Simple Calculator

32214362 모바일시스템공학과 조윤서

Introduction

본 프로젝트에서는 하드웨어(프로세서)와 소프트웨어의 접점인 ISA(Instruction Set Architecture)를 디자인하고 구현하여 간단한 계산기, 즉, 나만의 컴퓨터를 구현하였다. 여기서 더 나아가 두 수의 최대공약수(GCD)를 계산하는 프로그램을 만들어봄으로써 설계된 ISA가 실제 문제 해결에 어떻게 활용될 수 있는지 알아보았다.

Important Concept

Opcode	Operation
ADD	두 수를 더한다. 결과는 R0에 저장된다.
SUB	첫 번째 피연산자에서 두 번째 피연산자를 뺀다. 결과는 R0에 저장된다.
MUL	두 수를 곱한다. 결과는 R0에 저장된다.
DIV	첫 번째 피연산자를 두 번째 피연산자로 나눈다. 결과는 R0에 저장된다.
MOV	한 레지스터의 값을 다른 레지스터로 이동하거나, 특정 값을 레지스터에 저장한다.
HLT	프로그램 실행을 중지한다.
CMP	두 피연산자를 비교하고, 결과에 따라 R0에 -1, 0, 또는 +1을 저장한다.
JMP	주어진 줄 번호로 점프한다.
BEQ	R0가 0일 경우에만 주어진 줄 번호로 점프한다.

Implementation Process

Step1: Basic Calculator

제일 먼저 기본 사칙연산 기능을 제공하는 간단한 계산기 구현부터 시작했다.

Step2: File-based Input Calculator

사용자 입력 대신 파일로부터 입력을 받는 계산기로 확장하였다.

Step3: Register Handling Calculator

숫자뿐만 아니라 레지스터 참조값도 처리할 수 있는 계산기로 확장하였다.

Step4: Advanced Instructions

'M', 'HLT', 'C', 'J', 'BEQ'와 같은 명령어도 추가적으로 구현하였다.

Step5: Program Counter

'J' 명령어 구현 과정에서 program counter(pc)의 필요성을 깨닫고 pc를 추가하였다.

Step6: Memory-based Execution

pc사용을 위해 명령어 세트를 메모리에 미리 로드하여 순차적 실행 방식을 도입하였다.

Step7: Code Readability and Maintenance

'enum' 사용으로 연산자(opcode)의 가독성을 향상시켰고, 구조체 도입으로 연산자(opcode), 피연산자1(operand1), 피연산자2(operand2)를 한꺼번에 관리함으로써 데이터 관리를 간소화하였다. 또한 기능별 함수 분리로 코드를 세분화하여 관리 용이성을 향상시켰다.

```
38    Opcode get_opcode(const char *op);
39    int load_instructions(const char *filename);
40    void execute_instruction(Instruction instruction);
41    void execute_instructions(int num_instructions);
```

본 프로그램은 main함수를 제외하고 크게 4개의 기능으로 나누었다.

'get_opcode' 함수는 문자열로 주어진 연산자 또는 명령어를 내부적으로 정의된 'Opcode' 열거형으로 반환하는 역할을 한다. 이는 프로그램이 문자열 형태의 명령어를 실제 실행 가능한 코드로 인식할 수 있게 하는 기능을 수행한다.

'load_instructions' 함수는 파일에서 명령어들을 읽어와서 이를 파싱하는 역할을 한다. 파싱된 명령어들은 프로그램의 메모리 상에 저장되어 실행 단계에서 사용된다. 이 함수는

프로그램이 작동하기 위해 필요한 초기 데이터를 준비하는 과정을 담당한다.

‘execute_instruction’함수는 개별 명령어를 실행하는 역할을 한다. 이 함수는 명령어에 따라 피연산자(숫자나 레지스터 참조값)들을 적절하게 처리하고, 계산 결과를 레지스터에 저장하거나 프로그램의 실행 흐름을 제어한다.

‘execute_instructions’ 함수는 명령어들을 순차적으로 실행하면서 프로그램 카운터(pc)의 이동을 관리한다. 이 함수는 프로그램의 전체적인 실행 흐름을 제어하며, pc값에 따라 다음에 실행할 명령어를 결정한다.

