes the contents of the second byte of this instruction to be subtracted from the contents of register r. The result of the subtraction replaces the contents of register r.

The subtraction is performed by taking the binary two's complement of the contents of the second instruction byte and adding that result to the contents of register r.

Note: Subtract with Borrow may be effected. See Carry bit.

Processor Registers Affected C, CC, IDC, OVF

Condition Code Setting

Register r	CC1	CC0
Positive	0	1
Zero	0	0
Negative	1	0

SUBTRACT RELATIVE

(Relative Addressing)

Mnemonic SUBR,r (*)a

Binary Code

7	6	5	4	3	2	1	0
1	0	1	0	1	0	r	

7	6	5	4	3	2	1	0
I	a						

Execution Time 3 cycles (9 clock periods)

Description

This two-byte instruction causes the contents of the byte of memory pointed to by the effective address to be subtracted from the contents of register r. The result of the subtraction replaces the contents of register r.

The subtraction is performed by taking the binary two's complement of the contents of the byte of memory and adding that result to the contents of register r.

Indirect addressing may be specified.

Note: Subtract with Borrow may be effected. See Carry bit.

Processor Registers Affected C, CC, IDC, OVF

Condition Code Setting

Register r	CC1	CC0
Positive	0	1
Zero	0	0
Negative	1	0

SUBTRACT ABSOLUTE

(Absolute Addressing)

Mnemonic SUBA,r (*)a(,X)

Binary Code

7	6	5	4	3	2	1	0
1	0	1	0	1	1	r	

7	6	5	4	3	2	1	0
I	IC		a high order				

7	6	5	4	3	2	1	0
a low order							

Execution Time 4 cycles (12 clock periods)

Description

This three-byte instruction causes the contents of the byte of memory pointed to by the effective address to be subtracted from the contents of register r. The result of the subtraction replaces the contents of register r.

The subtraction is performed by taking the binary two's complement of the contents of the memory byte and adding that result to the contents of register r.

Indirect addressing and/or indexing may be specified. If indexing is specified, bits 1 and 0, byte 0, indicate the index register and the destination of the operation implicitly becomes register zero.

Note: Subtract with Borrow may be effected. See Carry bit.

Processor Registers Affected C, CC, IDC, OVF

Condition Code Setting

Register r	CC1	CC0
Positive	0	1
Zero	0	0
Negative	1	0

AND TO REGISTER ZERO (Register Addressing)

Mnemonic ANDZ r

Binary Code

7	6	5	4	3	2	1	0
0	1	0	0	0	0	r	

Execution Time 2 cycles (6 clock periods)

Description

This one-byte instruction causes the contents of the specified register, r, to be logically ANDed with the contents of register zero. The result of the operation replaces the contents of register zero. The contents of register r remain unchanged.

The AND operation treats each bit of the argument bytes as in the truth table below:

Bit (0-7)	Bit (0-7)	AND Result
0	0	0
0	1	0
1	1	1
1	0	0

Note: Register r may not be specified as zero. This operation code, '01000000', is reserved for HALT.

Processor Registers Affected CC

Condition Code Setting

Register Zero	CC1	CC0
Positive	0	1
Zero	0	0
Negative	1	0

AND IMMEDIATE

(Immediate Addressing)

Mnemonic ANDI,r v

Binary Code

0	1	0	0	0	1	r						v			
7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0

Execution Time 2 cycles (6 clock periods)

Description

This two-byte instruction causes the contents of the specified register r to be logically ANDed with the contents of the second byte of this instruction. The result of this operation replaces the contents of register r.

The AND operation treats each bit of the argument bytes as in the truth table below:

Bit (0-7)	Bit (0-7)	AND Result
0	0	0
0	1	0
1	1	1
1	0	0

Processor Registers Affected CC

Condition Code Setting

Register Zero	CC1	CC0
Positive	0	1
Zero	0	0
Negative	1	0

AND RELATIVE

(Relative Addressing)

Mnemonic ANDR,r (*)a

Binary Code

7	6	5	4	3	2	1	0
0	1	0	0	1	0	r	

7	6	5	4	3	2	1	0
I	a						

Execution Time 3 cycles (9 clock periods)

Description

This two-byte instruction causes the contents of the specified register r to be logically ANDed with the contents of the memory byte pointed to by the effective address. The result of this operation replaces the contents of register r.

The AND operation treats each bit of the argument bytes as in the truth table below:

Bit (0-7)	Bit (0-7)	AND Result
0	0	0
0	1	0
1	1	1
1	0	0

Processor Registers Affected CC

Condition Code Setting

Register Zero	CC1	CC0
Positive	0	1
Zero	0	0
Negative	1	0

AND ABSOLUTE

(Absolute Addressing)

Mnemonic ANDA,r (*)a(,X)

Binary Code

7	6	5	4	3	2	1	0
0	1	0	0	1	1	r	

7	6	5	4	3	2	1	0
I	IC		a high order				

7	6	5	4	3	2	1	0
a low order							

Execution Time 4 cycles (12 clock periods)

Description

This three-byte instruction causes the contents of Register r to be logically ANDed with the contents of memory byte pointed to by the effective address. The result of the operation replaces the contents of register r.

The AND operation treats each bit of the argument bytes as in the truth table below:

Bit (0-7)	Bit (0-7)	AND Result
0	0	0
0	1	0
1	1	1
1	0	0

Indirect addressing and/or indexing may be specified. If indexing is specified, bits 1 and 0, byte 0, indicate the index register and the destination of the operation implicitly becomes register zero.

Processor Registers Affected CC

Condition Code Setting

Register Zero	CC1	CC0
Positive	0	1
Zero	0	0
Negative	1	0

INCLUSIVE OR TO REGISTER ZERO

(Register Addressing)

Mnemonic IORZ r

Binary Code

0	1	1	0	0	0	r	
7	6	5	4	3	2	1	0

Execution Time 2 cycles (6 clock periods)

Description

This one-byte instruction causes the contents of the specified register, r, to be logically Inclusive ORed with the contents of register zero. The result of this operation replaces the contents of register zero. The contents of register r remain unchanged.

The Inclusive OR operation treats each bit of the argument bytes as in the truth table below:

Bit (0-7)	Bit (0-7)	Inclusive OR Result
0	0	0
0	1	1
1	1	1
1	0	1

Processor Registers Affected CC

Condition Code Setting

Register Zero	CC1	CC0
Positive	0	1
Zero	0	0
Negative	1	0

INCLUSIVE OR IMMEDIATE

(Immediate Addressing)

Mnemonic IORI,r v

Binary Code

7	6	5	4	3	2	1	0
0	1	1	0	0	1	r	

7	6	5	4	3	2	1	0
			v				

Execution Time 2 cycles (6 clock periods)

Description

This two-byte instruction causes the contents of the specified register r to be logically Inclusive ORed with the contents of the second byte of this instruction. The result of this operation replaces the contents of register r.

The Inclusive OR operation treats each bit of the argument bytes as in the truth table below:

Bit (0-7)	Bit (0-7)	Inclusive OR Result
0	0	0
0	1	1
1	1	1
1	0	1

Processor Registers Affected CC

Condition Code Setting

Register r	CC1	CC0
Positive	0	1
Zero	0	0
Negative	1	0

INCLUSIVE OR RELATIVE

(Relative Addressing)

Mnemonic IORR,r (*)a

Binary Code

7	6	5	4	3	2	1	0
0	1	1	0	1	0	r	

7	6	5	4	3	2	1	0
I	a						

Execution Time 3 cycles (9 clock periods)

Description

This two-byte instruction causes the contents of the specified register r to be logically Inclusive ORed with the contents of the memory byte pointed to by the effective address. The result of this operation replaces the previous contents of register r.

Indirect addressing may be specified.

The Inclusive OR operation treats each bit of the argument byte as in the truth table below:

Bit (0-7)	Bit (0-7)	Inclusive OR Result
0	0	0
0	1	1
1	1	1
1	0	1

Processor Registers Affected CC

Condition Code Setting

Register r	CC1	CC0
Positive	0	1
Zero	0	0
Negative	1	0

INCLUSIVE OR ABSOLUTE (Absolute Addressing)

Mnemonic IORA,r (*)a(,X)

Binary Code

0	1	1	0	1	1	r	
7	6	5	4	3	2	1	0

I	IC		a high order				
7	6	5	4	3	2	1	0

a low order							
7	6	5	4	3	2	1	0

Execution Time 4 cycles (12 clock periods)

Description

This three-byte instruction causes the contents of register r to be logically Inclusive ORed with the contents of the memory byte pointed to by the effective address. The result of the operation replaces the previous contents of register r.

Indirect addressing and/or indexing may be specified. If indexing is specified, bits 1 and 0, byte 0, indicate the index register and the destination of the operation implicitly becomes register zero.

The Inclusive OR operation treats each bit of the argument bytes as in the truth table below:

Bit (0-7)	Bit (0-7)	Inclusive OR Result
0	0	0
0	1	1
1	1	1
1	0	1

Processor Registers Affected CC

Condition Code Setting

Register Zero	CC1	CC0
Positive	0	1
Zero	0	0
Negative	1	0

EXCLUSIVE OR TO REGISTER ZERO (Register Addressing)

Mnemonic EORZ r

Binary Code

7	6	5	4	3	2	1	0
0	0	1	0	0	0	r	

Execution Time 2 cycles (6 clock periods)

Description

This one-byte instruction causes the contents of the specified register r to be logically Exclusive ORed with the contents of register zero. The result of this operation replaces the contents of register zero. The contents of register r remain unchanged.

The Exclusive OR operation treats each bit of the argument bytes as in the truth table below:

Bit (0-7)	Bit (0-7)	Exclusive OR Result
0	0	0
0	1	1
1	1	0
1	0	1

Processor Registers Affected CC

Condition Code Setting

Register Zero	CC1	CC0
Positive	0	1
Zero	0	0
Negative	1	0

EXCLUSIVE OR IMMEDIATE (Immediate Addressing)

Mnemonic EORI,r v

Binary Code

7	6	5	4	3	2	1	0
0	0	1	0	0	1	r	

7	6	5	4	3	2	1	0
				v			

Execution Time 2 cycles (6 clock periods)

Description

This two-byte instruction causes the contents of the specified register r to be logically Exclusive ORed with the contents of the second byte of this instruction. The result of this operation replaces the previous contents of register r.

The Exclusive OR operation treats each bit of the argument bytes as in the truth table below:

Bit (0-7)	Bit (0-7)	Exclusive OR Result
0	0	0
0	1	1
1	1	0
1	0	1

Processor Registers Affected CC

Condition Code Setting

Register r	CC1	CC0
Positive	0	1
Zero	0	0
Negative	1	0

EXCLUSIVE OR RELATIVE

(Relative Addressing)

Mnemonic EORR,r (*)a

Binary Code

0	0	1	0	1	0	r			I	a						
7	6	5	4	3	2	1	0		7	6	5	4	3	2	1	0

Execution Time 3 cycles (9 clock periods)

Description

This two-byte instruction causes the contents of the specified register r to be logically Exclusive ORed with the contents of the memory byte pointed to by the effective address. The result of this operation replaces the previous contents of register r.

Indirect addressing may be specified.

The Exclusive OR operation treats each bit of the argument bytes as in the truth table below:

Bit (0-7)	Bit (0-7)	Exclusive OR Result
0	0	0
0	1	1
1	1	0
1	0	1

Processor Registers Affected CC

Condition Code Setting

Register r	CC1	CC0
Positive	0	1
Zero	0	0
Negative	1	0

EXCLUSIVE OR ABSOLUTE

(Absolute Addressing)

Mnemonic EORA,r (*)a(,X)

Binary Code

0	0	1	0	1	1	r	
7	6	5	4	3	2	1	0

I	IC		a high order				
7	6	5	4	3	2	1	0

a low order							
7	6	5	4	3	2	1	0

Execution Time 4 cycles (12 clock periods)

Description

This three-byte instruction causes the contents of register r to be Exclusive ORed with the contents of the memory byte pointed to by the effective address. The result of the operation replaces the previous contents of register r.

Indirect addressing and/or indexing may be specified. If indexing is specified, bits 1 and 0, byte 0, indicate the index register and the destination of the operation implicitly becomes register zero.

The Exclusive OR operation treats each bit of the argument bytes as in the truth table below:

Bit (0-7)	Bit (0-7)	Exclusive OR Result
0	0	0
0	1	1
1	1	0
1	0	1

Processor Registers Affected CC

Condition Code Setting

Register r	CC1	CC0
Positive	0	1
Zero	0	0
Negative	1	0

COMPARE TO REGISTER ZERO (Register Addressing)

Mnemonic COMZ r

Binary Code

1	1	1	0	0	0	r	
7	6	5	4	3	2	1	0

Execution Time 2 cycles (6 clock periods)

Description

This one-byte instruction causes the contents of the specified register r to be compared to the contents of register zero. The comparison will be performed in either "arithmetic" or "logical" mode depending on the setting of the COM bit in the Program Status Word.

When COM=1 (logical mode) the values will be interpreted as 8-bit positive binary numbers; when COM=0, the values will be interpreted as 8-bit two's complement numbers.

The execution of this instruction *only* causes the Condition Code to be set as in the following table.

Processor Registers Affected CC

Condition Code Setting

	CC1	CC0
Register zero greater than Register r	0	1
Register zero equal to Register r	0	0
Register zero less than Register r	1	0

COMPARE IMMEDIATE

(Immediate Addressing)

Mnemonic COMI,r v

Binary Code

7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0
1	1	1	0	0	1	r		v							

Execution Time 2 cycles (6 clock periods)

Description

This two-byte instruction causes the contents of the specified register r to be compared to the contents of the second byte of this instruction. The comparison will be performed in either the "arithmetic" or "logical" mode depending on the setting of the COM bit in the Program Status Word.

When COM=1 (logical mode), the values will be treated as 8-bit positive binary numbers; when COM=0, the values will be treated as 8-bit two's complement numbers.

The execution of this instruction *only* causes the Condition Code to be set as in the following table.

Processor Registers Affected CC

Condition Code Setting

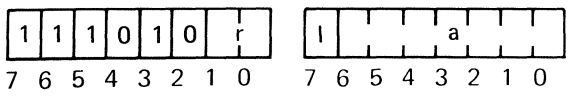
	CC1	CC0
Register r greater than v	0	1
Register r equal to v	0	0
Register r less than v	1	0

COMPARE RELATIVE

(Relative Addressing)

Mnemonic COMR,r (*)a

Binary Code



Execution Time 3 cycles (9 clock periods)

Description

This two-byte instruction causes the contents of the specified register r to be compared to the contents of the memory byte pointed to by the effective address. The comparison will be performed in either the "arithmetic" or "logical" mode depending upon the setting of the COM bit in the Program Status Word.

When COM=1 (logical mode), the values will be treated as 8-bit positive binary numbers; when COM=0, the values will be treated as 8-bit, two's complement numbers.

The execution of this instruction *only* causes the Condition Code to be set as in the following table.

Processor Registers Affected CC

Condition Code Setting

	CC1	CC0
Register r greater than memory byte	0	1
Register r equal to memory byte	0	0
Register r less than memory byte	1	0

COMPARE ABSOLUTE

(Absolute Addressing)

Mnemonic COMA,r (*)a(,X)

Binary Code

1	1	1	0	1	1	r	
7	6	5	4	3	2	1	0

I	IC		a high order				
7	6	5	4	3	2	1	0

a low order							
7	6	5	4	3	2	1	0

Execution Time 4 cycles (12 clock periods)

Description

This three-byte instruction causes the contents of register r to be compared to the contents of the memory byte pointed to by the effective address. The comparison will be performed in either the "arithmetic" or "logical" mode depending on the setting of the COM bit in the Program Status Word.

Where COM=1 (logical mode), the values will be treated as 8-bit, positive binary numbers; when COM=0 (arithmetic mode), the values will be treated as 8-bit, two's complement numbers.

Indirect addressing and/or indexing may be specified. If indexing is specified, bits 1 and 0, byte 0, indicate the index register and the destination of the operation implicitly becomes register zero.

The execution of this instruction *only* causes the Condition Code to be set as in the following table.

Processor Registers Affected CC

Condition Code Setting

	CC1	CC0
Register r greater than memory byte	0	1
Register r equal to memory byte	0	0
Register r less than memory byte	1	0

ROTATE REGISTER LEFT

(Register Addressing)

Mnemonic RRL,r

Binary Code

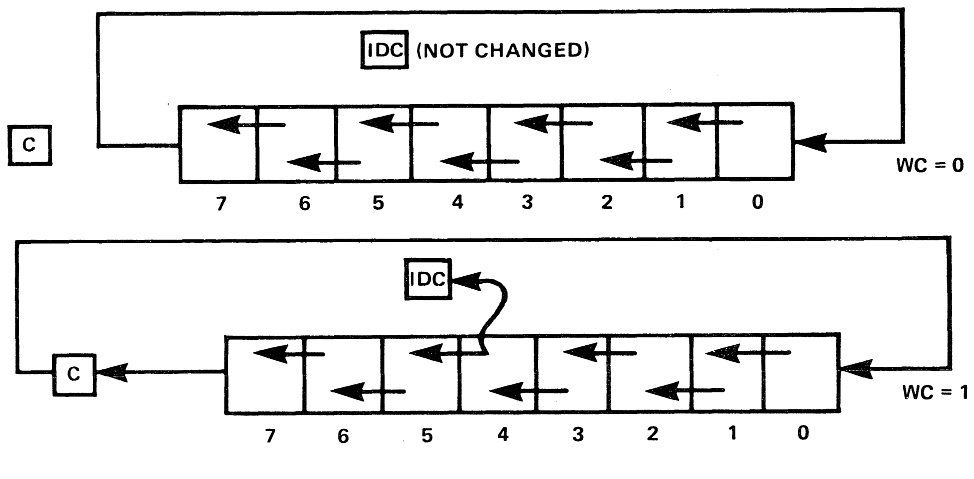
7	6	5	4	3	2	1	0
1	1	0	1	0	0	r	

Execution Time 2 cycles (6 clock periods)

Description

This one-byte instruction causes the contents of the specified register r to be shifted left one bit. If the WC bit in the Program Status Word is set to zero, bit #7 of register r flows into bit #0; if WC=1, then bit #7 flows into the Carry bit and the Carry bit flows into bit #0.

Register bit #4 flows into the IDC if WC=1.



Note: Whenever a rotate causes bit #7 of the specified register to change polarity, the OVF bit is set in the PSL.

Processor Registers Affected C, CC, IDC, OVF

Condition Code Setting

Register r	CC1	CC0
Positive	0	1
Zero	0	0
Negative	1	0

ROTATE REGISTER RIGHT (Register Addressing)

Mnemonic RRR,r

Binary Code

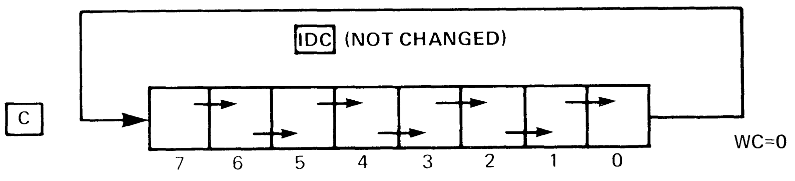
0	1	0	1	0	0	r	
7	6	5	4	3	2	1	0

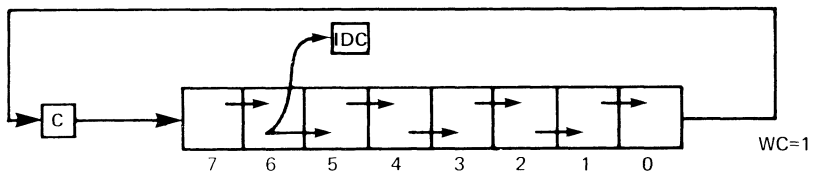
Execution Time 2 cycles (6 clock periods)

Description

This one-byte instruction causes the contents of the specified register r to be shifted right one bit. If the WC bit in the Program Status Word is set to zero, bit #0 of the register r flows into bit #7; if WC=1, then bit #0 of the register r flows into the Carry bit and the Carry bit flows into bit #7.

Register bit #6 flows into the IDC if WC=1.





Note: Whenever a rotate causes bit #7 of the specified register to change polarity, the OVF bit is set in the PSL.

Processor Registers Affected C, CC, IDC, OVF

Condition Code Setting

Register r	CC1	CC0
Positive	0	1
Zero	0	0
Negative	1	0

LOAD PROGRAM STATUS, UPPER

Mnemonic LPSU

Binary Code

1	0	0	1	0	0	1	0
7	6	5	4	3	2	1	0

Execution Time 2 cycles (6 clock periods)

Description

This one-byte instruction causes the current contents of the Upper Program Status Byte to be replaced with the contents of register zero.

See Program Status Word description for bit assignments. Bits #4 and #3 of the PSU are unassigned and will always be regarded as containing zeroes.

Processor Registers Affected F, II, SP

Condition Code Setting N/A

LOAD PROGRAM STATUS, LOWER

Mnemonic LPSL

Binary Code

1	0	0	1	0	0	1	1
7	6	5	4	3	2	1	0

Execution Time 2 cycles (6 clock periods)

Description

This one-byte instruction causes the current contents of the Lower Program Status Byte to be replaced with the contents of register zero.

See Program Status Word description for bit assignments.

Processor Registers Affected CC, IDC, RS, WC, OVF, COM, C

Condition Code Setting

The CC will take on the value in bits #7 and #6 of register zero.

STORE PROGRAM STATUS, UPPER

Mnemonic	SPSU

Binary Code

0	0	0	1	0	0	1	0
7	6	5	4	3	2	1	0

Execution Time	2 cycles (6 clock periods)

Description

This one-byte instruction causes the contents of the Upper Program Status Byte to be transferred into register zero.

See Program Status Word description for bit assignments. Bits #4 and #3 which are unassigned will always be stored as zeroes.

Processor Registers Affected CC

Condition Code Setting

Register Zero	CC1	CC0
Positive	0	1
Zero	0	0
Negative	1	0

STORE PROGRAM STATUS, LOWER

Mnemonic SPSL

Binary Code

0	0	0	1	0	0	1	1
7	6	5	4	3	2	1	0

Execution Time 2 cycles (6 clock periods)

Description

This one-byte instruction causes the contents of the Lower Program Status Byte to be transferred into register zero.

See Program Status Word description for bit assignments.

Processor Registers Affected CC

Condition Code Setting

Register Zero	CC1	CC0
Positive	0	1
Zero	0	0
Negative	1	0

PRESET PROGRAM STATUS UPPER, SELECTIVE (Immediate Addressing)

Mnemonic PPSU v

Binary Code

0	1	1	1	0	1	1	0					v			
7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0

Execution Time 3 cycles (9 clock periods)

Description

This two-byte instruction causes individual bits in the Upper Program Status Byte to be selectively set to binary one. When this instruction is executed, each bit in the v field of the second byte of this instruction is tested for the presence of a one and if a particular bit in the v field contains a one, the corresponding bit in the status byte is set to binary one. Any bits in the status byte which are not selected are not modified.

Processor Registers Affected F, II, SP

Condition Code Setting N/A

PRESET PROGRAM STATUS LOWER, SELECTIVE (Immediate Addressing)

Mnemonic PPSL v

Binary Code

0	1	1	1	0	1	1	1					v			
7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0

Execution Time 3 cycles (9 clock periods)

Description

This two-byte instruction causes individual bits in the Lower Program Status Byte to be selectively set to binary one. When this instruction is executed, each bit in the v field of the second byte of this instruction is tested for the presence of a one and if a particular bit in the v field contains a one, the corresponding bit in the status byte is set to binary one. Any bits in the status byte which are not selected are not modified.

Processor Registers Affected CC, IDC, RS, WC, OVF, COM, C

Condition Code Setting

The CC bits may be set by the execution of this instruction.

CLEAR PROGRAM STATUS UPPER, SELECTIVE (Immediate Addressing)

Mnemonic CPSU v

Binary Code

0	1	1	1	0	1	0	0		v							
7	6	5	4	3	2	1	0		7	6	5	4	3	2	1	0

Execution Time 3 cycles (9 clock periods)

Description

This two-byte instruction causes individual bits in the Upper Program Status Byte to be selectively cleared. When this instruction is executed, each bit in the v field of the second byte of this instruction is tested for the presence of a one and if a particular bit in the v field contains a one, the corresponding bit in the status byte is cleared to zero. Any bits in the status byte which are not selected are not modified.

Processor Registers Affected F, II, SP

Condition Code Setting N/A

CLEAR PROGRAM STATUS LOWER, SELECTIVE (Immediate Addressing)

Mnemonic CPSL v

Binary Code

0	1	1	1	0	1	0	1					v			
7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0

Execution Time 3 cycles (9 clock periods)

Description

This two-byte instruction causes individual bits in the Lower Program Status Byte to be selectively cleared. When this instruction is executed, each bit in the v field of the second byte of this instruction is tested for the presence of a one and if a particular bit in the v field contains a one, the corresponding bit in the status byte is cleared to zero. Any bits in the status byte which are not selected are not modified.

Processor Registers Affected CC, IDC, RS, WC, OVF, COM, C

Condition Code Setting

The CC bits may be cleared by the execution of this instruction.

TEST PROGRAM STATUS UPPER, SELECTIVE (Immediate Addressing)

Mnemonic TPSU v

Binary Code

1	0	1	1	0	1	0	0	v							
7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0

Execution Time 3 cycles (9 clock periods)

Description

This two-byte instruction tests individual bits in the Upper Program Status Byte to determine if they are set to binary one. When this instruction is executed, each bit in the v field of this instruction is tested for the presence of a one, and if a particular bit in the v field contains a one, the corresponding bit in the status byte is tested for a one or zero. The Condition Code is set to reflect the result of this operation.

If a bit in the v field is zero, the corresponding bit in the status byte is not tested.

Processor Registers Affected CC

Condition Code Setting

	CC1	CC0
All of the selected bits in PSU are 1s	0	0
Not all of the selected bits in PSU are 1s	1	0

TEST PROGRAM STATUS LOWER, SELECTIVE (Immediate Addressing)

Mnemonic TPSL v

Binary Code

1	0	1	1	0	1	0	1					v			
7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0

Execution Time 3 cycles (9 clock periods)

Description

This two-byte instruction tests individual bits in the Lower Program Status Byte to determine if they are set to binary one. When this instruction is executed, each bit in the v field of this instruction is tested for a one, and if a particular bit in the v field contains a one, the corresponding bit in the status byte is tested for a one or zero. The Condition Code is set to reflect the result of this operation.

Processor Registers Affected CC

Condition Code Setting

	CC1	CC0
All of the selected bits in PSL are 1s	0	0
Not all of the selected bits in PSL are 1s	1	0

ZERO BRANCH RELATIVE

(Relative Addressing)

Mnemonic ZBRR (*) a

Binary Code

1	0	0	1	1	0	1	1	I				a			
7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0

Execution Time 3 cycles (9 clock periods)

Description

This two-byte unconditional relative branch instruction directs the processor to calculate the effective address differently than the usual calculation for the Relative Addressing mode.

The specified value, a, is interpreted as a relative displacement from page zero, byte zero. Therefore, displacement may be specified from −64 to +63 bytes. The address calculation is modulo 8192_{10}, so the negative displacement actually will develop addresses at the end of page zero. For example, ZBRR −8, will develop an effective address of 8184_{10}, and a ZBRR +52 will develop an effective address of 52_{10}.

This instruction causes the processor to clear, address bits 13 and 14, the page address bits; and to replace the contents of the Instruction Address Register with the effective address of the instruction. This instruction may be executed anywhere within addressable memory.

Indirect addressing may be specified.

Processor Registers Affected None

Condition Code Setting N/A

BRANCH ON CONDITION TRUE, RELATIVE

(Relative Addressing)

Mnemonic	BCTR,v	(*)a

Binary Code

0	0	0	1	1	0	v	
7	6	5	4	3	2	1	0

I				a			
7	6	5	4	3	2	1	0

Execution Time 3 cycles (9 clock periods)

Description

This two-byte conditional branch instruction causes the processor to fetch the next instruction to be executed from the memory location pointed to by the effective address only if the two-bit v field matches the current Condition Code field (CC) in the Program Status Word.

If the v field and CC field do not match, the next instruction is fetched from the location following the second byte of this instruction.

Indirect addressing may be specified.

If the v field is set to 3_{16}, an unconditional branch is effected.

Processor Registers Affected	None
Condition Code Setting	N/A

BRANCH ON CONDITION TRUE, ABSOLUTE (Absolute Addressing)

Mnemonic BCTA,v (*)a

Binary Code

0	0	0	1	1	1	v		I	a high order							a low order							
7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0

Execution Time 3 cycles (9 clock periods)

Description

This three-byte conditional branch instruction causes the processor to fetch the next instruction to be executed from the memory location pointed to by the effective address only if the two-bit v field matches the two-bit Condition Code field (CC) in the Program Status Word.

If the v field and CC field do not match, the next instruction is fetched from the location following the second byte of this instruction.

Indirect addressing may be specified.

If the v field is set to 3_{16}, an unconditional branch is effected.

Processor Registers Affected None

Condition Code Setting N/A

BRANCH ON CONDITION FALSE, RELATIVE (Relative Addressing)

Mnemonic BCFR,v (*)a

Binary Code

1	0	0	1	1	0	v		I	a						
7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0

Execution Time 3 cycles (9 clock periods)

Description

This two-byte branch instruction causes the processor to fetch the next instruction to be executed from the memory location pointed to by the effective address only if the two-bit v field does not match the two-bit Condition Code field (CC) in the Program Status Word. If there is no match, the contents of the Instruction Address Register are replaced by the effective address.

If the v field and CC field match, the next instruction is fetched from the location following the second byte of this instruction.

Indirect addressing may be specified.

The v field may not be set to 3_{16} as this bit combination is used for the ZBRR operation code.

Processor Registers Affected None

Condition Code Setting N/A

BRANCH ON CONDITION FALSE, ABSOLUTE (Absolute Addressing)

Mnemonic BCFA,v (*)a

Binary Code

1	0	0	1	1	1	v		I	a high order							a low order							
7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0

Execution Time 3 cycles (9 clock periods)

Description

This three-byte instruction causes the processor to fetch the next instruction to be executed from the memory location pointed to by the effective address only if the two-bit v field does not match the two-bit Condition Code field (CC) in the Program Status Word. If there is no match, the contents of the Instruction Address Register are replaced by the effective address.

If the v field and CC field match, the next instruction is fetched from the location following the second byte of this instruction.

Indirect addressing may be specified.

The v field may not be set to 3_{16} as this bit combination is used for the BXA operation code.

Processor Registers Affected	None
Condition Code Setting	N/A

BRANCH ON INCREMENTING REGISTER, RELATIVE (Relative Addressing)

Mnemonic BIRR,r (*)a

Binary Code

1	1	0	1	1	0	r			I	a						
7	6	5	4	3	2	1	0		7	6	5	4	3	2	1	0

Execution Time 3 cycles (9 clock periods)

Description

This two-byte branch instruction causes the processor to increment the contents of the specified register by one. If the new value in the register is non-zero, the next instruction to be executed is taken from the memory location pointed to by the effective address, i.e., the effective address replaces the previous contents of the Instruction Address Register. If the new value in register r is zero, the next instruction to be executed follows the second byte of this instruction.

Indirect addressing may be specified.

Processor Registers Affected None

Condition Code Setting N/A

BRANCH ON INCREMENTING REGISTER, ABSOLUTE

(Absolute Addressing)

Mnemonic BIRA,r (*)a

Binary Code

7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0
1	1	0	1	1	1	r		I	a high order							a low order							

Execution Time 3 cycles (9 clock periods)

Description

This three-byte branch instruction causes the processor to increment the contents of the specified register by one. If the new value in the register is non-zero, the next instruction to be executed is taken from the memory location pointed to by the effective address, i.e., the effective address replaces the previous contents of the Instruction Address Register. If the new value of register r is zero, the next instruction to be executed follows the second byte of this instruction.

Indirect addressing may be specified.

Processor Registers Affected None

Condition Code Setting N/A

BRANCH ON DECREMENTING REGISTER, RELATIVE

(Relative Addressing)

Mnemonic BDRR,r (*)a

Binary Code

1	1	1	1	1	0	r	
7	6	5	4	3	2	1	0

I			a				
7	6	5	4	3	2	1	0

Execution Time 3 cycles (9 clock periods)

Description

This two-byte branch instruction causes the processor to decrement the contents of the specified register by one. If the new value in the register is non-zero, the next instruction to be executed is taken from the memory location pointed to by the effective address, i.e., the effective address replaces the previous contents of the Instruction Address Register. If the new value in register r is zero, the next instruction to be executed follows the second byte of this instruction.

Indirect addressing may be specified.

Processor Registers Affected None

Condition Code Setting N/A

BRANCH ON DECREMENTING REGISTER, ABSOLUTE

(Absolute Addressing)

Mnemonic BDRA,r (*)a

Binary Code

1	1	1	1	1	1	r	
7	6	5	4	3	2	1	0

I	a high order						
7	6	5	4	3	2	1	0

a low order							
7	6	5	4	3	2	1	0

Execution Time 3 cycles (9 clock periods)

Description

This three-byte instruction causes the processor to decrement the contents of the specified register by one. If the new value in the register is non-zero, the next instruction to be executed is taken from the memory location pointed to by the effective address, i.e., the effective address replaces the previous contents of the Instruction Address Register. If the new address in register r is zero, the next instruction to be executed follows the second byte of this instruction.

Indirect addressing may be specified.

Processor Registers Affected	None
Condition Code Setting	N/A

BRANCH ON REGISTER NON-ZERO, RELATIVE (Relative Addressing)

Mnemonic BRNR,r (*)a

Binary Code

0	1	0	1	1	0	r	
7	6	5	4	3	2	1	0

I			a				
7	6	5	4	3	2	1	0

Execution Time 3 cycles (9 clock periods)

Description

This two-byte branch instruction causes the contents of the specified register r to be tested for a non-zero value. If the register contains a non-zero value, the next instruction to be executed is taken from the location pointed to by the effective address, i.e., the effective address replaces the current contents of the Instruction Address Register.

If the specified register contains a zero value, the next instruction is fetched from the location following the second byte of this instruction.

Indirect addressing may be specified.

Processor Registers Affected	None
Condition Code Setting	N/A

BRANCH ON REGISTER NON-ZERO, ABSOLUTE (Absolute Addressing)

Mnemonic BRNA,r (*)a

Binary Code

0	1	0	1	1	1	r	
7	6	5	4	3	2	1	0

I	a high order						
7	6	5	4	3	2	1	0

a low order							
7	6	5	4	3	2	1	0

Execution Time 3 cycles (9 clock periods)

Description

The three-byte branch instruction causes the contents of the specified register r to be tested for a non-zero value. If the register contains a non-zero value, the next instruction to be executed is taken from the location pointed to by the effective address, i.e., the effective address replaces the contents of the Instruction Address Register.

If the specified register contains a zero value, the next instruction is fetched from the location following the third byte of this instruction.

Indirect addressing may be specified.

Processor Registers Affected None

Condition Code Setting N/A

BRANCH INDEXED, ABSOLUTE

(Absolute Addressing)

Mnemonic BXA (*)a,X

Binary Code

1	0	0	1	1	1	1	1
7	6	5	4	3	2	1	0

I	a high order						
7	6	5	4	3	2	1	0

a low order							
7	6	5	4	3	2	1	0

Execution Time 3 cycles (9 clock periods)

Description

This three-byte branch instruction causes the processor to perform an unconditional branch. Indexing is required and register #3 must be specified as the index register because the entire first byte of this instruction is decoded by the processor. When executed, the content of the Instruction Address Register (IAR) is replaced by the effective address.

If indirect addressing is specified, the value in the index register is added to the indirect address to calculate the effective branch address.

Processor Registers Affected None

Condition Code Setting N/A

ZERO BRANCH TO SUBROUTINE, RELATIVE (Relative Addressing)

Mnemonic ZBSR (*)a

Binary Code

1	0	1	1	1	0	1	1
7	6	5	4	3	2	1	0

I				a			
7	6	5	4	3	2	1	0

Execution Time 3 cycles (9 clock periods)

Description

This two-byte unconditional subroutine branch instruction directs the processor to calculate the effective address differently than the usual calculation for the Relative Addressing mode.

The specified value a is interpreted as a relative displacement from page zero, byte zero. Therefore, displacement may be specified from -64 to +63 bytes. The address calculation is modulo 8192_{10}, so the negative displacement will develop addresses at the end of page zero. For example, ZBSR -10, will develop an effective address of 8182_{10}, and ZBSR 31 will develop an effective address of 31_{10}.

This instruction causes the processor to clear the page address bits, address bits 14 and 13, and may be executed anywhere within addressable memory.

Indirect addressing may be specified.

When executed, this instruction causes the Stack Pointer to be incremented by one, the address of the byte following this instruction is pushed into the Return Address Stack (RAS), and control is transferred to the effective address.

Processor Registers Affected	SP
Condition Code Setting	N/A

BRANCH TO SUBROUTINE ON CONDITION TRUE, RELATIVE

(Relative Addressing)

Mnemonic BSTR,v (*)a

Binary Code

0	0	1	1	1	0	v		I			a				
7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0

Execution Time 3 cycles (9 clock periods)

Description

This two - byte conditional subroutine branch instruction causes the processor to perform a subroutine branch *only* if the two-bit v field matches the current Condition Code field (CC) in the Program Status Word. If the fields match, the Stack Pointer is incremented by one and the current contents of the Instruction Address Register, which points to the byte following this instruction, is pushed into the Return Address Stack. The effective address replaces the previous contents of the IAR.

If the v field and CC field do not match, the next instruction is fetched from the location following the second byte of this instruction and the SP is unaffected.

Indirect addressing may be specified.

If v is set to 3_{16}, the BSTR instruction branches unconditionally.

Processor Registers Affected	SP
Condition Code Setting	N/A

BRANCH TO SUBROUTINE ON CONDITION TRUE, ABSOLUTE

(Absolute Addressing)

Mnemonic BSTA,v (*)a

Binary Code

0	0	1	1	1	1	v	
7	6	5	4	3	2	1	0

I	a high order						
7	6	5	4	3	2	1	0

a low order							
7	6	5	4	3	2	1	0

Execution Time 3 cycles (9 clock periods)

Description

This three-byte conditional subroutine branch instruction causes the processor to perform a subroutine branch only if the two-bit v field matches the current Condition Code Field (CC) in the Program Status Word. If the fields match, the Stack Pointer is incremented by one and the current contents of the Instruction Address Register, which points to the byte following this instruction is pushed into the Return Address Stack. The effective address replaces the previous contents of the IAR.

If the v field and the CC field do not match, the next instruction is fetched from the location following the third byte of this instruction and the Stack Pointer is unaffected.

Indirect addressing may be specified.

If v is set to 3_{16}, the BSTA instruction branches unconditionally.

Processor Registers Affected	SP
Condition Code Setting	N/A

BRANCH TO SUBROUTINE ON CONDITION FALSE, RELATIVE

(Relative Addressing)

Mnemonic BSFR,v (*)a

Binary Code

1	0	1	1	1	0	v	
7	6	5	4	3	2	1	0

I	a						
7	6	5	4	3	2	1	0

Execution Time 3 cycles (9 clock periods)

Description

This two-byte conditional subroutine branch instruction causes the processor to perform a subroutine branch *only* if the two-bit v field does *not* match the current Condition Code field (CC) in the Program Status Word. If the fields do not match, the Stack Pointer is incremented by one and the current content of the Instruction Address Register, which points to the location following this instruction, is pushed into the Return Address Stack. The effective address replaces the previous contents of the IAR.

If the v field and the CC match, the next instruction is fetched from the location following this instruction and the SP is unaffected.

Indirect addressing may be specified.

The v field may not be coded as 3_{16} because this combination is used for the ZBSR operation code.

Processor Registers Affected SP

Condition Code Setting N/A

BRANCH TO SUBROUTINE ON CONDITION FALSE, ABSOLUTE

(Absolute Addressing)

Mnemonic BSFA,v (*)a

Binary Code

7	6	5	4	3	2	1	0
1	0	1	1	1	1	v	

7	6	5	4	3	2	1	0
I	a high order						

7	6	5	4	3	2	1	0
a low order							

Execution Time 3 cycles (9 clock periods)

Description

This three-byte conditional subroutine branch instruction causes the processor to perform a subroutine branch *only* if the two-bit v field does *not* match the current Condition Code (CC) in the Program Status Word. If the fields do not match, the Stack Pointer is incremented by one and the current content of the Instruction Address Register, which points to the location following this instruction, is pushed into the Return Address Stack. The effective address replaces the previous contents of the IAR.

If the v field and the CC match, the next instruction is fetched from the location following this instruction and the SP is unaffected.

Indirect addressing may be specified.

The v field may not be coded as 3_{16} as this combination is used for the BSXA operation code.

Processor Registers Affected SP

Condition Code Setting N/A

BRANCH TO SUBROUTINE ON NON-ZERO REGISTER, RELATIVE

(Relative Addressing)

Mnemonic BSNR,r (*)a

Binary Code

0	1	1	1	1	0	r		I	a						
7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0

Execution Time 3 cycles (9 clock periods)

Description

This two-byte subroutine branch instruction causes the contents of the specified register r to be tested for a non-zero value. If the register contains a non-zero value, the next instruction to be executed is taken from the location pointed to by the effective address. Before replacing the contents of the Instruction Address Register with the effective address, the Stack Pointer (SP) is incremented by one and the address of the byte following the instruction is pushed into the Return Address Stack (RAS).

If the specified register contains a zero value, the next instruction is fetched from the location following this instruction.

Indirect addressing may be specified.

Processor Registers Affected	SP
Condition Code Setting	N/A

BRANCH TO SUBROUTINE ON NON-ZERO REGISTER, ABSOLUTE

(Absolute Addressing)

Mnemonic BSNA,r (*)a

Binary Code

0	1	1	1	1	1	r	
7	6	5	4	3	2	1	0

I	a high order						
7	6	5	4	3	2	1	0

a low order							
7	6	5	4	3	2	1	0

Execution Time 3 cycles (9 clock periods)

Description

This three-byte subroutine branch instruction causes the contents of the specified register r to be tested for a non-zero value. If the register contains a non-zero value, the next instruction to be executed is taken from the location pointed to by the effective address. Before replacing the current contents of the Instruction Address Register (IAR) with the effective address, the Stack Pointer (SP) is incremented by one and the address of the byte following the instruction is pushed into the Return Address Stack (RAS).

If the specified register contains a zero value, the next instruction is fetched from the location following this instruction.

Indirect addressing may be specified.

Processor Registers Affected SP

Condition Code Setting N/A

BRANCH TO SUBROUTINE INDEXED, ABSOLUTE, UNCONDITIONAL

(Absolute Addressing)

Mnemonic BSXA (*)a,X

Binary Code

1	0	1	1	1	1	1	1	I	a high order							a low order							
7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0

Execution Time 3 cycles (9 clock periods)

Description

This three-byte instruction causes the processor to perform an unconditional subroutine branch. Indexing is required and register #3 must be specified as the index register because the entire first byte of this instruction is decoded by the processor.

Execution of this instruction causes the Stack Pointer (SP) to be incremented by one, the address of the byte following this instruction is pushed into the Return Address Stack (RAS), and the effective address replaces the contents of the Instruction Address Register.

If indirect addressing is specified, the value in the index register is added to the indirect address to calculate the effective address.

Processor Registers Affected SP

Condition Code Setting N/A

RETURN FROM SUBROUTINE, CONDITIONAL

Mnemonic RETC,v

Binary Code

0	0	0	1	0	1	v	
7	6	5	4	3	2	1	0

Execution Time 3 cycles (9 clock periods)

Description

This one-byte instruction is used by a subroutine to conditionally effect a return of control to the program which last issued a subroutine branch instruction.

If the two-bit v field in the instruction matches the Condition Code field (CC) in the Program Status Word, the following action is taken: The address contained in the top of the Return Address Stack replaces the previous contents of the Instruction Address Register (IAR), and the Stack Pointer is decremented by one.

If the v field does not match CC, the return is not effected and the next instruction to be executed is taken from the location following this instruction.

If v is specified as 3_{16}, the return is executed unconditionally.

Processor Registers Affected	SP
Condition Code Setting	N/A

RETURN FROM SUBROUTINE AND ENABLE INTERRUPT, CONDITIONAL

Mnemonic RETE,v

Binary Code

0	0	1	1	0	1	v	
7	6	5	4	3	2	1	0

Execution Time 3 cycles (9 clock periods)

Description

This one-byte instruction is used by a subroutine to conditionally effect a return of control to the program which last issued a subroutine branch instruction. Additionally, if the return is effected, the Interrupt Inhibit (II) bit in the Program Status Word is cleared to zero, thus enabling interrupts. This instruction is mainly intended to be used by an interrupt handling routine because receipt of an interrupt causes a subroutine branch to be effected and the Interrupt Inhibit bit to be set to 1. The interrupt handling routine must be able to return and enable simultaneously so that the interrupt routine cannot be interrupt unless that is specifically desired.

If the two-bit v field in the instruction matches the Condition Code field (CC) in the Program Status Word, the following action is taken: The address contained in the top of the Return Address Stack (RAS) replaces the previous contents of the Instruction Address Register (IAR), the Stack Pointer is decremented by one and the II bit is cleared to zero.

If the v field does not match CC, the return is not effected and the next instruction to be executed is taken from the location following this instruction.

If v is specified as 3_{16}, the return is executed unconditionally.

Processor Registers Affected	SP , II
Condition Code Setting	N/A

READ DATA

(Register Addressing)

Mnemonic REDD,r

Binary Code

0	1	1	1	0	0	r	
7	6	5	4	3	2	1	0

Execution Time 2 cycles (6 clock periods)

Description

This one-byte input instruction causes a byte of data to be transferred from the data bus into register r. Signals on the data bus are considered to be true signals, i.e., a high level will be set into the register as a one.

When executing this instruction, the processor raises the Operation Request (OPREQ) line, simultaneously switching the M/$\overline{\text{IO}}$ line to $\overline{\text{IO}}$ and the $\overline{\text{R}}$/W to $\overline{\text{R}}$ (Read). Also, during the OPREQ signal, the D/$\overline{\text{C}}$ line switches to D (Data) and the E/$\overline{\text{NE}}$ switches to $\overline{\text{NE}}$ (Non-extended).

See Input/Output section of this manual.

Processor Registers Affected CC

Condition Code Setting

Register r	CC1	CC0
Positive	0	1
Zero	0	0
Negative	1	0

READ CONTROL

(Register Addressing)

Mnemonic REDC,r

Binary Code

7	6	5	4	3	2	1	0
0	0	1	1	0	0	r	

Execution Time 2 cycles (6 clock periods)

Description

This one-byte input instruction causes a byte of data to be transferred from the data bus into register r. Signals on the data bus are considered to be true signals, i.e., a high level will be set into the register as a one.

When executing this instruction, the processor raises the Operation Request (OPREQ) line, simultaneously switching the M/$\overline{\text{IO}}$ line to $\overline{\text{IO}}$, the $\overline{\text{R}}$/W line to $\overline{\text{R}}$ (Read), the D/$\overline{\text{C}}$ line to $\overline{\text{C}}$ (Control), and the E/$\overline{\text{NE}}$ line to $\overline{\text{NE}}$ (Non-extended).

See Input/Output section of this manual.

Processor Registers Affected CC

Condition Code Setting

Register r	CC1	CC0
Positive	0	1
Zero	0	0
Negative	1	0

READ EXTENDED (Immediate Addressing)

Mnemonic REDE,r v

Binary Code

7	6	5	4	3	2	1	0		7	6	5	4	3	2	1	0
0	1	0	1	0	1	r			v							

Execution Time 3 cycles (9 clock periods)

Description

This two-byte input instruction causes a byte of data to be transferred from the data bus into register r. During the execution of this instruction, the content of the second byte of this instruction is made available on the address bus. Signals on the data bus are true signals, i.e., a high level is interpreted as a one.

During execution, the processor raises the Operation Request (OPREQ) line, simultaneously placing the contents of the second byte of the instruction on the address bus. During the OPREQ signal, the M/$\overline{\text{IO}}$ line is switched to $\overline{\text{IO}}$, the $\overline{\text{R}}$/W line to $\overline{\text{R}}$ (Read), line and the E/$\overline{\text{NE}}$ line to E (Extended).

See Input/Output section of this manual.

Processor Registers Affected CC

Condition Code Setting

Register r	CC1	CC0
Positive	0	1
Zero	0	0
Negative	1	0

WRITE DATA

(Register Addressing)

Mnemonic WRTD,r

Binary Code

1	1	1	1	0	0	r	
7	6	5	4	3	2	1	0

Execution Time 2 cycles (6 clock periods)

Description

This one-byte output instruction causes a byte of data to be made available to an external device. The byte to be output is taken from register r and made available on the data bus. Signals on the data bus are true signals, i.e., high levels are ones.

When executing this instruction, the processor raises the Operation Request (OPREQ) line and simultaneously places the data on the Data Bus. Along with the OPREQ, the M/$\overline{\text{IO}}$ line is switched to $\overline{\text{IO}}$, the $\overline{\text{R}}$/W signal is switched to W (Write), and a Write Pulse (WRP) is generated. Also, during the valid OPREQ signals, the D/$\overline{\text{C}}$ line is switched to D (Data) and the E/$\overline{\text{NE}}$ line is switched to $\overline{\text{NE}}$ (Non-extended).

See Input/Output section of this manual.

Processor Registers Affected	None
Condition Code Setting	N/A

WRITE CONTROL

(Register Addressing)

Mnemonic WRTC,r

Binary Code

1	0	1	1	0	0	r	
7	6	5	4	3	2	1	0

Execution Time 2 cycles (6 clock periods)

Description

This one-byte output instruction causes a byte of data to be made available to an external device.

The byte to be output is taken from register r and made available on the data bus. Signals on the data bus are true signals, i.e., high levels are ones

When executing this instruction, the processor raises the Operation Request (OPREQ) line and simultaneously places the data on the Data Bus. Along with the OPREQ signal, the M/$\overline{\text{IO}}$ line is switched to $\overline{\text{IO}}$, the $\overline{\text{R}}$/W signal is switched to W (Write), the D/$\overline{\text{C}}$ line is switched to $\overline{\text{C}}$ (Control), the E/$\overline{\text{NE}}$ is switched to $\overline{\text{NE}}$ (Non-extended), and a Write Pulse (WRP) is generated.

See the Input/Output section of this manual.

Processor Registers Affected	None
Condition Code Setting	N/A

WRITE EXTENDED

(Immediate Addressing)

Mnemonic WRTE,r v

Binary Code

1	1	0	1	0	1	r	
7	6	5	4	3	2	1	0

			v				
7	6	5	4	3	2	1	0

Execution Time 3 cycles (9 clock periods)

Description

This two-byte output instruction causes a byte of data to be made available to an external device. The byte to be output is taken from register r and is made available on the data bus. Simultaneously, the data in the second byte of this instruction is made available on the address bus. The second byte, v, may be interpreted as a device address.

Signals on the busses are true levels, i.e., high levels are ones.

When executing this instruction, the processor raises the Operation Request (OPREQ) line and simultaneously places the data from register r on the data bus and the data from the second byte of this instruction on the address bus. Along with OPREQ, the M/$\overline{\text{IO}}$ line is switched to $\overline{\text{IO}}$, the $\overline{\text{R}}$/W line is switched to W (Write), the E/$\overline{\text{NE}}$ line is switched to E (Extended), and a Write Pulse (WRP) is generated.

See the Input/Output section of this manual.

Processor Registers Affected	None
Condition Code Setting	N/A

NO OPERATION

Mnemonic NOP

Binary Code

1	1	0	0	0	0	0	0
7	6	5	4	3	2	1	0

Execution Time 2 cycles (6 clock periods)

Description

This one-byte instruction causes the processor to take no action upon decoding it. No registers are changed, but fetching and executing a NOP instruction requires two processor cycles.

Processor Registers Affected None

Condition Code Setting N/A

TEST UNDER MASK IMMEDIATE

(Immediate Addressing)

Mnemonic TMI,r v

Binary Code

7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0
1	1	1	1	0	1	r		v							

Execution Time 3 cycles (9 clock periods)

Description

This two-byte instruction tests individual bits in the specified register r to determine if they are set to binary one. During execution, each bit in the v field of the instruction is tested for a one, and if a particular bit in the v field contains a one, the corresponding bit in register r is tested for a one or zero. The condition code is set to reflect the result of the operation.

If a bit in the v field is zero, the corresponding bit in register r is not tested.

Processor Registers Affected CC

Condition Code Setting

	CC1	CC0
All of the selected bits are 1s	0	0
Not all of the selected bits are 1s	1	0

DECIMAL ADJUST REGISTER

(Register Addressing)

Mnemonic DAR,r

Binary Code

7	6	5	4	3	2	1	0
1	0	0	1	0	1	r	

Execution Time 3 cycles (9 clock periods)

Description

This one-byte instruction conditionally adds a decimal ten (two's complement negative six in a four-bit binary number system) to either the high order 4 bits and/or the low order 4 bits of the specified register r.

The truth table below indicates the logical operation performed. The operation proceeds based on the contents of the Carry (C) and Interdigit Carry (IDC) bits in the Program Status Word. The C and IDC remain unchanged by the execution of this instruction.

This instruction allows BCD sign magnitude arithmetic to be performed on packed digits by the following procedure.

BCD Addition:	1.	add 66_{16} to augend
	2.	perform addition of addend and augend
	3.	perform DAR instruction
BCD Subtraction:	1.	perform subtraction (2's complement of subtrahend is added to the minuend)
	2.	perform DAR instruction

Since this operation is on sign-magnitude numbers, it is necessary to establish the sign of the result prior to executing in order to properly control the definition of the subtrahend and minuend.

Carry	Interdigit Carry	Added to Register r
0	0	AA_{16}
0	1	$A0_{16}$
1	1	00_{16}
1	0	$0A_{16}$

Processor Registers Affected CC

Condition Code Setting

The Condition Code is set to a meaningless value.

HALT, ENTER WAIT STATE

Mnemonic HALT

Binary Code

0	1	0	0	0	0	0	0
7	6	5	4	3	2	1	0

Execution Time 2 cycles (6 clock periods)

Description

This one-byte instruction causes the processor to stop executing instructions and enter the WAIT state. The RUN/$\overline{\text{WAIT}}$ line is set to the WAIT state.

The only way to enter the RUN state after a HALT has been executed, is to reset the 2650 or to interrupt the processor.

Processor Registers Affected	None
Condition Code Setting	N/A

APPENDIX A

MEMORY INTERFACE

Figure A-1 shows a complete interface between the 2650 and a 256 x 8 R/W random access memory. Since the memory chips are MOS they can be driven directly by the address lines and the control lines. The gates shown are assumed to be standard 7400 series TTL so that some signal buffering is assumed to be necessary. If CMOS or 74LS gates are used, some of the buffering inverters may not be necessary. The same is true of the data bus. Depending on the number and nature of the I/O devices being interfaced, it may or may not be necessary to buffer the data bus.

Because the data in and data out signals for the memory chips are bussed together, care must be taken to avoid overlap of drivers on the data bus. In this example, the problem is solved by using the write pulse into the memory as the chip select input instead of using the $\overline{R}$/W line as is conventionally done. The $\overline{R}$/W output from the processor is a level and is valid when Operation Request is true. Write Pulse from the processor is gated with the OPREQ and M/$\overline{IO}$ signals to assure proper operation.

For a large memory the next address line (ADR8) could be gated into the chain that generates the chip select signals, with similar write pulse generation for the higher order memory.

The $\overline{OPACK}$ signal is assumed to be false for the duration of all memory operations. This eliminates some gating from that control input. No problems will be encountered with this approach as long as the memories are fast enough for the clock speed being used with the processor. At a cycle time of 2.4μs, data must be returned to the processor by 1μs or less time from the OPREQ leading edge.

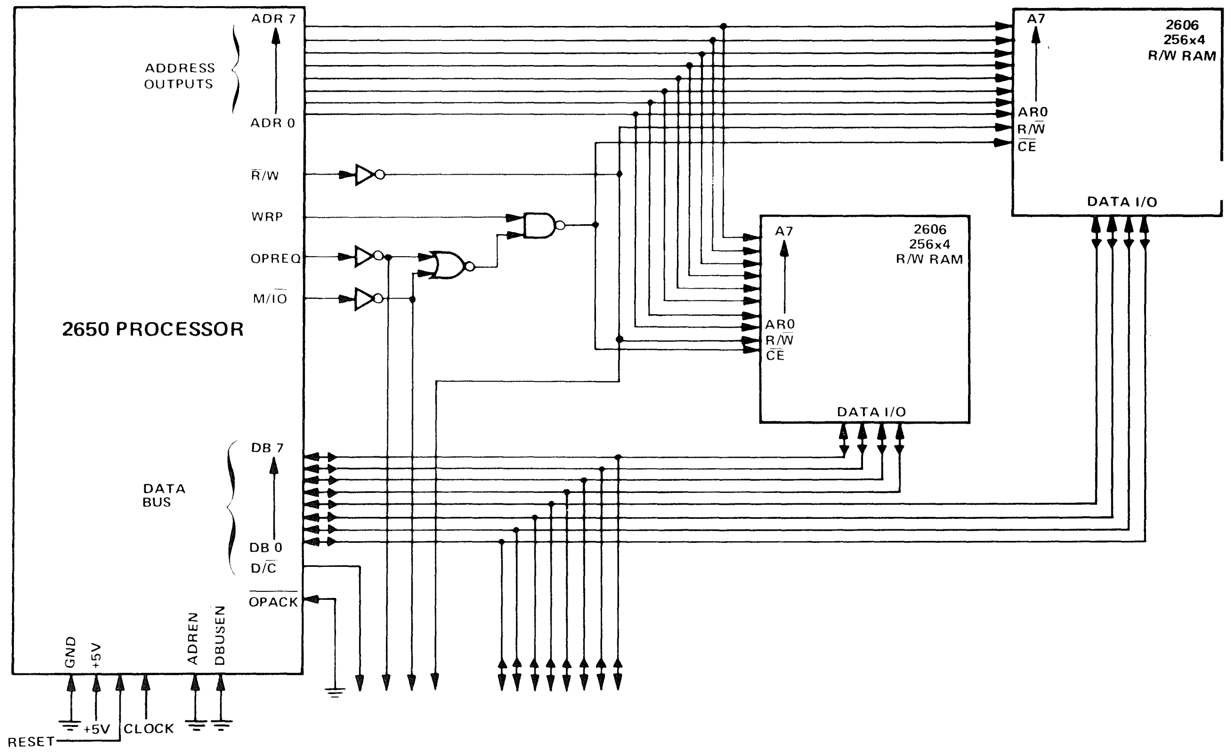


Figure A-1

APPENDIX B

I/O INTERFACE

Figure B-1 shows one of many possible methods for buffering the data bus and interfacing it to several devices. There are advantages to be gained by using the Signetics 8T26. It has a PNP input buffer that keeps its low input level current at 200μA instead of 1.6mA. This lightens the load on the processor bus drivers and allows the processor to interface to several 8T26's if necessary. The 8T26 has four complete driver/receiver pairs in a package, so two packages can fully buffer the 8-bit data bus.

The control signals generated for use with I/O interfaces are very straightforward. Combining M/$\overline{\text{IO}}$ with OPREQ generates a signal that can often be used conveniently at the I/O devices instead of having each device derive the signal individually. In the figure it is gated with the Read/Write information in order to control the bus buffer.

Each I/O device must handle four basic processor interface functions:

(a) bus interface
(b) data transfer logic
(c) device selection logic
(d) transfer acknowledge logic

Depending on the nature of the complete system and the particular I/O device, these functions can be either extremely simple or fairly complex.

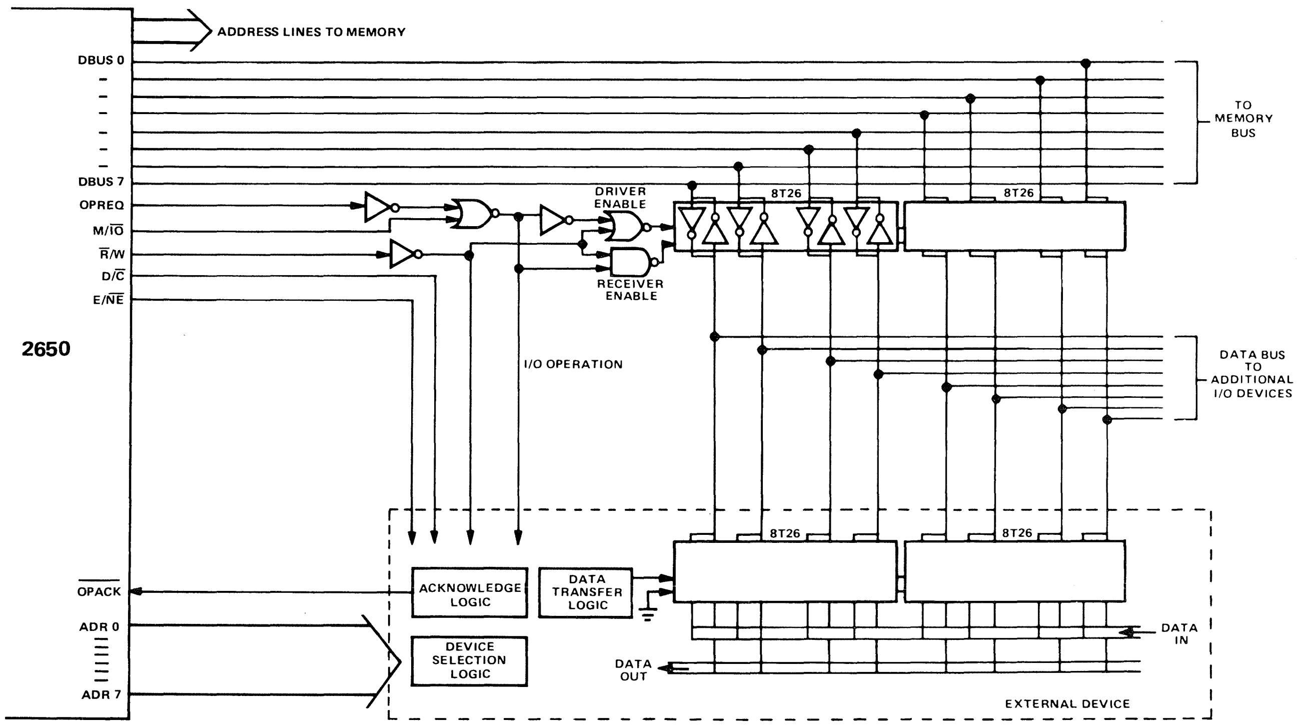


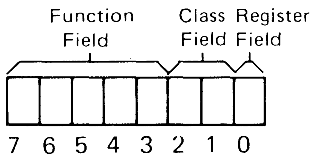
Figure B-1

APPENDIX C

INSTRUCTIONS, ADDITIONAL INFORMATION

The 2650 uses variable length instructions that are one, two or three bytes long. The instruction length is determined by the nature of the operation being performed and the addressing mode being used. Thus, the instruction can be expressed in one byte when no memory operand addressing is necessary, as with register-to-register or rotate instructions. On the other hand, for direct addressing instructions, three bytes are allocated. The relative and immediate addressing modes allow two-byte instructions to be implemented.

The 2650 uses explicit operand addressing; that is, each instruction specifies the operand address. The first byte of each 2650 instruction is divided into three fields and specifies the operation to be performed, the addressing mode to be used and, where appropriate, the register or condition code mask to be used.



The CLASS field specifies the instruction group, the major address mode and the number of processor cycles required for each instruction. The CLASS field also specifies, with one exception, the number of bytes in the instruction. The following table shows the specifications for each class.

CLASS FIELD	INSTRUCTION GROUP	ADDRESS REGISTER	BYTE LENGTH	DIRECT CYCLES
0	Arithmetic	Register	1	2
1	Arithmetic	Immediate	2	2
2	Arithmetic	Relative	2	3
3	Arithmetic	Absolute	3	4
4	Control (inc. rotate)		1	2
5	Control		1-2	3
6	Branch	Relative	2	3
7	Branch	Absolute	3	3

Within the arithmetic groups (classes 0, 1, 2, and 3) the function field specifies one of the eight operations as follows:

FUNCTION FIELD	ARITHMETIC OPERATION
0	LOAD
1	EXCLUSIVE OR
2	AND
3	INCLUSIVE OR
4	ADD
5	SUBTRACT
6	STORE
7	COMPARE

Within the branch group (classes 6 and 7) the function field specifies one of eight operations as follows:

FUNCTION FIELD	BRANCH OPERATION
0	Branch On Condition True
1	Branch To Subroutine On Condition True
2	Branch On Register Non-Zero
3	Branch To Subroutine On Register Non-Zero
4	Branch On Condition False
5	Branch To Subroutine On Condition False
6	Branch On Incrementing Register
7	Branch On Decrementing Register

There is very little pattern to the use of the function field within the control group (classes 4 and 5).

The register field is used to specify the index register, to specify the operand source register, to specify the destination register, or a condition code mask. For the register-to-register and the indexed instructions, register zero is implicitly assumed to be the source or the destination of the instruction. For all other instructions that involve a register, the register field allows any of four registers to be specified, except for indexed branch instructions which require that register 3 be specified.

Conditional branch instructions utilize the 2-bit register field as a condition code mask field. A few instructions use the register field as part of the operation code and consequently allow no variation in register usage.

APPENDIX D

INSTRUCTION SUMMARY

SIGNETICS 2650 PROCESSOR

ALPHABETIC LISTING

HEX	OP	Pg.
8C	ADDA	58
8D		
8E		
8F		
84	ADDI	56
85		
86		
87		
88	ADDR	57
89		
8A		
8B		
80	ADDZ	55
81		
82		
83		
4C	ANDA	66
4D		
4E		
4F		
44	ANDI	64
45		
46		
47		
48	ANDR	65
49		
4A		
4B		
41	ANDZ	63
42		
43		
9C	BCFA	95
9D		
9E		
98	BCFR	94
99		
9A		
1C	BCTA	93
1D		
1E		
1F		
18	BCTR	92
19		
1A		
1B		
FC	BDRA	99
FD		
FE		
FF		
F8	BDRR	98
F9		
FA		
FB		
DC	BIRA	97
DD		
DE		
DF		
D8	BIRR	96
D9		
DA		
DB		
5C	BRNA	101
5D		
5E		
5F		
58	BRNR	100
59		
5A		
5B		
BC	BSFA	107
BD		
BE		
B8	BSFR	106
B9		
BA		
7C	BSNA	109
7D		
7E		
7F		
78	BSNR	108
79		
7A		
7B		
3C	BSTA	105
3D		
3E		
3F		
38	BSTR	104
39		
3A		
3B		
BF	BSXA	110
9F	BXA	102
EC	CØMA	78
ED		
EE		
EF		

HEX	OP	Pg.
E4	CØMI	76
E5		
E6		
E7		
E8	CØMR	77
E9		
EA		
EB		
E0	CØMZ	75
E1		
E2		
E3		
75	CPSL	88
74	CPSU	87
94	DAR	121
95		
96		
97		
2C	EØRA	74
2D		
2E		
2F		
24	EØRI	72
25		
26		
27		
28	EØRR	73
29		
2A		
2B		
20	EØRZ	71
21		
22		
23		

HEX	OP	Pg.
40	HALT	122
6C	IØRA	70
6D		
6E		
6F		
64	IØRI	68
65		
66		
67		
68	IØRR	69
69		
6A		
6B		
60	IØRZ	67
61		
62		
63		
0C	LØDA	51
0D		
0E		
0F		
04	LØDI	49
05		
06		
07		
08	LØDR	50
09		
0A		
0B		
00	LØDZ	48
01		
02		
03		

HEX	OP	Pg.
93	LPSL	82
92	LPSU	81
C0	NØP	119
77	PPSL	86
76	PPSU	85
30	REDC	114
31		
32		
33		
70	REDD	113
71		
72		
73		
54	REDE	115
55		
56		
57		
14	RETC	111
15		
16		
17		
34	RETE	112
35		
36		
37		
D0	RRL	79
D1		
D2		
D3		

HEX	OP	Pg.
50	RRR	80
51		
52		
53		
13	SPSL	84
12	SPSU	83
CC	STRA	54
CD		
CE		
CF		
C8	STRR	53
C9		
CA		
CB		
C1	STRZ	52
C2		
C3		
AC	SUBA	62
AD		
AE		
AF		
A4	SUBI	60
A5		
A6		
A7		
A8	SUBR	61
A9		
AA		
AB		
A0	SUBZ	59
A1		
A2		
A3		

HEX	OP	Pg.
F4	TMI	120
F5		
F6		
F7		
B5	TPSL	90
B4	TPSU	89
B0	WRTC	117
B1		
B2		
B3		
F0	WRTD	116
F1		
F2		
F3		
D4	WRTE	118
D5		
D6		
D7		
9B	ZBRR	91
BB	ZBSR	103

SIGNETICS 2650 PROCESSOR

NUMERIC LISTING

HEX	OP	Pg.
00	LØDZ	48
01		
02		
03		
04	LØDI	49
05		
06		
07		
08	LØDR	50
09		
0A		
0B		
0C	LØDA	51
0D		
0E		
0F		
12	SPSU	83
13	SPSL	84
14	RETC	111
15		
16		
17		
18	BCTR	92
19		
1A		
1B		
1C	BCTA	93
1D		
1E		
1F		
20	EØRZ	71
21		
22		
23		
24	EØRI	72
25		
26		
27		
28	EØRR	73
29		
2A		
2B		
2C	EØRA	74
2D		
2E		
2F		
30	REDC	114
31		
32		
33		
34	RETE	112
35		
36		
37		
38	BSTR	104
39		
3A		
3B		
3C	BSTA	105
3D		
3E		
3F		
40	HALT	122
41	ANDZ	63
42		
43		
44	ANDI	64
45		
46		
47		
48	ANDR	65
49		
4A		
4B		
4C	ANDA	66
4D		
4E		
4F		
50	RRR	80
51		
52		
53		
54	REDE	115
55		
56		
57		
58	BRNR	100
59		
5A		
5B		
5C	BRNA	101
5D		
5E		
5F		
60	IØRZ	67
61		
62		
63		
64	IØRI	68
65		
66		
67		

HEX	OP	Pg.
68	IØRR	69
69		
6A		
6B		
6C	IØRA	70
6D		
6E		
6F		
70	REDD	113
71		
72		
73		
74	CPSU	87
75	CPSL	88
76	PPSU	85
77	PPSL	86
78	BSNR	108
79		
7A		
7B		
7C	BSNA	109
7D		
7E		
7F		
80	ADDZ	55
81		
82		
83		
84	ADDI	56
85		
86		
87		
88	ADDR	57
89		
8A		
8B		
8C	ADDA	58
8D		
8E		
8F		
92	LPSU	81
93	LPSL	82
94	DAR	121
95		
96		
97		
98	BCFR	94
99		
9A		
9B	ZBRR	91
9C	BCFA	95
9D		
9E		
9F	BXA	102
A0	SUBZ	59
A1		
A2		
A3		
A4	SUBI	60
A5		
A6		
A7		
A8	SUBR	61
A9		
AA		
AB		
AC	SUBA	62
AD		
AE		
AF		
B0	WRTC	117
B1		
B2		
B3		
B4	TPSU	89
B5	TPSL	90
B8	BSFR	106
B9		
BA		
BB	ZBSR	103
BC	BSFA	107
BD		
BE		
BF	BSXA	110

HEX	OP	Pg.
C0	NØP	119
C1	STRZ	52
C2		
C3		
C8	STRR	53
C9		
CA		
CB		
CC	STRA	54
CD		
CE		
CF		
D0	RRL	79
D1		
D2		
D3		
D4	WRTE	118
D5		
D6		
D7		
D8	BIRR	96
D9		
DA		
DB		
DC	BIRA	97
DD		
DE		
DF		
E0	CØMZ	75
E1		
E2		
E3		

HEX	OP	Pg.
E4	CØMI	76
E5		
E6		
E7		
E8	CØMR	77
E9		
EA		
EB		
EC	CØMA	78
ED		
EE		
EF		
F0	WRTD	116
F1		
F2		
F3		
F4	TMI	120
F5		
F6		
F7		
F8	BDRR	98
F9		
FA		
FB		
FC	BDRA	99
FD		
FE		
FF		

2650 INSTRUCTIONS

ORGANIZED BY FUNCTION

LOAD/STORE		Pg.	ARITHMETIC		Pg.	ARITHMETIC		Pg.
00	LØDZ	48	80	ADDZ	55	68	IØRR	69
01			81			69		
02			82			6A		
03			83			6B		
04	LØDI	49	84	ADDI	56	6C	IØRA	70
05			85			6D		
06			86			6E		
07			87			6F		
08	LØDR	50	88	ADDR	57	20	EØRZ	71
09			89			21		
0A			8A			22		
0B			8B			23		
0C	LØDA	51	8C	ADDA	58	24	EØRI	72
0D			8D			25		
0E			8E			26		
0F			8F			27		
C1	STRZ	52	A0	SUBZ	59	28	EØRR	73
C2			A1			29		
C3			A2			2A		
			A3			2B		
C8	STRR	53	A4	SUBI	60	2C	EØRA	74
C9			A5			2D		
CA			A6			2E		
CB			A7			2F		
CC	STRA	54	A8	SUBR	61	41	ANDZ	63
CD			A9			42		
CE			AA			43		
CF			AB					
			AC	SUBA	62	44	ANDI	64
			AD			45		
			AE			46		
			AF			47		
			60	IØRZ	67	48	ANDR	65
			61			49		
			62			4A		
			63			4B		
			64	IØRI	68	4C	ANDA	66
			65			4D		
			66			4E		
						4F		

BRANCH		Pg.
18	BCTR	92
19		
1A		
1B		
1C	BCTA	93
1D		
1E		
1F		
98	BCFR	94
99		
9A		
9C	BCFA	95
9D		
9E		
58	BRNR	100
59		
5A		
5B		
5C	BRNA	101
5D		
5E		
5F		
D8	BIRR	96
D9		
DA		
DB		
DC	BIRA	97
DD		
DE		
DF		
F8	BDRR	98
F9		
FA		
FB		
FC	BDRA	99
FD		
FE		
FF		
9F	BXA	102
9B	ZBRR	91

SUBROUTINE BRANCH		Pg.
38	BSTR	104
39		
3A		
3B		
3C	BSTA	105
3D		
3E		
3F		
B8	BSFR	106
B9		
BA		
BC	BSFA	107
BD		
BE		
78	BSNR	108
79		
7A		
7B		
7C	BSNA	109
7D		
7E		
7F		
BF	BSXA	110
BB	ZBSR	103

SUBROUTINE RETURN		
14	RETC	111
15		
16		
17		
34	RETE	112
35		
36		
37		

COMPARE		Pg.
E0	CØMZ	75
E1		
E2		
E3		
E4	CØMI	76
E5		
E6		
E7		
E8	CØMR	77
E9		
EA		
EB		
EC	CØMA	78
ED		
EE		
EF		

INPUT/OUTPUT		
30	REDC	114
31		
32		
33		
70	REDD	113
71		
72		
73		
B0	WRTC	117
B1		
B2		
B3		
F0	WRTD	116
F1		
F2		
F3		
54	REDE	115
55		
56		
57		
D4	WRTE	118
D5		
D6		
D7		

PROGRAM STATUS MANIPULATION

		Pg.
92	LPSU	81
93	LPSL	82
12	SPSU	83
13	SPSL	84
74	CPSU	87
75	CPSL	88
76	PPSU	85
77	PPSL	86
B4	TPSU	89
B5	TPSL	90

ROTATE INSTRUCTIONS

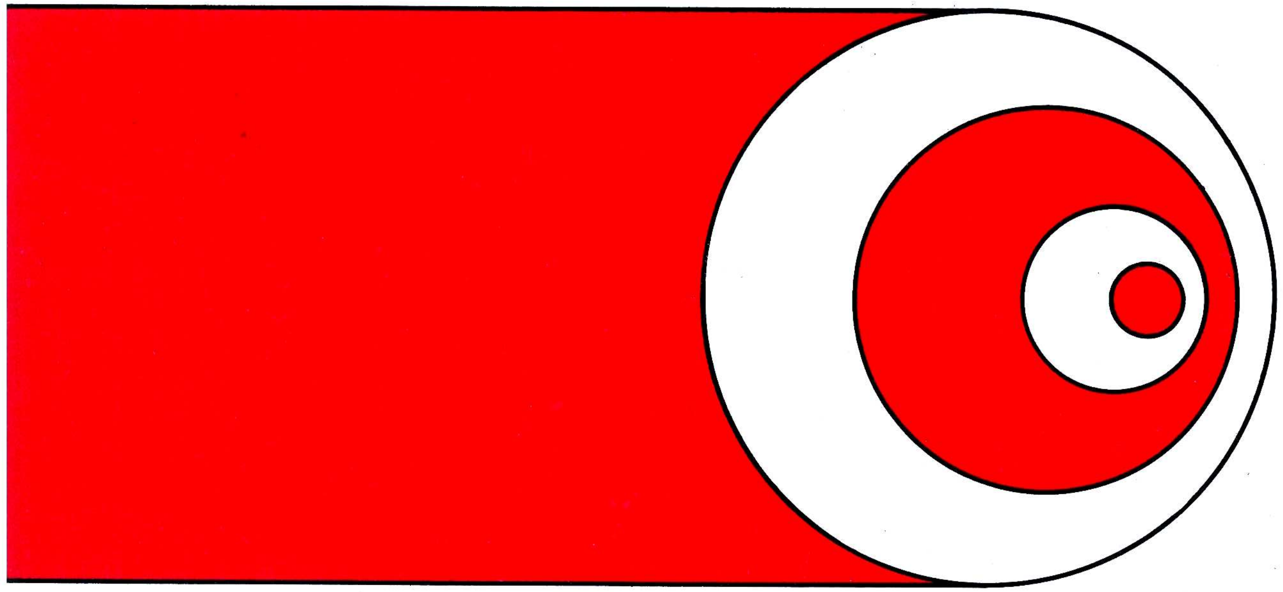
D0 D1 D2 D3	RRL	79
50 51 52 53	RRR	80

MISCELLANEOUS

		Pg.
C0	NØP	119
40	HALT	122
F4 F5 F6 F7	TMI	120
94 95 96 97	DAR	121

SIGNETICS
2650 MICROPROCESSOR

ASSEMBLER LANGUAGE



2650
ASSEMBLER LANGUAGE MANUAL

CONTENTS

I INTRODUCTION

The assembly language described in this document is a symbolic language designed specifically to facilitate the writing of programs for the Signetics 2650 processor. The 2650 Assembler is a program which accepts symbolic source code as input and produces a listing and/or an object module as output.

