



Advanced Parallel Architecture

Lesson 3



Annalisa Massini - 2014/2015

Von Neumann Architecture

-
- ▶ Summary of the traditional computer architecture:
Von Neumann architecture
 - ▶ <http://WilliamStallings.com/COA/COA7e.html>

Architecture & Organization

In describing computers, a distinction is often made:

- ▶ **Architecture** - attributes visible to the programmer
 - ▶ Instruction set, number of bits used for data representation, I/O mechanisms, addressing techniques
 - ▶ e.g. Is there a multiply instruction?
- ▶ **Organization** - operational units and their interconnections that realize the architectural specifications
 - ▶ Control signals, interfaces, memory technology
 - ▶ e.g. Is there a hardware multiply unit or is it done by repeated addition?

Structure & Function

- ▶ A **computer** is a **complex system** containing millions of elementary electronic components
- ▶ The key to describe a computer is to recognize its **hierarchical nature**, as for most complex systems:
 - ▶ At each level, the system consists of a **set of components**
 - ▶ The **interrelationships** between components
 - ▶ The **behavior at each level** depends only on a simplified, abstracted characterization of the system at the next lower level

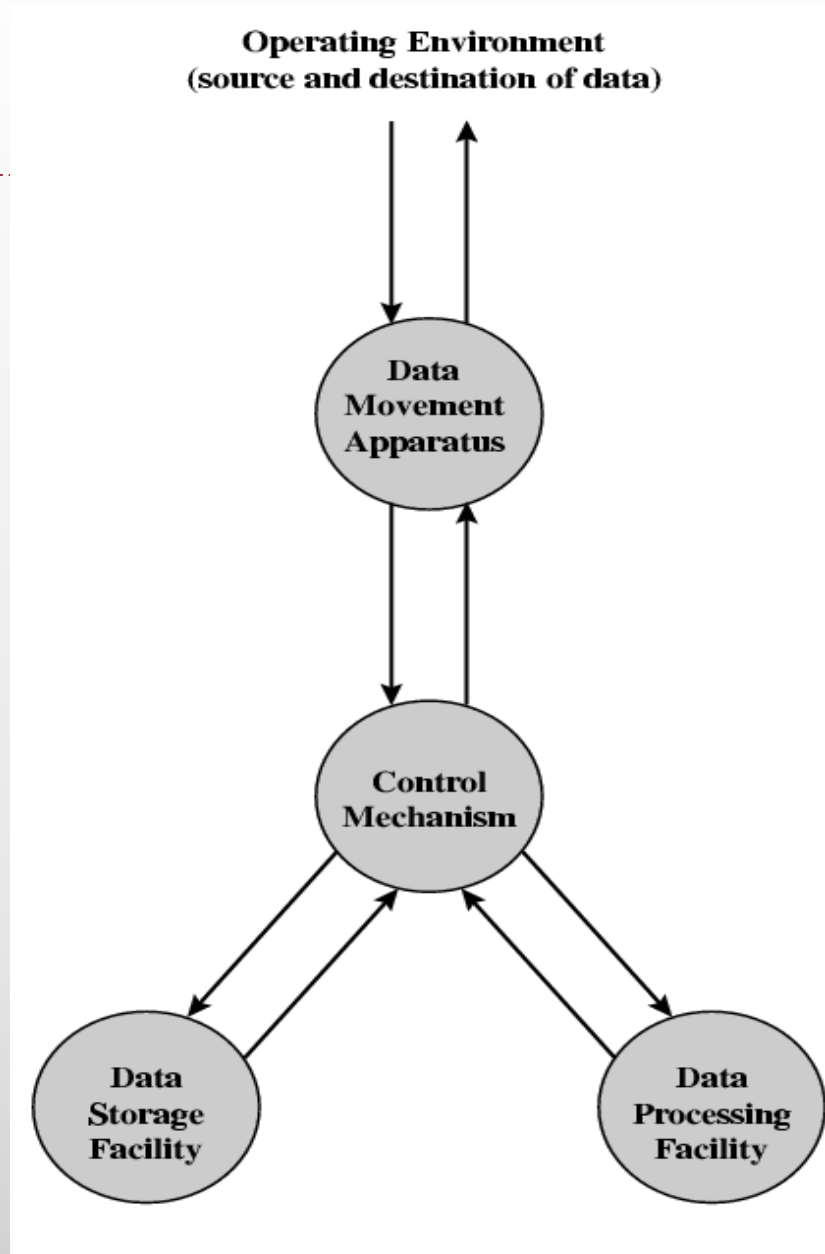
Structure & Function

- ▶ At each level, we are concerned with:
 - ▶ **Structure** - the way in which components relate to each other
 - ▶ **Function** - the operation of individual components as part of the structure
- ▶ The basic functions that a computer can perform are:
 - ▶ Data processing
 - ▶ Data storage
 - ▶ Data movement
 - ▶ Control

Structure & Function

The computer must be able:

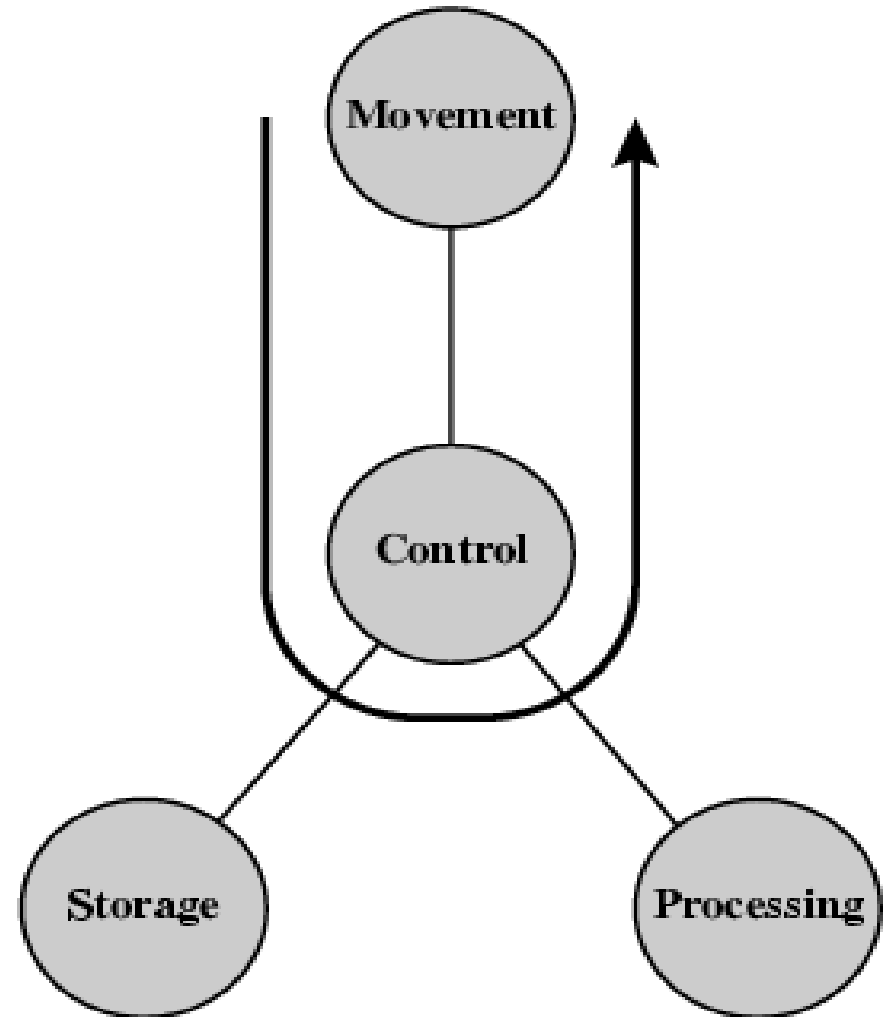
- To **process data**, that may take a wide variety of forms
- To **store data** - temporarily store at least those pieces of data that are being worked on at any given moment
- To **move data** between itself and the outside world
- To **control** these three functions



Operations - Data movement

Four possible types of operations

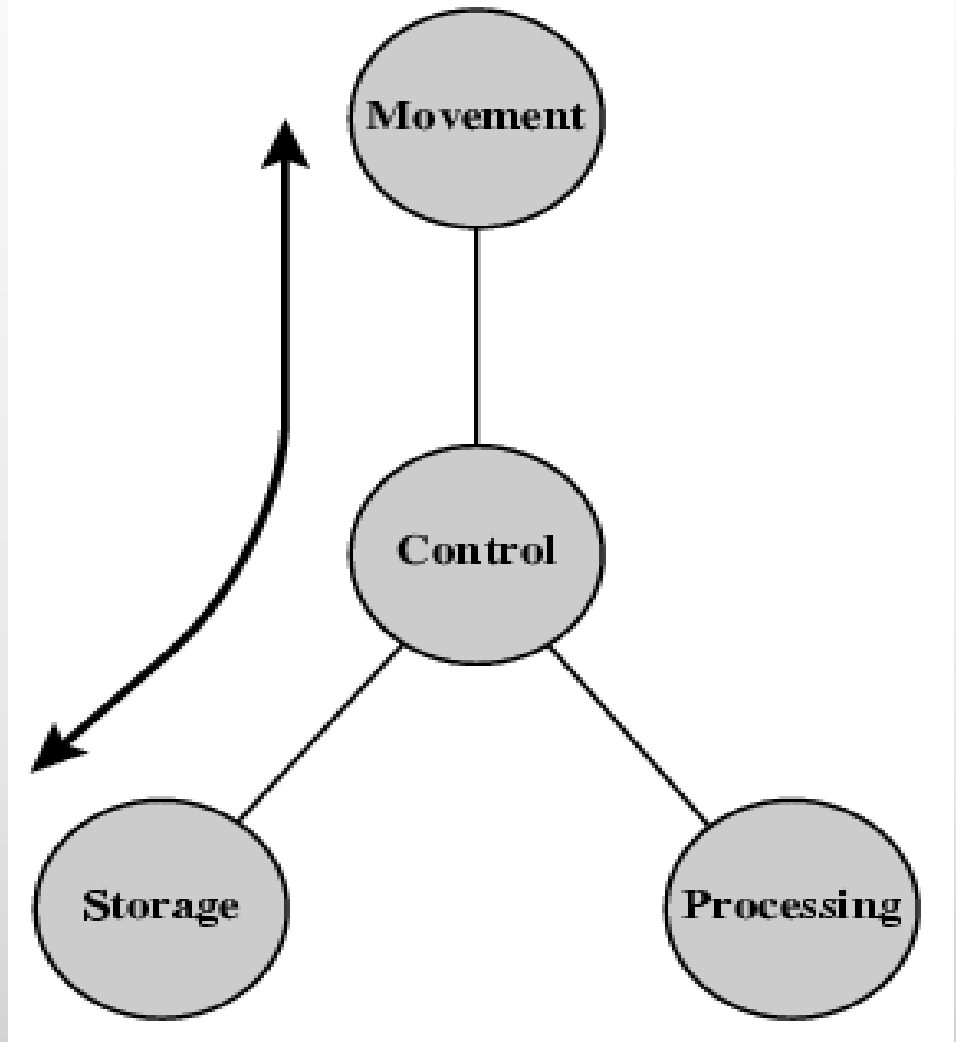
The **computer** can function as a **data movement device**, simply transferring data from one peripheral or communications line to another



Operations - Storage

Four possible types of operations

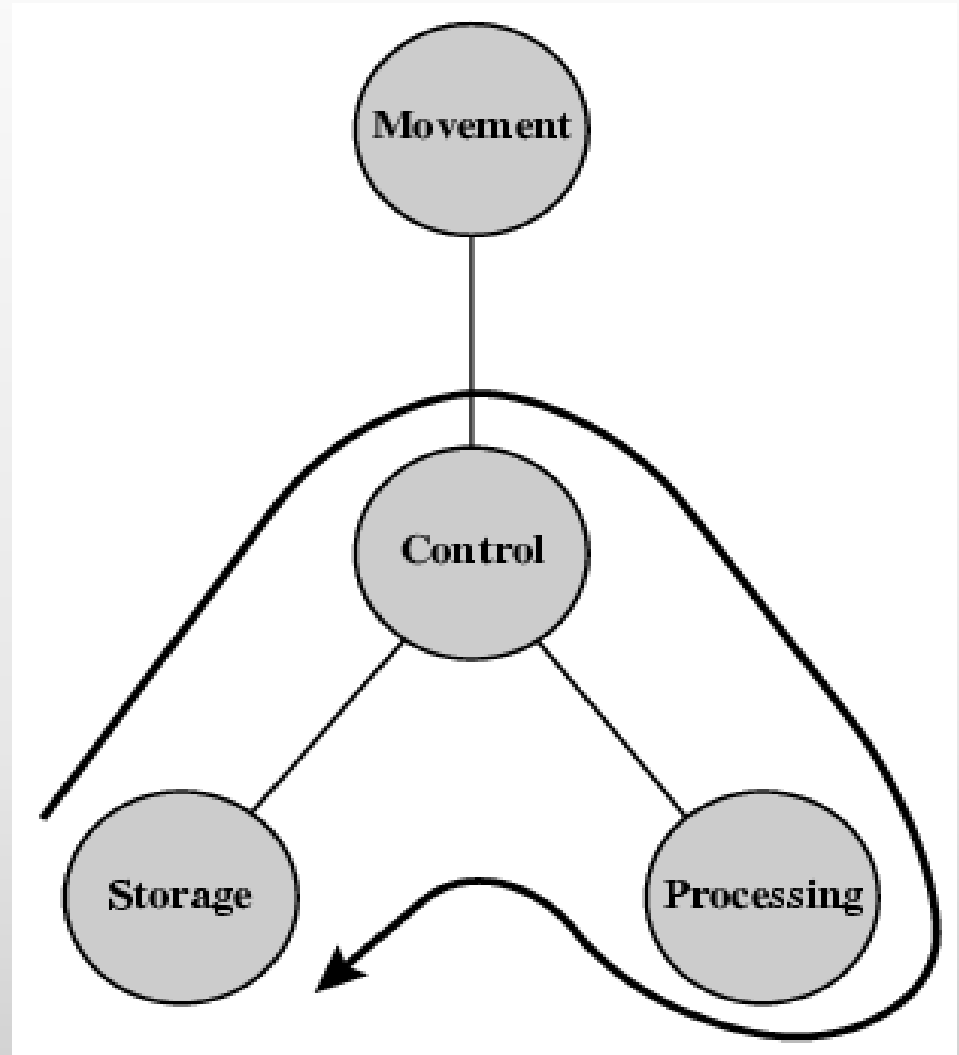
The **computer** can function as a **data data storage device**, with data transferred from the external environment to computer storage (**read**) and vice versa (**write**)



Operation - Processing from/to storage

Four possible types of operations

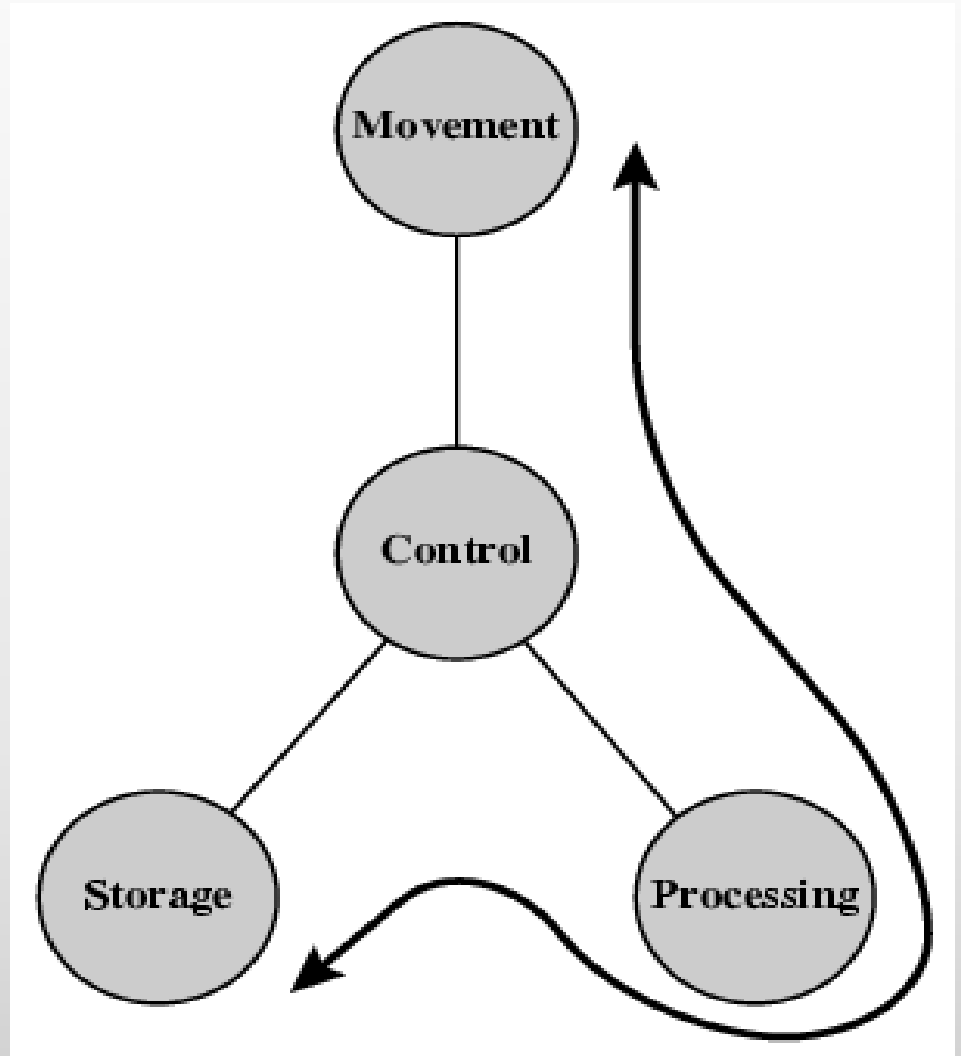
The **computer** can **execute operations** involving data processing, on **data in storage**



