FINAL REPORT

FOR THE

SUMC RECONFIGURABLE MICRO-ASSEMBLER

17 DECEMBER 1973

PREPARED FOR:

NATIONAL AERONAUTICS & SPACE ADMINISTRATION
MARSHALL SPACE FLIGHT CENTER
HUNTSVILLE, ALABAMA 35812

PREPARED BY:

MCDONNELL DOUGLAS ASTRONAUTICS COMPANY
5301 BOLSA AVENUE
HUNTINGTON BEACH, CALIFORNIA 92647

A. J. EDWARDS
PRINCIPAL INVESTIGATOR

D. W. GIEDT
PROJECT MANAGER

PREFACE

This document constitutes the final report for the Micro-Assembler segment (S4/4) of the Techniques for the Generation of Support Software project being performed for NASA MSFC under Contract NAS8-27202.

If additional information is required, please contact any of the following McDonnell Douglas or NASA representatives:

- Mr. D. W. Giedt, Project Manager
  Huntington Beach, California
  Telephone: (714) 896-4908

- Mr. A. J. Edwards, Principal Investigator
  Huntington Beach, California
  Telephone: (714) 896-3872

- Mr. G. M. Jones, Contract Administrator
  Huntington Beach, California
  Telephone: (714) 896-2795

- Mr. B. C. Hodges, Project COR, S&E-COMP-C
  Marshall Space Flight Center, Alabama
  Telephone: (205) 453-1385

CONTENTS

I. SCOPE

The objective of this effort was to develop a reconfigurable micro-assembler to provide the micro-programmer the capability to specify micro-instructions in concise, meaningful terms. The micro-assembler adds an important capability to the SUMC software development facility since firmware micro-programming is an essential part of the SUMC software development process.

II. TECHNICAL APPROACH

The implementation plan for the development of the micro-assembler was predicated on the existing capabilities of the SUMC Reconfigurable Assembler. Utilizing the reconfigurable assembler as a base, new directives and existing directive modifications were implemented to provide the micro-assembly as a new capability of the reconfigurable assembler.

III. MICRO-ASSEMBLER LANGUAGE CONCEPT

The micro-assembler language allows the specification of all micro-instruction control field settings in one concise assembler source statement. Via instruction definition directives, "micromacs" are defined to the micro-assembler which designate actual field settings for a selected combination of one or more control fields. An assembler source statement then is composed of one or more micromacs which collectively describe the operational function of a micro-instruction. This capability to define micromacs provides open-ended micro-assembler instruction sets and overall design flexibility.

This approach benefits the micro-programming task by upgrading the micro-assembler language from control field terms, inherent in the micro-instructions, to micro-instruction subfunction terms. It will additionally result in more readable, self-documenting programs.

IV. MICRO-ASSEMBLER LANGUAGE SYNTAX

The micro-instruction assembler language appears very similar to a conventional machine instruction assembler language. The machine instruction assembler language has the characteristic of one operation specification per statement whereas, the micro-instruction assembler language allows multiple operations to be designated per statement.

The source statement format consists of two primary fields:

1. Label field
2. Command field

The optional label field consists of a symbol that is equated to the location counter or a value. The command field is mandatory and is composed of an operation/operand (micromac) list. Each operation may have multiple operands, in which case they appear separated by commas.

Label Field	Command Field
[symbol]	operation [(operand)] [,operation[(operand)]....]

where: [ ] means "optional"

... means "and more of the same"

V. MICRO-ASSEMBLER UTILIZATION

The design objective for the creation of micromacs is to identify a complete set of data gating functions that is a "natural" subset of the operational capability of one micro-instruction. A micro-instruction then is constructed from one or more micromacs that collectively describe the functions to be performed.

A sample micromac instruction set has been designed and semantically described for discussion purposes in Appendix A. This candidate micromac mnemonic list has been functionally categorized as follows:

1. Main memory operations
2. Scratchpad memory operations
3. Adder operations
4. Register operations
5. Control operations

It should be noted that these categories are arbitrarily assigned for illustrative purposes only and are based on the example micromac instruction set.