The assembler is written in standard FORTRAN IV and is available either through a timesharing service or in batch form directly from Signetics. This is done to assure compatibility and ease of installation on a user's own computer equipment. It is modular and may be executed in an overlay mode should memory restrictions make that necessary. The program is approximately 1,250 FORTRAN card images in length.

An attempt was made in the design of the language to make it similar to other contemporary assembler languages because it was felt that such similarity would reduce the learning time necessary to become proficient in this language. The 2650 assembler features forward references, self-defining constants, free format source code, symbolic addressing, syntax error checking, load module generation, and source statement listing.

In order to understand the 2650 instruction set, architecture, timing, interface requirements and electrical characteristics, the reader is referred to the Signetics 2650 Hardware Specification section.

The assembler is a two pass program that builds a symbol table, issues helpful error messages, produces an easily readable program listing and outputs a computer readable object (load) module.

The assembler features symbolic and relative addressing, forward references, complex expression evaluations and a versatile set of Pseudo-Operations. These features aid the programmer/engineer in producing well-documented, working programs in a minimum of time. Additionally, the assembler is capable of generating data in several number based systems as well as both ASCII and EBCDIC character codes.

Assembler Language

The assembler language provides a means to create a computer program. The features of the Assembler are designed to meet the following goals:

- Programs should be easy to create
- Programs should be easy to modify
- Programs should be easy to read and understand
- A machine readable, machine language module to be output

This assembler language has been developed with the following features:

- Symbolic machine operation codes (op-codes, mnemonics)
- Symbolic address assignment and references
- Relative addressing
- Data creation statements
- Storage reservation statements
- Assembly listing control statements
- Addresses can be generated as constants
- Character codes may be specified as ASCII or EBCDIC
- Comments and remarks may be encoded for documentation

As Assembly language program is a program written in symbolic machine language. It is comprised of statements. A statement is either a symbolic machine instruction, a pseudo-operation statement, or a comment.

The symbolic machine instruction is a written specification for a particular machine operation expressed by symbolic operation codes and sometimes symbolic addresses or operands. For example:

```
LOC2          STRR, R0      SAV
```

Where:

LOC2	is a symbol which will represent the memory address of the instruction.
STRR	is a symbolic op-code which represents the bit pattern of the "store relative" instruction.
R0	is a symbol which has been defined as register 0 by the "EQU pseudo-op".
SAV	is a symbol which represents the memory location into which the contents of register 0 are to be stored.

A pseudo-operation statement is a statement which is not translated into a machine instruction, but rather is interpreted as a directive to the assembler program. Example:

```
SCHD          ACON          REDY
```

Where:

ACON	is a pseudo-op which directs the assembler program to allocate two bytes of memory.
REDY	is a symbol, representing an address. The assembler is directed to place the equivalent memory address into the byte allocated space.
SCHD	is a symbol. The assembler is to assign the memory address of the first byte of the two allocated to this symbol.

Statements

Statements are always written in a particular format. The format is depicted below:

```
LABEL FIELD  OPERATION FIELD  OPERAND FIELD  COMMENT FIELD
```

The statement is always assumed to be written on an 80 column data processing card or an 80 column card image.

The Label Field is provided to assign symbolic names to bytes of memory. If present, the Label Field must begin in logical column one.

The Operation Field is provided to specify a symbolic operation code or a pseudo-operation code. If present, the Operation Field must either begin past column one or be separated from logical column one by one or more blanks.

The Operand Field is provided to specify arguments for the operation in the Operation Field. The Operand Field, if present, is separated from the Operation Field by one or more blanks.

The Comment Field is provided to enable the assembly language programmer to optionally place an English message stating the purpose or intent of a statement or a group of statements. The Comment Field must be separated from the preceding field by one or more blanks.

Comment Statement

A Comment Statement is a statement that is not processed by the assembler program. It is merely reproduced on the assembly listing. A Comment Statement is indicated by encoding an asterisk in logic column one. Example:

```
*THIS IS A COMMENT STATEMENT
```

Logical columns 72-80 are never processed by the assembler, they are always reproduced on the assembly listing without processing. This field is a good place for sequence numbers, if desired.

Symbolic Addressing

When writing statements in symbolic machine language, i.e., assembler language, the machine operation code is usually expressed symbolically. For example, the machine instruction that stores data from register 0 into a memory location named SAV, may be expressed as:

```
            STRA, R0     SAV
```

The assembler, when translating this symbolic operation code and its arguments into machine language for the 2650, defines three bytes containing H'CC0020', where '0020' is the value of SAV.

The address of the translated bytes is known because the Assembly Program Counter is always set to the address of the next byte to be assembled.

The user can attach a label to an instruction:

```
SAVR        STRR,R0      SAV
```

The assembler, upon seeing a valid symbol in the label field, assigns the equivalent address to the label. In the given example, if the STRR instruction is to be stored in the address H'0127', then the symbol SAVR would be made equivalent to the value H'0127' for the duration of the assembly.

The symbol could then be used anywhere in the source program to refer to the address value or, more typically, it could be used to refer to the instruction location. The important concept is that the address of the instruction need not be known; only the symbol need to be used to refer to the instruction location. Thus, when branching to the STRR instruction, one could write:

```
            BCTA,3       SAVR
```

When the three byte branch instruction is translated by the assembler, the address of the STRR instruction is placed in the address field of the branch instruction.

It is also possible to use symbolic addresses which are near other locations to refer to those locations without defining new labels. For example:

```
          BCTR,3    BEG
          BCTR,0    BEG+4
          ANDZ      3
          BSTR,3    S+48
BEG       LODA,2    PAL
          HALT
          SUBI,2    3
```

In the above example, the instruction "BCTR,3 BEG" refers to the LODA,2 PAL instruction. The instruction "BCTR,0 BEG+4" refers to the SUBI,2 3 instruction.

BEG+4 means the address BEG plus four bytes. This type of expression is called relative symbolic addressing and given a symbolic address; it can be used as a landmark to express several bytes before or after the symbolic address. Examples:

```
          BCTR,3    PAL+23
          BSTA,0    STT-18
```

The arguments are evaluated like any other expression and cannot exceed in value the maximum number that can be contained in a FORTRAN integer constant.

Program Counter

During the assembly process the assembler maintains a FORTRAN Integer cell that always contains the address of the next memory location to be assembled. This cell is called the Program Counter. It is used by the assembler to assign addresses to assembled bytes, but it is also available to the programmer.

The character "$" is the only valid symbol containing a special character that the assembler recognizes without error. "$" is the symbolic name of the Program Counter. It may be used like any other symbol, but it may not appear in the label field.

When using the "$", the programmer may think of it as expressing the idea "$" = "address of myself". For example,

108_{16} BCTR,3 $

This branch instruction is in location 108_{16}. The instruction directs the microprocessor to "branch to myself". The Program Counter in this example contains the value 108_{16}.

II LANGUAGE ELEMENTS

Input to the assembler consists of a sequence of characters combined to form assembly language elements. These language elements include symbols, instruction mnemonics, constants and expressions which make up the individual program statements that comprise a source program.

CHARACTERS

Alphabetic:	A through Z
Numeric:	0 through 9
Special characters:	blank
	(left parenthesis
	) right parenthesis
	+ add or positive value
	- subtract or negative value
	* asterisk
	' single quote
	, comma
	/ slash
	$ dollar sign
	< less than sign
	> greater than sign

SYMBOLS

Symbols are formed from combination of characters. Symbols provide a convenient means of identifying program elements so they can be referenced by other elements.

1. Symbols may consist of 1 to 4 alphanumeric characters: A through Z, 0 through 9.
2. Symbols must begin with an alphabetic character.
3. The character $ is a special symbol which may be used in the argument field of a statement to represent the current value of the Location Counter.
4. The character * is a special symbol which is used as an indirect address indicator.
5. The characters + and - are also used as auto-increment/auto-decrement indicators.

The following are examples of valid symbols:

```
DOP1        RAV3
AA          TEMZ
```

The following are examples of invalid symbols:

1LAR	begins with numeric
PA N	imbedded blank

CONSTANTS

A constant is a self-defining language element. Unlike a symbol, the value of a constant is its own "face" value and is invariant. Internal numbers are represented in 2's complement notation. There are two forms in which constants may be written: the Self-Defining Constant and the General Constant.

Self-Defining Constant

The self-defining constant is a form of constant which is written directly in an instruction and defines a decimal value. For example:

```
LODA,R3       BUFF+65
```

In this example, 65 is a self-defining constant. The maximum value of the integer constant expressed by a self-defining constant is that which, when expressed in binary, will fit within the basic arithmetic unit of the host computer (typically 1 word).

General Constant

The general constant is also written directly in an instruction, but the interpretation of its value is dictated by a code character and delimited by quotation marks.

```
LODA,R3       BUFF+H'3E'
```

In this example, the code letter H specifies that 3E is a hexadecimal constant equivalent to decimal value 62.

The maximum size of a number generated by a general constant form (B, O, D, H) may be no larger than the size of the FORTRAN integer cell of the host computer. However, the most important concept to understand when using constant forms is that the final value of a resolved expression must fit the constraints of the actual field destined to contain the value. For example:

```
LODA,R2       PAL+H'3EE2'-H'3EE0'
```

In this case, the argument, when resolved, must fit into the 13 bits in the actual machine instruction. Even though each of the two hexadecimal constants are larger than can fit into 13 bits, the final value of the expression is containable in 13 bits and therefore the constants are permitted. Similarly, the statement DATA H'3FE' is not allowed, as the DATA statement defines one byte quantities and H'3FE' specifies more than 8 bits. Summarily, the size of the evaluated expressions must be less than or equal to their corresponding data fields. There are 6 types of General Constants:

Code	Type
B	Binary Constant
O	Octal Constant
D	Decimal Constant
H	Hexadecimal Constant
E	EBCDIC Character Constant
A	ASCII Character Constant

B: Binary Constant

A binary constant consists of an optionally signed binary number of up to 8 bits enclosed in single quotes and preceded by the letter B, e.g., B'1011011'. Binary information is stored right justified.

O: Octal Constant

An octal constant consists of an optionally signed octal number enclosed by single quotation marks and preceded by the letter O, e.g., O'352'. The value will be right justified.

D: Decimal Constant

A decimal constant consists of an optionally signed decimal number enclosed by single quotation marks and preceded by the letter D, e.g., D'249'. The value will be right justified.

H: Hexadecimal Constant

A hexadecimal constant consists of an optionally signed hexadecimal number enclosed in single quotation marks and preceded by the letter H, e.g., H'3F'. The value will be right justified.

E: EBCDIC Character Constant

An EBCDIC character consists of a string of EBCDIC characters enclosed by single quotation marks and preceded by the letter E, e.g., E'ARE YOU THERE?'. Each character will be encoded in 8-bit EBCDIC and stored in successive bytes. The maximum number of characters which may be specified in one character string constant is 16.

A: ASCII Character Constant

An ASCII character constant consists of a string of ASCII characters enclosed by quotation marks and preceded by the letter A. For example: A'HELLO THERE'. Each character will be encoded in 7-bit ASCII and stored in successive bytes. The high order bit is always set to zero in each allocated byte. Up to 16 characters may be specified in one statement.

Note: See Appendix C for permissible characters and their equivalent ASCII and EBCDIC codes. To specify a single quotation mark as a character constant it must appear twice in the character string, e.g., A'TYPE' 'HELP' 'NOW' will appear in storage as TYPE'HELP'NOW.

MULTIPLE CONSTANT SPECIFICATIONS

General constant forms, except A and E, allow multiple specifications within the constant expression. For example: D'52, 21, 208, 27'. A comma separates each byte specification and successive specifications determine successive bytes of storage. Only 16 bytes of information may be specified in any one general constant form and each byte may be optionally signed. For example:

```
H'03,-F2,+11,-8,33,0'
O'271,133'.
```

EXPRESSIONS

An expression is an assembly language element that represents a value. It consists of a single term or a combination of terms separated by arithmetic operators. A term may be a valid symbolic reference, a self-defining constant or a general constant.

It is important to understand that although individual terms in a expression may exceed the number size restriction of the 2650 (one or two bytes), they may not cause the number size of the host computer's integer FORTRAN constant to be exceeded.

Examples of valid expressions:

```
LOOP              PAL-$
LOOP+5            $-PAL+3
SAM+3-LOOP        BIT-3+H'3A'
```

Note: The special symbol '$' represents the current value of the location counter.

SPECIAL OPERATORS

There are two special operators that are recognized by the assembler. They are:

< less than sign
> greater than sign

The assembler interprets these operators in a special way:

< perform a modulo 256 divide (use high order byte)
> perform a divide by 256 (use low order byte)

These operators, when used, must appear as the first character in the argument field. If they are imbedded in an expression, the results are unpredictable.

These special operators are intended to be used to access a two byte address in one byte parts using a minimum of storage. For example, if it is desired to get the high order bits of an address (ADDB) into register 2 and the low order bits into register 1 it could be done as follows:

```
             LODR,R2     APAL
             LODR,R1     APAL+1
             • • •
             • • •
             • • •
APAL         ACON        ADDB
```

or, by utilizing the special operators, it could be done as follows:

```
             LODI,R2     <ADDB
             LODI,R1     >ADDB
```

The first method uses 6 bytes to accomplish what the second method can do in 4 bytes.

The special operators care most often used to facilitate the passing of an address in registers.

III SYNTAX

Assembly language elements may be combined to symbolically express both 2650 instructions and assembler directives. There are specific rules for writing these instructions. This set of rules is known as the Syntax of the symbolic assembler language. The following description assumes a logical input of an 80-column data processing card, but since the host assembler is written in Fortran, the input media may be magnetic tape, magnetic disk, paper tape, etc. Only the format statement for input need be changed to accommodate the various input media.

FIELDS

A statement prepared for processing by the assembler is logically divided into four fields: the Name Field, the Operation Field, the Argument Field and the Comment Field. Each field is separated by at least one blank character. No continuation cards are allowed, and only logical columns 1 through 72 are scanned by the assembler. Logical columns 73 through 80 inclusive may be used for any desired purpose.

Name Field

The name (or label) field optionally contains a symbolic name which the assembler assigns to the instruction specified in the remaining part of the line. If a name is specified, it must begin in logical column 1. The assembler assumes that there is no name if logical column 1 is blank. The name field, if present, must contain only a valid symbol.

Operation Field

The operation field contains a mnemonic code which represents a 2650 processor operation or an assembly directive. The operation field must be present in every non-comment line. See Appendix A for a list of the valid mnemonic codes. Additionally, depending on the instruction type, the operation field may also specify a general purpose register or a condition code.

Argument Field

The argument field contains one or more symbols, constants or expressions separated by commas. The argument field specifies storage locations, constants, register specifications and any other information necessary to completely specify a machine operation or an assembler directive. Embedded blanks are not permitted as they are considered field terminators.

Comment Field

The comment field contains any valid characters in any combination. The comment field is not processed by the assembler, but is merely reproduced on the listing next to the accompanying instruction. It is usually used to explain the purpose or intention of a particular instruction or group of instructions.

Comment Card

An entire 72 column line may be utilized to print comments by coding an asterisk (*) in column 1. This entire card is merely reproduced on the assembly listing without processing by the assembler.

SYMBOLS

Symbols are used in the name field of a symbolic machine instruction to identify that particular instruction and to represent its address. Symbols may be used for other purposes, such as the symbolic representation of some memory address, the symbolic representation of a constant, the symbolic representation of a register, etc.

No matter how the symbol is used, it must be defined. A symbol is defined when the assembler knows what value the symbol represents. There is only one way to define a symbol. The symbol must at some time appear either in the name field of an instruction or of an assembler directive. The symbol will be assigned the current value of the Location Counter when it appears in the name field of a machine instruction, or it may be assigned some other value through use of the EQU assembler directive. A symbol may not appear in the name field more than once in a program, because this would cause the assembler to try to redefine an already defined label. The assembler will not do this and will flag the second appearance of a particular label as an error.

SYMBOLIC REFERENCES

Symbols may be used to refer to storage designations, register assignments, constants, etc. For example:

```
Address          Name          Operation          Argument
  101            MAZE            DATA               H'F5'
  102                            LODA,3             MAZE
```

The symbolic label "MAZE" represents the address 101. It is used in the machine instruction at address 102 to tell the assembler to build an instruction LODA,3   101. The symbolic label, in this case, is a way for the programmer to specify an address without knowing exactly what the address should be when he writes the program. In this example, assume there was a need to modify this sequence of code: a data statement was inserted between the original two statements.

```
Address          Name          Operation          Argument
 99              MAZE            DATA             H'F5''
 9A,9B                           DATA             H'FE,3A'
 9C                              LODA,3           MAZE
```

Even though there was a program change which caused the data at MAZE to be located at address 99, the load instruction referencing the data didn't have to be rewritten because the assembler could provide the proper physical address for the symbolic address MAZE. The instruction at address 9C will be assembled as LODA,3   99.

SYMBOLIC ADDRESSING

When writing instructions in the symbolic assembler language for the 2650, the addresses may be expressed through symbolic equivalents. The assembler will translate the symbolic address to its numeric equivalent during the assembly process.

It is good programming practice to make all address references symbolic, as this greatly eases the programmer's job in producing a working program. To make the register specification symbolic, one could equate a symbol to the register number:

```
RG3          EQU          3
             • • •
             • • •
             • • •
             • • •
             LODA,RG3     MAZE
```

Forward References

A previously defined symbol is one which has appeared in the name field before it is referenced (as above). In contrast, a forward reference is a symbolic reference to a line of code when the symbol has not yet appeared in the name field. For example:

```
             ADDA,2       COEF
             • • •
             • • •
             • • •
COEF         DATA         D'123'
```

Forward references may be used anywhere in a program with the following exceptions:

1. The register/condition field.
2. The symbolic argument fields of EQU, RES, ORG and DATA statements.

Relative Addressing

The programmer may reference a memory cell either directly or via relative addressing. To refer directly to a memory cell of symbolic address MAIN, one has merely to use the name MAIN in the argument field of the referencing instruction. For example:

```
             BIRA,R2      MAIN
```

It is also possible to express the address of a memory cell symbolically if some nearby cell is symbolically assigned. For example, to load the memory cell which is 5 cells higher in memory than the cell named MAIN, one need only to refer to it as MAIN+5:

```
             LODA,2       MAIN+5
```

This later method is called relative addressing, and the relative count may be given as + or – the maximum value which can be held in one integer variable of the host computer's FORTRAN compiler.

The Location Counter and Symbol "$"

There is one symbolic name, "$", which is automatically defined by the assembler. This single character name is always symbolically equated to the assembler's Location Counter. Since the Location Counter is used by the assembler during the assembly process and is usually equated to the address of the next byte to be assembled, it represents the address of the instruction or data currently being specified. For example: BCTR,3 $+5. The branch address will be interpreted by the assembler to be the address of the first byte of the branch instruction plus 5 bytes.

Hardware Relative Addressing

When using instructions which use "hardware relative addressing" (as distinguished from relative addressing discussed earlier in this section), it is important to realize the assembler will not only evaluate the expression which is given as an operand address, but will convert it to a hardware relative address (see the Hardware Specifications manual for a description of the addressing modes). For example:

```
Address        Name              Operation        Argument
  100          SAM               LODA,R2            PAL
  103                            SUBI,R2            -3
  105                            BIRR,R3            SAM
  107          next instruction
```

In this code, the BIRR instruction specifies hardware relative addressing. Even though the equivalent value of the symbolic address SAM is 100, the relative addressing instruction requires a displacement relative to the address of the next sequential instruction. Therefore, the operand SAM will be evaluated as = -(current location counter+length of BIRR instruction-SAM) = -(105+2-100) = -(+7) = -7. Remember, where the hardware instruction calls for "hardware relative addressing", the expression in the operand field will be evaluated as the displacement from the address of the next sequential instruction. The value of this displacement may range from -64 to +63.

Indirect Addressing

The symbol "*" is used to specify indirect addressing. For example:

```
             BCTA,3        *SAM
             • • •
             • • •
             • • •
SAM          ACON          SUBR
```

In this code, the BCTA instruction specifies indirect addressing. The assembler will set the indirect bit (byte #1, bit #7) for this instruction.

Auto-Increment and Auto-Decrement

The symbol "+" and "-" are used to specify auto-increment and auto-decrement, respectively. For example:

```
             LODA,R0       BUF,R3,+
```

In this code, which specifies auto-increment, the assembler sets bits #6 and #5 of byte #1 to "01" for this instruction. This option is specified in the instruction set tables as (,X).

IV PROCESSOR INSTRUCTIONS

2650 machine instructions may be written in symbolic code. All features provided by the assembler such as symbolic addressing and constant generation may be used. The fields described below are free form and are separated by at least one blank character. The name, however, if present, must begin in logical column 1.

LABEL	OPERATION	OPERAND	COMMENTS
name	opcode	operand(s)	

Where:

LABEL FIELD	contains an optional label which the assembler will assign as the symbolic address of the first byte of the instruction.
OPERATION FIELD	contains any of the 2650 processor mnemonic operation codes as detailed in Appendix A, or any Assembler Directive. This field may include an expression which specifies a register or value as required by the instruction. All symbols used in this field must have been previously defined, i.e., no symbolic forward references are allowed.
OPERAND FIELD	contains one or more operand elements such as indirect address indicator, operand expression, index register specification, auto-increment/auto-decrement indicator, constant specification, etc., depending on the requirements of the particular instruction.
COMMENTS FIELD	any characters following the argument field will be reproduced in the assembly listing without processing. The Comments Field must be separated from the argument field by at least one blank.

Note: Refer to Appendix A for a summary of the mnemonic op-codes and see 2650 Hardware Specification manual.

V DIRECTIVES TO THE 2650 ASSEMBLER

There are eleven directives which the assembler will recognize. These assembler directives, although written much like processor instructions, are simply commands to the assembler instead of to the processor. They direct the assembler to perform specific tasks during the assembly process, but have no meaning to the 2650 processor. These assembler directives are:

ORG
EQU
ACON
DATA
RES
END
EJE
PRT
SPC
TITL
PCH

ORG SET LOCATION COUNTER

The ORG directive sets the assembly Location Counter to the location specified. The assembler assumes an ORG 0 at the beginning of the program if no ORG statement is given.

LABEL	OPERATION	OPERAND
{name}	ORG	expression

Where:

name	optionally provides a symbol whose value will be equated to the specified location.
expression	when evaluated, results in a positive integer value. This value will replace the contents of the location counter, and bytes, subsequently assembled will be assigned sequential memory addresses beginning with this value. Any symbols which appear in the argument must have been previously defined.

Examples:

```
LARR      ORG       YORD
STAR      ORG       H'100'
```

EQU SPECIFY A SYMBOL EQUIVALENCE

The EQU directive tells the assembler to equate the symbol in the name field with the evaluatable expression in the argument field.

LABEL	OPERATION	OPERAND
name	EQU	expression

Where:

name	is the symbol which is to be assigned some value by the execution of this directive.
expression	may be resolved to zero or some integer value which is containable in the host computer's FORTRAN integer cell. If a symbol is used in the argument, it must have been previously defined.

Examples:

```
PAL         EQU         H'10F'
LOP2        EQU         PAL
RAMP        EQU         SLOP-3+PAL
REG1        EQU         1
```

ACON DEFINE ADDRESS CONSTANT

The ACON directive tells the assembler to allocate two successive bytes of storage. The evaluated argument will be stored in the two bytes, the low order 8 bits in the second byte and the high order bits in the first byte. This directive is mainly intended to provide a double byte containing an address for use as the indirect address for any instruction executing in the indirect addressing mode.

LABEL	OPERATION	OPERAND
{name}	ACON	expression

Where:

name	is an optional label. If specified, the name becomes the symbolic address of the first byte allocated.
expression	is some expression which must resolve to a positive value or zero. If positive, the value should be no larger than that which can be contained in two bytes.

Example:

```
ASUB        ACON        SUBR
```

DATA DEFINES MEMORY DATA

The DATA directive tells the assembler to allocate the exact number of bytes required to hold the data specified in the argument field of this directive. Up to 16 bytes can be specified with one DATA directive, but the argument field may not extend past logical column 72.

LABEL	OPERATION	OPERAND
{name}	DATA	expression

Where:

name	is an optional label. If used, the name becomes the symbolic address of the first byte allocated by the directive.
expression	is a general constant, a self-defining constant or a symbolic address. If a symbol is specified, it must have been previously defined. A multiple constant specification in the argument field will cause a corresponding number of bytes to be allocated. Any other expression that can be resolved to a single value will result in one byte being allocated.

Examples:

```
PAL         DATA        LOOP
            DATA        H'03,22,FC,A1'
            DATA        +127
            DATA        D'28'
```

Note: If the expression evaluates to a value between 0 and 255 the result is an eight bit absolute binary number. DATA +127 results in H'7F'. Also, if the expression evaluates to a value which is less than 0 the result is a 2's complement, binary number. DATA H'-5' results in H'FB'.

RES RESERVE MEMORY STORAGE

The RES directive tells the assembler to reserve contiguous bytes of storage. The number of bytes so reserved is determined by the argument. The reserved bytes are not set to a known value, but rather the effect of this directive is to increment the location counter.

LABEL	OPERATION	OPERAND
{name}	RES	expression

Where:

name	is an optional label. If used, the name becomes the symbolic address of the first byte allocated.
expression	is some evaluatable expression which must resolve to some positive integer or zero. The value of this expression may not exceed the maximum positive value containable in a FORTRAN cell of the host computer. If a symbol is specified, it must have been previously defined.

Example:

```
LOR         RES         23
MASK        RES         LOR+5
            RES         H'1A'
```

END END OF ASSEMBLY

The END directive informs the assembler that the last statement to be assembled has been input and the assembler may proceed with the assembly. The END directive causes the assembler to communicate the program start address to the object module.

LABEL	OPERATION	OPERAND
	END	expression

Where:

expression	may be resolved to the starting address of the program. If this parameter is not specified, the start address is set to zero.

EJE EJECT THE LISTING PAGE

The EJE directive tells the assembler to advance the listing to the top of the next page regardless of the line position on the current listing page.

The directive is used primarily to organize listing for documentation purposes and does not appear in the listing.

LABEL	OPERATION	OPERAND
	EJE	

PRT PRINTER CONTROL

The PRT directive tells the assembler to resume or discontinue printing of the assembled program.

This directive is used primarily to shorten assembly time by listing only that portion of the program which the user needs to see. Only the PRT OFF will appear in the listing.

LABEL	OPERATION	OPERAND
	PRT	$\left\{\begin{matrix}\text{on}\\\text{off}\end{matrix}\right\}$

Note: PRT is set ON at the beginning of an assembly of the assembler.

SPC SPACE CONTROL

The SPC directive tells the assembler to skip or space a number of lines.

This directive is used primarily to organize listings for documentation purposes and does not appear in the listing.

LABEL	OPERATION	OPERAND
	SPC	expression

Where:

expression	is some evaluatable expression which must resolve to some positive integer. If the value of this expression is equal to, or greater than, the number of lines remaining on the page, the effect is the same as the EJE directive.

Example:

```
SPC         5
```

TITL TITLE

The TITL directive tells the assembler to skip to the top of the next page and insert a given title into the main header.

This directive is used primarily for documentation purposes and does not appear in the listing.

LABEL	OPERATION	OPERAND
	TITL	expression

Where:

expression	is the title information not to exceed forty character positions.

Example:

```
TITL        MAIN PROGRAM SUBROUTINE
```

PCH PUNCH CONTROL

The PCH directive tells the assembler to selectively resume or discontinue the output of the load module.

This directive is used primarily to shorten assembly time when a load module is not desired or when only a portion of the load module is desired.

LABEL	OPERATION	OPERAND
	PCH	$\begin{Bmatrix} \text{on} \\ \text{off} \end{Bmatrix}$

Note: PCH is set ON at the beginning of an assembly by the assembler. When PCH OFF is specified, any prior load module data is output.

VI THE ASSEMBLY PROCESS

The 2650 assembler translates symbolic source code into machine language instructions. The assembler examines every source statement for syntactic validity and produces the equivalent machine code for the 2650 processor.

This is a two pass assembler, which means, the entire source code is scanned twice by the assembler. On the first pass, all defined labels and their equivalent values are stored in a symbol table, the first byte of every instruction is fully determined, and some errors may be detected. During pass 2, symbolic address references are replaced by their values, errors may be detected, and a listing and load/object module is generated.

Symbol Table

The assembler builds and maintains a symbol table during the assembly process. The symbol table contains an entry for each symbol in the assembled program. The entry consists of the symbol itself and its value. Up to 400 symbols may be used in each program assembled. If a symbol, which appears in the argument field of an instruction has never been defined (never appeared in the NAME field), the assembler will generate an error code on the listing because it is unable to resolve an undefined symbol and will place zero as the unresolved value in the object module.

Location Counter

The assembler maintains a memory cell which it uses as a Location Counter. This Location Counter keeps track of the address of the next byte of storage to be allocated by the assembler. During coding, the programmer may think of the Location Counter as containing the address of the first byte of the instruction being written. In this assembler, the Location Counter is also used to provide load information. This means that the addresses displayed on an assembly listing are the actual addresses which are to contain the corresponding information upon loading of the object program.

Error Detection

During an assembly, the source program is checked for syntax errors. If errors are found, appropriate notification is given and the assembly proceeds. Although an assembled program containing errors generally will not run properly, it is considered good practice to complete the assembly to locate all errors at one time, rather than terminate it when an error is encountered.

Error Codes

As shown in the listing illustration, there are three columns on the listing in which an error indication may appear. An error displayed, in the first column usually indicates that the error was in the Name Field, the second column corresponds to the Operation Field, and the third corresponds to the Argument Field. Sometimes because an error causes the assembler to view the next field incorrectly, a valid field may be flagged as an error. This is a consequence of the free format source language. A good rule is to fix errors in a particular line of code as they are discovered. In this way, erroneously flagged program errors may then be passed as valid.

The following alphabetic characters are printed in the error indicator columns and imply the corresponding message.

L — Label error. The label contains too many characters, contains invalid characters, has been previously defined, or is an invalid symbol.

O — Op-code error. The op-code mnemonic has not been recognized as a valid mnemonic.

R — Register field error. The register field expression could not be evaluated, or when evaluated, was less than 0 or greater than 3, or the register field was not found.

S — Syntax error. The instruction has violated some syntax rule.

U — Undefined symbol. There is a symbol in the argument field which has not been previously defined.

A — Argument error. The argument has been coded in such a way that it cannot be resolved to a unique value.

P — Paging error. A memory access instruction has attempted to address across a page boundary.

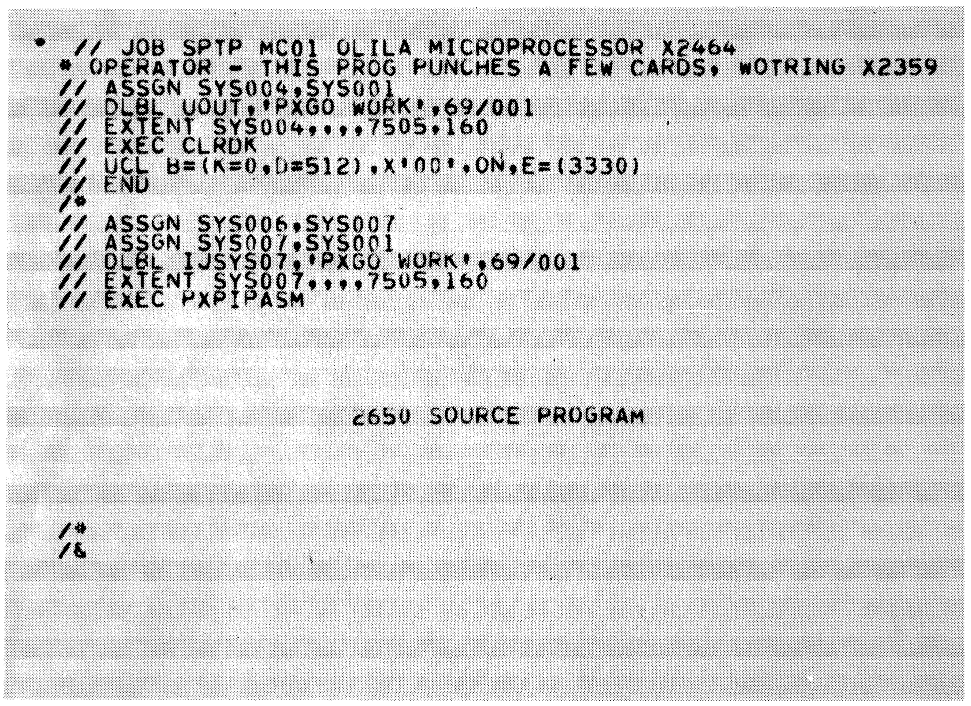
W — Warning. The assembler has detected a syntactically correct but unusual construction. The error will not be counted and will not inhibit the production of the object module.

Using the Assembler

The program is prepared by punching it into cards or otherwise transferring the program statements into a logical card image file. An ORG statement usually occurs early in the program. If no ORG appears, the assembler assumes an ORG 0 to occur before the first assembled statement. An END statement must occur as the last statement. A program written in the 2650 Symbolic Assembler Language should be preceded and possibly followed by control cards for the particular computer system which is being used. Illustration VI-1 shows the control cards for an IBM/370 DOS system. Although the control cards may vary from system to system, the format of the actual 2650 source program will be the same in the system.

The object module produced by the Assembler during pass 2 is directed to the FORTRAN standard device #2, in this instance the card punch. The source program is read by the assembler at standard device #1, the card reader. In some systems the device assignments may be altered if desired, through assign cards. In other systems, however, the assembler must be recompiled with the device numbers desired being set in the main program module.

ILLUSTRATION VI-1



```
• // JOB SPTP MC01 OLILA MICROPROCESSOR X2464
  * OPERATOR - THIS PROG PUNCHES A FEW CARDS, WOTRING X2359
  // ASSGN SYS004,SYS001
  // DLBL UOUT,'PXGO WORK',69/001
  // EXTENT SYS004,,,,7505,160
  // EXEC CLRDK
  // UCL B=(K=0,D=512),X'00',ON,E=(3330)
  // END
  /*
  // ASSGN SYS006,SYS007
  // ASSGN SYS007,SYS001
  // DLBL IJSYS07,'PXGO WORK',69/001
  // EXTENT SYS007,,,,7505,160
  // EXEC PXPIPASM


                    2650 SOURCE PROGRAM


  /*
  /&
```

Object Module

The format of the object module is: The first card or card image is always all 9's.

bb999999999999999

The second and all subsequent data cards are in the following format. Logical columns (1-5) contain the load address in decimal. Each three columns (6-71) contain the data to be loaded in decimal. Each three columns represent a byte of data; columns (6-8), (9-11), (12-14), etc. Beginning at the address indicated in columns (1-5) each sequential data byte is to be loaded into sequentially ascending addresses in memory. If a '999' appears in a particular data byte position, that byte of information is to be ignored by the loader and the contents of the corresponding location is not modified.

Because there is address and data on every card image, each card image is independent. Therefore, the order of the data cards is unimportant and patch cards may be prepared manually by preparing a data card in the object module format.

The last two card images each serve a special purpose. The next to last card contains a series of '-1' punches. This card is used to signal the end of load information and has no other function.

The last card, which follows the '-1' card, contains either the start address (specified in assembler END statement) or zero in columns (1-5), the remainder of the card contains '-1' punches which have no meaning.

ASSEMBLY LISTING

Illustration VI-2 is a sample of a program listing produced by the 2650 Assembler. The following explanations are keyed to the listing.

1. Page heading — which displays the current version and level of the 2650 Assembler.
2. Line number — every assembled line is assigned a line number for the programmer's convenience.
3. Address column — The numbers in this column are equal to the value of the assembly Location Counter and indicate the address at which the first byte (B1) is to be loaded.
4. Label column — If there is a symbol in the Label Field of a line of code, the value of the label will appear in this column. For example, in line number 17 the value of the label SORT is H'0007'.
5. Data field — This field describes the data bytes which are to be stored sequentially starting at the address in the Address Column.
6. Error columns — These columns may contain the error codes as detailed elsewhere in this chapter.
7. Source code — This area of the listing reproduces the source code as it was read by the assembler.
8. Page number — Every page of the listing is numbered sequentially.
9. Cumulative errors — This field indicates the total of errors detected by the assembler during the assembly process. Warning messages (W) are not included in this total.

ILLUSTRATION VI-2

```
     ASSEMBLER VERSION 2 LEVEL 0                                                              PAGE    1
  LINE  ADDR  LABL  B1 B2 B3 B4 ERROR SOURCE

     1                                     *   ARITHMETIC BUBBLE SORT PROGRAM FOR PIP
     2  0000  0000                         R0   EQU     0
     3  000C  0001                         R1   EQU     1
     4  000C  0002                         R2   EQU     2
     5  000C  0003                         R3   EQU     3
     6  0000  0003                         UN   EQU     3
     7  0000  0001                         GT   EQU     1
     8  0000  0002                         LT   EQU     2
     9  0000  0000                         EQ   EQU     0
    10  000C  0000  00                     ZERO DATA    0
    11  0001  C001                         CNT  RES     1         NUMBER OF ITERATIONS TO PERFORM
    12                                     *
    13                                     *
    14                                     * MAIN PROGRAM
    15  0002  0002  0F 00 C8               STRT LODA,R3   LEN     LOAD BUFFER LENGTH INTO REG 3
    16  0005        75 02                       CPSL      2       SET FOR ARITHMATIC CCMPARISONS (NOT LOGICAL)
    17  0007  0007  A7 01                  SORT SUBI,R3   1       DECREMENT LOOP COUNTER
    18  0009        CF 00 01                    STRA,R3   CNT     STORE LOOP COUNTER
    19  000C        7F 00 12                    BSNA,R3   SUB1    IF NCT ZERO, CALL SUBROUTINE
    20  000F        58 76                       BRNR,R3   SORT    IF NCT ZERO, LOOP BACK AGAIN
    21  0011        40                          HALT
    22                                     *
    23                                     * SUBRCUTINE FOR PERFORMING ONE ITERATION THROUGH BUFFER
    24  0012  0012  0E 00 00               SUB1 LCDA,R2   ZERO    REG 2 COUNTS COMPARISONS
    25  0015  0015  EE 00 01               LOOP CCMA,R2   CNT     IF EQUAL, ITERATICN COMPLETE
    26  0018        14                          RETC,EQ
    27  0019        0E 60 C9                    LCDA,R0   BUF,R2  LOAD FIRST NUMBER OF CURRENT PAIR
    28  001C        EE 20 C9                    CCMA,R0   BUF,R2,+ COMPARE WITH SECOND NUMBER
    29  001F        99 74                       BCFR,GT   LOOP    IF FIRST LT OR = SECOND, LOOP BACK
    30  0021        C1                          STRZ  R1          MOVE LARGER NUMBER TO REG 1
    31  0022        0E 60 C9                    LCDA,R0   BUF,R2  LOAC SMALLER NUMBER INTO REG 0
    32  0025        CE 60 C8                    STRA,R0   BUF-1,R2 STORE SMALLER NUMBER IN FIRST LOCATION
    33  0028        01                          LCDZ  R1          MOVE LARGER NUMBER TO REG 0
    34  0029        CE 60 C9                    STRA,R0   BUF,R2  STORE LARGER NUMBER IN SECOND LOCATION
    35  002C        18 67                       BCTR,UN   LCOP    LOOP BACK
    36                                     *
    37                                     *
    38  00C8                                    CRG       200
    39  00C8  00C8  0F                     LEN  CATA      15      LENGTH OF BUFFER TO BE SORTED
    40  00C9  00C9                         BUF  RES       15      BUFFER TO BE SORTED
    41  0CD8                                    END       STRT


 TOTAL ASSEMBLER ERRORS =      C
```

APPENDIX A

SUMMARY OF 2650 INSTRUCTION MNEMONICS

In these tables parentheses are used to indicate options. In no case are they coded in any instruction. The following abbreviations are used:

r — register expression, must evaluate to $0 \leqslant r \leqslant 3$.
v — value expression
* — indirect indicator
a — address expression
x — index register expression
X — index register expression with optional auto-increment or auto-decrement

NOTE:
— the use of the indirect indicator is always optional.
— when an index register expression is specified, it can be followed by ', +' or ', -' which indicates use of auto-increment or auto-decrement of the index register. Example:

LODA,0 DPR,R3,+

BXA, BSXA are exceptions and do not permit auto-increment or auto-decrement.
— even though an address expression is specified in a hardware relative addressing instruction, the assembler develops it into a value of ($-64 \leqslant V \leqslant +63$).
— a memory reference instruction which requires indexing may use only register 0 as the destination of the operation.
— if an index register expression is used with either the BXA or BSXA instructions it must specify index register #3 (either register bank) for indexing. Any other value in the index field will produce an error during assembly. However, it is not necessary to use an index register expression with these instructions; a blank in this field will default to register 3.

LOAD/STORE INSTRUCTIONS			Length (bytes)
LODZ	r	Load Register Zero	1
LODI,r	v	Load Immediate	2
LODR,r	(*)a	Load Relative	2
LODA,r	(*)a(,X)	Load Absolute	3
STRZ	r	Store Register Zero	1
STRR,r	(*)a	Store Relative	2
STRA,r	(*)a(,X)	Store Absolute	3
ARITHMETIC INSTRUCTIONS			
ADDZ	r	Add to Register Zero	1
ADDI,r	v	Add Immediate	2
ADDR,r	(*)a	Add Relative	2
ADDA,r	(*)a(,X)	Add Absolute	3
SUBZ	r	Subtract from Register Zero	1
SUBI,r	v	Subtract Immediate	2
SUBR,r	(*)a	Subtract Relative	2
SUBA,r	(*)a(,X)	Subtract Absolute	3
LOGICAL INSTRUCTIONS			
ANDZ	r	And to Register Zero	1
ANDI,r	v	And Immediate	2
ANDR,r	(*)a	And Relative	2
ANDA,r	(*)a(,X)	And Absolute	3
IORZ	r	Inclusive or to Register Zero	1
IORI,r	v	Inclusive or Immediate	2
IORR,r	(*)a	Inclusive or Relative	2
IORA,r	(*)a(,X)	Inclusive or Absolute	3
EORZ	r	Exclusive or to Register Zero	1
EORI,r	v	Exclusive or Immediate	2
EORR,r	(*)a	Exclusive or Relative	2
EORA,r	(*)a(,X)	Exclusive or Absolute	3
COMPARISON INSTRUCTIONS			
COMZ	r	Compare to Register Zero	1
COMI,r	v	Compare Immediate	2
COMR,r	(*)a	Compare Relative	2
COMA,r	(*)a(,X)	Compare Absolute	3

ROTATE INSTRUCTIONS			Length (bytes)
RRR,r		Rotate Register Right	1
RRL,r		Rotate Register Left	1
BRANCH INSTRUCTIONS			
BCTR,v	(*)a	Branch on Condition True Relative	2
BCFR,v	(*)a	Branch on Condition False Relative	2
BCTA,v	(*)a	Branch on Condition True Absolute	3
BCFA,v	(*)a	Branch on Condition False Absolute	3
BRNR,r	(*)a	Branch on Register Non-Zero Relative	2
BRNA,r	(*)a	Branch on Register Non-Zero Absolute	3
BIRR,r	(*)a	Branch on Incrementing Register Relative	2
BIRA,r	(*)a	Branch on Incrementing Register Absolute	3
BDRR,r	(*)a	Branch on Decrementing Register Relative	2
BDRA,r	(*)a	Branch on Decrementing Register Absolute	3
BXA	(*)a(,x)	Branch Indexed Absolute, Unconditional	3
ZBRR	(*)a	Zero Branch Relative, Unconditional	2
SUBROUTINE BRANCH/RETURN INSTRUCTIONS			
BSTR,v	(*)a	Branch to Subroutine on Condition True, Relative	2
BSFR,v	(*)a	Branch to Subroutine on Condition False, Relative	2
BSTA,v	(*)a	Branch to Subroutine on Condition True, Absolute	3
BSFA,v	(*)a	Branch to Subroutine on Condition False, Absolute	3
BSNR,r	(*)a	Branch to Subroutine on Non-Zero Register, Relative	2
BSNA,r	(*)a	Branch to Subroutine on Non-Zero Register, Absolute	3
BSXA	(*)a(,x)	Branch to Subroutine, Indexed, Unconditional	3
RETC,v		Return From Subroutine, Conditional	1
RETE,v		Return From Subroutine and Enable Interrupt, Conditional	1
ZBSR	(*),a	Zero Branch to Subroutine Relative, Unconditional	2
PROGRAM STATUS INSTRUCTIONS			
LPSU		Load Program Status, Upper	1
LPSL		Load Program Status, Lower	1
SPSU		Store Program Status, Upper	1
SPSL		Store Program Status, Lower	1
CPSU	v	Clear Program Status, Upper, Selective	2
CPSL	v	Clear Program Status, Lower, Selective	2
PPSU	v	Preset Program Status, Upper, Selective	2
PPSL	v	Preset Program Status, Lower, Selective	2
TPSU	v	Test Program Status, Upper, Selective	2
TPSL	v	Test Program Status Lower, Selective	2
INPUT/OUTPUT INSTRUCTIONS			
WRTD,r		Write Data	1
REDD,r		Read Data	1
WRTC,r		Write Control	1
REDC,r		Read Control	1
WRTE,r	v	Write Extended	2
REDE,r	v	Read Extended	2
MISCELLANEOUS INSTRUCTIONS			
HALT		Halt, Enter Wait State	1
DAR,r		Decimal Adjust Register	1
TMI,r	v	Test Under Mask Immediate	2
NOP		No Operation	1

APPENDIX B

NOTES ABOUT THE 2650 PROCESSOR

1. AUTO-INCREMENT, DECREMENT of index register. This feature is optional on any instruction which uses indexing with the exception of BXA and BSXA. The increment or decrement occurs before the index register is added to the displacement in the instruction.

2. The contents of registers when used for indexing are considered to be unsigned absolute numbers. Consequently, index registers can contain values from 0 to 255. They "wrap-around" so that the number following 255 is 0.

3. Only absolute addressing instructions can be indexed.

4. The Branch on Incrementing Register or Decrementing Register instructions perform the increment or decrement before testing for zero. The only time the branch address is not taken, is when the register contains zero.

5. All hardware relative addressing is implemented as modulo 8K and therefore relative addressing across the top of a page boundary will result in a physical address near the bottom of the page being accessed. For example:

   ```
   1FFC16          LODR,R2          $+16
   ```

   This instruction results, during execution, in accessing the byte at location 000C in the same page as the instruction. Similarly, negative relative addresses from near the bottom of a page may result in an effective address near the top of the page.

6. Page boundaries cannot be indexed across.

7. Data can always be accessed across a page boundary through use of relative indirect or absolute indirect addressing modes.

8. The only way to transfer control to a program in some other page is to branch absolute or branch indirectly to the new page. Program execution cannot flow across a page boundary.

9. Unconditional branch or branch to subroutine instructions are coded by specifying a value of 3 in the register/value field of BSTA, BSTR, BCTA or BCTR. Example:

```
UN          EQU         3
            • • •
            • • •
            • • •
            BSTA,UN     PAL
            BCTR,3      LOOP
```

Unconditional branches on conditions false (BCFA, BCFR) are not allowed.

APPENDIX C

ASC II AND EBCDIC CODES

This table presents the only characters that the assembler will recognize in an A or E type constant and their equivalent codes in hexadecimal.

VALID CHARACTERS	EBCDIC CODE	ASC II CODE	VALID CHARACTERS	EBCDIC CODE	ASC II CODE
0	F0	30	V	E5	56
1	F1	31	W	E6	57
2	F2	32	X	E7	58
3	F3	33	Y	E8	59
4	F4	34	Z	E9	5A
5	F5	35	blank	40	20
6	F6	36	.	4B	2E
7	F7	37	(	4D	28
8	F8	38	+	4E	2B
9	F9	39	¦	4F	7C
A	C1	41	&	50	26
B	C2	42	!	5A	21
C	C3	43	$	5B	24
D	C4	44	*	5C	2A
E	C5	45	)	5D	29
F	C6	46	;	5E	3B
G	C7	47	¬ or ~	5F	7E*
H	C8	48	-	60	2D
I	C9	49	/	61	2F
J	D1	4A	,	6B	2C
K	D2	4B	%	6C	25
L	D3	4C	— or ←	6D	5F*
M	D4	4D	>	6E	3E
N	D5	4E	?	6F	3F
O	D6	4F	:	7A	3A
P	D7	50	#	7B	23
Q	D8	51	@	7C	40
R	D9	52	'	7D	27
S	E2	53	=	7E	3D
T	E3	54	”	7F	22
U	E4	55	<	4C	3C

*may have different graphic symbols on different computer systems

APPENDIX D

COMPLETE ASCII CHARACTER SET

			(MSB) b_7	0	0	1	1	1	1
			b_6	1	1	0	0	1	1
			b_5	0	1	0	1	0	1
b_4	b_3	b_2	b_1						
0	0	0	0	SP	0	@	P	`	p
0	0	0	1	!	1	A	Q	a	q
0	0	1	0	"	2	B	R	b	r
0	0	1	1	#	3	C	S	c	s
0	1	0	0	$	4	D	T	d	t
0	1	0	1	%	5	E	U	e	u
0	1	1	0	&	6	F	V	f	v
0	1	1	1	'	7	G	W	g	w
1	0	0	0	(	8	H	X	h	x
1	0	0	1	)	9	I	Y	i	y
1	0	1	0	*	:	J	Z	j	z
1	0	1	1	+	;	K	[	k	{
1	1	0	0	'	<	L	\	l	|
1	1	0	1	–	=	M	]	m	}
1	1	1	0	·	>	N	↑	n	~
1	1	1	1	/	?	O	←	o	DEL

APPENDIX E

POWERS OF TWO TABLE

2^n	n	2^{-n}
1	0	1.0
2	1	0.5
4	2	0.25
8	3	0.125
16	4	0.062 5
32	5	0.031 25
64	6	0.015 625
128	7	0.007 812 5
256	8	0.003 906 25
512	9	0.001 953 125
1 024	10	0.000 976 562 5
2 048	11	0.000 488 281 25
4 096	12	0.000 244 140 625
8 192	13	0.000 122 070 312 5
16 384	14	0.000 061 035 156 25
32 768	15	0.000 030 517 578 125
65 536	16	0.000 015 258 789 062 5
131 072	17	0.000 007 629 394 531 25
262 144	18	0.000 003 814 697 265 625
524 288	19	0.000 001 907 348 632 812 5
1 048 576	20	0.000 000 953 674 316 406 25
2 097 152	21	0.000 000 476 837 158 203 125
4 194 304	22	0.000 000 238 418 579 101 562 5
8 388 608	23	0.000 000 119 209 289 550 781 25
16 777 216	24	0.000 000 059 604 644 775 390 625
33 554 432	25	0.000 000 029 802 322 387 695 312 5
67 108 864	26	0.000 000 014 901 161 193 847 656 25
134 217 728	27	0.000 000 007 450 580 596 923 828 125
268 435 456	28	0.000 000 003 725 290 298 461 914 062 5
536 870 912	29	0.000 000 001 862 645 149 230 957 031 45
1 073 741 824	30	0.000 000 000 931 322 574 615 478 515 625
2 147 483 648	31	0.000 000 000 465 661 287 307 739 257 812 5
4 294 967 296	32	0.000 000 000 232 830 643 653 869 628 906 25
8 589 934 592	33	0.000 000 000 116 415 321 826 934 814 453 125
17 179 869 184	34	0.000 000 000 058 207 660 913 467 407 226 562 5
34 359 738 368	35	0.000 000 000 029 103 830 456 733 703 613 281 25
68 719 476 736	36	0.000 000 000 014 551 915 228 366 851 806 640 625
137 438 953 472	37	0.000 000 000 007 275 957 614 183 425 903 320 312 5
274 877 906 944	38	0.000 000 000 003 637 978 807 091 712 951 660 156 25
549 755 813 888	39	0.000 000 000 001 818 989 403 545 856 475 830 078 125
1 099 511 627 776	40	0.000 000 000 000 909 494 701 772 928 237 915 039 062 5

APPENDIX F

HEXADECIMAL-DECIMAL CONVERSION TABLES

From hex: locate each hex digit in its corresponding column position and note the decimal equivalents. Add these to obtain the decimal value.

From decimal: (1) locate the largest decimal value in the table that will fit into the decimal number to be converted, and (2) note its hex equivalent and hex column position. (3) Find the decimal remainder. Repeat the process on this and subsequent remainders.

Note: Decimal, hexadecimal, (and binary) equivalents of all numbers from 0 to 255 are listed on panels 9 - 12.

HEXADECIMAL COLUMNS											
6		5		4		3		2		1	
HEX	DEC	HEX	DEC	HEX	DEC	HEX	DEC	HEX	DEC	HEX	DEC
0	0	0	0	0	0	0	0	0	0	0	0
1	1,048,576	1	65,536	1	4,096	1	256	1	16	1	1
2	2,097,152	2	131,072	2	8,192	2	512	2	32	2	2
3	3,145,728	3	196,608	3	12,288	3	768	3	48	3	3
4	4,194,304	4	262,144	4	16,384	4	1,024	4	64	4	4
5	5,242,880	5	327,680	5	20,480	5	1,280	5	80	5	5
6	6,291,456	6	393,216	6	24,576	6	1,536	6	96	6	6
7	7,340,032	7	458,752	7	28,672	7	1,792	7	112	7	7
8	8,388,608	8	524,288	8	32,768	8	2,048	8	128	8	8
9	9,437,184	9	589,824	9	36,864	9	2,304	9	144	9	9
A	10,485,760	A	655,360	A	40,960	A	2,560	A	160	A	10
B	11,534,336	B	720,896	B	45,056	B	2,816	B	176	B	11
C	12,582,912	C	786,432	C	49,152	C	3,072	C	192	C	12
D	13,631,488	D	851,968	D	53,248	D	3,328	D	208	D	13
E	14,680,064	E	917,504	E	57,344	E	3,584	E	224	E	14
F	15,728,640	F	983,040	F	61,440	F	3,840	F	240	F	15
0 1 2 3		4 5 6 7		0 1 2 3		4 5 6 7		0 1 2 3		4 5 6 7	
BYTE				BYTE				BYTE			

The table provides for direct conversion of hexadecimal and decimal numbers in these ranges:

Hexadecimal	Decimal
000 to FFF	0000 to 4095

In the table, the decimal value appears at the intersection of the row representing the most significant hexadecimal digits (16^2 and 16^1) and the column representing the least significant hexadecimal digit (16^0).

Example: $C21_{16} = 3105_{10}$

HEX	0	1	2
C0	3072	3073	3074
C1	3088	3089	3090
C2	3104	3105	3106
C3	3120	3121	3122

APPENDIX F Cont'd.

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
00	0000	0001	0002	0003	0004	0005	0006	0007	0008	0009	0010	0011	0012	0013	0014	0015
01	0016	0017	0018	0019	0020	0021	0022	0023	0024	0025	0026	0027	0028	0029	0030	0031
02	0032	0033	0034	0035	0036	0037	0038	0039	0040	0041	0042	0043	0044	0045	0046	0047
03	0048	0049	0050	0051	0052	0053	0054	0055	0056	0057	0058	0059	0060	0061	0062	0063
04	0064	0065	0066	0067	0068	0069	0070	0071	0072	0073	0074	0075	0076	0077	0078	0079
05	0080	0081	0082	0083	0084	0085	0086	0087	0088	0089	0090	0091	0092	0093	0094	0095
06	0096	0097	0098	0099	0100	0101	0102	0103	0104	0105	0106	0107	0108	0109	0110	0111
07	0112	0113	0114	0115	0116	0117	0118	0119	0120	0121	0122	0123	0124	0125	0126	0127
08	0128	0129	0130	0131	0132	0133	0134	0135	0136	0137	0138	0139	0140	0141	0142	0143
09	0144	0145	0146	0147	0148	0149	0150	0151	0152	0153	0154	0155	0156	0157	0158	0159
0A	0160	0161	0162	0163	0164	0165	0166	0167	0168	0169	0170	0171	0172	0173	0174	0175
0B	0176	0177	0178	0179	0180	0181	0182	0183	0184	0185	0186	0187	0188	0189	0190	0191
0C	0192	0193	0194	0195	0196	0197	0198	0199	0200	0201	0202	0203	0204	0205	0206	0207
0D	0208	0209	0210	0211	0212	0213	0214	0215	0216	0217	0218	0219	0220	0221	0222	0223
0E	0224	0225	0226	0227	0228	0229	0230	0231	0232	0233	0234	0235	0236	0237	0238	0239
0F	0240	0241	0242	0243	0244	0245	0246	0247	0248	0249	0250	0251	0252	0253	0254	0255

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
10	0256	0257	0258	0259	0260	0261	0262	0263	0264	0265	0266	0267	0268	0269	0270	0271
11	0272	0273	0274	0275	0276	0277	0278	0279	0280	0281	0282	0283	0284	0285	0286	0287
12	0288	0289	0290	0291	0292	0293	0294	0295	0296	0297	0298	0299	0300	0301	0302	0303
13	0304	0305	0306	0307	0308	0309	0310	0311	0312	0313	0314	0315	0316	0317	0318	0319
14	0320	0321	0322	0323	0324	0325	0326	0327	0328	0329	0330	0331	0332	0333	0334	0335
15	0336	0337	0338	0339	0340	0341	0342	0343	0344	0345	0346	0347	0348	0349	0350	0351
16	0352	0353	0354	0355	0356	0357	0358	0359	0360	0361	0362	0363	0364	0365	0366	0367
17	0368	0369	0370	0371	0372	0373	0374	0375	0376	0377	0378	0379	0380	0381	0382	0383
18	0384	0385	0386	0387	0388	0389	0390	0391	0392	0393	0394	0395	0396	0397	0398	0399
19	0400	0401	0402	0403	0404	0405	0406	0407	0408	0409	0410	0411	0412	0413	0414	0415
1A	0416	0417	0418	0419	0420	0421	0422	0423	0424	0425	0426	0427	0428	0429	0430	0431
1B	0432	0433	0434	0435	0436	0437	0438	0439	0440	0441	0442	0443	0444	0445	0446	0447
1C	0448	0449	0450	0451	0452	0453	0454	0455	0456	0457	0458	0459	0460	0461	0462	0463
1D	0464	0465	0466	0467	0468	0469	0470	0471	0472	0473	0474	0475	0476	0477	0478	0479
1E	0480	0481	0482	0483	0484	0485	0486	0487	0488	0489	0490	0491	0492	0493	0494	0495
1F	0496	0497	0498	0499	0500	0501	0502	0503	0504	0505	0506	0507	0508	0509	0510	0511

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
20	0512	0513	0514	0515	0516	0517	0518	0519	0520	0521	0522	0523	0524	0525	0526	0527
21	0528	0529	0530	0531	0532	0533	0534	0535	0536	0537	0538	0539	0540	0541	0542	0543
22	0544	0545	0546	0547	0548	0549	0550	0551	0552	0553	0554	0555	0556	0557	0558	0559
23	0560	0561	0562	0563	0564	0565	0566	0567	0568	0569	0570	0571	0572	0573	0574	0575
24	0576	0577	0578	0579	0580	0581	0582	0583	0584	0585	0586	0587	0588	0589	0590	0591
25	0592	0593	0594	0595	0596	0597	0598	0599	0600	0601	0602	0603	0604	0605	0606	0607
26	0608	0609	0610	0611	0612	0613	0614	0615	0616	0617	0618	0619	0620	0621	0622	0623
27	0624	0625	0626	0627	0628	0629	0630	0631	0632	0633	0634	0635	0636	0637	0638	0639
28	0640	0641	0642	0643	0644	0645	0646	0647	0648	0649	0650	0651	0652	0653	0654	0655
29	0656	0657	0658	0659	0660	0661	0662	0663	0664	0665	0666	0667	0668	0669	0670	0671
2A	0672	0673	0674	0675	0676	0677	0678	0679	0680	0681	0682	0683	0684	0685	0686	0687
2B	0688	0689	0690	0691	0692	0693	0694	0695	0696	0697	0698	0699	0700	0701	0702	0703
2C	0704	0705	0706	0707	0708	0709	0710	0711	0712	0713	0714	0715	0716	0717	0718	0719
2D	0720	0721	0722	0723	0724	0725	0726	0727	0728	0729	0730	0731	0732	0733	0734	0735
2E	0736	0737	0738	0739	0740	0741	0742	0743	0744	0745	0746	0747	0748	0749	0750	0751
2F	0752	0753	0754	0755	0756	0757	0758	0759	0760	0761	0762	0763	0764	0765	0766	0767

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
30	0768	0769	0770	0771	0772	0773	0774	0775	0776	0777	0778	0779	0780	0781	0782	0783
31	0784	0785	0786	0787	0788	0789	0790	0791	0792	0793	0794	0795	0796	0797	0798	0799
32	0800	0801	0802	0803	0804	0805	0806	0807	0808	0809	0810	0811	0812	0813	0814	0815
33	0816	0817	0818	0819	0820	0821	0822	0823	0824	0825	0826	0827	0828	0829	0830	0831
34	0832	0833	0834	0835	0836	0837	0838	0839	0840	0841	0842	0843	0844	0845	0846	0847
35	0848	0849	0850	0851	0852	0853	0854	0855	0856	0857	0858	0859	0860	0861	0862	0863
36	0864	0865	0866	0867	0868	0869	0870	0871	0872	0873	0874	0875	0876	0877	0878	0879
37	0880	0881	0882	0883	0884	0885	0886	0887	0888	0889	0890	0891	0892	0893	0894	0895
38	0896	0897	0898	0899	0900	0901	0902	0903	0904	0905	0906	0907	0908	0909	0910	0911
39	0912	0913	0914	0915	0916	0917	0918	0919	0920	0921	0922	0923	0924	0925	0926	0927
3A	0928	0929	0930	0931	0932	0933	0934	0935	0936	0937	0938	0939	0940	0941	0942	0943
3B	0944	0945	0946	0947	0948	0949	0950	0951	0952	0953	0954	0955	0956	0957	0958	0959
3C	0960	0961	0962	0963	0964	0965	0966	0967	0968	0969	0970	0971	0972	0973	0974	0975
3D	0976	0977	0978	0979	0980	0981	0982	0983	0984	0985	0986	0987	0988	0989	0990	0991
3E	0992	0993	0994	0995	0996	0997	0998	0999	1000	1001	1002	1003	1004	1005	1006	1007
3F	1008	1009	1010	1011	1012	1013	1014	1015	1016	1017	1018	1019	1020	1021	1022	1023

APPENDIX F Cont'd.

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
40	1024	1025	1026	1027	1028	1029	1030	1031	1032	1033	1034	1035	1036	1037	1038	1039
41	1040	1041	1042	1043	1044	1045	1046	1047	1048	1049	1050	1051	1052	1053	1054	1055
42	1056	1057	1058	1059	1060	1061	1062	1063	1064	1065	1066	1067	1068	1069	1070	1071
43	1072	1073	1074	1075	1076	1077	1078	1079	1080	1081	1082	1083	1084	1085	1086	1087
44	1088	1089	1090	1091	1092	1093	1094	1095	1096	1097	1098	1099	1100	1101	1102	1103
45	1104	1105	1106	1107	1108	1109	1110	1111	1112	1113	1114	1115	1116	1117	1118	1119
46	1120	1121	1122	1123	1124	1125	1126	1127	1128	1129	1130	1131	1132	1133	1134	1135
47	1136	1137	1138	1139	1140	1141	1142	1143	1144	1145	1146	1147	1148	1149	1150	1151
48	1152	1153	1154	1155	1156	1157	1158	1159	1160	1161	1162	1163	1164	1165	1166	1167
49	1168	1169	1170	1171	1172	1173	1174	1175	1176	1177	1178	1179	1180	1181	1182	1183
4A	1184	1185	1186	1187	1188	1189	1190	1191	1192	1193	1194	1195	1196	1197	1198	1199
4B	1200	1201	1202	1203	1204	1205	1206	1207	1208	1209	1210	1211	1212	1213	1214	1215
4C	1216	1217	1218	1219	1220	1221	1222	1223	1224	1225	1226	1227	1228	1229	1230	1231
4D	1232	1233	1234	1235	1236	1237	1238	1239	1240	1241	1242	1243	1244	1245	1246	1247
4E	1248	1249	1250	1251	1252	1253	1254	1255	1256	1257	1258	1259	1260	1261	1262	1263
4F	1264	1265	1266	1267	1268	1269	1270	1271	1272	1273	1274	1275	1276	1277	1278	1279

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
50	1280	1281	1282	1283	1284	1285	1286	1287	1288	1289	1290	1291	1292	1293	1294	1295
51	1296	1297	1298	1299	1300	1301	1302	1303	1304	1305	1306	1307	1308	1309	1310	1311
52	1312	1313	1314	1315	1316	1317	1318	1319	1320	1321	1322	1323	1324	1325	1326	1327
53	1328	1329	1330	1331	1332	1333	1334	1335	1336	1337	1338	1339	1340	1341	1342	1343
54	1344	1345	1346	1347	1348	1349	1350	1351	1352	1353	1354	1355	1356	1357	1358	1359
55	1360	1361	1362	1363	1364	1365	1366	1367	1368	1369	1370	1371	1372	1373	1374	1375
56	1376	1377	1378	1379	1380	1381	1382	1383	1384	1385	1386	1387	1388	1389	1390	1391
57	1392	1393	1394	1395	1396	1397	1398	1399	1400	1401	1402	1403	1404	1405	1406	1407
58	1408	1409	1410	1411	1412	1413	1414	1415	1416	1417	1418	1419	1420	1421	1422	1423
59	1424	1425	1426	1427	1428	1429	1430	1431	1432	1433	1434	1435	1436	1437	1438	1439
5A	1440	1441	1442	1443	1444	1445	1446	1447	1448	1449	1450	1451	1452	1453	1454	1455
5B	1456	1457	1458	1459	1460	1461	1462	1463	1464	1465	1466	1467	1468	1469	1470	1471
5C	1472	1473	1474	1475	1476	1477	1478	1479	1480	1481	1482	1483	1484	1485	1486	1487
5D	1488	1489	1490	1491	1492	1493	1494	1495	1496	1497	1498	1499	1500	1501	1502	1503
5E	1504	1505	1506	1507	1508	1509	1510	1511	1512	1513	1514	1515	1516	1517	1518	1519
5F	1520	1521	1522	1523	1524	1525	1526	1527	1528	1529	1530	1531	1532	1533	1534	1535

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
60	1536	1537	1538	1539	1540	1541	1542	1543	1544	1545	1546	1547	1548	1549	1550	1551
61	1552	1553	1554	1555	1556	1557	1558	1559	1560	1561	1562	1563	1564	1565	1566	1567
62	1568	1569	1570	1571	1572	1573	1574	1575	1576	1577	1578	1579	1580	1581	1582	1583
63	1584	1585	1586	1587	1588	1589	1590	1591	1592	1593	1594	1595	1596	1597	1598	1599
64	1600	1601	1602	1603	1604	1605	1606	1607	1608	1609	1610	1611	1612	1613	1614	1615
65	1616	1617	1618	1619	1620	1621	1622	1623	1624	1625	1626	1627	1628	1629	1630	1631
66	1632	1633	1634	1635	1636	1637	1638	1639	1640	1641	1642	1643	1644	1645	1646	1647
67	1648	1649	1650	1651	1652	1653	1654	1655	1656	1657	1658	1659	1660	1661	1662	1663
68	1664	1665	1666	1667	1668	1669	1670	1671	1672	1673	1674	1675	1676	1677	1678	1679
69	1680	1681	1682	1683	1684	1685	1686	1687	1688	1689	1690	1691	1692	1693	1694	1695
6A	1696	1697	1698	1699	1700	1701	1702	1703	1704	1705	1706	1707	1708	1709	1710	1711
6B	1712	1713	1714	1715	1716	1717	1718	1719	1720	1721	1722	1723	1724	1725	1726	1727
6C	1728	1729	1730	1731	1732	1733	1734	1735	1736	1737	1738	1739	1740	1741	1742	1743
6D	1744	1745	1746	1747	1748	1749	1750	1751	1752	1753	1754	1755	1756	1757	1758	1759
6E	1760	1761	1762	1763	1764	1765	1766	1767	1768	1769	1770	1771	1772	1773	1774	1775
6F	1776	1777	1778	1779	1780	1781	1782	1783	1784	1785	1786	1787	1788	1789	1790	1791

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
70	1792	1793	1794	1795	1796	1797	1798	1799	1800	1801	1802	1803	1804	1805	1806	1807
71	1808	1809	1810	1811	1812	1813	1814	1815	1816	1817	1818	1819	1820	1821	1822	1823
72	1824	1825	1826	1827	1828	1829	1830	1831	1832	1833	1834	1835	1836	1837	1838	1839
73	1840	1841	1842	1843	1844	1845	1846	1847	1848	1849	1850	1851	1852	1853	1854	1855
74	1856	1857	1858	1859	1860	1861	1862	1863	1864	1865	1866	1867	1868	1869	1870	1871
75	1872	1873	1874	1875	1876	1877	1878	1879	1880	1881	1882	1883	1884	1885	1886	1887
76	1888	1889	1890	1891	1892	1893	1894	1895	1896	1897	1898	1899	1900	1901	1902	1903
77	1904	1905	1906	1907	1908	1909	1910	1911	1912	1913	1914	1915	1916	1917	1918	1919
78	1920	1921	1922	1923	1924	1925	1926	1927	1928	1929	1930	1931	1932	1933	1934	1935
79	1936	1937	1938	1939	1940	1941	1942	1943	1944	1945	1946	1947	1948	1949	1950	1951
7A	1952	1953	1954	1955	1956	1957	1958	1959	1960	1961	1962	1963	1964	1965	1966	1967
7B	1968	1969	1970	1971	1972	1973	1974	1975	1976	1977	1978	1979	1980	1981	1982	1983
7C	1984	1985	1986	1987	1988	1989	1990	1991	1992	1993	1994	1995	1996	1997	1998	1999
7D	2000	2001	2002	2003	2004	2005	2006	2007	2008	2009	2010	2011	2012	2013	2014	2015
7E	2016	2017	2018	2019	2020	2021	2022	2023	2024	2025	2026	2027	2028	2029	2030	2031
7F	2032	2033	2034	2035	2036	2037	2038	2039	2040	2041	2042	2043	2044	2045	2046	2047

APPENDIX F Cont'd.