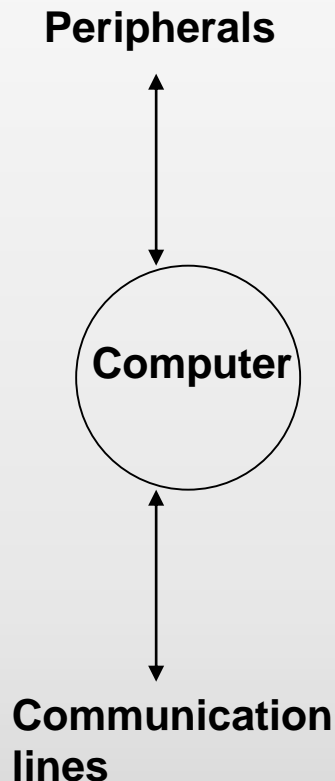
Operation - Processing from storage to I/O

Four possible types of operations

The **computer** can **execute operations** involving data processing, on **data en route** between storage and the external environment



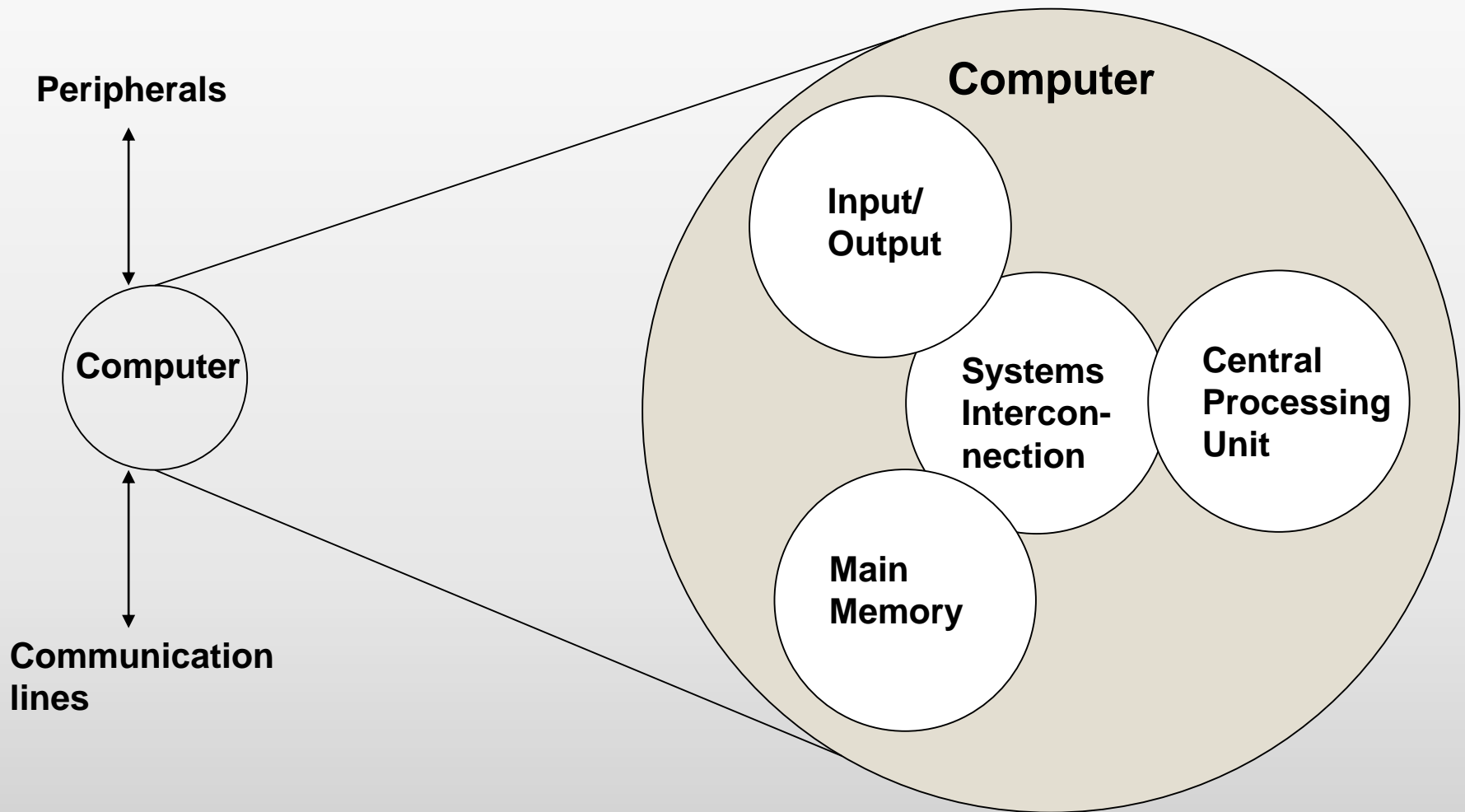
Structure - Top Level



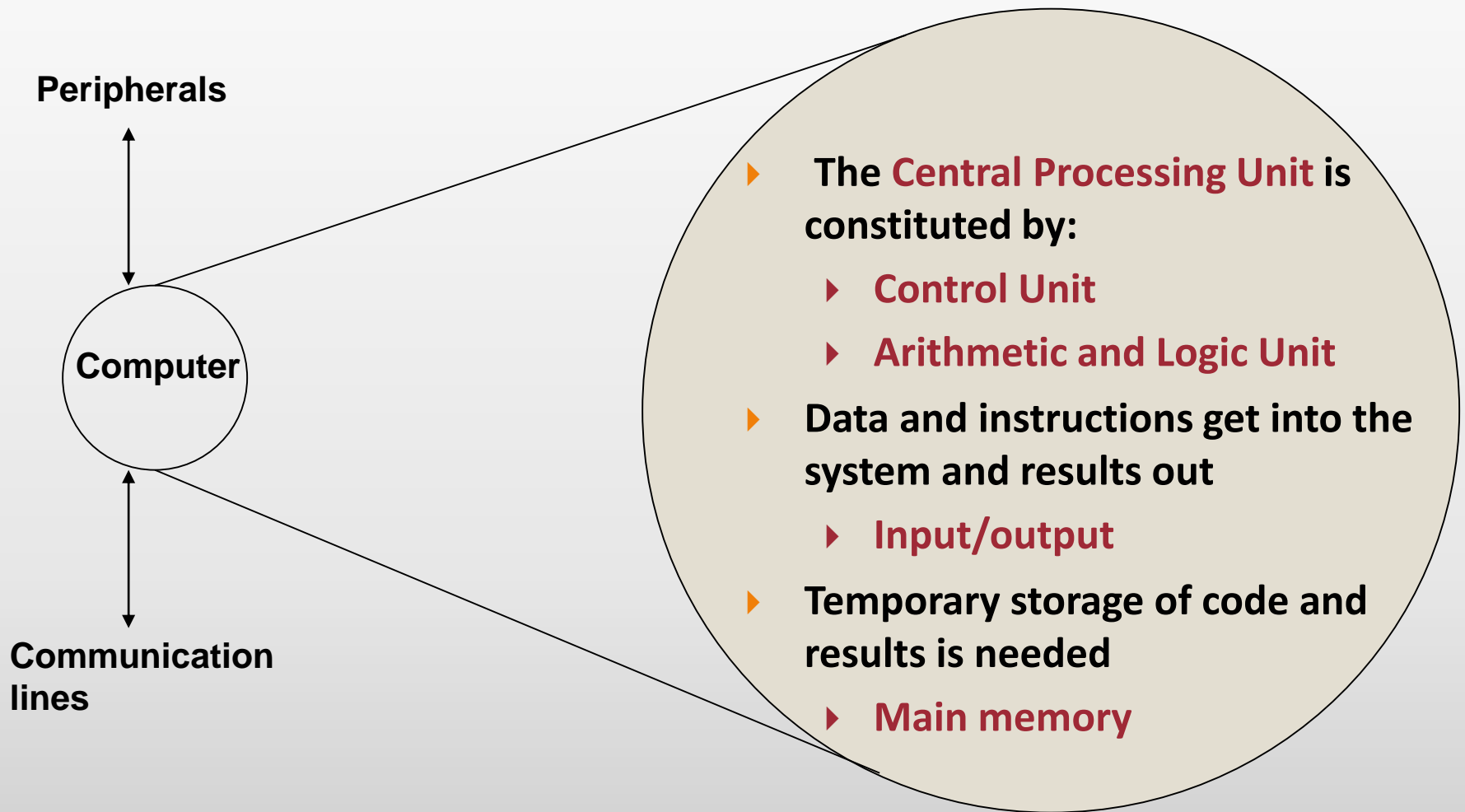
The internal structure of the computer consists of four main structural components:

- ▶ **Central processing unit (CPU):** Controls the operation of the computer and performs its data processing functions (**processor**)
- ▶ **Main memory:** Stores data
- ▶ **I/O:** Moves data between the computer and its external environment
- ▶ **System interconnection:** Some mechanism that provides for communication among CPU, main memory, and I/O (for example a **system bus**)

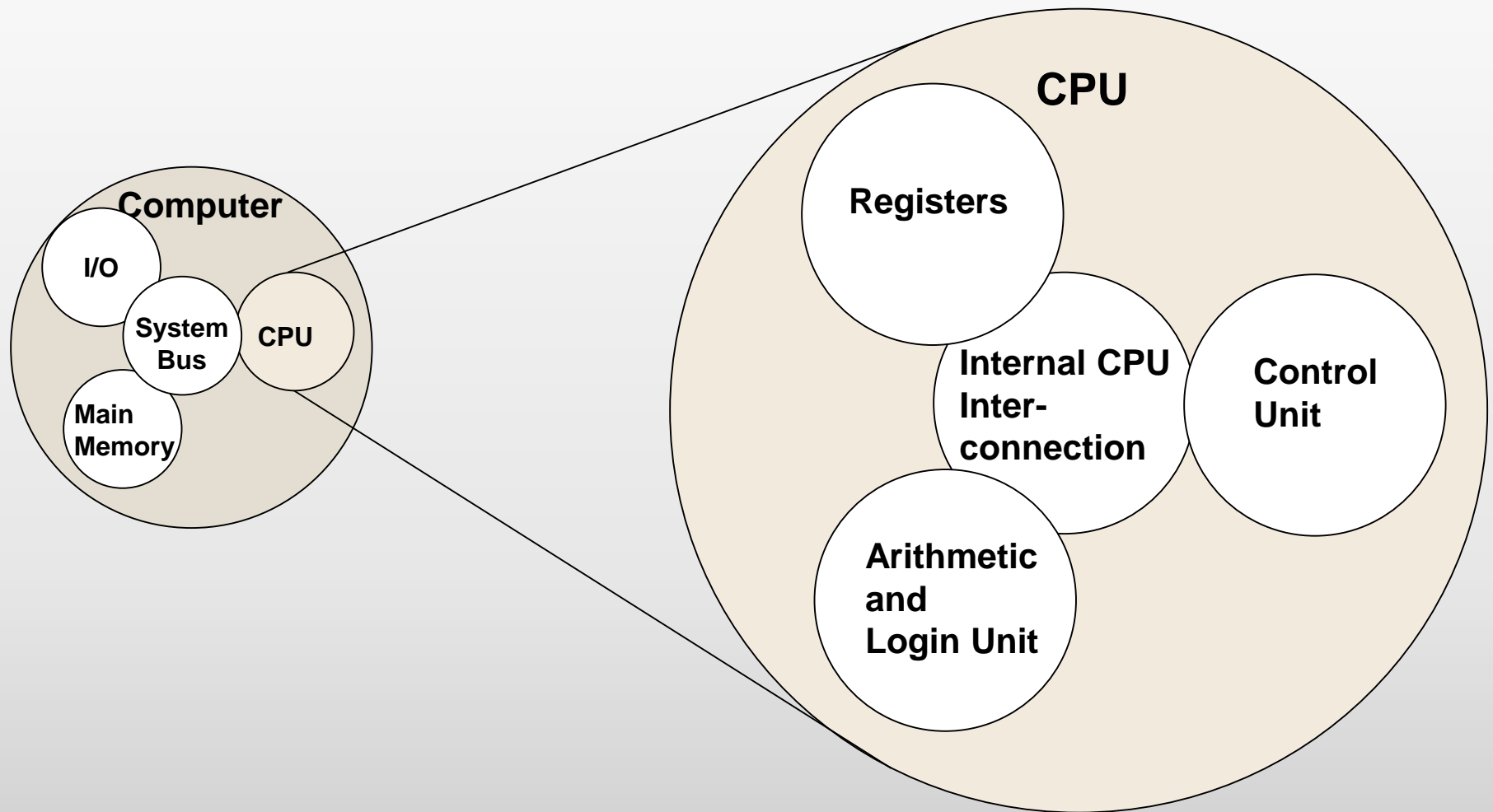
Structure - Top Level



Structure - Top Level



Structure - The CPU

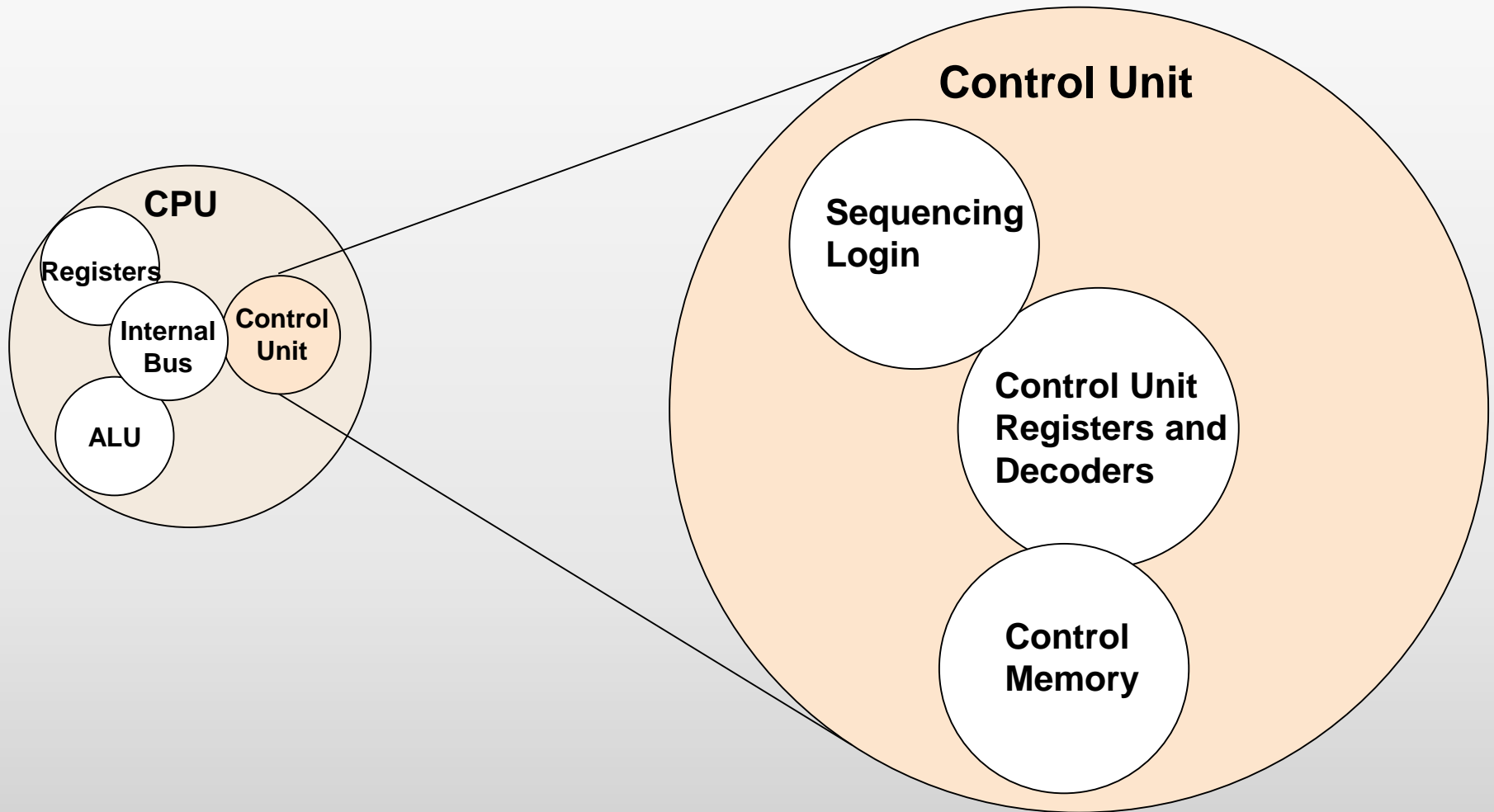


Components

Basic element of a Central Processing Unit (processor)

- ▶ Control Unit
- ▶ ALU Arithmetic and Logic Unit
- ▶ Registers
- ▶ Internal data paths
- ▶ External data paths

