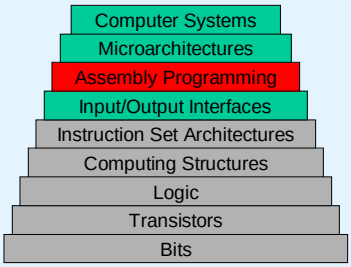
Assembly Programming: Ch 7



1

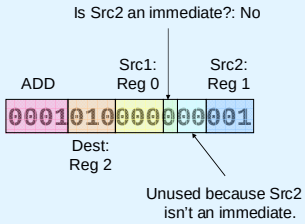
Example LC-3 instruction

0001010000000001

Add the contents of register 0 with the contents of register 1, and put the result in register 2.

2

Example LC-3 instruction



Add the contents of register 0 with the contents of register 1, and put the result in register 2.

3

Example LC-3 instruction

Machine language:

0001010000000001

Assembly language:

ADD R2, R0, R1

Add the contents of register 0 with the contents of register 1, and put the result in register 2.

4

Sample LC-3 assembly program

```
; SUM ← N+…+1

      .ORIG x1000
      AND R1, R1, #0    ; R1 ← 0    R1=Running total
      LD  R0, N         ; R0 ← N    R0=Counter: N…0
LOOP:
      BRz DONE          ; R0 = 0 ?
      ADD R1, R1, R0    ; R1 ← R1 + R0
      ADD R0, R0, #-1   ; R0 ← R0 - 1
      BR  LOOP
DONE:
      ST  R1, SUM       ; SUM ← R1
      TRAP x25          ; Halt program

N:    .FILL 17
SUM:  .BLKW 1
      .END
```

5

Sample LC-3 assembly program

```
; SUM ← N+…+1

      .ORIG x1000
      AND R1, R1, #0    ; R1 ← 0    R1=Running total
      LD  R0, N         ; R0 ← N    R0=Counter: N…0
LOOP:
      BRz DONE          ; R0 = 0 ?
      ADD R1, R1, R0    ; R1 ← R1 + R0
      ADD R0, R0, #-1   ; R0 ← R0 - 1
      BR  LOOP
DONE:
      ST  R1, SUM       ; SUM ← R1
      TRAP x25          ; Halt program

N:    .FILL 17
SUM:  .BLKW 1
      .END
```

Arithmetic

6

Sample LC-3 assembly program

```
; SUM ← N+…+1

      .ORIG x1000
      AND R1, R1, #0    ; R1 ← 0    R1=Running total
      LD  R0, N         ; R0 ← N    R0=Counter: N…0
LOOP:
      BRz DONE          ; R0 = 0 ?
      ADD R1, R1, R0    ; R1 ← R1 + R0
      ADD R0, R0, #-1   ; R0 ← R0 - 1
      BR  LOOP
DONE:
      ST  R1, SUM       ; SUM ← R1
      TRAP x25          ; Halt program

N:    .FILL 17
SUM:  .BLKW 1
      .END
```

Data Movement

Sample LC-3 assembly program

```
; SUM ← N+…+1

      .ORIG x1000
      AND R1, R1, #0    ; R1 ← 0    R1=Running total
      LD  R0, N         ; R0 ← N    R0=Counter: N…0
LOOP:
      BRz DONE          ; R0 = 0 ?
      ADD R1, R1, R0    ; R1 ← R1 + R0
      ADD R0, R0, #-1   ; R0 ← R0 - 1
      BR  LOOP
DONE:
      ST  R1, SUM       ; SUM ← R1
      TRAP x25          ; Halt program

N:    .FILL 17
SUM:  .BLKW 1
      .END
```

