SANCET

Regulation 2017

EE6008 MICROCONTROLLER BASED SYSTEM DESIGN L T P C 3 0 0 3

UNIT I - INTRODUCTION TO PIC MICROCONTROLLER 9

Introduction to PIC Microcontroller–PIC 16C6x and PIC16C7x Architecture–PIC16cxx–-Pipelining -Program Memory considerations – Register File Structure - Instruction Set - Addressing modes –Simple Operations.

UNIT II - INTERRUPTS AND TIMER 9

PIC micro controller Interrupts- External Interrupts-Interrupt Programming–Loop time subroutine -Timers-Timer Programming– Front panel I/O-Soft Keys– State machines and key switches– Display of Constant and Variable strings.

UNIT III - PERIPHERALS AND INTERFACING 9

I2C Bus for Peripherals Chip Access– Bus operation-Bus subroutines– Serial EEPROM—Analog to Digital Converter–UART-Baud rate selection–Data handling circuit–Initialization - LCD and keyboard Interfacing -ADC, DAC, and Sensor Interfacing.

UNIT IV - INTRODUCTION TO ARM PROCESSOR 9

ARM Architecture –ARM programmer's model –ARM Development tools- Memory Hierarchy –ARM Assembly Language Programming–Simple Examples–Architectural Support for operating systems.

UNIT V - ARM ORGANIZATION 9

3-Stage Pipeline ARM Organization– 5-Stage Pipeline ARM Organization–ARM Instruction Execution- ARM Implementation– ARM Instruction Set– ARM coprocessor interface– Architectural support for High Level Languages – Embedded ARM Applications.

TOTAL: 45 PERIODS

1. Aim & Objective of the subject

- To introduce the architecture of PIC microcontroller
- To educate on use of interrupts and timers
- To educate on the peripheral devices for data communication and transfer
- To introduce the functional blocks of ARM processor
- To educate on the architecture of ARM processors

2. Need & Importance of the subject

- This subject will be helpful to do the project by their own because the secrets behind the electronic goods can be extracted by grabbing the subject with at most interest.
- This subject will give a basic knowledge in embedded system
- This subject will enhance the knowledge in mobile phones

3. Industrial Connectivity & Latest Development

- Cortex-A9 used in Samsung Galaxy S II, Sony Xperia U, Apple iPad
- ARM926EJ-S used in Sony Ericsson (K,W series), LG arena
- Cortex-A8 used in HTC Desire , Apple iPhone3GS
- ARM710 used in Acorn RISC PC 700

Department of Electrical and Electronics Engineering

Detailed Lesson Plan

Name of the Subject& Code: EE 6008 - MICROCONTROLLER BASED SYSTEM DESIGN

Text Book

1. Peatman,J.B., "Design with PIC Micro Controllers"PearsonEducation,3rdEdition, 2004.
2. Furber,S., "ARM System on Chip Architecture" Addison Wesley trade Computer Publication,

References

1. Mazidi, M.A.,"PIC Microcontroller" Rollin Mckinlay, Danny causey Printice Hall of India, 2007.

(Copies Available in Library)

S.No	Unit	Topic to be covered	Hours	Cumulativ	Books
UNIT I - INTRODUCTION TO PIC MICROCONTROLLER					
1	1	Introduction to PIC	1	1	T 1
2	1	PIC 16C6x Architecture	1	2	T 1
3	1	PIC16C7x Architecture	1	3	T 1
4	1	Pipelining	1	4	T 1
5	1	Program Memory considerations	2	6	T 1
6	1	Register File Structure	1	7	T 1
7	1	Instruction Set	1	8	T 1
8	1	Addressing modes	1	9	T 1
9	1	Simple Operations	2	11	T 1
UNIT II - INTERRUPTS AND TIMER					
10	2	PIC micro controller Interrupts	1	12	T1
11	2	External Interrupts	1	13	T1

12	2	Interrupt Programming	1	14	T1
13	2	Loop time subroutine	1	15	T1
14	2	Timers	1	16	T1
15	2	Timer Programming	1	17	T1
16	2	Front panel I/O	1	18	T1
17	2	Soft Keys	1	19	T1
18	2	State machines and key switches	1	20	T1
19	2	Display of Constant and Variable strings	1	21	T1
UNIT III - PERIPHERALS AND INTERFACING					
20	3	I2C Bus for Peripherals Chip Access	1	22	T1
21	3	Bus operation	1	23	T1
22	3	Bus subroutines	1	24	T1
23	3	Serial EEPROM	1	25	T1
24	3	Analog to Digital Converter	1	26	T1
25	3	UART	1	27	T1
26	3	Baud rate selection	1	28	T1
27	3	Data handling circuit Initialization	1	29	T1
28	3	LCD and keyboard Interfacing	2	31	T1
29	3	ADC, DAC Sensor Interfacing	2	33	T1
UNIT IV - INTRODUCTION TO ARM PROCESSOR					
30	4	ARM Architecture	1	34	T2
31	4	ARM programmer’s model	1	35	T2
32	4	ARM Development tools	1	36	T2
33	4	Memory Hierarchy	1	37	T2
34	4	ARM Assembly Language Programming	1	38	T2
35	4	Simple Examples	2	40	T2
36	4	Architectural Support for operating	2	42	T2
UNIT V - ARM ORGANIZATION					
37	5	Introduction to ARM mobiles	1	43	T2
38	5	3-Stage Pipeline ARM Organization	1	44	T2

39	5	5-Stage Pipeline ARM Organization	1	45	T2
40	5	ARM Instruction Execution	1	46	T2
41	5	ARM Implementation	1	47	T2
42	5	ARM Instruction Set	1	48	T2
43	5	ARM coprocessor interface	2	50	T2
44	5	Architectural support for High Level Languages	2	52	T2
45	5	Embedded ARM Applications	1	53	T2

INDEX

UNIT-1

INTRODUCTION TO PIC MICROCONTROLLER

PART - A

1. What is the difference between 8051 and PIC?

- 8051 is a 8 bit microcontroller by Motorola. It contains 128 byte of RAM, 4KB of ROM, four I/O ports and two timers.
- PIC refers to Peripheral Interface Control. PIC is a Harvard architecture microcontroller 8 bit and 16 bit by Microchip Technology. It is based on RISC, 32 KB of ROM , five I/O ports and four timers.

2. What is I/O port of PIC?

I/O port is used to get and send the data from/to external devices.Some I/O pins have multifunctions.

3. Write any four instructions of PIC microcontroller and state in a line the operation performed.

MOVLW 25H ; WREG=25

ADDLW 0X34 ; ADD 34H TO WREG

ADDLW ; ADD 11H TO WREG

ADDLW ; W=W+12H=7CH

4. What is RISC?

- RISC means Reduced Instruction set computer.
- Small set of frequently used instructions.
- Acess speed is high and low cost.

5. What are the groups of instruction set in PIC micro controller?

1. Bit oriented Instructions 2. Instructions using a literal value

3. Byte oriented instructions 4. Table read and writes instructions

5. Control instructions using branch and call.

6. Using the instruction of PIC microcontroller, convert BCD to Hex.

MOVLW 0X54

ADDLW 0X87

DAA.

9. Give the architecture of PIC 16C6x series.

PIC16C6x series of microcontrollers uses Harvard architecture. It uses separate buses for accessing code and data.ie., four set of buses two each (address and data) for data and code

10. Explain about watchdog timer?

It is a timer circuit which monitors the continuous functioning of processor with respect to time and prevents the endless loop hanging condition.

11. List the significance of brown out reset mode?

When the power supply falls below a certain voltage,it causes PIC to reset.This is called as brown out to reset mode.

12. List the importance of status register?

It indicates the program status after the ALU calculations are carried out, with its bit position in the status register.

13. Give the bit position of PSW.

D0-carry, D1-Digit carry,D2-Zero,D3-power down D4-time out D5-Register bank select D6,D7-undefined.

14. What is meant by PCLATH? Give its use.

It is program counter latch. Any write to PCL will cause the contents of PCLATH to be transferred to PC higher locations.

PART - B

1. Explain about the features PIC Microcontroller.

PIC Microcontrollers

PIC stands for Peripheral Interface Controller given by Microchip Technology to identify its single-chip microcontrollers. These devices have been very successful in 8-bit microcontrollers. The main reason is that Microchip Technology has continuously upgraded the device architecture and added needed peripherals to the microcontroller to suit customers' requirements.

The architectures of various PIC microcontrollers can be divided as follows.

Low - end PIC Architectures :

Microchip PIC microcontrollers are available in various types. When PIC microcontroller MCU was first available from General Instruments in early 1980's, the microcontroller consisted of a simple processor executing 12-bit wide instructions with basic I/O functions. These devices are known as low-end architectures. They have limited program memory and are meant for applications requiring simple interface functions and small program & data memories. Some of the low-end device numbers are

- ❖ 12C5XX , 16C5X, 16C505

Mid range PIC Architectures

Mid range PIC architectures are built by upgrading low-end architectures with more number of peripherals, more number of registers and more data/program memory. Some of the mid-range devices are

- ❖ 16C6X , 16C7X, 16F87X

C=EPROM ; F=Flash ;RC = Mask ROM

Popularity of the PIC microcontrollers is due to the following factors.

1. **Speed:** Harvard Architecture, RISC architecture, 1 instruction cycle = 4 clock cycles.
2. **Instruction set simplicity:** The instruction set consists of just 35 instructions (as opposed to 111 instructions for 8051).
3. **Power-on-reset and brown-out reset:** Brown-out-reset means when the power supply goes below a specified voltage (say 4V), it causes PIC to reset;. A watch dog timer (user programmable) resets the processor if the software/program ever malfunctions and deviates from its normal operation.
4. PIC microcontroller has four optional **clock sources:**

   Low power crystal

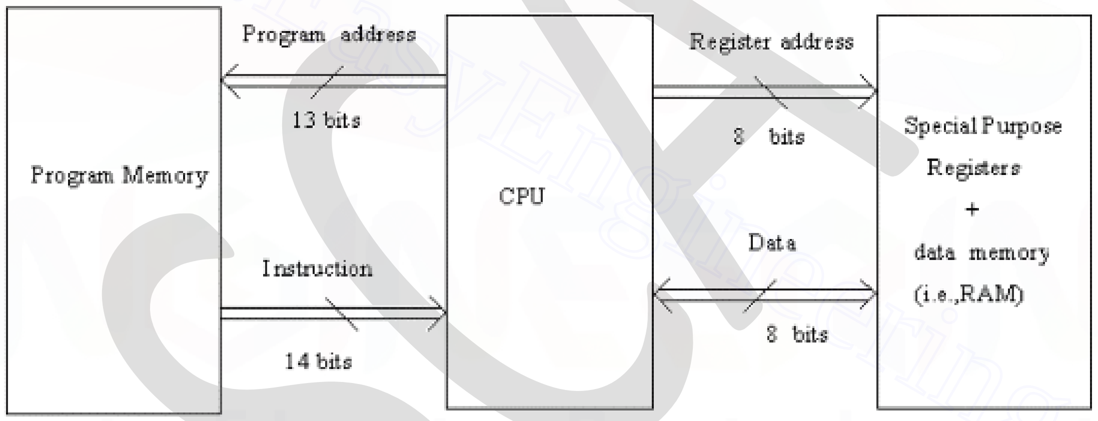
   Mid range crystal

   High range crystal

   RC oscillator (low cost).

5. Programmable timers and on-chip ADC.
6. Up to 12 independent interrupt sources.
7. Powerful output pin control (25 mA (max.) current sourcing capability per pin.)
8. EPROM/OTP/ROM/Flash memory option.
9. I/O port expansion capability.

CPU Architecture:



The CPU uses Harvard architecture with separate Program and Variable (data) memory interface. This facilitates instruction fetch and the operation on data/accessing of variables simultaneously.

Pipelining of instruction fetch successive addressing

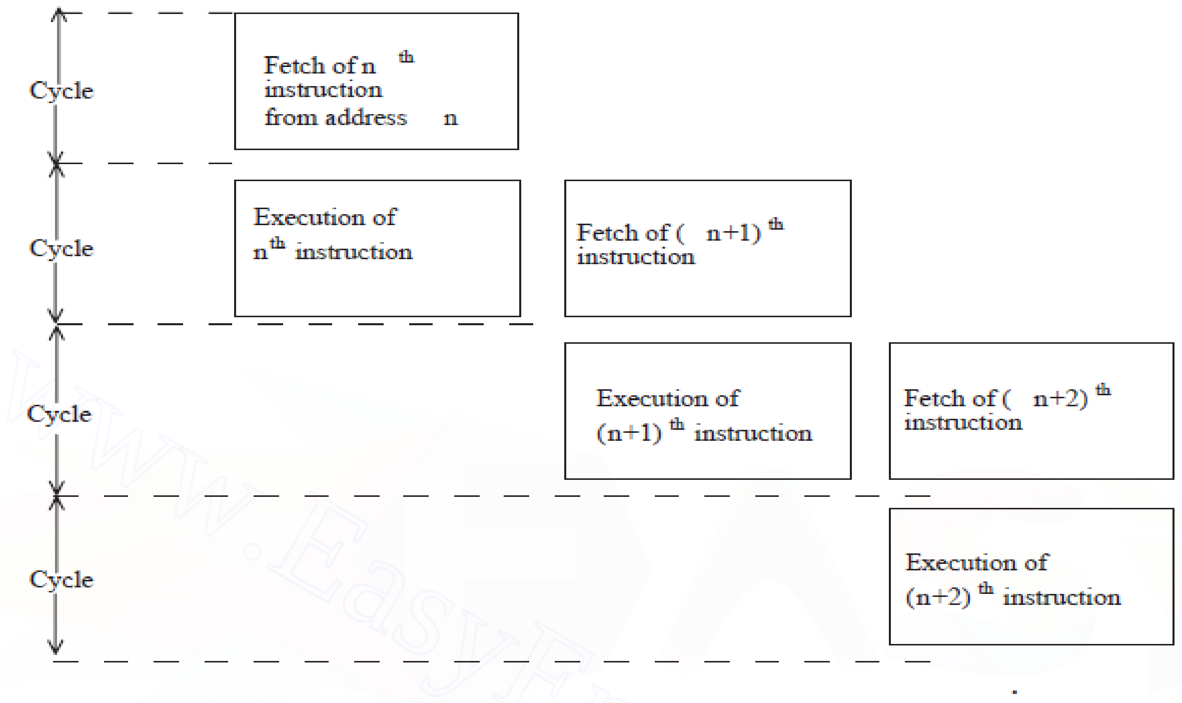

Cycle
Cycle
Cycle
Cycle
Fetch of n th instruction from address n
Execution of n th instruction
Fetch of (n+1) th instruction
Execution of (n+1) th instruction
Fetch of (n+2) th instruction
Execution of (n+2) th instruction

Fig:Introduction of extra cycle for a jump/goto instruction

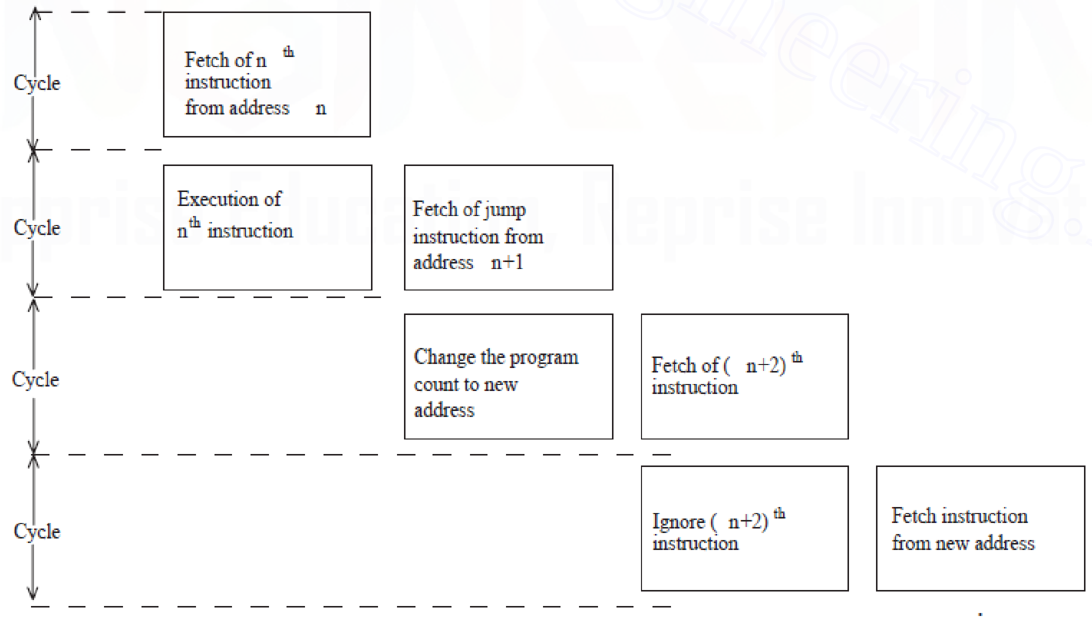

Cycle
Cycle
Cycle
Cycle
Fetch of n th instruction from address n
Execution of n th instruction
Fetch of jump instruction from address n+1
Change the program count to new address
Fetch of (n+2) th instruction
Ignore (n+2) th instruction
Fetch instruction from new address

Fig: fetching cycle

2.Explain about the Architecture of PIC 16C7X with necessary diagrams.

The PICI6C72 is a low-power, high-performance CMOS 8-bit microcomputer with 2K bytes Flash programmable and erasable read only memory (PEROM). PICI6C72 is a powerful microcomputer, which provides a highly flexible and cost-effective solution to many embedded control application.

ARCHITECTURE OF PIC 16C72 MICRO CONTROLLER: -

The PIC16C7X belongs to the Mid-Range family of the PIC micro devices. The program memory contains 2K words, which translate to 2048 instructions, since each 14-bit program memory word is the same width as each device instruction. The data memory (RAM) contains 128 bytes. There are 22 I/O pins that are user configurable on a pin-to-pin basis. Some pins are multiplexed with other device functions.

These functions include:

*External interrupt	*Change on PORTB interrupts	*Timer0 clock input
*Timer1 clock/oscillator	*Capture/Compare/PWM	*A/D converter
*SPI/I^{2}C	*Low Voltage Programming	*Incircuit Debuuger

PIN DESCRIPTION:

RA0 - RA5:

These are the bi-directional Input / output PORTA pins.

RA1, RA2, are the analog inputs 1, analog input2.

RA3 can also be analog input3 or analog reference voltage.

RA4 can also be the clock input to the Timer0 module. Output is open drain type.

RA5 can also be analog input4 or the slave select for the synchronous serial port.

RB0 – RB7:

These are the bi-directional I/O PORTB pins. PORTB can be software programmed for internal weak pull-up on all inputs.

RB0/IN is the external interrupt pin.

RB1, RB2, RB3 are the bi-directional pins.

RB4 is the Interrupt-on-change pin.

RB5 is the Interrupt-on-change pin.

RB6/PGC is the Interrupt-on-change pin. Serial programming clock.

RB7/PGD is the Interrupt-on-change pin. Serial programming data.

RC0 – RC7:

These are the bidirectional Input / Output PORTC pins.

RC0/T1OSO/ T1CK. RC0 can also be the Timer1 oscillator output or Timer1 Clock input.

RC1/T1OSI is the Timer1 oscillator input.

RC2/CCP is the Capture1 input/Compare1 output/ PWM1 output.

RC3/SCK/SCL. RC3 can also be the synchronous serial clock input/output for Both SPI and I2C modes.

RC4/SDI/SDA is the SPI Data In (SPI mode) or Data I/O (I2C mode).

RC5/SDO is e the SPI Data Out (SPI mode).

Power Control/Status Register (PCON):

The Power Control/Status Register, PCON, has two bits to indicate the type of RESET that last occurred. Bit0 is Brown-out Reset Status bit, BOR. Bit BOR is unknown on a Power-on Reset.

Power-on Reset (POR):

A Power-on Reset pulse is generated on-chip when VDD rise is detected (in the range of 1.2V - 1.7V).

Watchdog Timer (WDT):

Watchdog Timer is a free running, on-chip RC oscillator that does not require any external components. This RC oscillator is separate from the RC oscillator of the SC1/CLKI pin. That means that the WDT will run, even if the clock on the OSC1/CLKI and OSC2/ CLKO pins of the device has been stopped, for example, by execution of a SLEEP instruction.