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
80	2048	2049	2050	2051	2052	2053	2054	2055	2056	2057	2058	2059	2060	2061	2062	2063
81	2064	2065	2066	2067	2068	2069	2070	2071	2072	2073	2074	2075	2076	2077	2078	2079
82	2080	2081	2082	2083	2084	2085	2086	2087	2088	2089	2090	2091	2092	2093	2094	2095
83	2096	2097	2098	2099	2100	2101	2102	2103	2104	2105	2106	2107	2108	2109	2110	2111
84	2112	2113	2114	2115	2116	2117	2118	2119	2120	2121	2122	2123	2124	2125	2126	2127
85	2128	2129	2130	2131	2132	2133	2134	2135	2136	2137	2138	2139	2140	2141	2142	2143
86	2144	2145	2146	2147	2148	2149	2150	2151	2152	2153	2154	2155	2156	2157	2158	2159
87	2160	2161	2162	2163	2164	2165	2166	2167	2168	2169	2170	2171	2172	2173	2174	2175
88	2176	2177	2178	2179	2180	2181	2182	2183	2184	2185	2186	2187	2188	2189	2190	2191
89	2192	2193	2194	2195	2196	2197	2198	2199	2200	2201	2202	2203	2204	2205	2206	2207
8A	2208	2209	2210	2211	2212	2213	2214	2215	2216	2217	2218	2219	2220	2221	2222	2223
8B	2224	2225	2226	2227	2228	2229	2230	2231	2232	2233	2234	2235	2236	2237	2238	2239
8C	2240	2241	2242	2243	2244	2245	2246	2247	2248	2249	2250	2251	2252	2253	2254	2255
8D	2256	2257	2258	2259	2260	2261	2262	2263	2264	2265	2266	2267	2268	2269	2270	2271
8E	2272	2273	2274	2275	2276	2277	2278	2279	2280	2281	2282	2283	2284	2285	2286	2287
8F	2288	2289	2290	2291	2292	2293	2294	2295	2296	2297	2298	2299	2300	2301	2302	2303

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
90	2304	2305	2306	2307	2308	2309	2310	2311	2312	2313	2314	2315	2316	2317	2318	2319
91	2320	2321	2322	2323	2324	2325	2326	2327	2328	2329	2330	2331	2332	2333	2334	2335
92	2336	2337	2338	2339	2340	2341	2342	2343	2344	2345	2346	2347	2348	2349	2350	2351
93	2352	2353	2354	2355	2356	2357	2358	2359	2360	2361	2362	2363	2364	2365	2366	2367
94	2368	2369	2370	2371	2372	2373	2374	2375	2376	2377	2378	2379	2380	2381	2382	2383
95	2384	2385	2386	2387	2388	2389	2390	2391	2392	2393	2394	2395	2396	2397	2398	2399
96	2400	2401	2402	2403	2404	2405	2406	2407	2408	2409	2410	2411	2412	2413	2414	2415
97	2416	2417	2418	2419	2420	2421	2422	2423	2424	2425	2426	2427	2428	2429	2430	2431
98	2432	2433	2434	2435	2436	2437	2438	2439	2440	2441	2442	2443	2444	2445	2446	2447
99	2448	2449	2450	2451	2452	2453	2454	2455	2456	2457	2458	2459	2460	2461	2462	2463
9A	2464	2465	2466	2467	2468	2469	2470	2471	2472	2473	2474	2475	2476	2477	2478	2479
9B	2480	2481	2482	2483	2484	2485	2486	2487	2488	2489	2490	2491	2492	2493	2494	2495
9C	2496	2497	2498	2499	2500	2501	2502	2503	2504	2505	2506	2507	2508	2509	2510	2511
9D	2512	2513	2514	2515	2516	2517	2518	2519	2520	2521	2522	2523	2524	2525	2526	2527
9E	2528	2529	2530	2531	2532	2533	2534	2535	2536	2537	2538	2539	2540	2541	2542	2543
9F	2544	2545	2546	2547	2548	2549	2550	2551	2552	2553	2554	2555	2556	2557	2558	2559

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
A0	2560	2561	2562	2563	2564	2565	2566	2567	2568	2569	2570	2571	2572	2573	2574	2575
A1	2576	2577	2578	2579	2580	2581	2582	2583	2584	2585	2586	2587	2588	2589	2590	2591
A2	2592	2593	2594	2595	2596	2597	2598	2599	2600	2601	2602	2603	2604	2605	2606	2607
A3	2608	2609	2610	2611	2612	2613	2614	2615	2616	2617	2618	2619	2620	2621	2622	2623
A4	2624	2625	2626	2627	2628	2629	2630	2631	2632	2633	2634	2635	2636	2637	2638	2639
A5	2640	2641	2642	2643	2644	2645	2646	2647	2648	2649	2650	2651	2652	2653	2654	2655
A6	2656	2657	2658	2659	2660	2661	2662	2663	2664	2665	2666	2667	2668	2669	2670	2671
A7	2672	2673	2674	2675	2676	2677	2678	2679	2680	2681	2682	2683	2684	2685	2686	2687
A8	2688	2689	2690	2691	2692	2693	2694	2695	2696	2697	2698	2699	2700	2701	2702	2703
A9	2704	2705	2706	2707	2708	2709	2710	2711	2712	2713	2714	2715	2716	2717	2718	2719
AA	2720	2721	2722	2723	2724	2725	2726	2727	2728	2729	2730	2731	2732	2733	2734	2735
AB	2736	2737	2738	2739	2740	2741	2742	2743	2744	2745	2746	2747	2748	2749	2750	2751
AC0	2752	2753	2754	2755	2756	2757	2758	2759	2760	2761	2762	2763	2764	2765	2766	2767
AD0	2768	2769	2770	2771	2772	2773	2774	2775	2776	2777	2778	2779	2780	2781	2782	2783
AE0	2784	2785	2786	2787	2788	2789	2790	2791	2792	2793	2794	2795	2796	2797	2798	2799
AF0	2800	2801	2802	2803	2804	2805	2806	2807	2808	2809	2810	2811	2812	2813	2814	2815

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
B0	2816	2817	2818	2819	2820	2821	2822	2823	2824	2825	2826	2827	2828	2829	2830	2831
B1	2832	2833	2834	2835	2836	2837	2838	2839	2840	2841	2842	2843	2844	2845	2846	2847
B2	2848	2849	2850	2851	2852	2853	2854	2855	2856	2857	2858	2859	2860	2861	2862	2863
B3	2864	2865	2866	2867	2868	2869	2870	2871	2872	2873	2874	2875	2876	2877	2878	2879
B4	2880	2881	2882	2883	2884	2885	2886	2887	2888	2889	2890	2891	2892	2893	2894	2895
B5	2896	2897	2898	2899	2900	2901	2902	2903	2904	2905	2906	2907	2908	2909	2910	2911
B6	2912	2913	2914	2915	2916	2917	2918	2919	2920	2921	2922	2923	2924	2925	2926	2927
B7	2928	2929	2930	2931	2932	2933	2934	2935	2936	2937	2938	2939	2940	2941	2942	2943
B8	2944	2945	2946	2947	2948	2949	2950	2951	2952	2953	2954	2955	2956	2957	2958	2959
B9	2960	2961	2962	2963	2964	2965	2966	2967	2968	2969	2970	2971	2972	2973	2974	2975
BA	2976	2977	2978	2979	2980	2981	2982	2983	2984	2985	2986	2987	2988	2989	2990	2991
BB	2992	2993	2994	2995	2996	2997	2998	2999	3000	3001	3002	3003	3004	3005	3006	3007
BC	3008	3009	3010	3011	3012	3013	3014	3015	3016	3017	3018	3019	3020	3021	3022	3023
BD	3024	3025	3026	3027	3028	3029	3030	3031	3032	3033	3034	3035	3036	3037	3038	3039
BE	3040	3041	3042	3043	3044	3045	3046	3047	3048	3049	3050	3051	3052	3053	3054	3055
BF	3056	3057	3058	3059	3060	3061	3062	3063	3064	3065	3066	3067	3068	3069	3070	3071

APPENDIX F Cont'd.

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
C0	3072	3073	3074	3075	3076	3077	3078	3079	3080	3081	3082	3083	3084	3085	3086	3087
C1	3088	3089	3090	3091	3092	3093	3094	3095	3096	3097	3098	3099	3100	3101	3102	3103
C2	3104	3105	3106	3107	3108	3109	3110	3111	3112	3113	3114	3115	3116	3117	3118	3119
C3	3120	3121	3122	3123	3124	3125	3126	3127	3128	3129	3130	3131	3132	3133	3134	3135
C4	3136	3137	3138	3139	3140	3141	3142	3143	3144	3145	3146	3147	3148	3149	3150	3151
C5	3152	3153	3154	3155	3156	3157	3158	3159	3160	3161	3162	3163	3164	3165	3166	3167
C6	3168	3169	3170	3171	3172	3173	3174	3175	3176	3177	3178	3179	3180	3181	3182	3183
C7	3184	3185	3186	3187	3188	3189	3190	3191	3192	3193	3194	3195	3196	3197	3198	3199
C8	3200	3201	3202	3203	3204	3205	3206	3207	3208	3209	3210	3211	3212	3213	3214	3215
C9	3216	3217	3218	3219	3220	3221	3222	3223	3224	3225	3226	3227	3228	3229	3230	3231
CA	3232	3233	3234	3235	3236	3237	3238	3239	3240	3241	3242	3243	3244	3245	3246	3247
CB	3248	3249	3250	3251	3252	3253	3254	3255	3256	3257	3258	3259	3260	3261	3262	3263
CC	3264	3265	3266	3267	3268	3269	3270	3271	3272	3273	3274	3275	3276	3277	3278	3279
CD	3280	3281	3282	3283	3284	3285	3286	3287	3288	3289	3290	3291	3292	3293	3294	3295
CE	3296	3297	3298	3299	3300	3301	3302	3303	3304	3305	3306	3307	3308	3309	3310	3311
CF	3312	3313	3314	3315	3316	3317	3318	3319	3320	3321	3322	3323	3324	3325	3326	3327

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
D0	3328	3329	3330	3331	3332	3333	3334	3335	3336	3337	3338	3339	3340	3341	3342	3343
D1	3344	3345	3346	3347	3348	3349	3350	3351	3352	3353	3354	3355	3356	3357	3358	3359
D2	3360	3361	3362	3363	3364	3365	3366	3367	3368	3369	3370	3371	3372	3373	3374	3375
D3	3376	3377	3378	3379	3380	3381	3382	3383	3384	3385	3386	3387	3388	3389	3390	3391
D4	3392	3393	3394	3395	3396	3397	3398	3399	3400	3401	3402	3403	3404	3405	3406	3407
D5	3408	3409	3410	3411	3412	3413	3414	3415	3416	3417	3418	3419	3420	3421	3422	3423
D6	3424	3425	3426	3427	3428	3429	3430	3431	3432	3433	3434	3435	3436	3437	3438	3439
D7	3440	3441	3442	3443	3444	3445	3446	3447	3448	3449	3450	3451	3452	3453	3454	3455
D8	3456	3457	3458	3459	3460	3461	3462	3463	3464	3465	3466	3467	3468	3469	3470	3471
D9	3472	3473	3474	3475	3476	3477	3478	3479	3480	3481	3482	3483	3484	3485	3486	3487
DA	3488	3489	3490	3491	3492	3493	3494	3495	3496	3497	3498	3499	3500	3501	3502	3503
DB	3504	3505	3506	3507	3508	3509	3510	3511	3512	3513	3514	3515	3516	3517	3518	3519
DC	3520	3521	3522	3523	3524	3525	3526	3527	3528	3529	3530	3531	3532	3533	3534	3535
DD	3536	3537	3538	3539	3540	3541	3542	3543	3544	3545	3546	3547	3548	3549	3550	3551
DE	3552	3553	3554	3555	3556	3557	3558	3559	3560	3561	3562	3563	3564	3565	3566	3567
DF	3568	3569	3570	3571	3572	3573	3574	3575	3576	3577	3578	3579	3580	3581	3582	3583

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
E0	3584	3585	3586	3587	3588	3589	3590	3591	3592	3593	3594	3595	3596	3597	3598	3599
E1	3600	3601	3602	3603	3604	3605	3606	3607	3608	3609	3610	3611	3612	3613	3614	3615
E2	3616	3617	3618	3619	3620	3621	3622	3623	3624	3625	3626	3627	3628	3629	3630	3631
E3	3632	3633	3634	3635	3636	3637	3638	3639	3640	3641	3642	3643	3644	3645	3646	3647
E4	3648	3649	3650	3651	3652	3653	3654	3655	3656	3657	3658	3659	3660	3661	3662	3663
E5	3664	3665	3666	3667	3668	3669	3670	3671	3672	3673	3674	3675	3676	3677	3678	3679
E6	3680	3681	3682	3683	3684	3685	3686	3687	3688	3689	3690	3691	3692	3693	3694	3695
E7	3696	3697	3698	3699	3700	3701	3702	3703	3704	3705	3706	3707	3708	3709	3710	3711
E8	3712	3713	3714	3715	3716	3717	3718	3719	3720	3721	3722	3723	3724	3725	3726	3727
E9	3728	3729	3730	3731	3732	3733	3734	3735	3736	3737	3738	3739	3740	3741	3742	3743
EA	3744	3745	3746	3747	3748	3749	3750	3751	3752	3753	3754	3755	3756	3757	3758	3759
EB	3760	3761	3762	3763	3764	3765	3766	3767	3768	3769	3770	3771	3772	3773	3774	3775
EC	3776	3777	3778	3779	3780	3781	3782	3783	3784	3785	3786	3787	3788	3789	3790	3791
ED	3792	3793	3794	3795	3796	3797	3798	3799	3800	3801	3802	3803	3804	3805	3806	3807
EE	3808	3809	3810	3811	3812	3813	3814	3815	3816	3817	3818	3819	3820	3821	3822	3823
EF	3824	3825	3826	3827	3828	3829	3830	3831	3832	3833	3834	3835	3836	3837	3838	3839

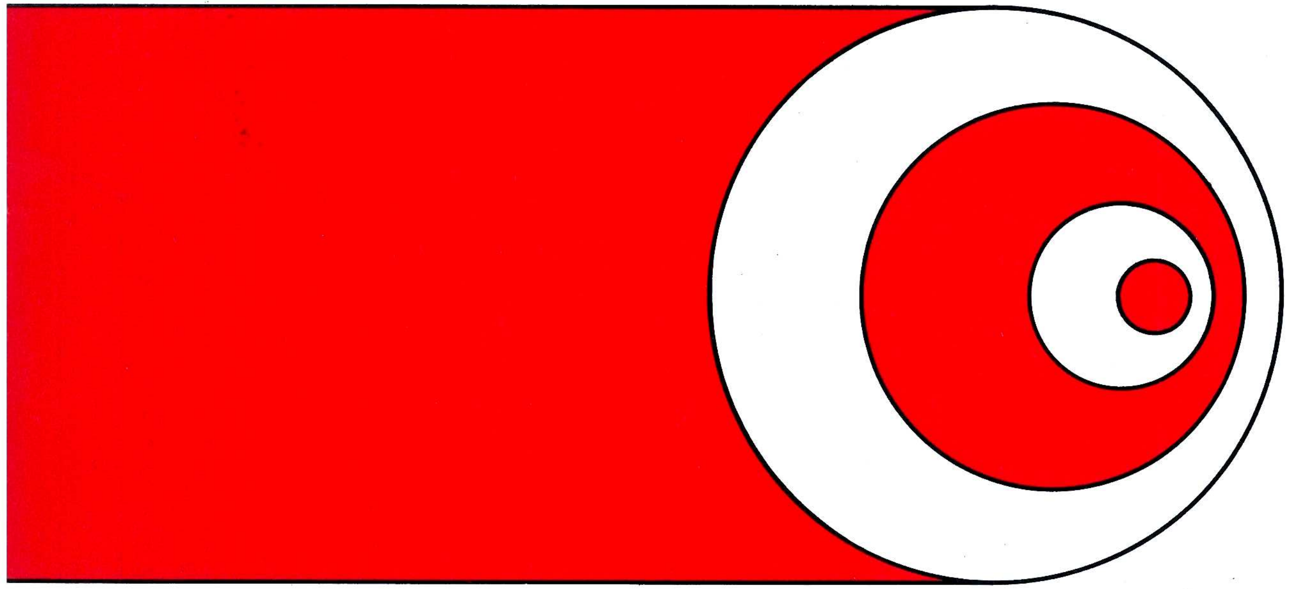
	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
F0	3840	3841	3842	3843	3844	3845	3846	3847	3848	3849	3850	3851	3852	3853	3854	3855
F1	3856	3857	3858	3859	3860	3861	3862	3863	3864	3865	3866	3867	3868	3869	3870	3871
F2	3872	3873	3874	3875	3876	3877	3878	3879	3880	3881	3882	3883	3884	3885	3886	3887
F3	3888	3889	3890	3891	3892	3893	3894	3895	3896	3897	3898	3899	3900	3901	3902	3903
F4	3904	3905	3906	3907	3908	3909	3910	3911	3912	3913	3914	3915	3916	3917	3918	3919
F5	3920	3921	3922	3923	3924	3925	3926	3927	3928	3929	3930	3931	3932	3933	3934	3935
F6	3936	3937	3938	3939	3940	3941	3942	3943	3944	3945	3946	3947	3948	3949	3950	3951
F7	3952	3953	3954	3955	3956	3957	3958	3959	3960	3961	3962	3963	3964	3965	3966	3967
F8	3968	3969	3970	3971	3972	3973	3974	3975	3976	3977	3978	3979	3980	3981	3982	3983
F9	3984	3985	3986	3987	3988	3989	3990	3991	3992	3993	3994	3995	3996	3997	3998	3999
FA	4000	4001	4002	4003	4004	4005	4006	4007	4008	4009	4010	4011	4012	4013	4014	4015
FB	4016	4017	4018	4019	4020	4021	4022	4023	4024	4025	4026	4027	4028	4029	4030	4031
FC	4032	4033	4034	4035	4036	4037	4038	4039	4040	4041	4042	4043	4044	4045	4046	4047
FD	4048	4049	4050	4051	4052	4053	4054	4055	4056	4057	4058	4059	4060	4061	4062	4063
FE	4064	4065	4066	4067	4068	4069	4070	4071	4072	4073	4074	4075	4076	4077	4078	4079
FF	4080	4081	4082	4083	4084	4085	4086	4087	4088	4089	4090	4091	4092	4093	4094	4095

NOTES NOTES NOTES NOTES NOTES NOTES NOTES NOTES NOTES NOTES NOTES NOTES NOTES NOTES

NOTES NOTES NOTES NOTES NOTES NOTES NOTES NOTES NOTES NOTES NOTES NOTES NOTES

SIGNETICS
2650 MICROPROCESSOR

SOFTWARE SIMULATOR



2650 SIMULATOR MANUAL

CONTENTS

I INTRODUCTION

The 2650 Simulator is a FORTRAN program which allows a user to simulate the execution of his program without utilizing the 2650 processor.

The Simulator executes a 2650 program by maintaining its own internal FORTRAN storage registers to describe the 2650 program itself, the microprocessor registers, the ROM/RAM memory configuration, and the input data to be read dynamically from I/O devices. Multiple simulations of the same program may be executed during a single simulation run. In addition, statistical timing information may be generated.

The Simulator requires as input both the program object module produced by the 2650 Assembler and a deck of user commands. It produces a listing of the user's commands, executes the program and prints ("displays") both static and dynamic information as requested by the user's commands.

II SIMULATOR OPERATION

GENERAL

Once the Simulator is loaded and started, it performs the following actions:

- Presets each register in simulated memory to a "HALT" instruction. Thus, if the user's program attempts to branch to some undefined area of memory, the current execution of the simulated program is terminated and only relevant data is printed.
- Reads and stores the user's commands. These commands control the performance of the Simulator during program execution. They are stored in a simulator table for reference before, during, and after execution.
- Loads the 2650 object module into simulated memory.
- Starts the simulated program. The simulated program is started at the address specified in the START command. If no START command is submitted, the program is started in the location specified in the END statement of the simulated program (see Assembler manual). If no location is specified in the END statement, the Simulator starts in location 0.
- Oversees the execution of each instruction. Before an instruction is executed, the Simulator checks the address of the instruction and the address of the referenced memory location to see if either of these addresses is referenced by any one of the user's commands. If so, the command is executed. The Simulator then executes the current instruction, updates all affected registers and retrieves the next instruction for execution.
- Terminates the simulated program. The simulation is terminated either by the execution of a "HALT" instruction, or by having executed a preset number of instructions or by having satisfied the conditions of the STOP. command.
- Once the execution of one simulation is complete, the Simulator prints any statistical timing information requested (STAT), and proceeds with the next simulation (TEND) or terminates itself (FEND).

SIMULATED PROCESSOR STATE

The Simulator maintains a number of FORTRAN integer cells which are used to simulate the microprocessor's state, i.e. the general purpose registers, the upper and lower program status bytes, the location counter or instruction address register (IAR), the address of the instruction referenced and the contents of the location referenced.

These simulated registers and status bits may be displayed dynamically, (INSTR., REFER., TRACE.) i.e., while the simulated program is executing. Also the general purpose registers and the status bytes may be altered dynamically (SETR., SETP.).

SIMULATED MEMORY

The Simulator maintains a 2048 cell FORTRAN integer array which is used to simulate read-write random access memory.

It is possible to configure parts of this memory into a ROM-RAM environment by using the SROM Command. If part of the simulated memory is set to Read-Only and an instruction attempts to store data into that memory segment, the Simulator bypasses storing the data, prints a warning message and continues with the next program instruction.

Using Simulator commands, the user may change parts of memory before the program executes (PATCH) and he may display parts of memory dynamically (DUMP.).

The simulated memory is smaller in many cases than the total memory size of the user's physical system. This restriction encourages the construction of modular programs. Because the simulated memory is smaller than a 2650 page, it is not possible to fully test programs which utilize the 2650 paging system, i.e., programs larger than 8192 bytes.

SIMULATED INPUT/OUTPUT INSTRUCTIONS

The Simulator maintains a 200-byte First In, First Out (FIFO) buffer to store the data read from a simulated input device. This buffer must be preset by the user command, INPUT.

When any 2650 input instruction is simulated (REDE, REDC, REDD), the Simulator accesses the buffer. If there is data in the buffer, the next byte of data is inserted in the simulated register specified by the input instruction. If the buffer contents have been exhausted, a warning message is displayed on the simulator listing.

To simulate the execution of any 2650 output instruction (WRTE, WRTC, WRTD), the Simulator takes the data byte from the register specified in the output instruction and displays it along with the address of the output instruction.

III USER COMMANDS

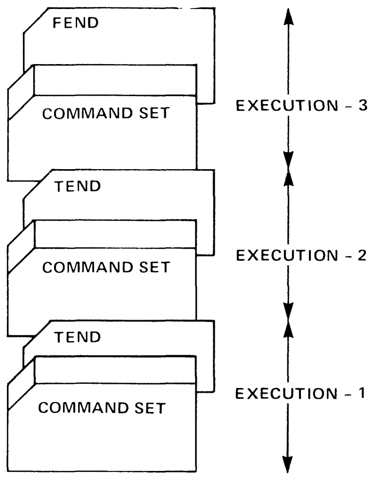
GENERAL

The 2650 Simulator accepts commands which specify how the program is to run and what data is to be recorded.

In any one Simulator run, the user may specify that his program be executed any number of times. The user submits a new set of commands for each execution. The final command set is followed by a final end card (FEND), while all prior command sets are terminated with a temporary end card (TEND) (Illust. III-1).

ILLUSTRATION III-1

THREE SETS OF COMMANDS



Within any one command set, the user may specify:

- That the program execution start at a specific memory location (START).
- That the execution of the program be complete either when the number of instructions executed equals a specified number (LIMIT) or when the instruction at a specific address executes (STOP.) or when the simulated program itself executes a "HALT" instruction.
- That statistics be displayed at the end of execution (STAT). The Simulator accumulates a count of the total number of instructions executed, the number of each type of instruction executed, and the total number of 2650 machine cycles expended. This information provides a measure of efficiency by indicating how many 1-, 2-, or 3-byte instructions were executed and may be used to calculate program timings.
- That certain areas of simulated memory be designated as Read-Only (SROM) and are therefore inaccessible to any memory write operation.
- That the contents of memory be initialized with specific data (PATCH).
- That a FIFO (First In, First Out) buffer be used to simulate data read from I/O devices (INPUT).
- That the processor state be recorded whenever a specific memory location executes (INSTR.), whenever a specific memory location is referenced (REFER.), or whenever any instruction executes which lies within a specified range of memory addresses (TRACE.). The processor state consists of the location counter, the instruction referenced and its contents, the upper and the lower program status bytes, and the contents of all the general purpose registers.

- That an area of memory be dumped whenever an instruction at a specific memory location executes (DUMP.).
- That certain general purpose registers (SETR.) or the program status bytes (SETP.) be set dynamically, i.e., whenever a specific memory location executes.
- That comments (**) be interspersed between control cards.

Some of these commands execute dynamically, i.e., when an instruction at a specific memory location executes or when that location is referenced. Since the simulator storage capacity limits the total number of locations which may be retained simultaneously (while a program is executing), a total of 30 memory locations may be specified on all the "dynamic" commands submitted for any one execution, i.e., in any one command set. These dynamic commands are identified by a trailing period (.), e.g., "STOP.". This period is treated as a field separator, i.e., it is not treated as part of the command name by the Simulator and is therefore optional. The description for each dynamic command identifies which of its parameters count toward the 30 "dynamic" command limit, i.e., the limit of 30 memory locations.

In addition, the number of DUMP. commands is limited to five (5); the number of SETR. commands is limited to four (4); the number of SETP. commands is limited to two (2); and the number of data read on all INPUT cards in one command set is limited to 200.

All "dynamic commands" are executed *before* the simulated instruction is executed.

For those commands which accept only one set of parameters (LIMIT, SROM, START) only the last set of parameters encountered is used.

COMMAND FORMATS

Illustration III-2 contains a list of the commands, their parameters and a brief description of the commands themselves. In addition, the Simulator treats as a comment card, any card with two consecutive asterisks (**) starting in column 1.

The Simulator accepts information in card image form. The entire card is read in FORTRAN "A" format. A command must be complete on one card as continuation cards are not allowed. Comments may appear in any order within a command set.

The command name starts in column 1 and must appear as shown, except for the optional period.

The field of characters which lies between the command name and its parameters or between the parameters themselves is called a field separator. A field separator may contain any number of characters, but none of these characters may be hexadecimal characters (0-9, A-F). For the sake of clarity in all the examples, the following field separators are used to indicate the following functions:

Field Separator	Function
	Identifies a command which counts toward the "dynamic" command limit.
blank (s)	Separate a command from its parameters.
( )	Encloses optional parameters.
;	Separates one set of parameters from another.
,	Separates one parameter from another within a set of parameters.
; . . . ;	Indicates that multiple parameters or sets of parameters are legal. If a period flags a command, each of its parameter sets counts toward the "dynamic" command limit. E.g., the following sets of commands are identical: 1. INST. 100 INST. 200 2. INST. 100; 200

The parameters themselves must be hexadecimal numbers (0-9, A-F). The following labels identify parameters in Illustration III-2:

LOC	Location or address of an instruction which is to be executed or the address of data which is to be referenced.
NO	A number of data, e.g., the total number of instructions to be executed.
FWA	First Word Address of some area of memory.
LWA	Last Word Address of some area of memory.
VALUE	The value to which some location is to be set.
R0, R1 . . . R6	General Purpose Registers 0-6.
PSL	Identifies Lower Program Status Byte.
PSU	Identifies Upper Program Status Byte.

ILLUSTRATION III-2
COMMAND SUMMARY

COMMAND NAME	PARAMETERS	DESCRIPTION
DUMP.	LOC, FWA-LWA (; . . . ;LOC, FWA-LWA)	Display the area of memory, FWA-LWA, whenever the instruction at LOC executes.
FEND	None	Execute the last simulation and terminate the entire run.
INPUT	VALUE(; . . . ;VALUE)	Define the data to be read by simulated I/O instructions.
INSTR.	LOC(; . . . ;LOC)	Display the processor registers whenever the instruction at LOC executes.
LIMIT	NO	Specify the total number of instructions executed.
PATCH	LOC,VALUE(; . . . ;LOC,VALUE)	Initialize each memory location, LOC, to VALUE.
REFER.	LOC(; . . . ;LOC)	Display the processor register whenever the ir struction at LOC is referenced by another instruction.
SETP.	LOC(,PSL=VALUE) (,PSU=VALUE)	Set the program status byte (lower and/or upper) to VALUE whenever the instruction at LOC executes.
SETR.	LOC(,R0=VALUE). . .(R6=VALUE)	Set the general purpose registers to VALUE whenever the instruction at LOC executes.
SROM	FWA-LWA	Specify the boundaries of Read-Only Memory.
START	LOC	Start the simulated program execution at LOC.
STAT	None	Display instruction statistics at end of program execution.
STOP.	LOC(; . . . ;LOC)	Terminate the program execution when the instruction at LOC executes.
TEND	None	Execute the last simulation and prepare to read the User Commands for the next simulatior
TRACE.	FWA-LWA(; . . . ;FWA-LWA)	Display the processor registers whenever an instruction executes, which lies within the area of memory, FWA-LWA.

COMMAND DESCRIPTIONS

The following command descriptions are alphabetized by command name. As previously discussed all parameters are entered in hexadecimal notation (0-9, A-F). All address parameters (LOC, FWA, LWA) are limited to the size of simulated memory.

DUMP. DUMP SIMULATED MEMORY

This command causes the Simulator to display selected portions of memory whenever the location counter matches LOC.

Each LOC counts as one "dynamic" command. The total number of "dynamic" commands is limited to thirty (30). The total number of LOC's submitted in DUMP. commands is limited to five (5).

```
DUMP.    LOC,FWA-LWA(; . . . ;LOC,FWA-LWA)
```

Where: DUMP. is the command name.

LOC is the address of the 2650 instruction at which the dump occurs.

FWA is the first address of the area to be dumped.

LWA is the last address of the area to be dumped. LWA must be larger than FWA.

Example:

```
DUMP.    5A,0-3FF   100-11A-21A
DUMP.    EO-400-4FF
```

Note: More data may be dumped than was specified since the FWA dumped always has a least significant digit of 0, e.g. 30, 100, etc. Similarly, LWA always has a least significant digit of F, e.g. 3F, 10F, etc.

FEND FINAL END COMMAND

This command signals the Simulator that the preceding commands complete the directives for the final simulator run. After FEND is read, the Simulator performs the last simulation and comes to its final termination.

FEND

Where: FEND — specifies the command name.

Example:

```
START    1A
TRACE    0, 100
TEND
START    AA
PATCH    11, C2
FEND
```

INPUT DEFINE DATA FOR INPUT

This command loads data into a FIFO storage buffer from which the same data is used to supply I/O instructions with input data. The first data point specified becomes the first one accessed by a 2650 read instruction. The last point specified becomes the last one accessed. Should the buffer become empty during the simulated execution, an error message is printed, the input register remains unchanged and the simulation continues.

Any number of these command cards may be submitted as long as the total number of data specified in one run does not exceed the size of the FIFO storage buffer (200).

```
INPUT    VALUE(;...;VALUE)
```

Where: INPUT — specifies the command name.

VALUE — specifies a 2-digit hexadecimal value.

Example:

```
INPUT    0, 1, 2, 3, 10, 1A, FF
```

INSTR. INSTRUCTION TRACE

This command sets a break point at the specified address. When the instruction at this address executes, the Simulator prints out the internal state of the simulated processor. The break point occurs before the instruction is executed.

Each address specified in an INSTR. command counts as one "dynamic" command.

```
INSTR.    LOC(; . . . ;LOC)
```

Where:

INSTR. — specifies the command.

LOC — specifies the address for a break point. The address must be within simulated memory.

Example:

```
INSTR.    1CE, 1A, 22
INSTR.    123-200-5E
INSTR.    74
```

LIMIT LIMIT THE NUMBER OF INSTRUCTIONS EXECUTED

This command determines how many instructions will be executed. If the number given in the LIMIT command is exceeded before the instruction specified by a STOP. command executes or before a 2650 HALT instruction is simulated, the Simulator terminates the current program operation.

Without this command, the Simulator assumes a limit of 1000_{10} instructions. The maximum LIMIT which may be specified is determined by the maximum integer constant of the FORTRAN compiler used.

```
LIMIT     NO
```

Where: LIMIT — specifies the command.

NO — is a number which determines the maximum number of instructions to be executed.

Example:

```
LIMIT    200
LIMIT    2F
```

PATCH PATCH SIMULATED MEMORY

This command alters the contents of memory before a simulation run. It may be used to alter the contents of any byte in memory and overrides load information in the object module for the duration of one simulation run.

Any number of these commands may be given in a simulator command stream.

```
PATCH     LOC,VALUE(; . . . ;LOC,VALUE)
```

Where:

PATCH — specifies the command.

LOC — specifies the simulated memory address which is to be changed.

VALUE — specifies a 2-digit hexadecimal number to be stored at LOC.

Example:

```
PATCH     0, 1F   1, 0   2. 5E
PATCH     102, EE
```

REFER. MEMORY REFERENCE TRACE

This command causes a break point to occur whenever one of the specified addresses is referenced by a simulated instruction. During the break point, the Simulator prints out the internal state of the simulated processor. The data byte of immediate addressing instructions is handled like an ordinary operand address.

Each address specified in a REFER. command counts as one "dynamic" command.

```
REFER.    LOC(;LOC. . . ;LOC)
```

Where: REFER. — specifies the command.

LOC — specifies the effective operand address for a break point. The address must be within simulated memory.

Example:

```
REFER.    3FF/21/18E
REFER.    200
REFER.    5, 50, 22F
```

SETP. SET PROGRAM STATUS SYTE

The SETP. command dynamically alters the upper and/or the lower program status bytes. The specified program status byte is set when the address parameter supplied in the command, LOC, equals the location counter.

A SETP. command must set at least one program status byte. Up to two SETP. commands may be given in a simulator command stream. Each LOC submitted counts as one "dynamic" command.

The PSL and PSU may be entered in any order.

```
SETP.    LOC(,PSL=VALUE)   (,PSU=VALUE)
```

Where:

SETP. — specifies the command.

LOC — specifies the simulated execution address where the program status byte is to be set.

PSL — specifies that a value is to be entered into PSL.

PSU — specifies that a value is to be entered into PSU.

VALUE — specifies the 2-digit hexadecimal value to be entered into the program status byte.

Example:

```
SETP.    5A PSL=05
SETP.    10E, PSL=01   PSU=00
```

SETR. SET GENERAL PURPOSE REGISTER

This command dynamically sets the general purpose registers during simulated program execution. Using this command, any or all of the general purpose registers can be set when the location counter value is equal to the address parameter, LOC, supplied in this command.

A SETR. command without parameters is not permitted. Up to four SETR. commands may be given in a simulator command stream. Each LOC counts as one "dynamic" command.

Register identifiers may appear in any order.

```
SETR.   LOC(,R0=VALUE)...(,R6=VALUE)
```

Where:

SETR. — specifies the command.

LOC — specifies the simulated execution address where the registers are to be set.

R0 — indicates the general purpose register to be set. R0
R1 always refers to general purpose register 0. R1, R2, and
R2 R3 specify the registers in register bank zero. R4, R5
R3 and R6 specify R1, R2, and R3 in register bank one.
R4
R5
R6

VALUE — specifies the 2-digit hexadecimal value to be stored in the selected register.

Example:

```
SETR.   10A R1=3F, R2=00, R3=5
SETR.   2F3 R0=FF, R5=00
```

SROM DEFINE THE BOUNDARIES OF READ ONLY MEMORY

This command allows the user to simulate a Read Only/Read Write Memory environment. Whenever a 2650 instruction attempts to store data in the area defined as Read Only, a warning message is printed on the simulation listing. The data is not actually stored, but the simulation run continues.

```
SROM     FWA-LWA
```

Where: SROM — specifies the command.

FWA — specifies the first address of the simulated ROM area.

LWA — specifies the last address of the simulated ROM area. LWA must be greater in value than the FWA. The addresses specified are inclusive.

Example: SROM 100-FF

START START SIMULATION

This command specifies the address at which simulated execution begins. The address specified in the START command supersedes the start address in the load object module. The start address in the load object module is set by an END statement during program assembly and is used by the Simulator if no START command is given (see the 2650 Assembler Language Manual for the END statement).

```
START      LOC
```

Where: START — specifies the command.

LOC — specifies a start address for the program to be simulated.

Example:

```
START   10A
START   2
```

STAT DISPLAY INSTRUCTION STATISTICS

This command causes a list of 2650 instructions with the number of times each was executed to be printed out at the end of the simulation run.

```
STAT
```

Where: STAT — specifies the command.

STOP. STOP SIMULATED EXECUTION

This command terminates the current simulated instruction execution when the location counter matches the command argument, LOC.

Each LOC counts as one "dynamic" command.

```
STOP.     LOC(; . . . ;LOC)
```

Where: STOP. — specifies the command.

LOC — specifies the instruction address at which simulated execution ceases.

TEND TEMPORARY END COMMAND

This command signals the Simulator that the preceding commands complete the directives for a simulator run. After the TEND is read, the Simulator begins simulated execution of the 2650 program. Because TEND is a temporary end, the Simulator assumes that there is another command stream following it. The last command stream in a simulation run must be terminated with a FEND (final end) command.

TEND

Where: TEND — specifies the command.

Example:

```
PATCH    01, 15   0A, FF
TEND
START    100
PATCH    01, E2   0A, FF
FEND
```

TRACE. TRACE PROGRAM FLOW

This command causes break points to occur at each instruction within an area of memory. The user specifies two addresses. If the simulated processor accesses an instruction at an address that falls between the specified addresses, the Simulator prints out the internal state of the simulated processor.

Each set of FWA,LWA counts as one "dynamic" command.

```
TRACE.    FWA-LWA(; . . . ;FWA-LWA)
```

Where:

TRACE. — specifies the command.

FWA — specifies from what address the trace is in effect.

LWA — specifies to what address the trace is in effect. LWA must be larger in value than FWA. The addresses specified are inclusive.

Example:

```
TRACE.   0-15F, 250-3FF
TRACE.   1-A, 3FF-40A
TRACE.   10-1A   50-5A   60-7A
```

IV SIMULATOR DISPLAY (LISTING)

As the Simulator reads each command set, it prints the card images of the command set and then executes the program. During program execution the following commands result in some form of display:

DUMP.
INSTR.
REFER.
TRACE.

DUMP. results in the display of an entire area of memory while the last three commands result in some form of trace, i.e., a display of the processor state:

Instruction address register (IAR) or location counter
Instruction executed (INST)
Instruction referenced or effected (EADDR)
Contents of the instruction referenced or effected (EADDR)
Program status byte upper (PSU)
Program status byte lower (PSL)
General purpose registers (R0, R1, R2, R3, R4, R5, R6)

Illustrations IV-1 through IV-4 contain the printout or display output from one Simulator run. Illustration IV-1 shows the first command set, which contains commands to:

- Start at location 0 (START)
- Initialize locations 55-5F, locations 61-6B and location 19 (PATCH)
- Dump locations 55-77 whenever either location 0 or location 3 executes (DUMP)
- Trace locations 14-1A (TRACE)

Illustrations IV-1 and IV-2 show the results of the first command set:

- A dump of locations 55-77. Note that a larger area is dumped than was specified.
- 30 traces
- A final dump of locations 55-77

When the program execution for the first command set is complete, the Simulator reports:

- The number of machine cycles executed
- The number of instructions executed

Illustration IV-3 shows the second command set. It is exactly the same as the first command set except that it initializes locations 12 and 33 instead of location 19.

The output of the second command set is just like the output of the first command set except that it results in 33 traces, not 30.

ILLUSTRATION IV-1

```
START  00
PATCH 55,0 56,1 57,2 58,2 59,3
PATCH 5A,5 5B,4 5C,3 5D,2 5E,1
PATCH  5F,0
PATCH 61,0
PATCH 62,0 63,0 64,1 65,1 66,3
PATCH 67,2 68,9 69,8 6A,2 6B,1
DUMP 3,55,77
DUMP  0,55,77
PATCH  19,55
TRACE  14,1A
TEND


COMMAND DUMP
0050     17 07 15 1B 65 00 01 02 02 03 05 04 03 02 01 00
0060     00 00 00 00 01 01 03 02 09 08 02 01 40 40 40 40
0070     40 40 40 40 40 40 00 00 00 40 40 40 40 40 40 40
TRACE COMMAND
IAR         INST                     EADDR (EADDR)     PSBU PSBL     R0 R1 R2 R3 R4 R5 R6
0014     LODA,0    0061,3,-          006B   0001        01   00      00 0A 00 0B 00 00 00
TRACE COMMAND
IAR         INST                     EADDR (EADDR)     PSBU PSBL     R0 R1 R2 R3 R4 R5 R6
0017     ADDA,0    0055,1            005F   0000        01   40      01 0A 00 0A 00 00 00
TRACE COMMAND
IAR         INST                     EADDR (EADDR)     PSBU PSBL     R0 R1 R2 R3 R4 R5 R6
001A     COMI,0    0A                001B   000A        01   40      01 0A 00 0A 00 00 00
TRACE COMMAND
IAR         INST                     EADDR (EADDR)     PSBU PSBL     R0 R1 R2 R3 R4 R5 R6
0014     LODA,0    0061,3,-          006A   0002        01   80      01 09 00 0A 00 00 00
TRACE COMMAND
IAR         INST                     EADDR (EADDR)     PSBU PSBL     R0 R1 R2 R3 R4 R5 R6
0017     ADDA,0    0055,1            005E   0001        01   40      02 09 00 09 00 00 00
TRACE COMMAND
IAR         INST                     EADDR (EADDR)     PSBU PSBL     R0 R1 R2 R3 R4 R5 R6
001A     COMI,0    0A                001B   000A        01   40      03 09 00 09 00 00 00
TRACE COMMAND
IAR         INST                     EADDR (EADDR)     PSBU PSBL     R0 R1 R2 R3 R4 R5 R6
0014     LODA,0    0061,3,-          0069   0008        01   80      03 08 00 09 00 00 00
TRACE COMMAND
IAR         INST                     EADDR (EADDR)     PSBU PSBL     R0 R1 R2 R3 R4 R5 R6
0017     ADDA,0    0055,1            005D   0002        01   40      08 08 00 08 00 00 00
TRACE COMMAND
IAR         INST                     EADDR (EADDR)     PSBU PSBL     R0 R1 R2 R3 R4 R5 R6
001A     COMI,0    0A                001B   000A        01   40      0A 08 00 08 00 00 00
TRACE COMMAND
IAR         INST                     EADDR (EADDR)     PSBU PSBL     R0 R1 R2 R3 R4 R5 R6
0014     LODA,0    0061,3,-          0068   0009        01   21      00 07 00 08 00 00 00
TRACE COMMAND
IAR         INST                     EADDR (EADDR)     PSBU PSBL     R0 R1 R2 R3 R4 R5 R6
0017     ADDA,0    0055,1            005C   0003        01   61      09 07 00 07 00 00 00




TRACE COMMAND
IAR         INST                     EADDR (EADDR)     PSBU PSBL     R0 R1 R2 R3 R4 R5 R6
001A     COMI,0    0A                001B   000A        01   40      0C 07 00 07 00 00 00
TRACE COMMAND
IAR         INST                     EADDR (EADDR)     PSBU PSBL     R0 R1 R2 R3 R4 R5 R6
0014     LODA,0    0061,3,-          0067   0002        01   61      02 06 00 07 00 00 00
TRACE COMMAND
IAR         INST                     EADDR (EADDR)     PSBU PSBL     R0 R1 R2 R3 R4 R5 R6
0017     ADDA,0    0055,1            005B   0004        01   61      02 06 00 06 00 00 00
TRACE COMMAND
IAR         INST                     EADDR (EADDR)     PSBU PSBL     R0 R1 R2 R3 R4 R5 R6
001A     COMI,0    0A                001B   000A        01   40      06 06 00 06 00 00 00
TRACE COMMAND
IAR         INST                     EADDR (EADDR)     PSBU PSBL     R0 R1 R2 R3 R4 R5 R6
0014     LODA,0    0061,3,-          0066   0003        01   80      06 05 00 06 00 00 00
TRACE COMMAND
IAR         INST                     EADDR (EADDR)     PSBU PSBL     R0 R1 R2 R3 R4 R5 R6
0017     ADDA,0    0055,1            005A   0005        01   40      03 05 00 05 00 00 00
TRACE COMMAND
IAR         INST                     EADDR (EADDR)     PSBU PSBL     R0 R1 R2 R3 R4 R5 R6
001A     COMI,0    0A                001B   000A        01   40      08 05 00 05 00 00 00
TRACE COMMAND
IAR         INST                     EADDR (EADDR)     PSBU PSBL     R0 R1 R2 R3 R4 R5 R6
0014     LODA,0    0061,3,-          0065   0001        01   80      08 04 00 05 00 00 00
TRACE COMMAND
IAR         INST                     EADDR (EADDR)     PSBU PSBL     R0 R1 R2 R3 R4 R5 R6
0017     ADDA,0    0055,1            0059   0003        01   40      01 04 00 04 00 00 00
TRACE COMMAND
IAR         INST                     EADDR (EADDR)     PSBU PSBL     R0 R1 R2 R3 R4 R5 R6
001A     COMI,0    0A                001B   000A        01   40      04 04 00 04 00 00 00
TRACE COMMAND
IAR         INST                     EADDR (EADDR)     PSBU PSBL     R0 R1 R2 R3 R4 R5 R6
0014     LODA,0    0061,3,-          0064   0001        01   80      04 03 00 04 00 00 00
TRACE COMMAND
IAR         INST                     EADDR (EADDR)     PSBU PSBL     R0 R1 R2 R3 R4 R5 R6
0017     ADDA,0    0055,1            0058   0002        01   40      01 03 00 03 00 00 00
TRACE COMMAND
IAR         INST                     EADDR (EADDR)     PSBU PSBL     R0 R1 R2 R3 R4 R5 R6
001A     COMI,0    0A                001B   000A        01   40      03 03 00 03 00 00 00
TRACE COMMAND
IAR         INST                     EADDR (EADDR)     PSBU PSBL     R0 R1 R2 R3 R4 R5 R6
0014     LODA,0    0061,3,-          0063   0000        01   80      03 02 00 03 00 00 00
TRACE COMMAND
IAR         INST                     EADDR (EADDR)     PSBU PSBL     R0 R1 R2 R3 R4 R5 R6
0017     ADDA,0    0055,1            0057   0002        01   00      00 02 00 02 00 00 00
TRACE COMMAND
IAR         INST                     EADDR (EADDR)     PSBU PSBL     R0 R1 R2 R3 R4 R5 R6
001A     COMI,0    0A                001B   000A        01   40      02 02 00 02 00 00 00
TRACE COMMAND
IAR         INST                     EADDR (EADDR)     PSBU PSBL     R0 R1 R2 R3 R4 R5 R6
0014     LODA,0    0061,3,-          0062   0000        01   80      02 01 00 02 00 00 00
TRACE COMMAND
IAR         INST                     EADDR (EADDR)     PSBU PSBL     R0 R1 R2 R3 R4 R5 R6
0017     ADDA,0    0055,1            0056   0001        01   00      00 01 00 01 00 00 00
```

ILLUSTRATION IV-2

```
TRACE COMMAND
IAR         INST                    EADDR (EADDR)      PSBU PSPL      R0 R1 R2 R3 R4 R5 R6
001A      COMI,0     0A             CC1B    000A        01   40       C1 01 00 01 00 00 00
COMMAND DUMP
0050      17 07 15 1B 65 00 C1 02 C2 03 05 04 C3 02 01 00
0060      00 00 01 02 03 04 C8 C6 C2 C0 03 01 40 40 40 40
0070      40 40 40 40 40 40 C0 C0 00 40 40 4C 40 40 40 40
```

```
NO. OF MACHINE CYCLES EXECUTED =        232


NO. OF INSTRUCTIONS EXECUTED =        73
```

ILLUSTRATION IV-3

```
START 00
PATCH  55,0 56,1 57,2 58,2 59,3
PATCH  5A,5 5B,4 5C,3 5D,2 5E,1
PATCH  5F,0
PATCH  61,0
PATCH 62,0  63,0 64,1 65,1 66,3
PATCH  67,2 68,9 69,8 6A,2 6B,1
DUMP  3,55,77
DUMP  0,55,77
TRACE  14,1A
PATCH  33,08
PATCH 12,08
FEND


COMMAND DUMP
0050     17 07 15 1B 65 00 01 02 02 03 05 04 03 02 01 00
0060     00 00 00 00 01 01 03 02 09 08 02 01 40 40 40 40
0070     40 40 40 40 40 40 00 00 00 40 40 40 40 40 40 40
TRACE COMMAND
IAR          INST                       EADDR (EADDR)      PSBU PSBL      R0 R1 R2 R3 R4 R5 R6
0014      LODA,0     0061,3,-           C06B    0001        01   08       08 08 00 08 00 00 00
TRACE COMMAND
IAR          INST                       EADDR (EADDR)      PSBU PSBL      R0 R1 R2 R3 R4 R5 R6
0017      ADDA,0     0054,1             CC5F    0000        01   48       01 08 00 0A 00 00 00
TRACE COMMAND
IAR          INST                       EADDR (EADDR)      PSBU PSBL      R0 R1 R2 R3 R4 R5 R6
001A      COMI,0     0A                 C01B    000A        01   48       01 08 00 0A 00 00 00
TRACE COMMAND
IAR          INST                       EADDR (EADDR)      PSBU PSBL      R0 R1 R2 R3 R4 R5 R6
0014      LODA,0     0061,3,-           CC6A    0002        01   88       01 0A 00 0A 00 00 00
TRACE COMMAND
IAR          INST                       EADDR (EADDR)      PSBU PSBL      R0 R1 R2 R3 R4 R5 R6
0017      ADDA,0     0054,1             CC5F    0001        01   48       02 0A 00 09 00 00 00
TRACE COMMAND
IAR          INST                       EADDR (EADDR)      PSBU PSBL      R0 R1 R2 R3 R4 R5 R6
001A      COMI,0     0A                 C01B    000A        01   48       C3 0A 00 09 00 00 00
TRACE COMMAND
IAR          INST                       EADDR (EADDR)      PSBU PSBL      R0 R1 R2 R3 R4 R5 R6
0014      LODA,0     0061,3,-           C069    0008        01   88       03 09 00 09 00 00 00
TRACE COMMAND
IAR          INST                       EADDR (EADDR)      PSBU PSBL      R0 R1 R2 R3 R4 R5 R6
0017      ADDA,0     0054,1             C05D    0002        01   48       C8 09 00 08 00 00 00
TRACE COMMAND
IAR          INST                       EADDR (EADDR)      PSBU PSBL      R0 R1 R2 R3 R4 R5 R6
001A      COMI,0     0A                 CC1B    000A        01   48       0A 09 00 08 00 00 00
TRACE COMMAND
IAR          INST                       EADDR (EADDR)      PSBU PSBL      R0 R1 R2 R3 R4 R5 R6
0014      LODA,0     0061,3,-           CC68    0009        01   29       C0 08 00 08 00 00 00
TRACE COMMAND
IAR          INST                       EADDR (EADDR)      PSBU PSBL      R0 R1 R2 R3 R4 R5 R6
0017      ADDA,0     0054,1             CC5C    0003        01   69       C9 08 00 07 00 00 00
```

```
TRACE COMMAND
IAR          INST                       EADDR (EADDR)      PSBU PSBL      R0 R1 R2 R3 R4 R5 R6
001A      COMI,0     0A                 CC1B    000A        01   48       C0 08 00 07 00 00 00
TRACE COMMAND
IAR          INST                       EADDR (EADDR)      PSBU PSBL      R0 R1 R2 R3 R4 R5 R6
0014      LODA,0     0061,3,-           C067    0002        01   69       C3 07 00 07 00 00 00
TRACE COMMAND
IAR          INST                       EADDR (EADDR)      PSBU PSBL      R0 R1 R2 R3 R4 R5 R6
0017      ADDA,0     0054,1             CC5B    0004        01   69       C2 07 00 06 00 00 00
TRACE COMMAND
IAR          INST                       EADDR (EADDR)      PSBU PSBL      R0 R1 R2 R3 R4 R5 R6
001A      COMI,0     0A                 CC1B    000A        01   48       C7 07 00 06 00 00 00
TRAGE COMMAND
IAR          INST                       EADDR (EADDR)      PSBU PSBL      R0 R1 R2 R3 R4 R5 R6
0014      LODA,0     0061,3,-           CC66    0003        01   88       07 06 00 06 00 00 00
TRACE COMMAND
IAR          INST                       EADDR (EADDR)      PSBU PSBL      R0 R1 R2 R3 R4 R5 R6
0017      ADDA,0     0054,1             CC5A    0005        01   48       03 06 00 05 00 00 00
TRACE COMMAND
IAR          INST                       EADDR (EADDR)      PSBU PSBL      R0 R1 R2 R3 R4 R5 R6
001A      COMI,0     0A                 C01B    000A        01   48       C8 06 00 05 00 00 00
TRACE COMMAND
IAR          INST                       EADDR (EADDR)      PSBU PSBL      R0 R1 R2 R3 R4 R5 R6
0014      LODA,0     0061,3,-           CC65    0001        01   88       08 05 00 05 00 00 00
TRACE COMMAND
IAR          INST                       EADDR (EADDR)      PSBU PSBL      R0 R1 R2 R3 R4 R5 R6
0017      ADDA,0     0054,1             CC59    0003        01   48       01 05 00 04 00 00 00
TRACE COMMAND
IAR          INST                       EADDR (EADDR)      PSBU PSBL      R0 R1 R2 R3 R4 R5 R6
001A      COMI,0     0A                 001B    000A        01   48       04 05 00 04 00 00 00
TRACE COMMAND
IAR          INST                       EADDR (EADDR)      PSBU PSBL      R0 R1 R2 R3 R4 R5 R6
0014      LODA,0     0061,3,-           CC64    0001        01   88       C4 04 00 04 00 00 00
TRACE COMMAND
IAR          INST                       EADDR (EADDR)      PSBU PSBL      R0 R1 R2 R3 R4 R5 R6
0017      ADDA,0     0054,1             C058    0002        01   48       01 04 00 03 00 00 00
TRACE COMMAND
IAR          INST                       EADDR (EADDR)      PSBU PSBL      R0 R1 R2 R3 R4 R5 R6
001A      COMI,0     0A                 C01B    000A        01   48       C3 04 00 03 00 00 00
TRACE COMMAND
IAR          INST                       EADDR (EADDR)      PSBU PSBL      R0 R1 R2 R3 R4 R5 R6
0014      LODA,0     0061,3,-           CC63    0000        01   88       03 03 00 03 00 00 00
TRACE COMMAND
IAR          INST                       EADDR (EADDR)      PSBU PSBL      R0 R1 R2 R3 R4 R5 R6
0017      ADDA,0     0054,1             CC57    0002        01   08       C0 03 00 02 00 00 00
TRACE COMMAND
IAR          INST                       EADDR (EADDR)      PSBU PSBL      R0 R1 R2 R3 R4 R5 R6
001A      COMI,0     0A                 CC1B    000A        01   48       02 03 00 02 00 00 00
TRACE COMMAND
IAR          INST                       EADDR (EADDR)      PSBU PSBL      R0 R1 R2 R3 R4 R5 R6
0014      LODA,0     0061,3,-           CC62    0000        01   88       C2 02 00 02 00 00 00
TRACE COMMAND
IAR          INST                       EADDR (EADDR)      PSBU PSBL      R0 R1 R2 R3 R4 R5 R6
0017      ADDA,0     0054,1             CC56    0001        01   08       C0 02 00 01 00 00 00
```

ILLUSTRATION IV-4

```
TRACE COMMAND
IAR          INST                         EADDR (EADDR)      PSBU PSBL      R0 R1 R2 R3 R4 R5 R6
001A      CCMI,0     0A                   C01B    000A        01   48       01 02 00 01 00 00 00
TRACF CCMMAND
IAR          INST                         EADDR (EADDR)      PSBU PSBL      R0 R1 R2 R3 R4 R5 R6
0014      LCCA,0     0061,3,-             CC61    0000        01   88       01 01 00 01 00 00 00
TRACF CCMMAND
IAR          INST                         EADCR (EADCR)      PSBU PSBL      R0 R1 R2 R3 R4 R5 R6
0017      ACCA,0     0054,1               CC55    0000        01   08       C0 01 00 00 00 00 00
TRACF CCMMAND
IAR          INST                         EADCR (EADDR)      PSBU PSBL      R0 R1 R2 R3 R4 R5 R6
001A      CCMI,0     0A                   CC18    000A        01   08       C0 01 00 00 00 00 00
COMMAND DUMP
0050     17 07 15 1B 65 00 01 02 C2 C3 05 04 03 02 01 00
C060     00 00 01 02 03 04 08 C7 C3 00 03 01 40 40 40 40
0070     40 40 40 40 40 40 00 C0 00 40 40 40 40 40 40 40
```

```
NO. OF MACHINE CYCLFS EXECUTEC =        252


NO. CF INSTRUCTIONS EXECUTEC =        79
```

APPENDIX A

COMMAND SUMMARY

COMMAND NAME	PARAMETERS	DESCRIPTION
DUMP.	LOC, FWA-LWA(; . . . ;LOC, FWA-LWA)	Display the area of memory. FWA-LWA, whenever the instruction at LOC executes.
FEND	None	Execute the last simulation and terminate the entire run.
INPUT	VALUE(; . . . ;VALUE)	Define the data to be read by simulated I/O instructions.
INSTR.	LOC(; . . . ;LOC)	Display the processor registers whenever the instruction at LOC executes.
LIMIT	NO	Specify the total number of instructions executed.
PATCH	LOC,VALUE(;. . . ;LOC,VALUE)	Initialize each memory location, LOC, to VALUE.
REFER.	LOC(; . . . ;LOC)	Display the processor register whenever the i struction at LOC is referenced by anothe instruction.
SETP.	LOC(,PSL=VALUE) (,PSU=VALUE)	Set the program status byte (lower and/or upper) to VALUE whenever the instruction at LOC executes.
SETR.	LOC(,R0=VALUE). . .(R6=VALUE)	Set the general purpose registers to VALUE whenever the instruction at LOC executes.
SROM	FWA-LWA	Specify the boundaries of Read-Only Memory.
START	LOC	Start the simulated program execution at LOC.
STAT	None	Display instruction statistics at end of program execution.
STOP.	LOC(; . . . ; LOC)	Terminate the program execution when the instruction at LOC executes.
TEND	None	Execute the last simulation and prepare to read the User Commands for the next simulation.
TRACE.	FWA-LWA(; . . . ;FWA-LWA)	Display the processor registers whenever an i struction executes, which lies within the area of memory, FWA-LWA.

APPENDIX B

ERROR MESSAGES

Whenever the Simulator detects an error in the User Commands, it prints one of the following error messages:

ERROR IN OBJECT MODULE CARD NUMBER
the 2650 object module is incorrectly formatted.

INPUT DATA TABLE OVERFLOW
an INPUT command attempted to expand the simulated data input buffer beyond its limit (200 bytes).

PARAMETER OUT OF RANGE
a User Command either contains an address which is outside the bounds of simulated memory or the command defines a datum which is larger than one byte (255_{10}).

SIM MEMORY EXCEEDED
a 2650 object module loads into an area which is outside of simulated memory.

SYNTAX ERROR IN COMMAND
the command parameters are either missing or in error.

TOO MANY COMMANDS
the maximum number of dynamic commands has been exceeded.

TOO MANY DUMP COMMANDS
the maximum number of DUMP commands has been exceeded.

TOO MANY SET REGISTER COMMANDS
the maximum number of SETR. commands has been exceeded.

TOO MANY SET PSB COMMANDS
the maximum number of SETP. commands has been exceeded.

UNRECOGNIZED COMMAND
a command has been read which is unknown to the Simulator.

UNEXPECTED END OF FILE
either the object module or the set of User Commands is missing, or one of their respective card decks is incorrectly formatted, or the FEND command is missing.

Whenever the Simulator detects an error while the simulated program is executing it prints one of the following error messages:

ADDRESS OUT OF RANGE
an instruction attempted to access a location which lies outside of simulated memory.

INSUFFICIENT INPUT DATA
a I/O instruction attempted to read another datum from the input data buffer (INPUT) after all the data from the buffer had been read. The simulated input register remains unchanged i.e., the instruction is essentially ignored, and program execution continues.

LC=   ATTEMPT TO STORE INTO ROM
an instruction attempted to store data into the area designated as ROM (SROM).

LC EXCEEDS MEMORY

the program attempted to execute a memory location which lies outside of simulated memory.

NO KNOWN OPCODE

the program attempted to execute a memory location which did not contain a valid instruction. Either the program was modified during execution or the program is attempting to execute data.

APPENDIX C

SIMULATOR RESTRICTIONS

SIMULATOR RESTRICTIONS

1. The simulated memory reserved by the Simulator for program storage is limited to 2048 bytes.* Thus, the Simulator will accept only programs or program segments which fit into this area. This implies that the 2650 paging facility (page size = 8192 bytes) cannot be simulated.
2. Some User Commands are limited in the amount of entries they may accept.

COMMAND	LIMIT
DUMP.	5 LOC's
SETR.	4 LOC's
SETP.	2 LOC's
INPUT	200 VALUE's
All "dynamic" commands	30 LOC's (for TRACE. count 1 for each set of FWA-LWA)

*This may be expanded to 8192 bytes if sufficient memory is available.

APPENDIX D

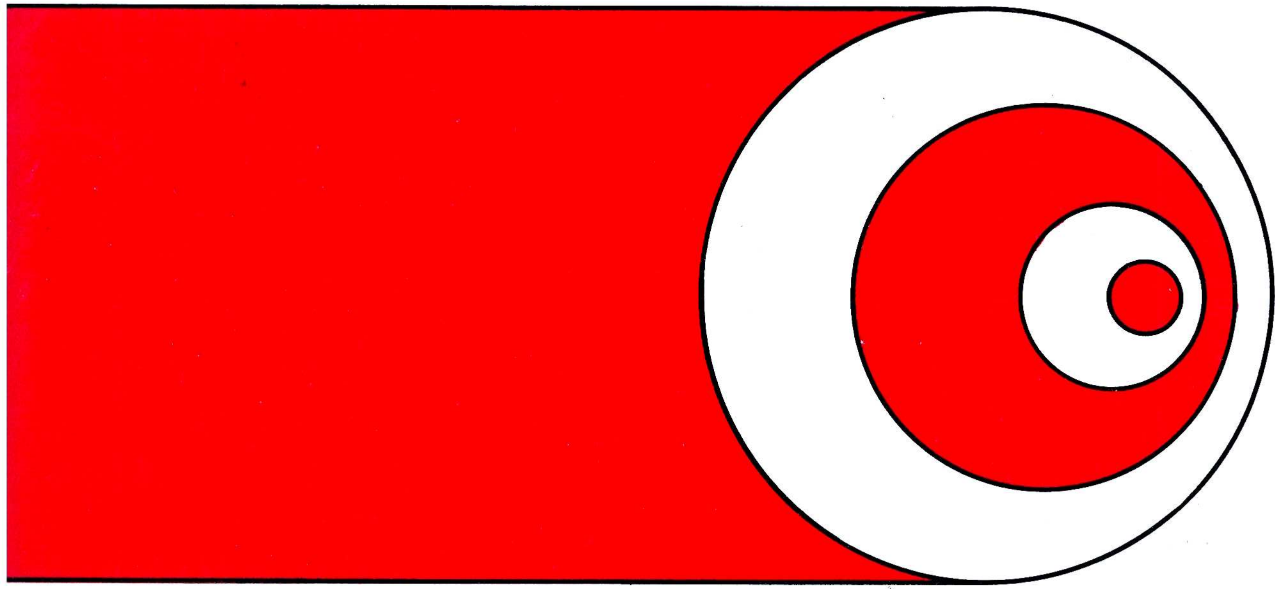
SIMULATOR RUN PREPARATION

In order to prepare a program for execution by the Simulator, the programmer:

1. Codes a program in 2650 Assembly Language.
2. Assembles the program until no assembly errors occur.
3. Obtains the object module and listing for the assembled program.
4. Generates command cards using addresses from the listing of the assembled program.
5. Submits the object module and the command cards in that order for a Simulator run.

SIGNETICS
2650 MICROPROCESSOR

SYSTEM
APPLICATION NOTES



signetics

MICROPROCESSOR

2650 EVALUATION PRINTED CIRCUIT BOARD LEVEL SYSTEM (PC1001)....................SP50

signetics

2650 EVALUATION PRINTED CIRCUIT BOARD (PC1001) SP50

APPLICATIONS MEMO

GENERAL

The PC1001 is an evaluation and design tool for the 2650 microprocessor. Each PC1001 board has a 2650 microprocessor, 1k bytes of RAM, 1k bytes of PROM loaded with PIPBUG*, a crystal clock, and sufficient additional logic to allow the user to exercise all aspects of the 2650 microprocessor. There is a serial I/O port on the board that can be used to drive a current loop driven terminal or an RS232 type terminal. The PC1001 provides the system engineer with a very flexible design tool from which he can easily develop a pre-production prototype of his product designed around the 2650 microprocessor.

FEATURES

The PC1001 has many features that make it a valuable design aid. The most noteworthy features are:

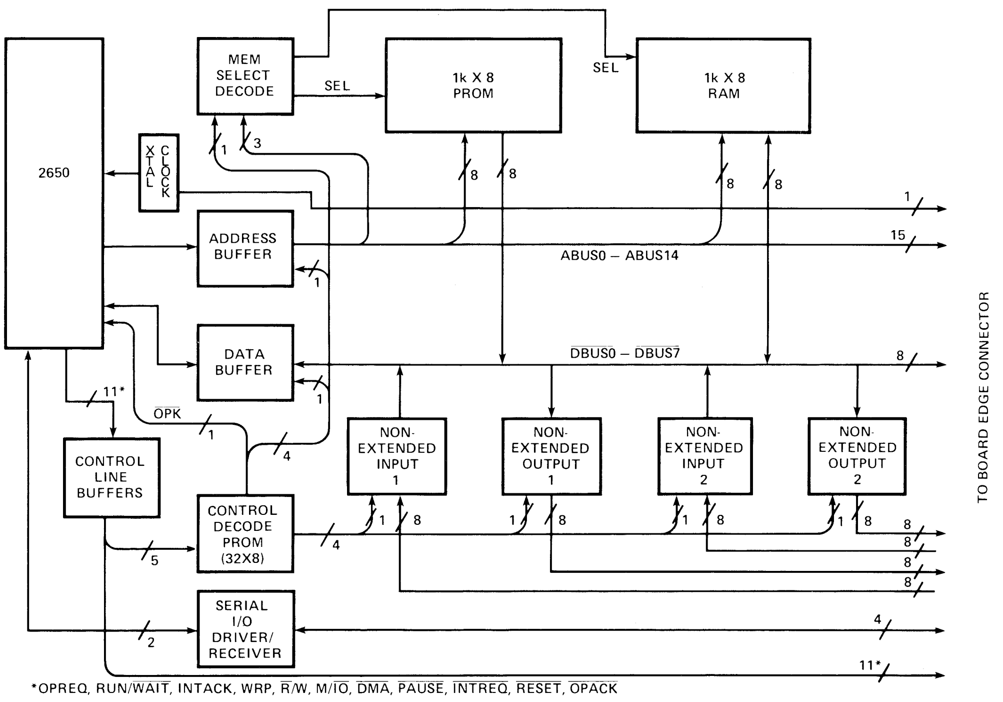
- The Signetics 2650 N-MOS, 8-bit microprocessor
- 1k – bytes of RAM memory
- 1k – bytes of PROM memory
- A 1MHz crystal oscillator
- A serial I/O channel
- Two Non-Extended 8-bit parallel input ports
- Two Non-Extended 8-bit parallel output ports
- Buffered address, data, and control lines for implementing additional 8-bit parallel I/O ports or expanded memory
- Direct Memory Access (DMA) capability, including the memory on the PC1001 board
- Display indicators on the board for the RUN/$\overline{\text{WAIT}}$, OPREQ, M/$\overline{\text{IO}}$, R/$\overline{\text{W}}$ control lines, and the Non-Extended output ports
- Vectored interrupts
- A program debug module (called PIPBUG) written for use with the 2650

*PIPBUG – a program debug module

DESCRIPTION

The PC1001 is configured as a very flexible, general purpose microprocessor board to allow the system designer to easily expand memory, implement input/output functions and execute programs written for the 2650. A functional description of the PC1001 is given in this section. A functional block diagram of the PC1001 is shown in Figure 1.

PC 1001 BLOCK DIAGRAM



*OPREQ, RUN/$\overline{\text{WAIT}}$, INTACK, WRP, R/$\overline{\text{W}}$, M/$\overline{\text{IO}}$, $\overline{\text{DMA}}$, PAUSE, $\overline{\text{INTREQ}}$, RESET, $\overline{\text{OPACK}}$

FIGURE 1

CPU

The 2650 is the heart of the PC1001, executing instructions from memory and controlling the I/O functions. The address, data, and control lines of the 2650 are buffered and available at the edge connector of the PC1001. The on-board bus drivers allow the user to build a microprocessor system around the PC1001 without additional buffering. The tri-state function of the 2650 address and data busses is transferred to the buffer gates which drive the lines used by the system designer. The address and data bus buffers are in the tri-state mode whenever the OPREQ line from the 2650 is a logic ZERO.

MEMORY

The 1024 bytes of read only memory are implemented with 82S129 256X4 bipolar PROM's. The PROM's are accessed by addressing the first 1024 bytes of the address space (locations 0_{16} – $3FF_{16}$). The PROM's are mounted in sockets on the PC1001 board and are loaded with the PIPBUG debug program. The sockets on the PC1001 board allow the user to put different 82S129 PROM's in the first 1k bytes of the memory address space when developing a prototype system.

The 1024 bytes of random access memory are implemented with 2606 256X4 MOS RAM's. The RAM's are accessed by addressing the second 1024 bytes of the address space (locations 400_{16} – $7FF_{16}$).

PARALLEL I/O

The buffered address, data, and control lines available to the user of the PC1001 allow any of the 2650 parallel I/O modes to be implemented, or to expand memory beyond the 2k bytes already on the board. The extended I/O instructions provide device select capability for 256 I/O functions by decoding the least significant 8-bits of the address bus (ABUS 0 – ABUS 7). The buffered data bus is a bi-directional tri-state bus so that input devices may use the data bus by driving it with tri-state drivers.

If the Non-Extended I/O instructions are used, two latched output ports and two gated input ports are already provided on the PC1001, and no control line decoding is necessary.

When the 2650 executes memory reference instructions or Non-Extended I/O instructions, the control decode PROM generates the operation acknowledge signal ($\overline{\text{OPACK}}$) in response to operation request (OPREQ). When the 2650 executes Extended I/O instructions, the selected I/O device must generate $\overline{\text{OPACK}}$. By requiring the I/O device to return the $\overline{\text{OPACK}}$ signal, the PC1001 gives the user the flexibility of connecting peripheral functions that may require more than one microsecond to respond to an I/O request. If the Extended I/O functions are all faster than one microsecond they will not slow down the 2650, and $\overline{\text{OPACK}}$ may be tied to logic ZERO.

