INTRODUCTION OF 8085 MICROPROCESSOR

CHAPTER OUTLINE

- Block diagram of a computer system
  - Basic components of a computer system using block diagrams:
    - Cpu
    - Memory
    - Input and output unit
- Evolution of microprocessor : 4,8,16,32 dan 64 byte
- Nibble, byte, word dan longword
- Fecthing and execution cycles.
- Internal structure and basic operation of a microprocessor (arithmetic and logic unit, control unit, register sets, accumulator, condition code register, program counter, stack pointer)
- Bus system: data bus, address bus and control bus.
- Microprocessor clock system
- Examples of microprocessor: 8085,8086.

Introduction

- A **computer** is a programmable machine that receives input, stores and manipulates data//information, and provides output in a useful format.

1.1 DIAGRAM OF A COMPUTER SYSTEM

A **computer** is a programmable machine that receives input, stores and manipulates data//information, and provides output in a useful format.

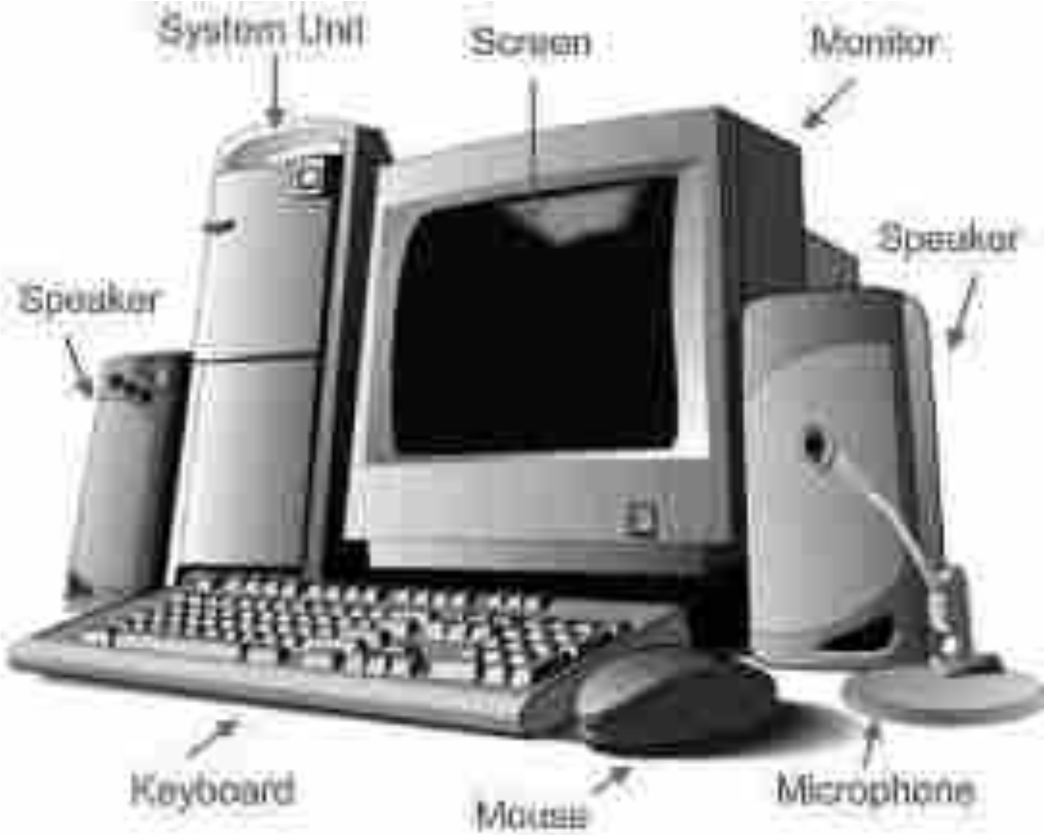
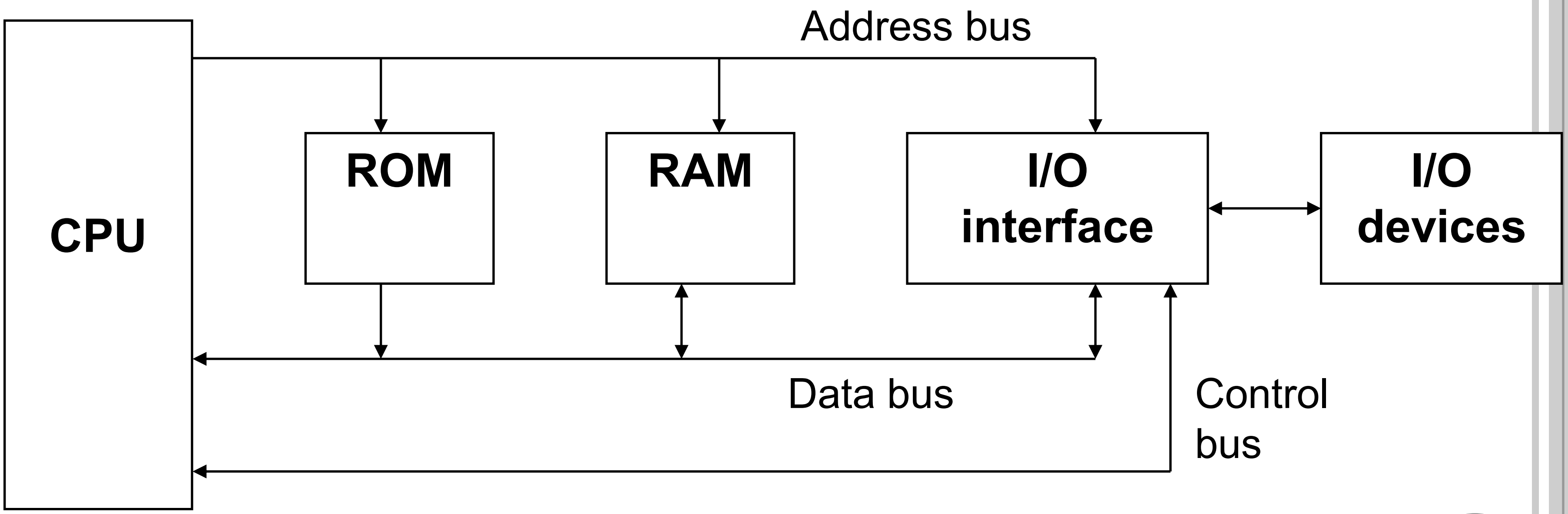


Diagram Of A Computer System

1.1 BLOCK DIAGRAM OF A BASIC COMPUTER SYSTEM

Basic computer system consist of a Central processing unit (CPU), memory (RAM and ROM), input/output (I/O) unit.



Block diagram of a basic computer system

Basic component of microcomputer

1. CPU - Central Processing Unit
   - the portion of a computer system that carries out the instructions of a computer program
   - the primary element carrying out the computer's functions. It is the unit that reads and executes program instructions.
   - The data in the instruction tells the processor what to do.

Pentium D dual core processors

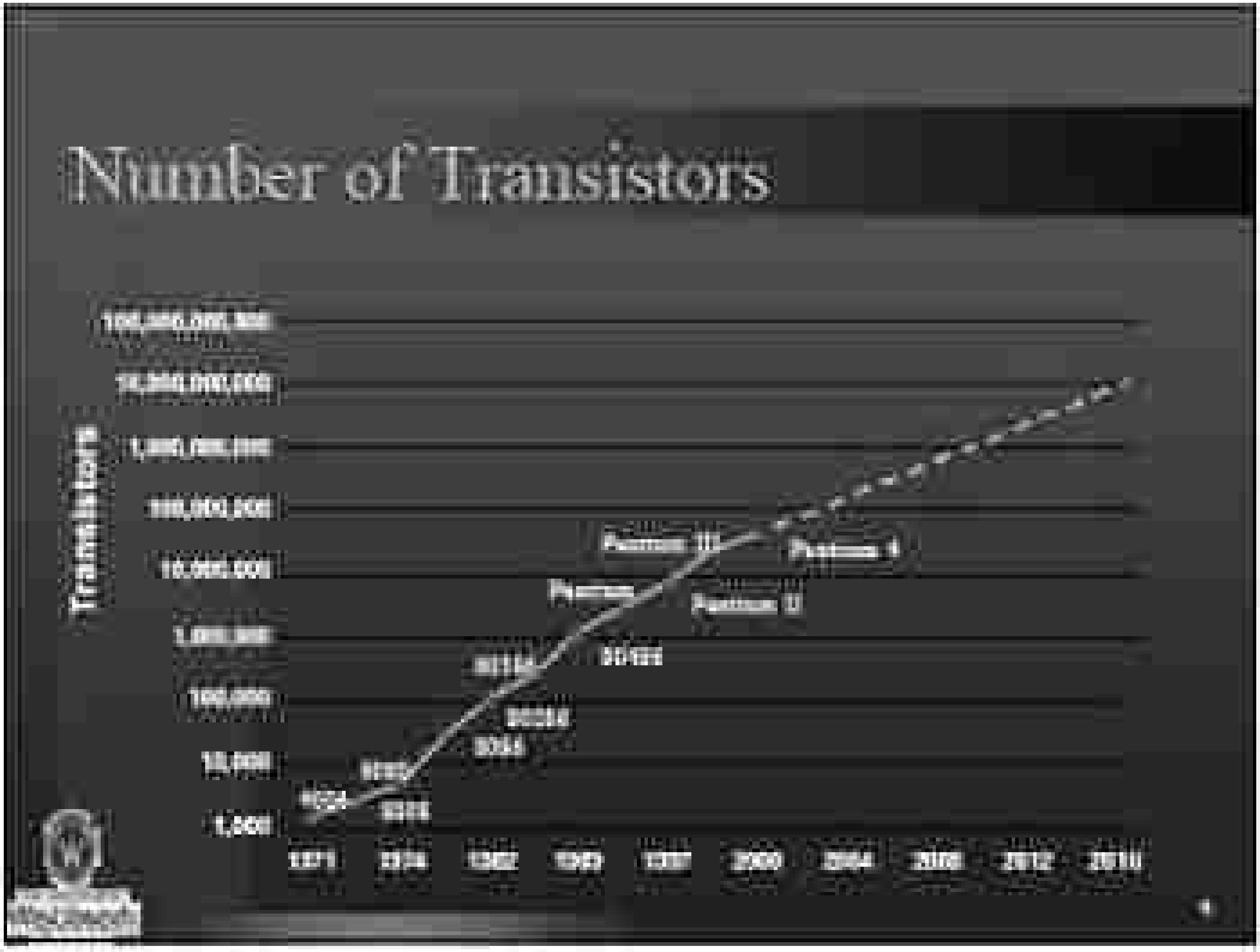
2. Memory

- physical devices used to store data or programs (sequences of instructions) on a temporary or permanent basis for use in an electronic digital computer.
- Computer main memory comes in two principal varieties: random-access memory (RAM) and read-only memory (ROM).
- RAM can be read and written to anytime the CPU commands it, but ROM is pre-loaded with data and software that never changes, so the CPU can only read from it.
- ROM is typically used to store the computer's initial start-up instructions.
- In general, the contents of RAM are erased when the power to the computer is turned off, but ROM retains its data indefinitely.
- In a PC, the ROM contains a specialized program called the BIOS that orchestrates loading the computer's operating system from the hard disk drive into RAM whenever the computer is turned on or reset.

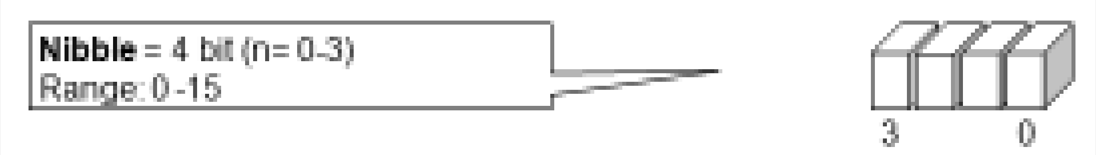
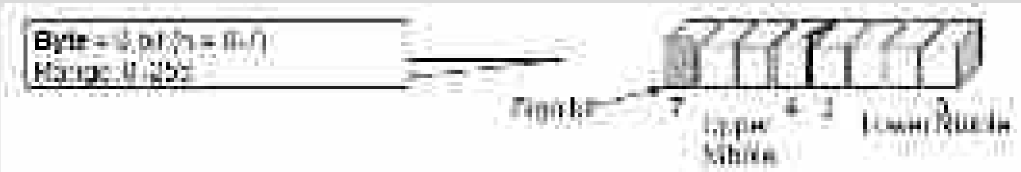
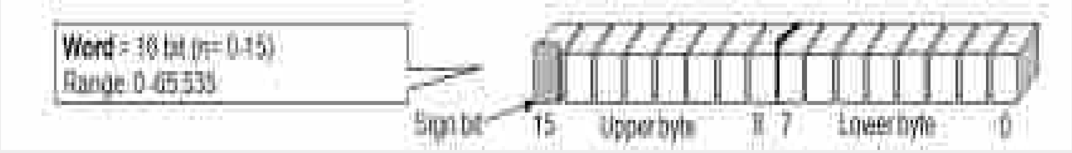
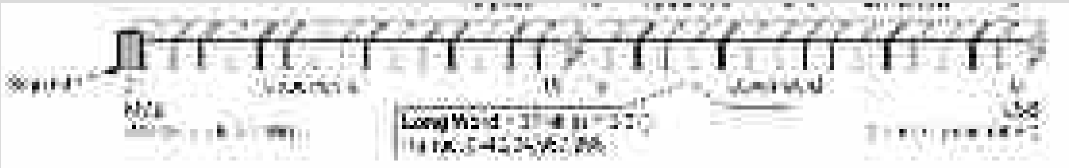
3. I/O Unit

- **Input/output (I/O)**, refers to the communication between an information processing system (such as a computer), and the outside world possibly a human, or another information processing system.
- Inputs are the signals or data received by the system, and outputs are the signals or data sent from it
- Devices that provide input or output to the computer are called peripherals
- On a typical personal computer, peripherals include input devices like the keyboard and mouse, and output devices such as the display and printer. Hard disk drives, floppy disk drives and optical disc drives serve as both input and output devices. Computer networking is another form of I/O.

Evolution of Microprocessor



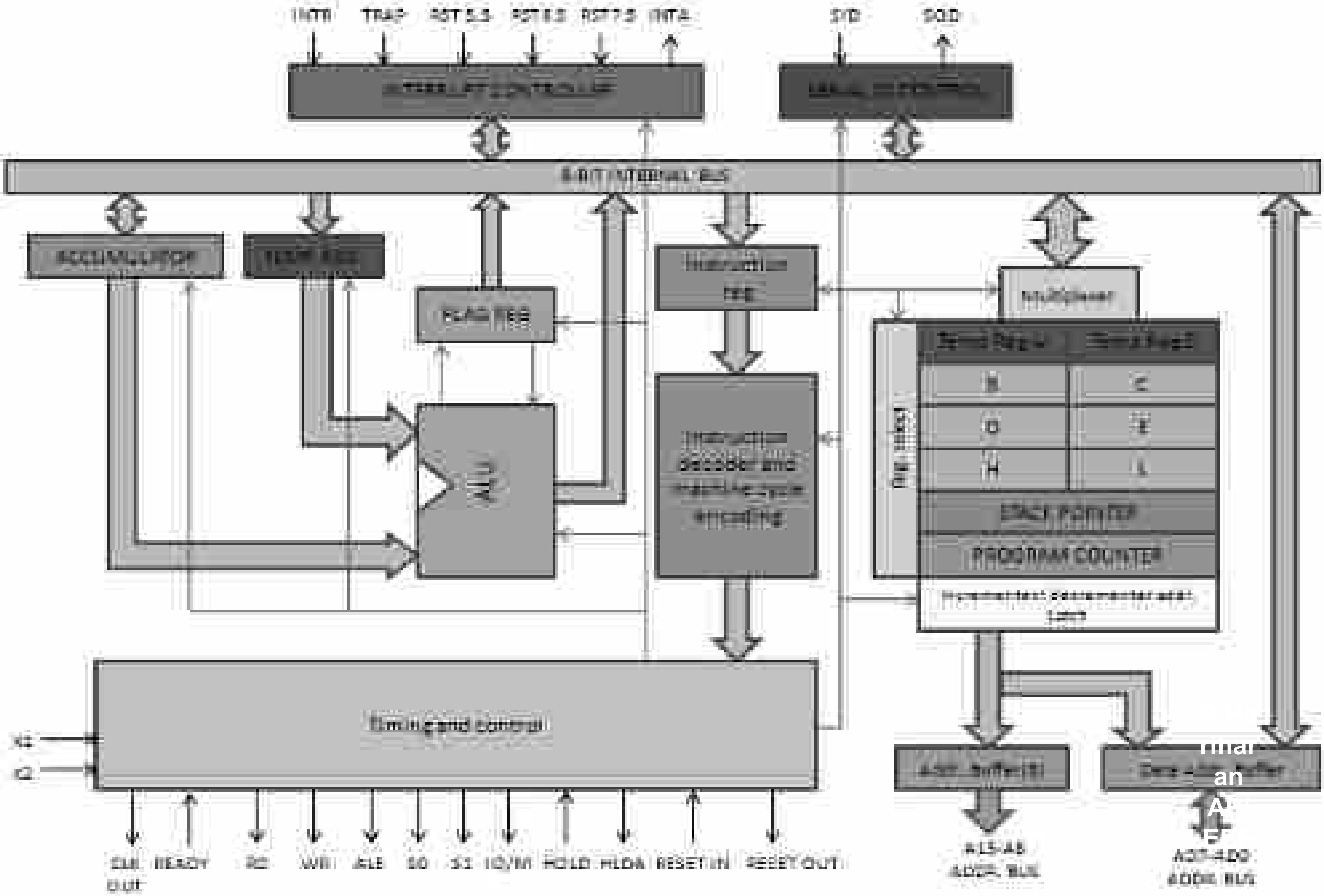
DATA SIZE

Nibble	**4 bit**	
Byte	**8 bit**	
Word	**16 bit**	
Long word	**32 bit**	

8085

- 8-bit general purpose microprocessor
- Capable to address 64k of memory
- Forty pins, requires +5 V single power supply
- 3-MHz single-phase clock.

BLOCK DIAGRAM



rnal
an
A
E

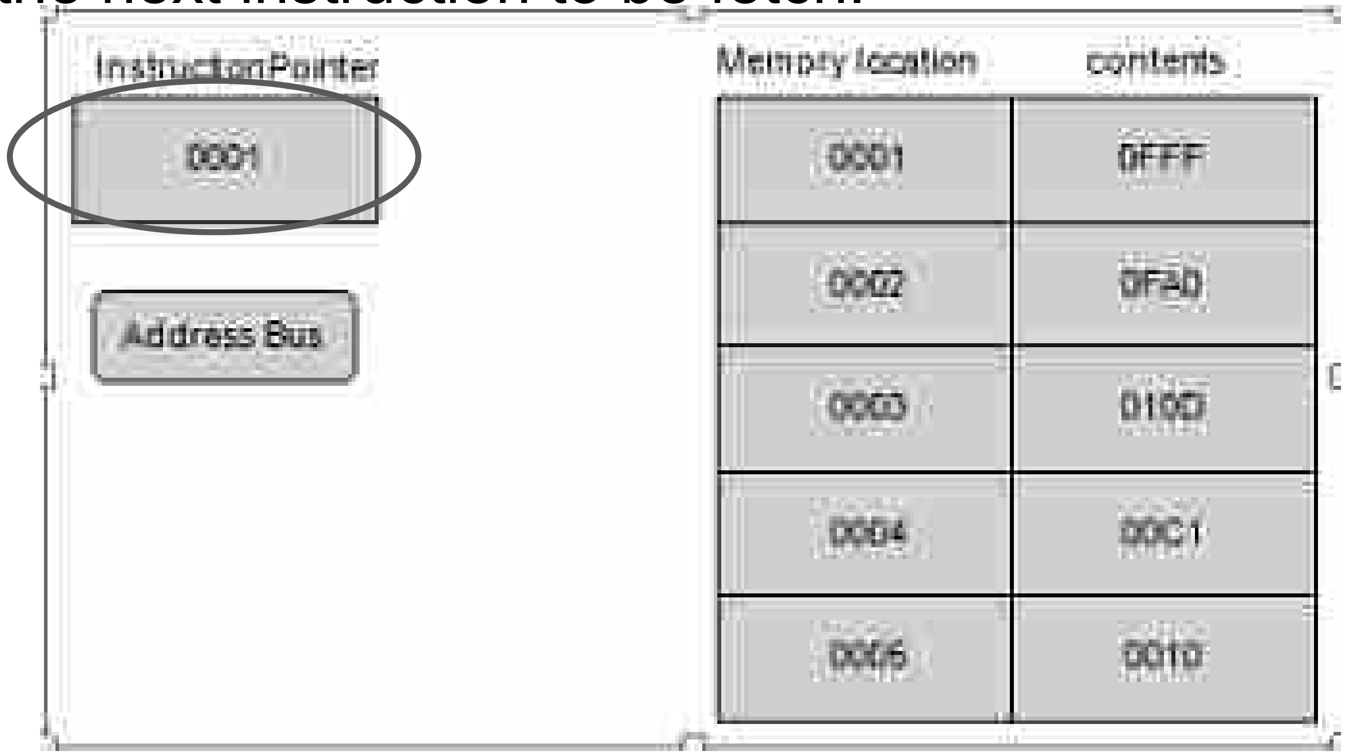
FETCHING & EXECUTION CYCLES

- **Fetching Cycles**
  - The fetch cycle takes the instruction required from memory, stores it in the instruction register, and
  - moves the program counter on one so that it points to the next instruction.
- **Execute cycle**
  - The actual actions which occur during the execute cycle of an instruction.
  - depend on both the instruction itself and the addressing mode specified to be used to access the data that may be required.

FETCHING AN INSTRUCTION

- **Step 1**

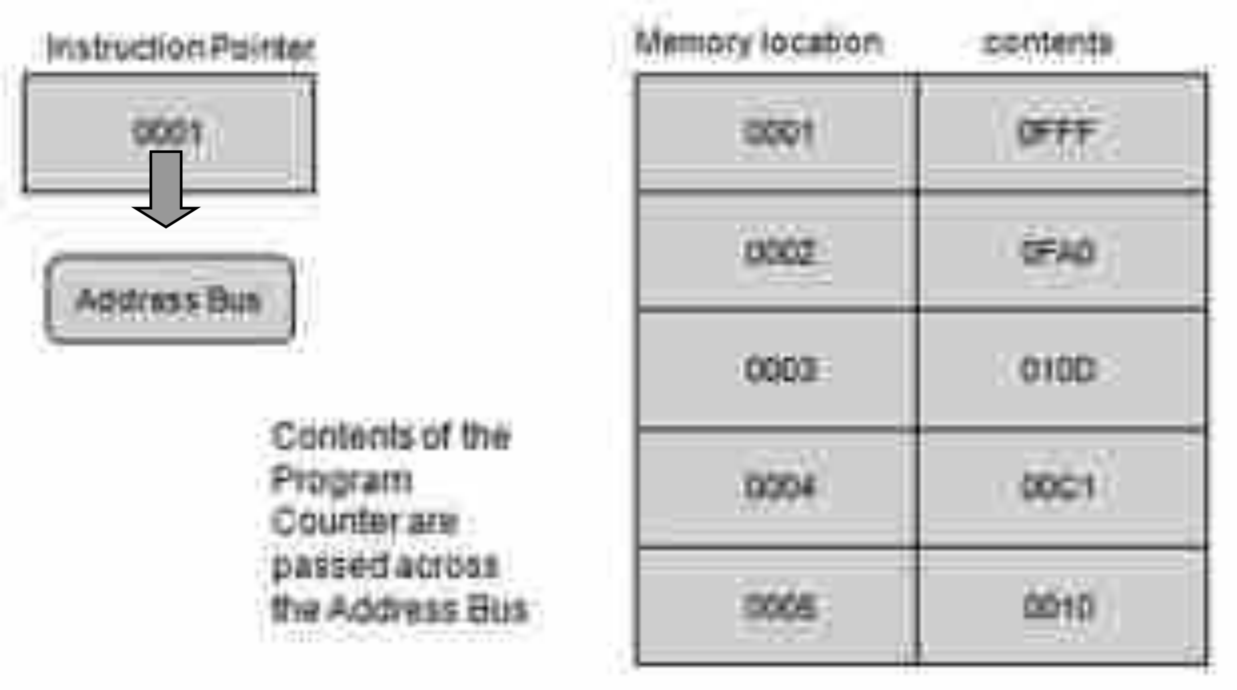
Instruction pointer (program counter) hold the address of the next instruction to be fetch.



Memory location	contents
0001	0FFF
0002	0FAD
0003	0100
0004	00C1
0005	0010

FETCHING AN INSTRUCTION (cont.)

- **Step 2**

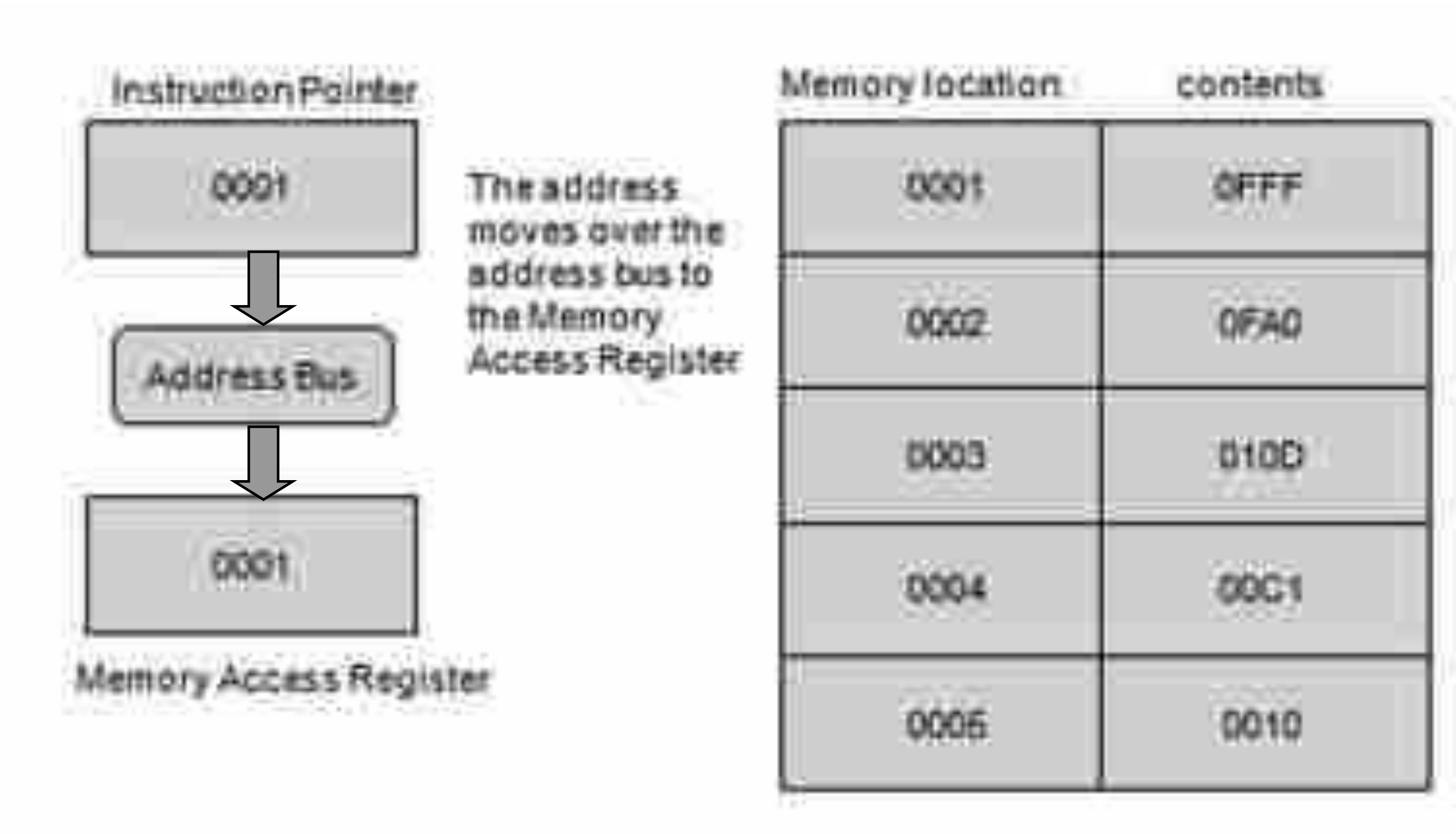


Contents of the Program Counter are passed across the Address Bus

Memory location	contents
0001	0FFF
0002	0FA0
0003	010D
0004	00C1
0005	0010

FETCHING AN INSTRUCTION (cont.)

- **Step 3**

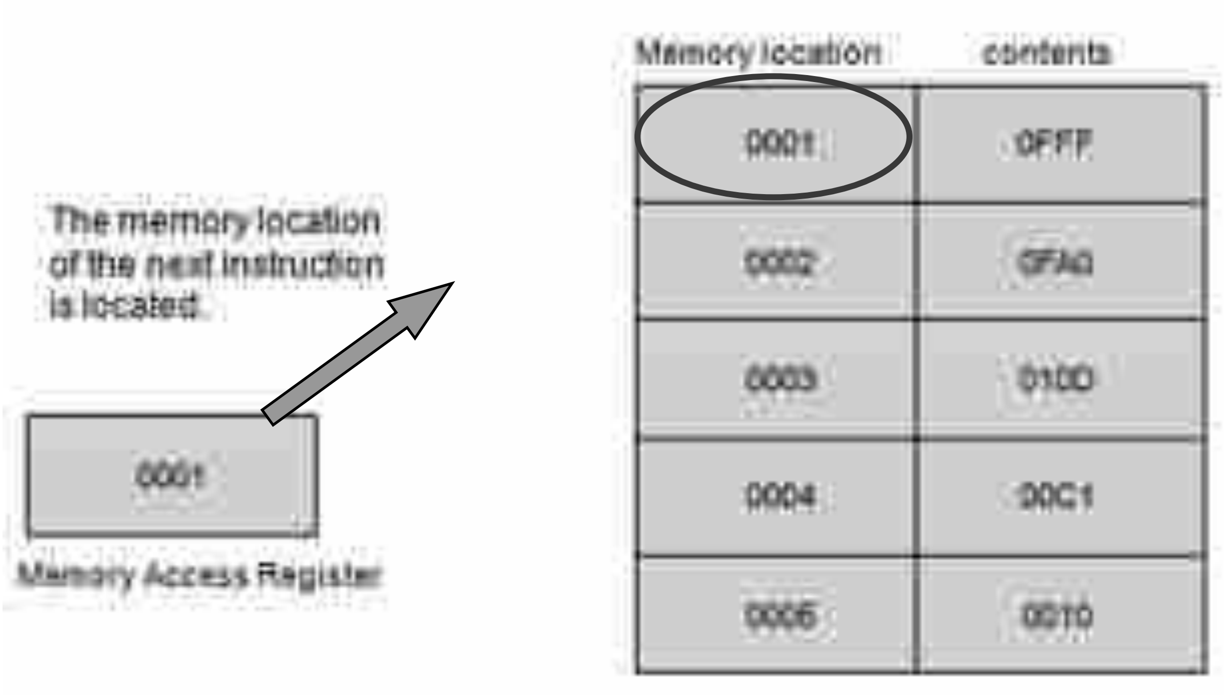


The address moves over the address bus to the Memory Access Register

Memory location	contents
0001	0FFF
0002	0FA0
0003	010D
0004	00C1
0005	0010

FETCHING AN INSTRUCTION (cont.)

- **Step 4**

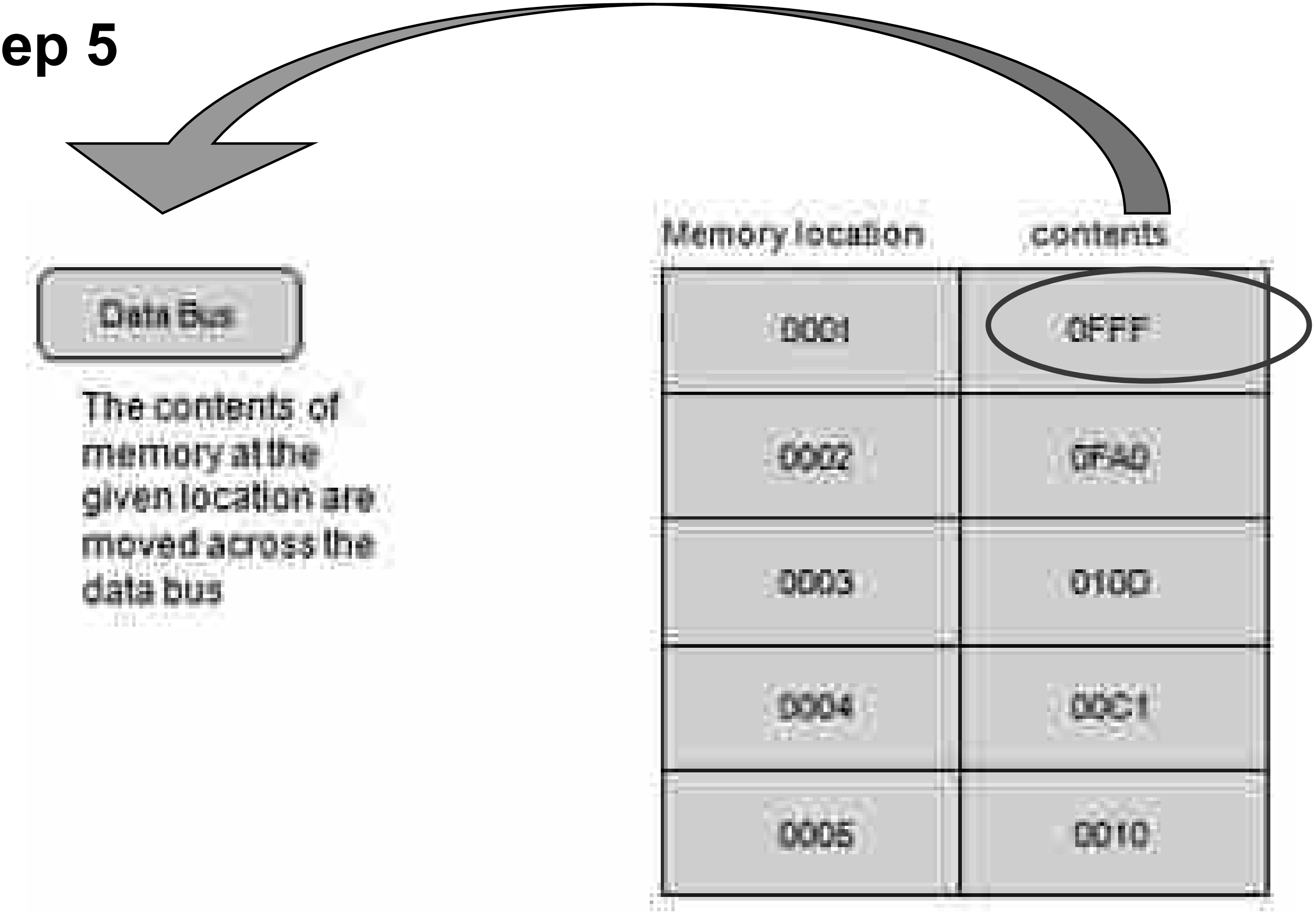


The memory location of the next instruction is located.

Memory location	contents
0001	0FFF
0002	0FA0
0003	010D
0004	00C1
0005	0010

FETCHING AN INSTRUCTION (cont.)

- **Step 5**

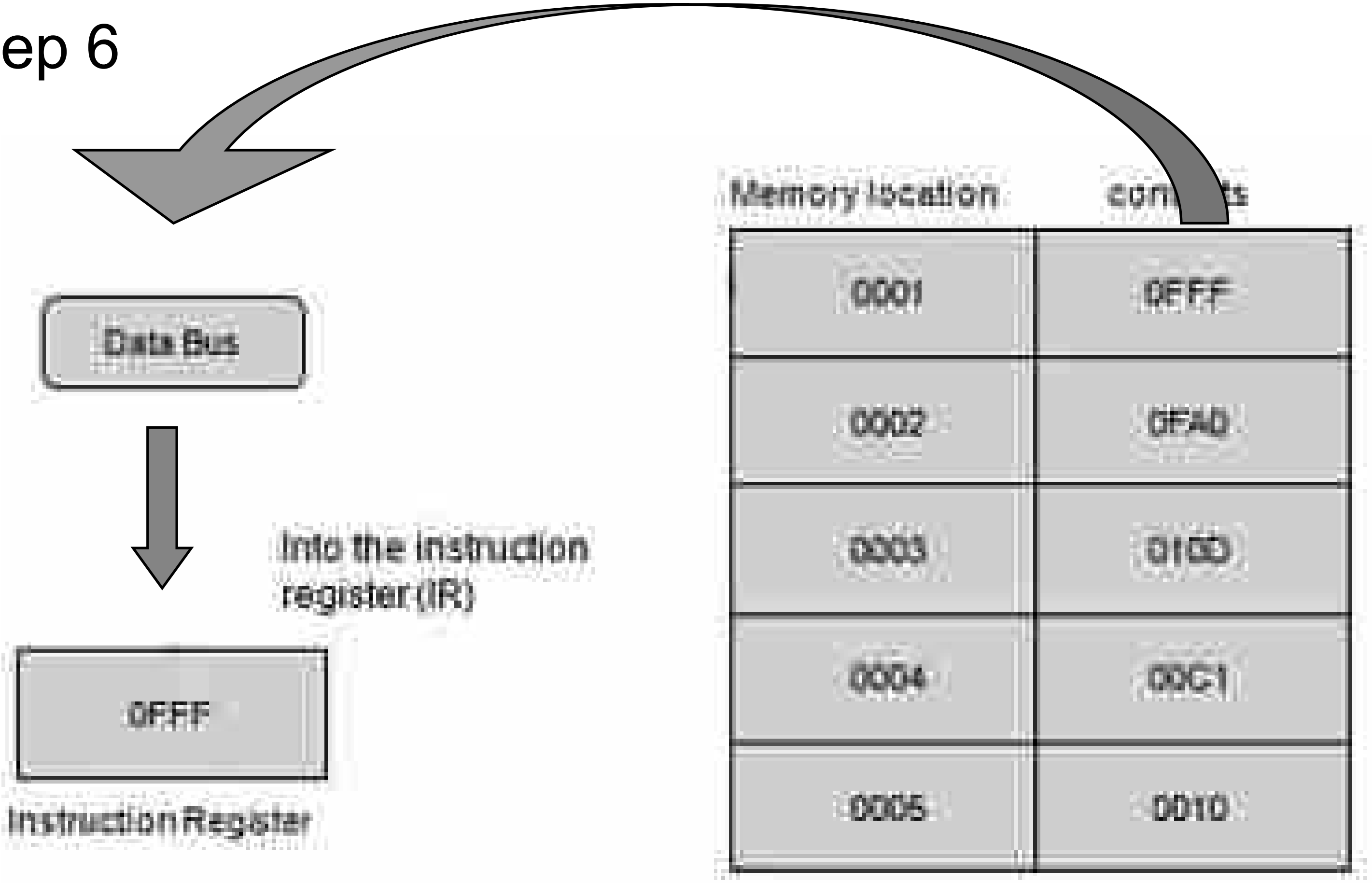


The contents of memory at the given location are moved across the data bus

Memory location	contents
0001	0FFF
0002	0FA0
0003	010D
0004	00C1
0005	0010

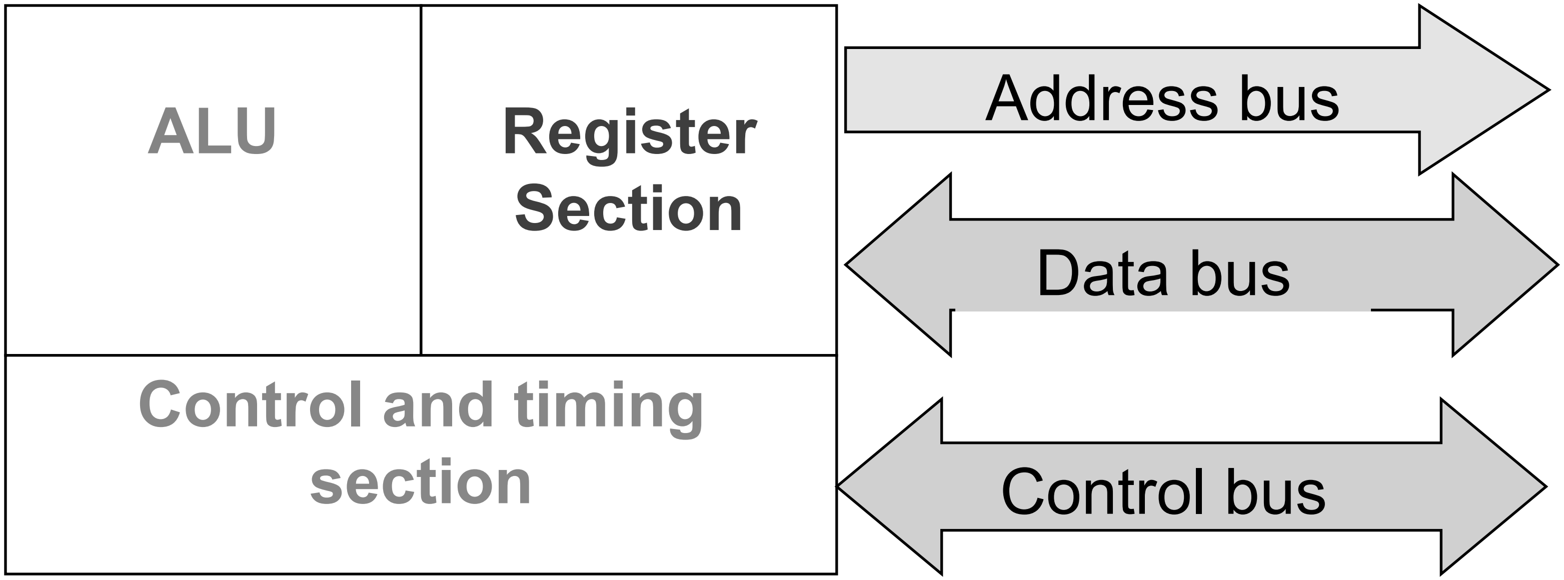
FETCHING AN INSTRUCTION (cont.)

- Step 6



Memory location	contents
0001	0FFF
0002	0FAD
0003	010D
0004	00C1
0005	0010

INTERNAL STRUCTURE AND BASIC OPERATION OF MICROPROCESSOR

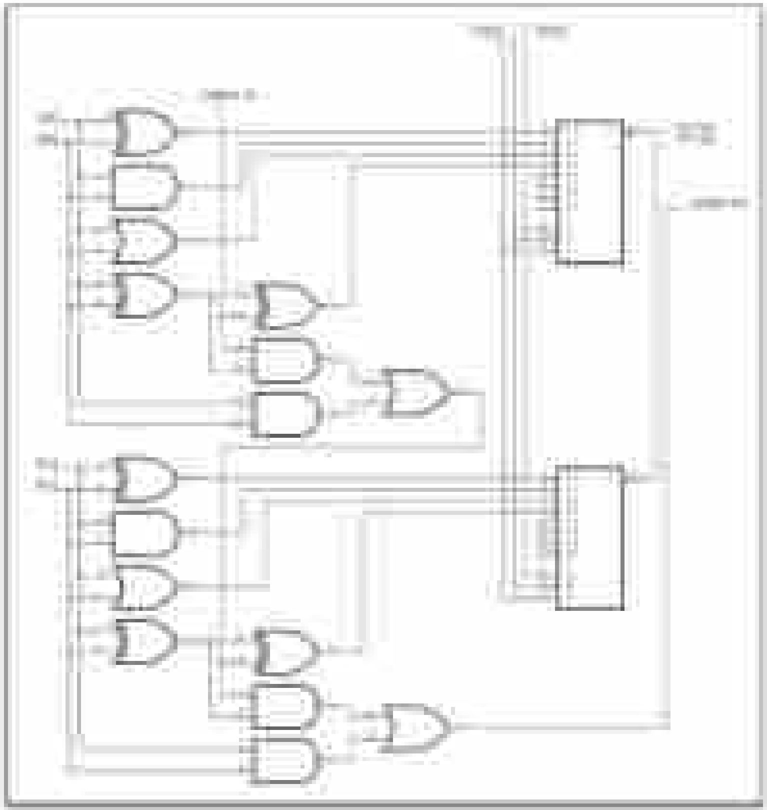


Block diagram of a microprocessor

ARITHMETIC AND LOGIC UNIT (ALU)

- The component that performs the arithmetic and logical operations
- the most important components in a microprocessor, and is typically the part of the processor that is designed first.
- able to perform the basic logical operations (AND, OR), including the addition operation.
- The inclusion of inverters on the inputs enables the same ALU hardware to perform the subtraction operation (adding an inverted operand), and the operations NAND and NOR.

INTERNAL STRUCTURE OF ALU

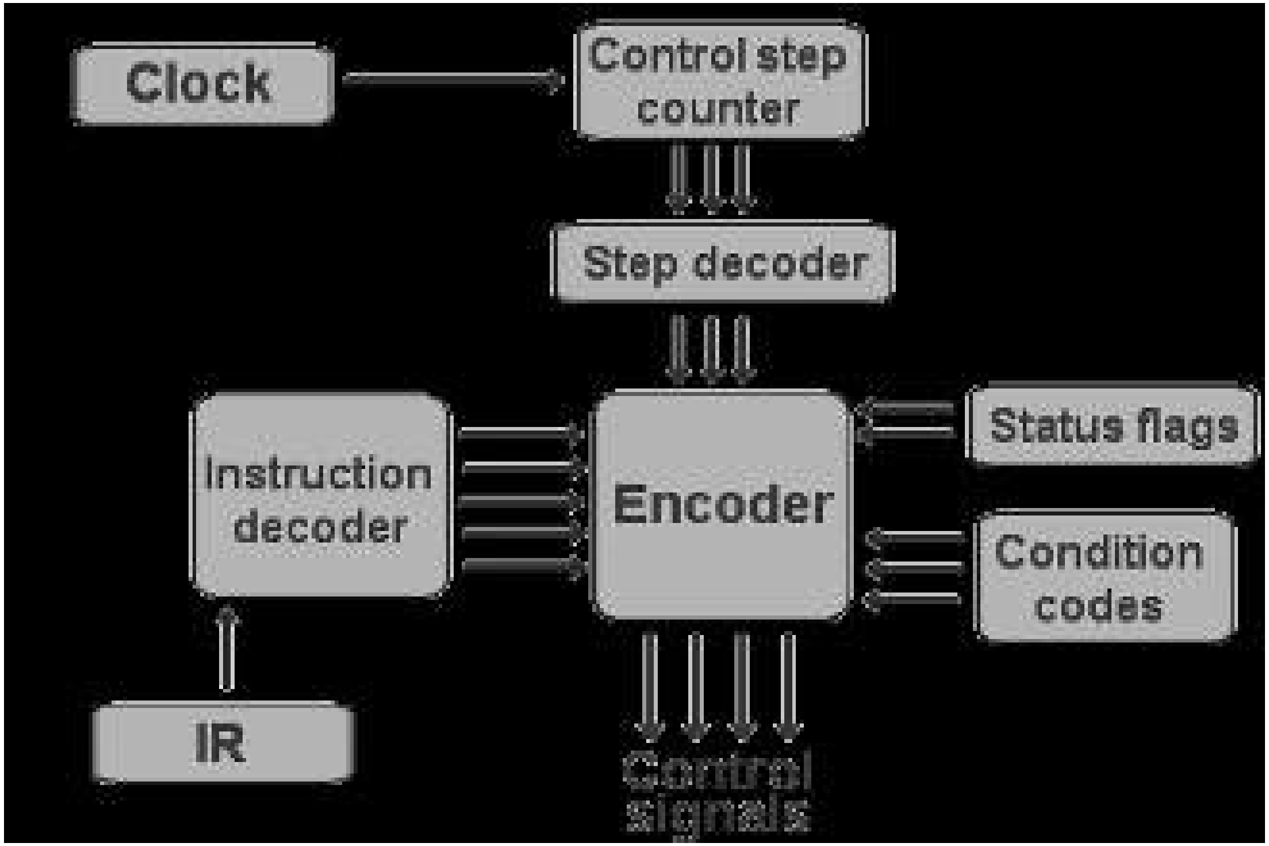


2 bits of ALU

CONTROL UNIT

- The circuitry that controls the flow of information through the processor, and coordinates the activities of the other units within it.
- In a way, it is the "brain within the brain", as it controls what happens inside the processor, which in turn controls the rest of the PC.
- On a regular processor, the control unit performs the tasks of fetching, decoding, managing execution and then storing results.

INTERNAL STRUCTURE OF CONTROL UNIT



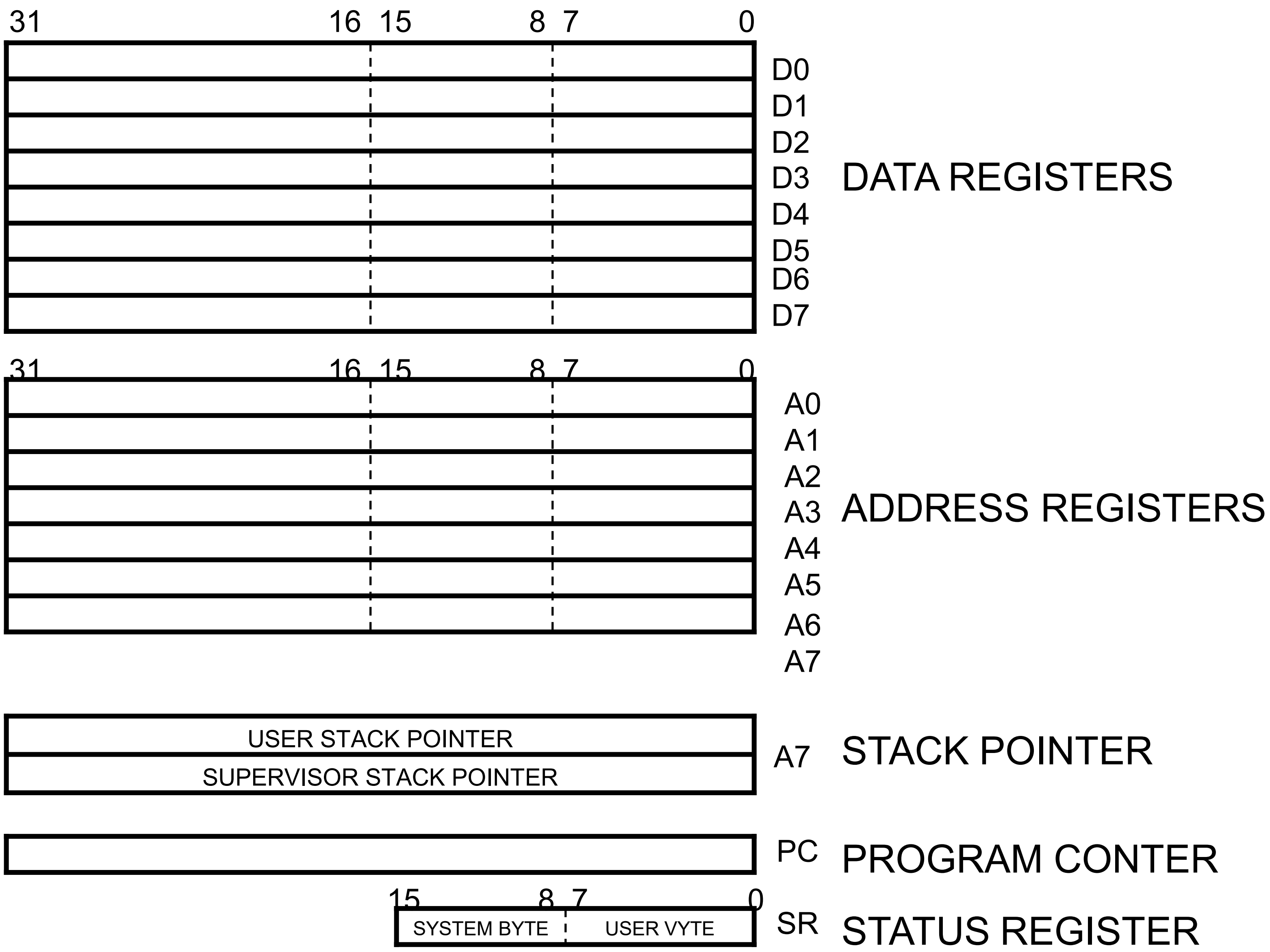
REGISTER SETS

- The register section/array consists completely of circuitry used to **temporarily store data or program codes until they** are sent to the ALU or to the control section or to memory.

- The number of registers are different for any particular CPU and the more register a CPU have will result in easier programming tasks.

- Registers are normally measured by the number of bits they can hold, for example, an "8-bit register" or a "32-bit register".

REGISTER IN MOTOROLA 68000 MICROPROCESSOR



31 16 15 8 7 0

D0
D1
D2
D3
D4
D5
D6
D7

DATA REGISTERS

31 16 15 8 7 0

A0
A1
A2
A3
A4
A5
A6
A7

ADDRESS REGISTERS

USER STACK POINTER
SUPERVISOR STACK POINTER

A7 STACK POINTER

PC PROGRAM CONTER

15 8 7 0

SYSTEM BYTE
USER VYTE

SR STATUS REGISTER

ACCUMULATOR

- a register in which intermediate arithmetic and logic results are stored.
- example for accumulator use is summing a list of numbers.
  - The accumulator is initially set to zero, then each number in turn is added to the value in the accumulator.
  - Only when all numbers have been added is the result held in the accumulator written to main memory or to another, non-accumulator, CPU register.

CONDITION CODE REGISTER (CCR)

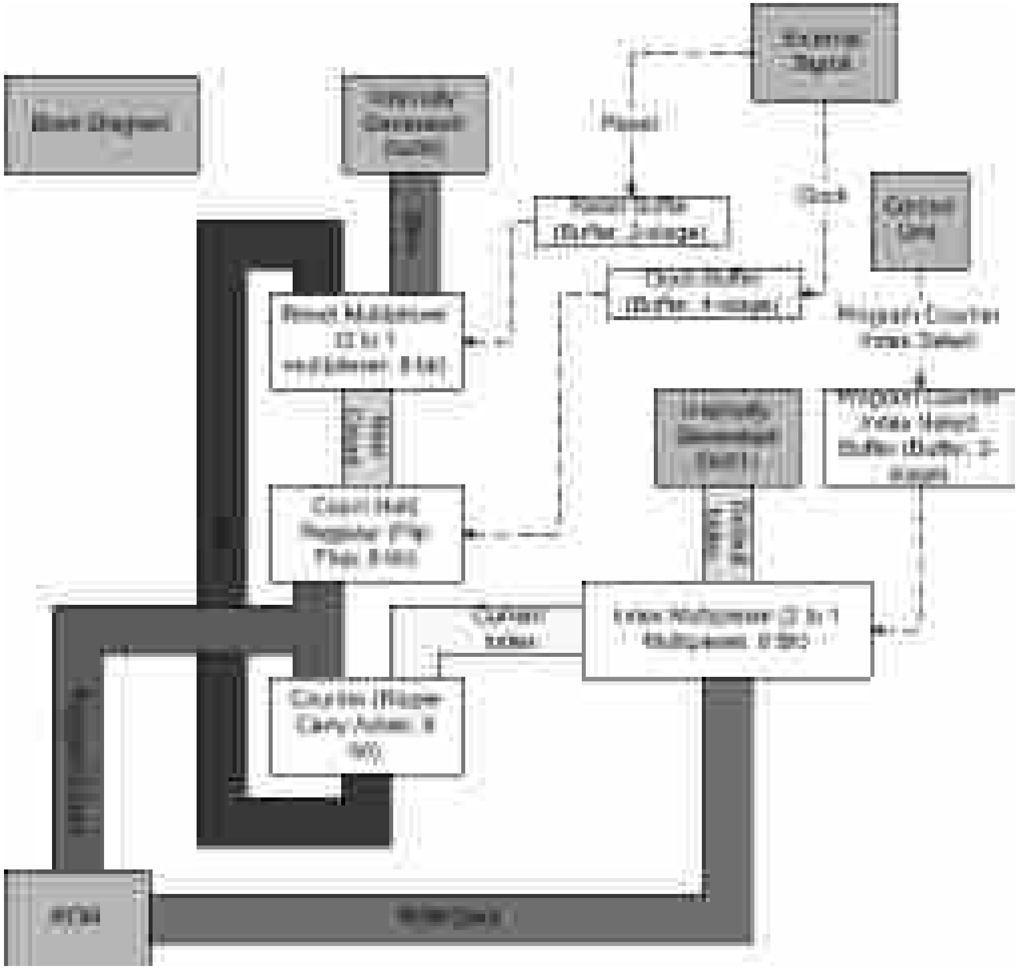
- an 8 bit register used to store the status of CPU, such as carry, zero, overflow and half carry.

Flag	Name	Description
Z	Zero flag	Indicates that the result of a mathematical or logical operation was zero.
C	Carry flag	Indicates that the result of an operation produced an answer greater than the number of available bits. (This flag may also be set before a mathematical operation as an extra operand to certain instructions, e.g. "add with carry".)
X	Extend flag	*Masks the XIRQ request when set. It is set by the hardware and cleared by the software as well is set by unmaskable XIRQ.*
N	Negative/ Sign flag	Indicates that the result of a mathematical operation is negative. In some processors, the N and S flags have different meanings: the S flag indicates whether a subtraction or addition has taken place, whereas the N flag indicates whether the last operation result is positive or negative.
V	Overflow Flag	Indicates that the result of an operation has overflowed according to the CPU's word representation, similar to the carry flag but for signed operations.
I	interrupts	Interrupts can be enabled or disabled by respectively setting or clearing this flag. Modifying this flag may be restricted to programs executing in supervisor mode

PROGRAM COUNTER (PC)

- a 16 bit register, used to store the next address of the operation code to be fetched by the CPU.
- Not much use in programming, but as an indicator to user only.
- Purpose of PC in a Microprocessor
  - to store address of tos (top of stack)
  - to store address of next instruction to be executed.
  - count the number of instructions.
  - to store base address of the stack.

INTERNAL STRUCTURE OF PC


Block Diagram
Externally Generated
External Signal
Reset
Clock
Reset Buffer (Buffer, 2-stage)
Clock Buffer (Buffer, 4-stage)
Control Unit
Program Counter Index Select
Reset Multiplexer (2 to 1 multiplexer, 8-bit)
Count Hold Register (Flip Flop, 8-bit)
Externally Generated
Program Counter Index Select Buffer (Buffer, 3-stage)
Current Index
Index Multiplexer (2 to 1 Multiplexer, 8-bit)
Counter (Ripple Carry Adder, 8-bit)

STACK POINTER (SP)

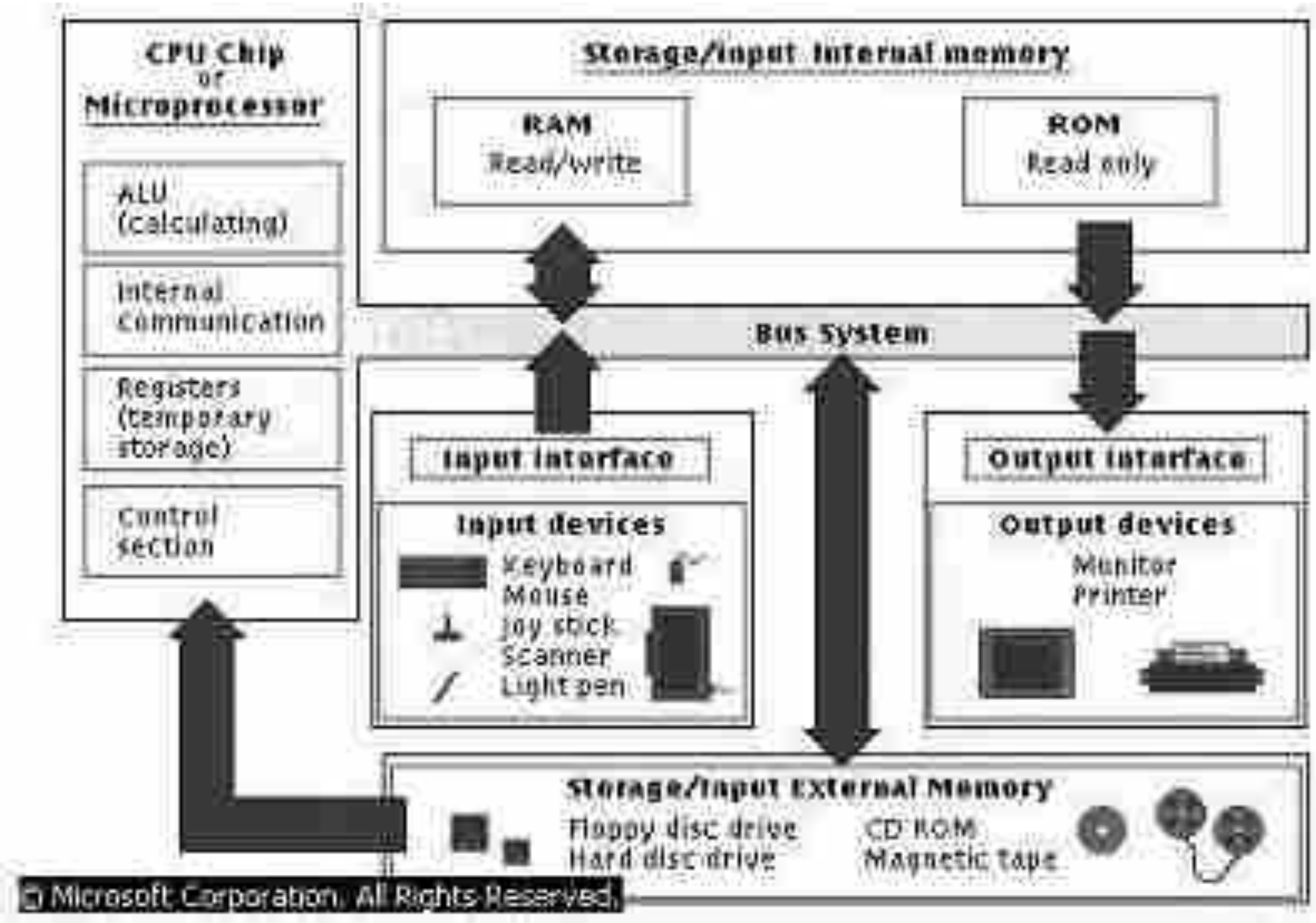
- The stack is configured as a data structure that grows downward from high memory to low memory.
- At any given time, the SP holds the 16-bit address of the next free location in the stack.
- The stack acts like any other stack when there is a subroutine call or on an interrupt. ie. pushing the return address on a jump, and retrieving it after the operation is complete to come back to its original location.

Bus system

- a subsystem that transfers data between computer components inside a computer or between computers.

4 PCI Express bus card slots (from top to bottom: x4, x16, x1 and x16), compared to a traditional 32-bit PCI bus card slot (very bottom).

Bus system connection



DATA BUS

- The data bus is 'bi-directional'
  - data or instruction codes from memory or input/output.are transferred into the microprocessor
  - the result of an operation or computation is sent out from the microprocessor to the memory or input/output.
- Depending on the particular microprocessor, the data bus can handle 8 bit or 16 bit data.

ADDRESS BUS

- The address bus is '**unidirectional**', over which the microprocessor sends an address code to the memory or input/output.
- The size (width) of the address bus is specified by the number of bits it can handle.
- The more bits there are in the address bus, the more memory locations a microprocessor can access.
- A 16 bit address bus is capable of addressing 65,536 (64K) addresses.

CONTROL BUS

- The control bus is used by the microprocessor to send out or receive timing and control signals in order to coordinate and regulate its operation and to communicate with other devices, i.e. memory or input/output.

MICRO PROCESSOR CLOCK

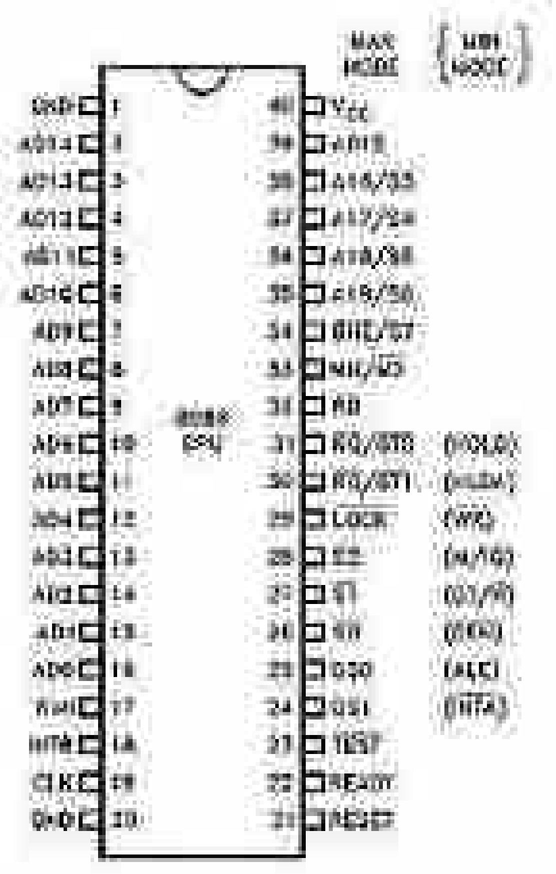
- Also called clock rate, the speed at which a microprocessor executes instructions. Every computer contains an internal clock that regulates the rate at which instructions are executed and synchronizes all the various computer components.
- The CPU requires a fixed number of clock ticks (or clock cycles) to execute each instruction. The faster the clock, the more instructions the CPU can execute per second. Clock speeds are expressed in megahertz (MHz) or gigahertz ((GHz).
- Some microprocessors are superscalar, which means that they can execute more than one instruction per clock cycle.
- Like CPUs, expansion buses also have clock speeds. Ideally, the CPU clock speed and the bus clock speed should be the same so that neither component slows down the other. In practice, the bus clock speed is often slower than the CPU clock speed, which creates a bottleneck. This is why new local buses, such as AGP, have been developed.

