Machine Independent Assembler Features

(Program blocks, Control Session and Linking)

Program Blocks

- Collect many pieces of code/data that scatter in the source program but have the same kind into a single block in the generated object program.
  - For example, code block, initialized data block, un-initialized data block. (Like code, data segments on a Pentium PC).
  - Advantage:
    - Because pieces of code are closer to each other now, format 4 can be replaced with format 3, saving space and execution time.
    - Code sharing and data protection can better be done.
- With this function, in the source program, the programmer can put related code and data near each other for better readability.

Program Block Example

There is a default block.

```
 5    COPY      START   0               COPY FILE FROM INPUT TO OUTPUT
10    FIRST     STL     RETADR          SAVE RETURN ADDRESS
15    CLOOP     JSUB    RDREC           READ INPUT RECORD
20              LDA     LENGTH          TEST FOR EOF (LENGTH = 0)
25              COMP    #0
30              JEQ     ENDFIL          EXIT IF EOF FOUND
35              JSUB    WRREC           WRITE OUTPUT RECORD
40              J       CLOOP           LOOP
45    ENDFIL    LDA     =C'EOF'         INSERT END OF FILE MARKER
50              STA     BUFFER
55              LDA     #3              SET LENGTH = 3
60              STA     LENGTH
65              JSUB    WRREC           WRITE EOF
70              J       @RETADR         RETURN TO CALLER
92              USE     CDATA
95    RETADR    RESW    1
100   LENGTH    RESW    1               LENGTH OF RECORD
103             USE     CBLKS
105   BUFFER    RESB    4096            4096-BYTE BUFFER AREA
106   BUFEND    EQU     *               FIRST LOCATION AFTER BUFFER
107   MAXLEN    EQU     BUFEND-BUFFER   MAXIMUM RECORD LENGTH
110
```

```
110     .
115     .       SUBROUTINE TO READ RECORD INTO BUFFER
120     .
123             USE
125     RDREC   CLEAR   X               CLEAR LOOP COUNTER
130             CLEAR   A               CLEAR A TO ZERO
132             CLEAR   S               CLEAR S TO ZERO
133             +LDT    #MAXLEN
135     RLOOP   TD      INPUT           TEST INPUT DEVICE
140             JEQ     RLOOP           LOOP UNTIL READY
145             RD      INPUT           READ CHARACTER INTO REGISTER A
150             COMPR   A,S             TEST FOR END OF RECORD (X'00')
155             JEQ     EXIT            EXIT LOOP IF EOR
160             STCH    BUFFER,X        STORE CHARACTER IN BUFFER
165             TIXR    T               LOOP UNLESS MAX LENGTH
170             JLT     RLOOP             HAS BEEN REACHED
175     EXIT    STX     LENGTH          SAVE RECORD LENGTH
180             RSUB                    RETURN TO CALLER
183             USE     CDATA
185     INPUT   BYTE    X'F1'           CODE FOR INPUT DEVICE
195
```

Use the default block.

```
195     .
200     .       SUBROUTINE TO WRITE RECORD FROM BUFFER
205     .
208             USE
210     WRREC   CLEAR   X               CLEAR LOOP COUNTER
212             LDT     LENGTH
215     WLOOP   TD      =X'05'          TEST OUTPUT DEVICE
220             JEQ     WLOOP           LOOP UNTIL READY
225             LDCH    BUFFER,X        GET CHARACTER FROM BUFFER
230             WD      =X'05'          WRITE CHARACTER
235             TIXR    T               LOOP UNTIL ALL CHARACTERS
240             JLT     WLOOP             HAVE BEEN WRITTEN
245             RSUB                    RETURN TO CALLER
252             USE     CDATA
253             LTORG
255             END     FIRST
```

Use the default block.

The default block (unnamed) contains the executable instructions.

The CDATA block contains all data areas that are a few words or less.

The CBLKS block contain all data areas that consist of large blocks of memory.

Assembler's Job

- A program block may contain several separate segments of the source program.
- The assembler will (logically) rearrange these segments to gather together the pieces of each block.
- These blocks will then be assigned addresses in the object program, with the blocks appearing in the same order in which they were first begun in the source program.
- The result is the same as if the programmer had physically rearranged the source statements to group together all the source lines belonging to each block.

