

Lecture 3.2

Machine-Dependent Assembler Features
SIC/XE formats, displacements, relocation

Md Mahbub E Noor

Assistant Professor, Department of CSE

University of Barishal

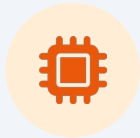
countx.lst (excerpt)

```
0000 STL      RETADR  172032
0003 +LDB     #TOTAL  69101041
0019 JLT      LOOP    3B2FF0
001F STA      TOTAL   0F4000
0022 +JSUB    REPORT  4B101044
M00000405
M00002305
```

What you will learn today



Assemble SIC/XE instructions in formats 1–4, including #, @ and +.



Choose PC-relative or base-relative addressing and compute the displacement, even a negative one.



Explain why programs must be relocatable, and which instructions break when a program moves.



Write M records, and apply them when the program is loaded at another address.

Today's plan (60 min)

0–5

Recap of Lecture 3.1

5–15

SIC/XE source, formats 1, 2

15–35

Formats 3, 4: the choice

35–50

Program relocation

50–55

M records

55–60

Checkpoint, homework

Extending your C assembler (slides 23–24) is the lab.

Quick recap of Lecture 3.1

Q1

What does Pass 1 build?

LOCCTR for every line, and SYMTAB. Today Pass 1 must also know each instruction's format, because sizes are now 1, 2, 3 or 4 bytes.

Q2

Which record tells the loader where to start running?

The E record. Today we add the M record: "fix this field after loading".

Q3

What is the most a T record can hold?

30 bytes (1E hex), as 60 hex digits.

PART 1

SIC/XE source programs

New notation in the source, and what it means for the assembler.

New notation in SIC/XE source, read through C

Notation	Meaning	Example in COUNTX	C idea
#value	immediate: the operand is the value itself	LDT #6	t = 6;
@symbol	indirect: the word at symbol holds the address	J @RETADR	jump through a pointer, like (*fp)()
+op	format 4: a full 20-bit address	+JSUB REPORT	a "far" call
register names	format 2: work between registers	CLEAR X · TIXR T	s += a; with two local variables
BASE symbol	directive: "B will hold symbol"	BASE TOTAL	a promise, like assert(b == &total)
symbol,X	indexed (as in SIC)	LDCH TEXT,X	text[i]

Why machine-dependent? Every row exists because of a feature of the SIC/XE **hardware**. On a different CPU the assembler would need different rules.

Book: Beck §2.2, §2.2.1

COUNT again, now for SIC/XE: countx.asm

```
countx.asm

COUNTX  START  0
FIRST    STL    RETADR
          +LDB   #TOTAL
          BASE   TOTAL
          CLEAR  X
          LDT    #6
LOOP     CLEAR  A
          LDCH   TEXT,X
          COMP   #65
          JEQ    FOUND
NEXT     TIXR    T
          JLT    LOOP
          LDA    CNT
          STA    TOTAL
          +JSUB  REPORT
          J      @RETADR
FOUND    LDA    CNT
          ADD    #1
          STA    CNT
          J      NEXT
RETADR   RESW    1
CNT      WORD    0
TEXT     BYTE    C 'BANANA '
BUFFER   RESB    4096
TOTAL    RESW    1
REPORT   RSUB
END      FIRST
```

Registers instead of memory

The index is X, the length is in T (LDT #6), the byte is in A. TIXR T compares X with T. No ZERO, ONE or LENGTH words needed.

Immediate constants

COMP #65 puts 'A' inside the instruction. In SIC we needed LETTERA WORD 65.

A big buffer in the middle

BUFFER RESB 4096 pushes TOTAL and REPORT far away. That forces base-relative addressing and format 4 (slides 12-13).

A subroutine call and return

+JSUB REPORT uses format 4. J @RETADR returns through the address saved by STL.

PART 2

Choosing the format and the displacement

Pass 2 now has decisions to make for every instruction.

