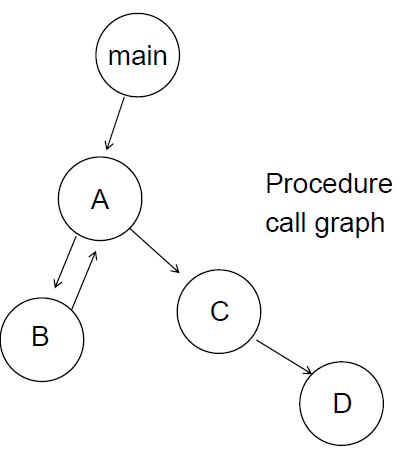
#HW9 2017069598 박상지

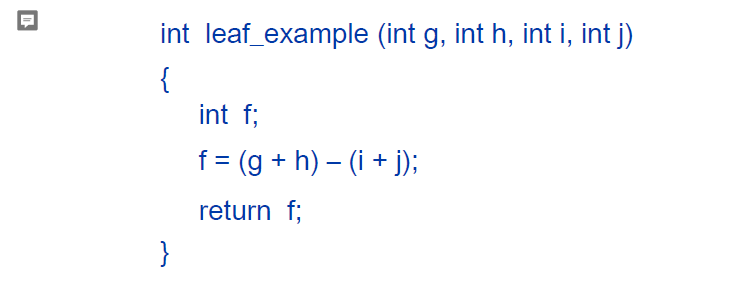
run c programs, procedure calls

프로그램이 실행시킨다는 것은?



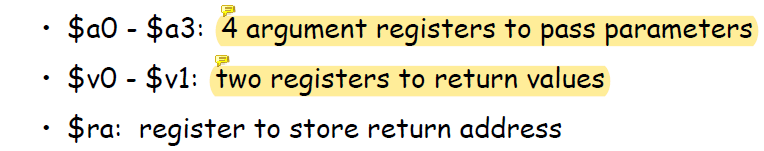
프로그램을 실행시킨다는 것은 ISA 관점에서 machine instruction을 하나씩 실행시킨 것, 또는 fetch-decode-execute을 반복적으로 실행하는 것이다. HLL(high level language) 관점에서는 statement를 하나씩 실행시키는 것, 또는 function call return이 반복되는 것이다(procedure level).

leaf procedure



해당 코드에서는 function에 대한 argument가 4개가 있는 간단한 코드가 있다. leaf procedure은 call graph에서 맨 끝에 위치해 있는 procedure, 즉 다른 procedure을 call하지 않는 것이다.

procedure에서는 caller와 callee로 나누어 지는데 caller 관점에서는 다른 callee를 call 하기 때문에 jump and link로 return address를 저장해야 한다. callee 입장에서는 caller가 사용한 레지스터 값을 생각을 해서 기존의 있는 레지스터 값을 백업을 해서 마음데로 쓸 수 있게 원하는 계산을 하고 적절한 방법으로 결과를 jump register를 사용하여 caller로 return 한다.



32개의 레지스터에서 4개의 argument 전용 레지스터를 따로 만들고, 2개의 return value 레지스터를 지정하였다. ra는 return address를 저장하는 레지스터이다.

질문) leaf procedures에서 4개 이상의 argument보다 필요한다면?

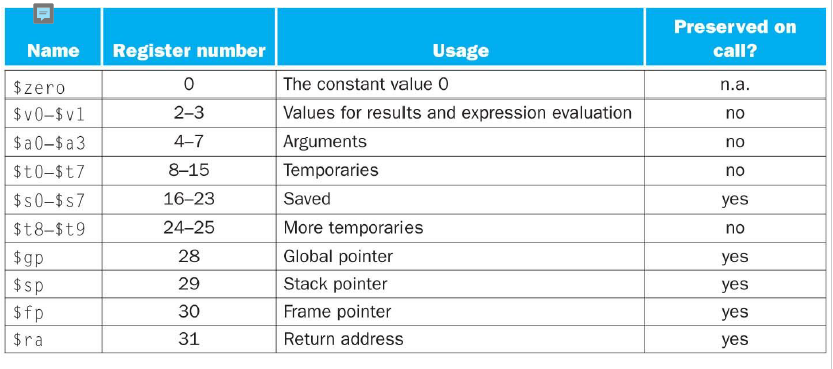
벤치 마크 프로그램 조사 결과 통상 argument의 사용 개수가 4개 이하였다. 그렇기 때문에 argument register를 4개로 지정한 것이다. 하지만 그 4개 수보다 많이 필요한다면 메모리에 접근하여 값을 할당하여 사용해야한다. 이럴 경우 속도는 줄어 든다.

