Instructions:

- This is a closed-book, closed-notes, closed-neighbor exam.
- Only a writing utensil is needed for this exam.
- No calculators allowed.
- If you need to go to the bathroom, get a drink or have a snack, do so prior to the start of the exam; you will not be allowed to leave the room until you are finished with your exam (i.e. once you leave, you are done); eating during the exam is prohibited.
- All cell phones and electronic equipment must be silenced and stored until after you leave the exam room.
- This is a timed exam. You have the university designated 1 hour 45 minute time frame with a start and end time as specified by the university and listed on your syllabus since the first day of the term. When time is called, you must stop writing immediately.
- Once the exam starts, and until every exam has been turned in, you will not talk with anyone in the classroom.
- In accordance with The Ohio State University Code of Student Conduct, I certify that I have neither received nor given aid on this examination, that I shall not discuss the contents of this examination with anyone in CSE 2421 who has not already taken the exam, and that I have not written down and taken from the room any questions or answers from this exam.
- You must read these instructions and sign your name below for this exam to be graded.
- Failure to follow any of these instructions will result in point penalization, up to and including a score of zero on the exam, to be determined by the instructor.

Signature ____________________________________________________________
MULTIPLE CHOICE (8 pts) – Fill in the blank (do NOT circle the letter!)

________ IA32 and Y86 have the following in common EXCEPT:
   A. No memory to memory transfer with a single instruction
   B. 3 types of operands: immediate, register and memory
   C. Memory addressing including absolute, indirect, base+displacement, and indexed.
   D. Single-bit condition code registers.

________ The IA-32 “push” instruction:
   A. adds 4 to the stack pointer then stores a value onto the stack
   B. stores a value onto the stack then adds 4 to the stack pointer
   C. subtracts 4 from the stack pointer then stores a value onto the stack
   D. stores a value onto the stack then subtracts 4 from the stack pointer

________ The IA32 instruction, movl %esp, %ebp, can do all of the following at some point EXCEPT:
   A. moves the old base pointer onto the stack
   B. sets the stack pointer and the base pointer to the same location
   C. sets the base pointer to point to the bottom of the stack
   D. sets the starting point of a new frame on the stack

________ The condition codes that result from adding two single-byte (i.e. signed char) values 0xAC and 0x8A on an IA32 machine are:
   A. ZF = 1, SF = 0, CF = 0, OF = 1
   B. ZF = 0, SF = 0, CF = 1, OF = 1
   C. ZF = 0, SF = 1, CF = 0, OF = 0
   D. ZF = 1, SF = 0, CF = 1, OF = 1

________ The IA32 instructions jl and cmovl both have the same option to execute when certain condition codes are set, which is:
   A. D  ZF
   B. D  ~SF
   C. D  SF ^ OF
   D. D  ~(SF ^ OF) & ~ZF

________ IA-32 implements a jump table for which of the following C constructs:
   A. For loop
   B. IF structure
   C. Switch statement
   D. None of the above

________ The stack is used to:
   A. Pass function arguments
   B. Store local variables
   C. Save registers for later restoration
   D. All of the above

________ The offset of each structure member is determined at:
   A. Compile time
   B. Assembly time
   C. Link time
   D. Execution
TRUE/FALSE (8 pts) - Circle T or F for each statement

T  F  The machine level program generated has no information about data types.
T  F  A byte is the smallest addressable unit of memory.
T  F  A disassembler is a tool that determines the instruction sequence represented by an executable program.
T  F  Both IA32 and Y86 call statements/instructions push the return address onto the stack.
T  F  Both IA32 and Y86 are considered load/store architectures.
T  F  Relocatable object files do not yet have a starting address in memory.
T  F  The IA32 cmpl instruction executes the same as the Y86 subl instruction.
T  F  Conditional move statements in Y86 execute the movement operations according to condition codes.

(4 pts) Determine if the following descriptions are generally considered true for CISC or RISC instruction sets by putting a ‘C’ for CISC and an ‘R’ for RISC.