Assembler's Processing

- Pass 1:
  - Maintain a separate location counter for each program block.
  - The location counter for a block is initialized to 0 when the block is first begun.
  - The current value of this location counter is saved when switching to another block, and the saved value is restored when resuming a previous block.
  - Thus, during pass 1, each label is assigned an address that is relative to the beginning of the block that contains it.
  - After pass 1, the latest value of the location counter for each block indicates the length of that block.
  - The assembler then can assign to each block a starting address in the object program.

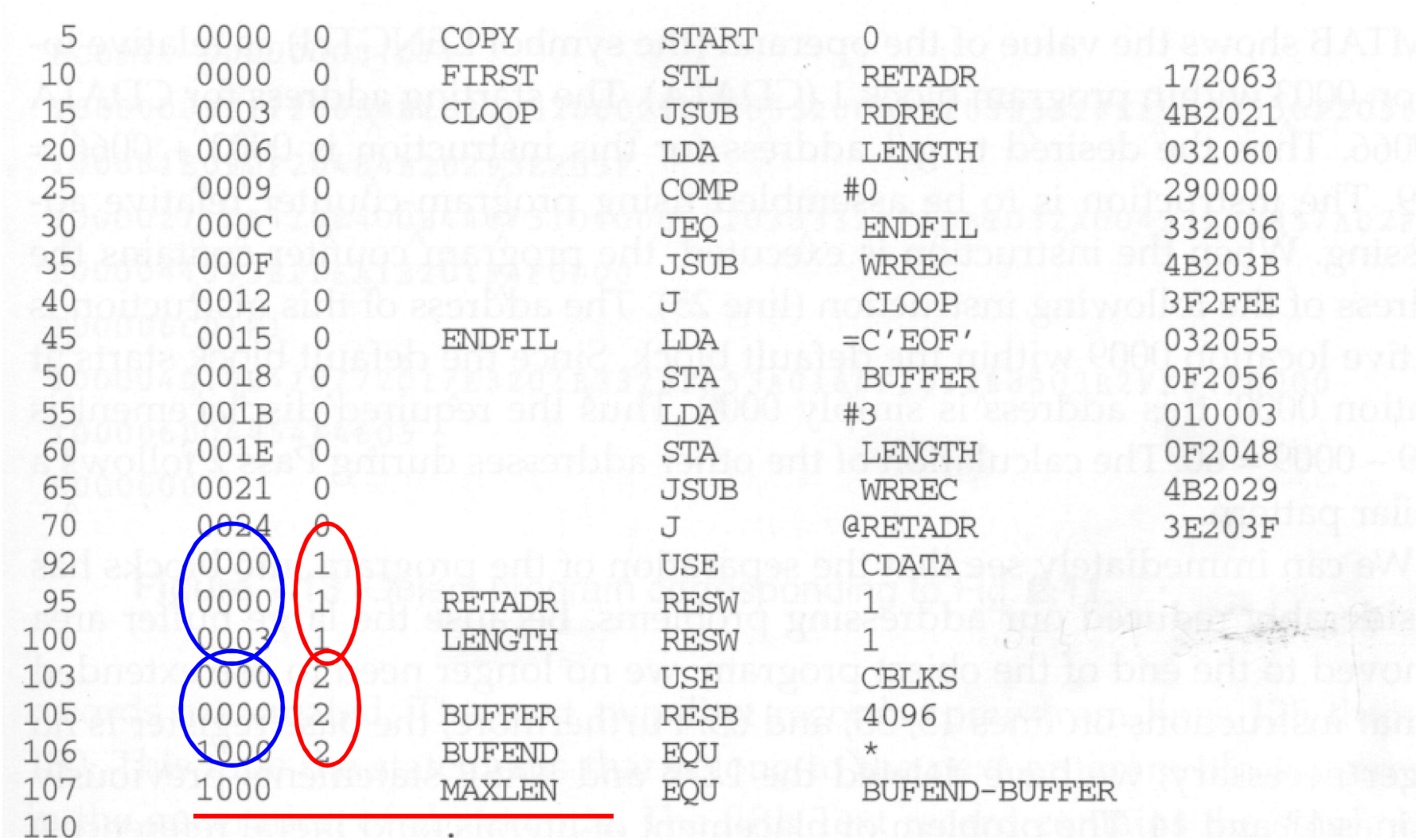
Assembler's Processing

- Pass 2
  - When generating object code, the assembler needs the address for each symbol relative to the start of the object program (not the start of an individual problem block)
  - This can be easily done by adding the location of the symbol (relative to the start of its block) to the assigned block starting address.

Example Program

Loc/Block

```
 5    0000  0     COPY      START   0
10    0000  0     FIRST     STL     RETADR          172063
15    0003  0     CLOOP     JSUB    RDREC           4B2021
20    0006  0               LDA     LENGTH          032060
25    0009  0               COMP    #0              290000
30    000C  0               JEQ     ENDFIL          332006
35    000F  0               JSUB    WRREC           4B203B
40    0012  0               J       CLOOP           3F2FEE
45    0015  0     ENDFIL    LDA     =C'EOF'         032055
50    0018  0               STA     BUFFER          0F2056
55    001B  0               LDA     #3              010003
60    001E  0               STA     LENGTH          0F2048
65    0021  0               JSUB    WRREC           4B2029
70    0024  0               J       @RETADR         3E203F
92    0000  1               USE     CDATA
95    0000  1     RETADR    RESW    1
100   0003  1     LENGTH    RESW    1
103   0000  2               USE     CBLKS
105   0000  2     BUFFER    RESB    4096
106   1000  2     BUFEND    EQU     *
107   1000        MAXLEN    EQU     BUFEND-BUFFER
110               .
```



```
 5     0000  0     COPY      START   0
10     0000  0     FIRST     STL     RETADR          172063
