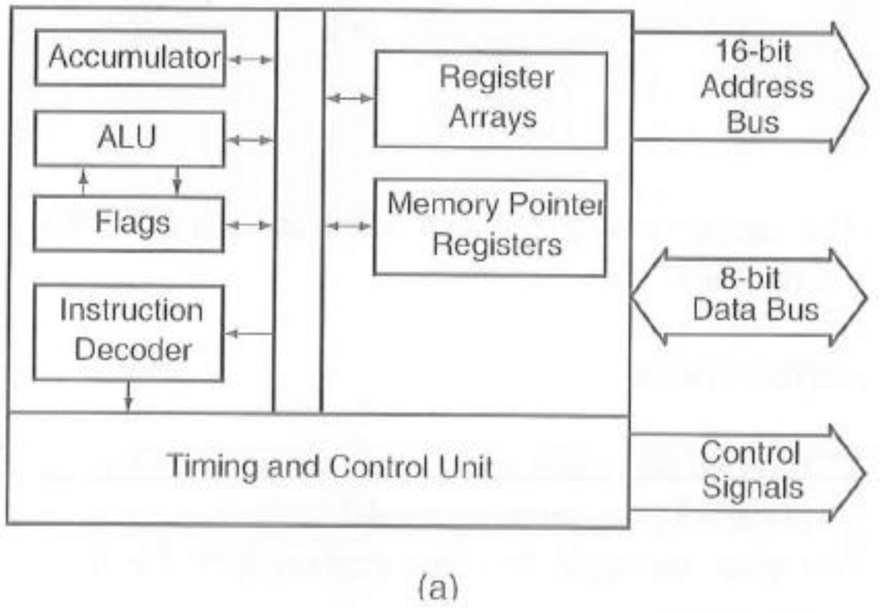
HOW TO WRITE & EXECUTE AN ASSEMBLY LANGUAGE PROGRAM

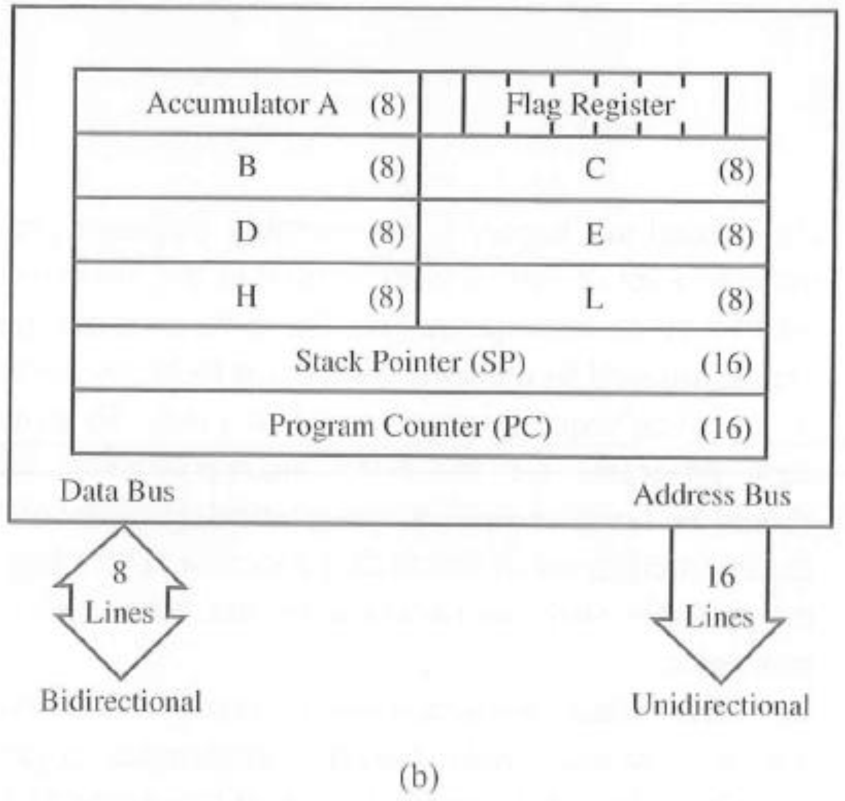
8085 Assembly Language Programming - RECAP

- 8085 Hardware model
- 8085 Programming model
- Flag Register
- Opcode and operand
- Instruction set
- Addressing mode
- Logical steps to solve a simple programming problem

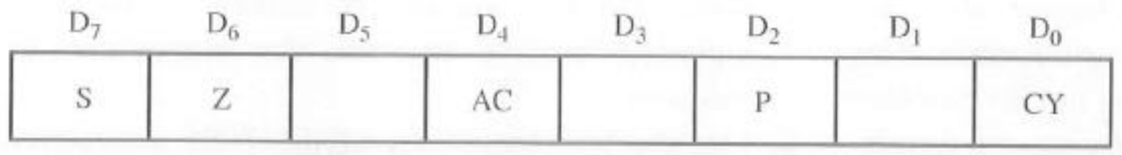
8085 Hardware & Programming model

- **A model is a conceptual representation of a real object.**
- **A Microprocessor can be represented in terms of its hardware (physical electronics components)**
- **and a programming model (information needed to write programs).**





8085 Hardware Model (a), Programming Model (b),



Flag Register (c)

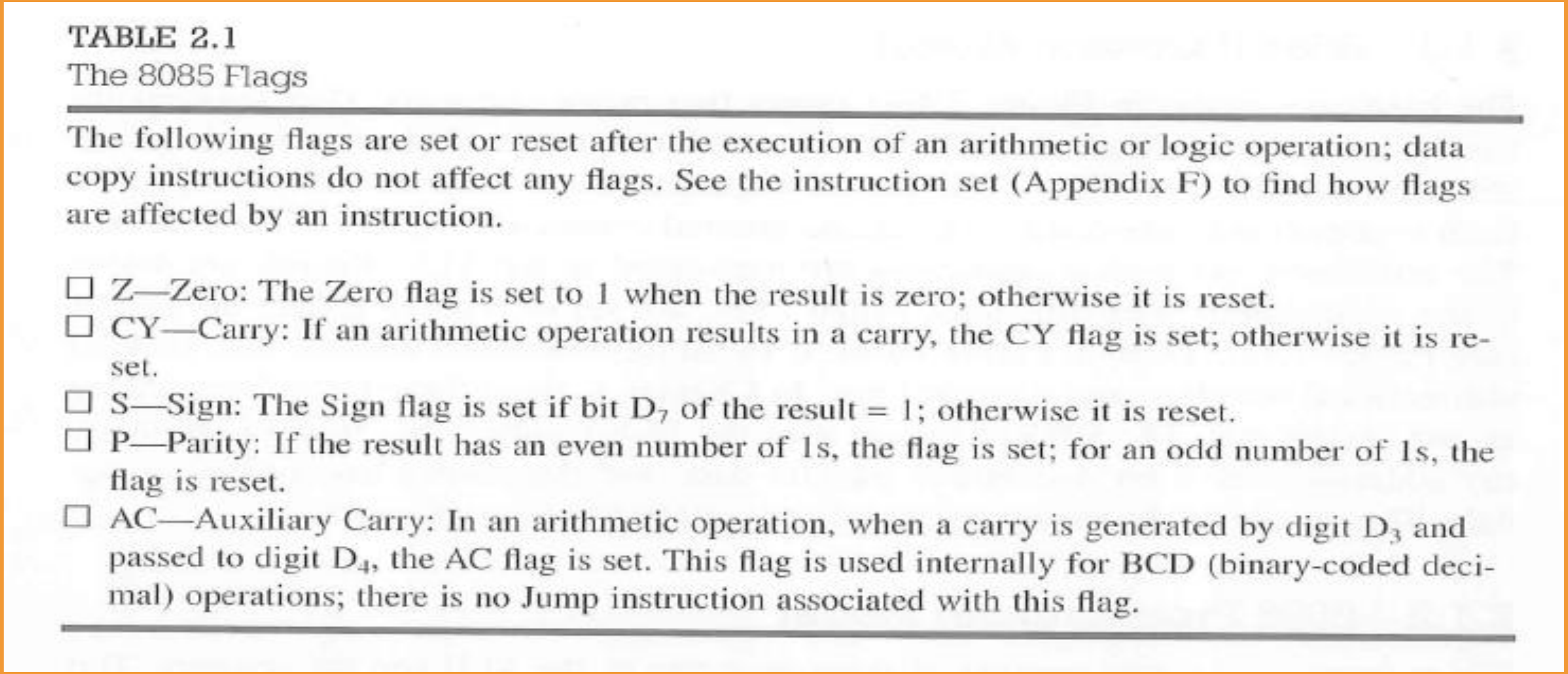


TABLE 2.1
The 8085 Flags

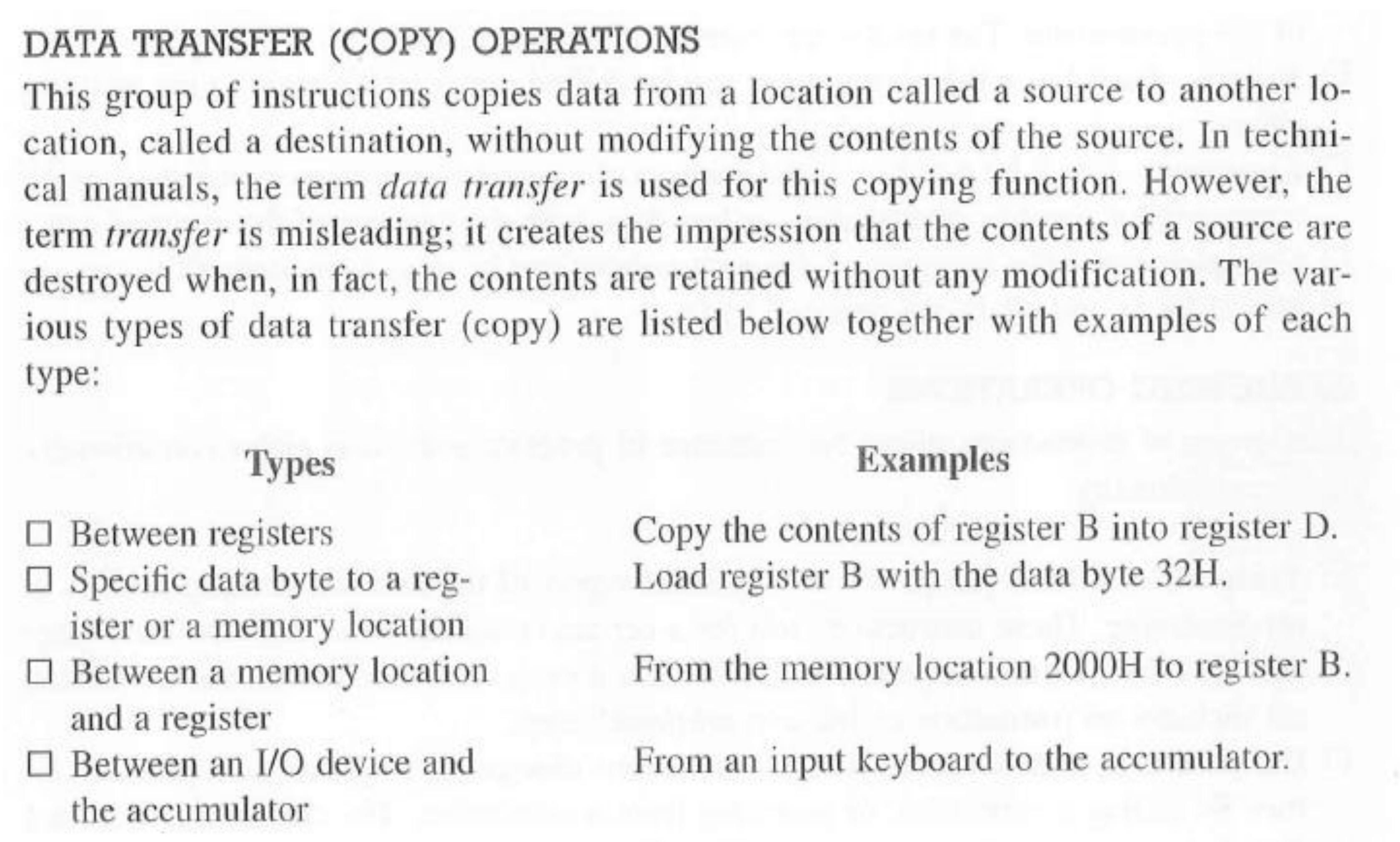
The following flags are set or reset after the execution of an arithmetic or logic operation; data copy instructions do not affect any flags. See the instruction set (Appendix F) to find how flags are affected by an instruction.

