



**Division** MISSILE SYSTEMS  
**Operation** Bedford Laboratories  
**Department** Data Processing Systems

**Contract No.**

**Distribution** aa

**To** Alan Deerfield (904)

**File No.**

**From** Steven Wallach (904)

**Memo No.** SW-71-7

**Subject** CMS-2 I/O

**Date** 13 July 1971

Enclosure: Appendix A (2 pages)

1.0 INTRODUCTION

The following is a description of the CMS-2 I/O system as viewed from the compiler. The I/O drivers and peripheral control programs will not be mentioned, other than the interfaces provided for them by the CMS-2 I/O System. In reality, I/O requires coordinated compiler and I/O executive systems. Lacking information on the MEC (Master Executive System) I/O system and areas where interfaces are required, assumptions and brief specification of this common area will be made. Also, taking advantage of this lack of specification, different philosophies will be expoused wherever possible. Tradeoffs will be indicated wherever possible.

The reading of this document will be made much easier if the reader has first read: "CMS-2, SW-71-6; 11 June 1971". Some of the concepts used in this memo are introduced and explained in the referenced document.

This memo is based on the material presented in: "CMS-2 Manual, Volume 1, 9 June 1969, pages I-4-34 thru I-4-61". Appendix A lists new CMS-2 Instructions Defined.



### 1.1 CMS-2 I/O Orientation

CMS-2 I/O is structured around files and their manipulations. In the programmer's manual, files are said to be stored on magnetic tape. It is quite probable that future applications will find I/O centered around random access devices, mainly disk drives. This in itself presents no problem, since disk drives can be logically accessed as tape drives, thereby affording the same I/O interface to the programmer. However, additional CMS-2 I/O statements may be desirable to reference disk drives and thereby taking advantage of the inherent benefits over tape drives.

There are two types of CMS-2 files: logical and physical. Physical files are portions of a magnetic tape that are delineated by a hardware recognizable end-of-file mark. A logical file is one or more physical files. A logical file is defined by the FILE declaration. CMS-2 I/O primitives handle the different types of files.

### 1.2 CMS-2 I/O

The following sections directly deal with CMS-2 I/O. However, before proceeding with the handling of these I/O statements, some mention must be made concerning program linkage with the I/O drivers of the executive system. The AADC will presumably be a multiprogrammed system. As such, when an I/O task is invoked, the executive assumes control. The executive must first ascertain the reason for it being invoked, and then carry out the necessary tasks. Since I/O is typically the slowest computer operation and since data requested is buffered, a task swap usually occurs when I/O is invoked. The task invoking the I/O is temporarily suspended until the I/O is complete and a new task becomes resident in the

processing element. Since the CMS-2 compiler is somewhat autonomous from the executive system, means must exist to provide communications between the two. For this purpose a SYSTEM ORIGIN is defined. The System Origin contains information and pointers to information that is necessary for executive system/compiler communications. Since each task's data design is stored in different areas in memory, a fixed area must be defined where linkage information is passed. The linkage area can be likened to a mailbox. It is always in the same location. It is the drop-off point for data.

For I/O purposes, the following system origin mailboxes are defined.

FIRST WORD OF FILE DELINEATION
RECEPTACLES/IMMEDIATE OPERAND
TASK CONTROL BLOCK POINTER
I/O STATUS WORD
INTERRUPT MASK
FORMAT CONTROL POINTER

#### SYSTEM ORIGIN

##### 1.2.1 File Word

This mailbox contains the first word of the delineation of the file declaration involved in the I/O operation. See Section 2.0 for the handling of the FILE declaration.

##### 1.2.2 Receptacles/Immediate Operand

The execution of an I/O statement results in the transmission of data to/from a CMS-2 data element(s). This mailbox

contains a structor pointing to the data element to be effected, or a linked list of structors. A linked list of structors is necessary when more than one data element stored in noncontiguous memory locations is manipulated. CMS-2 I/O is buffered. Therefore, it is necessary to interrogate the receptacle mailbox when writing into the buffer on a write operation where a data design to buffer move is initiated, or moving from the buffer into a data design on a read operation.

