```
┌─────┼─────┐
| B U G S |
└─────┼─────┘
```

META 4A Principles of Operation

This publication is one of the machine reference manuals for the Brown University Graphics System (BUGS). The META 4A is the general-purpose processor component of BUGS. Readers are assumed to have a knowledge of the principles of operation of the IBM System/360.[1]

April 7, 1975

---

[1]The research for this project was done under a grant from the National Science Foundation (GJ-28401X) and a contract from the Office of Naval Research (n00014-67-A-0191-0023). Reproduction in whole or in part is permitted for any purpose of the United States Government.

TABLE OF CONTENTS

1 INFORMATION

1.1 FORMATS OPERAND

Information on the META 4A is stored in main memory in 8-bit units, called "bytes", as in the S/360. Bytes may be handled separately or grouped together in fields. The most common field consists of 2 bytes, and is sometimes called a "halfword"; these halfwords are the basic building block of CPU instructions and also the size of the fixed-point 2's complement numbers operated upon by arithmetic instructions. These instructions require the halfwords to be located on an even byte boundary. Other instructions operate upon variable length fields of bytes, called "character strings". These character strings may be located anywhere in memory and may be of any length.

1.2 ADDRESSING

2.10.76

Bytes in main storage are addressed consecutively from 0. The META 4A uses a 16-bit address, allowing for a maximum of 64K bytes. ~~Currently we have only 32K, and addresses wrap from X'7FFF' to 0.~~ A field of bytes (1 or more) is usually addressed by its leftmost byte.

1.3 ARITHMETIC

All arithmetic on the META 4A is performed on 16-bit 2's complement binary numbers using 2's complement arithmetic. Any overflow that occurs is ignored in some operations (such as address computation), but causes program action in certain others (such as the Add instructions).

2 CENTRAL PROCESSING UNIT (CPU)

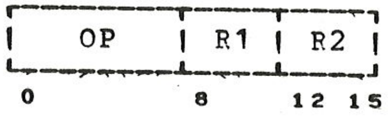
2.1 REGISTERS

Instructions can address information in 16 registers, 3 of which serve special purposes. Registers have a capacity of 1 halfword and are addressed by a 4-bit number from 0 to 15. Register 0 is called the Machine Status Register (MSR), register 1 is the Program Counter (PC), and register 15 is the Stack Frame Pointer (SFP). These will be explained later.

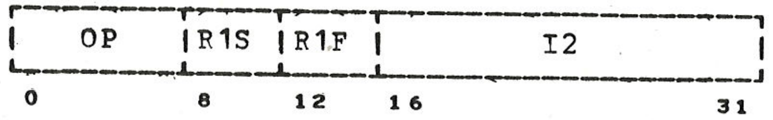
2.2 INSTRUCTION FORMATS

The META 4A has 8 instruction formats, denoted by the mnemonics RR, RI, RS, RX, BX, SI, FSS, and VSS. The formats are as follows:

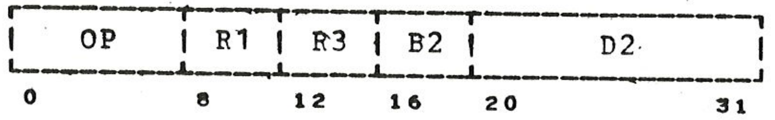
Register-Register (RR)
R1,R2



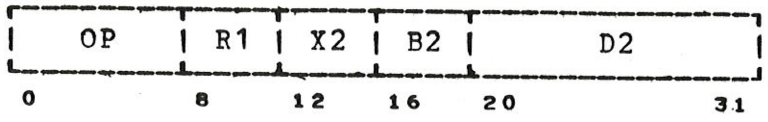
Register-Immediate (RI)
R1S,R1F,I2



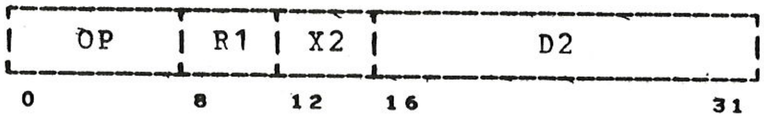
Register-Storage (RS)
R1,R3,D2(B2)



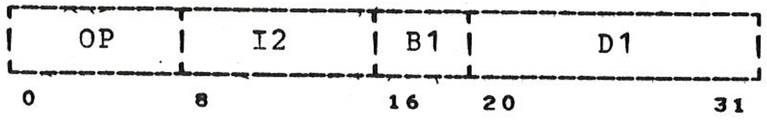
Register-indeXed storage (RX)
R1,D2(X2,B2)



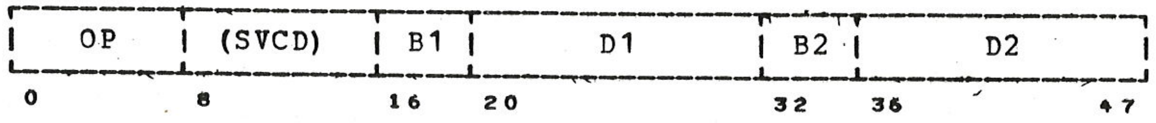
Branch indeXed (BX)
R1,D2(X2)



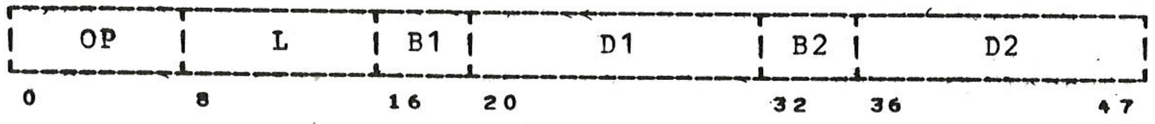
Storage-Immediate (SI)
D1(B1),I2



Fixed Storage-Storage (FSS)
D1(B1),D2(B2)



Variable Storage-Storage (VSS)
D1(L,B1),D2(B2)



The various instruction formats and fields will be referred to by the above mnemonics.

2.2.1 RI FORMAT

Most RI format instructions are 3-address; whenever the first operand must be fetched, it is fetched from the R1F register. Whenever it must be stored, it is stored into the R1S register. Certain instructions perform only R1 fetches (i.e., TMI, STDI, CLI, and CI), thus ignoring the R1S field, while LI and LDI perform only a store and ignore the R1F field. Both registers will be referred to as Operand 1. Operand 2 is always the 16-bit immediate halfword.

2.2.2 ADDRESS GENERATION

Operand addresses are generated for all instructions except those in BX format by taking the contents of the base register (B1/B2) and the displacement (D1/D2) (expanded to 16 bits by adding 4 high-order 0's) and performing 2's complement addition on them. Furthermore, in the case of the RX format, the contents of the index register (X2) is also added in. The resulting sum is the address of the operand. If a base and/or index register of 0 is specified in an instruction, the contents of the MSR is not used in the address computation, but rather a 0 is added.

2.2.3 BX FORMAT

The branch address in a BX format instruction is obtained by adding the signed immediate Displacement to the contents of the Program Counter (the next instruction address). Optionally on all intructions except BXH and BXLE, the contents of the X2 register is added in as an index, unless the X2 field is zero (as in RX format).

2.2.4 FSS (SVCD) FIELD

This field is ignored except in the SVCD instruction.

2.2.5 VSS LENGTH FIELD

The VSS format is used to operate upon character strings. The "length" field in the instruction specifies the length of the strings. Each VSS instruction has 2 operation codes, 1 allowing the length to be specified immediately as a number from 0 to 255 (actual length, unlike the S/360). The other ignores bits 9 through 11 and uses bits 12 through 15 as the number of a register containing the length, and will have the same mnemonic as the immediate type with an appended "L", for "Long". Thus, any length character string can be operated upon with only 1 instruction.

2.3 PROGRAM COUNTER (PC)

The PC (register 1) is the Program Counter (or instruction address register) for the META 4A. It is incremented by 2, during instruction fetching, for each halfword of the instruction fetched. During parsing and execution, it will contain the address of the next sequential instruction. It may be operated upon, in all respects, just as any of the other registers, although the effects upon program execution should be obvious. ~~For the purpose of fetching instructions, bit 15 of the PC is always ignored and assumed to be 0.~~

2.4 MACHINE STATUS REGISTER (MSR)

2.13.76 The MSR, register 0, contains the information required for proper program control and execution. It format is:

← soon to be Floating point bit

C0C1C2	F	ILC	W	PR	O	S	~~////~~	~~IØ~~	~~IO~~	IS	P
0	3	4	6	7	8	9	10 11	12	13	14	15

0 1 2 C0C1C2: This is the 3-bit Condition Code, which is set by various instructions. The 3 bits will be referred to as C0, C1, and C2. An instruction may affect none, some, or all of these bits.

3 F: This bit is undefined but may be used as a flag since it can be tested by the Branch Condition instructions (see Section 5).

4 5 ILC: These 2 bits are significant only when the MSR is stored after an SVC or Program interrupt (see Section 7), when they will specify the length, in halfwords, of the instruction that caused the interrupt. The ILC will never be set to 0.

6 W: This bit, when on, specifies that the META 4A is to be kept in wait state until the occurrence of some sort of I/O or S/360 interrupt.

7 PR: This bit is the Privilege bit, which must be on to perform an I/O or S/360 instruction. If this convention is violated, a Privilege Program interrupt will occur. See Section 8 for a more detailed explanation.

8 O: This is the Overflow interrupt mask. When off, arithmetic and conversion overflows on those instructions for which they are significant merely set the Condition Code. When on, they also cause a Program interrupt and the contents of the operand which was to receive the result remains unchanged.

9 S: This is the Stack Overflow/Underflow interrupt mask. When off, stack overflow or underflow merely sets the Condition Code. When on, either of these 2 conditions also causes a Program interrupt.

12 I/O: This bit controls the ability of local I/O units to cause an I/O interrupt. If off, these units cannot interrupt the CPU, and the interrupt request remains pending.

14 IS: This bit controls S/360 Initial Select interrupts. If 1, an initial selection causes an immediate interrupt. If 0, the interrupt is held pending.

15 P: This bit controls Parity Check interrupts. When on, Parity Checks cause an I/O interrupt. When off, the interrupt is kept pending.

3 ARITHMETIC INSTRUCTIONS

The description of each instruction includes a list of possible Program interrupts that can occur. These interrupts are explained in Section 7.

3.1 LOAD INSTRUCTIONS

Load Register
LR R1,R2

```
┌───────────┬──────┬──────┐
│    33     │  R1  │  R2  │
└───────────┴──────┴──────┘
0           8      12   15
```

Load heX digit
LX R1,R2

```
┌───────────┬──────┬──────┐
│    23     │  R1  │  R2  │
└───────────┴──────┴──────┘
0           8      12   15
```

Load Deferred Register
LDR R1,R2

```
┌───────────┬──────┬──────┐
│    13     │  R1  │  R2  │
└───────────┴──────┴──────┘
0           8      12   15
```

Load Immediate
LI R1S,R1F,I2

```
┌───────────┬──────┬──────┬───────────────────────┐
│    B3     │ R1S  │ R1F  │          I2           │
└───────────┴──────┴──────┴───────────────────────┘
0           8      12     16                     31
```

Load Deferred Immediate
LDI R1S,R1F,I2

```
┌───────────┬──────┬──────┬───────────────────────┐
│    B8     │ R1S  │ R1F  │          I2           │
└───────────┴──────┴──────┴───────────────────────┘
0           8      12     16                     31
```

LOAD

Load
L R1,D2(X2,B2)

73	R1	X2	B2	D2
0	8	12	16	20 31

Load Deferred
LD R1,D2(X2,B2)

53	R1	X2	B2	D2
0	8	12	16	20 31

Load Signed Byte
LSB R1,D2(X2,B2)

5B	R1	X2	B2	D2
0	8	12	16	20 31

The second operand is placed in the first operand location. The second operand for LX is the R2 field expanded with 0's to 16 bits, while the second operand for LDR is the halfword pointed to by the contents of R2. The LDI instruction uses the immediate halfword operand as the address of the halfword to load. LD adds an extra level of indirectness by using the second operand as an address to pick up still another halfword. LSB loads a byte, propagating its sign through bits 0 to 7 of the register. Alignment interrupts can occur for L, LD, LDR, LZ, and LDI.

Load and Zero
LZ R1,D2(X2,B2)

5E	R1	X2	B2	D2
0	8	12	16	20 31

The second operand halfword is placed in R1. Following this load, the halfword is zeroed. An Alignment interrupt can occur.

Load Complement Register
LCR R1,R2

```
┌──────────┬─────┬─────┐
│    0E    │ R1  │ R2  │
└──────────┴─────┴─────┘
0          8     12  15
```

Load Complement
LC R1,D2(X2,B2)

```
┌──────────┬─────┬─────┬─────┬────────────────┐
│    4E    │ R1  │ X2  │ B2  │       D2       │
└──────────┴─────┴─────┴─────┴────────────────┘
0          8     12    16    20             31
```

The 2's complement of the second operand is placed in the first operand location. C1 is set if overflow occurs, or reset otherwise. Arithmetic Overflow and Alignment (LC only) interrupts can occur.

Load Positive Register
LPR R1,R2

```
┌──────────┬─────┬─────┐
│    0C    │ R1  │ R2  │
└──────────┴─────┴─────┘
0          8     12  15
```

Load Positive
LP R1,D2(X2,B2)

```
┌──────────┬─────┬─────┬─────┬────────────────┐
│    4C    │ R1  │ X2  │ B2  │       D2       │
└──────────┴─────┴─────┴─────┴────────────────┘
0          8     12    16    20             31
```

The absolute value of the second operand is placed in the first operand location. C1 is set if overflow occurs, or reset otherwise. Arithmetic Overflow and Alignment (LP only) interrupts can occur.

Load Negative Register
LNR R1,R2

```
┌──────────┬─────┬─────┐
│    0D    │ R1  │ R2  │
└──────────┴─────┴─────┘
0          8     12  15
```

Load Negative
LN   R1,D2(X2,B2)

```
┌────────┬────┬────┬────┬────────────┐
│   4D   │ R1 │ X2 │ B2 │     D2     │
└────────┴────┴────┴────┴────────────┘
 0        8    12   16   20         31
```

The 2's complement of the absolute value of the second operand is placed in the first operand location. An Alignment interrupt can occur for LN.

