2020

RANK Improvement WORKBOOK

Answer key and Hint of
Objective & Conventional *Questions*

Electrical Engineering

Microprocessors

MADE EASY Publications

MADE EASY
Publications

1 Intel 8085 and Intel 8086

LEVEL 1 Objective Solutions

1.	(c)
2.	(c)
3.	(b)
4.	(c)
5.	(a)
6.	(a)
7.	(255)

LEVEL 2 Objective Solutions

8.	(a)
9.	(a)
10.	(a)
11.	(d)
12.	(b)
13.	(c)
14.	(c)
15.	(d)
16.	(400)

■■■■

LEVEL 3 Conventional Solutions

Solution: 1

Subroutine call	Interrupt request
1. It occurs only at the place where the programmer has written it.	1. It can occur at any place during the execution of a program.
2. The location of the subroutine is specified by the programmer.	2. The location of a service routine is fixed for a particular kind of interrupt.
3. It does not require any enabling command.	3. It requires the interrupt flip flop to be enabled by the command EI and SIM.
4. It is internally initiated.	4. It is externally initiated.

Yes, it is possible to use a common memory stack for both. But precaution has to be exercised to use POP and PUSH instructions in correct sequence.

Solution: 2

Instruction cycle: It is defined as the time required to complete the execution of an instruction. The 8085 instruction cycle consists of one of six machine cycles or one to six operations.

Machine cycle: It is defined as the time required to complete one operation of accessing memory, I/O, or acknowledging an external request.

Basic machine cycles of 8085:

1. **Memory read machine cycle:** If refers to the process of reading one byte from memory. It requires 3 T states.
2. **Memory write machine cycle:** If refers to the process of writing one byte into the memory. It requires 3 T states.
3. **Opcode fetch machine cycle:** It is the first operation in any instruction. It involves reading the opcode from memory. It requires 4 T states.
4. **I/O read/write machine cycle:** These involve reading from and writing to an input and output port respectively. Each requires 3-T states.
5. **Interrupt acknowledge:** This is the machine cycle to get the address of the interrupt service routine in order to service the interrupt device. It requires 10 T states.

Solution: 3

- Mode of specifying address of operand involved in the operation is known as Addressing mode.
- 8085 microprocessor I.S.A supports 5 addressing mode which are implicit addressing mode, immediate addressing mode, direct addressing mode and indirect addressing mode.

1. **Register addressing mode**

   In register addressing mode, operands involved in operation is/are register.
   This addressing mode is also known as register direct addressing mode.

```
Ex:        MOV A, B
           ADD H
           XRA A
```

2. Implicit Addressing mode
   In implicit addressing mode, operand involved in operation is not specified explicitly in instruction. As such, microprocessor implicitly assumes accumulator is the operand i.e., it is implied that accumulator is the operand.

Ex:	DAA
	CMA
	RAL
	RLC
	RAR
	RRC

3. Immediate addressing mode
   In immediate addressing mode, the operand involved in operation is given in the instruction itself which is either 8-bit number or 16-bit number. i.e., value given in the instruction is operand involved in operation.

Ex:	MVI A, 77 H
	LXI SP, 2600 H
	JMP 2000 H

4. Direct Addressing mode:
   In direct addressing mode, one of the operands involved in operation is memory location and 16-bit address of memory location is given in instruction.

