CMSC 313 Fall 2010
Midterm Exam 2
Section 01
Nov 22, 2010

Name_____Score _____ out of **75**

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 for each question in the corresponding box at the bottom of the page. Write your answer in UPPERCASE.

1.. Which of the following registers stores an integer function return value in IA32?

A. %eax
B. %ebx
C. %ecx
D. %esp

2. By default in Intel x86, the stack

A. is located at the "bottom" of memory
B. grows down toward smaller addresses
C. grows up towards larger addresses
D. is located in the heap

3. Arguments passed to functions in Intel IA32 are passed via

A. the stack
B. registers
C. a combination of the stack and registers
D. main memory

4. In two's complement arithmetic, what is the value of ***-TMin***?

A. 0
B. -1
C. TMin
D. TMax

5. With respect to byte ordering, Intel x86 machines such as the GL servers

A. are little endian
B. are big endian
C. have no "endianess"
D. have their "endianess" determined by the operating system

1	2	3	4	5

Points this Page_____

6. Given the declaration **`int A[5][7];`**, the memory address of **`A[r][c]`** can be calculated as

A. `A + 20 *(r-1) + 28 *(c-1)`
B. `A + 20 * r + 28 * c`
C. `A + 10 * r + 14 * c`
D. `A + 5 * r +  7 * c`

7. What is the difference between arithmetic and logical right shift?

A. C uses arithmetic right shift exclusively; Java uses logical right shift exclusively
B. They fill in different bits on the left
C. Logical shifts are only used for logical comparison such as "less than"
D. There is no difference

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

6	7	8	9	10

Points this Page_____

11. (5 points) Assume we are running a program on a **6-bit** Linux/IA32 machine using **2's complement arithmetic**. Complete the entries in the table below using the following definitions. Entries marked with " --- " are unused. For your convenience, the powers of two from 2^0 to 2^{10} are listed here

```
int powersOf2[ ] = {1, 2, 4, 8, 16, 32, 64, 128, 256, 512, 1024};
```

```
int x = -4;
unsigned y = x;
```

Expression	Decimal Representation	Binary Representation
----		10 1010
x	---	
y		
x - y		
TMin		
TMax		

Points this Page_____

12. **(5 points)** Examine the assembly language code below. Then complete the C code for the function named Mystery.

Dump of assembler code for function Mystery:

```
0x08048384 <Mystery+0>:     push   %ebp
0x08048385 <Mystery+1>:     mov    %esp,%ebp
0x08048387 <Mystery+3>:     mov    0xc(%ebp),%eax
0x0804838a <Mystery+6>:     shl    $0x2,%eax
0x0804838d <Mystery+9>:     add    0x8(%ebp),%eax
0x08048390 <Mystery+12>:    add    $0x4,%eax
0x08048393 <Mystery+15>:    mov    (%eax),%edx
0x08048395 <Mystery+17>:    mov    0x10(%ebp),%eax
0x08048398 <Mystery+20>:    shl    $0x2,%eax
0x0804839b <Mystery+23>:    add    0x8(%ebp),%eax
0x0804839e <Mystery+26>:    sub    $0x4,%eax
0x080483a1 <Mystery+29>:    mov    (%eax),%eax
0x080483a3 <Mystery+31>:    lea    (%edx,%eax,1),%eax
0x080483a6 <Mystery+34>:    pop    %ebp
0x080483a7 <Mystery+35>:    ret
```

```
int Mystery(int a[],  int i, int j)
{

     return ____________________________________________;
}
```

13. (5 points) Complete the C code for the following puzzle which might have appeared in your third project. Recall that constants may not be more than 8-bits long (i.e. 0xFF)

```
/* getByte - Extract byte n from word x
 *   Bytes numbered from 0 (LSB) to 3 (MSB)
 *   Examples: getByte(0x12345678,1) = 0x56
 *   Legal ops: ! ~ & ^ | + << >>
 *   Max ops: 6
 */
int getByte(int x, int n) {




}
```

13. (12 points) Consider the structure definitions below, then answer the questions below. Write your answer in the corresponding box at the bottom of the page. More questions using these structs are found on the following page.

```
struct s1 {
  short x;
  int y;
};
```

```
struct s2 {
  struct s1 a;
  struct s1 *b;
  int x;
  char c;
  int y;
  char e[3];
  int z;
};
```

A. What is the size of **`struct s1`**?
B. How many bytes of **`struct s1`** are wasted for padding?
C. What is the size of **`struct s2`**?
D. How many bytes of **`struct s2`** are wasted for padding including those wasted in **`s1`**?)

