Final Project

Logic Design Lab.
Spring 2023

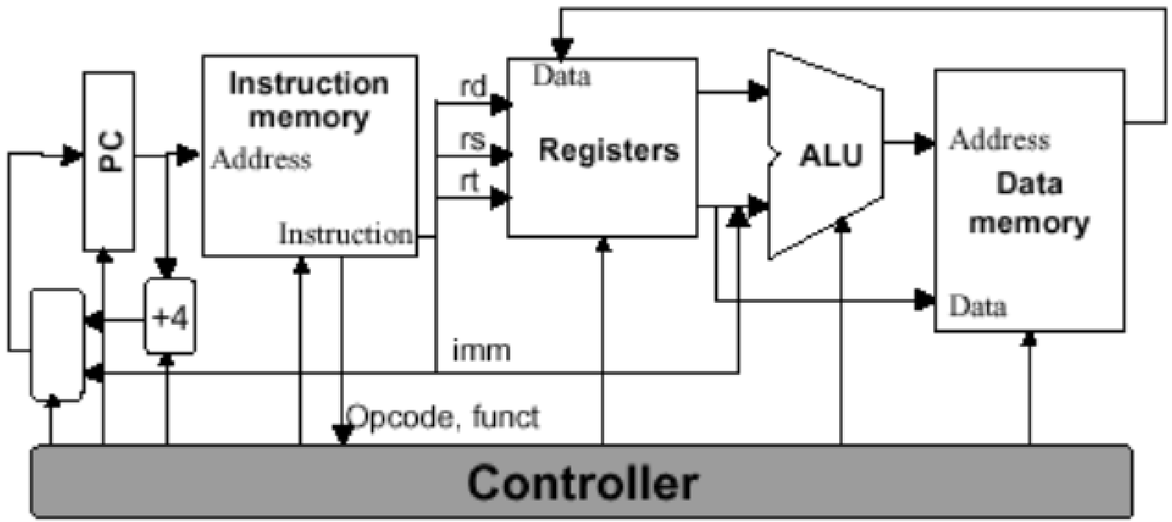
Prof. Chang-Gun Lee
(cglee@snu.ac.kr)
TA. Seonghyeon Park
TA. Jihwan Kim
TA. Hoyong Lee
(rubis.ld.ta@gmail.com)

Contents

Project Overview

- Implement a Simple Microprocessor in Verilog and program it on FPGA. Mimic a real computer.
- Your FPGA implementation should consist of an ALU, a control unit, a system memory, registers, and so on.
- Use the on-board 50MHz clock oscillator to create 1Hz clock (1 tick per second).

Computer Architecture (w/o pipeline)



- Execute instruction and update register, data memory every clock cycle
- The 32-bit (64-bit) per memory address can be used as instruction or data
- CPU (Controller) decode instruction (load/store/add/jump/…)

