Computer Architecture

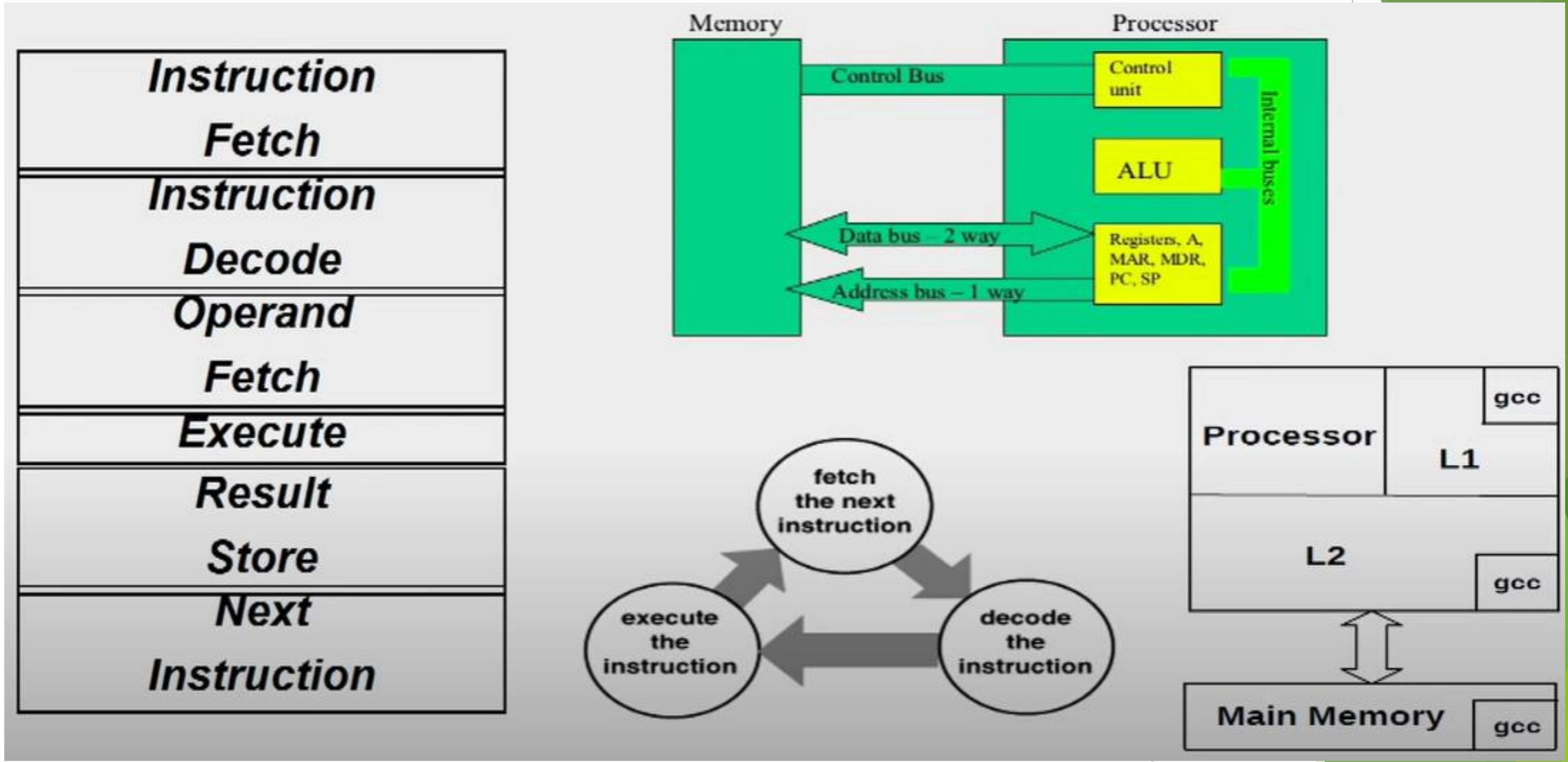
Instruction Set Architecture

Dr. Mohammad Reza Selim

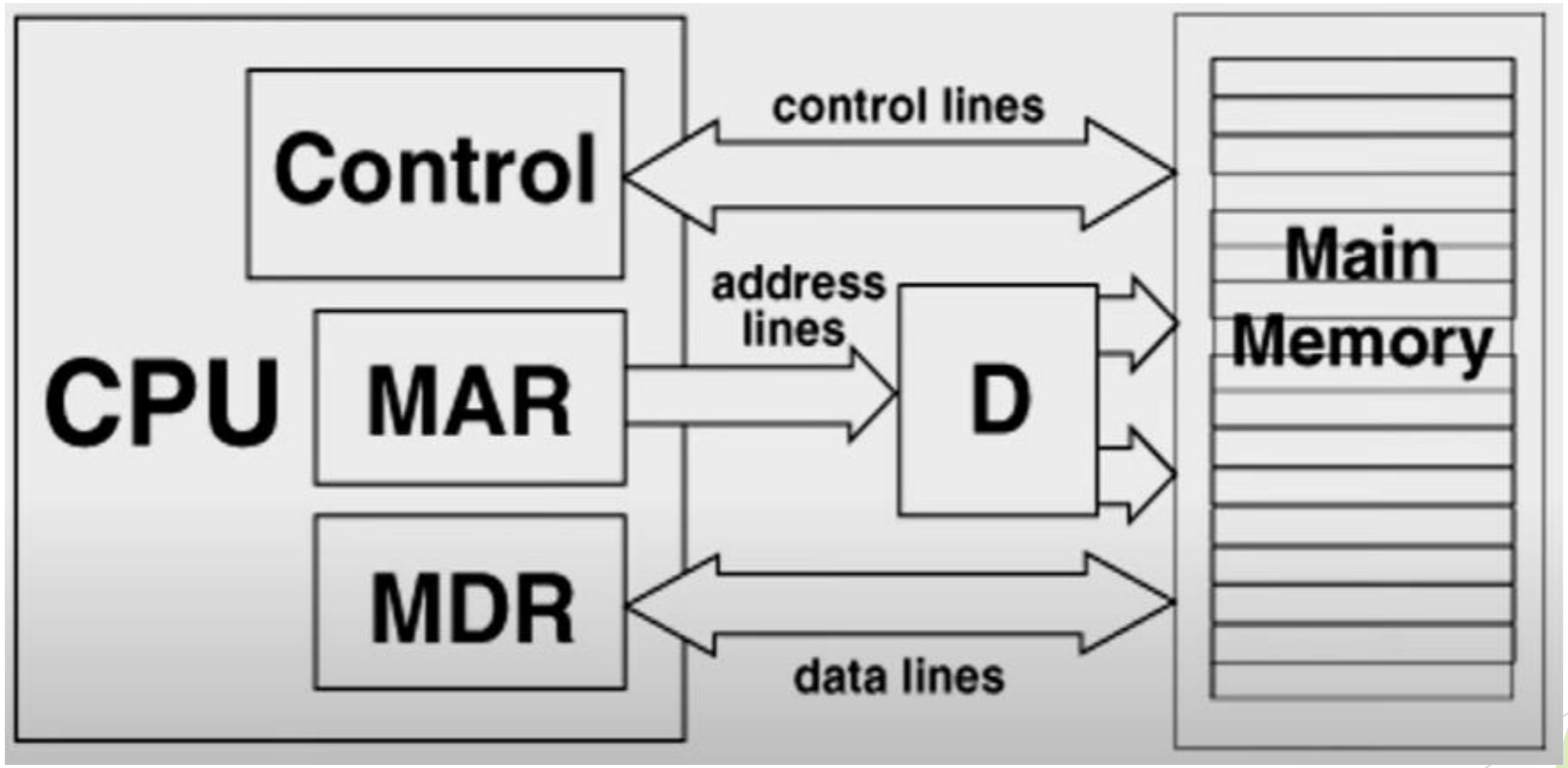
Lecture Outline

- Processor – Memory Interaction
- Inside the CPU
- Instruction Execution Cycle
- Instruction Set Architecture
- CISC vs RISC

