Bryant Chapter 3

3.5

```c
void decode1(long *xp, long *yp, long *zp) {
    // xp in %rdi, yp in %rsi, zp in %rdx
    long x = *xp;
    long y = *yp;
    long z = *zp;

    *yp = x;
    *zp = y;
    *xp = z;

    return;
}
```

3.6

`%rbx` holds p and `%rdx` holds q.

Instruction	Result
`leaq 9(%rdx), %rax`	`%rax` $\leftarrow q + 9$
`leaq (%rdx, %rbx), %rax`	`%rax` $\leftarrow p + q$
`leaq (%rdx, %rbx, 3), %rax`	`%rax` $\leftarrow 3p + q$
`leaq 2(%rbx, %rbx, 7), %rax`	`%rax` $\leftarrow 7p + p + 2$
`leaq 0xE(,%rdx, 3), %rax`	`%rax` $\leftarrow 3q + 14$
`leaq 6(%rbx, %rdx, 7), %rax`	`%rax` $\leftarrow 7q + p + 6$

3.7

```asm
; short scale3(short x, short y, short z)
; x in %rdi, y in %rsi, z in %rdx
scale3:
    leaq (%rsi, %rsi, 9), %rbx        ; rbx = 9y + y = 10y
    leaq (%rbx, %rdx), %rbx           ; rbx = z + rbx = z
        + 10y
    leaq (%rbx, %rdi, %rsi), %rbx     ; rbx = xy + rbx =
        xy + z + 10y
    ret
```

```c
short scale3(short x, short y, short z) {
    short t = x * y + z + 10 * y;
    return t;
}
```

3.8

Initial values:

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

Effects of the instructions:

Instruction	Destination	Value
`addq %rcx, (%rax)`	$M[0x100]$	0x1
`subq %rdx, 8(%rax)`	$M[0x108]$	0xA8
`imulq $16, (%rax, %rdx, 8)`	$M[0x118]$	0x110
`incq 16(%rax)`	$M[0x110]$	0x14
`decq %rcx`	`%rcx`	0x0
`subq %rdx, %rax`	`%rax`	0xFD

3.18

```c
short test(short x, short y, short z) {
    short val = y + z - x;
    if (z > 5) {
        if (y > 2)
            val = x / z;
        else
            val = x / y;
    } else if (z < 3)
        val = z / y;
    return val;
}
```

3.20

```
; x in %rdi
arith:
    leaq 15(%rdi), %rbx     ; %rbx <- x + 15
    testq %rdi, %rdi        ; test if x is zero
    cmovns %rdi, %rbx       ; if x > 0, %rbx <- x
    sarq $4, %rbx           ; shift %rbx right by 4 bits
    ret
```

The `OP` operation is division. We use the bias of 15 to round the result of division to the nearest integer, since we are dividing by 16.

3.24

```
short loop_while(short a, short b) {
    short result = 0;
    while (a > b) {
        result = result + a * b;
        a = a - 1;
    }
    return result;
}
```

```
; short loop_while(short a, short b)
; a in %rdi, b in %rsi
loop_while:
    movl $0, %eax
    jmp .L2
.L3:
    leaq (,%rsi,%rdi), %rdx
    addq %rdx, %rax
    subq $1, %rdi
.L2:
    cmpq %rsi, %rdi
    jg .L3
    rep; ret
```

3.25

```
long loop_while2(long a, long b) {
long result = b;
while (b > 0) {
    result = result * a;
    b = b - 1;
}
return result;
}
```

```
; a in %rdi, b in %rsi
loop_while2:
    testq %rsi, %rsi
    jle .L8
    movq %rsi, %rax
.L7:
    imulq %rdi, %rax
    subq $1, %rsi
    testq %rsi, %rsi
    jg .L7
    rep; ret
.L8:
    movq %rsi, %rax
    ret
```

3.32

Label	PC	Instruction	%rdi	%rsi	%rax	%rsp	* %rsp	Description
M1	0x400560	callq	10	–	–	0x7fffffffe820	–	Call first(10)
F1	0x400548	lea	10	–	–	0x7fffffffe818	0x400565	Load x+1 into %rdi
F2	0x40054c	sub	10	11	–	0x7fffffffe818	0x400565	Subtract 1 from %rdi
F3	0x400550	callq	9	11	–	0x7fffffffe818	0x400565	Call last(x-1, x+1)
L1	0x400540	mov	9	11	–	0x7fffffffe810	0x400555	Move x to %rax
L2	0x400543	imul	9	11	9	0x7fffffffe810	0x400555	Multiply x by 11
L3	0x400547	ret	9	11	99	0x7fffffffe810	0x400555	Return 99 from last
F4	0x400555	repz retq	9	11	99	0x7fffffffe818	0x400565	Return 99 from first
M2	0x400565	mov	10	11	99	0x7fffffffe820	–	Move 99 to %rdx

3.35

```
long rfun(unsigned long x) {
    if (x == 0)
        return 0;
    unsigned long nx = x >> 2;
    long rv rfun(nx);
    return x + rv;
}
```

```
; long rfun(unsigned long x)
; x in %rdi
rfun:
    pushq %rbx              ; save %rbx
    movq %rdi, %rbx         ; use callee saved register %rbx
        to store x
    movl $0, %eax           ; set return value to 0
    testq %rdi, %rdi        ; if x == 0
    je .L2                  ; go to .L2 if x == 0
    shrq $2, %rdi           ; shift x right by 2 bits
    call rfun               ; call rfun with x >> 2
    addq %rbx, %rax         ; add x to return value
.L2:
    popq %rbx               ; restore %rbx
    ret
```

The `%rbx` register is used to store the value of `x` in the current function call.

3.37

Expression	**Type**	**Value**	**Assembly code**
`P[1]`	short	$M[x_p + 2]$	`movw 2(%rdx), %ax`
`P + 3 + i`	short*	$x_p + 6 + 2i$	`leaq 6(%rdx, %rcx, 2), %rax`
`P[i * 6 - 5]`	short	$M[x_p + 12i - 10]$	`movw -10(%rdx, %rcx, 12), %ax`
`P[2]`	short	$M[x_p + 4]$	`movw 4(%rdx), %ax`
`&P[i + 2]`	short*	$x_p + 2i + 4$	`leaq 4(%rdx, %rcx, 2), %rax`

3.38

```
; long sum_element(long i, long j)
; i in %rdi, j in %rsi

sum_element:
    leaq 0(,%rdi,8), %rdx          ; %rdx <- 8i
    subq %rdi, %rdx                ; %rdx <- 7i
    addq %rsi, %rdx                ; %rdx <- 7i + j
    leaq (%rsi, %rsi, 4), %rax     ; %rax <- 5j
    addq %rax, %rdi                ; %rdi <- 5j + i
    movq Q(,%rdi,8), %rax          ; %rax <- M[x_q + 8(5j
       + i)]
    addq P(,%rdx,8), %rax          ; %rax <- M[x_p + 8(7i
       + j)] + M[x_q + 8(5j + i)]
    ret
```

The index for `P` is $7i + j$ and the index for `Q` is $5j + i$. Therefore, `M` $= 6$ and `N` $= 8$.

3.41

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

Offsets:

- `p`: 8 bytes
- `s.x`: 2 bytes
- `s.y`: 2 bytes
- `next`: 8 bytes