

# Lab3 Report

이름: 권도현

학번: 2023065350

학과: 컴퓨터소프트웨어학부

## Overall Structure

### 1. CPP

ALU.cpp, RF.cpp: HW02 코드와 동일하다.

CTRL.cpp – splitInst()

```
void CTRL::splitInst(uint32_t inst, ParsedInst *parsed_inst) {
    parsed_inst->opcode = (inst >> 26) & 0x3F;
    parsed_inst->rs = (inst >> 21) & 0x1F;
    parsed_inst->rt = (inst >> 16) & 0x1F;
    parsed_inst->rd = (inst >> 11) & 0x1F;
    parsed_inst->shamt = (inst >> 6) & 0x1F;
    parsed_inst->funct = inst & 0x3F;
    parsed_inst->immi = inst & 0xFFFF;
    parsed_inst->immj = inst & 0x3FFFFFFF;
}
```

- 32bit 명령어에서 opcode와 관계없이 나올 수 있는 모든 경우를 세팅했다. >> 연산은 하위 비트를 무시하고 상위 비트만을 사용하기 위해 사용했다. & 연산은 shift 연산을 적용한 후, 상위 비트를 무시하기 위해 사용했다.

CTRL.cpp – signExtend()

```
void CTRL::signExtend(uint32_t immi, uint32_t SignExtend, uint32_t *ext_imm) {
    if (SignExtend) {
        *ext_imm = static_cast<int32_t>(immi << 16) >> 16;
    }
    else {
        *ext_imm = immi;
    }
}
```

Control signal의 SignExtend = 1 이라면, Sign Extension을 하도록 했다. Unsigned인 immi를 16bit Left shift하고 Signed로 바꾸어 right shift하였다. CPP에서는 Signed에 대해선

right shift가 Arithmetic인 점을 이용하였다.

CRTL.cpp – controlSignal

```
void CTRL::controlSignal(uint32_t opcode, uint32_t funct, Controls *controls) {
    controls->RegDst = 0;
    controls->Jump = 0;
    controls->Branch = 0;
    controls->JR = 0;
    controls->MemRead = 0;
    controls->MemtoReg = 0;
    controls->MemWrite = 0;
    controls->ALUSrc = 0;
    controls->SignExtend = 1;
    controls->RegWrite = 0;
    controls->ALUOp = 0b1111;
    controls->SavePC = 0;
}
```

Control Singal을 초기화하는 부분이다. 특이한 부분으로 ALUOp는 1111로 초기화했는데, 이는 정의되지 않은 연산으로 초기화하기 위해서이다. SignExtend를 1로 초기화한 이유는 우리가 사용하는 I-type의 명령어에서 Sign Extend를 하는 경우가 Zero Extend의 경우보다 많았기 때문이다.

```
if (opcode == 0) {
    controls->RegDst = 1;
    controls->RegWrite = (funct != FUNCT_JR) ? 1 : 0; // R-type 중 JR만 따로

    switch (funct) // ALUOp
    {
        case FUNCT_SLL:
            controls->ALUOp = ALU_SLL;
            break;
        case FUNCT_SRL:
            controls->ALUOp = ALU_SRL;
            break;
        case FUNCT_SRA:
            controls->ALUOp = ALU_SRA;
            break;
        case FUNCT_JR:
            controls->JR = 1;
            controls->Jump = 1;
            break;
        case FUNCT_ADDU:
            controls->ALUOp = ALU_ADDU;
            break;
        case FUNCT_SUBU:
            controls->ALUOp = ALU_SUBU;
            break;
        case FUNCT_AND:
            controls->ALUOp = ALU_AND;
            break;
        case FUNCT_OR:
            controls->ALUOp = ALU_OR;
            break;
        case FUNCT_XOR:
            controls->ALUOp = ALU_XOR;
            break;
        case FUNCT_NOR:
            controls->ALUOp = ALU_NOR;
            break;
        case FUNCT_SLT:
            controls->ALUOp = ALU_SLT;
            break;
        case FUNCT_SLTU:
            controls->ALUOp = ALU_SLTU;
            break;
        default:
            break;
    }
}
```

