

Computer Organization

Fall 2004

Test #1

Name _____ *Answer Key* _____

There are 6 pages - make sure you have all of them.
 Answer all questions - pay attention to the # of points for each question.
 Don't leave anything blank - partial credit is always possible!

Question 1 (15 pts): Complete the following tables by filling in each blank space with the appropriate value.

Decimal	8 bit 2's complement binary	Hex
1	00000001	01
-72	10111000	B8
-63	11000001	C1
-127	10000001	81

Decimal	8 bit unsigned binary	Hex
232	11101000	E8
193	11000001	C1
65	1000001	41

Decimal	IEEE Single Precision (32 bit)		
	Sign (1)	Exponent (8)	Significand (23)
14.25	0	10000010	11001000000000000000000

Question 2: This question has three parts, all involve the following C statement:

$$y = 12 * x * x - 3 * x + 4;$$

Part a (10 pts): Write IA32 (x86) assembly code that will accomplish the computation described by the C statement shown above. Your assembly code should assume the following:

- all variables are of type `int`
- `x` is initially in register `%edx`
- At the end of your assembly code, the value of `y` should be placed in register `%eax`

Your code must be based on the assembly syntax used by `gcc/gas` (the `gnu` assembly syntax). You do **not** need to write an assembly *subroutine*, just code that computes the expression. Comment your code!

```
movl    %edx,%ecx          # ecx = x
imull   %ecx,%ecx          # ecx = x * x
imull   $12,%ecx           # ecx = 12 * x * x

leal    (%edx,%edx,2),%edx # edx = 3 * x
subl    %edx,%ecx          # ecx = 12*x*x - 3 * x
addl    $4,%ecx            # ecx = 12*x*x - 3 * x + 4
movl    %ecx,%eax
```

Question 2 continued (parts b and c involve the same C statement, shown here again:)

$$y = 12*x*x - 3*x + 4;$$

Part b (5pts): Show how this computation might look when using a processor based on a *stack-based instruction set architecture* (generic assembly is fine, with instructions like "push y", "add" and "pop tmp").

```
push x
push x
push 12
mult
mult
push x
push 3
mult
subtract          # could be either order!
push 4
add
pop y
```

Part c (5pts): Show how the computation might look when using a processor based on an *accumulator based instruction set*. Generic assembly is fine here also, instructions like "add y", "load z" and "store x" are expected.

```
load x
mult 12
subtract 3        # acc = 12*x-3
mult x           # acc = (12*x-3)*x
add 4            # acc = (12*x-3)*x+4
store y
```

Question 3 (20 pts): Write a C function named `parity()` with the following prototype:

```
int parity(int x);
```

The function should return 0 if the number of bits of `x` that have the value 1 is even, 1 if the number of bits with the value 1 is odd. For example, `parity(0)` is 0, `parity(9)` is 0 and `parity(4)` is 1.

You should assume that the datatype `int` is 32 bits.

```
int parity(int x) {
    int cnt=0;
    int i;
    for (i=0;i<32;i++) {
        if (x & (0x01 << i))
            cnt++;
    }
    return(cnt%2);
}
```

Question 4 (10 pts): Write an IA32 function named `minimum()` with the following prototype:

```
unsigned int minimum(unsigned int x, unsigned int y);
```

I've provided the subroutine startup and cleanup code for you - just write the IA32 code that goes in the middle (including comments!)

```
minimum:
    pushl    %ebp                # setup
    movl    %esp, %ebp          # setup

    movl    8(%ebp), %eax        # eax is x
    movl    12(%ebp), %edx       # edx is y

    cmpl    %eax, %edx
    jae    .L1
    movl    %edx, %eax

.L1:

    leave   # prepare for return
    ret     # return
```

Question 6 (15 pts): Show the output generated by the following C program:

```
#include <stdio.h>

int *foo( int *x, int *y ) {
    if (*x > *y)
        return(x);
    else
        return(y);
}

int blah(char **s) {
    int i=0;
    while (*s) {
        s++; i++;
    }
    return(i);
}

void britney(int x, int y) {
    int *tmp;

    if (x>y) {
        tmp = &x;
        x = y;
        y = *tmp;
    }
}

char *hellos[] = {
    "Hi",
    "Hola",
    "Privet",
    "Konnichiwa",
    NULL
};

int main() {
    int a,b;
    int *p;

    a=30; b=30;
    p = foo(&a,&b);
    printf("p is %d\n",*p);

    britney(a,b);
    printf("p is now %d\n",*p);

    printf("blah returns %d\n",blah(hellos));
    return(0);
}
```

Output is:
p is 30
p is now 30
blah returns 4

Question 6 (8 pts): Consider the following assembly code for a for loop:

```
foo:
    pushl    %ebp
    movl    %esp, %ebp

    movl    8(%ebp), %edx
    movl    12(%ebp), %ecx
    xorl    %eax, %eax

.L2:
    cmpl    $0, %ecx
    jle    .L3
    subl    %edx, %ecx
    incl    %eax
    jmp    .L2

.L3:
    leave
    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 foo(int x, int y) {
    int result;

    for ( _result=0__ ; _ y>0 _; result++ ) {
        _____ y=y-x _____;
    }

    return(_ result__);
}
```

Question 7 (12 pts). This question involves some IA32 assembly language instructions. For each part you should assume that the registers initially have the values shown here:

`%eax = 0x00000000`

`%ebx = 0x80000000`

`%ecx = 0x00000010`

`%edx = 0xF0000000`

`%esp = 0x00001000`

Part a (4 pts): What is the address of the source operand in the following IA32 instruction:

```
movl 12(%ebx, %ecx, 4), %esi
```

```
0x80000000+0x10*4+12 = 0x8000004c
```

Part b (4 pts): After execution of the instruction `pushl %edx`, show the contents of the following registers:

`%edx = 0xF0000000`

`%esp = 0x00000FFC`

Part c (4 pts): After execution of the instruction `leal 4(%ecx, %eax, 2), %edx`, what is the

```
%edx = 0x00000010 + 0x00000000*2 + 4 = 0x00000014
```

Part d (4 pts): We want to increment the 32bit integer stored at memory location 0x80000010. Show a *single* IA32 instruction that will accomplish this (assuming the register values are all as shown at the top of this page).

```
incl (%ebx,%ecx)
```

```
or
```

```
incl 16(%ebx)
```