SERIAL I/O

The 2650 is equipped with a SENSE input and a FLAG output. These two functions provide a serial I/O data path directly into the 2650. Part of the PIPBUG PROM program is dedicated to implementing an asynchronous serial communications port for the PC1001. The program checks the SENSE line for a start bit from the serial device to achieve synchronization. Once a start bit is detected, the 2650 shifts the next eight character bits into register R0. The PC1001 is designed for full duplex serial I/O, and will echo the transmitted character back to the serial device using the FLAG output. The timing loops that determine when to sample a character bit are written for a ten character per second serial data rate (110 baud), but the 2650 is capable of handling much higher serial data rates.

The serial I/O device used with the PC1001 may be a 20 milliamp current loop device, or it may be RS232 compatible (voltage driven). A current loop driver and receiver, and an RS232 driver and receiver are on the PC1001 board. The type of driver and receiver is selected with a wire jumper. If the RS232 driver and receiver is used, external ±15 volt power supplies are required. If the current loop driver and receiver is used, the PC1001 requires only a single +5 volt power supply.

The PIPBUG debug program includes a read paper tape control function. The program sets a bit in the output register of Non-Extended I/O port C (WRTC instruction) to advance the tape reader one character at a time. This function can be used by modifying a standard teletype to include a tape reader control relay and driving it with the TTY TAPE READER OUT SIGNAL.

It should be pointed out that the tape reader control bit and bit 7 of the Non-Extended I/O port (OPC7) are the same and caution should be exercised to avoid a conflict between the two functions.

CLOCK

The clock circuit on the PC1001 is a hybrid circuit crystal oscillator that runs at a frequency of 1.000 MHz. Instruction loops are used to determine bit times and the crystal controlled clock minimizes errors due to changes in the system clock.

The clock input to the 2650 that is driven by the crystal controlled clock (pin 38) is available at the edge connector of PC1001. If the user chooses to drive the PC1001 with an external clock he must first remove the crystal clock circuit. The clock input to the 2650 is fully TTL compatible and requires no special drive circuitry.

DISPLAYS

Minature LED indicator displays are driven by the three basic control lines (OPREQ, M/$\overline{\text{IO}}$, and R/$\overline{\text{W}}$), and the Non-Extended output latches. A logic ONE state on the control lines, or in the output latches, "lights" the corresponding LED. The minature LED's are mounted on the PC1001 board and are shown in Figure 2.

PRINTED CIRCUIT BOARD LAYOUT

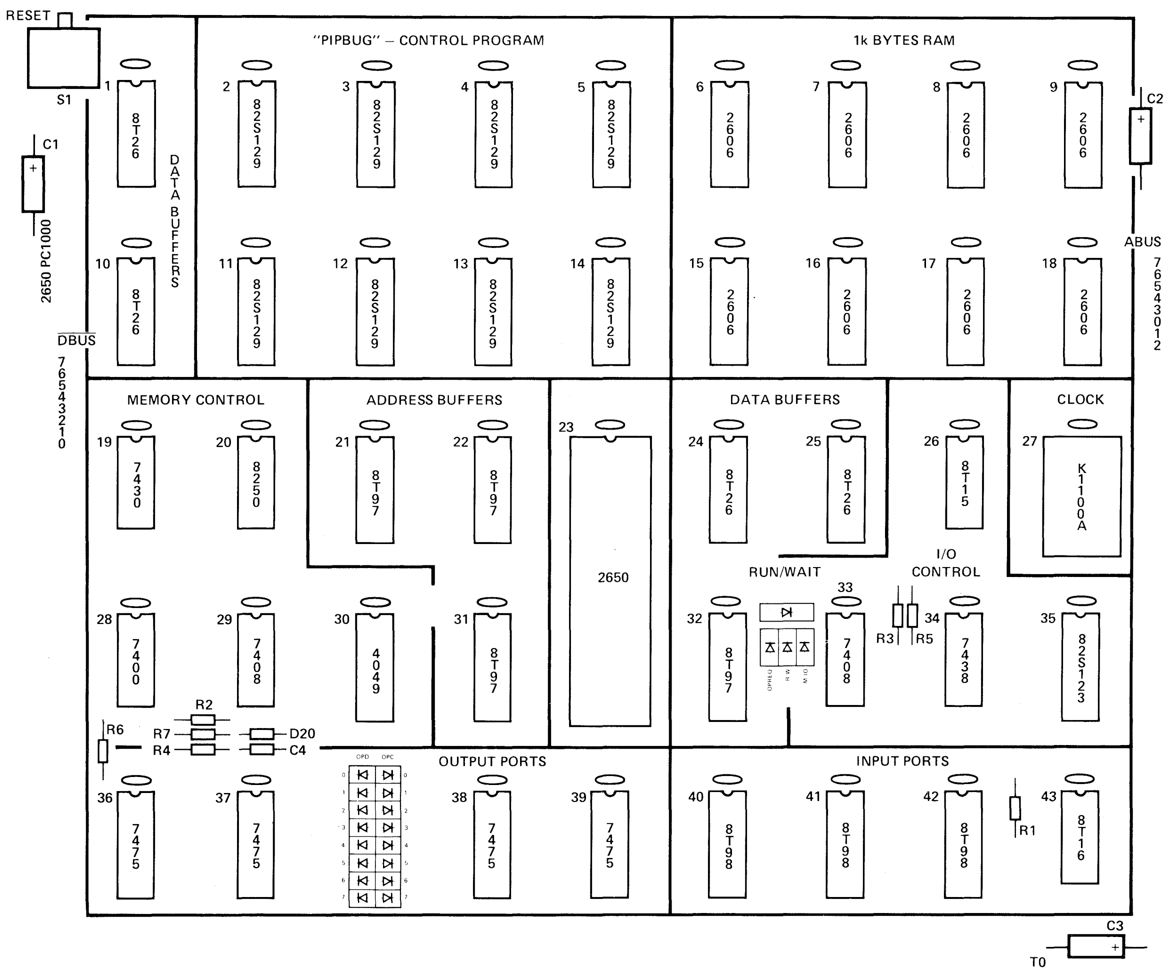


FIGURE 2

DMA

Direct access to memory by an external device (DMA) is easily accomplished with the PC1001. An input to the board is provided for direct memory access and the signal name of that input is $\overline{\text{DMA}}$ (PC1001 pin 14). When $\overline{\text{DMA}}$ is pulled "low" the 2650 finishes executing the current instruction and enters the wait state. To avoid interrupting a memory or I/O transfer in progress the $\overline{\text{DMA}}$ line should not be pulled "low" while OPREQ is "high". When the RUN/$\overline{\text{WAIT}}$ lines goes "low" the external device may drive the address, data, and control lines (except OPREQ, and RUN/$\overline{\text{WAIT}}$) to accomplish the necessary DMA transfer.

An external operation request line ($\overline{\text{OPEX}}$) is provided for DMA transfers to the memory on the PC1001 board. Since OPREQ is only driven by the 2650, and is used in the memory select decoders, the user must drive $\overline{\text{OPEX}}$ to access the memory on the PC1001.

Because the DMA function is implemented with the pause feature of the 2650, and since the 2650 is a static device, the length of time that the DMA device may be active for any one transfer is limited only by the other processing responsibilities of the 2650.

INTERRUPTS

The 2650 has a true vectored interrupt system. The user must first drive the interrupt line ($\overline{\text{INTREQ}}$) on the PC1001, then wait to be acknowledged (INTACK), and finally drive the data bus with a 7-bit signed displacement relative to page zero, location zero. The displacement vector may also indicate indirect addressing, allowing the interrupt service sub-routine to be located anywhere in the 32k-byte address space.

INTERRUPTS (Continued)

The $\overline{\text{INTREQ}}$ line may be driven by several interrupting devices in a "wired OR" configuration. When a priority exists between the various interrupting devices, and to prevent confusion from multiple simultaneous interrupts, the user must arrange the interrupt hardware to resolve priority and simultaneity conflicts.

The PC1001 board comes with PIPBUG stored in the first 1k-bytes of ROM and therefore the user cannot store an interrupt service subroutine or an indirect address in this part of the memory address space. But the interrupt displacement vector may be a negative number referring to the last 64 locations in page zero ($1FBF_{16}$ to $1FFF_{16}$). If an indirect address or interrupt service subroutine is placed in one of the last 64 locations of page zero, the user must also provide external memory at the locations used (the PC1001 has only 2k-bytes of memory on the board).

There is another way to accomplish a "link" to an interrupt service subroutine through the ROM on the PC1001. It is possible that PIPBUG instructions themselves could provide an indirect address to the second 1k-bytes of RAM on the PC1001 board. An example of a very useable indirect address to an interrupt service routine may be found at locations 8_{16} and 9_{16} of PIPBUG. If these locations are used as an indirect address, the program would branch to location 477_{16} where it would expect to find a subroutine to service the active interrupt.

A timing diagram for interrupt processing is shown in Figure 3, as well as the format for the displacement vector.

LOGIC

The logic on the PC1001 board is uncomplicated and very general purpose. It includes:

1. 2650 CPU and memory
2. Address bus, and data bus drivers and receivers
3. Control line drivers and receivers
4. Control line decode
5. Memory select decode
6. Serial I/O transmitter and receiver
7. Non-Extended parallel I/O latches and receivers

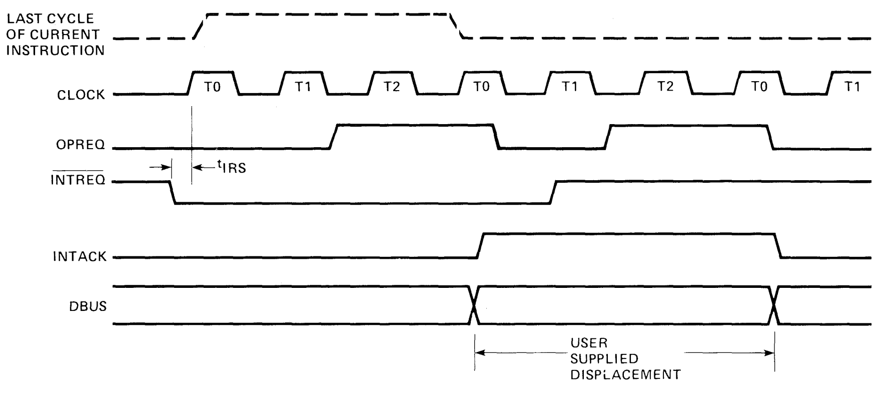
The PC1001 logic drawing will be referred to during this description and is shown in Figure 4. The integrated circuit numbers used in Figure 4 may be cross-correlated to those used on Figure 2 for locating an integrated circuit on the PC1001 board.

CPU and MEMORY

CPU – The address bus, and the data bus from the 2650 are buffered for easy system expansion. With the exception of the address tri-state control line ($\overline{\text{ADREN}}$) and the data bus tri-state control line ($\overline{\text{DBUSEN}}$), all of the control lines from the 2650 are also buffered. The $\overline{\text{ADREN}}$ and $\overline{\text{DBUSEN}}$ lines are tied "low" on the PC1001 board, and the tri-state function of the address, data, and control lines is fulfilled by the buffers.

The clock input is driven directly from the K1100A clock circuit (IC #27). The clock output is available off-board on PC1001 pin 23 (the signal name is CLOCK). The K1100 clock circuit has a frequency stability of ±.01% and will drive 10 standard TTL (7400 series) unit loads. The 2650

INTERRUPT TIMING



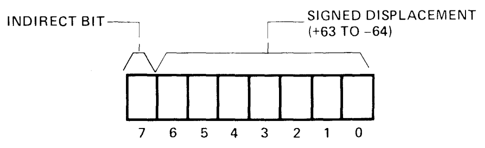


FIGURE 3

PC1001 LOGIC DIAGRAM

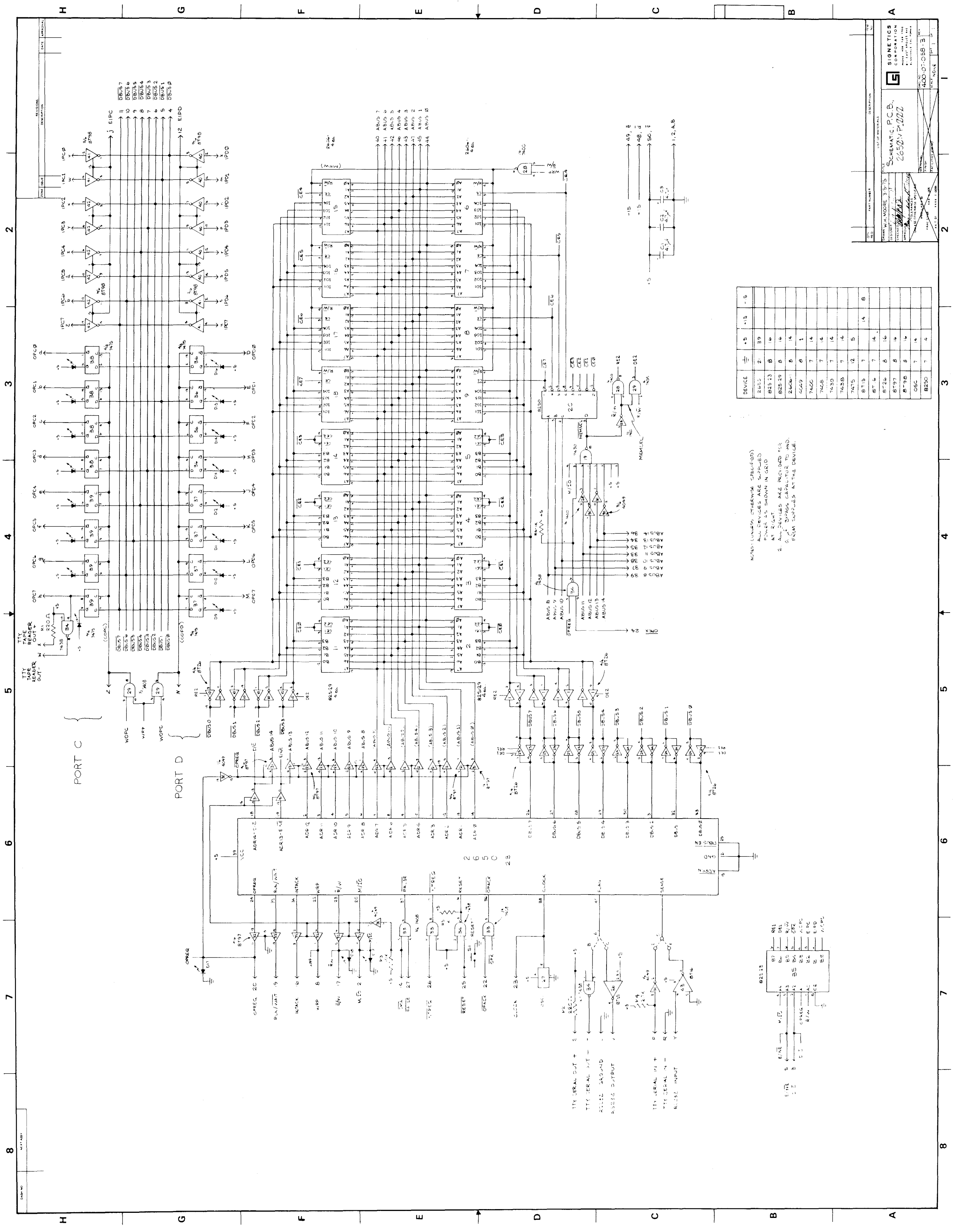


FIGURE 4

CPU and MEMORY (Continued)

is the only load the clock must drive on the PC1001, using only 10 μamps of its drive capability.

Memory – The memories are of two types: 82S129 256X4 PROMs, and 2606 256X4 RAMs. All 16 memory IC's (IC's 2-9, and 11-18) are addressed by the least significant 8-bits of the buffered address bus. The memories drive and receive the data bus through 8T26 tri-state transceivers to prevent an expanded system from presenting too great a capacitive load for the MOS memories.

The PROM memories (IC's 2-5, and IC's 11-14) are plugged into sockets and come programmed with PIPBUG. Any user's program may be stored in these PROM locations if PIPBUG is not required.

When the PC1001 is used to develop programs, and PIPBUG is resident in the first 1k-bytes of memory space, all of the 1k-bytes of RAM memory is available for use except the first 64 bytes (400_{16} to $43F_{16}$). The 64 locations are used by PIPBUG for temporary storage.

ADDRESS AND DATA BUS DRIVERS AND RECEIVERS

The data bus (DBUS0 - DBUS7) is buffered with 8T26 quad tri-state transceivers (IC's 24 and 25). These transceivers are inverting, and therefore the data bus transferred off of the PC1001 board is negative true ($\overline{\text{DBUS0}}$ - $\overline{\text{DBUS7}}$). The tri-state transceivers are controlled by RE1 (receiver control) and DE1 (driver control) from the control decode PROM (IC 35). The receiver control RE1 is a negative true signal (active "low") and has the following logic equation:

$$\text{RE1} = \overline{\text{OPREQ} \bullet \text{R}/\overline{\text{W}}}$$

The driver control DE1 is a positive true signal and has the following logic equation:

$$\text{DE1} = \text{OPREQ} \bullet \overline{\text{R}}/\text{W}$$

The logic equations reflect the fact that the 2650 drives the external data bus ($\overline{\text{DBUS0}}$ - $\overline{\text{DBUS7}}$) during all write operations (memory or I/O), and receives the external data bus during all read operations. But, when OPREQ is not a "high" the external data bus transceivers are in the tri-state mode.

The memory on the PC1001 board is buffered from the user's data bus ($\overline{\text{DBUS0}}$ - $\overline{\text{DBUS7}}$) with 8T26 quad tri-state transceivers. These transceivers are inverting so that information stored in memory is not complimented relative to the 2650. These transceivers are controlled by RE2 (receiver control) and DE2 (driver control) from the memory select decode logic. The logic for these control lines is shown below IC 20 (IC's 28 and 29) in Figure 4 and they have the following logic equations:

$$\text{RE2} = \text{MEMSEL} \bullet \overline{\text{R}}/\text{W}$$
$$\text{DE1} = \text{MEMSEL} \bullet \text{R}/\overline{\text{W}}$$

The RE2 control line is a negative true signal and is active when the memory on the PC1001 is selected to be written into. The DE1 control line is positive true and active when the memory on the PC1001 is selected to be read from.

The address bus is buffered with 8T97 tri-state buffers (IC's 21, 22, and 31). These buffers are in the tri-state mode whenever OPREQ is inactive.

CONTROL LINE DRIVERS AND RECEIVERS

The two control lines OPREQ and RUN/$\overline{\text{WAIT}}$ are buffered with 8T97 tri-state buffers (IC 32), but are never placed in the tri-state mode.

The control lines INTACK, WRP, $\overline{\text{R}}$/W, and M/$\overline{\text{IO}}$ are also driven by 8T97 tri-state buffers (IC 32), and are switched to the tri-state mode when the $\overline{\text{DMA}}$ line is pulled "low". The pause input to the 2650 may be activated by driving the $\overline{\text{DMA}}$ line (PC1001 pin 14) or the $\overline{\text{PAUSE}}$ line (PC1001 pin 27) "low".

The interrupt request line and the reset line to the 2650 are buffered by TTL AND gates (IC 33). The reset switch on the PC1001 (upper left corner of Figure 2) is "wire ORed" with the $\overline{\text{RESET}}$ line to the PC1001 board (PC1001 pin 25).

The operation acknowledge line to the 2650 ($\overline{\text{OPACK}}$) is buffered with a TTL AND gate (IC 33), and has as its inputs an external acknowledge ($\overline{\text{OPACK}}$, PC1001 pin 22) and an internal acknowledge (OPK). The internal acknowledge is generated for all memory access cycles and Non-Extended I/O cycles initiated by the 2650. For Extended I/O cycles the external device must generate the external operation acknowledge ($\overline{\text{OPACK}}$).

CONTROL LINE DECODE

A control line decoder is implemented with a 32X8 PROM (82S123) to generate secondary control lines used by the logic supporting the 2650. The primary control lines from the 2650 ($\overline{\text{R}}$/W, OPREQ, M/$\overline{\text{IO}}$ E/$\overline{\text{NE}}$, and D/$\overline{\text{C}}$) are used to address the PROM, and each address represents one combination of the primary control lines. Stored at each memory location are eight bits, each one of which represents the logical state of a secondary control line. There are five address inputs to the PROM, and the 32 (2^5) possible addresses exhaust all of the logical combination of the primary control lines. The secondary control lines, their logic equations, and their functions are given in Table 1. Table 2 shows the contents of each of the 32 locations of the PROM. The control line decode PROM is shown in Figure 4 (IC 35).

MEMORY SELECT DECODE

The memory select decode logic is shown in Figure 4 (IC's 19, 20, 28, 29, 30 and 34). The 2k-bytes of memory are implemented with 256X4 bit memory chips. The memory chips are arranged into eight 256-byte sections.

The ninth, tenth, and eleventh bits of the address bus (ABUS8-ABUS10) are decoded to select one of the eight 256-byte sections of memory. The one-of-eight decoder (IC 20) is enabled by MEMSEL, which has the following logic equations:

$$\overline{\text{MEMSEL}} = (\text{OPREQ} + \text{OPEX}) \bullet \text{M}/\overline{\text{IO}} \bullet \overline{\text{ABUS11}} \bullet \overline{\text{ABUS12}} \bullet \overline{\text{ABUS13}} \bullet \overline{\text{ABUS14}}$$

The MEMSEL line is also used to enable the 8T26 quad tri-state transceivers that buffer the memory on the PC1001 from the external data bus ($\overline{\text{DBUS0}}$-$\overline{\text{DBUS7}}$).

SERIAL I/O TRANSMITTER AND RECEIVER

A serial I/O port is implemented on the PC1001 with the flag and sense line of the 2650. The PIPBUG program handles the serial I/O using software timing loops to sample the SENSE input and build eight bit ASCII characters. The PC1001 is capable of interfacing to a current loop type terminal, or an RS232 compatible terminal.

The current loop driver uses an open collector NAND gate (IC 34) as the switching element. The 20 milliamp source is a 220Ω resistor connected to +5 volts on the PC1001 (PC1001 pin S), and the open collector NAND gate either provides a return path for the 20 milliamps (NAND output "on") or it does not (NAND output "off"). The current loop receiver is a CMOS hex inverter (IC 30) with the input pulled to +5 volts through a 2.7kΩ resistor (PC1001 pin P). The teletype transmitter is a contact closure and connects the input of the CMOS inverter to the receiver return line (PC1001 pin R), which is tied to ground on the PC1001 board.

The RS232 driver is an 8T15 EIA Line Driver (IC 26), and the RS232 receiver is an 8T16 EIA Line Receiver (IC 43). The 8T15 is the only chip on the PC1001 that does not operate on the +5 volt power supply, and ±15 volt power supplies are specified for this driver.

CONTROL LINE DECODE PROM DESCRIPTION

SIGNAL NAME	OUTPUT	PIN #	LOGIC EQUATION	FUNCTION
WOPD	B0	1	$\text{WOPD} = \text{OPREQ} \bullet \overline{\text{M}/\overline{\text{IO}}} \bullet \overline{\text{E}/\overline{\text{NE}}} \bullet \text{D}/\overline{\text{C}} \bullet \overline{\text{R}}/\text{W}$	LOADS NON-EXTENDED OUTPUT LATCH, PORT D
EIPD*	B1	2	$\overline{\text{EIPD}} = \text{OPREQ} \bullet \overline{\text{M}/\overline{\text{IO}}} \bullet \overline{\text{E}/\overline{\text{NE}}} \bullet \text{D}/\overline{\text{C}} \bullet \overline{\overline{\text{R}}/\text{W}}$	ENABLES NON-EXTENDED INPUT GATES, PORT D
EIPC*	B2	3	$\overline{\text{EIPC}} = \text{OPREQ} \bullet \overline{\text{M}/\overline{\text{IO}}} \bullet \overline{\text{E}/\overline{\text{NE}}} \bullet \overline{\text{D}/\overline{\text{C}}} \bullet \overline{\overline{\text{R}}/\text{W}}$	ENABLES NON-EXTENDED INPUT GATES, PORT C
WOPC	B3	4	$\text{WOPC} = \text{OPREQ} \bullet \overline{\text{M}/\overline{\text{IO}}} \bullet \overline{\text{E}/\overline{\text{NE}}} \bullet \overline{\text{D}/\overline{\text{C}}} \bullet \overline{\text{R}}/\text{W}$	LOADS NON-EXTENDED OUTPUT LATCH, PORT C
OPK*	B4	5	$\overline{\text{OPK}} = \text{OPREQ}\ [(\text{M}/\overline{\text{IO}}) + (\overline{\text{M}/\overline{\text{IO}}} \bullet \text{E}/\overline{\text{NE}})]$	RETURNS $\overline{\text{OPACK}}$ FOR ALL OPREQ EXCEPT EXTENDED I/O
$\text{R}/\overline{\text{W}}$	B5	6	$\text{R}/\overline{\text{W}} = \overline{\overline{\text{R}}/\text{W}}$	INVERTS $\overline{\text{R}}/\text{W}$
DE1	B6	7	$\text{DE1} = \text{OPREQ} \bullet \overline{\text{R}}/\text{W}$	DRIVES EXTERNAL DATA BUS ($\overline{\text{DBUS0}}$ – $\overline{\text{DBUS7}}$)
RE1*	B7	9	$\overline{\text{RE1}} = \text{OPREQ} \bullet \overline{\overline{\text{R}}/\text{W}}$	ENABLES RECEIVERS OF EXTERNAL DATA BUS ($\overline{\text{DBUS0}}$ – $\overline{\text{DBUS7}}$)

*NEGATIVE TRUE SIGNALS

TABLE 1

CONTROL LINE DECODE PROM

ADDRESS	INPUT					OUTPUT								$OUTPUT_{16}$
	A4	A3	A2	A1	A0	7	6	5	4	3	2	1	0	
0	0	0	0	0	0	1	0	1	1	0	1	1	0	B6
1	0	0	0	0	1	1	0	0	1	0	1	1	0	96
2	0	0	0	1	0	0	0	1	0	0	0	1	0	22
3	0	0	0	1	1	1	1	0	0	1	1	1	0	CE
4	0	0	1	0	0	1	0	1	1	0	1	1	0	B6
5	0	0	1	0	1	1	0	0	1	0	1	1	0	96
6	0	0	1	1	0	0	0	1	0	0	1	0	0	24
7	0	0	1	1	1	1	1	0	0	0	1	1	1	C7
8	0	1	0	0	0	1	0	1	1	0	1	1	0	B6
9	0	1	0	0	1	1	0	0	1	0	1	1	0	96
10	0	1	0	1	0	0	0	1	1	0	1	1	0	36
11	0	1	0	1	1	1	1	0	1	0	1	1	0	D6
12	0	1	1	0	0	1	0	1	1	0	1	1	0	B6
13	0	1	1	0	1	1	0	0	1	0	1	1	0	96
14	0	1	1	1	0	0	0	1	1	0	1	1	0	36
15	0	1	1	1	1	1	1	0	1	0	1	1	0	D6
16	1	0	0	0	0	1	0	1	1	0	1	1	0	B6
17	1	0	0	0	1	1	0	0	1	0	1	1	0	96
18	1	0	0	1	0	0	0	1	0	0	1	1	0	26
19	1	0	0	1	1	1	1	0	0	0	1	1	0	C6
20	1	0	1	0	1	1	0	0	1	0	1	1	0	B6
21	1	0	1	0	1	1	0	0	1	0	1	1	0	96
22	1	0	1	1	0	0	0	1	0	0	1	1	0	26
23	1	0	1	1	1	1	1	0	0	0	1	1	0	C6
24	1	1	0	0	0	1	0	1	1	0	1	1	0	B6
25	1	1	0	0	1	1	0	0	1	0	1	1	0	96
26	1	1	0	1	0	0	0	1	0	0	1	1	0	26
27	1	1	0	1	1	1	1	0	0	0	1	1	0	C6
28	1	1	1	0	0	1	0	1	1	0	1	1	0	B6
29	1	1	1	0	1	1	0	0	1	0	1	1	0	96
30	1	1	1	1	0	0	0	1	0	0	1	1	0	26
31	1	1	1	1	1	1	1	0	0	0	1	1	0	C6
	$M/\overline{IO}$	$E/\overline{NE}$	$D/\overline{C}$	OPREQ	$\overline{R}/W$	REI	DEI	$R/\overline{W}$	OPK	WOPC	EIPC	EIPD	WOPD	

TABLE 2

The current loop driver/receiver pair or the RS232 driver/receiver pair is selected by a hardwire jumper on the PC1001 board. The connection of these jumpers is described in Table 3, and shown in Figure 4 (2650 pin 40/FLAG, 2650 pin 1/SENSE).

SERIAL I/O DRIVER/RECEIVER MODE

2650 FUNCTION	JUMPER	DESCRIPTION
FLAG	A-B	CURRENT LOOP DRIVER
FLAG	A-C	RS232 DRIVER
SENSE	E-D	CURRENT LOOP RECEIVER
SENSE	F-D	RS232 RECEIVER

TABLE 3

PARALLEL I/O LATCHES AND RECEIVERS

The logic used to implement the two parallel I/O ports on the PC1001 is identical. The output ports are 7475 quad bistable latches (IC's 36, 37, 38, and 39), and are loaded when a Non-Extended write I/O instruction is executed (WRTC, WRTD). The input ports use 8T98 tri-state high speed hex inverters (IC's 40, 41, and 42), and are gated on the external data bus ($\overline{DBUS0}$ - $\overline{DBUS7}$) when a Non-Extended read I/O instruction is executed (REDC, REDD).

The control signals used to activate the tri-state gates (EIPD, and EIPC) are generated by the control line decode PROM (IC 35).

The control signals used to load the output latches are designated COPC and COPD, and have the following logic equations:

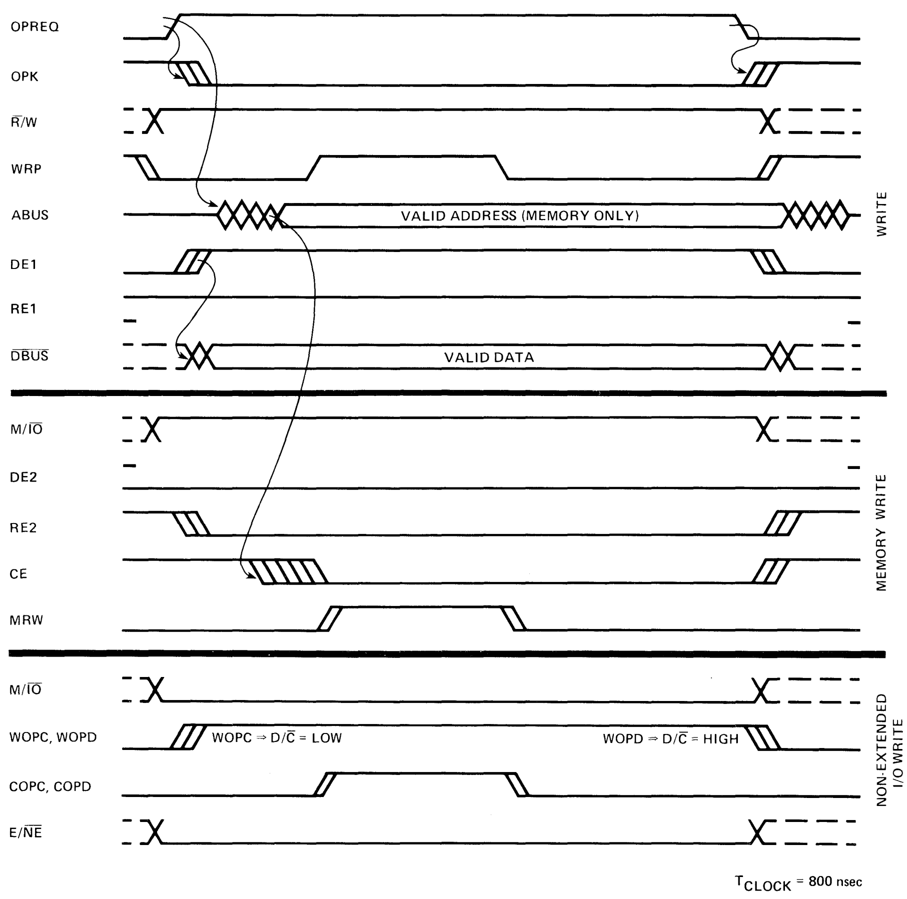
$$\mathrm{COPD} = \mathrm{WRP} \bullet \mathrm{WOPD}$$
$$\mathrm{COPC} = \mathrm{WRP} \bullet \mathrm{WOPC}$$

The WRP signal is the "write pulse" from the 2650, while the WOPD and WOPC signals are generated by the control line decode PROM (IC 35).

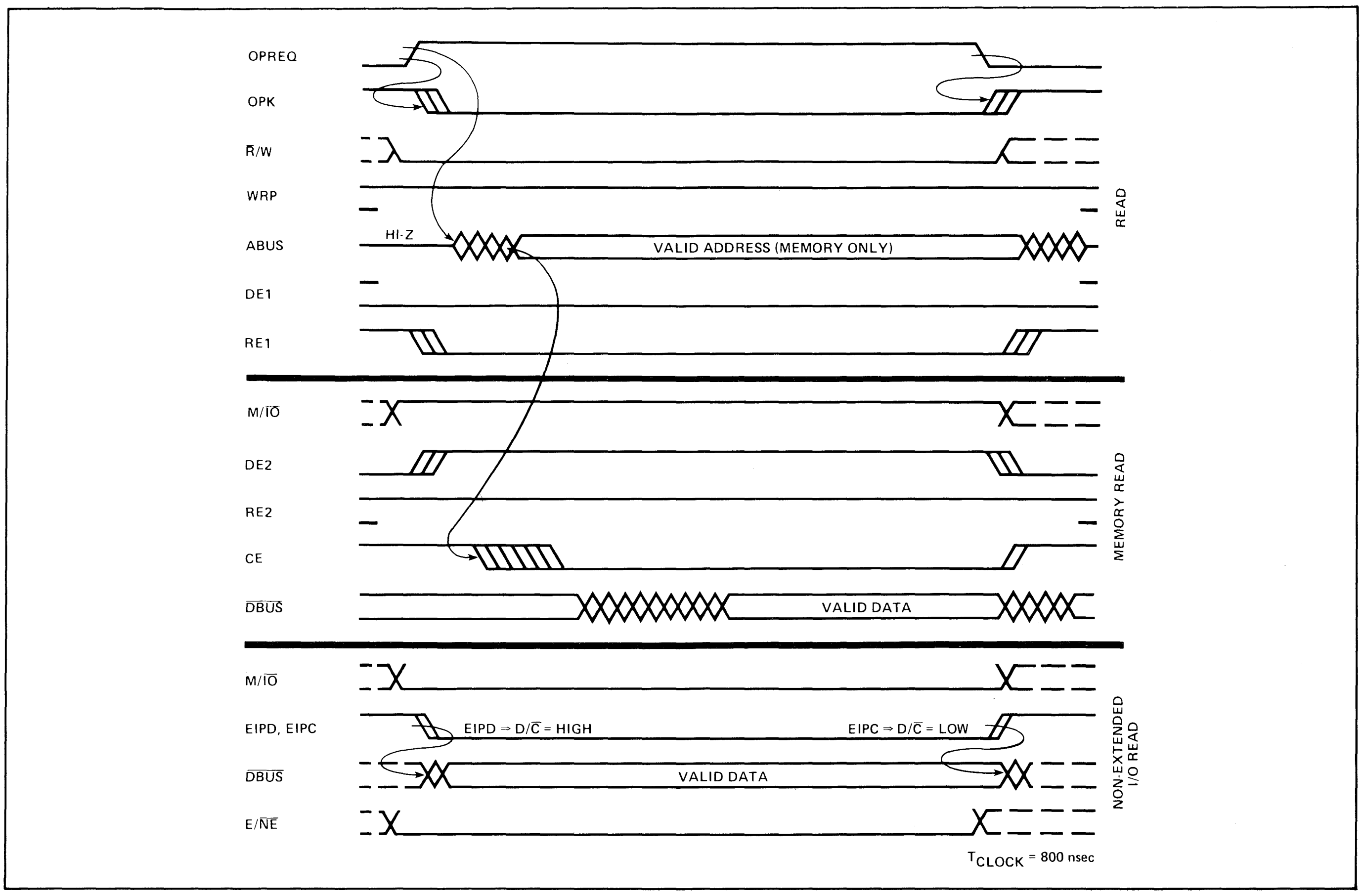
The output latches drive LED's on the PC1001 board. A logic ONE from the 2650 lights the corresponding LED. The output latches are loaded from the external data bus ($\overline{\mathrm{DBUS0}}$ - $\overline{\mathrm{DBUS7}}$), and to obtain the required inversion at the latch output (OPD0 - OPD7, and OPC0 - OPC7) the $\overline{\mathrm{Q}}$ pin is used.

APPENDIX

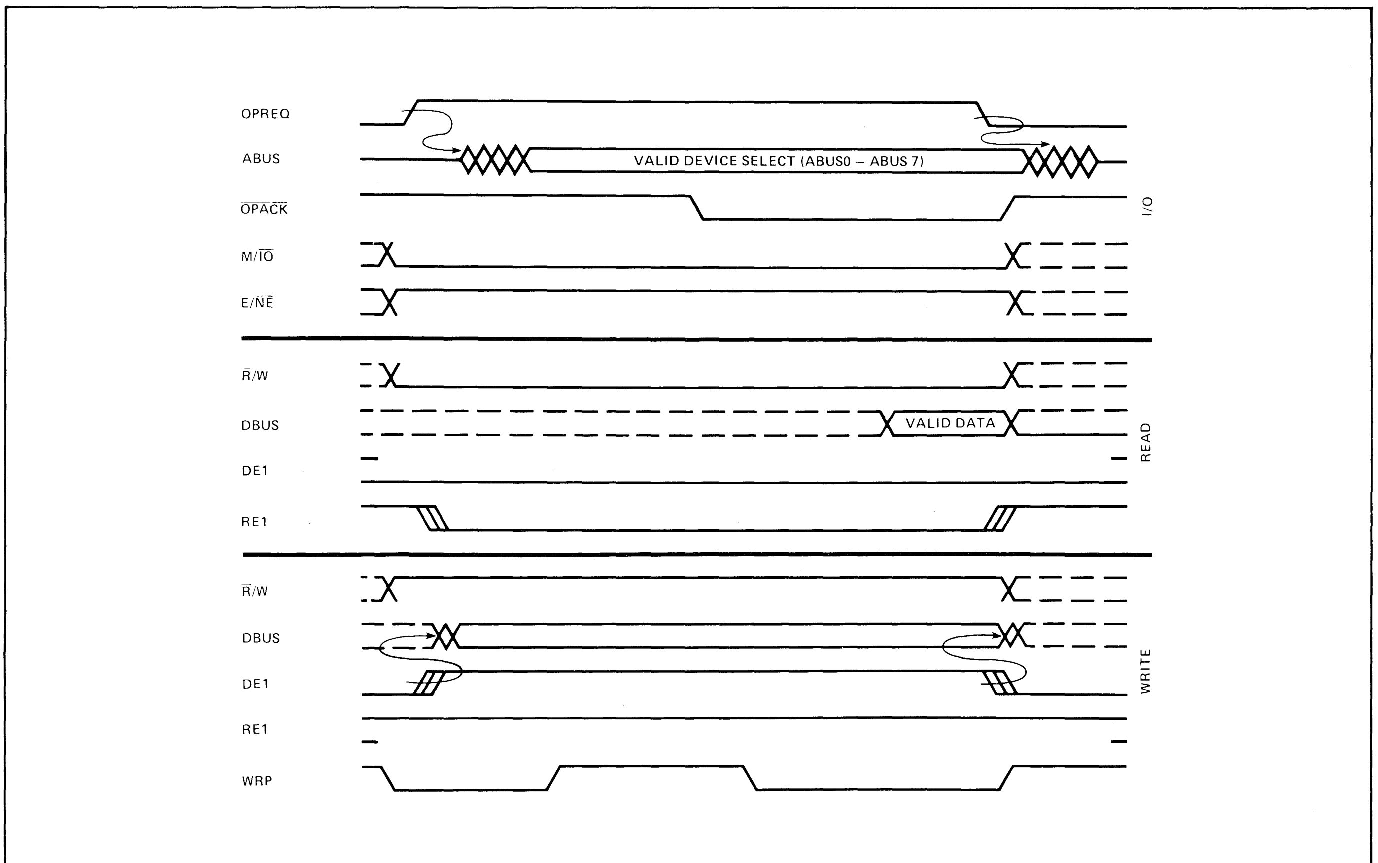
WRITE TIMING



READ TIMING



EXTENDED I/O TIMING



POWER REQUIREMENTS

+5 VOLT POWER SUPPLY:	±15 VOLT POWER SUPPLIES:
LINE REGULATION – 0.1%	LINE REGULATION – 0.1%
LOAD REGULATION – 0.1%	LOAD REGULATION – 0.1%
RIPPLE – 10 millivolts (MAX)	RIPPLE – 10 millivolts (MAX)
RESPONSE – 30μsec (MAX)	RESPONSE – 30μsec (MAX)
CURRENT – 2 amps	CURRENT – 50 milliamps

PARTS LIST

IC #	PART #	TYPE	QTY
28	7400	QUAD 2-INPUT NAND	1
29, 33	7408	QUAD 2-INPUT AND	2
19	7430	8-INPUT NAND	1
34	7438	QUAD 2-INPUT NAND OPEN COLLECTOR	1
36, 37, 38, 39	7475	QUAD BISTABLE LATCH	4
26	8T15	E1A DRIVER (RS232)	1
43	8T16	E1A RECEIVER (RS232)	1
1, 10 24, 25	8T26	QUAD BUS DRIVER/RECEIVER	4
40, 41, 42	8T98	HEX HIGH SPEED INVERTER	3
20	8250	1 OF 8 DECODER	1
6, 7, 8, 9 15, 16, 17, 18	2606	256X4 NMOS RAM	8
30	4049	HEX INVERTER (CMOS)	1
35	82S123	32X8 BIPOLAR PROM	1
2	82S129	PIPBUG PROM CK267	1
11	82S129	PIPBUG PROM CK268	1
3	82S129	PIPBUG PROM CK269	1
12	82S129	PIPBUG PROM CK270	1
4	82S129	PIPBUG PROM CK271	1
13	82S129	PIPBUG PROM CK272	1
5	82S129	PIPBUG PROM CK273	1
14	82S129	PIPBUG PROM CK274	1
23	2650	MICROPROCESSOR	1
27	MOTOROLA K1100A	XTAL OSCILLATOR	1
D20	IN914	DIODE	1
D1-D19	DIALCO 555-3007	LED INDICATOR	19
S1	GREYHILL 39-201	MINIATURE, PUSH BUTTON SWITCH	1
(IC 23)*	VERMON H23-20302	40-PIN DIP SOCKET	1
(IC 2, 3 4, 5, 11, 12, 13, 14)	AMPHENOL 821-25011-164	16-PIN DIP SOCKET	8

*#'s in parenthesis indicate the IC's that are plugged into the listed socket.

PARTS LIST (Continued)

IC #	PART #	TYPE	QTY
(IC 27)	AMPHENOL 821-25011-144	14-PIN DIP SOCKET	1
C1, C2 C3	230-1250-004-230	4.7μ FARAD CAP	3
–	EMCON 5021ES50RD104M	0.1μ FARAD CAP	45
C4		0.047μ FARAD CAP	1
R4	230-0910-332-230	51Kr, ¼ WATT RES	1
R3		10Kr, ¼ WATT RES	1
R7		7.4Kr, ¼ WATT REST	1
R5, R6	230-0910-297-230	1Kr, ¼ WATT RES	1
R1, R2	230-0910-282-230	220r, ¼ WATT RES	2
	AMPHENOL 225-804-50	100 PIN P.C. EDGE CONNECTOR	1

RS232C STANDARD CONNECTOR

The RS232 Electronic Industries Association (EIA) standard for "interface between terminals and communications equipment using serial binary data interchange" describes a commonly used signal definition and connector pin assignment. The table below lists the pin numbers and signal names most frequently used by. data terminals.

PIN #	DESCRIPTION
1	PROTECTIVE GROUND
2	TRANSMITTED DATA
3	RECEIVED DATA
5	CLEAR TO SEND
6	DATA SET READY
7	SIGNAL GROUND
8	RECEIVED LINE SIGNAL DETECTOR
20	DATA TERMINAL READY

Transmitted Data (pin 2) is received by the PC1001, therefore pin 2 of the RS232 connector is routed to the SENSE input of the 2650. Received Data (pin 3) is transmitted from the PC1001, therefore pin 3 of the RS232 is routed to the FLAG output of the 2650.

The signals on pins 5, 6, 8, and 20 are used between data terminals and communications MODEMs. Since the PC1001 does not provide these "handshake" lines they can be simulated by shorting them all together. In this configuration the Data Terminal Ready line drives the other 3 lines to the proper state for enabling the communication channel. This is not required for all data terminals (not teletypes), but is required for some.

Further information on RS232C specifications can be obtained from the EIA RS-232-C Standard available from the Electronic Industries Association in Washington D.C.

The type of connector commonly used for RS232 compatible data terminals is a 25-pin TRW Cinch type connector of the DB25 series.

TELETYPE CONNECTION

Connection to a teletype may be made at the terminal strip inside of the teletype. The pin numbers and signal names are listed in the table below.

PIN #	DESCRIPTION
6	RECEIVER – (TTY SERIAL IN –)
7	RECEIVER + (TTY SERIAL IN +)
3	TRANSMITTER – (TTY SERIAL OUT –)
4	TRANSMITTER + (TTY SERIAL OUT +)

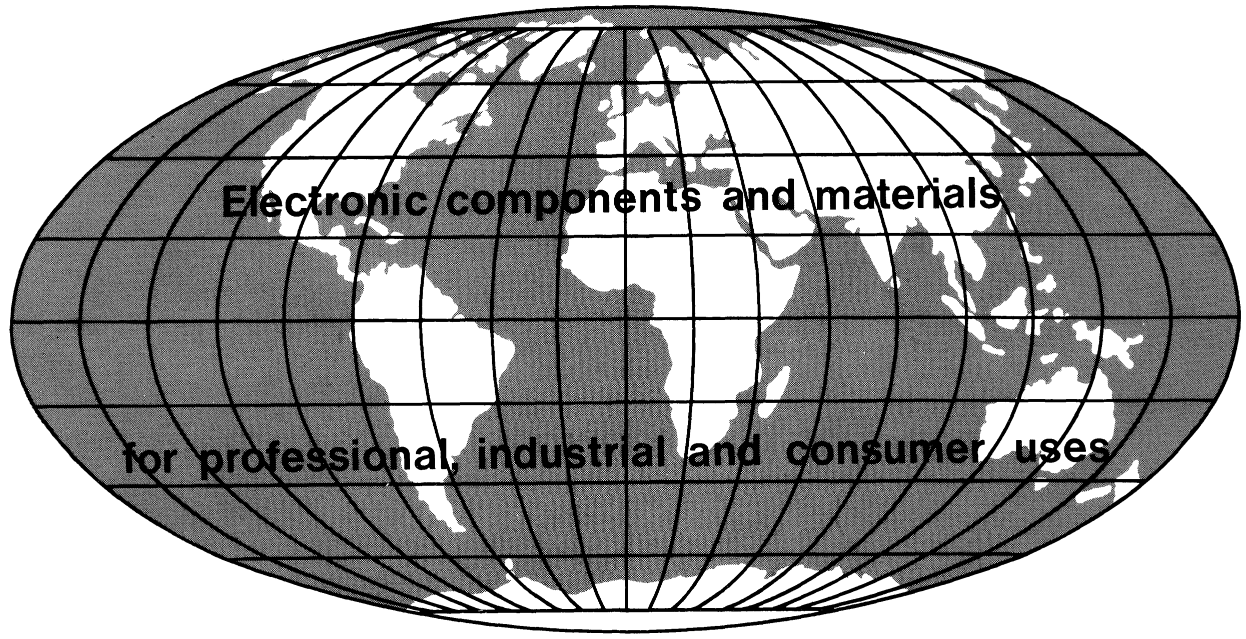
The teletype is a 20 milliamp current loop type of receiver and a contact closure type of transmitter. The PIPBUG debug program on the PC1001 board communicates with the teletype in a full duplex mode, echoing characters as they are received.

EDGE CONNECTOR SIGNAL LIST

PIN #	FUNCTION	PIN #	FUNCTION
1	GND	A	GND
2	GND	B	GND
3	NC*	C	NC
4	$\overline{\text{DBUS0}}$	D	OPD 0
5	$\overline{\text{DBUS1}}$	E	OPD 1
6	$\overline{\text{DBUS2}}$	F	OPD 2
7	$\overline{\text{DBUS3}}$	H	OPD 3
8	$\overline{\text{DBUS4}}$	J	OPD 4
9	$\overline{\text{DBUS5}}$	K	OPD 5
10	$\overline{\text{DBUS6}}$	L	OPD 6
11	$\overline{\text{DBUS7}}$	M	OPD 7
12	EIPD	N	COPD
13	D/$\overline{\text{C}}$	P	TTY SERIAL IN +
14	$\overline{\text{DMA}}$	R	TTY SERIAL IN −
15	E/$\overline{\text{NE}}$	S	TTY SERIAL OUT +
16	INTACK	T	TTY SERIAL OUT −
17	$\overline{\text{R}}$/W	U	RS232 GROUND
18	WRP	V	RS232 OUTPUT
19	RUN/$\overline{\text{WAIT}}$	W	TTY TAPE READER OUT −
20	OPREQ	X	TTY TAPE READER OUT +
21	M/$\overline{\text{IO}}$	Y	RS232 INPUT
22	$\overline{\text{OPACK}}$	Z	COPC
23	$\overline{\text{CLOCK}}$	a	OPC 0
24	$\overline{\text{OPEX}}$	b	OPC 1
25	$\overline{\text{RESET}}$	c	OPC 2
26	$\overline{\text{INTREQ}}$	d	OPC 3
27	$\overline{\text{PAUSE}}$	e	OPC 4
28	NC*	f	OPC 5
29	NC*	g	OPC 6
30	NC*	h	OPC 7
31	NC*	j	EIPC
32	NC*	k	IPD 0
33	ABUS 11	m	IPD 1
34	ABUS 13	n	IPD 2
35	ABUS 12	p	IPD 3
36	ABUS 14	r	IPD 4
37	ABUS 9	s	IPD 5
38	ABUS 10	t	IPD 6
39	ABUS 8	u	IPD 7
40	ABUS 7	v	IPC 0
41	ABUS 6	w	IPC 1
42	ABUS 5	x	IPC 2
43	ABUS 3	y	IPC 3
44	ABUS 0	z	IPC 4
45	ABUS 1	$\overline{\text{a}}$	IPC 5
46	ABUS 4	$\overline{\text{b}}$	IPC 6
47	ABUS 2	$\overline{\text{c}}$	IPC 7
48	+15V	$\overline{\text{d}}$	+15V
49	−15V	$\overline{\text{e}}$	−15V
50	+5V	$\overline{\text{f}}$	+5V

*NC = NO CONNECTION

TABLE 4


Electronic components and materials
for professional, industrial and consumer uses

from a world-wide Group of Companies

EUROPEAN SALES OFFICES

Austria: Österreichische Philips, Bauelemente Industrie G.m.b.H., Zieglergasse 6, Tel. 93 26 11, A-1072 WIEN.
Belgium: M.B.L.E., 80, rue des Deux Gares, Tel. 523 00 00, B-1070 BRUXELLES.
Denmark: Miniwatt A/S, Emdrupvej 115A, Tel. (01) 69 16 22, DK-2400 KØBENHAVN NV.
Finland: Oy Philips Ab, Elcoma Division, Kaivokatu 8, Tel. 1 72 71, SF-00100 HELSINKI 10.
France: R.T.C., La Radiotechnique-Compelec, 130 Avenue Ledru Rollin, Tel. 355-44-99, F-75540 PARIS 11.
Germany: Valvo, UB Bauelemente der Philips G.m.b.H., Valvo Haus, Burchardstrasse 19, Tel. (040) 3296-1, D-2 HAMBURG 1.
Greece: Philips S.A. Hellénique, Elcoma Division, 52, Av. Syngrou, Tel. 915 311, ATHENS.
Ireland: Philips Electrical (Ireland) Ltd., Newstead, Clonskeagh, Tel. 69 33 55, DUBLIN 14.
Italy: Philips S.p.A., Sezione Elcoma, Piazza IV Novembre 3, Tel. 2-6994, I-20124 MILANO.
Netherlands: Philips Nederland B.V., Afd. Elonco, Boschdijk 525, Tel. (040) 79 33 33, NL-4510 EINDHOVEN.
Norway: Electronica A.S., Vitaminveien 11, Tel. (02) 15 05 90, P. O. Box 29, Grefsen, OSLO 4.
Portugal: Philips Portuguesa S.A.R.L., Av. Eng. Duharte Pacheco 6, Tel. 68 31 21, LISBOA 1.
Spain: COPRESA S.A., Balmes 22, Tel. 301 63 12 BARCELONA 7.
Sweden: ELCOMA A.B., Lidingövägen 50, Tel. 08/67 97 80, S-10 250 STOCKHOLM 27.
Switzerland: Philips A.G., Elcoma Dept., Edenstrasse 20, Tel. 01/44 22 11, CH-8027 ZÜRICH.
Turkey: Türk Philips Ticaret A.S., EMET Department, Gümüssuyu Cad. 78-80, Tel. 45.32.50, Beyoglü, ISTANBUL.
United Kingdom: Mullard Ltd., Mullard House, Torrington Place, Tel. 01-580 6633, LONDON WC1E 7HD.

Printed in The Netherlands 2-76 9399 509 52161

signetics

MICROPROCESSOR

PIPBUG...SS50

signetics

PIPBUG | SS50

2650 MICROPROCESSOR
APPLICATIONS MEMO

INTRODUCTION

The PIPBUG program is provided as part of the 2650 PC1001 so that the user has immediately available to him the tools necessary to run programs on the 2650 microprocessor. Features include support of a user terminal, papertape load and dump, memory examine and alter, and breakpoints. The 2650 PC1001 card itself is described in detail in applications note SP 50.

DESCRIPTION

The PIPBUG program is started by pressing the reset button on the card. It outputs the user prompt character of '*'. A command is then entered, starting with an alpha character indicating the operation wanted, followed by any required parameters separated by spaces, and all terminated by a carriage return. The parameters must be given as hexadecimal numbers. Leading zeros are unnecessary. For example, '008F' and '8F' are the same address. The error message for an illegal command or parameter is '?', after which the user can enter a new command line. The delete key can be used to delete the previous character.

The program fits in the first 1K bytes of memory in the PROM. Also, the 63 bytes of RAM from location 1024 to 1087 are required for buffers and temporary storage. Locations 0 to 63 are part of the interrupt vector. To fit within 1K bytes the program uses subroutines with a maximum nested depth of three.

In the explanations of the commands CR means the carriage return key and LF means the line feed key. The symbol ƀ means there must be at least one space.

COMMANDS

I. Alter Memory — Aaaaa CR

Action: Outputs aaaaƀcc where 'aaaa' is a memory location and 'cc' is its content. User can respond with:

1) CR which ends the command
2) LF which will display the next memory location
3) nn CR which will replace 'cc' by 'nn' at location 'aaaa' and end the command
4) nn LF which will replace 'cc' by 'nn' and then display the next location.

II. Load from Papertape — L CR

Action: Will start reading papertape expecting blocks of data in the hex object format. In case of illegal characters, a BCC error, or a length error, the papertape will be stopped and the command ended with the standard error message.

At the end of a successful load, control is passed to the address in the EOF block. This would usually be back to the PIPBUG program.

III. Dump to Papertape — Dssssƀeeee CR

Action: Will punch a leader of 50 blanks and then output the contents of locations 'ssss' to 'eeee', inclusive, in hex object format. When done, the EOF block and a trailer of 50 blanks are punched.

IV. See and Set the Microprocessor Registers — Sn CR

Action: The parameter 'n' is in the range 0 to 8 and selects a particular register;

0 = register 0
1 = register 1 bank #0
2 = register 2 bank #0
3 = register 3 bank #0
4 = register 1 bank #1
5 = register 2 bank #1
6 = register 3 bank #1
7 = PSW upper
8 = PSW lower

The contents will be displayed. The user can respond with:

1) CR which ends the command
2) LF which displays the next register's content
3) nn CR which resets the register to 'nn' and ends the command
4) nn LF which resets the register to 'nn' and displays the next register's content

V. Go To — Gaaaa CR

Action: Control will be transferred to location 'aaaa' after restoring the register contents.

VI. Clear Breakpoints — Ci CR

Action: Will clear the ith breakpoint. If the ith breakpoint is not set, gives error message.

VII. Set Breakpoints — Biƀaaaa CR

Action: Will set the ith breakpoint at the address 'aaaa'. The current firmware supports two breakpoints.

BREAKPOINTS

Breakpoints are a way to get a snapshot of the program and microprocessor's status immediately prior to executing at the breakpoint address. PIPBUG allows two breakpoints to be set. So i equals 1 or 2 in the breakpoint commands.

BREAKPOINTS (Continued)

Setting a breakpoint at location '1053' with the command 'B1 1053' causes the two bytes of program at '1053' and '1054' to be stored in a table in PIPBUG's RAM area. They are replaced by the two byte instruction 'ZBRR *BKP1'. At location 'BKP1' in the interrupt vector is the address of the 1st breakpoint handling routine. There is a separate routine for the 2nd breakpoint.

When the user program executes the instruction at location '1053', the ZBRR instruction jumps to the breakpoint routine. This routine first saves the microprocessor registers, then restores the two bytes of user program to locations '1053' and '1054', prints the breakpoint address '1053', and finally jumps to PIPBUG. Now the user can use the See command to examine the microprocessor registers.

Since the breakpoints are software implemented and are cleared when reached, there will not be another breakpoint when the user program is re-executed. It must be explicitly re-set with the Breakpoint command. Breakpoints will remain in memory until executed or explicitly cleared with the Clear command.

SUGGESTIONS ON USING

Having written and assembled a program, the user has a papertape containing the object code for the program. The Load command is used to read the code into the RAM of the 2650 PC1001 card. In the operand field of the END directive of the program, the user should put blanks or a zero, so that after reading the tape PIPBUG restarts itself.

Most commonly the loaded program is still under development. The user wants to run and test only parts of the program. He can use the Goto and Breakpoint commands to isolate the particular code sequence. The two breakpoints can be set at the normal and error exits of the code. Using the Goto command the user then transfers control to the starting address of the code. Remember that the microprocessor's registers can be pre-set using the See command.

If there is a bug, the user can make machine language patches to the program with the Alter command. Great care should be taken when doing this, since assemblers are more methodical than people. The Dump command can be used to save on papertape the program and all patches so that the debugging can be continued at some later time.

SUMMARY

A	Alter memory
B	Set Breakpoint
C	Clear Breakpoint
D	Dump memory to papertape
G	Goto address
L	Load memory from papertape
S	See and alter registers

