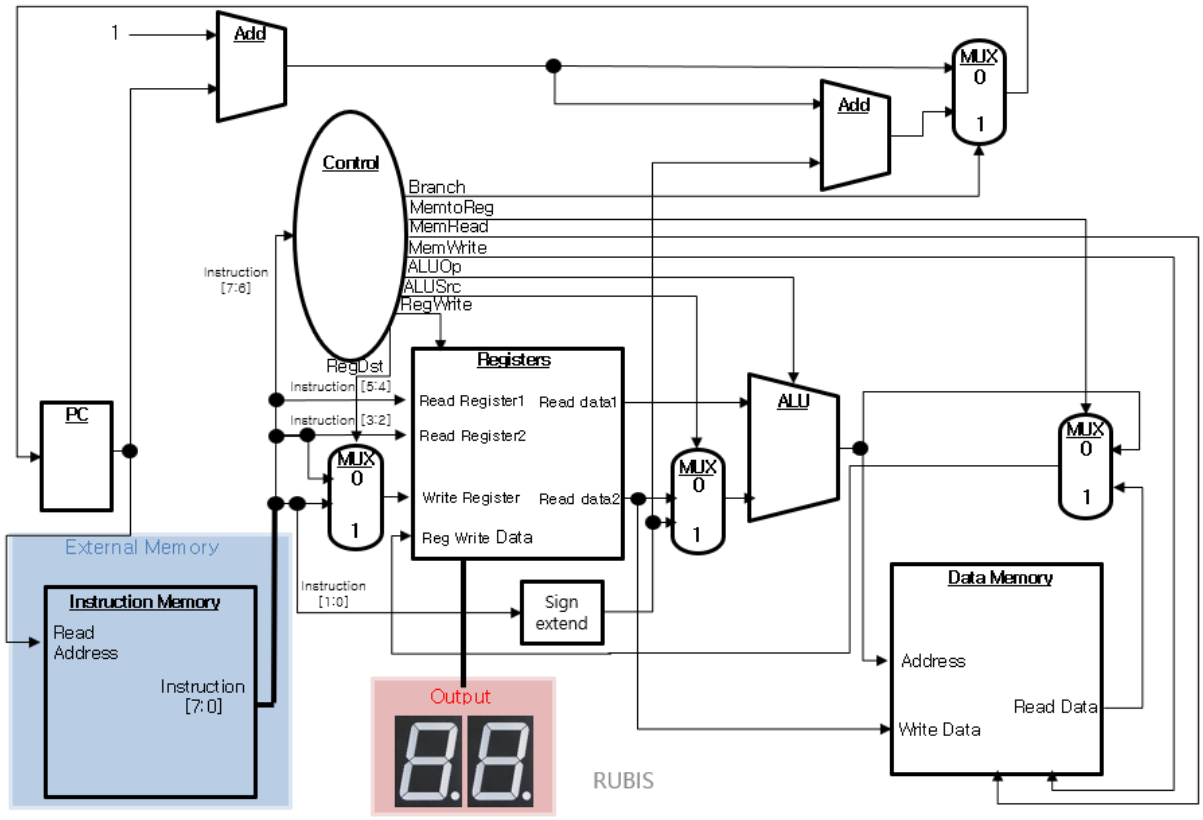
Final Project

-Simple Microprocessor-

Logic Design Lab.

