**CPU가상 시뮬레이터 프로그램 최종보고서**

**공과대학 컴퓨터공학과**

**200701322 안종현**

[1. 개요 1](#_Toc375082502)

[1.1 개발배경 및 동기 1](#_Toc375082503)

[1.2 프로그램 개요 1](#_Toc375082504)

[2. 프로젝트 구성 2](#_Toc375082505)

[2.1 알고리즘 2](#_Toc375082506)

[2.2 구현 2](#_Toc375082507)

[2.3 개발환경 3](#_Toc375082508)

[3. 어셈블리어 규칙 4](#_Toc375082509)

[3.1 System Regulation 4](#_Toc375082510)

[3.2 Instruction Format 5](#_Toc375082511)

[4. Path Diagram 8](#_Toc375082512)

[4.1 A-Type Path Diagram (ADD, SUB, MUL, DIV) 8](#_Toc375082513)

[4.2 L-Type Path Diagram (MOV) 8](#_Toc375082514)

[4.3 L-Type Path Diagram (AND, ORR) 9](#_Toc375082515)

[4.4 L-Type Path Diagram (CMP) 9](#_Toc375082516)

[4.5 M-Type Path Diagram (LDR) 10](#_Toc375082517)

[4.6 M-Type Path Diagram (STR) 10](#_Toc375082518)

[4.7 B-Type Path Diagram (B, BL, IRET) 11](#_Toc375082519)

[4.8 S-Type Path Diagram (PUSH) 11](#_Toc375082520)

[4.9 S-Type Path Diagram (POP) 12](#_Toc375082521)

[4.10 Combination Path Diagram 12](#_Toc375082522)

[5. CLU & MM 13](#_Toc375082523)

[5.1 CLU 13](#_Toc375082524)

[5.2 Memory Map 13](#_Toc375082525)

[6. Sample Code 14](#_Toc375082526)

[5.1 Demonstration Result 14](#_Toc375082527)

[5.2 Module Test Code 14](#_Toc375082528)

[5.3 Data Test Code 15](#_Toc375082529)

[5.4 Push-Pop Test Code 15](#_Toc375082530)

[7. 발전방향 16](#_Toc375082531)

[A. 참고문헌 17](#_Toc375082532)

1. 개요

# 1.1 개발배경 및 동기

시스템 구조에서 CPU를 직접 설계하고 구현해봄으로써 동작원리, 구조, 성능을 이해하고 소프트웨어적/하드웨어적 관점 향상 도모를 목적으로 본 주제를 선정하였다.

# 1.2 프로그램 개요

본 프로그램은 C언어를 사용하였으며, 16bit의 assembly언어체계를 재정의하여 동작하도록 구현하였다. CPU구조는 Single Cycle 형식의 Circuit Diagram을 따르며, 각 Instruction은 다섯 가지의 종류로 구별한다. CPU를 각 Component별로 구분하여 Instruction Memory, Data Memory, Stack Memory, Register Bank, CLU, CPSR, ALU로 표현하였다.

2. 프로젝트 구성

# 2.1 알고리즘

**가. 16bit Instruction Set 구성도**

1. 각 명령어는 16bit 크기를 가진다.
2. 명령 타입은 A(Arithmetic), L(Logic), M(Memory), B(Branch), S(Stack Operation)가 있으며 각 타입마다 명령을 수행하기 위한 구성요소가 다르게 존재한다.
3. 각 타입은 바이너리 해석 포맷자체가 동일한 것으로 Opcode 별로 행하는 CLU 제어신호가 다르다.
4. CPU는 Opcode로 Instruction Type을 판별하여 정해진 규칙에 따라 명령어를 분석하고 그에 맞는 신호와 데이터를 주게 된다.

**나. 각 타입 별 안산구조**

1. A-Type: Opcode(4) | RD(4) | R1(4) | R2(4)
2. L-Type: Opcode(4) | RD(4) | F(2) | Operand(6)
3. M-Type: Opcode(4) | RD(4) | Address(4) | Offset(4)
4. B-Type: Opcode(4) | Type(4) | CPSR-Flag(4) | Address(4)
5. S-Type: Opcode(4) | Reversed(4) | Flag(2) | Reversed(4) | Operand(4)

※ R1, R2, RD는 Register Index를 가리키는 값이며, 자세한 사항은 3.2 참조

**다. 각 명령어 수행 구조**

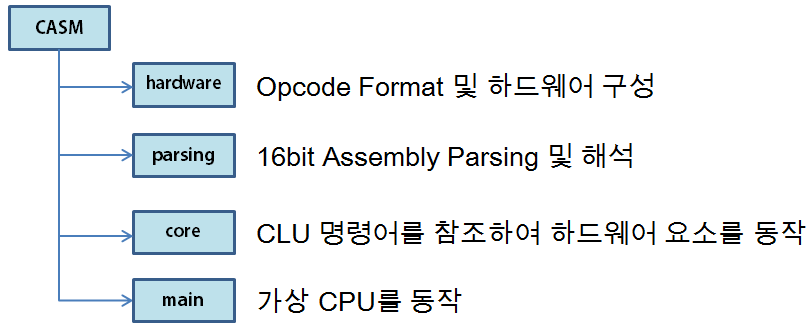
1. A-Type: ADD, SUB, MUL, DIV
2. L-Type: MOV, AND, OR, CMP
3. M-Type: LDR, STR
4. B-Type: B, BL, IRET
5. S-Type: PUSH, POP

# 2.2 구현

**가. Block Diagram**

1. 각 명령어 별로 16bit Architecture 형식의 single CPU 블록 다이어그램을 그린다.
2. CPU구성은 Memory(Instruction, Data, Stack), Register Bank (a1-a4, v1-v8, ip, sp, lr, pc), CPSR Register, CLU, ALU를 사용하여 표현한다.

