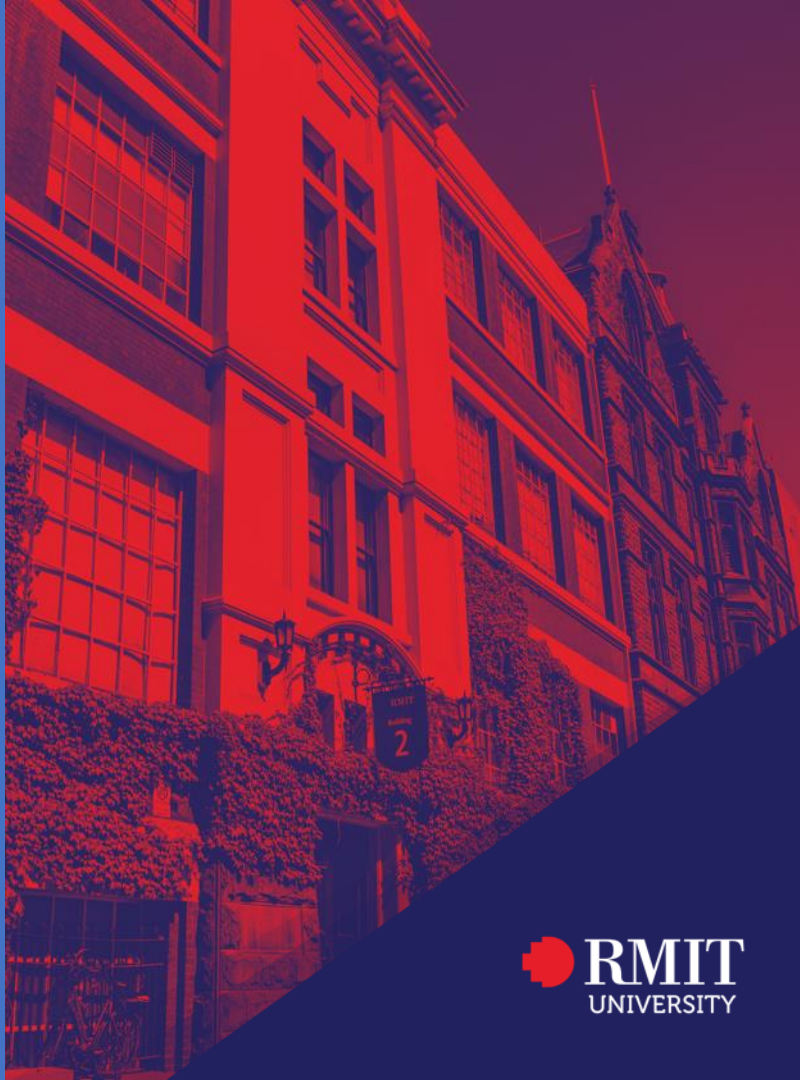
Two's complement, branching

Michael Dann
School of Computing Technologies

What's next...