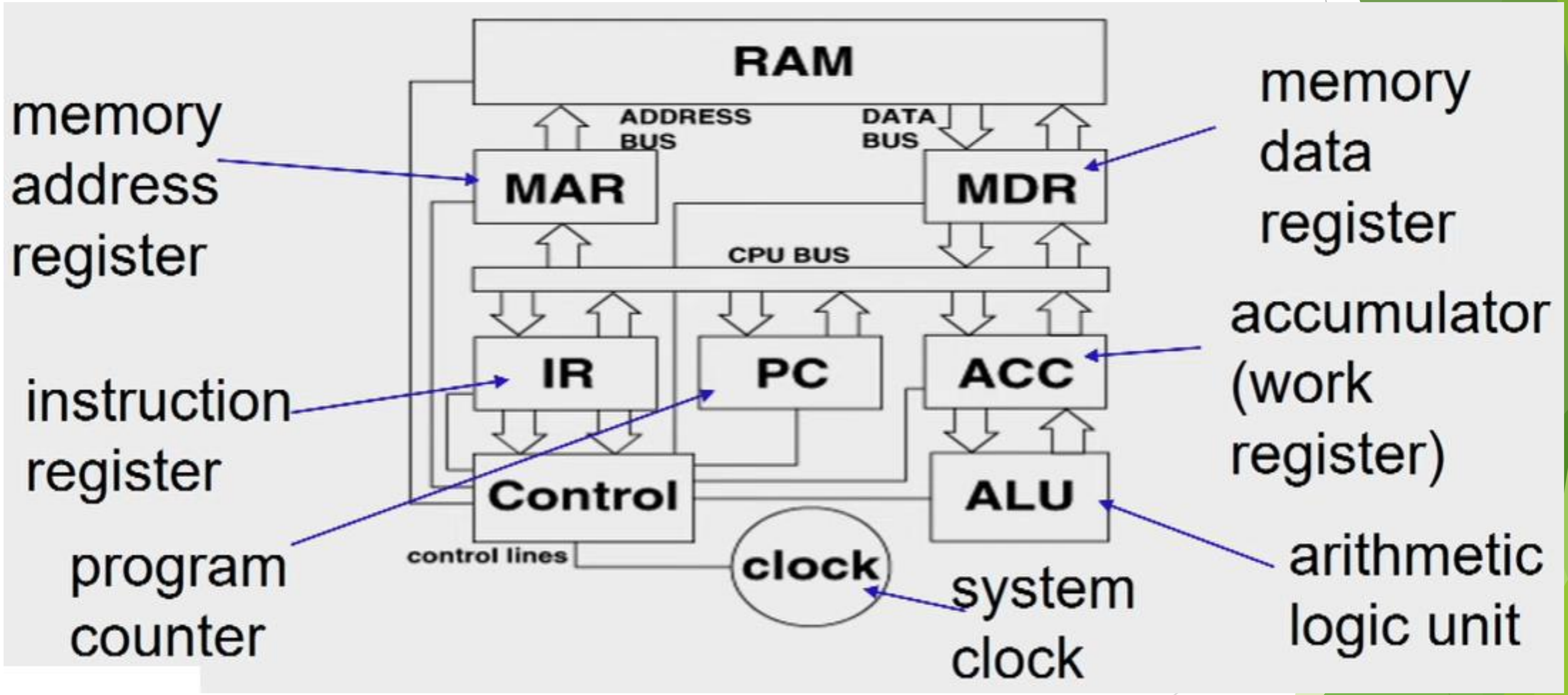
Processor Memory Interaction



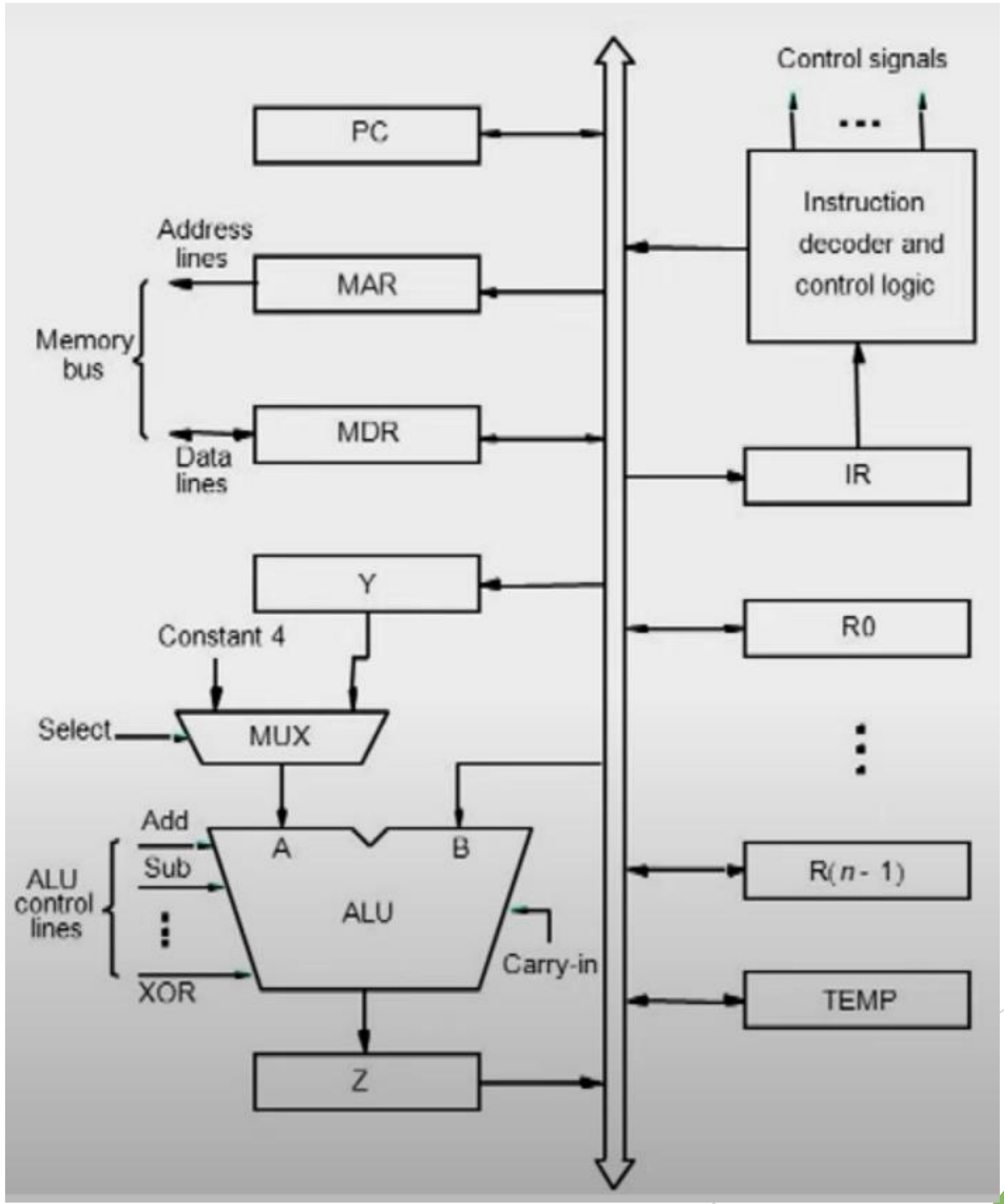
Processor Memory Interaction



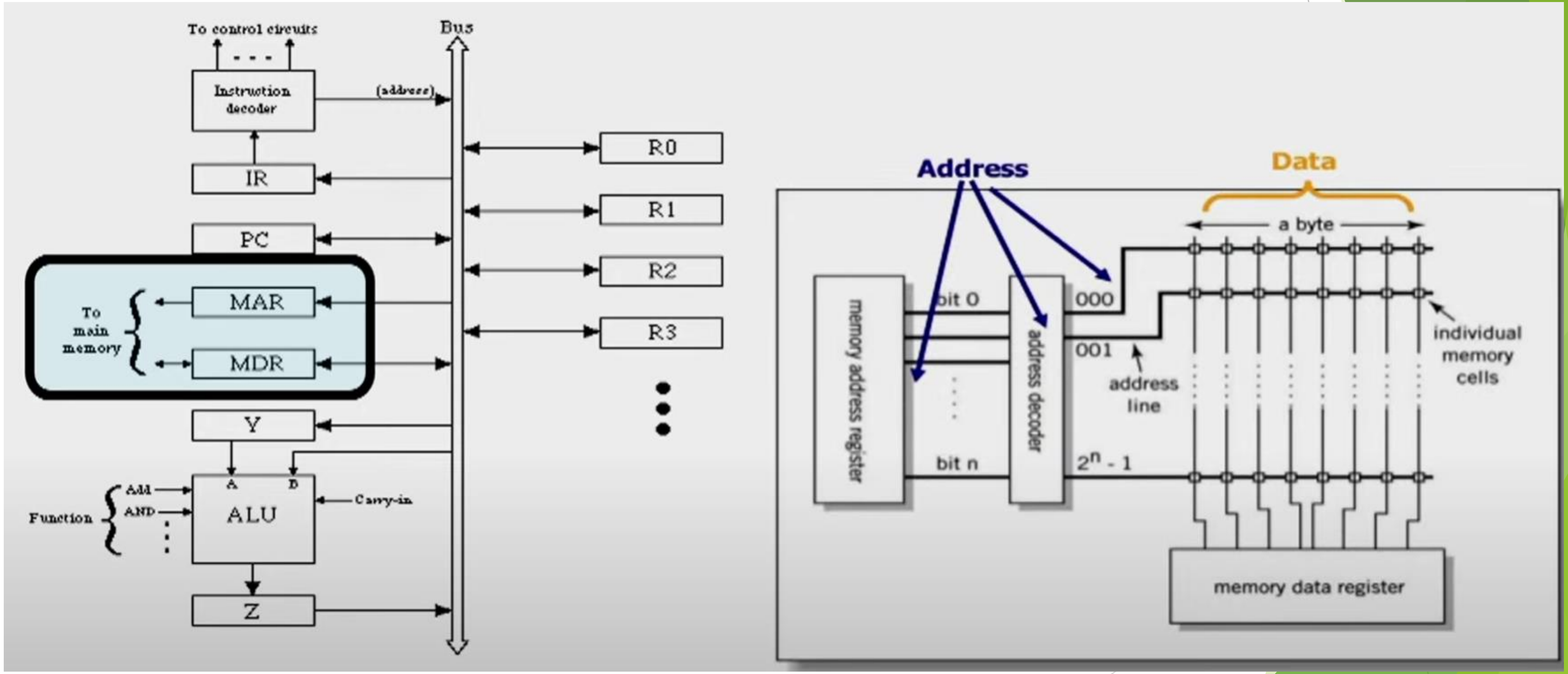
Inside the CPU



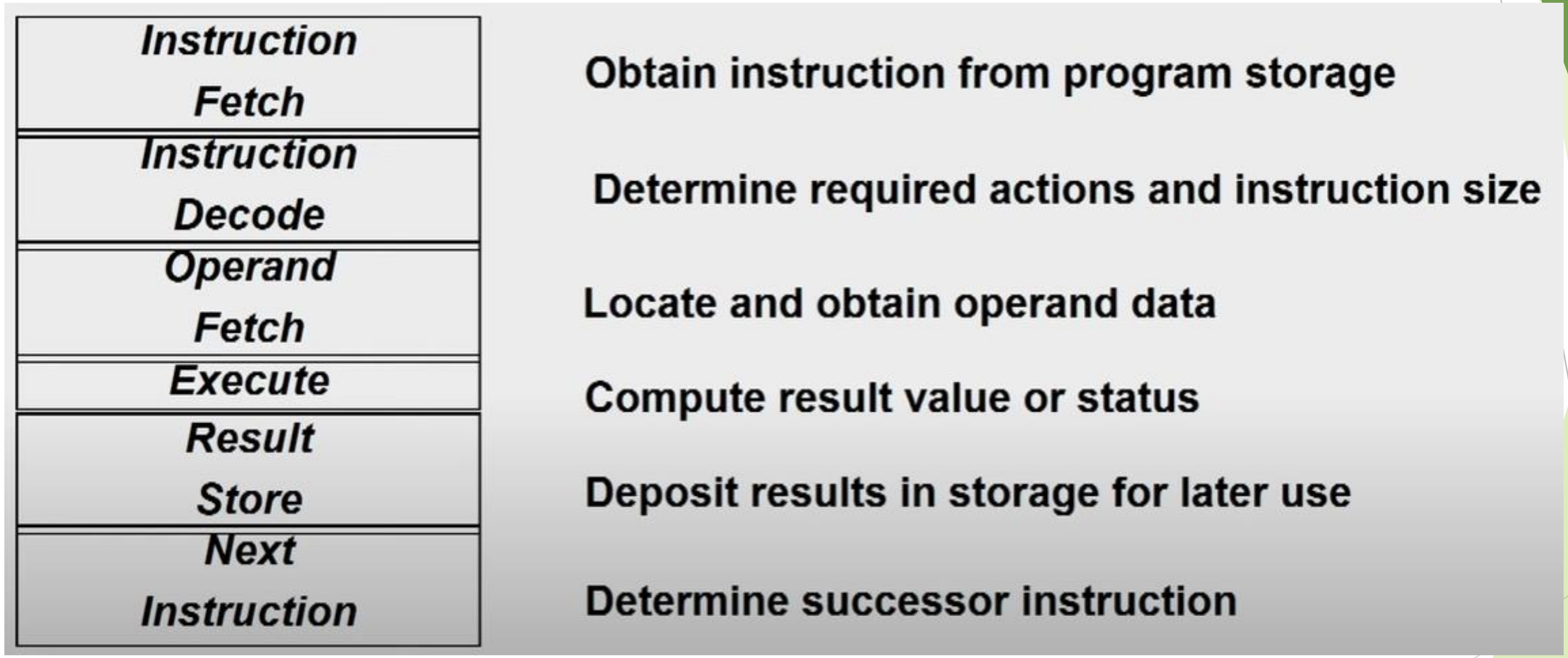
Inside the CPU



How Memory Works



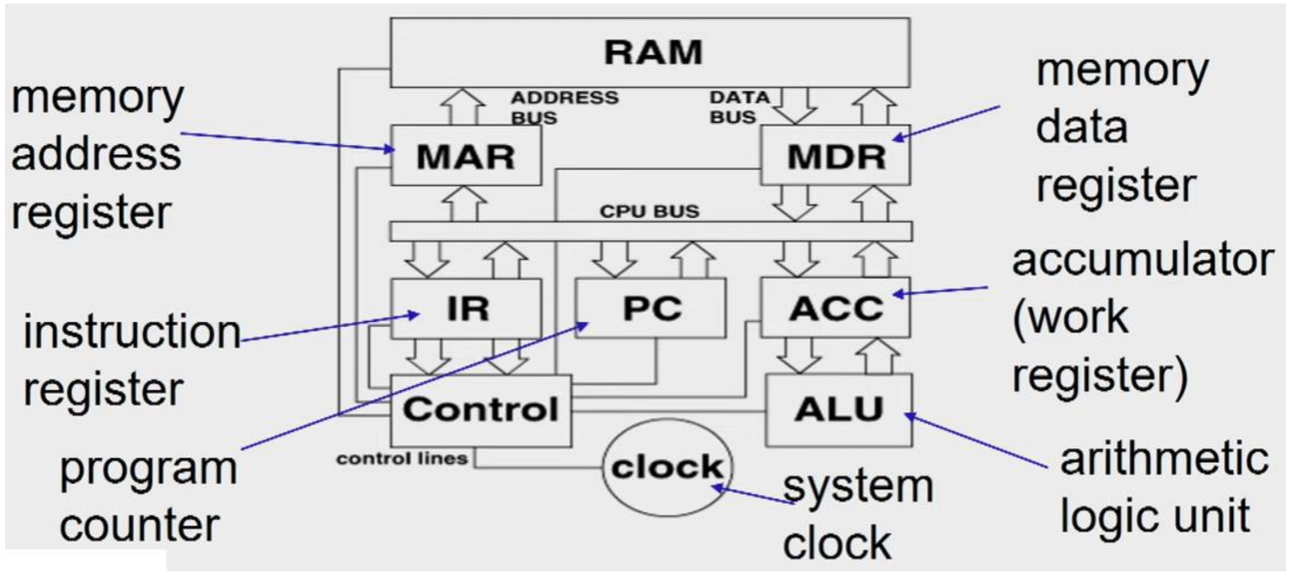
Instruction Execution Cycle



Instruction Fetch	Obtain instruction from program storage
Instruction Decode	Determine required actions and instruction size
Operand Fetch	Locate and obtain operand data
Execute	Compute result value or status
Result Store	Deposit results in storage for later use
Next Instruction	Determine successor instruction

