

# **(Advanced) Computer Architecture**

**Prof. Dr. Hasan Hüseyin BALIK**  
**(11<sup>th</sup> Week)**

# Outline

## 4. The central processing unit

- Processor Structure and Function
- Reduced Instruction Set Computers (RISCs)
- Instruction-Level Parallelism and Superscalar Processors
- Control Unit Operation and Microprogrammed Control



+

## 4.4 Control Unit Operation and Microprogrammed Control

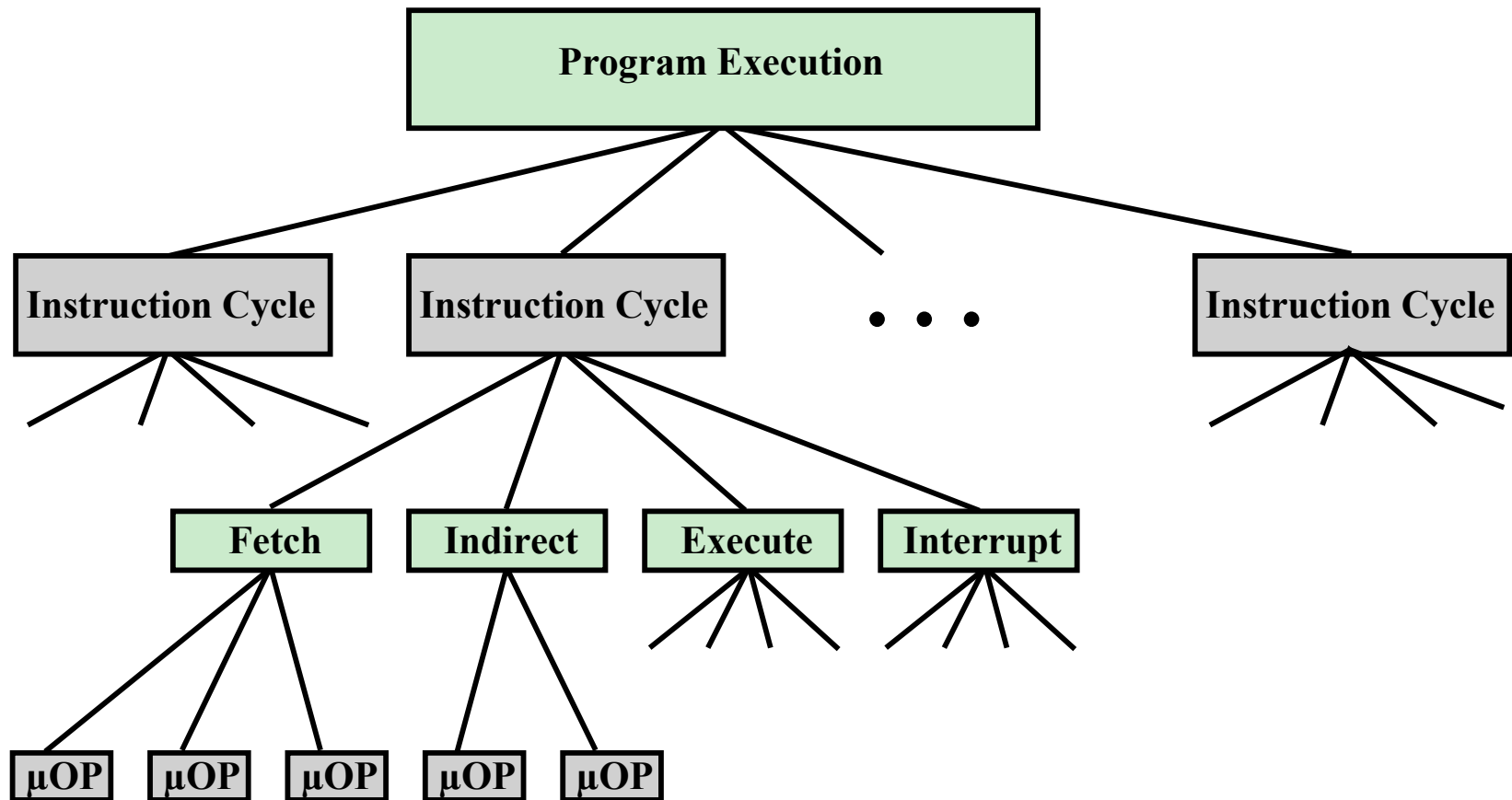
## 4.4 Outline

- Micro-operations
- Control of the Processor
- Hardwired Implementation
- Microprogrammed Control

# Micro-Operations

- The functional, or atomic, operations of a processor
- Series of steps, each of which involves the processor registers
- *Micro* refers to the fact that each step is very simple and accomplishes very little
- The execution of a program consists of the sequential execution of instructions
  - Each instruction is executed during an instruction cycle made up of shorter subcycles (fetch, indirect, execute, interrupt)
  - The execution of each subcycle involves one or more shorter operations (micro-operations)

# Figure: Constituent Elements of a Program Execution



# The Fetch Cycle

- Occurs at the beginning of each instruction cycle and causes an instruction to be fetched from memory
- Four registers are involved:
  - Memory Address Register (MAR)
    - Connected to address bus
    - Specifies address for read or write operation
  - Memory Buffer Register (MBR)
    - Connected to data bus
    - Holds data to write or last data read
  - Program Counter (PC)
    - Holds address of next instruction to be fetched
  - Instruction Register (IR)
    - Holds last instruction fetched

# Figure: Sequence of Events, Fetch Cycle

tMAR	
MBR	
PC	0000000001100100
IR	
AC	

(a) Beginning (before  $t_1$ )

MAR	0000000001100100
MBR	
PC	0000000001100100
IR	
AC	

(b) After first step

MAR	0000000001100100
MBR	0001000000100000
PC	0000000001100101
IR	
AC	

(c) After second step

MAR	0000000001100100
MBR	0001000000100000
PC	0000000001100101
IR	0001000000100000
AC	

(d) After third Step

# Indirect Cycle

- Once an instruction is fetched, the next step is to fetch source operands
- Assuming a one-address instruction format, with direct and indirect addressing allowed:
  - If the instruction specifies an indirect address, then an indirect cycle must precede the execute cycle
  - The address field of the instruction is transferred to the MAR
  - This is then used to fetch the address of the operand
  - Finally, the address field of the IR is updated from the MBR, so that it now contains a direct rather than an indirect address
  - The IR is now in the same state as if indirect addressing had not been used, and it is ready for the execute cycle