**나. Software Diagram(Using repository:** [**github.com/eldora/CASM**](https://github.com/eldora/CASM)**)**



# 2.3 개발환경

1) OS: Ubuntu 12.04LTS

2) Compiler: GCC 4.6

3) Tool: Make, Vim, ctags

3. 어셈블리어 규칙

# 3.1 System Regulation

**가. CPU State Register**

|  |  |  |
| --- | --- | --- |
| CPSR | Set(1) | Clear(0) |
| [Z] | Equal | Not Equal |
| [N] | Negative Number | 0 or Positive Number |
| [V] | Overflow | No Overflow |

Table 1. CPU CPSR Register Regulation

**나. Register Usage Map**

|  |  |  |  |
| --- | --- | --- | --- |
| Name | Register Number | Usage | 함수호출 후  값 보존여부 |
| a1-a2 | 0-1 | 인수/결과값/스크래치 레지스터 | X |
| a3-a4 | 2-3 | 인수/스크래치 레지스터 | X |
| v1-v8 | 4-11 | 지역 루틴용 변수 | O |
| ip | 12 | 프로시저 내 스크래치 레지스터 | X |
| sp | 13 | 스택 포인터 | O |
| lr | 14 | 링크 레지스터(복귀 주소) | O |
| pc | 15 | 프로그램 카운터 | n.a |

Table 2. Register Usage Regulation

# 3.2 Instruction Format

**가. A-Type Description (ADD, SUB, MUL, DIV)**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Opcode | Rd | R1 | R2 | Usage |
| 4bit | 4b | 4b | 4b | Register |

Table 3. A-Type Instruction Format

1. Opcode는 ADD(0000), SUB(0001), MUL(0010), DIV(0011)를 가진다.
2. Rd, R1, R2는 Register Bank의 인덱스이다.
3. Rd = R1 + R2 형식으로 사용된다.

**나. L-Type Description (MOV, AND, ORR, CMP)**

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| F=? | Op | Rd | F | S | Rev | Operand | Usage | Detail |
| 0 | 4b | 4b | 2b | 0b | 2b | 4b | Register | Register Index |
| 1 | 4b | 4b | 2b | 0b | 0b | 6b | Immediate | 6bit Constant |
| 2 | 4b | 4b | 2b | 2b | 0b | 4b | Shift | Operand<<4\*S |

Table 4. L-Type Instruction Format

1. Opcode는 MOV(0100), AND(0101), ORR(0110), CMP(0111)를 가진다.
2. L-Type은 F의 값에 따라 Operand가 유동적으로 변화한다.
3. F=0: Operand를 Register로 사용한다.
4. F=1: Operand를 2^6 범위의 Immediate Value로 사용한다.
5. F=2: S(2bit)를 Shift Bit로 사용하여 Operand(4bit) << 4\*S(2bit)로 사용한다.
6. FORMAT은 다음과 같다. MOV: Rd = Operand, AND: Rd &= Operand,   
   ORR: Rd |= Operand, CMP: Rd == Operand

**다. M-Type Description (LDR, STR)**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Op | Rd | Rn | Operand | Usage | Detail |
| 4b | 4b | 4b | 4b | Register | LDR: Rd = [Rn(BaseAddr)+Operand(Offset)]  STR: Rn(BaseAddr)+Operand(Offset) = [Rd] |

Table 5. M-Type Instruction Format

1. Opcode는 LDR(1000), STR(1001) 를 가진다.
2. Rd, Rn은 Register Bank의 인덱스이고, Operand는 Immediate Value이다.
3. Rn번째 레지스터의 값을 주소 값으로 사용하여 Operand만큼의 변위 값을 사용하여 주소 값을 만든 뒤 해당주소의 있는 내용을 저장하거나 불러온다.

**라. B-Type Description (B, BL, IRET)**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Opcode | Type | CPSR Flag | Rd(Address) | Detail |
| 4bit | 4b | 4b | 4b | mov pc, lr |

Table 6. B-Type Instruction Format

|  |  |  |
| --- | --- | --- |
| Type | CPSR F | Detail |
| 0 | 0 | B |
| 1 | 0 | BL |
| 2 | 0 | IRET |
| 3 | 0 | Rev |

|  |  |  |  |
| --- | --- | --- | --- |
| Type | CPSR Flag | Suffix | Condition |
| 0 | 0 | NONE | None |
| 0 | 10 | EQ | [Z] Set |
| 0 | 11 | NE | [Z] Clear |
| 0 | 100 | GT | [Z] Clear && (N=V) |
| 0  Table 6.1. Type Detail Options | 101 | LT | [N] ≠ [V] |
| 0 | 110 | GE | [N] = [V] |
| 0 | 111 | LE | [Z] Set || ([N]≠[V]) |

**Table 6.2. CPSR Flag Detail Options**

1. Opcode는 LDR(1000), STR(1001) 를 가진다.
2. Rd, Rn은 Register Bank의 인덱스이고, Operand는 Immediate Value이다. Rn번째 레지스터의 값을 주소 값으로 사용하여 Operand만큼의 변위 값을 사용하여 주소 값을 만든 뒤 해당주소의 있는 내용을 저장하거나 불러온다.

