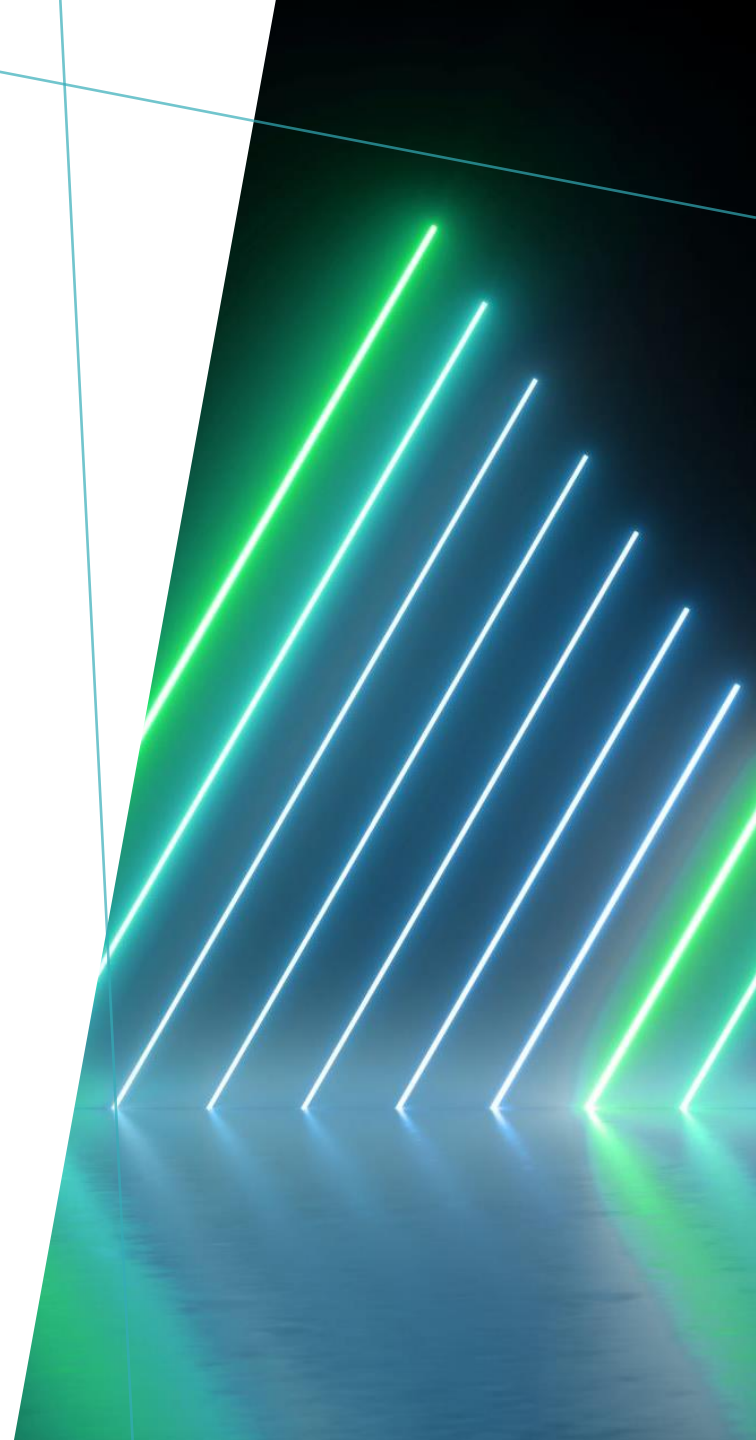
디지털 로직 설계 엔지니어
김민성 포트폴리오



김민성
010-8849-5636
ms9131206@naver.com

목차

자기소개

· 학력

2024 동아대학교 전자공학과 학사 졸업

· 수상 및 대외활동

2022 현대차 온드림 다빈치스쿨 교육봉사활동

2023 동아대 캡스톤 디자인 경진대회 장려상

2024 IDEC 시스템반도체 실무인력양성교육 진행 중

· 수료

Idec Verilog을 이용한 Digital System 설계 수료

Idec 프로세서 연산기 기본 및 설계 실습 수료

Idec SPI 설계 및 실습 수료

Idec 디지털 시스템 반도체 ASIC 설계 수료

Idec AMD Xilinx FPGA HW 구성 이해 및 설계 수료

e-koreatech 반도체 개발 디지털 회로 설계 part 1 수료

e-koreatech 디지털 회로 설계 수료

프로젝트 소개

1. 32bit pipeline mips cpu 아키텍처 설계 및 검증

2. 전송 latency를 최적화한 FIFO 자체 설계 및 검증

3. alu_acc with mul_div_control 설계 및 검증

4. 급발진 인식 및 행동대처 매뉴얼 전파 시스템 개발

1. 32bit pipeline mips cpu

프로젝트 배경

: cpu 는 여러 설계 분야에서 사용되는 지식이 모두 함축 되어있는 IP라고 생각하기 때문

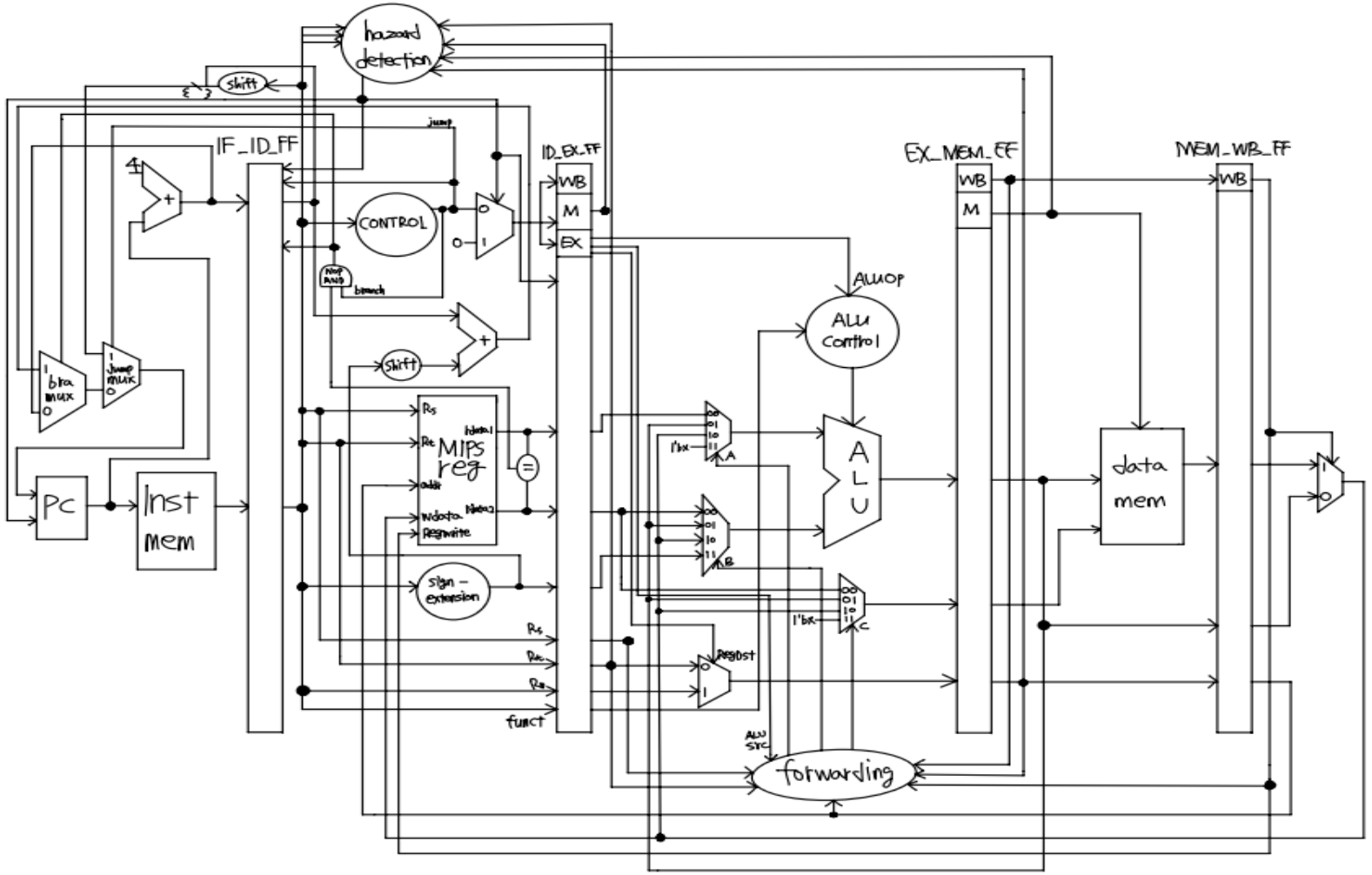
설계 명시(RTL design specification)

: 32비트 길이의 R,I,J타입의 명령어를 순차적으로 생성한 후 이것을 오류 없이 명령대로 수행시키는 것

하드웨어 구조도(high level design)

