Next Thursday

LC-3 Assembly Language

(Ch7)
LC-3 is a load/store RISC architecture

- Has 8 general registers
- Has a flat 16-bit addressing range
- Has a 16-bit word size
- Load variables from memory to register
Syntax of LC-3

• One instruction, declaration per line
• Comments are anything on a line following ";"
• Comments may not span lines

LC-3 has 2 basic data types
  • Integer
  • Character

Both take 16-bits of space (a word) though a character is only 8-bits in size.
Labels

- Symbolic names that are used to identify memory locations
- Location for target of a branch or jump
- Location for a variable for loading and storing
- Can be 1-20 characters in size
**Directives** give information to the assembler. All directives start with `.‘’ (period)

<table>
<thead>
<tr>
<th>Directive</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>.ORIG</td>
<td>Where to start in placing things in memory</td>
</tr>
<tr>
<td>.FILL</td>
<td>Declare a memory location</td>
</tr>
<tr>
<td>.BLKW</td>
<td>Reserve a group of memory locations</td>
</tr>
<tr>
<td>.STRINGZ</td>
<td>Declare a group of characters in memory</td>
</tr>
<tr>
<td>.END</td>
<td>Tells assembly where your program source ends</td>
</tr>
</tbody>
</table>

. ORIG X'3000
Here be dragons

x3000

x5000
.ORIG

• Tells simulator where to put your code in memory
• Only one allowed per program
• PC gets set to this address at start up
• Similar to the “main” in “C”
"C"

```

ty
type

name;
```

varname:

int (integer)
char (character)
float (floating point)

"LC-3"

```

value

varname .FILL value
```

value is required – the initial value
LC-3 Syntax

.FILL

flag .FILL x0001
counter .FILL x0002
letter .FILL x0041 ; A
letters .FILL x4241 ; BA

- One declaration per line
- Always declaring 16-bits, the word size of LC-3
- Don’t mix in with your code, will be treated like an instruction
ADD R1, R2, R3
4 foo.fill x1
bar.fill -1  b
1111111111
.BLKW

- Tells assembler to set aside some number of sequential memory locations
- Useful for arrays
- Can be initialized
LC-3 Syntax

Examples of .BLKW:

;set aside 3 locations
.BLKW 3

;set aside 1 location and label it.
Bob .BLKW 1

;set aside 1 location, label and initialize to 4.
Num .BLKW 1 #4
LD R0, 33
LD R0, FOO

F00: FILL 4

X3000
X3001
X3010
LC-3 Syntax

.STRINGZ

- Used to declare a string of characters
- Is terminated by x0000
- One character per memory location

Example:

hello .STRINGZ “Hello World!”
LC-3 Syntax

.END

• Tells the assembler where your program ends
• Only one allowed in your program
<table>
<thead>
<tr>
<th>“LC-3”</th>
<th>“C”</th>
</tr>
</thead>
<tbody>
<tr>
<td>LD R1, X</td>
<td></td>
</tr>
<tr>
<td>LD R2, Y</td>
<td></td>
</tr>
<tr>
<td>ADD R3, R2, #0</td>
<td>Z = Y</td>
</tr>
<tr>
<td>ADD R3, R1, R2</td>
<td>Z = X + Y</td>
</tr>
<tr>
<td>????</td>
<td>Z = X - Y</td>
</tr>
<tr>
<td>????</td>
<td>Z = X * Y</td>
</tr>
<tr>
<td>ST R3, Z</td>
<td>Z = X / Y</td>
</tr>
</tbody>
</table>

An immediate is a value specified in an instruction, not by a .FILL declaration.
Simple LC-3 program

```
.ORIG x3000
LD R2, Zero
LD R0, M0
LD R1, M1
BRz Done
ADD
ADD
BR ST
HALT
.END
```

- What does this program do?
- What is in “Result” at the end?

\[ R2 = R2 + R0 \]
\[ R1 = R1 - 1 \]

M0 \times M1

\[ x0000 \]
\[ x0000 \]
\[ x0004 \]
\[ x0002 \]
R₀ = 2
R₀ - 2 = 0
Program Execution

- Assembler translates to executable – machine language
- Linker combines multiple LC-3 files – if any
- Loader puts executable into memory and makes the CPU jump to first instruction, .ORIG.
- Executes
- When executing is done returns control to OS
  - Or simulator or monitor
- Load again to run again with different data
  - In this case, assemble again, too, since data is in program.
HLL – if/else statements...