15     0003  0     CLOOP     JSUB    RDREC           4B2021
20     0006  0               LDA     LENGTH          032060
25     0009  0               COMP    #0              290000
30     000C  0               JEQ     ENDFIL          332006
35     000F  0               JSUB    WRREC           4B203B
40     0012  0               J       CLOOP           3F2FEE
45     0015  0     ENDFIL    LDA     =C'EOF'         032055
50     0018  0               STA     BUFFER          0F2056
55     001B  0               LDA     #3              010003
60     001E  0               STA     LENGTH          0F2048
65     0021  0               JSUB    WRREC           4B2029
70     0024  0               J       @RETADR         3E203F
92     0000  1               USE     CDATA
95     0000  1     RETADR    RESW    1
100    0003  1     LENGTH    RESW    1
103    0000  2               USE     CBLKS
105    0000  2     BUFFER    RESB    4096
106    1000  2     BUFEND    EQU     *
107    1000        MAXLEN    EQU     BUFEND-BUFFER
110
```

There is no block number for MAXLEN.
This is because MAXLEN is an absolute symbol.

```
115                      .        SUBROUTINE TO READ RECORD INTO BUFFER
120                      .
123     0027  0          USE
125     0027  0  RDREC   CLEAR     X                   B410
130     0029  0          CLEAR     A                   B400
132     002B  0          CLEAR     S                   B440
133     002D  0          +LDT      #MAXLEN             75101000
135     0031  0  RLOOP   TD        INPUT               E32038
140     0034  0          JEQ       RLOOP               332FFA
145     0037  0          RD        INPUT               DB2032
150     003A  0          COMPR     A,S                 A004
155     003C  0          JEQ       EXIT                332008
160     003F  0          STCH      BUFFER,X            57A02F
165     0042  0          TIXR      T                   B850
170     0044  0          JLT       RLOOP               3B2FEA
175     0047  0  EXIT     STX       LENGTH              13201F
180     004A  0          RSUB                          4F0000
183     0006  1          USE       CDATA
185     0006  1  INPUT    BYTE      X'F1'               F1
```

Symbol Table After Pass 1

Block name	Block number	Address	Length
(default)	0	0000	0066
CDATA	1	0066	000B
CBLKS	2	0071	1000

