0) Compare zero-, one-, two-, and three-address machines by writing programs to compute

\[ X = \frac{A + B}{D - E \times B} \]

for each of the four machines. The instructions available for use are as follows:

<table>
<thead>
<tr>
<th>3 Address</th>
<th>2 Address</th>
<th>1 Address (Accumulator machine)</th>
<th>0 Address (Stack machine)</th>
</tr>
</thead>
<tbody>
<tr>
<td>MOVE (X ← Y)</td>
<td>MOVE (X ← Y)</td>
<td>LOAD M</td>
<td>PUSH M</td>
</tr>
<tr>
<td>ADD (X ← Y + Z)</td>
<td>ADD (X ← X + Y)</td>
<td>ADD M</td>
<td>ADD</td>
</tr>
<tr>
<td>SUB (X ← Y - Z)</td>
<td>SUB (X ← X - Y)</td>
<td>SUB M</td>
<td>SUB</td>
</tr>
<tr>
<td>MUL (X ← Y * Z)</td>
<td>MUL (X ← X * Y)</td>
<td>MUL M</td>
<td>MUL</td>
</tr>
<tr>
<td>DIV (X ← Y / Z)</td>
<td>DIV (X ← X / Y)</td>
<td>DIV M</td>
<td>DIV</td>
</tr>
</tbody>
</table>

Team #: ___________________________  Name: ___________________________
1) How relocatable (i.e., can it be moved in memory) is the code in memory if direct addressing is used?

2) How many bits are needed to represent a direct address on a 32-bit machine?

3) From your programming experience, what range of Integer values would cover 90% of the constant Integer values used in your programs?

4) How many binary bits would you need to represent this range of values?

5) What determines how many bits are needed to represent a register in a machine language instruction?

6) Which of the following programming language constructs would be good candidates for using PC-relative addressing?
   a) conditional branch used when implementing loops
   b) calling a subprogram
   c) accessing a global variable
   d) accessing a local variable
### Type of Instruction

#### Memory Access (Load and Store)
- `lw $4, Mem`  
  MIPS Assembly Language: `lw $4, Mem`  
  Register Transfer Language: [Mem]  
  Description: Load Memory
- `sw $4, Mem`  
  MIPS Assembly Language: `sw $4, Mem`  
  Register Transfer Language: $4← Mem  
  Description: Store Memory
- `lw $4, 16($3)`  
  MIPS Assembly Language: `lw $4, 16($3)`  
  Register Transfer Language: $4← [Mem at address in $3 + 16]  
  Description: Load Memory
- `sw $4, Mem`  
  MIPS Assembly Language: `sw $4, Mem`  
  Register Transfer Language: [Mem at address in $3 + 16]← $4  
  Description: Store Memory

#### Move
- `move $4, $2`  
  MIPS Assembly Language: `move $4, $2`  
  Register Transfer Language: $4← $2  
  Description: Move
- `li $4, 100`  
  MIPS Assembly Language: `li $4, 100`  
  Register Transfer Language: $4← 100  
  Description: Immediate

#### Load Address
- `la $5, mem`  
  MIPS Assembly Language: `la $5, mem`  
  Register Transfer Language: $4← load address of mem  
  Description: Load Address

#### Arithmetic Instruction (reg. operands only)
- `add $4, $2, $3`  
  MIPS Assembly Language: `add $4, $2, $3`  
  Register Transfer Language: $4← $2 + $3  
  Description: Addition
- `mul $10, $12, $8`  
  MIPS Assembly Language: `mul $10, $12, $8`  
  Register Transfer Language: $10← $12 * $8  
  Description: Multiplication
- `sub $4, $2, $3`  
  MIPS Assembly Language: `sub $4, $2, $3`  
  Register Transfer Language: $4← $2 - $3  
  Description: Subtraction

#### Arithmetic with Immediates (last operand must be an integer)
- `addi $4, $2, 100`  
  MIPS Assembly Language: `addi $4, $2, 100`  
  Register Transfer Language: $4← $2 + 100  
  Description: Add Immediate
- `mul $4, $2, 100`  
  MIPS Assembly Language: `mul $4, $2, 100`  
  Register Transfer Language: $4← $2 * 100  
  Description: Multiply Immediate

#### Conditional Branch
- `bgt $4, $2, LABEL`  
  MIPS Assembly Language: `bgt $4, $2, LABEL`  
  Register Transfer Language: Branch to LABEL if $4 > $2  
  Description: Branch if Greater Than
- `beq $4, $2, LABEL`  
  MIPS Assembly Language: `beq $4, $2, LABEL`  
  Register Transfer Language: Branch to LABEL if $4 = $2  
  Description: Branch if Equal
- `bne $4, $2, LABEL`  
  MIPS Assembly Language: `bne $4, $2, LABEL`  
  Register Transfer Language: Branch to LABEL if $4 ≠ $2  
  Description: Branch if Not Equal

#### Unconditional Branch
- `j LABEL`  
  MIPS Assembly Language: `j LABEL`  
  Register Transfer Language: Always Branch to LABEL  
  Description: Jump to LABEL

