컴퓨터구조및모바일프로세서 과제1 보고서

과목: 컴퓨터구조및모바일프로세서
교수명: 유시환 교수님
학과: 모바일시스템공학과
학번: 32224020
이름: 전은지
**Rest Freedays: 4

Index

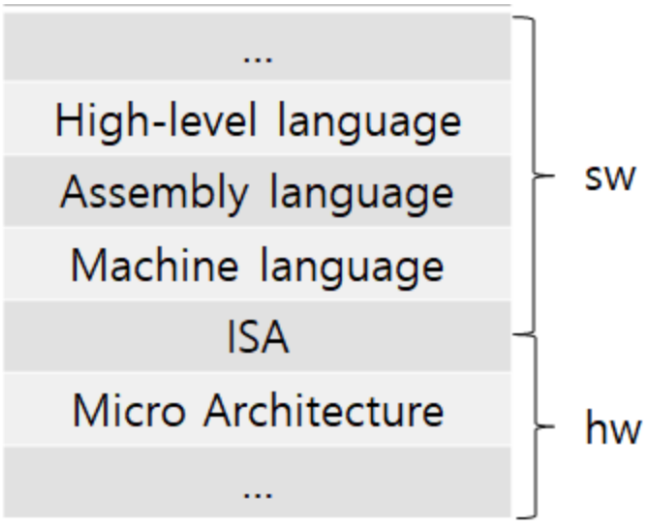
I. 서론

컴퓨터는 주어진 명령에 따라 '연산'을 수행한다. 연산은 산술 연산, 논리 연산, 정보 기억/저장 등이 될 수 있다. 컴퓨터의 연산을 이해하기 위해 먼저 간단한 계산기를 만들 것이다. 간단한 계산기는 더하기, 빼기, 곱하기, 나누기와 같은 이진 연산(피연산자 두 개)을 할 수 있다. 계산기를 실행하려면 '입력 명령어'가 필요하고 'input.txt'파일을 입력 파일로 사용할 것이다. 계산기는 파일에 입력된 명령어를 읽고 하나씩 실행한다.

II. 본론

1. Background information

1) ISA(Instruction set architecture)



ISA란 마이크로프로세서가 인식해서 기능을 이해하고 실행할 수 있는 기계어 명령어로 소프트웨어와 하드웨어가 서로 통신할 수 있게 해주는 명령어 집합 구조이다. 이는 최하위 레벨의 프로그래밍 인터페이스로, 프로세서가 실행할 수 있는 모든 명령어들을 포함한다. 명령어 집합 구조는 자료형, 명령어, 레지스터, 어드레싱 모드, 메모리 구조, 인터럽트, 예외 처리, 외부 입출력을 포함한 프로그래밍 관련 컴퓨터 아키텍처의 일부이다.

ISA에 포함되는 명령어들에는 ALU를 이용해 사칙연산/논리연산 등을 수행하는 산술 논리 명령어, 메모리 간 데이터를 전송하는 데이터 전송 명령어, 조건에 따라 서로 다른 실행 흐름으로 분기하는 실행 흐름 제어 명령어, 부동소수점 실수의 연산을 수행하는 부동소수점 연산 명령어가 있다. 각 명령어는 보통 opcode와 operand를 가지며 일반적으로 피연산자를 나타내는 하나 이상의 지정자를 가진다. Opcode는 수행할 명령어를 나타내는 부호를 말하고 Operand는 opcode의 연산 대상이다. Operand는 프로세서 레지스터, 메모리 주소, 리터럴 상수, 레이블이 될 수 있다.

2) CISC(Complex Instruction Set Computer)

CISC는 복잡한 명령어 집합을 갖는 CPU 아키텍처로 한국어로는 복잡 명령어 집합 컴퓨터라고 한다. 명령어가 복잡하기 때문에 명령어를 해석하는 데 시간이 오래 걸리며, 명령어 해석에 필요한 회로도 복잡하여 효율성이 떨어지는 단점이 있다. 하지만 보통 풍부한 어드레싱 기능을 갖추고 있어 명령의 직교성이 좋으며, 어느 어드레싱 모드에서도 임의의 연산을 수행할 수 있다. 연산에 대해서는 레지스터와 레지스터 연산, 레지스터와 메모리 연산, 메모리와 메모리 연산을 모두 갖추고 있는 것이 보통이다. 피연산자는 2개에서 3개까지 지정할 수 있는 경우가 많다. x86 아키텍처는 CISC 설계를 채택하였다.

