제 7강. 1. Supporting Procedures in Computer Hardware 2. Communicating with People

2.8 하드웨어의 프로시져 지원

프로그램이 프로시져를 실행하는 6단계 ★ 암기!

- **프로시져** : 이해하기 쉽고 재사용 가능하도록 프로그램을 구조화하는 방법 중 하나

  ㄴ ‘스파이’ 에 비유할 수 있음.

  비밀 계획을 지니고 출발해서 필요한 자원을 획득하여 임무를 완수하고, 흔적을 없앤 후 원하는 결과를 가지고 출발 장소로 되돌아옴

1. 프로시져가 접근할 수 있는 곳에 인수를 넣는다.
   a. $a0 - $a3 : 4 argument registers
2. 프로시져로 제어를 넘긴다.
3. 프로시져가 필요로 하는 메모리 자원을 획득한다.
4. 필요한 작업을 수행한다.
5. 호출한 프로그램이 접근할 수 있는 장소에 결과값을 넣는다.
   a. $v0 - $v1 : 2 value registers for result values
6. 프로시져는 프로그램 내의 여러 곳에 호출될 수 있으므로 원래 위치로 제어를 돌려준다.
   a. $ra : 1 return address register to return to the point of origin

**** 프로시져 호출시 레지스터 16개를 할당**

- r0-r3 : 전달할 인수를 가지고 있는 인수 레지스터 4개
- lr : 호출한 곳으로 되돌아가기 위한 복귀 주소를 가지고 있는 링크 레지스터 1개

**** 프로시져를 위한 명령어**

- by ARM 어셈블리 언어
- **BL 명령어** (Branch-Link instruction) 지정된 주소로 점프하면서 동시에 다음 명령어의 주소를 lr 레지스터에 저장하는 명령
- **복귀 주소** : 레지스터 lr에 기억되는 링크

레지스터 사용법

- $a0 - $a3 : arguments (reg's 레지스터 4-7)
- $v0 - $v1 : result values (reg's 2 - 3)
- $t0 - $t9 : 임시값
  - 프로시져에 의해 overwritten될 수 있음
- $s0 - $s7 : 지역 변수 (local variables)를 저장하는 데 사용
- $gp : global pointer for static data (reg 28)
- $sp : stack pointer (reg 29)
- $fp : frame pointer (뒤에서 자세히 배움) (reg 30)
- $ra : return address (reg 31)

레지스터 스필링 (spilling register)

컴파일러가 자주 사용하는 변수를 가능한 한 많이 레지스터에 넣고 나머지 변수는 메모리에 저장했다가 필요할 때 꺼내서 레지스터에 넣는 것

→ **자주 사용하지 않는 변수를 메모리에 넣는 일**

→ 레지스터 스필링에 이상적인 구조는 **stack (스택)**

스택 포인터는 레지스터 값 하나가 스택에 저장되거나 복구될 때 마다 한 워드씩 조정됨

- push, pop

프로시져 호출 명령어

- Procedure call : jump and link
- jal : ProcedureLabel

    - 다음 명령어의 주소를 $ra에 전달
    - 타켓 주소로 건너뜀
- jr $ra
    - 프로그램 카운터(PC, 현재 실행하고 있는 명령어의 메모리 주소를 가짐)로 $ra를 복사함
    - computed jumps 할 때 사용

중첩된 프로시져 Leaf Procedure Example

- Leaf procedure : 다른 프로시져를 호출하지 않는 프로시져
- But, 모든 프로시져가 잎 프로시져는 아니고, 따라서 잎 프로시져가 아닌 것을 호출할 때는 조심해야 한다.

> 💡 자동(automatic) 변수 : 프로시져 내에서만 정의되는 것으로 프로시져가 종료되면 없어짐
> 정적(static) 변수 : 프로시져로 들어간 후나 프로시져에서 빠져나온 후에도 계속 존재

프로시져 호출 후에도 보존되는 것과 보존 안 되는 것

보존되는 것	보존 안 되는 것
변수 레지스터 : r4 - r11	인수 레지스터 : r0 - r3
스택 포인터 레지스터 : sp	스크래치 레지스터 : r12
링크 레지스터 : lr	스택 포인터 아래의 스택
스택 포인터보다 윗 부분의 스택	

중첩되지 않은 프로시져

C Code

```
int fact (int n)
{
    if (n < 1) return f;
    else return n * fact(n-1);
```

