Project 3: Simulation Pipelined Execution

Submitter: Jiwon Jang (202211167)

Contents Table

- 1. Compile and Execution
- 2. Definition of Class and Function
- 3. Code/Structure idea
- 4. Code flow
- 5. Results

1. Compile and Execution

언어는 C++를 사용하였고, 컴파일러는 g++ 9.5.0을 이용하였다. 다음 명령어를 치면 컴파일할 수 있다.

g++ -o homework3 homework3.cpp

이후 파일 실행은 다음과 같이 진행했다. /homework3 sample.o 순서로 입력하면, 프로그램이 binary file 읽어와 모든 인스트럭션이 수행되고난 후의 결과를 출력을 해준다. 이 외에도 -m, -n, -d, -p, -atp, -antp 옵션을 입력하면 assignment에서 요구한 출력이 나온다.

-m begin_memory:end_memory: begin_memory 부터 end_memory에 담겨있는 내용 출력

-n num of instruction: num of instructruction 만큼의 instruction 수행 후 레지스터 값 출력

-d: 매 사이클 마다 레지스터 값 출력

-p: 매 사이클 마다 각 파이프라인 단계에서 실행되고 있는 instruction의 PC 값 출력

ex. $\{0x400000|0x400004||||\}$

-atp: always taken 분기 예측기 사용

-antp: always not taken 분기 예측기 사용

<주의 사항>

-n 0 option과 -d option이 함께오면 아무것도 출력되지 않는다.

-n num option을 입력하면 WB stage가 num개 만큼 실행되었을 때의 결과를 출력한다

ex. -n 3: {7|6|5|4|3} (3 번째 instruction 이 WB에 왔을 때 종료)

-n option을 사용하지 않으면 completion cycle을 출력한다. 이 때 completion 조건은 {||||}}, 즉 파이프 라인이 다 비었을 때 이다.

-m option과 -d option이 함께오면 매 출력마다 메모리가 출력된다.

-p option 사용시 stall된 instruction은 빈 칸 으로 표현된다.

2. Definition of Class and Function

이번 과제에서 구현한 클래스와 함수에 대한 설명이다. 코드의 흐름을 설명하기 위한 배경 설명이다.

library>

총 5개의 library를 이용하였다.

iostream: 표준 입출력을 지원한다.

fstream: 파일 생성 및 read/write를 지원한다.

string: 문자열 형식을 지원한다.

vector: vector 자료형을 지원한다.

bitset: 이진수 변화을 지원한다.

<class>

class는 Memory(instruction, data), Register(register, state reg), Program Counter를 관리하기 위해서 사용하였다.

1) PROGRAMCOUNTER class definition

private:

int program counter: PC값을 담고있다.

vector<DATA*> data list: 해당 라벨에 속해 있는 data들이 담겨있는 리스트이다.

string line: 해당 라벨이 속해있는 '줄'이 담겨있다.

public:

PROGRAMCOUNTER(): PRGRAMCOUNTER 클래스의 constructor이다. constructor에서는 initializing 진행한다. PC를 0x400000로 초기화 하고, PC_pointer 값을 해당 instance로 초기화 한다.

static PROGRAMCOUNTER* PCPointer: 모든 함수에서 PC값에 접근할 수 있게 하기 위한 static pointer 변수이다.

void UpdateProgramCounter(int update_address): PC값을 업데이트 해주는 함수이다.

int GetPC(): PC값을 가져오는 함수이다.

2) REGISTER class definition

private:

int register num: 레지스터 번호가 담겨있다.

int data: 레지스터에 담겨있는 데이터가 담겨있다.

public:

REGISTER(int register_num): REGISTER 클래스의 constructor이다. constructor에서는 레지스터 번호를 초기화 하고, register list에 해당 instance를 추가해 준다.

static vector<REGISTER*>register list: RGISTER class의 instance들이 담겨있는 리스트이다.

int ReadRegNum(): 레지스터 번호를 읽기 위한 함수이다.

void UpdateData(int update_data): 레지스터 값을 업데이트하기 위한 함수이다.

int ReadData(): 레지스터 데이터를 읽기 위한 함수이다.

3) STATE REGISTER class definition

private:

int register num: state 레지스터 번호가 담겨있다.

int data: state 레지스터에 담겨있는 데이터가 담겨있다.

public:

STATE_REGISTER(int register_num): STATE_REGISTER 클래스의 constructor이다. constructor에서는 state 레지스터 번호를 초기화 하고, register list에 해당 instance를 추가해 준다.

static vector< STATE_REGISTER *>register_list: STATE_REGISTER class의 instance들이 담겨있는 리스트이다.

int ReadRegNum(): 레지스터 번호를 읽기 위한 함수이다.

void UpdateData(int update data): 레지스터 값을 업데이트하기 위한 함수이다.

int ReadData(): 레지스터 데이터를 읽기 위한 함수이다.

3) TEXT MEMORY class definition

private:

string line: Text가 담겨있다.

int address: Text의 주소가 담겨있다.

public:

TEXT_MEMORY(string line, int address): TEXT_MEMORY 클래스의 constructor이다. constructor에서는 line과 address를 초기화 해주고, text_memory_list에 해당 instance를 추가해 준다.

static vector<TEXT_MEMORY*>text_memory_list: TEXT_MEMORY class의 instance들이 담겨있는 리스트이다.

string ReadData(): line(Text)을 읽기 위한 함수이다.

int ReadAddress(): line(Text)가 담겨있는 주소를 읽기 위한 함수이다.

void UpdateData(string update data): line(Text)을 업데이트 하기 위한 함수이다.

3) DATA MEMORY class definition

private:

string line: Data가 담겨있다.

int address: Data의 주소가 담겨있다.

public:

DATA_MEMORY(string line, int address): DATA_MEMORY 클래스의 constructor이다. constructor에서는 line과 address를 초기화 해주고, data memory list에 해당 instance를 추가해 준다.

static vector<DATA_MEMORY*>data_memory_list: DATA_MEMORY class의 instance들이 담겨있는 리스트이다.

string ReadData(): line(Data)을 읽기 위한 함수이다.

int ReadAddress(): line(Data)가 담겨있는 주소를 읽기 위한 함수이다.

void UpdateData(string update data): line(Data)을 업데이트 하기 위한 함수이다.

<other functions>

main 함수를 제외한 함수의 종류는 크게 일곱 가지로 구분할 수 있다.

- 진수 변환을 하는 함수들: Hexadecimaler(string num), Binaryer(int num, int size)
- 출력을 위한 함수들: PrintRgisterData(), PrintMemoryContents(int begin_address, int end address), PrintPipeLineContents(int a, int b, int c, int d, int e)
- 산술 연산을 함수: ALU()
- 레지스터와 메모리에 접근해 데이터를 읽고, 쓰는 함수들: int ReadReg(int RegNum), void WriteReg(int RegNum, int Data), string ReadMem(int Address), void WriteMem(int Address, string Data), int ReadStateReg(int RegNum), void WriteStateReg(int RegNum, int Data)
- 각 instruction을 관리하는 함수들: int R_format(int int_line), int Instruction(string line)
- Pipeline의 각 stage를 담당하는 함수들: int IF_stage(string line), vector<int> ID_stage(int 32bit_line, int atp), void EX_stage(vector<int> control_signal), void MEM_stage(vector<int> control_signal)
- Emulator 함수: void Emulator(ifstream& input, int d, int n, int p, int atp, int num_of_instruction, int begin_address_int, int end_address_int, int m)

이를 조금 더 자세히 알아보자.

1) 진수 변환을 하는 함수들

Hexadecimaler(string num): 2진수를 16진수로 변환하는 함수이다. 이 때 16진수의 자리를 고정하지 않는다.

Binaryer(int num, int size): 10진수를 2진수로 변환하는 함수이다. bitset 라이브러리를 사용하였다.

2) 출력을 위한 함수들

PrintRgisterData(): 레지스터를 읽어와 레지스터의 정보를 assignment가 요구하는 출력 형식으로 출력하는 함수이다.

PrintMemoryContents(int begin_address, int end_address): 메모리를 읽어와 메모리의 정보를 assignment가 요구하는 출력 형식으로 출력하는 함수이다.

PrintPipeLineContents(int a, int b, int c, int d, int e): 각 파이프라인 스테이지에 들어있는 instruction의 PC값을 assignment가 요구하는 출력 형식으로 출력하는 함수이다.

3) 산술 연산을 함수

ALU(): ALU control(ex. 0000: AND, 0001: OR ... etc ...)과 인자 두 개를 받아서 ALU 연산을 하는 함수이다.

4) 레지스터와 메모리에 접근해 데이터를 읽고, 쓰는 함수들

int ReadReg(int RegNum): 레지스터의 값을 읽는 함수이다.

void WriteReg(int RegNum, int Data): 레지스터의 값을 업데이트하는 함수이다.

string ReadMem(int Address): 메모리의 값을 읽는 함수이다.

void WriteMem(int Address, string Data): 메모리의 값을 업데이트하는 함수이다.

int ReadStateReg(int RegNum): 레지스터의 값을 읽는 함수이다.

void WriteStateReg(int RegNum, int Data): 레지스터의 값을 업데이트하는 함수이다.

6) <u>각 instruction을 관리하는 함수들</u>

int R_format(int int_line): function field와 state reg들을 바탕으로 instruction들을 처리하는 함수이다.

int Instruction(string line): opcode와 state reg들을 바탕으로 instruction들을 처리하는 함수이다.

6) Emulator 함수

void Emulator(ifstream& input, int d, int n, int p, int atp, int num_of_instruction, int begin_address_int, int end_address_int, int m): assignment 3의 요구에 맞게 파이프라이닝을 하여 병렬적으로 instruction을 실행시키고, 시스템(reg, mem)을 관리하는 함수이다.

다음은 Emulator 함수의 일부이다.

```
control_signal_WB = control_signal_MEM;

if (pipelining[4] != 0) { // ==0 인 경우는 noop 가 들어온 상황

(= stall)

WB_stage(control_signal_WB);

pipelining_finish[4] = 1;

instruction_counter++;

}

control_signal_MEM = control_signal_EX;
```

```
if (pipelining[3] != 0) { // ==0 인 경우는 noop 가 들어온 상황
(= stall)
              MEM_stage(control_signal_MEM, atp);
              pipelining_finish[3] = 1;
           control_signal_EX = control_signal_ID;
           if (pipelining[2] != 0) { // ==0 인 경우는 noop 가 들어온 상황
(= stall)
               EX_stage(control_signal_EX);
              pipelining_finish[2] = 1;
           if (pipelining[1] != 0) { // ==0 인 경우는 noop 가 들어온 상황
(= stall)
               control_signal_ID = ID_stage(ReadStateReg(0), atp);
              pipelining_finish[1] = 1;
           if (inst_end_checker == 0) { // instruction 메모리 끝나면
              IF_stage(TEXT_MEMORY::text_memory_list[inst]-
>ReadData());
              pipelining_finish[0] = 1;
```

위 코드 처럼 pipelining Emulator가 실행되면 각 instruction은 순차적으로 다섯 개의 stage를 거치게 된다. 이 때 stall(bubble, 또는 flush)인 경우에는 pipelining vector에 0이 들어가게 되고, 따라서 해당 pipeline stage를 실행하지 않는다.

```
if (pipelining_finish[0] == 0 && pipelining_finish[1] == 0 &&
pipelining_finish[2] == 0 && pipelining_finish[3] == 0 &&
pipelining_finish[4] == 0)
```

그리고 모든 stage가 비게 된다면 Emulator가 종료된다.

<main function>

main 함수는 input 값을 해석하여, 해석 결과(n, m, d, p, option의 여부, 및 각 option에서의 정보)를 Emulator 함수에 전달하는 역할을 한다.

3. Code idea

assignment에 사용한 몇 가지 코딩 아이디어이다.

1. State Reg의 활용

이번 assignment에서는 총 24개의 state register를 활용하였다. 다음 표는 각 state reg가 어떤 역할

을 하는 지에 대한 설명이 담긴 표이다.(가독성을 위해 stage 별로 정렬하였다. 코드에서는 필요한 stage reg를 필요한 시점에서 만들어 사용했기 때문에 stage 순서대로 정렬이 되어 있지 않다)