Code Generation in Pass 2

- 20 0006 0 LDA LENGTH 032060

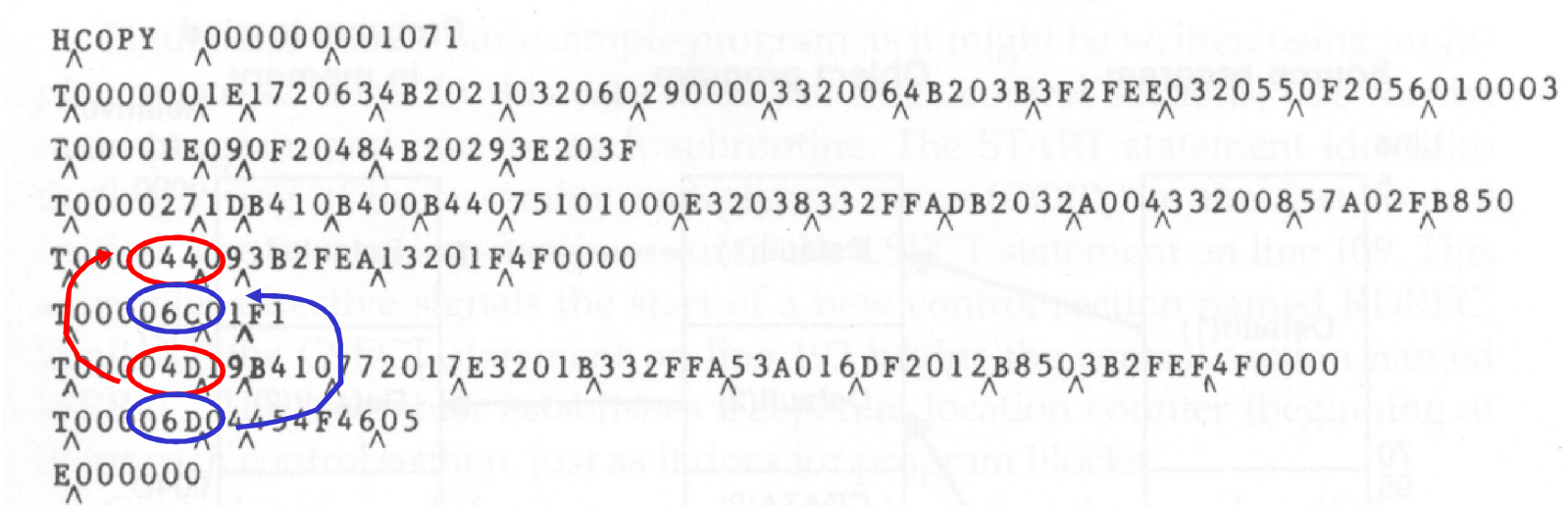
- The SYMTAB shows that LENGTH has a relative address 0003 within problem block 1 (CDATA).
- The starting address for CDATA is 0066.
- Thus the desired target address is 0066 + 0003 = 0069.
- Because this instruction is assembled using program counter-relative addressing, and PC will be 0009 when the instruction is executed (the starting address for the default block is 0), the displacement is 0069 – 0009 = 60.

Advantages

- Because the large buffer area is moved to the end of the object program, we no longer need to use format 4 instructions on line 15, 35, and 65.
- For the same reason, use of the base register is no longer necessary; the LDB and BASE have been deleted.
- Code sharing and data protection can be more easily achieved.