Formats 1 and 2: the easy ones

Register	A	X	L	B	S	T	F	PC	SW
Number	0	1	2	3	4	5	6	8	9

CLEAR X

B4

1

0

→ B410

TIXR T

B8

5

0

→ B850

RMO S,A

AC

4

0

→ AC40

ADDR A,S

90

0

4

→ 9004

How to encode format 2

1. Opcode from OPTAB: one byte.
2. r1 = number of the first register.
3. r2 = number of the second (0 if there is none).

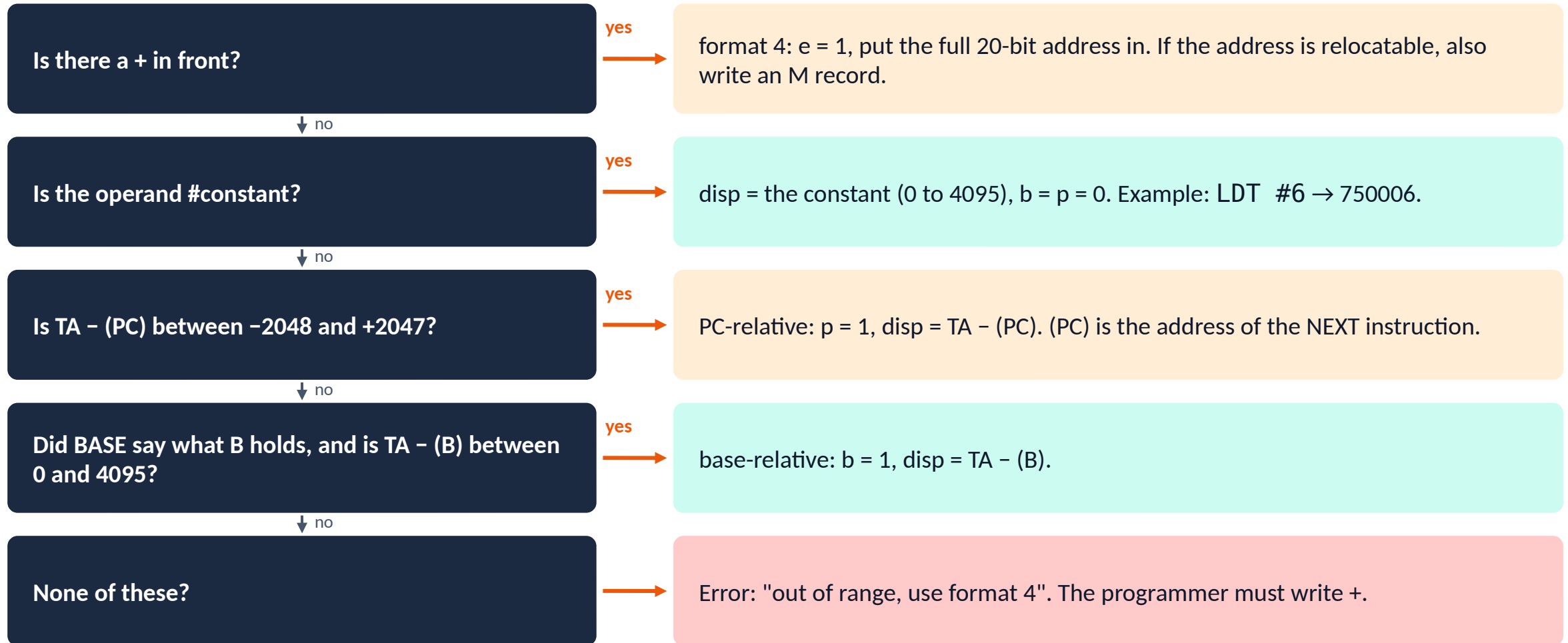
C idea

ADDR A,S is $s = s + a$; with both values already in registers. No memory, no address, so nothing to relocate.

Format 1 (e.g. FIX) is just the opcode byte.

Book: Beck §2.2.1, Appendix A

Format 3 or 4: how the assembler decides



Book: Beck §2.2.1 (instruction formats and addressing modes)

Worked example 1: PC-relative, forward

0000 FIRST STL RETADR with RETADR at 0035 (from SYMTAB)

1 Opcode: STL = 14 = 0001 01|00. Simple addressing: n = 1, i = 1 → 0001 0111 = 17.

2 (PC) = address of the next instruction = 0000 + 3 = 0003.

3 disp = TA - (PC) = 0035 - 0003 = 032. Fits in -2048 ... 2047, so p = 1.

4 Flags x b p e = 0 0 1 0 = 2. disp in 12 bits = 032.

5 Join: 17 | 2 | 032 → 172032



The trap

(PC) is **not** the address of STL itself. During execution the CPU has already fetched STL, so PC points to the next instruction.