Fetching 한 instruction을 Decoding 단계로 진단하기 위해 사용하는 레시스트: DE 간편에서는 의 2016	Fething 한 instruction을 Decoding 단계로 전달하기 위해 사용하는 레지스는다 10 전쟁에서는 이 215m로 Instruction을 전쟁 받아서 대로달한다. Fething 단계의 PC값을 Decoding 단계로 전달하기 위해 사용하는 레지스는다 이 PC값은 but, but target 안년 중에 사용하다. Decoding 단계의 PC값을 Decoding 단계로 전달하기 위해 사용하는 레지스는다 이 PC값은 but, but target 안년 중에 사용하다. Decoding 단계에서 해석한 Decoding 단계로 전달하기 위해 사용하는 레지스는다 이 Instruction 및 연설을 수행한다. Decoding 단계에서 해석한 opecode를 Execution 단계로 전달하기 위해 사용하는 레지스는다 이 Instruction 및 연설을 수행한다. Decoding 단계에서 해석한 opecode Execution 단계로 전달하기 위해 사용하는 레지스는다 이 Instruction 및 연설을 수행한다. Decoding 단계에서 해석한 Instruction 및 연설을 구행한다. Decoding 단계에서 해석한 Instruction 및 연설을 구행하기 위해 사용하는 데시스는 In 및 하면으로 Haband를 제공한다. Instruction 단계에서 및 연구 집급을 Instruction 및 전달 Instruction 및 Instructi	State Reg	Description
### Instruction을 전혀 받아서 미료답합다 Fetching 반계의 PC값을 Decoding 단계로 전달하기 위해 사용하는 레지스트: 이 PC값은 beach to pet parget 안선 등에 사용되다. Decoding 단계의 PC값을 Execution 단계로 전달하기 위해 사용하는 레지스트: 이 PC값은 beach beq target 안선 등에 사용되다. Decoding 단계에서 해석한 petcode를 Execution 단계로 전달하기 위해 사용하는 레지스트: 이 값 바탕으로 Execution 단계에서 해석한 petcode를 Execution 단계로 전달하게 위해 사용하는 레지스트: 이 값 바탕으로 Execution 단계에서 해석한 petcode를 Execution 단계로 전달하기 위해 사용하는 레지스트: 이 값 바탕으로 Execution 단계에서 해석한 petcode 을 구했다다. Decoding 단계에서 해석한 petcode 를 Execution 단계로 전달하기 위해 사용하는 레지스트: 이 값 바탕으로 Execution 단계에서 하다 다를 마당하는 데지스트: 이 값 바탕으로 Execution 단계에서 다른 petcode 이 값 바탕으로 Execution 단계에서 다른 petcode 이 값 바탕으로 Execution 단계에서 가 petcode 이 없 바탕으로 Execution 단계에서 가 petcode 이 없 바탕으로 Execution 단계에서 가 petcode P	FIDNPC FICKING 단계의 PC경을 Decoding 단계로 전략가 위해 사용하는 레지스트 이 PC경을 Decoding 단계로 전략하기 위해 사용하는 레지스트 이 PC경을 Decoding 단계로 전략하기 위해 사용하는 레지스트 이 PC경을 Decoding 단계의 PC경을 Decoding 단계에서 해석한 Opecode를 Execution 단계로 전략하기 위해 사용하는 레지스트 이 경 바탕으로 Execution 단계로 전략하기 위해 사용하는 레지스트 이 경 바탕으로 Execution 단계에서 해석한 Opecode를 Execution 전계에서 해석한 Opecode를 Execution 전계에서 해석한 Opecode를 Execution 전계에서 해석한 pcode를 Execution 전계에 Pcode에	State Reg	
Fetching 단계의 PC값을 Decoding 단계로 전달하기 위해 사용하는 레지스트: 이 PC값을 Mice Mayer 인상 등에 사용되다. Decoding 단계의 PC값을 Execution 단계로 전달하기 위해 사용하는 레지스트: 이 PC값을 Execution 단계로 전달하기 위해 사용하는 레지스트: 이 PC값을 Mice Mayer 인상 등에 사용되다. Decoding 단계에서 해석한 rs에 들어있는 값을 Execution 단계로 전달하기 위해 사용하는 레지스트: 이 값 바탕으로 Execution 단계에서 해석한 rs에 들어있는 값을 Execution 단계에서 해석한 rs에 모습니는 이 값 바탕으로 Execution 단계에서 작 instruction 별 연산을 수행한다. Decoding 단계에서 해석한 대체, target. 등의 값을 Execution 단계에서 작가 with Call 는 이 값 바탕으로 Execution 단계에서 작가 with Call 는 이 값 바탕으로 Execution 단계에서 가능하는 레지스트: 이 값 바탕으로 Execution 단계에서 하수한 대체 는 이 값 바탕으로 Execution 단계에서 해석한 shamile Execution 단계로 건달하기 위해 사용하는 레지스트: 이 값 바탕으로 Execution 단계에서 해석한 Tinstruction 별 연산을 수행한다 Decoding 단계에서 해석한 Will Execution 단계로 전달하기 위해 사용하는 레지스트: 이 값 바탕으로 Hacand를 제고한다. Decoding 단계에서 해석한 Tinstruction 면서로 전달하기 위해 사용하는 레지스트: 이 값 바탕으로 Hacand를 제고한다. Decoding 단계에서 해석한 Till 는 Execution 단계로 전달하기 위해 사용하는 레지스트: 이 값 바탕으로 Hacand를 제고한다. Decoding 단계에서 해석한 Till 로를 Execution 단계로 전달하게 위해 사용하는 레지스트: 이 값 바탕으로 Hacand를 제고한다. EX_MEM.ALU_OUT 위해 사용하는 레지스트: MEM 단계 선생 기원이 모든 MEM 단계 인체 모든 MEM 단계 인체 모든 MEM 단계 전체 모든 MEM 단계 인체 모든 MEM 단계 인체 모든 MEM 단계 인체 모든 MEM 단계 인체 모든 MEM 단계 시용하는 레지스트: MEM 단계 시용하는 레지스트: MEM 단계 시용하는 레지스트: MEM 단계 시용하는 레지스트: MEM 단계 인체 모든 MEM 단계 시용하는 레지스트: MEM MEM 단계 전체 모든 MEM 단계 인체 지용 New Pale Pale Pale Pale Pale Pale Pale Pale	Fiching 단계의 PC값을 Decoding 단계로 전략하기 위해 사용하는 레지스트: 이 PC값을 Execution 단계로 전략하기 위해 사용하는 레지스트: 이 PC값은 Execution 단계로 전략하기 위해 사용하는 레지스트: 이 값 해당으로 Hazand를 제고하다. Decoding 단계에서 해석한 19 번호를 Execution 단계로 전략하기 위해 사용하는 레지스트: 이 값 해당으로 Hazand를 제고하다. Decoding 단계에서 해석한 19 번호를 Execution 단계로 전략하기 위해 사용하는 레지스트: 이 값 해당으로 Hazand를 제고하다. Decoding 단계에서 해석한 19 번호를 Execution 단계로 전략하기 위해 사용하는 레지스트: 이 값 해당으로 Hazand를 제고하다. Execution 단계로 전략하기 위해 사용하는 데지스트: 이 값 해당으로 Hazand를 제고하다. Execution 단계로 전략하기 위해 사용하는 데지스트: MEM 단계 로 전략하기 위해 사용하는 레지스트: MEM 단계 로 전략하기 위해 사용하는 데지스트: MEM 단계 로 전략하기 위해 사용하는 데지스트: MEM 단계 도 전략하기 위해 사용하는 데지스트: MEM 단계 도 전략하기 위해 사용하는 데지스트: MEM 단계 도 전략하기 위해 사용하는 데지스트: MEM 단계 되었다. EXEMEM.EX.NUM 전체 전체 전체 전체 전략하기 위해 사용하는 데지스트: MEM 단계 되었다. EXEMEM.EX.NUM 전체	IF_ID.Instr	
### ID_EX.NPC ### ID_EX.NPC ### ID_EX.NPC ### ID_EX.NPC Decoding 단계의 PC값을 Execution 단계로 전달하기 위해 사용하는 레지스트: 이 PC값을 be beq target 연설 등에 사용된다. Decoding 단계에서 해석한 opecode를 Execution 단계로 전달하기 위해 사용하는 레지스트: 이 ## 해보으로 Execution 단계로 전달하기 위해 사용하는 레지스트: 이 ## 해보으로 Execution 단계로 전달하기 위해 사용하는 레지스트: 이 ## 해보고는 ### ### ### ### ### ### ### ### ### #	### IDNPC ### 생용하는 레지스타: 이 PC 값은 but bet unger 현실 등이 사용된다. ### Decoding 단계의 PC값을 but bet unger 현실 등이 사용된다. #### Decoding 단계의 PC값을 but but and 한편 전원 등이 사용된다. #### Decoding 단계에서 해석한 opecode를 Execution 단계로 전달하기 위해 사용하는 레지스타: 이 값 바탕으로 Execution 단계에서 해석한 opecode를 Execution 단계로 전달하기 위해 사용하는 레지스타. 이 값 바탕으로 Execution 단계에서 해석한 이에 등이었는 값을 Execution 단계에서 해석한 이에 등이었는 값을 Execution 단계에서 해석한 이에 등이었는 값을 Execution 단계에서 해석한 아이에 등이었는 값을 Execution 단계에서 해석한 아이에 들어있는 값을 Execution 단계에서 해석한 아이에 들어있는 값을 Execution 단계에서 해석한 네마. unget 등이 값을 Execution 단계에서 해석한 imm. unget 등이 값을 보는 FM 한데. unget 등이 값을 보는 FM 한데. unget 등이 값을 바탕으로 Execution 단계에서 해석한 imm. unget 등이 값을 바탕으로 Execution 단계에서 해석한 imm. unget 등이 값을 바탕으로 Execution 단계에서 해석한 imm. unget 등이 값을 보는 FM 한데. unget 이 값 바탕으로 Execution 단계에서 해석한 imm. unget 등이 값을 보는 FM 한데. unget 이 값 바탕으로 Execution 단계로 전달하기 위해 사용하는 레지스타. 이 값 바탕으로 Execution 단계로 전략하기 위해 사용하는 레지스바. 이 값 바탕으로 HB 단계에서 이는 제지 산업하기 위해 사용하는 레지스바. 이 값 바탕으로 HB 단계에서 이는 제지 산업하기 위해 사용하는 데지스바. 이 값 바탕으로 HB 단계에서 이는 IM 산업하기 위해 사용하는 데지스바. 이 값 바탕으로 HB 단계에서 이는 IM 산업하기 위해 사용하는 데지스바. MEM 단계 일본하기 위해 사용하는 데지스바. MEM 단계 일본 원칙하기 위해 사용하는 데지스바. MEM 단계 IMM 단계 일본 환경하기 위해 사용하는 데지스바. MEM 단계 IMM EXP IMM 단계 IMM EXP IMM EX		
Decoding 단계의 PC값을 Execution 단계로 전달하기 위해 사용하는 레지스타: 이 PC값을 bne, beq target 안설 보안지 위해 사용하는 레지스타: 이 요를 바탕으로 Execution 단계로 전달하기 위해 사용하는 레지스타: 이 요를 바탕으로 Execution 단계로 전달하기 위해 사용하는 레지스타: 이 요를 바탕으로 Execution 단계에서 해석한 proper 없음을 Execution 단계로 전달하기 위해 사용하는 레지스타: 이 요를 바탕으로 Execution 단계에서 해석한 proper 값음을 Execution 단계에서 해석한 proper 값음을 Execution 단계로 전달하기 위해 사용하는 레지스타: 이 요를 바탕으로 Execution 단계에서 작가 instruction 말 안설을 수행한다	Decoding 단계의 PC 2을 Execution 단계로 전달하기 위해 사용하는 레지스트: 이 R 가장으로 Execution 단계로 전달하기 위해 사용하는 레지스트: 이 2 바탕으로 Execution 단계로 전달하기 위해 사용하는 레지스트: 이 2 바탕으로 Execution 단계에서 해석한 pcode Execution 단계로 전달하기 위해 사용하는 레지스트: 이 2 바탕으로 Execution 단계에서 해석한 pcode Execution 단계로 전달하기 위해 사용하는 레지스트: 이 2 바탕으로 Execution 단계에서 해석한 pcode Execution 단계로 전달하기 위해 사용하는 레지스트: 이 2 바탕으로 Execution 단계로 전달하기 위해 사용하는 레지스트: 이 2 바탕으로 Execution 단계로 전달하기 위해 사용하는 레지스트: 이 2 바탕으로 Execution 단계로 전달하기 위해 사용하는 레지스트: 이 2 바탕으로 Execution 단계로 전달하기 위해 사용하는 레지스트: 이 2 바탕으로 Execution 단계로 전달하기 위해 사용하는 레지스트: 이 2 바탕으로 Execution 단계에서 각 mstruction 별 만산을 수 행한다 Execution 단계에서 각 mstruction 별 만산을 보는 함께 함께 하는 레지스트: 이 2 바탕으로 Execution 단계에서 각 mstruction 별 만산을 수 행한다 Execution 단계에서 각 mstruction 별 만산을 수 행한다 Execution 단계에서 가능하는 레지스트: 이 2 바탕으로 Execution 단계에서 각 mstruction 별 만산을 수 행한다 Execution 단계에서 각 mstruction 별 만산을 수 행한다 Execution 단계에서 각 mstruction 별 만산을 수 행한다 Decoding 단계에서 위적한 mclafe Execution 단계로 전달하기 위해 사용하는 레지스트: 이 2 바탕으로 Execution 단계에서 위적한 mclafe Execution 단계로 전달하기 위해 사용하는 레지스트: 이 2 바탕으로 Execution 단계에서 위적한 mclafe Execution 단계로 전달하기 위해 사용하는 레지스트: 이 2 바탕으로 Execution 단계로 전달하기 위해 사용하는 레지스트: 이 2 바탕으로 Execution 단계로 전달하기 위해 사용하는 레지스트: 이 3 바탕으로 Haard를 체크한다 Decoding 단계에서 해석한 1만 번호로 Execution 단계로 전달하기 위해 사용하는 레지스트: 이 3 바탕으로 Haard를 체크한다 Execution 단계로 전달하기 위해 사용하는 레지스트: 이 3 바탕으로 Haard를 체크한다 Execution 단계로 전달하기 위해 사용하는 레지스트: 이 3 바탕으로 Haard를 체크한다 Execution 단계로 전달하기 위해 사용하는 레지스트: 이 3 바탕으로 Haard를 체크한다 Execution 단계로 전달하기 위해 사용하는 레지스트: 이 3 바탕으로 Haard를 제크한다 Execution 단계로 전달하기 위해 사용하는 레지스트: 이 3 바탕으로 Haard를 제크한다 Execution 단계로 전달하기 위해 사용하는 레지스트: 에 3 바탕으로 Haard를 제크한다 Execution 단계로 전달하기 위해 사용하는 레지스트: MBM 단계 보안하기 위해 사용하는 레지스트: MBM 단계 모델하기 위해 사용하는 레지스트: MBM 단계 모델하기 위해 사용하는 레지스트: MBM 단계 모델하기 위해 사용하는 레지스트: 에 3 바탕으로 단하기 위해 사용하는 레지스트: 에 3 바탕으로 다 전달하기 위해 사용하는 레지스트: 에 3 바탕으로 나타이 전로 전달하기 위해 사용하는 레지스트: 에 3 바탕으로 나타이 전로 전달하기 위해 사용하는 레지스트: 에 3 바탕으로 다 전달하기 위해 사용하는 레지스트: 에 3 바탕으로 다 전달하기 위해 사용하는 레지스트: 에 3 바탕으로 다 시설하기 위해 사용하는 레지스트: 인크로 다		
Decoding 단계의 PC값을 Execution 단계로 전달하기 위해 사용하는 레지스타: 이 PC값은 bne, beg target 안상들이 사용된다. Decoding 단계에서 해석한 opecode를 Execution 단계로 전달하기 위해 사용하는 레지스타: 이 경 바탕으로 Execution 단계에서 해석한 pecode를 Execution 단계에서 해석한 pecode를 Execution 단계에서 해석한 pecode을 들었는 없는 없는 Execution 단계에서 가용하는 레지스타: 이 경 바탕으로 Execution 단계에서 해석한 pecode을 들었는 없는 Execution 단계로 전달하기 위해 사용하는 레지스타: 이 경 바탕으로 Execution 단계에서 해석한 pecode을 이 Decoding 단계에서 해석한 pecode을 하는 마찬에서 한 instruction 별 연산을 수행한다. Decoding 단계에서 해석한 pecode을 Execution 단계에서 해석한 pecodes 를 구해한다. Decoding 단계에서 해석한 pecodes 를 Page PC Instruction 별 연산을 수행한다. Decoding 단계에서 해석한 pecodes Execution 단계로 전달하기 위해 사용하는 레지스타: 이 값 바탕으로 Execution 단계에서 해석한 pecodes Execution 단계로 전달하기 위해 사용하는 레지스타: 이 값 바탕으로 Execution 단계에서 해석한 pecodes Execution 단계에서 pecodes Execution 단계에서 pecodes Execution 단계로 전달하기 위해 사용하는 레지스타: 이 값 바탕으로 Execution 단계로 전달하기 위해 사용하는 레지스타: 이 값 바탕으로 Hazard를 체크한다. Decoding 단계에서 해석한 pecodes Execution 단계로 전달하기 위해 사용하는 레지스타: 이 값 바탕으로 Hazard를 체크한다. Execution 단계에서 역한 pecodes Execution 단계로 전달하기 위해 사용하는 레지스타: Pecodes Execution 단계로 전달하게 위해 사용하는 레지스타: Pecodes Execution 단계로 전달하게 위해 사용하는 레지스타: Pecodes Execution 단계로 Perovarding Perovard	Decoding 단계의 PC값을 Execution 단계로 전략하기 위해 사용하는 레지스트: 이 PC값을 Onc. Dea target 인상 등에 사용되다. Decoding 단계에서 해석한 opecode를 Execution 단계로 전략하기 위해 사용하는 레지스트: 이 값 바탕으로 Execution 단계로 전략하기 위해 사용하는 레지스트: 이 값 바탕으로 Execution 단계로 전략하기 위해 사용하는 레지스트: 이 값 바탕으로 Execution 단계로 전략하기 위해 사용하는 레지스트: 이 값 바탕으로 Execution 단계로 전략하기 위해 사용하는 레지스트: 이 값 바탕으로 Execution 단계로 전략하기 위해 사용하는 레지스트: 이 값 바탕으로 Execution 단계에서 강 instruction 별 인상 을 수했한다. Decoding 단계에서 해석한 roll 들이있는 값을 Execution 단계로 전략하기 위해 사용하는 레지스트: 이 값 바탕으로 Execution 단계에서 강 instruction 별 인상 을 수했한다. Decoding 단계에서 해석한 imm, target, 등의 값을 Execution 단계로 전략하기 위해 사용하는 레지스트: 이 값 바탕으로 Execution 단계로 전략하기 위해 사용하는 레지스트: 이 값 바탕으로 Execution 단계로 전략하기 위해 사용하는 레지스트: 이 값 바탕으로 Execution 단계로 전략하기 위해 사용하는 레지스트: 이 값 바탕으로 Execution 단계로 전략하기 위해 사용하는 레지스트: 이 값 바탕으로 Execution 단계로 전략하기 위해 사용하는 레지스트: 이 값 바탕으로 Execution 단계로 전략하기 위해 사용하는 레지스트: 이 값 바탕으로 Execution 단계로 전략하기 위해 사용하는 레지스트: 이 값 바탕으로 Execution 단계로 전략하기 위해 사용하는 레지스트: 이 값 바탕으로 Execution 단계로 전략하기 위해 사용하는 레지스트: 이 값 바탕으로 Execution 단계로 전략하기 위해 사용하는 레지스트: 이 값 바탕으로 Execution 단계로 전략하기 위해 사용하는 레지스트: 이 값 바탕으로 Hanad를 제고한다. Decoding 단계에서 해석한 바탕 Execution 단계로 전략하기 위해 사용하는 레지스트: 이 값 바탕으로 Hanad를 제고한다. Decoding 단계에서 해석한 바탕을 Execution 단계로 전략하기 위해 사용하는 레지스트: 이 값 바탕으로 Hanad를 제고한다. Execution 단계에서 대약은 Targete MEM 단계로 전략하기 위해 사용하는 레지스트: 이 값 바탕으로 Hanad를 제고한다. Execution 단계에서 만든 Targete MEM 단계로 전략하기 위해 사용하는 레지스트: MEM 단계 서용하는 레지스트: MEM 단계 서용하는 레지스트: MEM 단계 사용하는 데지스트: MEM 단계 사용하는 레지스트: MEM 단계 사용하는 레지스트: MEM 단계 사용하는 데지스트: MEM 단계 사용하는 레지스트: MEM 단계 사용하는 데지스트: MEM 단계 사용하는 데지스트: MEM 단계 사용하는 데지스트: MEM 단계 사용하는 데지스트: MEM 단계 시용하는 데지스트: MEM 단계 시용하는 레지스트: MEM 단계 시용하는 에지스트: MEM 단계 시용하는 데지스트: MEM 단계 시용하는 데지스트: MEM 단계 사용하는 데지스트: MEM 단계 사용하	IF_ID.NPC	
## A # S 하는 레지스타: 이 P C # C # Dee to target 인산	ID_EX.NPC 의 사용하는 레지스타: 이 PC 많은 bee, beq target 영상 등에 사용된다. Decoding 단계에서 해석한 opecode를 Execution 단계로 전달하기 위해 사용하는 레지스타: 이 값 바탕으로 Execution 단계에서 해석한 opecode를 Execution 단계를 전달하기 위해 사용하는 레지스타 이 값 바탕으로 Execution 단계에서 해석한 rol 들어있는 값을 Execution 단계로 진달하기 위해 사용하는 레지스타 이 값 바탕으로 Execution 단계에서 해석한 rol 들어있는 값을 Execution 단계로 진달하기 위해 사용하는 레지스타 이 값 바탕으로 Execution 단계에서 해석한 rol 들어있는 값을 Execution 단계에서 해석한 rol 들어 없는 이 값 바탕으로 Execution 단계에서 해석한 rol 로 인한 글 수행한다 Decoding 단계에서 해석한 min. target 등이 값을 Execution 단계에서 가 instruction 앱 인한 글 수행한다 Decoding 단계에서 해석한 A instruction 앱 인한 글 수행한다 Decoding 단계에서 해석한 A instruction 앱 인한 글 수행한다 Decoding 단계에서 해석한 A instruction 앱 인한을 수행한다 Decoding 단계에서 해석한 A instruction 앱 인한을 수행한다 Decoding 단계에서 해석한 Roution Field을 Execution 단계로 진달하기 위해 사용하는 레지스타 이 값 바탕으로 Execution 단계에서 대석한 Note Recution 단계로 진달하기 위해 사용하는 레지스타 이 값 바탕으로 Execution 단계에서 대석한 Note Execution 단계로 진달하기 위해 사용하는 레지스타 이 값 바탕으로 Execution 단계로 진달하기 위해 사용하는 레지스타 이 값 바탕으로 Hacard를 체크한다. Decoding 단계에서 해석한 Roution Field을 Execution 단계로 진달하기 위해 사용하는 레지스타 이 값 바탕으로 Hacard를 제고한다. Decoding 단계에서 해석한 Roution Field을 Execution 단계로 진달하기 위해 사용하는 레지스타 이 값 바탕으로 Hacard를 제고한다. Execution 단계에서 해석한 Roution Field을 Execution 단계로 진달하기 위해 사용하는 레지스타 이 값 바탕으로 Hacard를 제고한다. Execution 단계에서 의한 Note Execution 단계로 진달하기 위해 사용하는 레지스타 이 없 바탕으로 Hacard를 제고한다. Execution 단계에서 의한 Roution Field을 Execution 단계로 진달하게 위해 사용하는 레지스타 에 Roution Pair 이 있으로 분기한다. (Pd 분기) Execution 단계에서 위한 Roution Field을 Execution 단계를 Rowarding인 IMM을 다시 MEM 단계 외 사용하는 레지스타 MEM 단계 외 산용하는 레지스타 MEM 단계 외 안 환경을 Execution 단계로 전 안녕한다 위해 사용하는 레지스타 MEM 단계 외 안녕하고 위해 사용하는 레지스타 MEM 단계 외 산용하는 레지스타 MEM 단계 외 산용하는 레지스타 에 있는 한다 MEM 단계 외 한계와 Pa		
Decoding 단계에서 해석한 opecode를 Execution 단계로 전달하기 위해 사용하는 레지스트: 이 값 바탕으로 Execution 단계에서 해석한 cm를 들어보고 값을 함께 당으로 대한 바에서 각 instruction 낼 인산을 수행한다. Decoding 단계에서 해석한 cm를 들어있는 값을 Execution 단계로 진달하기 위해 사용하는 레지스트: 이 값 바탕으로 Execution 단계에서 해석한 cm를 되지스는: 이 값 바탕으로 Execution 단계에서 해석한 cm를 Execution 단계로 진달하기 위해 사용하는 레지스는: 이 값 바탕으로 Execution 단계에서 해석한 cm를 Execution 단계에서 해석한 cm를 되지스는: 이 값 바탕으로 Execution 단계에서 해석한 cm를 되지스는: 이 값 바탕으로 Execution 단계에서 해석한 function field 다양에서 각 instruction 낼 인산을 수행한다 Decoding 단계에서 해석한 function field 다양에서 각 instruction 낼 인산을 수행한다 Decoding 단계에서 해석한 function field 다양에서 가능하는 레지스는: 이 값 바탕으로 Execution 단계에서 해석한 function field 다양에서 가능하는 레지스는: 이 값 바탕으로 Execution 단계에서 해석한 function field 다양에서 가능하는 레지스는: 이 값 바탕으로 Execution 단계에서 해석한 function field 다양에서 가능하는 레지스는: 이 값 바탕으로 Hazand를 체크한다. Decoding 단계에서 해석한 cm 번호를 Execution 단계로 전달하기 위해 사용하는 레지스는: 이 값 바탕으로 Hazand를 체크한다. Decoding 단계에서 해석한 cm 번호를 Execution 단계로 전달하기 위해 사용하는 레지스는: 이 값 바탕으로 Hazand를 체크한다. Decoding 단계에서 해석한 cm 번호를 Execution 단계로 전달하기 위해 사용하는 레지스는: 이 값 바탕으로 Hazand를 체크한다. Decoding 단계에서 해석한 cm 번호로 Execution 단계로 전달하기 위해 사용하는 레지스는: 이 값 바탕으로 Hazand를 체크한다. Decoding 단계에서 해석한 cm 번호로 Execution 단계로 전달하기 위해 사용하는 레지스는: 이 값 바탕으로 Hazand를 제크한다. Decoding 단계에서 해석한 cm 번호로 Execution 단계로 전달하기 위해 사용하는 레지스는: 이 값 바탕으로 Hazand를 제크한다. Decoding 단계에서 현은 Target을 MEM 단계 본 전략하기 위해 사용하는 레지스는: 사용 전략으로 바탕으로 전략하게 다양한다. Decoding 단계에서 한 연합 한 대전을 보였다. Decoding 단계에서 한 연합 한 대전을 보였다. Decoding 단계에서 해석한 cm 만호를 Execution 단계로 전략하기 위해 사용하는 레지스는: 이 값 바탕으로 Execution 단계에서 얻은 Target을 MEM 단계 본 전략하기 위해 사용하는 레지스는: 이 값 바탕으로 Execution 단계 사용한는 레지스는: 이 값 바탕으로 Execution 단계 사용한는 레지스는: 이 값 바탕으로 Execution 단계 사용한다. Decoding 만화를 보고 반호로 Execution 단계 사용한다. Deco	DECODING 단계에서 웨어한 OPCOCK를 Execution 단계로 전단하기 위해 사용하는 레지스타: 이 집 바탕으로 Execution 단계에서 해서한 naturation 뿐 연산을 수행한다. DECODING UT에서 해석한 대통	ID EY NDC	
