LC3 Assembly Programming

1

Today

- Quick review from lecture (on Assembly Prog)
- Exercise – to be submitted or demo

2

2

LC-3 Assembly Language Syntax

- Each line of a program is one of the following:
  - an instruction
  - an assembler directive (or pseudo-op)
  - a comment
- Whitespace (between symbols) and case are ignored.
- Comments (beginning with ";") on one line are also ignored.

- An instruction has the following format:

```
LABEL OPCODE OPERANDS ; COMMENTS
```

optional *mandatory*

3

3

Assembler Directives

- Pseudo-operations.. To make programmer's life easier
  - do not refer to operations executed by program
  - used by assembler
  - **looks like instruction, but "opcode" starts with dot**

Opcode	*Operand*	*Meaning*
`.ORIG`	address	starting address of program
`.END`		end of program
`.BLKW`	n	allocate n words of storage
`.FILL`	A	allocate one word, initialize with value A
`.STRINGZ`	n-character string	allocate n+1 locations, initialize w/characters and null terminator

4

4

Trap Codes

- LC-3 assembler provides "pseudo-instructions" for each trap code, so you don't have to remember them… *more on TRAP instructions later*…

Code	Equivalent	Description
HALT	TRAP x25	Halt execution and print message to console.
IN	TRAP x23	Print prompt on console, read (and echo) one character from keybd. Character stored in R0[7:0].
OUT	TRAP x21	Write one character (in R0[7:0]) to console.
GETC	TRAP x20	Read one character from keyboard. Character stored in R0[7:0].
PUTS	TRAP x22	Write null-terminated string to console. Address of string is in R0.

5

5

Label: refers to a memory location

```
; assembly program for temp2= temp1 +2;
        .ORIG x3000   ;program starts at address x3000
        LD     R1, temp1    ; temp1 is location in memory
               ; note: offset not specified by programmer
        BRn Done   ;if number is Negative goto end
        ADD    R3, R1, #2    ; Add 2 store into R3
        ST     R3, temp2   ; store result into temp2
Done    HALT   ;halt program
;
temp2   .BLKW  1
temp1   .FILL  x0005
        .END   ; end of program
```

Must have Opcode and Operands

int temp2;

Immediate values/constants

Decimal	#
Binary	b
Hex	x

int temp1 =5;

.BLKW is Assembler Directive (reserve one location with label 'temp2')

.FILL is Assembler Directive (reserve one location with label 'temp1') and Initialize the value there to be x0005

6

6

Recap: Problem Solving and Problem Decomposition

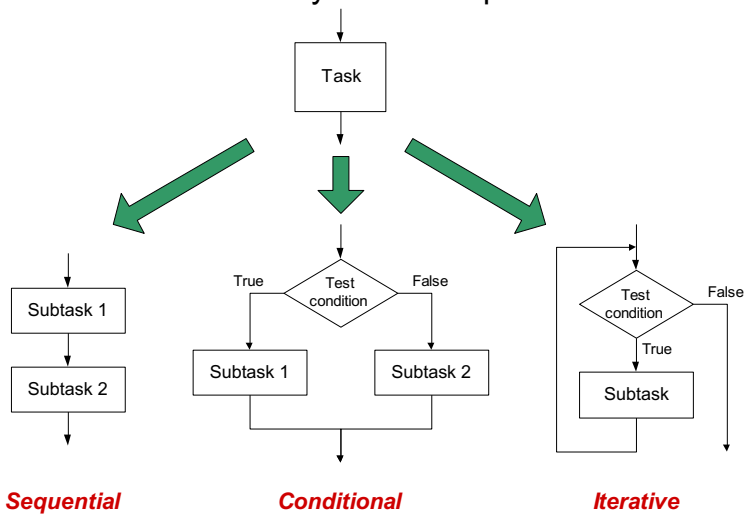
- With an eye towards writing assembly programming/low-level software
- Flowcharts anyone ?
- Decomposition:
  - Break problem/solution into sub-problems/modules
    - Structured programming
  - Connect the modules…
    - With conditionals, iterations, sequence,….

7

7

Three Basic Constructs

- There are three basic ways to decompose a task:



8

8

LC-3 Control Instructions

▪How do we use LC-3 instructions to encode the three basic constructs?

▪Sequential

- Instructions naturally flow from one to the next, so no special instruction needed to go from one sequential subtask to the next.

Two instructions to negate R1, i.e. to compute 2's complement of R1

```
NOT R1, R1
ADD R1, R1, #1
ADD R2, R1, R0
BRz equal
```

▪Conditional and Iterative

- Create code that converts condition into N, Z, or P.
  Example:
  Condition: "Is R0 = R1?"
  Code: Subtract R1 from R0; if equal, Z bit will be set.
- Then use BR instruction to transfer control to the proper subtask.

9

9

Converting Code to Assembly

- Can use a standard template approach
- Typical Constructs
  - if/else
  - while
  - do/while
  - for
- Important: if-else could entail both conditional branch and unconditional branch

10

10

if/else

```
if(x > 0)
{
     r2 = r3 + r4;
}
else
{
     r5 = r6 + r7;
}
/* DONE */
```

```
        LD    R1, X
        BRP   THEN
        ADD R5,R6,R7
        BRNZP DONE
THEN    ADD R2,R3,R4
DONE    ...
```

Unconditional branch

11

if/else

```
if(x > 0)
{
     r2 = r3 + r4;
}
else
{
     r5 = r6 + r7;
}
```

```
        LD    R1,X
        BRNZ ELSE
        ADD R2,R3,R4
        BRNZP DONE
ELSE    ADD R5,R6,R7
DONE    ...
```

12

while

```
x = 0;
i = 10;
while(i > 0)
{
      x = x + i;
      i--;
}
```

```
        AND R1,R1,#0
        AND R2,R2,#0
        ADD R1,R1,#10
WHL     BRNZ DONE
        ADD R2,R2,R1
        ADD R1,R1,#-1
        BRNZP WHL
```

Clearing a register/variable

Initialize register/variable

Decrement count by 1

13

Example (Labs this week): Multiplication

- No multiplication instruction in LC3
- Implement multiplication by repeated addition
  - X * Y = Y added to itself X times
- Simplifying assumption: Assume both are non-negative
  - But could be zero
- Important Note on using LC3 simulator
  - Open file (or start new one), type your assembly code
  - Assemble
  - Open Simulator…. **Set breakpoint at last instruction/HALT of your program– this will stop the simulation after the instruction**
  - **Set it by click on small dot (exclamation) next to where you want to set breakpoint…it goes red to indicate breakpoint set.**

14

14

Exercise: Multiplication code in LC3

C code:

```
/* X,Y are variables */
if (X==0)|| (Y==0))
      mult=0;
else
      {  i=X;
         while i>0 {
         mult = mult + Y;
          i = i-1;}
```

- LC 3 Code….
- Set X=5 and Y=7, check result

```
; initialize R0=0 (R0 is mult)
; read X from memory to R1
; using LD R1, X instruction
; check if X=0 (go to DONE)
; read Y from memory to R2
; check if Y=0 (goto DONE)
; else part
; copy X to R3 (var i)
; check if i>0 (R3 Positive?)
; add Y to mult (add R0,R2)
; decrement i (R3= R3 -1)
; loop back to start of while
; DONE
```

15