EXAMPLES OF MICRO PROCESSOR

- Intel 8085
- Intel 8086

8086

- The 8086 is a 16-bit microprocessor chip designed by Intel, which gave rise to the x86 architecture; development work on the 8086 design started in the spring of 1976 and the chip was introduced to the market in the summer of 1978.
- The Intel 8088, released in 1979, was a slightly modified chip with an external 8-bit data bus (allowing the use of cheaper and fewer supporting logic chips and is notable as the processor used in the original IBM PC.



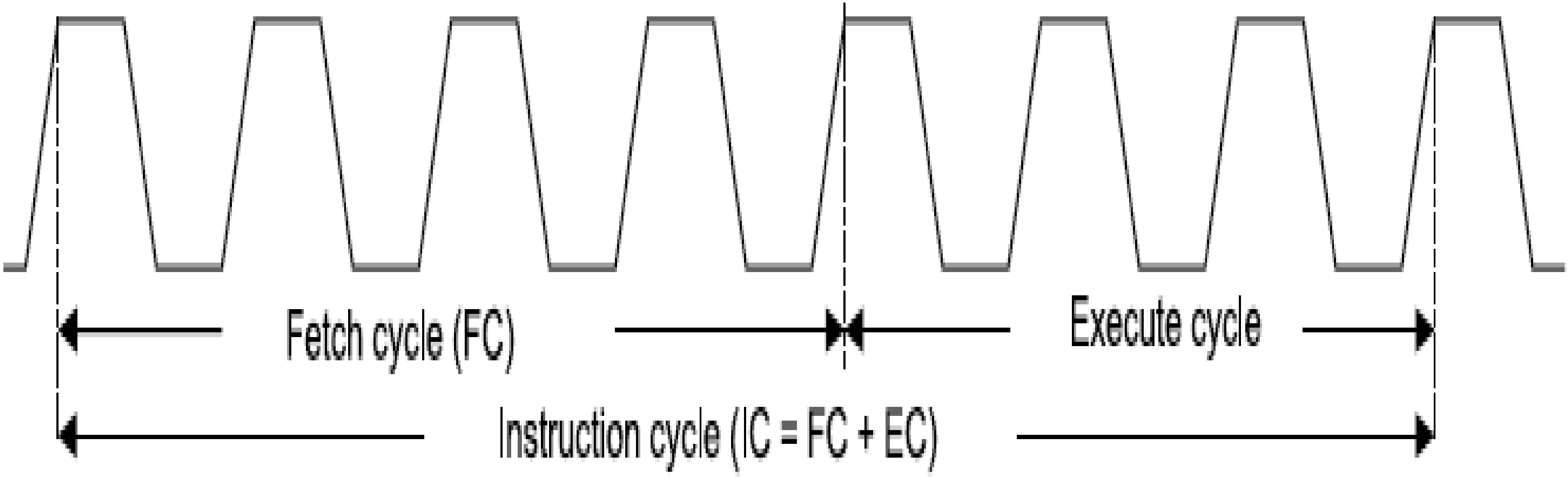
8085

- The Intel 8085 is an 8-bit microprocessor introduced by Intel in 1977.
- It was binary-compatible with the more-famous Intel 8080 but required less supporting hardware, thus allowing simpler and less expensive microcomputer systems to be built.

An Intel 8085AH processor.	
Produced	From 1977 to 1990s
Common manufacturer(s)	•Intel and several others
Max. CPU clock rate	3,5 and 6 MHz
Instruction set	pre x86
Package(s)	•40 pin DIP

TIMING DIAGRAM

Instruction Cycle:

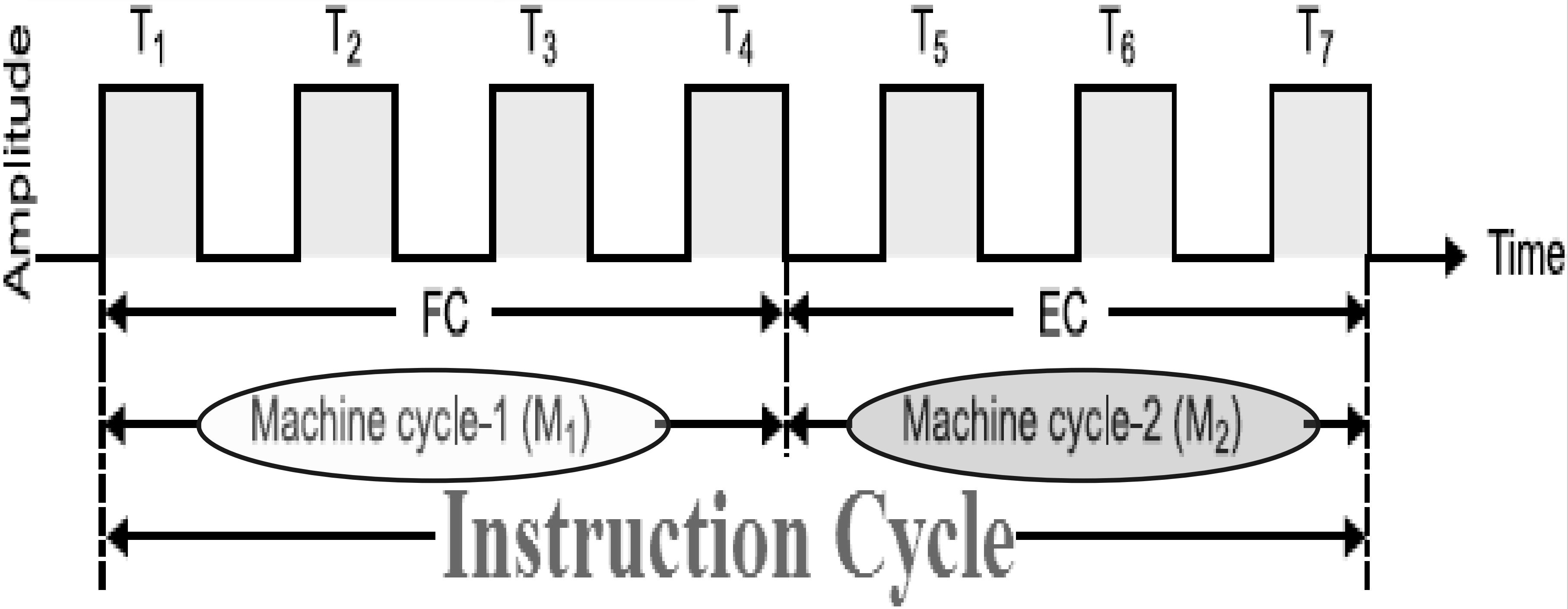


Fetch

Decode

Execute

Machine Cycle:

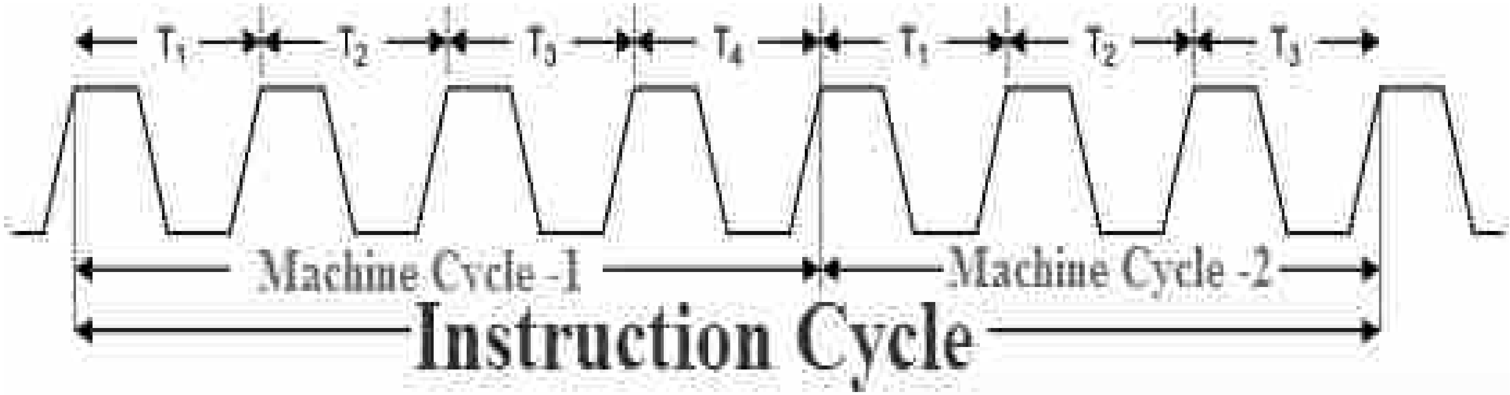


time required to access the memory or input/output devices is called machine cycle

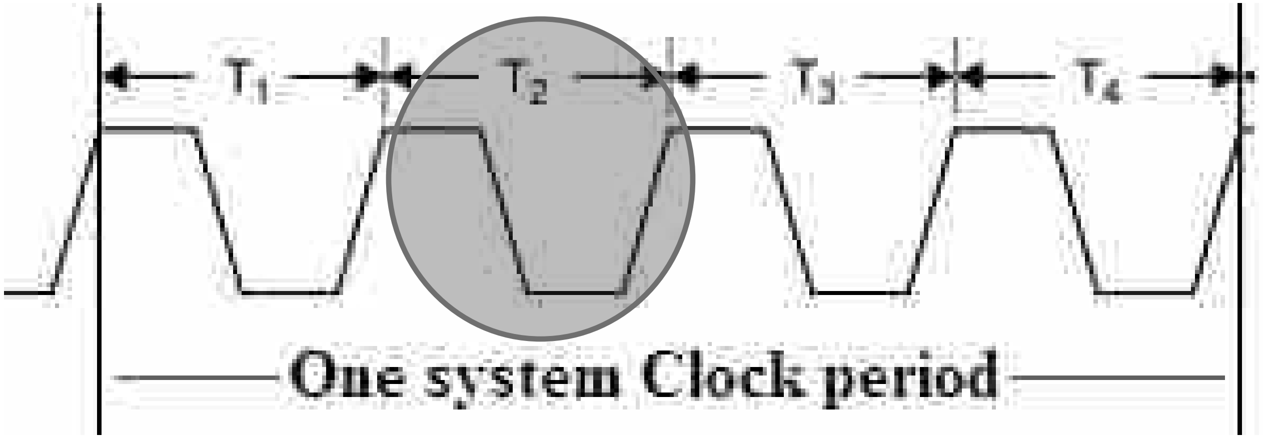
rihar
an
AP/

T- States:

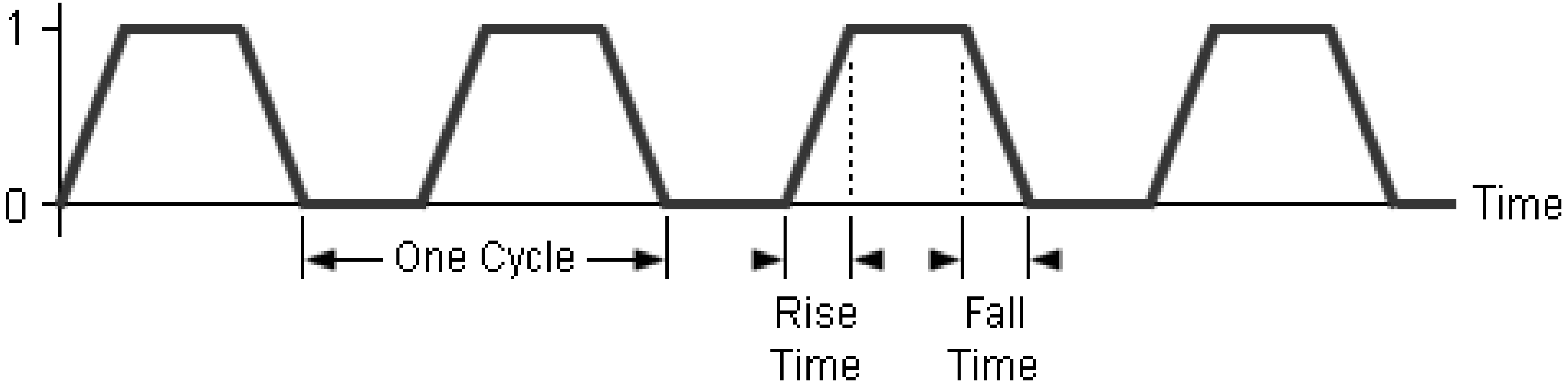
The machine cycle and instruction cycle takes **multiple clock periods**.



A portion of an operation carried out in one system clock period is called as T-state

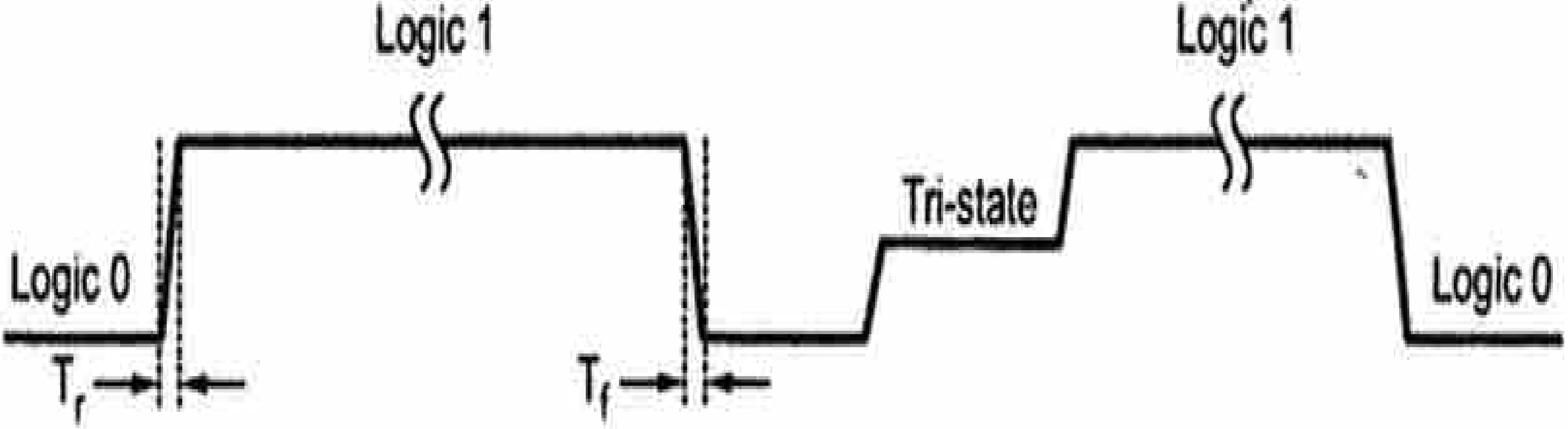


Clock Signal:

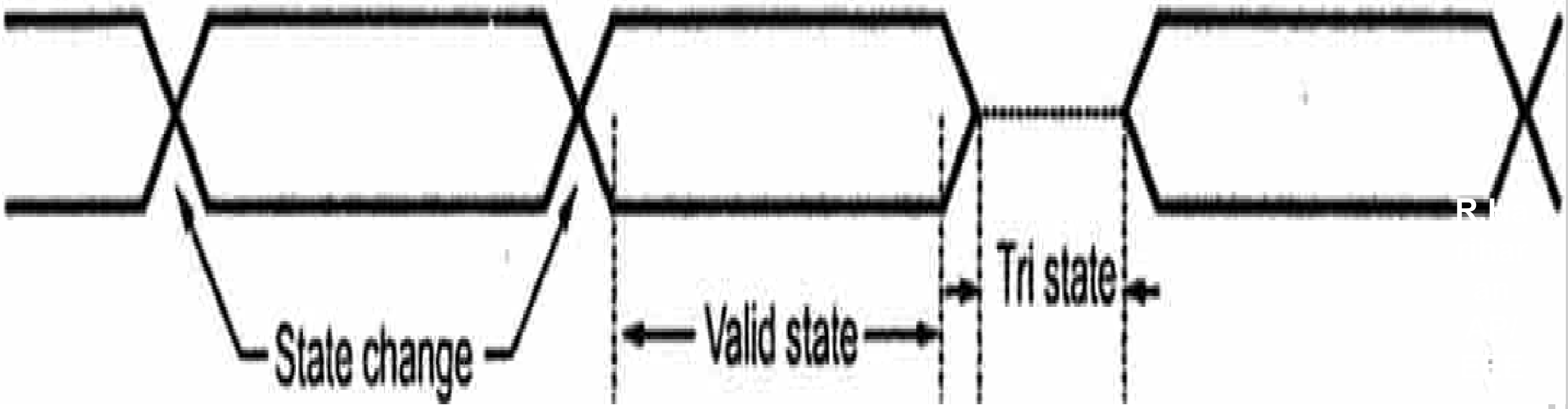


- **Microprocessor operates with reference to clock signals.**
- **X1 and X2 we provide clock signals and this frequency is divided by two.**
- **This frequency is called as the operating frequency.**

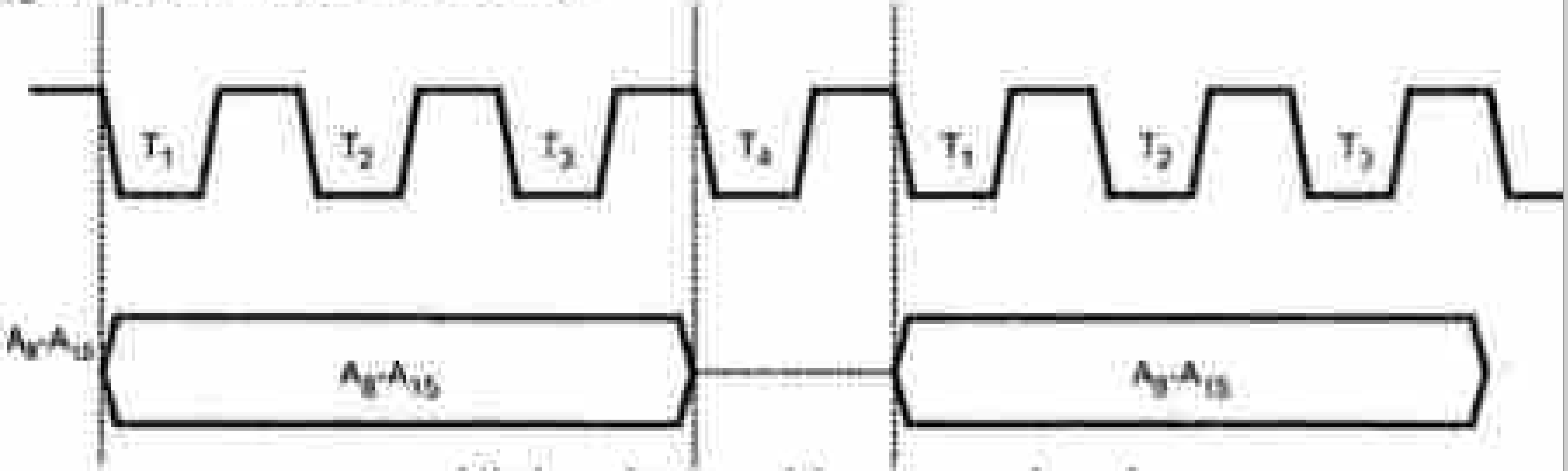
Single signal representation



Group of Signals :

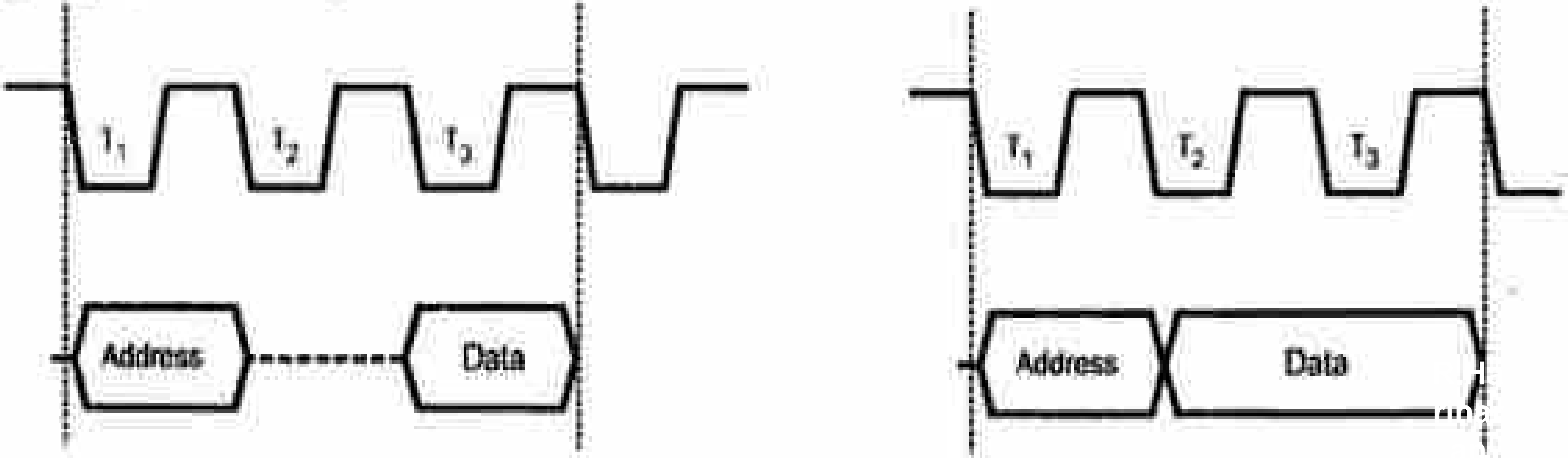


A_8 - A_{15} (Higher Byte Address) :



Higher byte address on A_8 - A_{15}

D_0 - D_7 (Data Bus) :

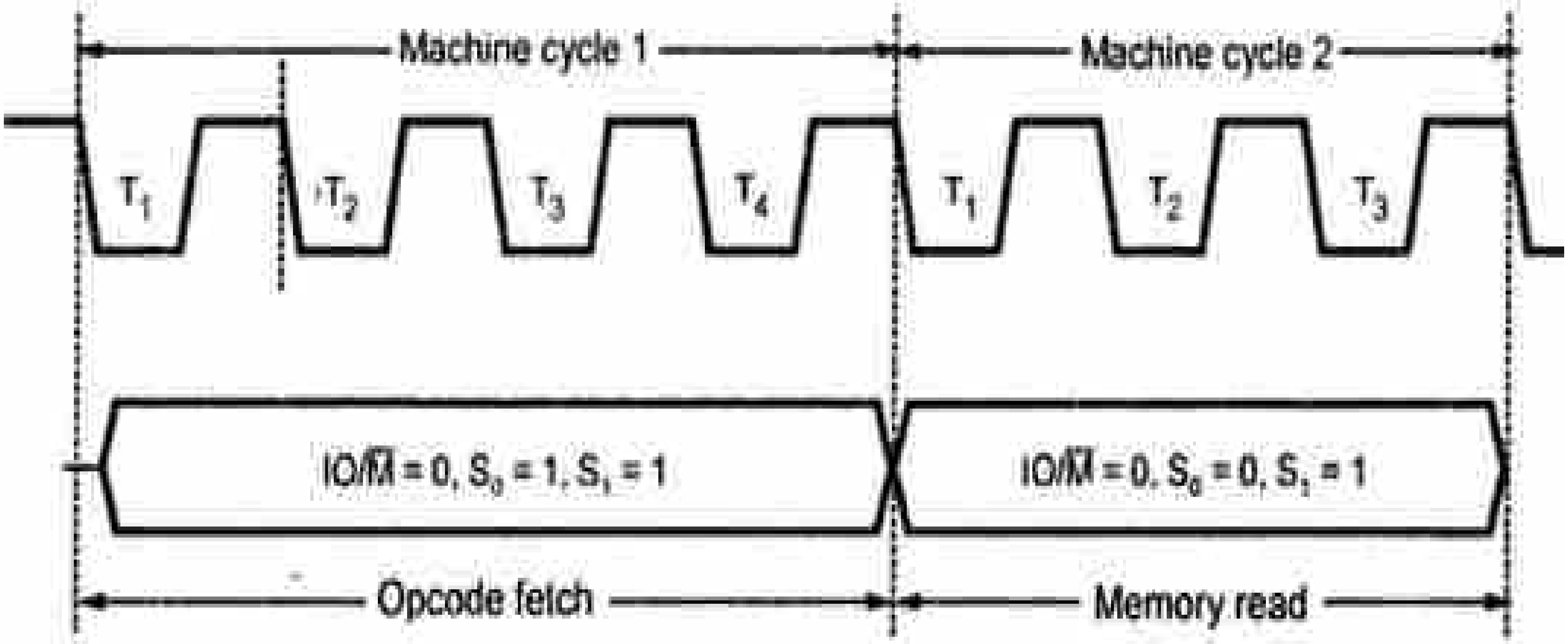


(a) read machine cycle

(b) write cycle

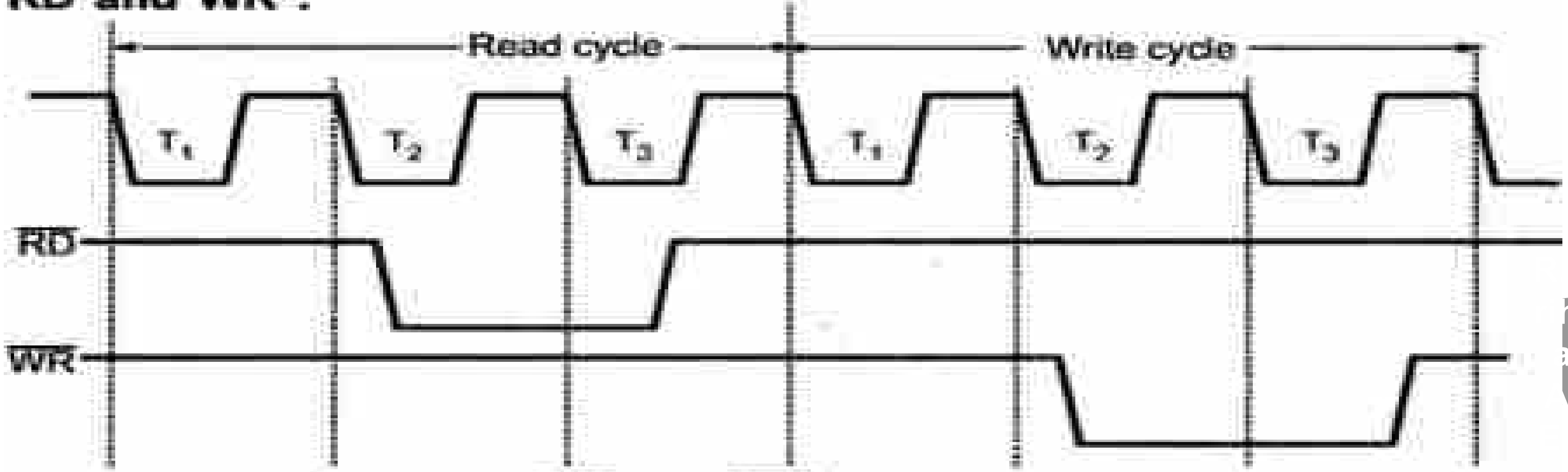
Data bus

$IO/\overline{M}$, S_0, S_1 :



Status signals

$\overline{RD}$ and $\overline{WR}$:



$\overline{RD}$ and $\overline{WR}$ signals

Rule for timing diagram:

1. ALE=1. T1 States.

2. A0-A7=Group Signal. T1 States.

3. A8-A15 = Group Signal. T1,T2,T3 States.

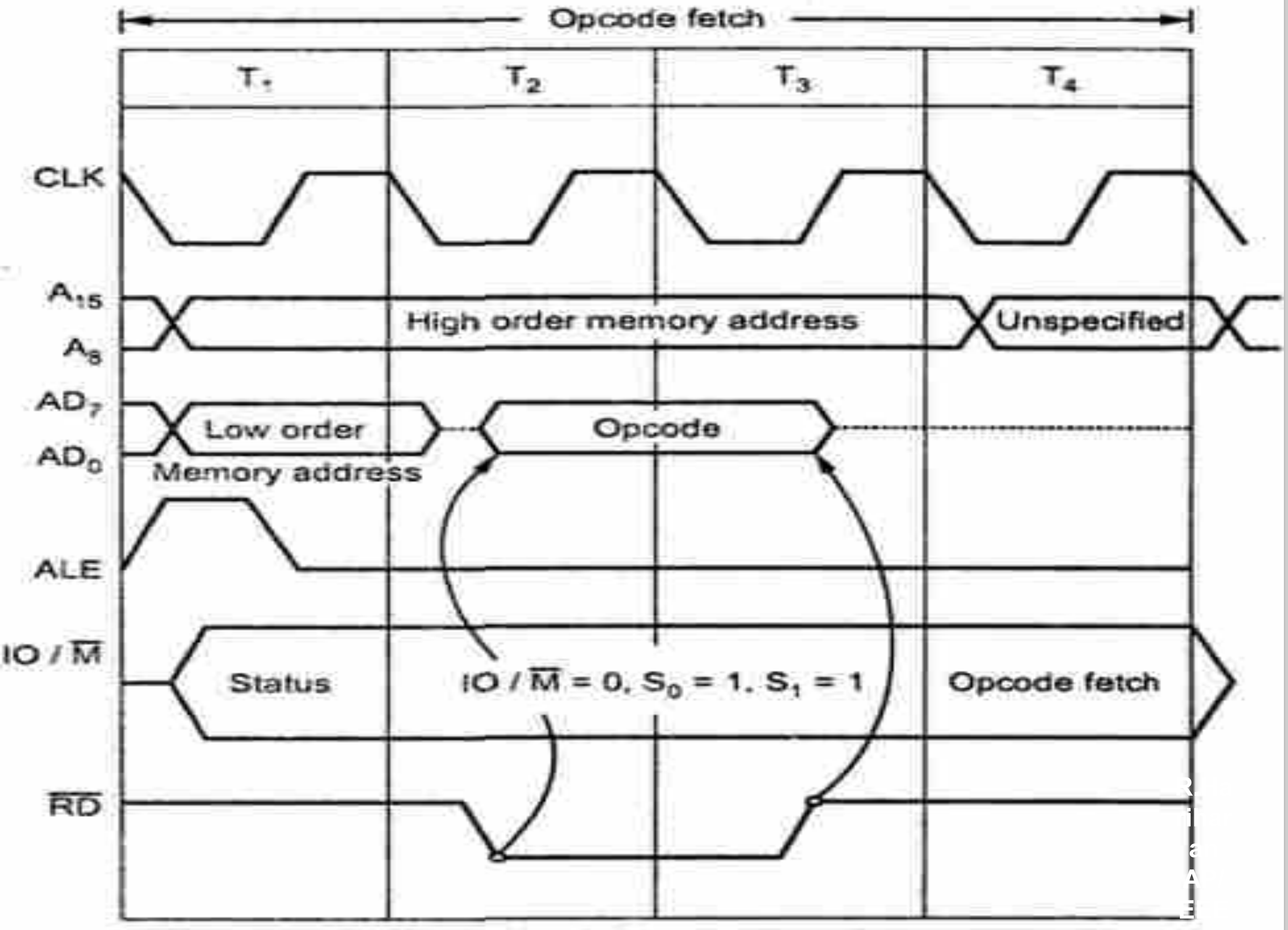
4. D0-D7=Group Signal.

   Read Cycle : T1-Address, T3 - Data

   Write Cycle : T1-Address, T2,T3 -Data

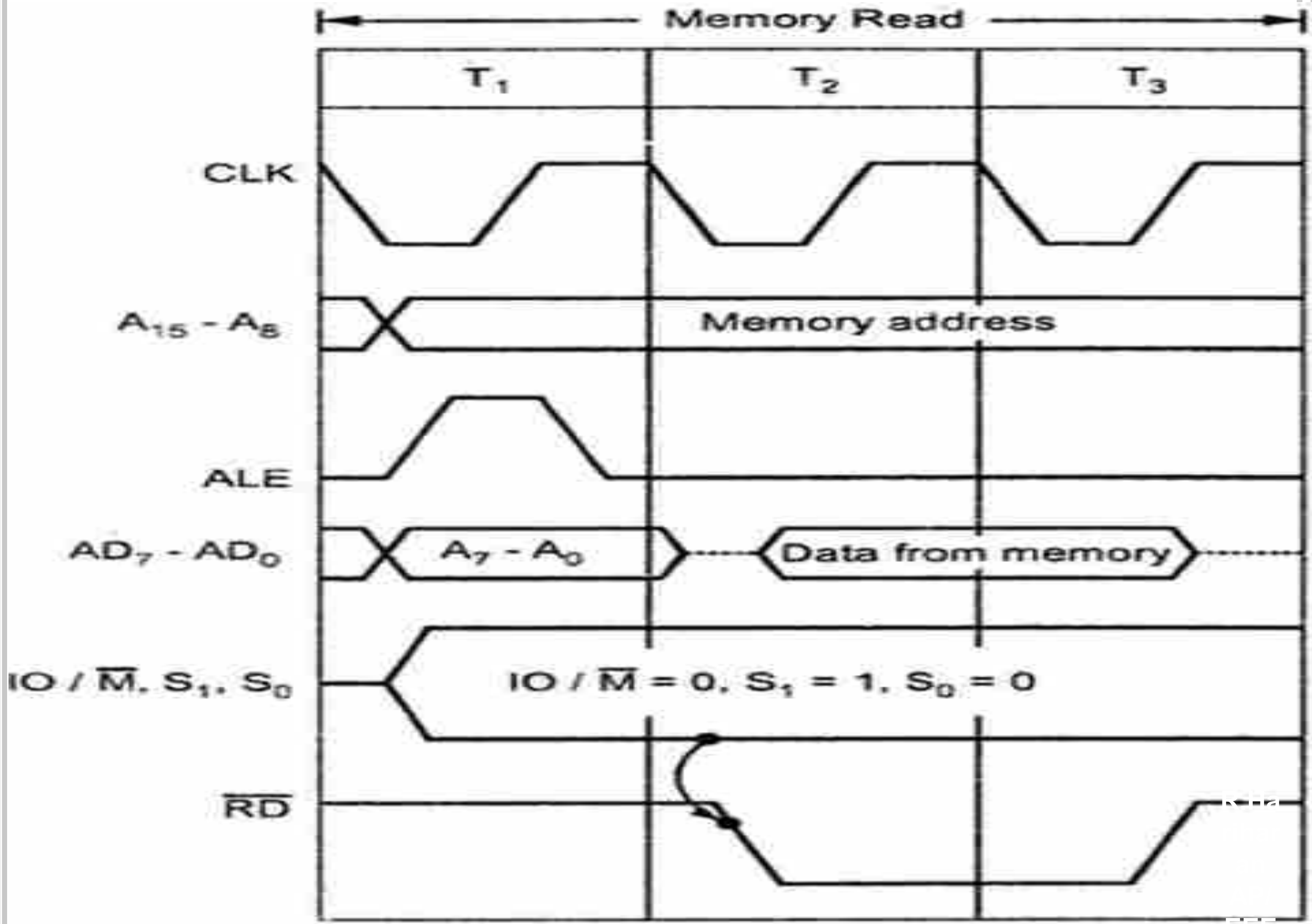
5. IO/M S0,S1 = Group Signal. T1,T2,T3,T4 States.

6. RD,WR = 0. T2,T3 States

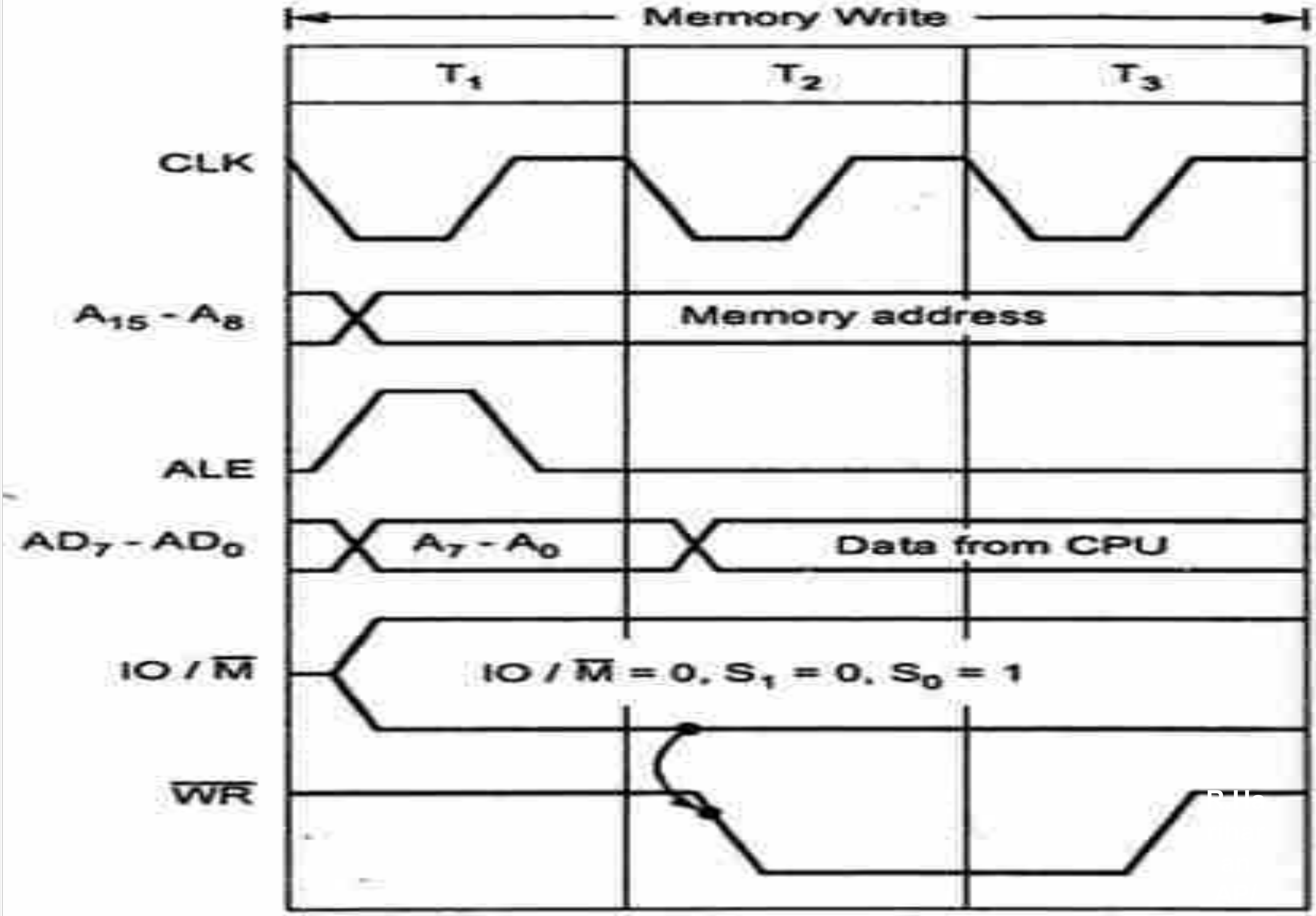


(b) Opcode fetch machine cycle

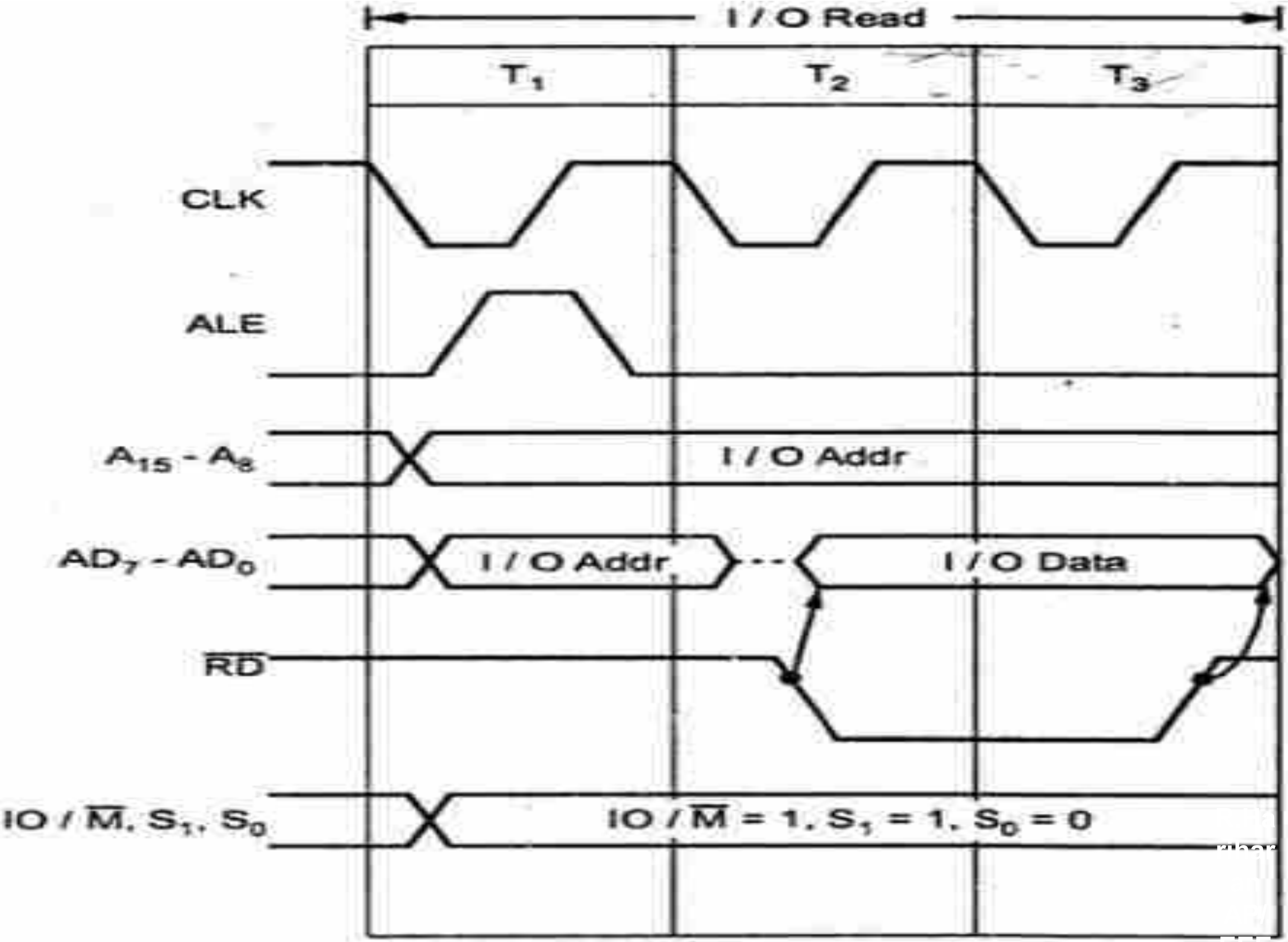
R
i
a
A
E



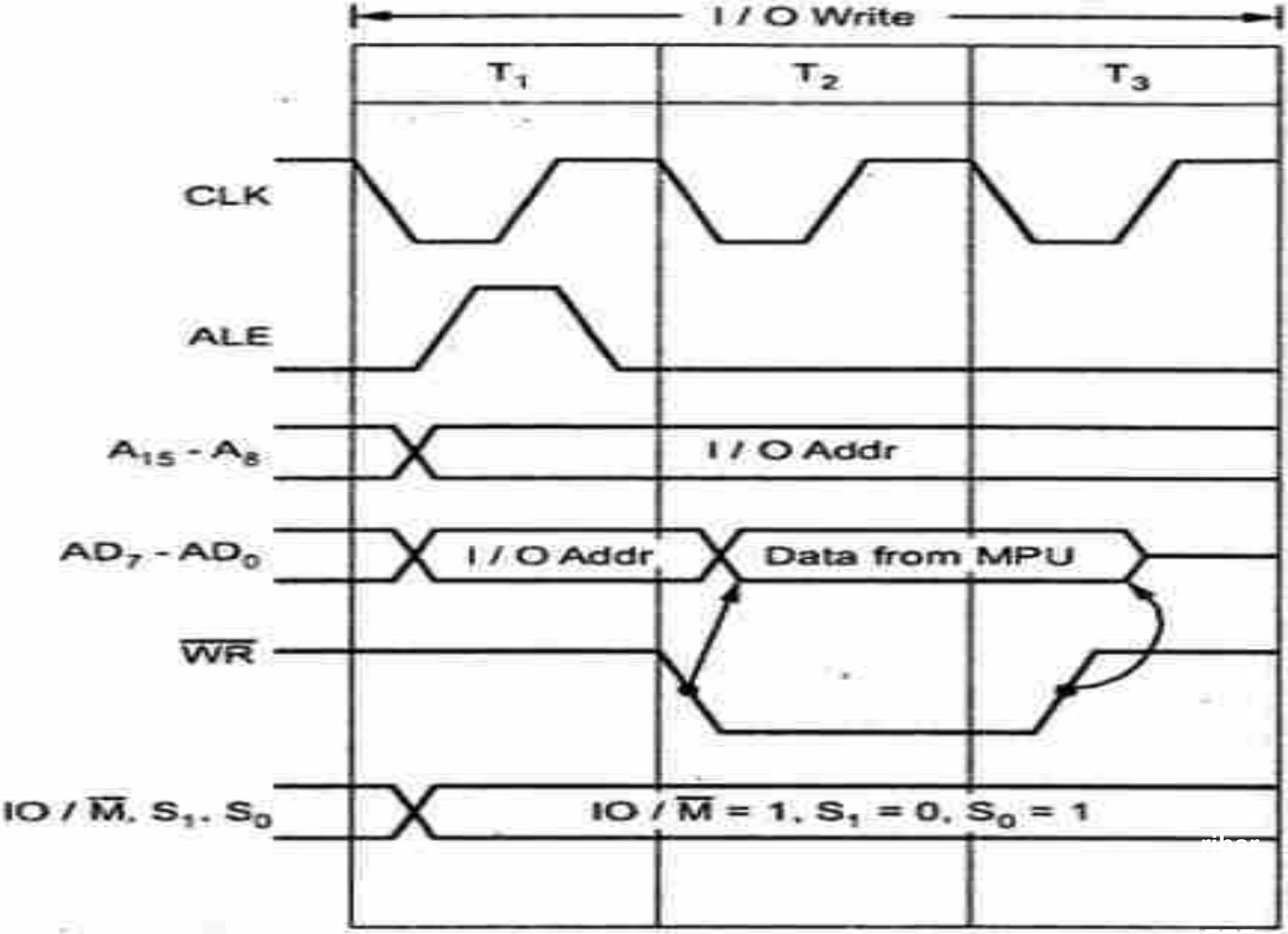
(b) Memory read machine cycle



(b) Memory write machine cycle



(b) I/O read memory cycle



(b) I/O write machine cycle

Addressing Modes

TYPES OF ADDRESSING MODES

- Intel 8085 uses the following addressing modes:
  1. Direct Addressing Mode
  2. Register Addressing Mode
  3. Register Indirect Addressing Mode
  4. Immediate Addressing Mode
  5. Implicit Addressing Mode

Direct Addressing Mode

- In this mode, the address of the operand is given in the instruction itself.
- LDA

LDA 2500 H	Load the contents of memory location 2500 H in accumulator.

- 2500 H is the address of source.
- Accumulator is the destination.

REGISTER ADDRESSING MODE

- In this mode, the operand is in general purpose register.

MOV A, B	**Move the contents of register B to A.**

- MOV is the operation.
- B is the source of data.
- A is the destination.

Register Indirect Addressing Mode

- In this mode, the address of operand is specified by a register pair.

MOV A, M	**Move data from memory location specified by H-L pair to accumulator.**

- MOV is the operation.
- M is the memory location specified by H-L register pair.
- A is the destination.

IMMEDIATE ADDRESSING MODE

- In this mode, the operand is specified within the instruction itself.

MVI A, 05 H	**Move 05 H in accumulator.**

- MVI is the operation.
- 05 H is the immediate data (source).
- A is the destination.

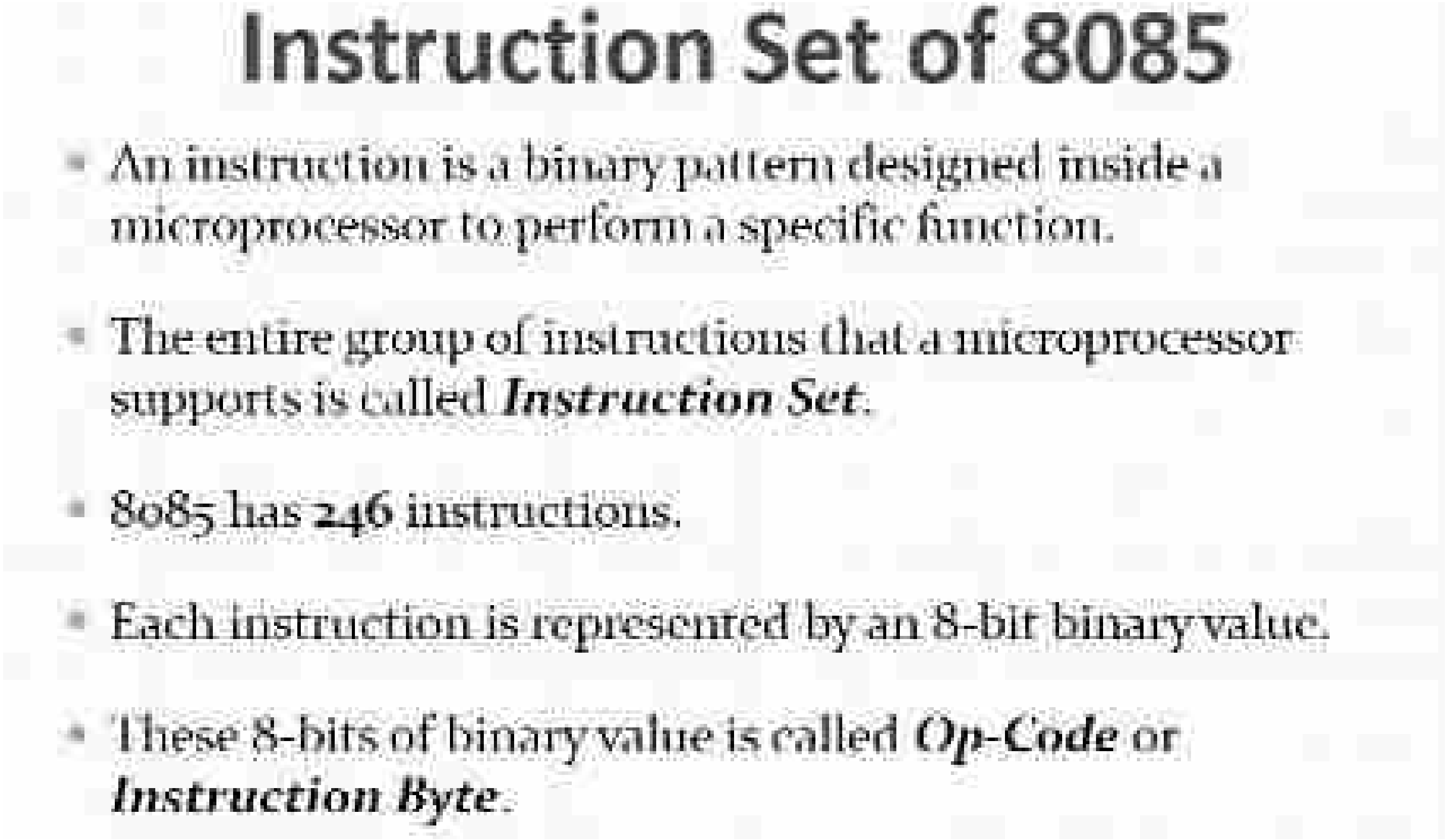
IMPLICIT ADDRESSING MODE

- If address of source of data as well as address of destination of result is fixed, then there is no need to give any operand along with the instruction.

CMA	Complement accumulator.

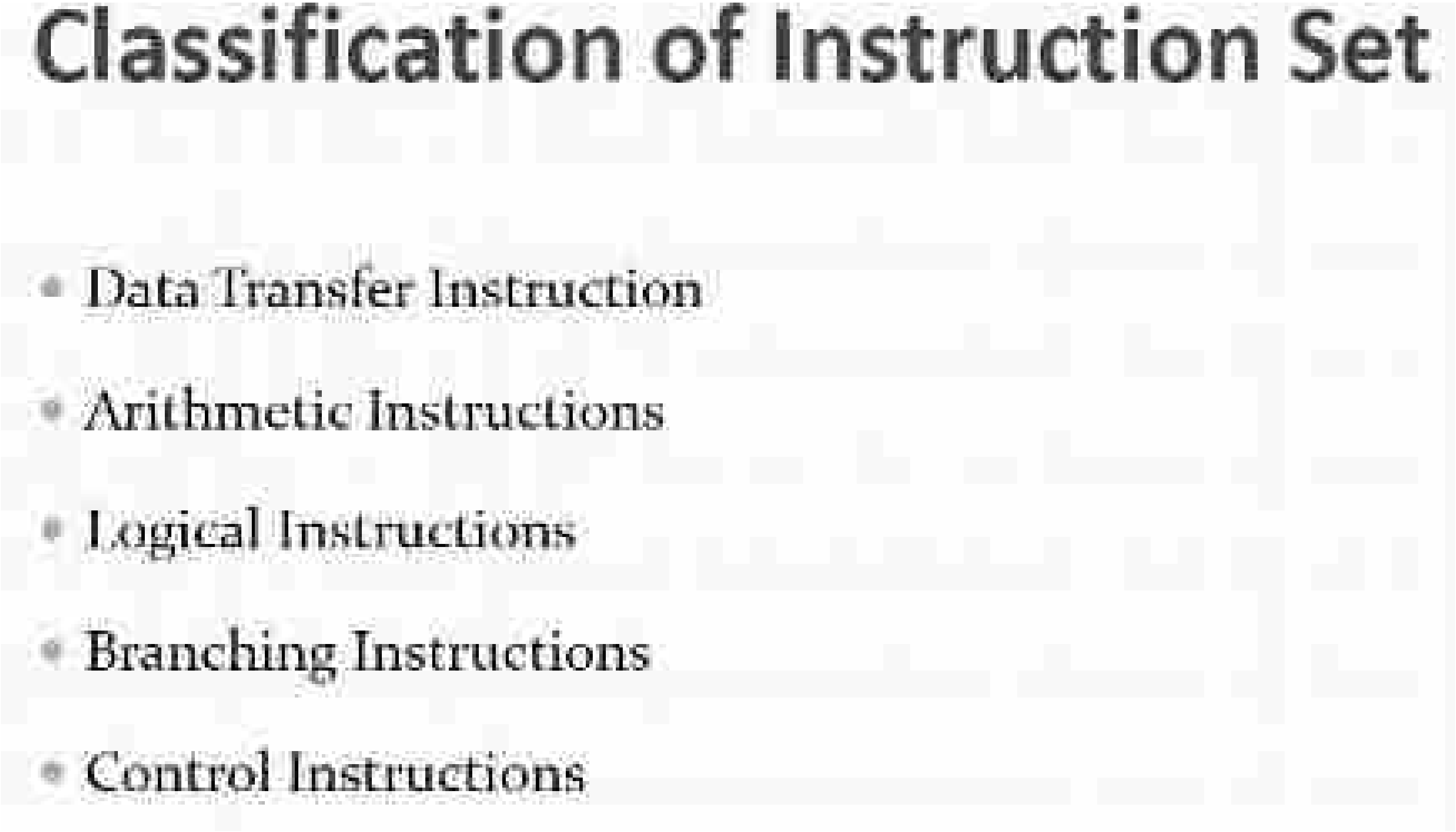
- CMA is the operation.
- A is the source.
- A is the destination.

8085 Instruction Set



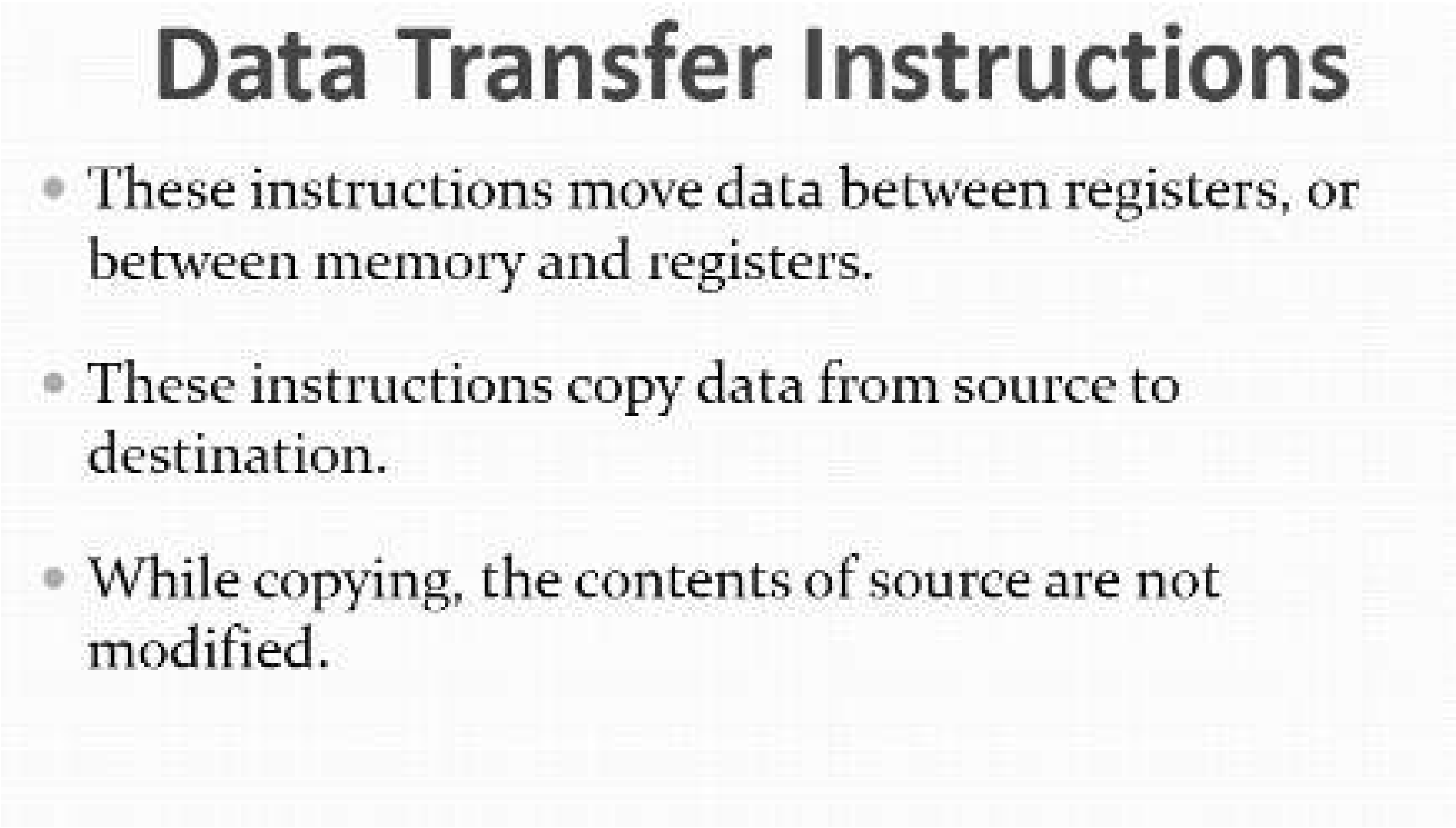
Instruction Set of 8085

- An instruction is a binary pattern designed inside a microprocessor to perform a specific function.
- The entire group of instructions that a microprocessor supports is called ***Instruction Set***.
- 8085 has **246** instructions.
- Each instruction is represented by an 8-bit binary value.
- These 8-bits of binary value is called ***Op-Code*** or ***Instruction Byte***.



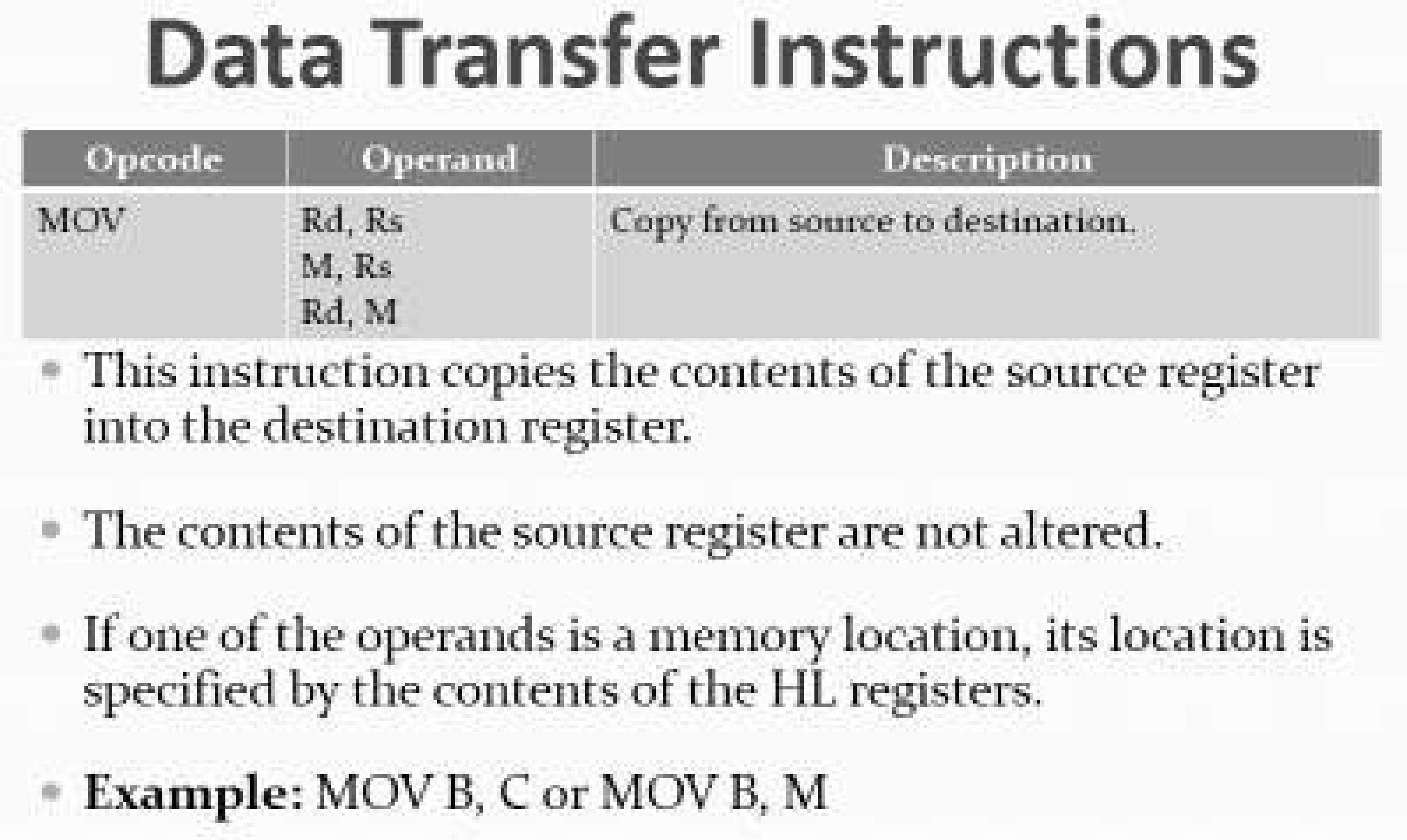
Classification of Instruction Set

- Data Transfer Instruction
- Arithmetic Instructions
- Logical Instructions
- Branching Instructions
- Control Instructions



Data Transfer Instructions

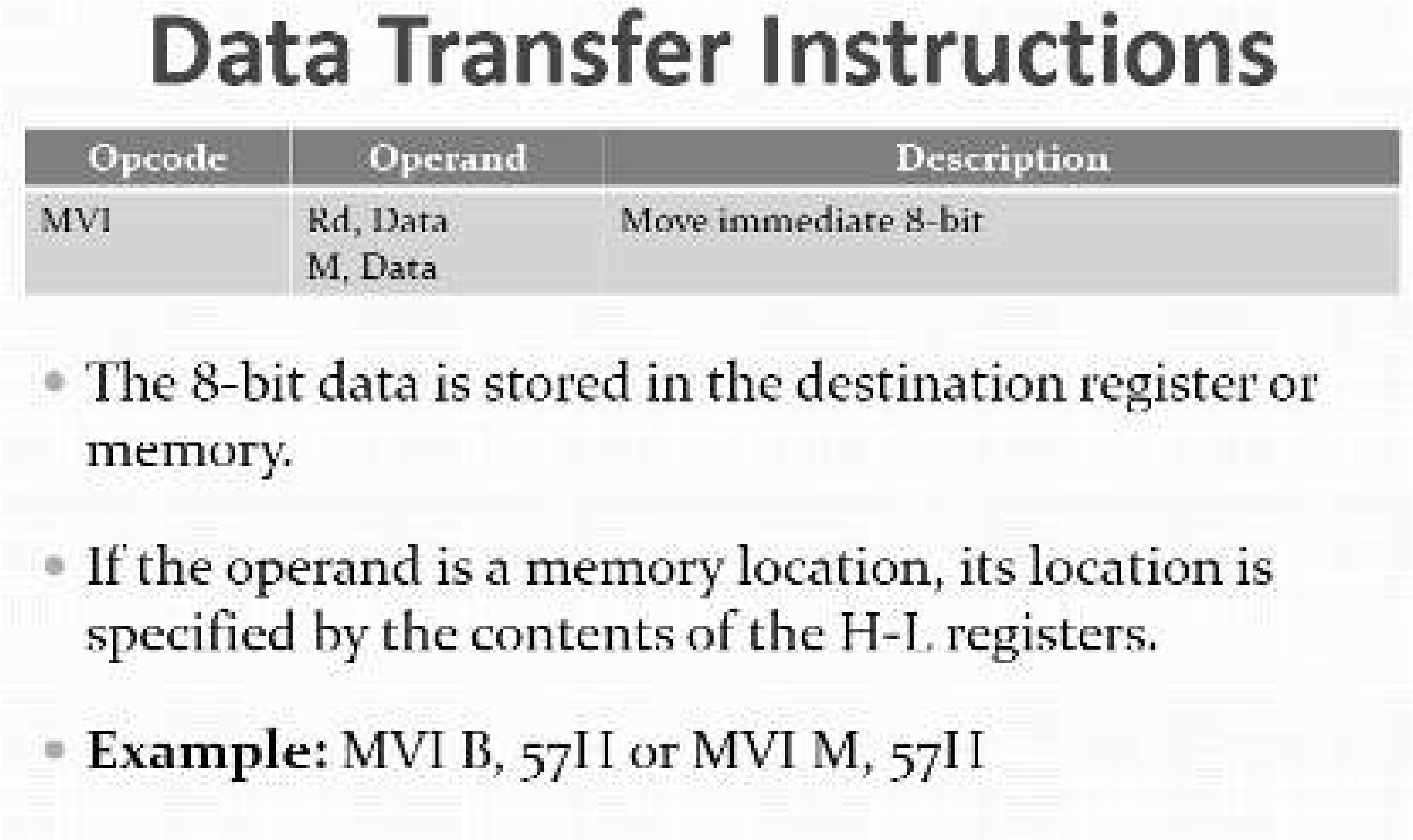
- These instructions move data between registers, or between memory and registers.
- These instructions copy data from source to destination.
- While copying, the contents of source are not modified.