Decoding 단계에서 해석한 opecode를 Execution 단계로 전달하기 위해 사용하는 레지스타: 이 값 바탕으로 Execution 단계에서 해석한 roll 들어있는 값을 Execution 단계에서 해석한 roll 들어 가능한으로 Execution 단계에서 가를 가려보다는 이 값 바탕으로 Execution 단계에서 가를 가려보다는 이 값 바탕으로 Execution 단계에서 가를 마라보다는 이 값 바탕으로 Execution 단계에서 가를 자라보다는 이 값 바탕으로 Execution 단계에서 하려한 roll 로 프로 Execution 단계에서 해석한 roll 로 Execution 단계에서 해석한 roll 로 프로 Execution 단계에서 해석한 roll 로 를 들어한다 Decoding 단계에서 해석한 roll 로 를 Execution 단계에서 해석한 roll 로 한 다음이 함께 related Execution 단계에서 해석한 roll 로 한 다음이 함께 roll 라를 모으면 모든 Execution 단계에서 해석한 roll 로 한 다음이 모든 모든 Execution 단계에서 해석한 roll 를 Execution 단계로 전 달하기 위해 사용하는 레지스타 이 값 바탕으로 Execution 단계에서 해석한 roll를 Execution 단계로 전 달하기 위해 사용하는 레지스타 이 값 바탕으로 Horard Part Part Part Part Part Part Part Part	Decoding 단계에서 폐석한 opecode를 Execution 단계로 전달하기 위해 사용하는 레지스타: 이 값 바탕으로 Execution 단계에서 기 mstruction 별 현실을 수행한다. Decoding 단계에서 예석한 rol 들어있는 값을 Execution 단계로 전달하기 위해 사용하는 레지스타: 이 값 바탕으로 Execution 단계에서 각 instruction 별 현실을 수행한다. Decoding 단계에서 해석한 rol 들어있는 값을 Execution 단계로 전달하기 위해 사용하는 레지스타: 이 값 바탕으로 Execution 단계에서 각 instruction 별 연설을 수행한다. Decoding 단계에서 해석한 roll 들어있는 값을 Execution 단계에서 각 instruction 별 연설을 수행한다. Decoding 단계에서 해석한 imm, target 등의 값을 Execution 단계에서 각 instruction 별 연설을 수행한다. Decoding 단계에서 해석한 roll 보증을 Execution 단계로 전달하기 위해 사용하는 레지스타: 이 값 바탕으로 Execution 단계에서 각 instruction 별 연설을 수행한다. Decoding 단계에서 해석한 shamif를 Execution 단계로 전달하기 위해 사용하는 레지스타: 이 값 바탕으로 Execution 단계에서 기 mstruction 별 연설을 수행한다. Decoding 단계에서 해석한 shamif를 Execution 단계로 전달하기 위해 사용하는 레지스타: 이 값 바탕으로 Execution 단계에서 가능하는 라지스타: 이 값 바탕으로 Execution 단계에서 가능하는 레지스타: 이 값 바탕으로 Execution 단계에서 기가 instruction 별 연설을 수행한다. Decoding 단계에서 해석한 shamif를 Execution 단계로 전달하기 위해 사용하는 레지스타: 이 값 바탕으로 Execution 단계에서 기가 instruction 별 연설을 수행한다. Decoding 단계에서 해석한 Finds of instruction 별 연설을 수행한다. Decoding 단계에서 해석한 Finds of instruction 별 연설을 수행한다. Decoding 단계에서 해석한 Finds of instruction 별 연설을 수행한다. Decoding 단계에서 해석한 Instruction 별 연설을 수행한다. Decoding 단계에서 해석한 Finds of instruction 를 연설을 수행한다. Decoding 단계에서 해석한 Finds of instruction 를 연설을 수행한다. Decoding 단계에서 해석한 Finds of instruction 를 연설을 수행한다. Decoding 단계에서 해석한 Finds of instruction 를 연설을 수행한다. Decoding 단계에서 해석한 Finds of instruction 단계로 전달하기 위해 사용하는 레지스타: 이 값 바탕으로 Mscall 를 제고하는 이 값 바탕으로 Mscall 를 제고하는 대한데 단계로 전달하기 위해 사용하는 레지스타: MscM 단계 에서 사용하는 레지스타: MscM 단계 에서 사용하는 레지스타: MscM 단계 어제 어제 모든 Forwarding된 Find 들어있는 집은 다시 Mem 단계로 전달하기 위해 사용하는 레지스타: MscM 단계 에서 이 사용하는 레지스타: MscM 단계 이 사용하는 데지스타: MscM 단계 에서 이 사용하는 레지스타: MscM 단계 에서 이 사용하는 레지스타: MscM 단계 에서 이 사용하는 레지스타: MscM 단계 에서 이 사용하는 데지스타: MscM 단계 에서 MscM 단계 에서 MscM 단계 에서 MscM 단계 에서 MscM 단	ID_EA.IVI C	
전변하기 위해 사용하는 레지스트: 이 값 바란으로 Execution 단계에서 라이 제시한 등에 들어있는 값을 Execution 단계에서 해석한 대에 들어있는 값을 Execution 단계에서 해석한 배매, target 등의 값을 Execution 단계에서 해석한 배매를 Execution 단계로 전달하기 위해 사용하는 레지스트: 이 값 바탕으로 Execution 단계에서 해석한 function field 대로 전달하기 위해 사용하는 레지스트: 이 값 바탕으로 Execution 단계에서 해석한 WB를 Execution 단계에서 해석한 WB를 Execution 단계에서 해석한 B 전후를 Execution 단계로 전달하기 위해 사용하는 레지스트: 이 값 바탕으로 Hazand를 제고한다. Decoding 단계에서 해석한 WB를 Execution 단계로 전달하기 위해 사용하는 레지스트: 이 값 바탕으로 Hazand를 제고한다. Decoding 단계에서 해석한 B 전후를 Execution 단계로 전달하기 위해 사용하는 레지스트: 이 값 바탕으로 Hazand를 제고한다. Decoding 단계에서 해석한 대 한후로 Execution 단계로 전달하기 위해 사용하는 레지스트: 이 값 바탕으로 Hazand를 제고한다. Execution 단계에서 연은 Target을 MEM 단계로 전달하기 위해 사용하는 레지스트: 이 값 바탕으로 MEM 단계 연사 살 수행하게 위해 사용하는 레지스트: 세계 전략 한 수 함께 가게 함께 사용하는 레지스트: 세계 전에 가게 되었으로 본거한다 단계로 전달하지 위해 사용한다 레지스트: 세계 전에 가용된다. Execution 단계로 forwarding된 IMM를 다시 MEM 단계 시용 및 전환을 기해 사용하는 레지스트: 세계 사용된다. Execution 단계로 forwarding된 IMM를 다시 MEM 단계 시용 및 전환을 기해 사용하는 레지스트: 세계 사용된다. Execution 단계로 forwarding된 IMM를 다시 MEM 단계 보게 보게 보게 보게 하는 레지스트: 세계 사용된다. Execution 단계로 forwarding된 IMM를 다시 MEM 단계 모든 전환을 가게 위해 사용하는 레지스트: 세계 사용된다. Execution 단계로 전함하기 위해 사용한다. 레지스트: 세계 MEM 단계 보게 함께 사용하는 레지스트: 세계 MEM 단계 보게 보게 보게 보게 보게 보게 보게 함께 보게 NEM 트레지트: MEM EXECUTION 단계로 전함하기 위해 사용한다. 레지스트: 세계 MEM 단계 보게 NEM 단계에서 MEM 단계 보게 NEM EXECUTION THE M	D_EX.Opcode 전날하기 위해 사용하는 레지스타. 이 값 바탕으로 Execution 단계에서 각 instruction 별 연산을 수행한다. Decoding 단계에서 해석한 rs에 들어있는 값을 Execution 단계로 전달하기 위해 사용하는 레지스타. 이 값 바탕으로 Execution 단계에서 해석한 rs에 들어있는 값을 Pedigity Decoding 단계에서 해석한 rs에 들어있는 값을 Execution 단계에서 해석한 rsm, target, 등의 값을 Execution 단계에서 제시한 rsm, target, 등의 값을 Execution 단계에서 제시한 rsm, target, 등의 값을 Execution 단계에서 해석한 rsm, target, 등의 값을 Execution 단계에서 해석한 rsm, target, 등의 값을 Execution 단계에서 해석한 rsm, target, 등의 값을 Execution 단계에서 제시한 rsm, target, 등의 값을 Hsp으로 Execution 단계에서 제시한 rsm, target, 등의 대상을 A target, Execution 단계에서 연안을 제시한 다개한 전문에 전체 사용하는 레시스타, 이 값 마탕으로 Hazard를 체크한다, (Target) 등의 대상을 A target, Execution 단계에서 연안을 A target, Execution 단계에서 모든 전환하기 위해 사용하는 레시스타, (MEM 단계 에서 모든 전환하기 위해 사용하는 레시스타, (MEM 단계 위해 사용하는 레시스타, of target, of target, of target, Execution 단계로 forwarding된 rs 번호를 다시 MEM 단계 어내 모든 전환하기 위해 사용하는 레시스타, of target, of		
ID_EX.rs Decoding 단계에서 해석한 rs에 들어있는 값을 Execution 단계로 전달하기 위해 사용하는 레지스타: 이 값 바탕으로 Execution 단계에서 각 instruction 별 연산을 수행한다. Decoding 단계에서 해석한 ri에 들어있는 값을 Execution 단계에서 각 instruction 별 연산을 수행한다. Decoding 단계에서 해석한 ri에 들어있는 값을 Execution 단계에서 각 instruction 별 연산을 수행한다. Decoding 단계에서 해석한 ri에 만기로 전달하기 위해 사용하는 레지스타: 이 값 바탕으로 Execution 단계에서 각 instruction 별 연산을 수행한다. Decoding 단계에서 해석한 rid 민호를 Execution 단계로 전달하기 위해 사용하는 레지스타 대로 전달하기 위해 사용하는 레지스타 이 값 바탕으로 Execution 단계에서 각 instruction 별 연산을 수행한다. Decoding 단계에서 해석한 shamt를 Execution 단계로 전달하기 위해 사용하는 레지스타 이 값 바탕으로 Execution 단계에서 각 instruction 별 연산을 수행한다. Decoding 단계에서 해석한 rid 만기로 이 값 바탕으로 Execution 단계에서 해석한 rid 만기로 한 단계로 전달하기 위해 사용하는 레지스타 이 값 바탕으로 Execution 단계로 전달하기 위해 사용하는 레지스타 이 값 바탕으로 Execution 단계로 전달하기 위해 사용하는 레지스타 이 값 바탕으로 Execution 단계로 전달하기 위해 사용하는 레지스타 이 값 바탕으로 Execution 단계로 전달하기 위해 사용하는 레지스타 이 값 바탕으로 Execution 단계로 전달하기 위해 사용하는 레지스타 이 값 바탕으로 Hazard를 체크한다. Decoding 단계에서 해석한 rid 및 전달 Execution 단계로 전달하기 위해 사용하는 레지스타 이 값 바탕으로 Hazard를 체크한다. Decoding 단계에서 해석한 대신을 Execution 단계로 전달하기 위해 사용하는 레지스타 이 값 바탕으로 Maxard를 체크한다. Decoding 단계에서 해석한 rid 및 전달하기 위해 사용하는 레지스타 이 값 바탕으로 Hazard를 제크한다. Ex_MEM.BR_TARGET Ex_MEM.BR_TARGET EX_MEM.IMM Decoding 단계에 등에 사용하는 레지스타 MEM 단계 모게 있는 없을 다시 MEM 단계 모게 있는 없을 다시 MEM 단계 에서 되어 되어 되어 되었다. MEM 단계 모레 이라 MEM 단계 모렌 이라는 MEM 단계 에서 이라 MEM 단계 모렌 이라는 MEM 단계 에서 이라 MEM 단계 모렌 이라는 MEM 단계 에서 이라는 MEM 단계 에서 이라는 MEM 단계 에서 이라는 MEM 단계 에서 이라는 대신하는 MEM 단계 에서 이라는 MEM 단계 에서 이라는 데시스타 MEM 단계 에서 이라는 데시스타 MEM 단계 에서 이라는 데시스타 MEM 단계 대신하는 MEM 단계 에서 이라는 데시스타 MEM 단계 대신하는 데시스타 MEM 단계 에서 이라는 데시스타 MEM 단계 에서 이라는 MEM 단계 대신하는 MEM 단계 대신하는 MEM 단계 에서 이라는 데시스타 MEM 단계 대신하는 MEM 단계 에서 이라는 MEM 단계 에서 이라는 데시스타 MEM 단계 모든 전달하기 위해 사용하는 레시스타 MEM 단계 모든 제소나 MEM 단계 대신하는 MEM 단계 대신하는 MEM 단계 대신하는 MEM 단계 에서 이라는 MEM 단계 에서 이라는 MEM 단계 대신하는	Decoding 단계에서 해석한 roll 들어있는 값을 Execution 단계로 전달하기 위해 사용하는 레지스타 이 값 바탕으로 Execution 단계에서 가 instruction 앱 연산을 수행한다. Decoding 단계에서 해석한 roll 들어있는 값을 Execution 단계에서 각 instruction 앱 연산을 수행한다. Decoding 단계에서 해석한 imm, target 등의 값을 Execution 단계에서 해석한 imm, target 등의 값을 Execution 단계에서 해석한 imm, target 등의 값을 Execution 단계에서 해석한 imm, target 등의 값을 가능한으로 Execution 단계에서 작 instruction 앱 연산을 수행한다. Decoding 단계에서 해석한 imm, target 등의 값을 보는 Fexeution 단계에서 해석한 imm, target 등의 값을 가능한으로 Execution 단계에서 해석한 imm, target 등의 값을 보는 Fexeution 단계에서 해석한 imm, target 등의 값을 Execution 단계에서 해석한 imm, target 이 값 바탕으로 Harand를 제고한다. Decoding 단계에서 해석한 imm, target 이 값 바탕으로 Harand를 제고한다. Decoding 단계에서 해석한 imm, target 이 값 바탕으로 Harand를 제고한다. Decoding 단계에서 해석한 imm, target 이 값 바탕으로 Harand를 제고한다. Execution 단계에서 얻은 캠프를 MEM 단계 로만하기 위해 사용하는 레지스는 이 값 바탕으로 Harand를 제고한다. Execution 단계에서 얻은 캠프를 MEM 단계 로만하기 위해 사용하는 레지스는 에서 MEM 단계 로만하기 위해 사용하는 레지스는 MEM 단계 에서 ssx sb 선산을 수행하기 위해 사용하는 레지스는 MEM 단계 에서 ssx sb 선산을 수행하기 위해 사용하는 레지스는 MEM 단계 당한 선산을 수행하기 위해 사용하는 레지스는 MEM 단계 당한 선산을 수행하기 위해 사용하는 레지스는 MEM 단계 당한 전체로 한다가 되었다 한다는 제고는 IMEM 단계 전원하기 위해 사용하는 레지스는 에서 MEM 단계 당한 전환이 기위에 사용하는 레지스는 이 값 바탕으로 Harand를 제공한다. Execution 단계로 forwarding인 ImM을 the MEM 단계 에서 ssx sb 선산을 수행하기 위해 사용하는 레지스는 에서 MEM 단계 당한 선산을 수행하기 위해 사용하는 레지스는 이 값 바탕으로 Harand를 사용한다. 데지스는 이 값 바탕으로 Harand를 사용한다. Imm, target MEM 단계 에서 ssx sb 선산을 수행하기 위해 사용하는 레지스는 이 값 바탕으로 Harand를 사용한다. Imm, target MEM 단계 에서 ssx sb 선산을 수행하기 위해 사용하는 레지스는 이 값 바탕으로 Harand를 사용한다. Imm, target MEM 단계 에서 ssx sb 선산을 수행하기 위해 사용하는 레지스는 이 값 바탕으로 Haran	ID EX.Opcode	
Execution 단계로 전달하기 위해 사용하는 레지스타: 이 값 바탕으로 Execution 단계에서 각 instruction 별 연산 을 수행한다 Decoding 단계에서 해석한 roll 들어있는 값을 Execution 단계에서 각 instruction 별 연산 을 수행한다 Decoding 단계에서 해석한 roll 들어있는 값을 Execution 단계에서 각 instruction 별 연산 을 수행한다 Decoding 단계에서 해석한 roll roll 자료 전달하기 위해 사용하는 레지스타: 이 값 바탕으로 Execution 단계에서 각 instruction 별 연산 을 수행한다 Decoding 단계에서 해석한 roll 대표 전달하기 위해 사용하는 레지스타: 이 값 바탕으로 Execution 단계에서 각 instruction 별 연산 을 수행한다 Decoding 단계에서 해석한 대로 대로대로에 별 연산을 수행한다 전계에서 해석한 shamil를 Execution 단계로 전달하기 위해 사용하는 레지스타: 이 값 바탕으로 Execution 단계에서 각 instruction 를 연산을 수행한다 Decoding 단계에서 해석한 function field를 Execution 단계로 전달하기 위해 사용하는 레지스타: 이 값 바탕으로 Execution 단계에서 각 instruction 별 연산을 수행한다 Decoding 단계에서 해석한 대를 제고하는 이 값 바탕으로 Execution 단계로 전달하기 위해 사용하는 레지스타: 이 값 바탕으로 Execution 단계로 전달하기 위해 사용하는 레지스타: 이 값 바탕으로 Hazard를 제고한다. Decoding 단계에서 해석한 Top	Execution 단계로 전달하기 위해 사용하는 레지스타: 이 값 바탕으로 Execution 단계에서 작 instruction 별 연산 을 수행한다. Decoding 단계에서 해석한 마이 들어있는 젊을 Execution 단계에서 해석한 마이 들어있는 집을 가장한다. Decoding 단계에서 해석한 instruction 별 연산 을 수행한다. Decoding 단계에서 해석한 ind 번호를 Execution 단계로 전달하기 위해 사용하는 레지스타: 이 값 바탕으로 Execution 단계에서 각 instruction 별 연산 을 수행한다. Decoding 단계에서 해석한 대체스타: 이 값 바탕으로 Execution 단계로 전달하기 위해 사용하는 레지스타: 이 값 바탕으로 Execution 단계로 제식으로 이 값 바탕으로 Execution 단계로 제식으로 이 값 바탕으로 Execution 단계에서 각 instruction 별 연산을 수행한다. Decoding 단계에서 해석한 instruction 별 연산을 수행한다. 다기로 전달하기 위해 사용하는 레지스타: 이 값 바탕으로 Execution 단계로 전달하기 위해 사용하는 레지스타 이 값 바탕으로 Hacard를 체크한다. Decoding 단계에서 해석한 in 번호를 Execution 단계로 전달하기 위해 사용하는 레지스타 이 값 바탕으로 Hacard를 체크한다. Decoding 단계에서 해석한 in 번호를 Execution 단계로 전달하기 위해 사용하는 레지스타 이 값 바탕으로 Hacard를 제크한다. Execution 단계에서 얻은 결과를 MEM 단계 연산을 수용 등 위해 위해 사용하는 레지스타 MEM 단계 연수를 가위해 사용하는 레지스타 MEM 단계 이 사용되다. Execution 단계에서 얻은 집과를 함께 단계로 전달하기 위해 사용하는 레지스타 MEM 단계 이 사용되다. Execution 단계로 forwarding인 In M를 만시로 MEM 단계 있 No. 와 안성을 수용하기 위해 사용되다. Execution 단계로 forwarding인 In M를 만시로 다시로 Hacard를 제크한다. Execution 단계로 forwarding인 In M를 인치는 In MEM 단계 안 No. 와 안성을 수용하기 위해 사용된다. Execution 단계로 forwarding인 In M를 만시로 Execution 단계로 forwarding인 In M를 만시로 Hacard를 제크한다. Execution 단계로 전달하기 위해 사용하는 레시스타 MEM 단계 안 No. 와 안성을 수용하기 위해 사용한다. Execution 단계로 forwarding인 In M를 만시로 Hacard를 제크한다. Execution 단계 로 forwarding인 In M를 만시로 Hacard를 제크한다. Execution 단계 모든 forwarding인 In M를 만시로 Hacard를 제크한다. Execution 단계 보다 MEM 단계 안 No. 와 안성을 수행하기 위해 사용하는 데시스타 MEM 단계 에서 Pad 전 No. 와 안성을 수행하기 위해 사용하는 데시스타 MEM 단계 No. 와 안성을 수행하기 위해 사용하는 데시스타 MEM 단계 No. 와 안성을 수		Execution 단계에서 각 instruction 별 연산을 수행한다.
### ### ### ### ### ### ### ### ### ##	### D_EX.rs BD_EX.rt		•
ID_EX.rt ID_EX.rt Decoding 단계에서 해석한 r/에 들어있는 값을 Execution 단계로 전달하기 위해 사용하는 레지스터: 이 값 바탕으로 Execution 단계에서 해석한 r/에 대표 전달하기 위해 사용하는 레지스터: 이 값 바탕으로 Execution 단계에서 해석한 r/에 대표	Decoding 단계에서 해석한 roll 들어있는 값을 보ceution 단계로 전달하기 위해 사용하는 레지스타: 이 값 바탕으로 Execution 단계로 전달하기 위해 사용하는 레지스타 이 값 바탕으로 Execution 단계로 전달하기 위해 사용하는 레지스타 이 값 바탕으로 Execution 단계로 전달하기 위해 사용하는 레지스타 이 값 바탕으로 Execution 단계로 전달하기 위해 사용하는 레지스타 이 값 바탕으로 Execution 단계로 전달하기 위해 사용하는 레지스타 이 값 바탕으로 Execution 단계로 전달하기 위해 사용하는 레지스타 이 값 바탕으로 Execution 단계로 전달하기 위해 사용하는 레지스타 이 값 바탕으로 Execution 단계로 전달하기 위해 사용하는 레지스타 이 값 바탕으로 Execution 단계로 전달하기 위해 사용하는 레지스타 이 값 바탕으로 Execution 단계로 전달하기 위해 사용하는 레지스타 이 값 바탕으로 Execution 단계로 전달하기 위해 사용하는 레지스타 이 값 바탕으로 Execution 단계로 전달하기 위해 사용하는 레지스타 이 값 바탕으로 Execution 단계로 전달하기 위해 사용하는 레지스타 이 값 바탕으로 Execution 단계로 전달하기 위해 사용하는 레지스타 이 값 바탕으로 Execution 단계로 전달하기 위해 사용하는 레지스타 이 값 바탕으로 Execution 단계로 전달하기 위해 사용하는 레지스타 이 값 바탕으로 Execution 단계로 전달하기 위해 사용하는 레지스타 이 값 바탕으로 Hazard를 체크한다 Decoding 단계에서 해석한 ro 번호를 Execution 단계로 전달하기 위해 사용하는 레지스타 이 값 바탕으로 Hazard를 제크한다 Perioding 단계에서 해석한 ro 번호를 Execution 단계로 전달하기 위해 사용하는 레지스타 이 값 바탕으로 Hazard를 제크한다 Perioding 단계에서 해석한 ro 번호를 Execution 단계로 전달하기 위해 사용하는 레지스타 이 값 바탕으로 Hazard를 제크한다 Perioding 단계에서 열 길 캠탕으로 MEM 단계 연 선을 수행하게 되어 사용하다 대체로 전달하기 위해 사용하는 레지스타 에를 전계 연 선을 수행하게 되어 사용하다 대체로 전달하기 위해 사용하는 레지스타 MEM 단계 에서 SNS 한 전실을 구행하기 위해 사용하는 레지스타 MEM 단계 에서 SNS 한 전실을 구행하기 위해 사용하는 레지스타 MEM 단계 에서 SNS 한 전실을 구행하기 위해 사용하는 레지스타 MEM 단계 에서 SNS 한 전실을 구행하기 위해 사용하는 레지스타 MEM 단계 에서 SNS 한 전실을 구행하기 위해 사용하는 레지스타 MEM 단계 에서 SNS 한 전실을 구행하기 위해 사용하는 레지스타 MEM 단계 에서 SNS 한 전실을 구행하기 위해 사용하는 레지스타 MEM 단계 에서 SNS 한 전실을 구행하기 위해 사용하는 레지스타 MEM 단계 에서 SNS 한 전실을 구행하기 위해 사용하는 레지스타 MEM 단계 에서 SNS 한 전실을 구행하기 위해 사용하는 레지스타 MEM 단계 에서 SNS 한 전실을 구행하기 위해 사용하는 레지스타 MEM 단계 에서 SNS 한 전실을 구행하기 위해 사용하는 레지스타 이 값 바탕으로 Execution 단계로 전상을 구행하기 위해 사용하는 레지스타 MEM 단계 에서 SNS 한 전실을 구행하기 위해 사용하는 레지스타 MEM 단계 에서 SNS 한 전실을 구행하기 위해 사용하는 레지스타 에는 Fexicution 단계로 전상을 구행하기 위해 사용하는 레지스타 이 값 바탕으로 Execution 단계로 전상을 구행하기 위해 사용하는 레지스타 IV MEM 단계 SNS 한 전실을 구행하기 위해 사용하는 레지스타 IV MEM 단계 SNS 한 전실을 구행하기 위해 사용하는 레지스타 IV MEM 단계 SNS 한 전실을 구행하기 위해 사용하는 레지스타 IV MEM 단계 SNS 한 전실을 구행하기 위해 사용하는 레지스타 IV MEM 단계 SNS 한 전실을 구행하기 위해 SNS 한 전체로 MEM Them Them Them Them Them Them Them Them	ID EX.rs	
Decoding 단계에서 해석한 ro에 들어있는 값을 Execution 단계로 전달하기 위해 사용하는 레지스타: 이 값 바탕으로 Execution 단계에서 해석한 ro에 들어있는 값을 보는 모든 전략에서 해석한 row 대로 제품 대로 대로 제품 대로 대로 제품 대로 대로 제품 대로	ID_EX.rt Decoding 단계에서 해석한 roll 들이었는 값을 Execution 단계로 전달하기 위해 사용하는 레지스티: 이 값 바탕으로 Execution 단계에서 각 instruction 별 연산을 수행한다 Decoding 단계에서 해석한 imm, target, 등의 값을 Execution 단계에서 각 instruction 별 연산을 수행한다 Decoding 단계에서 해석한 rd 번호를 Execution 단계로 전달하기 위해 사용하는 레지스티: 이 값 바탕으로 Execution 단계에서 하시한 rd 번호를 Execution 단계로 전달하기 위해 사용하는 레지스티: 이 값 바탕으로 Execution 단계에서 해석한 rd 번호를 Execution 단계로 전달하기 위해 사용하는 레지스티: 이 값 바탕으로 Execution 단계에서 해석한 instruction 별 연산을 수행한다 Decoding 단계에서 해석한 shamt를 Execution 단계로 전달하기 위해 사용하는 레지스티: 이 값 바탕으로 Execution 단계에서 해석한 instruction 별 연산을 수행한다 Decoding 단계에서 해석한 instruction 별 연산을 수행한다 이 값 바탕으로 Handle 제 전원 instruction 별 연산을 수행한다 이 값 바탕으로 Hazard를 체크한다. Decoding 단계에서 해석한 ri 번호를 Execution 단계로 전달하기 위해 사용하는 레지스티: 이 값 바탕으로 Hazard를 체크한다. Decoding 단계에서 해석한 ri 번호를 Execution 단계로 전달하기 위해 사용하는 레지스티: 이 값 바탕으로 Hazard를 체크한다. Execution 단계에서 얻은 결과를 MEM 단계 연산 선을 수행하게서, forwarding 만나 전계 연산 일을 하하게서, forwarding 만나 전계 연산 일을 하하게서, forwarding 만나 대한 기기에 사용하는 레지스티: MEM 단계 에서 su, su 한 실으로 수행하기 위해 사용하는 데지스티: MEM 단계 에서 su, su 한 설으로 수행하기 위해 사용하는 레지스티: MEM 단계 에서 su, su 한 설을 수행하기 위해 사용하는 레지스티: MEM 단계 에서 su, su 한 전을 수행하기 위해 사용하는 레지스티: MEM 단계 에서 su, su 한 선을 수행하기 위해 사용하는 레지스티: MEM 단계 에서 su, su 한 선을 수행하기 위해 사용하는 레지스티: MEM 단계 에서 su, su 한 선실을 수행하기 위해 사용하는 레지스티: MEM 단계 에서 su, su 한 선을 수행하기 위해 사용하는 레지스티: MEM 단계 에서 su, su 한 선을 수행하기 위해 사용하는 레지스티: MEM 단계 에서 su, su 한 선을 수 한 경기 위해 사용하는 레지스티: MEM 단계 에서 su, su 한 선을 수 한 경기 위해 사용하는 레지스티: MEM 단계 에서 su, su 한 선을 수 한 경기 위해 사용하는 레지스티: MEM 단계 에서 su, su 한 선을 수 한 경기 위해 사용하는 레지스티: MEM 단계 에서 su, su 한 선을 수 한 경기 위해 사용하는 레지스티: MEM 단계 에서 su, su 한 선을 수 한 기기 위해 사용하는 레지스티: MEM 단계 에서 su, su 한 선을 수 한 기기 위해 사용하는 레지스티: MEM 단계 에서 su, su 한 선을 수 한 계시 su, su 한 선을 수 한 기기 위해 사용하는 레지스티: MEM 단계 su su 한 선상을 수 한 기계 에서 su 한 선상을 수 하게 기계 su su su 한 선을 수 하게 하기 위해 사용하는 레지스티 : MEM 단계 su su su de de 수 하기 시를 내용하는 레지스티 : MEM 단계 su su su de de 수 하기 있다. MEM 단계 su su su de	_	
