

# Practice 10

## Final Project

M1522.000700 논리설계 002

2018학년도 1학기

based on lecture notes from RUBIS lab

# Contents

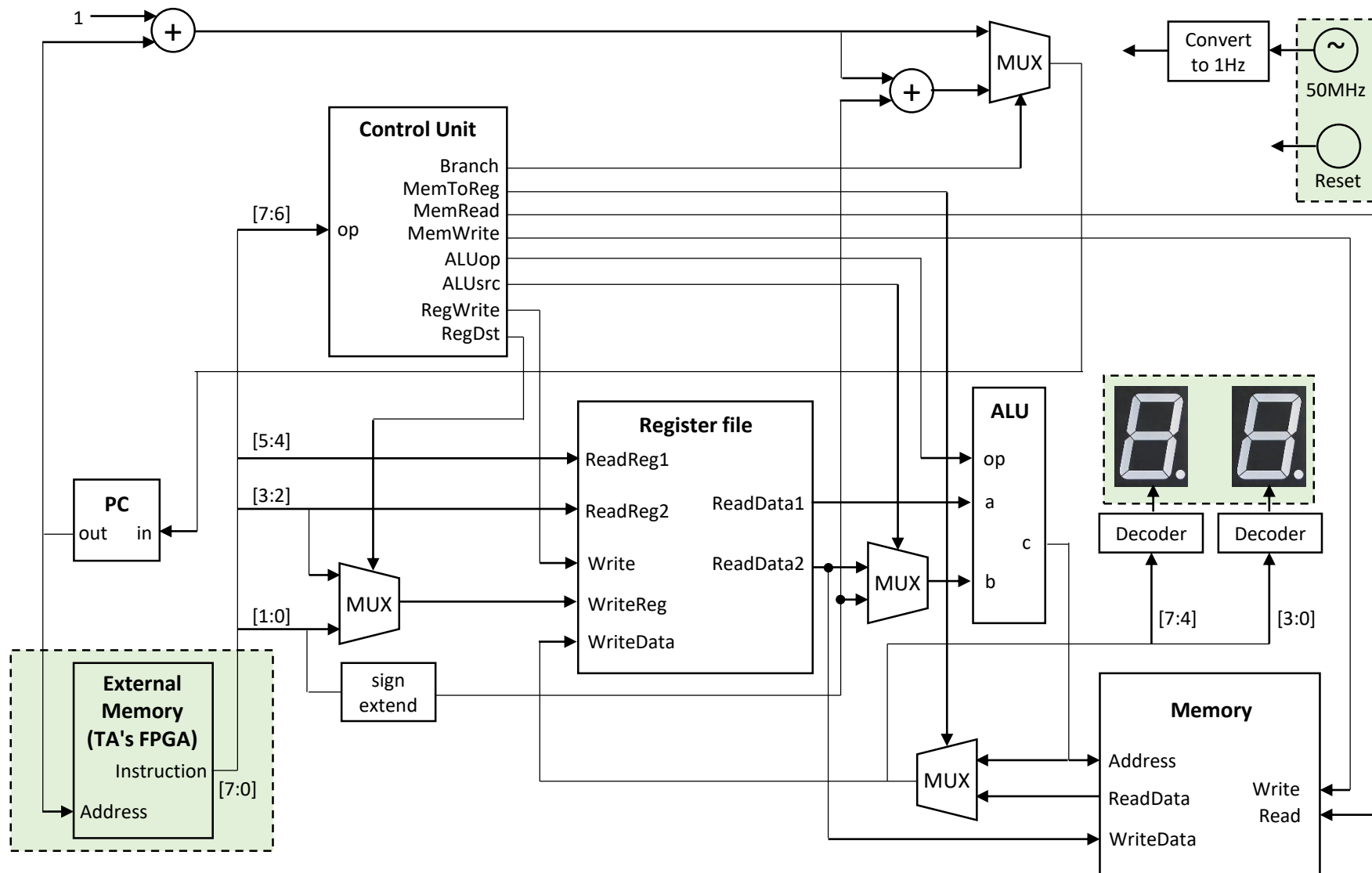
- Project overview
- Microprocessor design
  - Overview
  - Instruction Set Architecture (ISA)
  - Explanation on each component
- Example test set
- Test environment
- Grading

# Project Overview

- Implement a simple 8-bit microprocessor from [1] in Verilog and program it on FPGA.
- 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 WriteData of register file.
  - Use 7 segment displays. Display the value in Hexadecimal.