Structure - The Control Unit



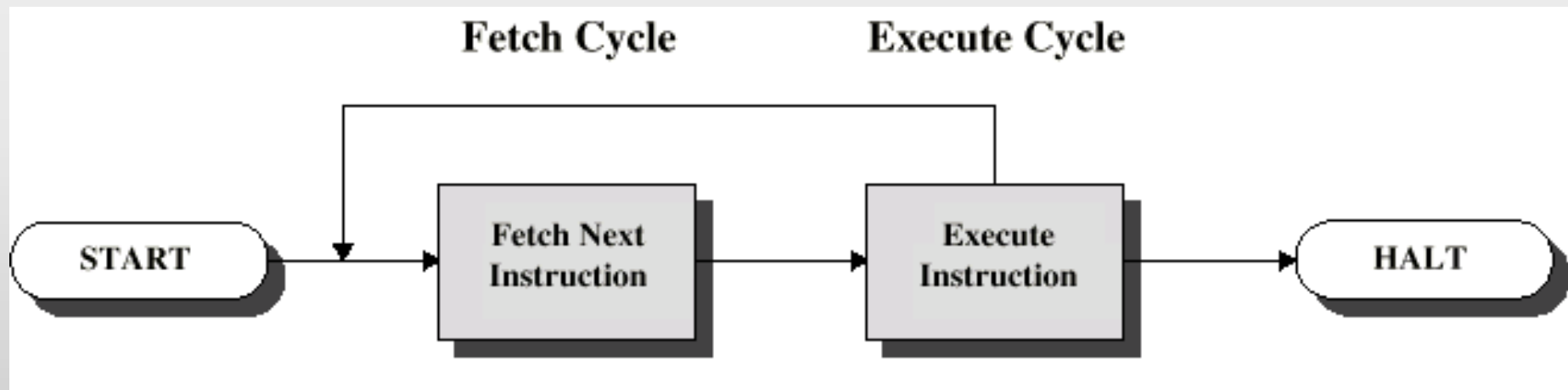
Observations

- ▶ Traditionally, the computer has been viewed as a **sequential** machine
- ▶ Most **computer programming languages** require the programmer to specify **algorithms as sequences of instructions**
- ▶ **Processors** execute programs by executing machine instructions in a **sequence** and one at a time
- ▶ Each **instruction** is executed in a **sequence of operations** (fetch instruction, fetch operands, perform operation, store results)
- ▶ *This view of the computer has never been entirely true*

Instruction execution

Instruction Cycle

- ▶ The processing required for a single instruction is called an *instruction cycle*
- ▶ The instruction cycle can be illustrated using a simplified two-step description
- ▶ The two steps are referred to as the *fetch cycle* and the *execute cycle*



Fetch Cycle

- ▶ Program Counter (PC) holds address of next instruction to fetch
- ▶ Processor fetches instruction from memory location pointed to by PC
- ▶ Increment PC
 - ▶ Unless told otherwise
- ▶ Instruction loaded into Instruction Register (IR)
- ▶ Processor interprets instruction and performs required actions

Execute Cycle

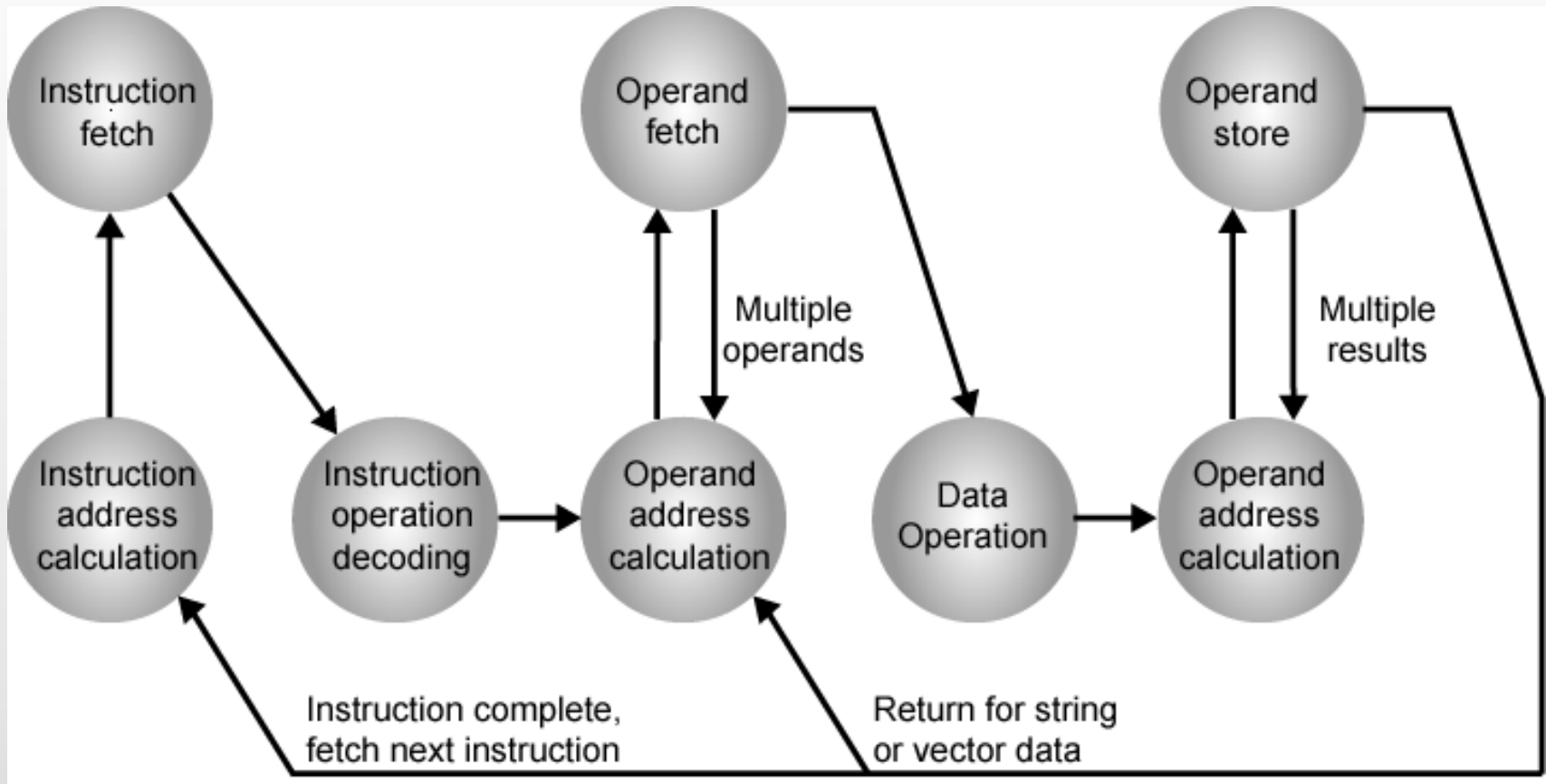
- ▶ Processor-memory
 - ▶ data transfer between CPU and main memory
- ▶ Processor I/O
 - ▶ Data transfer between CPU and I/O module
- ▶ Data processing
 - ▶ Some arithmetic or logical operation on data
- ▶ Control
 - ▶ Alteration of sequence of operations
 - ▶ e.g. jump
- ▶ Combination of above

Instruction Execution

The requirements placed on the processor (that is the things that it must do):

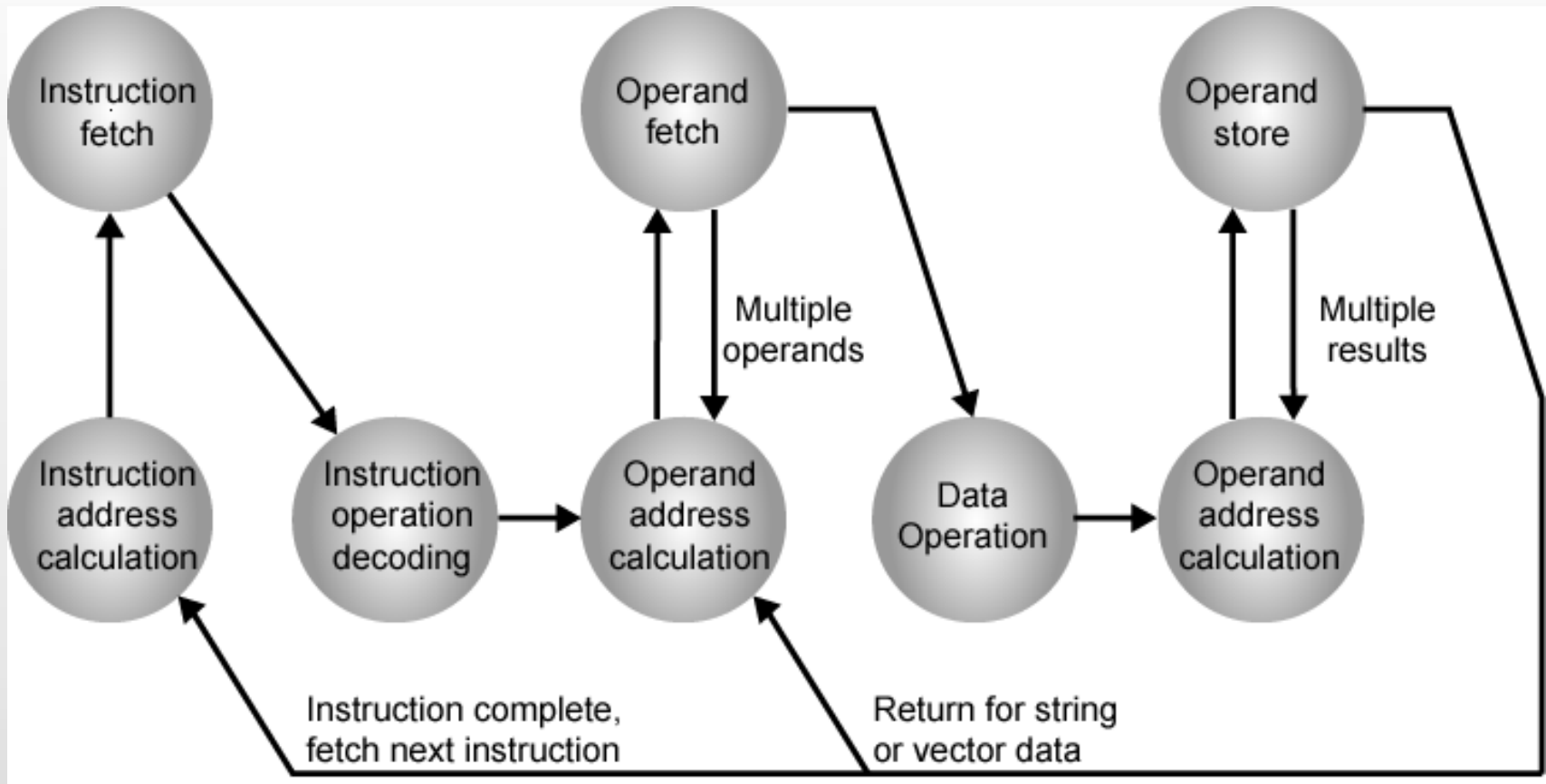
- ▶ **Fetch instruction:** The processor reads an instruction from memory (register, cache, main memory)
- ▶ **Interpret instruction:** The instruction is decoded to determine what action is required
- ▶ **Fetch data:** The execution of an instruction may require reading data from memory or an I/O module
- ▶ **Process data:** The execution of an instruction may require performing some arithmetic or logical operation on data.
- ▶ **Write data:** The results of an execution may require writing data to memory or an I/O module

Instruction Cycle State Diagram



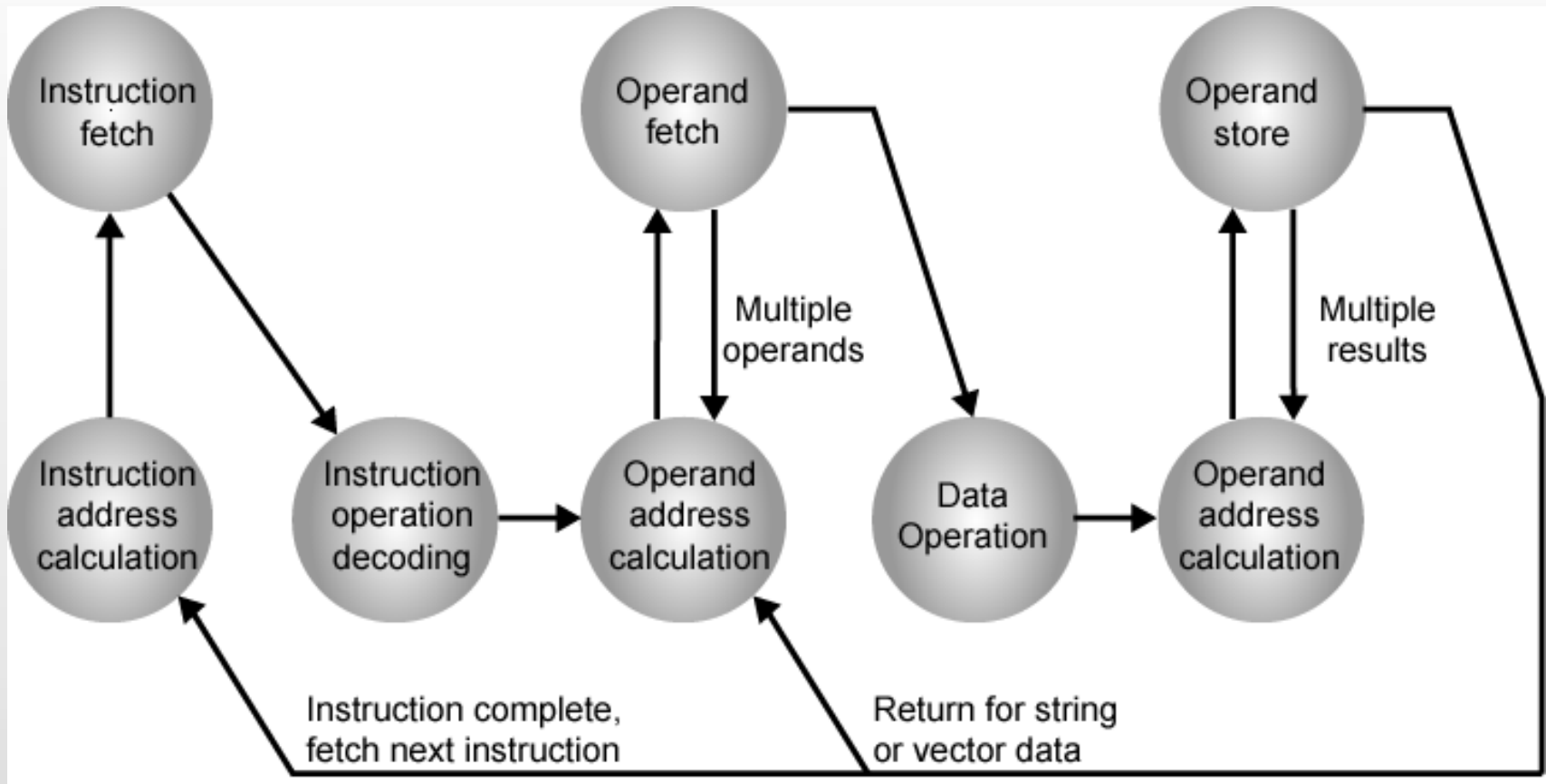
Instruction address calculation (iac): Determine the address of the next instruction to be executed (usually, adding a fixed number to the address of the previous instr.)

Instruction Cycle State Diagram



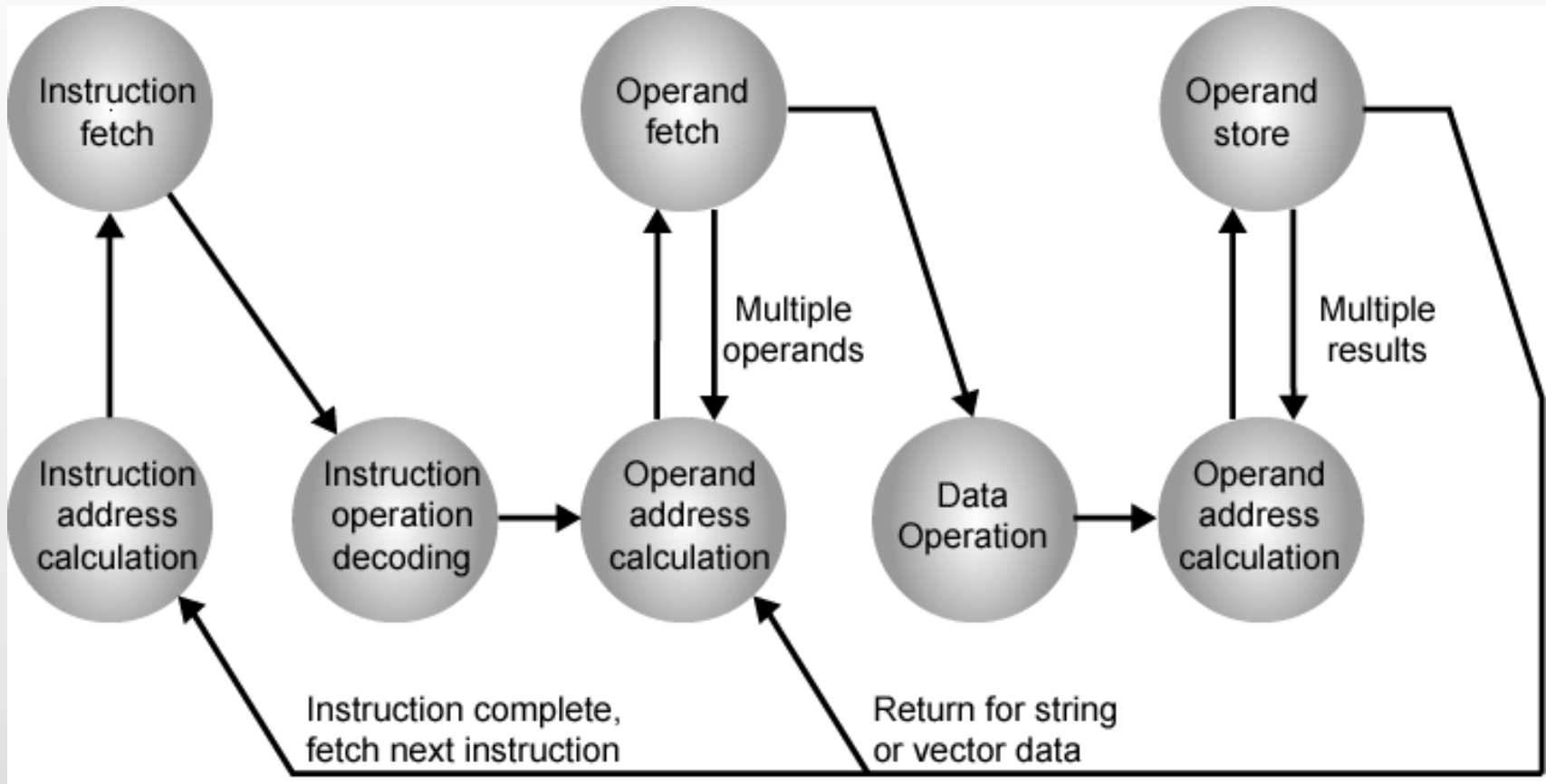
Instruction fetch (if): Read instruction from its memory location into the processor

