MIPS Instruction Simulation 프로젝트

프로젝트 개요

MIPS 명령어 실행을 C/C++ 로 시뮬레이션하는 프로젝트

전체는 2단계로 진행됩니다.

1. MIPS Assembler (Assembly to Binary)

기능 요구사항

- MIPS Assembly 코드를 분석하여 바이너리 코드로 변환
- 명령어 아래 Table 참고
- 출력 파일 형식: 바이너리 파일(.bin)

세부 설계

1. **입력**
    - .asm 파일을 읽어들여 명령어를 파싱
    - 공백 및 주석 제거

2. **어셈블러 처리 과정**
    - **1단계: 토큰화(Tokenize)**
        1. 명령어(예: add $t0, $t1, $t2)를 토큰 단위로 분해
    - **2단계: 명령어 변환**
        1. MIPS 명령어를 분석하고 바이너리 변환
    - **3단계: 레이블(Symbol Table) 처리**

1. jump, branch 명령어의 주소 계산
2. Program의 주소는 0으로 가정

- **4단계: 바이너리 출력**
  1. 변환된 명령어를 .bin 파일로 저장

Instruction 포멧

- R-format instruction

Instruction	Opcode (6비트)	Funct (6비트)
`add`	`000000`	`100000` (0x20)
`sub`	`000000`	`100010` (0x22)
`and`	`000000`	`100100` (0x24)
`or`	`000000`	`100101` (0x25)
`xor`	`000000`	`100110` (0x26)
`nor`	`000000`	`100111` (0x27)
`slt`	`000000`	`101010` (0x2A)
`sll`	`000000`	`000000` (0x00)
`srl`	`000000`	`000010` (0x02)
`jr`	`000000`	`001000` (0x08)

- I format instruction

Instruction	Opcode (6비트)
`addi`	`001000`
`andi`	`001100`
`ori`	`001101`
`xori`	`001110`
`lw`	`100011`
`sw`	`101011`
`lui`	`001111`
`slti`	`001010`
`beq`	`000100`
`bne`	`000101`

- J format instruction

Instruction	Opcode (6비트)
`j`	`000010`
`jal`	`000011`

Register Address map

강의자료 19, 40page 참고

추가

0: $zero

1, 26, 27 번은 구현하지 않음.

예제

입력(input.asm)

assembly

복사편집

```
addi $t0, $zero, 5  # t0 = 5
addi $t1, $zero, 10 # t1 = 10
add  $t2, $t0, $t1  # t2 = t0 + t1
```

출력(output.bin)

복사편집

00100000000010000000000000000101

00100000000010010000000000001010

00000001000010010101000000100000

2. MIPS CPU Simulator (Binary Execution)

기능 요구사항

- 바이너리 명령어를 읽어 실행
- CPU 내부 비트 단위 시뮬레이션
- 레지스터 파일(32개 레지스터) 구현
- 메모리(RAM) 관리
- 명령어 실행 흐름 제어 (PC, 분기 처리)
- Step-by-step 실행 모드 지원 및 Step별 레지스터 파일 출력

세부 설계

1. **CPU 내부 구성**
    - **32-bit 레지스터(32개): 구조체로 구현**
    - 메모리 (1MB RAM)
    - **프로그램 카운터(PC)**
    - **ALU 연산 수행 (C/C++함수로 구현)**
    - **명령어 디코더(Instruction Decoder)**

2. **입력 및 실행 과정**
    - .bin 파일을 읽어 32-bit 명령어로 변환
    - 명령어 타입에 따라 해석
    - 해당 명령을 실행 (연산 수행 및 레지스터 값 변경)
    - 메모리 접근(load/store) 및 분기 수행

CPU 내부 상태

- 32개 레지스터($zero, $at, $v0, ..., $t0, ..., $s0, ..., $ra)
- RAM(1MB, 0x10000000부터 시작)
- 프로그램 카운터(PC)
- 명령어 레지스터(IR)
- ALU(덧셈, 뺄셈, AND, OR 등)

명령어 실행 흐름

1. fetch (PC에서 명령어 가져오기)
2. decode (명령어 해석)
3. execute (ALU 연산, 레지스터 변경)
4. memory (load/store 수행)
5. write back (결과 저장)
6. update PC (다음 명령어 실행)

3. 테스트

Sample.asm을 실행하고 결과를 확인 (코드는 바뀔 수 있음)

```
addi $t0, $zero, 1    # $t0 = 1 (시작 숫자)
    addi $t1, $zero, 0    # $t1 = 0 (합계 초기화)
    addi $t2, $zero, 100 # $t2 = 100 (최대 숫자)

loop:
    add  $t1, $t1, $t0    # $t1 = $t1 + $t0 (현재 숫자를 합산)
    addi $t0, $t0, 1      # $t0 = $t0 + 1 (다음 숫자로 증가)
    ble  $t0, $t2, loop  # $t0 <= $t2 이면 loop로 이동 (1부터 100까지)

    sw    $t1, 0($zero)   # 메모리 주소 0x00000000에 총합 저장 (결과 확인용)

end                       # MIPS에 없는 명령어로 프로그램 종료를 의미
```

4. 처리해야 할 instruction set

(1) 산술연산: add, addi, sub, mul

(2) jw, sw

(3) beq, bne, j, jr

(4) and, or, xor, slt