레지스터

2021년 2월 1일 월요일 오후 6:53

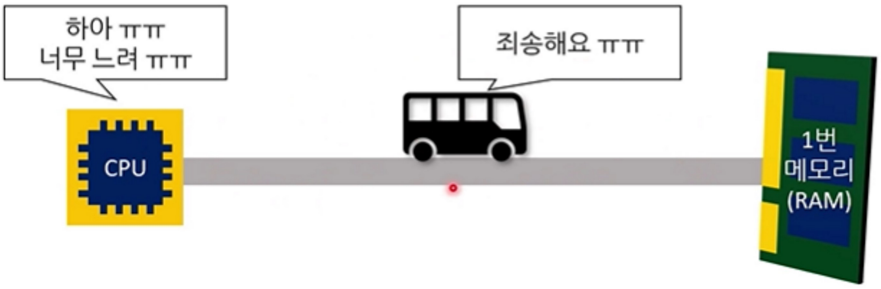
레지스터는 왜 있나요?

- 앞에서 CPU는 뇌, 메모리는 공책이라 했음
- 머릿속에서 생각한 것을 공책에 옮겨 적으려면 시간이 걸리듯이
  CPU가 생각한 것을 메모리에 적거나 그로부터 읽을 때도 시간이 걸림

메모리를 읽고 쓰는 게 느린 이유 1

- CPU가 메모리에 접근할 때마다 버스를 타야 함

- 즉, CPU가 연산할 때마다 메모리에 접근하는 시간이 발생
  - 버스가 크면 한 번에 많이 읽어오겠지만, 메모리 낭비가 생기고
  - 버스가 작으면 메모리 낭비는 적을 수 있겠지만, 여러 번 왔다갔다 할 수 있음



메모리를 읽고 쓰는 게 느린 이유 2

- 대부분 컴퓨터에 장착하는 메모리는 DRAM임
  - 영어로는 Dynamic Random Access Memory
- DRAM은 가격이 저렴한 대신, 한 가지 큰 단점이 있음
  - 기록된 내용을 유지하기 위해서 주기적으로 정보를 다시 써야 함
  - 다시 쓰는 동안 또 시간을 소모
- 이러한 단점이 없는 메모리가 있긴 함
  - SRAM(Static RAM)이라고 함
  - 비용이 DRAM에 비해 매우 비쌈
  - 이걸 몇 기가나 달기에는 좀 부담스러움

그래서 컴퓨터 공학자들은 고민하기 시작

- 컴퓨터 공학자들은 '비용은 저렴하지만, 속도가 빠른 컴퓨터'를 어떻게 만들지 고민했음
- 그래서 나온 방법이 SRAM을 CPU와 메모리 사이에 두는 것이었음
  - 단, 너무 비싸니 매우 적은 용량만
  - 일반적인 SRAM하고는 조금 다름
- CPU랑 가까이 두고 싶어서 아예 CPU안에 넣어버림
- 그게 바로 레지스터임

레지스터(register)

- register (동사, 형용사)
  1. 등록하다, 신고하다
  2. 기록하다
  3. 공식적인 서류나 기록
     - 레지스터는 CPU가 공식적으로 사용할 수 있는 저장공간 이라고 생각하면 됨
- 레지스터는 CPU가 사용하는 저장 공간 중에 가장 빠른 저장 공간
- CPU가 연산을 할 때 보통 레지스터에 저장되어 있는 데이터를 사용
- 그 연산결과도 레지스터에 다시 저장하는 게 보통
- 다시 강조하는데 레지스터는 흔히 말하는 '메모리'가 아님
  - CPU가 레지스터에 접근하는 방법과 메모리 접근하는 방법이 다른 게 보통
  - 레지스터는 메모리와 달리 주소 개념이 없다
- 사실 이걸 예전에 본 적이 있음

```
     8: int main(void)
     9: {
006C1020  push        ebp
006C1021  mov         ebp,esp
006C1023  sub         esp,1Ch
006C1026  mov         dword ptr [ebp-4],0
    10: int num1;
    11: int num2;
    12: int result;
    13:
    14: num1 = 100;
006C102D  mov         dword ptr [num1],64h
    15: num2 = 29;
006C1034  mov         dword ptr [num2],1Dh
    16:
    17: result = add(num1, num2);
006C103B  mov         eax,dword ptr [num2]
006C103E  mov         ecx,dword ptr [num1]
006C1041  mov         dword ptr [esp],ecx
006C1044  mov         dword ptr [esp+4],eax
006C1048  call        add (06C1000h)
006C104D  mov         dword ptr [result],eax
```

