Mock Final Exam

Instructions:

• This is a mock midterm exam.

• This exam is OPEN BOOK. You may use any books or notes you like. However, computer usage is not permitted.
Problem 1. (xxx points):
Assume we are running code on a 6-bit machine using two’s complement arithmetic for signed integers. A “short” integer is encoded using 3 bits. Fill in the empty boxes in the table below. The following definitions are used in the table:

```java
short sy = -3;
int y = sy;
int x = -17;
unsigned ux = x;
```

Note: You need not fill in entries marked with “–”.

<table>
<thead>
<tr>
<th>Expression</th>
<th>Decimal Representation</th>
<th>Binary Representation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Zero</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>–</td>
<td>−6</td>
<td></td>
</tr>
<tr>
<td>–</td>
<td>−6</td>
<td>01 0010</td>
</tr>
<tr>
<td>ux</td>
<td></td>
<td></td>
</tr>
<tr>
<td>y</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Tmax</td>
<td></td>
<td></td>
</tr>
<tr>
<td>−TMin</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Problem 2. (xxx points):
Consider the source code below, where M and N are constants declared with #define.

```c
int mat1[M][N];
int mat2[N][M];

int sum_element(int i, int j)
{
    return mat1[i][j] + mat2[i][j];
}
```

A. Suppose the above code generates the following assembly code:

```
sum_element:
    movslq %esi, %rsi
    movslq %edi, %rdi
    leaq (%rsi,%rdi,8), %rdx
    leaq (%rdi,%rdi,4), %rax
    leaq (%rdi,%rax,2), %rax
    addq %rax, %rsi
    movl mat2(,%rsi,4), %eax
    addl mat1(,%rdx,4), %eax
    ret
```

What are the values of M and N?

\[ M = \]

\[ N = \]
Problem 3. (3 points):
Consider the following C functions and assembly code:

```c
int fun1(int a)
{
    return a * 31;
}

int fun2(int a)
{
    return a * 34;
}

int fun3(int a)
{
    return a * 18;
}
```

Which of the functions compiled into the assembly code shown?
Problem 4. (xxx points):

This next problem will test your understanding of stack frames. It is based on the following C function:

typedef struct {
    int a[2];
    double d;
} struct_t;

double fun(int i) {
    volatile struct_t s;
    s.d = 3.14;
    s.a[i] = 1073741824;
    return s.d;
}

This yields the following machine code:

fun:
    movq $-1, -16(%rsp)
    movslq %edi, %rdi
    movl $1, -24(%rsp,%rdi,4)
    movq -16(%rsp), %rax
    ret
A. Is the variable $s$ stored on the stack? If so, at what byte offset (relative to `%rsp`) is it stored? At what byte offset is $s.a$ stored and what byte offset is $s.d$ stored?

B. What is the output of `fun` if its argument is 1, 2?

C. What (if anything) is stored at `(%rsp)`? If something is stored there, why is it necessary to store it?

D. What happens if `fun` is invoked with argument $i=6$? Please explain.
Problem 5. (xxx points):
Ben Bitdiddle is working on a machine that only supports 8-bit integer computation. Since his computation requires using bigger integers, Ben decides to emulate 16-bit integer arithmetic in software.

Ben’s code skeleton is shown below. Ben uses a struct of two 8-bit integers to represent a 16-bit integer in 2’s complement representation. The high field of BigInt is a signed 8-bit integer which contains the high-order 8 bits in the 2’s complement representation including the 2’s complement sign bit. The low field of BigInt is an unsigned 8-bit integer which contains the low-order 8 bits in the 2’s complement representation. For example, suppose BigInt x represents decimal 260 (0x0104), then x.high = 0x01 and x.low = 0x04. As another example, suppose x represents decimal -1 (0xffff), then x.high = 0xff and x.low = 0xff.

typedef struct BigInt {
    char high;       /* high-order 8 bits including 2’s complement sign bit */
    unsigned char low; /* low-order 8 bits of 2’s complement */
} BigInt

BigInt add(BigInt a, BigInt b) {
    BigInt c;
    /* compute c = a + b */
    return c;
}

BigInt sub(BigInt a, BigInt b) {
    BigInt c;
    /* compute c = a - b */
    return c;
}

(a) (2 pts) Given BigInt x such that x.high = 0x80 and x.low = 0x00, what is x’s value in decimal?

(b) (10 pts) Please complete add and sub functions for Ben Bitdiddle.

(c) (8 pts) Alyssa P. Hacker has convinced Ben to test the correctness of his code on x86 which does support 16-bit integers. Alyssa wrote the following test code skeleton for Ben. Please help Ben complete the two functions BigInt_to_short and short_to_BigInt.
short
BigInt_to_short(BigInt a)
{
    short x;
    /* convert BigInt a to a short (16-bit) integer*/

    return x;
}

BigInt
short_to_BigInt(short x)
{
    BigInt a;
    /* convert short integer x to BigInt representation*/

    return a;
}

void test(short x, short y) {
    short z;
    BigInt a,b,c;

    a = short_to_BigInt(x);
    b = short_to_BigInt(y);

    c = add(a,b);
    z = BigInt_to_short(c);
    assert(z == (short)(x+y));

    c = sub(a,b);
    z = BigInt_to_short(c);
    assert(z == (short)(x-y));
}

void main() {
    int i;
    for (i = 0; i <10000000; i++) {
        test(random(), random());
    }
}
Problem 6. (Multiple Choice points):

Answer the following multiple-choice questions. Circle all answers that apply. Each problem is worth 4 points. Each missing or wrong answer costs -2 point.

A. Which of the following C statement calculates the remainder of a divided by 32? Variable a is of int type.
   1. a % 32;
   2. a / 32;
   3. a >> 5;
   4. a << 5;
   5. a | ~0x1f;
   6. a & 0x1f;
   7. None of the above

B. Which of the following creates a 32-bit integer whose left-most i bits are 1 and the rest of the bits are zero?
   1. 1 << i
   2. (1<<i) - 1
   3. ((1<<i)-1) << (32 - i)
   4. 1<<(32-i)-1
   5. ~((1<<(32-i))-1)
   6. 0xf8000000
   7. None of the above

C. What is the value of variable b after executing the following code snippet?

```c
float a = -2.05;
int b;
b = *(int *)&a;
```
   1. -2
   2. -1
   3. 1
   4. 2
   5. 2.05
   6. None of the above
D. What is the value of \(p\) and \(q\) after executing the following code snippet?

```c
int *p;
char *q;
p = (int *)1;
p++;
q = (char *) (p+2)
```

1. \(p = 2, q = 3\)
2. \(p = 5, q = 13\)
3. \(p = 5, q = 7\)
4. \(p = 2, q = 7\)
5. \(p = 2, q = 13\)
6. None of the above

E. What is the printf output in the following code snippet?

```c
void swap(int a, int b) {
    a = b;
    b = a;
}
void main() {
    int a = 1, b = 2;
    swap(a, b);
    printf("a = %d, b = %d\n", a, b);
}
```

1. \(a = 1, b = 2\)
2. \(a = 1, b = 1\)
3. \(a = 2, b = 1\)
4. \(a = 2, b = 2\)
5. None of the above