Data Transfer Instructions

Opcode	Operand	Description
MOV	Rd, Rs M, Rs Rd, M	Copy from source to destination.

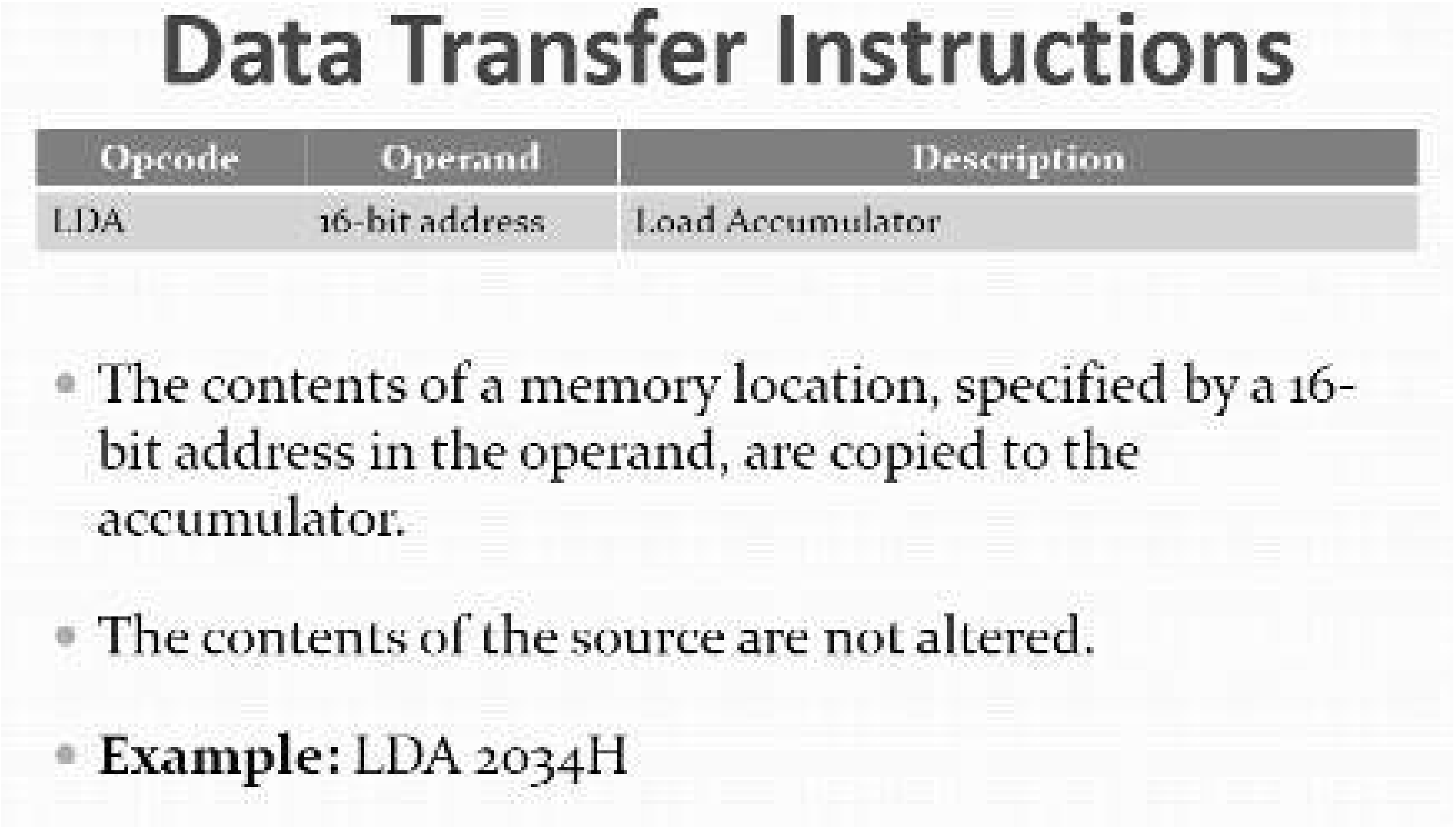
- This instruction copies the contents of the source register into the destination register.
- The contents of the source register are not altered.
- If one of the operands is a memory location, its location is specified by the contents of the HL registers.
- **Example:** MOV B, C or MOV B, M



Data Transfer Instructions

Opcode	Operand	Description
MVI	Rd, Data M, Data	Move immediate 8-bit

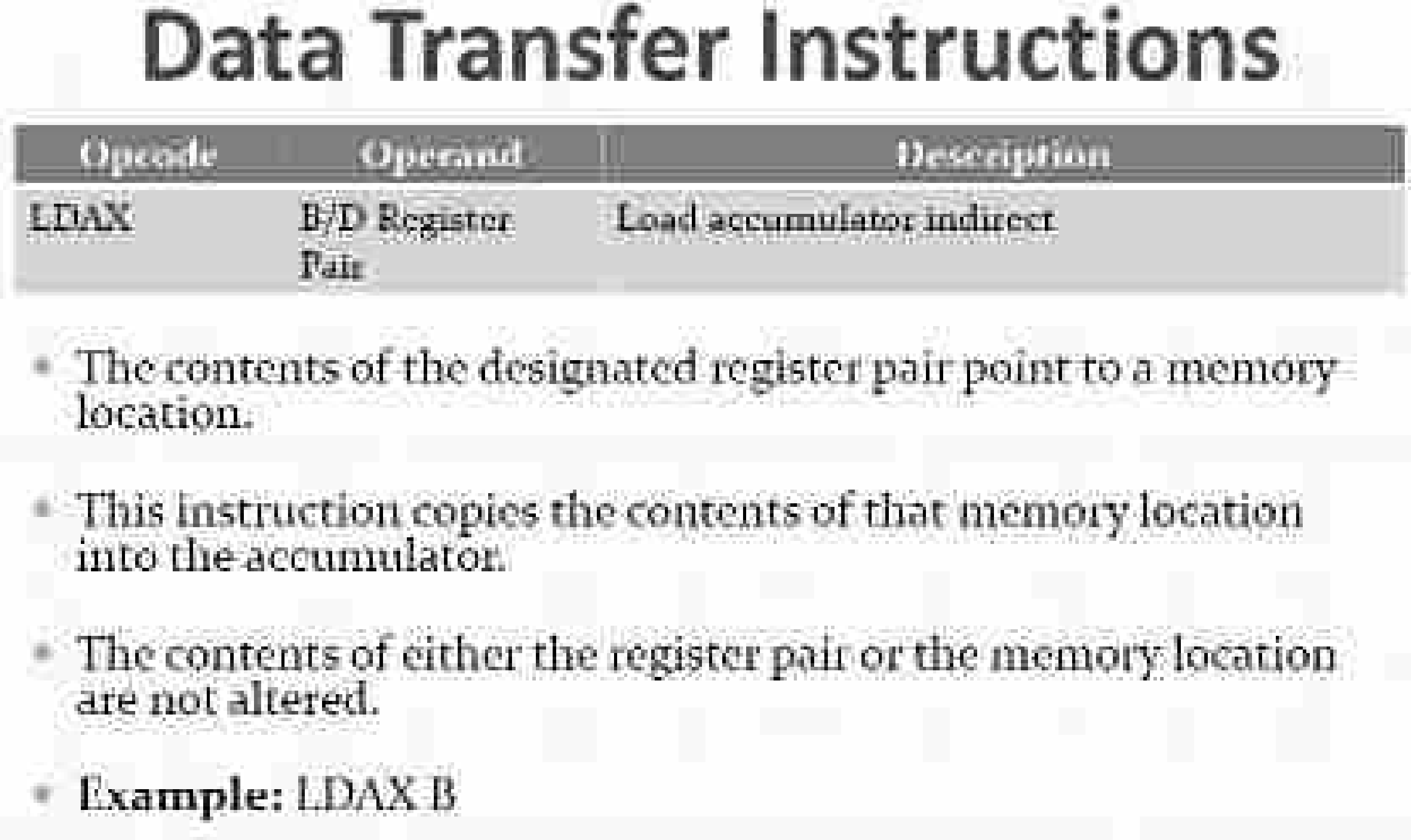
- The 8-bit data is stored in the destination register or memory.
- If the operand is a memory location, its location is specified by the contents of the H-L registers.
- **Example:** MVI B, 57H or MVI M, 57H



Data Transfer Instructions

Opcode	Operand	Description
LDA	16-bit address	Load Accumulator

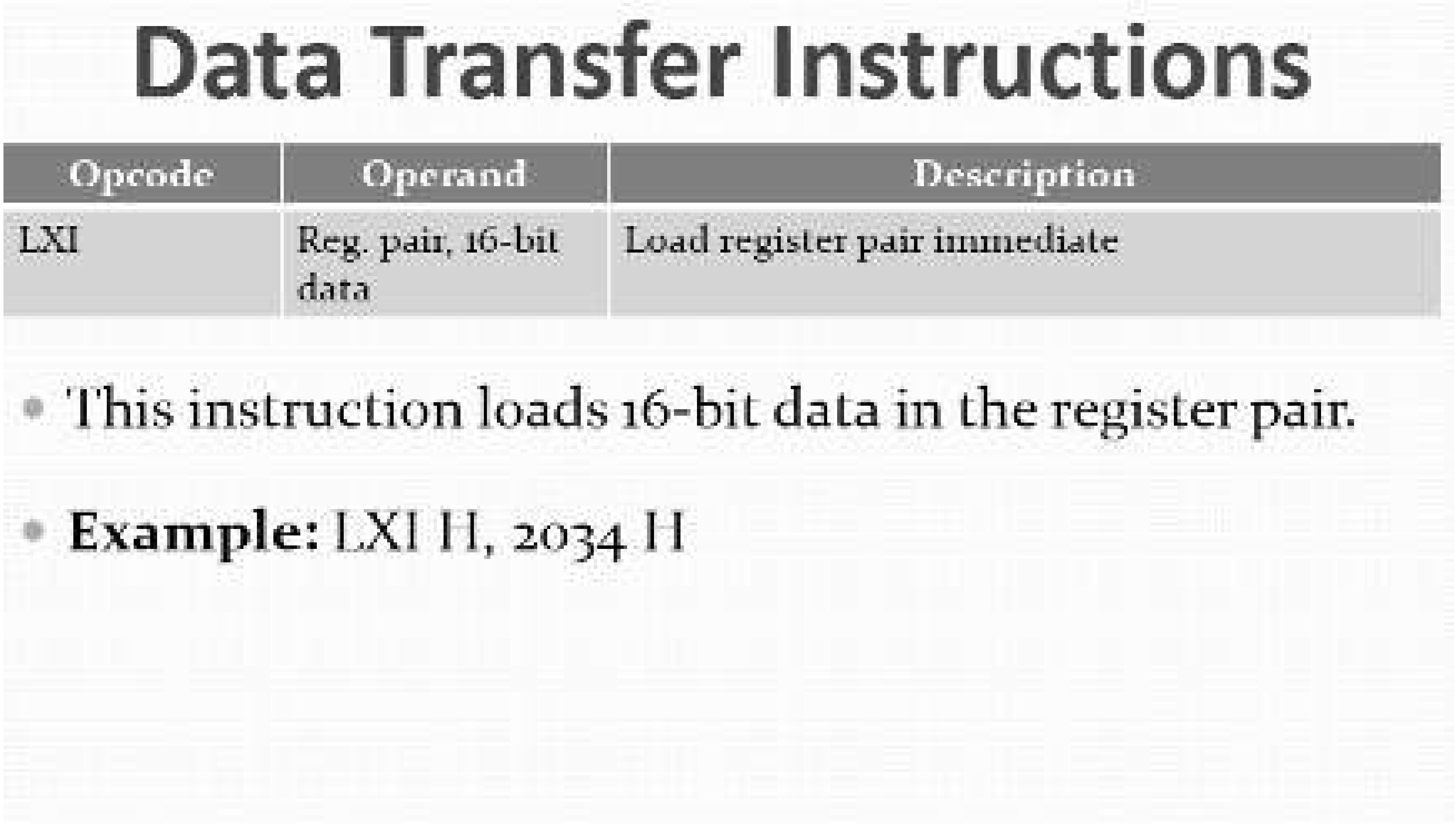
- The contents of a memory location, specified by a 16-bit address in the operand, are copied to the accumulator.
- The contents of the source are not altered.
- **Example:** LDA 2034H



Data Transfer Instructions

Opcode	Operand	Description
LDAX	B/D Register Pair	Load accumulator indirect

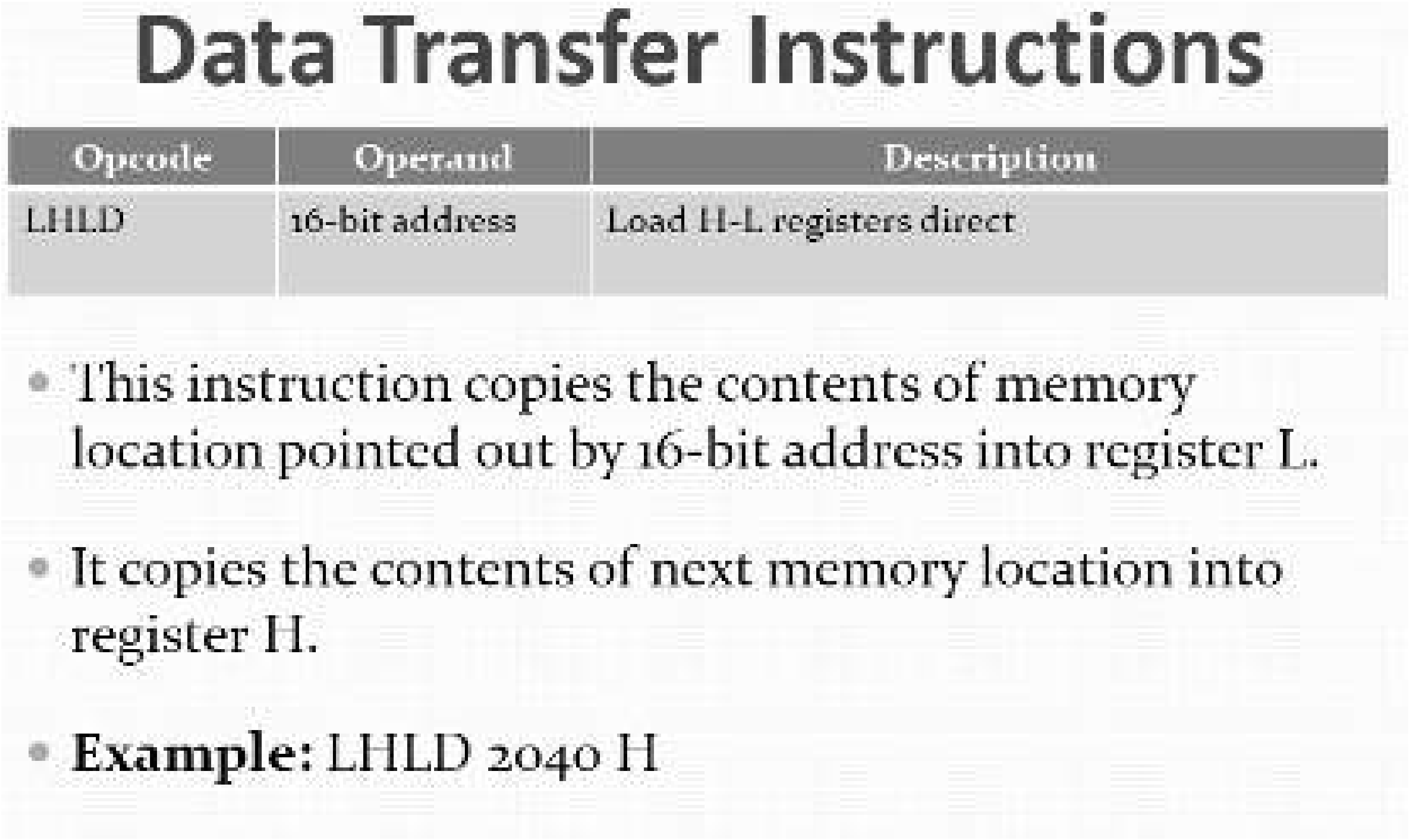
- The contents of the designated register pair point to a memory location.
- This instruction copies the contents of that memory location into the accumulator.
- The contents of either the register pair or the memory location are not altered.
- **Example:** LDAX B



Data Transfer Instructions

Opcode	Operand	Description
LXI	Reg. pair, 16-bit data	Load register pair immediate

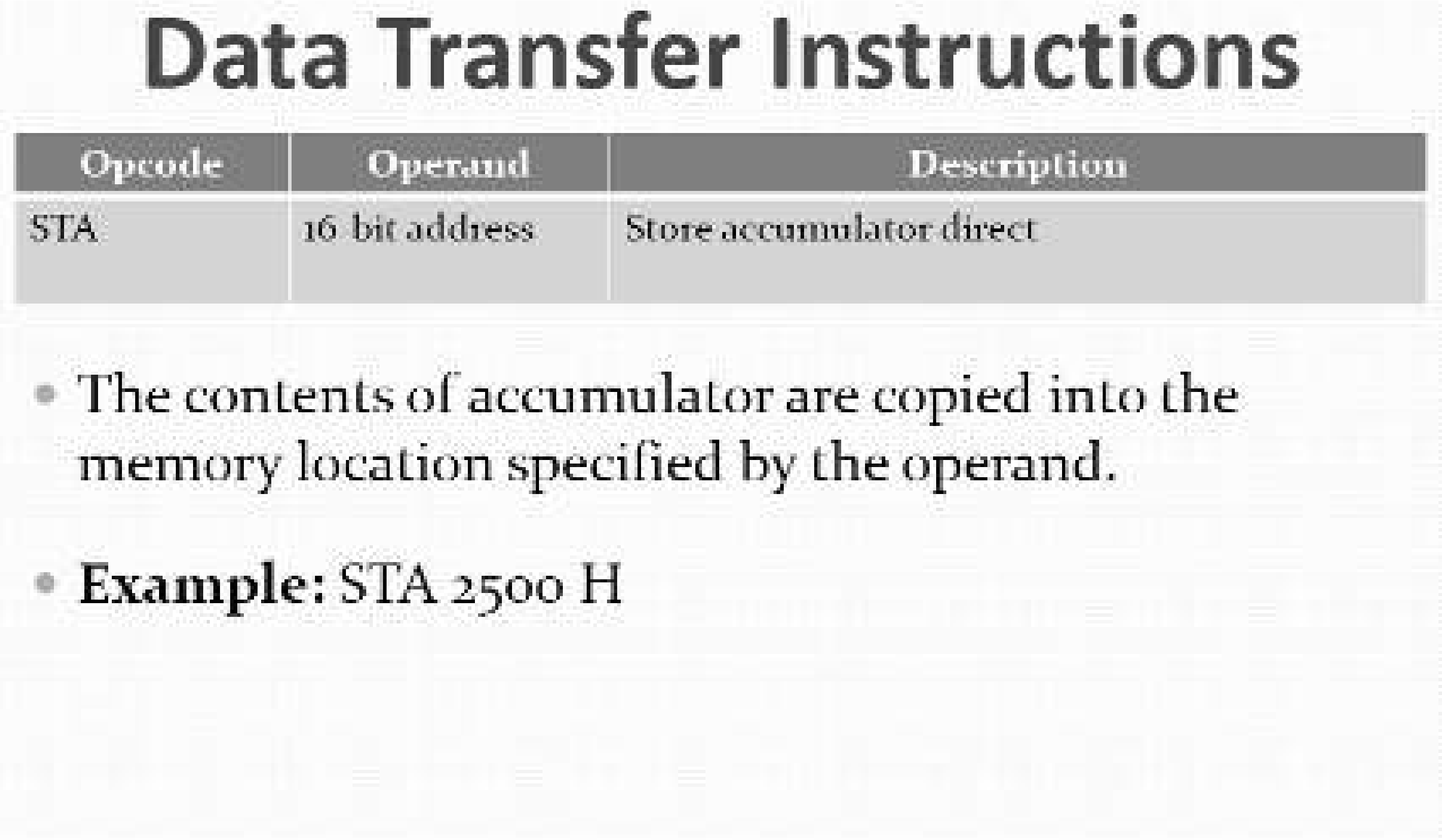
- This instruction loads 16-bit data in the register pair.
- **Example:** LXI H, 2034 H



Data Transfer Instructions

Opcode	Operand	Description
LHLD	16-bit address	Load H-L registers direct

- This instruction copies the contents of memory location pointed out by 16-bit address into register L.
- It copies the contents of next memory location into register H.
- **Example:** LHLD 2040 H

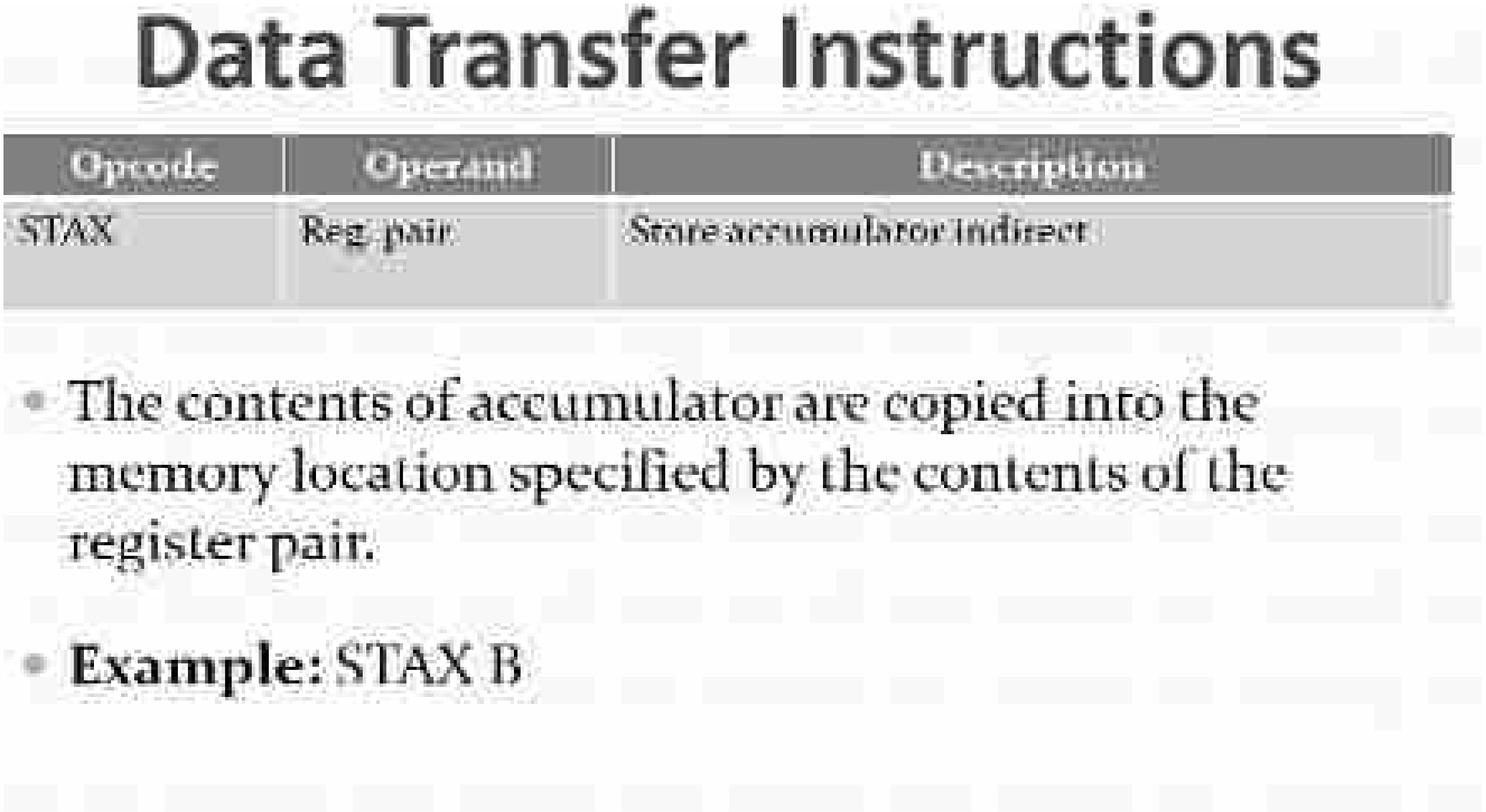


Data Transfer Instructions

Opcode	Operand	Description
STA	16 bit address	Store accumulator direct

- The contents of accumulator are copied into the memory location specified by the operand.

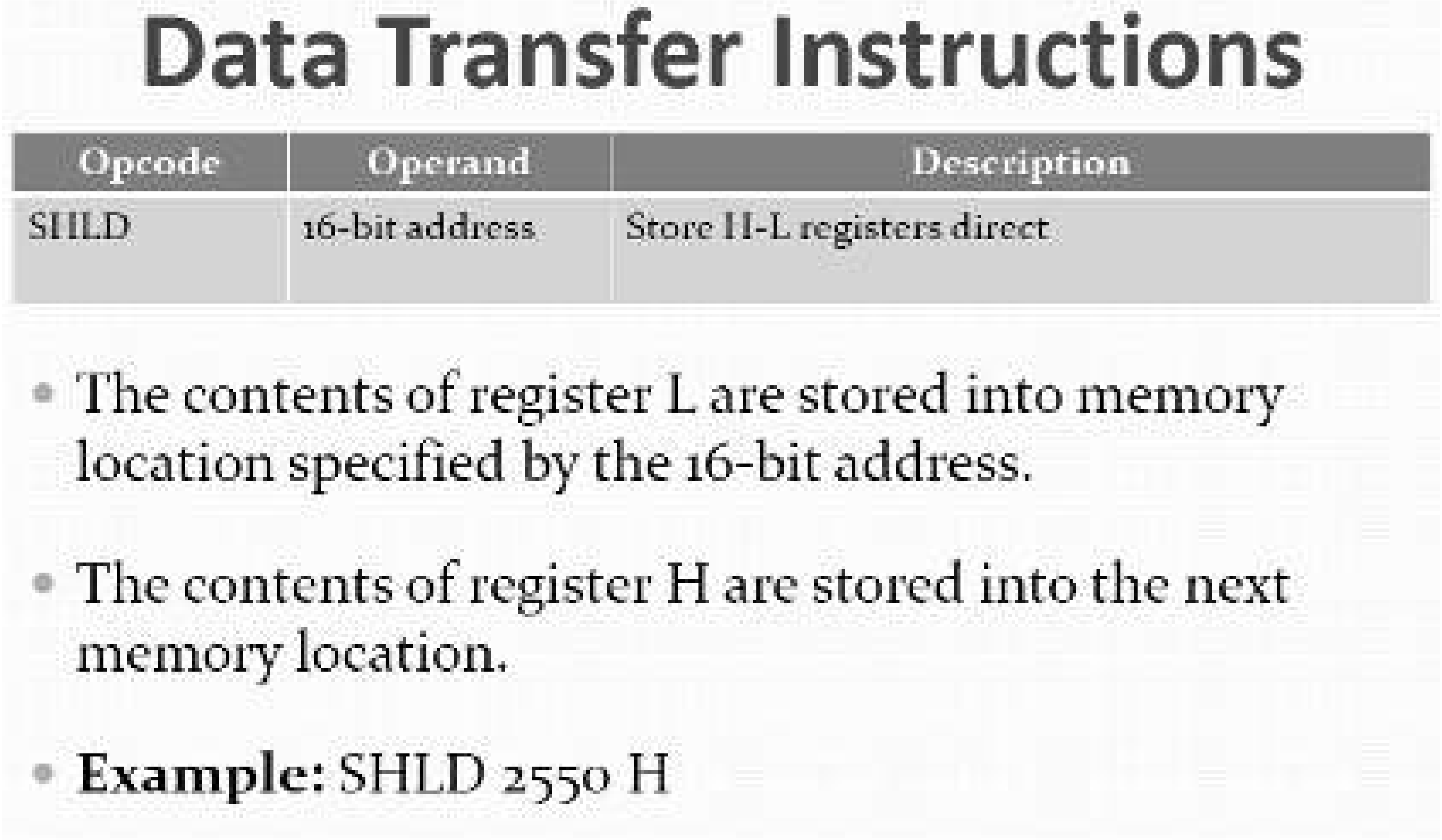
- **Example:** STA 2500 H



Data Transfer Instructions

Opcode	Operand	Description
STAX	Reg. pair	Store accumulator indirect

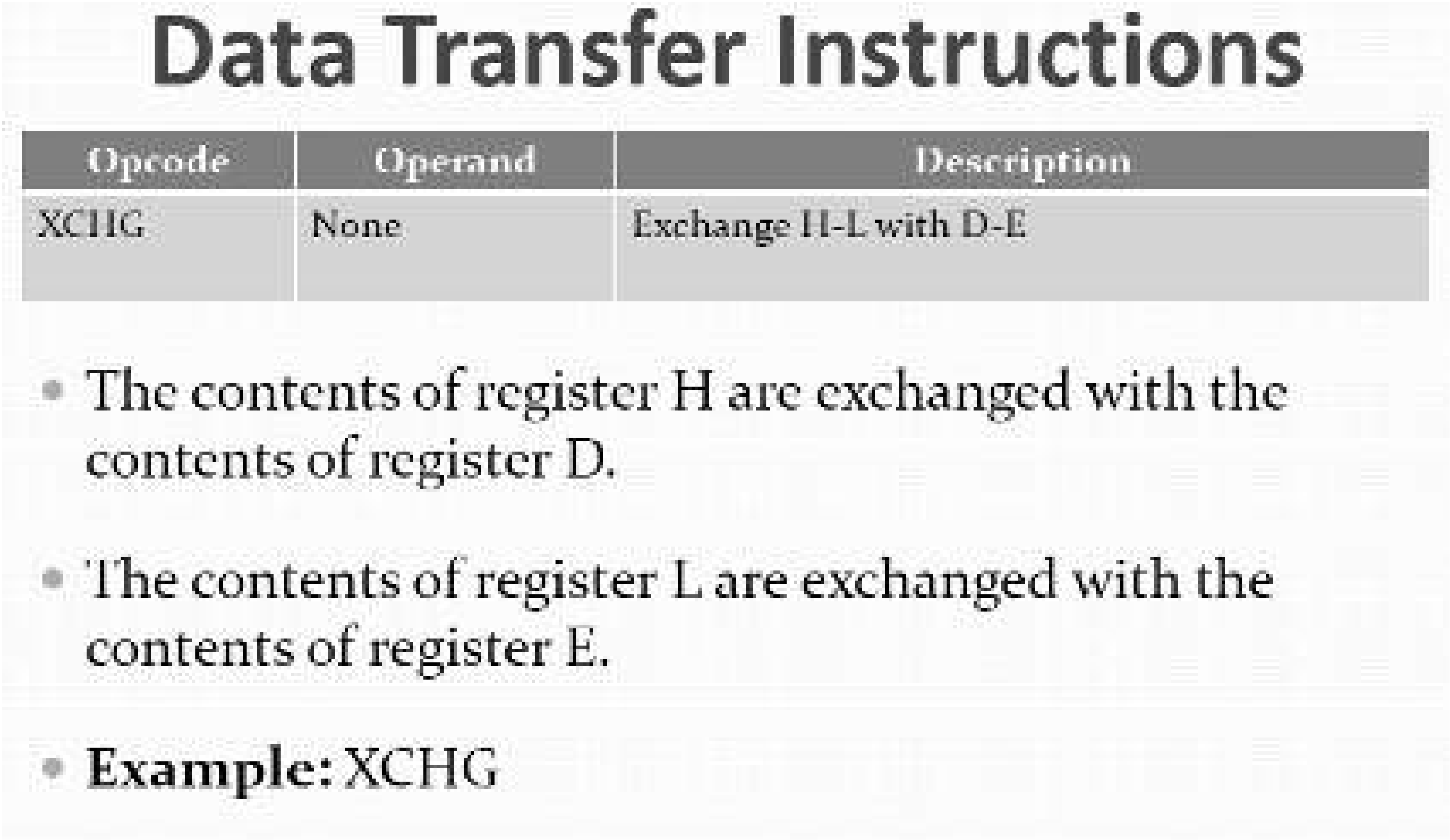
- The contents of accumulator are copied into the memory location specified by the contents of the register pair.
- **Example:** STAX B



Data Transfer Instructions

Opcode	Operand	Description
SHLD	16-bit address	Store H-L registers direct

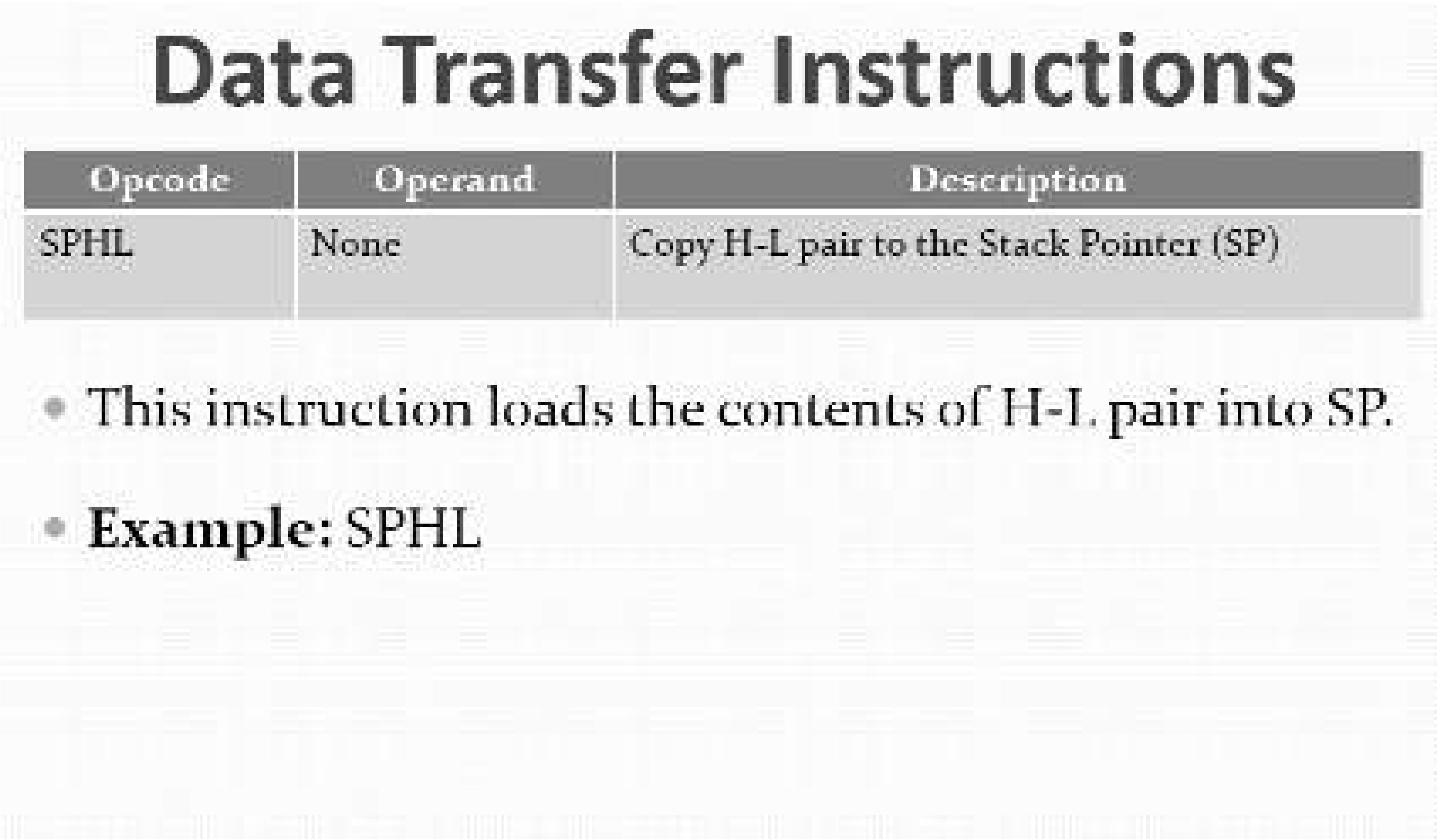
- The contents of register L are stored into memory location specified by the 16-bit address.
- The contents of register H are stored into the next memory location.
- **Example:** SHLD 2550 H



Data Transfer Instructions

Opcode	Operand	Description
XCHG	None	Exchange H-L with D-E

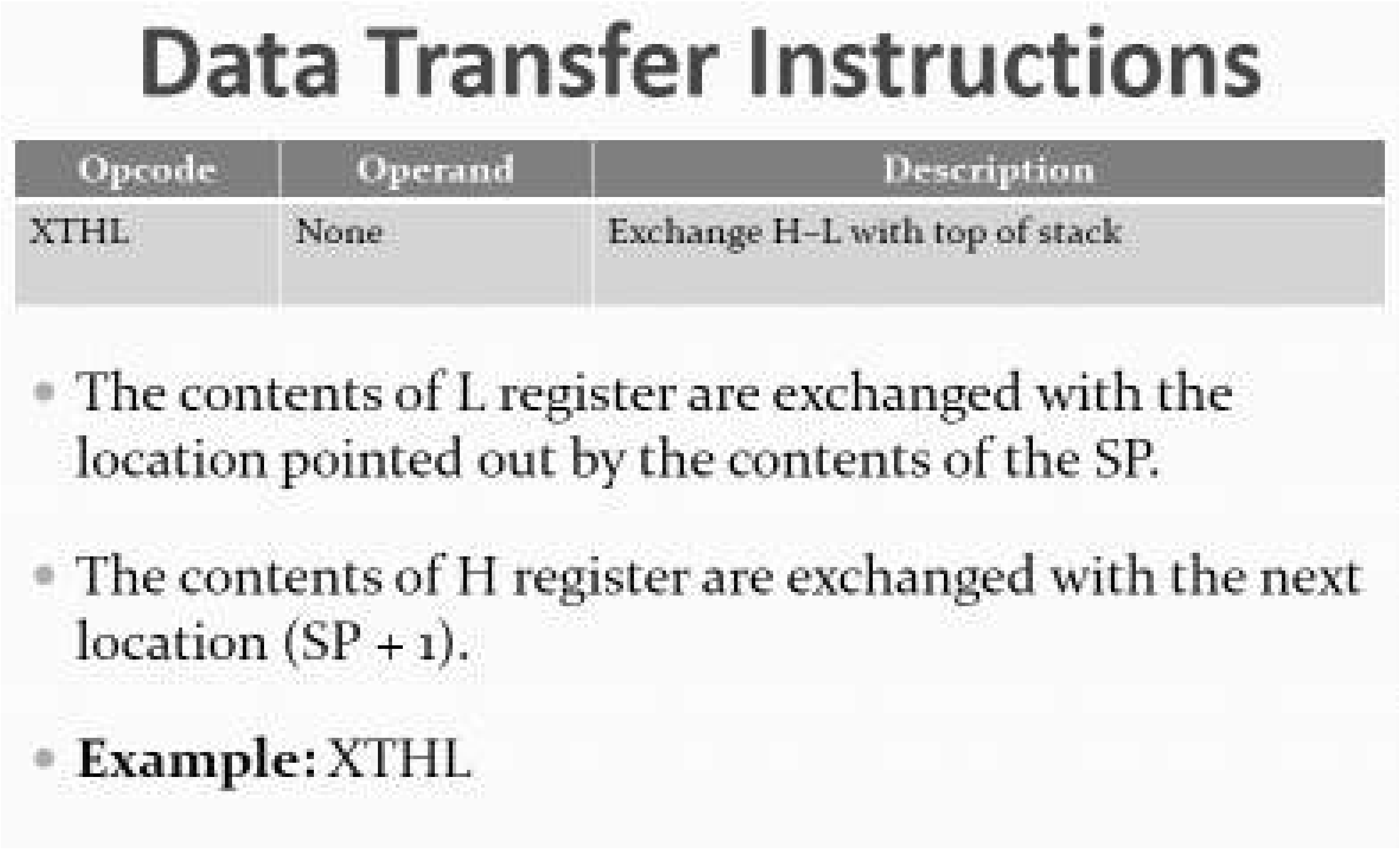
- The contents of register H are exchanged with the contents of register D.
- The contents of register L are exchanged with the contents of register E.
- **Example:** XCHG



Data Transfer Instructions

Opcode	Operand	Description
SPHL	None	Copy H-L pair to the Stack Pointer (SP)

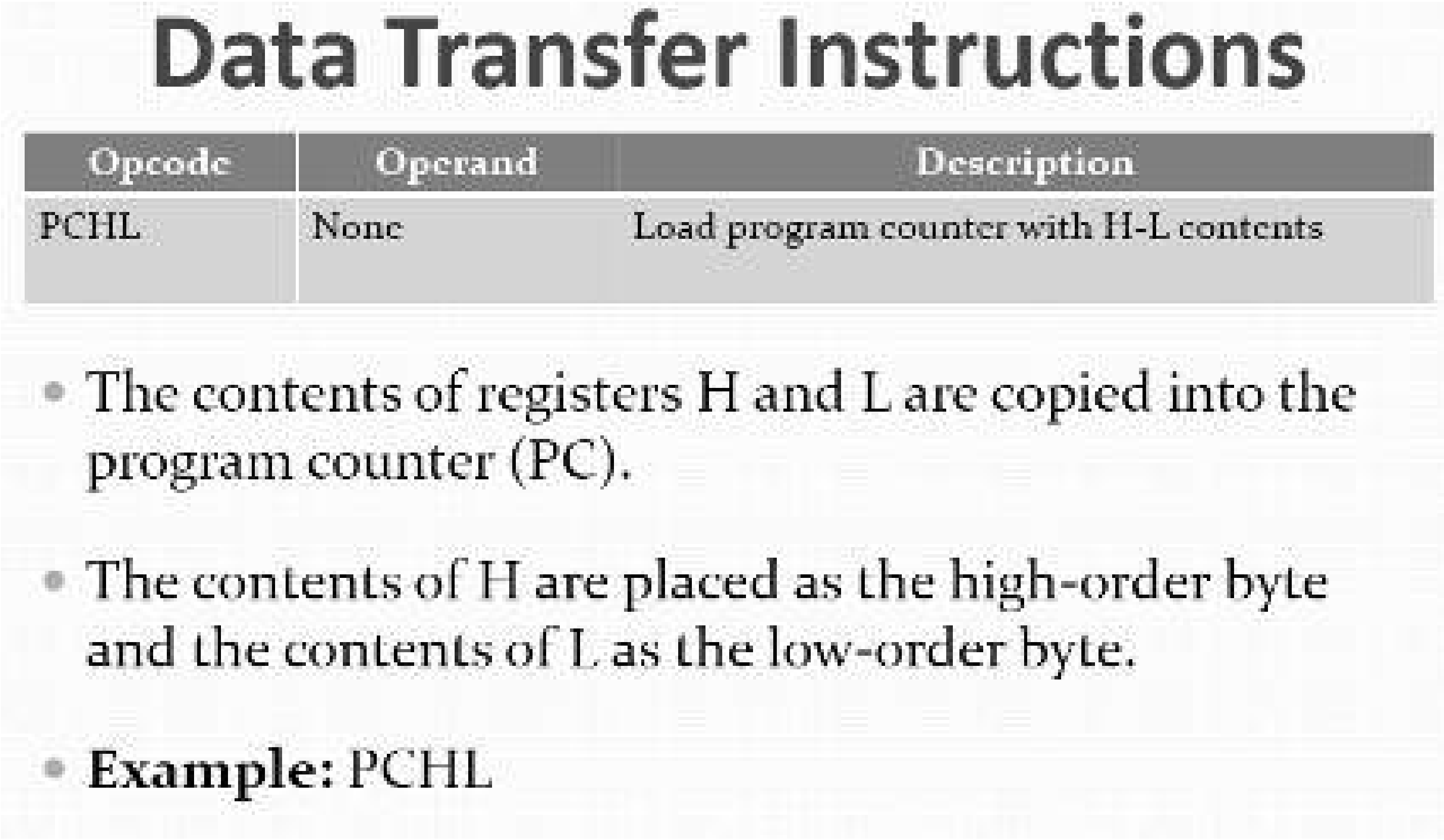
- This instruction loads the contents of H-L pair into SP.
- **Example:** SPHL



Data Transfer Instructions

Opcode	Operand	Description
XTHL	None	Exchange H–L with top of stack

- The contents of L register are exchanged with the location pointed out by the contents of the SP.
- The contents of H register are exchanged with the next location (SP + 1).
- **Example:** XTHL



Data Transfer Instructions

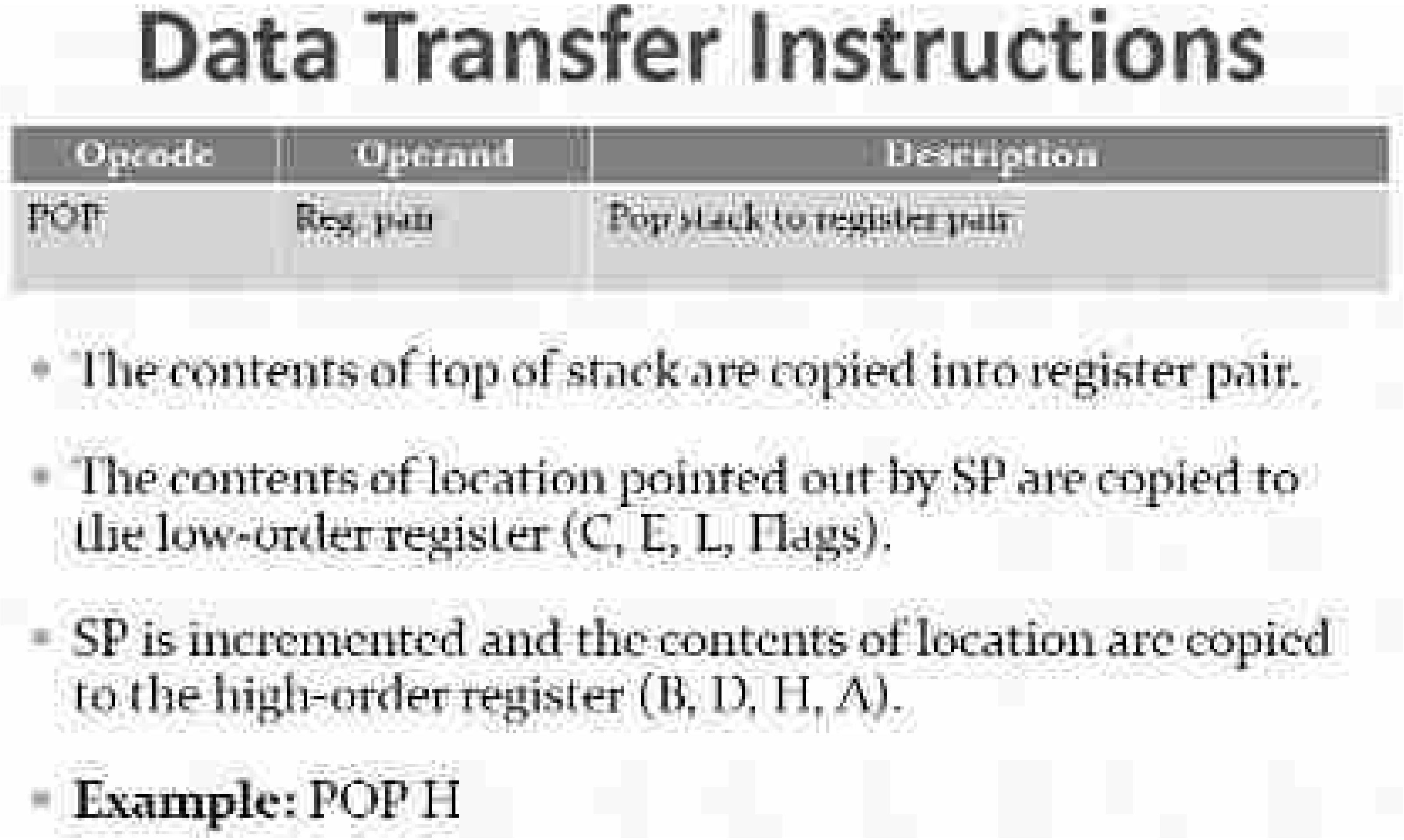
Opcode	Operand	Description
PCHL	None	Load program counter with H-L contents

- The contents of registers H and L are copied into the program counter (PC).
- The contents of H are placed as the high-order byte and the contents of L as the low-order byte.
- **Example:** PCHL

Data Transfer Instructions

Opcode	Operand	Description
PUSH	Reg. pair	Push register pair onto stack

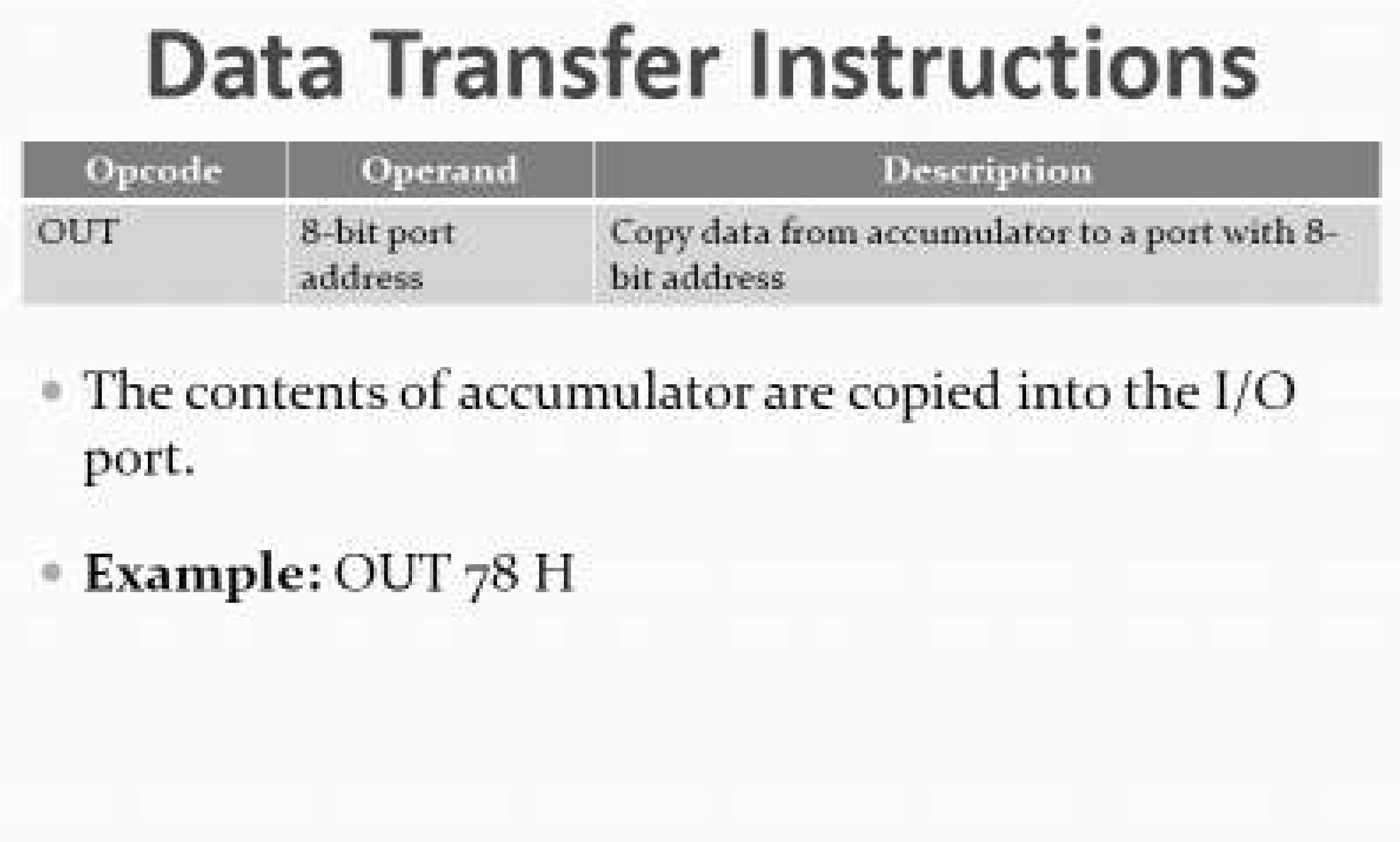
- The contents of register pair are copied onto stack.
- SP is decremented and the contents of high-order registers (B, D, H, A) are copied into stack.
- SP is again decremented and the contents of low-order registers (C, E, L, Flags) are copied into stack.
- **Example:** PUSH B



Data Transfer Instructions

Opcode	Operand	Description
POP	Reg. pair	Pop stack to register pair

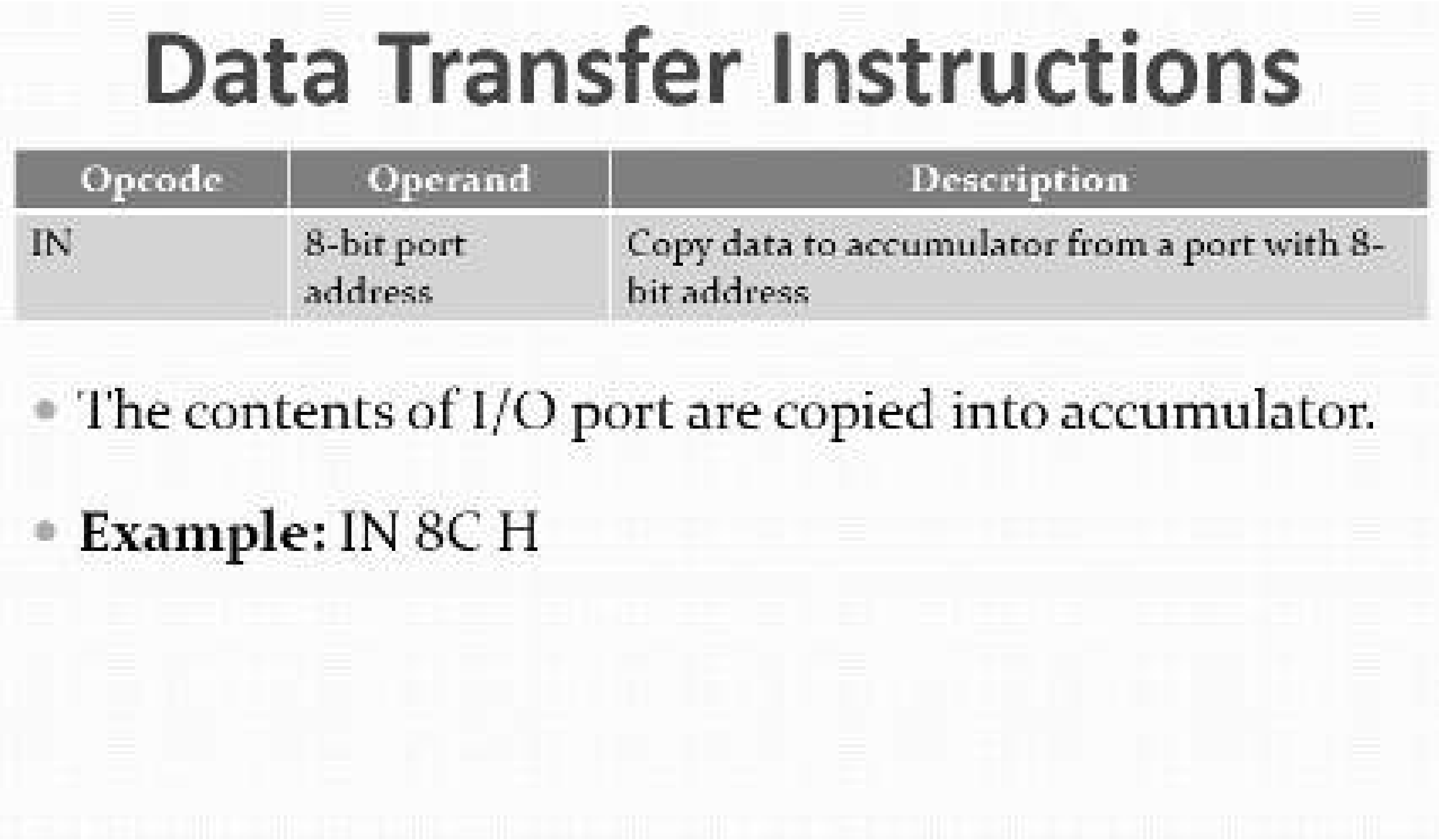
- The contents of top of stack are copied into register pair.
- The contents of location pointed out by SP are copied to the low-order register (C, E, L, Flags).
- SP is incremented and the contents of location are copied to the high-order register (B, D, H, A).
- **Example:** POP H



Data Transfer Instructions

Opcode	Operand	Description
OUT	8-bit port address	Copy data from accumulator to a port with 8-bit address

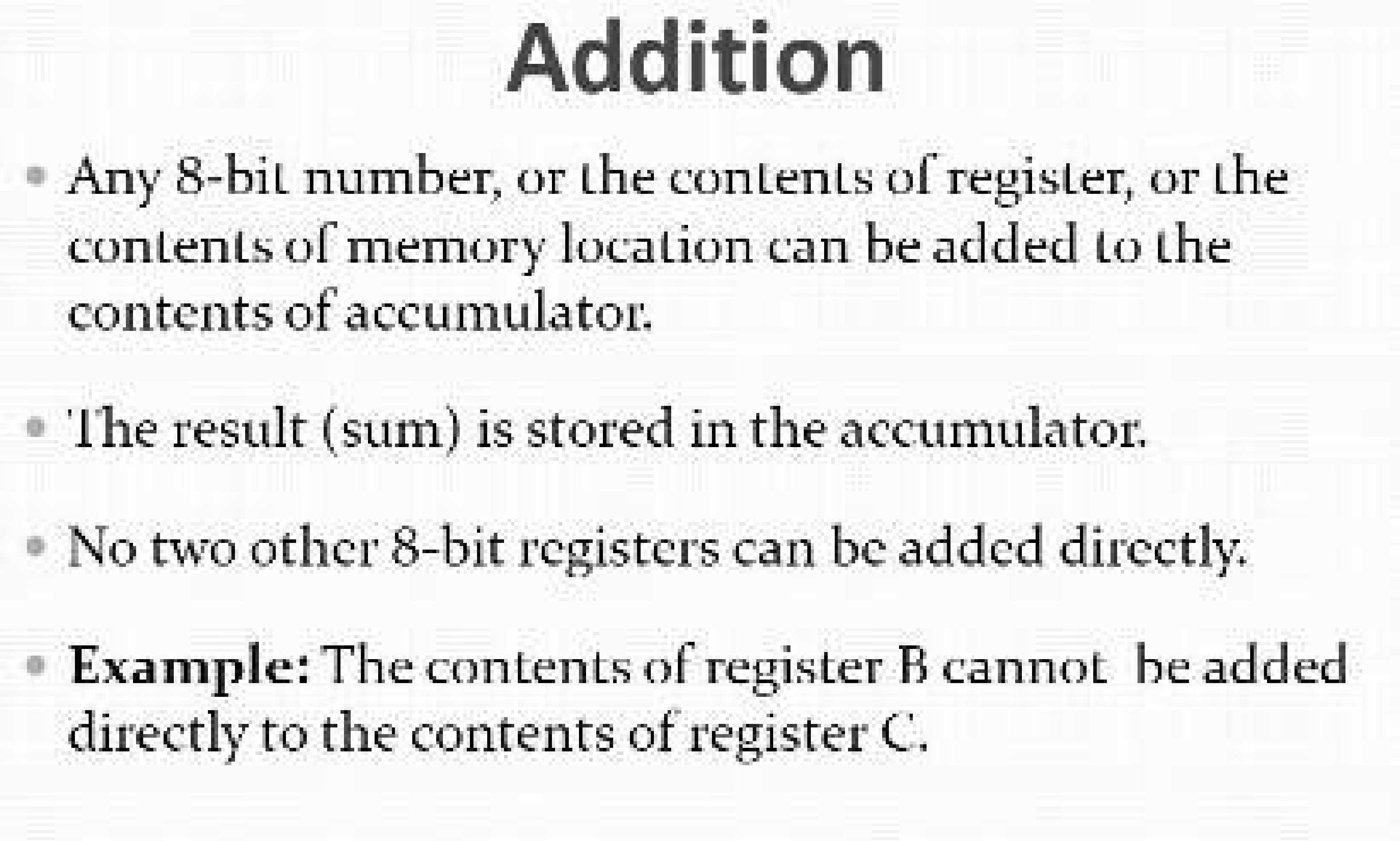
- The contents of accumulator are copied into the I/O port.
- **Example:** OUT 78 H



Data Transfer Instructions

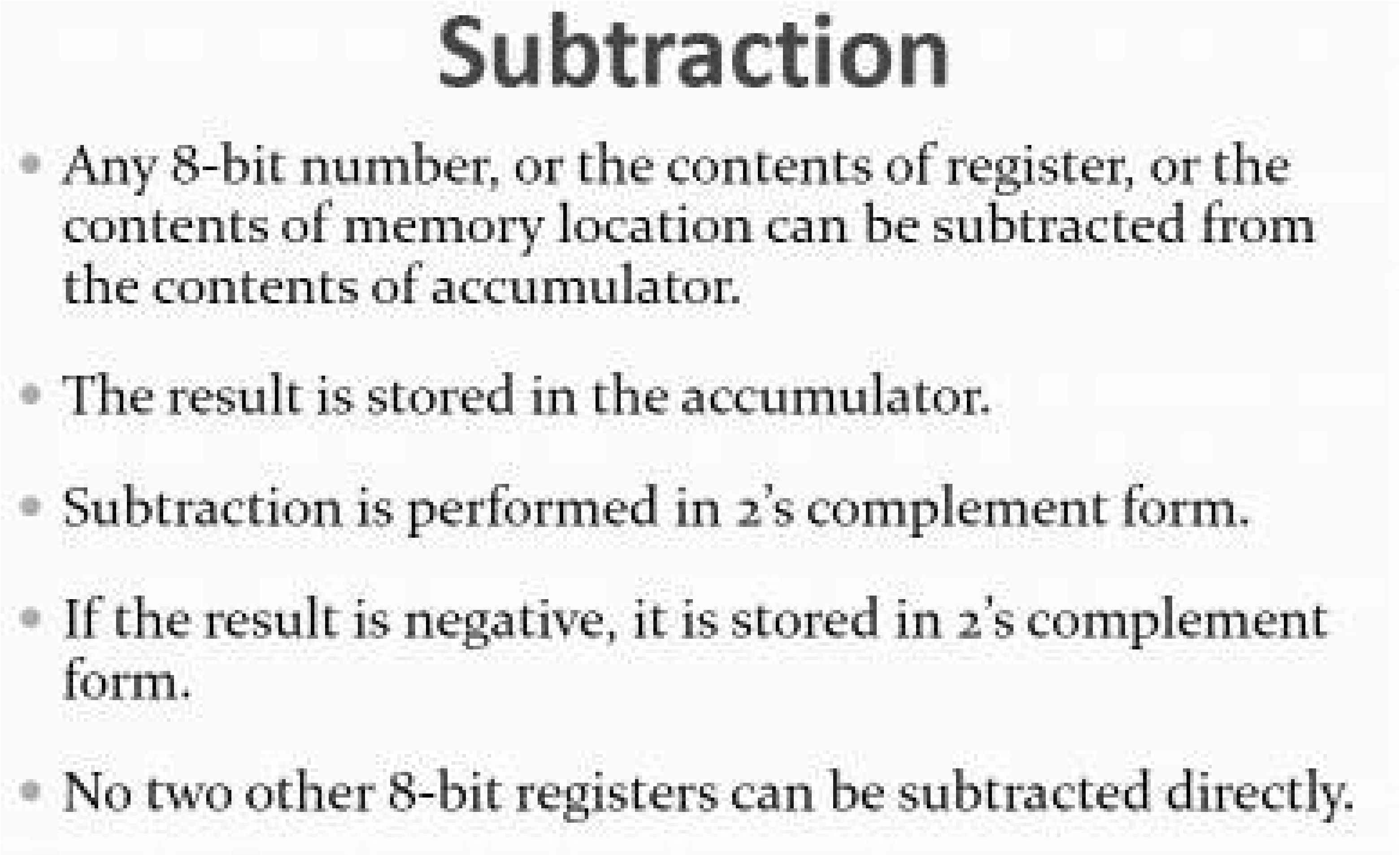
Opcode	Operand	Description
IN	8-bit port address	Copy data to accumulator from a port with 8-bit address

- The contents of I/O port are copied into accumulator.
- **Example:** IN 8C H



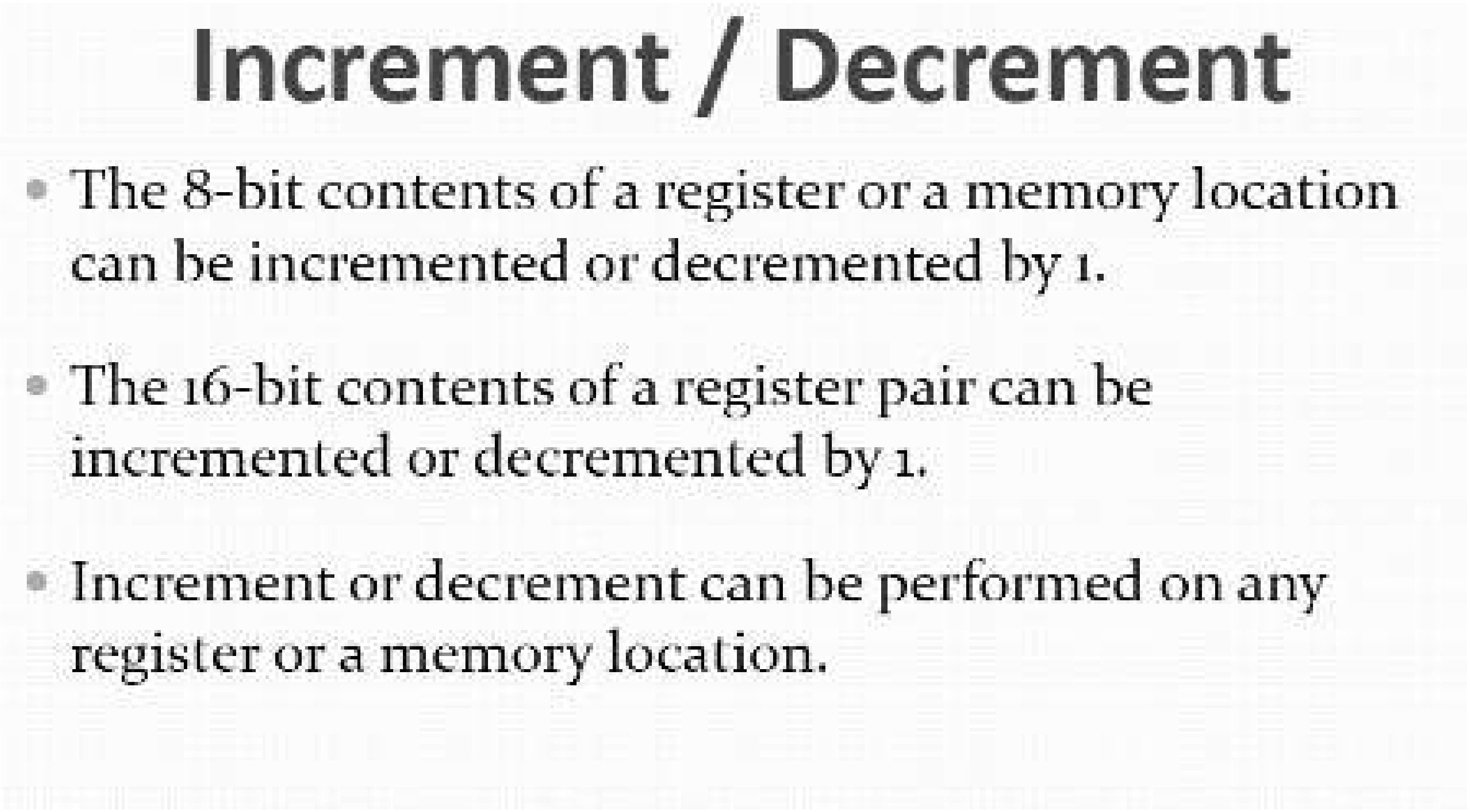
Addition

- Any 8-bit number, or the contents of register, or the contents of memory location can be added to the contents of accumulator.
- The result (sum) is stored in the accumulator.
- No two other 8-bit registers can be added directly.
- **Example:** The contents of register B cannot be added directly to the contents of register C.