Dong-hak Lee

Data Path:



ISA:

(8bit processor)

* op	operation code	* Registers (8bits)	
rd	destination register	Program Counter(PC)	
rs	source register	$s0 - $s3	
rt	source register two		
imm	immediate (constant)	* Memories (8bits)	
		Instruction Mem.	address range: 0x00-0xFF
op, rd, rs, rt	unsigned value	Data Mem.	address range: 0x00-0xFF
imm	signed value		

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

※ for Add operation, you don't have to care about the overflow.
※ only the immediate is signed value, other values are unsigned

Signal Name	Effect when deasserted (0)	Effect when asserted (1)
RegDst	The register file destination number for the Write register comes from the rt field (bits4-5)	The register file destination number for the Write register comes from the rd field (bits2-3)
RegWrite	None	Write result to rd
ALUSrc	The second ALU operand comes from the second register 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 computes the value of PC + 1	The PC is repaced by the output of the adder that computes the branch target
MemRead	None	Data memory contents designated by the address input are put on the Read data output
MemWrite	None	Data memory contents designated by the address input are replaced by the value on the Write data input
MemtoReg	The value fed to the register Write data input comes from the ALU	The value fed to the register Write data input comes from the data memory
ALUOp	None	Choose ALU operation (in this project, only Add operation exists. So this signal 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

Microprocessor.v (main code): 50MHz clock과 reset, instruction을 입력 받아서 실행 결과를 표시하기 위한 7 segment 2개와 다음 address, 그리고 추가기능으로 구현한 LED 6개를 출력한다.

```
clk CLOCK(.clkin(_clk), .clkout(clock));
pc PC(.in(_pc), .clkin(clock), .reset(reset), .out(address));
control CON(.in(instruction[7:6]), .out(cu));
memory
MEM(.address(alu_c), .wd(_rd2), .clock(clock), .read(cu[5]), .write(cu[4]), .reset(reset), .rd(mem_rd));
register
REG(.rr1(instruction[5:4]), .rr2(instruction[3:2]), .clock(clock), .write(cu[1]), .reset(reset),
.wr(_wr), .wd(_wd), .rd1(_rd1), .rd2(_rd2));
alu ALU(.in1(_rd1), .in2(alu_b), .out(alu_c));
mux8 PCMUX(.in1(nextaddress), .in2(jumpedaddress), .enable(cu[7]), .out(_pc));
mux8 ALUMUX(.in1(_rd2), .in2(extinst), .enable(cu[2]), .out(alu_b));
mux8 MEMMUX(.in1(alu_c), .in2(mem_rd), .enable(cu[6]), .out(_wd));
mux WMUX(.in1(instruction[3:2]), .in2(instruction[1:0]), .enable(cu[0]), .out(_wr));
add ADD1(.in1(one), .in2(address), .out(nextaddress));
add ADDJ(.in1(nextaddress), .in2(extinst), .out(jumpedaddress));
BCDto7 BNUM(.bcd(_wd[7:4]), .seg(bnum));
BCDto7 SNUM(.bcd(_wd[3:0]), .seg(snum));
signextend EXT(.in(instruction[1:0]), .out(extinst));
Bcount ADDITIONAL(.clock(clock), .reset(reset), .LED(LED));
```

Clk.v: 50MHz oscillator를 1Hz clock으로 바꿔준다.

Pc.v: reset 시그널이 들어오면 0으로 초기화하고 아니면 다음 pc값을 반환한다.

Control.v: operation code를 입력 받아 control signal을 출력한다.

Memory.v: 8bit address를 가지는 데이터를 32개 저장함. Reset, read, write 가능.

Register.v: 8bit 레지스터 4개로 구성. Read, write 가능.

Alu.v: 수학적 연산들을 수행하나 Final Project에서는 add operation만 필요로 함.

Mux8.v: 8bit 입력 값 두개 중에서 하나를 골라 출력함.

- PCMUX: 다음 pc값을 (pc+1)로 할지 점프한 pc로 할지 결정
- ALUMUX: input값을 register에 저장된 값으로 할지 instruction에 있는 상수로 할지 결정
- MEMMUX: Write Data를 memory에서 불러온 값으로 할지 ALU의 output으로 할지 결정

Mux.v: 2bit 입력 값 두개 중에서 하나를 골라 출력함.

Add.v: 두개의 입력 값을 더해서 출력함.