- ☐ Z—Zero: The Zero flag is set to 1 when the result is zero; otherwise it is reset.
- ☐ CY—Carry: If an arithmetic operation results in a carry, the CY flag is set; otherwise it is reset.
- ☐ S—Sign: The Sign flag is set if bit D_7 of the result = 1; otherwise it is reset.
- ☐ P—Parity: If the result has an even number of 1s, the flag is set; for an odd number of 1s, the flag is reset.
- ☐ AC—Auxiliary Carry: In an arithmetic operation, when a carry is generated by digit D_3 and passed to digit D_4, the AC flag is set. This flag is used internally for BCD (binary-coded decimal) operations; there is no Jump instruction associated with this flag.

The 8085 instruction set- RECAP

1. Data transfer instructions
2. Arithmetic instructions
3. Logical instructions
4. Branching instructions and
5. Control instructions.

1. Data transfer instructions

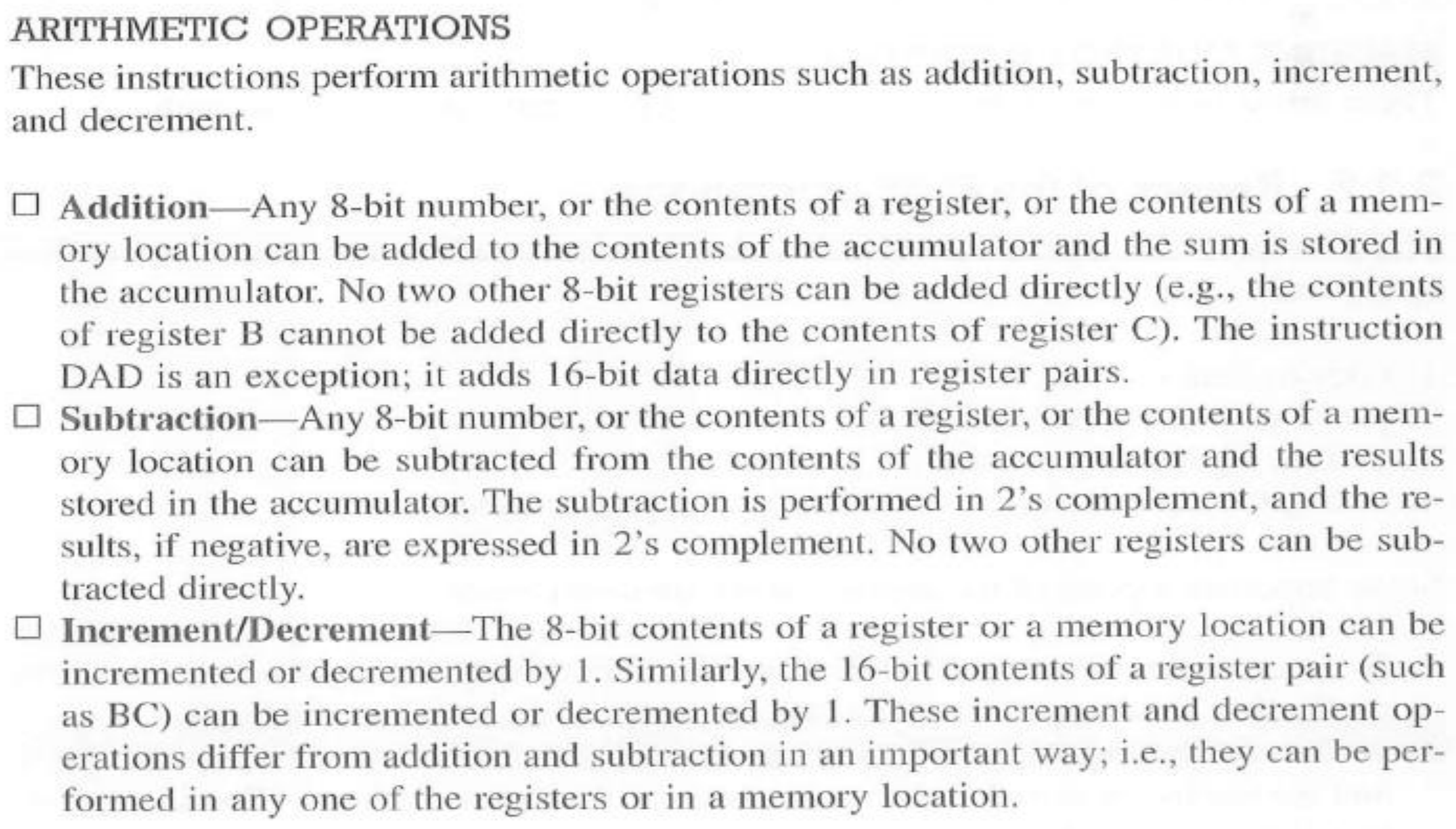


DATA TRANSFER (COPY) OPERATIONS

This group of instructions copies data from a location called a source to another location, called a destination, without modifying the contents of the source. In technical manuals, the term *data transfer* is used for this copying function. However, the term *transfer* is misleading; it creates the impression that the contents of a source are destroyed when, in fact, the contents are retained without any modification. The various types of data transfer (copy) are listed below together with examples of each type:

Types	Examples
☐ Between registers	Copy the contents of register B into register D.
☐ Specific data byte to a register or a memory location	Load register B with the data byte 32H.
☐ Between a memory location and a register	From the memory location 2000H to register B.
☐ Between an I/O device and the accumulator	From an input keyboard to the accumulator.

2. Arithmetic instructions

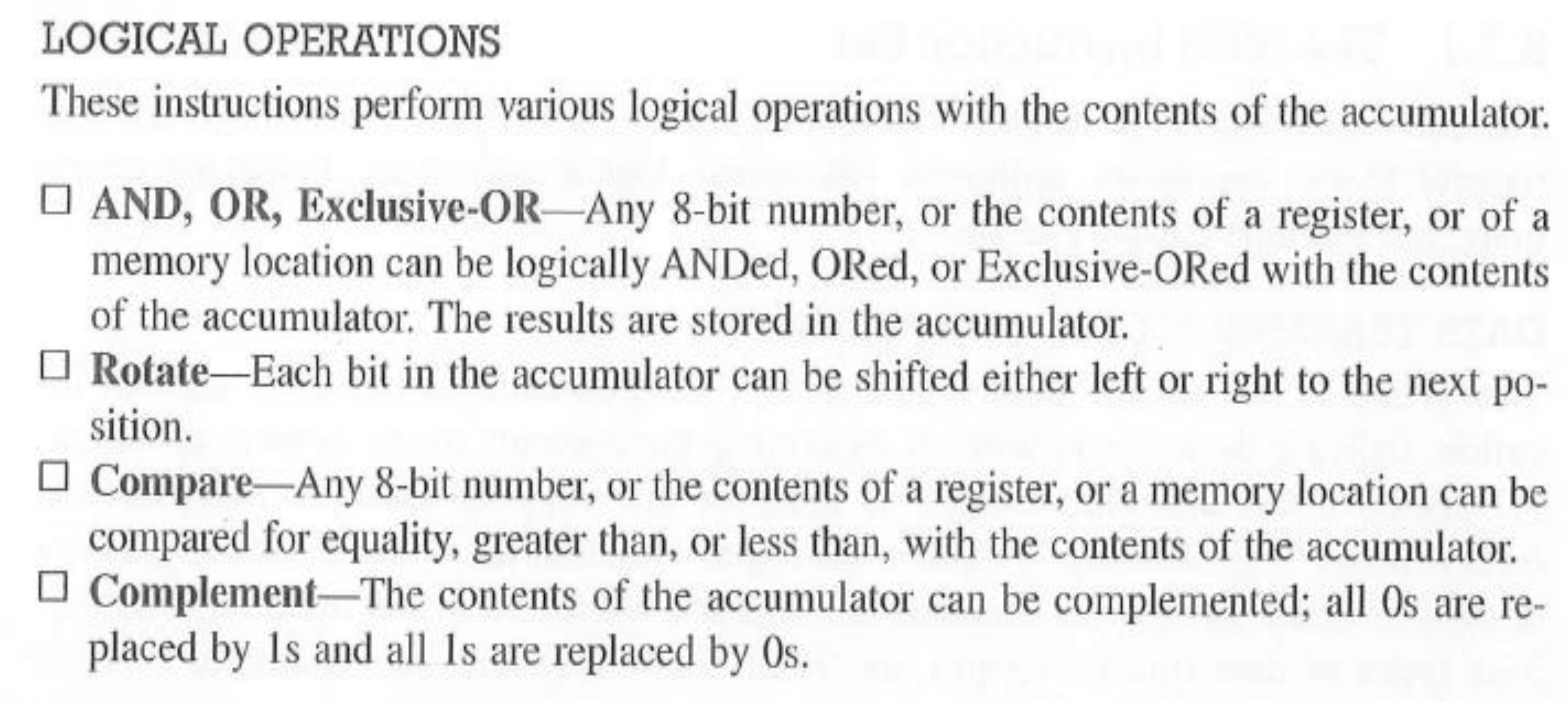


ARITHMETIC OPERATIONS

These instructions perform arithmetic operations such as addition, subtraction, increment, and decrement.

- ☐ **Addition**—Any 8-bit number, or the contents of a register, or the contents of a memory location can be added to the contents of the accumulator and the sum is stored in the accumulator. No two other 8-bit registers can be added directly (e.g., the contents of register B cannot be added directly to the contents of register C). The instruction DAD is an exception; it adds 16-bit data directly in register pairs.
- ☐ **Subtraction**—Any 8-bit number, or the contents of a register, or the contents of a memory location can be subtracted from the contents of the accumulator and the results stored in the accumulator. The subtraction is performed in 2's complement, and the results, if negative, are expressed in 2's complement. No two other 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. Similarly, the 16-bit contents of a register pair (such as BC) can be incremented or decremented by 1. These increment and decrement operations differ from addition and subtraction in an important way; i.e., they can be performed in any one of the registers or in a memory location.

