N.V.PHILIPS Natuurkundig Laboratorium

V.O.S.
NR : ALDEF
DD : 2024-07-18
PAR : JH

TWIN 2650 ASSEMBLER AND LINK EDITOR

1. Introduction

The TWIN Assembler ver 2.0 has certain limitations which present difficulties with larger software systems. The most important drawbacks are:

1. It is very complicated to link separately assembled programs and all links are to be updated manually after every change in one of these programs.

2. an assembled program can be executed only in the memory locations specified at assembly time.

   Suppose that consecutive memory locations are assigned to a number of programs. When, due to some change, a program in the beginning is enlarged, all next programs are to be re-assembled too.

3. the execution of an assembler run needs a large amount of time.

To avoid these drawbacks a new assembler, called "2650 Assembler", has been developed. The main differences with the original assembler are:

1. A relocatable output is produced, so the locations in which a program is to be executed can be determined after assembling;

2. It is possible to introduce itmes which are defined in other programs, so links between separately assembled programs can be established mechanically;

3. The assembly time is reduced with a factor 4 to 5 (an assembly run of 20 min. with the original assembler is reduced till 4-5 min. with the new assembler);

4. The mnemonic branch-instruction codes are extended by the inclusion of the condition in the mnemonic (for example, BER is equivalent to BCTR,0) which increases the readability of programs;

5. Default operands are possible (except for addresses), the default value is 0;

6. ASCII data constants and other types of data constants may be combined in a single statement;

7. Data definitions may be preceded by a multiplication factor;

8. The number of bytes in a character string (ASCII data constant) can be specified explicitely;

9. The assembler instructions PRT and PCH are deleted;

10. Finally, the new assembler accepts most correct version 2.0 programs (see for exceptions 4.5 "instruction ORG").

To link separately assembled programs a Link Editor has been constructed. The 2650 Assembler produces an object module. The Link Editor links a number of object modules and edits these modules into one or more load modules. These modules can be loaded by means of the system commands LOAD and XEQ.

2. Evaluation of expressions

Due to the relocation facility and the ability to refer to external symbols, several types of operands in expressions are to be distinguished. These types are:

1. Constant
   A constant is an internally defined item with an invariant value. Constants can be defined by the assembler instruction EQU, for example:

   ```
   CR   EQU   H'0D'
   ```

2. Address
   An address is also an internally defined item, but its final value is determined by the Link Editor when the program is edited. Labels of machine instructions are examples of addresses.

3. External reference
   An external reference is a reference to a item which is defined in another program. There it is either an address or a constant. External references are introduced by means of the instruction EXTRN.

All operations are allowed on constants, but only a limited number of operations are useful on addresses and external references. When an operation, not giving a usable result, is performed on an address or an external reference, the result is a constant. The operations, types of operands and the type of the result are given below.

operation	operands	result
addition:	A + C	A
	E + C	E
		C in all other cases
subtraction:	A - C	A
	E - C	E
		C in all other cases
any other operation:		C in all cases

In this table means:

A : address
E : external reference
C : constant

3. External Symbol Dictionary

In order to allow links between separately assembled programs, each object module comprises and External Symbol Dictionary (ESD). The ESD is printed at the beginning of the program listing. An ESD can contain the following types of records:

1. SD - Section definition
   The first record in each ESD is of the type SD. It contains the name of the section, its assembled origin and its length. The section name can be specified by the instruction CSECT (see below).

2. LD - Label definition
   An ESD can contain several records of the type LD. They identify symbols, associated to addresses and defined in the corresponding program, together with their values. Other (separately assembled) programs may refer to these symbols. The instruction ENTRY is responsible for the generation of the LD-records.

3. CD - Constant definitions
   A CD-record is similar to a LD-record, exept that the value is not an address but a constant. The CD-records are also generated by ENTRY-instructions.

4. ER - External reference
   The ER-records form the counterpart of the LD-records and the CD-records. They contain the symbols (with a sequence number) which are defined as "external" in the associated program. The instruction EXTRN is to be used for this purpose.