- ADD1: PC에 1을 더함
- ADDJ: PC에 1을 더한 값에 jump를 함

BCDto7.v: 10진수로 0~15 사이의 4bit 숫자가 입력되면 7segment로 16진수로 표현되는 7bit data를 출력한다.

Signextend.v: 2bit data를 입력 받아 2's complement를 만족하는 8bit로 변환하여 출력한다.

Bcount.v: 추가기능, reset한 이후로 흐른 시간을 6개의 LED를 통해 2진수로 표현한다.

위에 첨부한 Data path와 ISA를 그대로 구현함.

코드는 별첨하였으므로 보고서에 따로 첨부하지는 않음.

Test code:

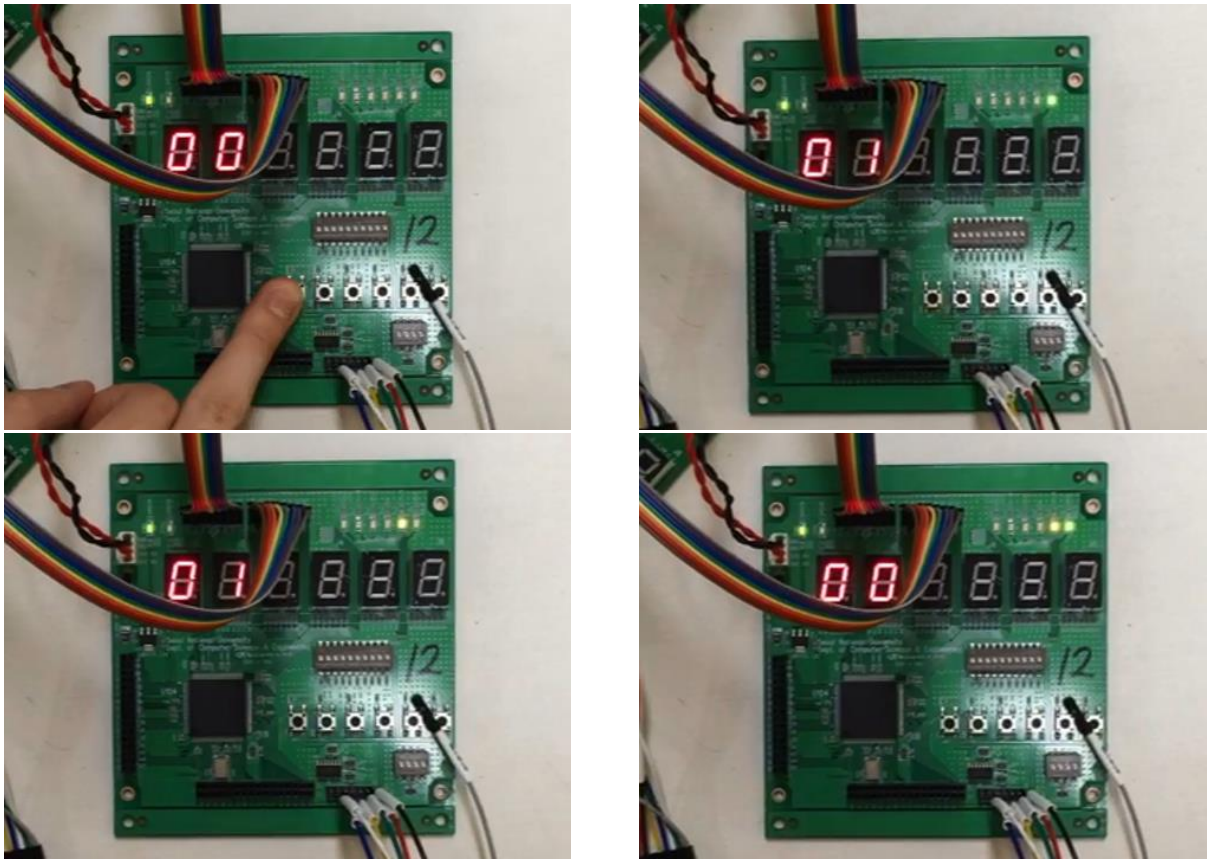
```
module IMEM(
	output [7:0] Instruction,
	input [7:0] Read_Address
	);
		wire [7:0] MemByte[20:0];

		assign MemByte[0] = 8'b01000100;
		assign MemByte[1] = 8'b01001001;
		assign MemByte[2] = 8'b00011001;
		assign MemByte[3] = 8'b10000100;
		assign MemByte[4] = 8'b01000100;
		assign MemByte[5] = 8'b01001001;
		assign MemByte[6] = 8'b00011001;
		assign MemByte[7] = 8'b10000100;
		assign MemByte[8] = 8'b01000100;
		assign MemByte[9] = 8'b01001001;
		assign MemByte[10] = 8'b00011001;
		assign MemByte[11] = 8'b10000100;
		assign MemByte[12] = 8'b01000100;
		assign MemByte[13] = 8'b01001001;
		assign MemByte[14] = 8'b00011001;
		assign MemByte[15] = 8'b10000100;
		assign MemByte[16] = 8'b01000100;
		assign MemByte[17] = 8'b01001001;
		assign MemByte[18] = 8'b00011001;
```

