MODC2 Procedures for Assembly of MODCOMP-II Programs Using the Sigma 5 Assembler

J. W. Layland
Communications Systems Research Section

This article describes a set of programs which have been written to enable the METASYMBOL assembler of the Sigma 5 to assemble programs for an attached MODCOMP-II minicomputer. This program set is a follow-on to previously developed program sets which facilitated assemblies for the PDP-11 and SDS-930.

I. Introduction

The METASYMBOL assembler for the Sigma 5 is a very powerful macro-assembler which we have used in the past to build programs for the SDS 920/930's or for the PDP-11 minicomputer (Ref. 1). The flexibility of this assembler has more recently been used to build programs for the MODCOMP-II. In this way, the more powerful and familiar features of the Sigma assembler are made available for minicomputer software development without requiring a user to become familiar with the total software system of the minicomputer, which he may not need for his application. One potential pitfall here is that software developed with the Sigma host assembler cannot be readily maintained on the MODCOMP, or vice versa, due to lexical differences of the two assemblers. This potential problem is not deemed serious for our application, which will use MODCOMP without peripherals, which is to be direct-link connected to the Sigma 5.

The assembly package for the MODCOMP consists of two parts: a system procedure deck "MODC2," which allows METASYMBOL to assemble a source language similar to the MODCOMP's native assembler, and a secondary loader which reformats the Sigma 5 core-image load module into proper binary format for loading into the target minicomputer and punches it onto paper tape or cards. Figure 1 describes the operation flow for use of MODC2.

The procedure deck defines the valid operators to the Sigma 5 METASYMBOL assembler and determines what code will be generated for the valid source statements. METASYMBOL procedures are similar to macro definitions. The code produced from the source program under control of the procedures is formatted by METASYMBOL into a Sigma relocatable object module (ROM) containing relocation information, external references and definitions, and generated code.

A number of read-only memories (ROMs) may then be linked together, and the external references and definitions resolved by the Sigma loader. Normally the loader gives the user the option of saving relocation information, creating a task-control block (TCB), and satisfying unresolved external references from the system library. These are SIGMA-oriented functions and should be disallowed during loading for MODCOMP-II programs by specifying the options (ABS), (NOTCB), and (NOSYSLIB) on the load control card. The Sigma loader also has the capability of relocating the program to any boundary which is a multiple of 800 (hex) bytes. It will automatically relocate to the background lower limit unless the BIAS option is specified on the load card; (BIAS, 0) will cause the first ROM to be not relocated. The Sigma loader structures its output into a file called a load module (LMN), which consists of the core image program and several records of control information. The Sigma 5 has write protection, so the core image is in several pieces, one for each protection type.

At present, we are using the secondary loader "SLOAD: DSN" developed for the PDP-11 (Ref. 1) to format and punch the developed program into portable form. The secondary loader reads a Sigma load module and writes the 00 protection-type core image data in the format which is loaded by the PDP-11 absolute loader. So that the program need not start on a multiple of 800 hex, the secondary loader skips all data until the first nonzero 16-bit word. Thus the first valid word in the MODCOMP program must be nonzero. A special one-card bootstrap loader has been written for the MODCOMP which accepts programs in the PDP-11 absolute binary format from the twin-coax intercomputer communication links (Ref. 2).

II. MODC2 Language Definitions

The source language is defined by the lexical analyzer and directives of the METASYMBOL assembler and by the procedure definitions of SYSTEM MODC2. The METASYMBOL reference manual (Ref. 3) contains a complete description of the structure imposed upon character strings, symbols, expressions, and statements that are to be processed by this assembler, and also describes the data definition facilities and conditional assembly features of this assembler.

Character strings in the Sigma 5 are intrinsically EBCDIC, while strings are represented internally as ASCII in the MODCOMP-II. Thus, the Sigma's text-string generating directives, while available to the MODCOMP programmer, are relatively useless. A procedure "ANSCI" is provided which can convert short strings of one to four characters into ASCII character codes for use on the MODCOMP. An example of its use is given later.

Symbols may consist of 1–63 alphameric characters, not containing embedded blanks. All characters in a symbol are significant. An extended alphabetic character set includes the characters $, @, #, :, and __ (underscore). The special symbols $, and $$ represent the values of the location counters and must be given an intrinsic two-byte resolution by the directive

```
ORG,2      x'sss'
```

which both establishes address resolution, and sets the starting address of the program to the hexadecimal value 'sss'.

