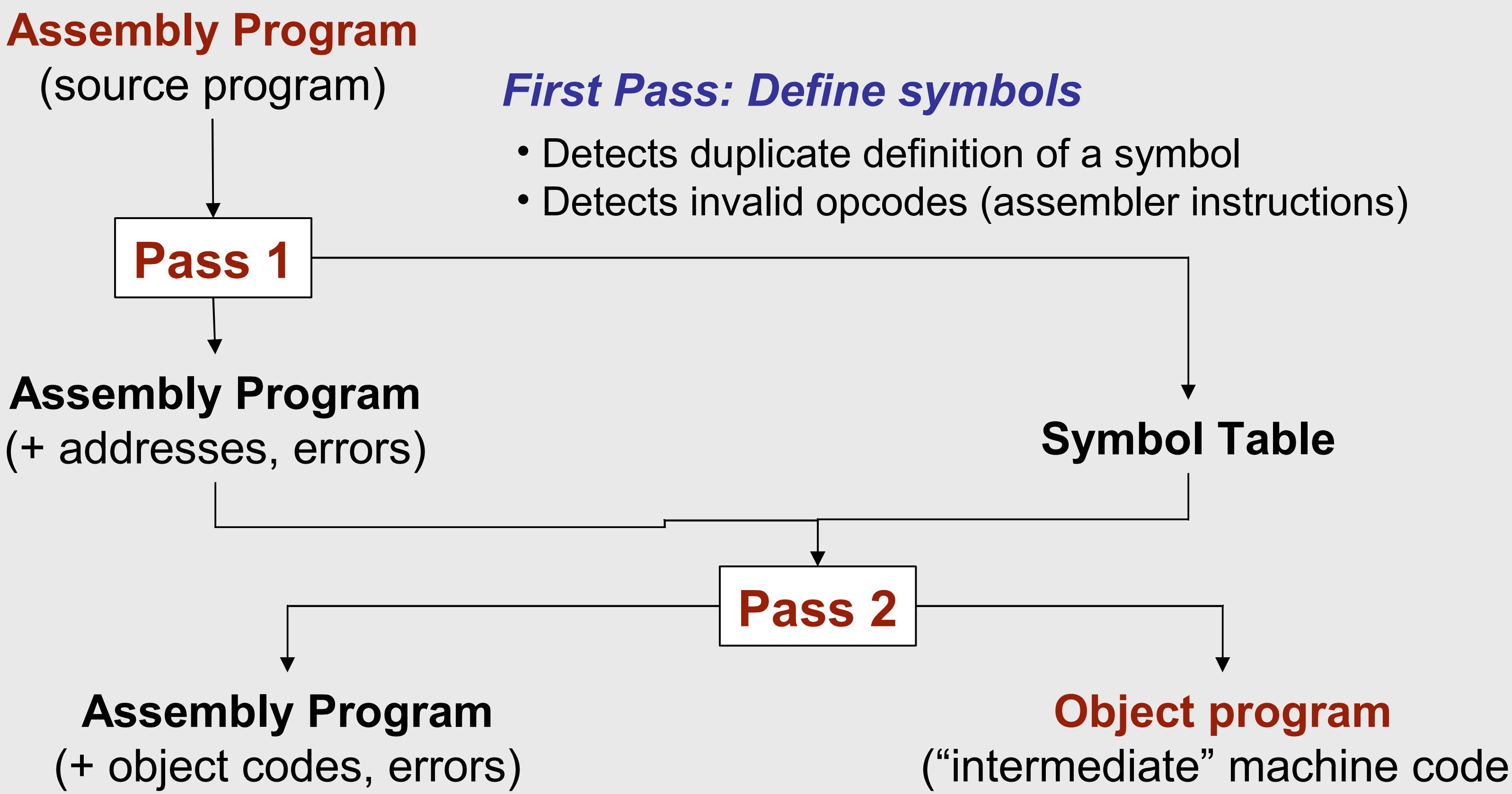
More on Assemblers

COMP 229 (Section PP) Week 2

Prof. Richard Zanibbi
Concordia University
January 16, 2006

Last Week: Two-Pass SIC Assembler



First Pass: Define symbols

- Detects duplicate definition of a symbol
- Detects invalid opcodes (assembler instructions)

Second Pass: Assemble commands

- Detects undefined symbol references

This Week: SIC/XE Assembler, More on Assemblers

Machine-Dependent Features (pp. 52-65)

Instruction Formats and Addressing Modes
Program Relocation

Machine-Independent Features (pp. 66-92)

Literals
Symbol-Defining Statements
Expressions
Program Blocks
Control Sections and Program Linking

Design Options: One-Pass and Multi-Pass (pp. 92-108)

One-Pass Assemblers
Multi-Pass Assemblers
MASM, SPARC assembler examples (assigned reading)

Machine-Dependent Assembler Features

pp. 52-65 (Part I of textbook)

SIC, SIC/XE Architecture

CPU / CPU-X

I/O Devices		
Device (Bin)	Data	Ready
0000 0000	0000 0000	?
…		
1111 1111		?

Registers	
(#) Name	Value (Binary)
(0) A	0000 0000 0000 0000 0000 0000
(1) X	0000 0000 0000 0000 0000 0000
(2) L	0000 0000 0000 0000 0000 0000
(3) B	0000 0000 0000 0000 0000 0000
(4) S	0000 0000 0000 0000 0000 0000
(5) T	0000 0000 0000 0000 0000 0000
(6) F	0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000
(8) PC	0000 0000 0000 0000 0000 0000
(9) SW	0000 0000 0000 0000 0000 0000

Memory	
Address(Hex)	Value (Binary)
0000	0000 0000
. . .	. . .
SIC MAX: 3FFF	0000 0000
. . .	. . .
SIC/XE MAX: 7FFFF	0000 0000

Addressing Modes in SIC, SIC/XE

Target Address (TA)

Address in memory containing the operand for an instruction

Addressing Types: Computing the Target Address Value

Relative: TA computed using register B (base-relative) or register PC (program-counter relative)

Base-relative: displacement field interpreted as unsigned integer

PC-relative: displacement interpreted as 2's complement signed integer

Direct: target address computed without the use of B or PC

Indexed: TA computed using the index register (X) (can be used with relative or direct addressing)

Addressing Types: Use of Data at Target Address

Simple: target address contains operand

Indirect: target address contains address of the operand

Immediate: target address *is* the operand (no lookup)

SIC Instruction Format

8 bits

(24 bits)

opcode	x	address – 15 bits

All SIC opcodes are of the form “XXXX XX00”

SIC/XE Instruction Formats

(displacement)

3
(24 bits)

opc.	n	i	x	b	p	e	disp – 12 bits

1
(8 bits)

opcode

2
(16 bits)

opcode	r1	r2

4
(32 bits)

opc.	n	i	x	b	p	e	address – 20 bits

Upward compatability:
SIC instructions may be represented using SIC/XE Format 3
(n,i bit flags set to zero)

SIC, SIC/XE Flag Bits

Flag (=1)	Effect on Addressing
n	If i=0, indirect addressing (target address contains address of operand)
i	If n=0, immediate addressing (target "address" is the operand)
x	Indexed (+X)
b	Base register-relative (+B)
p	Program counter register-relative (+PC)
e	None: indicates Format 4 SIC/XE instruction

SIC

CO

Figure 2.1: Read a record from an input device, write to output device

Main Loop

```
        COPY      START   1000            COPY FILE FROM INPUT TO OUTPUT
        FIRST     STL     RETADR          SAVE RETURN ADDRESS
        CLOOP     JSUB    RDREC           READ INPUT RECORD
                  LDA     LENGTH          TEST FOR EOF (LENGTH = 0)
                  COMP    ZERO
                  JEQ     ENDFIL          EXIT IF EOF FOUND
                  JSUB    WRREC           WRITE OUTPUT RECORD
                  J       CLOOP           LOOP
        ENDFIL    LDA     EOF             INSERT END OF FILE MARKER
                  STA     BUFFER
                  LDA     THREE           SET LENGTH = 3
                  STA     LENGTH
                  JSUB    WRREC           WRITE EOF
                  LDL     RETADR          GET RETURN ADDRESS
                  RSUB                    RETURN TO CALLER
        EOF       BYTE    C'EOF'
        THREE     WORD    3
        ZERO      WORD    0
        RETADR    RESW    1
        LENGTH    RESW    1               LENGTH OF RECORD
        BUFFER    RESB    4096            4096-BYTE BUFFER AREA
        .
        .         SUBROUTINE TO READ RECORD INTO BUFFER
        .
```

```
RDREC   LDX     ZERO        CLEAR LOOP COUNTER
        LDA     ZERO        CLEAR A TO ZERO
RLOOP   TD      INPUT       TEST INPUT DEVICE
        JEQ     RLOOP       LOOP UNTIL READY
        RD      INPUT       READ CHARACTER INTO REGISTER A
        COMP    ZERO        TEST FOR END OF RECORD (X'00')
        JEQ     EXIT        EXIT LOOP IF EOR
        STCH    BUFFER,X    STORE CHARACTER IN BUFFER
        TIX     MAXLEN      LOOP UNLESS MAX LENGTH
        JLT     RLOOP         HAS BEEN REACHED
EXIT    STX     LENGTH      SAVE RECORD LENGTH
        RSUB                RETURN TO CALLER
INPUT   BYTE    X'F1'       CODE FOR INPUT DEVICE
MAXLEN  WORD    4096
.
.       SUBROUTINE TO WRITE RECORD FROM BUFFER
.
WRREC   LDX     ZERO        CLEAR LOOP COUNTER
WLOOP   TD      OUTPUT      TEST OUTPUT DEVICE
        JEQ     WLOOP       LOOP UNTIL READY
        LDCH    BUFFER,X    GET CHARACTER FROM BUFFER
        WD      OUTPUT      WRITE CHARACTER
        TIX     LENGTH      LOOP UNTIL ALL CHARACTERS
        JLT     WLOOP         HAVE BEEN WRITTEN
        RSUB                RETURN TO CALLER
OUTPUT  BYTE    X'05'       CODE FOR OUTPUT DEVICE
        END     FIRST
```