APPENDIX

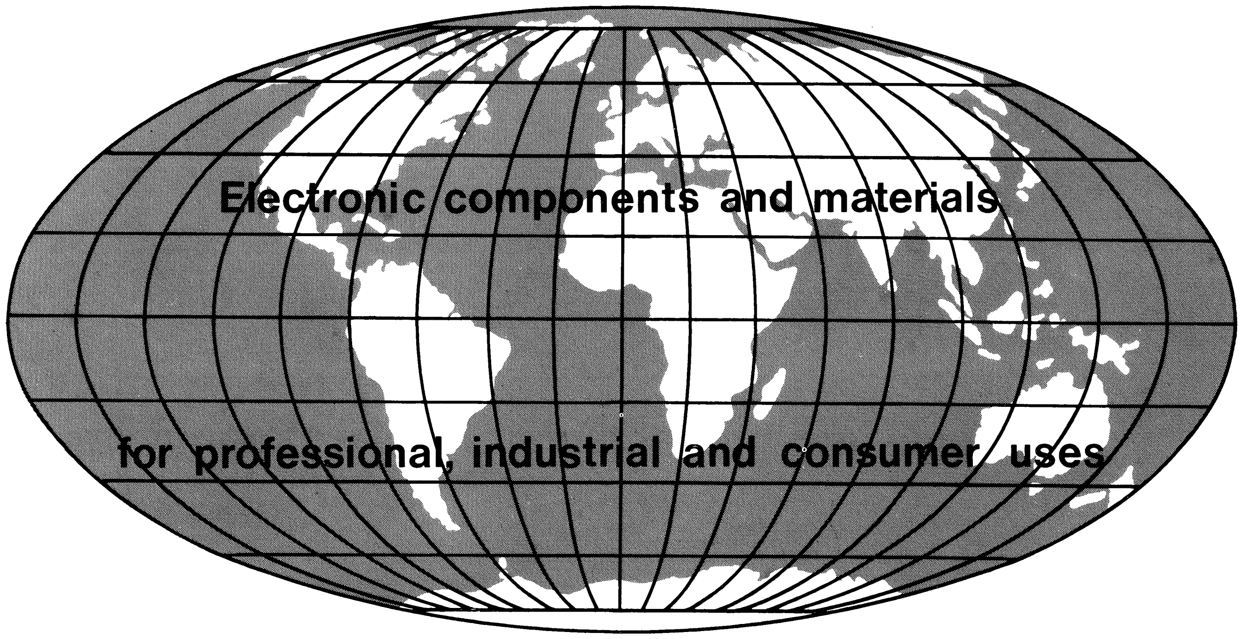
```
PIP ASSEMBLER VERSION 3 LEVEL 1                                      PAGE    1

LINE ADDR B1 B2 B3 B4 ERR SOURCE

  1 0001                  P      EQU          1
  2 0002                  N      EQU          2
  3 0000                  Z      EQU          0
  4 0002                  LCOM    EQU          H'02'              LOGICAL COMPARE
  5 0001                  CAR     EQU          H'01'              CARRY
  6 0080                  SENS    EQU          H'80'              SENSE
  7 0040                  FLAG    EQU          H'40'              FLAG
  8 0020                  II      EQU          H'20'              INTERRUPT INHIB
  9 0020                  IDC     EQU          H'20'              INTER DIGIT CAR
 10 0004                  OVF     EQU          H'04'              OVEFFLOW
 11 0000                  R0      EQU          0
 12 0001                  R1      EQU          1
 13 0002                  R2      EQU          2
 14 0003                  R3      EQU          3
 15 0003                  UN      EQU          3
 16 0000                  EQ      EQU          0
 17 0002                  LT      EQU          2
 18 0001                  GT      EQU          1
 19 0008                  WC     EQU          H'08'
 20 0010                  RS     EQU          H'10'
 21 0020                  SPAC   EQU          H'20'
 22 0001                  BMAX   EQU          1           NO. BKPTS - 1
 23 007F                  DELE   EQU          H'7F'
 24 000D                  CR     EQU          13
 25 000A                  LF     EQU          10
 26 0014                  BLEN   EQU          20
 27 003A                  STAR   EQU          A':'
 28                       *
 29                              ORG          0
 30 0000 07 3F            INIT   LODI,R3      63          ZERO MARK VECTOR AND O
 31 0002 20                      EORZ         R0
 32 0003 CF 44 00         AINI   STRA,R0      COM,R3,-
 33 0006 5B 7B                   BRNR,R3      AINI
 34 0008 04 77                   LODI,R0      H'77'
 35 000A CC 04 09                STRA,R0      XGOT        LOAD THE RAM CODE TO S
 36 000D 04 1B                   LODI,R0      H'1B'
 37 000F CC 04 0B                STRA,R0      XGOT+2
 38 0012 04 80                   LODI,R0      H'80'
 39 0014 CC 04 0C                STRA,R0      XGOT+3
 40 0017 1B 09                   BCTR,UN      MBUG
 41 0019 01 60            VEC    ACON         BK01        BREAKPOINT VECTOR
 42 001B 01 6E                   ACON         BK02
 43                       *
 44                       * COMMAND HANDLER
 45 001D 04 3F            EBUG   LODI,R0      A'?'        ERROR RETURN FOR ALL R
 46 001F 3F 02 B4                BSTA,UN      COUT
 47 0022 75 FF            MBUG   CPSL         H'FF'       START OF CMD LOOP, RES
 48 0024 3F 00 8A                 BSTA,UN      CRLF
 49 0027 04 2A                   LODI,R0      A'*'
 50 0029 3F 02 B4                BSTA,UN      COUT
 51 002C 3B 2D                   BSTR,UN      LINE         DONT CARE IF THERE IS
 52 002E 20                      EORZ         R0
 PIP ASSEMBLER VERSION 3 LEVEL 1                                             PAGE    2

LINE ADDR B1 B2 B3 B4 ERR SOURCE

  53 002F CC 04 27                  STRA,R0     BPTR
  54 0032 0C 04 13                  LODA,R0     BUFF
  55 0035 E4 41                     COMI,R0     A'A'
  56 0037 1C 00 AB                   BCTA,EQ     ALTE
  57 003A E4 42                     COMI,R0     A'B'
  58 003C 1C 01 E5                   BCTA,EQ    BKPT
  59 003F E4 43                      COMI,R0     A'C'
  60 0041 1C 01 CA                  BCTA,EQ     CLR
  61 0044 E4 44                     COMI,R0     A'D'
  62 0046 1C 03 10                   BCTA,EQ    DUMP
  63 0049 E4 47                     COMI,R0      A'G'
  64 004B 1C 01 3A                   BCTA,EQ    GOTO
  65 004E E4 4C                     COMI,R0     A'L'
  66 0050 1C 03 B5                   BCTA,EQ     LOAD
  67 0053 E4 53                     COMI,R0     A'S'
  68 0055 1C 00 F4                   BCTA,EQ    SREG
  69 0058 1F 00 1D                  BCTA,UN     EBUG
  70                         * INPUT A CMD LINE INTO BUFFER
  71                         * CODE IS 1=CR  2=LF  3=MSG+CR  4=MSG+LF
  72 005B 07 FF              LINE   LODI,R3     -1
  73 005D CF 04 27                  STRA,R3     BPTR
  74 0060 E7 14              LLIN   COMI,R3     BLEN
  75 0062 18 19                     BCTR,EQ     ELIN         ON BUFFER OVERFLOW FOR
  76 0064 3F 02 86                  BSTA,UN     CHIN         GET CHAR
  77 0067 E4 7F                     COMI,R0     DELE
  78 0069 98 0E                     BCFR,EQ     ALIN
  79 006B E7 FF                     COMI,R3     -1           ECHO AND BACK PTR
  80 006D 18 71                     BCTR,EQ     LLIN
  81 006F 0F 64 13                  LODA,R0     BUFF,R3
  82 0072 3F 02 B4                  BSTA,UN     COUT
  83 0075 A7 01                     SUBI,R3     1
  84 0077 1B 67                     BCTR,UN     LLIN
  85 0079 E4 0D              ALIN   COMI,R0     CR
  86 007B 98 18                     BCFR,EQ     BLIN
  87 007D 05 01              ELIN   LODI,R1     1
  88 007F 03                 CLIN   LODZ        R3
  89 0080 1A 02                     BCTR,N      DLIN
  90 0082 85 02                     ADDI,R1     2
  91 0084 CD 04 2A           DLIN   STRA,R1     CODE
  92 0087 CF 04 29                  STRA,R3     CNT
  93 008A 04 0D              CRLF   LODI,R0     CR
  94 008C 3F 02 B4                  BSTA,UN     COUT
  95 008F 04 0A                     LODI,R0     LF
  96 0091 3F 02 B4                  BSTA,UN     COUT
  97 0094 17                        RETC,UN
  98 0095 05 02              BLIN   LODI,R1     2
  99 0097 E4 0A                     COMI,R0     LF
 100 0099 18 64                     BCTR,EQ     CLIN
 101 009B CF 24 13                  STRA,R0     BUFF,R3,+    STROE CHAR AND ECHO
 102 009E 3F 02 B4                  BSTA,UN     COUT
 103 00A1 1F 00 60                  BCTA,UN     LLIN
 104                         *
```

```
PIP ASSEMBLER VERSION 3 LEVEL 1                                    PAGE    3

LINE ADDR B1 B2 B3 B4 ERR SOURCE

                          * SUBR THAT STORES DOUBLE PRECISION INTO TEMP
     00A4 CD 04 0D        STRT    STRA,R1     TEMP
     00A7 CE 04 0E                STRA,R2     TEMP+1
     00AA 17                      RETC,UN
                          * DISPLAY AND ALTER MEMORY
     00AB 3F 02 DB        ALTE    BSTA,UN     GNUM
     00AE 3B 74           LALT    BSTR,UN     STRT
     00B0 3F 02 69                BSTA,UN     BOUT
     00B3 0D 04 0E                LODA,R1     TEMP+1
     00B6 3F 02 69                BSTA,UN     BOUT
     00B9 3F 03 5B                BSTA,UN     FORM
     00BC 0D 84 0D                LODA,R1     *TEMP          DISPLAY CONTENT
     00BF 3F 02 69                BSTA,UN     BOUT
     00C2 3F 03 5B                BSTA,UN     FORM
     00C5 3F 00 5B                BSTA,UN     LINE
     00C8 0C 04 2A                LODA,R0     CODE
     00CB E4 02                   COMI,R0     2
     00CD 1E 00 22                BCTA,LT     MBUG
     00D0 18 11                   BCTR,EQ     DALT
     00D2 CC 04 11        CALT    STRA,R0     TEMR
     00D5 3F 02 DB                BSTA,UN     GNUM
     00D8 CE 84 0D                STRA,R2     *TEMP          UPDATE CONTENTS
     00DB 0C 04 11                LODA,R0     TEMR
     00DE E4 04                   COMI,R0     4
     00E0 9C 00 22                BCFA,EQ     MBUG
     00E3 06 01           DALT    LODI,R2     1              INCR CURRENT ADDRESS
     00E5 8E 04 0E                ADDA,R2     TEMP+1
     00E8 05 00                   LODI,R1     0
     00EA 77 08                   PPSL        WC
     00EC 8D 04 0D                ADDA,R1     TEMP
     00EF 75 08                   CPSL        WC
     00F1 1F 00 AE                BCTA,UN     LALT
                          * SELECTIVELY DISPLAY AND ALTER REGISTERS
     00F4 3F 02 DB        SREG    BSTA,UN     GNUM           GET INDEX OF REG
     00F7 E6 08           LSRE    COMI,R2     8              CHECK RANGE
     00F9 1D 00 1D                BCTA,GT     EBUG
     00FC CE 04 11                STRA,R2     TEMR
     00FF 0E 64 00                LODA,R0     COM,R2         DISPLAY CONTENTS
     0102 C1                      STRZ        R1
     0103 3F 02 69                BSTA,UN     BOUT
     0106 3F 03 5B                BSTA,UN     FORM
     0109 3F 00 5B                BSTA,UN     LINE
     010C 0C 04 2A                LODA,R0     CODE
     010F E4 02                   COMI,R0     2
     0111 1E 00 22                BCTA,LT     MBUG           CR
     0114 18 1C                   BCTR,EQ     CSRE           LF
     0116 CC 04 0F        ASRE    STRA,R0     TEMQ           UPDATE CONTENTS, THEN
     0119 3F 02 DB                BSTA,UN     GNUM
     011C 02                      LODZ        R2
     011D 0E 04 11                LODA,R2     TEMR
     0120 CE 64 00                STRA,R0     COM,R2
     0123 E6 08                   COMI,R2     8              MUST UPDATE PSW LOWER
```

```
  PIP ASSEMBLER VERSION 3 LEVEL 1                                     PAGE    4

LINE ADDR B1 B2 B3 B4 ERR SOURCE

     0125 98 03                  BCFR,EQ     BSRE
     0127 CC 04 0A               STRA,R0     XGOT+1
     012A 0C 04 0F         BSRE  LODA,R0     TEMQ
     012D E4 03                  COMI,R0     3
     012F 1C 00 22               BCTA,EQ     MBUG
     0132 0E 04 11         CSRE  LODA,R2     TEMR
     0135 86 01                  ADDI,R2     1
     0137 1F 00 F7               BCTA,UN     LSRE
                           *  GOTO ADDRESS
     013A 3F 02 DB         GOTO  BSTA,UN     GNUM
     013D 3F 00 A4               BSTA,UN     STRT        PUT ADDR IN RAM
     0140 0C 04 07               LODA,R0     COM+7
     0143 92                     LPSU
     0144 0D 04 01               LODA,R1     COM+1       BANK ZERO
     0147 0E 04 02               LODA,R2     COM+2
     014A 0F 04 03               LODA,R3     COM+3
     014D 77 10                  PPSL        RS        BANK ONE
     014F 0D 04 04               LODA,R1     COM+4
     0152 0E 04 05               LODA,R2     COM+5
     0155 0F 04 06               LODA,R3     COM+6
     0158 0C 04 00               LODA,R0     COM
     015B 75 FF                  CPSL        H'FF'
     015D 1F 04 09               BCTA,UN     XGOT      AND BCTA,UN $TEMP
                           *
                           *BREAKPOINT RUNTIME CODE
     0160 CC 04 00         BK01  STRA,R0     COM         ENTRY FOR BKPT-1 VIA V
     0163 13                     SPSL
     0164 CC 04 08               STRA,R0     COM+8
     0167 CC 04 0A               STRA,R0     XGOT+1      IN RAM FOR REG RESTORE
     016A 04 00                  LODI,R0     0           BKPT INDEX
     016C 1B 0C                  BCTR,UN     BKEN
     016E CC 04 00         BK02  STRA,R0     COM         ENTRY FOR BKPT-2
     0171 13                     SPSL
     0172 CC 04 08               STRA,R0     COM+8
     0175 CC 04 0A               STRA,R0     XGOT+1      IN RAM FOR REG RESTORE
     0178 04 01                  LODI,R0     1
     017A CC 04 11         BKEN  STRA,R0     TEMR
     017D 12                     SPSU
     017E CC 04 07               STRA,R0     COM+7
     0181 77 10                  PPSL        RS
     0183 CD 04 04               STRA,R1     COM+4
     0186 CE 04 05               STRA,R2     COM+5
     0189 CF 04 06               STRA,R3     COM+6
     018C 75 10                  CPSL        RS          FORCE TO BANK ZERO
     018E CD 04 01               STRA,R1     COM+1
     0191 CE 04 02               STRA,R2     COM+2
     0194 CF 04 03               STRA,R3     COM+3
     0197 0E 04 11               LODA,R2     TEMR
     019A 3B 0F                  BSTR,UN     CLBK
     019C 0D 04 0D               LODA,R1     TEMP        PRINT BKPT ADDR
     019F 3F 02 69               BSTA,UN     BOUT
     01A2 0D 04 0E               LODA,R1     TEMP+1
```

```
 PIP ASSEMBLER VERSION 3 LEVEL 1                                   PAGE    5

LINE ADDR B1 B2 B3 B4 ERR SOURCE

 209 01A5 3F 02 69              BSTA,UN     BOUT
 210 01A8 1F 00 22              BCTA,UN     MBUG
 211                    * SUBR TO CLEAR A BKPT    LIKE MANY SUBR HAS REL ADDR
 212 01AB 20            CLBK    EORZ        R0
 213 01AC CE 64 2D              STRA,R0     MARK,R2
 214 01AF 0E 64 33              LODA,R0     HADR,R2
 215 01B2 CC 04 0D              STRA,R0     TEMP
 216 01B5 0E 64 35              LODA,R0     LADR,R2
 217 01B8 CC 04 0E              STRA,R0     TEMP+1
 218 01BB 0E 64 2F              LODA,R0     HDAT,R2
 219 01BE CC 84 0D              STRA,R0    *TEMP
 220 01C1 0E 64 31              LODA,R0     LDAT,R2
 221 01C4 07 01                 LODI,R3     1
 222 01C6 CF E4 0D              STRA,R0    *TEMP,R3
 223 01C9 17                    RETC,UN
 224                    * BREAK POINT      MARK INDICATES IF SET
 225                    * HADR +LADR IS BKPT ADDR.   HDAT + LDAT  IS TWO BYTE
 226 01CA 3B 0B         CLR     BSTR,UN     NOK
 227 01CC 0E 64 2D              LODA,R0     MARK,R2      CLEAR IT IF SET
 228 01CF 1C 00 1D              BCTA,Z      EBUG
 229 01D2 3B 57                 BSTR,UN     CLBK
 230 01D4 1F 00 22              BCTA,UN     MBUG
 231 01D7 3F 02 DB      NOK     BSTA,UN     GNUM        CHECK RANGE ON BKPT NUMB
 232 01DA A6 01                 SUBI,R2     1
 233 01DC 1E 02 50              BCTA,N      ABRT
 234 01DF E6 01                 COMI,R2     BMAX
 235 01E1 1D 02 50              BCTA,GT     ABRT
 236 01E4 17                    RETC,UN
 237 01E5 3B 70         BKPT    BSTR,UN     NOK          SET BKPT AND CLR ANY E
 238 01E7 0E 64 2D              LODA,R0     MARK,R2
 239 01EA BC 01 AB              BSFA,Z      CLBK        CLEAR EXISTING
 240 01ED CE 04 11              STRA,R2     TEMR
 241 01F0 3F 02 DB              BSTA,UN     GNUM         GET BKPT ADDR
 242 01F3 3F 00 A4              BSTA,UN     STRT         SUBR TO STORE R1-R2 IN
 243 01F6 0F 04 11              LODA,R3     TEMR
 244 01F9 02                    LODZ        R2
 245 01FA CF 64 35              STRA,R0     LADR,R3
 246 01FD 01                    LODZ        R1
 247 01FE CF 64 33              STRA,R0     HADR,R3
 248 0201 0C 84 0D              LODA,R0    *TEMP         SAVE CONTENTS
 249 0204 CF 64 2F              STRA,R0     HDAT,R3
 250 0207 05 9B                 LODI,R1     H'9B'        = ZBRR
 251 0209 CD 84 0D              STRA,R1    *TEMP
 252 020C 06 01                 LODI,R2     1
 253 020E 0E E4 0D              LODA,R0    *TEMP,R2
 254 0211 CF 64 31              STRA,R0     LDAT,R3
 255 0214 0F 62 22              LODA,R0     DISP,R3
 256 0217 CE E4 0D              STRA,R0    *TEMP,R2
 257 021A 04 FF                 LODI,R0     -1
 258 021C CF 64 2D              STRA,R0     MARK,R3
 259 021F 1F 00 22              BCTA,UN     MBUG
 260 0222 99            DISP    DATA        VEC+H'80'
 PIP ASSEMBLER VERSION 3 LEVEL 1                                        PAGE    6

LINE ADDR B1 B2 B3 B4 ERR SOURCE

 261 0223 9B                    DATA        VEC+H'80'+2
 262                      *
 263                      * INPUT TWO HEX CHARS AND FORM  AS BYTE IN R1
 264 0224 3F 02 86        BIN     BSTA,UN     CHIN
 265 0227 3B 1D                   BSTR,UN     LKUP
 266 0229 D3                      RRL,R3
 267 022A D3                      RRL,R3
 268 022B D3                      RRL,R3
 269 022C D3                      RRL,R3
 270 022D CF 04 12                STRA,R3     TEMS
 271 0230 3F 02 86                BSTA,UN     CHIN
 272 0233 3B 11                   BSTR,UN     LKUP
 273 0235 6F 04 12                IORA,R3     TEMS
 274 0238 03                      LODZ        R3
 275 0239 C1                      STRZ        R1
 276 023A 3B 01                   BSTR,UN     CBCC
 277 023C 17                      RETC,UN
 278                      * CALCULATE THE BCC CHAR,  EOR AND THEN ROTATE LEFT
 279 023D 01              CBCC    LODZ        R1
 280 023E 2C 04 2C                EORA,R0     BCC
 281 0241 D0                      RRL,R0
 282 0242 CC 04 2C                STRA,R0     BCC
 283 0245 17                      RETC,UN
 284                      * LOOKUP ASCII CHAR IN HEX VALUE TABLE
 285 0246 07 10           LKUP    LODI,R3     16
 286 0248 EF 42 59        ALKU     COMA,R0     ANSI,R3,-
 287 024B 14                      RETC,EQ
 288 024C E7 01                   COMI,R3     1
 289 024E 9A 78                   BCFR,LT     ALKU
 290                      * ABORT EXIT FROM ANY LEVEL OF SUBR
 291                      *  USE RAS PTR SINCE POSSIBLE BKPT PROG USING IT
 292 0250 0C 04 07        ABRT    LODA,R0     COM+7
 293 0253 64 40                   IORI,R0     H'40'
 294 0255 12                      SPSU
 295 0256 1F 00 1D                BCTA,UN     EBUG
 296 0259 30 31 32 33     ANSI     DATA        A'0123456789ABCDEF
          34 35 36 37
          38 39 41 42
          43 44 45 46
 297                      * BYTE IN R1 OUTPUT IN HEX
 298 0269 CD 04 12        BOUT    STRA,R1     TEMS
 299 026C 3B 4F                   BSTR,UN     CBCC
 300 026E 51                      RRR,R1
 301 026F 51                      RRR,R1
 302 0270 51                      RRR,R1
 303 0271 51                      RRR,R1
 304 0272 45 0F                   ANDI,R1     H'0F'
 305 0274 0D 62 59                LODA,R0     ANSI,R1
 306 0277 3F 02 B4                BSTA,UN     COUT
 307 027A 0D 04 12                LODA,R1     TEMS
 308 027D 45 0F                   ANDI,R1     H'0F'
 309 027F 0D 62 59                LODA,R0     ANSI,R1
```

```
PIP ASSEMBLER VERSION 3 LEVEL 1                                        PAGE    7

LINE ADDR B1 B2 B3 B4 ERR SOURCE

 310 0282 3F 02 B4                     BSTA,UN     COUT
 311 0285 17                           RETC,UN
 312                            * 110 BAUD INPUT FOR PAPERTAPE AND CHAR   1MHZ CLOCK
 313 0286 77 10                 CHIN    PPSL        RS
 314 0288 04 80                        LODI,R0     H'80'        ENABLE TAPE READER
 315 028A B0                           WRTC,R0
 316 028B 05 00                        LODI,R1     0
 317 028D 06 08                        LODI,R2     8
 318 028F 12                    ACHI   SPSU
 319 0290 1A 74                        BCTR,LT     CHIN         LOOK FOR START BIT
 320 0292 20                           EORZ        R0
 321 0293 B0                           WRTC,R0                  DISABLE TAPE READER
 322 0294 3B 17                        BSTR,UN     DLY
 323 0296 3B 10                 BCHI   BSTR,UN     DLAY         WAIT TO MIDDLE OF DATA
 324 0298 12                           SPSU
 325 0299 44 80                        ANDI,R0     H'80'        MOVE BIT 7 OF R0 INTO
 326 029B 51                           RRR,R1
 327 029C 61                           IORZ        R1
 328 029D C1                           STRZ        R1
 329 029E FA 76                        BDRR,R2     BCHI
 330 02A0 3B 06                        BSTR,UN     DLAY
 331 02A2 45 7F                        ANDI,R1     H'7F'        DELETE PARITY BIT
 332 02A4 01                           LODZ        R1
 333 02A5 75 18                        CPSL        RS+WC
 334 02A7 17                           RETC,UN
 335                            * DELAY FOR ONE BIT TIME
 336 02A8 20                    DLAY   EORZ        R0
 337 02A9 F8 7E                        BDRR,R0     $
 338 02AB F8 7E                        BDRR,R0     $
 339 02AD F8 7E                 DLY    BDRR,R0     $
 340 02AF 04 E5                        LODI,R0     H'E5'
 341 02B1 F8 7E                        BDRR,R0     $
 342 02B3 17                           RETC,UN
 343                            *
 344 02B4 77 10                 COUT   PPSL        RS
 345 02B6 76 40                        PPSU        FLAG
 346 02B8 C2                           STRZ        R2
 347 02B9 05 08                        LODI,R1     8
 348 02BB 3B 6B                        BSTR,UN     DLAY
 349 02BD 3B 69                        BSTR,UN     DLAY
 350 02BF 74 40                        CPSU        FLAG
 351 02C1 3B 65                 ACOU    BSTR,UN     DLAY
 352 02C3 52                           RRR,R2
 353 02C4 1A 04                        BCTR,LT     ONE
 354 02C6 74 40                        CPSU        FLAG
 355 02C8 1B 02                        BCTR,UN     ZERO
 356 02CA 76 40                 ONE    PPSU        FLAG
 357 02CC F9 73                 ZERO   BDRR,R1     ACOU
 358 02CE 3B 58                        BSTR,UN     DLAY
 359 02D0 76 40                        PPSU        FLAG
 360 02D2 75 10                        CPSL        RS
 361 02D4 17                           RETC,UN
 PIP ASSEMBLER VERSION 3 LEVEL 1                                     PAGE    8

LINE ADDR B1 B2 B3 B4 ERR SOURCE

                          *
                          * GET A NUMBER FROM THE BUFFER INTO R1 - R2
     02D5 0C 04 2A        DNUM   LODA,R0     CODE
     02D8 18 07                  BCTR,Z      LNUM        SKIP SPACES UNTIL REAC
     02DA 17                     RETC,UN                 OR SPACE ENDING NUMBER
     02DB 20              GNUM   EORZ        R0
     02DC C1                     STRZ        R1
     02DD C2                     STRZ        R2
     02DE CC 04 2A               STRA,R0     CODE
     02E1 0F 04 27        LNUM   LODA,R3     BPTR
     02E4 EF 04 29               COMA,R3     CNT         CHECK FOR E O B
     02E7 14                     RETC,EQ
     02E8 0F 24 13                LODA,R0      BUFF,R3,+   GET CHAR
     02EB CF 04 27               STRA,R3     BPTR
     02EE E4 20                  COMI,R0     SPAC
     02F0 18 63                  BCTR,EQ     DNUM
     02F2 3F 02 46        BNUM   BSTA,UN     LKUP
     02F5 04 0F           CNUM   LODI,R0     H'0F'       R1=AB R2=DD
     02F7 D2                     RRL,R2
     02F8 D2                     RRL,R2
     02F9 D2                     RRL,R2
     02FA D2                     RRL,R2
     02FB 42                     ANDZ        R2
     02FC D1                     RRL,R1
     02FD D1                     RRL,R1
     02FE D1                     RRL,R1
     02FF D1                     RRL,R1
     0300 45 F0                  ANDI,R1     H'F0'
     0302 46 F0                  ANDI,R2     H'F0'        R0=C R1=B0 R2=D0 R3=V
     0304 61                     IORZ        R1
     0305 C1                      STRZ       R1
     0306 03                     LODZ        R3
     0307 62                     IORZ        R2
     0308 C2                     STRZ        R2          R1=BC R2=DV
     0309 04 01                   LODI,R0     1
     030B CC 04 2A               STRA,R0     CODE
     030E 1B 51                  BCTR,UN     LNUM
                          * DUMP TO PAPER TAPE IN OBJECT FORMAT
     0310 3B 49           DUMP   BSTR,UN     GNUM        START ADDRESS
     0312 3F 00 A4               BSTA,UN     STRT        SUBR TO STORE R1-R2 IN
     0315 3B 44                  BSTR,UN     GNUM
     0317 86 01                  ADDI,R2     1
     0319 77 08                  PPSL        WC
     031B 85 00                  ADDI,R1     0
     031D 75 08                  CPSL        WC          MAKE END ADDR NOT INCL
     031F CD 04 0F                STRA,R1     TEMQ
     0322 CE 04 10               STRA,R2     TEMQ+1
     0325 3B 38           FDUM   BSTR,UN     GAP
     0327 04 FF                  LODI,R0     -1
     0329 CC 04 29               STRA,R0     CNT
     032C 3F 00 8A               BSTA,UN     CRLF        PUNCH FOR CR/LF AND ST
     032F 04 3A                  LODI,R0     STAR
 PIP ASSEMBLER VERSION 3 LEVEL 1                                    PAGE    9

LINE ADDR B1 B2 B3 B4 ERR SOURCE

 414 0331 3F 02 B4              BSTA,UN     COUT
 415 0334 20                    EORZ        R0
 416 0335 CC 04 2C              STRA,R0     BCC
 417 0338 0D 04 0F              LODA,R1     TEMQ
 418 033B 0E 04 10              LODA,R2     TEMQ+1
 419 033E AE 04 0E              SUBA,R2     TEMP+1      GET BYTE COUNT
 420 0341 77 08                 PPSL        WC
 421 0343 AD 04 0D              SUBA,R1     TEMP
 422 0346 75 08                 CPSL        WC
 423 0348 1E 00 1D              BCTA,N      EBUG        START >  END ADDR
 424 034B 19 1C                 BCTR,P      ADUM        CNT >  NORMAL BLOCK SI
 425 034D 5A 1C                 BRNR,R2     BDUM        THIS IS SHORT BLOCK
 426 034F 07 04                 LODI,R3     4           EOF.   PUNCH ZERO BLK
 427 0351 3F 02 69       CDUM   BSTA,UN     BOUT
 428 0354 FB 7B                 BDRR,R3     CDUM
 429 0356 3B 07                 BSTR,UN     GAP
 430 0358 1F 00 22              BCTA,UN     MBUG
 431                     * SUBRS FOR OUTPUTTING BLANKS
 432 035B 07 03          FORM   LODI,R3     3
 433 035D 1B 02                 BCTR,UN     AGAP
 434 035F 07 32          GAP    LODI,R3     50
 435 0361 04 20          AGAP   LODI,R0     SPAC
 436 0363 3F 02 B4              BSTA,UN     COUT
 437 0366 FB 79                 BDRR,R3     AGAP
 438 0368 17                    RETC,UN
 439 0369 06 FF          ADUM   LODI,R2     255
 440 036B CE 04 28       BDUM   STRA,R2     MCNT
 441 036E 0D 04 0D              LODA,R1     TEMP        STARTING ADDRESS
 442 0371 3F 02 69              BSTA,UN     BOUT
 443 0374 0D 04 0E              LODA,R1     TEMP+1
 444 0377 3F 02 69              BSTA,UN     BOUT
 445 037A 0D 04 28              LODA,R1     MCNT        COUNT OF DATA BYTES IN
 446 037D 3F 02 69              BSTA,UN     BOUT
 447 0380 0D 04 2C              LODA,R1     BCC
 448 0383 3F 02 69              BSTA,UN     BOUT
 449 0386 0F 04 29       DDUM   LODA,R3     CNT
 450 0389 0F A4 0D              LODA,R0    *TEMP,R3,+
 451 038C EF 04 28              COMA,R3     MCNT
 452 038F 18 09                 BCTR,EQ     EDUM        OUTPUT BCC
 453 0391 CF 04 29              STRA,R3     CNT
 454 0394 C1                    STRZ        R1
 455 0395 3F 02 69              BSTA,UN     BOUT
 456 0398 1B 6C                 BCTR,UN     DDUM
 457 039A 0D 04 2C       EDUM    LODA,R1     BCC
 458 039D 3F 02 69              BSTA,UN     BOUT
 459 03A0 0E 04 0E              LODA,R2     TEMP+1
 460 03A3 8E 04 28              ADDA,R2     MCNT
 461 03A6 05 00                 LODI,R1     0
 462 03A8 77 08                 PPSL        WC
 463 03AA 8D 04 0D              ADDA,R1     TEMP
 464 03AD 75 08                 CPSL        WC
 465 03AF 3F 00 A4              BSTA,UN     STRT
 PIP ASSEMBLER VERSION 3 LEVEL 1                                     PAGE   10

LINE ADDR B1 B2 B3 B4 ERR SOURCE

 466 03B2 1F 03 25                 BCTA,UN      FDUM
 467                        * LOAD FROM PAPERTAPE IN OBJECT FORMAT
 468 03B5 3F 02 86          LOAD   BSTA,UN      CHIN         LOOK FOR START CHAR
 469 03B8 E4 3A                    COMI,R0      STAR
 470 03BA 98 79                    BCFR,EQ      LOAD
 471 03BC 20                       EORZ         R0
 472 03BD CC 04 2C                 STRA,R0      BCC
 473 03C0 3F 02 24                 BSTA,UN      BIN          READ ADDR AND COUNT IN
 474 03C3 CD 04 0D                 STRA,R1      TEMP
 475 03C6 3F 02 24                  BSTA,UN      BIN
 476 03C9 CD 04 0E                 STRA,R1      TEMP+1
 477 03CC 3F 02 24                 BSTA,UN      BIN
 478 03CF 59 03                    BRNR,R1      ALOA         CNT = 0 MEANS EOF
 479 03D1 1F 84 0D                 BCTA,UN     *TEMP
 480 03D4 CD 04 28          ALOA   STRA,R1      MCNT
 481 03D7 3F 02 24                 BSTA,UN      BIN          CHECK BCC ON INFORMATI
 482 03DA 0C 04 2C                 LODA,R0      BCC
 483 03DD 9C 00 1D                 BCFA,Z       EBUG
 484 03E0 C3                       STRZ         R3           READ DATA
 485 03E1 CF 04 29          BLOA   STRA,R3      CNT
 486 03E4 3F 02 24                 BSTA,UN      BIN
 487 03E7 0F 04 29                 LODA,R3      CNT
 488 03EA EF 04 28                 COMA,R3      MCNT
 489 03ED 18 06                    BCTR,EQ      CLOA         HAVE READ BCC
 490 03EF 01                       LODZ         R1
 491 03F0 CF E4 0D                 STRA,R0     *TEMP,R3      STORE DATA
 492 03F3 DB 6C                    BIRR,R3      BLOA
 493 03F5 0C 04 2C          CLOA   LODA,R0      BCC
 494 03F8 9C 00 1D                 BCFA,Z       EBUG
 495 03FB 1F 03 B5                 BCTA,UN      LOAD
 496                        *
 497                               ORG          H'400'
 498                        *******      RAM DEFINITIONS
 499 0400                   COM    RES          9
 500 0409 77 00             XGOT   PPSL         0
 501 040B 1B 80                    BCTR,UN     *$+2          MUST PREDEED THE TEMP
 502 040D                   TEMP   RES          2
 503 040F                   TEMQ   RES          2
 504 0411                   TEMR   RES          1
 505 0412                   TEMS   RES          1
 506 0413                   BUFF   RES          BLEN
 507 0427                   BPTR   RES          1
 508 0428                   MCNT   RES          1
 509 0429                   CNT    RES          1
 510 042A                   CODE   RES          1
 511 042B                   OKGO  RES   1
 512 042C                   BCC  RES   1
 513 042D                   MARK  RES  BMAX+1
 514 042F                   HDAT  RES  BMAX+1
 515 0431                   LDAT  RES  BMAX+1
 516 0433                   HADR  RES  BMAX+1
 517 0435                   LADR  RES  BMAX+1
```


Electronic components and materials
for professional, industrial and consumer uses

from a world-wide Group of Companies

EUROPEAN SALES OFFICES

Austria: Österreichische Philips, Bauelemente Industrie G.m.b.H., Zieglergasse 6, Tel. 93 26 11, A-1072 WIEN.
Belgium: M.B.L.E., 80, rue des Deux Gares, Tel. 523 00 00, B-1070 BRUXELLES.
Denmark: Miniwatt A/S, Emdrupvej 115A, Tel. (01) 69 16 22, DK-2400 KØBENHAVN NV.
Finland: Oy Philips Ab, Elcoma Division, Kaivokatu 8, Tel. 1 72 71, SF-00100 HELSINKI 10.
France: R.T.C., La Radiotechnique-Compelec, 130 Avenue Ledru Rollin, Tel. 355-44-99, F-75540 PARIS 11.
Germany: Valvo, UB Bauelemente der Philips G.m.b.H., Valvo Haus, Burchardstrasse 19, Tel. (040) 3296-1, D-2 HAMBURG 1.
Greece: Philips S.A. Hellénique, Elcoma Division, 52, Av. Syngrou, Tel. 915 311, ATHENS.
Ireland: Philips Electrical (Ireland) Ltd., Newstead, Clonskeagh, Tel. 69 33 55, DUBLIN 14.
Italy: Philips S.p.A., Sezione Elcoma, Piazza IV Novembre 3, Tel. 2-6994, I-20124 MILANO.
Netherlands: Philips Nederland B.V., Afd. Elonco, Boschdijk 525, Tel. (040) 79 33 33, NL-4510 EINDHOVEN.
Norway: Electronica A.S., Vitaminveien 11, Tel. (02) 15 05 90, P. O. Box 29, Grefsen, OSLO 4.
Portugal: Philips Portuguesa S.A.R.L., Av. Eng. Duharte Pacheco 6, Tel. 68 31 21, LISBOA 1.
Spain: COPRESA S.A., Balmes 22, Tel. 301 63 12 BARCELONA 7.
Sweden: ELCOMA A.B., Lidingövägen 50, Tel. 08/67 97 80, S-10 250 STOCKHOLM 27.
Switzerland: Philips A.G., Elcoma Dept., Edenstrasse 20, Tel. 01/44 22 11, CH-8027 ZÜRICH.
Turkey: Türk Philips Ticaret A.S., EMET Department, Gümüssuyu Cad. 78-80, Tel. 45.32.50, Beyoglü, ISTANBUL.
United Kingdom: Mullard Ltd., Mullard House, Torrington Place, Tel. 01-580 6633, LONDON WC1E 7HD.

Printed in The Netherlands 2-76 9399 509 52261

signetics

MICROPROCESSOR

SERIAL INPUT/OUTPUT................AS50

signetics

SERIAL INPUT/OUTPUT AS50

2650 MICROPROCESSOR

APPLICATION MEMO

INTRODUCTION

The Sense/Flag capability of the Signetics 2650 microprocessor can be used for serial I/O interfaces. The Sense input pin is directly connected to a bit in the microprocessor's Program Status Word. A high level on the Sense pin appears as a binary one while a low level appears as a binary zero. The Sense bit in the PSW can be stored or tested by the program. The Flag bit in the PSW is a simple latch that drives the Flag output pin. A program can set the Flag bit to a binary one, which causes a high level, one TTL load on the flag output pin. Setting the Flag bit to binary zero causes a low level on the Flag output pin.

APPLICATIONS

The most common use for the Sense/Flag capability would be in interfacing to a keyboard based terminal where the data is received or transmitted as bit serial. All bit manipulation and timings such as 8-bit serial-to-parallel conversion can be done by software running on the 2650. The software works by storing or setting the two bits in the Program Status Word which reflect or control the levels at the pins of the chip. External hardware is required simply to interface with line levels. No clock synchronization or address decoding hardware is necessary, since the Sense and Flag pins are independent of the normal I/O bus structure.

Two examples of device interfaces and software are given below; for a 1200 baud RS232-type CRT terminal and for a 110 baud Teletype. Figure 1 shows the RS232 interface. Half of the Signetics 8T15 dual line driver is used to transmit to the terminal from the Flag pin, while half of a

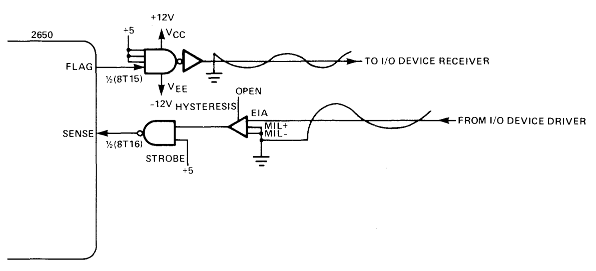


FIGURE I
RS-232 INTERFACE

Signetics 8T16 dual line receiver is used to receive from the device into the Sense pin. The interface to a Teletype model 33 is shown in Figure II. A TTL open collector gate is used to provide the 20 milli-amp loop to the TTY

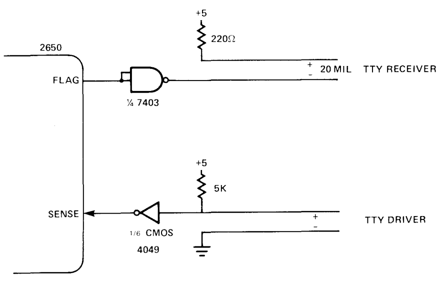


FIGURE II
TTY MODEL 33 INTERFACE FULL DUPLEX

receiver. For receiving from the TTY a CMOS gate is used to provide the necessary noise immunity.

SOFTWARE

All definitions of baud rate, character formats, and line characteristics are done in the software. For these examples, communication is asynchronous bit-serial over a full duplex line. Figure III shows the format of a 8-bit character (seven bits plus parity) headed by a start bit and followed by stop bits. The line levels are:

low = start bit or binary zero
high = stop bit or binary one

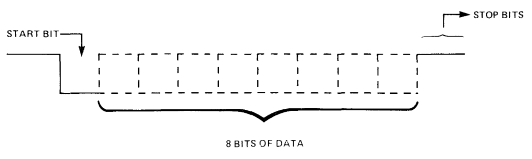


FIGURE III

```
     PIP ASSEMBLER VERSION 2 LEVEL 1                                                            PAGE     1

LINE  ADDR  LABL  B1 B2 B3 B4 ERROR SOURCE
                                           *
             0001                          P       EQU        1
             0002                          N       EQU        2
             0000                          Z       EQU        0
             0002                          LCOM     EQU        H'02'          LOGICAL COMPARE
             0001                          CAR      EQU        H'01'          CARRY
             0080                          SENS     EQU        H'80'          SENSE
             0040                          FLAG     EQU        H'40'          FLAG
             0020                          II       EQU        H'20'          INTERRUPT INHIBIT
             0020                          IDC      EQU        H'20'          INTER DIGIT CARRY
             0004                          OVF      EQU        H'04'          OVEFFLOW
             0000                          R0       EQU        0
             0001                          R1       EQU        1
             0002                          R2       EQU        2
             0003                          R3       EQU        3
             0003                          UN       EQU        3
             0000                          EQ       EQU        0
             0002                          LT       EQU        2
             0001                          GT       EQU        1
             0008                          WC      EQU        H'08'
             0010                          RS      EQU        H'10'
                                                   ORG        H'500'
      0500   0500  76 40                   CHIO    PPSU        FLAG       INPUT WITH A BIT BY BIT ECHO
      0502         75 08                           CPSL     WC
      0504         05 00                           LODI,R1     0
      0506         06 08                           LODI,R2     8
      0508   0508  12                      TEST    SPSU                   LOOP TESTS FOR THE START BIT
      0509         1A 7D                           BCTR,LT     TEST
      050B         3F 05 29                        BSTA,UN    DLY         HALF A DELAY TO MIDDLE OF BIT
      050E         74 40                           CPSU        FLAG
      0510   0510  3F 05 2D                BIT     BSTA,UN     DLAY       DELAY, THEN READ THE NEXT BIT
      0513         12                              SPSU
      0514         44 80                           ANDI,R0     H'80'
      0516         51                              RRR,R1
      0517         61                              IORZ        R1
      0518         C1                              STRZ        R1
      0519         1A 04                           BCTR,LT     ZERO       ECHO THE BIT
      051B         74 40                           CPSU        FLAG
      051D         1B 02                           BCTR,UN     NEXT
      051F   051F  76 40                   ZERO    PPSU        FLAG
      0521   0521  FA 6D                   NEXT    BDRR,R2     BIT
      0523         3F 05 2D                        BSTA,UN     DLAY
      0526         45 7F                           ANDI,R1     H'7F'
      0528         17                              RETC,UN
                                           *
                                           *  TIMING DELAY FOR 1200 BAUD RS232 TERMINAL
      0529   0529  04 3A                   DLY     LODI,R0     H'3A'
      052B         1B 02                           BCTR,UN     DL1
      052D   052D  04 59                   DLAY    LODI,R0     H'59'
      052F   052F  F8 7E                   DL1     BDRR,R0     $
      0531         17                              RETC,UN
                                           *  TIMING DELAY FOR 110 BAUD TELETYPE


     PIP ASSEMBLER VERSION 2 LEVEL 1                                                            PAGE     2

LINE  ADDR  LABL  B1 B2 B3 B4 ERROR SOURCE

      0532   0532  20                      TDLA    EORZ        R0
      0533         F8 7E                           BDRR,R0     $
      0535         F8 7E                           BDRR,R0     $
      0537   0537  F8 7E                   TDLY    BDRR,R0     $
      0539         04 E5                           LODI,R0     H'E5'
      053B         F8 7E                           BDRR,R0     $
      053D         17                              RETC,UN
                                                   END


TOTAL ASSEMBLER ERRORS =     0
```

FIGURE IV

The internal logic of the program shown in Figure IV (the program listing) is to sense each incoming bit of the character and to output the bit in turn for the full duplex line. The Sense input is tested in the loop at 'TEST' for the transition to zero indicating the start bit. The program then delays one half of a bit time to the center of the start bit. At this point the echoing of the character starts by clearing the Flag bit which outputs the start bit transition. At 'BIT' the program then delays one full bit time to the center of the data bit. The Sense line is tested and that bit value is rotated into register one. The bit value is then used to set or clear the Flag bit for the echo. At 'NEXT' is the test that controls the loop to get only eight bits. Figure V is a picture of the levels and timings when echoing a 'U'.

The bit timing is done by a subroutine which simply counts cycles for the appropriate baud rate. The example program shows both a 1200 baud delay at 'DLAY' and a 110 baud delay at 'TLAY'. The conversion from instruction cycles to milliseconds is based on a 1MHz clock rate. Clock stability is only moderately important since each character involves only nine sample times and each start bit redefines the base line for all timings.

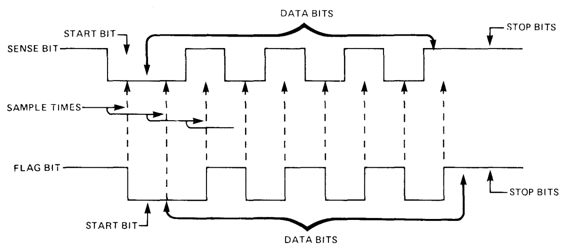
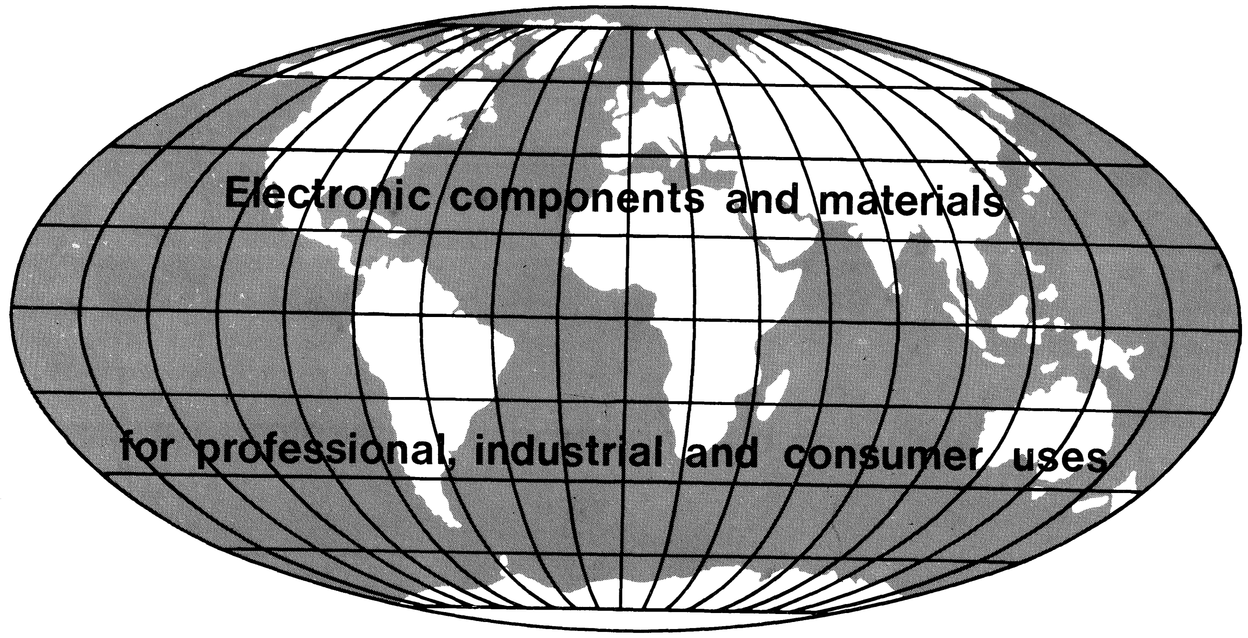


FIGURE V


Electronic components and materials
for professional, industrial and consumer uses

from a world-wide Group of Companies

EUROPEAN SALES OFFICES

Austria: Österreichische Philips, Bauelemente Industrie G.m.b.H., Zieglergasse 6, Tel. 93 26 11, A-1072 WIEN.
Belgium: M.B.L.E., 80, rue des Deux Gares, Tel. 523 00 00, B-1070 BRUXELLES.
Denmark: Miniwatt A/S, Emdrupvej 115A, Tel. (01) 69 16 22, DK-2400 KØBENHAVN NV.
Finland: Oy Philips Ab, Elcoma Division, Kaivokatu 8, Tel. 1 72 71, SF-00100 HELSINKI 10.
France: R.T.C., La Radiotechnique-Compelec, 130 Avenue Ledru Rollin, Tel. 355-44-99, F-75540 PARIS 11.
Germany: Valvo, UB Bauelemente der Philips G.m.b.H., Valvo Haus, Burchardstrasse 19, Tel. (040) 3296-1, D-2 HAMBURG 1.
Greece: Philips S.A. Hellénique, Elcoma Division, 52, Av. Syngrou, Tel. 915 311, ATHENS.
Ireland: Philips Electrical (Ireland) Ltd., Newstead, Clonskeagh, Tel. 69 33 55, DUBLIN 14.
Italy: Philips S.p.A., Sezione Elcoma, Piazza IV Novembre 3, Tel. 2-6994, I-20124 MILANO.
Netherlands: Philips Nederland B.V., Afd. Elonco, Boschdijk 525, Tel. (040) 79 33 33, NL-4510 EINDHOVEN.
Norway: Electronica A.S., Vitaminveien 11, Tel. (02) 15 05 90, P. O. Box 29, Grefsen, OSLO 4.
Portugal: Philips Portuguesa S.A.R.L., Av. Eng. Duharte Pacheco 6, Tel. 68 31 21, LISBOA 1.
Spain: COPRESA S.A., Balmes 22, Tel. 301 63 12 BARCELONA 7.
Sweden: ELCOMA A.B., Lidingövägen 50, Tel. 08/67 97 80, S-10 250 STOCKHOLM 27.
Switzerland: Philips A.G., Elcoma Dept., Edenstrasse 20, Tel. 01/44 22 11, CH-8027 ZÜRICH.
Turkey: Türk Philips Ticaret A.S., EMET Department, Gümüssuyu Cad. 78-80, Tel. 45.32.50, Beyoglü, ISTANBUL.
United Kingdom: Mullard Ltd., Mullard House, Torrington Place, Tel. 01-580 6633, LONDON WC1E 7HD.

Printed in The Netherlands 2-76 9399 509 52461

PHILIPS

Electronic components and materials

Technical note 072

Introducing the Signetics 2651 PCI

Terminology, formats and operation modes

Data exchange between microprocessors and peripherals is normally performed in the parallel mode, using the data bus. This means that the data is transferred along a number of parallel connections, all bits of the data word being transferred simultaneously. However, when a peripheral is remote from the microcomputer system, it is usually more economical and sometimes obligatory to communicate via a single data line. This means that the parallel data within the microcomputer must be converted into a serial form before transmission and vice versa.

The conversion between serial and parallel data can be done either by hardware or software, but the hardware solution must usually be used as the software conversion puts too great a load on the microprocessor. The Signetics 2651 Programmable Communications Interface (PCI) has been developed to perform this task of parallel/serial conversion: it is a single chip providing the complete hardware for virtually any mode of serial data communication. Figure 1 shows two typical applications of serial data interfaces to microcomputers. Note that for communication over a telephone line, a modulator/demodulator (modem) is required.

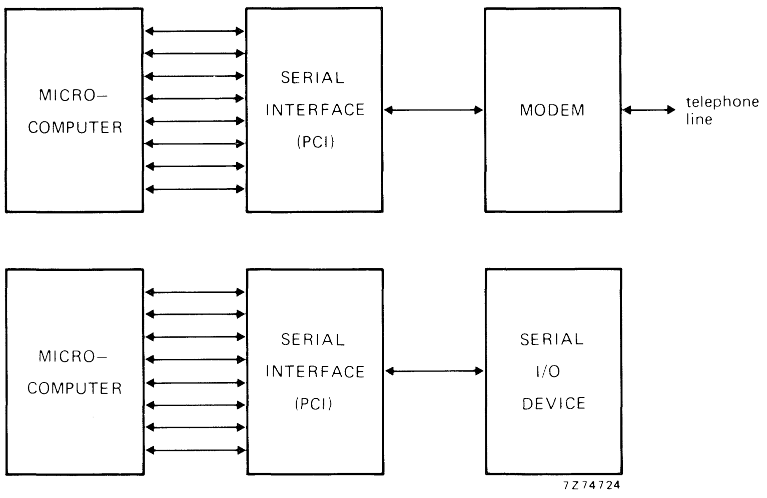


Fig. 1 Typical applications of serial data interfaces to microcomputers.

Serial data communication formats

Figure 2 shows the serial bit stream equivalent to three eight-bit data words. Whereas parallel data bits can be recognized at a receiver by their separate connections, serial data bits can only by distinguished by their separation in time. The receiver must therefore be supplied with *timing information. Framing information* is necessary to be able to recover the original data words from the serial bit stream

Timing information enables the receiver to distinguish between consecutive bits of the serial data stream, while framing information enables the receiver to recognize the start and finish of each data word.

Figure 3 shows the use of timing information to recover the original data. Each bit of the serial data stream must be transmitted for a fixed duration, called the *unit interval.*

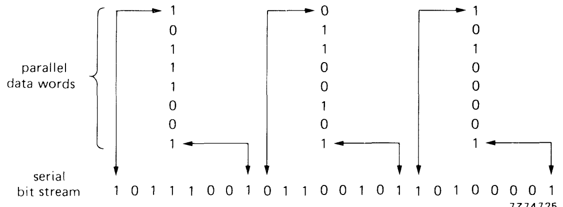


Fig. 2 Relation between the parallel data words and the serial bit stream transmitted or received.

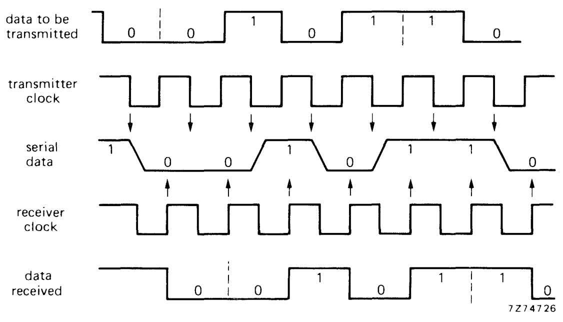


Fig. 3 Use of a receiver clock to recover data from the serial bit stream.

The receiver clock must be synchronized to the frequency of the transmitter clock, with a fixed phase delay, to allow sampling of the serial data waveform at the same time in each unit interval. The maximum rate at which information can be sent over the data line is known as the *baud rate*; it is equal to the number of unit intervals per second. Thus for a unit interval of 20 ms, the baud rate is 50 *baud.* Commonly used baud rates range from 50 baud up to 19,2 kbaud. A standard teletype uses 110 baud.

To recover the original data, the timing information must contain:
- the baud rate;
- bit synchronization information.

To reform the original data words, the framing information must contain:

- identification of the first bit of a data word;
- the number of bits per word;
- the sequence in which the bits are sent (msb or lsb first).

Of these, the baud rate, the number of bits per word and the sequence in which the bits are sent, are usually fixed and already known by the receiver before the transmission of data. Thus the data signal must contain the bit synchronization and first-bit identification information. Several serial data communication formats have been devised for this purpose. The two basic formats are *synchronous* and *asynchronous,* while a mixture of the two is called *isochronous.*

Asynchronous format

When using the asynchronous format, the transmitter transmits each word separately. Each word is preceded by one start bit and followed by a parity bit and 1, 1½ or 2 stop bits. This format is illustrated in Fig. 4.

When the data line is quiet, the signal is a one. The start bit is a zero, which tells the receiver that a data word is coming. Since the start bit can occur at any moment, synchronization of the receiver clock to the data signal must be repeated for each data word. Therefore, the receiver clock runs at a multiple (usually 16x or 64x) of the actual baud rate. Figure 5 shows the synchronization principle with a clock running at 16 times the baud rate. The receiver clock is derived from this 16x clock by means of a divide-by-16 circuit, which starts at the moment a start bit is detected (falling edge of the previously quiet line).

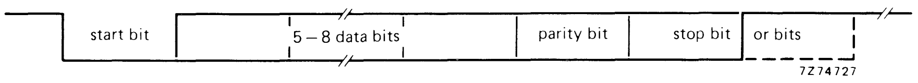


Fig. 4 The asynchronous serial data format.

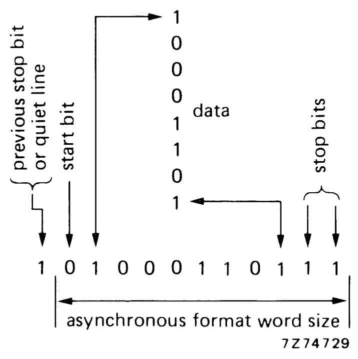


Fig. 6 Example of an asynchronous data format using eight data bits and two stop bits.

After eight pulses of the 16x clock, the data line is sampled again and if the line is still a zero, the start bit is accepted. After eight more pulses, the first data bit appears on the line and is sensed 16 pulses after the start bit was accepted. This means that the bit is sensed at the middle of the bit-time to avoid switching transients. Each following group of 16 clock pulses will represent a unit interval, after which a new bit is sensed.

The maximum inaccuracy resulting from this method of synchronization is initially one sixteenth (in the case of a 64x clock, 1/64th) of a bit-time. The accuracy of synchronization of the following data bits depends on the equality of the transmitter and receiver clock frequencies. However, since the next data word will provide a new start bit, this synchronization only has to last for the duration of one data word.

From the foregoing description, it is clear that all the bits transmitted, including the start bit, must be exactly one unit interval long, and that the time between two consecutive data words need not be an integral number of unit intervals. To ensure correct detection of the next start bit (data word) the line must be forced to the quiet state at the end of the data bits of every word. This is done by adding the stop bit or bits onto the end of the word, which, being logical one, provide the quiet state. Thus the receiver will always see a clear falling edge at the beginning of the start bit of the next data word, even if the receiver clock runs slightly slower than the transmitter clock. Figure 6 shows an example of an asynchronous data format with eight data bits and two stop bits.

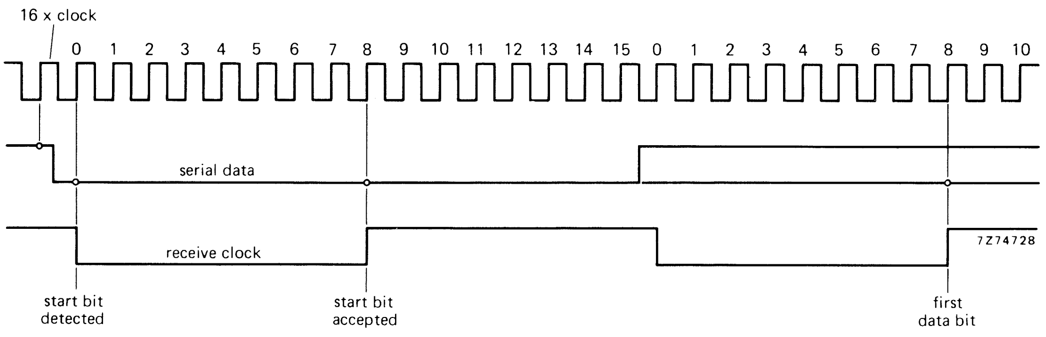


Fig. 5 Synchronization principle for the asynchronous format, here using 16x clock.

Synchronous format

In the synchronous format, data words are grouped into blocks before transmission. This provides a continuous stream of valid data bits, one per unit interval. Once the transmitter and receiver clocks are synchronized, the receiver looks for framing information. This is done in the *hunt mode* in which it continually checks the received bit sequence for synchronization characters.

Each block is preceded by one or more synchronization words, of a fixed character, see Fig. 7. The sync words are called SYN (in the case of one sync word) or SYN1 and SYN2 in the case of two sync words. The bit patterns corresponding to these characters must not occur in the data to be transmitted.

When a bit sequence conforming to the sync character(s) is recognized, the receiver switches to the data mode: the first bit following the sync character(s) is the first bit of the first data word. Figure 8 shows the bit sequence of a data block in synchronous format with two sync characters and five bits per word.

If, during transmission of a data block, the microprocessor fails to supply a new data word for transmission, the transmitter automatically inserts sync characters (SYN or SYN1-SYN2 pairs as appropriate) to prevent a gap occurring. Sync characters are inserted until a new data word is available. These sync characters can be automatically discarded by the receiver, while providing both framing and timing synchronization.

Since synchronization of the receiver and transmitter clocks must be maintained over a long stream of data, the timing information is usually continuously extracted from the data signal. Sometimes, however, a synchronization signal is sent to the receiver separately.

Isochronous format

In order to recover the data words from the serial data stream, the isochronous method employs the framing of the asynchronous format (start and stop bits) and the timing of the synchronous method (clock frequency equal to the baud rate). The asynchronous framing permits gaps in the data, although these must now be an integral number of unit intervals so that the timing can remain synchronized.

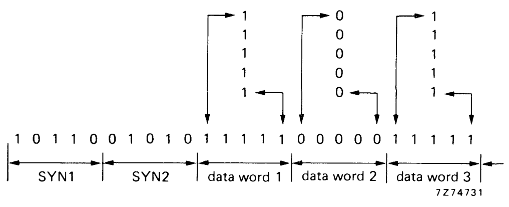


Fig. 8 Synchronization at the start of a data block in the synchronous format. Double sync operation with five bits per data word.

Modem control

Telephone lines are designed for speech transmission and will only transmit analogue signals in the range 300 Hz to 3400 Hz. Digital signals cannot be transmitted. Thus when transmitting digital data over telephone lines, a modem (modulator-demodulator) is required to provide the conversion to and from analogue signals. In order to regenerate the modulated data, a synchronous modem must be able to generate a receiver clock for the conversion of the serial data into parallel data. Figure 9 shows the use of a modem to interface digital signals to a telephone line. As can be seen from the diagram, several control lines are required between the serial interface and the modem. These are:

- TxC: *Transmitter Clock,* timing information for synchronous modem;
- TxD: *Transmit Data* in serial form to the modem;
- RTS: *Request to Send,* request to the modem to prepare for the transmission;
- DTR: *Data Terminal Ready,* request to the modem to establish the connection to the telephone line and enter the data mode (as opposed to the dial and talk modes).
- RxC: *Receiver Clock,* for synchronous modem;
- RxD: *Receive Data,* from the modem to the serial interface;
- DSR: *Data Set Ready,* signals that the modem is connected to the telephone line and in the data mode;
- CTS: *Clear to Send,* signals that the modem is ready to accept serial data for transmission;
- DCD: *Data Carrier Detect,* signals that the modem is receiving a signal suitable for demodulation.

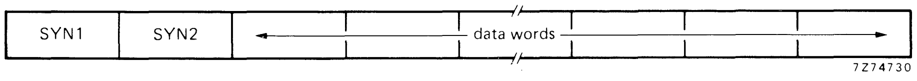


Fig. 7 The synchronous serial data format.

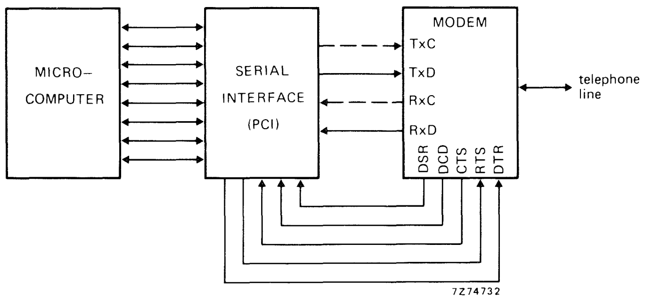


Fig. 9 Transmitting digital data over a telephone line by using a modem, TxC and RxC are required if a synchronous modem is used.

Transparent operation – synchronous mode

Data communication systems commonly employ a seven-bit, 128-character code, known as seven-bit ISO code, see Appendix A. This includes alphanumerics, general purpose control characters and ten transmission control characters (TC_n). *Communication control procedures* have been established to provide rules for the use of these transmission control characters.

In some cases, when it is required to obtain the maximum throughput of the communication network, data may be transmitted without using ISO code, in a network that normally uses ISO code. In this case, data words with the same bit pattern as the transmission control characters will be misinterpreted by the system as transmission control characters.

To overcome this problem, the communication control procedures have been extended to specify *transparent operation,* to allow *code-independent* data transfer. This operation is governed by the transmission control character DLE (Data Link Escape).

The start of code-independent data is preceded by two transmission control characters DLE and STX (Start of Text). The end of the sequence is indicated by the characters DLE and ETB (End of Transmission Block) or DLE and ETX (End of Text). Between these start and finish character sequences, all bit patterns except DLE are treated as data. Control characters can still be used for control purposes if preceded by the DLE character. Sync characters used to fill gaps in the synchronous format should also be preceded by DLE to prevent interpretation of these characters as data. The exception is the case where the data word has the same bit pattern as DLE plus parity bit. This is transmitted as DLE DLE and the second DLE is interpreted as data, not control. Table 1 shows the interpretation of various character sequences.

Table 1 Transparent operation using the DLE character.

sequence transmitted and received	interpretation	
	data	control
DLE ETB		ETB
DLE DLE	DLE	
X ETB	X ETB	
X ETX	X ETX	
X SYN	X SYN	
DLE SYN		gap filler
DLE STX		STX
DLE DLE DLE ETB	DLE	ETB
DLE DLE DLE DLE	DLE DLE	
DLE DLE ETB	DLE ETB	

Note: X is any character other than DLE.

Error detection

Parity check

Regardless of the transmission medium, the data signal entering the receiver will contain noise. At times, this noise content may be sufficient to cause incorrect detection of data in the receiver.

The parity check provides a simple method of detecting a single error in a data word; a parity bit is added to each data word so that the total number of ones in each word is always even, or always odd. This is called *even parity* or *odd parity.* The receiver can then detect an odd number of incorrect bits in the data word. An even number of errors will not be detected.

Framing error

In the asynchronous mode, the receiver may detect the wrong number of stop bits, called a framing error.

Overrun error

The microprocessor should read any received data words before the following data word is received. If this is not done, the contents of the receiver register are overwritten and the earlier data word lost. This is called an overrun error.

The Signetics 2651 PCI

The Signetics 2651 Programmable Communications Interface is a universal synchronous/asynchronous data communications controller chip. Although designed specifically for use with the Signetics 2650 microprocessor, the PCI can easily be used with other CPUs in either polled or interrupt-driven environments.

The 2651 accepts programmed instructions from the microprocessor and supports many serial data communication disciplines in half or full duplex. The PCI supports isochronous, asynchronous and synchronous operations, including transparent synchronous operation.

The PCI converts the parallel data received from the microprocessor into a serial data stream, ready for transmission. At the same time, because of the independent transmit and receive circuitry, it can receive a serial data stream and convert this into parallel data for the microprocessor.

The block diagram of the PCI is shown in Fig. 10. All data and control transfers between the PCI and the microprocessor are accomplished via the data bus buffer, connecting the internal data bus to that of the microprocessor.

Overall control of the PCI is by the Chip Enable ($\overline{CE}$) signal. Address lines A0 and A1 allow selection of the required registers, while the Read/Write ($\overline{R}$/W) signal controls the direction of data flow between the PCI and the microprocessor.

The operation of the PCI is determined by the contents of the two mode registers and the command register. The registers are loaded via the data bus during system initialization.

The data in mode register 1 controls:

- the format:
  - synchronous
  - isochronous (asynchronous with 1x clock)
  - asynchronous (with 16x clock)
  - asynchronous (with 64x clock);
- the data word length:
  - 5, 6, 7 or 8 bits per word;
- the error checking:
  - no checking
  - odd parity
  - even parity;
- the number of stop bits for isochronous or asynchronous operation:
  - 1, 1½ or 2 bits;
- the use of SYN and DLE characters:
  - single sync operation (SYN1)
  - double sync operation (SYN1 + SYN2)
  - transparent operation (DLE + SYN).

The data in mode register 2 controls clock selection and the baud rate when internal clocks are selected:

- receiver clock:
  - internal or external;
- transmitter clock:
  - internal or external;
- baud rate:
  - 16 commonly used baud rates in the range 50 baud to 19 200 baud.

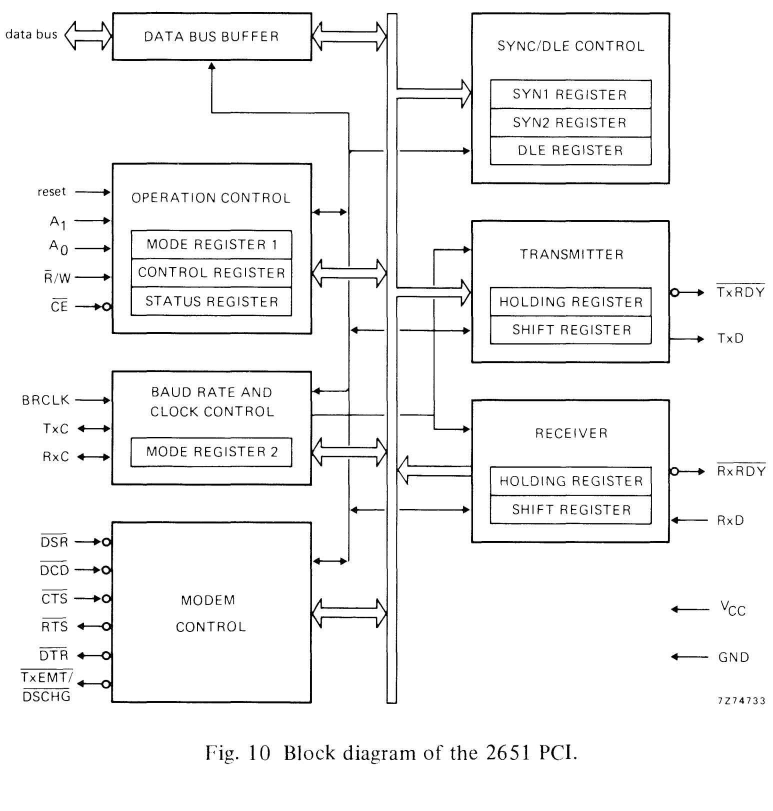


Fig. 10 Block diagram of the 2651 PCI.

The internal baud rate is derived from an externally generated 5,0688 MHz clock. When both receiver and transmitter clocks are programmed as external, the 5,0688 MHz clock is not required. If an internal clock is programmed, the programmed baud rate clock appears at the corresponding pin (TxC: transmitter clock; RxC: receiver clock). If an external clock is programmed, these pins become inputs for the external clock(s). With an external clock (max. 0,8 MHz) the baud rate is no longer governed by mode register 2. Thus the baud rates can be:

- synchronous and isochronous operation:
  0 to 800 kbaud;
- asynchronous operation with 16x clock:
  0 to 50 kbaud;
- asynchronous operation with 64x clock:
  0 to 12,5 kbaud.

The data in the command register controls:

- the transmitter:
  enable or disable;
- the receiver:
  enable or disable;
- the modem control signals DTR and RTS;
- resetting error flags in the status register;
- forced breaks in asynchronous format:
  The quiet line signal is a one. Break is a continuous zero sent by the transmitter which can be detected by the receiver.
- sending DLE in the synchronous format;
- the operation mode:
  - normal operation (Fig. 11a)
  - automatic echo mode, asynchronous (Fig. 11b). In this mode, the receiver controls the transmitter, causing it to transmit an echo of the received data to the external device.
  - SYN/DLE stripping mode, synchronous.
    SYN and DLE characters received by the PCI are not passed to the microprocessor.
  - local loop-back mode (Fig. 11c). The transmitter output is connected to the receiver input internally to provide a test facility or the CPU-PCI system.
  - remote loop-back mode (Fig. 11d). The receiver output is internally connected to the transmitter input to divert the received data to another computer.

The status register provides the microprocessor with information about the transmitter, the receiver, the modem control signals DSR and DCD, parity, overrun and framing errors and the detection of SYN/DLE characters.

The transmitter and receiver are double buffered, allowing the microprocessor to read/write one data word while another is being received/transmitted. This allows the microprocessor one complete serial word transmit/receive time to read/write the data word

When using synchronous formats, the SYN and DLE characters are supplied by the microprocessor during the initialization phase of the program and stored in PCI registers.

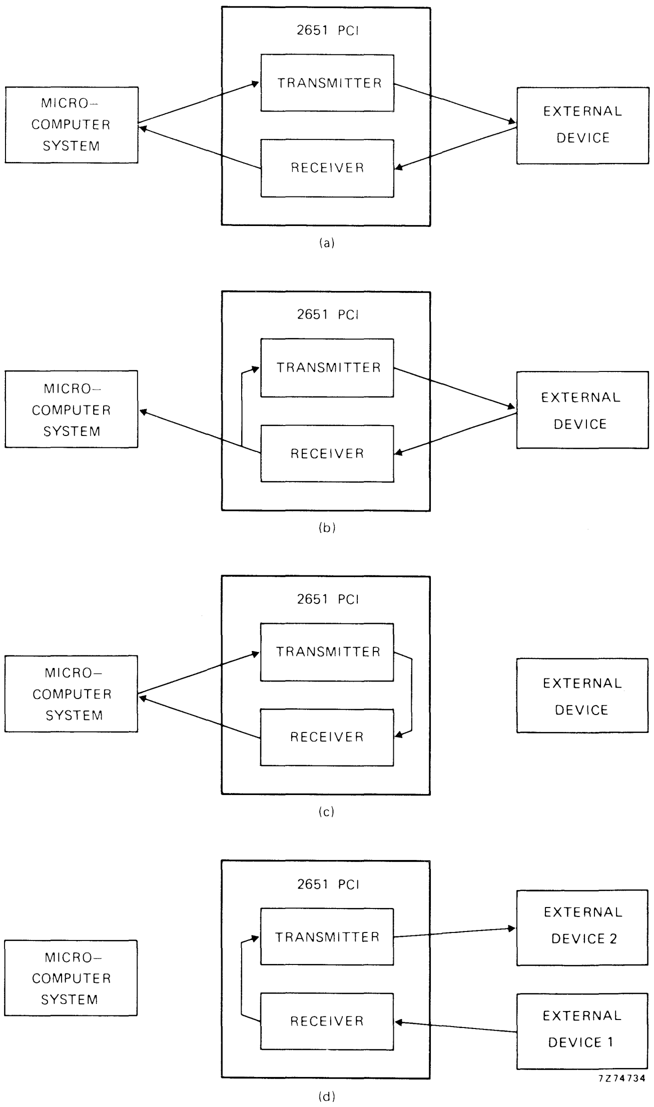
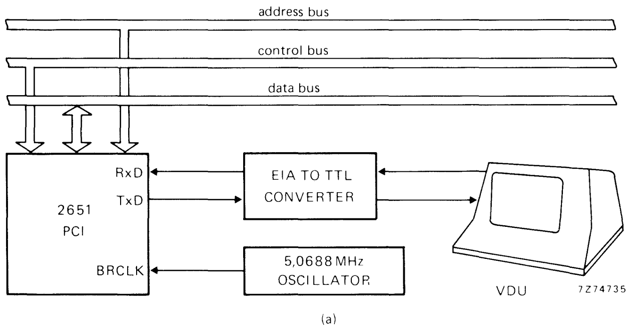


Fig. 11 Operation modes of the 2651 PCI.
(a) Normal operation mode.
(b) Automatic echo mode.
(c) Local loop-back mode.
(d) Remote loop-back mode.

Applications of the 2651 PCI

The 2651 PCI is suitable for almost any application where parallel source data must be transmitted over a single data line. Figure 12 shows two typical examples.



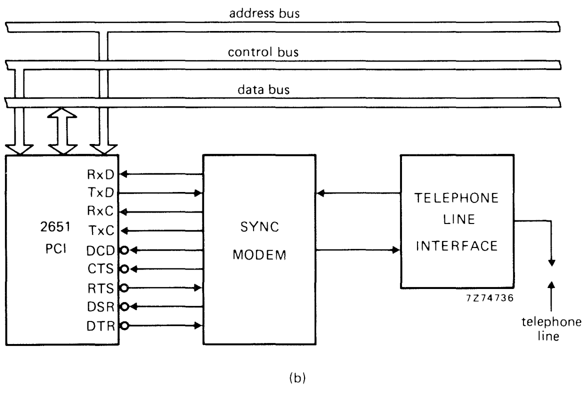


Fig. 12 Typical application for the 2651 PCI.
(a) Asynchronous interface to visual display unit.
(b) Synchronous interface to telephone line.

Appendix A

The international standard (ISO) 7-bit code as defined in Appendix B ref. 4:

7	6	5	4	3	2	1	7-bit character format.

$\underbrace{7\;6\;5}_{\text{column (0-7) (MS character)}}\;\underbrace{4\;3\;2\;1}_{\text{row (1-15) (LS character)}}$

column
(0-7)
(MS character)

row
(1-15)
(LS character)

CHARACTER SET								
M.S. CHAR / L.S. CHAR	0	1	2	3	4	5	6	7
0	NUL	(TC$_7$) DLE	SP	0	@	P	`	p
1	(TC$_1$) SOH	DC$_1$	!	1	A	Q	a	q
2	(TC$_2$) STX	DC2	”	2	B	R	b	r
3	(TC$_3$) ETX	DC$_3$	#	3	C	S	c	s
4	(TC$_4$) EOT	DC$_4$	$	4	D	T	d	t
5	(TC$_5$) ENQ	(TC$_8$) NAK	%	5	E	U	e	u
6	(TC$_6$) ACK	(TC$_9$) SYN	&	6	F	V	f	v
7	BEL	(TC$_{10}$) ETB	'	7	G	W	g	w
8	FE$_0$ (BS)	CAN	(	8	H	X	h	x
9	FE$_1$ (HT)	EM	)	9	I	Y	i	y
10	FE$_2$ (LF)	SUB	*	:	J	Z	j	z
11	FE$_3$ (VT)	ESC	+	;	K	[	k	
12	FE$_4$ (FF)	IS$_4$ (FS)	,	<	L	\	l	
13	FE$_5$ (CR)	IS$_3$ (GS)	–	=	M	]	m	
14	SO	IS$_2$ (RS)	•	>	N	↑	n	‾
15	SI	IS$_1$ (US)	/	?	O	_	o	DEL

Appendix B

International standards governing serial data communication can be found in:

1. CCITT V22 and V22 bis; data signalling rates (standard baud rates).
2. CCITT V21, V23, V26, V26 bis, V27, V30 and V35; modem standards.
3. EIA RS232-C, CCITT V24; interfaces employing serial binary data interchange.
4. ISO 646; 7-bit coded character set.
5. ISO 1177; character structure for start/stop and synchronous transmission.
6. ISO 1745; basic mode-control procedures for data communication systems.*
7. ISO 2111; code-independent transmission procedures.

CCITT:	Comité Consultatif International dé Telegraphie et du Téléphonie.
EIA:	Electronic Industries Association (USA).
ISO:	International Standardization Organization.

* Commonly called Communications Control Procedures.

Related 2650 publications

no.	title	summary
AS50	Serial Input/Output	Using the Sense/Flag capability of the 2650 for serial I/O interfaces.
AS51	Bit & Byte Testing Procedures	Several methods of testing the contents of the internal registers in the 2650.
AS52	General Delay Routines	Several time delay routines for the 2650, including formulas for calculating the delay time.
AS53	Binary Arithmetic Routines	Examples for processing binary arithmetic addition, subtraction, multiplication, and division with the 2650.
AS54	Conversion Routines	● Eight-bit unsigned binary to BCD ● Sixteen-bit signed binary to BCD ● Signed BCD to binary ● Signed BCD to ASCII ● ASCII to BCD ● Hexadecimal to ASCII ● ASCII to Hexadecimal
AS55	Fixed Point Decimal Arithmetic Routines	Methods of performing addition, subtraction, multiplication and division of BCD numbers with the 2650.
SP50	2650 Evaluation Printed Circuit Board (PC1001)	Detailed description of the PC1001, an evaluation and design tool for the 2650.
SP51	2650 Demo System	Detailed description of the Demo System, a hardware base for use with the 2650 CPU prototyping board (PC1001 or PC1500).
SP52	Support Software for use with the NCSS Timesharing System	Step-by-step procedures for generating, editing, assembling, punching, and simulating Signetics 2650 programs using the NCSS timesharing service.
SP53	Simulator, Version 1.2	Features and characteristics of version 1.2 of the 2650 simulator.
SP54	Support Software for use with the General Electric Mark III Timesharing System	Step-by-step procedures for generating, editing, assembling, simulating, and punching Signetics 2650 programs using General Electric's Mark III timesharing system.
SP55	The ABC 1500 Adaptable Board Computer	Describes the components and applications of the ABC 1500 system development card.
SS50	PIPBUG	Detailed description of PIPBUG, a monitor program designed for use with the 2650.
SS51	Absolute Object Format	Describes the absolute object code format for the 2650.
MP51	Initialization	Procedures for initializing the 2650 microprocessor, memory, and I/O devices to their required initial states.
MP52	Low-Cost Clock Generator Circuits	Several clock generator circuits, based on 7400 series TTL, that may be used with the 2650. They include RC, LC and crystal oscillator types.
MP53	Address and Data Bus Interfacing Techniques	Examples of interfacing the 2650 address and data busses with ROMs and RAMs, such as the 2608, 2606 and 2602.
MP54	2650 Input/Output Structures and Interfaces	Examines the use of the 2650's versatile set of I/O instructions and the interface between the 2650 and I/O ports. A number of application examples for both serial and parallel I/O are given.
TN 064	Digital cassette interface for a 2650 microprocessor system	Interface hardware and software for the Philips DCR digital cassette drive.
TN 069	2650 Microprocessor keyboard interfaces	Simple interfaces for low-cost keyboard systems.

Electronic components and materials

for professional, industrial and consumer uses

from the world-wide Philips Group of Companies

Printed in The Netherlands   Date of release: 9 February 1978   T.D.   9399 500 07201

signetics

MICROPROCESSOR

2650 INITIALIZATION..............MP51

signetics

2650 INITIALIZATION MP51

APPLICATIONS MEMO

At power-up the status of the 2650 is undefined. The Reset signal should be raised for at least three clock periods. This forces execution of the instruction at location 0. Once the system is started up, the first program to run is generally responsible for initializing the microprocessor, memory, and I/O devices to their desired initial states. The type of I/O initialization is dependent on the particular device. Contents of RAM are undefined at power-up and must be set to their desired initial states.

Program status word initialization:

1. Interrupts can be inhibited as a first step in initialization. The Reset clears the Interrupt Inhibit bit and the internal Interrupt Waiting signal. After the remainder of the status bits, the memory, and the I/O is initialized, interrupts can be permitted. This procedure will prevent unwanted interrupts during system initialization. If the system does not utilize interrupts, the Interrupt Inhibit bit can be left set on when system initialization is complete. This approach will assure that a spurrious interrupt will not occur.
2. The Stack Pointer may be initialized to zero. The Stack Pointer should not be modified during the execution of a program. This pointer is under the control of the processor. Modification by a program could have unwanted results, i.e., to the instruction address register.
3. It is generally unnecessary to initialize the Condition Code, Interdigit Carry, Overflow, and Carry bits. These bits are normally set by arithmetic and logical operations before they are tested. However, if the With Carry bit is set on, then the Carry bit should be initialized correctly for the first arithmetic instruction.
4. The Register Select bit should be set to a known state, e.g. if bank 1 registers are reserved for interrupt routines, the register select bit should be initialized to bank 0.
5. The With Carry bit can be initialized to the state desired for most arithmetic and rotate operations. Then if a different state is desired for some operations, the With Carry bit can be changed and then restored after these operations.
6. The same philosophy used for the With Carry bit also applies to the Compare bit. Set the Compare bit initially to the most frequent types of compares made, logical or arithmetic.
7. The Sense bit cannot be modified by a program. The Flag bit may need to be initialized if there is a device connected to it such as a TTY which needs stop bits (binary one) when not receiving data.

signetics

MICROPROCESSOR

SIMULATOR, VERSION 1.2......SP53

SIMULATOR, VERSION 1.2 SP53

APPLICATIONS MEMO

A new version of the Simulator is available. This version performs the same functions as Version 1.0 (see Simulator Manual) with the following additional features:

1. Hexadecimal Object Module
   The Simulator accepts an object module produced by the Assembler in either decimal or hexadecimal format. The Simulator assumes that the object module is hexadecimal, unless the user specifies a decimal module by adding a fourth parameter, FORMAT, to the "EXECUTE SIMULATOR" command. This command is formatted differently depending upon the computer system on which the Simulator is installed.
2. 8K (8192 bytes) Object Module
   The Simulator reads and executes an object module with up to 8192 bytes.
3. Decimal Input to LIMIT Command
   The LIMIT command expects the number of instructions to be entered in decimal, not hexadecimal. Thus, a "LIMIT 40" command causes the program to execute 40_{10} not 64_{10} instruction. All other commands still expect their input parameters to be in hexadecimal.
4. Stack Wraparound Notification
   Whenever a RETC or a RETE is executed with the stack pointer equal to 0 or whenever a branch to subroutine instruction is executed with the stack pointer equal to 7, the Simulator prints the following message:

   STACK WRAPAROUND, IAR=XXXX

   Where XXXX identifies the address at which the wraparound occurred.
5. Termination Messages
   The Simulator prints a message for every kind of program termination:

TYPE OF TERMINATION	SIMULATOR RESPONSE
1. STOP. command	A trace of the last instruction executed is printed.
2. HALT instruction	A trace of the last instruction executed is printed.
3. LIMIT command	"LIMIT REACHED=XXXX, IAR=XXXX" is printed. A trace of the last instruction executed is printed.
4. Attempt to access area outside of memory	"ADDRESS OUT OF RANGE, IAR=XXXX" is printed. A trace of the last instruction executed is printed.
5. Attempt to execute instruction outside of memory	"IAR EXCEEDS MEMORY, IAR=XXXX" is printed.

6. Simulator Version Notification
   The simulator prints the following message whenever it starts to execute a program:

   2650 SM 1000 "PIPSIM" VERSION X.X

   X.X identifies the version of the simulator currently executing.

signetics

MICROPROCESSOR

ABSOLUTE OBJECT FORMAT.....SS51

signetics

ABSOLUTE OBJECT FORMAT SS51

2650 MICROPROCESSOR

APPLICATIONS MEMO

INTRODUCTION

The format for absolute code produced for the 2650 is described in this application note.