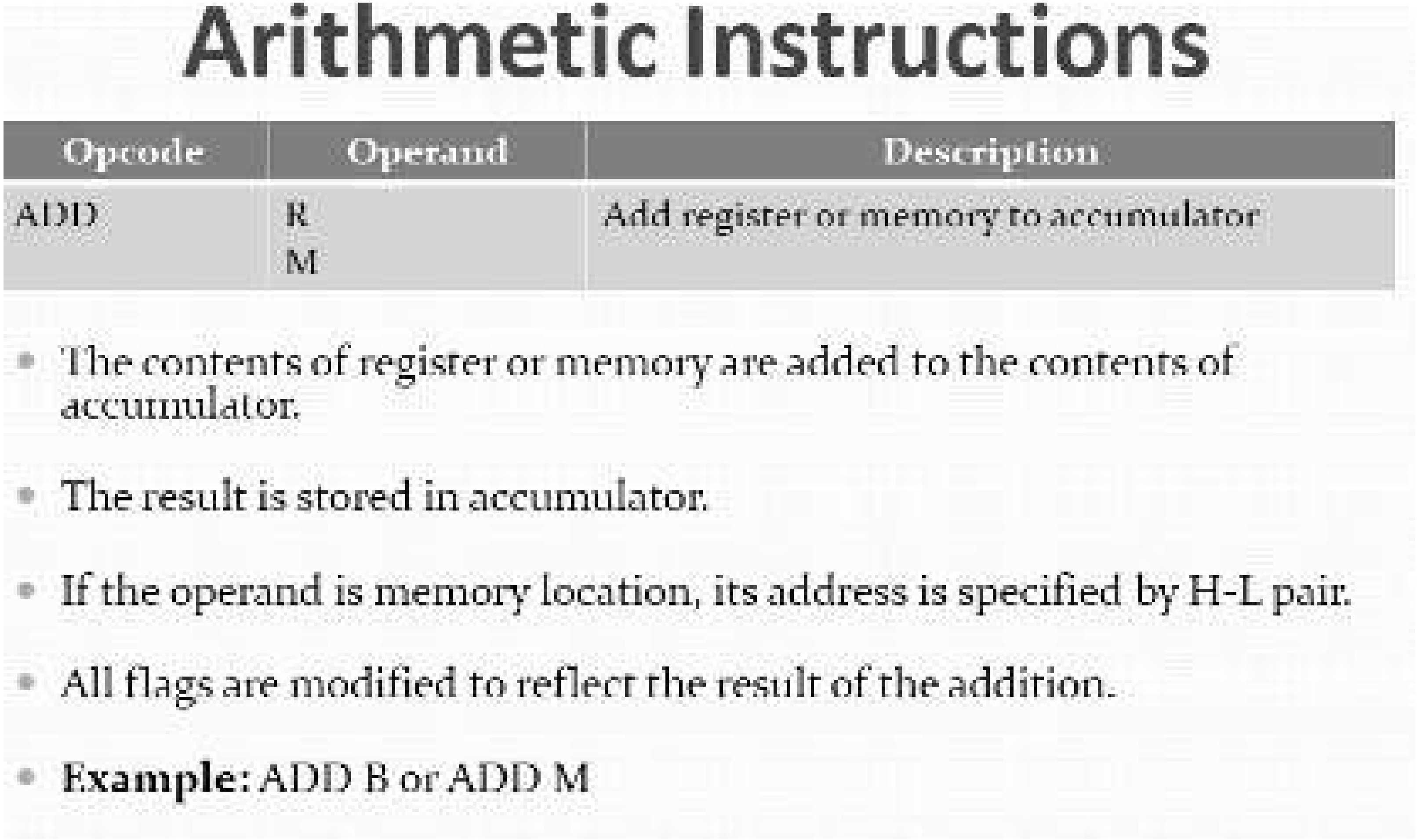
Subtraction

- Any 8-bit number, or the contents of register, or the contents of memory location can be subtracted from the contents of accumulator.
- The result is stored in the accumulator.
- Subtraction is performed in 2's complement form.
- If the result is negative, it is stored in 2's complement form.
- No two other 8-bit registers can be subtracted directly.



Increment / Decrement

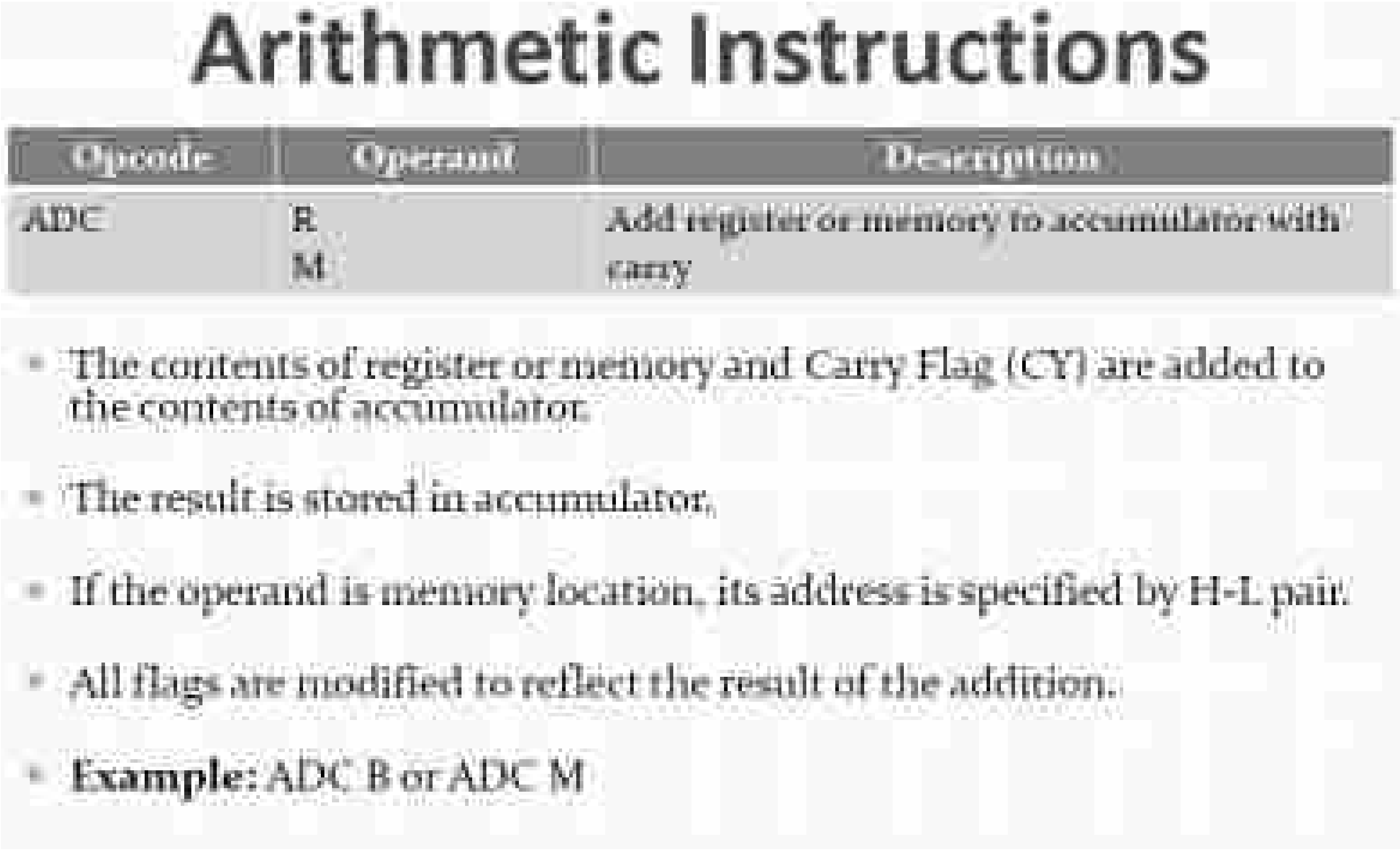
- The 8-bit contents of a register or a memory location can be incremented or decremented by 1.
- The 16-bit contents of a register pair can be incremented or decremented by 1.
- Increment or decrement can be performed on any register or a memory location.



Arithmetic Instructions

Opcode	Operand	Description
ADD	R M	Add register or memory to accumulator

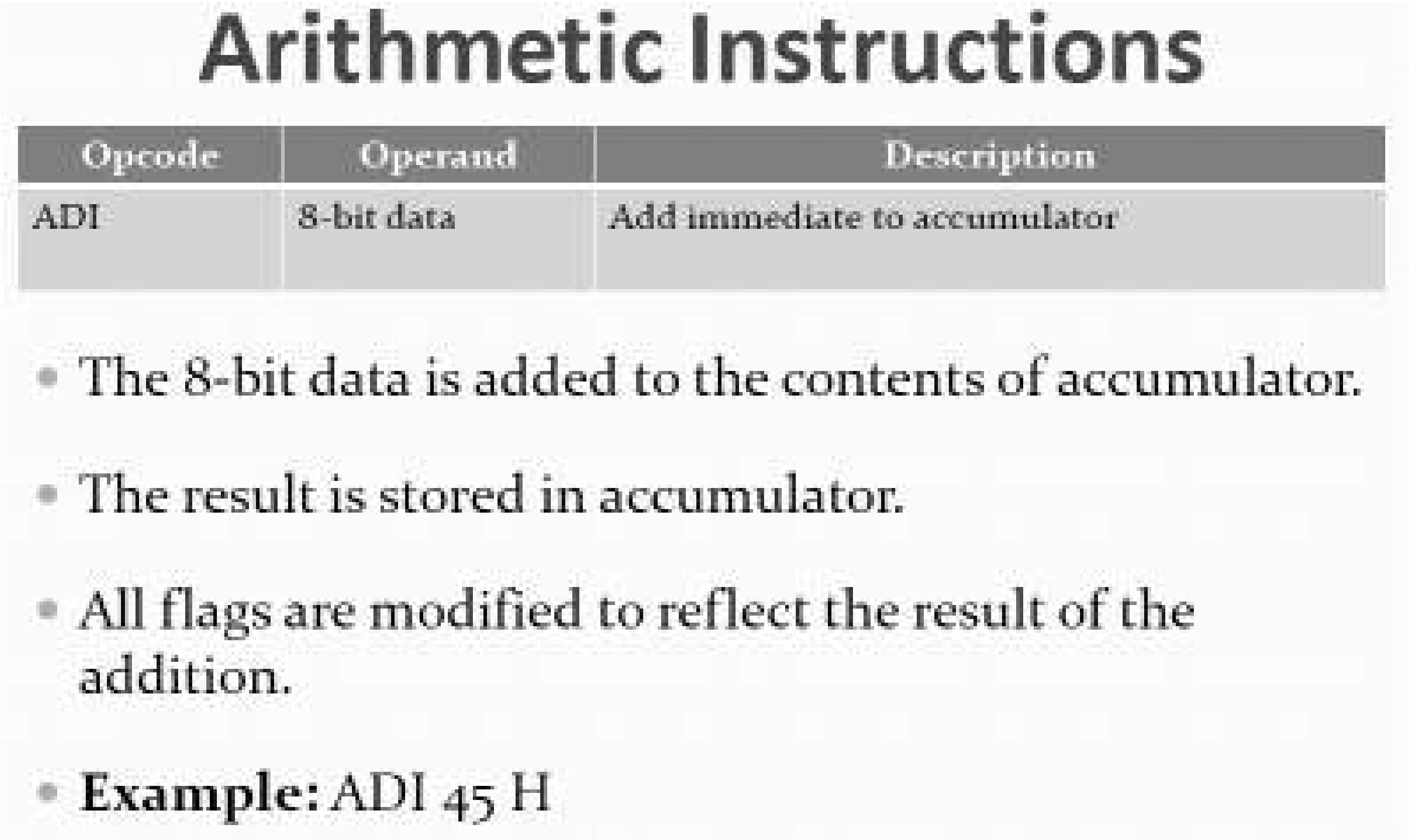
- The contents of register or memory are added to the contents of accumulator.
- The result is stored in accumulator.
- If the operand is memory location, its address is specified by H-L pair.
- All flags are modified to reflect the result of the addition.
- **Example:** ADD B or ADD M



Arithmetic Instructions

Opcode	Operand	Description
ADC	R M	Add register or memory to accumulator with carry

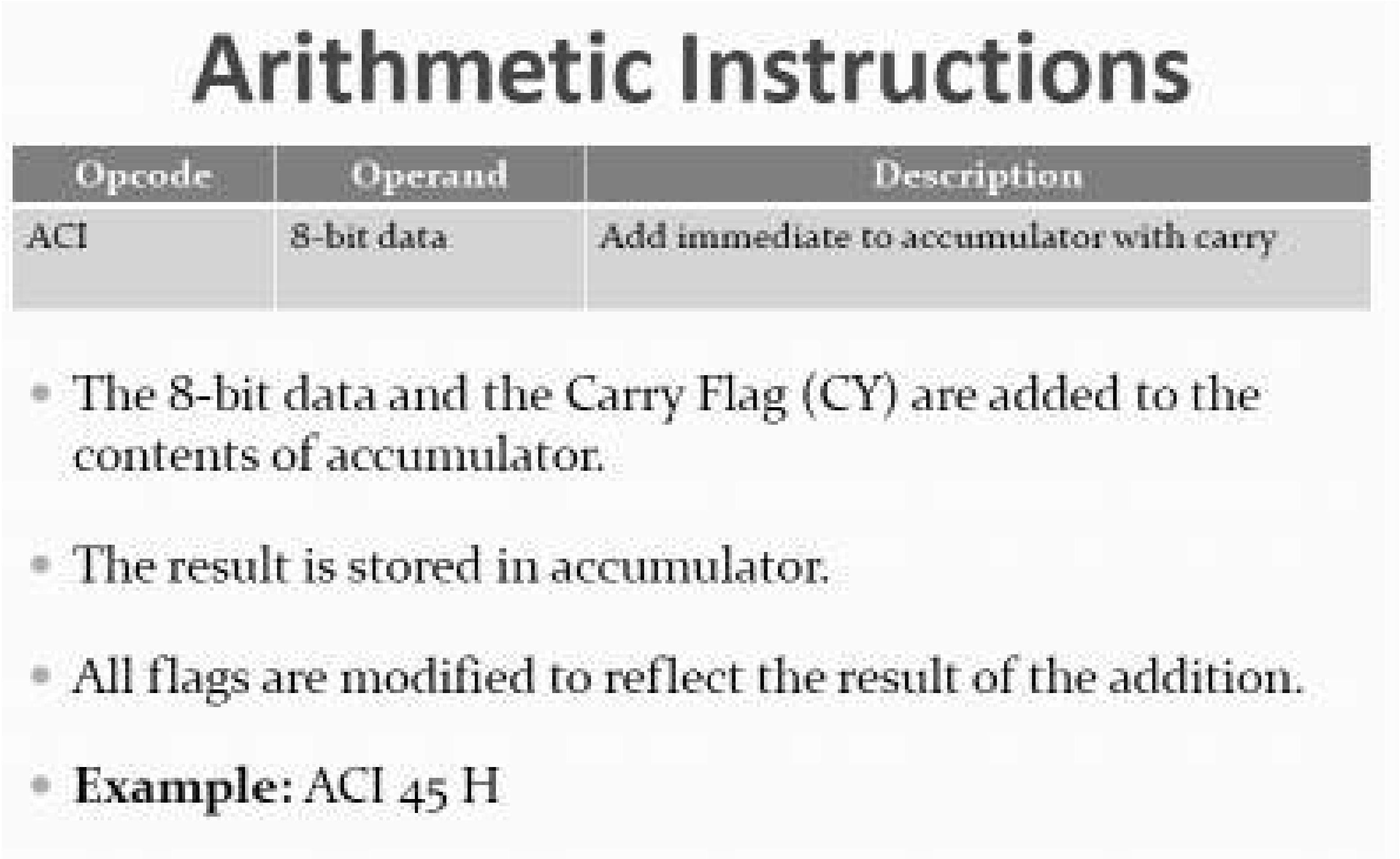
- The contents of register or memory and Carry Flag (CY) are added to the contents of accumulator.
- The result is stored in accumulator.
- If the operand is memory location, its address is specified by H-L pair.
- All flags are modified to reflect the result of the addition.
- **Example:** ADC B or ADC M



Arithmetic Instructions

Opcode	Operand	Description
ADI	8-bit data	Add immediate to accumulator

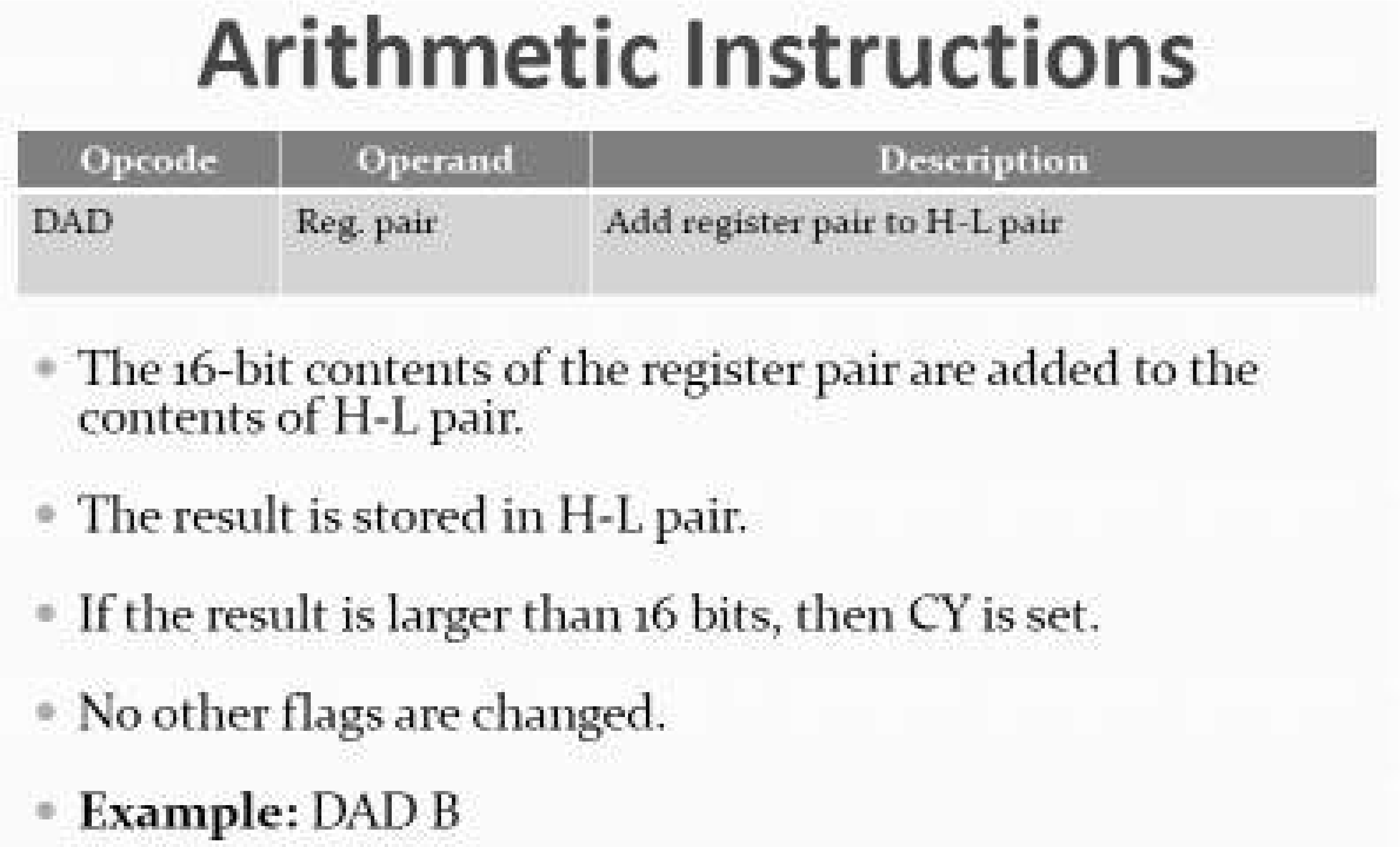
- The 8-bit data is added to the contents of accumulator.
- The result is stored in accumulator.
- All flags are modified to reflect the result of the addition.
- **Example:** ADI 45 H



Arithmetic Instructions

Opcode	Operand	Description
ACI	8-bit data	Add immediate to accumulator with carry

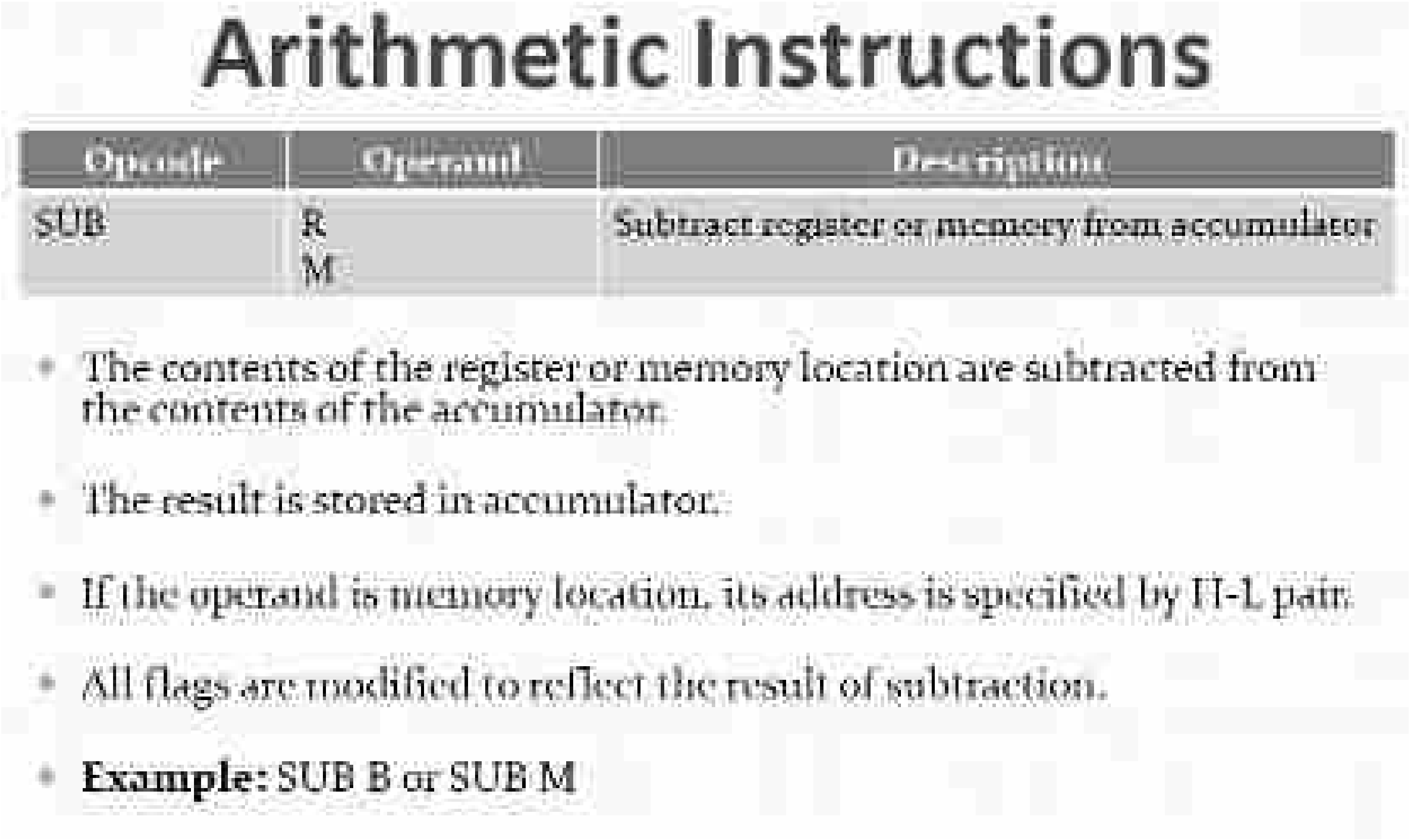
- The 8-bit data and the Carry Flag (CY) are added to the contents of accumulator.
- The result is stored in accumulator.
- All flags are modified to reflect the result of the addition.
- **Example:** ACI 45 H



Arithmetic Instructions

Opcode	Operand	Description
DAD	Reg. pair	Add register pair to H-L pair

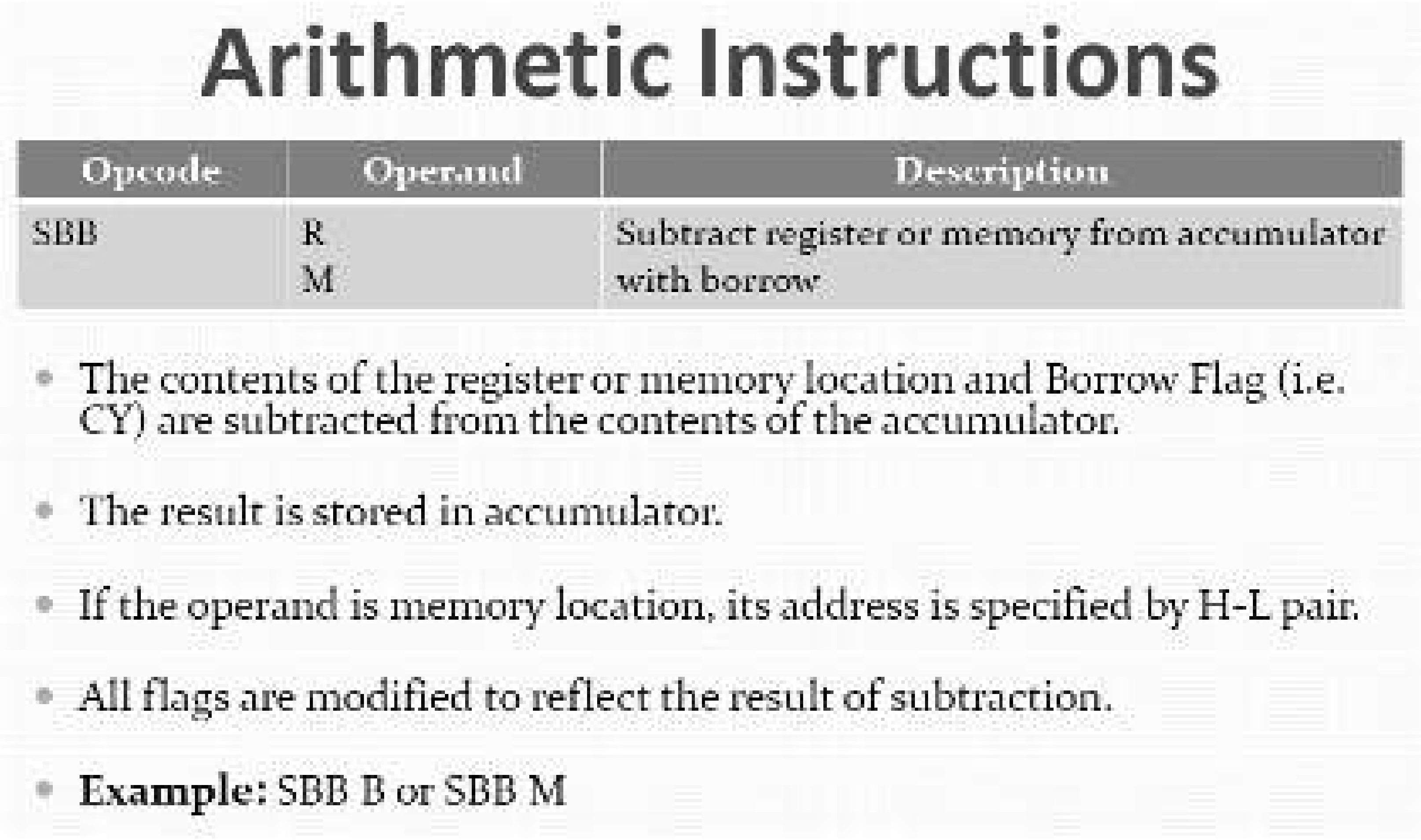
- The 16-bit contents of the register pair are added to the contents of H-L pair.
- The result is stored in H-L pair.
- If the result is larger than 16 bits, then CY is set.
- No other flags are changed.
- **Example:** DAD B



Arithmetic Instructions

Opcode	Operand	Description
SUB	R M	Subtract register or memory from accumulator

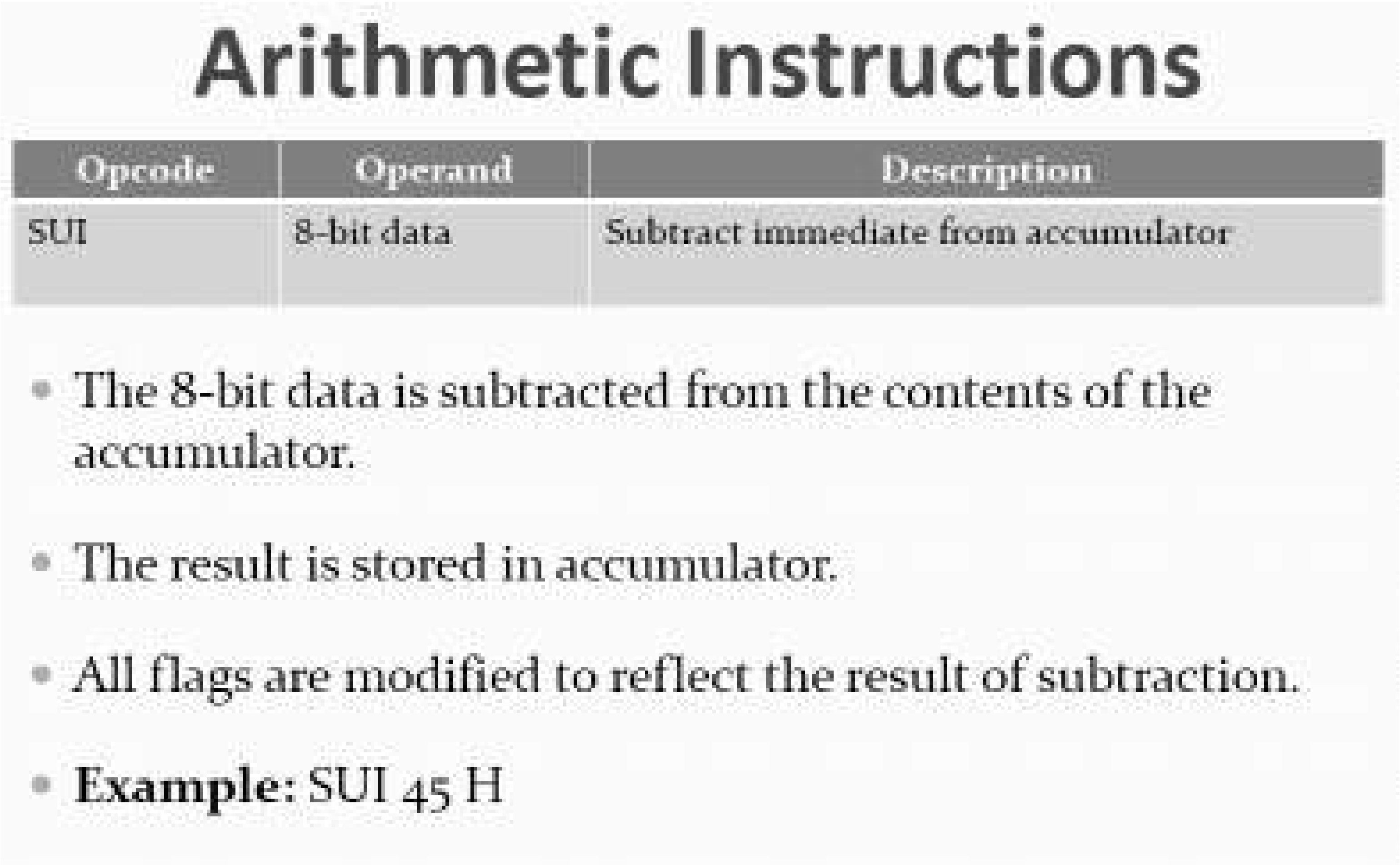
- The contents of the register or memory location are subtracted from the contents of the accumulator.
- The result is stored in accumulator.
- If the operand is memory location, its address is specified by H-L pair.
- All flags are modified to reflect the result of subtraction.
- **Example:** SUB B or SUB M



Arithmetic Instructions

Opcode	Operand	Description
SBB	R M	Subtract register or memory from accumulator with borrow

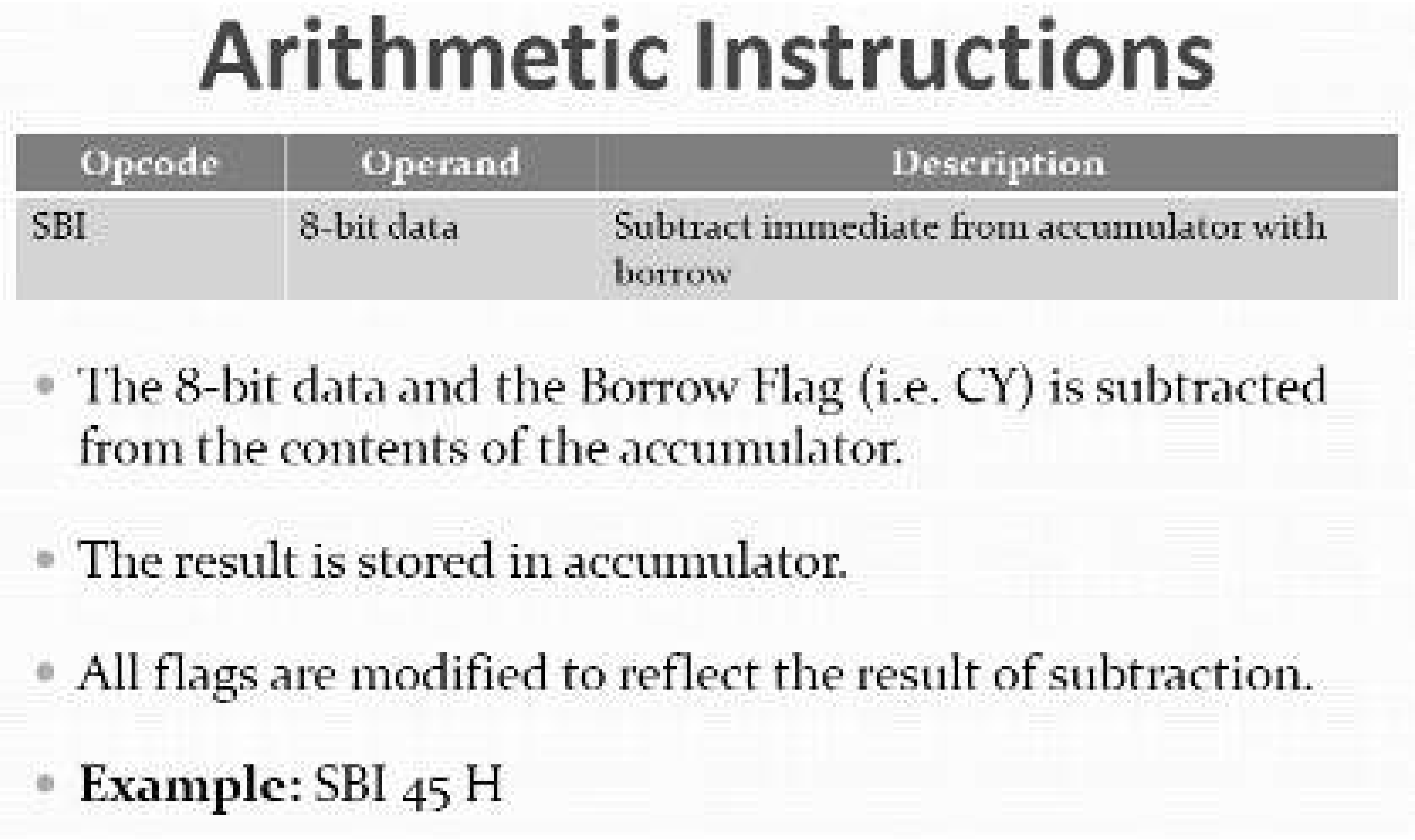
- The contents of the register or memory location and Borrow Flag (i.e. CY) are subtracted from the contents of the accumulator.
- The result is stored in accumulator.
- If the operand is memory location, its address is specified by H-L pair.
- All flags are modified to reflect the result of subtraction.
- **Example:** SBB B or SBB M



Arithmetic Instructions

Opcode	Operand	Description
SUI	8-bit data	Subtract immediate from accumulator

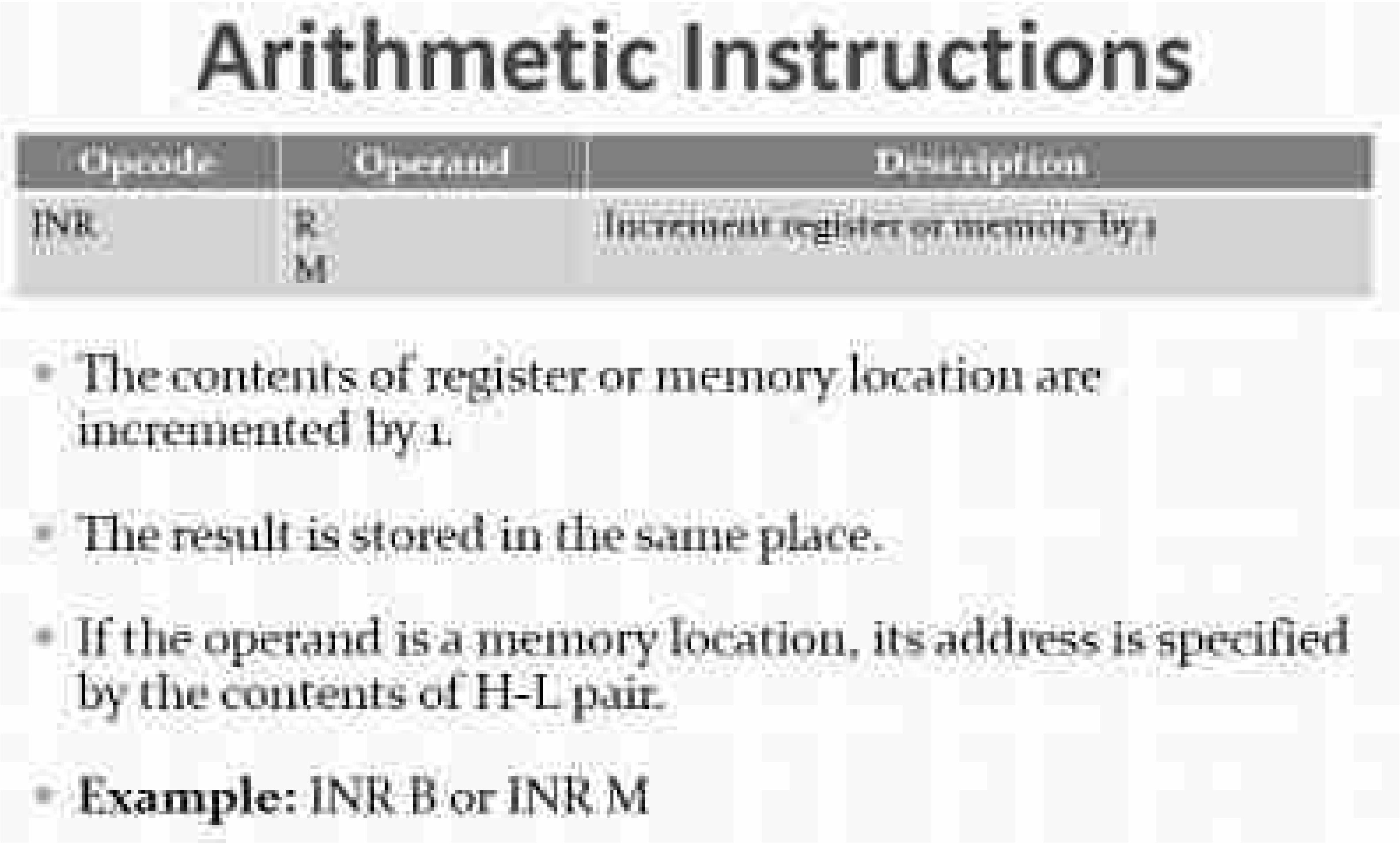
- The 8-bit data is subtracted from the contents of the accumulator.
- The result is stored in accumulator.
- All flags are modified to reflect the result of subtraction.
- **Example:** SUI 45 H



Arithmetic Instructions

Opcode	Operand	Description
SBI	8-bit data	Subtract immediate from accumulator with borrow

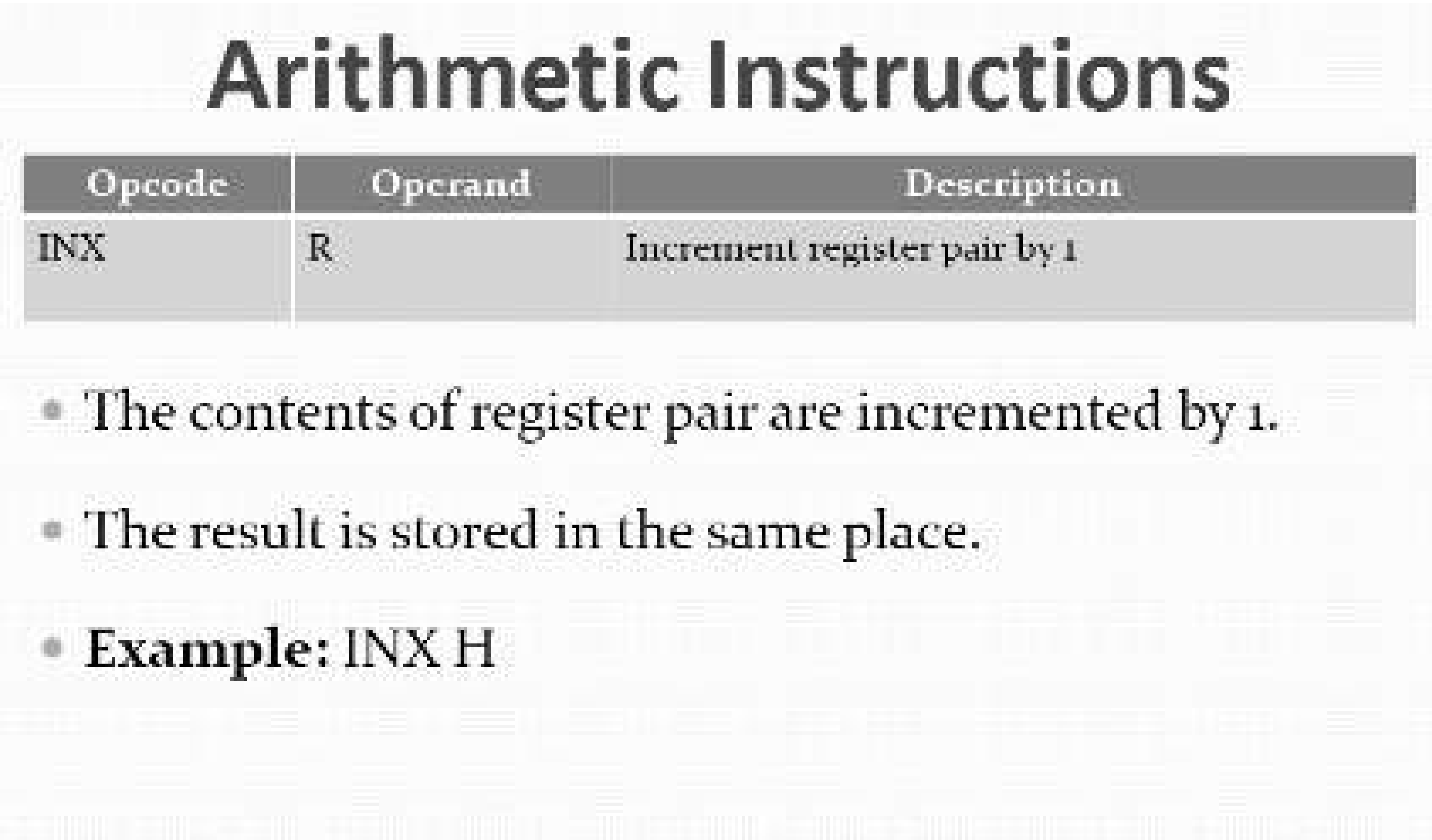
- The 8-bit data and the Borrow Flag (i.e. CY) is subtracted from the contents of the accumulator.
- The result is stored in accumulator.
- All flags are modified to reflect the result of subtraction.
- **Example:** SBI 45 H



Arithmetic Instructions

Opcode	Operand	Description
INR	R M	Increment register or memory by 1

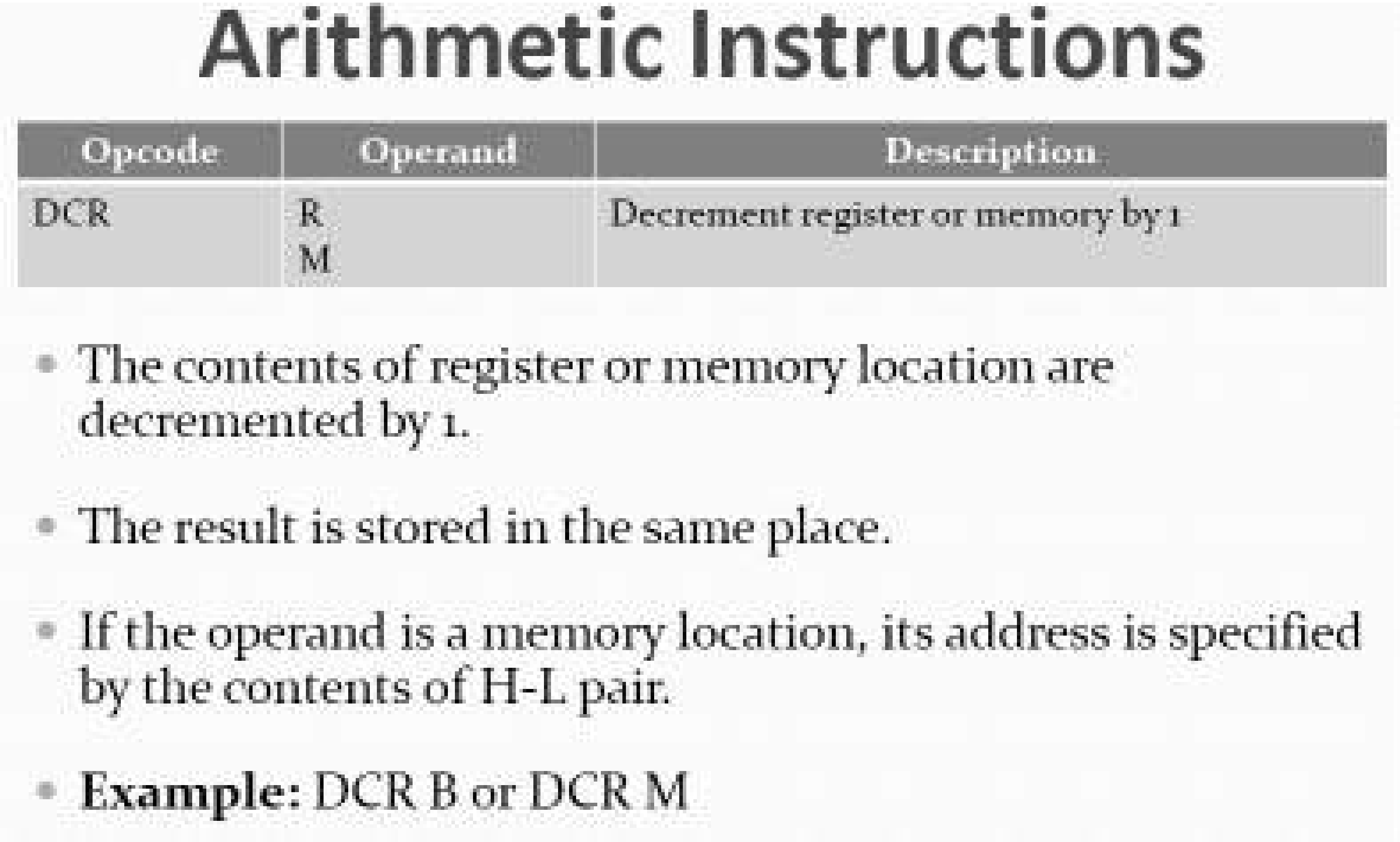
- The contents of register or memory location are incremented by 1.
- The result is stored in the same place.
- If the operand is a memory location, its address is specified by the contents of H-L pair.
- **Example:** INR B or INR M



Arithmetic Instructions

Opcode	Operand	Description
INX	R	Increment register pair by 1

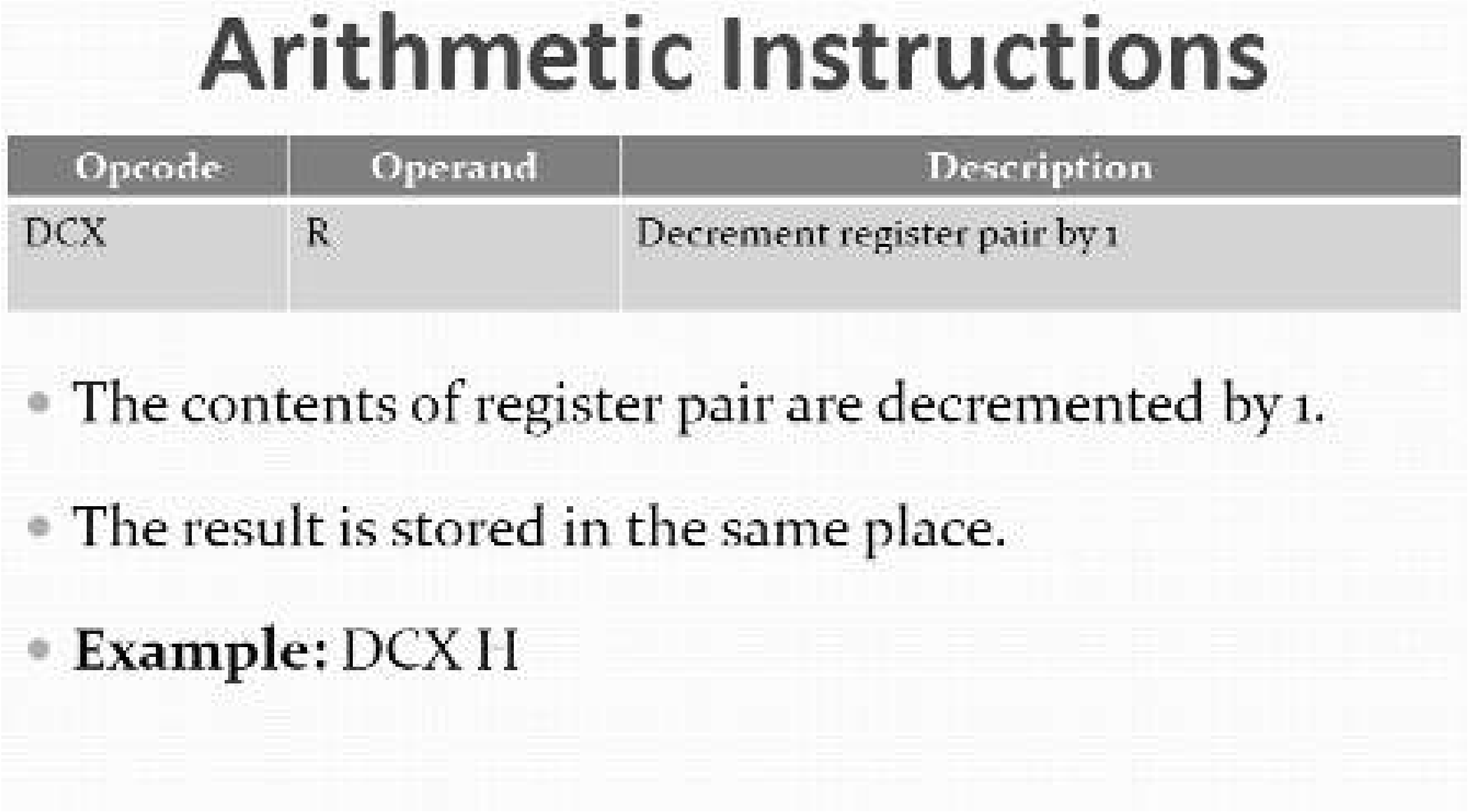
- The contents of register pair are incremented by 1.
- The result is stored in the same place.
- **Example:** INX H



Arithmetic Instructions

Opcode	Operand	Description
DCR	R M	Decrement register or memory by 1

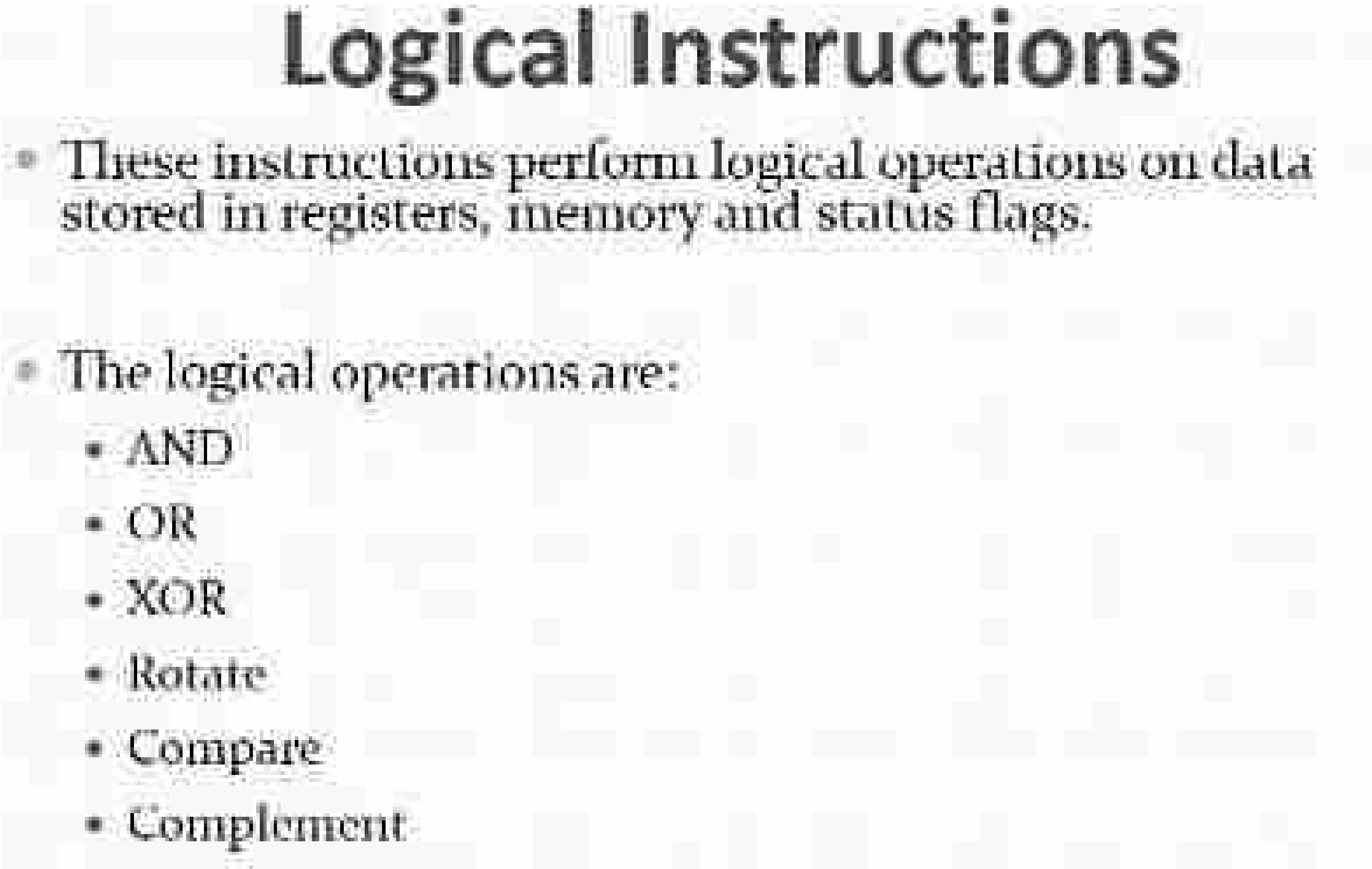
- The contents of register or memory location are decremented by 1.
- The result is stored in the same place.
- If the operand is a memory location, its address is specified by the contents of H-L pair.
- **Example:** DCR B or DCR M



Arithmetic Instructions

Opcode	Operand	Description
DCX	R	Decrement register pair by 1

- The contents of register pair are decremented by 1.
- The result is stored in the same place.
- **Example:** DCX H

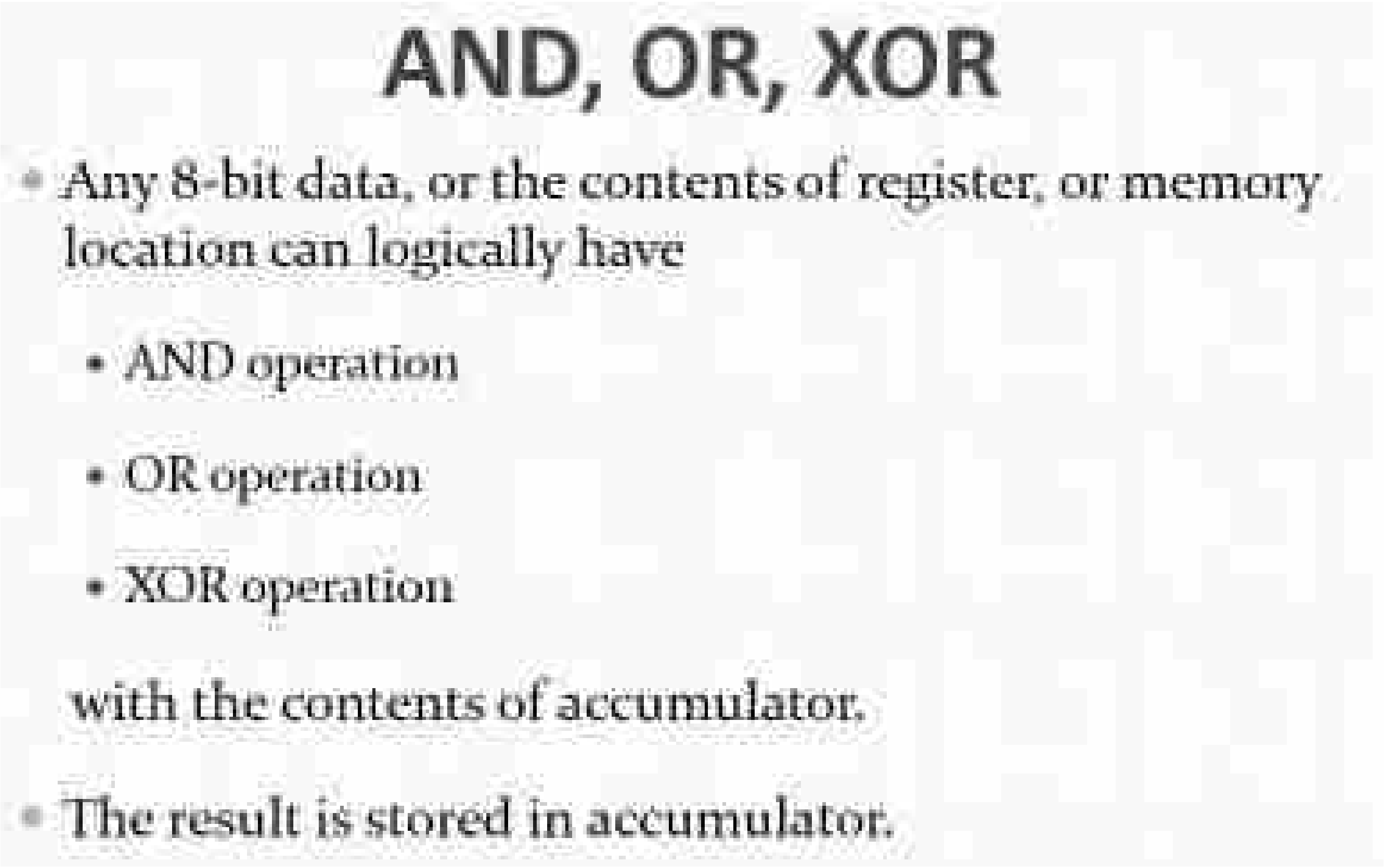


Logical Instructions

- These instructions perform logical operations on data stored in registers, memory and status flags.