Load Multiple
LM   R1,R3,D2(B2)

```
┌────────┬────┬────┬────┬────────────┐
│   93   │ R1 │ R3 │ B2 │     D2     │
└────────┴────┴────┴────┴────────────┘
 0        8    12   16   20         31
```

Load Multiple Deferred
LMD   R1,R3,D2(B2)

```
┌────────┬────┬────┬────┬────────────┐
│   91   │ R1 │ R3 │ B2 │     D2     │
└────────┴────┴────┴────┴────────────┘
 0        8    12   16   20         31
```

The set of registers from R1 to R3 (wrapping from register 15 to the MSR if necessary) is loaded from consecutive halfwords at the second operand address (LM) or at the address specified by the second operand halfword (LMD). If R1 specifies the MSR, no I/O or S/360 interrupts can occur after execution. An Alignment interrupt can occur.

3.2 ADD INSTRUCTIONS

Add Registers
AR   R1,R2

```
┌────────┬────┬────┐
│   34   │ R1 │ R2 │
└────────┴────┴────┘
 0        8    12 15
```

Add heX digit
AX   R1,R2

```
┌──────────┬─────┬─────┐
|    24    | R1  | R2  |
└──────────┴─────┴─────┘
0          8     12  15
```

Add Immediate
AI   R1S,R1F,I2

```
┌──────────┬─────┬─────┬────────────────────┐
|    B4    |R1S  |R1F  |         I2         |
└──────────┴─────┴─────┴────────────────────┘
0          8     12    16                  31
```

Add
A   R1,D2(X2,B2)

```
┌──────────┬─────┬─────┬─────┬──────────────┐
|    74    | R1  | X2  | B2  |      D2      |
└──────────┴─────┴─────┴─────┴──────────────┘
0          8     12    16    20            31
```

Add to Halfword Immediate
AHI   D1(B1),I2

```
┌──────────┬───────────┬─────┬──────────────┐
|    84    |    I2     | B1  |      D1      |
└──────────┴───────────┴─────┴──────────────┘
0          8           16    20            31
```

Add Halfwords
AH   D1(B1),D2(B2)

```
┌──────────┬───────────┬─────┬──────────────┬─────┬──────────────┐
|    F4    |     0     | B1  |      D1      | B2  |      D2      |
└──────────┴───────────┴─────┴──────────────┴─────┴──────────────┘
0          8           16    20             32    36            47
```

The second operand is added using 2's complement arithmetic to the first operand, and the sum is put in the first operand location. The second operand for AX is the R2 field treated as a 4-bit number from 0 to 15. For AHI, the second operand is the immediate byte extended with 8 high-order 0's. C1 is set to the overflow condition resulting from the addition. Arithmetic Overflow and Alignment (except for AR, AX, and AI) interrupts can occur.

Add Logical Registers
ALR R1,R2

```
┌────────────┬──────┬──────┐
│     3C     │  R1  │  R2  │
└────────────┴──────┴──────┘
 0            8      12  15
```

Add Logical Immediate
ALI R1S,R1F,I2

```
┌────────────┬──────┬──────┬──────────────────────────┐
│     BC     │ R1S  │ R1F  │            I2            │
└────────────┴──────┴──────┴──────────────────────────┘
 0            8      12     16                      31
```

Add Logical
AL R1,D2(X2,B2)

```
┌────────────┬──────┬──────┬──────┬───────────────────┐
│     7C     │  R1  │  X2  │  B2  │        D2         │
└────────────┴──────┴──────┴──────┴───────────────────┘
 0            8      12     16     20              31
```

The second operand is logically added to the first, and the result is placed in the first operand location. C1 is set to the resulting carry out. An Alignment interrupt can occur for AL.

3.3 SUBTRACT INSTRUCTIONS

Subtract Registers
SR R1,R2

```
┌────────────┬──────┬──────┐
│     35     │  R1  │  R2  │
└────────────┴──────┴──────┘
 0            8      12  15
```

Subtract heX digit
SX R1,R2

```
┌────────────┬──────┬──────┐
│     25     │  R1  │  R2  │
└────────────┴──────┴──────┘
 0            8      12  15
```

Subtract Immediate
SI R1S,R1F,I2

```
+----------+------+------+-------------------+
|    B5    | R1S  | R1F  |        I2         |
+----------+------+------+-------------------+
0          8      12     16                 31
```

Subtract
S R1,D2(X2,B2)

```
+----------+------+------+------+---------------+
|    75    |  R1  |  X2  |  B2  |      D2       |
+----------+------+------+------+---------------+
0          8      12     16     20             31
```

Subtract from Halfword Immediate
SHI D1(B1),I2

```
+----------+-------------+------+---------------+
|    85    |     I2      |  B1  |      D1       |
+----------+-------------+------+---------------+
0          8             16     20             31
```

Subtract Halfwords
SH D1(B1),D2(B2)

```
+----------+-------------+------+---------------+------+---------------+
|    F5    |      0      |  B1  |      D1       |  B2  |      D2       |
+----------+-------------+------+---------------+------+---------------+
0          8             16     20              32     36             47
```

The second operand is subtracted from the first, and the result replaces the first operand. The SX second operand is the R2 field treated as a 4-bit number from 0 to 15. For SHI, the second operand is the immediate byte extended with 8 high-order 0's. C1 is set to the overflow condition resulting from the subtraction. Arithmetic Overflow and Alignment (except for SR, SX, and SI) interrupts can occur.

Subtract Logical Registers
SLR R1,R2

```
+----------+------+------+
|    3D    |  R1  |  R2  |
+----------+------+------+
0          8      12    15
```

Subtrace Logical Immediate
SLI R1S,R1F,I2

```
┌───────────────┬────────┬────────┬───────────────────────────────────┐
│      BD       │ R1S    │ R1F    │                I2                 │
└───────────────┴────────┴────────┴───────────────────────────────────┘
 0               8        12       16                                31
```

Subtract Logical
SL R1,D2(X2,B2)

```
┌───────────────┬────────┬────────┬────────┬──────────────────────────┐
│      7D       │  R1    │  X2    │  B2    │           D2             │
└───────────────┴────────┴────────┴────────┴──────────────────────────┘
 0               8        12       16       20                      31
```

The second operand is logically subtracted from the first, and the result replaces the first operand. C1 is set to the resulting carry. An Alignment interrupt can occur on SL. Note that carry is the complement of borrow.

3.4 ARITHMETIC COMPARE INSTRUCTIONS

Compare Registers
CR R1,R2

```
┌───────────────┬────────┬────────┐
│      30       │  R1    │  R2    │
└───────────────┴────────┴────────┘
 0               8        12    15
```

Compare Immediate
CI R1S,R1F,I2

```
┌───────────────┬────────┬────────┬───────────────────────────────────┐
│      B0       │ R1S    │ R1F    │                I2                 │
└───────────────┴────────┴────────┴───────────────────────────────────┘
 0               8        12       16                                31
```

Compare
C R1,D2(X2,B2)

```
┌───────────────┬────────┬────────┬────────┬──────────────────────────┐
│      70       │  R1    │  X2    │  B2    │           D2             │
└───────────────┴────────┴────────┴────────┴──────────────────────────┘
 0               8        12       16       20                      31
```

Compare Halfword to Immediate
CHI D1(B1),I2

```
┌──────────┬──────────┬─────┬──────────────┐
│    80    │    I2    │ B1  │      D1      │
└──────────┴──────────┴─────┴──────────────┘
 0          8          16    20           31
```

Compare Halfwords
CH D1(B1),D2(B2)

```
┌──────────┬──────────┬─────┬──────────────┬─────┬──────────────┐
│    F0    │    0     │ B1  │      D1      │ B2  │      D2      │
└──────────┴──────────┴─────┴──────────────┴─────┴──────────────┘
 0          8          16    20             32    36           47
```

The first operand is compared algebraically with the second, and the result determines the setting of the Condition Code: C0, C1, and C2 are reset and then C0 is set if equal, C1 if first operand greater, or C2 if less. The second operand for CHI is the immediate byte treated as an 8-bit signed number. Alignment interrupts can occur for C, CHI, and CH.

3.5 MULTIPLY INSTRUCTIONS

Multiply Registers
MR R1,R2

```
┌──────────┬─────┬─────┐
│    36    │ R1  │ R2  │
└──────────┴─────┴─────┘
 0          8     12  15
```

Multiply Immediate
MI R1S,R1F,I2

```
┌──────────┬─────┬─────┬────────────────────┐
│    B6    │ R1S │ R1F │         I2         │
└──────────┴─────┴─────┴────────────────────┘
 0          8     12    16                 31
```

Multiply
M R1,D2(X2,B2)

```
┌──────────┬─────┬─────┬─────┬──────────────┐
│    76    │ R1  │ X2  │ B2  │      D2      │
└──────────┴─────┴─────┴─────┴──────────────┘
 0          8     12    16    20           31
```

Multiply Halfwords
MH   D1(B1),D2(B2)

```
┌────────┬────────┬────┬────────────┬────┬────────────┐
│   F6   │   0    │ B1 │     D1     │ B2 │     D2     │
└────────┴────────┴────┴────────────┴────┴────────────┘
 0        8        16   20           32   36          47
```

The first and second operands are multiplied and the 2-halfword (32-bit) product is placed in the first operand location and the following register or halfword. C1 is reset unless the product requires more than 16 bits, in which case it is set. An Alignment interrupt can occur for M and MH, and a Register Specification interrupt for MR and M if register 15 is specified as the first operand or for MI if register 15 is specified in the R1S field.

3.6 DIVIDE INSTRUCTIONS

Divide Registers
DR   R1,R2

```
┌────────┬────┬────┐
│   37   │ R1 │ R2 │
└────────┴────┴────┘
 0        8    12 15
```

Divide Immediate
DI   R1S,R1F,I2

```
┌────────┬────┬────┬────────────────┐
│   B7   │R1S │R1F │       I2       │
└────────┴────┴────┴────────────────┘
 0        8    12   16              31
```

Divide
D   R1,D2(X2,B2)

```
┌────────┬────┬────┬────┬────────────┐
│   77   │ R1 │ X2 │ B2 │     D2     │
└────────┴────┴────┴────┴────────────┘
 0        8    12   16   20          31
```

Divide Halfwords
DH   D1(B1),D2(B2)

```
┌────────┬────────┬────┬────────────┬────┬────────────┐
│   F7   │   0    │ B1 │     D1     │ B2 │     D2     │
└────────┴────────┴────┴────────────┴────┴────────────┘
 0        8        16   20           32   36          47
```

The double register/halfword (32-bit) first operand is divided by the second operand and the quotient and remainder, respectively, replace the 2 registers/halfwords of the first operand. Alignment interrupts can occur for D and DH, Register Specification interrupts for DR, DI, and D (if register 15 is specifed as the first operand), and Division by 0 and Arithmetic Overflow interrupts for all 4 formats. C1 is set to the overflow condition.

3.7 CONVERT INSTRUCTIONS

ConVert to Binary
CVB R1,D2(X2,B2)

```
┌──────────┬─────┬─────┬─────┬──────────────────────┐
│    50    │ R1  │ X2  │ B2  │          D2          │
└──────────┴─────┴─────┴─────┴──────────────────────┘
 0          8     12    16    20                   31
```

This instruction takes a string of EBCDIC digit characters (with optional leading blanks and/or minus sign) and converts the number to binary, placing it in R1. Register 2 is set to the address of the first invalid character following the digits. C0, C1, and C2 are initially set to 0. C0 is set if there were no valid digits, C1 is set if there was overflow (the result required more than 16 bits to represent), or C2 is set if the instruction was completed successfully. A Conversion Overflow interrupt can occur.

ConVert to Decimal
CVD R1,D2(X2,B2)

```
┌──────────┬─────┬─────┬─────┬──────────────────────┐
│    51    │ R1  │ X2  │ B2  │          D2          │
└──────────┴─────┴─────┴─────┴──────────────────────┘
 0          8     12    16    20                   31
```

This instruction takes the contents of R1 and converts it to a 6-digit EBCDIC decimal number with leading 0's replaced by blanks and a floating minus sign, if appropriate. Register 2 is set to the absolute address of the first non-blank generated, while register 3 is set to the length of the remaining string.

3.8 STORE INSTRUCTIONS

STore Deferred Register
STDR R1,R2

```
┌──────────┬─────┬─────┐
│    12    │ R1  │ R2  │
└──────────┴─────┴─────┘
0          8     12  15
```

STore Deferred Immediate
STDI R1S,R1F,I2

```
┌──────────┬─────┬─────┬─────────────────────┐
│    B1    │ R1S │ R1F │         I2          │
└──────────┴─────┴─────┴─────────────────────┘
0          8     12    16                   31
```

STore
ST R1,D2(X2,B2)

```
┌──────────┬─────┬─────┬─────┬───────────────┐
│    72    │ R1  │ X2  │ B2  │      D2       │
└──────────┴─────┴─────┴─────┴───────────────┘
0          8     12    16    20             31
```

STore Deferred
STD R1,D2(X2,B2)

```
┌──────────┬─────┬─────┬─────┬───────────────┐
│    52    │ R1  │ X2  │ B2  │      D2       │
└──────────┴─────┴─────┴─────┴───────────────┘
0          8     12    16    20             31
```

STDR stores the contents of R1 at the halfword location pointed to by R2. The STDI instruction uses the immediate operand as the address of where to store the register. ST stores R1 at the RX address. STD stores R1 at the halfword location whose address is Operand 2. An Alignment interrupt can occur.

STore Multiple
STM R1,R3,D2(B2)

```
┌──────────┬─────┬─────┬─────┬───────────────┐
│    92    │ R1  │ R3  │ B2  │      D2       │
└──────────┴─────┴─────┴─────┴───────────────┘
0          8     12    16    20             31
```

STore Multiple Deferred
STMD   R1,R3,D2(B2)

90	R1	R3	B2	D2
0	8	12	16	20 ... 31

The set of registers from R1 to R3 (wrapping from register 15 to the MSR, if necessary) is stored in consecutive halfwords at the second operand location (STM) or at the address specified by the second operand halfword (STMD). An Alignment interrupt can occur.

