Advanced Parallel Architecture

Lesson 3

Annalisa Massini - 2014/2015

Von Neumann Architecture

Two lessons

- Summary of the traditional computer architecture

→ **Von Neumann architecture**

- http://WilliamStallings.com/COA/COA7e.html

What is a computer?

- A computer was a job title, not a piece of equipment!
- Requirements of a computer:
  - Process data
  - Store data
  - Move data between the computer and the outside world
  - Control the operation of the above

Architecture & Organization

- Architecture is those 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 is how features are implemented
  - Control signals, interfaces, memory technology
  - e.g. Is there a hardware multiply unit or is it done by repeated addition?

Structure & Function

- Structure is the way in which components relate to each other
- Function is the operation of individual components as part of the structure

- All computer functions are:
  - Data processing
  - Data storage
  - Data movement
  - Control

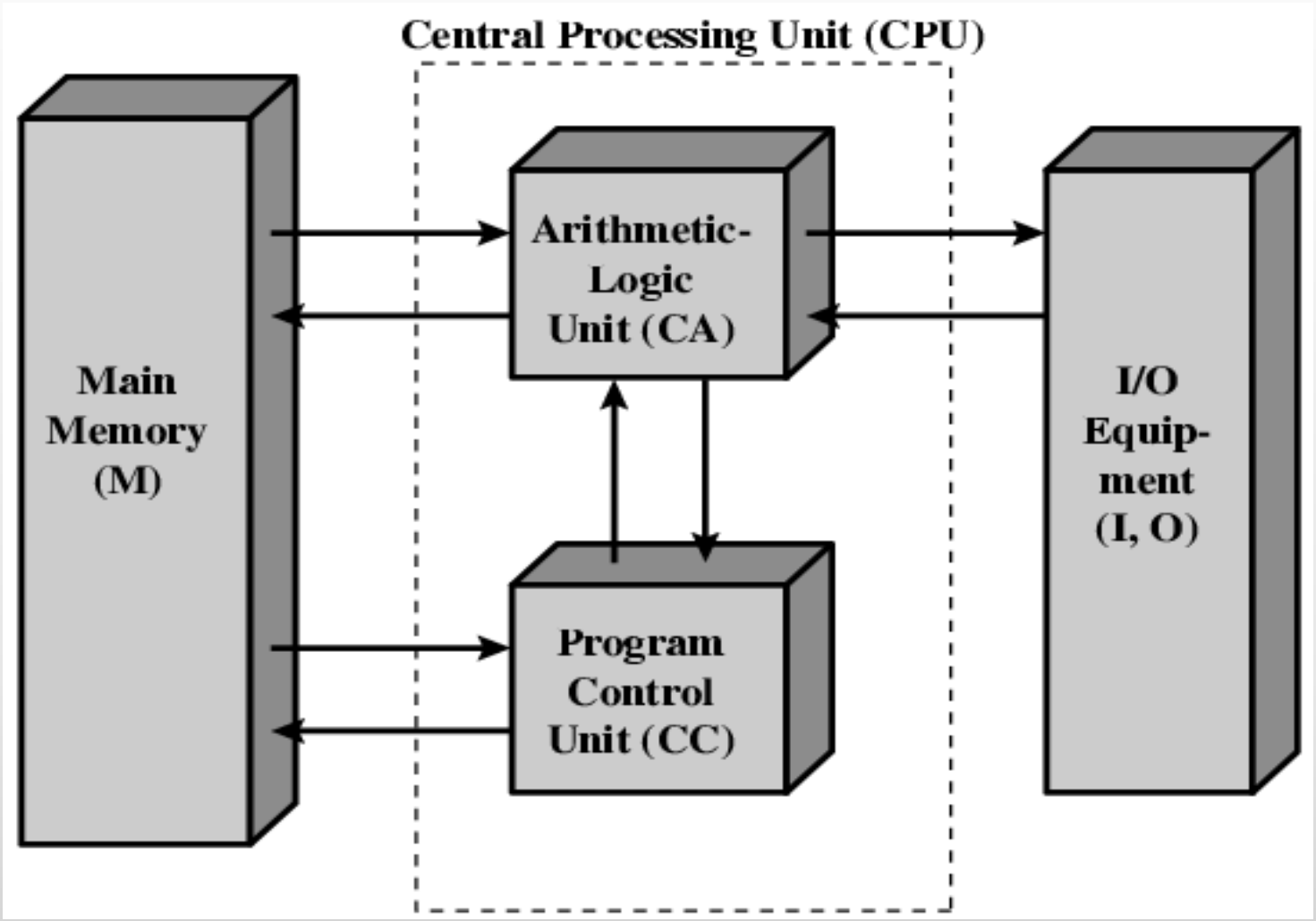
ENIAC - background

- Electronic Numerical Integrator And Computer
- Eckert and Mauchly - University of Pennsylvania
- Trajectory tables for weapons
- Started 1943 - Finished 1946 - Used until 1955
- Decimal (not binary) - 20 accumulators of 10 digits
- Programmed manually by switches
- 18,000 vacuum tubes
- 30 tons - 15,000 square feet
- 140 kW power consumption
- 5,000 additions per second

von Neumann/Turing

- Stored Program concept
- Main memory storing programs and data
- ALU operating on binary data
- Control unit interpreting instructions from memory and executing
- Input and output equipment operated by control unit
- Princeton Institute for Advanced Studies - IAS
- Completed 1952

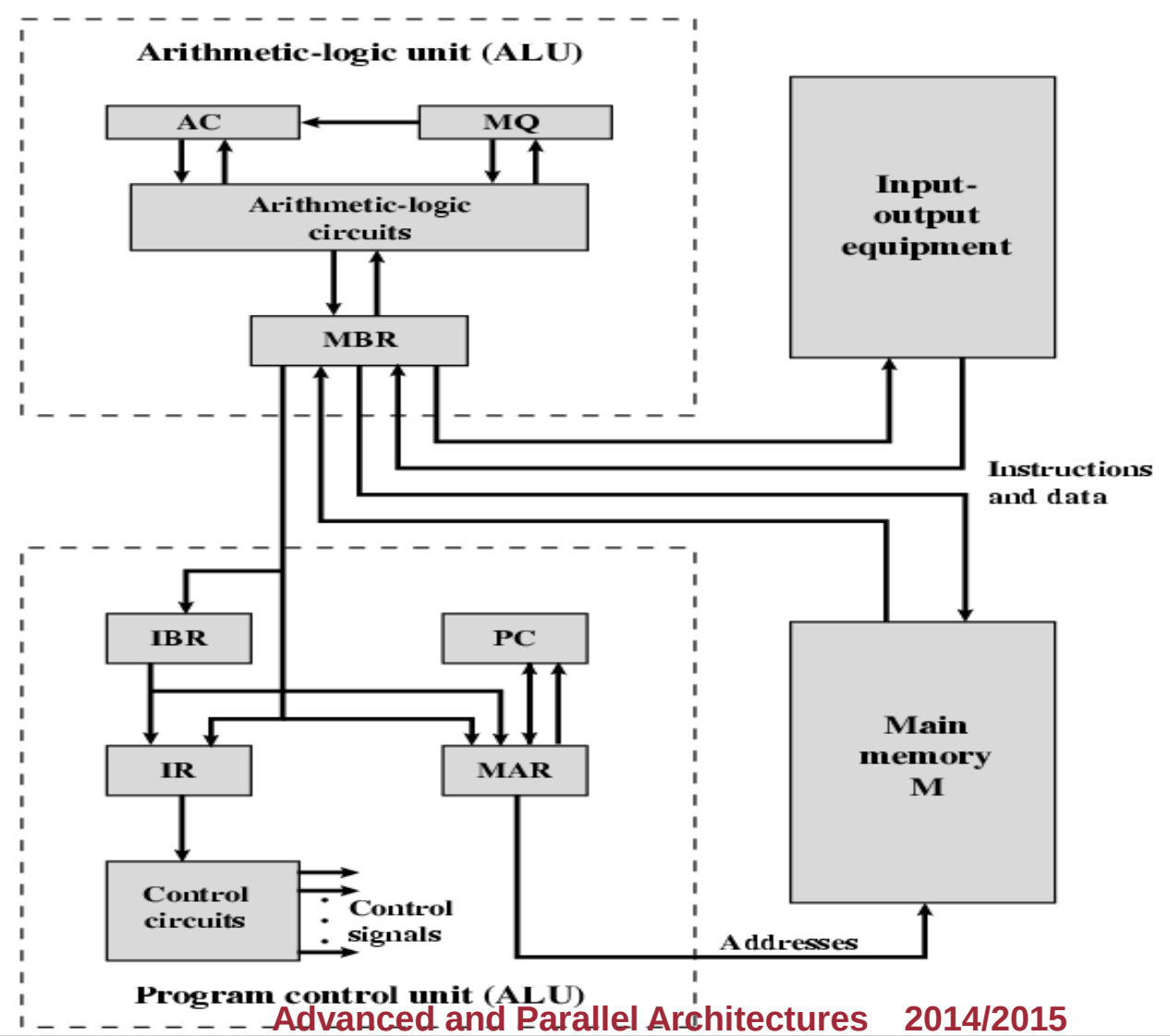
Structure of von Neumann machine



IAS

- 1000 x 40 bit words
  - Binary number
  - 2 x 20 bit instructions
- Set of registers (storage in CPU)
  - Memory Buffer Register
  - Memory Address Register
  - Instruction Register
  - Instruction Buffer Register
  - Program Counter
  - Accumulator
  - Multiplier Quotient

