516-45
ADH, CC
12/30/70

516/316 Assembler and Binder

A DDP-516 segmented program (assembler) has been written to assemble relocatable (unsegmented or "system") programs for the DDP-516 and the DDP-316. A companion program (binder) has been written to bind several such relocatable programs into an absolute binary program, including the linking of symbolic locations in separate programs and the desectoring of addresses which cross sector boundaries. Hence, these programs provide approximately the same facility as the GE assembler and post-processor described in 516-16. However, the GE assembler provides some special MACRO's for simplifying assembling of the 516 multi-programming system, but the 516 assembler does not contain such MACRO's.

This assembler was derived from the segment assembler described in 516-41. In particular, the free-field symbolic input format is compatible with the segment assembler, but not with GMAP. However, CVGEAF (516-41-6) may be used to effect a partial conversion from GMAP format into the appropriate GE format.

Calling Sequence

The assembler is called from the Executive by the line

```
H16ASM,file
```

where "file" is the name of the symbolic input file. Similarly, the binder is called by the line

```
H16BND,file
```

where "file" is the name of a symbolic control file. The assembler and binder each produce a binary file, named "file*B", and a listing file, named "file*L". Previous data in these files is overwritten; a missing output file is automatically created. The binder binary output file is compatible with the absolute loader. Thus, this file may be sent to the GE, punched on cards, and the cards loaded through the card reader. One simple way to send the cards to the GE for punching is to call

```
GE*L*B,file
```

where "file" is the binder symbolic control file. This procedure will also list file*L for the convenience of the user.

Assembler Symbolic Format

The source program is assumed to begin at the start of the source file and end with an END card (card=line). Each symbolic line has the following format:

label_1:...label_n: op variable field\comments

where blanks may be placed wherever desired. The label field may be omitted or may contain any number of labels; each label is followed by a colon. The operation code is normally terminated by a blank, but this is optional if the next character is a break (e.g., comma, left parenthesis, left slant). The blank operation code is null; hence, blank cards are acceptable, as are cards with only a comment (preceded by left slant). The variable field is terminated (except for ASCII) by a left slant or an end-of-line. Comments may be placed between a left slant and an end-of-line.

Symbols

Symbols are composed of 1-6 contiguous characters which may be alphabetic (lower case = upper case), the ten digits and period. Two special symbols are null (absolute value 0) and * (relocatable value = current program counter).

A defined symbol has a 16-bit value; hence, it is possible to imbed indirect and/or index bits in an address symbol. When a symbol value is required, the evaluator first attempts to evaluate the symbol as an integer in the appropriate mode (decimal, octal, or hexadecimal, depending on context). If this fails, the symbol is looked up in the symbol table.

There are three symbol types (other than undefined): absolute, relocatable, and external (external symbols are implicitly relocatable). The type is implicit in the definition, as described later.

Expressions

An expression is a string of symbols separated by the operators + or -. If an expression begins or ends with an operator, there is assumed to be a leading or trailing null symbol, respectively. Symbols in an expression must be absolute or relocatable (not external); the value of an expression must be absolute or relocatable.

Labels

The use of a symbol as a label causes the symbol to be defined as relocatable, with value equal to the current location counter contents. Labels may not be redefined; moreover, the definitions on the two passes must agree.

Machine Operations

The operation symbols are the same as for the other assemblers; the DAP symbols are used for all except input/output instructions. The DAP instructions OCP, INA, OTA, SNK, and SKS have not been implemented; each I/O instruction has been given its own symbol in the manner of the generic instructions. The Honeywell I/O instructions defined in this assembler are given below, for the mask register, clock, and ASR-33.

Mnemonic	Octal	Mnemonic	Octal
WTMR	170020	NBTT	070104
CKON	030020	NITT	070404
CKOF	030220	RATT	131004
NSCK	070020	RBTT	131204
ERTT	030004	WATT	170004
EWTT	030104	WBTT	170204
SRTT	070004	NSTT	070504

The special I/O instructions for Dept. 1383 hardware are described in 516-2 and 316-5, and the corresponding symbols have been defined in the assembler.