3.9 MOVE HALFWORD INSTRUCTIONS

MoVe to Halfword Immediate
MVHI   D1(B1),I2

83	I2	B1	D1
0	8	16	20 ... 31

MoVe Halfword
MVH   D1(B1),D2(B2)

F3	0	B1	D1	B2	D2
0	8	16	20	32	36 ... 47

The second operand is placed in the first operand halfword. The second operand for MVHI is the immediate signed byte expanded to a halfword. An Alignment interrupt can occur.

3.10 SWAP INSTRUCTIONS

SwaP Registers
SWPR   R1,R2

11.18.75

38 ~~88~~	R1	R2
0	8	12 15

SHIFT

SwaP
SWP R1,D2(X2,B2)

78	R1	X2	B2	D2
0	8	12	16	20 31

The contents of the first operand register is interchanged with the contents of the second operand register/halfword. An Alignment interrupt can occur for SWP.

3.11 ARITHMETIC SHIFT INSTRUCTIONS

Shift Left Algebraic Immediate
SLAI R1,R2

1C	R1	R2
0	8	12 15

Shift Left Algebraic
SLA R1,R3,D2(B2)

9C	R1	R3	B2	D2
0	8	12	16	20 31

Shift Right Algebraic Immediate
SRAI R1,R2

1D	R1	R2
0	8	12 15

Shift Right Algebraic
SRA R1,R3,D2(B2)

9D	R1	R3	B2	D2
0	8	12	16	20 31

Shift Left Double Algebraic Immediate
SLDAI R1,R2

1E	R1	R2
0	8	12 15

Shift Left Double Algebraic
SLDA R1,R3,D2(B2)

9E	R1	R3	B2	D2
0	8	12	16	20 31

Shift Right Double Algebraic Immediate
SRDAI R1,R2

1F	R1	R2
0	8	12 15

Shift Right Double Algebraic
SRDA R1,R3,D2(B2)

9F	R1	R3	B2	D2
0	8	12	16	20 31

The register or pair of registers specifed by R1 is shifted left or right the number of bits specified by the R2 field for the RR form, or by the 5 low order bits of the computed address for the RS form. Right shifts cause vacated bit positions to be filled with the original sign bit. For left shifts, if a bit unlike the sign bit is shifted out of bit position 1 of R1, overflow is set into C1 and Arithmetic Overflow interrupts can occur. C2 is set to the last bit shifted out. Register Specification interrupts occur for double shifts if R1 is specified as register 15.

Shift for Divide
SD R1,R2

17	R1	R2
0	8	12 15

The register specified by R1 is shifted right into the next higher register, and its sign is propagated through R1. This instruction can be used to set up a 32-bit dividend for a divide instruction. C2 is set to the original bit 0 of the next higher register. Register Specification interrupts can occur if R1 specifies register 15.

Logical Instructions

COMPARE

Compare Logical Immediate
CLI R1S,R1F,I2

BF	R1S	R1F	I2	
0	8	12	16	31

Compare Logical
CL R1,D2(X2,B2)

7F	R1	X2	B2	D2	
0	8	12	16	20	31

Compare Logical to Byte
CLB R1,D2(X2,B2)

5F	R1	X2	B2	D2	
0	8	12	16	20	31

Compare Logical Byte to Immediate
CLBI D1(B1),I2

8F	I2	B1	D1	
0	8	16	20	31

Compare Logical Halfwords
CLH D1(B1),D2(B2)

FB	0	B1	D1	B2	D2	
0	8	16	20	32	36	47

Compare Logical Characters (Long)
CLC/CLCL D1(L,B1),D2(B2)

CF/DF	L	B1	D1	B2	D2	
0	8	16	20	32	36	47

The first operand is compared logically with the second, and the result determines the setting of the Condition Code as in the algebraic compares. For CLB, Operand 1 is the low order byte of R1. CLC/CLCL causes register 2 to be set to the address in Operand 1 of the first unequal byte, unless an equal compare occurs. An equal compare always occurs if the length specified is zero, as two null strings are considered equal. An Alignment interrupt can occur for CL and CLH.

AND

4.3 AND INSTRUCTIONS

aNd Registers
NR R1,R2

RR

39	R1	R2
0	8	12 15

aNd Immediate
NI R1S,R1F,I2

RI

B9	R1S	R1F	I2
0	8	12	16 31

aNd
N R1,D2(X2,B2)

RX

79	R1	X2	B2	D2
0	8	12	16	20 31

aNd to Byte Immediate
NBI D1(B1),I2

SI

89	I2	B1	D1
0	8	16	20 31

aNd Characters (Long)
NC/NCL D1(L,B1),D2(B2)

VSI

C9/D9	L	B1	D1	B2	D2
0	8	16	20	32	36 47

The first operand is ANDed with the second and the result replaces the first operand. C0 is set to 1 if the result is all 0's or the length is 0, to 0 otherwise. An Alignment interrupt can occur for N.

4.4 OR INSTRUCTIONS

Or Registers
OR R1,R2

```
┌───────────┬──────┬──────┐
|    3A     | R1   | R2   |
└───────────┴──────┴──────┘
 0           8      12  15
```

Or Immediate
OI R1S,R1F,I2

```
┌───────────┬──────┬──────┬───────────────────────┐
|    BA     | R1S  | R1F  |          I2           |
└───────────┴──────┴──────┴───────────────────────┘
 0           8      12     16                    31
```

Or
O R1,D2(X2,B2)

```
┌───────────┬──────┬──────┬──────┬─────────────────┐
|    7A     | R1   | X2   | B2   |       D2        |
└───────────┴──────┴──────┴──────┴─────────────────┘
 0           8      12     16     20              31
```

Or to Byte Immediate
OBI D1(B1),I2

```
┌───────────┬─────────────┬──────┬─────────────────┐
|    8A     |     I2      | B1   |       D1        |
└───────────┴─────────────┴──────┴─────────────────┘
 0           8             16     20              31
```

Or Characters (Long)
OC/OCL D1(L,B1),D2(B2)

```
┌───────────┬─────────────┬──────┬──────────────────┬──────┬──────────────────┐
|  CA/DA    |      L      | B1   |        D1        | B2   |        D2        |
└───────────┴─────────────┴──────┴──────────────────┴──────┴──────────────────┘
 0           8             16     20                 32     36               47
```

The first operand is ORed with the second and the result replaces the first operand. C0 is set to 1 if the result is all 0's or the length is 0, to 0 otherwise. An Alignment interrupt can occur for O.

4.5 EXCLUSIVE OR INSTRUCTIONS

eXclusive or Registers
XR R1,R2

```
┌───────────┬──────┬──────┐
│    3B     │  R1  │  R2  │
└───────────┴──────┴──────┘
0           8      12   15
```

eXclusive or Immediate
XI R1S,R1F,I2

```
┌───────────┬──────┬──────┬────────────────────────┐
│    BB     │ R1S  │ R1F  │           I2           │
└───────────┴──────┴──────┴────────────────────────┘
0           8      12     16                      31
```

eXclusive or
X R1,D2(X2,B2)

```
┌───────────┬──────┬──────┬──────┬──────────────────┐
│    7B     │  R1  │  X2  │  B2  │        D2        │
└───────────┴──────┴──────┴──────┴──────────────────┘
0           8      12     16     20                31
```

eXclusive or to Byte Immediate
XBI D1(B1),I2

```
┌───────────┬─────────────┬──────┬──────────────────┐
│    8B     │     I2      │  B1  │        D1        │
└───────────┴─────────────┴──────┴──────────────────┘
0           8             16     20                31
```

exclusive or Characters (Long)
XC/XCL D1(L,B1),D2(B2)

```
┌───────────┬─────────────┬──────┬──────────────────┬──────┬──────────────────┐
│  CB/DB    │      L      │  B1  │        D1        │  B2  │        D2        │
└───────────┴─────────────┴──────┴──────────────────┴──────┴──────────────────┘
0           8             16     20                 32     36                47
```

The first operand is EXCLUSIVE ORed with the second and the result replaces the first operand. C0 is set to 1 if the result is all 0's or the length is 0, to 0 otherwise. An Alignment interrupt can occur for X.

4.6 TEST UNDER MASK INSTRUCTIONS

Test under Mask Register
TMR R1,R2

32	R1	R2
0	8	12 15

Test under Mask Immediate
TMI R1S,R1F,I2

B2	R1S	R1F	I2
0	8	12	16 31

11.18.75

Test Byte under Mask Immediate
TMBI ~~TBMI~~ D1(B1),I2

82	I2	B1	D1
0	8	16	20 31

The 1 bits in the second operand mask select the corresponding bits in the first operand. These bits are then tested and the Condition Code is set as follows: all bits are reset and then C0 is set if the tested bits were all 0's or the mask was 0, C1 is set if all tested bits were 1, or C2 is set if they were mixed 0's and 1's.

4.7 BYTE INSTRUCTIONS

Insert Byte
IB R1,D2(X2,B2)

58	R1	X2	B2	D2
0	8	12	16	20 31

Load Byte
LB R1,D2(X2,B2)

5A	R1	X2	B2	D2
0	8	12	16	20 31

The byte at the second operand location is placed into bits 8 through 15 of R1, and bits 0 to 7 are left the same (IB) or zeroed (LB).

STore Byte
STB R1,D2(X2,B2)

```
┌───────────┬──────┬──────┬──────┬──────────────────┐
│    59     │  R1  │  X2  │  B2  │        D2        │
└───────────┴──────┴──────┴──────┴──────────────────┘
 0           8      12     16     20              31
```

Bits 8 to 15 of R1 are stored as a byte into the second operand location.

FILL (Long)
FILL
FILLL D1(L,B1),D2(B2)

```
┌───────────┬─────────────┬──────┬──────────────────┬──────┬──────────────────┐
│  C8/D8    │      L      │  B1  │        D1        │  B2  │        D2        │
└───────────┴─────────────┴──────┴──────────────────┴──────┴──────────────────┘
 0           8             16     20                 32     36              47
```

The first operand character string is filled with the low-order byte of the second operand <u>address</u>.

Load and eXchange Bytes Register
LXBR R1,R2

```
┌───────────┬──────┬──────┐
│    0F     │  R1  │  R2  │
└───────────┴──────┴──────┘
 0           8      12  15
```

Load and eXchange Bytes
LXB R1,D2(X2,B2)

```
┌───────────┬──────┬──────┬──────┬──────────────────┐
│    4F     │  R1  │  X2  │  B2  │        D2        │
└───────────┴──────┴──────┴──────┴──────────────────┘
 0           8      12     16     20              31
```

The second operand halfword is loaded into R1, with its 2 bytes exchanged. An Alignment interrupt can occur for LXB.

4.8 ADDRESS MANIPULATION INSTRUCTIONS

Load Address
LA R1,D2(X2,B2)

5D	R1	X2	B2	D2
0	8	12	16	20 31

Subtract Address
SA R1,D2(X2,B2)

5C	R1	X2	B2	D2
0	8	12	16	20 31

The computed RX address is loaded into (LA) or subtracted from (SA) R1.

MoVe Address
MVA D1(0,B1),D2(B2)

E2	0	B1	D1	B2	D2
0	8	16	20	32	36 47

The computed second operand address is moved to the first operand location.

4.9 TRANSLATE AND SCAN INSTRUCTIONS

TRanslate (Long)
TR/TRL D1(L,B1),D2(B2)

CE/DE	L	B1	D1	B2	D2
0	8	16	20	32	36 47

The bytes at the first operand location are used as arguments to reference 1 of the 256 bytes in the second operand table, by adding each argument byte to the second operand address. The resulting byte then replaces the original argument in Operand 1.

Scan using Table Right (Long)
STR/STRL D1(L,B1),D2(B2)

CD/DD	L	B1	D1	B2	D2
0	8	16	20	32	36 47

Scan using Table Left (Long)
STL/STLL D1(L,B1),D2(B2)

CC/DC	L	B1	D1	B2	D2
0	8	16	20	32	36 47

Each byte of Operand 1 is used (as in TR) to reference a byte in the table pointed to by Operand 2. If the byte thus referenced is 0, referencing continues. If it is non-0, it is loaded into register 3 (the high-order byte of which is zeroed), the argument byte address is placed in register 2, and C2 of the Condition Code is set. If all referenced bytes are 0 or the length is 0, C2 is reset to 0. STL addresses Operand 1 by its rightmost byte.

Scan Right Equal (Long)
SRE/SREL D1(L,B1),D2(B2)

C1/D1	L	B1	D1	B2	D2
0	8	16	20	32	36 47

Scan Right Not Equal (Long)
SRNE
SRNEL D1(L,B1),D2(B2)

C3/D3	L	B1	D1	B2	D2
0	8	16	20	32	36 47

Scan Left Equal (Long)
SLE
SLEL D1(L,B1),D2(B2)

C0/D0	L	B1	D1	B2	D2
0	8	16	20	32	36 47

Scan Left Not Equal (Long)
SLNE
SLNEL D1(L,B1),D2(B2)

C2/D2	L	B1	D1	B2	D2
0	8	16	20	32	36 47

This instruction uses the low-order byte of the second operand address as a character to scan Operand 1. A scan for equality (xxE) or inequality (xxNE) is performed. If the scan is satisfied, the absolute address of the satisfying character in Operand 1 is placed in register 2 and C2 is set. If it is not, C2 is reset to 0 (C2 is always reset if the length is 0). The scan can be performed from left to right (xRxx) or right to left (xLxx), in which case Operand 1 is addressed by its rightmost byte.