Using 0000 instead of 0003 gives 035: a different, wrong address.

Worked example 2: PC-relative, backward (negative)

0019 JLT LOOP with LOOP at 000C

1 Opcode: JLT = **38**, plus $n = i = 1 \rightarrow$ **3B**.

2 (PC) = $0019 + 3 =$ **001C**.

3 $\text{disp} = 000C - 001C =$ **-10 hex** (-16). Negative, but within -2048 ... 2047.

4 12-bit two's complement of -10 hex: $1000 - 010 =$ **FF0**.

5 Join: $3B \mid 2 \mid FF0 \rightarrow$ **3B2FF0**

SAME AS C

Two's complement

A negative int in C is two's complement too: -16 in 32 bits is FFFFFFF0. Here only 12 bits are kept: FF0.

SAME AS C

Like a negative index

$p[-16]$ in C means "16 elements before p". A negative disp means "before the next instruction": a jump back to the top of a loop.

Worked example 3: too far for PC, so base-relative

001F STA TOTAL with TOTAL at **1041**, after BUFFER RESB 4096

1

Try PC-relative: $(PC) = 0022$. $TA - (PC) = 1041 - 0022 = \mathbf{101F} = \mathbf{4127}$. Too big: the limit is 2047.

2

Try base-relative: $BASE \text{ TOTAL}$ told the assembler $(B) = \mathbf{1041}$.

3

$disp = TA - (B) = 1041 - 1041 = \mathbf{000}$. Fits in 0 ... 4095, so $b = 1$.

4

$STA = 0C$, plus $n = i = 1 \rightarrow \mathbf{0F}$. $Flags \ x \ b \ p \ e = 0100 = \mathbf{4}$.

5

Join: $0F \mid 4 \mid 000 \rightarrow \mathbf{0F4000}$

DIFFERENT FROM C

BASE is only a promise

$BASE \text{ TOTAL}$ makes **no code**. The program itself must put the address of TOTAL into B, here with $+LDB \ #TOTAL$ at 0003.

Forget that instruction, and everything still assembles, but STA writes to a wrong address at run time. Like using an **uninitialised pointer** in C: no error message, just wrong results.

Worked example 4: immediate, indirect and format 4

LDT #6

750006

LDT = 74, immediate (n = 0, i = 1) → 75. A constant: no PC, no base, disp = 006.

C: t = 6;

COMP #65

290041

COMP = 28 → 29. disp = 65 = 041 hex. The 'A' lives inside the instruction.

C: a == 'A'

J @RETADR

3E200C

J = 3C, indirect (n = 1, i = 0) → 3E. PC-relative to RETADR: 0035 - 0029 = 00C.

C: return via a saved address

+JSUB REPORT

4B101044

JSUB = 48 → 4B. e = 1: flags 0001 = 1. Full address 01044.

C: a far call

+LDB #TOTAL

69101041

LDB = 68, immediate → 69. e = 1. Address 01041: B gets the address of TOTAL.

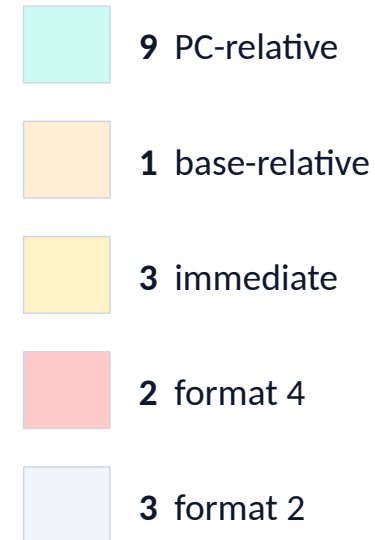
C: b = &total;

Red rows are format 4: their addresses are absolute, so each one will need an M record (Part 3).