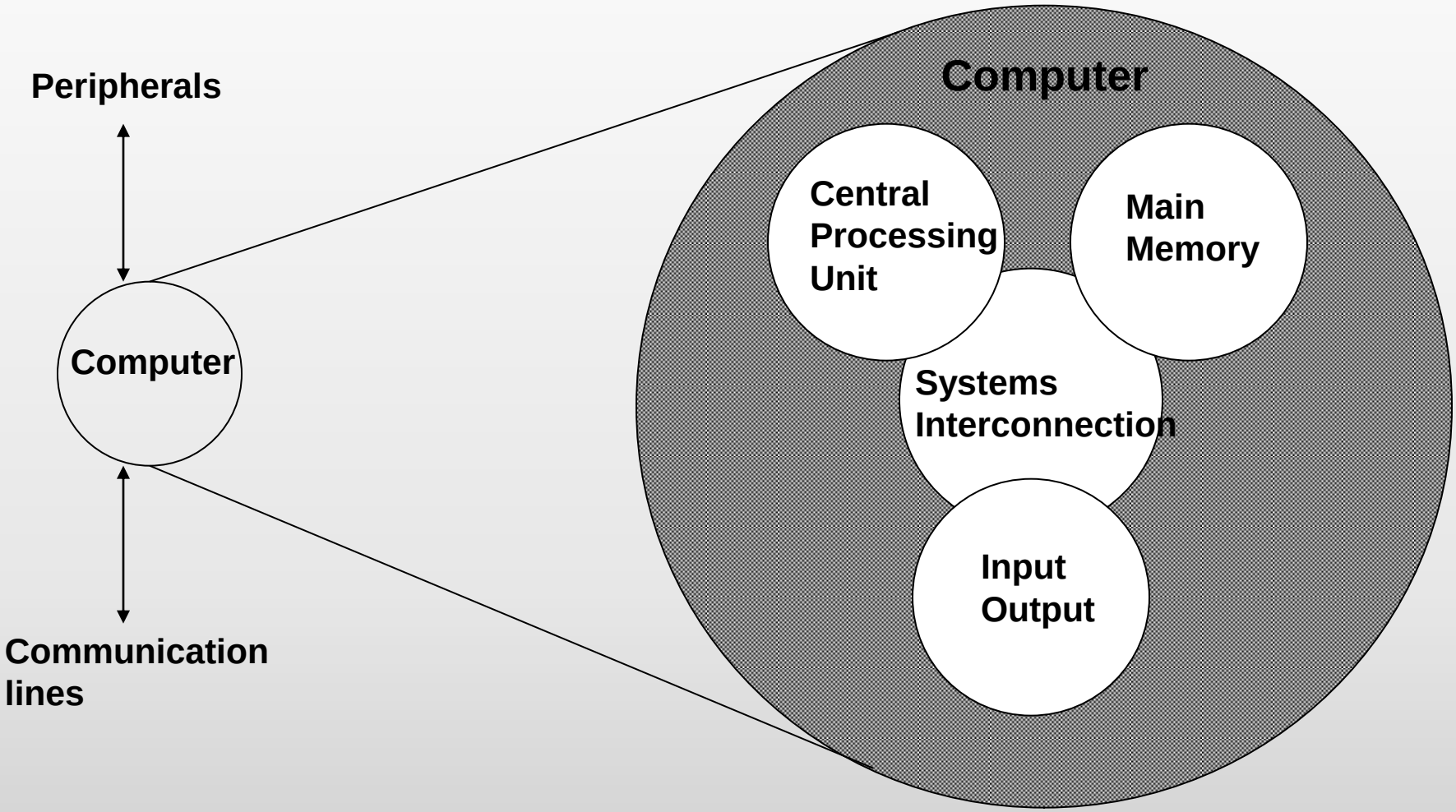
Structure of IAS



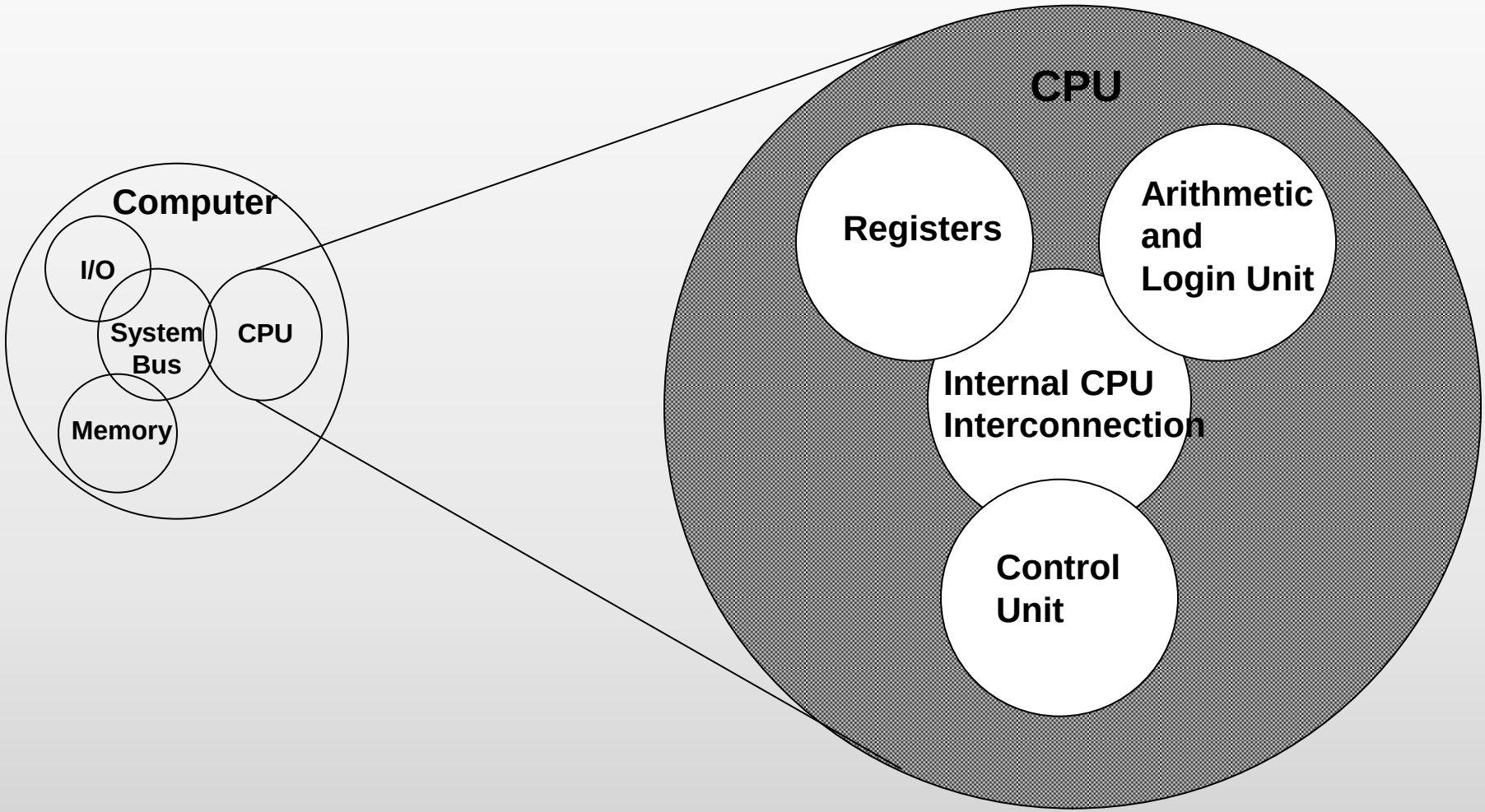
Generations of Computer

- Vacuum tube - 1946-1957
- Transistor - 1958-1964
- Small scale integration - 1965 on
  - Up to 100 devices on a chip
- Medium scale integration - to 1971
  - 100-3,000 devices on a chip
- Large scale integration - 1971-1977
  - 3,000 - 100,000 devices on a chip
- Very large scale integration - 1978 -1991
  - 100,000 - 100,000,000 devices on a chip
- Ultra large scale integration – 1991 -
  - Over 100,000,000 devices on a chip

Structure - Top Level



Structure - The CPU



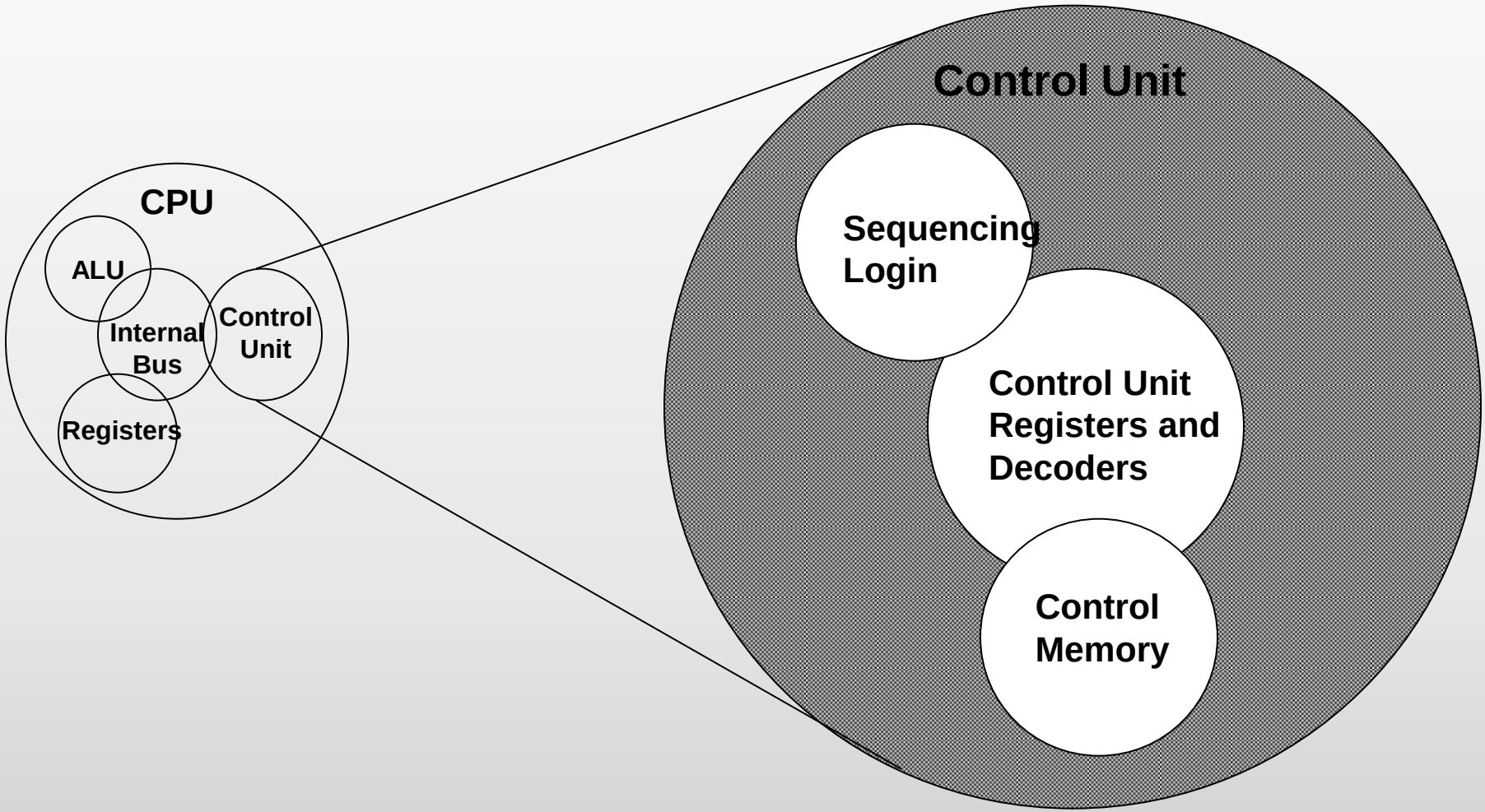
Components

- The Control Unit and the Arithmetic and Logic Unit constitute the Central Processing Unit
- Data and instructions get into the system and results out
  - Input/output
- Temporary storage of code and results is needed
  - Main memory

Basic element of a processor

  - ALU
  - Registers
  - Internal data pahs
  - External data paths
  - Control Unit

Structure - The Control Unit



Functional Requirements

- Define basic elements of processor
- Describe micro-operations processor performs
- Determine functions control unit must perform

Four Levels of Computer Description

- Global system structure
  - Overall system structure is defined
  - Major components identified
    - Processors
    - Control modules
    - Memory modules
    - Interconnection structure
  - Mostly a static description – “black box” approach

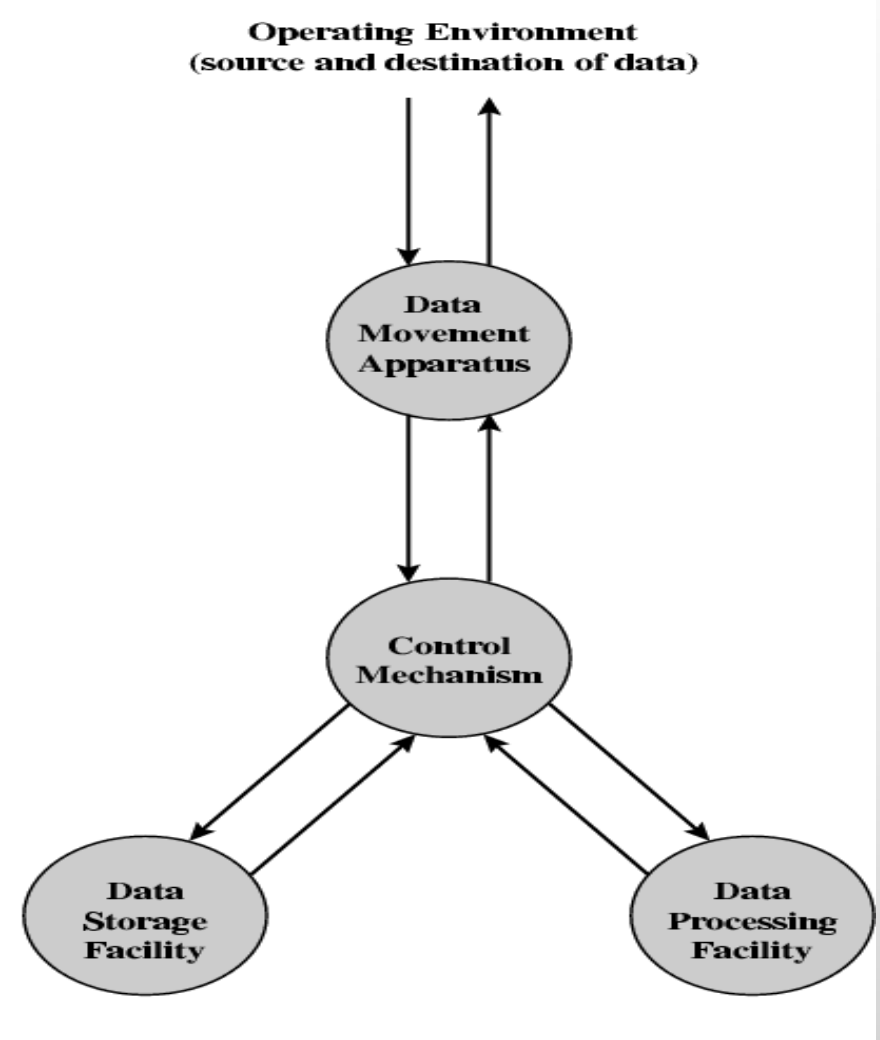
Four Levels of Computer Description

- Processor level
  - Architectural Features specified
    - Interfaces
    - Instruction sets
    - Data Representation
  - More detailed individual component specification

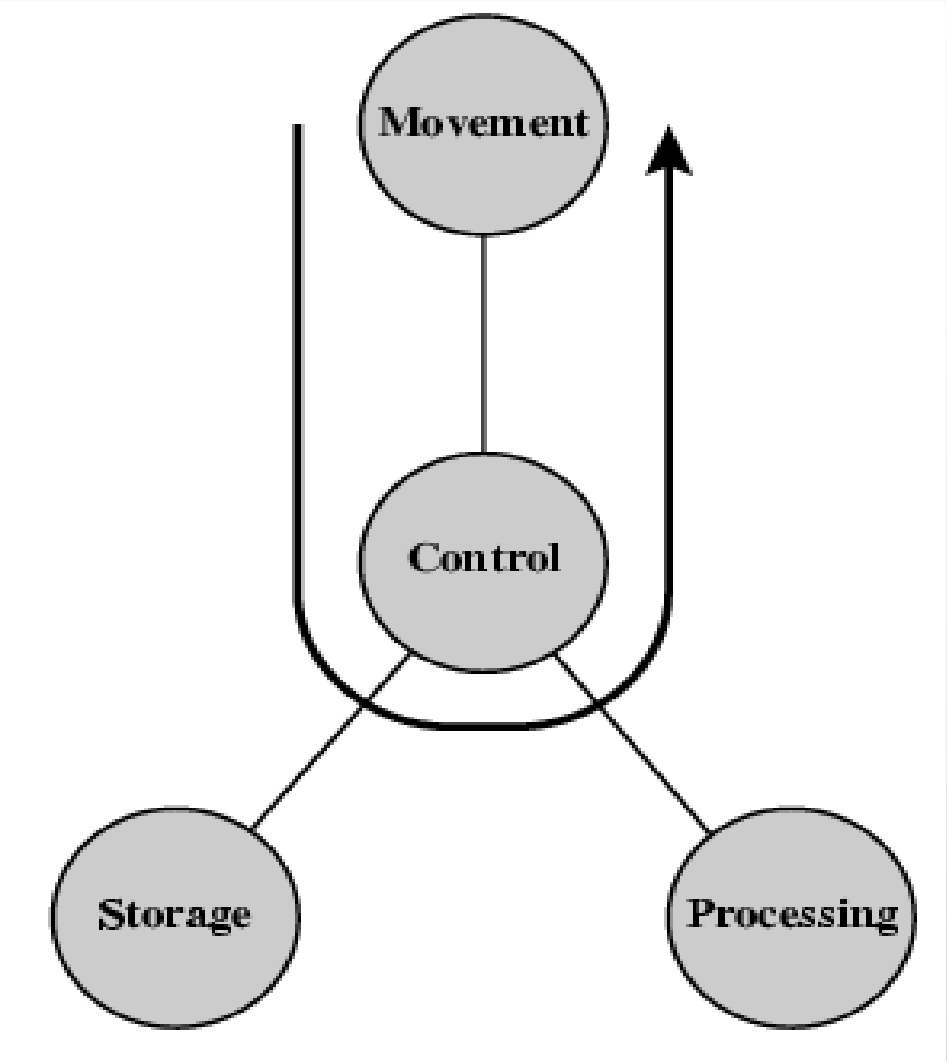
Four Levels of Computer Description

- Register level
  - Specify internal operation of processor-level components at the word level
    - Primitives
    - Registers
    - Counters
    - Memories
    - ALUs
    - Clocks
- Combinational logic Gate level
  - Specify operations at the individual bit level
  - Gates are primitive elements

