Computer Arch

Homework 2:

Report of Single Cycle

Name : SeokBeom Lim (임석범)

Department : Mobile System Engineering

Student ID : 32203743

Table of contents

서론

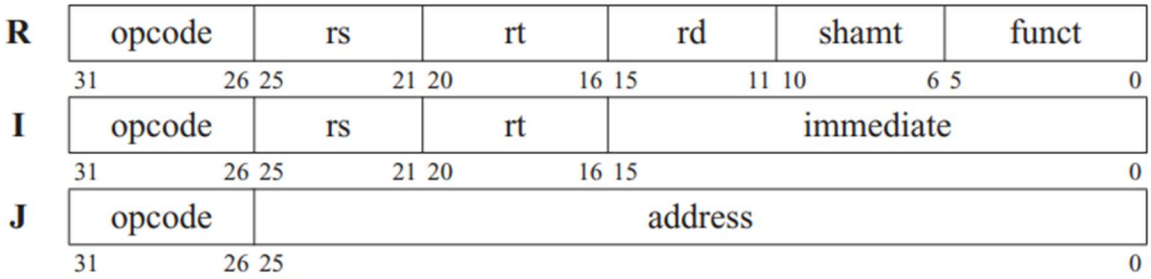
Von Neumann archictecture 는 현재에도 많은 컴퓨터에 사용되는 구조이다. Von Neumann archictecture 는 명령어를 메모리에 저장하여 사용하는 Stored Program 의 특징을 가지며, Instruction 을 순차적으로(Sequential) 처리하는 특징을 가진다. 이러한 특징들은 기존의 계산기와는 다른 “컴퓨터” 라는 개념을 보다 구체화 하였으며, 발전할 수 있는 영역을 넓혀주었다. 지난 과제 에서 simple calculator 를 통해 명령어를 통한 계산을 하는 프로그램을 설계하였지만, 해당 프로그램은 Microprocessor 가 구현되지 않아 간단한 연산만 처리할 수 있었다. 이번 과제에서는 Single Cycle 을 구현하여 복잡한 연산을 처리할 수 있는 MIPS Simulator 를 만들고자 한다.

본론

MIPS ISA 개념)

MIPS(Microprocessor without Interlocked Pipelined Stages) 는 RISC 명령어 집합 체계로 간단한 명령어 체계를 가진다. 명령어의 수가 CISC 에 비해 적고 길이와 포맷이 일정하게 설계되었다. 이러한 설계는 비교적 HW 설계의 구현 난이도가 내려가며, 설계 구조를 단순하게 만들 수 있다. MIPS ISA 는 3 가지 type 의 명령어를 갖는다. 명령어의 포맷은 각각 R-type, I-type, J-type 이며 3 개의 타입은 모두 opcode 라는 명령 코드를 가진다. opcode 를 통해 어떤 명령어를 수행하는지 분류하여 실행되며, opcode 를 제외한 나머지 부분은 포맷 타입에 따라 달라지며 모든 명령어는 같은 크기의 길이를 가진다. 이를 그림으로 표기하면 다음과 같다.

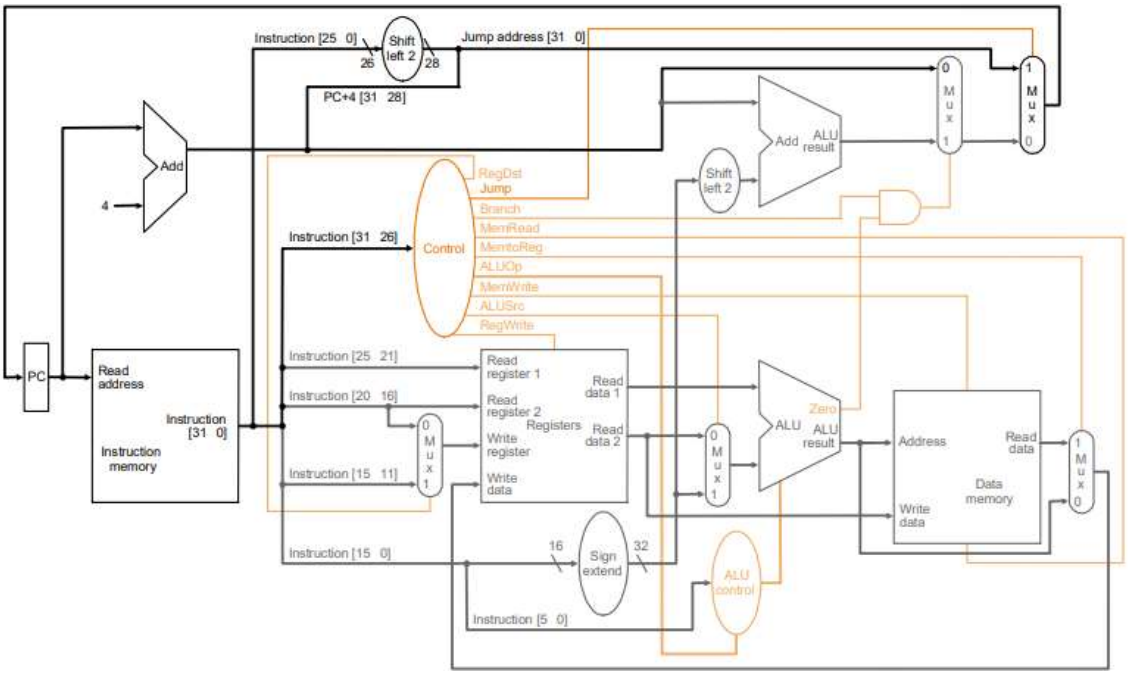
BASIC INSTRUCTION FORMATS



위 그림과 같이 3 가지 타입에 따라, 수행하는 명령어와 명령어의 대상이 달라진다. MIPS ISA 의 주요 특징 중 하나는 적재 명령어(LW)와 저장 명령어(SW)를 통해서만 데이터 메모리에 접근이 가능하다. 이러한 ISA 특징은 컴파일러가 컴파일을 최적화 하기 쉬운 구조가 제공된다.