Power-up Timer (PWRT):

The Power-up Timer operates on an internal RC oscillator. The chip is kept in RESET as long as the PWRT is active. The PWRT's time delay allows VDD to rise to an acceptable level.

Brown-out Reset (BOR):

The configuration bit, BOREN, can enable or disable the Brown-out Reset circuit.If VDD falls below VBOR for less than TBOR, a RESET may not occur.

PIN DESCRIPTION:

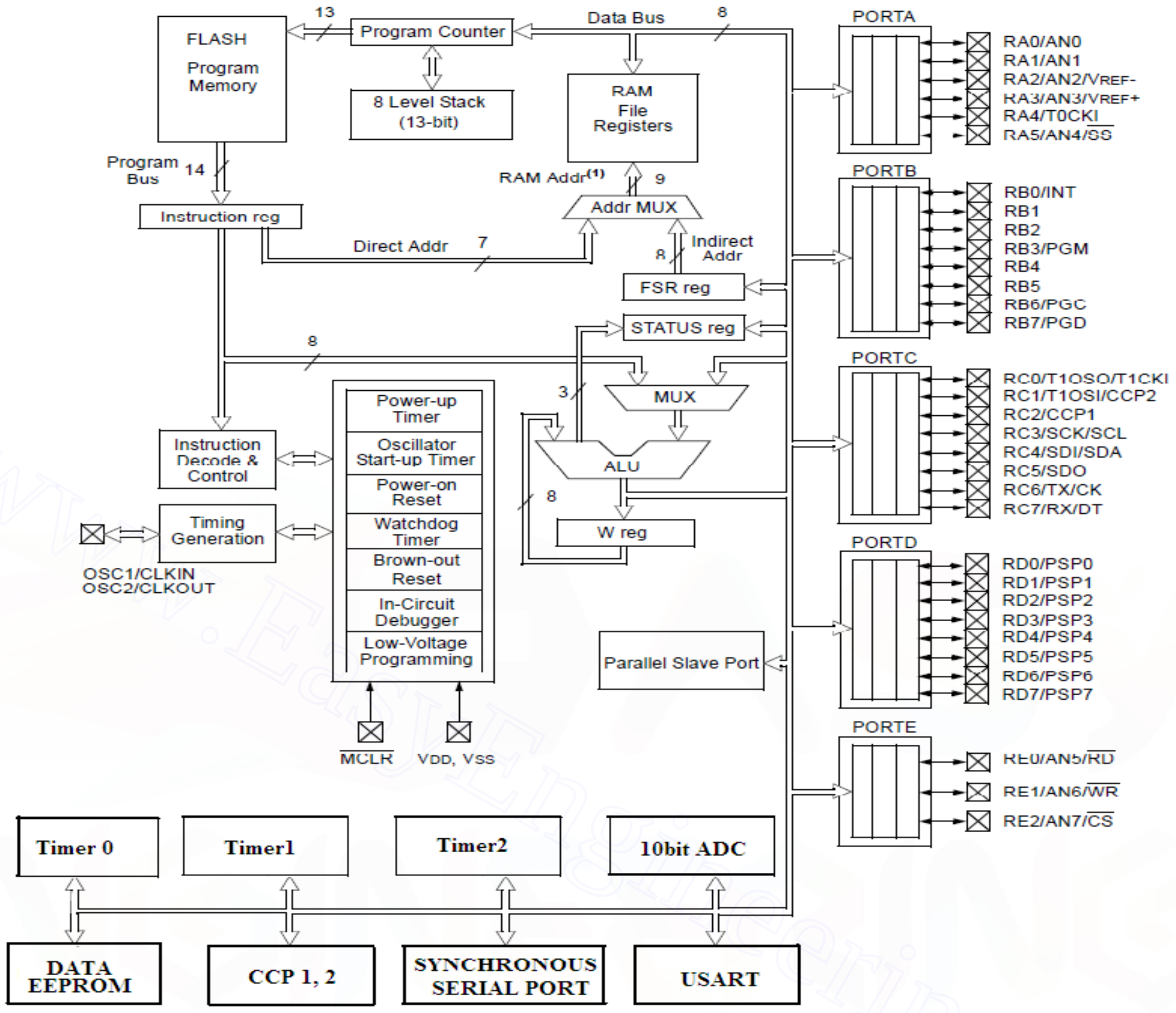
PORTA and the TRISA Register:

PORTA is a 6-bit wide, bi-directional port. The corresponding data direction register is TRISA. Setting a TRISA bit (= 1) will make the corresponding PORTA pin an input (i.e., put the corresponding output driver in a Hi-Impedance mode). Clearing a TRISA bit (= 0) will make the corresponding PORTA pin an output (i.e., put the contents of the output latch on the selected pin

PORTB and the TRISB Register:

PORTB is an 8-bit wide, bi-directional port. The corresponding data direction register is TRISB. Setting a TRISB bit (= 1) will make the corresponding PORTB pin an input (i.e., put the corresponding output driver in a Hi-Impedance mode). Clearing a TRISB bit (= 0) will make the corresponding PORTB pin an output (i.e., put the contents of the output latch on the selected pin).

ARCHITECTURE DIAGRAM OF PIC 16C72:



PORTC and the TRISC Register:

PORTC is an 8-bit wide, bi-directional port. The corresponding data direction register is TRISC. Setting a TRISC bit (= 1) will make the corresponding PORTC pin an input (i.e., put the corresponding output driver in a Hi-Impedance mode). Clearing a TRISC bit (= 0) will make the corresponding PORTC pin an output (i.e., put the contents of the output latch on the selected pin). PORTC is multiplexed with several peripheral functions. PORTC pins have Schmitt Trigger input buffers.

Capture Mode:

In Capture mode, CCPR1H: CCPR1L captures the 16-bit value of the TMR1 register when an event occurs on pin RC2/CCP1. An event is defined as:

- Every falling edge
- Every rising edge

- Every 4th rising edge
- Every 16th rising edge

An event is selected by control bits CCP1M3:CCP1M0 (CCP1CON<3:0>). When a capture is made, the interrupt request flag bit CCP1IF (PIR1<2>) is set. It must be cleared in software. If another capture occurs before the value in register CCPR1 is read, the old captured value is overwritten by the new captured value.

Compare Mode:

In Compare mode, the 16-bit CCPR1 register value is constantly compared against the TMR1 register pair value. When a match occurs, the RC2/CCP1 pin is:

- Driven High
- Driven Low
- Remains Unchanged

The action on the pin is based on the value of control bits .

ANALOG-TO-DIGITAL CONVERTER (A/D) MODULE:

The analog-to-digital (A/D) converter module has five inputs for the PIC16C7X. The A/D allows conversion of an analog input signal to a corresponding 8-bit digital number.

The A/D module has three registers:

• A/D Result Register	ADRES
• A/D Control Register 0	ADCON0
• A/D Control Register 1	ADCON1

OSCILLATOR CONFIGURATIONS:

The PIC16C7X can be operated in four different Oscillator modes. The user can program two configuration bits (FOSC1 and FOSC0) to select one of these four Modes:

• LP	Low Power Crystal
• XT	Crystal/Resonator
• HS	High Speed Crystal/Resonator
• RC	Resistor/Capacitor

3. Explain the instruction set of PIC microcontroller.

I**NSTRUCTION SET**

While writing the instructions the following guidelines are followed.

a) Write the instructions mnemonics in lower case (example: xorwf)

b) Write special Register names, RAM variable names and bit names in upper case (example: STATUS, RPO….)

c) Write instruction and subroutine labels in mixed case (example: Mainline, LoopTime..)

The instruction set of PIC is divided into Three based on size. They are

(a) Byte oriented Instructions

(b) Bit oriented Instructions

(c) Literal and Control Instructions

Byte Oriented Instructions

In a byte oriented Instructions **f** represents a file register and **d** represents destination register.The destination specifies where the result of operation is to be placed. If D= 0 the result is placed in W register(Accumulator) and if d = 1 , the result is placed in the file register specified in the instruction.

```
ADDWF f, d        ;  Add  W and  f
CLRF  f           ; Clear f
MOVWF f ,d        ; Move f
NOP               ; No operation
SUBWF  f ,d       ; Subtract W from f
```

Bit Oriented Instruction

In bit oriented instructions, b represents a bit field designator which selects the number of the bit affected by the operation and f represents the number of the file in which the bit is located**.**

```
BCF f , b         ; Bit clear f
BSF   f, b        ; Bit set f
BTFSC    f , b    ; Bit test f ,skip if  set
```

Literal and Control Instrucrtions

In literal and control instructions K represents an 8 or 11 bit constant or literal value.

```
ADDLW   k   ; Add literal and W
ANDLW   k   ; AND   literal with W
CALL k      ; Call subroutine
MOVLW k     ;  Move  literal to W
SUBLW k     ; Subtract W from literal
```

Based on the type of operation PIC supports various Instructions. They are explained below.

CLASSIFICATION OF INSTRUCTIONS

All the instructions of the PIC microcontroller are classified into nearly 9 groups. They are given below with examples.

(i).Arithmetic Operations :

ADDLW k ; Add literal value k to W

ADDWF f, d ; The contents of the W register are added with the register f.

SUBWF f ,d ; the contents of W register are subtracted from register f

(ii).Logical Instructions :

ANDLW k ; The contents of W register are ANDED with the 8-bit litweral k .The result is stored in the W register.

IORLW k ;Inclusive OR the literal value into W register

XORWF f,d ; The contents of W register are XORed with register f and the result is stored in W or f.

COMF f, d ; Complement f .

(iii).Increment/Decrement Instructions

INCF f ,d ; Increment contents of f register by 1

DECF f , d ; Decrement f by 1

(iv).Data Transfer instructions :

MOVF f,d ; Move f to W i.e The contents of register f is moved to a destination depending on d

MOVLW k ; Move literal k to W

MOVWF f ; Move W to f

(v) Clear Instructions

CLRF ;Clear file f

CLR W ; Clear the contents of W register and zero bit is set

CLRWDT ; Clear Watch dog timer

BCF ; Clear bit b of register f.

(vi)Rotate Instructions

RLF ; Rotate Left f through carry

RRF ; Rotate Right f through carry

(vii). Branch Instructions : There are two types of Branch instructions.(i)Conditional Branch and (ii) Un conditional Branch instructions**.**

(i) Conditional Branch Instructions

BTFSC f , b ; Bit Test skip if clear

BTFSS f , b ; Bit test f , skip if set

DECFSZ f,d ; Decrement f ,skip if zero.

INCFSZ f,d ;Increment f ,skip if zero

(ii) Unconditional Instructions

CALL k ; Call the subroutine k unconditionally

GOTO k ; Unconditional k branch

RETURN ; Return from subroutine

REETLW k ; Return with literal in W register.

(viii Miscellaneous

BSF f,b ; Set bit b of register f

SLEEP ; Go into stand by mode

NOP ; No operation i.e Do nothing , wait one clock cycle.

The various instructions used in PIC are presented in the Table below.

Single-bit manipulation		Operation
bcf	PORTB, 0	Clear bit 0 of PORTB
bsf	STATUS, C	Set the carry bit
Clear/move		
Clrw		Clear the working register, W
clrf	TEMP1	Clear temporary variable TEMP1
movlw	5	Load 5 into W
movlw	10	Load D ‘10’ or H ‘10’ into W depending upon default representation
movwf	TEMP1	Move W into TEMP1
movwf	TEMP1, F	Incorrect syntax
movf	TEMP1, W	Move TEMP1 into W
swapf	TEMP1, F	Swap 4-bit nibbles of TEMP1
swapf	TEMP1, W	Move TEMP1 to W, swapping nibbles and leave TEMP1 unchanged

Increment/decrement/complement		
incf	TEMP1, F	Increment TEMP1
incf	TEMP1, W	W < - TEMP1 + 1; TEMP1 unchanged
decf	TEMP1, F	Decrement TEMP1
comf	TEMP1, F	Change 0s to 1s and 1s to 0s
Multiple-bit manipulation		
andlw	B’00000111’	Force upper 5 bits of W to zero
andwf	TEMP1, F	TEMP1 < - TEMP1 and W
andwf	TEMP1, W	W < - TEMP1 AND W
iorlw	B’00000111’	Force lower 3 bits of W to one
iorwf	TEMP1, F	TEMP1 < - TEMP1 or W
xorlw	B’00000111’	Complement lower 3 bits of W
xorwf	TEMP1, W	W < - TEMP1 XOR W
Addition/Subtraction		
addlw	5	Add 5 to W
addwf	TEMP1, F	TEMP1 < - TEMP1 + W
sublw	5	W < - 5 – W (not W < - W – 5!)
subwf	TEMP1, F	TEMP1 < - TEMP1 – W
Rotate		
rlf rrf	TEMP1, F TEMP1, W	Nine-bit left rotate through C (C < - TEMP1, 7; TEMP1, I+1 < - TEMP1, I TEMP1, 0 < - C) Leave TEMP1 unchanged copy to W and rotate W right through C
Conditional branch		
btfsc btfss C decfsz incfsz	TEMP1, 0 STATUS, TEMP1 , F TEMP1, W	Skip the next instruction if bit 0 of TEMP1 equals zero Skip if C = 1 Decrement TEMP1; skip if zero Leave TEMP1 unchanged; skip if TEMP1 = H’FF’; W< - TEMP1 + 1

Goto/call/return/return from interrupt		
goto	There	Next instruction to be executed is labeled “There”
call	Task1	Pushed return address; next instruction to be executed is labeled “Task1”
return		Pop return address off of stack
retlw	5	Pop return address; W < -5
retfie		Pop return address; reenable interrupts
Miscellaneous		
Clrwdt		If watchdog timer is enabled, this; instruction will reset it (before it,;resets the CPU)
sleep		Stop clock; reduce power; wait,;for watchdog timer or external signal;to begin program execution again ;
nop		Do nothing; wait one clock cycles

4. With a neat diagram discuss in detail about memory organization of a PIC microcontroller.

Memory of the PIC16F877 divided into 3 types of memories:

- **Program Memory** - A memory that contains the program , after the user write it. Program Counter executes commands stored in the program memory, one after the other.
- **Data Memory** – This is RAM memory type, which contains a special registers like SFR (Special Faction Register) and GPR (General Purpose Register). The variables that we store in the Data Memory during the program are deleted after we turn of the micro.
- **Data EEPROM (Electrically Erasable Programmable Read-Only Memory)** - A memory that allows storing the variables as a result of burning the written program. Each one of them has a different role. Program Memory and Data Memory two memories that are needed to build a program, and Data EEPROM is used to save data after the microcontroller is turn off.Program Memory and Data EEPROM they are non-

volatile memories, which store the information even after the power is turn off. These memories called Flash or EEPROM.

(i) PROGRAM MEMORY:

- PIC16F87XA devices have a 13-bit program counter capable of addressing an 8K word x 14 bit program memory space. This memory is used to store the program after we burn it to the microcontroller.
- Program Counter (PC) keeps track of the program execution by holding the address of the current instruction. It is automatically incremented to the next instruction during the current instruction execution.
- PIC16F87XA family has a 13-bit wide hardware stack. The stack space is not part of either program or data space and the stack pointer is not readable or writable. In the PIC microcontrollers, this is a special block of RAM memory used only for this purpose.
- The CALL instruction is used to jump to a subroutine, which must be terminated with the RETURN instruction. CALL has the address of the first instruction in the subroutine as its operand. When the CALL instruction is executed, the destination address is copied to the PC. The PC is PUSHed onto the stack when a CALL instruction is executed, or an interrupt causes a branch. The stack is POP'ed in the event of a RETURN, RETLW or a RETFIE instruction execution.
- The stack operates as a circular buffer. This means that after the stack has been PUSHed eight times, the ninth push overwrites the value that was stored from the first push. The tenth push overwrites the second push (and so on).
- Each time the main program execution starts at address 0000 - Reset Vector. The address 0004 is "reserved" for the "interrupt service routine" (ISR). Program Memory is divided into pages, where the program is stored. Data Memory is divided into banks. The banks are located inside the RAM, where the special registers and the data located

(ii) Data Memory :

- The data memory is partitioned into multiple banks which contain the General Purpose registers and Special Function Registers. Number of banks may vary depending on the microcontroller; for example, micro PIC16F84 has only two banks.

- Each bank extends up to 7Fh (128 bytes). The lower locations of each bank are reserved for the Special Function Registers. Above the Special Function Registers are General Purpose Registers, implemented as static RAM.
- To access a register that is located in another bank, one should access it inside the program. There are special registers which can be accessed from any bank, such as STATUS register.

In order to start programming and build automated system, the important registers of the memory map are:

- STATUS register – changes/moves from/between the banks
- PORT registers – assigns logic values ("0"/"1") to the ports
- TRIS registers - data direction register (input/output)

Status Register

Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
IRP	RP	RP	TO	PD	Z	DC	C

BIT 7:

IRP: Register Bank Select bit (used for indirect addressing)

0 = Bank 0, 1 (00h - FFh)

1 = Bank 2, 3 (100h - 1FFh)

The IRP bit is not used by the PIC16F8X. IRP should be maintained clear.

BIT 6-5:

RP1:RP0: Register Bank Select bits (used for direct addressing)

00 = Bank 0 (00h - 7Fh)

01 = Bank 1 (80h - FFh)

10 = Bank 2 (100h - 17Fh)

11 = Bank 3 (180h - 1FFh)

BIT 4:

TO: Time-out bit

1 = After power-up, CLRWDT instruction, or SLEEP instruction

0 = A WDT time-out occurred

BIT 3:

PD: Power-down bit

1 = After power-up or by the CLRWDT instruction

0 = By execution of the SLEEP instruction

BIT 2:

Z: Zero bit

1 = The result of an arithmetic or logic operation is zero

0 = The result of an arithmetic or logic operation is not zero

BIT 1:

DC: Digit carry/borrow

1 = A carry-out from the 4th low order bit of the result occurred

0 = No carry-out from the 4th low order bit of the result bit

BIT 0:

C: Carry/borrow

1 = A carry-out from the most significant bit of the result occurred

0 = No carry-out from the most significant bit of the result occurred

Direct Addressing:

Using this method we are accessing the registers directly by detecting location inside Data Memory from Opcode and by selecting the bank using bits RP1 and RP0 of the STATUS register.

Indirect Addressing:

To implement indirect addressing, a File Select Register (FSR) and indirect register (INDF) are used. In addition, when using this method we choose bank using bit IRP of the STATUS register. Indirect addressing treated like a stack pointer, allowing much more efficient work with a number of variables. INDF register is not an actual register is a virtual register that is not found in any bank.

The role of the PORT register is to receive the information from an external source (e.g. sensor) or to send information to the external elements (e.g. LCD). The 28-pin devices have 3 I/O ports, while the 40/44-pin devices, like PIC16F877, have 5 I/O ports located in the BANK 0.

PORT REGISTERS:

1. PORTA is a 6-bit wide, bidirectional port. The corresponding data direction register is TRISA.Setting a TRISA bit (= 1) will make the corresponding PORTA pin an input. Clearing a TRISA bit (= 0) will make the corresponding PORTA pin an output.
2. PORTB is an 8-bit wide, bidirectional port. The corresponding data direction register is TRISB. Setting a TRISB bit (= 1) will make the corresponding PORTB pin an input. Clearing a TRISB bit (= 0) will make the corresponding PORTB pin an output.
3. PORTC is an 8-bit wide, bidirectional port. The corresponding data direction register is TRISC. Setting a TRISC bit (= 1) will make the corresponding PORTC pin an input. Clearing a TRISC bit (= 0) will make the corresponding PORTC pin an output.
4. PORTD is an 8-bit port with Schmitt Trigger input buffers. Each pin is individually configurable as an input or output.
5. PORTE has three pins (RE0/RD/AN5, RE1/WR/AN6 and RE2/CS/AN7) which are individually configurable as inputs or outputs. These pins have Schmitt Trigger input buffers.

(iii)Data EEPROM

The data EEPROM and Flash program memory is readable and writable during normal operation (over the full VDD range). This memory is not directly mapped in the register file space. Instead, it is indirectly addressed through the Special Function Registers.