The whole COUNTX listing, coloured by addressing mode

LOC	source			code	mode
0000	FIRST	STL	RETADR	172032	PC-relative
0003		+LDB	#TOTAL	69101041	format 4
0007		CLEAR	X	B410	format 2
0009		LDT	#6	750006	immediate
000C	LOOP	CLEAR	A	B400	format 2
000E		LDCH	TEXT,X	53A02A	PC-relative
0011		COMP	#65	290041	immediate
0014		JEQ	FOUND	332012	PC-relative
0017	NEXT	TIXR	T	B850	format 2
0019		JLT	LOOP	3B2FF0	PC-relative
001C		LDA	CNT	032019	PC-relative
001F		STA	TOTAL	0F4000	base-relative
0022		+JSUB	REPORT	4B101044	format 4
0026		J	@RETADR	3E200C	PC-relative
0029	FOUND	LDA	CNT	03200C	PC-relative
002C		ADD	#1	190001	immediate
002F		STA	CNT	0F2006	PC-relative
0032		J	NEXT	3F2FE2	PC-relative
0035	RETADR	RESW	1		reserve
0038	CNT	WORD	0	000000	data
003B	TEXT	BYTE	C 'BANANA '	42414E414E41	data
0041	BUFFER	RESB	4096		reserve
1041	TOTAL	RESW	1		reserve
1044	REPORT	RSUB		4F0000	no operand

How the 19 instructions were assembled



Plus 1 RSUB with no operand.

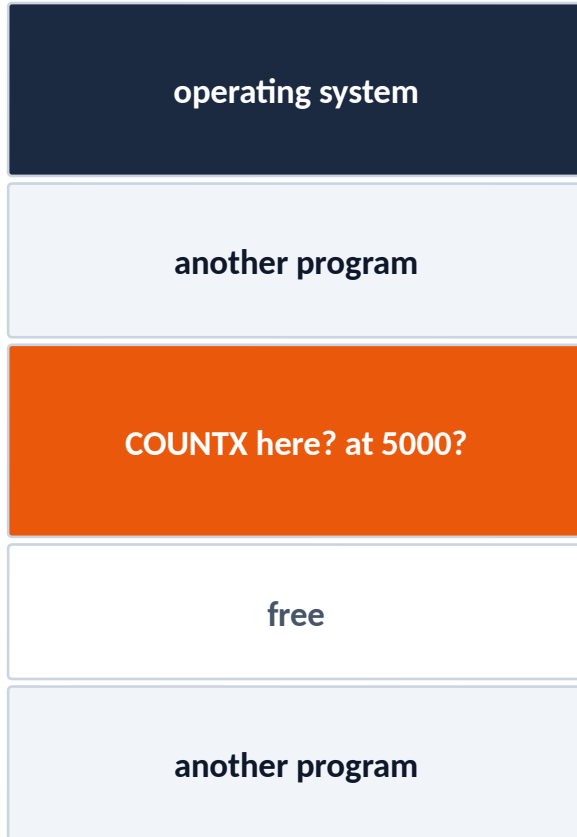
Most instructions are PC-relative: short, and they keep working wherever the program is loaded.

PART 3

Program relocation

The assembler does not know where the program will live. The loader decides, later.

Why must a program work at any address?



memory while several programs run

With several programs in memory at once (multiprogramming), the **loader** picks a free place each time. The assembler, which ran long before, cannot know where.

So **START 0** means: "assemble as if at 0, but be ready to move". The program must still work when its first byte lands at 5000.

● ● ● where.c run twice on the lab PC

```
$ ./where
main    is at 0x55c51af75149
counter is at 0x55c51af78010
$ ./where
main    is at 0x559168e93149
counter is at 0x559168e96010
```

SAME AS C

Your C programs move too

Linux loads every run at a new address (ASLR). But the **distance** from main to counter is always 0x2ec7: relative addresses survive the move.

Which instructions break when the program moves?

Kind of operand	What is stored	If the program moves 5000 up	Needs fixing?
PC-relative	a distance	the distance is the same	no
base-relative	a distance from B	B is loaded at run time with the moved address	no
immediate constant	a number	a number is a number	no
format 4 with a symbol	an absolute address	points to the old place	YES
SIC direct address	an absolute address	points to the old place	YES

2 of 19

instructions in COUNTX need fixing.

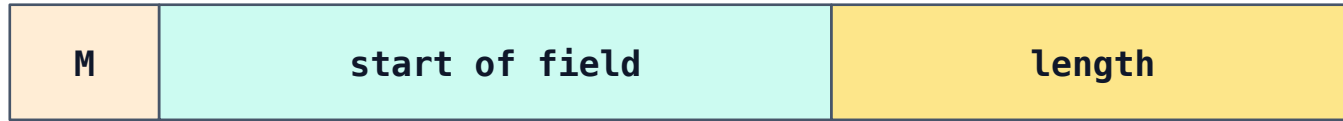
13 of 13

in the SIC program COUNT: every address in plain SIC is absolute.