- Value at address is the operand involved in operation

Ex:	STA 2050 H
	LHLD 2345 H
	OUT F7 H
	IN F9 H

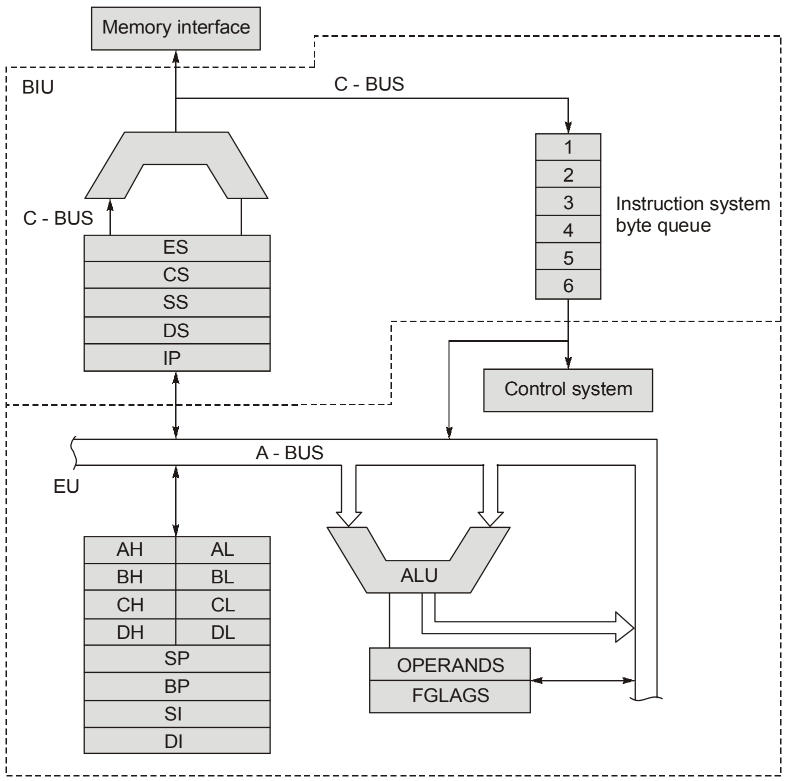
5. Indirect addressing mode
   In indirect addressing mode, one of the operands involved in operation is memory location and 16-bit address of memory location is made available in register pointer (like HL pair, BC pair, DE pair, Stack pointer).
   Also known as register indirect addressing mode

Ex:	MOV A, M
	PUSH PSW
	STAX B
	RET

Solution: 4

The 8086 is a 16 bit microprocessor. It has 16 bit data bus and 20 bit address bus. Words will be stored in two consecutive memory locations. If the first byte of a word is at an even address, the 8086 can read the entire word in one operations. If the first byte of the word is at an odd address, the 8086 will read the first byte in one operation and the second byte in another operation.

Functional block diagram of 8086



The 8086 CPU is divided into two independent functional parts, the bus interface unit or BIU and the execution unit or EU.

Bus interface unit: It handles all data and addresses on the buses for the execution unit such as it sends out addresses, fetches instructions from memory, reads data from parts and memory as well as writes data to ports and memory.

Instruction queue: To increase the execution speed, BIU fetches as many as six instruction bytes ahead of time from memory. These are held for EU in the instruction queue group of registers.

Segment registers: These are four 16-bit segment registers. They are the extra segment (ES) register, the code segment (CS) registers, the data segment (DS) registers and the stack segment (SS) registers. These are used to hold the upped 16 bits of the starting address for each of the segments. The part of a segment starting address stored in a segment register is often called the segment base.

Instruction pointer: It holds the 16 bit address of the next code byte within this code segment.

Execution unit: The EU tells the BIU where to fetch instructions or data from, decode instructions and executes instructions. The functional parts of the EU are control circuitry or system, instruction decoder and ALU.

Flag register: A 16-bit flag register is a flip flop which indicates for some conditions produced by the execution of an instructions. It has a flags-carry, auxiliary, parity, zero, sign, overflow, trap, interrupt and direction flags.

General purpose registers: The EU has 8 general purpose registers labelled AH, AL, BH, BL, CH, CL, DH and DL. These registers can be used individually for temporary storage of 8 bit data. The AL registers is also called the accumulator. Contain pairs of these general purpose registers can be used together to store 16-bit data. The valid register pairs are Ah and AL, BH and BL, CH and CL and DH and DL. The pairs are referred to as AX, BX, CX and DX.

1. **AX register:** For 16 bit operations, AX is called the accumulator register that stores operands for arithmetic operations.
2. **BX register:** This register is mainly used as a base register. It holds the starting base location of a memory region within a data segment.
3. **CX register:** It is defined as a counter. It is primarily used in loop instructions to store loop counter.
4. **DX register:** DX register is used to contain I/O port addresses for I/O instruction.
5. **Stack pointer register:** It contains 16 bit offset from the start of the segment to the memory location where a word was most recently stored on the stack, called as the top of the stack.
   Other registers like SI, BP and DI are mainly used for temporary storage of 16 bit data just like a general purpose register.

Memory organisation: The size of address bus is 20 and is able to address 1 MB of physical memory, but all this memory is not active at one time. It is divided into 16 segments and only is of these segments are active at a time. These are code segment, stack segment, data segment and extra segment.

Solution : 5

The 8085 instructions can be classified into the following five functional categories: data transfer (copy) operations, arithmetic operations, logical operations, branching operations and machine-control operations.

Data Transfer (Copy) Operation:

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

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 sub-tracted 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.

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 Os are replaced by Is and all Is are replaced by Os.

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.

