USN | | | | | | | | | | |

Internal Assessment Test II – APRIL 2019

Sub:	Microprocessor and Microcontroller					Sub Code:	17CS44	Branch:	CSE
Date:	16/04/2019	Duration:	90 min's	Max Marks:	50	Sem/Sec:	A,B,C		OBE

Answer any FIVE FULL Questions

		MARKS	CO	RBT
1.	Explain the following instructions with suitable examples. i) RCR ii)DAA iii)AAM iv)DAS v)SHR	[10]	CO2	L2
2.	Write an ALP to convert the ASIIC data '98473211' to the following and store the result to a reserved memory location. i) Unpacked BCD ii) Packed BCD	[10]	CO2	
3a.	What is an interrupt? Explain various types with an interrupt vector table.	[06]	CO2	L2
b.	List the steps involved while processing an Interrupt.	[04]	CO2	L2
4.	Write an Alp to perform the following. a) Clear the screen b) Set the cursor at Row=8, Column =10. c) prompt "There is a message from CMRIT, Enter Y to read it $" . If the user enters 'Y' or 'y' the message "Hello, All the best " appears on the screen. If the user enters any other key then the messge "No more Messages " must appear on the screen.	[10]	CO2	

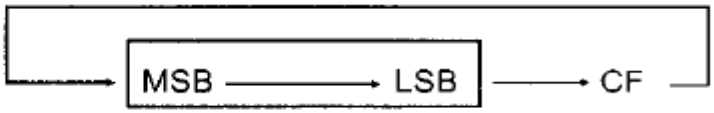
		MARKS	CO	RBT
5.	Write an ALP to scan the string "cMRITcSE" and replace the letter 'c' with 'C' and display the corrected string. Write appropriate comments	[10]	CO2	
6 a.	With suitable examples, explain how to identify overflow using flags for performing arithmetic operations on a 16 bit signed numbers.	[05]	CO2	L2
b.	Explain the following instructions with suitable examples. i) CBW ii) IDIV iii) XLAT	[05]	CO2	L2
7 a.	Explain the control word format of 8255 in I/O mode and BSR mode.	[05]	CO3	L2
b.	Write an Alp to read from P_b and check the number of ones in a given 8 bit data at P_B and display 0FFh on P_A if it is even parity else 00h on P_A if it is odd parity.	[05]	CO3	
8 a.	Differentiate between RISC and CISC processors.	[04]	CO4	L2
b.	Explain the architecture of embedded system hardware with the help of suitable block diagram.	[06]	CO4	L2

SOLUTION:

1. Explain the following instructions with suitable examples.
   i) RCR ii)DAA iii)AAM iv)DAS v)SHR

RCR rotate right through carry

In RCR, as bits are shifted from left to right, they exit the right end (LSB) to the carry flag, and the carry flag enters the left end (MSB). In other words, in RCR the LSB is moved to CF and CF is moved to the MSB. In reality, CF acts as if it is part of the operand. This is shown in the diagram. If the operand is to be rotated once, the 1 is coded, but if it is to be rotated more than once, the register CL holds the number of times.




```
        CLC             ;make CF=0
        MOV   AL,26H    ;AL=0010 0110
        RCR   AL,1      ;AL=0001 0011 CF=0
        RCR   AL,1      ;AL=0000 1001 CF=1
        RCR   AL,1      ;AL=1000 0100 CF=1
or:
        CLC             ;make CF=0
        MOV   AL,26H    ;AL=0010 0110
        MOV   CL,3      ;CL=3 number of times to rotate
        RCR   AL,CL     ;AL=1000 0100 CF=1

;the operand can be a word
        STC             ;make CF=1
        MOV   BX,37F1H  ;BX=0011 0111 1111 0001
        MOV   CL,5      ;CL=5 number of times to rotate
        RCR   BX,CL     ;BX=0001 1001 1011 1111 CF=0
```

DAA

The DAA (decimal adjust for addition) instruction in 80x86 microprocessors is provided exactly for the purpose of correcting the problem associated with BCD addition. DAA will add 6 to the lower nibble or higher nibble if needed; otherwise, it will leave the result alone. The following example will clarify these points:

```
DATA1  DB    47H
DATA2  DB    25H
DATA3  DB    ?

       MOV   AL,DATA1     ;AL holds first BCD operand
       MOV   BL,DATA2     ;BL holds second BCD operand
       ADD   AL,BL        ;BCD addition
       DAA                ;adjust for BCD addition
       MOV   DATA3,AL     ;store result in correct BCD form
```

After the program is executed, the DATA3 field will contain 72H (47 + 25 = 72). Note that DAA works only on AL. In other words, while the source can be an operand of any addressing mode, the destination must be AL in order for DAA to work. It needs to be emphasized that DAA must be used after the addition of BCD operands and that BCD operands can never have any digit greater than 9. In other words, no A - F digit is allowed. It is also important to note that DAA works only after an ADD instruction; it will not work after the INC instruction.