Instruction Cycle State Diagram



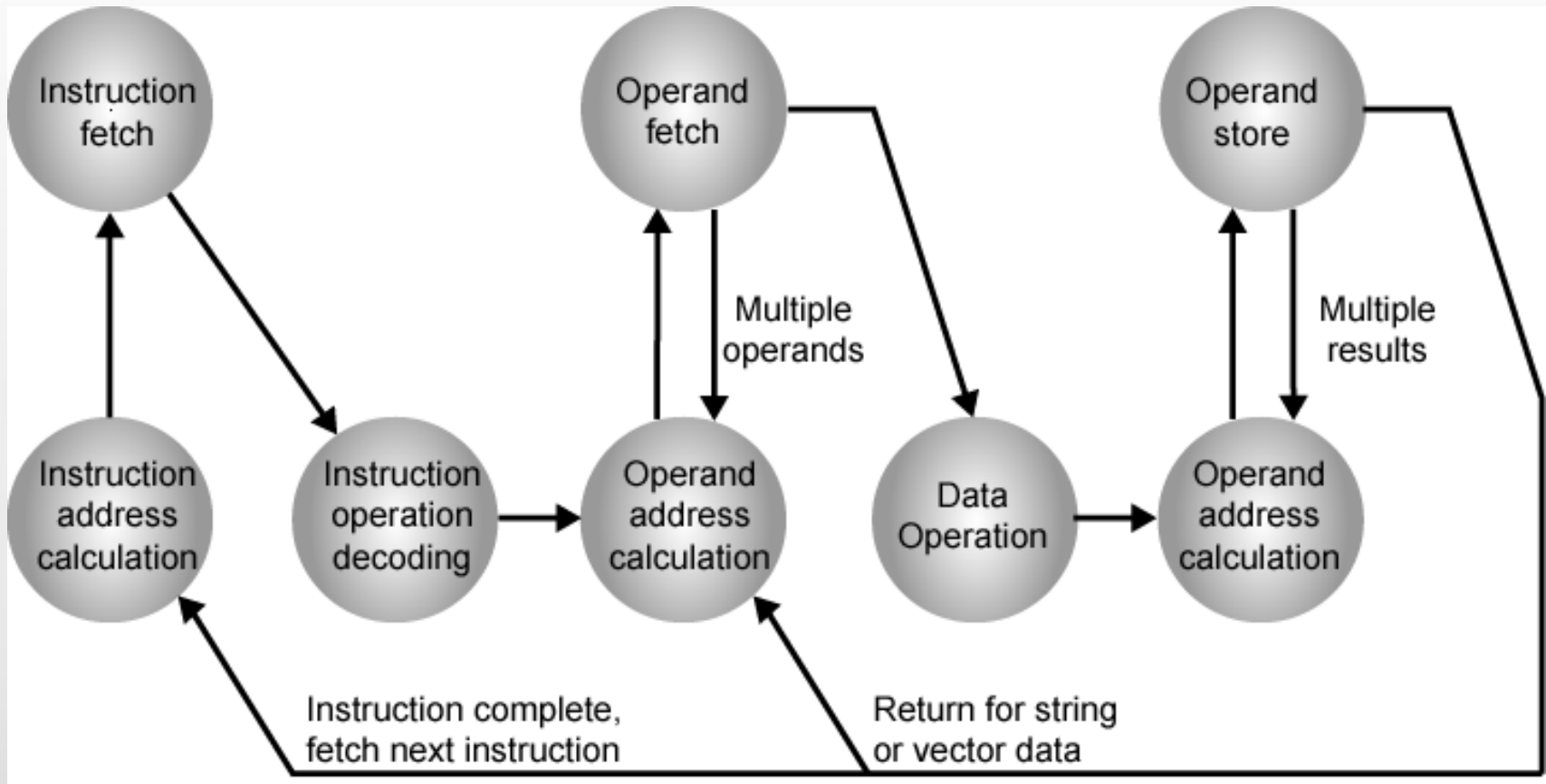
Instruction operation decoding (iod): Analyze instruction to determine type of operation to be performed and operand(s) to be used

Instruction Cycle State Diagram



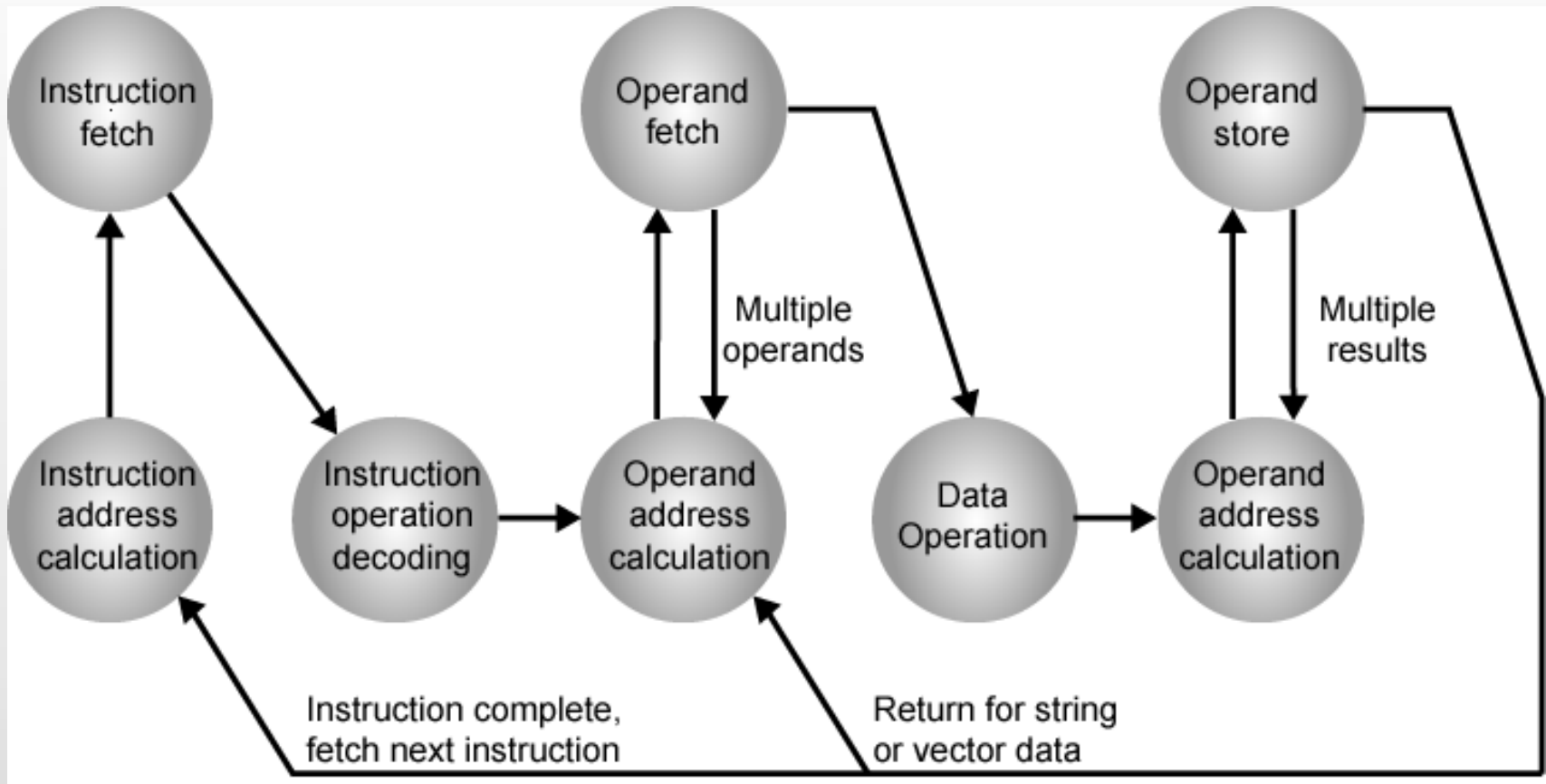
Operand address calculation (oac): If the operation involves reference to an operand in memory or available via I/O, then determine the address of the operand

Instruction Cycle State Diagram



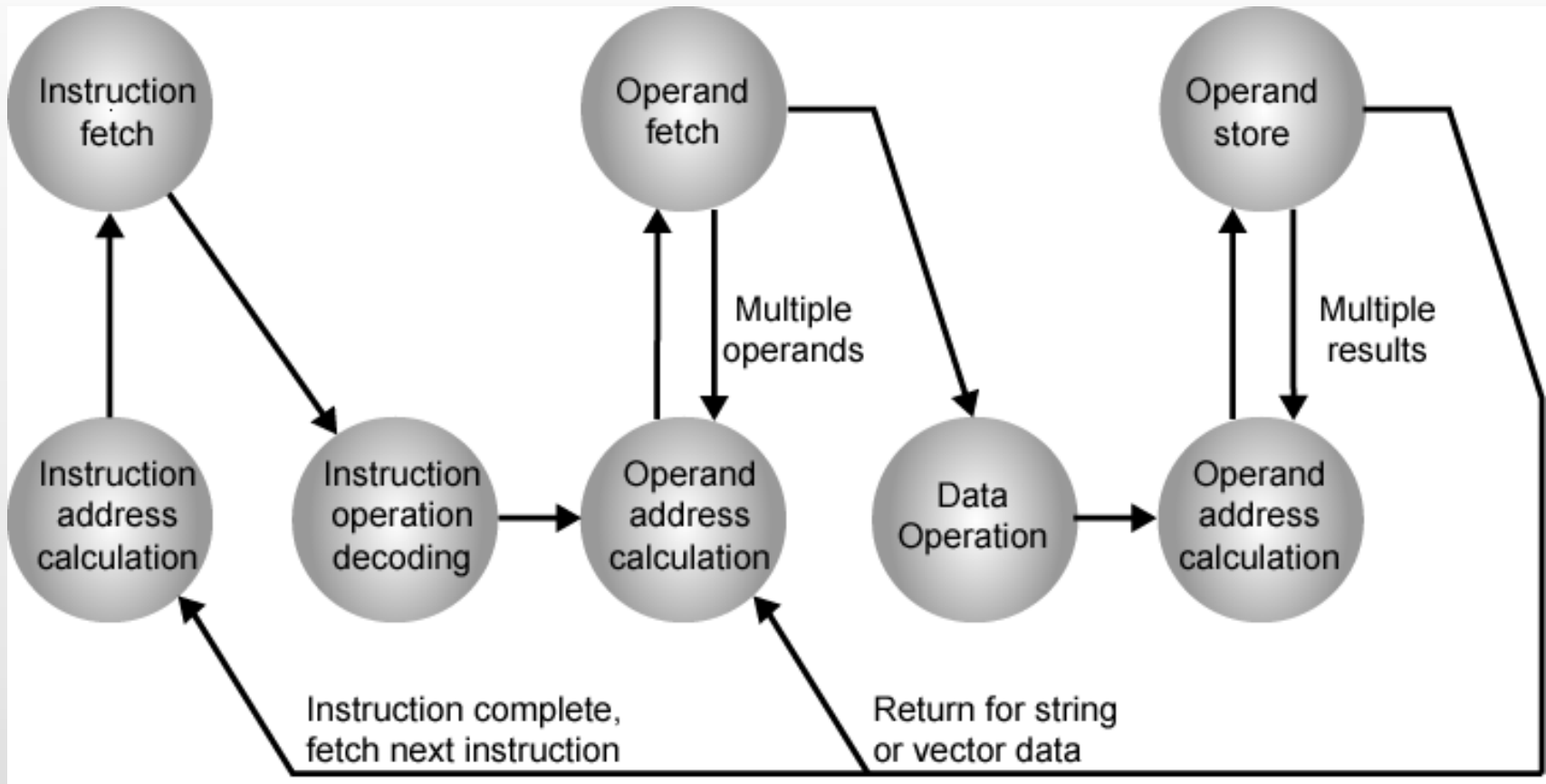
Operand fetch (of): Fetch the operand from memory or read it in from I/O

Instruction Cycle State Diagram



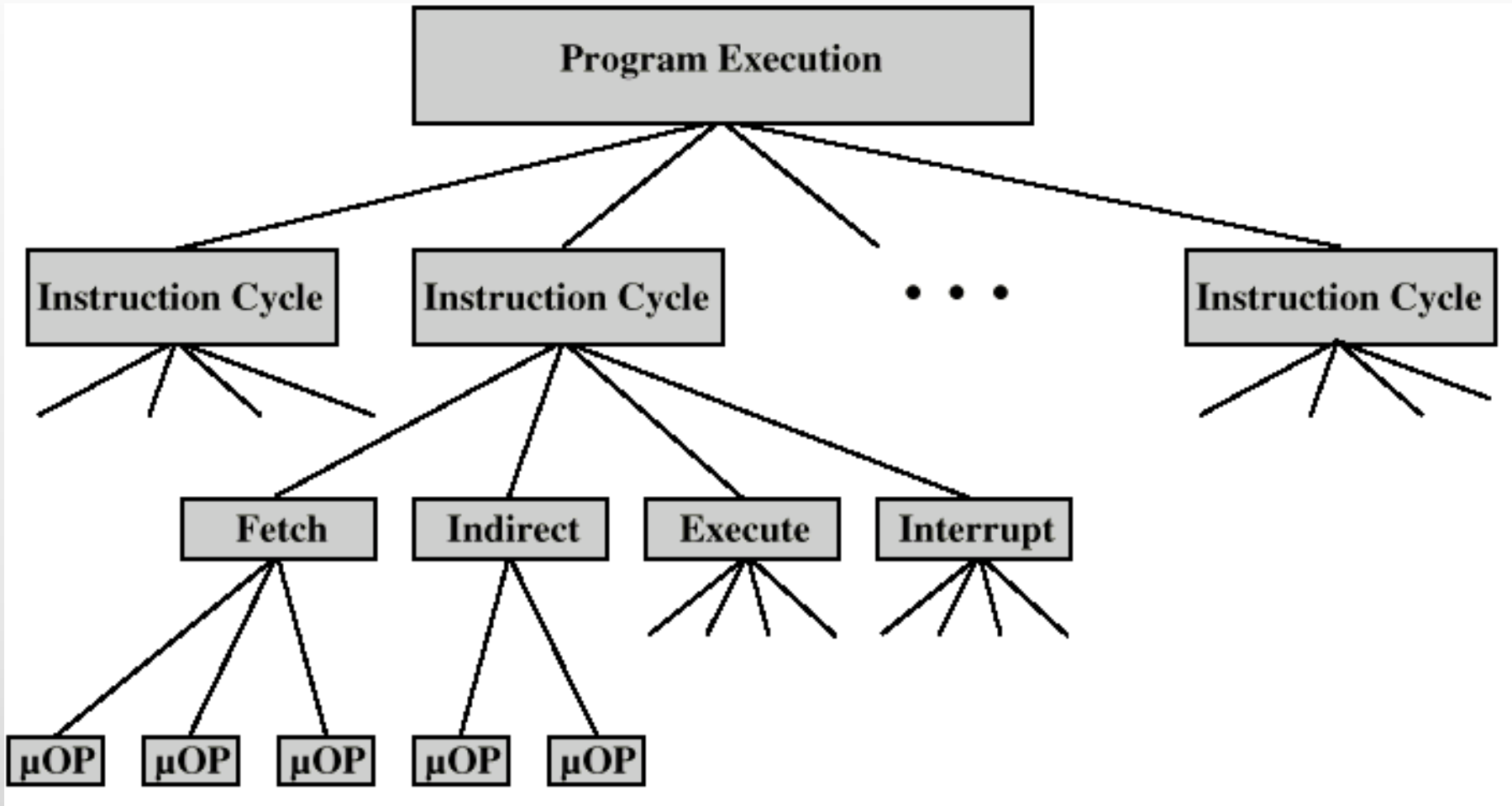
Data operation (do): Perform the operation indicated in the instruction

Instruction Cycle State Diagram



Operand store (os): Write the result into memory or out to I/O

Constituent Elements of Program Execution



The instruction cycle is decomposed into sequence of elementary *micro-operations*