Solution : 6

The microprocessor is a sequential machine. As soon as a microprocessor-based system is turned on, it begins the execution of the code in memory. The execution continues in a sequence, one code after another (one memory location after another) at the speed of its clock until the system is turned off (or the clock stops). If an unconditional loop is set up in a program, the execution will continue until the system is either reset or turned off. Now a puzzling question is: How does the microprocessor differentiate between a code and data when both are binary numbers? The answer lies in the fact that the microprocessor interprets the first byte it fetches as an opcode. When the 8085 is reset, its program counter is cleared to 0000H and it fetches the first code from the location 00O0H. In the example of the previous section, we tell the processor that our program begins at location 2000H. The first code it fetches is 3EH. When it decodes that code, it knows that it is a two-byte instruction. Therefore, it assumes that the second code, 32H, is a data byte. If we forget to enter 32H and enter the next code, 06H, instead, the 8085 will load 06H in the accumulator, interpret the next code, 48H, as an opcode, and continue the execution in sequence. As a consequence, we may encounter a totally unexpected result.

Solution: 7

(i) Data Transfer Instructions :

These type of instructions are used to transfer data from source operand to destination operand. All the store, move, load, exchange, input and output instructions belong to this category. E.g. MOV, EXCHG, etc.

(ii) Arithmetic and Logical Instructions :
All the instructions performing arithmetic, logical, increment, decrement, compare and scan instructions belong to this category. E.g. ADD, SUB, MUL, etc.

(iii) Branch Instructions :
These instructions transfer control of execution to the specified address. All the call, jump, interrupt and return instructions belong to this category. E.g. JMP, CALL, INTO etc.

(iv) Loop Instructions :
If these instructions have REP prefix with CX used as count register, they can be used to implement unconditional and conditional loops. These are useful to implemented different loop structures. E.g. LOOP, LOOPNZ, LOOPZ, etc.

(v) Machine Control Instructions :
These instructions control the machine status. E.g. NOT, HLT, WAIT, etc.

(vi) Flag Manipulation Instructions :
All the instructions which directly affect the flag register, come under this group of instructions. E.g. CLD, STD, etc.

(vii) Shift and Rotate Instructions :
These instructions involve the bitwise shifting or rotation in either direction with or without a count in CX.
e.g. SCL, ROR, etc.

(viii) String Instructions :
These instructions involve various string manipulation operations like load, store, scan, compare, etc. E.g. SCAS, STOS, LODS, etc.

■■■■

2 Programming of Microprocessor

LEVEL 1 Objective Solutions

1.	(b)
2.	(b)
3.	(c)
4.	(c)
5.	(14)
6.	(b)
7.	(a)
8.	(b)
9.	(a)
10.	(a)

LEVEL 2 Objective Solutions

11.	(c)
12.	(c)
13.	(48)
14.	(c)
15.	(b)
16.	(d)
17.	(a)
18.	(70)
19.	(22)
20.	(1)

■■■■

LEVEL 3 Conventional Solutions

Solution: 1

```
START:      IN     01 H;
            XRI    FFH;       // to test if input is FFH
            JZ     FIRST;     // jump if input is FFH as result of XR7 will be all zeros
            MVI    A, 00 H;
            OUT    05 H;      // new coin instruction at port 5
            JMP    START;
FIRST:      IN     02 H;      // to check if requirement is for tea or coffee
            MOV    B, A;
            XRI    04 H;
            JZ     TEA;
            MOV    A, B;
            XRI    40 H;
            JZ     COFFEE;
TEA:        MVI    A, 04 H;
            OUT    03 H;
            JMP    CLOSE;
COFFEE:     MVI    A, 04H;
            OUT    03 H;
            JMP    CLOSE;
CLOSE:      IN     04 H;      // to check for the command to close the value
            XRI    00 H;
            JNZ    CLOSE;
            MVI    A, FFH;
            OUT    03 H;
            JMP    START;
            HLT
```

Solution: 2

```
        LXI H, 7000H;
        LXI D, A00FH;
        MVI B, 10H;
START;
        MOV   A, M;
        STAX  D;
        INX   H;
        DCX   D;
        DCR   B;
        JNZ   START;
        HLT
```

Solution: 3

Main Program:

```
            LXI D, 2050H   ;  Point index to readings
            LXI H, 2090H   ;  Point index to maximum limits
            MVI C, 05H     ;  Set up C as a counter with a count value of 5
NEXT:       CALL SBTRAC    ;  Call the subroutine to perform 16-bit subtraction to determine the
                              difference between a reading and its maximum limit
            INX D          ;  Increment the contents of DE pair to point the location of next reading
            INX H          ;  Increment the contents of HL pair to point the location of maximum limit
                              corresponding to the next reading
            DCR C          ;  Decrement the count value after completion of the process corresponding
                              to each reading
            JNZ NEXT       ;  Jump to the location labeled as "NEXT" till the count value of the counter
                              becomes zero, to start the process corresponding to the next reading
            HLT            ;  Halt the execution
```

