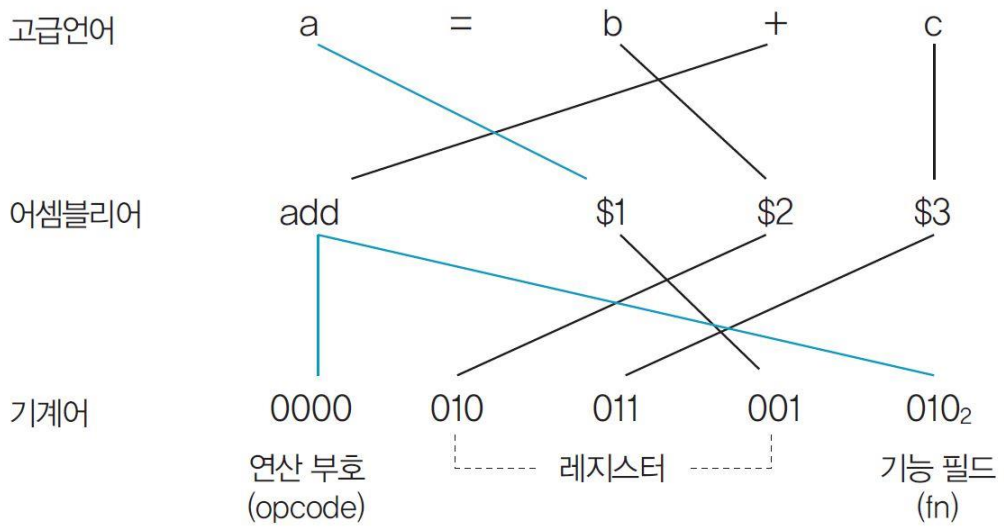
목차

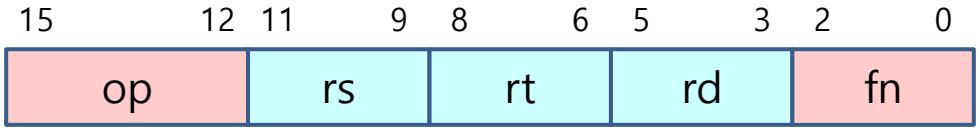
1. picoMIPS 컴퓨터

- **32bit MIPS 아키텍처를 모방한 16bit 아키텍처 (워드 길이 16bit)**
  - 모든 명령어는 16bit로 구성
  - 정수 데이터는 16bit 길이의 2의 보수방식 사용
  - 기본적으로 3-주소 명령어 형식을 사용
  - 적재-저장 명령어 구조
  - 8개의 범용 레지스터



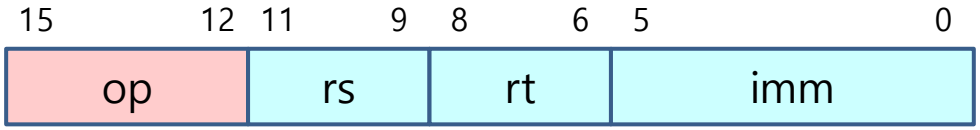
picoMIPS 명령어 구조

- **R-형식 명령어:** Reg[**rd**] ← **fn**(Reg[**rs**], Reg[**rt**])
  - rs와 rt에 명시 된 2개의 레지스터를 사용하여 연산,
    연산 결과는 rd에 명시된 레지스터에 저장
  - **fn**: **and**, **or**, **add**, **sub**, **mul**, **div**



picoMIPS 명령어 구조

- **I-형식 명령어**
  - 6bit **imm**(immediate) 필드
    - 6bit 2의 보수 표현, -32~31 사이의 상수
    - 워드 단위(2byte)로 메모리에 정렬되므로 ×2 하여 사용
  - 적재 명령 **lw**(load word): Reg[**rt**] ← M[Reg[**rs**] + **imm** × 2]
  - 저장 명령 **sw**(store word): M[Reg[**rs**] + **imm** × 2] ← Reg[**rt**]
  - 조건 분기 명령 **beq**(branch equal):
    if (Reg[**rs**] == Reg[**rt**]) PC ← PC + 2 + **imm** × 2
  - 기타 명령 **addi**, **subi**: Reg[**rt**] ← **f**(Reg[**rs**], **imm**)



