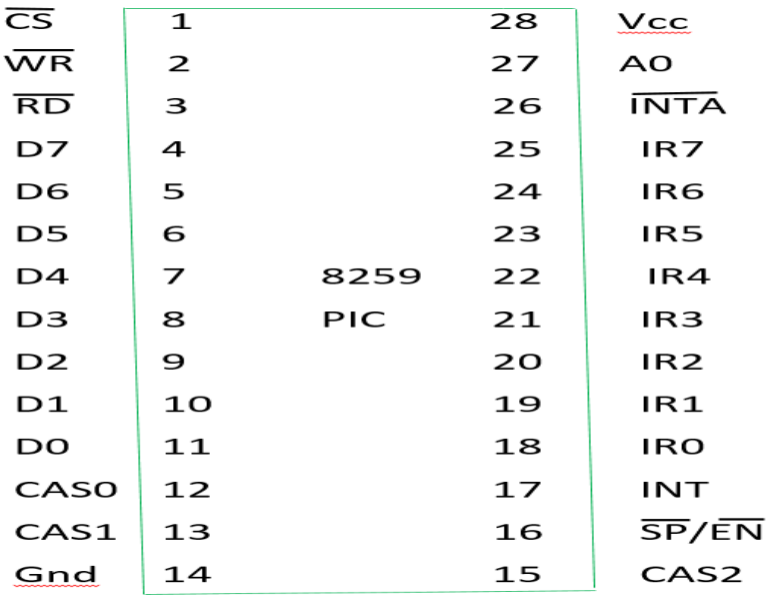
Programmable Interrupt Controller

8259 microprocessor is defined as **Programmable Interrupt Controller (PIC)** microprocessor. There are 5 hardware interrupts and 2 hardware interrupts in 8085 and 8086 respectively. But by connecting 8259 with CPU, we can increase the interrupt handling capability. 8259 combines the multi interrupt input sources into a single interrupt output. Interfacing of single PIC provides 8 interrupts inputs from IR0-IR7.
For example, Interfacing of 8085 and 8259 increases the interrupt handling capability of 8085 microprocessor from 5 to 8 interrupt levels.

Features of 8259 PIC microprocessor –

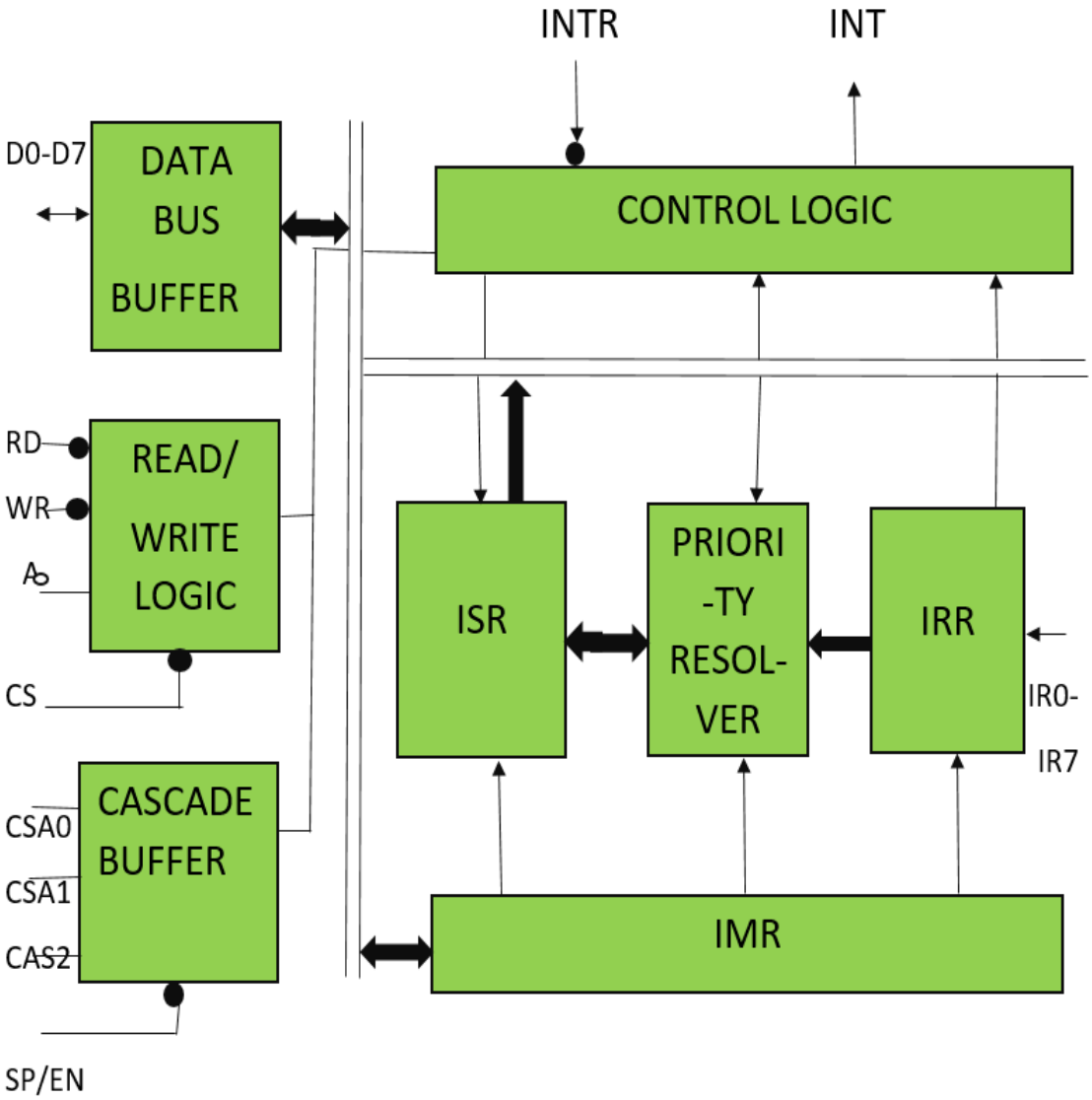
1. Intel 8259 is designed for Intel 8085 and Intel 8086 microprocessor.
2. It can be programmed either in level triggered or in edge triggered interrupt level.
3. We can masked individual bits of interrupt request register.
4. We can increase interrupt handling capability upto 64 interrupt level by cascading further 8259 PIC.
5. Clock cycle is not required.

