

# CENG3420

## Lab 2-1: LC-3b Simulator

**Bei Yu**

Department of Computer Science and Engineering  
The Chinese University of Hong Kong

[byu@cse.cuhk.edu.hk](mailto:byu@cse.cuhk.edu.hk)

Spring 2019



香港中文大學

The Chinese University of Hong Kong

# Overview

LC-3b Basis

LC-3b Assembly Examples

LC-3b Simulator

Task



# Overview

LC-3b Basis

LC-3b Assembly Examples

LC-3b Simulator

Task



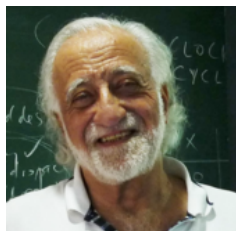
# Assembler & Simulator

- ▶ Assembly language – symbolic (MIPS, LC-3b, ...)
- ▶ Machine language – binary
  
- ▶ **Assembler** is a program that
  - ▶ turns symbols into machine instructions.
  - ▶ EX: `lc3b_asm`, SPIM, ...
  
- ▶ **Simulator** is a program that
  - ▶ mimics the behavior of a processor
  - ▶ usually in high-level language
  - ▶ EX: `lc3b_sim`, SPIM, ...



# LC-3b

- ▶ LC-3b: **Little Computer 3, b** version.
- ▶ Relatively simple instruction set
- ▶ Most used in teaching for CS & CE
- ▶ Developed by Yale Patt@UT & Sanjay J. Patel@UIUC



# LC-3 Architecture

- ▶ RISC – only 15 instructions
- ▶ 16-bit data and address
- ▶ 8 general-purpose registers (GPR)

Plus 4 special-purpose registers:

- ▶ Program Counter (PC)
- ▶ Instruction Register (IR)
- ▶ Condition Code Register (CC)
- ▶ Process Status Register (PSR)



# Memory

$2^k \times m$  array of stored bits:

## Address

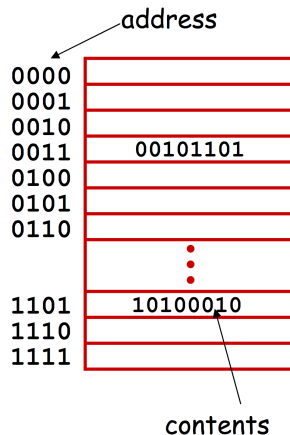
- ▶ unique (k-bit) identifier of location
- ▶ LC-3:  $k = 16$

## Contents

- ▶ m-bit value stored in location
- ▶ LC-3:  $m = 16$

## Basic Operations:

- ▶ READ (Load): value in a memory location  $\rightarrow$  the Processor
- ▶ WRITE (Store): value in the Processor  $\rightarrow$  a memory location



# Interface to Memory

How does the processing unit get data to/from memory?

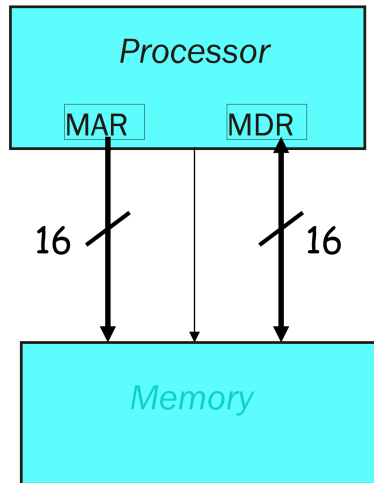
- ▶ **MAR**: Memory Address Register
- ▶ **MDR**: Memory Data Register

## To LOAD from a location (A):

1. Write the address (A) into the MAR.
2. Send a “read” signal to the memory.
3. Read the data from MDR.

## To STORE a value (X) into a location (A):

1. Write the data (X) to the MDR.
2. Write the address (A) into the MAR.
3. Send a “write” signal to the memory.





# CPU-only Tasks

In addition to input & output a program also:

- ▶ Evaluates arithmetic & logical functions to determine values to assign to variable.
- ▶ Determines the order of execution of the statements in the program.
- ▶ In assembly this distinction is captured in the notion of **arithmetic/logical**, and **control** instructions.



# Processing Unit

## Functional Units:

- ▶ ALU = Arithmetic/Logic Unit
- ▶ could have many functional units.
- ▶ some of them special-purpose (floating point, multiply, square root, . . .)

## Registers

- ▶ Small, temporary storage
- ▶ Operands and results of functional units
- ▶ LC-3 has eight registers (R0, . . . , R7), each 16 bits wide

## Word Size

- ▶ number of bits normally processed by ALU in one instruction
- ▶ also width of registers
- ▶ LC-3 is 16 bits



# Instructions

The instruction is the fundamental unit of work.

