LC-3
More LC-3 Programming
Control Instructions

Used to alter the sequence of instructions. This allows us to move to a particular instruction.

Conditional Branch

– branch is *taken* if a specified condition is true
– else, the branch is *not taken*
  • next sequential instruction is executed

Maxwell James Dunne
Unconditional Branch (or Jump)
- always changes instruction

TRAP
- changes to an OS “service routine”
- routine will return control to the next instruction (after TRAP) when finished
**Condition Codes**

LC-3 has three *condition code* bits:

- **N** -- negative
- **Z** -- zero
- **P** -- positive (greater than zero)

Set by any instruction that writes a value to a register (ADD, AND, NOT, LD, LDR, LDI, LEA)

Exactly one will be set at all times — Based on the last instruction that altered a register
Branch Instruction

- Branch specifies one or more condition codes.
- If the set bit is specified, the branch is taken.
  - PC-relative addressing is used
  - Target address is made by adding signed offset (IR[8:0]) to current PC.
If the branch is not taken, the next sequential instruction is executed.

There are hardware limits on how far you can branch.
BR (Branch)

• It does no computation, only looks at condition codes

• If condition code is set, go to LABEL, can combine codes
  - BRz
  - BRn
  - BRp
  - BRzp

  ADD R0,R1,R2
  BRz FOO ; if zero we go to label FOO
BR (unconditionally)

- Degenerate case, always goes to LABEL
  - BRnzp

BRnzp FOO
BRnzp FOO ; always go to label FOO
Example: Using a Branch

Compute sum of 12 integers

Numbers start at label NUMS. Program starts at location x3000.
**Example: Using a Branch**

<table>
<thead>
<tr>
<th>Instruction</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>LEA R1,NUMS</strong></td>
<td>R1 ← NUMS</td>
</tr>
<tr>
<td><strong>AND R3,R3,0</strong></td>
<td>R3 ← 0</td>
</tr>
<tr>
<td><strong>AND R2,R2,0</strong></td>
<td>R2 ← 0</td>
</tr>
<tr>
<td><strong>ADD R2,R2,12</strong></td>
<td>R2 ← 12</td>
</tr>
<tr>
<td><strong>START BRz END</strong></td>
<td>If Z, goto END</td>
</tr>
<tr>
<td><strong>LDR R4,R1,0</strong></td>
<td>Load next value to R4</td>
</tr>
<tr>
<td><strong>ADD R3,R4,R3</strong></td>
<td>R3 ← R4 + R3</td>
</tr>
<tr>
<td><strong>ADD R1,R1,1</strong></td>
<td>Increment R1 (pointer)</td>
</tr>
<tr>
<td><strong>ADD R2,R2,-1</strong></td>
<td>Decrement R2 (counter)</td>
</tr>
<tr>
<td><strong>BRnzp START</strong></td>
<td>Goto START</td>
</tr>
<tr>
<td><strong>END</strong></td>
<td># done adding</td>
</tr>
</tbody>
</table>
Instructions

JMP

Jump is an unconditional branch -- *always* taken.
- Target is contents of a register, not a label.

LEA Ri, foo
JMP R1
LC-3 Subtraction

Immediate

\[ R3 = R2 - R1 \]

\[ \text{NOT R1, R1} \]

\[ \text{ADD R1, R1, 1} \]

\[ \text{ADD R3, R2, R1} \]
Print Single Digit Number

0 - 9  '0' - '9'

LD R0, ASCII\texttt{OFF} 7 -> '7'
ADD R0, R0, R1
OUT

ASCII\texttt{OFF}, FILL 48
LC-3 Multiply

3 \times 6 = 3 + 3 + 3 + 3 + 3 + 3

R3 = 3

AND RO, RO, 0

AND R1, R1, 0

ADD R1, R1, 6

R3 = 3

START

BRZ2 END

[ADD RO, RO, R3]

ADD R1, R1, -1

BRNZ2P START

END
LC-3 Integer Division

3x6 = 3.3.3... 16/3 = 5

16 - 3 = 13 > 0 1
13 - 3 = 10
10 - 3 = 7
7 - 3 = 4
4 - 3 = 1
3 - 3 = -2

[Drawing of a number line with arrows and numbers 2, 3, 4, 5]
LC-3
Assembly Language
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.
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>
The Assembler

- We are writing source code. We need to translate that to binary so it can be run on the LC-3.
- This is the job of the assembler

```
ADD R3, R3, 0
0110 110, 110, 0000
```
Memory

- Our program needs to be stored in our memory and it is placed there by the assembler.
- Each line of our code causes the assembler to store data at memory locations.
- Data is placed sequentially by instruction
  - Instructions themselves are encoded as 16-bit binary numbers.
.ORIG

• Tells simulator where to put your code in memory
• Does not use memory itself
• Only one allowed per program
• We start at this address
.ORIG in Memory

- .ORIG x3000
- ADD R1,R3,R7
- NOT R1,R1

<table>
<thead>
<tr>
<th>Address</th>
<th>Instruction</th>
<th>Memory (Hex)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x3000</td>
<td>ADD R1,R3,R7</td>
<td>x12C7</td>
</tr>
<tr>
<td>0x3001</td>
<td>NOT R1,R1</td>
<td>x927F</td>
</tr>
<tr>
<td>0x3002</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