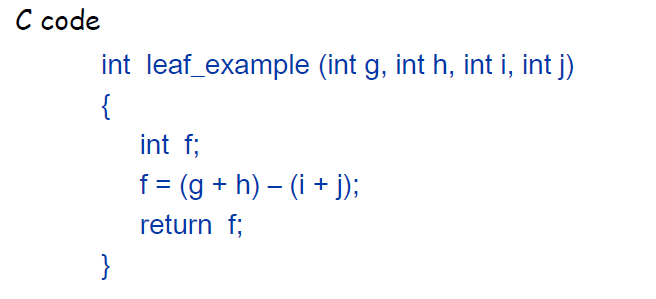
질문) return value의 register은 왜 2개 필요한가?

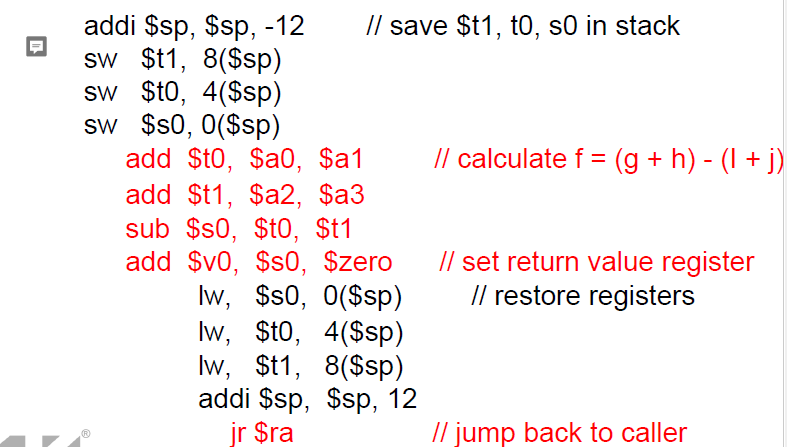
c프로그램은 return 값을 1개만 지정한다. 하지만 double return value는 64-bit의 공간이 필요하기 때문에 double value를 return 할 때를 대비하여 레지스터 2개를 만들어 놓은 것이다.

질문) 하나 이상의 아이템을 return할 때에는 어떻게 하는가?

여러 개의 아이템을 return을 할 때에는 구조체로 묶어서 포인터로 주소 값을 return을 하든가, executable value을 사용하여 return을 한다.



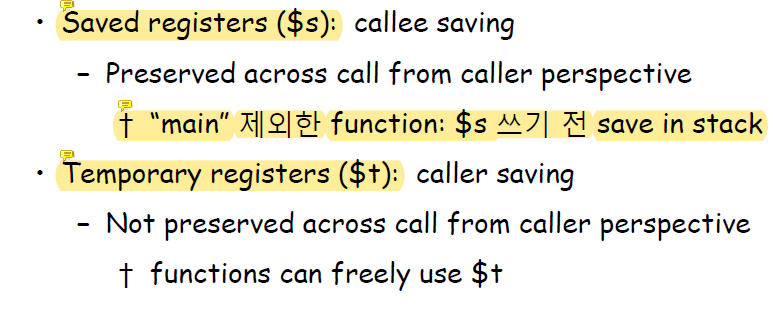




해당 C코드에 대한 MIPS 코드이다. 빨강색으로 칠한 부분이 실제로 코드가 실행되는 부분이다. 만약 해당 C코드를 실행하려면 s0, t0, t1 레지스터를 사용한다는 가정하에 해당 3개의 레지스터가 caller에서 중요한 값이 저장될 수 있기 때문에 stack 레지스터에 복원을 해야 한다. 3개 이기 때문에 12의 주소 값을 할당해줘야 하고, 실제 실행 코드를 실행시킨 다음에 다시 스택 레지스터에 저장된 값을 백업을 해줘야 한다. 그 다음에 return을 해준다.

