Assembler

Machine Dependent Assembler Features

- Machine Dependent Assembler Features SIC/XE
  - Instruction formats and addressing modes
  - Program relocation

Machine Dependent Assembler Features

- Instruction Formats and Addressing Modes
  - PC-relative or Base-relative addressing: **op m**
    - The assembler directive BASE is used with base-relative addressing
    - If displacements are too large to fit into a 3-byte instruction, then 4-byte extended format is used
  - Indirect addressing : **op @m**
  - Immediate addressing : **op #c**
  - Extended format : **+op m3**
  - Index addressing : **op m, x**
  - register-to-register instructions
  - larger memory -> multi-programming (program allocation)

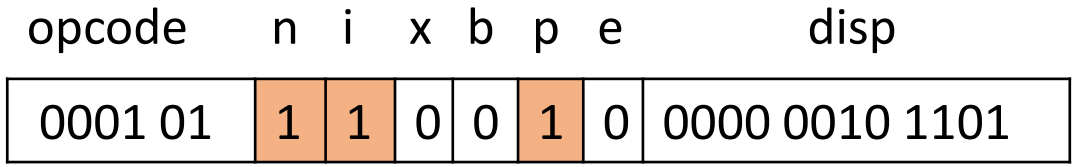
Address Translation

- Register translation
  - Register names (A, X, L, B, S, T, F, PC, SW) and their values (0,1, 2, 3, 4, 5, 6, 8, 9)
  - Preloaded in SYMTAB
- Address translation
  - Most register-memory instructions use program counter relative or base relative addressing
  - Format 3: 12-bit displacement field
    - base-relative : 0~4095
    - pc-relative : -2048~2047
  - Format 4: 20-bit address field

Line	Loc	Source statement			Object code
5	0000	COPY	START	0	
10	0000	FIRST	STL	RETADR	17202D
12	0003		LDB	#LENGTH	69202D
13			BASE	LENGTH	
15	0006	CLOOP	+JSUB	RDREC	4B101036
20	000A		LDA	LENGTH	032026
25	000D		COMP	#0	290000
30	0010		JEQ	ENDFIL	332007
35	0013		+JSUB	WRREC	4B10105D
40	0017		J	CLOOP	3F2FEC
45	001A	ENDFIL	LDA	EOF	032010
50	001D		STA	BUFFER	0F2016
55	0020		LDA	#3	010003
60	0023		STA	LENGTH	0F200D
65	0026		+JSUB	WRREC	4B10105D
70	002A		J	@RETADR	3E2003
80	002D	EOF	BYTE	C'EOF'	454F46
95	0030	RETADR	RESW	1	
100	0033	LENGTH	RESW	1	
105	0036	BUFFER	RESB	4096	
110		.			
115		.	SUBROUTINE TO READ RECORD INTO BUFFER		
120		.			
125	1036	RDREC	CLEAR	X	B410
130	1038		CLEAR	A	B400
132	103A		CLEAR	S	B440
133	103C		+LDT	#4096	75101000
135	1040	RLOOP	TD	INPUT	E32019
140	1043		JEQ	RLOOP	332FFA
145	1046		RD	INPUT	DB2013
150	1049		COMPR	A,S	A004
155	104B		JEQ	EXIT	332008
160	104E		STCH	BUFFER,X	57C003

PC Relative Addressing Mode

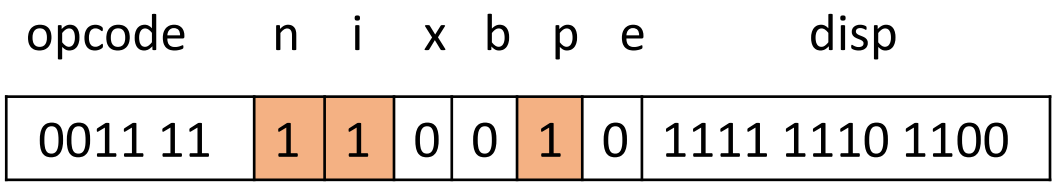
- **Line Loc. Label Instruction Operand Object Code**
- **10 0000 FIRST STL RETADR 17202D**
- RETADR=30
- Opcode STL (6 bits) =14 = 0001 0100
- Displacement= RETADR –(PC) = 30-3 = 2D
- nixbpe=110010
  - n=1, i = 1: indicate neither indirect nor immediate addressing
  - p = 1: indicate PC-relative addressing



Object Code = 17202D

PC Relative Addressing Mode

- **Line Loc. Label Instruction Operand Object Code**
- **40 0017 J CLOOP 3F2FEC**
- Address of CLOOP=6
- Opcode, J=3C = 0011 1100
- Displacement= CLOOP - (PC) = 6 - 1A = -14 = FEC (2’ s complement for negative number)
- nixbpe=110010



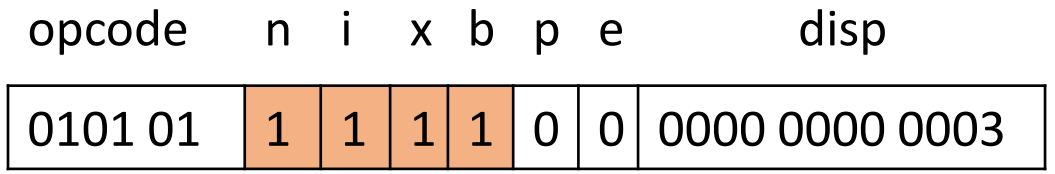
Object Code = 3F2FEC

Base-Relative Addressing Mode

- Base register is under the control of the programmer
- Assembler directive **BASE**
  - to specify which value to be assigned to base register (B)
- Assembler directive **NOBASE**
  - inform the assembler that the contents of base register no longer be used for addressing
- BASE and NOBASE produce no executable code

Base-Relative Addressing Mode

- 12 LDB #LENGTH ; address of the symbol LENGTH (0033) is loaded into reg. B
- 13 BASE LENGTH ; no object code
- **160 104E STCH BUFFER, X 57C003**
- Address of BUFFER=36
- Address of LENGTH =33, therefore (B)=33
- Opcode, LDB =54=0101 0100
- Displacement= BUFFER – (B) = 0036 –0033= 3
- nixbpe=111100



Object Code = 57C003

Immediate Addressing Mode

- Convert the immediate operand to its internal representation and insert it into the instruction
- **Line Loc. Label Instruction Operand Object Code**
- **55 0020 LDA #3 010003**
- Opcode=00 ; nixbpe=010000
- i = 1: immediate addressing

opcode	n	i	x	b	p	e	disp
0000 00	0	1	0	0	0	0	0000 0000 0003

Object Code = 010003

Indirect Addressing Mode

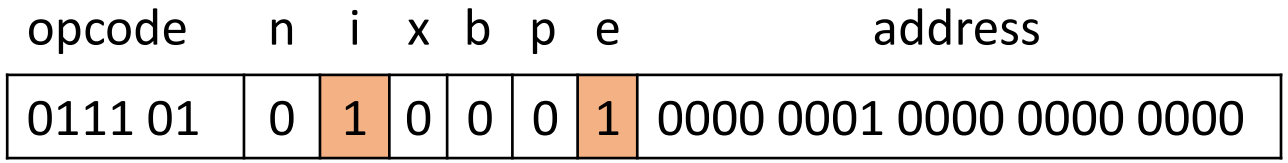
- The contents stored at the location represent the address of the operand, not the operand itself
- Target addressing is computed as usual (PC relative or BASE-relative)
- n bit is set to 1
- **70 002A J @RETADR 3E2003**
- Address of RETADR=30; Opcode= 3C=00111100
- Displacement= RETADR- (PC) = 0030 –002D =3
- nixbpe=100010 ; n = 1: Indirect addressing ; p = 1: PC-relative addressing

opcode	n	i	x	b	p	e	disp
0011 11	1	0	0	0	1	0	0000 0000 0011

Object Code = 3E2003

Format 4 Instruction

- **Line Loc. Label Instruction Operand Object Code**
- **133 103C +LDT #4096 75101000**
- Opcode=74
- n=0, i=1: Immediate addressing
- e=1: Format 4 instruction
- nixbpe=010001

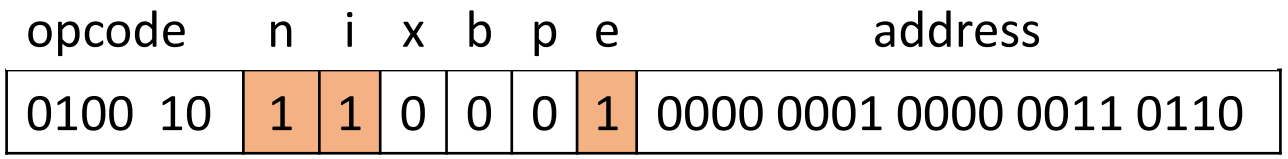


Object Code = 75101000

Format 4 Instruction

Source Statement		Object Code
+JSUB	**RDREC**	**4B101036**

- JSUB-48
- RDREC-1036



Object Code = 4B101036

Program Relocation

- The larger main memory of SIC/XE
  - Several programs can be loaded and run at the same time.
  - This kind of sharing of the machine between programs is called multiprogramming
- To take full advantage
  - Load programs into memory wherever there is room
  - Not specifying a fixed address at assembly time
  - Called program relocation
  - The actual starting address of the program is not known until run time

Program Relocation

Line	Loc	Source statement			Object code
5	0000	COPY	START	0	
10	0000	FIRST	STL	RETADR	17202D
12	0003		LDB	#LENGTH	69202D
13			BASE	LENGTH	
15	0006	CLOOP	+JSUB	RDREC	4B101036
20	000A		LDA	LENGTH	032026
	.				
	.				
	.				
125	1036	RDREC	CLEAR	X	B410
130	1038		CLEAR	A	B400
132	103A		CLEAR	S	B440
	.				
	.				
	.				