Line	Source statement		
5	COPY	START	1000
10	FIRST	STL	RETADR
15	CLOOP	JSUB	RDREC
20		LDA	LENGTH
25		COMP	ZERO
30		JEQ	ENDFIL
35		JSUB	WRREC
40		J	CLOOP
45	ENDFIL	LDA	EOF
50		STA	BUFFER
55		LDA	THREE
60		STA	LENGTH
65		JSUB	WRREC
70		LDL	RETADR
75		RSUB	
80	EOF	BYTE	C'EOF'
85	THREE	WORD	3
90	ZERO	WORD	0
95	RETADR	RESW	1
100	LENGTH	RESW	1
105	BUFFER	RESB	4096

Figure 2.1

Line	Source statement		
5	COPY	START	0
10	FIRST	STL	RETADR
12		LDB	#LENGTH
13		BASE	LENGTH
15	CLOOP	+JSUB	RDREC
20		LDA	LENGTH
25		COMP	#0
30		JEQ	ENDFIL
35		+JSUB	WRREC
40		J	CLOOP
45	ENDFIL	LDA	EOF
50		STA	BUFFER
55		LDA	#3
60		STA	LENGTH
65		+JSUB	WRREC
70		J	@RETADR
80	EOF	BYTE	C'EOF'
95	RETADR	RESW	1
100	LENGTH	RESW	1
105	BUFFER	RESB	4096

Figure 2.5

```
125     RDREC     LDX       ZERO
130               LDA       ZERO
135     RLOOP     TD        INPUT
140               JEQ       RLOOP
145               RD        INPUT
150               COMP      ZERO
155               JEQ       EXIT
160               STCH      BUFFER,X
165               TIX       MAXLEN
170               JLT       RLOOP
175     EXIT      STX       LENGTH
180               RSUB
185     INPUT     BYTE      X'F1'
190     MAXLEN    WORD      4096
195     .
200     .         SUBROUTINE TO WRITE
205     .
210     WRREC     LDX       ZERO
215     WLOOP     TD        OUTPUT
220               JEQ       WLOOP
225               LDCH      BUFFER,X
230               WD        OUTPUT
235               TIX       LENGTH
240               JLT       WLOOP
245               RSUB
250     OUTPUT    BYTE      X'05'
255               END       FIRST
```

```
125     RDREC     CLEAR     X
130               CLEAR     A
132               CLEAR     S
133              +LDT       #4096
135     RLOOP     TD        INPUT
140               JEQ       RLOOP
145               RD        INPUT
150               COMPR     A,S
155               JEQ       EXIT
160               STCH      BUFFER,X
165               TIXR      T
170               JLT       RLOOP
175     EXIT      STX       LENGTH
180               RSUB
185     INPUT     BYTE      X'F1'
195     .
200     .         SUBROUTINE TO WRITE
205     .
210     WRREC     CLEAR     X
212               LDT       LENGTH
215     WLOOP     TD        OUTPUT
220               JEQ       WLOOP
225               LDCH      BUFFER,X
230               WD        OUTPUT
235               TIXR      T
240               JLT       WLOOP
245               RSUB
250     OUTPUT    BYTE      X'05'
255               END       FIRST
```

SIC/XE Assembler: Translating Opcode-Only (Format 1) and Register to Register (Format 2) Instructions

Format 1

- Contain no arguments; just translate opcode

Format 2

- Opcode translated, register mnemonics replaced by numeric equivalents (in Pass 2)
- Register mnemonics and numeric values can be added to the Symbol Table (preloaded)

SIC/XE Assembler: Translating Instructions using Registers and Memory (in Formats 3, 4)

Default: Relative Addressing (Format 3)

Most instructions assembled using PC-relative or Base-relative addressing (in Format 3)

Displacement must fit 12-bit displacement field

- Base-relative: displacement (operand) in [0,4095]
- PC-relative: displacement between -2048, +2047

Special Case: Direct Addressing (Format 4)

Format 4 must be used if displacement won't fit in 12 bits

Operand is absolute memory address, stored in 20 bits (enough to reference all memory locations)

SIC/XE Assembler: Addressing for Format 3 Instructions

Format 3

The *assembler* determines whether addressing is PC-relative or Base-relative

Assembler Choice of Addressing for Format 3

1. PC-relative addressing attempted first.
2. If PC-relative fails (i.e. displacement is out of range), then Base-relative addressing attempted
3. If Base-relative addressing fails as well, report an error.

SIC/XE Addressing Modes: Program Counter Relative

Program Counter Relative Addressing

Address represented by offset (displacement) from program counter register value

Notes

PC register contains address of the *next* instruction to be executed

Programmer provides operand address; assembler computes the displacement from PC

Can be used with indexing

Example: Program Counter Relative Addressing

	Address				
10	0000	FIRST	STL	RETADR	17202D

PC: 0003 (next instruction)

RETADR: 0030 (Hex)

Displacement = Operand Address - PC

= 30 - 3 (Decimal: 48 – 3)

= 2D (Decimal: 45)

Bit p is set (first ’2’ in ‘202D’)

Bits n and i also set (changes ’14’ to ’17’)

(for simple addressing)

Example 2: Program Counter Relative Addressing

	Address				
40	0017		J	CLOOP	3F2FEC

PC: 001A (next instruction)

CLOOP: 0006 (Hex)

Displacement = Operand Address - PC

= 6 - 1A (Decimal: 6 – 26)

= -14 (Decimal: -20)

= FEC (2's complement, 12 bits)

Bit p is set ('2' in '2FEC')

Bits n and i also set (changes '3C' to '3F')

SIC/XE Addressing Modes: Base Relative

Base Relative Addressing

Address represented by displacement from base register value

SIC/XE Assembler Directives

(note: these generate no code!)

BASE <symbol>	tells assembler the base register value (symbol address)
NOBASE	indicates base register cannot be used for addressing

Base Relative Addressing, Cont'd

Further on Base-Relative Addressing

- Unlike the program counter, the base register is under user control
- Program must explicitly load values into the base register
- Program must also contain BASE directives, to tell the assembler the value of the base register
- NOBASE indicates when base register may no longer be used for addressing (e.g. if used for temporary storage)
- Can be used with indexing

Line	Source statement		
5	COPY	START	1000
10	FIRST	STL	RETADR
15	CLOOP	JSUB	RDREC
20		LDA	LENGTH
25		COMP	ZERO
30		JEQ	ENDFIL
35		JSUB	WRREC
40		J	CLOOP
45	ENDFIL	LDA	EOF
50		STA	BUFFER
55		LDA	THREE
60		STA	LENGTH
65		JSUB	WRREC
70		LDL	RETADR
75		RSUB	
80	EOF	BYTE	C'EOF'
85	THREE	WORD	3
90	ZERO	WORD	0
95	RETADR	RESW	1
100	LENGTH	RESW	1
105	BUFFER	RESB	4096

Figure 2.1

Line	Source statement		
5	COPY	START	0
10	FIRST	STL	RETADR
12		LDB	#LENGTH
13		BASE	LENGTH
15	CLOOP	+JSUB	RDREC
20		LDA	LENGTH
25		COMP	#0
30		JEQ	ENDFIL
35		+JSUB	WRREC
40		J	CLOOP
45	ENDFIL	LDA	EOF
50		STA	BUFFER
55		LDA	#3
60		STA	LENGTH
65		+JSUB	WRREC
70		J	@RETADR
80	EOF	BYTE	C'EOF'
95	RETADR	RESW	1
100	LENGTH	RESW	1
105	BUFFER	RESB	4096

Figure 2.5

Example: Base Relative Addressing

Address

160	104E		STCH	BUFFER,X	57C003

B: 0033 (Address of LENGTH: given by BASE)

BUFFER: 0036 (Hex)

Displacement = Operand Address - B

= 36 - 33 (Decimal: 54 – 51)

= 3 (Decimal: 3)

Bits b and x are set (first 'C' in 'C003')

(for base relative and indexed addressing)

Bits n and i are set (changes '54' to '57')

Comparison: Program Counter and Base Relative

PC:	20	000A		LDA	LENGTH	032026
BASE:	175	1056	EXIT	STX	LENGTH	134000

Base relative addressing was chosen by the assembler for the second instruction because the required displacement was too large for program counter relative addressing

Recall that PC-relative addressing attempted first for Format 3 instructions

SIC/XE Assembler: Format 4 Instructions

SIC/XE Assembly Syntax

+<opcode> <operand>

Notes

- Direct addressing (operand is address of operand)
- Programmer must indicate Format 4
- If opcode is Format 3 or 4 and "+" absent, relative addressing assumed
- May use indexing with Format 4 ('x' bit)
- Sets the 'e' bit

