Practice/Sample questions:

1. What do you understand by Instruction Set Architecture?

Instruction Set Architecture refers to instruction types, instruction format in low level as well as in machine code and detailed descriptions of operand fields (addressing modes of instructions). Instruction Set Architecture also defines:

- Operations that the processor can execute (add, sub, mult, …, how is it specified)
- Data Transfer mechanisms
- How to access data
- Number of operands (0, 1, 2, 3)
- Operand storage (where besides memory)
- Memory address (how is memory location specified)
- Type and size of operands (byte, int, float, …)
- Control Mechanisms (branch, jump, etc)

2. Discuss types/classification of processors based on Instruction set architecture.

Processors are divided into following classes based on instruction set architecture

Stack-based: The CPU is designed to read

Accumulator-based

Register-Register

Register-Memory

3. What is addressing mode? Discuss different types of addressing modes.

Operands in instructions may be indicated differently. For example, operands may be provided directly, operands may be indicated by CPU registers, operands may be indicated by memory addresses, assuming operands are saved in memory, operands may be saved in memory but memory addresses are initially loaded into registers and operands are indicated indirectly by registers in the operand fields.

How the operands are addressed/indicated in the operand fields, called addressing mode of instruction. It is not related to operation or opcode of the instruction.

Addressing modes may be classified in three broad categories:

- Immediate addressing mode
- Register addressing mode
- Memory addressing mode.

If operands or one of the operands is provided in operands/one of the operand fields, called immediate addressing mode. Example:

Opcode	Result field	Data-1	Data-2

..

Opcode	Result field	Data-1	Register

If all operands are indicated by CPU registers, called register addressing mode.

Opcode	Result field	Register for Data-1	Register for Data-2

..

Assume that, all or one of the operands are saved in memory. In an instruction, if operands or one of the operands is indicated by memory address in operand fields/one of the operand fields, called memory addressing mode. Example:

Opcode	Memory address for Result field	Memory address for Data-1	Memory address for Data-2

..

Opcode	Result field	Memory address for Data-1	Register

If memory address is 16 bits or less, the address is directly provided in the operand field, called direct memory addressing mode. Just to differentiate from data in immediate mode, memory address is either enclosed within bracket or different opcode is used.

Opcode	Result field	[Memory address for Data-1]	Register

..

Different Opcode	Result field	Memory address for Data-1	Register

For large memory addresses (20 bits or more), memory addresses are initially loaded into some special registers, called base or pointer or index or special purpose registers. In instructions,

operand or operands are indicated by one of those registers or register instead. This addressing mode is called register indirect memory addressing mode.

Opcode	Register pointing Memory address for Result field	Register pointing Memory address for Data-1	Register pointing Memory address for Data-2

..

Opcode	Result field	Register pointing Memory address for Data-1	Register

In such addressing modes, registers pointing memory addresses are marked differently, mostly enclosed within bracket '[ ]'.

If base register is used in one of the operand fields to point memory address, called indirect base register addressing mode.

If index register is used in one of the operand fields to point memory address, called indirect index register addressing mode.

A number (8 bits/16bits) is also used with base or index register. This number is added to the contents of base or index register to point memory location of operand, called relative base or relative index register addressing mode.

Opcode	Result field	Number[BaseRegister] pointing Memory address for Data-1	Register pointing Data-2

..

Opcode	Result field	Number[IndexRegister] pointing Memory address for Data-1	Register

To use structured data or arrays, more than one registers are used to point memory location of operand. The contents of these registers are simply added or content of one register is added to a pre-defined multiple (scaled) of the content of another register and memory address is calculated. In instructions, either both the registers are indicated within bracket and separated by comma of plus sign or one is indicated while other remains implicit. Depending on registers used, addressing mode is named. For example, in based-indexed addressing mode, one of the base and one of the index registers are used to point memory address of one of the operands.

Opcode	Result field	[baseReg+indexReg] pointing Memory address for Data-1	Register

Opcode	[baseReg+indexReg] pointing Memory address for Result field	Register	Register

Moreover, in some processor family, a number (8 bits or 16 bits) is also added to the contents of multiple registers (base and index registers for example) and memory address is calculated. In such instructions, number is also indicated with base and index register to indicate one of the operands, called relative base-indexed addressing mode.

