# EMMYXL USER'S GUIDE

by
Walter A. Wallach

March 1976

Technical Note No. 84

Digital Systems Laboratory
Stanford Electronics Laboratories
Stanford University
Stanford, California

The work herein was supported in part by the Army Research Office-Durham under Grant No. DAAG29-76-G-0001.

Digital Systems Laboratory

Stanford Electronics Laboratories

Technical Note No. 84

March 1976

# EMMYXL USER'S GUIDE

by

Walter A. Wallach

## ABSTRACT

This document is intended as a guide to the use of EMMYXL, the expression-oriented line-by-line assembler developed by Hedges for the Stanford Emulation Lab. It is intended to be used along with the Principles of Operation for the Stanford EMMY (TN # 65, Dec., 1975) and the EMMY/36Ø Assembler (TN #74, Dec. 1975). Various IBM OS/37Ø and VSII documents may also prove useful.

The work herein was supported in part by the Army Research Office-Durham under contract DAAG29-76-G-0001.

Table of Contents

## 1.1 EMMYXL

EMMYXL is an expression oriented, line-by-line assembler language which provides a concise, straightforeward expression for each possible EMMY operation. The two sides of each microinstruction are coded by seperate expressions delimited by a semicolon (";"). Conditional expressions are enclosed entirely in parenthesis (that is, a CONDITIONAL T-op would be coded by enclosing the entire instruction, both T- and A-side, in parenthesis. A conditional A-op branch would be coded by enclosing only the A-side in parenthesis). See TN#74.

The syntax of the EMMYXL language is presented in a seperate document (TN#74) and will not be repeated here. Some points will be clarified and discussed.

## 1.2 Expressions

An expression is a sequence of identifiers and symbols seperated by compile time operators (double operators, such as ++ or --). Expressions are evaluated left to right. More than two terms in an expression may lead to unpredctable results. Note, since expressions are evaluated right to left, the expression:

```
8 - 3 +  4
```

is evaluated to 8-(3+4)=1, rather than (8-3)+4=9.

Expressions may appear as arguments of DC and EQU pseudo ops. See TN#74 for a more detailed discussion.

## 1.3 Multiply Step

Operation of the EMMY multiply step is described here. Sample code for a 32 bit multiply is included in the Appendix.

The multiply step is coded as:

MUS(AF,BF)

where AF and BF are host register identifiers. A single multiply step proceeds as follows:

1) The double length value obtained by concatenating the AF register contents (placed in bits <63:32>) and AF ⊕ 1 contents (placed in bits <31:Ø>) form the PRODUCT. Thus, the AF operand specifies an even/odd register pair. Either the even or the odd register may be specified, and it becomes the high order 32 bits of the PRODUCT. The other register becomes the low order 32 bits. (e.g., if AF is specified as 3, the product is REG[3]|REG[2]). (Note: the symbol ⊕ denotes EXCLUSIVE OR).

2) The least significant bit of the product is saved, and the product shifted right arithmetically by one bit. If OVERFLOW was set (by the previous MUS), then the sign bit is inverted.

3) If the least significant bit of the product was one (before the shift), then the multiplier, obtained from REG[BF], is added to the high 32 bits of the product. OVERFLOW is set or reset as needed. If no addition is performed, OVERFLOW is reset.

4) PRODUCT<63:32> is returned to REG[AF] and PRODUCT<31:0> returned to REG[AF ⊕ 1].

Since the arithmetic shift preceeds the addition of the multiplier, an extra alignment step is necessary to properly align the 64 bit result. Thus, when multiplying two 32 bit numbers, 33 multiply steps are required, or 32 multiply steps followed by a right double arithmetic shift.

Programming Considerations

1) Operands may be either positive or negative two's complement values.

2) When doing arithmetic on quantities of fewer than 32 bits, post shifting of the result may be required.

3) The register specified as AF must be cleared to zero before the first MUS is executed. Register AF ⊕ 1 must contain the

multiplicand. Values should not be modified between multiply steps.

4) After the required number of multiply steps, REG[AF]|REG[AF ⊕ 1] contain the 64 bit binary product.

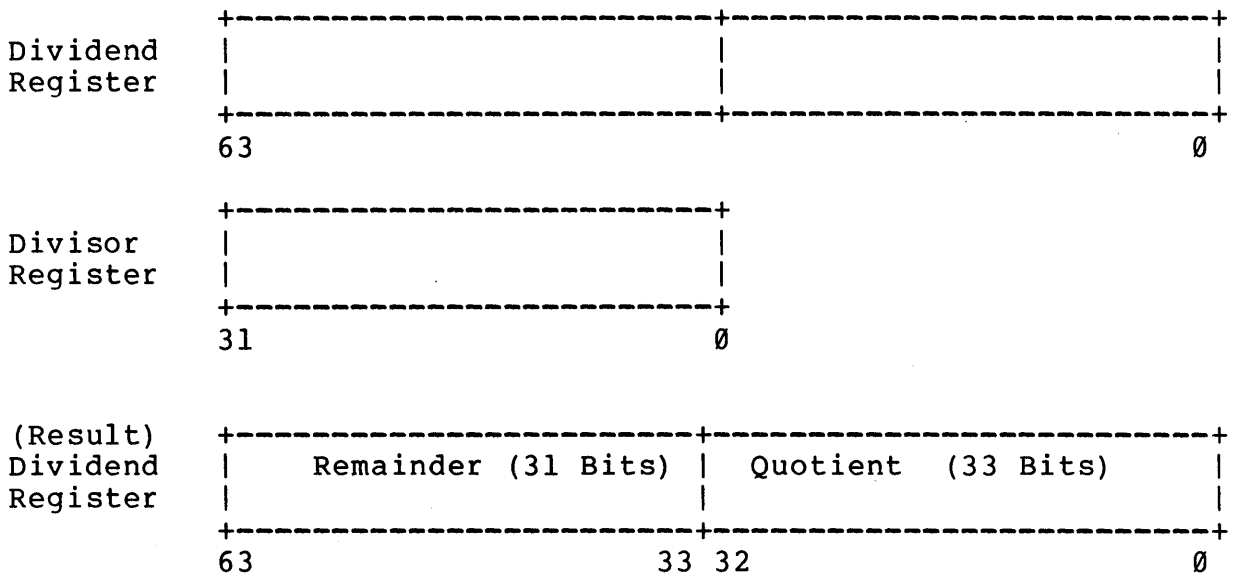
## 1.4 Divide Step

Operation of the EMMY Divide Step is described. Sample code for a 32 bit binary divide is included in the Appendix.