This mailbox is also used to contain operands returned as a result of the execution of certain types of I/O operations. These operations will be described in subsequent sections.

### 1.2.3 Task Control Block

When a task is swapped and another task is initiated (as in the case of a multiprogrammed environment), the identity of both tasks must be defined. This identity is necessary to restore a task once it is temporarily terminated. For this purpose a Task Control Block (TCB) must be defined. The TCB contains information which uniquely defines the state of the task upon interruption. Included among this information is: contents of the deferral mechanism, mode indicators, program counters, etc.

Given this information, a means of TCB swapping must be determined. In the Burroughs stack oriented machines, the equivalent of a TCB is pushed into the stack. This portion of the stack is in main memory. A pointer to the base of the TCB is kept in the executive system. In the AADC, the 4K limitation of the task memory may eliminate this manner of TCB handling. In a multiprocessor configuration, one normally wants TCB's stored in a common memory which can be accessed by all processors. (Thus,

any available processor can execute a task in a ready state). As presently configured, the AADC processing element has direct access only to its task memory.

Therefore, it appears that storage of a TCB in a common AADC memory is called for. The TCB mailbox in the system origin provides the address of the present TCB in the task memory. The one level of indirection to access the TCB is provided so that the TCB need not be stored in the same task memory locations. This should facilitate memory management. Complete specification of a TCB is dependent upon final system definitions.

#### 1.2.4 I/O Status Word

CMS-2 provides the facility for the programmer to test for certain I/O status conditions, such as: end-of-file, error checks, etc. To facilitate this capability, the I/O drivers of the executive system deposits the I/O status word upon termination of the specified I/O operation. This requires that the peripheral subsystem be capable of transmitting an I/O status word upon request from the executive system.

All CMS-2 logical statements that test I/O status, refer to this fixed location in the system origin. See Section 10.0 for the use of this feature.

#### 1.2.5 Interrupt Mask

The Interrupt Mask is provided for processing element/executive system communications. When the processing element wishes to interrupt the executive, a mask is deposited in this mailbox. The contents identify the reason for the interruption to the executive system. Via this approach only one executive call instruction is defined. The address field of this instruction is used as an immediate operand and is deposited in the interrupt mask mailbox.

### 1.2.6 Format Control Pointer

This memory location contains the address of a linked list which contains format control directives. These directives are subsequently used by I/O operations involving formatting, and the DECODE and ENCODE Instructions.

## 2.0 FILE DECLARATION

The FILE declaration defines the environment in which one or more physical files may be written on a particular device. All data that is to be inputted/outputted on a device is organized according to the information in its FILE declaration. All I/O commands under CMS-2 generate references to a FILE. The general form of the FILE declaration is:

```
FILE  internal-name type maximum number of records  
      length-descriptor max-record-size hardware-name states $
```

where -internal name is used to reference the FILE declaration

- type indicates Hollerith or Binary records
- max. # of records indicates the maximum number of records to be accessed
- length-descriptor indicates whether the records are variable (V) in length or rigid (R) in length
- Max-record-size indicates the maximum length of the records
- hardware name is a mnemonic used to reference a particular device
- states defines I/O states to be tested

The CMS-2 compiler makes an entry in the name-table. The entry contains the internal name of the file and the address of the delineation of the FILE declaration in the data partition. The delineation of the FILE declaration is:

DEVICE #	TYPE	LENGTH	ACCESS RIGHTS	UNUSED	MAXIMUM RECORD SIZE
1	8	9	10	11 12	13 20 21 32

MAXIMUM NUMBER OF RECORDS
1 32

At compile time, a translation from hardware device name to internal device number is made. This requires that the compiler have an associative table which relates logical device name with physical device number.