- The logical operations are:
  - AND
  - OR
  - XOR
  - Rotate
  - Compare
  - Complement

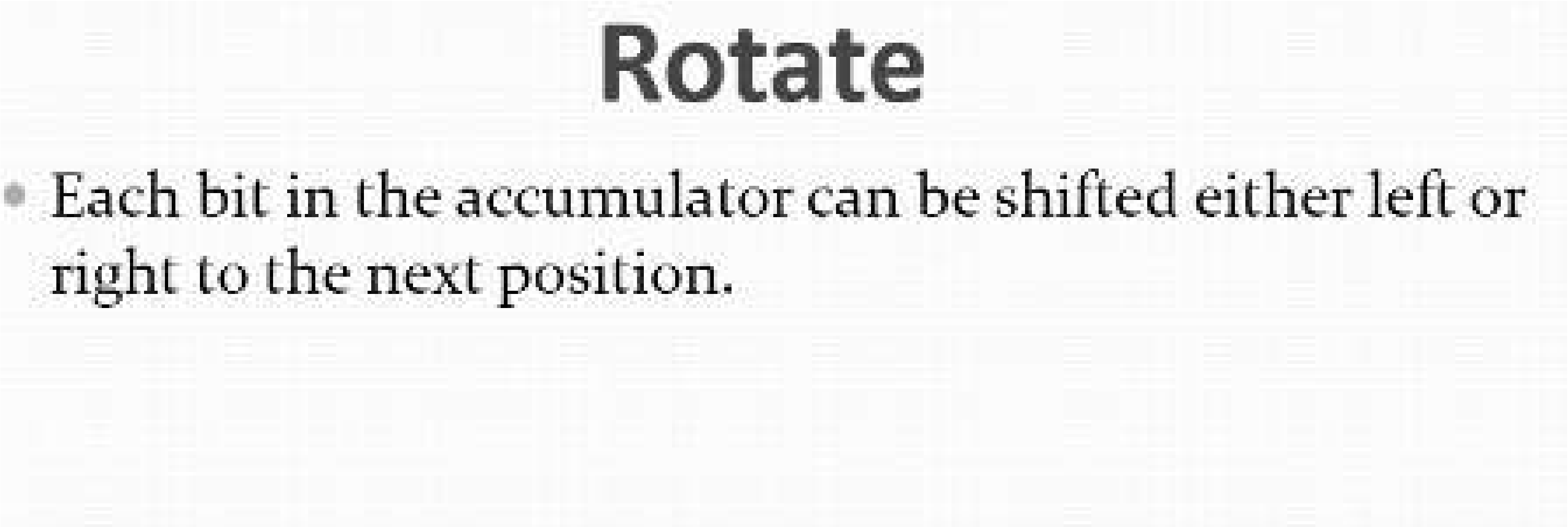
- PSW (Program Status word)
- - Flag unaffected
- * affected
- 0 reset
- 1 set
- S Sign (Bit 7)
- Z   Zero (Bit 6)
- AC Auxiliary Carry (Bit 4)
- P   Parity (Bit 2)
- CY Carry (Bit 0)



AND, OR, XOR

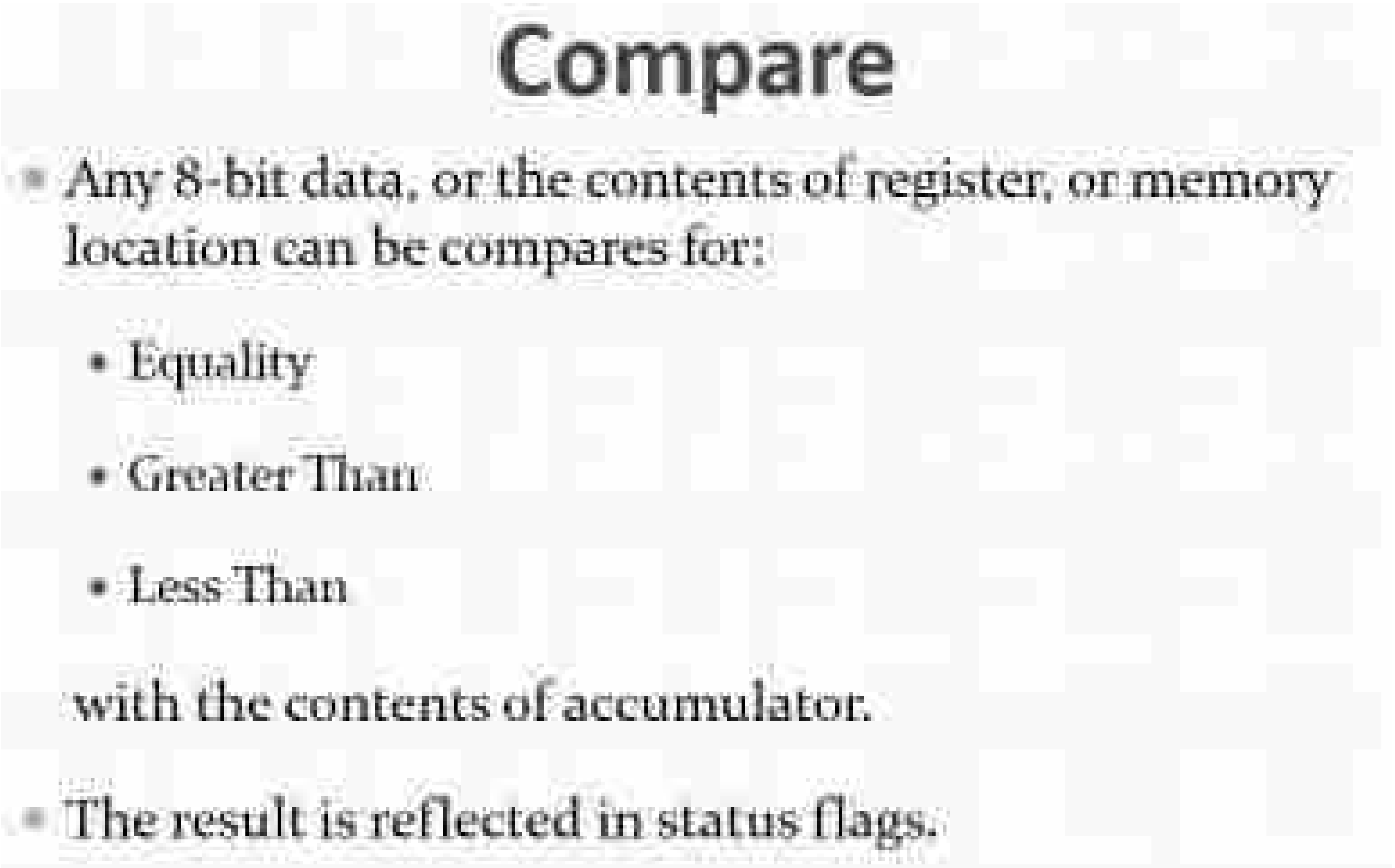
- Any 8-bit data, or the contents of register, or memory location can logically have
  - AND operation
  - OR operation
  - XOR operation

  with the contents of accumulator.
- The result is stored in accumulator.



Rotate

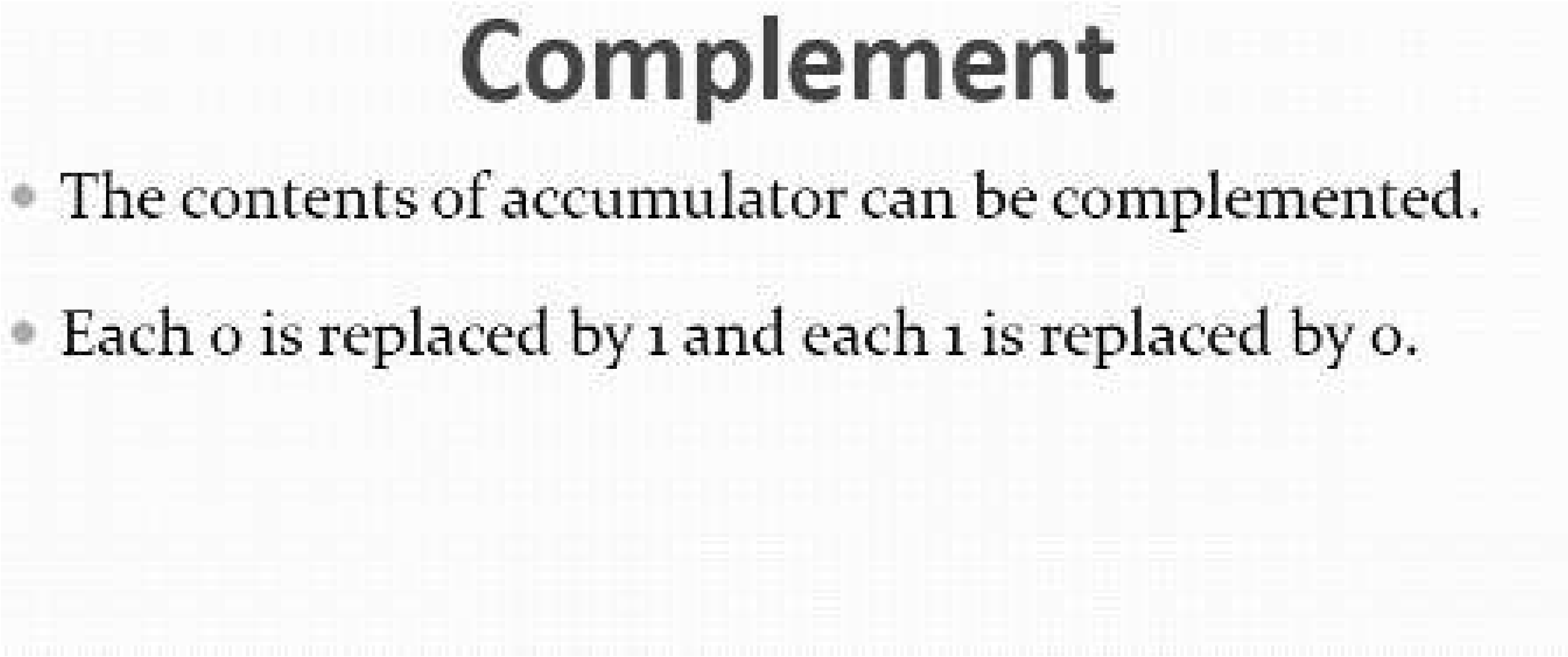
- Each bit in the accumulator can be shifted either left or right to the next position.



Compare

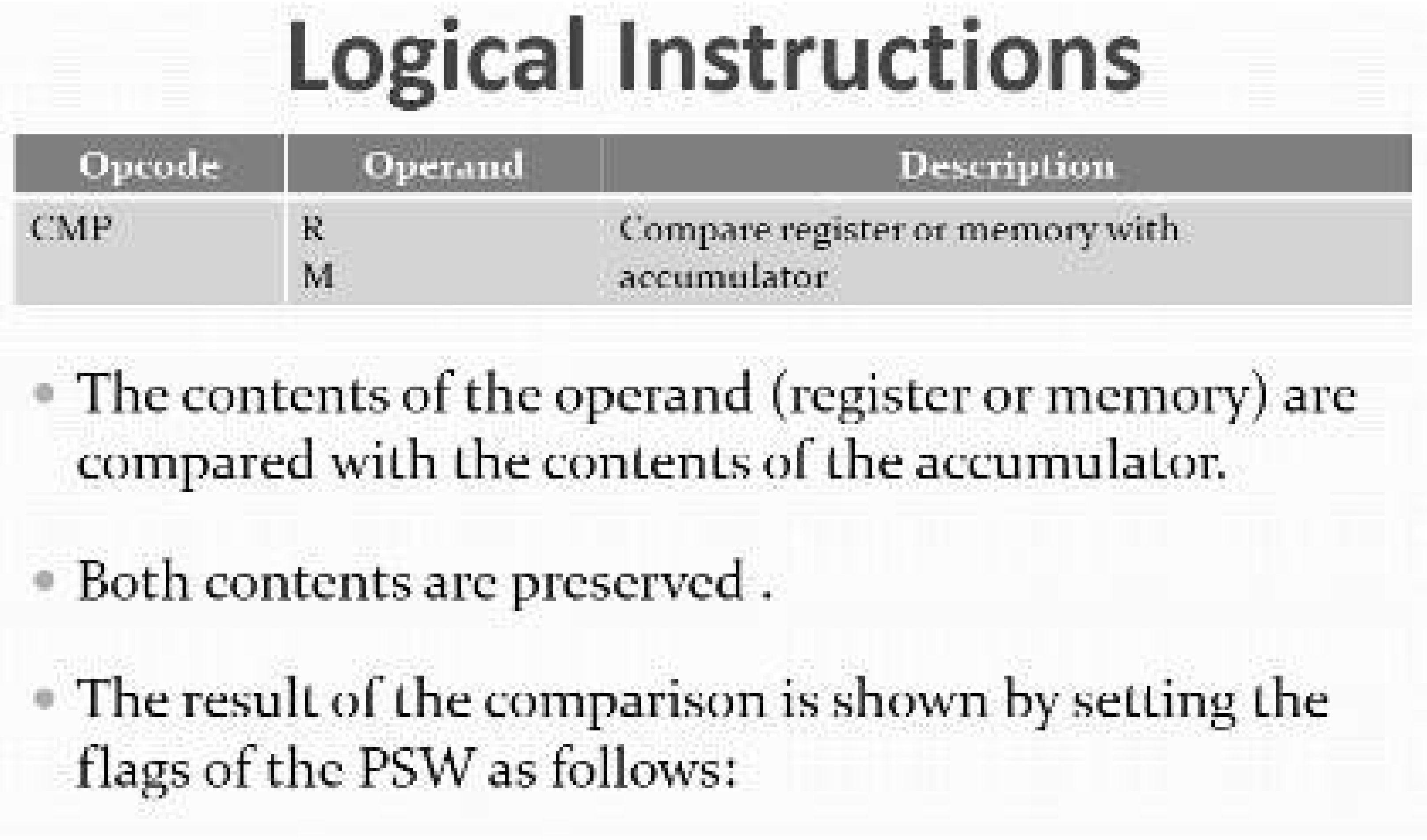
- Any 8-bit data, or the contents of register, or memory location can be compares for:
  - Equality
  - Greater Than
  - Less Than

  with the contents of accumulator.
- The result is reflected in status flags.



Complement

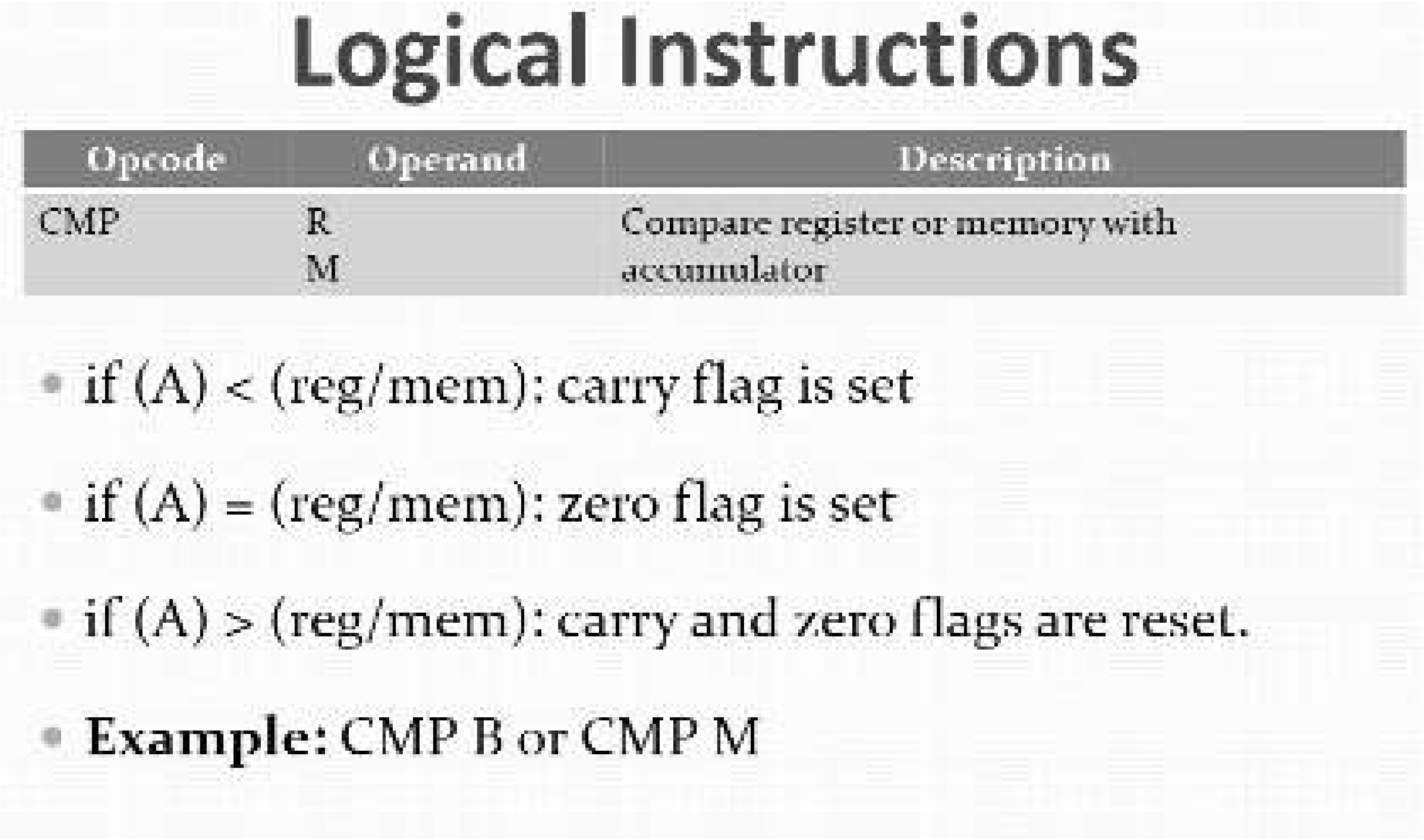
- The contents of accumulator can be complemented.
- Each 0 is replaced by 1 and each 1 is replaced by 0.



Logical Instructions

Opcode	Operand	Description
CMP	R M	Compare register or memory with accumulator

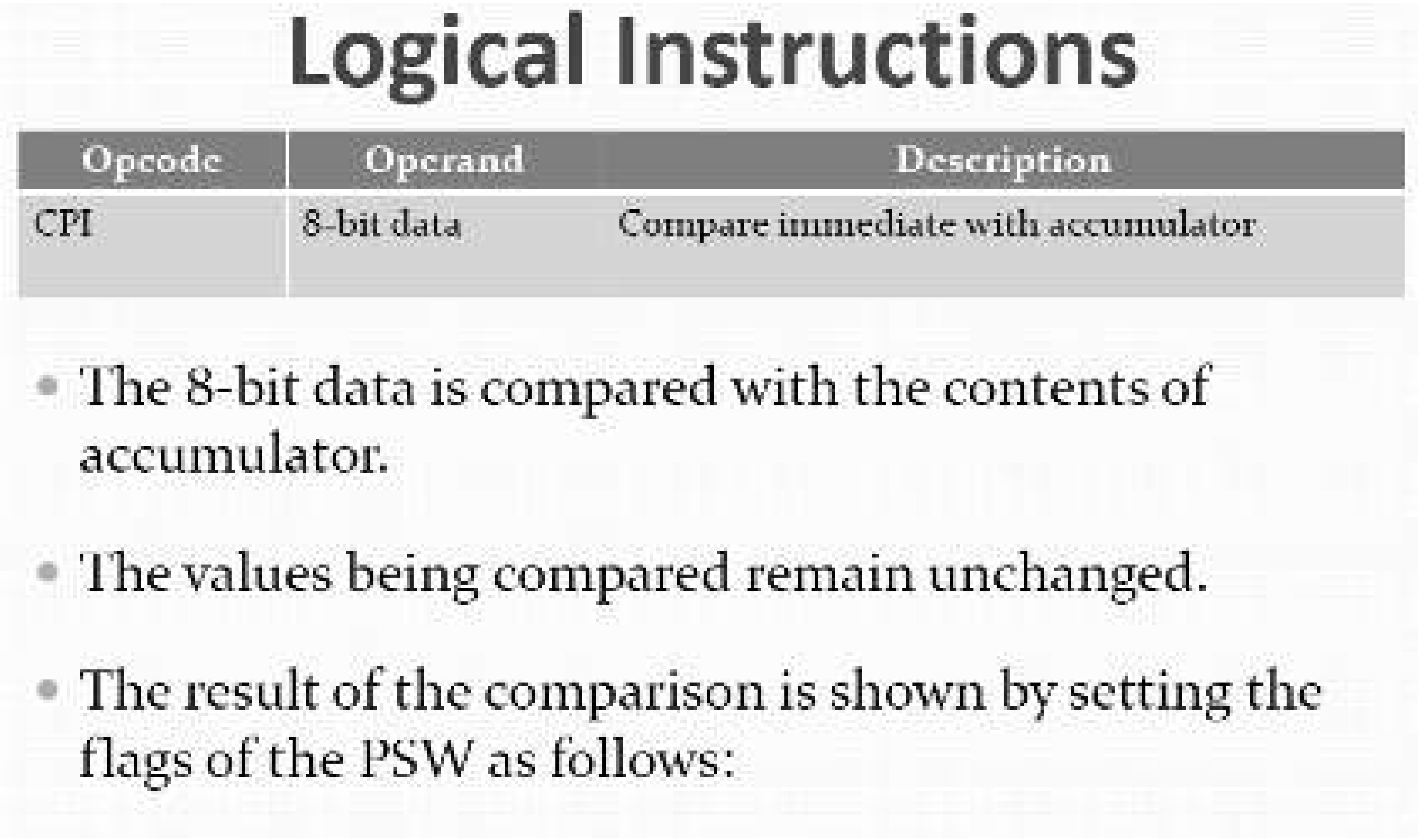
- The contents of the operand (register or memory) are compared with the contents of the accumulator.
- Both contents are preserved .
- The result of the comparison is shown by setting the flags of the PSW as follows:



Logical Instructions

Opcode	Operand	Description
CMP	R M	Compare register or memory with accumulator

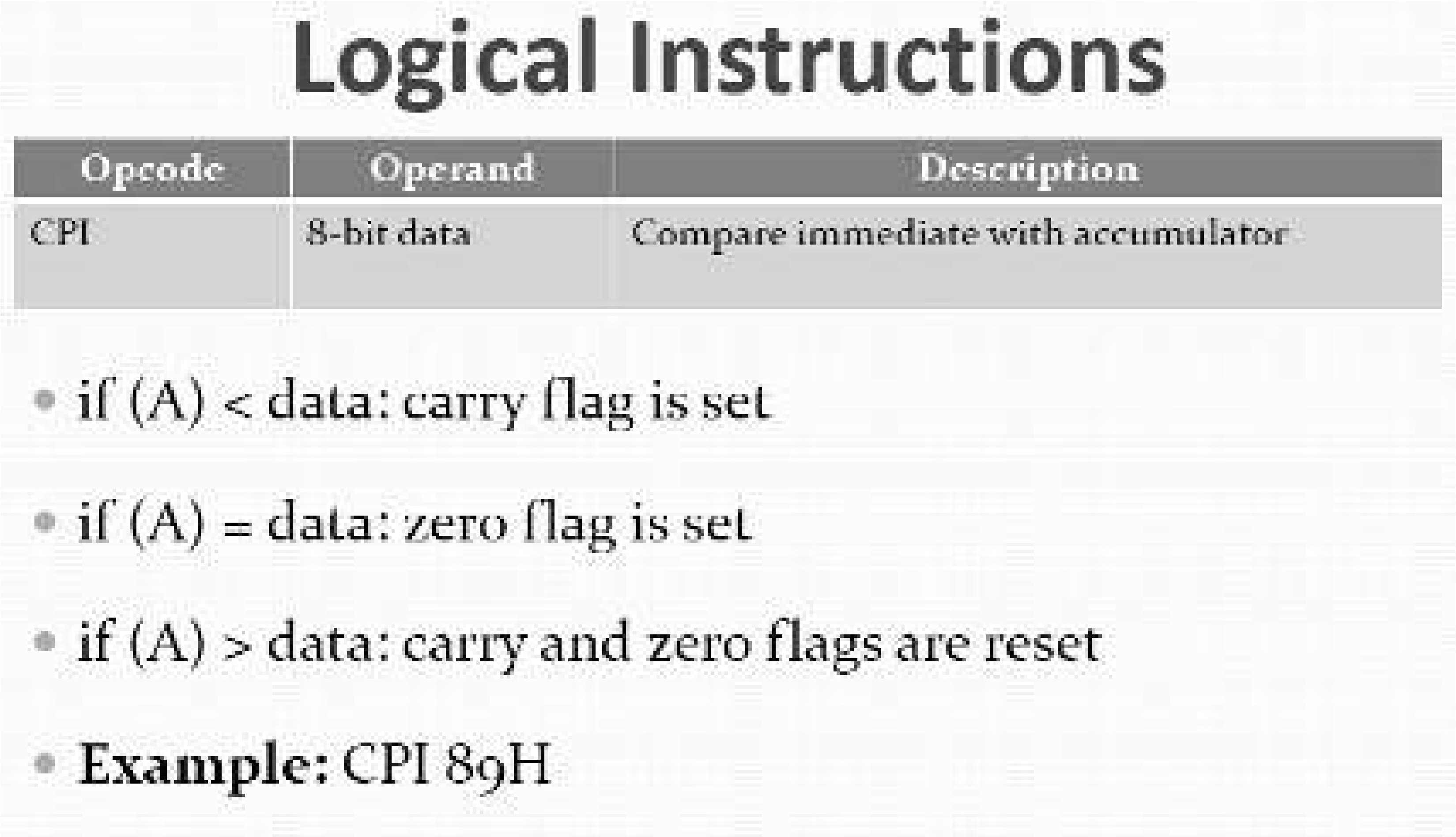
- if (A) < (reg/mem): carry flag is set
- if (A) = (reg/mem): zero flag is set
- if (A) > (reg/mem): carry and zero flags are reset.
- **Example:** CMP B or CMP M



Logical Instructions

Opcode	Operand	Description
CPI	8-bit data	Compare immediate with accumulator

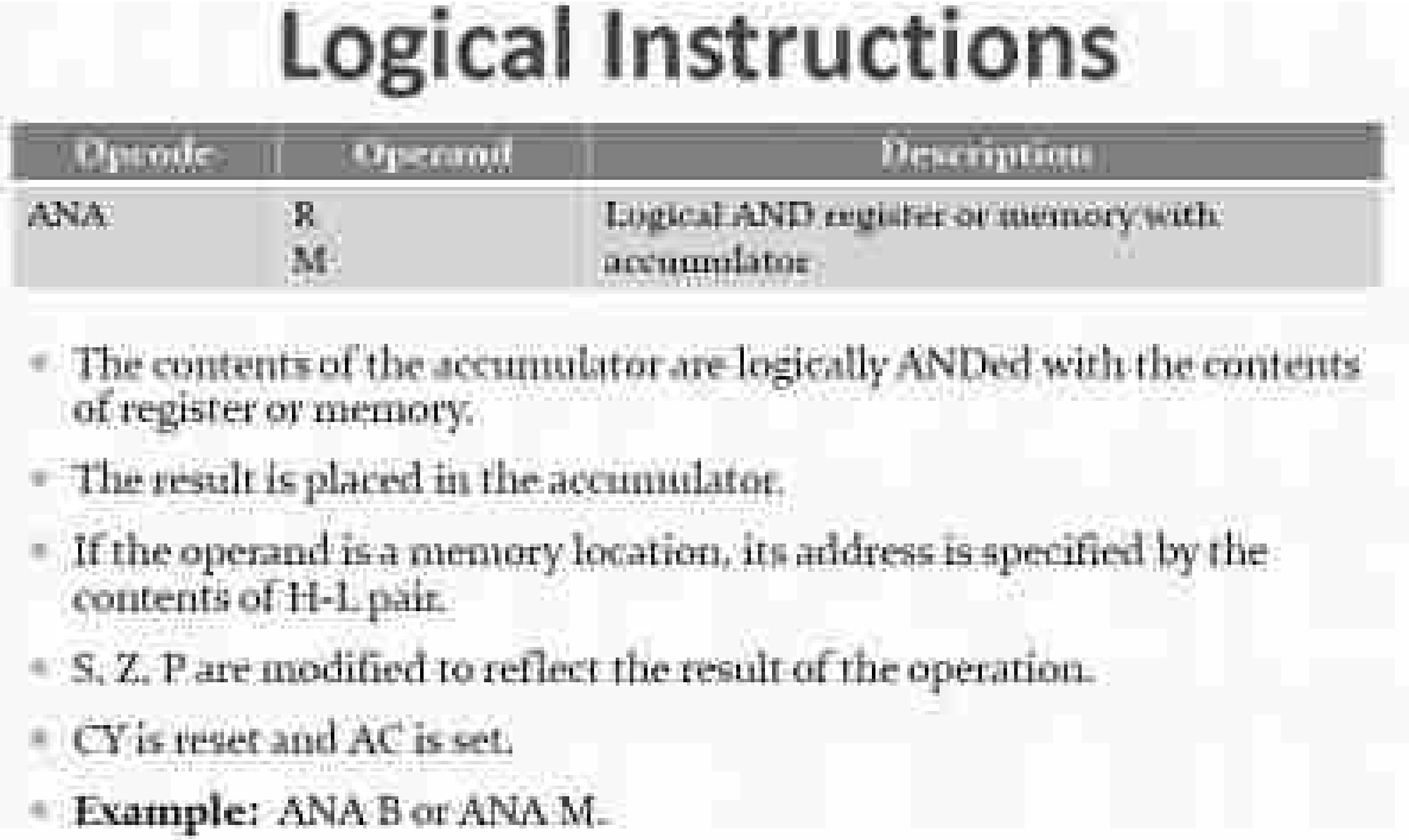
- The 8-bit data is compared with the contents of accumulator.
- The values being compared remain unchanged.
- The result of the comparison is shown by setting the flags of the PSW as follows:



Logical Instructions

Opcode	Operand	Description
CPI	8-bit data	Compare immediate with accumulator

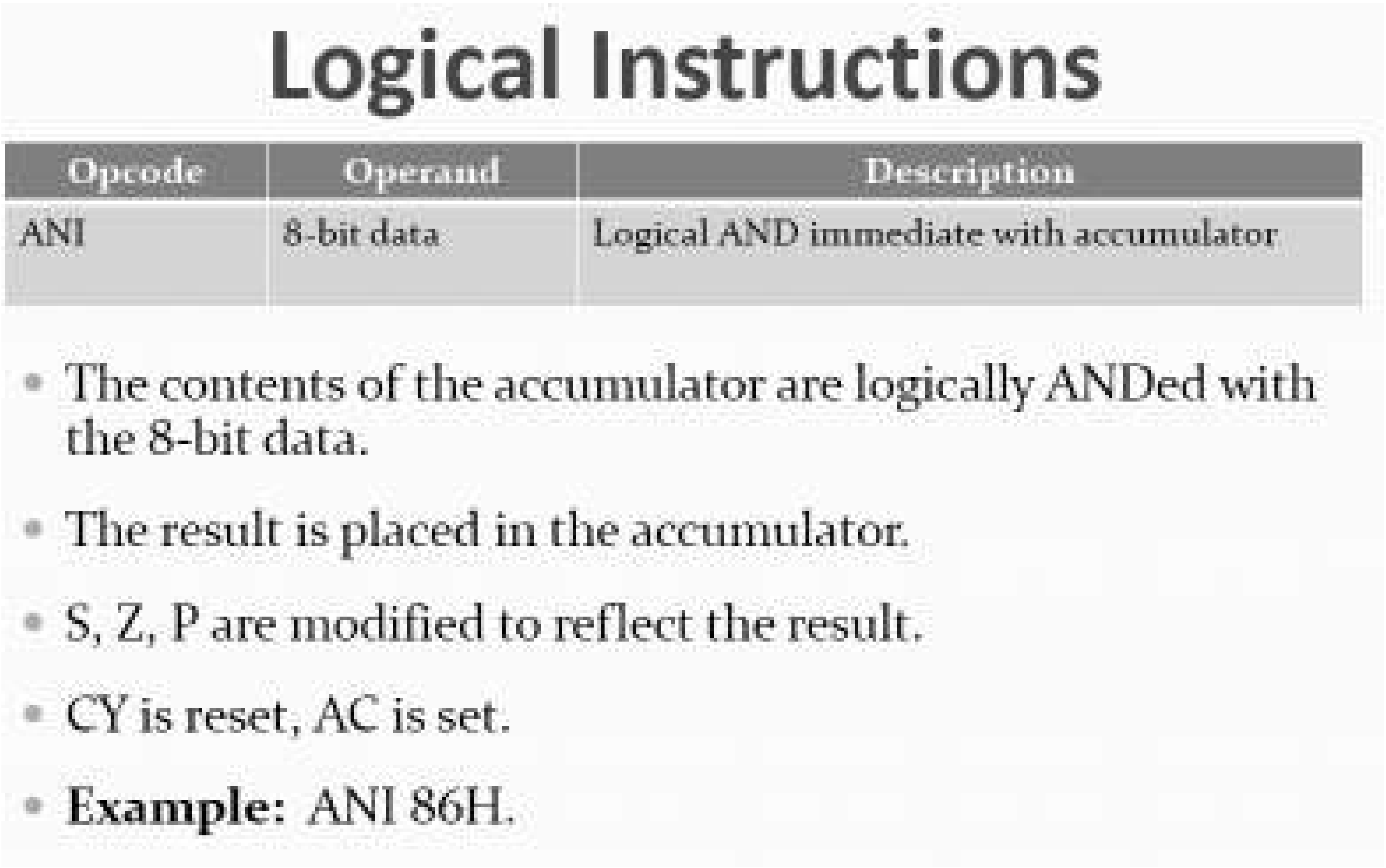
- if (A) < data: carry flag is set
- if (A) = data: zero flag is set
- if (A) > data: carry and zero flags are reset
- **Example:** CPI 89H



Logical Instructions

Opcode	Operand	Description
ANA	R M	Logical AND register or memory with accumulator

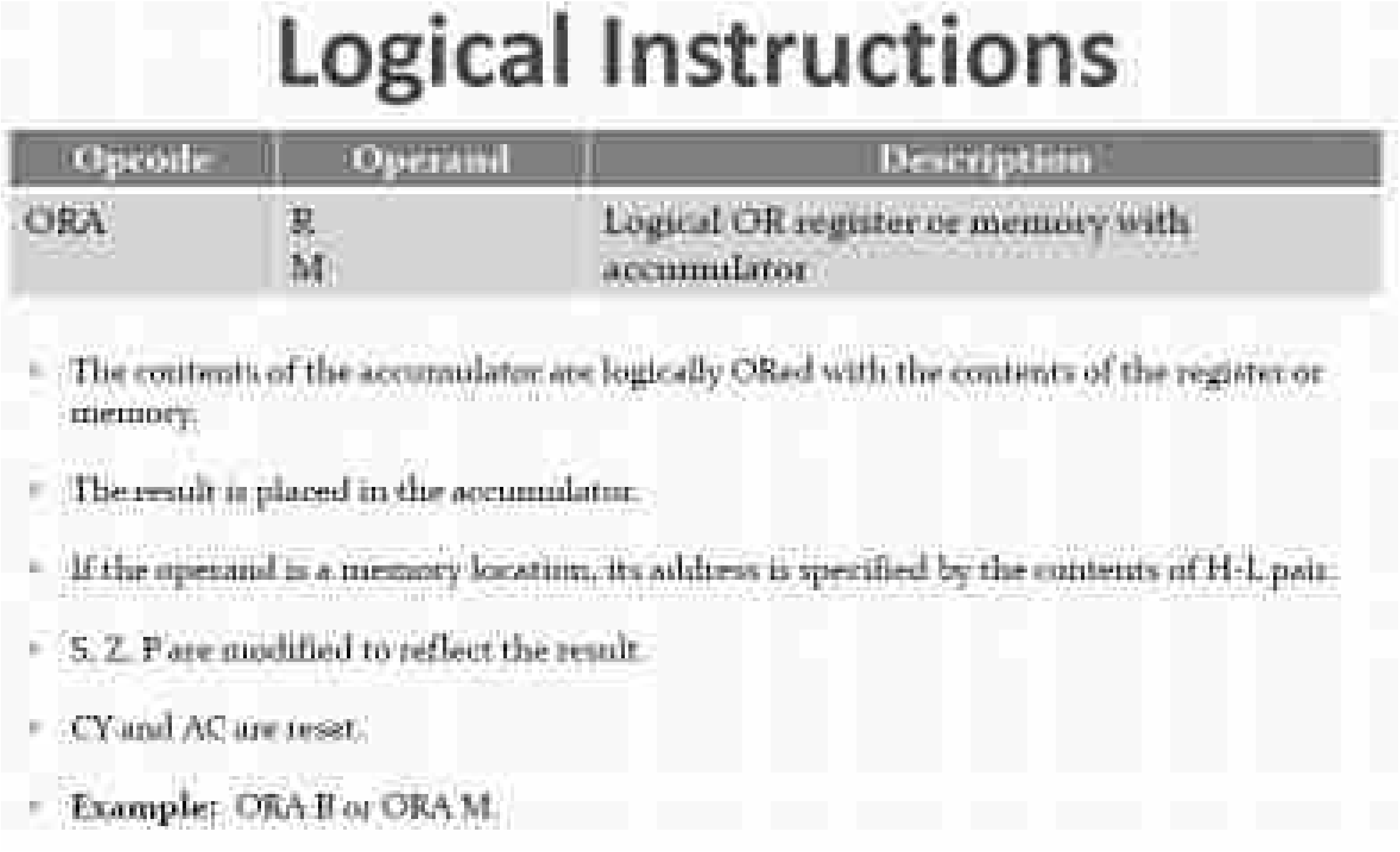
- The contents of the accumulator are logically ANDed with the contents of register or memory.
- The result is placed in the accumulator.
- If the operand is a memory location, its address is specified by the contents of H-L pair.
- S, Z, P are modified to reflect the result of the operation.
- CY is reset and AC is set.
- **Example:** ANA B or ANA M.



Logical Instructions

Opcode	Operand	Description
ANI	8-bit data	Logical AND immediate with accumulator

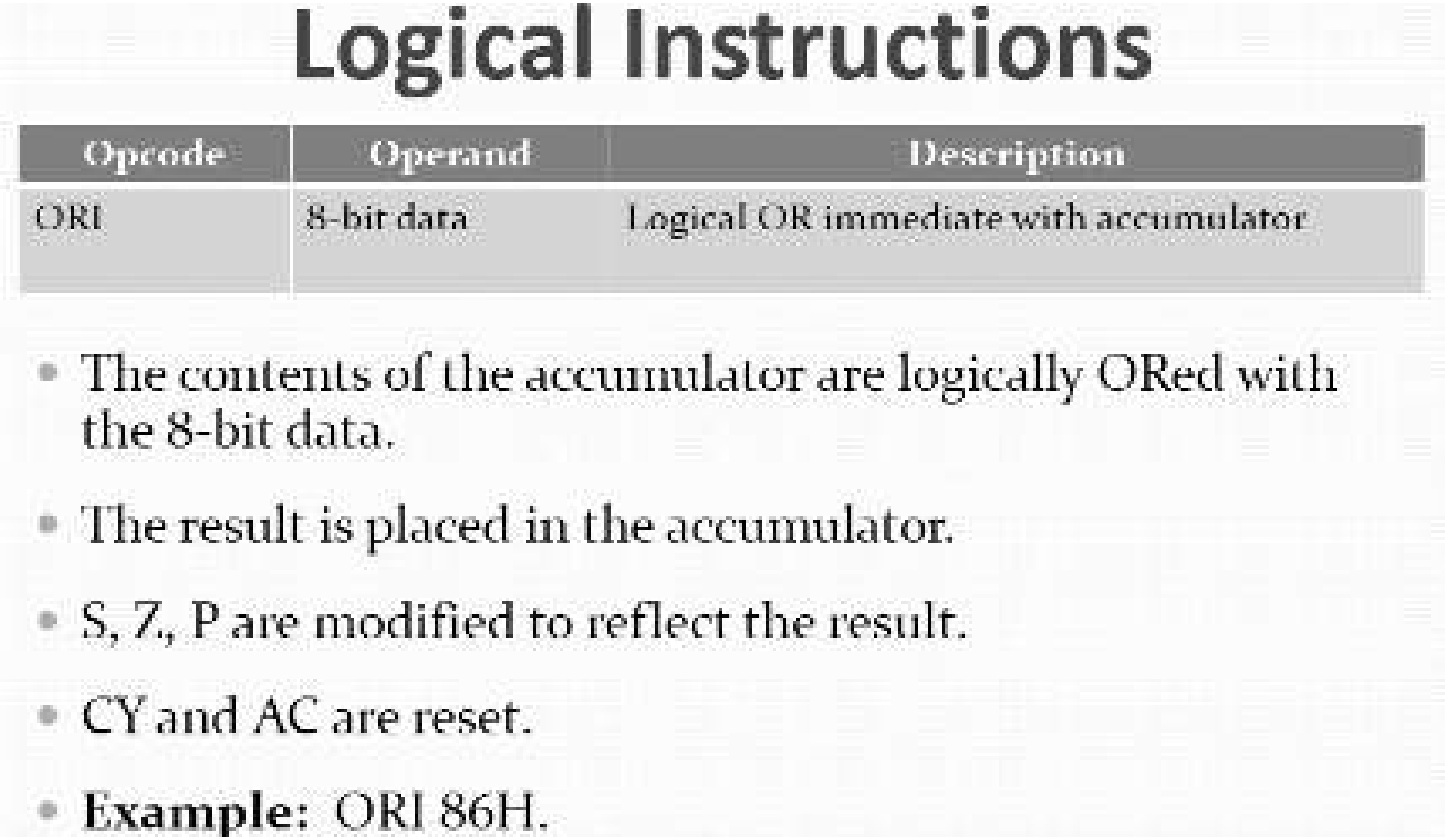
- The contents of the accumulator are logically ANDed with the 8-bit data.
- The result is placed in the accumulator.
- S, Z, P are modified to reflect the result.
- CY is reset, AC is set.
- **Example:** ANI 86H.



Logical Instructions

Opcode	Operand	Description
ORA	R M	Logical OR register or memory with accumulator

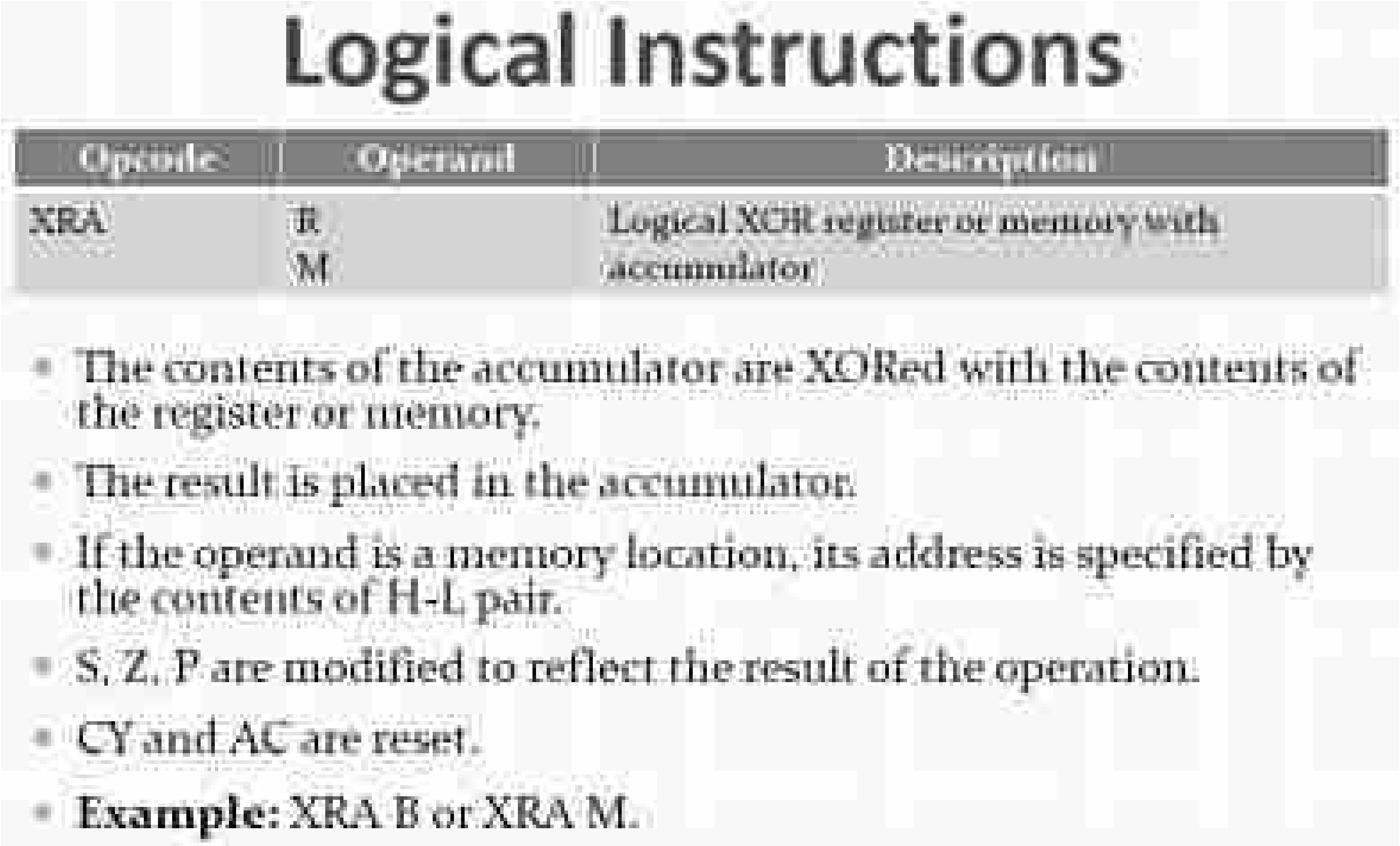
- The contents of the accumulator are logically ORed with the contents of the register or memory.
- The result is placed in the accumulator.
- If the operand is a memory location, its address is specified by the contents of H-L pair.
- S, Z, P are modified to reflect the result.
- CY and AC are reset.
- **Example:** ORA B or ORA M.



Logical Instructions

Opcode	Operand	Description
ORI	8-bit data	Logical OR immediate with accumulator

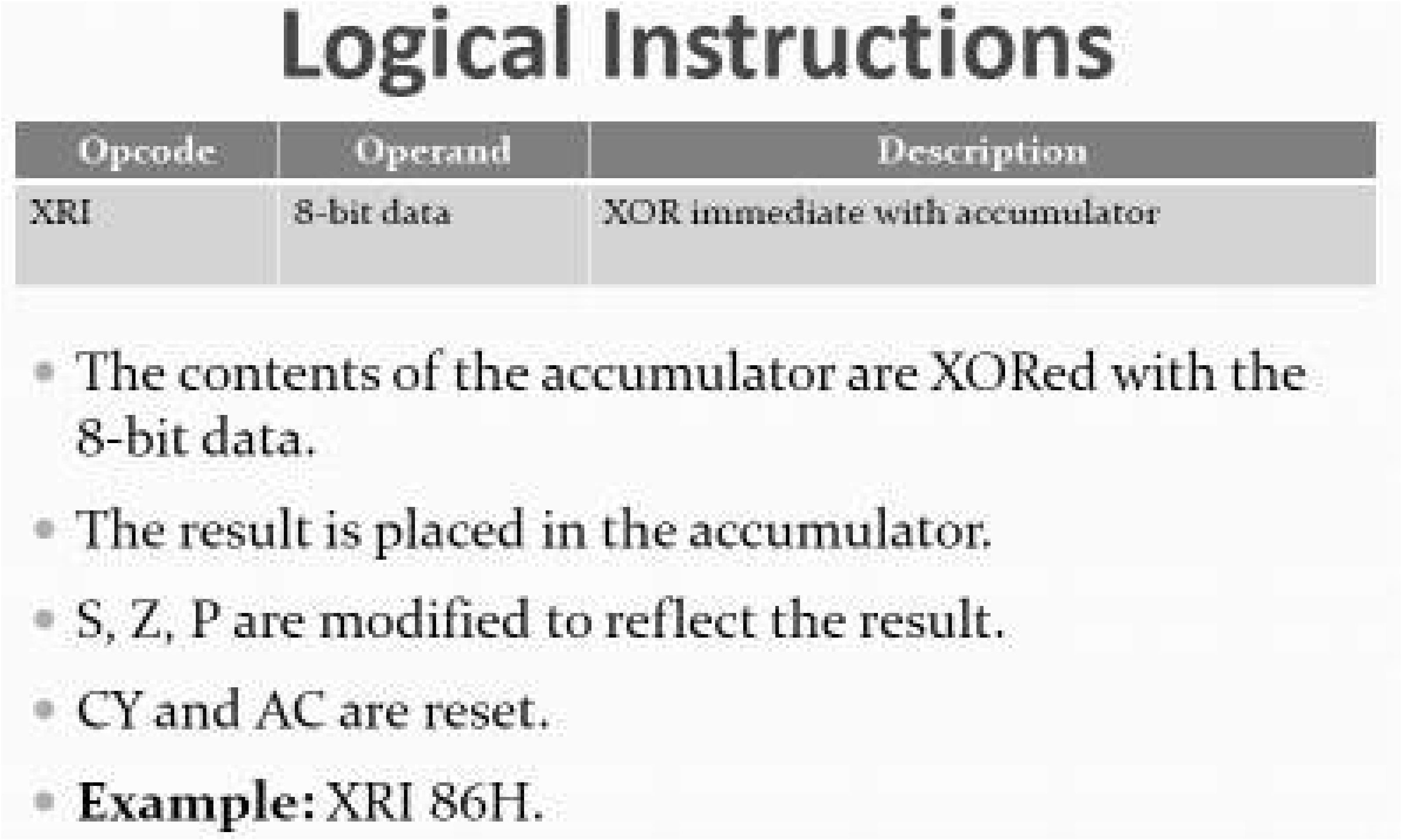
- The contents of the accumulator are logically ORed with the 8-bit data.
- The result is placed in the accumulator.
- S, Z, P are modified to reflect the result.
- CY and AC are reset.
- **Example:** ORI 86H.



Logical Instructions

Opcode	Operand	Description
XRA	R M	Logical XOR register or memory with accumulator

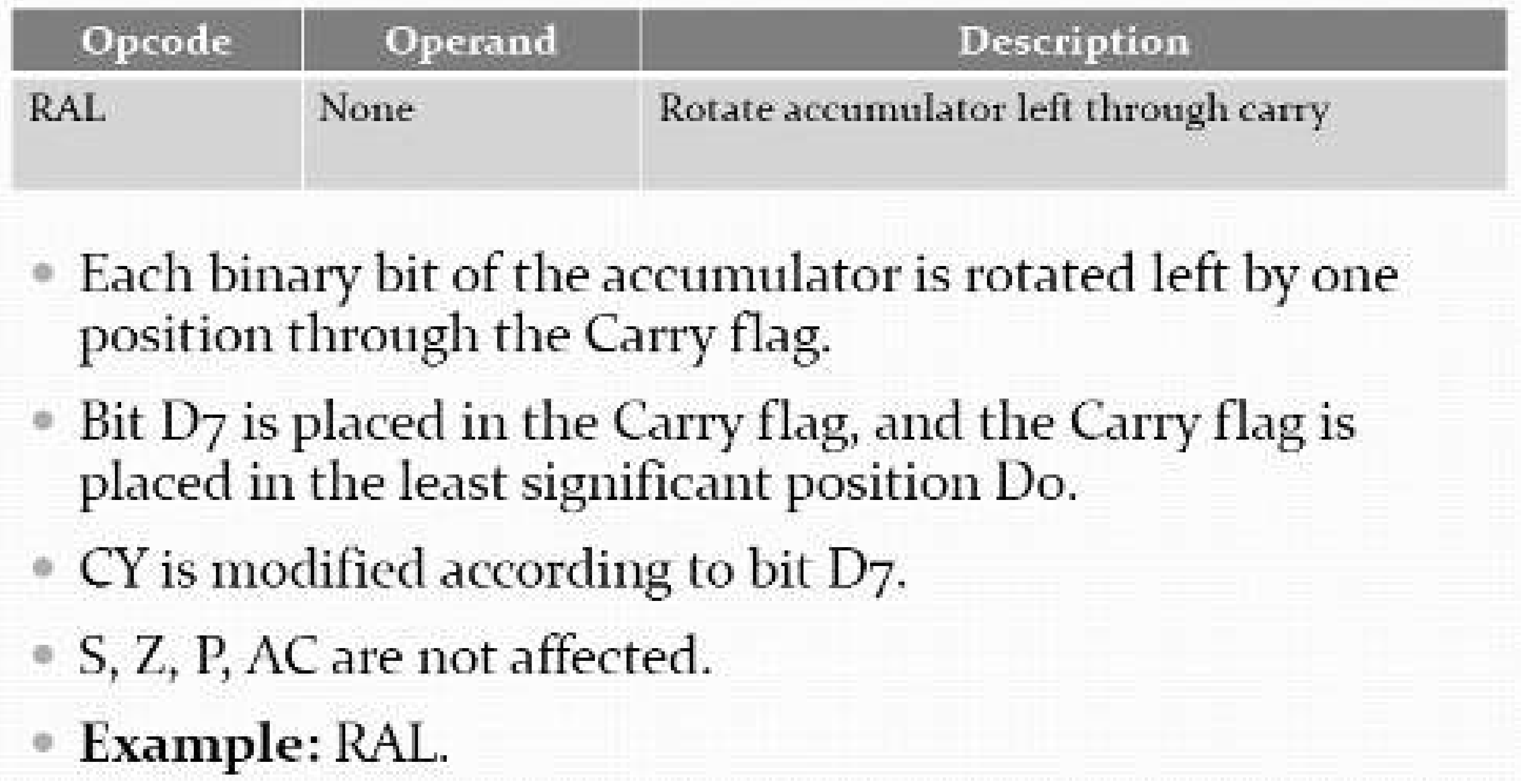
- The contents of the accumulator are XORed with the contents of the register or memory.
- The result is placed in the accumulator.
- If the operand is a memory location, its address is specified by the contents of H-L pair.
- S, Z, P are modified to reflect the result of the operation.
- CY and AC are reset.
- **Example:** XRA B or XRA M.



Logical Instructions

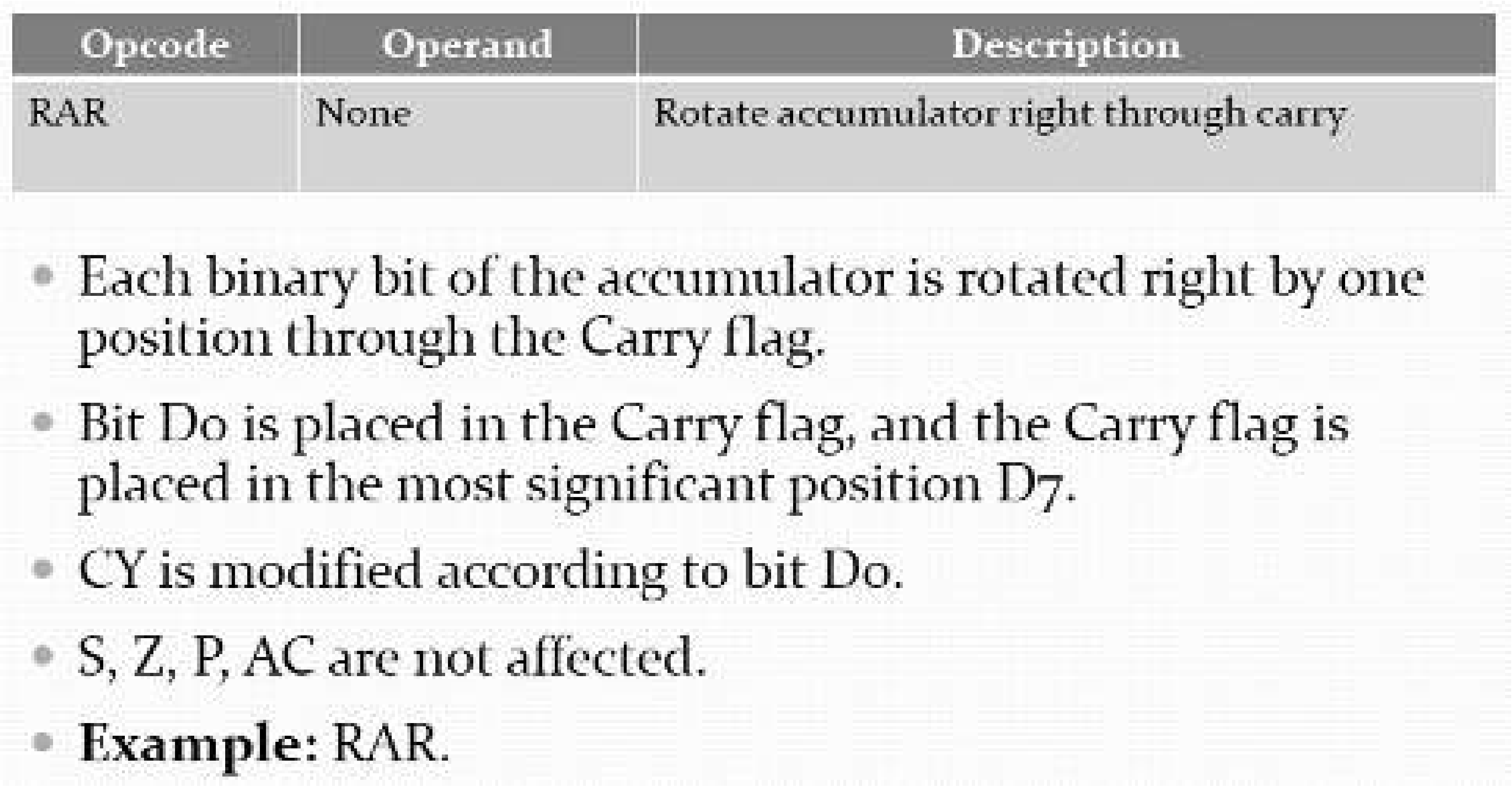
Opcode	Operand	Description
XRI	8-bit data	XOR immediate with accumulator

- The contents of the accumulator are XORed with the 8-bit data.
- The result is placed in the accumulator.
- S, Z, P are modified to reflect the result.
- CY and AC are reset.
- **Example:** XRI 86H.



Opcode	Operand	Description
RAL	None	Rotate accumulator left through carry

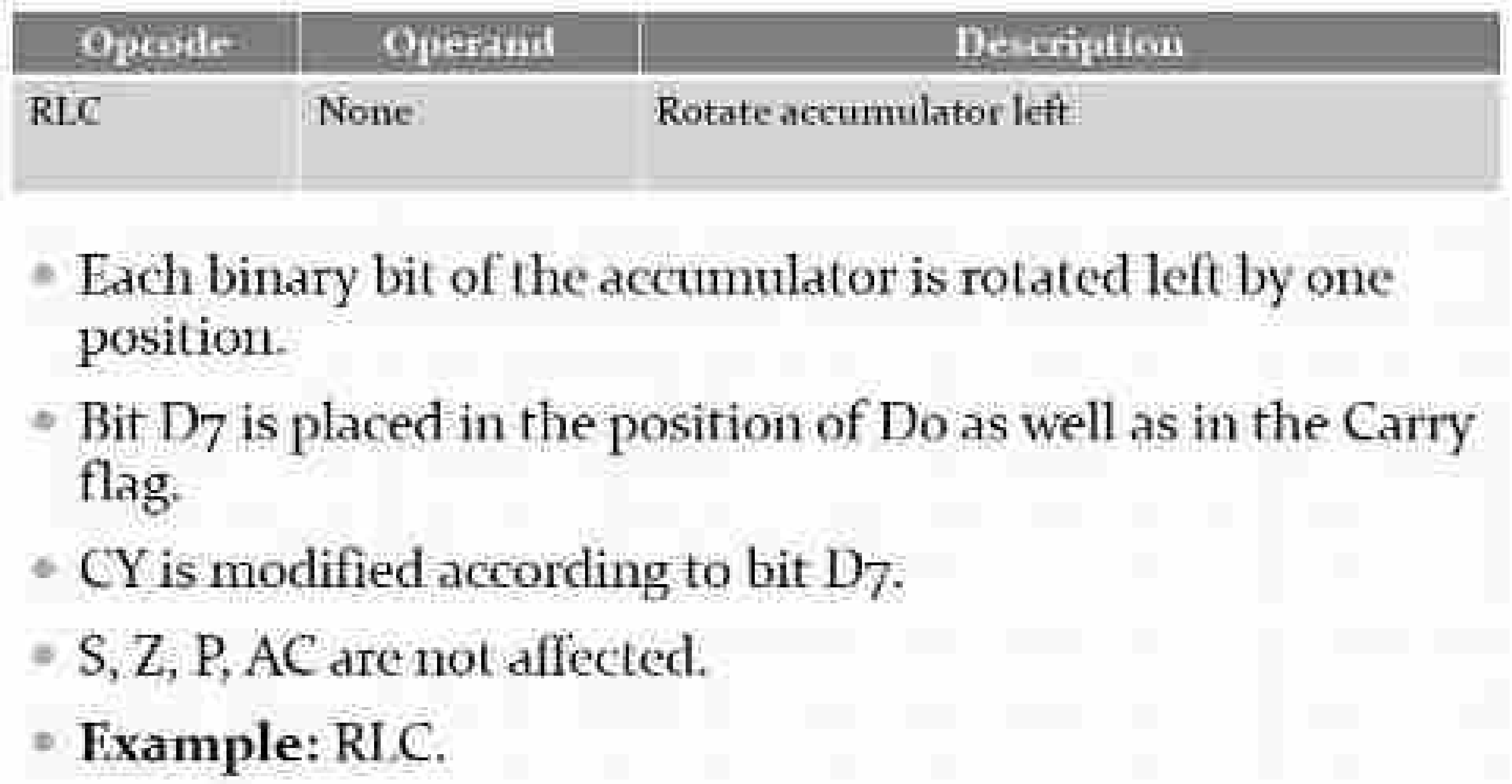
- Each binary bit of the accumulator is rotated left by one position through the Carry flag.
- Bit D7 is placed in the Carry flag, and the Carry flag is placed in the least significant position D0.
- CY is modified according to bit D7.
- S, Z, P, AC are not affected.
- **Example:** RAL.



Opcode	Operand	Description
RAR	None	Rotate accumulator right through carry

- Each binary bit of the accumulator is rotated right by one position through the Carry flag.
- Bit Do is placed in the Carry flag, and the Carry flag is placed in the most significant position D7.
- CY is modified according to bit Do.
- S, Z, P, AC are not affected.
- **Example:** RAR.

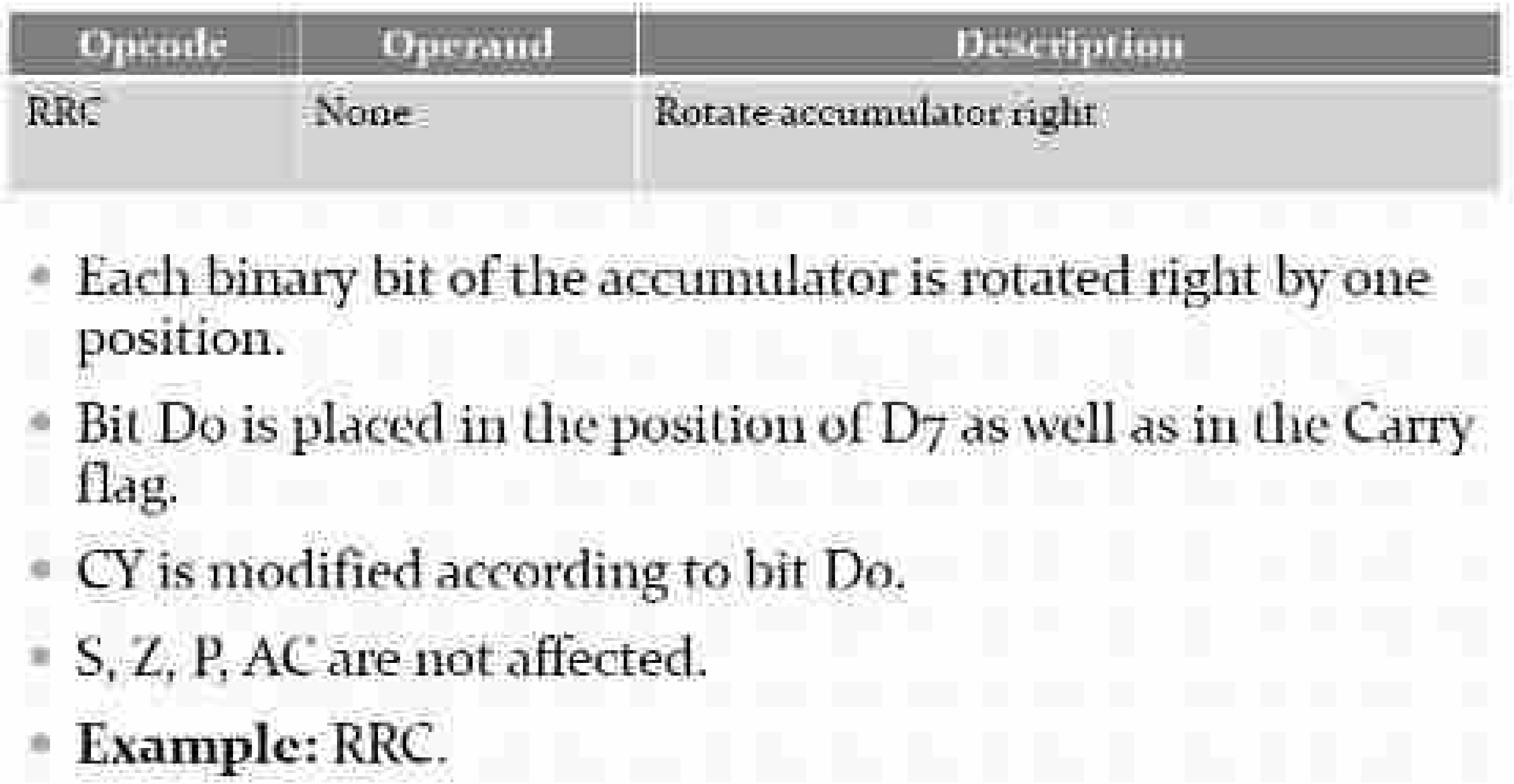
circular Left shift



Opcode	Operand	Description
RLC	None	Rotate accumulator left

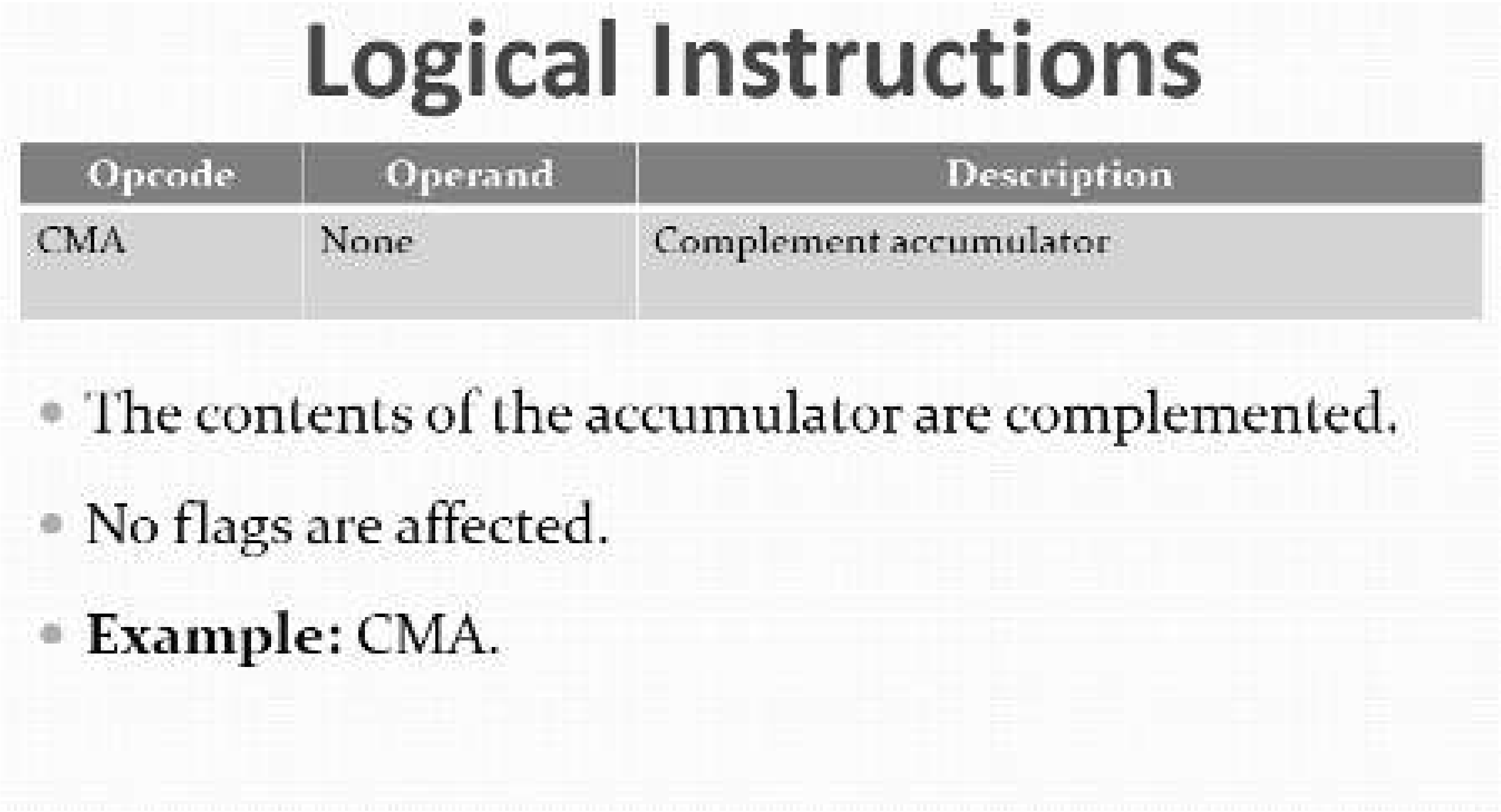
- Each binary bit of the accumulator is rotated left by one position.
- Bit D7 is placed in the position of Do as well as in the Carry flag.
- CY is modified according to bit D7.
- S, Z, P, AC are not affected.
- **Example:** RLC.

