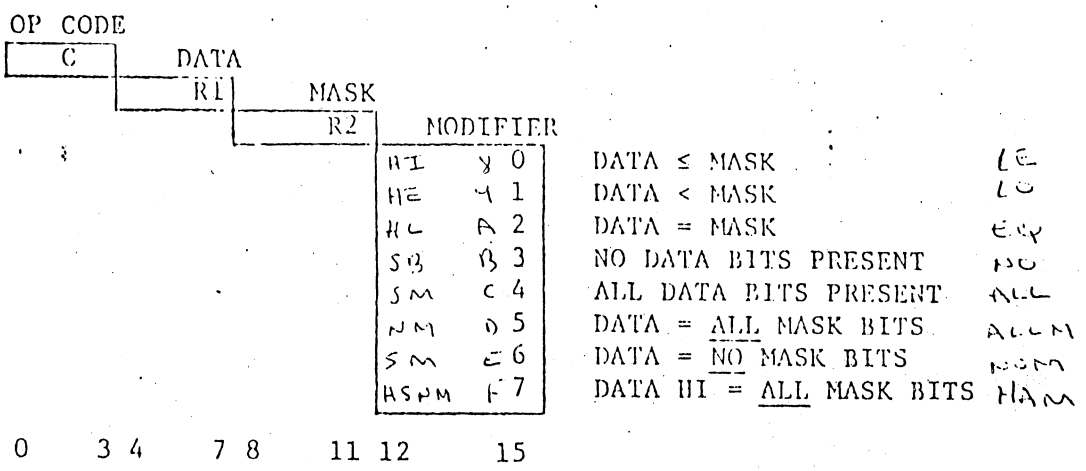
2.0 MICRO-INSTRUCTION SET

The micro-instructions are designed to support a wide range of applications including Input/Output control, Desk Calculator Functions, Remote Work Stations, Interactive Terminal Operations, or a completely selfcontained General Purpose Computer. The micro-instructions selected for PALM are composed of sixteen bit control words, grouped into six basic op code classifications. In general, they support instruction/ operand fetch and store, arithmetic and logical operations, test under mask and jump, bit manipulation, input/output control, and data transfer. Figure 11 is a summary of the microinstruction set.

A detailed description of each group follows.

2.1 JUMP



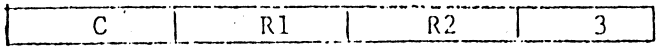
NOTE: Values of 8-F for bits 12-15 cause the corresponding test to be performed for a jump on false condition.

By convention, register #0 of each group of sixteen registers is reserved as the micro-instruction address register. Register #1 is similarily reserved as the link register. Each time a jump instruction is encountered, an address of the jump instruction plus four is computed. AND STORED IN #1 OF PALM.

If, as a result of the conditional test, the Jump is not taken, the address of Jump + 4 is placed in the link register (R1). In this case, it is assumed that the next sequential instruction will be a Load R0 resulting in an unconditional branch to a subroutine. If the last instruction of the subroutine is a MOVE R0, R1, control will be returned to the beginning of the next instruction following the main line unconditional branch. If conditional branching occurs within the subroutine, it is the responsibility of the programmer to save the link address before issuing the conditional branch or jump instruction. If the mainline jump is taken, the updated address of Jump + 4 replaces the contents of register #0, with processing continuing from this point on. By way of illustration, consider the following example.

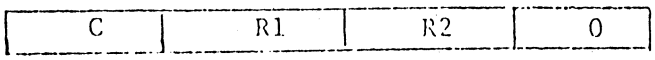
```
                200  SUB R2,R3        SUBTRACT R3 FROM R2
                202  EHLT R4,XX       PLACE MASK IN R4 LO
IAR             204  JEQ R3,R4        COMPARE R3 & R4 & JUMP IF EQUAL
                206  LDHD 0,XX        LOAD UIAR with 0400 & BRANCH
IAR+4           208  ADD R2,R3        ADD R3 to R2
                210
                212
                214
                216
                400  LDHD R4,XX       LOAD R4 FROM 00XX
                402
                404
                406
                408
                410  MOVE R0,R1       RETURN TO MAINLINE PROGRAM
```

2.1.1 JUMP NO DATA BITS PRESENT (JNO R1)



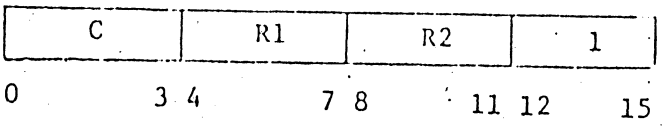
The low order byte of the register specified by the R1 field is tested for a zero condition. If the result is zero and no bits are equal to '1', the next sequential instruction is bypassed. The mask does not participate in this operation since the test is performed by a forced ALU condition. This eliminates the requirement to load a mask prior to execution of this instruction. The data byte is not altered as a result of instruction execution.