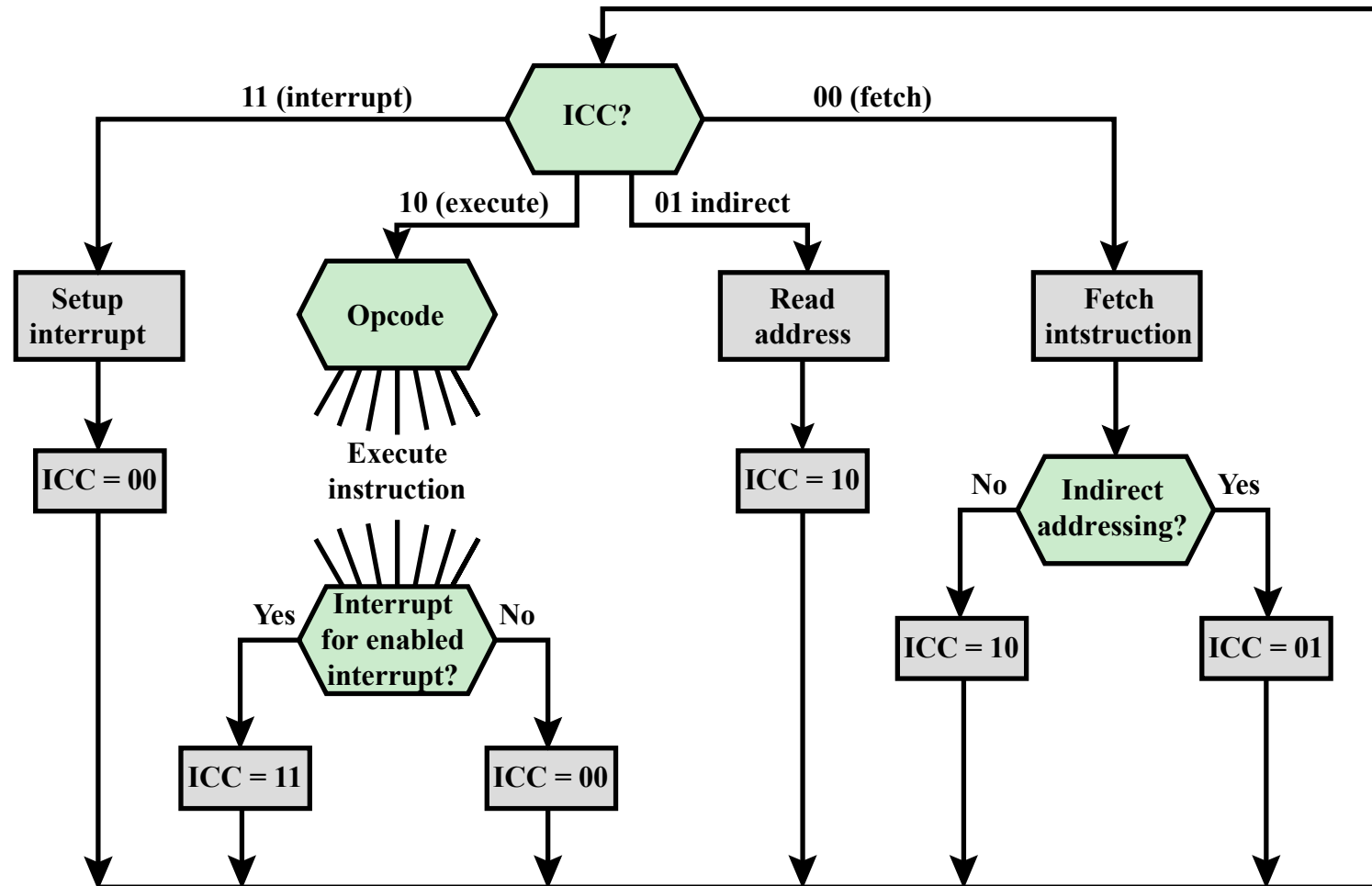
# Interrupt Cycle

- At the completion of the execute cycle, a test is made to determine whether any enabled interrupts have occurred, and if so, the interrupt cycle occurs
- The nature of this cycle varies greatly from one machine to another
- In a simple sequence of events:
  - In the first step the contents of the PC are transferred to the MBR so that they can be saved for return from the interrupt
  - Then the MAR is loaded with the address at which the contents of the PC are to be saved, and the PC is loaded with the address of the start of the interrupt-processing routine
    - These two actions may each be a single micro-operation
    - Because most processors provide multiple types and/or levels of interrupts, it may take one or more additional micro-operations to obtain the Save\_Address and the Routine\_Address before they can be transferred to the MAR and PC respectively
  - Once this is done, the final step is to store the MBR, which contains the old value of the PC, into memory
  - The processor is now ready to begin the next instruction cycle

# Execute Cycle

- Because of the variety of opcodes, there are a number of different sequences of micro-operations that can occur
- Instruction decoding
  - The control unit examines the opcode and generates a sequence of micro-operations based on the value of the opcode
- A simplified add instruction:
  - ADD R1, X (which adds the contents of the location X to register R1)
    - In the first step the address portion of the IR is loaded into the MAR
    - Then the referenced memory location is read
    - Finally the contents of R1 and MBR are added by the ALU
    - Additional micro-operations may be required to extract the register reference from the IR and perhaps to stage the ALU inputs or outputs in some intermediate registers