The EMMY Divide Step uses a restoring division algorithm to accomplish binary division of a 64 bit dividend by a 32 bit divisor. The result after 33 divide steps is a 33 bit quotient and a 31 bit remainder.

The algorithm proceeds in much the same way as ordinary long division. The divisor is subtracted from the high-order 32 bits of the dividend. If the result is >= Ø, it replaces the high-order 32 bits of the dividend. The dividend is then shifted left by one bit, shifting in a low-order one if the subtraction was successful (result replaced dividend<64:32>), or a zero if unsuccessful (dividend<64:32> unchanged). After 32 divide steps, the least significant bit of the dividend is aligned at the low end of the high 32 bits of the dividend register (bit <32:32>). One final divide step is required to calculate the remainder and least significant quotient bit. This final step shifts the remainder left by one bit however, placing the quotient most significant bit in bit <32:32> (the remainder now occupies bits <63:33> of the dividend register).

When processing very large binary numbers, the correct 33 bit quotient and 31 bit remainder result following 33 divide steps.




EMMY Divide Step Operation

Divide Step proceeds as follows:

1) Contents of Reg[AF]|Reg[AF ⊕ 1] form the DIVIDEND and Reg[BF] the DIVISOR.

2) DIVISOR is subtracted from DIVIDEND<63:32>. If the result is >=0, it replaces DIVIDEND<63:32>.

3) DIVIDEND is shifted LEFT LOGICALly by one bit. If result of step 2 was >=0, a one is shifted into the low-order bit

position, otherwise if the result of step 2 was <Ø, a zero is shifted in.

4) DIVIDEND<63:32> is returned to Reg[AF], and DIVIDEND<31:Ø> returned to Reg[AF ⊕ 1].

Programming Considerations

1) Reg[AF]|Reg[AF ⊕ 1] initially contain the 64 bit dividend, while Reg[BF] contains the 32 bit divisor.

2) Before entering any divide steps, be sure all operand values are positive. Complement any negative values, remembering which operands were complemented.

Note: dividend must be treated as a 64 bit 2's complement binary number. Sign bit is Reg[AF]<31:31>. It must be complemented as a 64 bit number, not two 31 bit numbers.

3) Divisor value must not be zero.

4) 33 Divide Steps are required to divide a 64 bit dividend by a 32 bit divisor.

5) After 33 Divide Steps:

Reg[AF]<Ø:Ø>|Reg[AF ⊕ 1]<31:Ø> contains the 33 bit quotient

Reg[AF]<31:1> contains the 31 bit remainder

Since the quotient will rarely be greater than 32 bits, the high order bit can usually be ignored, and Reg[AF] shifted right logically by one bit to right align the remainder in that register.

6) Divide Step is coded as follows:

DIV(AF,BF)

7) Overflow may occur if the number of significant bits in DIVIDEND<63:32> is more than one greater than the number of significant bits in the Divisor. (Quotient requires more than 33 bits). This condition will not be detected by the hardware.

8) For other than general case operands, fewer Divide Steps and some pre- and post-shifting may be used to accomplish division.

## 1.5 Branching and Conditional

The EMMY Processor contains two conditonally executed instructions, one a T-op, the other an A-op. In each case, an 8-bit mask and 3 modifier bits are specified. The modifier bits are denoted V, C, and S.

The mask and modifier bits dictate a test of either the CCODES or ICODES of RØ.

The mask specification is used to select a subset of the specified codes, ie CCODES or ICODES if S=Ø or 1 respectively. A code bit is selected for test if its corresponding mask bit is 1. If the C bit is one, the subset of codes is inverted.

The V bit controls the sense of the test. If V=Ø, the test will be true if any subset bit is true, otherwise the test fails. If V=1, the test will be true if all subset bits are zero, and false otherwise.

In summary, the mask is used to select a subset of the codes specified by S. If C=1, the subset is inverted. The test is true if V=Ø and any subset bit (or inverted subset bit if C=1) is one, or if V=1 and all subset bits (or inverted subset bits if C=1) are zero. Otherwise, the test fails.

## 1.51 T-Machine Conditional

The T-op conditional controls the execution of the A-op contained in bits <17:Ø>. If the conditional test specified proves true, the A-op is supressed; if the test proves false, the A-op is executed.

## 1.52 A-Machine Conditional

The A-machine conditional controls sequencing of microinstructions. A conditional test is specified as well as a 4-bit value, VAL. If the test proves true, then VAL, sign extended to 12 bits, is added to RØ<11:Ø>, the microinstruction counter. Foreward branches of up to 8 instructions beyond the current instruction (RØ + 7), or backward branches of up to 7 instructions prior to the current instruction (RØ - 8) are possible.

## Programming Considerations

1) Conditions which may be detected are "any selected bit one or zero" and "all selected bits one or zero".

2) To build a conditional:

i) set S to specify ICODES (S=1) or CCODES (S=Ø).

ii) set mask to select desired code bits

iii) for T-op conditional, if you want the A-op executed if any selected bit is one/zero, set V=1, otherwise, if you want the A-op executed if all selected bits are one/zero, set V=Ø. For the A-op branch, if you wish to branch if any selected bit is one/zero, set V=Ø, otherwise, if you wish to branch if all selected bits are one/zero, set V=1.

iv) All mask definitions should be referred to the T-side, as described above. The assembler will make any adjustments for "NOT" conditions and A-side branches.

v) If you wish to detect any selected

bit one or all selected bits zero, set C=Ø, otherwise, if you wish to detect any selected bit zero or all selected bits one, set C=1.

2) A conditional mask is defined in EMMYXL by EQUating a tag to a MASK function specified as follows:

```
<tag>      EQU      MASK(<mask>,<v>,<c>,<s>)
```

where <mask> is an absolute numeric value corresponding to the desired 8 bit mask, ie 255 ==> B'11111111' ==> select all bits. <v>,<c>,<s> specify V,C,and S bits and must be either one or zero. A mask function should not be coded directly in a conditional, rather EQUate a tag to the function and use the tag.

## 2.Ø Crossassembler Operation

This section describes the use of EMMYXL as implemented under IBM VSII. The actual assembler is currently written in ALGOLW, but will eventually be converted to PL36Ø.

### 2.1 Titles and Comments

Comments are coded by placing a period (".") prior to the first character of text. This causes the assembler to ignore the rest of the input card. This text is printed with the source listing. In addition, blank lines may be included to improve the readability of source lstings.