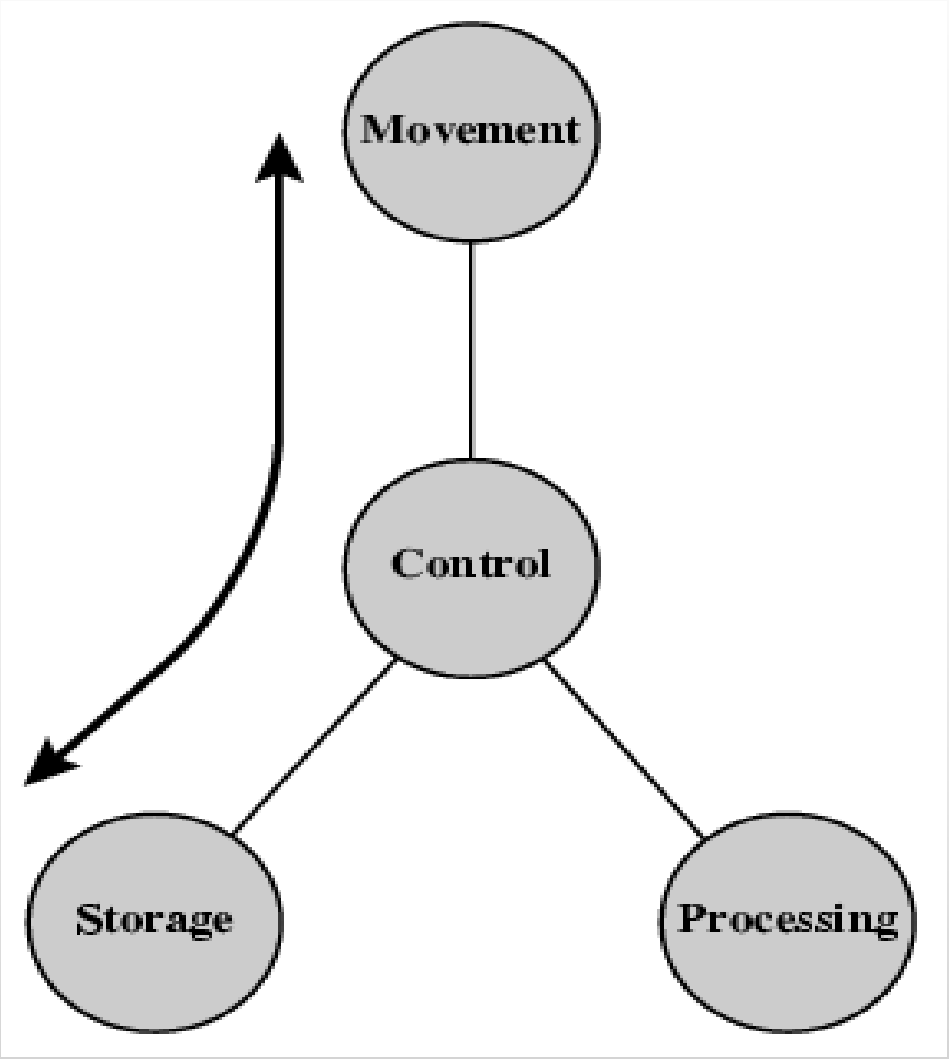
Functional View



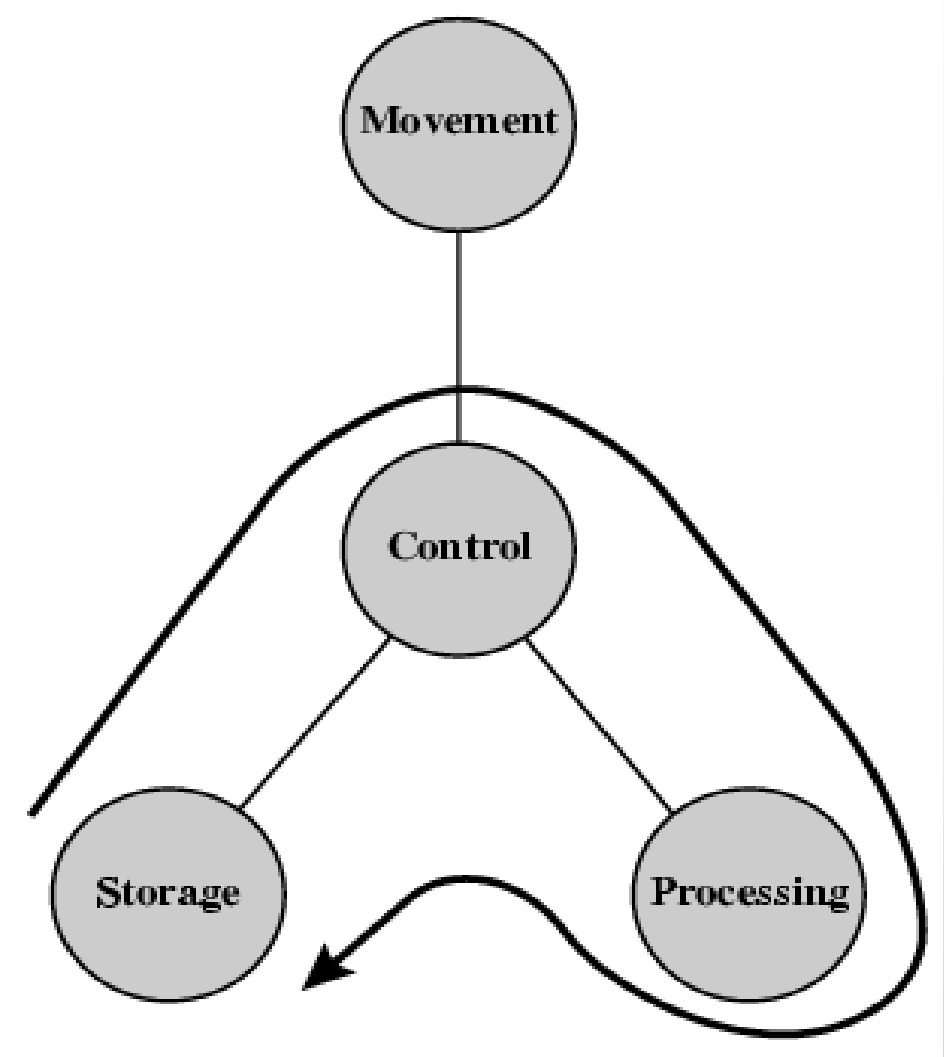
Operations - Data movement



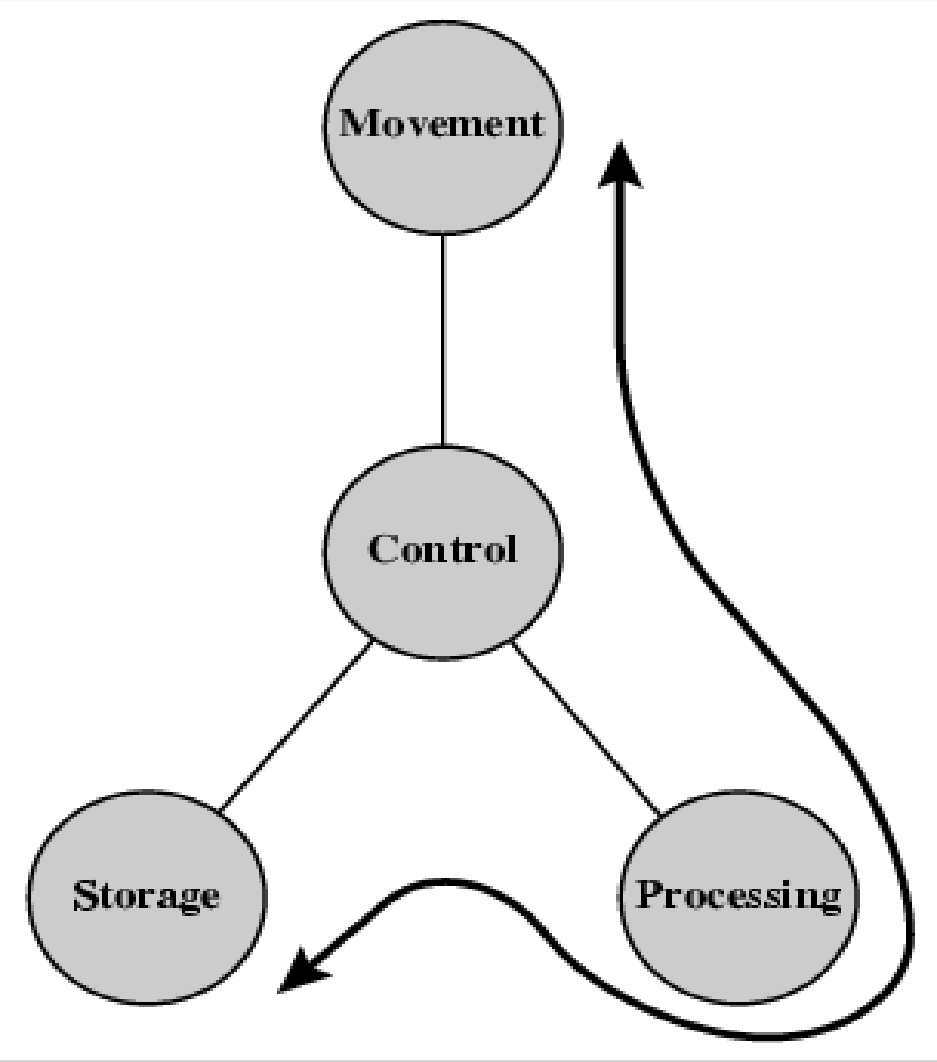
Operations - Storage



Operation - Processing from/to storage



Operation - Processing from storage to I/O



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

Observations

- At the micro-operation level, multiple control signals are generated at the same time
- Instruction pipelining, at least to the extent of overlapping fetch and execute operations, has been around for a long time

- Both of these are examples of performing functions in parallel

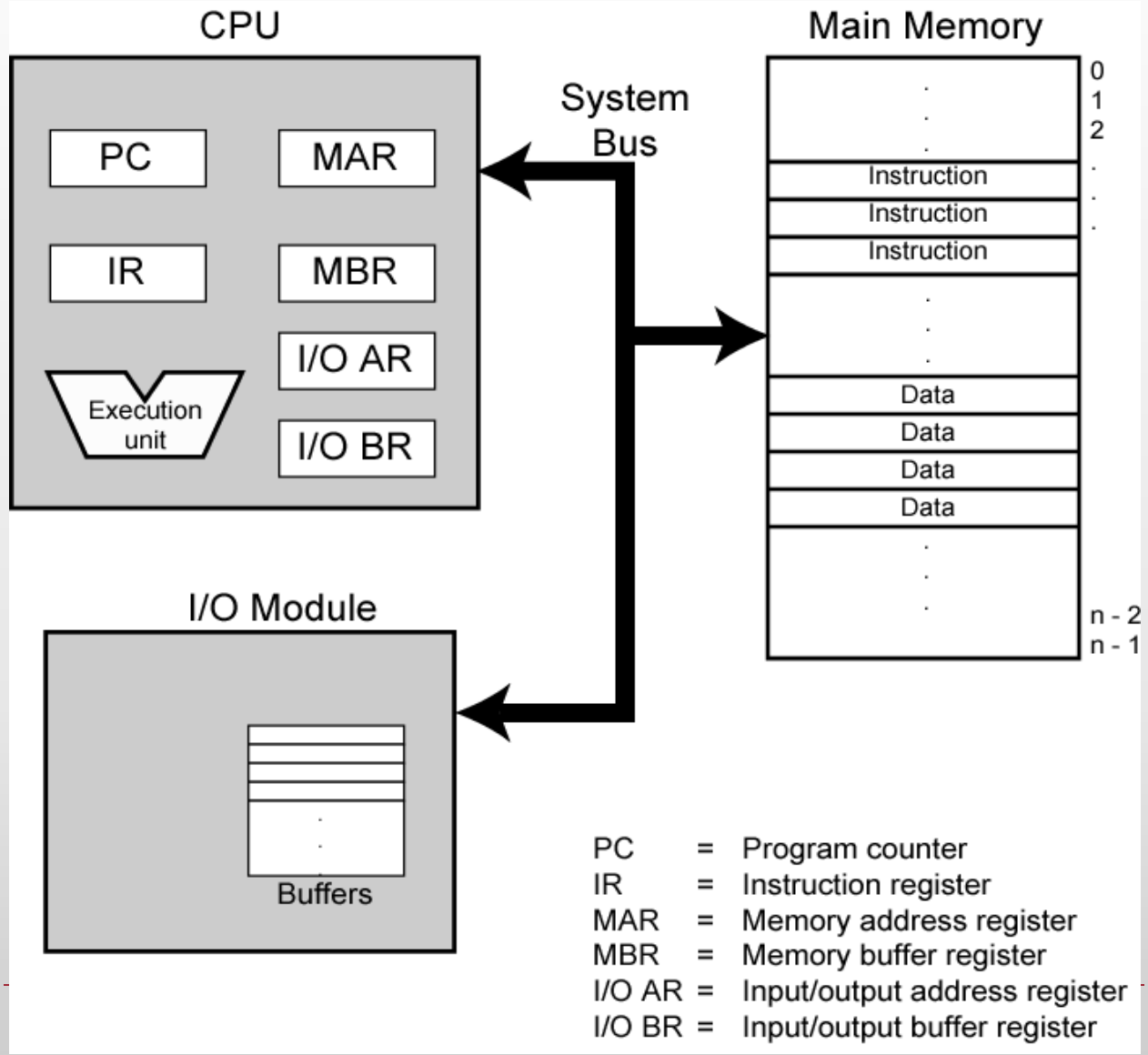
What is a program?