At link editing time a symbol-table is constructed. This symbol-table contains all the symbols from the SD-records, the LD-records and the CD-records. Each symbol may occur only once. During the editing of an object module the values corresponding to the "ER-symbols" are retrieved from this symbol-table and the values are added to the object module.

The following assembler instructions exist for the construction of an ESD:

1. CSECT
   The instruction CSECT can be used to define the name of an object module. Each source program may contain at most 1 CSECT instruction. Normally this instruction is placed at the beginning of the program. The instruction format is:

   ```
   <lbl> CSECT
   ```

   The label is optional, it denotes the name of the object module. The address of the assembled origin of the module is assigned to a present label. The instruction has no operands.

2. ENTRY
   The instruction ENTRY is to be used to specify the symbols which may be referred to in other programs. The format is:

   ```
           ENTRY   <sym>  [ , <sym> ]
   ```

   The instruction may not be labeled. The operand constists of a string of symbols; the symbols are separated by commas. These symbols must previously be defined as the label of some machine instruction or assembler instruction.

3. EXTRN
   The instruction EXTRN is to be used to specify the symbols which are defined in some

other program but to which is referred in this program. The format is:

```
        EXTRN   <sym>  [ , <sym> ]
```

The instruction may not be labeled. The operand consists of a string of symbols; the symbols are separated by commas. These symbols may not be defined as label of any instruction.

The only exception is given by EXTRN-statements which are generated by means of a macro-instruction. Then the attribute ER is ignored if the concerned symbol is also defined in another statement.

4. Extension of the instruction set

In order to symplify the writing of programs and to increase the readability of programs, the mnemonic branch-instruction codes are extended and the assembler instructions ACON and DATA are modified. Also two new assembler instructions are introduced, a listing control instruction and an instruction to copy source lines into the object module. Because the Link Editor produces load modules, the use of the instruction ORG is restricted. Finally, the PRT-instruction and the END-instruction are slightly changed.

4.1. Mnemonic branch-instructions

The additional branch-instructions are intended to be used after compare-instructions, load- and arithmetic instructions and the instruction TMI. These instructions with their equivalents are:

situation		extension	equivalent
after compare	branch if high	BHR	BCTR,1
		BHA	BCTA,1
	equal	BER	BCTR,0
		BEA	BCTA,0
	low	BLR	BCTR,2
		BLA	BCTA,2
	not high	BNHR	BCFR,1
		BNHA	BCFA,1
	not equal	BNER	BCFR,0
		BNEA	BCFA,0
	not low	BNLR	BCFR,2
		BNLA	BCFA,2
after load- and arithmetic operations	branch if positive	BPR	BCTR,1
		BPA	BCTA,1
	zero	BZR	BCTR,0
		BZA	BCTA,0
	minus	BMR	BCTR,2
		BMA	BCTA,2
	not positive	BNPR	BCFR,1
		BNPA	BCFA,1
	not zero	BNZR	BCFR,0
		BNZA	BCFA,0
	not minus	BNMR	BCFR,2
		BNMA	BCFA,2
after TMI	branch if ones	BOR	BCTR,0
		BOA	BCTA,0
	mixed	BMR	BCTR,2
		BMA	BCTA,2

Also the following unconditional branch-instructions exist:

extension	equivalent
BR	BCTR,3
BA	BCTA,3
BSR	BSTR,3
BSA	BSTA,3
RET	RETC,3

4.2. Instructions DATA and ACON

The data definitions in the instructions DATA and ACON may be preceded by a multiplication factor and the length of character strings can be defined explicitely.

Two types of data constants are distinguished: character strings and other constants. Both types may be specified in the same DATA instruction. A character string occupies a variable number of bytes, this number is either implicitely or explicitely specified. An other constant occupies always a single byte.

Each data item which is defined by means of the instruction ACON uses always 2 bytes. This instruction cannot be used to define character strings.