4.10 LOGICAL SHIFT INSTRUCTIONS

Shift Left Logical Immediate
SLLI R1,R2

18	R1	R2
0	8	12 15

Shift Left Logical
SLL R1,R3,D2(B2)

98	R1	R3	B2	D2
0	8	12	16	20 31

Shift Right Logical Immediate
SRLI R1,R2

19	R1	R2
0	8	12 15

Shift Right Logical
SRL R1,R3,D2(B2)

```
┌────────────┬──────┬──────┬──────┬──────────────────┐
|     99     |  R1  |  R3  |  B2  |        D2        |
└────────────┴──────┴──────┴──────┴──────────────────┘
 0            8      12     16     20               31
```

Shift Left Double Logical Immediate
SLDLI R1,R2

```
┌────────────┬──────┬──────┐
|     1A     |  R1  |  R2  |
└────────────┴──────┴──────┘
 0            8      12   15
```

Shift Left Double Logical
SLDL R1,R3,D2(B2)

```
┌────────────┬──────┬──────┬──────┬──────────────────┐
|     9A     |  R1  |  R3  |  B2  |        D2        |
└────────────┴──────┴──────┴──────┴──────────────────┘
 0            8      12     16     20               31
```

Shift Right Double Logical Immediate
SRDLI R1,R2

```
┌────────────┬──────┬──────┐
|     1B     |  R1  |  R2  |
└────────────┴──────┴──────┘
 0            8      12   15
```

Shift Right Double Logical
SRDL R1,R3,D2(B2)

```
┌────────────┬──────┬──────┬──────┬──────────────────┐
|     9B     |  R1  |  R3  |  B2  |        D2        |
└────────────┴──────┴──────┴──────┴──────────────────┘
 0            8      12     16     20               31
```

A left or right shift is performed on the register or contiguous pair of registers indicated by R1. The shift count is specifed by bits 12 to 15 of the instruction for the RR formats, or by the low order 5 bits of the Operand 2 <u>address</u> for the RS formats. C2 is set to the last bit shifted out. A Register Specification interrupt can occur on a double shift if R1 specifies register 15.

4.11 STACK INSTRUCTIONS

PuSH Multiple
PSHM   R1,R3,D2(B2)

96	R1	R3	B2	D2
0	8	12	16	20 31

PuSH Halfwords
PSHH
PSHHL   D1(L,B1),D2(B2)

C6/D6	L	B1	D1	B2	D2
0	8	16	20	32	36 47

These instruction are used to push data into the stack described by the Stack Descriptor Block (SDB) pointed to by the second operand address. The format of an SDB is:

current top	stack address	stack limit
0	2	4 6

current top: This halfword will always point to the top halfword in the stack.
stack address: This is the address of the stack itself and controls the amount of popping that can be performed.
stack limit: This is the address of the highest halfword that can be used as part of the stack. A pushing limit is thus established and checked.

PSHM causes the registers from R1 to R3 (wrapping from the SFP to the MSR if necessary) to be pushed into the stack and the current top pointer to be updated to point at the top halfword. PSHH/PSHHL pushes the number of bytes specified by the length field (rounded up to the next higher even number if odd) into the stack and leaves the current top pointer pointing at the last halfword pushed. These bytes are treated as a sequence of halfwords and thus must be aligned. It is obvious that the ~~stack~~ current top should be initialized to 2 less than the stack address.

Before any pushing is performed, the stack limit is checked to ensure that it will not be exceeded. If it is not, C1 is reset; if it is, C1 is set and a Stack Overflow Interrupt occurs if enabled.

POP Multiple
POPM R1,R3,D2(B2)

97	R1	R3	B2	D2
0	8	12	16	20 — 31

POP Halfwords
POPH
POPHL D1(L,B1),D2(B2)

C7/D7	L	B1	D1	B2	D2
0	8	16	20	32	36 — 47

POP instructions perform the complement of the pushing ones. The current top is first checked against the stack address to see if enough halfwords exist in the stack to do the pop. If not, C1 is set and a Stack Underflow interrupt occurs if enabled. If it is, C1 is reset and the halfwords are popped into the registers (POPM) or to the first operand address (POPH/POPHL). The current top is then decremented so as to point to the new top of the stack. The Operand 2 address is that of the SDB, as described above. An Alignment interrupt can occur.

4.12 LINKED LIST INSTRUCTIONS

SeaRCH
SRCH D1(B1),D2(B2)

F9	0	B1	D1	B2	D2
0	8	16	20	32	36 — 47

This instruction is used to search a table or a linked list for a key which holds some relation to a search argument. The table or linked list is described by an Entry Descriptor Block (EDB) pointed to by the second operand address, in addition to the contents of certain registers. An EDB looks as follows:

For a table

11.18.75

Logical Instructions

TCCC field: If T is 1, a Test under Mask is done on the key and search arguments. If T is 0, a Compare Logical is done. CCC is a mask for the condition code.

LINKED LIST

entry length	key disp.	key length	0/TCCC
0	2	4	6 7

For a linked list

pointer disp.	key disp.	key length	1/TCCC
0	2	4	6 7

entry length: This is the length in bytes of each entry in the table.

pointer disp.: This gives the offset in bytes of a halfword containing the relative address of the next entry in the linked list, called the "forward pointer". The address is relative to the area base as described below.

key disp.: This is the offset in bytes to the key in an entry.

key length: This is the length in bytes of the key in an entry and of the search argument.

The information in the EDB will most likely remain constant throughout the program. The search argument is pointed to by the first operand address and has a length equal to the key length. Registers 2 through 5 must be initialized prior to issuing a SRCH, as follows:

register 2: This register contains the relative address of the first entry to be searched. For a table it merely points to some entry, while for a linked list it must point to the forward pointer in the predecessor of the first entry to be searched. In order to start searching at the first entry, therefore, a "dummy pointer" to it must be available.

register 3: This contains a count of the number of entries to be searched and will be decremented by 1 after each entry checked unsuccessfully A 0 count causes the instruction to be ignored except for setting C1 as described below.

register 4: This is the area base address. It is added to all addresses pertaining to the table or linked list, including the initial entry address (in register 2) and all forward pointers. This allows relocatable linked lists (useful in computer graphics), or, if set to 0, absolute addressing.

register 5: When the search is completed, register 5 will be set to the relative address of the last entry searched, while register 2 will have been updated to contain the relative address of the previous entry. There are two exceptions to this, the first being when the initial count in register 3 is 0 or the forward pointer in the initial entry for a linked list is null (0), in which case both registers 2 and 5 will point to the initial entry. The other exception if for tables when the count is exhausted; register 2 will point to the last entry and register 5 to the first byte following the table.

If the search was successful, C2 will be set. Otherwise, C1 will be set if the entry count in register 3 was decremented to or initially 0, or C0 if a null forward pointer (0) was encountered in a linked list. For key lengths of 0, or for arguments of all 0's on a TM SRCH, a successful match occurs on the first entry if the criterion specifies equal or 0's; the search is unsuccessful otherwise. An Alignment interrupt can occur.

NOTE: SRCH can be used to perform a PL/I-like INDEX or an LSD-like FIND.

ENQueue
ENQ D1(B1),D2(B2)

F8	0	B1	D1	B2	D2
0	8	16	20	32	36 47

The first operand address plus the area base address in register 4 points to a new entry, which is be inserted into the linked list described by the EDB at the second operand address. The area base plus the relative address in register 2 gives the address of the forward pointer in the entry which is to become the predecessor of the new inserted entry. Note that SRCH sets up register 2 so that an ENQ or DEQ can be executed immediately. An Alignment interrupt can occur.

DEQueue
DEQ R1,D2(X2,B2)

48	R1	X2	B2	D2
0	8	12	16	20 31

The second operand address points to an EDB describing a linked list from which an entry is to be removed. The contents of register 4 (the area base address) plus the relative address in register 2 give the address of some entry's forward pointer; the next entry is removed from the list and C0 is reset. In case this forward pointer is null, no

LINKED LIST

dequeueing is performed and CO is set. An Alignment interrupt can occur.

5 BRANCHING INSTRUCTIONS

No RR format branching instructions cause a branch if R2 specifies register 0.

Test and Branch Positive Register
TBPR R1,R2

```
┌────────────────┬─────────┬─────────┐
|      29        |   R1    |   R2    |
└────────────────┴─────────┴─────────┘
 0                8         12    15
```

Test and Branch Positive
TBP R1,D2(X2)

```
┌────────────────┬─────────┬─────────┬──────────────────────────────────────┐
|      A9        |   R1    |   X2    |                 D2                   |
└────────────────┴─────────┴─────────┴──────────────────────────────────────┘
 0                8         12        16                                  31
```

Test and Branch Not Positive Register
TBNPR R1,R2

```
┌────────────────┬─────────┬─────────┐
|      2E        |   R1    |   R2    |
└────────────────┴─────────┴─────────┘
 0                8         12    15
```

Test and Branch Not Positive
TBNP R1,D2(X2)

```
┌────────────────┬─────────┬─────────┬──────────────────────────────────────┐
|      AE        |   R1    |   X2    |                 D2                   |
└────────────────┴─────────┴─────────┴──────────────────────────────────────┘
 0                8         12        16                                  31
```

Test and Branch Zero Register
TBZR R1,R2

```
┌────────────────┬─────────┬─────────┐
|      2A        |   R1    |   R2    |
└────────────────┴─────────┴─────────┘
 0                8         12    15
```

Test and Branch Zero
TBZ R1,D2(X2)

```
┌────────┬─────┬─────┬───────────────────┐
│   AA   │ R1  │ X2  │        D2         │
└────────┴─────┴─────┴───────────────────┘
 0        8     12    16                31
```

Test and Branch Not Zero Register
TBNZR R1,R2

```
┌────────┬─────┬─────┐
│   2D   │ R1  │ R2  │
└────────┴─────┴─────┘
 0        8     12  15
```

Test and Branch Not Zero
TBNZ R1,D2(X2)

```
┌────────┬─────┬─────┬───────────────────┐
│   AD   │ R1  │ X2  │        D2         │
└────────┴─────┴─────┴───────────────────┘
 0        8     12    16                31
```

Test and Branch Minus Register
TBMR R1,R2

```
┌────────┬─────┬─────┐
│   2C   │ R1  │ R2  │
└────────┴─────┴─────┘
 0        8     12  15
```

Test and Branch Minus
TBM R1,D2(X2)

```
┌────────┬─────┬─────┬───────────────────┐
│   AC   │ R1  │ X2  │        D2         │
└────────┴─────┴─────┴───────────────────┘
 0        8     12    16                31
```

Test and Branch Not Minus Register
TBNMR R1,R2

```
┌────────┬─────┬─────┐
│   2B   │ R1  │ R2  │
└────────┴─────┴─────┘
 0        8     12  15
```

Test and Branch Not Minus
TBNM R1,D2(X2)

```
┌────────┬─────┬─────┬───────────────────┐
│   AB   │ R1  │ X2  │        D2         │
└────────┴─────┴─────┴───────────────────┘
 0        8     12    16                31
```

A branch is taken to the address in ~~contents of~~ R2 (RR format) or to the second operand address (BX form) if the condition of R1 is as specified by the instruction.

Branch Conditions Zero Register
BCZR R1,R2

```
┌────────────┬──────┬──────┐
│     26     │  R1  │  R2  │
└────────────┴──────┴──────┘
 0            8      12   15
```

Branch Conditions Zero
BCZ R1,D2(X2)

```
┌────────────┬──────┬──────┬─────────────────────────┐
│     A6     │  R1  │  X2  │           D2            │
└────────────┴──────┴──────┴─────────────────────────┘
 0            8      12     16                      31
```

Branch Condition One Register
BCOR R1,R2

```
┌────────────┬──────┬──────┐
│     27     │  R1  │  R2  │
└────────────┴──────┴──────┘
 0            8      12   15
```

Branch Condition One
BCO R1,D2(X2)

```
┌────────────┬──────┬──────┬─────────────────────────┐
│     A7     │  R1  │  X2  │           D2            │
└────────────┴──────┴──────┴─────────────────────────┘
 0            8      12     16                      31
```

The R1 field is used as a mask to select any of the Condition Code bits or bit 3 in the MSR. If all selected bits are 0 for a BCZ(R), or if 1 or more are 1 for BCO(R), then a branch is taken to the contents of R2 (RR form), or to the second operand address (BX). Otherwise no branch is taken. If R1=0, then the branch is always taken. 2.10.76

Branch and Link Register
BALR R1,R2

```
┌────────────┬──────┬──────┐
│     28     │  R1  │  R2  │
└────────────┴──────┴──────┘
 0            8      12   15
```

Branching Instructions

Branch and Link
BAL   R1,D2(X2)

```
┌───────┬────┬────┬────────────────┐
│   A8  │ R1 │ X2 │       D2       │
└───────┴────┴────┴────────────────┘
0       8    12   16              31
```

The branch address (contents of R2 for RR form or second operand address for BX form) is computed and saved.   The PC is then copied into R1 and a branch is taken to the saved address.

Branch on CounT Register
BCTR   R1,R2

```
┌───────┬────┬────┐
│   2F  │ R1 │ R2 │
└───────┴────┴────┘
0       8    12 15
```

Branch on CounT
BCT   R1,D2(X2)

```
┌───────┬────┬────┬────────────────┐
│   AF  │ R1 │ X2 │       D2       │
└───────┴────┴────┴────────────────┘
0       8    12   16              31
```

R1 is decremented by 1, and if the result is positive, a branch is taken to the contents of R2 (RR form) or to the second operand address (BX). If the result is minus or 0, no branch is taken.

Branch on indeX High
BXH   R1,D2(X2)

```
┌───────┬────┬────┬────────────────┐
│   A0  │ R1 │ X2 │       D2       │
└───────┴────┴────┴────────────────┘
0       8    12   16              31
```

Branch on indeX Low or Equal
BXLE   R1,D2(X2)

```
┌───────┬────┬────┬────────────────┐
│   A1  │ R1 │ X2 │       D2       │
└───────┴────┴────┴────────────────┘
0       8    12   16              31
```

The contents of X2 is added algebraically to R1 and the result is compared algebraically to the previous contents of X2 (if it is an odd-numbered register) or to the contents of the next higher register (if X2 is even-numbered).   A branch is taken to the second operand address if R1 is greater than the compare register

(BXH) or if it is less than or equal to it (BXLE). Otherwise no branch is taken.

EXecute
EX   R1,D2(X2,B2)

```
┌──────────┬────┬────┬────┬──────────────┐
│    54    │ R1 │ X2 │ B2 │      D2      │
└──────────┴────┴────┴────┴──────────────┘
0          8    12   16   20            31
```

This instruction causes execution of the single instruction at the second operand address. The contents of R1 is temporarily ORed with the first halfword of this subject instruction before it is executed, unless R1 specifies register 0. Control is eventually returned to the instruction following the EX, unless the subject instruction modifies the PC. An Alignment interrupt can occur. Any Program interrupts that can occur for the executed instruction are possible. I/O and S/360 Initial Select interrupts cannot occur between and EX and its executed instruction. An Execute interrupt occurs if the subject of an EX instruction is another EX.

