

# Introduction to LC-3 assembly

Michael Dann

School of Computing Technologies

---

What's next...



# LC-3 Assembly

- Assignments 1 & 2 focus on LC-3 assembly programming.
- For some of you, LC-3 may be a steep learning curve. You'll need to get used to thinking about programming problems at a lower-level.
- If you get through Labs 1 – 6 in the manual, you should be well-placed to attempt most problems in the assignment.

# 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 performed 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.
JSSR	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.



Covered already



Covered later



Covered today



Not covered

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 performed 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 offset6 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.

# “Hello World!” in LC-3

This directive specifies the address where the program should be stored in memory. The "default" value is x3000. (And note that the address is specified in hexadecimal.)

```
1 ; LC-3 Program that displays
2 ; "Hello World!" to the console
3 .ORIG    x3000
4 LEA      R0, HW ; load address of string
5 PUTS     ; output string to console
6 HALT     ; end program
7 HW       .STRINGZ "Hello World!"
8 .END
```

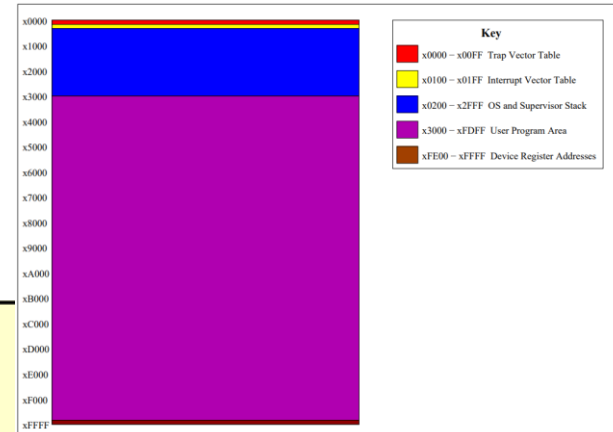


Figure 1: LC-3 memory map: the various regions.

Note that there is no argument to PUTS. It assumes that the address of the string to be printed is in register 0.

“HW” is an arbitrary *label*. We could change it to anything else, e.g., “hello\_world\_str”, and the program would still do the same thing (provided we also update line 4 to match).

# Clearing a register

- When the LC-3 web simulator first starts, the register values are set to 0.
- However, it's bad practice to assume that the registers will always hold 0 when the program begins (since other programs may have run beforehand).
- To clear out a register, we can use the AND instruction:

AND R0, R0, #0

↑  
Destination register, i.e.,  
the register where the  
result should be stored.

↙ ↘  
The operands, i.e., the  
values that the AND is  
performed on.

From the lab manual:

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 performed on the values in R1 and R2 and the result stored in R0.
-----	---------------------------------------	---	--

**Question:** Can you guess why this works, i.e., why is the result always zero?

# Addition

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.

.ORIG x3000

AND R0, R0, #0 ; Clear out register 0

AND R1, R1, #0 ; Clear out register 1

ADD R0, R0, #2 ; Calculate  $R0 + 2$ , store result in R0

ADD R1, R1, xF ; Calculate  $R1 + 15$ , store result in R1

ADD R2, R0, R1 ; Calculate  $R0 + R1$ , store result in R2

HALT

.END

# Loading a number from a label

```
.ORIG x3000
```

```
LD R0, FIRST_NUMBER
```

```
LD R1, SECOND_NUMBER
```

```
ADD R2, R0, R1
```

```
HALT
```

```
FIRST_NUMBER .FILL #10
```

```
SECOND_NUMBER .FILL #7
```

; Use .FILL to store a number at  
; a labelled memory address.

```
.END
```

# Loading from a specific address

```
.ORIG x3000
```

```
LDI R0, NUMBER_ADDRESS      ; Note we're using LDI this time,  
                              ; not LD. Instead of loading the  
ADD R1, R0, #2               ; value x3100 into R0, this  
                              ; instruction loads the value at  
HALT                         ; memory address x3100 into R0.
```

```
NUMBER_ADDRESS .FILL x3100
```

```
.END
```



# Storing

```
.ORIG x3000
```

```
AND R0, R0, #0
```

```
ADD R0, R0, #5
```

```
ST R0, SOME_LABEL
```

```
ADD R0, R0, #2
```

```
STI R0, STORAGE_ADDRESS
```

```
HALT
```


```
SOME_LABEL .FILL #0
```

```
STORAGE_ADDRESS .FILL x3100
```

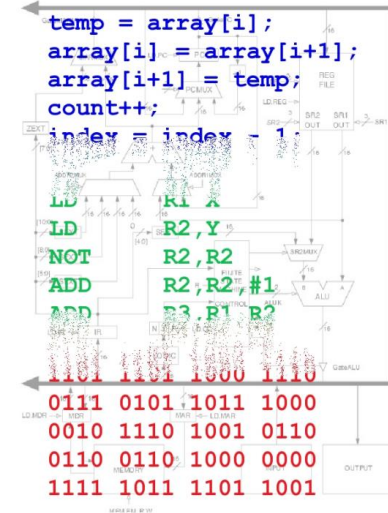
```
.END
```

ST and STI do essentially the same things as LD and LDI, except that they store instead of loading.

# Getting Started with LC-3

- See this link on the front page of Canvas:  
[Getting Started with LC-3](#). This page explains how to set up an LC-3 development environment for Assignment #2.
- Download a local copy of the lab manual. 
- Familiarise yourself with the web simulator.
- Start on the exercises in the lab manual. We'll go through specific examples in class by popular request, and post solutions periodically.

## LC-3 Assembly Language A Manual



George M. Georgiou and Brian Strader

California State University, San Bernardino

August 2005