Statements consist typically of four fields known as Label, Command, Address, and Comment. Statements are free-form, and each field is terminated by a blank, or end-of-line, although semirigid field definition is to be preferred for readability. The Label field is optional, and may be eliminated by beginning any statement with a blank. A statement is a comment-only if it begins with an asterisk. The Label, Command, or Address fields may consist of one or more subfields separated by commas. The first subfield of the command field must invoke a directive, a procedure from SYSTEM MODC2, or a user-defined procedure. Any subfield within the address field may be preceded by an asterisk. This feature is used within SYSTEM MODC2 to invoke indirect addressing where appropriate. Multiple label subfields are allowed, and may be used in a user-defined procedure to label different lines within a procedure which generates multiple code lines, although there is no use of this feature within SYSTEM MODC2.

The MODCOMP-II instruction set as described in the Computer Reference Manual (Ref. 4) is available, with the exception of the floating point arithmetic set. These can be added at a later date, along with procedures to define floating constants in the MODCOMP format. The instructions are grouped into eight distinct classes, distinguished by the instruction addressing mode, and the format of the executable instruction in memory.

Class 1 is the immediate mode instructions which have the format

```
CMD,R1     value
```

The instruction 'LDI' is an example from this class which, when executed, will place "value" into register R1. Value may be a number, a symbol, or an expression which is evaluated by the METASYMBOL assembler.

Class 2 is the register-to-register mode instructions which have the format

CMD,R1 R2

The instruction 'LDX' is an example from this class, which when executed, will place the contents of the memory cell whose address is in register R2 into register R1. This class also contains the bit-manipulating instructions and the register I-O instructions.

Class 3 is the memory-to-register instructions which have the format

CMD,R1 [*]LOCATION[,XR]

Brackets denote that their contents are optional. The asterisk, if present, invokes indirect addressing, and the ",XR", if present, invokes indexing by the contents of register XR. The instruction 'LDM' is an example from this class which, when executed, will place the value contained in the memory cell addressed by the instruction into the register R1. Determination of the effective address of instructions under the optional indexing and/or indirect addressing is specified in the Computer Reference Manual (Ref. 4).

Class 4 is the register-to-register test instructions with conditional branching. They have the format

CMD,R1 R2,B-LOC

These instructions execute as their class 2 counterparts except that a branch to B-LOC is performed if the instruction's test conditions are satisfied.

Class 5 is the memory-to-register test instructions with conditional branching. They have the format

CMD,R1 [*]LOC[,XR],B-LOC

These instructions execute as their class 3 counterparts except that a branch to B-LOC is performed if the test conditions are satisfied.

Class 6 is the register-to-register comparison instructions which have the format

CMD,R1 R2,B-LOC1,B-LOC2

Class 7 is the register-to-memory comparison instructions which have the format

CMD,R1 [*]LOC[,XR],B-LOC1,B-LOC2

The address structure of class 6 and 7 corresponds to that of class 2 and 3, respectively, with the addition of the conditional branch location words.

Class 8 is a varied collection of two-byte instructions with the format

CMD value

where "value" may be missing for some specific instructions.

A summary of MODCOMP-II instructions may be found in Appendix E of the Computer Reference Manual (Ref. 4).

III. The Secondary Loader

The secondary loader reformats Sigma 5 load modules into the format required by the minicomputer's absolute binary loader. Input to the secondary loader is through the M:EI DCB; output is through the M:PO DCB. The input load module must be on a disk file, and output is typically to paper tape, or to cards. The M:EI and M:PO DCBs must be assigned to the appropriate files or devices before execution of the secondary loader. The control information expected by the absolute loader at the beginning and end of each physical record is supplied by the secondary loader.

The present secondary loader was initially written for the PDP-11. Data punched by this loader are organized into 16-bit (two-byte) words, with the least significant byte first. The following three control words precede the data to be loaded: a word containing the value '01', a word containing the total byte count in the physical record, and a word containing the starting location for loading the data from the record. As noted above, a special bootstrap loader has been written to load data in this format into the MODCOMP memory from an intercomputer communications link. Should the need arise, a secondary loader which punches data compatible with the MODCOMP-II loaders could be developed.

The length of a core-image segment in a Sigma load module is a multiple of a Sigma double word (64 bits), so the last byte of a program created through MODC2 will load on a MODCOMP-II byte address which is 1 less than a multiple of 8. This means that several bytes of zeros may follow the actual program.

IV. Assembly Example

Figure 2 is an example of a job-step sequence for the assembly of a simple nonsense program for the MODCOMP-II. At least one instruction from each class is included to illustrate the addressing and listing structure which results. A listing line consists of the input source-line number, the (4-byte) word address at which the instruction begins, and (if nonzero) a byte-offset from that word boundary. This is followed by a hexadecimal copy of the generated instruction and a copy of the input source line. Some source lines may result in more than one line of listing to accommodate the generated instruction(s), as for example, lines 16 or 37. The eight instruction classes are each indicated by a "*n" comment line preceding the appearance of that instruction class, as on line 9, which precedes the class 3 instruction LDM. Data are defined as 2-byte units by "DATA,2" as on line 38. Generation of ASCII Text is illustrated on lines 41 and 42.