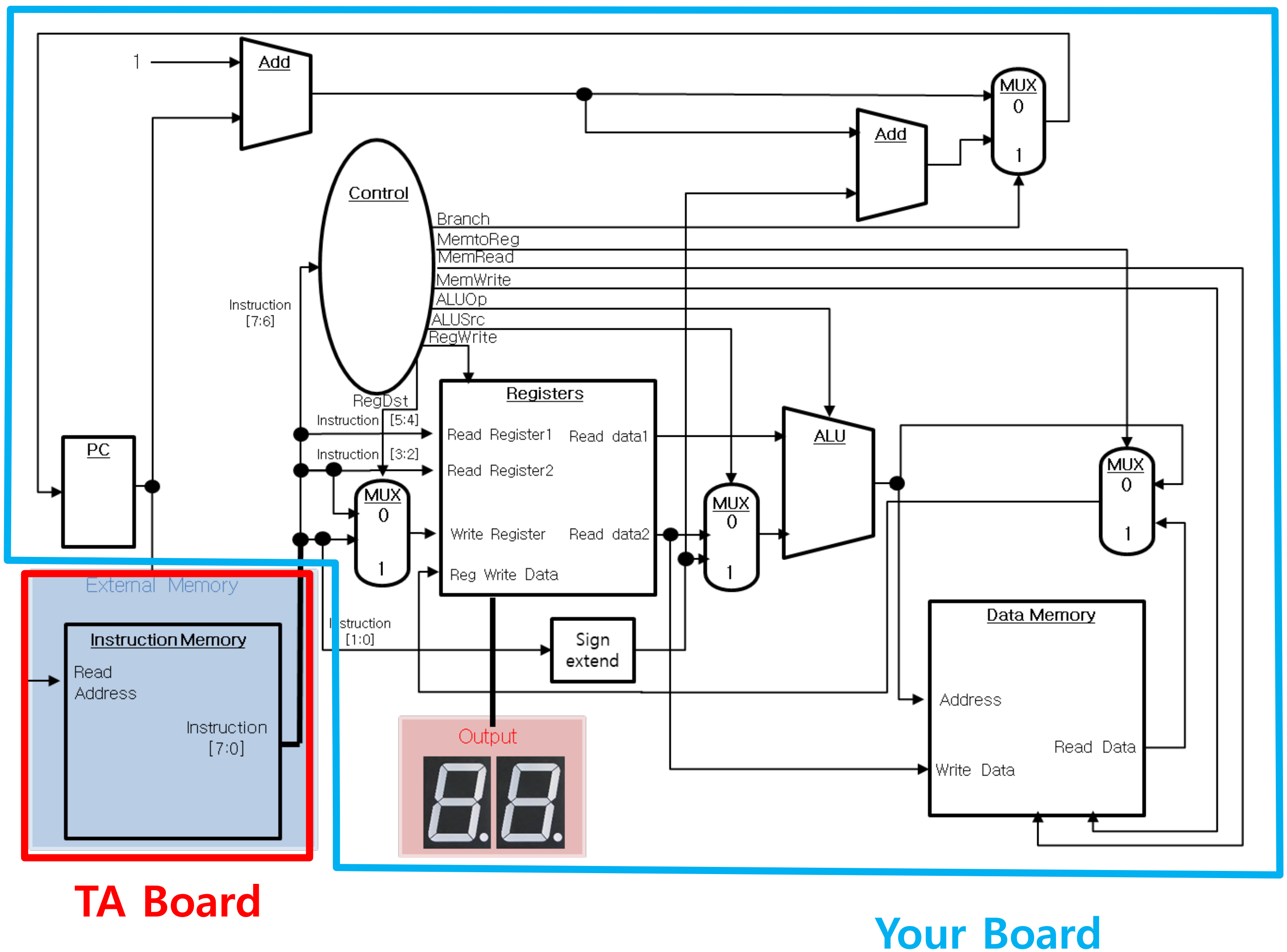
Project Overview

- 8-bit Microprocessor
- Instruction size : 8-bit
- Type of instructions : 4 (add, load, store, jump)
- Register size : 8-bit
- # of registers : 4

- Input : Instruction codes from external memory (TA's FPGA)
  - ✓ TA will test the result with the memory implemented in another FPGA chip. Several sets of instruction codes will be tested. Specific pin assignments between your microprocessor and instruction memory will be given.

- Output : Current value of Reg Write Data.
  - ✓ Use 7 segment displays. Display the value in **Hexadecimal**.

Microprocessor Design – Data Path



Microprocessor Design – ISA

- Formats of the entire instruction set

(8bit processor)

* op	operation code
rd	destination register
rs	source register
rt	source register two
imm	immediate (constant)
op, rd, rs, rt	unsigned value
imm	signed value

* Registers (8bits)
· Program Counter(PC)
· $s0 - $s3

* **2's complement** range : -2 ~ 1
· Instruction Mem. address range: 0x00-0xFF
· Data Mem. address range: 0x00-0xFF

Machine Code

CMD	**Type**	**Example**				**Meaning (Assembly Code)**	**Meaning (in C language)**	**Description**
		7 6	**5 4**	**3 2**	**1 0**			
	R	op	rs	rt	rd			
	I	op	rs	rt	imm			
	J	op			imm			
add	R	0	1	2	0	add $s0, $s1, $s2	$s0 = $s1 + $s2	3 operands, add with registers
lw	I	1	3	0	0	lw $s0, 0[$s3]	$s0 = Mem[$s3+0]	transfer value from memory to reg.
sw	I	2	3	0	1	sw $s0, 1[$s3]	Mem[$s3+1] = $s0	transfer value from reg. to mem.
j	J	3			-2	j -2	go to (next PC -2)	jump to next PC with an offset

