The LEAPSIG Sigma 5-Mac 16 Cross-Assembler

H. C. Wilck
Communications Systems Research Section

A cross-assembler, called LEAPSIG, has been developed to permit the Sigma 5 computer to assemble programs for the Mac 16 minicomputer. It was obtained by translating the Mac 16 assembler into a Sigma 5 program by means of Sigma 5 METASYMBOL "procedures." This article describes the LEAPSIG program and discusses the method by which it was generated. Information for using LEAPSIG on the Sigma 5 is also given.

I. Introduction

LEAPSIG runs on the Sigma 5 computer and is used to assemble software for the Mac 16 minicomputer. This cross-assembler accepts standard LEAP 8 (Lockheed Mac 16 assembler) source language as input and produces the same object code and listing as LEAP 8. LEAPSIG is a two-pass assembler with macro capability.

The need for this cross-assembler stems from fact that the Mac 16 minicomputers, used by JPL in control applications, lack the peripherals, particularly a fast line printer and card reader, necessary for efficient program preparation. Since the assembly process is typically input/output (I/O) limited, shifting this task from the minicomputer to a larger machine with a good set of peripherals can cut assembly time by a factor of ten or more. A further discussion of minicomputer software support can be found in an article by J. W. Layland (Ref. 1). The XDS Sigma 5 was chosen as a host computer because of its availability at JPL and because it has a very flexible assembler (METASYMBOL) which made it possible to build the LEAPSIG program in an efficient manner.

This article primarily discusses the development of the LEAPSIG program. In addition, operating instructions for LEAPSIG are provided in Section IV. Some familiarity with the Sigma 5 computer, METASYMBOL, the Mac 16 minicomputer, and LEAP 8 on part of the reader is assumed. The respective manufacturers' manuals are recommended as a source of additional information (Refs. 2, 3, 4, and 5).

II. The Method Used to Implement LEAPSIG

The most direct method to obtain a cross-assembler on the Sigma 5 would be to write it completely in Sigma 5 assembly language. This approach demands a large amount of programmer time.

An alternative technique that requires less programming work is to develop a set of procedures (macro definitions) which allow Sigma 5 METASYMBOL to assemble programs written in a modified version of the minicomputer assembly language. A second program must also be provided to translate the resultant Sigma 5 load modules into

the format required by the minicomputer loader. This method has been used by C. C. Klimasauskas and D. E. Erickson to assemble programs for the XDS 930 and the PDP 11 on the Sigma 5 (Refs. 6 and 7). This process requires an input source language compatible in syntax and format with METASYMBOL. Since, in general, minicomputer languages do not meet this requirement, each minicomputer program to be assembled by this method must be written in a special METASYMBOL-compatible language.

A different approach, the translation of the Mac 16 LEAP 8 assembler into a Sigma 5 program, has been chosen for the implementation of LEAPSIG. This translation was accomplished in several steps. First the LEAP 8 assembler source program, written in a subset of the LEAP 8 language itself, was modified to make it syntactically compatible with METASYMBOL. Next a set of procedures was written to define the operation codes contained in the LEAP 8 source to the METASYMBOL assembler. Then the METASYMBOL assembler was used to translate the LEAP 8 assembler source program according to those procedures into a Sigma 5 machine language program. This program, augmented by a set of short I/O routines written in METASYMBOL, constitutes the LEAPSIG cross-assembler.

It is important to note that only the LEAP 8 assembler itself is translated by METASYMBOL procedures. The result of this translation is LEAPSIG. Assembly of other Mac 16 programs by means of LEAPSIG is not done by these procedures. LEAPSIG, in its object form, is a Sigma 5 machine language program. Language compatibility problems are therefore confined to the development of LEAPSIG and do not affect its use.

LEAPSIG is the functional equivalent of LEAP 8. LEAPSIG accepts standard LEAP 8 source language and outputs the same object and listing when run on a Sigma 5 as does LEAP 8 operating in a Mac 16. Any Mac 16 program can be assembled either by LEAPSIG on the Sigma 5 or by LEAP 8 on the Mac 16 with identical results.

III. The Translation Process

Some aspects of the translation process used to generate LEAPSIG are discussed in greater detail below.

A. Preprocessing

The LEAP 8 assembler is written in a subset of the LEAP 8 language not entirely compatible with METASYMBOL. Therefore, the LEAP 8 source program had to be preprocessed to make it suitable for translation by the METASYMBOL assembler. The LEAP 8 source program was inspected, analyzed, and modified with the aid of a set of FORTRAN routines written for this purpose. These routines take advantage of the fact that the LEAP 8 is written in fixed field form. It was found that only 60% of the LEAP 8 instructions and only half of the directives were used in the LEAP 8 source and that there were no instances of macros, logical operators, Boolean operators, or floating point constants. LEAP 8 almost agrees with METASYMBOL in the definition of source statement fields and subfields, symbols, operators, and expressions, at least to the extent of their actual presence in the LEAP 8 source. Usage of * for indirection and comment and usage of = for literals is the same in both languages. Furthermore, the only three cases of mnemonics common to both languages, NOP, EQU, and END, also have equivalent definitions in LEAP 8 and METASYMBOL. All of this simplifies translation and contributes significantly to the feasibility of this technique of cross-assembler building. However, there were some compatibility problems that had to be resolved by modifying the LEAP 8 source, as outlined below:

(1) *Hexadecimal constants.* The symbol $ followed by a string of hexadecimal digits (the LEAP 8 representation for hexadecimal constants) had to be changed to a string of hexadecimal digits surrounded by quotation marks and preceded by X.

