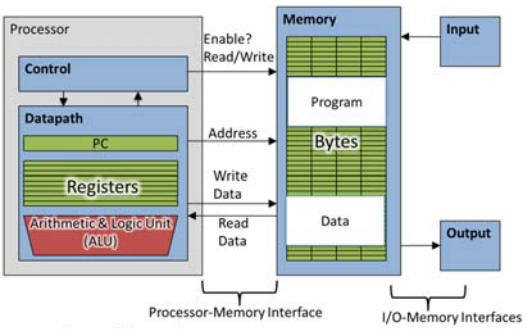
Term Project: MIPS Simulator

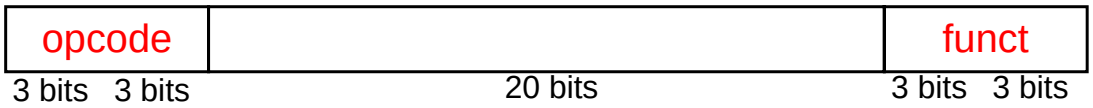
- Introduction
- Overall operations
- Internal
- Example
- Schedule

Introduction

- Simulator
  - Simple MIPS Instruction Execution
  - Machine Language Level
  - Simple Interface
  - Implement Using C language
- Purpose
  - Understand the internal operations of Computer System
    - CPU operations and Memory interface
  - Instruction cycle and Instruction coding format
  - CPU register usage and ALU operation
- C programming for hardware modeling
  - Console Input/Output interface
    - Menu processing
  - Binary file access and memory modeling
    - Executable image and memory model
  - Simulation method
    - Function usage and flow control



Instruction To be

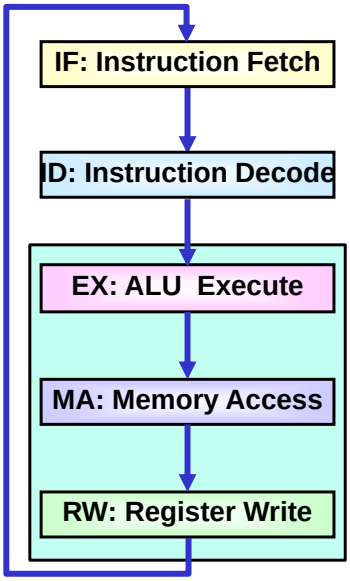


28-26 31-29	0(000)	1(001)	2(010)	3(011)	4(100)	5(101)	6(110)	7(111)
0(000)	R-format	bltz	j	jal	beq	bne		
1(001)	addi		slti		andi	ori	xori	lui
2(010)								
3(011)								
4(100)	lb			lw	lbu			
5(101)	sb			sw				

2-0 5-3	0(000)	1(001)	2(010)	3(011)	4(100)	5(101)	6(110)	7(111)
0(000)	sll		srl	sra				
1(001)	jr				syscall			
2(010)	mfhi		mflo					
3(011)	mul							
4(100)	add		sub		and	or	xor	nor
5(101)			slt					

Instruction Execution

- CPU processes instructions sequentially
  - Branch instruction can change instruction flow
- Instruction execution cycle
  - Instruction fetch
    - PC: fetch instruction from instruction memory
    - PC ← PC + 4
  - Instruction decode
    - Opcode → instruction type and field lengths
    - Register numbers → register file, read registers
  - ALU
    - Arithmetic result
    - Memory address for load/store
    - Branch target address
  - Memory access
    - Access data memory for load/store
  - Register write: result store



Overall operation

- 주어진 실행 파일의 프로그램을 실행하고 간단하게 디버깅하는 기능 구현
  - **MIPS** 명령어들로 구성된 실행 파일을 시뮬레이터 메모리에 로드
    - 명령어는 프로그램 메모리로, 데이터는 데이터 메모리로 로드
  - **CPU**의 프로그램 카운터**(PC)**를 시뮬레이션 시작 명령어 위치로 설정
    - **SP** 와 기타 레지스터도 초기화 시킴
  - 적재된 프로그램 명령어를 **Step** 명령어로 하나씩 처리하거나
  - 또는 **Go** 명령어로 끝까지 처리하도록 지시
    - 프로그램은 **syscall 10**으로 종료한다고 가정
  - 메모리의 내용과 **CPU** 레지스터의 내용을 읽을 수 있도록 함

- **Simple interface**
  - **Console** 환경으로 동작
  - 시뮬레이터 명령
    - **l, j, g, s, m, r, x, sr, sm**
    - 기타 유용한 명령어를 추가해도 됨**, ex) break**
  - 수행 결과를 확인할 수 있도록 변경 사항을 보여줌

Simulator Commands

- **l: Load program**
  - **l <**실행 파일이름**>:** 실행 파일이 시뮬레이터의 메모리에 올라간다**.**
    - 실행 파일은 바이너리 파일로 구성
    - 프로그램은 프로그램 메모리 **0x400000**번지부터 로드된다고 가정
    - 데이터는 데이터 메모리 **0x10000000**번지에 로드
    - 따라서 **PC**의 초기값은 **0x400000** 로 설정
    - **SP**의 초기값은 **0x80000000** 로 설정
  - **Executable file format**
    - **4B:** 명령어 개수**, 4B:** 데이터 개수**(**워드 기준**)**
    - 명령어 들 **…**
    - 데이터 들 **…**

- **j: Jump program**
  - **j <**프로그램 시작 위치**>:** 입력한 위치에 시뮬레이터 실행 준비
    - **PC**를 특정한 주소 값으로 설정하여 그 주소부터 실행할 수 있게 한다**.**
    - 디버깅 용