Microprocessor Design – ISA

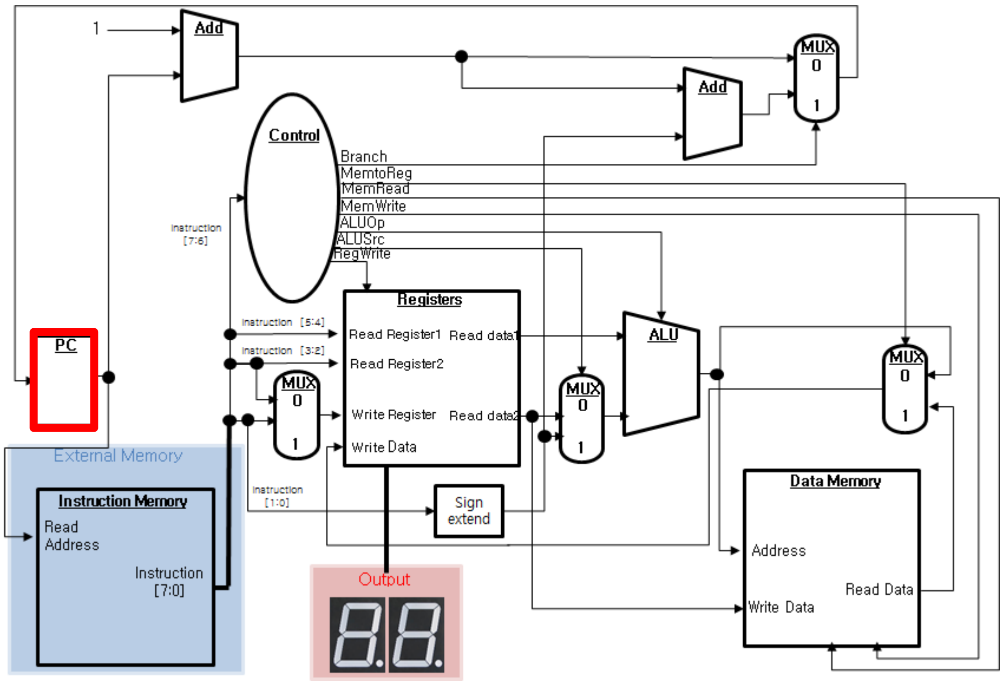
- Control Signal Table

Signal Name	Effect when deasserted (0)	Effect when asserted (1)
RegDst	The register file destination number for the Write regi ster comes from the rt field (bits3-2)	The register file destination number for the Write regi ster comes from the rd field (bits1-0)
RegWrite	None	Write result to rd, rt
ALUSrc	The second ALU operand comes from the second regi ster file output (Read data 2)	The second ALU operand is the sign-extended, lower 2 bits of the instruction
Branch	The PC is repaced by the output of the adder that co mputes the value of PC + 1	The PC is repaced by the output of the adder that co mputes the branch target
MemRead	None	Data memory contents designated by the address inp ut are put on the Read data output
MemWrite	None	Data memory contents designated by the address inp ut are replaced by the value on the Write data input
MemtoReg	The value fed to the register Write data input comes f rom the ALU	The value fed to the register Write data input comes f rom the data memory
ALUOp	None	Choose ALU operation (in this project, only Add operation exists. So this sign al is not effective, but is for later expandability)

Instruction	RegDst	RegWrite	ALUSrc	Branch	MemRead	MemWrite	MemtoReg	ALUOP
R-format	1	1	0	0	0	0	0	1
lw	0	1	1	0	1	0	1	0
sw	x	0	1	0	0	1	x	0
j	x	0	0	1	0	0	x	0

David A. Patterson, John L. Hennessy. (2005). *Computer Organization and Design* (3rd ed., pp.306). Morgan Kaufmann.

