DOCNAME: PALM-2

DOC ID: PALM-2

OPERATOR: 015

PRINT QUEUE: ibmcpq

PRINT COMMAND: pj

TIME/DATE: 14:23 03/21/72

IBM CONFIDENTIAL

Document Number 72-28C-002 March 16, 1972

Document Title: PALM MICROINSTRUCTION SET

FUNCTIONAL SPECIFICATIONS

IBM CONFIDENTIAL

G. F. Nielsen

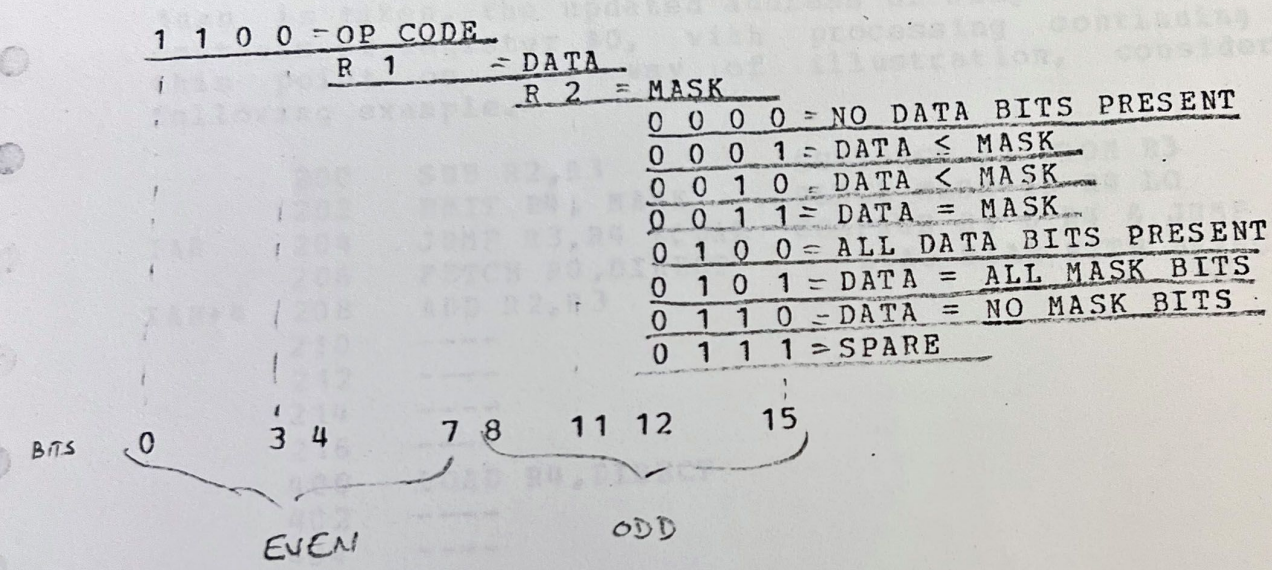
R. E. Abernathy

3.0 MICROINSTRUCTION SET

This section describes the operation of a general purpose microgragrammed input/output controller. Microprograms are designed to implement a subset of the System/3 BASIC programming language, as discussed in section 5.0. The microinstructions 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 #7 is a summary of the microinstruction set.

Each group is described in detail in a subsequent section of this document.

3.1 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 microinstruction 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. 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 Ro resulting in an unconditional branch to a subroutine. If the last instruction of the subroutine is a MOVE R1,R0, 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 R2 FROM R3
        202   EMIT R4, MASK          PLACE MASK IN R4 LO
IAR     204   JUMP R3,R4 EQUAL       COMPARE R3 & R4 & JUMP IF EQUAL
        206   FETCH R0,DIRECT           LOAD UIAR WITH BRANCH TO ADDRESS
IAR+4   208   ADD R2,R3
        210   ----
        212   ----
        214   ----
        216   ----
        400   LOAD R4,DIRECT
        402   ----
        404   ----
        406   ----
        408   ----
        410   MOVE R1,R0             RETURN TO MAINLINE PROGRAM
```

3.1.1 JUMP NO DATA BITS PRESENT

1 1 0 0 R 1 R 2 0 0 0 0

The low order byte of the register specified by the R1 field is tested for a non 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.

3.1.2 JUMP DATA ≤ MASK

1 1 0 0 R 1 R 2 0 0 0 1

The low order byte of the register specified by the R1 field is algebraically 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 false condition for this instruction is equivalent to a 'Jump Data > Mask' instruction.

3.1.3 JUMP DATA < MASK

1 1 0 0 R 1 R 2 0 0 1 0

The low order byte of the register specified by the R1 field is algebraically 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.

3.1.4 JUMP DATA = MASK

1 1 0 0 R 1 R 2 0 0 1 1

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.

3.1.5 JUMP ALL DATA BITS PRESENT

1 1 0 0 R 1 R 2 0 1 0 0

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 this 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.

3.1.6 JUMP DATA = ALL MASK BITS

1 1 0 0 R 1 R 2 0 1 0 1

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.

3.1.7 JUMP DATA = NO MASK BITS

1 1 0 0 R 1 R 2 0 1 1 0

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:

1 0 1 0 1 1 0 0 MASK

TEST SUCCESSFUL

C D 0 D 0 0 D D DATA

3.2 ARITHMETIC/LOGICAL OPERATION

```
0 0 0 0  OP CODE
       R 1    SOURCE
            R 2   DESTINATION
                  0 0 0 0      MOVE HALFWORD -2
                  0 0 0 1      MOVE HALFWORD -1
                  0 0 1 0      MOVE HALFWORK +1
                  0 0 1 1      MOVE HALFWORD +2
                  0 1 0 0      MOVE HALFWORK +0
                  0 1 0 1      AND
                  0 1 1 0      OR
                  0 1 1 1      XOR
                  1 0 0 0      ADD
                  1 0 0 1      SUB
                  1 0 1 0      ADD SPL #1
                  1 0 1 1      ADD SPL #2
                  1 1 0 0      SPARE
                  1 1 0 1      SPARE
                  1 1 1 0      SPARE
                  1 1 1 1      SPARE

0     3 4    7 8    11 12  15
```

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' and 'Add Special #1' 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.

3.2.1 MOVE HALFWORD

```
0 0 0 0   R 1   R 2    0 0 0 0  SUBTRACT 2
                       0 0 0 1  SUBTRACT 1
                       0 0 1 0  ADD 1
                       0 0 1 1  ADD 2
                       0 1 0 0  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.