3. Logical instructions

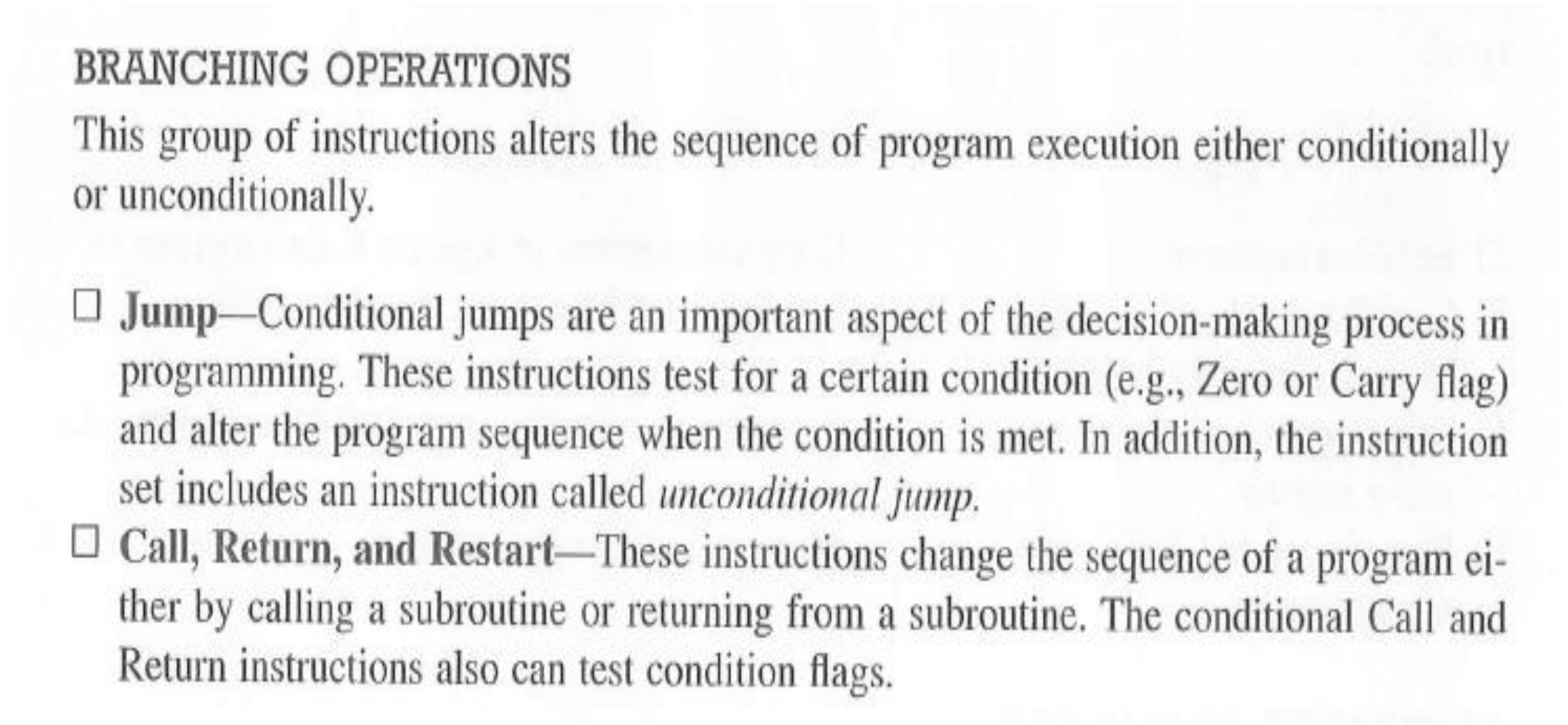


LOGICAL OPERATIONS

These instructions perform various logical operations with the contents of the accumulator.

- ☐ **AND, OR, Exclusive-OR**—Any 8-bit number, or the contents of a register, or of a memory location can be logically ANDed, ORed, or Exclusive-ORed with the contents of the accumulator. The results are stored in the accumulator.
- ☐ **Rotate**—Each bit in the accumulator can be shifted either left or right to the next position.
- ☐ **Compare**—Any 8-bit number, or the contents of a register, or a memory location can be compared for equality, greater than, or less than, with the contents of the accumulator.
- ☐ **Complement**—The contents of the accumulator can be complemented; all 0s are replaced by 1s and all 1s are replaced by 0s.

4. Branching instructions

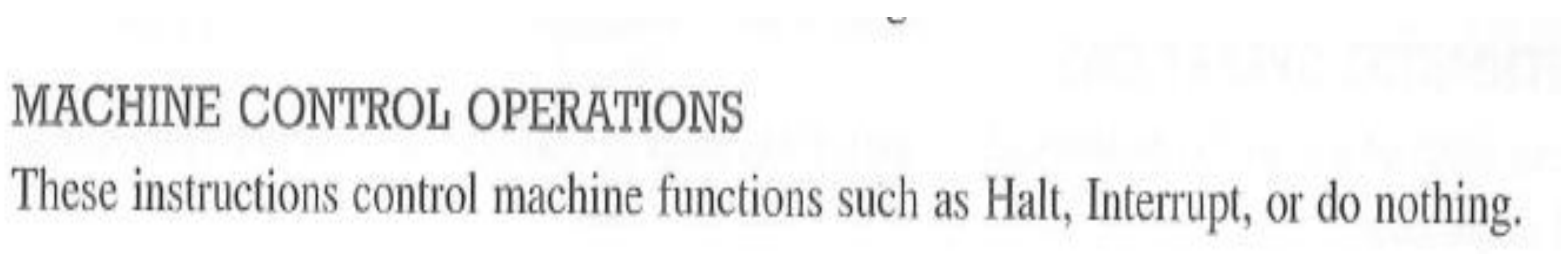


BRANCHING OPERATIONS

This group of instructions alters the sequence of program execution either conditionally or unconditionally.

- ☐ **Jump**—Conditional jumps are an important aspect of the decision-making process in programming. These instructions test for a certain condition (e.g., Zero or Carry flag) and alter the program sequence when the condition is met. In addition, the instruction set includes an instruction called *unconditional jump.*
- ☐ **Call, Return, and Restart**—These instructions change the sequence of a program either by calling a subroutine or returning from a subroutine. The conditional Call and Return instructions also can test condition flags.

5. Control instructions.

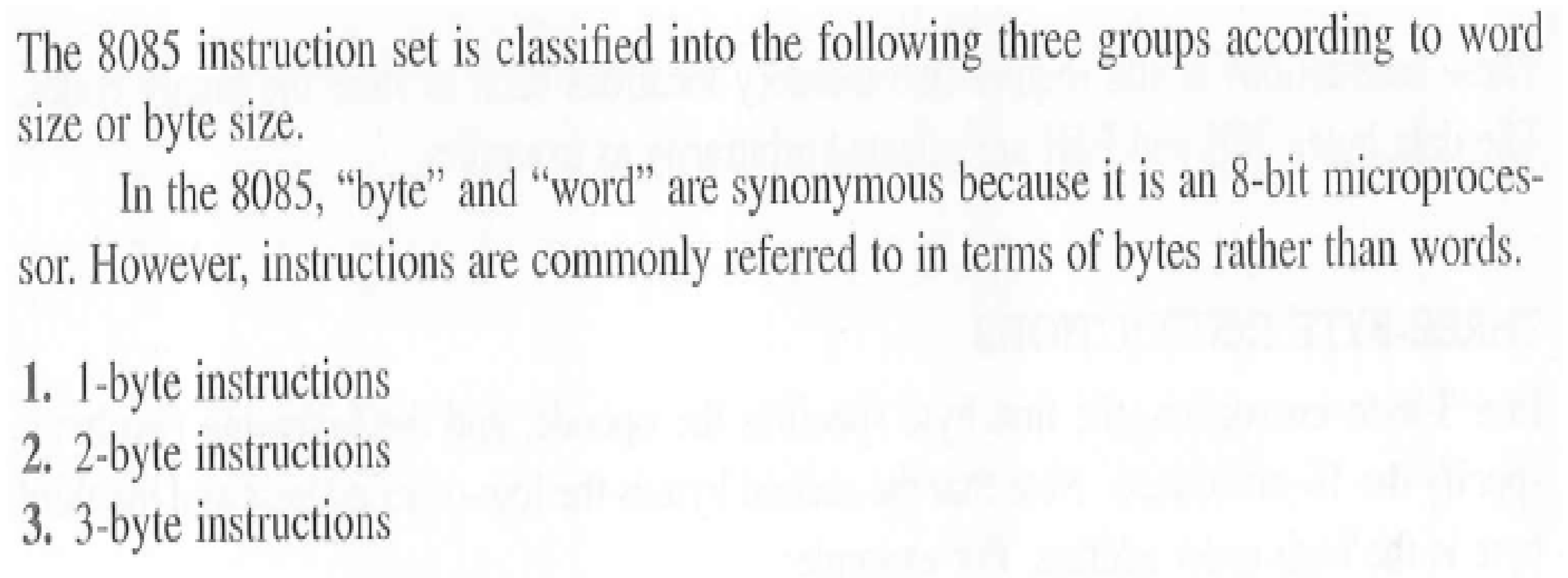

MACHINE CONTROL OPERATIONS

These instructions control machine functions such as Halt, Interrupt, or do nothing.

INSTRUCTION, DATA FORMAT & STORAGE

- **It is a command that can make the microprocessor perform a specific function.**
- **An Instruction is basically a combination of two parts one is operation code and the next is operands**
- **The opcode will be used to identify what kind of operation needs to be performed.**
- **Operands will be used to spot the source and destination**

INSTRUCTION WORD SIZE

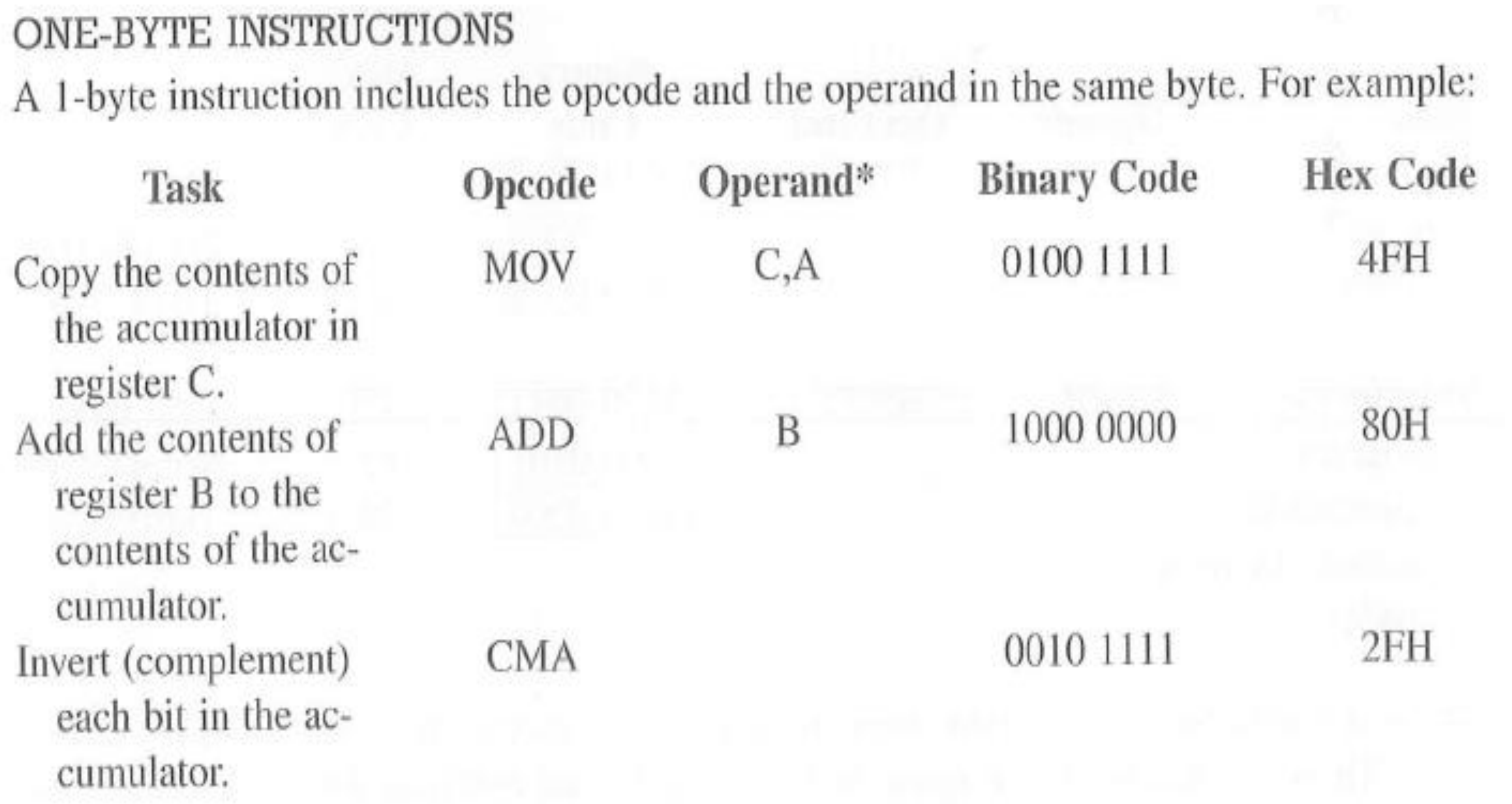


The 8085 instruction set is classified into the following three groups according to word size or byte size.

In the 8085, "byte" and "word" are synonymous because it is an 8-bit microprocessor. However, instructions are commonly referred to in terms of bytes rather than words.