There are six SFRs used to read and write to this memory:

EECON1	EECON2	EEDATA
EEDATH	EEADR	EEADRH

When interfacing to the data memory block, EEDATA holds the 8-bit data for read/write and EEADR holds the address of the EEPROM location being accessed. These devices have 128 or 256 bytes of data EEPROM (depending on the device), with an address range from 00h to FFh.

5.Explain in detail about register file structure in PIC microcontroller.

In PIC Microcontrollers the Register File consists of two parts namely

a) General Purpose Register File b)Special Purpose Register File

a) General Purpose Register File:

The general purpose register file is another name for the microcontroller's RAM . Data can be written to each 8-bit location updated and retrieved any number of times.

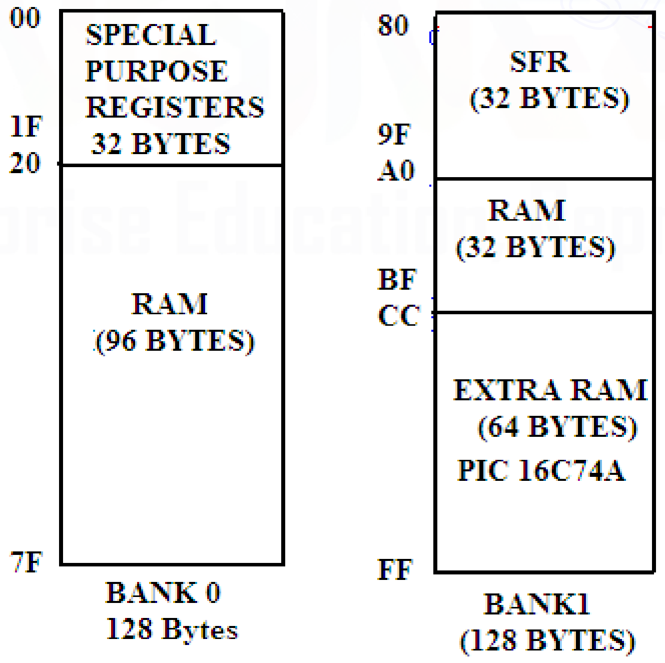
b) Special Purpose Register File:

The special function register file consists of input, output ports and control registers used to configure each 8-bit port either as input or output. It contains registers that provide the data input and data output to a chip resources like Timers, Serial Ports and Analog to Digital converter and also the registers that contains control bits for selecting the mode of operation and also enabling or disabling its operation.

CPU REGISTERS

The CPU registers are used in the execution of the instruction of the PIC microcontroller. The PIC **PIC16F877** Microcontroller has the following registers.

1. Working Register-W (Similar to Accumulator)
2. Status Register
3. FSR – File Select Register (Indirect Data memory address pointer)
4. INDF
5. Program Counter



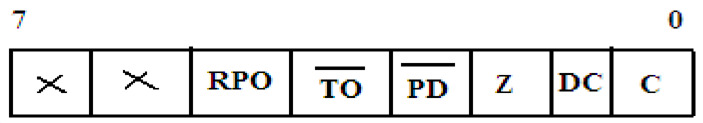
RP0	RP1	Bank
0	0	Bank 0
0	1	Bank 1

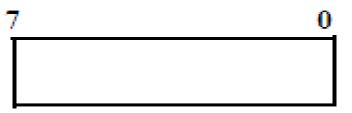
1. Working Register:

Working Register is used by many instructions as the source of an operand. It also serves as the destination for the result of instruction execution and it is similar to accumulator in other μcs and μps.

2.Status Register:

This is an 8-bit register which denotes the status of ALU after any arithmetic operation and also RESET status and the bank select bits for the data memory.





WORKING REGISTER

C: Carry/borrow bit   DC: Digit carry/borrow bit   Z: Zero bit

NOT_PD : Reset Status bit (Power-down mode bit)

NOT_TO : Reset Status bit (tme- out bit)

RPO: Register bank Select

The bits 7 and 6 of Status Register are unused by 16c6x/7x. The 'C' bit is set when two 8-bit operands are added together and a 9-bit result occurs. This 9-bit is placed in the carry bit.The DC or Digit carry bit indicates that a carry from the lower 4 bits occurred during an 8-bit addition.

The reset status bits NOT_TO and NOT_PD are used in conjunction with PIC's sleep mode. The micro controller can put itself to sleep mode to save power during intervals when it has nothing to do. It can be reset by any of three kinds. Upon reset the CPU can check these two reset status bits to determine which kind of event resettled it and then respond accordingly.

The Register bank select bit RPO is used to select either bank or bank.When RPO=0, select Bank 0, RPO=1, select Bank 1.

Example:	bcf STATUS, RPO	;	Select bank 0
	bsf STATUS, RPO	;	Select bank 1.

3.FSR – (File Select Register):

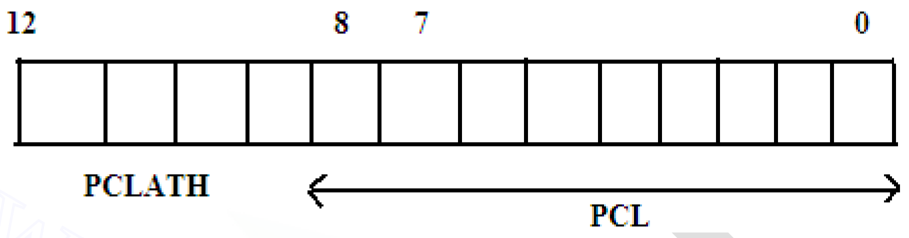
It is the pointer used for indirect addressing. In the indirect addressing mode the 8-bit register file address is first written into FSR. It is a special purpose register that serves as an address pointer to any address through out the entire register file.

4.INDF – (Indirect File):

It is not a physical register addressing but this INDF will cause indirect addressing. Any instruction using the INDF register actually access the register pointed to by the FSR.

5 .PROGRAM COUNTER

PIC PIC16F877A has a 13 bit program counter in which PCL is the lower 8-bits of the PC and PCLATH is the write buffer for the upper 5 bits of the PC.



PCLATH (program counter Latch can be read or from or written to without affecting the Program Counter(PC).The upper 3 bits of PCLATH remain zero.It is only when PCL is written to that PCLATH is automatically written into the PC at the same time.

UNIT- II

INTERRUPTS AND TIMERS

1. What is hardware and software interrupts?

PIC Microcontroller consists of both Hardware and Software Interrupts. If the interrupts are generated by external hardware at certain pins of microcontroller, or by inbuilt devices like timer, they are called Hardware Interrupts. While Software interrupts are generated by a piece of code in the program. Also known as external and internal interrupts.

2. What are the interrupts available in PIC?

Interrupt Source	Enabled by	Completion Status
External interrupt from INT	INTE = 1	INTF = 1
TMR0 interrupt	T0IE = 1	T0IF = 1
RB4–RB7 state change	RBIE = 1	RBIF = 1
EEPROM write complete	EEIE = 1	

3. **What are the features of timer0?**

The Timer0 module timer/counter has the following features:

- 8-bit timer/counter
- Readable and writable
- 8-bit software programmable prescaler
- Internal (4 Mhz) or external clock select
- Interrupt on overflow from FFh to 00h
- Edge select (rising or falling) for external clock

4. What are the features of timer 1?

- The Timer1 module, timer/counter, has the following features:
- 16-bit timer/counter consisting of two 8-bit registers (TMR1H and TMR1L)
- readable and writable
- 8-bit software programmable prescaler
- Internal (4 Mhz) or external clock select
- Interrupt on overflow from FFFFh to 0000h

5. What are the features of timer 2?

- The Timer2 module, timer/counter, has the following features:

- two 8-bit registers (TMR2 and PR2)
- readable and writable
- a prescaler and a postscaler
- Connected only to an internal clock - 4 MHz crystal
- Interrupt on overflow

6. What is state machine model.

When an 'A' is detected in the stream in state 0, the machine makes a transition from state 0 to state 1, following the edge in the direction of the arrow. If a 'B' is detected in state 1, a transition is made to state 0. Since the state machine can only occupy one state at a time, the active state indicates whether the last character detected was either 'A' or 'B'. Conceivably, another character could be received ('C'), in which case no transition occurs.

7. What is key switch?

Push button switch is connected to the first bit of PORT D (RD0) which is configured as an input pin. Which is connected to a pull up resistor such that this pin is at Vcc potential when the switch is not pressed. When the switch is pressed this pin RD0 will be grounded. The LED is connected to the first bit of PORT B (RB0) and a resistor is connected in series with it to limit the current.

8.Multiply the following two unsigned bytes 81H and 04H and save the result in registers 10H and 11H respectively.

```
MOVLW 0X81
MULLW 0X04
MOVFF PRODL, 0X10
MOVFF P\RODH, 9X11.
```

9. What are the issues in front panel I/O?

→Conversion number entered digit by digit from keypad to binary equivalent

→Display of fixed variable strings

10. **Give the timer 1 registers**

TMR1 consists of two 8-bits registers:

- TMR1H
- TMR1L

PART-B

1. Explain the concepts of interrupts in detail.

Interrupts vs. polling

A single microcontroller can serve several devices. There are two methods by which devices receive service from the microcontroller: interrupts or polling. In the →Interrupt method, whenever any device needs the microcontroller's service, the device notifies it by sending an interrupt signal. Upon receiving an interrupt signal, the microcontroller stops whatever it is doing and serves the device. The program associated with the interrupt is called the interrupt service routine (ISR) or interrupt handler.

→In polling, the microcontroller continuously monitors the status of a given device; when the status condition is met, it performs the service. After that, it moves on to monitor the next device until each one is serviced.

The polling method cannot assign priority because it checks all devices in a round-robin fashion. More importantly, in the interrupt method the microcontroller can also ignore (mask) a device request for service.

Interrupt service routine

For every interrupt, there must be an interrupt service routine (ISR), or interrupt handler. When an interrupt is invoked, the microcontroller runs the interrupt service routine. Generally, in most microprocessors, for every interrupt there is a fixed location in memory that holds the address of its ISR.

Steps in executing an interrupt

Upon activation of an interrupt, the microcontroller goes through the following steps:

1. It finishes the instruction it is executing and saves the address of the next instruction (program counter) on the stack.

2. It jumps to a fixed location in memory called the interrupt vector table. The interrupt vector table directs the microcontroller to the address of the interrupt service routine (ISR).

3. The microcontroller gets the address of the ISR from the interrupt vector table and jumps to it. It starts to execute the interrupt service subroutine until it reaches the last instruction of the subroutine, which is RETFIE (return from interrupt exit).

4. Upon executing the RETFIE instruction, the microcontroller returns to the place where it was interrupted. First, it gets the program counter (PC) address from the stack by popping the top bytes of the stack into the PC. Then it starts to execute from that address.

Sources of interrupts in the PIC18

There are many sources of interrupts in the PIC 18, depending on which peripheral is incorporated into the chip. The following are some of the most widely used sources of interrupts in the PIC 18:

I. There is an interrupt set aside for each of the timers, Timers 0, 1, 2, and so on.

2. Three interrupts are set aside for external hardware interrupts. Pins RBO (PORTB.O), RBI (PORTB.J), and RB2 (PORTB.2) are for the external hard- ware interrupts INTO, INTI, and INT2, respectively.

3. Serial communication's USART has two interrupts, one for receive and another for transmit.

4. The PORTB-Change interrupt.

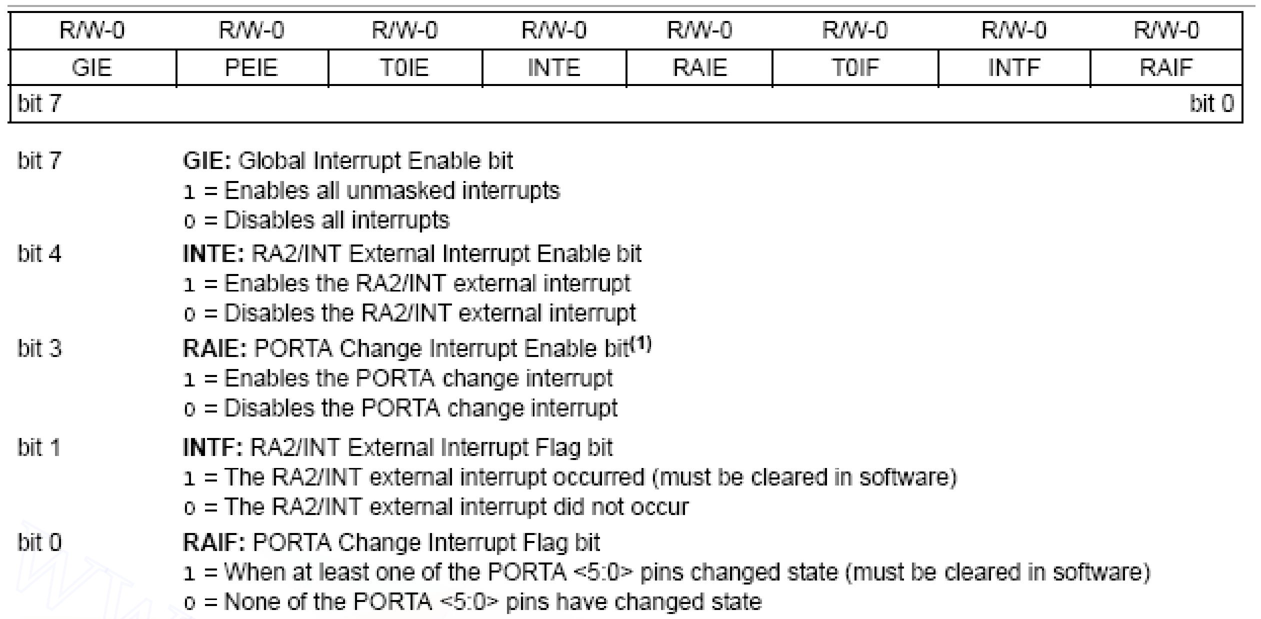
5. The ADC (analog-to-digital converter).

6. The CCP (compare capture pulse-width-modulation).

INTCON

Bit 7 (GIE) is the global interrupt enable, which is the master switch for all interrupts. Turn it off and no interrupts are enabled (regardless of the state of their individual enable bits). Turn it on and interrupts whose individual enable bits are set will be enabled.

Bit 6 (PEIE) is a mini-master switch for a group of interrupts which are known as ‘peripheral interrupts’. These interrupts have their own enable bits in the PIE1 register. Therefore, in order to use these interrupts you have to enable three bits –the individual enable bit in PIE1, this bit, and the global interrupt enable.



R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
GIE	PEIE	T0IE	INTE	RAIE	T0IF	INTF	RAIF
bit 7							bit 0

bit 7	**GIE:** Global Interrupt Enable bit 1 = Enables all unmasked interrupts 0 = Disables all interrupts
bit 4	**INTE:** RA2/INT External Interrupt Enable bit 1 = Enables the RA2/INT external interrupt 0 = Disables the RA2/INT external interrupt
bit 3	**RAIE:** PORTA Change Interrupt Enable bit(1) 1 = Enables the PORTA change interrupt 0 = Disables the PORTA change interrupt
bit 1	**INTF:** RA2/INT External Interrupt Flag bit 1 = The RA2/INT external interrupt occurred (must be cleared in software) 0 = The RA2/INT external interrupt did not occur
bit 0	**RAIF:** PORTA Change Interrupt Flag bit 1 = When at least one of the PORTA <5:0> pins changed state (must be cleared in software) 0 = None of the PORTA <5:0> pins have changed state

Bit 5 (T0IE) to use the TMR0 overflow interrupt – this simply triggers an interrupt whenever TMR0 overflows from 255 to 0. In the interrupt service routine you can test bit 2 (T0IF) to see if a TMR0 overflow interrupt has occurred

Bit 4 (INTE) controls the ‘External Interrupt’ which depends on the state of the pin labelled INT (GP2). The interrupt can be set to trigger on the rising edge orfalling edge of the signal on this pin. This is done using bit 6 of the OPTION register: if bit 6 of OPTION is clear, the INT interrupt will occur on the falling edge of the INT pin. If bit 6 of OPTION is set, the INT interrupt will occur on the risingedge.

Bit 3 (GPIE) of the INTCON register controls the GPIO change inter-rupt. This interrupt can trigger when any one of the GPIO pins changes. To use this interrupt you need to set this bit, and also select which GPIO pin should be able to trigger the interrupt. This is done with the IOC (Interrupt OnChange) reg-ister.

Enabling and disabling an interrupt

Upon reset, all interrupts are disabled (masked), meaning that none will be responded to by the microcontroller if they are activated. The interrupts must be enabled (unmasked) by software in order for the microcontroller to respond to them. The D7 bit of the INTCON (Interrupt Control) register is responsible forenabling and disabling the interrupts globally. shows the INTCON register. The GIE bit makes the job of disabling all the interrupts easy. With a single instruction (BCF INTCON,GIE), we can make GIE = 0 during the operation of a critical task.

PROGRAMMING EXTERNAL HARDWARE INTERRUPTS

The PIC 18 has three external hardware interrupts. Pins RBO (PORTB.O), RBI (PORTB.l), and RB2 (PORTB.2), designated as INTO, INTI, and INT2 respectively, are used as external hardware interrupts.

External interrupts INTO, INT1, and INT2

There are three external hardware interrupts in the PIC IS: INTO, INTI, and INT2. They are located on pins RBO, RBI, and RB2, respectively. On default, all three hardware interrupts are directed to vector table location 0008H, unless we specify otherwise. They must be enabled before they can take effect.

2. **Explain the Timers of PIC microcontroller in detail.**

he PIC18 has two to five timers depending on the family member. They are referred to as Timers 0, I, 2, 3, and 4. They can be used either as timers to generate a time delay or as counters to count events happening outside the microcontroller.

PROGRAMMING TIMERS 0 AND 1

Every timer needs a clock pulse to tick. The clock source can be internal or external. If we use the internal clock source, then 1 / 4th of the frequency of the crystal oscillator on the OSC1 and OSC2 pins (Fosc/4) is fed into the timer.

Therefore, it is used for time delay generation and for that reason is called a timer. By choosing the external clock option, we feed pulses through one of the PIC18's pins: this is called a counter. In this section we discuss the PIC 18 timer and in the next section we program the timer as a counter.

Basic registers of the timer :

Many of the PIC 18 timers are 16 bits wide. Because the PIC 18 has an 8-bit architecture, each 16-bit timer is accessed as two separate registers of low byte (TMRxL) and high byte (TMRxH). Each timer also has the TCON (timer control) register for setting modes of operation. Next, we discuss each timer separately.

Timer0 registers and programming

Timer0 can be used as an 8-bit or a 16-bit timer. The 16-bit register of TimerO is accessed as low byte and high byte, as shown in Figure 9-1. The low-byte register is called TMROL (TimerO low byte) and the high-byte register is referred to as TMROH

(TimerO high byte). These registers can be accessed like any other special function registers. For example, the instruction "MOVWF TMROL" moves the value in WREG into TMROL, the low byte of TimerO. These registers can also be read like any other register. For example, "MOVFF TMROL, PORTE" copies TMROL (low byte ofTimerO) to PORTB.

D15	D14	D13	D12	D11	D10	D9	D8	D7	D6	D5	D4	D3	D2	D1	D0

Fig: Timer 0 High & Low register

TOCON (TimerO control) register

Each timer has a control register, called TCON, to set the various timer operation modes. TOCON is an 8-bit register used for control of TimerO.

TMROON	TOBBIT	TOGS	TOSE	PSA	TOPS2	TOPS1	TOPSO

Fig: Timer 0   TCON register

TMROON D7 : TimerO ON and OFF control bit

I = Enable (start) TimerO

0 = Stop TimerO

TOSBIT D6 : TimerO 8-bit/16-bit selector bit

I = TimerO is configured as an 8-bit timer/counter.

0 = TimerO is configured as a 16-bit timer/counter.

TOCS D5   : TimerO clock source select bit

I = External clock from RA4/TOCKI pin

0 =Internal clock (Fosc/4 from XTAL oscillator)

TOSE D4   : TimerO source edge select bit

I = Increment on H-to-L transition on TOCKI pin

0 =Increment on L-to-H transition on TOCKI pin

PSA D3   : TimerO prescaler assignment bit

I = TimerO clock input bypasses prescaler.