SIC/XE Addressing Modes: Immediate

SIC/XE Assembly Syntax

<opcode> #<operand>

Notes

- Operand treated as the operand value
- Can be used with relative and direct addressing (Formats 3 and 4)
- Cannot be used with indexing
- Sets the ‘i’ bit

Examples: Immediate Addressing

'e' bit

Format 4:	133	103C		+LDT	#4096	75101000
Format 3:	12	0003		LDB	#LENGTH	69202D

'p' bit

- 4096 too large for 12 bits (F4 instruction)
- F3 instruction loads base register (B) with address of LENGTH: *uses program counter-relative and immediate addressing*

SIC/XE Addressing Modes: Indirect

SIC/XE Assembler Syntax

<opcode> @<operand>

Notes

- Given operand address contains address of the operand
- Also can be used with Format 3 or 4
- Cannot be used with indexing
- Sets the ‘n’ bit

Line	Source statement		
5	COPY	START	1000
10	FIRST	STL	RETADR
15	CLOOP	JSUB	RDREC
20		LDA	LENGTH
25		COMP	ZERO
30		JEQ	ENDFIL
35		JSUB	WRREC
40		J	CLOOP
45	ENDFIL	LDA	EOF
50		STA	BUFFER
55		LDA	THREE
60		STA	LENGTH
65		JSUB	WRREC
70		LDL	RETADR
75		RSUB	
80	EOF	BYTE	C'EOF'
85	THREE	WORD	3
90	ZERO	WORD	0
95	RETADR	RESW	1
100	LENGTH	RESW	1
105	BUFFER	RESB	4096

Figure 2.1

Line	Source statement		
5	COPY	START	0
10	FIRST	STL	RETADR
12		LDB	#LENGTH
13		BASE	LENGTH
15	CLOOP	+JSUB	RDREC
20		LDA	LENGTH
25		COMP	#0
30		JEQ	ENDFIL
35		+JSUB	WRREC
40		J	CLOOP
45	ENDFIL	LDA	EOF
50		STA	BUFFER
55		LDA	#3
60		STA	LENGTH
65		+JSUB	WRREC
70		J	@RETADR
80	EOF	BYTE	C'EOF'
95	RETADR	RESW	1
100	LENGTH	RESW	1
105	BUFFER	RESB	4096

Figure 2.5

Machine-Dependent Features: Program Relocation

Multiprogramming

Running multiple programs (processes) that share system resources (e.g. memory, CPU)

Easier to achieve if programs can be loaded whenever there is room for them (starting address determined at load time)

Absolute Programs

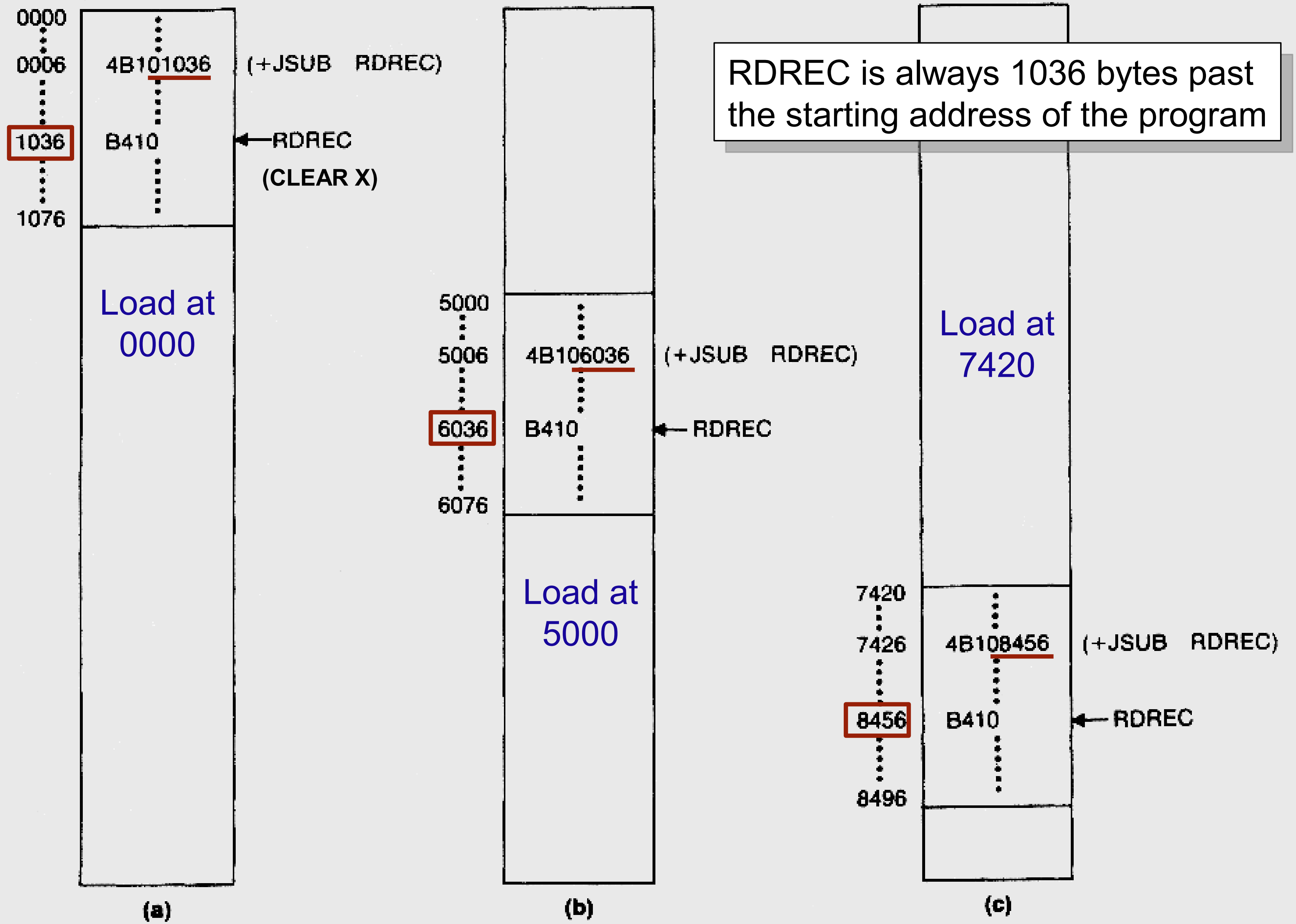
Must be loaded at an exact address to run correctly (e.g. Figure 2.1)

If relocated, could result in incorrect memory accesses into other programs!

Relocatable Programs

Can be loaded at different addresses

Trick: addresses defined as “program start relative”


RDREC is always 1036 bytes past the starting address of the program
0000
0006
1036
1076
4B101036
(+JSUB RDREC)
B410
RDREC
(CLEAR X)
Load at 0000
(a)
5000
5006
6036
6076
4B106036
(+JSUB RDREC)
B410
RDREC
Load at 5000
(b)
Load at 7420
7420
7426
8456
8496
4B108456
(+JSUB RDREC)
B410
RDREC
(c)

Relocation and Relative Addressing

Program Counter Relative Addressing

- Does not need to be modified for relocation
- Distance between instructions within a program is fixed, PC is updated as needed at run time

Base Relative Addressing

- If the base register value is set using a direct address, this address must be altered
- If base is set using PC-relative addressing (as in Figs. 2.5/2.6), no change needed

Line	Source statement		
5	COPY	START	1000
10	FIRST	STL	RETADR
15	CLOOP	JSUB	RDREC
20		LDA	LENGTH
25		COMP	ZERO
30		JEQ	ENDFIL
35		JSUB	WRREC
40		J	CLOOP
45	ENDFIL	LDA	EOF
50		STA	BUFFER
55		LDA	THREE
60		STA	LENGTH
65		JSUB	WRREC
70		LDL	RETADR
75		RSUB	
80	EOF	BYTE	C'EOF'
85	THREE	WORD	3
90	ZERO	WORD	0
95	RETADR	RESW	1
100	LENGTH	RESW	1
105	BUFFER	RESB	4096

Figure 2.1

Line	Source statement		
	(assembled as PC-relative instruction-Fig2.6)		
5	COPY	START	0
10	FIRST	STL	RETADR
12		LDB	#LENGTH
13		BASE	LENGTH
15	CLOOP	+JSUB	RDREC
20		LDA	LENGTH
25		COMP	#0
30		JEQ	ENDFIL
35		+JSUB	WRREC
40		J	CLOOP
45	ENDFIL	LDA	EOF
50		STA	BUFFER
55		LDA	#3
60		STA	LENGTH
65		+JSUB	WRREC
70		J	@RETADR
80	EOF	BYTE	C'EOF'
95	RETADR	RESW	1
100	LENGTH	RESW	1
105	BUFFER	RESB	4096

Figure 2.5

Assembling Relocatable Programs (with SIC/XE Assembler)

1. Use a starting (START) address of 0, to make it easier to compute displacements (e.g. Fig 2.5)
2. Instruct loader in object file to add the starting program address to all direct memory addresses

SIC/XE Object Code Format

New record type indicating where addresses that need to be modified (*have program start address added to them*) are located

Line	Source statement		
5	COPY	START	1000
10	FIRST	STL	RETADR
15	CLOOP	JSUB	RDREC
20		LDA	LENGTH
25		COMP	ZERO
30		JEQ	ENDFIL
35		JSUB	WRREC
40		J	CLOOP
45	ENDFIL	LDA	EOF
50		STA	BUFFER
55		LDA	THREE
60		STA	LENGTH
65		JSUB	WRREC
70		LDL	RETADR
75		RSUB	
80	EOF	BYTE	C'EOF'
85	THREE	WORD	3
90	ZERO	WORD	0
95	RETADR	RESW	1
100	LENGTH	RESW	1
105	BUFFER	RESB	4096