1. 1-byte instructions
2. 2-byte instructions
3. 3-byte instructions

ONE BYTE INSTRUCTIONS

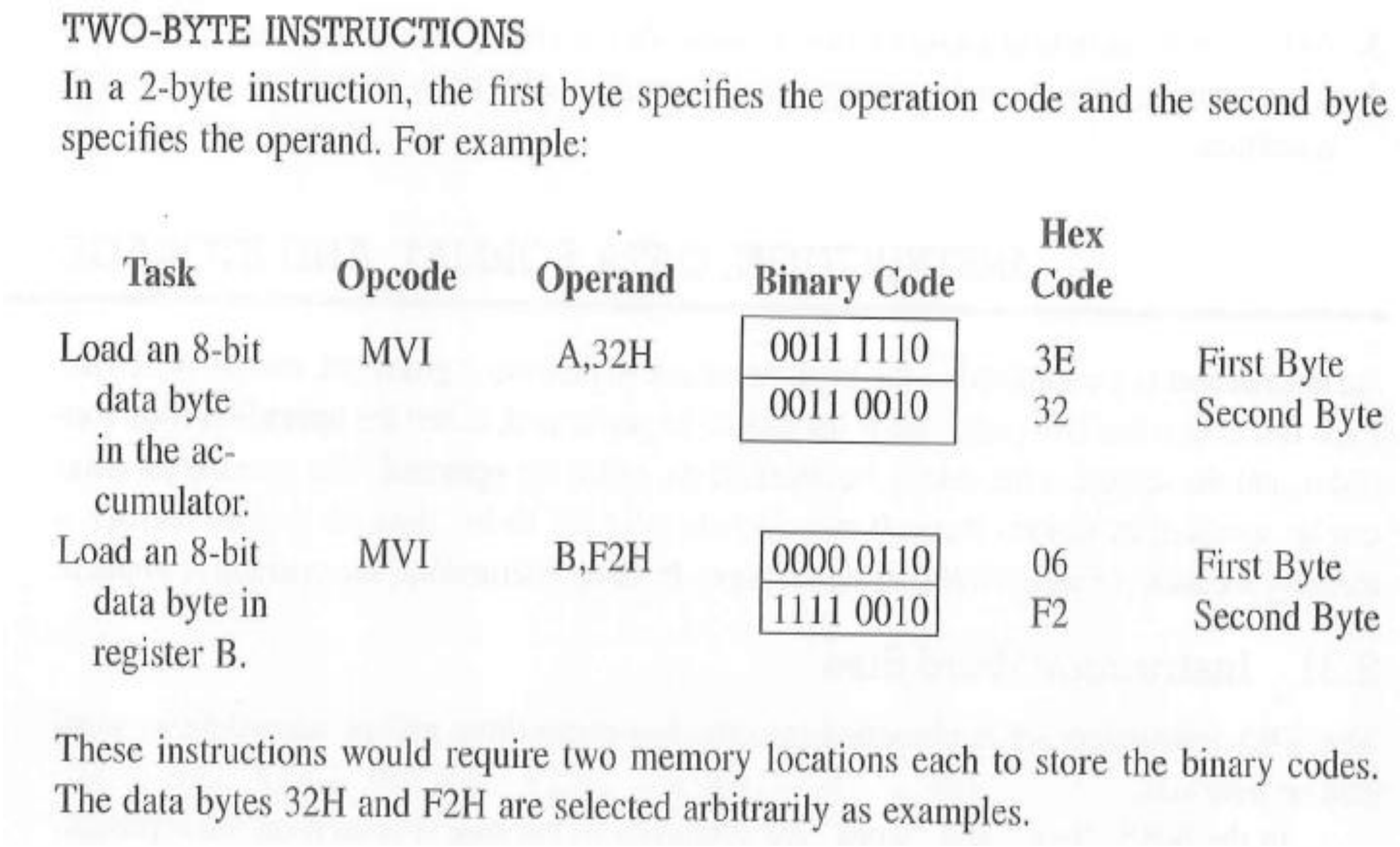


ONE-BYTE INSTRUCTIONS

A 1-byte instruction includes the opcode and the operand in the same byte. For example:

Task	Opcode	Operand*	Binary Code	Hex Code
Copy the contents of the accumulator in register C.	MOV	C,A	0100 1111	4FH
Add the contents of register B to the contents of the accumulator.	ADD	B	1000 0000	80H
Invert (complement) each bit in the accumulator.	CMA		0010 1111	2FH

TWO BYTE INSTRUCTIONS



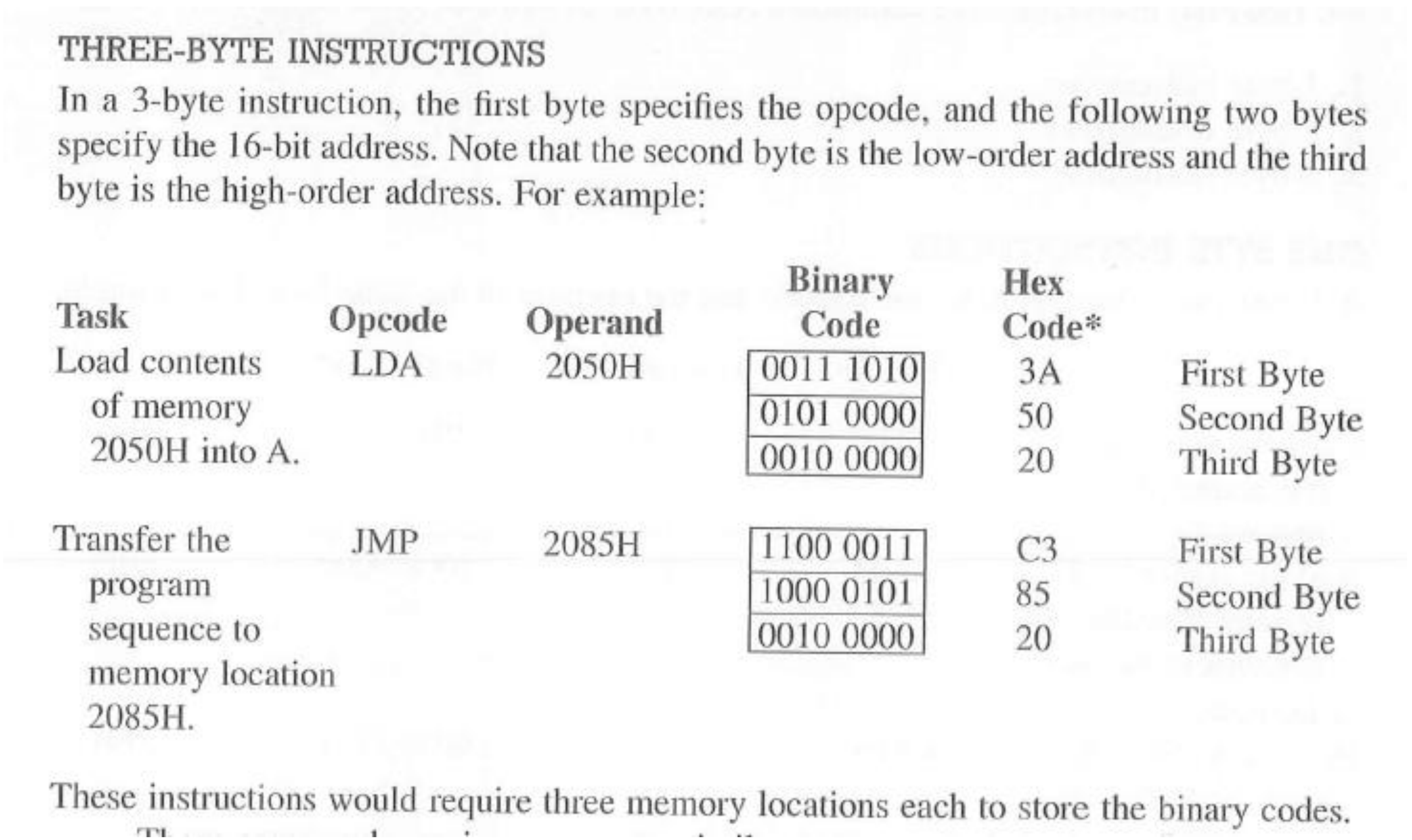
TWO-BYTE INSTRUCTIONS

In a 2-byte instruction, the first byte specifies the operation code and the second byte specifies the operand. For example:

Task	Opcode	Operand	Binary Code	Hex Code	
Load an 8-bit data byte in the accumulator.	MVI	A,32H	0011 1110	3E	First Byte
			0011 0010	32	Second Byte
Load an 8-bit data byte in register B.	MVI	B,F2H	0000 0110	06	First Byte
			1111 0010	F2	Second Byte

These instructions would require two memory locations each to store the binary codes. The data bytes 32H and F2H are selected arbitrarily as examples.

THREE BYTE INSTRUCTIONS



THREE-BYTE INSTRUCTIONS

In a 3-byte instruction, the first byte specifies the opcode, and the following two bytes specify the 16-bit address. Note that the second byte is the low-order address and the third byte is the high-order address. For example:

Task	Opcode	Operand	Binary Code	Hex Code*	
Load contents of memory 2050H into A.	LDA	2050H	0011 1010	3A	First Byte
			0101 0000	50	Second Byte
			0010 0000	20	Third Byte
Transfer the program sequence to memory location 2085H.	JMP	2085H	1100 0011	C3	First Byte
			1000 0101	85	Second Byte
			0010 0000	20	Third Byte

These instructions would require three memory locations each to store the binary codes.

ADDRESSING MODES- RECAP

8085 supports five types of addressing:

The addressing modes are

1. Immediate Addressing
2. Direct Addressing
3. Register Addressing
4. Register Indirect Addressing
5. Implicit Addressing

Mode	Description	Example
Immediate addressing mode	Simplest mode, where the instruction itself will have the information to be moved or to be processed. The data to be processed can be moved to any register or memory location.	◆ **MVI A,05H** - 05H is the data to be moved to Accumulator with MVI instruction which is abbreviation of Move Immediate. Similarly immediate addressing mode can be used in arithmetic operations as well. Simple example would be: ◆ ADI 01H, where 1 is added to the content already available in the accumulator.
Direct addressing mode	In this mode one of the operands will be the address itself where the information is available	◆ **LDA 2000H** – Where, 2000H is the address of the source and LDA is the instruction which is expanded as load accumulator. The contents available in the memory location 2000H will be stored in accumulator once this instruction is executed. ◆ **STA 2000H** – This is yet another instruction which falls in the category of direct addressing more. This instruction prompts the microprocessor to store the content of the accumulator into the specified address location 2000H.