0 = TimerO clock input comes from prescaler output.

TOPS2TOPS1TOPSO   D2DIDO   TimerO prescaler selector

0 0 0 = I :2 Prescale value (Fosc / 4 / 2)

0 0 I = I :4 Prescale value (Fosc / 4 / 4)

0 I 0 = I :8 Prescale value (Fosc / 4 / 8)

0 I I= 1:16 Prescale value (Fosc / 4 / 16)

I 0 0 = I :32 Prescale value (Fosc / 4 / 32)

I 0 I = I :64 Prescale value (Fosc / 4 / 64)

I I 0 =I: 128 Prescale value (Fosc / 4 / 128)

I I I = I :256 Prescale value (Fosc / 4 / 256)

Eg. To find the value for TOCON if we want to program TimerO in 16-bit mode, no prescaler. Use Pic18's Fosc/4 crystal oscillator for the clock source, increment on positive-edge.

TOCON = 0000 1000 16-bit, Fosc/4 clock source, no prescaler, TimerO off

Steps to program 8-bit mode of TimerO

To generate a time delay using TimerO in 8-bit mode, take the following steps:

I. Load the TOCON value register indicating 8-bit mode is selected.

2. Load the TMROL registers with the initial count value.

3. Start the timer.

4. Keep monitoring the timer flag (TMROIF) to see if it is raised. Get out of the loop when TMROIF becomes HIGH.

5. Stop the timer with the instruction "BCF TO CON, TMROON".

6. Clear the TMROIF flag for the next round.

7. Go back to Step 2 to load TMROL again.

3 .(a) Give detailed note on state machines and key switches in PIC microcontroller.

- Key switches are not very fast they can be checked each time around the main loopin a keyswitch subroutine. Recall the looptime 10ms was selected because the maximum keybounce time of mechanical key switches is less than 10 ms.

- Consequently if key switches detects a new key is pressed, it can be assured that next time it is called 10 ms later.

- The press and release of key switchesoccur over an interval of many tens of milliseconds. If key is pressed and releases relatively fast rate of four times per second the switch may be closed for 12 looptimes.

- The key switch subroutine will use the state variable called KEYSTATE to keep track from one call to next of the following tasks.
    - Debounce the keyswitch
    - Determine which key is pressed
    - Take appropriate action once for that press of key
    - Wait for the release of that key

Each time keyswitch is called , if no key has been pressed during last several calls then Keystate will equal to zero. The job of key switch subroutine is to determine whether any new key pressed

If Z=1 a return from keyswitch subroutine occurs & If Z=0 a key is newly pressed. So Keystate is incremented to H'01' before returning the subroutine.

Ten ms later keyswitch subroutine is reentered this time with keystate= H'01' if a key is detected last time. By now the key bounce is settled out. A scankey subroutine is called. It returns Z=1 and a keycode ram variable loaded with a value that identifies the pressed key.

For keycode value 0,1,2,3,…15 corresponding table is linked with 4 upper bits of PORTB to identify it. If the value of key pressed matches with key code it returns Z=1 , if not it returns Z=0

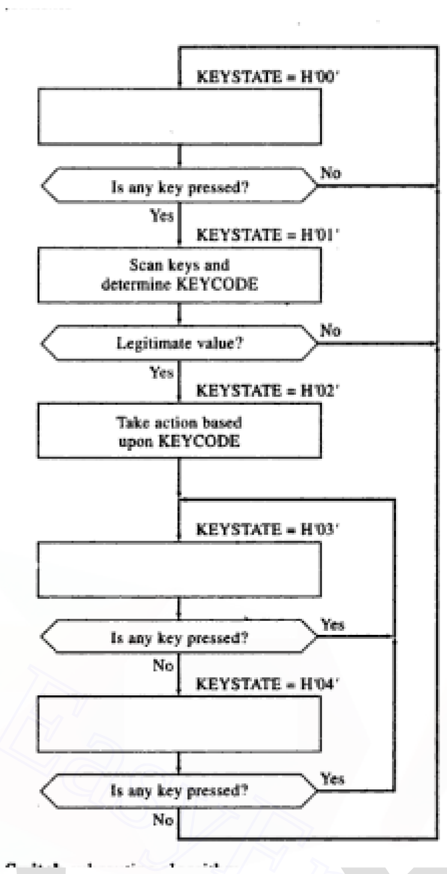


Fig: Flowchart of keyswitches

3.(b).With a simple program explain the concept of timer in detail.

Reading 16bit of free running Timer 1

```
movf TMR1H          ;       read high byte
movwf TMPH          ;       store in TMPH
movf TMR1L          ;       read low byte
movwf TMPL          ;       store in TMPL
movf TMR1H, W       ;       read high byte in W
subwf TMPH, W       ;       subtract 1 st read with 2 nd read
btfsc STATUS, Z     ;       and check for equality
goto next ; if the high bytes differ, then there is an overflow  read the high byte again
followed by the low byte
movf TMR1H, W       ;       read high byte
movwf TMPH
movf TMR1L, W       ;       read low byte
```

```
        movwf TMPL
        next : nop
```

4. (a) Assuming that XTAL = 10 MHz, write a program to tum on pin PORTB4 when TMR2 reaches value I 00 (decimal).

Solution:

Because XTAL = 10 MHz, TMR2 counts up every 0.4 microsecond.

TMR2H = PR2 = 100, PORTB4 will be turned on.

```
BCF  TRISB,4        ;        make  PORTB4  an output
BCF  PORTB,4        ;        turn off PORTB4
MOVLW OxO           ;        Timer2,  no prescale or post scale
MOVWF T2CON       ;load T2CON reg
MOVLW OxO           ;        TMR2  =  0
MOVWF TMR2          ;        load Timer2
MOVLW D'l00'        ;        PR2  =  100, the period register
MOVWF PR2           ;        load PR2
BCF  PIRl,TMR2IF                  ;        clear timer interrupt flag
BSF  T2CON,TMR20N                 ;        start Timer2
AGAIN BTFSS PIRl,TMR2IF          ;monitor Timer2 flag
BRA  AGAIN
BSF  PORTB,4                     ;turn on PORTB4
BCF  T2CON,TMR20N                 ;stop Timer2
HERE  BRA  HERE
```

4.(b)Write a program to create a delay of 1 sec using timer 0.

```
void InitUsart(void) {
     // TX Pin - output
     TRISC6 = 0;
     // RX Pin - input
     TRISC7 = 1;
     // RX Setting, 8bit, enable receive,
     RCSTA = 0x90;
     // TX Setting, 8bit, Asinchronius mode, High speed
     TXSTA = 0x24;
```

```
   // Set Baudrade - 9600 (from datasheet baudrade table)
   SPBRG = 129;
}

#define high_start 76
int seconds, high_count;
//#INT_RTCC
  void interrupt isr(void) {
   high_count -= 1;

  if(high_count==0) {
      ++seconds;
      high_count=high_start;
   }
}
void main() {                              //A simple stopwatch program
   int start, time;
   high_count = high_start;
     T0CS = 0;
    TMR0IE = 1;    // Enable interrupt on TMR0 overflow
    T0SE = 0;
     PSA  = 0;
     PS2  = 1;
     PS1  = 1;
     PS0  = 1;
    TMR0IF = 0;  // clear the overflow bit
    GIE=1;   // enable_interrupts(GLOBAL);
    PEIE=1;
   do {
      printf("Press any key to begin.nr");
     getc();
      start = seconds;
```

```
        printf("Press any key to stop.rn");
        getc();
        time = seconds - start;
        printf("%c seconds.nr", time);
    } while (1);
}
```

5. Explain how to set Timer1 and Timer 2 in PIC .

Timer1 programming

Timer1 is a 16-bit timer, and its 16-bit register is split into two bytes, referred to as TMRlL (Timerl low byte) and TMRlH (Timerl high byte). Timerl can be programmed in 16-bit mode only and unlike TimerO, Timer! also has the Tl CON (Timer 1 control) regis-ter in addition to the TMR1IF (Timerl interrupt flag). The TMR1IF flag bit goes HIGH when TMRlH:TMRlL overflows from FFFF to 0000. Timerl also has the prescaler option, but it only supports factors of 1:1, 1:2, 1:4, and 1:8. The P1Rl register contains the TMR1IF flags.

RD16	-	T1CKPS2	T1CKPS1	T1OSCEN	T1SYNC	TMR1CS	TMR1ON

Fig: Timer 1 TCON register

RD16 D7 : 16-bit read/write enable bit

I = Timer! 16-bit is accessible in one 16-bit operation.

0 =Timer! 16-bit is accessible in two 8-bit operations.

D6 Not used

T1CKPS2:T1CKPS1 D5 D4 Timer1 prescaler selector

0 = Stop Timer I

0 0 = I: I Prescale value

0 I = I :2 Prescale value

I 0 = I :4 Prescale value

I I = I :8 Prescale value

T1OSCEN D3 :Timer1 oscillator enable bit

I = Timer1 oscillator is enabled.

0 = Timer1 oscillator is shutoff.

T1SYNC D2 :Timer I synchronization (used only when TMR I CS = I for counter mode to synchronize external clock input) If TMR I CS = 0 this bit is not used.

TMR1CS Dl : Timer1 clock source select bit

I = External clock from pin RCO/Tl CKI

0 =Internal clock (Fosc/4 from XTAL)

TMR1ON DO : Timer1 ON and OFF control bit

I = Enable (start) Timer!

Timer 0: 8-bit timer/counter with 8-bit prescalar

Timer 1: 16-bit timer/counter with prescalar, can be incremented during sleep via external crystal/clock.

Timer 2: 8-bit timer/counter with 8-bit period register, prescalar, post scalar.

Timer 2 is an 8-bit timer with a prescalar and a port scalar. It can be used on the PWM mode of CCP modules.

The output of TMR2 goes through a 4-bit post scalar (1:1, 1:2 to 1:16) to generate a TMR2 interrupt by setting TMR2IF flag.

-	TOUTPS3	TOUTPS2	TOUTPSI	TOUTP	TMR20	T2CKPSI	T2CKPSO

D7	Not used
TOUTPS3:TOUTPSO D6D5D4D3	Timer2 Output Postcale Select bits

00 0 0 = 1 :I Postscale value

00 0 I = I :2 Postscale value

00 I 0 = I :3 Postscale value

00 I I = I :4 Postscale value

II I 0 = I: 15 Postscale value

II I I = I: 16 Postscale value

TMR20N	D2	Timer2 ON and OFF Control bit
	I = Enable (Start) Timer2	
	0 = Stop Timer2	
T2CKPS1:T2CKPSO	DIDO	Timer2 Clock Prescale Select bits
	0 0 = Prescale is I	
	0 I = Prescale is 4	

UNIT-3

PHERIPHERALS & INTERFACING

Part-A

1.what is I2C bus?

The I2c bus is a bidirectional two wire serial bus that provides a communication link between integrated circuits.

2.List the three data transfer speed levels in I2C.

- Standard mode-100Kbps
- Fast mode-400Kbps
- High speed mode-3.4 Kbps

3. Define the term glitch in ADC.

It's defined as when a high speed ADC fail to give an output correctly within its conversion time and gives the missing 1s in between. It is simply said as appearance of false output in ADC is known as glitch.

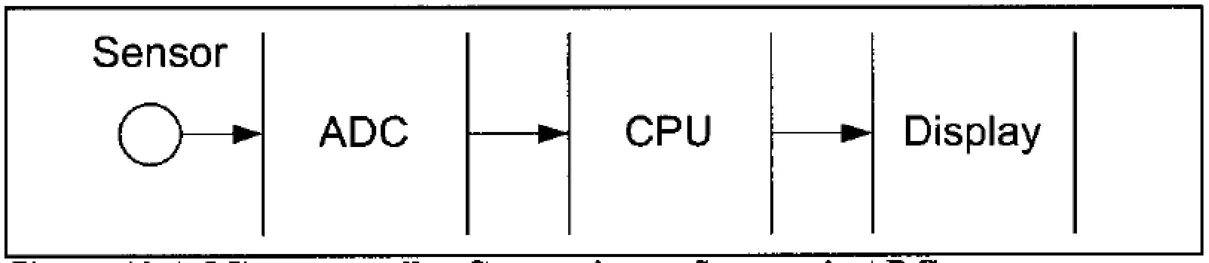
4. Mention the delay types occurred in interfacing LCD with PIC.

- L-Delay(Long Delay)
- S-Delay(Short Delay)

5. Write the steps involved in to read busy flag.

- Select command register
- Select read operation
- Send enable signal
- Read the flag

6.Draw the block diagram for interfacing a sensor with PIC.



7.Write the formula used to calculate the digital output in ADC.

$D_{OUT} = V_{IN} / STEP\ SIZE$

Where; D_{OUT} =Digital output

V_{IN} =Input Voltage

8.What is the purpose of busy flag in LCD.

In LCD there is a need of some delay to process the command or data this can be done by using busy flag by making RS=0

9.List the two processes used to interface keypad with PIC.

- Key Press detection
- Key Identification

10.List the registers used to program an ADC.

1. ADCON0(A/D Control Register 0)
2. ADCON1(A/D Control register 1)

11.For a PIC18 based system we have V_{ref} =V_{dd} =5V. Find the step size.

Step size=5/1024=4.8mv

12.Assuming that R=5KΩ and I_{ref} =2mA calculate V_{OUT} for the binary input 10011001.

I_{out} =2mA(153/256)=1.195mA

V_{out} =1.195mA*5K=5.975

PART-B

1.Explain briefly about LCD interfacing with PIC microcontroller.

LCD operation:

In recent years the LCD has been finding widespread use replacing LED's (seven-segment LEDs or other multisegment LEDs). This is due to the following reasons:

I. The declining prices of LCDs.

2. The ability to display numbers, characters, and graphics. This is in contrast to LEOs, which are limited to numbers and a few characters.

3. Incorporation of a refreshing controller into the LCD, thereby relieving the CPU of the task of refreshing the LCD. In contrast, the LED must be refreshed by the CPU (or in some other way) to keep displaying the data.

4. Ease of programming for characters and graphics.

LCD pin descriptions: The LCD discussed in this section has 14 pins.

Vcc• Vss, And VEE:

While Vcc and Vss provide +5 V and ground, respectively, VEE is used for controlling LCD contrast.

RS, Register Select:

There are two very important registers inside the LCD. The RS pin is used for their selection as follows. If RS = 0, instruction command code register is selected, allowing the user to send a command such as clear display, cursor at home, and so on. If RS = 1 the data register is selected, allowing the user to send data to be displayed on the LCD.

R/W; Read/Write:

R/W input allows the user to write information to the LCD or read information from it. R/W = 1 when reading; R/W = 0 when writing.

E, enable:

The LCD to latch information presented to its data pins uses the enable pin. When data is supplied to data pins, a high-to-low pulse must be applied to the En pin in order for the LCD to latch in the data present at the data pins. This pulse must be a minimum of 450 ns wide. Here we call this delay the SDELAY (short delay) to distinguish it from other delays.

PIN DESCRIPTION FOR LCD

Pin	Symbol	I/O port	Description
1	Vss	-	GND
2	Vcc	-	+ 5V Power Supply
3	VEE	-	Power Supply to Control Contrast

4	RS	I	RS =0 To Select Command Register,
			RS=1 To Select Data Register

5	R/W	I	R/W =0 for write, R/W =0 for read
6	E	I/O	Enable
7	DBO	I/O	The 8-bit data bus
8	DB1	I/O	The 8-bit data bus
9	DB2	I/O	The 8-bit data bus
10	DB3	I/O	The 8-bit data bus
11	DB4	I/O	The 8-bit data bus
12	DB5	I/O	The 8-bit data bus
13	DB6	I/O	The 8-bit data bus
14	DB7	I/O	The 8-bit data bus

D0-07:

The 8-bit data pins, DO-D7, are used to send information to the LCD or read the contents of the LCD's internal registers.

To display letters and numbers, we send ASCII codes for the letters A-Z, a-z, and numbers 0-9 to these pins while making RS = 1.

There are also instruction command codes that can be sent to the LCD to clear the display.

The below Table contains lists of instruction command codes. To send any of the commands listed in the Table to the LCD, make pin RS = 0. For data, make RS = 1. Then send a high-to-low pulse to the E pin to enable the internal latch of the LCD.

There are two ways to send characters (command/data) to the LCD:

(1) Use a delay before sending the next one,

(2) Use the busy flag to see if the LCD is ready for the next one

LCD COMMAND CODES

Code	**Command to LCD Instruction**
1	Clear dtsplay screen
2	Return home
4	Decrement cursor (shift cursor to left)
5	Increment cursor (shift cursor to right)
6	Shift display right
7	Shift display left
8	Display off, cursor off
A	Display off, cursor on
C	Display on, cursor off
E	Display on, cursor blinking
F	Display on, cursor blinking
10	Shift cursor position to left
14	Shift cursor position to right
18	Shift the entire display to the left
1C	Shift the entire display to the right
80	Force cursor to begin the 1st line
C0	Force cursor to beginning of 2nd line
38	2 lines and 5 x 7 matrix

Sending commands and data to LCDs with a time delay:

The Program shows how to send characters (command/data) to the LCD without checking the busy flag. Here 5-10 ms DELAY is mentioned between issuing each character to the LCD. This is know as simply DELAY. In programming an LCD, we can also mention a long delay for the power-up process. This is known as **LDELAY** (long delay). **SDELAY** (short delay) is used to make the **En(Enable)** signal wide enough for the LCD's enable input. The below figure shows the LCD connections to the microcontroller.

Sending command or data to the LCD using busy flag:

We use RS = 0 to read the busy flag bit to see if the LCD is ready to receive information. The busy flag is D7, and can be read when *R/W* = 1 and RS = 0, as follows:

If R/W = 1, RS = 0. When D7 = 1 (busy flag= 1), the LCD is busy taking care of internal operations and will not accept any new information. When D7= 0, the LCD is ready to receive new information.

The busy flag is D7 of the command register. To read the command register, we make R/W =1 and RS = 0, and a L-to-H pulse for the E pin will provide us the command register.

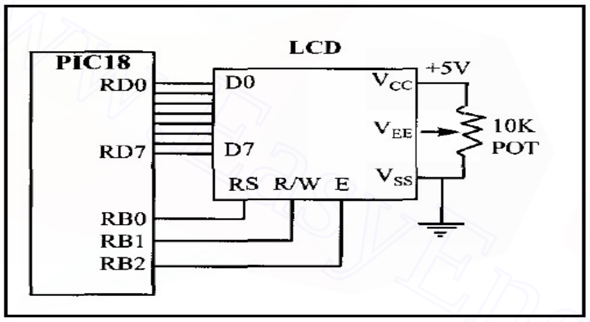


Fig : LCD interfacing

After reading the command register, if bit D7 (the busy flag) is HIGH, the LCD is busy and no information(command or data) should be issued to it. Only when D7 = 0 can we send data or commands to the LCD.

Notice that no time delays are used in this method because we are checking the busy flag before issuing commands or data to the LCD.

2.Explain briefly about Keyboard interfacing with PIC microcontroller.

At the lowest level, keyboards are organized in a matrix of rows and columns. The CPU accesses both rows and columns through ports; therefore, with two 8-bit ports, an 8 x 8 matrix of keys can be connected to a microprocessor.

When a key is pressed, a row and a column make a contact; otherwise, there is no connection between rows and columns. In IBM PC keyboards, a single microcontroller takes care of hardware and software interfacing of the keyboard. In such systems, programs stored in the ROM of the microcontroller scan the keys continuously, identify which one has been activated, and present it to the motherboard.

For keypad interfacing we must have two processes:

(a) key press detection,

(b) key identification.

There are two ways by which the PICJ8 can perform key press detection:

(1) the interrupt method,

(2) the scanning method.

Interrupt method of key press detection:

Figure(c) shows a 4 x 4 matrix keypad connected to PORTB. The rows are connected to PORTB. Low (RB3-RBO) and the columns are connected to PORTB. High (RB7-RB4), which is the PORTB-Change interrupt.