### MIPS Logical Instructions
- `and $4, $5, $6`  
  MIPS Assembly Language: `and $4, $5, $6`  
  Register Transfer Language: $4←$5 (bit-wise AND) $6  
  Description: Logical AND
- `andi $4, $5, 0x5f`  
  MIPS Assembly Language: `andi $4, $5, 0x5f`  
  Register Transfer Language: $4←$5 (bit-wise AND) 5f_{16}  
  Description: Logical AND with Immediate
- `or $4, $5, $6`  
  MIPS Assembly Language: `or $4, $5, $6`  
  Register Transfer Language: $4←$5 (bit-wise OR) $6  
  Description: Logical OR
- `ori $4, $5, 0x5f`  
  MIPS Assembly Language: `ori $4, $5, 0x5f`  
  Register Transfer Language: $4←$5 (bit-wise OR) 5f_{16}  
  Description: Logical OR with Immediate
- `xor $4, $5, $6`  
  MIPS Assembly Language: `xor $4, $5, $6`  
  Register Transfer Language: $4←$5 (bit-wise Exclusive-OR) $6  
  Description: Logical Exclusive-OR
- `xori $4, $5, 0x5f`  
  MIPS Assembly Language: `xori $4, $5, 0x5f`  
  Register Transfer Language: $4←$5 (bit-wise Exclusive-OR) 5f_{16}  
  Description: Logical Exclusive-OR with Immediate
- `nor $4, $5, $6`  
  MIPS Assembly Language: `nor $4, $5, $6`  
  Register Transfer Language: $4←$5 (bit-wise Exclusive-OR) $6  
  Description: Logical Exclusive-OR
- `not $4, $5`  
  MIPS Assembly Language: `not $4, $5`  
  Register Transfer Language: $4←$5 (bit-wise NOT) $6  
  Description: Logical NOT
- `nori $4, $5, 0x5f`  
  MIPS Assembly Language: `nori $4, $5, 0x5f`  
  Register Transfer Language: $4←$5 (bit-wise NOT) 5f_{16}  
  Description: Logical NOT with Immediate
- `not $4, $5`  
  MIPS Assembly Language: `not $4, $5`  
  Register Transfer Language: $4←$5 (bit-wise NOT) $6  
  Description: Logical NOT

### MIPS Shift and Rotate Instructions
- `sll $4, $5, 3`  
  MIPS Assembly Language: `sll $4, $5, 3`  
  Register Transfer Language: $4←shift left $5 by 3 positions. Shift in zeros (only least significant 5-bits of immediate value are used to shift)  
  Description: Logical Left Shift
- `sllv $4, $5, $6`  
  MIPS Assembly Language: `sllv $4, $5, $6`  
  Register Transfer Language: Similar to sll, but least significant 5-bits of $6 determine the amount to shift.  
  Description: Logical Left Shift with Immediate
- `srl $4, $5, 3`  
  MIPS Assembly Language: `srl $4, $5, 3`  
  Register Transfer Language: $4←shift right $5 by 3 positions. Shift in zeros  
  Description: Logical Right Shift
- `srlv $4, $5, $6`  
  MIPS Assembly Language: `srlv $4, $5, $6`  
  Register Transfer Language: Similar to srl, but least significant 5-bits of $6 determine the amount to shift.  
  Description: Logical Right Shift with Immediate
- `sra $4, $5, 3`  
  MIPS Assembly Language: `sra $4, $5, 3`  
  Register Transfer Language: $4←shift right $5 by 3 positions. Sign-extend (shift in sign bit)  
  Description: Logical Right Shift with Sign Extend
- `srav $4, $5, $6`  
  MIPS Assembly Language: `srav $4, $5, $6`  
  Register Transfer Language: Similar to sra, but least significant 5-bits of $6 determine the amount to shift.  
  Description: Logical Right Shift with Sign Extend
- `rol $4, $5, 3`  
  MIPS Assembly Language: `rol $4, $5, 3`  
  Register Transfer Language: $4←rotate left $5 by 3 positions  
  Description: Logical Rotate Left
- `rol $4, $5, $6`  
  MIPS Assembly Language: `rol $4, $5, $6`  
  Register Transfer Language: Similar to above, but least significant 5-bits of $6 determine the amount to rotate.  
  Description: Logical Rotate Left with Immediate
- `ror $4, $5, 3`  
  MIPS Assembly Language: `ror $4, $5, 3`  
  Register Transfer Language: $4←rotate right $5 by 3 positions  
  Description: Logical Rotate Right
- `ror $4, $5, $6`  
  MIPS Assembly Language: `ror $4, $5, $6`  
  Register Transfer Language: Similar to above, but least significant 5-bits of $6 determine the amount to rotate.  
  Description: Logical Rotate Right with Immediate
A high-level language program to calculate the $n$th fibonacci number would be:

A complete assembly language MIPS program to calculate the $n$th fibonacci number.

A Trace of Program (time $\rightarrow$):