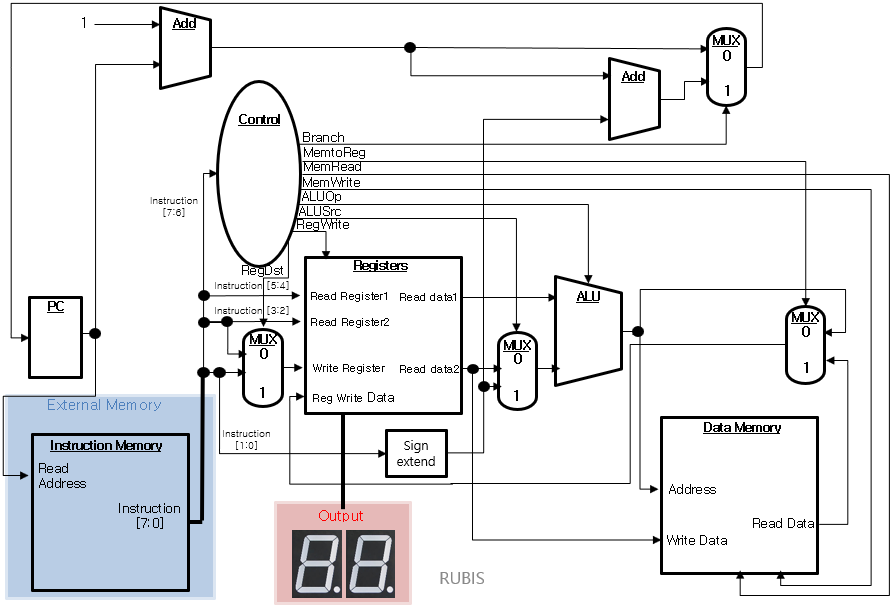
**Final Project**

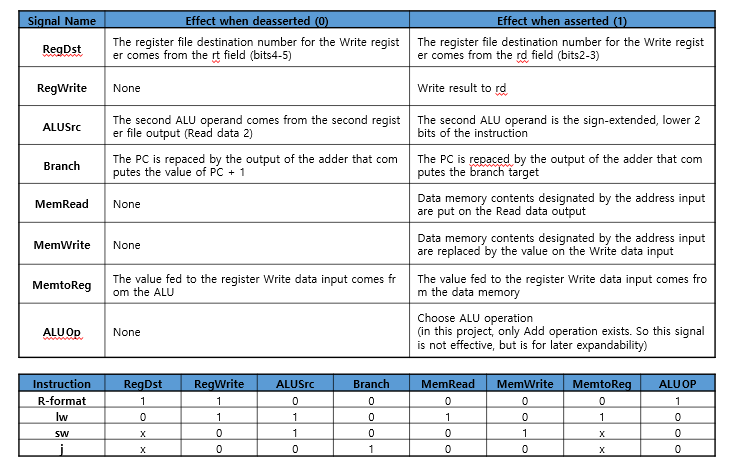
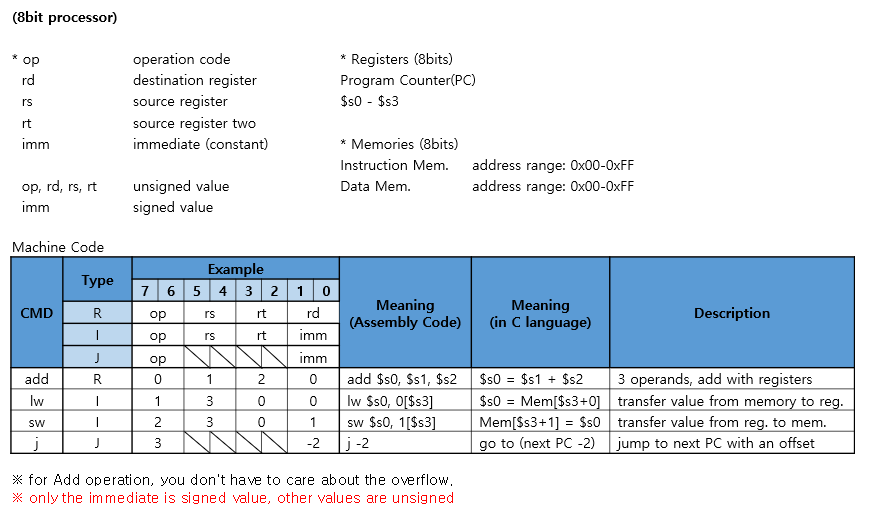
-Simple Microprocessor-

Logic Design Lab.