Since micromacs must be defined to the assembler, the mnemonic assignment and format design is also arbitrary. This provides flexibility to the language definer at language definition time as well as to the programmer at assembly time.

Each micromac is defined in terms of micro-instruction fields and their respective value settings. Appendix B describes the micro-instruction field values for the example micromac instruction set. Appendix D illustrates the assembler directives to implement these micromacs for micro-instruction assembly. Appendix E is an example of a micro-instruction assembly utilizing the example micromac instruction set.

VI. NEW ASSEMBLER DIRECTIVES AND FUNCTIONS

1. Option for Micro-Assembly

The OPTION statement has an additional operand to designate the type of assembly, machine instruction level or micro-instruction level.

operand format:

TYPE = t

where t = NORMAL or MICRO

2. Rescan Source Operand

The IFORM directive has an additional operand for resetting the scan cursor to multiply process a source statement operand. This provides the ability to set multiple fields based on an operand since one field can be set per scan.

operand format:

RC

3. Micro-Instruction Statement Processing

The internal statement processing logic is altered for micro-instructions since they are composed of one or more micromacs. This necessitates multiple IFORM's to be processed per statement.

4. Default Values for Micro-Instruction Fields

A new assembler directive has been implemented for specifying the default values for micro-instruction fields. Default values are then used for those fields that are not set by micromacs for a micro-instruction.

directive format:

Label	Operation	Operand
	DFIELDS	value=(start-end)[,value=(start-end)...]

where: value - field default value

(start-end) - bit positions of a field

5. Description of IROM data for the Micro-Assembly Module

A new assembler directive has been implemented for specifying the IROM data word structures that correspond to the micro-assembly module. The object output occurs in the vector symbol dictionary section and is utilized by the linkage editor to produce a listing of the IROM memory map for the load module.

directive format:

Label	Operation	Operand
symbol	VECTOR	len,O=op,L=(start-end)[,value=(start-end)...]

where: len - No. bits in IROM data word

symbol - micro-assembly module (vector) entry point label

op - value for O (op code)

(start-end) - bit positions of the location (L) or a value field in the IROM data word

value - field setting value

6. Micro-Assembly Diagnostics

A new assembler directive has been implemented for designating micro-instruction fields to be diagnostically controlled by the assembler for multiple value settings for one micro-instruction statement.

directive format:

Label	Operation	Operand
	MFIELDS	(start-end)[,(start-end)...]

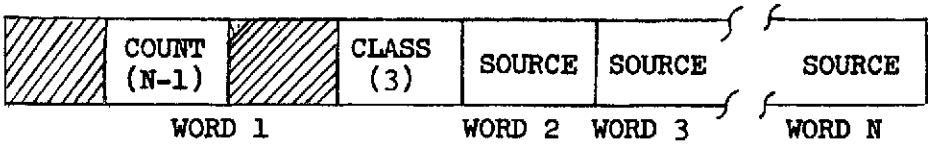
where: (start-end) - bit positions of one of the multiple fields

7. Object Output

There are two new records incorporated in the object output module.

Source Image Records

The source image records are generated as a subtype of the TXT object section. Each source image record is formatted as follows:



Count (word 1, bits 16-6)

The number of words remaining in the current source image record: the number of words necessary to hold 96 characters.

Class (word 1, bits 4-1)

The text source image record sub-type: 3

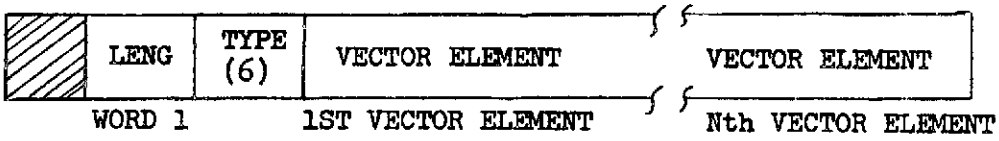
Source (words 2-N)

The source image representing 96 characters in packed integer code format.

