

REPORT

HW1



과목명 : 컴퓨터구조론 (2분반)
교 수 : 남재현 교수님
소속학과: 소프트웨어학과
학 번 : 32190192
이 름 : 구영민
마 감 : 2024.04.15 (월요일까지)

목 차

1. Instruction
2. Background
3. Design
4. Implementation
5. Test
6. Lesson

1. Instruction

- 과제 목표

RISC중 하나인 MIPS 어셈블리어를 통한 계산기를 직접 코딩해보는 과제입니다. 파일 입출력으로 어셈블리어로 코딩 된 텍스트파일을 읽어와 파싱하여 연산 후 결과를 출력해주는 프로그램을 c언어로 구현하는 것 입니다.

- 요구사항 분석

1. 구현해야하는 연산에는 산술연산(add, sub, mul, div), 조건연산(BEQ, BNE, SLT), JMP, NOP, LW(레지스터로 가져오는 연산) 총 10가지의 연산을 구현하면 됩니다.
2. 레지스터 : MIPS 아키텍처에서는 총 32개의 레지스터가 있습니다. 이 과제에서는 10개의 임시저장 레지스터 (t0 ~ t9)와 변수를 가져올 때 사용되는 8개의 레지스터 (s0 ~ s7)를 지원합니다. 또한, 반환 값을 위한 레지스터 v0를 사용하고 0은 zero레지스터를 이용합니다.

- 예외처리

1. 16진수가 아닐 때 예외처리를 해줘야합니다.
2. DIV instruction에서 0으로 나눌 때 예외처리가 필요합니다.
3. 명령어 당 token 개수가 맞지 않으면 예외처리를 해주어야합니다.
4. 레지스터 검증 예외 처리를 해주어야합니다. EX) LW에는 s레지스터만 올 수 있다.
5. 파일이 1000줄 넘어갔을 때 예외처리를 해주어야합니다.

- 구현 시 주의 해야 할 점

1. r0을 v0으로 변경한다.
 2. 파일을 다 읽으면 v0값을 출력 후 프로그램을 종료한다.
 3. 한라인에 인스트럭션은 하나이다.
 4. 0x로 시작하면 16진수로 취급한다.
 5. [value from string]은 원래 메모린데 상수값이 있으며 상수 값을 읽어와 처리할 것이다.
-

2. Background

MIPS

MIPS(Microprocessor without Interlocked Pipeline Stages) 아키텍처는 RISC(Reduced Instruction Set Computer) 아키텍처의 하나로, 단순하고 효율적인 명령어 집합을 갖고 있습니다.

MIPS Register

zero 레지스터는 Register[0]을 사용하고

v0레지스터는 Register[2]

t0 ~ t9 레지스터는 Register[8] ~ Register[15]

s0 ~ s7 레지스터는 Register[16] ~ Register[23] 를 사용합니다.

Offset

이 과제를 수행하기 위해 가장 중요하다고 생각하는 개념은 offset 개념입니다.

offset 개념을 사용하여 t, r, zero, v0 레지스터를 Register[32]라는 하나의 배열에 사용하여 레지스터를 효율적으로 사용할 수 있었고 코드의 재사용성을 높일 수 있었습니다.

MIPS instruction의 역할

ADD t1 s2 s3 : t1에 $s2 + s3$ 의 값을 저장합니다.

SUB t1 s2 s3 : t1에 $s2 - s3$ 의 값을 저장합니다.

MUL t1 s2 s3 : t1에 $s2 \times s3$ 의 값을 저장합니다.

DIV t1 s2 s3 : t1에 $s2 / s3$ 의 값을 저장합니다.

NOP : 빈공간입니다.

JMP [line number] : [line number]로 점프합니다.

LW : 메모리에서 레지스터로 가져오는 명령어입니다.

BEQ [SRC1] [SRC2] [line number] : SRC1과 SRC2가 같으면 [line number]로 점프합니다.

BNE [SRC1] [SRC2] [line number] : SRC1과 SRC2가 다르면 [line number]로 점프합니다.