2.1.2 JUMP DATA ≤ MASK (JLE R1,R2)



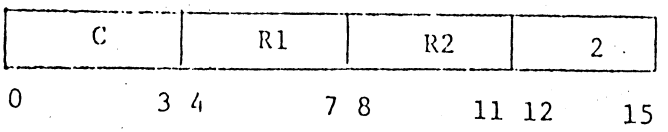
The low order byte of the register specified by the R1 field is logically compared with the low order mask byte denoted by the R2 field. If the data byte (R1) is less than or equal to the mask byte, the next sequential instruction is bypassed. Execution of this instruction assumes that a valid mask has been loaded previously. Neither the mask nor data byte are altered as a result of instruction execution. Specifying a flase condition for this instruction is equivalent to a 'JUMP DATA > MASK' instruction.

2.1.3 JUMP DATA < MASK (JLO R1,R2)



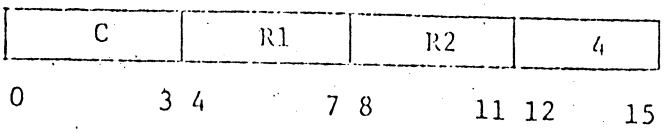
The low order byte of the register specified by the R1 field is logically compared with the low order mask byte denoted by the R2 field. If the data byte (R1) is less than the mask byte (R2), the next sequential instruction is skipped, and the instruction address is incremented to the Jump Address + 4. The data and mask bytes are not altered as a result of instruction execution.

2.1.4 JUMP DATA = MASK (JEQ R1,R2)



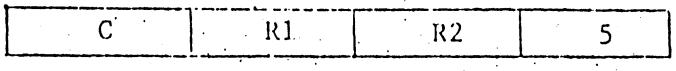
The data and mask bytes are logically compared and tested for an equal condition. If an equal condition is found, the next sequential instruction is skipped and the instruction address is incremented to the address of Jump + 4. Neither operand is altered as a result of instruction execution.

2.1.5 JUMP ALL DATA BITS PRESENT (JALL R1)



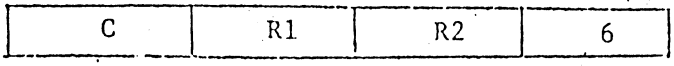
The low order byte of the register specified by R1 is tested for an 'ALL ONES' condition. This test, like the 'NO BITS PRESENT' test, is a forced condition and does not require that a mask be established prior to execution of the instruction. If the data bits are equal to FF, the next sequential instruction is skipped with execution continuing at the address of Jump + 4. If the data is not equal to FF, the updated address is automatically placed in R1 and the next sequential instruction following the jump instruction is executed. The data byte is not altered by this instruction.

2.1.6 JUMP DATA = ALL MASK BITS (JALLM R1,R2)

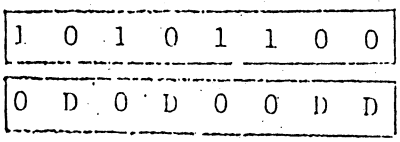


The low order data byte specified by R1 is compared with the mask byte to determine whether the corresponding positions of the data byte contain a binary value of '1'. In order that the test be successful, each position of the mask byte which contains a '1' must have a corresponding position within the data byte also equal to '1'. Zero positions in the mask byte represent a don't care condition. If the test is successful, the updated instruction address (Jump + 4) is used as the address of the next logical instruction.

2.1.7 JUMP DATA = NO MASK BITS (JNOM R1,R2)

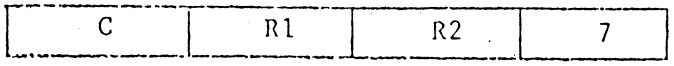


This test is similar to the 'DATA = ALL MASK BITS' except in this case the corresponding positions of the data byte must contain a binary value of '0'. The test is successful if for each position of the mask byte which contains a '1', the corresponding data byte position is equal to '0'. Mask byte positions which contain zeros are don't care conditions. As an example:



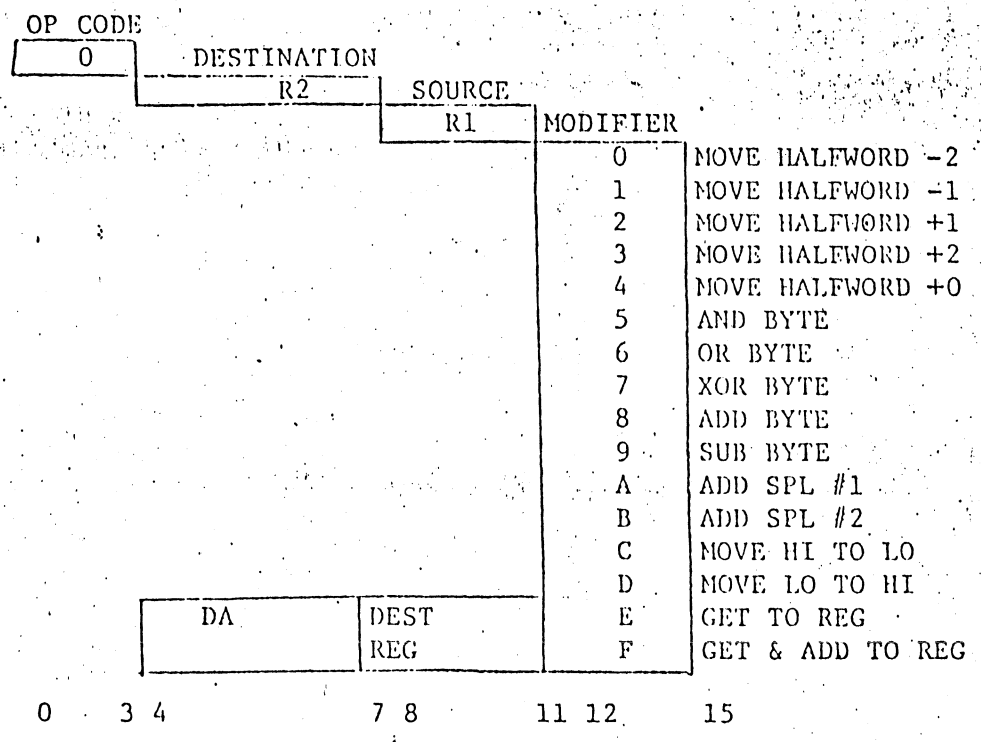
TEST SUCCESSFUL

2.1.8 JUMP DATA HI = ALL MASK BITS (JHAM R1,R2)



Execution of this instruction is identical to that described in section 2.1.6 except that the data byte involved in the operation is the HI order byte of the register specified by the R1 field. If each of the hi order bits are equal to their corresponding mask '1' bits, then a jump of the next instruction is executed.

2.2 ARITHMETIC/LOGICAL OPERATION



These instructions are used to perform arithmetic and logical operations on data contained in the source and destination registers. In general, the R1 field specifies one of sixteen registers in which the low order byte is selected as the source operand. Exceptions to this rule occur for the 'MOVE', 'GET TO REGISTER', and 'GET AND ADD TO REGISTER' instructions. The R2 field denotes a halfword destination register, which contains an operand prior to execution, and the result of the operation at the end of instruction execution. The following subsections describe each operation in detail.

2.2.1 MOVE HALFWORD

0	R2	R1	
			0 SUBTRACT 2
			1 SUBTRACT 1
			2 ADD 1
			3 ADD 2
			4 NO MODIFICATION

This instruction provides a means for incrementing and decrementing counters, creating multiple copies of operands, and effecting a subroutine return. The contents (16 BITS) of the register specified by the R1 field replaces the contents of the register denoted by the R2 field. By proper selection of the modifier, the source operand (R1) can be moved and incremented or decremented before insertion in the destination register. The source operand is unaltered by this instruction. If the source and destination operands are one and the same, and no incrementing or decrementing is performed, the instruction functions as a no-op.

Corresponding mnemonics are as follows:

(MVM2 R2,R1)
(MVM1 R2,R1)
(MVP1 R2,R1)
(MVP2 R2,R1)
(MOVE R2,R1)

2.2.2 AND BYTE (AND R2,R1)

0	R2	R1	5

0 3 4 7 8 11 12 15

The low order byte of the source register (R1) is logically 'ANDED' with the low order byte of the destination register (R2). The result replaces the low order byte of the destination register while the high order byte remains uneffected. The source operand is not altered by this operation.

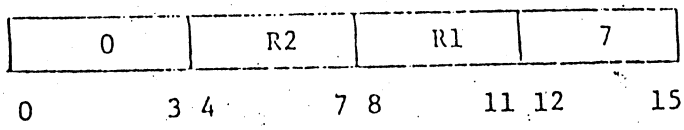
2.2.3 OR BYTE (ORB R2,R1)

0	R2	R1	6

0 3 4 7 8 11 12 15

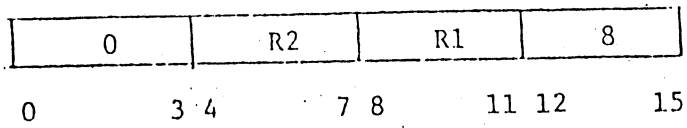
The low order byte of the source register (R1) is logically 'ORED' with the low order byte of the destination register (R2). The result replaces the low order byte of the destination register while the high order byte of the destination register and the source operand are unaltered by this operation.