# Figure: Flowchart for Instruction Cycle

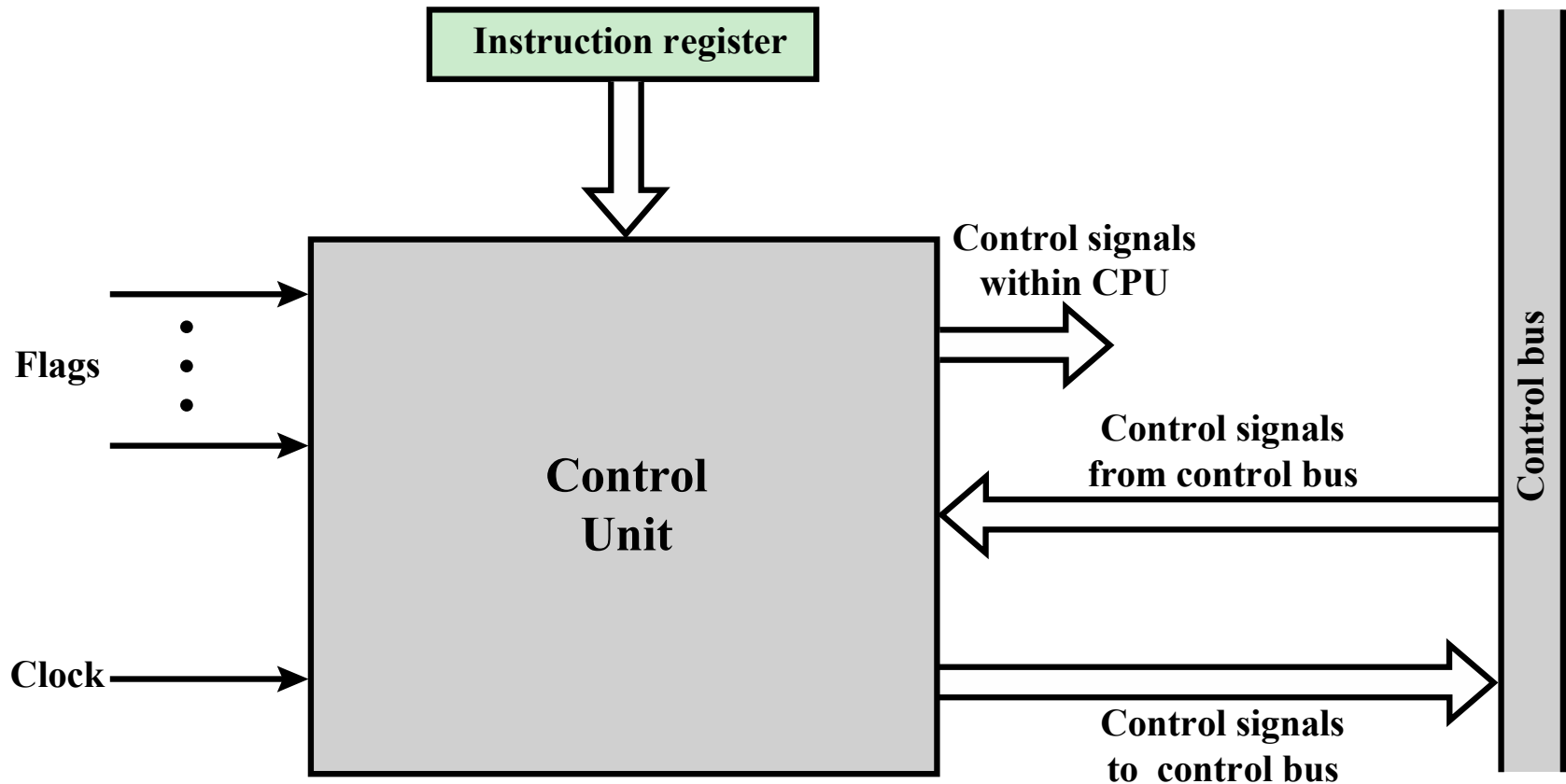


# Control Unit

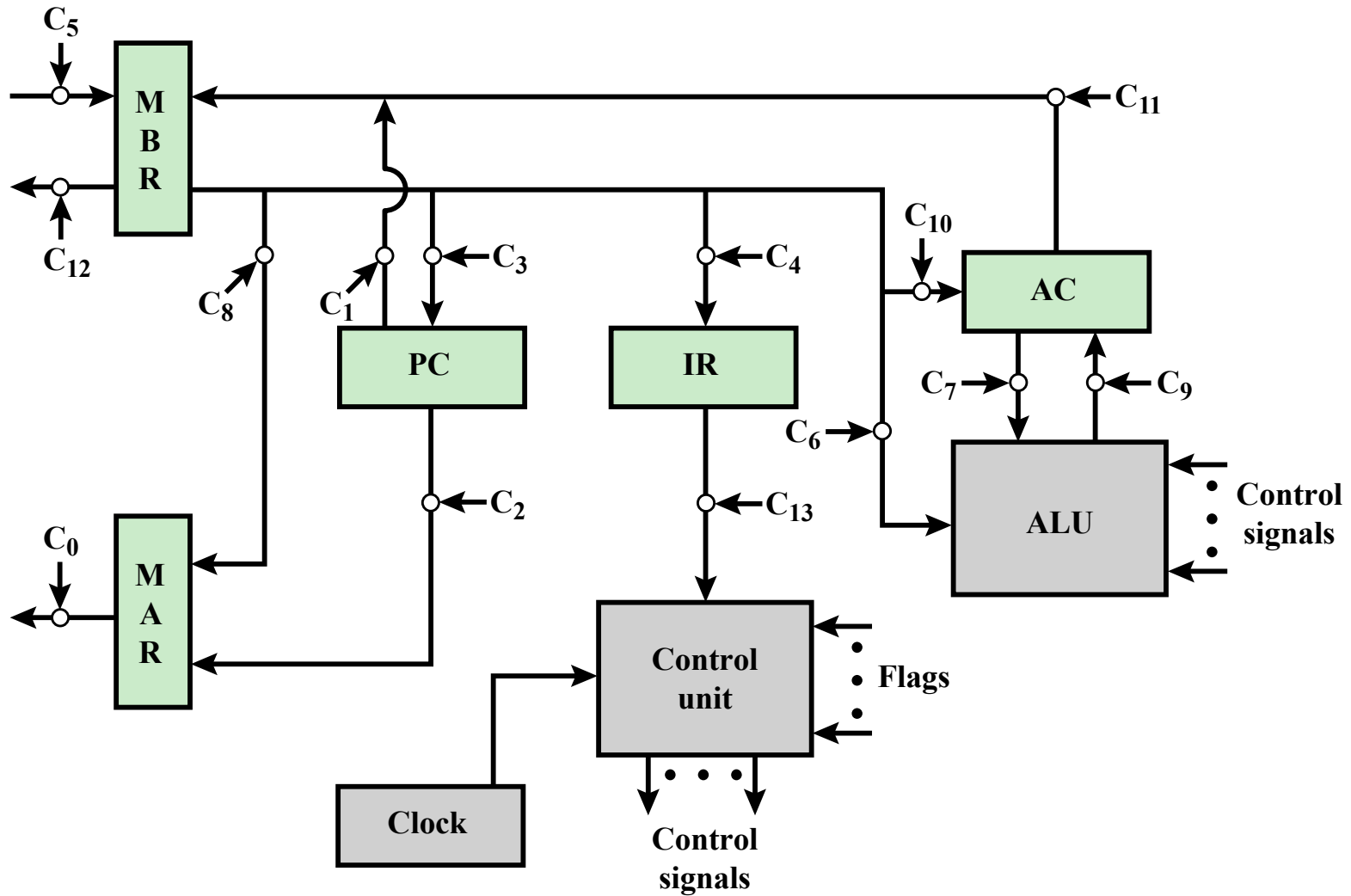
## Functional Requirements

- By reducing the operation of the processor to its most fundamental level we are able to define exactly what it is that the control unit must cause to happen
- Three step process to lead to a characterization of the control unit:
  - Define basic elements of processor
  - Describe micro-operations processor performs
  - Determine the functions that the control unit must perform to cause the micro-operations to be performed
- The control unit performs two basic tasks:
  - Sequencing
  - Execution

# Figure: Block Diagram of the Control Unit



# Figure: Data Paths and Control Signals



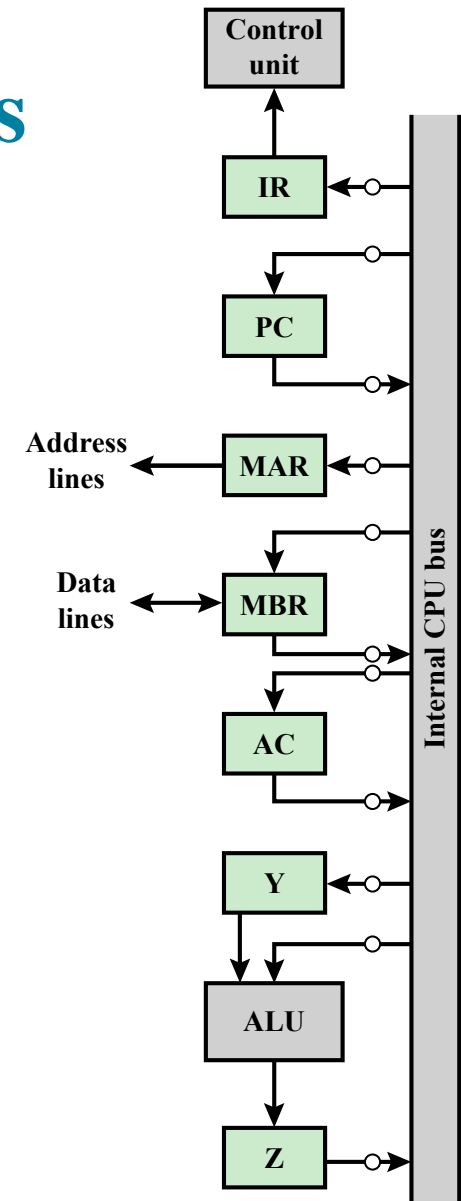
# Table: Micro-operations and Control Signals

	Micro-operations	Active Control Signals
<b>Fetch:</b>	$t_1: \text{MAR} \leftarrow (\text{PC})$	$C_2$
	$t_2: \text{MBR} \leftarrow \text{Memory}$ $\text{PC} \leftarrow (\text{PC}) + 1$	$C_5, C_R$
	$t_3: \text{IR} \leftarrow (\text{MBR})$	$C_4$
<b>Indirect:</b>	$t_1: \text{MAR} \leftarrow (\text{IR}(\text{Address}))$	$C_8$
	$t_2: \text{MBR} \leftarrow \text{Memory}$	$C_5, C_R$
	$t_3: \text{IR}(\text{Address}) \leftarrow (\text{MBR}(\text{Address}))$	$C_4$
<b>Interrupt:</b>	$t_1: \text{MBR} \leftarrow (\text{PC})$	$C_1$
	$t_2: \text{MAR} \leftarrow \text{Save-address}$ $\text{PC} \leftarrow \text{Routine-address}$	
	$t_3: \text{Memory} \leftarrow (\text{MBR})$	$C_{12}, C_W$

