2017 Exam - Symbolic Programming
CS3011

INSTRUCTOR: Tim Fernando
Tim.Fernando@tcd.ie

Two Hour Exam

Answer 3 out of 4 Questions

20 marks per question

40 mins per question

Question 1

i) Briefly describe IA32 and x64 procedure calling conventions. Make sure that you describe the register set, draw a diagram of a typical procedure stack frame and list the volatile registers. In the case of x64 explain the use of shadow space. [6 marks]

IA32

When calling a function in IA32 all parameters must be pushed onto the stack from right to left. So if calling *min(a,b,c)* you must first push a, then b, then c. Once parameters have been pushed you must then call and enter the function.

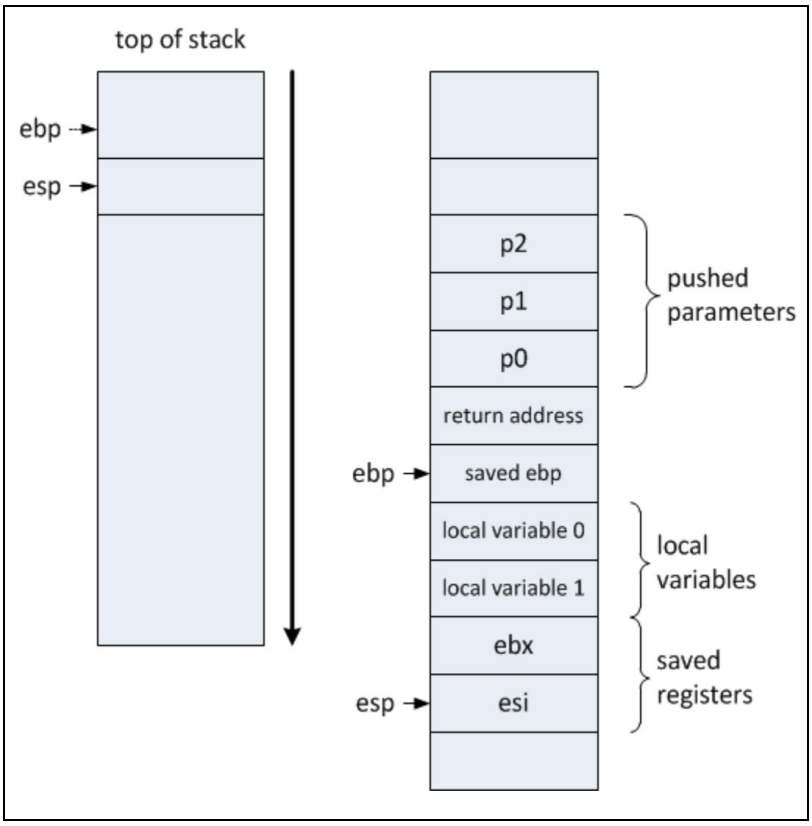
1. Push parameters to stack (right to left)
2. Call and enter function

Within the function you must initially push the frame pointer to the stack to preserve it for when exiting. If there are any local variables used, space must be allocated for these upon the stack initially. Also, if any non volatile registers are used they must also be pushed to the stack and preserved.

3. Preserve frame pointer (**push ebp**)
4. Update frame pointer with stack pointer (**mov ebp, esp**)
5. Allocate space for local variables (**sub esp, 8** -> space for two local variables)
6. Preserve non-volatile registers if used (**push ebx**)

From here the function body can proceed and fulfill its desired functionality. Before exiting the function must perform the following:

7. Restore non-volatile registers saved (**pop ebx**)
8. De-allocate space for local variables (**add esp, 8**)
9. Restore esp (**mov esp, ebp**)
10. Restore frame pointer (**pop ebp**)
11. Return (**ret 0**)



Parameters are pushed right to left onto the stack.

p0 @ ebp + 8

p1 @ ebp + 12

p2 @ ebp + 16

Local variables are accessed using a negative offset.