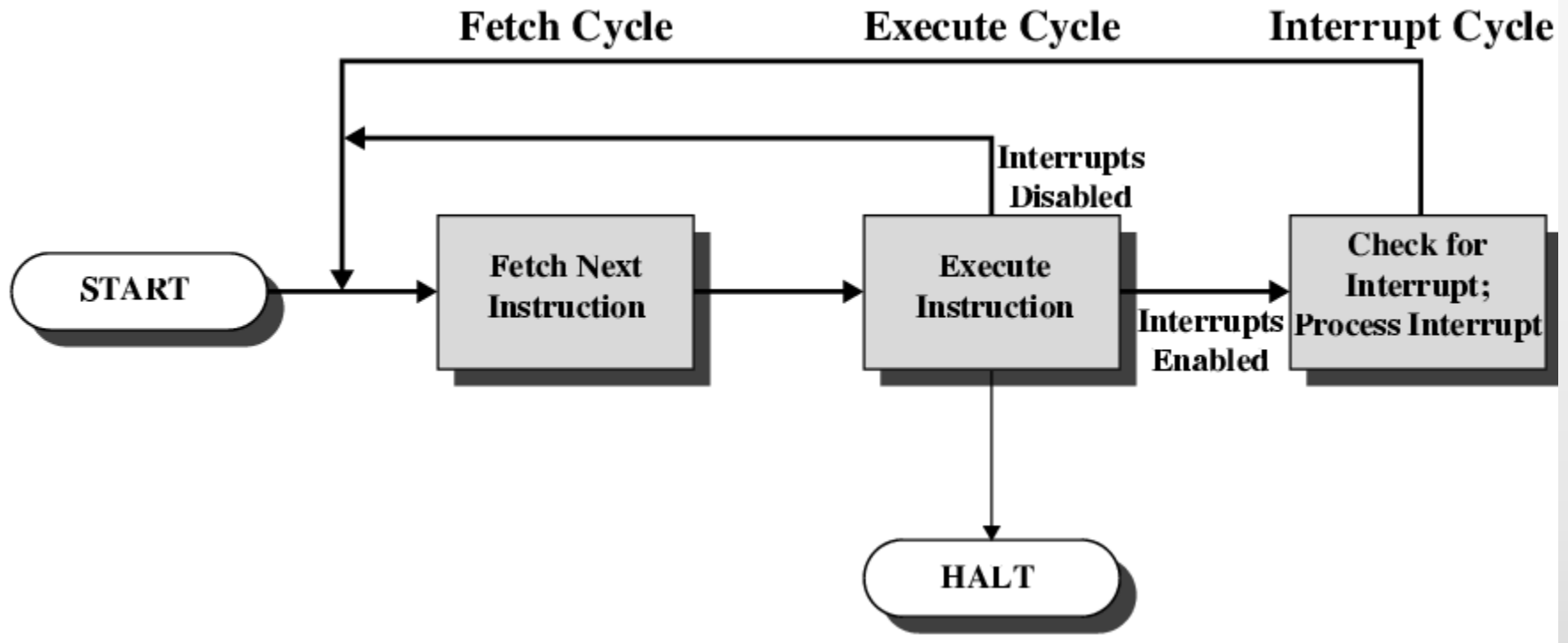
Interrupts

- ▶ *Mechanism by which other modules may interrupt normal sequence of processing*
- ▶ Program
 - ▶ e.g. overflow, division by zero
- ▶ Timer
 - ▶ Generated by internal processor timer
 - ▶ Used in pre-emptive multi-tasking
- ▶ I/O
 - ▶ from I/O controller
- ▶ Hardware failure
 - ▶ e.g. memory parity error

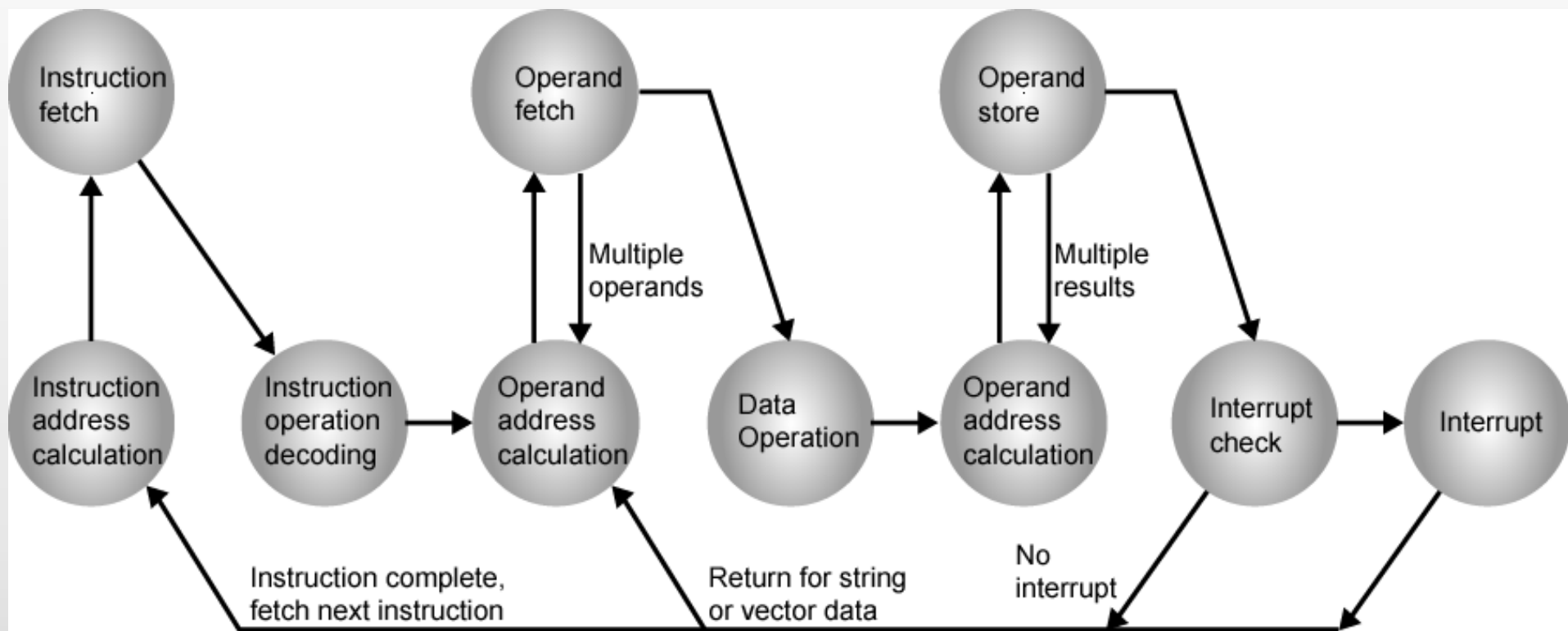
Interrupt Cycle

- ▶ ***Added to instruction cycle***
- ▶ Processor checks for interrupt
 - ▶ Indicated by an interrupt signal
- ▶ If no interrupt, fetch next instruction
- ▶ If interrupt pending:
 - ▶ Suspend execution of current program
 - ▶ Save context
 - ▶ Set PC to start address of interrupt handler routine
 - ▶ Process interrupt
 - ▶ Restore context and continue interrupted program

Instruction Cycle with Interrupts



Instruction Cycle with Interrupts - State Diagram



Multiple Interrupts

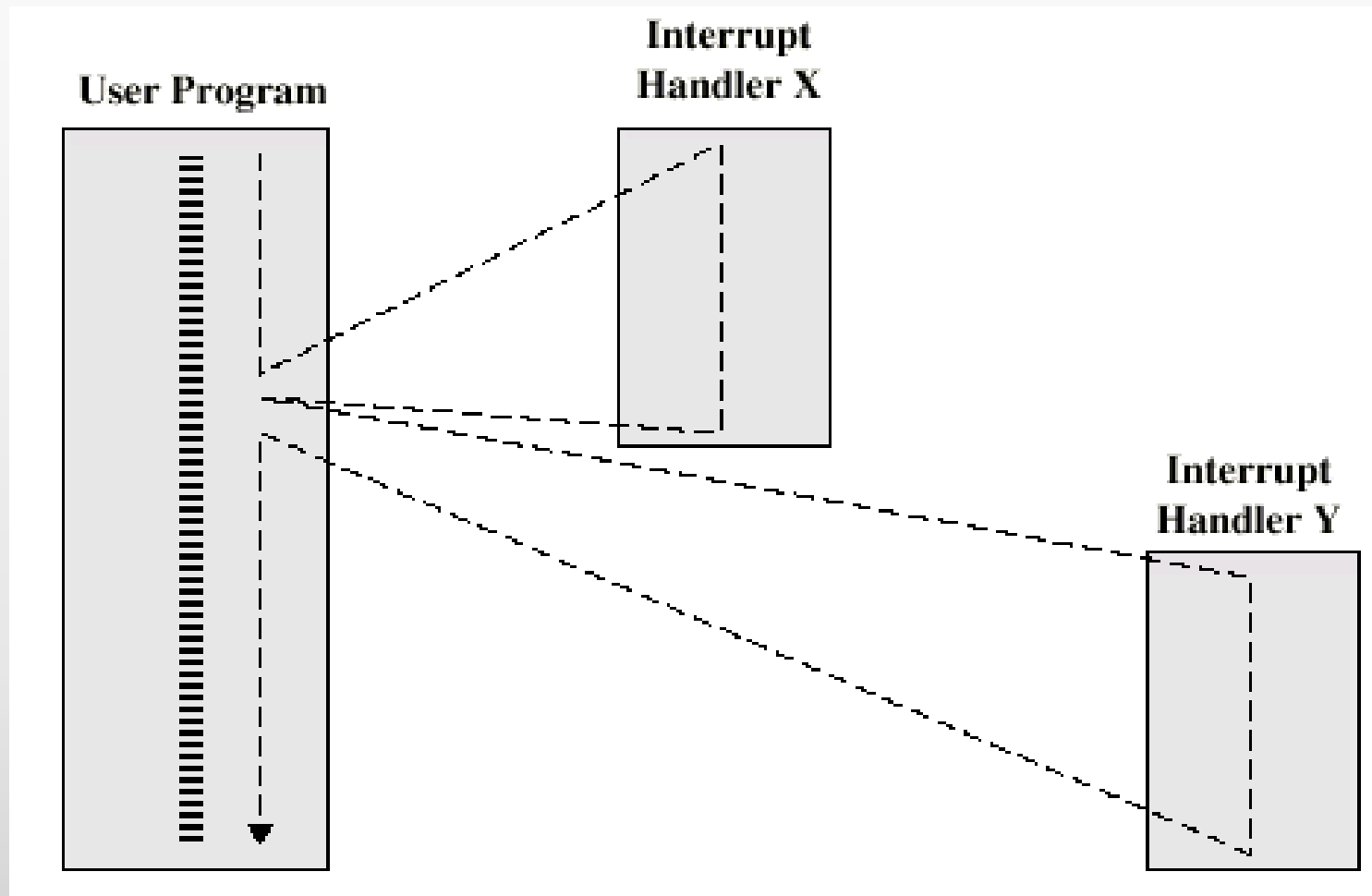
▶ Disable interrupts

- ▶ Processor will ignore further interrupts whilst processing one interrupt
- ▶ Interrupts remain pending and are checked after first interrupt has been processed
- ▶ Interrupts handled in sequence as they occur

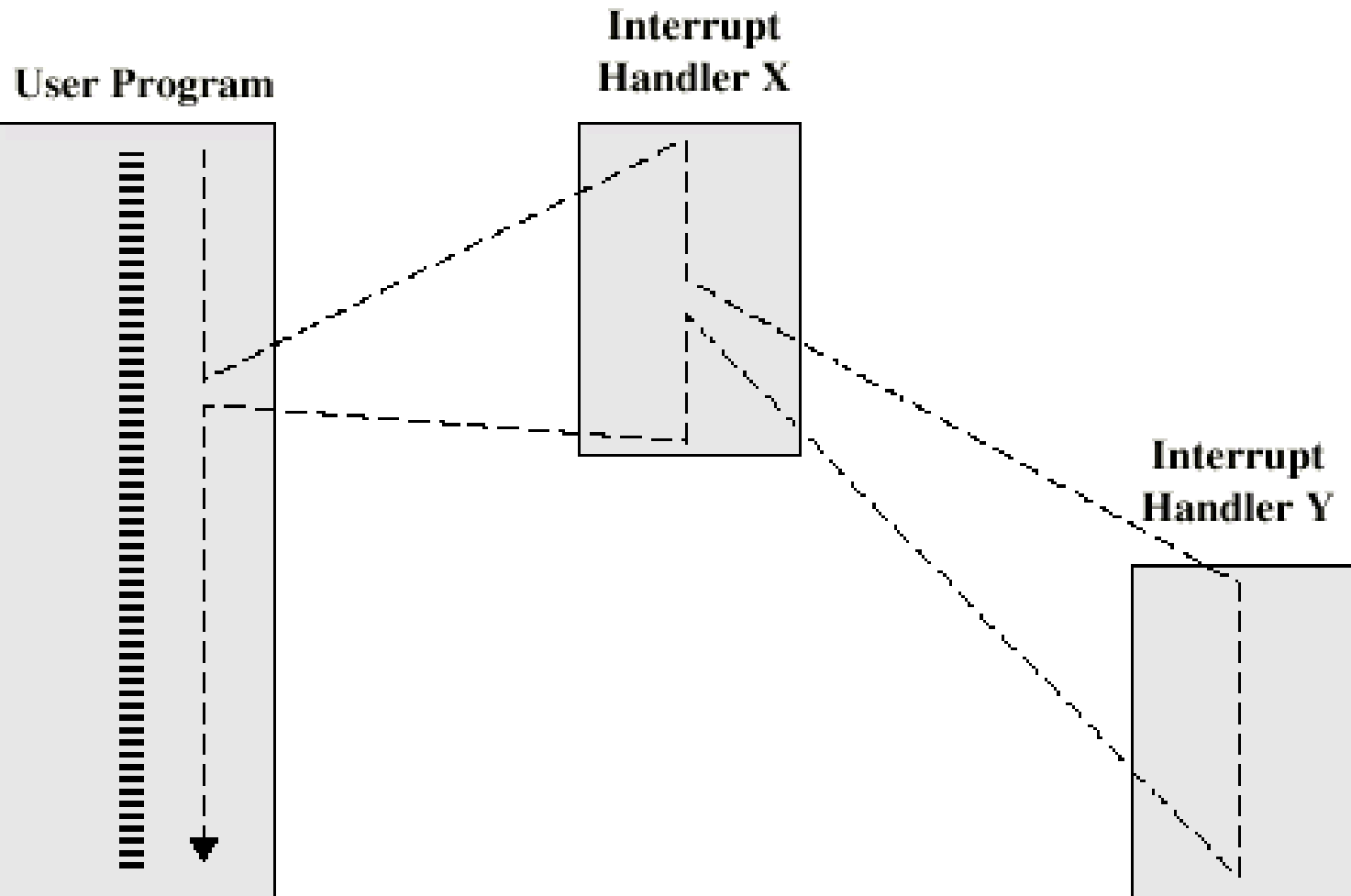
▶ Define priorities

- ▶ Low priority interrupts can be interrupted by higher priority interrupts
- ▶ When higher priority interrupt has been processed, processor returns to previous interrupt

Multiple Interrupts - Sequential



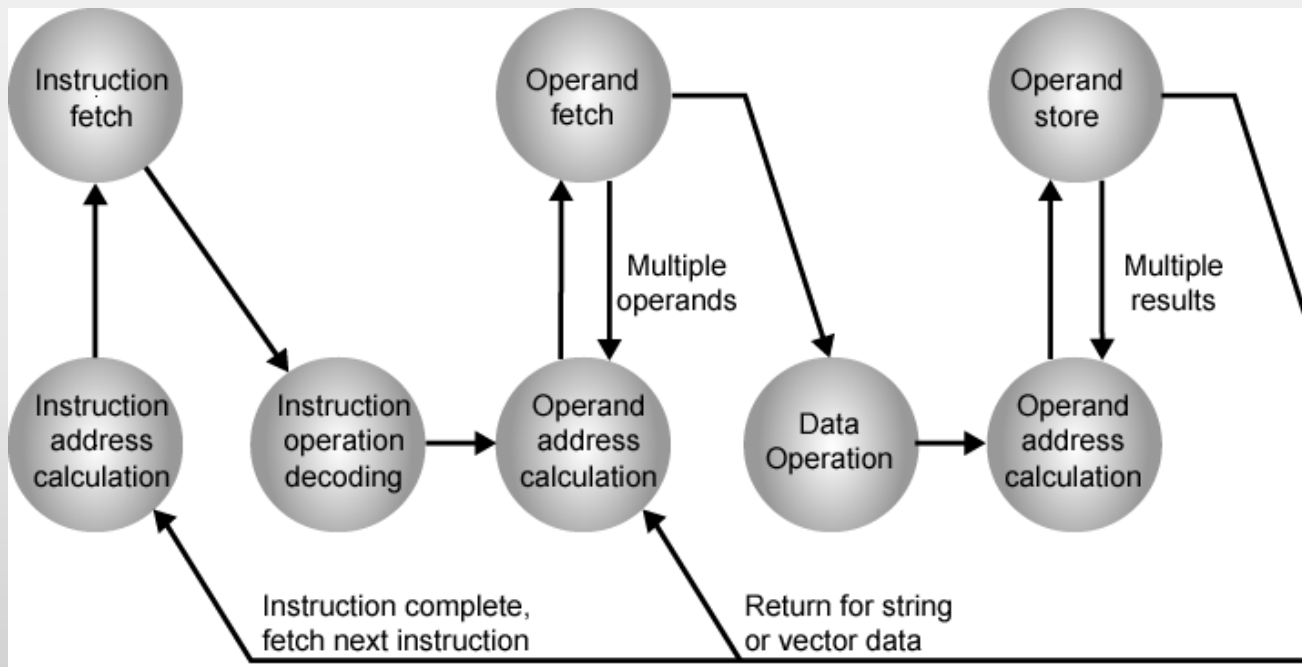
Multiple Interrupts – Nested



Instruction characteristics

Instruction Set

- ▶ The **instruction set** is the complete collection of instructions that the processor can execute
- ▶ Each **instruction must contain the information** required by the processor for execution



Elements of an Instruction

Elements of a machine instruction are:

- ▶ **Operation code (Op code)**
 - ▶ Do this
- ▶ **Source Operand reference**
 - ▶ To this
- ▶ **Result Operand reference**
 - ▶ Put the answer here
- ▶ **Next Instruction Reference**
 - ▶ When you have done that, do this...

