Computer Organisation And Systems

Assignment-3

Submitted by
Sayam Kumar
S20180010158

Problem 3.58

Parameters x, y and z are passed in registers %rdi, %rsi and %rdx. As the return value is stored in %rax. We will use a variable result to store result value.

decode 2:

		Explanation
subq	%rdx, %rsi	y=y-z ; Dest = Dest - Src
imulq	%rsi, %rdi	x = x * y
movq	%rsi, %rax	result=y
salq	$63, %rax	result = result << 63
sarq	$ 63, % rax	result <<= 63
xorq	%rdi, % rax	result = result ^ x
ret		

The corresponding C code is -

```
long decode2 (long x, long y, long z)
{
    long result = y-z;          // subq
    return (result << 63 >> 63) ^ (result * x);
}
```

$\underbrace{\texttt{result << 63}}_{\text{sal}}$, $\underbrace{\texttt{>> 63}}_{\text{sar}}$, $\underbrace{\texttt{result * x}}_{\text{imulq}}$, $\underbrace{\texttt{\^{}}}_{\text{xor}}$

Now explaining the Ccode -
First, y-z is stored in result.
Then corresponding left and right shift are done and result is xor with result * x. That's why the C code is written.

Final ans -

```c
long decode2 (long x, long y, long z)
{
    long result = y-z;
    return (result << 63 >> 63) ^ (result * x);
}
```

Problem 3.60 x in %rdi, n in %esi

Instruction	Operands	Explanation
loop:		
movl	%esi, %ecx	Storing n in %ecx
movl	\$1, %edx	A variable (call Mask) = 1
movl	\$0, %eax	result = 0
jmp	.L2	
.L3		
movq	%rdi, %r8	A variable (call temp) = x
andq	%rdx, %r8	temp = temp & mask
orq	%r8, %rax	result |= temp
salq	%cl, %rdx	mask <<= n
.L2		
testq	%rdx, %rdx	mask & mask
jne	.L3	mask != 0
rep; ret		

Filling in the C code -

```c
long loop (long x, long n)
{
    long result = 0;
    long mask;
                          testq
    for (mask = 1; mask != 0 ; mask = mask << n) {
                                            salq
            result |= mask & x;
                   orq    andq
    }
    return result;
}
```

A Which registers hold program values x, n, result and mask?

Sol- It is given in question that x is stored in %rdi, n in %esi. Through the assembly code, it is clear that result is stored in %rax as %rax stores return value and mask is stored in %rdx

B What are the initial values of result and mask?

Sol- Result = 0, mask = 1

because movl $1, %edx => mask = 1

movl $0, %eax => result = 0

C What is the test condition for mask?

Sol- mask != 0

because testq %rdx, %rdx => mask != 0

jne .L3

D How does mask get updated?

Sol- mask = mask << n

because salq %cl, %rdx => mask = mask << n

E How does the result get updated?

Sol result |= mask & x

because

andq %rdx, %r8 => result |= mask & x

orq %r8, %rax

F Already written on previous page.

Problem 3.62 Understanding the assembly Code line by line

p1 in %rdi, p2 in %rsi, action in %edx

.L8 — Explanation

movl	$27, %eax	result = 27
ret		break

.L3

movq	(%rsi), %rax	result = *p2
movq	(%rdi), %rdx	%rdx = *p1
movq	%rdx, (%rsi)	*p2 = %rdx
ret		break

.L5

movq	(%rdi), %rax	result = *p1
addq	(%rsi), %rax	result += *p2
movq	%rax, (%rdi)	*p1 = result
ret		break

.L6

movq	$59, (%rdi)	*p1 = 59
movq	(%rsi), %rax	result = *p2
ret		break

.L7

movq	(%rsi), %rax	result = *p2
movq	%rax, (%rdi)	*p1 = result
movl	$27, %eax	result = 27
ret		break