The absolute object code is formatted into blocks. The first character of every block is a colon. Inside of a block, all the characters are hexadecimal, i.e., 0 to 9 or A to F inclusive. In the gap between the blocks all characters are ignored. A CR/LF is used within the interblock gap to reset the TTY or terminal after each block.

Each block is independent. For example, papertape can be positioned prior to any block and a load started. The loading of absolute object code will be halted by:

A BCC error on the address + count fields
A BCC error on the data field
An incorrect block length
A non-hex character within the block

The block length field contains the number of bytes of actual data which is half the number of hex characters in the data field. The size of the data field can range from 2 to 510 characters. A block length of zero indicates this is an EOF block. The address field of an EOF block contains the start address of the loaded program.

The Block Control Character is 8 bits formed from the actual bytes and not from the ASCII characters. The bytes are in turn exclusive or'ed to the BCC byte, and then the BCC byte is left rotated one bit. It appears as two hex characters. Both the address and count fields and the data field are followed by a BCC character pair. The BCC prevents storing into memory at an invalid address or storing bad data.

EXAMPLE An object tape that loads ten bytes starting at location 500

```
:05000A3C0455B024FFF01F05040030
:000000
```

FORMAT

1. Interblock gap of any characters including spaces
2. Start of block character;
   a colon
3. Address field;
   four hex characters
4. Count field;
   two hex characters in range 0 to FF
5. BCC for address and count fields;
   two hex characters
6. Data field;
   twice the value in the count field which is the number of memory locations loaded by the current block
7. BCC for the data field;
   two hex characters

signetics

MOS MICROPROCESSOR

BIT AND BYTE TESTING PROCEDURES AS51

BIT AND BYTE TESTING PROCEDURES | AS51

2650 MICROPROCESSOR

APPLICATIONS MEMO

SUMMARY

This applications memo describes several methods of testing the contents of the internal registers in the Signetics 2650 Microprocessor.

The following test examples are given:

- Specific bit(s) in a register.
- Positive, negative, or zero-contents of a register.
- Contents of a register compared with a value (equals, greater than, or less than).
- Interdigit-carry (IDC), overflow (OVF), and carry (C) flags in the program status word.

INTRODUCTION

As a result of an operation on register(s) of the 2650 register bank, five bits (bits 7, 6, 5, 2, and 0) in the Program Status Lower (PSL) portion of the Program Status Word (PSW) register can be affected.

7	6	5	4	3	2	1	0
CC1	CC0	IDC	RS	WC	OVF	COM	C

PROGRAM STATUS LOWER (PSL)

These bits are affected as follows:

CC1, CC0: Condition Code Bits

CONDITION CODE		RESULT OF		
CC1	CC0	LOAD/STORE, ARITHMETIC, LOGICAL INSTRUCTIONS	COMPARE INSTRUCTION	SELECTIVE TESTS ON BITS (TMI, TPSU, & TPSL)
0	0	Zero	Equal	All bits 1
0	1	Positive	Greater Than	– – –
1	0	Negative	Less Than	Not all bits 1

IDC: Interdigit Carry/$\overline{\text{Borrow}}$ Bit

The IDC bit is affected by arithmetic operations as well as rotation.

0 = Interdigit borrow/no interdigit carry
1 = Interdigit carry/no interdigit borrow

OVF: Overflow Bit. Arithmetic Operation

The overflow bit in arithmetic operations is set as follows:

Operand 1 ± Operand 2 → Result

SIGN			ADD OVF	SUB OVF
OPERAND 1	OPERAND 2	RESULT		
+	+	+	0	0
+	+	–	1	0
+	–	+	0	0
+	–	–	0	1
–	+	+	0	1
–	+	–	0	0
–	–	+	1	0
–	–	–	0	0

OVF: Overflow Bit. Rotate Operation

Condition: WC = 1; if WC = 0, the OVF bit is not affected.

The overflow bit is set as follows:

OPERAND SIGN		OVF
BEFORE ROTATE	AFTER ROTATE	
+	+	0
+	–	1
–	+	0
–	–	0

C: Carry/$\overline{\text{Borrow}}$ Bit

The Carry bit is affected by arithmetic operations as well as rotation.

0 = borrow/no carry
1 = carry/no borrow

BIT TESTING PROCEDURES

The bits of a register Rx (register zero Ro or any register R1, R2 or R3 in the selected register bank) can be tested as follows:

```
                                              B    C
                                              Y    Y
                                              T    C
                                              E    L
                                              S    E
                                                   S
TEST FOR '0' IN BIT 3 OF Rx
TMI, Rx    H'08'                          1)  2    3
BCTR, 2    LBL    *Branch if bit 3 is zero.   2    3
                                              4    6
or:
ANDI, Rx   H'08'                          2)  2    2
BCTR, 0    LBL    *Branch if bit 3 is zero.   2    3
                                              4    5
```

While the second test is faster, it affects the contents of Rx.

BIT TESTING PROCEDURES (Continued)

TEST FOR '1' IN BIT 3 OF Rx

```
TMI, Rx    H'08'                              1)   2 | 3
BCTR, 0    LBL     *Branch if bit 3 is one.        2 | 3
                                                  ---+---
                                                   4 | 6
```

or:

```
ANDI, Rx   H'08'                              2)   2 | 2
BCFR, 0    LBL     *Branch if bit 3 is one.        2 | 3
                                                  ---+---
                                                   4 | 5
```

While the second test is faster, it affects the contents of Rx.

TEST FOR '0' IN BIT 1 OR BIT 3 OR BIT 6 OF Rx

```
TMI, Rx    H'4A'                              1)   2 | 3
BCTR, 2    LBL     *Branch if one of the           2 | 3
                   tested bits is zero.           ---+---
                                                   4 | 6
```

TEST FOR '1' IN BIT 1 OR BIT 3 OR BIT 6 OF Rx

```
ANDI, Rx   H'4A'                              2)   2 | 2
BCFR, 0    LBL     *Branch if one of the           2 | 3
                   tested bits is one.            ---+---
                                                   4 | 5
```

TEST FOR '0' IN BIT 1 AND BIT 3 AND BIT 6 OF Rx

```
ANDI, Rx   H'4A'                              2)   2 | 2
BCTR, 0    LBL     *Branch if all tested           2 | 3
                   bits are zero.                 ---+---
                                                   4 | 5
```

TEST FOR '1' IN BIT 1 AND BIT 3 AND BIT 6 OF Rx

```
TMI, Rx    H'4A'                              1)   2 | 3
BCTR, 0    LBL     *Branch if all tested           2 | 3
                   bits are one.                  ---+---
                                                   4 | 6
```

TEST FOR PATTERN IN Rx; e.g., x10xx01x

x = don't care

```
IORI, Rx   H'99'                              2)   2 | 2
COMI, Rx   H'DB'                                   2 | 2
BCTR, 0    LBL     *Branch if pattern              2 | 3
                   is true.                       ---+---
                                                   6 | 7
```

1) Contents of register Rx kept
2) Contents of register Rx lost

BYTE TESTING PROCEDURES

TEST FOR POSITIVE, NEGATIVE AND ZERO

All of the tests described below must be preceded by an operation on Rx which updates the contents of the condition register, e.g., by instructions such as LOAD, ADD, AND, COMPARE, ROTATE, I/O, etc.

	CC	OPERATION
Test for (Rx) $\geqslant$0	00 or 01	BCFR, 2
Test for (Rx) >0	01	BCTR, 1
Test for (Rx) = 0	00	BCTR, 0
Test for (Rx) <0	10	BCTR, 2
Test for (Rx) $\leqslant$0	00 or 10	BCFR, 1

TESTS ON THE CONTENTS OF A REGISTER BY USING COMPARE INSTRUCTIONS

Logical compare: (COM = 1 in PSL)
Comparison is made between two 8-bit unsigned binary numbers.

Arithmetic compare: (COM = 0 in PSL)
Comparison is made between two 8-bit signed numbers.

After execution of the logic or arithmetic compare instruction, the condition register (CC) is set to a specific value and tested as follows:

REGISTER-TO-REGISTER COMPARE		
Instruction used: **COMZ Rx**		
RESULT	CC	TEST
(Ro) $\geqslant$ (Rx)	00 or 01	BCFR, 2
(Ro) > (Rx)	01	BCTR, 1
(Ro) = (Rx)	00	BCTR, 0
(Ro) < (Rx)	10	BCTR, 2
(Ro) $\leqslant$ (Rx)	00 or 10	BCFR, 1

REGISTER TO CONSTANT OR MEMORY LOCATION		
Instructions used:		
COMI, Rx	**DATA**	
COMR, Rx	**RELATIVE LOCATION OF DATA**	
COMA, Rx	**LOCATION OF DATA**	
RESULT V=VALUE	CC	TEST
(Rx) $\geqslant$ V	00 or 01	BCFR, 2
(Rx) > V	01	BCTR, 1
(Rx) = V	00	BCTR, 0
(Rx) < V	10	BCTR, 2
(Rx) $\leqslant$ V	00 or 10	BCFR, 1

Whenever a compare instruction is used, the IDC, OVF, or C bits in the PSL are *not* affected.

TEST ON OVERFLOW (OVF in PSL)

The overflow bit is affected whenever arithmetic or rotate instructions are executed.

The *OVF bit* is set during an addition whenever the two operands have the same sign and the result has a different sign. During a subtraction, the *OVF bit* is set when the operands differ in sign and the result has a different sign than the first operand.

Examples:	(+A) + (+B) = (−C)	OVF
	(−A) + (−B) = (+C)	OVF
	(+A) − (−B) = (−C)	OVF
	(−A) − (+B) = (+C)	OVF

```
Test:   TPSL      H'04'    *OVF test
        BCTR, 0   LBL      *Branch if OVF = set
```

The *OVF bit* is set during rotate instructions with WC = 1 whenever the sign changes from positive to negative. If WC = 0, then rotate instructions do not affect the OVF bit.

Example:

```
        RRR, Rx            *Rotate right
        TPSL      H'04'    *Test OVF bit
        BCTR, 0   LBL      *Branch if OVF = set
```

TEST ON CARRY (C in PSL)

The carry bit is set to 1 by an add instruction that generates a carry and a sub-instruction that does *not* generate a borrow.

Example:

ADDITION

```
LODI, Rx  H'88'
ADDI, Rx  H'99'
TPSL      H'01'    *Test carry
BCTR, 0   LBL      *Branch if carry
```

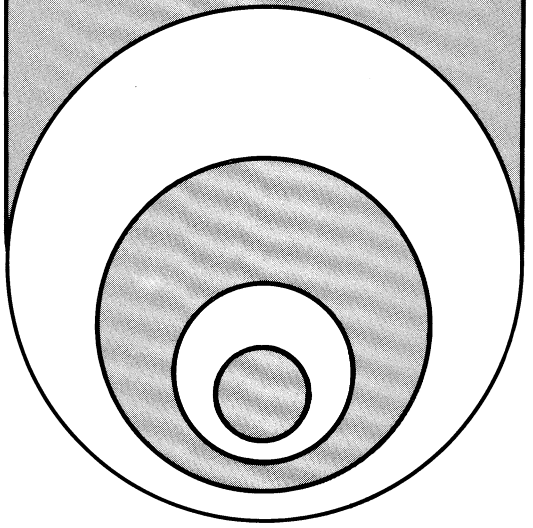
SUBTRACTION

```
LODI, Rx  H'40'
SUBI, Rx  H'30'
TPSL      H'01'    *Test borrow
BCTR, 0   LBL      *Branch if no borrow
```

When a rotate instruction is executed with WC = 1, the carry bit is also affected. Refer to the Signetics 2650 Microprocessor manual for a description of this operation.

signetics

MOS
MICROPROCESSOR



GENERAL DELAY ROUTINES AS52

signetics

GENERAL DELAY ROUTINES | AS52

2650 MICROPROCESSOR

APPLICATIONS MEMO

SUMMARY

In microprocessing applications, delay times are often required. A typical example is a delay time for a serial Teletypewriter interface. While delay times can be generated by counters, monostables, multivibrators, and other hardware, it is often simpler and more economical to use a short software routine.

This applications memo describes several ways of writing software delay time routines for the Signetics 2650 microprocessor. Time restrictions and formulas for calculating the delay time are given for each routine.

DELAY ROUTINES

In general, a delay can be implemented by setting a counter with a number N and decrementing this number by one until it is zero. If decrementing the number takes one clock period, then the total delay time is N clock periods.

In the 2650 microprocessor, the internal registers may be used as counters. The most useful instructions for decrementing are the "Branch on Decrementing Register" (BDRR and BDRA) instructions, which also test the content of a register for zero.

Figure 1 illustrates a flowchart of a delay routine. This routine consists of a setup part and a count loop. The count loop will be executed n times and the setup only once. Hence, the delay time is:

$$t_d = t_{su} + n \cdot t_{ct}$$

It is possible to increase the delay time by increasing n or by making t_{ct} longer. The latter can be done by inserting a fixed delay such as a No Operation (NOP) instruction in the count loop.

DELAY ROUTINE FLOWCHART

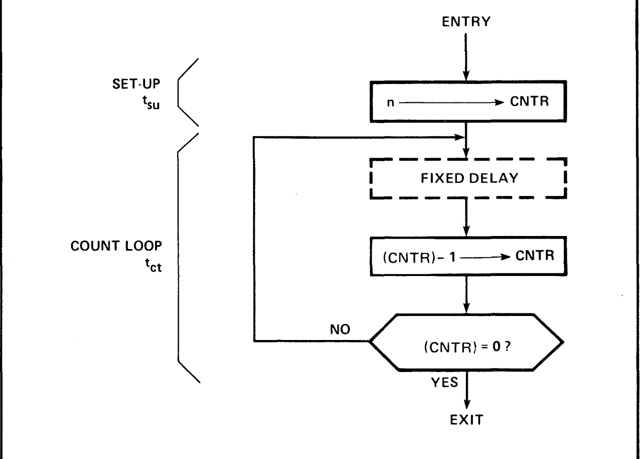


FIGURE 1

The program of the routine shown in Figure 1 is as follows:

	LODI, Rx n	Load n into register Rx	6 cp*
LOOP	NOP	No operation; fixed delay of 6 cp	6 cp
	BDRR, Rx LOOP	Decrement Rx; branch to loop if the result is not zero	9 cp

*cp = clock periods

With one NOP, the delay time is: $t_d = (6 + 15 \cdot n)$ cp. Without the NOP, the delay time is: $t_d = (6 + 9 \cdot n)$ cp. The maximum delay time is obtained when Rx is loaded with zero, since Rx will cycle through all the 256 possible states. When Rx = R0, the LODI, R0 0 instruction can be replaced by the EORZ R0 instruction, which saves one byte of code.

DELAY ROUTINE WITH FOUR REGISTERS

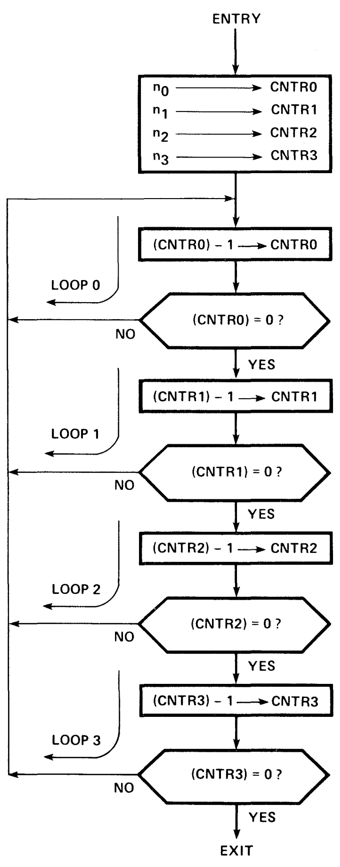


FIGURE 2

Another possible way of increasing the delay time is to repeat the count loop of Figure 1 several times. This can be done by repeating the instructions or by counting the repetitions of the count loop in another register. For example, this latter method can be expanded to include four internal registers. A flowchart of a delay routine using this technique is illustrated in Figure 2.

The number of times the processor executes the different loops shown in Figure 2 are:

loop 3	n_3
loop 2	$n_2 + (n_3 - 1)\ 256$
loop 1	$n_1 + (n_2 - 1)\ 256 + (n_3 - 1)\ 256^2$
loop 0	$n_0 + (n_1 - 1)\ 256 + (n_2 - 1)\ 256^2 + (n_3 - 1)\ 256^3$

Hence, the delay time of this routine is:

$$t_d = [24 + \{n_0 + n_1 + (n_1 - 1)\ 256 + n_2 + (n_2 - 1)\ (256 + 256^2) + n_3 + (n_3 - 1)\ (256 + 256^2 + 256^3)\}\ 9]\ \text{cp}$$

(If Rx is loaded with a zero, then n = 256 in the formula):

Table 1 shows six different delay routine programs along with specifications for each program. The delay time for these routines can be computed from the following equations.

Routine	**Delay Time**
a	$t_d = (6 + 9 \cdot n_0)\ \text{cp}$
b	$t_d = (6 + 15 \cdot n_0)\ \text{cp}$
c	$t_d = (2310 + 9 \cdot n_0)\ \text{cp}$
d	$t_d = \{12 + [n_0 + n_1 + (n_1 - 1)\ 256]\ 9\}\ \text{cp}$
e	$t_d = \{18 + [n_0 + n_1 + (n_1 - 1)\ 256 + n_2 + (n_2 - 1)\ (256^2 + 256)]\ 9\}\ \text{cp}$
f	$t_d = \{24 + [n_0 + n_1 + (n_1 - 1)\ 256 + n_2 + (n_2 - 1)\ (256^2 + 256) + n_3 + (n_3 - 1)\ (256^3 + 256^2 + 256)]\ 9\}\ \text{cp}$

TABLE 1

ROUTINE	POSSIBLE DELAY TIME (cp)		DELAY STEP (cp)	NUMBER OF BYTES	NUMBER OF REGISTERS	PROGRAM
	MIN*	MAX				
a	15	2310	9	4	1	LODI, R0 n_0 LOOP BDRR, R0 LOOP
b	21	3846	15	5	1	LODI, R0 n_0 LOOP NOP BDRR, R0 LOOP
c	2319	4614	9	6	1	LODI, R0 n_0 LOP 1 BDRR, R0 LOP 1 LOP 2 BDRR, R0 LOP 2
d	30	592.140	9	8	2	LODI, R0 n_0 LODI, R1 n_1 LOOP BDRR, R0 LOOP BDRR, R1 LOOP
e	45	$\approx 151.6 \times 10^6$**	9	12	3	LODI, R0 n_0 LODI, R1 n_1 LODI, R2 n_2 LOOP BDRR, R0 LOOP BDRR, R1 LOOP BDRR, R2 LOOP
f	60	$\approx 38.8 \times 10^9$***	9	16	4	LODI, R0 n_0 LODI, R1 n_1 LODI, R2 n_2 LODI, R3 n_3 LOOP BDRR, R0 LOOP BDRR, R1 LOOP BDRR, R2 LOOP BDRR, R3 LOOP

* cp = clock period. For 1MHz clock 1 cp = 1μs.
** For 1MHz clock this is about 2.5 minutes.
*** For 1MHz clock this is about 10.46 hours.

signetics

MOS
MICROPROCESSOR

BINARY ARITHMETIC ROUTINES.....AS53

signetics

BINARY ARITHMETIC ROUTINES AS53

2650 MICROPROCESSOR

APPLICATIONS MEMO

INTRODUCTION

Binary arithmetic routines, like addition, subtraction, multiplication, and division, are often used in microprocessor-based systems. This applications memo provides several suggested examples for processing binary arithmetic routines on the 2650 microprocessor. These examples include:

- SIGNED BINARY ADDITION/SUBTRACTION
  Two-byte operands giving a two-byte result.
- UNSIGNED BINARY MULTIPLICATION
  One-byte operands giving a two-byte result.
  Two-byte operands giving a four-byte result.
- SIGNED BINARY MULTIPLICATION
  One-byte operands giving a two-byte result.
  Two-byte operands giving a four-byte result.
- BINARY DIVISION – UNSIGNED AND SIGNED
  Two-byte dividend and quotient with one-byte divisor and remainder.

In these examples, emphasis is placed on minimizing program memory requirements rather than on processing speed. The different branch instructions and the indexing features of the Signetics 2650 proved useful in minimizing memory requirements.

	HARDWARE AFFECTED						
REGISTERS	R0 X	R1 X	R2	R3	R1′	R2′	R3′
PSU	F	II	SP				
PSL	CC X	IDC X	RS	WC X	OVF X	COM	C X

RAM REQUIRED (BYTES):	6
ROM REQUIRED (BYTES):	45
EXECUTION TIME:	Variable
MAXIMUM SUBROUTINE NESTING LEVELS:	None
ASSEMBLER/COMPILER USED:	PIPHASM

1. BINARY ADDITION/SUBTRACTION FOR TWO-BYTE SIGNED INTEGERS

FUNCTION:

Performs the addition or subtraction of two 2-byte signed integers giving a two-byte result.
(OPR1, OPR1 + 1) +/– (OPR2, OPR2 + 1) ⟶
RSLT, RSLT + 1

PARAMETERS:

Input: OPR1, OPR1 + 1 contains augend/subtrahend
OPR2, OPR2 + 1 contains addend/minuend
COM-flag in PSL indicates addition/subtraction:
COM = 0 addition
COM = 1 subtraction

Output: RSLT, RSLT + 1 contains sum/difference.
The condition code CC is set to the proper value of the two byte result.
OPR1, OPR2 and RSLT are MS-bytes.

SPECIAL REQUIREMENTS

None

Refer to Figures 1.1 and 1.2 for flowchart and program listing.

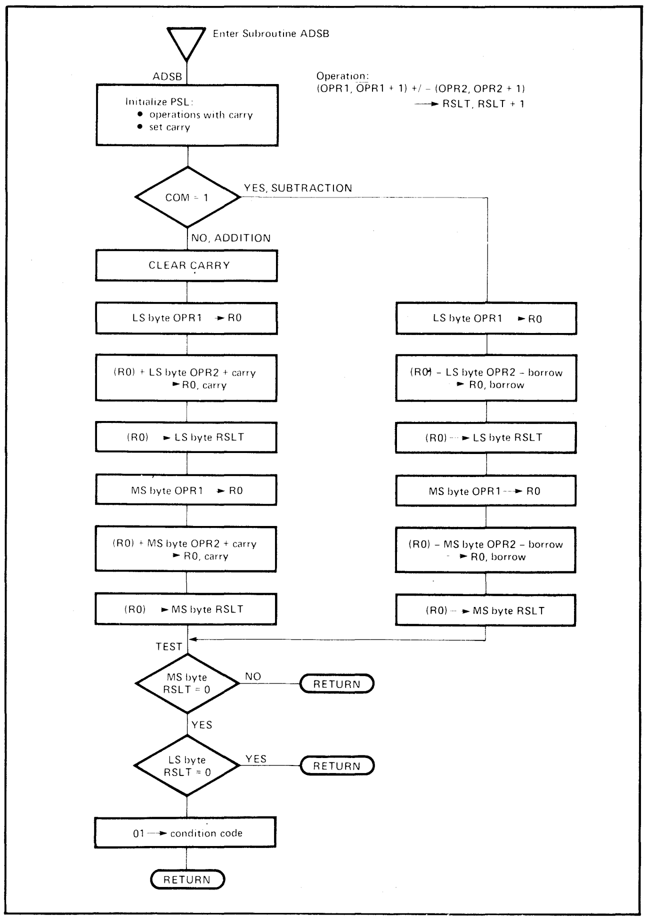


FIGURE 1-1 Flowchart for Double Precision Addition/Subtraction

```
                          * PD760010
                          **********************************************************
                          * BINARY DOUBLE PRECISION ADDITION/SUBTRACTION
                          **********************************************************
                          * OPERATION:
                          *    (OPR1,OPR1+1)+/-(OPR2,OPR2+1)-->RSLT,RSLT+1
                          * OPR1,OPR2,RSLT ARE MOST SIG BYTES
                          * COM IN PSL IS USED AS ADD/SUB FLAG
                          *    COM=0 IS ADD; COM=1 IS SUBTRACT
                          * AFTER ADD/SUB THE CC,OVF,AND C BITS IN PSL
                          *    ARE VALID FOR THE RESULT
                          *
                          * DEFINITION OF SYMBOLS
                          *
      0000                R0   EQU     0         PROCESSOR REGISTERS
      0001                R1   EQU     1
      0002                R2   EQU     2
      0003                R3   EQU     3
      0080                CC1  EQU     H'80'     PSL: MSB OF CONDITION CODE
      0040                CC0  EQU     H'40'          LSB OF CONDITION CODE
      0008                WC   EQU     H'08'          1=WITH,0=WITHOUT CARRY
      0002                COM  EQU     H'02'          1=LOGICAL,0=ARITH COMP
      0001                C    EQU     H'01'          CARRY/BORROW
      0000                Z    EQU     0         BRANCH COND: ZERO
      0003                UN   EQU     3                      UNCONDITIONAL
      0000                ON   EQU     0                      ALL BITS ARE 1
                          *
                               ORG     H'500'    START OF SUBROUTINE
                          *
0500  0500  77 09         ADSB PPSL    WC+C      ARITH WITH CARRY;SET CARRY
0502        05 02              LODI,R1 2         LOAD INDEX REGISTER
0504        B5 02              TPSL    COM
0506        18 0F              BCTR,ON LPSB      BRANCH IF SUBTRACTION
0508        75 01              CPSL    C         ADDITION,CLEAR CARRY
050A  050A  0D 45 2D      LPAD LODA,R0 OPR1,R1,- BYTE OF FIRST OPERAND TO R0
050D        8D 65 2F           ADDA,R0 OPR2,R1   ADD BYTE OF SECOND OPERAND
0510        CD 65 31           STRA,R0 RSLT,R1   STORE RESULT
0513        59 75              BRNR,R1 LPAD      BRANCH IF NOT DONE
0515        1B 0B              BCTR,UN TEST
0517  0517  0D 45 2D      LPSB LODA,R0 OPR1,R1,- BYTE OF FIRST OPERAND TO R0
051A        AD 65 2F           SUBA,R0 OPR2,R1   SUB BYTE OF SECOND OPERAND
051D        CD 65 31           STRA,R0 RSLT,R1   STORE RESULT
0520        59 75              BRNR,R1 LPSB      BRANCH IF NOT DONE
0522  0522  98 08         TEST BCFR,Z  RTRN      RETURN IF MS BYTE NOT ZERO
0524        0C 05 32           LODA,R0 RSLT+1
0527        14                 RETC,Z            RETURN IF LS BYTE ALSO ZERO
0528        75 80              CPSL    CC1       SET CC. TO 01 (POSITIVE)
052A        77 40              PPSL    CC0
052C  052C  17            RTRN RETC,UN
                          *
      052D                OPR1 RES     2         LOCATION OF: FIRST  OPERAND
      052F                OPR2 RES     2                      SECOND OPERAND
      0531                RSLT RES     2                      RESULT
                               END
```

FIGURE 1-2

2. BINARY MULTIPLICATION FOR ONE-BYTE UNSIGNED INTEGERS

FUNCTION:

One byte by one byte multiplication for unsigned integers, giving a two-byte result.

(OPR1) X (OPR2) $\longrightarrow$ RSLT, RSLT + 1

PARAMETERS:

Input	OPR1 contains multiplier OPR2 contains multiplicand
Output:	RSLT contains high-order product-byte. RSLT + 1 contains low-order product-byte.

SPECIAL REQUIREMENTS:

None

Refer to Figures 2.1 and 2.2 for flowchart and program listing.

	HARDWARE AFFECTED						
REGISTERS	R0	R1	R2	R3	R1′	R2′	R3′
	X	X	X	X			
PSU	F	II	SP				
PSL	CC	IDC	RS	WC	OVF	COM	C
	X	X		X	X		X

RAM REQUIRED (BYTES):	4
ROM REQUIRED (BYTES):	29
EXECUTION TIME:	Variable
MAXIMUM SUBROUTINE NESTING LEVELS:	None
ASSEMBLER/COMPILER USED:	PIPHASM

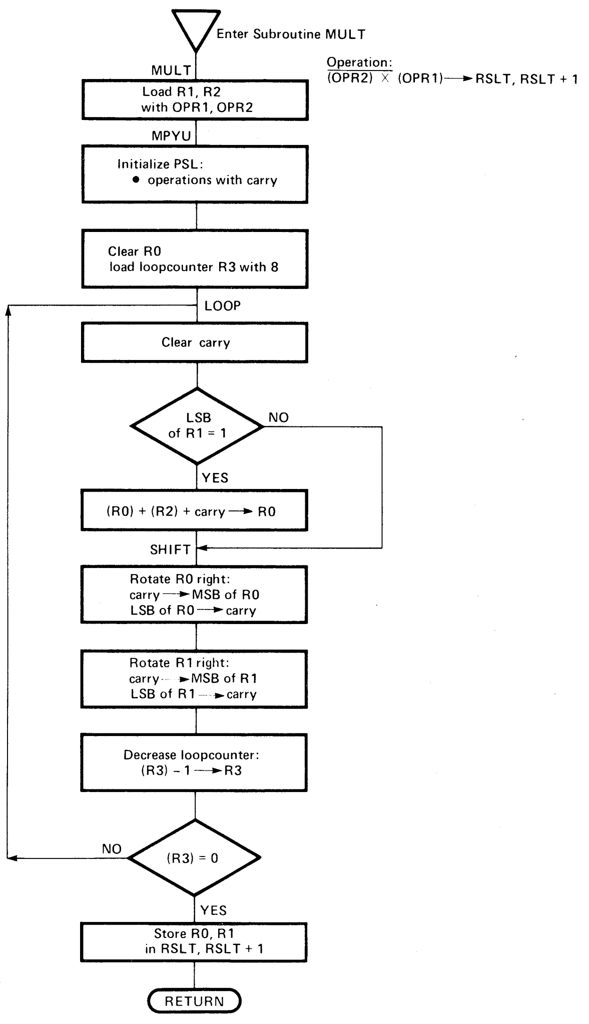

Enter Subroutine MULT
Operation:
(OPR2) X (OPR1) → RSLT, RSLT + 1
MULT
Load R1, R2 with OPR1, OPR2
MPYU
Initialize PSL: • operations with carry
Clear R0 load loopcounter R3 with 8
LOOP
Clear carry
LSB of R1 = 1
NO
YES
(R0) + (R2) + carry → R0
SHIFT
Rotate R0 right: carry → MSB of R0 LSB of R0 → carry
Rotate R1 right: carry → MSB of R1 LSB of R1 → carry
Decrease loopcounter: (R3) - 1 → R3
NO
(R3) = 0
YES
Store R0, R1 in RSLT, RSLT + 1
RETURN

FIGURE 2-1 Flowchart for Unsigned Multiplication (One-Byte Operands; Two-Byte Result)

```
                          *      PD760030                                         *
                          ******************************************************************
                          *    BINARY MULTIPLICATION FOR 2 UNSIGNED INTEGERS
                          ******************************************************************
                          *
                          *    MULTIPLIER IS IN OPR1
                          *    MULTIPLICAND IS IN OPR2
                          *    RESULT WILL BE STORED IN RSLT,RSLT+1 (RSLT = MS BYTE)
                          *
                          *
                          *
                          *
                          *        SYMBOL DEFINITIONS
        0000              R0      EQU     0
        0001              R1      EQU     1
        0002              R2      EQU     2
        0003              R3      EQU     3
        0001              R4      EQU     1
        0002              R5      EQU     2
        0003              R6      EQU     3
        0003              UN      EQU     3              UNCONDITIONAL BRANCHING
        0000              ON      EQU     0
        0002              LT      EQU     2
        0000              Z       EQU     0
        0001              P       EQU     1
        0002              N       EQU     2
        0008              WC      EQU     8
        0001              C       EQU     1
        0040              F       EQU     H'40'
        0004              OVF     EQU     4
        0002              COM     EQU     2
                          *
                          * R/W MEMORY
                          *
                                ORG   H'500'
        0500              OPR1    RES     2
        0502              OPR2    RES     2
        0504              RSLT    RES     4
                          *
                          *
                          *
                                  ORG  H'600'
0600    0600   0D 05 00   MULT    LODA,R1  OPR1          GET OPERAND IN R1
0603           0E 05 02           LODA,R2  OPR2          GET OPERAND IN R2
0606    0606   77 08      MPYU    PPSL     WC            ARITH
0608           20                 EORZ     R0            CLEAR R0
0609           07 08              LODI,R3  8             LOAD LOOP COUNTER R3
060B    060B   75 01      LOOP    CPSL     C             CLEAR CARRY
060D           F5 01              TMI,R1   H'01'
060F           98 01              BCFR,ON  SHFT          SKIP ADDITION IF LSB R1=0
0611           82                 ADDZ     R2            ADD MULTIPLICAND TO PARTIAL PROD
0612    0612   50         SHFT    RRR,R0                 ROTATE PARTIAL PROD AND MULTIPLIER
0613           51                 RRR,R1
0614           FB 75              BDRR,R3  LOOP          BRANCH TO LOOP IF NOT READY
0616           CC 05 04           STRA,R0  RSLT          SAVE RESULT IN RESULT AREA
0619           CD 05 04           STRA,R1  RSLT+1        SAVE RESULT IN RESULT AREA
061C           17                 RETC,UN                RETURN TO MAIN PROGRAM
```

FIGURE 2-2

3. BINARY MULTIPLICATION FOR TWO-BYTE UNSIGNED INTEGERS

FUNCTION:

Two byte by two byte multiplication for unsigned integers, giving a four byte result.
(OPR2, OPR2 + 1) X (OPR1, OPR1 + 1) ⟶
RSLT, RSLT + 1, RSLT + 2, RSLT + 3

PARAMETERS:

Input:	(OPR1, OPR1 + 1) contains multiplier (OPR2, OPR2 + 1) contains multiplicand
Output:	RSLT, RSLT + 1, RSLT + 2, RSLT + 3 contains product. OPR1, OPR2, and RSLT are most-significant bytes.

SPECIAL REQUIREMENTS:

None

Refer to Figures 3.1 and 3.2 for flowchart and program listing.

	HARDWARE AFFECTED						
REGISTERS	R0 X	R1 X	R2	R3 X	R1′	R2′	R3′
PSU	F	II	SP				
PSL	CC X	IDC X	RS	WC X	OVF X	COM	C X

RAM REQUIRED (BYTES): 8

ROM REQUIRED (BYTES): 57

EXECUTION TIME: Variable

MAXIMUM SUBROUTINE
NESTING LEVELS: None

ASSEMBLER/COMPILER USED: PIPHASM

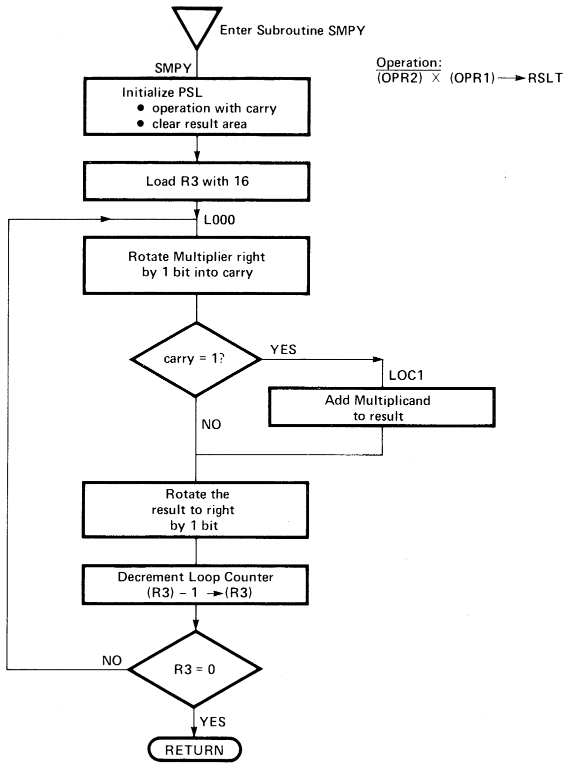


FIGURE 3-1 Flowchart for Unsigned Multiplication (Two-Byte Operands; Four-Byte Result)

```
                         *    PD760031                                                    *
                         **********************************************************************
                         *  BINARY MULTIPLICATION FOR 2 TWO-BYTE INTEGERS
                         **********************************************************************
                         *
                         * MULTIPLIER IS IN OPR1 , OPR1+1
                         * MULTIPLICAND IS IN OPR2 ,OPR2+1
                         * RESULT WILL BE IN RSLT ,RSLT+1 ,RSLT+2 ,RSLT+3
                              ORG   H'790'
                         *
0790  0790  77 08        SMPY     PPSL     WC               SET MODE
0792        20                    EORZ     R0
0793        CC 05 04              STRA,R0  RSLT             CLEAR  RESULT
0796        CC 05 05              STRA,R0  RSLT+1           CLEAR RESULT +1
0799        07 10                 LODI,R3  16               LOAD COUNT
079B  079B  05 FE        LOOO     LODI,R1  -2               TO GET 254
079D        75 01                 CPSL     C                CLEAR CARRY
079F  079F  0D 64 02     LOCO     LODA,R0  OPR1-256+2,R1    FOR INDEXING INTO OPR1
07A2        50                    RRR,R0                    ROTATE RIGHT WITH C
07A3        CD 64 02              STRA,R0  OPR1-256+2,R1
07A6        D9 77                 BIRR,R1  LOCO             ROTATE 2ND TIME
                         *                   THIS ROTATES MULTIPLIER BY 1 BIT TO GET THE LSB
                         *                   INTO CARRY
07A8        20                    EORZ     R0               CLEAR R0
07A9        D0                    RRL,R0                    GET CARRY INTO LSB
07AA        F8 02                 BDRR,R0  LOC1
07AC        1B 0D                 BCTR,UN  LOC4
                         *
07AE  07AE  05 02        LOC1     LODI,R1  2                GET INDEX
07B0  07B0  0D 65 03     LOC2     LODA,R0  RSLT-1,R1        ADD MULTIPLICAND TO PRODUCT
07B3        8D 65 01              ADDA,R0  OPR2-1,R1
07B6        CD 65 03              STRA,R0  RSLT-1,R1
07B9        F9 75                 BDRR,R1  LOC2             FINISH THE ADD
                         *
                         *
07BB  07BB  05 FC        LOC4     LODI,R1  -4               ROTATE THE PRODUCT TO RIGHT
07BD  07BD  0D 64 08     LOC5     LODA,R0  RSLT-256+4,R1
07C0        50                    RRR,R0                    ROTATE RESULT
07C1        CD 64 08              STRA,R0  RSLT-256+4,R1
07C4        D9 77                 BIRR,R1  LOC5
07C6        FB 53                 BDRR,R3  LOOO             FINISH THE LOOP
07C8        17                    RETC,UN
```

FIGURE 3-2

4. BINARY MULTIPLICATION FOR ONE-BYTE SIGNED INTEGERS

FUNCTION:

One byte by one byte multiplication for signed integers giving a two-byte result.
(OPR1) X (OPR2) ⟶ RSLT, RSLT + 1
The Booth algorithm is used (see Figure 4.1).

PARAMETERS:

Input:	OPR1 contains multiplier
	OPR2 contains multiplicand
Output:	RSLT contains high-order product byte.
	RSLT + 1 contains low-order product byte.

SPECIAL REQUIREMENTS:

None

Refer to Figures 4.1 and 4.2 for flowcharts and to Figure 4.3 for program listing.

RAM REQUIRED (BYTES):	4
ROM REQUIRED (BYTES):	51
EXECUTION TIME:	Variable
MAXIMUM SUBROUTINE NESTING LEVELS:	None
ASSEMBLER/COMPILER USED:	PIPHASM

	HARDWARE AFFECTED						
REGISTERS	R0	R1	R2	R3	R1'	R2'	R3'
	X	X	X	X			
PSU	F	II	SP				
PSL	CC	IDC	RS	WC	OVF	COM	C
	X	X		X	X		X

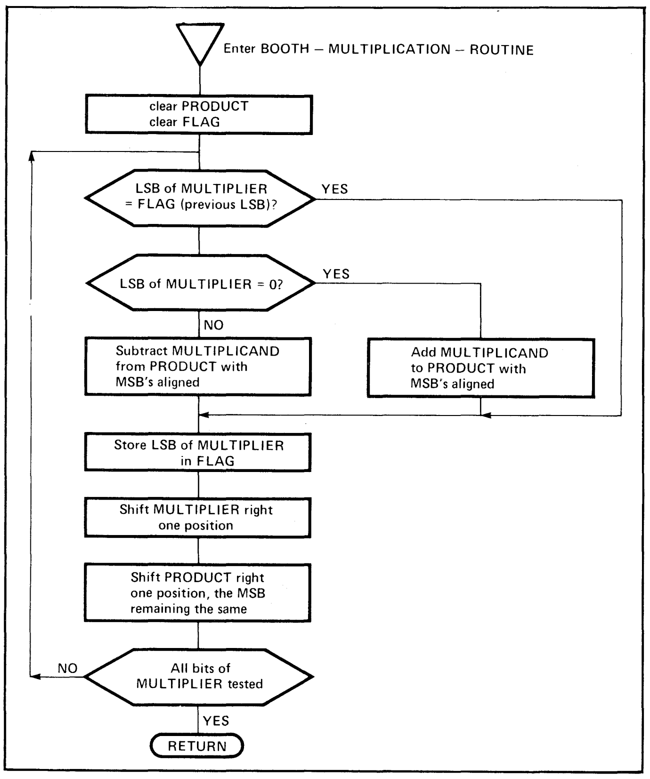

Enter BOOTH – MULTIPLICATION – ROUTINE
clear PRODUCT
clear FLAG
LSB of MULTIPLIER = FLAG (previous LSB)?
YES
LSB of MULTIPLIER = 0?
YES
NO
Subtract MULTIPLICAND from PRODUCT with MSB's aligned
Add MULTIPLICAND to PRODUCT with MSB's aligned
Store LSB of MULTIPLIER in FLAG
Shift MULTIPLIER right one position
Shift PRODUCT right one position, the MSB remaining the same
NO
All bits of MULTIPLIER tested
YES
RETURN

FIGURE 4-1 Flowchart of Booth Algorithm
Multiplicand X Multiplier → Product

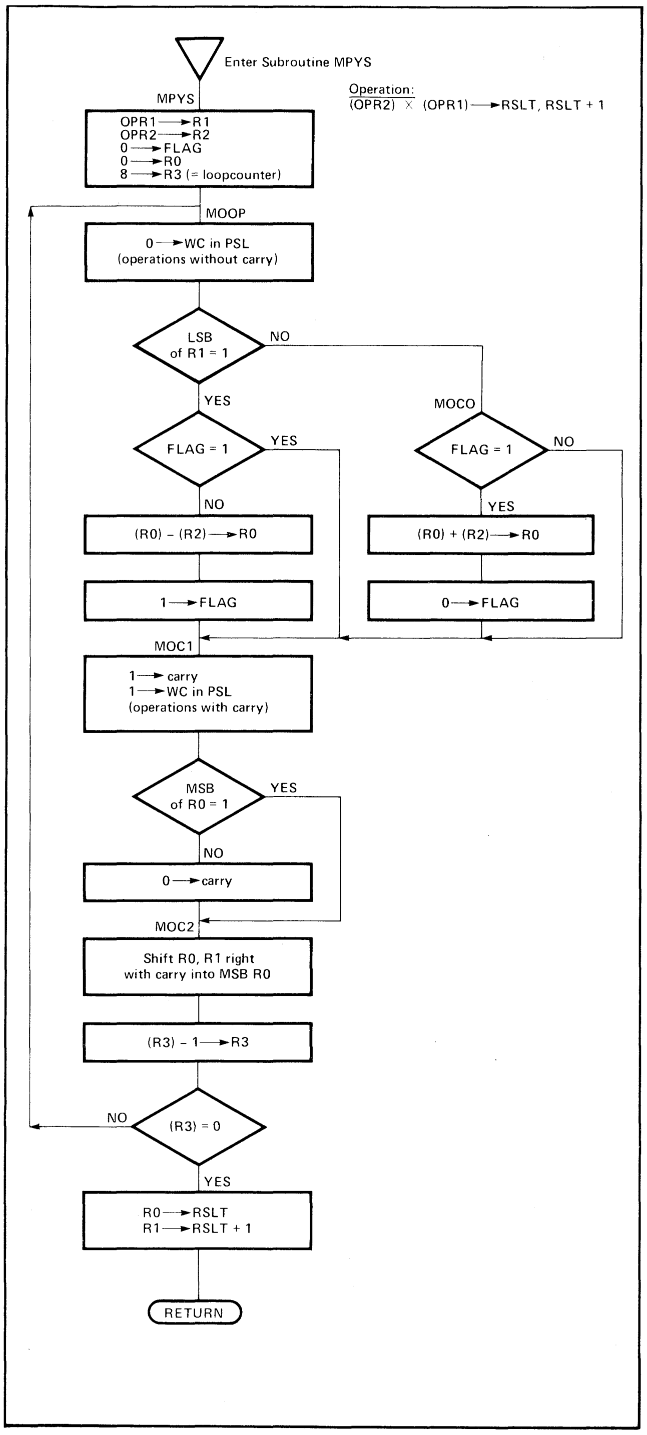


FIGURE 4-2 Flowchart for Signed Multiplication Using Booth Algorithm (One-Byte Operands; Two-Byte Result)

```
                            *       PD760032
                            ****************************************************************
                            * BINARY MULTIPLICATION USING BOOTH-ALGORITHM
                            * FOR 2 ONE-BYTE SIGNED  INTEGERS.
                            ******************************************************************
                            * FIRST OPERAND IS IN OPR1
                            * SECOND OPERAND IS IN OPR2 (OPR2) ≠ H'80'
                            * PRODUCT WILL BE IN RSLT,RSLT+1
                            *
                                    ORG   H'800'
0800  0800  74 40           MPYS    CPSU      F          CLEAR FLAG IN PSU
0802        0D 05 00                LODA,R1   OPR1       GET 1ST OPERAND
0805        0E 05 02                LODA,R2   OPR2       GET 2ND OPERAND
0808        07 08                   LODI,R3   8          LOAD LOOP COUNTER R3
080A        20                      EORZ      R0         CLEAR R0
080B  080B  75 08           MOOP    CPSL      WC         CLEAR WC IN PSL
080D        F5 01                   TMI,R1    H'01'
080F        98 09                   BCFR,ON   MOC0       LSB OF R1 SET?
0811        B4 40                   TPSU      F          YES
0813        18 0C                   BCTR,ON   MOC1       FLAG =1?
0815        A2                      SUBZ      R2         NO,SUBTRACT WITHOUT BORROW
0816        76 40                   PPSU      F          SET FLAG
0818        1B 07                   BCTR,UN   MOC1       BRANCH TO DOUBLE SHIFT
081A  081A  B4 40           MOC0    TPSU      F          LSB OF R1 WAS 0
081C        98 03                   BCFR,ON   MOC1       FLAG =1?
081E        82                      ADDZ      R2         YES,ADD WITHOUT CARRY
081F        74 40                   CPSU      F          CLEAR FLAG
0821  0821  77 09           MOC1    PPSL      WC+C       SET C AND WC
0823        60                      IORZ      R0
0824        1A 02                   BCTR,N    MOC2       MSB OF R0 SET?
0826        75 01                   CPSL      C          NO,CLEAR CARRY
0828  0828  50              MOC2    RRR,R0               SHIFT R0 R1 RIGHT
0829        51                      RRR,R1               MSB OF R0 IS SAME
082A        FB 5F                   BDRR,R3   MOOP       BRANCH TO LOOP IF NOT READY
082C        CC 05 04                STRA,R0   RSLT       STORE RESULT
082F        CD 05 05                STRA,R1   RSLT+1
0832        17                      RETC,UN              EXIT SUBROUTINE MPYS
                            *
```

FIGURE 4-3

5. BINARY MULTIPLICATION FOR TWO-BYTE SIGNED INTEGERS

FUNCTION:

Two byte by two byte multiplication for signed integers giving a four byte result.
(OPR1, OPR1 + 1) X (OPR2, OPR2 + 1)
⟶ RSLT, RSLT + 1, RSLT + 2, RSLT + 3.
The Booth algorithm (Figure 4.1) is used.

PARAMETERS:

Input:	OPR1, OPR1 + 1 contains multiplicand OPR2, OPR2 + 1 contains multiplier
Output:	RSLT, RSLT + 1, RSLT + 2, RSLT + 3 contains product. OPR1, OPR2, and RSLT are most-significant bytes.

SPECIAL REQUIREMENTS

None

Refer to Figure 5.1 for flowchart and to Figure 5.2 for program listing.

	HARDWARE AFFECTED						
REGISTERS	R0	R1	R2	R3	R1′	R2′	R3′
	X	X	X	X			
PSU	F	II	SP				
PSL	CC	IDC	RS	WC	OVF	COM	C
	X	X		X	X		X

RAM REQUIRED (BYTES):	8
ROM REQUIRED (BYTES):	71
EXECUTION TIME:	Variable
MAXIMUM SUBROUTINE NESTING LEVELS:	None
ASSEMBLER/COMPILER USED:	PIPHASM

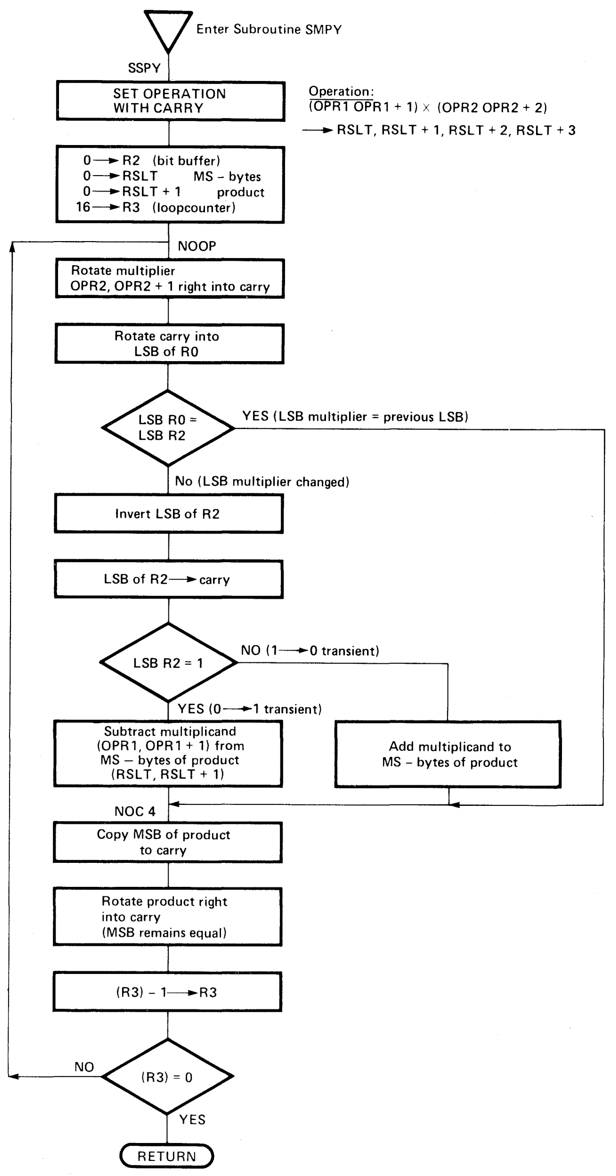

Enter Subroutine SMPY
SSPY
SET OPERATION WITH CARRY
Operation:
(OPR1 OPR1 + 1) × (OPR2 OPR2 + 2)
⟶ RSLT, RSLT + 1, RSLT + 2, RSLT + 3
0 ⟶ R2 (bit buffer)
0 ⟶ RSLT MS – bytes
0 ⟶ RSLT + 1 product
16 ⟶ R3 (loopcounter)
NOOP
Rotate multiplier OPR2, OPR2 + 1 right into carry
Rotate carry into LSB of R0
LSB R0 = LSB R2
YES (LSB multiplier = previous LSB)
No (LSB multiplier changed)
Invert LSB of R2
LSB of R2 ⟶ carry
LSB R2 = 1
NO (1 ⟶ 0 transient)
YES (0 ⟶ 1 transient)
Subtract multiplicand (OPR1, OPR1 + 1) from MS – bytes of product (RSLT, RSLT + 1)
Add multiplicand to MS – bytes of product
NOC 4
Copy MSB of product to carry
Rotate product right into carry (MSB remains equal)
(R3) – 1 ⟶ R3
NO
(R3) = 0
YES
RETURN

FIGURE 5-1 Flowchart for Signed Multiplication Using Booth Algorithm (Two-Byte Operands; Four-Byte Result)

```
                                  *    PD760033
                                  ************************************************************************
                                  *  BINARY MULTIPLICATION FOR TWO BYTE SIGNED INTEGERS
                                  ************************************************************************
                                  * MULTIPLICAND IS IN LOCATIONS OPR1,OPR1+1
                                  * MULTIPLIER IS IN LOCATIONS OPR2,OPR2+1
                                  *
                                  * RESULT WILL BE STORED IN RSLT,RSLT+1,RSLT+2,RSLT+3
                                  *
                                  *   AFTER MULTIPLICATION THE MULTIPLICAND IS UNCHANGED
                                  *   THE MULTIPLIER IS DESTROYED
                                  *   THE MULTIPLICAND MUST BE UNEQUAL H'8000'
                                  ************************************************************************
0833  0833  77 08                 SSPY     PPSL      WC             ARITH AND ROTATE WITH C
0835        20                             EORZ      R0             CLEAR R0
0836        C2                             STRZ      R2             CLEAR R2
0837        CC 05 04                       STRA,R0   RSLT           CLEAR 2 MSBYTES OF PRODUCT
083A        CC 05 05                       STRA,R0   RSLT+1
083D        07 10                          LODI,R3   16             LOAD LOOP COUNTER R3
083F  083F  05 FE                 NOOP     LODI,R1   -2             LOAD INDEX REG WITH 254
0841  0841  0D 64 04              NOC0     LODA,R0   OPR2-256+2,R1  ROTATE MULTIPLIER
0844        50                             RRR,R0                   INTO CARRY
0845        CD 64 04                       STRA,R0   OPR2-256+2,R1
0848        D9 77                          BIRR,R1   NOC0           BRANCH IF NOT DONE
084A        20                             EORZ      R0             CLEAR R0
084B        D0                             RRL,R0                   ROTATE CARRY IN LSB OF R0
084C        22                             EORZ      R2             LSB OF R0 BECOMES 1 FOR CHANGE
084D        18 19                          BCTR,Z    NOC4           BRANCH IF NO CHANGE
084F        22                             EORZ      R2             INVERT LSB OF R2
0850        C2                             STRZ      R2             RESTORE NEW R2
0851        50                             RRR,R0                   LSB OF R2 INTO CARRY OR BORROW
0852        05 02                          LODI,R1   2              LOAD INDEX
0854  0854  0D 45 04              NOC1     LODA,R0   RSLT,R1,-      LOAD BYTE OF RSLT IN R0
0857        F6 01                          TMI,R2    1
0859        18 05                          BCTR,ON   NOC2           BRANCH TO SUBTRACT IF LSB R2=1
085B        8D 65 00                       ADDA,R0   OPR1,R1        ADD BYTE MPLCND TO RSLT
085E        1B 03                          BCTR,UN   NOC3
0860  0860  AD 65 00              NOC2     SUBA,R0   OPR1,R1        SUB BYTE MPLCND FROM RSLT
0863  0863  CD 65 04              NOC3     STRA,R0   RSLT,R1        RESTORE INTERMEDIATE RSLT
0866        59 6C                          BRNR,R1   NOC1           BRANCH IF ADD SUBTRACT NOT READY
                                  *
0868  0868  0C 05 04              NOC4     LODA,R0   RSLT
086B        D0                             RRL,R0
086C        04 FC                          LODI,R0   -4             LOAD INDEX
086E  086E  0D 64 08              NOC5     LODA,R0   RSLT-256+4,R1  FETCH MS BYTE PRODUCT
0871        50                             RRR,R0                   ROTATE RSLT,PROD+1 ETC TO RIGHT
0872        CD 64 08                       STRA,R0   RSLT-256+4,R1  KEEPING MSB SAME
0875        D9 77                          BIRR,R1   NOC5           BRANCH IF NOT DONE
0877        FB 46                          BDRR,R3   NOOP           BRANCH IF LOOP NOT READY
0879        17                             RETC,UN                  RETURN TO MAIN PROGRAM
                                            END
```

FIGURE 5-2

6. BINARY DIVISION

A. UNSIGNED INTEGERS TWO-BYTE DIVIDEND; ONE-BYTE DIVISOR

FUNCTION:

Division of a two byte dividend by a one byte divisor, resulting in a two-byte quotient and a one-byte remainder.

$$\frac{(\text{DVDN, DVDN} + 1)}{(\text{DVSR})} \longrightarrow \begin{cases} \text{DVDN, DVDN} + 1 & \text{(quotient)} \\ \text{R1} & \text{(remainder)} \end{cases}$$

PARAMETERS:

Input:	DVDN, DVDN + 1	contains dividend
	DVSR	contains divisor
	DVDN is most-significant byte	
Output:	DVDN, DVDN + 1	contains quotient
	R1	contains remainder
	DVDN is most-significant byte.	
	Dividend is destroyed after execution of division.	

SPECIAL REQUIREMENTS:

None

Refer to Figure 6.1 for flowchart and to Figure 6.2 for program listing.

	HARDWARE AFFECTED						
REGISTERS	R0 X	R1 X	R2 X	R3 X	R1′	R2′	R3′
PSU	F	II	SP				
PSL	CC X	IDC X	RS	WC X	OVF X	COM X	C X

RAM REQUIRED (BYTES):	3
ROM REQUIRED (BYTES):	45
EXECUTION TIME:	Variable
MAXIMUM SUBROUTINE NESTING LEVELS:	None
ASSEMBLER/COMPILER USED:	PIPHASM

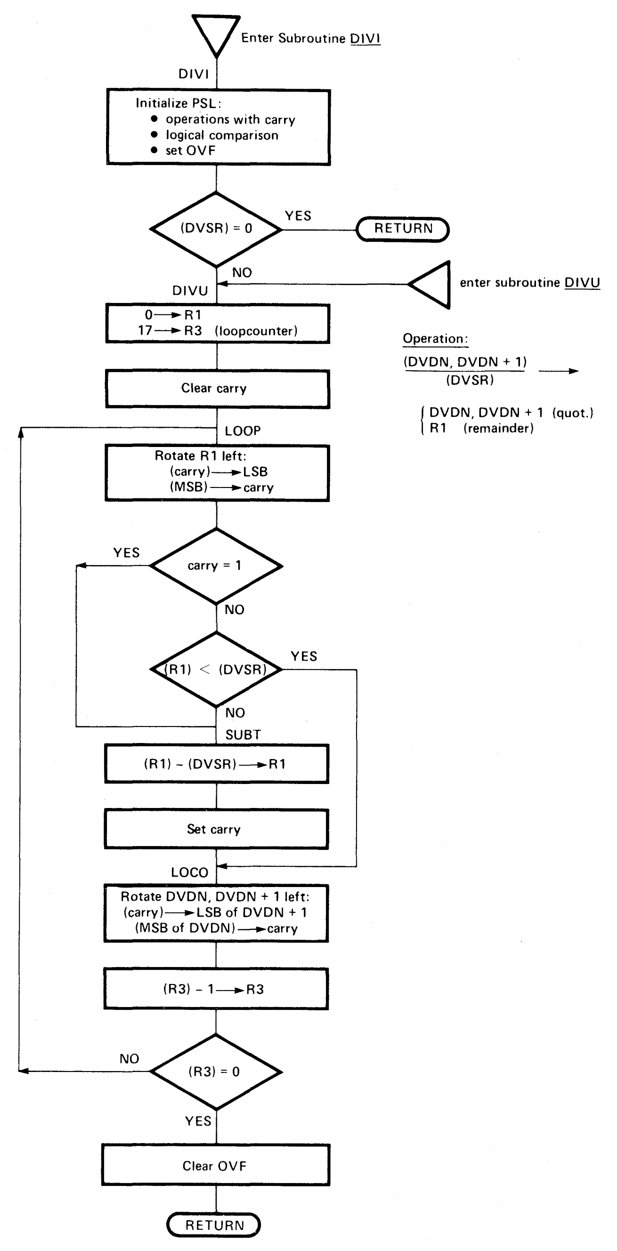


FIGURE 6-1 Flowchart for Unsigned Division (Dividend or Quotient: Two-Bytes; Divisor or Remainder: One-Byte)

```
                              *      PD760040                                                  *
                              ******************************************************************
                              *    BINARY DIVISIONS FOR INTEGERS
                              ******************************************************************
                              * DIVIDEND IS IN DVDN,DVDN+1 16 BITS
                              * DIVISOR IS IN  DVSR        8 BITS
                              * QUOTIENT WILL BE IN DVDN,DVDN+1 16 BITS
                              * AFTER DIVISION, DIVIDEND WILL BE DESTROYED
                              * R1 WILL HOLD REMAINDER
                              * OVF=1 IMPLIES OVERFLOW
                              *
                              *
                              *          SYMBOL DEFINITIONS
      0000                    R0     EQU     0
      0001                    R1     EQU     1
      0002                    R2     EQU     2
      0003                    R3     EQU     3
      0001                    R4     EQU     1
      0002                    R5     EQU     2
      0003                    R6     EQU     3
      0003                    UN     EQU     3             UNCONDITIONAL BRANCHING
      0001                    C      EQU     1
      0000                    ON     EQU     0
      0002                    LT     EQU     2
      0000                    Z      EQU     0
      0000                    EQ     EQU     0
      0001                    P      EQU     1
      0002                    N      EQU     2
      0008                    WC     EQU     8
      0004                    OVF    EQU     4
      0002                    COM    EQU     2
                              *
                                     ORG     H'500'        UNSIGNED DIVISION SUBROUTINE
                              *
0500  0500  77 0E             DIVI   PPSL    WC+OVF+COM    ARITH ROTATE WITH CARRY
0502        0C 06 02                 LODA,R0 DVSR          FETCH DIVISOR
0505        14                       RETC,Z                RETURN WITH OVF =1 IF DVSR =0
                              *
0506  0506  05 00             DIVU   LODI,R1 0             CLR R1
0508        07 11                    LODI,R3 17            LOAD LOOP COUNTER R3
050A        75 01                    CPSL    C             CLEAR CARRY
050C  050C  D1                LOOP   RRL,R1                ROTATE CARRY IN LSB OF R1
050D        B5 01                    TPSL    C
050F        18 05                    BCTR,ON SUBT          GO TO SUBTRACT IF CARRY =1
0511        ED 06 02                 COMA,R1 DVSR
0514        1A 07                    BCTR,LT LOC0          IF R1<DVSR,NO SUBTRACTION
0516  0516  77 01             SUBT   PPSL    C             CLR BORROW
0518        AD 06 02                 SUBA,R1 DVSR          SUBTR DVSR FROM REMAINDER
051B        77 01                    PPSL    C             SET CARRY
051D  051D  06 02             LOC0   LODI,R2 2             LOAD INDEX REGISTERR
051F  051F  0E 46 00          LOC1   LODA,R0 DVDN,R2,-     ROTATE QUOTIENT BIT
0522        D0                       RRL,R0                DVDN,DVDN+1 AND MSB OF
0523        CE 66 00                 STRA,R0 DVDN,R2       DVDN INTO CARRY
0526        5A 77                    BRNR,R2 LOC1          BRANCH IF ROTATE NOT READY
0528        FB 62                    BDRR,R3 LOOP          BRANCH IF DIVISION NOT READY
052A        75 04                    CPSL    OVF           CLEAR OVF IN PSL
052C        17                       RETC,UN               RETURN TO MAIN PROGRAM
                              *
```

FIGURE 6-2

B. SIGNED INTEGERS TWO-BYTE DIVIDEND; ONE-BYTE DIVISOR

FUNCTION:

Division of a two-byte dividend by a one-byte divisor, resulting in a two-byte quotient and a one-byte remainder.

$$\frac{\text{(DVDN, DVDN + 1)}}{\text{(DVSR)}} \longrightarrow \begin{cases} \text{DVDN, DVDN + 1} & \text{(quotient)} \\ \text{R1} & \text{(remainder)} \end{cases}$$

PARAMETERS:

Input:	DVDN, DVDN + 1	contains dividend
	DVSR	contains divisor
	DVDN is most-significant byte.	
Output:	DVDN, DVDN + 1	contains quotient
	R1	contains remainder
	DVDN is most-significant byte.	
	Dividend is destroyed after execution of division; negative divisor becomes positive	

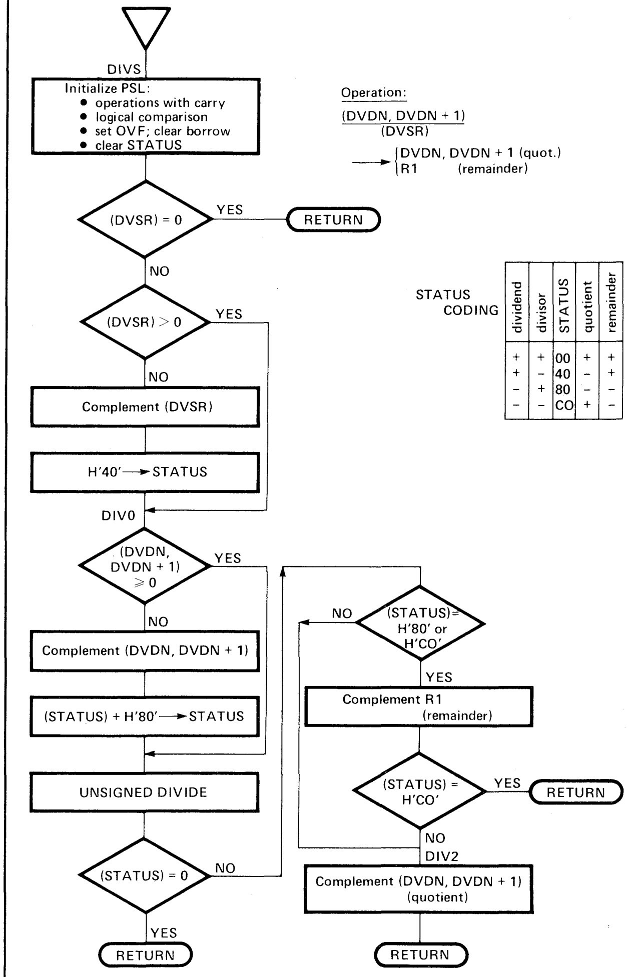
SPECIAL REQUIREMENTS:

Software: Unsigned division subroutine

Refer to Figure 6.3 for flowchart and to Figure 6.4 for program listing.

	HARDWARE AFFECTED						
REGISTERS	R0	R1	R2	R3	R1′	R2′	R3′
	X	X	X	X			
PSU	F	II	SP				
PSL	CC	IDC	RS	WC	OVF	COM	C
	X	X		X	X	X	X

RAM REQUIRED (BYTES):	4
ROM REQUIRED (BYTES):	61
EXECUTION TIME:	Variable
MAXIMUM SUBROUTINE NESTING LEVELS:	1
ASSEMBLER/COMPILER USED:	PIPHASM



dividend	divisor	STATUS	quotient	remainder
+	+	00	+	+
+	−	40	−	+
−	+	80	−	−
−	−	C0	+	−

FIGURE 6-3 Flowchart for Signed Division (Dividend & Quotient: 2 Bytes; Divisor & Remainder: 1 Byte)

```
                            *       PD760041
                            ************************************************************
                            *     SIGNED DIVISION
                            ************************************************************
                            *
                            * NEGATIVE DIVIDEND AND OR DIVISOR ARE COMPLEMENTED
                            * PRIOR TO EXECUTION OF DIVISION
                            *
                            * SIGNS ARE CODED IN STATUS:
                            *   STATUS CODING:DVDN DVSR STAT QUOT RMDR
                            *                  +    +   00    +    +
                            *                  +    -   40    -    +
                            *                  -    +   80    -    -
                            *                  -    -   C0    +    -
                            * DIVIDEND MUST BE UNEQUAL H'8000' (NO CORRECT OVF)
                            * NEGATIVE SIGN OF DIVISOR IS LOST AFTER EXECUTION.
052D 052D 77 0D             DIVS    PPSL     WC+OVF+C     ARITH ROTATE WITH CARRY ETC
052F      20                        EORZ     R0
0530      C1                        STRZ     R1           CLEAR R1
0531      0E 06 02                  LODA,R2  DVSR         FETCH DIVISOR IN R2
0534      14                        RETC,Z                RETURN WITH OVF SET IF DVSR= 0
0535      19 06                     BCTR,P   DIV0         BRANCH IF DIVISOR >0
0537      A2                        SUBZ     R2           TAKE 2S COMPLEMENT OF DVSR
0538      CC 06 02                  STRA,R0  DVSR         RESTORE DIVISOR
053B      05 40                     LODI,R1  H'40'        LOAD STATUS IN R1
053D 053D 0E 06 00          DIV0    LODA,R2  DVDN         FETCH MS BYTE OF DIVIDEND
0540      9A 04                     BCFR,N   DIV1         BRANCH IF DIVIDEND NOT<0
0542      3B 18                     BSTR,UN  CMPL         TAKE 2S COMPLEMENT OF DIVIDEND
0544      85 80                     ADDI,R1  H'80'        UPDATE STATUS
0546 0546 CD 06 03          DIV1    STRA,R1  STAT         SAVE STATUS
0549      3F 05 06                  BSTA,UN  DIVU         CALL UNSIGNED DIVISION
054C      0F 06 03                  LODA,R3  STAT         LOAD STATUS IN R3
054F      14                        RETC,Z                RETURN IF BOTH DVDN AND DVSR NOT<0
0550      19 07                     BCTR,P   DIV2         BRANCH IF DVDN WAS NOT <0 AND DVSR<0
0552      77 01                     PPSL     C            CLEAR BORROW
0554      20                        EORZ     R0           CLEAR R0
0555      A1                        SUBZ     R1           TAKE 2 S COMPLEMENT OF REMAINDER
0556      C1                        STRZ     R1           RESTORE REMAINDER IN R1
0557      D3                        RRL,R3                SHIFT R3 LEFT
0558      16                        RETC,N                RETURN IF BOTH DVDN,DVSR<0
0559 0559 3B 01             DIV2    BSTR,UN  CMPL         TAKES 2S COMPL. OF QUOTIENT
055B      17                        RETC,UN               RETURN TO MAINPROGRAM
                            *
                            *
                            *  SUBROUTINE TO TAKE 2S COMPL
                            *    OF (DVDN,DVDN+1)
                            *
055C 055C 77 01             CMPL    PPSL     C            CLEAR BORROW
055E      07 02                     LODI,R3  2            LOAD INDEX REG
0560 0560 20                CMP0    EORZ     R0           CLR R0
0561      AF 46 00                  SUBA,R0  DVDN,R3,-    COMPLEMENT BYTE
0564      CF 66 00                  STRA,R0  DVDN,R3      RESTORE RESULT
0567      5B 77                     BRNR,R3  CMP0         BRANCH IF NOT DONE
0569      17                        RETC,UN
                                    ORG      H'600'
     0600                   DVDN    RES      2            DIVIDEND AND QUOTIENT
     0602                   DVSR    RES      1            DIVISOR
     0603                   STAT    RES      1            STATUS REG
                                END
```

FIGURE 6-4

Signetics 2650 Microprocessor application memos currently available:

AS50	Serial Input/Output
AS51	Bit and Byte Testing Procedures
AS52	General Delay Routines
AS54	Conversion Routines
SP50	2650 Evaluation Printed Circuit Board Level System (PC1001)
SP51	2650 Demo Systems
SP52	Support Software for use with the NCSS Timesharing System
SP53	Simulator, Version 1.2
SP54	Support Software for use with the General Electric Mark III Timesharing System
SS50	PIPBUG
SS51	Absolute Object Format (Revision 1)
MP51	2650 Initialization
MP52	Low Cost Clock Generator Circuits

9399 509 53861

signetics

MOS
MICROPROCESSOR

CONVERSION ROUTINES. AS54

signetics

CONVERSION ROUTINES AS54

2650 MICROPROCESSOR

APPLICATIONS MEMO

INTRODUCTION

Conversion routines like binary to BCD, BCD to binary, and BCD to ASCII are often used in microprocessor based systems. This applications memo describes routines for converting:

- Eight-bit unsigned binary to BCD.
- Sixteen-bit signed binary to BCD.
- Signed BCD to binary conversion 1 (using an addition method).
- Signed BCD to binary conversion 2 (using a multiplication method).
- Signed BCD to ASCII
- ASCII to BCD
- Hexadecimal to ASCII
- ASCII to Hexadecimal

1. EIGHT-BIT UNSIGNED BINARY-TO-BCD CONVERSION

FUNCTION:

Converts an unsigned binary number to a BCD number (3 digits).

(BINN) $\xrightarrow{\text{Conversion}}$ R0, R1

A multiplication method is used.

PARAMETERS:

Input:	BINN contains the binary number (8 bits unsigned.
Output:	Registers R0, R1 contain the BCD result (3 BCD digits). R0 is the most-significant byte. The maximum BCD result is 256 decimal.

Refer to figures 1.1 and 1.2 for flowchart and program listing.