picoMIPS 명령어 구조

- **J-형식 명령어:** PC ← PC + 2 + **addr** × 2
  - 무조건 분기 명령 **j**(jump 명령) 에 사용
  - 12bit **addr** 필드
    - 12bit 2의 보수 표현, -2048~2047 사이의 상수
    - 워드 단위(2byte)로 메모리에 정렬되므로 ×2 하여 사용

picoMIPS 명령어 목록

명령어	형식	op	fn	의미
and $rd, $rs, $rt	R	0000	000	$rd ← $rs and $rt
or $rd, $rs, $rt	R	0000	001	$rd ← $rs or $rt
add $rd, $rs, $rt	R	0000	010	$rd ← $rs + $rt
sub $rd, $rs, $rt	R	0000	011	$rd ← $rs - $rt
mul $rd, $rs, $rt	R	0000	100	$rd ← $rs x $rt
div $rd, $rs, $rt	R	0000	101	$rd ← $rs / $rt
addi $rt, $rs, imm	I	1010		$rt ← $rs + imm
subi $rt, $rs, imm	I	1011		$rt ← $rs – imm
lw $rt, imm($rs)	I	0100		$rt ← M[$rs + imm x 2]
sw $rt, imm($rs)	I	0101		M[$rs + imm x 2] ← $rt
beq $rt, $rs, imm	I	0001		if ($rt == $rs) then PC ← PC + 2 + imm x 2
bles $rt, $rs, imm	I	0010		if ($rt < $rs) then PC ← PC + 2 + imm x 2
j addr	J	0011		PC ← PC + 2 + addr x 2
halt	J	1111		stop execution

2. 과제 개요

- **제공되는 시뮬레이터 프로젝트의 구성을 이해**
  - CodeBlocks에서 C-언어로 작성된 시뮬레이터 프로젝트 제공
  - 메모리 모듈과 프로그램 로딩을 포함한 메인 모듈이 구현되어 있음
  - 시뮬레이터 테스트 프로그램 제공

- **picoMIPS 시뮬레이터의 프로세서 부분 작성**
  - 제공된 프로젝트에 프로세서 모듈을 구현하여 프로젝트를 완성
  - 제공된 시뮬레이터 테스트 프로그램을 이용하여 프로세서의 동작을 확인
  - CodeBlocks가 아닌 다른 개발 환경을 사용해도 무방함

3. picoMIPS 프로젝트

- **프로젝트 구성 (프로세서 모듈을 제외하고 AccCom 프로젝트와 동일)**

구분	주요 구성 요소	설명
sim-com.h	메모리 인터페이스	메모리 크기 및 읽기/쓰기 함수 선언
	프로세서 인터페이스	runProcessor 함수 선언
sim-memory.c	메모리 인터페이스	메모리 배열 및 읽기/쓰기 함수
sim-main.c	loadProgram()	기계어 프로그램 로딩 함수
	main()	① 프로그램 로딩 ② 테스트 케이스 입력 ③ 프로세서 동작으로 프로그램 실행 ④ 실행 결과 확인
sim-proc-picomips.c	**char author[]**	**작성자의 학번, 이름 문자열**
	runProcessor()	**프로세서 실행 함수**

- **sim-proc-picomips.c**
  - 학생들이 작성해야 하는 picoMIPS 프로세서 모듈 (다른 코드는 변경하지 않음)
  - **author** 문자열에 학번과 영문 이름 기재
  - **runProcessor** 함수 구현 (필요한 변수 및 서브 함수 자유롭게 구현)