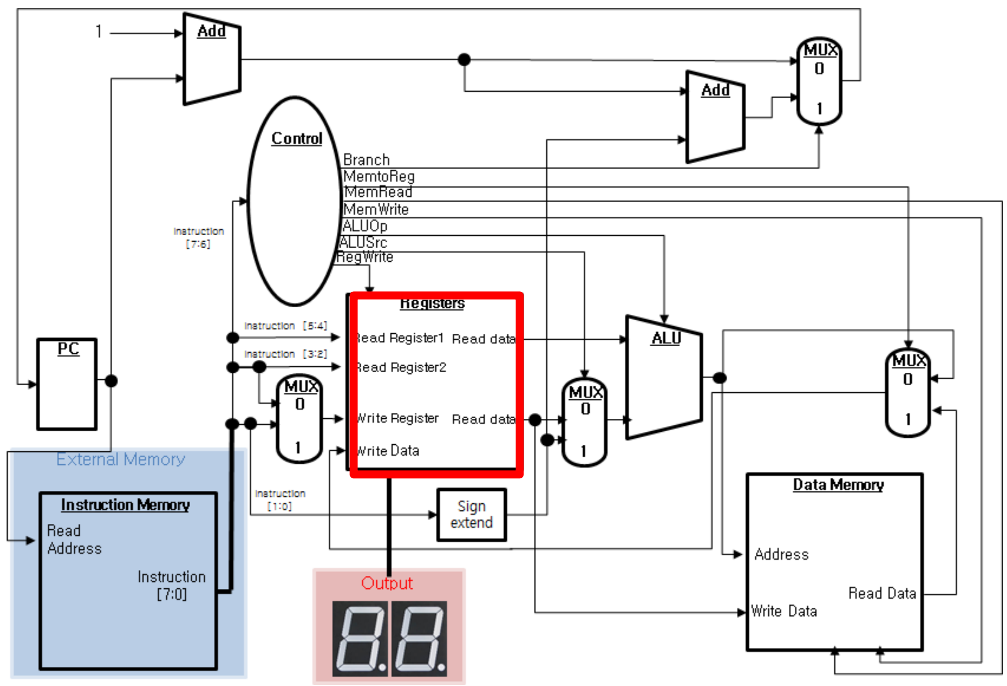
Microprocessor Design – Components



- Program Counter

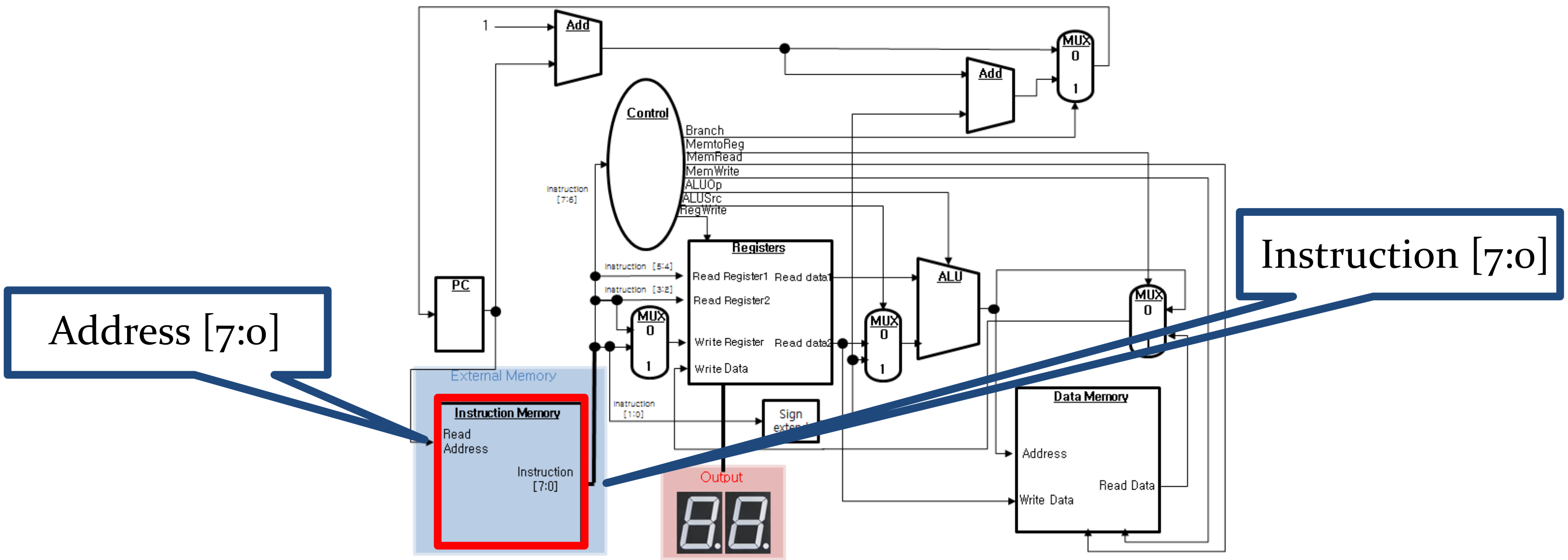
  Indicates the address of instructions. Initially, PC is set to start address as zero.

Microprocessor Design – Components



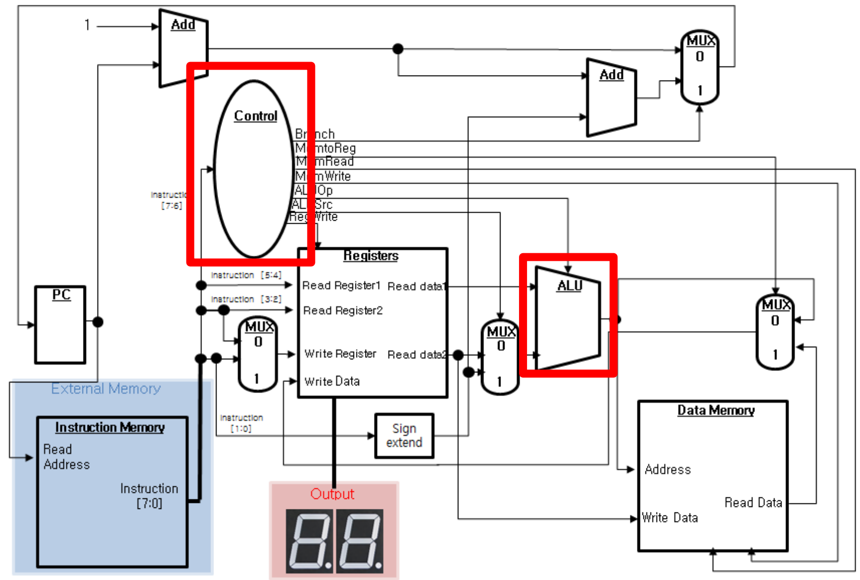
- Register
  - ✓ Consists of four 8-bit general purpose Registers.
  - ✓ Write data to Register or output the read data from Register according to control inputs.
  - ✓ Initialize register values to zero.

Microprocessor Design – Components



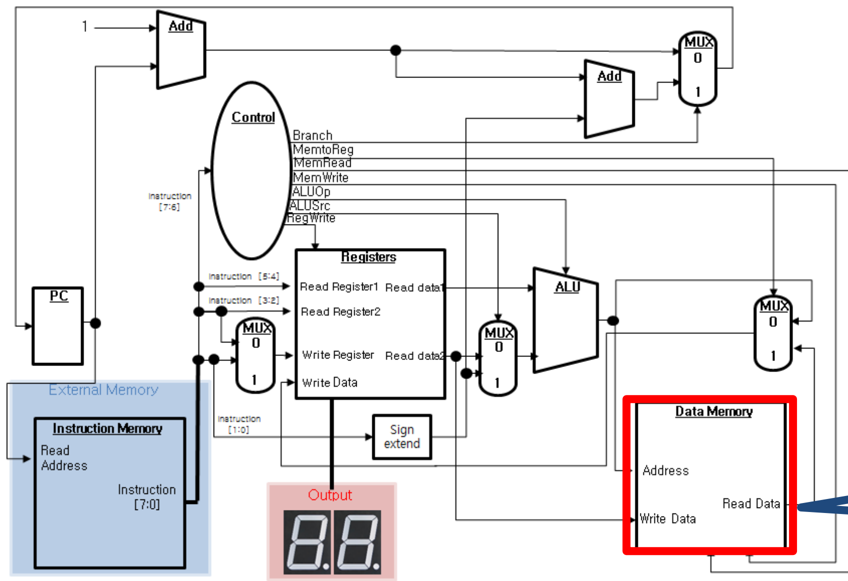
- Instruction Memory
  - ✓ Instruction size : 8-bit
  - ✓ Type of instructions : 4 (add, load, store, jump)
  - ✓ It has pre-defined instructions. Each instruction is 8-bit long. 2 MSBs are the instruction, remaining LSBs are used for different purpose. (Refer to ISA format)
  - ✓ # of input ports : 8 (8-bit Instruction Address)
  - ✓ # of output ports : 8 (8-bit Instruction Register)
  - ✓ Assume Instruction Memory for test. Instruction Memory will be given by external memory on evaluation by TA.