	HARDWARE AFFECTED						
REGISTERS	R0 X	R1 X	R2	R3	R1′	R2′	R3′
PSU	F	II	SP				
PSL	CC X	IDC X	RS	WC X	OVF X	COM X	C X

RAM REQUIRED (BYTES):	1
ROM REQUIRED (BYTES):	28
EXECUTION TIME:	Variable
MAXIMUM SUBROUTINE NESTING LEVELS:	0
ASSEMBLER/COMPILER USED:	PIPHASM

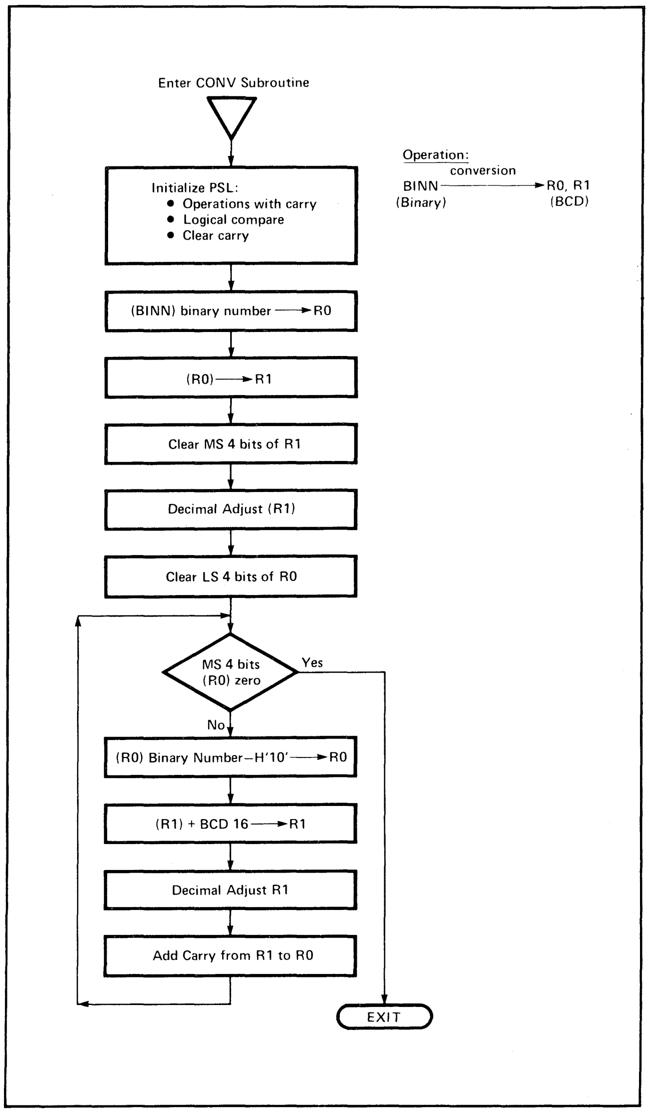


FIGURE 1-1 Flowchart for Eight-Bit Unsigned Binary-to-BCD Conversion (Multiplication Method)

```
                       *    PD760050
                       *********************************************
                       *   8 BIT UNSIGNED BINARY TO BCD CONVERSION
                       *********************************************
                       *
                       *THIS ROUTINE CONVERTS AN 8 BIT UNSIGNED BINARY
                       *NUMBER INTO AN UNSIGNED BCD NUMBER.
                       *
                       *BINARY NUMBER IS IN BINN.
                       *BCD NUMBER (AFTER CONVERSION) IS IN R0,R1.
                       *   HUNDREDS IN R0
                       *   TENS,UNITS IN R1.
                       *
                       *DEFINITIONS OF SYMBOLS:
                       *
      0000             R0   EQU    0         PROCESSOR-REGISTERS
      0001             R1   EQU    1
      0008             WC   EQU    H'08'     PSL: 1=WITH, 0=WITHOUT CARRY
      0002             COM  EQU    H'02'          1=LOGIC, 0=ARITH.COMPARE
      0001             C    EQU    H'01'          CARRY:BORROW
      0003             UN   EQU    3         BRANCH COND.: UNCONDITIONAL
      0002             LT   EQU    2                       LESS THAN
                       *
                       *
                            ORG    H'600'
                       *
      0600             BINN RES    1         BINARY NUMBER.
                       *
                            ORG    H'500'    START ADDRESS OF ROUTINE.
                       *
                       *                     INITIALISATION:
0500  0500  77 0A      CONV PPSL   WC+COM    WITH CARRY,LOGICAL COMPARE
0502        75 01           CPSL   C         CLEAR CARRY FLAG IN PSL.
0504        0C 06 00        LODA,R0 BINN     8 BIT BIN.NUMBER -> R0.
0507        C1              STRZ   R1        (R0) -> R1.
0508        45 0F           ANDI,R1 H'0F'    CLEAR MS 4 BITS BIN. NUMBER
050A        85 66           ADDI,R1 H'66'    PREPARE R1 FOR DECIMAL ADJUST.
050C        95              DAR,R1
                       *
050D        44 F0           ANDI,R0 H'F0'    CLEAR LS 4 BITS.
                       *
050F  050F  E4 10      LOOP COMI,R0 H'10'
0511        1A 09           BCTR,LT EXIT    IF MS 4 BITS ZERO THEN RETRUN.
0513        A4 0F           SUBI,R0 H'10'-1  SUBTRACT 1 FROM MS 4 BITS
0515        85 7B           ADDI,R1 H'16'+H'66'-1 ADD BCD 16 AND PREPARE
0517        95              DAR,R1           FOR DECIMAL ADJUST.
0518        84 00           ADDI,R0 0       ADD CARRY TO MS BCD DIGIT
051A        1B 73           BCTR,UN LOOP     BRANCH AGAIN
                       *
051C  051C  40         EXIT HALT             END OF CONVERSION.
                            END
```

FIGURE 1-2 Program Listing for Eight-Bit Unsigned Binary-to-BCD Conversion

2. SIXTEEN-BIT SIGNED BINARY-TO-BCD CONVERSION

FUNCTION:

Converts a signed 16-bit binary number to a signed BCD number.
Subtraction of base numbers is used.

PARAMETERS:

Input:	BINN, BINN+1 contain the signed binary number. BINN is the most-significant byte. Binary number is destroyed after conversion.
Output:	BCDD, BCDD+1, BCDD+2 contain the BCD result. BCDD contains the sign and the most-significant BCD digit. The minimum BCD result is –32768 decimal. The maximum BCD result is +32767 decimal.

Refer to figures 2.1 and 2.2 for flowchart and program listing.

	HARDWARE AFFECTED						
REGISTERS	R0 X	R1 X	R2 X	R3 X	R1′	R2′	R3′
PSU	F	II	SP				
PSL	CC X	IDC X	RS	WC X	OVF X	COM	C X

RAM REQUIRED (BYTES): 5

ROM REQUIRED (BYTES): 106

EXECUTION TIME: Variable

MAXIMUM SUBROUTINE NESTING LEVELS: 0

ASSEMBLER/COMPILER USED: PIPHASM

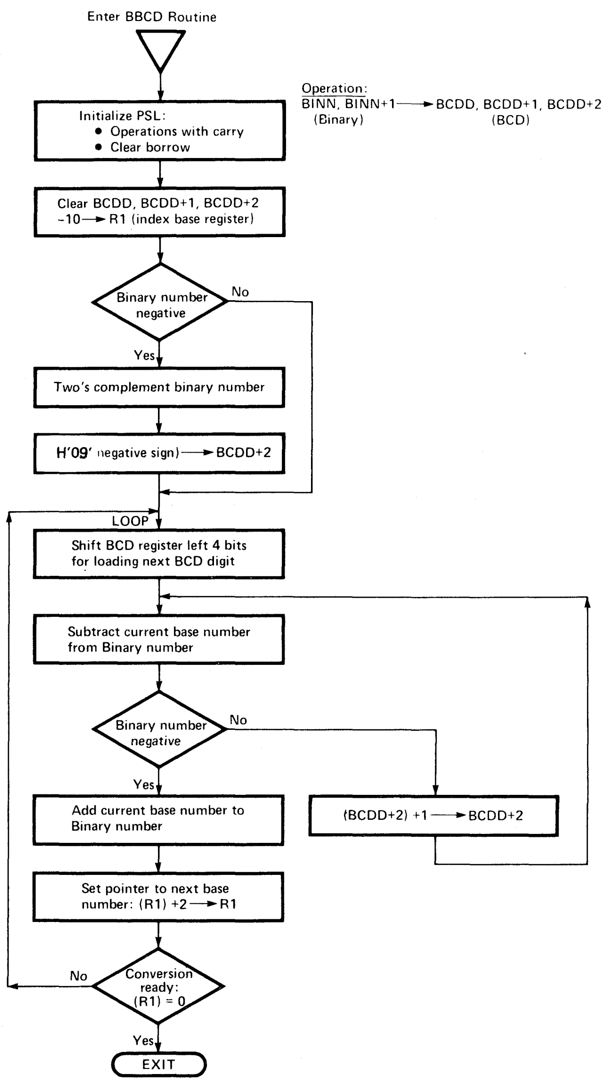

Enter BBCD Routine
Operation:
BINN, BINN+1 → BCDD, BCDD+1, BCDD+2
(Binary) (BCD)
Initialize PSL:
• Operations with carry
• Clear borrow
Clear BCDD, BCDD+1, BCDD+2
-10 → R1 (index base register)
Binary number negative
No
Yes
Two's complement binary number
H'09' (negative sign) → BCDD+2
LOOP
Shift BCD register left 4 bits for loading next BCD digit
Subtract current base number from Binary number
Binary number negative
No
Yes
Add current base number to Binary number
(BCDD+2) +1 → BCDD+2
Set pointer to next base number: (R1) +2 → R1
No
Conversion ready: (R1) = 0
Yes
EXIT

FIGURE 2-1 Flowchart for Signed Binary-to-BCD Conversion

```
                        *    PD76ØØ51
                        ++++++++++++++++++++++++++++++++++++++++++++++++
                        *      BINARY TO BCD CONVERSION                +
                        ++++++++++++++++++++++++++++++++++++++++++++++++
                        *
                        *THIS ROUTINE CONVERTS A SIGNED BINARY NUMBER
                        *(16 BITS) INTO A SIGNED BCD NUMBER
                        *(24 BITS: SIGN + 5 BCD DIGITS).
                        *
                        *THE BINARY NUMBER IS IN BINN,BINN+1.
                        *THE BCD NUMBER IS IN BCDD,BCDD+1,BCDD+2.
                        *BINN AND BCDD ARE MOST SIGNIFICANT BYTES.
                        *MS NIBBLE OF BCDD=Ø FOR POSITIVE BINARY NUMBERS.
                        *MS NIBBLE OF BCDD=9 FOR NEGATIVE BINARY NUMBERS.
                        *
                        *SUBTRAHENDS ARE PLACED IN REGISTER BASE (1Ø BYTES)
                        *
                        *DEFINITION OF SYMBOLS:
                        *
      ØØØØ              RØ   EQU    Ø          PROCESSOR-REGISTERS
      ØØØ1              R1   EQU    1
      ØØØ2              R2   EQU    2
      ØØØ3              R3   EQU    3
      ØØØ8              WC   EQU    H'Ø8'      PSL: 1=WITH, Ø=WITHOUT CARRY
      ØØØ1              C    EQU    H'Ø1'          CARRY+BORROW
      ØØØ2              N    EQU    2          BRANCH COND.: NEGATIVE
      ØØØ3              UN   EQU    3                         UNCONDITIONALY
                        *
                        *
                             ORG    H'6ØØ'     START ADDRESS
                        *
      Ø6ØØ              BINN RES    2          BINARY NUMBER MEMORY LOCATION
      Ø6Ø2              BCDD RES    3          BCD REGISTER
Ø6Ø5  Ø6Ø5  27 1Ø       BASE DATA   H'27,1Ø'   1ØØØØ
Ø6Ø7        Ø3 E8            DATA   H'Ø3,E8'   1ØØØ
Ø6Ø9        ØØ 64            DATA   H'ØØ,64'   1ØØ
Ø6ØB        ØØ ØA            DATA   H'ØØ,ØA'   1Ø
Ø6ØD        ØØ Ø1            DATA   H'ØØ,Ø1'   1
      ØØØA              LEN  EQU    $-BASE     LENGTH BASE REGISTER
                             ORG    H'5ØØ'     START ADDR. OF PROGRAM
                        *
Ø5ØØ  Ø5ØØ  77 Ø9       BBCD PPSL   WC+C       ARITHMETIC+ROTATE WITH CARRY:
                        *                      CLEAR BORROW.
Ø5Ø2        2Ø               EORZ   RØ         INITIALISATION: CLEAR RØ.
Ø5Ø3        Ø7 Ø3            LODI,R3 3
Ø5Ø5  Ø5Ø5  CF 46 Ø2    LOCØ STRA,RØ BCDD,R3,- CLEAR 3 BYTES OF BCD REGISTER.
Ø5Ø8        5B 7B            BRNR,R3 LOCØ
Ø5ØA        Ø5 F6            LODI,R1 -LEN      LENGTH OF BASE REGISTER.
                        *
Ø5ØC        ØE Ø6 ØØ         LODA,R2 BINN      MS 4 BITS BINARY NUMBER.
Ø5ØF        9A 1Ø            BCFR,N  LOOP      IF POS. GO TO LOOP
Ø511  Ø511  Ø6 Ø2       COMP LODI,R2 2         LOAD INDEX REGISTER.
Ø513  Ø513  2Ø          LOC1 EORZ    RØ        TWO'S COMPLEMENT BY
Ø514        AE 46 ØØ         SUBA,RØ BINN,R2,- SUBTRACTING FROM ZERO.
Ø517        CE 66 ØØ         STRA,RØ BINN,R2
Ø51A        5A 77            BRNR,R2 LOC1      RETURN IF NOT READY.
Ø51C        Ø4 Ø9            LODI,RØ H'Ø9'     NEGATIVE SIGN INDICATION.
Ø51E        CC Ø6 Ø4         STRA,RØ BCDD+2    SIGN IN LSB OF BCD REGISTER.
                        *                      SHIFT BCD REG. LEFT 4 TIMES.
Ø521  Ø521  75 Ø1       LOOP CPSL    C         CLEAR CARRY FOR ROTATE.
Ø523        Ø6 Ø4            LODI,R2 4         BIT COUNT.
Ø525  Ø525  Ø7 Ø3       LP2  LODI,R3 3         INDEX BYTE SHIFT.
Ø527  Ø527  ØF 46 Ø2    LP1  LODA,RØ BCDD,R3,- BCD DIGIT INTO RØ.
Ø52A        DØ               RRL,RØ            CARRY (PREVIOUS MS BIT)-> LSB
Ø52B        CF 66 Ø2         STRA,RØ BCDD,R3   AND MS BIT -> CARRY.
Ø52E        5B 77            BRNR,R3 LP1
Ø53Ø        FA 73            BDRR,R2 LP2
                        *
Ø532  Ø532  85 Ø2       SUBL ADDI,R1 2         RESTORE BASE INDEX.
Ø534        Ø6 Ø2            LODI,R2 2         INDEX REGISTER
Ø536        77 Ø1            PPSL    C         CLEAR BORROW
Ø538  Ø538  ØE 46 ØØ    LOC2 LODA,RØ BINN,R2,- LOAD BINN AND SUBTRACT
Ø53B        AD 45 ØF         SUBA,RØ BASE-256+LEN,R1,- CORRESPONDING
Ø53E        CE 66 ØØ         STRA,RØ BINN,R2   BASE DIGIT
Ø541        5A 75            BRNR,R2 LOC2
Ø543        1A Ø9            BCTR,N  CORR      IF BINN NEG. THEN CORRECTION.
Ø545        ØC Ø6 Ø4         LODA,RØ BCDD+2
Ø548        82               ADDZ    R2        ADD 1 TO LSB OF BCD NUMBER
Ø549        CC Ø6 Ø4         STRA,RØ BCDD+2    C=1 IN PSL AND (R2)=Ø
Ø54C        1B 64            BCTR,UN SUBL
                        *
Ø54E  Ø54E  Ø6 Ø2       CORR LODI,R2 2         INDEX COUNT
Ø55Ø  Ø55Ø  ØE 46 ØØ    LOC3 LODA,RØ BINN,R2,- ADD CORRESPONDING BASE BYTE TO
Ø553        8D 45 11         ADDA,RØ BASE-256+LEN+2,R1,- BINARY NUMBER.
Ø556        CE 66 ØØ         STRA,RØ BINN,R2
Ø559        5A 75            BRNR,R2 LOC3      RETURN IF NOT READTY
Ø55B        85 Ø3            ADDI,R1 3         UPDATE BASE POINTER:C=1IN PSL
Ø55D        59 42            BRNR,R1 LOOP      RETURN IF CONVERSION NOT READY
Ø55F  Ø55F  4Ø          EXIT HALT    END OF CONVERSION
                             END
```

FIGURE 2-2 Program Listing for Signed Binary-to-BCD Conversion

3. SIGNED BCD-TO-BINARY CONVERSION 1

FUNCTION:

Converts a five-digit signed BCD number to a sixteen-bit signed binary number.

Addition of base numbers is used.

PARAMETERS:

Input:	BCDD, BCDD+1, BCDD+2 contain the BCD number. BCDD contains the sign plus the most-significant BCD digit. The range of BCD numbers is: −32768<BCD Number<+32767. BCDD is destroyed after the conversion.
Output:	BINN, BINN+1 contain the signed binary number. BINN is the most-significant byte.

Refer to figures 3.1 and 3.2 for flowchart and program listing.

	HARDWARE AFFECTED						
REGISTERS	R0 X	R1 X	R2 X	R3 X	R1′	R2′	R3′
PSU	F	II	SP				
PSL	CC X	IDC X	RS	WC X	OVF X	COM	C X

RAM REQUIRED (BYTES): 5

ROM REQUIRED (BYTES): 86

EXECUTION TIME: Variable

MAXIMUM SUBROUTINE NESTING LEVELS: 0

ASSEMBLER/COMPILER USED: PIPHASM

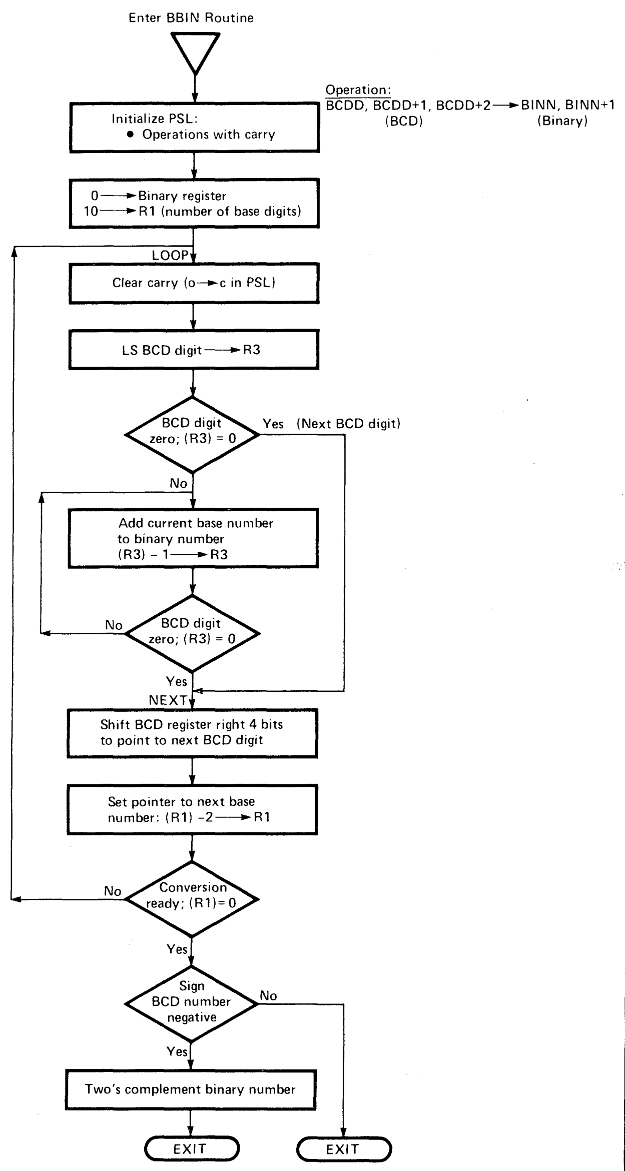

Enter BBIN Routine
Operation:
BCDD, BCDD+1, BCDD+2 ⟶ BINN, BINN+1
(BCD) (Binary)
Initialize PSL:
• Operations with carry
0 ⟶ Binary register
10 ⟶ R1 (number of base digits)
LOOP
Clear carry (o ⟶ c in PSL)
LS BCD digit ⟶ R3
BCD digit zero; (R3) = 0
Yes (Next BCD digit)
No
Add current base number to binary number
(R3) − 1 ⟶ R3
No
BCD digit zero; (R3) = 0
Yes
NEXT
Shift BCD register right 4 bits to point to next BCD digit
Set pointer to next base number: (R1) −2 ⟶ R1
No
Conversion ready; (R1) = 0
Yes
Sign BCD number negative
No
Yes
Two's complement binary number
EXIT
EXIT

FIGURE 3-1: Flowchart for signed BCD-to-Binary Conversion

```
                            * PD760052
                            *********************************************
                            * BCD TO BINARY CONVERSION
                            *********************************************
                            *
                            * THIS ROUTINE CONVERTS A SIGNED BCD NUMBER
                            * (24 BITS: SIGN+5 BCD DIGITS) INTO A SIGNED
                            * BINARY NUMBER (16 BITS).
                            * -32768 <BCD NUMBER <+32767
                            *   BCD NUMBER IS LOST AFTER CONVERSION.
                            *
                            * THE BINARY NUMBER IS IN BINN,BINN+1.
                            * THE BCD NUMBER IS IN BCDD,BCDD+1,BCDD+2 (A0-A4).
                            * THE BASE NUMBERS ARE IN BASE,- -,BASE+9 (R0,R4).
                            * BINN AND BCDD ARE MOST SIGNIFICANT BYTES.
                            *
                            * PRINCIPLE OF CONVERSION IS:
                            * BINN = A0.R0+ A1.R1+ A2.R2+ A3.R3+ A4.R4
                            *   A0 -A4 = NUMBER OF DIGITS OF BCD NUMBER.
                            *   R0 -R4 = BASE NUMBERS FOR CONVERSION.
                            *
                            * DEFINITIONS OF SYMBOLS:
      0000                  R0   EQU  0        PROCESSOR-REGISTERS
      0001                  R1   EQU  1
      0002                  R2   EQU  2
      0003                  R3   EQU  3
      0008                  WC   EQU  H'08'    PSL: 1=WITH, 0=WITHOUT CARRY
      0001                  C    EQU  H'01'         CARRY:BORROW
      0000                  Z    EQU  0        BRANCH COND: ZERO
      0000                  ON    EQU  0                      ALL BITS ARE 1
      0008                  SIGN EQU  H'08'    TO TEST BCD. NUMBER
      000A                  LEN  EQU  10       INDEX NUMBER (LENGTH BASE REG)
                            *
                              ORG     H'600'
                            *
      0600                  BINN RES  2        BINARY NUMBER
      0602                  BCDD RES  3        BCD NUMBER
0605  0605  27 10           BASE DATA H'27,10'  10000
0607        03 E8                DATA H'03,E8'  1000
0609        00 64                DATA H'00,64'  100
060B        00 0A                DATA H'00,0A'  10
060D        00 01                DATA H'00,01'  1
                                 ORG     H'450'   START OF PROGRAM
0450  0450  77 08           BBIN PPSL    WC       ARITHMETIC+ROTATE WITH CARRY
0452        20                   EORZ    R0       CLEAR R0
0453        CC 06 00             STRA,R0 BINN     CLEAR BINARY REGISTERS
0456        CC 06 01             STRA,R0 BINN+1
0459        05 0A                LODI,R1 LEN      INDEX FOR BASE DIGITS
045B  045B  75 01           LOOP CPSL    C        CLEAR CARRY
045D        0F 06 04             LODA,R3 BCDD+2   LOAD LS BCD DIGIT IN R3
0460        47 0F                ANDI,R3 H'0F'    CLEAR MS 4 BITS
0462        18 11                BCTR,Z  NEXT     IF ZERO GO TO NEXT
0464  0464  06 02           LOC1 LODI,R2 2        LOAD INDEX
0466  0466  0E 46 00        LOC2 LODA,R0 BINN,R2,-
0469        8D 46 05             ADDA,R0 BASE,R1,-  ADD BASE DIGIT TO BIN. NUMBER
046C        CE 66 00             STRA,R0 BINN,R2
046F        5A 75                BRNR,R2 LOC2
0471        85 02                ADDI,R1 2        RESTORE BASE POINTER
0473        FB 6F                BDRR,R3 LOC1     IF NOT READY RETURN TO LOC1
                            *
0475  0475  06 04           NEXT LODI,R2 4        BIT COUNT
0477  0477  07 FD           LP2  LODI,R3 -3       INDEX FOR BYTE COUNT
0479  0479  0F 65 05        LP1  LODA,R0 BCDD-256+3,R3 BCD DIGIT INTO R0
047C        50                   RRR,R0           CARRY (PREVIOUS LS BIT) -> MSB
047D        CF 65 05             STRA,R0 BCDD-256+3,R3    AND LS BIT -> CARRY.
0480        DB 77                BIRR,R3 LP1      NEXT BCDD BYTE
0482        75 01                CPSL    C
0484        FA 71                BDRR,R2 LP2      NEXT SHIFT OF BCD REG. BIT
0486        F9 00                BDRR,R1 $+2      UPDATE BASE POINTER WITHOUT
0488        F9 51                BDRR,R1 LOOP       AFFECTING C FLAG IN PSL AND
                            *                               GO TO LOOP IF NOT READY
048A        F4 08                TMI,R0  SIGN
048C        98 0D                BCFR,ON EXIT     IF SIGN POS. THEN READY.
048E  048E  77 01           COMP PPSL    C        CLEAR BORROW
0490        06 02                LODI,R2 2        NUMBER OF DIGITS
0492  0492  20              LP3  EORZ    R0       TWO'S COMPLEMENT BY
0493        AE 46 00             SUBA,R0 BINN,R2,- SUBTRACTION FROM ZERO
0496        CE 66 00             STRA,R0 BINN,R2
0499        5A 77                BRNR,R2 LP3
                            *
049B  049B  40              EXIT HALT             END OF CONVERSION
                                 END
```

FIGURE 3-2 Program Listing for Signed BCD-to-Binary Conversion

4. SIGNED BCD-TO-BINARY CONVERSION 2

FUNCTION:

Converts a five-digit signed BCD number to a sixteen-bit signed binary number.

A multiplication method is used.

PARAMETERS:

Input:	BCDD, BCDD+1 contain the BCD number. BCDD contains the sign plus the most-significant BCD digit. The range of BCD numbers is: −32768<BCD Number<+32767
Output:	BINN, BINN+1 contain the signed binary number. BINN is the most-significant byte.

Refer to figures 4.1 and 4.2 for flowchart and program listing.

	HARDWARE AFFECTED						
REGISTERS	R0 X	R1 X	R2 X	R3 X	R1'	R2'	R3'
PSU	F	II	SP				
PSL	CC X	IDC X	RS	WC X	OVF X	COM	C X

RAM REQUIRED (BYTES):	6
ROM REQUIRED (BYTES):	87
EXECUTION TIME:	Variable
MAXIMUM SUBROUTINE NESTING LEVELS:	0
ASSEMBLER/COMPILER USED:	PIPHASM

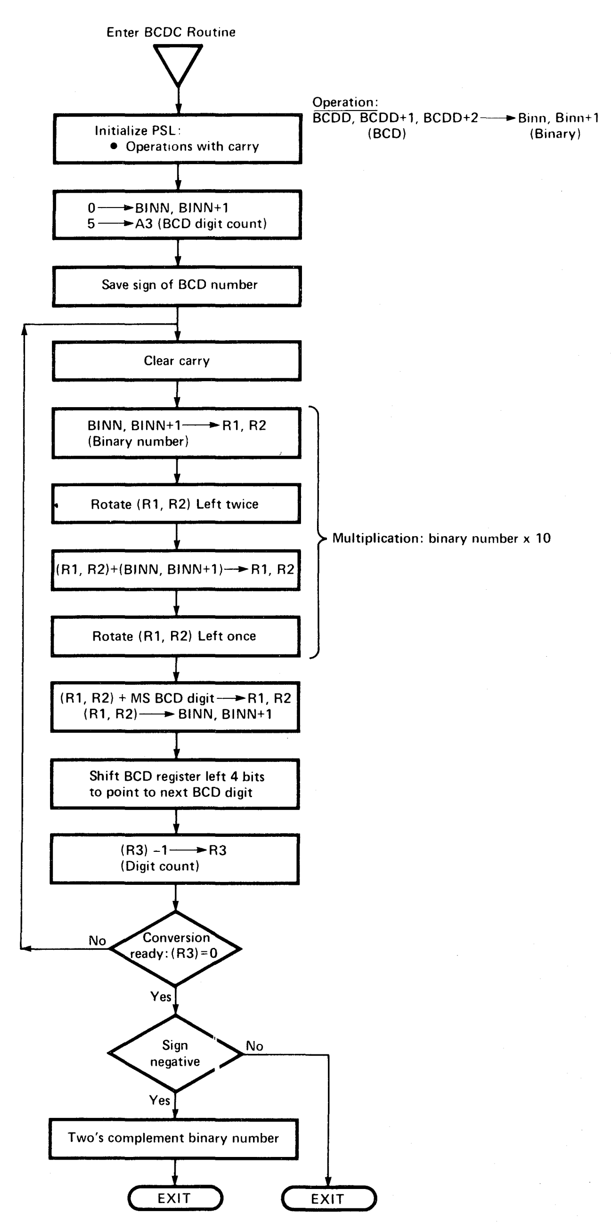

Enter BCDC Routine
Initialize PSL:
• Operations with carry
Operation:
BCDD, BCDD+1, BCDD+2 → Binn, Binn+1
(BCD) (Binary)
0 → BINN, BINN+1
5 → A3 (BCD digit count)
Save sign of BCD number
Clear carry
BINN, BINN+1 → R1, R2
(Binary number)
Rotate (R1, R2) Left twice
(R1, R2)+(BINN, BINN+1) → R1, R2
Rotate (R1, R2) Left once
Multiplication: binary number x 10
(R1, R2) + MS BCD digit → R1, R2
(R1, R2) → BINN, BINN+1
Shift BCD register left 4 bits
to point to next BCD digit
(R3) −1 → R3
(Digit count)
No
Conversion ready: (R3) = 0
Yes
Sign negative
No
Yes
Two's complement binary number
EXIT
EXIT

FIGURE 4-1: Flowchart for signed BCD-to-Binary Conversion (Multiplication Method).

```
                              *   PD760053
                              *****************************************
                              *  BCD TO BINARY CONVERSION              *
                              *****************************************
                              *
                              * THIS ROUTINE CONVERTS A SIGNED BCD NUMBER
                              * (24 BITS: SIGN  + 5 BCD DIGITS) INTO A SIGNED
                              * BINARY NUMBER (16 BITS).
                              *   -32768 < BCD NUMBER < +32767
                              *     BCD NUMBER IS LOST AFTER CONVERSION
                              *
                              * PRINCIPLE:
                              * BIN.=(((((((A*10) +B) *10) +C) *10) +D) *10) +E
                              *  ABCDE= BCD NUMBER
                              *
                              * MULTIPLICATION BY 10 IS DONE BY:
                              *  LOAD R2,R1 WITH BIN. NUMBER, SHIFT LEFT TWICE,
                              *   ADD BIN. NUMBER TO R2,R1, SHIFT LEFT ONCE,
                              *    STORE R2,R1 IN BINN,BINN+1 AS RESULT
                              *
                              * DEFINITION OF SYMBOLS:
      0000                    R0   EQU    0         PROCESSOR-REGISTERS
      0001                    R1   EQU    1
      0002                    R2   EQU    2
      0003                    R3   EQU    3
      0008                    WC   EQU    H'08'     PSL: 1=WITH, 0=WITHOUT CARRY
      0001                    C    EQU    H'01'          CARRY:BORROW
      0002                    N    EQU    2         COND: NEGATIVE
      0005                    NUM  EQU    5         INDEX FOR NUMBER OF BCD DIGITS
                              *
                                   ORG    H'600'
                              *
      0600                    BINN RES    2         BINARY NUMBER
      0602                    BCDD RES    3         BCD NUMBER AND SIGN
      0605                    SIGN RES    1         SAVE SIGN DIGIT
                              *
                              *
                                   ORG    H'450'    START OF PROGRAM
                              *
0450  0450  77 08             BCDC PPSL   WC        ARITH.:ROTATE WITH CARRY
0452        20                     EORZ   R0        CLEAR R0
0453        CC 06 00               STRA,R0 BINN     CLEAR BINARY NUMBERS
0456        CC 06 01               STRA,R0 BINN+1
0459        07 05                  LODI,R3 NUM      BCD INDEX REGISTER
                              *
045B        0C 06 02               LODA,R0 BCDD     SAVE SIGN OF BCD NUMBER IN
045E        CC 06 05               STRA,R0 SIGN     MEMORY LOC. SIGN
                              *
                              *                     MULTIPLY BINARY NUMBER BY 10
0461  0461  75 01             LOOP CPSL   C         CLEAR CARRY
0463        0D 06 00               LODA,R1 BINN     LOAD BIN. NUMBER IN R1,R2
0466        0E 06 01               LODA,R2 BINN+1
0469        D2                     RRL,R2           ROTATE REGISTERS R1,R2 LEFT 2
046A        D1                     RRL,R1
046B        D2                     RRL,R2
046C        D1                     RRL,R1
046D        8E 06 01               ADDA,R2 BINN+1   ADD BIN. NUMBER TO R1,R2
0470        8D 06 00               ADDA,R1 BINN
0473        D2                     RRL,R2           SHIFT R1,R2 LEFT ONCE
0474        D1                     RRL,R1
                              *
                              *
0475        0C 06 02               LODA,R0 BCDD     LOAD MS BCD DIGIT IN R0
0478        44 0F                  ANDI,R0 H'0F'    CLEAR MS 4 BITS
047A        82                     ADDZ   R2        ADD BCD TO BINARY NUMBER
047B        85 00                  ADDI,R1 0        ADD CARRY TO MS BYTE
047D        CD 06 00               STRA,R1 BINN     STORE RESULT IN BINN,BINN+1
0480        CC 06 01               STRA,R0 BINN+1
                              *                     ROTATE BCD NUMBER 4 TIMES LEFT
                              *                      TO POINT TO NEXT BCD DIGIT
0483        05 04                  LODI,R1 4        BIT COUNT
0485  0485  06 03             LP2  LODI,R2 3        INDEX FOR BYTE COUNT
0487  0487  0E 46 02          LP1  LODA,R0 BCDD,R2,-
048A        D0                     RRL,R0           SHIFT BCD BYTE LEFT
048B        CE 66 02               STRA,R0 BCDD,R2
048E        5A 77                  BRNR,R2 LP1      NEXT BYTE OF BCD REGISTER
0490        F9 73                  BDRR,R1 LP2      NEXT BIT SHIFT
0492        FB 4D                  BDRR,R3 LOOP     TO LOOP IF MULTIPLY NOT READY
                              *
0494        0C 06 05               LODA,R0 SIGN
0497        9A 0D                  BCFR,N  EXIT     IF SIGN POS. THEN READY
0499        77 01                  PPSL   C         CLEAR CARRY
049B        06 02                  LODI,R2 2        INDEX LOADING
049D  049D  20                LP3  EORZ   R0        TWO'S COMPLEMENT BY
049E        AE 66 00               SUBA,R0 BINN,R2  SUBTRACTING FROM
04A1        CC 06 00               STRA,R0 BINN
04A4        5A 77                  BRNR,R2 LP3
                              *
04A6  04A6  40                EXIT HALT   END OF CONVERSION
                                   END
```

FIGURE 4-2 Program Listing for Signed BCD-to-Binary Conversion

5. SIGNED BCD-TO-ASCII CONVERSION

FUNCTION:

Converts n BCD digits plus sign to n + 1 ASCII characters (sign included).

PARAMETERS:

Input:	BCDD, BCDD+1, ------ BCDD+ (numb – 1) BCDD contains the sign plus the most-significant digit (2 BCD digits/byte). Numb is the number of BCD bytes.
Output:	ASCI, ASCI+1, ------, ASCI + (num – 1) contains the signed result. ASCI contains the sign. ASCI+1 contains the most-significant byte.

Refer to figures 5.1 and 5.2 for flowchart and program listing.

	HARDWARE AFFECTED						
REGISTERS	R0 X	R1 X	R2 X	R3 X	R1′	R2′	R3′
PSU	F	II	SP				
PSL	CC X	IDC X	RS	WC X	OVF X	COM	C X

RAM REQUIRED (BYTES):	N Numb, N Num+1
ROM REQUIRED (BYTES):	53
EXECUTION TIME:	Variable
MAXIMUM SUBROUTINE NESTING LEVELS:	0
ASSEMBLER/COMPILER USED:	PIPHASM

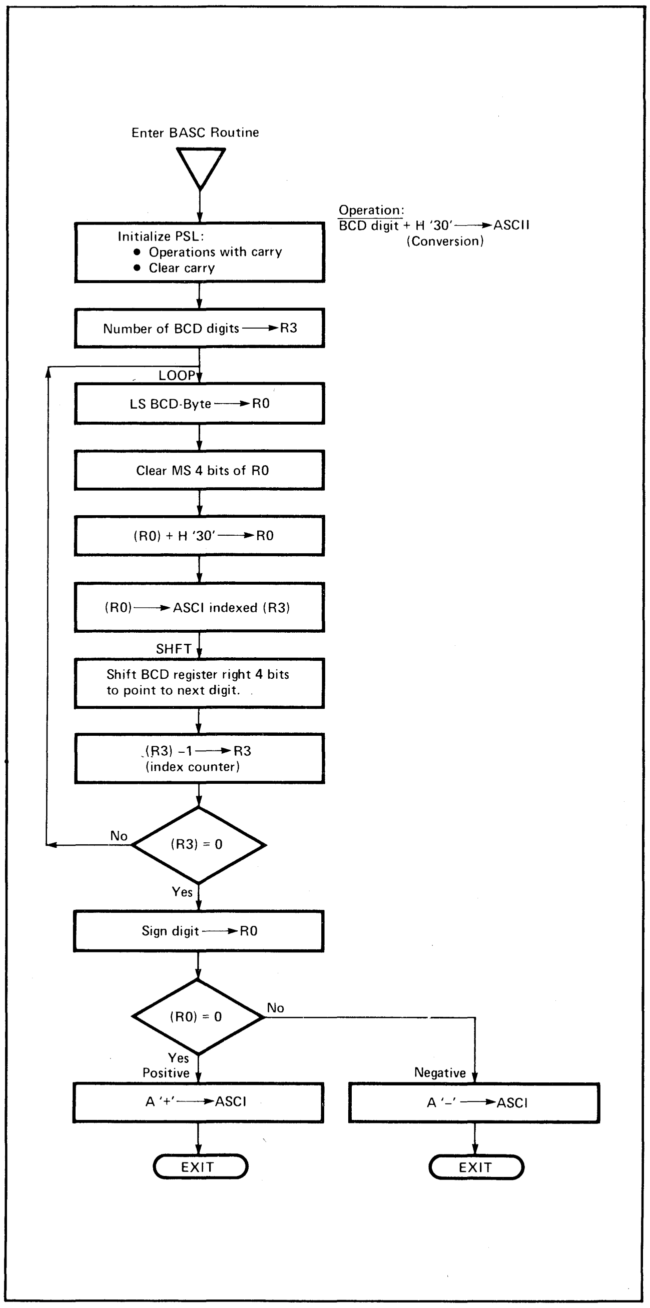

Enter BASC Routine
Operation: BCD digit + H '30' → ASCII (Conversion)
Initialize PSL: • Operations with carry • Clear carry
Number of BCD digits → R3
LOOP
LS BCD-Byte → R0
Clear MS 4 bits of R0
(R0) + H '30' → R0
(R0) → ASCI indexed (R3)
SHFT
Shift BCD register right 4 bits to point to next digit.
(R3) –1 → R3 (index counter)
No
(R3) = 0
Yes
Sign digit → R0
(R0) = 0
No
Yes Positive
Negative
A '+' → ASCI
A '–' → ASCI
EXIT
EXIT

FIGURE 5-1 Flowchart for BCD-to-ASCII Conversion (signed)

```
                              *
                              *   PD760054
                              ++++++++++++++++++++++++++++++++++++++++++++
                              *  BCD TO ASCII CONVERSION                  +
                              ++++++++++++++++++++++++++++++++++++++++++++
                              *
                              * THIS ROUTINE CONVERTS A SIGNED BCD NUMBER
                              * INTO ASCII CHARACTERS (SIGN INCLUDED).
                              * BCD FORMAT: SIGN + BCD DIGITS (TWO DIGITS:BYTES)
                              * THE NUMBER OF BCD DIGITS -> R3 = NUM
                              * THE NUMBER OF BCD BYTES -> R2 = NUMB
                              * BCD NUMBER IS IN BCDD,BCDD+1,---,BCDD+(N-1)
                              * ASCII CHARACTERS ARE IN ASCII,ASCII+1,---,ASCII+NUM
                              *          (SIGN) (BCD DIGITS)
                              *
                              * DEFINITIONS OF SYMBOLS:
                              *
      0000                    R0  EQU     0
      0001                    R1  EQU     1
      0002                    R2  EQU     2
      0003                    R3  EQU     3
      0008                    WC  EQU     H'08'      PSL: 1=WITH, 0=WITHOUT CARRY
      0001                    C   EQU     H'01'            CARRY:BORROW
      0003                    UN  EQU     3          COND: UNCONDITIONAL
      0000                    Z   EQU     0                ZERO
                              *
                              * IN THIS EXAMPLE THE CONVERSION OF 5 BCD DIGITS
                              *  IS PERFORMED.
                              *
      0003                    NUMB  EQU    3          NUMBER OF BCD BYTES
      0005                    NUM  EQU     5          NUMBER OF BCD DIGITS
                              *
                              *
                                  ORG     H'4E0'
                              *
      04E0                    BCDD RES    NUMB       RESERVE FOR BCD NUMBER
      04E3                    ASCI RES    NUM+1      RESERVE FOR SIGN,ASCII DIGITS
                              *
                                  ORG     H'500'     PROGRAM START HERE
                              *
0500  0500  77 08             BASC PPSL   WC         ARITHMETIC:ROTATE WITH CARRY
0502        75 01                 CPSL    C          CLEAR CARRY
0504        07 05                 LODI,R3 NUM        INDEX REGISTER
                              *
0506  0506  0C 04 E2          LOOP LODA,R0 BCDD+NUMB-1 LOAD LS BCD DIGIT IN R0
0509        44 0F                 ANDI,R0 H'0F'      CLEAR MS 4 BITS
050B        84 30                 ADDI,R0 H'30'      ASCII CHARACTER
050D        CF 64 E3              STRA,R0 ASCI,R3    STORE ASCII CHARACTER
                              *
0510  0510  05 04             SHFT LODI,R1 4         BIT COUNT
0512  0512  06 FD             LP2  LODI,R2 -NUMB     INDEX FOR BYTE SHIFT
0514  0514  0E 63 E3          LP1  LODA,R0 BCDD-256+NUMB,R2
0517        50                    RRR,R0             CARRY (PREVIOUS LS BIT) -> MSB
0518        CE 63 E3              STRA,R0 BCDD-256+NUMB,R2   AND LS BIT ->CARRY
051B        DA 77                 BIRR,R2 LP1
051D        75 01                 CPSL    C          CLEAR CARRY
051F        F9 71                 BDRR,R1 LP2
0521        FB 63                 BDRR,R3 LOOP       IF NOT READY GO TO LOOP
                              *
0523  0523  0C 04 E2          SIGN LODA,R0 BCDD+NUMB-1   SIGN -> R0
0526        18 07                 BCTR,Z  POS
0528  0528  04 2D             NEG  LODI,R0 A'-'
052A        CC 04 E3              STRA,R0 ASCI
052D        1B 05                 BCTR,UN EXIT
052F  052F  04 2B             POS  LODI,R0 A'+'
0531        CC 04 E3              STRA,R0 ASCI
                              *
0534  0534  40                EXIT HALT    END OF CONVERSION
                                  END
```

FIGURE 5-2 Program Listing for BCD-to-ASCII Conversion (Signed)

6. ASCII-TO-BCD CONVERSION

FUNCTION:

Converts n ASCII digits to n BCD digits.
ASCII ⟶ BCD

PARAMETERS:

Input:	ADIG, ADIG+1, ----, ADIG+(n – 1) contain ASCII digits. The most-significant digit is in ADIG (byte/digit).
Output:	BCDD, BCDD+1, ----, BCDD + (n–1) contains BCD digits. The most-significant digit is in BCDD (2 digits/byte).

Refer to figures 6.1 and 6.2 for flowchart and program listing.

	HARDWARE AFFECTED						
REGISTERS	R0 X	R1 X	R2 X	R3 X	R1′	R2′	R3′
PSU	F	II	SP				
PSL	CC X	IDC X	RS	WC X	OVF X	COM	C X

RAM REQUIRED (BYTES):	nADIG + nBCDD
ROM REQUIRED (BYTES):	37
EXECUTION TIME:	Variable
MAXIMUM SUBROUTINE NESTING LEVELS:	0
ASSEMBLER/COMPILER USED:	PIPHASM

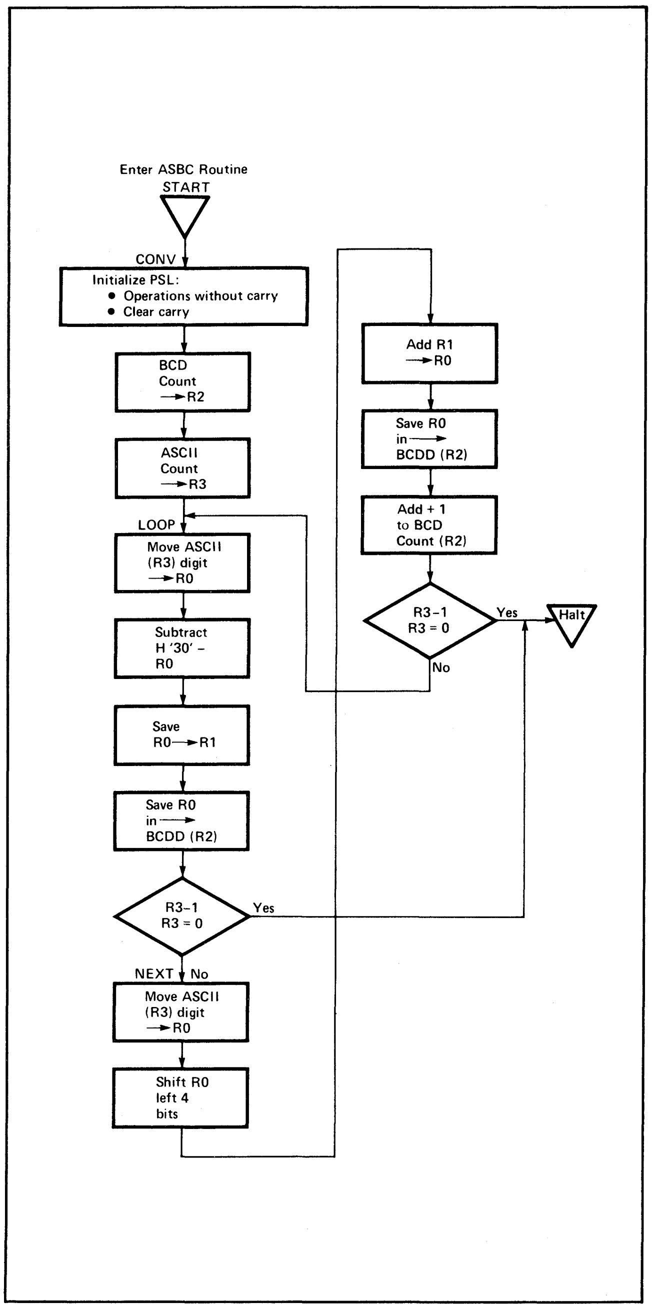


FIGURE 6-1 Flowchart for ASCII-to-BCD Conversion

```
                   *    PD760055
                   *+++++++++++++++++++++++++++++++++++++++++++++
                   *   ASCII TO BCD CONVERSION                   +
                   *+++++++++++++++++++++++++++++++++++++++++++++
                   *
                   * THIS ROUTINE CONVERTS A STRING OF ASCII
                   * DIGITS TO A STRING OF BCD DIGITS.
                   *
                   * ADIG IS MS DIGIT ASCII
                   * BCDD IS MS DIGIT BCD
                   *
                   * DEFINITIONS OF SYMBOLS:
0000               R0   EQU     0         PROCESSOR-REGISTERS
0001               R1   EQU     1
0002               R2   EQU     2
0003               R3   EQU     3
0008               WC   EQU     H'08'     PSL: 1-WITH, 0-WITHOUT
0001               C    EQU     H'01'          CARRY:BORROW
0003               UN   EQU     3         BR.COND: ALWAYS
                   *
                   * IN THIS EXAMPLE THE CONVERSION OF 5
                   * ASCII CHARACTERS IS PERFORMED.
                   *
0005               NUM  EQU     5
0003               NUM1 EQU     3
                   *
                        ORG     H'750'    RAM DEFINITIONS
0750               ADIG RES     NUM       ASCII BYTES RESERVED
0005               ACNT EQU     $-ADIG    ASCII DIGIT COUNT
0755               BCDD RES     NUM1      BCD BYTES RESERVED
0003               BCNT EQU     $-BCDD    BCD BYTE COUNT
                   *
                        ORG     H'500'    START OF SUBROUTINE
0500 75 09         CONV CPSL    WC+C      ARITH.WITHOUT:NO CARRY
0502 06 03              LODI,R2 BCNT      BCD COUNT -> R2
0504 07 05              LODI,R3 ACNT      ASCII COUNT ->R3
0506 0F 67 4F      LOOP LODA,R0 ADIG-1,R3 R0 HAS ASCII DIGIT
0509 A4 30              SUBI,R0 H'30'     MAKE IT BCD
050B C1                 STRZ    R1        R0 ->R1
050C CE 67 54           STRA,R0 BCDD-1,R2 SAVE 1 BCD DIGIT
050F FB 03              BDRR,R3 NEXT      DECREMENT -NON ZERO BR
0511 1F 05 25           BCTA,UN BYE       CONVERSION COMPLETE
0514 0F 67 4F      NEXT LODA,R0 ADIG-1,R3 NEXT ASCII DIGIT
0517 A4 30              SUBI,R0 H'30'     MAKE IT BCD
0519 D0                 RRL,R0            SHIFT LEFT 4 BITS
051A D0                 RRL,R0
051B D0                 RRL,R0
051C D0                 RRL,R0
051D 61                 IORZ    R1        INCLUSIVE OR LOW ORDER
051E CE 67 54           STRA,R0 BCDD-1,R2 STORE 2 BCD DIGITS
0521 FA 00              BDRR,R2 $+2       DECREMENT BCD COUNT
0523 FB 61              BDRR,R3 LOOP      DECREMENT-NON ZERO BR.
0525 40            BYE  HALT              END OF ASCII -> BCD
                        END
```

FIGURE 6-2 Program Listing for ASCII-to-BCD Conversion

7. HEXADECIMAL-TO-ASCII CONVERSION

FUNCTION:

Converts a string of hexadecimal digits to a string of ASCII digits.

PARAMETERS:

Input:	HEX, HEX+1, ----, HEX + (n – 1) HEX is the most-significant digit (2 Hex. digit/byte).
Output:	ASCI, ASCI+1, ----, ASCI + (n – 1) ASCI is the most-significant digit.

Refer to figures 7.1 and 7.2 for flowchart and program listing.

	HARDWARE AFFECTED						
REGISTERS	R0 X	R1 X	R2 X	R3 X	R1′	R2′	R3′
PSU	F	II	SP				
PSL	CC X	IDC X	RS	WC X	OVF X	COM	C X

RAM REQUIRED (BYTES):	nHEX + nASCI
ROM REQUIRED (BYTES):	59
EXECUTION TIME:	Variable
MAXIMUM SUBROUTINE NESTING LEVELS:	0
ASSEMBLER/COMPILER USED:	PIPHASM

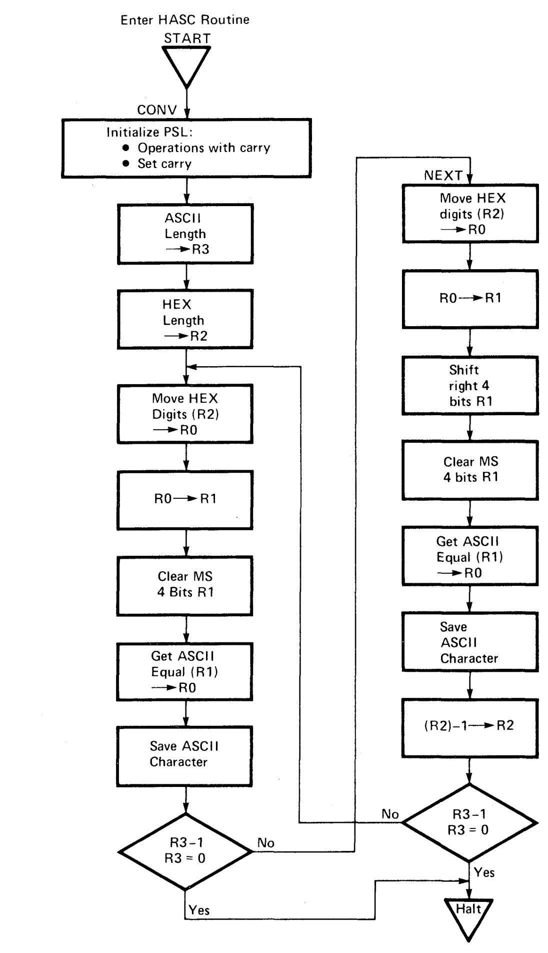


FIGURE 7-1 Flowchart for Hexadecimal-to-ASCII Conversion

```
                    *    PD760056
                    ++++++++++++++++++++++++++++++++++++++++
                    *  HEXIDECIMAL TO ASCII CONVERSION      +
                    ++++++++++++++++++++++++++++++++++++++++
                    *
                    * THIS ROUTINE CONVERTS A STRING OF ASCII
                    * DIGITS TO A STRING OF HEX DIGITS.
                    *
                    * ASCI IS MS DIGIT ASCII.
                    * HEX IS MS DIGIT HEXIDECIMAL.
                    *
                    * DEFINITION OF SYMBOLS:
0000                R0   EQU     0          PROCESSOR-REGISTER
0001                R1   EQU     1
0002                R2   EQU     2
0003                R3   EQU     3
0008                WC   EQU     H'08'      ARITHMETIC CARRY
0001                C    EQU     H'01'      CARRY:BORROW
0003                UN   EQU     3          UNCOND. BRANCH
                    *
                    * IN THIS EXAMPLE 3 HEXIDECIMAL
                    * CHARACTERS ARE CONVERTED.
0002                NUM  EQU     2          HEX BYTE COUNT
0003                NUM1 EQU     3          ASCII BYTE COUNT
                    *
                         ORG     H'600'     RAM DEFINITIONS
0600                HEX  RES     NUM        RESERVES HEX BYTES
0002                HLEN EQU     $-HEX      LENGTH OF HEX
0602                ASCI RES     NUM1       RESERVES ASCII BYTES
0003                ALEN EQU     $-ASCI     LENGTH OF ASCII
                    *
                         ORG     H'500'     START OF ROUTINE
0500 77 09          CONV PPSL    WC+C       ARITH.WITH, SET CARRY
0502 07 03               LODI,R3 ALEN       R3= ASCII LENGTH
0504 06 02               LODI,R2 HLEN       R2= HEX LENGTH
0506 0E 65 FF       CHEX LODA,R0 HEX-1,R2   GET HEX DIGITS
0509 C1                  STRZ    R1         R0 ->R1
050A 45 0F               ANDI,R1 H'0F'      CLEAR MS 4 BITS
050C 0D 65 2C            LODA,R0 ANSI,R1    LOAD ASCII CORRESPONDI
050F CF 66 01            STRA,R0 ASCI-1,R3  SAVE IT
0512 FB 03               BDRR,R3 NEXT       R3-1, R3<> BRANCH
0514 1F 05 2B            BCTA,UN BYE        END OF CONVERSION
0517 0E 65 FF       NEXT LODA,R0 HEX-1,R2   GET HEX DIGITS
051A C1                  STRZ    R1         R0 -> R1
051B 51                  RRR,R1             SHIFT RIGHT 4 BITS
051C 51                  RRR,R1
051D 51                  RRR,R1
051E 51                  RRR,R1
051F 45 0F               ANDI,R1 H'0F'      CLEAR MS 4 BITS
0521 0D 65 2C            LODA,R0 ANSI,R1    LOAD ASCII CORRESPONDI
0524 CF 66 01            STRA,R0 ASCI-1,R3  SAVE IT
0527 FA 00               BDRR,R2 $+2        R2 - 1 CONT.
0529 FB 5B               BDRR,R3 CHEX       R3-1, R3<> BRANCH
                    *
052B 40             BYE  HALT               END OF CONVERSION
                    *
052C 30 31 32 33    ANSI DATA    A'0123456789ABCDEF'
     34 35 36 37
     38 39 41 42
     43 44 45 46
                    *
                         END
```

FIGURE 7-2 Program Listing for Hexadecimal-to-ASCII Conversion

8. ASCII-TO-HEXADECIMAL CONVERSION

FUNCTION:

Converts a string of ASCII digits to a string of hexadecimal digits. The conversion is done by table look-up. Non-numeric ASCII halts this routine. It may be changed to report non-numeric.

PARAMETERS:

Input:	ASCI, ASCI+1, ----, ASCI + (n – 1) ASCI is the most-significant digit.
Output:	HEX, HEX+1, ----, HEX + (n – 1) HEX is the most-significant digit (2 Hex. digits/byte)

Refer to figures 8.1 and 8.2 for flowchart and program listing.

	HARDWARE AFFECTED						
REGISTERS	R0 X	R1 X	R2 X	R3 X	R1'	R2'	R3'
PSU	F	II	SP				
PSL	CC X	IDC X	RS	WC X	OVF X	COM	C X

RAM REQUIRED (BYTES):	nASCI + nHEX
ROM REQUIRED (BYTES):	68
EXECUTION TIME:	Variable
MAXIMUM SUBROUTINE NESTING LEVELS:	1
ASSEMBLER/COMPILER USED:	PIPHASM

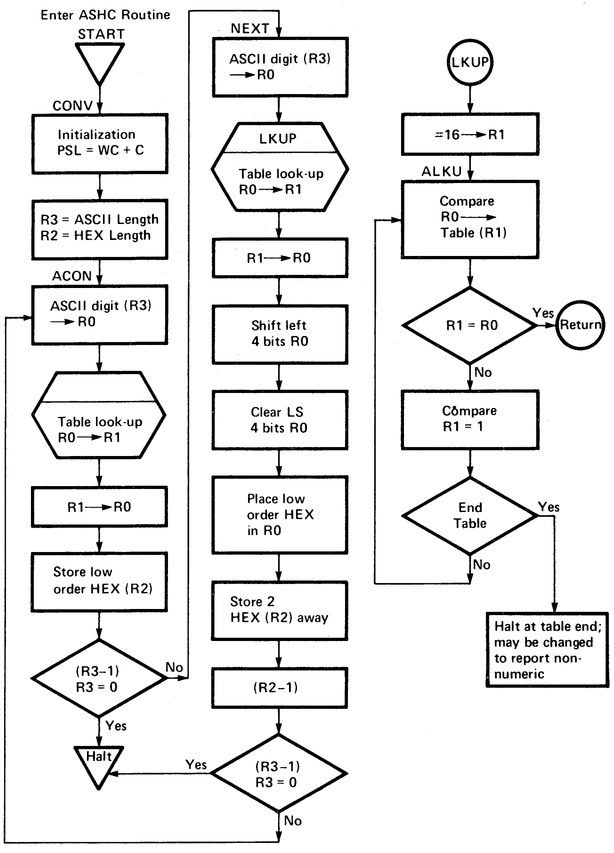


FIGURE 8-1 Flowchart for ASCII-to-Hexadecimal Conversion

```
                      *    PD760057
                      ****************************************
                      *   ASCII TO HEX CONVERSION            *
                      ****************************************
                      *
                      * THIS ROUTINE CONVERTS A STRING OF ASCII
                      * DIGITS TO A STRING OF HEXIDECIMAL DIGITS
                      * ASCI IS MS DIGIT ASCII
                      * HEX IS MS DIGIT HEXIDECIMAL
                      * CONVERSION DONE BY TABLE LOOKUP
                      * NON NUMERIC ASCII HALT ROUTINE
                      *
                      * DEFINITION OF SYMBOLS:
0000                  R0   EQU      0         REGISTER-PROCESSOR
0001                  R1   EQU      1
0002                  R2   EQU      2
0003                  R3   EQU      3
0008                  WC   EQU      H'08'     ARITHMETIC CARRY
0001                  C    EQU      H'01'     CARRY:BORROW
0003                  UN   EQU      3         BRANCH UNCOND.
0002                  LT   EQU      2                 LESS THAN
0000                  EQ   EQU      0                 EQUAL
                      *
                      * IN THIS EXAMPLE 3 ASCII DIGITS
                      * ARE CONVERTED TO HEXIDECIMAL
                      *
0003                  NUM  EQU      3         ASCII BYTE COUNT
0002                  NUM1 EQU      2         HEX BYTE COUNT
                      *
                      *
                           ORG      H'600'    RAM DEFINITIONS
0600                  ASCI RES      NUM       RESERVED ASCII BYTES
0003                  ALEN EQU      $-ASCI    LENGTH OF ASCII
0603                  HEX  RES      NUM1      RESERVED HEX BYTES
0002                  HLEN EQU      $-HEX     LENGTH OF HEX
                      *
                           ORG      H'500'    START OF ROUTINE
0500 77 09            CONV PPSL     WC+C      ARITH.WITH + CARRY SET
0502 07 03                 LODI,R3 ALEN       R3 = ASCII LENGTH
0504 06 02                 LODI,R2 HLEN       R2 = HEX LENGTH
0506 0F 65 FF         ACON LODA,R0 ASCI-1,R3  GET ASCII DIGIT
0509 3B 08                 BSTR,UN LKUP       LOOKUP SUBROUTINE
050B 01                    LODZ    R1         R1 -> R0
050C CE 66 02              STRA,R0 HEX-1,R2   SAVE HEX CORRESPONDING
050F FB 1D                 BDRR,R3 NEXT       (R3-1), R3 <> BRANCH
0511 1B 31                 BCTR,UN BYE        END OF CONVERSION
                      *
0513 05 10            LKUP LODI,R1 16         LOOP CONSTANT
0515 ED 45 1E         ALKU COMA,R0 ANSI,R1,-  COMPARE TO TABLE
0518 14                    RETC,EQ            RETURN - MATCH FOUND
0519 E5 01                 COMI,R1 1          TEST END OF TABLE
051B 9A 78                 BCFR,LT ALKU       NO- LOOK AGAIN
051D 40                    HALT               ERROR - NON NUMERIC HE
                      *
051E 30 31 32 33      ANSI DATA    A'0123456789ABCDEF'
     34 35 36 37
     38 39 41 42
     43 44 45 46
                      *
052E 0F 65 FF         NEXT LODA,R0 ASCI-1,R3  GET NEXT ASCII DIGIT
0531 3B 60                 BSTR,UN LKUP       LOOK UP SUBROUTINE
0533 01                    LODZ    R1         R1 -> R0
0534 D0                    RRL,R0             SHIFT LEFT 4 BITS
0535 D0                    RRL,R0
0536 D0                    RRL,R0
0537 D0                    RRL,R0
0538 44 F0                 ANDI,R0 H'F0'      CLEAR LS 4 BITS
053A 6E 66 02              IORA,R0 HEX-1,R2   COMBINE LOW ORDER
053D CE 66 02              STRA,R0 HEX-1,R2   SAVE 2 HEX DIGITS
0540 FA 00                 BDRR,R2 $+2        (R2-1), CONTIUNE
0542 FB 42                 BDRR,R3 ACON       (R3-1), R3 <> BRANCH
                      *
0544 40               BYE  HALT               END OF CONVERSION
                           END
```

FIGURE 8-2 Program Listing for ASCII-to-Hexadecimal Conversion

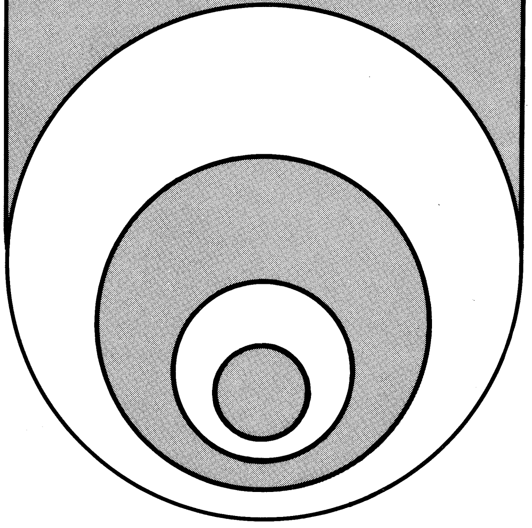
NOTES

Signetics 2650 Microprocessor application memos currently available:

AS50	Serial Input/Output
AS51	Bit and Byte Testing Procedures
AS52	General Delay Routines
AS53	Binary Arithmetic Routines
AS54	Conversion Routines
SP50	2650 Evaluation Printed Circuit Board Level System (PC1001)
SP51	2650 Demo Systems
SP52	Support Software for use with NCSS Timesharing System
SP53	Simulator, Version 1.2
SP54	Support Software for use with the General Electric Mark III Timesharing System
SS50	PIPBUG
SS51	Absolute Object Format (Revision 1)
MP51	2650 Initialization
MP52	Low Cost Clock Generator Circuits
MP53	Address and Data Bus Interfacing Techniques

signetics

MOS
MICROPROCESSOR



2650 DEMO
SYSTEM
SP51

GENERAL

The Demo System (DS) is a hardware base for use with the 2650 CPU printed circuit board (PC1001). The DS provides the user of the 2650 with a convenient "lab bench" set-up for exercising the PC1001 CPU board. The user may expand memory, implement I/O functions, and step through program instructions one at a time using the DS. When the DS is combined with a CPU board (PC1001) and a keyboard terminal, the user is equipped with everything he needs to exercise any of the software or hardware features of the 2650. There are two versions of the DS, the DS1000 and the DS2000. The two Demo Systems are the same except that the DS2000 has a built-in power supply and therefore does not have the power supply binding posts.

FEATURES

The DS provides several connectors to aid the user in exercising the PC1001 CPU board including one for the CPU board itself, one for a memory expansion board, four for I/O ports, and two for communicating with the user's terminal. There are four sets of LED lamps that display the information on the address bus, the data bus, and the two non-extended I/O ports. Two control switches (RUN/PAUSE, and STEP) allow the user to place the 2650 in the WAIT mode and step through program execution one instruction at a time. A reset button is provided on the DS. The DS1000 version has five-way binding posts for connection to external power supplies. The DS2000 has built-in power supplies and does not have the five-way binding posts.

CONNECTORS:

2650 CPU Board Edge Connector (J8). The CPU board connector is an Amphenol dual 50-pin connector (series 225) with 0.125-inch contact centers. The 2650 CPU board (PC1001) is inserted into J8 to complete the Demo

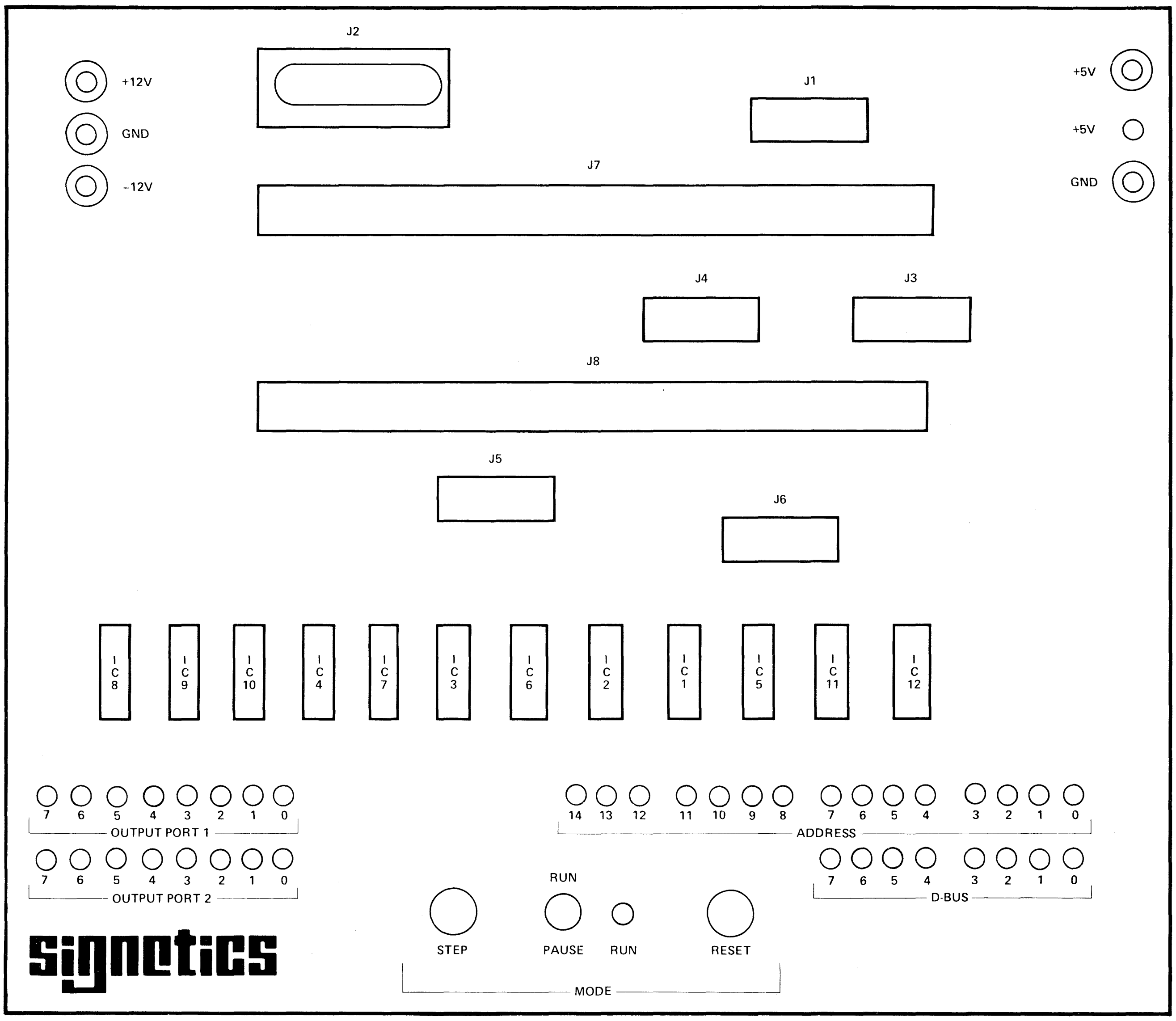
CPU BOARD AND USER BOARD CONNECTORS

PIN#	FUNCTION (J7 & J8)	PIN#	FUNCTION (J8 ONLY)*
1	GND	A	GND
2	GND	B	GND
3	NC**	C	NC
4	$\overline{\text{DBUS0}}$	D	OPD 0
5	$\overline{\text{DBUS1}}$	E	OPD 1
6	$\overline{\text{DBUS2}}$	F	OPD 2
7	$\overline{\text{DBUS3}}$	H	OPD 3
8	$\overline{\text{DBUS4}}$	J	OPD 4
9	$\overline{\text{DBUS5}}$	K	OPD 5
10	$\overline{\text{DBUS6}}$	L	OPD 6
11	$\overline{\text{DBUS7}}$	M	OPD 7
12*	EIPD	N	COPD
13	D/$\overline{\text{C}}$	P	TTY SERIAL IN +
14	$\overline{\text{DMA}}$	R	TTY SERIAL IN –
15	E/$\overline{\text{NE}}$	S	TTY SERIAL OUT +
16	INTACK	T	TTY SERIAL OUT –
17	$\overline{\text{R}}$/W	U	RS232 GROUND
18	WRP	V	RS232 OUTPUT
19	RUN/$\overline{\text{WAIT}}$	W	TTY TAPE READER OUT –
20	OPREQ	X	TTY TAPE READER OUT +
21	M/$\overline{\text{IO}}$	Y	RS232 INPUT
22	$\overline{\text{OPACK}}$	Z	COPC
23	CLOCK	a	OPC 0
24	$\overline{\text{OPEX}}$	b	OPC 1
25	$\overline{\text{RESET}}$	c	OPC 2
26	$\overline{\text{INTREQ}}$	d	OPC 3
27	$\overline{\text{PAUSE}}$	e	OPC 4
28	NC	f	OPC 5
29	NC	g	OPC 6
30	NC	h	OPC 7
31	NC	j	EIPC
32	NC	k	IPD 0
33	ABUS 11	m	IPD 1
34	ABUS 13	n	IPD 2
35	ABUS 12	p	IPD 3
36	ABUS 14	r	IPD 4
37	ABUS 9	s	IPD 5
38	ABUS 10	t	IPD 6
39	ABUS 8	u	IPD 7
40	ABUS 7	v	IPC 0
41	ABUS 6	w	IPC 1
42	ABUS 5	x	IPC 2
43	ABUS 3	y	IPC 3
44	ABUS 0	z	IPC 4
45	ABUS 1	$\overline{a}$	IPC 5
46	ABUS 4	$\overline{b}$	IPC 6
47	ABUS 2	$\overline{c}$	IPC 7
48	+12V	$\overline{d}$	+12V
49	–12V	$\overline{e}$	–12V
50	+5V	$\overline{g}$	+5V

*J7 has no connections to these pins.
**NC = No Connection

TABLE 1

DEMO SYSTEM LAYOUT



NOTE:
THE POWER SUPPLY BINDING POSTS ARE ONLY ON THE DS1000, NOT ON THE DS2000.
THE BINDING POSTS ARE SHOWN ON THIS DRAWING AND MARKED +5V, +12V, -12V, AND GND.

