Chapter 1

Homework Problems

Problem 1.1

A. $25 \div ((1000 \div 100) + (1500 \div 150)) = 1.25\times$

B. $1500 \div (25 \div 1.67 - 10) = 300(km/h)$

Problem 1.2

$(1 - 0.1) \div (\frac{1}{4} - 0.1) = 6\times$

Chapter 2

Practice Problems

Problem 2.1

A. `0010 0101 1011 1001 1101 0010`

B. `0xAE49`

C. `1010 1000 1011 0011 1101`

D. `0x322D96`

Problem 2.2

n	2^n(decimal)	2^n(hexadecimal)
5	32	`0x20`
23	8388608	`0x80000`
15	32768	`0x8000`
13	8192	`0x2000`
12	4096	`0x1000`
6	64	`0x40`
8	256	`0x100`

Problem 2.3

Decimal	Binary	Hexadecimal
0	`0000 0000`	`0x00`
158	`1001 1110`	`0x9E`
76	`0100 1100`	`0x7C`
145	`1001 0001`	`0x91`
174	`1010 1110`	`0xAE`
60	`0011 1100`	`0x3C`
241	`1111 0001`	`0xF1`
116	`0111 0101`	`0x75`
189	`1011 1101`	`0xBD`
245	`1111 0101`	`0xF5`

Problem 2.4

A. `0x605C+0x5=0x6061`

B. `0x605C-0x20=0x603C`

C. `0x605C+32=0x607C`

D. `0x60FA-0x605C=0x9e`

Problem 2.5

A. Little endian: 78 Big endian: 12

B. Little endian: 78 56 Big endian: 12 34

C. Little endian: 78 56 34 Big endian: 12 34 56

Problem 2.6

A. `0x0027C8F8: 0000 0000 0010 0111 1100 1000 1111 1000`
`0x4A1F23E0: 0100 1010 0001 1111 0010 0011 1110 0000`

Problem 2.7

```
6d 6e 6f 70 71 72
```

Problem 2.8

Operation	Result
a	01001110
b	11100001
~a	10110001
~b	00011110
a&b	01000000
a|b	11101111
a^b	10101111

Problem 2.9

A.

Color	Complement
White	Black
Blue	Yellow
Green	Magenta
Cyan	Red

B. Blue|Green = Cyan
Yellow&Cyan = Green
Red^ Magenta = Blue

Problem 2.10

Step	*x	*y
Initially	a	b
Step 1	a	a^b
Step 2	b	a^b
Step 3	b	a

Problem 2.11

A. first=last=k

B. The funtion implace_swap was given two identical arguments.

C. Change line 4 to "`first < last`".

Problem 2.12

A. `x&0xFF`

B. `x^~0xFF`

C. `x|0xFF`

Problem 2.13

```
bis(x, y)
```

```
bis(bic(x, y), bitc(y, x))
```

Problem 2.14

Expression	Value	Expression	Value
`a&b`	`0x44`	`a&&b`	1
`a|b`	`0x57`	`a||b`	1
`~a|~b`	`0xBB`	`!a||!b`	0
`a&!b`	0	`a&&~b`	1

Problem 2.15

```
bool equal(int x, int y) {
    return !(x ^ y);
}
```

Problem 2.16

a		a<<2		a>>3(Logical)		a>>3(Arithmetic)	
Hex	Binary	Hex	Binary	Hex	Binary	Hex	Binary
0xD4	1101 0100	0x50	0101 0000	0x1A	0001 1010	0xFA	1111 1010
0x64	0110 0100	0x90	1001 0000	0x0C	0000 1100	0x0C	0000 1100
0x72	0111 0010	0xC8	1100 1000	0x0E	0000 1110	0x0E	0000 1110
0x44	0100 0100	0x10	0001 0000	0x08	0000 1000	0x08	0000 1000