Microprocessor Design – Components



- Control Unit
  - ✓ Manages the process of moving data and instruction.
  - ✓ Get 2-bit operation as input and outputs 1-bit control signals (Refer to Control Signal Table)
- ALU
  - ✓ Performs add operation (don't care **overflow**)
  - ✓ Two operands (such as Rs, Rt, offset of I-type instructions)

Microprocessor Design – Components

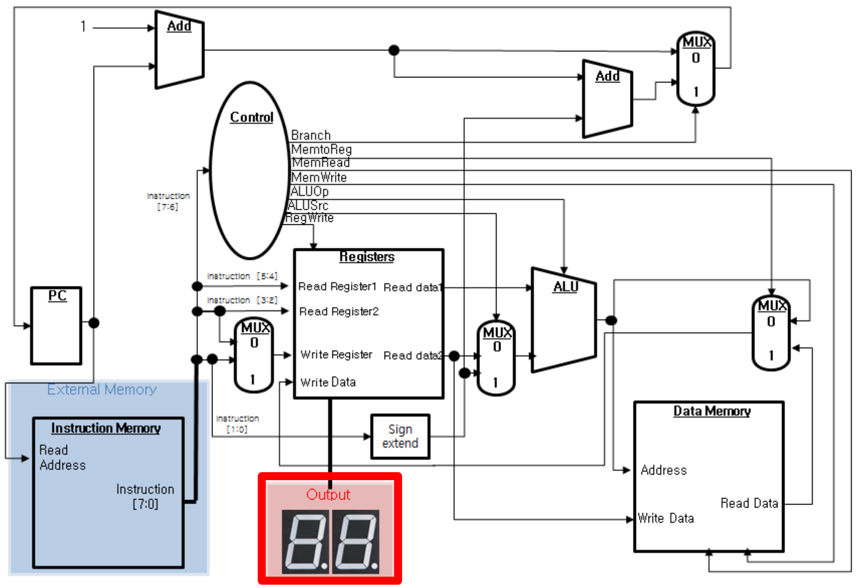


Data Memory

[0] 0	[16] 0
[1] 1	[17] -1
[2] 2	[18] -2
[3] 3	[19] -3
…	…
[14] 14	[30] -14
[15] 15	[31] -15

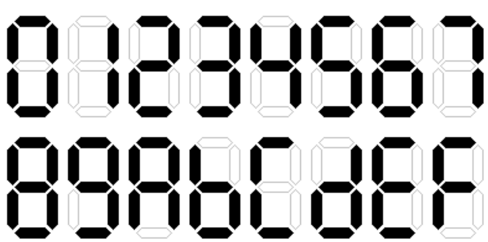
- Data Memory
  - ✓ 8-bit address, # of data : 256
    But, because of the total capacity of the FPGA we use, we will use only 32 of 8-bit registers (0x00 ~ 0x1F)
  - ✓ Initialize the data memory like above. This means, when you press 'Reset' button, data should be like above.
    (hint: use always block sensitive to reset signal.)