The syntax of the operands of the DATA- and ACON-instructions is:

```
<operand>       ::= <constant> [ , <constant> ]

<constant>      ::= <factor> <A-constant>
                  ! <factor> <expr-string>
                  ! expr

<factor>        ::= self defining constant
                  ! ( expr )
                  ! nill

<A-constant>    ::= A <length> chr-string

<expr-string>   ::= H ' H-expr [ , H-expr ] '
                  ! B ' B-expr [ , B-expr ] '
                  ! O ' O-expr [ , O-expr ] '
                  ! D ' D-expr [ , D-expr ] '

<length>        ::= L self defining constant
                  ! L ( expr )
                  ! nill
```

All operators are allowed in any type of expression. Also all types of operands are allowed in normal expressions (syntax element "expr"). Symbols which are used in normal expressions must previously been defined.

The operands used in other types of expressions are interpreted conform to the expression identification, e.g. the operands of a hexadecimal expression (element H-expr) are interpreted as hexadecimal digits.

The value of a multiplication factor (element "<factor>") may not be an address; the

allowed range for the value is 1 - 255. The value of an explicite length definition of a character string (element "<length>") must be a constant in the range 1 - 127.

The multiplication factor indicates the number of times the following constant is to be generated. If no factor is specified the value 1 is assumed. For example, the constant definition 2H'0,0' produces 4 bytes with the value 0. This definition is equivalent to the definition H'0,0,0,0'.

If the explicite length of a character string is greater than the length of the following string, the given data are left aligned and are padded with blanks, e.g. the constant AL4'ON' produces the character string 'ON '. If the explicite length is less than the following string, this string is truncated when the explicite length is exhausted, the constant AL2'DATA' results in the string 'DA'. The truncation is not signalled.

4.3. Instruction CEJE

The new listing control instruction is a conditional eject. The format is:

```
CEJE   <expr>
```

In which <expr> is a normal expression. This instruction is equivalent to a normal EJE-instruction if the remaining number of lines on the current page is less than the value of the expression. If the remaining number of lines is equal to or greater than the value of the expression, the instruction has no effects.

4.4. Instruction REPRO

Often it is desirable to apply certain link editing commands to the Link Editor. Such commands are not automatically supplied by the Assembler, but the Assembler has an instruction to copy such commands from the source module. This is the instruction REPRO. The format of this instruction is:

```
REPRO
```

The instruction may not be labeled and it has no operands. The instruction means:

copy the next source line into the object module and do not assemble that line.

The instruction REPRO is executed during pass I of an assembly run, so the reproduced statements appear at the beginning of the object module before the ESD.

4.5. Instruction ORG

Because the Link Editor produces load modules, the use of the assembler instruction ORG is restricted. The instruction ORG may not set the location counter to a lower value.

The Link Editor fills the space between two consecutive parts of defined code with arbitrary data. An ORG-instruction that sets the location counter to a value which is lower than the current value introduces a "negative" space. Such a space cannot be handled by the Link Editor.

4.6. Instruction END

The meaning of the END-instruction is changed as follows:

1. An END-instruction is optional;

2. An END-instruction does not require an operand;

3. The operand of an END-instruction is ignored, except when this is an address. Then it is used as the entry point of the module. Otherwise the module has no explicitely defined entry point.

Note: A phase, as produced by the Link Editor (see below), needs an entry point. The entry point of the last module edited in a phase is used. When no entry point exists, the begin address of the phase is taken as the entry point.

5. Link Editor commands

Although the Link Editor is able to produce an executable program from single object modules, some Link Editor commands might be useful. The following Link Editor commands exist:

1. PHASE
   This command is used to identify a phase. A phase is a separtely stored load module. Such a module is equivalent to the modules produced by the SDOS-command MODULE. The format of a PHASE-command is:

   <b> PHASE <name> [, <origin>] [<id>]

   The command line must begin with a blank. <name> denotes the name of the load module; this is the only required operand.

   <origin> specifies the begin address of the phase. It is represented by a normal expression. The default value is the current state of the location counter.