Opcode	Result field	Number[baseReg+indexReg] pointing Memory address for Data-1	Register

Opcode	Number[baseReg+indexReg] pointing Memory address for Result field	Register	Register

Memory address may have following variants

- direct memory addressing mode
- indirect base register addressing mode
- indirect index register addressing mode
- relative base register indirect addressing mode
- relative index register indirect addressing mode

<table>
<tr><td></td><td>

based-indexed indirect register addressing mode
relative base-indexed indirect register addressing mode

</td></tr>
<tr><td>4.</td><td>
What do you understand by instruction format? Discuss different types of instruction formats of Instruction set architecture and CPU.
Following Instruction formats are used in different instruction set architecture and CPU design.

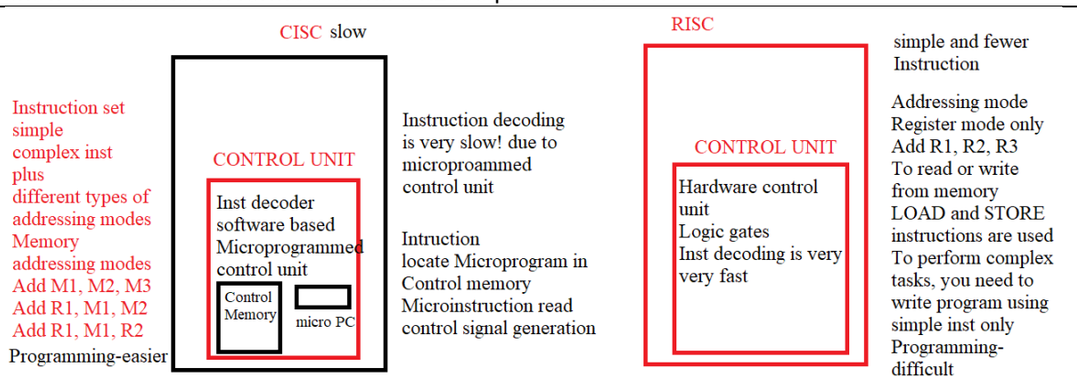
a) Three address formats: One destination and up to two operand sources per instruction
<table>
<tr><td>Opcode</td><td>Result field</td><td>Operand-1</td><td>Operand-2</td></tr>
</table>
..
<table>
<tr><td>Opcode</td><td>Operand-1</td><td>Operand-2</td><td>Result field</td></tr>
</table>