SLT [DST1] [SRC1] [SRC2] : SRC1이 SRC2보다 작으면 1을 작지 않으면 0을 DST1에 저장합니다.

파일 입출력

C프로그래밍에서 파일 입출력을 통해 프로그램은 외부 파일에 데이터를 쓰거나 파일에서 데이터를 읽을 수 있습니다. 이를 통해 프로그램은 외부 데이터와 상호작용할 수 있습니다.

파일 입출력 함수

fopen()

fopen 함수는 파일을 열기 위해 사용됩니다. 파일을 열 때 모드에는 읽기 모드(r), 쓰기 모드(w), 추가 모드(a) 등이 있습니다.

fgets()

fgets 함수는 파일에서 한 줄씩 데이터를 읽어옵니다. 문자열 포인터와 최대 읽을 문자 개수, 그리고 파일 포인터를 인자로 받습니다. 파일에서 한 줄씩 읽어올 때에는 개행 문자('\n')를 만날 때까지 읽습니다.

strtok()

strtok 함수는 문자열을 특정 구분자(delimiter)를 기준으로 토큰으로 분할합니다.

첫 번째 호출 시 문자열 포인터와 구분자를 인자로 받습니다. 이후 호출 시에는 NULL을 전달하여 이전 호출에서 반환된 문자열의 다음 부분을 처리합니다.

EOP(End of File)

파일의 끝(End of File)을 나타내는 것으로, 파일을 읽을 때 파일의 끝을 나타내는 신호입니다. 파일을 읽을 때 fgets 함수가 NULL을 반환하면서 파일의 끝에 도달했음을 나타냅니다. 이를 이용하여 파일의 끝을 인식하고 파일 읽기를 종료합니다.

3. Design

- 프로그램 구조

1. 코드는 main 함수를 중심으로 구성되어 있습니다. Main 함수에서는 파일을 읽어들이고 파싱한 후에 명령어를 실행하고 결과를 출력합니다.
2. 각 명령어에 대한 처리는 별도의 함수로 분리되어 있습니다. ADD, SUB, MUL, DIV, LW, NOP, JMP, BEQ, BNE, SLT 함수가 각각의 명령어를 처리합니다.
3. 마지막에는 동적으로 할당한 메모리를 해제해주는 memory_free() 함수가 존재합니다.

- 파일 파싱

1. 파일은 한 줄씩 읽어들이며 명령어를 파싱하고, 이를 2차원 배열(arr)에 저장합니다.
JMP를 구현할 때 strtok으로 파싱한 값을 일차원 배열로 사용하기보단 2차원 배열로 사용해야 row를 이용하여 JMP를 쉽게 구현할 수 있을 것 같아서 2차원배열을 사용하였습니다.
2. 예외처리를 할 때 필요한 Instruction당 operand의 수를 저장하기 위해 row_cnt배열에 operand의 수를 저장합니다.

- 명령어 처리

1. 명령어를 파싱하면 해당하는 명령어 처리 함수를 호출하도록 설계하였습니다. 이때, 각 명령어에 따라 필요한 오퍼랜드를 사용하여 동작을 수행합니다.
2. 각 명령어 함수에서 연산이 필요하면 base_pointer() + move_offset()으로 offset 위치를 설정한 후 연산을 수행합니다.
3. 명령어 함수가 조건문이면 먼저 참 거짓을 판별한 후 결과에 따른 JMP함수를 호출

합니다.

- 레지스터 및 메모리 관리

1. 코드에서 Register 배열은 32bit 레지스터입니다.

```
zero = Register[0]
```

```
v0 = Register[2]
```

```
t0 ~ t9 = Register[8] ~ Register[15]
```

```
s0 ~ s7 = Register[16] ~ Register[23]
```

2. 코드에서 base_pointer() 함수는 명령어에서 사용된 레지스터가 어느 범위에 해당하는지 base_pointer 위치를 잡아줍니다.
3. 코드에서 move_offset() 함수는 명령어에서 사용된 레지스터의 오프셋 값을 계산합니다. 명령어에서 사용된 레지스터 이름의 두 번째 문자를 정수로 변환하여 오프셋 값을 구합니다.