ID_EX.rt ID_EX.lt ID_EX.rt ID_E	ID_EX.rt ID_EX.IMM		
### DECA.H ### Decoding 단계에서 가 instruction 별 연산을 수행한다 Decoding 단계에서 혜석한 imm, target, 등의 값을 Execution 단계로 전달하기 위해 사용하는 레지스타: 이 값 바탕으로 Execution 단계에서 각 instruction 별 연산을 수행한다 Decoding 단계에서 해석한 대한 번호를 Execution 단계로 전달하기 위해 사용하는 레지스타: 이 값 바탕으로 Execution 단계에서 작 instruction 별 연산을 수행한다 Decoding 단계에서 해석한 instruction 별 연산을 수행한다 ### Decoding 단계에서 해석한 WB를 Execution 단계로 전달하기 위해 사용하는 레지스타: 이 값 바탕으로 Execution 단계에서 이 해석한 WB를 Execution 단계로 전달하기 위해 사용하는 레지스타: 이 값 바탕으로 Hazard를 체크한다 Decoding 단계에서 해석한 rs 번호를 Execution 단계로 전달하기 위해 사용하는 레지스타: 이 값 바탕으로 Hazard를 제고한다. Decoding 단계에서 해석한 rt 번호를 Execution 단계로 전달하기 위해 사용하는 레지스타: 이 값 바탕으로 Hazard를 제고한다. Execution 단계에서 연은 결과를 MEM 단계로 전달하기 위해 사용하는 레지스타: 이 값 바탕으로 MEX 대원 연산을 수행하거나, forwarding 한다. Execution 단계에서 연은 Target을 MEM 단계로 전달하기 위해 사용하는 레지스타: MEM 단계 연산을 가용하는 레지스타: MEM 단계 연간을 가용하다 단계 사용하는 레지스타: MEM 단계 전원하기 위해 사용하는 레지스타: MEM 단계 로 전달하기 위해 사용하는 레지스타: MEM 단계 로 전달하기 위해 사용하는 레지스타: MEM 단계 에서 sw. sb 연산을 수행하기 위해 사용하다 레지스타: MEM 단계 에서 sw. sb 연산을 수행하기 위해 사용하다 레지스타: MEM 단계 에서 sw. sb 연산을 수행하기 위해 사용하다 레지스타: MEM 단계 로 전달하기 위해 사용하다 레지스타: MEM 단계 로 전달하기 위해 사용하는 레지스타: MEM 단계로 전달하기 위해 사용하는 레지스타: MEM Execution 단계로 전달하기 위해 사용하는 레지스타: MEM 단계로 전달하기 위해 사용하는 레지스타:	### ### ### ### ### ### ### ### ### ##		
Decoding 단계에서 해설한 imm, target, 등의 값을 Execution 단계로 전달하기 위해 사용하는 레지스타: 이 값 바탕으로 Execution 단계에서 각 instruction 별 연산을 수행한다	Decoding 단계에서 해석한 imm, target, 등의 값을 Execution 단계로 전달하기 위해 사용하는 레지스터: 이 값 바탕으로 Execution 단계에서 강 instruction 별 연산을 수행한다 Decoding 단계에서 해석한 대 번호를 Execution 단계로 전달하기 위해 사용하는 레지스터: 이 값 바탕으로 Execution 단계에서 각 instruction 별 연산을 수행한다 Decoding 단계에서 해석한 하는 레지스터: 이 값 바탕으로 Execution 단계로 전달하기 위해 사용하는 레지스터: 이 값 바탕으로 Execution 단계로 전달하기 위해 사용하는 레지스터: 이 값 바탕으로 Execution 단계에서 각 instruction 별 연산을 수행한다 Decoding 단계에서 예석한 function 터리를 Execution 단계에서 각 instruction 별 연산을 수행한다 다기로 전달하기 위해 사용하는 레지스터: 이 값 바탕으로 Execution 단계에서 해석한 WB를 Execution 단계로 전달하기 위해 사용하는 레지스터: 이 값 바탕으로 Execution 단계에서 기 하다 한다	ID_EX.rt	
Decoding 단계에서 해석한 imm, target, 등의 값을 Execution 단계로 전달하기 위해 사용하는 레지스터: 이 값 바탕으로 Execution 단계에서 각 instruction 별 연산을 수행한다 Decoding 단계에서 해석한 대 번호를 Execution 단계로 전달하기 위해 사용하는 레지스터: 이 값 바탕으로 Execution 단계에서 각 instruction 별 연산을 수행한다 Decoding 단계에서 각 instruction 별 연산을 수행한다 Decoding 단계에서 각 instruction 별 연산을 수행한다 Decoding 단계에서 작 instruction 별 연산을 수행한다 Decoding 단계에서 해석한 shamt를 Execution 단계로 전달하기 위해 사용하는 레지스터: 이 값 바탕으로 Execution 단계에서 각 instruction 별 연산을 수행한다 Decoding 단계에서 해석한 function field 를 Execution 단계로 전달하기 위해 사용하는 레지스터: 이 값 바탕으로 Execution 단계에서 작 instruction 별 연산을 수행한다 Decoding 단계에서 해석한 WB를 Execution 단계로 전달하기 위해 사용하는 레지스터: 이 값은 WB까지 forwarding되고, 이를 바탕으로 WB 단계에서 어느 레지스터 이 값 바탕으로 Hazard를 제고한다. Decoding 단계에서 해석한 rs 번호를 Execution 단계로 전달하기 위해 사용하는 레지스터: 이 값 바탕으로 Hazard를 제고한다. Decoding 단계에서 해석한 r 번호를 Execution 단계로 전달하기 위해 사용하는 레지스터: 이 값 바탕으로 Hazard를 제고한다. Decoding 단계에서 해석한 rd 번호를 Execution 단계로 전달하기 위해 사용하는 레지스터: 이 값 바탕으로 Hazard를 제고한다. Execution 단계에서 얻은 결과를 MEM 단계로 전달하기 위해 사용하는 레지스터: 이 값 바탕으로 MEM 단계 연소을 수행하게나 forwarding 한다. Execution 단계에서 얻은 결과를 MEM 단계로 전달하기 위해 사용하는 레지스터: MEM 단계 에서 app. antp.의 TF 여부에 따라 이 값으로 분기한다. (T면 분기) Execution 단계에 당하는 레지스터: MEM 단계 에서 sw. sb. 연산을 수행하기 위해 사용하는 레지스터: MEM 단계 에서 sw. sb. 연산을 수행하기 위해 사용하는 레지스터: MEM 단계 에서 Sw. sb. 연산을 수행하기 위해 사용하는 레지스터: MEM 단계 에서 Execution 단계를 forwarding된 IMM을 다시 MEM 단계 로 전달하기 위해 사용하는 레지스터: MEM 단계 에서 Sw. sb. 연산을 수행하기 위해 사용하는 레지스터: MEM 단계 에서 MEM 단계 로 전달하기 위해 사용하는 레지스터: MEM 단계 에서 Sw. sb. 연산을 수행하기 위해 사용하는 레지스터: MEM 단계 에서 MEM 단계 로 전달하기 위해 사용하는 레지스터: MEM 단계 로 전달하기 위해 사용하는 레지스터: MEM 단계 로 전달하기 위해 사용하는 레지스터: MEM 단계 에서 MEM 단계 로 전달하기 위해 사용하는 레지스터: MEM 단계로 전달하기 위해 사용하는 레지스터: MEM 단계로 전달하기 위해 사용하는 레지스터: MEM 단계로 전달하기 위해 사용하는 레지스터 제로	Decoding 단계에서 해석한 imm, target, 등의 값을 Execution 단계에서 해석한 imm, target, 등의 값을 Execution 단계에서 가해 사용하는 레지스터: 이 값 바탕으로 Execution 단계에서 전한다 이 없 바탕으로 Execution 단계로 전달하기 위해 사용하는 레지스터: 이 값 바탕으로 Execution 단계에서 해석한 rd 번호를 Execution 단계로 전달하기 위해 사용하는 레지스터: 이 값 바탕으로 Execution 단계에서 해석한 shaml를 Execution 단계로 전달하기 위해 사용하는 레지스터: 이 값 바탕으로 Execution 단계에서 해석한 inction field Execution 단계에서 제상하는 레지스터: 이 값 바탕으로 Execution 단계로 전달하기 위해 사용하는 레지스터: 이 값 바탕으로 Hay에서 와 #한		
Execution 단계로 전달하기 위해 사용하는 레지스터: 이 값 바탕으로 Execution 단계에서 각 instruction 별 연산을 수행한다 Decoding 단계에서 해석한 rd 반호를 Execution 단계로 전달하기 위해 사용하는 레지스터: 이 값 바탕으로 Execution 단계에서 경 instruction 별 연산을 수행한다 Decoding 단계에서 해석한 sham를 Execution 단계로 전달하기 위해 사용하는 레지스터: 이 값 바탕으로 Execution 단계에서 경 instruction 별 연산을 수행한다 Decoding 단계에서 해석한 function field를 Execution 단계로 전달하기 위해 사용하는 레지스터: 이 값 바탕으로 Execution 단계에서 각 instruction 별 연산을 수행한다 Decoding 단계에서 해석한 function field를 Execution 단계로 전달하기 위해 사용하는 레지스터: 이 값 바탕으로 Execution 단계로 전달하기 위해 사용하는 레지스터: 이 값 바탕으로 Heard에서 해석한 rs 번호를 Execution 단계로 전달하기 위해 사용하는 레지스터: 이 값 바탕으로 Heard를 체크한다. Decoding 단계에서 해석한 rs 번호를 Execution 단계로 전달하기 위해 사용하는 레지스터: 이 값 바탕으로 Heard를 체크한다. Decoding 단계에서 해석한 rs 번호를 Execution 단계로 전달하기 위해 사용하는 레지스터: 이 값 바탕으로 Heard를 체크한다. Decoding 단계에서 해석한 rs 번호를 Execution 단계로 전달하기 위해 사용하는 레지스터: 이 값 바탕으로 Heard를 제크한다. Decoding 단계에서 연은 결과를 MEM 단계로 전달하기 위해 사용하는 레지스터: 이 값 바탕으로 MEM 단계 연산을 수행하게 forwarding 한다. Execution 단계에서 얻은 Target을 MEM 단계 연산을 수행하게 forwarding 한다. Execution 단계로 forwarding IMM을 다시 MEM 단계 로 전달하기 위해 사용하는 레지스터: MEM 단계 에서 sw. sb. 연산을 수행하기 위해 사용하는 레지스터: MEM 단계 에서 sw. sb. 연산을 수행하기 위해 사용하는 레지스터: MEM 단계 에서 sw. sb. 연산을 수행하기 위해 사용하는 레지스터: MEM 단계 에서 sw. sb. 연산을 수행하기 위해 사용하는 레지스터: MEM 단계 에서 Sw. sb. 연산을 수행하기 위해 사용하는 레지스터: MEM 단계 를 다시되는 MEM 전계 를 모르고 Util 다 전기로 forwarding IMM을 다시 MEM 단계 에서 Sw. sb. 연산을 수행하기 위해 사용하는 레지스터: MEM 단계 를 전작한다. MEM 단계 를 전작으로 보다 MEM 단계 를 주가보다 한다. MEM 단계 를 전작으로 Lexition 단계로 forwarding IMM을 다시 MEM 단계 를 전각하기 위해 사용하는 레지스터: MEM 단계 를 전각하기 위해 사용하는 레지스터: MEM 단계 에서 Sw. sb. 연산을 수행하기 위해 사용하는 레지스터: MEM 단계 를 전간하기 위해 사용하는 레지스터: MEM 단계 를 전기는 IMEM 전체 사용하는 레지스터: MEM 단계 를 전기로 MEM 단계 된 전 한다. MEM 단계 를 전 한다. MEM 단계 된 전 한다. MEM 단계 단계로 전달하기 위해 사용하는 레지스터: MEM 단계 에서 Sw. sb. 연산을 수행하기 위해 사용하는 레지스터: MEM 단계 단계 를 전 한다. MEM 단계 단계 를 전 한다. MEM 단계 단계 를 전 MEM 단계 전 MEM 단계 를 전 MEM The Target MEM The Target MEM T	Execution 단계로 전달하기 위해 사용하는 레지스터: 이 값 바탕으로 Execution 단계에서 각 instruction 별 연산을 수행한다 Decoding 단계에서 해석한 rd 번호를 Execution 단계로 전달하기 위해 사용하는 레지스터: 이 값 바탕으로 Execution 단계에서 해석한 rd 번호를 Execution 단계로 전달하기 위해 사용하는 레지스터: 이 값 바탕으로 Execution 단계에서 해석한 shamt를 Execution 단계로 전달하기 위해 사용하는 레지스터: 이 값 바탕으로 Execution 단계에서 해석한 shamt를 Execution 단계로 전달하기 위해 사용하는 레지스터: 이 값 바탕으로 Execution 단계에서 해석한 function_field를 Execution 단계에서 해석한 function_field를 Execution 단계로 전달하기 위해 사용하는 레지스터: 이 값 바탕으로 Execution 단계에서 해석한 WB를 Execution 단계로 전달하기 위해 사용하는 레지스터: 이 값 바탕으로 Hand에 되지 하는 제 기 하는 대로 전달하기 위해 사용하는 레지스터: 이 값 바탕으로 Hand에를 체크한다. Decoding 단계에서 해석한 rs 번호를 Execution 단계로 전달하기 위해 사용하는 레지스터: 이 값 바탕으로 Hand에를 체크한다. Decoding 단계에서 해석한 rd 번호를 Execution 단계로 전달하기 위해 사용하는 레지스터: 이 값 바탕으로 Hand에를 체크한다. Execution 단계에서 얻은 결과를 MEM 단계 연설을 수행하기 위해 사용하는 레지스터: 이 값 바탕으로 Hand는 이 값 바탕으로 Hand는 다 제 가 하는 데지스터: 이 값 바탕으로 Hand는 대자를 제고하다. If 모든 보이는 따를 보이기 위해 사용하는 레지스터: 이 값 바탕으로 Hand는 대자를 제고하다. If 모든 보이기 위해 사용하는 레지스터: 이 값 바탕으로 Hand는 대자를 제고하다. If 모든 보이기 위해 사용하는 레지스터: If 모든 보이게 에서 Suc.sb 인산을 수행하기 위해 사용된다. Execution 단계로 forwarding한 IT에 들어있는 때를 다시 MEM 단계 에서 Suc.sb 인산을 수행하기 위해 사용된다. Execution 단계로 forwarding된 IT에 들어있는 때를 다시 MEM 단계 에서 Suc.sb 인산을 수행하기 위해 사용된다. Execution 단계로 forwarding된 IT에 들어있는 때를 다시 MEM 단계 에서 Suc.sb 인산을 수행하기 위해 사용된다. Execution 단계로 forwarding된 IT에 들어있는 때를 다시 MEM 단계 에서 Suc.sb 인산을 수행하기 위해 사용된다. Execution 단계로 forwarding된 IT에 들어있는 때를 다시 MEM 단계 에서 Suc.sb 인산을 수행하기 위해 사용된다. Execution 단계로 forwarding된 IT IT에 들어있는 IT에 MEM 단계 에서 Suc.sb 인산을 수행하기 위해 사용된다. Execution 단계로 forwarding된 IT에 들어나는 IT에 MEM 단계 에서 Suc.sb 인산을 수행하기 위해 사용된다. Execution 단계로 forwarding된 IT IT에 되어 IT에 MEM 단계 에서 Suc.sb 인산을 수행하기 위해 사용된다. IT에 MEM 단계 에서 Suc.sb 인산을 수행하기 위해 사용된다. Execution 단계로 forwarding된 IT IT에 를 다시 MEM 단계 에서 Suc.sb 인산을 수행하기 위해 사용된다. Execution 단계로 forwarding된 IT IT에 IT에 MEM 단계 에서 Suc.sb 인산을 수행하기 위해 사용된다. IT에 MEM 단계 에서 Suc.sb 인산을 수행하기 위해 사용된다. IT에 MEM 단계 에서 Suc.sb 인산을 수행하기 위해 사용된다. IT에 MEM 단계 에서 Suc.sb 인산을 수행하기 위해 사용된다. IT에 MEM 단계 에서 Suc.sb 인산을 수행하기 위해 사용된다. IT에 MEM 단계 에서 Suc.sb 인산을 수행하기 위해 사용된다. IT에 I		
ID_EX.rd Decoding 단계에서 해석한 rd 번호를 Execution 단계로 전발하기 위해 사용하는 레지스타: 이 값 바탕으로 Execution 단계에서 참석한 rd 번호를 Execution 단계로 전발하기 위해 사용하는 레지스타: 이 값 바탕으로 Execution 단계에서 착한 shamt를 Execution 단계로 전발하기 위해 사용하는 레지스타: 이 값 바탕으로 Execution 단계에서 착한 shamt를 Execution 단계로 전발하기 위해 사용하는 레지스타: 이 값 바탕으로 Execution 단계에서 작 instruction 별 연산을 수행한다 Decoding 단계에서 해석한 function_field를 Execution 단계로 전발하기 위해 사용하는 레지스타: 이 값 바탕으로 Execution 단계에서 각 instruction 별 연산을 수행한다 마케로 전발하기 위해 사용하는 레지스타: 이 값 바탕으로 Execution 단계에서 작 instruction 별 연산을 수행한다 마케로 전발하기 위해 사용하는 레지스타: 이 값 바탕으로 Hazarding되고, 이를 바탕으로 WB 단계에서 이스 레지스타 이 값 바탕으로 Hazard를 체크한다. Decoding 단계에서 해석한 rs 번호를 Execution 단계로 전발하기 위해 사용하는 레지스타: 이 값 바탕으로 Hazard를 체크한다. Decoding 단계에서 해석한 rd 번호를 Execution 단계로 전발하기 위해 사용하는 레지스타: 이 값 바탕으로 Hazard를 체크한다. Execution 단계에서 얻은 결과를 MEM 단계로 전달하기 위해 사용하는 레지스타: 이 값 바탕으로 Hazard를 제고한다. Execution 단계에서 얻은 Target을 MEM 단계 연산을 수행하거나, forwarding 한다. Execution 단계에서 얻은 Target을 MEM 단계로 전달하기 위해 사용하는 레지스타: MEM 단계 에서 ap, anp의 T/F 여부에 따라 이 값으로 분기한다. [T면 분기] Execution 단계로 forwarding된 IMM을 다시 MEM 단계 에서 sw, sb 연산을 수행하기 위해 사용하는 레지스타 MEM 단계 에서 sw, sb 연산을 수행하기 위해 사용하는 레지스타 MEM 단계 에서 Sw, sb 연산을 수행하기 위해 사용하는 레지스타 MEM 단계 에서 Sw, sb 연산을 수행하기 위해 사용하는 레지스타 MEM 단계 에서 Sw, sb 연산을 수행하기 위해 사용하는 레지스타 MEM 단계 에서 MEM 단계로 전달하기 위해 사용하는 레지스타 MEM 단계 에서 Sw, sb 연산을 수행하기 위해 사용하는 레지스타 MEM 단계 MEM 단계로 전달하기 위해 사용하는 레지스타 MEM 단계 에서 sw, sb 연산을 수행하게 위해 사용하는 레지스타 MEM 단계 MEM 단계로 전달하기 위해 사용하는 레지스타 MEM 단계 MEM 단계 MEM 단계로 전략 제품 전략 MEM 단계 MEM 단계로 전략 제품 전략 MEM 단계로 제품	DEX.rd Decoding 단계에서 해석한 rd 번호를 Execution 단계로 전달하기 위해 사용하는 레지스터: 이 값 바탕으로 Execution 단계에서 해석한 rd 번호를 Execution 단계로 전달하기 위해 사용하는 레지스터: 이 값 바탕으로 Execution 단계에서 해석한 shamt를 Execution 단계로 전달하기위해 사용하는 레지스터: 이 값 바탕으로 Execution 단계에서 해석한 shamt를 Execution 단계로 전달하기위해 사용하는 레지스터: 이 값 바탕으로 Execution 단계에서 해석한 function field를 Execution 단계로 전달하기위해 사용하는 레지스터: 이 값 바탕으로 Execution 단계에서 해석한 function 별 연산을 수행한다 다 모르는 마이에서 가 instruction 별 연산을 수행한다 다 마이에서 가 instruction 별 연산을 수행한다 다 마이에서 가 instruction 별 연산을 수행한다 다 마이에서 바탕으로 WB 단계에서 이스를 사용하는 레지스터: 이 값 e WB 까지 forwarding 되고 이를 바탕으로 WB 단계에서 이스를 제고 는 타에 write 할 지 결정한다. Decoding 단계에서 해석한 rs 번호를 Execution 단계로 전달하기위해 사용하는 레지스터: 이 값 바탕으로 Hazard를 체크한다. Decoding 단계에서 해석한 rt 번호를 Execution 단계로 전달하기위해 사용하는 레지스터: 이 값 바탕으로 Hazard를 제크한다. EX_MEM.ALU_OUT 원리에서 대적 연은 결과를 MEM 단계로 전달하기위해 사용하는 레지스터: 이 값 바탕으로 MEM 단계 연산을 수행하게 위해 사용하는 레지스터: MEM 단계 에서 되고 대한 기위해 사용하는 레지스터: MEM 단계 에서 모르 집달하기위해 사용하는 레지스터: MEM 단계 에서 Sw. sb 연산을 수행하기위해 사용되다. EX_MEM.rt EX_MEM.rt EX_MEM.rs_num EX_MEM.rs_num Decoding 단계에서 해석한 rt 번호를 Execution 단계로 전달하기위해 사용하는 레지스터: MEM 단계 에서 sp. sp. ap 대한 다 제로 전달하기위해 사용되다. Execution 단계로 forwarding된 rt에 들어있는 값을 다시 MEM 단계 에서 sw. sb 연산을 수행하기위해 사용되다. Execution 단계로 forwarding된 rt에 들어있는 값을 다시 MEM 단계 에서 sw. sb 연산을 수행하기위해 사용되다. Execution 단계로 forwarding rt rto 들어있는 값을 다시 MEM 단계 에서 sw. sb 연산을 수행하기위해 사용되다. Execution 단계로 forwarding rto 대들어있는 대상 대표 단계에서 sw. sb 연산을 수행하기위해 사용되다. Execution 단계로 forwarding rto 대들어있는 대상 대표 단계 에서 sw. sb 연산을 수행하기위해 사용되다. Execution 단계로 forwarding rto 대들어있는 대상 대표 전달하기위해 사용하는 레지스터: MEM 단계 대로 전달하기위해 사용하는 레지스터: 이 값 바탕으로 Hazard를 제고한다.	ID EX IMM	
Decoding 단계에서 해석한 rd 번호를 Execution 단계로 전달하기 위해 사용하는 레지스타: 이 값 바탕으로 Execution 단계에서 각 instruction 별 연산을 수행한다 Decoding 단계에서 해석한 shamt를 Execution 단계로 전달하기 위해 사용하는 레지스터: 이 값 바탕으로 Execution 단계에서 작 instruction 별 연산을 수행한다 Decoding 단계에서 해석한 shamt를 Execution 단계로 전달하기 위해 사용하는 레지스터: 이 값 바탕으로 Execution 단계에서 각 instruction 별 연산을 수행한다 Decoding 단계에서 해석한 tuncton 등ield Execution 단계에서 각 instruction 별 연산을 수행한다 P에 한 사용하는 레지스터: 이 값 바탕으로 Execution 단계에서 작 instruction 별 연산을 수행한다 P에 한 사용하는 레지스터: 이 값 바탕으로 Execution 단계로 전달하기 위해 사용하는 레지스터: 이 값 바탕으로 WB 단계에서 이트리어 Write 할 지 결정한다. Decoding 단계에서 해석한 rs 번호를 Execution 단계로 전달하기 위해 사용하는 레지스터: 이 값 바탕으로 Hazard를 체크한다. Decoding 단계에서 해석한 r 번호를 Execution 단계로 전달하기 위해 사용하는 레지스터: 이 값 바탕으로 Hazard를 체크한다. EX_MEM.ALU_OUT 위해 사용하는 레지스터: 이 값 바탕으로 MEM 단계 연산을 수행하거나, forwarding 한다. Execution 단계에서 얻은 Target을 MEM 단계로 전달하기 위해 사용하는 레지스터: MEM 단계 에서 SW, Sb 연산을 수행하기 위해 사용하는 레지스터: MEM 단계 에서 SW, Sb 연산을 수행하기 위해 사용하다 레지어서 MEM 단계 에서 SW, Sb 연산을 수행하기 위해 사용하다 레지어 HEM 단계 에서 SW, Sb 연산을 수행하기 위해 사용하는 레지스터: MEM 단계 에서 SW, Sb 연산을 수행하기 위해 사용하는 레지스터: MEM 단계 에서 SW, Sb 연산을 수행하기 위해 사용하는 레지스터: MEM 단계 에서 SW, Sb 연산을 수행하기 위해 사용하는 레지스터: MEM 단계 에서 SW, Sb 연산을 수행하기 위해 사용하다 레지어나 SW, Sb 연산을 수행하기 위해 사용하는 레지스터: MEM 단계 에서 SW, Sb 연산을 수행하기 위해 사용하는 레지스터: MEM 단계 에서 SW, Sb 연산을 수행하기 위해 사용하는 레지스터: MEM 단계 에서 SW, Sb 연산을 수행하기 위해 사용하는 레지스터: MEM 단계 SW, Sb 연산을 수행하기 위해 SP 한 S	ID_EX.rd Decoding 단계에서 해석한 rd 번호를 Execution 단계로 천당하기 위해 사용하는 레지스터: 이 값 바탕으로 Execution 단계에서 강 instruction 별 연산을 수행한다 Decoding 단계에서 해석한 shamt를 Execution 단계로 전 당하기 위해 사용하는 레지스터: 이 값 바탕으로 Execution 단계에서 각 instruction 별 연산을 수행한다 Decoding 단계에서 작 instruction 별 연산을 수행한다 Decoding 단계에서 해석한 function field를 Execution 단계로 전당하기 위해 사용하는 레지스터: 이 값 바탕으로 Execution 단계에서 각 instruction 별 연산을 수행한 도로 Execution 단계에서 하석한 WB를 Execution 단계로 전당하기 위해 사용하는 레지스터: 이 값 바탕으로 Execution 단계에서 해석한 WB를 Execution 단계로 전당하기 위해 사용하는 레지스터: 이 값 e WB 자지 forwarding되고, 이를 바탕으로 WB 단계에서 어느 레지스터에 write 할 지 결정한다. Decoding 단계에서 해석한 rs 번호를 Execution 단계로 전당하기 위해 사용하는 레지스터: 이 값 바탕으로 Hazard를 체코한다. Decoding 단계에서 해석한 rt 번호를 Execution 단계로 전당하기 위해 사용하는 레지스터: 이 값 바탕으로 Hazard를 제코한다. EX_MEM.ALU_OUT EX_MEM.ALU_OUT Execution 단계에서 얻은 결과를 MEM 단계로 전당하기 위해 사용하는 레지스터: 이 값 바탕으로 MEM 단계 연산을 수행하거나, forwarding 한다. Execution 단계에서 얻은 제공약을 MEM 단계로 전당하기 위해 사용하는 레지스터: MEM 단계 에서 atp, antp의 17년 여부에 따라 이 값으로 분기한다. (T면 분기) Execution 단계에서 안으로 MEM 단계 에서 atp, antp의 17년 이부에 따라 이 값으로 분기한다. (T면 분기) Execution 단계를 forwarding된 IMM을 다시 MEM 단계 에서 5%, 50 연산을 수행하기 위해 사용하는 레지스터: MEM 단계 에서 5%, 50 연산을 수행하기 위해 사용하는 레지스터: MEM 단계 에서 5%, 50 연산을 수행하기 위해 사용하는 레지스터: MEM 단계 에서 5%, 50 연산을 수행하기 위해 사용하는 레지스터: MEM 단계 에서 5%, 50 연산을 수행하기 위해 사용하는 레지스터: MEM 단계 에서 5%, 50 연산을 수행하기 위해 사용하는 레지스터: MEM 단계 에서 5%, 50 연산을 수행하기 위해 사용하는 레지스터: MEM 단계 에서 5%, 50 연산을 수행하기 위해 사용하는 레지스터: MEM 단계 에서 5%, 50 연산을 수행하기 위해 사용하는 레지스터: MEM 단계 에서 5%, 50 연산을 수행하기 위해 사용하는 레지스터: MEM 단계 에서 5%, 50 연산을 수행하기 위해 사용하는 레지스터: MEM 단계 에서 5%, 50 연산을 수행하기 위해 사용하는 레지스터: 이 값 바탕으로 Hazard를 체코한다. Execution 단계로 forwarding된 In 번호를 다시 MEM 단계 세수 5%, 50 연산을 수행하기 위해 사용하는 레지스터: 이 값 바탕으로 Hazard를 체코한다.	ID_EX.IIVIVI	