- circular right shift



Opcode	Operand	Description
RRC	None	Rotate accumulator right

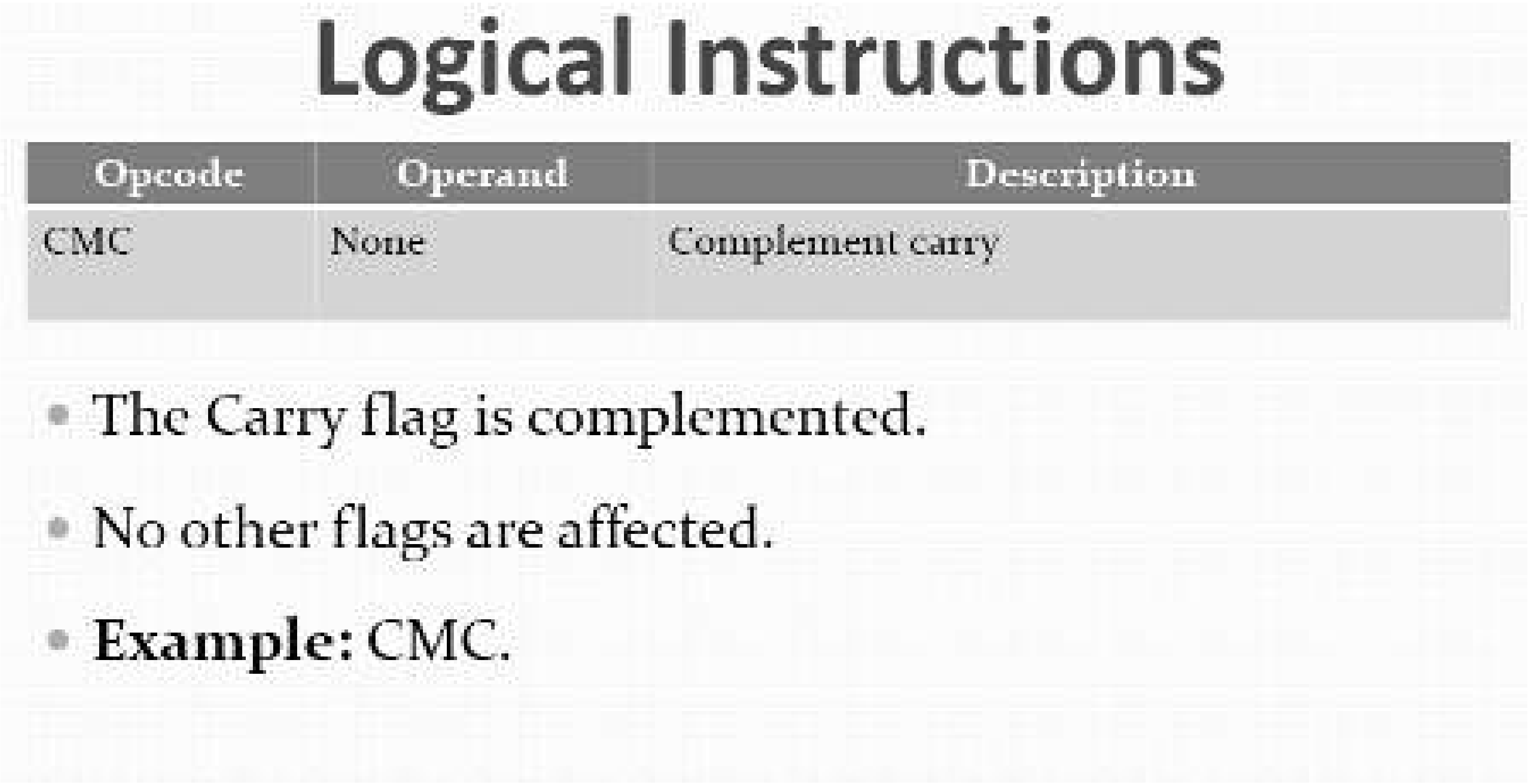
- Each binary bit of the accumulator is rotated right by one position.
- Bit Do is placed in the position of D7 as well as in the Carry flag.
- CY is modified according to bit Do.
- S, Z, P, AC are not affected.
- **Example:** RRC.



Logical Instructions

Opcode	Operand	Description
CMA	None	Complement accumulator

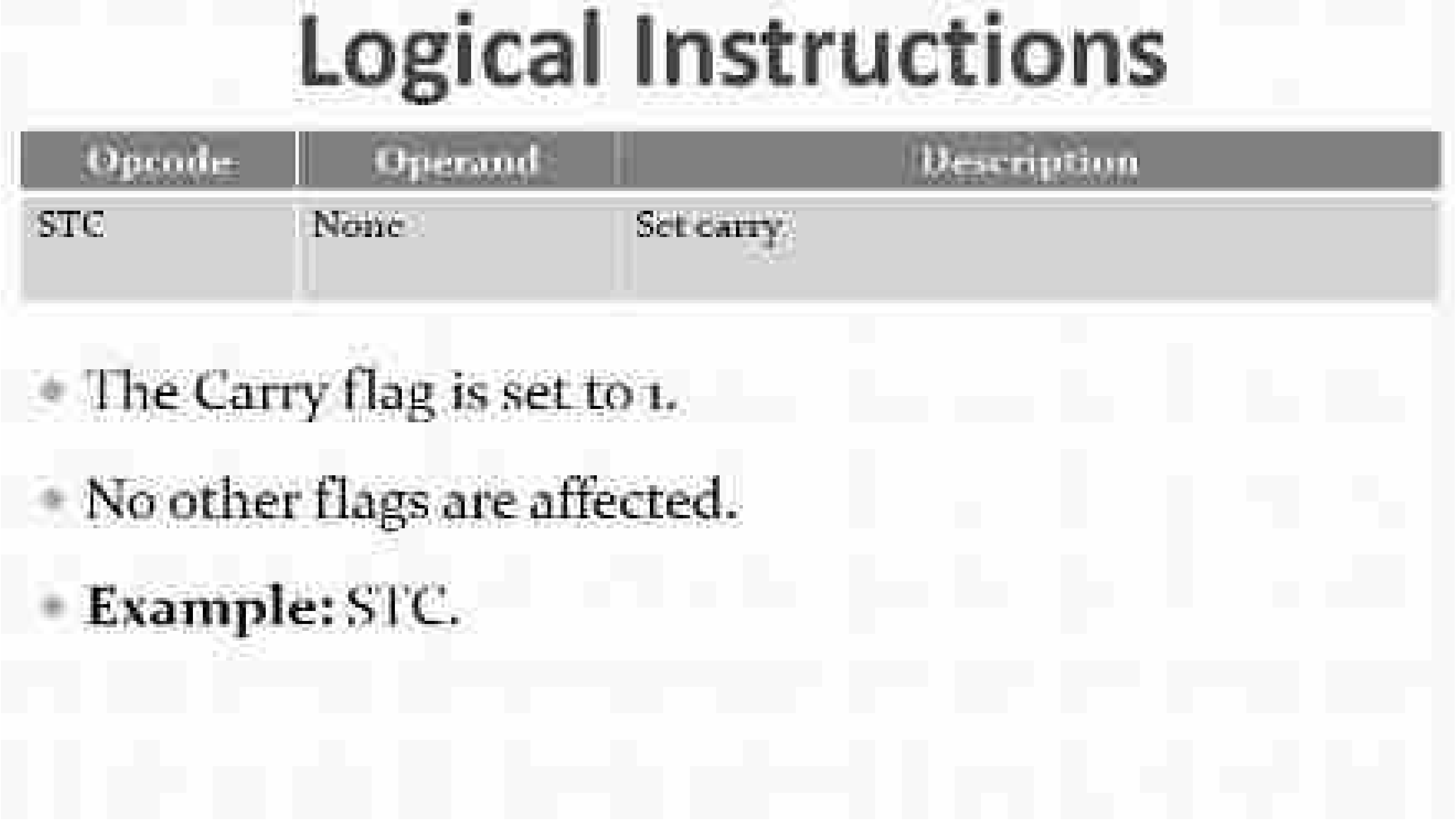
- The contents of the accumulator are complemented.
- No flags are affected.
- **Example:** CMA.



Logical Instructions

Opcode	Operand	Description
CMC	None	Complement carry

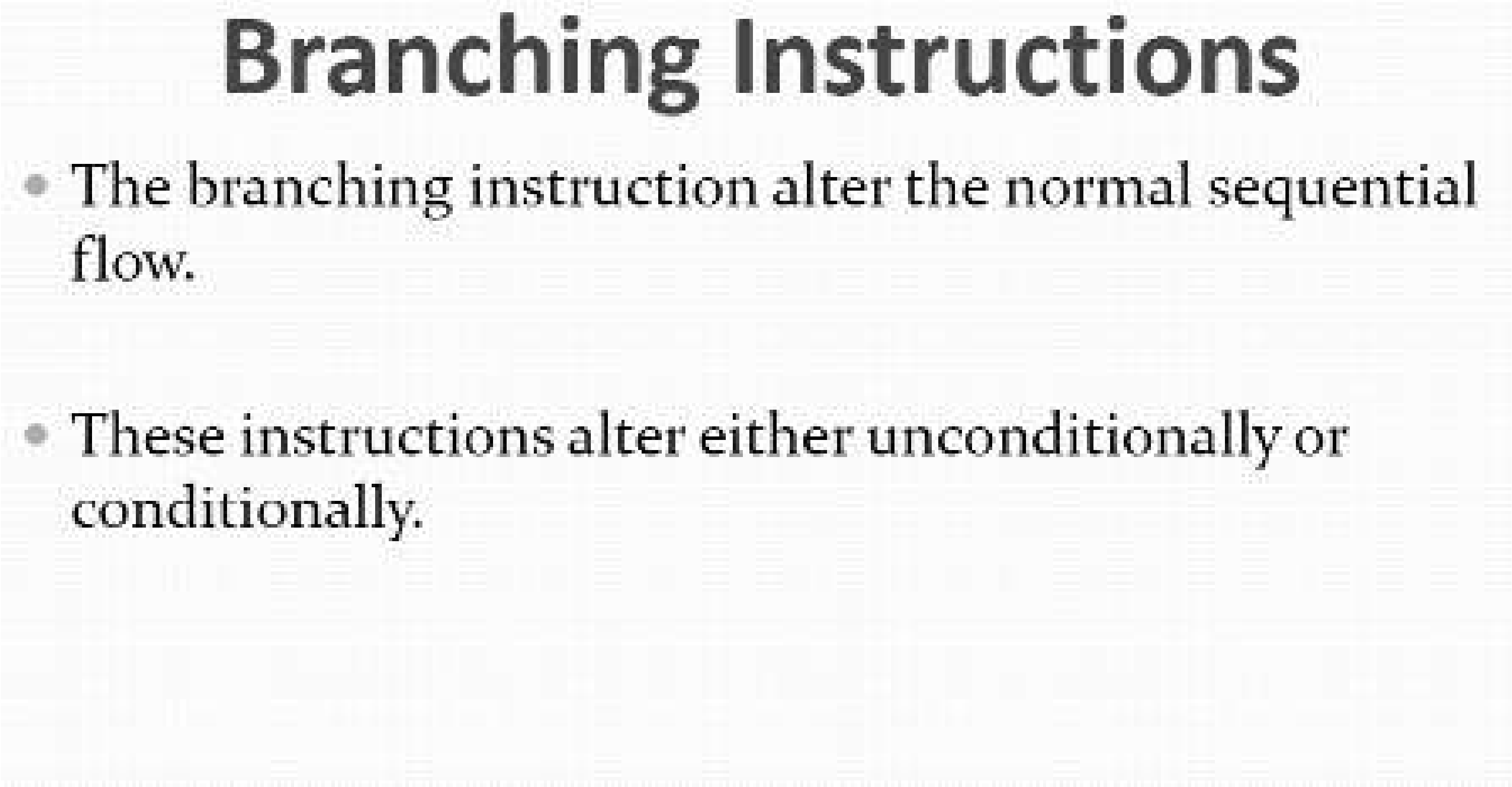
- The Carry flag is complemented.
- No other flags are affected.
- **Example:** CMC.



Logical Instructions

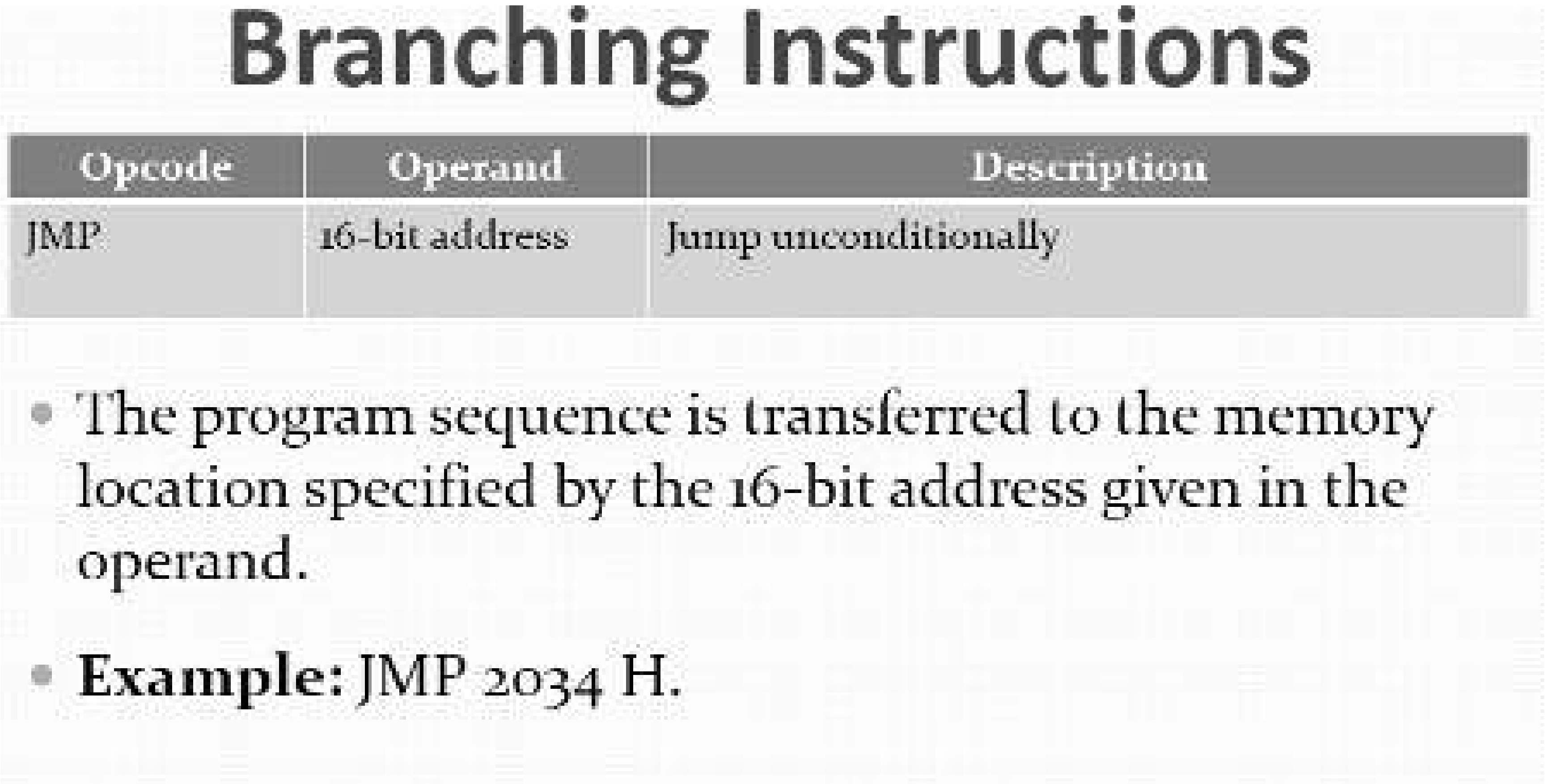
Opcode	Operand	Description
STC	None	Set carry

- The Carry flag is set to 1.
- No other flags are affected.
- **Example:** STC.



Branching Instructions

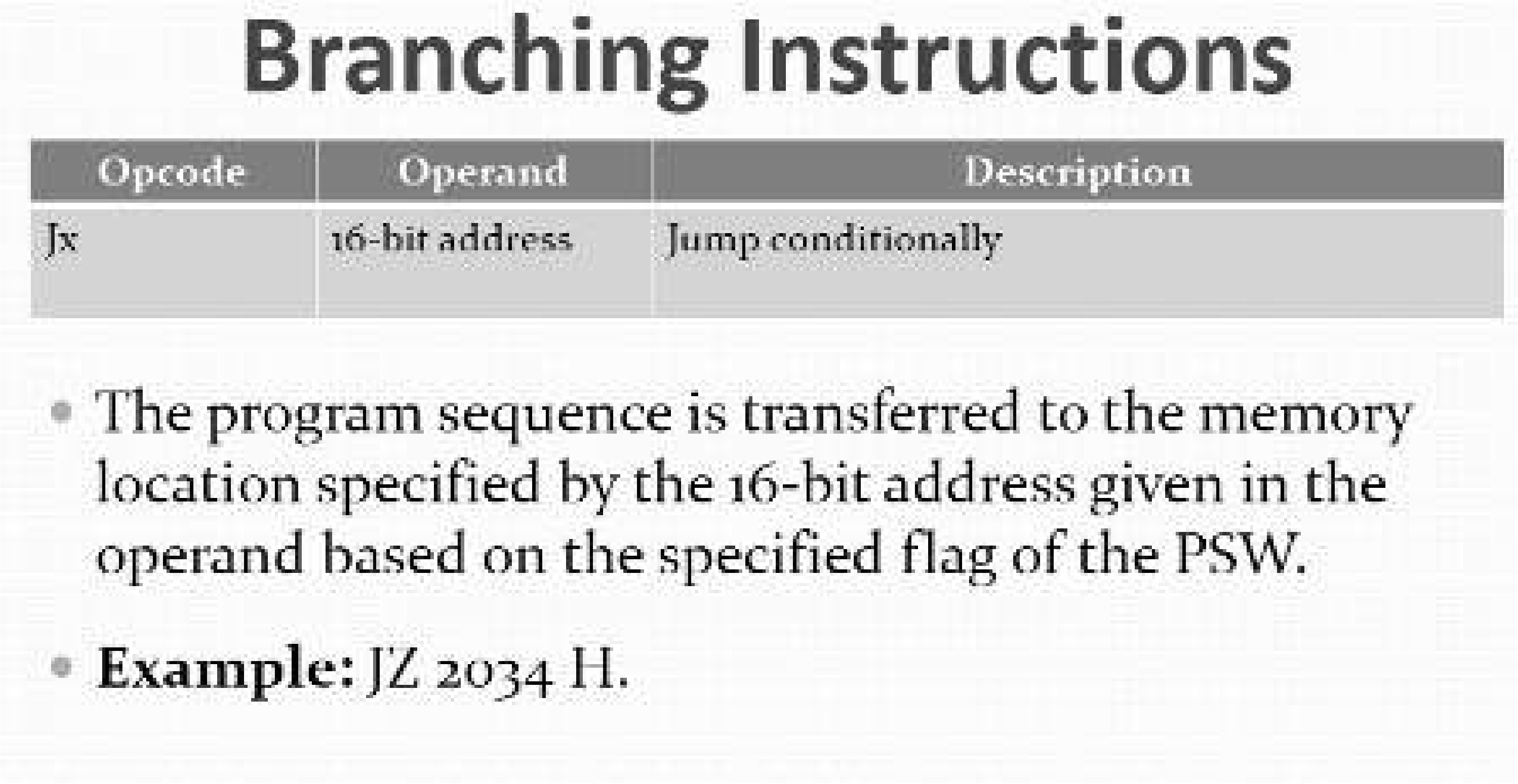
- The branching instruction alter the normal sequential flow.
- These instructions alter either unconditionally or conditionally.



Branching Instructions

Opcode	Operand	Description
JMP	16-bit address	Jump unconditionally

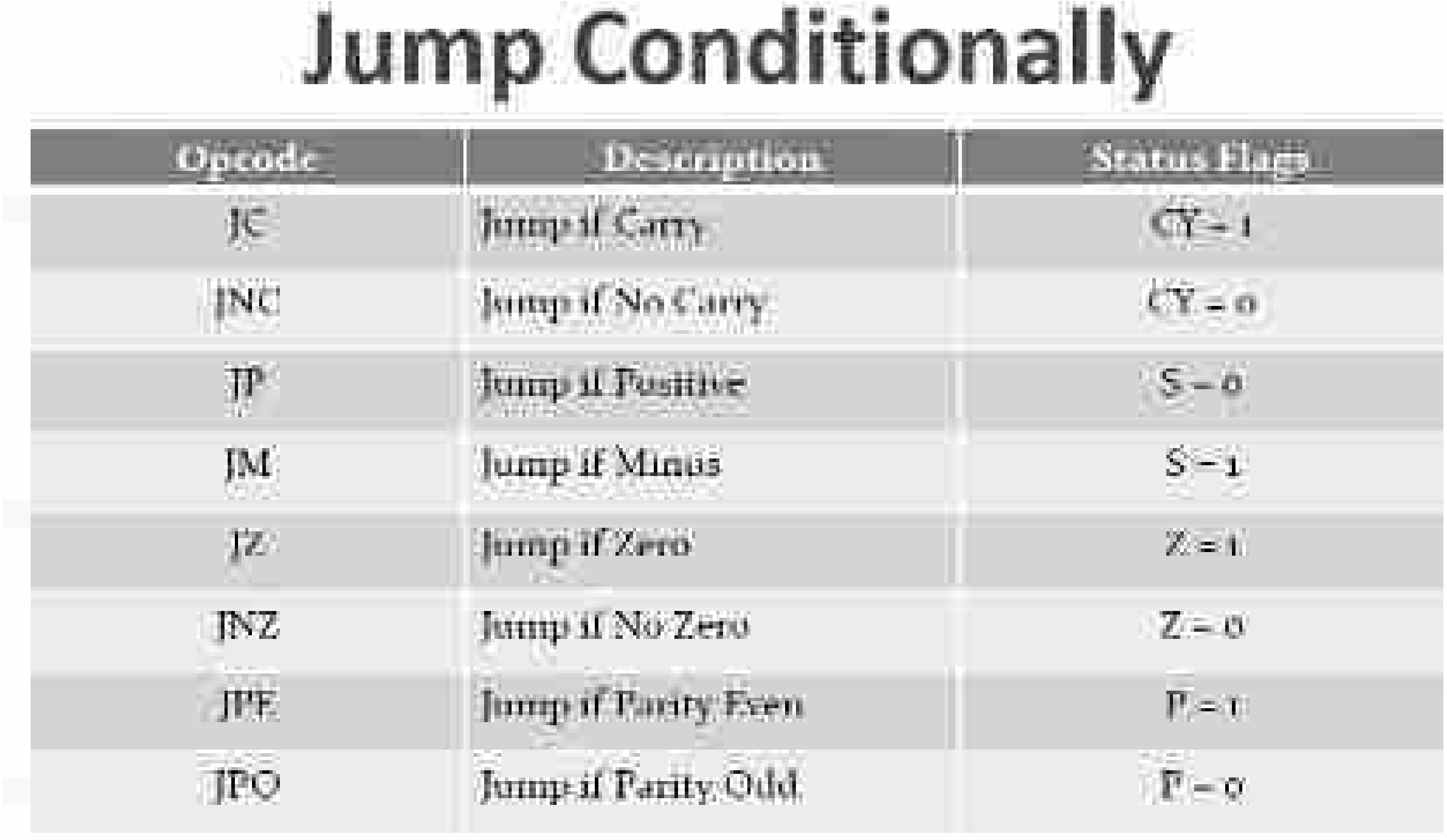
- The program sequence is transferred to the memory location specified by the 16-bit address given in the operand.
- **Example:** JMP 2034 H.



Branching Instructions

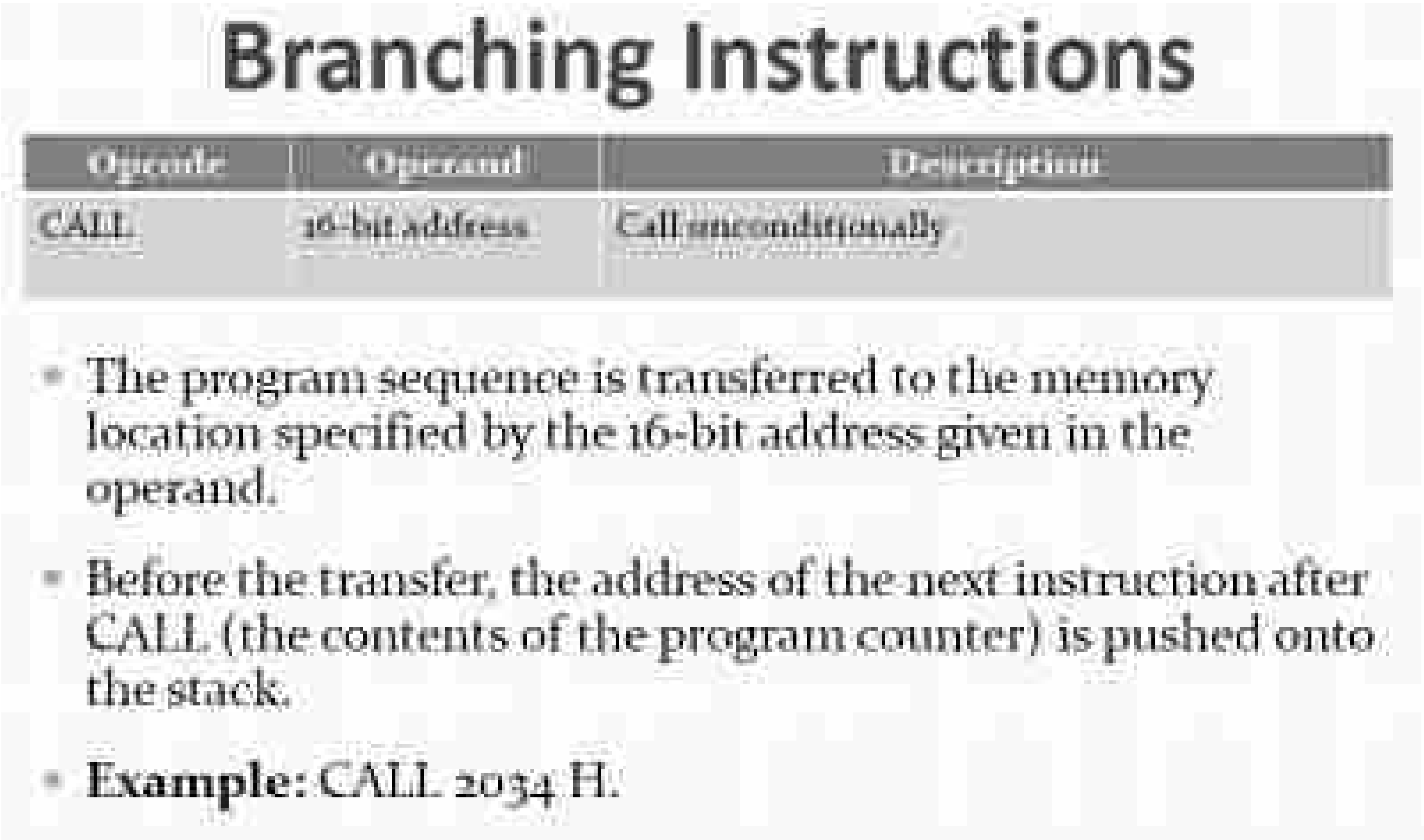
Opcode	Operand	Description
Jx	16-bit address	Jump conditionally

- The program sequence is transferred to the memory location specified by the 16-bit address given in the operand based on the specified flag of the PSW.
- **Example:** JZ 2034 H.



Jump Conditionally

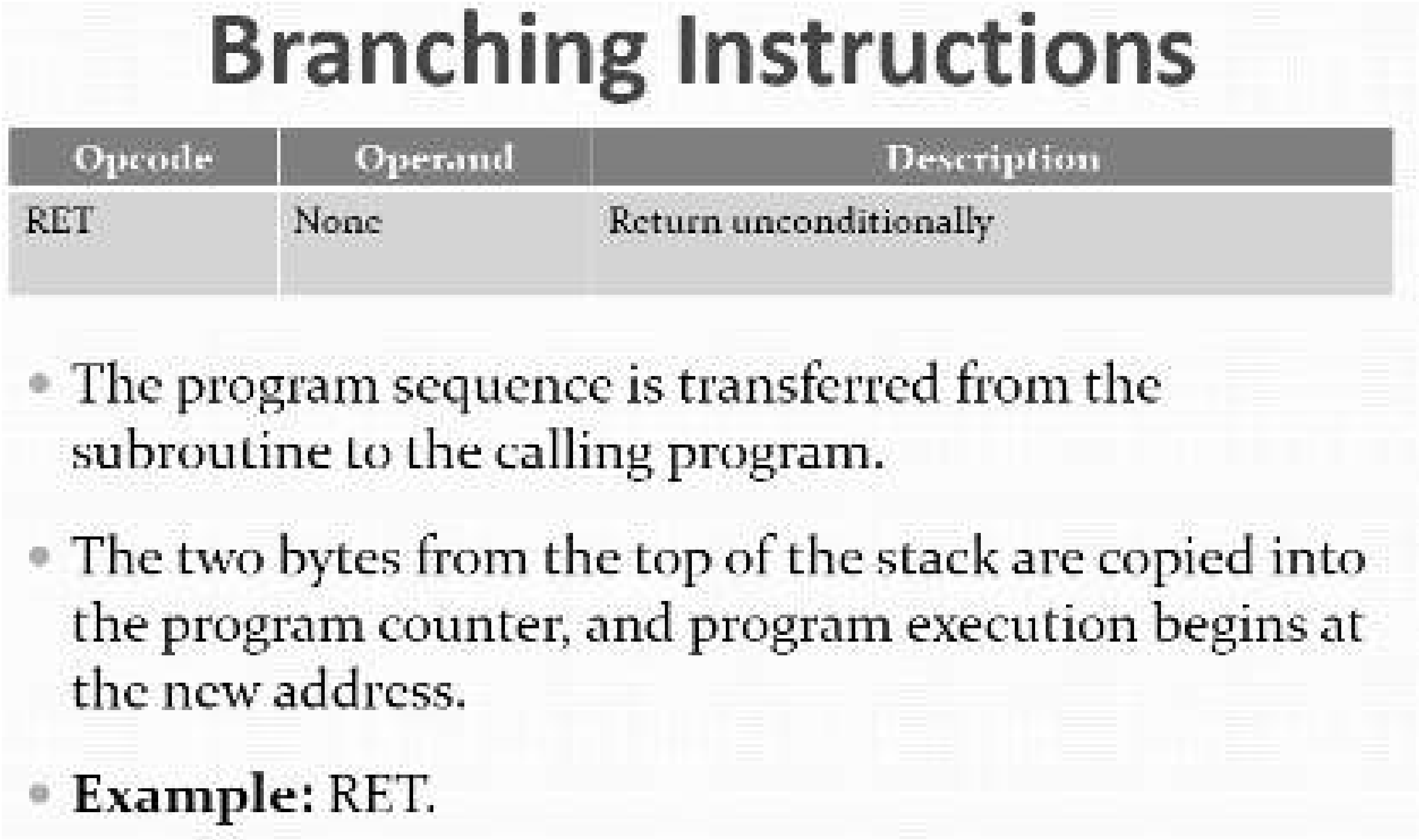
Opcode	Description	Status Flags
JC	Jump if Carry	CY = 1
JNC	Jump if No Carry	CY = 0
JP	Jump if Positive	S = 0
JM	Jump if Minus	S = 1
JZ	Jump if Zero	Z = 1
JNZ	Jump if No Zero	Z = 0
JPE	Jump if Parity Even	P = 1
JPO	Jump if Parity Odd	P = 0



Branching Instructions

Opcode	Operand	Description
CALL	16-bit address	Call unconditionally

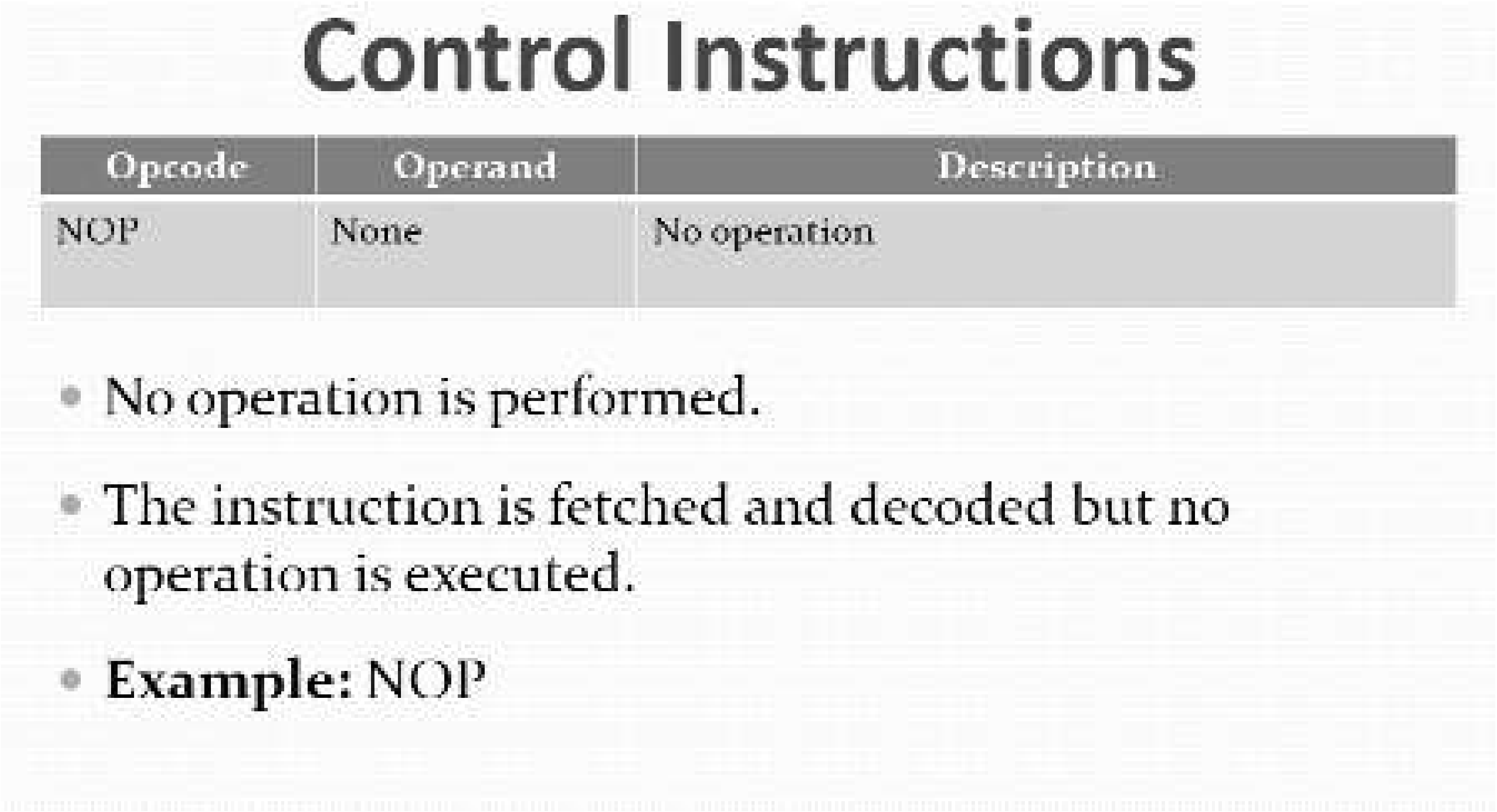
- The program sequence is transferred to the memory location specified by the 16-bit address given in the operand.
- Before the transfer, the address of the next instruction after CALL (the contents of the program counter) is pushed onto the stack.
- **Example:** CALL 2034 H.



Branching Instructions

Opcode	Operand	Description
RET	None	Return unconditionally

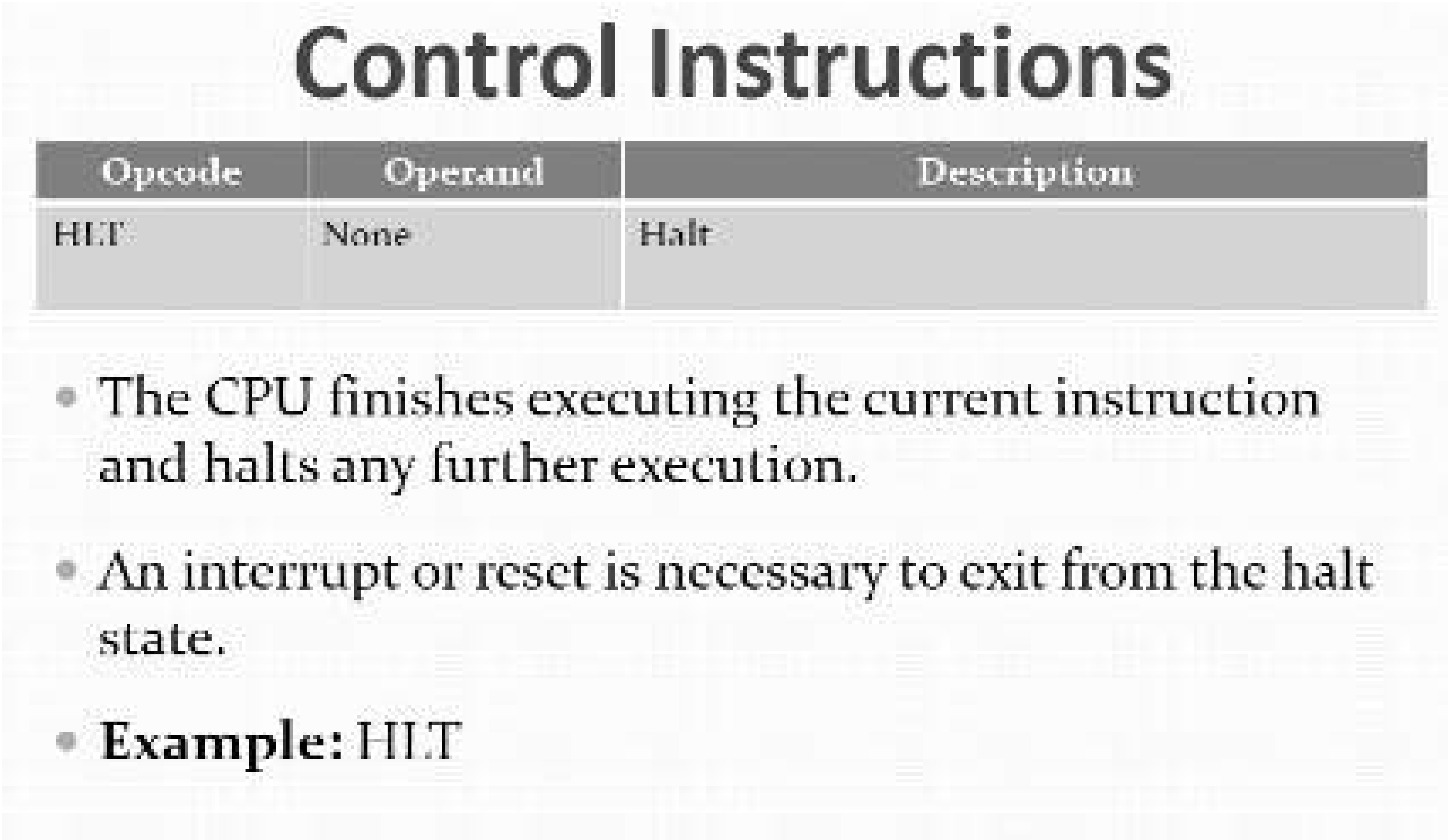
- The program sequence is transferred from the subroutine to the calling program.
- The two bytes from the top of the stack are copied into the program counter, and program execution begins at the new address.
- **Example:** RET.



Control Instructions

Opcode	Operand	Description
NOP	None	No operation

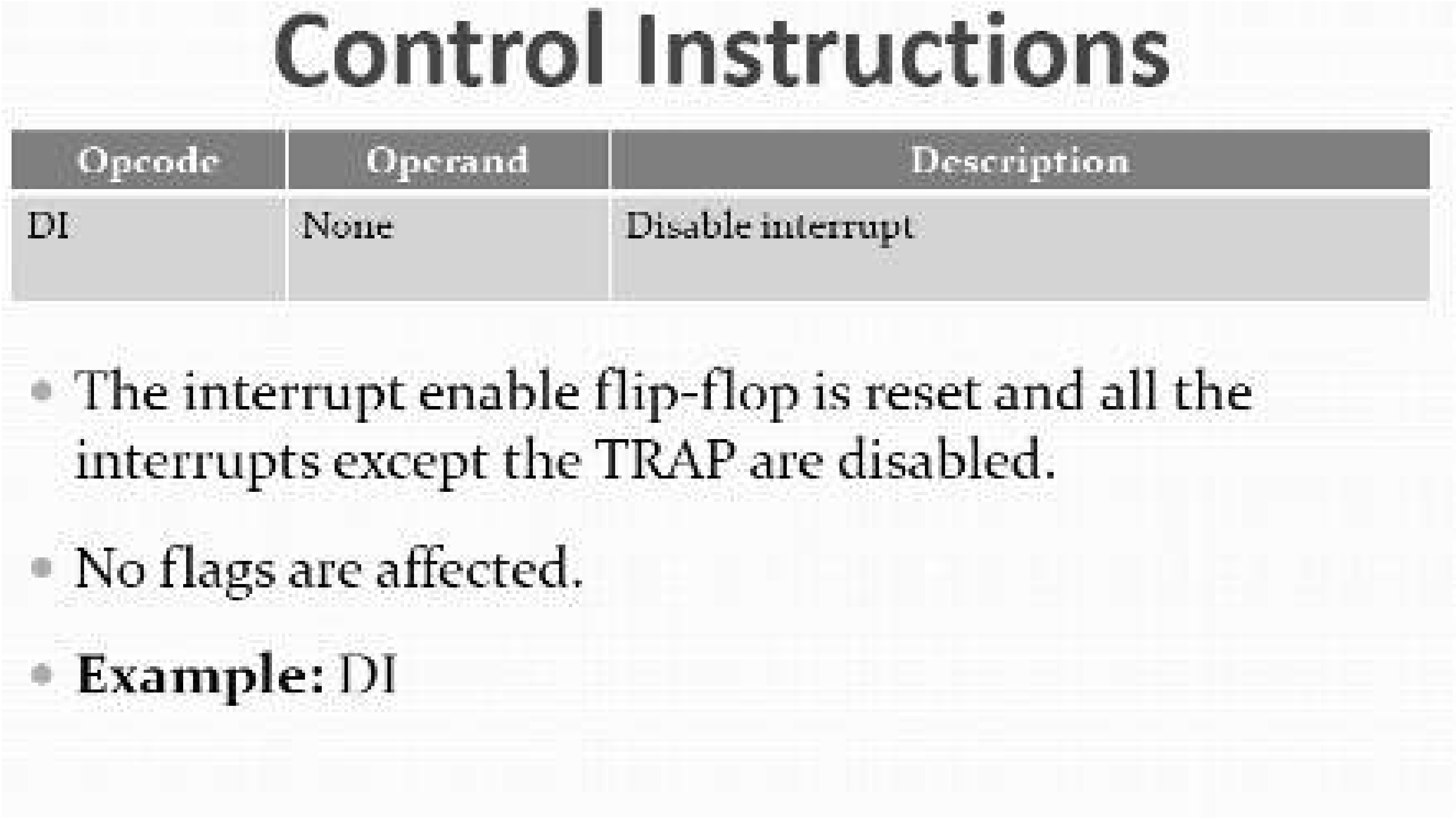
- No operation is performed.
- The instruction is fetched and decoded but no operation is executed.
- **Example:** NOP



Control Instructions

Opcode	Operand	Description
HLT	None	Halt

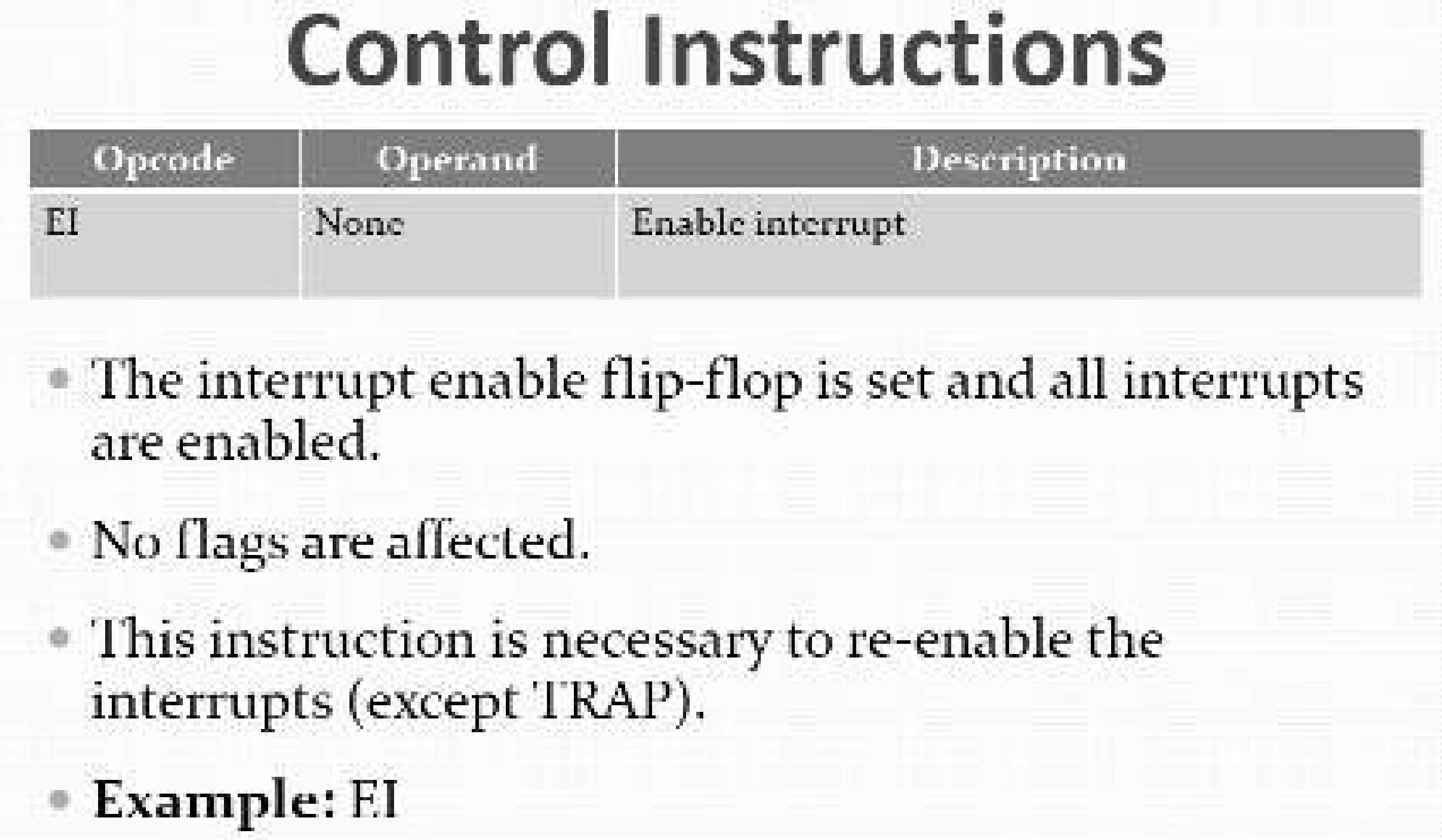
- The CPU finishes executing the current instruction and halts any further execution.
- An interrupt or reset is necessary to exit from the halt state.
- **Example:** HLT



Control Instructions

Opcode	Operand	Description
DI	None	Disable interrupt

- The interrupt enable flip-flop is reset and all the interrupts except the TRAP are disabled.
- No flags are affected.
- **Example:** DI



Control Instructions

Opcode	Operand	Description
EI	None	Enable interrupt

- The interrupt enable flip-flop is set and all interrupts are enabled.
- No flags are affected.
- This instruction is necessary to re-enable the interrupts (except TRAP).
- **Example:** EI

Summary – Data transfer

MOV	Move
MVI	Move Immediate
LDA	Load Accumulator Directly from Memory
STA	Store Accumulator Directly in Memory
LHLD	Load H & L Registers Directly from Memory
SHLD	Store H & L Registers Directly in Memory

Summary Data transfer

- An 'X' in the name of a data transfer instruction implies that it deals with a register pair (16-bits);

LXI	Load Register Pair with Immediate data
LDAX	Load Accumulator from Address in Register Pair
STAX	Store Accumulator in Address in Register Pair
XCHG	Exchange H & L with D & E
XTHL	Exchange Top of Stack with H & L

Summary - Arithmetic Group

- Add, Subtract, Increment / Decrement data in registers or memory.

- ADD Add to Accumulator
- ADI Add Immediate Data to Accumulator
- ADC Add to Accumulator Using Carry Flag
- ACI Add Immediate data to Accumulator Using Carry
- SUB Subtract from Accumulator
- SUI Subtract Immediate Data from Accumulator
- SBB Subtract from Accumulator Using Borrow (Carry) Flag
- SBI Subtract Immediate from Accumulator Using Borrow (Carry) Flag
- INR Increment Specified Byte by One
- DCR Decrement Specified Byte by One
- INX Increment Register Pair by One
- DCX Decrement Register Pair by One
- DAD Double Register Add; Add Content of Register Pair to H & L Register Pair

Summary Logical Group

- This group performs logical (Boolean) operations on data in registers and memory and on condition flags.
- These instructions enable you to set specific bits in the accumulator ON or OFF.

ANA	Logical AND with Accumulator
ANI	Logical AND with Accumulator Using Immediate Data
ORA	Logical OR with Accumulator
OR	Logical OR with Accumulator Using Immediate Data
XRA	Exclusive Logical OR with Accumulator
XRI	Exclusive OR Using Immediate Data

- The Compare instructions compare the content of an 8-bit value with the contents of the accumulator;

- CMP Compare
- CPI Compare Using Immediate Data

- The rotate instructions shift the contents of the accumulator one bit position to the left or right:

- RLC Rotate Accumulator Left
- RRC Rotate Accumulator Right
- RAL Rotate Left Through Carry
- RAR Rotate Right Through Carry

- Complement and carry flag instructions:

- CMA Complement Accumulator
- CMC Complement Carry Flag
- STC Set Carry Flag

Summary - Branch Group

- Unconditional branching
  - JMP Jump
  - CALL Call
  - RET Return
- Conditions
  - NZ Not Zero (Z = 0)
  - Z Zero (Z = 1)
  - NC No Carry (C = 0)
  - C Carry (C = 1)
  - PO Parity Odd (P = 0)
  - PE Parity Even (P = 1)
  - P Plus (S = 0)
  - M Minus (S = 1)
- Conditional branching

Summary - Stack

PUSH	Push Two bytes of Data onto the Stack
POP	Pop Two Bytes of Data off the Stack
XTHL	Exchange Top of Stack with H & L
SPHL	Move content of H & L to Stack Pointer

I/0 instructions

IN	Initiate Input Operation
OUT	Initiate Output Operation

Summary -Machine Control instructions

EI	Enable Interrupt System
DI	Disable Interrupt System
HLT	Halt
NOP	No Operation

8086 MICROPROCESSOR

Block diagram of 8086

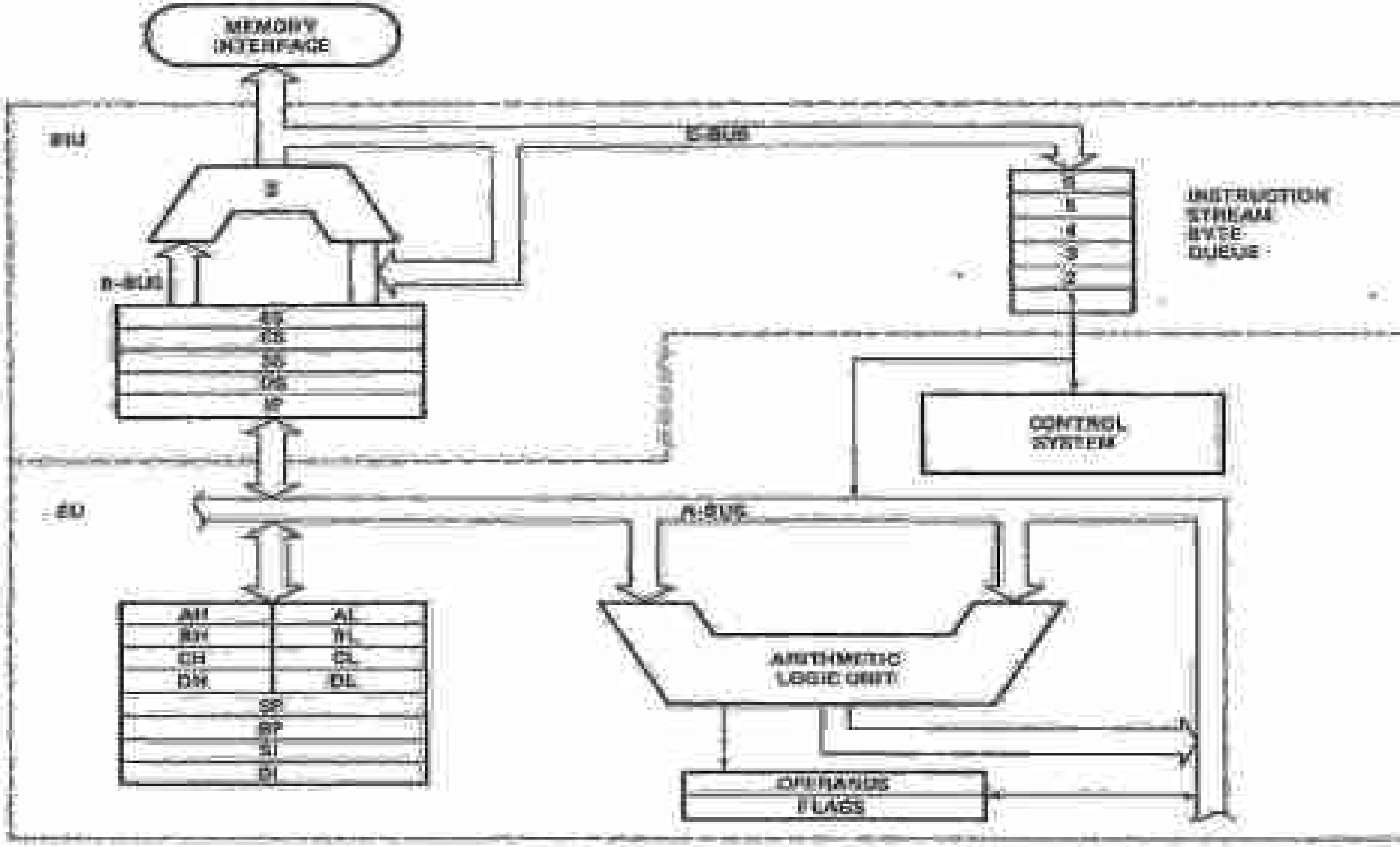
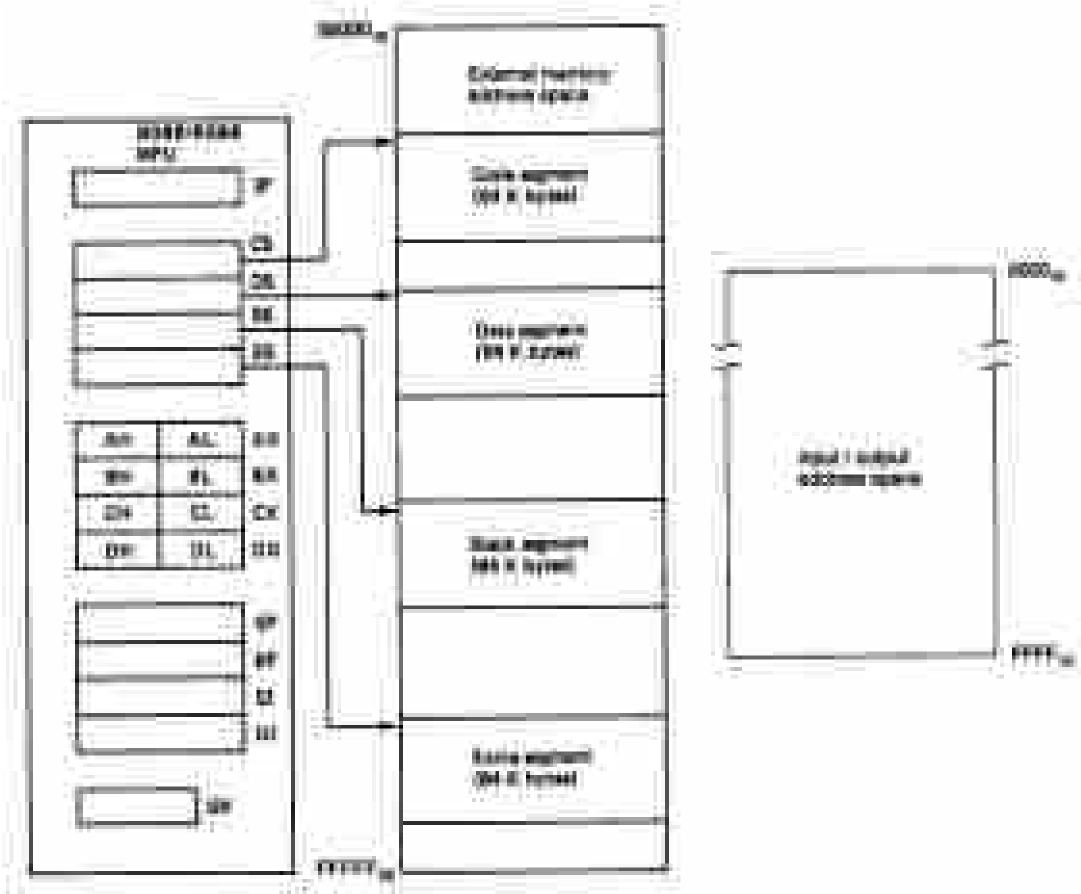


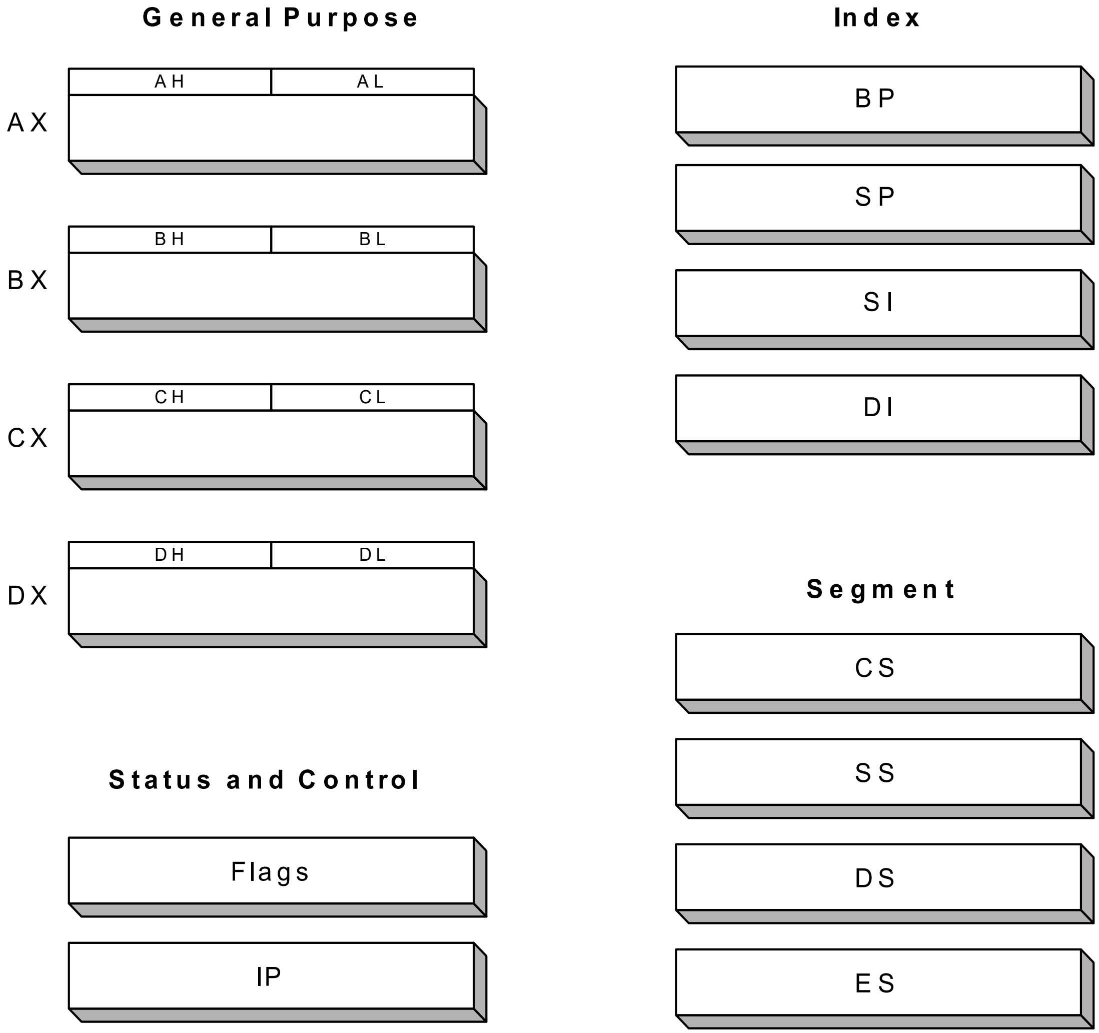
FIGURE 2-7 8086 internal block diagram. (*Intel Corp.*)

Software Model of the 8086 Microprocessors



00000₁₆
8088/8086 MPU
IP
CS
DS
SS
ES
AH
AL
AX
BH
BL
BX
CH
CL
CX
DH
DL
DX
SP
BP
SI
DI
SR
External memory address space
Code segment (64 K bytes)
Data segment (64 K bytes)
Stack segment (64 K bytes)
Extra segment (64 K bytes)
FFFFF₁₆
0000₁₆
Input / output address space
FFFF₁₆

8086 Registers



General Purpose

AX: AH, AL

BX: BH, BL

CX: CH, CL

DX: DH, DL

Status and Control

Flags

IP

Index

BP

SP

SI

DI

Segment

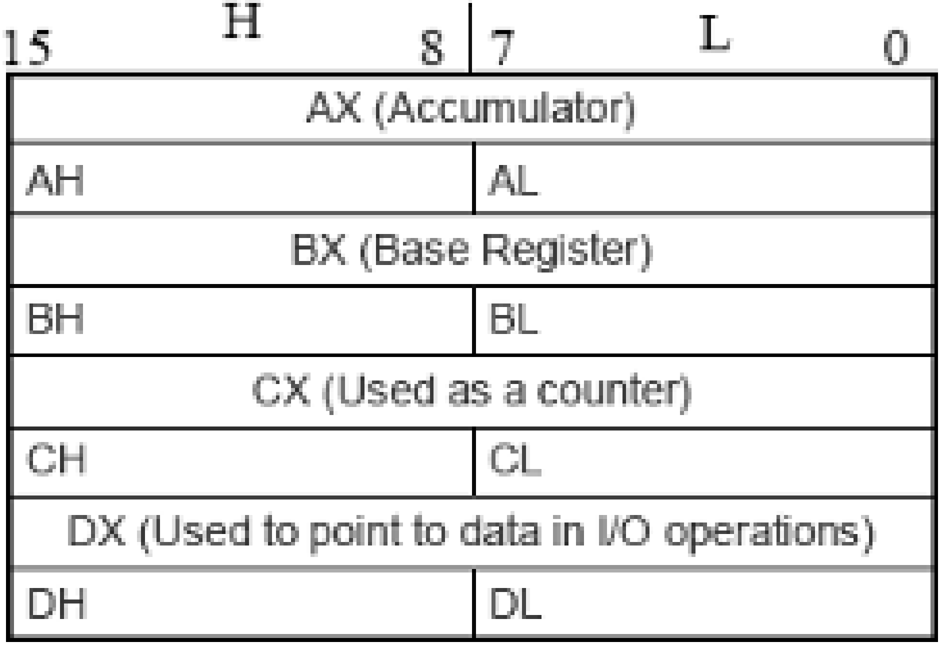
CS

SS

DS

ES

General Purpose Registers



15 H 8	7 L 0
AX (Accumulator)	
AH	AL
BX (Base Register)	
BH	BL
CX (Used as a counter)	
CH	CL
DX (Used to point to data in I/O operations)	
DH	DL

AX - the Accumulator
BX - the Base Register
CX - the Count Register
DX - the Data Register

- Normally used for storing temporary results
- Each of the registers is 16 bits wide **(AX, BX, CX, DX)**
- Can be accessed as either 16 or 8 bits AX, AH, AL

General Purpose Registers

- **AX**
  - Accumulator Register
  - Preferred register to use in arithmetic, logic and data transfer instructions because it generates the shortest Machine Language Code
  - Must be used in multiplication and division operations
  - Must also be used in I/O operations

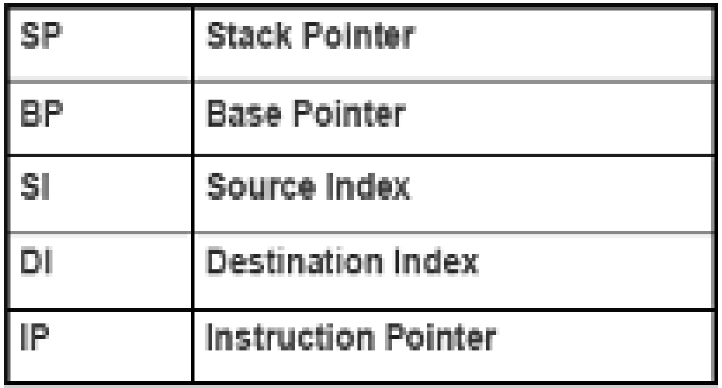
- **BX**
  - Base Register
  - Also serves as an address register

General Purpose Registers

- **CX**
  - Count register
  - Used as a loop counter
  - Used in shift and rotate operations

- **DX**
  - Data register
  - Used in multiplication and division
  - Also used in I/O operations

Pointer and Index Registers



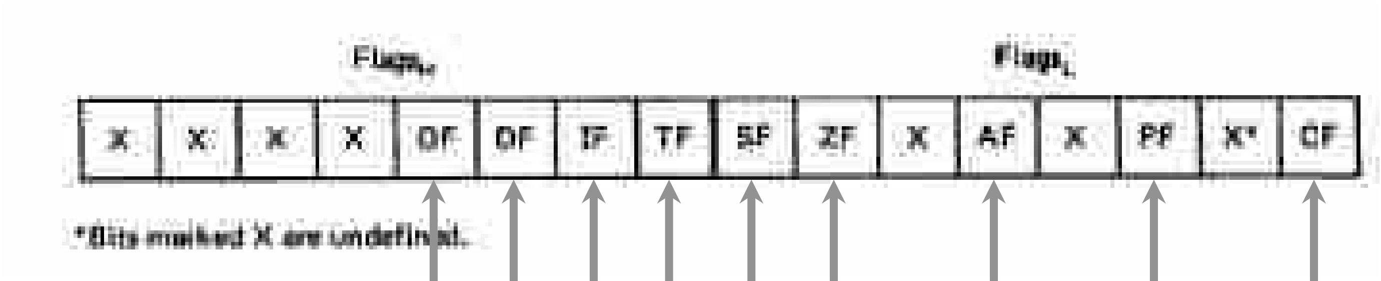
SP	Stack Pointer
BP	Base Pointer
SI	Source Index
DI	Destination Index
IP	Instruction Pointer

- All 16 bits wide, L/H bytes are not accessible

- Used as memory pointers
  - Example: MOV AH, [SI]
    - *Move the byte stored in memory location whose address is contained in register SI to register AH*

- IP is not under direct control of the programmer

Flag Register



Flags$_H$ Flags$_L$

X	X	X	X	OF	DF	IF	TF	SF	ZF	X	AF	X	PF	X*	CF

*Bits marked X are undefined.

