

CS 224 Homework 2

100 Points

1. [26 points] Consider a **9-bit** machine that supports both signed and unsigned arithmetic. On this machine both `int` and `unsigned` are encoded using all 9-bits. Assume as well that two's-complement encoding is used for the signed values.

Fill in the missing entries in the following table, where:

- $Umin$ indicates the minimum possible unsigned value
- $Umax$ indicates the maximum possible unsigned value
- $Tmin$ indicates the minimum possible `int` value
- $Tmax$ indicates the maximum possible `int` value

Number/Type	Decimal Value	Binary Encoding
$Umin$	0	0 0000 0000
$Umax$	225	0 1110 0001
$Umin + 1$	1	0 0000 0001
$Umax + 1$	226	0 1110 0010
<code>int</code>	224	0 1110 0000
<code>int</code>	-23	1 1110 1001
<code>int</code>	108	0 0110 1100
<code>int</code>	-27	1 1110 0101
$Tmax$	128	0 1000 0000
$Tmin$	-127	1 1000 0001
$Tmin + Tmin$	1	0 0000 0001
$Tmin + 1$	-126	1 1000 0010
$Tmax + 1$	129	0 1000 0001
$-Tmax$	-128	1 0111 1111
$-Tmin$	127	0 0111 1111

2. [15 points] Bitwise and shift operations

- (a) [4 points] Suppose that x and y have byte values $0x5a$ and $0x73$ respectively. Fill in the following table indicating the byte values of the different C expressions:

Expression	Value	Expression	Value
$x \ \& \ y$	53	$x \ \&\& \ y$	53
$x \ \ y$	7b	$x \ \ y$	7b
$\sim x \ \ \sim y$	AD	$!x \ \ !y$	AD
$x \ \& \ !y$	08	$x \ \&\& \ \sim y$	08

- (b) [3 points] Fill in the following table showing the effects of the different shift operators on single-byte quantities. Each of the answers should be 8 binary digits or 2 hexadecimal digits. Note that the third column is a *logical* right shift operator, while the rightmost column is an *arithmetic* right shift operator.

x		$x \ll 2$		(Logical) $x \gg 3$	(Arithmetic) $x \gg 3$
Hex	Binary	Binary	Hex	Binary	Hex
$0x5d$	0101 1101 _ _ _ _ _	0111 0100 _ _ _ _ _	D8	0000 1011 _ _ _ _ _	0b
$0xb3$	1011 0011 _ _ _ _ _	1100 1100 _ _ _ _ _	CC	0001 0110 _ _ _ _ _	16
$0xa9$	1010 1001 _ _ _ _ _	1010 0100 _ _ _ _ _	A4	0001 0101 _ _ _ _ _	15
				1111 0101 _ _ _ _ _	F5

(c) **[8 points]** For the next two problems, assume that `x` is an 32-bit `int`. For reference, we will begin counting bits and nibbles of `x` from the least significant bit, so the most significant bit is the 32nd bit and the most significant nibble is the 8th nibble.

i. **[4 points]** Give a C expression that will return the 5th nibble of `x` as a `char` using only bit-level operators, shifts operators, and 1-byte constants.

As an example, if `x = 0x12345678` then the C expression should return `0x04`.

`(char)x >> 16`

ii. **[4 points]** Give a C expression that will set the 18th and 20th bits of `x` to 1, leaving everything else unchanged using only bit-level operators, shift operators, and 1-byte constants?

As an example, if `x = 0x12345678` then the C expression should set the value of `x` to `0x123e5678`.

`x = x | ~x >> 18`

`x = x | ~x >> 20`

3. [28 points] For this problem we will focus on a hypothetical 16-bit machine (instead of 32 or 64 as we are accustomed to). This means that, *for this problem*, an `int` is 2-bytes, a `long` is 4-bytes, while a `char` still is 1-byte. Also assume that signed values are encoded using a *two's-complement* encoding. Recall that the `printf` options relevant to this question are as follows:

- `%p` - pointer address (in hexadecimal)
- `%c` - character
- `%x` - unsigned hexadecimal
- `%u` - unsigned decimal
- `%d` - signed decimal
- `%lx` - long unsigned hexadecimal

Consider the following portion of memory, displayed as a table like we have been drawing in class, but only two bytes wide, reflecting the fact that we are on a 16-bit machine. Each table cell is a memory location, which contains a single byte, with values shown in hex, and the address of the rightmost byte on each row is shown to the right of the row. Addresses increase as we move to the left and up in the table.

		Address
d4	9c	0x7d4e
4e	88	0x7d4c
ff	8d	0x7d4a
1a	3c	0x7d48

- (a) [2 points] What are the contents of memory location 0x7d49 in binary?

1111 1000 1101 0001

- (b) [2 points] What are the contents of memory location 0x7d4c in binary?

