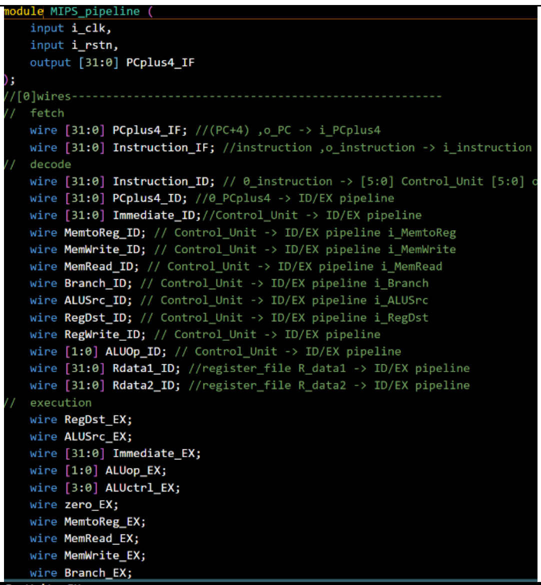
# 제목: 5-stage pipelined MIPS 설계

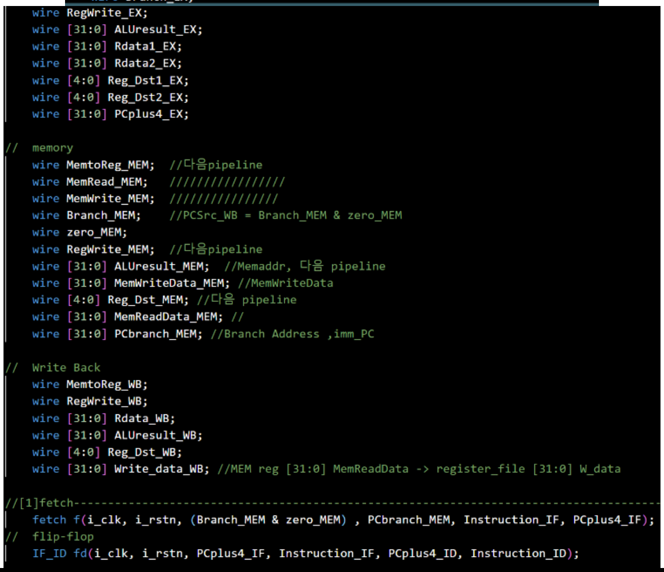
2019043890 융합전자공학부

이창민

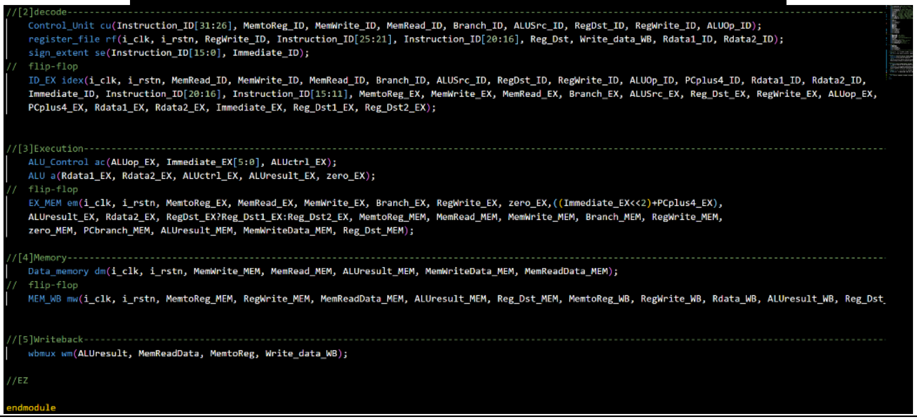
1. Mips pipeline 설계, 모듈과 플립플랍들을 연결했습니다.



```
module MIPS_pipeline (
    input i_clk,
    input i_rstn,
    output [31:0] PCplus4_IF
);
//[0]wires---------------------------------------------------
//  fetch
    wire [31:0] PCplus4_IF; //(PC+4) ,o_PC -> i_PCplus4
    wire [31:0] Instruction_IF; //instruction ,o_instruction -> i_instruction
//  decode
    wire [31:0] Instruction_ID; // 0_instruction -> [5:0] Control_Unit [5:0] c
    wire [31:0] PCplus4_ID; //0_PCplus4 -> ID/EX pipeline
    wire [31:0] Immediate_ID;//Control_Unit -> ID/EX pipeline
    wire MemtoReg_ID; // Control_Unit -> ID/EX pipeline i_MemtoReg
    wire MemWrite_ID; // Control_Unit -> ID/EX pipeline i_MemWrite
    wire MemRead_ID; // Control_Unit -> ID/EX pipeline i_MemRead
    wire Branch_ID; // Control_Unit -> ID/EX pipeline i_Branch
    wire ALUSrc_ID; // Control_Unit -> ID/EX pipeline i_ALUSrc
    wire RegDst_ID; // Control_Unit -> ID/EX pipeline i_RegDst
    wire RegWrite_ID; // Control_Unit -> ID/EX pipeline
    wire [1:0] ALUOp_ID; // Control_Unit -> ID/EX pipeline
    wire [31:0] Rdata1_ID; //register_file R_data1 -> ID/EX pipeline
    wire [31:0] Rdata2_ID; //register_file R_data2 -> ID/EX pipeline
//  execution
    wire RegDst_EX;
    wire ALUSrc_EX;