Subroutine:

```
SBTRAC:     MOV A, M       ;  Load the accumulator with lower byte of the maximum limit corresponding
                              to the reading
            XCHG           ;  Exchange the contents of HL and DE pairs
            SUB M          ;  Subtract lower byte of the reading from the lower byte of the
                              corresponding maximum limit
            MOV M, A       ;  Store the lower byte of the difference in the location corresponding to
                              the lower byte of the reading
            XCHG           ;  Exchange the contents of HL and DE pairs
            MOV A, M       ;  Load the accumulator with higher byte of the maximum limit
                              corresponding to the reading
            XCHG           ;  Exchange the contents of HL and DE pairs
            SBB M          ;  Subtract (with borrow) higher byte of the reading from the higher byte of
                              the corresponding maximum limit
            CC INDCTR      ;  Call the indicator subroutine if the value of reading is higher than the
                              corresponding maximum limit
            MOV M, A       ;  Store the higher byte of the difference in the location corresponding to
                              the higher byte of the reading
            XCHG           ;  Exchange the contents of HL and DE pairs
            RET            ;  Return to the Main Program
```

Solution: 4

- The total delay produced by the given subroutine program can be calculated by determining the total number of T-states required to execute the program.
- The total number of T-states required to execute the program can be determined by analyzing the given program as shown in the following table:

	Instruction	Number of times executed	Number of T-states for one time execution
Delay :	MVI B, 02H	1	7
LOOP2 :	MVI C, FFH	$(1 \times 2) = 2$	7
LOOP1 :	DCR C	$(255 \times 2) = 510$	4
	JNZ LOOP1	$(254 \times 2) = 508 \Rightarrow$ true $(1 \times 2) = 2 \Rightarrow$ false	$10 \Rightarrow$ true $7 \Rightarrow$ false
	DCR B	2	4
	JNZ LOOP2	$1 \Rightarrow$ true $1 \Rightarrow$ false	$10 \Rightarrow$ true $7 \Rightarrow$ false
	RET	1	10

- The total delay produced by the program in terms of T-states can be given by,

$$\begin{aligned}\text{Delay} &= (1 \times 7T) + (2 \times 7T) + (510 \times 4T) + (508 \times 10T) + (2 \times 7T) + (2 \times 4T) + (10T + 7T) + (10T)\\ &= 7T + 14T + 2040T + 5080T + 14T + 8T + 17T + 10T\\ &= 7190T\end{aligned}$$

- The time delay corresponds to one T-state is,

$$T = \frac{1}{f_{clk}} = \frac{1}{2}\,\mu s = 0.5\,\mu s$$

$\because$ given that, $f_{clk} = 2$ MHz

- So, the total delay produced by the program is,

$$\text{Delay} = \frac{7190}{2}\,\mu s = 3595\,\mu s \simeq 3.6\text{ ms}$$

Solution: 5

```
          ORG 0000 H
          JMP MAIN
          ORG 0100 H
MAIN :    LDA 4000 H
          MVI C, 08 H
LOOP :    RLC
          JNC SKIP
          INR B
SKIP :    DCR C
          JNZ LOOP
          MOV A, B
          RAR
          JC EVEN
          MVI A, DD H
          STA 4000 H
          HLT
EVEN :    MVI A, EE H
          STA 4000 H
          HLT
```

■■■■

3 Memory Interfacing and Peripheral Devices

LEVEL 1 Objective Solutions

1.	(c)
2.	(b)
3.	(65792)
4.	(d)
5.	(b)
6.	(a)
7.	(d)
8.	(b)

LEVEL 2 Objective Solutions

9.	(c)
10.	(c)
11.	(d)
12.	(c)
13.	(d)
14.	(c)
15.	(d)
16.	(d)
17.	(c)

■■■■

LEVEL 3 Conventional Solutions

Solution: 1

The address is given by

$$\underbrace{A_{15}\,A_{14}\,A_{13}\,A_{12}}\;\underbrace{A_{11}\,A_{10}\,A_{9}\,A_{8}}\;\underbrace{A_{7}\,A_{6}\,A_{5}\,A_{4}}\;\underbrace{A_{3}\,A_{2}\,A_{1}\,A_{0}}$$