AAM

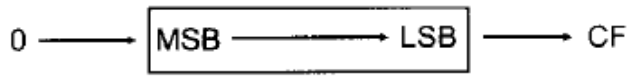
The Intel manual says that this mnemonic stands for "ASCII adjust multiplication," but it really is unpacked multiplication correction. If two unpacked BCD numbers are multiplied, the result can be converted back to BCD by AAM.

```
MOV     AL,'7'          ;AL=37H
AND     AL,0FH          ;AL=07 unpacked BCD
MOV     DL,'6'          ;DL=36H
AND     DL,0FH          ;DL=06 unpacked BCD
MUL     DL              ;AX=ALxDL. =07x06=002AH=42
AAM                     ;AX=0402 (7x6=42 unpacked BCD)
OR      AX,3030H        ;AX=3432 result in ASCII
```

The multiplication above is byte by byte and the result is HEX. Using AAM converts it to unpacked BCD to prepare it for tagging with 30H to make it ASCII.

AAD

SHR: This is the logical shift right. The operand is shifted right bit by bit, and for every shift the LSB (least significant bit) will go to the carry flag (CF) and the MSB (most significant bit) is filled with 0. Examples 3-9 and 3-10 should help to clarify SHR.



Example 3-9

Show the result of SHR in the following:

```
MOV     AL,9AH
MOV     CL,3        ;set number of times to shift
SHR     AL,CL
```

Solution:

```
9AH = 10011010
      01001101      CF=0    (shifted once)
      00100110      CF=1    (shifted twice)
      00010011      CF=0    (shifted three times)
```

After three times of shifting right, AL = 13H and CF = 0.

DAS

1. If after a SUB or SBB instruction the lower nibble is greater than 9, or if AF = 1, subtract 0110 from the lower 4 bits.
2. If the upper nibble is greater than 9, or CF = 1, subtract 0110 from the upper nibble.

Due to the widespread use of BCD numbers, a specific data directive, DT, has been created. DT can be used to represent BCD numbers from 0 to $10^{20}-1$ (that is, twenty 9s). Assume that the following operands represent the budget, the expenses, and the balance, which is the budget minus the expenses.

```
BUDGET      DT      87965141012
EXPENSES    DT      31610640392
BALANCE     DT      ?                       ;balance = budget - expenses

            MOV     CX,10                   ;counter=10
            MOV     BX,00                   ;pointer=0
            CLC                             ;clear carry for the 1st iteration
BACK:       MOV     AL,BYTE PTR BUDGET[BX]  ;get a byte of the BUDGET
            SBB     AL,BYTE PTR EXPENSES[BX] ;subtract a byte from it
            DAS                             ;correct the result for BCD
            MOV     BYTE PTR BALANCE[BX],AL ;save it in BALANCE
            INC     BX                      ;increment for the next byte
            LOOP    BACK                    ;continue until CX=0
```

2. Write an ALP to convert the ASIIC data '98473211' to the following and store the result to a reserved memory location.
   i) Unpacked BCD ii) Packed BCD

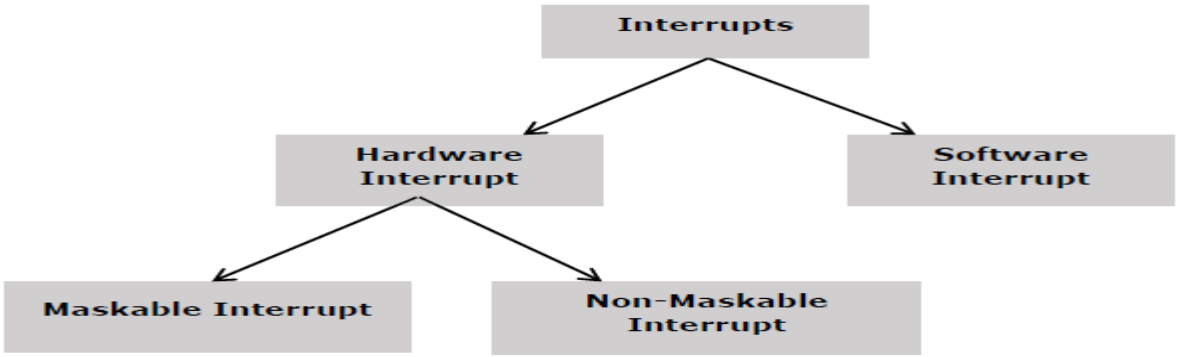
```
.MODEL SMALL
.STACK 64H
.DATA
Data_ASC DB '98473211'

LEN DW (LEN-DATA_ASC)/2

PK_BCD DB LEN DUP(0)
UN_BCD DWLEN DUP(0)

.CODE

MOV AX,@DATA
MOV DS,AX

LEA BX, DATA_ASC
LEA SI, PK_BCD
LEA DI,UN_BCD
MOV CX,LEN

AGAIN: MOV AX,[BX]      ; AH= 38H, AL=39H
AND AX, 0F0FH       ; AH=08H, AL=09H
```

```
MOV [DI], AX
PUSH CX
MOV CL,4
SHL AL,CL ; AL=90H
POP CX

ADD AL, AH ; AL=98H, AH=08H
MOV [SI], AL
ADD BX, 2
INC SI
INC DI
LOOP AGAIN


MOV AH, 4CH
INT 21H
END
```

