CMSC 313 Spring 2010
Midterm Exam 2
Section 01
April 19, 2010

Name_____Score _____ out of 80

UMBC Username _____@umbc.edu

Notes:

a. Please write clearly. Unreadable answers receive no credit.
b. There are no intentional syntax errors in any code provided with this exam. If you think you see an error that would affect your answer, please bring it to my attention.

Multiple Choice - 2 points each.

Write the letter of the BEST answer in the blank provided in UPPERCASE

1. _____ Which sequence of operations is performed prior to **ret**

A.
```
movl %ebp, %esp
popl %ebp
```

B.
```
popl %ebp
movl $ebp, %esp
```

C.
```
popl %esp
movl %ebp, %esp
```

D.
```
pushl %ebp
movl %esp, %ebp
```

2. _____ What code is responsible for storing the return address when a function is called?

A. The caller
B. The callee
C. The kernel
D. The CPU

3. _____ On what variable types does C perform logical right shifts?

A. Signed types
B. Unsigned types
C. Signed and unsigned types
D. none

4. _____ On 32-bit x86 systems (eg Linux), where is the value of the caller's `%ebp` saved in relation to the current value of `%ebp` (assume a pointer is 32 bits)

A. there is no relationship between them
B. old `%ebp` is always stored at (`%ebp` - 4)
C. old `%ebp` is always stored at (`%ebp` + 4)
D. old `%ebp` is always stored at (`%ebp`)

5. _____Which of the following `movl` instructions is invalid?

A.
```
movl %esp, %ebp
```
B.
```
movl 0xdeadbeef, %eax
```
C.
```
movl (0xdeadbeef), %esp
```
D.
```
movl %ebx, 0xdeadbeef
```

6. _____ Given the declaration `short S[10][6]`, the memory address of `S[r][c]` can be calculated as

A. `S + 12*(r-1) + 2*(c-1)`
B. `S + 12 * r + 2*c`
C. `S + 6*(r-1) + 10*(c-1)`
D. `S + 6*r + 10*c`

7. _____ In C, the result of shifting by a value greater than it's type's width (size in bits) is

A. Illegal
B. Undefined
C. Zero
D. Encouraged by the C99 standard

8. _____ Extending a stack can be done by

A. Swapping the base pointer and the stack pointer
B. Subtracting a value from the stack pointer
C. Adding a value to the stack pointer
D. Executing the `ret` instruction

9. _____ The instruction pointer (%eip) contains

A. the address of the next instruction to be executed
B. the address of the current instruction being executed
C. the address of the current function being executed
D. the address of the calling function

10._____ The compiler DOES NOT use a jump table to implement a switch statement

A. If the values of the cases are "close" to each other
B. If the values of the cases are sorted
C. If the values of the cases are not "close" to each other
D. Unless there is no other choice

11. (6 points) Assume we are running a program on a 5-bit Linux/IA32 machine using 2's complement arithemetic. Complete the entries in the table below using the following definitions. Entries marked with " ---- " are unused.

```
int y = -9;
unsigned z = y;
```

Expression	Decimal Representation	Binary Representation
----	-5	
----		1 0010
y		
z		
y - z		
TMin		
TMax		

Points this Page_____

12. (10 points)

Consider the source code below, where M and N are constants declared with #define.

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

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

Suppose the above code generates the following assembly code:

```
sum_element:
        pushl %ebp
        movl %esp,%ebp
        movl 8(%ebp),%eax
        movl 12(%ebp),%ecx
        sall $2,%ecx
        leal 0(,%eax,4),%edx
        addl %eax,%edx
        leal (%eax,%eax,2),%eax
        movl mat2(%ecx,%eax,4),%eax
        addl mat1(%ecx,%edx,4),%eax
        movl %ebp,%esp
        popl %ebp
        ret
```

What are the values of M and N?

M	**N**

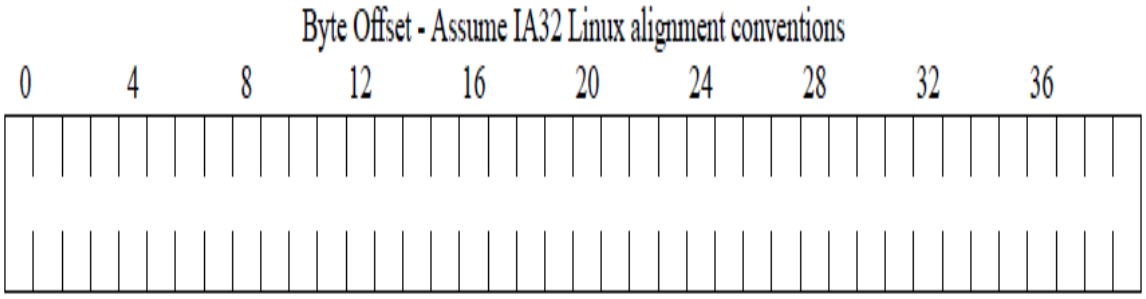
Points this Page_____

13. (8 points)

Consider the following definition of the struct named BOB.

```
typedef struct {
      char c;
      double d;
      short s;
      double *pd;
      float f;
      char *pc;
} BOB;
```

Using the template below (allowing a maximum of 40 bytes), indicate the allocation of data for a structure of type Bob. Mark off and label the areas for each individual element (there are 6 of them). Cross hatch the parts that are allocated, but not used (to satisfy alignment). **Clearly indicate the right hand (end) boundary of the data structure with a vertical line**.