This is one big reason SIC/XE has relative addressing: programs become easy to move.

Book: Beck §2.2.2 (program relocation)

The Modification (M) record



columns: 1 · 2-7 (address of the field, relative to the program start) · 8-9 (length of the field in half-bytes)

It tells the loader: "after loading, **add the program's start address** to this field".

Example: +JSUB REPORT at 0022 → 4B101044



the 20-bit address field = the last 5 half-bytes

Rule for format 4: M record address = instruction address + 1, length 05.

M00002305 says:

- the field starts in byte **0023** (the byte after the opcode),
- it is **05** half-bytes long (20 bits).

The field starts in the middle of byte 0023, so the length is counted in half-bytes, not bytes.

The object program of COUNTX

HCOUNTX000000001047

T0000001C17203269101041B410750006B40053A02A290041332012B8503B2FF0

T00001C190320190F40004B1010443E200C03200C1900010F20063F2FE2

T00003809000000042414E414E41

T001044034F0000

M00000405

M00002305

E000000

H: start 000000 (relocatable), length 001047 = 4167 bytes.

4 T records: 1C + 19 + 09 + 03 bytes. The gaps are RETADR (RESW), BUFFER and TOTAL.

M00000405: fix the address in +LDB #TOTAL at 0003.

M00002305: fix the address in +JSUB REPORT at 0022.

Watch the loader fix the program

Relocation: one program, two load addresses

object program countx.obj

```
HCOUNTX000000001047
T00000001C17203269101041B410750006B40053A02A2...
T000001C190320190F40004B1010443E200C03200C190...
T0000380900000042414E414E41
T001044034F0000
M00000405
M00002305
E000000
```

instruction		kind
STL	RETADR	PC-relative
+LDB	#TOTAL	format 4
JLT	LOOP	PC-relative
STA	TOTAL	base-relative
+JSUB	REPORT	format 4
J	@RETADR	PC-relative

loaded at 0000

run it: **TOTAL = 3**

address	bytes in memory
---------	-----------------

0000	172032
0003	69101041
0019	3B2FF0
001F	0F4000
0022	4B101044
0026	3E200C

loaded at 5000

address	bytes in memory
---------	-----------------

5000	172032
5003	69106041
5019	3B2FF0
501F	0F4000
5022	4B106044
5026	3E200C

And the others? STL, JLT, J @RETADR are PC-relative; STA TOTAL is base-relative. Their bytes are the same at 0000 and 5000: a distance does not change when the whole program moves.

Watch for

1. Loaded at **0000**, nothing needs fixing.
2. Loaded at **5000**, the two red rows still hold old addresses.
3. Each **M record** adds 5000 to one field.
4. The PC-relative rows have the **same bytes** at both addresses.
5. Both runs give **TOTAL = 3**.

Click the video to play or pause. About 50 seconds, no sound.

Your C object file has M records too

```
readelf -r hello.o (Lecture 1's hello.c)
```

Relocation section '.rel.text' contains 2 entries:

Offset	Type	Sym. Name + Addend
000000000000b	R_X86_64_PC32	.rodata - 4
0000000000013	R_X86_64_PLT32	puts - 4

```
objdump -d hello.o (Lecture 1)
```

```
8: 48 8d 05 00 00 00 00 lea
12: e8 00 00 00 00      call
# hole at 8+3 = 0b
# hole at 12+1 = 13
```

SAME AS C

Same idea as M

Each entry says **which bytes to patch** (offset 0b, offset 13) once an address is known. Exactly the two holes we saw in Lecture 1.

DIFFERENT FROM C

But richer

An ELF entry also names a **symbol** (puts) and a **type**: PC32 means "PC-relative, 32 bits". SIC's M record only adds the start address. Lecture 3.3 adds symbols to M records too.

LAB 3.2

Relocation in your own assembler

Add M records to the C assembler from Lab 3.1, then move a program by hand.

Lab 3.2 tasks, step by step

1

Hand-assemble on paper

Lines 0000–0032 of countx.asm. Write `n i x b p e`, the `disp` and the code. Check against slide 14.