**마. S-Type Description (PUSH, POP)**

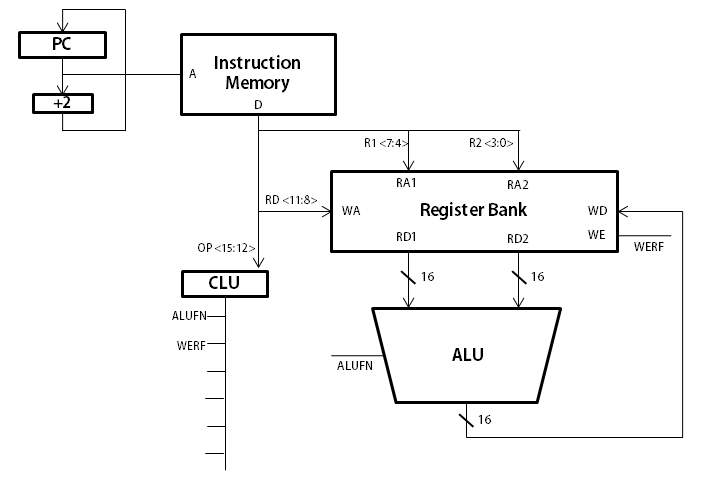
|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| F=? | Op | Rev | F | Rev | Operand | Usage | Detail |
| 0 | 4b | 4b | 2b | 2b | 4b | Register | Register Push/Pop |
| 1 | 4b | 4b | 2b | 2b | 4b | No Use | Reg.v1-v8 Push/Pop |
| 2 | 4b | 4b | 2b | 2b | 4b | No Use | Reg.All Push/Pop |

Table 7. S-Type Instruction Format

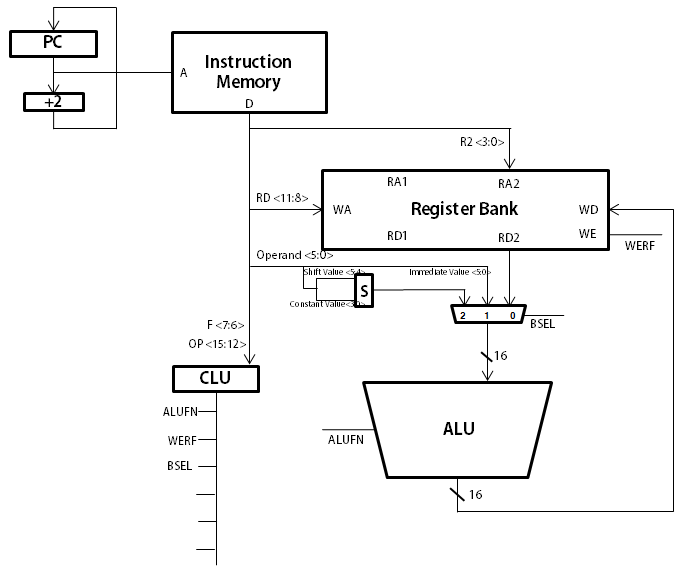
1. Opcode는 PUSH(1011), POP(1100) 를 가진다.
2. S-Type은 F의 값에 따라 다양한 형식은 PUSH/POP을 제공한다.
3. F=0: Operand를 인덱스로 갖는 단일 레지스터를 PUSH/POP한다.
4. F=1: 지역 레지스터로 사용하는 v1-v8 레지스터를 PUSH/POP한다.
5. F=2: 모든 레지스터를 PUSH/POP한다.

4. Path Diagram

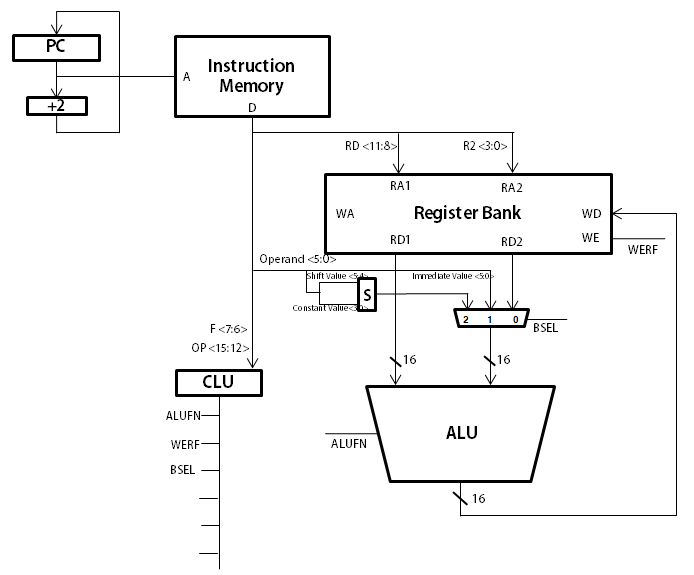
# 4.1 A-Type Path Diagram (ADD, SUB, MUL, DIV)



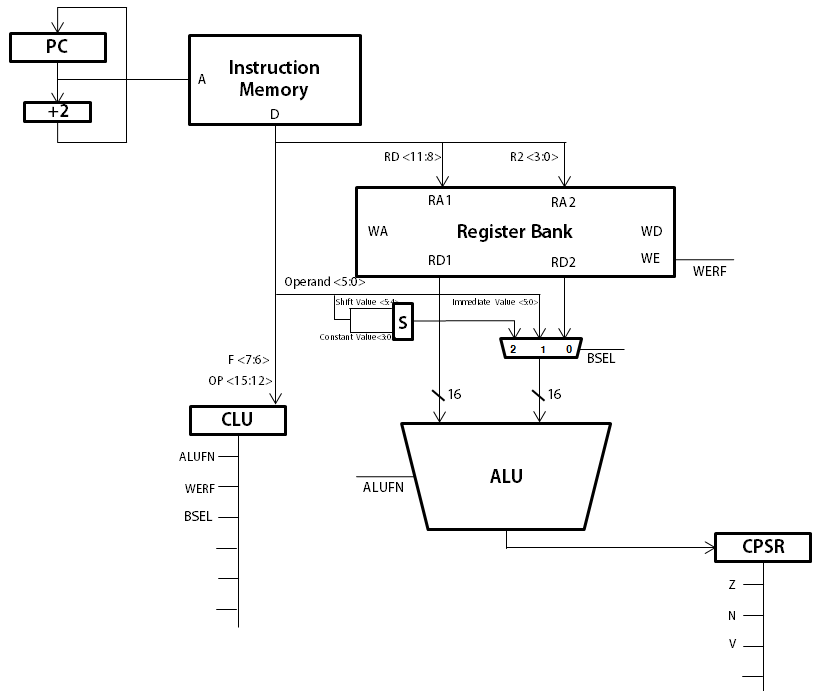
# 4.2 L-Type Path Diagram (MOV)