[1] David A. Patterson, John L. Hennessy. (2005). Computer Organization and Design (3rd ed.). Morgan Kaufmann.

# Microprocessor Design - Overview

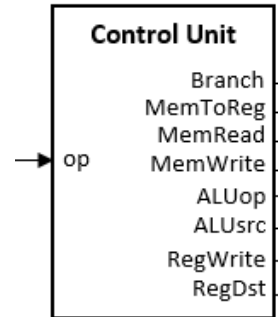


# Microprocessor Design - ISA

op	bit								Assembly	C	Description
	7	6	5	4	3	2	1	0			
add	0		r1		r2		rd		add \$rd, \$r1, \$r2	rd = r1 + r2	add r1 and r2, save result in rd
load	1		r1		r2		imm		load \$r2, imm(\$r1)	r2 = Mem[r1 + imm]	memory -> reg
store	2		r1		r2		imm		store \$r2, imm(\$r1)	Mem[r1 + imm] = r2	reg -> memory
jump	3							imm	jump imm	PC = (PC + 1) + (imm)	jump to (PC in next cycle) + (offset)

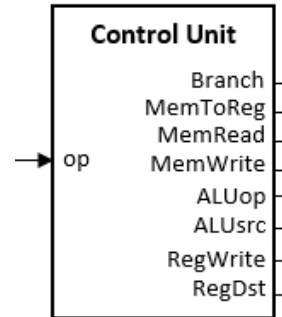
- \* for add operation, you don't have to care about the overflow.
- \* only the immediate is signed, others are unsigned.

# Microprocessor Design - Control Unit



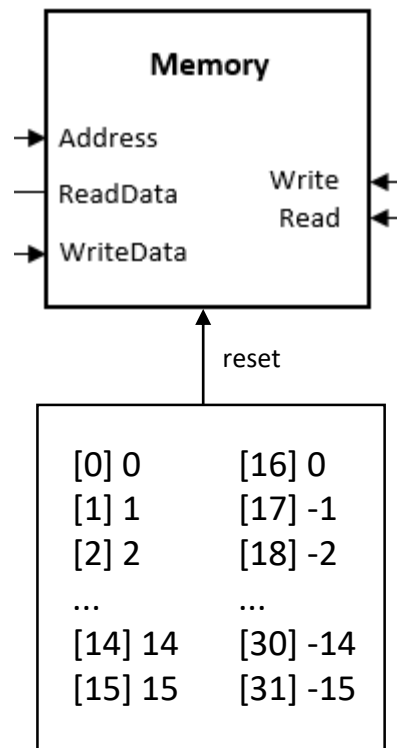
Signal	Effect when deasserted (0)	Effect when asserted (1)
Branch	The PC is replaced by the output of the adder that computes the value of PC + 1	The PC is replaced by the output of the adder that computes the branch target
MemToReg	The value fed to the register file's "WriteData" comes from the ALU	The value fed to the register file's "WriteData" comes from the memory
MemRead	None	Data memory contents designated by the address input are put on "ReadData" of the memory
MemWrite	None	Data memory contents designated by the address input are replaced by the value on the "WriteData"
ALUop	None	Choose ALU operation (In this project, only add operation exists. This signal is for later expandability.)
ALUsrc	The second ALU operand comes from the second register file output	The second ALU operand is the sign-extended, lower 2 bits of the instruction
RegWrite	None	Write result to register selected by "WriteReg"
RegDst	"WriteReg" comes from the r1 field	"WriteReg" comes from the r2 field

# Microprocessor Design - Control Unit



op	Branch	MemToReg	MemRead	MemWrite	ALUop	ALUsrc	RegWrite	RegDst
add	0	0	0	0	1	0	1	1
load	0	1	1	0	0	1	1	0
store	0	x	0	1	0	1	0	x
jump	1	x	0	0	0	0	0	x

# Microprocessor Design - Data Memory

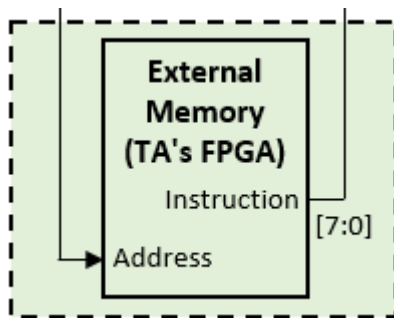


Data memory

- 8-bit word size
- 8-bit address (= 256 words)
  - due to FPGA capacity, we will use only 32 words
- Initialize the data memory when you press reset button.
  - hint : use always block sensitive to reset signal



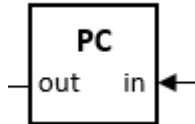
# Microprocessor Design - Instruction Memory



## Instruction memory

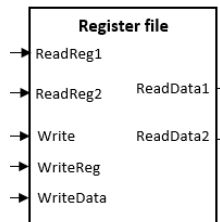
- 8-bit word size
- 8-bit address (= 256 words)
- It will be given by external memory on evaluation

# Microprocessor Design - Others



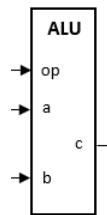
Program counter

- Indicates the address of instructions.
- Set to zero when reset.