2.2.4 EXCLUSIVE OR (XOR R2,R1)



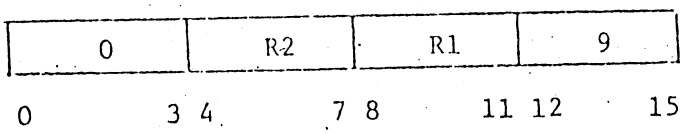
The low order byte of the source register (R1) is 'EXCLUSIVELY ORED' with the low order byte of the destination register. The result replaces the low order byte of the destination register while the high order byte of the destination register and the source operand are unaltered by this operation.

2.2.5 ADD (ADD R2,R1)



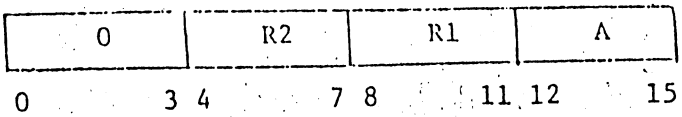
The low order byte of the source register (R1) is logically added to the low order byte of the destination register (R2). Resulting carries from the addition are propagated into the high order byte of the destination register (R2 HI). If the original value of the high order destination byte is equal to zero, a resulting carry will be trapped alone at the end of instruction execution.

2.2.6 SUBTRACT (SUB R2,R1)



The low order byte of the source register (R1) is logically subtracted from the low order byte of the destination register (R2). If a borrow occurs, it is a signal to decrement the high order byte of the destination register by one. In order to perform variable length operand subtraction, the 'ADD SPECIAL #2' instruction is used to propagate borrows as successive bytes are subtracted. This will be more fully explained in Section 2.2.8. The sign of the result can be determined by executing an 'ADD SPECIAL #1' to a destination register containing zero, emitting FF to a mask register, and issuing a 'JUMP EQUAL' instruction. If the result is negative, the high order byte of the destination register should contain the value FF. For address arithmetic, this instruction can be used to subtract an eight bit unsigned binary quantity from a sixteen bit unsigned quantity.

2.2.7 ADD SPECIAL #1 (ADDS1, R2,R1)

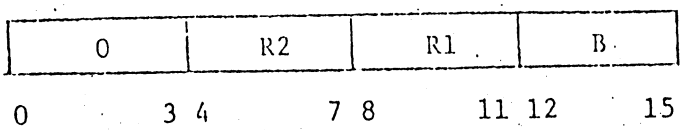


The primary purpose of this instruction is to provide a means for adding together two variable length numeric fields. The instruction allows the carry from a previous 'ADD' operation to be used as input to a current partial sum. The low order byte of the source register (R1) is added to the high order byte of the destination register and the result is placed in the low order byte of the destination register (R2). It is assumed that the high order byte of the destination register originally contains a carry, if any, from a normal 'ADD' operation. If an additional carry is generated as a result of adding the previous carry, it will be propagated into the high order byte of the destination register. Combining this instruction and the normal 'ADD' and 'LOAD BYTE INDIRECT' instructions provides a means for positioning individual bits or hexadecimal digits within a particular byte. As an example:

```
LDBI    R2,R3,0   FETCH AN OPERAND AND PLACE IN R2 LO
ADD     R2,R2     SHIFT LEFT ONE POSITION
ADDS1   R1,R2     ADD SPILL BIT TO LO BYTE OF R2
```

This sequence is equivalent to a shift left and rotate.

2.2.8 ADD SPECIAL #2 (ADDS2 R2,R1)



This instruction is used to propagate 'BORROWS' during field subtraction operations. The low order byte of the source register (R1) is added to the high order byte of the destination register (R2) and the result is placed in the low order byte of the destination register. The destination high order byte is cleared and a 'NO CARRY' condition causes a 'BORROW' to propagate into it, ie FF.