- A sequence of steps
- For each step, an arithmetic or logical operation is done
- For each operation, a different set of control signals is needed
- For each operation a unique code is provided
  - e.g. ADD, MOVE
- A hardware segment accepts the code and issues the control signals

Micro-Operations

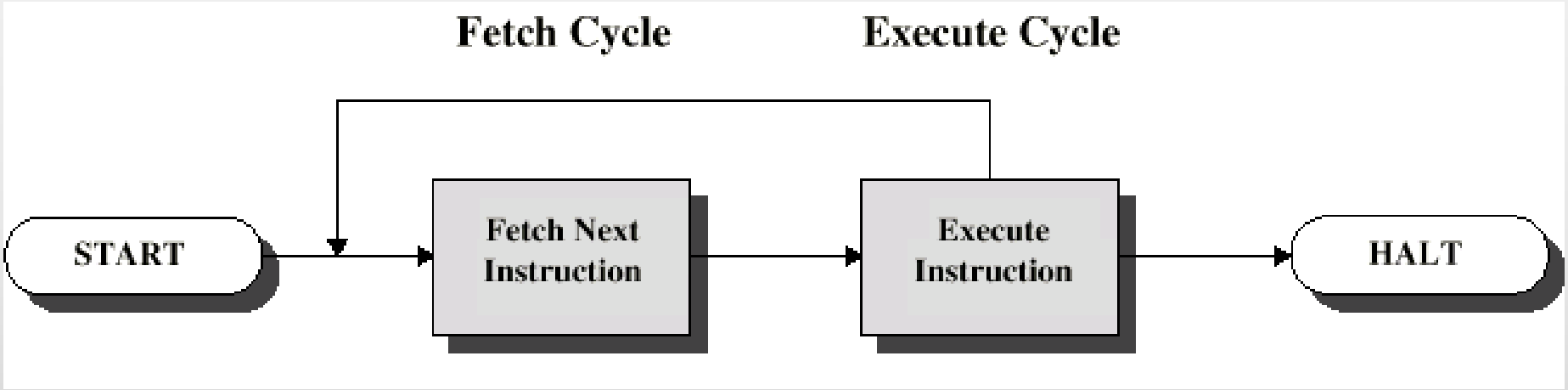
- A computer executes a program
- Fetch/execute cycle
- Each cycle has a number of steps
  - see pipelining
- Called micro-operations
- Each step does very little
- Atomic operation of CPU

Computer Components: Top Level View



Instruction Cycle

- Two steps:
  - Fetch
  - Execute



Fetch Cycle

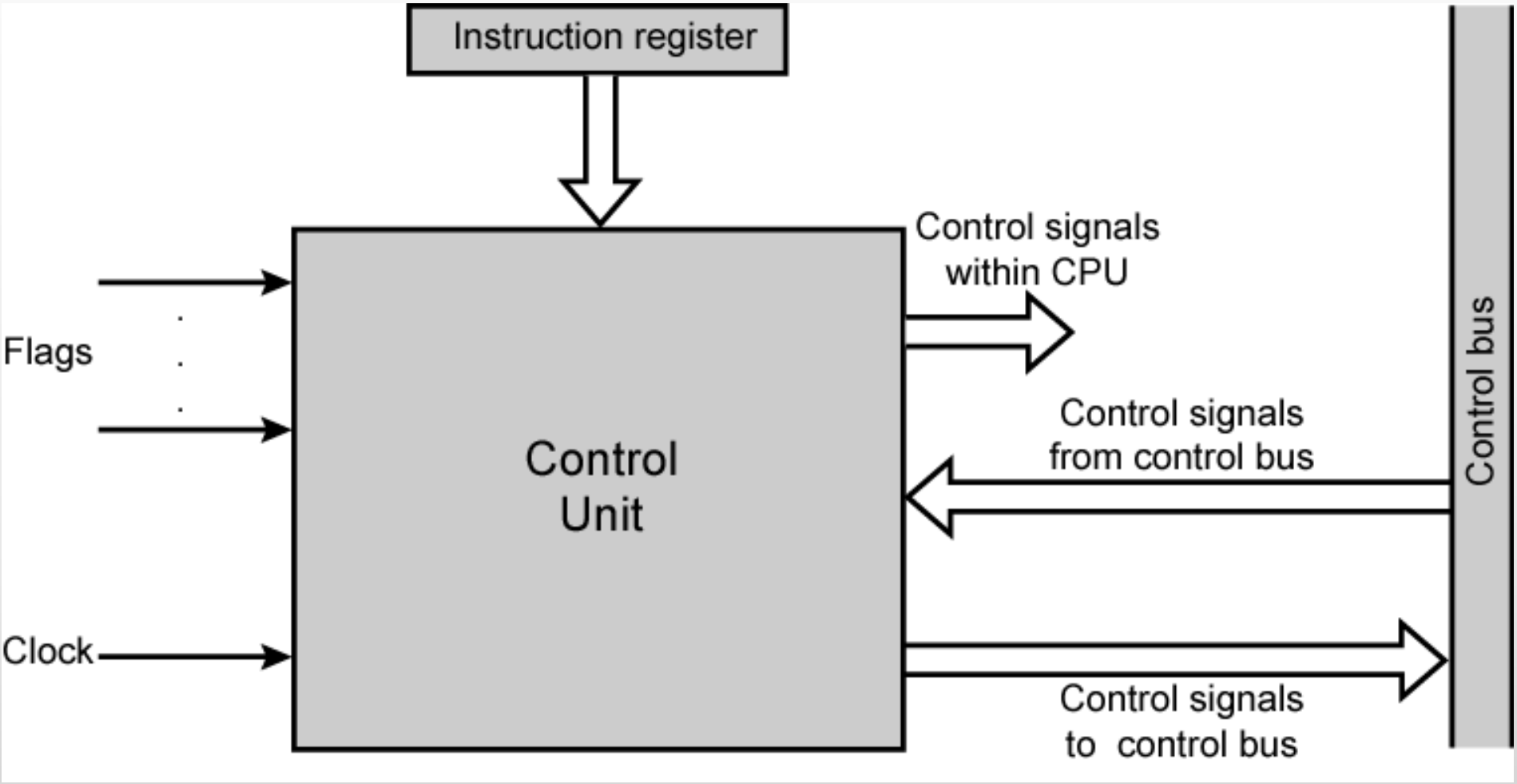
- Program Counter (PC) holds address of next instruction to fetch
- Processor fetches instruction from memory location pointed to by PC
- Increment PC
  - Unless told otherwise
- Instruction loaded into Instruction Register (IR)
- Processor interprets instruction and performs required actions

Execute Cycle

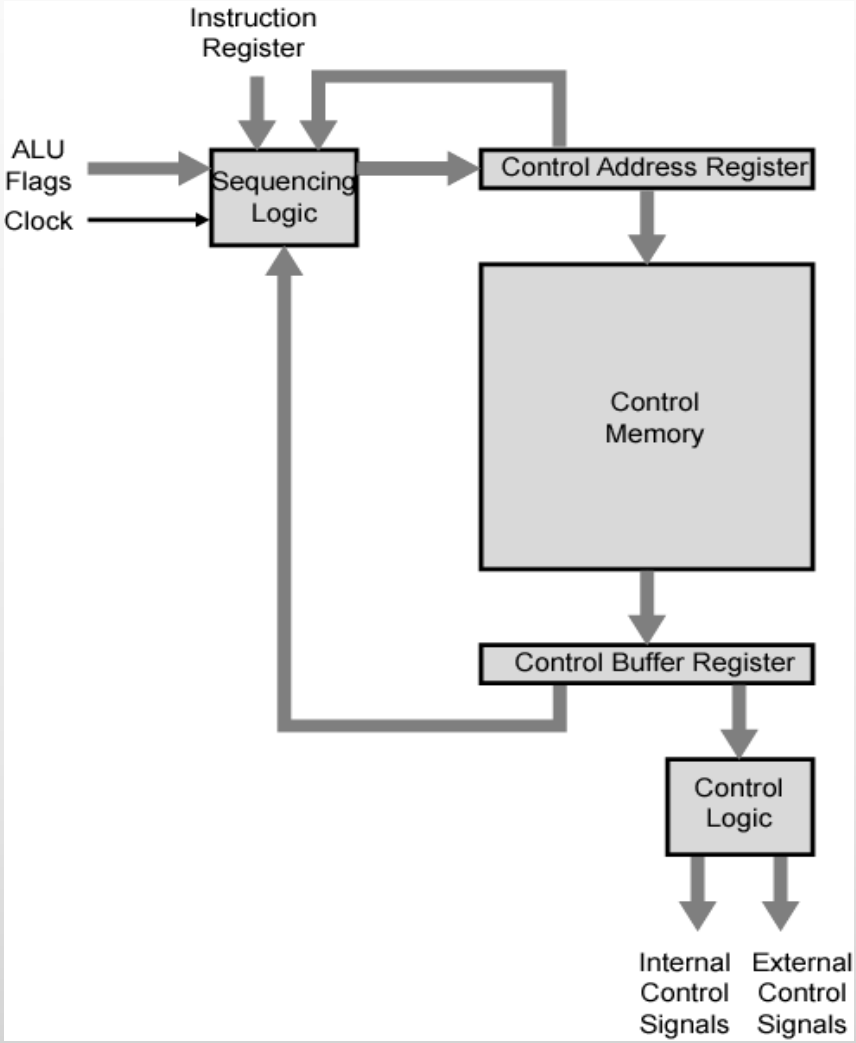
- Processor-memory
  - data transfer between CPU and main memory
- Processor I/O
  - Data transfer between CPU and I/O module
- Data processing
  - Some arithmetic or logical operation on data
- Control
  - Alteration of sequence of operations
  - e.g. jump
- Combination of above

Control unit

Model of Control Unit



Control Unit Organization



Types of Micro-operation

- Transfer data between registers
- Transfer data from register to external
- Transfer data from external to register
- Perform arithmetic or logical ops

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 <- (PC)
  - Control unit activates signal to open gates between PC and MAR
- MBR <- (memory)
  - Open gates between MAR and address bus
  - Memory read control signal
  - Open gates between data bus and MBR