A	B	C	D

Points this Page_____

13. **(continued)** Consider the following C functions which use the structures above.

```
int fun1( struct s2 *s )
{
  return s->a.y;
}
```

```
int fun3( struct s2 *s )
{
  return s->z;
}
```

```
int *fun2( struct s2 *s )
{
  return &s->z;
}
```

```
int fun4( struct s2 *s )
{
  return s->b->y;
}
```

Indicate which assembly language program below corresponds to the functions above. Each function is used only once. Write the name of the function in the corresponding box at the bottom of the page.

```
Code Block "A"

push      %ebp
mov       %esp, %ebp
mov       0x8(%ebp), %eax
add       $0x1c, %eax
pop       %ebp
ret
```

```
Code Block "B"

push      %ebp
mov       %esp, %ebp
mov       0x8(%ebp), %eax
mov       0x8(%eax), %eax
mov       4(%eax), %eax
pop       %ebp
ret
```

```
Code Block "C"

push      %ebp
mov       %esp, %ebp
mov       0x8(%ebp), %eax
mov       $0x1c(%eax), %eax
pop       %ebp
ret
```

```
Code Block "D"

push      %ebp
mov       %esp, %ebp
mov       0x8(%ebp), %eax
mov       4(%eax), %eax
pop       %ebp
ret
```

Code Block	**A**	**B**	**C**	**D**
Function				

Points this Page_____

14. (10 points) The assembly code is the disassembled output of the function **`for1`** below. Fill in the blanks in the C code for the function

```
gdb) disassemble for1

0x08048384 <for1+0>:    push   %ebp
0x08048385 <for1+1>:    mov    %esp,%ebp
0x08048387 <for1+3>:    sub    $0x10,%esp
0x0804838a <for1+6>:    movl   $0x1,-0x4(%ebp)
0x08048391 <for1+13>:   mov    0x8(%ebp),%eax
0x08048394 <for1+16>:   imul   0xc(%ebp),%eax
0x08048398 <for1+20>:   mov    %eax,-0x8(%ebp)
0x0804839b <for1+23>:   jmp    0x80483a5 <for1+33>
0x0804839d <for1+25>:   addl   $0x2,-0x4(%ebp)
0x080483a1 <for1+29>:   subl   $0x3,-0x8(%ebp)
0x080483a5 <for1+33>:   cmpl   $0xc,-0x8(%ebp)
0x080483a9 <for1+37>:   jg     0x804839d <for1+25>
0x080483ab <for1+39>:   mov    -0x4(%ebp),%eax
0x080483ae <for1+42>:   leave
0x080483af <for1+43>:   ret
End of assembler dump.

int for1(int a, int b)
{
     int x, y;

     y = ___________;

     for (x = _________;  ________; _________________)

          y = _________________;


     return y;
}
```