________  Fewer, simpler instructions  (R)
________  Arithmetic instructions can access memory  (C)
________  Variable length instructions  (C)
________  Stack intensive procedure/function linkages  (C)

SHORT ANSWER

(8 pts) On a Little Endian machine that is byte addressable, show how the following values are stored in memory. FYI: ‘A’ is 0x41 (in hex). NOTE: Only fill in the boxes that apply. That is, if you do not fill in the memory location, i.e. the value does not use the memory location, then be sure to leave it blank. Assume the starting memory location for value to be stored is 0x100.

<table>
<thead>
<tr>
<th>Value</th>
<th>0x100</th>
<th>0x101</th>
<th>0x102</th>
<th>0x103</th>
</tr>
</thead>
<tbody>
<tr>
<td>“ABC”</td>
<td>41</td>
<td>42</td>
<td>43</td>
<td>\0 or empty</td>
</tr>
<tr>
<td>0x1234</td>
<td>34</td>
<td>12</td>
<td>empty</td>
<td>empty</td>
</tr>
</tbody>
</table>

(4 pts) Determine the pointer generated to point to the structure member a[2] in the given structure where the start of the structure is designated by address R, the size of an integer and a pointer are both 4 bytes, and IDX designates the subscript value (in this case, the 2 in a[2]).

```c
struct rec     {
    int i;
    int a[3];
    int *p;   }                         

____________________________________  r+4+4*idx
```
(4 pts) Given the signed char value 0xAF, determine the binary byte value (i.e. not hex) for this byte in memory only, which remains after *each* of the following actions (start with the given value for each):

- Shift the value 0xAF left by 3
- Shift the value 0xAF right arithmetically by 2

(10 pts) Given the following initial values and IA32 instructions, use these initial values for each question given below (i.e. each question is based on the initial values, not any changes the previous question might or might not implement).

<table>
<thead>
<tr>
<th>Address</th>
<th>Value</th>
<th>Register</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x100</td>
<td>0xFF</td>
<td>%eax</td>
<td>0x100</td>
</tr>
<tr>
<td>0x104</td>
<td>0xAB</td>
<td>%ecx</td>
<td>0x1</td>
</tr>
<tr>
<td>0x108</td>
<td>0x13</td>
<td>%edx</td>
<td>0x3</td>
</tr>
<tr>
<td>0x10C</td>
<td>0x11</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

A. movl 0x100, %eax  What is stored in %eax? _________________ 0xFF
B. movl (%eax,%edx,4), %ecx  What is stored in %ecx? _________________ 0x11
C. decl %ecx  What is stored in %ecx? _________________ 0x0
D. subl %edx, 4(%eax)  Which “value” is changed? Be sure to designate the appropriate destination address that is changed and what value is stored there.

Destination address is _________________ to value _________________ 0x104/0xA8

(6 pts) Give the IA-32 instruction format encoding (machine code) in hex notation for the following instructions where the opcode for part A is 0x8B, the opcode for part B is 0x8D, and the opcode for part C is 0x89:

A. mov 0x8(ebp), %ecx  _________________ 8B 4D 08
B. lea 0x4(%ecx,%eax,1),%eax  _________________ 8D 44 01 04
C. mov %eax, (%ecx)  _________________ 89 01

(4 pts) Why doesn’t Y86 have a CF flag? (do not use more than the line given for your answer)

____________________________________________________________________________

Y86 does not have an unsigned option; Y86 is signed only; CF is for unsigned arithmetic

(8 pts) The following function is called by function domath:

```c
int arith(int x, int y, int z)
```

Assume that %ebp(1) to %esp to the left of the actual stack is the stack frame for the domath function. Fill in the 4 locations on the stack between the ... and %ebp(1)
CODE EVALUATION

(4 pts) Given the following C code, fill in the blanks to create equivalent IA-32 code:

```c
int max(int x, int y)
{
    if (x > y)
        return x;
    else
        return y;
}
```

```assembly
movl 8(%ebp), %edx                // where edx = x
movl 12(%ebp), %eax             // where eax = y
___________ %eax, %edx           // fill in blank
cmpl_________  L9                // fill in blank
jle
movl %edx, %eax
L9:                        // done
```