Opcode 별로 나누어서 control signal을 결정할 것이다. 먼저 R-type에 대해서 살펴보자. R-type 명령어는 기본적으로 RegDst와 RegWrite가 1이어야 한다. JR의 경우만 RegWrite가 0임으로 예외처리 해주었다. R-type 내에서의 명령은 funct로 구별된다. 따라서 funct와 ALUOp를 매핑해 줄 필요가 있었다. Funct에 따라 ALUOp를 매핑하는 동시에 JR은 JR = JUMP = 1로 추가적으로 세팅해주었다.

```
else if (opcode == 2) {
    controls->Jump = 1;
}
else if (opcode == 3) {
    controls->Jump = 1;
    controls->RegWrite = 1;
    controls->SavePC = 1;
}
else if (opcode == 4) {
    controls->ALUOp = ALU_EQ;
    controls->Branch = 1;
}
else if (opcode == 5) {
    controls->ALUOp = ALU_NEQ;
    controls->Branch = 1;
}
else if (opcode == 9) {
    controls->ALUOp = ALU_ADDU;
    controls->ALUSrc = 1;
    controls->RegWrite = 1;
}
else if (opcode == 10) {
    controls->ALUOp = ALU_SLT;
    controls->ALUSrc = 1;
    controls->RegWrite = 1;
}
else if (opcode == 11) {
    controls->ALUOp = ALU_SLTU;
    controls->ALUSrc = 1;
    controls->RegWrite = 1;
}
else if (opcode == 12) {
    controls->ALUOp = ALU_AND;
    controls->ALUSrc = 1;
    controls->RegWrite = 1;
    controls->SignExtend = 0;
}
else if (opcode == 13) {
    controls->ALUOp = ALU_OR;
    controls->ALUSrc = 1;
    controls->RegWrite = 1;
    controls->SignExtend = 0;
}
else if (opcode == 14) {
    controls->ALUOp = ALU_XOR;
    controls->ALUSrc = 1;
    controls->RegWrite = 1;
    controls->SignExtend = 0;
}
```

```

else if (opcode == 15) {
    controls->ALUOp = ALU_LUI;
    controls->RegWrite = 1;
}
else if (opcode == 35) {
    controls->ALUOp = ALU_ADDU;
    controls->MemRead = 1;
    controls->ALUSrc = 1;
    controls->MemtoReg = 1;
    controls->RegWrite = 1;
}
else if (opcode == 43) {
    controls->ALUOp = ALU_ADDU;
    controls->MemWrite = 1;
    controls->ALUSrc = 1;
}
}

```

명령어 별로 Control signal을 설정하였다. I - type의 경우, 각 명령어에 맞는 ALUOp를 지정해주었다. 각 명령어가 사용해야 하는 Control signal을 각각 1로 세팅했다.

```

// Access the instruction memory
mem.imemAccess(PC, &inst);
if (status != CONTINUE) return 0;

// IF state
// Split the instruction & set the control signals
ctrl.splitInst(inst, &parsed_inst);
ctrl.controlSignal(parsed_inst.opcode, parsed_inst.funct, &controls);
ctrl.signExtend(parsed_inst.immi, controls.SignExtend, &ext_imm);
if (status != CONTINUE) return 0;

// ID state
rf.read(parsed_inst.rs, parsed_inst.rt, &rs_data, &rt_data);

operand1 = rs_data;
operand2 = (controls.ALUSrc == 1 || controls.ALUOp == 13) ? ext_imm : rt_data;

// EX state
alu.compute(operand1, operand2, parsed_inst.shamt, controls.ALUOp, &alu_result);
if (status != CONTINUE) return 0;

// MEM (+PC Update)
mem.dmemAccess(alu_result, &mem_data, rt_data, controls.MemRead, controls.MemWrite);
if (status != CONTINUE) return 0;

// Update the PC
if (controls.JR) {
    PC_next = rs_data;
} else if (controls.Jump) {
    PC_next = ((PC + 4) & 0xF0000000) | (parsed_inst.immj << 2);
} else if (controls.Branch && alu_result) {
    PC_next = (PC + 4) + (ext_imm << 2);
} else {
    PC_next = PC + 4;
}

// WB
if (controls.RegDst) {
    wr_addr = parsed_inst.rd;
} else if (controls.SavePC) {
    wr_addr = 31;
} else {
    wr_addr = parsed_inst.rt;
}