Dong-hak Lee

Data Path:



ISA: 

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;

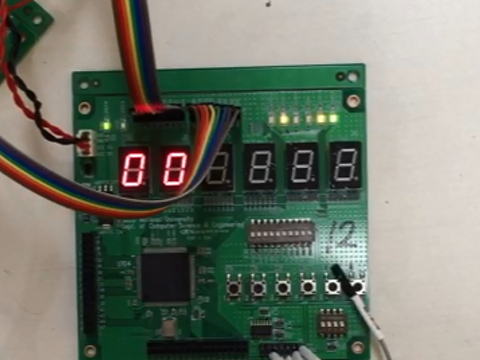
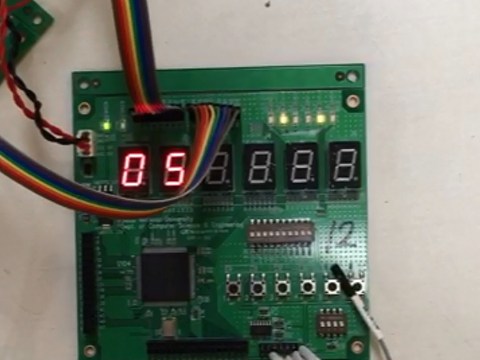
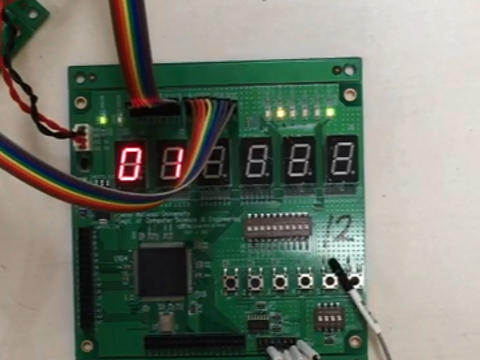
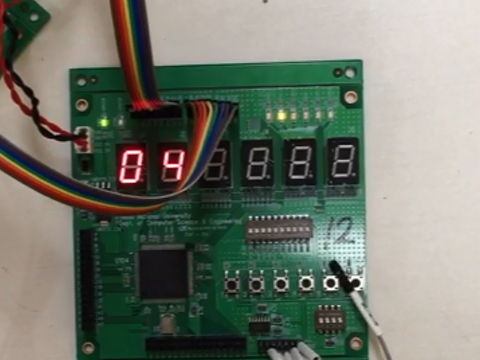
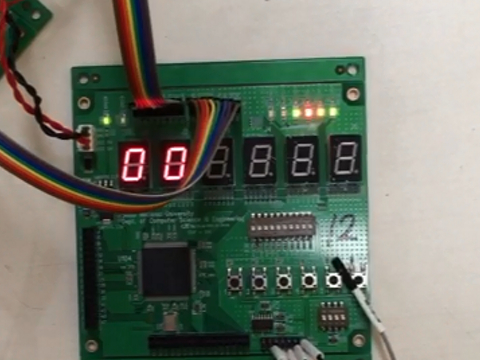
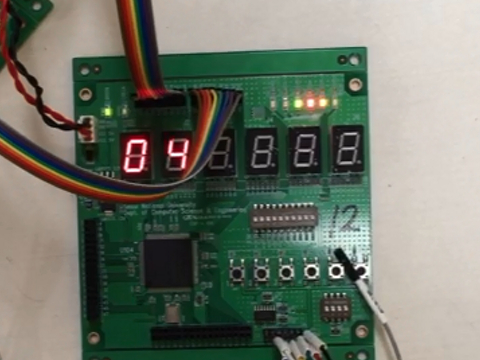