RMIT UNIVERSITY

LC3 overview

Instruction Set

Op	Format	Description	Example
ADD	ADD DR, SR1, SR2 ADD DR, SR1, imm5	Adds the values in SR1 and SR2/imm5 and sets DR to that value.	ADD R1, R2, #5 The value 5 is added to the value in R2 and stored in R1.
AND	AND DR, SR1, SR2 AND DR, SR1, imm5	Performs a bitwise and on the values in SR1 and SR2/imm5 and sets DR to the result.	AND R0, R1, R2 A bitwise and is preformed on the values in R1 and R2 and the result stored in R0.
BR	BR(n/z/p) LABEL Note: (n/z/p) means any combination of those letters can appear there, but must be in that order.	Branch to the code section indicated by LABEL, if the bit indicated by (n/z/p) has been set by a previous instruction. n: negative bit, z: zero bit, p: positive bit. Note that some instructions do not set condition codes bits.	BRz LPBODY Branch to LPBODY if the last instruction that modified the condition codes resulted in zero. BRnp ALT1 Branch to ALT1 if last instruction that modified the condition codes resulted in a positive or negative (non-zero) number.
JMP	JMP SR1	Unconditionally jump to the instruction based upon the address in SR1.	JMP R1 Jump to the code indicated by the address in R1.
JSR	JSR LABEL	Put the address of the next instruction after the JSR instruction into R7 and jump to the subroutine indicated by LABEL.	JSR POP Store the address of the next instruction into R7 and jump to the subroutine POP.
JSRR	JSSR SR1	Similar to JSR except the address stored in SR1 is used instead of using a LABEL.	JSSR R3 Store the address of the next instruction into R7 and jump to the subroutine indicated by R3's value.
LD	LD DR, LABEL	Load the value indicated by LABEL into the DR register.	LD R2, VAR1 Load the value at VAR1 into R2.
LDI	LDI DR, LABEL	Load the value indicated by the address at LABEL's memory location into the DR register.	LDI R3, ADDR1 Suppose ADDR1 points to a memory location with the value x3100. Suppose also that memory location x3100 has the value 8. 8 then would be loaded into R3.
LDR	LDR DR, SR1, offset6	Load the value from the memory location found by adding the value of SR1 to offset6 into DR.	LDR R3, R4, #-2 Load the value found at the address (R4 –2) into R3.
LEA	LEA DR, LABEL	Load the address of LABEL into DR.	LEA R1, DATA1 Load the address of DATA1 into R1.
NOT	NOT DR, SR1	Performs a bitwise not on SR1 and stores the result in DR.	NOT R0, R1 A bitwise not is preformed on R1 and the result is stored in R0.
RET	RET	Return from a subroutine using the value in R7 as the base address.	RET Equivalent to JMP R7.
RTI	RTI	Return from an interrupt to the code that was interrupted. The address to return to is obtained by popping it off the supervisor stack, which is automatically done by RTI.	RTI Note: RTI can only be used if the processor is in supervisor mode.
ST	ST SR1, LABEL	Store the value in SR1 into the memory location indicated by LABEL.	ST R1, VAR3 Store R1's value into the memory location of VAR3.
STI	STI SR1, LABEL	Store the value in SR1 into the memory location indicated by the value that LABEL's memory location contains.	STI R2, ADDR2 Suppose ADDR2's memory location contains the value x3101. R2's value would then be stored into memory location x3101.
STR	STR SR1, SR2, offset6	The value in SR1 is stored in the memory location found by adding SR2 and offest6 together.	STR R2, R1, #4 The value of R2 is stored in memory location (R1 + 4).
TRAP	TRAP trapvector8	Performs the trap service specified by trapvector8. Each trapvector8 service has its own assembly instruction that can replace the trap instruction.	TRAP x25 Calls a trap service to end the program. The assembly instruction HALT can also be used to replace TRAP x25.

Covered already

Covered later

Covered today

Not covered

Unsigned Integers

An n-bit unsigned integer represents 2^n values: from 0 to 2^n-1:

2^2	2^1	2^0	
0	0	0	0
0	0	1	1
0	1	0	2
0	1	1	3
1	0	0	4
1	0	1	5
1	1	0	6
1	1	1	7

How can we represent negative numbers though?

- Assign half to non-negative integers (0 through 2^{n-1} - 1) and half to negative (-2^{n-1} through -1).
- Unsigned: 0, 1, 2, 3, 4, 5, 6, 7.
- Signed: 0, 1, 2, 3, -4, -3, -2, -1.
- With 16 bits, can represent [-32768, 32767].

Two’s Complement

We’d like subtraction to work the same way as adding a negative number, i.e.:

7 – 5 = 2

7 + (-5) = 2

Two’s complement representation was developed to make arithmetic easy to implement in circuits.