Microprocessor Design – Components

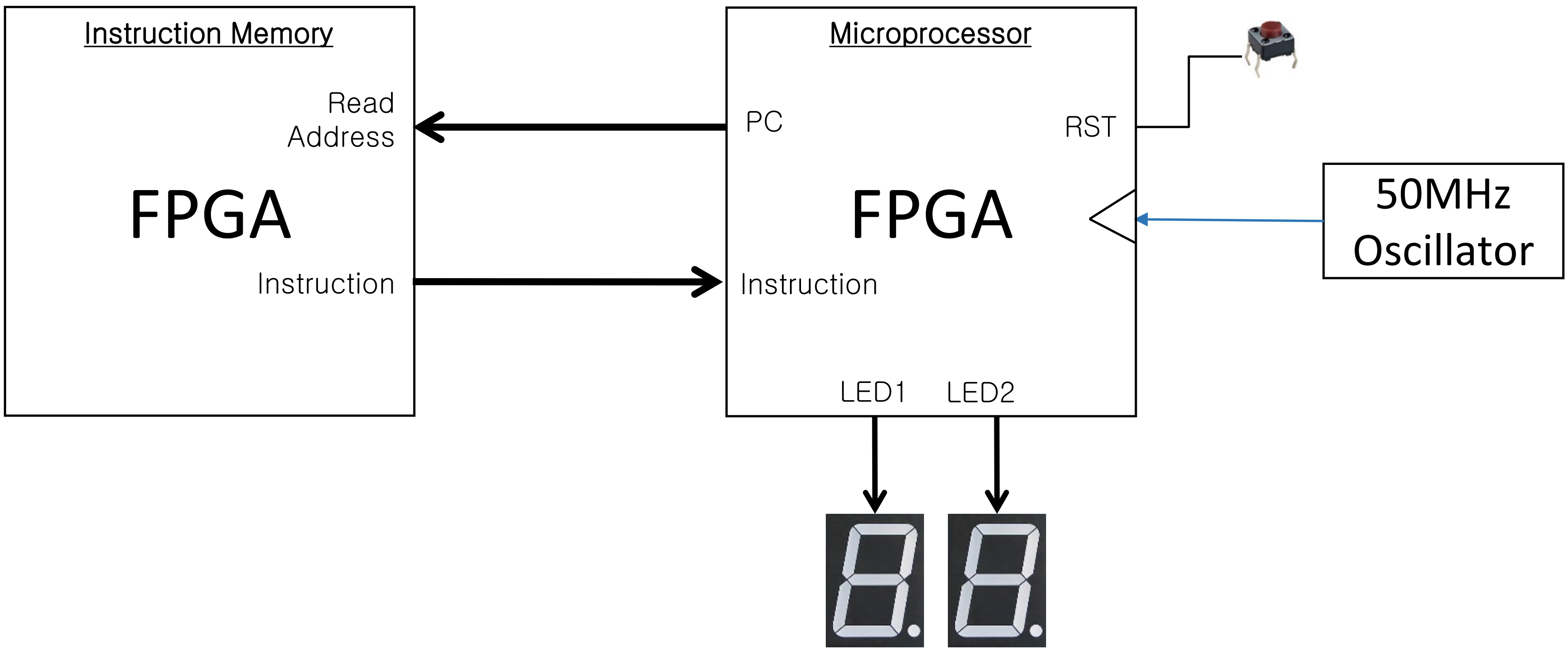


- Output 7 segment display

  Two 7 segment displays that show the contents of Register for Reg Write Data in <u>**Hexadecimal**</u>.



Microprocessor Design – Components



Example Test Set : Input

3rd instruction is skiped (jumped)

The actual machine instructions stored in Instruction Memory

signed value in 2's complement. Refer to ISA

Instructions

```
[0] lw $s2, 1($s0)        // 2'b01 2'b00 2'b10 2'b01
[1] j + 1                 // 2'b11 2'b00 2'b00 2'b01
[2] add $s0, $s1, $s2     // 2'b00 2'b01 2'b10 2'b00
[3] sw $s2, 1($s2)        // 2'b10 2'b10 2'b10 2'b01
[4] lw $s3, 1($s0)        // 2'b01 2'b00 2'b11 2'b01
```

Data stored in Data Memory

Register value

Data Memory

[0] 0
[1] 1
[2] 2
[3] 3
⋮

Register

[0] 0
[1] 0
[2] 0
[3] 0

Example Test Set : Input

Instructions

```
[0] lw $s2, 1($s0)        // 2'b01 2'b00 2'b10 2'b01     $s2 = Mem[$s0+1]
[1] j + 1                 // 2'b11 2'b00 2'b00 2'b01
[2] add $s0, $s1, $s2     // 2'b00 2'b01 2'b10 2'b00
[3] sw $s2, 1($s2)        // 2'b10 2'b10 2'b10 2'b01     Mem[$s2+1] = $s2
[4] lw $s3, 1($s0)        // 2'b01 2'b00 2'b11 2'b01     $s3 = Mem[$s0+1]
```