A page eject is indicated by a plus ("+") in column 1. If columns 2 through 61 are not blank, they become the new title and are printed at the top of all subsequent pages (until a new title is defined). If no new title is indicated, the previous one, if any, is printed.

### 2.2 Location Counter Control

All addresses processed by EMMYXL are EMMY bus addresses.

Thus, micromemory addresses are X'FF0000' through X'FF0FFF', and the host register file addresses X'FF1000' through X'FF1007'. Main store addresses are currently X'000000' through X'003FFF'. The assembler can be used to initialize any location in the EMMY system memory, including the registers and main store.

The assembler location counter can be set using the ORIGIN pseudo op. The argument of the ORG instruction must be a valid bus address or a symbol whose value is a valid bus address. All tags attached to machine instructions are assigned the current value of the location counter, which is, of cource, a bus address. When coding a micromemory location as a literal, be sure to code a control store address. (eg-to begin assembly at location X'100', code ORG X'FF0100').

## 2.3 Listing Options

By default, the assembler will produce a source listing during PASS 2. This listing consists of a title, which contains the language processor identifier and version date, a user supplied title, and page number The title/page eject card is discussed in section 2.1. Following the title are up to 55 lines of assembled source text. Column 1 contains an error flag, if there was any error in that source statement, or a space, followed

by the location counter value, which is an EMMY bus address. An 8 hexadecimal digit constant follows, representing the assembled object text. The card number and source text complete the line. Comments and assembler directives are printed with blank location counter and object text fields. EQU's are printed with blank location counter fields and the EQUated value in the object text field.

The source listing may be supressed by including a "&NOLIST" card anywhere in the source stream. This card in interpreted during PASS1 and will prevent any source listing from being produced.

Portions of a source listing can be supressed by coding "&NOPRINT" card just prior to the point at which the listing should be supressed, and placing a "&PRINT" card just prior to the point at which the listing should resume. These cards affect only the printed listing, not the assembled object text.

## 2.4 Code Generation

When the assembler directive "&CODE" is included in the source text, the assembler produces an object text file consisting of a bus address and text unpacked in ASCII representation of

hexadecimal values. This object file is included at the end of the assembler listing, preceeded by the title:

********** EMMY OBJECT LISTING **********

and several ASCII control characters for use in transmitting the object text to the lab's Datapoint terminal. See Uniterm II User's Guide for details.

The default is to generate no object text file. A "&CODE" card must be included if such an output is desired.

See Appendix C for listing formats and object text formats

## 3.Ø EMMY Simulator

The EMMYXL package includes a simulator program, whereby the EMMY code assembled can be tested. Operation of the simulator is the same as the actual EMMY.

When a "&SIM" card in included, and no errors occurred during assembly, the assembled object code is loaded into simulated control store, mainstore and host registers. Microinstructions are fetched from the location contained in RØ<11:Ø>. The starting oint for the simulation can be set by ORGing to RØ and defining a constant equal to that address, or as an argument of the END statement.

Simulation ends when the HALT bit becomes set, or an illegal operation is attempted. At this time, a post execution dump of micromemory and the host register file is printed.

When a "&TRACE" card in included, an instruction trace is performed as the simuaton progresses. Each instruction cycle, the microinstruction register and host registers are printed in hexadecimal.

Default conditions are NOSIM and NOTRACE

## References and Related Material

1.Ø Hedges, Thomas, EMMY/36Ø Cross Assembler, Stanford Technical Note #74, December, 1975.

2.Ø Neuhauser, Charles, Dynamic Microprogrammable Processor (Version III), Stanford Technical Note #65, December, 1975.

3.Ø Datapoint Corporation,Uniterm2/Uniterm3,3.1 User's Guide, June 27, 1974, Datapoint Corporation, 9725 Datapoint Dr, San Antonio, Texas 78284.

## Appendix A Error Flags

A Illegal A-statement

a) missing ")"

b) missing ";"

c) illegal syntax or operator (eg, missing "=")

d) missing expression or illegal "-"

B "BLK" statement error

a) <abs> not > Ø

C Illegal constant

a) Hex constant specifies other than Ø-9, A-F

b) Octal constant specifies other than Ø-7

c) Binary constant specifies other than Ø-1

d) illegal DC

D a) illegal conditional

b) address not > Ø in "ORG"

E a) illegal END card

b) illegal expression

I Insert/Extract error

a) specified field(s) can not be assembled into a legal literal mask (see L flag)

K illegal "MASK" function

a) missing "("

b) <cmask> or <bmask> out of range (not >=Ø,<=255)

c) V,C,S not Ø or 1

d) values not seperated by comma (",")

L illegal literal

a) literal does not specify 16 bit constant zero or one filled on right or left to 32 bits in expanded B-field

) literal out of range (not >= Ø, <=4Ø95 in LOAD IMMEDIATE)

M syntax error

a) composite operator illegally written (eg ":=" written as ": =")

P procedural error

a) literal out of range (lit not =>-8,<=7) in branch, pointer mod

b) illegal conditinal in pointer mod

Q undetermined error

R relocatability error

S PASS2 LC not = PASS1 LC - internal error has occurred

T illegal T-statement

a) illegal syntax

b) illegal operator

c) ilegal operand

U undefined symbol

X illegal pointer mod after indirect access

a) other than CF to DF specified

b) illegal syntax (ie not "+" or "-")

# Appendix B Using EMMYXL

All necessary JCL to use EMMYXL has been included in the cataloged procedure EMMYXL. The input specified must be a Wylbur EDIT format data set containing all text and control cards. This data set must be cataloged. If the name contains qualifiers (or indicies, in SCIP terminology), that is, qualifying name(s) seperated by period(s), it must be enclosed in single quotes in the EXEC statement. Additionally, no qualifier may be longer than 8 characters, nor begin with other than an alphabetic character (the same restrictions apply to simple names).

A GROUP and USER must be specified, reflecting the account under which the source data set is stored. The source data set must be cataloged

If the source text contains both upper and lower case characters, an uplow listing may be obtained by coding ",SYSOT=D" in the EXEC statement (note the misspelling of SYSOUT). Only the assembler listing will be printed uplow, so watch for two listings.

examples:

```
// EXEC EMMYXL,SOURCE=myfile,GROUP=gg,USER=uuu
```

file accessed is 'WYL.gg.uu.myfile'. Listing is upper case only.

```
// EXEC EMMYXL,SOURCE=myprog.versl,USER=uuu,GROUP=gg
```

illegal - SOURCE ia a qualified name (contains special character,".") and is not enclosed in single quotes.

```
// EXEC EMMYXL,SOURCE=360emu,GROUP=gg,USER=uuu,SYSOT=d
```

illegal - SOURCE contains qualifier which begins with numeric.

```
// EXEC EMMYXL,SOURCE='text.versl',GROUP=gg,USER=uuu
```

file accessed is 'WYL.gg.uuu.text.versl', Listing is upper case only.