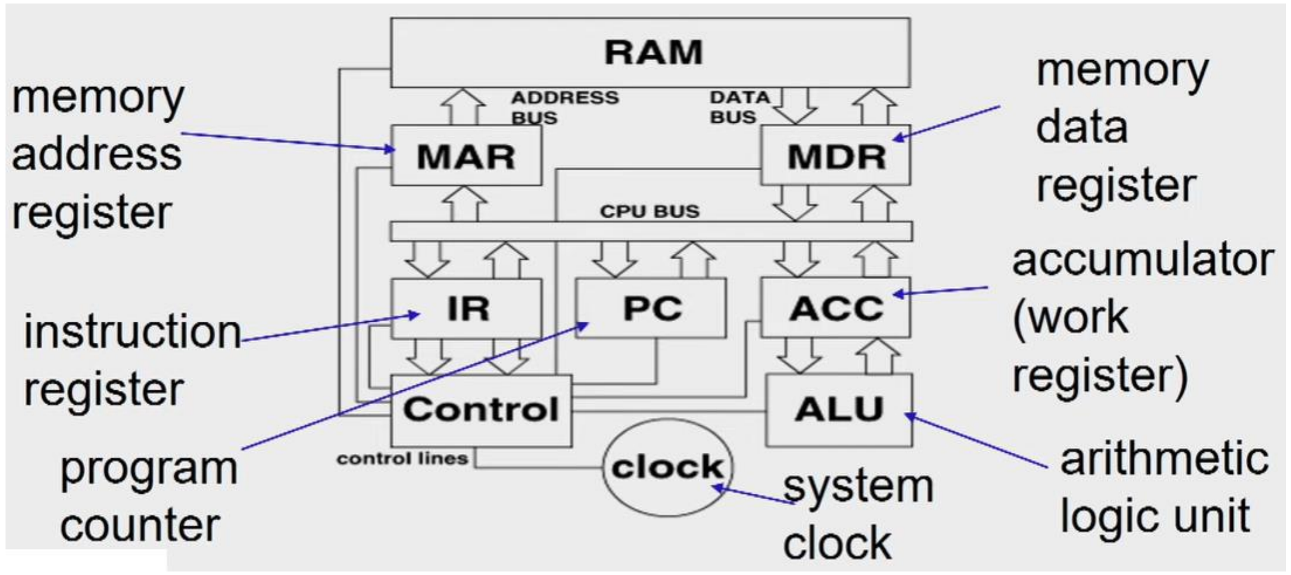
Fetch the Instruction

- Address of the next instruction is transferred from PC to MAR
- The Instruction is located in memory
- Instruction is copied from memory to MDR



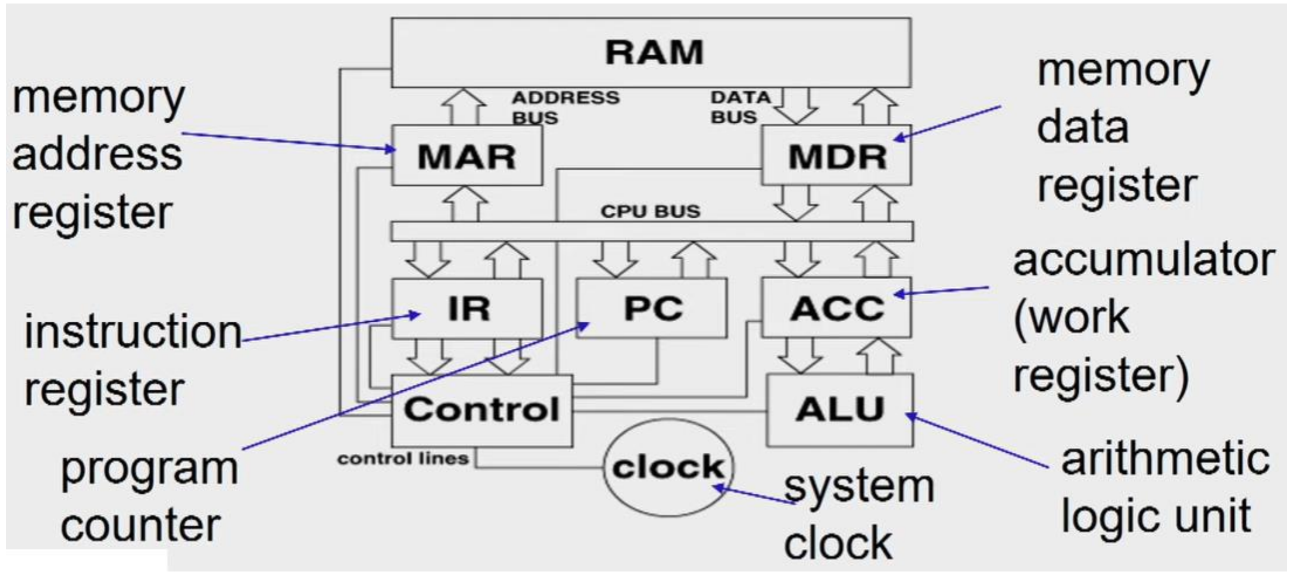
Decode the Instruction

- Instruction is transferred to and decoded in IR



Execute the Instruction

- Control unit sends signals to appropriate devices to cause execution the instruction



Instruction Set Architecture (ISA)

- **ISA** is an abstract model of a **computer**.
- CPU is an implementation of ISA.
- It specifies what the processor is capable of doing.
- It is the interface between your hardware and the software.

Instruction Set Architecture (ISA)

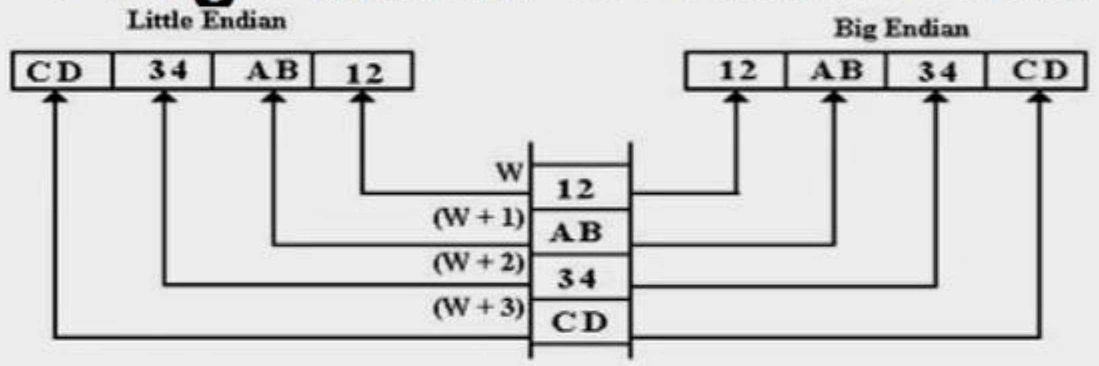
Several Aspects that make differences among ISAs

- Encoding of ISA
  - Fixed vs variable length coding
  - ARM and MIPS instructions are 32 bit long. Programs requires more space
  - 80x86 instructions are 1 to 18 bytes. Programs requires less space
- Size and Types of Operands
  - 80x86, ARM, and MIPS support operand sizes of
    - 8-bit (ASCII character), 16-bit (Unicode character or half word), 32-bit (integer or word), 64-bit (double word or long integer), and IEEE 754 floating point in 32-bit (single precision) and 64-bit (double precision).
    - The 80x86 also supports 80-bit floating point (extended double precision)
- Types of Operations
- Memory Addressing
- Classes of ISA
- Addressing Modes

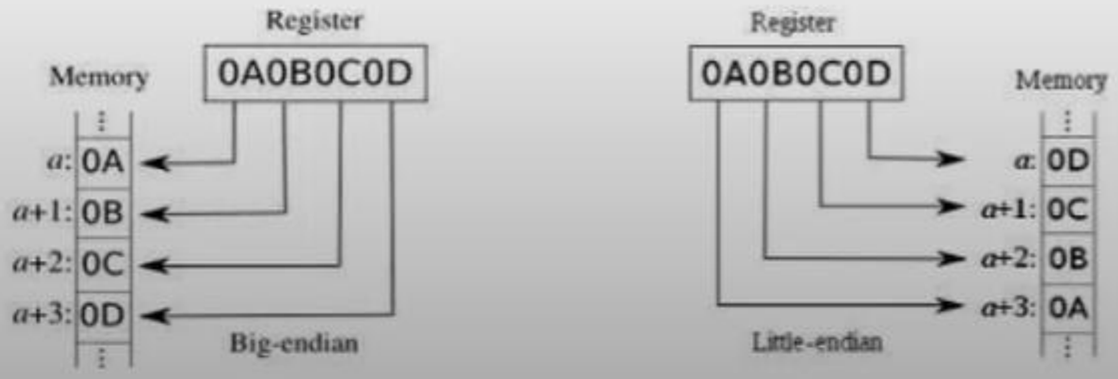
Instruction Set Architecture (ISA)