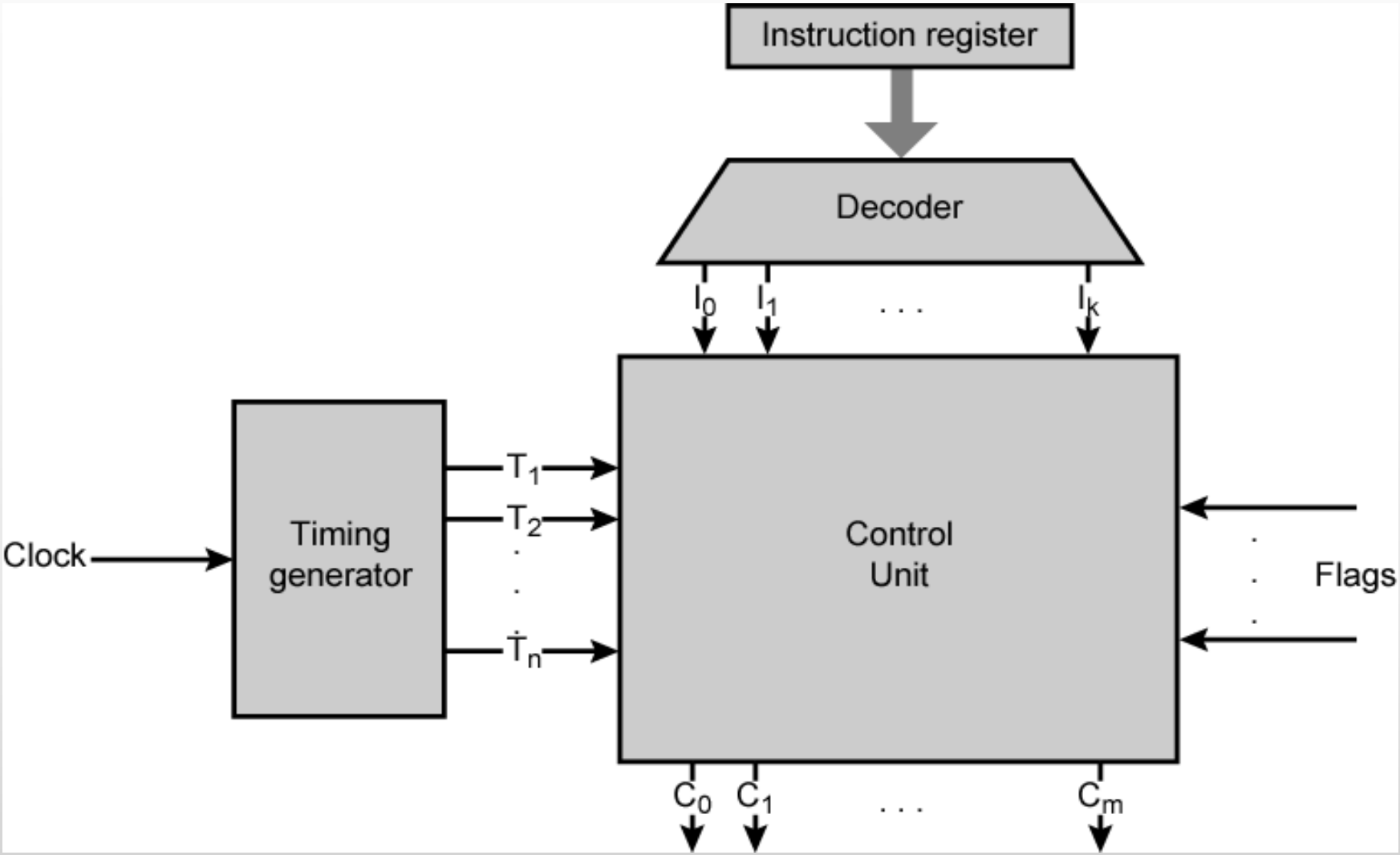
Internal Organization

- 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

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 Hard Wired Designs

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

Micro-programmed Control

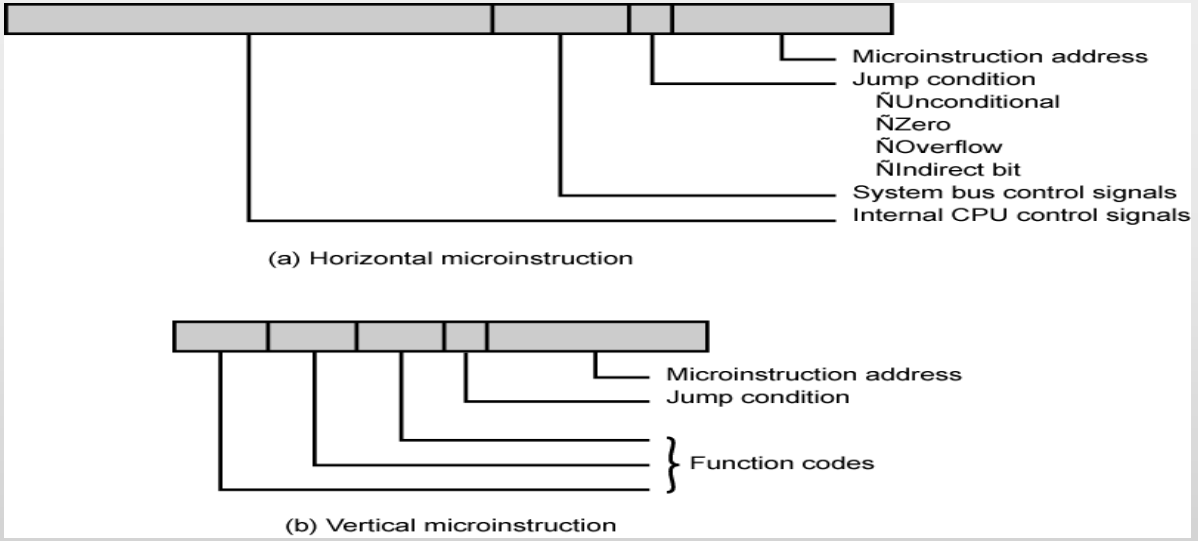
- Use sequences of instructions to control complex operations
- Called micro-programming or firmware

Implementation

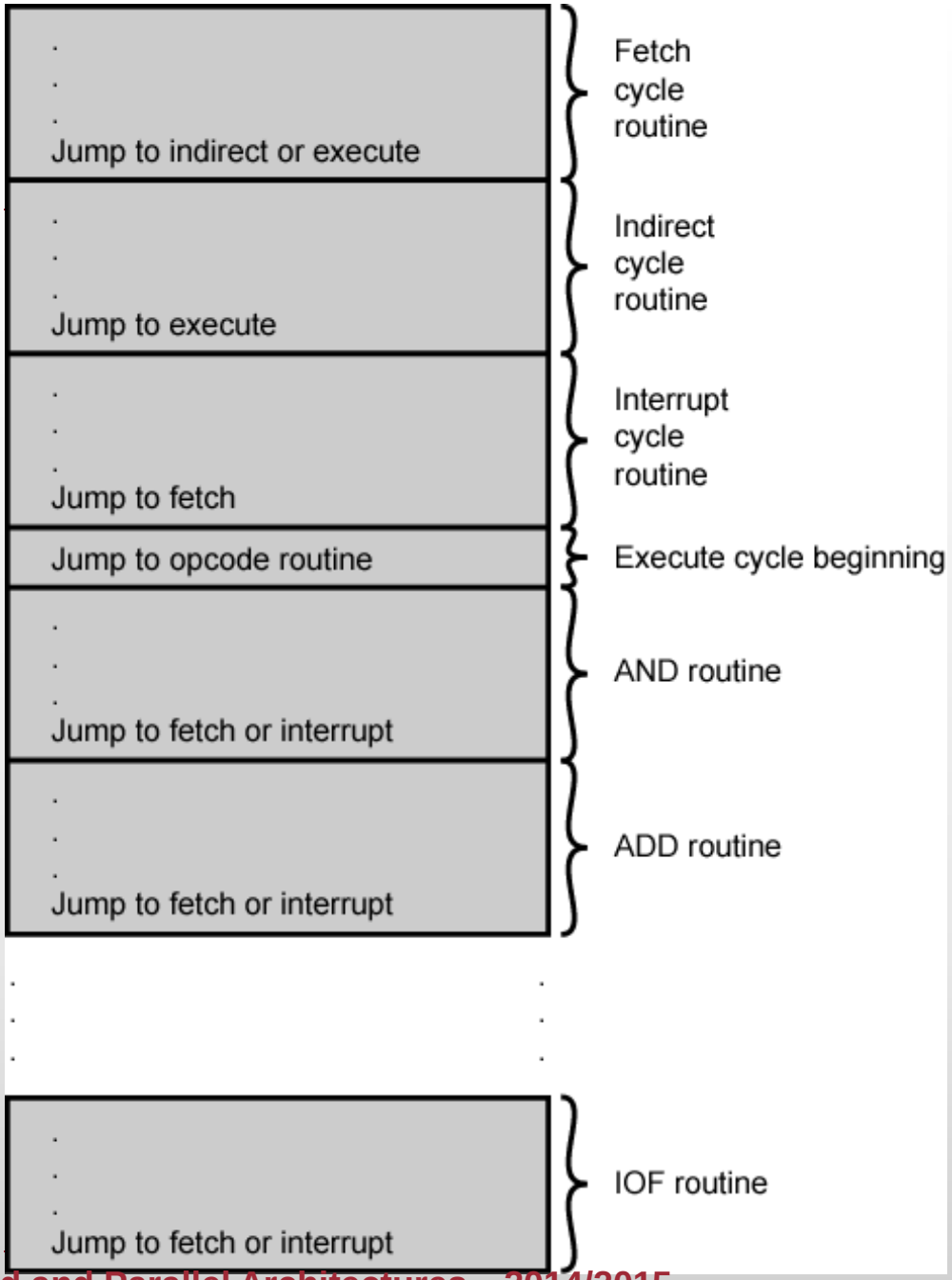
- All the control unit generates a set of control signals
- Each control signal is on or off
- Represent each control signal by a bit
- Have a control word for each micro-operation
- Have a sequence of control words for each machine code instruction
- Add an address to specify the next micro-instruction, depending on conditions
- Today's large microprocessor
  - Many instructions and associated register-level hardware
  - Many control points to be manipulated

Micro-instruction Types

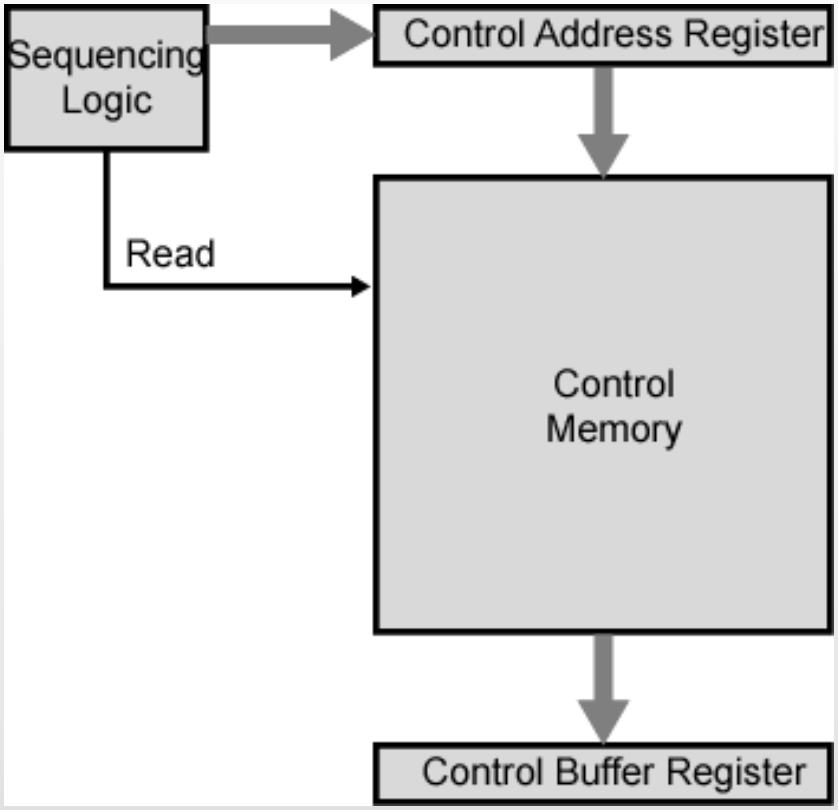
- Each micro-instruction specifies single (or few) micro-operations to be performed
  - (*vertical* micro-programming)
- Each micro-instruction specifies many different micro-operations to be performed in parallel
  - (*horizontal* micro-programming)



Organization of Control Memory



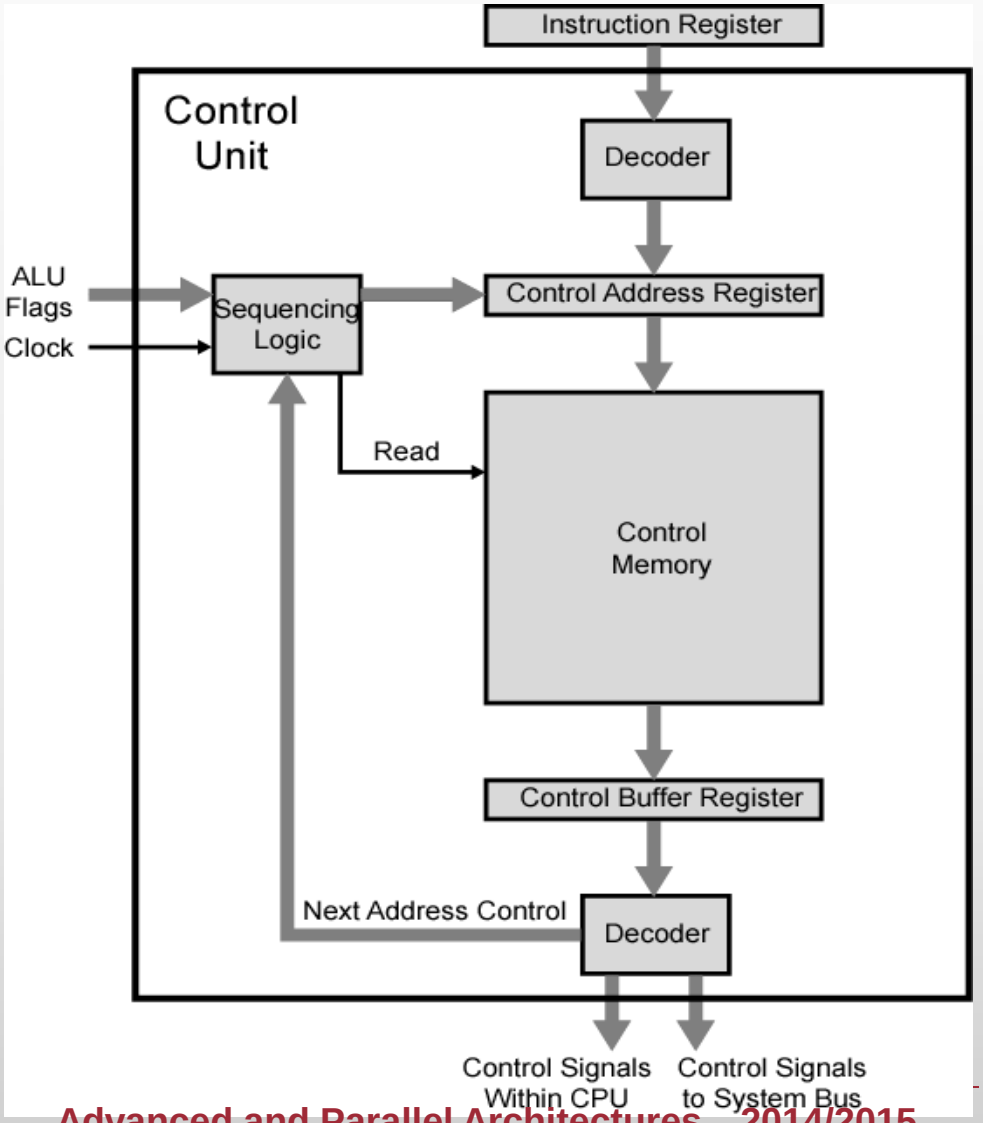
Control Unit



Control Unit Function

- Sequence login unit issues read command
- Word specified in control address register is read into control buffer register
- Control buffer register contents generates control signals and next address information
- Sequence login loads new address into control buffer register based on next address information from control buffer register and ALU flags

Functioning of Microprogrammed Control Unit



Instructions

What is an Instruction Set?