Figure 2.1

Line	Source statement		
5	COPY	START	0
10	FIRST	STL	RETADR
12		LDB	#LENGTH
13		BASE	LENGTH
15	CLOOP	+JSUB	RDREC
20		LDA	LENGTH
25		COMP	#0
30		JEQ	ENDFIL
35		+JSUB	WRREC
40		J	CLOOP
45	ENDFIL	LDA	EOF
50		STA	BUFFER
55		LDA	#3
60		STA	LENGTH
65		+JSUB	WRREC
70		J	@RETADR
80	EOF	BYTE	C'EOF'
95	RETADR	RESW	1
100	LENGTH	RESW	1
105	BUFFER	RESB	4096

Figure 2.5

Object Code Format for SIC Assembler Output		
	Cols	Contents
Header (H)	2-7	Program name
	8-13	(Hex) Starting address of object program
	14-19	(Hex) Length of object program in bytes
Text (T)	2-7	(Hex) Starting address for object code in the record
	8-9	(Hex) Length of object code in bytes
	10-69	(Hex) Object Code: 2 cols per byte
Mod (M)	2-7	(Hex) Starting location of instruction with address to modify (relative to program start, in bytes)
	8-9	(Hex) Length of address to be modified in half-bytes
End (E)	2-7	(Hex) Address of first executable instruction

```
H^COPY  ^000000^001077
T^000000^1D^17202D^69202D^4B101036^032026^290000^332007^4B10105D^3F2FEC^032010
T^00001D^13^0F2016^010003^0F200D^4B10105D^3E2003^454F46
T^001036^1D^B410^B400^B440^75101000^E32019^332FFA^DB2013^A004^332008^57C003^B850
T^001053^1D^3B2FEA^134000^4F0000^F1^B410^774000^E32011^332FFA^53C003^DF2008^B850
T^001070^07^3B2FEF^4F0000^05
M^000007^05
M^000014^05
M^000027^05
E^000000
```

Figure 2.8 Object program corresponding to Fig. 2.6.

Machine-Independent Assembler Features

pp. 66-92 (Part I of textbook)

This Week: SIC/XE Assembler, More on Assemblers

Machine-Dependent Features (pp. 52-65)

Instruction Formats and Addressing Modes
Program Relocation

Machine-Independent Features (pp. 66-92)

Literals
Symbol-Defining Statements
Expressions
Program Blocks
Control Sections and Program Linking

Design Options: One-Pass and Multi-Pass (pp. 92-108)

One-Pass Assemblers
Multi-Pass Assemblers
MASM, SPARC assembler examples (assigned reading)

Machine-Independent Features: Literals

Purpose

- Literals allow a programmer to give the value of a constant operand as part of the instruction using it
- Assembler computes the value and size of the argument, and allocates space to store the value

SIC/XE Assembler Syntax

1. <opcode> =C’<character_string>’
2. <opcode> =H’<hexadecimal_string>’
3. (?) <opcode> =<decimal_number>

Immediate Addressing vs. Literals

Immediate Addressing

Operand value assembled as part of instruction

Literals

The literal (constant) value is stored in a memory address, which is assembled into the instruction

e.g. LDA #LENGTH $\rightarrow$ LDA *(value of LENGTH)*

LDA =X'05' $\rightarrow$ LDA *(address of gen. constant)*

Literal Pools

Storing Literals

- Assembler places literals in one or more pools
- Normally at the end of a program
- In SIC/XE, allow us to avoid WORD/BYTE directives

LTORG ("Literal Origin") Assembler Directive

- Instructs assembler to create a literal pool at the location of the directive
- All unallocated literals at that point in the program are generated (since start/last LTORG directive)
- Can be used to avoid direct addressing, by keeping data close to where it is called (e.g. Fig 2.9/2.10)

Assembler Data Structure Addition: Literal Table (in Pass 1)

Literal Table

Another hash table used in SIC/XE Assembler

Stores literal name (string), value, length, address

In Pass 1 of assembler, literal table is updated while scanning the assembly listing

- New literal value: added to table (name,value,length)
- Existing literal value: ignored

LTORG Statements

In Pass 1, when LTORG seen, all unallocated literals are assigned an address

Location counter updated to reflect #bytes in each literal (making space for the literals)

Literal Table in Pass 2

Literal Table: Pass 2

Literal Table is searched for the address of each literal encountered

Literal values placed at correct locations in the object program (in the literal pools)

Special case: modification record needed if the literal represents a program address

- e.g. if * is used to represent the location counter
- (useful for setting base registers, LDB =*)

Machine-Independent Features: Symbol-Defining Statements

Purpose

- Allow defining symbols other than address labels
- Commonly used to improve readability, define a single symbol for numeric values (constants)
- Also useful for generating labels/offsets for data (e.g. for indexing through tables)

SIC/XE Assembler Directives for Defining Symbols

EQU (“Equate”) Directive

<symbol> EQU <value>

Defines symbol as <value>

ORG (“Origin”) Directive

ORG <value>

During assembly, resets location counter to <value>

For both commands, <value> must be defined using constants and/or previously defined symbols

EQU Example:
Constant and Register Name Def's

Operands

```
+LDT        #4096
```

vs.

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

Register Names (for non-SIC/XE machine, e.g. SPARC)

```
A     EQU     0
X     EQU     1
...
BASE2       EQU     16   (for "R0")
INDEX2      EQU     17   (for "R1")
...
```

(General Purpose Registers)

ORG Example: Setting Location Counter Using ORG

Indexed Addressing of Data Tables

STAB	RESB	1100	
	ORG	STAB	
SYMBOL	RESB	6	*(value: STAB)*
VALUE	RESW	1	*(value: STAB+6)*
FLAGS	RESB	2	*(value: STAB+9)*
	ORG	STAB+1100	*(resets Loc.Ctr)*

STAB (100 entries)

Symbol	Value	Flags

ORG Example Cont'd: Indexed Addressing

Example of Indexed Addressing:

LDA VALUE, X

where the index register (X) contains the table entry, and VALUE provides the offset to the VALUE field in the entry

(X would need to be initialized to STAB and then incremented by 11, the table entry size)

Source statement

```
COPY     START   0         COPY FILE FROM INPUT TO OUTP
FIRST    STL     RETADR    SAVE RETURN ADDRESS
         LDB    #LENGTH    ESTABLISH BASE REGISTER
         BASE    LENGTH
CLOOP   +JSUB    RDREC     READ INPUT RECORD
         LDA     LENGTH    TEST FOR EOF (LENGTH = 0)
         COMP   #0
         JEQ     ENDFIL    EXIT IF EOF FOUND
        +JSUB    WRREC     WRITE OUTPUT RECORD
         J       CLOOP     LOOP
ENDFIL   LDA     EOF       INSERT END OF FILE MARKER
         STA     BUFFER
         LDA    #3         SET LENGTH = 3
         STA     LENGTH
        +JSUB    WRREC     WRITE EOF
         J      @RETADR    RETURN TO CALLER
EOF      BYTE    C'EOF'
RETADR   RESW    1
LENGTH   RESW    1         LENGTH OF RECORD
BUFFER   RESB    4096      4096-BYTE BUFFER AREA
```

Figure 2.5

```
        .
RDREC   CLEAR   X               CLEAR LOOP COUNTER
        CLEAR   A               CLEAR A TO ZERO
        CLEAR   S               CLEAR S TO ZERO
       +LDT    #4096
RLOOP   TD      INPUT           TEST INPUT DEVICE
        JEQ     RLOOP           LOOP UNTIL READY
        RD      INPUT           READ CHARACTER INTO REGISTER
        COMPR   A,S             TEST FOR END OF RECORD (X'00
        JEQ     EXIT            EXIT LOOP IF EOR
        STCH    BUFFER,X        STORE CHARACTER IN BUFFER
        TIXR    T               LOOP UNLESS MAX LENGTH
        JLT     RLOOP             HAS BEEN REACHED
EXIT    STX     LENGTH          SAVE RECORD LENGTH
        RSUB                    RETURN TO CALLER
INPUT   BYTE    X'F1'           CODE FOR INPUT DEVICE
        .
        .       SUBROUTINE TO WRITE RECORD FROM BUFFER
        .
WRREC   CLEAR   X               CLEAR LOOP COUNTER
        LDT     LENGTH
WLOOP   TD      OUTPUT          TEST OUTPUT DEVICE
        JEQ     WLOOP           LOOP UNTIL READY
        LDCH    BUFFER,X        GET CHARACTER FROM BUFFER
        WD      OUTPUT          WRITE CHARACTER
        TIXR    T               LOOP UNTIL ALL CHARACTERS
        JLT     WLOOP             HAVE BEEN WRITTEN
        RSUB                    RETURN TO CALLER
OUTPUT  BYTE    X'05'           CODE FOR OUTPUT DEVICE
        END     FIRST
```

Machine-Independent Features: Expressions

Purpose

Define operand addresses/values using (constrained) arithmetic

Expression Operations

Arithmetic ops (+, -, /, *) ; Division: integer result (usually)

Expression Terms

Constants, user-defined symbols, or special terms

- e.g. * (current value of the location counter, Pass 1)