}

argument n in $a0

return value $v0

MIPS Code 알아야 하나,,,

새 데이터를 위한 스택 공간의 할당

- 프로시져의 저장된 지역 변수를 가지고 있는 스택 영역 : 프로시져 프레임, 액티베이션 레코드

그림 2.14

fp : frame pointer

sp : stack pointer

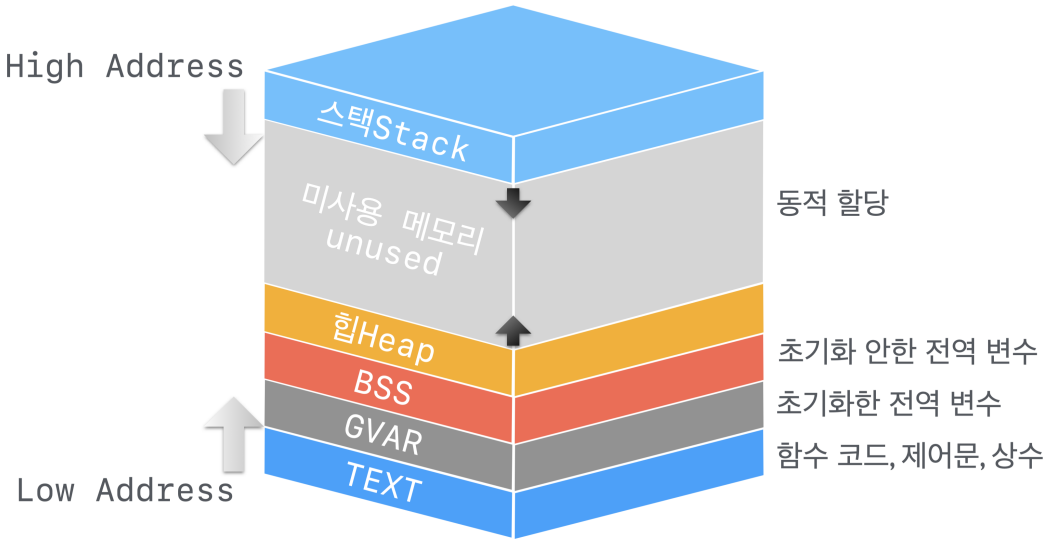
메모리 구조

그림 2.15

- 텍스트 : 프로그램 코드
  - pc
- 정적 데이터 : 전역 변수
  - 1000 8000hex ~ 1000 0000hex 중간 값
- 동적 데이터
  - malloc in C , new in Java
  - hip : 스택을 향해 자란다.
- 스택 : 스택 overflow 스택이 계속 증가해서 dynamic data의 hip 공간을 침범한 것
  - 지역 변수
  - activation record
  - automatic storage

새 데이터를 위한 힙 공간의 할당

Memory Model



- 스택 : 최상위 주소에서 시작해서 아래쪽으로 자람
  - 지역 변수
  - activation record
  - automatic storage
- 동적 데이터 : malloc in C, new in Java
- 힙 : LinkedLIst 같은 자료 구조
- 정적 데이터 세그먼트 : 상수와 기타 정적 변수들
- 텍스트 세그먼트 : ARM 기계어 코드
  - 프로그램 코드

2.9 문자와 문자열

문자 데이터

- Byte-encoded character sets
  - ASCII : 128 문자
    - 95 graphics, 33 control
  - Latin-1 : 256

- Unicode : 32-bit character set
  - Java, C++ 등에서 사용
  - 알파벳, 심볼 등 거의 구현 가능
  - UTF-8, UTF-16

** 자바는 문자 표현에 16비트를 사용하는 것이 디폴트

Byte / Halfword 연산

ARM 명령어 집합에는 하프워드(Halfword)라 불리는 16비트 데이터에 대한 적재 · 저장 명령이 포함돼 있음.

- LDRH(load register half)

ㄴ LDRB와 유사하게 하프워드를 부호없는 수로 취급하므로 레지스터의 상위 16비트를 0 확장하여 채움

- LDRSH(load register signed halfword) : 부호 있는 정수 적재
- LDRH >>>> LDRSH : 사용 빈도

LDRH r0, [sp, #0] : 메모리에서 16비트를 읽어와서 레지스터의 우측 16자리에 넣음

STRH r0, [r12, #0] : 레지스터의 우측 16비트를 메모리에 쓴다.

MIPS byte/halfword load/store

- String processing is a common case
  - lb rt, offset(rs) lh rt, offset(rs) : 32bits extend해서 레지스터에 넣음
  - lbu rt, offset(rs) lhu rt, offset(rs)
  - sb rt, offset(rs) sh rt, offset(rs)

String copy example

```
void strcpy (char x[], char y[])
{ int i;
i = 0;
while((x[i] = y[i]) != '\0')
  i += 1;
}
```

Leaf Procedure

- function이 다른 function을 부르지 않기 때문
- MIPS code 생략…

jal procedureLabel 명령어의 의미는?

jump and link

1. Address of following instruction put in $ra (명령어 다음에 있는 명령어로 명령어가 있는 메모리 주소를 $ra 레지스터에 넣음)
2. Jumps to target address (타겟 주소로 점프)