Control Flow

Sample LC-3 assembly program

```
; SUM ← N+…+1

      .ORIG x1000
      AND R1, R1, #0    ; R1 ← 0    R1=Running total
      LD  R0, N         ; R0 ← N    R0=Counter: N…0
LOOP:
      BRz DONE          ; R0 = 0 ?
      ADD R1, R1, R0    ; R1 ← R1 + R0
      ADD R0, R0, #-1   ; R0 ← R0 - 1
      BR  LOOP
DONE:
      ST  R1, SUM       ; SUM ← R1
      TRAP x25          ; Halt program

N:    .FILL 17
SUM:  .BLKW 1
      .END
```

Comments

Sample LC-3 assembly program

```
; SUM ← N+…+1

      .ORIG x1000
      AND R1, R1, #0    ; R1 ← 0    R1=Running total
      LD  R0, N         ; R0 ← N    R0=Counter: N…0
LOOP:
      BRz DONE          ; R0 = 0 ?
      ADD R1, R1, R0    ; R1 ← R1 + R0
      ADD R0, R0, #-1   ; R0 ← R0 - 1
      BR  LOOP
DONE:
      ST  R1, SUM       ; SUM ← R1
      TRAP x25          ; Halt program

N:    .FILL 17
SUM:  .BLKW 1
      .END
```

Labels

Sample LC-3 assembly program

```
; SUM ← N+…+1

      .ORIG x1000
      AND R1, R1, #0    ; R1 ← 0    R1=Running total
      LD  R0, N         ; R0 ← N    R0=Counter: N…0
LOOP:
      BRz DONE          ; R0 = 0 ?
      ADD R1, R1, R0    ; R1 ← R1 + R0
      ADD R0, R0, #-1   ; R0 ← R0 - 1
      BR  LOOP
DONE:
      ST  R1, SUM       ; SUM ← R1
      TRAP x25          ; Halt program

N:    .FILL 17
SUM:  .BLKW 1
      .END
```

Directives

LC-3 arithmetic/logic instructions

		DR	SR1		SR2
ADD R2, R0, R1	R2 ← R0 + R1	0001	xxx	xxx	0 00 xxx
ADD R2, R0, #5	R2 ← R0 + 5	0001	xxx	xxx	1 xxxxx
AND R2, R0, R1	R2 ← R0 · R1	0101	xxx	xxx	0 00 xxx
AND R2, R0, #5	R2 ← R0 · 5	0101	xxx	xxx	1 xxxxx
NOT R2, R0	R2 ← R0'	1001	xxx	xxx	111111

LC-3 data movement instructions (1)

LD R2, #5 R2 ← [PC + 5]	0010 xxx xxxxxxxxx DR PCoffset
LDI R2, #5 R2 ← [[PC + 5]]	1010 xxx xxxxxxxxx DR PCoffset
LDR R2, R0, #5 R2 ← [R0 + 5]	0110 xxx xxx xxxxxx DR BaseR offset
LEA R2, #5 R2 ← PC + 5	1110 xxx xxxxxxxx DR PCoffset

13

LC-3 data movement instructions (2)

ST R2, #5 [PC + 5] ← R2	0011 xxx xxxxxxxxx DR PCoffset
STI R2, #5 [[PC + 5]] ← R2	1011 xxx xxxxxxxxx DR PCoffset
STR R2, R0, #5 [R0 + 5] ← R2	0110 xxx xxx xxxxxx DR BaseR offset

14

LC-3 control flow instructions (1)

BRz #5 If z, PC ← PC + 5	0000 nzp xxxxxxxxx PCoffset
JMP R0 PC ← R0	1110 000 xxx 000000 BaseR

Instructions setting condition codes:
ADD, **AND**, **LD**, **LDI**, **LDR**, **LEA**, **NOT**

15

LC-3 control flow instructions (2)

JSR #5	0100 1 xxxxxxxxxxx PCoffset
JSRR R2, #5	0010 000 xxx 000000 BaseR
RET	1100 000 111 000000

Used to implement functions. Explained later.