6 STATUS-SWITCHING INSTRUCTIONS

SuperVisor Call
SVC I1

```
┌────────┬────────┐
│   31   │   I1   │
└────────┴────────┘
0        8      15
```

SuperVisor Call Single-register
SVCS D1(B1),I2

```
┌────────┬────────┬────┬────────────┐
│   81   │   I2   │ B1 │     D1     │
└────────┴────────┴────┴────────────┘
0        8        16   20         31
```

SuperVisor Call Double-register
SVCD D1(B1),D2(B2),I3

```
┌────────┬────────┬────┬────────────┬────┬────────────┐
│   F1   │   I3   │ B1 │     D1     │ B2 │     D2     │
└────────┴────────┴────┴────────────┴────┴────────────┘
0        8        16   20           32   36         47
```

This instruction causes a Supervisor Call (SVC) interrupt using the second byte of the instruction as the interrupt code. Before causing the interrupt, SVCS loads the first operand address into register 2. SVCD does this also, plus loads register 3 with the second operand address.

Test and Set Lock
TSL D1(B1),I2

```
┌────────┬────────┬────┬────────────┐
│   87   │   I2   │ B1 │     D1     │
└────────┴────────┴────┴────────────┘
0        8        16   20         31
```

The bits of the first operand byte selected by the second operand mask are tested, and the Condition Code is set as for the TM instructions. If the test indicates 0's, the mask is ORed into the first operand. Memory accessing by local I/O units is prevented between testing and modification.

ENTer subroutine
ENT R1S,R1F,I2

BE	R1S	R1F	I2
0	8	12	16 31

This instruction causes save area chaining and automatic storage allocation. It is generally executed as the first instruction of a subroutine and assumes that the SFP (register 15) points into a "Stack Frame", which must be on a halfword boundary:

```
        |--------------------|
SFP-->  |previous pointer    |
        |--------------------|
        |   next pointer     |
        |--------------------|
        |                    |
        |                    |
        |      caller's      |
        |     save area      |
        |                    |
        |--------------------|
        |                    |
        |      caller's      |
        |     automatic      |
        |      storage       |
        |                    |
        |--------------------|
        |       length       |
        |--------------------|
        |                    |
```

SEE GMS MANUAL

The immediate halfword operand specifies the number of bytes of automatic storage desired. This number (rounded to the next higher halfword, if necessary) plus 30 bytes for a register save area plus 4 bytes for the new pointers is compared against the length, which specifies the remaining space in the Frame (including itself). If not enough space exists, a Stack Frame Overflow interrupt occurs. If there is space, it is allocated to the subroutine, and a new, updated length field is built below it. The caller's MSR through register 14 are saved in the save area in his storage ~~(with the PC set equal to register 14. the return address),~~ The old length field is replaced by the contents of the SFP (a previous pointer), the halfword following the length is set to point at the newly built length field (a next pointer), and finally the SFP is updated to point to the new save area:

11.18.75

SEE GMS MANUAL

```
          |--------------------|
          |previous pointer    |
          |--------------------|
          |   next pointer     |
          |--------------------|
          |                    |
           MSR-register 14
          |   save area        |
          |--------------------|
          |     caller's       |
                automatic
          |      storage       |
          |--------------------|
SFP-->    |previous pointer    |
          |--------------------|
          |   next pointer     |
          |--------------------|
          |                    |
              subroutine's
          |   save area        |
          |--------------------|
          |   subroutine's     |
               automatic
          |      storage       |
          |--------------------|
          |      length        |
          |--------------------|
          |                    |
```

An Alignment interrupt can occur.

RETurn from subroutine
RET   R1,R2

```
|------------|----|----|
|     0B     | R1 | R2 |
|------------|----|----|
0            8    12  15
```

RET is executed to return from a subroutine and to free the save area and automatic storage gotten by ENT. It first checks to see if the back pointer (the SFP is assumed to be left as set up by ENT) points to a save area which points forward to the current one. If not, a Stack Frame Underflow interrupt occurs. If so, registers 2 through 13 are reloaded from the current save area, the old length and 0 fields are rebuilt at the top of it (so as to free it), the SFP is backed up to the previous save area, and a branch is taken to the new contents of register 13, the return address. An Alignment interrupt can occur.

7 INTERRUPTS

There are four types of interrupts on the META 4A: SuperVisor Call (SVC), Program, I/O, and S/360. When an interrupt is detected, and if it is not disabled by the MSR or I/O Mask, the following sequence occurs:

1) The current MSR and PC are stored in 2 halfwords in low core (see table below), and an interrupt code is generated and stored. If the interrupt is an SVC or Program check, the ILC field in the stored MSR will contain the length, in halfwords, of the instruction causing the interrupt.
2) A new MSR and PC are loaded from 2 other halfwords in low core.
3) Execution continues with this new machine status.

The low core locations reserved for these MSR's, PC's, and interrupt codes are as follows:

address	contents
4	SVC old MSR
6	SVC old PC
8	SVC interrupt code
A	SVC new MSR
C	SVC new PC
E	Program old MSR
10	Program old PC
12	Program interrupt code
14	Program new MSR
16	Program new PC
18	I/O old MSR
1A	I/O old PC
1C	I/O interrupt code
1E	I/O new MSR
20	I/O new PC
22	S/360 old MSR
24	S/360 old PC
26	S/360 interrupt code
28	S/360 new MSR
2A	S/360 new PC

The SVC interrupt code is determined by the second byte of the SVC instruction that caused the interrupt. Program interrupt codes are determined by the type of Program interrupt, as follows:

Operation (2): An instruction with an invalid operation code was encountered. When this occurs, various information is stored in low core before the interrupt occurs. See Appendix I for a description of this information.

Stack Frame Overflow (4): On an ENT instruction, no room existed in the stack frame for the registers and the requested automatic storage.

Stack Frame Underflow (6): On a RET instruction, the previous save area was not in the same stack frame as the current one.

Arithmetic Overflow (8): Overflow occurred on an arithmetic instruction and was not masked out by the MSR. The resulting value was not stored.

Conversion Overflow (A): On a CVB instruction, the resulting number required more than 16 bits and the interrupt was not masked out in the MSR. The resulting binary number was not loaded into R1.

Division by 0 (C): An attempt was made to divide by 0.

Alignment (E): A halfword was not located on a even byte boundary where required.

Register Specification (10): Register 15 was specified in some double register operation.

Privilege (12): An I/O instruction was attempted with the privilege bit in the MSR off. See Section 8.

Stack Overflow (14): On a PSHM or PSHH/PSHHL, there was not enough room in the stack for the requested data, and the interrupt was not masked out in the MSR.

Stack Underflow (16): On a POPM or POPH/POPHL, there was not enough data in the stack to fill the pop request, and the interrupt was not masked out in the MSR.

Execute (18): The subject of an EX instruction was another EX. This situation cannot be handled by the firmware.

Invalid Unit Address (1A): Some error was detected in the unit address portion of an IOCC. See Section 8.

Invalid UCB Address (1C): The UCB address in the UCBT for the unit specified in an IOCC, or for the S/360, was 0 or odd. See Section 8.

I/O and S/360 interrupts are described in more detail in the next Section.

8 INPUT/OUPUT OPERATIONS

8.1 I/O UNITS

The META 4A can communicate with various external I/O units. These units consist of some number of local I/O units and the S/360.

8.1.1 LOCAL UNITS

The local I/O units which can be operated by the META 4A include a Disk, Card Reader, Keyboard/Typewriter, Control Panel, and META 4B (the graphics processor of BUGS). Up to a total of 11 local units can be supported by the firmware. The instructions used to perform I/O with these are SIOR and SIO.

8.1.2 THE S/360

The META 4A can also communicate with an IBM S/360. The S/360 regards the META 4A as a standard unit, performing I/O with it via a Multiplexor Channel and the standard IBM S/360 Interface. The instructions offered by the firmware to support S/360 communication are SS, TRB, OST, IST, and EXCC.

8.2 CONTROL OF LOCAL UNITS

8.2.1 UNIT INFORMATION

An I/O unit is designated by an I/O address. This address is a 5-bit binary number ranging from x0000 through x1010 (the first bit is ignored; 0 will be assumed here). Associated with each unit is a Unit Control Block (UCB). This storage block can contain all the information necessary to operate the unit, its status, I/O request queue, etc. Its format as regarded by the firmware is as follows:

```
  ┌────────────────┐
 0│00000000000xaddr│
  ├────────────────┤
 2│      USH       │
  ├────────────────┤
 4│                │
  │    variable    │
  │                │
  │      ...       │
```

The UCBs, because of their varying number and length, are not located in fixed memory locations. Instead, there exists the Unit Control Block Table (UCBT), which contains 12 pointers to UCBs. The first pointer corresponds to unit 00000, the second to unit 00001, etc. (hence, twice a unit's address is the UCBT offset of its UCB pointer), while the twelfth pointer is for the S/360 UCB. The UCBT occupies core locations 30 through 47. The only common UCB entries are the unit address and the Unit Status Halfword (USH). This halfword contains flags describing the current operating status of the associated unit (e.g., offline, busy, I/O error, etc.). The exact bit meanings will be described with each unit.

8.2.2 STARTING I/O

SIOR R1,R2

```
┌────────┬────┬────┐
│   16   │ R1 │ R2 │
└────────┴────┴────┘
 0        8    12 15
```

Start I/O
SIO R1,D2(X2,B2)

```
┌────────┬────┬────┬────┬──────────────┐
│   56   │ R1 │ X2 │ B2 │      D2      │
└────────┴────┴────┴────┴──────────────┘
 0        8    12   16   20          31
```

These instructions cause I/O to be initiated at a specified unit. In order to specify the type of I/O, a 2-halfword I/O Control Command (IOCC) is required. Operand 2 of SIO is this IOCC, while, for SIOR, the first halfword is in R1, and the second in R2. An IOCC has the following format:

```
┌───────────────────┬──────┬─────────┬────┐
|     address       |xaddr|    op    |mod|
└───────────────────┴──────┴─────────┴────┘
0                  16     21        2931
```

address: The use of this field varies from IOCC to IOCC. It normally specifies a memory address or is unused.

xaddr: This is the 5-bit unit address of the I/O unit on which the I/O is to be performed. If the UCB pointer in the UCBT for this unit is 0 or odd, an Invalid UCB Address Program interrupt will occur. Furthermore, if no unit with the specified address is installed, or if the unit address in the corresponding UCB does not match, an Invalid Unit Address interrupt will occur, with the unit address in error being stored in core location 50.

op: this 8-bit code specifies the type of I/O operation to be performed.

mod: this portion of the IOCC provides additional information for the unit and operation specified.

Once an IOCC is initiated, a new USH is automatically sensed and stored in the UCB. In addition, a bit is set in the Condition Code (the others being reset) as follows:

C0- Busy or Offline. The unit was found to be in Busy status or offline at the time of IOCC initiation.

C1- Unit Operating. An I/O operation was begun which will, upon completion at a later time, produce an I/O interrupt. The new USH will indicate Busy.

C2- Immediate Operation Complete. The operation performed completed immediately. The unit can now respond to another SIO(R).

An Alignment interrupt can occur on an SIO(R) for the following reasons:

1) The IOCC was not on a halfword boundary. This can only occur for SIO.

3) The address portion of the IOCC is to be used as a memory address, but is not on a halfword boundary.

A Privilege interrupt can also occur for both formats if the Privilege bit is not on; this prevents user programs from "accidentally" performing I/O.

8.3 I/O INTERRUPTS

At the completion of an interrupting I/O operation, for which C1 was set at SIO(R), or upon an externally caused interrupt, the operating unit requests an I/O interrupt. At this time, the META 4A performs the following functions:

1) Checks the I/O interrupt bit in the MSR. If off, the interrupt is kept pending until such time as the bit is set on.
2) Next checks the I/O Mask in location 2E. There exists one bit for each of the 11 possible I/O units, the unit address being the bit position. If the bit for the interrupting unit is off, the interrupt is again kept pending until such time as the mask bit is set on.
3) If the interrupt can be allowed, an I/O interrupt occurs with the interrupt code equal to the unit address (see Section 7). A new USH is stored in the UCB, unless the UCB address in the UCBT is odd or 0, in which case the USH is ignored <u>and</u> bit 0 of the I/O interrupt code is set on. Finally, the unit interrupt request is reset.

If two or more units compete for an interrupt at the same time, they are taken in priority order by ascending unit address.

8.4 I/O TO THE S/360

8.4.1 GENERAL INFORMATION

Viewed from either the META 4A or the S/360, I/O to the other processor differs from I/O to other units in several respects. While most I/O consists of a program issuing commands to a unit with a particular set of characteristics, S/360 I/O consists of a program in one processor communicating with another program in the other. Since this communication takes place over a S/360 channel, it is relatively simple and allows high data rates. However, it is also rigidly defined by the channel hardware, and deviations from the established protocol can have dire consequences for other devices sharing the channel and for CP. For this reason, before I/O between the S/360 and the META 4A is attempted, the descriptions of

I/O in the System/360 Principles of Operation manual should be thoroughly understood.