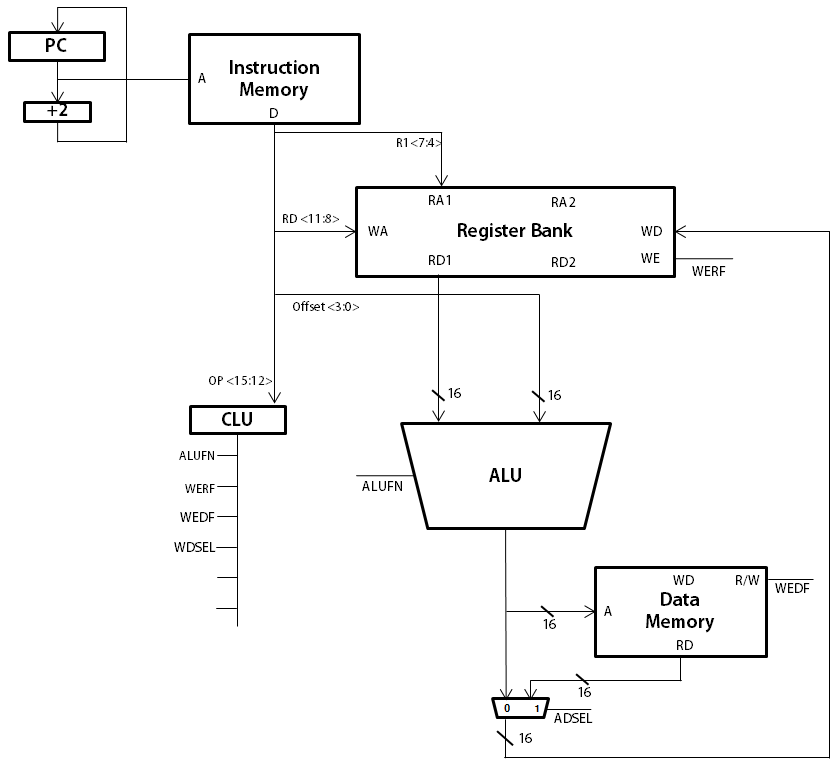
# 4.3 L-Type Path Diagram (AND, ORR)



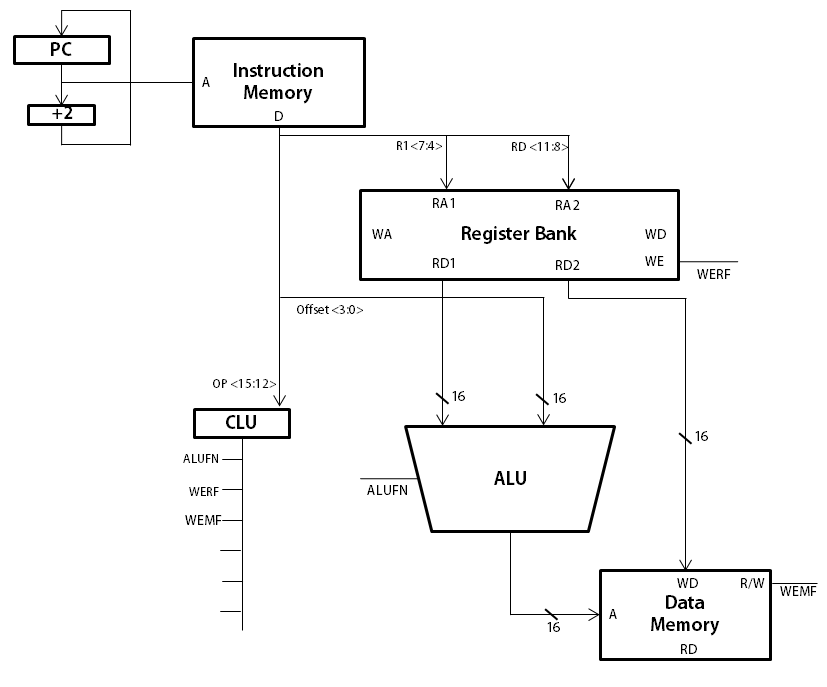
# 4.4 L-Type Path Diagram (CMP)