- 예외처리 및 예외 상황

1. 16진수가 아닐 때 예외처리를 해주었습니다.
2. DIV instruction에서 0으로 나눌 때 예외 처리를 해주었습니다.
3. row_cnt 배열을 사용하여 한 줄 당 명령어 개수를 저장해주어 명령어 개수가 맞지 않을 때 예외처리를 해주는 로직을 작성하였습니다.
4. 레지스터 검증 예외처리 EX) LW에는 S레지스터만 올 수 있다.
5. 파일이 1000줄이 넘어갔을 때 예외처리를 해주었습니다.

- 기타

1. hex_string_to_int() 함수는 16진수를 10진수로 바꿔주는 함수입니다.

2. Parse() 함수는 파일에서 읽어온 값을 파싱 하여 한 줄 씩 2차원배열에 저장하는 함수입니다.
3. memory_free() 함수는 동적으로 할당된 메모리를 해제해주는 함수입니다

4. Implementation

- 모든 instruction 구현 완료 하였습니다.

instruction	구현 여부
ADD	O
SUB	O
MUL	O
DIV	O
NOP	O
LW	O
JMP	O
BEQ	O
BNE	O
SLT	O

5. Test

[Testcase1] : LW, ADD instruction 테스트 케이스입니다.

```

testcase7.txt X testcase1.txt X testcase8.txt testcas
testcase1.txt
1    LW    s0 0xF
2    LW    s1 0x4
3    ADD   v0 s0 s1

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS
● koicloud@ca-32190192:~/vscode$ gcc -o 32190192 32190192.c
● koicloud@ca-32190192:~/vscode$ ./32190192 testcase1.txt
(line 1) LW s0 0xF
32190192> Loaded 15 to s0
(line 2) LW s1 0x4
32190192> Loaded 4 to s1
(line 3) ADD v0 s0 s1
32190192> Added s0(15) to s1(4) and changed v0 to 19
32190192> v0 is 19
○ koicloud@ca-32190192:~/vscode$
  
```

[Testcase2] : LW, MUL instruction 테스트 케이스입니다.

```

testcase7.txt testcase2.txt X testcase1.txt testcases
testcase2.txt
1    LW    s0 0x2
2    LW    s1 0x4
3    MUL   v0 s0 s1

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS
● koicloud@ca-32190192:~/vscode$ gcc -o 32190192 32190192.c
● koicloud@ca-32190192:~/vscode$ ./32190192 testcase2.txt
(line 1) LW s0 0x2
32190192> Loaded 2 to s0
(line 2) LW s1 0x4
32190192> Loaded 4 to s1
(line 3) MUL v0 s0 s1
32190192> Multiplied s0(2) to s1(4) and changed v0 to 8
32190192> v0 is 8
○ koicloud@ca-32190192:~/vscode$
  
```

[Testcase3] : SUB instruction 테스트 케이스입니다.

≡ testcase3.txt X

≡ testcase6.txt

≡ testcase7.txt

≡ test

≡ testcase3.txt

```

1  LW  s0 0x1
2  LW  s1 0x2
3  LW  s2 0x3
4  LW  s3 0x4
5  ADD t0 s0 s1
6  ADD t1 s2 s3
7  SUB v0 t0 t1
  
```

PROBLEMS

OUTPUT

DEBUG CONSOLE

TERMINAL

PORTS

```

● koicloud@ca-32190192:~/vscode$ gcc -o 32190192 32190192.c
● koicloud@ca-32190192:~/vscode$ ./32190192 testcase3.txt
(line 1) LW  s0  0x1
32190192> Loaded 1 to s0
(line 2) LW  s1  0x2
32190192> Loaded 2 to s1
(line 3) LW  s2  0x3
32190192> Loaded 3 to s2
(line 4) LW  s3  0x4
32190192> Loaded 4 to s3
(line 5) ADD  t0  s0  s1
32190192> Added s0(1) to s1(2) and changed t0 to 3
(line 6) ADD  t1  s2  s3
32190192> Added s2(3) to s3(4) and changed t1 to 7
(line 7) SUB  v0  t0  t1
32190192> Subtracted t0(3) to t1(7) and changed v0 to -4
32190192> v0 is  -4
○ koicloud@ca-32190192:~/vscode$ █
  
