CS222 Computer Organization and Architecture

Hardwired and Micro Programmed CU

Ref: Chap 15 & 16 of the Text Book: Organization and Architecture By W. Stalling

S. Tripathy

Course web

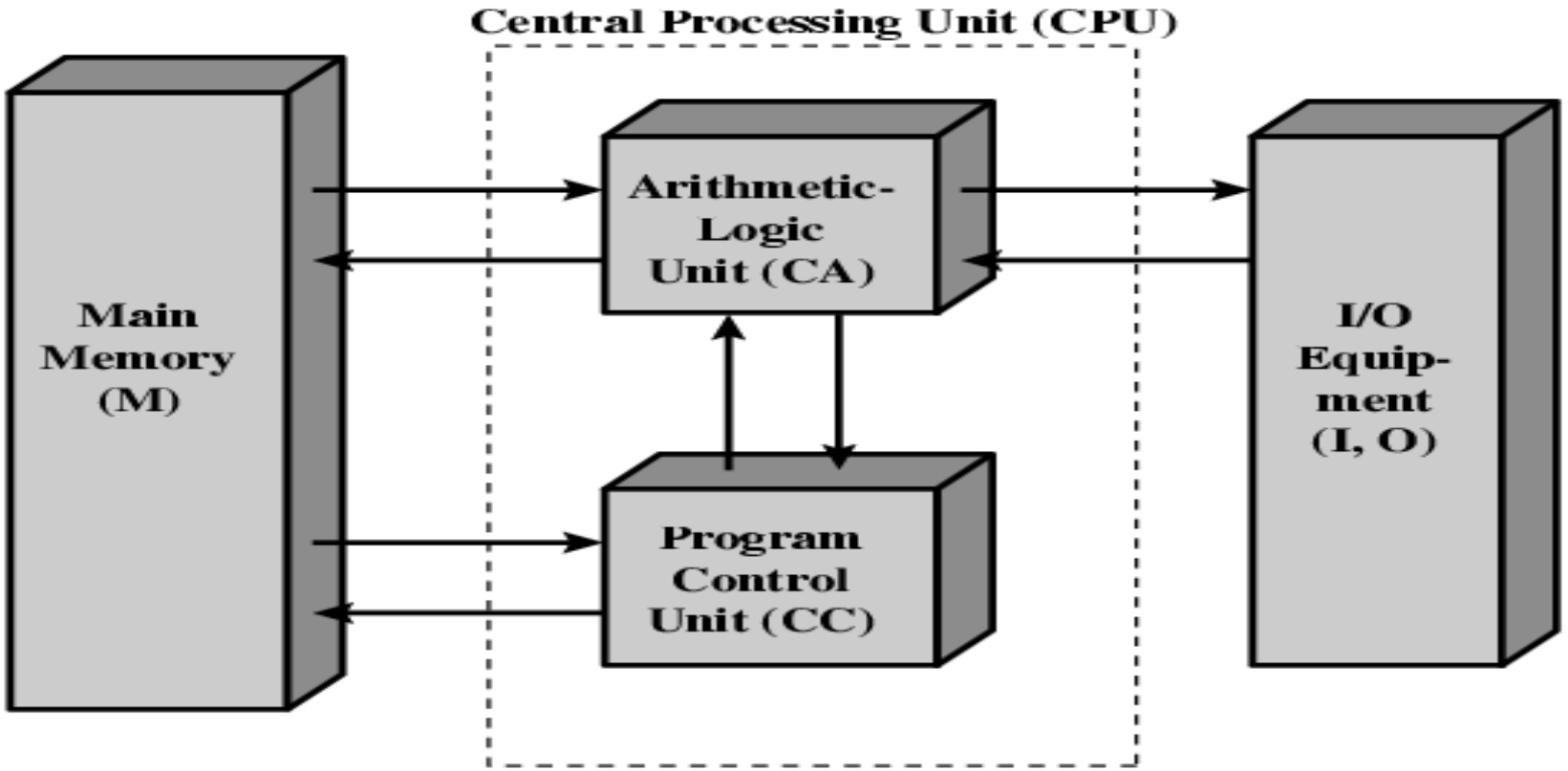
- Information Related to CS 222 (COA)
  - 172.16.1.3/~som/CS222
- Information Related to CS 223 (Lab)
  - 172.16.1.3/~som/CS223
  - Email: som@iitp.ac.in

Before Start!!!

- Why do you study computer architecture?
  - Is it to design a new computer?
  - To know how the computer works
  - How to achieve the high performance
  - Using a high speed processor can the performance increased as desired?

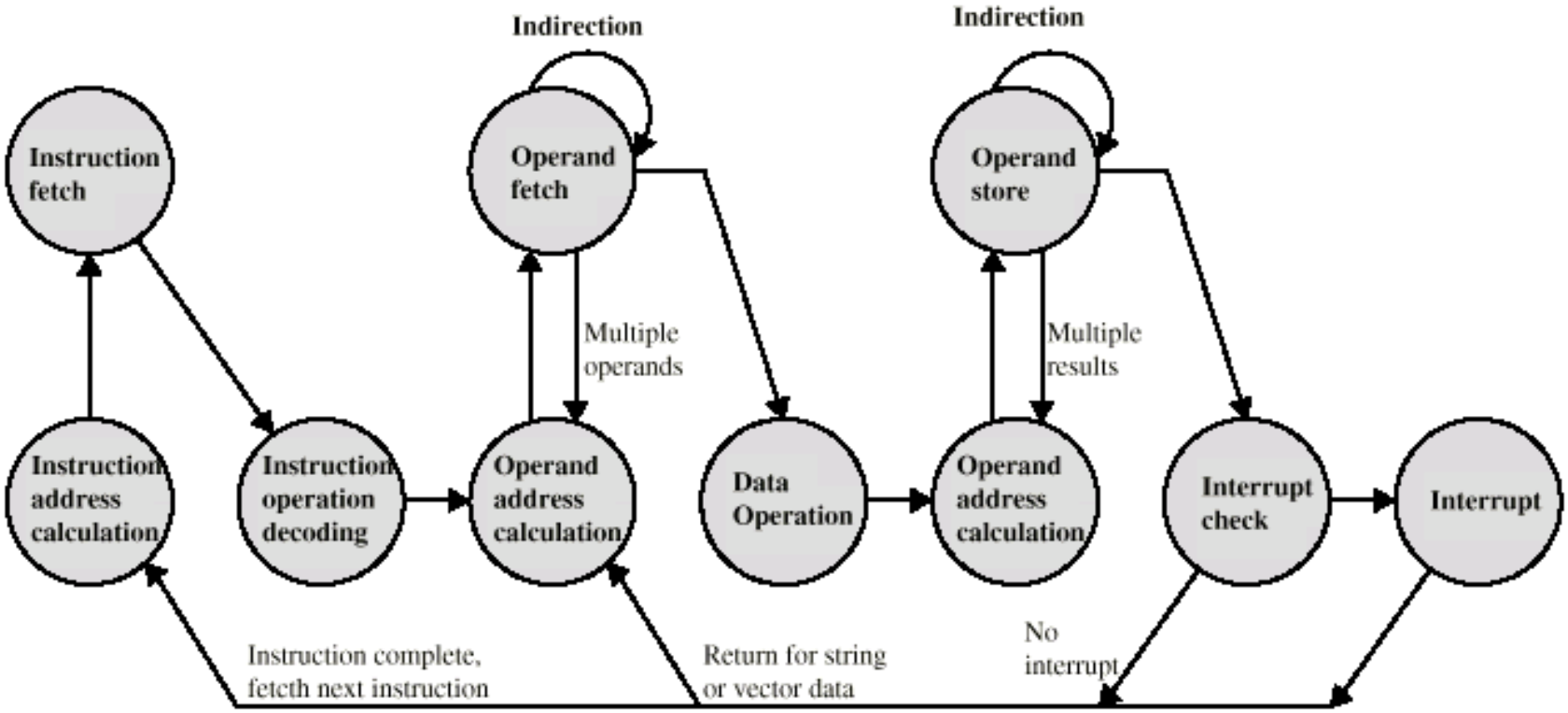
von Neumann Machine(1946)

- Stored Program concept
- Princeton Institute for Advanced Studies
  - IAS



Central Processing Unit

- Instruction Cycle State diagram



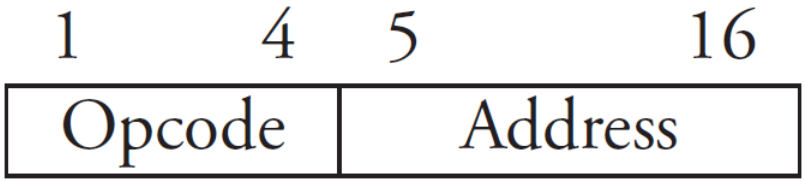
Instruction Execution

- CPU executes a sequence of instruction to execute a program
- Each instruction execution process is referred as instruction cycle carried out by several steps

- Each cycle has a number of steps
  - Ex. Instruction Fetch has 4 steps

CPU

- CPU is a sequential circuit
- repeatedly reads and executes an instruction from its memory
  - fetch-and-execute cycle
- A ***machine language*** program is a set of instructions drawn from the CPU instruction set
- Each instruction consists of two parts: an **opcode** and an **address**



Design of a Simple CPU

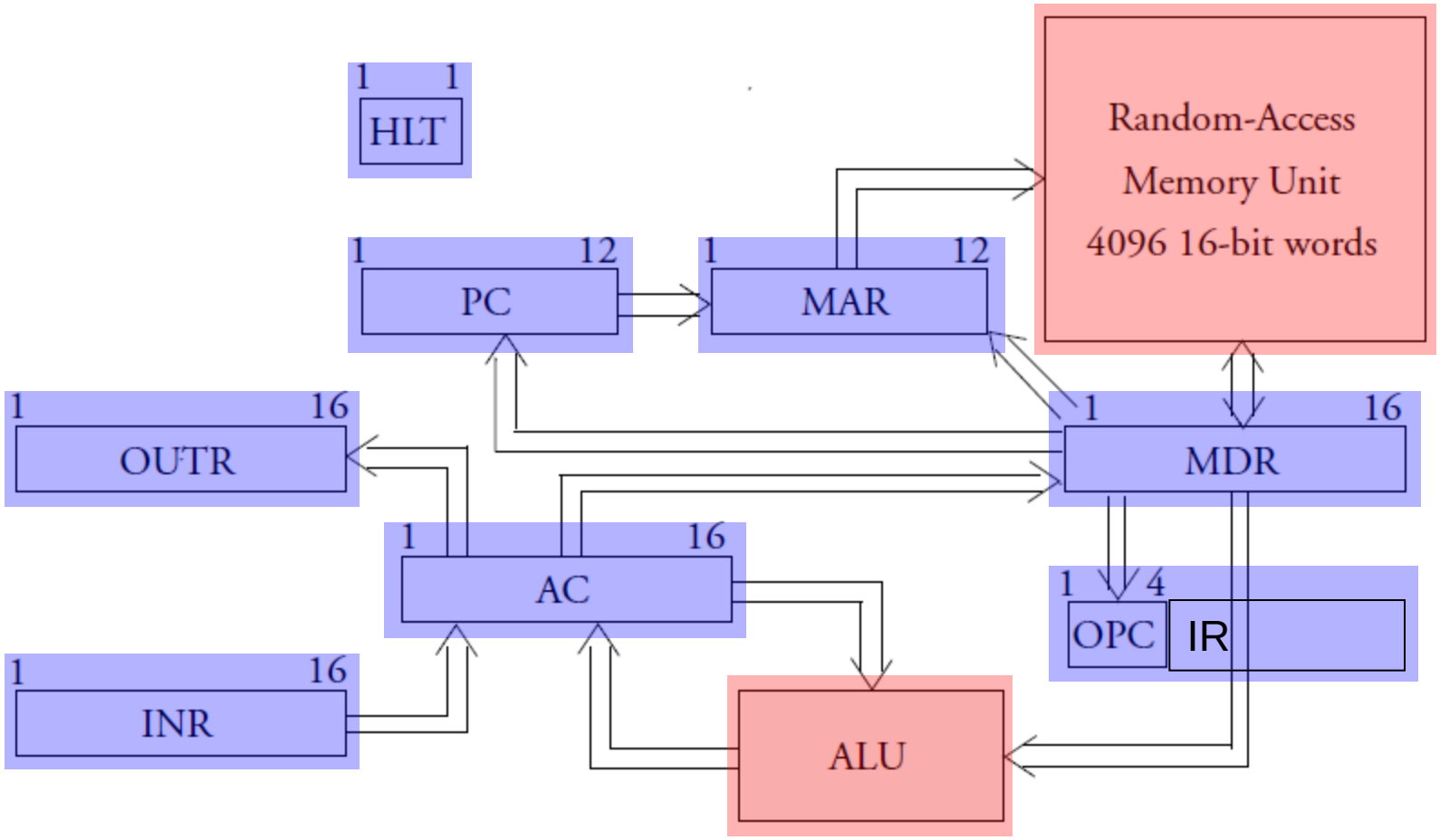
- 11-instruction CPU with 2^{12} (4096) 16-bit RAM.
- The design is simplified for clarity