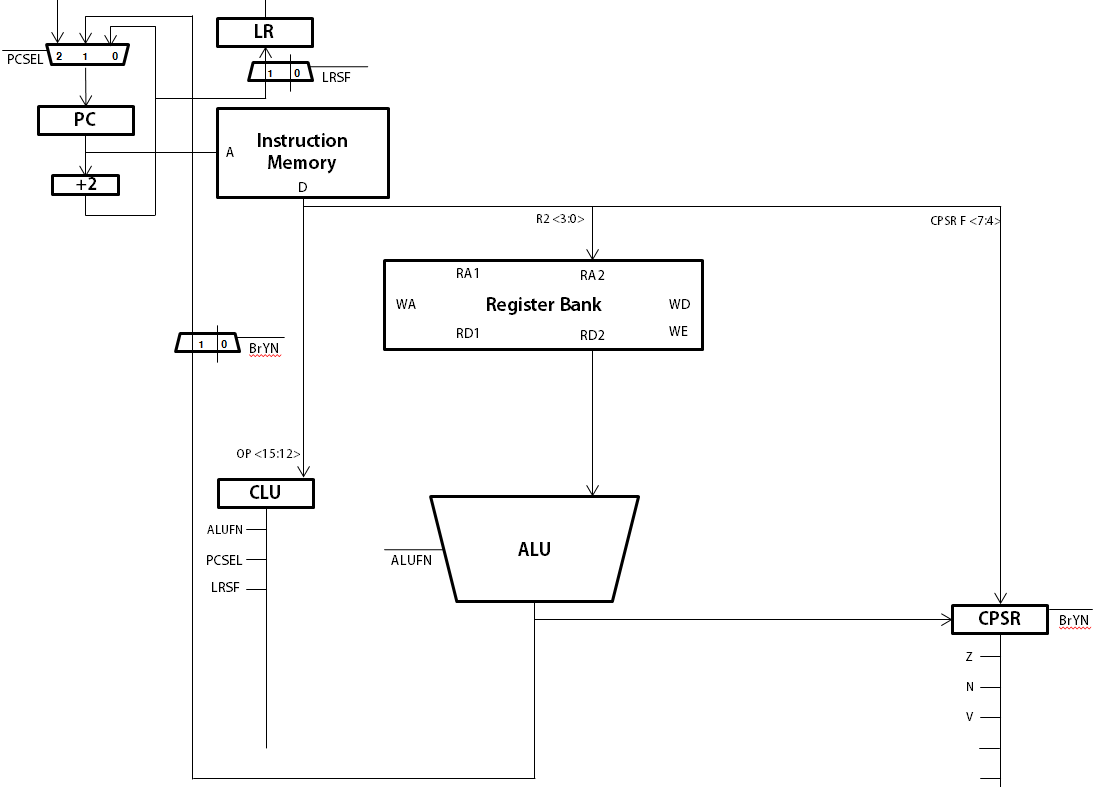
# 4.5 M-Type Path Diagram (LDR)



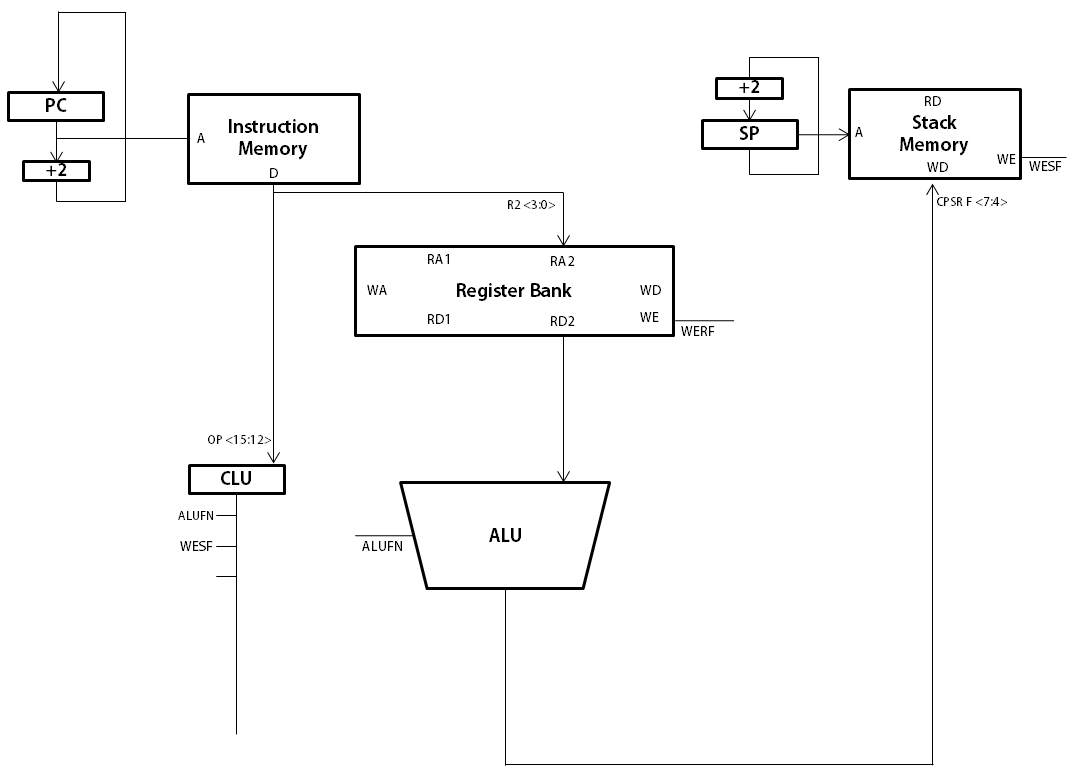
# 4.6 M-Type Path Diagram (STR)