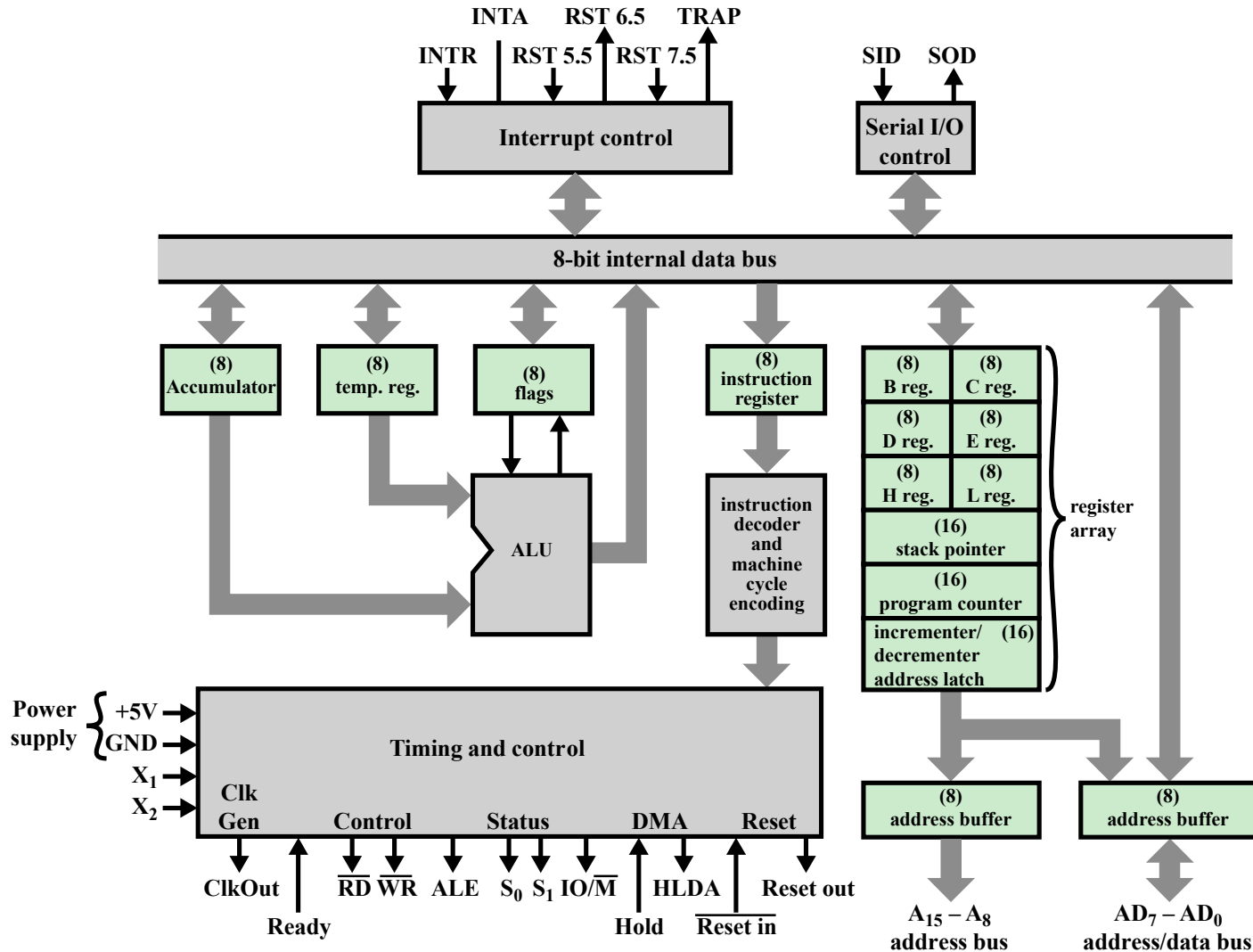
$C_R$  = Read control signal to system bus.

$C_W$  = Write control signal to system bus.

# Figure: CPU with Internal Bus



# Figure: Intel 8085 CPU Block Diagram



### *Address and Data Signals*

#### **High Address (A15–A8)**

The high-order 8 bits of a 16-bit address.

#### **Address/Data (AD7–AD0)**

The lower-order 8 bits of a 16-bit address or 8 bits of data. This multiplexing saves on pins.

#### **Serial Input Data (SID)**

A single-bit input to accommodate devices that transmit serially (one bit at a time).

#### **Serial Output Data (SOD)**

A single-bit output to accommodate devices that receive serially.

### *Timing and Control Signals*

#### **CLK (OUT)**

The system clock. The CLK signal goes to peripheral chips and synchronizes their timing.

#### **X1, X2**

These signals come from an external crystal or other device to drive the internal clock generator.

#### **Address Latch Enabled (ALE)**

Occurs during the first clock state of a machine cycle and causes peripheral chips to store the address lines.

This allows the address module (e.g., memory, I/O) to recognize that it is being addressed.

#### **Status (S0, S1)**

Control signals used to indicate whether a read or write operation is taking place.

#### **IO/M**

Used to enable either I/O or memory modules for read and write operations.

#### **Read Control (RD)**

Indicates that the selected memory or I/O module is to be read and that the data bus is available for data transfer.

#### **Write Control (WR)**

Indicates that data on the data bus is to be written into the selected memory or I/O location.

**Table:**  
**Intel**  
**8085**  
**External**  
**Signals**  
(page 1 of 2)

### **Hold**

Requests the CPU to relinquish control and use of the external system bus. The CPU will complete execution of the instruction presently in the IR and then enter a hold state, during which no signals are inserted by the CPU to the control, address, or data buses. During the hold state, the bus may be used for DMA operations.

### **Hold Acknowledge (HOLDA)**

This control unit output signal acknowledges the HOLD signal and indicates that the bus is now available.

### **READY**

Used to synchronize the CPU with slower memory or I/O devices. When an addressed device asserts READY, the CPU may proceed with an input (DBIN) or output (WR) operation. Otherwise, the CPU enters a wait state until the device is ready.

### *Interrupt-Related Signals*

### **TRAP**

Restart Interrupts (RST 7.5, 6.5, 5.5)

### **Interrupt Request (INTR)**

These five lines are used by an external device to interrupt the CPU. The CPU will not honor the request if it is in the hold state or if the interrupt is disabled. An interrupt is honored only at the completion of an instruction. The interrupts are in descending order of priority.

### **Interrupt Acknowledge**

Acknowledges an interrupt.

### *CPU Initialization*

### **RESET IN**

Causes the contents of the PC to be set to zero. The CPU resumes execution at location zero.

### **RESET OUT**

Acknowledges that the CPU has been reset. The signal can be used to reset the rest of the system.