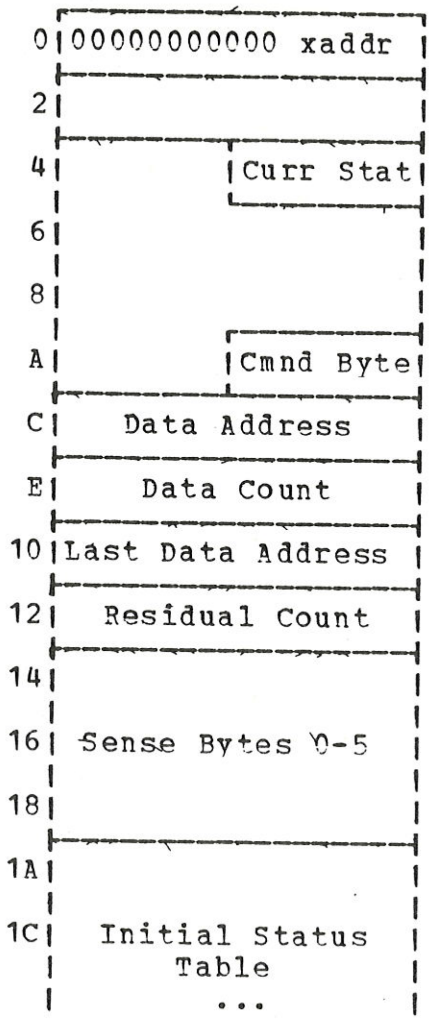
The META 4A is connected to the S/360 Multiplexor channel through an interface housed within BUGS. In the interface are the logic circuits necessary for address recognition, automatic propagation of Select Out, and command byte and status transfer. Two 16-bit registers in the interface reflect the current state of the tag and bus lines from the channel, along with certain other status flags. Although these registers can be directly controlled (via the IST and OST instructions), there will rarely be any need to do so.

8.4.2 S/360 INTERFACE ADDRESSES

The S/360 interface will respond to initial selections directed to any device address in the range 050 to 05F. However, only the first four of these (050 to 053) are treated as valid by the META4A, and I/O to an address higher than 053 should never be attempted. The four valid device addresses correspond to META 4A unit addresses 01011 through 01110, but these unit addresses are effectively ignored, and the device address is specified in the S/360 I/O instruction by the low order digit of the S/360 address (i.e., 0 to 3).

8.4.3 S/360 UCB FORMAT

Each of the S/360 UCBs has the following format:



Curr Stat: The current status byte for the device.
Cmnd Byte: The last (non-zero) command received from the channel.
Data Address: A pointer to the data area for the EXCC instruction.
Data Count: The number of bytes to be transferred by the EXCC instruction.
Last Data address: The address of the last data byte read or written by an EXCC instruction.
Residual Count: The count remaining if the EXCC instruction was prematurely terminated.
Sense Bytes 0-5: 6 sense bytes. (The firmware only references the first of these. The rest are software defined.)
Initial Status Table: There is one initial status byte for every possible command which can be received from the S/360. Note that the first byte i the

table (for a commad byte of 0) is not used. A 0 command (Test I/O) will cause the curret status byte to be sent as initial status.

8.4.4 FIRMWARE OPERATIONS

There are 3 basic functions which must be performed in handling the S/360 interface: initial status presentation, data transfer, and ending status presentation. These are now described:

8.4.4.1 Initial Status Presentation

When an Initial Selection occurs, the firmware examines the command byte, the UCB table, the UCB, and the MSR to determine what initial status should be sent to the S/360. The following situations can occur:

Test I/O and valid UCB pointer: The current status bye is used as iniial status and then reset to 0 (except possible for busy). Since no interrupt is generated, the state of the initial Select bit in the MSR is ignored.

Test I/O and invalid UCB address: An initial status of busy is sent.

Non-zero command and Initial Select disabled: initial status of Busy is sent.

Non-zero command and initial selection enabled: The UCB pointer for the appropriate device is checked. If it is valid (even and non-zero), the current status field is checked. If the current status field is non-zero, it is ORed with busy, sent as initial status, and then reset. If the current status field is 0, then initial status byte for the command is found in the initial status table and sent. The command byte is then stored in the UCB and a S/360 interrupt occurs, the interrupt code being the device address with bit 2 on.If the UCB pointer for the device is found to be invalid, unit exception is sent as initial status, and the device address with bit 0 on serves as the interrupt code.

8.4.4.2 Data Transfer

Once the software has been given an Initial Select interrupt, it must examine the device address and command fields of the UCB to determine what action to take. For operations involving data transfer between the META 4A and the S/360, a choice can be made between proceeding in "burst mode" or in "byte mode". In the first case, control is returned to the firmware via an EXecute Channel Command (EXCC) instruction. The firmware then retrieves a data address and count from the UCB and proceeds with the data transfer as indicated by the command byte which was previously stored in the UCB. On termination of the command, the condition code is set appropriately (see description of EXCC below), and the program is continued.

In byte mode, the software explicitly requests each data byte transfer. using the TRansfer Byte (TRB) instruction. While slower than the use of EXCC, this allows more flexibility in the placement of data in META 4A memory, for example.

8.4.4.3 Ending Status Presentation

At the termination of data transfer (due either to the count from the UCB reaching 0 or to Interface Stop), ending status must be sent to the channel with the Send Status (SS) instruction. Normal ending status consists of Channel End and Device End. Unusual conditions encountered during data transfer (such as Bus-Out Check) are indicated by ORing the appropriate bits in the status byte.

8.4.4.4 System or Selective Reset

If the S/360 performs a System or Selective Reset, a S/360 interrupt will unconditionally occur with an interrupt code of 2. Whatever instruction was executing on the META 4A at this time is <u>aborted</u>, thus this is an irrecoverable condition.

8.4.5 S/360 INSTRUCTIONS

Input/Output S or T register
IOST   R1,R2

20	R1	R2
0	8	12 15

This instruction is used for direct control of the interface S and T registers. The R1 field refers to the META 4A register which is to be snet to or loaded from the interface. The R2 field contains the operation to be performed, as follows:

0: The contents of S are placed into R1.
1: The contents of R1 are loaded into S.
2: The contents of T are placed into R1.
3: The contents of R1 are loaded into T.

Bits 12 and 13 of the R2 field are ignored.

EXecute Channel Command
EXCC   R1,R2

22	R1	R2
0	8	12 15

This instruction is used to perform block data transfers to and from the S/360. The address and length of this data are found in the S/360 UCB, for the device whose address is in the R1 field. Bit 7 of the Command byte determine the direction of data transfer: 1 means from the S/360, 0 means to it. If a Bus-Out Check is encountered during transfer, transfer is stopped and the error is recorded in bit 2 of Sense Byte 0 and in bit 6 (Unit Check) of the Current Status (Cur Stat). Transfer is also halted if Stop or Halt I/O is received from the channel. In whatever manner transfer is completed, the Final Data Address will be set to the address of the last byte transferred plus 1, while the Residual Count is set to the number of bytes remaining to be transferred. The Condition Code will be set as follows:

C0 - Transfer successful or Bus-Out Check.
C1 - Stop received.
C2 - Halt I/O received.

Privilege and Invalid UCB Address interrupts can occur.

TRansfer Byte
TRB R1,D2(X2,B2)

```
┌──────────┬─────┬─────┬─────┬───────────────┐
│    62    │ R1  │ X2  │ B2  │      D2       │
└──────────┴─────┴─────┴─────┴───────────────┘
0          8     12    16    20            31
```

This instruction operates exactly as EXCC except that the Data Count is assumed to be 1 and the byte to be transferred is found at the Operand 2 address. Because only 1 byte is transferred, the Final Data Address and Residual Count fields are not modified. Invalid UCB Address and Privilege interrupts can occur.

Send Status
SS R1,D2(X2,B2)

```
┌──────────┬─────┬─────┬─────┬───────────────┐
│    55    │ R1  │ X2  │ B2  │      D2       │
└──────────┴─────┴─────┴─────┴───────────────┘
0          8     12    16    20            31
```

This instruction causes a status byte to be sent from the device specified by the R1 field to the S/360. The status byte is either the low order byte of the second operand address or the current status field of the UCB. If the second operand address is non-zero, it is taken as the status byte, and the status is considered to be asynchronous (e.g., suppressable). If the address is 0, the current status field is examined. If this is also zero, C0 is set and nothing more is done. If the current status is non-zero, it is sent to the S/360. In either case, if status is stacked, the status byte is stored in the current status field of the UCB and C2 is set. (The program can then either try to re-issue the SS instruction or assume that the channel will eventually request the stacked status.) The Condition Code is set as follows:

C0 - Status accepted.
C1 - Status was suppressed or stacked.
C2 - Halt I/O received.

Privilege and Invalid UCB Address Program Interrupts can occur.

9 LOCAL I/O UNIT DESCRIPTIONS

9.1 3461 CARD READER (UNIT ADDRESS 00100) OBSOLETE

9.1.1 DESCRIPTION

The D.S.C. Model 3461 Card Reader provides punched card input for the META 4A processor. It is rated at a maximum throughput of 300 cards per minute, reading serially column by column. The 3461 reads cards in Hollerith image only; no code translation is provided by it or the firmware. A column image is stored in bits 0 through 11 of a halfword; bits 12 through 15 are set to 0.

9.1.2 I/O CONTROL COMMANDS

The 3461 is addressed by the 5-bit unit address 00100. It utilizes a single IOCC wich is an Initiate Read, to start up a card read cycle:

Initiate Read IOCC (X'2620')

```
┌─────────────────────┬──────┬──────────┬───┐
│ buffer address      │00100 │11000100  │///│
└─────────────────────┴──────┴──────────┴───┘
0                      16     21         2931
```

The address portion of the IOCC specifies the halfword memory address of the read buffer. The first halfword of the buffer designates the number of columns to be read. This count ("n" in the diagram below) is located in bit positions 9 through 15 and should never exceed 80 (information in succeeding memory locations will be destroyed). The halfwords following the count will receive the column data.

Buffer

```
     ┌──────────────────────┐
    0|/////////     n       |
     ├──────────────────────┤
    2|     column 1         |
     ├──────────────────────┤
    4|     column 2         |
     ├──────────────────────┤
    6|                      |

     |                      |
     ├──────────────────────┤
  2*n|     column n         |
     ├──────────────────────┤
     |                      |
```

Once the card reading operation is complete, the 3461 will request an Operation Complete I/O interrupt.

9.1.3 USH FORMAT

The format of the USH for the 3461 and the meanings of the bits are as follows:

```
|///|///|EC|LC|OC|///|///|///|/////////////|B |OB|
 0   1   2  3  4  5   6   7   8           13 14 15
```

EC:	Error Check. This indicates a feed check or a read check. It will appear along with OC.
LC:	Last Card. This bit comes on after the last card is read and will appear with OC. It remains on until more cards are placed in the hopper.
OC:	Operation Complete. This bit is set just prior to the request for an I/O interrupt after card reading is completed. It will appear in the USH that is stored in conjunction with the I/O interrupt.
B:	Busy. This indicates that a card read is in progress and therefore another read cannot be initiated. This indicator turns off when the Operation Complete interrupt occurs.
OB:	Offline or Busy. Offline is on when the Read Stop light is lit on the 3461. This bit is also on when the B (Busy) bit is on.

9.2 TERMINAL INTERFACE (UNIT ADDRESS 00110)

9.2.1 DESCRIPTION

The Terminal Interface provides a means of interactive communication between the META 4A processor and a terminal. Any terminal which conforms to the RS232-C conventions may be plugged into this interface. This unit also controls the black plexiglass box (hereafter called the magic box) connected to the terminal cable. The input and output codes are right justified ASCII.

9.2.2 CONTROL COMMANDS

The Terminal Interface is addressed by the 5-bit unit address 00110. The control IOCC uses a set of status codes to control the ablitity of the Terminal Interface to interrupt the CPU, and to control the lights on the terminal and the magic box.

Control IOCC (X'3430')

```
┌────────────────────────┬─────┬────────┬───┐
|     status codes       |00110|10000110|000|
└────────────────────────┴─────┴────────┴───┘
0                        16    21       2931
```

Status Codes Format:

```
┌──┬──┬──┬──┬──┬──┬──┬──┬────────────┬──┬──┬──┐
|OC|IR|//|//|//|//|//|//|PR////////FL|LL|RL|
└──┴──┴──┴──┴──┴──┴──┴──┴────────────┴──┴──┴──┘
0  1  2  3  4  5  6  7  8            13 14 15
```

OC: Output Complete Enable. If this bit is on, the output complete interrupt will be enabled.

IR: Input Complete Enable. If this bit is on, the input complete interrupt will be enabled.

PR: Proceed. If the terminal has a wait/proceed light, this bit will control its status.

FL: Flash Light. If on, the light on the left side of the magic box will flash.

LL: Left Light. If this bit is on, the left hand light on the magic box will be on.

RL: Right Light. If this bit is on the light on the right side of the magic box will be on.

Read Command

When the user stikes a key on the terminal, an Input Ready interrupt will occur if a previous Control IOCC enabled the interrupt. At this time, a Read IOCC should be performed to store the ASCII character into a halfword in core. This IOCC has the following format:

Read IOCC (X'3208')

```
┌──────────────────┬─────┬────────┬───┐
|  memory address  |00110|01000001|000|
└──────────────────┴─────┴────────┴───┘
0                  16    21       2931
```

Output Command

The Write IOCC will write one ASCII character to the terminal plugged into the interface.

Initiate Write IOCC (X'3130')

```
┌──────────────────┬─────┬────────┬───┐
| ASCII character  |00110|00100110|000|
└──────────────────┴─────┴────────┴───┘
0                  16    21       2931
```

9.2.3 USH FORMAT

The format of the USH for the Terminal Interface and the meanings of the bits are as follows:

```
┌──┬──┬──┬──┬──┬──┬──┬──┬─────────────┬───┬───┐
|OC|IR|BR|AT|TB|TO|PE|OR|/////////////|///|///|
└──┴──┴──┴──┴──┴──┴──┴──┴─────────────┴───┴───┘
0  1  2  3  4  5  6  7  8             13  14  15
```

OC: Output Complete. This bit is set when the interface is ready for another character. Following the setting of this bit, an I/O interrupt is requested by the Terminal Interface.

IR: Input Ready. This bit comes on after the user has hit a key and just prior to the request by the interface for an I/O interrupt. A Read IOCC should be performed after receiving this interrupt.

BR: Break. This bit is set when the user strikes the BREAK key on the terminal. It is provided as a

	means of signalling an attention to the META 4A. An I/O interrupt will occur after the setting of this bit.
AT:	Attention: This bit is set when the button on the magic box is pressed. An I/O interrupt will be requested after this bit is set.
TB:	Terminal Busy. When on, this indicates that the interface is busy transmitting a character.
TO:	Terminal Offline or Busy. This bit is off when the terminal is powered on and plugged into the interface.
PE:	Parity Error. This bit is set when a parity error is detected in a received character.
OR:	Overrun. This bit is set when the interface detects an overrun condition.