```

[Testcase4] : JMP instruction 테스트 케이스입니다.

testcase7.txt
testcase4.txt X
testcase2.txt
testcase1.tx

testcase4.txt

```

1  LW  s0 0x5
2  LW  s1 0xA
3  ADD t0 s0 s1
4  JMP 0x8
5  LW  s0 0x1
6  LW  s1 0x2
7  ADD t0 s0 s1
8  LW  s2 0x3
9  MUL v0 t0 s2

```

PROBLEMS
OUTPUT
DEBUG CONSOLE
TERMINAL
PORTS

- koicloud@ca-32190192:~/vscode\$ gcc -o 32190192 32190192.c
- koicloud@ca-32190192:~/vscode\$./32190192 testcase4.txt

```

(line 1) LW  s0  0x5
32190192> Loaded 5 to s0
(line 2) LW  s1  0xA
32190192> Loaded 10 to s1
(line 3) ADD  t0  s0  s1
32190192> Added s0(5) to s1(10) and changed t0 to 15
(line 4) JMP  0x8
32190192> jumped to line 8
(line 8) LW  s2  0x3
32190192> Loaded 3 to s2
(line 9) MUL  v0  t0  s2
32190192> Multiplied t0(15) to s2(3) and changed v0 to 45
32190192> v0 is  45
koicloud@ca-32190192:~/vscode$

```

[Testcase5] : BEQ instruction 테스트 케이스입니다.

testcase5.txt X	slt.txt	add2.txt	bne.txt	testcase3.txt
-----------------	---------	----------	---------	---------------

```

testcase5.txt
1  LW  s0 0x5
2  LW  s1 0xA
3  BEQ s0 s1 0x6
4  ADD v0 s0 zero
5  JMP 0x7
6  ADD v0 s1 zero
7  NOP
  
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

```

koicloud@ca-32190192:~/vscode$ gcc -o 32190192 32190192.c
koicloud@ca-32190192:~/vscode$ ./32190192 testcase5.txt
(line 1) LW  s0 0x5
32190192> Loaded 5 to s0
(line 2) LW  s1 0xA
32190192> Loaded 10 to s1
(line 3) BEQ s0 s1 0x6
32190192> Checked if s0(5) is not equal to s1(10) and didn't jumped to line 6
(line 4) ADD v0 s0 zero
32190192> Added s0(5) to zero(0) and changed v0 to 5
(line 5) JMP 0x7
32190192> jumped to line 7
(line 7) NOP
32190192> No operation
32190192> v0 is 5
koicloud@ca-32190192:~/vscode$ 
  
```

[Testcase6] : BNE instruction 테스트 케이스입니다.

slt.txt	testcase6.txt X	add2.txt	bne.txt	te
---------	-----------------	----------	---------	----

```

testcase6.txt
1  LW  s0 0x5
2  LW  s1 0xA
3  BNE s0 s1 0x6
4  ADD v0 s0 zero
5  JMP 0x7
6  ADD v0 s1 zero
7  NOP
  
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

```

koicloud@ca-32190192:~/vscode$ gcc -o 32190192 32190192.c
koicloud@ca-32190192:~/vscode$ ./32190192 testcase6.txt
(line 1) LW  s0 0x5
32190192> Loaded 5 to s0
(line 2) LW  s1 0xA
32190192> Loaded 10 to s1
(line 3) BNE s0 s1 0x6
32190192> Checked if s0(5) is not equal to s1(10) and jumped to line 6
(line 6) ADD v0 s1 zero
32190192> Added s1(10) to zero(0) and changed v0 to 10
(line 7) NOP
32190192> No operation
32190192> v0 is 10
koicloud@ca-32190192:~/vscode$ 
  
```

[Testcase7] : SLT instruction 테스트 케이스입니다.