(2) *Binary scaling.* All constants with binary scaling were interpreted and replaced by their unscaled equivalents.

(3) *Character strings.* It was necessary to change all character strings into hexadecimal constants, according to the USASCII code, since leaving their interpretation to METASYMBOL would lead to EBCDIC representation, which is incorrect for LEAPSIG.

(4) *Location counter references.* The symbol $ had to be substituted for * where used as reference to the location counter.

(5) *Assembler directives.* A number of LEAP 8 directives for which METASYMBOL procedures cannot be written had to be handled by source editing. EJECT was replaced by PAGE. The conditional assembly directives SKIPT and SKIPF were interpreted and then deleted together with the skipped source lines. EXTRN, LSTSY, and TITLE were expunged because their use was found unnecessary.

(6) *Storing into instructions at run time.* This practice causes problems if a Mac 16 instruction being

changed at run time translates into more than one Sigma 5 instruction or into an instruction that does not allow the type of modification attempted. Although any instruction can be modified or replaced at execution time, some are more likely targets than others. Eight instances of run time instruction changing were found in the LEAP 8 program by scanning its source for labeled NOPs and instructions of the immediate type with zero arguments. Analysis showed that seven of these cases would still work after translation, and the remaining one required rewriting a segment (12 source lines) of the LEAP 8 program.

B. The LEAP 8 Procedure Set

The LEAP 8 procedure set, which consists of 500 statements, allows the METASYMBOL assembler to translate each LEAP 8 operation code into one or more Sigma 5 instructions. Writing these procedures was greatly eased by the fact that only those LEAP 8 operations and pseudo-operations that actually occur in the LEAP 8 assembler source program needed to be defined.

Two Sigma 5 registers were set aside for representing the Mac 16 accumulator and index register. Another two Sigma registers were assigned to the Mac 16 carry and overflow indicators. (The other four Mac 16 status indicators are never used in LEAP 8.)

Since Mac 16 is a 16-bit machine while the Sigma 5 word is 32 bits long, the procedures for data generating directives store data words into the lower halfword with the sign extended 16 bits to the left. Because of this difference in word length, carry and overflow do not occur in the same manner in both machines. Therefore, procedures for arithmetic instructions must make provision for detecting and saving Mac 16 carry and overflow and for extending the sign of the result of the operation.

To avoid address arithmetic problems all Mac 16 instructions that resulted in more than one Sigma 5 instruction were translated into a branch to a separate auxiliary program section where the actual translation was stored, followed by a branch back to the next location in the main section. The METASYMBOL directives CSECT and USECT allow easy switching between two assembly sections.

Figure 1 shows a procedure for a number of Mac 16 instructions, each of which translates into one Sigma 5 instruction.

The procedure in Fig. 2 is an example of a Mac instruction (STL) expanding into several Sigma instructions. Line 109 shows the branch to the auxiliary section. The branch instruction itself is assembled into the main section. Line 110 establishes an address for the branch back from the auxiliary section. Line 111 switches assembly to the auxiliary section. Lines 112 through 116 contain the actual translation of the STL instruction. Line 117 is the branch back to the main section. Line 118 sets the address for the next branch to the auxiliary section and line 119 returns assembly to the main section. XTWO and XONE on lines 114 and 115 refer to two index registers that contain a 2 and a 1 respectively.

Excluding the symbol table and the I/O routines, LEAP 8 occupies roughly 4000 words of Mac 16 storage. Translation expands it into approximately 6000 Sigma 5 words.

C. I/O Routines

Five Sigma 5 routines, comprising a total of 200 source statements, have been written in METASYMBOL to enable LEAPSIG to communicate with the outside world. These routines replace the I/O programs used by LEAP 8 when running on the Mac 16.

During pass 1 of the LEAPSIG assembler the *source input* routine reads the source from the input device (usually the card reader) and translates it from EBCDIC (the character code used by the Sigma 5) to USASCII (the character code required by LEAPSIG). The translated source is also saved on the RAD and later retrieved from there for use by the second pass.

The *list output* routine translates the assembly listing from USASCII to EBCDIC and outputs it on the listing device.

The *object output* subprogram punches the Mac 16 binary object on the paper tape punch.

The *device ready* routine performs a check for I/O completion. This is necessary in order to take advantage of the LEAP 8 I/O buffer switching feature, designed to speed execution.

The *executive* routine reads and interprets control messages that govern the execution of LEAPSIG. (See Section IV below.)