3a.What is an interrupt? Explain various types with an interrupt vector table.

Interrupt is the method of creating a temporary halt during program execution and allows peripheral devices to access the microprocessor. The microprocessor responds to that interrupt with an **ISR** (Interrupt Service Routine), which is a short program to instruct the microprocessor on how to handle the interrupt.

The figure shows the types of interrupts that is present in a 8086 microprocessor –



Hardware Interrupts

Hardware interrupt is caused by any peripheral device by sending a signal through a specified pin to the microprocessor.

The 8086 has two hardware interrupt pins, i.e. NMI and INTR. NMI is a non-maskable interrupt and INTR is a maskable interrupt having lower priority. One more interrupt pin associated is INTA called interrupt acknowledge.

NMI:

It is a single non-maskable interrupt pin (NMI) having higher priority than the maskable interrupt request pin (INTR) and it is of type 2 interrupt.

INTR:

The INTR is a maskable interrupt because the microprocessor will be interrupted only if interrupts are enabled using set interrupt flag instruction. It should not be enabled using clear interrupt Flag instruction. The INTR interrupt is activated by an I/O port. If the interrupt is enabled and NMI is disabled, then the microprocessor first completes the current execution and sends ‘0’ on INTA pin twice. The first ‘0’ means

INTA informs the external device to get ready and during the second '0' the microprocessor receives the 8 bit, say X, from the programmable interrupt controller.
There are 8 maskable interrupts present in 8086. They are mapped to INT 8H to INT 0FH

Software Interrupts

Some instructions are inserted at the desired position into the program to create interrupts. These interrupt instructions can be used to test the working of various interrupt handlers.

INT- Interrupt instruction with type number

It is 2-byte instruction. First byte provides the op-code and the second byte provides the interrupt type number. The first five pointers are dedicated interrupt pointers. i.e. −

TYPE 0 interrupt represents division by zero situation and division overflow.

TYPE 1 interrupt represents single-step execution during the debugging of a program.

TYPE 2 interrupt represents non-maskable NMI interrupt.

TYPE 3 interrupt represents break-point interrupt.

TYPE 4 interrupt represents overflow interrupt.

INT 3-Break Point Interrupt Instruction

This instruction is inserted into the program so that when the processor reaches there, then it stops the normal execution of program and follows the break-point procedure.