Byte Ordering

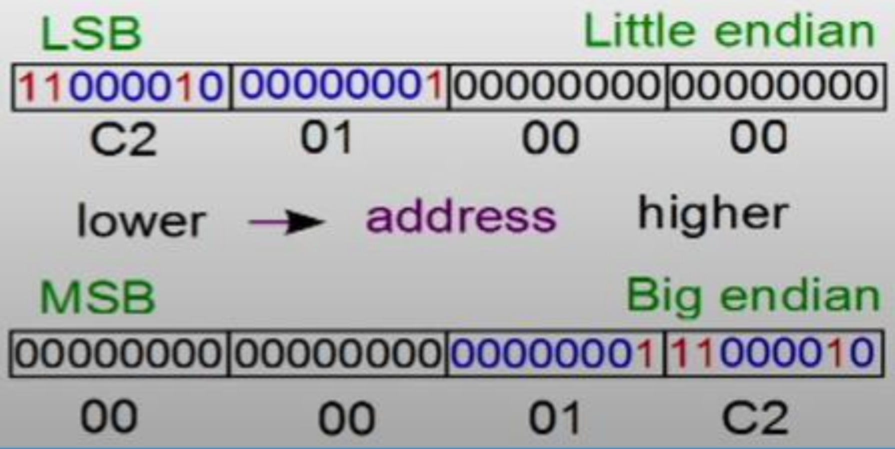
❖ **Big Endian vs Little Endian**



Big Endian vs. Little Endian

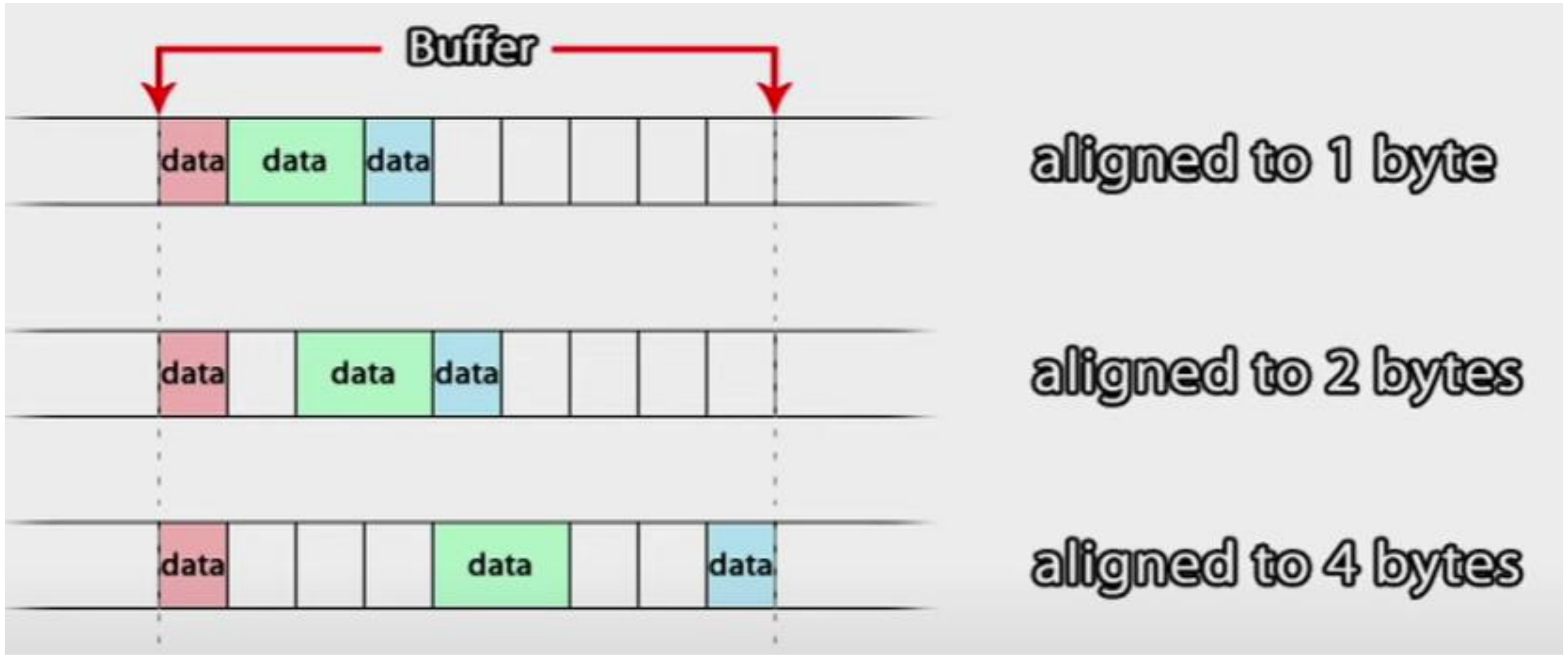


Int i = 450 = $2^8 + 2^7 + 2^6 + 2$ = x000001C2



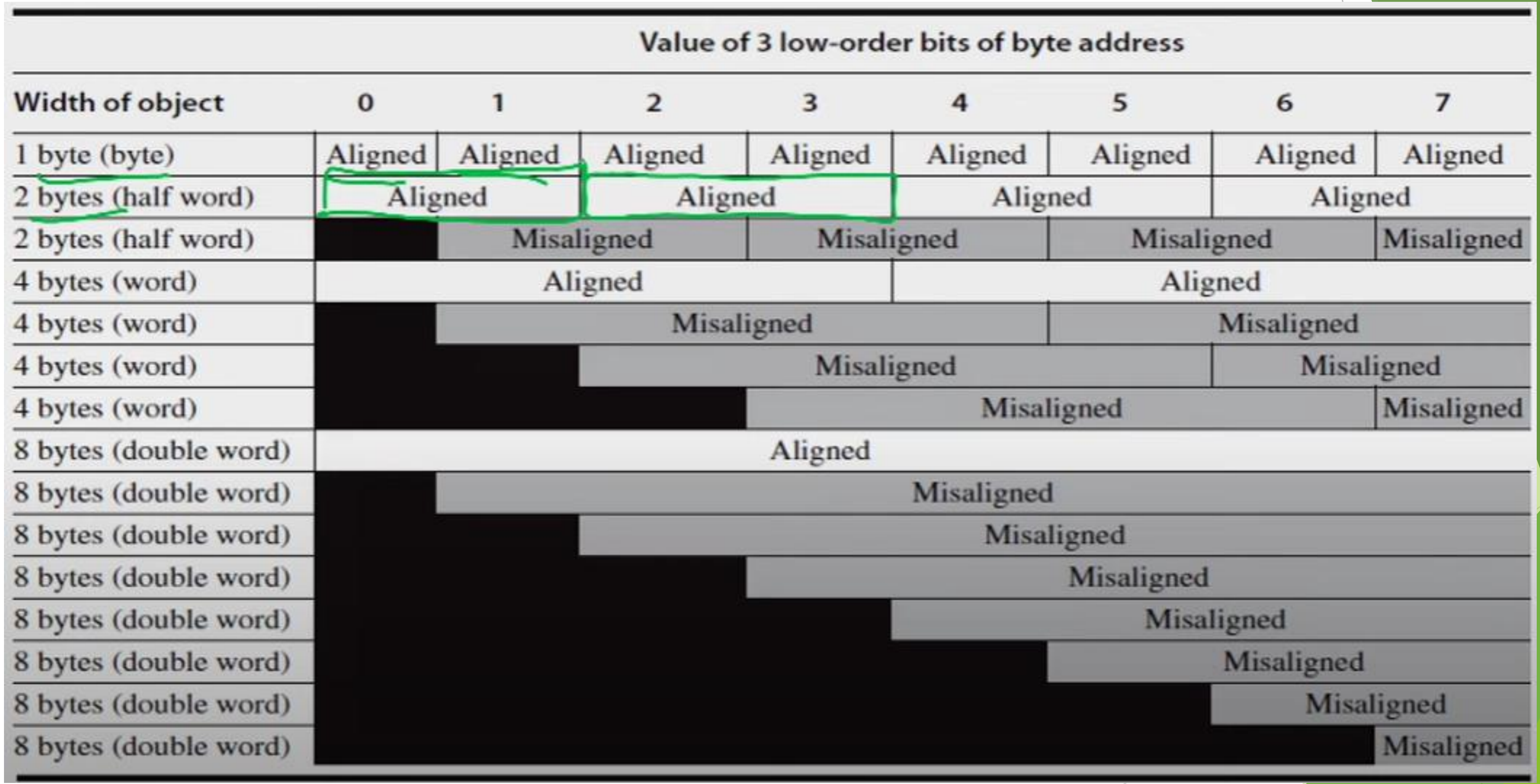
Instruction Set Architecture (ISA)