Program Relocation

15	0006	CLOOP	+JSUB	RDREC	4B101036
125	1036	RDREC	CLEAR	X	B410

- **Starting address 0**

  15 0006 CLOOP +JSUB RDREC 4B101036

- **Relocate the program to 1000**

  15 0006 CLOOP +JSUB RDREC 4B102036

- **Relocate the program to 3000**

  15 0006 CLOOP +JSUB RDREC 4B104036

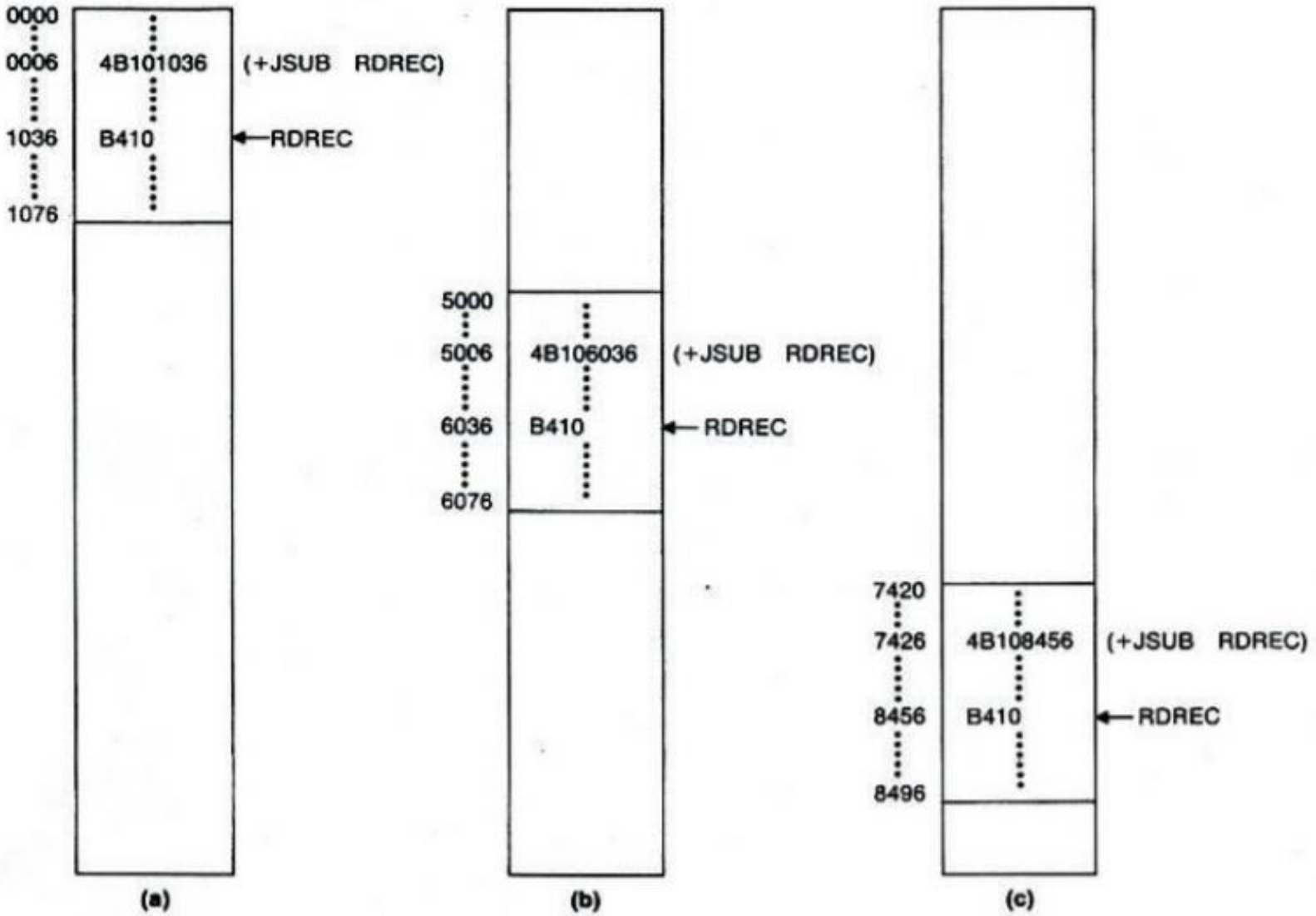
- Each absolute address should be modified

- Except for absolute address, the rest of the instructions need not be modified
  - Make some changes in the address portion of the instruction to retrieve the correct value

Program Relocation

- An object program should contain the information necessary to perform address modification for relocation
- The assembler must identify for the loader those parts of object program that need modification.
- An object program that contains the information necessary to perform this kind of modification is called a **relocatable program**.
- No instruction modification is needed for
  - Immediate addressing (not a memory address)
  - PC-relative
  - Base-relative addressing
- Only parts of the program that require modification at load time are those that specify direct addresses
  - In SIC/XE, only found in extended format instructions

Program Relocation



Program Relocation

- Solution to relocation problem
  - When the assembler generate the object code for the +JSUB RDREC instruction, it will insert the address of RDREC *relative to the start of the program.* (This is the reason for initializing the LOCCTR to 0).
  - The assembler will produce a command for the loader instructing it to add the beginning address of the program to the address field in the JSUB instruction at load time.
  - Command for the loader must be a part of the object program
  - Accomplished with a **modification record**

Relocatable Program

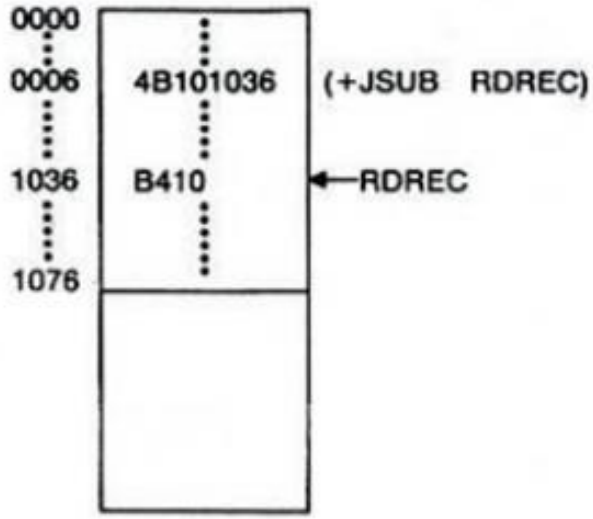
- Modification record
  - Col 1 M
  - Col 2-7 Starting location of the address field to be modified, relative to the beginning of the program
  - Col 8-9 Length of the address field to be modified, in half-bytes

- The modification record **+JSUB RDREC** is

  M^000007^05

- There is one modification record for each address

field that is to be modified when the program is loaded



Relocatable Program

0000	COPY	START	0	
0000	FIRST	STL	RETADR	17202D
		…..		
0006	CLOOP	+JSUB	RDREC	4B101036
		…..		
0013		+JSUB	WRREC	4B10105D
		……		
0026		+JSUB	WRREC	4B10105D
		…….		
1036	RDREC	CLEAR	X	B410
		…….		
105D	WRREC	CLEAR	X	B410

Object Program

```
HCOPY  000000001077
T00000001D17202D69202D4B10103603202629000033200074B10105D3F2FEC032010
T00001D130F20160100030F200D4B10105D3E2003454F46
T001036IDB410B400B44075101000E32019332FFADB2013A00433200857C003B850
T0010531D3B2FEA1340004F0000F1B410774000E32011332FFA53C003DF2008B850
T001070073B2FBF4F000005
M00000705
M00001405
M00002705
E000000
```

Machine Independent Assembler Features

Features of the assembler that are not closely related to machine architecture

1. Literals
2. Symbol defining statements
3. Expressions
4. Program blocks
5. Control sections and program linking

Literals

- It is convenient for a programmer to write the value of the constant operand as part of the instruction that uses it
  - Avoids having to define the constant elsewhere and make up a label for it
- Such an operand is called a literal since the value is stated ***literally*** in the instruction
- Literal is identified by the prefix '=' followed by the literal value

45 001A ENDFIL LDA =C'EOF' 032010

specifies a three byte operand whose value is the character string EOF

Literals vs. Immediate Operands

- Immediate Operands
  - The operand value is assembled as part of the machine instruction
  - e.g. 55 0020 LDA #3 010003
- Literals
  - The assembler generates the specified value as a constant at some other memory location
  - The address of this generated constant is used as the target address for the machine instruction

  - e.g.

```
45   001A   ENDFIL   LDA      =C'EOF'   032010
93                    LTORG
          002D     *        =C'EOF'             454F46
```

Literal - Implementation

- **Literal pools**
- Normally literals are placed into a pool at the end of the program
- In some cases, it is desirable to place literals into a pool at some other location in the object program
  - assembler directive LTORG
  - reason: keep the literal operand close to the instruction

- When the assembler encounters LTORG statement, it creates a literal pool that contains all the literal operands used since the previous LTORG