    wire [31:0] Immediate_EX;
    wire [1:0] ALUop_EX;
    wire [3:0] ALUctrl_EX;
    wire zero_EX;
    wire MemtoReg_EX;
    wire MemRead_EX;
    wire MemWrite_EX;
    wire Branch_EX;
```

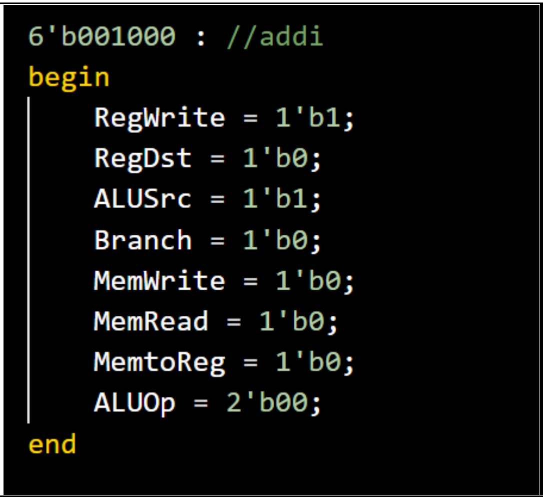


```
    wire RegWrite_EX;
    wire [31:0] ALUresult_EX;
    wire [31:0] Rdata1_EX;
    wire [31:0] Rdata2_EX;
    wire [4:0] Reg_Dst1_EX;
    wire [4:0] Reg_Dst2_EX;
    wire [31:0] PCplus4_EX;

//  memory
    wire MemtoReg_MEM;  //다음pipeline
    wire MemRead_MEM;   /////////////////
    wire MemWrite_MEM;  ////////////////
    wire Branch_MEM;    //PCSrc_WB = Branch_MEM & zero_MEM
    wire zero_MEM;
    wire RegWrite_MEM;  //다음pipeline
    wire [31:0] ALUresult_MEM;  //Memaddr, 다음 pipeline
    wire [31:0] MemWriteData_MEM; //MemWriteData
    wire [4:0] Reg_Dst_MEM; //다음 pipeline
    wire [31:0] MemReadData_MEM; //
    wire [31:0] PCbranch_MEM; //Branch Address ,imm_PC

//  Write Back
    wire MemtoReg_WB;
    wire RegWrite_WB;
    wire [31:0] Rdata_WB;
    wire [31:0] ALUresult_WB;
    wire [4:0] Reg_Dst_WB;
    wire [31:0] Write_data_WB; //MEM reg [31:0] MemReadData -> register_file [31:0] W_data

//[1]fetch-----------------------------------------------------------------------
    fetch f(i_clk, i_rstn, (Branch_MEM & zero_MEM) , PCbranch_MEM, Instruction_IF, PCplus4_IF);
//  flip-flop
    IF_ID fd(i_clk, i_rstn, PCplus4_IF, Instruction_IF, PCplus4_ID, Instruction_ID);
```