PC

5

Data Memory

[0] 0
[1] 1
[2] 1
[3] 3
⋮

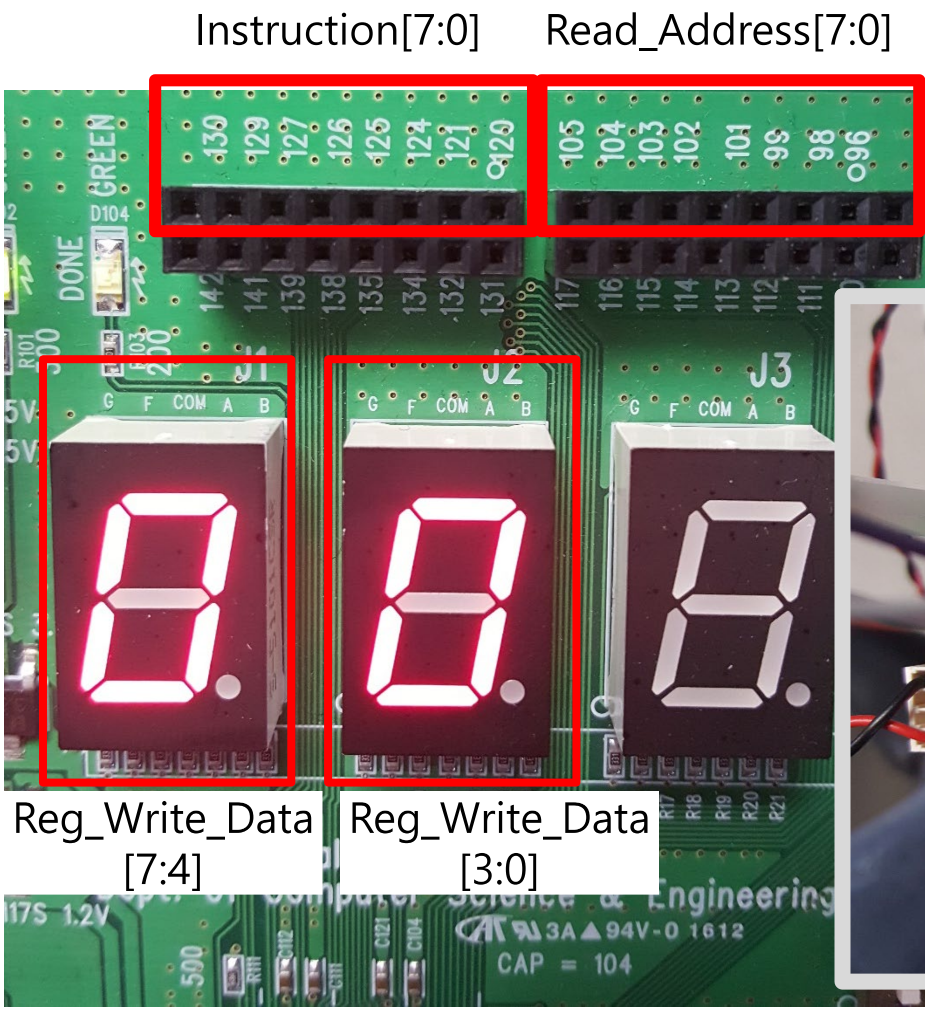
Register

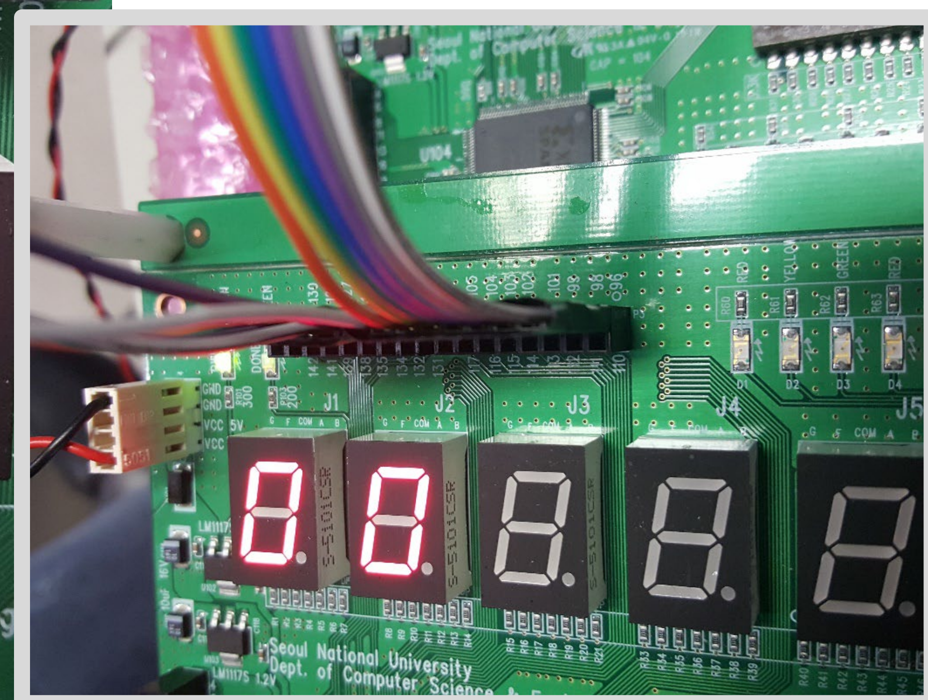
[0] 0
[1] 0
[2] 1
[3] 1

Test Environment

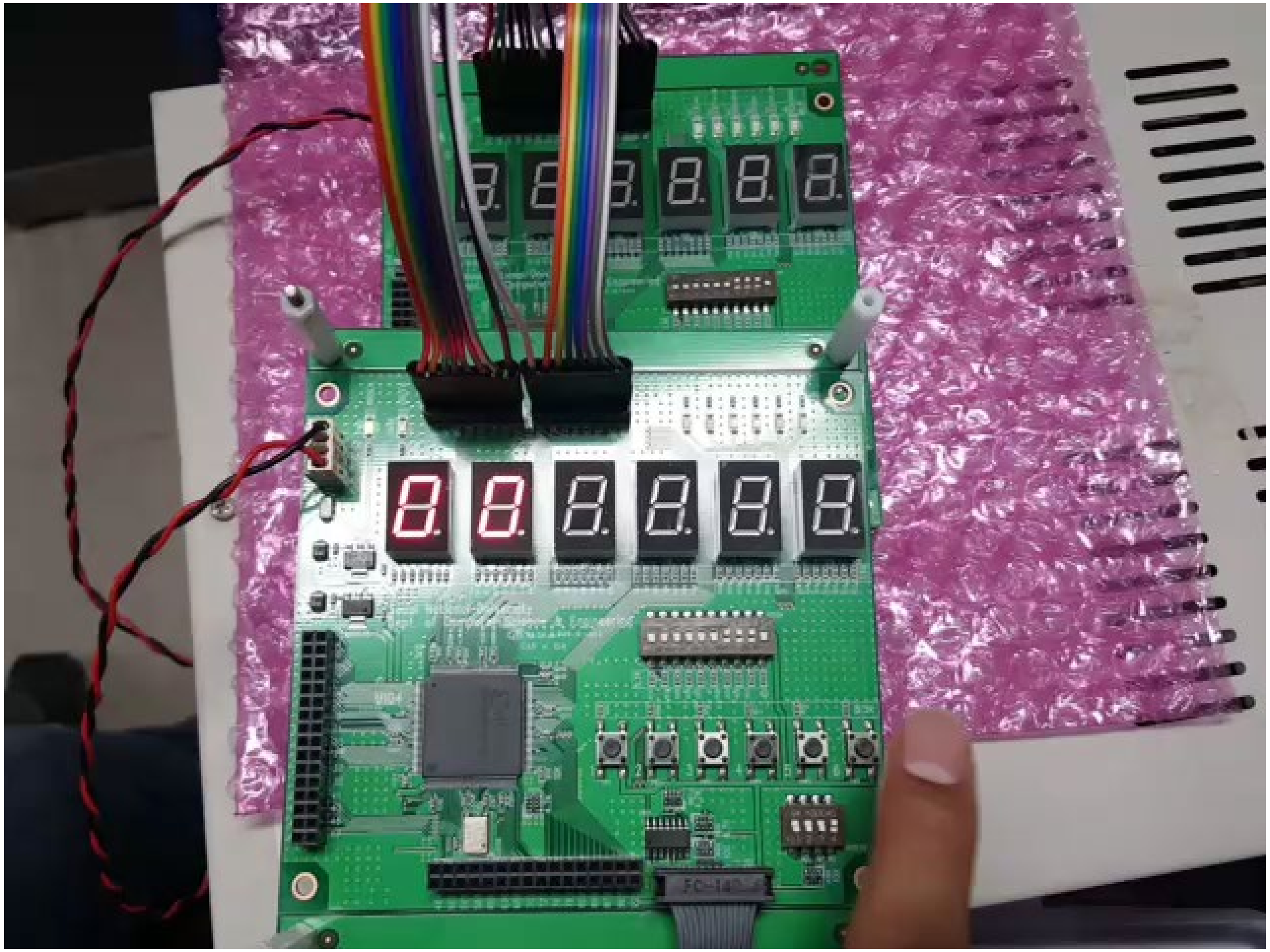
```
`timescale 1ns / 1ps

//module instruction memory
module IMEM ( instruction, Read_Address);

    output [7:0] instruction;

    input  [7:0] Read_Address;

    wire   [7:0] MemByte[31:0]; //32 words(bytes) of memory, just example..

    ////// Basic Operation Test Set ///

    // lw $s2, 1($s0)
    assign MemByte[0] = {2'b01, 2'b00, 2'b10, 2'b01};

    // j + 1
    assign MemByte[1] = {2'b11, 2'b00, 2'b00, 2'b01};

    // add $s0, $s1 $s2
    assign MemByte[2] = {2'b00, 2'b01, 2'b10, 2'b00};

    // sw $s2, 1($s2)
    assign MemByte[3] = {2'b10, 2'b10, 2'b10, 2'b01};

    // lw $s3, 1($s0)
    assign MemByte[4] = {2'b01, 2'b00, 2'b11, 2'b01};

    assign instruction = MemByte[Read_Address];


endmodule
```

Test Environment





Test Environment



Project Grading

- **Report (individual) (45%)**
  - Explain the overall design and structure in detail
  - Specify the functionality of each module in your implementation
  - Verify your implementation is correct with simulation result
- **Completeness (55%)**
  - Sets of Test Instructions (testing from basic operations to combinations of operations) will be tested