- CPU design can be very complex.
- Since the CPU is the heart of every computer, emphasis is put on making it fast

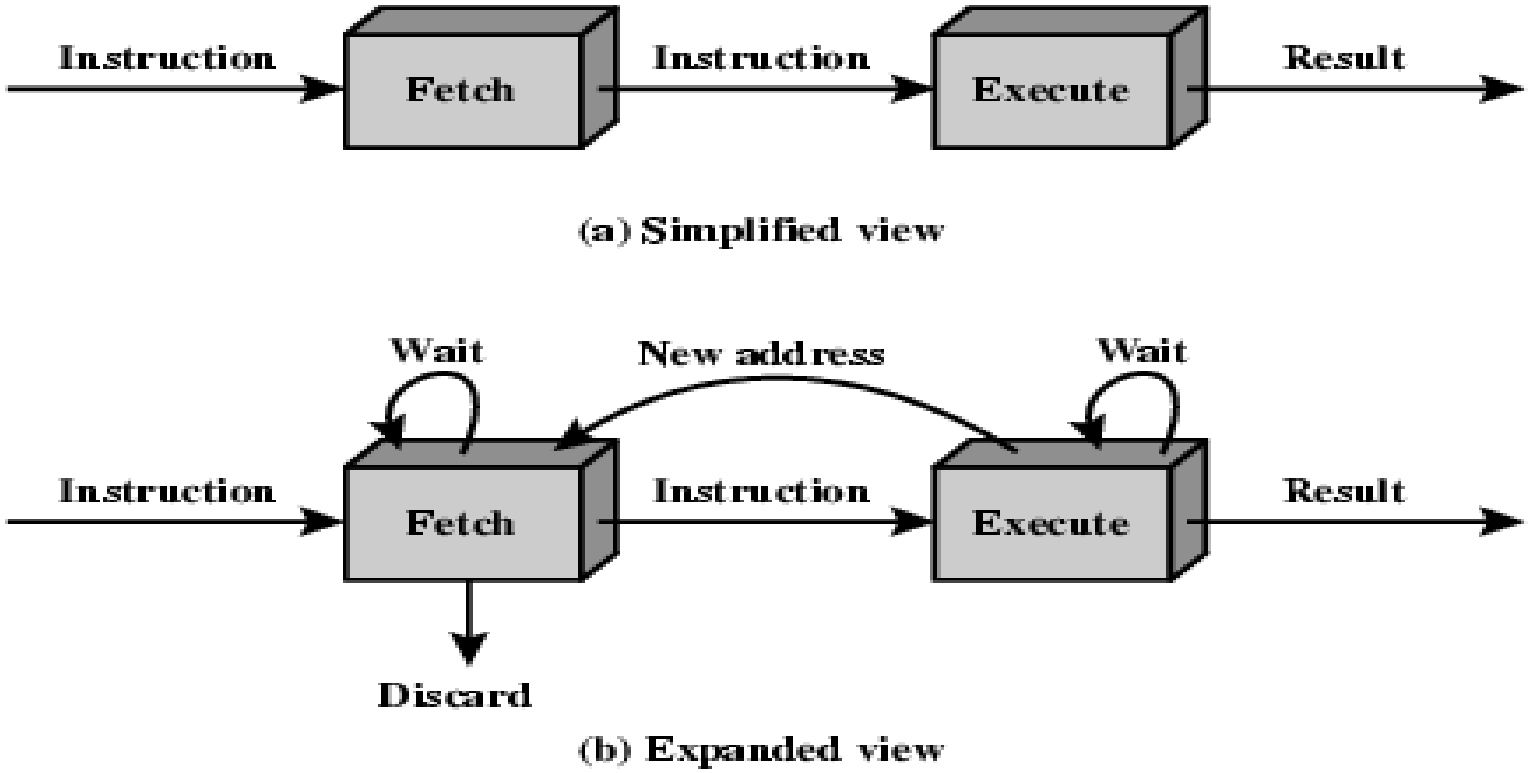
Our Simple CPU

- 11 instructions require a 4-bit opcode.
- 12 bits of the 16-bit word remain for addressing 4096 16-bit words of the RAM.
- The CPU has (PC)
- 4-bit ***opcode*** (OPC)
- 12-bit ***memory address register*** (MAR)
- 16-bit ***memory data register*** (MDR)
- 16-bit ***input register*** (INR)
- 16-bit ***output register*** (OUTR)
- 1-bit ***halt register*** (HLT) 8 special registers:
- 16-bit ***accumulator*** (AC)
- 12-bit ***program counter***

The Registers Data Transfer in the CPU



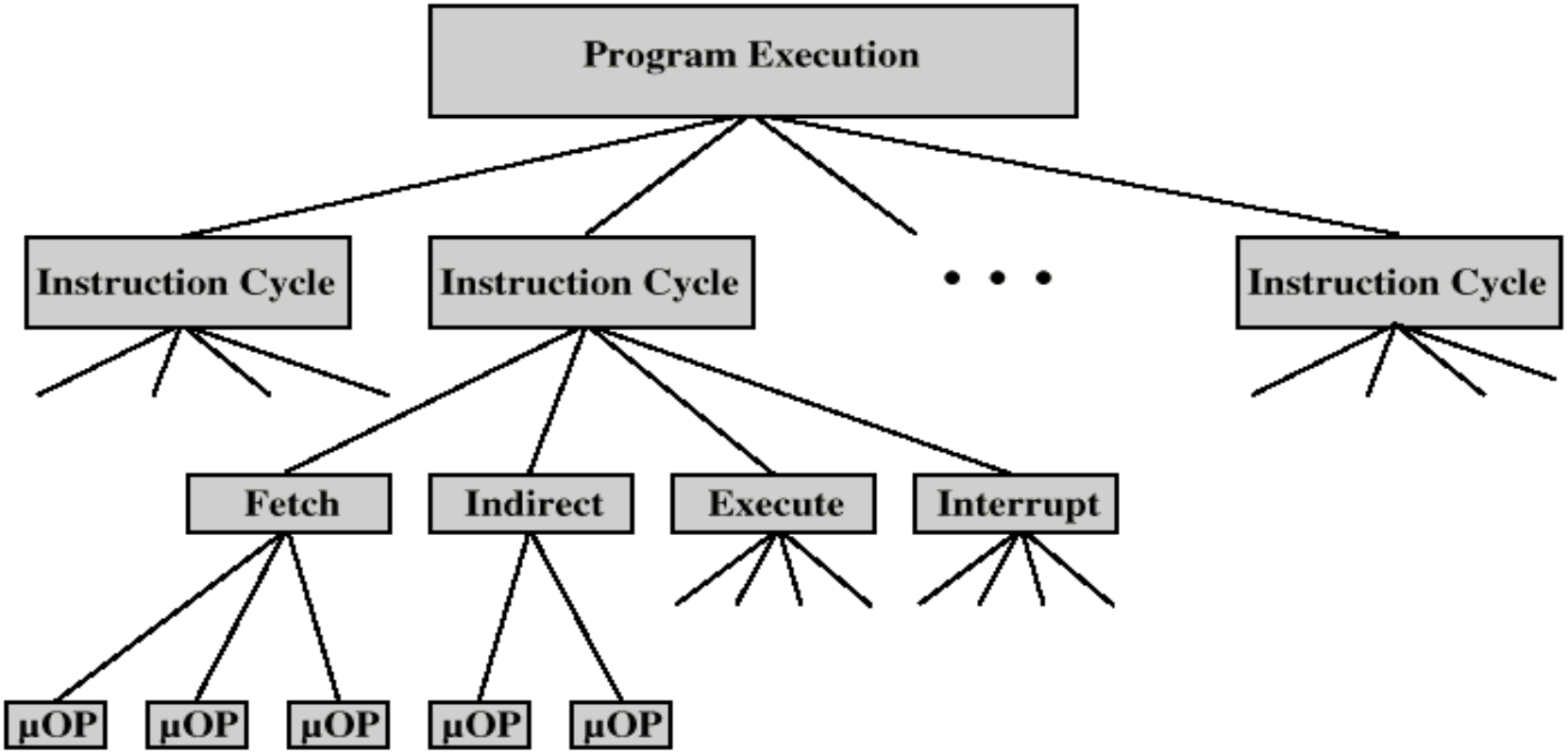
The Fetch-and-Execute



- Fetch portion is always the same:
- Instruction, whose address is in the PC, is fetched into the MDR
- Opcode portion of this register is copied into the OPC

Micro-Operations

- Each step does very little
  - Atomic operation of CPU
  - Called micro-operations



Fetch Sequence (symbolic)

- t1: MAR <- (PC)
- t2: MBR <- memory
- PC <- (PC) +1
- t3: IR <- (MBR)
  - (tx = time unit/clock cycle)
- or
- t1: MAR <- (PC)
- t2: MBR <- memory
- t3: PC <- (PC) +1
- IR <- (MBR)

Indirect Cycle

- T1:MAR <- ($IR_{address}$)
- T2: MBR <- memory
- T3: $IR_{address}$ <- (MBR)

- MBR contains an address
- IR is now in same state as if direct addressing had been used

Interrupt Cycle

- t1: MBR <-(PC)
- t2:MAR <- save-address
- PC <- routine-address
- t3:memory <- (MBR)
- This is a minimum
  - May be additional micro-ops to get addresses
  - No context saving is showed here! Why?

- Thus the action of the CPU is based on the OPC
- Suppose OPC is a **load accumulator** instruction.
  - The action required is to copy the word specified by the address part of the instruction into the accumulator
- **load accumulator** is decomposed into 8 **microinstructions** executed in 6 **microcycles**

Cycle	*Microinstruction*	*Microinstruction*
1	Copy contents of PC to MAR.	
2	Fetch word at address MAR into MDR.	Increment PC.
3	Copy opcode part of MDR to OPC.	
4	Interpret OPC	Copy address part of MDR to MAR.
5	Fetch word at address MAR into MDR.	
6	Copy MDR into AC.	