Byte/Word Alignment



Instruction Set Architecture (ISA)

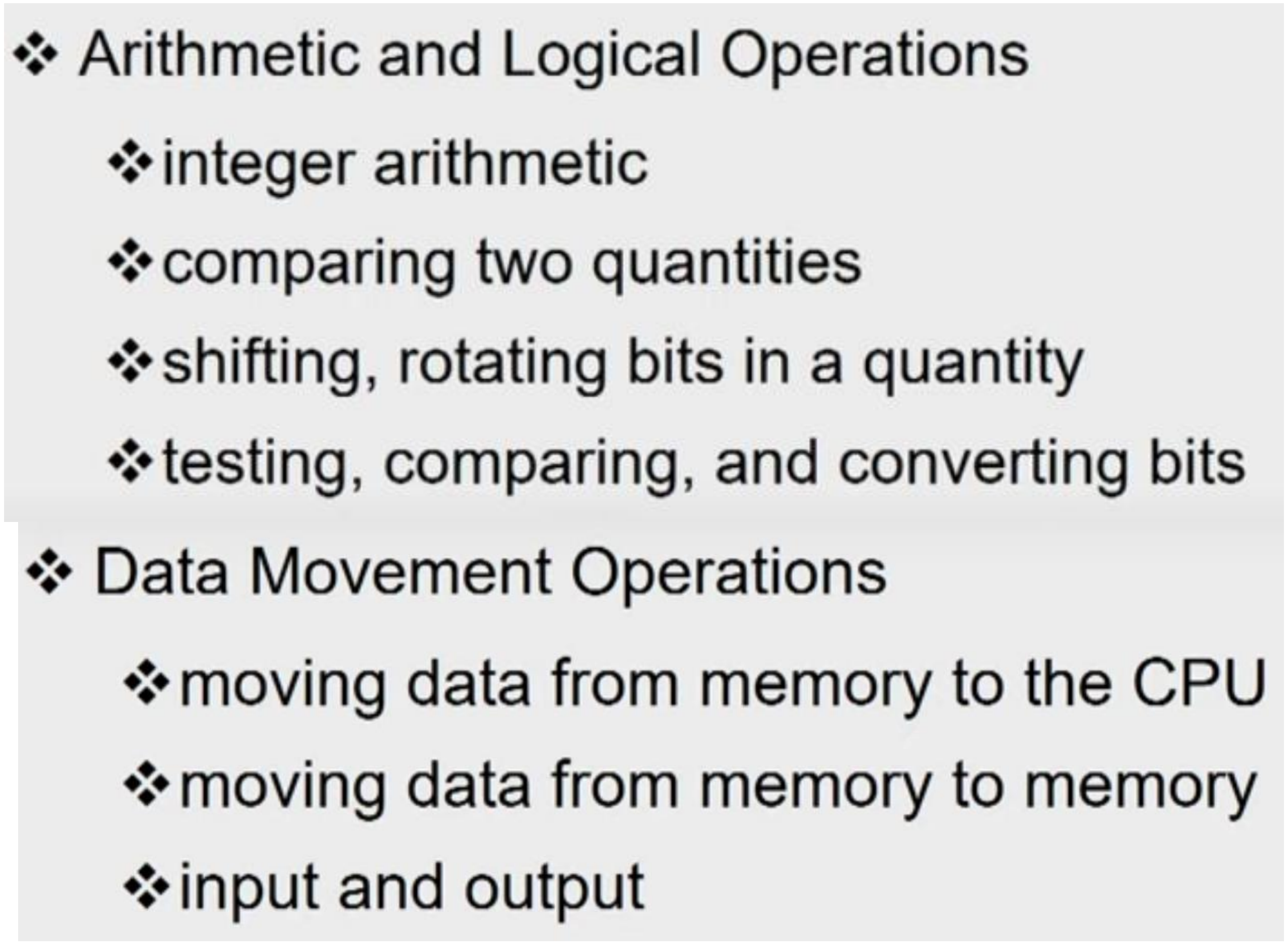
Byte/Word Alignment



	Value of 3 low-order bits of byte address							
Width of object	0	1	2	3	4	5	6	7
1 byte (byte)	Aligned	Aligned	Aligned	Aligned	Aligned	Aligned	Aligned	Aligned
2 bytes (half word)	Aligned		Aligned		Aligned		Aligned	
2 bytes (half word)		Misaligned		Misaligned		Misaligned		Misaligned
4 bytes (word)	Aligned				Aligned			
4 bytes (word)		Misaligned				Misaligned		
4 bytes (word)			Misaligned				Misaligned	
4 bytes (word)				Misaligned				Misaligned
8 bytes (double word)	Aligned							
8 bytes (double word)		Misaligned						
8 bytes (double word)			Misaligned					
8 bytes (double word)				Misaligned				
8 bytes (double word)					Misaligned			
8 bytes (double word)						Misaligned		
8 bytes (double word)							Misaligned	
8 bytes (double word)								Misaligned

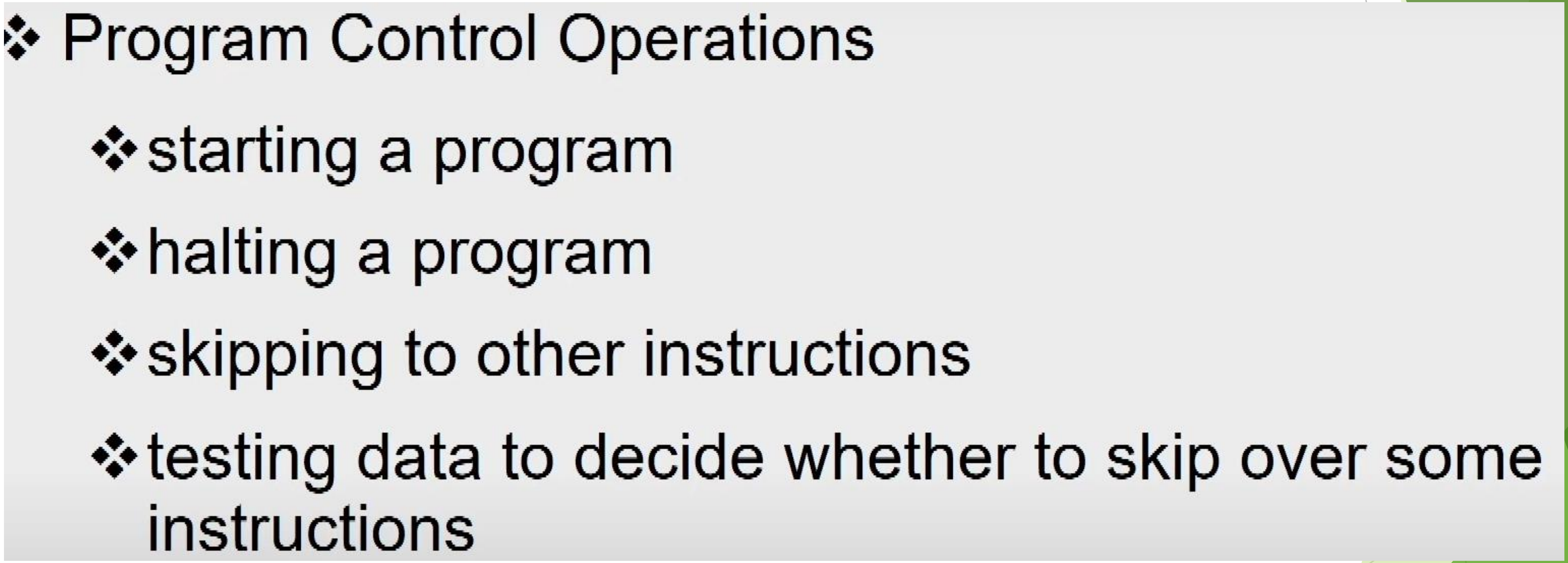
Instruction Set Architecture (ISA)

Types of Processor Operations



- Arithmetic and Logical Operations
  - integer arithmetic
  - comparing two quantities
  - shifting, rotating bits in a quantity
  - testing, comparing, and converting bits
- Data Movement Operations
  - moving data from memory to the CPU
  - moving data from memory to memory
  - input and output