INTO - Interrupt on overflow instruction

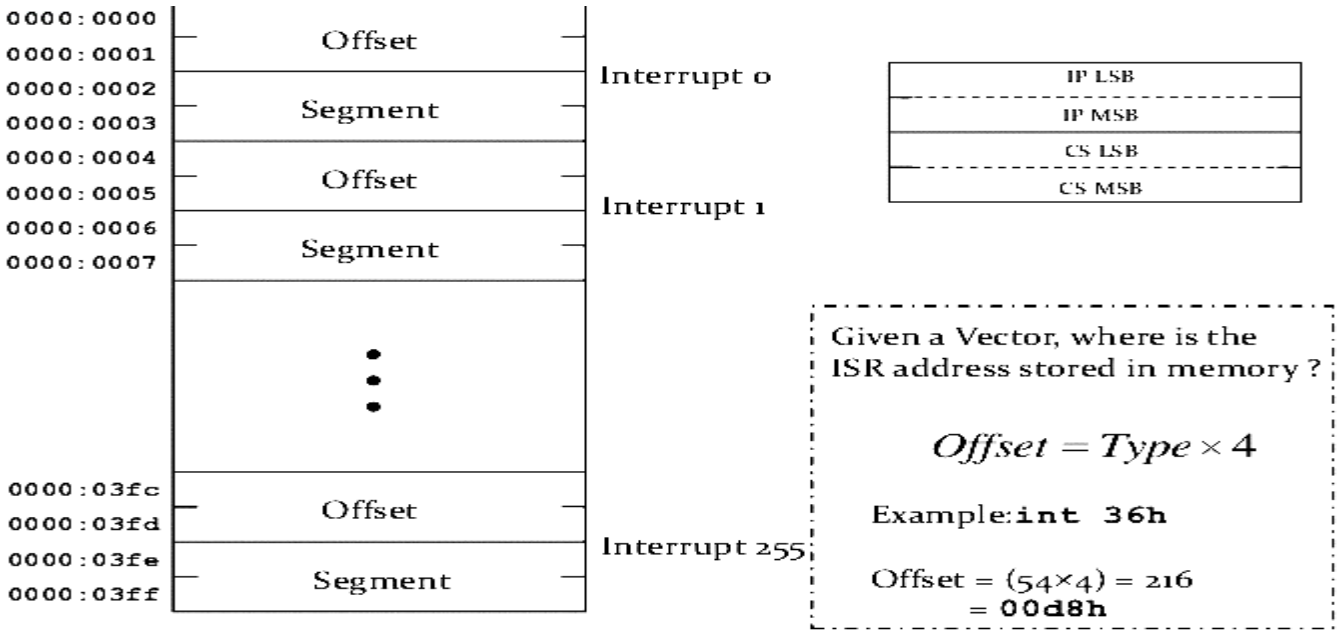
It is a 1-byte instruction and their mnemonic **INTO**. As the name suggests it is a conditional interrupt instruction, i.e. it is active only when the overflow flag is set to 1 and branches to the interrupt handler whose interrupt type number is 4. If the overflow flag is reset then, the execution continues to the next instruction.

Processing of Interrupts by 8086:

Interrupt Service Routine

For every interrupt, there must be an interrupt service routine (ISR), or interrupt handler. When an interrupt is invoked, the microprocessor runs the interrupt service routine. For every interrupt, there is a fixed location in memory that holds the address of its ISR. The group of memory locations set aside to hold the addresses of ISRs is called the interrupt vector table. When an interrupt is occurred, the microprocessor stops execution of current instruction. It transfers the content of flag register, CS and IP onto stack. After this, it jumps to the memory location specified by

Interrupt Vector Table The first 1Kbyte of memory of 8086 (00000 to003FF) is set aside as a table for storing the starting addresses of Interrupt Service Routines(ISR).Since 4-bytes are required for storing starting addresses of ISRs (CS and IP), the table can hold 256 Interrupt procedures. The starting address of an ISR is often called the Interrupt Vector or Interrupt Pointer. Therefore the table is referred as Interrupt Vector Table. In this table, IP value is put at lower word of the vector & CS is put at higher vector.



b. List the steps involved while processing an Interrupt.

Processing Interrupts in 8086:

If an interrupt has been requested, the 8086 responds to the interrupt by stepping through the following series of major actions:

- Decrements the stack pointer by 2 and pushes the flag register on the stack.
- Disables the 8086 INTR interrupt input by clearing the interrupt flag in the flag register.
- Resets the trap flag in the flag register.
- Decrements the stack pointer by 2 and pushes the current code segment register contents on the stack
- Decrements the stack pointer again by 2 and pushes the current instruction pointer contents on the stack.
- The interrupt type number is multiplied by 4 to get the physical location in IVT to fetch the CS and IP of corresponding ISR
- Processor executes ISR
- The last instruction is IRET, on execution of this the processor gets back IP, CS and FR from stack and continues with execution of the program.

4. Write an Alp to perform the following.

a) Clear the screen b) Set the cursor at Row=8, Column =10.

c) prompt “There is a message from CMRIT, Enter Y to read it $” . If the user enters ‘Y’ or ‘y’ the message “Hello, All the best “ appears on the screen. If the user enters any other key then the messge “No more Messages “ must appear on the screen.

```
.MODEL SMALL
.STACK  64H
.DATA
MSG1 DB ‘There is a message from CMRIT, Enter Y to read it $’
MSG2 DB ‘Hello, All the best$’
MSG3 DB ‘No more Messages$’

.CODE
```

```
MOV AX,@DATA
MOV DS,AX

; CLEAR SCREEN
MOV AL,0
MOV BH,07H
MOV CX,0
MOV DX,184FH
MOV AH,06H
INT 10H

;SET CURSOR AT ROW 8 COLUMN 10

MOV BH,0
MOV DL, 10
MOV DH, 8
MOV AH,02
INT 10H

; PRINT MEASSAGE 1

LEA DX, MSG1
MOV AH, 09
INT 21H

; ACCEPT A CHARACETR

MOV AH,01H
INT 21H

; CHECK WHETHER THE ENTERED CHARACTER IS y OR Y AND PRINT APPROPRIATE
MESSAGE

CMP AL,'y'
JE L1
CMP AL,'Y'
JNE L2
L1: LEA DX, MSG2
JMP L3
L2: LEA DX, MSG3

L3: MOV AH, 09
INT 21H

MOV AH, 4CH
INT 21H

END
```