Overflow

Direction

Interrupt enable

Trap

Sign

Zero

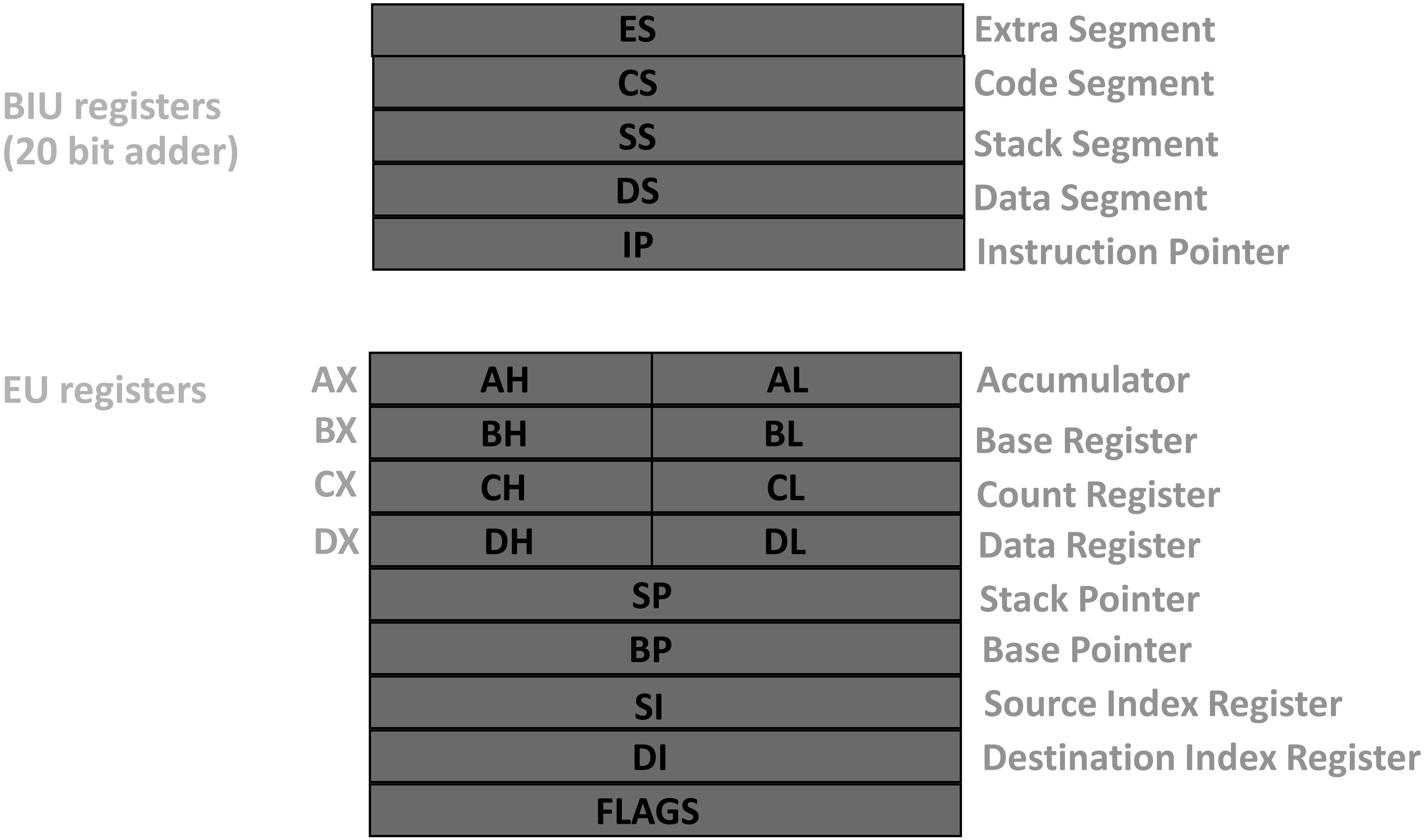
Auxiliary Carry

Parity

Carry

6 are status flags
3 are control flag

8086 Programmer’s Model



BIU registers (20 bit adder)

ES	Extra Segment
CS	Code Segment
SS	Stack Segment
DS	Data Segment
IP	Instruction Pointer

EU registers

AX	AH	AL	Accumulator
BX	BH	BL	Base Register
CX	CH	CL	Count Register
DX	DH	DL	Data Register
	SP		Stack Pointer
	BP		Base Pointer
	SI		Source Index Register
	DI		Destination Index Register
	FLAGS		

The Stack

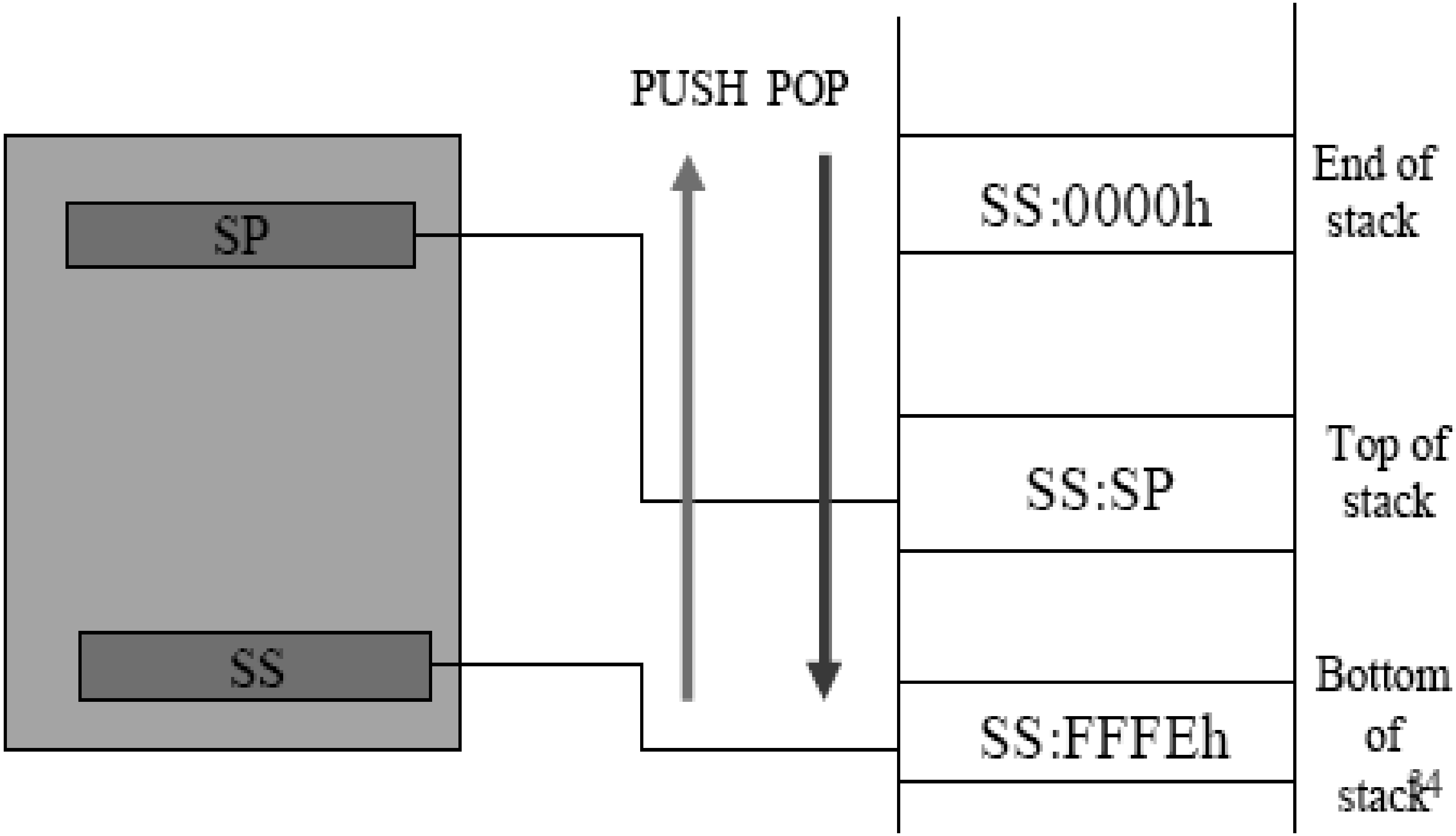
- The stack is used for temporary storage of information such as data or addresses.

- When a **CALL** is executed, the 8086 automatically **PUSH**es the current value of CS and IP onto the stack.

- Other registers can also be pushed

- Before return from the **subroutine**, **POP** instructions can be used to pop values back from the stack into the corresponding registers.

The Stack



SP

SS

PUSH POP

SS:0000h

End of stack

SS:SP

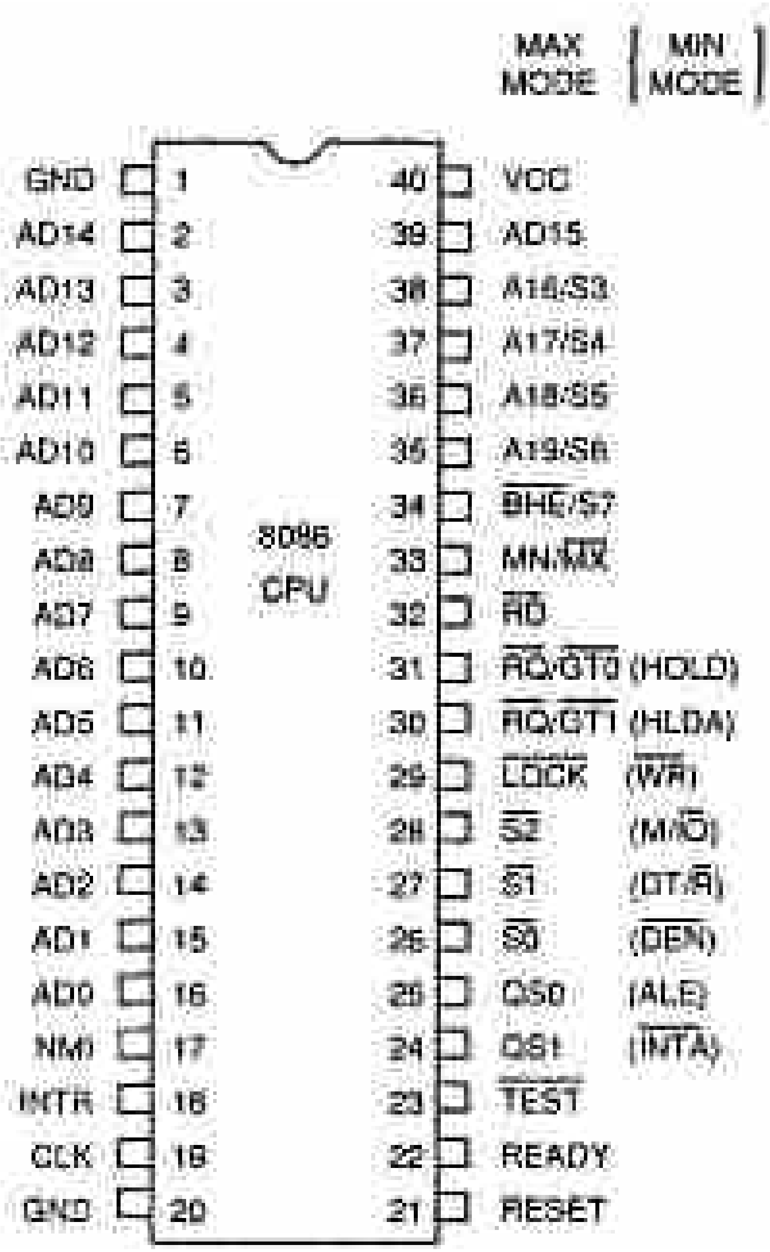
Top of stack

SS:FFFEh

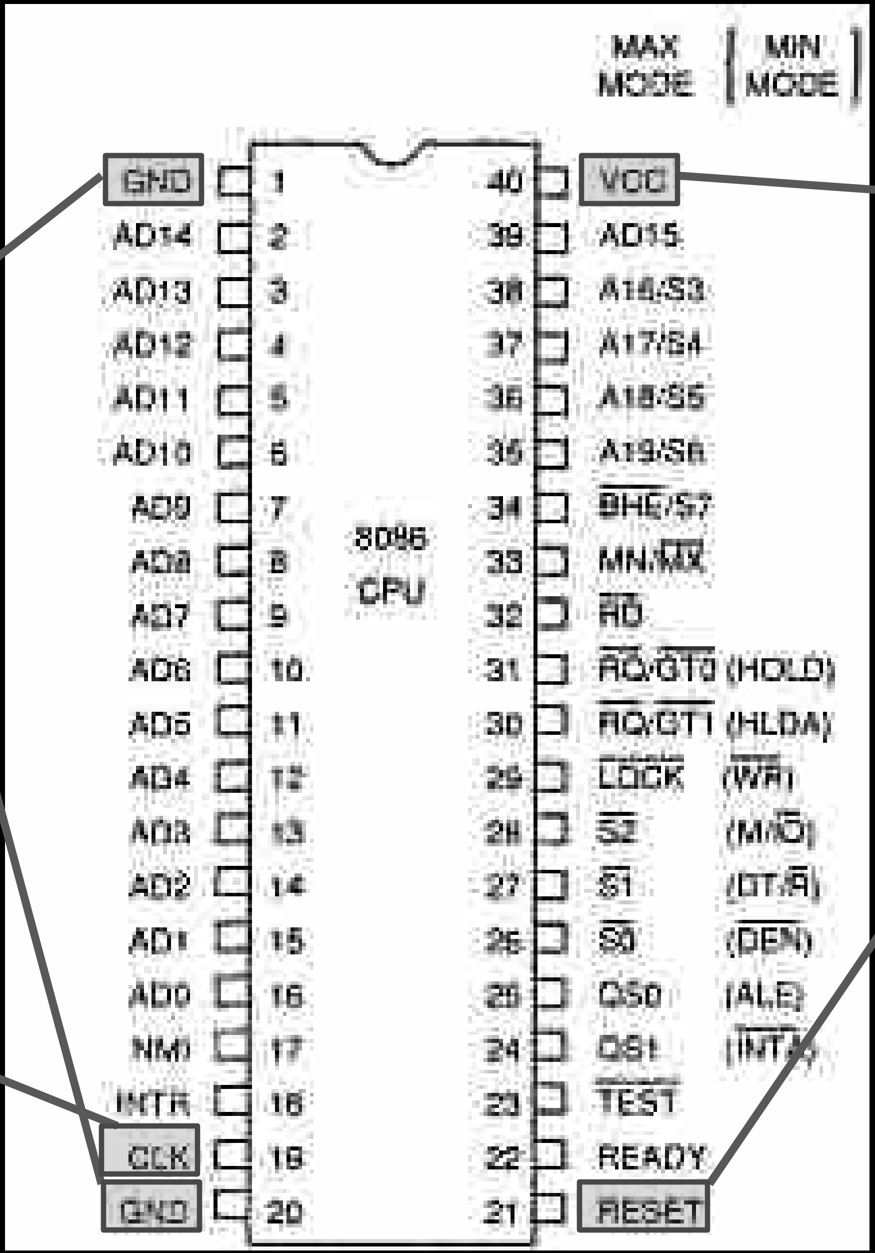
Bottom of stack

34

INTEL 8086 - Pin Diagram



INTEL 8086 - Pin Details



Power Supply

5V ± 10%

Ground

Reset

Registers, seg regs, flags

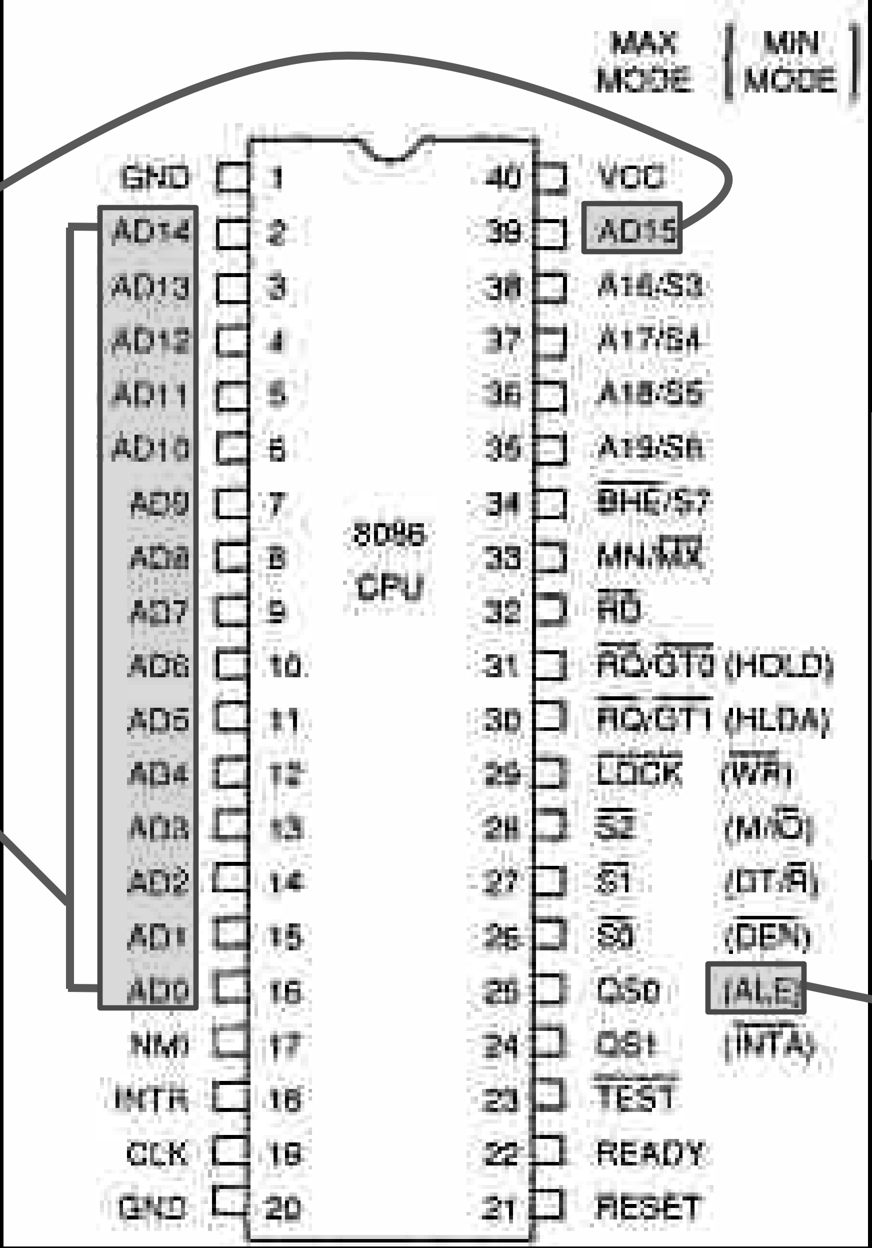
CS: FFFFH, IP: 0000H

If high for minimum 4 clks

Clock

Duty cycle: 33%

INTEL 8086 - Pin Details



Address/Data Bus:

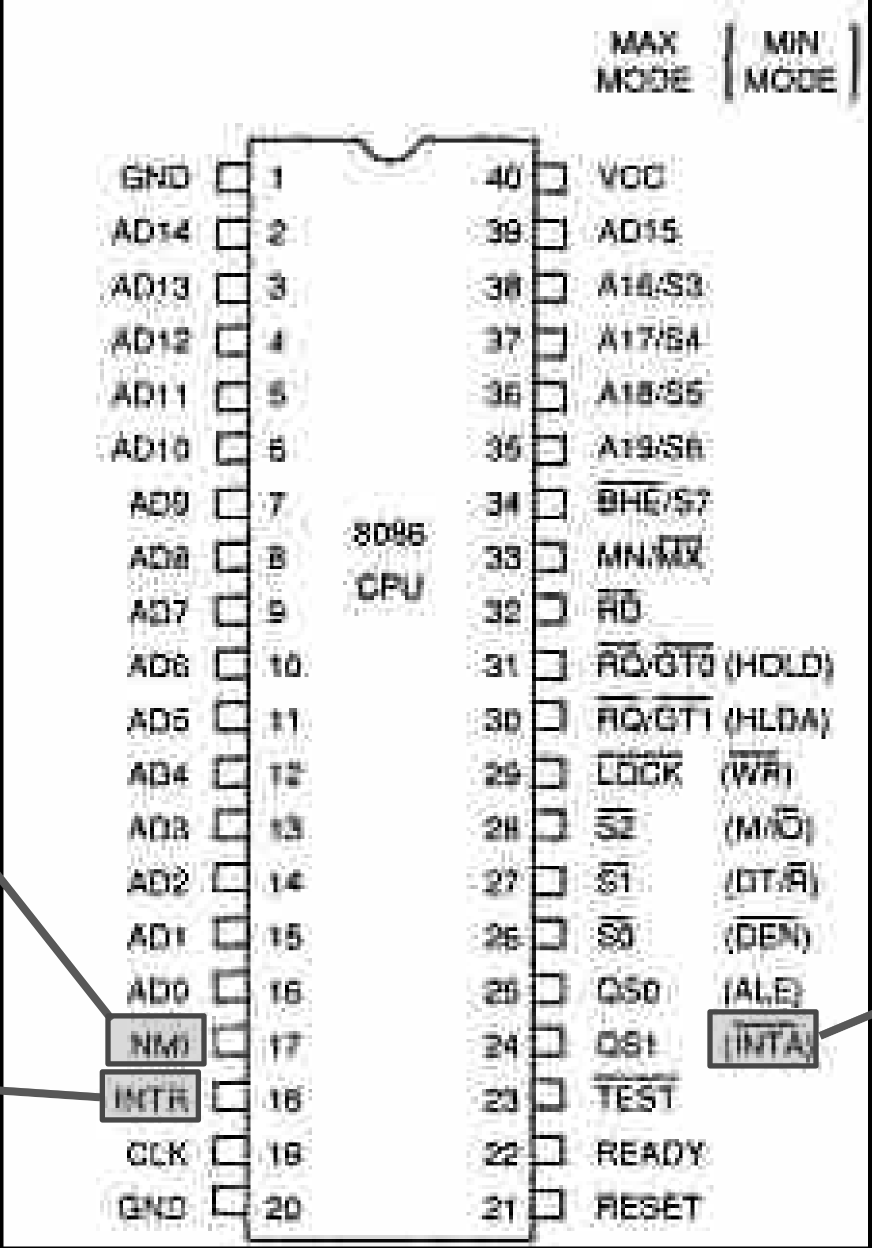
Contains address bits A_{15}-A_0 when ALE is 1 & data bits D_{15} – D_0 when ALE is 0.

Address Latch Enable:

When high, multiplexed address/data bus contains address information.

INTEL 8086 - Pin Details

INTERRUPT

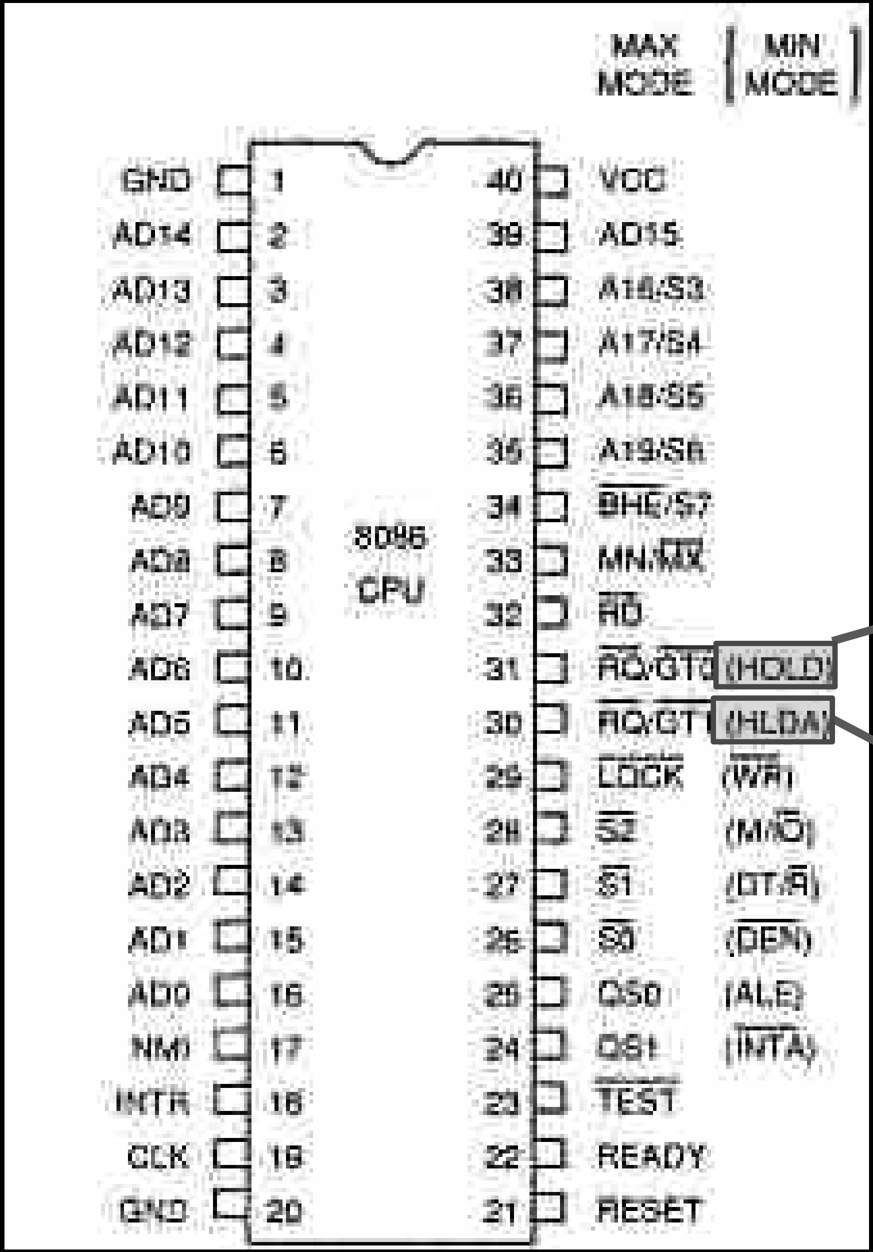


Non - maskable interrupt

Interrupt request

Interrupt acknowledge

INTEL 8086 - Pin Details



Direct Memory Access

Hold

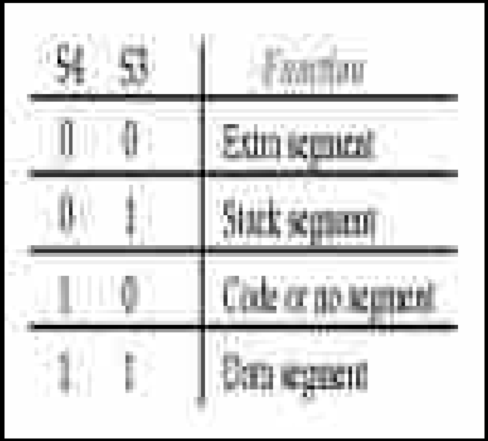
Hold acknowledge

INTEL 8086 - Pin Details

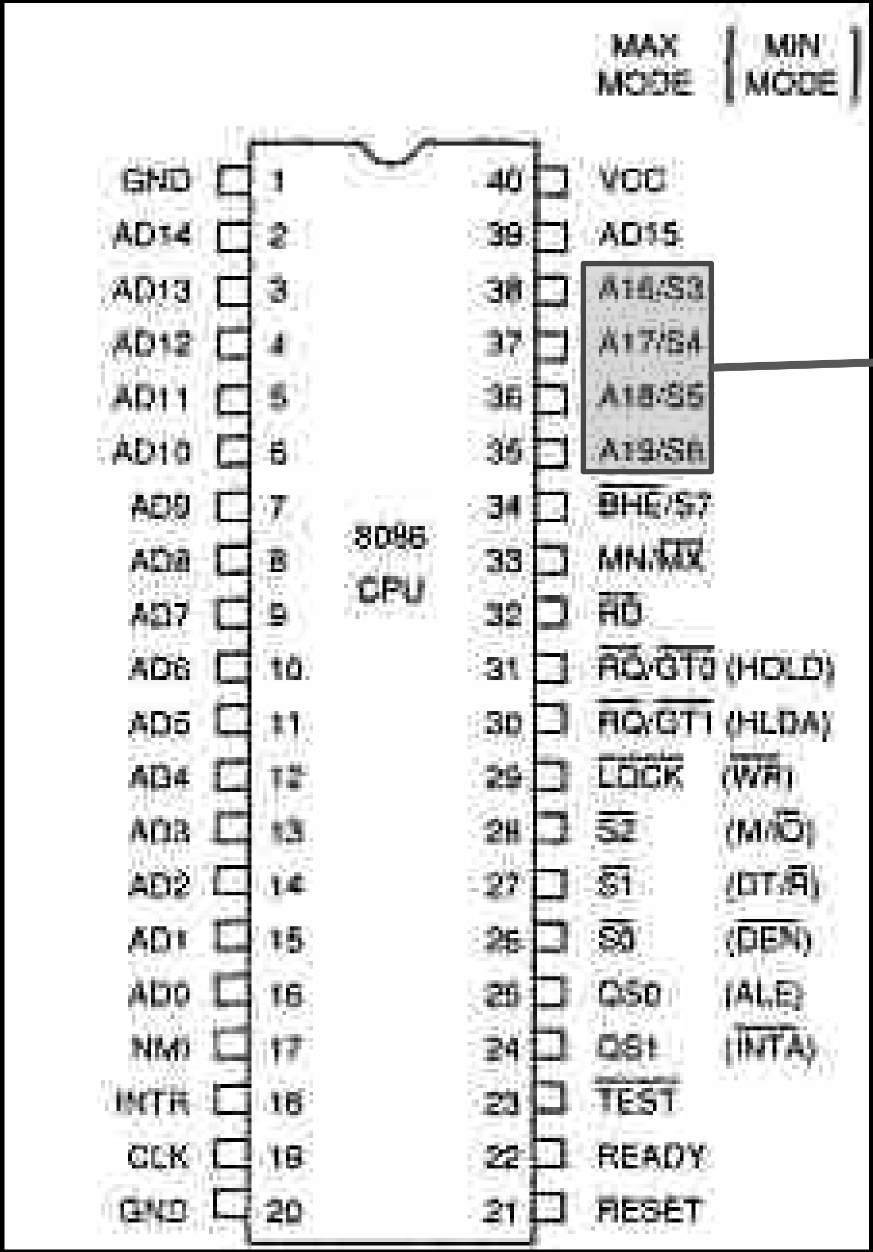
S6: Logic 0.

S5: Indicates condition of IF flag bits.

S4-S3: Indicate which segment is accessed during current bus cycle:



S4	S3	Function
0	0	Extra segment
0	1	Stack segment
1	0	Code or no segment
1	1	Data segment



Address/Status Bus

Address bits A_{19} – A_{16} & Status bits S_6 – S_3

INTEL 8086 - Pin Details

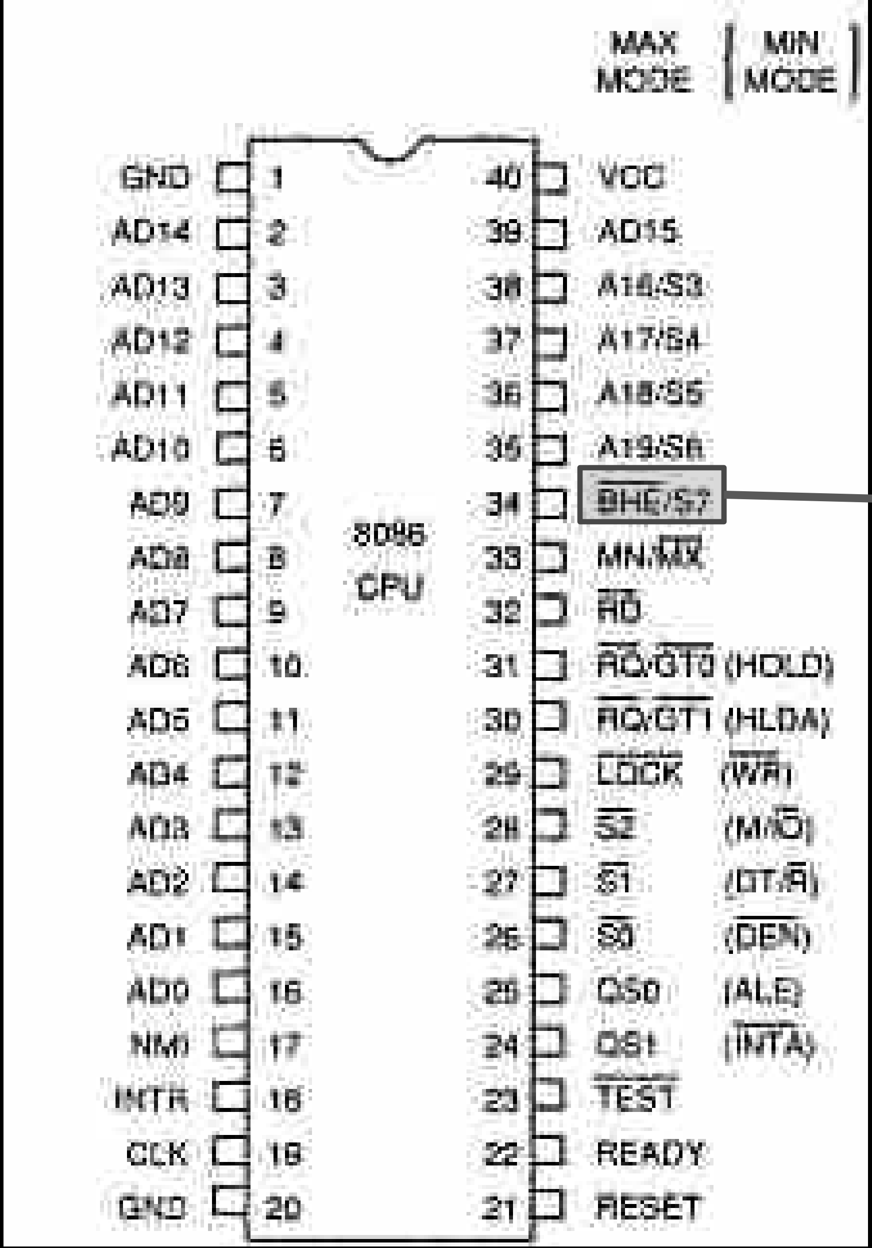
BHE#, A_0:

0,0: Whole word (16-bits)

0,1: High byte to/from odd address

1,0: Low byte to/from even address

1,1: No selection

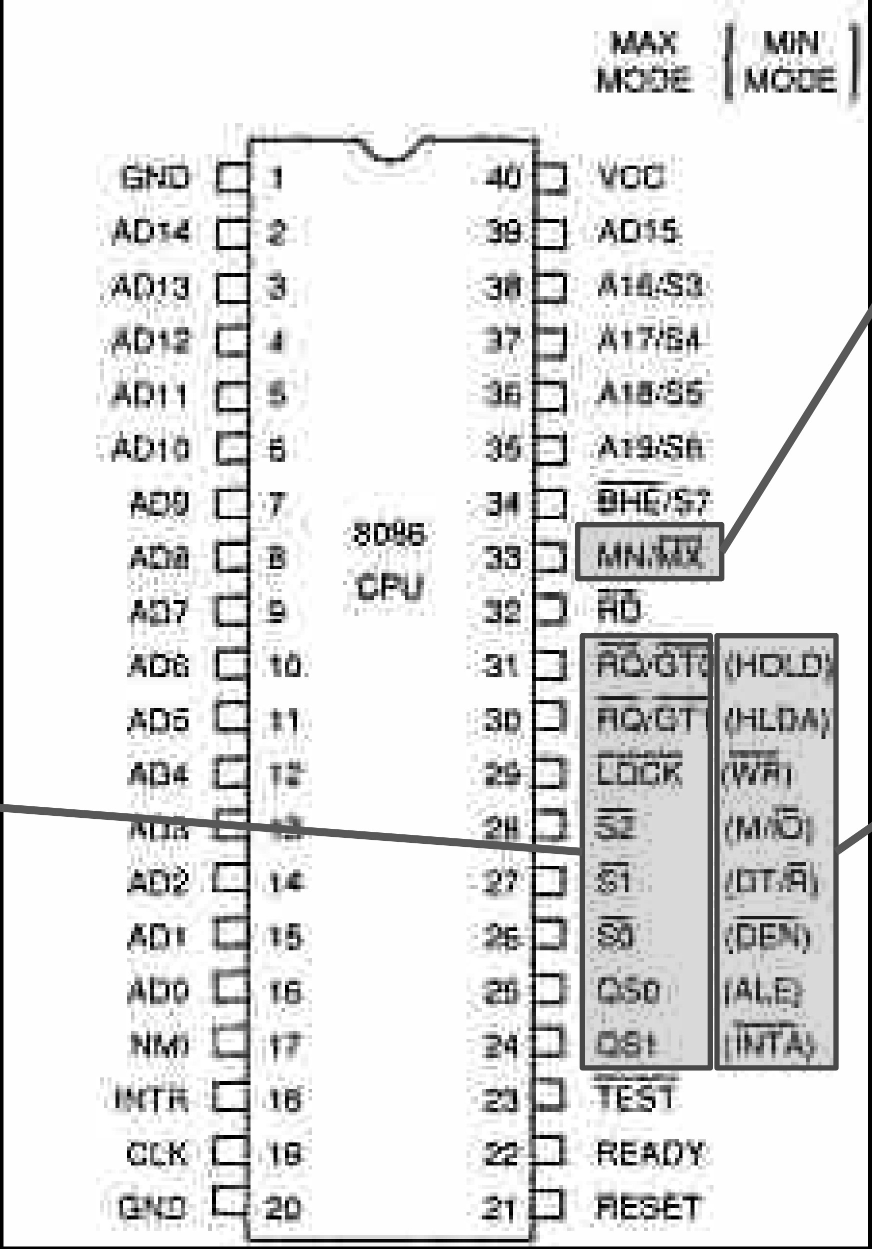


Bus High Enable/S7

Enables most significant data bits $D_{15} – D_8$ during read or write operation.

S_7: Always 1.

INTEL 8086 - Pin Details



Min/Max mode

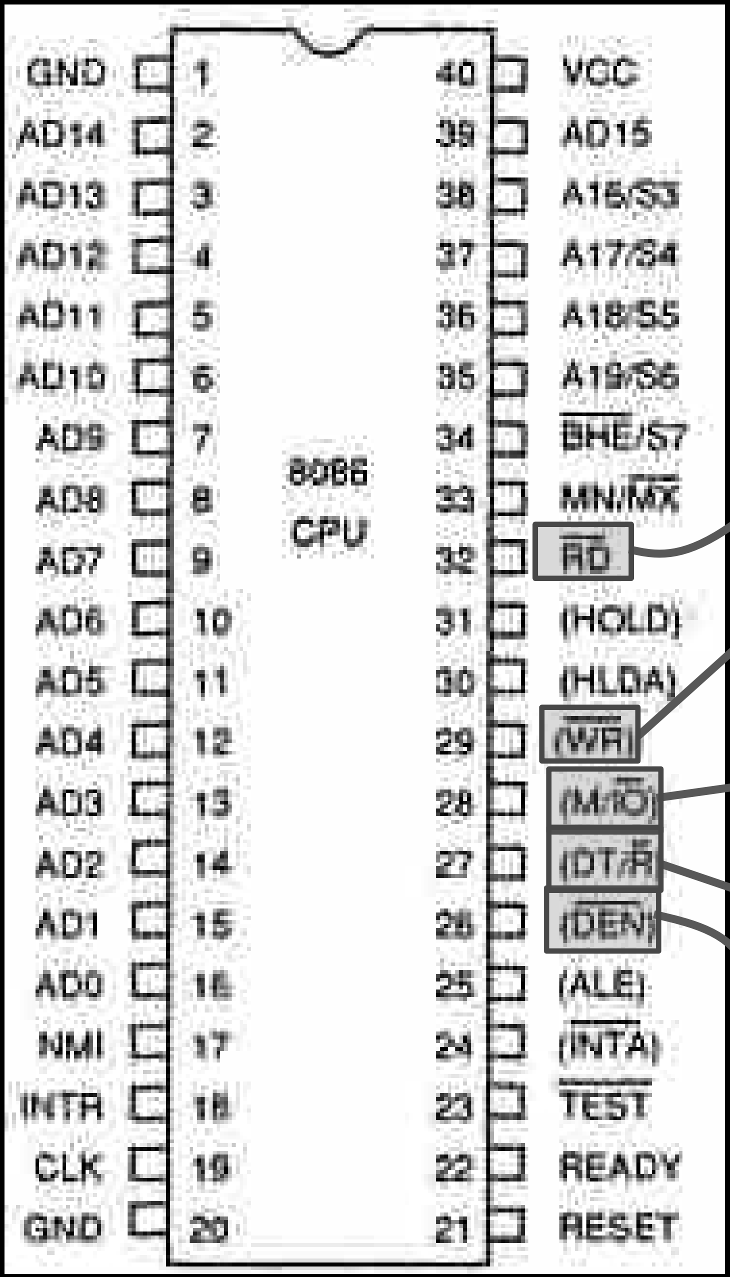
Minimum Mode: +5V

Maximum Mode: 0V

Minimum Mode Pins

Maximum Mode Pins

Minimum Mode- Pin Details



Read Signal

Write Signal

Memory or I/0

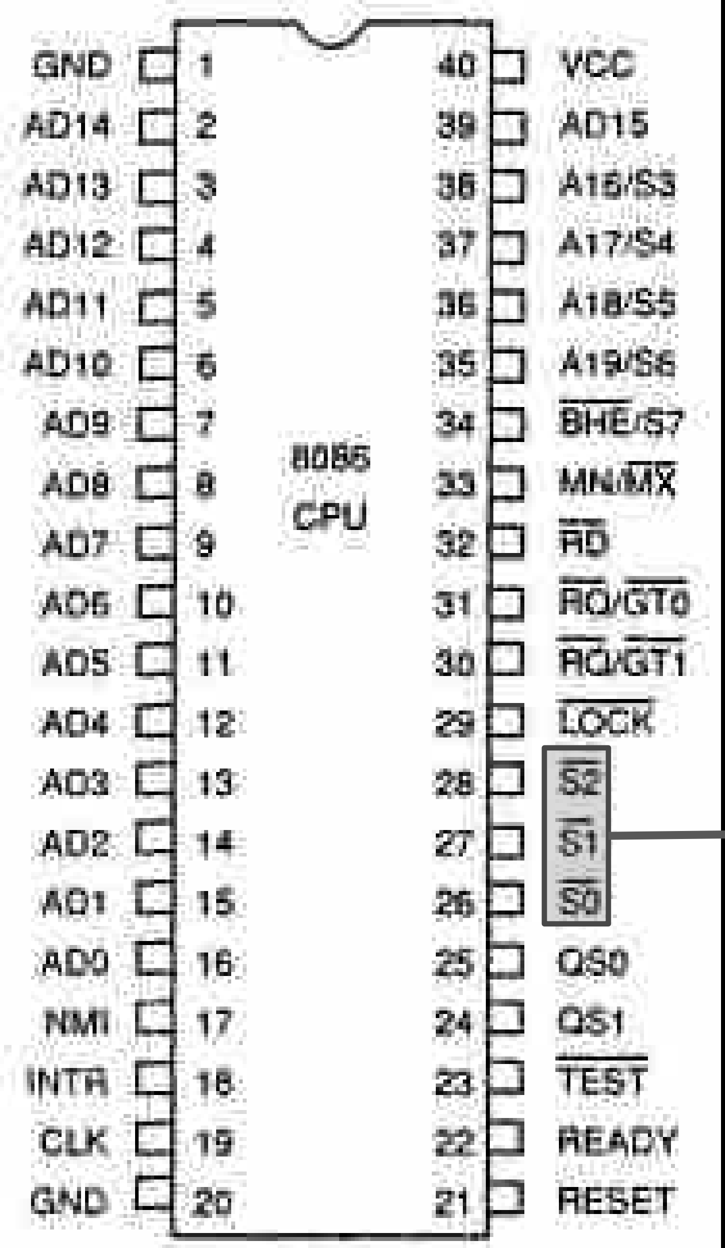
Data Transmit/Receive

Data Bus Enable

Maximum Mode - Pin Details

S2 S1 S0

000: INTA
001: read I/O port
010: write I/O port
011: halt
100: code access
101: read memory
110: write memory
111: none -passive



Status Signal

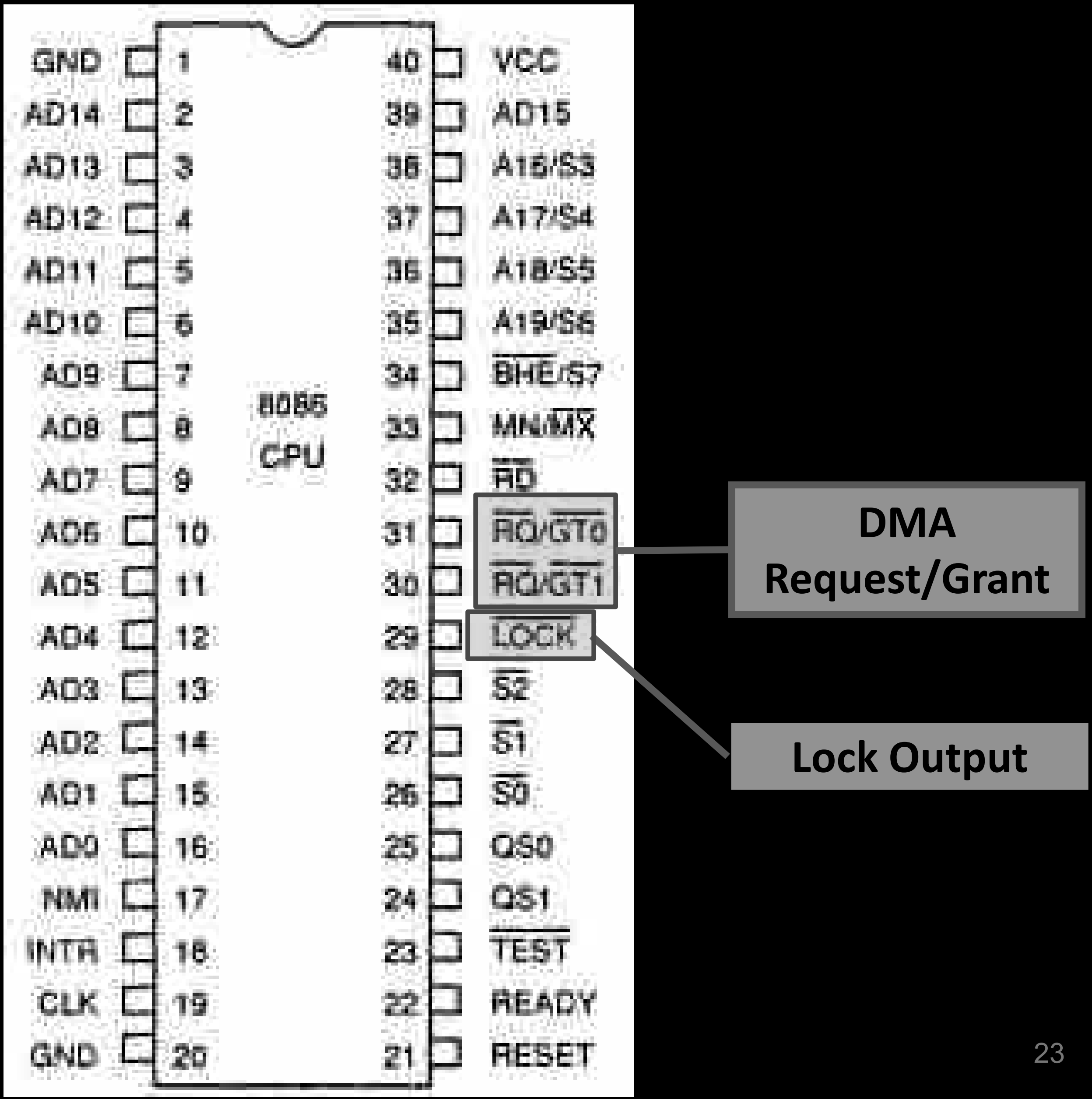
Inputs to 8288 to generate eliminated signals due to max mode.

Maximum Mode - Pin Details

Lock Output

Used to lock peripherals off the system

Activated by using the LOCK: prefix on any instruction



DMA Request/Grant

Lock Output

Maximum Mode - Pin Details

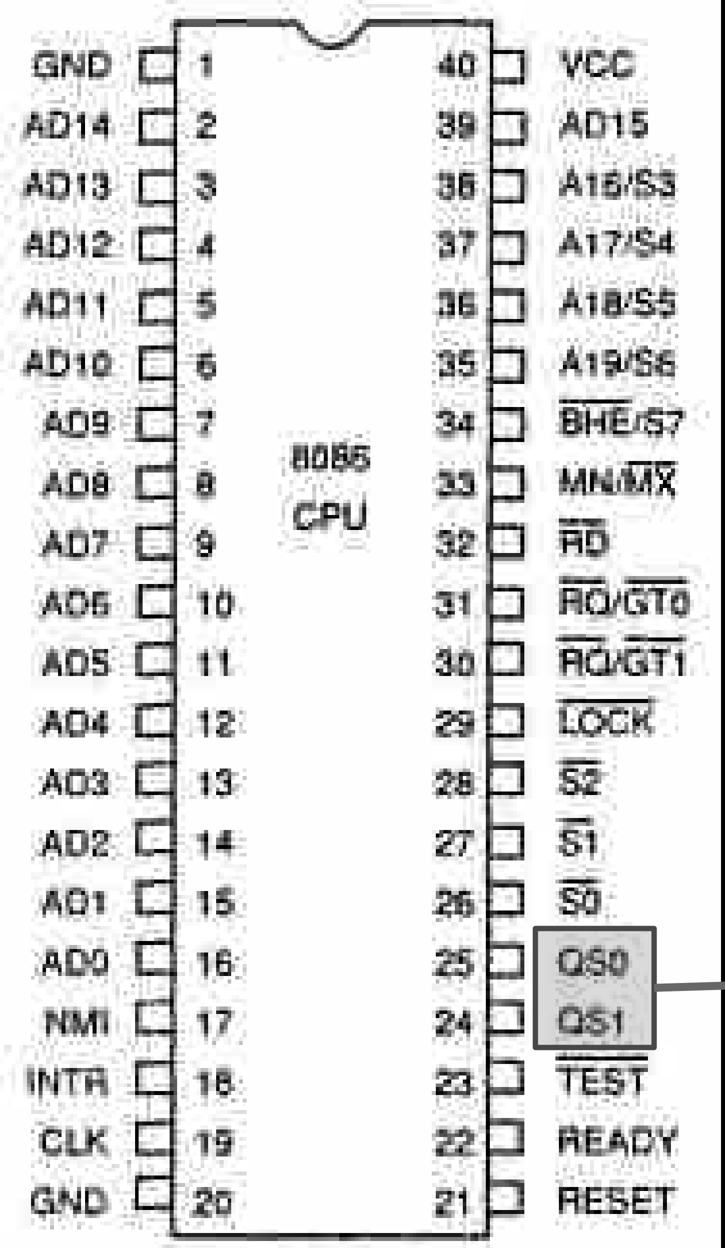
QS1 QS0

00: Queue is idle

01: First byte of opcode

10: Queue is empty

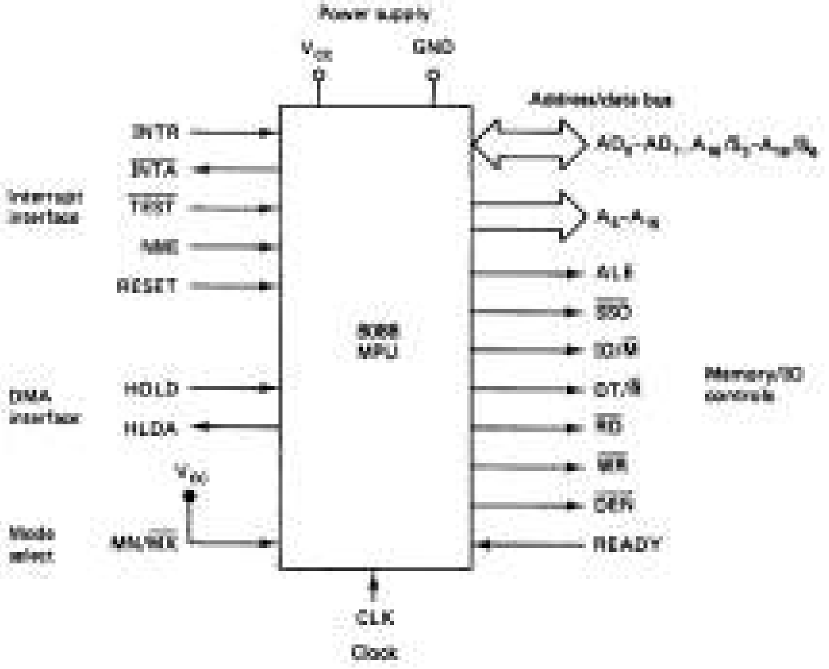
11: Subsequent byte of opcode



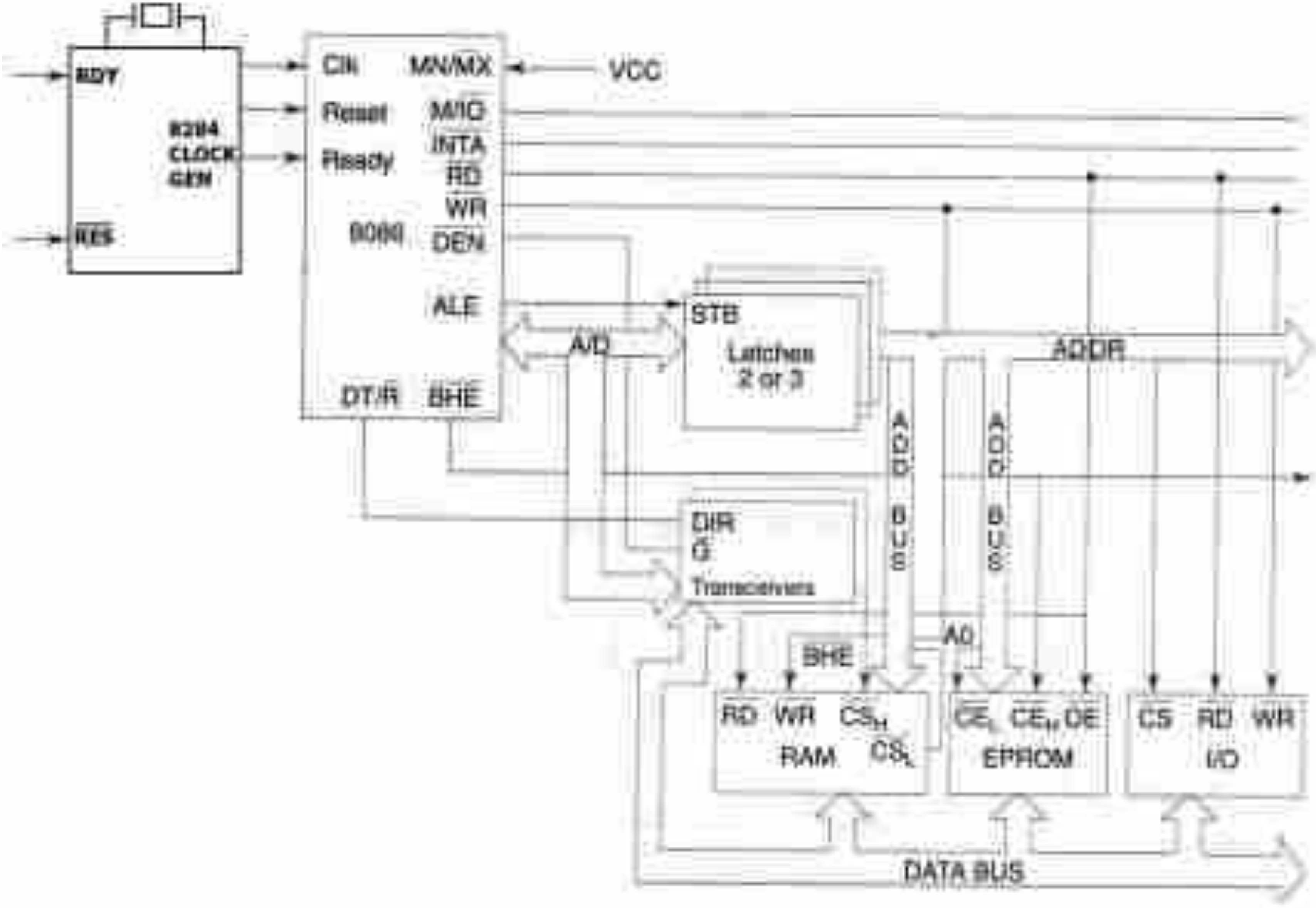
Queue Status

Used by numeric coprocessor (8087)