IV. Using LEAPSIG on the Sigma 5

The I/O routines used by this assembler require assignments for the following logical devices:

F:CTL Control input device (usually assigned to the card reader).

F:CRD Source input device (usually assigned to the card reader).

F:PRT Listing output (usually assigned to the line printer).

F:PCH Object output (usually assigned to the paper tape punch).

F:RAD Scratch file (usually assigned to the RAD).

The execution of the LEAPSIG program is controlled by special control cards. These cards must contain a / in column 1 followed in column 2 by 0, 1, 2, 3, 4 or /. Columns 3 through 80 are ignored. The meaning of these control cards is explained below:

/0 Execute pass 1 (similar to the Mac 16 control message 'EX,,400').

/1 Execute pass 2, punch object and print listing (similar to 'EX,,401').

/2 Execute pass 2, print listing (similar to 'EX,,402').

/3 Execute pass 2, punch object (similar to 'EX,,403').

/4 Execute pass 2, print errors only (similar to 'EX,,404').

// Exit to Monitor.

If F:CTL and F:CRD are assigned to the same device (e.g., the card reader) the control messages must be contained in the source input stream. The assembler looks for control messages when it starts execution and after finishing each pass.

The typical deck shown below will produce an assembly with listing and object paper tape. It is assumed here that the LEAPSIG program is available from the RAD.

```
!JOB ACCOUNT,NAME
!ASSIGN F:CTL, (DEVICE CRA03)
!ASSIGN F:CRD, (DEVICE CRA03)
!ASSIGN F:PRT, (DEVICE,LPB02), (VFC)
!ASSIGN F:PCH, (DEVICE PPA01), (BIN)
!ASSIGN F:RAD, (FILE,SCRATCH), (OUTIN)
!RUN (LMN,LEAP,ACCOUNT)
!DATA
/0
(Source deck in LEAP 8 language)
/1
//
!FIN
```

V. Conclusion

The LEAPSIG cross-assembler has been extensively used in the preparation and documentation of the Programmed Oscillator software.

The relatively small programming effort required for generating LEAPSIG makes it worthwhile to investigate the applicability of the techniques described here to other cross-assemblers. The feasibility of this approach is primarily governed by the amount of preprocessing involved and by the expansion of assembler core size caused by the translation.

References

1. Layland, J. W., "An Introduction to Minicomputer Software Support," in *The Deep Space Network Progress Report*, Technical Report 32-1526, Vol. VII, pp. 84–85. Jet Propulsion Laboratory, Pasadena, Calif., Feb. 1972.
2. *XDS Sigma 5 Computer Reference Manual*, 90 09 59D. Xerox Data Systems, El Segundo, Calif., Feb. 1970.
3. *SYMBOL/META-SYMBOL Reference Manual for Sigma 5/7 Computers*, 90 09 52C. Xerox Data Systems, El Segundo, Calif., Dec. 1969.
4. *Mac 16 Computer Reference Manual, TM13010009800.* Lockheed Electronics Company, Los Angeles, Calif., Nov. 1970.
5. *Mac 16 LEAP Assembler Manual, TM13013041101.* Lockheed Electronics Company, Los Angeles, Calif., Dec. 1970.
6. Klimasauskas, C. C., "The X930 Program Set for Sigma 5 Assembly," in *The Deep Space Network Progress Report*, Technical Report 32-1526, Vol. VII, pp. 86–90. Jet Propulsion Laboratory, Pasadena, Calif., Feb. 1972.
7. Erickson, D. E., "The SAPDP Program Set for Sigma 5 Assembly," in *The Deep Space Network Progress Report*, Technical Report 32-1526, Vol. VII, pp. 91–96. Jet Propulsion Laboratory, Pasadena, Calif., Feb. 1972.

```
53      *
54      JMP     CNAME   0,X'68'
55      JMM     CNAME   X,X'6A'
56      LDA     CNAME   A,X'32'
57      LDX     CNAME   X,X'32'
58      STA     CNAME   A,X'35'
59      STX     CNAME   X,X'35'
60      ANA     CNAME   A,X'4B'
61      ORA     CNAME   A,X'49'
62              PROC
63              BOUND   4
64      LF      GEN,1,7,4,3,17 AFA(1),NAME(2),NAME(1),AF(2),AF(1)
65              PEND
66      *
```

Fig. 1. Example of a procedure for one-to-one translation of Mac 16 instructions into Sigma 5 instructions

```
106     *
107     STL     CNAME
108             PROC
109     LF      B       BB
110     RR      SET     $
111             USECT   AUXF
112             BOUND   4
113             GEN,1,7,4,3,17 AFA(1),X'32',A1,AF(2),AF(1)
114             STB,A   A1,XTW0
115             LH,A1   A1,X0NE
116             GEN,1,7,4,3,17 AFA(1),X'35',A1,AF(2),AF(1)
117             B       RR
118     BB      SET     $
119             USECT   PROG
120             PEND
121     *
```

Fig. 2. Example of a procedure that expands one Mac 16 instruction into several Sigma 5 instructions