- The complete collection of instructions that are understood by a CPU
- Machine Code
- Binary
- Usually represented by assembly codes

Elements of an Instruction

- 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 consumption (well, programmers anyway) a symbolic representation is used
  - e.g. ADD, SUB, LOAD
- Operands can also be represented in this way
  - ADD A,B

Instruction Types

- Data processing
- Data storage (main memory)
- Data movement (I/O)
- Program flow control

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
- 2 addresses
  - One address doubles as operand and result - a = a + b
  - Reduces length of instruction
  - Requires some extra work
- 1 address (Common on early machines)
  - Implicit second address - Usually a register (accumulator)
- 0 addresses
  - All addresses implicit - Uses 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

Design Decisions

- Operation repertoire
  - How many ops? What can they do? How complex are they?
- Data types
- Instruction formats
  - Length of op code field
  - Number of addresses
- Registers
  - Number of CPU registers available
  - Which operations can be performed on which registers?
- Addressing modes
- RISC v CISC

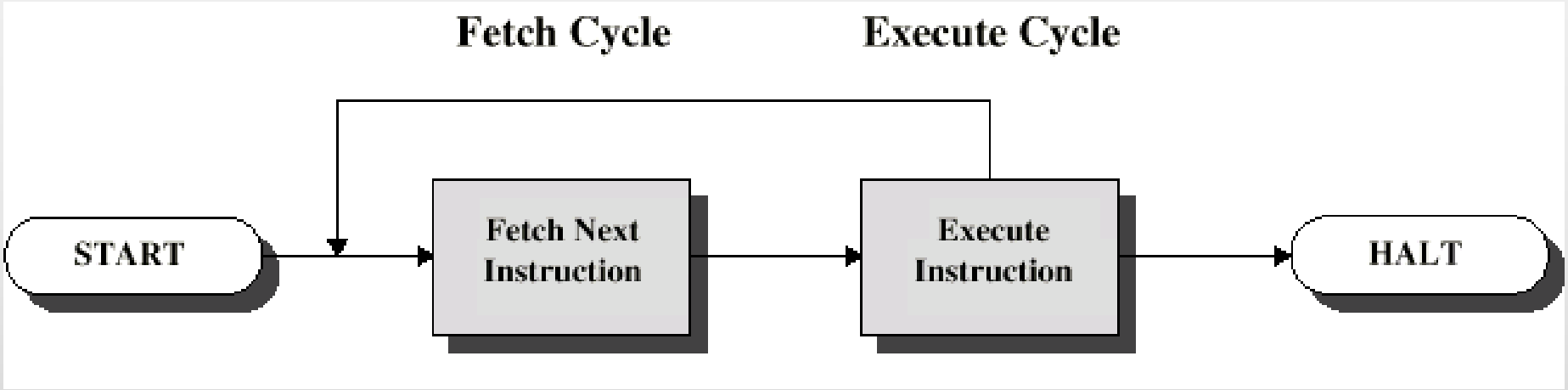
Types of Operand

- Addresses
- Numbers
  - Integer/floating point
- Characters
  - ASCII etc.
- Logical Data
  - Bits or flags
- (Aside: Is there any difference between numbers and characters? Ask a C programmer!)

Instructions execution

Instruction Cycle

- Two steps:
  - Fetch
  - Execute



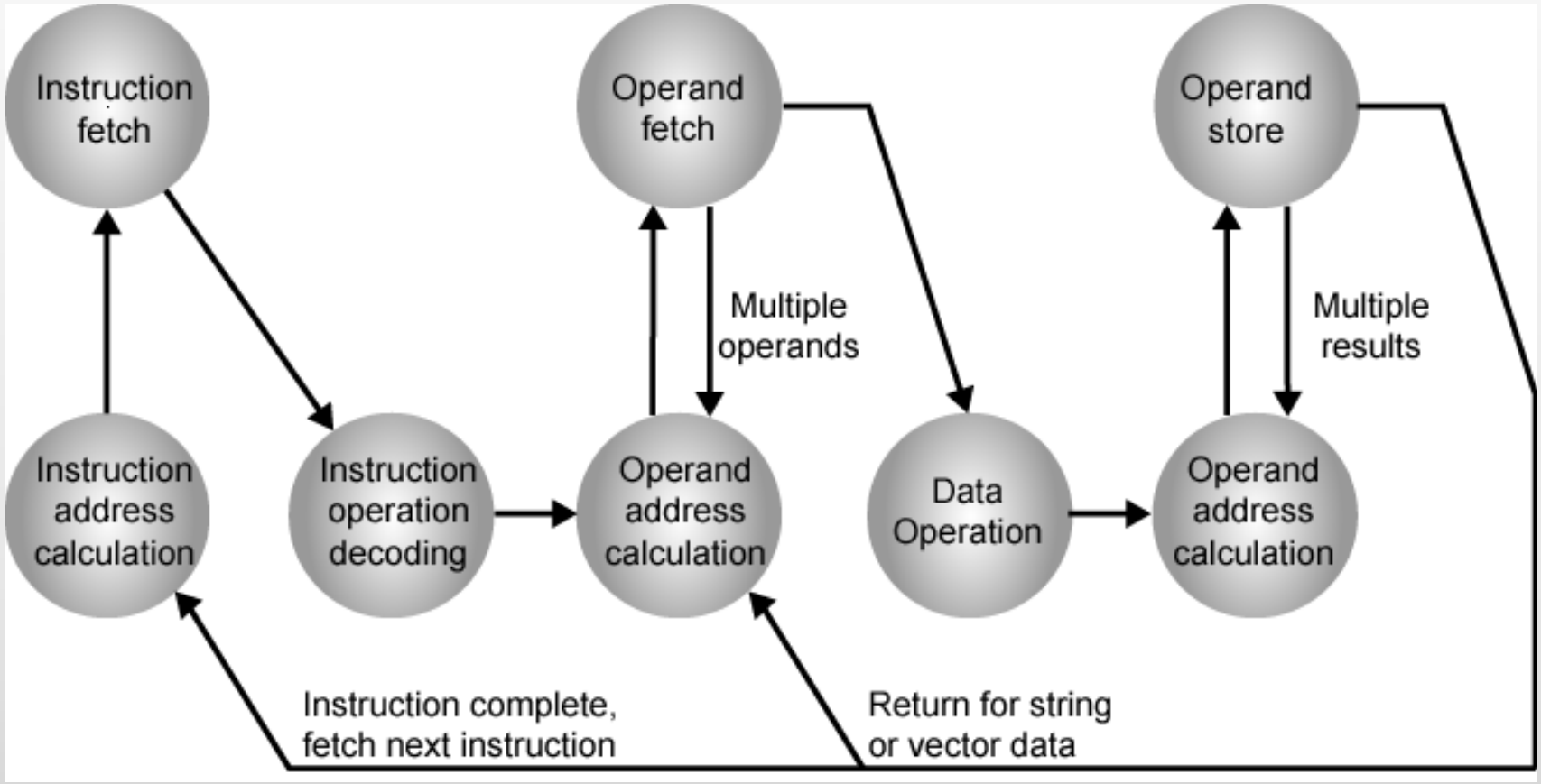
Fetch Cycle

- 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

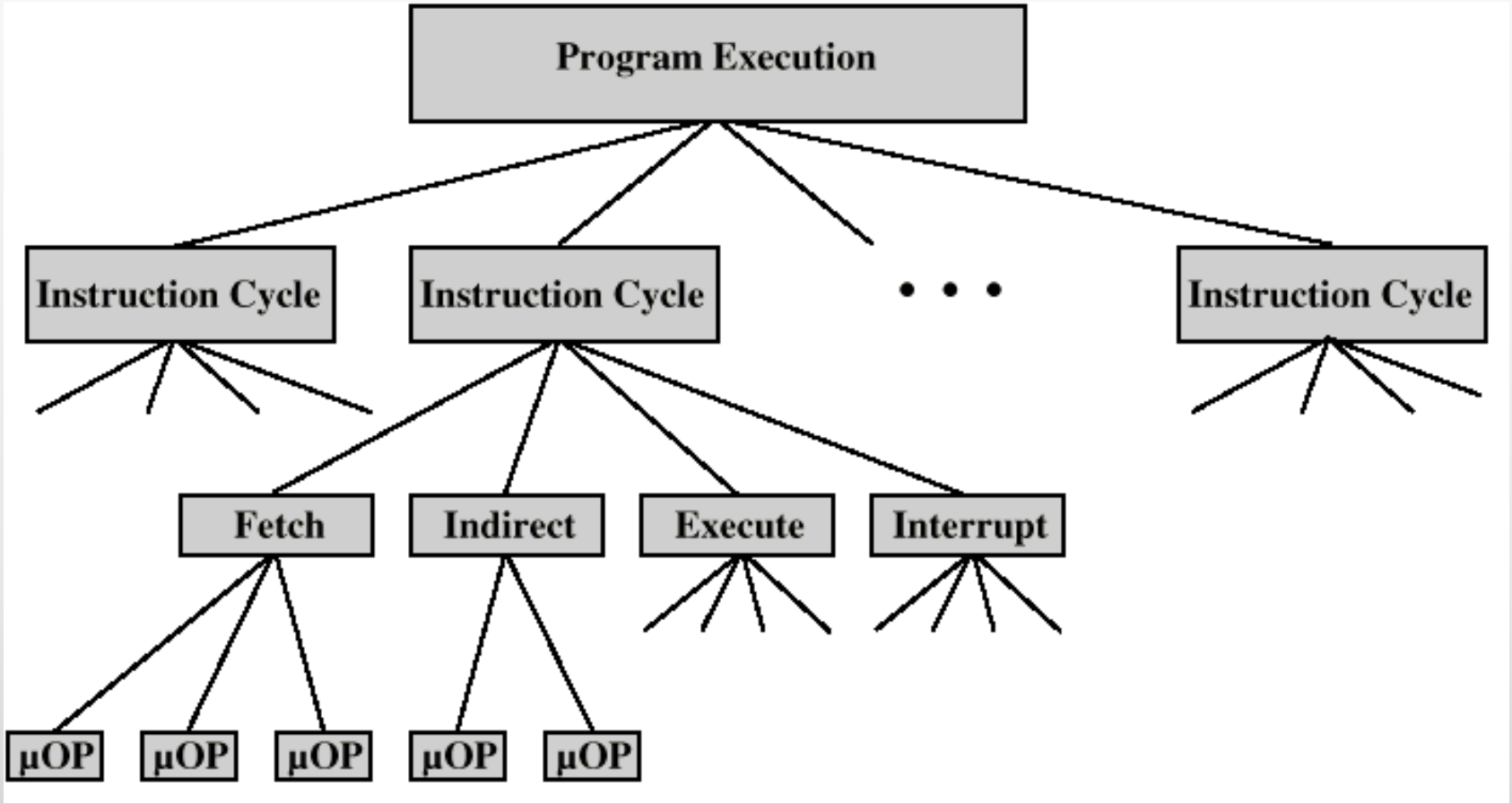
Execute Cycle

- 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 Cycle State Diagram



Constituent Elements of Program Execution



Instruction Cycle

- Each phase decomposed into sequence of elementary micro-operations
- E.g. fetch, indirect, and interrupt cycles
- Execute cycle
  - One sequence of micro-operations for each opcode
- Need to tie sequences together
- Assume new 2-bit register
  - Instruction cycle code (ICC) designates which part of cycle processor is in
    - 00: Fetch
    - 01: Indirect
    - 10: Execute
    - 11: Interrupt

Interrupts

- Mechanism by which other modules (e.g. I/O) 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