2

Add -m to sicasasm

With `./sicasasm -m file.asm`, write `M{loc+1}04` for every instruction with an operand.

3

Count them

Run it on count.asm: you should get 13 M records. Compare with the 2 of countx.asm.

4

Relocate by hand

Load count.obj at 5000 instead of 2000: add 3000 to each marked field. `LDCH TEXT,X` must become `50D027`.

5

Write apply_m() in C

`void apply_m(unsigned char *mem, int addr, int halfbytes, int delta);` Test it on count.obj.

6

Explain

In 3 sentences: why does SIC need 13 M records and SIC/XE only 2? What would a big program look like?

Submit: your updated sicasasm.c, count.obj with M records, and your apply_m() with a test.

Mistakes and the messages they produce

Forgot BASE

```
● ● ● assembler output
# countx.asm, BASE
removed
line 13: TOTAL is out
of PC and base range;
use format 4
```

Neither PC- nor base-relative reaches TOTAL. Add BASE TOTAL (and load B!), or write +STA.

Constant too big

```
● ● ● assembler output
# LDA #4096
line 2: constant 4096
too big for format 3,
use +
```

12 bits hold at most 4095. Write +LDA #4096 (format 4: 20 bits).

Wrong (PC)

```
● ● ● assembler output
# STL RETADR at 0000
wrong: 0035 - 0000 = 035
right: 0035 - 0003 = 032
```

No message at all: the code is just wrong. (PC) is the address of the **next** instruction.

Negative disp

```
● ● ● assembler output
# JLT LOOP at 0019
wrong: FFF0 (16 bits)
right: FF0 (12 bits)
```

Keep exactly 12 bits for disp. 4 hex digits would push into the flags.

Checkpoint: test yourself

Q1 Encode CLEAR S (CLEAR = B4, S = 4).

→ B440

Q2 JEQ FOUND at 0014, FOUND at 0029. What is the disp?

→ $0029 - 0017 = 012 \rightarrow 332012$

Q3 Why does STA TOTAL use base-relative and not PC-relative?

→ TOTAL is 4127 bytes away; PC-relative reaches only 2047

Q4 Which COUNTX instructions get an M record?

→ the two format-4 ones: +LDB #TOTAL and +JSUB REPORT

Q5 What does M00002305 ask the loader to do?

→ add the start address to the 5 half-bytes starting in byte 0023

Key ideas from today



SIC/XE Pass 1 needs each format's size: 1, 2, 3 or 4 bytes.



Format 3: try PC-relative ($-2048 \dots 2047$), then base-relative ($0 \dots 4095$). + forces format 4.



(PC) = address of the next instruction. Negative disp = 12-bit two's complement.



BASE is a promise to the assembler; the program must load B itself.



Distances survive a move; absolute addresses do not. Only those need fixing.



An M record = where (address + 1) and how long (05 half-bytes) to fix.

Homework 3.2 (due before Lecture 3.3)

1. Assemble

Assemble every line of `sumx.asm` (in the lab files). For each, give LOC, `n i x b p e`, `disp` or address, and the code.

2. Explain

Which instructions in `sumx.asm` are PC-relative, which base-relative, which immediate? Why must `STA TOTAL` be base-relative?

3. Records

Write the complete object program of `sumx.asm`, including every M record.

4. Relocate

The loader puts `sumx` at 3000. Which bytes change? Write the new `+LDB` instruction.

5. C

Finish `apply_m()` from the lab and use it to relocate `count.obj` to 5000. Print the first T record after relocation.

Next lecture and reading

NEXT: LECTURE 3.3

Machine-independent features and design options

Literals, EQU and expressions, program blocks and control sections: features any assembler can have.

One-pass and multi-pass assemblers.

Main text

L. L. Beck, **System Software**, 3rd ed. Today: §2.2 (2.2.1 formats and addressing, 2.2.2 relocation). Read §2.3 before Lecture 3.3.

x86-64 relocation

R. E. Bryant & D. R. O'Hallaron, **CS:APP**, 3rd ed., §7.7 (relocation): the ELF version of today's M records.

ASLR

`man 8 ld.so` and `/proc/sys/kernel/randomize_va_space` on the lab PC.

Lab files

`lab3_files.zip`: `countx.asm`, `sumx.asm` and the test files for `-m`.