Vector Symbol Dictionary (VSD)

A new object type has been created to contain the vector (IROM data word) description from the VECTOR assembler directive.

VSD records are formatted as follows:



Leng (word 1, bits 31-17)

The number of data words in the binary object record.

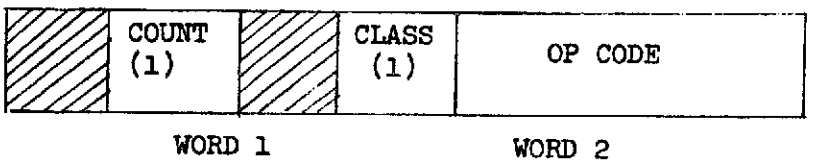
Type (word 1, bits 16-1)

The new VSD object type: 6

Vector Element

Each vector element is composed of three sub-types and contains the data from one VECTOR assembler directive.

A. Control



Count (word 1, bits 16-6)

The number of words remaining in the current control sub-type: 1

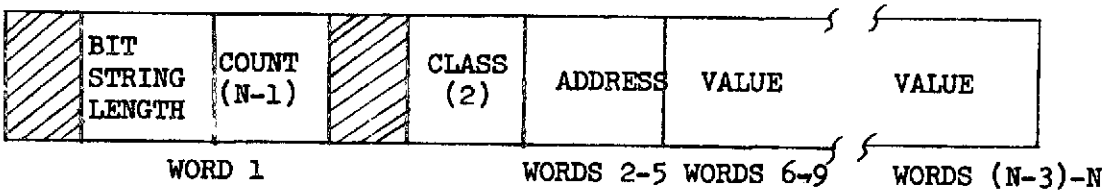
Class (word 1, bits 4-1)

The vector element control sub-type: 1

Op code (word 2, right justified)

The specified op code value.

B. Data Fields



Bit string length (word 1, bits 31-17)

The length in bits of the vector word to be constructed from the information contained in the attached address and value fields.

Count (word 1, bits 16-6)

The number of words remaining in the current data fields sub-type.

Class (word 1, bits 4-1)

The vector element data fields sub-type: 2

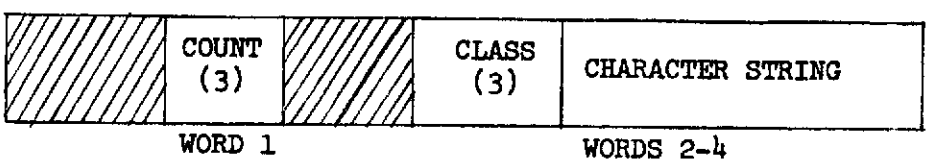
Address (words 2-5)

The address assigned by the assembler for the designated vector entry point: represented by the normal "get/put" element object structure as described in Section 5 of the Reconfigurable Assembler Detail Specifications.

Value (subsequent 4 word blocks)

The values to be assigned to additional fields within a vector entry word: also represented by the normal "get/put" element object structure.

C. Identification



Count (word 1, bits 16-6)

The number of words remaining in the current identification subtype: 3

Class (word 1, bits 4-1)

The vector element identification sub-type: 3

Character string (words 2-4)

The character string identifying the entry point: the symbol from the VECTOR directive, in 3 words in packed integer code format.

VII. USER MANUAL DOCUMENTATION

The user manual documentation for the micro-assembler has been created as change pages to the SUMC Reconfigurable Assembler Users Manual. These are contained in Appendix F.

APPENDIX A

SEMANTIC DESCRIPTION OF EXAMPLE MICROMAC INSTRUCTION SET

Main Memory Operations

RMM — Read Main Memory

The contents of main memory specified by the adder are gated to the memory register.

WMM — Write Main Memory

The contents of the adder are gated to main memory specified by the memory address register.

Scratchpad Memory Operation

WSM(x) — Write Scratchpad Memory

The contents of the adder are gated to scratchpad memory specified by x.

Adder Operations

LDA(x) — Load Adder

Contents of x are gated to the adder.