Problem 2.17

Hexadecimal	Binary	$B2U_4(\vec{x})$	$B2T_4(\vec{x})$
0xA	1010	$2^3 + 2^1 = 10$	$-2^3 + 2^1 = -6$
0x1	0001	$2^0 = 1$	$2^0 = 1$
0xB	1011	$2^3 + 2^1 + 2^0 = 11$	$-2^3 + 2^1 + 2^0 = -5$
0x2	0010	$2^1 = 2$	$2^1 = 2$
0x7	0111	$2^2 + 2^1 + 2^0 = 7$	$2^2 + 2^1 + 2^0 = 7$
0xC	1100	$2^3 + 2^2 = 12$	$-2^3 + 2^2 = -4$

Problem 2.18

A. `0x2e0=736`

B. `-0x58=-88`

C. `0x28=40`

D. `-0x30=-48`

E. `0x78=120`

F. `0x88=136`

G. `0x1f8=504`

H. `0xc0=192`

I. `-0x48=-72`

Problem 2.19

x	$T2U_4(x)$
-1	15
-5	11
-6	10
-4	12
1	1
8	8

Problem 2.20

pass

Problem 2.21

Expression	Type	Evaluation
`-2147483647-1 == 2147483648U`	Unsigned	1
`-2147483647-1 < 2147483647`	Signed	1
`-2147483647-1U < 2147483647`	Unsigned	0
`-2147483647-1 < -2147483647`	Signed	1
`-2147483647-1U < -2147483647`	Unsigned	1

Problem 2.22

A. $1100_2 = -2^3 + 2^2 = -4$

B. $11100_2 = -2^4 + 2^3 + 2^2 = -4$

C. $111100_2 = -2^5 + 2^4 + 2^3 + 2^2 = -4$

Problem 2.23

	w	fun1(w)	fun2(w)
A.	`0x00000076`	`0x00000076`	`0x00000076`
	`0x87654321`	`0x00000021`	`0x00000021`
	`0x000000C9`	`0x000000C9`	`0xFFFFFFC9`
	`0xEDCBA987`	`0x00000087`	`0xFFFFFF87`

B. fun1 return the zero extension of the least significant byte of w.
fun2 return the sign extension of the least significant byte of w.

Problem 2.24

Hex		Unsigned		Signed	
Original	Truncated	Original	Truncated	Original	Truncated
1	1	1	1	1	1
3	3	3	3	3	3
5	5	5	5	5	5
C	4	12	4	-4	-4
E	6	14	6	-2	-2

Problem 2.25

Reason: When length equals 0, length minus 1 equals $UMax_{32}$, so the expression $i \leq UMax_{32}$ holds for any unsigned i and hence the for loop would never stop.

Correction: Change the expression `i <= length - 1` to `i < length`.

Problem 2.26

A. When string s is shorter than string t.

B. The data type of strlen(s)-strlen(t) is unsigned, so it will be greater than 0 for any different strlen(s) and strlen(t). So when s is short than t, this function will return a wrong answer.

C. Change the return value to strlen(s) > strlen(t).

Problem 2.27

```
int uadd_ok(unsigned x, unsigned y) {
    return x + y < x || x + y < y;
}
```

Problem 2.28

x		$-_4^u x$	
Hex	Decimal	Decimal	Hex
1	1	15	F
4	4	11	B
7	7	9	9
A	10	6	6
E	14	2	2

Problem 2.29

x	y	$x+y$	$x +_5^t y$	Case
-12	-15	-27	5	1
10100	10001	100101	00101	1
-8	-8	-16	-16	2
11000	11000	110000	10000	2
-9	8	-1	-1	2
10111	01000	11111	11111	2
2	5	7	7	3
00010	00101	00111	00111	3
12	4	16	-16	4
01100	00100	10000	10000	4

Problem 2.30

```
int tadd_ok(int x, int y) {
    int z = x + y;
    return !((x > 0 && y > 0 && z <= 0) || (x < 0 && y < 0 && z >= 0));
}
```

Problem 2.31

Signed addition is associative and commutative, so (x+y)-x = y+(x-x)=y and hence whether or not there is overflow, this function will always return 1.