The job-control-language shown produces a paper tape copy of the program. Cards would be produced if the explicit ASSIGNment of M:PO were deleted, and the default assignment used instead.

The SYSTEM MODC2 has been used successfully to generate programs totalling many hundreds of lines of code for the MODCOMP-II. These programs have been loaded into the MODCOMP via the direct link and successfully executed. While the SYSTEM MODC2 cannot be guaranteed to be correct, it at present contains no known problems.

References

1. Layland, J. W., Klimasauskas, C. C., and Ericksen, D. E., "An Introduction to Minicomputer Software Support," "The X930 Program Set for Sigma 5 Assembly," and "The SAPDP Program Set for Sigma 5 Assembly," *The DSN Progress Report*, Technical Report 32-1526, Vol. VII, pp. 84–96, Jet Propulsion Laboratory, Pasadena, Calif., Feb. 15, 1972.
2. Lushbaugh, W. A., "A Driver/Receiver Unit for an Intercomputer Communications Link," *The DSN Progress Report*, Technical Report 32-1526, Vol. XV, pp. 109–115, Jet Propulsion Laboratory, Pasadena, Calif., June 15, 1973.
3. "Xerox Data Systems Sigma 5-9 METASYMBOL Reference Manual," Publication 90-09-52F, Xerox Data Systems, Sept. 1972.
4. "MODCOMP-II Computer Reference Manual," 210-102000-000, Modular Computer Systems, May 1974.