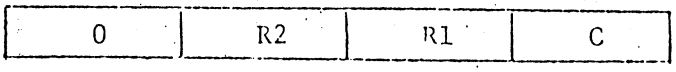
As an example -- Subtract Field #1 from Field #2:

FIELD #1		OP 7	OP 5	OP 3	OP 1
FIELD #2		OP 8	OP 6	OP 4	OP 2
ORIGINAL SOURCE	S	HI 0 0	LO OP 1	SUBTRACT OP1,OP2	
ORIGINAL DESTINATION	D	HI 0 0	LO OP 2		
RESULT	D	BORROW	OP2-OP1	STORE PARTIAL RESULT	
NEW SOURCE OPERAND	S	0 0	OP 3		
				ADDS2	
RESULT	D	X X	OP 3 + BORROW		

00 If carry from destination Lo
FF If no carry from destination Lo

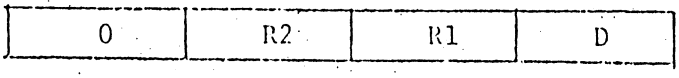
NOTE: Before executing an 'ADD SPECIAL #2' instruction, the program should test the high order byte to see if a borrow actually did occur as a result of a previous subtract operation. If a borrow had not occurred, and an 'ADD SPECIAL #2' is executed, a borrow will propagate into the high order byte of the destination register. Subsequent execution of this instruction can result in the Addition/Subtraction of an erroneous borrow.

2.2.9 HI TO LO (HTL R2,R1)



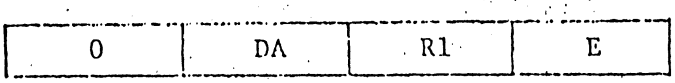
The high order byte of the source register (R1) replaces the low order byte of the destination register (R2). The high order bytes of the source and destination registers are not altered as a result of instruction execution.

2.2.10 LO TO HI (LTH R2, R1)



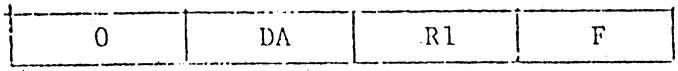
The low order byte of the source register (R1) replaces the high order byte of the destination register (R2). The low order bytes of the source and destination registers are not altered as a result of instruction execution.

2.2.11 GET TO REGISTER (GETR DA,R1)



Execution of this instruction is used to transfer a byte of data from an I/O device, whose address is specified by the DA field, to the low order position of the register specified by the R1 field. The high order byte of this register is not altered as a result of instruction execution. In addition, the low order byte of the destination register is placed on bus out during instruction execution. The TAG line will be at a down level during this instruction.

2.2.12 GET TO REGISTER AND ADD (GETA DA,R1)



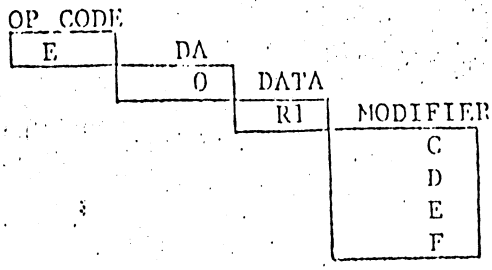
This instruction causes a three bit quantity, derived from the information placed on BUS IN by the specified device(s) (DA), to be added to the contents of the destination register (R1). The quantity to be added is derived from the following chart:

BUS BIT:	0	1	2	3	4	5	6	7	Quantity Added
	1	1	1	1	1	1	1	x	0
	1	1	1	1	1	1	0	x	2
	1	1	1	1	1	0	x	x	4
	1	1	1	1	0	x	x	x	6
	1	1	1	0	x	x	x	x	8
	1	1	0	x	x	x	x	x	A
	1	0	x	x	x	x	x	x	C
	0	x	x	x	x	x	x	x	E

The TAG line will be at a down level during this instruction. Bus In parity is not checked during this instruction.

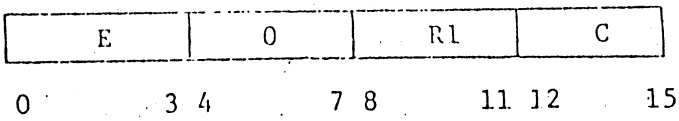
If the destination register (R1) is specified to be Register #0 (IAR), the result of the instruction is an eight-way priority branch, controlled by the specified device(s)(DA).

2.2.13 SHIFT & ROTATE



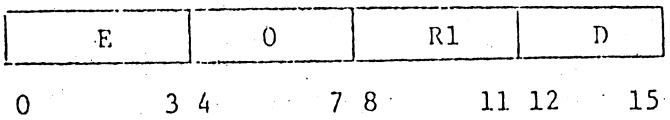
A shift right and rotate function is performed when the device address of a 'GET BYTE' instruction is equal to zero and modifiers C,D,E or F are specified. If the device address is zero, and modifiers other than C,D,E or F are used, the instruction executes as a normal GET BYTE. Due to the deliberate selection of shift quantities, any combination of right or left byte shifts with '0' or '1' padding can be accomplished with three micro-instructions or less.