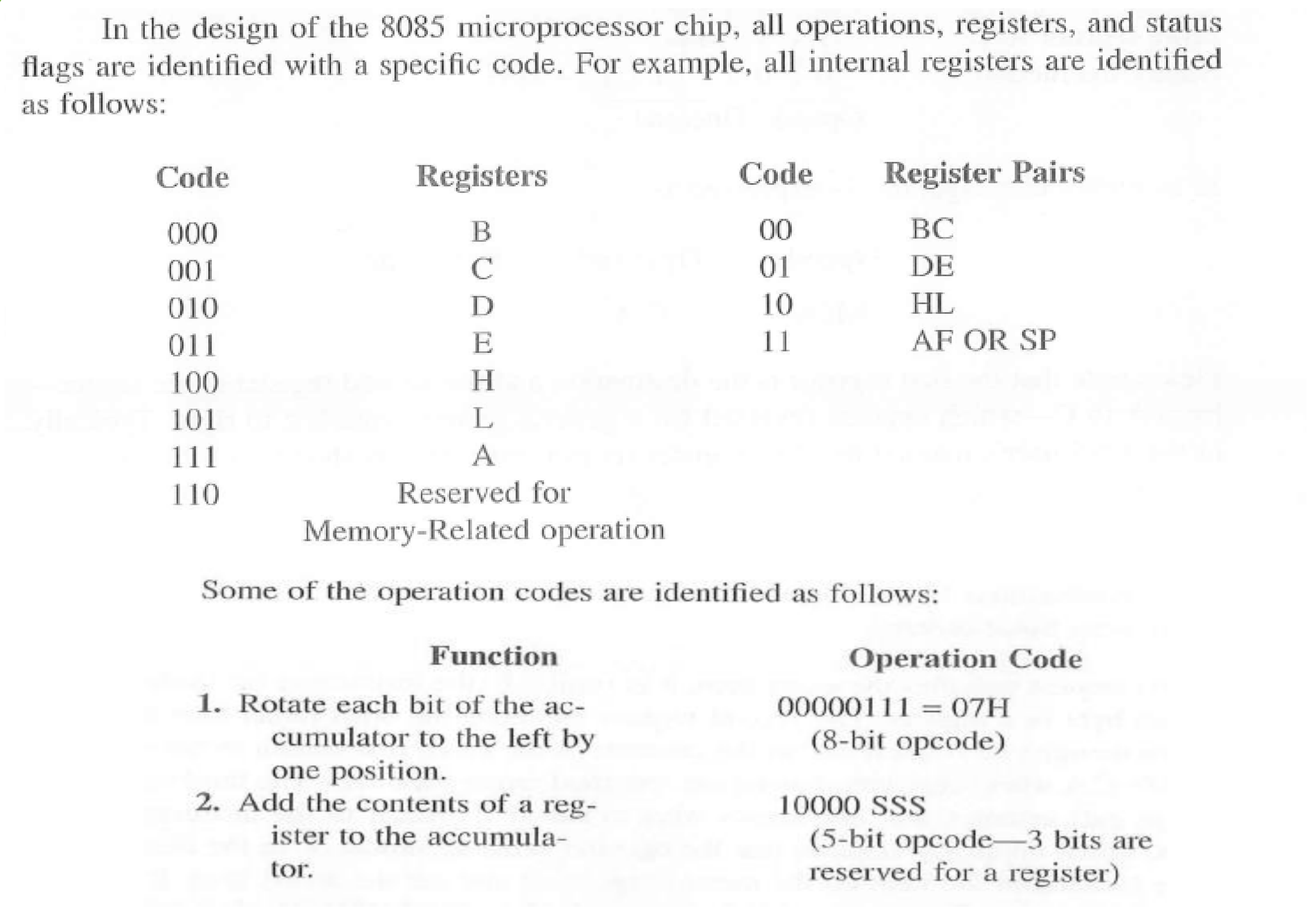
Indirect addressing mode	In the previous mode, address was specified right away, but here the place where the address is available will be specified.	**LDAX D** - Load the accumulator with the contents of the memory location whose address is stored in the register pair DE
Register addressing mode	Here the instruction will have the names of the register in which the data is available.	**MOV Rd, Rs – Rs** is the source register and Rd is the destination register. Data will be moved from source to destination with this instruction. One simple instance would be MOV C, D where the content of D is moved to the register C. Meanwhile the same addressing mode is applicable for arithmetic operations. **ADD C** will add the content of register C to accumulator and will store the result again in accumulator.
Implicit addressing mode	The instruction itself will have the data to be processed and there will not be any other operands.	CMC is one of such instructions which fall in this category of implicit addressing mode. CMC is the abbreviation of Complement Carry.

Addressing modes supported in 8085 microprocessor

ADRESSING MODE

1. Pass the butter(Direct value)	1. Immediate Addressing mode
2. Pass the bowl (Content in that)	2. Register Addressing mode
3. Let us eat (Assuming, one knows what to eat)	3. Implicit Addressing mode
4. I will have combination 17 on the menu (location of the item on the menu specified)	4. Direct Addressing mode
5. I will have what Susie ordered	5. Indirect Addressing mode

OPCODE FORMAT



In the design of the 8085 microprocessor chip, all operations, registers, and status flags are identified with a specific code. For example, all internal registers are identified as follows:

Code	Registers
000	B
001	C
010	D
011	E
100	H
101	L
111	A
110	Reserved for Memory-Related operation

Code	Register Pairs
00	BC
01	DE
10	HL
11	AF OR SP

Some of the operation codes are identified as follows:

Function	Operation Code
1. Rotate each bit of the accumulator to the left by one position.	00000111 = 07H (8-bit opcode)
2. Add the contents of a register to the accumulator.	10000 SSS (5-bit opcode—3 bits are reserved for a register)

This instruction is completed by adding the code of the register. For example,

Add	:	10000
Register B	:	000
to A	:	Implicit
Binary Instruction:		$\underbrace{10000}_{\text{Add}}\ \underbrace{000}_{\text{Reg.B}} = 80H$

In assembly language, this is expressed as

Opcode	Operand	Hex Code
ADD	B	80H

3. MOVE (Copy) the content of register Rs (source) to register Rd (destination)	01 2-bit Opcode for MOVE	DDD Reg. Rd	SSS Reg. Rs

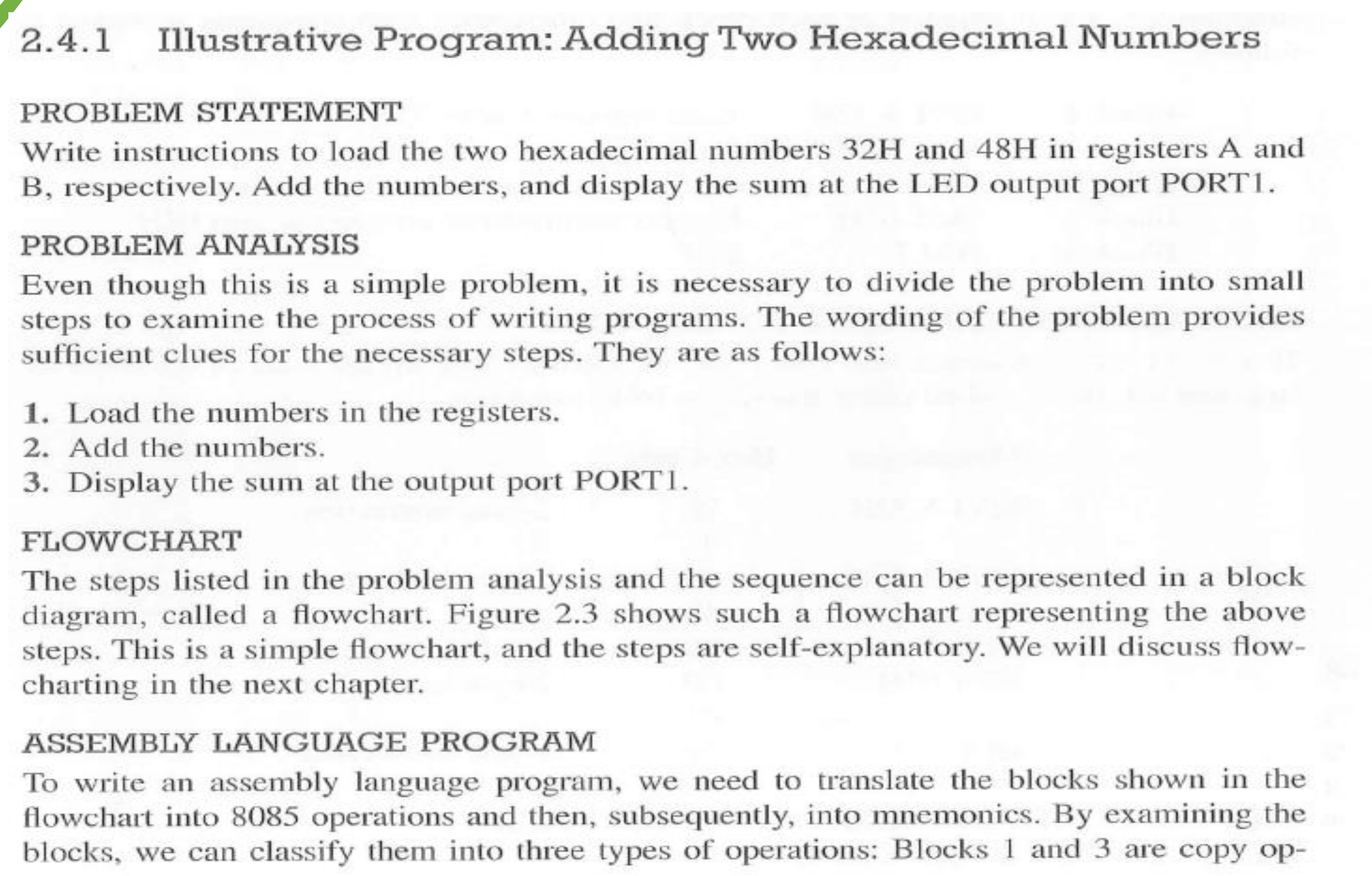
This instruction is completed by adding the codes of two registers. For example,

Move (copy) the content:		0 1
To register C	:	0 0 1 (DDD)
From register A	:	1 1 1 (SSS)
Binary Instruction	:	$\underbrace{0\ 1}_{\text{Opcode}}\ \underbrace{0\ 0\ 1\ \ 1\ 1\ 1}_{\text{Operand}} \rightarrow 4FH$

In assembly language, this is expressed as

Opcode	Operand	Hex Code
MOV	C,A	4F

HOW TO WRITE, ASSSEMBLE AND EXECUTE A SIMPLE PROGRAM



2.4.1 Illustrative Program: Adding Two Hexadecimal Numbers

PROBLEM STATEMENT

Write instructions to load the two hexadecimal numbers 32H and 48H in registers A and B, respectively. Add the numbers, and display the sum at the LED output port PORT1.

PROBLEM ANALYSIS

Even though this is a simple problem, it is necessary to divide the problem into small steps to examine the process of writing programs. The wording of the problem provides sufficient clues for the necessary steps. They are as follows:

1. Load the numbers in the registers.
2. Add the numbers.
3. Display the sum at the output port PORT1.

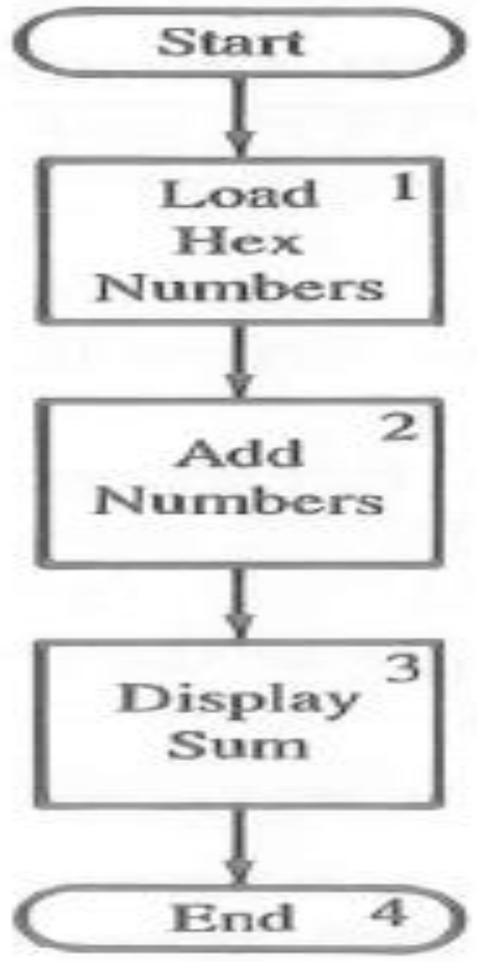
FLOWCHART

The steps listed in the problem analysis and the sequence can be represented in a block diagram, called a flowchart. Figure 2.3 shows such a flowchart representing the above steps. This is a simple flowchart, and the steps are self-explanatory. We will discuss flow-charting in the next chapter.