Pin Diagram of 8259 –



We can see through above diagram that there are total 28 pins in 8259 PIC microprocessor where Vcc :5V Power supply and Gnd: ground. Other pins use are explained below.

Block Diagram of 8259 PIC microprocessor –



The Block Diagram consists of 8 blocks which are – Data Bus Buffer, Read/Write Logic, Cascade Buffer Comparator, Control Logic, Priority Resolver and 3 registers- ISR, IRR, IMR.

1. **Data bus buffer –**
   This Block is used as a mediator between 8259 and 8085/8086 microprocessor by acting as a buffer. It takes the control word from the 8085 (let say) microprocessor and transfer it to the control logic of 8259 microprocessor. Also, after selection of Interrupt by 8259 microprocessor, it transfer the opcode of the selected Interrupt and address of the Interrupt service sub routine to the other connected microprocessor. The data bus buffer consists of 8 bits represented as D0-D7 in the block diagram. Thus, shows that a maximum of 8 bits data can be transferred at a time.
2. **Read/Write logic –**
   This block works only when the value of pin CS is low (as this pin is active low). This block is responsible for the flow of data depending upon the inputs of RD and WR. These two pins are active low pins used for read and write operations.

3. **Control logic –**
   It is the centre of the microprocessor and controls the functioning of every block. It has pin INTR which is connected with other microprocessor for taking interrupt request and pin INT for giving the output. If 8259 is enabled, and the other microprocessor Interrupt flag is high then this causes the value of the output INT pin high and in this way 8259 responds to the request made by other microprocessor.
4. **Interrupt request register (IRR) –**
   It stores all the interrupt level which are requesting for Interrupt services.
5. **Interrupt service register (ISR) –**
   It stores the interrupt level which are currently being executed.
6. **Interrupt mask register (IMR) –**
   It stores the interrupt level which have to be masked by storing the masking bits of the interrupt level.
7. **Priority resolver –**
   It examines all the three registers and set the priority of interrupts and according to the priority of the interrupts, interrupt with highest priority is set in ISR register. Also, it reset the interrupt level which is already been serviced in IRR.
8. **Cascade buffer –**
   To increase the Interrupt handling capability, we can further cascade more number of pins by using cascade buffer. So, during increment of interrupt capability, CSA lines are used to control multiple interrupt structure.

SP/EN (Slave program/Enable buffer) pin is when set to high, works in master mode else in slave mode. In Non Buffered mode, SP/EN pin is used to specify whether 8259 work as master or slave and in Buffered mode, SP/EN pin is used as an output to enable data bus.