2.2.13.1 SHIFT RIGHT 1 (SHFTR,R1)



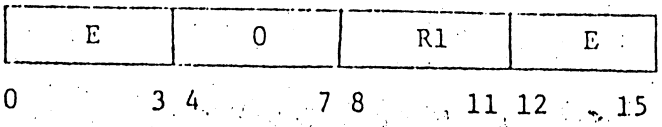
The low order byte of the register specified by the R1 field is shifted to the right one position and the vacated high order bit of the low order byte is padded with a value equal to the low order bit of the high order byte. The low order bit of the register is dropped each time a shift right instruction is executed. The reason for padding in the manner described is to help facilitate sixteen bit shift operations. The high order byte of the register is not affected by execution of a shift instruction.

2.2.13.2 SHIFT RIGHT & ROTATE 1 (ROTR,R1)



The low order byte of the register specified by the R1 field is shifted right one position. The low order bit replaces the high order bit of the byte shifted. The high order byte of the register is not affected by execution of the shift and rotate instruction.

2.2.13.3 SHIFT RIGHT & ROTATE 3 (SRR3,R1)



The low order byte of the register specified by the R1 field is shifted right three positions with the corresponding spill bits replacing the three high order bits of the byte shifted.

EXAMPLE

BEFORE EXECUTION

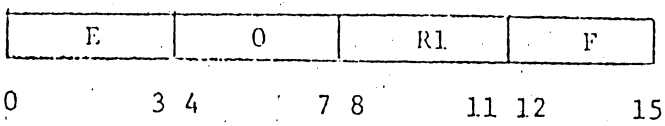
XXXXXXXX11010001

AFTER EXECUTION

XXXXXXXX00111010

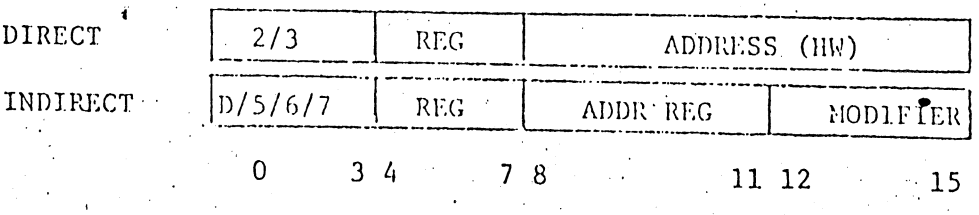
The high order byte of the register remains unchanged.

2.2.13.4 SHIFT RIGHT & ROTATE 4 (SRR4,R1)



This instruction executes in the same manner as that described for the SHIFT RIGHT & ROTATE 3 except that the degree of rotation is four rather than three. The high order byte remains unchanged.

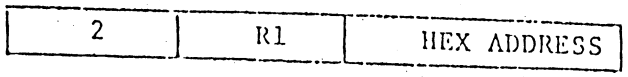
2.3 STORAGE OPERATIONS



Storage operations are classified as being direct or indirect. Direct fetch and store instructions derive the operand address from the low order byte of the micro-instruction. This eight bit address is used to fetch or store sixteen bit operands within the first 256 halfwords storage locations.

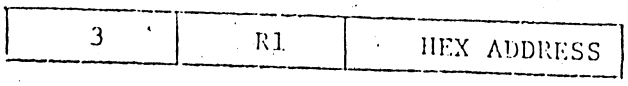
Indirect instructions are used to fetch and store either bytes or halfwords anywhere within physical storage. Operand addresses are indirect and are referenced by specifying one of sixteen halfword registers, which in turn contains the operand address. The individual storage operations are described as follows:

2.3.1 DIRECT HALFWORD FETCH (LDHD R1,##) or (LDHD R1,$A)



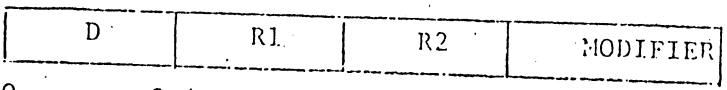
This instruction is used to fetch a sixteen bit quantity from the storage location defined by bits 8-15 and place it in one of sixteen halfword registers as defined by the R1 field. This operation replaces the original contents of the destination register. Storage addresses are on even halfword boundaries.

2.3.2 DIRECT HALFWORD STORE (STHD R1,##) or (STHD R1,$A)



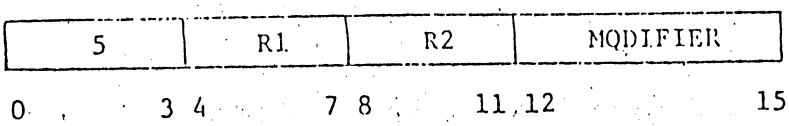
This instruction is used to store a sixteen bit quantity in the storage location defined by bits 8-15. The operand to be stored is contained in one of sixteen registers denoted by the R1 field. Storage addresses are located on halfword boundaries.