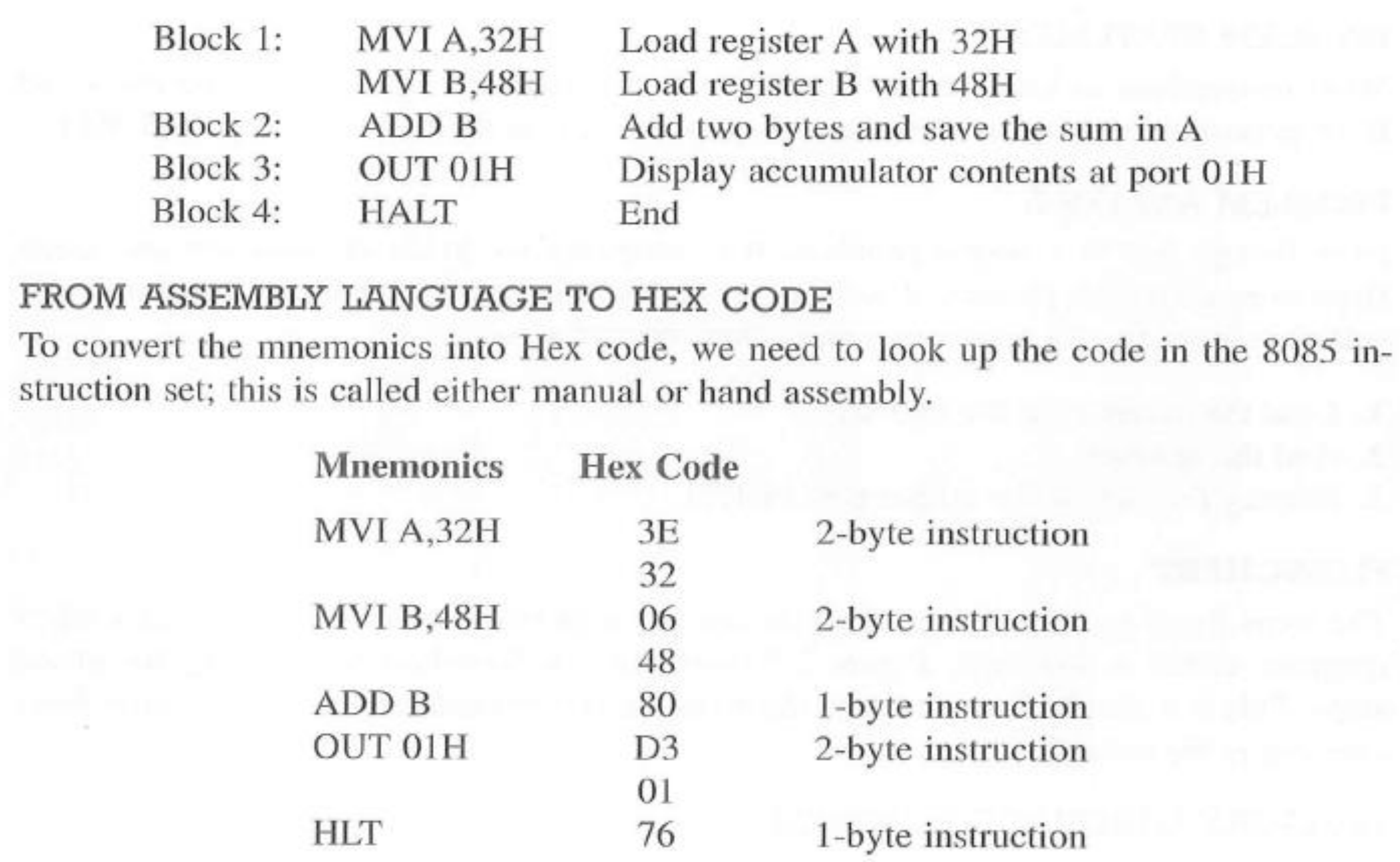
ASSEMBLY LANGUAGE PROGRAM

To write an assembly language program, we need to translate the blocks shown in the flowchart into 8085 operations and then, subsequently, into mnemonics. By examining the blocks, we can classify them into three types of operations: Blocks 1 and 3 are copy op-

Flow chart



Assembly Language Program



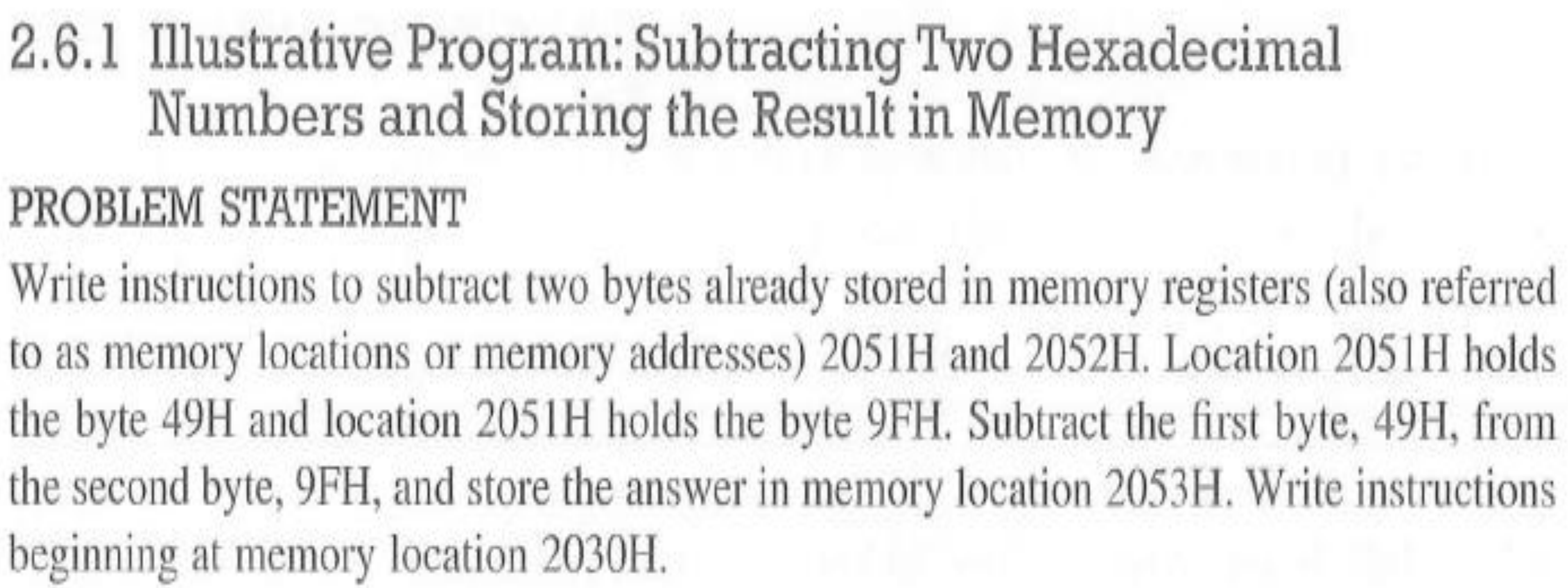
Block 1:	MVI A,32H	Load register A with 32H
	MVI B,48H	Load register B with 48H
Block 2:	ADD B	Add two bytes and save the sum in A
Block 3:	OUT 01H	Display accumulator contents at port 01H
Block 4:	HALT	End

FROM ASSEMBLY LANGUAGE TO HEX CODE

To convert the mnemonics into Hex code, we need to look up the code in the 8085 instruction set; this is called either manual or hand assembly.

Mnemonics	Hex Code	
MVI A,32H	3E	2-byte instruction
	32	
MVI B,48H	06	2-byte instruction
	48	
ADD B	80	1-byte instruction
OUT 01H	D3	2-byte instruction
	01	
HLT	76	1-byte instruction