### *Voltage and Ground*

### **VCC**

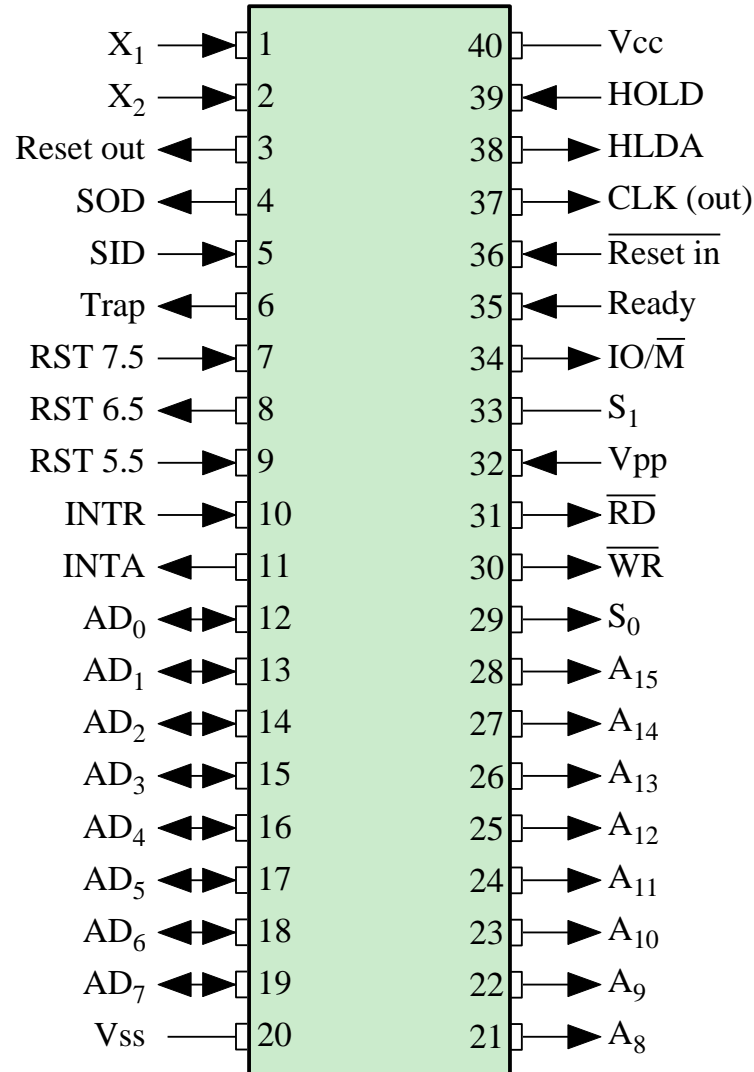
+5-volt power supply

### **VSS**

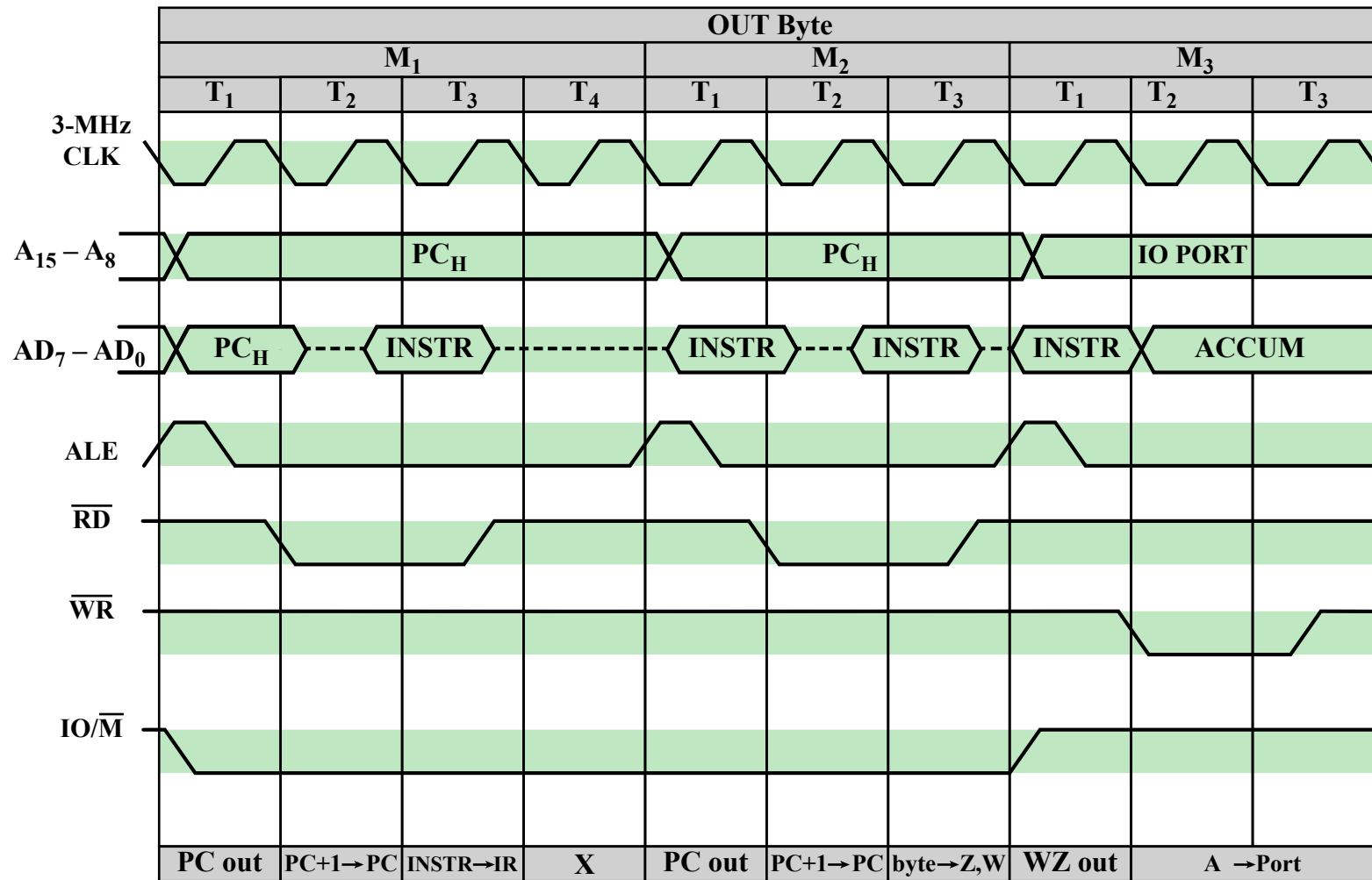
Electrical ground

**Table:  
Intel  
8085  
External  
Signals  
(page 2 of 2)**

# Figure: Intel 8085 Pin Configuration



# Figure: Timing Diagram for Intel 8085 OUT Instruction

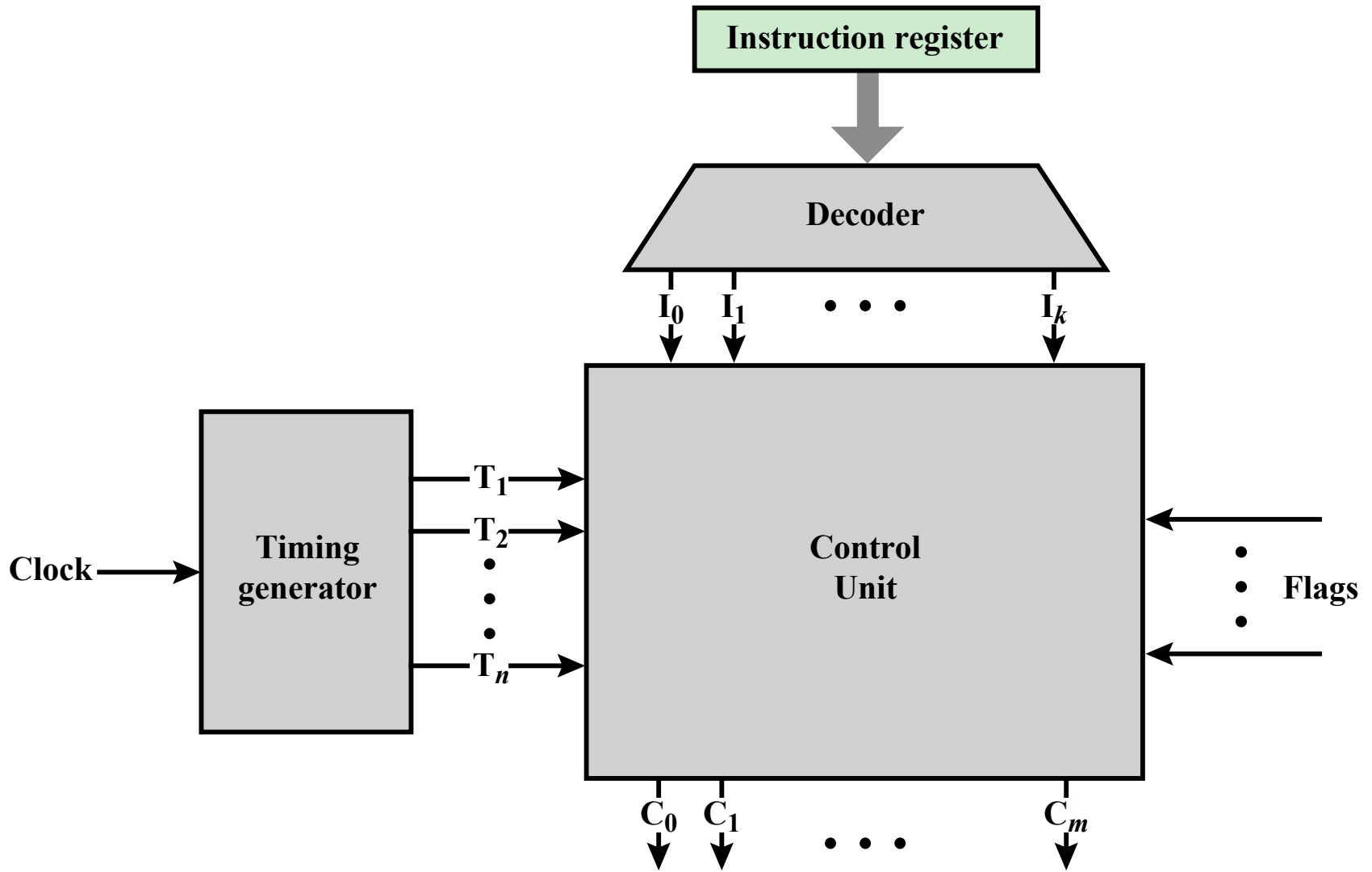


← Instruction fetch → Memory read → Output write →

# Control Unit Implementation

- A wide variety of techniques have been used for control unit implementation
- Most of these fall into two categories:
  - 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
  - Microprogrammed implementation

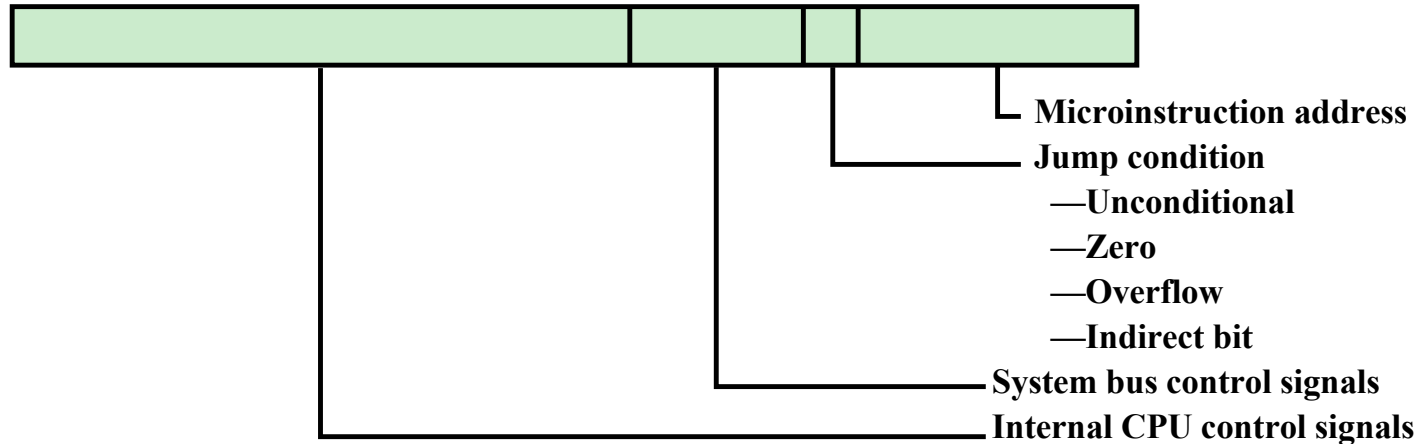