The Instruction Set

	Opcode	*Binary*	*Description*

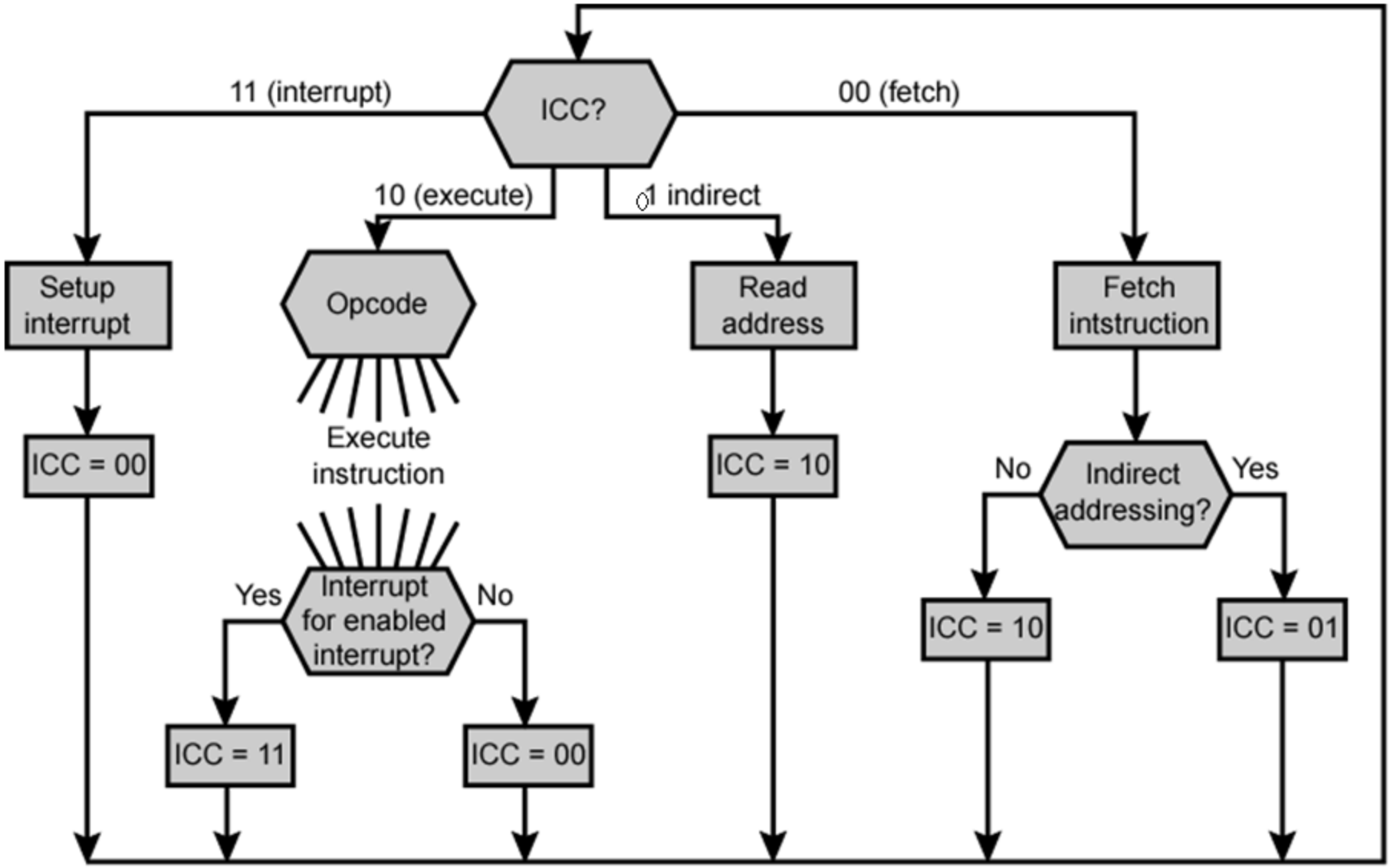
This is a ***direct memory instruction***

- Addresses refer directly to memory locations containing the ***operands*** (data) on which the program operates

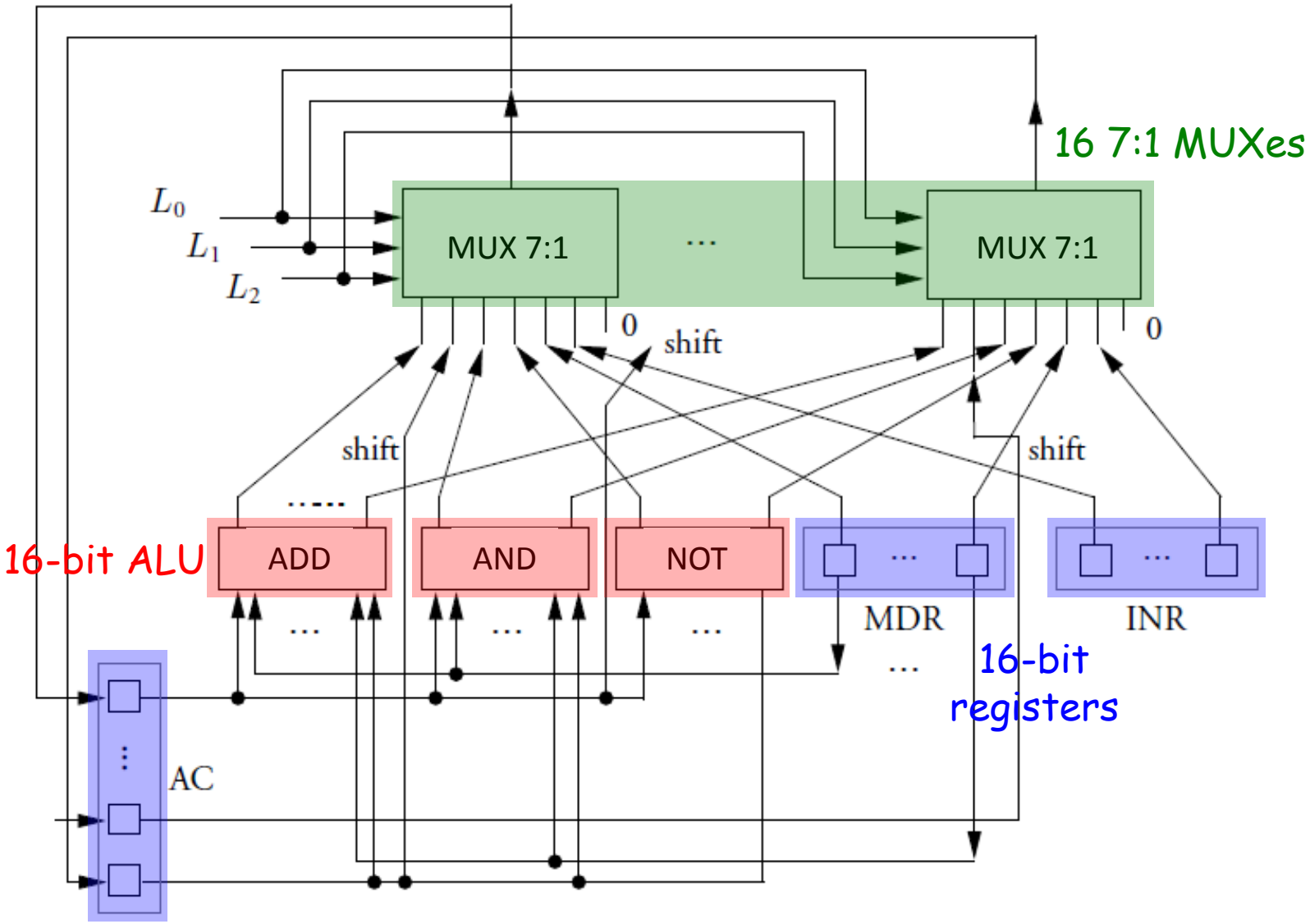
CPUs have also ***indirect memory instructions***

- Instructions in which an address is interpreted as the address at which to find the address containing the needed operand
- CPU does two memory fetches
  - the first to find the address of the operand
  - the second to find the operand itself