DMA CONTROLLER

DMA stands for Direct Memory Access. It is designed by Intel to transfer data at the fastest rate. It allows the device to transfer the data directly to/from memory without any interference of the CPU.

Using a DMA controller, the device requests the CPU to hold its data, address and control bus, so the device is free to transfer data directly to/from the memory. The DMA data transfer is initiated only after receiving HLDA signal from the CPU.

How DMA Operations are Performed?

Following is the sequence of operations performed by a DMA −

- Initially, when any device has to send data between the device and the memory, the device has to send DMA request (DRQ) to DMA controller.

- The DMA controller sends Hold request (HRQ) to the CPU and waits for the CPU to assert the HLDA.
- Then the microprocessor tri-states all the data bus, address bus, and control bus. The CPU leaves the control over bus and acknowledges the HOLD request through HLDA signal.
- Now the CPU is in HOLD state and the DMA controller has to manage the operations over buses between the CPU, memory, and I/O devices.

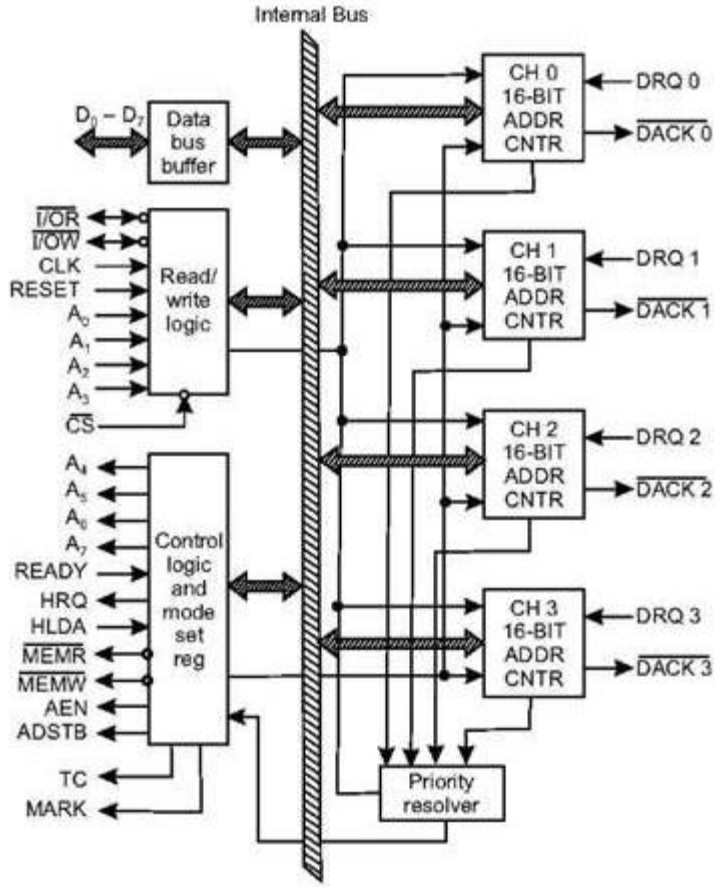
Features of 8257

Here is a list of some of the prominent features of 8257 −

- It has four channels which can be used over four I/O devices.
- Each channel has 16-bit address and 14-bit counter.
- Each channel can transfer data up to 64kb.
- Each channel can be programmed independently.
- Each channel can perform read transfer, write transfer and verify transfer operations.
- It generates MARK signal to the peripheral device that 128 bytes have been transferred.
- It requires a single phase clock.
- Its frequency ranges from 250Hz to 3MHz.
- It operates in 2 modes, i.e., **Master mode** and **Slave mode**.