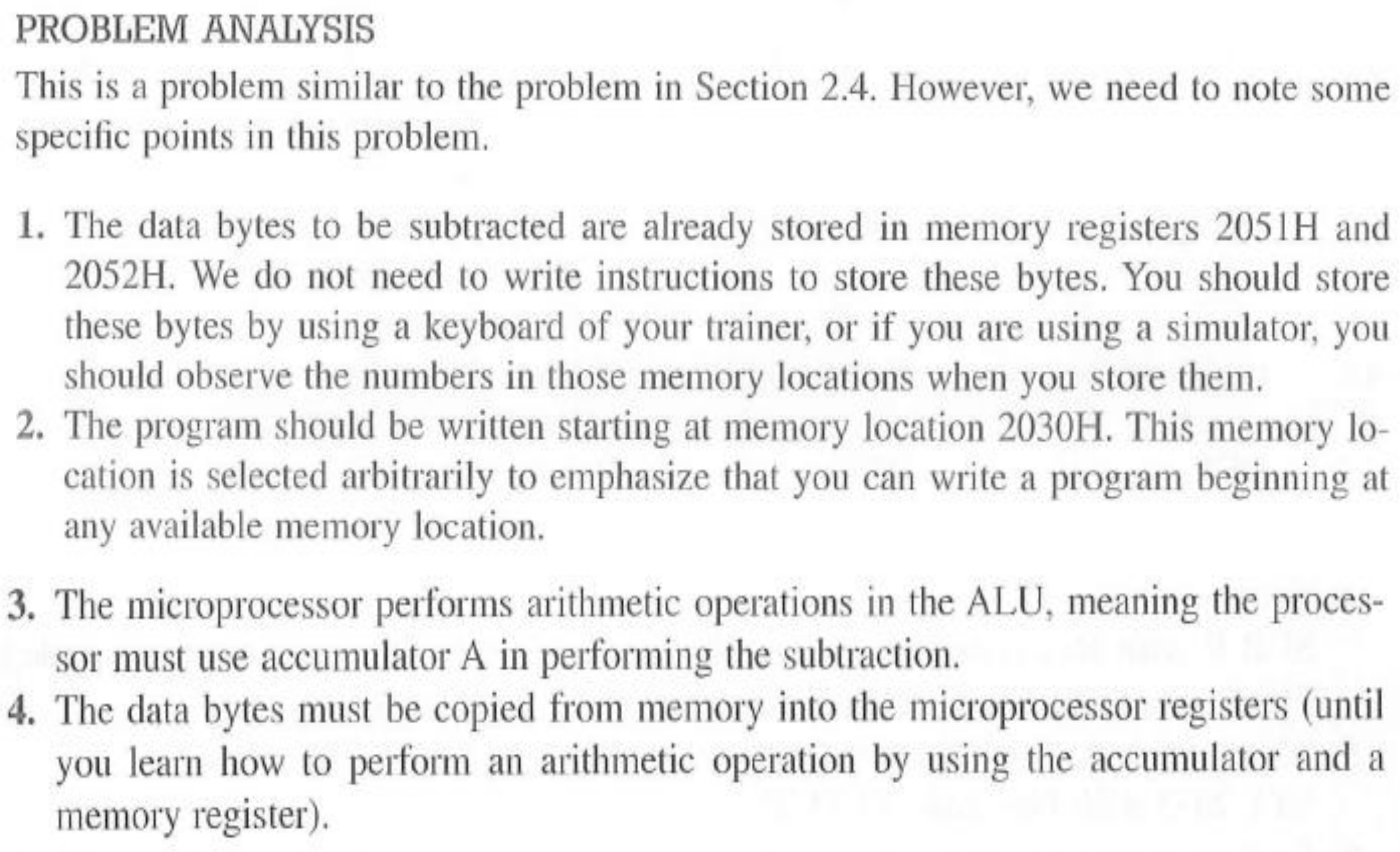
WRITING & HAND ASSEMBLING A PROGRAM



2.6.1 Illustrative Program: Subtracting Two Hexadecimal Numbers and Storing the Result in Memory

PROBLEM STATEMENT

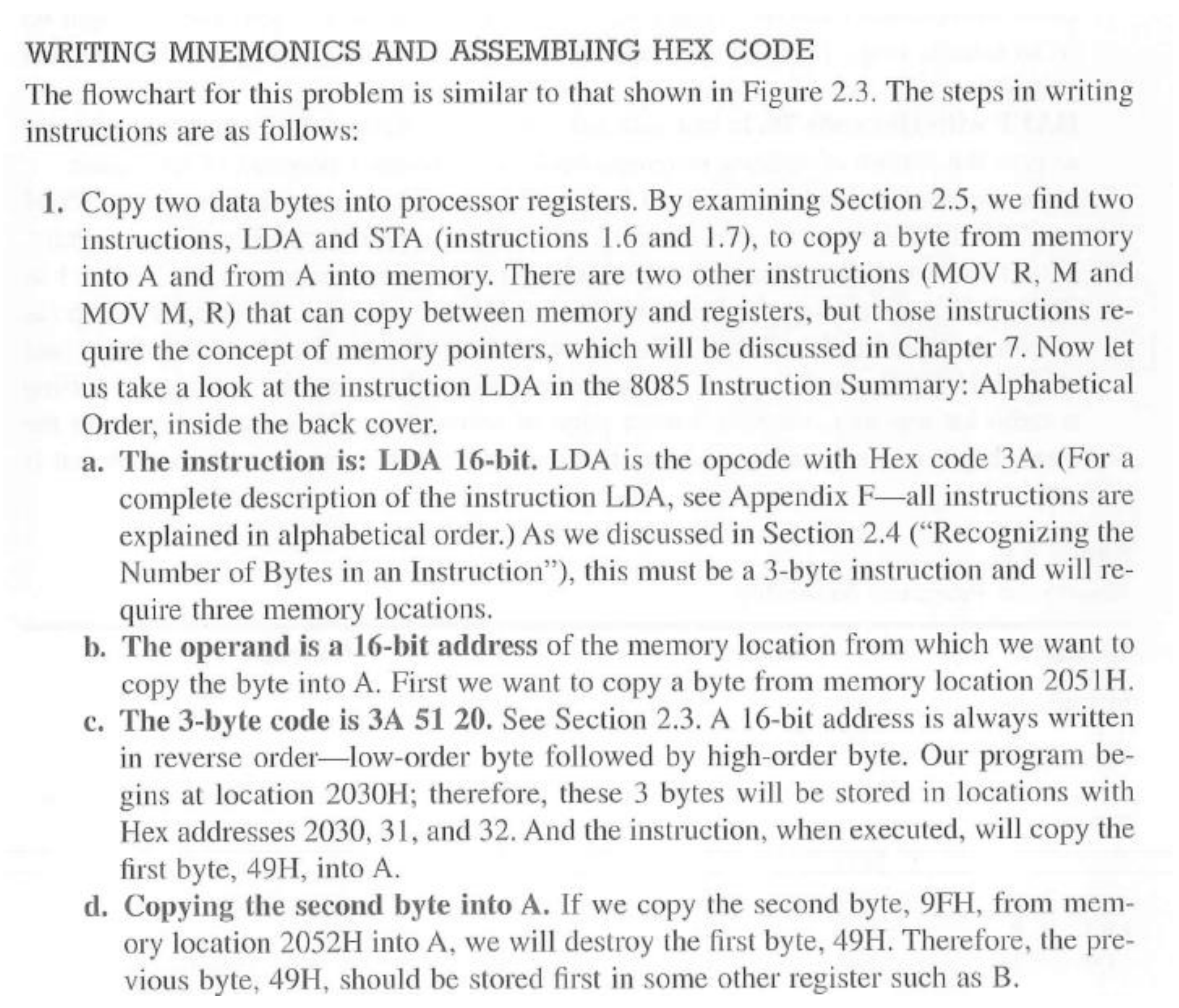
Write instructions to subtract two bytes already stored in memory registers (also referred to as memory locations or memory addresses) 2051H and 2052H. Location 2051H holds the byte 49H and location 2051H holds the byte 9FH. Subtract the first byte, 49H, from the second byte, 9FH, and store the answer in memory location 2053H. Write instructions beginning at memory location 2030H.



PROBLEM ANALYSIS

This is a problem similar to the problem in Section 2.4. However, we need to note some specific points in this problem.

1. The data bytes to be subtracted are already stored in memory registers 2051H and 2052H. We do not need to write instructions to store these bytes. You should store these bytes by using a keyboard of your trainer, or if you are using a simulator, you should observe the numbers in those memory locations when you store them.
2. The program should be written starting at memory location 2030H. This memory location is selected arbitrarily to emphasize that you can write a program beginning at any available memory location.
3. The microprocessor performs arithmetic operations in the ALU, meaning the processor must use accumulator A in performing the subtraction.
4. The data bytes must be copied from memory into the microprocessor registers (until you learn how to perform an arithmetic operation by using the accumulator and a memory register).



WRITING MNEMONICS AND ASSEMBLING HEX CODE

The flowchart for this problem is similar to that shown in Figure 2.3. The steps in writing instructions are as follows:

1. Copy two data bytes into processor registers. By examining Section 2.5, we find two instructions, LDA and STA (instructions 1.6 and 1.7), to copy a byte from memory into A and from A into memory. There are two other instructions (MOV R, M and MOV M, R) that can copy between memory and registers, but those instructions require the concept of memory pointers, which will be discussed in Chapter 7. Now let us take a look at the instruction LDA in the 8085 Instruction Summary: Alphabetical Order, inside the back cover.
   a. **The instruction is: LDA 16-bit.** LDA is the opcode with Hex code 3A. (For a complete description of the instruction LDA, see Appendix F—all instructions are explained in alphabetical order.) As we discussed in Section 2.4 ("Recognizing the Number of Bytes in an Instruction"), this must be a 3-byte instruction and will require three memory locations.
   b. **The operand is a 16-bit address** of the memory location from which we want to copy the byte into A. First we want to copy a byte from memory location 2051H.
   c. **The 3-byte code is 3A 51 20.** See Section 2.3. A 16-bit address is always written in reverse order—low-order byte followed by high-order byte. Our program begins at location 2030H; therefore, these 3 bytes will be stored in locations with Hex addresses 2030, 31, and 32. And the instruction, when executed, will copy the first byte, 49H, into A.
   d. **Copying the second byte into A.** If we copy the second byte, 9FH, from memory location 2052H into A, we will destroy the first byte, 49H. Therefore, the previous byte, 49H, should be stored first in some other register such as B.

2. As discussed in 1d above, the second instruction should be to copy the byte from A into B. The instruction is: MOV Rd, Rs (instruction 1.2 in Section 2.5), copying from one source register Rs to another destination register Rd. The terms Rs and Rd are generic; they represent any register A through L. The instruction is:
   MOV B, A with Hex code 47. Copy A into B—note the reversed order of A and B. This is a 1-byte instruction.
3. Now we can copy the second byte (9FH) from memory location 2052H into A by using the instruction
   LDA 2052H with the 3-byte Hex code 3A 52 20.
4. The next step is to subtract B from A. If we look in Section 2.5 under Arithmetic Instructions, we find three instructions: 2.4, SUB R; 2.5, SUI 8-bit; and 2.6, SUB M.

   The instruction is:
   SUB B with Hex code 90. (Look at the inside of the back cover to find the code.) This instruction subtracts B from A and saves the result in A.
5. The result in A should be stored in memory location 2053H. The instruction is
   STA 2053 with Hex code 32 53 20.
6. Each program must be terminated; otherwise, the processor continues to fetch and execute instructions from the remaining memory registers until it gets lost or caught up in an infinite loop. This step may appear trivial, but it is essential. Therefore, the last instruction is
   HALT with Hex code 76. In instructional trainers, the Restart (RST) instruction is used to pass the control of running programs back to the monitor program of the trainer.
7. Now we need to load the two data bytes 49H and 9FH in memory locations 2051H and 2052H. This step is a manual entry of the data bytes, independent of the program.
8. So far we have completed two steps: writing the program (as shown in Column 1 in Table 2.2) and entering the Hex code in the memory registers of a memory chip (as shown in Column 2 in Table 2.2). If you observe this program in a simulator (see Appendix H), it should appear as shown in Column 2. Using the analogy of putting a radio kit together, we now have a page of instructions. Now we need to find the page, begin to read, understand the instructions, and perform the task until the kit is

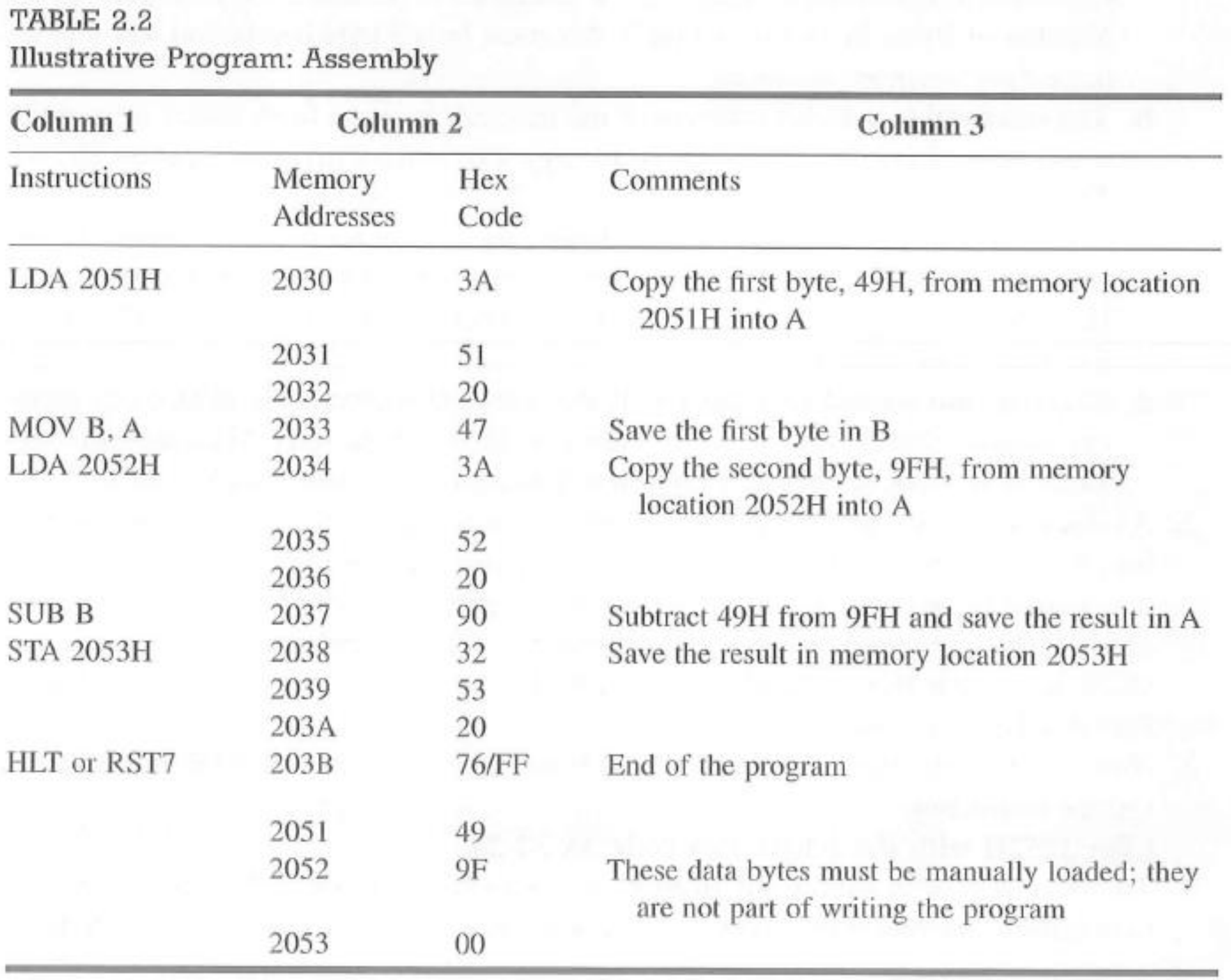


TABLE 2.2
Illustrative Program: Assembly

Column 1	Column 2		Column 3
Instructions	Memory Addresses	Hex Code	Comments
LDA 2051H	2030	3A	Copy the first byte, 49H, from memory location 2051H into A
	2031	51	
	2032	20	
MOV B, A	2033	47	Save the first byte in B
LDA 2052H	2034	3A	Copy the second byte, 9FH, from memory location 2052H into A
	2035	52	
	2036	20	
SUB B	2037	90	Subtract 49H from 9FH and save the result in A
STA 2053H	2038	32	Save the result in memory location 2053H
	2039	53	
	203A	20	
HLT or RST7	203B	76/FF	End of the program
	2051	49	
	2052	9F	These data bytes must be manually loaded; they are not part of writing the program
	2053	00	

HOW UP EXECUTES AN INSTRUCTION?