if (condition)
    statement;
else
    statement;
if (count < 0)
    count = count + 1;

LD
BRpz
ADD

R0, count
greatzero
R0, R0, #1

next instruction goes here
LD  R0, count
BRn  foo  x3010
BRzp  foo2
ADD  R0, R0, 1
if (R0 == 'A')
    R1 = R1 + R2
LD R3, ALPHA
NOT R3, R3
ADD R3, R3, 1
ADD R3, R3, R0
BRnp skip
    ADD R1, R1, R2
skip ; rest of program
ALPHA .FILL 'A' ; 65
if (condition)
else
Loops can be built out of IF’s – WHILE:

```
C

while (count > 0) {
    a = a + count;
    count--;  
}

BRn2p
```
“LC-3”

LD
LD
while BRnz
ADD
ADD
BR
ST
ST
endwhile

R1, a
R0, count

while
R1, R1, R0
R0, R0, #-1

R1, a
R0, count
Procedure Calls

Simple procedure calls require 2 instructions:

“JSR” or “JSRR” Jump Service Routine
  • Saves the return address into R7

“RET” Jump Return
  • Be careful with registers!!
  • Cannot nest unless R7 is saved elsewhere
  • Cannot be recursive without a stack
Example

; calculate R2 = R0 - R1
Sub
- NOT
- ADD
- ADD
- RET

JSR Sub
...

; calls procedure
R2, R1
R2, R2, #1
R2, R2, R0
; returns to line after
; JSR Sub

Maxwell James Dunne – Spring 2016
Repeat loops

/* do statement while expression is TRUE */
/* when expression is FALSE, exit loop */
do {
    if (a < b)
        a++;
    if (a > b)
        a--;}
while (a != b)
repeat
secondif
until

LD
LD
JSR
BRpz
ADD
JSR
BRnz
ADD
JSR
BRnp

R0, a
R1, b
Sub ; R2 = R0 - R1
secondif
R0, R0, #1
Sub
until
R0, R0, #-1
Sub
repeat
For loops

"C"

```
for ( l = 3; l <= 8; l++)
{ a = a+l; }
```
"LC-3"

; R0=a, R1=l, R2=temp

LD R0, a
AND R1, R1, #0 ; init l to zero
ADD R1, R1, #3 ; now make 3
ADD R2, R1, #-8
BRp
ADD R0, R0, R1 ; a=a+l
ADD R1, R1, #1 ; l++
BR for

forall

CMPE-012/L
Maxwell James Dunne – Spring 2016
TRAP
(System Calls)

• Very tedious and dangerous for a programmer to deal with IO at the OS level.

• Need an instruction though to get the attention of the OS.

Use the “TRAP” instruction and a “trap vector”.
<table>
<thead>
<tr>
<th>Trap Vector</th>
<th>Assembler Name</th>
<th>Usage &amp; Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x20</td>
<td>GETC</td>
<td>Read a character from console into R0, not echoed.</td>
</tr>
<tr>
<td>0x21</td>
<td>OUT</td>
<td>Write character in R0 to console.</td>
</tr>
<tr>
<td>0x22</td>
<td>PUTS</td>
<td>Write string of characters to console. Start with</td>
</tr>
<tr>
<td></td>
<td></td>
<td>character at address contained in R0. Stops when</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0x0000 is encountered.</td>
</tr>
<tr>
<td>0x23</td>
<td>IN</td>
<td>Print a prompt to console and read in a single</td>
</tr>
<tr>
<td></td>
<td></td>
<td>character into R0. Character is echoed.</td>
</tr>
<tr>
<td>0x24</td>
<td>PUTSP</td>
<td>Write a string of characters to console, 2</td>
</tr>
<tr>
<td></td>
<td></td>
<td>characters per address location. Start with</td>
</tr>
<tr>
<td></td>
<td></td>
<td>characters at address in R0. First [7:0] and then</td>
</tr>
<tr>
<td></td>
<td></td>
<td>[15:0]. Stops when 0x0000 is encountered.</td>
</tr>
<tr>
<td>0x25</td>
<td>HALT</td>
<td>Halt execution and print message to console.</td>
</tr>
</tbody>
</table>
To print a character
; the char must be in R0.
TRAP x21
or
OUT

To read in a character
; will go into R0, no echo.
TRAP x20
or
GETC
To end your program:

TRAP
or
HALT

x25