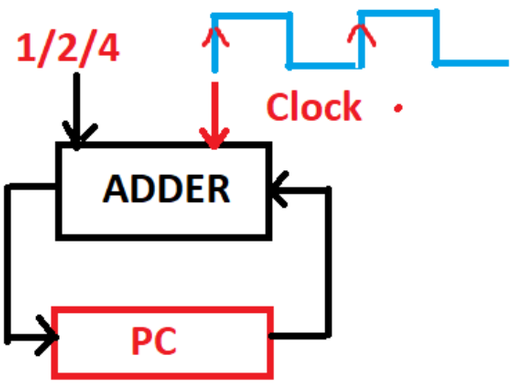
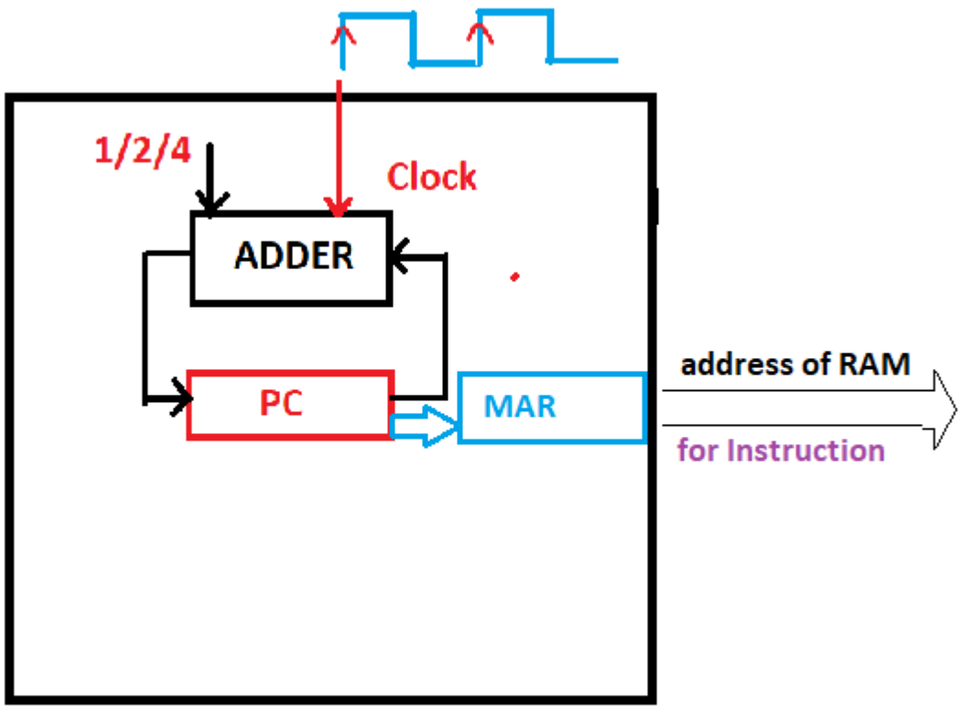
b) Two address formats: the destination is same as one of the operand sources
<table>
<tr><td>Opcode</td><td>Operand-1 (initially)
Result field (after operation)</td><td>Operand-2</td></tr>
</table>
…
<table>
<tr><td>Opcode</td><td>Operand-1</td><td>Operand-2 (initially)
Result field (after operation)</td></tr>
</table>

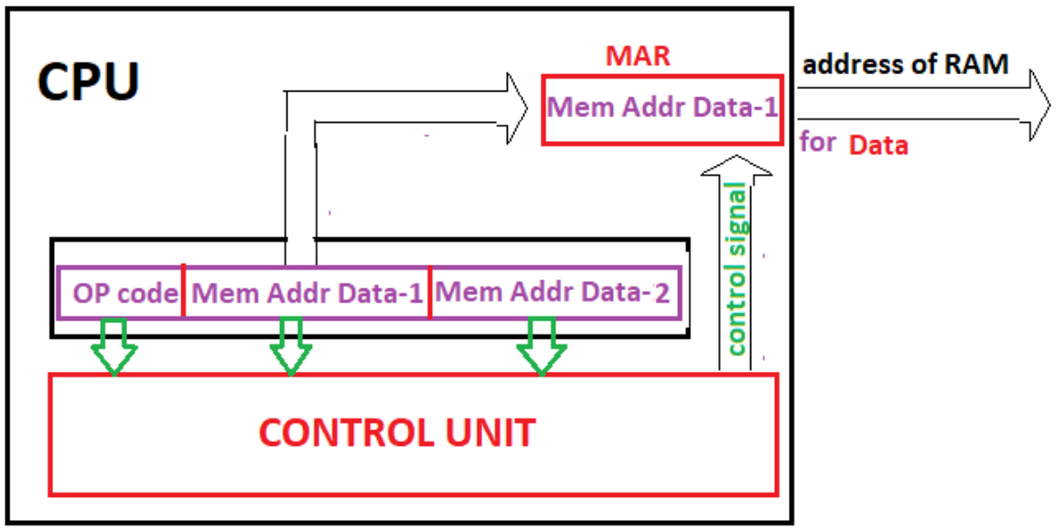
c) One address formats: Accumulator machines
<table>
<tr><td>Opcode</td><td>2nd operand</td></tr>
</table>
Here, accumulator is always other implicit operand and accumulator is implicit result field.