5. Write an ALP to scan the string “cMRITcSE” and replace the letter ‘c’ with ‘C’ and display the corrected string. Write appropriate comments

```
.MODEL SMALL
.STACK 64H
.DATA
SRC DB ‘cMRITcSE$’
LEN DW (LEN-DATA)

.CODE

MOV AX,@DATA
MOV DS, AX

LEA DI, SRC
MOV CX, LEN

MOV AL,’c’

L1: REPNE SCASB

JE REPLACE
JMP STOP

REPLACE: MOV [DI-1],’C’
        CMP CX,0
        JNZ L1


STOP: LEA DX, SRC
        MOV AH,09H
        INT 21H

        MOV AH, 4CH
        INT 21H
   END
```

7 a. Explain the control word format of 8255 in I/O mode and BSR mode.

8255

The 8255 is a 40-pin DIP chip. It has three separately accessible ports. The ports are each 8-bit, and are named A, B, and C. The individual ports of the 8255 can be programmed to be input or output, and can be changed dynamically. In addition, 8255 ports, have handshaking capability, thereby allowing interface with devices needs handshaking signals, such as printers.

Mode selection of the 8255

While ports A, B, and C are used to input or output data, it is the control register that must be programmed to select the operation mode of the three ports. The ports of the 8255 can be programmed in any of the following modes.

1. Mode 0, simple I/O mode. In this mode, any of the ports A, B, CL, and CU can be programmed as input or output. In this mode, all bits are out or all are in. In

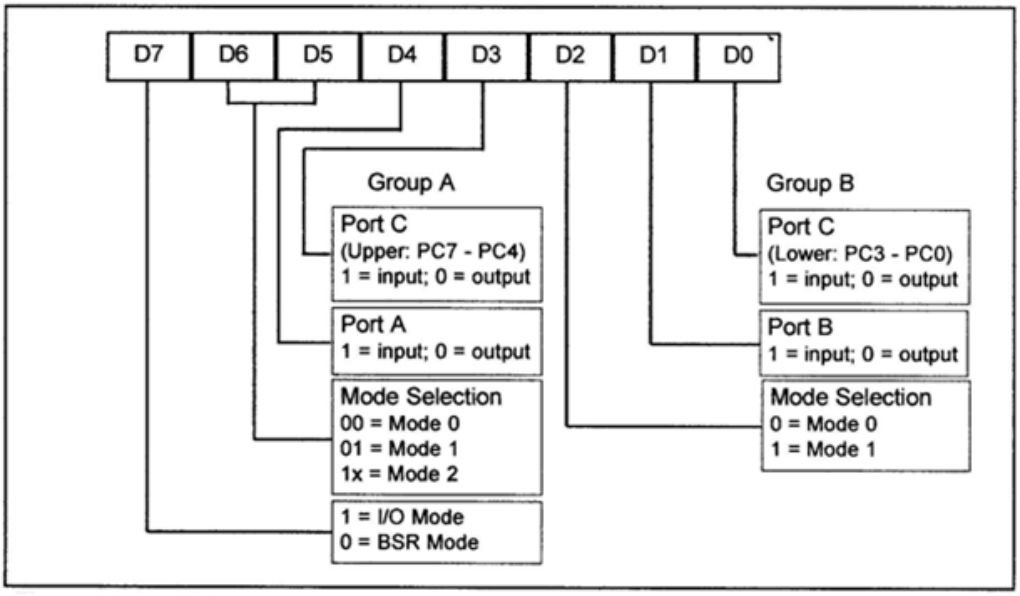
other words, there is no such thing as single-bit control as we have seen in PO – P3 of the 8051. Since the vast majority of applications involving the 8255 use this simple I/O mode, we will concentrate on this mode in this chapter.

2. Mode 1. In this mode, ports A and B can be used as input or output ports with handshaking capabilities. Handshaking signals are provided by the bits of port C.
3. Mode 2. In this mode, port A can be used as a bidirectional I/O port with hand shaking capabilities whose signals are provided by port C. Port B can be used either in simple I/O mode or handshaking mode 1.
4. BSR (bit set/reset) mode. In this mode, only the individual bits of port C can be programmed.