Single Cycle 개념)

Single cycle Process 는 명령어의 모든 단계를 하나의 사이클 내에서 완료하는 구조를 가진다. 모든 구조란 5 단계로 이루어져 있으며, 각 단계는 Fetch, Decode, Execute, Memory access, WriteBack 단계이다. 위 5 단계를 단일 클럭 사이클내에서 명령어를 수행 할 수 있게 만든 구조가 Single cycle micro architecture 이다. 위 단계를 구체화 한 그림(Data Path)은 다음과 같다.



위 그림의 모든 신호들이 하나의 사이클 안에서 완료된다. 하나의 사이클 안에서 실행되는 단계는 다음과 같다.

1. 명령어 인출 (Instruction fetch, IF)

2. 명령어 해석 (Instruction Decode, ID)

3. 실행 (Execute, EX)

4. 메모리 접근 (Memory Access, MEM)

5. 라이트 백 (Write Back, WB)

명령어 인출 단계(IF) 는 Instruction Memory 에서 pc 값을 통해 명령어를 읽어드리고 다음 명령어의 주소를 4 를 더하여 업데이트 한다. 명령어 해석 단계(ID)는 인출된 명령어를 해독하여 레지스터가 읽게 하고 opcode 에 따라 제어 신호를 생성한다. Execute(EX) 단계에서는 해독된 명령어를 실행하는 단계이며 ALU 와 control logic 을 통해 연산이 수행된다. Memory Access 단계에서는 ALU 에서 연산 된 주소에 Memory 값을 적재 및 저장하는 단계이다. MIPS ISA 에서는 적재 명령어 와 저장 명령어 만을 통해 memory 에 접근 할 수 있으므로 Memory Access 단계는 위 두 기능을 가진 명령어를 수행하는 단계이다. Write Back 은 ALU 연산 결과 또는 메모리에서 적재한 데이터를 레지스터에 다시 저장하는 단계이다. 위 다섯 단계를 통해 Single Cycle 을 구현 할 수 있다.

이러한 Mips 구조는 단순하고 정해진 공간과 시간 내에서 실행이 되어 실행 시간과 구조에 대한 예측이 용이하다. 하지만, 모든 명령어가 필요 없는 단계를 지나야 되기도 하며, 가장 긴 명령어의 시간에 맞춰 실행이 되도록 설계 되어서 비효율적으로 실행한다.

Single Cycle 코드 설계 고려 요소)

- 레지스터 및 메모리와 리틀 엔디안

Single Cylce 구현 하기 위한 기본 요소는 레지스터와 메모리 이다. 레지스터는 크게 PC 와 32 개의 기본 레지스터로 구성된다. PC 는 실행되는 명령어를 인출 할 수 있도록 한다. PC 값에는 메모리에 저장된 명령어의 주소 값을 가진다. 명령어는 4bytes(32bit) 크기를 차지 하므로 **PC 또한 uint32_t 데이터 타입으로 4bytes** 크기를 갖도록 구현하였다. 32 개의 레지스터 기능과 종류는 레지스터 번호에 따라 다르다. **각 레지스터의 크기는 4bytes** 로 동일하며 이는 uint32_t 를 통해 구현 되었다. 아래 그림은 레지스터의 이름, 번호와 기능을 보여준다

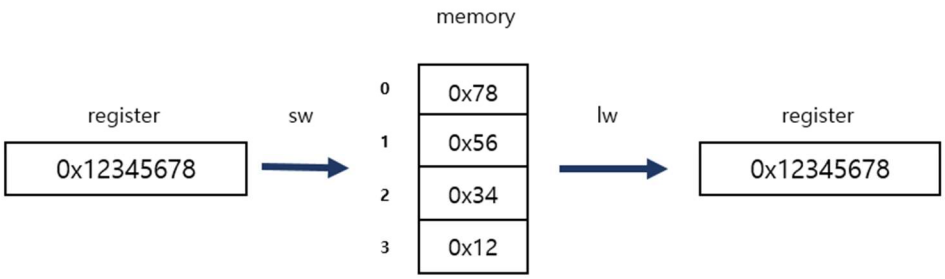
REGISTER NAME, NUMBER, USE, CALL CONVENTION

NAME	NUMBER	USE	PRESERVED ACROSS A CALL?
$zero	0	The Constant Value 0	N.A.
$at	1	Assembler Temporary	No
$v0-$v1	2-3	Values for Function Results and Expression Evaluation	No
$a0-$a3	4-7	Arguments	No
$t0-$t7	8-15	Temporaries	No
$s0-$s7	16-23	Saved Temporaries	Yes
$t8-$t9	24-25	Temporaries	No
$k0-$k1	26-27	Reserved for OS Kernel	No
$gp	28	Global Pointer	Yes
$sp	29	Stack Pointer	Yes
$fp	30	Frame Pointer	Yes
$ra	31	Return Address	No

Memory 는 긴 하나의 linear 한 배열이다. 배열의 개수는 **0x1000000** 개 이며 각 배열의 크기는 **1byte** 이다. 각 레지스터들의 크기는 **uint32_t (4byte)**로 구현하였다. 반면 Memory 는 1byte 이므로 레지스터와 다르게 **uint8_t** 를 통해 레지스터의 크기 1byte 를 구현하였다.