testcase7.txt X	add2.txt	bne.txt	testcase3.txt	testc
-----------------	----------	---------	---------------	-------

testcase7.txt

```

1  LW  s0 0x5
2  LW  s1 0xA
3  SLT t0 s0 s1
4  BNE t0 zero 0x7
5  ADD v0 s0 zero
6  JMP 0x8
7  ADD v0 s1 zero
8  NOP

```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

bash + v

```

koicloud@ca-32190192:~/vscode$ gcc -o 32190192 32190192.c
koicloud@ca-32190192:~/vscode$ ./32190192 testcase7.txt
(line 1) LW  s0 0x5
32190192> Loaded 5 to s0
(line 2) LW  s1 0xA
32190192> Loaded 10 to s1
(line 3) SLT t0 s0 s1
32190192> Checked if s0(5) is less than s1(10) and set 1 to t0
(line 4) BNE t0 zero 0x7
32190192> Checked if t0(1) is not equal to zero(0) and jumped to line 7
(line 7) ADD v0 s1 zero
32190192> Added s1(10) to zero(0) and changed v0 to 10
(line 8) NOP
32190192> No operation
32190192> v0 is 10
koicloud@ca-32190192:~/vscode$

```

[Testcase8] : NOP 테스트 케이스입니다.

```

testcase8.txt X testcase4.txt X testcase2.txt testcase
testcase8.txt
1 LW s0 0x2
2 LW s1 0x10
3 ADD v0 s0 s1
4 NOP

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

koicloud@ca-32190192:~/vscode$ gcc -o 32190192 32190192.c
koicloud@ca-32190192:~/vscode$ ./32190192 testcase8.txt
(line 1) LW s0 0x2
32190192> Loaded 2 to s0
(line 2) LW s1 0x10
32190192> Loaded 16 to s1
(line 3) ADD v0 s0 s1
32190192> Added s0(2) to s1(16) and changed v0 to 18
(line 4) NOP
32190192> No operation
32190192> v0 is 18
koicloud@ca-32190192:~/vscode$
Ln 3, Col 7 Spaces: 4

```

[Testcase9] : DIV 테스트 케이스입니다.

Testcase1 ~ Testcase8은 HW1 pdf에 있는 테스트 케이스이고 뒤로는 추가로 작성한 테스트 케이스들 입니다.

```

testcase9.txt X while.txt exception4.txt exce
testcase9.txt
1 LW s0 0xA
2 LW s1 0x2
3 DIV v0 s0 s1

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

koicloud@ca-32190192:~/vscode$ gcc -o 32190192 32190192.c
koicloud@ca-32190192:~/vscode$ ./32190192 testcase9.txt
(line 1) LW s0 0xA
32190192> Loaded 10 to s0
(line 2) LW s1 0x2
32190192> Loaded 2 to s1
(line 3) DIV v0 s0 s1
32190192> Divided s0(10) to s1(2) and changed v0 to 5
32190192> v0 is 5
koicloud@ca-32190192:~/vscode$

```

예외 테스트

Exception 1 : 16진수가 아닐 때 예외 테스트

exception1.txt X	jmp.txt	testcase5.txt	slt.tx
exception1.txt <pre> 1 LW s0 0x2 2 LW s1 10 3 ADD v0 s0 zero </pre>			

PROBLEMS	OUTPUT	DEBUG CONSOLE	TERMINAL	PORTS
<pre> koicloud@ca-32190192:~/vscode\$ gcc -o 32190192 32190192.c koicloud@ca-32190192:~/vscode\$./32190192 exception1.txt (line 1) LW s0 0x2 32190192> Loaded 2 to s0 (line 2) LW s1 10 exception! 10는 16진수가 아닙니다. koicloud@ca-32190192:~/vscode\$ </pre>				

Exception 2 : 0으로 나눌 때 예외 테스트

exception2.txt X	exception1.txt	jmp.txt	tes
exception2.txt <pre> 1 LW s0 0x2 2 DIV v0 s0 zero </pre>			