3) RISC(Reduced Instruction Set Computer)

RISC는 CPU 명령어의 개수를 줄여 명령어 해석시간을 줄임으로써 개별 명령어의 실행속도를 빠르게 한 컴퓨터로 한국어로는 축소 명령어 집합 컴퓨터라고 한다. 전통적인 CISC CPU에는 프로그래밍을 돕기 위한 많은 수의 명령어와 주소 모드가 존재했다. 그러나 그중에서 실제로 쓰이는 명령어는 몇 개 되지 않는다는 사실을 바탕으로, 적은 수의 명령어만으로 명령어 집합을 구성한 것이 RISC이다. 그래서, RISC는 CISC보다 구조가 더 단순하다. 따라서 RISC는 명령어 파이프라인을 통해 개별 명령어의 실행속도를 높여 복잡한 연산도 적은 수의 명령어들을 조합하는 방식으로 수행을 가능하게 하였다. RISC는 모든 연산이 하나의 클럭으로 실행되기 때문에 파이프라인을 기다리게 하지 않는다. 또한, 레지스터 사이의 연산만 실행하며, 메모리 접근은 로드, 스토어 명령어로 제한되어 회로가 단순해지고 불필요한 메모리 접근을 줄일 수 있다. 많은 수의 레지스터를 사용하여 메모리 접근을 줄이고, 지연 실행 기법을 사용하여 파이프라인의 위험을 줄인다는 특징이 있다. RISC 설계를 채택하 프로세서에는 MIPS 계열, IBM 파워 계열, ARM 계열 등이 있다.

2. Program description

제시된 C 프로그램은 ISA를 구현하는 코드이다. 주어진 코드는 기본적인 산술 연산, 데이터 이동, 비교, 그리고 프로그램의 종료를 포함하는 명령어 집합을 제공한다. 다음은 코드의 구성 요소와 작동 원리에 대한 설명이다.

[레지스터와 명령 레지스터]

프로그램은 총 10개의 레지스터('reg[10]')를 사용한다. 실제 하드웨어나 고급 시뮬레이션 환경이라면 더 많은 레지스터가 필요하겠지만 학습 목적의 프로그램이기 때문에 레지스터의 개수를 제한하였다. 명령 레지스터('inst_reg')의 경우 현재 실행중인 명령어를 저장한다. 이는 명령어를 해석하고 실행하기 전에 필요한 모든 정보를 포함한다.

[연산자 식별 및 연산]

프로그램은 'input.txt'로부터 명령어를 받아 '<연산자> <피연산자1> <피연산자2>'의 형태로 명령어를 처리한다. 연산자는 수행할 연산을 지정하고, 피연산자는 if문을 거쳐 연산에 사용될 입력값과 레지스터로 구분한다.

프로그램에는 총 7가지의 연산이 존재한다. 만약 연산자가 '+', '-', '*', '/' 중 하나라면 산술 연산이 시행된다. 산술 연산은 두 피연산자의 덧셈, 뺄셈, 곱셈, 나눗셈을 수행하고 결과를 'R0'에 저장한다. 연산자가 'M'이라면 데이터가 이동된다. 레지스터 간 이동 또는 특정 값을 레지스터로 로드하는 데 사용된다. 연산자가 'C'라면 비교 연산이 실행된다. 제시된 두 개의 피연산자의 값을 비교하여 동일하다면 0을, 첫 번째 피연산자가 더 크면 +1을, 그렇지 않으면 -1을 'R0'에 저장한다. 마지막으로, 연산자가 'H'라면 그 즉시 프로그램의 실행을 종료시킨다.