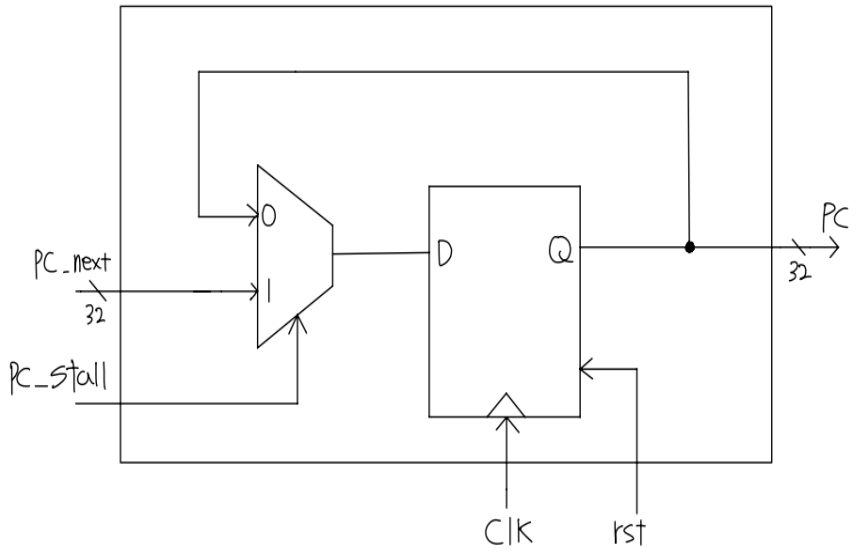
설계 절차

1. Single cycle을 먼저 설계
2. 5 stage로 명령어 사이클 분류
3. Hazard 문제 해결을 위한 모듈 추가 설계



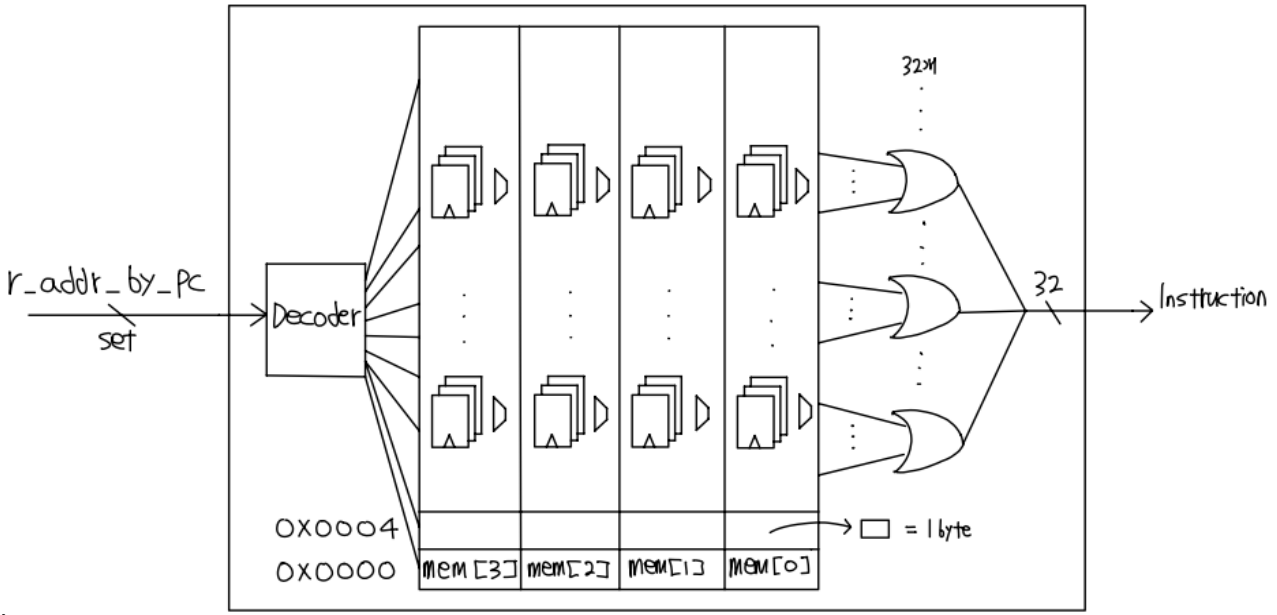
하드웨어 구조도(low level design)