PROBLEMS	OUTPUT	DEBUG CONSOLE	TERMINAL	PORTS
<pre> koicloud@ca-32190192:~/vscode\$ gcc -o 32190192 32190192.c koicloud@ca-32190192:~/vscode\$./32190192 exception2.txt (line 1) LW s0 0x2 32190192> Loaded 2 to s0 (line 2) DIV v0 s0 zero exception! 0으로 나눌수 없습니다. koicloud@ca-32190192:~/vscode\$ </pre>				

Exception 3 : token 개수가 맞지 않을 때 예외 테스트

exception3.txt X	exception4.txt	exception2.txt	
------------------	----------------	----------------	--

```

exception3.txt
1  LW  s0 0x2
2  LW  s1 0x4
3  ADD v0 s0 zero s1
  
```

PROBLEMS	OUTPUT	DEBUG CONSOLE	TERMINAL	PORTS
----------	--------	---------------	----------	-------

```

koicloud@ca-32190192:~/vscode$ gcc -o 32190192 32190192.c
koicloud@ca-32190192:~/vscode$ ./32190192 exception3.txt
(line 1) LW s0 0x2
32190192> Loaded 2 to s0
(line 2) LW s1 0x4
32190192> Loaded 4 to s1
(line 3) ADD v0 s0 zero s1
exception! ADD_OPERAND_SIZE는 3입니다.
koicloud@ca-32190192:~/vscode$ 
  
```

Exception 4 : 레지스터가 올바르지 않을 때 예외 테스트

exception4.txt X	exception3.txt	exception2.txt	
------------------	----------------	----------------	--

```

exception4.txt
1  LW  s0 0x2
2  LW  t1 0x4
3  ADD v0 s0 zero
  
```

PROBLEMS	OUTPUT	DEBUG CONSOLE	TERMINAL	PORTS
----------	--------	---------------	----------	-------

```

koicloud@ca-32190192:~/vscode$ gcc -o 32190192 32190192.c
koicloud@ca-32190192:~/vscode$ ./32190192 exception4.txt
(line 1) LW s0 0x2
32190192> Loaded 2 to s0
(line 2) LW t1 0x4
exception! LW의 레지스터는 s만 사용 가능합니다.
koicloud@ca-32190192:~/vscode$ 
  
```

추가! while문 테스트

```

int i = 0;
while (i != 2) {
    i++;
}
printf("%d", i);
  
```

위의 코드를 어셈블리어로 변환하여 JMP instruction이 위로도 점프가 가능한지 테스트 케이스를 만들어보았습니다. -> 위로 점프도 정확하게 출력되는 결과를 얻을 수 있었습니다.

while.txt	exception3.txt	exception4.txt	exception2.txt	exception1.txt
<pre> while.txt 1 LW s0 0x2 2 LW s1 0x1 3 ADD t1 s1 zero 4 BEQ s0 s2 0x8 5 ADD t0 s2 zero 6 ADD s2 t0 t1 7 JMP 0x4 8 ADD v0 s2 zero </pre>				
<pre> PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS koicloud@ca-32190192:~/vscode\$ gcc -o 32190192 32190192.c koicloud@ca-32190192:~/vscode\$./32190192 while.txt (line 1) LW s0 0x2 32190192> Loaded 2 to s0 (line 2) LW s1 0x1 32190192> Loaded 1 to s1 (line 3) ADD t1 s1 zero 32190192> Added s1(1) to zero(0) and changed t1 to 1 (line 4) BEQ s0 s2 0x8 32190192> Checked if s0(2) is not equal to s2(0) and didn't jumped to line 8 (line 5) ADD t0 s2 zero 32190192> Added s2(0) to zero(0) and changed t0 to 0 (line 6) ADD s2 t0 t1 32190192> Added t0(0) to t1(1) and changed s2 to 1 (line 7) JMP 0x4 32190192> jumped to line 4 (line 4) BEQ s0 s2 0x8 32190192> Checked if s0(2) is not equal to s2(1) and didn't jumped to line 8 (line 5) ADD t0 s2 zero 32190192> Added s2(1) to zero(0) and changed t0 to 1 (line 6) ADD s2 t0 t1 32190192> Added t0(1) to t1(1) and changed s2 to 2 (line 7) JMP 0x4 32190192> jumped to line 4 (line 4) BEQ s0 s2 0x8 32190192> Checked if s0(2) is equal to s2(2) and jumped to line 8 (line 8) ADD v0 s2 zero 32190192> Added s2(2) to zero(0) and changed v0 to 2 32190192> v0 is 2 koicloud@ca-32190192:~/vscode\$ </pre>				