The variable field (if any) of a machine instruction is of the form given below:

```
     op     address,modifier
```

The address field may contain an expression. The address may be absolute, relocatable, or external for a memory-reference instruction or just absolute for a shift/rotate instruction. The modifier field may be used with a memory-reference instruction; if used, it must contain 1 (index), * (indirect), or 1* (index then indirect), except for LDX/STX in which case indexing is illegal.

The pseudo-instruction ADDR has been defined to permit assembling indirect (full-word) addresses. The variable field is the same as for a memory-reference instruction.

External Symbols

The binder permits symbolic inter-program linkage. The pseudo-operations SYMDEF and SYMREF have been borrowed from GMAP to direct such linkage. The variable field of either operation may contain a list of symbols, separated by commas. SYMDEF symbols are relocatable symbols defined within the current program; their names and definitions are made available for use by other programs. SYMREF symbols are used in the current program but defined in some other program. As noted above, SYMREF symbols may not be elements of expressions.

Symbol Definition

In addition to their use as labels, symbols may be defined or redefined by means of the pseudo-operations SET, BOOL, and SETX. The symbol to be (re)defined is given as the first argument, followed by a comma; the second field is a symbol or an expression to which the first argument is equated. Any symbols in the second argument must have been previously defined. All types are legal: absolute, relocatable, or external. The particular operation governs the interpretation of integers in the expression: SET-decimal, BOOL-octal, SETX-hexadecimal. As implied above, redefinition of SET symbols is legal, unlike the rule for labels.

Operation Definition

A new operation symbol (mnemonic) may be equated to an existing one by means of the OPSYN pseudo-operation. The first argument (followed by comma) is the new symbol; the second argument is the existing operation symbol. As with SET, operation redefinition is legal.

A brand new operation symbol may be defined by means of OPDO or OPDX. The variable field has three arguments, separated by commas The first argument is the operation symbol (mnemonic). The second argument is the operation value. The third argument is the operat type. The value and type may be expressions in which all symbols have been previously defined; integers are interpreted in octal for OPDO or hexadecimal for OPDX. Without going into details, the recipe for constructing new generic instructions (including I/O instructions) is to give the operation definition as the value and set the type to one (1). If a consecutive list is given, the type need only be specified on the first definition.

Data Generation

The pseudo-operations DEC, OCT, and HEX may be used to generate integer data words coded in decimal, octal, or hexadecimal, respectively. One data word is generated per argument; arguments are separated by commas. The integers may be signed; negative is interpreted as two's complement in all cases. (In the current implementation, each argument may actually be an expression.)

The ASCII pseudo-operation is used for generating text. The argument begins with a left parenthesis and ends with a right parenthesis. Blanks within the argument are taken as they stand. The odd (first, third, etc.) characters go into the most significant half of sequential words; the even (second, fourth, etc.) characters go into the least significant half of these words. If the number of characters is odd, the least significant half of the last word contains rubout (377_8). The left slant "\" is converted to a null (0). A multi-line argument is legal; each line ends with carriage return <u>and</u> line feed.

The DCHAR and OCHAR pseudo-operations may be used to generate characters which ASCII cannot handle, such as \ and). The variable field has two arguments, separated by a comma. The first argument goes into the most significant half of the word; the second argument goes into the least significant half of the word. Either argument may be an expression in which all symbols have been defined; the expression value must be absolute. Integers are interpreted as decimal in DCHAR or as octal in OCHAR.

Storage Allocation

One storage allocation pseudo-operation, BSS, is available for reserving blocks of storage. The location counter is advanced by the value of the argument. The argument may be an expression; its value must be absolute and non-negative. All symbols in the expression must have been previously defined. Integers are interpreted as decimal.