The 8255 chip is programmed in any of the 4 modes mentioned by sending a byte to the control register of the 8255. We must first find the port addresses assigned to each of ports A, B, C, and the control register. This is called *mapping* the I/O port.
Instructions for input and output port transfer

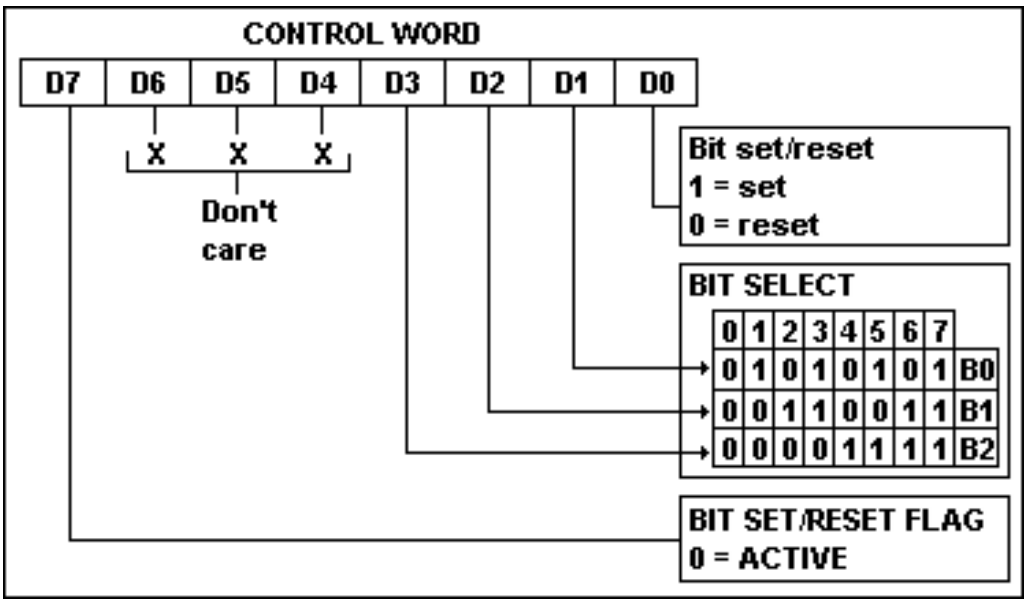
- IN − Used to read a byte from the provided port to the accumulator.
- OUT − Used to send out a byte from the accumulator to the provided port.

Control Word register format:
I/O mode



Control Word Format 8255A

BSR Mode



Control word if PA=o/p, PB =i/p, PCL=i/p, PCU=o/p
CONTROL WORD: 10000011 = 83H

Control word if PA=i/p, PB =0/p, PC=o/p
CONTROL WORD: 10010000 = 90H

b.Write an Alp to read from Pb and check the number of ones in a given 8 bit data at PB and display 0FFh on PA if it is even parity else 00h on PA if it is odd parity.

Control word if PA=o/p, PB =i/p, PC=o/p
CONTROL WORD: 10000010 = 82H

```
.MODEL SMALL
.STACK 100
.DATA
PA EQU 300H
PB EQU 301H
CT EQU 303H
.CODE
MOV AX, @DATA
MOV DS, AX
MOV DX, CT
MOV AL, 82H
OUT DX, AL
MOV DX, PB
IN AL, DX
; check the number of ones
MOV CX,8
MOV BL,00
BACK:SHR AL,1
JNC ZERO
INC BL; Number of ones
ZERO:LOOP BACK
SHR BL,1 ;check number of ones even number or not
```

```
JNC DISP
MOV AL,00H
JMP LAST
DISP:MOV AL,0FFH
LAST:MOV DX, PA
OUT DX, AL
MOV AH, 4CH
INT 21H
END
```

8 a. Differentiate between RISC and CISC processors.

The RISC (Reduced Instruction Set Computer) philosophy concentrates on reducing the complexity of instructions performed by the hardware because it is easier to provide greater flexibility and intelligence in software rather than hardware. As a result, a RISC design places greater demands on the compiler. In contrast, the traditional complex instruction set computer (CISC) relies more on the hardware for instruction functionality, and consequently the CISC instructions are more complicated