DEX.rd 전달하기 위해 사용하는 레지스터: 이 값 바탕으로 Execution 단계에서 각 instruction 별 연산을 수행한다 Decoding 단계에서 해석한 shamt를 Execution 단계로 전달하기 위해 사용하는 레지스터: 이 값 바탕으로 Execution 단계에서 해석한 function field를 Execution 단계에서 라시한 function field를 Execution 단계에서 각 instruction 별 연산을 수행한다 Decoding 단계에서 해석한 function field를 Execution 단계에서 각 instruction 별 연산을 수행한다 Decoding 단계에서 해석한 WB를 Execution 단계로 전달하기 위해 사용하는 레지스터: 이 값 바탕으로 WB 단계에서 이느 레지 forwarding되고, 이를 바탕으로 WB 단계에서 이느 레지	ID_EX.rd 전달하기 위해 사용하는 레지스타: 이 값 바탕으로 Execution 단계에서 각 instruction 별 연산을 수행한다 Decoding 단계에서 해석한 shamt를 Execution 단계로 전 달하기 위해 사용하는 레지스타: 이 값 바탕으로 Execution 단계에서 각 instruction 별 연산을 수행한다 Decoding 단계에서 해석한 function Field를 Execution 단계로 전달하기 위해 사용하는 레지스타: 이 값 바탕으로 Execution 단계에서 각 instruction 별 연산을 수행한다 Decoding 단계에서 해석한 function Field를 Execution 단계로 전달하기 위해 사용하는 레지스타: 이 값은 WB까지 forwarding되고, 이를 바탕으로 WB 단계에서 어느 래지 스타에 write 할 지 결정한다. Decoding 단계에서 해석한 rs 번호를 Execution 단계로 전달하기 위해 사용하는 레지스타: 이 값 바탕으로 Hazard를 체크한다. Decoding 단계에서 해석한 rt 번호를 Execution 단계로 전달하기 위해 사용하는 레지스타: 이 값 바탕으로 Hazard를 체크한다. Decoding 단계에서 해석한 rt 번호를 Execution 단계로 전달하기 위해 사용하는 레지스타: 이 값 바탕으로 Hazard를 체크한다. EX_MEM.ALU_OUT Execution 단계에서 얻은 결과를 MEM 단계로 전달하기 위해 사용하는 레지스타: 어 값 바탕으로 MEM 단계 연산을 수행하게, 위해 사용하는 레지스타: MEM 단계 에서 app, amp의 T/F 여부에 따라 이 값으로 분기한다. (T먼 분기) EX_MEM.IMM EX		
Execution 단계에서 각 instruction 별 연산을 수행한다	Execution 단계에서 각 instruction 별 연산을 수행한다 Decoding 단계에서 해석한 shaml를 Execution 단계로 전 달하기 위해 사용하는 레지스타: 이 값 바탕으로 Execution 단계에서 해석한 function_field를 Execution 단계로 전 달하기 위해 사용하는 레지스타: 이 값 바탕으로 Execution 단계에서 해석한 function_field를 Execution 단계로 전 달하기 위해 사용하는 레지스타: 이 값 바탕으로 Execution 단계에서 해석한 function field를 Execution 단계로 전 달하기 위해 사용하는 레지스타: 이 값 바탕으로 Execution 단계에서 해석한 WB를 Execution 단계로 전 달하기 위해 사용하는 레지스타: 이 값은 WB 까지 forwarding되고, 이를 바탕으로 WB 단계에서 이느 래지 스타에 write 할 지 결정한다. Decoding 단계에서 해석한 rs 번호를 Execution 단계로 전달하기 위해 사용하는 레지스타: 이 값 바탕으로 Hazard를 체크한다. Decoding 단계에서 해석한 rs 번호를 Execution 단계로 전달하기 위해 사용하는 레지스타: 이 값 바탕으로 Hazard를 체크한다. Execution 단계에서 얻은 결과를 MEM 단계로 전달하기 위해 사용하는 레지스타: MEM 단계 연소을 수행하게 나 forwarding한다. Execution 단계에서 얻은 결과를 MEM 단계로 전달하기 위해 사용하는 레지스타: MEM 단계 에서 app antp의 TVF 여부에 따라 이 값으로 분기한다 (T먼 분기) Execution 단계에도 forwarding 한다. Execution 단계에도 forwarding 된 IMM을 다시 MEM 단계 로 전달하기 위해 사용하는 레지스타: MEM 단계 에서 sw. sb 연산을 수행하기 위해 사용하는 레지스타 scecution 단계로 대표 sdeeth sale sale sale sale sale sale sale sale	ID EV -1	
Decoding 단계에서 해석한 shamt를 Execution 단계로 전달하기 위해 사용하는 레지스터: 이 값 바탕으로 Execution 단계에서 각 instruction 별 연산을 수행한다 Decoding 단계에서 해석한 function field	Decoding 단계에서 해석한 shamt를 Execution 단계로 전 달하기 위해 사용하는 레지스터: 이 값 바탕으로 Execution 단계에서 각 instruction 별 연산을 수행한다 Decoding 단계에서 해석한 function field를 Execution 단계로 전달하기 위해 사용하는 레지스터: 이 값 바탕으로 Execution 단계에서 각 instruction 별 연산을 수행한 다 보고 전달하기 위해 사용하는 레지스터: 이 값 바탕으로 Execution 단계에서 각 instruction 별 연산을 수행한 다 보고 전달하기 위해 사용하는 레지스터: 이 값 바탕으로 WB 단계에서 어느 래지 스타에게서 함께 한다. 먼지를 전달하기 위해 사용하는 레지스터: 이 값 바탕으로 Harard를 제고한다. Decoding 단계에서 해석한 rs 번호를 Execution 단계로 전달하기 위해 사용하는 레지스터: 이 값 바탕으로 Harard를 제고한다. Decoding 단계에서 해석한 rs 번호를 Execution 단계로 전달하기 위해 사용하는 레지스터: 이 값 바탕으로 Harard를 제고한다. Decoding 단계에서 해석한 rt 번호를 Execution 단계로 전달하기 위해 사용하는 레지스터: 이 값 바탕으로 Harard를 제고한다. Execution 단계에서 이에서 연은 결과를 MEM 단계로 전달하기 위해 사용하는 레지스터: 이 값 바탕으로 MEM 단계 연산을 수행하거나, forwarding 한다. Execution 단계에서 얻은 Target을 MEM 단계 로전달하기 위해 사용하는 레지스터: MEM 단계 에서 모르는 보기한다. (T편 분기) Execution 단계로 forwarding된 IMM을 다시 MEM 단계 에서 Sw., sb 연산을 수행하기 위해 사용된다. Execution 단계로 전달하기 위해 사용된다. Execution 단계로 전달하기 위해 사용된다. Execution 단계로 전달하기 위해 사용하는 레지스터: MEM 단계 에서 Sw., sb 연산을 수행하기 위해 사용된다. Execution 단계로 forwarding된 IT에를 들어있는 값을 다시 MEM 단계 에서 Sw., sb 연산을 수행하기 위해 사용된다. Execution 단계로 전달하기 위해 사용된다. Execution 단계로 전달하기 위해 사용된다. Execution 단계로 forwarding된 IT에를 어있는 값을 다시 MEM 단계 에서 Sw., sb 연산을 수행하기 위해 사용된다. Execution 단계로 forwarding된 IT에를 어있는 값을 다시 MEM 단계 에서 Sw., sb 연산을 수행하기 위해 사용된다. Execution 단계로 forwarding된 IT 및 한트를 다시 MEM 단계 에서 Sw., sb 연산을 수행하기 위해 사용된다. Execution 단계로 forwarding된 IT 및 한트를 다시 MEM 단계 에서 Sw., sb 연산을 수행하기 위해 사용된다. Execution 단계로 forwarding된 IT 및 한트를 다시 MEM 단계 에서 Sw., sb 연산을 수행하기 위해 사용한다. 레지스터: MEM 단계 에서 Sw., sb 연산을 수행하기 위해 사용한다. 레지스터: Of 값 바탕으로 Hazard를 제고한다.	ID_Ex.rd	
ID_EX.shamt	ID_EX.shamt 말하기 위해 사용하는 레지스터: 이 값 바탕으로 Execution 단계에서 각 instruction 별 연산을 수행한다 Decoding 단계에서 해석한 function field를 Execution 단계로 전달하기 위해 사용하는 레지스터: 이 값 바탕으로 Execution 단계에서 각 instruction 별 연산을 수행한다 Decoding 단계에서 가 instruction 별 연산을 수행한다 Decoding 단계에서 해석한 WB를 Execution 단계로 전달하기 위해 사용하는 레지스터: 이 값은 WB 장지 forwarding되고, 이를 바탕으로 WB 단계에서 어느 레지스터에 write 할 지 결정한다. Decoding 단계에서 해석한 rs 번호를 Execution 단계로 전달하기 위해 사용하는 레지스터: 이 값 바탕으로 Hazard를 체크한다. Decoding 단계에서 해석한 rs 번호를 Execution 단계로 전달하기 위해 사용하는 레지스터: 이 값 바탕으로 Hazard를 체크한다. EX_MEM.ALU_OUT Decoding 단계에서 해석한 rs 번호를 Execution 단계로 전달하기 위해 사용하는 레지스터: 이 값 바탕으로 Hazard를 제크한다. Execution 단계에서 연은 결과를 MEM 단계로 전달하기 위해 사용하는 레지스터: 이 값 바탕으로 MEM 단계 연산을 수행하거나, forwarding 한다. Execution 단계에서 얻은 Target을 MEM 단계로 전달하기 위해 사용하는 레지스터: MEM 단계 에서 app antp의 17 여부에 따라 이 값으로 분기한다. (T면 분기) Execution 단계에서 함의 다리 MEM 단계 에서 app antp의 17 여부에 따라 이 값으로 분기한다. (T면 분기) Execution 단계로 forwarding된 IMM을 다시 MEM 단계 에서 sw, sb. 연산을 수행하기 위해 사용된다. Execution 단계로 forwarding된 rn 에들어있는 값을 다시 MEM 단계 에서 sw, sb. 연산을 수행하기 위해 사용된다. Execution 단계로 forwarding된 rn 에들어있는 값을 다시 MEM 단계 에서 sw, sb. 연산을 수행하기 위해 사용된다. Execution 단계로 forwarding된 rs 번호를 다시 MEM 단계 에서 sw, sb. 연산을 수행하기 위해 사용된다. Execution 단계로 forwarding된 rs 번호를 다시 MEM 단계 에서 sw, sb. 연산을 수행하기 위해 사용된다.		
Execution 단계에서 각 instruction 별 연산을 수행한다 Decoding 단계에서 하석한 function_field를 Execution 단계로 전달하기 위해 사용하는 레지스터: 이 값 바탕으로 Execution 단계에서 각 instruction 별 연산을 수행한다 Decoding 단계에서 해석한 WB를 Execution 단계로 전달하기 위해 사용하는 레지스터: 이 값은 WB까지 forwarding되고, 이를 바탕으로 WB 단계에서 어느 레지스터에 배석한 rs 번호를 Execution 단계로 전달하기 위해 사용하는 레지스터: 이 값 바탕으로 Hazard를 체크한다. Decoding 단계에서 해석한 rs 번호를 Execution 단계로 전달하기 위해 사용하는 레지스터: 이 값 바탕으로 Hazard를 체크한다. Decoding 단계에서 해석한 rt 번호를 Execution 단계로 전달하기 위해 사용하는 레지스터: 이 값 바탕으로 Hazard를 체크한다. Ex_MEM.ALU_OUT Execution 단계에서 얻은 결과를 MEM 단계로 전달하기 위해 사용하는 레지스터: 이 값 바탕으로 MEM 단계 연산을 수행하거나, forwarding 한다. Execution 단계에서 얻은 Target을 MEM 단계로 전달하기 위해 사용하는 레지스터: MEM 단계 에서 atp, antp의 T/F 여부에 따라 이 값으로 분기한다. (T면 분기) Execution 단계로 forwarding된 IMM을 다시 MEM 단계 로 전달하기 위해 사용하는 레지스터: MEM 단계 에서 sw. sb 연산을 수행하기 위해 사용된다. Execution 단계로 forwarding된 IMM을 다시 MEM 단계 로 전달하기 위해 사용하는 레지스터: MEM 단계 에서 sw. sb 연산을 수행하기 위해 사용하는 레지스터: MEM 단계 에서 sw. sb 연산을 수행하기 위해 사용하는 레지스터: MEM 단계 에서 sw. sb 연산을 수행하기 위해 사용하는 레지스터: MEM 단계 에서 sw. sb 연산을 수행하기 위해 사용하는 레지스터: MEM 단계 에서 sw. sb 연산을 수행하기 위해 사용하는 레지스터: MEM 단계 에서 sw. sb 연산을 수행하기 위해 사용하는 레지스터: MEM 단계 에서 sw. sb 연산을 수행하기 위해 사용하는 레지스터: MEM 단계 에서 sw. sb 연산을 수행하기 위해 사용하는 레지스터: MEM 단계 에서 sw. sb 연산을 수행하기 위해 사용하는 레지스터: MEM 단계 대로 전달하기 위해 사용하는 레지스터: MEM 단계 대로 제로	Execution 단계에서 각 instruction 별 연산을 수행한다 Decoding 단계에서 해석한 function field를 Execution 단계로 전달하기 위해 사용하는 레지스터: 이 값 바탕으로 Execution 단계에서 각 instruction 별 연산을 수행한다 Decoding 단계에서 해석한 WB를 Execution 단계로 전달하기 위해 사용하는 레지스터: 이 값은 WB 까지 forwarding되고, 이를 바탕으로 WB 단계에서 어느 레지스터에 write 할 지 결정한다. Decoding 단계에서 해석한 rs 번호를 Execution 단계로 전달하기 위해 사용하는 레지스터: 이 값 바탕으로 Hazard를 제크한다 Decoding 단계에서 해석한 rt 번호를 Execution 단계로 전달하기 위해 사용하는 레지스터: 이 값 바탕으로 Hazard를 제크한다 Decoding 단계에서 해석한 rt 번호를 Execution 단계로 전달하기 위해 사용하는 레지스터: 이 값 바탕으로 Hazard를 제크한다 EX_MEM.ALU_OUT 원해 사용하는 레지스터: 이 값 바탕으로 MEM 단계 연산을 수행하거나, forwarding 한다. Execution 단계에서 얻은 결과를 MEM 단계로 전달하기 위해 사용하는 레지스터: MEM 단계 에서 1개로 전달하기 위해 1개로 전달	ID EX.shamt	
ID.EX.function_field Decoding 단계에서 해석한 function_field를 Execution 단계로 전달하기 위해 사용하는 레지스터: 이 값 바탕으로 Execution 단계에서 각 instruction 별 연산을 수행한다 ID.EX.WB Decoding 단계에서 해석한 WB를 Execution 단계로 전달하기 위해 사용하는 레지스터: 이 값은 WB까지 forwarding되고, 이를 바탕으로 WB 단계에서 어느 레지스터에 write 할 지 결정한다. ID.EX.rs_num Decoding 단계에서 해석한 rs 번호를 Execution 단계로 전달하기 위해 사용하는 레지스터: 이 값 바탕으로 Hazard를 체크한다. ID.EX.rt_num Decoding 단계에서 해석한 rs 번호를 Execution 단계로 전달하기 위해 사용하는 레지스터: 이 값 바탕으로 Hazard를 체크한다. EX_MEM.ALU_OUT Execution 단계에서 일은 결과를 MEM 단계 연선을 수행하거나, forwarding 한다. EX_MEM.BR_TARGET Execution 단계로 forwarding 한다. (T면 분기) EX_MEM.IMM Execution 단계로 forwarding된 IMM을 다시 MEM 단계 서울한는 레지스터: MEM 단계 에서 sw, sb 연산을 수행하기 위해 사용된다. MEM 단계 에서 sw, sb 연산을 수행하기 위해 사용된다. MEM 단계 에서 sw, sb 연산을 수행하기 위해 사용된다. MEM 단계 대체적 sw, sb 연산을 수행하기 위해 사용된다. MEM 단계 Execution 단계로 forwarding된 rm 들어있는 값을 다시 MEM 단계로 전달하기 위해 사용하는 레지스터: MEM 단계 Ham Execution 단계로 forwarding rm 를어있는 값을 다시 MEM 단계로 전달하기 위해 사용하는 레지스터: MEM 단계 Execution 단계로 forwarding rm 를어있는 값을 다시 MEM 단계로 전달하기 위해 사용하는 레지스터: MEM 단계 Execution 단계로 forwarding rm 를어있는 값을 다시 MEM 단계로 전달하기 위해 사용하는 레지스터: MEM Execution 단계로 forwarding rm 를어있는 값을 다시 MEM 단계로 전달하기 위해 사용하는 레지스터: MEM 단계 CPI Execution 단계로 forwarding rm 를어있는 값을 다시 MEM 단계로 전달하기 위해 사용하는 레지스터: MEM 단계 CPI Execution 단계로 forwarding rm 를어있는 값을 다시 MEM 단계로 전달하기 위해 사용하는 레지스터: MEM Execution 단계로 전달하기 위해 사용하는 레지스터: MEM 단계 CPI Execution 단계로 전달하기 위해 사용하는 레지스터: MEM 단계 CPI Execution 단계로 전달하기 위해 사용하는 레지스터: MEM 단계 CPI Execution 단계로 전달하기 위해 사용하는 레지스터: MEM 단계 CPI Execution 단계로 전달하기 위해 사용하는 레지스터: MEM 단계 CPI Execution 단계로 전달하기 위해 사용하는 레지스터: MEM CPI Execution Execution 단계로 전달하기 위해 사용하는 레지스터: MEM CPI Execution Execution Execution Execution Execution Execution Executi	Decoding 단계에서 해석한 function_field를 Execution 단계로 전달하기 위해 사용하는 레지스타: 이 값 바탕으로 Execution 단계에서 각 instruction 별 연산을 수행한다. Decoding 단계에서 해석한 WB를 Execution 단계로 전달하기 위해 사용하는 레지스타: 이 값은 WB 까지 forwarding되고 이를 바탕으로 WB 단계에서 이느 레지스타에 write 할 지 결정한다. Decoding 단계에서 해석한 TV 번호를 Execution 단계로 전달하기 위해 사용하는 레지스타: 이 값 바탕으로 Hazard를 체크한다. Decoding 단계에서 해석한 TV 번호를 Execution 단계로 전달하기 위해 사용하는 레지스타: 이 값 바탕으로 Hazard를 체크한다. Decoding 단계에서 해석한 TV 번호를 Execution 단계로 전달하기 위해 사용하는 레지스타: 이 값 바탕으로 Hazard를 체크한다. EX_MEM.ALU_OUT EX_MEM.ALU_OUT EX_MEM.BR_TARGET EX_MEM.BR_TARGET EX_CUTION 단계에서 얻은 결과를 MEM 단계로 전달하기 위해 사용하는 레지스타: 이 값 바탕으로 MEM 단계 연산을 수행하거나, forwarding 한다. EX_CUTION 단계에서 얻은 Target을 MEM 단계 전달하기 위해 사용하는 레지스타: MEM 단계 에서 ap, antp의 TF 여부에 따라 이 값으로 분기한다. (T면 분기) Execution 단계로 forwarding된 IMM을 다시 MEM 단계 외상하는 레지스타 MEM 단계 에서 SW, Sb 연산을 수행하기 위해 사용되는 대지스타: MEM 단계 에서 SW, Sb 연산을 수행하기 위해 사용되다. EX_MEM.rt EX_MEM.rt EX_MEM.rs_num EX_MEM.rs_num Decoding 단계에서 해석한 TV 해석한 New Yell New	_	
### DECOding 단계에서 각 instruction 별 연산을 수행한 다 #### DECODING 단계에서 해석한 WB를 Execution 단계로 전 달하기 위해 사용하는 레지스터: 이 값은 WB까지 forwarding되고, 이를 바탕으로 WB 단계에서 어느 레지스터에 write할 지 결정한다. #### DECODING 단계에서 해석한 rs 번호를 Execution 단계로 전달하기 위해 사용하는 레지스터: 이 값 바탕으로 Hazard를 체크한다. #### DECODING 단계에서 해석한 rs 번호를 Execution 단계로 전달하기 위해 사용하는 레지스터: 이 값 바탕으로 Hazard를 체크한다. #### DECODING 단계에서 해석한 rs 번호를 Execution 단계로 전달하기 위해 사용하는 레지스터: 이 값 바탕으로 Hazard를 체크한다. #### Execution 단계에서 얻은 결과를 MEM 단계로 전달하기 위해 사용하는 레지스터: 이 값 바탕으로 MEM 단계 연산을 수행하거나, forwarding 한다. #### Execution 단계에서 얻은 Target을 MEM 단계로 전달하기 위해 사용하는 레지스터: MEM 단계 에서 atp, antp의 T/F 여부에 따라 이 값으로 분기한다. (T면 분기) Execution 단계로 forwarding된 IMM을 다시 MEM 단계 생활한 기위해 사용하는 레지스터: MEM 단계 에서 sw, be 연산을 수행하기 위해 사용하는 레지스터: MEM 단계 에서 sw, be 연산을 수행하기 위해 사용하는 레지스터: MEM 단계 에서 sw, be 연산을 수행하기 위해 사용하는 레지스터: MEM 단계 MEM 단계로 전달하기 위해 사용하는 레지스터: MEM 단계 MEM 단계로 전달하기 위해 사용하는 레지스터: MEM 단계 에서 sw, be 연산을 수행하기 위해 사용하는 레지스터: MEM 단계 MEM 단계로 전달하기 위해 사용하는 레지스터: MEM 전 MEM 단계로 제로	### ID.EX.tunction_field ### #### ###########################		Decoding 단계에서 해석한 function_field를 Execution
Decoding 단계에서 작 instruction 열 년간을 구행한 다 Decoding 단계에서 해석한 WB를 Execution 단계로 전 달하기 위해 사용하는 레지스터: 이 값은 WB까지 forwarding 되고, 이를 바탕으로 WB 단계에서 어느 레지스터에 write 할 지 결정한다. Decoding 단계에서 해석한 rs 번호를 Execution 단계로 전달하기 위해 사용하는 레지스터: 이 값 바탕으로 Hazard를 체크한다. Decoding 단계에서 해석한 rt 번호를 Execution 단계로 전달하기 위해 사용하는 레지스터: 이 값 바탕으로 Hazard를 체크한다. EX_MEM.ALU_OUT EX_MEM.ALU_OUT EX_MEM.BR_TARGET EX_MEM.BR_TA	Decoding 단계에서 각 instruction 열 현산을 구행한 다 다기로 전달하기 위해 사용하는 레지스터: 이 값은 WB까지 forwarding되고, 이를 바탕으로 WB 단계에서 어느 레지스터에 write 할 지 결정한다. Decoding 단계에서 해석한 rs 번호를 Execution 단계로 전달하기 위해 사용하는 레지스터: 이 값 바탕으로 Hazard를 제크한다. Decoding 단계에서 해석한 rt 번호를 Execution 단계로 전달하기 위해 사용하는 레지스터: 이 값 바탕으로 Hazard를 제크한다. EX_MEM.ALU_OUT EX_MEM.BR_TARGET EX_MEM.BR_TARGET EX_MEM.BR_TARGET EX_MEM.IMM E	ID FX function field	
ID.EX.WB Decoding 단계에서 해석한 WB를 Execution 단계로 전달하기 위해 사용하는 레지스터: 이 값은 WB까지 forwarding되고, 이를 바탕으로 WB 단계에서 어느 레지스터에 write할 지 결정한다. ID.EX.rs_num Decoding 단계에서 해석한 rs 번호를 Execution 단계로 전달하기 위해 사용하는 레지스터: 이 값 바탕으로 Hazard를 체크한다. ID.EX.rt_num Decoding 단계에서 해석한 rt 번호를 Execution 단계로 전달하기 위해 사용하는 레지스터: 이 값 바탕으로 Hazard를 체크한다. EX_MEM.ALU_OUT Execution 단계에서 얻은 결과를 MEM 단계로 전달하기 위해 사용하는 레지스터: 이 값 바탕으로 MEM 단계 연산을 수행하거나, forwarding 한다. EX_MEM.BR_TARGET Execution 단계에서 얻은 Target을 MEM 단계로 전달하기 위해 사용하는 레지스터: MEM 단계 에서 atp, antp의 T/F 여부에 따라 이 값으로 분기한다. (T면 분기) Execution 단계로 forwarding된 IMM을 다시 MEM 단계 로 전달하기 위해 사용하는 레지스터: MEM 단계 에서 sw, sb 연산을 수행하기 위해 사용된다. Execution 단계로 forwarding된 rt에 들어있는 값을 다시 MEM 단계로 전달하기 위해 사용하는 레지스터: MEM 단계 드러 MEM 단계 Sw, sb 연산을 수행하기 위해 사용하는 레지스터: MEM MEM 단계로 전달하기 위해 사용하는 레지스터: MEM 전 MEM 단계로 전달하기 위해 사용하는 레지스터: MEM MEM 단계로 전달하기 위해 사용하는 레지스터: MEM MEM TARE 전 MEM TA	Decoding 단계에서 해석한 WB를 Execution 단계로 전달하기 위해 사용하는 레지스터: 이 값은 WB까지 forwarding되고, 이를 바탕으로 WB 단계에서 어느 레지스터에 write할 지 결정한다. Decoding 단계에서 해석한 rs 번호를 Execution 단계로 전달하기 위해 사용하는 레지스터: 이 값 바탕으로 Hazard를 체크한다. Decoding 단계에서 해석한 rt 번호를 Execution 단계로 전달하기 위해 사용하는 레지스터: 이 값 바탕으로 Hazard를 체크한다. EX_MEM.ALU_OUT EX_MEM.BR_TARGET EX_MEM.BR_TARGET EX_MEM.IMM EX_M	ID.EAT.Italiction_Incid	_
ID.EX.WB 달하기 위해 사용하는 레지스터: 이 값은 WB까지 forwarding되고, 이를 바탕으로 WB 단계에서 어느 레지스터에 write할 지 결정한다. Decoding 단계에서 해석한 rs 번호를 Execution 단계로 전달하기 위해 사용하는 레지스터: 이 값 바탕으로 Hazard를 체크한다. ID.EX.rt_num Decoding 단계에서 해석한 rt 번호를 Execution 단계로 전달하기 위해 사용하는 레지스터: 이 값 바탕으로 Hazard를 체크한다. EX_MEM.ALU_OUT Execution 단계에서 얻은 결과를 MEM 단계로 전달하기 위해 사용하는 레지스터: 이 값 바탕으로 MEM 단계 연산은 주행하거나, forwarding 한다. EX_MEM.BR_TARGET Execution 단계에서 얻은 Target을 MEM 단계로 전달하기 위해 사용하는 레지스터: MEM 단계 에서 atp, antp의 T/F 여부에 따라 이 값으로 분기한다. (T면 분기) EXECUTION 단계로 forwarding된 IMM을 다시 MEM 단계 로 전달하기 위해 사용하는 레지스터: MEM 단계 에서 sw, sb 연산을 수행하기 위해 사용된다. EXECUTION 단계로 forwarding된 rt에 들어있는 값을 다시 MEM 단계로 전달하기 위해 사용하는 레지스터: MEM 단계 에서 sw, sb 연산을 수행하기 위해 사용하는 레지스터: MEM 대시스터: MEM 단계 단계로 전달하기 위해 사용하는 레지스터: MEM 대시스터: MEM 단계로 전달하기 위해 사용하는 레지스터: MEM 전 MEM 단계로 전달하기 위해 사용하는 레지스터: MEM MEM 단계로 전달하기 위해 사용하는 레지스터: MEM 전 MEM TMEM TMEM TMEM TMEM TMEM TMEM	ID.EX.WB 달하기 위해 사용하는 레지스터: 이 값은 WB까지 forwarding되고, 이를 바탕으로 WB 단계에서 어느 레지스터에 write 할 지 결정한다. ID.EX.rs_num Decoding 단계에서 해석한 rs 번호를 Execution 단계로 전달하기 위해 사용하는 레지스터: 이 값 바탕으로 Hazard를 체크한다. ID.EX.rt_num Decoding 단계에서 해석한 rt 번호를 Execution 단계로 전달하기 위해 사용하는 레지스터: 이 값 바탕으로 Hazard를 체크한다. EX_MEM.ALU_OUT Execution 단계에서 얻은 결과를 MEM 단계로 전달하기 위해 사용하는 레지스터: 이 값 바탕으로 MEM 단계 연산을 수행하거나, forwarding 한다. EX_MEM.BR_TARGET Execution 단계에서 얻은 Target을 MEM 단계로 전달하기 위해 사용하는 레지스터: MEM 단계 에서 app, antp의 T/F 여부에 따라 이 값으로 분기한다. (T면 분기) EX.MEM.IMM Execution 단계로 forwarding된 IMM을 다시 MEM 단계 로 전달하기 위해 사용된다. Execution 단계로 forwarding된 Ir에 들어있는 값을 다시 MEM 단계 어서 sw, sb 연산을 수행하기 위해 사용된다. Execution 단계로 forwarding된 rs 번호를 다시 MEM 단계 에서 sw, sb 연산을 수행하기 위해 사용된다. Execution 단계로 forwarding된 rs 번호를 다시 MEM 단계 시용된다. Execution 단계로 forwarding rs 번호를 다시 MEM 단계 시용된다. Execution 단계로 forwarding rs 번호를 다시 MEM 단계로 전달하기 위해 사용하는 레지스터: MEM 단계로 전달하기 위해 사용하는 레지스터: MEM 단계로 전달하기 위해 사용하는 레지스터: Original Provesting Recurrence Hazard를 체크한다.		,