(12 pts) Given the following C code, fill in the blanks to create equivalent IA-32 code:

```c
int fact_while(int x)
{
    int result = 1;
    while (x > 1)
    {
        result *= x;
        x = x - 1;
    }
    return result;
}
```

```assembly
fact_while:
    pushl %ebp
    movl %esp, %ebp
    movl 8(%ebp), %edx
    movl $1, %eax
    cmpl $1, %edx
    jle .L3
    .L6:
    imull %edx, %eax
    subl $1, %edx
    jne .L6
    .L3:
    popl %ebp
    ret
```
(12 pts) Given the following C code, fill in the blanks to create equivalent Y86 code:

```c
// compute the sum of an
// integer array
int Sum(int *Start, int Count)
{
    int sum = 0;
    while (Count)
    {
        sum += *Start;
        Start++;  
        Count--;
    }
    return sum;
}
```

```y86
Sum:
pushl %ebp
    rrmovl %esp, %ebp
    mrmovl 8(%ebp), %ecx
    mrmovl 12(%ebp), %edx
    xorl %eax, %eax
    also subl %edx, %edx
    je   End
Loop:
    irmovl (%ecx), %esi
    addl %esi, %eax
    irmovl $4, %ebx
    addl %ebx, %ecx
    irmovl $-1, %ebx
    addl %ebx, %edx
    jne  Loop
End:
    rrmovl %ebp, %esp
    popl %ebp
    ret
```

(8 pts) Convert the given leal instruction to an equivalent group of IA32 assembly instructions. Make sure that when the converted code is complete, both the leal instruction and the converted code have the same values in registers %eax, %ecx and %ebx. You cannot use any other registers besides the 3 given in the leal instruction. NOTE: any instruction you need here has an example in the above IA32 code evaluation questions.

**GIVEN:**
leal 0x4(%eax, %ecx, 8), %ebx

```y86
GIVEN:
    movl %ecx, %ebx
    imull $8, %ebx
    addl %eax, %ebx
    addl $4, %ebx

    *** could switch first two
    movl $8, %ebx
    imull %ecx, %ebx

    *** could switch last two
    addl $4, %ebx
    addl %eax, %ebx
```
# Y86 Instruction Set

<table>
<thead>
<tr>
<th>Byte</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
</tr>
</thead>
<tbody>
<tr>
<td>halt</td>
<td>0</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>nop</td>
<td>1</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>rrmovl rA, rB</td>
<td>2</td>
<td>0</td>
<td>rA</td>
<td>rB</td>
<td></td>
<td></td>
</tr>
<tr>
<td>irmovl V, rB</td>
<td>3</td>
<td>0</td>
<td>F</td>
<td>rB</td>
<td>V</td>
<td></td>
</tr>
<tr>
<td>rrmovl rA, D(rB)</td>
<td>4</td>
<td>0</td>
<td>rA</td>
<td>rB</td>
<td>D</td>
<td></td>
</tr>
<tr>
<td>mmovl D(rB), rA</td>
<td>5</td>
<td>0</td>
<td>rA</td>
<td>rB</td>
<td>D</td>
<td></td>
</tr>
</tbody>
</table>

## Operations

- **addl**
  - 6 0
- **subl**
  - 6 1
- **andl**
  - 6 2
- **xorl**
  - 6 3

## Branches

<table>
<thead>
<tr>
<th>Branch</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
</tr>
</thead>
<tbody>
<tr>
<td>jmp</td>
<td>7</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>jle</td>
<td>7</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>jl</td>
<td>7</td>
<td>2</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>je</td>
<td>7</td>
<td>3</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