.L9

movl	$12, %eax	result = 12
ret		break

Now we have understood what each line is doing, we can put them all in C code on next page

```
long switch 3 (long *p1, long *p2, mode-t action)
{ long result = 0;
  switch (action) {
    case MODE_A:
        result = *p2;
        *p2 = *p1;
        break;
    case MODE_B:
        result = *p1 + *p2;
        *p1 = result;
        break
    case MODE_C;
        *p1 = 59;
        result = *p2;
        break;
    case MODE_D:
        *p1 = *p2;
    case MODE_E:
        result = 27;
        break;

    ~~case~~
    default:
        result = 12;
        break;
  }
  return result;
}
```

The reason why we have written break in case MODE_D because even MODEE is calculating the same thing of result =27. So no break statement in MODE-D.

Problen ~~6.63~~ ~~6.63~~ 3.63 x in %rdi, n in %rsi

for the jump table

0x4006f8:	0x 4005 a1 (60)	0x 4005c3 (61)
0x400708:	0x 4005a1 (62)	0x 4005aa (63)
0x400718:	0x 4005b2 (64)	0x 4005bf (65)

Now understanding the assembly line by line

		Explanation
sub	$0x3c, %rsi	n=n-60
cmp	$0x5, %rsi	n-5 calculating
ja	4005c3 <switch_prob +0x33>	jump above ⇒ switch cases are 60, 61, 62, 63, 64, 65.

So the address in jump table now map to 60, 61, 62, 63, 64, 65 sequentially.

lea	0x0(,%rdi,8), %rax	result = %rdi * 8 break
retq		
mov	%rdi, %rax	result = x
sar	$0x3, %rax	result >>= 3 break
retq		
mov	%rdi, %rax	result = x
shl	$0x4, %rax	result <<= 4
sub	%rdi, %rax	result -= x
mov	%rax, %rdi	x = result
imul	%rdi, %rdi	x *= x
lea	0x4b(%rdi), %rax	result = x+75 break
retq		

By looking at the labels of addresses given in front of these assembly instructions. the following C code can be written.

```
long switch-prob (long x, log n) {
    long result = x';
    switch (n) {
        Case 60:

        Case 62:
            result *= 8;
            break;
        case 63:
            result >>= 3;
            break;
        case 64:
            x = (x<<4) - x;
        case 65:
            x *= x;
        default:
            result = x+75;
            break;
    }
    return result;
}
```

Explanation - As the no ret statements in assembly for labels 64, 65, we donot write break statement in these cases. Moreover for case 61, it maps to default case as it is evident from the address of jump tables for label 61 (0x4005c3).

Scanned by CamScanner

Problem ~~6.64~~ 3.64

long store_ele (long i, long j, long k, long * dest)

i in %rdi, j in %rsi, k in %rdx, dest in %rcx

Store_ele:

		Explanation
leaq	(%rsi, %rsi, 2), %rax	rax = 3rsi
leaq	(%rsi, %rax, 4), %rax	rax = 13 rsi
movq	%rdi, %rsi	rsi = i
salq	$6, %rsi	rsi = 64i
addq	$rsi, %rdi	rdi = 65i
addq	%rax, %rdi	rdi = 65i + 13rsi
addq	%rdi, %rdx	k = k + 13j + 65i
movq	A(, %rdx, 8), %rax	result = A[8k]
movq	%rax, (%rcx)	* dest = A[8(k+13j+65i)]
movl	$3640, %eax	sets size of A as 3640
ret		

A) Extending the solution of 2D for 3D matrices -

$\& A[i][j][k] = x\_A + \text{size of}(\text{long})\,(S*T*i + T*j + k)$

B) $A[8(13(5i+j)+k)]$ from * dest

$T=13$, $S=5$

Size of A = 3640

$\Rightarrow R = \frac{A}{8 \times T \times S} = \frac{3640}{8 \times 13 \times 5} = 7$ (8 = size of (long))