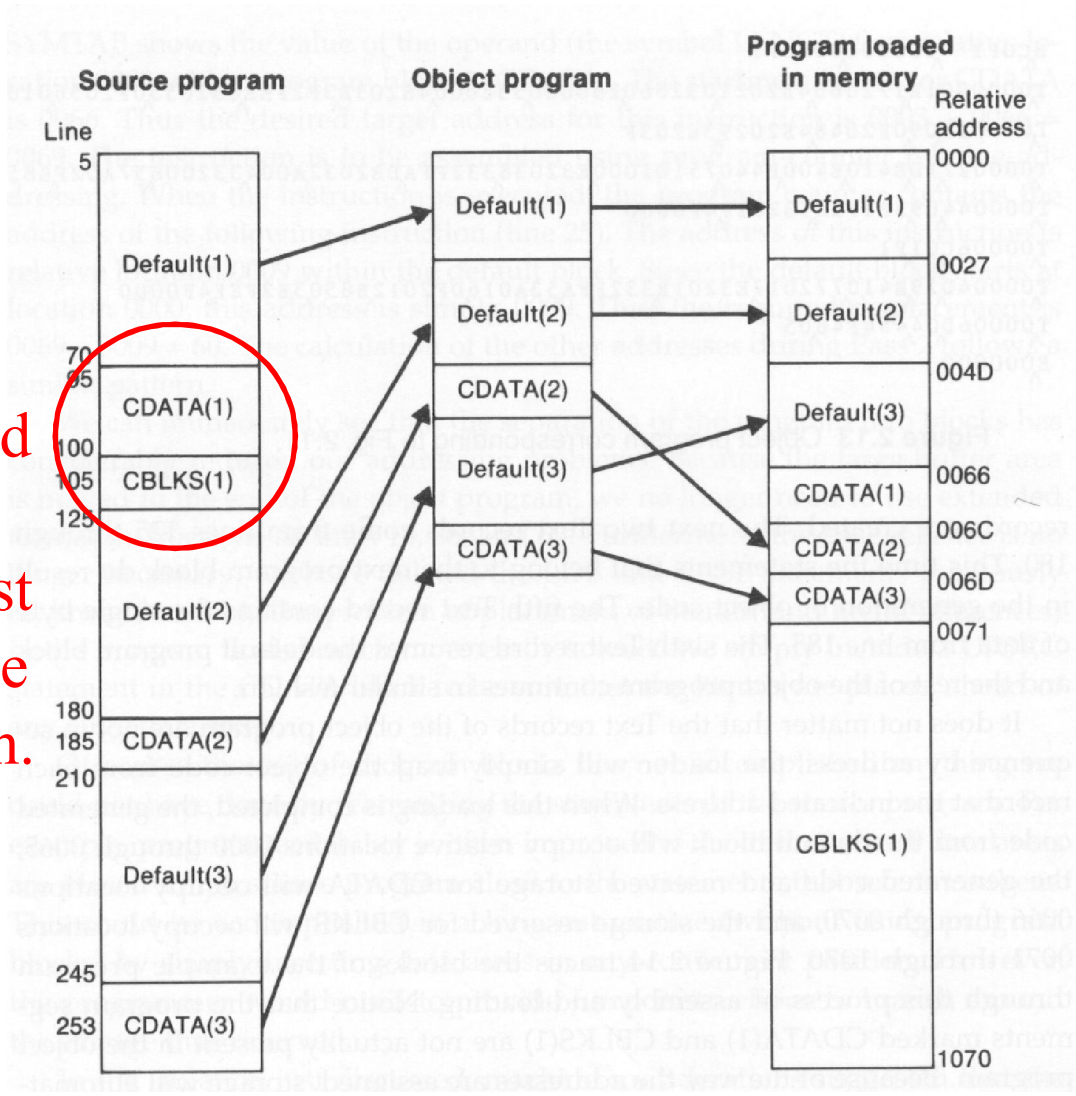
Object Code Layout

- Although the assembler internally rearranges code and data to form blocks, the generated code and data need not be physically rearranged. The assembler can simple write the object code as it is generated during pass 2 and insert the proper load address in each text record.



Leave the Job to Loader

No code need to be generated for these two blocks. We just need to reserve space for them.



Control Section & Program Linking

Control Section

- A control section is a part of the program that maintains its identity after assembly.
- Each such control section can be loaded and relocated independently of the others. (Main advantage)
- Different control sections are often used for subroutines or other logical subdivisions of a program.
- The programmer can assemble, load, and manipulate each of these control sections separately.