Relative terms: defined relative to program start address (address labels, location counter references (*)): usu. location in program

Absolute terms: independent of program location (constants)

User-defined symbols (e.g. using EQU) may be relative or absolute, depending on expression defining it's value

Defining Symbol Types in the Symbol Table

Symbol Types

Recorded in the Symbol Table using flags

- relative (R) , or
- absolute (A)

Symbol	*Type*	*Value*
RETADR	R	0030
BUFFER	R	0036
BUFEND	R	1036
MAXLEN	A	1000

Used for:

– Checking term types

– Generating modification records for relative values

Absolute and Relative Expressions

Absolute Expression

- Value is independent of the program location
- If relative terms used, must be able to be put in pairs with opposite signs (equal number of +, - signs)

```
107   MAXLEN   EQU    BUFEND – BUFFER
```

Relative Expression

- Value can be represented as (S + r), where S is the start address and r is an offset
- If relative terms used, contains an odd number of relative terms, with one more +ve term than –ve terms

Restrictions on Expressions

Other Restrictions on Relative Expressions

Relative terms cannot be used in multiplication or division operations

Restrictions insure expressions describe valid location within a program after relocation (S + r)

Invalid expressions:

```
3 * BUFFER, BUFEND + BUFFER, 100 - BUFFER
```

Invalid Expressions

Expressions that do not meet absolute or relative restrictions are flagged as errors by the assembler

Machine-Independent Features: Program Blocks

Purpose

The best order of instructions for a machine are not necessarily easy to read

Program blocks provide a mechanism for program sections to be easily reordered in an object program

SIC/XE Assembler Directive

USE <block_name>

- used to start or continue a (named) program block
- “empty” block_name is the first unlabelled block encountered

Line **Source statement**

```
        COPY     START   0                 COPY FILE FROM INPUT TO OUTPUT
        FIRST    STL     RETADR            SAVE RETURN ADDRESS
        CLOOP    JSUB    RDREC             READ INPUT RECORD
                 LDA     LENGTH            TEST FOR EOF (LENGTH = 0)
                 COMP    #0
                 JEQ     ENDFIL            EXIT IF EOF FOUND
                 JSUB    WRREC             WRITE OUTPUT RECORD
                 J       CLOOP             LOOP
        ENDFIL   LDA     =C'EOF'           INSERT END OF FILE MARKER
                 STA     BUFFER
                 LDA     #3                SET LENGTH = 3
                 STA     LENGTH
                 JSUB    WRREC             WRITE EOF
                 J       @RETADR           RETURN TO CALLER
                 USE     CDATA
        RETADR   RESW    1
        LENGTH   RESW    1                 LENGTH OF RECORD
                 USE     CBLKS
        BUFFER   RESB    4096              4096-BYTE BUFFER AREA
        BUFEND   EQU     *                 FIRST LOCATION AFTER BUFFER
        MAXLEN   EQU     BUFEND-BUFFER     MAXIMUM RECORD LENGTH
        .
        .        SUBROUTINE TO READ RECORD INTO BUFFER
```

Figure 2.11: Program with Multiple Blocks

```
        .
                USE
RDREC           CLEAR       X                    CLEAR LOOP COUNTER
                CLEAR       A                    CLEAR A TO ZERO
                CLEAR       S                    CLEAR S TO ZERO
               +LDT        #MAXLEN
RLOOP           TD          INPUT                TEST INPUT DEVICE
                JEQ         RLOOP                LOOP UNTIL READY
                RD          INPUT                READ CHARACTER INTO REGISTER A
                COMPR       A,S                  TEST FOR END OF RECORD (X'00')
                JEQ         EXIT                 EXIT LOOP IF EOR
                STCH        BUFFER,X             STORE CHARACTER IN BUFFER
                TIXR        T                    LOOP UNLESS MAX LENGTH
                JLT         RLOOP                  HAS BEEN REACHED
EXIT            STX         LENGTH               SAVE RECORD LENGTH
                RSUB                             RETURN TO CALLER
                USE         CDATA
INPUT           BYTE        X'F1'                CODE FOR INPUT DEVICE
        .
        .       SUBROUTINE TO WRITE RECORD FROM BUFFER
        .
                USE
WRREC           CLEAR       X                    CLEAR LOOP COUNTER
                LDT         LENGTH
WLOOP           TD         =X'05'                TEST OUTPUT DEVICE
                JEQ         WLOOP                LOOP UNTIL READY
                LDCH        BUFFER,X             GET CHARACTER FROM BUFFER
                WD         =X'05'                WRITE CHARACTER
                TIXR        T                    LOOP UNTIL ALL CHARACTERS
                JLT         WLOOP                  HAVE BEEN WRITTEN
                RSUB                             RETURN TO CALLER
                USE         CDATA
                LTORG
                END         FIRST
```

Assigning Addresses to Blocks

Generating Addresses with Blocks

Blocks merged and placed in order they appear in the program

Pass 1 Modification

Multiple Location Counters

- Separate LC for each block; the “blank” program LC is defined by default
- Appropriate LC is incremented while determining addresses
- End of Pass 1, each LC gives lengths of a program block; block addresses (symbols) defined

Symbol Table

- Modified again, this time we add a block name/number
- Symbol addresses recorded as the address relative to the start of block containing the symbol (in a “separate table”)

Pass 2 Modification

Symbol addresses defined by adding stored address to start of associated block address

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	
110			.			
115			.	SUBROUTINE TO READ RECORD INTO BUFFER		
120						

Absolute symbol

Figure 2.12: Program with Multiple Blocks (blocks, object code shown)

```
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
195            .
200            .       SUBROUTINE TO WRITE RECORD FROM BUFFER
205            .
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
```

Benefit of Program Blocks

Simplifies Addressing, Data Organization

- Allows us to use more relative addressing, as large data areas (e.g. our BUFFER) can be moved into a separate block (e.g. at end of pr.)
- Reduces need for base-relative addr. (LDB, BASE directives removed in Fig. 2.11/2.12)
- LTORG directives along with blocks allow us to locate literals more flexibly (e.g. before large data areas)

Generating Object Code

Block Order

- Translation occurs using the assembly program as given
- Write out records to locate blocks correctly in object programs

USE Triggers Record Writing

- In Pass 2, USE directives force the current Text Record to be written, and start a new one at the appropriate address

```
H^COPY  ^000000^001071
T^000000^1E^172063^4B2021^032060^290000^332006^4B203B^3F2FEE^032055^0F2056^010003
T^00001E^09^0F2048^4B2029^3E203F
T^000027^1D^B410^B400^B440^75101000^E32038^332FFA^DB2032^A004^332008^57A02F^B850
T^000044^09^3B2FEA^13201F^4F0000
T^00006C^01^F1
T^00004D^19^B410^772017^E3201B^332FFA^53A016^DF2012^B850^3B2FEF^4F0000
T^00006D^04^454F46^05
E^000000
```

Default (Parts 1 and 2)

CDATA (2)

Default (Part 3)

CDATA (3)

Figure 2.13 Object program corresponding to Fig. 2.11.