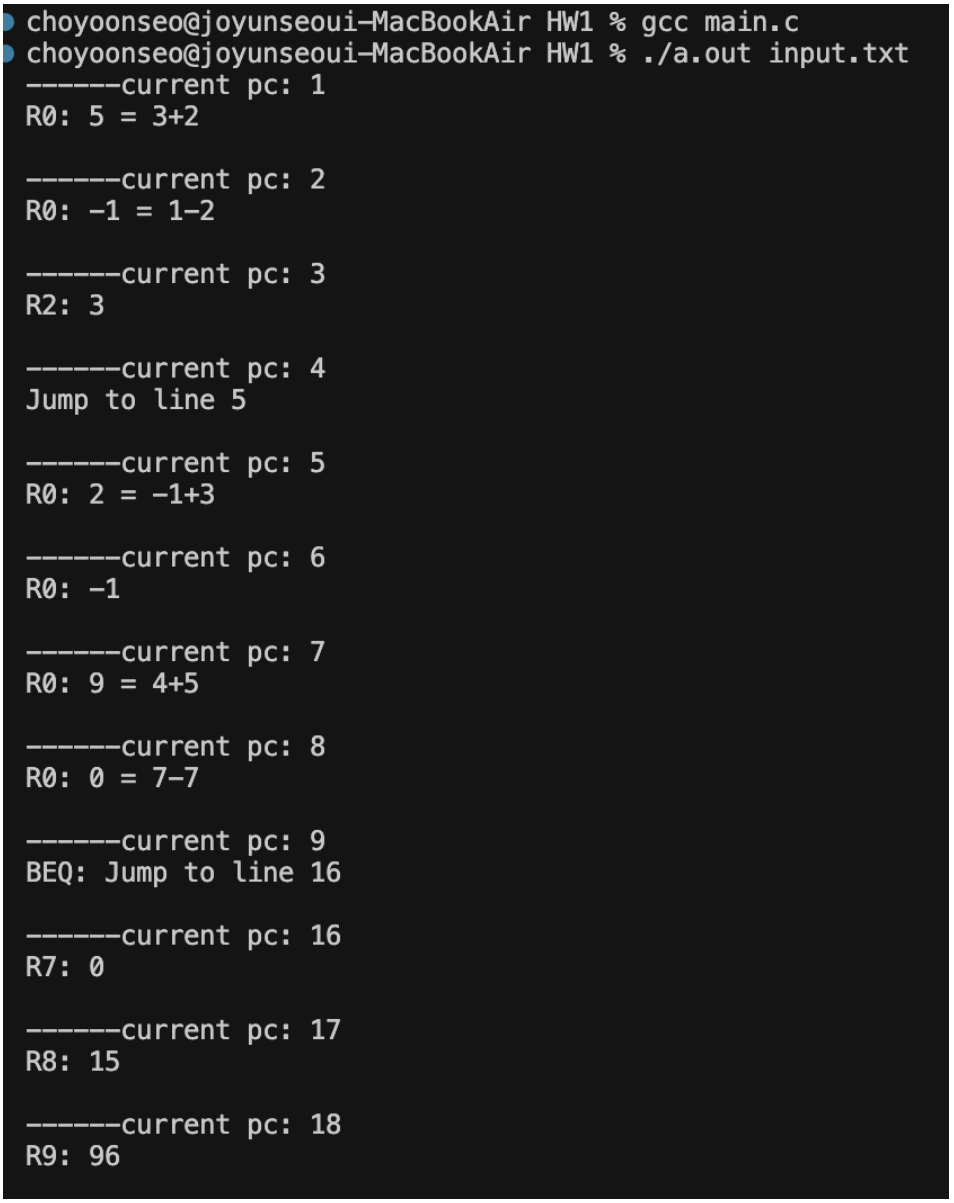
Build Configuration / Environment

- Development Environment: **Visual Studio Code**
- Programming Language: **C**

- Build Configuration:
  - ‘main.c’ 컴파일: ‘**gcc main.c -o calculator**’
  - 실행 파일과 ‘input.txt’ 함께 실행: **‘./calculator input.txt**’