Program Linking

- Instructions in one control section may need to refer to instructions or data located in another control section. (Like external variables used in C language)
- Thus, program (actually, control section) linking is necessary.
- Because control sections are independently loaded and relocated, the assembler is unable to know a symbol's address at assembly time. This job can only be delayed and performed by the loader.
- We call the references that are between control sections "external references".
- The assembler generates information for each external reference that will allow the loader to perform the required linking.

Control Section Example

Default control section

```
 5    COPY      START   0               COPY FILE FROM INPUT TO OUTPUT
 6              EXTDEF  BUFFER,BUFEND,LENGTH
 7              EXTREF  RDREC,WRREC
10    FIRST     STL     RETADR          SAVE RETURN ADDRESS
15    CLOOP    +JSUB    RDREC           READ INPUT RECORD
20              LDA     LENGTH          TEST FOR EOF (LENGTH = 0)
25              COMP    #0
30              JEQ     ENDFIL          EXIT IF EOF FOUND
35             +JSUB    WRREC           WRITE OUTPUT RECORD
40              J       CLOOP           LOOP
45    ENDFIL    LDA     =C'EOF'         INSERT END OF FILE MARKER
50              STA     BUFFER
55              LDA     #3              SET LENGTH = 3
60              STA     LENGTH
65             +JSUB    WRREC           WRITE EOF
70              J       @RETADR         RETURN TO CALLER
95    RETADR    RESW    1
100   LENGTH    RESW    1               LENGTH OF RECORD
103             LTORG
105   BUFFER    RESB    4096            4096-BYTE BUFFER AREA
106   BUFEND    EQU     *
107   MAXLEN    EQU     BUFEND-BUFFER
```

A new control section

```
109    RDREC     CSECT
110    .
115    .         SUBROUTINE TO READ RECORD INTO BUFFER
120    .
122               EXTREF    BUFFER,LENGTH,BUFEND
125               CLEAR     X              CLEAR LOOP COUNTER
130               CLEAR     A              CLEAR A TO ZERO
132               CLEAR     S              CLEAR S TO ZERO
133               LDT       MAXLEN
135    RLOOP      TD        INPUT          TEST INPUT DEVICE
140               JEQ       RLOOP          LOOP UNTIL READY
145               RD        INPUT          READ CHARACTER INTO REGISTER A
150               COMPR     A,S            TEST FOR END OF RECORD (X'00')
155               JEQ       EXIT           EXIT LOOP IF EOR
160               +STCH     BUFFER,X       STORE CHARACTER IN BUFFER
165               TIXR      T              LOOP UNLESS MAX LENGTH
170               JLT       RLOOP            HAS BEEN REACHED
175    EXIT       +STX      LENGTH         SAVE RECORD LENGTH
180               RSUB                     RETURN TO CALLER
185    INPUT      BYTE      X'F1'          CODE FOR INPUT DEVICE
190    MAXLEN     WORD      BUFEND-BUFFER
```

A new control section

```
193     WRREC     CSECT
195     .
200     .         SUBROUTINE TO WRITE RECORD FROM BUFFER
205     .
207               EXTREF    LENGTH,BUFFER
210               CLEAR     X                 CLEAR LOOP COUNTER
212               +LDT      LENGTH
215     WLOOP     TD        =X'05'            TEST OUTPUT DEVICE
220               JEQ       WLOOP             LOOP UNTIL READY
225               +LDCH     BUFFER,X          GET CHARACTER FROM BUFFER
230               WD        =X'05'            WRITE CHARACTER
235               TIXR      T                 LOOP UNTIL ALL CHARACTERS
240               JLT       WLOOP               HAVE BEEN WRITTEN
245               RSUB                        RETURN TO CALLER
255               END       FIRST
```

External References

- Symbols that are defined in one control section cannot be used directly by another control section.
- They must be identified as external references for the loader to handle.
- Two assembler directives are used:
  - EXTDEF (external definition)
    - Identify those symbols that are defined in this control section and can be used in other control sections.
    - Control section names are automatically considered as external symbols.
  - EXTREF (external reference)
    - Identify those symbols that are used in this control section but defined in other control sections.

