Name: Vin Sovichhay

ID: e20241261

Group: I2-AMS2B

TP5

1) Explain about Instruction Set Architecture. What is it? And what does it do?

+ Instruction Set Architecture (ISA) is the part of computer architecture that defines how a CPU is programmed and controlled. It acts as a bridge between software (programs) and hardware (processor). ISA can be understood as the "language of the CPU"**.** It specifies what instructions the processor can understand and execute, such as adding numbers, loading data, or jumping to another instruction.
+ It is also considered a programmer-visible model, meaning it describes everything a programmer needs to know to write programs that run on the CPU.
+ An Instruction Set Architecture defines:

    - Types of instructions (ADD, LOAD, STORE, JUMP)
    - Registers used by the CPU
    - Data types and formats
    - Instruction formats and opcodes
    - Addressing modes (how data is accessed)
    - Memory and input / output behavior.

+ ISA plays several important roles:
    - Connects software and hardware: It allows programs to communicate with the CPU in a standard way.
    - Defines what the CPU can do: It determines the operations a processor is capable of performing.
    - Ensure compatibility: Different processors with the same ISA can run the same programs, even if their internal designs are different.
    - Guides processor design and performance: ISA affects how efficiently programs run and how hardware is built.

2) What is Microarchitecture? How is it different from ISA?

+ Microarchitecture is the implementation of the Instruction Set Architecture (ISA). This means it shows how the instructions defined by the ISA are physically carried out by the hardware.
+ For example, two processors may support the same ISA (like Arm or x86), but they can have different microarchitectures, meaning they are built differently and may have different performance.
+ Difference between Microarchitecture and ISA:

Feature	ISA	Microarchitecture
Definition	Defines the instructions and operations of a CPU	Defines the internal design of the CPU

Focus	What the CPU can do	How the CPU does
Visibility	Seen by programmers	Hidden from programmers
Example	ADD, LOAD, JUMP instruction	Pipelines, cache, ALU design
Flexibility	Same ISA can run on different CPUs	Different designs can implement same ISA

3) What is Instruction Format?

+ Instruction Format is the way a computer instruction is structured and stored in memory. It defines how the different parts of an instruction are arranged so that the CPU can understand and execute it correctly. In simple terms, an instruction format acts like a blueprint that tells the processor:
    - What operation to perform (called the opcode)
    - Which data to use (called operands)
    - How to access that data (addressing mode).

4) Explain Zero-Address Instruction with example.

+ A Zero-Address Instruction is a type of instruction format that does not contain any operand (address) fields. Instead of specifying operands explicitly, it works on data that is already available in a predefined place usually the top of a stack.
+ For example: Consider the expression: $Y = (A + B)*(C + D)$ Then, executed like this:

Instruction	Stake
PUSH A	$TOP = A$
PUSH B	$TOP = B$
ADD	$TOP = A + B$
PUSH C	$TOP = C$
PUSH D	$TOP = D$
ADD	$TOP = C + D$
MUL	$TOP = (C + D) \times (A + B)$
POP Y	$M[Y] = TOP$

5) Explain about Three-Address Instruction. Give an example.

+ A Three-Address Instruction is a type of instruction format that contains three operand addresses along with the operation code. These addresses usually represent two source operands and one destination operand where the result will be stored.
+ For example: Consider the expression $Y = (A + B)*(C + D)$ Then, executed like this:

Instruction	Stake
ADD R1, A, B	$R1 = M[A] + M[B]$
ADD R2, C, D	$R2 = M[C] + M[D]$
MUL R3, R1, R2	$R3 = R1 \times R2$
MOV Y, R3	$M[Y] = R3$

6) Why is Three-Address Instruction being used in modern GPUs. Do some research and explain.

+ Modern GPUs are designed for massive parallel computation, meaning they execute thousands of operations at the same time. According to NVIDA, GPUs achieve very high performance because they can process many threads simultaneously and provide high instruction throughput.
    a) **Supports parallel execution clearly:** GPUs follow a model where many threads perform operations at the same time (SIMT model). Three-address instructions fit this well because:
        - They explicitly show two inputs and one output
        - Each thread can perform the same operation independently
    b) **Avoids overwriting data:** In GPUs computing, many thread run at once, so data must be handled carefully. Three-address instructions store results in a separate destination, instead of overwriting input values. This is useful because:
        - Multiple threads can reuse the same input data
        - Fewer conflicts occur between threads
    c) **Helps increase instruction throughput:** NVIDA explains that GPUs are designed to deliver high instruction throughput by executing many operations concurrently. Three-address instructions help by:
        - Breaking complex expressions into simple, independent steps
        - Allowing more instructions to be executed simultaneously
    d) **Matches GPU hardware design:** GPU processors use many small cores and rely on simple, repeated operations instead of complex instruction handling. Three-address instructions are:
        - Simple
        - Explicit
        - Easy to schedule across many cores
    e) **Easier for compiler and scheduling:** The CUDA model shows that work is divided into many threads and blocks that run independently. Three-address instructions help compilers:
        - Organize operations into small steps
        - Distribute them efficiently across threads
        - Optimize parallel execution

References

1) https://www.geeksforgeeks.org/computer-organization-architecture/microarchitecture-and-instruction-set-architecture/
2) https://www.geeksforgeeks.org/computer-organization-architecture/microarchitecture-and-instruction-set-architecture/
3) https://www.geeksforgeeks.org/computer-organization-architecture/computer-organization-instruction-formats-zero-one-two-three-address-instruction/
4) https://www.geeksforgeeks.org/computer-organization-architecture/computer-organization-instruction-formats-zero-one-two-three-address-instruction/
5) https://www.geeksforgeeks.org/computer-organization-architecture/computer-organization-instruction-formats-zero-one-two-three-address-instruction/
6) https://docs.nvidia.com/cuda/cuda-c-programming-guide/