Literal - Implementation

- **Duplicate literals**

```
215     1062    WLOOP   TD      =X'05'
230     106B            WD      =X'05'
```

- The assemblers should recognize duplicate literals and store only one copy of the specified data value

1. Compare character strings
   - Comparison of the defining expression
   - Issue: Same literal name with different value, e.g. LOCCTR =*
2. Comparison of the generated data value
   - Eg: C'EOF' and X'454F46'
   - The benefits of using generated data value are usually not great enough to justify the additional complexity in the assembler

Literal Table: LITTAB

- LITTAB
  - literal name, the operand value, length, the address assigned to the operand
  - Hash table with literal name or value as the key

C'EOF'	454F46	3	002D
X'05'	05	1	1076

- Pass 1
  - build LITTAB with literal name, operand value and length, leaving the address unassigned
  - when LTORG statement is encountered, assign an address to each literal not yet assigned an address
- Pass 2
  - search LITTAB for each literal operand encountered
  - data values specified by literals in the literal pool are inserted at the appropriate places in the object program
  - generate modification record for literals that represent an address in the program (eg. Location counter value)

Symbol-Defining Statements

- Most assemblers provide an assembler directive (**EQU**) that allows programmers to define symbols and specify their values.

  symbol EQU value

  - This statement defines the symbol (enters into SYMTAB) and assigns to it the value specified
- Value can be
  - constant,
  - expression involving constants and previously defined symbols
  - making the source program easier to understand
  - no forward reference

Symbol-Defining Statements

1. Use symbolic names to improve readability

```
+LDT   #4096
```

Include the statements

```
MAXLEN      EQU     4096
            +LDT    #MAXLEN
```

2. Defining mnemonic names for registers

```
BASE    EQU     R1
COUNT   EQU     R2
INDEX   EQU     R3
```

ORG (origin)

- Indirectly assign values to symbols

  ORG value

- Value is a constant or an expression involving constants and previously defined symbols

- When this statement is encountered, assembler resets location counter to the specified value

- ORG statement will affect the values of all symbols defined until the next ORG
  - Since the values of symbols used as labels are taken from LOCCTR

Expressions

- Assemblers allow to use expressions where single operand is permitted
- Each expression will be evaluated by the assembler to produce a single operand or address value
- Operators: +, -, *, /
- Division produces integer result
- Individual terms in the expression may be constants, user defined terms or special terms
- Most common special terms is the current value of location counter (often designated by *). This term represents the value of the next unassigned memory location

```
106     BUFEND    EQU   *
```

- Gives BUFEND a value that is the address of the next byte after the buffer area

Expressions

- Expressions can be classified as *absolute expressions or relative expressions*
  - Absolute term: constant
  - Relative term: Labels on instructions and data areas, references to location counter
  - **MAXLEN EQU BUFEND-BUFFER**
  - BUFEND and BUFFER both are relative terms, representing addresses within the program
  - However the expression BUFEND-BUFFER represents an absolute value
- When relative terms are *paired with opposite signs*, the dependency on the program starting address is canceled out; the result is an absolute value

Expressions

- None of the relative terms may enter into a multiplication or division operation
- Errors:
  - BUFEND+BUFFER
  - 100-BUFFER
  - 3*BUFFER
- To determine the type of an expression
  - keep track of the types of all symbols defined in the program
  - Flag in the symbol table indicates the type of value (absolute or relative) in addition to the value itself

Symbol	Type	Value
RETADR	R	30
BUFFER	R	36
BUFEND	R	1036
MAXLEN	A	1000

Program Blocks

- The programs logically contained subroutines, data areas etc.
- The programs were treated as a single unit by the assembler resulting in a single block of object code.
- Within the object program, the generated machine instructions and data appeared in the same order that were written in the source program
- Many assemblers provide features that allow more flexible handling of the source and object programs.
  - Some features allow the generated machine instructions and data to appear in a different order from the source program-Program Blocks
  - Other features allow creation of different independent program units-Control sections

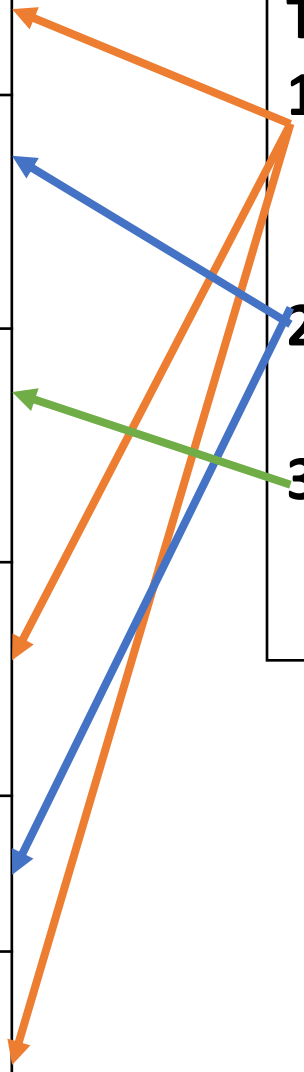
Program Blocks

- **Program blocks** refer to segments of code that are rearranged within a single object program unit
- Assembler directive **USE** indicates which portions of the program belong to which blocks

USE [blockname]

- Each program block may actually contain several separate segments of the source program

5	COPY	START	0
10	FIRST	STL	RETADR
		…	
92		**USE**	**CDATA**
95	RETADR	RESW	1
100	LENGTH	RESW	1
103		**USE**	**CBLKS**
105	BUFFER	RESB	4096
106	BUFEND	EQU	*
123		**USE**	
125	RDREC	CLEAR	X
		….	
183		**USE**	**CDATA**
185	INPUT	BYTE	X’F1’
208		**USE**	
210	WRREC	CLEAR X	



Three blocks:

1. **Unnamed Block:** contains executable instructions in the program
2. **CDATA:** Contains all data areas that are a few words or less in length
3. **CBLKS:** contains all data areas that consist of large blocks of memory

Program Blocks

- At the beginning of the program, statements are assumed to be part of the unnamed (default) block
- If no USE statement is included, the entire program belongs to a single block
- USE statement may also indicate the continuation of a previously begun block
- Each program block may actually contain several separate segments of the source program
- The assembler will logically rearrange these segments to gather together pieces of each block.
- These blocks will 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
- Result is same as if the programmer had physically rearranged the source statements to group together each source lines belonging to each block

Program Blocks

- **Pass 1**
  - Each program block has a separate location counter
  - The location counter for the block is initialized to zero when the block is first begun
  - The current value of the location counter is saved when switching to another block and restored when resuming a previous block
  - Each label is assigned an address that is relative to the start of the block that contains it
  - When labels are entered in the symbol table, the block name or number is also stored along with the relative address
  - At the end of Pass 1, the latest value of the location counter for each block indicates the length of that block
  - The assembler can then assign to each block a starting address in the object program

Line	Loc/Block		Source statement			Object code
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	

SUBROUTINE TO READ RECORD INTO BUFFER

```
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
```

SUBROUTINE TO WRITE RECORD FROM BUFFER

```
208     004D  0              USE
210     004D  0     WRREC    CLEAR   X             B410
212     004F  0              LDT     LENGTH        772017
215     0052  0     WLOOP    TD      =X'05'        E3201B
220     0055  0              JEQ     WLOOP         332FFA
225     0058  0              LDCH    BUFFER,X      53A016
230     005B  0              WD      =X'05'        DF2012
235     005E  0              TIXR    T             B850
240     0060  0              JLT     WLOOP         3B2FEF
245     0063  0              RSUB                  4F0000
252     0007  1              USE     CDATA
253                          LTORG
        0007  1     *        =C'EOF'               454F46
        000A  1     *        =X'05'                05
255                          END     FIRST
```