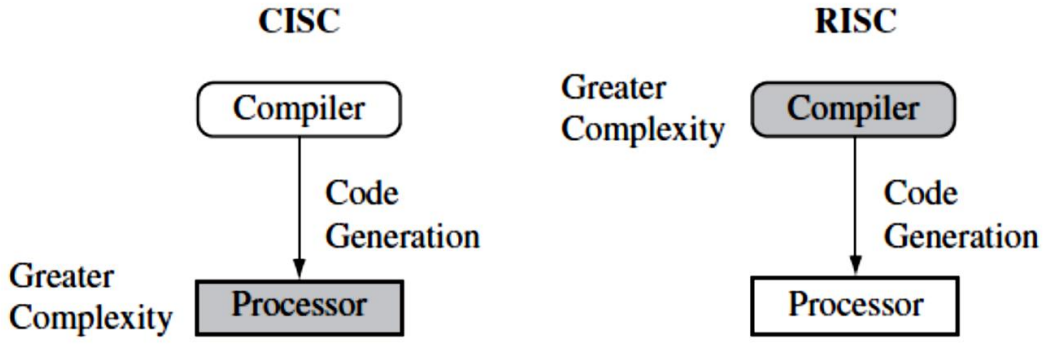
Instructions—RISC processors have a reduced number of instruction classes. These classes provide simple operations that can each execute in a single cycle. The compiler or programmer synthesizes complicated operations (for example, a divide operation) by combining several simple instructions. Each instruction is a fixed length to allow the pipeline to fetch future instructions before decoding the current instruction. In contrast, in CISC processors the instructions are often of variable size and take many cycles to execute.

Pipelines— The processing of instructions is broken down into smaller units that can be executed in parallel by pipelines. Ideally the pipeline advances by one step on each cycle for maximum throughput. Instructions can be decoded in one pipeline stage. There is no need for an instruction to be executed by a mini program called microcode as on CISC processors.

Registers—RISC machines have a large general-purpose register set. Any register can contain either data or an address. Registers act as the fast local memory store for all data processing operations. In contrast, CISC processors have dedicated registers for specific purposes.

Load-store architecture—The processor operates on data held in registers. Separate load and store instructions transfer data between the register bank and external memory. Memory accesses are costly, so separating memory accesses from data processing provides an advantage because you can use data items held in the register bank multiple times without needing multiple memory accesses. In contrast, with a CISC design the data processing operations can act on memory directly.

Hardware complexity- RISC emphasizes on software complexity while CISC emphasizes on hardware complexity



b. Explain the architecture of embedded system hardware with the help of suitable block diagram.

Embedded systems can control many different devices, from small sensors found on a production line, to the real-time control systems used on a NASA space probe. All these devices use a combination of software and hardware components. Each component is chosen for efficiency and, if applicable, is designed for future extension and expansion.

Embedded System Hardware:

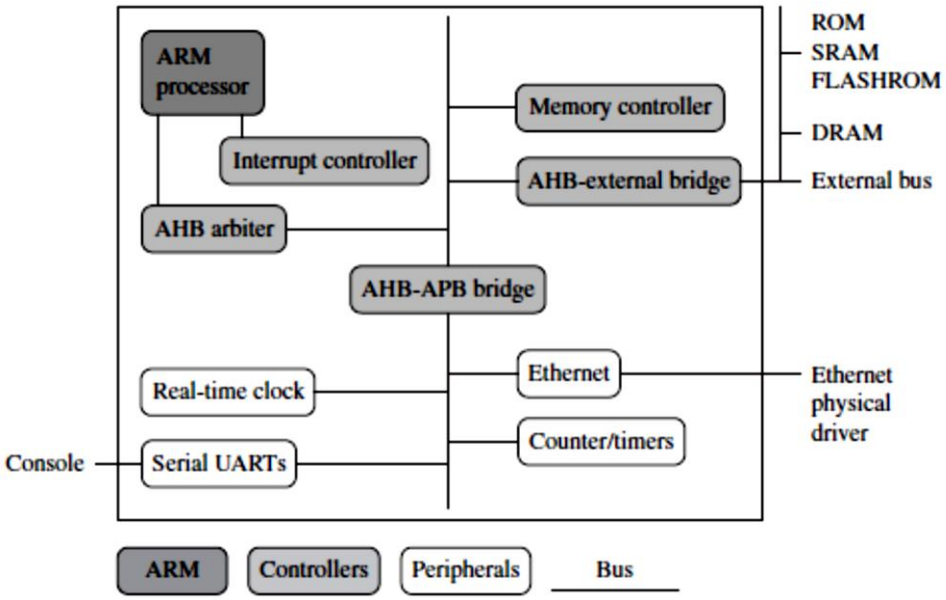


Figure 8.a.1 ARM based embedded device

Figure 8.a.1 shows a typical embedded device based on an ARM core. We can separate the device into four main hardware components:

1. The ARM processor controls the embedded device. Different versions of the ARM processor are available to suit the desired operating characteristics. An ARM processor comprises a core (the execution engine that processes instructions and manipulates data) plus the surrounding components that interface it with a bus. These components can include memory management and caches.
2. Controllers coordinate important functional blocks of the system. Two commonly found controllers are interrupt and memory controllers.
3. The peripherals provide all the input-output capability external to the chip and are responsible for the uniqueness of the embedded device.
4. A bus is used to communicate between different parts of the device.

