

Computer Organization

Spring 2005 Test #2

Name _____

There are 6 questions - 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 (18 pts): Write the C function `foo` that could be compiled into the IA32 assembly code shown below.

```
foo:
    pushl   %ebp
    movl    %esp, %ebp
    subl   $8, %esp
    movl   $0, -4(%ebp)
    movl   $0, %ecx

.L2:
    movl   %ecx, %eax
    cmpl  8(%ebp), %eax
    jl    .L4
    jmp   .L3

.L4:
    movl   %ecx, %edx
    leal  -4(%ebp), %eax
    addl  %edx, (%eax)
    jmp   .L2

.L3:
    movl  -4(%ebp), %eax
    leal (%eax,%eax,4), %eax
    leave
    ret
```

Question 2 (15 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 array_x[M][N];
int array_y[N][M];

int eswap(int i, int j) {
    int tmp;
    tmp = array_y[j][i];
    array_y[j][i] = array_x[i][j];
    array_x[i][j]=tmp;
}
```

M =

N =

```
eswap:
    pushl   %ebp
    movl   %esp, %ebp
    pushl   %edi
    pushl   %esi
    pushl   %ebx
    movl   8(%ebp), %edx
    movl   12(%ebp), %ecx
    leal   (%ecx,%ecx,4), %ebx
    addl   %edx, %ebx
    movl   $array_y, %esi
    movl   (%esi,%ebx,4), %edi
    leal   (%edx,%edx,4), %eax
    leal   (%edx,%eax,2), %eax
    addl   %ecx, %eax
    movl   $array_x, %ecx
    movl   (%ecx,%eax,4), %edx
    movl   %edx, (%esi,%ebx,4)
    movl   %edi, (%ecx,%eax,4)
    popl   %ebx
    popl   %esi
    popl   %edi
    popl   %ebp
    ret
```

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

```
typedef struct {
    char c;
    int x;
} myrec;

void clobber_me(char *s) {
    myrec m;
    char buf[4];
    strcpy(buf,s);
    printf("m.c is %c and m.x is %x\n",m.c,m.x);
}

int main() {
    clobber_me("01234567890123456789");
    return(1);
}
```

```
080483ec <clobber_me>:
80483ec:    55                push   %ebp
80483ed:    89 e5             mov   %esp,%ebp
80483ef:    83 ec 20          sub   $0x20,%esp
80483f2:    ff 75 08          pushl 0x8(%ebp)
80483f5:    8d 45 f4          lea  0xffffffff4(%ebp),%eax
80483f8:    50               push  %eax
80483f9:    e8 22 ff ff ff   call  8048320 <strcpy@plt>
80483fe:    ff 75 fc          pushl 0xffffffffc(%ebp)
8048401:    0f be 45 f8       movsbl 0xffffffff8(%ebp),%eax
8048405:    50               push  %eax
8048406:    68 68 85 04 08   push  $0x8048568
804840b:    e8 00 ff ff ff   call  8048310 <printf@plt>
8048410:    c9               leave
8048411:    c3               ret
```

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

Assume that `clobber_me()` is called by `main` (as shown in the C code) and passed the address of the string `"012345678901234567890"`.

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.
- a C `int` variable requires 4 bytes.

3a (8pts): What will the output of the program be? (what will the `printf` in `clobber_me` output?).

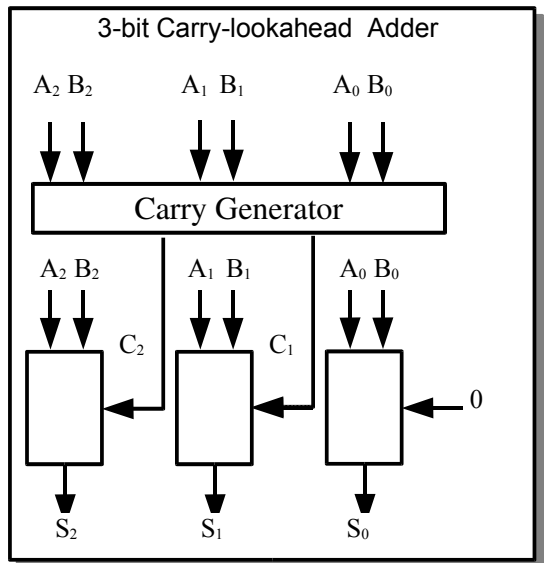
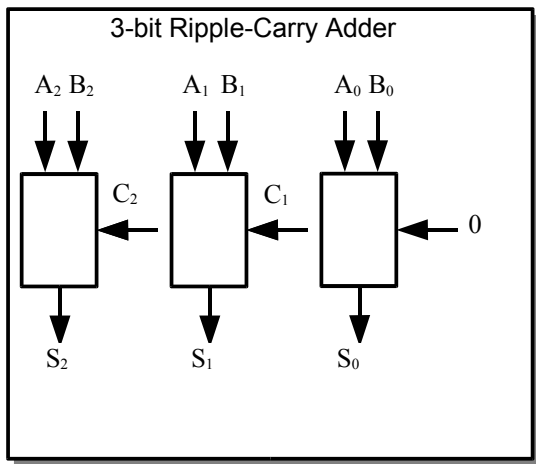
The questions below deal with the state of the registers and memory after the `leave` instruction has completed in the function `clobber_me` (`%eip` is `0x8048411`). Show the value of the following registers and memory locations as hex values:

3d (2pts): `buf[0]` (single byte): _____

3d (5pts): `%ebp` (4 bytes): _____

3d (5pts): To what address will the function return ? (4 bytes) _____

Question 4 (15 pts): A *carry*-lookahead adder computes the value of carry bits using dedicated combinational circuits so that the carry bits to each 1-bit adder are available as soon as possible. The diagram on the left shows a 3-bit ripple-carry adder. Your job is to design a carry-generator circuit that computes the first two carry bits, labeled C_1 and C_2 , from the inputs A_0 , A_1 , B_0 and B_1 . You can describe your circuits (there are two – one for C_1 and one for C_2) using truth tables, a circuit diagram (AND/OR/NOT GATES), with HCL expressions, or with boolean algebra.



Describe your Carry Generator functions here:

Question 5 (25 pts): The following C function returns the length of a string:

```
int mystrlen(char *s) {
    if (*s==0)
        return 0;
    else
        return(1 + mystrlen(s+1));
}
```

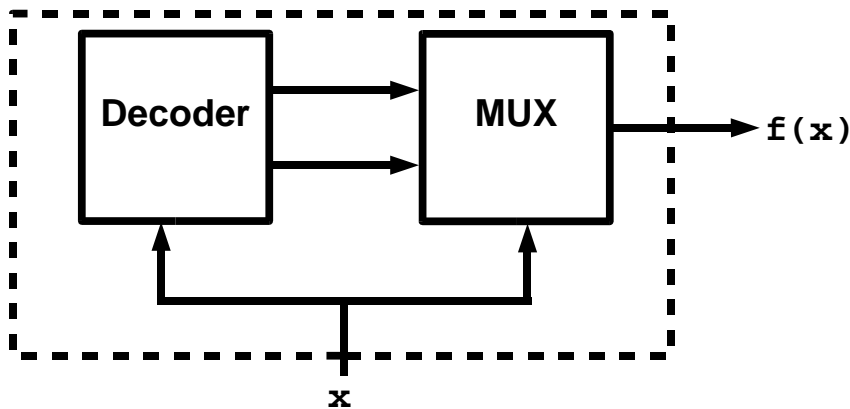
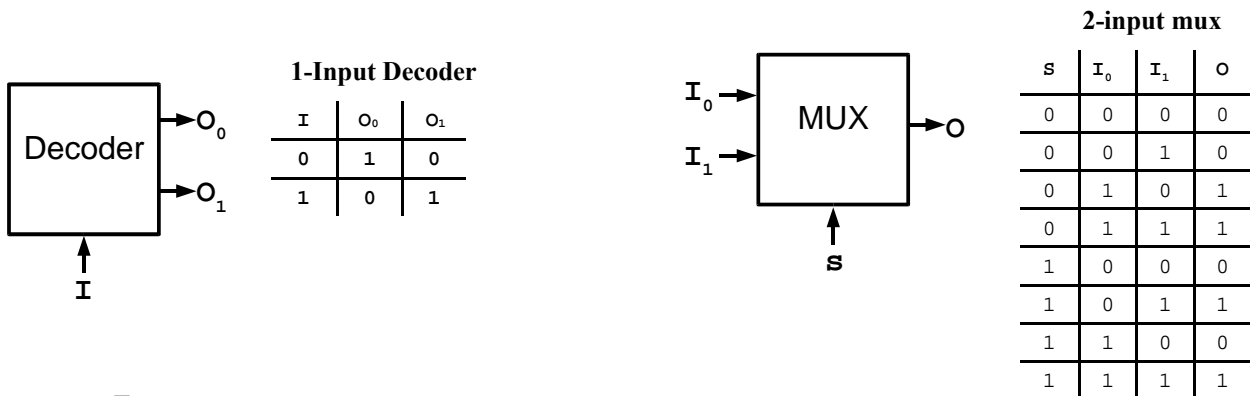
Translate this C code to an IA32 assembly language subroutine that follows the GCC calling convention and register usage conventions. Recall that %ebx, %esi and %edi are *callee-save* registers.

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

Question 6a (4 pts): You have lots of AND gates and lots of NOT gates, but you need an OR gate. Design an OR gate in the space below using only AND and NOT gates.



Question 6b (3 pts): This question involves a circuit that includes a 1-input decoder and a 2-input multiplexer. The truth tables for a 1-input decoder and a 2-input mux are shown for reference. What is the value of the output of the circuit show below (the output is labeled $f(x)$)?



Some IA32 Instructions

| | |
|-------------------------------|---------------------------------|
| <code>movl Src, Dest</code> | <i>Dest = Src</i> |
| <code>addl Src, Dest</code> | <i>Dest = Dest + Src</i> |
| <code>subl Src, Dest</code> | <i>Dest = Dest - Src</i> |
| <code>imull Src, Dest</code> | <i>Dest = Dest * Src</i> |
| <code>sall Src, Dest</code> | <i>Dest = Dest << Src</i> |
| <code>sarl Src, Dest</code> | <i>Dest = Dest >> Src</i> |
| <code>shrl Src, Dest</code> | <i>Dest = Dest >> Src</i> |
| <code>xorl Src, Dest</code> | <i>Dest = Dest ^ Src</i> |
| <code>andl Src, Dest</code> | <i>Dest = Dest & Src</i> |
| <code>orl Src, Dest</code> | <i>Dest = Dest Src</i> |
| <code>incl Dest</code> | <i>Dest = Dest + 1</i> |
| <code>decl Dest</code> | <i>Dest = Dest - 1</i> |
| <code>negl Dest</code> | <i>Dest = - Dest</i> |
| <code>notl Dest</code> | <i>Dest = ~ Dest</i> |
| <code>leal Src, Dest</code> | <i>Dest = address of Src</i> |
| <code>cmpl Src2, Src1</code> | <i>Sets CCs Src1 Src2</i> |
| <code>testl Src2, Src1</code> | <i>Sets CCs Src1 & Src2</i> |

| | |
|-------------------------|--|
| <code>jmp label</code> | <i>jump</i> |
| <code>je label</code> | <i>jump equal</i> |