PC



- Hazard detect 시 pipeline stall 위한 mux 추가

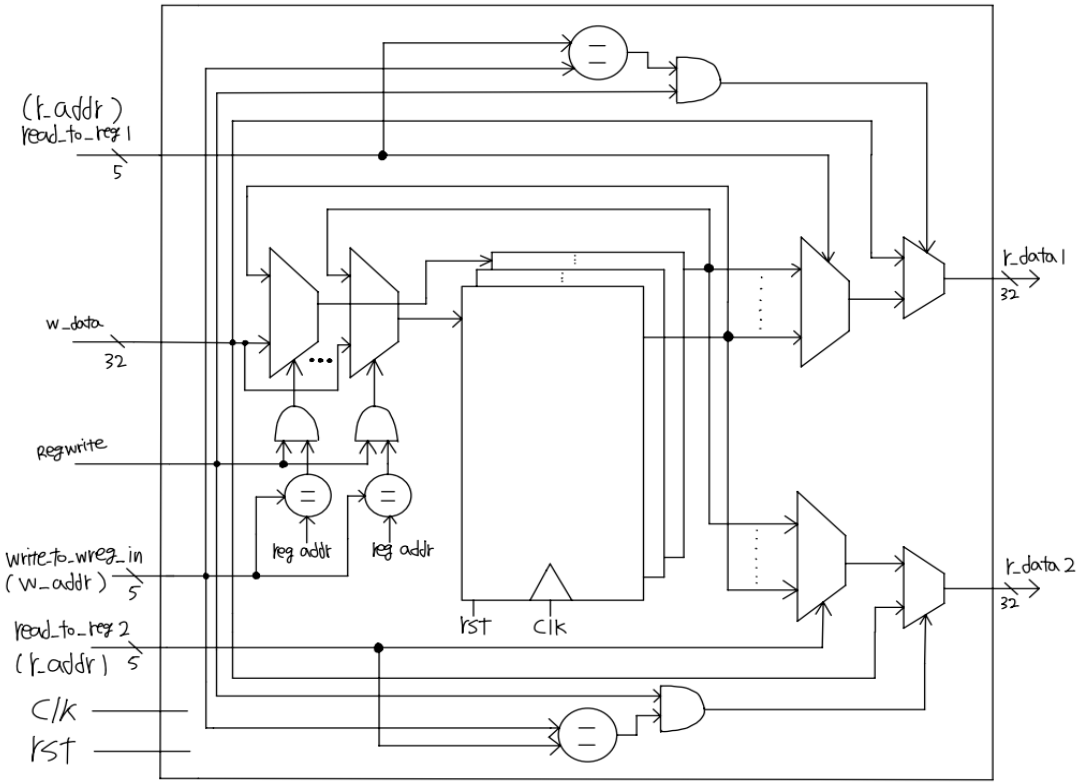
Instruction memory



- TR레벨의 mem cell 구조는 FF이 아닌 ROM 구조로 설계하여 power, area 최적화 필요

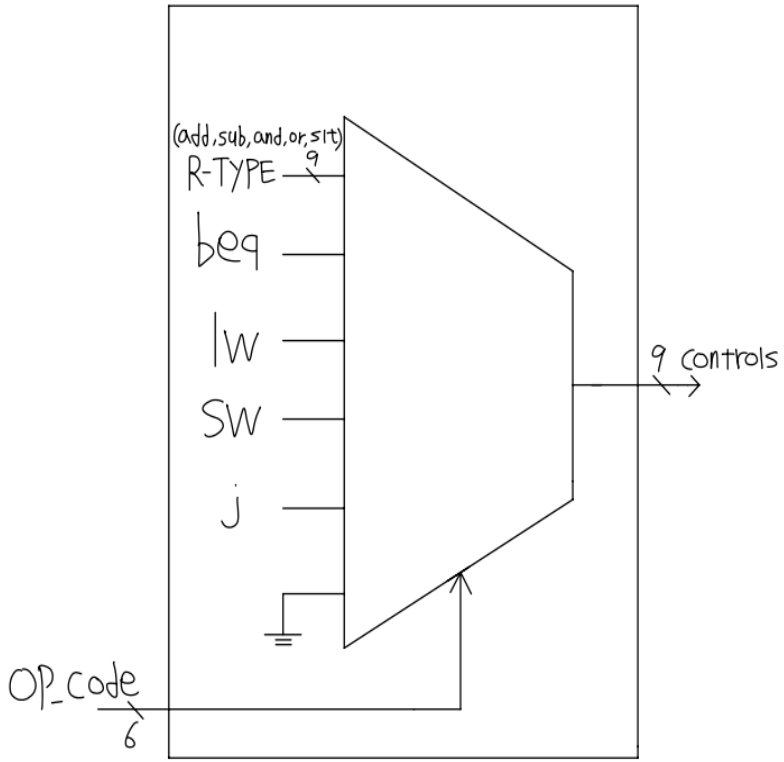
하드웨어 구조도(low level design)