- GCD program Build Configuration
  - ‘main.c’ 컴파일: ‘gcc main.c -o gcd’
  - 실행 파일과 ‘gcd3.txt’ 함께 실행: ‘./gcd gcd3.txt’

Screen Capture / Working Proofs



<input.txt 파일의 실행 결과**>**

```
choyoonseo@joyunseoui-MacBookAir HW1 % gcc main.c -o gcd3
choyoonseo@joyunseoui-MacBookAir HW1 % ./gcd3 gcd3.txt
------current pc: 1
R1: 15

------current pc: 2
R2: 96

------current pc: 3
R0: -1

------current pc: 4
R0 is not 0. Continue to the next instruction

------current pc: 5
R3: 1

------current pc: 6
R0: -1

------current pc: 7
R0 is not 0. Continue to the next instruction

------current pc: 8
R0: 81 = 96-15

------current pc: 9
R2: 81

------current pc: 10
Jump to line 3

------current pc: 3
R0: -1

------current pc: 4
R0 is not 0. Continue to the next instruction

------current pc: 5
R3: 1

------current pc: 6
R0: -1

------current pc: 7
R0 is not 0. Continue to the next instruction

------current pc: 8
R0: 66 = 81-15

------current pc: 9
R2: 66

------current pc: 10
Jump to line 3

------current pc: 3
```

<gcd3.txt의 실행결과**1>**

```
------current pc: 4
R0 is not 0. Continue to the next instruction

------current pc: 5
R3: 1

------current pc: 6
R0: -1

------current pc: 7
R0 is not 0. Continue to the next instruction

------current pc: 8
R0: 51 = 66-15

------current pc: 9
R2: 51

------current pc: 10
Jump to line 3

------current pc: 3
R0: -1

------current pc: 4
R0 is not 0. Continue to the next instruction

------current pc: 5
R3: 1

------current pc: 6
R0: -1

------current pc: 7
R0 is not 0. Continue to the next instruction

------current pc: 8
R0: 36 = 51-15

------current pc: 9
R2: 36

------current pc: 10
Jump to line 3

------current pc: 3
R0: -1

------current pc: 4
R0 is not 0. Continue to the next instruction

------current pc: 5
R3: 1

------current pc: 6
```

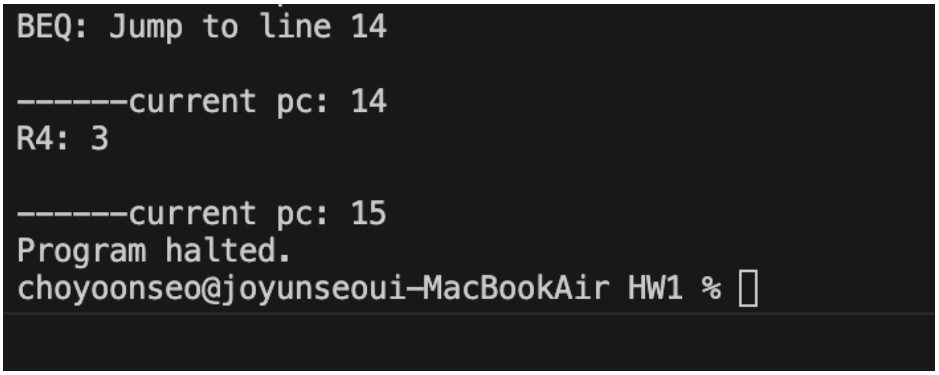
<gcd3.txt의 실행결과**2>**

```
R0: -1

------current pc: 7
R0 is not 0. Continue to the next instruction

------current pc: 8
R0: 21 = 36-15

------current pc: 9
R2: 21

------current pc: 10
Jump to line 3

------current pc: 3
R0: -1

------current pc: 4
R0 is not 0. Continue to the next instruction

------current pc: 5
R3: 1

------current pc: 6
R0: -1

------current pc: 7
R0 is not 0. Continue to the next instruction

------current pc: 8
R0: 6 = 21-15

------current pc: 9
R2: 6

------current pc: 10
Jump to line 3

------current pc: 3
R0: +1

------current pc: 4
R0 is not 0. Continue to the next instruction

------current pc: 5
R3: 1

------current pc: 6
R0: 0

------current pc: 7
BEQ: Jump to line 11

------current pc: 11
R0: 9 = 15-6

------current pc: 12
```