15. (10 points): The gdb output below represents the assembly language for the function **`mySwitch(int a, int b)`** which is found on the next page. Examine the assembly language code, then fill in the blanks in the C code for **`mySwitch( )`** and answer the questions that follow on the next page.

```
Dump of assembler code for function mySwitch:
0x080483b4 <mySwitch+0>:        push   %ebp
0x080483b5 <mySwitch+1>:        mov    %esp,%ebp
0x080483b7 <mySwitch+3>:        sub    $0x14,%esp
0x080483ba <mySwitch+6>:        movl   $0x0,-0x8(%ebp)
0x080483c1 <mySwitch+13>:       movl   $0xdeadbeef,-0x4(%ebp)
0x080483c8 <mySwitch+20>:       mov    0x8(%ebp),%eax
0x080483cb <mySwitch+23>:       mov    %eax,%edx
0x080483cd <mySwitch+25>:       imul   0xc(%ebp),%edx
0x080483d1 <mySwitch+29>:       mov    %edx,-0x14(%ebp)
0x080483d4 <mySwitch+32>:       cmpl   $0x7,-0x14(%ebp)
0x080483d8 <mySwitch+36>:       ja     0x804842a <mySwitch+118>
0x080483da <mySwitch+38>:       mov    -0x14(%ebp),%edx
0x080483dd <mySwitch+41>:       mov    0x8048588(,%edx,4),%eax
0x080483e4 <mySwitch+48>:       jmp    *%eax
0x080483e6 <mySwitch+50>:       movl   $0x24,-0x8(%ebp)
0x080483ed <mySwitch+57>:       jmp    0x8048431 <mySwitch+125>
0x080483ef <mySwitch+59>:       mov    0xc(%ebp),%eax
0x080483f2 <mySwitch+62>:       add    0x8(%ebp),%eax
0x080483f5 <mySwitch+65>:       mov    %eax,-0x4(%ebp)
0x080483f8 <mySwitch+68>:       mov    0x8(%ebp),%eax
0x080483fb <mySwitch+71>:       mov    %eax,-0x4(%ebp)
0x080483fe <mySwitch+74>:       mov    -0x8(%ebp),%eax
0x08048401 <mySwitch+77>:       imul   0xc(%ebp),%eax
0x08048405 <mySwitch+81>:       mov    %eax,-0x8(%ebp)
0x08048408 <mySwitch+84>:       jmp    0x8048431 <mySwitch+125>
0x0804840a <mySwitch+86>:       mov    -0x8(%ebp),%eax
0x0804840d <mySwitch+89>:       cmp    -0x4(%ebp),%eax
0x08048410 <mySwitch+92>:       sete   %al
0x08048413 <mySwitch+95>:       movzbl %al,%eax
0x08048416 <mySwitch+98>:       mov    %eax,0x8(%ebp)
0x08048419 <mySwitch+101>:      mov    -0x8(%ebp),%eax
0x0804841c <mySwitch+104>:      cmp    -0x4(%ebp),%eax
0x0804841f <mySwitch+107>:      setl   %al
0x08048422 <mySwitch+110>:      movzbl %al,%eax
0x08048425 <mySwitch+113>:      mov    %eax,0xc(%ebp)
0x08048428 <mySwitch+116>:      jmp    0x8048431 <mySwitch+125>
0x0804842a <mySwitch+118>:      movl   $0xdeadbeef,-0x8(%ebp)
0x08048431 <mySwitch+125>:      mov    -0x4(%ebp),%eax
0x08048434 <mySwitch+128>:      cmp    -0x8(%ebp),%eax
0x08048437 <mySwitch+131>:      setl   %al
0x0804843a <mySwitch+134>:      movzbl %al,%eax
0x0804843d <mySwitch+137>:      leave
0x0804843e <mySwitch+138>:      ret
End of assembler dump.

(gdb) x/12wx 0x8048588
0x8048588:      0x080483ef      0x080483e6      0x0804840a      0x0804842a
0x8048598:      0x080483f8      0x0804842a      0x0804842a      0x08048419
0x80485a8:      0x75706e49      0x77742074      0x6e69206f      0x203a7374
(gdb)
```

 Points this Page_____

One (1) point each blank and each question. Place your answer to the questions in the corresponding box at the the bottom of the page

```
int mySwitch (int a, int b)
{
     int y = 0;
     int x = ____________________;

     switch(_________________)
     {
     case 1:
          y =  36;
          break;

     case 0:
          x =  _______________;

     case ________:
          x = a;
          y *= b;
          break;

     case 2 :
          a = y == x;

     case _______:
          b = y < x;
          break;

     default:
          y = 0xdeadbeef;
     }

     return ____________________________ ;
}
```

A. What is the highest case number in the switch statement?
B. Which value(s) use the default case?
C. At what address (relative to **`%ebp`** or **`%esp`**) is **`x`** stored?
D. At what memory address is the code for **`case 2`** located?

A	B	C	D

16. (8 points) Consider the following C functions and blocks of assembly code. In the table below, indicate which assembly language code block 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 * x * x;
}



int F2( int x )
{
     return x * 48;
}



int F3( int x )
{
    return (x * 8 ) + 17;
}



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

A:

```
push   %ebp
mov    %esp,%ebp
mov    0x8(%ebp),%eax
shl    $0x3,%eax
add    $0x11,%eax
pop    %ebp
ret
```

B:

```
push   %ebp
mov    %esp,%ebp
mov    0x8(%ebp),%eax
imul   0x8(%ebp),%eax
add    %eax,%eax
pop    %ebp
ret
```

C:

```
push   %ebp
mov    %esp,%ebp
mov    0x8(%ebp),%eax
imul   0x8(%ebp),%eax
imul   0x8(%ebp),%eax
pop    %ebp
ret
```

D:

```
 push   %ebp
mov    %esp,%ebp
mov    0x8(%ebp),%edx
mov    %edx,%eax
add    %eax,%eax
add    %edx,%eax
shl    $0x4,%eax
pop    %ebp
ret
```

Function	**F1**	**F2**	**F3**	**F4**
Assembly Block				

 Points this Page_____