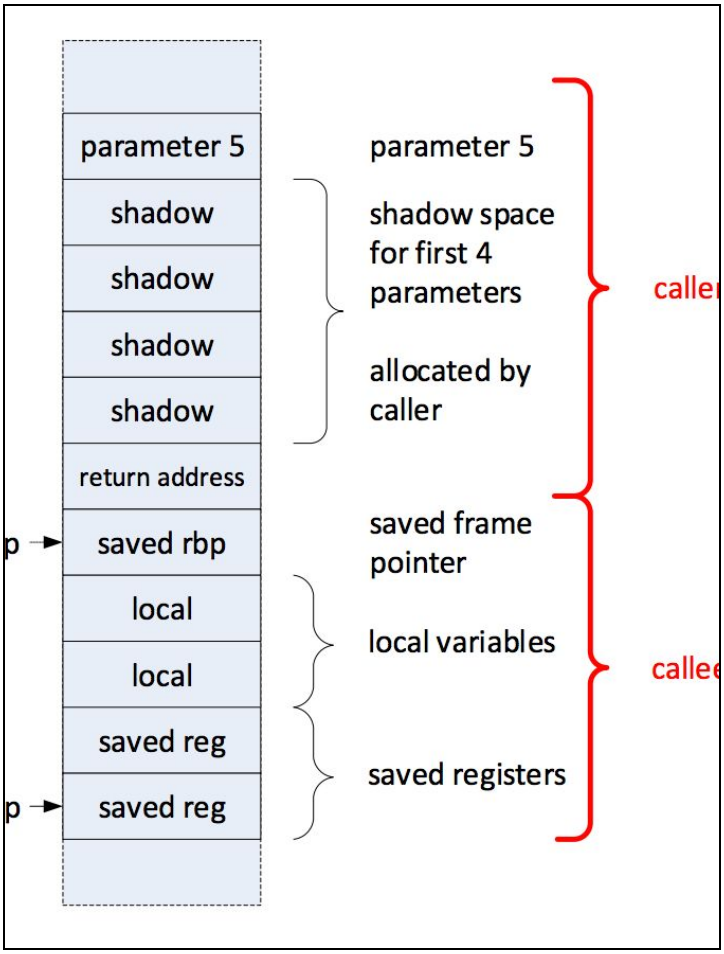
local0 @ ebp - 4

local1 @ ebp - 8

X64

X64 function calling is quite similar to that of IA32 just with an extended register set and what's known as *shadow space*. Shadow space is 32 bytes of pre-allocated space that every function caller must allocate before calling a function and de-allocate following a return. This way within every function there is 4 x 8 bytes available for parameters. Any additional parameters must be pushed onto the stack. Parameters, like in IA32 are passed from right to left.

1. Pass parameters to 1-4 to *rcx, rdx, r8 and r9* (**mov rcx, rax**)
2. Allocate shadow space (**sub esp, 32**)
3. Call function (**call min**)
4. De-allocate shadow space (**add esp, 32**)
5. Result found in rax



- *rax, rcx, rdx, r8, r9, r10, r11*

all volatile registers

Parameters are pushed right to left onto the stack.

p0 @ rcx

p1 @ rdx

p3 @ r8

p4 @ r9

ii) What is the main advantage of the x64 procedure calling convention compared with the IA32 procedure calling convention? [2 marks]

The main advantage of x64 procedure calling vs IA32 is the implementation of shadow space. This allows for 4 registers to be pre-allocated for use with parameters or any other desired usage for every given function. This removes the need for unique stack and frame pointer maintenance and handling throughout functions.

It implements a generic protocol/procedure that every function call must follow and allows for programmers to focus on writing their code and not worrying about stack and frame pointers being handled incorrectly.

iii) Convert the following code segment into IA32 and x64 assembly.

```
INT g = 4;

INT max(INT a, INT b, INT c) {
    INT v = a;
    if (b > v)
        v = b;
    if (c > v)
        v = c
    return v;
}

INT p(INT i, INT j, INT, k, INT l) {
    return max(max(g, i, j), k, l);
}
```

IA32

```
; function to calculate max(a, b, c)
;             a -> [ebp+8]
;             b -> [ebp+12]
;             c -> [ebp+16]
;
; returns result in eax

public    max                   ; make sure function name is exported

min:      push  ebp             ; push frame pointer
          mov   ebp, esp        ; update ebp

          mov   eax, [ebp+8]    ; v = a
          mov   ecx, [ebp+12]   ; ecx = b
          cmp   ecx, eax        ; if (b > v)
          jle   max_1           ;
          mov   eax, ecx        ; v = b

max_1:    mov   ecx, [ebp+16]   ; ecx = c
          cmp   ecx, eax        ; if (c > v)
          jle   max_2           ;
          mov   eax, ecx        ; v = c
max_2:    mov   esp, ebp        ; restore esp
          pop   ebp             ; restore ebp
          ret   0               ; return
```

```
; function to calculate p(i, j, k, l)
;              i -> [ebp+8]
;              j -> [ebp+12]
;              k -> [ebp+16]
;              l -> [ebp+20]
;
; returns max( max(g, i, j), k, l)

public    p                    ; make sure function name is exported

min:      push  ebp            ; push frame pointer
          mov   ebp, esp       ; update ebp

          push  [ebp+12]       ; push j for max(g, i, j)
          push  [ebp+8]        ; push i for max(g, i, j)
          push  g              ; push g for max(g, i, l)

          call max             ; eax = max(g, i, j)

          push [ebp+20]        ; push l for max(eax, k, l)
          push [ebp+16]        ; push k for max(eax, k, l)
          push eax             ; push eax for max(eax, k, l)

          call max             ; max( max(g, i, j), k, l)

          mov esp, ebp         ; restore stack pointer
          pop ebp              ; restore frame pointer
          ret 0                ; return 0
```

x64

```
; function to calculate max(a, b, c)
;              a -> rcx
;              b -> rdx
;              c -> r8
;
; returns result in eax

public    max                  ; make sure function name is exported

min:      mov   rax, rcx       ; v = a
          cmp   rdx, rax       ; if (b > v)
          jle   max_1          ;
          mov   rax, rdx       ; v = b

max_1:    cmp   r8, rax        ; if (c > v)
          jle   max_2          ;
          mov   rax, r8        ; v = c

max_2:    ret   0              ; return
```

```
; function to calculate p(i, j, k, l)
;              i -> rcx
;              j -> rdx
;              k -> r8
;              l -> r9
;
; returns max( max(g, i, j), k, l)

public    p                    ; make sure function name is exported

p:        push r8               ; preserve r8 (k)
          mov r8, rdx           ; r8 = j
          mov rdx, rcx          ; rdx = i
          mov rcx, g            ; rcx = g

          add esp, 32           ; allocate shadow space
          call max              ; rax = max (g, i, j)
          sub esp, 32           ; de-allocate shadow space

          mov r8, r9            ; r8 = l
          pop rdx               ; rdx = k
          mov rcx, rax          ; rcx = max(g,i,j)

          add esp, 32           ; allocate shadow space
          call max              ; rax = max(max(g, i, j), k, l)
          sub esp, 32           ; de-allocate shadow space

          ret 0                 ; return
```