hw7 2017069598 박상지

@@instruction: language of the computer@@

#instruction set

컴퓨터가 제공하는 인터페이스로 이것을 이용해서 작업지시서를 만든다. 초창기에는 IS는 간단하게 구현하였다. 그러다가 메모리가 작아서 CISC가 유행을 했다. 메모리가 커지면서 RISC가 자리가 잡고, 후에 회사에서는 IS를 독점성을 위해 다르게 만들기 시작하였다. 하나의 프로세서가 있다고 가정할 때 사용자의 프로세서는 해당 응용프로그램에 맞게 돌아가게끔 만들어서 사용자는 그 응용프로그램만 사용하게끔 독점성을 이용한다. 하지만 이런 것들은 RISC기반으로 만들어진다.

#the MIPS instruction set

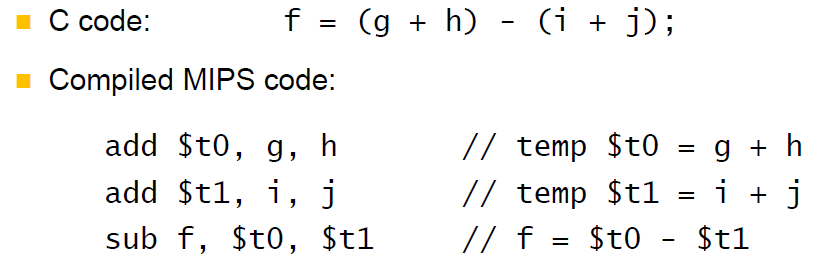
MIPS IS을 주되게 공부할 것이다. 임베디드 코어에 사용되기도 한다. 현대 ISA의 교육용으로 사용하기도 한다.

#ISA 감상: 생각의 초점

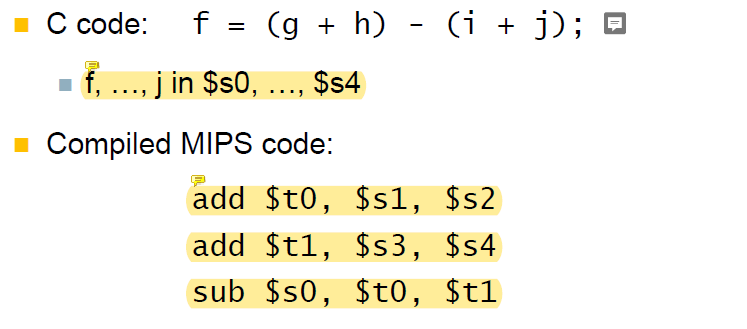
RISC ISA는 어떻게 생겼나? 왜 그렇게 생겼나?

RISC는 자주 나오는 operation을 single instruction을 만들어 준다. 자주 안 나오는 복잡한 것들은 여러 개의 instruction으로 만들어준다.

#Arithmetic Example



32개의 범용 레지스터를 갖고 있다. 32개는 상당히 많은 숫자여서 이것을 기억하기 편하게 어셈블러 언어를 사용한다. t는 임시 저장 값이고 s는 저장 값으로 사용한다. 이것은 레지스터 32개를 우리가 외우기 어렵기 때문이다.



f, g, … 는 s 레지스터에 저장했다고 가정하면 ALU instruction은 register-based mode 라고 불리는데 세 개의 operand를 사용한다. 뒤에 레지스터 두 개를 사용해서 해당 opcode에 따라 연산을 하고 앞에 있는 레지스터에 저장을 한다.

질문1)

만약에 f, g, … 해당 값들이 레지스터에 저장되어 있지 않고 메모리에 있다고 가정을 하면 해당 값들은 road를 하여서 레지스터에 각각 저장해줘야 한다.

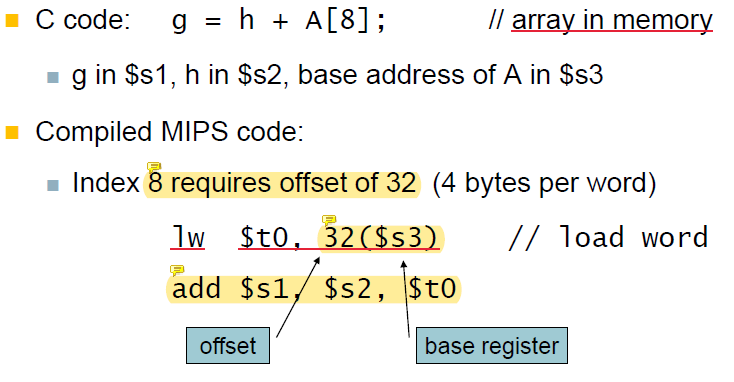
\*어느 value를 어떤 레지스터에 저장하는 것은 통상 컴파일러가 결정을 한다. 만약 어셈블리 언어라면 어셈블러 프로그래머가 직접 설정을 한다.