레지스터 중 mult 와 mflo 명령어를 위한 레지스터 hi 와 lo 또한 존재하며 HI 와 LO 의 크기는 4bytes 이다.

레지스터 값을 메모리에 저장할 때, 레지스터 크기는 4bytes 이며 메모리 크기는 1bytes 이므로 레지스터 값을 분할하여 저장 되어야 한다. 분할 하는 방식에는 2 가지가 있는데 대표적으로 빅엔디안과 리틀엔디안이다. 빅엔디안의 경우 사람에게 익숙한 방식의 메모리 분할 방식이다. 이에 반대되는 리틀엔디안은 메모리의 주소 중 가장 낮은 주소에 가장 오른쪽 데이터부터 저장된다. 이를 그림으로 표현하면 다음과 같다.



설계한 코드는 위와 같은 리틀 엔디안 구조로 구현되었으며 이를 구현한 각각의 코드는 다음과 같다.

[Store]

```
for (int i = 0; i < 4; i++) {
    mem[ex2mem[2] + i] = (ex2mem[3] >> 8 * i) & 0xff;
}
```

[Load]

```
uint32_t temp = 0;
for (int j = 3; j >= 0; j--) {
    temp <<= 8;
    temp |= mem[ex2mem[2] + j];
}
mem2wb[2] = temp; // lw로 저장해야되는값.
mem2wb[4] = ex2mem[4]; // write reg 인덱스
```

- **구현된 명령어와 비트 연산**

구현 된 명령어는 총 27 개이며 R type 은 14 개, I type 은 11 개 J type 은 2 개이다. 25 개의 opcode 와 funct 을 표로 표현하면 다음과 같다.

[name-opcode]

Name	Opcode
j	2
jal	3
beq	4
bne	5
addi	8
addiu	9
slti	10
sltiu	11
andi	12
ori	13
lw	35
sw	43
lui	0xf
r-type	0x0

[R-type : name-funct]

Name	Opcode
Add	0x20
Subtract	0x22
And	0x24
Or	0x25
Nor	0x27
sll	0x00
SRL	0x02
slt	0x2a
Sltu	0x2b
jr	0x08
Subu	0x23
addu	0x21

명령어는 32bits 로 이루어져 있으며 명령어안에는 type 에 따라 rs, rt, rd, imm, funct, shamt 를 추출 할 수 있다. 32bit 안에 들어 있는 값들을 추출 하기 위해서 shift-right 연산자와 & 연산자가 필요하다. 예를 들어, rs[25:21]값을 추출하기 위해서, 21 칸을 shift right 해주고 나머지 필요 없는 부분을 제거 하기 위해서 0x1f(0b11111)과 &연산을 해주면 원하는 rs 값을 얻을 수 있다. type 에 따라 추출 할 수 있는 것과 필요한 연산자를 정리해주면 다음과 같다.

1 로 표시된 부분은 값이 필요한 부분이다.

	shift right	and(&) with	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
address	0	0x3ff:ffff							1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
rs	21	0x1f							1	1	1	1	1																					
rt	16	0x1f												1	1	1	1	1																
rd	11	0x1f																	1	1	1	1	1											
imm	0	0xffff																	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
funct	0	0x3f																											1	1	1	1	1	1
shamt	6	0x1f																						1	1	1	1	1						
opcode	26	none	1	1	1	1	1	1																										

- control logic 사용

Single Cycle 을 구현할 때, 각각의 명령어가 필요한 작업을 수행하게 하려면 두가지의 방법이 있다. 첫번째 방법은 Control logic 을 통해 구현하는 것 이며 두번째 방법은 조건문(if-else)을 이용하는 것이다. 사용 된 Control logic 은 아래와 같으며 아래의 것 이외는 조건문을 통해 구현되었다.

가독성 편한 이름 (실제 변수 이름)

1. **mem read (mem_rd)** : memory 읽기 [lw 만 해당]

2. **mem write (mem_wr)** : memory 쓰기 [sw 만 해당]

3. **mem to reg (mem_to_reg)** : 1 이면 메모리에서 온 값이 WB 으로 전달

   0 이면 ALU result 값이 WB 으로 전달

4. **reg write (reg_wr)** : WB 단계에서 reg_write 이 0 이면 실행 되지 못하게

5. **alu control (alu_control)** : alu_control 값에 따라 수행하는 연산이 달라짐 (아래 표로 정리)

6. **ex_skip, ma_skip** : 필요 없는 연산이 수행되지 않도록 skip 하는 control logic

연산	alu control
and	0
or	1
add	10
sub	110
slt	111
nor	1100
sll	1110
srl	1111
mult	1001