Instruction Representation

- ▶ In machine code each instruction has a unique bit pattern
- ▶ For human understanding (e.g., programmers) a ***symbolic representation*** is used
 - ▶ **Opcodes** - e.g. ADD, SUB, LOAD
- ▶ Operands can also be represented symbolically
 - ▶ ADD A,B

Instruction Types

We can categorize instruction types as follows:

- ▶ **Data processing** - Arithmetic and logic instructions
- ▶ **Data storage** (main memory) - Movement of data into or out of register and or memory locations
- ▶ **Data movement** - I/O instructions
- ▶ **Control** - Test and branch instructions

Number of Addresses

- ▶ An instruction could plausibly be required to contain four address references:
 - ▶ two source operands
 - ▶ one destination operand
 - ▶ the address of the next instruction
- ▶ In most architectures, most instructions have:
 - ▶ one, two, or three **operand addresses**,
 - ▶ address of the **next instruction implicit** (obtained from the program counter)

Number of Addresses

- ▶ 3 addresses (not common)
 - ▶ Operand 1, Operand 2, Result -> $a = b + c$;
 - ▶ May be a forth - next instruction (usually implicit)
 - ▶ Needs very long words to hold everything
- ▶ Three-address instruction formats are not common because they require a relatively long instruction format to hold the three address references

Number of Addresses

▶ 2 addresses

- ▶ One address doubles as operand and result - $a = a + b$
- ▶ Reduces length of instruction
- ▶ Requires some extra work (to avoid altering the value of an operand, a MOVE instruction is used to move one of the values to a temporary location before performing the operation)

▶ 1 address (Common on early machines)

- ▶ Implicit second address - Usually a register (accumulator)

▶ 0 addresses

- ▶ All addresses implicit - applicable to a memory organization as a stack

How Many Addresses

- ▶ More addresses

- ▶ More complex (powerful?) instructions
- ▶ More registers
 - ▶ Inter-register operations are quicker
- ▶ Fewer instructions per program

- ▶ Fewer addresses

- ▶ Less complex (powerful?) instructions
- ▶ More instructions per program
- ▶ Faster fetch/execution of instructions

Instruction Set Design

- ▶ The design of an instruction set is very complex because it affects so many aspects of the computer system
- ▶ The instruction set:
 - ▶ **defines** many of the **functions** performed by the processor
 - ▶ **has** a significant effect on the **implementation** of the processor
 - ▶ **is** the programmer's means of controlling the processor

Instruction Set Design

The most important of fundamental design issues include:

- ▶ **Operation repertoire**

- ▶ How many and which operations to provide, and how complex operations should be

- ▶ **Data types**

- ▶ Various types of data upon which operations are performed

- ▶ **Instruction formats**

- ▶ Instruction length (in bits)
- ▶ Number of addresses
- ▶ Size of various fields

- ▶ **Registers**

- ▶ Number of CPU registers available

- ▶ Which operations can be performed on which registers?

Instruction Set Design

The most important of fundamental design issues include:

- ▶ **Registers**

- ▶ Number of CPU registers available
- ▶ Which operations can be performed on which registers

- ▶ **Addressing modes**

- ▶ The modes by which the address of an operand is specified

- ▶ **RISC v CISC**

Types of Operand

- ▶ Machine instructions operate on data
- ▶ The most important general categories of data are:
 - ▶ **Addresses**
 - ▶ **Numbers**
 - ▶ Integer/floating point
 - ▶ **Characters**
 - ▶ ASCII etc.
 - ▶ **Logical Data**
 - ▶ Bits or flags

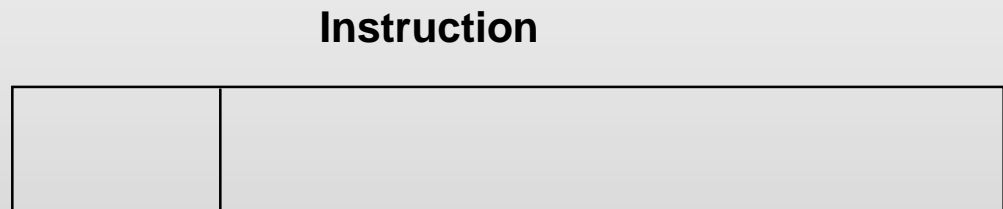
Addressing Modes and Formats

Addressing Modes

- ▶ The address field or fields in a typical instruction format are relatively small
- ▶ We would like to be able to reference a large range of locations in main memory
- ▶ A variety of addressing techniques has been employed
 - ▶ Immediate
 - ▶ Direct
 - ▶ Indirect
 - ▶ Register
 - ▶ Register Indirect
 - ▶ Displacement (Indexed)
 - ▶ Stack

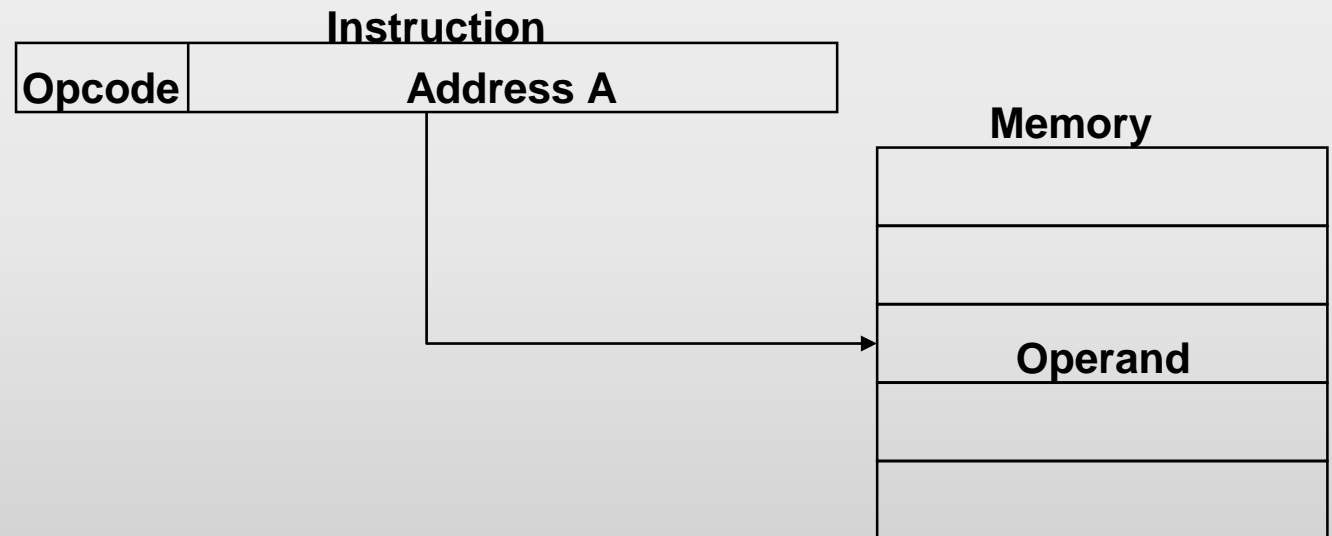
Immediate Addressing

- ▶ Operand is part of instruction
- ▶ Operand = address field
- ▶ e.g. ADD 5
 - ▶ Add 5 to contents of accumulator
 - ▶ 5 is operand
- ▶ No memory reference to fetch data
- ▶ Fast
- ▶ Limited range



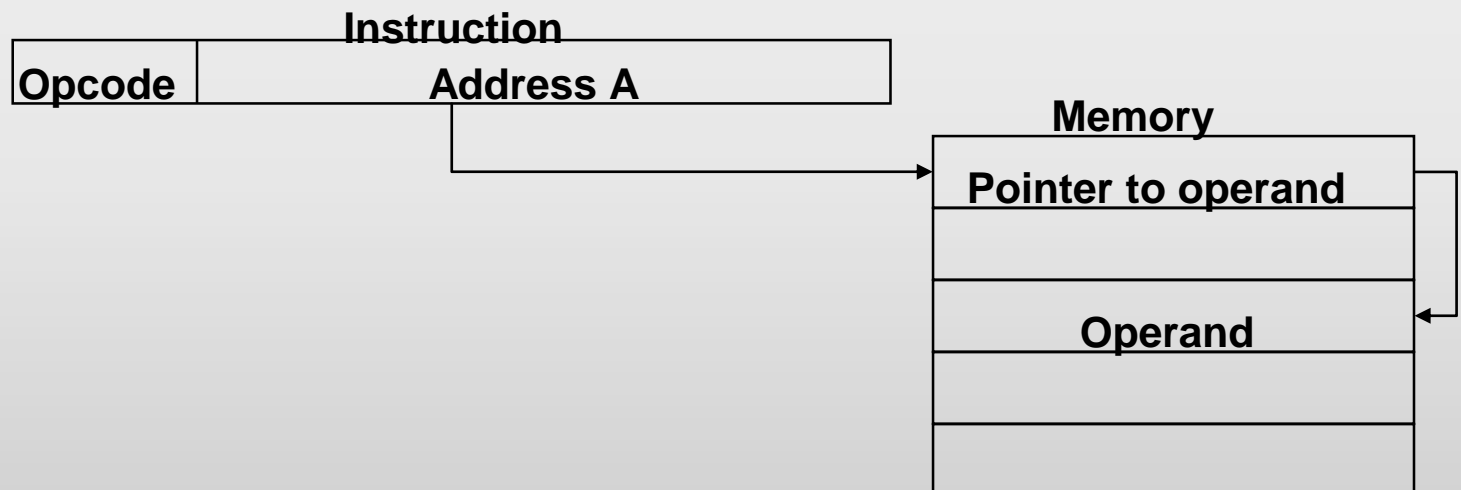
Direct Addressing

- ▶ Address field contains address of operand
- ▶ Effective address (EA) = address field (A)
- ▶ Single memory reference to access data
- ▶ No additional calculations to work out effective address
- ▶ Limited address space



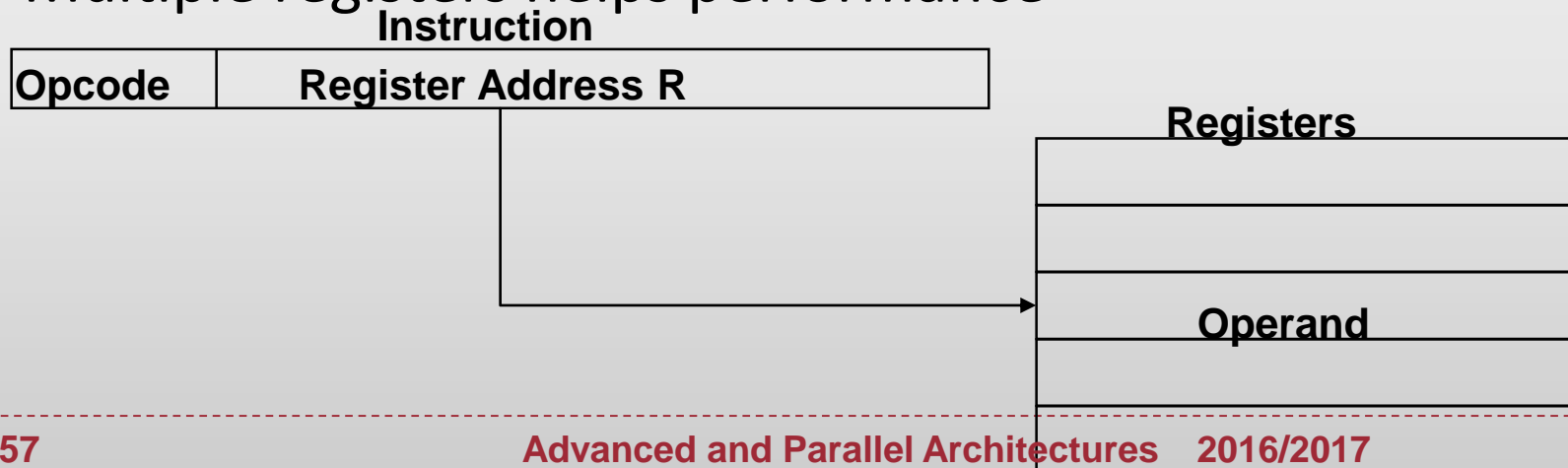
Indirect Addressing

- ▶ Memory cell pointed to by address field contains the address of (pointer to) the operand
- ▶ $EA = (A)$
 - ▶ Look in A, find address (A) and look there for operand
- ▶ Large address space - 2^n where n = word length
- ▶ Multiple memory accesses to find operand - hence slower



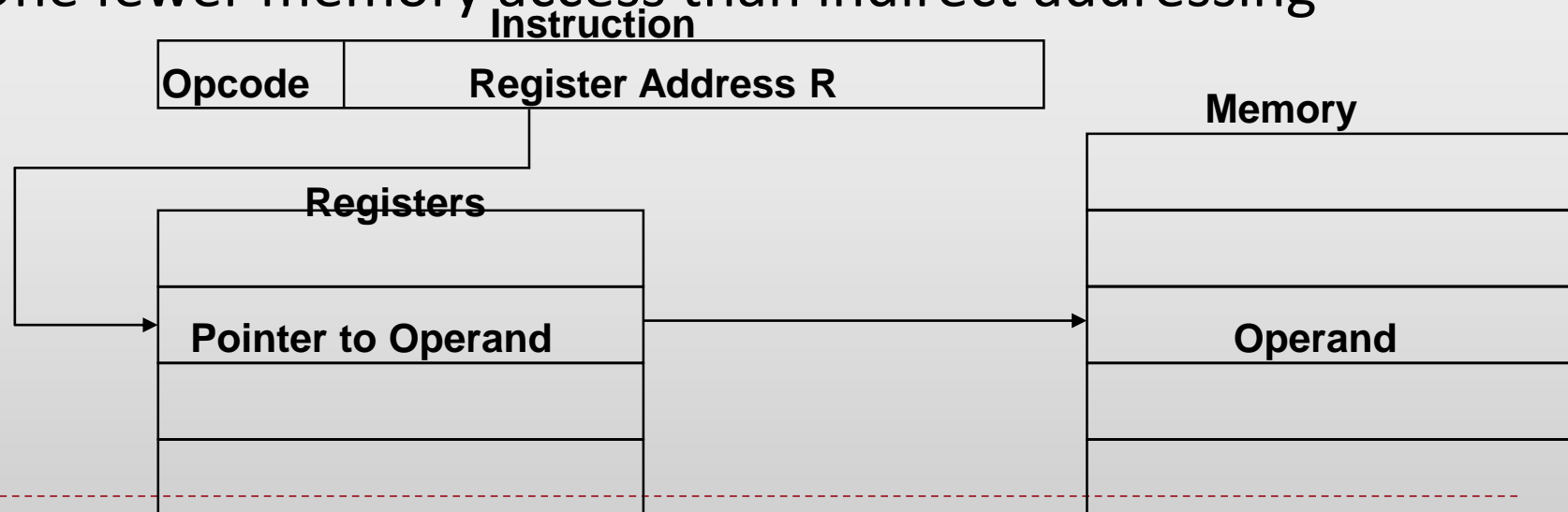
Register Addressing

- ▶ Operand is held in register named in address field
- ▶ Limited number of registers
- ▶ Very small address field needed
 - ▶ Shorter instructions - Faster instruction fetch
- ▶ No memory access - Very fast execution
- ▶ Very limited address space
- ▶ Multiple registers helps performance



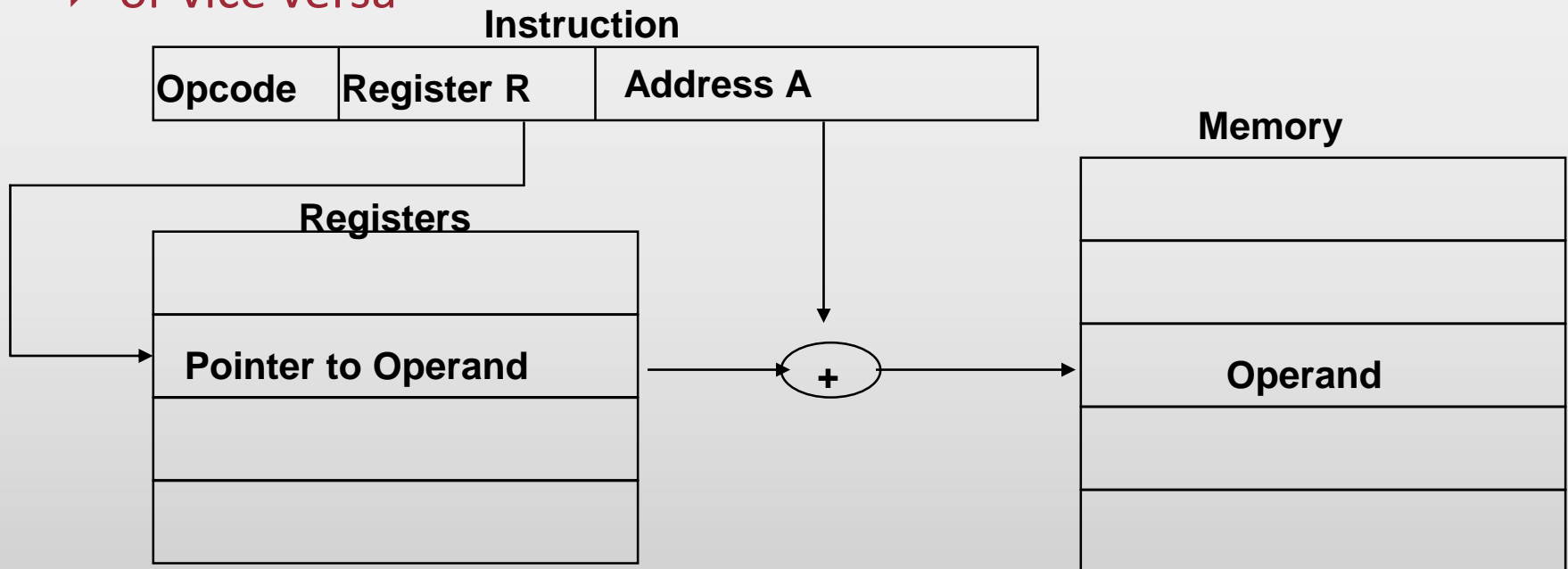
Register Indirect Addressing

- ▶ C.f. indirect addressing
- ▶ $EA = (R)$
- ▶ Operand is in memory cell pointed to by contents of register R
- ▶ Large address space (2^n)
- ▶ One fewer memory access than indirect addressing