To enable the clip select

$$A_{15} = 0$$
$$A_{14} = 0$$
$$A_{13} = 1$$
$$A_{12} = 0$$
$$A_{11} = 0$$

So, the address is given by

$$\underbrace{0010}\;\underbrace{0\,A_{10}\,A_{9}\,A_{8}}\;\underbrace{A_{7}\,A_{6}\,A_{5}\,A_{4}}\;\underbrace{A_{3}\,A_{2}\,A_{1}\,A_{0}}$$

So, it varies in the range

$$\underbrace{0010}\;\underbrace{0000}\;\underbrace{0000}\;\underbrace{0000}$$

$$= 2000\text{ H}$$

to

$$\underbrace{0010}\;\underbrace{0000}\;\underbrace{1111}\;\underbrace{1111}$$

$$= 20\text{ FFH}$$

Hence proved.

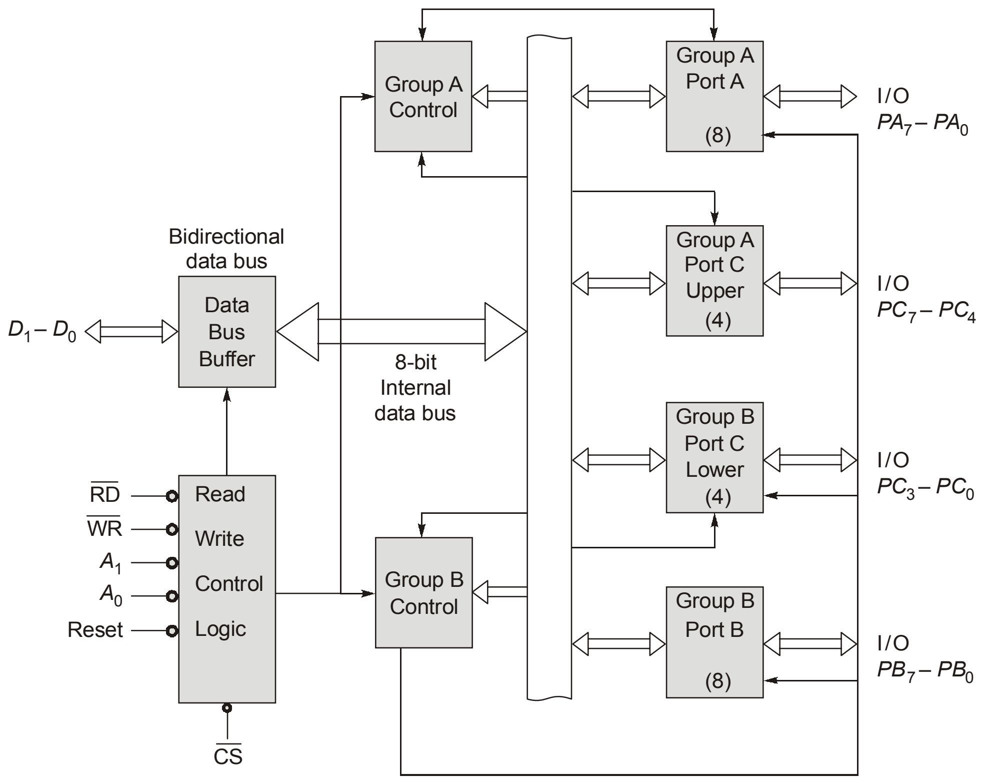
Solution: 2

The function of direct memory access is to allow in I/O device to send or receive data directly to or from the main memory, bypassing the CPU to speed up memory operations. The process is managed by a chip known as DMA controller.

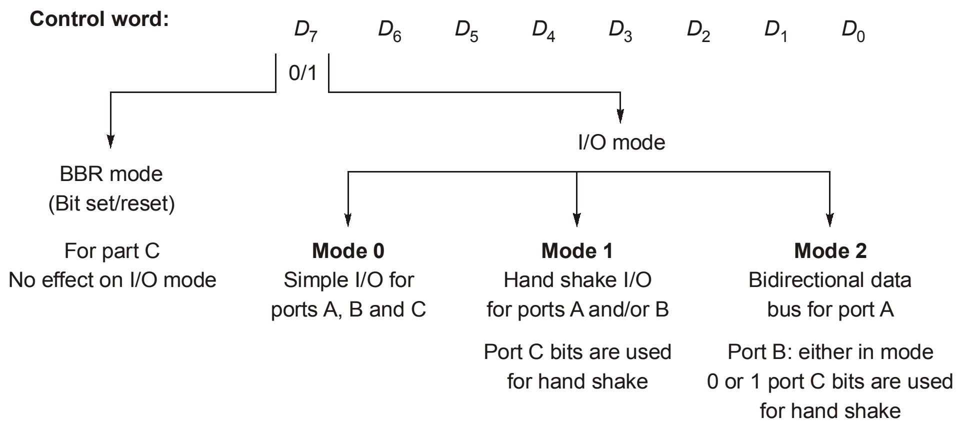
During DMA operations, the DMA controller sends logic high on HOLD input of the microprocessor. After receiving this signal, the MPU relinquishes the buses in the following machine cycle. All buses are tri-stated and the HLDA signal is sent out. The MPU regains the control of the buses after HOLD goes low. Data transfer modes of DMA controller are as follows:

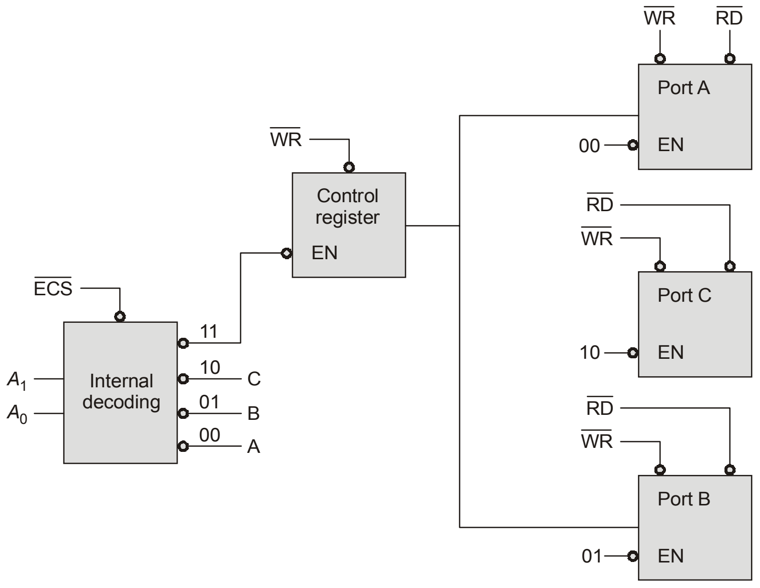
1. **Burst on block transfer DMA:** An entire block of data is transferred in one contiguous sequence. It transfers all bytes of data in the data block before releasing control of the system buses back to the CPU, but renders the CPU inactive for relatively longer periods of time.
2. **Cycle stealing mode:** The DMA controller gives back the control of buses back to the MPU after transferring 1 byte and then requests the control of buses again. It follows this process until the entire data is transferred. The DMA controller essentially interleaves instruction and data transfers.
3. **Transparent mode:** The DMA transfers the data only when the CPU is performing operations that do not use the system buses.

Solution: 3



Block schematic of 8255


Control word:
D_7 D_6 D_5 D_4 D_3 D_2 D_1 D_0
0/1
BBR mode (Bit set/reset)
For part C No effect on I/O mode
I/O mode
Mode 0 Simple I/O for ports A, B and C
Mode 1 Hand shake I/O for ports A and/or B
Port C bits are used for hand shake
Mode 2 Bidirectional data bus for port A
Port B: either in mode 0 or 1 port C bits are used for hand shake