- Latch 정의 및 사용

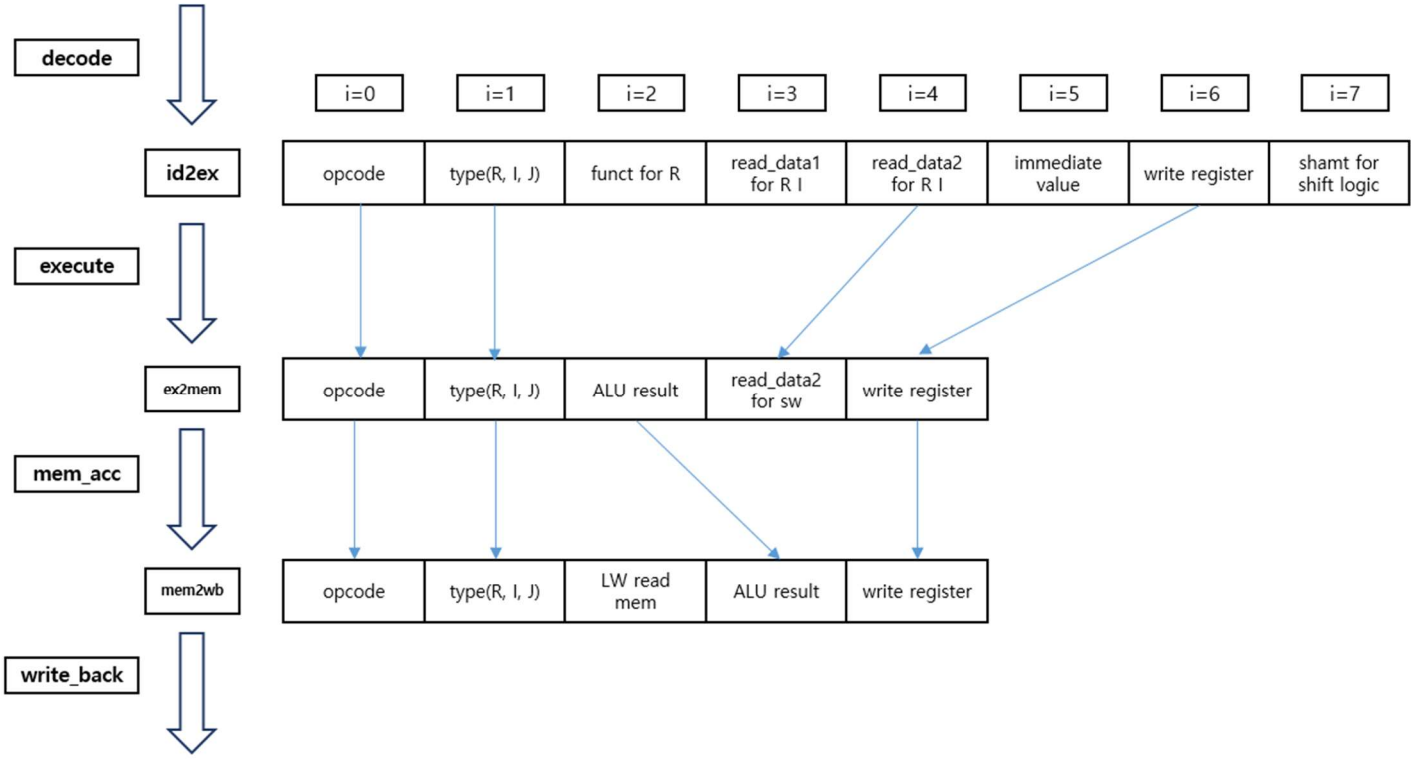
Latch 는 각 단계에서 완료 된 결과 값을 저장하는 temporary pipeline register 이다. 해당 Temporary Register 는 Pipeline 에서 사용 되는 레지스터이지만, Single Cycle 을 구현할 때, 값을 임시적으로 저장하면, 다음 단계로 값을 보내기 용이하여 사용하였다. 또한, Latch 는 데이터를 이후의 데이터에서도 사용 할 수 있으므로 코드의 완성도와 효율성이 높아져 채택하게 되었다. 요약하자면, Latch 는 각각의 단계에서 입력과 출력으로 사용 된다.

코드 실행 순서)

코드 실행은 Main 함수 에서 binary 파일을 읽어 드리고 memory 에 저장 한다. 이후 레지스터를 초기화 시킨후 data path 함수 내부의 Loop 문에서 한 사이클의 Fetch 부터 Write Back 까지를 반복한다. 모든 사이클이 종료되면 요구된 사이클의 개수, 여러 type 들의 사이클 개수와 반환값을 출력한 후 종료된다.

main	1. Read binary file and save to memory
main	2. Initialize register structure
data path	3. initialize the latch & control unit, move to loop
data path	4. Loop : fetch -> decode -> execute -> memory access -> writeback
data path	5. Print significant instruction count
main	6. back to main

Datapath 내부 Loop 문에서 각각의 latch 들이 필요한 정보를 입력하고 출력하는데 사용이 된다. Latch 는 id2ex, ex2mem, mem2wb 로 3 개의 배열로 구성 되어 있다. 각각의 Latch 는 8 개와 5 개의 배열로 이루어져있으며 각각의 필요 하지 않은 값이 latch 에 들어 오는 것을 최소화 하였으며, 필요 하지 않은 값이 들어 갔더라도 코드에 유의미한 영향을 주지 못하는 Don't care 부분이다. 아래는 Latch 의 흐름과 흐름에 따른 변화 이다.



빌드와 코드 결과)

코딩은 VS 2022 에서 진행하였으며 운영체제는 window10, 시스템은 64 비트 운영체제이다.

KOI 에서 gcc -o 32203743 32203743.c 를 여러 번 해보았지만 오류가 나는 관계로 terminal 에 명령어를 넣지 않고 **665 번째 줄 fopen 함수 첫번째 파라미터(파일이름)를 바꾸어주는 것** 으로 빌드를 했다. 아래 사진은 KOI 의 오류 부분이다.