LAI(x) — Load Adder and Increment

Contents of x are gated to the adder and are incremented by one.

SAD(x,y) — Save Add

Contents of x and y are gated to and added in the adder and the result is gated to the PRR.

CAD(x) — Continuation Add

Contents of x and the value in the PRR are gated to and added to the adder.

Register Operations

ZMR — Zero Multiplexor Registers

Zeros are gated to the MQR, MAR and PRR.

LIR — Load Instruction Register

Contents of the memory register are gated to the instruction register.

Control Operations

JMP — Jump to Micro Program

Control is transferred to the micro program associated with the machine instruction operation code.

(NOTE: This operation normally is used to conclude an instruction fetch.)

PNO — Perform Next Operation

Control is passed to the next sequential micro instruction.

(NOTE: The assembler will select this operation automatically when a control operation is not specified by the programmer; typical programmer use of PNO is to idle one memory cycle following an RMM.)

JMI(x) — Jump to Micro Instruction

Control is transferred to the micro instruction specified by x.

JIN(x) — Jump on Interrupt

If the interrupt flag IOG is on, transfer to the micro instruction specified by x; otherwise the next sequential micro instruction is performed.

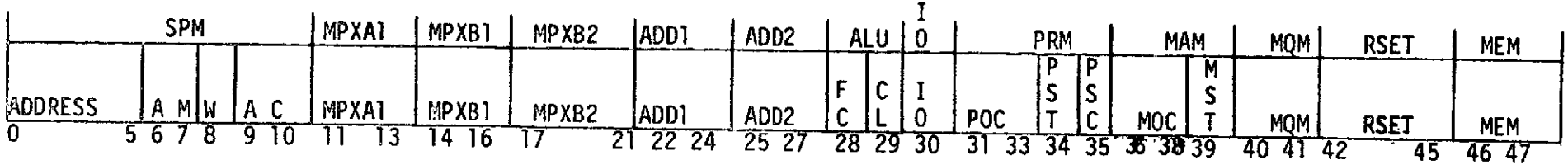
APPENDIX B

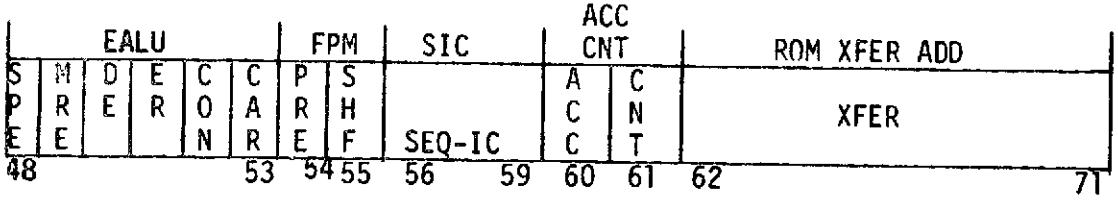
MICRO-INSTRUCTION FIELD VALUE DESCRIPTION

MICROMAC	MICRO-INSTRUCTION FIELDS/VALUES (REFER TO APPENDIX C FOR FIELD DESCRIPTIONS)
RMM	MOC=B'111'; RSET=B'0010'; MEM=B'10'
WSM[(X)]	[ADDRESS=(X); AM=(X);] W=1; AC=B'00'; POC=B'001'; MQM=B'00'; RSET=B'0001'
LAI(X)	ADDRESS=(X); AM=(X); AC=(X); MPXA1=B'100'; MPXB1=B'110'; ADD1=B'011'; ADD2=B'011'; FC=1; EALU=B'111100'
JMI(X)	SEQ-IC=B'1101'; XFER=(X)
PNO	AC=B'11'; MPXA1=B'100'; EALU=B'111100'; SEQ-IC=B'0000'
LIR	MPXA1=B'100'; RSET=B'1000'; EALU=B'111100'
JIN(X)	SEQ-IC=B'1000'; XFER=(X)
SAD(X,Y)	ADDRESS=(X); AM=(X); AC=(X); MPXB1=B'110'; MPXB2=(Y); ADD1=B'011'; ADD2=B'011'; POC=B'001'; RSET=B'0100'
CAD(X)	ADDRESS=(X); AM=(X); AC=(X); MPXA1=B'000'; MPXB1=B'110'; ADD1=B'011'; ADD2=B'011'; RSET=B'0001'; EALU=B'111100'
ZMR	AC=B'11'; MPXA1=B'100'; POC=B'000'; MOC=B'000'; MQM=B'00'; RSET=B'0100'; EALU=B'111100'
JMP	SEQ-IC=B'0101'
LDA(X)	ADDRESS=(X); AM=B'01'; AC=(X); MPXA1=B'100'; MPXB1=B'110'; MPXB2=B'00000'; ADD1=B'011'; ADD2=B'011'; EALU=B'111100'
WMM	POC=B'001'; RSET=B'0100'; MEM=B'01'

