

Computer Organization

Fall 2004 Test #2

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 (20 pts): Consider the following IA32 assembly code for a function named `floop`:

```
floop:
    pushl   %ebp
    movl    %esp, %ebp
    movl    8(%ebp), %edx
    movl    $13, %eax
    movl    $0, %ecx
    cmpl   %edx, %ecx
    jge    .L8
.L6:
    subl   %ecx, %eax
    decl  %edx
    incl  %ecx
    cmpl  %edx, %ecx
    jl    .L6
.L8:
    leave
    ret
```

Fill in the blanks in the following C version of `floop` to provide the same functionality as the IA32 version:

```
int floop(int x) {
    int i;
    int result= 13, 4 points each

    for ( i=0 ; i<x ; i++) {

        result = result-i;

        x = x-1;
    }
    return result;
}
```

Question 2 (20 pts): Below is the C code and corresponding assembly code for a function that deals with 2-dimensional arrays where M and N are defined elsewhere. Your job is to determine what M and N are by looking at the assembly code. Show your work (partial credit is available).

```
int a[M][N];
int b[N][M];

int foo(int i, int j) {
    a[i][j] = b[j][i];
    return(i);
}
```

```
foo:
    pushl   %ebp
    movl   %esp, %ebp
    movl   8(%ebp), %eax
    sall   $3, %eax
    movl   %eax, %ecx
    addl   12(%ebp), %ecx
    movl   12(%ebp), %edx
    movl   %edx, %eax
    sall   %eax
    addl   %edx, %eax
    leal   0(,%eax,4), %edx
    addl   %edx, %eax
    addl   8(%ebp), %eax
    movl   b(,%eax,4), %eax
    movl   %eax, a(,%ecx,4)
    movl   8(%ebp), %eax
    leave
    ret
```

M = 15
N = 8

access to a[i][j] is based on a + (j+i*8)*4 which indicates that N is 8

access to b[j][i] is based on b + [i+(3*j)*4+3*j]*4 which is (i+j*15)*4

This means M is 15

Partial credit for any correct work shown. If they get the numbers backwards (M=8 and N=15) take off 5 points.

Question 3 (20 pts): Consider the C code for a function named `silly()` and a function named `callit()` that calls `silly`. The C code and corresponding IA32 assembly (generated by `objdump`) is shown below.

```
void silly(char *s) {
    int x=12;
    char buf[2];
    strcpy(buf,s);
}

void callit() {
    silly("01234567890123456789");
}
```

```
08048370 <silly>:
8048370:    55                push   %ebp
8048371:    89 e5             mov    %esp,%ebp
8048373:    83 ec 08         sub    $0x8,%esp
8048376:    c7 45 fc 0c 00 00 00  movl  $0xc,-4(%ebp)
804837d:    83 ec 08         sub    $0x8,%esp
8048380:    ff 75 08         pushl 0x8(%ebp)
8048383:    8d 45 fa         lea   -6(%ebp),%eax
8048386:    50                push  %eax
8048387:    e8 24 ff ff ff   call  80482b0 <strcpy>
804838c:    83 c4 10         add    $0x10,%esp
804838f:    c9                leave
8048390:    c3                ret

08048391 <callit>:
8048391:    55                push   %ebp
8048392:    89 e5             mov    %esp,%ebp
8048394:    83 ec 08         sub    $0x8,%esp
8048397:    83 ec 0c         sub    $0xc,%esp
804839a:    68 9c 84 04 08   push  $0x804849c
804839f:    e8 cc ff ff ff   call  8048370 <silly>
80483a4:    83 c4 10         add    $0x10,%esp
80483a7:    c9                leave
80483a8:    c3                ret

Note: 0x804849c is the address of the string "012345678901234567890"
```

Question 3 continued (these questions refer to the code on the previous page)

Assume that `callit()` is called which then calls `silly("012345678901234567890")`. The questions below deal with the state of the registers and memory after the `leave` instruction has completed (`%eip` is `0x0848390`).

Notes:

- The ASCII codes for '0' through '9' are `0x30 ... 0x39`
- The `leave` instruction does the following:

```
movl %ebp, %esp
popl %ebp
```
- IA32 machines are little-endian.
- C strings are null terminated.

Show the value of the following registers and memory locations as hex values (4 points each):

`buf[0]` (single byte): 0x30

`buf[1]` (single byte): 0x31

`%ebp` (4 bytes): 0x39383736

`x` (4 bytes): 0x35343332

To what address will the function return ? (4 bytes) 0x33323130

4 points for each answer.

Take off 5 points total for byte order mistakes.

partial credit is possible:

- if the last three are off by two bytes take off 5 points total (they think there are two bytes between the end of `buf` and the start of `x`).