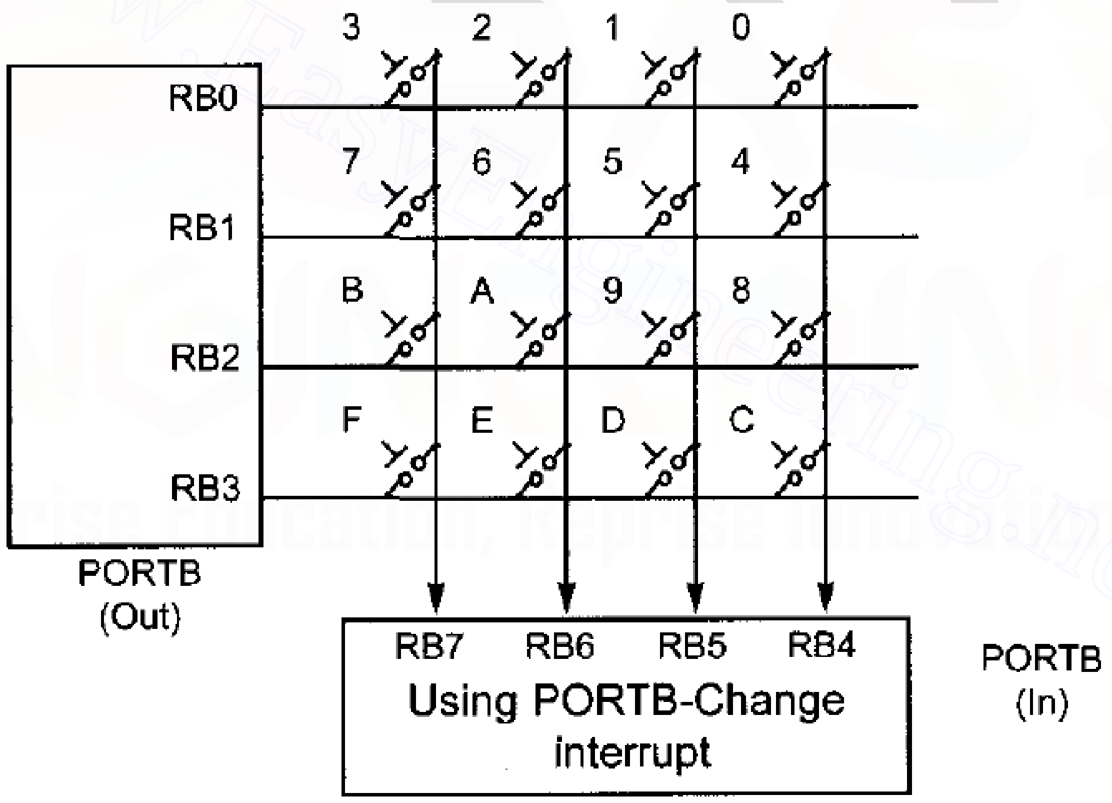


Fig: Keypad

Any changes on the RB7-RB4 pins will cause an interrupt indicating a key press. Which goes through the following stages:

1. To make sure that the preceeding key has been released, O's are output to all rows at once, and the columns are read and checked repeatedly until all the columns are HIGH. When all columns are found to be HIGH, the program waits for a short amount of time before it goes to the next stage of waiting for a key to be pressed.

2. To see if any key is pressed, the columns are connected to the PORTB-Change interrupt. Therefore, any key press will cause an interrupt and the microcontroller will execute the ISR. The ISR must do two things:

(a) ensure that the first key press detection was not erroneous due to spike noise,

(b) wait 20ms to prevent the same key press from being interpreted as multiple key Presses.

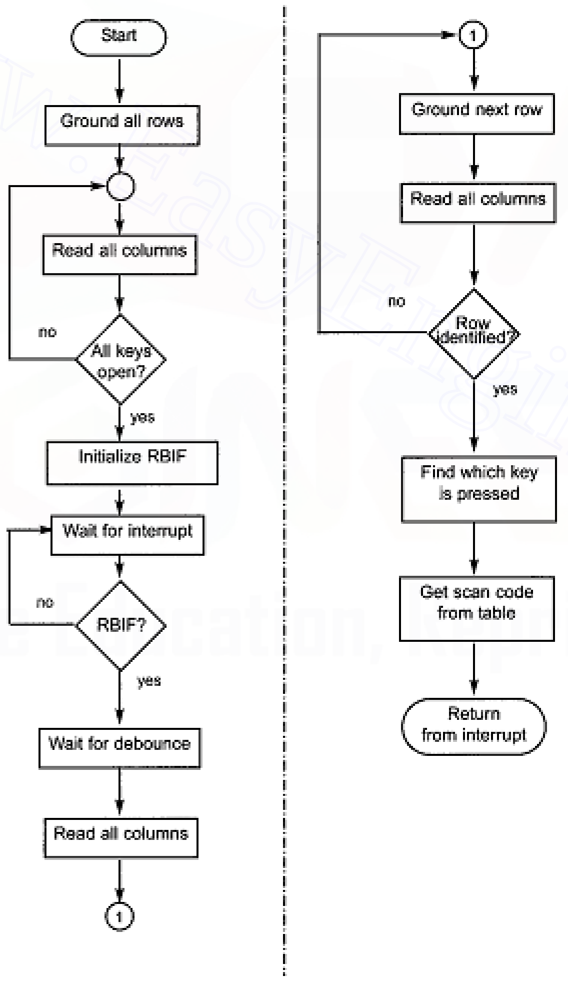


Fig: flow chart of Keypad interfacing

3. To detect which row the key press belongs to, the microcontroller grounds one row at a time, reading the columns each time.

If it finds that all columns are HIGH, this means that the key press cannot belong to that row; therefore, it grounds the next row and continues until it finds the row the key press belongs to.

Upon finding the row that the key press belongs to, it sets up the starting address for the look-up table holding the scan codes (or the ASCll value) for that row and goes to the next stage to identify the key.

4. To identify the key press, the microcontroller rotates the column bits, one bit at a time, into the carry flag and checks to see if it is LOW.

Upon finding the zero, it pulls out the ASCII code for that key from the look-up table; otherwise, it increments the pointer to point to the next element .

Scanning method for key press detection:

Another method for key press detection is by scanning. In this method, to detect a pressed key, the microcontroller grounds all rows by providing 0 to the output latch, then it reads the columns.

If the data read from the columns are equal to 1111, no key has been pressed and the process continues until a key press is detected.

If one of the column bits has a zero, however, this means that a key press has occurred. After a key press is detected, the microcontroller will go through the process of identifying the key.

Starting with the top row, the microcontroller grounds it by providing a LOW to the first row only; then it reads the column.

3.Explain the Mechanism in PIC for interfacing ADC & DAC.

<u>ADC INTERFACING:</u>

The following steps must be followed for data conversion by an ADC chip:

1. Select a channel.
2. Activate the start conversion (SC) signal to start the conversion of analog input.
3. Keep monitoring the end-of-conversion (EOC) signal.
4. After the EOC has been activated, we read data out of the ADC chip.

PIC18F452/458 ADC features:

The ADC peripheral of the PIC 18 has the following characteristics:

(a) It is a 10-bitADC.

(b) It can have 5 to 15 channels of analog input channels, depending on the family member. In PIC18452/458, pins RAO--RA7 of PORTA are used for the 8 analog channels. See Figures A and B.

c) The converted output binary data is held by two special function registers called ADRESL *(*A/D Result Low) and ADRESH (A/D Result High).

(d) Because the ADRESH:ADRESL registers give us 16 bits and the ADC data out is only 10 bits wide, 6 bits of the 16 are unused. We have the option of making either the upper 6 bits or the lower 6 bits unused.

(e) We have the option of using Vdd (Vee), the voltage source of the PICI8 chip itself, as the Vref or connecting it to an external voltage source for the Vref.

(f) The conversion time is dictated by the F osc of crystal frequency connected to the OSCs pins. While the Fosc for PICI8 can be as high as 40 MHz, the conversion time can not be shorter than 1.6 ms.

(g) It allows the implementation of the differential Vref voltage using the Vref(+) and Vref(-) pins, where Vref= Vref(+)- Vref(-).

Many of the above features can be programmed by way of ADCONO (A/D control register 0) and ADCON1 *(*A/D control register 1)

ADCON1 register:

Another major register of the PICI8's ADC feature is ADCON1. The ADCON1 register is used to select the Vref voltage among other things. After the A/D conversion is complete, the result sits in registers ADRESL (A/D Result Low Byte) and ADRESH *(*A/D Result High Byte). The ADFM bit of the ADCONI is used for making it right-justified or left-justified because we need only 10 bits of the 16.

ADCONO register:

The ADCONO register is used to set the conversion time and select the analog input channel among other things.. In order to reduce the power consumption of the PIC 18, the ADC feature is turned off when the microcontroller is powered up. We turn on the ADC with the ADON bit of the ADCONO register. The other important bit is the GO/DONE bit. We use this bit to start conversion and monitor it to see if conversion has ended. Notice in ADCCONO that not all family members have all the 8 analog input channels. The conversion time is set with the ADCS bits. **Calculating A/D conversion time:**

By using the ADCS *(A/D* clock source) bits of both the ADCONO and ADCON1 registers we can set the *A/D* conversion time. The conversion time is defined in terms of Tad, where Tad is the conversion time per bit. To calculate the Tad, we can select a conversion clock source ofFosc/2, Fosc/4, Fosc/8, Fosc/l6, Fosc/32, or Fosc/64, where Fosc is the speed of the crystal frequency connected to the PIC18 chip. For the PIC18, the conversion time is 12 times the Tad. Notice thatthe Tad cannot be faster than 1.6 ms.

Steps in programming the A/D converter using polling:

To program the A/D converter of the PIC 18, the following steps must be taken:

1. Turn on the ADC module of the PICI8 because it is disabled upon power-on
reset to save power. We can use the "BSF ADCONO, ADON" instruction.
2. Make the pin for the selected ADC channel an input pin. We use "BSF
TRISA, x." or "BSF TRISE, x" where x is the channel number.
3. Select voltage reference and A/C input channels. We use registers ADCONO
and ADCONl.
4. Select the conversion speed. We use registers ADCONO and ADCONI.
5. Wait for the required acquisition time.
6. Activate the start conversion bit of GO/DONE.
7. Wait for the conversion to be completed by polling the end-of-conversion (GO/DONE) bit.
8. After the GO/DONE bit has gone LOW, read the ADRESL and ADRESH registers
to get the digital data output.
9. Go back to step 5.

Programming A/D converter using interrupts:

To program the A/D using the interrupt method,we need to set HIGH the ADIE (A/D interrupt enable) flag. If ADIE = 1, then upon the completion of the conversion, the ADIF (A/D interrupt flag) becomes HIGH, which will force the CPU to jump to read binary outputs.

<u>DAC INTERFACING:</u>

The digital-to-analog converter (DAC) is a device widely used to convert
digital pulses to analog signals. In this section we discuss the basics of interfacing DAC .

The vast majority of integrated circuit DACs, including the MC1408 (DAC0808) used here, use the R/2R method because it can achieve a much higher degree of precision. The first criterion for judging a DAC is its resolution, which is a function of the number of binary inputs. The common ones are 8, 1O, and 12 bits. The number of data bit inputs decides the resolution of the DAC because the number of analog output levels is equal to 2^n, where n is the number of data bit inputs. Therefore, an 8-input DAC such as the DAC0808 provides 256 discrete voltage (or current) levels of output. Similarly, the 12-bit DAC provides 4,096 discrete voltage levels.There are also 16-bit DACs.

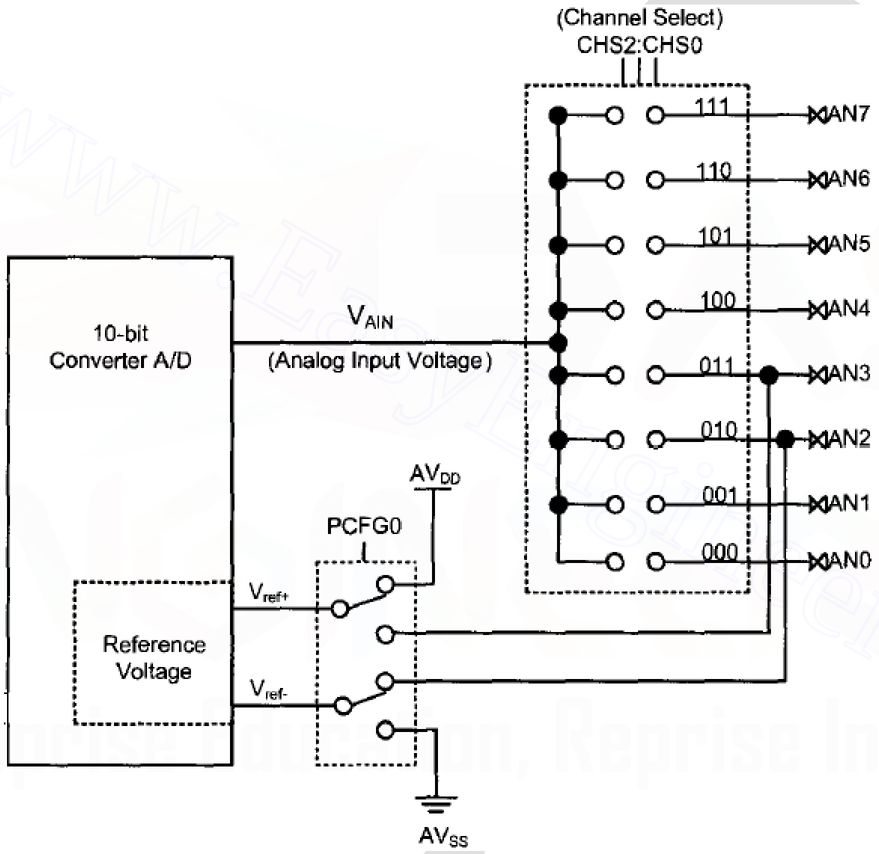


Fig PIC18 ADC Channel and Reference Selection

MC1408 DAC (or DAC0808):

In the MC1408 (DAC0808), the digital inputs are converted to current (I_{out}), and by connecting a resistor to the I_{out} pin, we convert the result to voltage.

The total current provided by the I_{out} pin is a function of the binary numbers at the DO-D7 inputs of the DAC0808 and the reference current (I_{ref}), and is as follows:

$$I_{out} = I_{ref}\left(\frac{D7}{2} + \frac{D6}{4} + \frac{D5}{8} + \frac{D4}{16} + \frac{D3}{32} + \frac{D2}{64} + \frac{D1}{128} + \frac{D0}{256}\right)$$

where DO is the LSB, D7 is the MSB for the inputs, and I_{ref} is the input current that must be applied to pin 14. The I_{ref} current is generally set to 2.0 mA. Figure below shows the generation of current reference (setting I_{ref}= 2 mA) by using the standard 5 V power supply. Now assuming that I_{ref}= 2 mA, if all the inputs to the DAC are high, the maximum output current is 1.99mA .

Converting I_{out} to voltage in DAC0808:

Ideally we connect the output pin lout to a resistor, convert this current to voltage, and monitor the output on the scope. In real life, however, this can cause in accuracy because the input resistance of the load where it is connected will also affect the output voltage. For this reason, the I_{ref} current output is isolated by connecting it to an op-amp such as the 741 with Rr= 5 kOhms for the feedback resistor.

Generating a sine wave:

To generate a sinewave, we first need a table whose values represent the magnitude of the sine of angles between 0 and 360 degrees. The values for the sine function vary from -1.0to + 1.0 for 0- to 360-degree angles. Therefore, the table values are integer numbers representing the voltage magnitude for the sine of theta. This method ensures that only integer numbers are output to the DAC by the PIC 18 microcontroller. Full-scale output of the DAC is achieved when all the data inputs of the DAC are HIGH. Therefore, to achieve the full-scale 10 V output, we use the following equation.

$$V_{out} = 5\text{ V} + (5 \times \sin\theta)$$

To find the value sent to the DAC for various angles, we simply multiply the V_{out} voltage by 25.60 because there are 256 steps and full-scale V_{out} is 10volts.

Therefore, 256 steps / 10 V = 25.6 steps per volt.

4.Write short notes about temperature sensor and Explain how it is interfaced with PIC18 series microcontroller.

Temperature sensors:

Transducers convert physical data such as temperature, light intensity, flow, and speed to electrical signals. Depending on the transducer, the output produced is in the form of voltage, current, resistance, or capacitance. For example, temperature is converted to electrical signals using a transducer called a ***thermistor****.* A thermistor responds to temperature change by changing resistance, but its response is not linear. Simple and widely used linear temperature sensors include the LM34 and LM35 series from National Semiconductor Corp.

LM34 and LM35 temperature sensors:

The sensors of the LM34 series are precision integrated-circuit temperature sensors whose output voltage is linearly proportional to the Fahrenheit temperature. The LM34 requires no external calibration because it is internally calibrated. It outputs 10 mV for each degree of Fahrenheit temperature.

The LM35 series sensors are precision integrated-circuit temperature sensors whose output voltage is linearly proportional to the Celsius (centigrade) termperature. The LM35 requires no external calibration because it is internally calibrated. It outputs 10 m V for each degree of centigrade temperature.

Signal conditioning and interfacing the LM35 to the PIC18

Signal conditioning is widely used in the world of data acquisition. The most common transducers produce an output in the form of voltage, current, charge,capacitance, and resistance. We need to convert these signals to voltage, however, in order to send input to an A-to-D converter.

This conversion (modification) is commonly called *signal conditioning.* Signal conditioning can be a current to voltage conversion or a signal amplification. For example, the thermistor changes resistance with temperature.

The change of resistance must be translated into voltages in order to be of any use to an ADC. Look at the case of connecting an LM34 to an ADC of the PIC18F458. The A/D has 10-bit resolution with a maximum of 1,024 steps and the LM34 (or LM35) produces 10 m V for every degree of temperature change.

Now, if we use the step size of 10 mV, the Vout will be 10,240 mV (10.24 V) for full-scale output. This is not acceptable even though the maximum temperature sensed by the LM34 is 300 degrees F, and the highest output for the A/D we will get is 3,000 m V

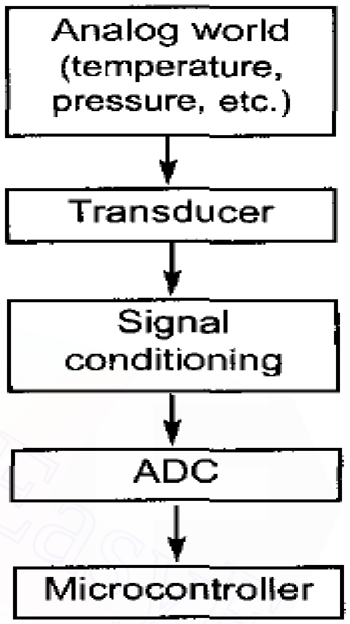


Fig. Getting data from analog world

Now, if we use the step size of2.5 mV, the V001 will be 1,024 x 2.5 mV= 2,560 mV (2.56 V) for full-scale output. That means we must set Vref= 2.56 V. This makes the binary output number for the A/D 4 times the real temperature (10mV/2.5 m V = 4). We can scale it by dividing it by 4 to get the real number for temperature.

Notice that we use the LM336-2.5 zener diode to fix the voltage across the 10 K pot at 2.5 volts. The use of the LM336-2.5 should overcome any fluctuations in the power supply.

Reading and displaying temperature:

For reading and displaying an ALP program is given below, in this program following points are to be noted;

(1) The LM34 (or LM35) is connected to channel 0 (RAO pin).

(2) The channel AN3 (RA3 pin) is connected to the Vref of 2.56 V. That makes PCFG = 0010 for the ADCONl register.

(3) The 1O-bit output of the A/D is divided by 4 to get the real temperature.