For each positive number (X), assign value to its negative (-X), such that:
X + (-X) = 0 with “normal” addition, ignoring arithmetic overflow.

Arithmetic overflow:

```
  1111111111111111
+ 0000000000000001
= 0000000000000000
```

Here we can see that 1111111111111111 = -1, since adding 1 to it gives 0.

Two's Complement

Observe that x + not(x) always yields 1111111111111111 (= -1).

Example:

```
       x    = 1011000101101011
    not(x)  = 0100111010010100
x + not(x)  = 1111111111111111
```

This gives us:

```
x + not(x) = -1
⇒ not(x) = -1 – x
⇒ not(x) + 1 = -x
⇒ -x = not(x) + 1
```

In other words, to calculate –x:

- Start with the positive number.
- Flip every bit.
- Add one.

This is referred to as "calculating the two's complement of x".

Two's Complement

Let's do some practice!

2.1 Problem Statement

The numbers X and Y are found at locations **x3120** and **x3121**, respectively. Write a program in LC-3 assembly language that does the following:

- Compute the difference $X - Y$ and place it at location **x3122**.

Starter code:

```
.ORIG x3000
; Put your code here!
HALT
.END
```

Two's complement:
-x = not(x) + 1

Branching

The n, z, p flags are set based on the result of the last arithmetic operation.

BR

Conditional Branch

Assembler Formats

BRn	LABEL	BRzp	LABEL
BRz	LABEL	BRnp	LABEL
BRp	LABEL	BRnz	LABEL
BR†	LABEL	BRnzp	LABEL

Encoding

15–12	11	10	9	8–0
0000	n	z	p	PCoffset9

Operation

```
if ((n AND N) OR (z AND Z) OR (p AND P))
   PC = PC‡ + SEXT(PCoffset9);
```

Description

The condition codes specified by the state of bits [11:9] are tested. If bit [11] is set, N is tested; if bit [11] is clear, N is not tested. If bit [10] is set, Z is tested, etc. If any of the condition codes tested is set, the program branches to the location specified by adding the sign-extended PCoffset9 field to the incremented PC.

Examples

BRzp	LOOP	; Branch to LOOP if the last result was zero or positive.
BR†	NEXT	; Unconditionally branch to NEXT.

Branching

Let's do some more practice!

2.1 Problem Statement

The numbers X and Y are found at locations **x3120** and **x3121**, respectively. Write a program in LC-3 assembly language that does the following:

- Compute the difference $X - Y$ and place it at location **x3122**.
- Place the absolute values $|X|$ and $|Y|$ at locations **x3123** and **x3124**, respectively.
- Determine which of $|X|$ and $|Y|$ is larger. Place 1 at location **x3125** if $|X|$ is, a 2 if $|Y|$ is, or a 0 if they are equal.

Traps

- A *trap* is essentially just some predefined subroutine.

- Two of the main instructions that we've used so far are aliases for traps:

  - **`PUTS = TRAP 0x22`**
  - **`HALT = TRAP 0x25`**

- That is, you can replace all instances of `PUTS` and `HALT` with `TRAP 0x22` and `TRAP 0x25` respectively, and your program will assemble to the same instructions in binary.

- We have added several custom traps (from `0x27` to `0x2D`) to provide access to some of the Minecraft APIs. See the Assignment 2 specification for the details of these traps.

Example using the Minecraft traps

- The program below places a stone block 5 units above the player, then teleports the player so that they are standing on top of the new stone block.

```
.ORIG x3000
TRAP 0x29         ; Store the player's tile position in R0, R1, R2
ADD R1, R1, #5    ; Add 5 to the y-coordinate
AND R3, R3, #0    ; Clear R3
ADD R3, R3, #1    ; Store value 1 in R3 (== block.STONE)
TRAP 0x2C         ; Set block
ADD R1, R1, #1    ; Add 1 to the y-coordinate
TRAP 0x2A         ; Put the player here
HALT
.END
```