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

**LAB 09: MACHINE CONTROL**

**Practice Problem 3.8**Suppose we want to generate assembly code for the following C function:

int shift\_left2\_rightn(int x, int n)

{

x <<= 2;

x >>= n;

return x;  
}

The code that follows is a portion of the assembly code that performs the actual shifts and leaves the final value in register %eax. Two key instructions have been omitted. Parameters x and n are stored at memory locations with offsets 8 and 12, respectively, relative to the address in register %ebp.

1 movl 8(%ebp), %eax *Get x*

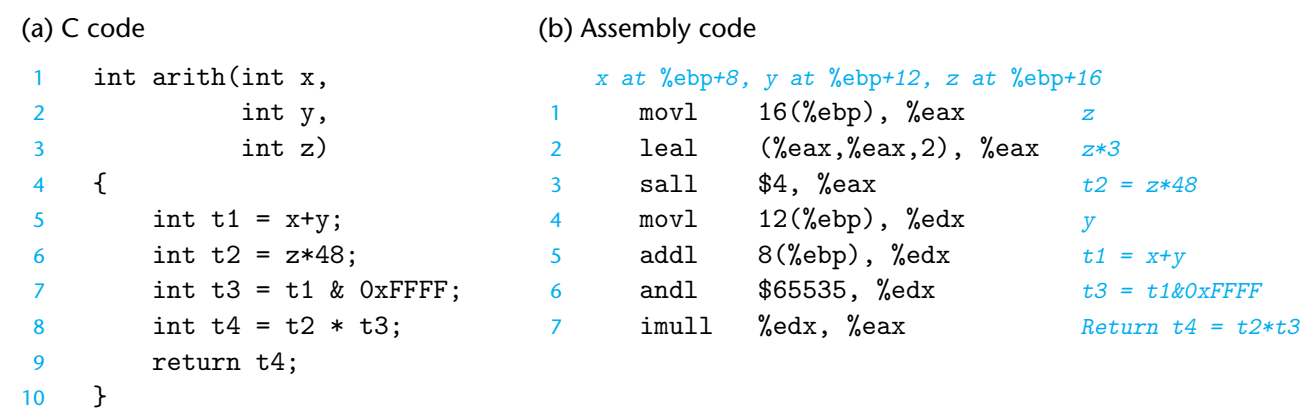
2 sall $2, %eax *x <<= 2*

3 movl 12(%ebp), %ecx *Get n*

4 sarl %ecx, %eax\_\_\_\_\_\_\_ *x >>= n*

Fill in the missing instructions, following the annotations on the right. The right shift should be performed arithmetically.

**Practice Problem 3.9**In the following variant of the function of Figure



the expressions have been replaced by blanks:  
1 int arith(int x,  
2 int y,  
3 int z)  
4 {  
5 int t1 = x^y;  
6 int t2 = t1 >> 3;  
7 int t3 = ~t2;  
8 int t4 = t3 - z;  
9 return t4;  
10 }

The portion of the generated assembly code implementing these expressions is as follows:  
*x at* %ebp*+8, y at* %ebp*+12, z at* %ebp*+16*1 movl 12(%ebp), %eax  
2 xorl 8(%ebp), %eax  
3 sarl $3, %eax  
4 notl %eax  
5 subl 16(%ebp), %eax  
Based on this assembly code, fill in the missing portions of the C code.

**Practice Problem 3.10**It is common to find assembly code lines of the form

xorl %edx,%edx

in code that was generated from C where no Exclusive-Or operations were present.

1. Explain the effect of this particular Exclusive-Or instruction and what useful operation it implements.

Instruction is set %edx to zero, because x^x = 0 with with any x, C statement x = 0

B. What would be the more straightforward way to express this operation in assembly code?

Direct way is set %edx to zero = movl $0, %edx

C. Compare the number of bytes to encode these two different implementations of the same operation.

Disassembling code => xorl require 2 byte instruction

* Movl require 5 byte instruction.

**Practice Problem 3.13**The following C code

int comp(data\_t a, data\_t b) {

return a COMP b;

}  
shows a general comparison between arguments a and b, where we can set the data type of the arguments by declaring data\_t with a typedef declaration, and we can set the comparison by defining COMP with a #define declaration.  
Suppose a is in %edx and b is in %eax. For each of the following instruction sequences, determine which data types data\_t and which comparisons COMP could cause the compiler to generate this code. (There can be multiple correct answers; you should list them all.)

1. cmpl %eax, %edx  
   setl %al
2. cmpw %ax, %dx  
   setge %al
3. cmpb %al, %dl  
   setb %al
4. cmpl %eax, %edx  
   setne %al

**Practice Problem 3.15**

In the following excerpts from a disassembled binary, some of the information has been replaced by Xs. Answer the following questions about these instructions.

A. What is the target of the je instruction below? (You don’t need to know anything about the call instruction here.)

804828f: 74 05 je XXXXXXX

8048291: e8 1e 00 00 00 call 80482b4

B. What is the target of the jb instruction below?

8048357: 72 e7 jb XXXXXXX

8048359: c6 05 10 a0 04 08 01 movb $0x1,0x804a010

C. What is the address of the mov instruction?

XXXXXXX: 74 12 je 8048391

XXXXXXX: b8 00 00 00 00 mov $0x0,%eax

D. In the code that follows, the jump target is encoded in PC-relative form as a 4-byte, two’s-complement number. The bytes are listed from least significant to most, reflecting the little-endian byte ordering of IA32. What is the address of the jump target?

80482bf: e9 e0 ff ff ff jmp XXXXXXX

80482c4: 90 nop

E. Explain the relation between the annotation on the right and the byte coding on the left.

80482aa: ff 25 fc 9f 04 08 jmp \*0x8049ffc