Problem 2.32

For any x and y = $TMin$, this function will give incorrect results.

Problem 2.33

x		$-_4^t x$	
Hex	Decimal	Decimal	Hex
2	2	-2	E
3	3	-3	D
9	9	-9	7
B	11	-11	5
C	12	-12	4

The bit patterns generated by two's complement and unsigned negation are identical.

Problem 2.34

Mode	x		y		$x \cdot y$		Truncated $x \cdot y$	
	Hex	Binary	Hex	Binary	Hex	Binary	Hex	Binary
Unsigned	4	100	5	101	20	010100	4	100
Two's complement	-4	100	-3	101	12	001100	-4	100
Unsigned	2	010	7	111	14	001110	6	110
Two's complement	2	010	-1	111	-2	111110	-2	110
Unsigned	6	110	6	110	36	100100	4	110
Two's complement	-2	110	-2	110	4	000100	-4	100

Problem 2.35

pass

Problem 2.36

```
int tmult_ok(int x, int y) {
    int64_t z1 = (int64_t)x * y;
    int z2 = x * y;
    return (int64_t)z2 == z1;
}
```

Problem 2.37

A. No improvement at all. Although variable asize is 64-bit and its value is accurate, when it is passed to malloc as a parameter with type size_t, it will still be truncated to 32 bit as well.

B. Since the parameter of malloc is size_t with 32 bit, it's impossible to allocate more than 2^{32} bytes. What we can do is to determine whether there is overflow before malloc. If there is, do not call malloc and return NULL.

Problem 2.38

A power of 2($2^k, for\ any\ k > 0$) or A power of 2 plus 1($2^k + 1, for\ any\ k > 0$).

Problem 2.39

$-(x << m)$

Problem 2.40

K	Shifts	Add/Subs	Expression
7	1	1	$(x << 3) - x$
30	4	3	$(x << 4) + (x << 3) + (x << 2) + (x << 1)$
28	2	1	$(x << 5) - (x << 2)$
55	2	2	$(x << 6) - (x << 3) - x$

Problem 2.41

When m = n and m + 1 = n, choose form A, otherwise form B.

Problem 2.42

```
int div16(int x) {
    return (x + ((x >> 31) & 0xF)) >> 4;
}
```

Problem 2.43

M = 31, N = 8.

Problem 2.44

A. False for x = -2147483648.

B. True. If (x & 7) != 7 is false, namely (x & 7) == 7, the least 3 significant bits must be [111]. So the most 3 significant bits of x « 29 will be 111 and hence x « 29 < 0.

C. False for x = 50000 where the value of x * x equals 2500000000 > 2147483647, causes positive overflow and yields a negative value.

D. True. For any x >= 0, -x must be smaller than or equal to 0. Negation of a nonnegative value will never cause overflow.

E. False. This is true for any value of type `int` except -2147483648. Negation of -2147483648 is still -2147483648 and is smaller than 0.

F. True. Two's complement addition has the same bit-level represetation as unsigned.

G. True

Problem 2.45

Fractional value	Binary represetation	Decimal represetation
$\frac{1}{8}$	0.001	0.125
$\frac{3}{4}$	0.11	0.75
$\frac{5}{16}$	0.0101	0.3125
$2\frac{11}{16}$	10.1011	2.6875
$1\frac{1}{8}$	1.001	1.125
$5\frac{7}{8}$	101.111	5.875
$3\frac{3}{16}$	11.0011	3.1875