sim-com.h

※ AccCom 프로젝트와 동일

```
typedef unsigned char  BYTE;
typedef unsigned short WORD;

#define MEM_SIZE0x1000     // memory size
#define END_OF_ARG    0xFFFF     // end of argument

// Read a byte/word data from memory
BYTE readByte(WORD addr);
WORD readWord(WORD addr);

// Write a byte/word data to memory
void writeByte(WORD addr, BYTE data);
void writeWord(WORD addr, WORD data);

// Write variable # of words data to memory until END_OF_ARG
WORD writeWords(WORD addr, WORD data, ...);

// Input and check variable for main function
void inputData(char *vid, WORD addr, WORD data);
void checkData(char *vid, WORD addr, WORD data);

// STUDENT id and name
extern char author[];

// Processor simulation function coded by STUDENT
int runProcessor(WORD start_addr);
```

sim-memory.c

※ AccCom 프로젝트와 동일 (단, 10진수 출력 추가)

- **메모리 배열**
  - BYTE memory[MEM_SIZE];

- **메모리 인터페이스**
  - readByte(addr) // 바이트 단위 메모리 읽기
  - readWord(addr) // 워드 단위 메모리 읽기
  - writeByte(addr, data) // 메모리에 바이트 단위 데이터 쓰기
  - writeWord(addr, data) // 메모리에 워드 단위 데이터 쓰기
  - writeWords(addr, data, ...) // 메모리에 복수개의 워드 단위 데이터 쓰기
  - inputData(vid, addr, data) // 변수에 data를 입력하고 화면에 출력
  - checkData(vid, addr, data) // 변수의 값을 확인하고 판정을 화면에 출력

sim-main.c

※ AccCom 프로젝트와 동일

- **프로그램 로딩 및 메인 함수**
  - loadProgram()
    - 기계어 프로그램을 메모리에 로딩하는 함수
      - 일반적으로 데이터는 0x0100번지, 코드는 0x0200번지에 로딩
      - 프로그램의 시작 주소를 리턴 (시작 주소가 0x0200번지가 아닐 수 있음)
  - main()
    - 복수개의 테스트 케이스에 대해 기계어 프로그램의 실행을 제어
    - 각 테스트 케이스의 입력 데이터에 대해
      ① loadProgram 함수를 호출하여 프로그램 로딩
      ② inputData 함수를 이용하여 테스트 케이스 입력
      ③ runProcessor 함수를 호출하여 프로그램 실행
      ④ checkData 함수를 이용하여 실행 결과 확인

sim-proc-picomips.c

```c
#include <stdio.h>
#include <stdlib.h>
#include "sim-com.h"

char author[] = "201912345 Gildong Hong";   // student id and name

//=======================================
// picoMIPS processor simulation function
// - start_addr: start address of program
// - return exit state = 0: normal exit
//                       1: error exit
//=======================================
int runProcessor(WORD start_addr) {
    WORD pc = start_addr;       // set PC to start address of program

    while (1) {
        // fetch cycle
        WORD ir = readWord(pc);
        pc += (unsigned int)2;  // set PC to next instruction addr

        // execution cycle
        if (ir == 0xF000)       // halt
            return 0;
    }

    return 1;
}
```

picoMIPS 프로그램 예제

- a~b의 합 s를 구하는 프로그램

```
int a, b, s;

// a = input();
// b = input();

r1 = a;          // counter
r2 = b + 1;
r3 = 0;          // sum

loop_begin:
if (r1 == r2) goto loop_end
r3 = r3 + r1;
r1 = r1 + 1;
goto loop_begin
loop_end:
s = r3;

// check(s);
```

<알고리즘>

```
int a, b, s;

// a = input();
// b = input();

sub   r0, r0, r0       // r0 = 0
addi  r0, r0, #0x10    // r0 = 0x10
mul   r0, r0, r0       // r0 = 0x0100, addr of a
lw    r1, #0(r0)       // r1 = a
lw    r2, #1(r0)       // r2 = b
addi  r2, r2, #1       // r2 = b + 1
sub   r3, r3, r3       // r3 = 0
loop_begin:
beq   r1, r2, loop_end(+3)
add   r3, r3, r1       // r3 = r3 + r1
addi  r1, r1, #1       // r1 = r1 + 1
j     loop_begin(-4)
loop_end:
sw    r3, #2(r0)       // s = r3
halt

// check(s);
```