Errors

Errors detected during pass 2 are typed to the user as a one-character flag followed by the number (decimal) of the line in which the error occurred. The approximate meanings of the flags are as follows:

F	End of file (no END card)
U	Undefined symbol or operator
A	Address error, including relocation error or modifier field error
M	Multiple symbol definition - the new definition replaces the old one
O	Operation code undefined - treated as null

Assembler List File

The listing (*L) file is a copy of the source file in which the value of the program counter is inserted in front of each line before the line is processed. The listing may be printed at the GE-635 by use of GELIST (give the name of the list file as an argument).

Assembler Binary File

The binary (*B) file contains the relocatable data which is ready for the binder. The user need not be concerned with the format of this file; however, he may obtain a deck as back-up by using GEBKUP.

For the record, the binary file format is as follows. The first character of the file is 21_8, which is used by the binder as a check that the file contains legitimate data. The remainder of the file consists of triples of characters (24 bits) which are arranged as follows:

16	6	2
Address	Subtype	Type

If type $\neq 0$, the Address field is treated as :

Type	Address
1	Absolute
2	Relocatable
3	External (index into SYMREF list)

If the data represents a memory-reference instruction, for which desectoring is required, then the subtype contains the upper 6 bits (operation field) of the instruction. The index and indirect bits are always assembled into the top two bits of the address field (except for LDX/STX, where the index bit is part of the op code), even in the case of external addresses (just the lower 14 bits form the index in this case). If subtype = 0, then the address is the data, and no desectoring is performed.

If type = 0, then the subtype is interpreted as follows:

Subtype	Address
0	Unused (illegal)
1	(Relocatable) loading address
2	SYMDEF definition
3	SYMREF index

For SYMDEF and SYMREF, the following six characters give the associated ASCII symbol, left-adjusted, with trailing blanks. It is assumed that the first loading address will be explicity set by the assembler, even if it is 0. Also, the loading address in effect at the end of the file will be assumed to establish the size of the program.

Binder Control File

The binder control file is a symbolic file which tells the binder which programs to load where. Its format is rigid; each line consists of a control character followed by a single blank and an argument. If the argument is followed by a blank, a comment may conclude the line. There are three legal control characters:

L	file	load file
O	origin	set origin
S	sa	starting address

The file name on the L (load) card is the exact name of the (relocatable binary) file to be loaded at the next available location; in general, this name will end with *B. The origin and starting address arguments are unsigned octal integers; the initial (default) value in both cases is 1000_8.

The control file simply ends at the end-of-file.

Desectoring

When a sector-0 address must be generated to desector a memory-reference-instruction address, it is inserted in a pool which grows downward (toward lower addresses) from location 777_8. The pool is scanned so that two identical desectoring addresses will always use the same word. However, management of the desectoring pool is independent of data loaded into sector 0 by the user. In fact, the binder will not detect an overlap between user data and the desector pool. More generally, the binder will not detect any data overlaps, as might occur if the user loads two programs into the same space.

Binder Errors

Binder errors are typed to the user as a flag character followed by a decimal line number and (when appropriate) a symbol. The line number is the line of the control file being processed at the time the error was encountered. The approximate flag meanings are as follows:

M	Multiple definition - two SYMDEF's for the same symbol
U	Undefined symbol - SYMREF with no matching SYMDEF
O	Overflow - too many SYMREF's in one program
E	Error in control file

A few other error comments are possible that are self-explanatory, such as FILE? for an unavailable control file, or a comment in case the desector pool overflows.

Binder List File

The list (*L) file contains a list of all SYMDEF symbols and their definitions. It also lists the UB or upper break, that is, the address at which a new program would be loaded in the absence of a new O (origin) setting, the DB or desector break, that is, the location into which the next desector address would be inserted, and the SA, or starting address which is put on the binary transfer card. The UB, DB, and SA are also automatically typed to the user at the conclusion of binding.

Binder Object File

The object (*B) file of absolute binary card images produced by the binder is compatible with the binary decks accepted by the DDP-516 and -316 loaders (see Binary and Patch loaders, 516-20). Columns 73-78 contain the name of the binder control file, and columns 79-80 contain sequence numbers. The deck is always concluded with a binary transfer card.