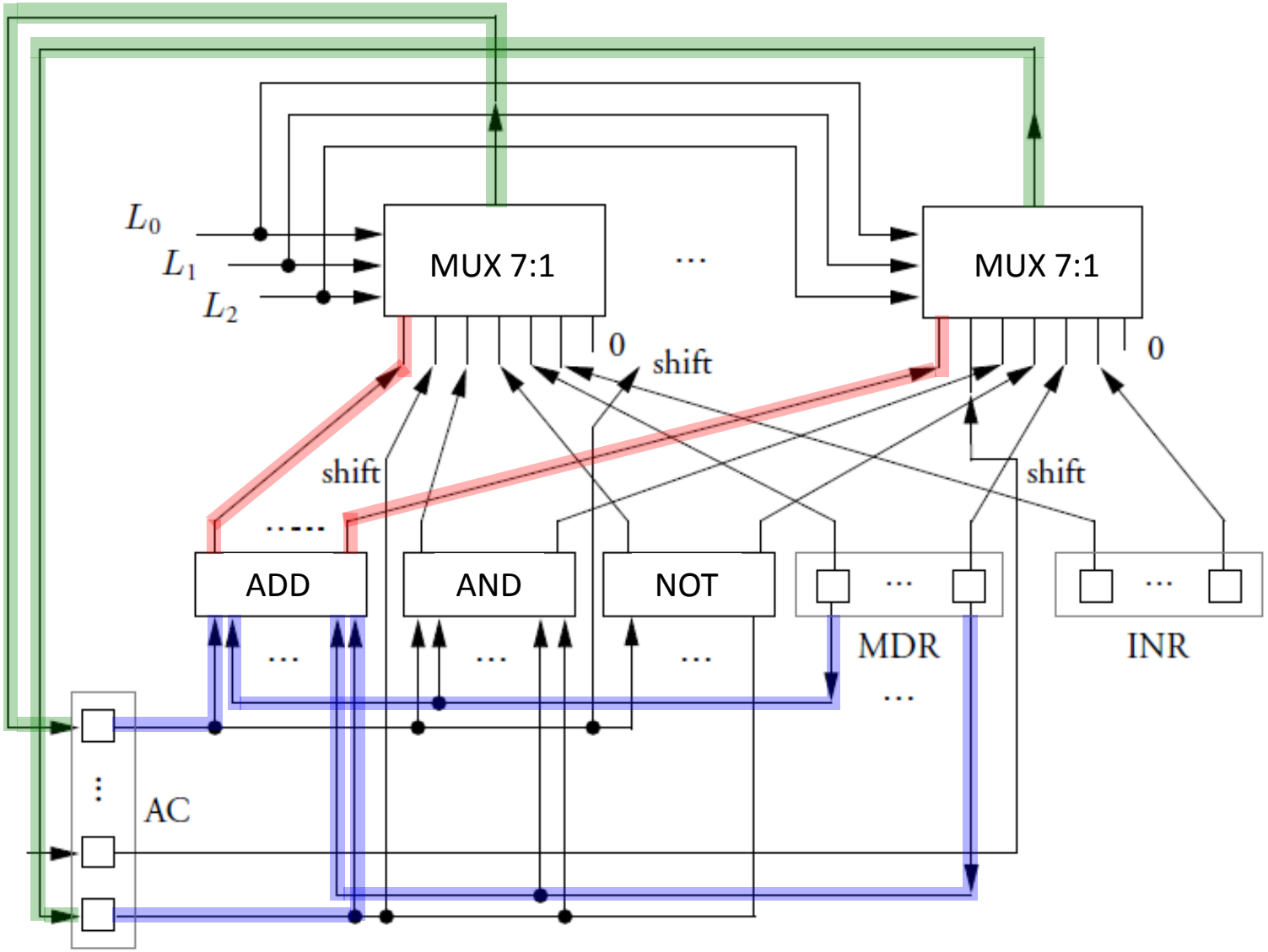
Flowchart for Instruction Cycle



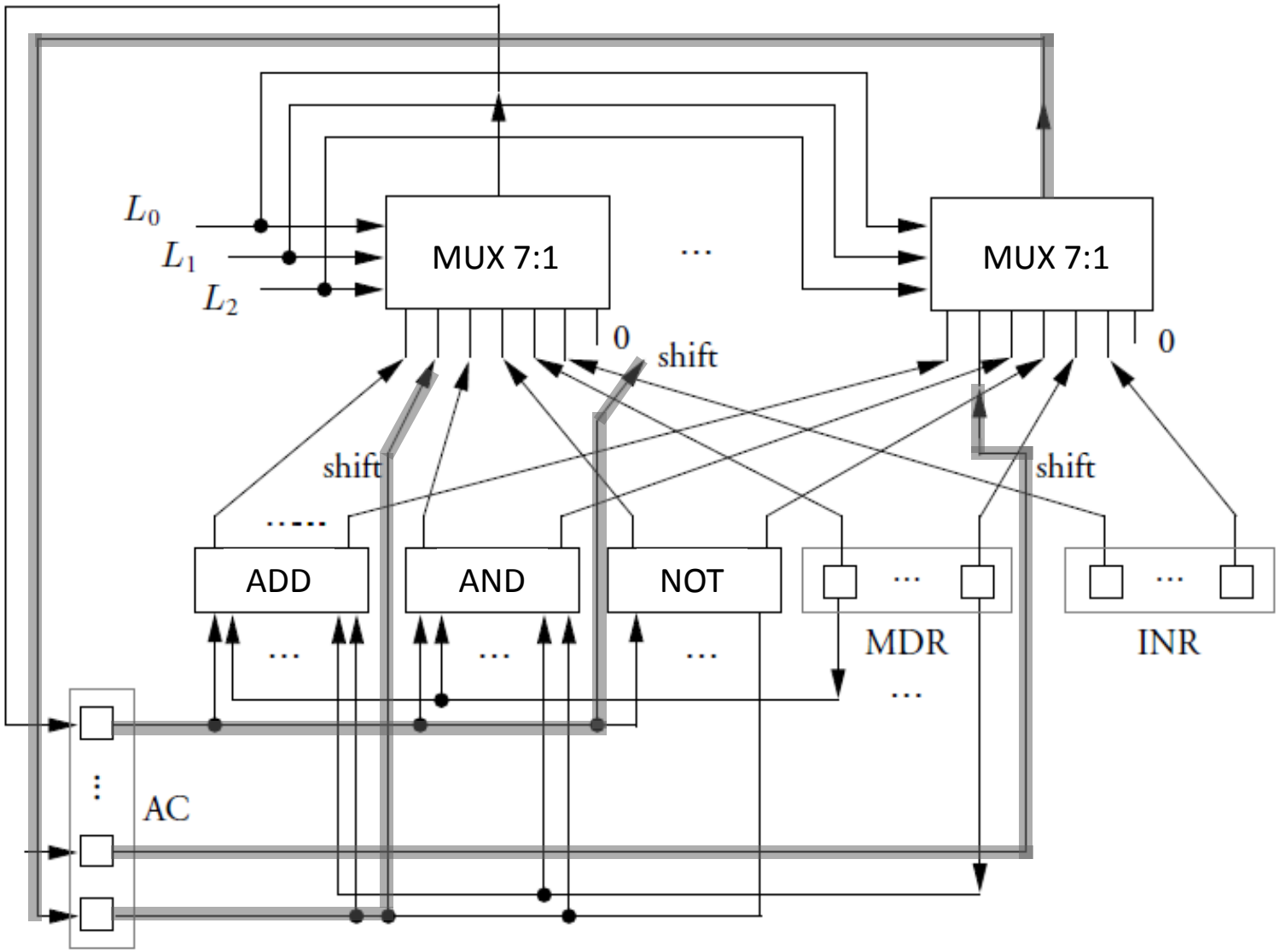
Hardware Implementation



Execution of $AC \leftarrow AC + MDR$



Execution of $AC \leftarrow Shift(AC)$



Timing and Control

Control Unit: We need to design circuits that controls the operation and movement of data.

Microinstructions:

Register transfer notation uses the assignment operations as well as timing information to break down a machine-level instruction into ***microinstructions*** that are executed in successive microcycles.

microinstructions for the Fetch portion

	Timing	*Microinstructions*
timing variable t_j $1 \leq j \leq k$	t_1	MAR ← PC
	t_2	MDR ← M, PC ← PC+1
	t_3	OPC ← MDR

The microcode for the fetch portion of each instruction.

microinstructions for the execute portion

Control		*Microcode*	
ADD			
c_{ADD}	t_4	$\text{MAR} \leftarrow \text{MDR}$	($\text{IR}_{\text{address}}$)
c_{ADD}	t_5	$\text{MDR} \leftarrow \text{M}$	
c_{ADD}	t_6	$\text{AC} \leftarrow \text{AC} + \text{MDR}$	
AND			
c_{AND}	t_4	$\text{MAR} \leftarrow \text{MDR}$	
c_{AND}	t_5	$\text{MDR} \leftarrow \text{M}$	
c_{AND}	t_6	$\text{AC} \leftarrow \text{AC AND MDR}$	

instruction variable

instruction variable

Control		*Microcode*
CLA		
c_{CLA}	t_4	AC $\leftarrow$ 0
CIL		
c_{CIL}	t_4	AC $\leftarrow$ Shift(AC)
LDA		
c_{LDA}	t_4	MAR $\leftarrow$ MDR
c_{LDA}	t_5	MDR $\leftarrow$ M
c_{LDA}	t_6	AC $\leftarrow$ MDR

Control		*Microcode*
STA		
c_{STA}	t_4	MAR ← MDR
c_{STA}	t_4	MDR ← AC
c_{STA}	t_5	M ← MDR
CMA		
c_{CMA}	t_4	AC ← ¬ AC
JZ		
c_{JZ}	t_4	if (AC = 0) PC ← MDR

instruction variable

instruction variable

Control		*Microcode*
IN		
c_{IN}	t_4	AC $\leftarrow$ INR
OUT		
c_{OUT}	t_4	OUTR $\leftarrow$ AC
HLT		
c_{HLT}	t_4	$t_j \leftarrow 0$ for $1 \leq j \leq k$

Generating Control Signals

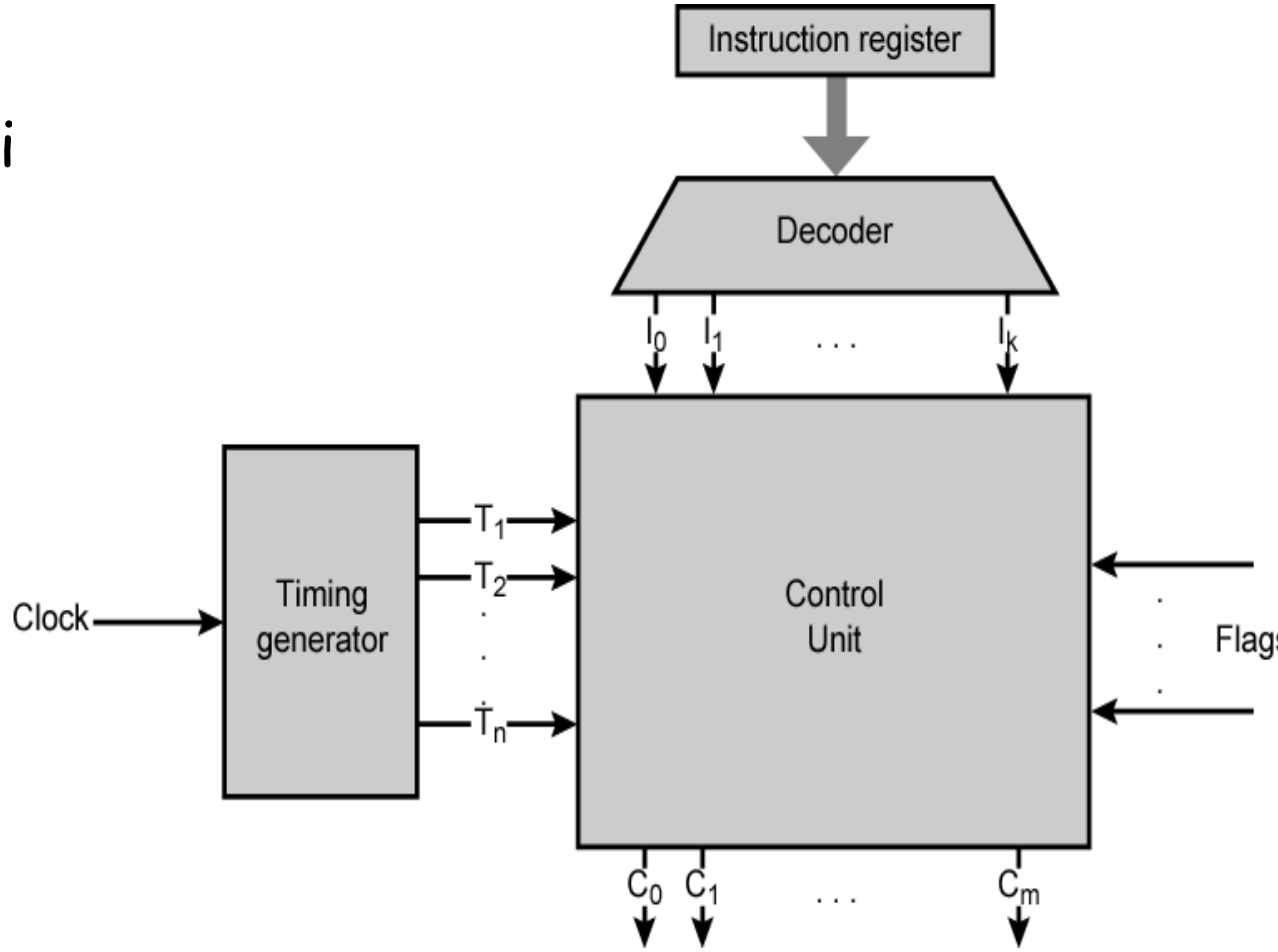
Define ***control variables*** that control the movement of data between registers or combine the contents of two registers and assign the result to another register.

Associate a control variable L(A,B) if a microinstruction results in the movement of data from register B to register A, denoted $A \leftarrow B$.