3.2.2 AND BYTE

0 0 0 0 R 1 R 2 0 1 0 1

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 is uneffected. The source operand is not altered by this operation.

3.2.3 OR BYTE

0 0 0 0 R 1 R 2 0 1 1 0

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 and the source operand are unaltered by this operation.

3.2.4 EXCLUSIVE OR

0 0 0 0 R 1 R 2 0 1 1 1

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 and the source operand are unaltered by this operation.

3.2.5 ADD

0 0 0 0 R 1 R 2 1 0 0 0

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 was equal to zero, a resulting carry will be trapped alone at the end of instruction execution.

By use of the 'ADD SPECIAL' instruction, the carry can now be propagated into subsequent field additions. This will be more fully explained in Section 3.2.7. This instruction is also useful for performing address arithmetic. An eight bit modifier (source) can be added to a sixteen bit (destination) register to form a new storage address. A carry out of the high order position of the destination register is not retained by the system.

3.2.6 SUBTRACT

0 0 0 0 R 1 R 2 1 0 0 1

The low order byte of the source register (R1) is logically subtracted from the low order byte of the destination register. 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 3.2.8. 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 is used to subtract an eight bit unsigned binary quantity from a sixteen bit unsigned quantity.

3.2.7 ADD SPECIAL # 1

0 0 0 0 R 1 R 2 1 0 1 0

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 the 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 progagated into the high order byte of the destination register. Combining this instruction and the normal 'ADD' instruction provides a means for positioning individual bits or hexadecimal digits within a particular byte. As an example:

```
LBD   R1,R2      Fetch an operand and place it in R1 low
ADD   R1,R1      Shift left one position
ADDS1 R1,R1      Add spill bit to low order position of R1
ADD   R1,R1      Shift left one position
ADDS1 R1,R1      Add spill bit into low order position of R1
```

This sequence is equivalent to a shift left and rotate.

3.2.8 ADD SPECIAL #2

0 0 0 0 R 1 R 2 1 0 1 1

This instruction is used to propagate 'borrows' during field subtraction operation. 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 destination register. The destination high order byte is cleared and a 'NO CARRY; condition causes a 'borrow' to propagate into it.

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
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

3.3 STORAGE OPERATIONS

DIRECT	2/3	REG	ADDRESS (HW)					
INDIRECT	4/5/6/7	REG	ADDR REG	MODIFIER				
	0	3 4	7 8	11	12	15		

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 microinstruction. This eight bit address is used to fetch or store sixteen bit operands within the first 256 halfword 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 contain the operand address. The individual storage operations are described as follows:

3.3.1 DIRECT HALFWORD FETCH

```
0 0 1 0    R 1        HEX  ADDRESS

0     3 4        7 8              15
```

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.

3.3.2 DIRECT HALFWORD STORE

```
0 0 1 1    R 1      HEX      ADDRESS

0     3 4        7 8              15
```

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.

3.2.3 INDIRECT HALFWORD FETCH

```
0 1 0 0    R 1    R 2      MODIFIER
```

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 values 0-7 add the corresponding numeric amount to the indirect address, while values 8-F subtract the corresponding numeric values of 1-8 from the indirect address. The updated indirect address is then written back to the register specified by the R2 field.

3.3.4 INDIRECT HALFWORD STORE

0 1 0 1 R 1 R 2 MODIFIER

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 3.2.3.

3.3.5 INDIRECT BYTE FETCH

0 1 1 0 R 1 R 2 MODIFIER

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.

3.3.6 INDIRECT BYTE STORE

0 1 1 1 R 1 R 2 MODIFIER

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.

3.4 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.

3.4.1 EMIT

```
1 0 0 0   R 1    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 the destination register is unaffected by this operation.

3.4.2 CLEAR BIT

```
1 0 0 1   R 1    BIT    MASK
0     3 4      7 8            15
```

Bits 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.

3.4.3 ADD IMMEDIATE + 1

```
1 0 1 0   R 1    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.

SET BIT

```
1 0 1 1    R 1          BIT MASK
0     3 4      7 8                15
```

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.

3.5 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.

3.5.1 CONTROL

```
0 0 0 1   DA      CCMMAND
0     3 4     7 8                15
```

Bits 8-15 of this instruction are transfered to 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. A list of system commands will be provided at a later date.

3.6 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 transfered for each I/O instruction executed. In the case of a 'Get Byte' instruction, the type of data transfered depends on the form of the last control instruction executed ie, the byte may be status information, an identifier, or normal data to be stored and processed.

3.6.1 PUT BYTE

1 1 1 0	DA	REG	MODIFIER
0	3 4	7 8	11 12 15

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 microinstruction. The address of the data to be transfered 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 3.2.3. Main storage is not altered as a result of instruction execution.

3.2.6 GET BYTE

1 1 1 1	DA	REG	MODIFIER
0	3 4	7 8	11 12 15

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 3.2.3