   <id> is the module identification. This is a character string of at most 21 characters; longer strings are truncated after 21 characters.

2. INCLUDE
   This command is used to include a separtely assembled module. The format of the command is:

   <b> INCLUDE <name>

   The command line must begin with a blank. <name> denotes the name of the object module to include.

   The Link Editor replaces this command by the module to include.

Besides the input of the Link Editor may comprise comment lines. The first byte in a comment line is the character '*'. Comment lines are only listed in the store map. Comment lines must contain at least 2 characters (exclusive "EOL"); less characters cause an I/O-error.

6. Execution of the assembler program

The 2650 Assembler is initiated by means of the system command ASM. The format of this command is:

```
ASM <ifile> <lfile> <ofile> <gen>
```

<ifile> denotes the file containing the source text. This is the only required parameter. <lfile> identfies the file that is to contain the program listing. If this parameter is omitted, no listing is produced. If CONO is specified as <lfile>, the listing lines are truncated after 79 characters. <ofile> identifies the file that is to contain the object module. No object module is stored if this parameter is omitted. <gen> indicates if macro expansions are to be listed. When a fourth parameter is specified, the listing will also contain the instructions that are generated by the macro-processor.

After successfully initiation, the Assembler displays the message:

```
** 2650 ASSEMBLER **
```

All detected errors are displayed on "CONO", the errors which are encountered during pass II are also indicated in the listing. Upon termination of the assembly program the total number of assembly errors is displayed. The same message is added to the listing.

The detected errors in the source text are identified by a letter. These are the same as used by the Assembler ver 2.0.

Upon the detection of an input/output error a message is displayed and the program is terminated. The format of such a message is:

```
** ERR <cc> <id>FILE **
```

<cc> denotes the SRB status (see TWIN Operating System); <id> identifies the concerned file. The possible identifiers are:

```
S : source file;
L : listing file;
O : object file.
```

7. Execution of the link editing program

The Link Editor is invoked by means of the system command LNK. The format of this command is:

```
LNK <ofile> <lfile> <bfile>
```

<ofile> denotes the file which contains the object code (including PHASE-commands and INCLUDE-commands). This is the only required parameter. <lfile> specifies the file which is to contain the store map. If this parameter is omitted, no store map is produced. The parameter <bfile> is useful when the object code does not contain a PHASE-command. In this case the edited programs are stored in <bfile>. When both the object code contains no PHASE-commands and the parameter <bfile> is omitted, the edited programs are not stored.

After successfully initiation, the program displays the message:

```
** 2650 LNKEDT **
```

The store map contains the following data:

1. For each phase are listed the phase name (if any), the begin address, the end address, the entry point and all object modules contained in that phase;

2. For each object module are listed the module name (if such a name exists), the offset with respect to the assembled origin and all external symbols defined in the module with their value.

The Link Editor can display certain error messages. These messages are:

1. INVALID PARAMETER
   The system command contains an invalid parameter. The program is terminated.

2. TOO DEEP NESTING
   The nesting level of included modules exceeds 4. The program is terminated.

3. TOO MANY PHASES
   The object file specifies more than 20 phases. The program is terminated.

4. INVALID ASSEMBLED ORIGIN
   The indicated section started with an ORG-instruction, but the specified location is already occupied. The section is edited at the first available position.

5. TOO MANY SYMBOLS
   The symbol list is full. The program is terminated.

6. s <record>
   Syntax error in record. The record is ignored.

7. u <record>
   Undefined symbol in record. The record is ignored.

8. m <symbol> <file>
   The external symbol <symbol> is defined a second time in <file>. This occurrence of the symbol is ignored.

9. u <symbol> <file>
   The external symbol <symbol> is defined as EXTRN in <file>, but it has not been defined as ENTRY in another module. The value 0 is assigned to this symbol.

10. p <section> <file>
    A paging error is detected in the section <section>. The address is resolved for the current page.

The Link Editor displays input/output errors in the same way as the Assembler.