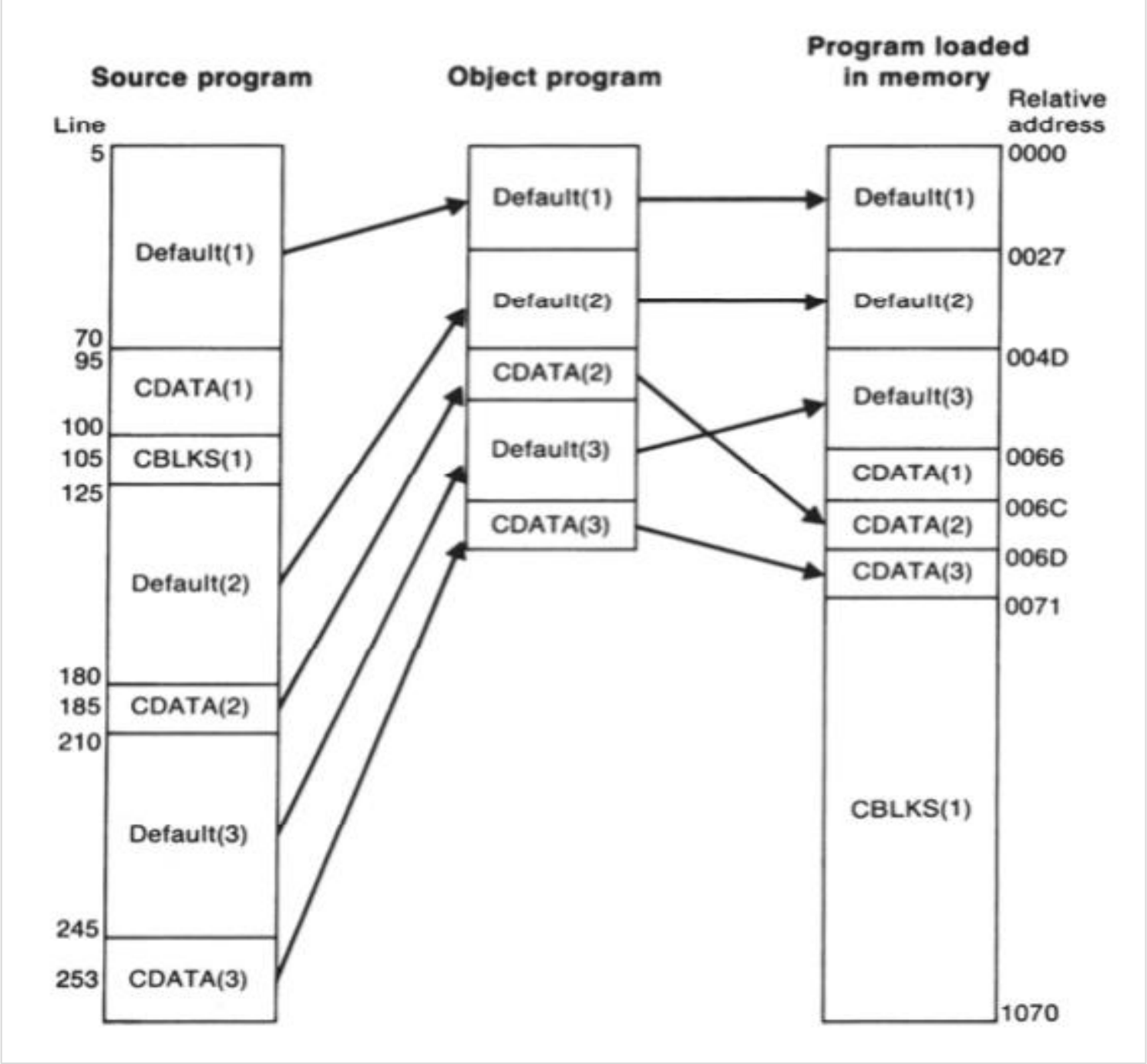
Program Blocks

- **Pass 2**
  - The address of each symbol can be computed by adding the assigned block starting address and the relative address of the symbol to that block
- Each source line is given a relative address assigned and a block number

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

- If CDATA starting address is 0066 and the following statement is an instruction within the block,
- 0003 LENGTH RESW 1
- Target address of LENGTH=(Address of CDATA)+0003= 0066+0003= 0069

5	COPY	START	0
10	FIRST	STL	RETADR
		…	
92		**USE**	**CDATA**
95	RETADR	RESW	1
100	LENGTH	RESW	1
103		**USE**	**CBLKS**
105	BUFFER	RESB	4096
106	BUFEND	EQU	*
123		**USE**	
125	RDREC	CLEAR	X
		….	
183		**USE**	**CDATA**
185	INPUT	BYTE	X’F1’
208		**USE**	
210	WRREC	CLEAR X	



Program Block

- Object program

```
HCOPY  000000001071
T0000001E1720634B20210320602900003320064B203B3F2FEE0320550F2056010003
T00001E090F20484B20293E203F                                    Default(1)
T0000271DB410B400B44075101000E32038332FFADB2032A00433200857A02FB850
T000044093B2FEA13201F4F0000                                    Default(2)
T00006C01F1                                                    CDATA(2)
T00004D19B410772017E3201B332FFA53A016DF2012B8503B2FEF4F0000    Default(3)
T00006D04454F4605                                              CDATA(3)
E000000
```

Control Sections and program Linking

- **Control sections** refer to segments of code that are translated into independent object program units
- 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 others
- Different control sections are most often used for subroutines and logical subdivisions of the program
- Programmer can assemble, load and manipulate each of these control sections separately
- Resulting flexibility is the major benefit of using control sections

Control Sections and Program Linking

- Control sections are handled separately by the assembler
- Symbols defined in one control section cannot be used in another control section directly; they should be identified as external references for the loader to handle
- The assembler must remember via symbol table in which control section a symbol is defined
- EXTDEF (External Definition) indicates symbols that are defined in the current control section and may be used by other sections
- EXREF (External Reference) indicates symbols defined in some other sections and used in the current control section

5	COPY	START	0
6		**EXTDEF**	**BUFFER, BUFEND, LENGTH**
7		**EXTREF**	**RDREC, WRREC**
10	FIRST	STL	RETADR
	BUFFER	RESB	4096
106	BUFEND	EQU	*
107	MAXLEN	EQU	BUFEND-BUFFER
109	**RDREC**	**CSECT**	
		EXTREF	**BUFFER, BUFEND, LENGTH**
190	MAXLEN	WORD	BUFEND-BUFFER
193	**WRREC**	**CSECT**	
		EXTREF	**LENGTH, BUFFER**
		CLEAR	X
		….	
		RSUB	
		END	FIRST

Three control sections: one for main program and two for subroutines

Separate location counter for each control section

COPY, WRREC and RDREC are automatically considered as external symbols

BUFFER, BUFEND and LENGTH are defined in control section COPY and are made available to other sections by EXTDEF statement

External Definition and References

- **<symbol> CSECT**
  - Assembler directive
  - Signals the start of a new control section
- External definition
  - **EXTDEF name [, name]**
  - EXTDEF names symbols that are defined in this control section and may be used by other sections
- External reference
  - **EXTREF name [,name]**
  - EXTREF names symbols that are used in this control section and are defined elsewhere

Control Sections and Program Linking

- When controls sections form logically related parts of a program, there should be a mechanism to link different parts together.
- Program linking is used to link together logically related control sections
- Problem: The assembler does not know where any other control section will be located at execution time.
  - When an instruction needs to refer to instructions or data located in another control section, the assembler is unable to process this reference.
  - References between control sections are called **external references**
  - The assembler has to generate information for external references that will allow the loader to perform the required linking.

Control Sections and Program Linking

- The assembler must include information in the object program that will cause the loader to insert proper values where they are required
- Three records
  - Define Record
  - Refer Record
  - Modification Record

Implementation

- **Define record**
  - Col. 1 D
  - Col. 2-7 Name of external symbol defined in this control section
  - Col. 8-13 Relative address within this control section (hexadecimal)
  - Col.14-73 Repeat information in Col. 2-13 for other external symbols
- Example: D^BUFFER 000033^BUFEND 001033^ LENGTH^00002D
- **Refer record**
  - Col. 1 R
  - Col. 2-7 Name of external symbol referred to in this control section
  - Col. 8-73 Name of other external reference symbols
- Example: R^RDREC^WRREC

Modification Record

- **Modification record**
  - Col. 1 M
  - Col. 2-7 Starting address of the field to be modified (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

- Note: Control section name is automatically an external symbol, i.e. it is available for use in Modification records.

Control Sections and Program Linking

15 0003 CLOOP +JSUB RDREC 4B100000

- The operand RDREC is named in EXTREF
- The assembler has no idea where the control section named RDREC will be loaded
- The assembler will insert an address of zero and pass this information to the loader
- The loader will insert the proper address at load time

- M^000004^05^+^RDREC

Control Sections and Program Linking

- There is a separate set of object program records (from Header through End) for each control section
- The object program of each control section will be the same as if the sections were assembled separately

Control Section	**Program Block**
CSECT directive	USE directive
Refers to segments of code that are translated into independent object program units	Refer to segments of code that are rearranged within a single object program unit
Handled separately by assembler	Not handled separately
All control sections need not be assembled at the same time	All program blocks should be assembled at the same time

Control Section : COPY

Line	Loc	Source statement			Object code
5	0000	COPY	START	0	
6			EXTDEF	BUFFER,BUFEND,LENGTH	
7			EXTREF	RDREC,WRREC	
10	0000	FIRST	STL	RETADR	172027
15	0003	CLOOP	+JSUB	RDREC	4B100000
20	0007		LDA	LENGTH	032023
25	000A		COMP	#0	290000
30	000D		JEQ	ENDFIL	332007
35	0010		+JSUB	WRREC	4B100000
40	0014		J	CLOOP	3F2FEC
45	0017	ENDFIL	LDA	=C'EOF'	032016
50	001A		STA	BUFFER	0F2016
55	001D		LDA	#3	010003
60	0020		STA	LENGTH	0F200A
65	0023		+JSUB	WRREC	4B100000
70	0027		J	@RETADR	3E2000
95	002A	RETADR	RESW	1	
100	002D	LENGTH	RESW	1	
103			LTORG		
	0030	*	=C'EOF'		454F46
105	0033	BUFFER	RESB	4096	
106	1033	BUFEND	EQU	*	
107	1000	MAXLEN	EQU	BUFEND-BUFFER	

Control Section : RDREC

```
109         0000    RDREC     CSECT
110                 .
115                 .         SUBROUTINE TO READ RECORD INTO BUFFER
120                 .
122                           EXTREF    BUFFER,LENGTH,BUFEND
125         0000              CLEAR     X                    B410
130         0002              CLEAR     A                    B400
132         0004              CLEAR     S                    B440
133         0006              LDT       MAXLEN               77201F
135         0009    RLOOP     TD        INPUT                E3201B
140         000C              JEQ       RLOOP                332FFA
145         000F              RD        INPUT                DB2015
150         0012              COMPR     A,S                  A004
155         0014              JEQ       EXIT                 332009
160         0017              +STCH     BUFFER,X             57900000
165         001B              TIXR      T                    B850
170         001D              JLT       RLOOP                3B2FE9
175         0020    EXIT      +STX      LENGTH               13100000
180         0024              RSUB                           4F0000
185         0027    INPUT     BYTE      X'F1'                F1
190         0028    MAXLEN    WORD      BUFEND-BUFFER        000000
```

