## Practice Problem 3.5 (solution page 363)

You are given the following information. A function with prototype

```
void decode1(long *xp, long *yp, long *zp);
```

is compiled into assembly code, yielding the following:

```
  void decode1(long *xp, long *yp, long *zp)
  xp in %rdi, yp in %rsi, zp in %rdx
decode1:
  movq    (%rdi), %r8
  movq    (%rsi), %rcx
  movq    (%rdx), %rax
  movq    %r8, (%rsi)
  movq    %rcx, (%rdx)
  movq    %rax, (%rdi)
  ret
```

Parameters `xp`, `yp`, and `zp` are stored in registers `%rdi`, `%rsi`, and `%rdx`, respectively.

Write C code for `decode1` that will have an effect equivalent to the assembly code shown.

## Practice Problem 3.6 (solution page 363)

Suppose register `%rbx` holds value $p$ and `%rdx` holds value $q$. Fill in the table below with formulas indicating the value that will be stored in register `%rax` for each of the given assembly-code instructions:

| Instruction | Result |
|---|---|
| `leaq 9(%rdx), %rax` | ______ |
| `leaq (%rdx,%rbx), %rax` | ______ |
| `leaq (%rdx,%rbx,3), %rax` | ______ |
| `leaq 2(%rbx,%rbx,7), %rax` | ______ |
| `leaq 0xE(,%rdx,3), %rax` | ______ |
| `leaq 6(%rbx,%rdx,7), %rax` | ______ |

## Practice Problem 3.7 (solution page 364)

Consider the following code, in which we have omitted the expression being computed:

```
short scale3(short x, short y, short z) {
  short t = _________;
  return t;
}
```

Compiling the actual function with GCC yields the following assembly code:

```
  short scale3(short x, short y, short z)
  x in %rdi, y in %rsi, z in %rdx
scale3:
  leaq (%rsi,%rsi,9), %rbx
  leaq (%rbx,%rdx), %rbx
  leaq (%rbx,%rdi,%rsi), %rbx
  ret
```

Fill in the missing expression in the C code.

## Practice Problem 3.8 (solution page 364)

Assume the following values are stored at the indicated memory addresses and registers:

| Address | Value | Register | Value |
|---|---|---|---|
| 0x100 | 0xFF | %rax | 0x100 |
| 0x108 | 0xAB | %rcx | 0x1 |
| 0x110 | 0x13 | %rdx | 0x3 |
| 0x118 | 0x11 | | |

Fill in the following table showing the effects of the following instructions, in terms of both the register or memory location that will be updated and the resulting value:

| Instruction | Destination | Value |
|---|---|---|
| addq %rcx,(%rax) | _________ | _________ |
| subq %rdx,8(%rax) | _________ | _________ |
| imulq $16,(%rax,%rdx,8) | _________ | _________ |
| incq 16(%rax) | _________ | _________ |
| decq %rcx | _________ | _________ |
| subq %rdx,%rax | _________ | _________ |

## Practice Problem 3.18 (solution page 368)

Starting with C code of the form

```
short test(short x, short y, short z) {
    short val = __________;
    if (__________) {
        if (__________)
            val = __________;
        else
            val = __________;
    } else if (__________)
        val = __________;
    return val;
}
```

GCC generates the following assembly code:

```
  short test(short x, short y, short z)
  x in %rdi, y in %rsi, z in %rdx
test:
  leaq    (%rdx,%rsi), %rax
  subq    %rdi, %rax
  cmpq    $5, %rdx
  jle     .L2
  cmpq    $2, %rsi
  jle     .L3
  movq    %rdi, %rax
  idivq   %rdx, %rax
  ret
.L3:
  movq    %rdi, %rax
  idivq   %rsi, %rax
  ret
.L2:
  cmpq    $3, %rdx
  jge     .L4
  movq    %rdx, %rax
  idivq   %rsi, %rax
.L4:
  rep; ret
```

Fill in the missing expressions in the C code.

## Practice Problem 3.20 (solution page 369)

(solution page 369)

In the following C function, we have left the definition of operation `OP` incomplete:

```
#define OP ________ /* Unknown operator */

short arith(short x) {
    return x OP 16;
}
```

When compiled, GCC generates the following assembly code:

```
  short arith(short x)
  x in %rdi
arith:
  leaq    15(%rdi), %rbx
  testq   %rdi, %rdi
  cmovns  %rdi, %rbx
  sarq    $4, %rbx
  ret
```

A. What operation is `OP`?

B. Annotate the code to explain how it works.

### Practice Problem 3.24 (solution page 371)

For C code having the general form

```
short loop_while(short a, short b)
{
    short result = __________;
    while (__________) {
        result = __________;
        a = __________;
    }
    return result;
}
```

GCC, run with command-line option -Og, produces the following code:

```
    short loop_while(short a, short b)
    a in %rdi, b in %rsi
1   loop_while:
2     movl    $0, %eax
3     jmp     .L2
4   .L3:
5     leaq    (,%rsi,%rdi), %rdx
6     addq    %rdx, %rax
7     subq    $1, %rdi
8   .L2:
9     cmpq    %rsi, %rdi
10    jg      .L3
11    rep; ret
```

We can see that the compiler used a jump-to-middle translation, using the jmp instruction on line 3 to jump to the test starting with label .L2. Fill in the missing parts of the C code.