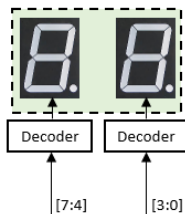
Register file

- Consists of four 8-bit general purpose registers.
- Write data to register or output the read data from register according to control inputs.



ALU

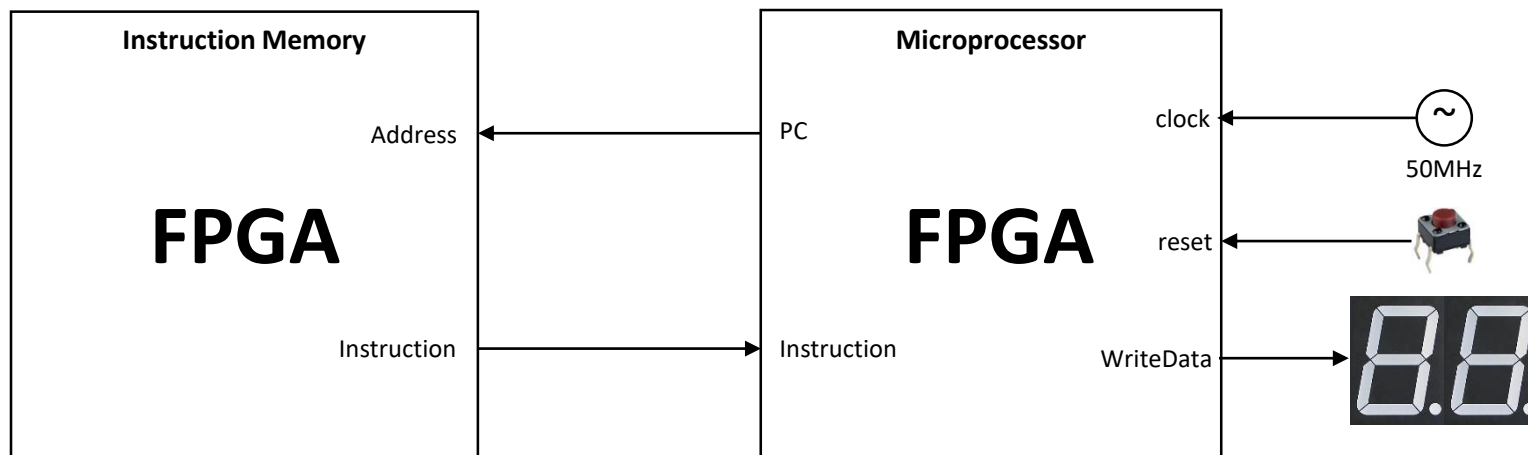
- Performs add operation (don't care overflow)



7-segment display

- Show "WriteData" in Hexadecimal.

# Microprocessor Design



# Example

## Instruction memory

```
[0] load $r1, 0($r0)    // 8'b01000100
[1] load $r2, 1($r0)    // 8'b01001001
[2] add $r0, $r1, $r2   // 8'b00011000
[3] store $r2, 1($r0)   // 8'b10001001
[4] jump -1             // 8'b11000011
```

Data memory	Register file	PC
[0] 0	[0] 0	0
[1] 1	[1] 1	
[2] 2	[2] 2	
[3] 3	[3] 3	
...		




0 0

# Example

## Instruction memory

```
[0] load $r1, 0($r0)    // 8'b01000100
[1] load $r2, 1($r0)    // 8'b01001001
[2] add $r0, $r1, $r2    // 8'b00011000
[3] store $r2, 1($r0)    // 8'b10001001
[4] jump -1              // 8'b11000011
```

Data memory		Register file	PC
[0] 0		[0] 0	1
[1] 1		[1] 0	
[2] 2		[2] 2	
[3] 3		[3] 3	
...			