첫번째는 남재현: 컴퓨터구조론, 두번째는 유시환:컴퓨터구조 및 모바일프로세서 결과값

```
File opening failed: Bad address
[1] + Done                       "/usr/bin/gdb" --interpreter=mi --tty=${DbgTerm} 0<"/tmp/Microsoft-MIEngine-In-32frmvm3.klj" 1>"/tmp/Microsoft-MIEngine-Out-npcojn4s.b5f"
koicloud@ca-32203743:~/vscode$ gcc -o 32203743 32203743.c
cc1: fatal error: 32203743.c: No such file or directory
compilation terminated.
```

아래는 665 번째 줄 부분이다.

```
FILE* file = fopen("fibonacci.bin", "rb"); // "filename.bin" => filename 변경해서 다른 파일 넣기.
// fopen함수의 첫번째 매개변수에 아래 값을 넣어주면 실행가능.
// "sum.bin"
// "func.bin"
// "fibonacci.bin"
// "factorial.bin"
// "power.bin"
```

다음은 위 빌드 방법을 통해 KOI 의 결과 값이다.(남재현 교수- 컴퓨터구조론)

[sum.bin]

```
32203743> Cycle : 111
        [Instruction Fetch]  0x8fbe000c (PC=0x00000060)
        [Instruction Decode] Type: I, Inst: lw r30 12(r29)
            opcode: 0x23, , rs: 29 (0xffffff0), rt: 30 (0xffffff0), imm: 12
            RegDst: 0, RegWrite: 1, ALUSrc: 1, PCSrc: 0, MemRead: 1, MemWrite: 0, MemtoReg: 1, ALUOp: 0
        [Execute] ALU = 0xffffffc
        [Memory Access] Load, Address: 0xffffffc, Value: 0x0
        [Write Back] newPC: 0x64

32203743> Cycle : 112
        [Instruction Fetch]  0x27bd0010 (PC=0x00000064)
        [Instruction Decode] Type: I, Inst: addiu r29 r29 16
            opcode: 0x9, , rs: 29 (0xffffff0), rt: 29 (0xffffff0), imm: 16
            RegDst: 0, RegWrite: 0, ALUSrc: 0, PCSrc: 0, MemRead: 0, MemWrite: 0, MemtoReg: 0, ALUOp: 0
        [Execute] ALU = 0x1000000
        [Memory Access] pass
        [Write Back] newPC: 0x68

32203743> Cycle : 113
        [Instruction Fetch]  0x03e00008 (PC=0x00000068)
        [Instruction Decode] Type: R, Inst: jr r31
            opcode: 0x0, , rs: 31 (0xffffffff), funct: 0x8
            RegDst: 1, RegWrite: 1, ALUSrc: 0, PCSrc: 0, MemRead: 0, MemWrite: 0, MemtoReg: 0, ALUOp: 2
        [Execute] Pass
        [Memory Access] Pass
        [Write Back] newPC: 0x3

32203743> Final Result  Cycles: 114, R-type instructions: 13, I-type instructions: 101, J-type instructions: 0
        Return value(v0) : 45
[1] + Done                       "/usr/bin/gdb" --interpreter=mi --tty=${DbgTerm} 0<"/tmp/Microsoft-MIEngine-In-un5b
koicloud@ca-32203743:~/vscode$
```

[func.bin]

```
32203743> Cycle : 85
        [Instruction Fetch]  0x8fbe0020 (PC=0x00000030)
        [Instruction Decode] Type: I, Inst: lw r30 32(r29)
            opcode: 0x23, , rs: 29 (0xffffd8), rt: 30 (0xffffd8), imm: 32
            RegDst: 0, RegWrite: 1, ALUSrc: 1, PCSrc: 0, MemRead: 1, MemWrite: 0, MemtoReg: 1, ALUOp: 0
        [Execute] ALU = 0xffffff8
        [Memory Access] Load, Address: 0xffffff8, Value: 0x0
        [Write Back] newPC: 0x34

32203743> Cycle : 86
        [Instruction Fetch]  0x27bd0028 (PC=0x00000034)
        [Instruction Decode] Type: I, Inst: addiu r29 r29 40
            opcode: 0x9, , rs: 29 (0xffffd8), rt: 29 (0xffffd8), imm: 40
            RegDst: 0, RegWrite: 0, ALUSrc: 0, PCSrc: 0, MemRead: 0, MemWrite: 0, MemtoReg: 0, ALUOp: 0
        [Execute] ALU = 0x1000000
        [Memory Access] pass
        [Write Back] newPC: 0x38

32203743> Cycle : 87
        [Instruction Fetch]  0x03e00008 (PC=0x00000038)
        [Instruction Decode] Type: R, Inst: jr r31
            opcode: 0x0, , rs: 31 (0xffffffff), funct: 0x8
            RegDst: 1, RegWrite: 1, ALUSrc: 0, PCSrc: 0, MemRead: 0, MemWrite: 0, MemtoReg: 0, ALUOp: 2
        [Execute] Pass
        [Memory Access] Pass
        [Write Back] newPC: 0x3

32203743> Final Result  Cycles: 88, R-type instructions: 24, I-type instructions: 60, J-type instructions: 4
        Return value(v0) : 10
[1] + Done                       "/usr/bin/gdb" --interpreter=mi --tty=${DbgTerm} 0<"/tmp/Microsoft-MIEngine-In
koicloud@ca-32203743:~/vscode$
```