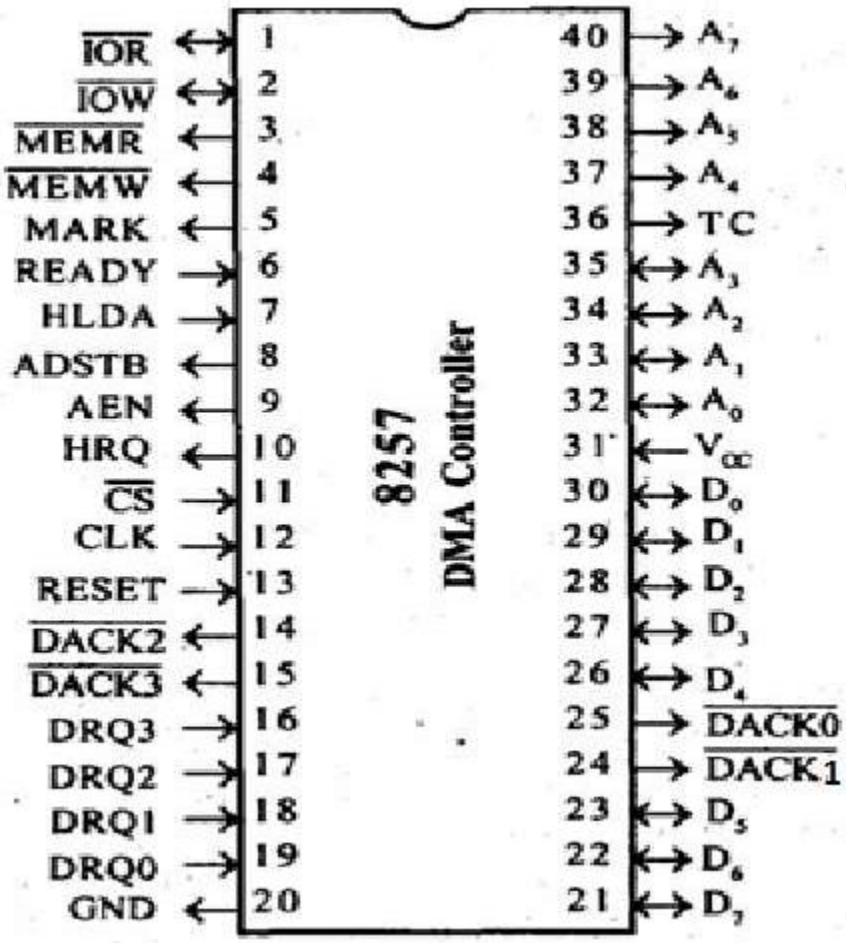
8257 Architecture

The following image shows the architecture of 8257 −



8257 Pin Description

The following image shows the pin diagram of a 8257 DMA controller −



DRQ_0−DRQ**3**

These are the four individual channel DMA request inputs, which are used by the peripheral devices for using DMA services. When the fixed priority mode is selected, then DRQ_0 has the highest priority and DRQ_3 has the lowest priority among them.

$DACK_o$ − $DACK_3$

These are the active-low DMA acknowledge lines, which updates the requesting peripheral about the status of their request by the CPU. These lines can also act as strobe lines for the requesting devices.

D_o − D_7

These are bidirectional, data lines which are used to interface the system bus with the internal data bus of DMA controller. In the Slave mode, it carries command words to 8257 and status word from 8257. In the master mode, these lines are used to send higher byte of the generated address to the latch. This address is further latched using ADSTB signal.

IOR

It is an active-low bidirectional tri-state input line, which is used by the CPU to read internal registers of 8257 in the Slave mode. In the master mode, it is used to read data from the peripheral devices during a memory write cycle.

IOW

It is an active low bi-direction tri-state line, which is used to load the contents of the data bus to the 8-bit mode register or upper/lower byte of a 16-bit DMA address register or terminal count register. In the master mode, it is used to load the data to the peripheral devices during DMA memory read cycle.

CLK

It is a clock frequency signal which is required for the internal operation of 8257.

RESET

This signal is used to RESET the DMA controller by disabling all the DMA channels.

A_0 - A_3

These are the four least significant address lines. In the slave mode, they act as an input, which selects one of the registers to be read or written. In the master mode, they are the four least significant memory address output lines generated by 8257.

CS

It is an active-low chip select line. In the Slave mode, it enables the read/write operations to/from 8257. In the master mode, it disables the read/write operations to/from 8257.

A_4 - A_7

These are the higher nibble of the lower byte address generated by DMA in the master mode.

READY

It is an active-high asynchronous input signal, which makes DMA ready by inserting wait states.

HRQ

This signal is used to receive the hold request signal from the output device. In the slave mode, it is connected with a DRQ input line 8257. In Master mode, it is connected with HOLD input of the CPU.

HLDA

It is the hold acknowledgement signal which indicates the DMA controller that the bus has been granted to the requesting peripheral by the CPU when it is set to 1.

MEMR

It is the low memory read signal, which is used to read the data from the addressed memory locations during DMA read cycles.

MEMW

It is the active-low three state signal which is used to write the data to the addressed memory location during DMA write operation.

ADST

This signal is used to convert the higher byte of the memory address generated by the DMA controller into the latches.

AEN

This signal is used to disable the address bus/data bus.