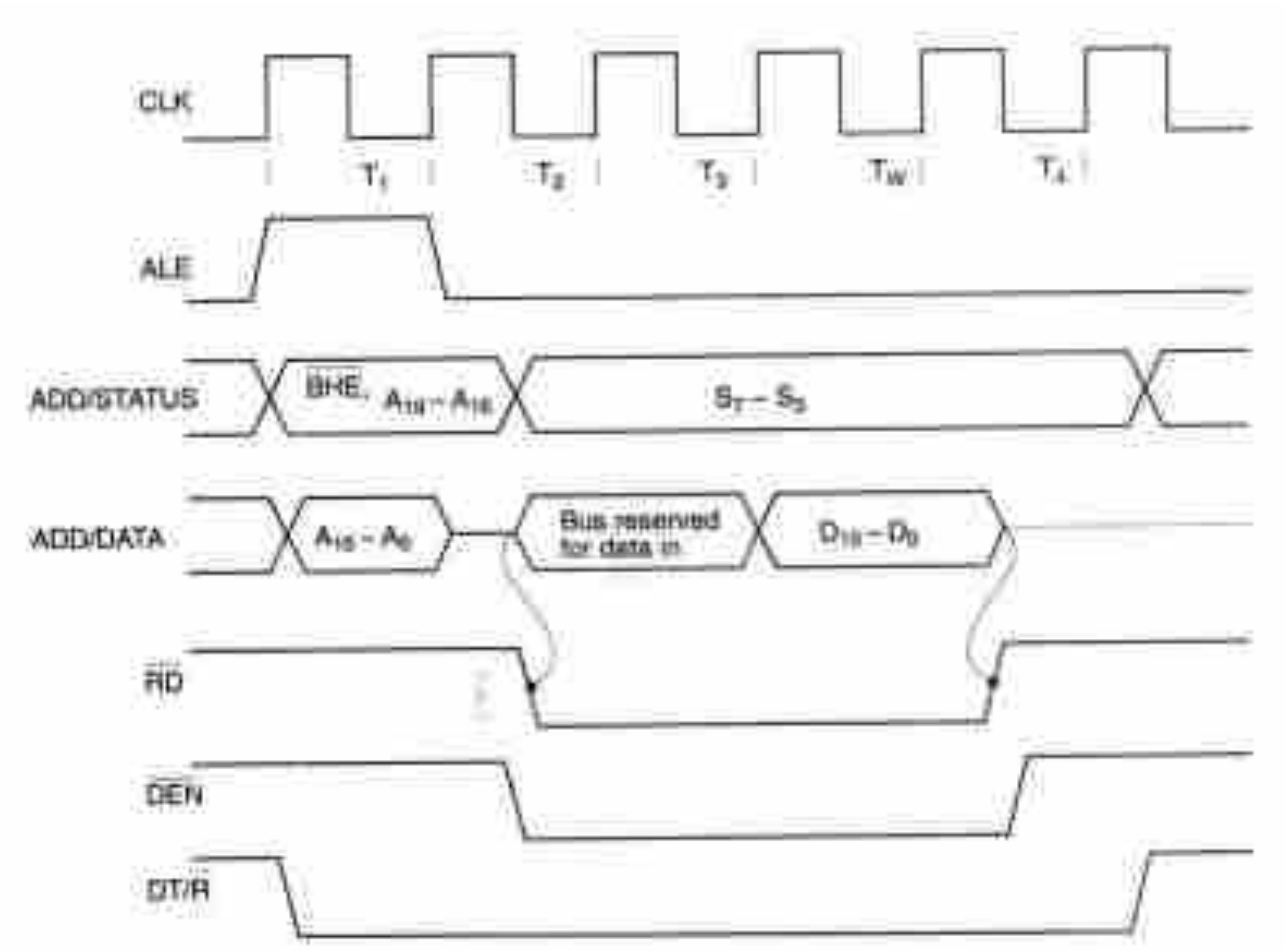
Minimum Mode 8086 System



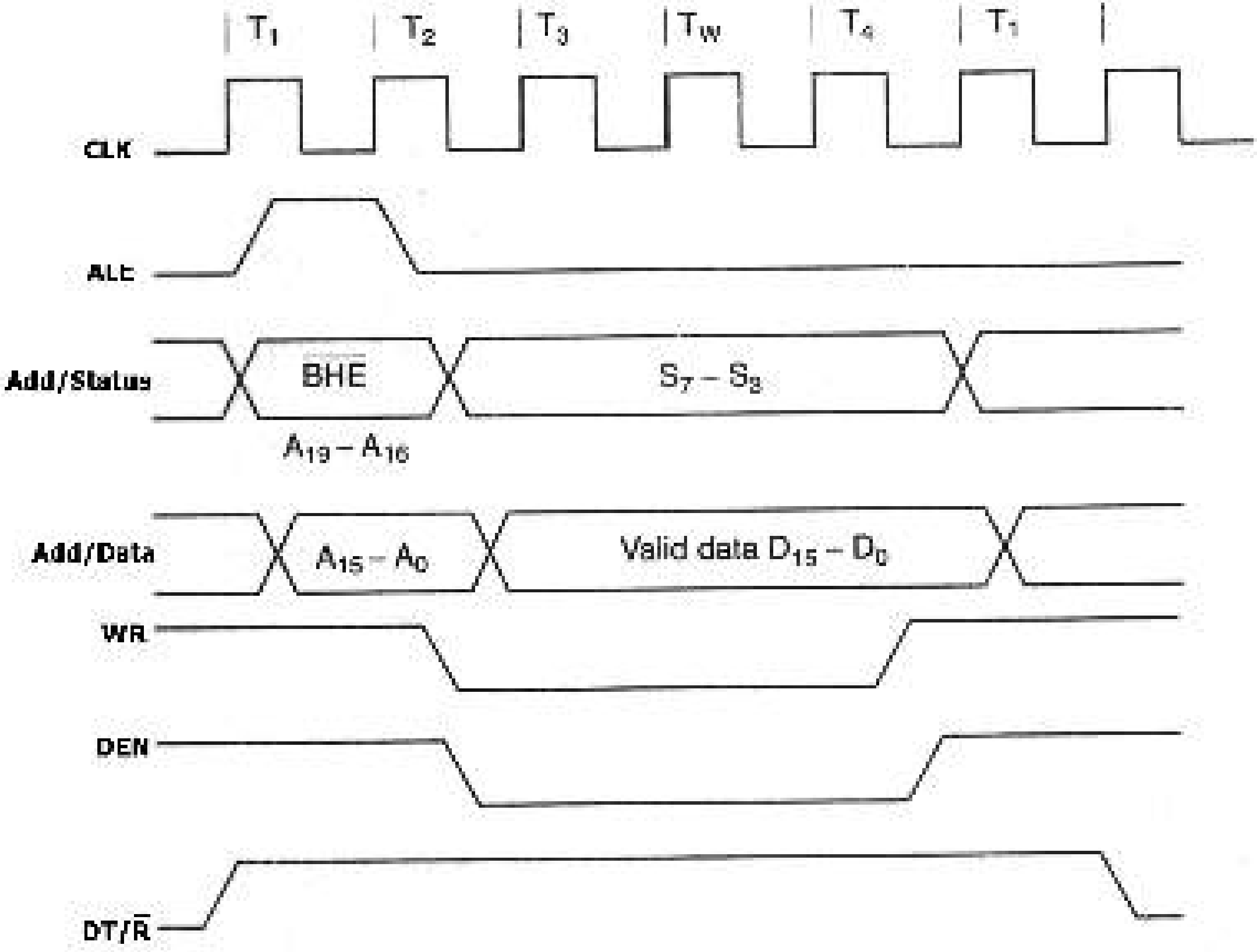
Minimum Mode 8086 System



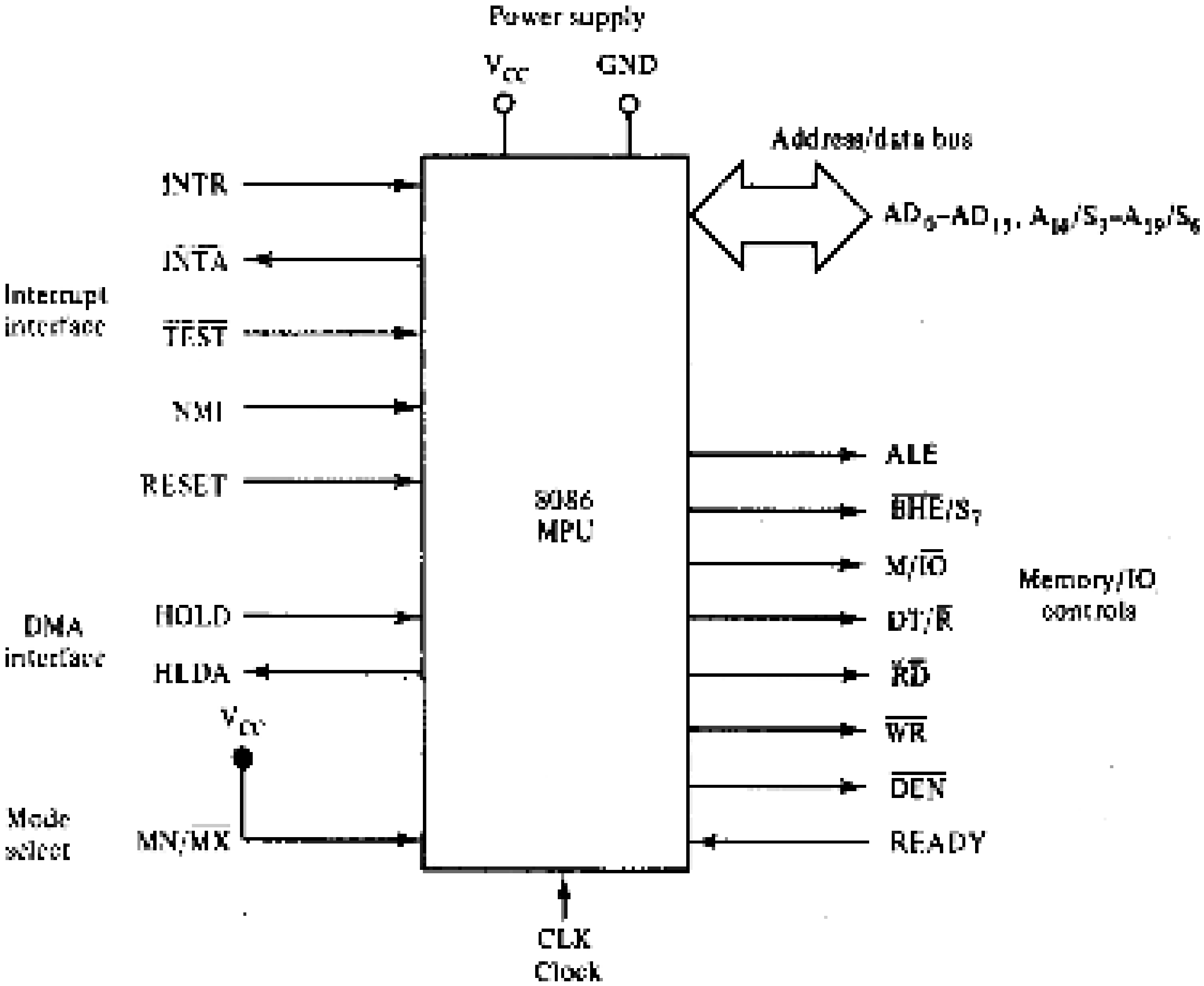
‘Read’ Cycle timing Diagram for Minimum Mode



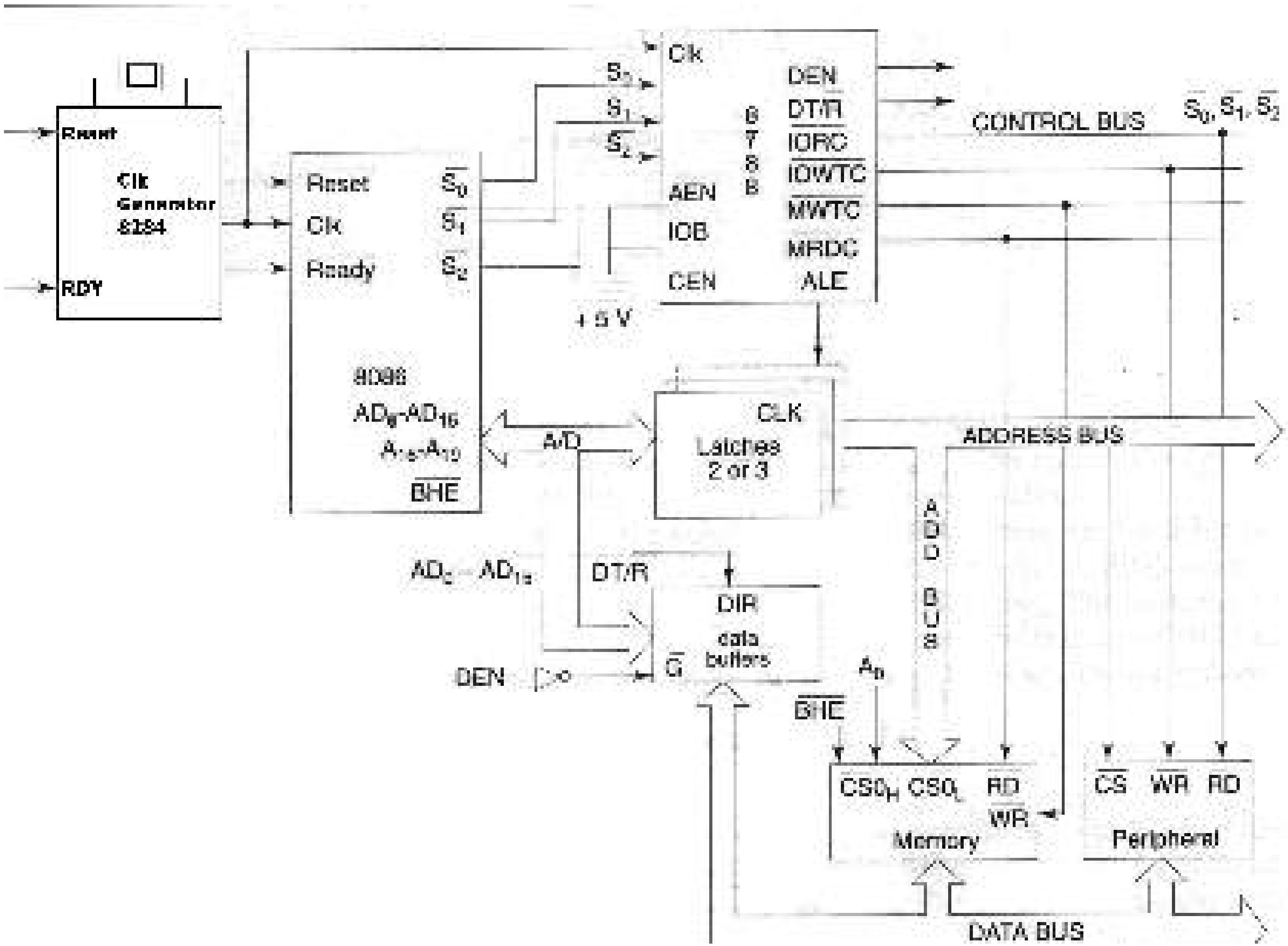
‘Write’ Cycle timing Diagram for Minimum Mode



Maximum Mode 8086 System



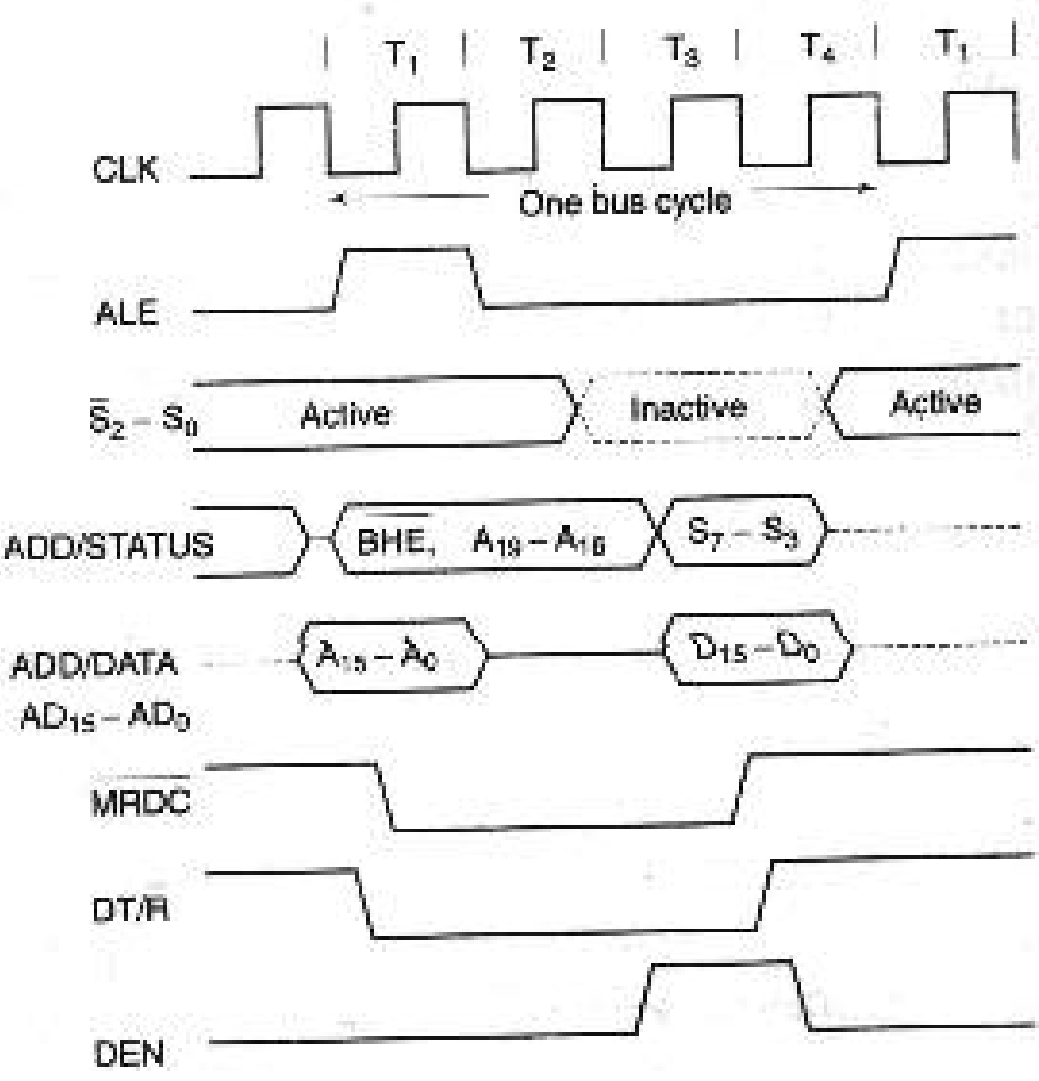
Maximum Mode 8086 System

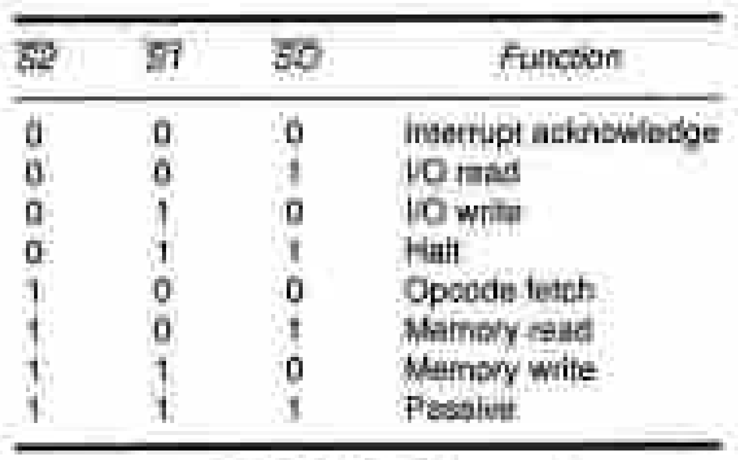


Maximum Mode 8086 System

- Here, either a numeric coprocessor of the type 8087 or another processor is interfaced with 8086.
- The Memory, Address Bus, Data Buses are shared resources between the two processors.
- The control signals for Maximum mode of operation are generated by the Bus Controller chip 8788.
- The three status outputs S0*, S1*, S2* from the processor are input to 8788.
- The outputs of the bus controller are the Control Signals, namely DEN, DT/R*, IORC*, IOWTC*, MWTC*, MRDC*, ALE etc.

Memory Read timing in Maximum Mode

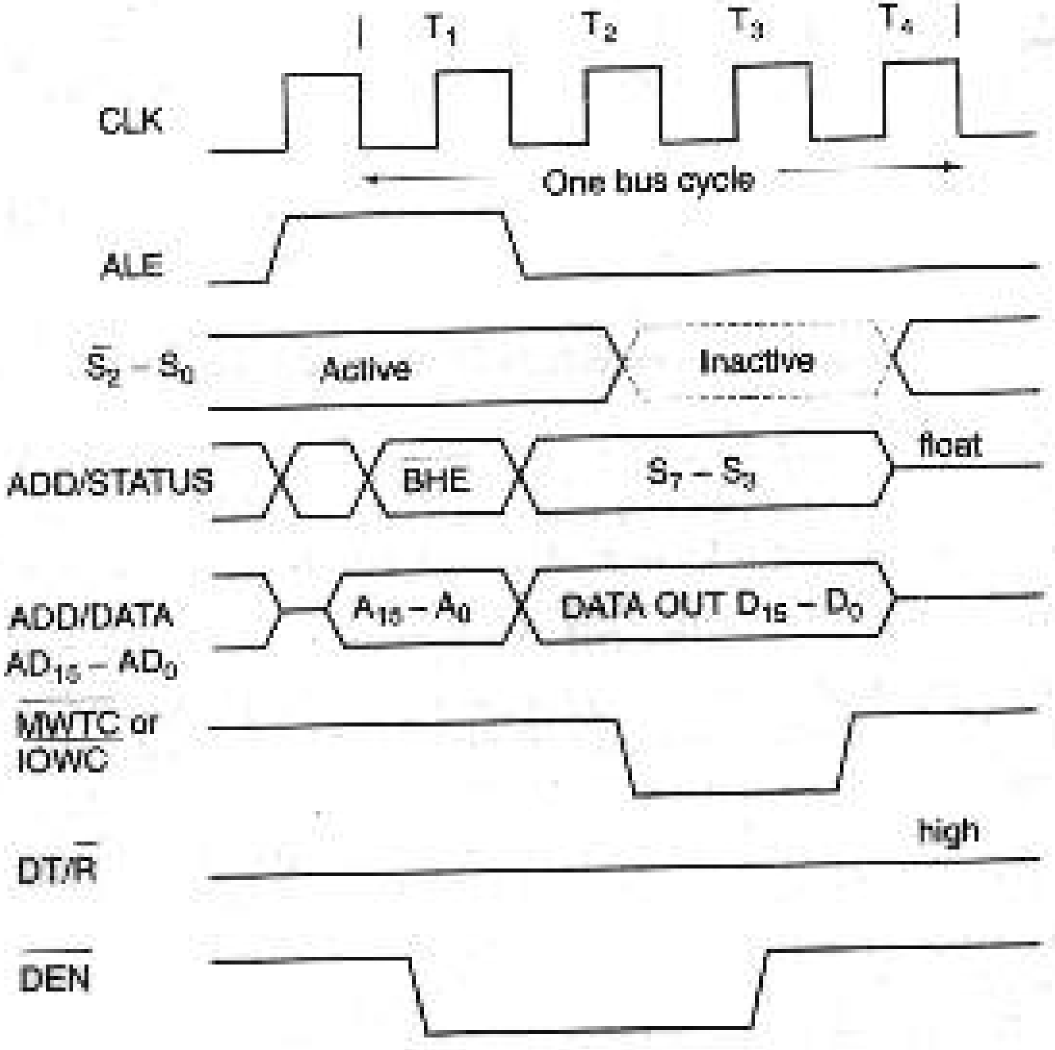


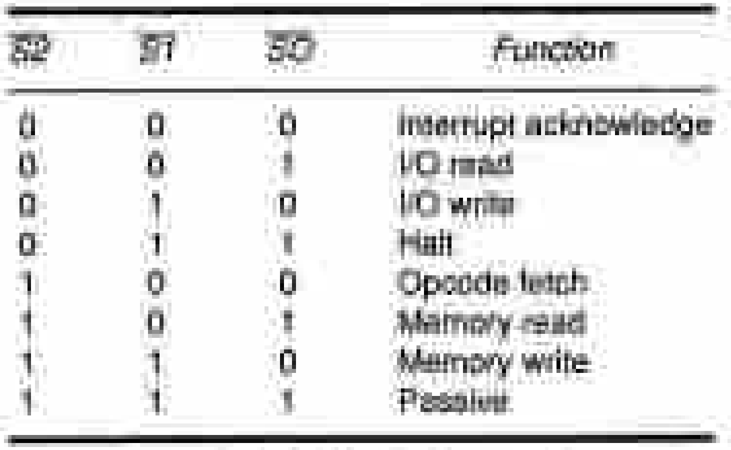


$\overline{S2}$	$\overline{S1}$	$\overline{S0}$	*Function*
0	0	0	Interrupt acknowledge
0	0	1	I/O read
0	1	0	I/O write
0	1	1	Halt
1	0	0	Opcode fetch
1	0	1	Memory read
1	1	0	Memory write
1	1	1	Passive

TABLE 9-6 Bus control functions generated by the bus controller (8288) using $\overline{S2}$, $\overline{S1}$, and $\overline{S0}$

Memory Write timing in Maximum Mode





$\overline{S2}$	$\overline{S1}$	$\overline{S0}$	*Function*
0	0	0	Interrupt acknowledge
0	0	1	I/O read
0	1	0	I/O write
0	1	1	Halt
1	0	0	Opcode fetch
1	0	1	Memory read
1	1	0	Memory write
1	1	1	Passive

TABLE 9-6 Bus control functions generated by the bus controller (8288) using $\overline{S2}$, $\overline{S1}$, and $\overline{S0}$

8086 Control Signals

1. **ALE**
2. **BHE**
3. **M/IO**
4. **DT/R**
5. **RD**
6. **WR**
7. **DEN**

Coprocessor and Multiprocessor configuration

- Multiprocessor Systems refer to the use of multiple processors that **executes instructions simultaneously** and **communicate with each other** using mail boxes and Semaphores.

- Maximum mode of 8086 is designed to implement 3 basic multiprocessor configurations:

  1. **Coprocessor (8087)**
  2. **Closely coupled (8089)**
  3. **Loosely coupled (Multibus)**

Coprocessor and Multiprocessor configuration

- **Coprocessors** and **Closely coupled** configurations are similar in that both the 8086 and the external processor shares the:

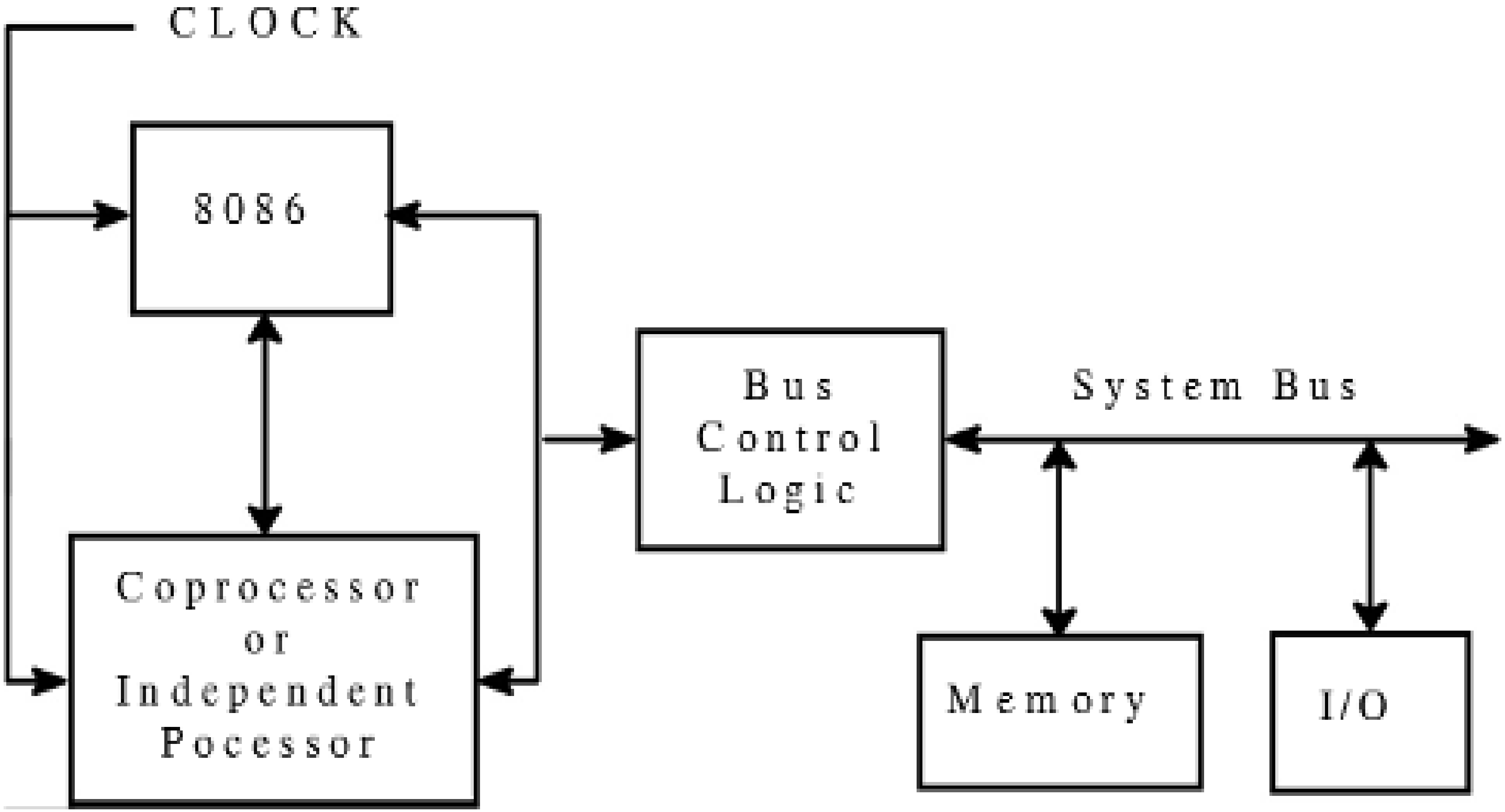
  - Memory

  - I/O system

  - Bus & bus control logic

  - Clock generator

Coprocessor / Closely Coupled Configuration



TEST pin of 8086

- Used in conjunction with the WAIT instruction in multiprocessing environments.

- This is input from the 8087 coprocessor.

- During execution of a wait instruction, the CPU checks this signal.

- If it is low, execution of the signal will continue; if not, it will stop executing.

Advantages of Multiprocessor Configuration

1. High **system throughput** can be achieved by having more than one CPU.

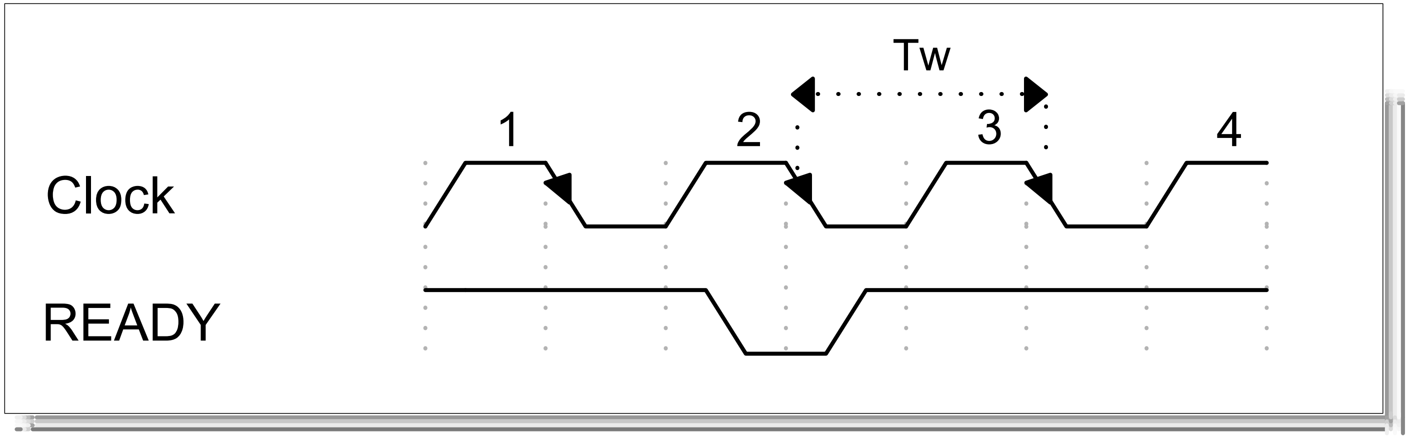
2. The system can be **expanded** in modular form.

   Each bus master module is an independent unit and normally resides on a separate PC board. One can be added or removed without affecting the others in the system.

3. A **failure** in one module normally does not affect the breakdown of the entire system and the faulty module can be easily detected and replaced

4. Each bus master has its own local bus to access dedicated memory or IO devices. So a greater **degree of parallel processing** can be achieved.

WAIT State



- A **wait state (T_w)** is an extra clocking period, inserted between T2 and T3, to lengthen the bus cycle, allowing slower memory and I/O components to respond.

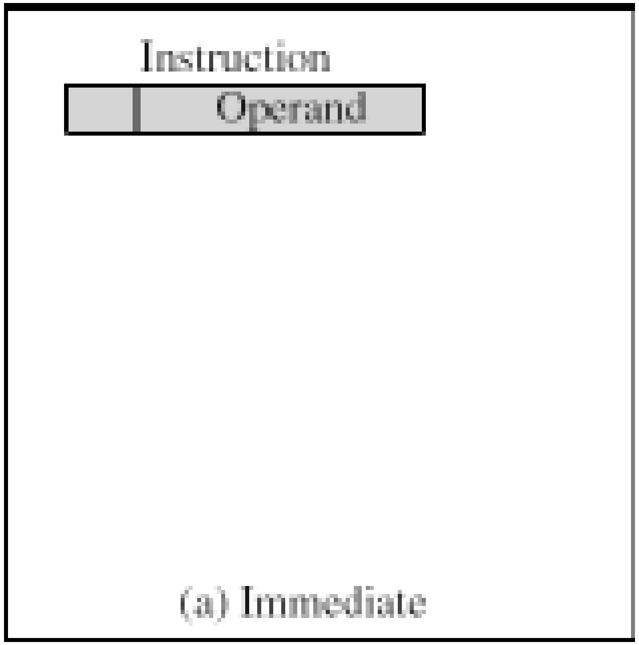
- The **READY** input is sampled at the end of **T2**, and again, if necessary in the middle of Tw. **If READY is ‘0’ then a Tw is inserted.**

Addressing Modes

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

Immediate Addressing

- Operand is part of instruction
- Operand = address field
- e.g. ADD AX, 5h
- LDA #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)

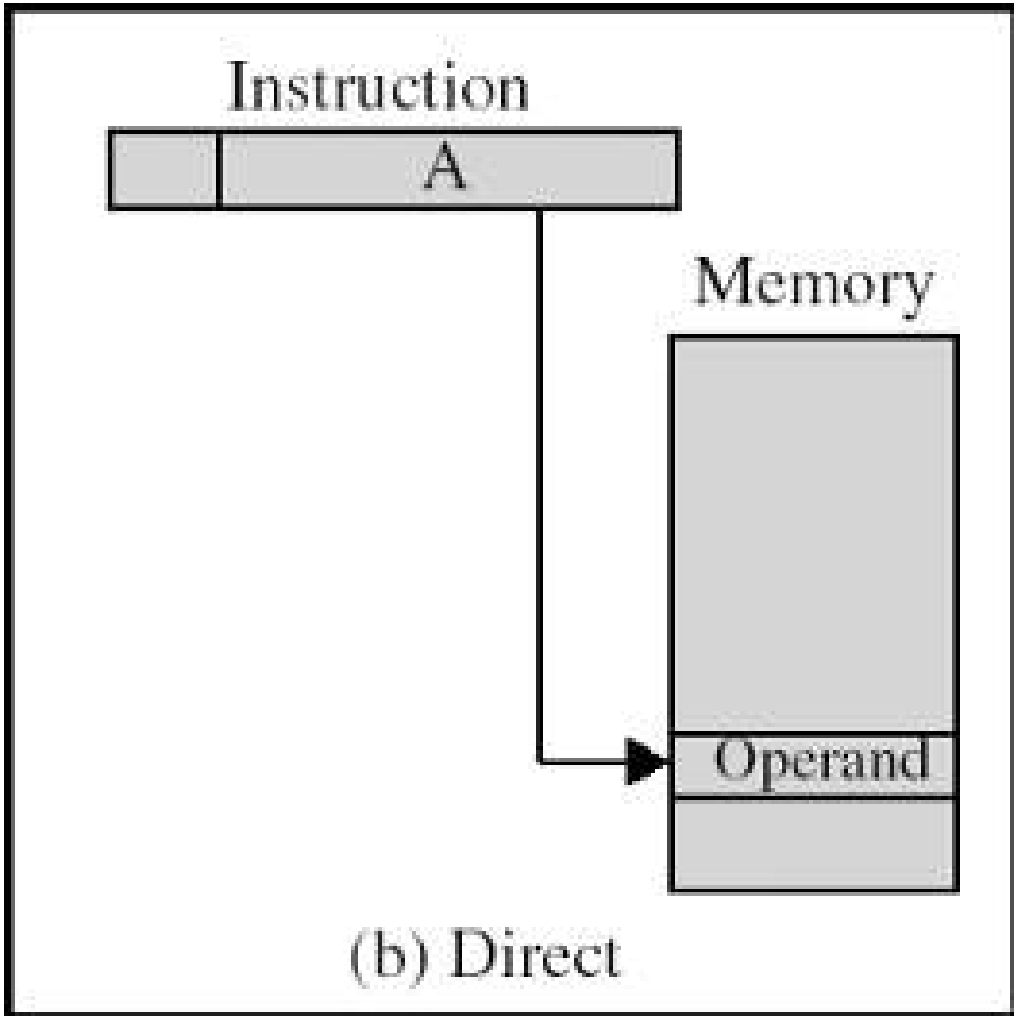
ADD AX, value

Value DB 05h

  - – Add contents of cell value to accumulator AX
  - – Look in memory at address value for operand

- Single memory reference to access data
- No additional calculations to work out effective address
- Limited address space

Direct Addressing Diagram



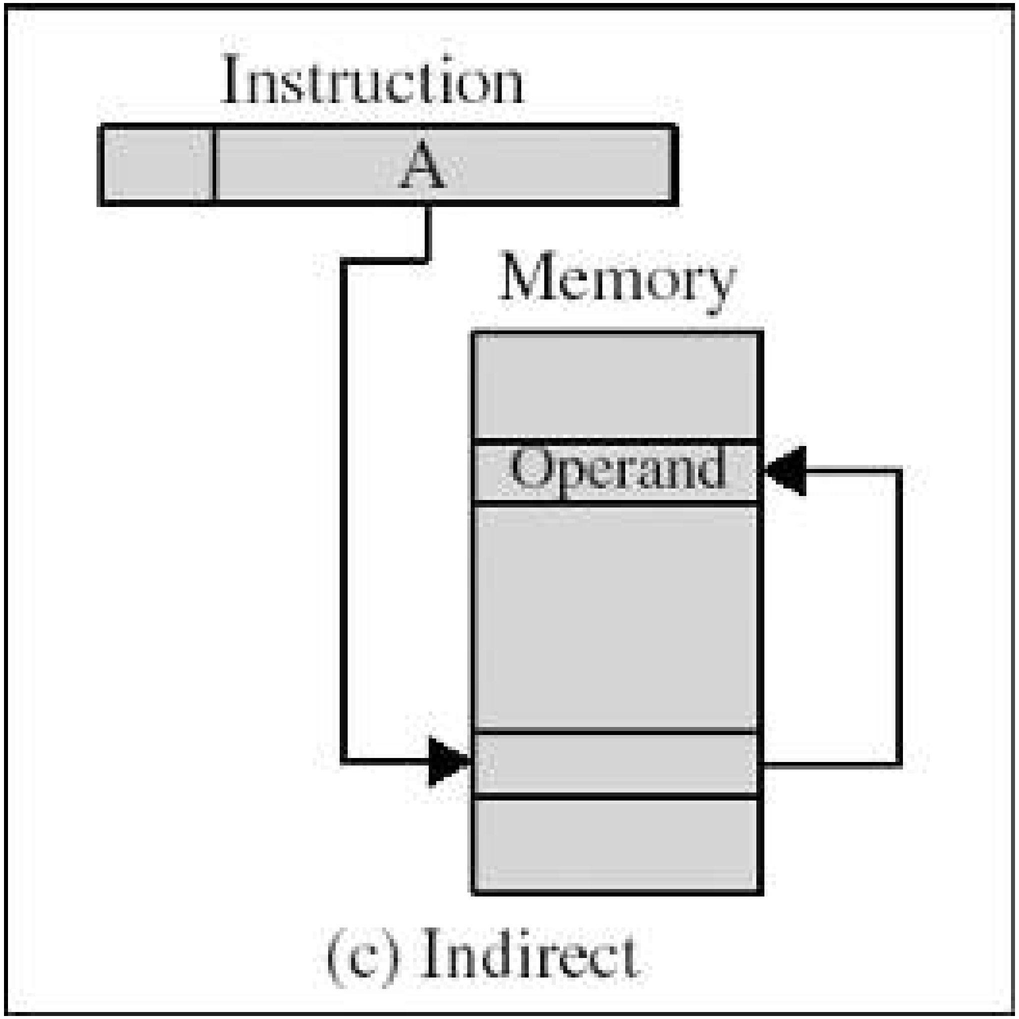
Indirect Addressing (1/2)

- 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
- e.g. ADD AX, (A)
  - Add contents of cell pointed to by contents of A to accumulator

Indirect Addressing (2/2)

- Large address space
- 2^n where n = word length
- May be nested, multilevel, cascaded
  - e.g. EA = (((A)))
    - Draw the diagram yourself
- Multiple memory accesses to find operand
- Hence slower

Indirect Addressing Diagram



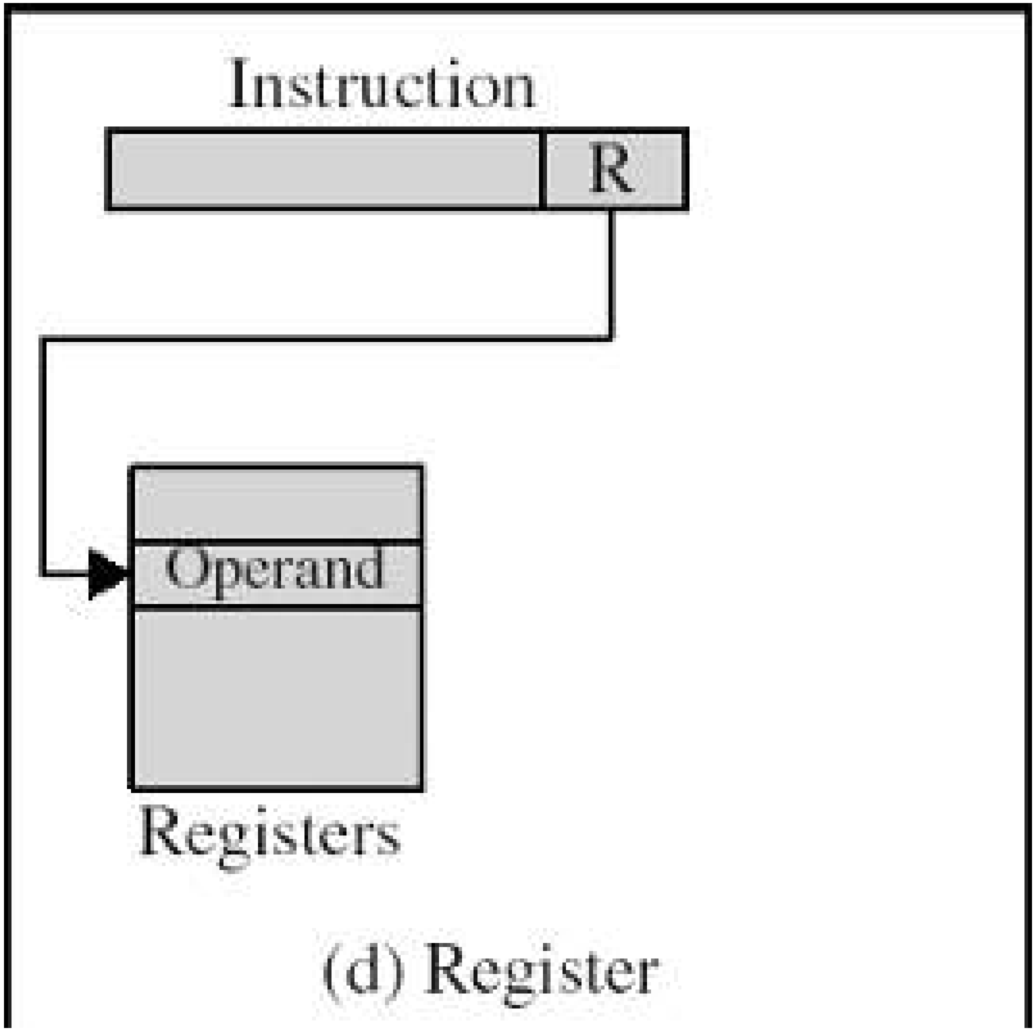
Register Addressing (1/2)

- Operand is held in register named in address filed
- EA = R
- Limited number of registers
- Very small address field needed
  - Shorter instructions
  - Faster instruction fetch
  - MOV AX, BX
  - ADD AX, BX

Register Addressing (2/2)

- No memory access
- Very fast execution
- Very limited address space
- Multiple registers helps performance
  - Requires good assembly programming or compiler writing
  - N.B. C programming
    - register int a;
- c.f. Direct addressing

Register Addressing Diagram

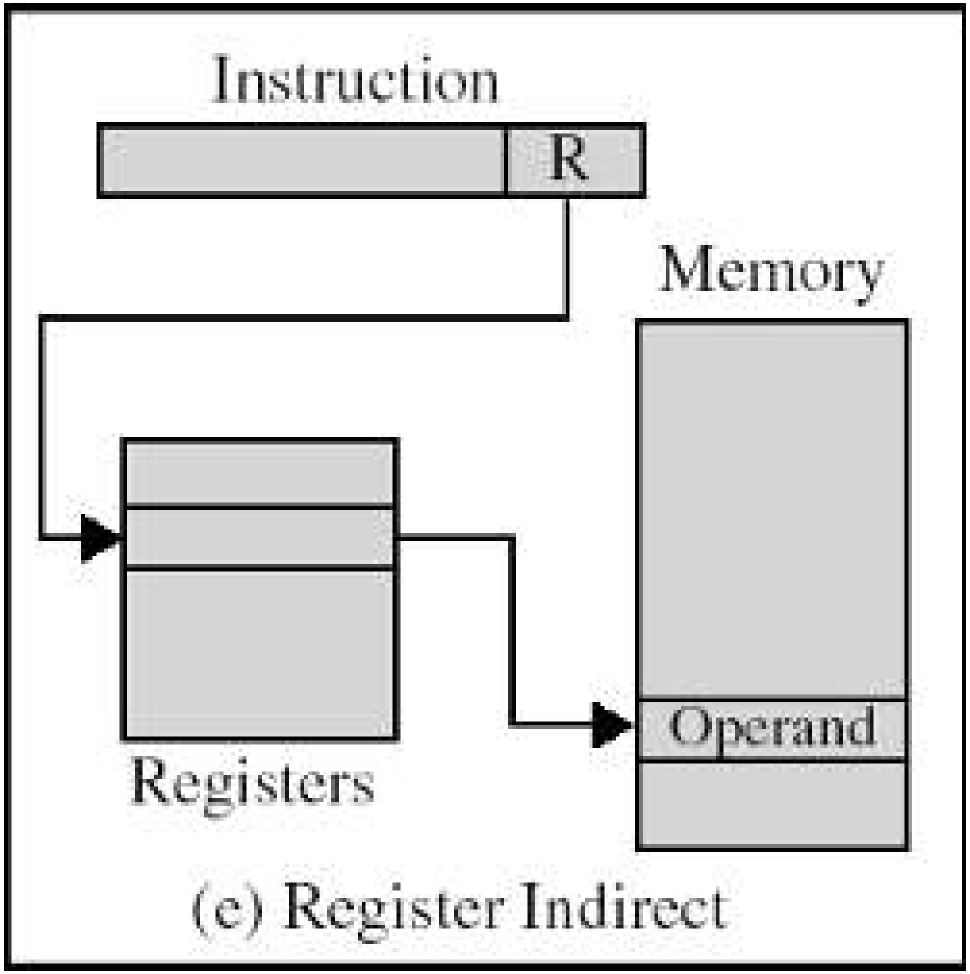


(d) Register

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

Register Indirect Addressing Diagram

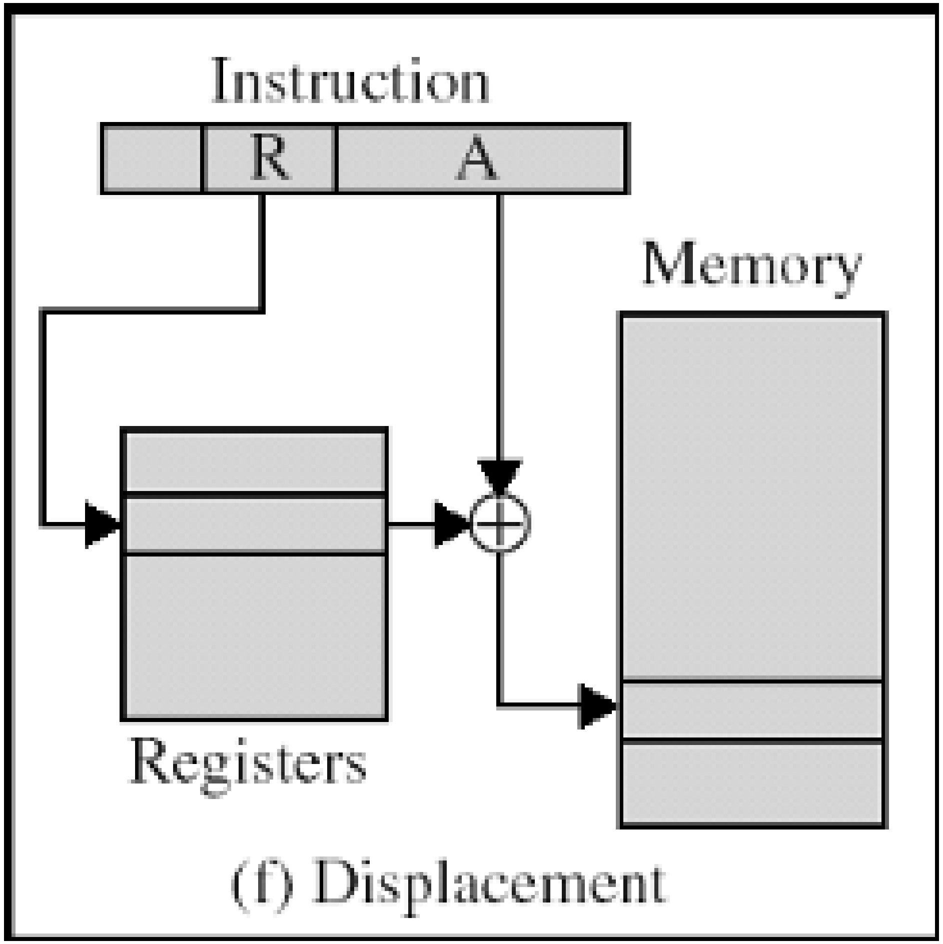


(c) Register Indirect

Displacement Addressing

- EA = A + (R)
- Effective address=start address + displacement
- Effective address=Offset + (Segment Register)
- Use direct and register indirect
- Address field hold two values
  - A = base value
  - R = register that holds displacement
  - or vice versa

Displacement Addressing Diagram



(f) Displacement

Relative Addressing (PC-Relative)

- 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
  - EA = (CS) + A
- 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++

Combinations

- Postindex
  - EA = (A) + (R)
- Preindex
  - EA = (A+(R))

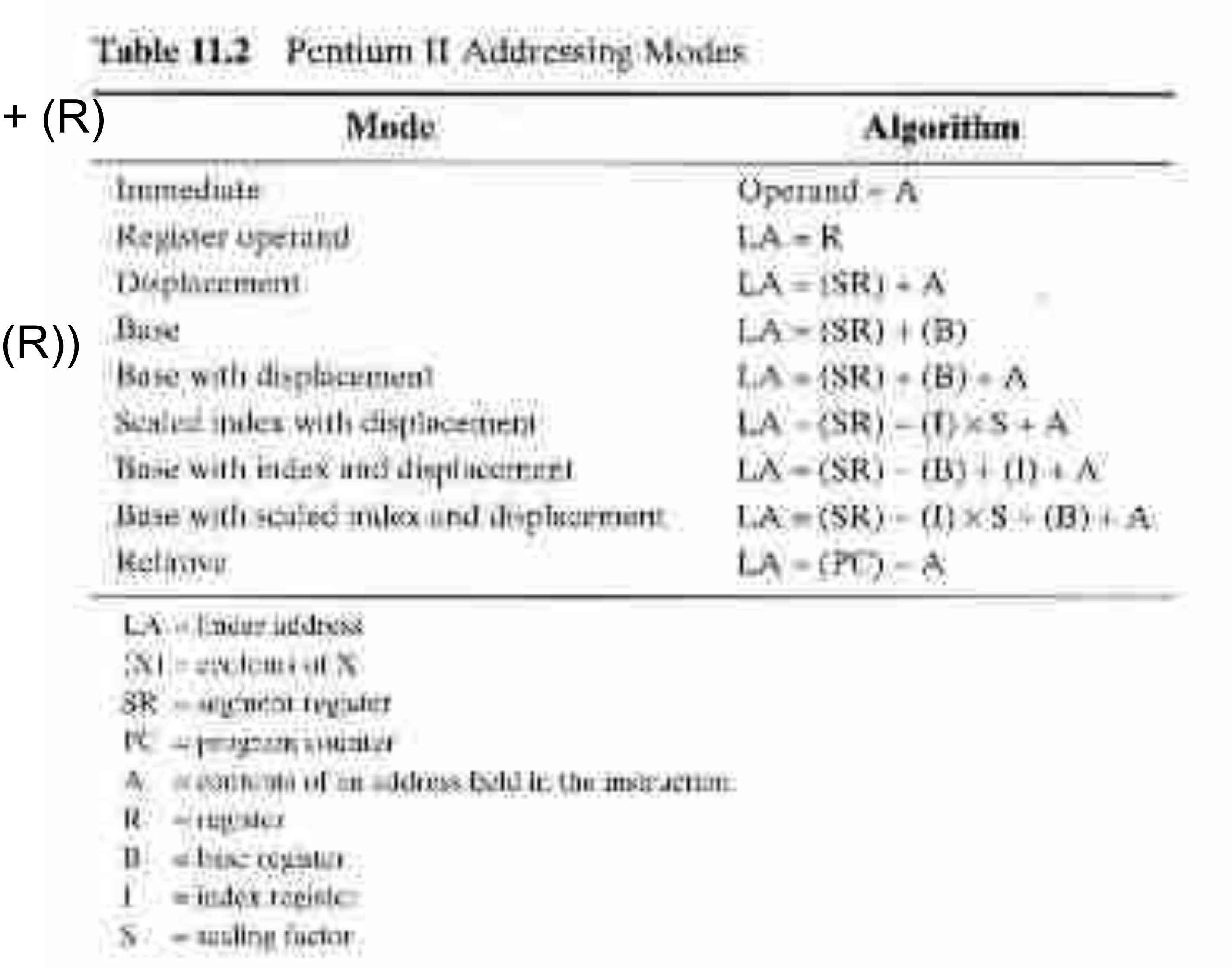


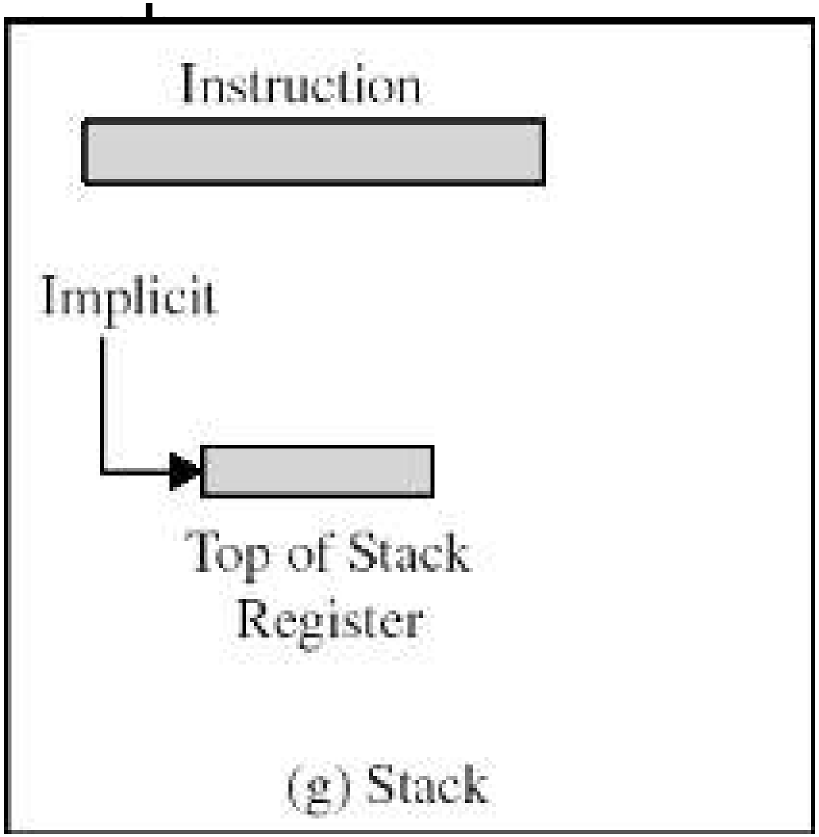
Table 11.2 Pentium II Addressing Modes

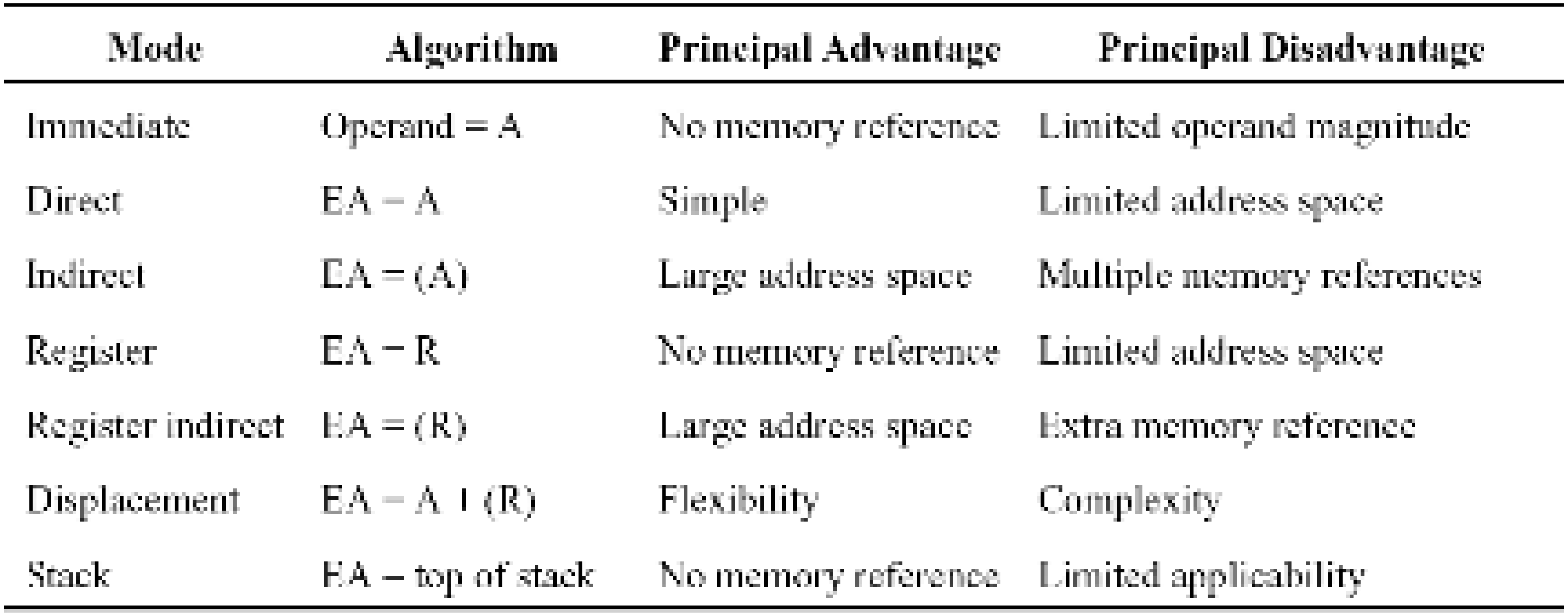
Mode	Algorithm
Immediate	Operand = A
Register operand	LA = R
Displacement	LA = (SR) + A
Base	LA = (SR) + (B)
Base with displacement	LA = (SR) + (B) + A
Scaled index with displacement	LA = (SR) + (I) × S + A
Base with index and displacement	LA = (SR) + (B) + (I) + A
Base with scaled index and displacement	LA = (SR) + (I) × S + (B) + A
Relative	LA = (PC) + A

LA = linear address
(X) = contents of X
SR = segment register
PC = program counter
A = contents of an address field in the instruction
R = register
B = base register
I = index register
S = scaling factor

Stack Addressing

- Operand is (implicitly) on top of stack
- e.g.
  - ADD Pop top two items from stack and add and





Mode	Algorithm	Principal Advantage	Principal Disadvantage
Immediate	Operand = A	No memory reference	Limited operand magnitude
Direct	EA = A	Simple	Limited address space
Indirect	EA = (A)	Large address space	Multiple memory references
Register	EA = R	No memory reference	Limited address space
Register indirect	EA = (R)	Large address space	Extra memory reference
Displacement	EA = A + (R)	Flexibility	Complexity
Stack	EA = top of stack	No memory reference	Limited applicability

Question Bank(Objective Type Questions)

1. In Microprocessor

a. program is stored in memory and data is stored in the registers.

b. program is stored in the registers and data is stored in memory.

c. both program and data are stored in the memory.

d. both program and data are stored in the registers.

2 .A Microprocessor contains.

a. most of the control and arithmetic logic functions of a computer.

b. most of the RAM .

c. most of the ROM.

d. peripheral drivers.

3. A PC in a micro-computer.

a. counts the number of instructions executed in a run.

b. counts the number of programs run after startying.

c. counts the points to the next executable instruction

d. points to the present instruction being executed.

4. An instruction cycle is made up of:

a. one or more execute cycles

b. one or more fetch cycles

c. one opcode and one execute cycle

d. none of the above.

5. The number of minimum clock cycles in a machine cycle for 8085 are.

a. 1

b. 2

c. 3

d. 5

6. In a 8-bit microprocessor ,the fetch reqired to fetch a 8 bytes instruction will be:

a. 1

b. 2

c. 3

d. depends on computer design

7. The maximun integer ahich can be stored on an 8-bit accunulator is

a. 2kb

b. 200

c. 224

d. 255.

8. The address bus of intel 8085 is 16 bit wide and hence the memory which can be accessed by this address bus is :

a.112

b.4kb

c.16kb

d.64 kb

9. A byte corresponds to

(a) 4 bits

(b) 8 bits

(c) 16 bits

(d) 32 bits

10. A gigabyte represents

(a) 1 billion bytes

(b) 1000 kilobytes

(c) 230 bytes

(d) 1024 bytes

11. A megabyte represents

(a) 1 million bytes

(b) 1000 kilobytes

(c) 220 bytes

(d) 1024 bytes

12. A Kb corresponds to

(a) 1024 bits

(b) 1000 bytes

(c) 210 bytes

(d) 210 bits

13. A superscalar processor has

(a) multiple functional units

(b) a high clock speed

(c) a large amount of RAM

(d) many I/O ports

14. A 32-bit processor has

(a) 32 registers

(b) 32 I/O devices

(c) 32 Mb of RAM

(d) a 32-bit bus or 32-bit registers

15. A 20-bit address busallows access to a memory of capacity

(a) 1 Mb

(b) 2 Mb

(c) 32Mb

(d) 64 Mb

16. A 32-bit **address bus** allows access to a memory of capacity

(a) 64 Mb

(b) 16 Mb

(c) 1 Gb

(d) 4 Gb

17.Clock speed is measured in

(a) bits per second

(b) baud

(c) bytes

(d) Hertz

18. An FPU

(a) makes integer arithmetic faster

(b) makes pipelining more efficient

(c) increases RAM capacity

(d) makes some arithmetic calculations faster

19. Pipelining improves CPU performance due to

(a) reduced memory access time

(b) increased clock speed

(c) the introduction of parallellism

(d) additional functional units

20. The system bus is made up of

(a) data bus

(b) data bus and address bus

(c) data bus and control bus

(d) data bus, control bus and address bus

21. A machine cycle refers to

(a) fetching an instruction

(b) clock speed

(c) fetching, decoding and executing an instruction

(d)executing an instruction

22. A Pentium processor comprises

(a) more than 1 million transistors

(b) more than 3 million transistors

(c) 500,000 transistors

(d) 900,000 transistors

23. Which of the following is **NOT** a type of processor

(a) PowerPC 601

(b) Motorola 8086

(c) Motorola 68000

(d) Intel Pentium

24. An RS-232 interface is

(a) a parallel interface

(b) a serial interface

(c) printer interface

(d) a modem interface

25. Multiprogramming refers to

(a) having several programs in RAM at the same time

(b)multitasking

(c) writing programs in multiple languages

(d) none of the previous

26. Multitasking refers to

(a) having several programs in RAM at the same time

(b) the ability to run 2 or more programs concurrently

(c) writing programs in multiple languages

(d) none of the previous

27. Multiprogramming is a prerequisite for

(a) multitasking

(b) an operating system

(c) to run more than one program at the same time

(d) none of the above

28. Multiprocessing is

(a) same as multitasking

(b) same as multiprogramming

(c)multiuser

(d) involves using more than one processor at the same time

29. A compiler is

(a) a fast interpreter

(b) slower than an interpreter

(c) converts a program to machine code

(d) none of the previous

30. An interpreter is

(a) faster than a compiler

(b) translates and executes programs statement by statement

(c) converts a program to machine code

(d) none of the previous

31. Which of the following is **not** part of the processor

(a) the ALU

(b) the CU

(c) the registers

(d) the system bus

32. Pipelining improves CPU performance due to

(a) reduced memory access time

(b) increased clock speed

(c) the introduction of parallellism

(d) additional functional units

33. The Pentium processor is

(a) 16-bit

(b) 32-bit

(c) 64 bit

(d) 8-bit

34. The IBM/Motorola PowerPC 601 processor is

(a) 16-bit

(b) 32-bit

(c) 64 bit

(d) 8-bit

35. An assembly language instruction

(a) always has a label

(b) always takes at least 1 operand

(c) always has an operation field

(d) always modifies the status register

36. An arithmetic instruction always modifies the

(a) stack pointer

(b) status register

(c) program counter

(d) an index register

37. A conditional jump instruction

(a) always cause a transfer of control

(b) always involves the use of the status register

(c) always modifies the program counter

(d) always involves testing the Zero flag

38. An interrupt instruction

(a) causes an unconditional transfer of control

(b) causes a conditional transfer of control

(c) modifies the status register

(d) is an I/O instruction

39. A data movement instruction will

(a) modify the status register

(b) modify the stack pointer

(c) modify the program counter

(d) transfer data from one location to another

40. The memory address register is used to store

(a) data to be transferred to memory

(b) data that has been transferred from memory

(c) the address of a memory location

(d) an instruction that has been transferred from memory.

41. The memory data register is used to store

(a) data to be transferred to or from memory

(b) data to be transferred to the stack

(c) the address of a memory location

(d) an instruction that has been transferred from memory

42. The instruction register stores

(a) an instruction that has been decoded

(b) an instruction that has been fetched from memory

(c) an instruction that has been executed

(d) the address of the next instruction to be executed

43 The program counter

(a) stores the address of the instruction that is currently being executed

(b) stores the next instruction to be executed

(c) stores the address of the next instruction to be executed

(d) stores the instruction that is being currently executed.

44. The stack pointer stores

(a) the address of the stack in memory

(b) address of the last item pushed on the stack

(c) the address of the next free stack location

(d) the address of the last item popped from the stack

45. Memory mapped I/O involves

(a) transferring information between memory locations

(b) transferring information between registers and memory

(c) transferring information between the CPU and I/O devices in the same way as between the CPU and memory

(d) transferring information between I/O devices and memory

46. A hardware interrupt is

(a) also called an internal interrupt

(b) also called an external interrupt

(c) an I/O interrupt

(d) a clock interrupt

47. An assembly language program is typically

(a) non-portable

(b) shorter than an equivalent HLL program

(c) harder to read than a machine code program

(d)slower to execute than a compiled HLL program

48. Programs are written in assembly language because they

(a) run faster than HLL programs

(b) are portable

(c) easier to write than machine code programs

(d) they allow the programmer access to registers or instructions that are not usually provided by a HLL

49. An assembly language program is translated to machine code by

(a) an assembler

(b) a compiler

(c) an interpreter

(d) a linker

50. An assembly language directive is

(a) the same as an instruction

(b) used to define space for variables

(c) used to start a program

(d) to give commands to an assembler