L(OPC,MDR) = t_3 : (IR) OPC is loaded with the contents of MDR when $t_3 = 1$

Hardwired Implementation

- Control unit is a combinatorial circui
  - Inputs:
    Flags
    and control bus
  - Outputs:
    Control signal(s)

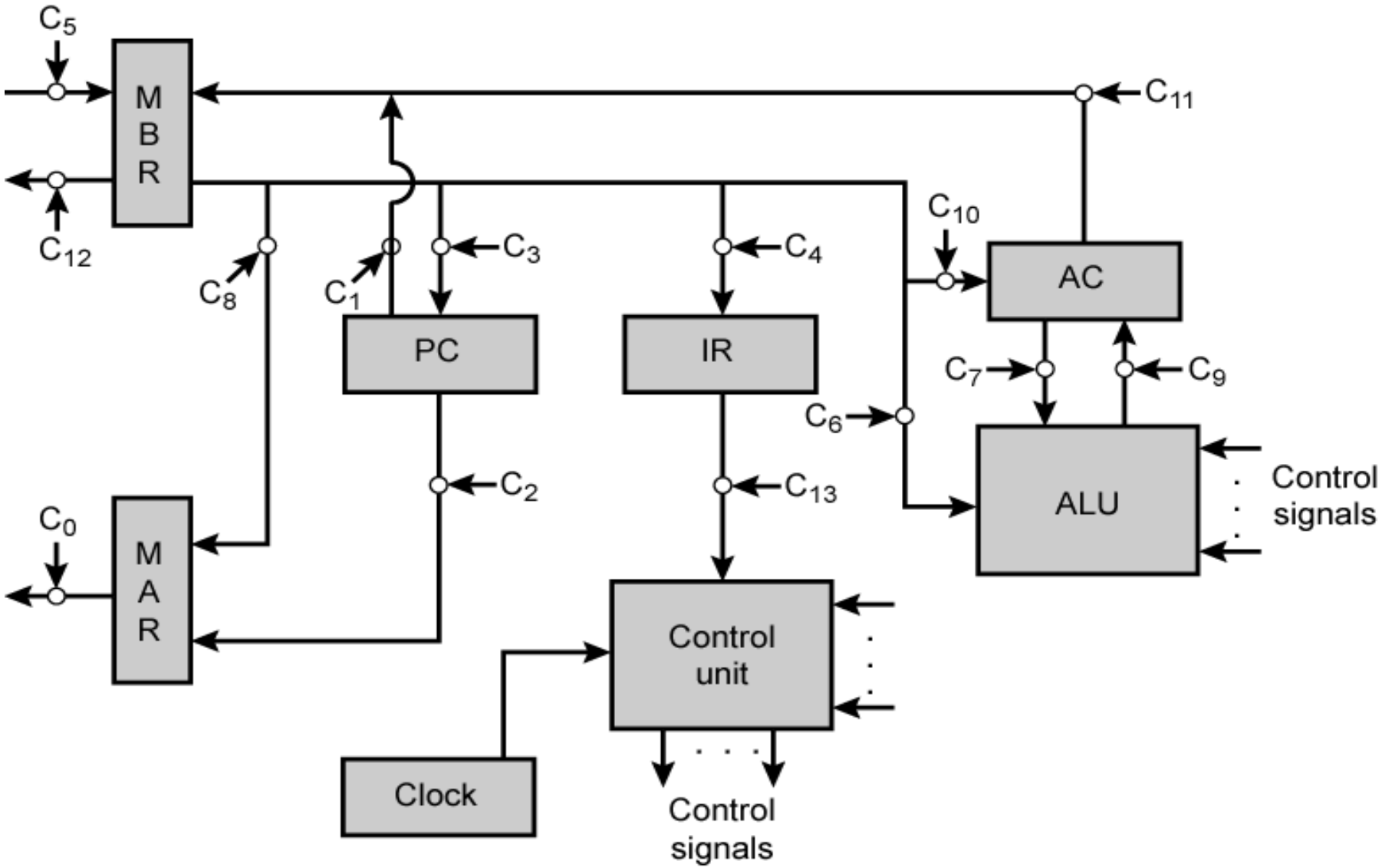


Hardwired Implementation

- Instruction register
  - Op-code causes 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
  - Different control signals at different times within instruction cycle
  - Must be long enough to allow signal propagation
  - Need a counter with different control signals for t1, t2 etc.

Data Paths and Control Signals



Group together all the microinstructions affecting a given register

Control		*Microcode*
MAR		
	t_1	MAR ← PC
c_{ADD}	t_4	MAR ← MDR
c_{AND}	t_4	MAR ← MDR
c_{LDA}	t_4	MAR ← MDR
c_{STA}	t_4	MAR ← MDR

Associate a control variable L(A,B) if a microinstruction assigns the register B to A denoted as A ← B

Derive control variables

$$L(\text{MAR}, \text{PC}) = t_1$$
$$L(\text{MAR}, \text{MDR}) = (c_{\text{ADD}} \vee c_{\text{AND}} \vee c_{\text{LDA}} \vee c_{\text{STA}}) \wedge t_4$$

Group together all the microinstructions affecting a given register

Control		*Microcode*
MDR		
	t_2	MDR $\leftarrow$ M
c_{ADD}	t_5	MDR $\leftarrow$ M
c_{AND}	t_5	MDR $\leftarrow$ M
c_{LDA}	t_5	MDR $\leftarrow$ M
c_{STA}	t_4	MDR $\leftarrow$ AC

Derive control variables

$$L(\mathrm{MDR}, \mathrm{M}) = t_2 \vee (c_{\mathrm{ADD}} \vee c_{\mathrm{AND}} \vee c_{\mathrm{LDA}}) \wedge t_5$$
$$L(\mathrm{MDR}, \mathrm{AC}) = c_{\mathrm{STA}} \wedge t_4$$

Control		*Microcode*
M		
c_{STA}	t_5	M ← MDR
PC		
	t_2	PC ← PC+1
c_{JZ}	t_4	if (AC = 0) PC ← MDR
OPC		
	t_3	OPC ← MDR

Group together all the microinstructions affecting a given register

Derive control variables

$$L(\text{M}, \text{MDR}) = c_{\text{STA}} \wedge t_5$$

$$L(\text{PC}, \text{PC+1}) = t_2$$

$$L(\text{PC}, \text{MDR}) = (AC = 0) \wedge c_{\text{JZ}} \wedge t_4$$

$$L(\text{OPC}, \text{MDR}) = t_3$$

Control	*Microcode*
OUTR	
$c_{\text{OUT}} \quad t_4$	$\text{OUTR} \leftarrow \text{AC}$
HLT	
$c_{\text{HLT}} \quad t_4$	$t_j \leftarrow 0$ for $1 \leq j \leq k$

Group together all the microinstructions affecting a given register

Derive control variables

$$L(\text{OUTR}, \text{AC}) = c_{\text{OUT}} \wedge t_4$$

$$L(t_j) = c_{\text{HLT}} \wedge t_4 \text{ for } 1 \leq j \leq 6$$

Control of Accumulator

Associate a control variable L(A,B $\odot$ C) if a microinstruction results in the combination of the contents of registers B and C with the operation $\odot$ and the assignment of the result to register A, denoted $A \leftarrow B \odot C$.

L(AC, AC+MDR) = $c_{ADD} \wedge t6$: The contents of AC are
added to those of MDR and copied into AC when $c_{ADD} \wedge t6 = 1$.

Control of Accumulator

Group together all the microinstructions affecting accumulator

Control		*Microcode*
AC		
c_{ADD}	t_6	AC $\leftarrow$ AC + MDR
c_{AND}	t_6	AC $\leftarrow$ AC AND MDR
c_{CLA}	t_4	AC $\leftarrow$ 0
c_{CIL}	t_4	AC $\leftarrow$ Shift(AC)
c_{LDA}	t_6	AC $\leftarrow$ MDR
c_{CMA}	t_4	AC $\leftarrow \neg$ AC
c_{IN}	t_4	AC $\leftarrow$ INR

Derive control variables

$$L(\text{AC}, \text{AC} + \text{MDR}) = c_{\text{ADD}} \wedge t_6$$
$$L(\text{AC}, \text{AC AND MDR}) = c_{\text{AND}} \wedge t_6$$
$$L(\text{AC}, 0) = c_{\text{CLA}} \wedge t_4$$
$$L(\text{AC}, \text{Shift(AC)}) = c_{\text{CIL}} \wedge t_4$$
$$L(\text{AC}, \text{MDR}) = c_{\text{LDA}} \wedge t_6$$
$$L(\text{AC}, \text{INR}) = c_{\text{IN}} \wedge t_4$$
$$L(\text{AC}, \neg\, \text{AC}) = c_{\text{CMA}} \wedge t_4$$

Problems With Hard Wired Designs

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

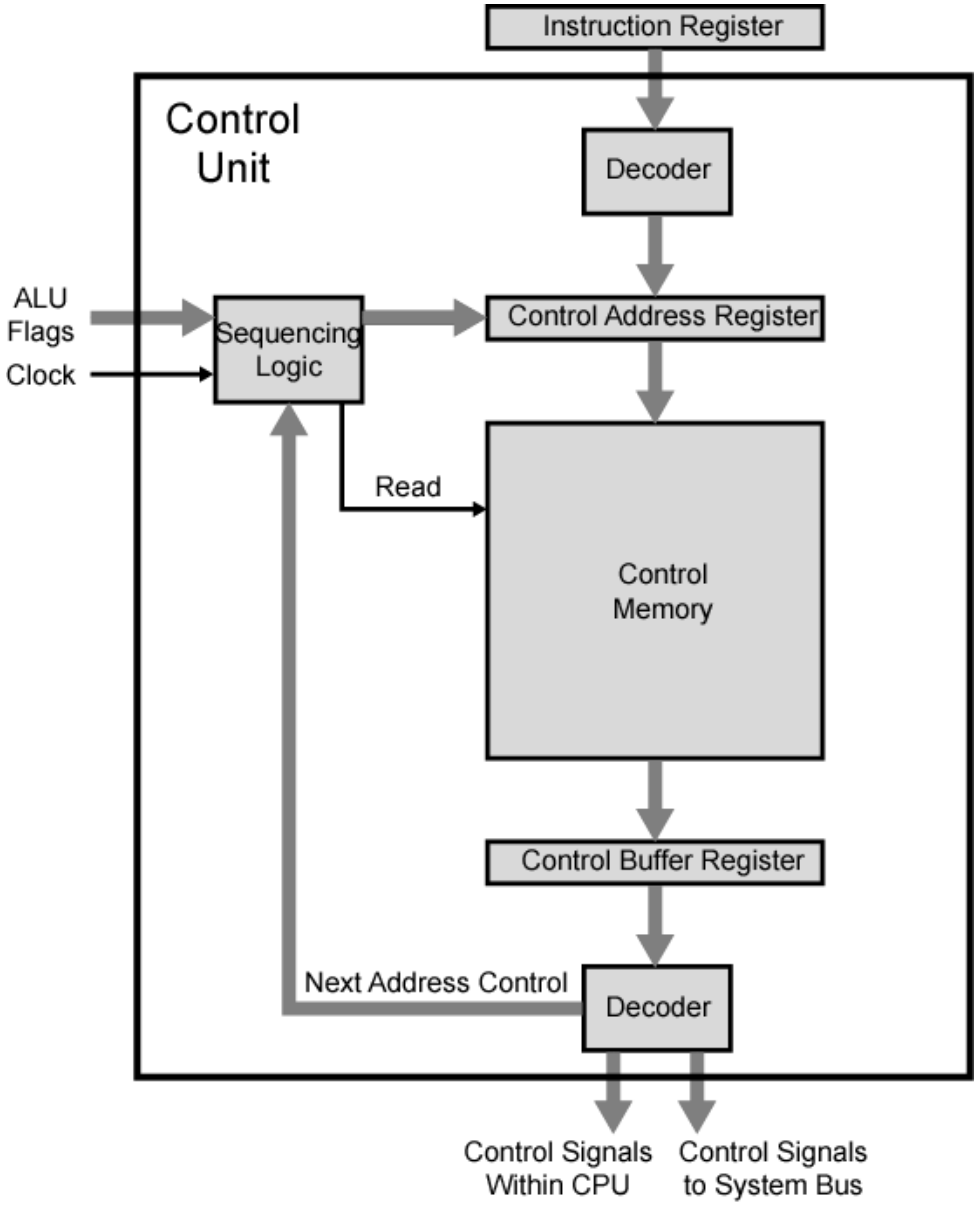
Micro-programmed Control

- Use sequences of instructions to control complex operations
- Called micro-programming or firmware
- A micro instruction is responsible for generating a control signal for desired control lines to implement a desired micro operation

- Advantages & Limitations:
- Simplifies design of control unit
  - Cheaper
  - Less error-prone
- Slower

Micro-programmed Control Unit

- Sequence logic 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 logic loads new address into control buffer register based on next address information from control buffer register and ALU flags