0 1

# Example

## Instruction memory

```
[0] load $r1, 0($r0)    // 8'b01000100
[1] load $r2, 1($r0)    // 8'b01001001
[2] add $r0, $r1, $r2    // 8'b00011000
[3] store $r2, 1($r0)    // 8'b10001001
[4] jump -1              // 8'b11000011
```

Data memory	Register file	PC
[0] 0	[0] 0	2
[1] 1	[1] 0	
[2] 2	[2] 1	
[3] 3	[3] 3	
...		




0 1

# Example

## Instruction memory

```
[0] load $r1, 0($r0)    // 8'b01000100
[1] load $r2, 1($r0)    // 8'b01001001
[2] add $r0, $r1, $r2    // 8'b00011000
[3] store $r2, 1($r0)    // 8'b10001001
[4] jump -1              // 8'b11000011
```

Data memory	Register file	PC
[0] 0	[0] 1 <- 0 + 1	3
[1] 1	[1] 0	
[2] 2	[2] 1	
[3] 3	[3] 3	
...		



x x

# Example

## Instruction memory

```
[0] load $r1, 0($r0)    // 8'b01000100
[1] load $r2, 1($r0)    // 8'b01001001
[2] add $r0, $r1, $r2   // 8'b00011000
[3] store $r2, 1($r0)   // 8'b10001001
[4] jump -1             // 8'b11000011
```

Data memory	Register file	PC
[0] 0	[0] 1	4
[1] 1	[1] 0	
[2] 1 ←	[2] 1	
[3] 3	[3] 3	
...		



x x



# Example

## Instruction memory

```
[0] load $r1, 0($r0)    // 8'b01000100
[1] load $r2, 1($r0)    // 8'b01001001
[2] add $r0, $r1, $r2   // 8'b00011000
[3] store $r2, 1($r0)   // 8'b10001001
[4] jump -1              // 8'b11000011
```