Problem 2.46

A. $0.00000000000000000000000[0011]\cdots_2$

B. $1/(2^{20} \times 10)$

C. $(3600 * 100 * 10)/(2^{20} \times 10) = 0.343s$

D. $2000m/s \times 0.343s = 687m$

Problem 2.47

Bits	e	E	2^E	f	M	$2^E \times M$	V	Decimal
0 00 00	0	0	$\frac{1}{1}$	$\frac{0}{4}$	$\frac{0}{4}$	$\frac{0}{4}$	0	0.0
0 00 01	0	0	$\frac{1}{1}$	$\frac{1}{4}$	$\frac{1}{4}$	$\frac{1}{4}$	$\frac{1}{4}$	0.25
0 00 10	0	0	$\frac{1}{1}$	$\frac{2}{4}$	$\frac{2}{4}$	$\frac{2}{4}$	$\frac{1}{2}$	0.5
0 00 11	0	0	$\frac{1}{1}$	$\frac{1}{4}$	$\frac{3}{4}$	$\frac{3}{4}$	$\frac{3}{4}$	0.75
0 01 00	1	0	$\frac{1}{1}$	$\frac{0}{4}$	$\frac{4}{4}$	$\frac{4}{4}$	1	1.0
0 01 01	1	0	$\frac{1}{1}$	$\frac{1}{4}$	$\frac{5}{4}$	$\frac{5}{4}$	$\frac{5}{4}$	1.25
0 01 10	1	0	$\frac{1}{1}$	$\frac{2}{4}$	$\frac{6}{4}$	$\frac{6}{4}$	$\frac{3}{2}$	1.5
0 01 11	1	0	$\frac{1}{1}$	$\frac{3}{4}$	$\frac{7}{4}$	$\frac{7}{4}$	$\frac{7}{4}$	1.75
0 10 00	2	1	2	$\frac{0}{4}$	$\frac{4}{4}$	$\frac{8}{4}$	2	2.0
0 10 01	2	1	2	$\frac{1}{4}$	$\frac{5}{4}$	$\frac{10}{4}$	$\frac{5}{2}$	2.5
0 10 10	2	1	2	$\frac{2}{4}$	$\frac{6}{4}$	$\frac{12}{4}$	3	3.0
0 10 11	2	1	2	$\frac{3}{4}$	$\frac{7}{4}$	$\frac{14}{4}$	$\frac{7}{2}$	3.5
0 11 00							∞	
0 11 01							NaN	
0 11 10							NaN	
0 11 11							NaN	

Problem 2.48

Step 1. From hexadecimal represetation 0x00359141, get the binary represetation: [0000 0000 0011 0101 1001 0001 0100 0001]

Step 2. Create normalized represetation by shift 21 positions to the right of binary point, giving 1.1010 1100 1000 1010 0000 1 $\times 2^{21}$

Step 3. Remove leading 1 of significand and add 2 zeros to the right, giving the fraction field [1010 1100 1000 1010 0000 100]

Step 4. Since $E = 21$, exponent field $exp = E + Bias = 21 + 127 = 148$, giving the exponent field [1001 0100]

Step 5. So the floating-point represetation of integer 3,510,593 is [0100 1010 0101 0110 0100 0101 0000 0100], of which the hexadecimal represetation is `0x4A564504`

Problem 2.49

A. $2^{n+1} + 1$

B. 16777217

Problem 2.50

Before		After	
Binary	Numeric value	Binary	Numeric value
10.111	2.875	11.0	3
11.010	3.25	11.0	3
11.000	3	11.0	3
10.110	2.75	11.0	3

Problem 2.51

A. 0.00011001100110011001101

B. 0.1×2^{-22}

C. $(3600 * 100 * 10/(10 * 2^{22})) = 0.0858s$

D. $0.0858s \times 2000m/s = 171m$

Problem 2.52

Format A		Format B	
Bits	Value	Bits	Value
011 0000	1	0111 000	1
101 1110	7.5	1001 111	7.5
010 1001	0.78125	0110 100	0.75
110 1111	15.5	1011 000	16
000 0001	$\frac{1}{2^6}$	0001 000	$\frac{1}{2^6}$

Problem 2.53

```
#define POS_INFINITY 1e309
#define NEG_INFINITY -1e309
#define NEG_ZERO -1e324
```

Problem 2.54

A. True, since casting an int variable to double will cause neither overflow or precision lost.

B. False. When casting int to float, the numeric value might be rounded, such as `0x7fffffff`.

C. False. Casting double to float might overflow and produce INFINITY, such as float 1e40.

D. True, just like (A.)

E. True. Negation only changes the sign bit, so there is neither overflow or rounding.

F. True. The lefthand and righthand are both equal to 0.5.

G. True. The sign bit of multiplication result is 0 and hence the result is nonnegative if and only if sign bits of two operands are identical.