### 3.0 OPEN STATEMENT

The OPEN command establishes the I/O access rights for a previous defined file. The general form of the OPEN statement is:

```
OPEN NAME ACTION $
```

where - name is the name of a file  
 - action specifies the access rights. The access identifiers are:

- INPUT - only input operations allowed
- OUTPUT - only output operations allowed
- SCRATCH - both input & output allowed

The CMS-2 compiler generates code which sets bits 11-12 of Word 1 of the file declaration with a binary configuration which denotes one of the access rights. When an I/O operation is called for at execution time, the I/O command checks the declared access rights before invoking the executive system. A file can not be declared reopened until it is closed. Thus, if a file is INPUT and it is to be changed, it must first be closed.

#### 4.0 CLOSE STATEMENT

The CLOSE statement deactivates the specified file. An end-of-file mark is written for output files when appropriate. The general form of the CLOSE statement is:

```
CLOSE    NAME    $
```

where name is the identifier of a file. For the purpose of supporting the OPEN and CLOSE statements, two instructions are defined. TSFT (Test File Status Field) and SFST (Set File Status Field). The format of these instructions are:

OPCODE		STATUS		ADDRESS
1	6 7 10 11 12 13 20 21			32

The status field is used as an immediate operand. The TSFT instruction does a comparison for equal with bits 11-12 of the addressed word and of the instruction and returns a Boolean value to the accumulator. The SFST instruction sets bits 11-12 of the addressed word with bits 11-12 of the SFST instruction.

Upon recognition of the CLOSE statement, the CMS-2 Compiler generates an SFST instruction and tests the device address field of the file declaration to see if an end-of-file mark must be written on the device. This last operation is a function of the device type.

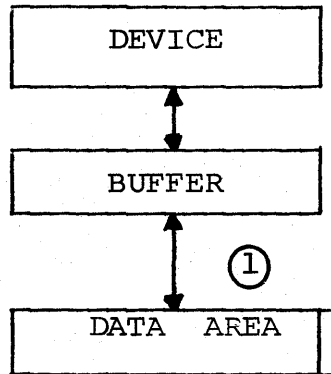
To support the CLOSE statement, the access right, CLOSED, is defined for bits 11-12 of the first word of the file delineation. Thus, a total of four access rights are defined.



## 5.0 INPUT AND OUTPUT COMMANDS

The INPUT and OUTPUT Commands transmit data between a hardware device and a user's program. The FILE declaration associated with the I/O operation reserves a buffer area.

I/O - operations proceed as follows:



The INPUT command results in the transfer of data from the hardware device to the buffer, and then to the user's data area. If formatting is used, the decoding operation is done at point ①. The OUTPUT command results in the transfer of data from the user's data area to the buffer, and then to the hardware device. If formatting is used, the encoding operation is done at point ①.

Prior to each input or output operation on a device, the buffer area associated with the file is preset either to blanks or zero depending on whether the records are declared Hollerith or binary, respectively.

The number of words transferred to the data area is the smaller of the input record length or receptacle size. Thus, if a 100 words of input is requested and the record is 50 words long, only 50 words of information are transferred to the data area with the remaining 50 words being blanks or zero. Conversely, if 20 Words

are requested and the input record is 50 words, the remaining 30 words are lost. Remember, CMS-2 is tape oriented, which implies the reading of one physical record at a time.

### 5.1 Input Statement

The general form of the INPUT statement is:

```
INPUT  NAME  RECEPTACLE  FORMAT-NAME  $
```

where - name is the name of a FILE declaration or the name of a standard device file (system dependent)

- receptacle is a CMS-2 data structure. More than one More than one receptacle can be declared by specifying receptacles separated by commas.

- format name (optional) refers to the name of a previously defined FORMAT declaration.