The EMMYXL procedure executes two job steps. The first UNPRESSes the EDIT format source and creates a card-image scratch data set, which is PASSed to the second step.

The second job step reads the scratch data set as input to the assembler. This data set is scratched at step termination.

The original source, however, is kept.

```
//MXFKU071 JOB MXF$KU,'EMMYXL DEMO',TIME=(0,5)                    JOB 1322
//  EXEC EMMYXL,SOURCE='MULTDIV',USER=MXF,GROUP=KU  (1)             2.
XXEMMYXL   PROC  SOURCE=,GROUP=,USER=,SYSOT=A
*********************************************************************
*********************************************************************
******                                                          ***
******      EMMYXL 370 CROSS ASSEMBLER FOR STANFORD EMMY        ***
******                                                          ***
******      SOURCE IS A WYLBUR EDIT FORMAT DATA SET WHICH       ***
******          CONTAINS SOURCE TEXT AND EMMYXL CONTROL CARDS.  ***
******      QUALIFIERS MUST BE <=8 CHARACTERS AND BEGIN WITH    ***
******          ALPHABETIC CHARACTER.                           ***
******          DATA SET MUST ALSO BE CATALOGED.                ***
******                                                          ***
******         SYSOT IS SYSOUT CLASS FOR ASSEMBLER LISTING      ***
******                                                          ***
******    CONSULTANT:      WALTER WALLACH                       ***
******    AFFILIATION:     DIGITAL SYSTEMS LAB-ELECT.ENGR.      ***
******    TELEPHONE:       (49)7-0377                           ***
******    ADDRESS:         ERL 322                              ***
******    SUBMITTED:       03/11/76                             ***
******    EXPIRES:                                              ***
******                                                          ***
*********************************************************************
*********************************************************************
XXUNPRES EXEC  PGM=IOPROGM
XXSTEPLIB  DD  DSN=SYS4.IOPROGM.LIB,DISP=SHR        (2)UNPRESS Step -unpress EDIT format text and create card image file
XXSYSPRINT DD  SYSOUT=A
XXSYSPUNCH DD  SYSOUT=B
XXINPUT    DD  DSN=WYL.&GROUP..&USER..&SOURCE,DISP=SHR
IEF653I SUBSTITUTION JCL - DSN=WYL.KU.MXF.MULTDIV,DISP=SHR
XXOUTPUT   DD  UNIT=SYSDA,SPACE=(3200,(30,10)),DISP=(NEW,PASS),
XX             DCB=(LRECL=80,BLKSIZE=3200,RECFM=FB)
XXSYSIN    DD  DSN='WYL.D2.N27.COMMAND',DISP=SHR
IEF236I ALLOC. FOR MXFKU071 UNPRES
IEF237I 16E   ALLOCATED TO STEPLIB
IEF237I D36   ALLOCATED TO SYSPRINT
IEF237I D23   ALLOCATED TO SYSPUNCH
IEF237I 17C   ALLOCATED TO INPUT
IEF237I 51F   ALLOCATED TO OUTPUT
IEF237I 17C   ALLOCATED TO SYSIN
IEF142I - STEP WAS EXECUTED - COND CODE 0000


XXEMMY   EXEC  PGM=EMMYXL,PARM='PAGES=500,SIZE=128,TIME=30'
XXSTEPLIB  DD  DSN=WYL.C2.N27.EMMYXL.JOBLIB,DISP=SHR        (3)Assemble, then scratch card image file
XXSYSPRINT DD  SYSOUT=&SYSOT
IEF653I SUBSTITUTION JCL - SYSOUT=A
XXFT10F001 DD  UNIT=SYSDA,SPACE=(3404,(125,25)),DISP=(NEW,DELETE),
XX             DCB=(RECFM=VBS,BLKSIZE=3404)
XXFT05F001 DD  DDNAME=SYSIN
XXSYSIN    DD  DSN=*.UNPRES.OUTPUT,DISP=(OLD,DELETE)
//                                                               3.
IEF236I ALLOC. FOR MXFKU071 EMMY
IEF237I 17F   ALLOCATED TO STEPLIB
IEF237I D36   ALLOCATED TO SYSPRINT
IEF237I 51E   ALLOCATED TO FT10F001
IEF237I 51F   ALLOCATED TO FT05F001
IEC130I SYSIN     DD STATEMENT MISSING
IEC130I SYSPUNCH DD STATEMENT MISSING
IEF142I - STEP WAS EXECUTED - COND CODE 0000
```

```
EMMY X L   3/28/76                EMMYXL EXAMPLE 1 FIXED POINT MULTIPLY                    PAGE   1
                                                                                                    2.000
                                                                                                    3.000
                                                                                                    4.000
                                                                                                    5.000
                                  .REGIATER DEFINITIONS                                             6.000
                                                                                                    7.000
           00000000               MAR             EQU   R0                                         8.000
           00000002               XR              EQU   R2                                         9.000
           00000004               P               EQU   R4     .DIVIDEND HIGH HALF                10.000
           00000005               Q               EQU   R5     .DIVIDEND LOW HALF                 11.000
           00000007               S               EQU   R7     .DIVISOR                           12.000
                                                                                                   13.000
                                  .P,Q ALSO FORM THE PRODUCT REGISTER (64 BITS)                     14.000
                                  .S IS ALSO THE MULTIPLICAND REGISTER, Q THE MULTIPLIER            15.000
                                                                                                   16.000
FF0000                                        ORG   X'FF0000'      .START AT LOCATION 0           17.000
                                  MULT:                                                            18.000
                                                                                                   19.000
                                  .FIXED POINT MULTIPLY   32 BIT OPERANDS                          20.000
                                                                                                   21.000
                                  .OPERANDS ASSUMED IN REGISTERS  Q (MULTIPLIER) AND S (MULTIPL'ND) 22.000
                                                                                                   23.000
FF0000   0C13A020                   P := 0        ; XR = 32       .CLEAR PRODUCT<63:32> SET ITERATION CTR 24.000
                                                                                                   25.000
FF0001   6FF220BF                   MUS(P,S)      ; DEC XR (¬<0 => MAR-1) .33 ITERATIONS           26.000
                                                                                                   27.000
                                                                                                   28.000
```