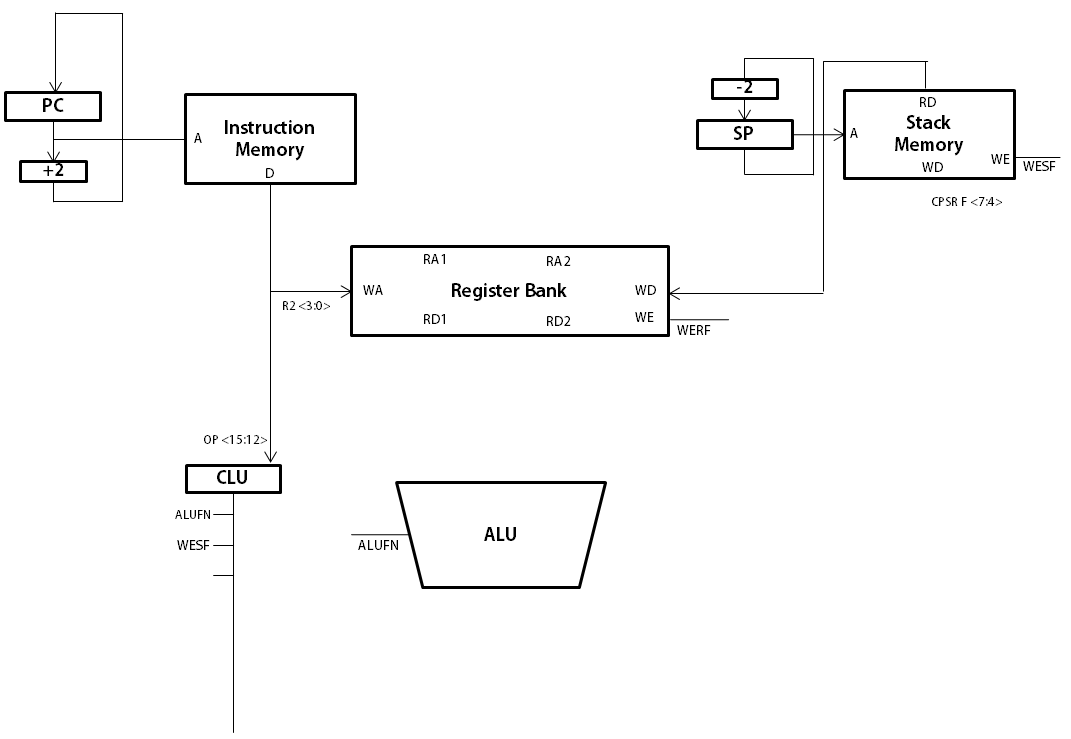
# 4.7 B-Type Path Diagram (B, BL, IRET)



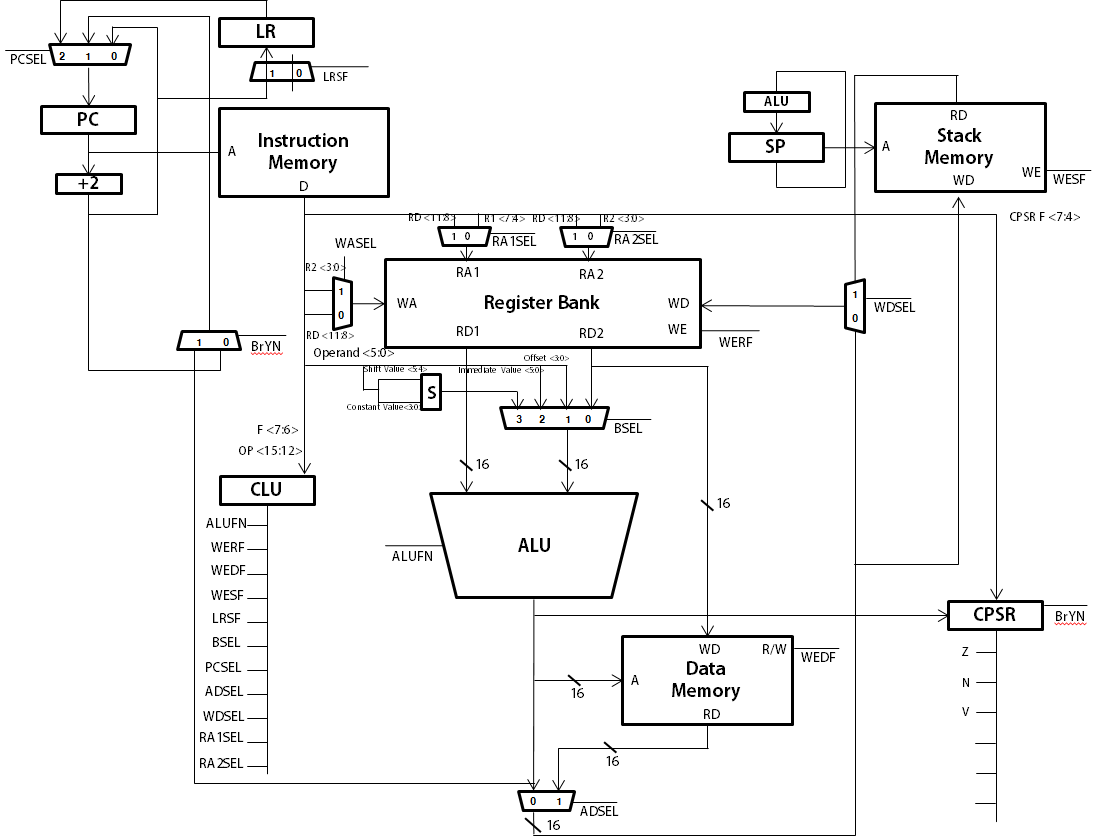
# 4.8 S-Type Path Diagram (PUSH)



# 4.9 S-Type Path Diagram (POP)



# 4.10 Combination Path Diagram



5. CLU & MM

# 5.1 CLU

CLU Table를 기본으로 하여 CPU가 동작한다. 각 Opcode에 따른 CLU 값들을 배열로 표현하였다. U는 Unused이며, R/W, 숫자는 해당 Mux의 값으로 나타내었다.