Points this Page_____

14. (12 points)

Consider the following assembly representation of a function name "mystery" containing a for loop:

```
mystery:
    pushl %ebp
    movl %esp,%ebp
    movl 12(%ebp),%edx
    movl 16(%ebp),%eax
    addl 8(%ebp),%edx
    testl %edx,%edx
    jle .L4
.L6:
    incl %eax
    decl %edx
    testl %edx,%edx
    jg .L6
.L4:
    movl %ebp,%esp
    popl %ebp
    ret
```

Fill in the blanks to provide the functionality of the loop:

```
int mystery(int a, int b, int c)
{
    int x, y;
    y = ____________________;

    for (_____________; _____________; _____________)
    {
        _____________;
    }

    return _____________;
}
```

15. (10 points):
Fill in the blanks of the C code. The corresponding assembly language is given below. Note that there are two columns of assembly code.

Assembly code:

```
08048430 <foo>:
8048430: push %ebp
8048431: mov %esp,%ebp
8048433: push %ebx
8048434: mov 0x8(%ebp),%ebx
8048437: xor %eax,%eax
8048439: cmp $0x4,%ebx
804843c: mov 0xc(%ebp),%ecx
804843f: mov 0x10(%ebp),%edx
8048442: ja 804846f <foo+0x3f>
8048444: jmp *0x8048500(,%ebx,4)
804844b: nop
804844c: mov %ecx,%eax
804844e: and %edx,%eax
8048450: jmp 804846f <foo+0x3f>
8048452: mov %esi,%esi
8048454: mov %ecx,%eax
8048456: or %edx,%eax
8048458: jmp 804846f <foo+0x3f>
804845a: mov %esi,%esi
804845c: mov %ecx,%eax
804845e: xor %edx,%eax
8048460: jmp 804846f <foo+0x3f>
8048462: mov %esi,%esi
8048464: mov %ecx,%eax
8048466: not %eax
8048468: jmp 804846f <foo+0x3f>
804846a: mov %esi,%esi
804846c: lea (%edx,%ecx,1),%eax
804846f: mov (%esp,1),%ebx
8048472: leave
8048473: ret
```

Memory information given by gdb -- the contents of memory starting at memory location `0x8048500`, displayed as 4-byte values

```
>gdb foo
(gdb) x /8w 0x8048500
0x8048500 : 0x0804844c 0x08048454 0x0804845c 0x08048464
0x8048510 : 0x0804846c 0x00000000 0x00000000 0x00000000
```

C code:

```
int foo(int op, int a, int b)
int result = 0;
switch (op)
{
     case 0: _____________________________________; break;

     case 1: _____________________________________; break;

     case 2: _____________________________________; break;

     case 3: _____________________________________; break;

     case 4: _____________________________________; break;
}
return result;
```

16. (8 points) Consider the following pairs of C functions and assembly code. In the table below, indicate which assembly language code fragment on the right corresponds to each C function on the left. There is not necessarily a one-to-one correspondence (i.e. some C function(s) may have no corresponding assembly code block and some C function(s) may have more than one corresponding assembly code block). If a function has no corresponding assembly code block, put an X in the table.

```
int F1( int x )
{
     return (x + 2) >> 12;
}

int F2( int x )
{
     return x * 2 + (x - 3);
}


int F3( int x )
{
     return x * 35;
}


int F4( int x )
{
     return (x + 2) * 2;
}
```

A:

```
pushl   %ebp
movl    %esp,%ebp
movl    0x8(%ebp),%eax
addl    %eax,%eax
addl    0x8(%ebp),%eax
subl    $0x3,%eax
popl    %ebp
ret
```

B:

```
pushl   %ebp
movl    %esp,%ebp
movl    0x8(%ebp),%eax
addl    %eax,%eax
addl    $0x4,%eax
popl    %ebp
ret
```

C:

```
pushl   %ebp
movl    %esp,%ebp
movl    0x8(%ebp),%eax
addl    $0x2,%eax
sarl    $0xc,%eax
popl    %ebp
ret
```

D:

```
pushl   %ebp
movl    %esp,%ebp
movl    0x8(%ebp),%edx
movl    %edx,%eax
shll    $0x2,%eax
addl    %edx,%eax
leal    0x0(,%eax,8),%edx
movl    %edx,%ecx
subl    %eax,%ecx
movl    %ecx,%eax
popl    %ebp
ret
```

Enter your answers in this table

Function	F1	F2	F3	F4
Assembly Block				

Points this Page_____

17. (6 points)

This problem tests your ability of matching assembly code to corresponding C pointer code. Note that some of the C code below doesn't do anything useful.

```
int fun10(int ap, int bp)
{
     int a = ap;
     int b = bp;

     return *(&a + b);
}

int fun20(int *ap, int bp)
{
     int *a = ap;
     int b = bp

     return *(a + b);
}

int fun30(int ap, int *bp)
{
     int a = ap;
     int b = *bp;

     return *(&a + b);
}
```

```
pushl %ebp
movl %esp,%ebp
subl $24, %esp;
movl 8(%ebp), %eax
movl %eax, -4(%ebp)
movl 12(%ebp), %edx
movl (%edx), %eax
sall $2, %eax
movl -4(%eax, %ebp), %eax
movl %ebp, %esp
popl %ebp
ret
```

Which of the functions above on the left generated the assembly language above on the right? Write the name of the function in the box.

Points this Page_____