### forwarding 되고, 이를 바탕으로 WB 단계에서 어느 레지으타에 write 할 지 결정한다. Decoding 단계에서 해석한 rs 번호를 Execution 단계로 전달하기 위해 사용하는 레지스터: 이 값 바탕으로 Hazard를 체크한다. Decoding 단계에서 해석한 rt 번호를 Execution 단계로 전달하기 위해 사용하는 레지스터: 이 값 바탕으로 Hazard를 체크한다. EX_MEM.ALU_OUT EX_MEM.BR_TARGET EX_MEM.BR_TARGET EX_MEM.IMM EX_MEM.IMM EX_MEM.IMM EX_MEM.IMM FY 이부에 따라 이 값으로 분기한다. (T면 분기) EXECUTION 단계로 forwarding된 IMM을 다시 MEM 단계 외사용하는 레지스터: MEM 단계 에서 sw, sb 연산을 수행하기 위해 사용하는 레지스터: MEM 단계 에서 sw, sb 연산을 수행하기 위해 사용하는 레지스터: MEM 단계 에서 Sw, sb 연산을 수행하기 위해 사용하는 레지스터: MEM 단계 에서 Sw, sb 연산을 수행하기 위해 사용하는 레지스터: MEM 단계 에서 SW, sb 연산을 수행하기 위해 사용하는 레지스터: MEM 단계 에서 SW, sb 연산을 수행하기 위해 사용하는 레지스터: MEM 단계 에서 SW, sb 연산을 수행하기 위해 사용하는 레지스터: MEM 단계 MEM 단계로 전달하기 위해 사용하는 레지스터: MEM 단계 MEM 단계로 전달하기 위해 사용하는 레지스터: MEM 단계 MEM 단계로 전달하기 위해 사용하는 레지스터: MEM MEM 단계로 전달하기로 제품	### ### ### ### #####################		
Decoding 단계에서 해석한 rs 번호를 Execution 단계로 전달하기 위해 사용하는 레지스터: 이 값 바탕으로 Hazard를 체크한다. Decoding 단계에서 해석한 rt 번호를 Execution 단계로 전달하기 위해 사용하는 레지스터: 이 값 바탕으로 Hazard를 체크한다. Decoding 단계에서 해석한 rt 번호를 Execution 단계로 전달하기 위해 사용하는 레지스터: 이 값 바탕으로 Hazard를 체크한다. Execution 단계에서 얻은 결과를 MEM 단계로 전달하기 위해 사용하는 레지스터: 이 값 바탕으로 MEM 단계 연산을 수행하거나. forwarding 한다. Execution 단계에서 얻은 Target을 MEM 단계로 전달하기 위해 사용하는 레지스터: MEM 단계 에서 atp, antp의 T/F 여부에 따라 이 값으로 분기한다. (T면 분기)	Decoding 단계에서 해석한 rs 번호를 Execution 단계로 전달하기 위해 사용하는 레지스터: 이 값 바탕으로 Hazard를 체크한다. Decoding 단계에서 해석한 rs 번호를 Execution 단계로 전달하기 위해 사용하는 레지스터: 이 값 바탕으로 Hazard를 체크한다. Decoding 단계에서 해석한 rs 번호를 Execution 단계로 전달하기 위해 사용하는 레지스터: 이 값 바탕으로 Hazard를 체크한다. EX_MEM.ALU_OUT	ID.EX.WB	
Decoding 단계에서 해석한 rs 번호를 Execution 단계로 전달하기 위해 사용하는 레지스터: 이 값 바탕으로 Hazard를 체크한다. Decoding 단계에서 해석한 rt 번호를 Execution 단계로 전달하기 위해 사용하는 레지스터: 이 값 바탕으로 Hazard를 체크한다. Execution 단계에서 얻은 결과를 MEM 단계로 전달하기 위해 사용하는 레지스터: 이 값 바탕으로 MEM 단계 연산을 수행하거나, forwarding 한다. Execution 단계에서 얻은 Target을 MEM 단계로 전달하기 위해 사용하는 레지스터: MEM 단계 연산을 수행하거나, forwarding 한다. Execution 단계에서 얻은 Target을 MEM 단계로 전달하기 위해 사용하는 레지스터: MEM 단계 에서 atp, antp의 T/F 여부에 따라 이 값으로 분기한다. (T면 분기) Execution 단계로 forwarding된 IMM을 다시 MEM 단계 로 전달하기 위해 사용하는 레지스터: MEM 단계 에서 sw. sb 연산을 수행하기 위해 사용된다. Execution 단계로 forwarding된 rt에 들어있는 값을 다시 MEM 단계로 전달하기 위해 사용하는 레지스터: MEM 대표M 단계 들어있는 값을 다시 MEM 단계로 전달하기 위해 사용하는 레지스터: MEM MEM 단계로 전달하기 위해 사용하는 레지스터: MEM	ID.EX.rs_num Decoding 단계에서 해석한 rs 번호를 Execution 단계로 전달하기 위해 사용하는 레지스터: 이 값 바탕으로 Hazard를 체크한다. ID.EX.rt_num Decoding 단계에서 해석한 rt 번호를 Execution 단계로 전달하기 위해 사용하는 레지스터: 이 값 바탕으로 Hazard를 체크한다. EX_MEM.ALU_OUT Execution 단계에서 얻은 결과를 MEM 단계로 전달하기 위해 사용하는 레지스터: 이 값 바탕으로 MEM 단계 연산을 수행하거나, forwarding 한다. EX_MEM.BR_TARGET Execution 단계에서 얻은 Target을 MEM 단계로 전달하기 위해 사용하는 레지스터: MEM 단계 에서 atp, antp의 T/F 여부에 따라 이 값으로 분기한다. (T면 분기) EX.MEM.IMM 로 전달하기 위해 사용하는 레지스터: MEM 단계 에서 sw, sb 연산을 수행하기 위해 사용된다. EX.MEM.rt Execution 단계로 forwarding된 rt에 들어있는 값을 다시 MEM 단계 에서 sw, sb 연산을 수행하기 위해 사용된다. EX.MEM.rs_num Execution 단계로 forwarding된 rs 번호를 다시 MEM 단계로 전달하기 위해 사용하는 레지스터: MEM 단계 에서 sw, sb 연산을 수행하기 위해 사용된다.		
TD.EX.rs_num 전달하기 위해 사용하는 레지스터: 이 값 바탕으로 Hazard를 체크한다. Decoding 단계에서 해석한 rt 번호를 Execution 단계로 전달하기 위해 사용하는 레지스터: 이 값 바탕으로 Hazard를 체크한다. Ex_MEM.ALU_OUT Execution 단계에서 얻은 결과를 MEM 단계로 전달하기 위해 사용하는 레지스터: 이 값 바탕으로 MEM 단계 연산을 수행하거나, forwarding 한다. Execution 단계에서 얻은 Target을 MEM 단계로 전달하기 위해 사용하는 레지스터: MEM 단계 에서 atp, antp의 T/F 여부에 따라 이 값으로 분기한다. (T면 분기) Execution 단계로 forwarding된 IMM을 다시 MEM 단계 로 전달하기 위해 사용하는 레지스터: MEM 단계 에서 sw, sb 연산을 수행하기 위해 사용된다. Execution 단계로 forwarding된 rt에 들어있는 값을 다시 MEM 단계로 전달하기 위해 사용하는 레지스터: MEM 단계 에서 sw, sb 연산을 수행하기 위해 사용하는 레지스터: MEM 단계 에서 sw, sb 연산을 수행하기 위해 사용하는 레지스터: MEM 단계 에서 sw, sb 연산을 수행하기 위해 사용하는 레지스터: MEM 단계 에서 sw, sb 연산을 수행하기 위해 사용하는 레지스터: MEM 단계로 전달하기 위해 사용하는 레지스터: MEM	TD.EX.rs_num 전달하기 위해 사용하는 레지스터: 이 값 바탕으로 Hazard를 체크한다. Decoding 단계에서 해석한 rt 번호를 Execution 단계로 전달하기 위해 사용하는 레지스터: 이 값 바탕으로 Hazard를 체크한다. Execution 단계에서 얻은 결과를 MEM 단계로 전달하기 위해 사용하는 레지스터: 이 값 바탕으로 MEM 단계 연산을 수행하거나, forwarding 한다. Execution 단계에서 얻은 Target을 MEM 단계로 전달하기 위해 사용하는 레지스터: MEM 단계 로전달하기 위해 사용하는 레지스터: MEM 단계 로전달하기 위해 사용하는 레지스터: MEM 단계 에서 atp, antp의 T/F 여부에 따라 이 값으로 분기한다. (T면 분기) Execution 단계로 forwarding된 IMM을 다시 MEM 단계 로전달하기 위해 사용하는 레지스터: MEM 단계 에서 sw, sb 연산을 수행하기 위해 사용된다. Execution 단계로 전달하기 위해 사용하는 레지스터: MEM 단계 에서 sw, sb 연산을 수행하기 위해 사용된다. Execution 단계로 forwarding된 rs 번호를 다시 MEM 단계 에서 sw, sb 연산을 수행하기 위해 사용된다. Execution 단계로 forwarding된 rs 번호를 다시 MEM 단계 에서 sw, sb 연산을 수행하기 위해 사용된다. Execution 단계로 forwarding된 rs 번호를 다시 MEM 단계 로전달하기 위해 사용하는 레지스터: 이 값 바탕으로 Hazard를 체크한다.		Decoding 단계에서 해석한 rs 번호를 Execution 단계로
Decoding 단계에서 해석한 rt 번호를 Execution 단계로 전달하기 위해 사용하는 레지스터: 이 값 바탕으로 Hazard를 체크한다. EX_MEM.ALU_OUT Execution 단계에서 얻은 결과를 MEM 단계로 전달하기 위해 사용하는 레지스터: 이 값 바탕으로 MEM 단계 연산을 수행하거나, forwarding 한다. EX_MEM.BR_TARGET Execution 단계에서 얻은 Target을 MEM 단계로 전달하기 위해 사용하는 레지스터: MEM 단계 에서 atp, antp의 T/F 여부에 따라 이 값으로 분기한다. (T면 분기) EXECUTION 단계로 forwarding된 IMM을 다시 MEM 단계로 전달하기 위해 사용하는 레지스터: MEM 단계에서 sw, sb 연산을 수행하기 위해 사용된다. EXECUTION 단계로 forwarding된 rt에 들어있는 값을 다시 MEM 단계로 전달하기 위해 사용하는 레지스터: MEM EXECUTION 단계로 forwarding된 rt에 들어있는 값을 다시 MEM 단계로 전달하기 위해 사용하는 레지스터: MEM	Decoding 단계에서 해석한 rt 번호를 Execution 단계로 전달하기 위해 사용하는 레지스터: 이 값 바탕으로 Hazard를 체크한다. EX_MEM.ALU_OUT Execution 단계에서 얻은 결과를 MEM 단계로 전달하기 위해 사용하는 레지스터: 이 값 바탕으로 MEM 단계 연산을 수행하거나, forwarding 한다. EX_MEM.BR_TARGET Execution 단계에서 얻은 Target을 MEM 단계로 전달하기 위해 사용하는 레지스터: MEM 단계 에서 atp, antp의 T/F 여부에 따라 이 값으로 분기한다. (T면 분기) EXECUTION 단계로 forwarding된 IMM을 다시 MEM 단계 에서 sw, sb 연산을 수행하기 위해 사용된다. Execution 단계로 forwarding된 rt에 들어있는 값을 다시 MEM 단계 에서 sw, sb 연산을 수행하기 위해 사용된다. EX.MEM.rt Execution 단계로 forwarding된 rt에 들어있는 값을 다시 MEM 단계 에서 sw, sb 연산을 수행하기 위해 사용된다. EX.MEM.rs_num Execution 단계로 forwarding된 rs 번호를 다시 MEM 단계로 전달하기 위해 사용된다. Execution 단계로 forwarding된 rs 번호를 다시 MEM 단계로 전달하기 위해 사용된다. Execution 단계로 forwarding된 rs 번호를 다시 MEM 단계로 전달하기 위해 사용하는 레지스터: 이 값 바탕으로 Hazard를 체크한다.	ID.EX.rs_num	전달하기 위해 사용하는 레지스터: <i>이 값 바탕으로</i>
전달하기 위해 사용하는 레지스터: 이 값 바탕으로 Hazard를 체크한다. EX_MEM.ALU_OUT Execution 단계에서 얻은 결과를 MEM 단계로 전달하기 위해 사용하는 레지스터: 이 값 바탕으로 MEM 단계 연산을 수행하거나, forwarding 한다. EX_MEM.BR_TARGET Execution 단계에서 얻은 Target을 MEM 단계로 전달하기 위해 사용하는 레지스터: MEM 단계 에서 atp, antp의 T/F 여부에 따라 이 값으로 분기한다. (T면 분기) EX.MEM.IMM Execution 단계로 forwarding된 IMM을 다시 MEM 단계 로 전달하기 위해 사용하는 레지스터: MEM 단계 에서 sw, sb 연산을 수행하기 위해 사용된다. EX.MEM.rt Execution 단계로 전달하기 위해 사용하는 레지스터: MEM	ID.EX.rt_num 전달하기 위해 사용하는 레지스터: 이 값 바탕으로 Hazard를 체크한다. Execution 단계에서 얻은 결과를 MEM 단계로 전달하기 위해 사용하는 레지스터: 이 값 바탕으로 MEM 단계 연 산을 수행하거나, forwarding 한다. Execution 단계에서 얻은 Target을 MEM 단계로 전달하기 위해 사용하는 레지스터: MEM 단계 에서 atp, antp의 T/F 여부에 따라 이 값으로 분기한다. (T면 분기) Execution 단계로 forwarding된 IMM을 다시 MEM 단계 로 전달하기 위해 사용하는 레지스터: MEM 단계 에서 sw, sb 연산을 수행하기 위해 사용된다. Execution 단계로 전달하기 위해 사용하는 레지스터: MEM 단계 에서 sw, sb 연산을 수행하기 위해 사용된다. Execution 단계로 전달하기 위해 사용하는 레지스터: MEM 단계 에서 sw, sb 연산을 수행하기 위해 사용된다. Execution 단계로 forwarding된 rt에 들어있는 값을 다시 MEM 단계 에서 sw, sb 연산을 수행하기 위해 사용된다. Execution 단계로 forwarding된 rs 번호를 다시 MEM 단계 로 전달하기 위해 사용하는 레지스터: 이 값 바탕으로 Hazard를 체크한다.		
### Hazard를 체크한다: Execution 단계에서 얻은 결과를 MEM 단계로 전달하기 위해 사용하는 레지스터: 이 값 바탕으로 MEM 단계 연 산을 수행하거나, forwarding 한다. Execution 단계에서 얻은 Target을 MEM 단계로 전달하기 위해 사용하는 레지스터: MEM 단계 에서 atp, antp의 T/F 여부에 따라 이 값으로 분기한다. (T면 분기) Execution 단계로 forwarding된 IMM을 다시 MEM 단계 로 전달하기 위해 사용하는 레지스터: MEM 단계 에서 sw, sb 연산을 수행하기 위해 사용된다. Execution 단계로 forwarding된 rt에 들어있는 값을 다시 MEM 단계로 전달하기 위해 사용하는 레지스터: MEM	Hazard를 체크한다. EX_MEM.ALU_OUT Execution 단계에서 얻은 결과를 MEM 단계로 전달하기 위해 사용하는 레지스터: 이 값 바탕으로 MEM 단계 연산을 수행하거나, forwarding 한다. EX_MEM.BR_TARGET Execution 단계에서 얻은 Target을 MEM 단계로 전달하기 위해 사용하는 레지스터: MEM 단계에서 atp, antp의 T/F 여부에 따라 이 값으로 분기한다. (T면 분기) EX.MEM.IMM Execution 단계로 forwarding된 IMM을 다시 MEM 단계 로 전달하기 위해 사용하는 레지스터: MEM 단계에서 sw, sb 연산을 수행하기 위해 사용된다. EX.MEM.rt Execution 단계로 forwarding된 rt에 들어있는 값을 다시 MEM 단계 에서 sw, sb 연산을 수행하기 위해 사용된다. EX.MEM.rs_num Execution 단계로 forwarding된 rs 번호를 다시 MEM 단계로 전달하기 위해 사용하는 레지스터: 이 값 바탕으로 Hazard를 체크한다.	W 777 .	
EX_MEM.ALU_OUT Execution 단계에서 얻은 결과를 MEM 단계로 전달하기 위해 사용하는 레지스터: 이 값 바탕으로 MEM 단계 연산을 수행하거나, forwarding 한다. EX_MEM.BR_TARGET Execution 단계에서 얻은 Target을 MEM 단계로 전달하기 위해 사용하는 레지스터: MEM 단계 에서 atp, antp의 T/F 여부에 따라 이 값으로 분기한다. (T면 분기) EX.MEM.IMM Execution 단계로 forwarding된 IMM을 다시 MEM 단계로 전달하기 위해 사용하는 레지스터: MEM 단계에서 sw, sb 연산을 수행하기 위해 사용된다. EX.MEM.rt Execution 단계로 forwarding된 rt에 들어있는 값을 다시 MEM 단계로 전달하기 위해 사용하는 레지스터: MEM	EX_MEM.ALU_OUT Execution 단계에서 얻은 결과를 MEM 단계로 전달하기 위해 사용하는 레지스터: 이 값 바탕으로 MEM 단계 연산을 수행하거나, forwarding 한다. EX_MEM.BR_TARGET Execution 단계에서 얻은 Target을 MEM 단계로 전달하기 위해 사용하는 레지스터: MEM 단계에서 atp, antp의 T/F 여부에 따라 이 값으로 분기한다. (T면 분기) EX.MEM.IMM Execution 단계로 forwarding된 IMM을 다시 MEM 단계로 전달하기 위해 사용하는 레지스터: MEM 단계에서 sw, sb 연산을 수행하기 위해 사용된다. EX.MEM.rt Execution 단계로 forwarding된 rt에 들어있는 값을 다시 MEM 단계로 전달하기 위해 사용하는 레지스터: MEM 단계 에서 sw, sb 연산을 수행하기 위해 사용된다. EX.MEM.rs_num Execution 단계로 forwarding된 rs 번호를 다시 MEM 단계로 전달하기 위해 사용하는 레지스터: 이 값 바탕으로 Hazard를 체크한다.	ID.EX.rt_num	
EX_MEM.ALU_OUT 위해 사용하는 레지스터: 이 값 바탕으로 MEM 단계 연산을 수행하거나, forwarding 한다. Execution 단계에서 얻은 Target을 MEM 단계로 전달하기 위해 사용하는 레지스터: MEM 단계 에서 atp, antp의 T/F 여부에 따라 이 값으로 분기한다. (T면 분기) Execution 단계로 forwarding된 IMM을 다시 MEM 단계로 전달하기 위해 사용하는 레지스터: MEM 단계에서 sw, sb 연산을 수행하기 위해 사용된다. Execution 단계로 forwarding된 rt에 들어있는 값을 다시 MEM 단계로 전달하기 위해 사용하는 레지스터: MEM	EX_MEM.ALU_OUT 위해 사용하는 레지스터: 이 값 바탕으로 MEM 단계 연산을 수행하거나, forwarding 한다. Execution 단계에서 얻은 Target을 MEM 단계로 전달하기 위해 사용하는 레지스터: MEM 단계 에서 atp, antp의 T/F 여부에 따라 이 값으로 분기한다. (T면 분기) EX.MEM.IMM Execution 단계로 forwarding된 IMM을 다시 MEM 단계로 전달하기 위해 사용하는 레지스터: MEM 단계에서 sw, sb 연산을 수행하기 위해 사용된다. EX.MEM.rt Execution 단계로 forwarding된 rt에 들어있는 값을 다시 MEM 단계로 전달하기 위해 사용하는 레지스터: MEM 단계 에서 sw, sb 연산을 수행하기 위해 사용된다. EX.MEM.rs_num Execution 단계로 forwarding된 rs 번호를 다시 MEM 단계로 전달하기 위해 사용하는 레지스터: 이 값 바탕으로 Hazard를 체크한다.		
산을 수행하거나, forwarding 한다. Execution 단계에서 얻은 Target을 MEM 단계로 전달하기 위해 사용하는 레지스터: MEM 단계 에서 atp, antp의 T/F 여부에 따라 이 값으로 분기한다. (T면 분기) Execution 단계로 forwarding된 IMM을 다시 MEM 단계로 전달하기 위해 사용하는 레지스터: MEM 단계에서 Sw, sb 연산을 수행하기 위해 사용된다. Execution 단계로 forwarding된 rt에 들어있는 값을 다시 MEM 단계로 전달하기 위해 사용하는 레지스터: MEM	산을 수행하거나, forwarding 한다. EX_MEM.BR_TARGET Execution 단계에서 얻은 Target을 MEM 단계로 전달하기 위해 사용하는 레지스터: MEM 단계 에서 atp, antp의 T/F 여부에 따라 이 값으로 분기한다. (T면 분기) EX.MEM.IMM Execution 단계로 forwarding된 IMM을 다시 MEM 단계로 전달하기 위해 사용하는 레지스터: MEM 단계에서 sw, sb 연산을 수행하기 위해 사용된다. EX.MEM.rt Execution 단계로 forwarding된 rt에 들어있는 값을 다시 MEM 단계로 전달하기 위해 사용하는 레지스터: MEM 단계 에서 sw, sb 연산을 수행하기 위해 사용된다. EX.MEM.rs_num Execution 단계로 forwarding된 rs 번호를 다시 MEM 단계로 전달하기 위해 사용하는 레지스터: 이 값 바탕으로 Hazard를 체크한다.	EX MEM.ALU OUT	
EX_MEM.BR_TARGET Execution 단계에서 얻은 Target을 MEM 단계로 전달하기 위해 사용하는 레지스터: MEM 단계 에서 atp, antp의 T/F 여부에 따라 이 값으로 분기한다. (T면 분기) EX_MEM.IMM Execution 단계로 forwarding된 IMM을 다시 MEM 단계로 전달하기 위해 사용하는 레지스터: MEM 단계에서 Sw., sb 연산을 수행하기 위해 사용된다. EX_MEM.rt Execution 단계로 forwarding된 rt에 들어있는 값을 다시 MEM 단계로 전달하기 위해 사용하는 레지스터: MEM	EX_MEM.BR_TARGET Execution 단계에서 얻은 Target을 MEM 단계로 전달하기 위해 사용하는 레지스터: MEM 단계 에서 atp, antp의 T/F 여부에 따라 이 값으로 분기한다. (T면 분기) EX.MEM.IMM Execution 단계로 forwarding된 IMM을 다시 MEM 단계로 전달하기 위해 사용하는 레지스터: MEM 단계에서 sw, sb 연산을 수행하기 위해 사용된다. EX.MEM.rt Execution 단계로 forwarding된 rt에 들어있는 값을 다시 MEM 단계로 전달하기 위해 사용하는 레지스터: MEM 단계 에서 sw, sb 연산을 수행하기 위해 사용된다. EX.MEM.rs_num Execution 단계로 forwarding된 rs 번호를 다시 MEM 단계로 전달하기 위해 사용하는 레지스터: 이 값 바탕으로 Hazard를 체크한다.		
EX_MEM.BR_TARGET 기 위해 사용하는 레지스터: MEM 단계 에서 atp, antp의 T/F 여부에 따라 이 값으로 분기한다. (T면 분기) Execution 단계로 forwarding된 IMM을 다시 MEM 단계로 전달하기 위해 사용하는 레지스터: MEM 단계에서 Sw, sb 연산을 수행하기 위해 사용된다. Execution 단계로 forwarding된 rt에 들어있는 값을 다시 MEM 단계로 전달하기 위해 사용하는 레지스터: MEM	EX_MEM.BR_TARGET 기 위해 사용하는 레지스터: MEM 단계 에서 atp, antp의 T/F 여부에 따라 이 값으로 분기한다. (T면 분기) Execution 단계로 forwarding된 IMM을 다시 MEM 단계 로 전달하기 위해 사용하는 레지스터: MEM 단계 에서 sw, sb 연산을 수행하기 위해 사용된다. Execution 단계로 forwarding된 rt에 들어있는 값을 다시 MEM 단계로 전달하기 위해 사용된다. Execution 단계로 전달하기 위해 사용하는 레지스터: MEM 단계 에서 sw, sb 연산을 수행하기 위해 사용된다. Execution 단계로 forwarding된 rs 번호를 다시 MEM 단계로 전달하기 위해 사용하는 레지스터: 이 값 바탕으로 Hazard를 체크한다.		Execution 단계에서 얻은 Target을 MEM 단계로 전달하
EX.MEM.IMM EX.ME	EX.MEM.IMM Execution 단계로 forwarding된 IMM을 다시 MEM 단계 로 전달하기 위해 사용하는 레지스터: MEM 단계 에서 sw, sb 연산을 수행하기 위해 사용된다. EX.MEM.rt Execution 단계로 forwarding된 rt에 들어있는 값을 다시 MEM 단계로 전달하기 위해 사용하는 레지스터: MEM 단계 에서 sw, sb 연산을 수행하기 위해 사용된다. EX.MEM.rs_num Execution 단계로 forwarding된 rs 번호를 다시 MEM 단계로 전달하기 위해 사용하는 레지스터: 이 값 바탕으로 Hazard를 체크한다.	EX_MEM.BR_TARGET	기 위해 사용하는 레지스터: MEM 단계 에서 atp, antp의
EX.MEM.IMM 로 전달하기 위해 사용하는 레지스터: MEM 단계 에서 sw, sb 연산을 수행하기 위해 사용된다. Execution 단계로 forwarding된 rt에 들어있는 값을 다시 MEM 단계로 전달하기 위해 사용하는 레지스터: MEM	EX.MEM.IMM 로 전달하기 위해 사용하는 레지스터: MEM 단계 에서 sw, sb 연산을 수행하기 위해 사용된다. Execution 단계로 forwarding된 rt에 들어있는 값을 다시 MEM 단계로 전달하기 위해 사용하는 레지스터: MEM 단계 에서 sw, sb 연산을 수행하기 위해 사용된다. EX.MEM.rs_num Execution 단계로 forwarding된 rs 번호를 다시 MEM 단계로 전달하기 위해 사용하는 레지스터: 이 값 바탕으로 Hazard를 체크한다.		
sw, sb 연산을 수행하기 위해 사용된다. Execution 단계로 forwarding된 rt에 들어있는 값을 다시 EX.MEM.rt MEM 단계로 전달하기 위해 사용하는 레지스터: MEM	sw, sb 연산을 수행하기 위해 사용된다. EXECUTION 단계로 forwarding된 rt에 들어있는 값을 다시 MEM 단계로 전달하기 위해 사용하는 레지스터: MEM 단계 에서 sw, sb 연산을 수행하기 위해 사용된다. EXECUTION 단계로 forwarding된 rs 번호를 다시 MEM 단계로 전달하기 위해 사용하는 레지스터: 이 값 바탕으로 Hazard를 체크한다.		
Execution 단계로 forwarding된 rt에 들어있는 값을 다시 EX.MEM.rt MEM 단계로 전달하기 위해 사용하는 레지스터: <i>MEM</i>	EX.MEM.rt Execution 단계로 forwarding된 rt에 들어있는 값을 다시 MEM 단계로 전달하기 위해 사용하는 레지스터: MEM 단계 에서 sw, sb 연산을 수행하기 위해 사용된다. Execution 단계로 forwarding된 rs 번호를 다시 MEM 단계로 전달하기 위해 사용하는 레지스터: 이 값 바탕으로 Hazard를 체크한다.	EX.MEM.IMM	
EX.MEM.rt MEM 단계로 전달하기 위해 사용하는 레지스터: MEM	EX.MEM.rt MEM 단계로 전달하기 위해 사용하는 레지스터: MEM 단계 에서 sw, sb 연산을 수행하기 위해 사용된다. Execution 단계로 forwarding된 rs 번호를 다시 MEM 단계로 전달하기 위해 사용하는 레지스터: 이 값 바탕으로 Hazard를 체크한다.		
	단계 에서 sw, sb 연산을 수행하기 위해 사용된다. Execution 단계로 forwarding된 rs 번호를 다시 MEM 단계로 전달하기 위해 사용하는 레지스터: 이 값 바탕으로 Hazard를 체크한다.	EX MEM rt	
<i>단계 에서 sw. sb 연산을 수행하기 위해 사용되다.</i>	Execution 단계로 forwarding된 rs 번호를 다시 MEM 단 EX.MEM.rs_num 계로 전달하기 위해 사용하는 레지스터: 이 값 바탕으로 Hazard를 체크한다.	LANITEATINE	
	EX.MEM.rs_num 계로 전달하기 위해 사용하는 레지스터: 이 값 바탕으로 Hazard를 체크한다.		
	Hazard를 체크한다.	EX.MEM.rs_num	
Hazard를 체크한다.	EX MEM rt num Fxecution 단계로 forwarding되 rt 버형를 다시 MEM 다		
EX MEM rt num Fxecution 단계로 forwarding되 rt 버형를 다치 MEM 다	Execution Calls forwarding to Case May MEM C	EX.MEM.rt_num	Execution 단계로 forwarding된 rt 번호를 다시 MEM 단