  - You don't have to follow the previous slides. Any implementation is acceptable if the result is correct.
- **FPGA Submission** : right after the test.

  If the FPGAs are broken or missing, you will get the 20% penalty of your final project.

Project Grading

- **Board Test Date & Time**
  - Class 001 : June 12th 19:00 ~
    - 19:00 ~ 20:00 : Team 1 ~ 10
    - 20:00 ~ 21:00 : Team 11~20
  - Class 002 : June 13th 19:00 ~
    - 19:00 ~ 20:00 : Team 1 ~ 10
    - 20:00 ~ 21:00 : Team 11~20
  - Class 003 : June 15th 19:00 ~
    - 19:00 ~ 20:00 : Team 1 ~ 10
    - 20:00 ~ 21:00 : Team 11~20

  (Same as the original lab session)

  - Only the designated teams can be in the lab.
  - Your Board should be ready in 5 minute.

Project Grading

- **Report & Code Submission**
  - **Due** : same as board test date, 19:00
    (Delay Policy is same as lab report)

  - **File name format**

    report : **LDLAB_YYMMDD_class#_team#_NAME_studentID.pdf**

    zip file (Verilog (.v, .ucf) files) : **LDLAB_YYMMDD_class#_team#_NAME_studentID.zip**

    YYMMDD : lab class date

  - **Mail format**

    Mail Name : **LDLAB_YYMMDD_class#_team#_NAME_studentID**

    Should attach report (pdf) and verilog files (zip)

<u>NO PLAGIARISM(COPY)</u> → All Lab score will be zero point !!!