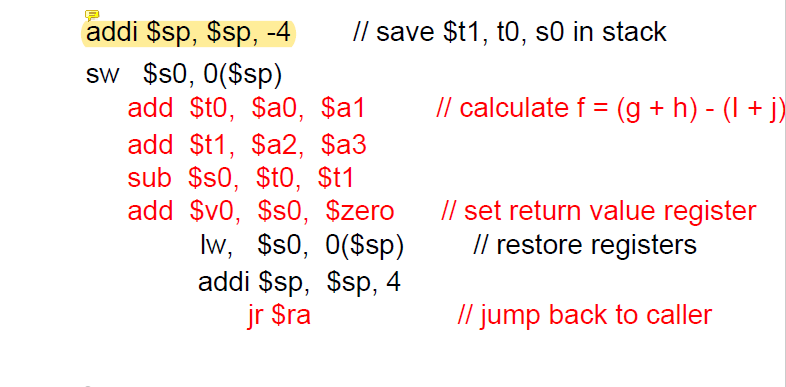
-컴파일러는 해당 레지스터에 값이 들어 있는지 알 수 없다. 최악의 상황을 생각해야 하기 때문에 안전하게 callee는 스택 레지스터에 복원 값을 저장해야 한다.

coordinating register usage

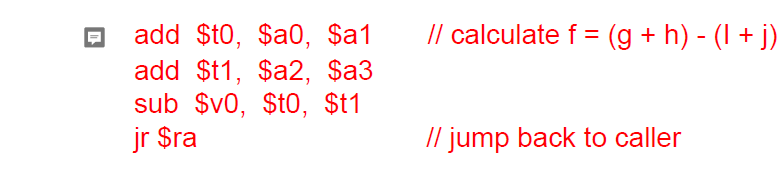


saved register는 callee 입장에서 복원을 담당하는 부분이기 때문에 caller 입장에서는 따로 백업을 할 필요가 없다. callee에서 function return call해서 돌아와도 해당 레지스터는 백업이 돼서 돌아온다. (이미 callee에서 saving을 해서 복원을 했기 때문이다.)

caller 입장에서 다른 function을 부르고 돌아올 경우 임시 레지스터는 바꿔서 돌아올 가능성이 있기 때문에 caller에서 백업을 해야 한다.



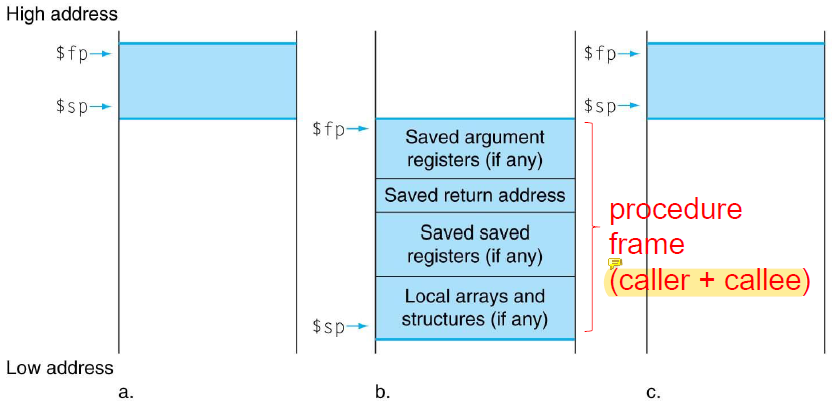
t 레지스터는 백업을 할 필요가 없기 때문에 해당 s 레지스터만 백업을 한다. 위에 MIPS 코드를 더욱 간단하게 한 것이다.

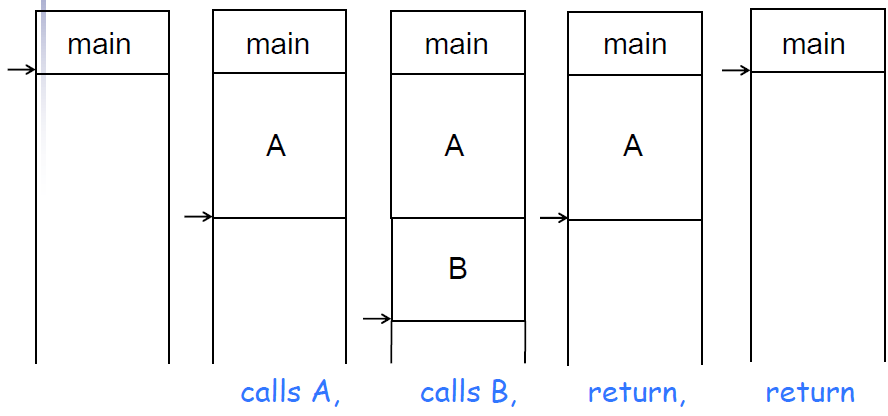


처음부터 saved register을 사용하지 않고 임시 레지스터의 연산 값을 value register에 저장해서 return 한다.