<gcd3.txt의 실행결과**3>**

```
R1: 9

------current pc: 13
Jump to line 3

------current pc: 3
R0: +1

------current pc: 4
R0 is not 0. Continue to the next instruction

------current pc: 5
R3: 1

------current pc: 6
R0: 0

------current pc: 7
BEQ: Jump to line 11

------current pc: 11
R0: 3 = 9-6

------current pc: 12
R1: 3

------current pc: 13
Jump to line 3

------current pc: 3
R0: -1

------current pc: 4
R0 is not 0. Continue to the next instruction

------current pc: 5
R3: 1

------current pc: 6
R0: -1

------current pc: 7
R0 is not 0. Continue to the next instruction

------current pc: 8
R0: 3 = 6-3

------current pc: 9
R2: 3

------current pc: 10
Jump to line 3

------current pc: 3
R0: 0

------current pc: 4
```

<gcd3.txt의 실행결과**4>**



<gcd3.txt의 실행결과**5>**

Trial & Errors

- Program Counter(PC)

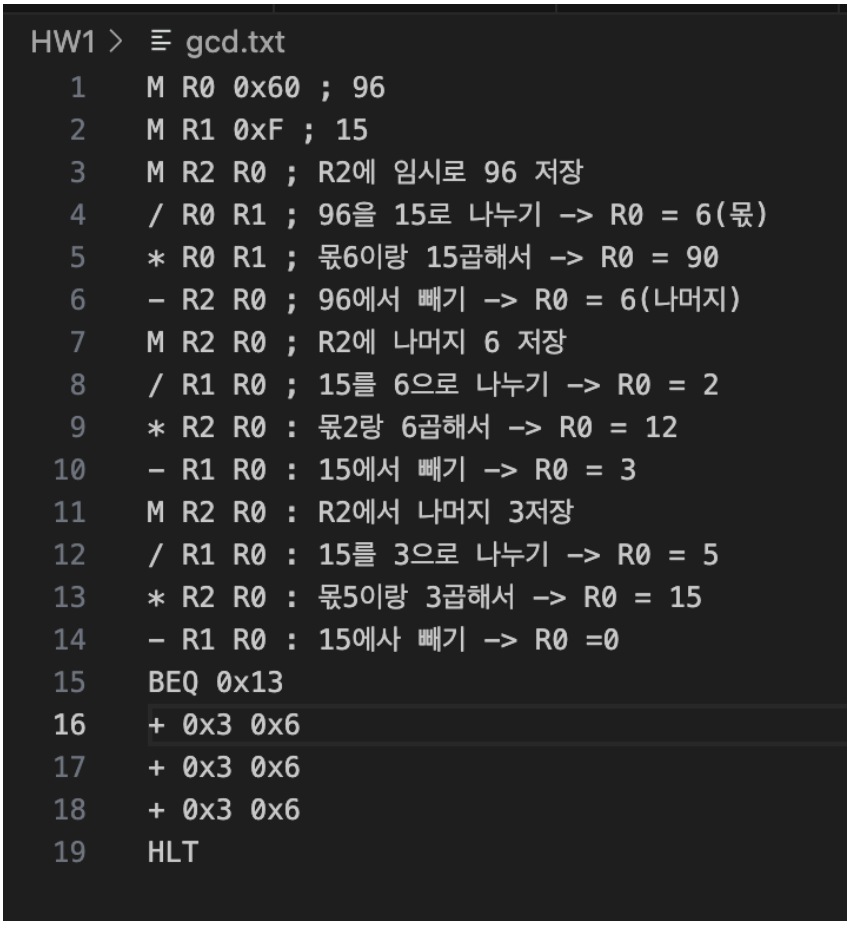
'JMP' 명령어 구현 과정에서 특정 line으로의 이동을 가능케 하는 program counter(pc)의 필수성을 인지하게 되었고, 이에 따라 pc 관리 기능을 추가로 도입하였다.