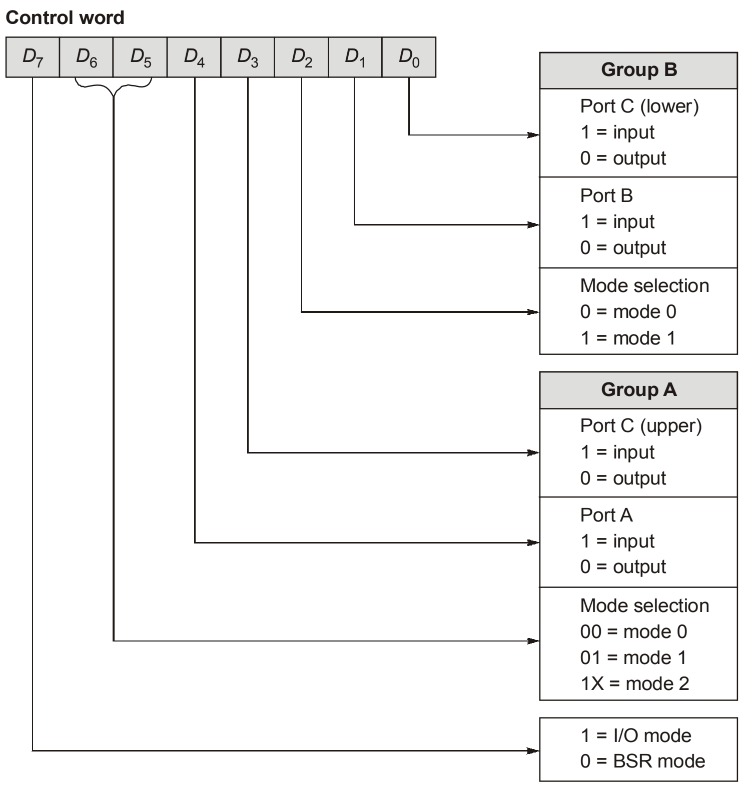


Expanded version of the control logic and I/O ports

Port addresses: This is a memory mapped I/O. When the address line A_{15} is high, the chip select line is enabled. Assuming all demit care lines are at logic 0, the port addresses are as follows:

$$\begin{aligned}\text{Port A} &= 8000\text{ H }(A_1 = 0;\ A_0 = 0)\\ \text{Port B} &= 8001\text{ H }(A_1 = 0;\ A_0 = 0)\\ \text{Port C} &= 8002\text{ H }(A_1 = 0;\ A_0 = 0)\\ \text{Control register} &= 8003\text{ H }(A_1 = 1;\ A_0 = 1)\end{aligned}$$

Control word: 10000011



Solution : 4

Input/output devices are the means through which the MPU communicates with "the outside world." The MPU accepts binary data as input from devices such as keyboards and A/D converters and sends data to output devices such as LEDs or printers. There are two different methods by which I/O devices can be identified one uses an 8-bit address and the other uses a 16-bit address.

I/Os with 8-Bit Addresses (Peripheral-Mapped I/O):

In this type of I/O, the MPU uses eight address lines to identify an input or an output device; this is known as peripheral-mapped I/O. This is an 8-bit numbering system for I/Os used in conjunction with Input and Output instructions. This is also known as I/O space, separate from memory space, which is a 16-bit numbering system. The eight address lines can have 256 (2^8 combinations) addresses; thus, the MPU can identify 256 input devices and 256 output devices with addresses ranging from 00H to FFH. The input and output devices are differentiated by the control signals; the MPU uses the I/O Read control signal for input devices and the I/O Write control signal for output devices. The entire range of I/O addresses from 00 to FF is known as an I/O map, and individual addresses are referred to as I/O device addresses or I/O port numbers.

The steps in communicating with an I/O device are:

1. The MPU places an 8-bit address on the address bus, which is decoded by external decode logic.
2. The MPU sends a control signal (I/O Read or I/O Write) and enables the I/O device.
3. Data are transferred using the data bus.

I/Os with 16-Bit Addresses (Memory-Mapped I/O):

In this type of I/O, the MPU uses 16 address lines to identify an I/O device; an I/O is connected as if it is a memory register. This is known as memory-mapped I/O. The MPU uses the same control signal (Memory Read or Memory Write) and instructions as those of memory. In some microprocessors, such as the Motorola 6800, all I/Os have 16-bit addresses; I/Os and memory share the same memory map (64K). In memory-mapped I/O, the MPU follows the same steps as if it is accessing a memory register.

Solution: 5

$$\text{Size of RAM chips available} = 1024 \times 1 = 2^{10} \times 1$$

$$\text{Size of RAM chips required} = 16 \text{ k bytes} = 16 \text{ k} \times 8 = 2^4 \times 2^{10} \times 2^3$$

$$\text{Number of chips required} = \frac{2^4 \times 2^{10} \times 2^3}{2^{10} \times 1} = 2^4 \times 2^3 = 128 \text{ chips}$$

We have a memory chip with size $2^{10} \times 1$ and we require to design 16 k bytes.

To design 1 k byte we have to arrange eight $2^{10} \times 1$ chips as given below in figure (a).

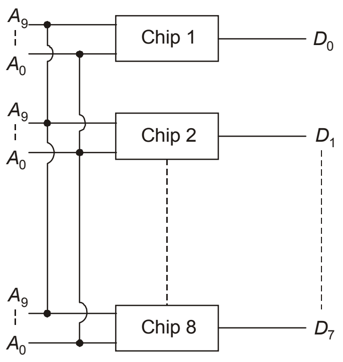


Figure (a)

Now, we can form a memory of size 16 k bytes by connecting 16 modules each of size 1 k bytes using a 4 × 16 decoder as given below in figure (b).