2.3.3 INDIRECT HALFWORD FETCH (LDHI R1,R2,#)



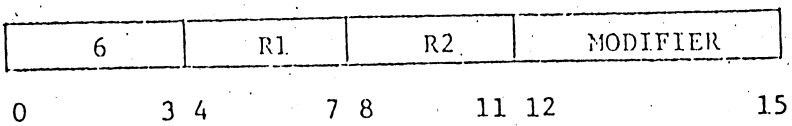
The halfword operand, located at an address specified by the contents of the R2 register, is fetched from storage and placed in the register denoted by the R1 field. The indirect addresses are located on halfword boundaries. The modifier field is used to increment or decrement the indirect address after the fetch operation has been performed. Modifier value 0-3 add the corresponding numeric values of 1-4 to the indirect address, while values 4-7 subtract the corresponding numeric values of 1-4 from the indirect address. Values > 7 perform no address modification. The updated indirect address is then written back to the register specified by the R2 field. If R1 and R2 specify the same register, the modifier has no affect, and the fetched data replaces the previous register content.

2.3.4 INDIRECT HALFWORD STORE (STHI R1,R2,#)



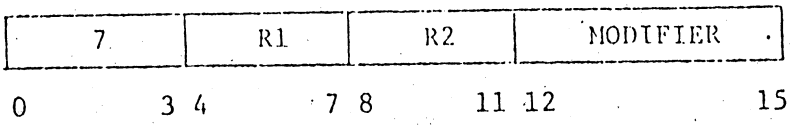
Execution of this instruction is basically the same as that of the 'INDIRECT HALFWORD FETCH'. In this case, however, the sixteen bit quantity contained by register R1 is stored at an indirect address specified by the contents of the register denoted as R2. Address modification is the same as that described in Section 2.3.3.

2.3.5 INDIRECT BYTE FETCH (LDBI R1,R2,#)



This instruction is used to fetch a byte from storage and place it in the low order byte position of the register specified by the R1 field. The high order byte of this register is automatically set to zero for a fetch byte operation. Indirect addresses are not limited to halfword boundaries and can therefore be used to address any individual byte within physical storage. Modification of the indirect address is the same as previously discussed.

2.3.6 INDIRECT BYTE STORE (SIBI R1,R2,#)



This instruction is used to store a byte of data at a location specified by the contents of the register denoted by R2. The byte to be stored is taken from the low order position of the register specified by the R1 field. The high and low order bytes of this register are not altered as a result of instruction execution. Storage addresses can be specified for any individual byte within physical storage. Address modification is performed in the same manner as previously described.

2.4 BIT MANIPULATION

These instructions are designed to provide a convenient means for setting or clearing individual data bits, emitting masks or constants from the program stream and effecting a one step emit/add operation.

2.4.1 EMIT (EMIT R1,##) or (EMIT R1,$A)

8	R1	DATA OR MASK

0 3 4 7 8 15

The Emit instruction provides a means for generating masks to be used by the 'JUMP' instruction, or emitting an eight bit binary quantity to the low order position of the register specified by the R1 field. The high order byte of the destination register is unaffected by this operation.

2.4.2 CLEAR BIT (CLRI R1,##)

9	R1	BIT MASK

0 3 4 7 8 15

R1 ← R1 ∧ ~MASK

Bit 8-15 of this instruction specify particular bit positions, within the low order byte of the register specified by R1, which are to be cleared or set to zero. Each bit position of the mask which contains a '1' will clear the corresponding bit position in the low order byte of the destination register. Mask bit positions which contain a '0' do not alter the corresponding destination register bit positions. The high order byte of the destination register (R1) is not altered by this instruction.

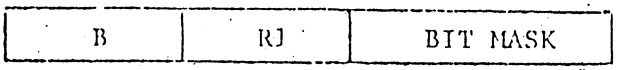
2.4.3 ADD IMMEDIATE + 1 (ADDI R1,##) or (ADDI R1,$A)

A	R1	DATA BYTE

0 3 4 7 8 15

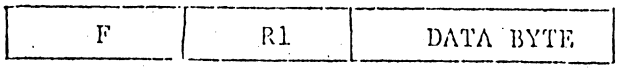
This instruction adds the eight bit quantity specified by bits 8-15 to the low order byte of the register specified by the R1 field. A carry, which may occur as a result of the addition, will be propagated into the high order byte of the destination register. Since the instruction is an 'ADD IMMEDIATE + 1', specifying 00 for bits 8-15 will cause a '1' to be added to the low order byte of the destination register. If no carry results from the addition, the high order byte of the destination register will not be altered.

2.4.4 SET BIT (SETI R1,##)



This instruction is similar to the clear bit instruction, except in this case mask bits which contain '1' set the corresponding bits of the low order destination register to a value of '1'. Mask bits containing '0' do not alter the corresponding bit positions of the destination register. The high order byte of the destination register is not altered by this operation.