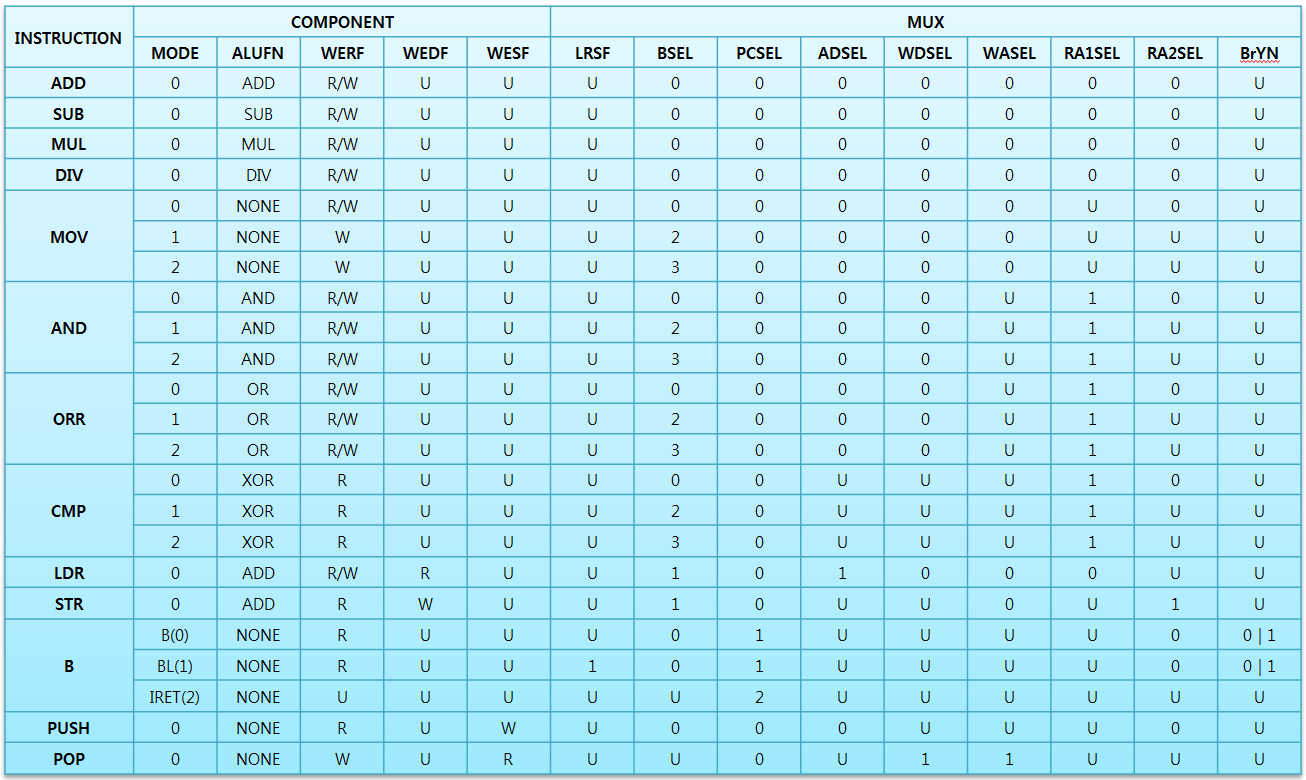


Figure . CLU Table

# 5.2 Memory Map

소스파일은 하나의 파일만 되어있지 않고 여러 파일로 Input이 될 수 있다. 때문에 Parsing단계에서 각 코드들을 메모리에 올릴 때 함수들의 위치를 기억하기 위한 Memory Map을 구현하였다. Memory Map 컴포넌트는 함수의 이름과 메모리상의 Index를 멤버변수로 갖는다.

6. Sample Code

# 5.1 Demonstration Result

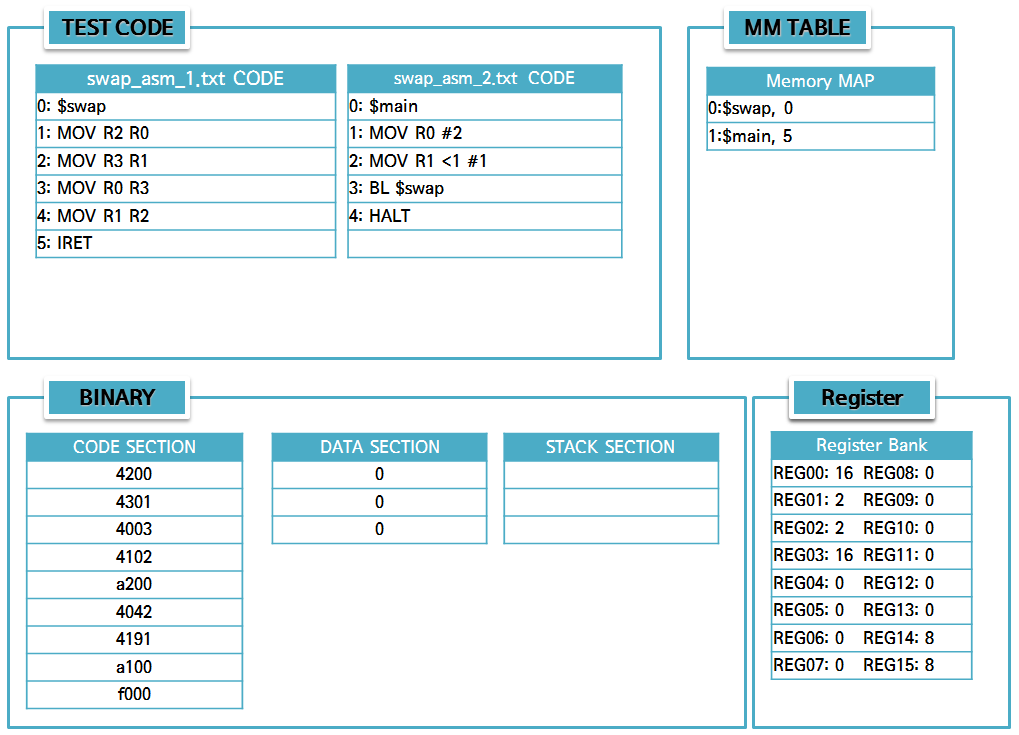
구현한 프로그램이 올바르게 동작하는지 테스트하기 위해 입증할 수 있는 3분류의 코드를 작성했다. 아래는 테스트 코드를 입력 값으로 실행한 결과값이다. 프로그램 Sample Code 실행 명령어는 아래와 같다.