H. False. f+d may overflow and produce INFINITY, such as f = 10.0 and d = {the max value of `double`}.

Homework Problems

Problem 2.55

- Little endian on macOS x86-64.
- Little endian on Ubuntu 16.04 x86-64.

Problem 2.56

pass

Problem 2.57

```
void show_short(short sx) {
    show_bytes((byte_pointer) &x, sizeof(short));
}

void show_long(long lx) {
    show_bytes((byte_pointer) &x, sizeof(long));
}

void show_double(double d) {
    show_bytes((byte_pointer) &x, sizeof(double));
}
```

Problem 2.58

```
int is_little_endian() {
    int32_t x = 0x12345678;
    return *((int8_t *)&x) == 0x78;
}
```

Problem 2.59

```
   (x & 0xff) | (y & ~0xff)
```

Problem 2.60

```
unsigned replace_byte(unsigned x, int i, unsigned char b) {
    return (x & ~(0xff << (8*i))) | (b << (8*i));
}
```

Chapter 3

Practice Problems

Problem 3.1

Operand	Value
%rax	0x100
0x104	0xAB
$0x108	0x108
(%rax)	0xFF
4(%rax)	0xAB
9(%rax, %rdx)	0x11
260(%rcx, %rdx)	0x13
0xFC(, %rcx, 4)	0xFF
(%rax, %rdx, 4)	0x11

Problem 3.2

```
movl        %eax        (%rsp)
movw       (%rax)         %dx
movb        0xFF          %bl
movb  (%rsp, %rdx, 4)     %dl
movq       (%rdx)        %rax
movw         %dx         (%rax)
```

Problem 3.3

1. %ebx is a 32-bit register, but memory address should be a 64-bit integer.
2. %rax is a 64-bit register, thus it cannot be source of `movl`.
3. Oprands cannot both be memory references.
4. There is no register named %sl.
5. Immediate cannot be destination.
6. The destination oprand of the question is not consistent with which of the solution.
7. %si is a 16-bit register, thus it cannot be source of `movb`.

Problem 3.4

```
    movq (%rdi), %rax
    movq %rax, (%rsi)
```

Problem 3.5

```
void decode(long *xp, long *yp, long *zp) {
    int t1 = *xp, t2 = *yp, t3 = *zp;
    *yp = t1;
    *zp = t2;
    *xp = t3;
    return t3;
}
```

Problem 3.6

Instruction	Result
leaq 9(%rdx), %rax	q+9
leaq (%rdx, %rbx), %rax	p+q
leaq (%rdx, %rbx, 3), %rax	q+3p
leaq 2(%rbx, %rbx, 7), %rax	8p+2
leaq 0xE(, %rdx, 3), %rax	3q+14
leaq 6(%rbx, %rdx, 7), %rax	p+7q+6

Problem 3.7

```
10*y+z+x*y
```

Problem 3.8

Instruction	Destination	Value
addq %rcx, (%rax)	0x100	0x100
subq %rdx, 8(%rax)	0x108	0xA9
imulq $16, (%rax, %rdx, 8)	0x118	0x110
incq 16(%rax)	0x110	0x14
decq %rcx	%rcx	0x0
subq %rdx, %rax	%rax	0xFD

Problem 3.9

```
salq $4, %rax
sarq %cl, %rax
```

Problem 3.10

```
short p1 = y|z;
short p2 = p1 >> 9;
short p3 = ~p2;
short p4 = y - p3;
```

Problem 3.11

A. Clear register %rcx.

B. `movq $0, %rcx`.

C.

Problem 3.12

Replace `idivq` to `divq` in line 5.