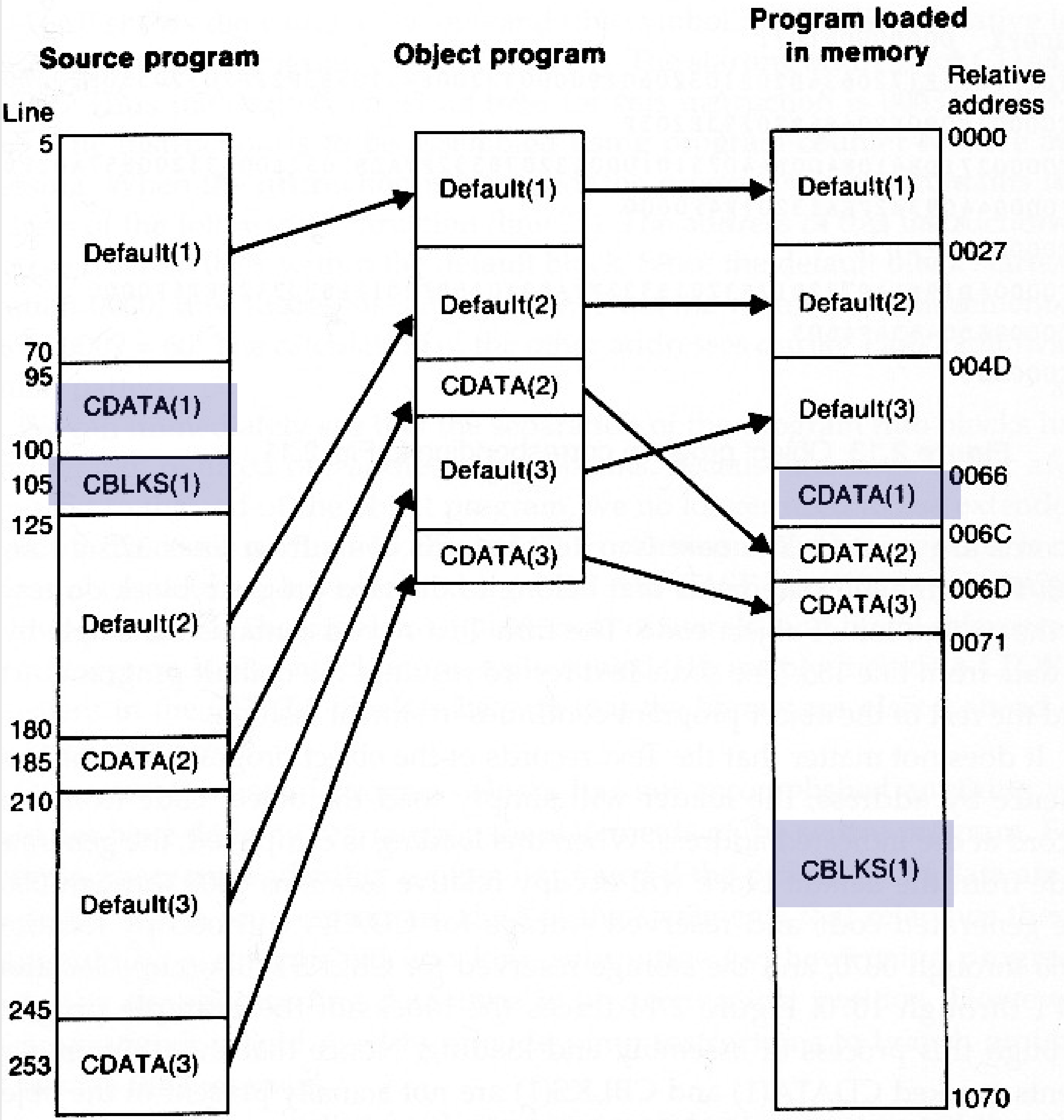


Figure 2.14 Program blocks from Fig. 2.11 traced through the assembly and loading processes.

Machine-Independent Features: Control Sections and Program Linking

Control Sections

Allow sections of an assembly programs to be written out as separate objects

Definition and reference of symbols in separate control sections (*external references*)

SIC/XE Assembler Directives

<symbol> CSECT *(symbol is name of Section)*

EXTDEF <symbol1,...,symboln>

EXTREF <symbol1,....,symboln>

```
Line      Source statement

  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

109   RDREC     CSECT
110   .
115   .         SUBROUTINE TO READ RECORD INTO BUFFER
120
```

```
         .
         EXTREF   BUFFER,LENGTH,BUFEND
         CLEAR    X                     CLEAR LOOP COUNTER
         CLEAR    A                     CLEAR A TO ZERO
         CLEAR    S                     CLEAR S TO ZERO
         LDT      MAXLEN
RLOOP    TD       INPUT                 TEST INPUT DEVICE
         JEQ      RLOOP                 LOOP UNTIL READY
         RD       INPUT                 READ CHARACTER INTO REGISTER A
         COMPR    A,S                   TEST FOR END OF RECORD (X'00')
         JEQ      EXIT                  EXIT LOOP IF EOR
        +STCH     BUFFER,X              STORE CHARACTER IN BUFFER
         TIXR     T                     LOOP UNLESS MAX LENGTH
         JLT      RLOOP                   HAS BEEN REACHED
EXIT    +STX      LENGTH                SAVE RECORD LENGTH
         RSUB                           RETURN TO CALLER
INPUT    BYTE     X'F1'                 CODE FOR INPUT DEVICE
MAXLEN   WORD     BUFEND-BUFFER

WRREC    CSECT
         .
         .        SUBROUTINE TO WRITE RECORD FROM BUFFER
         .
         EXTREF   LENGTH,BUFFER
         CLEAR    X                     CLEAR LOOP COUNTER
        +LDT      LENGTH
WLOOP    TD      =X'05'                 TEST OUTPUT DEVICE
         JEQ      WLOOP                 LOOP UNTIL READY
        +LDCH     BUFFER,X              GET CHARACTER FROM BUFFER
         WD      =X'05'                 WRITE CHARACTER
         TIXR     T                     LOOP UNTIL ALL CHARACTERS
         JLT      WLOOP                   HAVE BEEN WRITTEN
         RSUB                           RETURN TO CALLER
         END      FIRST
```

External Definitions and References

Symbols

Symbols defined in one section cannot be used directly by another (“external reference”)

Sections can define symbols for other sections using EXTDEF

Similarly, Sections list external references using EXTREF

**Section Names automatically defined as external symbols

External Symbols, Cont'd

External References

- Require Format 4 instruction to be used (for absolute address: cannot guarantee relative addressing)
- Value of operand initially set to zero
- For expressions using undefined symbols, need to generate new records that "do the math"
- New record types required for SIC/XE Object Program format

Object Code Format for SIC Assembler Output

(New Record Types)

	Cols	Contents
Define (D)	2-7	Name of external symbol defined in this section
	8-13	(Hex) Relative address of symbol within current control section
	14-73	(Repeat cols 2-13 for additional external defs)
Refer (R)	2-7	Name of external symbol references in this section
	8-73	(Repeat cols 2-7 for additional external references)
Mod (Revised) (M)	2-7	(Hex) Starting address of field to modify, relative to beginning of control section
	8-9	(Hex) Length of field to be modified in half-bytes
	10	Modification flag (+ or -)
	11-16	External symbol to add to/subtract from field

```
HCOPY  ^000000^001033
DBUFFER^000033^BUFEND^001033^LENGTH^00002D
RRDREC ^WRREC
T^000000^1D^172027^4B100000^032023^290000^332007^4B100000^3F2FEC^032016^0F2016
T^00001D^0D^010003^0F200A^4B100000^3E2000
T^000030^03^454F46
M^000004^05^+RDREC
M^000011^05^+WRREC
M^000024^05^+WRREC
E^000000


HRDREC ^000000^00002B
RBUFFER^LENGTH^BUFEND
T^000000^1D^B410^B400^B440^77201F^E3201B^332FFA^DB2015^A004^332009^57900000^B850
T^00001D^0E^3B2FE9^13100000^4F0000^F1^000000
M^000018^05^+BUFFER
M^000021^05^+LENGTH
M^000028^06^+BUFEND
M^000028^06^-BUFFER
E
```

```
HWRREC 00000000001C
RLENGTHBUFFER
T0000001CB410771000000E32012332FFA53900000DF2008B8503B2FEE4F000005
M00000305+LENGTH
M00000D05+BUFFER
E
```

Figure 2.17 Object program corresponding to Fig. 2.15.

Design Options: One-Pass and Multi-Pass Assemblers

pp. 92-102 (Part I of textbook)

Read at home:

(MASM, SPARC assembler examples)

pp. 103-108

Design Options: One-Pass Assemblers

Assumptions for our Discussion

- Considering only absolute programs (e.g. for SIC, Fig 2.1)
- Assume that instructions contain actual (not relative) operand addresses (again, as for SIC assembler)

Forward References

- “Main problem” for one pass assembly
- Handled in SIC/XE Assembler by using two passes
- Alternative: require data to be reserved before use, but allow forward references to lables of instructions

Data area: before use

Forward References

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
110		.			
115		.	SUBROUTINE TO READ RECORD INTO BUFFER		

Fig 2.18: One-Pass Assembly Version of Fig 2.1/2.2

Two Types of One Pass Assembler

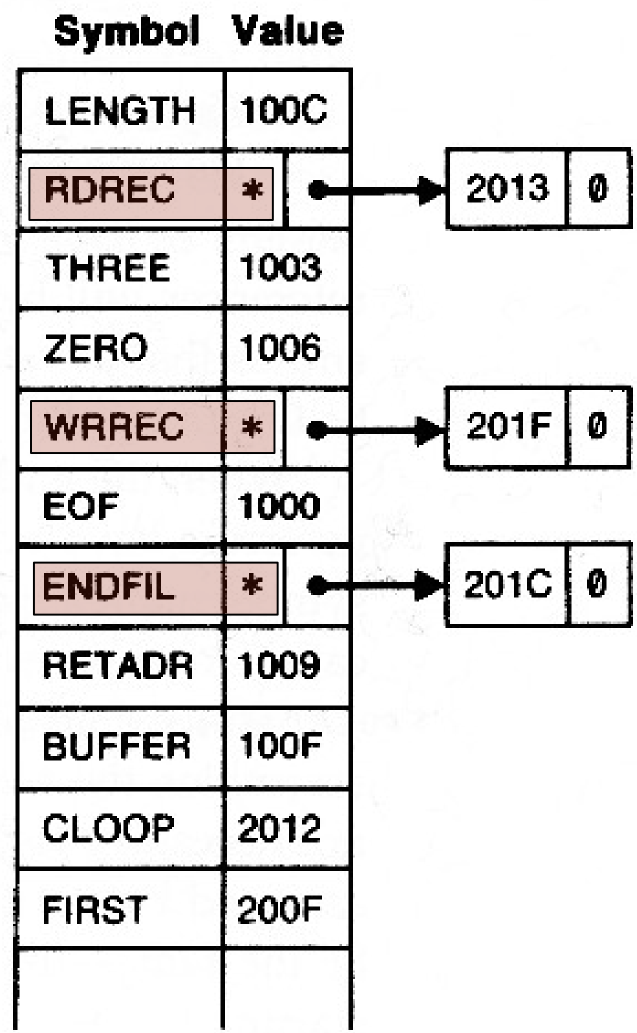
"Load-and-Go"

- Object code produced directly in memory, executed immediately
- Uses modified symbol table (records locations of forward references)
- Place holders used for operands defined by forward references
- When new symbols resolve forward references, operand addresses updated in memory

Object Program Output

- Object code saved in a file, but no intermediate file used for assembly
- Same use of symbol table as "load-and-go" assemblers
- Place holders (default value) saved in text records for forward references
- When symbol is defined, additional text record is produced for each forward reference to the symbol