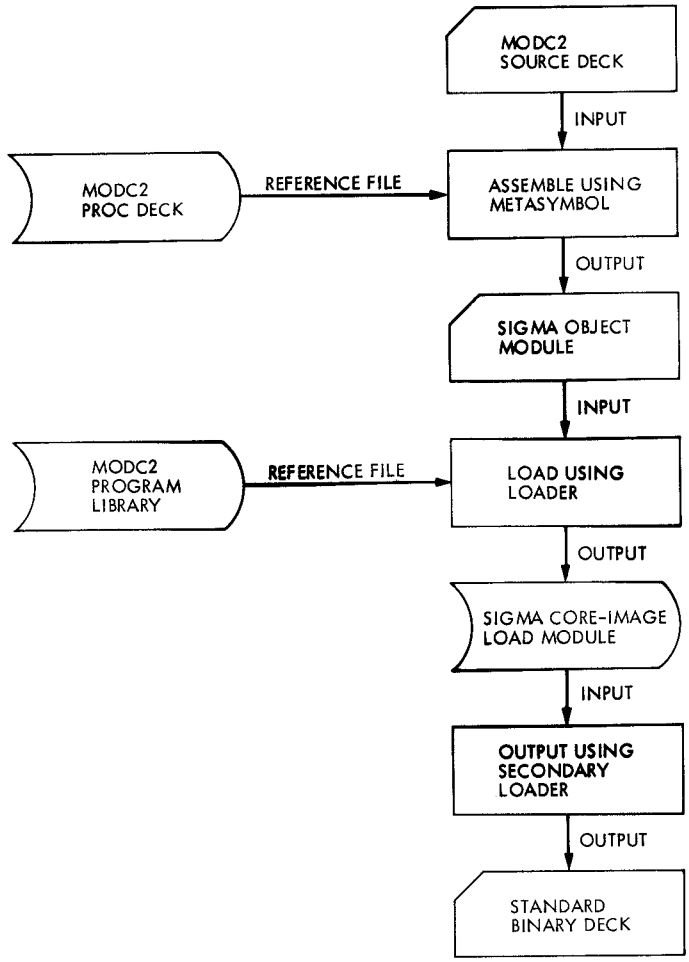

Fig. 1. Operational flowchart for the MODC2 program package

```
 14:05 JUN 17,'76 ID=0006=F00
! JOB MODC2,LYL
! ASSIGN M:BO,(FILE,TESTB)
!METASYM SI,LO,BO,AC(LYL)

 HOU    14:05  JUN 17, '76                                                                          1
         1                                              THIS IS A COMMENT
         2    01 000A0                                  ORG,2    X'140'             SET START LOCATION
              01 000A0
         3                                              SYSTEM   MODC2
         4                                              DEF      BEGIN
         5                                        *1
         6    01 000A0     ED200162 N             BEGIN  LDI,2    TABLE              SET REG#2 TO ADDRESS TABLE
         7                                        *2
         8    01 000A1     FD32     A                    LDX,3    2                  FIRST WORD OF TABLE
         9                                        *3
        10    01 000A1 2   E5480162 N                    LDM,4    *TABLE             LOAD THE WORD POINTED TO BY TABLE(1)
        11    01 000A2 2   E5530162 N                    LDM,5    TABLE,3            LOAD THE WORD IN TABLE(TABLE(1))
        12                                        *4
        13    01 000A3 2   7D65014A N                    THRB,6   5,JUMP1            COPY REG#5 TO REG#6, BRANCH IF NONZER
        14    01 000A4 2   0000     A                    HLT
        15                                        *5
        16    01 000A5     C4400161 N             JUMP1  ADMB,4   VALUE,JUMP2        ADD (REG#4) TO VALUE, BRANCH IF
                           0152
        17    01 000A6 2   C4420001 N                    ADMB,4   ONE,2,JUMP2        ADD (REG#4) TO (REG#2)+ONE, BRANCH IF
                           0152
        18    01 000A8     E700015D N                    BRU      EQU1
        19                                        *6
        20    01 000A9     DF320157 N             JUMP2  CRXB,3   2,EQUAL,LESS:THAN  SHOULD ALWAYS TEST EQUAL
                           0160
        21    01 000AA 2   E7000160 N                    BRU      GREATER:THAN
        22                                        *7
        23    01 000AB 2   87F80162 N             EQUAL  COMB,15 *TABLE,3,EQU1,LESS:THAN
                           015D0160
        24    01 000AD 2   E7000160 N                    BRU    GREATER:THAN
        25                                        *8
        26    01 000AE 2   6600     A             EQU1   NOP
        27    01 000AF     2601     A                    SIA    1                      SIMULATE PARITY TRAP
        28    01 000AF 2   0000     A                    HLT                           AND HALT PROCESSING
        29          01 000B0                      LESS:THAN EQU HA($)
        30          01 000B0                      GREATER:THAN EQU HA($)
        31    01 000B0     0000     A                    HLT
        32                                        *

 HOU    14:05  JUN 17, '76                                                                          2
        33                                        PZE    COM,16   0                  DEFINE PUT-ZERO COMMAND
        34          00000001                      ONE    EQU      1
        35                                        *
        36    01 000B0 2   0000     A             VALUE  PZE
        37    01 000B1     000A     A             TABLE  DATA,2   10,TABLE,BEGIN,0
              01 000B1 2   0162     N
              01 000B2     0140     N
              01 000B2 2   0000     A
        38    01 000B3                                   DO1    10
        39    01 000B3     00C1     A                    DATA,2       C'A'           TEN EBCDIC A'S
              01 000B3 2   00C1     A
              01 000B4     00C1     A
              01 000B4 2   00C1     A
              01 000B5     00C1     A
              01 000B5 2   00C1     A
              01 000B6     00C1     A
              01 000B6 2   00C1     A
              01 000B7     00C1     A
              01 000B7 2   00C1     A
        40                                        *
        41    01 000B8     41       A                    ANSCI,4    'ABCD'             AND SOME ANSCII TEXT
              01 000B8 1   42       A
              01 000B8 2   43       A
              01 000B8 3   44       A
        42    01 000B9     5A       A                    ANSCI,1    'Z'
        43                                               DEF      TABLE
        44          01 000A0                             END      BEGIN

   CONTROL SECTION SUMMARY: 01 000B9 1 PT 0
```

Fig. 2. Example assembly

```
HOU   14:05  JUN 17, '76                                                                                          3
 *    SYMBOL VALUES
          ANS::CII/LIST             EQUAL/01 000AB            EQU1/01 000AE            GREATER:THAN/01 00080
          I1/00000001               I2/00000001               J/00000000               JUMP1/01 000A5
          JUMP2/01 000A9            J1/00000029               LESS:THAN/01 00080       LO/LIST
          ONE/00000001              VALUE/01 00080
 *    EXTERNAL DEFINITIONS
          BEGIN/01 000A0            TABLE/01 00081
 * NO PRIMARY REFERENCES
 * NO SECONDARY REFERENCES
 * NO UNDEFINED SYMBOLS
 *    ERROR SEVERITY LEVEL: 0
 * NO ERROR LINES

!LOAD (BIAS,0),(NOSYSLIB),(NOTCB),(ABS),(MAP),(EF,(TESTB)),(LMN,TESTL)


* * ALLOCATION SUMMARY * *

  PROTECTION      LOCATION   PAGES

DATA (00)                      1
PROCEDURE (01)      200        1


  UDEF      A0 0   BEGIN
  UDEF      B1 0   TABLE
  CSEC         0
                   DATA
            BA     SIZE
           200     PROCEDURE
            16     SIZE

!ASSIGN M:EI,(FILE,TESTL)
!ASSIGN M:PO,(DEVICE,PPA01)
!RUN (LMN,SLOAD,:DSN)

!FIN
```

Fig. 2. (contd)