- Instruction Processing

이전까지는 명령어를 하나씩 입력 받아 처리하는 절차적 방식으로 구현하였으나, JUMP 명령어의 원활한 실행을 위해서 전체 명령어 세트를 메모리에 미리 로드하고 순차적으로 실행하는 구조로 전환하였다.

- GCD Implementation

유클리드 알고리즘을 바탕으로 최대공약수를 구하는 프로그램을 처음에 짜려고 했으나, 본인이 디자인한 ISA에는 나머지 연산자가 없었기 때문에 오직 기본 산술 연산만을 사용하여 gcd를 계산하는 방법에 중점을 두어 구현하였다. 다음은 최대공약수(GCD) 프로그램 구현을 위한 시도과정이다.



<gcd.txt>

Trial 1: 오직 15와 96의 최대공약수만 구할 수 있는 프로그램을 만들었다. 해당 프로그램은 다른 수의 최대공약수는 구하지 못한다.

HW1 > gcd2.txt

```
1   M R1 0x60 ; R1에 96 저장
2   M R2 0xF ; R2에 15 저장
3   C R1 R2 ; R1과 R2 값이 같은지, 누가 더 큰 지 비교
4   BEQ 0x15 ; 애초에 R1 R2가 같으면 종료
5   - R0 0x1 ; 근데 R0가 +1이라면, 즉, op1>op2라면, R0를 0으로 만들어서
6   BEQ 0xA ; 바로 뺄셈으로 가기
7   M R3 R1 ; R1 < R2라면 본격적인 뺄셈 전, swap 과정 하기 - R1을 임시로 R3에 저장
8   M R1 R2 ; R2값을 R1에 저장
9   M R2 R3 ; R3값을 R2에 저장
10  - R1 R2 ; R0 = 96-15=81
11  M R1 R0 ; 결과와 원래 작은 수(15)를 새로운 두 수로 취급
12  C R1 R2 ; R0 = +1이면, 즉, op1 > op2이면
13  BEQ 0x15 ; R1, R2값이 같으면 종료, 그렇지 않다면
14  - R0 0x1 ; R0 = 0을 만들어서
15  BEQ 0xA ; 뺄셈 반복
16  M R3 R1 ; 결과(6)가 원래 작은 수(15)보다 작아지면, 결과(6)을 잠시 R3에 저장
17  M R1 R2 ; R0 = -1이면, 즉, op1 < op2이면 원래 작은 수는 R2가 아닌 R1이 됨
18  M R2 R3 ; R3값을 원래 작은 수를 보관하는 R2에 저장
19  + R0 0x2 ; -2인 R0에 2를 더해서
20  BEQ 0xA ; 10번째 줄부터 뺄셈 다시 반복
21  HLT
```

<gcd2.txt>

Trial 2: 15와 96뿐만 아니라 다른 수의 최대공약수도 구할 수 있는 프로그램을 만들었다.

<gcd3.txt>

Trial 3: 두 번째 시도를 최적화하였다.

Lesson

사실 처음에 ISA라는 개념에 대한 모호함으로 인해 무엇을 어떻게 구현할 지에 대해 상당한 막막함을 느꼈다. 그러나 컴퓨터의 본질이 계산기라는 사실과 계산기의 기반인 사칙연산에서 출발한다는 것을 깨닫고 기본적인 사칙연산 기능을 갖춘 계산기 구현에서 출발하기로 결정했다. 초기 코드를 작성한 후 점차 구현에 대한 감을 잡아갔고, 이후 사용자가 아닌 파일을 통해 입력을 받는 사칙연산 계산기로의 확장, 그리고 숫자뿐만 아니라 레지스터 참조값을 처리하는 계산기로의 발전을 이루며 주어진 모든 요구사항을 충족할 수 있었다.

비록 이번에는 체계 없이 구현하였지만, 다음 프로젝트부터는 전체 구조에 대한 개략적인 계획을 세우고 구현할 것이다. 왜냐하면 이번 프로젝트를 마친 후 보완하고 싶은 부분이 많이 발견되었으나, 일부분만을 수정하려 해도 전체적인 구조를 재검토해야하는 어려움이 있었기 때문이다. 그래서 다음 프로젝트부터는 수정과 유지보수가 용이하도록 더 체계적인 코드를 작성할 것이다.