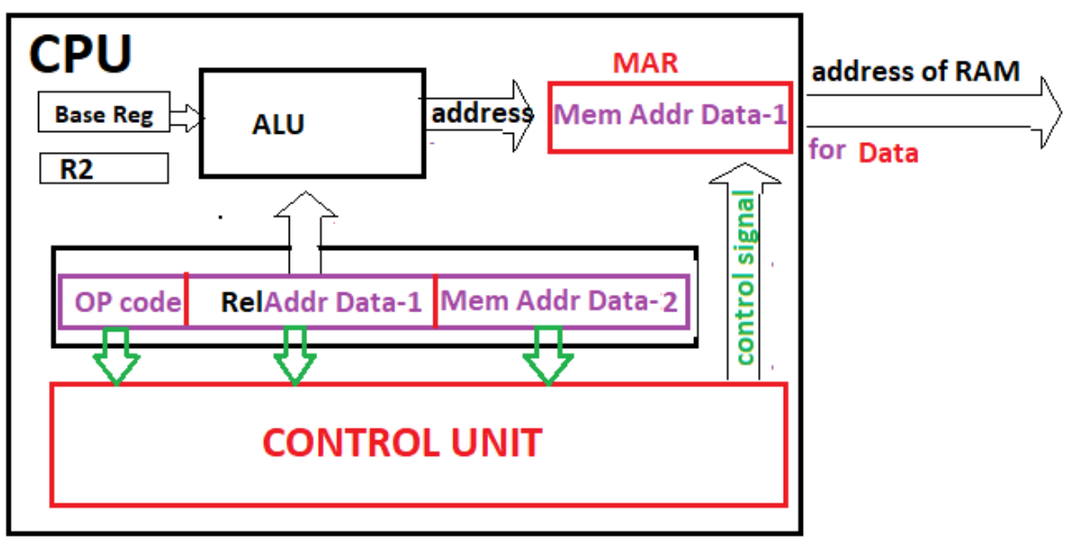
d) Zero address formats:
Both operands and result field remain implicit. Example: stack-based ALU instructions where both operands are read from stack and result is also saved to stack. In such case, operands must be stored to stack using special instruction: PUSH and then ALU instruction is used. The result is stored to stack by default. However, for further/later uses, results or partial results can be saved to RAM using POP instructions. A special register called Stack Pointer (SP) is used to point top location of stack. The SP is designed to auto-increment or auto-decrement following PUSH, POP and ALU instructions.
<table>
<tr><td>Opcode</td></tr>
</table>
add M[sp-1] = M[sp] + M[sp-1]
load M[sp] = M[M[sp]]
– Stack can be in registers or in memory
– usually top of stack cached in registers

In some processors, zero address format is used to set or reset a particular bit of flag or status register.
</td></tr>
<tr><td>5.</td><td>What do you understand by memory addressing mode? Discuss different variants of memory addressing mode.</td></tr>
<tr><td>6.</td><td>Show instruction</td></tr>
<tr><td>7.</td><td>What is Accumulator-based CPU? Discuss some instructions and addressing modes of a commercial accumulator based CPU (you can used 8085 as reference).</td></tr>
<tr><td>8.</td><td>What are the functions of following registers:
Program Counter (PC), Instruction register (IR), Memory Address Register (MAR), Accumulator.</td></tr>
<tr><td>9.</td><td>What do you understand by Instruction fetch?</td></tr>
<tr><td>10.</td><td>How does the CPU calculate memory address of next instruction to be fetched?</td></tr>
<tr><td>11.</td><td>How a CPU is designed to run program? Discuss the steps that the CPU is designed to follow in processing an instruction.</td></tr>
<tr><td>12.</td><td>How the Program Counter (PC) is updated while the CPU runs a program.</td></tr>
<tr><td>13.</td><td>What is Control unit? What are the approaches to design control unit of CPU?</td></tr>
<tr><td>14.</td><td>What do you understand by (i) microprogram (ii) microinstruction (iii) microoperation (iv) control memory and (v) micro-programmed control unit?</td></tr>
<tr><td>15.</td><td>List the important characteristics of CISC and RISC processors.</td></tr>
<tr><td>16.</td><td>Discuss addressing modes of CISC and RISC processors.</td></tr>
<tr><td>17.</td><td>What is pipelining architecture? List its benefits and challenges.</td></tr>
<tr><td>18.</td><td>What do you understand by data dependency? Show how it affects pipeline architecture.</td></tr>
<tr><td>19.</td><td>What is Branch Instruction? Show how it affects pipeline architecture.</td></tr>
<tr><td>20.</td><td>What do you understand by Cycles Per Instruction (CPI) of an instruction? How does pipeline architecture reduces average CPI of a program.</td></tr>
<tr><td>21.</td><td>Write CPU performance equation and explain each term.</td></tr>
<tr><td>22.</td><td>Compare the run time a program on RISC and CISC processor. Also comment on relative values.</td></tr>
</table>