6. Lesson

구조적 설계

과제를 듣고 맨 처음 든 생각이 시작할 때 구조를 신중하게 설계하는 것이 매우 중요하다고 생각했습니다.

1. Jmp 명령어를 처리하기 위해 1차원 배열보단 2차원 배열(arr)을 사용하여 명령어와 해당 오퍼랜드를 쉽게 저장하고 접근할 수 있었습니다.
2. 레지스터를 다룰 때, s, t, zero, v0레지스터를 따로 배열로 분리하려고 했지만 코드가 하드코딩될 우려가 있어서 offset 개념을 활용하여 하나의 배열로 관리했습니다.
3. 한 줄당 명령어 개수를 체크하기 위해서 row_cnt배열을 사용하여 파싱할 때 미리 배열에 저장해두어 예외처리를 쉽게 할 수 있었습니다. 이러한 접근은 코드의 가독성을 높이고 유지보수를 쉽게 해주었습니다.
4. base_pointer() 함수, move_offset()함수처럼 중복되는 로직들을 함수화 시켜 함수의 재사용성을 높여주었으며 코드가 가독성이 좋아지고 불필요한 코드를 제거할 수 있었습니다.

구조를 설계하는 데 많은 시간이 소요되었지만, 처음에 구조를 잘 설계하고 시작했기 때문에 구현은 비교적 빨리 진행될 수 있었습니다. 이 경험을 통해 앞으로도 문제에 직면했을 때 코드를 바로 작성하는 것이 아니라 구조를 깊게 고려하고 완벽하게 설계된 후에 코드를 작성하는 습관을 들여야 한다는 것을 배웠습니다.

MIPS에 대한 이해

구조 설계뿐만이 아니라 코드를 작성하면서 단순히 코드를 작성하는 것 이상으로, MIPS 아키텍처와 프로그램이 동작하는 방식에 대한 깊은 이해를 얻었습니다. 초기에는 고급 언어를 사용하여 프로그래밍하는 것이 보통이었지만, 이 프로젝트를 통해 어셈블리어를 통해 컴퓨터가 더 로우 레벨에서 동작하는 원리를 직접 이해하게 되었습니다.

예외처리의 필요성

평소에는 예외처리를 하지 않고 구현을 했지만, 이 과제를 할 때 예외 처리를 구현하며 코딩하는 경험을 했습니다. 이는 실제로 프로그래밍 과정을 매우 개선시킬 수 있다는 것을 깨달았습니다.

예외 처리 코드를 구현하면서 발견한 가장 큰 이점은 디버깅 과정에서의 편의성이었습니다. 이전에는 코드에서 오류가 발생했을 때 그 원인을 찾기 위해 많은 시간을 소비했었습니다. 하지만 예외 처리를 추가하고 나서는 오류가 발생했을 때 바로 그 위치를 찾아낼 수 있었습니다. 이는 디버깅 과정을 훨씬 빠르고 효율적으로 만들어주었습니다..

또한, 예외 처리를 통해 코드의 가독성과 유지보수성도 향상됩니다. 코드에 예외 처리를 명시적으로 추가하면 코드의 흐름이 명확해지고 각 예외 상황에 대한 처리 방법이 명시적으로 드러나기 때문입니다. 또한, 이를 통해 다른 개발자가 코드를 이해하고 수정하는 데도 도움이 됩니다.

이러한 경험을 토대로 저는 앞으로도 예외 처리를 고려하여 코딩하는 습관을 들일 것입니다.