The CMS-2 Compiler generates code which does the following:

1. The first word of the file delineation is deposited in the system origin.

2. The format declaration address (if specified) is deposited in the system origin.

3. The address of the receptacle which will accept the input data, is deposited in the system origin. This address is a structor which contains the starting address and number of memory words. The memory structor has the following format:

TYPE		COUNT	ADDRESS		
1	4	9	20	21	32

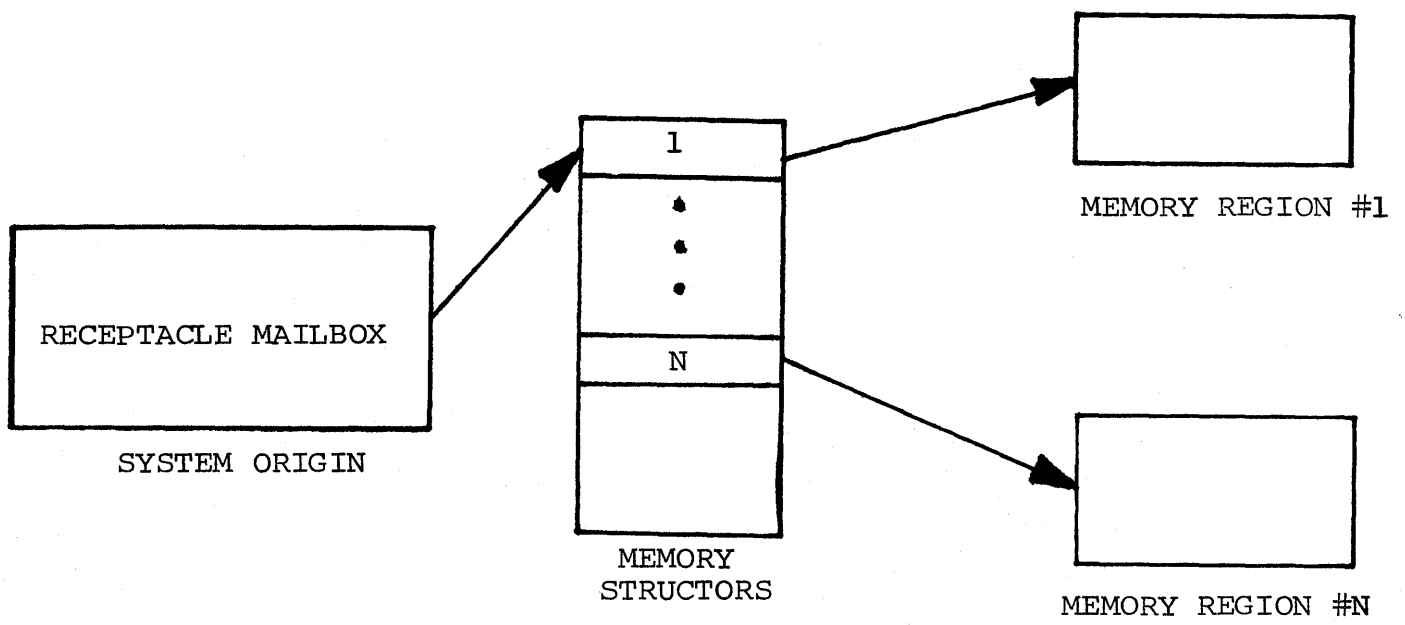
TYPE - Memory structor  
COUNT - Number of memory words  
ADDRESS- Starting address of first word

If more than one noncontiguous (memory locationwise) receptacles are specified in the INPUT statement, the compiler must build a list of memory structors which point to the receptacles to receive data. In this case, a list structor is deposited in the receptacle mailbox. The list structor has the following format:

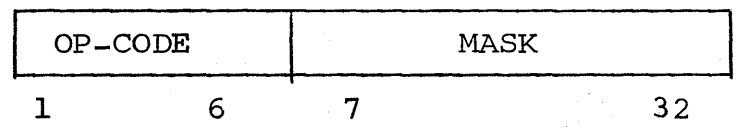
TYPE	EXTENT	ADDRESS
1 4	13 20	21 32

TYPE - List Structor  
EXTENT - Number of memory structors  
ADDRESS - Address of first memory structor

Defining this capability allows the CMS-2 compiler to build a list of memory structors which point to the receptacles involved in the input operation. The following depicts this linkage mechanism:



4. A Monitor Call instruction (MC) is executed. The Monitor Call instruction has the following format:



The execution of the Monitor Call deposits the mask field of the instruction in the interrupt mailbox of the system origin. The monitor is then invoked. The method of invoking the monitor is system dependent. In some systems, this might be a hardwired connection to a dedicated processor, or to a predetermined memory location. Once invoked, the monitor examines the interrupt mask to determine the reason for its awakening. In this case, an input operation is called for. The I/O drivers then assume system control.