| <code>jne label</code> | <i>jump not equal</i> |
| <code>js label</code> | <i>jump negative</i> |
| <code>jns label</code> | <i>jump non-negative</i> |
| <code>jg label</code> | <i>jump greater (signed)</i> |
| <code>jge label</code> | <i>jump greater or equal (signed)</i> |
| <code>jle label</code> | <i>jump less (signed)</i> |
| <code>jlt label</code> | <i>jump less or equal (signed)</i> |
| <code>ja label</code> | <i>jump above (unsigned)</i> |
| <code>jb label</code> | <i>jump below (unsigned)</i> |
| <code>push Src</code> | <i>Mem[%esp-4] = Src, %esp=%esp-4</i> |
| <code>pop Dest</code> | <i>Dest = Mem[%esp], %esp=%esp+4</i> |
| <code>call label</code> | <i>Mem[%esp-4] = %eip, %esp=%esp-4, %eip = label</i> |
| <code>ret</code> | <i>%eip = mem[%esp], %esp=%esp+4</i> |

IA32 Addressing Modes

| | | |
|--------------|--------------|--|
| Immediate | \$val | value |
| Normal | (R) | Mem[Reg[R]] |
| | | -Register R specifies memory address |
| | | <code>movl (%ecx), %eax</code> |
| Displacement | D(R) | Mem[Reg[R]+D] |
| | | -Register R specifies start of memory region |
| | | -Constant displacement D specifies offset |
| | | <code>movl 8(%ebp), %edx</code> |
| Indexed | D(Rb, Ri, S) | Mem[Reg[Rb]+S*Reg[Ri]+ D] |
| | | -D: Constant "displacement" 1, 2, or 4 bytes |
| | | -Rb: Base register: Any of 8 integer registers |
| | | -Ri: Index register: |
| | | -Scale: 1, 2, 4, or 8 |
| | | <code>movl -4(%ebx, %esi, 4), %edx</code> |

| Question | Possible | Score |
|----------|----------|-------|
| 1 | 18 | |
| 2 | 15 | |
| 3 | 20 | |
| 4 | 15 | |
| 5 | 25 | |
| 6 | 7 | |
| Total | 100 | |

IA32 Registers

| | | | | | | | |
|-------------------|-------------------|-------------------|-------------------|-------------------|-------------------|-------------------|-------------------|
| <code>%eax</code> | <code>%ebx</code> | <code>%ecx</code> | <code>%edx</code> | <code>%esi</code> | <code>%edi</code> | <code>%ebp</code> | <code>%esp</code> |
|-------------------|-------------------|-------------------|-------------------|-------------------|-------------------|-------------------|-------------------|