1. Language Processor Identifier and Version Date
2. User Title
3. Flag (none in this assembly)
4. Location (bus address)
5. Object Code (or value, in the case of EQU, Mask)
6. Input Card Number
7. Source Text

```
                        30
                        31  .THE DIVIDE ALGORITHM WILL WORK WITH POSITIVE VALUES ONLY.                    30.000
                        32  .THEREFORE, WE MUST COMPLEMENT NEGATIVE VALUES AND KEEP TRACK OF              31.000
                        33  .WHICH VALUES WERE COMPLEMENTED.  THE REMAINDER MUST BE COMPLEMENTED          32.000
                        34  .IF THE DIVIDEND WAS NEGATIVE, AND THE QUOTIENT COMPLEMENTED IF               33.000
                        35  .EITHER THE DIVIDEND OR DIVISOR, BUT NOT BOTH, WERE NEGATIVE.                 34.000
                        36  .ICCDE BIT <1:1> WILL INDICATE REMAINDER TO BE COMPLEMENTED, AND              35.000
                        37  .ICCDE BIT <0:0> INDICATE QUOTIENT TO BE COMPLEMENTED.                        36.000
                        38                                                                                37.000
 00000011               39  CCMPLREM    EQU   MASK(2,0,0,1)   .FLAG FOR REMAINDER                       38.000
 00000009               40  NEGT        EQU   MASK(1,0,0,1)   .FLAG FOR QUOTIENT                        39.000
                        41                                                                                40.000
 FF0002  6D01F018       42                ; S = M(DIVISOR)     .LOAD DIVISOR VALUE                     41.000
 FF0003  6D01C019       43                ; P = M(DIVID1)      .LOAD DIVIDEND HIGH HALF                42.000
 FF0004  6D01D01A       44                ; Q = M(DIVID2)      .LOAD DIVIDEND LOW HALF                 43.000
                        45                                                                                44.000
 FF0005  0AFC6001       46    S := S      ; ( NOT ZERO => MAR+1 )   .IS DIVISOR ZERO?                     45.000
 FF0006  18008000       47      MAR := MAR OR X'00008000'           .YES - HALT                         46.000
                        48                                                                                47.000
 FF0007  C283800A       49    ( POSITIVE => ; MAR = * ++ 3 )        .IS DIVISOR NEGATIVE?               48.000
 FF0008  19020001       50    MAR := MAR XOR X'00010000'          .YES - SET QUOTIENT FLAG              49.000
 FF0009  00FE7000       51    S := ¬S     ; INC S                 .COMPLEMENT - IGNORE OFL              50.000
                        52                                                                                51.000
 FF000A  0A900845       53    P := P      ; (NOT NEGATIVE => MAR+5) .IS DIVIDEND NEG?                   52.000
 FF000B  19020003       54    MAR := MAR XOR X'00030000'          .YES INVERT QUOTIENT FLAG             53.000
                        55                                        .SET REMAINDER FLAG                   54.000
 FF000C  00866060       56    Q := ¬Q                                                                   55.000
 FF000D  00928060       57    P := ¬P                             .ONES COMPL                           56.000
 FF000E  32140001       58    Q := Q+1                            .INCREMENT 64 BIT DIVIDEND            57.000
 FF000F  3310C000       59    P := P +C+ 0                                                              58.000
                        60                                                                                59.000
                        61                                                                                60.000
 FF0010  14080020       62    XR := 32                            .ITERATION CNTR                       61.000
                        63                                                                                62.000
 FF0011  6CF2208F       64    DIV(P,S)    ; DEC XR (¬<0 => MAR-1) .33 ITERATIONS                        63.000
 FF0012  52100001       65      P >> 1                          .TRUNCATE QUOTIENT TO 32 BITS         64.000
                        66                                      .AND LEAVE REMAINDER RIGHT JUST       65.000
                        67                                                                            66.000
 FF0013  C0478015       68    ( COMPLREM => ; MAR=*++2)  .MUST WE COMPLEMENT REM?                       67.000
 FF0014  C0924000       69    P := ¬P     ; INC P           .YES - IGNORE OFL                           68.000
                        70                                                                            69.000
 FF0015  C0278017       71    ( NEGT =>   ; MAR=*++2)       .HOW ABOUT QUOTIENT?                    70.000
 FF0016  00865000       72    Q := ¬Q     ; INC Q           .YES - IGNORE OFL                           71.000
                        73                                                                            72.000
                        74  .NOW Q CONTAINS QUOTIENT AND P CONTAINS REMAINDER                            73.000
                        75                                                                            74.000
 FF0017  18008000       76    MAR := MAR OR X'00008000'       .HALT                                   75.000
                        77                                                                            76.000
 FF0018  0000000D       78  DIVISOR:      DC   13                                                     77.000
 FF0019  FFFFFFFF       79  DIVID1:       DC   --1                                                    78.000
 FF001A  FFFFFF2A       80  DIVID2:       DC   --214                                                  79.000
```

```
                        82  .NOW INITIALIZE HOST REGISTERS                                  82.000
                        83                                                                  83.000
  FF1000                84          ORG   X'FF1000'                                       84.000
  FF1000  00000000      85          DC    X'00000000'  (1)  .R0 START AT LOCATION 0       85.000
  FF1001  00000003      86          BLK   3                 .UNUSED REGISTERS             86.000
  FF1004  00000000      87          DC    0                                               87.000
  FF1005  0000002D      88          DC    45                .Q-REG - MULTIPLIER           88.000
  FF1006  00000000      89          DC    0                 .R-REG NOT USED               89.000
  FF1007  0000012A      90          DC    298               .S-REG MULTIPLICAND           90.000
  000000                91          END                                                   91.000
```

1 Reset Location Counter to Host Register File

```
BUSY        MASK   0000000C
CARRY       MASK   00000104
COMPLREM    MASK   00000011
DIVID1      SYMB   00FF0019
DIVID2      SYMB   00FF001A
DIVISOR     SYMB   00FF0018
HIGH        MASK   00000084
LOW         MASK   00000044
MAR         REG    00000000
MULT        SYMB   00FFC000
NEGATIVE    MASK   00000084
NEGT        MASK   00000009
ODD         MASK   00000014
OVERFLOW    MASK   00000602
P           REG    00000004
POSITIVE    MASK   000000A0
C           REG    00000005
R0          REG    00000000
R1          REG    00000001
R2          REG    00000002
R3          REG    00000003
R4          REG    00000004
R5          REG    00000005
R6          REG    00000006
R7          REG    00000007
S           REG    00000007
SAME        MASK   00000024
XR          REG    00000002
ZERO        MASK   00000600
  (1)       (2)      (3)
```