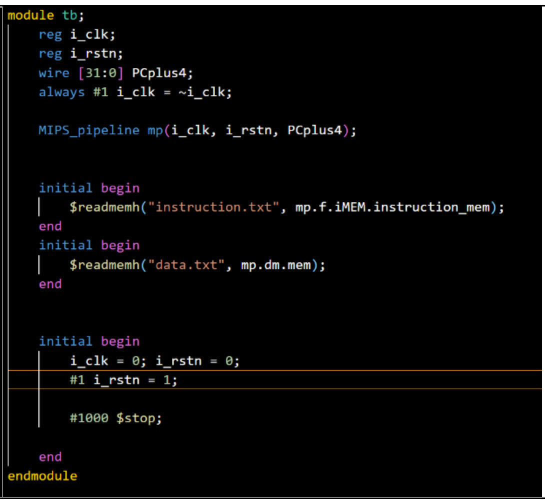


```
//[2]decode------------------------------------------------------------------------------------------------
    Control_Unit cu(Instruction_ID[31:26], MemtoReg_ID, MemWrite_ID, MemRead_ID, Branch_ID, ALUSrc_ID, RegDst_ID, RegWrite_ID, ALUOp_ID);
    register_file rf(i_clk, i_rstn, RegWrite_ID, Instruction_ID[25:21], Instruction_ID[20:16], Reg_Dst, Write_data_WB, Rdata1_ID, Rdata2_ID);
    sign_extent se(Instruction_ID[15:0], Immediate_ID);
//  flip-flop
    ID_EX idex(i_clk, i_rstn, MemRead_ID, MemWrite_ID, MemRead_ID, Branch_ID, ALUSrc_ID, RegDst_ID, RegWrite_ID, ALUOp_ID, PCplus4_ID, Rdata1_ID, Rdata2_ID,
    Immediate_ID, Instruction_ID[20:16], Instruction_ID[15:11], MemtoReg_EX, MemWrite_EX, MemRead_EX, Branch_EX, ALUSrc_EX, Reg_Dst_EX, RegWrite_EX, ALUop_EX,
    PCplus4_EX, Rdata1_EX, Rdata2_EX, Immediate_EX, Reg_Dst1_EX, Reg_Dst2_EX);


//[3]Execution---------------------------------------------------------------------------------------------
    ALU_Control ac(ALUop_EX, Immediate_EX[5:0], ALUctrl_EX);
    ALU a(Rdata1_EX, Rdata2_EX, ALUctrl_EX, ALUresult_EX, zero_EX);
//  flip-flop
    EX_MEM em(i_clk, i_rstn, MemtoReg_EX, MemRead_EX, MemWrite_EX, Branch_EX, RegWrite_EX, zero_EX,((Immediate_EX<<2)+PCplus4_EX),
    ALUresult_EX, Rdata2_EX, RegDst_EX?Reg_Dst1_EX:Reg_Dst2_EX, MemtoReg_MEM, MemRead_MEM, MemWrite_MEM, Branch_MEM, RegWrite_MEM,
    zero_MEM, PCbranch_MEM, ALUresult_MEM, MemWriteData_MEM, Reg_Dst_MEM);

//[4]Memory------------------------------------------------------------------------------------------------
    Data_memory dm(i_clk, i_rstn, MemWrite_MEM, MemRead_MEM, ALUresult_MEM, MemWriteData_MEM, MemReadData_MEM);
//  flip-flop
    MEM_WB mw(i_clk, i_rstn, MemtoReg_MEM, RegWrite_MEM, MemReadData_MEM, ALUresult_MEM, Reg_Dst_MEM, MemtoReg_WB, RegWrite_WB, Rdata_WB, ALUresult_WB, Reg_Dst_


//[5]Writeback---------------------------------------------------------------------------------------------
    wbmux wm(ALUresult, MemReadData, MemtoReg, Write_data_WB);

//EZ

endmodule
```