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			
•				
•				
•				



“Load-and-Go”

Figure 2.19(a) Object code in memory and symbol table entries for the program in Fig. 2.18 after scanning line 40.

emory ddress	Contents			
1000	454F4600	00030000	00xxxxxx	xxxxxxxx
1010	xxxxxxxx	xxxxxxxx	xxxxxxxx	xxxxxxxx
•				
•				
•				
2000	xxxxxxxx	xxxxxxxx	xxxxxxxx	xxxxxx14
2010	10094820	3D00100C	28100630	202448--
2020	--3C2012	0010000C	100F0010	030C100C
2030	48----08	10094C00	00F10010	00041006
2040	001006E0	20393020	43D82039	28100630
2050	----5490	0F		
•				
•				
•				

"Load-and-Go"

If any undefined (*) symbols remain at the end of scanning the program, there are undefined symbols (error)

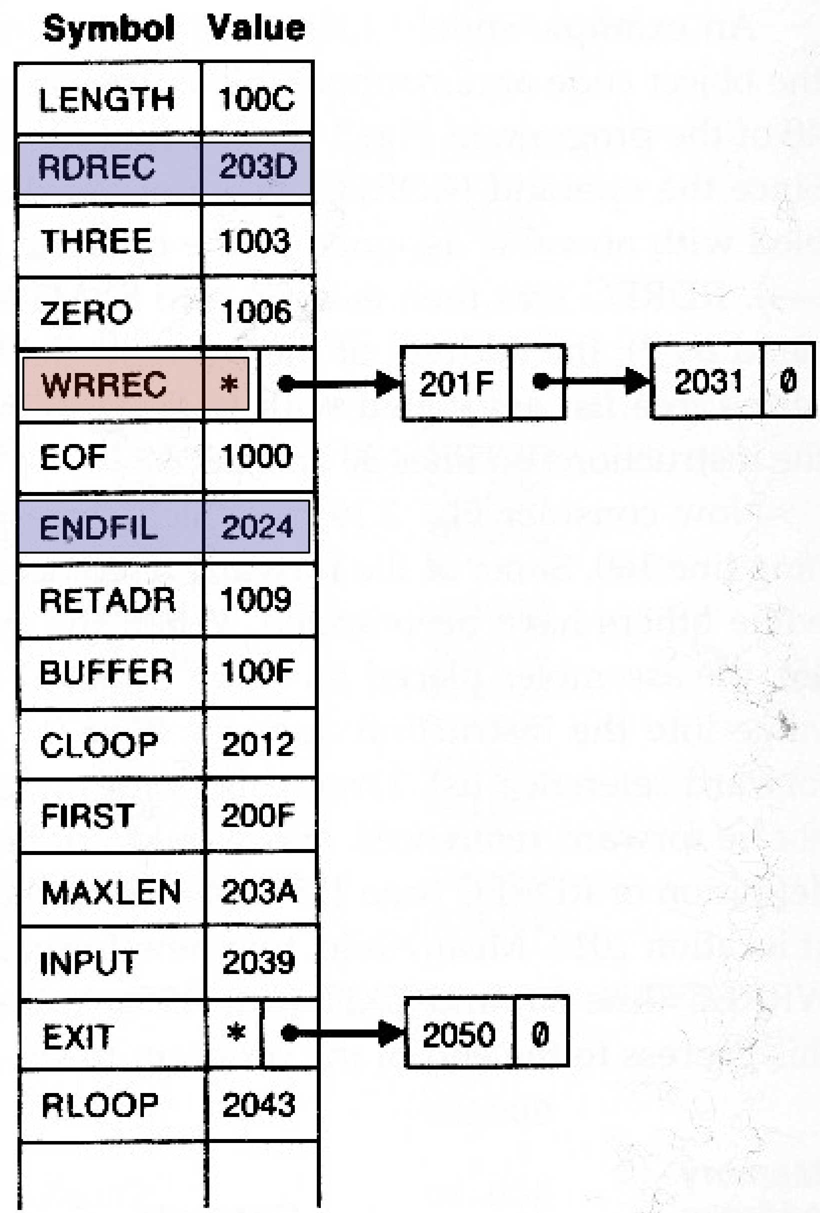


Figure 2.19(b) Object code in memory and symbol table entries for the program in Fig. 2.18 after scanning line 160.

Example: Object Program for Fig. 2.18 (One-Pass Assembler Producing Object Code)

```
HCOPY  00100000107A
T00100009454F460000030000000
T00200F1514100948000000100C2810063000004800003C2012
T00201C022024
T00202419001000C100F0010030C100C4800000810094C0000F1001000
T00201302203D
T00203D1E041006001006E02039302043D8203928100630000054900F2C203A382043
T00205002205B
T00205B0710100C4C000005
T00201F022062
T002031022062
T00206218041006E0206130206550900FDC20612C100C3820654C0000
E00200F
```

Forward Ref. Values: 0

(ENDFIL)

(RDREC)

(EXIT)

(WRREC – 2 forward references, two text records)

Figure 2.20 Object program from one-pass assembler for program in Fig. 2.18.

Design Options: Multi-Pass Assemblers

Pragmatics of Forward References

Restricting forward references (esp. for data) can make it easier for programmers to read each others' programs

Resolving Arbitrary Forward References

Could pass over the program as many times as necessary to define symbols

Use modified symbol table

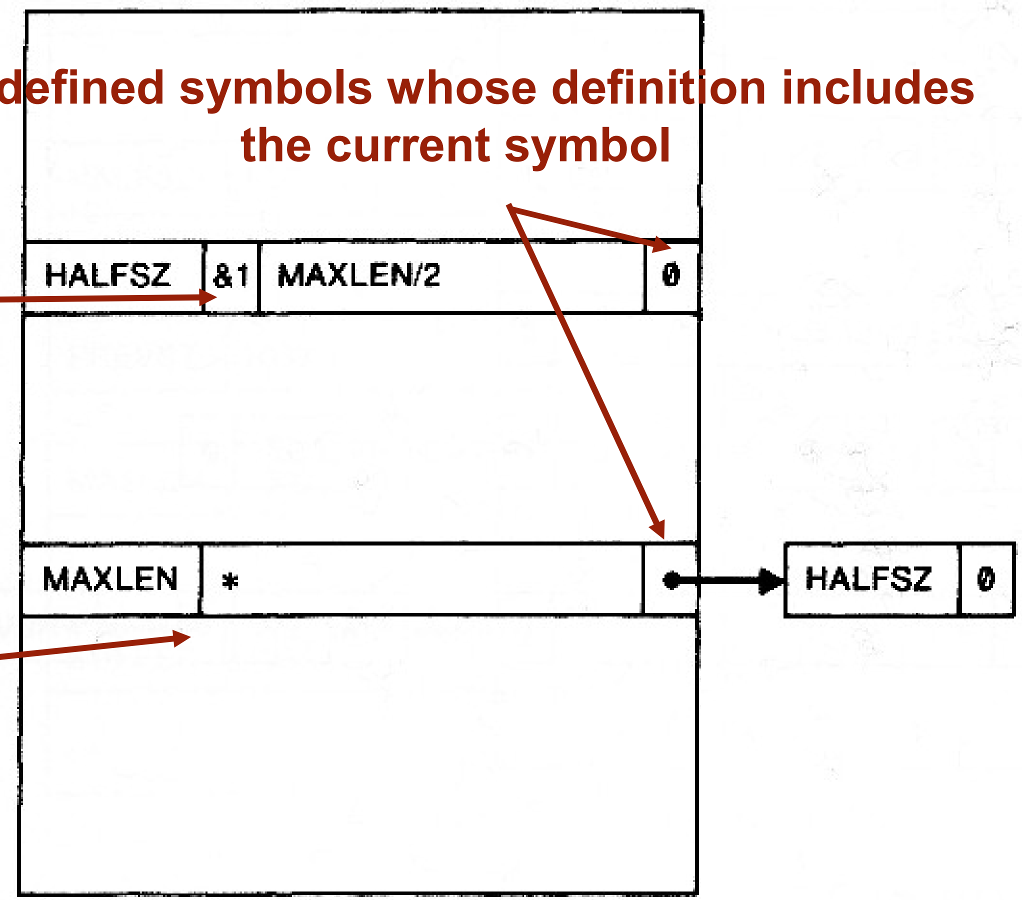
- Record forward references using lists, as we did for one-pass assemblers
- Also record which symbols depend on others for their definition
- This makes only two passes necessary

Example

Resolving forward references in user-defined symbols created with the "EQU" directive