ASSEMBLY INSTRUCTION:

```
LDA 4200
MVI B,78H
ADD B
STA 4500H
HLT
```

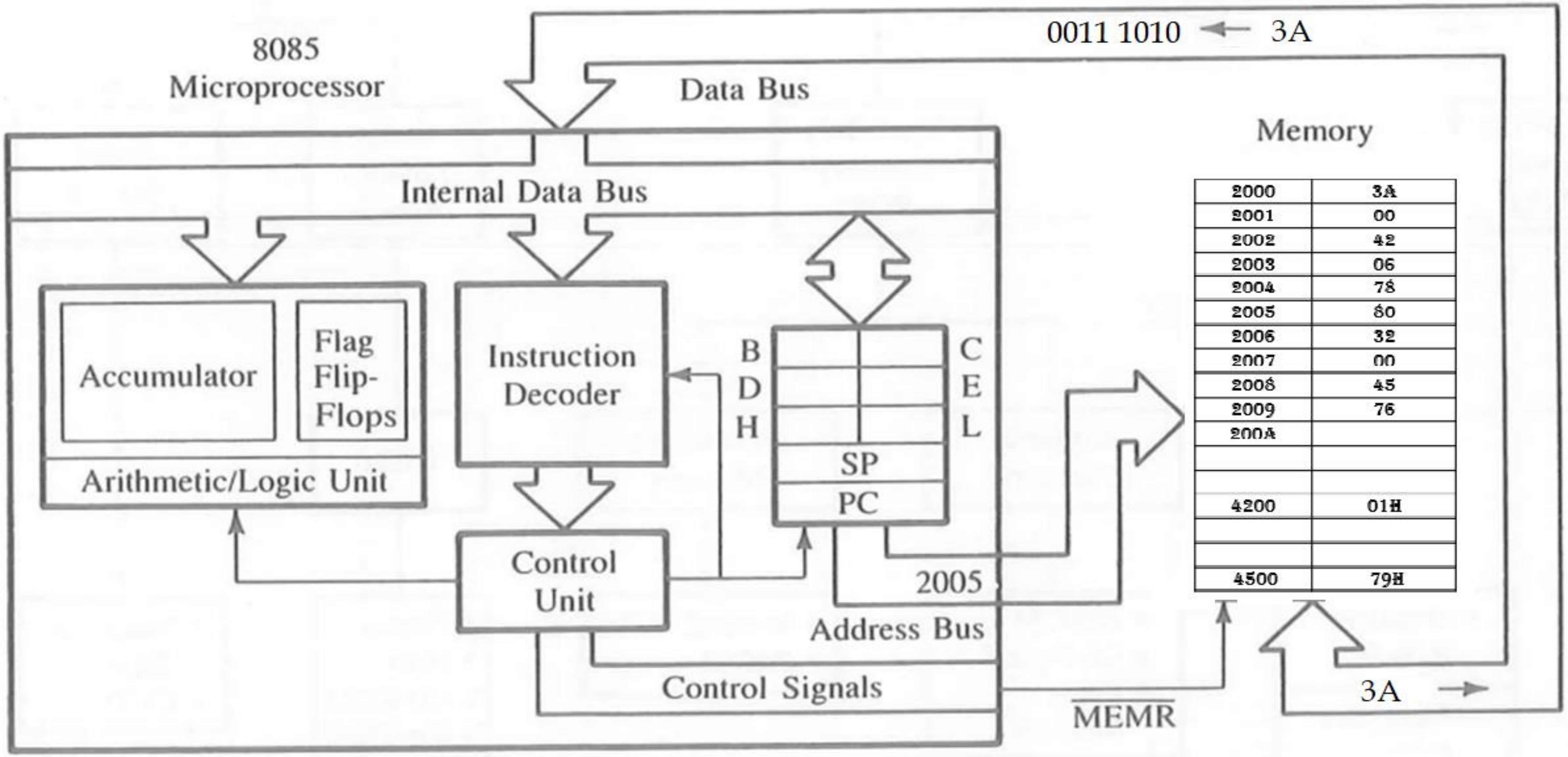
MACHINE CODE:

```
2000    3A          LDA 4200
2001    00
2002    42
2003    06        MVI B,78H
2004    78
2005    80        ADD B
2006    32        STA 4500H
2007    00
2008    45
2009    76        HLT
200A
```

MEMORY

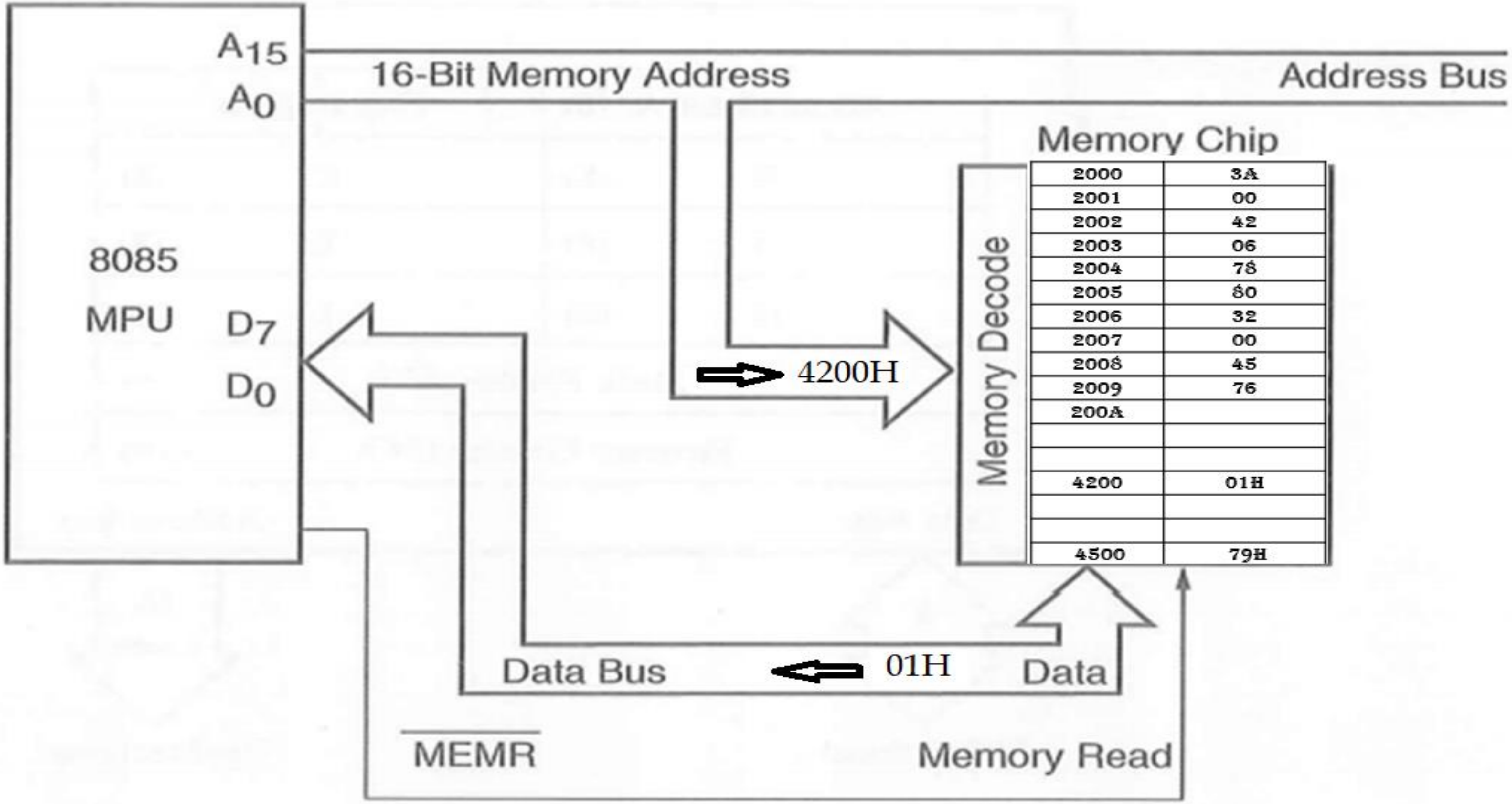
2000	3A
2001	00
2002	42
2003	06
2004	78
2005	80
2006	32
2007	00
2008	45
2009	76
200A	
4200	01H
4500	79H

OPCODE FETCH

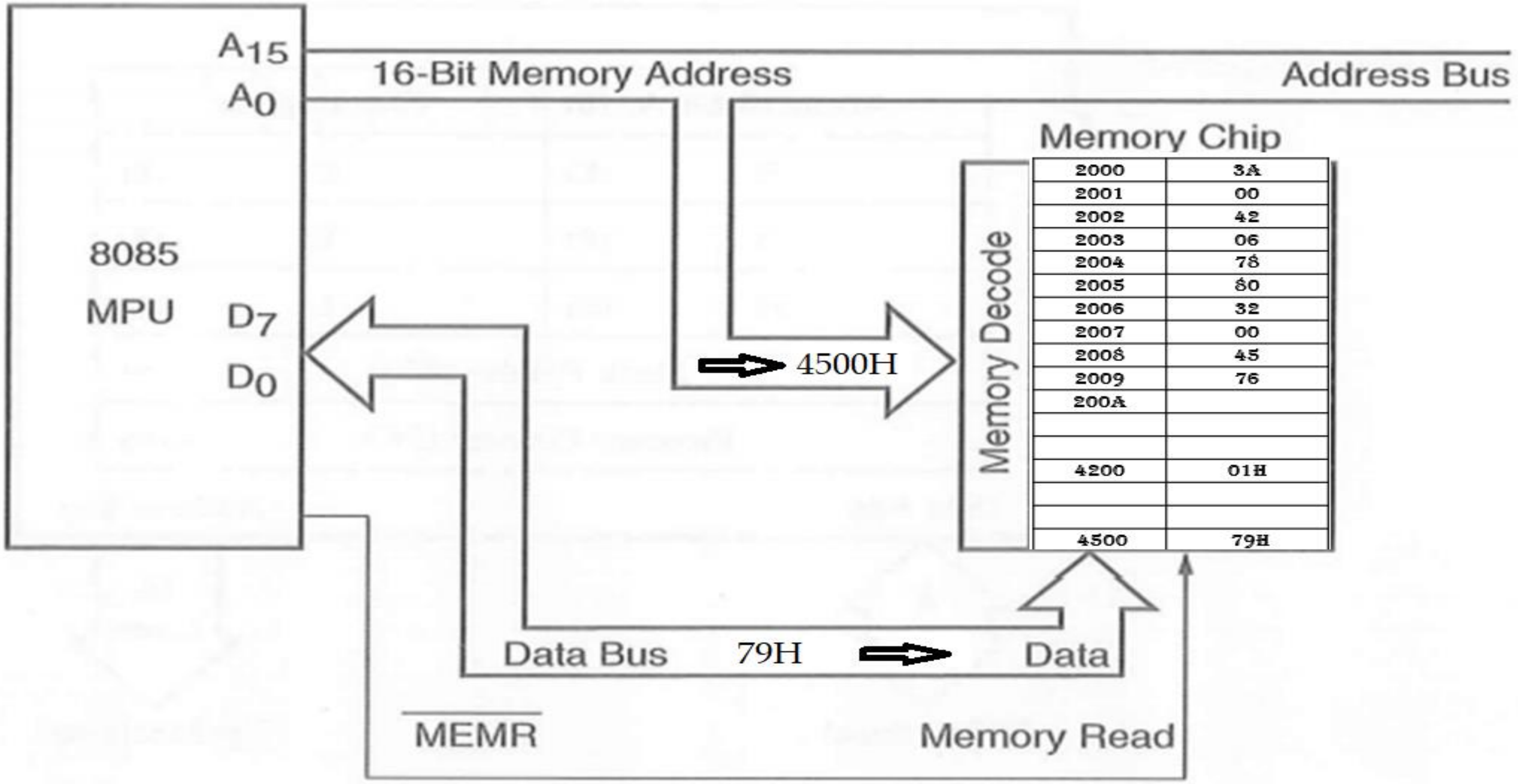


2000	3A
2001	00
2002	42
2003	06
2004	78
2005	80
2006	32
2007	00
2008	45
2009	76
200A	
4200	01H
4500	79H

MEMORY READ


A15
A0
16-Bit Memory Address
Address Bus
8085
MPU
D7
D0
4200H
Memory Chip
Memory Decode
2000 3A
2001 00
2002 42
2003 06
2004 78
2005 80
2006 32
2007 00
2008 45
2009 76
200A
4200 01H
4500 79H
Data Bus
01H
Data
MEMR
Memory Read

MEMORY WRITE



2000	3A
2001	00
2002	42
2003	06
2004	78
2005	80
2006	32
2007	00
2008	45
2009	76
200A	
4200	01H
4500	79H

Thank You

Reference: Chapter 2, Ramesh Gaonkar, 8085 Microprocessor Architecture Programming and applications with the 8085