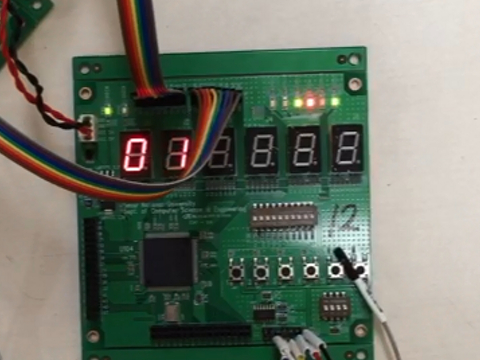
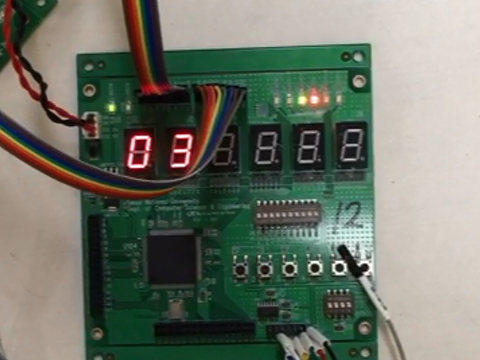
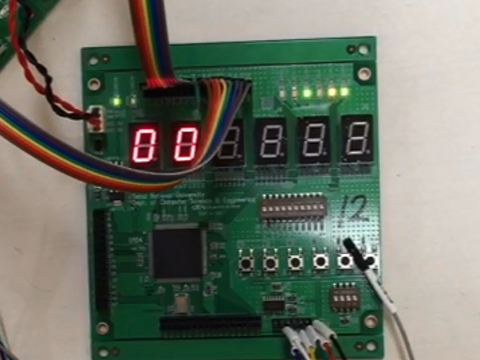
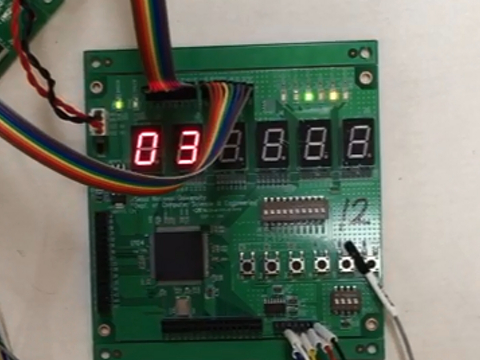
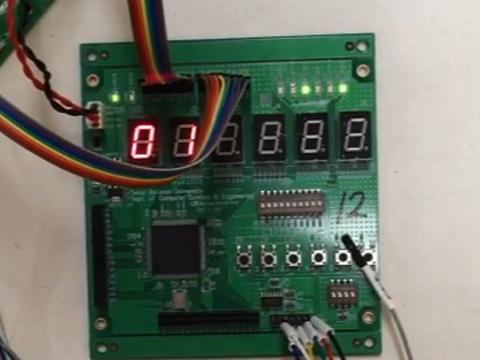
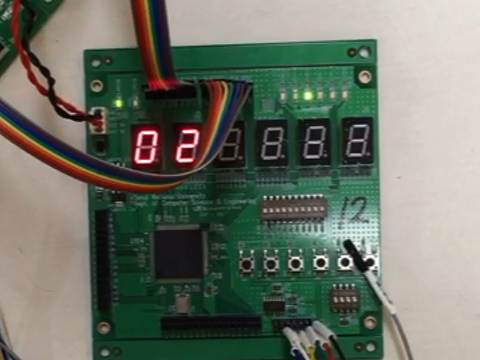
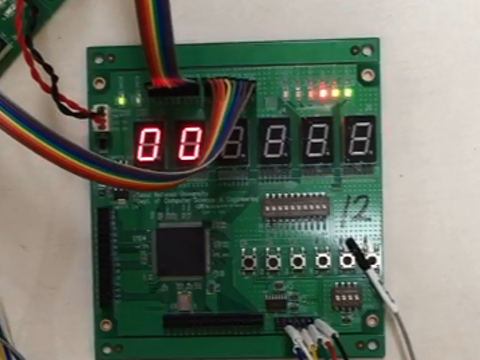
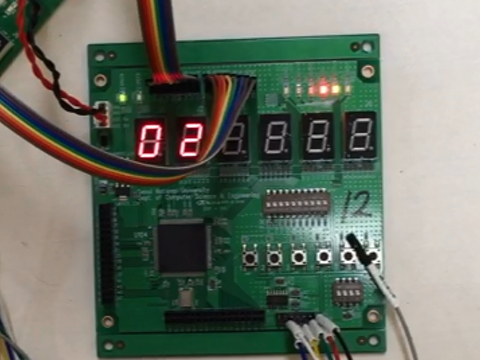