질문2) 범용 레지스터를 많이 만들면 좋지 않을까?

32개의 범용 레지스터를 쓰는 것은 몇 번 레지스터를 쓸려면 5bit을 사용하게 되는데, 레지스터를 몇 개를 만드는 것은 opcode, operand의 길이에 연관되어 있다. 그래서 64, 32, 16 레지스터 중에서 어느 레지스터를 사용해야 좋다. 라는 질문에서는 대답을 할 수 없다. 어떻게 설계를 하여 벤치 마킹을 빨리 돌리는게 목표이기 때문이다. 레지스터의 개수보다는 성능이 중요하다.

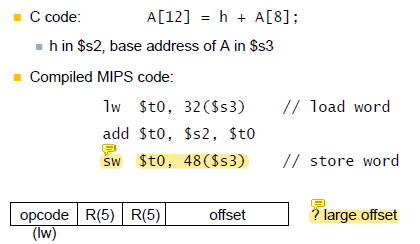
레지스터를 적게 사용하면 프로세서 입장에서는 레지스터에 있는 데이터를 접근하는 것이 빠르다. CPU가 메모리에 데이터를 접근하는 것은 수십억의 장소에서 하나를 찾는 것이다. 그래서 레지스터의 개수를 적게 써야 하나의 데이터를 찾고 읽고/쓰기 가 빨라진다. 즉 데이터 접근이 빨라진다.

#Memory Operand Example



A 배열을 메모리에서 읽어온다고 가정할 때, A 배열의 8번째의 값을 메모리에서 가져와야 한다. 메모리는 워드(word) 단위로 떨어져 있기 때문에 해당 배열은 8word, 즉 32(byte)의 상수 값을 갖게 된다. memory operand는 2개의 instruction이 있고 load 와 store이 존재한다. 레지스터 2개와 상수 값을 갖게 되고 이것을 base addressing mode 또는 offset addressing mode라고 한다. 뒤에 있는 레지스터를 읽어서 상수 값을 더해서 읽어서 그 데이터를 앞에 있는 레지스터에 저장한다.

32($s3) 은 기준이 되는 레지스터 s3에서 얼만큼 떨어져 있는지 32byte(8word)



store에서는 앞에 있는 레지스터의 데이터를 s3 레지스터에 48byte(12word)에 있는 레지스터 저장한다.

질문) load instruction은 뒤에 있는 레지스터에 왜 기준 레지스터와 상수 값을 주어지냐?

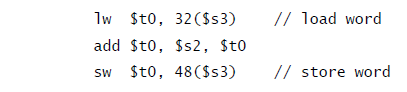
만약 32-bit address를 주게 되면 이미 instruction 자체가 32bit를 넘게 사용하기 때문에 공간을 두 개를 써야 된다. 그렇게 되면 접근을 두 번 해야 하기 때문에 성능이 떨어진다.

#Register Allocation and Spill

메모리에서 데이터를 road를 하면 레지스터에 저장된다. 어느 레지스터, 즉 몇 번 레지스터에 저장하는 것을 결정하는 게 컴파일러이다. 이런 작업을 register allocation이라 한다.

메모리에서 데이터를 레지스터 32개에 road를 하는데 만약 32개의 레지스터를 전부 다 사용하였다면 다음 메모리에서 읽은 데이터를 레지스터에 저장해야 하는데 사용하지 않은 레지스터, 즉 어느 레지스터에 저장해야 하는지 이런 작업을 register spill이라 한다. 이것도 역시 컴파일러에서 처리한다.

질문) Load-store Architecture 퀴즈



1. 몇 번의 메모리 접근을 하는가?

load instruction을 실행하려면 fetch-decode-execute가 일어나면 이 과정에서 메모리 접근이 3번이 일어난다. 그리고 해당 load, store instruction을 실행시킬 때 execute가 한번씩 일어나기 때문에 총 5번이 일어난다.

1. 몇 번의 instruction 접근을 하는가?

3번

1. 몇 번의 data 접근을 하는가?

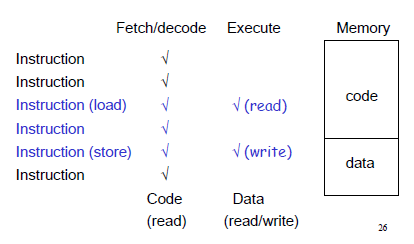
2번

1. 몇 번의 메모리 read를 하는가?

load instruction에서 fetch-decode-execute에서 메모리 read가 3번 일어나며 마지막 load instruction의 execute 자체가 read이기 때문에 총 메모리 read는 4번 일어난다.