## Data memory

```
[0] 0
[1] 1
[2] 1
[3] 3
...
```

## Register file

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

## PC

4



x x

**infinite loop**

# Testbench

```
`timescale 1ns / 1ps

module IMEM ( Instruction, Read_Address );

    output[7:0] Instruction;

    input [7:0] Read_Address;

    wire [7:0] MemByte[5:0]; // 6 words(bytes) of memory

    //// Basic Operation Test Set ////
    // lw $s1, 0($s0)
    assign MemByte[0] = { 2'b01, 2'b00, 2'b01, 2'b00 };

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

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

    // sw $s2, 1($s0)
    assign MemByte[3] = { 2'b10, 2'b00, 2'b10, 2'b01 };

    // j -1
    assign MemByte[4] = { 2'b11, 4'b0000, 2'b11 };

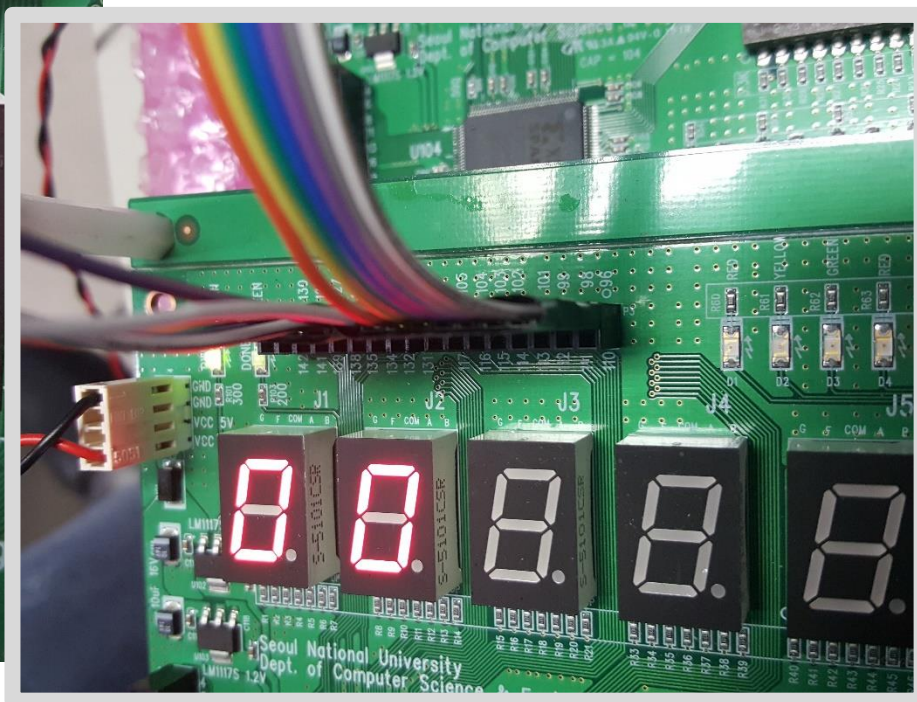
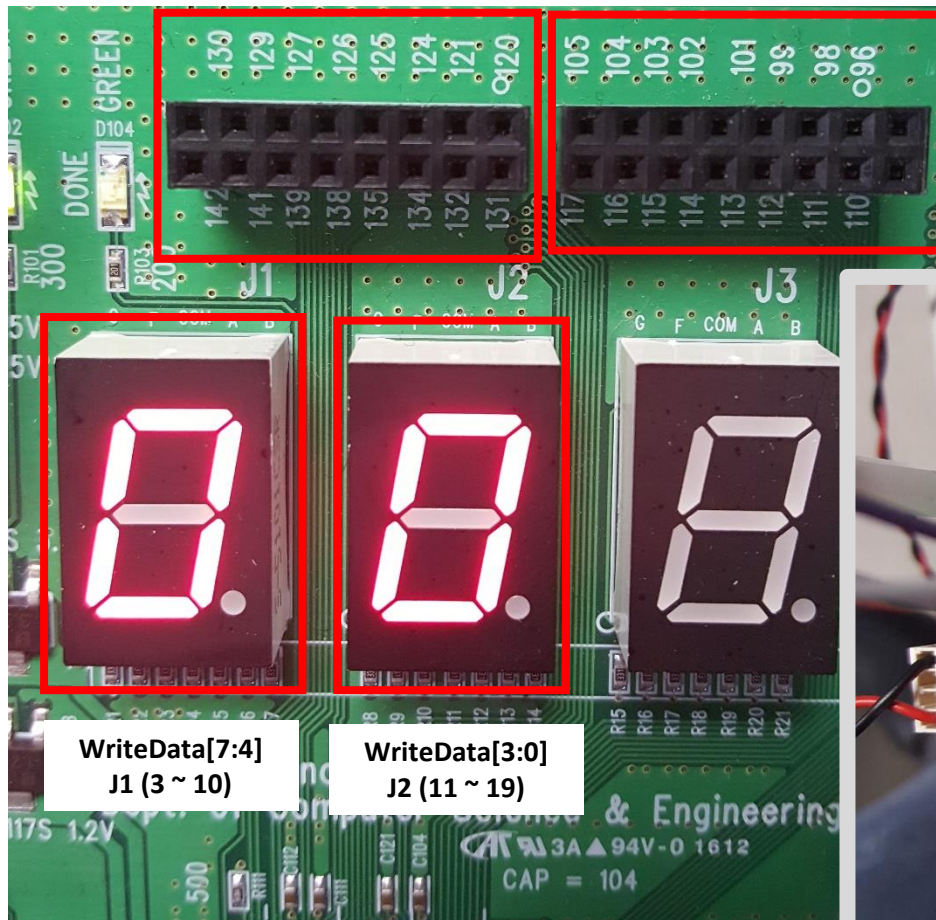
    assign Instruction = MemByte[Read_Address];

endmodule
```