2.control unit에 addi 추가/ testbench 디버깅 용도로 메인 모듈에 output인 pcplus4를 선언하였고 tb에서 wire로 연결하였습니다.



```
6'b001000 : //addi
begin
    RegWrite = 1'b1;
    RegDst = 1'b0;
    ALUSrc = 1'b1;
    Branch = 1'b0;
    MemWrite = 1'b0;
    MemRead = 1'b0;
    MemtoReg = 1'b0;
    ALUOp = 2'b00;
end
```

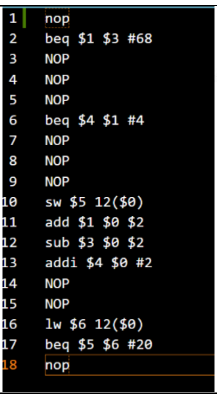


```
module tb;
    reg i_clk;
    reg i_rstn;
    wire [31:0] PCplus4;
    always #1 i_clk = ~i_clk;

    MIPS_pipeline mp(i_clk, i_rstn, PCplus4);


    initial begin
        $readmemh("instruction.txt", mp.f.iMEM.instruction_mem);
    end
    initial begin
        $readmemh("data.txt", mp.dm.mem);
    end


    initial begin
        i_clk = 0; i_rstn = 0;
        #1 i_rstn = 1;

        #1000 $stop;

    end
endmodule
```

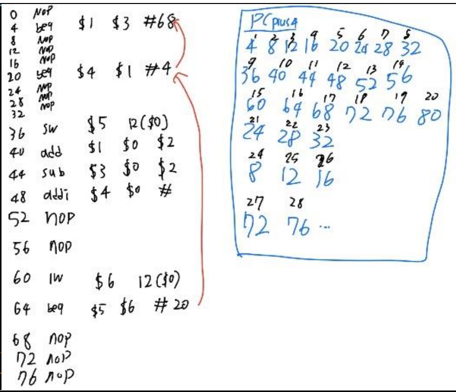
3. instruction 만들기

 우선 assembly code로 간단한 코딩을 하였습니다. PCplus4의 예상 값들을 구했습니다.

(beq $a $b #c 는 조건만족시 c인 절대 주소로 점프한다고 생각하고 짰습니다.)



```
nop
beq $1 $3 #68
NOP
NOP
NOP
beq $4 $1 #4
NOP
NOP
NOP
sw $5 12($0)
add $1 $0 $2
sub $3 $0 $2
addi $4 $0 #2
NOP
NOP
lw $6 12($0)
beq $5 $6 #20
nop
```



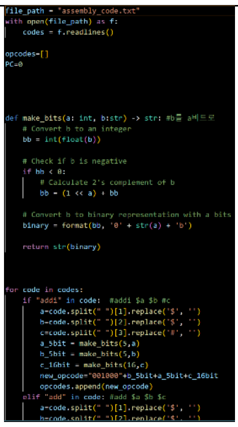
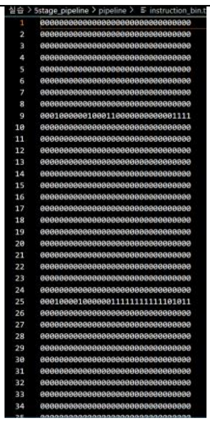
```
0   NOP
4   beq   $1  $3  #68
8   NOP
12  NOP
16  NOP
20  beq   $4  $1  #4
24  NOP
28  NOP
32  nop
36  sw    $5  12($0)
40  add   $1  $0  $2
44  sub   $3  $0  $2
48  addi  $4  $0  #
52  nop
56  nop
60  lw    $6  12($0)
64  beq   $5  $6  #20
68  nop
72  nop
76  nop
```

```
PCplus4
 1  2  3  4  5  6  7  8
 4  8  12 16 20 24 28 32
 9  10 11 12 13 14
 36 40 44 48 52 56
 15 16 17 18 19 20
 60 64 68 72 76 80
 21 22 23
 24 28 32
 24 25 26
 8  12 16
 27 28
 72 76 ...
```