Problem 3.13

Conditions	data_t	COMP
A.	int	<
B.	short	>=
C.	unsigned char	<=
D.	long	!=
	unsigned long	!=

Problem 3.14

Conditions	data_t	TEST
A.	long	>=
B.	short	==
	unsigned short	==
C.	unsigned char	>
D.	int	<=

Problem 3.15

A. 4003fe

B. 400425

C. 400547 400545

D. 400560

Problem 3.16

A.
```
void goto_cond(short a, short *p) {
      if (a == 0) {
          goto ret;
      }
      else {
          if (a <= *p) {
              goto ret;
          }
          *p = a;
      }
  ret:
      return;
  }
```

B. Because the compile split the if statement into to comparisons: (1) if a equals 0. (2) if *p < a.

Problem 3.17

A.
```
long gotodiff_se_alt(long x, long y)
{
    long result;
    if (x < y)
        goto x_lt_y;
    ge_cnt++;
    result = x - y;
    return result;
x_lt_y:
    lt_cnt++;
    result = y - x;
```

```
        return result;
    }
```

B. The original rule is better when there is no else statement.

Problem 3.18

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

Problem 3.19

A. Approximately 40 cycles.

B. 65 cycles.

Problem 3.20

A. / (integral divide).

B. leaq: assign x+15 to %rbx

testq: test x

cmovns: when x is nonnegative, move x to %rbx

sarq: now if x is negative, %rbx = x+15, otherwise x itself, so arithmetic right shift 4 bits will compute correctly both negative and nonnegative integer division.

Problem 3.21

```
short test(short x, short y) {
    short val = y + 12;
    if (x < 0) {
        if (x >= y) {
            val = x | y;
        } else {
            val = x * y;
        }
    } else if (y > 10) {
        val = x / y;
    }
    return val;
}
```

Problem 3.22

A. Computation overflows.

D. No.

Problem 3.23

A. %rbx for x, %rcx for y, %rdx for n.

B. By using instruction leaq on line 5.

Problem 3.24

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

Problem 3.25

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

Problem 3.26

A. Jump-to-middle method.

B.
```
short test_one(unsigned short x) {
        short val = 1;
        while (x) {
            val ^= x;
            x >>= 1
        }
        return val & 0;
    }
```

Problem 3.27

```
long fibonacci_gd_goto(long n) {
    long i = 2;
    long next, first = 0, second = 1;
    if (n <= 1)
        goto done;
loop:
    next = first + second;
    first = second;
    second = next;
    i++;
    if (i <= n)
        goto loop;
done:
    return n;
}
```

Problem 3.28

A.
```
short test_two(unsigned short x) {
        short val = 0;
        short i;
        for (i = 1, val = 65; i; i++) {
            val += val;
            val |= (x & 1);
            x >>= 1;
        }
        return val;
    }
```

Problem 3.29

A. The update-expr i++ will be skip, too. In this code, variable i will always equal to 1 and will never be updated.

B.
```
long sum = 0;
long i;
for (i = 0; i < 10;) {
    if (i & 1)
        goto update;
    sum += i;
update:
    i++;
}
```

Problem 3.30

A. From -2 to 6.

B. When x is -1, 1, 3, 6.

Problem 3.31

```
void switcher(long a, long b, long c, long *dest) {
    long val;
    switch(a) {
        case 5:
            c = b ^ 15;
        case 0:
            val = c+112;
            break;
        case 2:
        case 7:
            val = (c + b) << 2;
            break;
        case 4:
            val = a;
            break;
        default:
            val = b;
    }
    *dest = val;
}
```