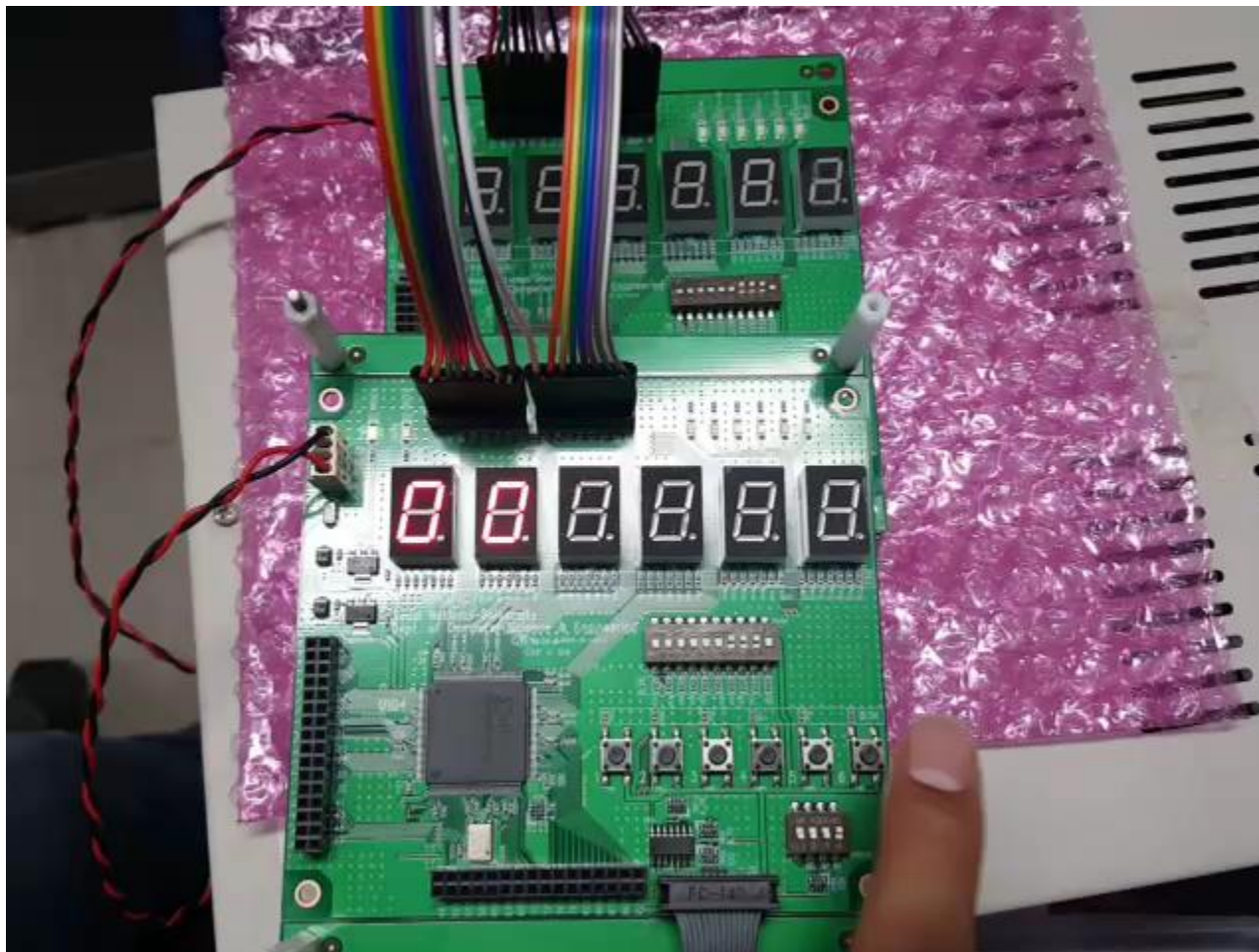
# Test Environment

Instruction[7:0]  
P4 (120 ~ 130)

Address[7:0]  
P3 (96 ~ 105)



# Test Environment



# Grading

- Report (individual) (30%)
  - 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 (60%)
  - Sets of test instructions will be tested
- Extra credits (10%)
  - Overflow detection (5%)
  - Any useful extra implementation (register status display, operation display, etc.) (5%)
- 부정행위 적발시 서울대학교 학칙에 따름



# Schedule

날짜	실습 수업	Report 제출
5월 16일	Practice 08 - Flip-Flop + Project 설명	5월 23일
5월 23일	Practice 09 - Counter	5월 30일
5월 30일	휴강	
6월 6일	휴강	
6월 13일	Project due	

# Schedule

- 6월 13일 수요일
  - 19:00 ~ 19:15 : Team 1 ~ 5
  - 19:15 ~ 19:30 : Team 6 ~ 10
  - 19:30 ~ 19:45 : Team 11 ~ 15
  - 19:45 ~ 20:00 : Team 16 ~ 20
  - 20:00 ~ 20:15 : Team 21 ~ 25
  - 20:15 ~ 20:30 : Team 26 ~ 30
  - 해당 팀만 교실에 입장 가능

# Report

- 보고서
  - ldjta@aces.snu.ac.kr로 제출
  - 개인 제출 (사진은 조원이랑 공유해도 됨)
  - **report.pdf 1개와 소스코드를 함께 압축하여 압축파일 1개 제출**
  - 자유 분량
  - 시간 제한 있는 첨부 금지 (네X버 대용량 첨부...)
  - Due : **June, 13th** (실습 수업 시작 시각인 **7:00pm** 전까지)
  - 메일 제목, 파일명 형식에 맞춰 보내야 채점
  - 메일 제목 : [논리설계실습]\_Project\_team#\_이름
  - 파일명 : [논리설계실습]\_Project\_team#\_이름.zip 또는 .tar.gz