## Practice Problem 3.25 (solution page 371)

For C code having the general form

```
long loop_while2(long a, long b)
{
    long result = _________;
    while (_________) {
        result = _________;
        b = _________;
    }
    return result;
}
```

GCC, run with command-line option -O1, produces the following code:

```
  a in %rdi, b in %rsi
loop_while2:
  testq   %rsi, %rsi
  jle     .L8
  movq    %rsi, %rax
.L7:
  imulq   %rdi, %rax
  subq    %rdi, %rsi
  testq   %rsi, %rsi
```

### Practice Problem 3.32 (solution page 375)

The disassembled code for two functions `first` and `last` is shown below, along with the code for a call of `first` by function `main`:

```
   Disassembly of last(long u, long v)
   u in %rdi, v in %rsi
 0000000000400540 <last>:
   400540:  48 89 f8           mov    %rdi,%rax          L1: u
   400543:  48 0f af c6        imul   %rsi,%rax          L2: u*v
   400547:  c3                 retq                      L3: Return

   Disassembly of last(long x)
   x in %rdi
 0000000000400548 <first>:
   400548:  48 8d 77 01        lea    0x1(%rdi),%rsi     F1: x+1
   40054c:  48 83 ef 01        sub    $0x1,%rdi          F2: x-1
   400550:  e8 eb ff ff ff     callq  400540 <last>      F3: Call last(x-1,x+1)
   400555:  f3 c3              repz retq                 F4: Return
     .
     .
     .
   400560:  e8 e3 ff ff ff     callq  400548 <first>     M1: Call first(10)
   400565:  48 89 c2           mov    %rax,%rdx          M2: Resume
```

Each of these instructions is given a label, similar to those in Figure 3.27(a). Starting with the calling of `first(10)` by `main`, fill in the following table to trace instruction execution through to the point where the program returns back to `main`.

| Instruction | | | State values (at beginning) | | | | | |
|---|---|---|---|---|---|---|---|---|
| Label | PC | Instruction | %rdi | %rsi | %rax | %rsp | *%rsp | Description |
| M1 | 0x400560 | callq | 10 | — | — | 0x7fffffffe820 | — | Call first(10) |
| F1 | | | | | | | | |
| F2 | | | | | | | | |
| F3 | | | | | | | | |
| L1 | | | | | | | | |
| L2 | | | | | | | | |
| L3 | | | | | | | | |
| F4 | | | | | | | | |
| M2 | | | | | | | | |

## Practice Problem 3.35 (solution page 376)

For a C function having the general structure

```
long rfun(unsigned long x) {
    if ( ________ )
        return ________;
    unsigned long nx = ________;
    long rv = rfun(nx);
    return ________;
}
```

GCC generates the following assembly code:

```
    long rfun(unsigned long x)
    x in %rdi
rfun:
  pushq   %rbx
  movq    %rdi, %rbx
  movl    $0, %eax
  testq   %rdi, %rdi
  je      .L2
  shrq    $2, %rdi
  call    rfun
  addq    %rbx, %rax
.L2:
  popq    %rbx
  ret
```

A. What value does `rfun` store in the callee-saved register `%rbx`?

B. Fill in the missing expressions in the C code shown above.

## Practice Problem 3.37 (solution page 377)

Suppose $x_P$, the address of short integer array `P`, and long integer index $i$ are stored in registers `%rdx` and `%rcx`, respectively. For each of the following expressions, give its type, a formula for its value, and an assembly-code implementation. The result should be stored in register `%rax` if it is a pointer and register element `%ax` if it has data type `short`.

| Expression | Type | Value | Assembly code |
|---|---|---|---|
| `P[1]` | ______ | ______ | ______ |
| `P + 3 + i` | ______ | ______ | ______ |
| `P[i * 6 - 5]` | ______ | ______ | ______ |
| `P[2]` | ______ | ______ | ______ |
| `&P[i + 2]` | ______ | ______ | ______ |

### Practice Problem 3.38 (solution page 377)

Consider the following source code, where $M$ and $N$ are constants declared with `#define`:

```
long P[M][N];
long Q[N][M];

long sum_element(long i, long j) {
    return P[i][j] + Q[j][i];
}
```

In compiling this program, GCC generates the following assembly code:

```
long sum_element(long i, long j)
i in %rdi, j in %rsi
sum_element:
  leaq    0(,%rdi,8), %rdx
  subq    %rdi, %rdx
  addq    %rsi, %rdx
  leaq    (%rsi,%rsi,4), %rax
  addq    %rax, %rdi
  movq    Q(,%rdi,8), %rax
  addq    P(,%rdx,8), %rax
  ret
```

Use your reverse engineering skills to determine the values of $M$ and $N$ based on this assembly code.

## Practice Problem 3.41 (solution page 379)

Consider the following structure declaration:

```
struct test {
    short *p;
    struct {
        short x;
        short y;
    } s;
    struct test *next;
};
```

This declaration illustrates that one structure can be embedded within another, just as arrays can be embedded within structures and arrays can be embedded within arrays.

The following procedure (with some expressions omitted) operates on this structure:

```
void st_init(struct test *st) {
    st->s.y  = __________;
    st->p    = __________;
    st->next = __________;
}
```

A. What are the offsets (in bytes) of the following fields?

```
   p: ________
 s.x: ________
 s.y: ________
next: ________
```

B. How many total bytes does the structure require?

C. The compiler generates the following assembly code for `st_init`:

```
void st_init(struct test *st)
st in %rdi
st_init:
  movl    8(%rdi), %eax
  movl    %eax, 10(%rdi)
  leaq    10(%rdi), %rax
  movq    %rax, (%rdi)
  movq    %rdi, 12(%rdi)
  ret
```

On the basis of this information, fill in the missing expressions in the code for `st_init`.