9.3 /360 TERMINAL INTERFACE (UNIT ADDRESS 00011)

9.3.1 DESCRIPTION

The /360 Terminal Interface is almost identical to the interface described in 9.2 above. It provides an interface to the /360 so that BUGS can look to CP as though it were a terminal. Only the differences in operation from the description above will be shown here.

Control IOCC (X'1C30')

```
┌─────────────────┬─────┬────────┬───┐
|     status codes|00011|10000110|000|
└─────────────────┴─────┴────────┴───┘
0                 16    21       2931
```

The status codes are identical to those described in , with the exception that turning bit 2 on in the status codes will transmit a BREAK to the /360.

Read IOCC (X'1A08')

```
┌─────────────────┬─────┬────────┬───┐
| memory address  |00011|01000001|000|
└─────────────────┴─────┴────────┴───┘
0                 16    21       2931
```

Initiate Write IOCC (X'1930')

ASCII character	00011	00100110	000
0	16	21	29 31

9.4 MODEL 1444 DISK STORAGE UNIT (UNIT ADDRESS 00010)

9.4.1 DESCRIPTION

The Model 1444 Disk Storage Unit provides the META 4A Computing System with a large(?) sequential or random access data store. The features of this store are described in the following paragraphs.

9.4.1.1 Storage Capacity

The storage capacity provided by the 1444 is 1,024K (1,048,576) bytes per disk cartridge. A single cartridge may be mounted on the drive at any one time; but cartridges may be interchanged in a matter of minutes.

9.4.1.2 Data Organization

The disk access mechanism, called the carriage, is moved back and forth by programmed I/O Control Commands and can be placed over any one of 203 cylinders, numbered from 0 to 202. Each cylinder is divided into 8 sectors, numbered 0 to 7. Sectors 0 through 3 are located on the upper surface of the disk, while sectors 4 through 7 are on the lower surface. Each sector contains 642 bytes of information and is the largest segment of data that can be read/written with a single IOCC.

$203 \times 8 \times 642 =$ ~~[illegible]~~
1042608

9.4.2 1444 I/O CONTROL COMMANDS

The 1444 is addressed by the unit address 00010, and is controlled by 4 IOCCs. The first of these is a Control, and is used to position the carriage over one of the 203 cylinders of the cartridge. Its format is:

Control IOCC (X'1430')

```
┌────────────────┬─────┬────────┬───┐
|///////increment|00010|10000110|D//|
└────────────────┴─────┴────────┴───┘
0                16    21       2931
```

The increment specifies the number of cylinders to move the carriage relative to its current position. If bit 29 (D) is 0, the carriage is moved toward the higher cylinders (the center of the cartridge); if 1, it is moved toward the lower cylinders (the edge of the cartridge). When movement is complete, an Operation Complete interrupt occurs, unless the increment was 0. Because movement is relative, it is necessary to keep in the UCB the current cylinder position, so that increments can be computed correctly.

The second IOCC is an Initiate Write, used for recording information on a particular sector of the current cylinder. Its format is:

Initiate Write IOCC (X'1520')

```
┌────────────────┬─────┬────────┬───┐
| memory address |00010|10100100|sec|
└────────────────┴─────┴────────┴───┘
0                16    21       2931
```

The memory address specifies the address of a buffer on a halfword boundary from which data is written on the sector specified by the 3-bit number sec. The buffer has the following format:

Buffer

```
    ┌────────────────────┐
   0|   count     n      |
    ├────────────────────┤
   2|    column  1       |
    ├────────────────────┤
   4|    column  2       |
    ├────────────────────┤
   6|                    |

    |                    |
    ├────────────────────┤
 2*n|    column  n       |
    ├────────────────────┤
    |                    |
```

The count specifies the number of halfwords of data to be written on the sector, and is followed by the halfwords themselves. If less than 642 bytes (321 halfwords) is written, the remaining bytes on the sector are zeroed. An Operation Complete interrupt occurs after the Write operation is finished.

Finally, there are 2 Initiate Read IOCCs. The first of these is used to read all or part of a sector into memory. Its format is:

Initiate Read IOCC (X'1620')

```
┌────────────────────┬─────┬────────┬───┐
| memory address     |00010|11000100|sec|
└────────────────────┴─────┴────────┴───┘
0                    16    21       2931
```

The memory address points to a buffer (in the same format as that for Initiate Write) which specifies a count of the number of halfwords to be read from the sector specified by the 3-bit number sec. An Operation Complete interrupt occurs after data transfer is complete.

The second of these, Initiate Read-Check, performs exactly the same function as Initiate Read, except that no data is stored in memory. Therefore, no data halfwords need folllow the count in the buffer. A Read-Check is used to check the validity of data written on the disk and should follow every Write if maximum reliablilty is to be maintained.

Initiate Read-Check IOCC (X'16A0')

```
┌──────────────────┬─────┬────────┬───┐
| memory address   |00010|11010100|sec|
└──────────────────┴─────┴────────┴───┘
0                  16    21       2931
```

NOTE: If a count of 0 is specified on any of the last 3 IOCCs, unpredictable results will occur.

9.4.3 USH FORMAT

The format of the USH for the 1444 is as follows:

```
┌──┬──┬──┬──┬──┬──┬──┬──┬─────────────┬──┬──┐
|E |OC|OB|B |C0|//|//|//|/////////////|SC|SC|
└──┴──┴──┴──┴──┴──┴──┴──┴─────────────┴──┴──┘
0  1  2  3  4  5  6  7  8             13 14 15
```

E: Error. This bit can appear at Operation Complete for the following reasons:
1) A data error occurred on an Initiate Read or Read-Check.
2) More than 642 bytes (321 halfwords) were specified by the count field. A sector is the maximum segment that can be transferred by 1 IOCC.
3) The carriage was moved to cylinder 0 or 202 by a Control IOCC, but the increment specified still more movement. The current D.S.C. 1444 controller circuitry goes berserk when this situation occurs.

OC: Operation Complete. This bit will appear in the USH after an Operation Complete interrupt.

OB: Offline or Busy. This bit is on when the 1444 drive is powered down, not up to speed, or when the unit is busy executing a previous IOCC.

B: Busy. This bit is on when the unit is busy executing a previous IOCC. OB will also be on in this case.

C0: Carriage at 0. This bit is always down unless the carriage is positioned over cylinder 0.

SC: Sector Count. These 2 bits specify the sector number (modulo 4) of the next sector to pass under the Read/Write Heads on the carriage. If a program can be arranged to read/write either of these sectors next, rotational delay will be minimized.

9.5 META 4A CONTROL PANEL (UNIT ADDRESS 00101)

9.5.1 DESCRIPTION

The Control Panel contains the switches, dials, and lights necessary to operate and control the META 4A and the whole BUGS system. In addition it is possible for a running program to perform limited I/O to the Panel.

9.5.2 SYSTEM CONTROL FUNCTIONS

POWER ON (black): This key is pressed to initiate a power-on sequence of the whole BUGS system.

POWER OFF (red): The Power Off key is pushed to initiate the power-off sequence of BUGS. The 1444 disk should be in SAFE state when this key is pressed.

STOP: This button causes all processing to stop after the completion of the current instruction. The META 4B is signalled to stop also, but other initiated I/O operations are allowed to come to completion. It is necessary for the META 4A to be in this "stopped" state if any Panel manipulations are to be performed.
NOTE: If S/360 Initial Select interrupts are disabled <u>and</u> the META 4A is online, the STOP button is ignored. This is to prevent Multiplexor Channel and CP crashes.

RESET: This button resets the META 4A and all online local I/O units; in other words, the whole BUGS system. All pending interrupts are cleared, Parity Checks are corrected, and the system is placed in an initialized state. The contents of the memory and registers after RESET are unpredictable.

IPL: This button provides a means of initially loading a program into memory and beginning execution. Before initiating an IPL, BUGS should be reset and the data switches set to specify from where the initial program should be obtained:

all off: bytes 2 through 641 of cylinder 0, sector 0 on the 1444.

switch 0 up: bytes 2 through 641 of cylinder 202, sector 0 on the 1444.

switch 1 up: a variable number of bytes from the S/360.

The IPL sequence involves lighting the IPL-ING light, loading the initial program starting at location 0, and obtaining the initial MSR and PC from locations 0 through 3 to start execution. Once the IPL is complete, the IPL-ING light is extinguished, unless an error occurs. If the light remains on, IPL should be re-attempted.

IPL-ING light: This light is on during an IPL sequence, as described above.

CPU RUN light: This light is on when the META 4A CPU is running. The CPU is considered to be running at all times except during panel manipulations performed after the STOP button is pressed.

WAIT light: This light is on when the META 4A CPU is running but the MSR indicates wait state.

9.5.3 USER INTERVENTION FACILITIES

Whenever the BUGS system has entered "stopped" state due to the STOP button being pressed, the user may execute one or more of the above control functions. In addition the Panel provides him with facilities to display and modify various components of the META 4A.

Displaying and Loading the MSR: The user can display the current contents of the MSR on the lower lights by setting the REGISTER SELECT (RS) dial to MSR and the MODE SELECT (MS) dial to DISPLAY. Furthermore, by turning the MS dial to LOAD, he may modify these contents by setting the desired halfword on the 16 data switches. The CLEAR switch will at all times zero the contents of the register currently being loaded.

Displaying and Loading the PC: This operation is completely synonomous to that for the MSR, except that the RS dial is set to the PC postion.

Displaying and Modifying the Next Sequential Instruction (NSI): The NSI is the first halfword of the instruction currently pointed to by the PC. This halfword may be manipulated exactly as the MSR and PC by setting the RS dial to NSI.

Displaying and Modifying Memory: In order to select a memory location for displaying or modification, a special register, called the Display Counter, is used. The RS dial has a COUNTER position, so that this register can be treated just as the MSR, PC and NSI explained above. Following the setting of the Display Counter to the desired address, the RS dial is turned to MEMORY DATA, and the halfword pointed to by the Display Counter is shown on the lower lights, with the Display Counter itself on the uppers. The memory halfword may then be modified by the normal procedure. If the START switch is depressed during a memory display, the Display Counter is incremented by 2 and the next halfword displayed.

Displaying and Modifying Registers: As you may have guessed, this operation is identical to that for memory, except that the RS dial is turned to REGS. Only bit 12 through 15 of the Display Counter are used to select the register.

Clearing Memory: A firmware memory clear may be performed by setting the RS dial to MEMORY DATA, the MS dial to LOAD, and depressing the IPL, CLEAR, and START buttons simultaneously. Memory is completely zeroed in approximately one-half second.

Starting the META 4A after Stopping: This is accomplished by setting the MS dial to STEP, RUN or INT RUN (described below), and pressing the START switch. RUN causes normal CPU execution to continue, while STEP causes "stopped" state to be re-entered after execution of for debugging system programs "interactively".

9.5.4 I/O TO THE CONTROL PANEL

9.5.4.1 Panel IOCCs

The Panel is addressed by the unit number 00101. 2 IOCCs are available to the programmer - the first of these is used to read the setting of the 16 data switches:

Read IOCC (X'2A08')

```
┌─────────────────┬─────┬────────┬───┐
| memory address  |00101|01000001|///|
└─────────────────┴─────┴────────┴───┘
0                 16    21       2931
```

The halfword switch setting is stored at the memory address specified.

The other IOCC is used for setting the lights to a certain value. This can be useful for operating system error halts or what have you. Its format is: Write IOCC (X'2800')

```
| memory address |00101|00000000|/LU|
 0                16    21       2931
```

The halfword at the memory address is set on the lower lights if L is on, the uppers if U is on, or both if both modifiers are set.

9.5.4.2 USH Format

The format of the USH for the Panel is:

```
|IR|IB|P |//|//|//|//|//|/////////////|//|//|
 0  1  2  3  4  5  6  7  8            13 14 15
```

IR: This bit is set after a Panel interrupt caused by running in INTerrupt RUN mode. This mode of execution causes an interrupt after every instruction.

IB: This bit will be set after a Panel interrupt caused by hitting the INTERRUPT button.

P: This bit is set after a Panel interrupt caused by a Parity Check. The PAR STOP light is automatically set by the firmware when this check is detected, and an interrupt then occurs, unless disabled by the MSR.

9.6 META 4B (UNIT ADDRESS 00001)

9.6.1 DESCRIPTION

The META 4B is the graphics processor of the BUGS configuration - a CPU of the same power as the META 4A, but with target instructions oriented towards graphic data manipulation and display. A complete description can be found in the META 4B Principles of Operation. This section describes the IOCCs used in communication between the 2 processors. It should be realized that the META 4B can perform equivalent operations (although they are programmed differently), thus completing the communications link.

9.6.2 I/O CONTROL COMMANDS

There are 3 IOCCs used to communicate with the META 4 B. These IOCCs manipulate registers in a special inter-processor interface, The Inter-Processor Interrupt (IPI). The IPI is addressed as unit 00000 The first IOCC is a Write:

Write IOCC (X'0930')

```
+-----------------+------+----------+----+
| /////////////// |00001 |00100110  |/// |
+-----------------+------+----------+----+
0                 16     21         2931
```

This IOCC sends an interrupt request to the META 4B.

The second IOCC is a Read, whose function is to allow the META 4A to read back its own USH. Its format is:

Read IOCC (X'0208')

```
+-----------------+------+----------+----+
| memory address  |00001 |01000001  |/// |
+-----------------+------+----------+----+
0                 16     21         2931
```

The USH is placed in the halfword at the memory address.

The final IOCC, a Control, is used by the META 4A to reset bits in the B's USH:

Control IOCC (X'0430')

```
+-----------------+------+----------+----+
|1111111111111111 |00001 |10000100  |/// |
+-----------------+------+----------+----+
0                 16     21         2931
```

This IOCC acknowledges an interrupt from the META 4B.

9.6.3 TYPICAL COMMUNICATION SEQUENCE

Let us assume that the META 4B wants to interrupt the META 4A. The following sequence of events might occur:

META 4B: Initiates the equivalent of a Write to set its USH to the interrupt code and request an interrupt at the A.