Control Section : WRREC

```
193       0000      WRREC     CSECT
195                 .
200                 .         SUBROUTINE TO WRITE RECORD FROM BUFFER
205                 .
207                           EXTREF    LENGTH,BUFFER
210       0000                CLEAR     X                    B410
212       0002               +LDT       LENGTH               77100000
215       0006      WLOOP     TD        =X'05'               E32012
220       0009                JEQ       WLOOP                332FFA
225       000C               +LDCH      BUFFER,X             53900000
230       0010                WD        =X'05'               DF2008
235       0013                TIXR      T                    B850
240       0015                JLT       WLOOP                3B2FEE
245       0018                RSUB                           4F0000
255                           END       FIRST
          001B      *        =X'05'                         05
```

```
HCOPY  000000001033
DBUFFER000033BUFEND001033LENGTH00002D
RRDREC WRREC
T00000001D1720274B1000003202329000033200 74B1000003F2FEC0320160F2016
T00001D0D0100030F200A4B1000003E2000
T0000300 3454F46
M00000405+RDREC
M00001105+WRREC
M00002405+WRREC
E000000
```

```
HRDREC 000000000002B
RBUFFERLENGTHBUFEND
T0000001DB410B400B44077201FE3201B332FFADB2015A00433200957900000B850
T00001D0E3B2FE9131000004F0000F1000000
M00001805+BUFFER
M00002105+LENGTH
M00002806+BUFEND
M00002806-BUFFER
E
```

```
HWRREC 000000000001C
RLENGTHBUFFER
T00000001CB41077100000E32012332FFA53900000DF2008B8503B2FEE4F000005
M00000305+LENGTH
M00000D05+BUFFER
E
```

Assembler Design Options

- **One pass Assemblers**
  - Main problem is with forward references
    - Instruction operands are symbols that have not been defined in the source program
- **Solution**
  - Define data before they are referenced
  - Place all storage reservation statements at the start of the program rather than at the end
    - But forward reference to labels cannot be eliminated easily
    - Assembler must make special provision for handling forward references
- Many one pass assemblers prohibit forward references
- Two main types of one pass assembler
  - **Load and Go Assembler**: produces object program in memory for immediate execution
  - **Object Program Output**: produces the usual kind of object program for later execution

Load and Go Assembler

- Useful for program development and testing
- Avoids the overhead of writing the object program out and reading it back
- No loader is required
- For a load-and-go assembler, the actual address must be known at assembly time, we can use an absolute program
- The assembler simply generates the object code instructions as it scans the source program directly in memory for immediate execution

Line	Loc	Source statement			Object code
0	1000	COPY	START	1000	
1	1000	EOF	BYTE	C'EOF'	454F46
2	1003	THREE	WORD	3	000003
3	1006	ZERO	WORD	0	000000
4	1009	RETADR	RESW	1	
5	100C	LENGTH	RESW	1	
6	100F	BUFFER	RESB	4096	
9					
10	200F	FIRST	STL	RETADR	141009
15	2012	CLOOP	JSUB	RDREC	48203D

Symbol	Value
LENGTH	100C
RDREC	* → 2013 | 0
THREE	1003
ZERO	1006
EOF	1000
RETADR	1009
BUFFER	100F
CLOOP	2012
FIRST	200F

Memory address	Contents			
1000	454F4600	00030000	00xxxxxx	xxxxxxxx
1010	xxxxxxxx	xxxxxxxx	xxxxxxxx	xxxxxxxx
•				
•				
•				
2000	xxxxxxxx	xxxxxxxx	xxxxxxxx	xxxxxx14
2010	100948--	--		
2020				
•				

Load and Go Assembler

- If the instruction operand is an undefined symbol
  - omit the operand address during instruction translation
  - insert the symbol into SYMTAB, and mark this symbol as undefined
  - the address that refers to the undefined symbol is added to a list of forward references associated with the symbol table entry
  - when the definition for a symbol is encountered, the forward reference list for that symbol is scanned and the proper address for the symbol is then inserted into any instructions previously generated
- At the end of the program
  - any SYMTAB entries that are still marked with * indicate undefined symbols
  - search SYMTAB for the symbol named in the END statement and jump to this location to begin execution

Sample program for a one-pass assembler

[Figure 2.18]

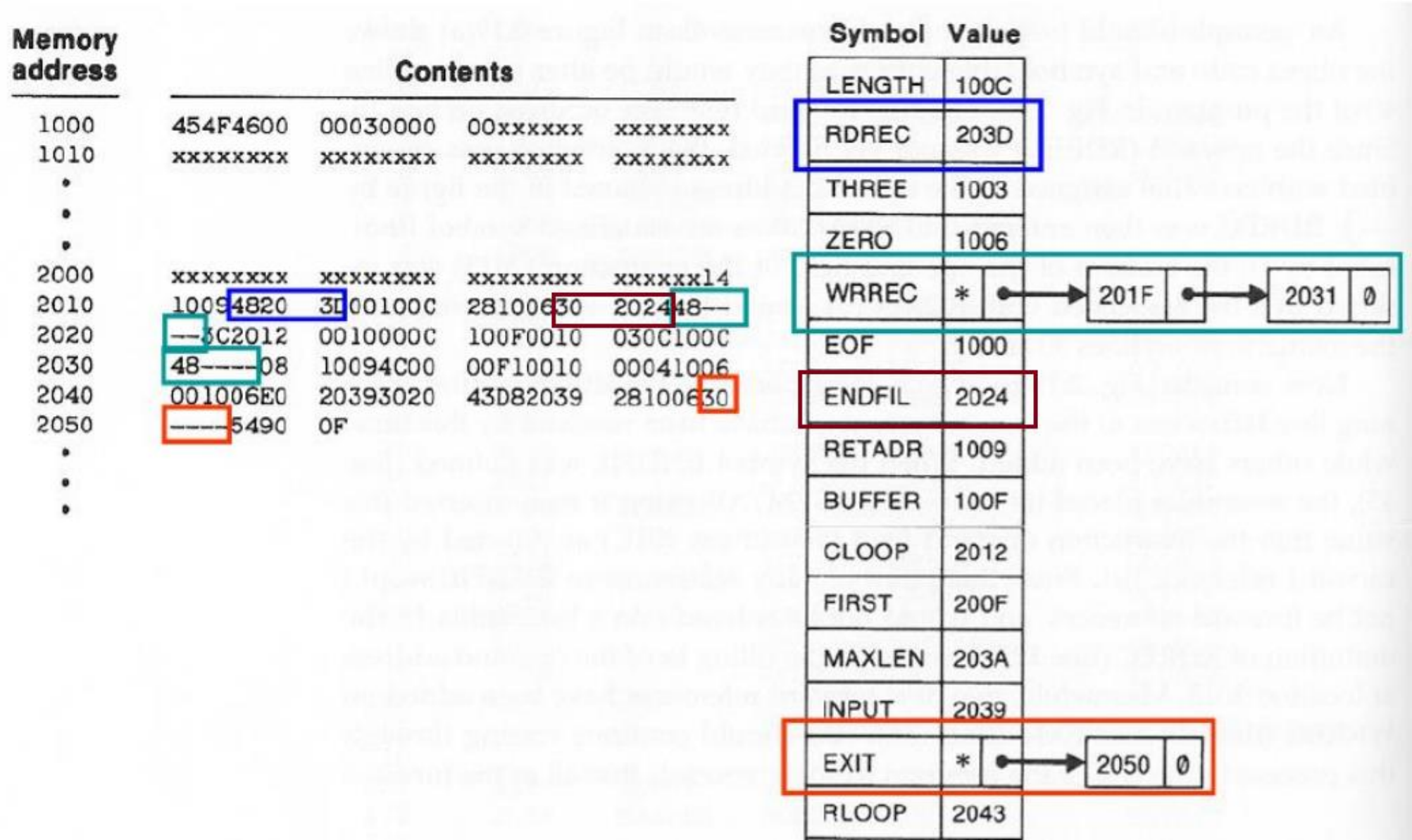
Line	Loc	Source statement			Object code
0	1000	COPY	START	1000	
1	1000	EOF	BYTE	C'EOF'	454F46
2	1003	THREE	WORD	3	000003
3	1006	ZERO	WORD	0	000000
4	1009	RETADR	RESW	1	
5	100C	LENGTH	RESW	1	
6	100F	BUFFER	RESB	4096	
9					
10	200F	FIRST	STL	RETADR	141009
15	2012	CLOOP	JSUB	RDREC	48203D
20	2015		LDA	LENGTH	00100C
25	2018		COMP	ZERO	281006
30	201B		JEQ	ENDFIL	302024
35	201E		JSUB	WRREC	482062
40	2021		J	CLOOP	302012
45	2024	ENDFIL	LDA	EOF	001000
50	2027		STA	BUFFER	0C100F
55	202A		LDA	THREE	001003
60	202D		STA	LENGTH	0C100C
65	2030		JSUB	WRREC	482062
70	2033		LDL	RETADR	081009
75	2036		RSUB		4C0000

Object code in memory and symbol table entries for program in Fig. 2.18 after scanning line 40

Memory address	Contents			
1000	454F4600	00030000	00xxxxxx	xxxxxxxx
1010	xxxxxxxx	xxxxxxxx	xxxxxxxx	xxxxxxxx
•				
•				
•				
2000	xxxxxxxx	xxxxxxxx	xxxxxxxx	xxxxxx14
2010	100948--	--00100C	28100630	-----48--
2020	--3C2012			
•				
•				
•				

Symbol	Value		
LENGTH	100C		
RDREC	*	2013	0
THREE	1003		
ZERO	1006		
WRREC	*	201F	0
EOF	1000		
ENDFIL	*	201C	0
RETADR	1009		
BUFFER	100F		
CLOOP	2012		
FIRST	200F		

When the definition for the symbols RDREC and ENDFILL are encountered, the reference list associated with the symbols is scanned and the address is inserted at proper location.