32203743 임석범

[fibonacci.bin]

```
32203743> Cycle : 41437
        [Instruction Fetch]  0x8fbe0018 (PC=0x00000028)
        [Instruction Decode] Type: I, Inst: lw r30 24(r29)
            opcode: 0x23, , rs: 29 (0xffffe0), rt: 30 (0xffffe0), imm: 24
            RegDst: 0, RegWrite: 1, ALUSrc: 1, PCSrc: 0, MemRead: 1, MemWrite: 0, MemtoReg: 1, ALUOp: 0
        [Execute] ALU = 0xfffff8
        [Memory Access] Load, Address: 0xfffff8, Value: 0x0
        [Write Back] newPC: 0x2c

32203743> Cycle : 41438
        [Instruction Fetch]  0x27bd0020 (PC=0x0000002c)
        [Instruction Decode] Type: I, Inst: addiu r29 r29 32
            opcode: 0x9, , rs: 29 (0xffffe0), rt: 29 (0xffffe0), imm: 32
            RegDst: 0, RegWrite: 0, ALUSrc: 0, PCSrc: 0, MemRead: 0, MemWrite: 0, MemtoReg: 0, ALUOp: 0
        [Execute] ALU = 0x1000000
        [Memory Access] pass
        [Write Back] newPC: 0x30

32203743> Cycle : 41439
        [Instruction Fetch]  0x03e00008 (PC=0x00000030)
        [Instruction Decode] Type: R, Inst: jr r31
            opcode: 0x0, , rs: 31 (0xffffffff), funct: 0x8
            RegDst: 1, RegWrite: 1, ALUSrc: 0, PCSrc: 0, MemRead: 0, MemWrite: 0, MemtoReg: 0, ALUOp: 2
        [Execute] Pass
        [Memory Access] Pass
        [Write Back] newPC: 0x3

32203743> Final Result  Cycles: 41440, R-type instructions: 9866, I-type instructions: 29601, J-type instructi
        Return value(v0) : 610
[1] + Done                       "/usr/bin/gdb" --interpreter=mi --tty=${DbgTerm} 0<"/tmp/Microsoft-MIEngine-I
koicloud@ca-32203743:~/vscode$
```

[factorial.bin]

```
32203743> Cycle : 117
        [Instruction Fetch]  0x8fbe0018 (PC=0x00000028)
        [Instruction Decode] Type: I, Inst: lw r30 24(r29)
            opcode: 0x23, , rs: 29 (0xffffe0), rt: 30 (0xffffe0), imm: 24
            RegDst: 0, RegWrite: 1, ALUSrc: 1, PCSrc: 0, MemRead: 1, MemWrite: 0, MemtoReg: 1, ALUOp: 0
        [Execute] ALU = 0xfffff8
        [Memory Access] Load, Address: 0xfffff8, Value: 0x0
        [Write Back] newPC: 0x2c

32203743> Cycle : 118
        [Instruction Fetch]  0x27bd0020 (PC=0x0000002c)
        [Instruction Decode] Type: I, Inst: addiu r29 r29 32
            opcode: 0x9, , rs: 29 (0xffffe0), rt: 29 (0xffffe0), imm: 32
            RegDst: 0, RegWrite: 0, ALUSrc: 0, PCSrc: 0, MemRead: 0, MemWrite: 0, MemtoReg: 0, ALUOp: 0
        [Execute] ALU = 0x1000000
        [Memory Access] pass
        [Write Back] newPC: 0x30

32203743> Cycle : 119
        [Instruction Fetch]  0x03e00008 (PC=0x00000030)
        [Instruction Decode] Type: R, Inst: jr r31
            opcode: 0x0, , rs: 31 (0xffffffff), funct: 0x8
            RegDst: 1, RegWrite: 1, ALUSrc: 0, PCSrc: 0, MemRead: 0, MemWrite: 0, MemtoReg: 0, ALUOp: 2
        [Execute] Pass
        [Memory Access] Pass
        [Write Back] newPC: 0x3

32203743> Final Result  Cycles: 120, R-type instructions: 34, I-type instructions: 81, J-type instructions: 5
        Return value(v0) : 120
[1] + Done                       "/usr/bin/gdb" --interpreter=mi --tty=${DbgTerm} 0<"/tmp/Microsoft-MIEngine-In-
koicloud@ca-32203743:~/vscode$
```

[power.bin]

```
32203743> Cycle : 90
        [Instruction Fetch]  0x8fbe0018 (PC=0x0000002c)
        [Instruction Decode] Type: I, Inst: lw r30 24(r29)
            opcode: 0x23, , rs: 29 (0xffffe0), rt: 30 (0xffffe0), imm: 24
            RegDst: 0, RegWrite: 1, ALUSrc: 1, PCSrc: 0, MemRead: 1, MemWrite: 0, MemtoReg: 1, ALUOp: 0
        [Execute] ALU = 0xfffff8
        [Memory Access] Load, Address: 0xfffff8, Value: 0x0
        [Write Back] newPC: 0x30

32203743> Cycle : 91
        [Instruction Fetch]  0x27bd0020 (PC=0x00000030)
        [Instruction Decode] Type: I, Inst: addiu r29 r29 32
            opcode: 0x9, , rs: 29 (0xffffe0), rt: 29 (0xffffe0), imm: 32
            RegDst: 0, RegWrite: 0, ALUSrc: 0, PCSrc: 0, MemRead: 0, MemWrite: 0, MemtoReg: 0, ALUOp: 0
        [Execute] ALU = 0x1000000
        [Memory Access] pass
        [Write Back] newPC: 0x34

32203743> Cycle : 92
        [Instruction Fetch]  0x03e00008 (PC=0x00000034)
        [Instruction Decode] Type: R, Inst: jr r31
            opcode: 0x0, , rs: 31 (0xffffffff), funct: 0x8
            RegDst: 1, RegWrite: 1, ALUSrc: 0, PCSrc: 0, MemRead: 0, MemWrite: 0, MemtoReg: 0, ALUOp: 2
        [Execute] Pass
        [Memory Access] Pass
        [Write Back] newPC: 0x3

32203743> Final Result  Cycles: 93, R-type instructions: 27, I-type instructions: 62, J-type instructions: 4
        Return value(v0) : 1000
[1] + Done                       "/usr/bin/gdb" --interpreter=mi --tty=${DbgTerm} 0<"/tmp/Microsoft-MIEngine-In
koicloud@ca-32203743:~/vscode$
```