Code Involving External Reference (1)

- 15 0003 CLOOP +JSUB RDREC 4B100000
- The operand (RDREC) is named in the EXTREF statement, therefore this is an external reference.
- Because the assembler has no idea where the control section containing RDREC will be loaded, it cannot assemble the address for this instruction.
- Therefore, it inserts an address of zero.
- Because the RDREC has no predictable relationship to anything in this control section, relative addressing cannot be used.
- Instead, an extended format instruction must be used.
  - This is true of any instruction whose operand involves an external reference.

Code Involving External Reference (2)

- 160 0017 +STCH BUFFER,X 57900000
- This instruction makes an external reference to BUFFER.
- The instruction is thus assembled using extended format with an address of zero.
- The x bit is set to 1 to indicate indexed addressing.

Code Involving External Reference (3)

- 190 0028 MAXLEN WORD BUFEND – BUFFER 000000
- The value of the data word to be generated is specified by an expression involving two external references.
- As such, the assembler stores this value as zero.
- When the program is loaded, the loader will add to this data area the address of BUFEND and subtract from it the address of BUFFER, which then results in the desired value.
- Notice the difference between line 190 and 107. In line 107, EQU can be used because BUFEND and BUFFER are defined in the same control section and thus their difference can be immediately calculated by the assembler.

External Reference Processing

- The assembler must remember (via entries in SYMTAB) in which control section a symbol is defined.
- Any attempt to refer to a symbol in another control section must be flagged as an error unless the symbol is identified (via EXTREF) as an external reference.
- The assembler must allow the same symbol to be used in different control sections.
  - E.g., the conflicting definitions of MAXLEN on line 107 and 190 should be allowed.

Two New Record Types

- We need two new record types in the object program and a change in the previous defined modification record type.
- Define record
  - Give information about external symbols that are defined in this control section
- Refer record
  - List symbols that are used as external references by this control section.

Define record:

Col. 1	D
Col. 2–7	Name of external symbol defined in this control section
Col. 8–13	Relative address of symbol within this control section (hexadecimal)
Col. 14–73	Repeat information in Col. 2–13 for other external symbols

Refer record:

Col. 1	R
Col. 2–7	Name of external symbol referred to in this control section
Col. 8–73	Names of other external reference symbols

Revised Modification Record

Modification record (revised):

Col. 1	M
Col. 2–7	Starting address of the field to be modified, relative to the beginning of the control section (hexadecimal)
Col. 8–9	Length of the field to be modified, in half-bytes (hexadecimal)
Col. 10	Modification flag (+ or –)
Col. 11–16	External symbol whose value is to be added to or subtracted from the indicated field

Object Program Example

```
HCOPY  000000001033
DBUFFER000033BUFEND001033LENGTH00002D
RRDREC WRREC
T0000001D1720274B1000000320232900003320074B1000003F2FEC032016 0F2016
T00001D0D0100030F200A4B1000003E2000
T000003003454F46
M00000405+RDREC
M00001105+WRREC
M00002405+WRREC
E000000


HRDREC 000000000002B
RBUFFERLENGTHBUFEND
T0000001DB410B400B44077201FE3201B332FFADB2015A004332009579000000B850
T00001D0E3B2FE9131000004F0000F1000000
M00001805+BUFFER
M00002105+LENGTH
M00002806+BUFEND
M00002806-BUFFER
E


HWRREC 000000000001C
RLENGTHBUFFER
T0000001CB41077100000E32012332FFA53900000DF2008B8503B2FEE4F000005
M00000305+LENGTH
M00000D05+BUFFER
E
```

Program Relocation

- The modified "modification record" can still be used for program relocation.

```
M00000705
M00001405
M00002705
```

to

Program name

```
M00000705+COPY
M00001405+COPY
M00002705+COPY
```

More Restriction on Expression

- Previously we required that all of the relative terms in an expression be paired to make the expression an absolute expression.
- With control sections, the above requirement is not enough.
- We must require that both terms in each pair must be relative within the same control section.
  - E.g.1. BUFEND- BUFFER (allowed) because they are defined in the same control section.
  - E.g.2. RDRED – COPY (not allowed) because the value is unpredictable.
- Otherwise, the expression is a relative expression and the unresolved terms must be handled by the loader using the modification records.