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

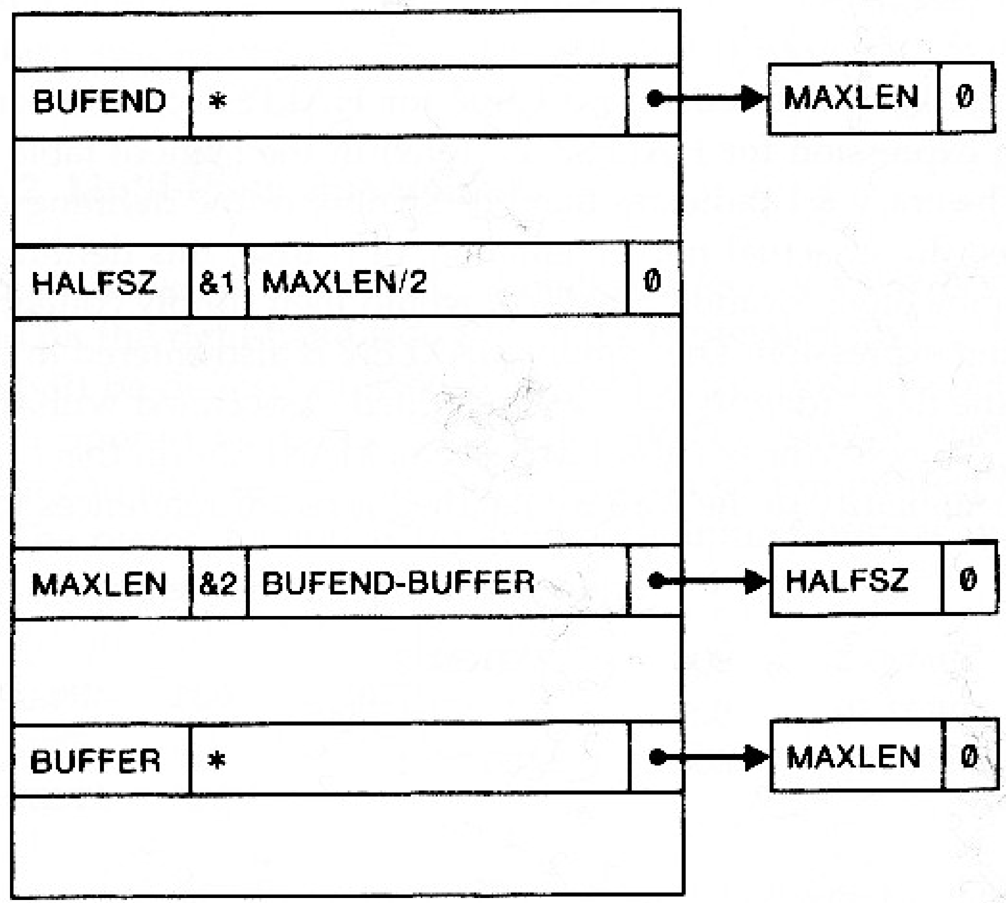
(a)



(b)

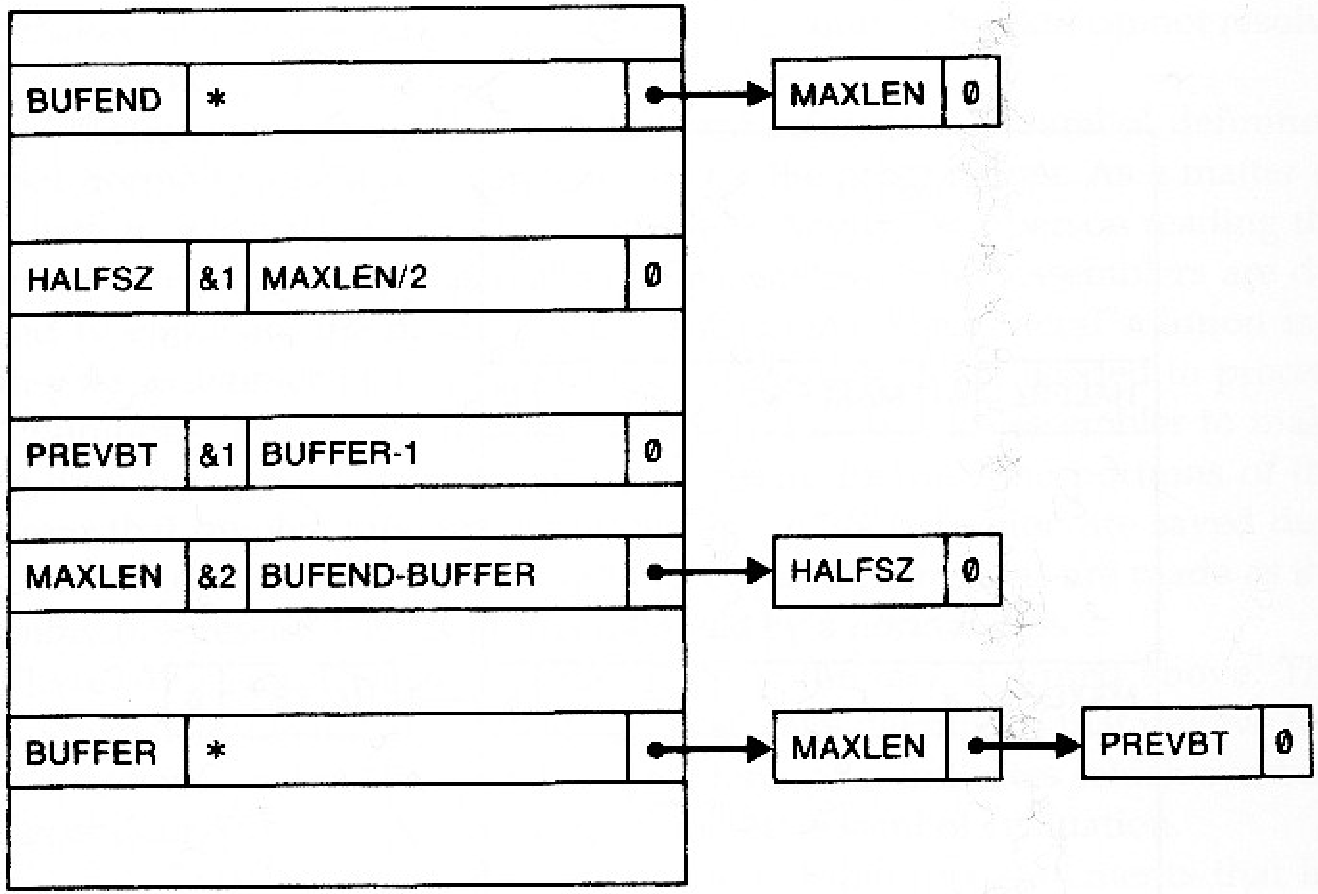
```
HALFSZ    EQU     MAXLEN/2
MAXLEN    EQU     BUFEND-BUFFER
PREVBT    EQU     BUFFER-1
                  .
                  .
                  .
BUFFER    RESB    4096
BUFEND    EQU     *
```

(a)



(c)

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

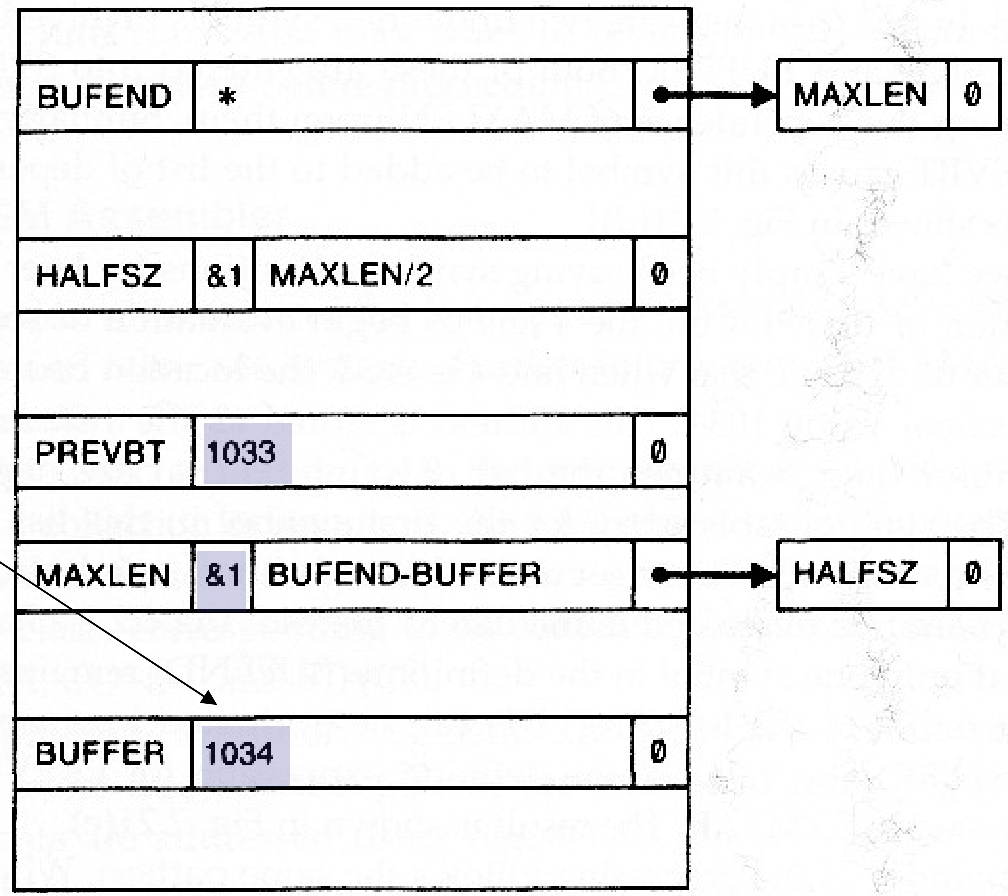


(d)

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

(a)

(assume location counter contains "1034" (Hex))



(e)

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

(a)

value of location counter (2034, Hex)

BUFEND	2034	0	1
HALFSZ	800	0	3
PREVBT	1033	0	
MAXLEN	1000	0	2
BUFFER	1034	0	

(f)

Next Week

Linkers and Loaders

read pp. 124-147 (Part I) of textbook