## Specifies two things:

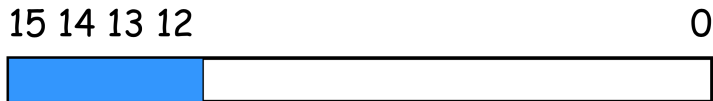
- ▶ opcode: operation to be performed
- ▶ Operands: data/locations to be used for operation

Three basic kinds of instructions:

- ▶ Computational instructions
- ▶ Data-movement instructions
- ▶ Flow-control instructions



# Instruction Encoding

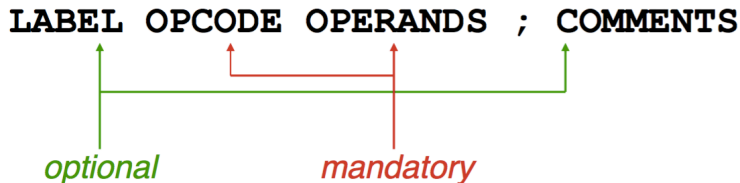


- ▶ in LC-3, the most-significant **four** bits contain the instruction's OPCODE always.
- ▶ The meaning of the other bits changes according to the instruction.
- ▶ Look up the “[LC-3b-ISA.pdf](#)” find all 16 instruction format descriptions



# LC-3b Instructions

- ▶ 16 bit instruction
- ▶ Memory address space is 16 bits  $\rightarrow 2^{16}$  locations
- ▶ Each memory address containing one byte (**eight** bits).
- ▶ One instruction or declaration per line



# LC-3 v.s. MIPS

## LC-3

1. 16 bit
2. NO floating point instruction
3. 8 registers
4. NO hardwired register value
5. Only has AND, NOT, and ADD

## MIPS

1. 32 bit
2. Floating point instruction
3. 32 registers
4. \$0 is hardwired to 0
5. Full complement of arithmetic, logical, and shift operations



# Overview

LC-3b Basis

LC-3b Assembly Examples

LC-3b Simulator

Task



## LC-3b Example 1: Do nothing

“./lc3b\_asm nop.asm nop.cod”

nop.asm:

```
.ORIG x3000  
NOP  
NOP  
.END
```

nop.cod:

```
0x3000  
0x0000  
0x0000
```

- ▶ NOP instruction translates into machine code 0x0000.





# Assembler Directives

- ▶ Directives give information to the assembler
- ▶ Not executed by the program
- ▶ All directives start with a period '.'

<code>.ORIG</code>	Where to start in placing things in memory
<code>.FILL</code>	Declare a memory location (variable)
<code>.END</code>	Tells assembly where your program source ends



# Assembler Directives: `.ORIG`

- ▶ Tells where to put code in memory ([starting location](#))
- ▶ Only one `.ORIG` allowed per program module
- ▶ `PC` is set to this address at start up
- ▶ Similar to the `main()` function in C
- ▶ **Example:**

```
.ORIG x3000
```



# Assembler Directives: `.FILL`

- ▶ Declaration and initialization of variables
- ▶ Always declaring words
- ▶ **Examples:**

```
flag      .FILL    x0001
counter   .FILL    x0002
letter    .FILL    x0041
letters   .FILL    x4241
```



# Assembler Directives: `.END`

- ▶ Tells the assembler where your program ends
- ▶ Only one `.END` allowed in your program module
- ▶ **NOT** where the execution stops!



## LC-3b Example 2: Count from 10 to 1

count10.asm:

```
.ORIG x3000
LEA R0, TEN
LDW R1, R0, #0
START ADD R1, R1, #-1
      BRZ DONE
      BR START

DONE  TRAP x25
TEN   .FILL x000A
      .END
```

count10.cod:

```
0x3000
0xE005
0x6200
0x127F
0x0401
0x0FFD

0xF025
0x000A
```

► More explanations will be in Lab2-2.



# Overview

LC-3b Basis

LC-3b Assembly Examples

LC-3b Simulator

Task



## LC-3b Simulator: `lc3b_sim`

- ▶ Download from course website ([lab2-assignment.tar.gz](http://lab2-assignment.tar.gz))
- ▶ The simulator will
  - ▶ Execute the input LC-3b program
  - ▶ One instruction at a time
  - ▶ Modify the architectural state of the LC-3b
- ▶ Two main sections: the `shell` and the `simulation routines`
- ▶ Only need to work on simulation routine part.