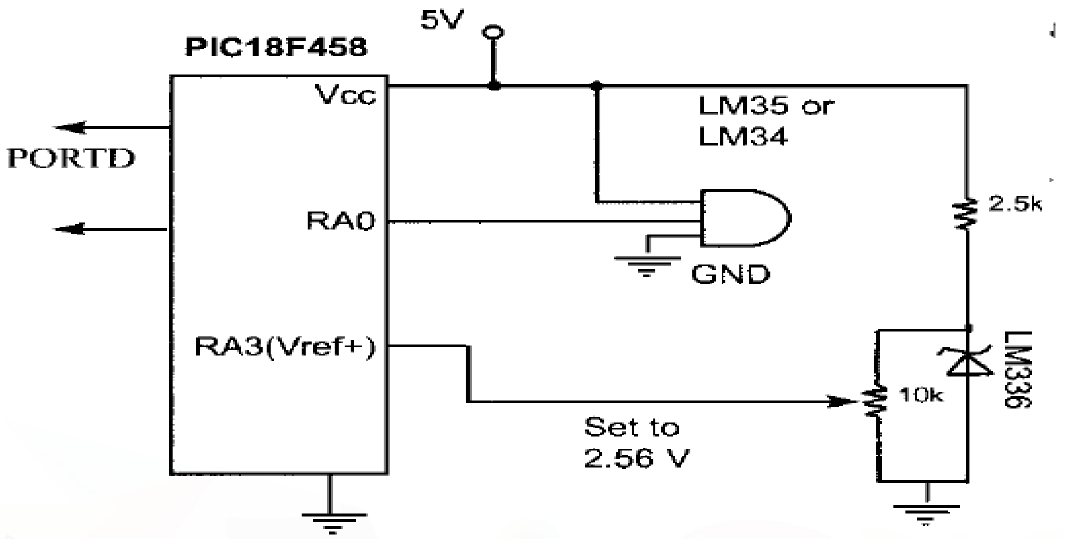


Fig . PIC18F458 Connection to Temperature Sensor

5. Explain the concept Of I^2C Bus In PIC Microcontroller.

I²C (pronounced I-squared-C) created by Philips Semiconductors and commonly written as 'I2C' stands for Inter-Integrated Circuit and allows communication of data between I2C devices over two wires. It sends information serially using one line for data (SDA) and one for clock (SCL).

Master and slave

The I2C protocol defines the concept of master and slave devices. A master device is simply the device that is in charge of the bus at the present time and this device controls the clock and generates START and STOP signals. Slaves simply listen to the bus and act on controls and data that they are sent.

The master can send data to a slave or receive data from a slave - slaves do not transfer data between themselves.

Multi Master

Multi master operation is a more complex use of I2C that lets you have different controlling devices on the same bus. You only need to use this mode if you have more than one microcontroller on the bus (and you want either of them to be the bus master).

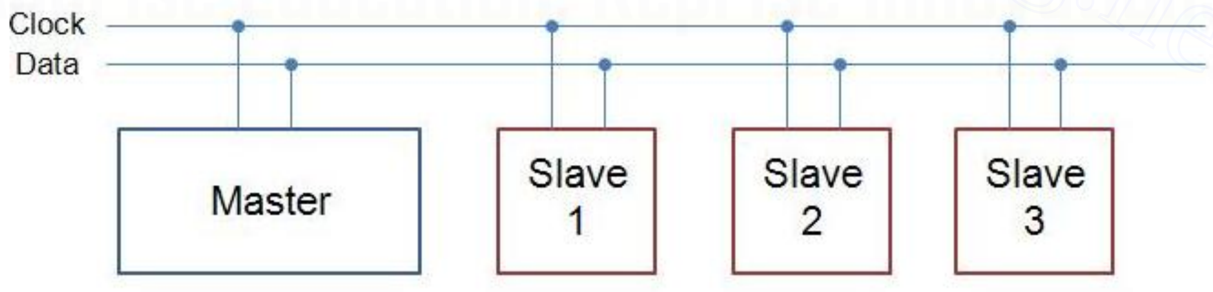
Multi master operation involves arbitration of the bus (where a master has to fight to get control of the bus) and clock synchronisation (each may a use a different clock e.g. because of separate crystal clocks for each micro).

Data and Clock

The I2C interface uses two bi-directional lines meaning that any device could drive either line. In a single master system the master device drives the clock most of the time - the master is in charge of the clock but slaves can influence it to slow it down The two wires must be driven as open collector/drain outputs and must be pulled high using one resistor each - this implements a 'wired AND function' - any device pulling the wire low causes all devices to see a low logic value - for high logic value all devices must stop driving the wire.

Speed

Standard clock speeds are 100 kHz and 10 kHz but the standard lets you use clock speeds from zero to 100 kHz and a fast mode is also available (400 kHz - Fast-mode). An even higher speed (3.4MHz - High-speed mode) for more demanding applications .



Block diagram of an i2c configuration.
Multiple integrated circuits are aligned on a two-wire bus. Assigning unique addresses allows a master device to control multiple slave devices.

Fig:I2C bus

Slow peripherals

A slow slave device may need to stop the bus while it gathers data or services an interrupt etc. It can do this while holding the clock line (SCL) low forcing the master into the wait state. The master must then wait until SCL is released before proceeding.

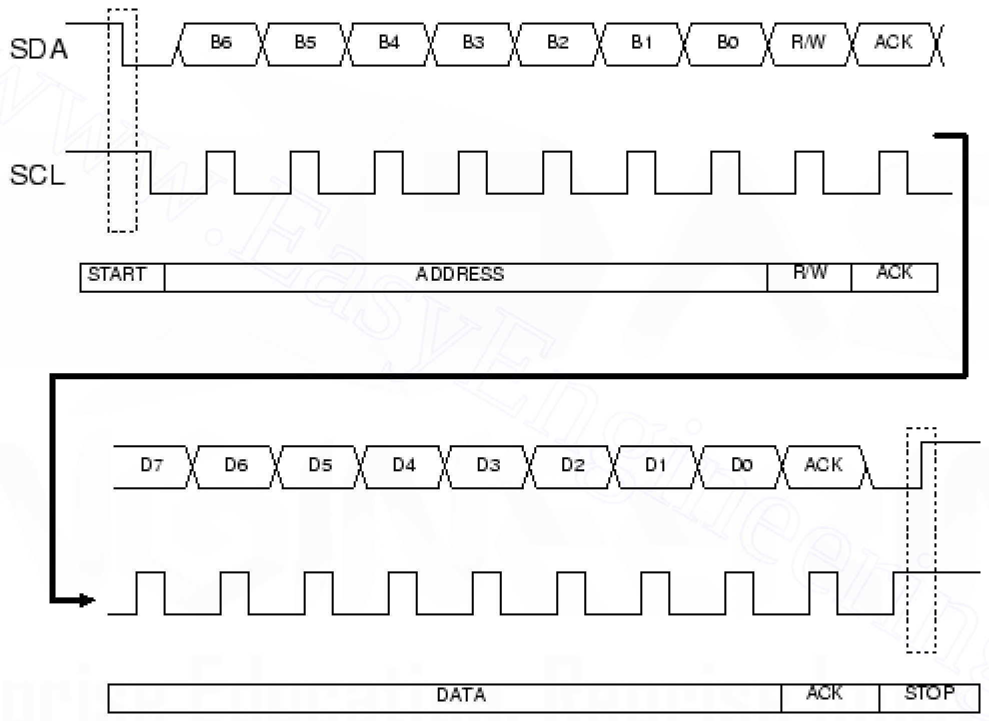


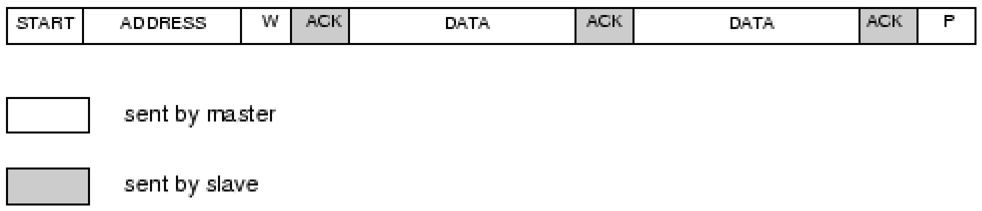
Fig:I2C clock pulses

Data transfer sequence

A basic Master to slave read or write sequence for I2C follows the following order:

Data Transfer from master to slave

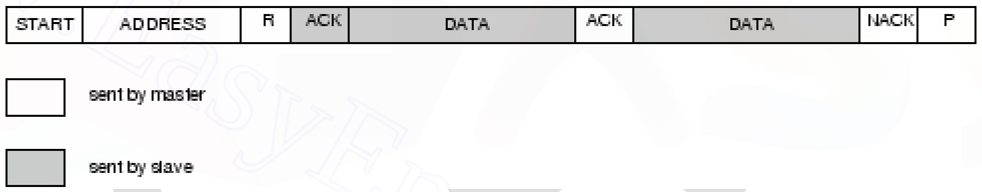
Instruction sequence data from master to slave



A master device sends the sequence S ADDR W and then waits for an acknowledge bit (A) from the slave which the slave will only generate if its internal address matches the value sent by the master. If this happens then the master sends DATA and waits for acknowledge (A) from the slave. The master completes the byte transfer by generating a stop bit (P) (or repeated start).

Data transfer from slave to master

Instruction sequence data from slave to master



A similar process happens when a master reads from the slave but in this case, instead of W, R is sent. After the data is transmitted from the slave to the master the **master** sends the acknowledge (A). If instead the master does not want any more data it must send a not-acknowledge which indicates to the slave that it should release the bus. This lets the master send the STOP or repeated START signal.

START (S) and STOP (P) bits

START (S) and STOP (P) bits are unique signals that can be generated on the bus but **only** by a bus master. Reception of a START bit by an I2C slave device resets its internal bus logic. This can be done at any time so you can force a restart if anything goes wrong even in the middle of communication. START and STOP bits are defined as rising or falling edges on the data line while the clock line is kept high.

Text definition of START and STOP signals

START condition (S)	SCL = 1, SDA falling edge
STOP condition (P)	SCL = 1, SDA rising edge

Repeated START (Sr)

This seems like a confusing term at first as you ask yourself why bother with it as it is functionally identical to the sequence :

S ADDR (R/W) DATA A P

The only difference is that for a repeated start you can repeat the sequence starting from the stop bit (replacing the stop bit with another start bit).

S ADDR (R/W) DATA A **Sr ADDR (R/W) DATA A P**

The main reason that the Sr bit exists is in a multi master configuration where the current bus master does not want to release its mastership. Using the repeated start keeps the bus busy so that no other master can grab the bus.

Data

All data blocks are composed of 8 bits. The initial block has 7 address bits followed by a direction bit (Read or Write). Following blocks have 8 data bits. Acknowledge bits are squeezed in between each block. Each data byte is transmitted MSB first including the address byte.

Acknowledge

The acknowledge bit (generated by the receiving device) indicates to the transmitter that the the data transfer was ok. Note that the clock pulse for the acknowledge bit is always created by the bus master.

ACK data master --> slave

In this case the slave generates the acknowledge signal.

ACK data slave --> master

In this case the master generates the acknowledge signal.

UNIT -4

INTRODUCTION TO ARM PROCESSOR

1. What are the features in Berkeley RISC designs rejected by ARM designers?

The features rejected by ARM designers are

- Register Windows
- Delayed branches
- Single cycle execution of all instructions

2. What are the basic components of a Processor?

The basic components of processor are a program counter (PC), an accumulator, an arithmetic-logic unit (ALU) and an instruction register (IR)

3. What are the various types of instructions?

The various types of instructions are

- Data processing instructions (add, subtract and multiply)
- Data movement instructions (copy data from one place in memory to another)
- Control flow instructions that (switch execution from one part of the program to another)
- Special instructions (control the processor's execution state)

4. What are the various addressing modes in Processors?

1. Immediate addressing 2. Absolute addressing 3. Indirect addressing 4. Register addressing 5. Register indirect addressing 6. Base plus offset addressing 7. Base plus index addressing 8. Base plus scaled index addressing 9. Stack addressing

5. Give the sequence of Pipeline in ARM

1. Fetch the instruction from memory (fetch). 2. Decode it to see what sort ofinstruction it is (dec). 3. Access any operands that may be required from the register bank (reg). 4. Combine the operands to form the result or a memory address (ALU). 5. Access memory for a data operand, if necessary (mem). 6. Write the result back to the register bank (res).

6.What are the advantages of RISC processor?

• A smaller die size.

• A shorter development time.

• A higher performance.

7. What are the various level of accuracy in ARMULATOR?

• Instruction-accurate modeling

• Cycle-accurate modeling

• Timing-accurate modeling

8. What are the Data Transfer Instructions in ARM?

• Single register load and store instructions.

• Multiple register load and store instructions.

• Single register swap instructions.

9.Define ASR, ROR and RRX.

ASR: arithmetic shift right by 0 to 32 places; fill the vacated bits at the most significant end of the word with zeros if the source operand was positive, or with ones if the source operand was negative.

• ROR: rotate right by 0 to 32 places; the bits which fall off the least significant end of the word are used, in order, to fill the vacated bits at the most significant end of the word.

• RRX: rotate right extended by 1 place; the vacated bit (bit 31) is filled with the old value of the C flag and the operand is shifted one place to the right. With appropriate use of the condition codes (see below) a 33-bit rotate of the operand and the C flag is performed.

10. What are the important features ARM instruction set?

• The load-store architecture • 3-address data processing instructions • conditional execution of every instruction • the inclusion of very powerful load and store multiple register instructions

• Open instruction set extension through the coprocessor instruction set, including adding new registers and data types to the programmer's model

• Very dense 16-bit compressed representation of the instruction set in the Thumb architecture.

11. What are the Drawbacks of RISC

The drawbacks of RISC are

• RISCs generally have poor code density compared with CISCs.

• RISCs don't execute x86 code.

12. What is the various Instruction set design

4-address instructions (ADD d, s1, s2, next_i ; d := s1 + s2)

3-address instructions (ADD d, s1, s2 ; d := s1 + s2)

2-address instructions (ADD d, s1 ; d := d + s1)

1-address instructions (ADD s1 ; accumulator := accumulator + s1)

0-address instructions (ADD ; top_of_stack := top_of_stack + next_on_stack)

PART-B

1. Explain the ARM programmer's model

When writing user-level programs things to be considered in the programmers model are 15 general-purpose 32-bit registers (r0 to r 14), Program Counter (r15) and Current Program Status Register (CPSR) need be considered. The remaining registers are used only for system-level programming and for handling exceptions (for example, interrupts).

(i) Current Program Status Register (CPSR)

The CPSR is used in user-level programs to store the condition code bits. The bits at the bottom of the register control the processor mode, instruction set and interrupt enables

The condition code flags are in the top four bits of the register and have the following meanings:

• N: Negative; the last ALU operation which changed the flags produced a negative result (the top bit of the 32-bit result was a one).

• Z: Zero; the last ALU operation which changed the flags produced a zero result (every bit of the 32-bit result was zero).

• C: Carry; the last ALU operation which changed the flags generated a carry-out, either as a result of an arithmetic operation in the ALU or from the shifter.

• V: oVerflow; the last arithmetic ALU operation which changed the flags generated an overflow into the sign bit.

(ii)The memory system

- Memory may be viewed as a linear array of bytes numbered from zero up to 2^{32}-l.
- Data items may be 8-bit bytes, 16-bit half-words or 32-bit words. Words are always aligned on 4-byte boundaries and half-words are aligned on even byte boundaries.

- A word-sized data item must occupy a group of four byte locations starting at a byte address which is a multiple of four
- Half-words occupy two byte locations starting at an even byte address.

(iii) Load-store architecture

ARM (RISC processors) employs a load-store architecture. This means that the instruction set will only process (add, subtract, and so on) values which are in registers and will always place the results of such processing into a register.

ARM instructions fall into one of the following three categories:

- Data processing instructions: These use and change only register values.
- Data transfer instructions: These copy memory values into registers
- Control flow instructions: Normal instruction execution uses instructions stored at consecutive memory addresses.

(iv) Supervisor mode

- The ARM processor supports a protected supervisor mode.
- The protection mechanism ensures that user code cannot gain supervisor privileges without appropriate checks being carried out to ensure that the code is not attempting illegal operations.
- The upshot of this for the user-level programmer is that system-level functions can only be accessed through specified supervisor calls.

(v) The ARM instruction set

All ARM instructions are 32 bits wide and are aligned on 4-byte boundaries in memory. The most notable features of the ARM instruction set are:

- 3-address data processing instructions
- conditional execution of every instruction;
- the ability to perform a general shift operation and a general ALU operation in a single instruction that executes in a single clock cycle;
- open instruction set extension through the coprocessor instruction set, including adding new registers and data types to the programmer's model;

(vi) The I/O system

- The ARM handles I/O (input/output) peripherals (such as disk controllers, network interfaces) as memory-mapped devices with interrupt support.

- The internal registers in these devices appear as addressable locations within the ARM's memory map and may be read and written using the same (load-store) instructions as any other memory locations.
- Peripherals may attract the processor's attention by making an interrupt request using either the normal interrupt (IRQ) or the fast interrupt (FIQ) input.
- Both interrupt inputs are level-sensitive and maskable. Normally most interrupt sources share the IRQ input, with just one or two time-critical sources connected to the higher-priority FIQ input.

(v)ARM exceptions

The ARM architecture supports a range of interrupts, traps and supervisor calls are known as exceptions.

The general way these are handled is the same in all cases:

1. The current state is saved by copying the PC into rl4_exc and the CPSR into SPSR_exc (where exc stands for the exception type).

2. The processor operating mode is changed to the appropriate exception mode.

3. The PC is forced to a value between 0016and 1C16, the particular value depending on the type of exception.

- The exception handler will use rl3_exc, which will normally have been initialized to point to a dedicated stack in memory, to save some user registers for use as work registers.
- The return to the user program is achieved by restoring the user registers and then using an instruction to restore the PC and the CPSR atomically.

2. Explain the various ARM development tools?

Software development for the ARM is supported by a high range of tools developed by ARM Limited, and there are also many third party and public domain tools available, such as an ARM back-end for *C* compiler.

It makes a good environment for software development; the tools are intended for **cross-development** from a platform such as a PC running Windows or a suitable UNIX workstation.

(i) The ARM C compiler

The ARM C compiler is compliant with the ANSI (American National Standards Institute) standard for C and is supported by the appropriate library of standard functions.

(ii) The ARM assembler

- The ARM assembler is a full macro assembler which produces ARM object format output that can be linked with output from the C compiler.
- Assembly source language is near machine-level, with most assembly instructions translating into single ARM instructions.

(iii)The linker

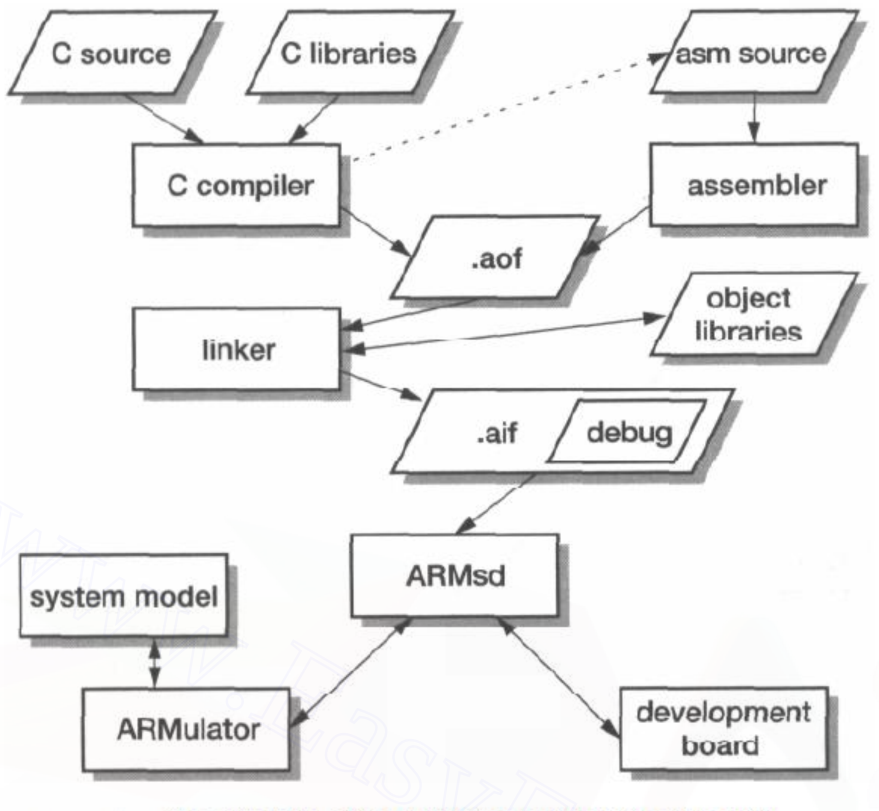
- The linker takes one or more object files and combines them into an executable program.
- It can assemble the various components of the program in a number of different ways, depending on whether the code is to run in RAM or ROM.