TC

It stands for ‘Terminal Count’, which indicates the present DMA cycle to the present peripheral devices.

MARK

The mark will be activated after each 128 cycles or integral multiples of it from the beginning. It indicates the current DMA cycle is the 128th cycle since the previous MARK output to the selected peripheral device.

V_{cc}

It is the power signal which is required for the operation of the circuit.

LINEAR SEARCH PROGRAM IN ASSEMBLY LANGUAGE

In this program we will see how to search an element in a block of bytes using 8085.

Problem Statement

Write 8085 Assembly language program to search a key value in a block of data using linear search technique.

Discussion

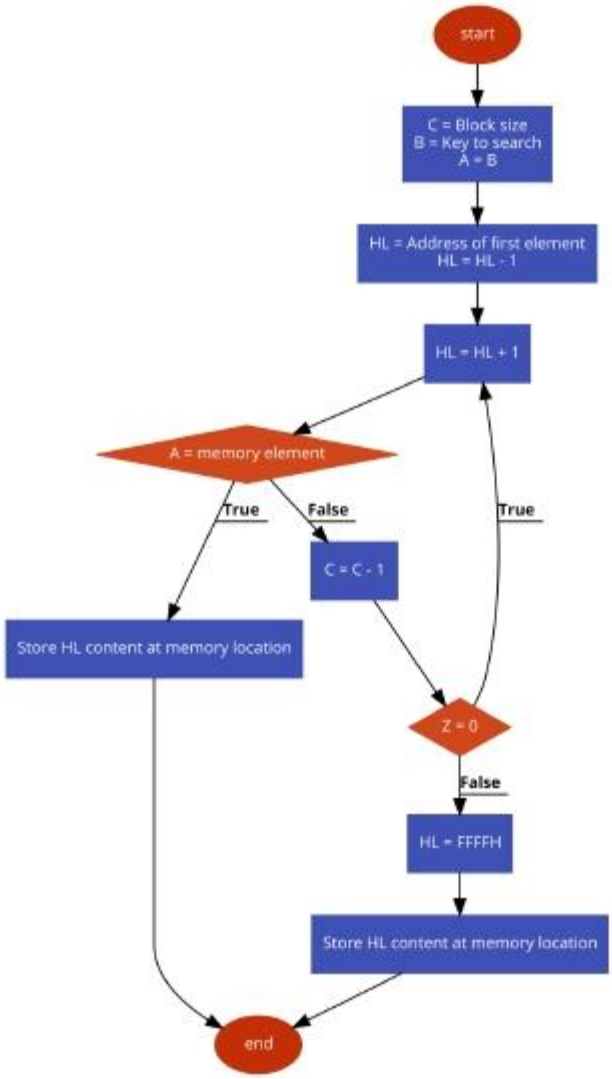
In this program the data are stored at location 8002H to 8007H. The 8000H is containing the size of the block, and 8001H is holding the key value to search.After executing this program, it will return the address of the data where the item is found and store the address at location 9000H and9001H. If the item is not found, it will return FFFFH.

If the data is present in the memory at FFFFH, then also it will store FFFFH, so it is ambiguous condition. We are assuming that the data are not stored atFFFFH, so we have chosen that value to indicate the data is not present.

Input

Address	Data
...	...
8000	06
8001	55
8002	11
8003	22
8004	33
8005	44
8006	55
8007	66
...	...

Flow Diagram



Program

Address	HEX Codes	Labels	Mnemonics	Comments
F000	21, 00, 80		LXI H,8000H	Point to get array size
F003	4E		MOV C, M	Get the size of array
F004	23		INX H	Point to next location
F005	46		MOV B, M	Get the key value to search