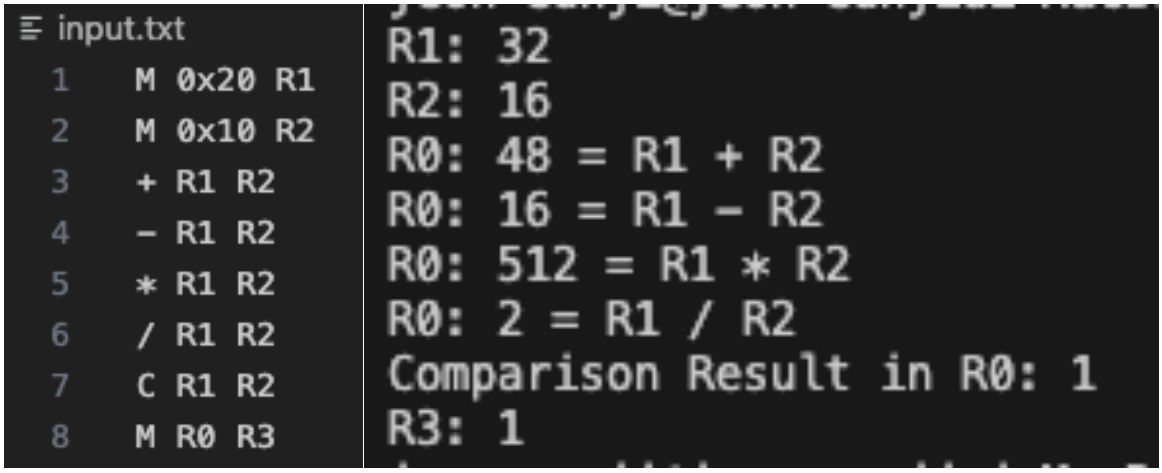
[프로그램 흐름 제어]

명령 포인터('inst_ptr')를 통해 현재 실행 중인 명령어의 위치를 추적한다. 이는 명령어가 순차적으로 실행됨을 나타내기 위해 사용되지만 이번 프로그램에서는 실질적으로 사용되지 않았다. 사용하기 위해 배열을 선언할 수 있었으나 프로그램이 복잡해지는 것을 막기 위해 사용하지 않았다.

[에러 처리]

산술 연산 중 나눗셈을 실행할 때 만약 분모가 0인 경우 에러 메시지를 출력하고 연산을 중지한다. 또한, switch문에 존재하지 않는 케이스의 연산자가 입력되었을 경우 지원되지 않는 연산임을 출력하고 연산이 실행되지 않는다. 마지막으로 파일이 열리지 않는다면 에러 메시지를 출력하고 프로그램을 실행하지 않는다.

3. Result



다음과 같이 'input.txt' 파일을 생성한다. 첫 번째 피연산자에 0x20(32)를 입력하고 두 번째 피연산자에 0x10(16)을 입력한 후 5가지의 연산을 실행하였다. 실행 결과 모든 연산이 막힘없이 진행되었고 R0의 입력값을 R3에 이동시키는 연산을 끝으로 프로그램을 종료시켰다. 종료 연산자인 'H'는 따로 실행하지 않았다.

III. Conclusion

이번 과제를 통해 ISA의 기본적인 개념에 대하여 알게 되었다. MIPS 아키텍처를 C언어로 구현해보면서 프로그램의 진행 원리에 대해 이해하고 명령어와 레지스터에 대한 이해도를 높일 수 있었다.

하지만 프로그램을 만들면서 여러가지 아쉬운 점이 있었다. 먼저, 명령 포인터의 이용이 제대로 되지 않았다. 현재 실행 주소를 가리키기 위해 사용되는 명령 포인터는 이번 프로그램에서 사용되지 않았다. 또한, 선택 구현에서 비교 연산 이외의 구현은 실행하지 못했다는 점이다. 점프를 구현하려고 하였으나 구현시 무한루프로 빠지는 문제점이 발생하여 코드에서 제외하였다. GCD는 코드에서 J와 BEQ가 제대로 작동되어야 구할 수 있기 때문에 구하지 못하였다.

다양한 문제점이 존재하지만 기본적인 산술 연산과 비교 연산이 제대로 실행되었고, 추후 공부를 통해 ISA에 대한 이해도가 높아진다면 구현하지 못 한 부분에 대한 코드를 작성할 수 있을 것이라 기대된다.

IV. References

Chat GPT-4

[ISA]
https://ko.wikipedia.org/wiki/%EB%AA%85%EB%A0%B9%EC%96%B4_%EC%A7%91%ED%95%A9

[[컴퓨터구조] ISA - CISC vs. RISC]
https://velog.io/@apphia39/%EC%BB%B4%ED%93%A8%ED%84%B0%EA%B5%AC%EC%A1%B0ISA-CISC-vs.-RISC

[CISC]
https://ko.wikipedia.org/wiki/%EB%B3%B5%EC%9E%A1_%EB%AA%85%EB%A0%B9%EC%96%B4_%EC%A7%91%ED%95%A9_%EC%BB%B4%ED%93%A8%ED%84%B0

[RISC]
https://ko.wikipedia.org/wiki/%EC%B6%95%EC%86%8C_%EB%AA%85%EB%A0%B9%EC%96%B4_%EC%A7%91%ED%95%A9_%EC%BB%B4%ED%93%A8%ED%84%B0