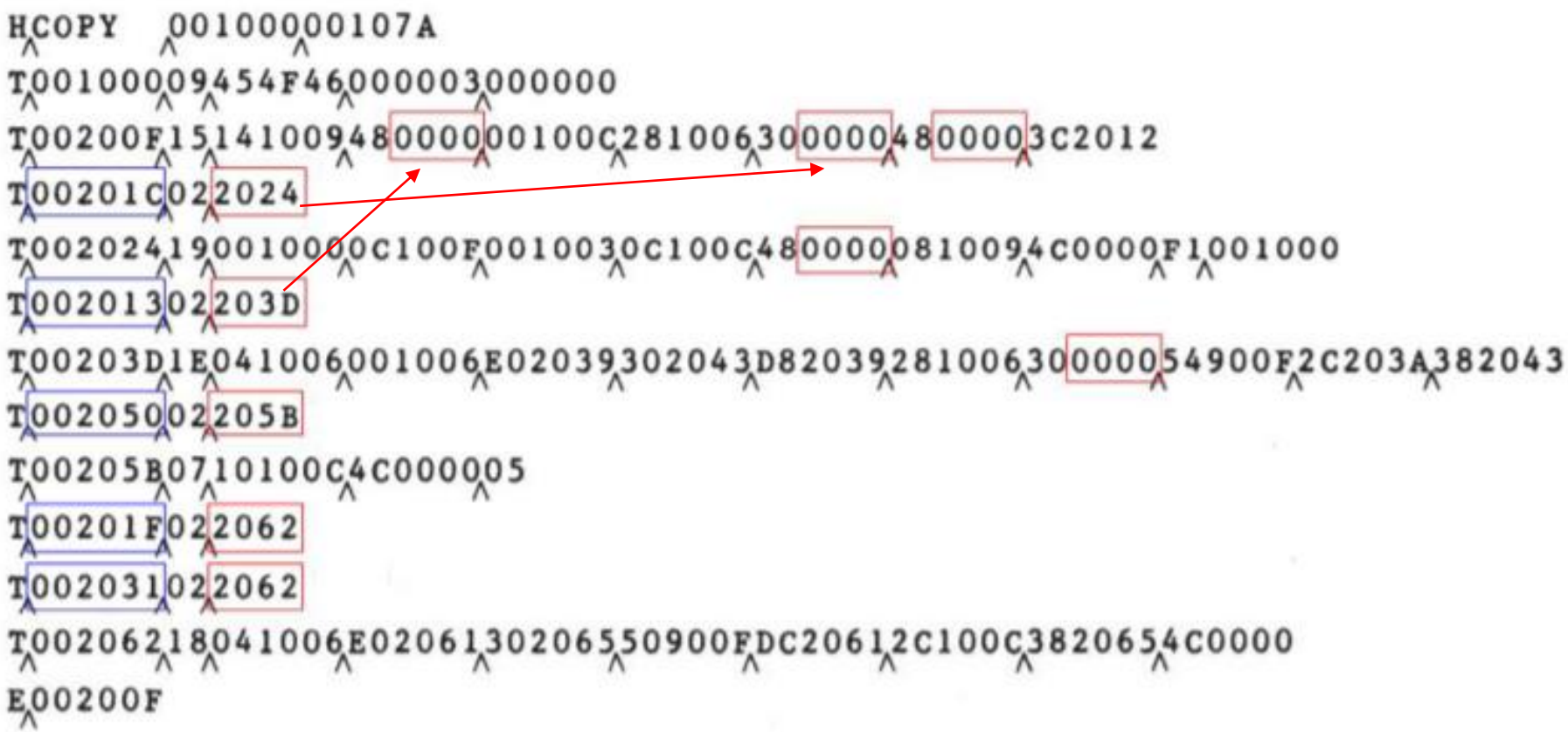
Memory address	Contents			
1000	454F4600	00030000	00xxxxxx	xxxxxxxx
1010	xxxxxxxx	xxxxxxxx	xxxxxxxx	xxxxxxxx
•				
•				
•				
2000	xxxxxxxx	xxxxxxxx	xxxxxxxx	xxxxxx14
2010	10094820	3D00100C	28100630	202448--
2020	--3C2012	00100000	100F0010	030C100C
2030	48----08	10094C00	00F10010	00041006
2040	001006E0	20393020	43D82039	28100630
2050	----5490	0F		
•				
•				
•				

Symbol	Value
LENGTH	100C
RDREC	203D
THREE	1003
ZERO	1006
WRREC	* → 201F → 2031 0
EOF	1000
ENDFIL	2024
RETADR	1009
BUFFER	100F
CLOOP	2012
FIRST	200F
MAXLEN	203A
INPUT	2039
EXIT	* → 2050 0
RLOOP	2043

Object Program Output

- Used on systems when external working-storage devices (for the intermediate file between the two passes) are not available or too slow
- **Solution:**
  - Forward references are entered into lists as before
  - When definition of a symbol is encountered, instructions that made forward references to that symbol will no longer be available in memory for modification
  - They will already have been written out as part of the Text record in the object program
  - The assembler must generate another Text record with the correct operand address
  - When the program is loaded, the address will be inserted into the instruction by the action of the loader
  - The object program records must be kept in their original order when they are presented to the loader

Object program from one-pass assembler for program in Fig. 2.18



Multi Pass Assembler

- In our definition of EQU directive, any symbol defined on the RHS should be defined previously in the source program
- Consider the sequence

```
ALPHA   EQU  BETA
BETA    EQU  DELTA
DELTA   RESW  1
```

- General solution of multi pass assembler is to make as many passes as needed to perform processing of the symbols

Multi Pass Assembler

```
ALPHA  EQU    BETA
BETA   EQU    DELTA
DELTA  RESW   1
```

- During first pass
  - ALPHA cannot be assigned a value since the value of BETA is not yet initialized
  - BETA cannot be assigned a value since DELTA is not yet defined before BETA
  - Location of DELTA is entered into the symbol table
- During the next pass
  - BETA can be assigned the value of DELTA
  - ALPHA cannot be assigned the value of BETA since the statement involving BETA comes after ALPHA
- Thus two passes are not sufficient to solve the problem

Multi Pass Assembler

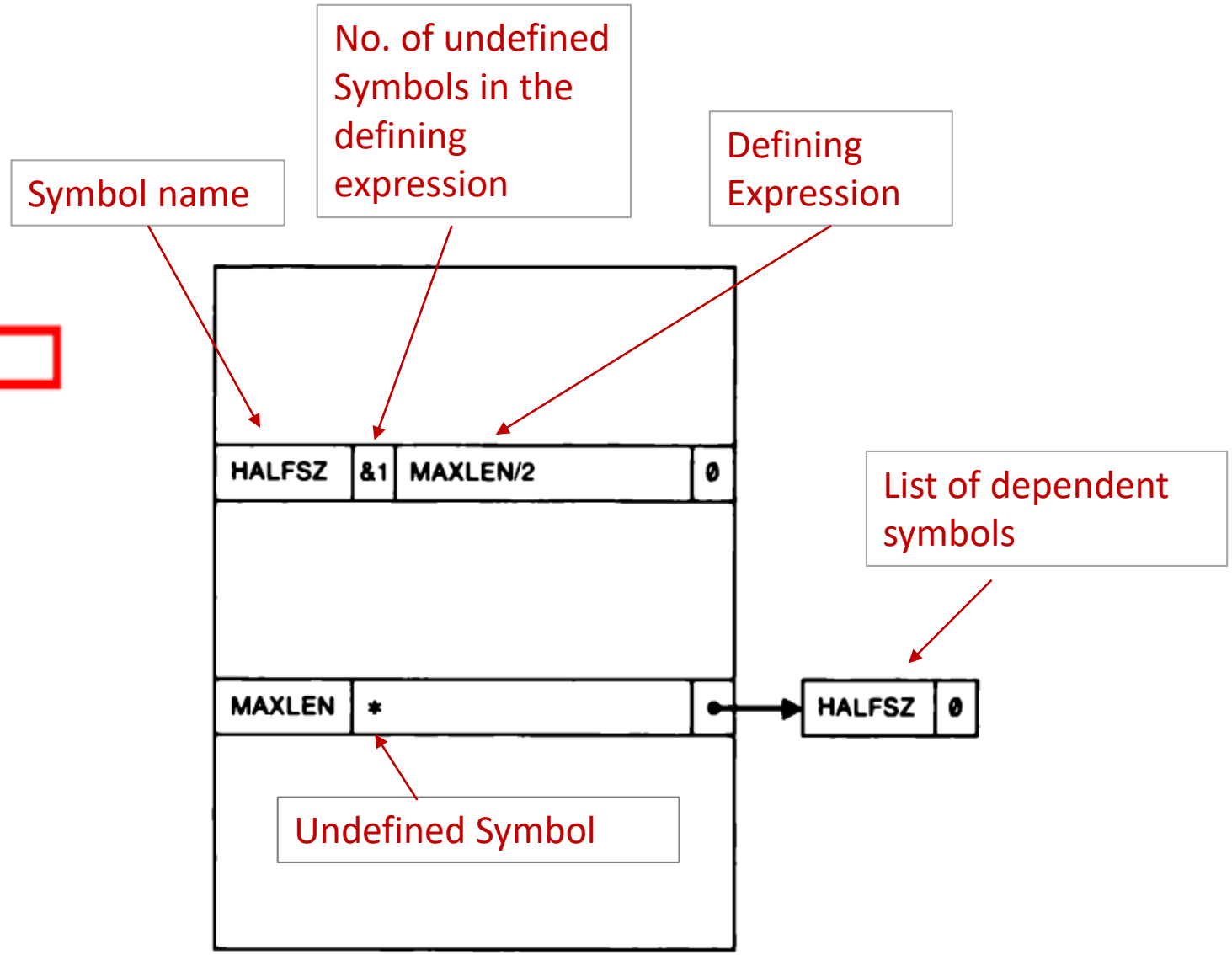
- It is not necessary to make more than two passes over the source program