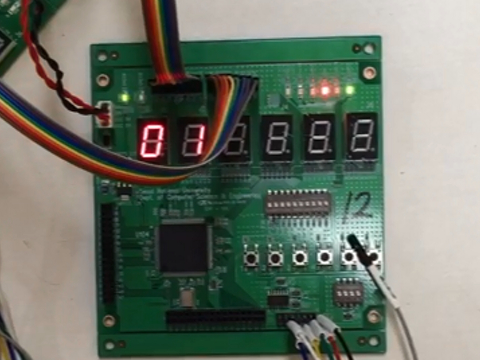
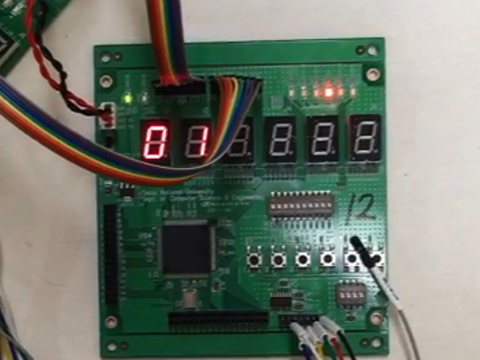
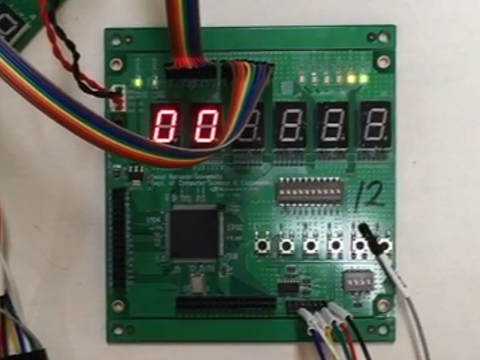
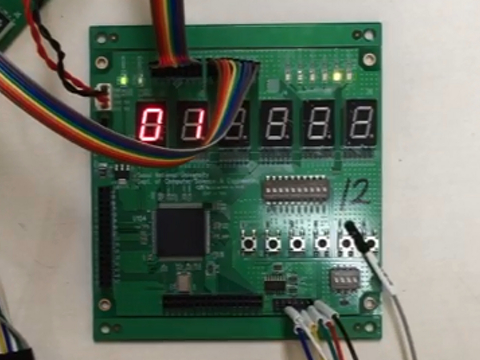
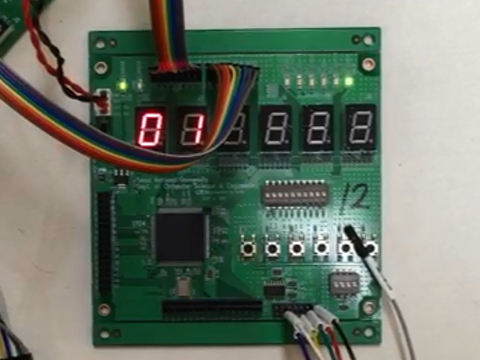
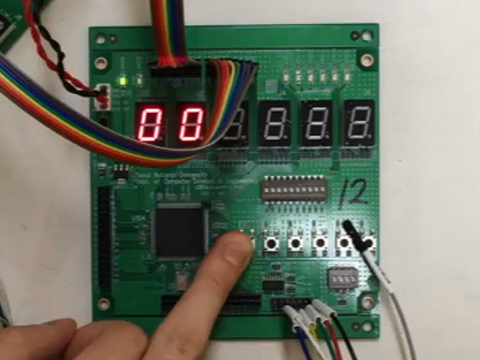
assign Instruction = MemByte[Read\_Address];

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