EMMY Symbol Table

1 Tag or Name

2 Identifier Type

3 value (in case of EQU, MASK, etc) or location (in case of Tag)

## Appendix D Sample Object Code and Object File Format

EMMYXL will produce an object text file at the end of the source listing if the assembler directive "&CODE" was included in the source. This file consists of EMMY bus addresses and object text, unpacked into ASCII representation of hexadecimal values. Thus, the hexadecimal value X'9C' would appear as two ASCII characters "9" and "C" printed as part of a string of such characters. These must be packed and converted to thier binary equivalents before loading into the EMMY memory system. A loader program is incorporated in the console debug program which accepts this object format, performs the necessary conversions, and loads the text into the EMMY system.

Object text records consist of a six character address, preceeded by an address identifier character ("@"). This is followed by up to 64 characters of object text. Each object text record contains an address.

The object text is preceeded by two control characters, which are used to control the tape unit of the Datapoint 2200. The Acsii "ENQ" is interpreted by the Uniterm (r) program as "enable cassette" (X'2D'), and enables data to be written to the front cassette unit. The second control character, "DC-2" or "TAPE-ON" (X'12') actually begins writing data to the cassette.

Following the object text appear two control characters which are the complement of the first two. A "DC-4" or "TAPE-OFF" stops writing data to the cassette and writes the last record (purges the Datapoint buffer), and an "EOT" (X'37') disables the cassette unit.

### Object File Format

```
"ENQ"    (X'2D')
"TAPE-ON" (X'12')
      any number of object text records
"TAPE-OFF" (X'3C')
"EOT" (X'37')
```

This follows the title "********** EMMY OBJECT CODE **********" in the source listing.

```
********** EMMY OBJECT CODE **********
      <Control Characters (unprintable) >
@FF0000 0C13A0206FF220BF6D01F0186D01C0196D01D01A0AFC60011B008000C283800A
@FF0008 1902000100FE70000A900845190200030B6806000928060321400013310000 0
@FF0010 1AD800206CF220BF52100001C04780150092400DC027801700865000180080 00
@FF0018 000C0CCDFFFFFFFFFFFFFF2A
@FF1000 00000000000000030000000C0000000000000000000002D000000000000012A

      <Control Characters (unprintable)>

*** END OF ASSEMBLY ***

000.67 SECONDS ASSEMBLY TIME

    NO STATEMENTS FLAGGED
```