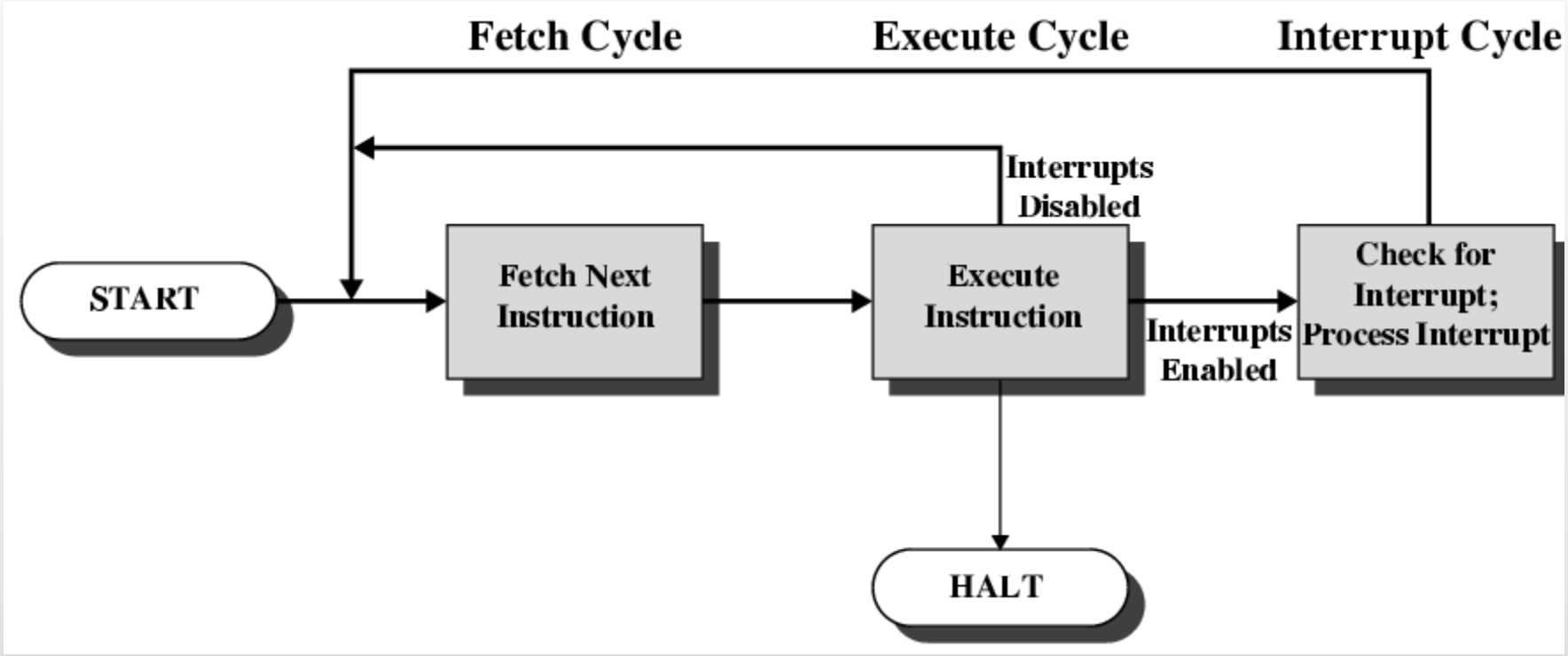
Multiple Interrupts

- 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

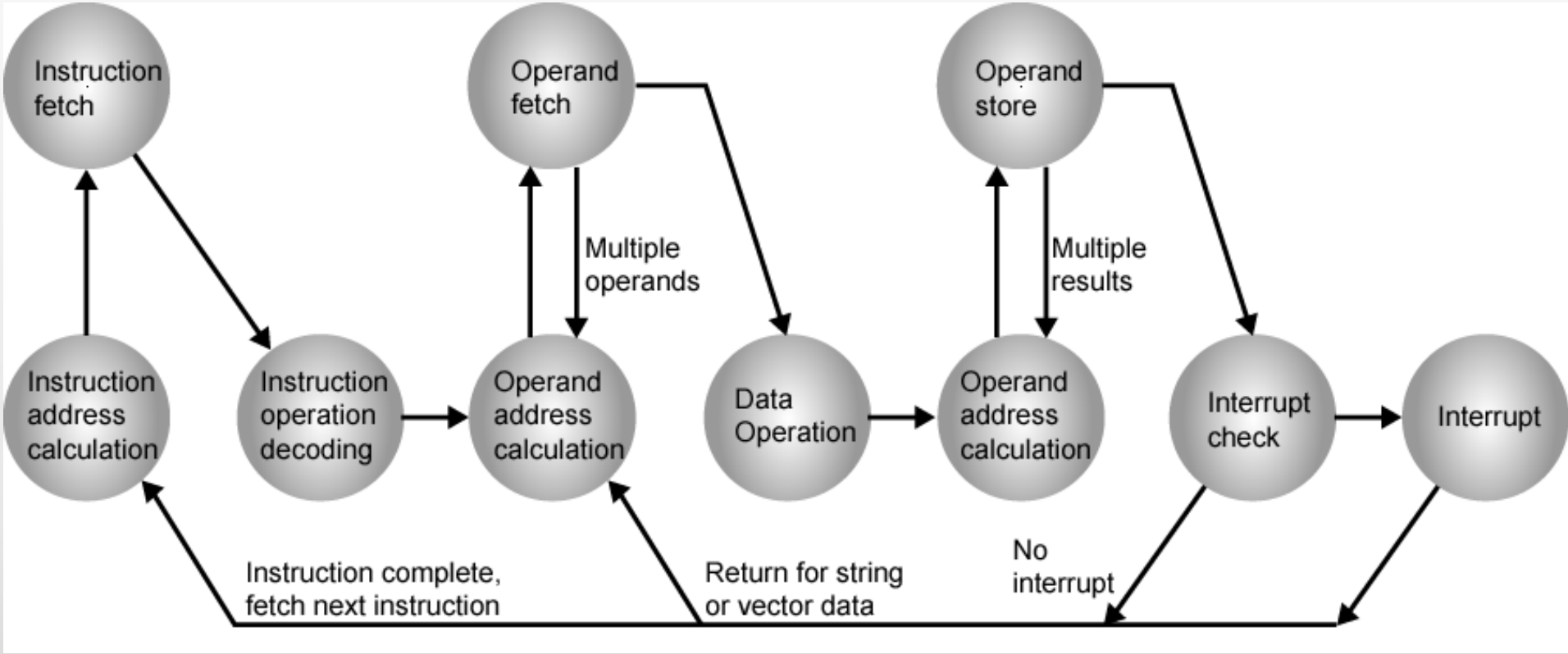
Interrupt Cycle

- 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

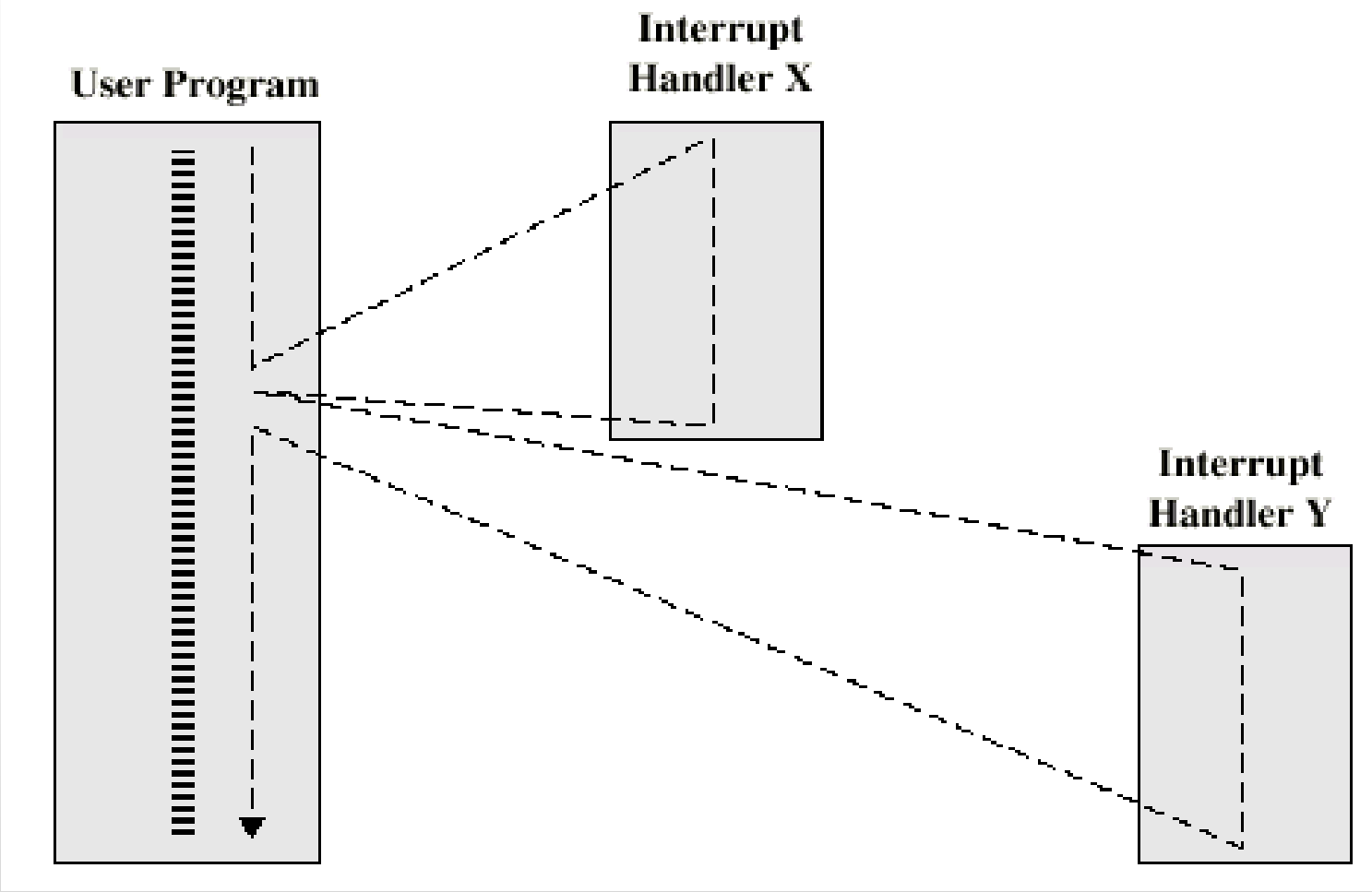
Instruction Cycle with Interrupts



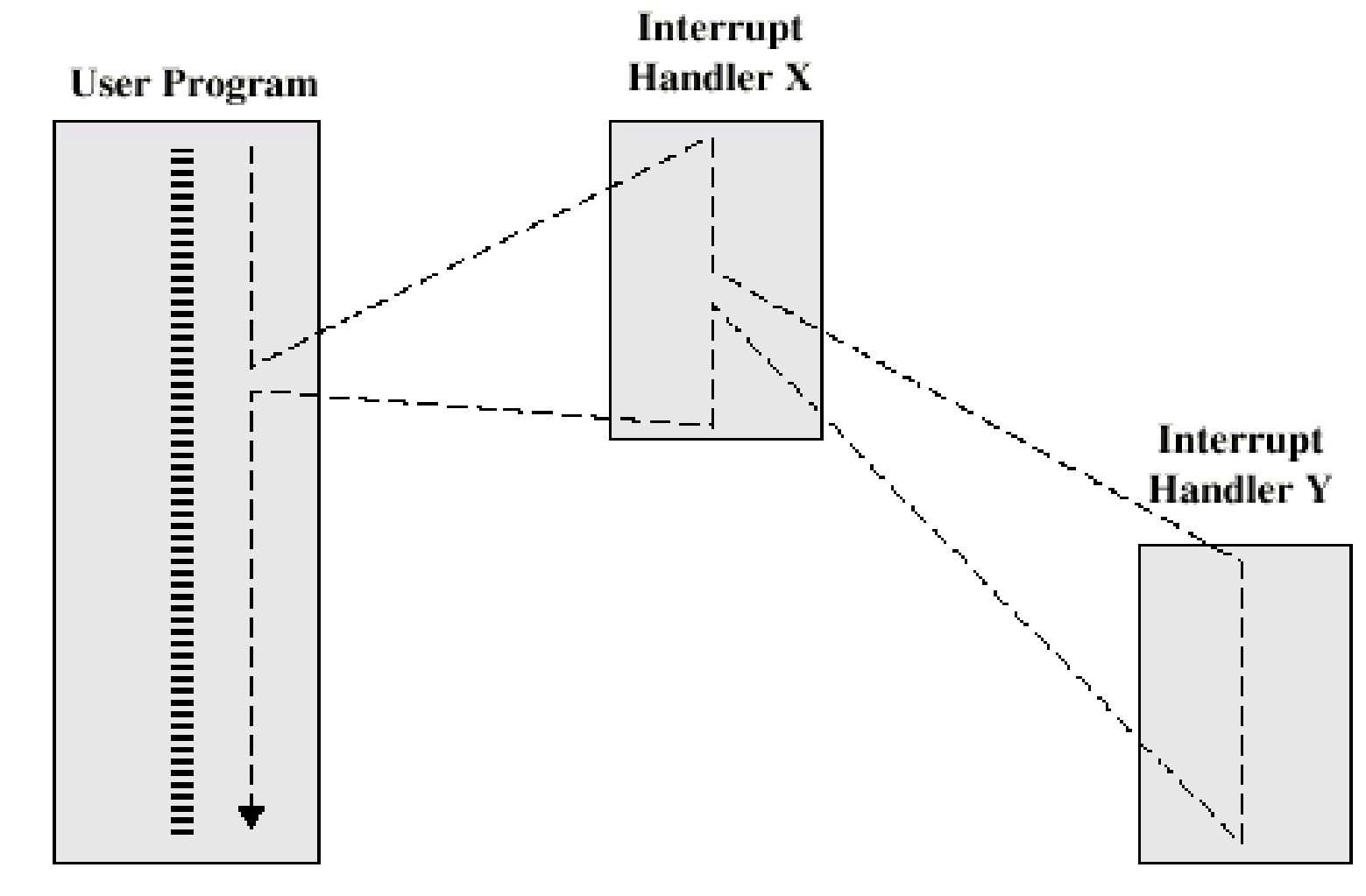
Instruction Cycle with Interrupts - State Diagram



Multiple Interrupts - Sequential



Multiple Interrupts – Nested



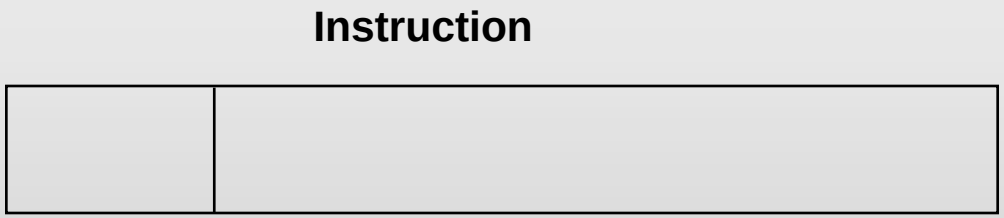
Addressing Modes and Formats

Addressing Modes

- Immediate
- Direct
- Indirect
- Register
- Register Indirect
- Displacement (Indexed)
- Stack