```

imemAccess(PC, Inst)를 통해 Pc에서 명령어를 가져오고, splitInst()를 통해 opcode, rs, rt 등에 값을 넣는다. Opcode, funct를 이용하여 Control signal을 세팅하고, sign extension이 필요한지 여부까지 확인한다. Register file에서 데이터를 읽어오고, rt값을 사용할지, immediate값을 사용할 지를 결정하여 ALU 연산을 한다. 메모리 read, write를 위해 write addr에 alu\_result를 넣고, Control signal에 맞추어 메모리 접근을 할 수 있도록 한다. 이후, Control signal에 맞추어 PC값을 업데이트하고, Register write back address도 업데이트한다.

```
if (controls.SavePC) {
    wr_data = PC + 4;
} else if (controls.MemtoReg) {
    wr_data = mem_data;
} else {
    wr_data = alu_result;
}

rf.write(wr_addr, wr_data, controls.RegWrite);
if (status != CONTINUE) return 0;
```

wr\_data도 control signal에 맞추어 업데이트하고, Register file에 write할 수 있도록 한다.

## 2. Verilog

전체적인 구조는 CPP와 비슷하다. CPP와의 차이점 위주로 서술하겠다.

ALU.v: HW02와 같으나, 즉시 값이 할당되도록 (=)를 사용하였다.

RF.v: HW02와 동일하다.

CTRL.v: CTRL.CPP와 완벽하게 동일하다.

CPU.v

```
assign halt = (inst == 32'b0);

assign opcode = inst[31:26];
assign rs     = inst[25:21];
assign rt     = inst[20:16];
assign rd     = inst[15:11];
assign shamt  = inst[10:6];
assign funct  = inst[5:0];
assign immi   = inst[15:0];
assign immj   = inst[25:0];

assign ext_imm = SignExtend ? ({16{immi[15]}}, immi) : {16'b0, immi}; // SignExtend

assign rd_addr1 = rs;
assign rd_addr2 = rt;

assign operand1 = rd_data1;
assign operand2 = (ALUSrc == 1 || ALUOp == 4'd13) ? ext_imm : rd_data2;

assign mem_addr = alu_result;
assign mem_write_data = rd_data2;
```

바로 값을 할당 받을 수 있도록 하였다.

```

always @(*) begin
    if (JR)
        PC_next = rd_data1;
    else if (Jump)
        PC_next = ((PC + 32'd4) & 32'hF0000000) | immj << 2;
    else if (Branch == 1 && alu_result == 1)
        PC_next = PC + 32'd4 + (ext_imm << 2);
    else
        PC_next = PC + 32'd4;

    if (RegDst)
        wr_addr = rd;
    else if (SavePC)
        wr_addr = 5'd31;
    else
        wr_addr = rt;

    if (SavePC)
        wr_data = PC + 32'd4;
    else if (MemtoReg)
        wr_data = mem_read_data;
    else
        wr_data = alu_result;
end

```

Reg 타입들은 always @(\*) begin 안에서 조건을 확인하고 값을 할당 받을 수 있도록 하였다.

```

CTRL ctrl (
    .opcode(opcode),
    .funct(funct),
    .RegDst(RegDst),
    .Jump(Jump),
    .Branch(Branch),
    .JR(JR),
    .MemRead(MemRead),
    .MemtoReg(MemtoReg),
    .MemWrite(MemWrite),
    .ALUSrc(ALUSrc),
    .SignExtend(SignExtend),
    .RegWrite(RegWrite),
    .ALUOp(ALUOp),
    .SavePC(SavePC)
);

RF rf (
    .clk(clk),
    .rst(rst),
    .rd_addr1(rd_addr1),
    .rd_addr2(rd_addr2),
    .rd_data1(rd_data1),
    .rd_data2(rd_data2),
    .RegWrite(RegWrite),
    .wr_addr(wr_addr),
    .wr_data(wr_data)
)

```

```

MEM mem (
    .clk(clk),
    .rst(rst),
    .inst_addr(PC),
    .inst(inst),
    .MemWrite(MemWrite),
    .mem_addr(mem_addr),
    .mem_write_data(mem_write_data),
    .mem_read_data(mem_read_data)
);

ALU alu (
    .operand1(operand1),
    .operand2(operand2),
    .shamt(shamt),
    .funct(ALUOp),
    .alu_result(alu_result)
);

```

CPU와 submodule을 연결하였다. 위의 assign 부분에서 연결한 것도 있고, .inst\_addr(PC) 나 .inst(inst) 처럼 바로 연결한 것도 존재한다. 연결되는 wire는 CPU.cpp와 동일한다.