Non-Leaf Procedures

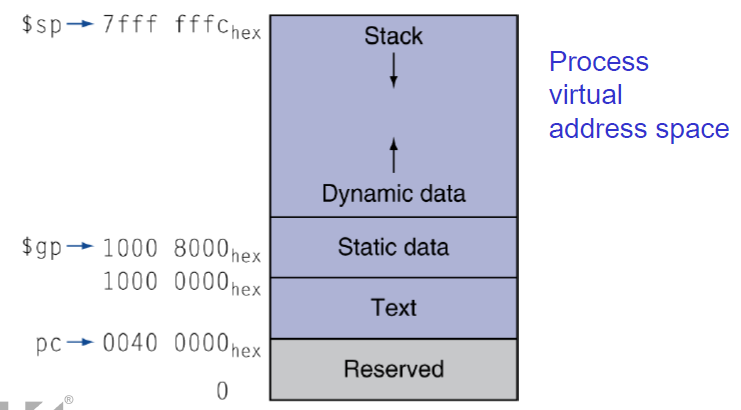
non-leaf procedures은 다른 procedure을 call하는 procedures로 해당 non-leaf procedure들은 main에서 사용했던 레지스터의 값들을 손상시키지 않는 범주 안에서 argument 레지스터를 사용하고 return address를 사용한다. 그리고 non-leaf procedure은 다른 procedure를 call하기 때문에 main에서 사용했던 레지스터와 해당 procedure에서 중요하게 사용했던 레지스터의 값들을 stack에 저장하고 main return address도 stack에 저장하고 leaf-procedure을 call해야 한다.





runtime stack

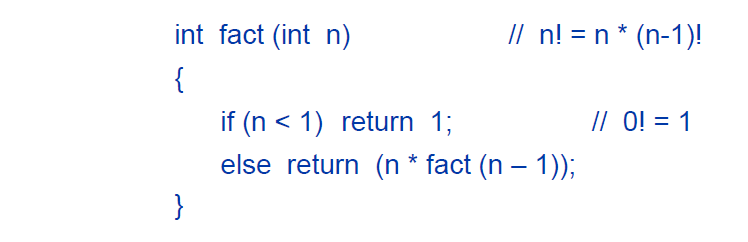
runtime stack이란 프로그램에서 현재 실행 중인 서브루틴에 관한 정보를 저장하는 스택 자료구조이다. 프로그램의 기능 수행에 있어 runtime stack의 관리가 중요함에도 불구하고, 상세한 구현은 고급 프로그래밍 언어에서는 보통 감추어지며 자동화되어 있다. 많은 컴퓨터들이 스택 관리를 위한 특별한 명령어를 제공하고 있다. 지역 변수는 컴파일러를 통해서 접근을 하는데 stack point에서 offset을 더해서 찾는다. 그러기 때문에 base addressing mode인 load word를 사용한다.



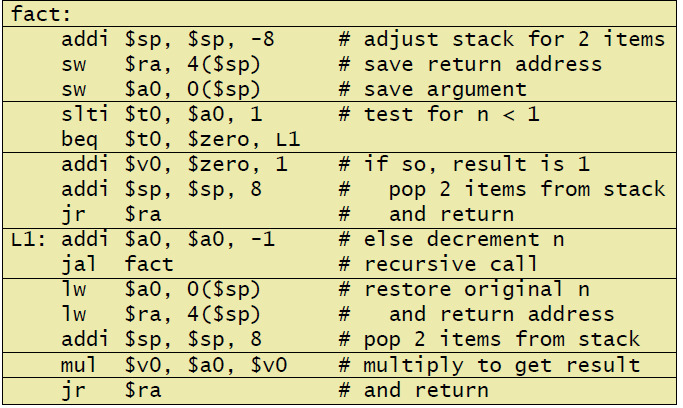
질문) stack을 사용하는 이유는?

1. 프로세서 콜 될 때 stack에 push되고 프로세서가 끝나면 pop을 한다. 만약 stack으로 하지 않는다면, 많은 프로세서를 위해 저장 공간을 따로 만들어야 한다. 그러면 비효율적으로 저장 공간을 사용하게 된다. 스택을 사용하면 프로세서 콜 할 때, push, pop을 하기 때문에 그렇게 많은 공간을 만들 필요가 없어 진다. 그렇기 때문에 공간을 효율적으로 사용 가능하다.
2. recursion- 똑 같은 프로세서가 계속 반복적으로 부를 때(self-call)이 몇 번 일어나는지 모를 때 저장 공간을 미리 만들어 놓으면 해당 공간을 얼마나 차지할 지 예상을 못하기 때문에 stack을 사용한다.

