Last Name: Daskalov

First Name: Daniel

UID: [redacted]

Computer Science 33, Winter 2012

Midterm 1

1 B

February 9, 2012

Instructions:

- Make sure that your exam is not missing any sheets, then write your full name on the front.
- Write your answers in the space provided below the problem. If you make a mess, clearly indicate your final answer.
- The exam has a maximum score of 100 points.
- The problems are of varying difficulty. The point value of each problem is indicated. Pile up the easy points quickly and then come back to the harder problems.
- This exam is closed book and closed notes.
- You have 1 hour and 50 minutes to complete the test. Good luck!

1 (9):	9
2 (12):	12
3 (15):	~~8~~ 8
4 (15):	15
5 (6):	6
6 (12):	~~8~~ 12
7 (10):	10
8 (6):	6
9 (15):	8
TOTAL (100):	~~81~~ 82 (86)

Problem 1. (9 points):

Answer the following questions about word size and endianness.

I. Identify each item below that has a size of one word in IA32. Note that word as used here refers to its meaning with reference to the IA32 architecture, not as the term is used within x86 assembly code. Use the blank below to indicate your answer (zero letters if none of the answers are correct, a single letter if exactly one answer is correct, or multiple letters if multiple answers are correct)

A. The encoding of a single IA32 assembly language instruction

B. A general purpose register

C. The minimal bit-level encoding of an ASCII character

D. A memory address

BD

II. Using hexidecimal, indicate the byte-wise storage of the number 1020 as a four byte integer in memory below, where the integer is stored starting at byte address 0x01A0 and ending at address 0x01A3:

Memory Address	0x01A0	0x01A1	0x01A2	0x01A3
Big-Endian storage	00	00	03	FC
Little-Endian storage	FC	03	00	00

12/12

Problem 2. (12 points):

In the following questions assume the variables a and b are signed integers and that the machine uses two's complement representation. Also assume that MAX_INT is the maximum integer, MIN_INT is the minimum integer, and W is one less than the word length (e.g., W = 31 for 32-bit integers).

Match each of the descriptions on the left with a line of code on the right (write in the letter). You will be given 2 points for each correct match.

1. One's complement of a   f
2. a.   b
3. a & b.   a
4. a * 7.   c
5. a / 4 .   e
6. (a < 0) ? 1 : -1 .   h

```
a. ~(~a | (b ^ (MIN_INT + MAX_INT)))
b. ((a ^ b) & ~b) | (~(a ^ b) & b)
c. 1 + (a << 3) + ~a
d. (a << 4) + (a << 2) + (a << 1)
e. ((a < 0) ? (a + 3) : a) >> 2
f. a ^ (MIN_INT + MAX_INT)
g. ~((a | (~a + 1)) >> W) & 1
h. ~((a >> W) << 1)
i. a >> 2
```

$1 + 8a + \sim a \rightarrow 8a - a = 7a$

$16a + 4a + 2a = \times$

7/15

Problem 3. (15 points):

Consider the following 16-bit floating point representation based on the IEEE floating point format:

- There is a sign bit in the most significant bit.
- The next seven bits are the exponent. The exponent bias is 63.
- The last eight bits are the significand.

The rules are like those in the IEEE standard (normalized, denormalized, representation of 0, infinity, and NAN).

As described in lecture, we consider the floating point format to encode numbers in a form:

$$(-1)^s \times m \times 2^E$$

where m is the *mantissa* and E is the exponent.

Fill in the table below for the following numbers, with the following instructions for each column:

Hex: The 4 hexadecimal digits describing the encoded form.

m**:** The fractional value of the mantissa. This should be a number of the form x or x/y, where x is an integer, and y is an integral power of 2. Examples include: 0, 67/64, and 1/256.

E: The integer value of the exponent.

Value: The numeric value represented. Use the notation x or $x \times 2^z$, where x and z are integers.

As an example, to represent the number 7/2, we would have $s = 0$, $m = 7/4$, and $E = 1$. Our number would therefore have an exponent field of 0x40 (decimal value $63 + 1 = 64$) and a significand field 0xC0 (binary 11000000_2), giving a hex representation 40C0.

You need not fill in entries marked "—".

Description	Hex	m	E	Value
−0	8000	0	0	—
Smallest value > 1	3F01	128/127	0	128/127
Largest Denormalized	00FF	127/128	−62	$\frac{1}{2^{62}} \cdot \frac{127}{128}$
−∞	FF00	—	—	—
Number with hex representation 3AA0	—	$\frac{13}{8}$	−3	13/64

Nice!

Problem 4. (15 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, and an int is, of course, encoded using 6 bits. Fill in the empty boxes in the table below. The following definitions are used in the table:

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

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

Expression	Decimal Representation	Binary Representation
Zero	0	00 0000
–	−6	11 1010
–	18	01 0010
ux	47	10 1111
y	−3	11 1101
$x >> 1$	−9	110111
TMax	31	01 1111
−TMin	TMin= −32	10 0000
TMin + TMin	0	00 0000

6

Problem 5. (6 points):

Which C function (fun7,fun8 or fun9) has the same effect as the assembly code shown? Write your answer in the blank space directly below.

fun8
____

```
int fun7(int a)
{
    return a * 30;
}

int fun8(int a)
{
    return a * 34;
}

int fun9(int a)
{
    return a * 18;
}
```

```
pushl %ebp
movl %esp,%ebp
movl 8(%ebp),%eax
sall $4,%eax
addl 8(%ebp),%eax
addl %eax,%eax
movl %ebp,%esp
popl %ebp
ret
```