## Results








### 1. CPP: Testcase 2의 경우

```

kwondh3236@NOTKDH:~/computer_architecture/HW03/cpp$ ./cpu testcase2
Loading instruction memory...
Starting CPU simulation...
c
Simulation done successfully.
RF states after the program execution
$zero 00000000
$at    fff90000
$v0    00000005
$v1    00000000
$a0    00000000
$a1    00000000
$a2    00000000
$a3    00000000
$t0    fff8e8df
$t1    fff9d354
$t2    fff8e8df
$t3    fff8e8df
$t4    ffea7884
$t5    fff9f0cb
$t6    00000000
$t7    00000000
$s0    fff8e8df
$s1    fff8e8df
$s2    fff8e8df
$s3    fff9d354
$s4    ffea7884
$s5    00042808
$s6    00000000
$s7    00000000
$t8    00000000
$t9    00000000
$k0    00000000
$k1    00000000
$gp    00001800
$sp    00003ffc
$fp    00000000
$ra    00000000

```

### 2. Verilog

Name	Value
>  i[31:0]	8192
>  FAILED[31:0]	0
 clk	1
 rst	0
 halt	1
>  register_file[0:31][31:0]	00000000,22
>  memory[0:8191][31:0]	3c01a14c,342

## Difficulties

1. SplitInst에서 아래와 같이 코드를 짰더니, Rtype의 경우 immj, immi 영역이 초기화가 되지 않고, I, J tpye의 경우 rd,rt,shamt, funct 등이 초기화되지 않아 에러가 발생하였다.

```

78 void CTRL::splitInst(uint32_t inst, ParsedInst *parsed_inst) {
79     parsed_inst->opcode = (inst >> 26) & 0x3F;
80
81     switch(static_cast<Opcode>(parsed_inst->opcode)) {
82     case OP_RTYPE:
83         parsed_inst->rs = (inst >> 21) & 0x1F;
84         parsed_inst->rt = (inst >> 16) & 0x1F;
85         parsed_inst->rd = (inst >> 11) & 0x1F;
86         parsed_inst->shamt = (inst >> 6) & 0x1F;
87         parsed_inst->funct = inst & 0x3F;
88         break;
89
90     case OP_J:
91         parsed_inst->immj = inst & 0x3FFFFFFF;
92         break;
93
94     case OP_JAL:
95         parsed_inst->immj = inst & 0x3FFFFFFF;
96         break;
97
98     default: // I-type
99         parsed_inst->rs = (inst >> 21) & 0x1F;
100        parsed_inst->rt = (inst >> 16) & 0x1F;
101        parsed_inst->immi = inst & 0xFFFF;
102        break;
103    }
104 }

```

2. Control Signal에서 아래와 같이 opcode마다 control signal을 세팅하는 것이 아니라, control signal을 opcode에 따라 세팅하도록 코드를 짰더니 특정 opcode에서 세팅 되면 안 되는 control signal이 세팅 되는 경우가 있었다. 이후, opcode마다 control signal을 세팅할 수 있도록 코드를 바꾸었다.