FIGURE 1

CONNECTORS (Continued)

System. The correlation between signal names and pin numbers for J8 is given in Table 1. The location of J8 on the DS is shown in Figure 1.

User Printed Circuit Board Edge Connector (J7). The user board connector is the same type of connector as J8 (the CPU board connector), and makes address, data and control lines available for user-defined interface functions. As shown in Table 1, the numbered pins of J7 and J8 have the same signals on them (except pin 12), while the lettered pins of J7 (pins A through $\bar{g}$) are not used. The J7 connector is typically used for memory expansion. The location of J7 on the DS is shown in Figure 1.

Extended Input/Output DIP Sockets (J5 & J6). The extended I/O DIP sockets make the signals shown in Table 2 available to the user of the DS system. With the signals available on J5 and J6, any type of I/O interface to the 2650 may be implemented. The user of these sockets must supply the cable between his system and the DS, as well as the two 18-pin DIP plugs. The location of J5 and J6 is shown in Figure 1.

EXTENDED INPUT/OUTPUT DIP SOCKETS

PIN #	FUNCTION	
	J5	J6
1	$\overline{\text{DBUS 0}}$	ABUS 0
2	$\overline{\text{DBUS 1}}$	ABUS 1
3	$\overline{\text{DBUS 2}}$	ABUS 2
4	$\overline{\text{DBUS 3}}$	ABUS 3
5	$\overline{\text{DBUS 4}}$	ABUS 4
6	$\overline{\text{DBUS 5}}$	ABUS 5
7	$\overline{\text{DBUS 6}}$	ABUS 6
8	$\overline{\text{DBUS 7}}$	ABUS 7
9	$\overline{\text{OPACK}}$	ABUS 8
10	M/$\overline{\text{IO}}$	ABUS 9
11	OPREQ	ABUS 10
12	RUN/$\overline{\text{WAIT}}$	ABUS 11
13	WRP	ABUS 12
14	$\overline{\text{R}}$/W	ABUS 13
15	INTACK	ABUS 14
16	E/$\overline{\text{NE}}$	$\overline{\text{PAUSE}}$
17	$\overline{\text{DMA}}$	$\overline{\text{INTREQ}}$
18	D/$\overline{\text{C}}$	CLOCK

TABLE 2

Non-Extended Input/Output DIP Sockets (J3 & J4). Each non-extended I/O DIP socket (J3 and J4) makes the signals shown in Table 3 available to the user of the DS system. These sockets may be used for data or command transfer between the 2650 CPU and a user-defined function, but transfers via these channels are initiated by the CPU only. The user of these sockets must supply the cable between his system and the DS, as well as the 18-pin DIP plugs. The location of J3 and J4 is shown in Figure 1.

NON-EXTENDED INPUT/OUTPUT DIP SOCKETS

PIN #	FUNCTION	
	J3	J4
1	OPC 0	OPD 0
2	OPC 1	OPD 1
3	OPC 2	OPD 2
4	OPC 3	OPD 3
5	OPC 4	OPD 4
6	OPC 5	OPD 5
7	OPC 6	OPD 6
8	OPC 7	OPD 7
9	COPC	COPD
10	EIPC	EIPD
11	IPC 7	IPD 7
12	IPC 6	IPD 6
13	IPC 5	IPD 5
14	IPC 4	IPD 4
15	IPC 3	IPD 3
16	IPC 2	IPD 2
17	IPC 1	IPD 1
18	IPC 0	IPD 0

TABLE 3

RS232 Interface Connector (J2). The RS232 interface connector is a TRW 25-pin connector (part #DB25S) for communicating with RS232-compatible input/output devices. The pins used on this connector are shown in Table 4 along with the corresponding signal names. The RS232 driver and receiver are on the PC1001 circuit board and are wired to J2 through the DS circuit board. The location of J2 on the DS board is shown in Figure 1.

RS232 INTERFACE CONNECTOR (J2)

PIN #	FUNCTION – J2
1	RS232 GROUND
2	RS232 INPUT
3	RS232 OUTPUT
5	JUMPER
6	JUMPER
7	RS232 GROUND
8	JUMPER
20	JUMPER

TTY INTERFACE DIP SOCKET (J1)

PIN #	FUNCTION – J1
1	TTY SERIAL IN +
2	TTY SERIAL IN –
8	TTY TAPE READER OUT –
9	TTY TAPE READER OUT +
13	TTL SERIAL OUT –
14	TTL SERIAL OUT +

TABLE 4

TTY Interface DIP Socket (J1). The TTY interface socket is a 14-pin DIP socket and is used for communicating with a current loop serial interface. The pins used on this connector are shown in Table 4 along with the corresponding signal names. The current loop driver and receiver circuits are on the PC1001 board and are wired to J1 through the DS circuit board. The location of J1 on the DS board is shown in Figure 1.

DISPLAYS:

Address Display LEDs. The address display LEDs reflect the information on the address bus (ABUS 0-ABUS 14) when the PC1001 board is plugged into J8. The logic circuits on the DS board loads the information from the address bus into D-type latches on the occurrence of every Operation Request (OPREQ) pulse. Open collector inverters at the output of the D-type latches drive the LED's in a common anode configuration.

Data Bus Display LEDs. The data bus display LEDs reflect the information on the data bus (DBUS 0-DBUS 7) when the PC1001 board is plugged into J8. The information on the data bus is stored into D-type latches on every OPREQ pulse. The LEDs are driven directly from the D-type latches in a common anode configuration.

DISPLAYS (Continued)

Non-Extended Input/Output Channel LEDs. The non-extended I/O channel LEDs are driven by open collector inverters in a common anode configuration. The inverters are driven by the output latches of the two non-extended I/O ports on the PC1001 printed circuit board. Output Port 1 (2), bit 0 corresponds to DBUS 0 and Output Port 1 (2), bit 7 corresponds to DBUS 7. A logic "1" output from the 2650 turns on the LEDs, and a logic "0" turns off the LED.

+5V LED and RUN LED. The +5V LED will glow when a +5 volt power supply is connected to the Demo System. The DS1000 requires an external power supply, but the DS2000 has the +5 volt power supply built into the base. The RUN LED will glow when the RUN/WAIT line from the 2650 is in the "high" logic state. The location of these LED's is shown in Figure 1.

CONTROLS:

RESET Button. The reset button is a momentary switch that is tied directly to the Reset input on J8 (pin 25), and pulls that pin "low" when the button is pushed. This button clears the program counter in the 2650 to zero. The location of the reset button is shown in Figure 1.

PAUSE Switch and STEP Button. The pause switch and the step button are used together to cause the 2650 microprocessor to execute one instruction at a time. When the pause switch is in the RUN position, the step button does not affect the operation of the microprocessor.

When the pause switch is placed into the PAUSE position, the $\overline{\text{PAUSE}}$ line on the 2650 is pulled "low". When the execution of the current instruction is completed, the 2650 will enter the WAIT mode and the RUN/$\overline{\text{WAIT}}$ line will go "low". If the step button is pressed, the $\overline{\text{PAUSE}}$ line to the 2650 will be pulled "high" until the RUN/$\overline{\text{WAIT}}$ line goes "high", indicating that the 2650 is in the RUN mode. As soon as the RUN/$\overline{\text{WAIT}}$ line goes "high", the DS will again pull the $\overline{\text{PAUSE}}$ line "low". The step button will allow one instruction to be executed each time it is pushed as long as the pause switch is in the PAUSE position. When the pause switch is placed back onto the RUN position, the $\overline{\text{PAUSE}}$ line will be pulled "high" and the 2650 will execute instructions in a continuous manner. The address and data displayed on the DS LEDs in the WAIT mode reflect the address and the first byte of the next instruction to be executed. The location of the pause and step switches on the DS base is shown in Figure 1.

LOGIC CIRCUITS

The logic circuits on the DS base are shown in Figure 2. The logic circuits consist of address bus (ABUS) and data bus (DBUS) latches, the pause and step logic, LED drivers, and a reset switch. The address and data bus are loaded into latches on the DS during every OPREQ. The displays for the address and data bus will flicker while the run LED is "lighted", and will display the address and first byte of the *next* instruction to be executed when in the step mode (run LED off). The pause and step logic allows one instruction to be executed at a time by pushing the step button when the run/pause switch is in the PAUSE position. The non-extended output ports are displayed on the DS, and the reset button provides complete system reset by pushing the button.

ADDRESS BUS:

The address bus latches are 74174 Hex D-type flip-flops (IC1, IC2, IC3). Open collector inverters (IC5, IC6, IC7) invert the "positive true" levels from the ABUS latches and drive the address bus LEDs (L1-L15) in a common anode configuration. A logic ONE on the address bus "lights" the corresponding LED, and ABUS 0 corresponds to the ADDRESS bit 0 LED. The ABUS latch is clocked by the STRB signal which is generated by 4 inverters (IC7, IC10). The inverters provide the logic function $\overline{\text{STRB}}$ = OPREQ • CLOCK. The ABUS latches are reset by $\overline{\text{RESET}}$.

DATA BUS:

The data bus latches are also 74174 Hex D-type flip-flops (IC3, IC4). Since the DBUS leaves the PC1001 with "negative true" logic levels, the DBUS latches drive the LEDs directly in a common anode configuration. A logic ONE in the DBUS latches is a low voltage level and "lights" the corresponding LED. The DBUS bit 0 LED corresponds to DBUS 0. The DBUS latch is also clocked by the signal STRB, and reset by $\overline{\text{RESET}}$.

PAUSE AND STEP:

The pause and step switches are de-bounced with S/R latches. The step switch uses two NAND gates (IC11), while the pause switch uses a D-type latch (IC12) to accomplish the de-bounce function. When the pause switch is in the RUN position, SPAUSE is a logic ZERO and the $\overline{\text{PAUSE}}$ line is held at logic ZERO (de-activated).

When the pause switch is set to the PAUSE position, SPAUSE is a logic ONE and $\overline{\text{PAUSE}}$ will switch to a logic ONE. When the $\overline{\text{PAUSE}}$ line switches to a logic ONE, the 2650 will finish executing the current instruction, fetch the first byte of the next instruction from memory, and enter the wait state. The RUN/$\overline{\text{WAIT}}$ line goes to a logic ZERO when the 2650 enters the wait state. If the step switch is pushed, LSTEP clocks a logic ONE into the CLSTEP latch (IC12) which sets $\overline{\text{PAUSE}}$ to a logic ZERO. The 2650 then returns to the run mode, and the RUN/$\overline{\text{WAIT}}$ line goes to a logic ONE. When the RUN/$\overline{\text{WAIT}}$ line switches to a logic ONE, the CLSTEP latch is reset and

DS1000 LOGIC DIAGRAM

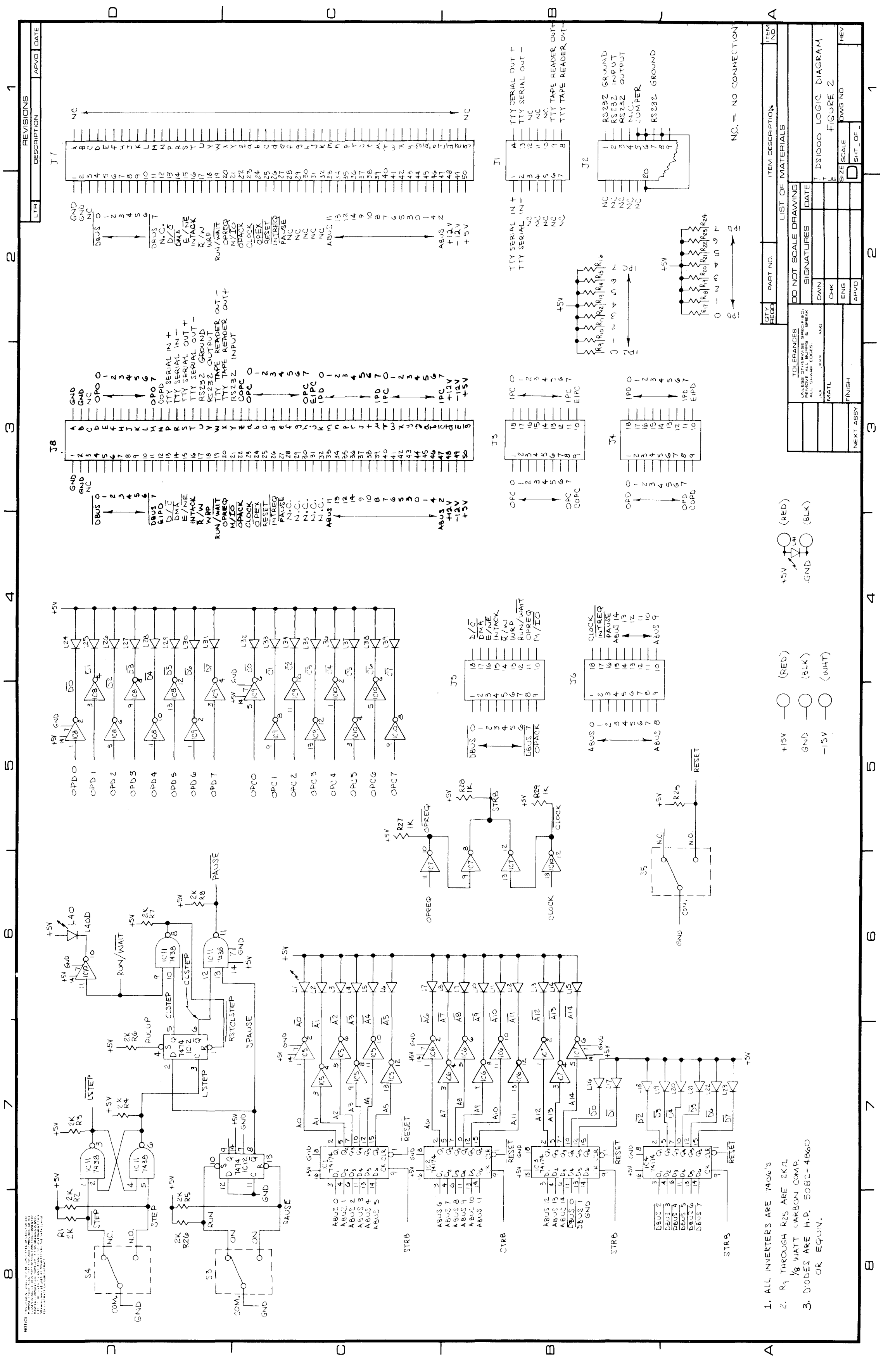


FIGURE 2

$\overline{\text{PAUSE}}$ returns to a logic ONE. This process is repeated once each time the step button is pushed. When the pause switch is returned to the RUN position, the $\overline{\text{PAUSE}}$ line is set to a logic ZERO and the 2650 will return to the run mode. The step/pause function is implemented with IC11 (NAND gate) and IC12 (D-type latch).

OUTPUT CHANNEL DISPLAYS:

The two non-extended output channels implemented on the PC1001 board are displayed on the DS. The output bits, (OPD 0 - OPC 7) are received by open collector inverters which in turn drive the LEDs. A logic ONE output to port 1 (WRTD instruction) will "light" the corresponding OPD LED, while a logic ONE to port 2 (WRTC instruction) will "light" the corresponding OPC LED. Signal OPD 0 corresponds to Output Port 1 bit 0, and OPC 0 corresponds to Output Port 2 bit 0.

RUN AND +5V DISPLAYS:

When the 2650 is in the run mode, the run LED will be "lighted". When +5 volts is applied across the red and black terminals of the DS1000, the +5V LED will be "lighted." When a.c. power is applied to the DS2000 (internal power supply), the +5V LED will be "lighted".

RESET:

The reset switch (S5) pulls the $\overline{\text{RESET}}$ line to a logic ONE when pushed. The $\overline{\text{RESET}}$ line is tied to the corresponding pin on the PC1001 board (pin 25) as well as the ABUS and DBUS latches on the DS.

DEMO SYSTEM PARTS LIST

Item #	Description	ID#	Mfg. and Part #
1.	Base Box	–	–
2.	Printed Circuit Board	–	–
3.	100-Pin Connector	J7, J8	Amphenol, series 225
4.	18-Pin Dip Socket	J3, J4 J6, J6	Cambion 703-3787-01-04-16
5.	14-Pin Dip Socket	J1	Cambion 703-4000-01-04-16
6.	SPDT Push Button Switch	S4, S5	Alco, MSP105F
7.	SPDT Toggle Switch	S3	Alco, MTA106D
8.	LED	L1-L41	H.P. 5082-4870450
9.	5-Way Binding Post		H.H. Smith
10.	RS232 Connector	J2	TRW Cinch DB25S
11.	Carbon Composition Resistors – 2KΩ	R1-R29	Allan Bradley RC05GF202J
12.	Aluminum Standoff		H.H. Smith 8352
13.	Tinnerman Speed Nuts		Tinnerman C8093-632

POWER SUPPLY SPECIFICATIONS

(DS1000 Only, Power Supply Included With DS2000)

5 Volt Power Supply	Line Regulation 0.1%
	Load Regulation 0.1%
	Ripple 10m Volts (maximum)
	Response Time 30 usec (maximum)
	Output Current 4 amps (To supply PC1001 only)
	Overvoltage Protection
	Current Overload Protection
±12 Volt Power Supply	Line Regulation 0.1%
	Load Regulation 0.1%
	Ripple 10m Volts (maximum)
	Response Time 30 usec (maximum)
	Output Current 50 milliamps (To supply PC1001 only)
	Overvoltage Protection
	Current Overload Protection

signetics

MOS
MICROPROCESSOR

SUPPORT SOFTWARE FOR USE WITH THE NCSS TIMESHARING SYSTEM SP52

SUPPORT SOFTWARE FOR USE WITH THE NCSS TIMESHARING SYSTEM

1. INTRODUCTION

A series of programs is described that provide the microprocessor application's design engineer with on-line support for the development of programs to be run on the Signetics 2650 microprocessor. These programs include a cross-assembler, a cross-simulator, and two utility programs that convert the object file produced by the assembler into either one of two tape formats – one suitable for loading into the 2650 microprocessor and the other suitable for burning PROMs. The programs are accessed through a communications terminal connected to a National CSS Data Center via standard telephone lines.

The first few sections describe the available programs and provide detailed instructions for using them. All available usage options are included as reference information. A final section, called "Operating Instructions," provides the user with step-by-step procedures for generating, editing, assembling, punching, and simulating Signetics 2650 programs. These procedures explain some of the more commonly used features of both the NCSS and the Signetics facilities and demonstrate how to use them.

2. USAGE OVERVIEW

The user creates the source file for his assembly language program by using the EDIT facility available on the NCSS system, or he may have his program punched onto cards and read into the system at a NCSS Data Center. Once the source file resides on the system, the user executes the assembler, which translates symbolic source statements into machine language instructions, and generates both an assembled listing of the source file and an object file. If the assembler reports any errors in the source file, the EDIT facility may again be invoked to correct the source file. The corrected source file is then resubmitted to the assembler. Once the assembler reports no errors, the user may input the object file to the simulator which then simulates execution of the program. The simulator provides the following capabilities:

1) Establishes initial program conditions.
2) Monitors execution sequences.
3) Modifies the program until it operates as desired.

Once the program operates correctly, the user may repeat the entire cycle: correct his source file; reassemble; and test the new program using the simulator. When the program is fully tested and debugged, it may be punched onto tape.

3. EXECUTING 2650 SUPPORT PROGRAMS

A. GENERAL

To execute any of the 2650 support programs, the following command must be entered:

```
ATTACH P2650
```

This causes the P2650 "PROTECT" Exec to execute. It prints:

```
P2650 Attached as XXX, (Y) RUN? >
P2650 - Version "No." - "Date"
Run on "DATE"
ENTER COMMAND (e.g., HELP) >
```

At this point the user may enter any one of the following commands:

HELP	Print Command List
HELP 'NAME'	Print Command in Detail
QUIT	Exit P2650 (Return to VP/CSS)
NEW	Print New Features
PIPHASM	Assemble 2650 Program
PIPSIM	Simulate 2650 Program
PIPHTAP	Punch PIPBUG Tape
PIPSTAP	Punch PROM Burning Tape

No other CSS command may be executed while under control of the P2650 "PROTECT" Exec; e.g., you cannot edit your file until you exit P2650 by typing "QUIT":

```
ENTER COMMAND > QUIT
```

B. HELP – AN ON-LINE INFORMATION RESOURCE FACILITY

To determine what commands are currently available on P2650, type:

```
HELP
```

To obtain information on how to enter any command except HELP or QUIT, type HELP followed by the name of the desired command; e.g.,

```
HELP PIPHASM
```

A description of the command and its format will be printed.

4. PROGRAM DESCRIPTIONS

A. PIPHASM – SIGNETICS 2650 PIP ASSEMBLER

PIPHASM supports the 2650 assembler languages as specified in the basic manual set (2650 BM 1000). It outputs a hexadecimal object module in a format acceptable to the two tape-punching programs, PIPHTAP and PIPSTAP, and to the simulator, PIPSIM.

Following is the format of the command for executing the assembler:

PIPHASM SOURCE (DISPLAY) (WIDTH)*

where

PIPHASM causes the assembler to execute.

SOURCE is the name of the user's source file. This file has a type of "SYSIN".

DISPLAY is an optional parameter specifying that the listing is to be printed either on the user's console (CON) or on the off-line printer (PTR). If this parameter is missing, CON is assumed.

WIDTH is an optional parameter specifying the line width of the user's console in characters per line—either 80 characters (1) or 120 characters (0). If no parameter is specified, 120 characters per line is assumed. This parameter may be specified only if CON has been specified by DISPLAY.

The object file produced by the assembler will have the same file name as the input file with ".OBJ" concatenated at the end; it will have a filetype of "DATA".

B. SIGNETICS 2650 SIMULATOR

The 2650 simulator, a program written in FORTRAN IV, simulates the execution of a program without using the 2650 processor. The simulator executes a 2650 program by maintaining its own internal FORTRAN storage registers to describe the program, the microprocessor registers, the ROM/RAM memory configuration, and the input data to be read dynamically from I/O devices. The user may request traces of the processor status, dumps of the memory contents, and program timing statistics. Multiple simulations of the same program with different parameters may be executed during one simulation run.

The simulator requires as input both the program object module produced by the 2650 assembler and a file of user commands. It produces a listing of user commands, executes the program, and prints ("displays") both static and dynamic information as requested by the user commands. The user may direct the input of the simulator either to a terminal or to a line printer.

PIPSIM SOURCE COMMAND (DISPLAY)

where

PIPSIM causes the simulator to execute.

*Parenthesis indicate an optional parameter with a default value.

SOURCE is the name of the source file originally submitted to the assembler. The simulator concatenates .OBJ onto the name of the source file and uses the designator, SOURCE.O, to find the file containing the object module of the program to be executed. File names are limited to eight characters. This object module is ordinarily produced by the assembler and has a filetype of "DATA."

COMMAND is the name of a file containing the user's commands. This file has a filetype of "DATA."

DISPLAY is an optional parameter specifying the destination of all printed output either to the user's console (CON) or to the off-line printer (PRT). If no parameter is specified, the user's console is assumed.

C. PAPER TAPE UTILITIES

1) PIPHTAP

PIPHTAP punches the "hex" object file onto tape in a format acceptable as input to the 2650 Prototyping Card (2650 PC 1000). See Signetics Applications Memo SS51 for the tape format specification.

The command format for PIPHTAP is:

PIPHTAP SOURCE

where

SOURCE is the name of the source file originally submitted to the assembler.

When "EXECUTION:" is printed, turn the punch on.

2) PIPSTAP

PIPSTAP punches the "hex" object file onto tape in a form suitable for burning PROMs in SMS format. PIPSTAP uses the same command format as PIPHTAP; i.e.,

PIPSTAP SOURCE

where

SOURCE is the name of the source file originally submitted to the assembler.

PIPSTAP responds with a request for the following information:

- The name of your object file.
- The value (two hexadecimal digits) representing the unburned state of your PROM.
- The byte size (four decimal digits) of the PROMs to be burned.
- Up to eight pairs of START/END addresses (four hexadecimal digits). Each address pair identifies an area of code in the object module.

NOTE: All numbers entered *must* contain leading zeros; e.g., when entering the size of a PROM as 256, you must enter 0256.

A START address larger than 7FFF, e.g., 8000, terminates the input mode. Once the input mode is terminated, the punch must be turned on. PIPSTAP punches and prints a record for each PROM specified.

START/END addresses are rounded down/up to the limits of the affected PROM. Thus if:

```
          INITIAL PROM VALUE  = FF,
          PROM SIZE           = 0256,
          START ADDR          = 0040,
     and
          END ADDR            = 0240,
```

PIPSTAP punches three records: 0000 – 00FF, 0100 – 01FF, and 0200 – 02FF. Each of the records is preceded by its initial address (0000, 0100, and 0200). This initial address is punched into the tape so that it is visible. This enables the tape to be separated into individual strips for each PROM. The areas 0000 - 003F and 0241 - 02FF are filled with FFs.

Each record is punched in exactly the order that its START/END address was entered so that multiple records may be punched for the same PROM. When PIPSTAP stops punching, turn the punch off.

5. OPERATING INSTRUCTIONS FOR USING THE NCSS TIMESHARING SERVICE

A. GENERAL

1. The computer requests the user to type information by printing a > character at the start of a line.

2. The user terminates each line typed with a carriage return.

3. The user deletes (tells the computer to ignore) characters that were erroneously typed by typing the @ character. The computer deletes one preceding character for each @ character typed; e.g., the message LANE@@@INE corrects the word LANE to LINE. The [character deletes all characters previously typed on the line.

4. In all of the following examples, lines typed by the user are underlined to distinguish them from lines printed by the computer.

B. LOGGING IN TO CSS

1. Set the terminal to "LINE" mode.

2. Select the half-duplex mode using the HALF/FULL duplex switch on your terminal (not required on some terminals).

3. Dial the NCSS-supplied telephone number.

4. When you hear a high-pitched tone (indicating that you have established communication with the computer), place the telephone receiver in the modem coupler.

5. Log on by typing an 'S' or a 'O' followed by a carriage return; i.e.,

S carriage return	(when using a 10 cps terminal)
O carriage return	(when using a 30 cps terminal)

In response, the system types

```
CSS ONLINE – XXXX
```

to signal that you have reached an NCSS monitor. XXXX is the name of the NCSS system with which you have established a connection. The system also types the prompt character >, indicating that it is ready to accept additional input from your terminal. In response, you should type:

```
> L WEST XXXXXX
```

where XXXXXX is your user ID number.

The system will respond with

```
PASSWORD
XXXXXXXX
```

providing a blocked-out area in which you enter your password. Type the password on top of the blocked-out area and press the carriage return.

When the system responds with

```
A/C INFO:
```

press the carriage return. (You may optionally enter some accounting information if you desire.).

Messages from the NCSS system are printed here.

```
CSS.211 data
time>
```

C. USING THE EDITOR TO CREATE A NEW SOURCE FILE AND/OR TO EDIT AN EXISTING SOURCE FILE

1) Creating a New Program Source File

a. On NCSS every file has a file name (FN) and a file type (FT). A file name is the unique name to be assigned to your program. Assign your program a file name of 1-to-4 alphanumeric characters beginning with an alphabetic character. The file type of your source program is "SYSIN." The object file created by the assembler is your unique file name plus the .OBJ appendage. The file type of object files is "DATA."

b. The timesharing computer stores all source and object files on disk. The user may obtain a directory of the files stored in his user area by typing the letter, L, e.g.,

```
time > L
FILENAME    FILETYPE   MODE   ITEMS
PROG        SYSIN      P      40
PROG.OBJ    DATA       P      5
```

c. To create a new program source file, the user calls the editor program with an indication of the file name and file type to be created. The editor recognizes that the file name specified is not in the directory and creates a new file.

```
time > E filename SYSIN
NEW FILE.
INPUT:
```

Type your program here. If you make mistakes, use the @ key or finish typing your program and make corrections as specified in step d below. Type an additional carriage return after the last line to exit the new file input mode. The system responds with:

```
EDIT:
```

d. If you wish to edit your program (i.e., correct any typing errors or omissions), proceed to step 2b below. If you do not wish to edit your program at this time, type "FILE" to exit the editor.

2) Editing a Program Source File

a. The edit mode can be entered directly when the editor is called by specifying a filename-filetype already on disk; e.g., if PROG SYSIN already exists on disk, enter:

```
time > E PROG SYSIN
EDIT:
```

Enter edit commands.

b. The system's editing capabilities are based on the pointer concept; i.e., any line in a file can be located by an imaginary pointer. This pointer can be moved up or down, positioned at the beginning or end of the file, or positioned at a specific line. The position of the pointer determines where the next edit request takes place. The position of the pointer is referred to as the "current line."

c. Following is a list of some of the most frequently used editing commands: *(NOTE: Whenever "n" is indicated in a command, it represents a decimal number. If "n" is left off the command, the number 1 is assumed.)*

>T	Moves the pointer to the first line of the file.
>DO n	Moves the pointer down n lines and prints the new current line.
>UP n	Moves the pointer up n lines and prints the new current line.
> L/string/	Moves the pointer to the next line which contains the character string specified between the slash delimiters. It then prints that line. It does not search the current line for the string. If the character string contains a /, then some other character, such as the $, may be used as the delimiter.
>P n	Print n lines starting with the current line. Also move the pointer to the last line printed. If n = 1 or is absent, the current line is printed and the pointer is not moved.
>DE n	Delete n lines starting with the current line.
>R text	Replace the entire line following the pointer with the text on the R line. The text is separated from the R by only 1 blank. Any additional spaces are considered part of the text.
>C /string 1/ string 2/	Replace character string 1 in the current line with character string 2. If the / character appears in either of the strings, use some other character, such as the $, as the string delimiter.
>I INPUT: >. . . . . > EDIT:	An I followed by a carriage return puts the editor into input mode. This request is issued to insert lines after the current line. After the "INPUT:" message is printed, the user types one or more lines to be inserted into the program. The last line typed should be followed by two carriage returns to return to EDIT mode. The pointer is moved to point to the last line inserted.

d. Error Messages

Editor error messages are as follows:

?	Invalid edit request.

EOF:	The end of file is reached by an edit request. The request is terminated, and the pointer is positioned after the last line of the file.
TRUNCATED	The following line was truncated as shown. Only 72 character lines are permitted.

e. Exiting the Editor

To exit the editor and save your new file, type:

```
>FILE
```

To exit the editor without changing your original file, type:

```
>QUIT
```

D. ASSEMBLING THE PROGRAM SOURCE FILE TO CREATE A HEXADECIMAL FORMAT OBJECT FILE FOR PROGRAM SIMULATION AND FOR PROGRAM DEBUGGING ON THE PROTOTYPING SYSTEM

```
time > ATTACH P2650
P2650 ATTACHED AS 192, (T)
P2650 - Version 2.0 - 1/5/76
RUN ON 'DATE'
P2650 COMMAND (e.g., HELP) > PIPHASM filename
P2650 ASSEMBLER . . .
RUN . . . (YES OR NO)? > YES
EXECUTION:
```

(Your assembly listing will be printed here. Be patient—there may be a short delay before printing starts.)

```
TOTAL ASSEMBLER ERRORS = X
```

E. LOGGING OFF NCSS

Exit P2650, log off the NCSS system, and review your program for logical and syntactical errors.

```
ENTER COMMAND > QUIT
time > LOGOUT
XXXX VPU'S,XX CONNECT HRS,XX I/O
LOGGED OFF AT time ON date
```

F. CHECKING OUT YOUR PROGRAM USING THE SIMULATOR

1. Log on the system as described in step B.

2. Using the editor program, create a file containing the simulator commands. This file is of type DATA. The file name may be the same as the source file name but with a .TST appendage:

```
time > E filename.TST DATA
NEW FILE
INPUT:
```

NOTE: The directions for using the editor described in steps C.1 and C.2 apply here also.

Enter commands here.

```
EDIT:
   >FILE
```

3. Request a simulator run.

```
time > ATTACH P2650
P2650 ATTACHED AS 192, (T)
P2650 - Version 2.0 - 1/5/76
RUN ON 'DATE'
P2650 COMMAND; e.g., 'HELP' > PIPSIM filename
  filename.TST
P2650 SIMULATOR . . .
RUN . . . (YES OR NO) ? > YES
EXECUTION:
```

The simulator listing is printed here.

G. LOGGING OFF

Exit P2650, log off the NCSS system, and review the simulator listing to determine program correctness.

```
P2650 COMMAND > QUIT
time > LOGOUT
XXXX VPU'S XX CONNECT HRS, XX I/O
LOGGED OFF AT time ON date.
```

H. PUNCHING A PAPER TAPE FOR DEBUGGING ON THE PROTOTYPE CARD SYSTEM

Check to ensure that the punch is off. After the "EXECUTION:" message is printed by the computer, turn the punch on. Turn the punch off after it stops punching.

```
P2650 COMMAND > PIPHTAP filename
```

(NOTE: Do not use the .OBJ extension on the filename. The punch program assumes this is the .OBJ file and automatically adds this extension.)

```
EXECUTION:
```

.OBJ file will be listed here.

```
P2650 COMMAND > QUIT
```

Log off the system as in step G above.

I. PUNCHING A PAPER TAPE FOR BURNING PROMS

Check to see that the punch is off, and log into the system using the procedures outlined in step B.

Execute PIPSTAP:

```
P2650 COMMAND (e.g., HELP) > PIPSTAP filename
P2650 PIPSTAP . . .
RUN . . . (YES OR NO)? > YES
```

PIPSTAP responds with a request for the unburned state of your PROM. Since PIPSTAP punches data into each location of the PROM, if your object module does not fill the entire PROM, PIPSTAP requires a value that can be used for the other locations. This value must be entered as two hexadecimal digits:

```
    INITIAL PROM VALUE? 00
```

PIPSTAP then asks for the size (in bytes) of your PROM, which must be entered in four decimal digits. The maximum allowable size is 1024.

```
    PROM SIZE? 0256
```

PIPSTAP requests both a START and an END address for the code to be punched. Use four hexadecimal digits for each address as shown below. Don't forget the leading zeros.

```
    START ADDR? 0000
    END ADDR?   000A
```

PIPSTAP will request up to eight pairs of START/END addresses. Enter a number larger than 7FFF, e.g., 8000, when you have completely described the object module:

```
    START ADDR? 8000
```

When you press

Carriage Return

PIPSTAP punches 50 frames of leader followed by the PROM record specified by your START and END addresses. The START address of your PROM, 0000, is punched into the tape so that it is visible.

When punching is complete, turn the punch off and log off the system.

6. REFERENCE DOCUMENTS

For additional information, consult the following manuals:

- Signetics 2650 Microprocessor Manual (2650BM 1000)
- VP/CSS Reference Manual (Form 106-3, available from NCSS)
- VP/CSS Edit Command (Form 108-4, available from NCSS)

signetics

MOS MICROPROCESSOR

SUPPORT SOFTWARE FOR USE WITH GE'S MARK III TIMESHARING SYSTEM SP54

signetics

SUPPORT SOFTWARE FOR USE WITH GE's MARK III TIMESHARING SYSTEM

SP54

2650 MICROPROCESSOR

APPLICATIONS MEMO

1. SUMMARY

A series of programs is described that provide the microprocessor application's design engineer with on-line support for the development of programs to be run on the Signetics 2650 microprocessor. These programs include a cross-assembler, a cross-simulator, and two tape utility programs that convert the object file produced by the assembler into either a "hex" format, suitable for loading into system memory by "PIPBUG," or into a format suitable for burning PROMs. The programs are accessed through a communications terminal connected to General Electric's Mark III Timesharing System via standard telephone lines.

2. USAGE OVERVIEW

The user creates the source file for his assembly language program by using the editing facility or his program may be punched onto cards and read into the system. Once the source file resides in the system, the user executes the assembler, which translates symbolic source statements into machine language instructions, and generates both an assembled listing of the source file and an object file. If the assembler reports any errors in the source file, the user may again invoke the editing facility to correct the errors. The corrected source file is then resubmitted to the assembler. Once the assembler reports no errors, the user may input the object file to the simulator which simulates execution of the program.

The simulator provides the following capabilities:

1) Establishes initial program conditions.
2) Monitors execution sequences.
3) Modifies the program until it operates as desired.

Once the program operates correctly, the user may repeat the entire cycle: correct his source file, reassemble, and test the new program using the simulator. When the program is fully tested and debugged, it may be punched onto tape in a format for loading into system memory and/or for burning PROMs.

3. PROGRAM DESCRIPTIONS

The next few sections describe the available programs and provide detailed instructions for using them. All available usage options are included as reference information. A final section, called "Operating Instructions," provides step-by-step procedures for generating, editing, assembling, simulating, and punching Signetics 2650 programs. These procedures explain some of the more commonly used features of both the General Electric Timesharing System and the Signetics facilities and demonstrate how to use them.

A. PIPHASM – SIGNETICS 2650 PIP ASSEMBLER (HEX TAPE FORMAT)

PIPHASM supports the 2650 assembler language as specified in the basic manual set (2650 BM 1000). It outputs a hexadecimal object module in a format acceptable to the two tape-punching programs, PIPHTAP and PIPSTAP, and to the simulator, PIPSIM.

To execute the assembler, enter the command:

```
/PIPHASM
```

The assembler will start executing and will request the following information:

- The name of the input (source) file.
- The name assigned to the assembler-produced object file. It is suggested that some naming convention be adopted; e.g., always name the object file with the first four letters from the name of the source file followed by ".OBJ".
- The width of your terminal carriage. Enter "0" if your terminal carriage has 120 characters; otherwise, enter "1".

To assemble your program, the assembler creates a scratch file on your user ID. If the assembly runs to completion, this file will be purged. But if the assembly is aborted, the file may remain on your user ID. You may collect up to ten of these scratch files before the assembler will be unable to assemble because it cannot find a scratch file name. The scratch file names that must be purged are referred to as: A . . . . 00, A . . . . 01, . . . , A . . . . 09.

B. SIGNETICS 2650 SIMULATOR

The 2650 simulator, a program written in FORTRAN IV, simulates the execution of a 2650 program without using the 2650 processor. The simulator executes a 2650 program by maintaining its own internal FORTRAN storage registers to describe the 2650 program, the microprocessor registers, the ROM/RAM memory configuration, and the input data to be read dynamically from I/O devices. The user may request traces of the processor status, dumps of the contents of memory, and program timing statistics. Multiple simulations of the same program with different parameters may be executed during one simulation run.

The simulator requires as input both the program object module produced by the 2650 assembler and a file of user commands. It produces a listing of the user's commands, executes the program, and prints ("displays") both static and dynamic information as requested by the user's commands.

The Signetics Basic Manual Set (2650 BM 1000) contains a description of the user commands and the general operation of the simulator.

To execute the simulator, enter the command:

```
/PIPSIM
```

The simulator starts executing and requests the following information:

- The name of the object module produced by the assembler for your program.
- The name of the file of simulator commands.

C. PAPER TAPE UTILITIES

The two paper tape utility programs, PIPHTAP and PIPSTAP, complete the series of programs discussed in this memo.

1) PIPHTAP

PIPHTAP punches the "hex" object file onto tape in a format acceptable as input to the 2650 Prototyping Card (2650 PC 1001). Refer to Signetics Applications Memo SS51 for the tape format specifications.

To execute PIPHTAP, enter the command:

```
/PIPHTAP
```

PIPHTAP responds with a request for the name of your object (input) file; it then requests that the punch be turned on and that the carriage return key be depressed. PIPHTAP punches about 50 frames of leader before it punches the object module. When the system responds with "READY," turn the punch off.

2) PIPSTAP

PIPSTAP punches the "hex" object file onto tape in a form suitable for burning a PROM. To execute PIPSTAP, enter the following command:

```
/PIPSTAP
```

PIPSTAP responds with a request for the following information:

- The name of the object file.
- The value (two hexadecimal digits) representing the unburned state of your PROM.
- The size in bytes (four decimal digits) of the PROMs to be burned.
- Up to eight pairs of START/END addresses (four hexadecimal digits). Each address pair identifies an area of code in the object module.

NOTE: All numbers entered must contain leading zeros; e.g., when entering the size of a PROM as 256, you must enter 0256.

A START address larger than 7FFF, e.g., 8000, terminates the input mode.

Once the input mode is terminated, PIPSTAP requests that the punch be turned on. It then punches and prints a record for each PROM specified.

START/END addresses are rounded down/up to the limits of the affected PROM. Thus if:

```
        INITIAL PROM VALUE  =  FF,
        PROM SIZE           =  0256,
        START ADDR          =  0040
and
        END ADDR            =  0240,
```

PIPSTAP punches three records: 0000 – 00FF, 0100 – 01FF, and 0200 – 02FF. Each of the records is preceded by its initial address (0000, 0100, 0200) punched into the tape so that it is visible. This enables the tape to be separated into individual strips for each PROM. The areas 0000 - 003F and 0241 - 02FF are filled with FFs.

Each record is punched in exactly the order in which its START/END address was entered so that multiple records may be punched for the same PROM. When the system types "READY," turn the punch off.

4. OPERATING INSTRUCTIONS

This section provides a synopsis of operating instructions for using the GE Mark III Timesharing Service to generate, edit, assemble, simulate, and punch Signetics 2650 programs. For more detailed information on the capabilities of the GE Mark III Timesharing Service, refer to the following manuals available from General Electric's Information Services Business Division:

1) *Command System* – Mark III Foreground Reference Manual No. 3501.01J.

2) *Editing Commands* – Mark III Foreground Reference Manual No. 3400.01F.

When using high-speed terminals (120 cps and up) or in the event of any difficulty, contact your local General

Electric Sales Office. A list of General Electric Sales Offices is provided at the end of this document.

A. LOGGING IN

- Set the terminal to "LINE" mode.
- Select the half-duplex mode, using the HALF/FULL duplex switch (if necessary).
- When you hear the high-pitched tone (indicating that you have established communication with the computer), place the telephone receiver in the modem coupler.

NOTE: In the following examples data typed by the user is underlined to distinguish it from data printed by the computer.

Log in as follows

```
    H carriage return
```

Depressing the carriage return key terminates all input lines. Some General Electric personnel recommend that four Hs, HHHH, be entered instead of one. The timesharing system determines the speed of your terminal from the speed at which these characters are received

The computer will respond to your H carriage return entry with

```
    U#=
```

At this point enter your user ID (3 alphabetic characters and 5 numeric characters) and press the carriage return:

```
    U#= AAANNNNN
```

The system responds

```
    PASSWORD
    XXXXXXX
```

providing a blocked-out area in which you may enter your password. Type the password on top of the blocked-out area and press the carriage return. At this point the system may send an informative message to your terminal. Some user IDs are equipped with a short log-on sequence. If this is true, the system responds with

```
    READY
```

If this is not true, the system responds with

```
    ID:
```

This is a request for accounting information. If you do not wish to enter any accounting information, simply press the carriage return:

```
    ID: carriage return
```

The computer will respond with:

```
    SYSTEM:
```

Specify FORTRAN IV

```
    SYSTEM: FIV
```

since both the assembler and the simulator are written in FORTRAN IV. The system will respond with:

```
    NEW OR OLD
```

This is the same as the READY message. The system is now ready to perform any task you request.

B. ERROR RECOVERY

Prior to issuing any commands, it is essential to know how to delete an unwanted command.

- *Character Delete:* To delete the last character typed, hold down the shift key and depress zero (0) (ASCII decimal code 95). The ASCII decimal code is included since the actual key used may differ from terminal to terminal.
- *Line Delete:* To abort a line before the carriage return key is depressed, hold down the control key and depress "X" (ASCII decimal code 24).
- *Break:* To abort a command while it is being executed (e.g., stop printing a long file), depress the BREAK or interrupt key twice.

C. CREATING AND/OR EDITING A SOURCE FILE

Both the assembler and the simulator expect you to identify a source file that you have created. The assembler expects the 2650 program source file and the simulator expects the user's command source file. To create the source file, the name of the file must be specified:

```
    NEW FILENAME
```

This command assigns the name, FILENAME, to the temporary working file. At this point, the file is empty. Notice that the file name, FILENAME, is eight characters long. We recommend that the first four characters be meaningful. Acceptable file names are 1-to-8 characters long using only the letters A through Z, numerals 0 through 9, and the period (.).

At this point enter each line of the source file into the temporary buffer:

```
100  *PROCESSOR SYMBOLS
110  R0      EQU      0
120  R3      EQU      3
130  *PROGRAM VARIABLE STORAGE
140          ORG      H'100'
150  TLEN    EQU      3        TABLE LENGTH
160  TBLA    RES      TLEN       TABLE A
170  TBLB    RES      TLEN       TABLE B
180  *MOVE DATA IN TBLA TO TBLB. TLEN MUST
185  *BE LESS THAN 256 BYTES
```

```
190          ORG      0
200          LODI,R3  TLEN
210   LOOP   LODA,R0  TBA-1,R3
220          STRA,R0  TBB-1,R3
226          NOP
228          NOP
230          HALT
240          END
```

Note that each line starts with a line number followed by a space and then the source data itself. Lines may be entered out of order, since the system will sort the source lines by line number. Once the data is entered, this temporary file must be saved in permanent storage using the following command:

```
SAVE
```

The system responds with a READY message, and the temporary file remains intact.

To list the contents of your temporary file, type:

```
LIST
```

The system responds by printing your file.

Should you want to change your source file, bear in mind that the only file that can be modified (or edited) is the temporary working file. At this point your source program still resides in the working file; however, if your source program resided in a permanent rather than a working file, enter the following command:

```
OLD FILENAME
```

The OLD command reads the contents of the permanent file, named FILENAME, into the temporary file and assigns the name, FILENAME, to the temporary file.

The source file is now ready for editing.

To add a line, simply type the line with a new line number:

```
225 BDRR,R1 LOOP
```

To change a line, retype the line using the same line number:

```
225 BDRR,R3 LOOP
```

To change all occurrences of the letters "TB" to "TBL" from lines 210 through line 220, enter the following command:

```
CHAVC 210/TB/TBL/220
```

This command changes the following two source lines:

```
210 LODA,R0 TBLA-1,R3
220 STRA,R0 TBLB-1,R3
READY
```

Lines 226 and 228 may be deleted with either one of the following two commands:

```
EDI DEL 226-228
```

or

```
EDI DEL 226,228
```

The first command deletes lines 226 through 228, while the second command deletes lines 226 and 228.

List your temporary file and verify all changes:

```
LIST
```

The system prints your file here and then prints:

```
READY
```

Save your file in the permanent file that was created with the SAVE command:

```
REPLACE
READY
```

The SAVE command creates a permanent file with the same name as the one assigned to the temporary file. The REPLACE command takes the content of the temporary file and stores it in the already existing permanent file that has the same file name.

NOTE: Most system commands may be shortened to the first three letters; e.g., REPLACE = REP.

D. ASSEMBLING THE PROGRAM TO CREATE AN OBJECT MODULE

The editing facility assumes that each line of your source program has a line number at the beginning. Since neither the assembler nor the simulator will accept these line numbers, the following command must be executed to remove them:

```
EDI DES FILENAME
READY
```

The assembler is now ready to be executed. Enter the command:

```
/PIPHASM
```

The assembler responds with a request for the name of your source program:

```
INPUT FILENAME? FILENAME
```

The assembler then requests the name of your object module:

```
OBJECT FILENAME? FILE.OBJ
```

This is a file that the assembler generates. Your file must be assigned a name. One useful technique is to use the first four letters of the name of the source program with .OBJ concatenated onto the end.

The computer prints:

```
TYPE '0' FOR WIDE CARRIAGE or
TYPE '1' FOR NARROW CARRIAGE 1
```

If your terminal prints 120 characters per line, type '0'. If your terminal prints less than 120 characters per line, type '1'.

The assembler responds by printing your listing. When the listing is complete, the system prints:

```
READY
```

Now that your listing is complete, you may restore the line numbers to your file by entering the following command. This is only necessary if you plan to edit your file.

```
EDI RES FILENAME
```

E. LOGGING OFF

Log off the GE Timesharing System and review your program for logical and syntactical errors.

```
BYE
00024.11 CRU     0000.41 TCH 0009.74 KC
OFF AT 16:20PDT 10/15/75
```

F. USING THE SIMULATOR TO TEST AND DEBUG YOUR PROGRAM

1. Log onto the system using the procedures outlined in step A.

2. Create a file containing the simulator commands. As with the object module, you could name this file by concatenating .TST onto the first four letters of FILENAME.

```
NEW FILE.TST
READY
100 PATCH 100,01 101,02 103,03
120 DUMP A, 100 - 105
130 FEND
SAVE
```

3. Request a simulator run.

   First, you must remove the line numbers from the command file:

```
EDI DES FILE.TST
READY
REP
READY
```

Then execute the simulator by entering the following command:

```
/PIPSIM
```

The simulator responds with a request for the following information:

```
OBJECT MODULE NAME? FILE.OBJ
```

Enter the name of the object module generated by the assembler.

```
COMMAND FILE NAME? FILE.TST
```

Enter the name of the simulator command file.

The simulator prints its output at this time.

Log off the General Electric Timesharing system and review the simulator listing to determine if any program corrections are required.

```
BYE
```

G. PUNCHING A PAPER TAPE FOR DEBUGGING ON THE PROTOTYPE CARD SYSTEM

Check to see that the punch is off, and log onto the system using the procedures outlined in step A.

When the system responds with

```
READY
```

enter the command:

```
/PIPHTAP
```

PIPHTAP responds with a request for the name of your input file:

```
ENTER INPUT FILE NAME? FILE.OBJ
```

When the input file name is entered, PIPHTAP prints the following instructional message:

```
TURN ON PUNCH AND HIT CARRIAGE RETURN.
```

When the carriage return key is depressed, PIPHTAP punches 50 frames of leader and then punches your object module. The object module is also printed.

When punching is complete, the system responds with

```
READY
```

Turn the punch off, and log off the system.

H. PUNCHING A PAPER TAPE FOR BURNING PROMS

Check to see that the punch is off, and log onto the system using the procedures outlined in step A.

When the system responds with

```
READY
```

enter the command:

```
/PIPSTAP
```

PIPSTAP responds with a request for the name of your input file:

```
        ENTER OBJECT FILE NAME? FILE.OBJ
```

PIPSTAP then requests that you enter the unburned state of your PROM. (Since PIPSTAP punches data into each location of the PROM, PIPSTAP requires a value that can be used for the other locations):

```
        INITIAL PROM VALUE? 00
```

This value must be entered as two hexadecimal digits.

PIPSTAP then asks for the size of your PROM (in bytes) which must be entered in four decimal digits. The maximum allowable size is 1024.

```
        PROM SIZE? 0256
```

PIPSTAP requests both a START and an END address for the code you want punched. Use four hexadecimal digits for each address as shown below. Don't forget the leading zeros.

```
        START ADDR? 0000
        END ADDR?   000A
```

PIPSTAP will request up to eight pairs of START/END addresses. Enter a number larger than 7FFF, e.g., 8000, when you have completely described the object module:

```
        START ADDR? 8000
```

PIPSTAP prints the following message:

```
    TURN ON PUNCH AND HIT CARRIAGE RETURN
```

When you press

Carriage Return

PIPSTAP punches 50 frames of leader followed by the PROM record specified by your START and END addresses. The START address of your PROM, 0000, is punched into the tape so that it visible. Part of the object module will be printed.

When punching is complete, the system responds with:

```
        READY
```

Turn the punch off, and log off the system.

5. GENERAL ELECTRIC SALES OFFICES

CHICAGO
233 South Wacker Drive
Chicago, Illinois 60666
(312) 781-7840

DETROIT
22150 Greenfield Road
Oak Park, Michigan 48237
(313) 968-8100

MINNEAPOLIS
1500 Lilac Drive, South
Minneapolis, Minnesota 55416
(612) 546-0990

MILWAUKEE
615 East Michigan Street
Milwaukee, Wisconsin 53202
(414) 271-7900

CINCINNATI
580 Walnut Street
Cincinnati, Ohio 45202
(513) 559-3660

LOUISVILLE
Citizens Plaza
Louisville, Kentucky 40202
(502) 452-4211

INDIANAPOLIS
Castleview Building
8000 Knue Road
Indianapolis, Indiana 46250
(317) 842-0100

FT. WAYNE
Lakeside II Building
2250 Lake Avenue
Ft. Wayne, Indiana 46805
(219) 423-1406

CLEVELAND
1000 Lakeside Avenue, N.E.
Cleveland, Ohio 44114
(216) 523-6251

COLUMBUS
Harrington Building
90 E. Wilson Bridge Road
Worthington, Ohio 43085
(614) 438-2170

PITTSBURGH
Two Gateway Center
Pittsburgh, Pennsylvania 15222
(412) 566-4330

NEW YORK FINANCIAL
Mc-Graw Hill Building
1221 Avenue of the Americas
New York, New York 10020
(212) 997-0317

NEW YORK INDUSTRIAL
Mc-Graw Hill Building
1221 Avenue of the Americas
New York, New York 10020
(212) 997-0351

LONG ISLAND
1 Huntington Quadrangle
Huntington Station
L. I. New York 11746
(516) 694-7636

EAST ORANGE TELEPHONE BRANCH
33 Evergreen Place
East Orange, New Jersey 07018
(201) 672-0700

PHILADELPHIA
1700 Market Street
Philadelphia, Pennsylvania 19103
(215) 864-7474

HARRISBURG
3800 Market Street
Camp Hill, Pennsylvania 17011
(717) 761-1481

SCHENECTADY
650 Granklin Street, 3rd Floor
Schenectady, New York 12305
(518) 372-6436

PITTSFIELD
395 Main Street
Dalton, Massachusetts
(413) 494-4308

BOSTON
98 Galen Street
Watertown, Massachusetts 02172
(617) 926-2911

BUFFALO
3980 Sheridan Drive
Buffalo, New York 14226
(716) 839-5222

SYRACUSE
202 Twin Oaks Drive
Syracuse, New York 13206
(315) 456-1995

ROCHESTER
One Marine Midland Plaza
Rochester, New York 14604
(716) 232-6523

STAMFORD
2777 Summer Street
Stamford, Connecticut 05905
(203) 359-2985

HARTFORD
111 Founders Plaza
East Hartford, Ct. 06108
(203) 289-7941

LOS ANGELES NORTH
3550 Wilshire Blvd.
Los Angeles, California 90010
(213) 388-9626

SAN FRANCISCO TELCO BRANCH
One Embarcadero Center
San Francisco, California 94111
(415) 781-1155

SEATTLE
1218 Bank of California Center
Seattle, Washington 98164
(206) 575-2990

PORTLAND
2154 N. E. Broadway
Portland, Oregon 97232
(503) 288-6916

SAN FRANCISCO
One Embarcadero Center
San Francisco, California 94111
(415) 989-1100

PALO ALTO BRANCH
1120 San Antonio Road
Palo Alto, California 94303
(415) 969-3772

LOS ANGELES SOUTH
3550 Wilshire Boulevard
Los Angeles, California 90010
(213) 385-9411

ATLANTA
2200 Century Parkway, N. E.
Atlanta, Georgia 30345
(404) 325-9889

BIRMINGHAM
300 Office Park Drive
Birmingham, Alabama 35223
(205) 879-1298

NASHVILLE
293 Plus Park Boulevard
Nashville, Tennessee 37217
(615) 259-4570

CHARLOTTE
301 S. McDowel Street
Charlotte, North Carolina 28204
(704) 374-1783

GREENSBORO
604 Green Valley Road
Greensboro, N. C. 27408
(919) 292-7230

GREENVILLE
252 South Pleasantburg Drive
Greenville, S. C. 29607
(803) 233-5335

MIAMI
8410 N.W. 53rd Terrace
Miami, Florida 33166
(305) 592-7610

TAMPA
5420 Bay Center Drive
Tampa, Florida 33609
(813) 877-8294

BETHESDA
4720 Montgomery Lane
Bethesda, Maryland 20014
(301) 654-7061

BALTIMORE
25 South Charles Street
Baltimore, Maryland 21201
(301) 539-6770

RICHMOND
Willow Oaks Office Building
6767 Forest Hill Avenue
Richmond, Virginia 23235
(804) 320-0192

WASHINGTON
777 - 14th Street, N.W.
Washington, D.C. 20005
(202) 628-4000

ST. LOUIS
1015 Locust Street
St. Louis, Missouri 63101
(314) 342-7780

KANSAS CITY
911 Commerce Tower
Kansas City, Missouri 64199
(816) 842-9745

DALLAS
1341 West Mockingbird Lane
917 East Tower
Dallas, Texas 75247
(214) 631-0910

SHREVEPORT
208-A Beck Building
Shreveport, Louisiana 71102
(318) 425-2476

HOUSTON
601 Jefferson
Houston, Texas 77002
(713) 224-8294

DENVER
201 University Boulevard
Denver, Colorado 80206
(303) 320-3174

PHOENIX
3225 North Central Avenue
Phoenix, Arizona 85004
(602) 264-7881

TULSA
1900 Fourth National Bank Building
Tulsa, Oklahoma 74119
(918) 582-0800

OKLAHOMA CITY
5700 North Portland
Oklahoma City, Oklahoma 73112
(405) 947-2376

signetics

MOS
MICROPROCESSOR

ABSOLUTE OBJECT FORMAT SS51

(REVISION NO. 1)

ABSOLUTE OBJECT FORMAT SS51

2650 MICROPROCESSOR

REVISION NO. 1

APPLICATIONS MEMO

INTRODUCTION

The format for absolute code produced for the 2650 is described in this application note.

The absolute object code is formatted into blocks. The first character of every block is a colon. Inside of a block, all the characters are hexadecimal, i.e., 0 to 9 or A to F, inclusive. Only non-printing ASCII control characters may occur within an interblock gap. These are the characters in the first two columns (columns 0 and 1) of the ASCII standard code table. A CR/LF is used within the interblock gap to reset the TTY or terminal after each block.

Each block is independent. For example, paper tape can be positioned prior to any block and a load started. The loading of absolute object code will be halted by:

A BCC error on the address + count fields
A BCC error on the data field
An incorrect block length
A non-hex character within the block

The block length field contains the number of bytes of actual data which is half the number of hex characters in the data field. While the size of the data field can range from 2 to 510 characters, a standard size of 60 characters has been established so that the tape may be easily generated and read on a variety of terminals and systems. A block length of zero indicates an End of File (EOF) block. The address field of an EOF block contains the start address of the loaded program.

The Block Control Character is 8 bits formed from the actual bytes and not from the ASCII characters. The bytes are in turn exclusive or'ed to the BCC byte, and then the BCC byte is left rotated one bit. It appears as two hex characters. Both the address and count fields and the data field are followed by a BCC character pair. The BCC prevents storing data at an invalid memory address or storing bad data into memory.

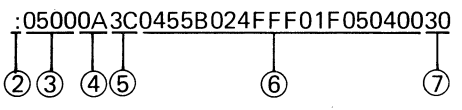
EXAMPLE: An object tape that loads ten bytes starting at location 500

```
:05000A3C0455B024FFF01F05040030
:000000
```

FORMAT

1. Interblock gap of any non-printing characters including spaces
2. Start of block character;
   a colon
3. Address field;
   four hex characters
4. Count field;
   two hex characters in range 0 to 1E
5. BCC for address and count fields;
   two hex characters
6. Data field;
   twice the value in the count field which is the number of memory locations loaded by the current block
7. BCC for the data field;
   two hex characters

EXAMPLE OF OBJECT FORMAT



2 — Start of block character (colon)
3 — Starting address for block (H'0500')
4 — Number of bytes in block (H'0A' = 10)
5 — BCC byte for fields 3 and 4 (H'3C')
6 — Data, two characters per byte
7 — BCC byte for field 6 (H'30')

signetics

MOS
MICROPROCESSOR

LOW COST CLOCK GENERATOR CIRCUITS MP52

signetics

LOW COST CLOCK GENERATOR CIRCUITS MP52

2650 MICROPROCESSOR
APPLICATIONS MEMO

GENERAL

The clock circuit requirements for microprocessors range from tightly specified, two-phase, non-overlapping types to simple single-phase, TTL compatible types. To lower system cost, the Signetics 2650 Microprocessor was designed to operate with a single-phase, TTL-level clock without any special clock driver circuitry. The clock input specifications for the 2650 are summarized in Table I.

This Applications Memo describes several clock generator circuits that may be used with the 2650. These circuits use standard TTL logic elements (7400 series). They include RC, LC, and crystal oscillator type circuits.

The stability required by the user's application will determine the type of clock generator that should be used. Tables showing the measured frequencies at several temperatures and supply voltages are presented.

RC OSCILLATOR

A circuit diagram of an RC oscillator is given in Figure 1. The first inverter is biased into its linear region by resistor R. The positive feedback capacitor (C) from node (B) to node (A) causes the circuit to oscillate. The third inverter acts as a buffer to drive the clock input of the 2650. The oscillation period is approximately equal to 3 RC. Measurements taken on this circuit showed a 10 ns rise time and a 7 ns fall time.

Table II shows how the frequency of the RC oscillator is affected by variations in V_{CC} and ambient temperature.

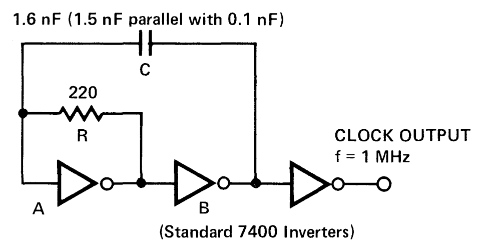

1.6 nF (1.5 nF parallel with 0.1 nF)
C
220
R
CLOCK OUTPUT
f = 1 MHz
A
B
(Standard 7400 Inverters)

FIGURE 1. RC Clock Generator

TABLE I

2650 CLOCK INPUT SPECIFICATIONS

SYMBOL	PARAMETER	TEST CONDITIONS	LIMITS MIN.	LIMITS MAX.	UNIT
I_{LI}	Input Load Current	V_{IN} = 0 to 5.25V		10	μA
V_{IL}	Input Low Voltage		-0.6	0.8	V
V_{IH}	Input High Voltage		2.2	V_{CC}	V
C_{IN}	Input Capacitance	V_{IN} = 0V		10	pF
t_{CH}	Clock High Phase		400	10,000	nsec
t_{CL}	Clock Low Phase		400	∞	nsec
t_{CP}	Clock Period		800	∞	nsec
t_r	Clock Rise Time			20	nsec
t_f	Clock Fall Time			20	nsec

Timing Reference = 1.5V
T_A = 0° to 70°C
V_{CC} = 5V ± 5%

TABLE II

RC OSCILLATOR STABILITY

Ambient Temperature (T_A)

	0°C	25°C	70°C	Stability$_T$* (V_{CC} = constant)
V_{CC} = 4.75V	1044.50 KHz	1028.95 KHz	998.50 KHz	+1.51%, -2.96%
V_{CC} = 5.0V	1043.20 KHz	1023.65 KHz	990.45 KHz	+1.91%, -3.24%
V_{CC} = 5.25V	1038.80 KHz	1013.63 KHz	979.65 KHz	+2.48%, -3.35%
Stability$_V$** (T_A = constant)	+0.12% −0.42%	+0.52% −0.98%	+0.20% −1.1%	

*Stability$_T$ with respect to T_A = 25°C
**Stability$_V$ with respect to V_{CC} = 5.0V

A second type of RC oscillator uses a monostable multivibrator circuit (N74123) as illustrated in Figure 2. The pulse width of each monostable is determined by the external resistor and capacitor:

$$t_W = \left(0.28\right)\left(R_{ext}\right)\left(C_{ext}\right)\left(1 + \frac{0.7}{R_{ext}}\right)$$

where

R_{ext} is in KΩ

C_{ext} is in pF,

and

t_W is in ns.

In this circuit, the oscillation is caused by the triggering of each monostable by the other one. The oscillation frequency can be derived from the following equation:

$$fosc = \frac{1}{t_{w1} + t_{w2},}$$

where:

t_{w1} is the pulse width of the first monostable, and
t_{w2} is the pulse width of the second monostable.

Measurements on frequency stability with a load of one TTL input are presented in Table III.

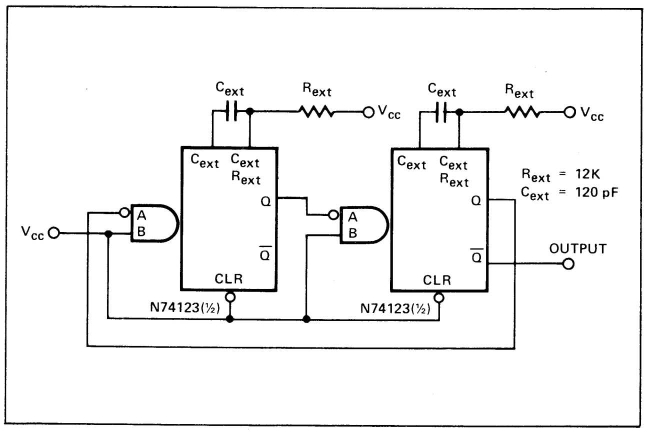


FIGURE 2. RC Clock Generator with Monostable Circuit N74123

TABLE III

MONOSTABLE MULTIVIBRATOR OSCILLATOR STABILITY

Ambient Temperature (T_A)

	0°C	25°C	70°C	Stability$_T$* (V_{CC} = constant)
V_{CC} = 4.75V	1063.65 KHz	1046.72 KHz	1041.16 KHz	+1.62%, -0.53%
V_{CC} = 5.0V	1063.80 KHz	1042.83 KHz	1032.63 KHz	+2.01%, -0.98%
V_{CC} = 5.25V	1063.80 KHz	1039.95 KHz	1024.02 KHz	+2.29%, -1.53%
Stability$_V$** (T_A = constant)	+0.00% −0.014%	+0.276% −0.373%	+0.826% −0.833%	

*Stability$_T$ with respect to T_A = 25°C
**Stability$_V$ with respect to V_{CC} = 5.0V

The observed rise and fall times at the output of this circuit were 10 ns and 8 ns, respectively. The stability of this circuit reflected a slight improvement over the stability of the RC oscillator shown in Figure 1.

LC OSCILLATOR

Figure 3 shows an LC oscillator circuit using standard TTL inverters.

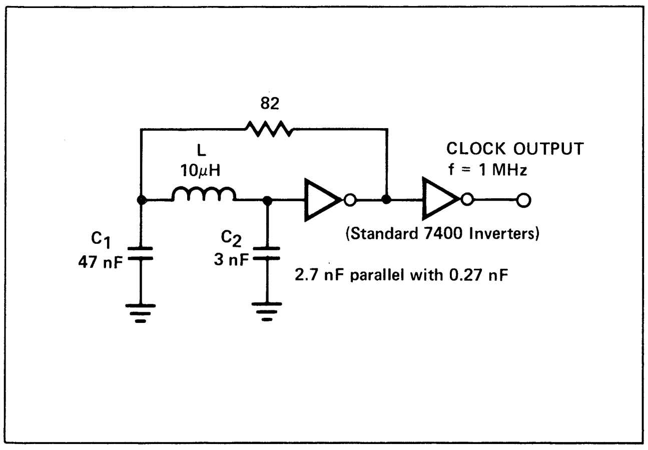


FIGURE 3. LC Clock Generator

The first inverter combined with the passive components forms a Colpitts oscillator. The resistor provides a feedback path for the first inverter and forces it into its linear region.

The second inverter "squares" the oscillator signal and provides an output buffer. The oscillator frequency can be derived from the following equation:

$$f_{osc} = \frac{1}{2\pi\sqrt{(L)\left[\frac{(C1)\cdot(C2)}{(C1)+(C2)}\right]}}$$

Measurements from the circuit in Figure 3 showed a 10 ns rise time and a 7 ns fall time. Measurements on frequency stability are provided in Table IV.

CRYSTAL OSCILLATORS

In 2650 Microprocessor applications requiring a highly stable clock, a crystal oscillator may be required. Some examples of crystal oscillator circuits are shown in Figures 4 and 5. The circuit shown in Figure 4 uses a 1.025 MHz crystal while the circuit shown in Figure 5 uses a low cost 4.433618 MHz crystal commonly found in European manufactured color TV sets. The output of the oscillator is divided by four to obtain a clock frequency of 1.1 MHz.

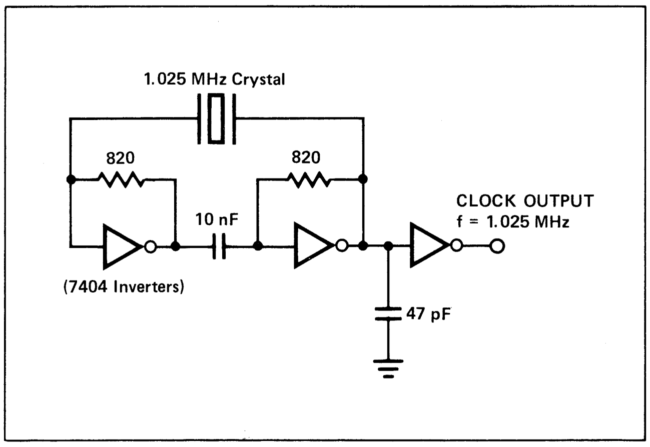


FIGURE 4. Clock Generator Using a Non-TV Standard Crystal

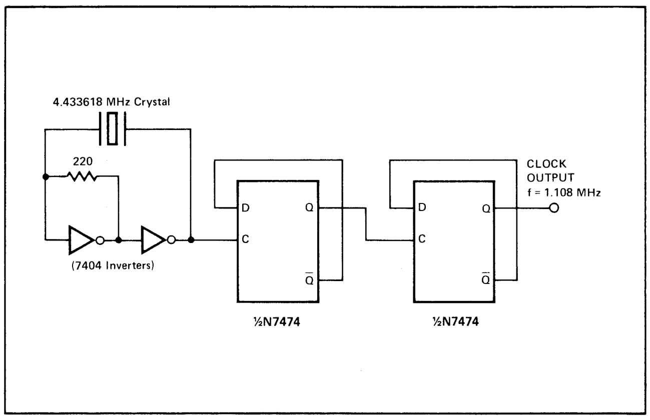


FIGURE 5. Low Cost Color TV Crystal Clock Generator

TABLE IV
LC OSCILLATOR STABILITY
Ambient Temperature (T_A)

	0°C	25°C	70°C	Stability* (V_{CC} = constant)
V_{CC} = 4.75V	1027.14 KHz	1017.75 KHz	1004.46 KHz	+0.92%, -1.31%
V_{CC} = 5.0V	1026.62 KHz	1016.99 KHz	1004.11 KHz	+0.95%, -1.26%
V_{CC} = 5.25V	1025.82 KHz	1016.30 KHz	1003.73 KHz	+0.94%, -1.24%
Stability** (T_A = constant)	+0.05% -0.08%	+0.07% -0.07%	+0.03% -0.04%	

*Stability$_T$ with respect to T_A = 25°C
**Stability$_V$ with respect to V_{CC} = 5.0V

The circuit of Figure 5 can also be used with a 3.5795 MHz United States color TV crystal to provide an output frequency of 895 KHz.

The stability of the crystal oscillator circuits is mainly determined by the stability of the crystal used. The circuits shown in Figures 4 and 5 had a stability of 0.003% over the 0°C to 70°C temperature range and 0.002% over a variation of power supply voltage from 4.75V to 5.25V.

SUMMARY

Table V is a summary of the stability measurements made for the oscillator circuits described in this application note. As the table shows, the crystal circuits exhibit great stability relative to the RC and LC oscillators, but they suffer the added expense of the crystal. Any of the oscillator circuits shown in this application note can be used to drive the 2650 microprocessor clock input.

TABLE V
SUMMARY OF OSCILLATOR STABILITY

CIRCUIT TYPE	STABILITY					
	(4.75V to 5.25V)			(0°C to 70°C)		
	0°C	25°C	70°C	4.75V	5.0V	5.25V
RC	+0.12% -0.42%	+0.52% -0.98%	+0.2% -1.1%	+1.51% -2.96%	+1.91% -3.24%	+2.48% -3.35%
RC MONO-STABLE	+0.00% -0.014%	+0.276% -0.373%	+0.826% -0.833%	+1.62% -0.53%	+2.01% -0.98%	+2.29% -1.53%
LC	+0.05% -0.08%	+0.07% -0.07%	+0.03% -0.04%	+0.92% -1.31%	+0.95% -1.26%	+0.94% -1.24%
CRYSTAL	+0.0003%	-0.0001%	+0.0002%	+0.001%	±0.0001%	+0.0004%

Signetics 2650 Microprocessor application memos currently available:

AS50	Serial Input/Output
AS51	Bit and Byte Testing Procedures
AS52	General Delay Routines
AS53	Binary Arithmetic Routines
AS54	Conversion Routines
SP50	2650 Evaluation Printed Circuit Board Level System (PC1001)
SP51	2650 Demo Systems
SP52	Support Software for use with the NCSS Timesharing System
SP53	Simulator, Version 1.2
SP54	Support Software for use with the General Electric Mark III Timesharing System
SS50	PIPBUG
SS51	Absolute Object Format (Revision 1)
MP51	2650 Initialization
MP52	Low Cost Clock Generator Circuits