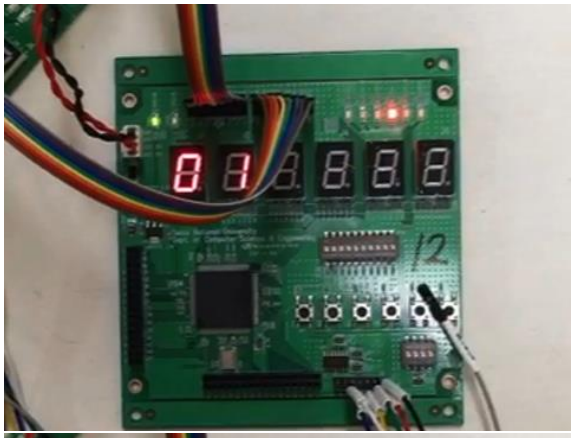
```
        assign MemByte[19] = 8'b10000100;

        assign MemByte[20] = 8'b11000011;


        assign Instruction = MemByte[Read_Address];
endmodule
```

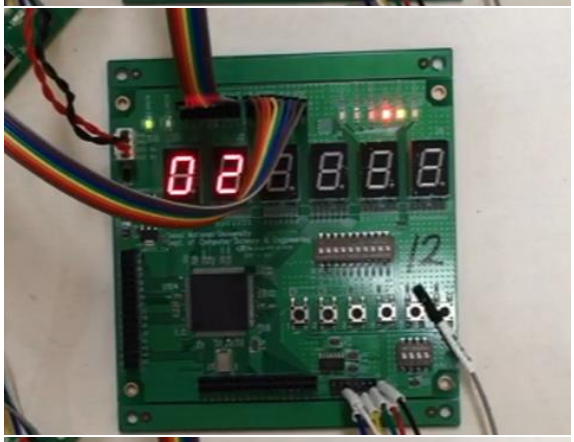
예상 실행 결과:

00 – 01 – 01 – 00 – 01 – 01 – 02 – 00 – 02 – 01 – 03 – 00 – 03 – 01 – 04 – 00 – 04 – 01 – 05 – 00