1. 몇 번의 메모리 write를 하는가?

store의 execute가 write이기 때문에 1번 일어난다.



#the Constant Zero

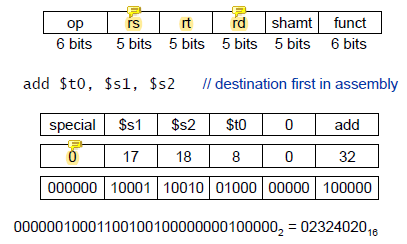
0번째 레지스터의 값을 항상 상수 값 0이 저장되어 있다. 그리고 데이터의 값은 변하지 않는다. 그렇다면 32개 중에서 하나를 사용하지 않는 꼴인데 왜 이렇게 사용하는가?

* 다른 instruction을 편리하게 사용하기 때문이다.
* 레지스터 사이에서 move를 생각해 보면 따로 move라는 instruction을 만들 필요 없이 하나의 레지스터와 zero 레지스터를 add를 하면 다른 레지스터에 하나의 레지스터의 값이 저장하게 되어진다.
* 레지스터의 값을 없앨 경우, 즉 clear를 할 때에는 해당 레지스터에서 zero 레지스터 두 개를 더해주면 해당 레지스터는 0의 값을 갖게 된다.

#Stored Program Computers

데이터를 binary로 표현하는 것처럼 프로그램 instruction도 binary로 표현되어야 한다.

#R-format Example



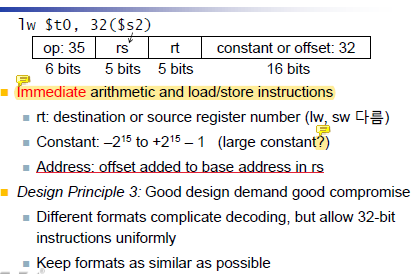
R-format은 그림과 같이 op, register source 두 개와 마지막 저장해주는 register destination 그리고 shift 연산을 할 때 사용되는 Shamt와 마지막 function 자리가 존재한다. funct에서는 opcode의 모자란 부분을 채워준다.

레지스터 기반으로 돌아가는 ALU 연산의 op는 0이다. 모든 opcode을 구분하기 위해서 function 코드로 구분한다. special에서 0을 보고 ALU 연산을 사용했다는 것을 짐작할 수 있다.

질문) 특정한 funct를 만들지 말고 opcode를 8-bit으로 왜 사용하지 않는가?

다른 field의 공간이 제한되어진다. opcode를 6-bit으로 제한 이유는 나머지 다른 field를 더 잘 사용하기 위함이다.

#I-format instructions



I-format에서는 op, register source, register destination, offset 필드가 존재한다. 상수 값은 instruction안에 들어 있기 때문에 레지스터에 따로 갈 필요 없이 instruction 안에 32의 값을 즉시 알 수 있다. 이것을 immediate number이라 한다.

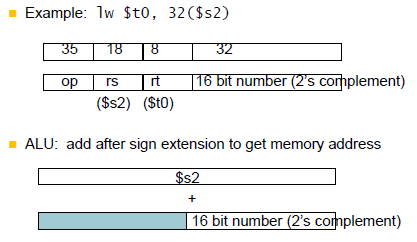
#To think about: R and I formats

format을 두 개가 있는 게 무슨 의미가 있을까? 다른 format을 처리하는 하드웨어를 만들어 줘야 한다. 많은 format을 만들면 하드웨어는 해당하는 format을 만들어 줘야한다. 그렇게 되면 하드웨어는 복잡해진다. MIPS는 format이 3개 밖에 없어서 하드웨어가 단순하고 빠르게 만들어 줄 수 있다.

opcode을 몇 bit를 사용할 것인가 그리고 레지스터를 몇 개를 사용할 것인지 대한 해당사항들은 16-bit 상수 값을 설정한 것에 연관되어 있다.

물론 6-bit opcode, 16 register, 18-bit offset, 8-bit opcode, 32 register, 14 bit offset 등 후보 format을 생각할 수 있다. 이 후보들 중에서 벤치마킹을 돌려서 가장 성능이 좋은 것을 선택을 하는 것이다.

#Sign Extension



sigh bit을 16개를 그대로 복제하면 값을 유지하면서 32-bit을 만들어 줄 수가 있다. 그 다음 ALU에 더해달라 하면 되는 것이다. 이것을 Sign Extension이라 한다.