Displacement Addressing

- ▶ $EA = A + (R)$
- ▶ Address field hold two values
 - ▶ A = base value
 - ▶ R = register that holds displacement
 - ▶ or vice versa



Relative Addressing

- ▶ A version of displacement addressing
- ▶ $R = \text{Program counter, PC}$
- ▶ $EA = A + (PC)$
- ▶ i.e. get operand from A cells from current location pointed to by PC
- ▶ c.f locality of reference & cache usage

Base-Register Addressing

- ▶ A holds displacement
- ▶ R holds pointer to base address
- ▶ R may be explicit or implicit
- ▶ e.g. segment registers in 80x86

Indexed Addressing

- ▶ $A = \text{base}$
- ▶ $R = \text{displacement}$
- ▶ $EA = A + R$
- ▶ Good for accessing arrays
 - ▶ $EA = A + R$
 - ▶ $R++$

Stack Addressing

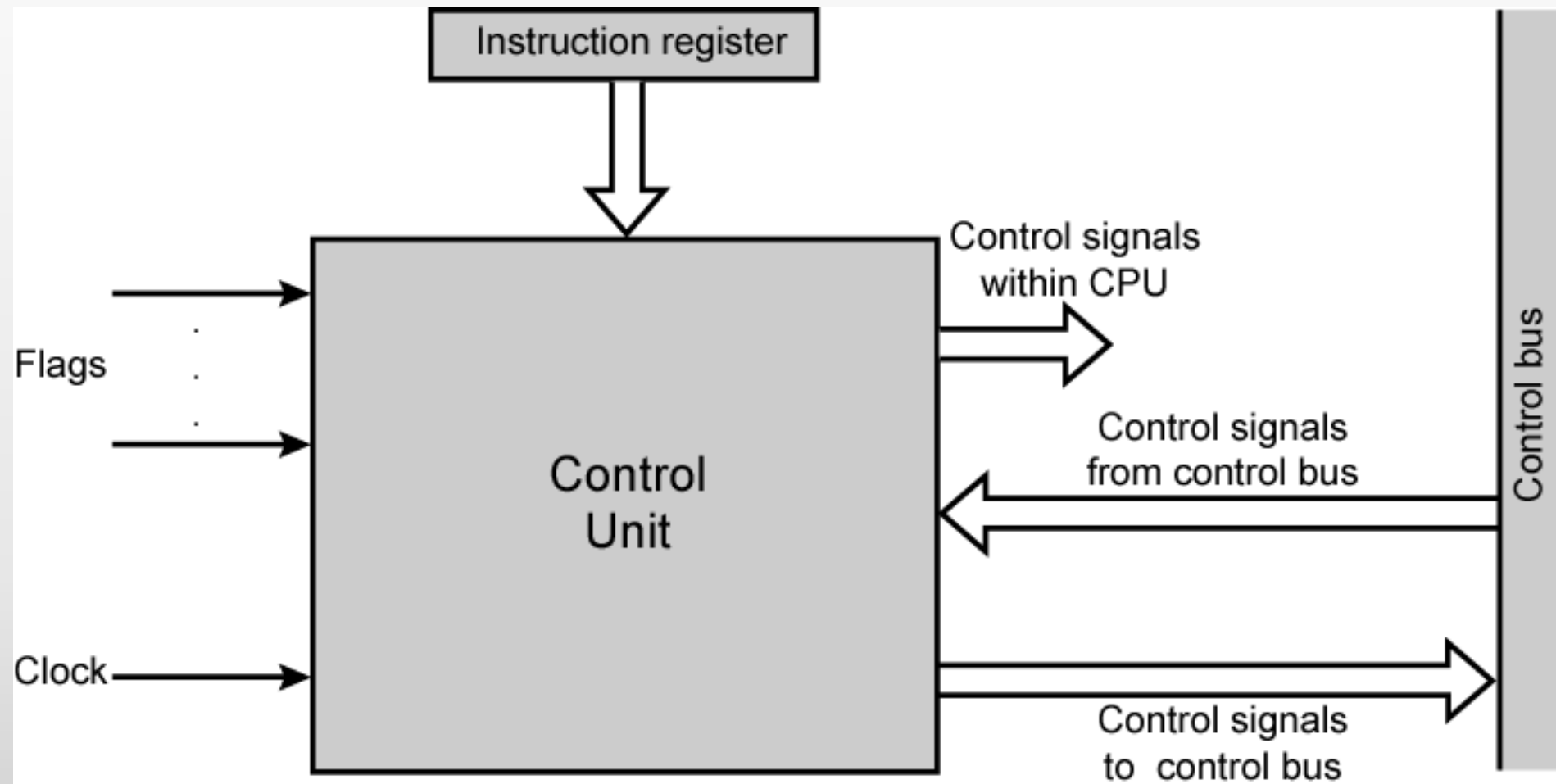
- ▶ A stack is a linear array of reserved memory locations
- ▶ Can be sometimes referred to as a *pushdown list* or *last-in-first-out queue*
- ▶ Items are appended to the top of the stack
- ▶ At any given time, the location block is partially filled
- ▶ Associated with the stack is a pointer - **stack pointer** - whose value is the address of the **top of the stack**

Control unit

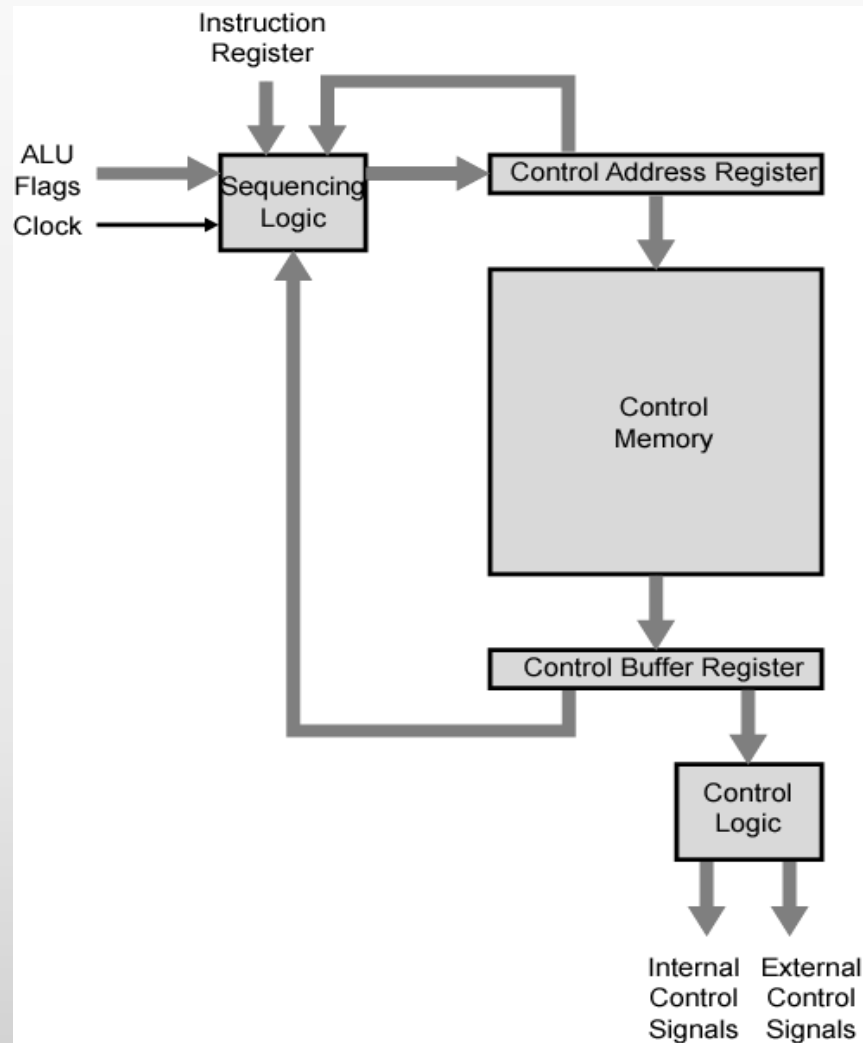
Control unit

- ▶ The **control unit** is that portion of the processor that actually causes things to happen
- ▶ The control unit issues:
 - ▶ Control signals external to the processor to cause data exchange with memory and I/O modules
 - ▶ Control signals internal to the processor to move data between registers, to cause the ALU to perform a specified function, and to regulate other internal operations
- ▶ Input to the control unit consists of the instruction register, flags, and control signals from external sources (e.g., interrupt signals).

Model of Control Unit



Control Unit Organization



Types of Micro-operation

- ▶ The execution of a program consists of *operations* involving different processor elements
- ▶ Operations consist of a sequence of **micro-operations**
- ▶ All micro-operations fall into one of the following categories:
 - ▶ Transfer data between registers
 - ▶ Transfer data from register to external
 - ▶ Transfer data from external to register
 - ▶ Perform an arithmetic or logical operation, using registers for input and output

Functions of Control Unit

- ▶ Sequencing
 - ▶ Causing the CPU to step through a series of micro-operations
- ▶ Execution
 - ▶ Causing the performance of each micro-op
- ▶ This is done using Control Signals

Control Signals

- ▶ Clock
 - ▶ One micro-instruction (or set of parallel micro-instructions) per clock cycle
- ▶ Instruction register
 - ▶ Op-code for current instruction
 - ▶ Determines which micro-instructions are performed
- ▶ Flags
 - ▶ State of CPU
 - ▶ Results of previous operations
- ▶ From control bus
 - ▶ Interrupts
 - ▶ Acknowledgements

Control Signals - output

- ▶ Within CPU
 - ▶ Cause data movement
 - ▶ Activate specific functions
- ▶ Via control bus
 - ▶ To memory
 - ▶ To I/O modules

Example Control Signal Sequence - Fetch

- ▶ $MAR \leftarrow (PC)$
 - ▶ Control unit activates signal to open gates between PC and MAR
- ▶ $MBR \leftarrow (\text{memory})$
 - ▶ Open gates between MAR and address bus
 - ▶ Memory read control signal
 - ▶ Open gates between data bus and MBR

Internal Organization

- ▶ Usually a single internal bus
- ▶ Gates control movement of data onto and off the bus
- ▶ Control signals control data transfer to and from external systems bus
- ▶ Temporary registers needed for proper operation of ALU

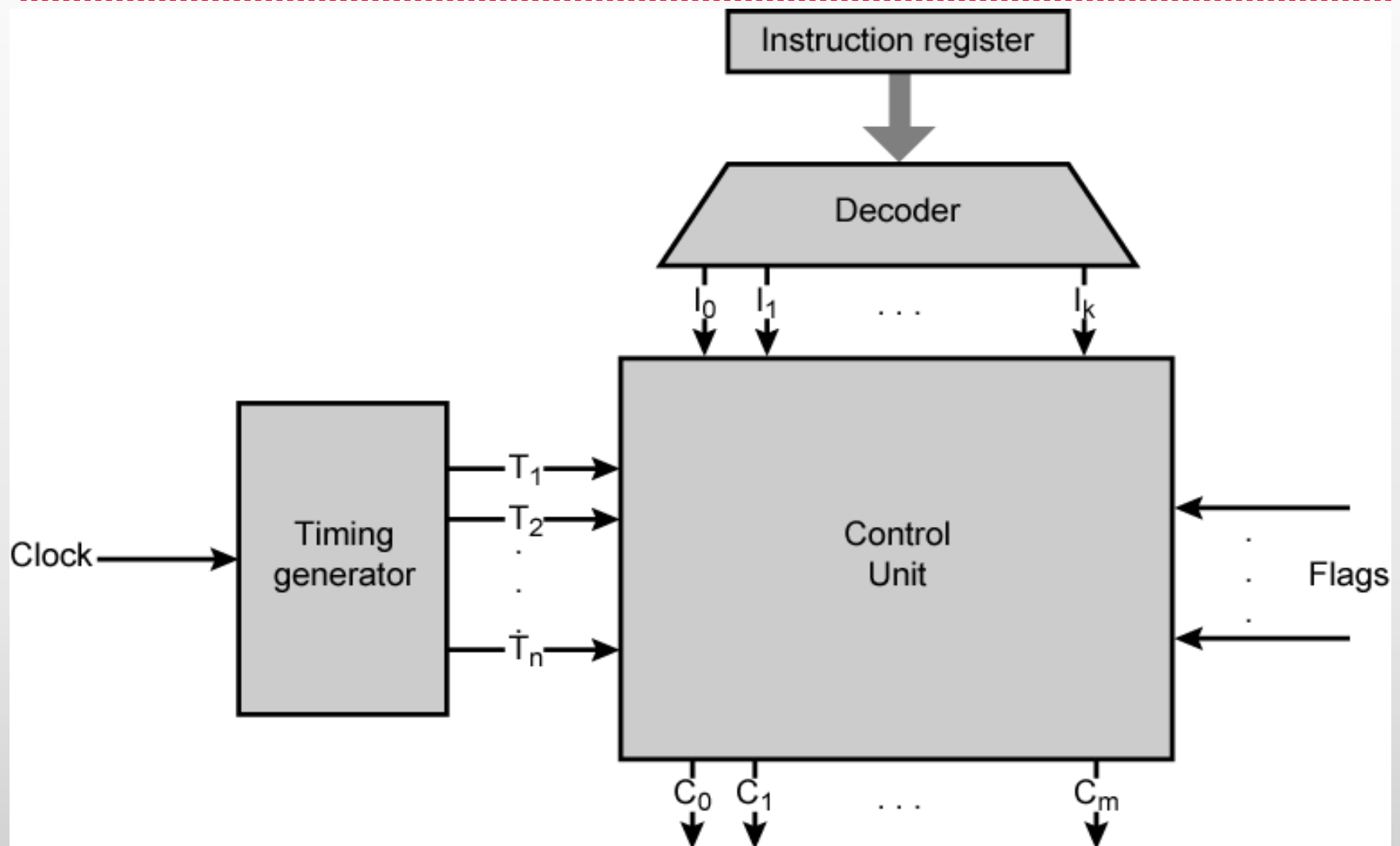
Implementations

- ▶ We have discussed the control unit in terms of its inputs, output, and functions
- ▶ A wide variety of techniques for the control unit implementation have been used
- ▶ Most of these fall into one of two categories:
 - ▶ Hardwired implementation
 - ▶ Microprogrammed implementation
- ▶ In a **hardwired** implementation, the control unit is essentially a **state machine circuit**:
 - ▶ Its input logic signals are transformed into a set of output logic signals, which are the control signals

Hardwired Implementation

- ▶ Control unit inputs
- ▶ Flags and control bus (each bit means something)
- ▶ Instruction register
 - ▶ Different control signals for each different instruction
 - ▶ Unique logic for each op-code
 - ▶ Decoder takes encoded input and produces single output
 - ▶ n binary inputs and 2^n outputs
- ▶ Clock
 - ▶ Repetitive sequence of pulses
 - ▶ Must be long enough to allow signal propagation
 - ▶ Different control signals at different times within instruction cycle

Control Unit with Decoded Inputs



Problems With Hardwired Designs

- ▶ Complex sequencing & micro-operation logic
- ▶ Difficult to design and test
- ▶ Inflexible design
- ▶ Difficult to add new instructions

Microprogrammed Control

- ▶ An alternative to a hardwired control unit is a **microprogrammed control unit**
- ▶ The logic of the control unit is specified by a **microprogram**, consisting of a sequence of instructions in a **microprogramming** language
- ▶ The (very simple) instructions specifies **micro-operations**
- ▶ A microprogrammed control unit is a relatively simple logic circuit that is capable of
 - ▶ **sequencing** through microinstructions
 - ▶ **generating control signals** to execute each microinstruction

Microprogrammed Control

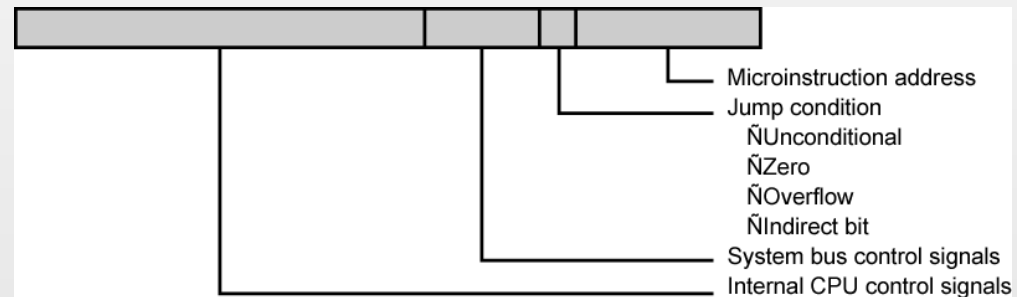
- ▶ Each **microinstruction** line describes a **set of micro-operations** occurring at one time
- ▶ For each **micro-operation**, all that the control unit is allowed to do is **generate a set of control signals**
- ▶ Thus, for any micro-operation, **each control line** emanating from the control unit **is either on or off**
- ▶ Each micro-operation is represented by a different pattern of 1s and 0s: the **control word**
- ▶ In a *control word* each bit represents one **control line**
- ▶ A **sequence of control words** represents the sequence of micro-operations performed by the control unit

Implementation

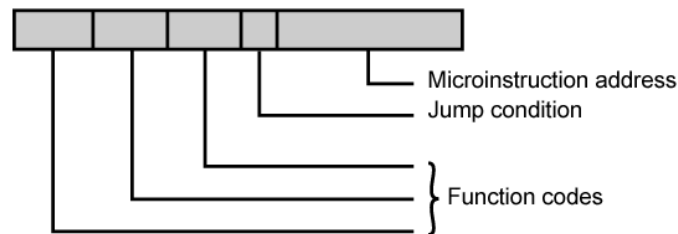
- ▶ **Control words** are put in a **memory** and each word has a **unique address**
- ▶ An address field is added to each control word indicating the location of the next control word to be executed if a certain condition is true
- ▶ There is a sequence of control words for each machine code instruction
- ▶ Today's large microprocessor
 - ▶ **Many instructions and associated register-level hardware**
 - ▶ **Many control points to be manipulated**