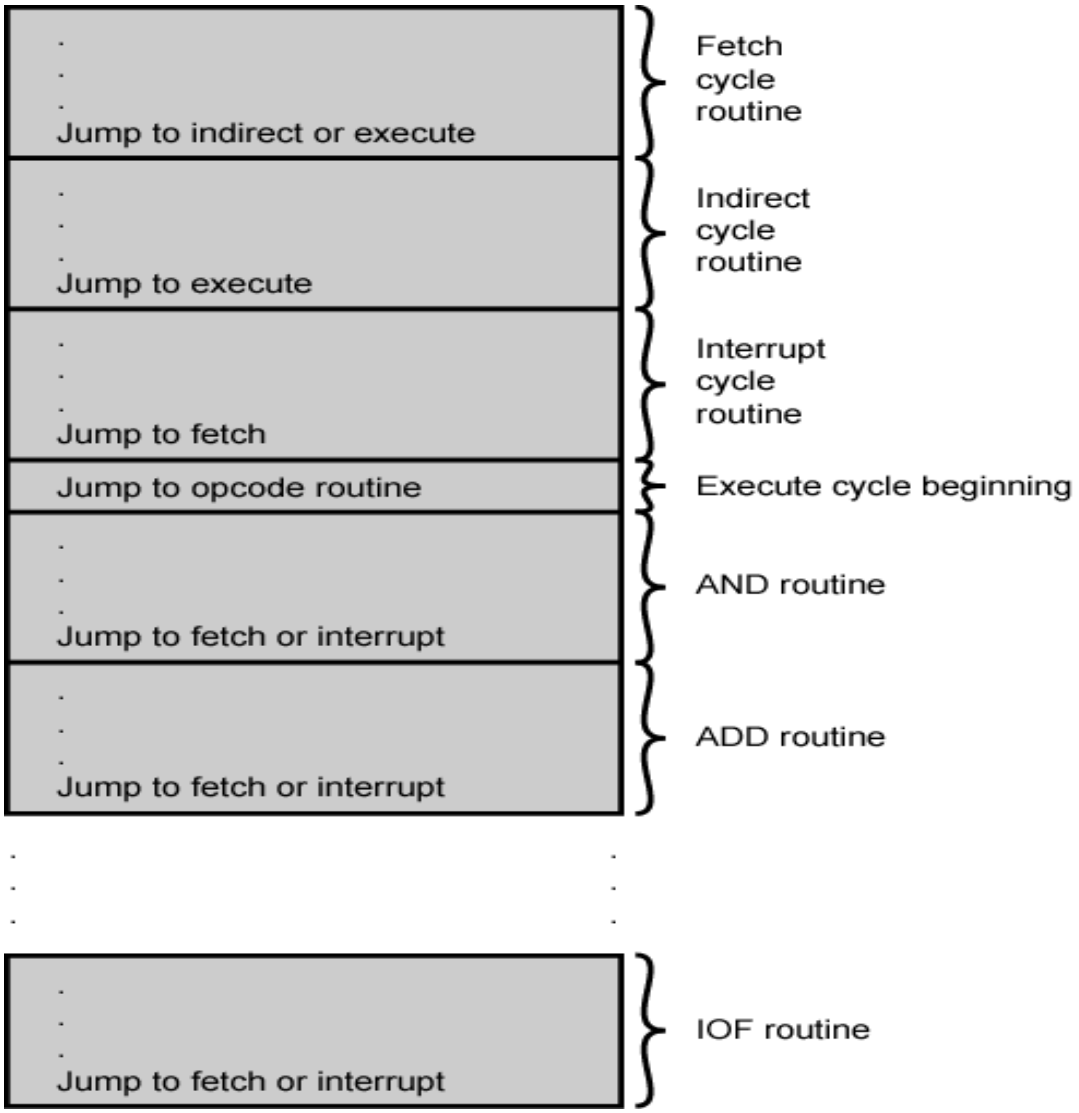
Implementation (1)

- 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

Implementation (2)

- Today's large microprocessor
  - Many instructions and associated register-level hardware
  - Many control points to be manipulated
- This results in control memory that
  - Contains a large number of words
    - corresponding to the number of instructions to be executed
  - Has a large word width
    - Due to the large number of control points to be manipulated

How is it organized in the control memory?



Next Address Decision

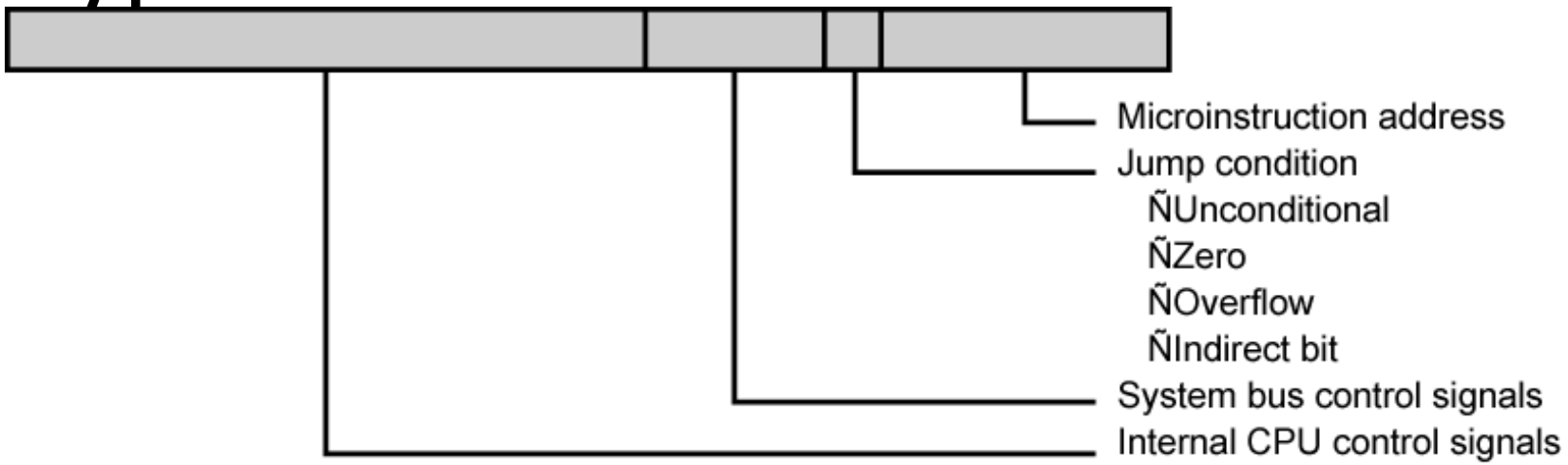
- Depending on ALU flags and control buffer register
  - Get next instruction
    - Add 1 to control address register
  - Jump to new routine based on jump microinstruction
    - Load address field of control buffer register into control address register
  - Jump to machine instruction routine
    - Load control address register based on opcode in IR

Micro-instruction Formats

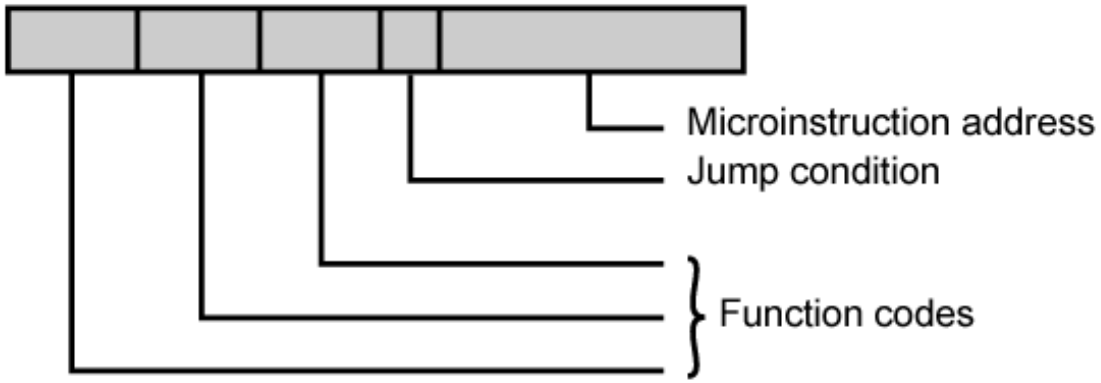
- *Horizontal* micro-instruction : Each bit represents a control signal directly controls a control line or system bus.
- micro-instruction specifies many different micro-operations to be performed in parallel

- *Vertical* micro-instruction:
- Each micro-instruction specifies one or few micro-operations to be performed
  - similar control signals are grouped into few micro instruction bits

Typical Microinstruction Formats



(a) Horizontal microinstruction



(b) Vertical microinstruction

Wilkes Control

- 1951
- Matrix partially filled with diodes
- During cycle, one row activated
  - Generates signals where diode present
  - First part of row generates control
  - Second generates address for next cycle

Wilkes's Microprogrammed Control Unit

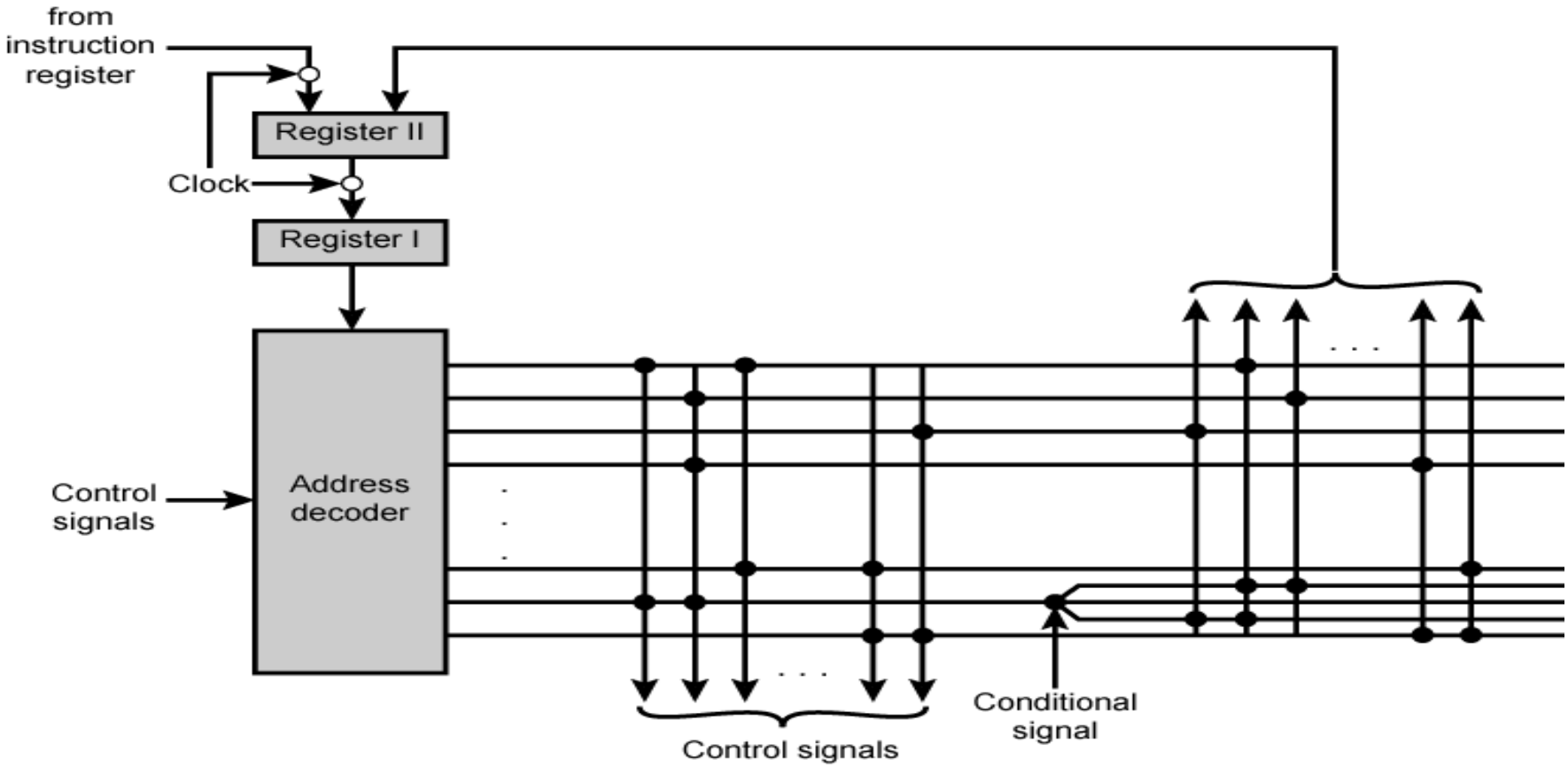
Matrix partially filled with diodes

During cycle, one row activated

Generates signals where diode present

First part of row generates control

Second generates address for next cycle



Improvements over Wilkes

- Wilkes had each bit directly produced a control signal or directly produced one bit of next address
- More complex address sequencing schemes using fewer microinstruction bits, are possible
- Control word bits can be saved by encoding and subsequently decoding control information