# Figure: Control Unit with Decoded Inputs



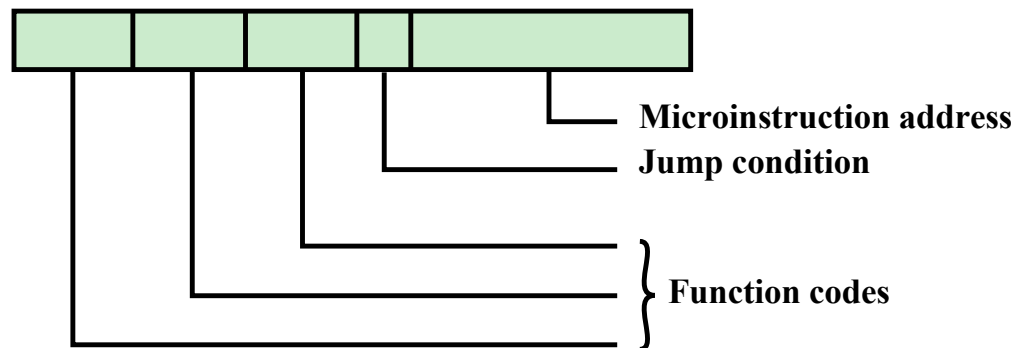
# Microprogrammed Control

- The term *microprogrammed* was first coined by M.V. Wilkes in the early 1950's
- Wilkes proposed an approach to control unit design that was organized and systematic and avoided the complexities of a hardwired implementation
- The idea intrigued many researchers but appeared unworkable because it would require a fast, relatively inexpensive control memory
- In April of 1964 IBM's System/360 was announced and all but the largest models were microprogrammed
- Microprogramming became a popular technique for implementing the control unit of CISC processors
- In recent years, microprogramming has become less used but remains a tool available to computer designers