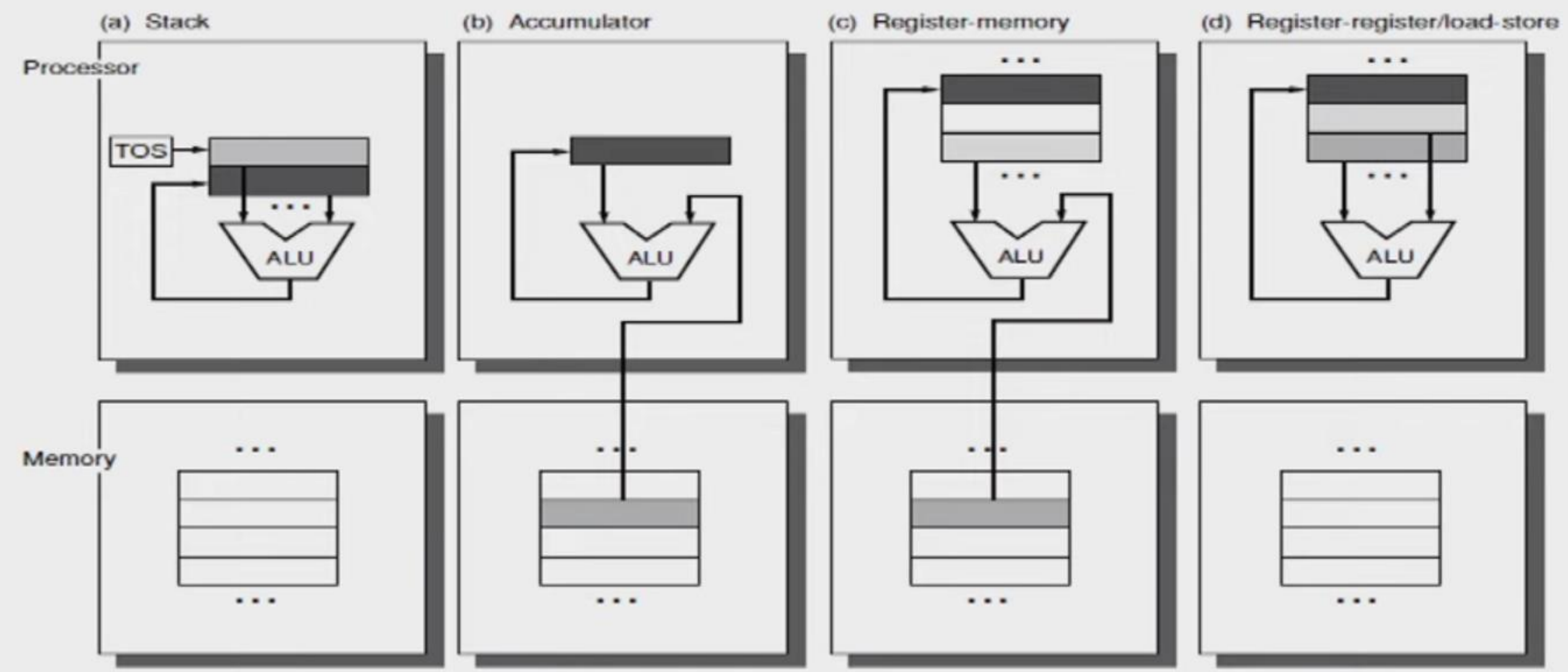
Types of Processor Operations



- Program Control Operations
  - starting a program
  - halting a program
  - skipping to other instructions
  - testing data to decide whether to skip over some instructions

Instruction Set Architecture (ISA)

Classes of ISA

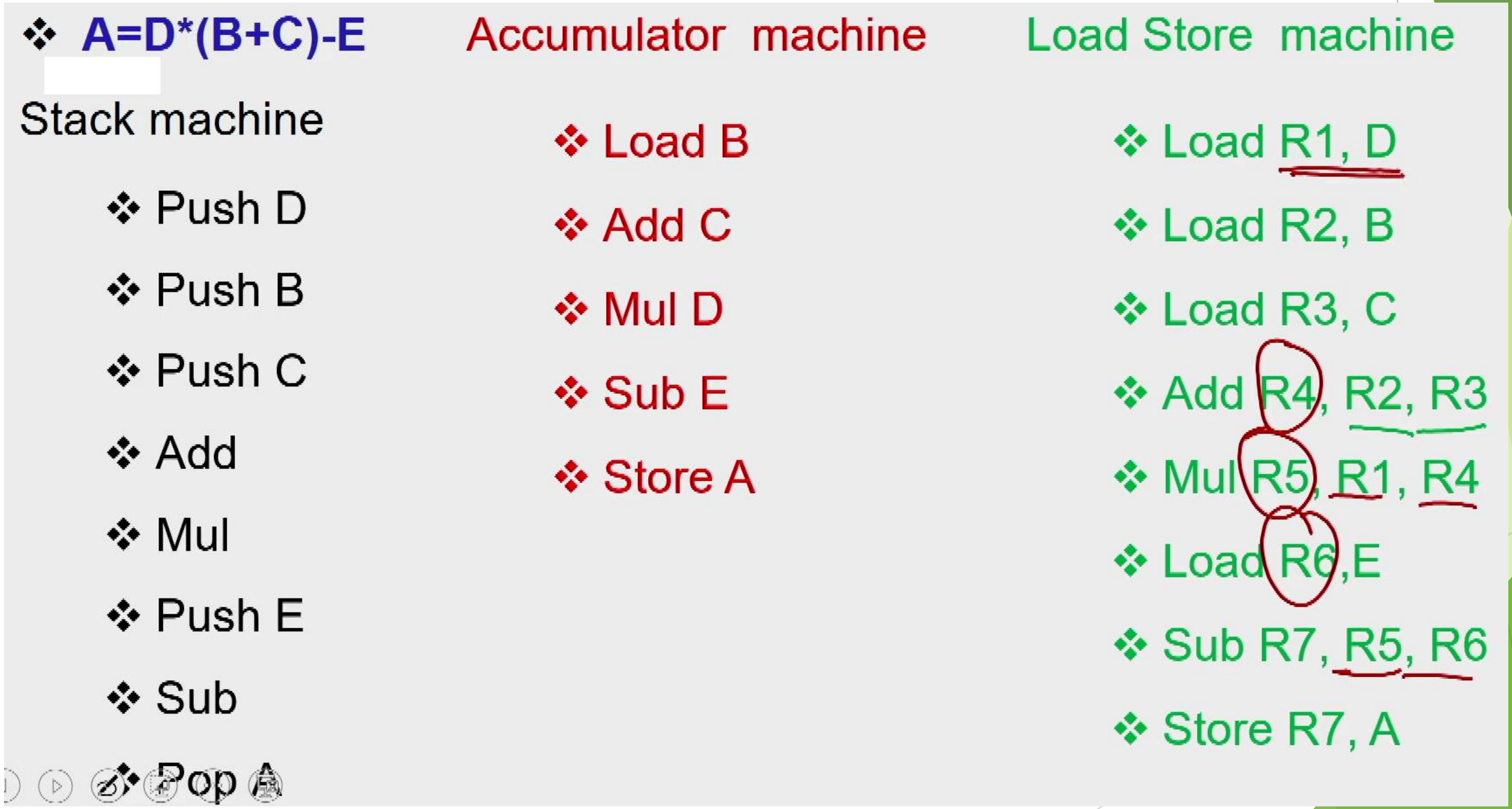


Stack	Accumulator	Register (register-memory)	Register (load-store)
Push A	Load A	Load R1,A	Load R1,A
Push B	Add B	Add R3,R1,B	Load R2,B
Add	Store C	Store R3,C	Add R3,R1,R2
Pop C			Store R3,C

The code sequence for C = A + B for four classes of instruction sets.

Instruction Set Architecture (ISA)

Classes of ISA (cont.)