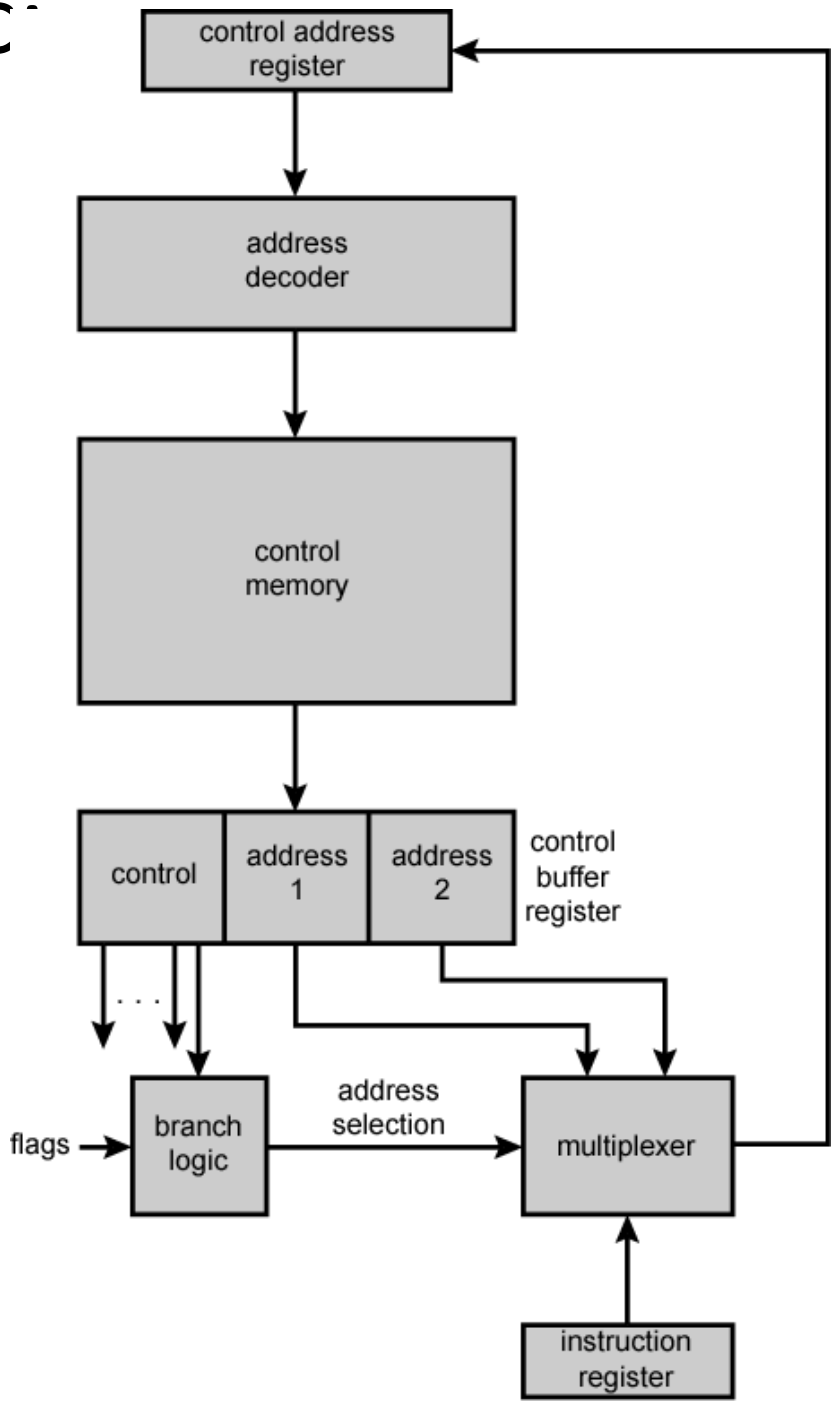
Design Considerations Micro-programed Control Unit

- Size of microinstructions
- Address generation
  - Determined by instruction register
    - Once per cycle, after instruction is fetched
  - Next sequential address
    - Common in most designed
  - Branches
    - Both conditional and unconditional

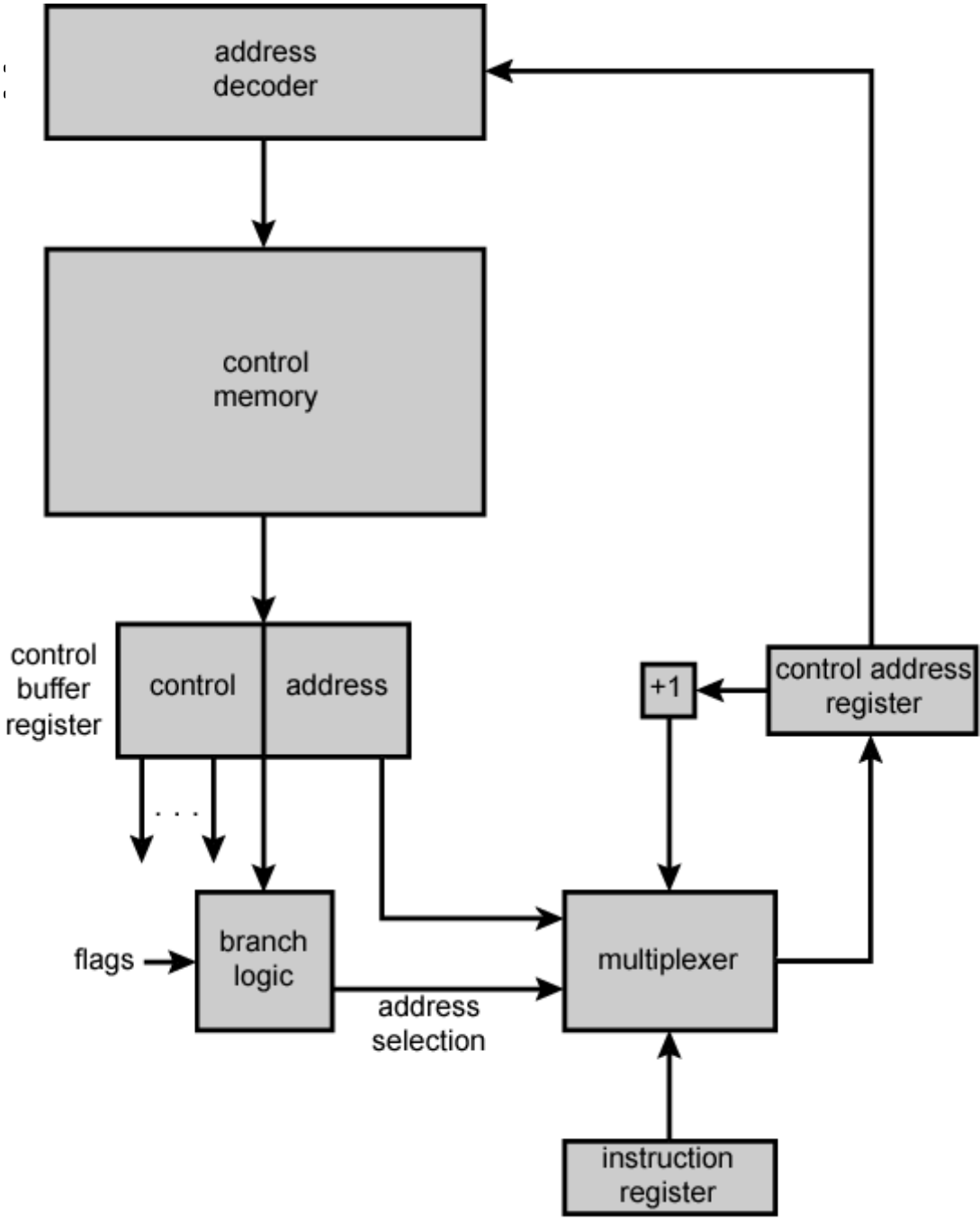
Sequencing Techniques

- Based on current microinstruction, condition flags & contents of IR, the control memory address must be generated
- Two Broad Categories:
- Explicit
  - Three different techniques:
    - Based on format of address information
      - Two address fields
      - Single address field
      - Variable format
- Implicit:

Branch Control Logic
Two Address Fields

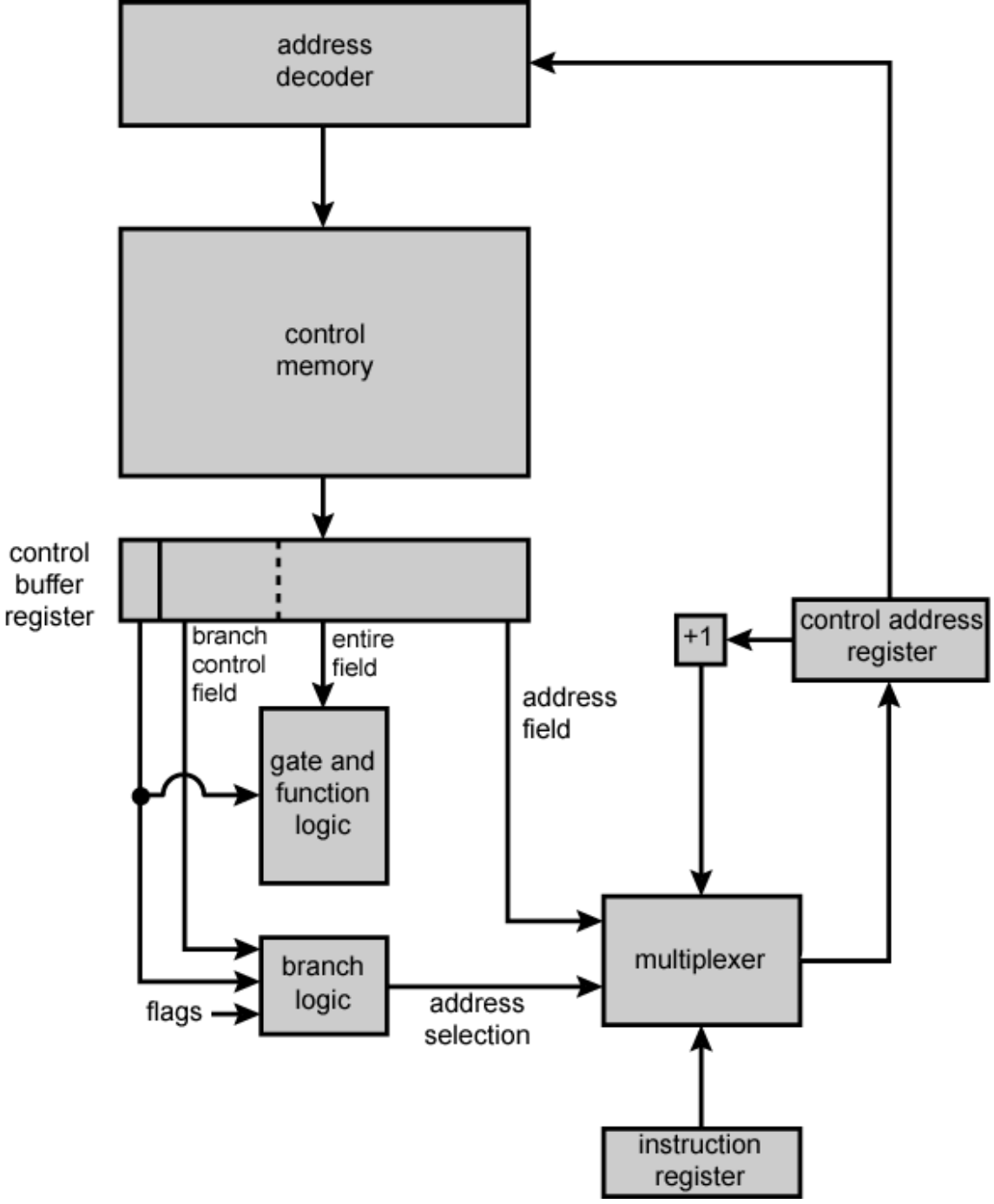


Branch Control Logic: Single Address Field



Branch Control Logic Variable Format

A bit designates whether the remaining bits in a field are control signals (one format) or (second format) represent some bits to drive branch logic and remaining bits represent address bits