Micro-instruction Types

- ▶ **Vertical micro-programming:** each micro-instruction specifies (single or few) micro-operations to be performed
- ▶ **Horizontal micro-programming:** each micro-instruction specifies many different micro-operations to be performed in parallel



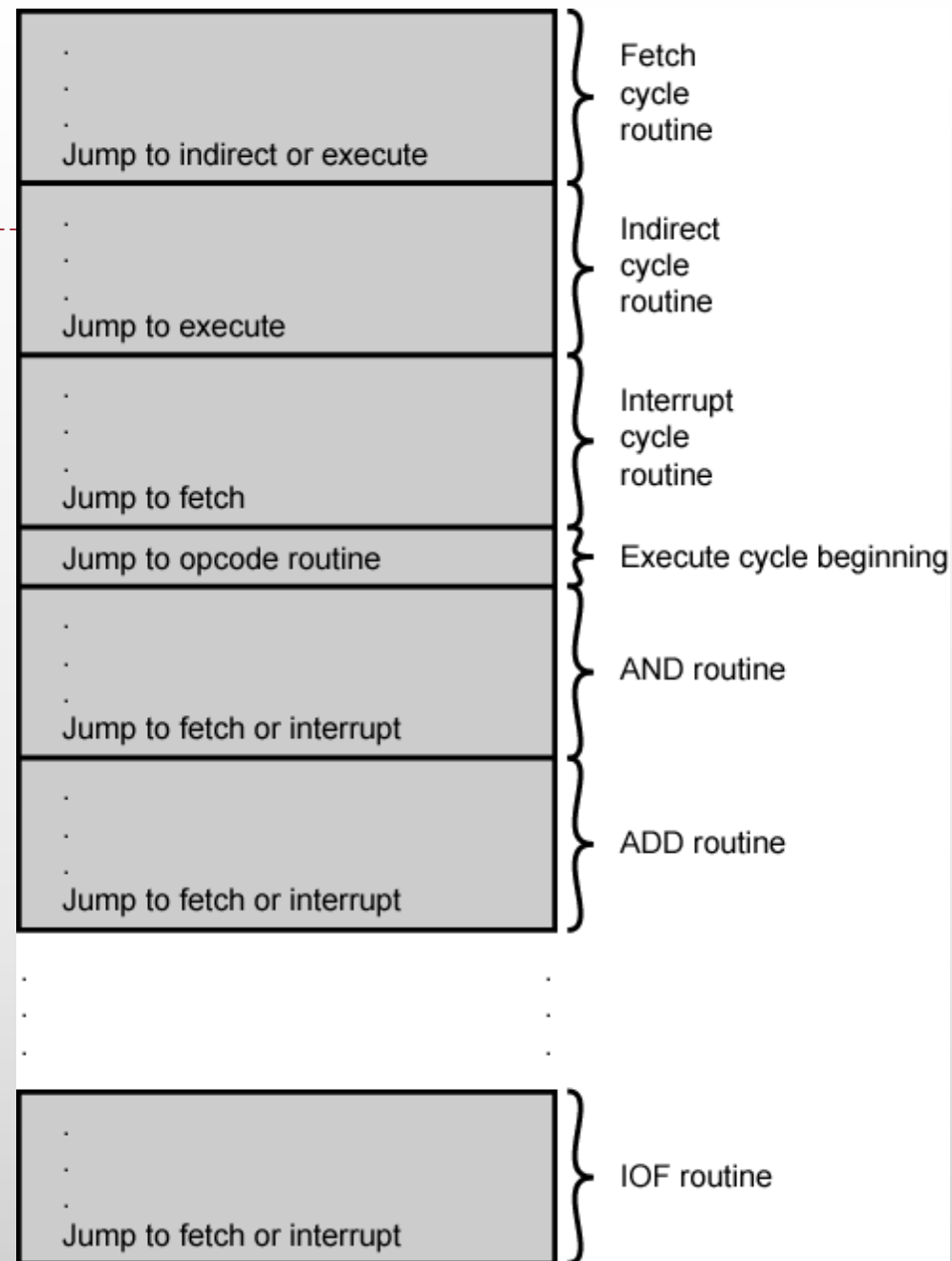
(a) Horizontal microinstruction



(b) Vertical microinstruction

Organization of Control Memory

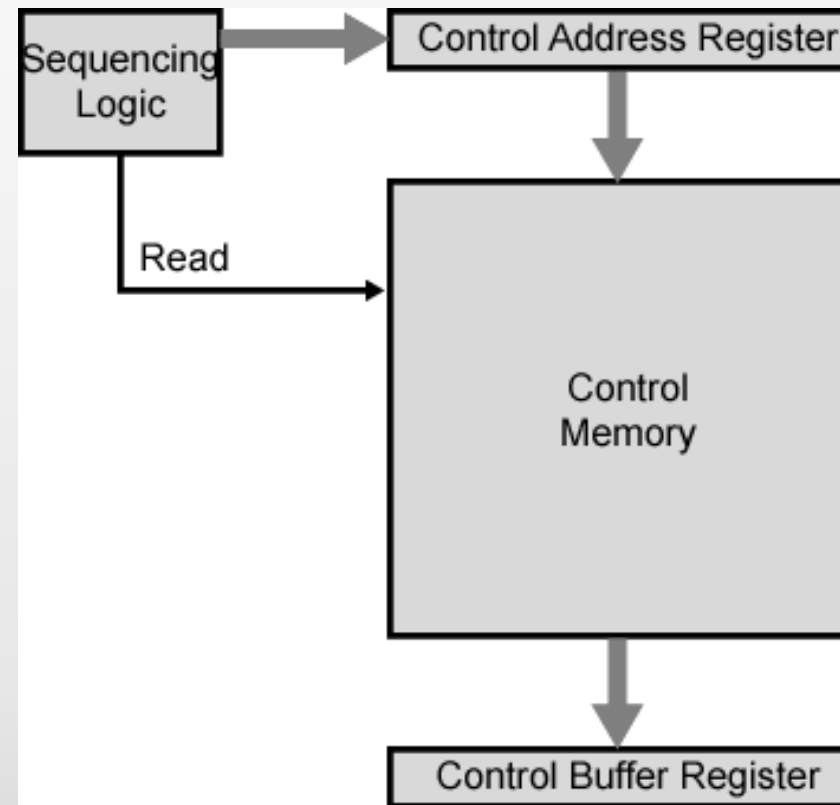
- ▶ Microinstructions (or control words) are arranged in a **control memory** (rom)
- ▶ Microinstructions in a routine are **executed sequentially**
- ▶ Each routine ends with a **branch** or **jump** instruction indicating where to go next
- ▶ A special execute cycle routine specifies the instruction routine (AND, ADD, and so on) to be executed next (according to the current opcode)



Control Unit

Key elements of **microprogrammed implementation**:

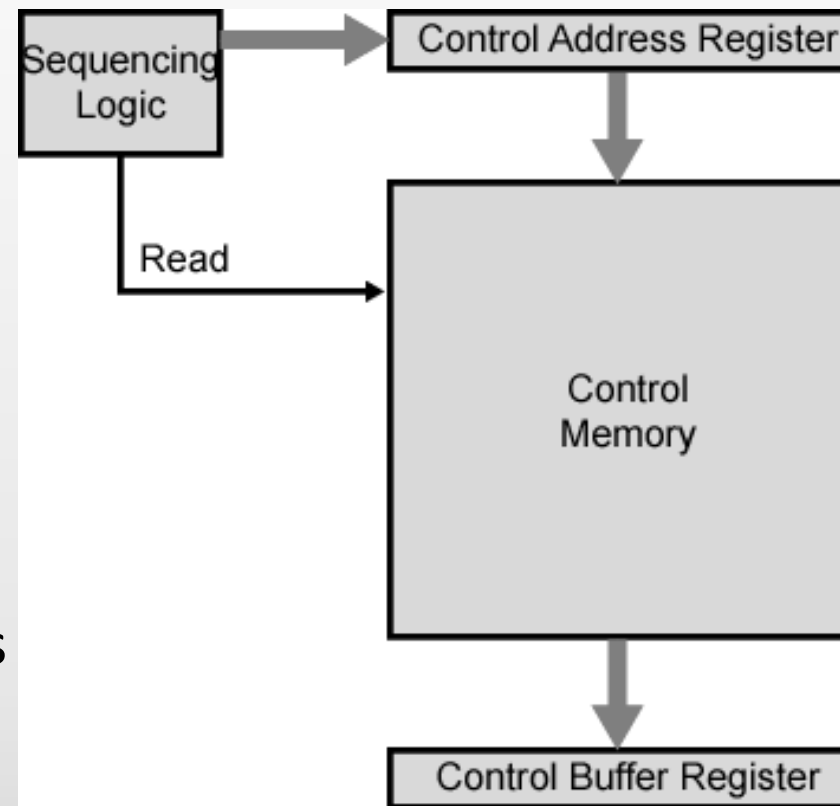
- ▶ The set of microinstructions is stored in the *control memory*
- ▶ The *control address register* contains the address of the next microinstruction to be read



Control Unit

Key elements of **microprogrammed implementation**:

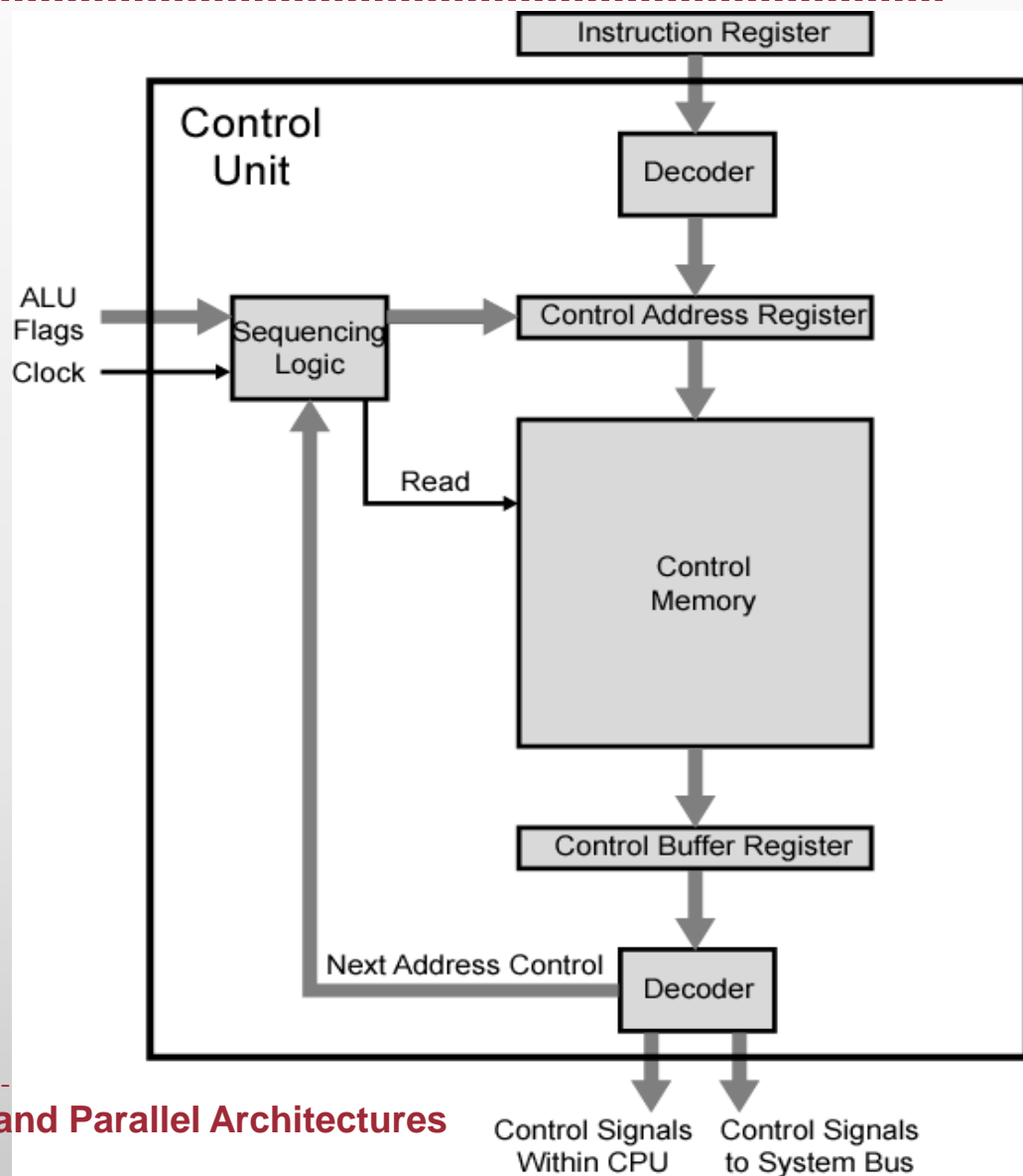
- ▶ When a microinstruction is **read** from the control memory, it is transferred to a *control buffer register*
- ▶ **Reading** a microinstruction from the control memory is the same as **executing** that microinstruction
- ▶ The *sequencing unit* loads the control address register and issues a read command



Functioning of Microprogrammed Control Unit

The control unit functions as follows:

1. To execute an instruction, the sequencing logic unit issues a READ command to the control memory
2. The word whose address is specified in the control address register is read into the control buffer register

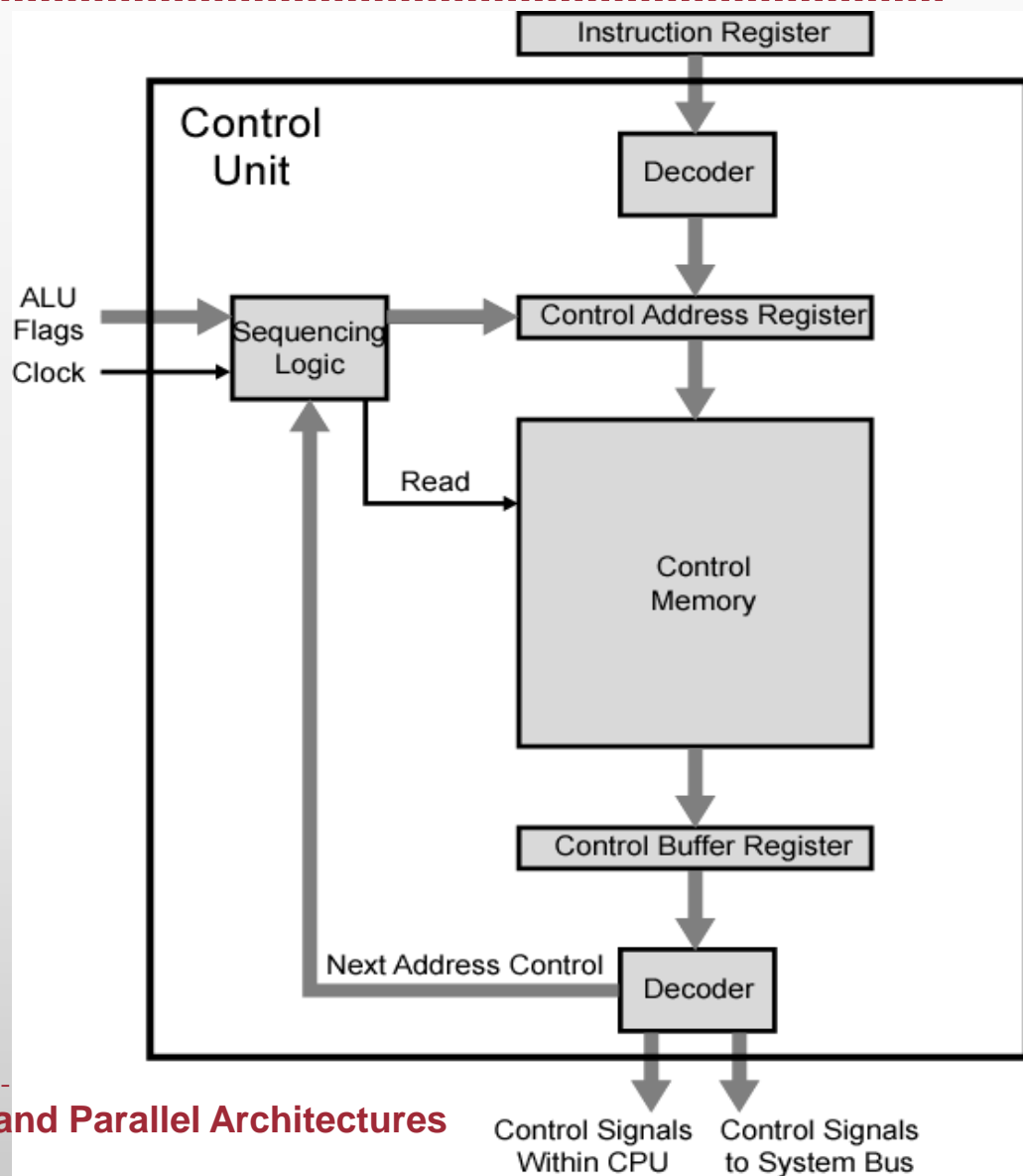


Functioning of Microprogrammed Control Unit

3. The content of the **control buffer register** generates control signals and next address information for the sequencing logic unit

4. The **sequencing logic unit** loads a new address into the control address register (based on the next-address information from the control buffer register and the ALU flags)

All this happens during one clock pulse



Functioning of Microprogrammed Control Unit

Figure shows **two decoder**:

- ▶ The upper decoder translates the opcode of the IR into a control memory address
- ▶ The lower decoder is used for **vertical microinstructions**