❖ A=D*(B+C)-E

Stack machine

- ❖ Push D
- ❖ Push B
- ❖ Push C
- ❖ Add
- ❖ Mul
- ❖ Push E
- ❖ Sub
- ❖ Pop A

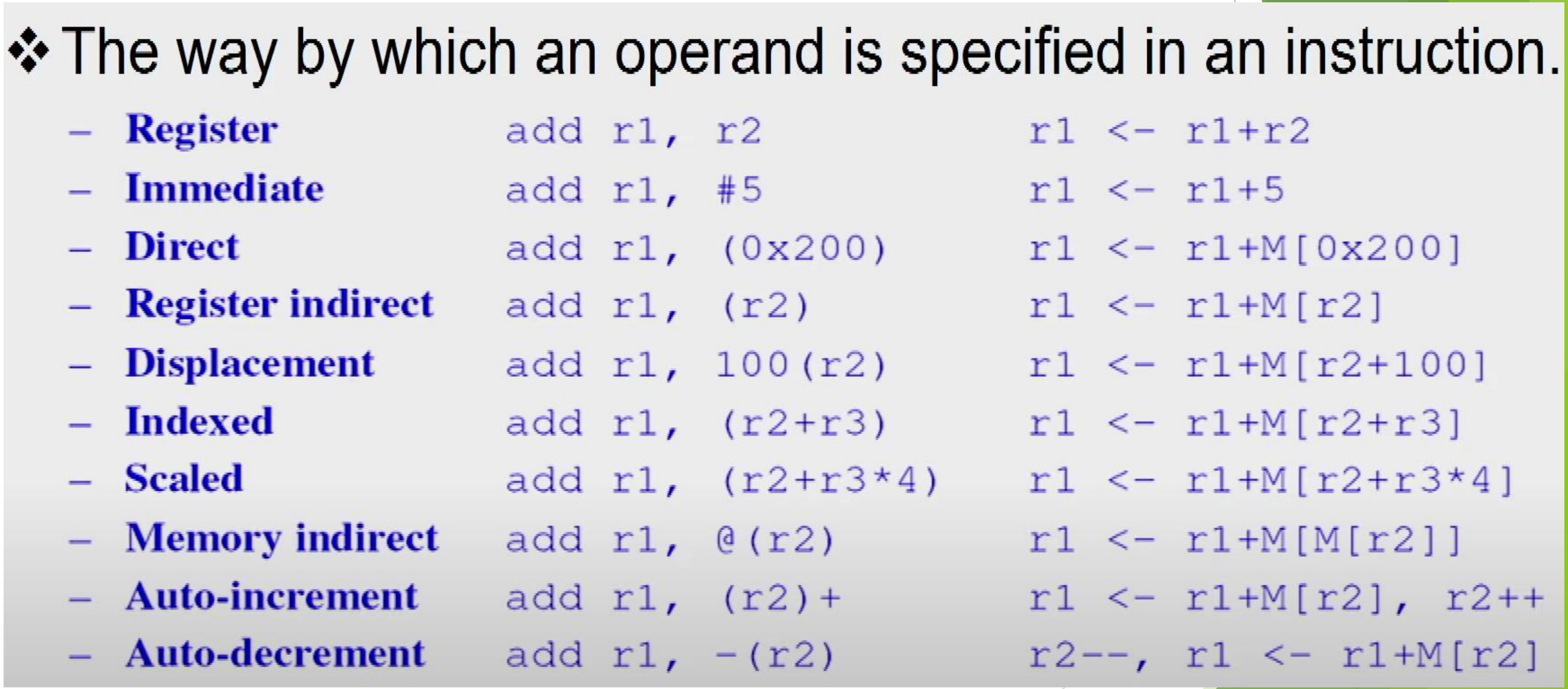
Accumulator machine

- ❖ Load B
- ❖ Add C
- ❖ Mul D
- ❖ Sub E
- ❖ Store A

Load Store machine

- ❖ Load R1, D
- ❖ Load R2, B
- ❖ Load R3, C
- ❖ Add R4, R2, R3
- ❖ Mul R5, R1, R4
- ❖ Load R6,E
- ❖ Sub R7, R5, R6
- ❖ Store R7, A

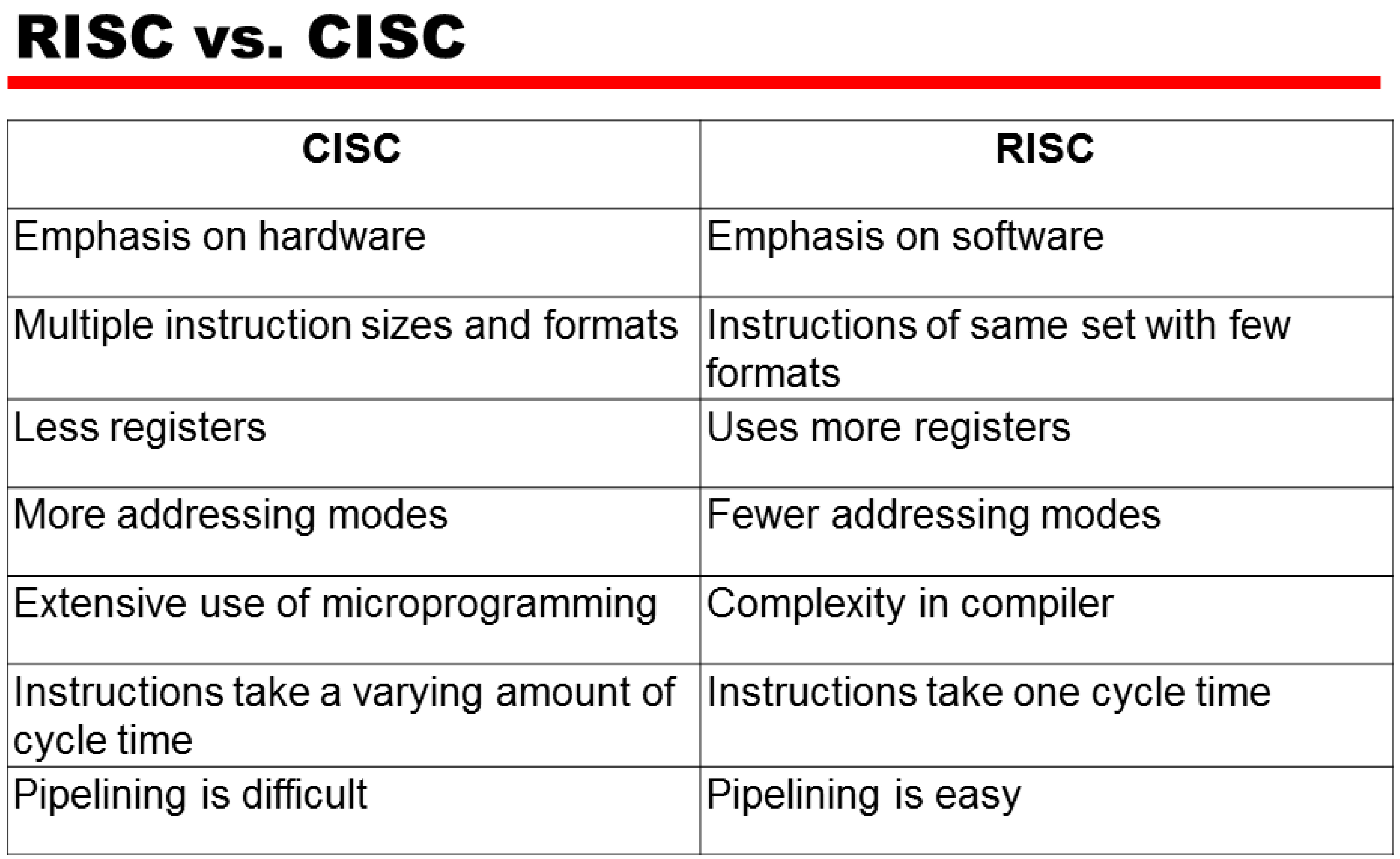
Instruction Set Architecture (ISA)
Addressing Modes



❖ The way by which an operand is specified in an instruction.

– **Register**	`add r1, r2`	`r1 <- r1+r2`
– **Immediate**	`add r1, #5`	`r1 <- r1+5`
– **Direct**	`add r1, (0x200)`	`r1 <- r1+M[0x200]`
– **Register indirect**	`add r1, (r2)`	`r1 <- r1+M[r2]`
– **Displacement**	`add r1, 100(r2)`	`r1 <- r1+M[r2+100]`
– **Indexed**	`add r1, (r2+r3)`	`r1 <- r1+M[r2+r3]`
– **Scaled**	`add r1, (r2+r3*4)`	`r1 <- r1+M[r2+r3*4]`
– **Memory indirect**	`add r1, @(r2)`	`r1 <- r1+M[M[r2]]`
– **Auto-increment**	`add r1, (r2)+`	`r1 <- r1+M[r2], r2++`
– **Auto-decrement**	`add r1, -(r2)`	`r2--, r1 <- r1+M[r2]`

Instruction Set Architecture (ISA)



RISC vs. CISC

CISC	RISC
Emphasis on hardware	Emphasis on software
Multiple instruction sizes and formats	Instructions of same set with few formats
Less registers	Uses more registers
More addressing modes	Fewer addressing modes
Extensive use of microprogramming	Complexity in compiler
Instructions take a varying amount of cycle time	Instructions take one cycle time
Pipelining is difficult	Pipelining is easy