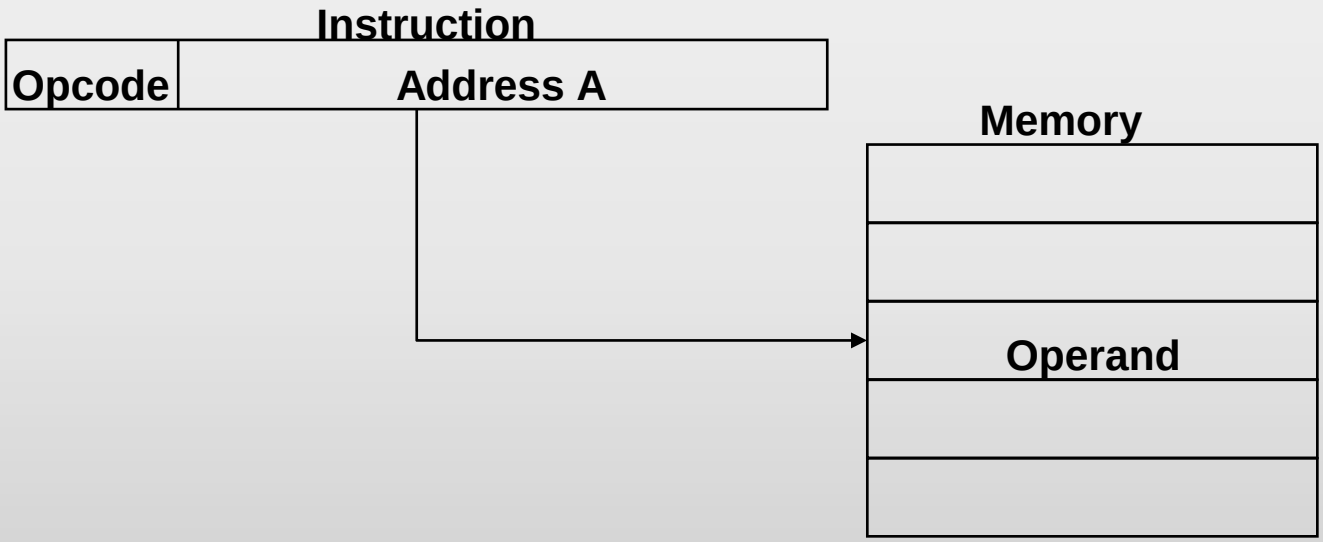
Immediate Addressing

- 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



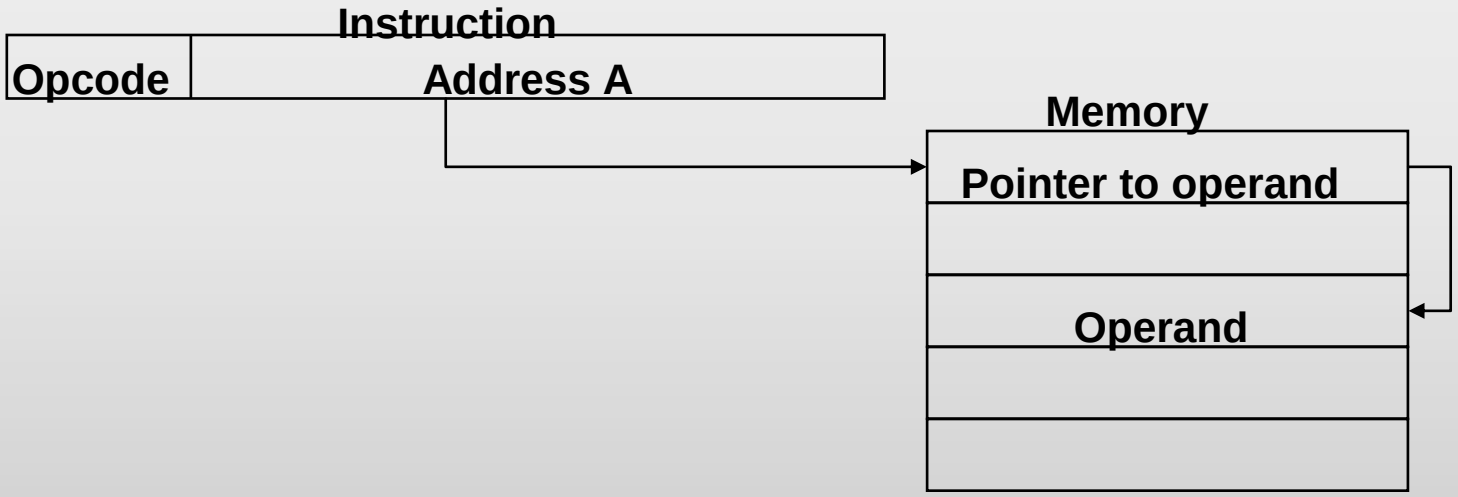
Direct Addressing

- 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



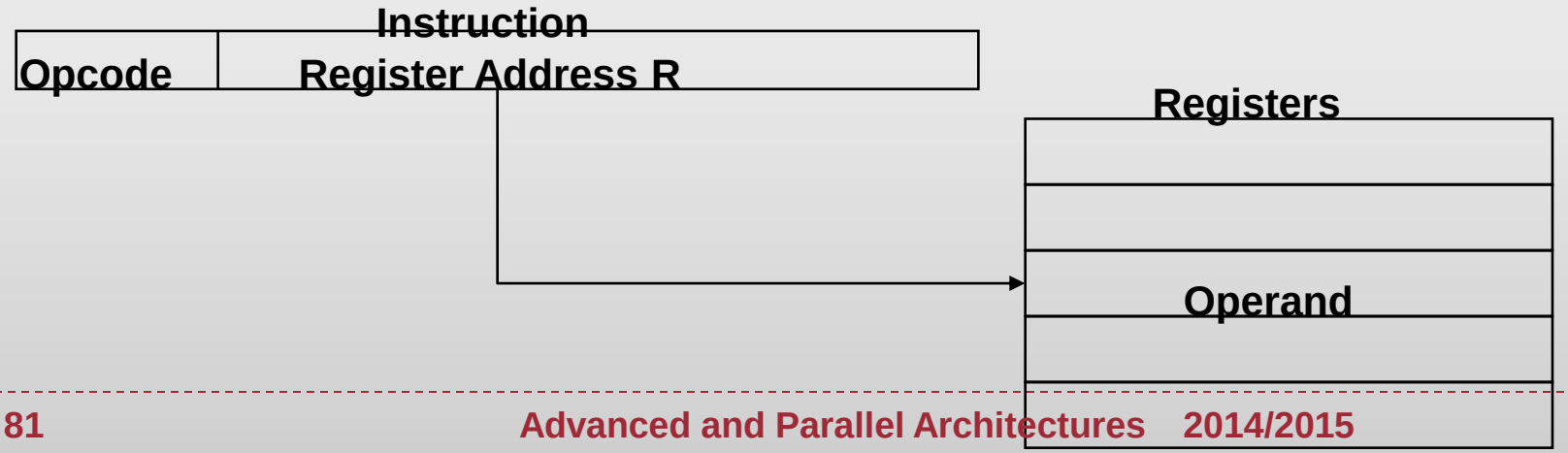
Indirect Addressing

- 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



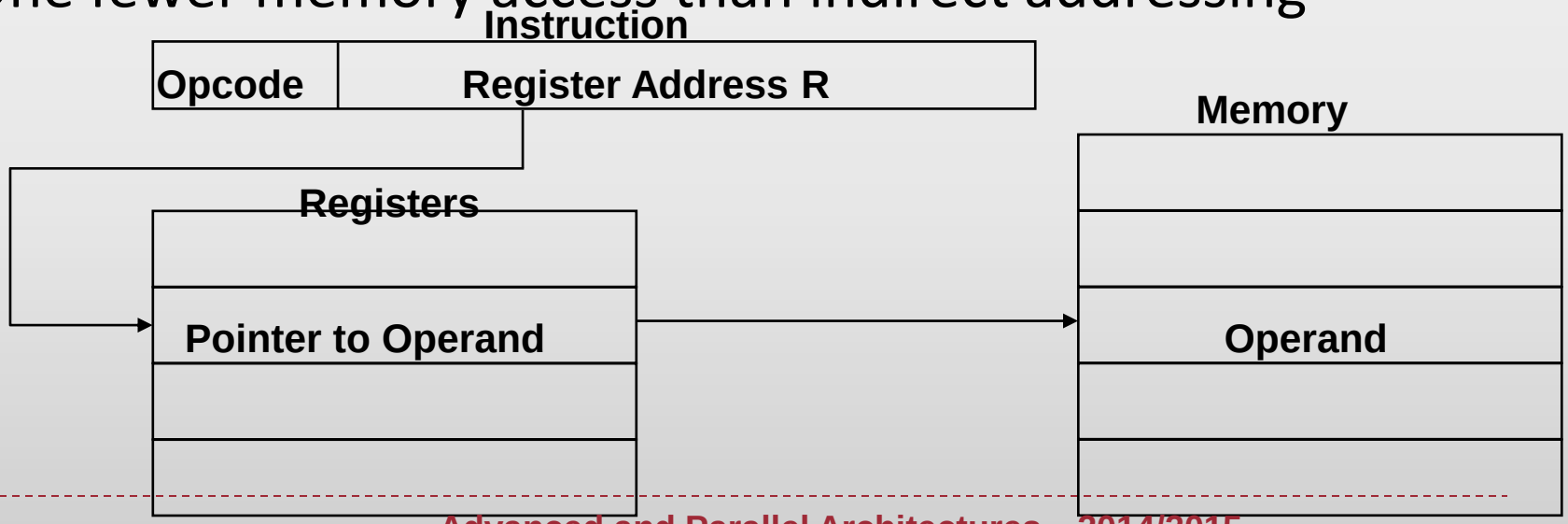
Register Addressing

- Operand is held in register named in address filed
- 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



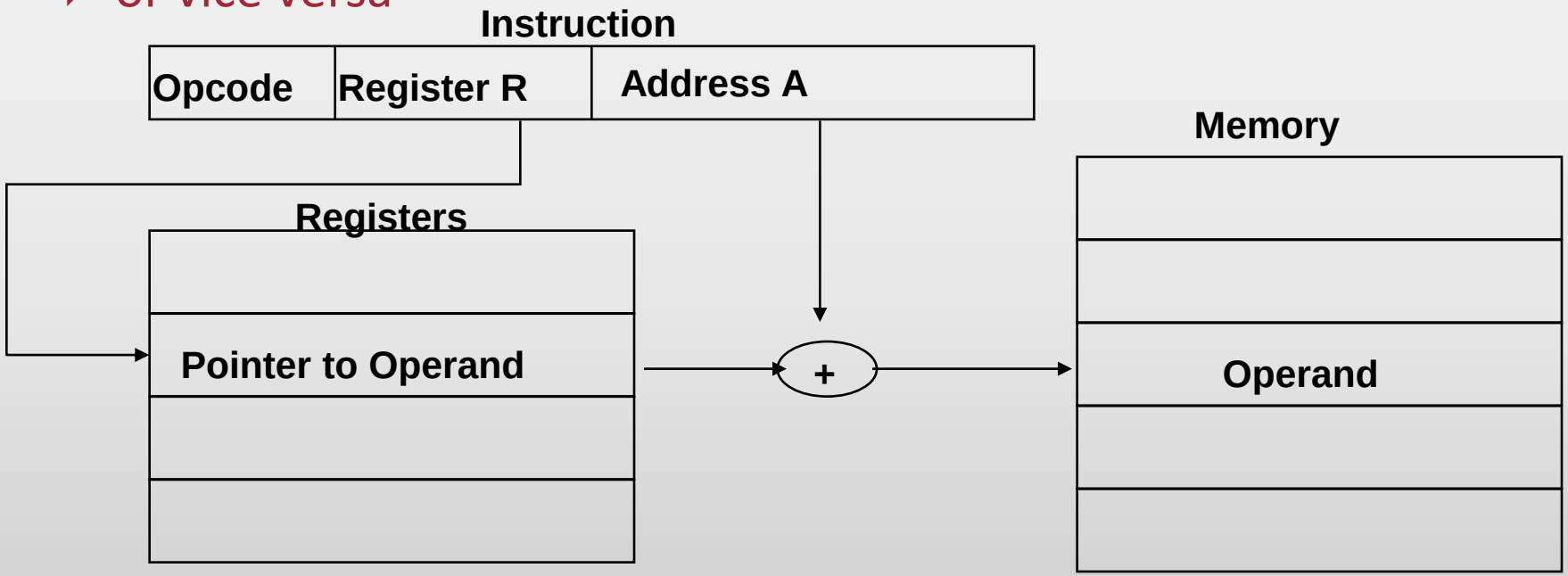
Register Indirect Addressing

- 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 = 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 = base
- R = displacement
- EA = A + R
- Good for accessing arrays
  - EA = A + R
  - R++

Instruction Formats

- Layout of bits in an instruction
- Includes opcode
- Includes (implicit or explicit) operand(s)
- Usually more than one instruction format in an instruction set

Instruction Length

- Affected by and affects:
  - Memory size
  - Memory organization
  - Bus structure
  - CPU complexity
  - CPU speed
- Trade off between powerful instruction repertoire and saving space