(iv) ARMsd

- The ARM symbolic debugger is a front-end interface to assist in debugging programs running either under emulation (on the ARMulator) or remotely on a target system such as the ARM development board.
- Debugging a system where the processor core is embedded within an application-specific system chip .

(v) ARMulator

- The ARMulator *(ARM emulator)* is a suite of programs that models the behaviour of various ARM processor cores in software on a host system. It can operate at various levels of accuracy:
- ***Instruction-accurate* modeling** gives the exact behaviour of the system state without regard to the precise timing characteristics of the processor.
- ***Cycle-accurate* modeling** gives the exact behaviour of the processor on a cycle by-cycle basis, allowing the exact number of clock cycles that a program requires to be established.
- ***Timing-accurate* modeling** presents signals at the correct time within a cycle, allowing logic delays to be accounted for.



The structure of the ARM cross-development toolkit.

(vi) ARM development board

The ARM Development Board is a circuit board incorporating a range of components and interfaces to support the development of ARM-based systems.

It includes an ARM core (for example, an ARM7TDMI), memory components which can be configured to match the performance and bus-width of the memory in the target system, and electrically programmable devices which can be configured to emulate application-specific peripherals

(vii) Software Toolkit

ARM Limited supplies the complete set of tools described above, with some support utility programs and documentation, as the 'ARM Software Development Toolkit'.

These files may be:

• source files (C, assembler, and so on);

• object files;

• library files.

(viii) JumpStart

The JumpStart tools from VLSI Technology include the same basic set of development tools but present a full X-windows interface on a suitable workstation rather than the command-line interface of the standard ARM toolkit.

3. Explain the architecture of ARM

- ARM is the acronym for Advanced RISC Machine
- RISC stands for Reduced Instruction Set Computer.
- RISC based architecture although invented quite earlier but has become popular and overtook its rival architecture Complex Instruction Set Computer (CISC) somewhere during late nineties. Most popular CISC architecture is 80x86 processors from Intel.
- The advantage of RISC is in the simplicity (in terms of processor resource consumption) of the instructions and processing time. Each instruction takes only single clock cycle. Overall power consumption is very less. Due to this fast response, low power consumption and coding flexibility, RISC architecture is highly suitable for embedded systems.
- However there is one drawback with RISC, that is the instruction set code is longer and takes more memory. This issue is no more a concern with the growth in the memory technology.

  Highlights of ARM7TDMI:
- There are 37 registers of 32 bit wide in this processor core. 16 registers are available for the programmer.
- It's pipeline architecture; that is 3 instructions are processed simultaneously at 3 different stages.
- The bus architecture is of Von Neumann type where single 32-bit data bus carry both instructions and data.
- The data-types can be of 8 bit /16 bit/32 bit wide.

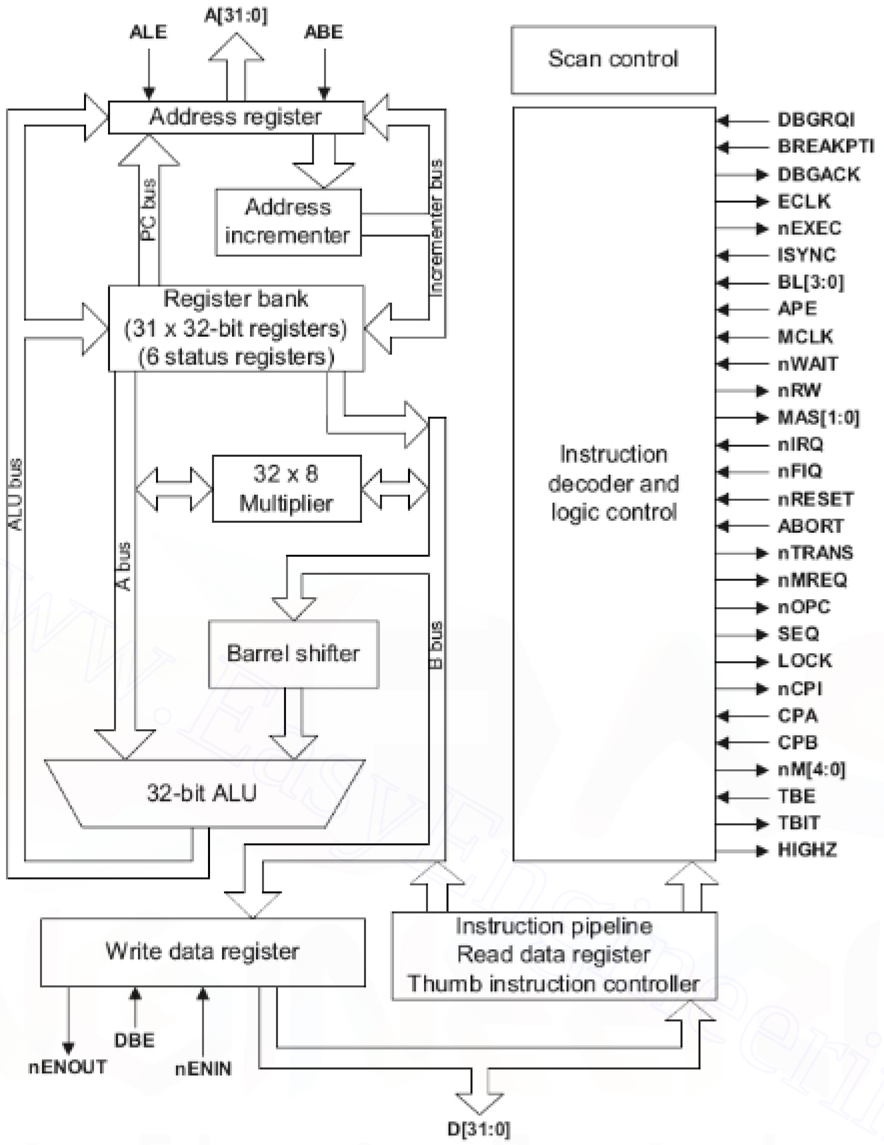


Fig:ARM architecture

- Processor can run on seven different modes based on the application requirement.
- Built in 32x8 multiplier and a 32 bit barrel shifter (both needed much for DSP functionality)
- This processor can also execute another instruction set called THUMB state (16 bit) to give the programmer an option to use this processor like CISC processor.
- The total instruction set can be tidier and takes less memory space. Built in SRAM of 16 KB and FlashROM of 128 KB.
- Robust clock network
- Interrupt controller supports 41 interrupt resources

- External memory controller to access ROM, SRAM, and I/O connected to the external memory space.
- Has system timer of 16bit auto reload timer with its interrupt given high priority.
- Built-in DMA controller enables direct data transfer between memory-memory, I/O -memory, and between I/O -I/O devices to spare the CPU from simple data transfer burden.
- Watchdog timer to monitor the program from running out of control and generate interrupt or reset signal.
- Built-in 4-channel, 10-bit resolution analog-to-digital converter supports two modes of operation: Scan mode sequentially converts input from the selected range of channels; select mode converts input from a single channel.
- 15 general purpose I/O ports: 8channels of 8-bit, 3 channels of 7-bit, 3 channels of 6-bits, and one channel of 5 bits.
- Integrates one channel of I2C bus interface, one I2S (serial audio interface) bus interface, one UART interface, one SIO interface and one SPI interface.
- Real time clock (RTC) with 10,000-year calendar with resolution down to 1 second.
- Flexible timer block with 6 channels of 16 bit timer.

4. **Explain the memory Hierarchy in ARM processor.**

(i) **Memory size and speed**

A typical computer memory hierarchy comprises several levels, each level having a characteristic size and speed.

- A RISC processor will typically have around thirty-two 32-bit registers making a total of 128 bytes, with an access time of a few nanoseconds.
- On-chip cache memory will have a capacity of eight to 32 Kbytes with an access time around ten nanoseconds.
- Main memory will be megabytes to tens of megabytes of dynamic RAM with an access time around 100 nanoseconds.
- Backup store, usually on a hard disk, will be hundreds of Mbytes up to a few Gbytes with an access time of a few tens of milliseconds.

(ii) On-chip memory

- Some form of on-chip memory is essential if a microprocessor is to deliver its

best performance.

- Only on-chip memory can support zero wait state access speeds, and it will also give better power-efficiency and reduced electromagnetic interference than off-chip memory.

  On-chip RAM is preferred to cache for a number of reasons:

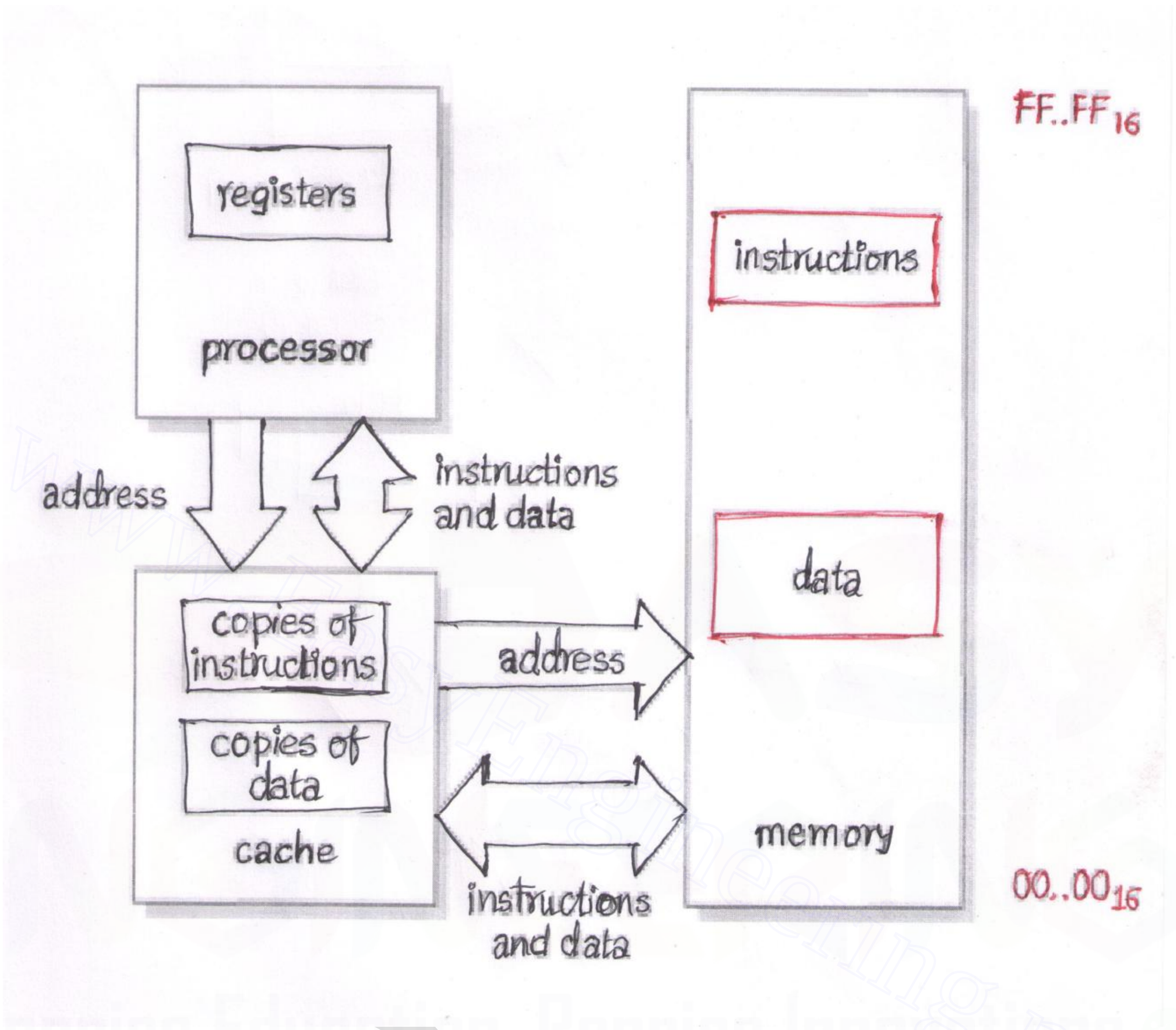
  - It is simpler, cheaper, and uses less power.
  - It has more deterministic behaviour.

(iii) Caches

- The first RISC processors were introduced at a time when standard memory parts were faster than their contemporary microprocessors by these caches.
- A cache memory is a small, very fast memory that retains copies of recently used memory values.
- It operates transparently to the programmer, automatically deciding which values to keep and which to overwrite.
- Caches can be built in many ways. At the highest level a processor can have one of the following two organizations:
  - ❖ A unified cache.
  - ❖ Separate instruction and data caches.
- The proportion of all the memory accesses that are satisfied by the cache is the hit rate, usually expressed as a percentage, and the proportions that are not is the miss rate.

  This, simplest, cache organization has a number of properties

  - A particular memory item is stored in a unique location in the cache; two items with the same cache address field will contend for use of that location.
  - Only those bits of the address that are not used to select within the line or to address the cache RAM need be stored in the tag field.



❖ Fig: unified cache.

The tag and data access can be performed at the same time, giving the fastest cache access time of any organization. **(iv) Memory management**

- ➢ A single processor can only execute instructions from one program at any instant.
- ➢ The rapid switching is managed by the operating system, so the application programmer can write his or her program as though it owns the whole machine.

The mechanism used to support this is described by the term memory management unit (MMU). There are two principal approaches to memory management, called segmentation and paging.

(v) Segmentation:

- The simplest form of memory management allows an application to view its memory as a set of segments, where each segment contains a particular sort of information.
- Every memory access provides a segment selector and a logical address to the MMU.
- Segmentation allows a program to have its own private view of memory and to coexist transparently with other programs in the same memory space.

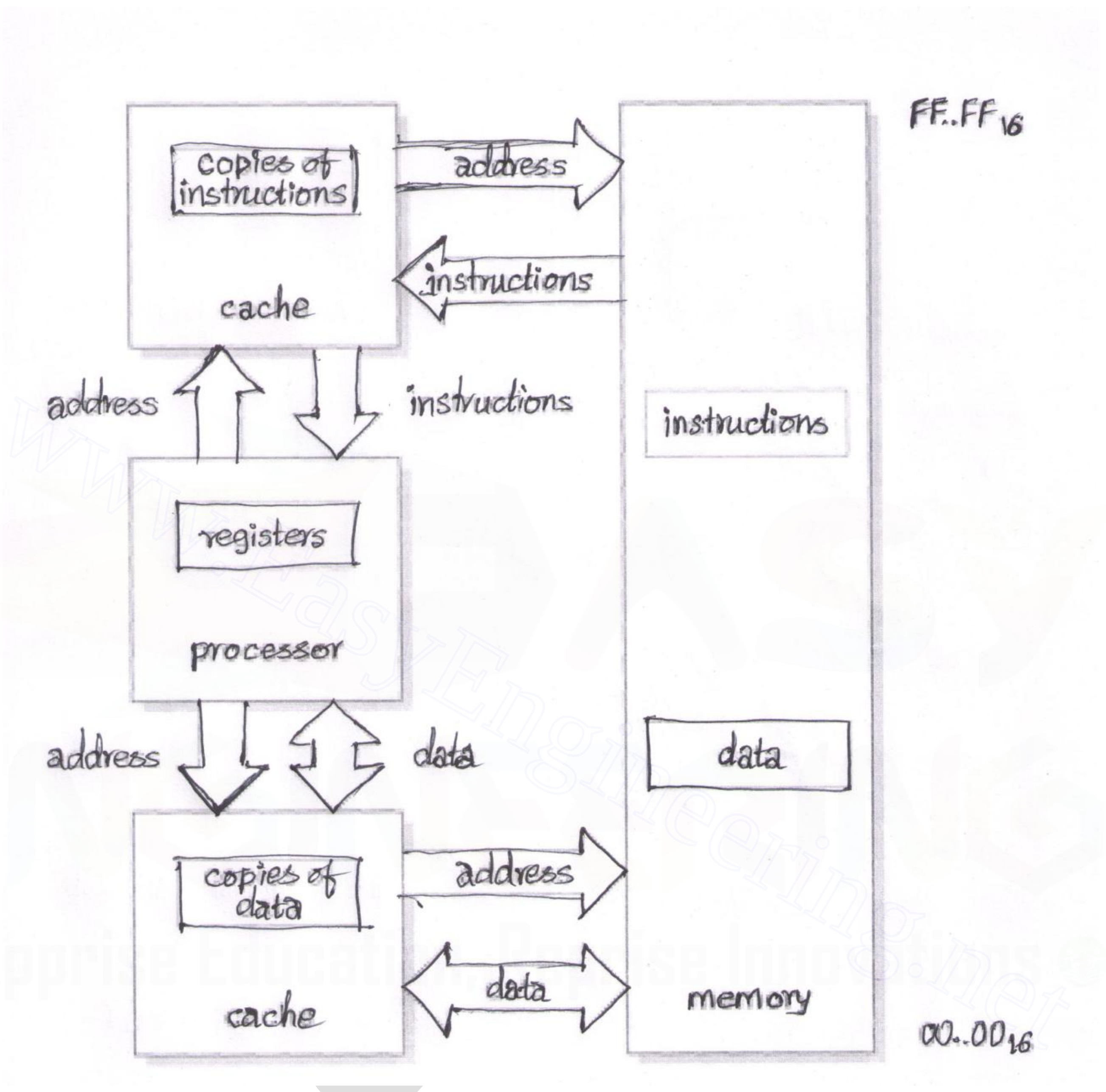
- It runs into difficulty, when the coexisting programs vary and the available memory is limited. Since the segments are of variable size, the free memory becomes fragmented over time and a new program may be unable to start, not because there is insufficient free memory, but because the free memory is all in small pieces none of which is big enough to hold a segment of the size required by the new program.
- Most processors now incorporate a memory mapping scheme based on fixed-size chunks of memory called pages.

(vi) Paging:

In a paging memory management scheme both the logical and the physical address spaces are divided into fixed-size components called pages.

A page is usually a few kilobytes in size, but different architectures use different page sizes. The relationship between the logical and physical pages is stored in page tables, which are held in main memory.

A simple sum shows that storing the translation in a single table requires a very large table: if a page is 4 Kbytes, 20 bits of a 32-bit address must be translated, which requires 2^{20} x 20 bits of data in the table.



❖ Fig: Separate instruction and data caches.

(vii) Virtual memory:

- An operating system which has run out of memory to allocate can transparently move a page or a segment out of main memory into backup store, which for this purpose is usually a hard disk, and mark it as absent.
- When implemented with the paged memory management scheme, this process is known as demand-paged virtual memory. A program can be written to occupy a virtual memory space that is larger than the available physical memory space.

5. **a)Write an assembly level program to print a text in r0 register.**

```
SWI_WriteC    EQU          &0            ;output character in r0
SWI_Exit      EQU          &11           ;finish program
              ENTRY                      ;code entry program
START         ADR          r1,TEXT       ;r1→ "HELLO"
LOOP          LDRB         r0, [r1], #1  ;get the next byte
      CMP           r0,[r1],#1   ;check for text end
      SWINE         SWI_WriteC ;if not end print
      BNE           LOOP         ;.. and loop back
      SWI           SWI_Exit     ;end of execution
      TEXT          =            "Hello World", &0a,&0d,0
      END                                    ; end of program
source
```

5. **b) Write a subroutine to output a text string immediately following the call.**

```
         AREA      Text_Out, CODE, READONLY
SWI_Exit      EQU  &0                ;      output character in r0
SWI_Exit      EQU  &11               ;      finish program
              ENTRY                         ;      code entry point
              BL             TextOut        ;      print following string
              =              *Test string* , &0a, &0d, 0;
              ALIGN
              SWI            SWI_Exit       ;      finish
```

```
TextOut     LDRB    r0, [r14], #1     ;   get next character
            CMP     r0, #0            ;   test for end mark
            SWINE   SWI_WriteC        ;   if not end, print…
            BNE     TextOut           ;    .. and loop
            ADD     r14, r14, #3      ;   pass next word boundary
            BIC     r14, r14, #3      ;   round back to boundary
            MOV     pc, r14           ;   return
            END
```

UNIT – V ARM ORGANIZATION

PART-A

1. Give some examples of embedded ARM applications.

- Cortex-A9 used in Samsung Galaxy S II, Sony Xperia U, Apple iPad
- ARM926EJ-S used in Sony Ericsson (K,W series), LG arena
- Cortex-A8 used in HTC Desire , Apple iPhone3GS
- ARM710 used in Acorn RISC PC 700

2. What are the 3- stage pipelines?

- Fetch
- Decode
- Execute

3. What are the five stage pipelines?

- Fetch
- Decode
- Execute
- Buffer / data
- Write back

4**. What are the main features of ARM instruction set?**

- All instructions are 32 bits long.
- Most instructions execute in a single cycle.
- Every instruction can be conditionally executed.
- Load/store architecture
- Data processing instructions act only on registers
- Three operand format
- Combined ALU and shifter for high speed bit manipulation

5. What is instruction pipelining?

Instruction pipelining is a technique that implements a form of parallelism called instruction-level parallelism within a single processor. It allows faster CPU throughput (the number of instructions that can be executed in a unit of time) than would otherwise be possible at a given clock rate. The basic instruction cycle is broken up into a series called a pipeline.