- **Solution to forward reference problem**
  - Store the symbol definitions that involve forward references in the symbol table
  - The symbol table also indicates which symbols are dependent on the value of others

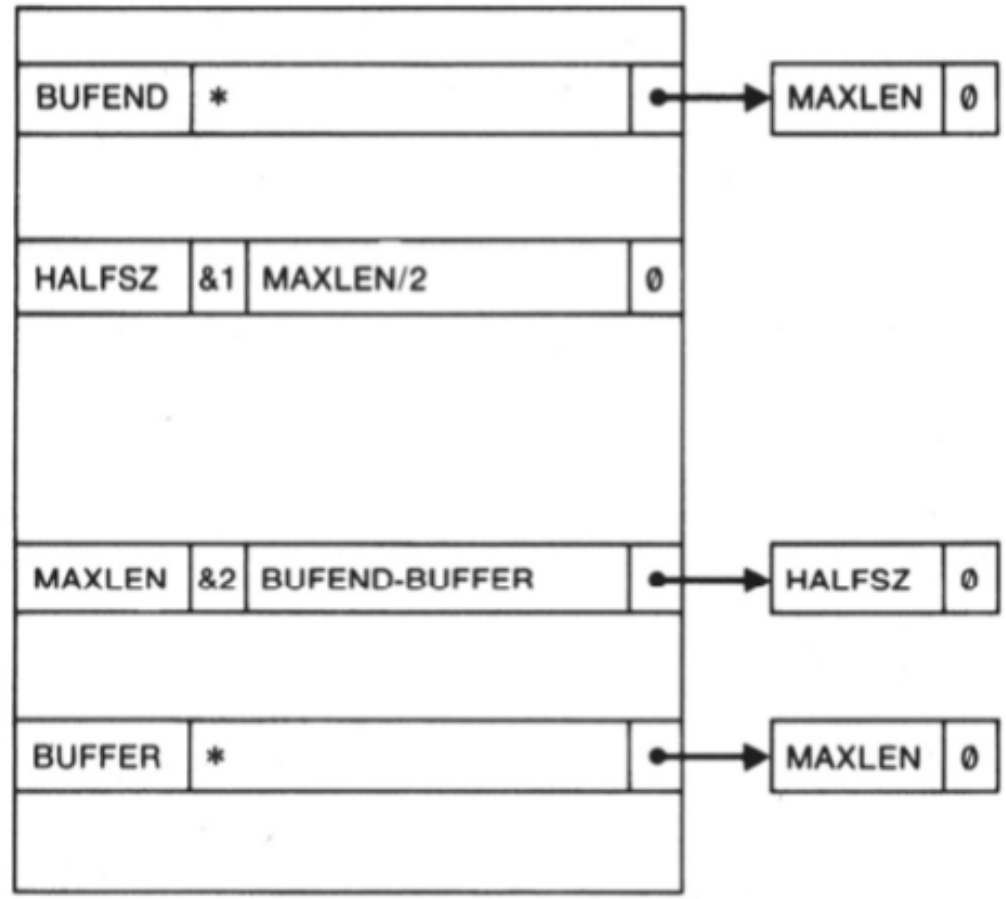
Multi Pass Assembler

- For a forward reference in symbol definition, we store in the SYMTAB:
  - The symbol name
  - The defining expression
  - The number of undefined symbols in the defining expression
  - The undefined symbol (marked as *) associated with a list of symbols depending on this undefined symbol.
- When a symbol is defined, we can recursively evaluate the symbol expressions depending on the newly defined symbol.
- The portions of the program that involve forward references in symbol definition are saved during Pass 1.
- Additional passes through these stored definitions are made as the assembly progresses.
- This process is followed by a normal Pass 2

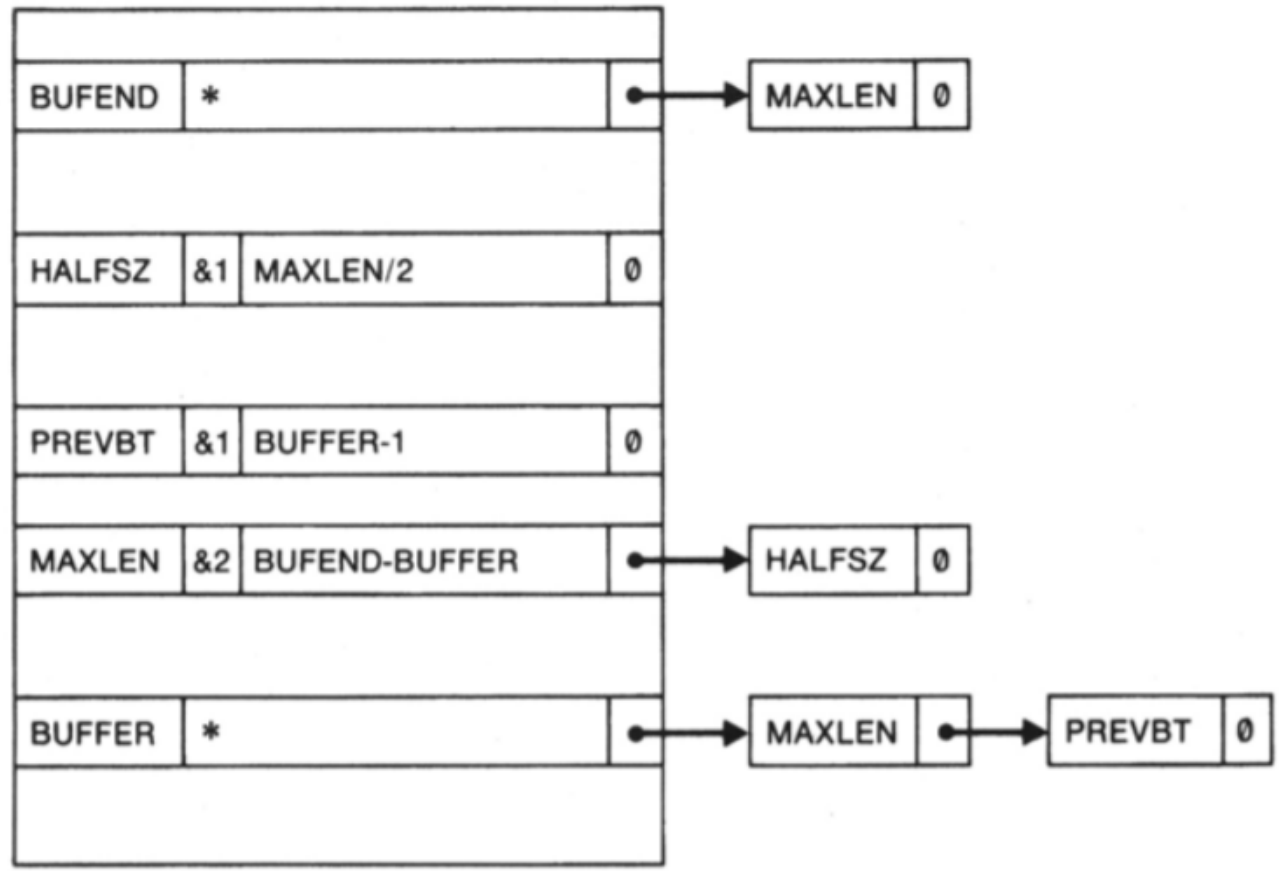
```
1       HALFSZ      EQU     MAXLEN/2
2       MAXLEN      EQU     BUFEND-BUFFER
3       PREVBT      EQU     BUFFER-1
                            .
                            .
                            .
4       BUFFER      RESB    4096
5       BUFEND      EQU     *
```