이를 op code로 바꿔주는 assembler를 Python을 이용하여 만들었습니다.

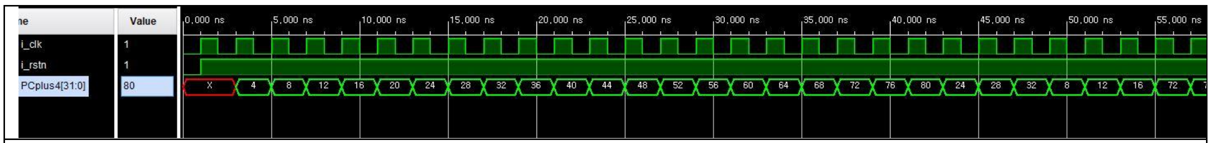
자세한 Python 코드는 깃허브에서 확인하실 수 있습니다.

https://github.com/chminsta/MIPS_assembler_with_Python

| | |
|---|---|
|  |  |
| 파이썬 일부 | 그 아웃풋 |

```python
file_path = "assembly_code.txt"
with open(file_path) as f:
    codes = f.readlines()

opcodes=[]
PC=0




def make_bits(a: int, b:str) -> str: #b를 a비트로
    # Convert b to an integer
    bb = int(float(b))

    # Check if b is negative
    if bb < 0:
        # Calculate 2's complement of b
        bb = (1 << a) + bb

    # Convert b to binary representation with a bits
    binary = format(bb, '0' + str(a) + 'b')

    return str(binary)



for code in codes:
    if "addi" in code:  #addi $a $b #c
        a=code.split(" ")[1].replace('$', '')
        b=code.split(" ")[2].replace('$', '')
        c=code.split(" ")[3].replace('#', '')
        a_5bit = make_bits(5,a)
        b_5bit = make_bits(5,b)
        c_16bit = make_bits(16,c)
        new_opcode="001000"+b_5bit+a_5bit+c_16bit
        opcodes.append(new_opcode)
    elif "add" in code: #add $a $b $c
        a=code.split(" ")[1].replace('$', '')
        b=code.split(" ")[2].replace('$', '')
```

```
실습 > 5stage_pipeline > pipeline > instruction_bin.t
00000000000000000000000000000000
00000000000000000000000000000000
00000000000000000000000000000000
00000000000000000000000000000000
00000000000000000000000000000000
00000000000000000000000000000000
00000000000000000000000000000000
00000000000000000000000000000000
00010000001000110000000000001111
00000000000000000000000000000000
00000000000000000000000000000000
00000000000000000000000000000000
00000000000000000000000000000000
00000000000000000000000000000000
00000000000000000000000000000000
00000000000000000000000000000000
00000000000000000000000000000000
00000000000000000000000000000000
00000000000000000000000000000000
00000000000000000000000000000000
00000000000000000000000000000000
00000000000000000000000000000000
00000000000000000000000000000000
00000000000000000000000000000000
00010000100000011111111111101011
00000000000000000000000000000000
00000000000000000000000000000000
00000000000000000000000000000000
00000000000000000000000000000000
00000000000000000000000000000000
00000000000000000000000000000000
00000000000000000000000000000000
00000000000000000000000000000000
00000000000000000000000000000000
```



우리가 예상한 값과 PCplus4가 일치함을 볼 수 있습니다.