## Moves

<table>
<thead>
<tr>
<th>Move</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
</tr>
</thead>
<tbody>
<tr>
<td>rrmovl</td>
<td>2</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>cmovnle</td>
<td>2</td>
<td>4</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>cmovle</td>
<td>2</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>cmovge</td>
<td>2</td>
<td>5</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>cmovl</td>
<td>2</td>
<td>2</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>cmove</td>
<td>2</td>
<td>3</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
### IA32 integer registers

Fast-access locations that the CPU uses, rather than storing all variables in memory (starts with “e” for extended)

<table>
<thead>
<tr>
<th>ORIGIN</th>
<th>31</th>
<th>ID#</th>
<th>15</th>
<th>8</th>
<th>7</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>accumulator</td>
<td></td>
<td>%eax</td>
<td>%ax</td>
<td>%ah</td>
<td>%al</td>
<td></td>
</tr>
<tr>
<td>counter</td>
<td></td>
<td>%ecx</td>
<td>%cx</td>
<td>%ch</td>
<td>%cl</td>
<td></td>
</tr>
<tr>
<td>data</td>
<td></td>
<td>%edx</td>
<td>%dx</td>
<td>%dh</td>
<td>%dl</td>
<td></td>
</tr>
<tr>
<td>base</td>
<td></td>
<td>%ebx</td>
<td>%bx</td>
<td>%bh</td>
<td>%bl</td>
<td></td>
</tr>
<tr>
<td>source index</td>
<td></td>
<td>%esi</td>
<td>%si</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>destination index</td>
<td></td>
<td>%edi</td>
<td>%di</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>STACK POINTER</td>
<td></td>
<td>%esp</td>
<td>%sp</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>BASE/FRAME POINTER</td>
<td></td>
<td>%ebp</td>
<td>%bp</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

ID=8 is “no register”

THE VIRTUAL REGISTERS DO NOT HAVE AN ADDRESS ASSOCIATED WITH THEM

16-bit virtual registers (backward compatibility)
All IA-32 instruction encodings

Are subsets of the general instruction format shown below, in the given order

- Instructions consist of:
  - optional instruction prefixes (in any order)
  - 1-3 opcode bytes – determines the action of the statement
  - an addressing-form specifier (if required) consisting of:
    - the ModR/M byte - addressing modes register/memory
    - sometimes the SIB (Scale-Index-Base) byte
    - a displacement (if required)
    - an immediate data field (if required).

<table>
<thead>
<tr>
<th>Instruction Prefixes</th>
<th>Opcode</th>
<th>ModR/M</th>
<th>SIB</th>
<th>Displacement</th>
<th>Immediate</th>
</tr>
</thead>
<tbody>
<tr>
<td>Up to four prefixes</td>
<td>1-, 2-, 3-byte opcode</td>
<td>1 byte</td>
<td>1 byte</td>
<td>Address</td>
<td>Immediate</td>
</tr>
<tr>
<td>of 1 byte each (optional)</td>
<td></td>
<td>(if required)</td>
<td>(if required)</td>
<td>of 1, 2, or 4 bytes or none</td>
<td>of 1, 2, or 4 bytes or none</td>
</tr>
</tbody>
</table>

- Mod=00
  - First operand a register, specified by Reg #
  - Second operand in memory; address stored in a register numbered by R/M.
    - That is, Memory[Reg[R/M]]
  - Exceptions:
    - R/M=100 (SP): SIB needed
    - R/M=101 (BP): disp32 needed
- Mod=01, same as Mod 00 with 8-bit displacement.
  - Second operand: Memory[disp8+Reg[R/M]].
  - Exception: SIB needed when R/M=100
- Mod=10, same as Mod 01 with 32-bit displacement
- Mod=11
  - Second operand is also a register, numbered by R/M.
  - Do not confuse displacement width with data width.
    - Data width is specified by the opcode.
    - For example, the use of disp8 does not imply 8-bit data.

*** For some opcodes, the reg# is used as an extension of the opcode.

- Specify how a memory address is calculated
  - Address = Reg[base] + Reg[Index] * 2^{scale}
- Exceptions:
  - SP cannot be an index, and
  - BP cannot be a base.