16

LC-3 control flow instructions (3)

RTI Return from interrupt.	1000 000000000000
TRAP #5 Give code #5 to OS.	1111 0000 xxxxxxxx TrapVec

17

LC-3 reserved instruction

reserved	1101 xxxxxxxxxxx

18

LC-3 directives

Opcode	Operand	Meaning
.ORIG	address	Starting address of program
.END		End of program text
.BLKW	n	Allocate n words of storage
.FILL	n	Allocate one word, initialize with value n
.STRINGZ	n-character string	Allocate n+1 locations, initialize with characters and terminating NUL (0x00) character

19

LC-3 trap codes

#	Name	Function
0x20	GETC	Read character from kbd, copy ASCII code into lower 8 bits of R0, clear higher 8 bits of R0
0x21	OUT	Print character R0[7:0] to console
0x22	PUTS	Print string starting at address in R0 (a string is a sequence of ASCII characters terminated by the ASCII zero, which is also called NUL)
0x23	IN	Print a prompt and read character like GETC
0x25	HALT	Halt execution and print message on console

LC-3 provides pseudo-instructions for each of those traps.

20

Another example

```
        .ORIG 0x3050      ; specify starting address
        LD    r2, SIX     ; set counter to 6
        LD    r3, ONE     ; set initial value to 1

; Now start the main loop
AGAIN   ADD   r3, r3, r3  ; double value
        ADD   r2, r2, #-1 ; decrement counter
        BRp   AGAIN       ; repeat until r2 ≤ 0
; Now store the result and stop
        ST    r3, RESULT  ; store value when done
        HALT              ; stop the program -- trap

; Data storage section
RESULT  .BLKW 1           ; reserve a word for result
SIX     .FILL x0006       ; provide the constant 6
ONE     .FILL x0001       ; provide the constant 1
        .END
```

21

Programming (coding) tips

1. Use meaningful label names.
2. Add comments to important instructions and between sections of code.
3. Add comments about each register or memory location.
4. Start label, opcode, operands, comment at same column in each line of code.
   - Except when entire line is a comment
5. Add a program header comment with name and purpose of program, name of author, date, etc.

22

How do we get program into machine format?

Assembler

- A program that takes an assembly language program as input and gives a machine language program as output.
- Translates opcodes, labels, operands, directives into machine-language format.
- Also checks for errors.
  - Immediate/direct mode operand that is out of range (value can't be represented with given number of bits)
  - Label used but never defined.
  - …

23

How does assembler work?

Two passes on code:

1. Construct a *symbol table*
   - For each label definition, add label with its address.
   - E.g., **`AGAIN`** in the example code gets 0x3052 (two words after **`ORIG`**), **`RESULT`** gets 0x3057 (seven words after **`ORIG`**), and **`SIX`** gets 0x3058.
2. Produce machine language program *(object file)*
   - Convert opcodes, operands, directives.
     - Fill in labels looked up from symbol table.
   - E.g., **`ST R3, RESULT`** ⇒ 0011 011 0 0101 0111

24

Problem: Multiple object files

We would like to have separate files for separate pieces of code or data.

Thus, want to use labels across separate files.

But, assembler chokes on undefined labels.

25

Solution: Linking

Add a new assembler directive:

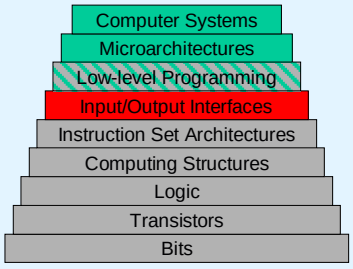
- **`.EXTERNAL LabelName`**
- Assembler leaves a hole in symbol table for **`LabelName`**.
- Fill in bogus value when **`LabelName`** used by instructions.
- Make a list of all instructions that use **`LabelName`**.

Run a separate *linker* program on <u>all</u> object files.

- *Resolves* (fills in the values for) external labels.
- Rewrites machine language instructions that use such labels.

26

Where are we headed? Ch 8



27