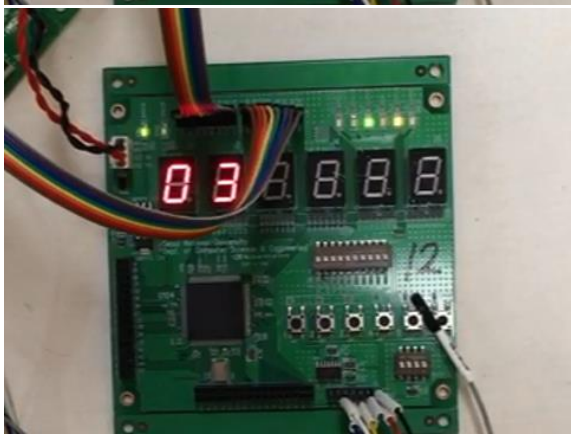
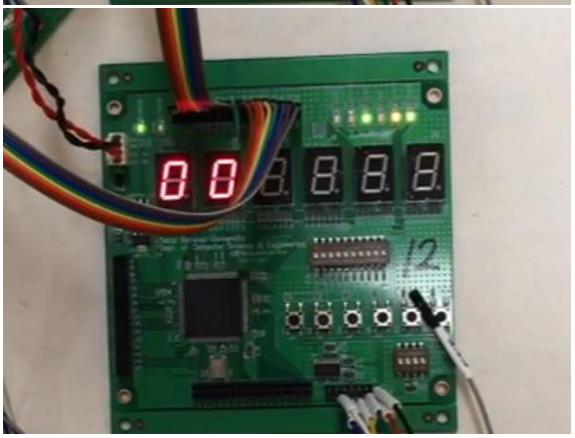
실제 실행 결과:



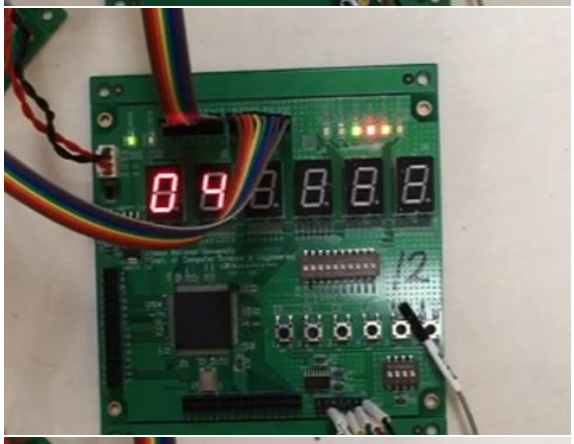




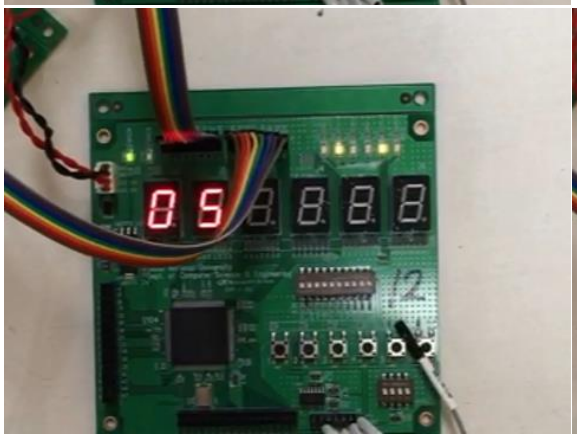
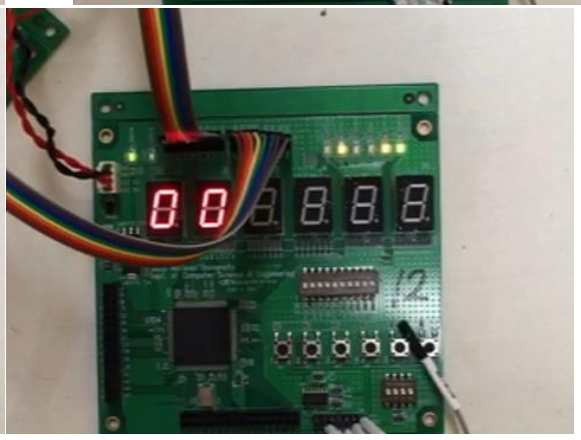








➔ 예상 실행 결과와 완벽하게 일치함