0x0000 - 0x3000 Traps
“a typed language”

```plaintext
type varname;
```

- type is
  - `int` (integer)
  - `char` (character)
  - `float` (floating point)

“LC-3”

```plaintext
varname(really label) .FILL value
```

value is required — the initial value
.FILL

flag .FILL x0001
counter .FILL x2
letter .FILL x0041 ; A
letters .FILL -436

- 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
  \( \text{HALT} \)
.FILL in Memory

- .ORIG x3000
- ADD R1,R3,R7
- NOT R1,R1
- .FILL x0001
- .FILL x2
- .FILL -436

<table>
<thead>
<tr>
<th>Address</th>
<th>Instruction</th>
<th>Memory (Hex)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x3000</td>
<td>ADD R1,R3,R7</td>
<td>x12C7</td>
</tr>
<tr>
<td>0x3001</td>
<td>NOT R1,R1</td>
<td>x927F</td>
</tr>
<tr>
<td>0x3002</td>
<td>No Instruction</td>
<td>x0001</td>
</tr>
<tr>
<td>0x3003</td>
<td>NOP</td>
<td>x0002</td>
</tr>
<tr>
<td>0x3004</td>
<td>NOP</td>
<td>xFE4C</td>
</tr>
<tr>
<td>0x3005</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0x3006</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0x3007</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0x3008</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
.BLKW

- Tells assembler to set aside some number of sequential memory locations
- Useful for arrays
- Can be initialized
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 \textit{x4}.
Num .BLKW 1 \textit{x4}

; set aside 10 locations, label and initialize to 37.
Num .BLKW 10 \textit{37}
.BLKW in Memory

- .ORIG x3000
- ADD R1,R3,R7
- NOT R1,R1
- .BLKW 2
- .BLKW 4 6

<table>
<thead>
<tr>
<th>Address</th>
<th>Instruction</th>
<th>Memory (Hex)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x3000</td>
<td>ADD R1,R3,R7</td>
<td>x12C7</td>
</tr>
<tr>
<td>0x3001</td>
<td>NOT R1,R1</td>
<td>x927F</td>
</tr>
<tr>
<td>0x3002</td>
<td>NOP</td>
<td>x0000</td>
</tr>
<tr>
<td>0x3003</td>
<td>NOP</td>
<td>x0000</td>
</tr>
<tr>
<td>0x3004</td>
<td>NOP</td>
<td>x0006</td>
</tr>
<tr>
<td>0x3005</td>
<td>NOP</td>
<td>x0006</td>
</tr>
<tr>
<td>0x3006</td>
<td>NOP</td>
<td>x0006</td>
</tr>
<tr>
<td>0x3007</td>
<td>NOP</td>
<td>x0006</td>
</tr>
<tr>
<td>0x3008</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
**.STRINGZ**

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

Example:

```plaintext
hello .STRINGZ "Hello World!"
```
.STRINGZ in Memory

- .ORIG x3000
- ADD R1,R3,R7
- NOT R1,R1
- .STRINGZ "Foo\nBar\n"

<table>
<thead>
<tr>
<th>Address</th>
<th>Instruction</th>
<th>Memory (Hex)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x3000</td>
<td>ADD R1,R3,R7</td>
<td>x12C7</td>
</tr>
<tr>
<td>0x3001</td>
<td>NOT R1,R1</td>
<td>x927F</td>
</tr>
<tr>
<td>0x3002</td>
<td>NOP</td>
<td>x0046</td>
</tr>
<tr>
<td>0x3003</td>
<td>NOP</td>
<td>x006F</td>
</tr>
<tr>
<td>0x3004</td>
<td>NOP</td>
<td>x006F</td>
</tr>
<tr>
<td>0x3005</td>
<td>NOP</td>
<td>x0042</td>
</tr>
<tr>
<td>0x3006</td>
<td>NOP</td>
<td>x0061</td>
</tr>
<tr>
<td>0x3007</td>
<td>NOP</td>
<td>x0072</td>
</tr>
<tr>
<td>0x3008</td>
<td>NOP</td>
<td>x000A</td>
</tr>
<tr>
<td>0x3009</td>
<td>NOP</td>
<td>x0000</td>
</tr>
</tbody>
</table>
.END

- Tells the assembler where your program ends
- Only one allowed in your program
Simple LC-3 program

```
.ORIG x3000
LD R2, Zero
LD R0, M0
LD R1, M1
BRz
ADD
ADD
BR
ST
HALT
; same as a “TRAP x25”
.FILL x0000
.FILL x0000
.FILL x0004
.FILL x0002
.END
```

- What does this program do?
- What is in “Result” at the end?
HLL – if/else statements...

if (condition)
    statement;
else
    statement;
"LC-3"

"Generic" if count<0

count = count + 1;

greatzero ; next instruction goes here

LD BRpz
ADD R0, count

LD R0, R0, #1

CMPE-012/1/
Loops can be built out of IF’s – WHILE:

“Generic”

while count > 0) {
    a = a + count;
    count=count-1;
}
“LC-3”

LD R1, a
LD R0, count
while BRnz endwhile
ADD R1, R1, R0
ADD R0, R0, #-1
BR while
ST R1, a
ST R0, count
For loops

“general”

for $l = 3$ to $8$
    $a = a + l$
“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

for

ADD R2, R1, #-8

BRp endfor

ADD R0, R0, R1 ; a=a+l

ADD R1, R1, #1 ; l++

BR for ; same as BRnzp

endfor