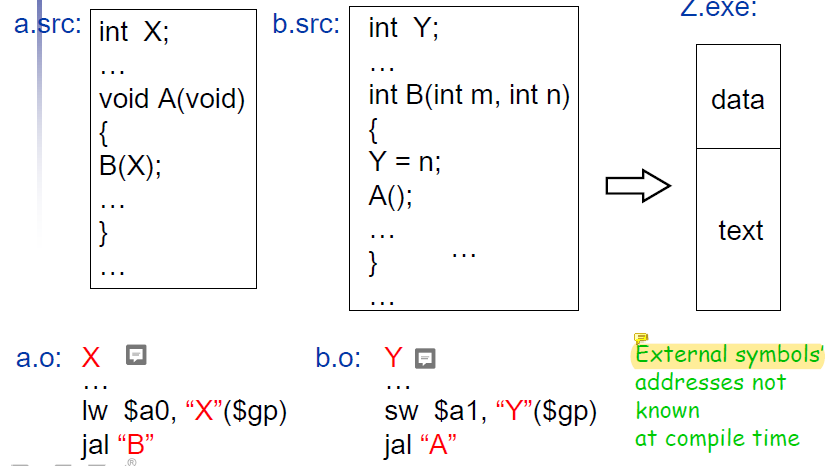
nested procedure call



해당 factorial function을 참고해서 보면 먼저 argument인 n을 받기 때문에 return address와 argument n을 stack에 push을 한다. 조건문으로 n이 1보다 작은 것을 표현하기 위해 slti연산을 사용하여 레지스터 저장하여 해당 레지스터가 0이면 beq연산을 사용하여 recursive call을 하여 argument n에 -1을 하여 다시 jal fact로 넘어간다. 만약 n이 1보다 작으면 새로운 레지스터에 1이라는 값을 할당하고 다른 레지스터는 사용하지 않았기 때문에 바로 스택을 다시 pop해주고 return을 해준다. 최종 return까지 오게 되면 load를 하여 처음 데이터를 복원해주고 새로운 레지스터에 argument를 더해서 곱해주고 return을 해준다.

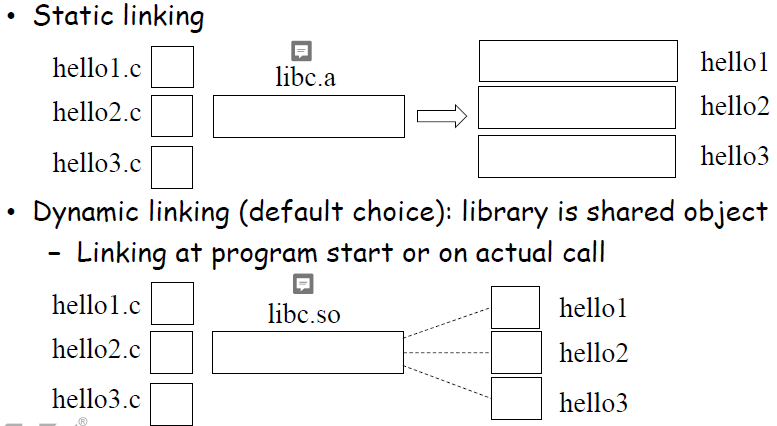


compile, link, and run



두 개의 소스 파일 보자면 첫번째 소스 파일은 x라는 변수가 있는데 해당 A function에서는 x라는 변수를 argument로 받는 B라는 function을 call 한다. MIPS 코드를 보면 static code에서는 x static data이기 때문에 x의 주소는 gp에서 얼만큼 떨어져 있는지 argument를 준비해 준다. 그 다음에 jump and link 명령을 통해서 B의 function에 jump를 한다.

static data는 Y값이 있고 두 번째 argument를 Y의 값을 저장해서 A의 function을 call한다.

  
static linking에서는 hello1.c … 3개의 c 프로그래밍 코드가 라이브러리를 통해서 object를 생성 하였는데 해당 object의 크기는 라이브러리와 합쳐서 만들어졌기 때문에 코드에 비해서 데이터가 커졌다. 하지만 dynamic linking에서는 라이브러리 shared object와 c 프로그래밍이랑 결합하지 않고 나중에 필요한 부분만 사용하기 위해서 linking을 한다. 그래서 상대적으로 execute 파일이 작다. 프로그램이 시작 또는 call로 인해서 불러졌을 때 linking으로 통해서 실행 된다.