코드의 실행결과 : 유시환 컴모프)

[simple2.bin]

```
0x0 :   27bdffe8        addiu alu : [ 1000000    ffffffe8 ]     alu result : ffffe8 -> reg[29]

0x4 :   afbe0014        sw alu : [ ffffe8        14 ]   sw : store val 0 -> Mem[fffffc]

0x8 :   03a0f021        addu alu : [ ffffe8      0 ]    alu result : ffffe8 -> reg[30]

0xc :   24020064        addiu alu : [ 0  64 ]   alu result : 64 -> reg[2]

0x10 :  afc20008        sw alu : [ ffffe8        8 ]    sw : store val 64 -> Mem[fffff0]

0x14 :  8fc20008        lw alu : [ ffffe8        8 ]    lw : load Mem[fffff0] -> 64 val -> reg[2]

0x18 :  03c0e821        addu alu : [ ffffe8      0 ]    alu result : ffffe8 -> reg[29]

0x1c :  8fbe0014        lw alu : [ ffffe8        14 ]   lw : load Mem[fffffc] -> 0 val -> reg[30]

0x20 :  27bd0018        addiu alu : [ ffffe8     18 ]   alu result : 1000000 -> reg[29]

0x24 :  03e00008        jr : move to ffffffff

================================================================================
Return register (r2)                : 100
Total cycle                         : 10
r-type count                        : 3
i-type count                        : 7
j-type count                        : 0
branch count                        : 0
memory access count                 : 4
================================================================================
```

[simple.bin]

```
0x0 :   27bdfff8        addiu alu : [ 1000000    fffffff8 ]     alu result : fffff8 -> reg[29]

0x4 :   afbe0004        sw alu : [ fffff8        4 ]    sw : store val 0 -> Mem[fffffc]

0x8 :   03a0f021        addu alu : [ fffff8      0 ]    alu result : fffff8 -> reg[30]

0xc :   NOP

0x10 :  03c0e821        addu alu : [ fffff8      0 ]    alu result : fffff8 -> reg[29]

0x14 :  8fbe0004        lw alu : [ fffff8        4 ]    lw : load Mem[fffffc] -> 0 val -> reg[30]

0x18 :  27bd0008        addiu alu : [ fffff8     8 ]    alu result : 1000000 -> reg[29]

0x1c :  03e00008        jr : move to ffffffff

================================================================================
Return register (r2)                : 0
Total cycle                         : 7
r-type count                        : 3
i-type count                        : 4
j-type count                        : 0
branch count                        : 0
memory access count                 : 2
================================================================================
```

[simple4.bin]

```
0x9c :  8fbe0020        lw alu : [ ffffb8        20 ]   lw : load Mem[ffffd8] -> ffffe0 val -> reg[30]

0xa0 :  27bd0028        addiu alu : [ ffffb8     28 ]   alu result : ffffe0 -> reg[29]

0xa4 :  03e00008        jr : move to 1c

0x1c :  03c0e821        addu alu : [ ffffe0      0 ]    alu result : ffffe0 -> reg[29]

0x20 :  8fbf001c        lw alu : [ ffffe0        1c ]   lw : load Mem[fffffc] -> ffffffff val -> reg[31]

0x24 :  8fbe0018        lw alu : [ ffffe0        18 ]   lw : load Mem[fffff8] -> 0 val -> reg[30]

0x28 :  27bd0020        addiu alu : [ ffffe0     20 ]   alu result : 1000000 -> reg[29]

0x2c :  03e00008        jr : move to ffffffff

================================================================================
Return register (r2)                : 55
Total cycle                         : 224
r-type count                        : 60
i-type count                        : 153
j-type count                        : 11
branch count                        : 10
memory access count                 : 100
================================================================================
```

[simple3.bin]

```
0x4c :  28420065        slti alu : [ 65  65 ]   alu result : 0 -> reg[2]

0x50 :  1440fff3        bne bne : [ 0   0 ]     bne false : move to next addr : 54

0x54 :  NOP

0x58 :  8fc2000c        lw alu : [ ffffe8        c ]    lw : load Mem[fffff4] -> 13ba val -> reg[2]

0x5c :  03c0e821        addu alu : [ ffffe8      0 ]    alu result : ffffe8 -> reg[29]

0x60 :  8fbe0014        lw alu : [ ffffe8        14 ]   lw : load Mem[ffffc] -> 0 val -> reg[30]

0x64 :  27bd0018        addiu alu : [ ffffe8     18 ]   alu result : 1000000 -> reg[29]

0x68 :  03e00008        jr : move to ffffffff

================================================================================
Return register (r2)                : 5050
Total cycle                         : 1025
r-type count                        : 104
i-type count                        : 920
j-type count                        : 1
branch count                        : 102
memory access count                 : 613
================================================================================
```