ARM Bus Technology: embedded devices use an on-chip bus that is internal to the chip and that allows different peripheral devices to be interconnected with an ARM core.

There are two different classes of devices attached to the bus. The ARM processor core is a bus master—a logical device capable of initiating a data transfer with another device across the same bus. Peripherals tend to be bus slaves—logical devices capable only of responding to a transfer request from a bus master device.

- AMBA Bus Protocol: The Advanced Microcontroller Bus Architecture (AMBA) has been widely adopted as the on-chip bus architecture used for ARM processors. The first AMBA buses introduced were the ARM System Bus (ASB) and the ARM Peripheral Bus (APB). Later ARM introduced another bus design, called the ARM High Performance Bus (AHB).

- Using AMBA, peripheral designers can reuse the same design on multiple projects. A peripheral can simply be bolted on to the on-chip bus without having to redesign an interface for each different processor architecture.
- ASB is a bidirectional bus design.
- APB is used with slower peripherals.
- AHB is based on a centralized multiplexed bus scheme, thus runs at higher clock speeds and provides higher data through put. AHB bus is used for the high performance peripherals.

Memory:

An embedded system has to have some form of memory to store and execute code. Cost, performance, and power consumption are the parameters considered while deciding upon specific memory characteristics, such as hierarchy, width, and type. Like if memory has to run twice as fast to maintain a desired bandwidth, then the memory power requirement may be higher.

- Hierarchy: Memory can be Cache, Main memory or Secondary memory.

The fastest memory cache is physically located nearer the ARM processor core and the slowest secondary memory is set further away. Generally the closer memory is to the processor core, the more it costs and the smaller its capacity. The cache is placed between main memory and the core. It is used to speed up data transfer between the processor and main memory. The main memory is large and is generally stored in separate chips. Load and store instructions access the main memory unless the values have been stored in the cache for fast access. Secondary storage is the largest and slowest form of memory. Hard disk drives and CD-ROM drives are examples of secondary storage. Many small embedded systems do not require the performance benefits of a cache.

- Width: The memory width is the number of bits the memory returns on each access—typically 8, 16, 32, or 64 bits. The memory width has a direct effect on the overall performance and cost ratio. If you have an un-cached system using 32-bit ARM instructions and 16-bit-wide memory chips, then the processor will have to make two memory fetches per instruction. Each fetch requires two 16-bit loads. This obviously has the effect of reducing system performance, but the benefit is that 16-bit memory is less expensive. In contrast, if the core executes 16-bit Thumb instructions, it will achieve better performance with a 16-bit memory. The higher performance is a result of the core making only a single fetch to memory to load an instruction. Hence, using Thumb instructions with 16-bit-wide memory devices provides both improved performance and reduced cost.
- Types: RAM or ROM
    - Read only memory (ROM) is the least flexible of all memory types because it contains an image that is permanently set at production time and cannot be reprogrammed. ROMs are used in high-volume devices that require no updates or corrections. Many devices also use a ROM to hold boot code.
    - Random Access memory (RAM)- SRAM,DRAM or SDRAM

Peripherals

Embedded systems that interact with the outside world need some form of peripheral device. A peripheral device performs input and output functions for the chip by connecting to other devices that are off-chip.

All ARM peripherals are memory mapped—the programming interface is a set of memory-addressed registers. The address of these registers is an offset from a specific peripheral base address.

Controllers are specialized peripherals that implement higher levels of functionality within an embedded system. Two important types of controllers are memory controllers and interrupt controllers. Memory controllers connect different types of memory to the processor bus. On power-on a memory controller is configured in hardware to allow certain memory devices to be active. These memory devices allow the initialization code to be executed. Some memory devices must be set up by software.

An interrupt controller provides a programmable governing policy that allows software to determine which peripheral or device can interrupt the processor at any specific time by setting the appropriate bits in the interrupt controller registers. There are two types of interrupt controller available for the ARM processor: the standard interrupt controller and the vector interrupt controller (VIC). The standard interrupt controller sends an interrupt signal to the processor core when an external device requests servicing. It can be programmed to ignore or mask an individual device or set of devices. The VIC is more powerful than the standard interrupt controller because it prioritizes interrupts and simplifies the determination of which device caused the interrupt.