## 5.2 OUTPUT Statement

The general form of the OUTPUT statement is:

```
OUTPUT   NAME   DATA-IMAGE   FORMAT NAME   $
```

The OUTPUT statement is handled in the same manner as the INPUT statement with the exceptions that:

- Data-image are receptacles that contain data to be outputted.
- The standard devices may be different (Printer,Punch)
- The interrupt mask of the monitor call instruction indicates an output operation.

## 6.0 FORMAT DECLARATION

The FORMAT declaration describes the conversion of data between internal and external forms. The external form is usually a Hollerith string, containing in addition to the data, certain spacing and control information. The CMS-2 format declaration is similar to the FORTRAN format statement.

The general form of the FORMAT declaration is:

```
FORMAT   NAME   Q1, Q2, ..., QN   $
```

where - name is the identifier to be used to reference the FORMAT declaration.

- Q<sub>i</sub> is a format descriptor. The format descriptors indicate the form and arrangement of data and the types of conversion to be performed.

The CMS-2 compiler delineates the FORMAT descriptors into a linked list in memory. The structure of the linked list is:

LINK CODE	DESCRIPTOR CODE	W	D	REP CODE	CODE LINKAGE	UNUSED
1	2 3	6 7 11	12 16	17 24	25 27	28 32

where - Link code is either: first descriptor, intermediate descriptor, or last descriptor.

- Descriptor Code is one of the ten allowable CMS-2 codes. The defined codes are: I,F,E,O,Z,L,A,X,H,/. (See pages I-4-48 to 49 of the CMS-2 manual for a description of the use of these codes.)

- W is an unsigned integer representing the maximum number of characters of the field in the external medium.

- D is an unsigned integer representing the number of characters in the field that appears to the right of the binary or decimal point.

- Repetition Code indicates that a group of descriptors are repeated M times. The binary value of M is contained in this field.

- Code Linkage indicates whether the format descriptor is the first, intermediate, or last in a repetition code group. If no repetition group is specified, this field is not use.

- Bits 17-32. If the descriptor code is H (Hollerith), the character string is stored beginning with Bit 17 and extending as many words as it is necessary to store the string.

The delineation of the FORMAT declaration is in a data design. It is never executed as an instruction sequence. If an I/O operation is formatted, the address of the FORMAT delineation is contained in the system origin.

## 7.0 ENCODE AND DECODE OPERATIONS

The ENCODE and DECODE statements direct an execution time routine to transfer data from one area of core to another while transforming the data from nonformatted to formatted status, or vice versa. In effect, an I/O operation is executed with the device being a buffer area in memory. Since, the routines which normally execute the formatting algorithms are part of the executive system, the ENCODE and DECODE instructions, to avoid duplication, should invoke the supervisor.

### 7.1 ENCODE Statement

The ENCODE statement specifies that the data elements contained in the image are to be converted according to a format identified by the FORMAT declaration and packed into a character string identified by a data name. The general form of the ENCODE statement is:

```
ENCODE      DATA UNIT IMAGE  FORMAT DECLARATION $
```

The CMS-2 compiler deposits the Hollerith structor of the data unit in the receptacle mailbox, the list structor of the image in the file address mailbox, and the address of the delineation of the format declarations in the format control mailbox.

In the AADC, with the task memory configuration, the system processor which normally contains the executive system will normally be the execute processor for the I/O drivers. Executing the ENCODE

and DECODE statement in this manner eliminates the need for duplicating that portion of the I/O drivers which deal with formatting data. Additionally, task memory space is saved as a result of this elimination in tradeoff for the time necessary to invoke the monitor. However, the frequency of usage for these operations and their importance makes this tradeoff acceptable.