Mips registers



- r_addr == w_addr 이면 write되는 데이터를 바로 read 하여 stage 최적화

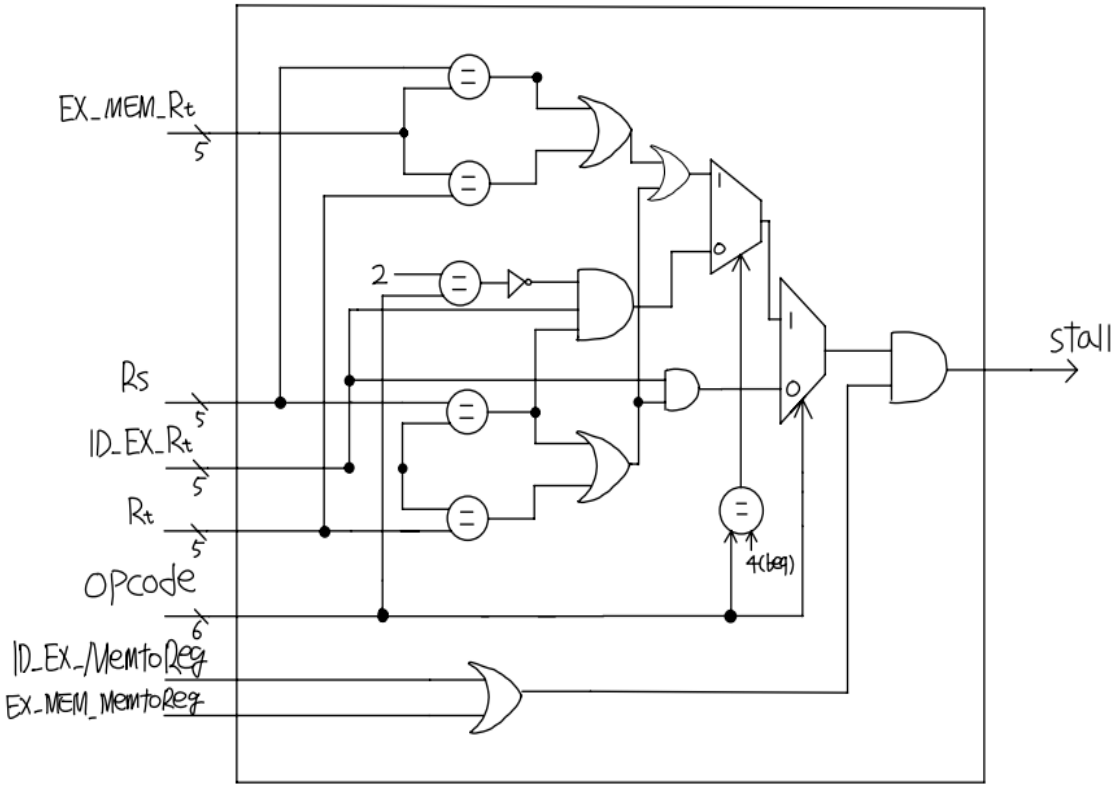
Control unit



- 연산, 읽기, 쓰기, 분기 명령어 기능 구현

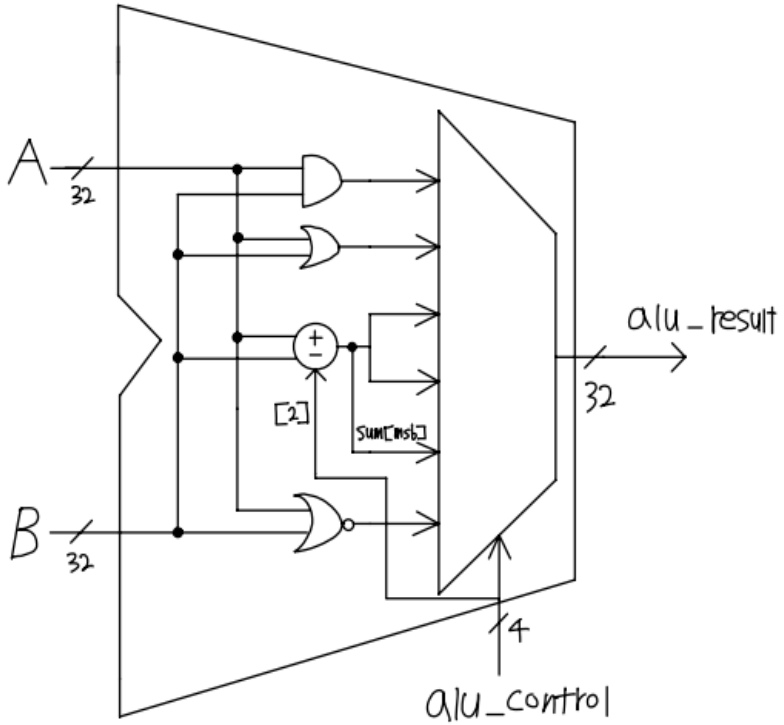
하드웨어 구조도(low level design)