APPENDIX C

MICRO-INSTRUCTION FIELD DESCRIPTION





ADDRESS = SPM address
AM = SPM address modifier
W = SPM write control
AC = SPM word access control
MPXA1 = Multiplexer A1 control
MPXB1 = Multiplexer B1 control
MPXB2 = Multiplexer B2 control
ADD1 = Arithmetic unit #1 operation
ADD2 = Arithmetic unit #2 operation
FC = Force carry

CL = Carry latch
IO = Input/output control
POC = PRM operation control
PST = PRM shift type
PSC = PRM shift control
MOC = MAM operation control
MST = MAM shift type
MQM = MQM operation control
RSET = Register set
MEM = Memory control

SPE = SPM exponent
MRE = Memory register exponent
DE = Derived exponent
ER = Exponent register
CON = EALU control
CAR = Forced carry
PRE = Precision
SHF = FPM shift allow
SEQ-IC = Sequencer iteration control
ACC = ACCS control signal
CNT = CNT control signal
XFER = ROM transfer address

```
*       EXAMPLE  SUMC  MICROINSTRUCTION  DEFINITION.

        MODE     MICRO
        SIZE     1024,72
        MFIELDS  (0-5),(6-7),(8),(9-10),(11-13),(14-16),(17-21),      *
                 (22-24),(25-27),(28),(29),(30),(31-33),(34),(35),    *
                 (36-38),(39),(40-41),(46-47),(48-53),(54),(55),      *
                 (56-59),(60),(61),(62-71)

SPMAC   CCODES   'PC'  = 0,  'A'  = 0,  'B'  = 3,  'X'  = 3
MPXB2   CCODES   'D'  = 1, 'MR'  = X'11',
SPMAD   CCODES   'PC'  = X'10',  'A' =0,  'B' = X'10',  'X'  = X'14'
SPMAM   CCODES   'PC' = 0,  'A'  = 1,  'B'  = 2,  'X'  = 3


IRMM    IFORM    72,0=(36-39),X'2'=(42-45),X'2'=(46-47)
IPNO    IFORM    72, 0 = (48-53),X'3'=(9-10), X'4' = (11 - 13),0=(56-59)
ILIR    IFORM    72, 0 = (48-53), 8 = (42-45),  4 = (11-13)
IZMR    IFORM    72,0=(48 - 53), 4=(11-13),4=(42-45),3=(9-10),       *
                 0 = (31-33), 0 = (36-39), 0 = (40-41)
IJMI    IFORM    72, 0 = (56-59),M=(L(62-71))
IWMM    IFORM    72,0= (31-33),4=(42-45),1=(46-47)
IWSM    IFORM    72, 0 = (31-33),SPMAD=(0-5),RC,SPMAM(6-7),0=(9-10),  *
                    1=(8),0=(40-41),1=(42-45)
ISAD    IFORM    72, 0 = (42-45),SPMAD =(0-5),RC,SPMAM =(6-7),RC,     *
                 SPMAC =(9-10),MPXB2=(17-21),6=(14-16),3=(22-24),     *
                 3=(25-27),1=(31-33)
ICAD    IFORM    72, 0 = (42-45),SPMAD=(0-5),RC,SPMAM=(6-7),RC,       *
                 SPMAC=(9-10),0=(11-13),6=(14-16),3=(22-24),3=(25-27),*
                 1=(42-45),X'3C'=(48-53)
ILDA    IFORM    72, 0 = (6-7),SPMAD=(0-5),RC,SPMAC=(9-10),4=(11-13), *
                 6=(14-16),0=(17-21),3=(22-24),3=(25-27),X'3C'=(48-53)
IJMP    IFORM    72, 0 = (56-59)
ILAI    IFORM    72, 0=(28),SPMAD=(0-5),RC,SPMAM=(6-7),RC,X'3C'=(48-53),*
                  SPMAC=(9-10),4=(11-13),6=(14-16),3=(22-24),3=(25-27)
IJIN    IFORM    72, 0 = (56-59), M = (L(62-71))


RMM     OPDEF      7, IRMM
PNO     OPDEF      X'3C',  IPNO
LIR     OPDEF      X'3C',  ILIR
ZMR     OPDEF      X'3C',  IZMR
JMP     OPDEF      X'5', IJMP
WMM     OPDEF      X'1', IWMM
WSM     OPDEF      X'1', IWSM
LAI     OPDEF      1, ILAI
JIN     OPDEF      8,  IJIN
SAD     OPDEF      4,  ISAD
CAD     OPDEF      1,  ICAD
LDA     OPDEF      1,  ILDA
JMI     OPDEF      13, IJMI
```

```
*         EXAMPLE  SUMC  MICROINSTRUCTION  CODE

          START

*                 FETCH NEXT MACHINE INSTRUCTION

FETCH LAI(PC),WSM,RMM,JIN(IOG)
      PNO
      LIK,SAD(X,D)
      CAD(B),RMM
      ZMK,JMP

*                 STORE ACCUMULATOR

STA   LDA(A),WMM,JMI(FETCH)

*                 STORE BASE REGISTER

STB   LDA(B),WMM,JMI(FETCH)

*                 STORE INDEX REGISTER

STX   LDA(X),WMM,JMI(FETCH)

*                 INTERRUPT HANDLING ROUTINE

        ORG   X'390'
IOG     EQU   *

        END
```