INSTRUCTION TRACE

```
IR= 0C13A020 R0 00000001 R1 00000003 R2 00000000 R3 00000000 R4 00000000 R5 0000002D R6 00000000 R7 0000012A
IR= 6FF220BF R0 04000002 R1 00000003 R2 00000020 R3 00000000 R4 00000000 R5 0000002D R6 00000000 R7 0000012A
IR= 6FF220BF R0 04000002 R1 00000003 R2 0000001F R3 00000000 R4 0000012A R5 00000016 R6 00000000 R7 0000012A
IR= 6FF220BF R0 04000002 R1 00000003 R2 0000001E R3 00000000 R4 00000095 R5 0000000B R6 00000000 R7 0000012A
IR= 6FF220BF R0 04000002 R1 00000003 R2 0000001D R3 00000000 R4 00000174 R5 80000005 R6 00000000 R7 0000012A
IR= 6FF220BF R0 04000002 R1 00000003 R2 0000001C R3 00000000 R4 000001E4 R5 40000002 R6 00000000 R7 0000012A
IR= 6FF220BF R0 04000002 R1 00000003 R2 00000018 R3 00000000 R4 000000F2 R5 20000001 R6 00000000 R7 0000012A
IR= 6FF220BF R0 04000002 R1 00000003 R2 0000001A R3 00000000 R4 000001A3 R5 10000000 R6 00000000 R7 0000012A
IR= 6FF220BF R0 04000002 R1 00000003 R2 00000019 R3 00000000 R4 000000D1 R5 88000000 R6 00000000 R7 0000012A
IR= 6FF220BF R0 04000002 R1 00000003 R2 00000018 R3 00000000 R4 00000068 R5 C4000000 R6 00000000 R7 0000012A
IR= 6FF220BF R0 04000002 R1 00000003 R2 00000017 R3 00000000 R4 00000034 R5 62000000 R6 00000000 R7 0000012A
IR= 6FF220BF R0 04000002 R1 00000003 R2 00000016 R3 00000000 R4 0000001A R5 31000000 R6 00000000 R7 0000012A
IR= 6FF220BF R0 04000002 R1 00000003 R2 00000015 R3 00000000 R4 0000000D R5 18800000 R6 00000000 R7 0000012A  (1)
IR= 6FF220BF R0 04000002 R1 00000003 R2 00000014 R3 00000000 R4 00000006 R5 8C400000 R6 00000000 R7 0000012A
IR= 6FF220BF R0 04000002 R1 00000003 R2 00000013 R3 00000000 R4 00000003 R5 46200000 R6 00000000 R7 0000012A
IR= 6FF220BF R0 04000002 R1 00000003 R2 00000012 R3 00000000 R4 00000001 R5 A3100000 R6 00000000 R7 0000012A
IR= 6FF220BF R0 04000002 R1 00000003 R2 00000011 R3 00000000 R4 00000000 R5 01880000 R6 00000000 R7 0000012A
IR= 6FF220BF R0 04000002 R1 00000003 R2 00000010 R3 00000000 R4 00000000 R5 68C40000 R6 00000000 R7 0000012A
IR= 6FF220BF R0 04000002 R1 00000003 R2 0000000F R3 00000000 R4 00000000 R5 34620000 R6 00000000 R7 0000012A
IR= 6FF220BF R0 04000002 R1 00000003 R2 0000000E R3 00000000 R4 00000000 R5 1A310000 R6 00000000 R7 0000012A
IR= 6FF220BF R0 04000002 R1 00000003 R2 0000000D R3 00000000 R4 00000000 R5 0D188000 R6 00000000 R7 0000012A
IR= 6FF220BF R0 04000002 R1 00000003 R2 0000000C R3 00000000 R4 00000000 R5 068C4000 R6 00000000 R7 0000012A
IR= 6FF220BF R0 04000002 R1 00000003 R2 0000000B R3 00000000 R4 00000000 R5 03462000 R6 00000000 R7 0000012A
IR= 6FF220BF R0 04000002 R1 00000003 R2 0000000A R3 00000000 R4 00000000 R5 01A31000 R6 00000000 R7 0000012A
IR= 6FF220BF R0 04000002 R1 00000003 R2 00000009 R3 00000000 R4 00000000 R5 00D18800 R6 00000000 R7 0000012A
IR= 6FF220BF R0 04000002 R1 00000003 R2 00000008 R3 00000000 R4 00000000 R5 0068C400 R6 00000000 R7 0000012A
IR= 6FF220BF R0 04000002 R1 00000003 R2 00000007 R3 00000000 R4 00000000 R5 00346200 R6 00000000 R7 0000012A
IR= 6FF220BF R0 04000002 R1 00000003 R2 00000006 R3 00000000 R4 00000000 R5 001A3100 R6 00000000 R7 0000012A
IR= 6FF220BF R0 04000002 R1 00000003 R2 00000005 R3 00000000 R4 00000000 R5 000D1880 R6 00000000 R7 0000012A
IR= 6FF220BF R0 04000002 R1 00000003 R2 00000004 R3 00000000 R4 00000000 R5 00068C40 R6 00000000 R7 0000012A
IR= 6FF220BF R0 04000002 R1 00000003 R2 00000003 R3 00000000 R4 00000000 R5 00034620 R6 00000000 R7 0000012A
IR= 6FF220BF R0 04000002 R1 00000003 R2 00000002 R3 00000000 R4 00000000 R5 0001A310 R6 00000000 R7 0000012A
IR= 6FF220BF R0 04000002 R1 00000003 R2 00000001 R3 00000000 R4 00000000 R5 0000D188 R6 00000000 R7 0000012A
IR= 6FF220BF R0 04000002 R1 00000003 R2 00000000 R3 00000000 R4 00000000 R5 00006BC4 R6 00000000 R7 0000012A
IR= 6D01F018 R0 04000003 R1 00000003 R2 FFFFFFFF R3 00000000 R4 00000000 R5 00003462 R6 00000000 R7 0000012A
IR= 6D01C019 R0 04000004 R1 00000003 R2 FFFFFFFF R3 00000000 R4 00000000 R5 00003462 R6 00000000 R7 0000000D  (2)
IR= 6D01D01A R0 04000005 R1 00000003 R2 FFFFFFFF R3 00000000 R4 FFFFFFFF R5 00003462 R6 00000000 R7 0000000D
IR= 0AFC6001 R0 04000006 R1 00000003 R2 FFFFFFFF R3 00000000 R4 FFFFFFFF R5 FFFFFF2A R6 00000000 R7 0000000D  (3)
IR= C2838UCA R0 88000008 R1 00000003 R2 FFFFFFFF R3 00000000 R4 FFFFFFFF R5 FFFFFF2A R6 00000000 R7 0000000D
IR= 0A900845 R0 8800000B R1 00000003 R2 FFFFFFFF R3 00000000 R4 FFFFFFFF R5 FFFFFF2A R6 00000000 R7 0000000D
IR= 19020003 R0 5E00000C R1 00000003 R2 FFFFFFFF R3 00000000 R4 FFFFFFFF R5 FFFFFF2A R6 00000000 R7 0000000D
IR= 00B68060 R0 82030000 R1 00000003 R2 FFFFFFFF R3 00000000 R4 FFFFFFFF R5 FFFFFF2A R6 00000000 R7 0000000D
IR= 00928060 R0 8A03000E R1 00000003 R2 FFFFFFFF R3 00000000 R4 FFFFFFFF R5 00000005 R6 00000000 R7 0000000D
IR= 32140001 R0 0403000F R1 00000003 R2 FFFFFFFF R3 00000000 R4 00000000 R5 000000D5 R6 00000000 R7 0000000D
IR= 33100000 R0 82030010 R1 00000003 R2 FFFFFFFF R3 00000000 R4 00000000 R5 000000D6 R6 00000000 R7 0000000D  (4)
IR= 1A080020 R0 04030011 R1 00000003 R2 FFFFFFFF R3 00000000 R4 00000000 R5 000000D6 R6 00000000 R7 0000000D
IR= 6CF220BF R0 82030012 R1 00000003 R2 00000020 R3 00000000 R4 00000000 R5 000000D6 R6 00000000 R7 0000000D
IR= 6CF220BF R0 82030012 R1 00000003 R2 0000001F R3 00000000 R4 00000000 R5 000001AC R6 00000000 R7 0000000D
IR= 6CF220BF R0 82030012 R1 00000003 R2 0000001E R3 00000000 R4 00000000 R5 00000358 R6 00000000 R7 0000000D
IR= 6CF220BF R0 82030012 R1 00000003 R2 0000001D R3 00000000 R4 00000000 R5 000006B0 R6 00000000 R7 0000000D
IR= 6CF220BF R0 82030012 R1 00000003 R2 0000001C R3 00000000 R4 00000000 R5 00000D60 R6 00000000 R7 0000000D
IR= 6CF220BF R0 82030012 R1 00000003 R2 00000018 R3 00000000 R4 00000000 R5 00001AC0 R6 00000000 R7 0000000D
IR= 6CF220BF R0 82030012 R1 00000003 R2 0000001A R3 00000000 R4 00000000 R5 00003580 R6 00000000 R7 0000000D
IR= 6CF220BF R0 82030012 R1 00000003 R2 00000019 R3 00000000 R4 00000000 R5 00006B00 R6 00000000 R7 0000000D
IR= 6CF220BF R0 82030012 R1 00000003 R2 00000018 R3 00000000 R4 00000000 R5 0000D600 R6 00000000 R7 0000000D
IR= 6CF220BF R0 82030012 R1 00000003 R2 00000017 R3 00000000 R4 00000000 R5 0001AC00 R6 00000000 R7 0000000D  (5)
IR= 6CF220BF R0 82030012 R1 00000003 R2 00000016 R3 00000000 R4 00000000 R5 00035800 R6 00000000 R7 0000000D
IR= 6CF220BF R0 82030012 R1 00000003 R2 00000015 R3 00000000 R4 00000000 R5 0006B000 R6 00000000 R7 0000000D
IR= 6CF220BF R0 82030012 R1 00000003 R2 00000014 R3 00000000 R4 00000000 R5 000D6000 R6 00000000 R7 0000000D
IR= 6CF220BF R0 82030012 R1 00000003 R2 00000013 R3 00000000 R4 00000000 R5 001AC000 R6 00000000 R7 0000000D
```

```
IR= 6CF220BF R0 82030012 R1 00000003 R2 00000012 R3 00000000 R4 00000000 R5 00358000 R6 00000000 R7 0000000D
IR= 6CF220BF R0 82030012 R1 00000003 R2 00000011 R3 00000000 R4 00000000 R5 006B0000 R6 00000000 R7 0000000D
IR= 6CF220BF R0 82030012 R1 00000003 R2 00000010 R3 00000000 R4 00000000 R5 00D60000 R6 00000000 R7 0000000D
IR= 6CF220BF R0 82030012 R1 00000003 R2 0000000F R3 00000000 R4 00000000 R5 01AC0000 R6 00000000 R7 0000000D
IR= 6CF220BF R0 82030012 R1 00000003 R2 0000000E R3 00000000 R4 00000000 R5 03580000 R6 00000000 R7 0000000D
IR= 6CF220BF R0 82030012 R1 00000003 R2 0000000D R3 00000000 R4 00000000 R5 06B00000 R6 00000000 R7 0000000D
IR= 6CF220BF R0 82030012 R1 00000003 R2 0000000C R3 00000000 R4 00000000 R5 0D600000 R6 00000000 R7 0000000D
IR= 6CF220BF R0 82030012 R1 00000003 R2 0000000B R3 00000000 R4 00000000 R5 1AC00000 R6 00000000 R7 0000000D
IR= 6CF220BF R0 82030012 R1 00000003 R2 0000000A R3 00000000 R4 00000000 R5 35800000 R6 00000000 R7 0000000D
IR= 6CF220BF R0 82030012 R1 00000003 R2 00000009 R3 00000000 R4 00000000 R5 6B000000 R6 00000000 R7 0000000D
IR= 6CF220BF R0 82030012 R1 00000003 R2 00000008 R3 00000000 R4 00000000 R5 D6000000 R6 00000000 R7 0000000D
IR= 6CF220BF R0 82030012 R1 00000003 R2 00000007 R3 00000000 R4 00000001 R5 AC000000 R6 00000000 R7 0000000D
IR= 6CF220BF R0 82030012 R1 00000003 R2 00000006 R3 00000000 R4 00000003 R5 58000000 R6 00000000 R7 0000000D
IR= 6CF220BF R0 82030012 R1 00000003 R2 00000005 R3 00000000 R4 00000006 R5 B0000000 R6 00000000 R7 0000000D
IR= 6CF220BF R0 82030012 R1 00000003 R2 00000004 R3 00000000 R4 0000000D R5 60000000 R6 00000000 R7 0000000D
IR= 6CF220BF R0 82030012 R1 00000003 R2 00000003 R3 00000000 R4 00000000 R5 C0000001 R6 00000000 R7 0000000D
IR= 6CF220BF R0 82030012 R1 00000003 R2 00000002 R3 00000000 R4 00000001 R5 80000002 R6 00000000 R7 0000000D
IR= 6CF220BF R0 82030012 R1 00000003 R2 00000001 R3 00000000 R4 00000003 R5 00000004 R6 00000000 R7 0000000D
IR= 6CF220BF R0 82030012 R1 00000003 R2 00000000 R3 00000000 R4 00000006 R5 00000008 R6 00000000 R7 0000000D
IR= 52100001 R0 82030013 R1 00000003 R2 FFFFFFFF R3 00000000 R4 0000000C R5 00000010 R6 00000000 R7 0000000D
IR= C0478015 R0 82030014 R1 00000003 R2 FFFFFFFF R3 00000000 R4 00000006 R5 00000010 R6 00000000 R7 0000000D
IR= 00924000 R0 82030015 R1 00000003 R2 FFFFFFFF R3 00000000 R4 00000006 R5 00000010 R6 00000000 R7 0000000D
IR= C0278017 R0 58030016 R1 00000003 R2 FFFFFFFF R3 00000000 R4 FFFFFFFA R5 00000010 R6 00000000 R7 0000000D
IR= 30B65000 R0 58030017 R1 00000003 R2 FFFFFFFF R3 00000000 R4 FFFFFFFA R5 00000010 R6 00000000 R7 0000000D
IR= 1B008000 R0 5A030018 R1 00000003 R2 FFFFFFFF R3 00000000 R4 FFFFFFFA R5 FFFFFFF0 R6 00000000 R7 0000000D

NORMAL END OF SIMULATION

000.10 SECONDS SIMULATION TIME
```

