Họ và tên: ……Trần Gia Nguyên Phong……. MSSV: ……22002575………………….

**LAB 09: MACHINE PROCEDURES**

**Thực hành**

**Practice Problem 3.30**The following code fragment occurs often in the compiled version of library routines:

1 call next  
2 next:  
3 popl %eax  
A. To what value does register %eax get set?

%eax is set to the address of popl instruction  
B. Explain why there is no matching ret instruction to this call.

This is not a true procedure, since the control follow same ordering as instruction and return address is popped in the stack.  
C. What useful purpose does this code fragment serve?

This is the only way in IA32 to get the value of the program counter (pc, %eip) into an interger register.

**Practice Problem 3.31**The following code sequence occurs right near the beginning of the assembly code generated by gcc for a C procedure:

1 subl $12, %esp  
2 movl %ebx, (%esp)  
3 movl %esi, 4(%esp)  
4 movl %edi, 8(%esp)  
5 movl 8(%ebp), %ebx  
6 movl 12(%ebp), %edi  
7 movl (%ebx), %esi  
8 movl (%edi), %eax  
9 movl 16(%ebp), %edx  
10 movl (%edx), %ecx

We see that just three registers (%ebx, %esi, and %edi) are saved on the stack (lines 2–4). The program modifies these and three other registers (%eax, %ecx, and %edx). At the end of the procedure, the values of registers %edi, %esi, and %ebx are restored (not shown), while the other three are left in their modified states.

Explain this apparent inconsistency in the saving and restoring of register states.

Register %edi, %esi, %ebx are callee – save

The procedure must save them on stack before altering their value and restore them before returning.

The other three register are caller-save. They can be altered without affecting the behavior of the caller.

**Practice Problem 3.32**A C function fun has the following code body:

\*p = d;  
return x-c;

The IA32 code implementing this body is as follows:

1 movsbl 12(%ebp),%edx  
2 movl 16(%ebp), %eax  
3 movl %edx, (%eax)  
4 movswl 8(%ebp),%eax  
5 movl 20(%ebp), %edx  
6 subl %eax, %edx  
7 movl %edx, %eax

Write a prototype for function fun, showing the types and ordering of the arguments p, d, x, and c.

The prototype of the function

* Int fun (short c, char d, int\* p, int x);

**Practice Problem 3.33**Given the C function

1 int proc(void)  
2 {  
3 int x,y;  
4 scanf("%x %x", &y, &x);  
5 return x-y;  
6 }

gcc generates the following assembly code:

1 proc:  
2 pushl %ebp  
3 movl %esp, %ebp  
4 subl $40, %esp  
5 leal -4(%ebp), %eax  
6 movl %eax, 8(%esp)  
7 leal -8(%ebp), %eax  
8 movl %eax, 4(%esp)  
9 movl $.LC0, (%esp) *Pointer to string "%x %x"*10 call scanf

*Diagram stack frame at this point*11 movl -4(%ebp), %eax  
12 subl -8(%ebp), %eax  
13 leave

14 ret

Assume that procedure proc starts executing with the following register values:

Register Value

%esp 0x800040

%ebp 0x800060

Suppose proc calls scanf (line 10), and that scanf reads values 0x46 and 0x53 from the standard input. Assume that the string “%x %x” is stored at memory location 0x300070.  
A. What value does %ebp get set to on line 3?

Start with %esp value 0x800040, pushl on line 2 %esp -4 > address 0x80003C, this is new value of %ebp.

B. What value does %esp get set to on line 4?

Line 4 decrement %esp by 40 (0x28)

> 0x80009C - 0x28 = 0x800014

C. At what addresses are local variables x and y stored?

Line 5,7 compute argument pass to scanf, Line 6,8 store them on stack. X stores 0x800038, Y stores 0x800034.

D. Draw a diagram of the stack frame for proc right after scanf returns. Include as much information as you can about the addresses and the contents of the stack frame elements.

|  |  |
| --- | --- |
| 0x46 | 0x300070 |
| 0x53 | 0x300066 |
|  |  |
|  | 0x800060 |
|  | 1. The stacks frame |
|  | 0x800040 |
| 0x800060 | 0x80003C |
| 0x53 | 38 |
| 0x46 | 34 |
|  | 30 |
|  | 26 |
|  | 28 |
|  | 24 |
|  | 20 |
| 0x800038 | 1C |
| 0x800034 | 18 |
| 0x800070 | 0x800014 |

E. Indicate the regions of the stack frame that are not used by proc.

Byte address 0x800020 - 0x800033 are unused

**Practice Problem 3.34**For a C function having the general structure

int rfun(unsigned x) {

if ( x==0 ) //testl %ebx, %ebx

return 0 ;

unsigned nx = x>>1; //shrl %eax

int rv = rfun(nx);

return x&0x1 + rx; //andl81, %edx

} //leal(%edx, %eax), %eax  
gcc generates the following assembly code (with the setup and completion code omitted):  
1 movl 8(%ebp), %ebx  
2 movl $0, %eax  
3 testl %ebx, %ebx  
4 je .L3

5 movl %ebx, %eax  
6 shrl %eax *Shift right by 1*  
7 movl %eax, (%esp)  
8 call rfun  
9 movl %ebx, %edx  
10 andl $1, %edx  
11 leal (%edx,%eax), %eax  
12 .L3:

1. What value does rfun store in the callee-save register %ebx?

Register %ebx hold value of parameter x, so that it can be used to compute result expression  
B. Fill in the missing expressions in the C code shown above.  
C. Describe in English what function this code computes.

Function compute sum of the bit in argument x. It recursive compute sum of all but the least significant bit, then adds the least significant bit to get the result.