## LC-3b Shell

```
./lc3b_sim [cod file]
```

```
LC-3b-SIM> ?  
  
-----LC-3b ISIM Help-----  
go           - run program to completion  
run n        - execute program for n instructions  
mdump low high - dump memory from low to high  
rdump        - dump the register & bus values  
?            - display this help menu  
quit        - exit the program  
  
LC-3b-SIM> █
```





## LC-3b Architecture State

- ▶ Please refer to [LC-3b\\_ISA](#) for more details
- ▶ PC
- ▶ General purpose registers (REGS): 8 registers
- ▶ Condition codes: **N** (negative); **Z** (zero); **P** (positive).

```
65 #define LC_3b_REGS 8
66
67 /* Data Structure for Latch */
68 typedef struct System_Latches_Struct{
69     int PC,           /* program counter */
70     N,               /* n condition bit */
71     Z,               /* z condition bit */
72     P;               /* p condition bit */
73     int REGS[LC_3b_REGS]; /* register file. */
74 } System_Latches;
75
76 System_Latches CURRENT_LATCHES, NEXT_LATCHES;
```



# LC-3b Memory Structure

```
46 /*****  
47 /* Main memory. */  
48 /*****  
49 /* MEMORY[A][0] stores the least significant byte of word at word address A  
50    MEMORY[A][1] stores the most significant byte of word at word address A  
51 */  
52  
53 #define WORDS_IN_MEM    0x08000  
54 int MEMORY[WORDS_IN_MEM][2];  
55
```

Two word-aligned locations are to store one 16-bit word.

- ▶ addresses differ only in bit 0
- ▶ Locations `x0006` and `x0007` are word-aligned

```
276 void init_memory() {  
277     int i;  
278  
279     for (i=0; i < WORDS_IN_MEM; i++) {  
280         MEMORY[i][0] = 0;  
281         MEMORY[i][1] = 0;  
282     }  
283 }  
284
```



# How to use LC-3b Simulator?

1. Compile your C codes through `make` command.
2. Run the compiled LC-3b simulator through `./lc3b_sim2 bench/xxx.cod`.  
Here the parameter is a machine code file.
3. In the simulator, run “n” instructions. When  $n = 3$ , run `3`
4. In the simulator, print out register information: `rdump`



# Overview

LC-3b Basis

LC-3b Assembly Examples

LC-3b Simulator

Task



# Lab2 Task 1

## architectural state:

- ▶ In `process_instruction()`, **update** `NEXT_LATCHES`
- ▶ At this moment, only update (increase PC value)

## memory:

- ▶ Given `CURRENT_LATCHES.PC`, read related word in memory
- ▶ Implement function `int memWord (int startAddr)`



# Task 1 Golden Results: nop.cod

## Output after run 2

```
process_instruction() | curInstr = 0x0000  
process_instruction() | curInstr = 0x0000
```

## Output after rdump:

```
Instruction Count : 2  
PC                : 0x3004  
CCs: N = 0  Z = 1  P = 0  
Registers:  
0: 0x0000  
1: 0x0000  
2: 0x0000  
3: 0x0000  
4: 0x0000  
5: 0x0000  
6: 0x0000  
7: 0x0000
```



# Task 1 Golden Results: count10.cod

## Output after run 7:

```
process_instruction() | curInstr = 0xe005
process_instruction() | curInstr = 0x6200
process_instruction() | curInstr = 0x127f
process_instruction() | curInstr = 0x0401
process_instruction() | curInstr = 0x0ffd
process_instruction() | curInstr = 0xf025
Simulator halted
```

## Output after rdump:

```
Instruction Count : 6
PC                : 0x0000
CCs: N = 0  Z = 1  P = 0
Registers:
0: 0x0000
1: 0x0000
2: 0x0000
3: 0x0000
4: 0x0000
5: 0x0000
6: 0x0000
7: 0x300c
```



# Task 1 Golden Results: toupper.cod

Output after run 18:

```
process_instruction() | curInstr = 0xe00f
process_instruction() | curInstr = 0x6000
process_instruction() | curInstr = 0x6000
process_instruction() | curInstr = 0xe20d
process_instruction() | curInstr = 0x6240
process_instruction() | curInstr = 0x6240
process_instruction() | curInstr = 0x2400
process_instruction() | curInstr = 0x0406
process_instruction() | curInstr = 0x14b0
process_instruction() | curInstr = 0x14b0
process_instruction() | curInstr = 0x3440
process_instruction() | curInstr = 0x1021
process_instruction() | curInstr = 0x1261
process_instruction() | curInstr = 0x0ff8
process_instruction() | curInstr = 0x3440
process_instruction() | curInstr = 0xf025
Simulator halted
```





# Task 1 Golden Results: toupper.cod (cont.)

Output after rdump:

```
Instruction Count : 16
PC                : 0x0000
CCs: N = 0  Z = 1  P = 0
Registers:
0: 0x0000
1: 0x0000
2: 0x0000
3: 0x0000
4: 0x0000
5: 0x0000
6: 0x0000
7: 0x3020
```