Hazard detection



- beq에 의한 hazard는 예외적으로 2 bubble을 발생시키도록 설계

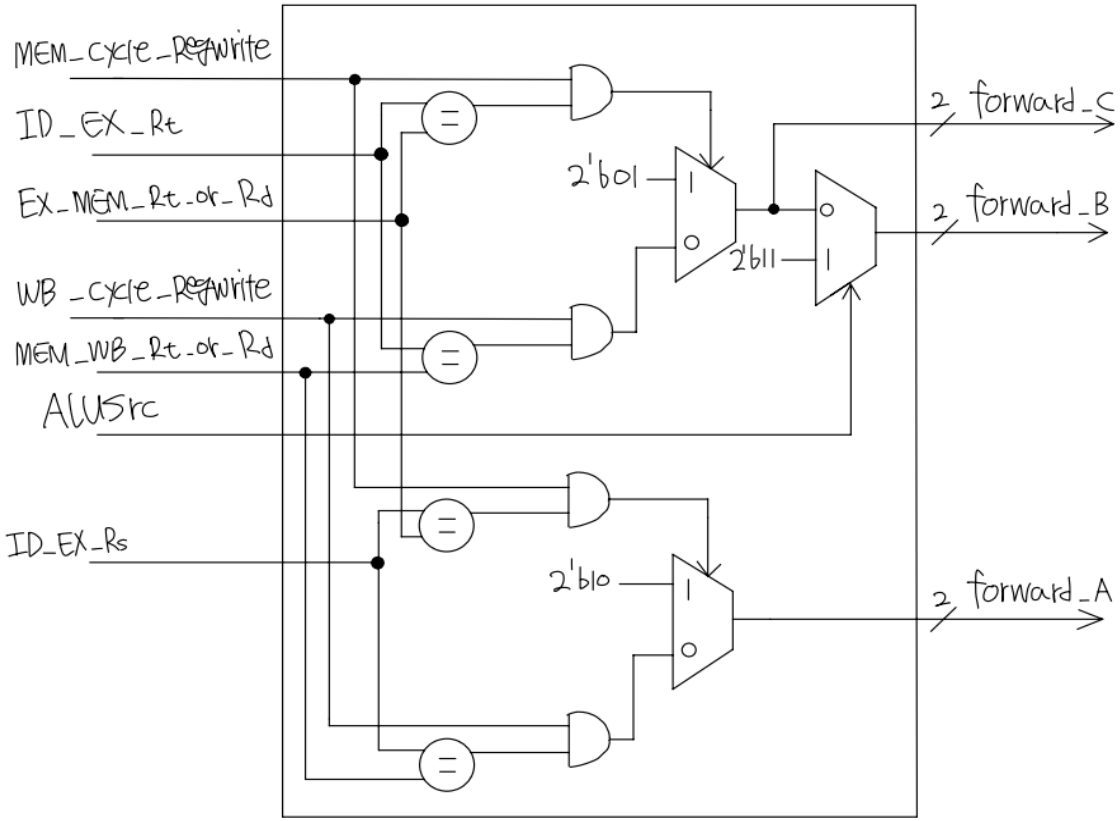
ALU



- Funct와 Aluop에 의한 연산 컨트롤러 별도 설계

하드웨어 구조도(low level design)

Forwarding



- Forward_A : Rs필드와의 hazard 발생관련
- Forward_B : w_data(to data mem)와의 hazard 발생관련
- Forward_C : Rt필드와의 hazard 발생관련

  + immediate 필드 값 출력 제어를 위한 mux 추가 설계

이하 설계 내용 및 검증은
별도 첨부하였습니다!

2. Latency를 최적화한 FIFO

프로젝트 배경

: M → FIFO → S 기능 학습 도중,

데이터가 AXI hs기반의 FIFO를 통과할 때 latency의 저해요소를 발견

설계 명시(RTL design specification)

: FULL과 EMPTY 상황에서 1cycle 지연 없이 최적화된 데이터 전달과정 설계

➡ 데이터가 FIFO에 저장되는 동시 사이클에 데이터를 access 가능하도록

+ 향후 과제는 크리티컬 패스 딜레이 분석을 통해 기존 대비 시스템 주파수를 감소시키지 않는지 확인하는 것!