<picoMIPS 어셈블리 코딩>

picoMIPS 프로그램 예제

```
unsigned int loadProgram() {

    // DATA section
    writeWords(0x0100,  0x0000, // 0100: a
                        0x0000, // 0102: b
                        0x0000, // 0104: s
                        END_OF_ARG);

    // CODE section
    writeWords(0x0200,  0x0003, // 0200: sub    r0, r0, r0
                        0xA010, //       addi   r0, r0, #0x10
                        0x0004, //       mul    r0, r0, r0
                        0x4040, //       lw     r1, #0(r0)
                        0x4081, //       lw     r2, #1(r0)
                        0xA481, //       addi   r2, r2, #1
                        0x06DB, //       sub    r3, r3, r3
                                //       loop_begin:
                        0x1283, //       beq    r1, r2, loop_end(+3)
                        0x065A, // 0210: add    r3, r3, r1
                        0xA241, //       addi   r1, r1, #1
                        0x3FFC, //       j      loop_begin(-4)
                                //       loop_end:
                        0x50C2, //       sw     r3, #2(r0)
                        0xF000, //       halt
                        END_OF_ARG);

    return 0x0200;  // return start address of program
}
```

picoMIPS 프로그램 예제

```
int main() {
    int exit_code;                      // 0: normal exit, 1: error exit
    WORD start_addr;                    // start address of program
    char program[] = "Sum of a to b";   // program title

    printf("================================\n");
    printf(" picoMIPS Computer Simulator\n");
    printf(" Author: %s\n", author);
    printf(" Program: %s\n", program);
    printf("================================\n");

    short a[]   = { -2,  0, -100,    1 };
    short b[]   = {  3, 10,   99,  100 };
    short sum[] = {  3, 55, -100, 5050 };
    int tc_size = 4;

    for (int t = 0; t < num_test; t++) {
        printf("\n*** Test Case %d ***\n", t + 1);

        // load program into memory
        start_addr = loadProgram();

        // input variable value
        inputData("a", 0x0100, a[t]);
        inputData("b", 0x0102, b[t]);

        // run processor module
        exit_code = runProcessor(start_addr);
        printf("  exit code = %d\n", exit_code);

        // check result value
        checkData("s", 0x0104, sum[t]);
    }
}
```

```
================================
 picoMIPS Computer Simulator
 Author: 198812345 Seokhoon Ko
 Program: Sum of a to b
================================

*** Test Case 1 ***
  a = 0100: FFFE(-2)
  b = 0102: 0003(3)
  exit code = 0
  s = 0104: 0003(3) // correct

*** Test Case 2 ***
  a = 0100: 0000(0)
  b = 0102: 000A(10)
  exit code = 0
  s = 0104: 0037(55) // correct

*** Test Case 3 ***
  a = 0100: FF9C(-100)
  b = 0102: 0063(99)
  exit code = 0
  s = 0104: FF9C(-100) // correct

*** Test Case 4 ***
  a = 0100: 0001(1)
  b = 0102: 0064(100)
  exit code = 0
  s = 0104: 13BA(5050) // correct
```

4. 시뮬레이터 테스트 프로그램

- **picoMIPS 시뮬레이터의 동작을 확인하기 위한 테스트 프로그램 제공**
  - inputData 함수로 테스트 케이스를 입력하고 프로그램을 실행한 후에 checkData 함수로 실행 결과를 자동으로 확인
    - 결과 값이 맞으면 **correct**, 틀리면 **incorrect!** 출력
  - sim-main-pmtest*n*.c 소스코드를 프로젝트의 sim-main.c에 복사하여 사용

- **시뮬레이터 테스트 프로그램 #1**
  - sim-main-pmtest1.c
  - lw, sw, and, or, add, sub, mul, div, addi, subi 명령어를 테스트