2.4.5 SUBTRACT IMMEDIATE MINUS 1 (SUBI R1,##) or (SUBI R1,$A)

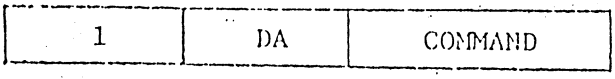


This instruction subtracts the eight bit quantity specified by bits 8-15 from the low order byte of the register denoted by the R1 field. A 'BORROW', which may occur as a result of the subtraction, will be propagated into the high order byte of the destination register. Since the instruction is a 'SUBTRACT IMMEDIATE MINUS 1', specifying 00 for bits 8-15 will cause a '1' to be subtracted from the low order byte of the destination register. If a 'BORROW' does not occur as a result of the subtraction, the high order byte of the destination register will not be altered.

2.5 CONTROL OPERATIONS

Control instructions are used to transmit command information to the various system input/output devices. Commands can be used to reset attachments, request device status, or cause the device to take some mechanical action such as rewinding a tape, positioning a print head, or causing a file to seek.

2.5.1 CONTROL (CTL DA,##)



Bits 8-15 of this instruction are transferred to the device whose address is specified by the DA field. Device commands may have bit positional significance, or may be binarily encoded for certain units.

Bus Out Bit

0	1	2	3	4	5	6	7
Processor Error RESET	Not Bit = Block Prog Lvl Switch	Not Bit = Allow Prog Lvl Switch	Note 1 Not Bit = Turn Display OFF	Note 1 Not Bit = Turn Display & Select Frame ON	Not Bit = Reset IPL, and Switch Select Ros Mode switch	Note 1 Frame Bit #2	Note 1 Frame Bit #1
						See Display Descritpion	

The individual bits are independent of each other and can be used in any logical combination required. The Bus Out bits numbered 0-7 are equivalent to the control command bits 8-15. The complement condition of the bits described are don't care situations. In general, it is recommended that a control command with a device address equal to Hex F be used as a general system reset recognized by each attachment.

NOTE 1: Valid only when PALM is connected to Video Display.

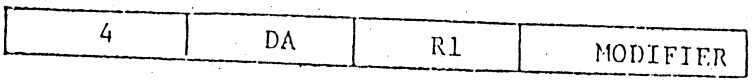
CONTROL DA=0 BIT ASSIGNMENTS

Control commands with device addresses between 1 and F are general purpose and are left to the user to define for his particular application. Device address 0 has pre-assigned functions which block CPU interrupts, reset error conditions, enable and disable display units (if used), resets the CPU IPL trigger, and toggles the select ROS enable switch. A summary of their respective functions is illustrated by a subsequent chart.

2.6 I/O DATA TRANSFER

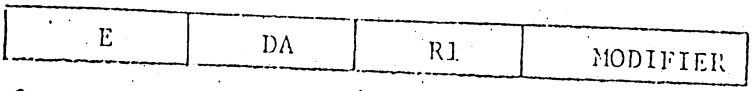
Two instructions are used to transfer data between main storage and the various input/output devices. A single byte of data is transferred for each I/O instruction executed.

2.6.1 PUT BYTE (PUTB DA,R1,#)



The 'PUT BYTE' instruction is used to transfer a byte of data from main storage to an I/O device whose address is specified by bits 4-7 of the micro-instruction. The address of the data to be transferred is specified by the contents of the register defined by bits 8-11. This indirect address can be modified in the same manner as discussed in Section 2.3.3. Main storage is not altered as a result of instruction execution.

2.6.2 GET BYTE (GETB DA,R1,#)



This instruction is used to transfer a byte of data from an I/O device, whose address is specified by bits 4-7, to main storage. The byte is placed in main storage at an address specified by the contents of the register defined by bits 8-11. The indirect storage address can be modified in the same manner as discussed in Section 2.3.3. The low order byte of the indirect address register is placed on bus out during instruction execution. The TAG line will be at a up level during this instruction.

The 'GET BYTE' instruction functions as described only for device addresses greater than zero and with modifiers less than 'C'. If the device address is not zero, and modifiers C,D,E or F are used, the instruction executes like a 'GET TO REGISTER' instruction. In this case, the R1 field actually specifies the Register into which the data will be placed. Data will be placed into the low order byte of the selected register.

Execution of a 'GET BYTE' instruction with a device address of zero and modifiers greater than 'B' will perform an entirely different function as described in Section 2.2.13.

2.7 Instruction Execution Times

See Figure 11.

Figure 11 assumes a storage cycle of 529.6ns and no stolen cycles. Execution times will be slightly larger when micro-instructions are executed out of FSU ROS.