APPENDIX F

USER MANUAL CHANGE PAGES

The following pages are replacements for the corresponding pages in the SUMC Reconfigurable Assembler Users Manual.

TABLE OF CONTENTS
(CONTINUED)

OPTION

Function

Designate assembly requirements for source library input, automatic assembly definition, listing output and/or object output.

Format

Name	Operation	Operand
blank	OPTION	list

Description

list - choices may be specified from one or more of the following options:

library - controls the availability of a source statement library input data set.

NOLIB : inhibit source library input

LIB=x : accept a user library named x

LIB = SYSLIB : accept the standard system library

segment - controls automatic assembly definition from a selected library input.

NOSEG : inhibit the processing of an assembly definition from the source library.

SEG=y : copy and process a user supplied library segment - this segment of source input should contain definitions of a machine configuration, an instruction set, and appropriate system macros.

SEG = SYSSEG : copy and process the assembly definition contained in the standard system library segments.

listing - controls the output of an assembly listing.

LIST[([n][,|XREF|NOXREF|])]: prepare an assembly listing with n lines per page, consistent with the subsequent use of the PRINT statement; also print (XREF) or inhibit the printing (NOXREF) of the program symbol cross-reference list.

NOLIST : inhibit the output of an assembly listing.

ERROR=s : print all level s diagnostics and higher

COMP=p : print the target computer name: p on columns 1-8 of the title line on each page of the assembly listing.

DATE=d : print the date: d on the assembly listing if the date is not provided by the host operating system.

object - controls the output of an object program deck.

DECK : prepare an object output deck that is named and identified according to the subsequent use of the START statement.

NODECK : inhibit the output of an object program deck.