Simulator Commands

- **g: Go program**
  - 현재 **PC** 위치에서 시뮬레이터가 명령어를 끝까지 처리
    - 프로그램의 끝은 **syscall 10** 명령어라 가정
    - **syscall 10** 명령어를 만나면 사용자 명령을 받는 상태로 중지
      - **Break** 가 구현되었다면 중간 **break point**에서 중지
- **s: Step**
  - 명령어 하나를 처리하고 사용자 명령을 받는 상태로 중지
  - 가능하면 명령어에 의하여 변경된 레지스터, 메모리 정보 출력
- **m: View memory**
  - **m <start> <end>: start~end** 범위의 메모리 내용 출력
- **r: View register**
  - **r:** 현재 레지스터 내용 출력
- **x: Program exit**
  - 시뮬레이터 프로그램의 종료
- **sr <register number> <value>:** 특정 레지스터의 값 설정
- **sm <location> <value>:** 메모리 특정 주소의 값 설정

Simulator Internals

- 저장소**:** 변수로 구현하고 함수로 인터페이스 구성
  - **Memory: 3** 개의 메모리를 구현한 **MEM** 함수 사용
  - **Register: 32** 개의 레지스터**, PC, HI/LO** 레지스터
    - **32**개의 레지스터 접근은 **MEM** 함수와 유사하게 구현
    - **PC, HI/LO** 레지스터는 독립적으로 구현
  - 기타 변수들**:** 시뮬레이터 구현을 위한 각종 변수들
    - **IR: instruction register**
- 필요 함수들
  - 시뮬레이터 명령어 처리 함수
    - 기존 명령어 해석 함수 사용
  - 메모리 접근 함수
    - **unsigned int MEM(unsigned int A, int V, int nRW, int S)**
  - 레지스터 접근 함수
    - **unsigned int REG(unsigned int A, unsigned int V, unsinged int nRW);**
  - **ALU** 함수 및 관련 함수
    - **int ALU(int X, int Y, int C, int *Z);**
  - **PC** 갱신 함수

Program Example

- Main and command function example

```
unsinged int PC, IR;
void main() {
  while(1) {
    Get command line;
    switch (command) {
      case ‘l’ :
        load program; break;
      case ‘j’ :
      case ‘g’ :
      case ‘s’ :
      case ‘m’ :
      case ‘r’ :
      case ‘x’ :
        exit program;
    }
  }
}
```

```
void showRegister(void) {
  int i;
  cout << “[REGISTER]” << endl;
  for(i=0; i<REG_SIZE, i++) {
    cout << “R” << i << “=“;
    cout << R[i] << endl;
  }
  cout << “PC” << PC << endl;
}

void setPC(unsigned int val) {
  PC = val;
  return;
}
```


Program Example

- One step example

```
void step(void) {
  // instruction fetch
  IR = MEM(PC, 0, 0, 2); PC += 4;
  // instruction decode
  op = getOp(IR);
  if (op == 0) {
    fn = getFn(IR);rs = getRs(IR);
    rt = getRt(IR);rd = getRd(IR);
    if (fn == 32){    // ADD
      R[rd]=ALU(ADD,R[rs],R[rt]);
    } else if (fn == ??) {
      …
    } else {
      cout << “Undefined Inst…”;
      goto STOP;
    }
  } else if (op == 1) {
    offset = getOffset(IR);
  }
  …
}
```

```
int ALU(int fct, int v1, int v2) {
  if (fct == ADD) {
    return v1 + v2;
  } else if (fct == SUB) {
    return v1 – v2;
  } else if (fct == ?){
  …
}
```

Program Example

- **Binary file example**

```
#include "stdio.h"

int main()
{
  FILE *pFile = NULL;
  errno_t err;
  int count;
  unsigned int data;
  unsigned int data1 = 0xAABBCCDD;
  unsigned int data2 = 0x11223344;

  err = fopen_s(&pFile, "test.bin", "wb");
  if (err) {
    printf("Cannot open file\n");
    return 1;
  }
  fwrite(&data1, sizeof(data1), 1, pFile);
  fwrite(&data2, sizeof(data2), 1, pFile);
  fclose(pFile);
```

```
  err = fopen_s(&pFile, "test.bin", "rb");
  if (err) {
    printf("Cannot open file\n");
    return 1;
  }

  while (1) {
    count = fread(&data, sizeof(data1), 1,
pFile);
    if (count  != 1)
      break;
    printf("%8x\n", data);
  }
  fclose(pFile);

  return 0;
}
```


To-Do List

- **User Interface**
  - **Command input and Result show**
  - **Error handling**
- **Instruction Execution**
  - **Instruction decoding**
  - **ALU-based instruction**
  - **Jump and branch instruction**
- **Simulator Environment**
  - **Test program**
    - 개별 명령어 구현 검사를 위한 프로그램 작성
    - 최종적으로 주어진 바이너리 파일 실행
  - **Image loading**
- **Documentation**
  - **PPT and presentation**

- **2~3 Students**

Schedule

- 7주
  - Introduction and Team Organization
  - MIPS instruction review
- 9주
  - C Input/output function (Console)
  - File management and memory loading
- 10주
  - Overall Design: function level
  - Test Program (Machine instruction)
- 11주
  - Program Test and Debugging
  - Middle Reporting: Current Progress
- 12주
  - (Option) GUI 구현
- 13주부터 구현한 팀 순서대로
  - Final Report and Presentation