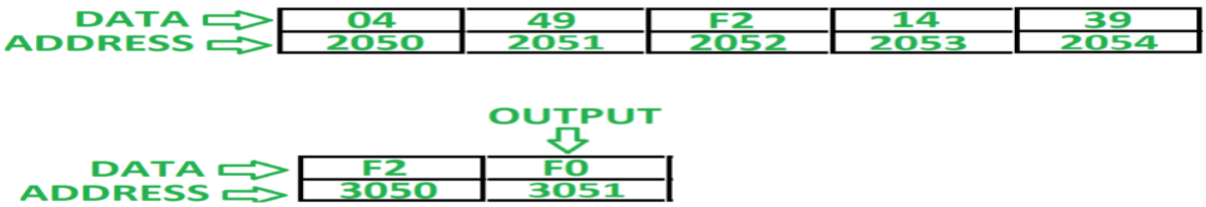
Address	HEX Codes	Labels	Mnemonics	Comments
F006	78		MOV A, B	Take the key into acc
F007	23	LOOP	INX H	Point to next location
F008	BE		CMP M	Compare memory element with Acc
F009	CA, 19, F0		JZ FOUND	When Z flag is set, go to FOUND
F00C	0D		DCR C	Decrease C by1
F00D	C2, 07, F0		JNZ LOOP	When Count is not 0, jump to LOOP
F010	21, FF, FF		LXI H, FFFF	Load FFFFHinto HL pair
F013	22, 00, 90		SHLD 9000H	Store at9000H
F016	C3, 1C, F0		JMP DONE	Jump to DONE
F019	22, 00, 90	FOUND	SHLD 9000H	Store at9000H
F01C	76	DONE	HLT	Terminate the program

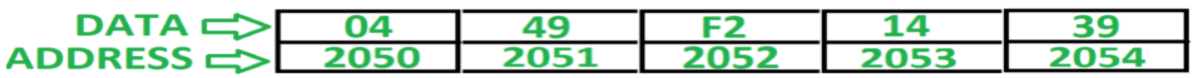
Output

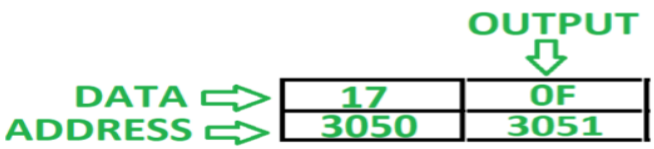
Address	Data
...	...
9000	06
9001	80

8085 program to search a number in an array of n numbers

Problem – Write an assembly language program in 8085 to search a given number in an array of n numbers. If number is found, then store F0 in memory location 3051 otherwise store 0F in 3051.
Assumptions – Count of elements in an array is stored at memory location 2050. Array is stored from starting memory address 2051 and number which user want to search is stored at memory location 3050.
Examples –







Algorithm –

1. Make the memory pointer points to memory location 2050 by help of **LXI H 2050** instruction
2. Store value of array size in register C
3. Store number to be search in register B
4. Increment memory pointer by 1 so that it points to next array index
5. Store element of array in accumulator A and compare it with value of B

6. If both are same i.e. if ZF = 1 then store F0 in A and store the result in memory location 3051 and go to step 9
7. Otherwise store 0F in A and store it in memory location 3051
8. Decrement C by 01 and check if C is not equal to zero i.e. ZF = 0, if true go to step 3 otherwise go to step 9
9. End of program

Program –

MEMORY ADDRESS	MNEMONICS	COMMENT
2000	LXI H 2050	H <- 20, L <- 50
2003	MOV C, M	C <- M
2004	LDA 3050	A <- M[3050]
2007	MOV B, A	B <- A
2008	INX H	HL <- HL + 0001
2009	MOV A, M	A <- M
200A	CMP B	A – B
200B	JNZ 2014	Jump if ZF = 0
200E	MVI A F0	A <- F0
2010	STA 3051	M[3051] <- A
2013	HLT	END
2014	MVI A 0F	A <- 0F
2016	STA 3051	M[3051] <- A
2019	DCR C	C <- C – 01
201A	JNZ 2008	Jump if ZF = 0

201D	HLT	END

Explanation – Registers used A, B, C, H, L and indirect memory M:

1. **LXI H 2050 –** initialize register H with 20 and register L with 50
2. **MOV C, M –** assign content of indirect memory location, M which is represented by registers H and L to register C
3. **LDA 3050 –** loads the content of memory location 3050 in accumulator A
4. **MOV B, A –** move the content of A in register B
5. **INX H –** increment HL by 1, i.e. M is incremented by 1 and now M will point to next memory location
6. **MOV A, M –** move the content of memory location M in accumulator A
7. **CMP B –** subtract B from A and update flags of 8085
8. **JNZ 2014 –** jump to memory location 2014 if zero flag is reset i.e. ZF = 0
9. **MVI A F0 –** assign F0 to A
10. **STA 3051 –** stores value of A in 3051
11. **HLT –** stops executing the program and halts any further execution
12. **MVI A 0F –** assign 0F to A
13. **STA 3051 –** stores value of A in 3051
14. **DCR C –** decrement C by 01
15. **JNZ 2008 –** jump to memory location 2008 if zero flag is reset
16. **HLT –** stops executing the program and halts any further execution