MODE=m : assemble this object program as relocatable (m=REL) or absolute (m=ABS).

type - controls the type of assembly

TYPE=t : this assembly is normal machine level (t=NORMAL) or micro level (t=MICRO).

Example

```
OPTION  LIB=SYSTEM7,SYSSEG,NODECK,LIST
```

The options stipulated in this example require the assembler to utilize the SYSTEM7 source library input data set and to automatically retrieve and assemble the machine configuration, instruction set and system macros that are stored under the standard assembly definition name: SYSSEG. An object deck is not wanted for this assembly, while the assembly listing is desired, and the type of assembly is normal machine level.

Usage

One OPTION statement may be specified for each program that is assembled; its use is optional. When omitted, the following options are assumed: TYPE=NORMAL, LIB=SYSLIB, SEG=SYSSEG, LIST(52,NOXREF), ERROR=1, COMP=SUMC, NODECK, MODE=REL. Options may be specified in any order and the OPTION statement must be the first statement in the program. The library option must be the same for separate assemblies that are part of the same job.

The naming of a library segment in the option statement prevents the programmer from having to define the assembly process himself. More specifically, the programmer may not redefine a target computer, its mnemonic operation codes, or any of its system macros if he has caused their definition via the OPTION statement. But, in this case, he may extend the instruction set by defining supplementary mnemonic operation codes and instruction formats, he may define additional macro instructions, and he may specify the availability of memory and registers to the program under assembly.

Function

Specify the numeric operation code(s) and the object instruction format(s) that correspond to a source instruction mnemonic operation code.

Format

Name	Operation	Operands
mnemonic	OPDEF	opcode,formats[,opcode,formats..]

Description

mnemonic - the symbolic name of a source instruction operation code.

opcode - the numeric value of the source instruction operation code - this value gives meaning to operand form 0 in the IFORM assembler-instruction for a machine level assembly.
Note - this operand field is omitted during a micro-assembly.

formats - identification of one or more instruction formats as:

fname (fname[,fname...])

fname - the label name of an appropriate IFORM assembler-instruction with which source instruction operands are formatted into appropriate bits of an equivalent machine instruction.

Example

```
LA OPDEF X'3F',TYPE1
```

The mnemonic operation code LA is defined as a hex 3F for a machine level assembly and its source operands are to be processed according to a TYPE1 IFORM pseudo-operation.

```
LA OPDEF X'3F0',SHORT,X'3F1',(LONG1,LONG2)
```

The mnemonic operation code LA is defined as a hex 3F0 for a machine level assembly if all of its source operands are specified in the SHORT IFORM assembler-instruction. Otherwise, LA is defined as a hex 3F1 for either LONG1 or LONG2 IFORM assembler-instructions depending on the presence of optional symbolic operand forms that cannot be contained in a short instruction format.

```
LA OPDEF IF1
```

The mnemonic operation code LA is defined and its source operands are to be processed according to an IF1 IFORM for a micro-assembly.

Usage

For each source instruction the designated machine instruction formats will be processed from left to right until either a successful assembly has been performed or until all of the available formats have failed. Therefore, multiple IFORM assembler-instruction names should be ordered consistent with the syntactic priorities of the allowable symbolic operands.

This assembler instruction must follow the referenced IFORM assembler instructions.

IFORM

Function

Specify an object instruction format for one or more source instruction mnemonic operation codes.

Format

Name	Operation	Operands
format	IFORM	length,O=(start-end) [,type[=(start-end / subtype=(start-end)[,]...])]...

Description

format - the symbolic name of an instruction form - this name may be referenced by the OPDEF assembler instruction.

length - the number of bits in this instruction format - this value must be divisible by or into the size of the location counter addressing unit (see the SIZE statement).

O= - an operation code reference - operation codes are symbols that appear in a source statement mnemonic operation field and have been equated to values via an OPDEF assembler instruction. The IFORM assembler instruction incorporates these assigned operation code values into machine instructions by means of this symbolic operation code reference for a machine level assembly.
Note - this operand field is omitted for a micro-assembly.