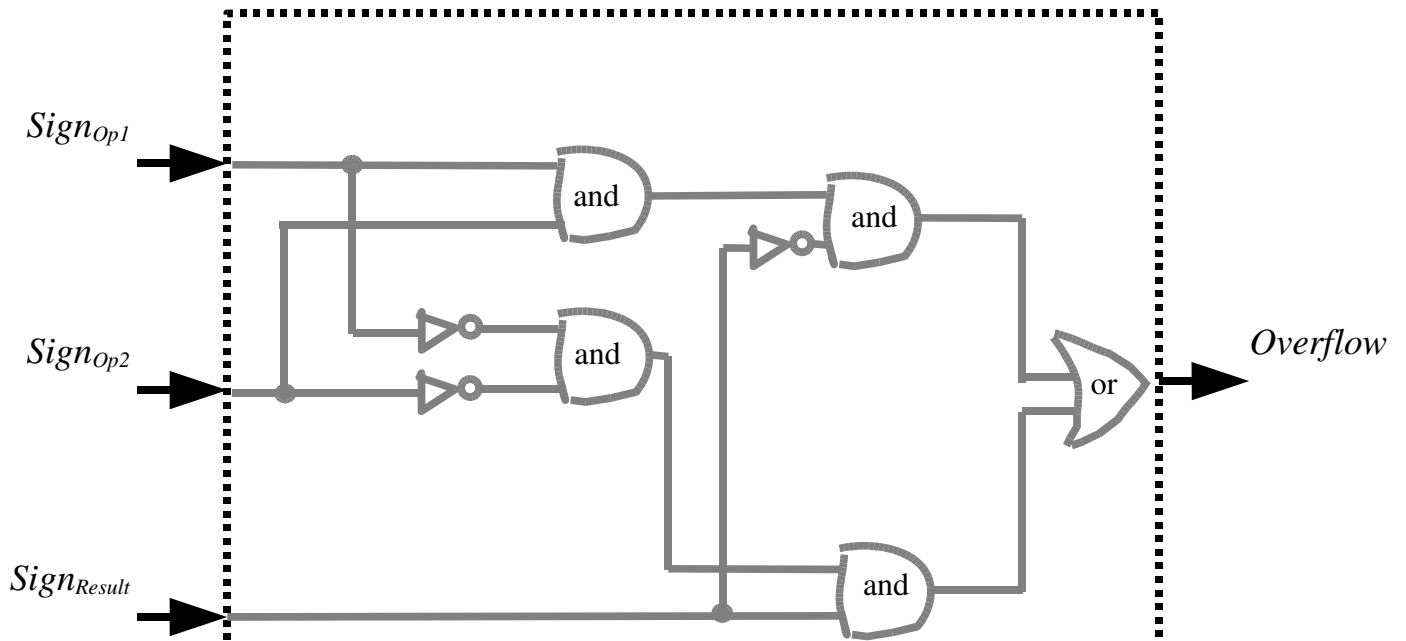
- anyone who seems to understand (a small mistake that makes many or all the answers wrong) should get at least 10 points.

- take off 5 points if they don't understand ASCII

Question 4 (15 pts): We want to create a combinational circuit that can detect an overflow condition after an addition operation. Recall that this function should be true (indicating that an overflow has occurred) when the sign bits of the two operands ($Sign_{Op1}$, $Sign_{Op2}$) are the same, and the sign of the result ($Sign_{Result}$) is different. The following truth table describes this as the boolean function *overflow* of three variables: $Sign_{Op1}$, $Sign_{Op2}$ and $Sign_{Result}$.

$Sign_{Op1}$	$Sign_{Op2}$	$Sign_{Result}$	<i>Over flow</i>
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	0
1	0	0	0
1	0	1	0
1	1	0	1
1	1	1	0

Draw a logic diagram that implements this circuit using AND, OR and NOT gates. you may also use multiplexors and/or decoders.



Grading Notes:

There are many other (and better) ways to do this...

Anyone that has the general idea gets 10 points.

Doesn't understand overflow or my table: -5 pts

Question 5 (25 pts): The following C function computes factorials:

```
int factorial(int x) {
    if (x<=1) {
        return(1);
    } else {
        return( x * factorial(x-1) );
    }
}
```

Translate this C code to an IA32 assembly language subroutine that follows the GCC calling convention and register usage conventions. Recall that %ebx, %esi and %edi are *callee-save* registers.

IMPORTANT! Your IA32 subroutine must be code that could be generated from the above C code, so it must be recursive!

factorial:

```
    push %ebp                # stack setup
    movl %esp,%ebp

    movl 8(%ebp),%ecx        # ecx is x
    movl $1,%eax            # default return value is 1
    cmpl $1,%ecx            # is ecx > 1?
    jle done                # no - return(1)

    decl %ecx                # x-1
    pushl %ecx
    call factorial          # factorial(x-1)
    imull 8(%ebp),%eax      # x * factorial(x-1)

done:
    leave
    ret
```

Grading Notes:

lose 15 points if not recursive!

things to look for any point deductions:

wrong jmp: -5

can't find x (is at 8(%ebp)) -10

return value not in eax: -5