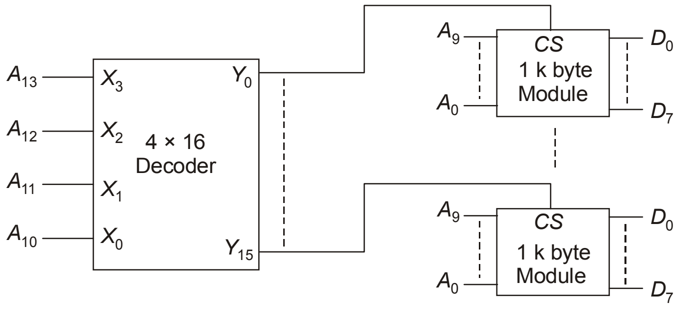


Figure (b)

Solution: 6

The 8237 is in the idle cycle if there is no pending request or the 8237 is waiting for a request from one of the DMA channels. Once a channel requests a DMA service, the 8237 sends the HOLD request to the CPU using the HRQ pin. If the CPU acknowledges the hold request on HLDA, the 8237 enters an active cycle. In the active cycle, the actual data transfer takes place in one of the following transfer modes, as is programmed.

Single Transfer Mode : In this mode, the device transfers only one byte per request. The word count is decremented and the address is decremented or incremented (depending on programming) after each such transfer. The Terminal Count (TC) state is reached, when the count becomes zero. For each transfer, the DREQ must be active until the DACK is activated, in order to get recognized. After TC, the bus will be relinquished for the CPU. For a new DREQ to 8237, it will again activate the HRQ signal to the CPU and the HLDA signal from the CPU will push the 8237 again into the single transfer mode. This mode is also called as “cycle stealing”.

Block Transfer Mode : In this mode, the 8237 is activated by DREQ to continue the transfer until a TC is reached, i.e. a block of data is transferred. The transfer cycle may be terminated due to $\overline{\text{EOP}}$ (either internal or external) which forces Terminal Count (TC). The DREQ needs to be activated only till the DACK signal is activated by the DMA controller. Auto-initialization may be programmed in this mode.

Demand Transfer Mode : In this mode, the device continues transfers until a TC is reached or an external $\overline{\text{EOP}}$ is detected or the DREQ signal goes inactive. Thus a transfer may exhaust the capacity of data transfer of an I/O device. After the I/O device is able to catch up, the service may be re-established activating the DREQ signal again. Only the $\overline{\text{EOP}}$ generated by TC or external $\overline{\text{EOP}}$ can cause the auto-initialization, and only if it is programmed for.

Cascade Mode : In this mode, more than one 8237 can be connected together to provide more than four DMA channels. The HRQ and HLDA signals from additional 8237s are connected with DREQ and DACK pins of a channel of the host 8237 respectively. The priorities of the DMA requests may be preserved at each level. The first device is only used for prioritizing the additional devices (slave 8237s), and it does not generate any address or control signal of its own. The host 8237 responds to DREQ generated by slaves and generates the DACK and the HRQ signals to co-ordinate all the slaves. All other outputs of the host 8237 are disabled.

Memory to Memory Transfer Mode : To perform the transfer of a block of data from one set of memory address to another one, this transfer mode is used. Programming the corresponding mode bit in the command word, sets the channel 0 and 1 to operate as source and destination channels, respectively. The transfer is initialized by setting the $DREQ_0$ using software commands. The 8237 sends HRQ (Hold Request) signal to the CPU as usual and when the HLDA signal is activated by the CPU, the device starts operating in block transfer mode to read the data from memory. The channel 0 current address register acts as a source pointer. The byte read from the memory is stored in an internal temporary register of 8237. The channel 1 current address register acts as a destination pointer to write the data from the temporary register to the destination memory location. The pointers are automatically incremented or decremented, depending upon the programming. The channel 1 word count register is used as a counter and is decremented after each transfer. When it reaches zero, a TC is generated, causing $\overline{EOP}$ to terminate the service.

The 8237 also responds to external $\overline{EOP}$ signals to terminate the service. This feature may be used to scan a block of data for a byte. When a match is found the process may be terminated using the external $\overline{EOP}$.

Under all these transfer modes, the 8237 carries out three basic transfers namely, write transfer, read transfer and verify transfer. In write transfer, the 8237 reads from an I/O device and writes to memory under the control of $\overline{IOR}$ and $\overline{MEMW}$ signals. In read transfer, the 8237 reads from memory and writes to an I/O device by activating the $\overline{MEMR}$ and $\overline{IOW}$ signals. In verify transfer, the 8237 works in the same way as the read or write transfer but does not generate any control signal.

■■■■