Below is the description of the hardware features of a desktop PC:

- Intel® 2nd Generation Core™ i3 (3.4GHz, L2 Cache Memory 3MB)
- System Memory: (RAM) 8GB DDR3 SDRAM
- Hard Drive: 1.5TB SATA (7200 rpm)
- Intel® HD Graphics 2000 (Video Memory Up to 4GB shared)
- Network Card: Built-in 10/100/1000Base-T Ethernet LAN
- Wireless Networking: Built-in 802.11b/g/n wireless LAN
- Recordable DVD Drive: 8x DVD+R DL; 8x DVD-R DL; 16x8x16 DVD+RW; 16x6x16 DVD-RW; 5x DVD-RAM; 40x24x40 CD-RW
- Digital Media Card Reader
- Available Expansion Bays: External: 1 (3.5"), 1 (5.25"); Internal: 1 (3.5"")
- Available Expansion Slots: 1 PCI Express x1, 1 PCI Express x16
- USB 2.0 Ports: 2 USB 3.0 (front); 4 USB 2.0 (rear)
- Wireless Keyboard with volume control
- Wireless optical mouse
- 20” widescreen flat panel monitor
- Operating System Windows 8

1) What does the processor do?

2) What is stored in main memory (RAM)?

3) What is stored on the hard disk?

4) What is the purpose of cache memory?

5) What terms relate to interconnection of internal PC components?

6) What terms relate to interconnection of external PC components?

7) What is a KB, MB, GB, Gigabit, MHz, GHz?

8) What is the role of the operating system?
Processing (Instruction/Machine) Cycle of stored-program computer - repeat all day

1. Fetch Instruction - read instruction pointed at by the program counter (PC) from memory into Instruction Reg. (IR)
2. Decode Instruction - figure out what kind of instruction was read
3. Fetch Operands - get operand values from the memory or registers
4. Execute Instruction - do some operation with the operands to get some result
5. Write Result - put the result into a register or in a memory location

(Note: Sometime during the above steps, the PC is updated to point to the next instruction.)
9) What is the role of a compiler?

10) What advantages do high-level languages (Ada, C, C++, Java, Python, etc.) have over assembly language?

11) Why do people program in assembly language (AL)?
### Type of Instruction | MIPS Assembly Language | Register Transfer Language Description
--- | --- | ---
Memory Access (Load and Store) | lw $4, Mem | $4 ← [Mem]
| sw $4, Mem | Mem ← $4 |
| lw $4, 16($3) | $4 ← [Mem at address in $3 + 16] |
| sw $4, Mem | [Mem at address in $3 + 16] ← $4 |
Move | move $4, $2 | $4 ← $2 |
| li $4, 100 | $4 ← 100 |
Load Address | la $5, mem | $4 ← load address of mem |
Arithmetic Instruction (reg. operands only) | add $4, $2, $3 | $4 ← $2 + $3 |
| mul $10, $12, $8 | $10 ← $12 * $8 (32-bit product) |
| sub $4, $2, $3 | $4 ← $2 - $3 |
Arithmetic with Immediates (last operand must be an integer) | addi $4, $2, 100 | $4 ← $2 + 100 |
| mul $4, $2, 100 | $4 ← $2 * 100 (32-bit product) |
Conditional Branch | bgt $4, $2, LABEL | Branch to LABEL if $4 > $2 |
(bge, blt, ble, beq, bne) |
Unconditional Branch | j LABEL | Always Branch to LABEL |

### Fibonacci Sequence:

<table>
<thead>
<tr>
<th>Position in Sequence</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
</tr>
</thead>
<tbody>
<tr>
<td>i</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>8</td>
</tr>
</tbody>
</table>

A high-level language program to calculate the $n$th fibonacci number would be:

```
temp2 = 0
temp3 = 1
for i = 2 to n do
    temp4 = temp2 + temp3
    temp2 = temp3
    temp3 = temp4
end for
result = temp4
```

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

```
.data
n:       .word 8     # variable in memory
result:  .word 0     # variable in memory

.text
.globl main
main:    li $2, 0     # $2 holds temp2
         li $3, 1     # $3 holds temp3
for_init: li $6, 2    # initialize i ($6) to 2
         lw $5, n     # load "n" into $5
for_loop: bgt $6, $5, end_for    # if $6 >= $5, then branch to end_for label
         add $4, $2, $3     # $4 holds temp4
         move $2, $3      # shift temp3 to temp2
         move $3, $4      # shift temp4 to temp3
         addi $6, $6, 1    # increment i ($6)
         j for_loop # unconditionally jump to for_loop label
end_for:  sw $4, result      # store the result to memory
         li $v0, 10      # system code for exit
         syscall        # call the operating system
```