(start-end) - location of the receiving field in the assembled machine instruction. The bit positions in the generated machine instruction bit string into which the value of the referenced source operand will be "OR"ed.

The allocation of a one-bit field may be designated by omitting the optional "end" specification: e.g., (start).

type - a symbolic operand form name expressed as:

scan
register
control-value
control-code
control-function
class-code
memory-reference

Most symbolic operands that are appropriate for a given machine instruction format are required. When applicable, particular symbolic operands may be designated as being "optional" by enclosing their type and field description(s) in parenthesis: e.g.,

(type = (start-end)).

SECTION 13. MICRO-ASSEMBLY STATEMENTS

The micro-instruction format appears very similar to the machine-instruction format as described in Section 9. Whereas the machine-instruction format has the characteristic of one operation specification per statement, the micro-instruction format allows multiple operation specifications per statement, separated by commas. The operand field for each operation specification immediately follows the operation mnemonic and must be enclosed in parentheses. The operand field may contain multiple operands, separated by commas.

The same directives utilized for target computer instruction set definition applies for the micro-instruction definition as described in Section 9.

Micro-Instruction Default Field Values

Micro-Instruction fields may be designated for initialization with a value with the DFIELDS directive. The specified field value will occur in the designated field if the IFORMs processed for the micro-instruction do not reference the same field.

Micro-Assembly Diagnostic Control

Micro-Instruction fields may be designated for diagnostic control by the assembler with the MFIELDS directive. Specified fields will be monitored for multiple field references with conflicting field values during the IFORMs processed for a micro-instruction.

Micro-Assembly Vector Definition

The micro-program entry point vectors are defined via the VECTOR directive. This provides the capability to describe the control core contents for linkage to the micro-assembly module.

DFIELDS

Function

Designate micro-instruction fields that are to be initialized with the specified value before each micro-instruction statement assembly.

Format

Name	Operation	Operand
	DFIELDS	value=(start-end)[,value=(start-end)...]

Description

value= - the initialization for the field.

(start-end) - the beginning and ending bit positions of the field to receive the value.

Example

```
DFIELDS     7=(1-5),X'A'=(21-24)
```

This statement specifies the micro-instruction fields defined as bit positions 1-5 and 21-24 to be initialized with the values 7 and X'A' before each micro-instruction statement assembly. Any micro-instruction statement may override the default value in the IFORM processing.

Usage

This assembler statement must follow the START statement in the source module. There is no limit to the number of DFIELDS statements within a source module.

MFIELDS

Function

Designate micro-instruction fields for diagnostic control by the assembler.

Format

Name	Operation	Operand
	MFIELDS	(start-end)[,(start-end)...]

Description

(start-end) - the beginning and ending bit positions of the field to be monitored.

Example

```
MFIELDS   (12-15), (19), (31-42)
```

This statement specifies the micro-instruction fields defined as bit positions 12-15, 19, and 31-42 to be monitored by the assembler for multiple field references with conflicting values during a micro-instruction statement assembly.

Usage

This assembler statement may occur anywhere in the source module. There is no limit to the number of MFIELDS statements within a source module.

VECTOR

Function

Define the control core entry contents for linkage to the micro-assembly module. The object that is generated for this directive serves to construct a listing of the control core contents.

Format

Name	Operation	Operand
symbol	VECTOR	len,O=op,L=(start-end)[,value-(start-end)...]

Description

symbol - micro-assembly module statement label which may be internally or externally defined.

len - No. bits in IROM data word

O= - designates the op code operand

op - op code value

L= - designates the location operand

(start-end) - the bit positions of the entry field

value - field setting value

Example

```
ADD VECTOR 11,Ø=X'1A',L=(3-10),1=(2)
```

This statement defines a control core entry structure for the micro-assembly module label ADD. The entry is 11 bits in length, the micro-assembly location for statement label ADD goes into bits 3-10 and bit 2 is always set to 1. The op code for linkage to the statement label ADD is X'1A'.

Usage

This assembler statement must follow the START statement in the source module. There is a limit of 256 VECTOR statements per assembly.