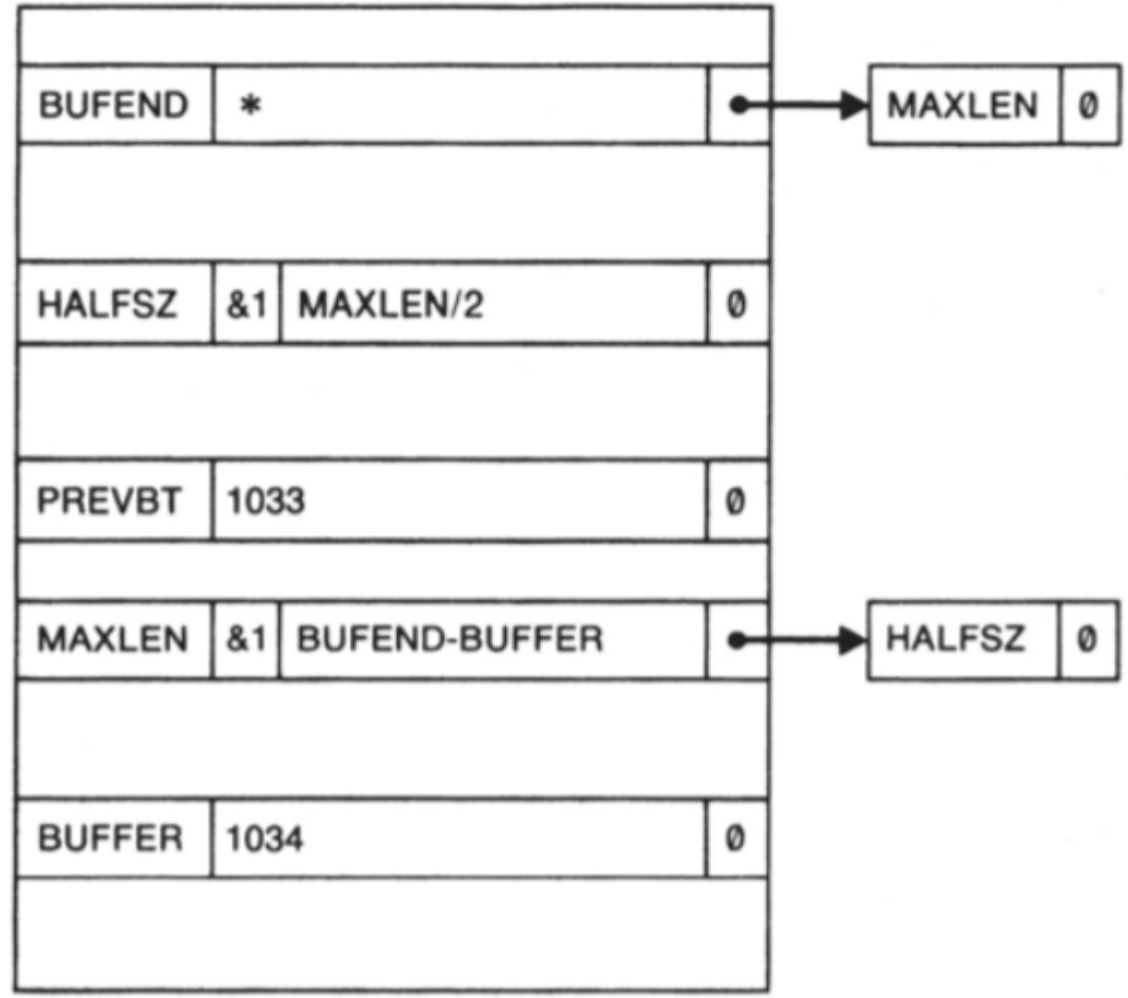
```
1       HALFSZ      EQU     MAXLEN/2
2       MAXLEN      EQU     BUFEND-BUFFER
3       PREVBT      EQU     BUFFER-1
                            .
                            .
                            .
4       BUFFER      RESB    4096
5       BUFEND      EQU     *
```



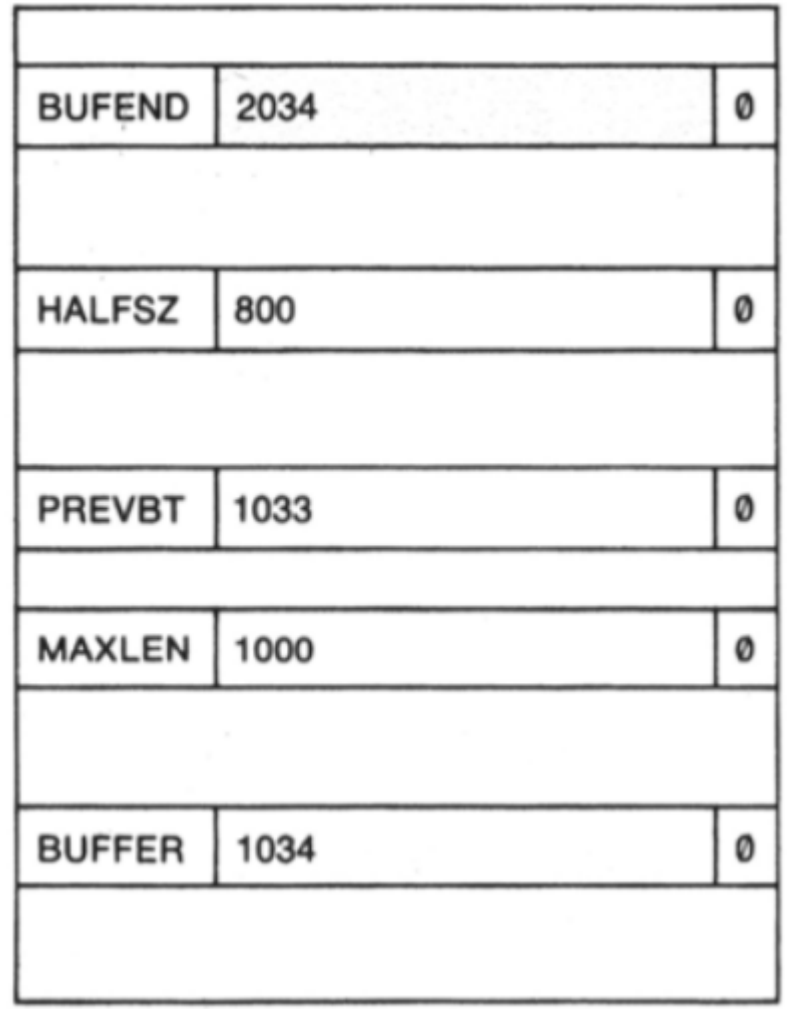
```
1       HALFSZ      EQU     MAXLEN/2
2       MAXLEN      EQU     BUFEND-BUFFER
3       PREVBT      EQU     BUFFER-1
                            .
                            .
                            .
4       BUFFER      RESB    4096
5       BUFEND      EQU     *
```



```
1       HALFSZ      EQU     MAXLEN/2
2       MAXLEN      EQU     BUFEND-BUFFER
3       PREVBT      EQU     BUFFER-1
                            .
                            .
                            .
4       BUFFER      RESB    4096
5       BUFEND      EQU     *
```



```
1       HALFSZ      EQU     MAXLEN/2
2       MAXLEN      EQU     BUFEND-BUFFER
3       PREVBT      EQU     BUFFER-1
                            .
                            .
                            .
4       BUFFER      RESB    4096
5       BUFEND      EQU     *
```



MASM Assembler

Microsoft MACRO Assembler (MASM)

- The **Microsoft Macro Assembler** (**MASM**)
  - is an x86 assembler that uses the Intel syntax for MS-DOS and Microsoft Windows
- SEGMENT
  - MASM assembler language program is written as a collection segments,
  - Each segment is defined as belonging to a particular class, CODE, DATA, CONST, STACK
  - Segments are addressed via registers: CS (code), SS (stack), DS (data), ES, FS, GS
  - similar to program blocks in SIC/XE
- ASSUME directive tells the assembler to associate segment name with a register
  - ASSUME ES:DATASEG2
  - associates ES register with the segment DATASEG2
- Similar to BASE in SIC

MASM Assembler

- JUMP instructions are assembled in two different ways:
  - Near jump: jump to a target in the same code segment
    - Assembled using same code segment register
    - Assembled instruction: 2 or 3 bytes
  - Far jump: jump to a target in a different code segment
    - Assembled using different segment register
    - Assembled instruction: 5 bytes
  - e.g. JMP TARGET
  - By default assembler assumes near jump
  - If the target is in another code segment, the programmer must warn the assembler by writing
    - Warning: JMP FAR PTR TARGET
  - If the jump address is within 128 bytes of the current instruction, the programmer can specify shorter (2 byte) Jump by specifying
    - Warning: JMP SHORT TARGET
  - Pass 1: reserves 3 bytes for jump instruction
  - Phase error: if the target address requires far jump and the programmer does not specify FAR PTR
  - Similarity between far jump and extended format instruction in SIC/XE

Microsoft MASM Assembler

- Length of the assembled instruction depends on the types of operands used
  - Registers, immediate operands (1 to 4 bytes) or memory locations (space required depends on the location of the operand)
- Segments can be written in more than one part.
- If the SEGMENT directive specifies the same name as a previously defined segment, it is considered as a continuation of the previous segment
- All parts of the segments are gathered together during the assembly process
- Similar to Program blocks in SIC/XE
- References between segments are handled by the assembler

MASM Assembler

- External references between separately assembled modules are handled by Linker
  - MASM directive PUBLIC similar to EXTDEF in SIC
  - MASM directive EXTRN similar to EXTREF in SIC
- Produces object code in different formats to allow execution of the program in different operating systems
- Also produces an instruction timing listing that shows the number of clock cycles required to execute each machine instruction