Problem 3.32

Instruction			State values(at beginning)					
Label	PC	Instruction	%rdi	%rsi	%rax	%rsp	*%rsp	Description
M1	0x400560	callq	10	-	-	0x7fffffffe820	-	Call first(10)
F1	0x400548	lea	10	-	-	0x7fffffffe818	0x400565	Entry of first
F2	0x40054c	sub	10	11	-	0x7fffffffe818	0x400565	Assign 11 to %rsi
F3	0x400550	callq	9	11	-	0x7fffffffe818	0x400565	Call last(9, 11)
L1	0x400540	mov	9	11	-	0x7fffffffe810	0x400555	Entry of last
L2	0x400543	imul	9	11	9	0x7fffffffe810	0x400555	Assign 9 to %rax
L3	0x400547	retq	9	11	99	0x7fffffffe810	0x400555	Return 99 from last
F4	0x400555	retq	9	11	99	0x7fffffffe818	0x400565	Resume first and return 99 from first
M2	0x400565	mov	9	11	99	0x7fffffffe820	-	Resume main

Problem 3.33

```
int a, char b, long *u, char *v
```

or

```
int b, char a, long *v, char *u
```

Problem 3.34

A. a0, a1, a2, a3, a4, a5 get stored in callee-saved registers.

B. a6. a7.

C. There is only 6 callee-saved registers, but procedure P needs to store 8 local variables.

Problem 3.35

A. x.

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

Problem 3.36

Array	Element size	Total size	Start address	Element i
P	4	20	x_P	$x_P + 4 * i$
Q	2	4	x_Q	$x_Q + 2 * i$
R	8	72	x_R	$x_R + 8 * i$
S	8	80	x_S	$x_S + 8 * i$
T	8	16	x_T	$x_T + 8 * i$

Problem 3.37

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

Problem 3.38

M = 5, N = 7

Problem 3.39

skip

Problem 3.40

```
void fix_set_diag_opt(fix_matrix A, int val) {
    int *ptr = &A[0][0];
    int *end = &A[N][N];
    while (ptr != end) {
        *ptr = val;
        ptr += 4*(N+1);
    }
}
```

Problem 3.41

A. p: 0
s.x: 8
s.y: 10
next: 12

B. 20

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

Problem 3.42

A.
```
short test(struct ACE *ptr) {
    short result = 1;
    while (ptr != NULL) {
        result *= ptr->val;
        ptr = ptr->p;
    }
    return result;
}
```

B. This structure implements link list and procedure test computes the multiplication of every element in this link list.

Problem 3.43

expr	*type*	Code
up->t1.u	long	movq (%rdi), %rax movq %rax, (%rsi)
up->t1.v	short	movw 8(%rdi), %ax movw %ax, (%rsi)
&up->t1.w	char *	addq $10, %rdi movq %rdi, (%rsi)
up->t2.a	int *	movq %rdi, (%rsi)

up->t2.a[up->t1.u]	int	movq (%rdi), %rax
		movl (%rdi, %rax, 4), %eax
		movl %eax, (%rsi)

*up->t2.p	char	movb 8(%rdi), %rax
		movb (%rax), %al
		movb %al, (%rsi)

Problem 3.44

struct	offsets	alignment requirement
P1	0, 4, 8, 16	multiples of 8
P2	0, 8, 16, 24	multiples of 8
P3	0, 16	multiples of 8
P4	0, 16	multiples of 8
P5	0, 64	multiples of 8

Problem 3.45

A. 0, 8, 12, 14, 16, 24, 32, 40

B. 48

C. Optimized struct declaration:

```
struct {
    int *a;      // 0
    long e;      // 8
    double f;    // 16
    char *h;     // 24
    int g;       // 32
    float b;     // 36
    short d;     // 40
    char c;      // 42
} rec;
```

total size: 43 bytes.

A.

00	00	00	00	00	40	00	76	Return address
01	23	45	67	89	AB	CD	EF	Saved %rbx
Unknown								
Unknown								%rsp pointed

B.

00	00	00	00	00	40	00	34	Corrupted return address
33	32	31	30	39	38	37	36	Corrupted saved %rbx
35	34	33	32	31	30	39	38	written by gets
37	36	35	34	33	32	31	30	%rsp pointed

C. 0x400734

D. %rbx

E. malloc should have strlen(buf+1) as its argument. And the code for get_line doesn't check if the returned value is NULL.

Problem 3.47

A. 2^{13}

B. $2^6 = 64$