- **시뮬레이터 테스트 프로그램 #2**
  - sim-main-pmtest2.c
  - beq, bles, j 명령어를 테스트

시뮬레이터 테스트 프로그램 #1

```
int a, b, c, d, e, f;
int g, h, i, j, k, l;

// a = input();
// b = input();

r1 = a;
r2 = b;
c = r1;

r3 = r1 & r2;
d = r3;
r4 = r1 | r2;
e = r4;

r5 = r1 + r2;
f = r5;
r6 = r1 - r2;
g = r6;
```

```
r7 = r1 * r2;
h = r7;
r3 = r1 / r2;
i = r3;

r4 = r1 + (-2);
j = r4;
r5 = r1 – 3;
k = r5;
r6 = r1 – (-4);
l = r6;

// check(c, a);
// check(d, a and b);
// check(e, a or b);
// check(f, a + b);
// check(g, a - b);
// check(h, a * b);
// check(i, a / b);
// check(j, a + (-2));
// check(k, a - 3);
// check(l, a – (-4));
```

<테스트 프로그램 #1 알고리즘>

▶

```
int a, b, c, d, e, f;
int g, h, i, j, k, l;

// a = input();
// b = input();

sub  r0, r0, r0
addi r0, r0, 0x10
mul  r0, r0, r0

lw   r1, #0(r0)  // a
lw   r2, #1(r0)  // b
sw   r1, #2(r0)  // c

and  r3, r1, r2
sw   r3, #3(r0)  // d
or   r4, r1, r2
sw   r4, #4(r0)  // e

add  r5, r1, r2
sw   r5, #5(r0)  // f
sub  r6, r1, r2
sw   r6, #6(r0)  // g
```

```
mul  r7, r1, r2
sw   r7, #7(r0)   // h
div  r3, r1, r2
sw   r3, #8(r0)   // i

addi r4, r1, #-2
sw   r4, #9(r0)   // j
subi r5, r1, #3
sw   r5, #10(r0)  // k
subi r6, r1, #-4
sw   r6, #11(r0)  // l
halt

// check(c, a);
// check(d, a and b);
// check(e, a or b);
// check(f, a + b);
// check(g, a - b);
// check(h, a * b);
// check(i, a / b);
// check(j, a + (-2));
// check(k, a - 3);
// check(l, a – (-4));
```

<picoMIPS 어셈블리 코딩>

시뮬레이터 테스트 프로그램 #1

```
================================
 picoMIPS Computer Simulator
 Author: 198812345 Seokhoon Ko
 Program: picoMIPS Test #1
================================

*** Test Case 1 ***
  a = 0100: 0006(6)
  b = 0102: 0002(2)
  exit code = 0
  c =   a    = 0104: 0006(6) // correct
  d = a & b  = 0106: 0002(2) // correct
  e = a | b  = 0108: 0006(6) // correct
  f = a + b  = 010A: 0008(8) // correct
  g = a - b  = 010C: 0004(4) // correct
  h = a * b  = 010E: 000C(12) // correct
  i = a / b  = 0110: 0003(3) // correct
  j = a+(-2) = 0112: 0004(4) // correct
  k = a - 3  = 0114: 0003(3) // correct
  l = a-(-4) = 0116: 000A(10) // correct
```

```
*** Test Case 2 ***
  a = 0100: 0009(9)
  b = 0102: FFFD(-3)
  exit code = 0
  c =   a    = 0104: 0009(9) // correct
  d = a & b  = 0106: 0009(9) // correct
  e = a | b  = 0108: FFFD(-3) // correct
  f = a + b  = 010A: 0006(6) // correct
  g = a - b  = 010C: 000C(12) // correct
  h = a * b  = 010E: FFE5(-27) // correct
  i = a / b  = 0110: FFFD(-3) // correct
  j = a+(-2) = 0112: 0007(7) // correct
  k = a - 3  = 0114: 0006(6) // correct
  l = a-(-4) = 0116: 000D(13) // correct

*** Test Case 3 ***
  a = 0100: FFF7(-9)
  b = 0102: 0005(5)
  exit code = 0
  c =   a    = 0104: FFF7(-9) // correct
  d = a & b  = 0106: 0005(5) // correct
  e = a | b  = 0108: FFF7(-9) // correct
  f = a + b  = 010A: FFFC(-4) // correct
  g = a - b  = 010C: FFF2(-14) // correct
  h = a * b  = 010E: FFD3(-45) // correct
  i = a / b  = 0110: FFFF(-1) // correct
  j = a+(-2) = 0112: FFF5(-11) // correct
  k = a - 3  = 0114: FFF4(-12) // correct
  l = a-(-4) = 0116: FFFB(-5) // correct
```