## 7.2 DECODE Statement

The DECODE statement specifies that the character string identified by the data name is to be converted according to the form specified by the format declaration and stored in the locations specified by the image. The DECODE operation is the reverse of the ENCODE operation. The general form of the DECODE statement is:

```
DECODE DATA UNIT IMAGE FORMAT DECLARATION NAME $
```

The CMS-2 compiler generates the same code as the ENCODE statement with the exception that the interrupt mask of the MC instruction indicates the DECODE operation.

## 8.0 ENDFILE STATEMENT

The ENDFILE statement is used to form a physical file by writing an end-of-file mark at the present location of the file of the device addressed. A logical file may be partitioned into many physical files. The general form of the ENDFILE statement is:

```
ENDFILE NAME $
```

where name is the name of a previously opened file.



The CMS-2 Compiler inserts the file address in the system origin and issues an MC instruction, with an interrupt mask indicating end-of-file. The record position for the named file is set to 0. More will be said concerning this in the next section.

## 9.0 DEVICE POSITIONING

CMS-2 allows the programmer to position devices by logical files or physical records within a logical file. To support this capability, the executive system must maintain within its domain, a device table. This device table contains the following information:

- Device Name (address)
- Present Record Position relative to the last end-of-file mark.
- Length of last record read (bytes)

The use of these tables will become apparent in the following sections.

### 9.1 Positioning By Files

A device may be positioned forward or backward a specified number of files. The device is always automatically positioned following the end-of-file mark. The general form of file positioning is:

```
SET    POS (NAME)  TO±    INTEGER CONSTANT    $  
                                OR  
                                DATA - NAME
```

where - POS(name) specifies the positioning of the device specified by the name of a file declaration.

- $\pm$  specifies forward position (+) or backward positioning (-).
- Integer constant or data name which specifies the number of files which the unit is to be positioned. If the value is 0, the device is positioned to the beginning (TAPE physical beginning of tape, DISK-track 0, cylinder 0). If the value is -0, the device is closed. (Same effect as a CLOSE statement.)

The compiler generates the following code:

If the positioning value is not -0, the positioning value is deposited in the receptacle mailbox in the system origin. (This will be interpreted as operand by the executive system.)

The file address is deposited in the file address mailbox.

The MC Instruction is executed with the interrupt mask indicating file positioning.

The executive system, once invoked, must associate the device with the named file, position the device as specified, and update the device/record table with a 0 entry for the device.

If the positioning value is -0, the file status field in the file declaration is changed to CLOSED by code contained in task memory, not by the executive system.

## 9.2 Positioning by Records

A device may be positioned forward or backward a specified number of records within the current file. Attempted record positioning beyond the bounds of the current file will cause an I/O abort. The general form of record positioning is:

```
SET    POS(name)  TO  POS(name)  ±    INTEGER-CONSTANT  
                                         OR  
                                         $  
                                         DATA NAME
```

where - POS(name) specifies the device named by a file declaration.

- ± specifies forward positioning (+) or backward positioning (-).
- Integer constant or data name specifies the number of of records to be spaced forward or backward.

The CMS-2 Compiler deposits the file address and the spacing operand (receptacle mailbox) in the system origin.

The MC instruction is executed with the interrupt mask indicating record positioning.

### 9.3 Record Position Determination

The record position within the current file is determined with the use of the POS modifier. The general form of the record position determination is:

```
SET      DATA NAME TO POS(NAME) $
```

The CMS-2 compiler generates

```
SET ( < FILE MAILBOX >  
(STORE IMMEDIATE) TOI ) < FILE ADDRESS >  
MC      RECORD DETERMINATION MASK  
SET     < ADDRESS OF DATA NAME >  
TO      < RECEPTACLE MAILBOX >
```

The executive system returns the value of the present record to the receptacle mailbox.

#### 9.4 Record Length Determination

The length of the last record transmitted by either an INPUT or an OUTPUT statement is determined by using the modifier LENGTH. The general form of record length determination is:

```
SET          DATA NAME    TO    LENGTH (NAME)  $
```

where - LENGTH(name) specifies the length of the previous record transmitted by an INPUT or OUTPUT operation on the named file.