$a \cdot 2^4$

$16 \cdot a + a$

$17a + 17a$

$34a$

8

Problem 6. (12 points):

Match each of the assembler routines on the left with the equivalent C function on the right.

```
foo1:
    pushl %ebp
    movl %esp,%ebp
    movl 8(%ebp),%eax       x
    sall $4,%eax            16x
    subl 8(%ebp),%eax       15x
    movl %ebp,%esp
    popl %ebp
    ret

foo2:
    pushl %ebp
    movl %esp,%ebp
    movl 8(%ebp),%eax       x
    testl %eax,%eax         x==0
    jge .L4
    addl $15,%eax           x+15
.L4:
    sarl $4,%eax            x>>4
    movl %ebp,%esp
    popl %ebp
    ret

foo3:
    pushl %ebp
    movl %esp,%ebp
    movl 8(%ebp),%eax       x
    shrl $31,%eax           x>>>31
    movl %ebp,%esp
    popl %ebp
    ret
```

```
int choice1(int x)
{
    return (x < 0);
}

int choice2(int x)
{
    return (x << 31) & 1;
}

int choice3(int x)
{
    return 15 * x;
}

int choice4(int x)
{
    return (x + 15) /4
}

int choice5(int x)
{
    return x / 16;
}

int choice6(int x)
{
    return (x >> 31);
}
```

Fill in your answers here:

foo1 corresponds to choice 3 — 4

foo2 corresponds to choice 5 — 4

foo3 corresponds to choice 1 — 0 +4

10

Problem 7. (10 points):

Condider the following assembly code for a C `for` loop:

```
loop:
        pushl %ebp
        movl %esp,%ebp
        movl 8(%ebp),%ecx      x
        movl 12(%ebp),%edx     y
        xorl %eax,%eax         result=0
        cmpl %edx,%ecx         x≤y
        jle .L4
.L6:
        decl %ecx              x--
        incl %edx              y++
        incl %eax              result ++
        cmpl %edx,%ecx         x>y
        jg .L6
.L4:
        incl %eax              ++result
        movl %ebp,%esp
        popl %ebp
        ret
```

Based on the assembly code above, fill in the blanks below in its corresponding C source code. (Note: you may only use the symbolic variables `x`, `y`, and `result` in your expressions below — *do not use register names.*)

```
int loop(int x, int y)
{
    int result;
    for ( result = 0 ; x > y ; result++ ) {
        x-- ;
        y++ ;
    }
    result++ ;
    return result;
}
```

6

Problem 8. (6 points):

Consider the following C functions and assembly code:

```
int fun1(int a, int b)
{
    if (a < b)
        return a;
    else
        return b;
}

int fun2(int a, int b)
{
    if (b < a)
        return b;
    else
        return a;
}

int fun3(int a, int b)
{
    unsigned ua = (unsigned) a;
    if (ua < b)
        return b;
    else
        return ua;
}
```

```
    pushl %ebp
    movl %esp,%ebp
    movl 8(%ebp),%edx      a
    movl 12(%ebp),%eax     b
    cmpl %eax,%edx         a≥b
    jge .L9
    movl %edx,%eax         if a<b -> ret a
.L9:
    movl %ebp,%esp
    popl %ebp
    ret
```

Which of the functions (fun1, fun2, or fun3) compiled into the assembly code shown? Write your answer in the blank space below.

fun1
____

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

```c
int silly(int n, int *p)
{
    int val, val2;

    if (n > 0)
        val2 = silly(n << 1, &val);
    else
        val = val2 = 0;

    *p = val + val2 + n;

    return val + val2;
}
```

This yields the following machine code:

```
silly:
        pushl %ebp
        movl %esp,%ebp
        subl $20,%esp
        pushl %ebx
        movl 8(%ebp),%ebx
        testl %ebx,%ebx
        jle .L3
        addl $-8,%esp
        leal -4(%ebp),%eax
        pushl %eax
        leal (%ebx,%ebx),%eax
        pushl %eax
        call silly
        jmp .L4
        .p2align 4,,7
.L3:
        xorl %eax,%eax
        movl %eax,-4(%ebp)
.L4:
        movl -4(%ebp),%edx
        addl %eax,%edx
        movl 12(%ebp),%eax
        addl %edx,%ebx
        movl %ebx,(%eax)
        movl -24(%ebp),%ebx
        movl %edx,%eax
        movl %ebp,%esp
        popl %ebp
        ret
```

8/15

Problem 9. (10 points):

1

A. Is the variable val stored on the stack? If so, at what byte offset (relative to %ebp) is it stored, and why is it necessary to store it on the stack?

Val isn't stored on the stack since its value is passed by reference after the func. returns

1

B. Is the variable val2 stored on the stack? If so, at what byte offset (relative to %ebp) is it stored, and why is it necessary to store it on the stack?

Stored at -4(ebp) since it is caller save register / in eax

3

C. What (if anything) is stored at -24(%ebp)? If something is stored there, why is it necessary to store it?

We return the value of n stored at the beginning of the code by the callee save routine back into ebx

3

D. If the assembly code above was assembled and then disassembled, what would the label 'silly' be replaced by in the disassembled code for the call instruction?

It will be replaced with the address of the function silly

0

E. What is implicitly retrieved from the stack by the ret instruction, and where is this information stored?

Implicitly we return a value stored in val through the pointer *p