23.	Show the instruction formats of MIPS R2000 processor.
24.	 In the early 1970s, telephone calls didn't instantly bounce between handheld devices and cell towers. Back then, the connection process required human operators to laboriously plug cords into the holes of a switchboard. Come 1974, a team of IBM researchers led by John Cocke set out in search of ways to automate the process. They envisioned a telephone exchange controller that would connect 300 calls per second (1 million per hour). Hitting that mark would require tripling or even quadrupling the performance of the company's fastest mainframe at the time — which would require fundamentally reimagining high-performance computing. IBM RISC technology originated in 1974 in a project to design a large telephone-switching network capable of handUng an average of three hundred calls per second. With an approximate 20 000 instructions per call and Stringent real-time response requirements, the performance target was 12 million instructions per second (MIPS) [1]. This specialized apphcation required a very fast processor, but did not have to perform compUcated instructions and had little demand for floating-point calculations. Other than moving data between registers and memory, the machine had to be able to add, combine fields extracted from several registers, perform branches, and carry out input/output operations. 1) Separate instruction and data caches, allowing a much higher bandwidth between memory and CPU; 2) No arithmetic operations to storage, which greatly simplified the pipeline; and 3) Uniform instruction length and simplicity of design, making possible a very short cycle time: ten levels of logic. (For example, all register-to-register operations executed in one cycle.)
25.	RISC Emer and Clark in 1984 found 20% of the VAX instructions needed 60% of the microcode and represented only 0.2% of the execution time. First, the RISC instructions were simplified so there was no need for a microcoded interpreter. The RISC instructions were typically as simple as microinstructions and could be executed directly by the hardware. Second, the fast memory, formerly used for the microcode interpreter of a CISC ISA, was repurposed to be a cache of RISC instructions. (A cache is a small, fast memory that buffers recently executed instructions, as such instructions are likely to be reused soon.) Third, register allocators based on Gregory Chaitin's graph-coloring scheme made it much easier for compilers to efficiently use registers, which benefited these register-register ISAs. Finally, Moore's Law meant there were enough transistors in the 1980s to include a full 32-bit datapath, along with instruction and data caches, in a single chip. Today, 99% of 32-bit and 64-bit processors are RISC. Emer, J. and Clark, D. A characterization of processor performance in the VAX-11/780. In *Proceedings of the 11th International Symposium on Computer Architecture* (Ann Arbor, MI, June). ACM Press, New York, 1984, 301–310. **DEC engineers showed that the more complicated CISC ISA executed about 75% of the number instructions per program as RISC (the first term), but in a similar**

	technology CISC executed about five to six more clock cycles per instruction (the second term), making RISC microprocessors approximately 4× faster.
26.	Using suitable diagram, show how PC can be incremented? 
27.	How addresses of Instructions are generated in CPU? Program Counter (PC) is first loaded with the address of 1st instruction of a program; either by operating system or by used. Once an instruction is fetched (read) from RAM, PC is incremented to point to next instruction in RAM. Memory Address Register is another special register in CPU used to interface address bus. So the contents of PC is loaded to MAR and then it is sent to address bus. 
28.	How does the CPU get/generate address of DATA? If the memory address of DATA is provided in the operand field of an instruction, the control unit decode the instruction and will also load the address of DATA from operand field of instruction to MAR.

<table>
<tr><td></td><td> 
</td></tr>
<tr><td>29.</td><td>How does the CPU get/generate address of DATA if the operand field contains relative address?
If the exact memory address of DATA is not provided in the operand field of an instruction, the control unit decode the instruction and will send the relative address to ALU/special ALU.
Contents of base register is added to relative address and memory address of data is generated.
This address is loaded to MAR.

 
</td></tr>
<tr><td>30.</td><td></td></tr>
</table>