Address Generation (Implicit)

- Common Implicit address Generation
  - Mapping
    - Opcode of machine instruction is mapped into a microinstruction address (occurs once for each instruction)
  - Combining:
    - Combining two portions of addresses to form the complete address.
  - Residual Control
    - Two parts of the address are distinguished (MSB (8-bit) part is fixed and remaining (5-bit) LSB are set to specific address

Micro-instruction Execution

- To generate control signal(s) to drive CPU (internal signal) or system bus (external signal)
  - Encoding of micro instruction

How to Encode?

- Let K different internal and external control signals to be driven
- Wilkes's:
  - K bits dedicated
  - 2^K control signals during any instruction cycle
- Not all used
  - Two sources may be connected by respective control signals to same destination
  - Register cannot be source and destination
  - Only one pattern presented to ALU at a time
  - Only one pattern presented to external control bus at a time
- Require $Q < 2^K$ which can be encoded with $q=\log_2 Q < K$ bits

- Let us look two different types
  - Unencoded Microinstruction
  - Highly encoded Microinstruction

Unencoded Microinstruction

- One bit for one signal
  - Micro instruction bit length would be high
  - Each control signal can be controlled individually
  - Concurrency can be exploited
  - As there is one-to-one correspondence no circuit is need for decoding
  - But, Difficult to program

Highly encoded micro instruction

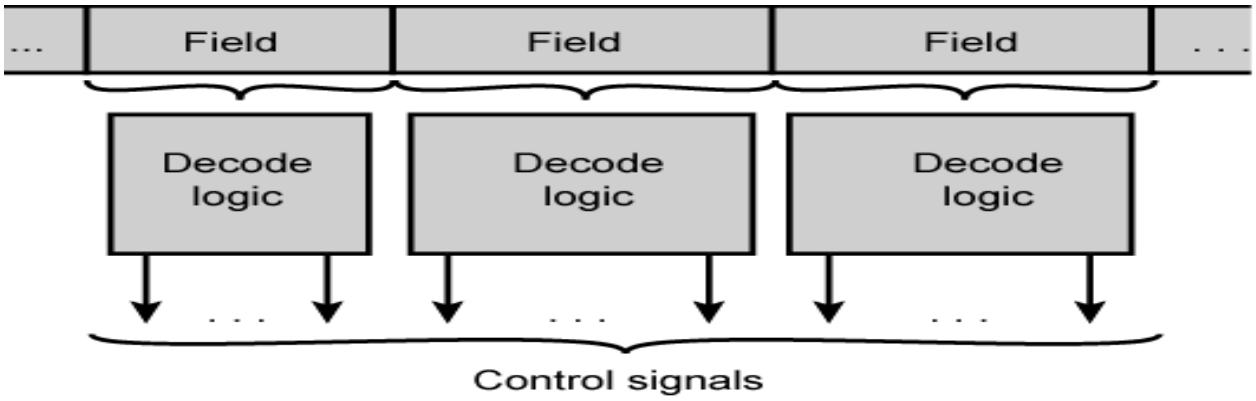
- A small number of encoded instruction
- The encoding simplifies programming
- Complex control circuitry for decoding is needed
- Difficult to exploit the concurrency

- Usually, Microprogrammed control unit is in between (not un-encoded, nor highly encoded)

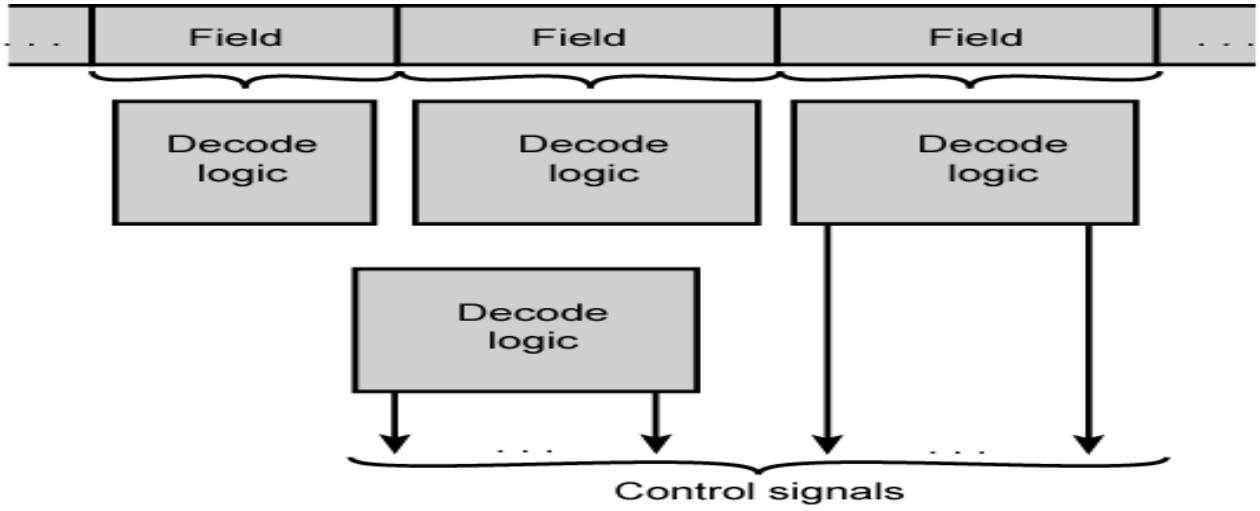
Specific Encoding Techniques

- Microinstruction organized as set of fields
- Each field contains code
  - Activates one or more control signals
- Organize format into independent fields
  - Field depicts set of actions (pattern of control signals)
  - Actions from different fields can occur simultaneously
- Alternative actions that can be specified by a field are mutually exclusive
  - Only one action specified for field could occur at a time

Microinstruction Encoding



(a) Direct encoding



(b) Indirect encoding

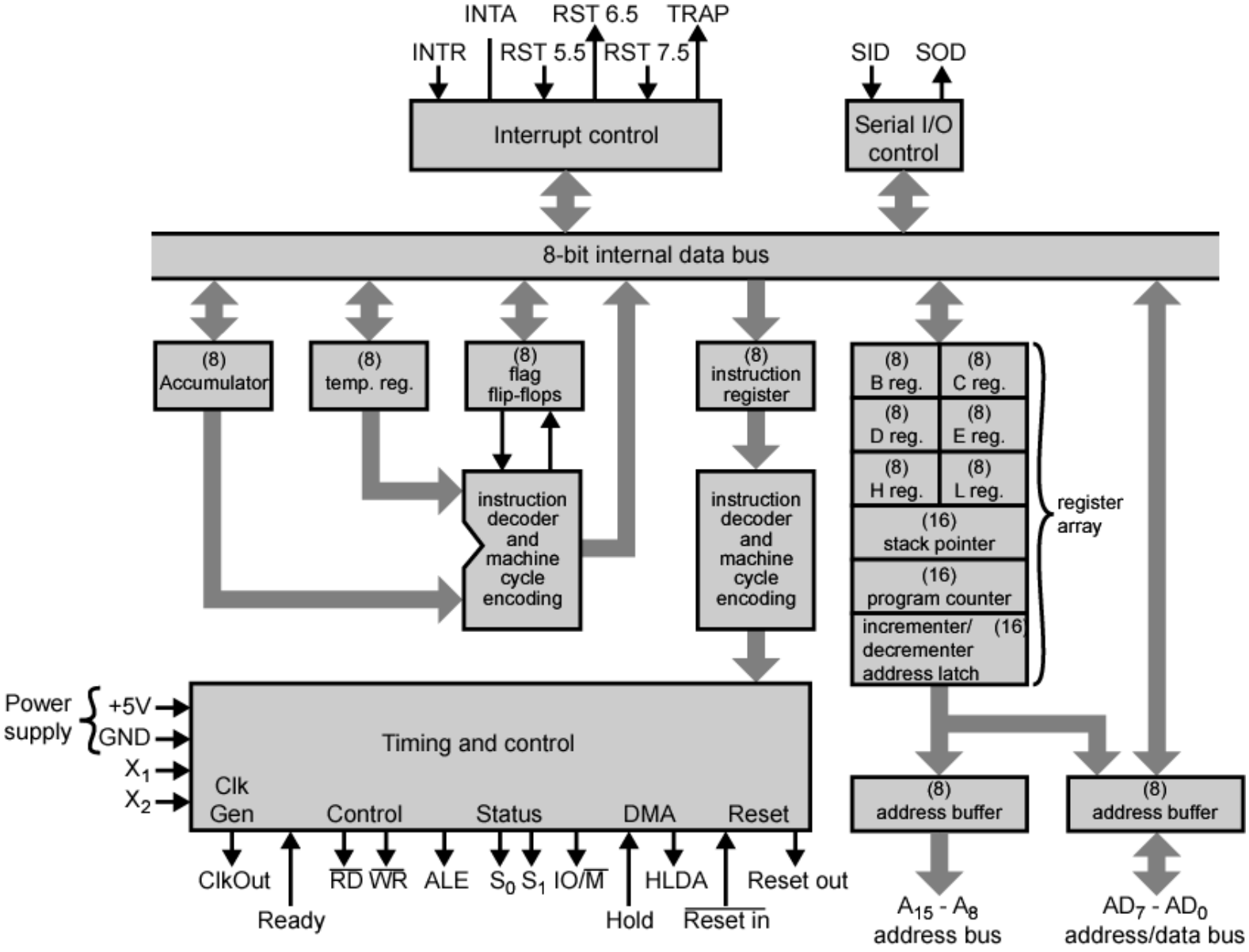
Control unit

- Timing and control
  - Inputs:
    - Uses clock
    - Accepts the current instruction and some external control signals
  - Outputs
    - Control signals to other components of CPU and external system bus

- Each instruction is divided into 1 to 5 machine cycle and each MC divided into 3 to 5 states.
- Each state lasts for a cycle
- During a state CPU performs one (or a simultaneous set of) micro op

- Important task of CU
- Machine Start-up
  - To initialize the registers
  - Reset

Intel 8085 CPU Block Diagram



Intel 8085 OUT Instruction Timing Diagram