Podem-se passar parâmetros através dos registos, a não ser que tenhamos mais de 6 parâmetros.

64: rdi, rsi, rdx, rcx, r8, r9

32: edi, esi, edx, ecx, r8d, r9d

16: di, si, dx, cx, r8w, r9w

8: dil, sil, dl, cl, r8b, r9b

Long f2(long c){

Return c+1;

}

F2:

#c em rdi

Incq %rdi #rdi = c+1

Movq %rdi, %rax #rax = c+1

Ret

Long f1(long a, long b, long c){

Return a+b+f2(c);

}

F1:

Addq %rsi, %rdi

Pushq %rdi #gravar na stack a soma e restaurá-la depois

Movq %rdx, %rdi

Call f2

Popq %rdi #restaurar a soma

Addq %rdi, %rax

ret

int f2(int a, char b){

return a\*b;

}

F2:

#a em edi, b em sil

Movsbl %sil, %eax #sign extended sil to eax

Imull %edi #edx:eax=a\*b

Ret

Void f1(int a, char b, int \*res){

\*res = f2(a,b);

}

F1:

#a em edi, b em sil, &res em rdx

Pushq %rdx #salvaguardar rdx na stack

Cal f2

Popq %rdx #restaurar rdx após o call

Movl %eax, (%rdx) #\*res = a\*b

Ret

P, procedimento invocador:

- copia argumentos de 1 a 6 para os registos apropriados

- coloca argumentos 7 a n na stack

- 7 no topo da stack pq é o primeiro a sair

Tamanhos de dados são arredondados para múltiplos de oito

Q, o invocado, acede aos primeiros 6 argumentos a partir dos registos e os outros a partir da stack

O stack pointer rsp aponta para o topo da stack

NÃO USAR ISTO PORQUE SE FIZERMOS UM PUSH O VALOR DE RSP VAI SER ATUALIZADO E DEPOIS VAMOS BUSCAR OS VALORES ERRADOS!!:

Arg\_8: 16(%rsp)

Arg\_7: 8(%rsp)

Ret\_address: (%rsp)

Solução: Copiar o valor do resgisto rsp para o registo rbp no início da função. Os pushs alteram o rsp mas não o rbp.

Prologue/Setup: aloca espaço no inicio do procedimento

Epilogue/Cleanup: desaloca o que foi alocado no Prologue/Setup

Function:

#prologue

Push %rbp #salvaguardar na stack o valor de rbp vindo do invocador

Movq %rsp, %rbp #copiar o stack pointer atual para rbp

#corpo da função

Epilogue

Movq %rbp, %rsp #colocar em sp o valor que estava no inicio desta função

Popq %rbp #restaurar o valor

Ret

Depois da invocação, a função invocadora deve remover os argumentos que colocou na stack, retornando assim a stack ao estado que tinha antes da invocação

Para remover podemos usar a instrução pop, mas pode-se também apenas mover o stack pointer (registo rsp) para o valor que tinha antes da invocação

Function:

#...

Pushq %rax #coloca o oitavo argumento para a função utilfunc na stack

Pushq %rbx #coloca o setimo argumento para a função utilfunc na stack

Call utilfunc #restaura o estado da stack

#...

Ret

Long f1(long a, long b, long c, long d, long e, long f){

Return a+f2(a\*b,b,c,d,e,f,10,-20);

}

F1:

#a em rdi, b em rsi, c em rdx, d em rcx, e em r8, f em r9

Pushq %rdi #salvaguarda rdi na stack

Movq %rdi, %rax #rax=a

Imulq %rsi #rdx:rax=a\*b

#preparar a chamada da funçao

Movq %rax, %rdi #rdi=a\*b (primeiro parâmetro de f2)

Pushq $-20

Pushq $10 #setimo parâmetro de f2

Call f2 #f2(a\*b,b,c,d,e,f,10,-20)

Addq $16, %rsp #remove o setimo e oitavo parâmetro da stack

Popq %rdi #restaura a em rdi

Addq %rdi, %rax #rax=a+f2(a\*b,b,c,d,e,f,10,-20)

Ret

Long f2(long a, long b, long c, long d, long e, long f, long g, long h){

Return h/g

}

F2:

#a em rdi, b em rsi, c em rdx, d em rcx, e em r8, f em r9, g em 16(%rbp), h em 24(%rbp)

Pushq %rbp #prologue

Movq %rsp, %rbp

Pushq %rbx

#utilizar os parametros

Movq 16(%rbp), %rbx #rbx=g (obtido da stack)

Movq 24(%rbp), %rax #rax=h (obtido da stack)

Cqto #prepara o dividendo, extendendo o sinal de rax para rdx

Idivq %rbx #rax = h/g

Popq %rbx

Movq %rbp, %rsp #epilogue

Popq %rbp

Por vezes dados locais precisam de ser armazenados na stack.

Ex: quando não há registos suficientes para armazenar todos os dados locais

O espaço para variáveis locais deve ser reservado após definir o registo rbp para apontar para o topo da stack

Function:

Pushq %rbp

Mpvq %rsp, %rbp

Subq $16, %rsp #reserva 16 bytes para variáveis locais

#...

Variáveis locais ficam em lugares como -8(%rbp) e -16(%rbp) -> acedidas através do registo rbp, mas com valores negativos

Com a desalocação do epilogo, ele já trata de eliminar da stack as variáveis locais