	계로 전달하기 위해 사용하는 레지스터: <i>이 값 바탕으로</i>
	Hazard를 체크한다.
EX.MEM.WB	Execution 단계로 forwarding된 WB를 다시 MEM 단계로 전달하기 위해 사용하는 레지스터: 이 값은 WB까지 forwarding되고, 이를 바탕으로 WB 단계에서 어느 레지스터에 write할지 결정한다.
EX.MEM.NPC:	Execution 단계의 PC값 다시 MEM 단계로 전달하기 위해 사용하는 레지스터: 이 PC값은 bne, beq target 연산 등에 사용된다.
MEM_WB.ALU_OUT	Mem 단계로 forwarding된 연산 결과를 다시 WB 단계로 전달하기 위해 사용하는 레지스터: 이 값 바탕으로 WB 단계에서 reg write를 한다.
MEM_WB.MEM_OUT	Mem 단계에서 얻은 결과를 WB 단계로 전달하기 위해 사용하는 레지스터: 이 값 바탕으로 WB 단계에서 reg write를 한다
MEM_WB.WB	Mem 단계로 forwarding된 WB를 다시 WB 단계로 전달 하기 위해 사용하는 레지스터: 이 값은 어느 레지스터에 write 할 지 결정한다.

위 표를 보면 ID_EX.rs, ID.EX.rt도 있고, ID.EX.rs_num, ID.EX.rt_num도 있는 것을 확인할 수 있다.