The CMS-2 Compiler generates:

```
SET    < FILE MAILBOX >  
TOI    < FILE ADDRESS >  
MC     RECORD LENGTH MASK  
SET    < ADDRESS OF DATA NAME >  
TO     < RECEPTACLE MAILBOX >
```

The executive system returns the value of the record length to the receptacle mailbox.

#### 10.0 DEVICE STATE CHECKING

The execution of any I/O operation results in the return of an I/O status word in the system origin. Each bit in this word signifies whether or not a specific condition resulted (e.g., end-of-file, transmission error, etc.). The programmer may test this status word by using the following form:

```
IF      NAME      EQ/NOT      STATE      THEN
```

The testing IF statement must precede any subsequent I/O operations to test the validity of the last executed I/O operation. The compiler generates

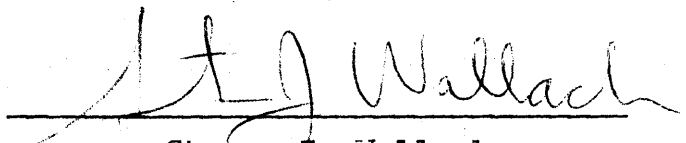
TBIF (Test Bit Immediate False)  
or TBIT (Test Bit Immediate True)

depending whether or not NOT or EQ was specified, respectively.

The I/O status word is in a fixed location, thereby eliminating an explicit address in these instructions. The CMS-2 compiler must know the bit position of each I/O state to correctly structure the TBIF or TBIT instructions. The format of these instructions is therefore:

OP-CODE	UNUSED	BIT ADDRESS	BRANCH ADDRESS
1	6 7 15	16 20	21 32

If the test is true, the next sequential instruction is executed, otherwise the branch address is taken.

  
Steven J. Wallach  
Dept. 7675 Ext. 3473

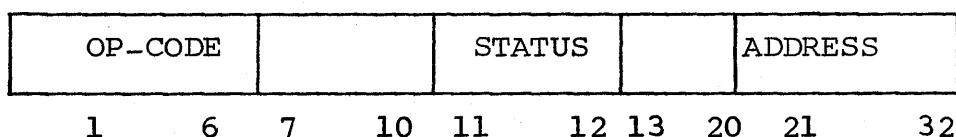
SJW/gb

dc: S. Nissen 904  
J. Baker 917  
B. Scheff 917

APPENDIX A

The following are instructions defined specifically to handle operations involving CMS-2 I/O.

- TSFT (Test File Status Field)

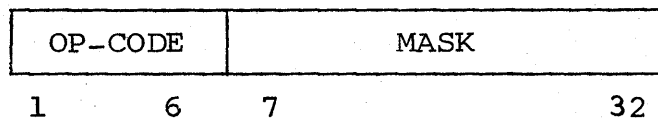


The status field of the instruction are compared for equal with bits 11-12 of the addressed word. A Boolean value is returned to the accumulator.

- SFST (Set File Status Field)

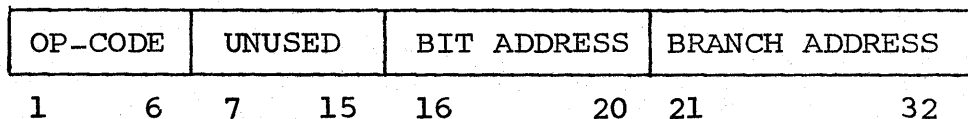
Same format as TSFT. Bits 11-12 of the instruction are set into bits 11-12 of the addressed word.

- MC (Monitor Call)



The mask is deposited in the interrupt mailbox of the system origin. The monitor is then invoked. Method of monitor awakening is system dependent.

- TBIF (Test Bit Immediate False)



The addressed bit position in the I/O status word in the system origin is tested for Boolean 0. If the test is successful, the next sequential instruction is executed, otherwise, a transfer to the branch address is taken.

- TBIT (Test Bit Immediate True)

Same as TBIF except the test is for Boolean 1.