  - ebp, esp, eax, ecx 모두 다 레지스터!
  - ptr [num1] 이런 것들은 스택 메모리에 접근하는 것

② X86 아키텍쳐에서 사용하는 레지스터

- 8개의 범용 레지스터(general-purpose register)
  - ESP, EBP, EAX, EBX, ECX, EDX 등등
  - 변수만들거나 덧셈하는데 주로 사용
- 6개의 세그먼트 레지스터(segment register)
- 1개의 플래그 레지스터(flags register)

- 1개의 명령어 포인터(instruction pointer)
- 등등

변수들을 계산하는 C코드의 어셈블리어에서 자주 나타나는 패턴

1. 변수의 값을 메모리 어딘가에 저장
2. 그 변수의 값을 읽어와 레지스터에 저장
3. 레지스터에 저장된 값을 가지고 계산한 뒤 계산 결과도 레지스터에 저장
4. 그 결과를 다시 메모리에 저장

5. 그리고 다시 그 결과를 사용해서 계산하면 레지스터에 또 복사

- CPU에서 어떤 연산을 할 때 굳이 레지스터에 갖고와서 연산을 해야 되냐 의문이 든다
- 프로그래머가 이런 빠른 레지스터를 직접 사용할 수는 없을까?
  - 어차피 레지스터에 불러와서 계산하는거 그냥 메모리 안 거치고 레지스터만 쓰면 빠를거 같은데
- 어셈블리어로 코드를 작성하면 언제든지 사용 가능
  - 단 CPU에 탑재된 레지스터 수가 몇 개 없기에 모든 것을 레지스터에 저장하는 것은 불가능

② C 코드에서 레지스터를 사용하는 법은?

- 레지스터를 사용해달라고 부탁은 할 수 있음
  - 들어줄지 말지는 컴파일러마음임
- 따라서 가능하면 레지스터를 사용해 달라는 힌트 정도만 줄 수 있음
- 레지스터 사용을 '요청' 하는 예

- register 키워드

- 저장 유형 지정자 (storage-class specifier)
- 가능하다면 해당 변수를 레지스터에 저장할 것을 요청
- 실제로 레지스터를 사용할지 말지는 컴파일러가 결정
- 레지스터는 메모리가 아님!
- 따라서, **레지스터 변수들은 몇 가지 제약을 받음**

제약1: 변수의 주소를 구할 수 없음

- 레지스터에는 주소가 없으므로 레지스터 지정자를 사용한 변수의 주소를 구할 수 없다
- 옛날 하드웨어는 레지스터에도 주소가 있었음. 하지만 대부분 하드웨어는 없으므로 C표준에서 안된다고 정했음

제약2: 레지스터 배열을 포인터로 사용 불가

제약3: 블록 범위에서만 사용 가능

- 전역 변수에는 사용할 수 없음
  - 전역 변수는 데이터 섹션에 들어가고 프로그램 실행 중에 전역 변수의 공간이 잡혀 있어야 함

- ② 그러나.. 의미 없다..
  - 데스크톱 컴파일러들은 register 키워드를 넣어준다고 특별히 해주는 일이 없음
  - 대부분 무시함
- ② 예전 임베디드 시스템에서는 의미가 있었음
  - CPU도 매우 느렸고
  - 메모리 용량도 적었고
  - 최적화를 잘 해주지 않은 컴파일러 때문에 프로그래머가 레지스터 사용까지 직접 지시해야 했음
- ② 그러나 그건 옛날 이야기 이젠 다름
  - 이제는 보통 컴파일러가 배포(release)모드에서 알아서 최적화
    - 불필요한 스택 메모리 접근 없애고
    - 레지스터에만 있으면 빠를 거 같은 변수는 그렇게 해주고
  - 더 이상 프로그래머가 수동으로 사용하지 않는 키워드