: reg num은 Hazard를 판단하기 위해, reg value는 forwarding을 위해서 사용하였다. forwarding을 하면 reg value state reg에 값이 전달되게 되고, 이를 바탕으로 EX 단계에서 업데이트 된 값으로 연산을 수행한다.

2. Control signal

이번 과제에서는 총 10개의 control signal을 사용하였다. 각 control signal의 역할은 밑 표와 같다. (-1은 X 표시와 같은 역할을 한다)

	RegD	ALUS	Brc	MemRe	MemWr	RegWri	MemToR	PCS	WordByteCh	BeqBneChec
	st	rc	h	ad	ite	te	eg	rc	eck	ker
R	1	0	0	0	0	1	0	0	-1	-1
format										
(jr X)										
R	1	0	0	0	0	0	0	0	-1	-1
format										
(jr)										
sw	-1	1	0	0	1	0	-1	0	1	-1
sb	-1	1	0	0	1	0	-1	0	0	-1
lw	0	1	0	1	0	1	0	0	1	-1
lb	0	1	0	1	0	1	0	0	0	-1
beq	-1	0	1	0	0	0	-1	1	-1	1
bne	-1	0	1	0	0	0	-1	1	-1	0
j	-1	0	0	0	0	0	0	1	-1	-1
jal	-1	0	0	0	0	0	0	1	-1	-1
else	0	0	0	0	0	1	0	0	0	-1

이 때 새롭게 추가한 WordByteCheck는 lw, sw, 그리고 lb, sb를 구분하기 위한 신호이고, BeqBneChecker는 Beq와 Bne를 구분하기 위한 신호이다.

3. Hazard 처리

A. Data Harzard

EX/MEM to EX forwarding: MEM단계에서 사용하는 reg를 EX단계의 instruction에서 사용한

```
i. // DataHazard: EX/MEM to EX Forwarding

if (control_signal[5] == 1) {

if (ReadStateReg(16) == ReadStateReg(20)) { // 16:

EX_MEM.WB, 4: ID_EX.rs & WB 일 때

iv. // WB 될 rt, 또는 rd 가 rs 와 같다면 -> ALU 값을

forwarding

v. WriteStateReg(4, ReadStateReg(11));

vi. // Forwarding

vii. }

if (ReadStateReg(16) == ReadStateReg(21)) { // 16:

EX_MEM.WB, 5: ID_EX.rt & WB 일 때

ix. // WB 될 rt, 또는 rd 가 rt 와 같다면

x. WriteStateReg(5, ReadStateReg(11));

xi. // Forwarding

xii. }

iii. }
```

MEM/WB to EX forwarding: WB단계에서 사용하는 reg를 EX단계의 instruction에서 사용한다면, stall 후 forwarding해준다.

```
DataHazard: MEM/WB to EX Forwarding
   if(control_signal[5] == 1) {
       if (ReadStateReg(17) == ReadStateReg(20)) { // 17:
MEM_WB.WB, WB 일 때
           if (ReadStateReg(16) != ReadStateReg(20)) { // 20은 굳이
forwarding 될 필요 없는 레지스터 -> state reg 는 다음 cycle 에 값 바뀌는
              WriteStateReg(4, ReadStateReg(13));
               // Forwarding
           }
       if (ReadStateReg(17) == ReadStateReg(21)) { // 17:
MEM_WB.WB, WB 일 때
           if (ReadStateReg(16) != ReadStateReg(21)) {
              WriteStateReg(5, ReadStateReg(13));
               // Forwarding
           }
       }
```

MEM/WB to MEM forwarding (only lw, lb to sw, sb): load 뒤에 store가 오는 경우 forwarding 을 해준다.

```
if(control_signal[3] == 1) {
    if (ReadStateReg(17) == ReadStateReg(23)) { // rt 같을 때
        WriteStateReg(19, ReadStateReg(14));
```

```
// Forwarding
}
```

Load use stall: load에서 사용하는 reg를 그 다음 instruction에서 사용한다면, stall 후 forwarding해준다. 단 PC.write, ID/IF.write는 유지하여 stall이후에 이 전의 instruction이 재시작될 수 있게 한다.

ex. {inst4| inst3|lw| inst2| inst1} ... stall ... {inst4| inst3|| lw| inst2}

```
// DataHazard: load use
    if (ReadStateReg(3) == 0x23 || ReadStateReg(3) == 0x20) { // Ex
    stage load 명령어가 있을 때
        if ((((line_32bit>>26)&0x3F) != 0x2b) &&
        (((line_32bit>>26)&0x3F) != 0x28)) {
            if ((ReadStateReg(20) == ((line_32bit>>21)&0x1F)) ||
        (ReadStateReg(21) == ((line_32bit>>16)&0x1F))) { // load use 의 rt 와
        ID stage 명령의 rs, 또는 rt 가 같다면
            pipelining_control[1] = 0; // flush => 왜 1 index 를
        nope 하는지는 '파이프라인 미루기' 부분 참고
            Load_use_Hazard = 1; // IF stage 유지 신호
            Load_use_Hazard_forwarding = 1; // forwarding 신호
            Load_use_Hazard_pipe = 1;
            line_buffer = line_32bit; // ID stage 유지
        }
    }
}
```

B. Control Hazard

conditional branch

(ID stage): atp, antp에 따라 Always Taken, Always not Taken한다. 만약 Always Taken이 라면한 사이클 stall 해준다.

ex.

```
atp: {|brch|inst3|inst2|inst1}
antp: {inst4| brch|inst3|inst2|inst1}
```

```
// Control Hazard: Brch
if ((((line_32bit>>26)&0x3F) == 4) || (((line_32bit>>26)&0x3F)
== 5)) {
    int offset = (line_32bit)&0xFFFF;
    if (atp == 1) { // Always Taken
        pipelining[0] = 0; // 1 cycle stall
        PC_buffer = ReadStateReg(1)+4+offset*4;
    }
    else { // Always not Taken
        // 그냥 진행
```

```
(MEM stage): 예측 결과가 맞다면 그대로 진행, 만약 틀리다면 세 사이클 stall 후 PC buffer
에 있는 instruction을 실행 시킨다.
ex.
   atp(true): {inst6|inst5| |brch|inst3}
   atp(false): {| | |brch|inst3}
   antp(true): {inst6|inst5|inst4|brch|inst3}
   antp(false): {| | |brch|inst3}
if (control signal[2] == 1) {
        // 이 때는 conditional branch (=beq, bne)
        if (control_signal[9] == 1) { // beq
            if (ReadReg(ReadStateReg(22)) ==
ReadReg(ReadStateReg(23))) { // 분기 0
                if (atp == 1) { // 맞춤
                else { // 틀림
                    pipelining[0] = 0;
                    pipelining[1] = 0;
                    pipelining[2] = 0;
                    PC_buffer = ReadStateReg(11);
            else { // 분기 X
                if (atp == 1) { // 틀림
                    pipelining[0] = 0;
                    pipelining[1] = 0;
                    pipelining[2] = 0;
                    PC_buffer = ReadStateReg(24)+4;
                else {
            }
        if (control_signal[9] == 0) { // bne
            if (ReadReg(ReadStateReg(22)) ==
ReadReg(ReadStateReg(23))) { // 분기 X
                if (atp == 1) { // 틀림
                    pipelining[0] = 0;
                    pipelining[1] = 0;
                    pipelining[2] = 0;
```

unconditional branch: 한 사이클 stall 후, PC_buffer에 있는 instruction을 실행 시킨다. ex. {inst4|j|inst3|inst2|inst1}

C. Structural Hazard

: 이번 과제에서 Structural Hazard는 없다고 가정한다.

4. Code flow

코드는

- 1) main 함수를 통한 파일 read 및 input 해석
- 2) Emulator 함수를 통한 input 요구에 맞는 실행 (ex. -n 옵션이 있다면 -n 옵션 뒤에 오는 숫자 만큼 실행)
 - 3) Emulator 함수를 통해 각 stage 실행
 - 5) PC 값 업데이트 (만약 brch, 또는 j라면 PC buffer 값 읽어서 업데이트, 아니라면 PC+4로 업데이트)
- 6) Emulator 함수를 통한 input 요구에 맞는 출력 제공 (-d 옵션이 있다면 매 instruction마다 출력, 없다면 대기)
 - 7) 다시 3),4),5),6) 반복
- 8) PC 값이 non을 가르키고, PC_buffer에도 아무런 값이 없다면 Emulator 종료 후 input 요구에 맞는 출력 제공

과정으로 실행된다.

이 때 프로그램은 {||||}일 때, 즉 파이프 라인이 모두 비었을 때 completion 되도록 설계 하였다.

5. Results

결과는 다음과 같다.

1. -p option with sample.o

./homework3 -p -atp sample.o
===== Completion cycle: 25 =====
Current Pipiline PC State:
{ }
Current register values:
PC: 0x400050
Registers:
R0: 0x0
R1: 0x0
R2: 0xa
R3: 0x800
R4: 0x1000000c
R5: 0x4d2
R6: 0x4d20000
R7: 0x4d2270f
R8: 0x4d2230f
R9: 0xfffff3ff

```
R10: 0x4ff
R11: 0x269000
R12: 0x4d2000
R13: 0x0
R14: 0x4
R15: 0xfffffb01
R16: 0x0
R17: 0x640000
R18: 0x0
R19: 0x0
R20: 0x0
R21: 0x0
R22: 0x0
R23: 0x0
R24: 0x0
R25: 0x0
R26: 0x0
R27: 0x0
R28: 0x0
R29: 0x0
R30: 0x0
R31: 0x0
```

2. -p, -atp option with sample2.o

```
./homework3 -p -atp sample2.o
==== Completion cycle: 63 =====
Current Pipiline PC State:
{||||}
Current register values:
PC: 0x400030
Registers:
R0: 0x0
R1: 0x1
R2: 0x0
R3: 0xf
R4: 0xf
R5: 0x0
R6: 0x0
R7: 0x0
R8: 0x10000000
R9: 0x5
```

```
R10: 0x0
R11: 0x0
R12: 0x0
R13: 0x0
R14: 0x0
R15: 0x0
R16: 0x0
R17: 0x0
R18: 0x0
R19: 0x0
R20: 0x0
R21: 0x0
R22: 0x0
R23: 0x0
R24: 0x0
R25: 0x0
R26: 0x0
R27: 0x0
R28: 0x0
R29: 0x0
R30: 0x0
R31: 0x400010
```

3. -p, -antp option with sample2.o

```
./homework3 -p -antp sample2.o
==== Completion cycle: 50 =====
Current Pipiline PC State:
{||||}
Current register values:
PC: 0x400030
Registers:
R0: 0x0
R1: 0x1
R2: 0x0
R3: 0xf
R4: 0xf
R5: 0x0
R6: 0x0
R7: 0x0
R8: 0x10000000
R9: 0x5
```

```
R10: 0x0
R11: 0x0
R12: 0x0
R13: 0x0
R14: 0x0
R15: 0x0
R16: 0x0
R17: 0x0
R18: 0x0
R19: 0x0
R20: 0x0
R21: 0x0
R22: 0x0
R23: 0x0
R24: 0x0
R25: 0x0
R26: 0x0
R27: 0x0
R28: 0x0
R29: 0x0
R30: 0x0
R31: 0x400010
```