Latency 최적화 핵심 로직

```
// full, empty logic
assign back_eq = (w_back_in == r_back_in);

assign empty_condi1 = (!back_eq) && (r_ptr == (FIFO_DATA_DEPTH-1)) && (w_ptr == 0);
assign empty_condi2 = back_eq && ( r_ptr == (w_ptr-1));
assign empty_en =  (!w_hs) && (empty_condi1 || empty_condi2);

assign full_condi1 = back_eq && (w_ptr == (FIFO_DATA_DEPTH-1)) && (r_ptr == 0);
assign full_condi2 = (!back_eq) && ( w_ptr == (r_ptr-1));
assign full_en  = (!r_hs) && (full_condi1 || full_condi2);
```

```
// w_hs, m_valid output
assign w_hs = s_valid && s_ready;
assign m_valid = (c_state != S_EMPTY) || w_hs;

// r_hs, s_ready output
assign r_hs = m_valid && m_ready;
assign s_ready = (c_state != S_FULL ) || r_hs;

// data transfer
assign w_data = s_data;
assign m_data = (w_hs && (c_state == S_EMPTY)) ? s_data : r_data;
```

```
// fsm
always @(posedge clk) begin
    if(rst) begin
        c_state <= S_EMPTY;
    end
    else begin
        c_state <= n_state;
    end
end

always @(*) begin
    n_state = c_state;
    case (c_state)
        S_RUN:
            if(w_hs && full_en)
                n_state = S_FULL;
            else if(r_hs && empty_en)
                n_state = S_EMPTY;
        S_EMPTY:
            if(w_hs && (!r_hs))
                n_state = S_RUN;
        S_FULL:
            if(r_hs && (!w_hs))
                n_state = S_RUN;
    endcase
end
```

- mem과 별도로 FSM 기반의 컨트롤러 설계

➡ FIFO mem 데이터 바탕의 full, empty 계산

- empty && S의 read 요청 시

➡ FIFO에 입력되고 있던 데이터를 동시에 S로 전송

- full && M의 write 요청 시

➡ S로 데이터를 내보내는 동시에 M로부터 write도 가능하도록 함.

결과 비교

기존의 FIFO(강의 자료)

```
run all
initialize value [                    0]
Reset! [                              0]
Start! [                            125]
Finish! [                           2435]
$finish called at time : 2435 ns : File
```

Latency 최적화 FIFO(자체 설계)

```
run all
initialize value [                    0]
rst! [                              0]
Start! [                            125]
Finish! [                           2305]
$finish called at time : 2305 ns : File
```

- testbench 검증 방식 :

1. 0~99의 전송데이터를 순차적으로 생성 후 전송 마다 hs를 카운트하는 방식
2. M의 valid 와 S의 ready 신호를 랜덤으로 생성

➡ hs 카운트가 99에 도달할 때 까지 걸린 시간으로 finish time을 계산

➡ 데이터의 오류 여부는 전송 데이터를 파일로 dump후 기존 FIFO와의 일치 여부로 판단

➡ 일치 확인 완료

3. Alu_acc with mul_div_control

프로젝트 배경

: 프로세서의 데이터 연산처리과정에 대해 흥미를 느끼고 이것을 이론적으로 배우는 것을 넘어 직접 설계해보는 과정을 통해 실무적인 RTL 설계 능력을 기르기 위함

설계 명시(RTL design specification) + 코드 설명 시 각 모듈의 기능 스펙에 대해 자세히 설명함.

: 4비트인 두 피연산자의 사칙연산이 모두 가능하며 이는 제어신호를 통해 선택할 수 있게 함. 연산 후 데이터는 ACC에 저장되고 업데이트 될 수 있도록 함

하드웨어 구조도(high level design)

ALU

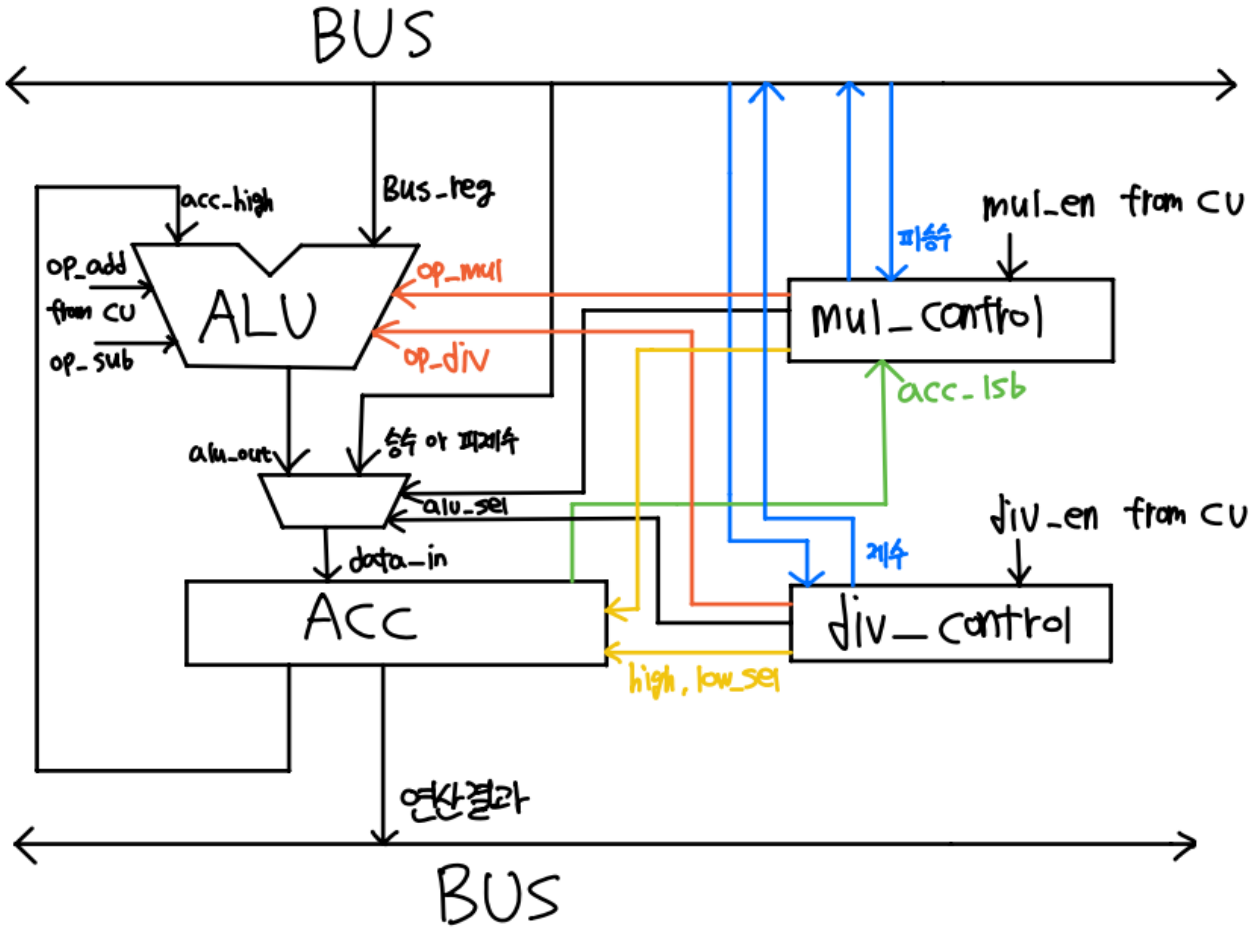
: 4비트 adder로 구성, op 제어신호를 통해 덧셈 or 뺄셈을 수행 후 연산결과를 ACC로 출력