META 4B: Goes into a loop reading its USH and testing for 0 to wait for the A to acknowledge the interrupt.

META 4A: Receives the interrupt and so as to avoid a continuous interrupt request, Controls the actual META 4B USH (stored in the UCB upon interrupt) to the B so as to clear its USH and the interrupt request.

META 4A: Branches off on the USH code to perform requested function.

META 4B: Drops out of the read loop and continues operation.

NOTE: The <u>only</u> way to clear an interrupt request is to zero the META 4B's USH by a Control IOCC. Not even the B can clear its request (by writing a USH of 0, for example).

<u>9.7 COMMON I/O CONTROL COMMANDS</u>

9.7.1 NO OPERATION

The No Operation IOCC is common to all I/O units. It performs no I/O at the unit and therefore completes immediately with Condition Code C2. It does, however, (as do all SIO(R)/IOCC combinations) cause the post-SIO(R) USH to be stored in the UCB, and can optionally be used to clear a pending I/O interrupt.

No Operation IOCC

```
┌────────────────┬─────┬────────┬───┐
|////////////////|xaddr|11100011|//C|
└────────────────┴─────┴────────┴───┘
0                16    21       2931
```

If bit 31 (C) is on, and an interrupt from the addressed unit has been requested but kept pending due to the MSR or

I/O Mask, that interrupt request is cleared. This bit is ignored when the META 4B is addressed.

9.7.2 INVALID I/O CONTROL COMMANDS

If an IOCC with an invalid operation code is initiated, unpredictable results will occur at the I/O unit, along with possible spurious I/O interrupts. Such operation codes should therefore never be used for any purposes whatsoever.

10 APPENDIX I: OPERATION PROGRAM INTERRUPT STORAGE

As mentioned in Section 7, certain extra information is stored in low core when an Operation Program interrupt occurs. This information facilitates simulating extended machine instructions in software. There are 5 halfwords of information stored in memory locations 52 through 5B; the exact nature of the halfwords stored depends on the format of the invalid instruction as shown in the following table:

				LOCATION		
First hex digit in opcode	FORMAT	52	54	56	58	5A
0,1,2,3	RR	First halfword of instruction		4*R1	4*R2	
B	RI	"		4*R1S		Operand 2 (immediate data)
9	RS	"		4*R1	Operand 2 address	
4,5,6,7	RX	"		4*R1	Operand 2 address	
A	BX	"		4*R1	Operand 2 (branch) address	
8	SI	"		Operand 1 address		Operand 2 (immediate data) in locaton 51 (0 in 50)
E,F	FSS	"		Operand 1 address	Operand 2 address	
C,D	VSS	"	length	Operand 1 address	Operand 2 address	

NOTE: Blank entries in the table signify that the contents of the location are undefined, or too ridiculous to mention.

11 APPENDIX II: OPERATION CODES AND TIMINGS

11.1 OPERATION CODES AND INSTRUCTION EXECUTION TIMES

Instruction	Code	Execution Time (in mics.)
A	74	4.62 (assuming no overflow)
AH	F4	7.62
AHI	84	5.37
AI	B4	3.54
AL	7C	4.77 (average)
ALI	BC	3.69
ALR	3C	3.36
AR	34	3.21
AX	24	3.30
BAL	A8	3.75
BALR	28	3.42
BCO	A7	3.65
BCOR	27	3.32
BCT	AF	3.99
BCTR	2F	3.66
BCZ	A6	3.65
BCZR	26	3.32
BXH	A0	5.25
BXLE	A1	5.25
C	70	5.14 (average)
CH	F0	7.42
CHI	80	5.38
CI	B0	3.96
CL	7F	4.83
CLB	5F	4.89
CLBI	8F	4.83
CLC	CF	7.56 + 2.04N (N is the number of characters compared)
CLCL	DF	7.83 + 2.04N
CLH	FB	7.11
CLI	BF	3.75
CLR	3F	3.42
CR	30	3.53
CVB	50	5.43 + 1.80N (N is the number of characters converted)
CVD	51	17.75 (average)
D	77	12.75
DEQ	48	7.08
DH	F7	16.71
DI	B7	12.09

DR	37	11.34
ENQ	F8	9.24
ENT	BE	23.88
EX	54	5.01 + normal time of subject instruction (R1=0)
		5.10 + normal time of subject instruction (R1≠0)
EXCC	22	
FILL	C8	7.11 + 2.04L (L is the number of bytes specified by the length field.)
FILLL	D8	7.38 + 2.04L
IB	58	4.71
IST	20	3.45 (average - one register 0)
L	73	4.32
LA	5D	4.29
LB	5A	4.41
LC	4E	4.92
LCR	0E	3.63
LD	53	4.92
LDI	B8	4.03
LDR	13	3.63
LI	B3	3.24
LM	93	5.64 + 1.02R (R is the number of registers specified)
LMD	91	5.85 + 1.02R
LN	4D	4.44 (if already negative)
		4.62 (if positive)
LNR	0D	3.15
		3.33
LP	4C	4.56
		5.04
LPR	0C	3.27
		3.75
LR	33	2.91
LSB	5B	4.80
LX	23	3.09
LXB	4F	4.41
LXBR	0F	3.12
LZ	5E	5.25
M	76	10.77
MH	F6	13.22
MI	B6	9.81
MR	36	9.36
MVA	62	7.06
MVBI	8C	5.61
MVC	C4	7.32 + 1.08L (both addresses even) 2.04
		7.23 + 3.06L (otherwise)
MVCL	D4	7.59 + 1.08L

		7.50 + 3.06L
MVCN	C5	7.53 + 1.08L (address1 < address2 and both even)
		7.80 + 3.06L (otherwise)
MVCNL	D5	7.80 + 1.08L
		8.07 + 3.06L
MVH	F3	7.41
MVHI	83	5.37
N	79	4.80 (average)
NBI	89	6.24
NC	C9	7.11 + 3.06L
NCL	D9	7.38 + 3.06L
NI	B9	3.72
NR	39	3.39
O	7A	4.89
OBI	8A	6.33
OC	CA	7.11 + 3.06L
OCL	DA	7.38 + 3.06L
OI	BA	3.84
OR	3A	3.51
POPH	C7	9.03 + 1.08L (L is the number of bytes specified by the length field)
POPHL	D7	9.30 + 1.08L
POPM	97	7.53 + 1.02R (R is the number of registers specified)
PSHH	C6	9.51 + 1.08L
PSHHL	D6	9.78 + 1.08L
PSHM	96	7.35 + 1.14R
RET	0B	21.99
S	75	4.72
SA	5C	4.50
SD	17	8.70
SH	F5	7.71
SHI	85	5.58
SI	B5	3.75
SIO	56	10.7 (approximate average)
SIOR	16	8.5
SL	7D	4.86
SLA	9C	6.72 + .30B (B is the number of bit positions specified)
SLAI	1C	5.10 + .30B
SLDA	9E	7.02 + .30B
SLDAI	1E	5.40 + .30B
SLDL	9A	6.93 + .30B
SLDLI	1A	5.31 + .30B
SLE	C0	7.29 + 1.02N (N is the number of bytes searched in the first operand)
SLEL	D0	7.56 + 1.02N
SLI	BD	3.90

SLL	98	6.63 + .30B
SLLI	18	5.01 + .30B
SLNE	C2	7.29 + 1.02N
SLNEL	D2	7.56 + 1.02N
SLR	3D	3.57
SR	35	3.42
SRA	9D	6.63 + .21B
SRAI	1D	5.01 + .21B
SRCH	F9	9.75 + 1.25N (see later note)
SRDA	9F	6.84 + .21B
SRDAI	1F	5.22 + .21B
SRDL	9B	6.75 + .21B
SRDLI	1B	5.13 + .21B
SRE	C1	7.20 + 1.02N
SREL	D1	7.47 + 1.02N
SRL	99	6.54 + .21B
SRLI	19	4.92 + .21B
SRNE	C3	7.20 + 1.02N
SRNEL	D3	7.47 + 1.02N
SS	55	
ST	72	5.16
STB	59	5.40
STD	52	5.37
STDI	B1	4.41
STDR	12	4.08
STL	CC	7.29 + 2.04N
STLL	DC	7.56 + 2.04N
STM	92	5.70 + 1.14R
STMD	90	5.91 + 1.14R
STR	CD	7.20 + 2.04N
STRL	DD	7.47 + 2.04N
SVC	31	8.70 (This reflects the time until the first instruction of the SVC interrupt handler is fetched.)
SVCD	F1	12.33
SVCS	81	10.11
SWP	78	5.16
SWPR	38	3.63
SX	25	3.51
TBM	AC	3.63
TBMR	2C	3.30
TBNM	AB	3.63
TBNMR	2B	3.30
TBNP	AE	3.63
TBNPR	2E	3.30
TBNZ	AD	3.63
TBNZR	2D	3,30
TBP	A9	3.63
TBPR	29	3.30
TBZ	AA	3.63

TBZR	2A	3.30	
TMBI	82	4.83	(average)
TMI	B2	3.75	
TMR	32	3.42	
TR	CE	7.02 + 3.18L	
TRB	62		
TRL	DE	7.29 + 3.18L	
TSL	87	5.94	
X	7B	4.80	
XBI	8B	6.42	
XC	CB	7.11 + 3.06L	
XCL	DB	7.38 + 3.06L	
XI	BB	3.84	
XR	3B	3.51	

The times shown reflect instruction fetching and parsing as well as individual execution time. For instructions like ST, which perform a memory operation late in the execution routine, the time shown includes the full memory cycle.

A non-0 base register is assumed wherever appropriate. Deduct .09 microseconds for each 0 base register. For RX and BX instructions, a 0 index register is assumed. Add .42 microseconds for each non-0 RX index and .39 microseconds for each non-0 BX index. For branches, the time shown is the average for a successful and an unsuccessful branch. The execution time for any instruction may be lengthened if a cycle-stealing I/O operation is in progress.

For SRCH, N is the number of memory reads that must be performed during searching. This is defined as follows: There is one memory read for each forward pointer fetched. For a TM search, there is one memory read for fetching the key in each entry. The argument is <u>not</u> refetched each time. For each entry in a CLC search, the argument and the key are each fetched one byte at a time until the strings are discovered to be unequal, or the key length is exhausted. One memory read is required for each byte fetched.

<u>11.2 OTHER TIMINGS</u>

A local I/O interrupt takes 9.60 microseconds between detection and the fetch of the first instruction of the I/O interrupt handler. A disabled local I/O interrupt adds .51 microseconds to each instruction.

An Operation Program interrupt takes 11.58 microseconds between the end of its parsing and the fetch of the first instruction of the Program interrupt handler. An invalid unit address or UCB address interrupt takes 6.48 microseconds between detection and the fetch of the first instruction of the Program interrupt handler. Any other program interrupt takes 6.18 microseconds after detection.

The times for instruction fetch and parsing alone are:

<u>Format</u>	<u>Time (in mics.)</u>
RR	2.61
RI	2.94
RX	3.90
RS	4.32
SI	3.93
BX	3.15
FSS	6.06
VSS	6.69
Long VSS	6.96

The assumptions made in earlier timing figures concerning base and index registers also apply here.

11.3 COMMENTS AND PROGRAMMING HINTS

To zero one or more registers, a sequence of instructions like:

```
        LY        R3,0
        LR        R4,R3
        LR        R5,R3
                .
                .
                .
```

should be used. Do <u>not</u> use a sequence of SRs.

For SRCH, note that a TM is significantly faster than a CLC.

Logical compares are faster than corresponding arithmetic compares and thus should be used to test for equality whenever it is not important to distinguish between arithmetically greater and arithmetically less.

For loading a constant into a register, note that LX < LI << LA.

For adding a constant to a register, note that AX < AI < ALI < LA. Arithmetic adds and subtracts are faster than logical adds and subtracts.

Remember that LDR and STDR take less core and less time than L and ST and thus should be used whenever the address of the operand is in a register. LDI and STDI are also faster than L and ST and can be used whenever indexing is not required. However, they are less easily relocatable than L and ST.

To determine if an area of core contains all zeroes, do _not_ use an NC or OC. Use SRNE to an X'00' and then check the Condition Code.

Wherever it is certain than two string will not overlap, use MVC instead of MVCN. Not only is setup time faster, but there is a greater chance that the string will be moved a halfword at a time.

Doing a LI into the PC is the fastest way of executing an unconditional branch.

12 APPENDIX IV: LAYOUT OF LOWER MEMORY

The following is a map of the various firmware-defined locations in lower memory:

Location	Contents
0	IPL new MSR
2	IPL new PC
4	SVC old MSR
6	SVC old PC
8	SVC interrupt code
A	SVC new MSR
C	SVC new PC
E	Program interrupt old MSR
10	Program interrupt old PC
12	Program interrupt code
14	Program interrupt new MSR
16	Program interrupt new PC
18	I/O interrupt old MSR
1A	I/O interrupt old PC
1C	I/O interrupt code
1E	I/O interrupt new MSR
20	I/O interrupt new PC
22	S/360 interrupt old MSR
24	S/360 interrupt old PC
26	S/360 interrupt code
28	S/360 interrupt new MSR
2A	S/360 interrupt new PC
2C-2D	reserved for use by firmware
2E	I/O interrupt mask
30	Interval Timer UCB address
32	META 4B UCB address
34	1444 Disk Storage Unit UCB address
36	
38	3461 Card Reader UCB address
3A	Control Panel UCB address
3C	4132 Keyboard/Typewriter UCB address
3E-44	
46	S/360 device 050 UCB ADDRESS

48	S/360 device 051 UCB address
4A	S/360 device 052 UCB address
4C	S/360 device 053 UCB address
4E	
50	Interval Timer
52-5B	Program interrupt scan-out area - 5 halfwords

Acknowledgements

We would like to thank the following people for their help in designing and implementing the META 4A:

Wolfgang W. Millbrandt
(We are pleased to Mr.) Robert G. Munck
William B. Rothman
George M. Stabler
John E. Stockenberg
Richard C. Waters

And a special thanks to all those dear friends in Holland, without whom register 0 would have been fixed at 0.

Paul Constantine Anagnostopoulos
Gary Howard Sockut