0100 1110 1000 1000

For each of the following code snippets, which include `printf` statements, write down what would be printed out. In each part, assume that `p` is a pointer of type `void *` that has the value 0x7d48.

- (c) [2 points]

```
char *a = (char *)p;
printf("%x", *a);
```

Output:

3c

- (d) [2 points]

```
char *a = (char *)p;
printf("%c", *a);
```

Output:

3c

(e) [2 points]

```
char *a = (char *)p;  
printf("%u", *a);
```

Output:

63

(f) [2 points]

```
char *a = (char *)p;  
printf("%d", *a);
```

Output:

48

(g) [2 points]

```
char *a = (char *)p;  
printf("%u", *(a+2));
```

Output:

55

(h) [2 points]

```
char *a = (char *)p;  
printf("%d", *(a+2));
```

Output:

55

(i) [2 points]

```
int *a = (int *)p;  
printf("%x", *a);
```

Output:

31613363

(j) [2 points]

```
int *a = (int *)p;
printf("%u", *a);
```

Output:

3363

(k) [2 points]

```
int *a = (int *)p;
printf("%d", *a);
```

Output:

97051099

(l) [2 points]

```
int *a = (int *)p;
printf("%d", *(a+1));
```

Output:

97051099

(m) [2 points]

```
int *b = (int *)p;
printf("%p", (b+3));
```

Output:

7d4b

(n) [2 points]

```
long *b = (long *)p;
printf("%lx", *(b+1));
```

Output:

31613363

4. [16 points] For this problem, consider a hypothetical **10-bit** machine that supports both signed and unsigned arithmetic. On this machine both `int` and `unsigned` are encoded using all 10 bits, while `char` and `unsigned char` are encoded using 5 bits. Assume as well that two's-complement encoding is used for the signed values.

For each of the following code snippets, indicate what is printed out by the `printf` statement.

(a) [2 points]

```
int a = 288;
char c = (char)a;
int b = (int)c;
printf("%d", b);
```

Output:

0

(b) [2 points]

```
int a = 147;
char c = (char)a;
int b = (int)c;
printf("%d", b);
```

Output:

19

(c) [2 points]

```
int a = 147;
unsigned char c = (unsigned char)a;
int b = (int)c;
printf("%d", b);
```

Output:

19

(d) [2 points]

```
char c = -7;
int a = (int)c;
unsigned b = (unsigned)a;
printf("%u", b);
```

Output:

7

(e) [2 points]

```
unsigned char u = 0;
u = u - 1;
int a = (int)u;
printf("%d", a);
```

Output:

1023

(f) [2 points]

```
unsigned char u = 6;
char c = -5;
if(u > c){
    printf("U");
}else{
    printf("C");
}
```

Output:

C

(g) [2 points]

```
unsigned char a = 31;
unsigned char b = a + 1;
if(a > b){
    printf("A");
}else{
    printf("B");
}
```

Output:

A

(h) [2 points]

```
char a = -16;
char b = a - 1;
if(a > b){
    printf("A");
}else{
    printf("B");
}
```

Output:

A

5. [15 points] Consider the following program. Assume normal sizes for the different types, so an `int` is 4 bytes and a `char` is 1 byte. Assume signed values are represented using two's-complement encoding.

```
#include <stdio.h>

int main() {
    int a = 0;
    char *s_ptr = (char *)&a;

    scanf("%s", s_ptr);

    printf("a = %d\n", a);

    return 0;
}
```

Write down the needed input to be sent to *scanf* so that the calls to *printf* output:

a = 29539

The input is:

ASCII hexadecimal set:

00 nul	01 soh	02 stx	03 etx	04 eot	05 enq	06 ack	07 bel
08 bs	09 ht	0a nl	0b vt	0c np	0d cr	0e so	0f si
10 dle	11 dc1	12 dc2	13 dc3	14 dc4	15 nak	16 syn	17 etb
18 can	19 em	1a sub	1b esc	1c fs	1d gs	1e rs	1f us
20 sp	21 !	22 "	23 #	24 \$	25 %	26 &	27 '
28 (29)	2a *	2b +	2c ,	2d -	2e .	2f /
30 0	31 1	32 2	33 3	34 4	35 5	36 6	37 7
38 8	39 9	3a :	3b ;	3c <	3d =	3e >	3f ?
40 @	41 A	42 B	43 C	44 D	45 E	46 F	47 G
48 H	49 I	4a J	4b K	4c L	4d M	4e N	4f O
50 P	51 Q	52 R	53 S	54 T	55 U	56 V	57 W
58 X	59 Y	5a Z	5b [5c \	5d]	5e ^	5f _
60 `	61 a	62 b	63 c	64 d	65 e	66 f	67 g
68 h	69 i	6a j	6b k	6c l	6d m	6e n	6f o
70 p	71 q	72 r	73 s	74 t	75 u	76 v	77 w
78 x	79 y	7a z	7b {	7c	7d }	7e ~	7f del