# Figure: Typical Microinstruction Formats

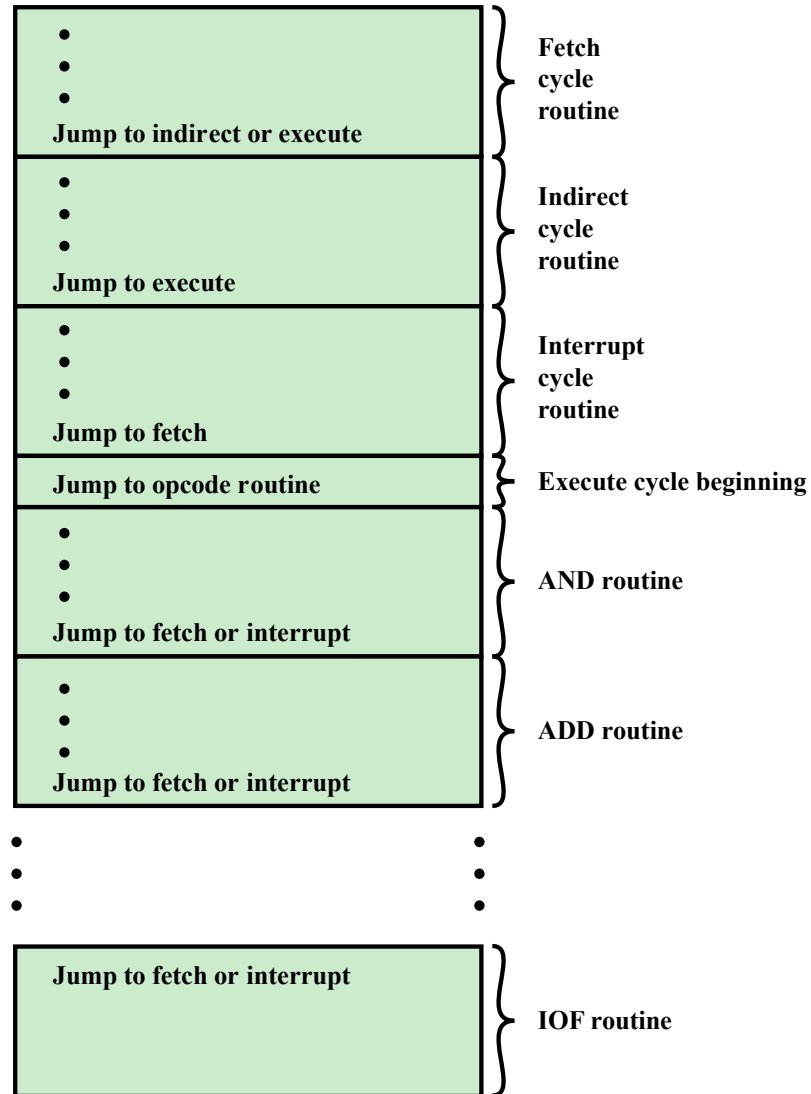


(a) Horizontal microinstruction

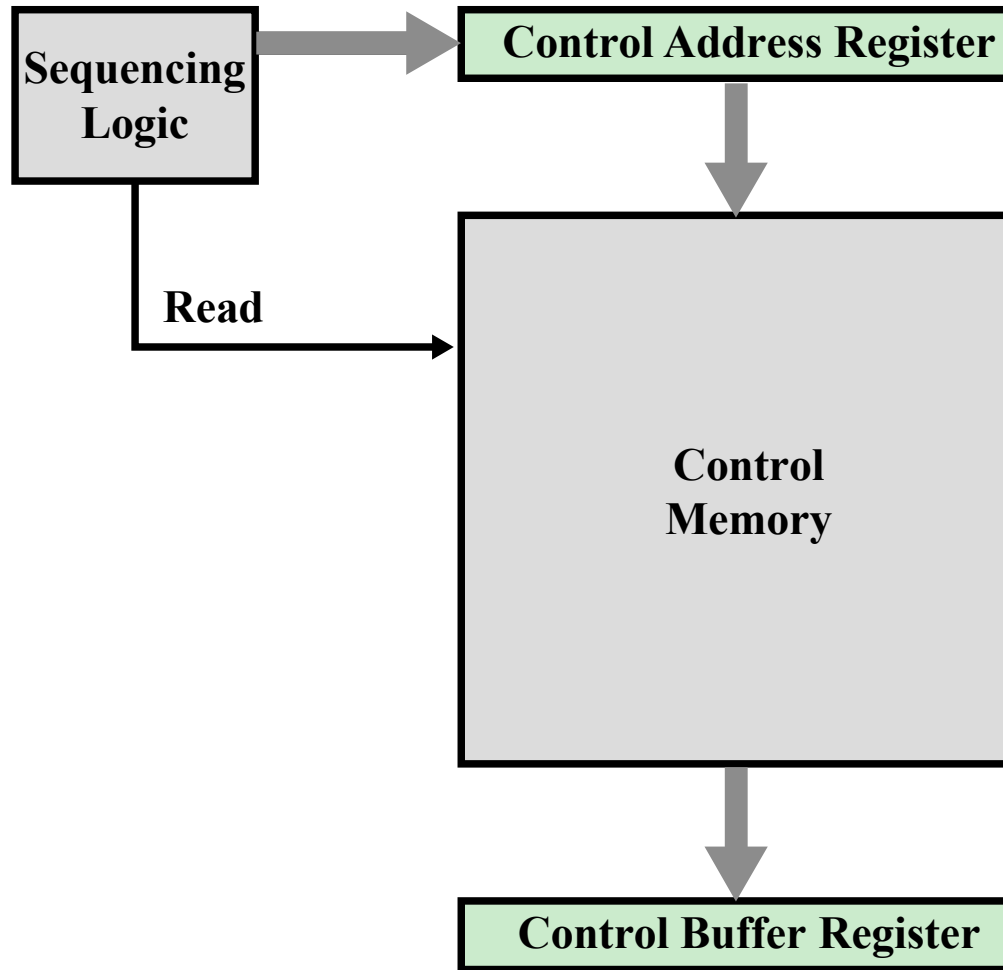


(b) Vertical microinstruction

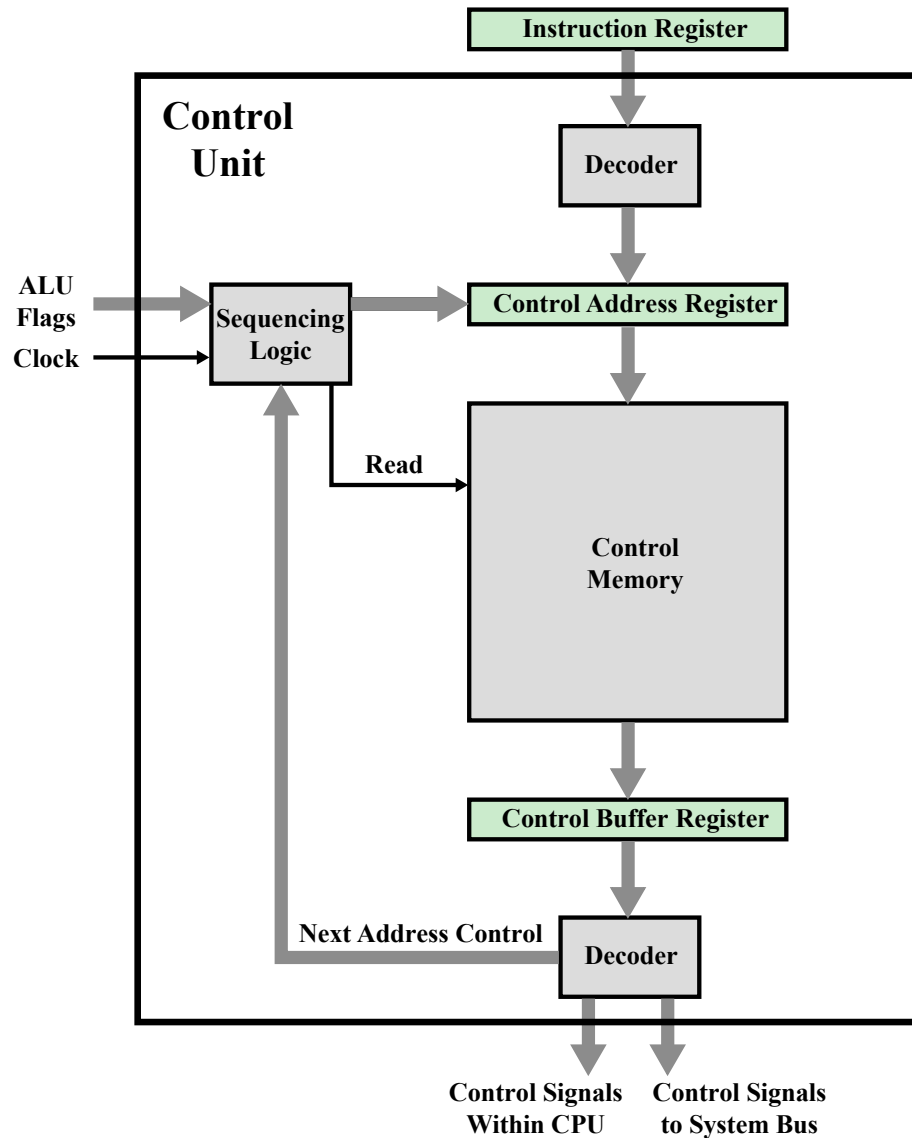
# Figure: Organization of Control Memory