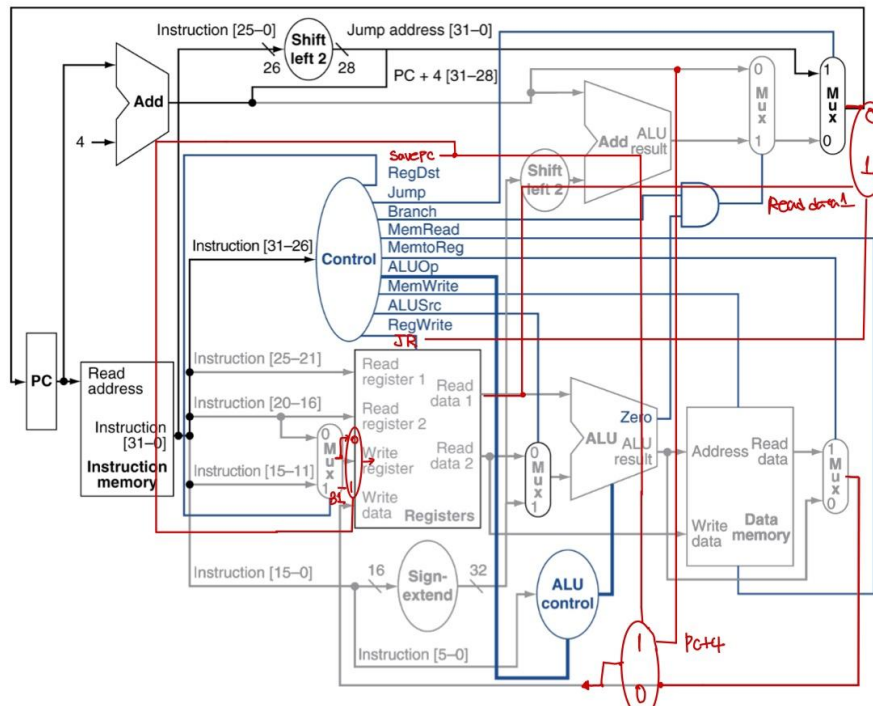
```

CTRL.cpp X
CTRL.cpp > controlSignal(uint32_t, uint32_t, Controls *)
9 void CTRL::controlSignal(uint32_t opcode, uint32_t funct, Controls *controls) {
23     if (opcode == 0) {
24         switch (funct) // ALUOp
64     }
65
66     controls->RegDst = opcode == 0 ? 1 : 0;
67     controls->Jump = (opcode == 2 || opcode == 3) ? 1 : 0; // completed
68     controls->Branch = (opcode == 4 || opcode == 5) ? 1 : 0;
69     controls->MemRead = opcode == 35 ? 1 : 0; // completed
70     controls->MemtoReg = opcode == 35 ? 1 : 0;
71     controls->MemWrite = opcode == 43 ? 1 : 0; //completed
72     controls->ALUSrc = (opcode != 0 && opcode != 4 && opcode != 5) ? 1 : 0;
73     controls->SignExtend = (opcode == 9 || opcode == 10 || opcode == 11 || opcode == 35 || opcode
74     controls->RegWrite = (opcode != 43 && opcode != 4 && opcode != 5 && opcode != 2 && !(opcode !=
75     controls->SavePC = (opcode == 3) ? 1 : 0;
76     controls->JR = (opcode == 0 && funct == FUNCT_JR) ? 1 : 0;
77
78     // ALUOp except R-type
79     if (opcode == 4) controls->ALUOp = 11;
80     else if (opcode == 5) controls->ALUOp = 12;
81     else if (opcode == 9 || opcode == 35 || opcode == 43) controls->ALUOp = 0;
82     else if (opcode == 12) controls->ALUOp = 1;
83     else if (opcode == 13) controls->ALUOp = 3;
84     else if (opcode == 15) controls->ALUOp = 13;
85     else if (opcode == 10) controls->ALUOp = 9;
86     else if (opcode == 11) controls->ALUOp = 10;
87     else if (opcode == 14) controls->ALUOp = 8;
88 }

```

3. Verilog는 CPP 기반으로 코딩하여 별 다른 어려움은 없었다.

## Implementation of JAL / JR instruction



- 기존 CPU 구조에 3개의 MUX를 추가하여 JAL과 JR 명령어를 구현하였다.

- JAL 먼저 살펴 보면, "SavePC = 1"일 때 Write data에 PC + 4, Write address에 d'31 (\$r31) 이 할당되어야 한다. 이 부분을 MUX로 MEM에서 읽어온 data와 PC + 4를 SavePC에 따라서 선택하도록 하였고, Write address에는 기존에 write address에 사용할 값과 31중 SavePC에 따라서 선택할 수 있도록 했다.
- JR을 보면 rs에 저장되어 있는 주소로 Jump 해야 한다. 따라서 MUX로 Read\_data1과 기존 Jump target을 JR signal에 따라서 선택할 수 있도록 했다.

코드로도 확인할 수 있다.

#### 1. JR

```
if (controls.JR) {
    PC_next = rs_data;
```

#### 2. JAL

```
else if (controls.SavePC) {
    wr_addr = 31;
```

```
if (controls.SavePC) {
    wr_data = PC + 4;
```