[gcd.bin]

```
0xf0 :  27bd0020        addiu alu : [ ffffb0     20 ]   alu result : ffffd0 -> reg[29]

0xf4 :  03e00008        jr : move to 30

0x30 :  afc20020        sw alu : [ ffffd0        20 ]   sw : store val 1 -> Mem[fffff0]

0x34 :  03c0e821        addu alu : [ ffffd0      0 ]    alu result : ffffd0 -> reg[29]

0x38 :  8fbf002c        lw alu : [ ffffd0        2c ]   lw : load Mem[ffffc] -> ffffffff val -> reg[31]

0x3c :  8fbe0028        lw alu : [ ffffd0        28 ]   lw : load Mem[fffff8] -> 0 val -> reg[30]

0x40 :  27bd0030        addiu alu : [ ffffd0     30 ]   alu result : 1000000 -> reg[29]

0x44 :  03e00008        jr : move to ffffffff

================================================================================
Return register (r2)                : 1
Total cycle                         : 924
r-type count                        : 222
i-type count                        : 637
j-type count                        : 65
branch count                        : 73
memory access count                 : 486
================================================================================
```

[fib.bin]

```
0xc8 :  27bd0030        addiu alu : [ ffffa8     30 ]   alu result : ffffd8 -> reg[29]

0xcc :  03e00008        jr : move to 24

0x24 :  afc2001c        sw alu : [ ffffd8        1c ]   sw : store val 37 -> Mem[fffff4]

0x28 :  03c0e821        addu alu : [ ffffd8      0 ]    alu result : ffffd8 -> reg[29]

0x2c :  8fbf0024        lw alu : [ ffffd8        24 ]   lw : load Mem[fffffc] -> ffffffff val -> reg[31]

0x30 :  8fbe0020        lw alu : [ ffffd8        20 ]   lw : load Mem[fffff8] -> 0 val -> reg[30]

0x34 :  27bd0028        addiu alu : [ ffffd8     28 ]   alu result : 1000000 -> reg[29]

0x38 :  03e00008        jr : move to ffffffff

================================================================================
Return register (r2)                : 55
Total cycle                         : 2407
r-type count                        : 546
i-type count                        : 1697
j-type count                        : 164
branch count                        : 109
memory access count                 : 1095
================================================================================
```

[input4.bin]

```
0x18ea8 :       NOP

0x18eac :       ac430014        sw alu : [ ff6534       14 ]   sw : store val 56 -> Mem[ff6548]

0x18eb0 :       8fc20008        lw alu : [ ff6398       8 ]    lw : load Mem[ff63a0] -> 65 val -> reg[2]

0x18eb4 :       NOP

0x18eb8 :       24420001        addiu alu : [ 65        1 ]    alu result : 66 -> reg[2]

0x18ebc :       afc20008        sw alu : [ ff6398       8 ]    sw : store val 66 -> Mem[ff63a0]

0x18ec0 :       8fc20008        lw alu : [ ff6398       8 ]    lw : load Mem[ff63a0] -> 66 val -> reg[2]

0x18ec4 :       NOP

0x18ec8 :       28420066        slti alu : [ 66  66 ]   alu result : 0 -> reg[2]

0x18ecc :       1440ffb7        bne bne : [ 0   0 ]     bne false : move to next addr : 18ed0

0x18ed0 :       NOP

0x18ed4 :       8fc201ac        lw alu : [ ff6398       1ac ]  lw : load Mem[ff6544] -> 55 val -> reg[2]

0x18ed8 :       NOP

0x18edc :       afc20014        sw alu : [ ff6398       14 ]   sw : store val 55 -> Mem[ff63ac]

0x18ee0 :       8fc20014        lw alu : [ ff6398       14 ]   lw : load Mem[ff63ac] -> 55 val -> reg[2]

0x18ee4 :       27dd1c78        addiu alu : [ ff6398    1c78 ]         alu result : ff8010 -> reg[29]

0x18ee8 :       8fbe7fec        lw alu : [ ff8010       7fec ]         lw : load Mem[fffffc] -> 0 val -> reg[30]

0x18eec :       27bd7ff0        addiu alu : [ ff8010    7ff0 ]         alu result : 1000000 -> reg[29]

0x18ef0 :       03e00008        jr : move to ffffffff

================================================================================
Return register (r2)                : 85
Total cycle                         : 18296212
r-type count                        : 5076368
i-type count                        : 13219741
j-type count                        : 103
branch count                        : 2029699
memory access count                 : 7116606
================================================================================

C:\Users\SeokBeom\source\repos\Single Cycle\Debug\Single Cycle.exe(프로세스 32724개)이(가) 종료되었습니다(코드: 0개).
이 창을 닫으려면 아무 키나 누르세요...
```

결론

이번 과제를 구현하며 각종 trade-off 와 현실적인 구현에 대한 고민들을 통해 ISA 와 Single Cycle 의 설계에 대한 이해를 높일 수 있었다. 메모리와 레지스터의 크기 설정, 메모리와 레지스터의 데이터 교환 방식(Little endian), 명령어의 비트 활용을 통한 정보 추출, Control logic 과 조건문 사이의 trade-off(조건문이 많아질 수 록 늘어나는 복잡도), Latch 의 채택등 여러 부분을 다시 고민 할 수 있었다. 이번 과제 및 코드에서 아쉬운 점은 branch, jump, mflo, mult 등이 execute 하는 과정에서 대부분 처리가 되고 이후의 과정은 크게 유의미 하지 않다. 코드를 짜고 디버깅하는 과정에 들어가서 제대로 된 처리를 하지 못하였다. 또한 control logic 을 더욱 자세하게 하지 못한 점이 코드를 복잡하게 만들었다고 생각한다. 이러한 점들을 고치면 더 나은 코드가 될 수 있을 것이라고 생각한다.