ACC

: 곱셈과 나눗셈은 덧셈과 뺄셈의 연속과정. 이 때 중간 연산결과를 임시적으로 저장 후 다시 ALU로 보냄

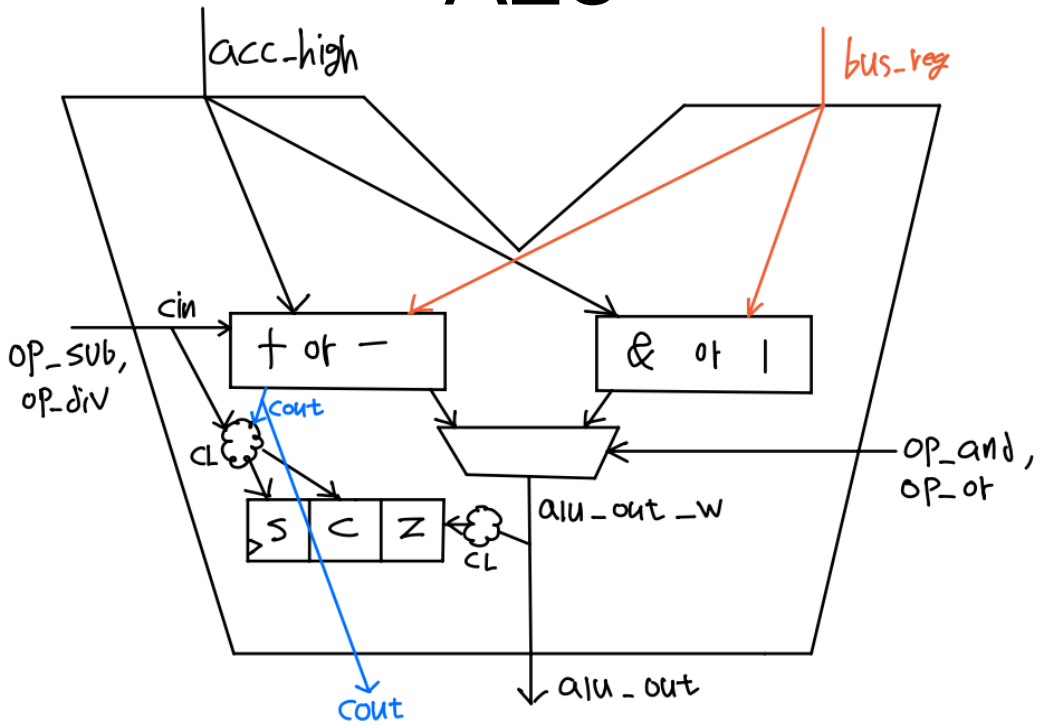
mul or div_control unit

: 곱셈, 나눗셈에 따른 적절한 ACC의 시프트 제어신호를 출력

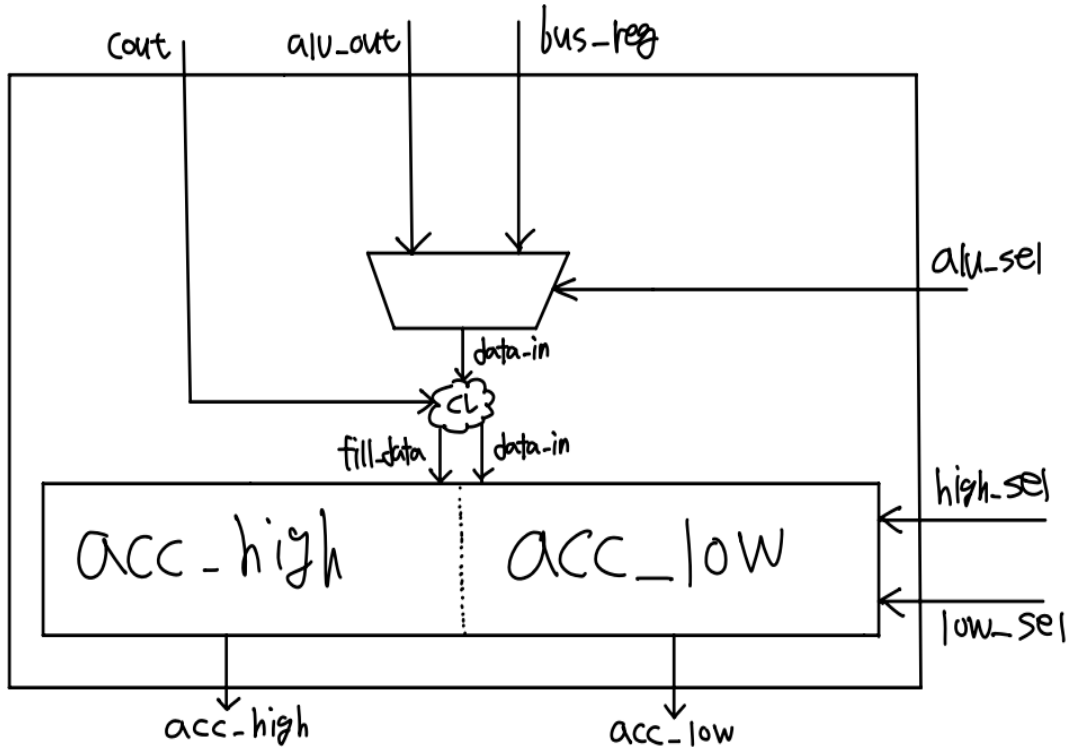


하드웨어 구조도(low level design)

ALU