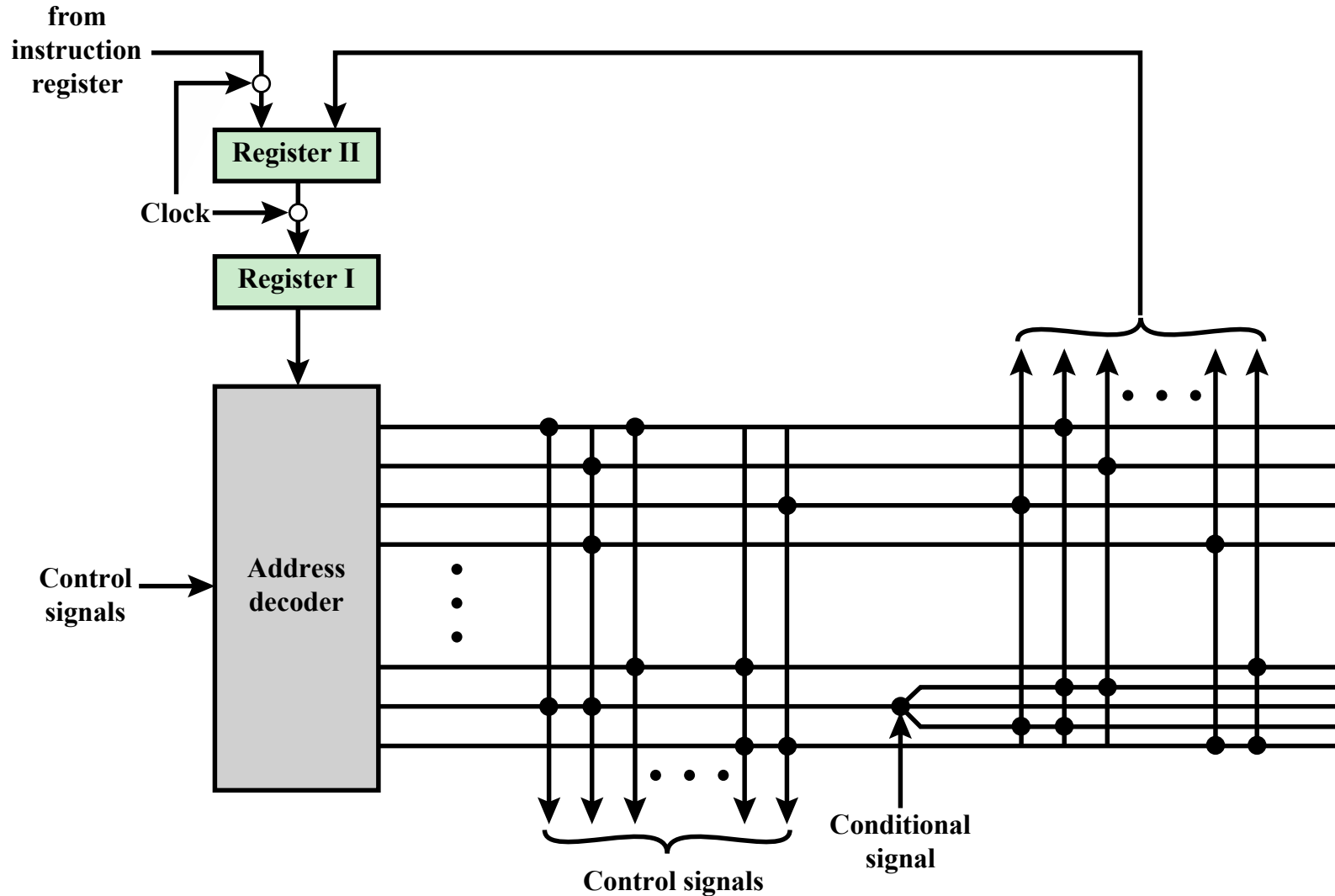
# Figure: Control Unit Microarchitecture



## Figure: Functioning of Microprogrammed Control Unit



# Figure: Wilkes's Microprogrammed Control Unit



Notations:  $A, B, C, \dots$  stand for the various registers in the arithmetical and control register units.  $C$  to  $D$  indicates that the switching circuits connect the output of register  $C$  to the input register  $D$ ;  $(D + A)$  to  $C$  indicates that the output register of  $A$  is connected to the one input of the adding unit (the output of  $D$  is permanently connected to the other input), and the output of the adder to register  $C$ . A numerical symbol  $n$  in quotes (e.g., “ $n$ ”) stands for the source whose output is the number  $n$  in units of the least significant digit.

		Arithmetical Unit	Control Register Unit	Conditional Flip-Flop		Next Microinstruction	
				Set	Use	0	1
	0		$F$ to $G$ and $E$			1	
	1		$(G$ to “1”) to $F$			2	
	2		Store to $G$			3	
	3		$G$ to $E$			4	
	4		$E$ to decoder			–	
$A$	5	$C$ to $D$				16	
$S$	6	$C$ to $D$				17	
$H$	7	Store to $B$				0	
$V$	8	Store to $A$				27	
$T$	9	$C$ to Store				25	
$U$	10	$C$ to Store				0	
$R$	11	$B$ to $D$	$E$ to $G$			19	
$L$	12	$C$ to $D$	$E$ to $G$			22	
$G$	13		$E$ to $G$	$(1)C_5$		18	
$I$	14	Input to Store				0	
$O$	15	Store to Output				0	
	16	$(D + \text{Store})$ to $C$				0	
	17	$(D - \text{Store})$ to $C$				0	
	18				1	0	1
	19	$D$ to $B$ ( $R$ )*	$(G - 1)$ to $E$			20	
	20	$C$ to $D$		$(1)E_5$		21	

**Table:**  
**Microinstructions**  
**for Wilkes Example**

(Page 1 of 2)

		Arithmetical Unit	Control Register Unit	Conditional Flip-Flop		Next Microinstruction	
				Set	Use	0	1
	21	$D$ to $C$ ( $R$ )			1	11	0
	22	$D$ to $C$ ( $L$ )†	$(G - 1)$ to $E$			23	
	23	$B$ to $D$		$(1)E_5$		24	
	24	$D$ to $B$ ( $L$ )			1	12	0
	25	"0" to $B$				26	
$A$	26	$B$ to $C$				0	
$S$	27	"0" to $C$	"18" to $E$			28	
$H$	28	$B$ to $D$	$E$ to $G$	$(1)B_1$		29	
$V$	29	$D$ to $B$ ( $R$ )	$(G - "1")$ to $E$			30	
$T$	30	$C$ to $D$ ( $R$ )		$(2)E_5$	1	31	32
$U$	31	$D$ to $C$			2	28	33
$R$	32	$(D + A)$ to $C$			2	28	33
$L$	33	$B$ to $D$		$(1)B_1$		34	
$G$	34	$D$ to $B$ ( $R$ )				35	
$I$	35	$C$ to $D$ ( $R$ )			1	36	37
$O$	36	$D$ to $C$				0	
	37	$(D - A)$ to $C$				0	

**Table:**  
**Microinstructions**  
**for Wilkes Example**

(Page 2 of 2)

\* Right shift. The switching circuits in the arithmetic unit are arranged so that the least significant digit of the register  $C$  is placed in the most significant place of register  $B$  during right shift micro-operations, and the most significant digit of register  $C$  (sign digit) is repeated (thus making the correction for negative numbers).

† Left shift. The switching circuits are similarly arranged to pass the most significant digit of register  $B$  to the least significant place of register  $C$  during left shift micro-operations.