So $R=7$, $S=5$, $T=13$

Scanned by CamScanner

Problem ~~6.65~~ 3.65

The assembly code for transpose of a matrix is given below →

Exploration

```
.L6:
    movq  (%rdx), %rcx
    movq  (%rax), %rsi
    movq  %rsi, (%rdx)
    movq  %rcx, (%rax)
    addq  $8, %rdx
    addq  $120, %rax
    cmpq  %rdi, %rax
    jne   .L6
```

rcx = A[i][j]

rsi = A[j][i]

$\left.\begin{array}{l} A[i][j] = rsi \\ A[j][i] = rcx \end{array}\right\}$ swap

$\left.\begin{array}{l} \text{taking next element} \\ \text{going to next row} \end{array}\right]$ size of long = 8

rax - rdx compare for loop

~~A) A - %rdx holds %.~~

A) %rdx holds A[i][j]

B) %rax holds A[j][i]

C) As size of long = 8 and we are 120 bytes ahead of A to get next row $\Rightarrow M = \frac{120}{8} = 15$

Problem 3.61

For this trial implementation of a condition move instruction, this can lead to segmentation fault -

```
long cread (long *xp) {
    return (xp? *xp :0);
}
```

Cread:

		Explanation
movq	(%rdi), %rax	v = *p ← can lead to segmentation fault
testq	%rdi, %rdi	Test x
movl	$0, %edx	Set ve = 0
Cmove	%rdx, %rax	If x = 0, v = ve
ret		Return v

To avoid this,
we can write

```
long cread-alt (long *xp)
{
    long temp = 0L;
    return * (xp? xp : & temp);
}
```

Hence, null value reference can be resolved with respect to adding a temporary variable and returning its address.

Problem 3.68

for the function setVal

mov 8(%rsi), %rax will store q→t. As int is 4 bytes long

$4 < B \leq 8$ — ①

Similarly

32(%rsi) will fetch q→u. Now as long it is 8 bytes long.

$\Rightarrow 2A < 20$

$A \leq 10$ — ②

Now (%rdi + 184) represents y

No of elements in AB ⇒

$4AB \leq 184 \Rightarrow AB \leq 46$ — ③

With equations ①, ② and ③

$B = 5$, $A = 9$ is the solution

Problem 3.71 The C program is as follows

```
#include <stdio.h>
# include <assert.h>
# define BUFFER 50
void good-echo(){
    char string[BUFFER];
    While (1){
        char *p= fgets(string, BUFFER, stdin);
        if (p==NULL){
            printf("No string entered.");
            break;
        }
        printf("%s\n", p);
    }
    return;
}
```

```
int main(){
    good_echo();
    return 0;
}
```

The error handled is in when no string is entered or there is a buffer overflow.

fgets throws error on ~~buffer over~~ bufferoverflow.

Problem 3.66   n in %rdi, A in %rsi, j in %rdx

sum_col: Explanation

```
leaq   1(,%rdi,4), %r8          t1 = 4n+1
leaq   (%rdi,%rdi,2), %rax      t2 = 3n
movq   %rax, %rdi               t3 = 3n
testq  %rax, %rax               test = 3n
jle    .L4                      3n <= 0
salq   $3, %r8                  t1 = 8t1 = 8(4n+1)
leaq   (%rsi,%rdx,8), %rcx      t4 = 8j + A
movl   $0, %eax                 t2 = 0
movl   $0, %edx                 t5 = 0

.L3
   addq (%rcx), %rax            t2 = *t4 = *(A+8j)
   addq $1, %rdx                t5 = t5+1
   addq $r8, %rcx               t4 + t1 = t4
   cmpq %rdi, %rdx              Cmp t5 & t3
   jne  .L3                     if t5 != 3n loop
   rep ret

.L4:
   movl $0, %eax                t5 return 0 %rax=0
   ret
   cmpq %rdi, %rdx              t5, t3 compare
   jne  .L3                     t5 != n3
```

(NR) n = 3n

```
leaq 1(,rdi,4), %r8             t1 = 4n+1
salq $3, %r8                    t1 = 8t1
addq %r8, %rcx                  t4 = t1+t4
```

In every loop, pointer move 8(4n+1)   NC(n) = 4n+1