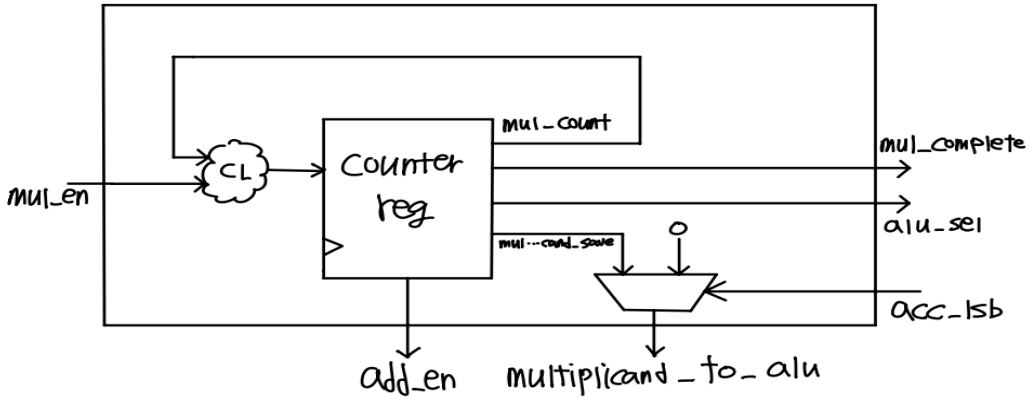


ACC

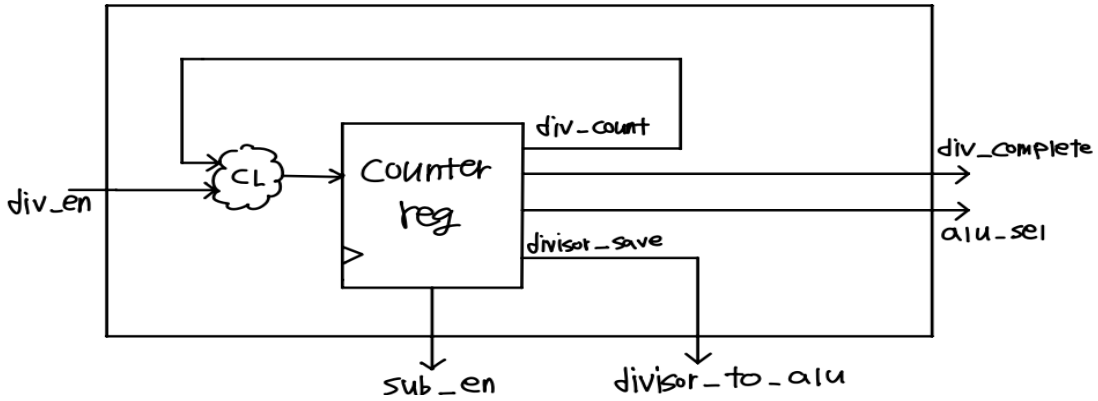


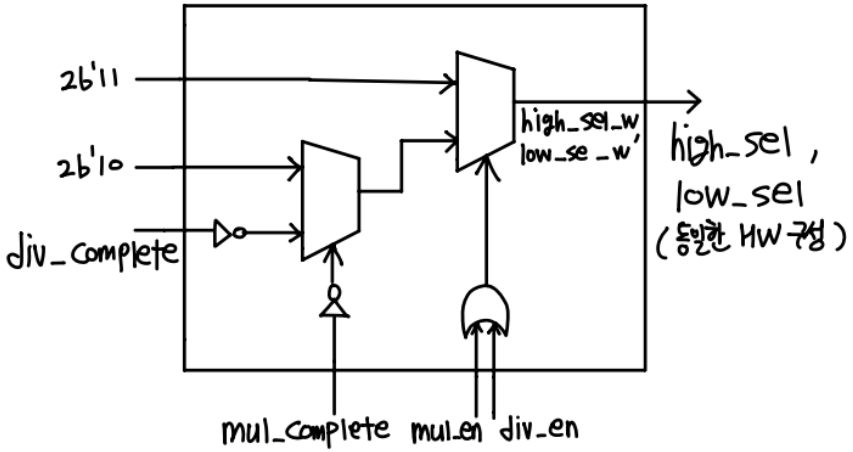
하드웨어 구조도(low level design)

Mul_controller



Div_controller





alu_acc(top 모듈)의 acc data sel 모듈

이하 설계 내용 및 검증은
별도 첨부하였습니다!

감사합니다!