1) make – 컴파일과 함께 Module Test Code로 구동

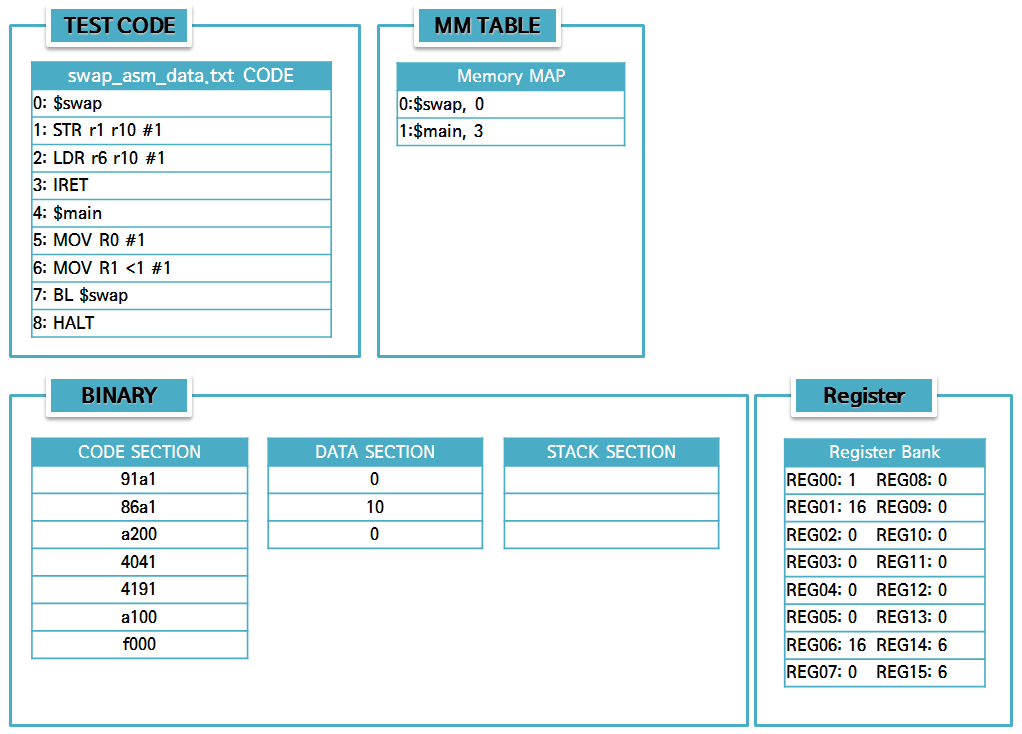
2) make data – Data Test Code로 구동

3) make pushpop – Push-Pop Test Code로 구동

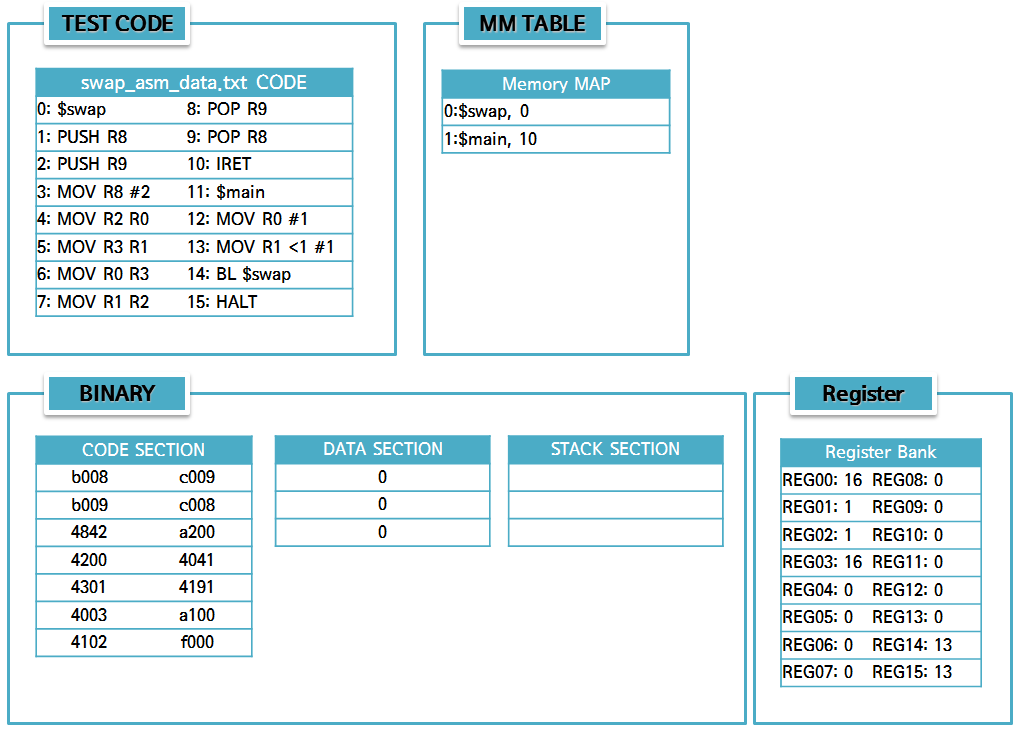
# 5.2 Module Test Code



# 5.3 Data Test Code



# 5.4 Push-Pop Test Code



7. 발전방향

가상 CPU 프로그램이 Assembly명령어를 해석하여 선언해 놓은 CLU Table를 바탕으로 컴포넌트들의 동작흐름을 가시적으로 보여줄 수 있는 GUI 기능을 제공하면 교육적인 측면으로써도 많은 도움이 될 것이라 생각된다.

A. 참고문헌

[1] Computer Organization and Design, Fourth Edition: The Hardware/Software Interface

[2] MIT Open Course Ware - Computation Structures, Spring 2009