(5)

(6)



1 33 Multiply Steps

2 Result = X'3462' = 13,410 = 45*298

3,4 Load values for Division
Complement Dividend and set flag in R0

5 33 Divide Steps

6 Complement Quotient and Remainder answer =-16 remainder -6

```
                    *** EMMY MEMORY DUMP ***

R0= 80038018 R1= 00000003 R2= FFFFFFFF R3= 00000000 R4= FFFFFFFA R5= FFFFFFF0 R6= 00000000 R7= 0000000D

000000   0C13A020  6FF220BF  6DC1F018  6D01C019  6D01D01A  0AFC6001  18008000  C283800A   *    OR ?M P M @ M P  !      B   *
000008   1902C001  00FE7000  0A900845  19020003  00B6806D  00928060  32140001  33100000   *     ¬P    E    6      2   3   *
000010   1AC80020  6CF220BF  52100001  C0478015  00924000  C0278017  00B65000  18008000   *    LR ?R   @G    @ @'   6P      *
000018   00000000  FFFFFFFF  FFFFFF2A  F8F8F8F8  F8F8F8F8  F8F8F8F8  F8F8F8F8  F8F8F8F8   *           *                      *

 *** END OF DUMP ***

000.78 SECONDS IN EXECUTION
```

## Appendix F: CONTROL CARD SUMMARY

| | |
|---|---|
| • <text> | - period followed by text - comments ignored by EMMYXL |
| + [<text>] | - plus sign in column 1 - page eject text (if any) replaces user title at head of page |
| ORG {<tag> / <abs>} | - set location counter - <abs> must be EMMY bus address |
| &NOLIST | - supress entire source listing |
| &NOPRINT | - turn off printing of that protion of source following the &NOPRINT card |
| &PRINT | - resume printing of all source text |
| &CODE | - produce EMMY object code |
| &SIM | - simulate assembled EMMY program if no errors occurred during assembly |
| &TRACE | - produce instruction trace during simulation |

Note: all "&" control cards and title card must begin in column 1.