<테스트 프로그램 #1 실행 결과>

시뮬레이터 테스트 프로그램 #2

```
int a, b;
int eq, max;
int TRUE = 0x0F;
int FALSE = 0x01;

// a = input();
// b = input();

r1_max:     // r2 < r1
eq = r4;    // FALSE
max = r1;   // a
jump final

main:
r1 = a;
r2 = b;
r3 = TRUE;
r4 = FALSE;
```

```
if (r1==r2) goto equal
if (r1<r2) goto r2_max
if (r2<r1) goto r1_max

final:
exit(0);

equal:      // r1 == r2
eq = r3;    // TRUE
max = r1;   // a
jump final

r2_max:     // r1 < r2
eq = r4;    // FALSE
max = r2;   // b
jump final

// check(eq, a == b);
// check(max(a,b));
```

<테스트 프로그램 #2 알고리즘>

```
int a, b;
int eq, max;
int TRUE = 0x0F;
int FALSE = 0x01;

// a = input();
// b = input();

r1_max: // r2 < r1
sw   r4, #2(r0) // eq
sw   r1, #3(r0) // max
j    final

main:
sub  r0, r0, r0
addi r0, r0, 0x10
mul  r0, r0, r0

lw   r1, #0(r0) // a
lw   r2, #1(r0) // b
lw   r3, #4(r0) // TRUE
lw   r4, #5(r0) // FALSE
```

```
beq  r1, r2, equal
bles r1, r2, r2_max
bles r2, r1, r1_max

final:
halt

equal: // r1 == r2
sw   r3, #2(r0) // eq
sw   r1, #3(r0) // max
j    final

r2_max: // r1 < r2
sw   r4, #2(r0) // eq
sw   r2, #3(r0) // max
j    final

// check(eq, a == b);
// check(max(a,b));
```

<picoMIPS 어셈블리 코딩>

시뮬레이터 테스트 프로그램 #2

```
================================
 picoMIPS Computer Simulator
 Author: 198812345 Seokhoon Ko
 Program: picoMIPS Test #2
================================

*** Test Case 1 ***
  a = 0100: 0003(3)
  b = 0102: 0003(3)
  exit code = 0
  eq = (a == b) = 0104: 000F(15) // correct
  max(a, b)     = 0106: 0003(3) // correct

*** Test Case 2 ***
  a = 0100: 0004(4)
  b = 0102: 0006(6)
  exit code = 0
  eq = (a == b) = 0104: 0001(1) // correct
  max(a, b)     = 0106: 0006(6) // correct
```

```
*** Test Case 3 ***
  a = 0100: FFFB(-5)
  b = 0102: FFFB(-5)
  exit code = 0
  eq = (a == b) = 0104: 000F(15) // correct
  max(a, b)     = 0106: FFFB(-5) // correct

*** Test Case 4 ***
  a = 0100: FFFE(-2)
  b = 0102: FFF9(-7)
  exit code = 0
  eq = (a == b) = 0104: 0001(1) // correct
  max(a, b)     = 0106: FFFE(-2) // correct
```

<테스트 프로그램 #2 실행 결과>

보고서 작성 요령

- **이 과제의 주요 문제 (= 설명해야 하는 문제 해결 아이디어)**
  - 주요 레지스터 구현 방법?
  - 명령어에서 연산 부호와 피연산자 구분 방법?
  - 16bit 2의 보수 방식 데이터의 산술 연산 방법?
  - imm과 addr의 부호 확장 처리 방법?

- **테스트 프로그램의 실행 결과를 첨부**
  - 실험/실습 보고서에는 항상 실험/실습 결과를 첨부해야 한다.
  - 두 개의 테스트 프로그램 실행 결과를 첨부한다.