6. What are the registers of ARM?

ARM has 37 registers in total, all of which are 32-bits long.

1 dedicated program counter

1 dedicated current program status register

5 dedicated saved program status registers

30 general purpose registers

7. What are the operating modes?

Seven operating modes:

User *Privileged *FIQ *IRQ

*Abort *Undefined *Supervisor

8. What is stack in ARM?

A stack is an area of memory which grows as new data is "pushed" onto the "top" of it, and shrinks as data is "popped" off the top. Two pointers define the current limits of the stack.

- Base pointer, used to point to the "bottom" of the stack (the first location).
- Stack pointer, used to point the current "top" of the stack.

9. What is co-processor interface in ARM processor?

The processor supports the connection of on-chip coprocessors through an external coprocessor interface. All types of coprocessor instruction are supported. The ARM instruction set supports the connection of 16 coprocessors, numbered 0-15, to an ARM processor.

10. What is debugging in ARM?

ARM processors include hardware debugging facilities, allowing software debuggers to perform operations such as halting, stepping, and breakpointing of code starting from reset. These facilities are built using JTAG support, though some newer cores optionally support ARM's own two-wire "SWD" protocol.

11. Define the term Fetch, Decode & Execute.

Fetch: Instruction is fetched from memory and placed in the instruction pipeline.

Decode:Instruction is decoded and the datapath control signals prepared for the next cycle. In this stage the instruction 'owns' the decode logic but not the datapath.

Execute:Instruction 'owns' the datapath; the register bank is read, an operand shifted, the ALU result generated and written back into a destination register.

12. Define the term buffer data and write back in 5 stage pipeline.

Buffer/data: Data memory is accessed if required. Otherwise the ALU result is simply buffered for one clock cycle to give the same pipeline flow for all instructions.

Write-back:Results generated by the instruction are written back to the register file, including any data loaded from memory.

PART-B

1. **Briefly explain the 3-STAGE pipeline ARM organization.**

The principal components are:

• The register bank, which stores the processor state. It has two read ports and one write port which can each be used to access any register, plus an additional read port and an additional write port that give special access to r15, the program counter.

• The barrel shifter, which can shift or rotate one operand by any number of bits.

• The ALU, which performs the arithmetic and logic functions required by the instruction set.

• The address register and incremental, which select and hold all memory addresses and generate sequential addresses when required.

• The data registers, which hold data passing to and from memory.

• The instruction decoder and associated control logic.

3-stage pipeline stages:

Fetch: the instruction is fetched from memory and placed in the instruction pipeline.

Decode: the instruction is decoded and the datapath control signals prepared for the next cycle. In this stage the instruction 'owns' the decode logic but not the datapath.

Execute: the instruction 'owns' the data path; the register bank is read, an operand shifted, the ALU result generated and written back into a destination register.

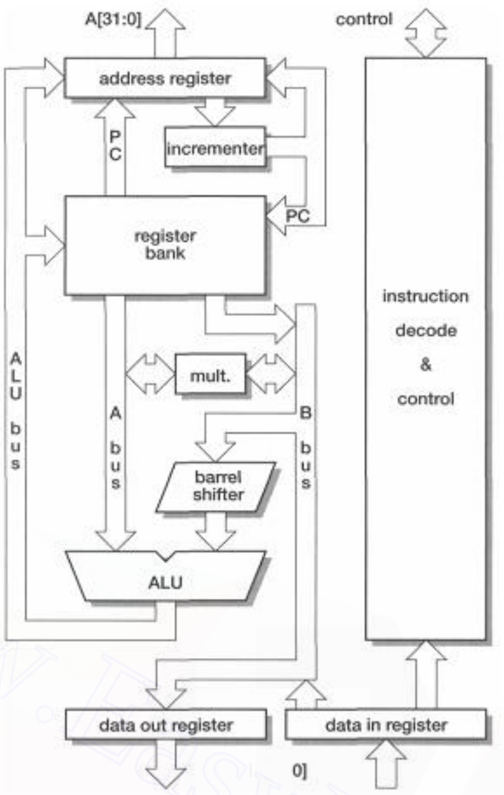
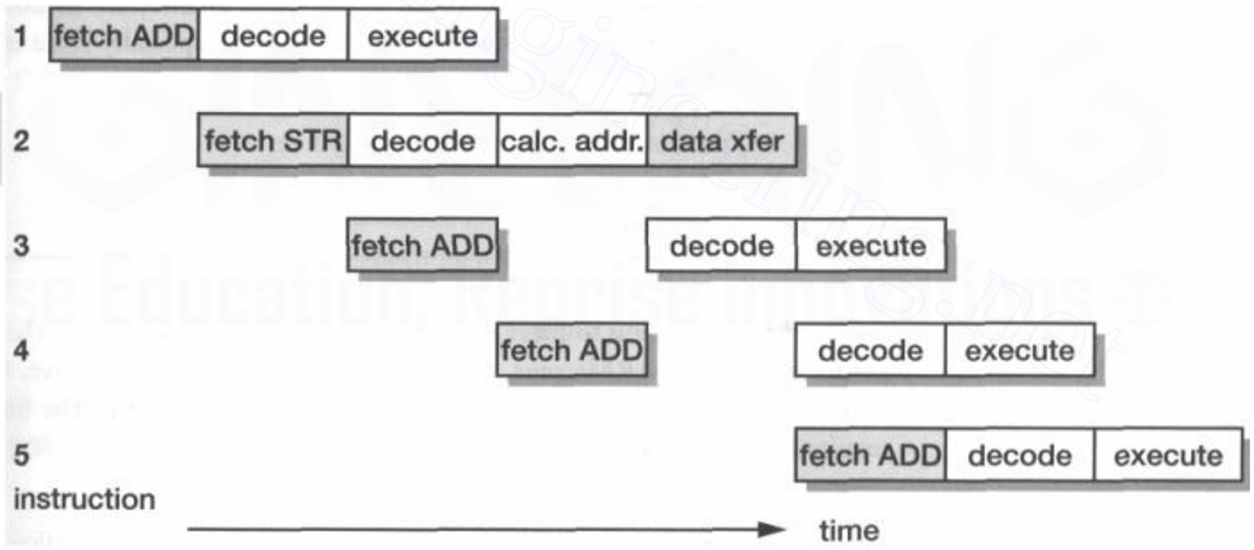


Fig:ARM 3 stage pipeline


1 fetch ADD decode execute
2 fetch STR decode calc. addr. data xfer
3 fetch ADD decode execute
4 fetch ADD decode execute
5 fetch ADD decode execute
instruction
time

ARM multi-cycle instruction 3-stage pipeline operation.

An individual instruction takes three clock cycles to complete, so it has a three-cycle latency, but the through-put is one instruction per cycle.

A sequence of single-cycle ADD instructions with a data store instruction, STR, occurring after the first ADD. The cycles that access main memory is shown with light shading so it can be seen that memory is used in every cycle. The data path is likewise used in every cycle, being involved in all the execute cycles, the address calculation and the data transfer.

The decode logic is always generating the control signals for the data path to use in the next cycle, so in addition to the explicit decode cycles it is also generating the control for the data transfer during the address calculation cycle of the STR.

Thus, in this instruction sequence, all parts of the processor are active in every cycle and the memory is the limiting factor, The simplest way to view breaks in the ARM pipeline is to observe that:

When the processor is executing simple data processing instructions the pipeline enables one instruction to be completed every clock cycle.

- All instructions occupy the datapath for one or more adjacent cycles.
- For each cycle that an instruction occupies the datapath, it occupies the decode logic in the immediately preceding cycle.
- During the first datapath cycle each instruction issues a fetch for the next instruction but one.
- Branch instructions flush and refill the instruction pipeline.

2. **Describe the 5-STAGE pipeline ARM organization.**

All processors have to develop to meet the demand for higher performance. The 3-stage pipeline used in the ARM cores up to the ARM is very cost-effective, but higher performance requires the processor organization to be rethought. The time, T , required to execute a given program is given by:

$$\mathbf{T_{prog}=(N_{inst} \times CPI) / f_{clk}}$$

Where, N_{inst} is the number of ARM instructions executed in the course of the program,

CPI is the average number of clock cycles per instruction

f_{clk} is the processor's clock frequency.

Since N_{inst} is constant for a given program, there are only two ways to increase performance:

- Increase the clock rate, f_{clk}. This requires the logic in each pipeline stage to be simplified and, therefore, the number of pipeline stages to be increased.
- Reduce the average number of clock cycles per instruction, CPI. This requires either that instructions which occupy more than one pipeline slot in a 3-stage pipeline ARM are re-implemented to occupy fewer slots, or that pipeline stalls caused by dependencies between instructions are reduced, or a combination of both.

To get a significantly better CPI, the memory system must deliver more than one value in each clock cycle either by delivering more than 32 bits per cycle from a single memory or by having separate memories for instruction and data accesses. As a result of the above issues, higher performance ARM cores employ a 5-stage pipeline and have separate instruction and data memories. Breaking instruction execution down into five components rather than three reduces the maximum work which must be completed in a clock cycle, and hence allows a higher clock frequency to be used. The separate instruction and data memories allow a significant reduction in the core's CPI.

A typical 5-stage ARM pipeline is that employed in the ARM9TDMI. The ARM processors which use a 5-stage pipeline have the following pipeline stages:

• Fetch; the instruction is fetched from memory and placed in the instruction pipeline.

• Decode; the instruction is decoded and register operands read from the register file. There are three operand read ports in the register file, so most ARM instructions can source all their operands in one cycle.

• Execute; an operand is shifted and the ALU result generated. If the instruction is a load or store the memory address is computed in the ALU.

• Buffer/data; data memory is accessed if required. Otherwise the ALU result is simply buffered for one clock cycle to give the same pipeline flow for all instructions.

• Write-back; the results generated by the instruction are written back to the register file, including any data loaded from memory.

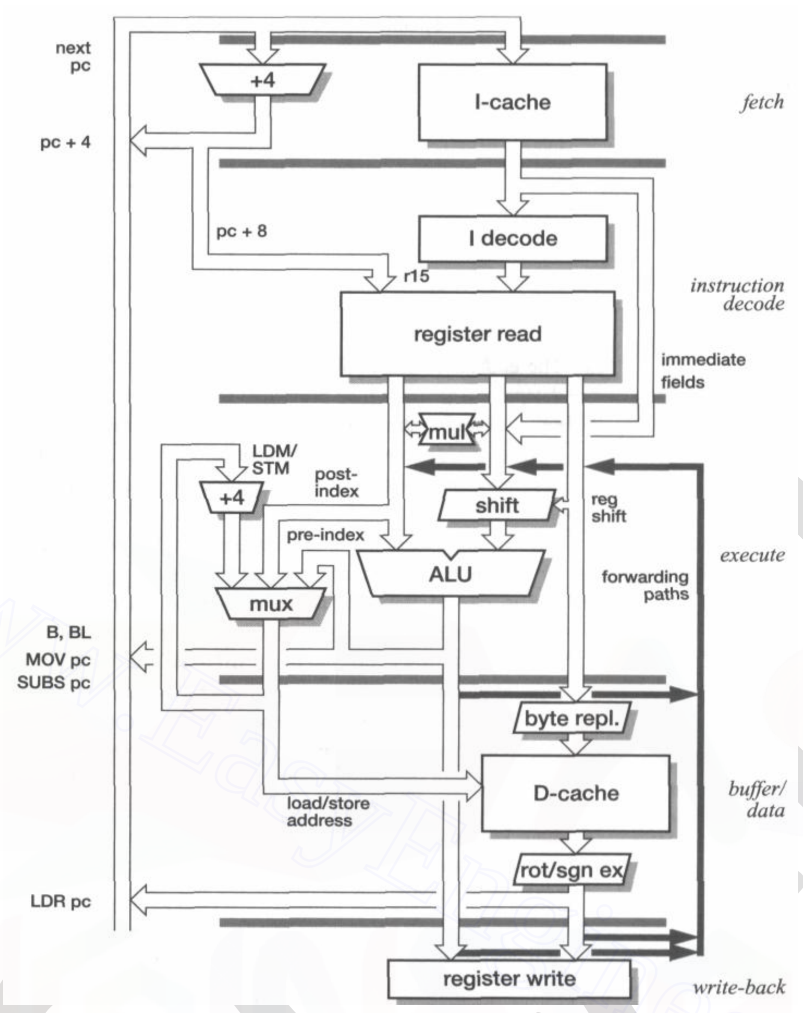


Fig. ARM9TDMI 5-stage pipeline organization

Data forwarding : A major source of complexity in the 5-stage pipeline (compared to the 3-stage pipeline) is that, because instruction execution is spread across three pipeline stages, the only way to resolve data dependencies without stalling the pipeline is to introduce forwarding paths.

Data dependencies arise when an instruction needs to use the result of one of its predecessors before that result has returned to the register file. Forwarding paths allow results to be passed between stages as soon as they are available, and the 5-stage ARM pipeline requires each of the three source operands to be forwarded from any of three intermediate result registers

There is one case where, even with forwarding, it is not possible to avoid a pipeline stall. Consider the following code sequence:

LDR rN, XX ; load rN from XX

ADD r2, r1, rN ; and use it immediately

The processor cannot avoid a one-cycle stall as the value loaded into rN only enters the processor at the end of the buffer/data stage and it is needed by the following instruction at the start of the execute stage. The only way to avoid this stall is to encourage the compiler (or assembly language programmer) not to put a dependent instruction immediately after a load instruction.

3. Describe the datatypes used for architectural support of ARM.

The definition of the ARM instruction set introduces an abstraction away from logic variables when it expresses the functions of the processor in terms of instructions, bytes, words and addresses.

Each of these terms describes a collection of logic variables viewed in a particular way. The difference is not in the way the information is stored but in the way it is used. A computer data type can therefore be characterized by:

- the number of bits it requires;
- the ordering of those bits;
- the uses to which the group of bits is put.

Some data types, such as addresses and instructions, exist principally to serve the Purposes of the computer, whereas others exist to represent information in a way that is accessible to human users. The most basic of the latter category, in the context of computation, is the number.

(i) Numbers :A complex mechanism capable of computing the behaviour of transistors a thousandth of a millimeter wide switching a hundred million times a second .

(ii) Roman numerals is a number written by a human:MCMXCV

(iii) Decimal Numbers: The interpretation of this **Roman** numeral is complex; the value of a symbol depends on the symbol and its positional relationship to its neighbors. This way of writing a number has largely been replaced by the **decimal** scheme where the same number appears as: 1995

(iv)Binary coded decimal

We need four Boolean variables to be able to represent each digit from 0 to 9 differently, so the first form of this number that could easily be handled by logic gates is:

0001 1001 1001 0101

This is the binary coded decimal scheme which is supported by some computers
and is commonly used in pocket calculators.

(v) BINARY NOTATION: Most computers, abandon the human-oriented decimal scheme altogether in favour of a pure binary notation where the same number becomes:11111001011

(vi) HEXADECIMAL NOTATION : Although machines use binary numbers extensively internally, a typical 32-bit binary number is fairly unmemorable, but rather than convert it to the familiar decimal form (which is quite hard work and error-prone), computer users often describe the number in **hexadecimal** (base 16) notation. This uses 0 to 9 as themselves and A to F to represent 10 to 15. Our number becomes: 7CB

(vii) NUMBER RANGES The ARM deals efficiently with 32-bit quantities, so the first data type that the architecture supports is the 32-bit (unsigned) integer, which has a value in the range: 0 to 4 294 967 295_{10} = 0 to $FFFFFFFF_{16}$

(vii) SIGNED INTEGERS In many cases it is useful to be able to represent negative numbers as well as positive ones. Here the ARM supports a 2's complement binary notation where the value of the top bit is made negative; in a 32-bit signed integer all the bits have the same value as they have in the unsigned case apart from bit 31, which has the value -231instead of +231. Now the range of numbers is:
-2 147 483 648_{10} to +2 147 483 647_{10} = 80000000_{16} to $7FFFFFFF_{16}$

The 'architectural support' for signed integers is the V flag in the program status registers which has no use when the operands are unsigned but indicates an **overflow**(out of range) error when signed operands are combined.

(viii)REAL NUMBERS The representation of real numbers in computers is a big issue that is deferred to the next section. An ARM core has no support for real data types, though ARM Limited has defined a set of types and instructions that operate on them. These instructions are either executed on a floating-point coprocessor or emulated in software.

(ix)PRINTABLE CHARACTERS After the number, the next most basic data type is the printable character. To control a standard printer we need a way to represent all the normal characters such as the upper and lower case alphabet, decimal digits from 0 to 9, punctuation marks and a number of special characters such as £, $, %, and so on.

(x) ASCII The normal way to store an ASCII character in a computer is to put the 7-bit binary code into an 8-bit byte. Many systems extend the code using, for example, the 8-bit ISO character set where the other 128 binary codes within the byte represent special characters (for instance, characters with accents). The most flexible way to represent characters is the 16-bit 'Unicode' which incorporates many such 8-bit character sets within a single encoding.'1995' encoded as 8-bit printable characters is:
00110001 00111001 00111001 00110101 =31 39 39 3516

ARCHITECTURAL SUPPORT FOR CHARACTERS

The support in the ARM architecture for handling characters is the unsigned byte load and store instructions; these have already been mentioned as being available to support small unsigned integers, but that role is rare compared with their frequency of use for transferring ASCII characters.

(a) Byte ordering

The above ASCII example highlights an area of some potential difficulty. It is written to be read from left to right, but if it is read as a 32-bit word, the least significant byte is at the right. A character output routine might print characters at successive increasing byte addresses, in which case, with 'little-endian' addressing, it will print '5991'.

(b) High-level languages

A high-level language defines the data types that it needs in its specification, usually without reference to any particular architecture that it may run on. Sometimes the number of bits used to represent a particular data type is architecture-dependent in order to allow a machine to use its most efficient size.

(c) ANSI C basic data types

The dialect of the 'C' language defined by the American National Standards Institute (ANSI), and therefore known as 'ANSI standard C' or simply 'ANSI C', defines the following basic data types:

• Signed and unsigned **characters** of at least eight bits.

• Signed and unsigned **short integers** of at least 16 bits.

• Signed and unsigned **integers** of at least 16 bits.

• Signed and unsigned **long integers** of at least 32 bits.

• **Floating-point, double** and **long double** floating-point numbers.

The ARM C compiler adopts the minimum sizes for each of these types except the standard integer, where it uses 32-bit values since this is the most frequently used data type and the ARM supports 32-bit operations more efficiently than 16-bit operations.

ANSI C derived data types

In addition, the ANSI C standard defines derived data types:

- **Arrays** of several objects of the same type.
- **Functions** which return an object of a given type.
- **Structures** containing a sequence of objects of various types.
- **Pointers** (which are usually machine addresses) to objects of a given type.
- Unions which allow objects of different types to occupy the same space at different times.

4.(a) Explain the ARM coprocessor interface

The ARM supports a general-purpose extension of its instruction set through the addition of hardware coprocessors, and it also supports the software emulation of these coprocessors through the undefined instruction trap.

Coprocessor architecture

The coprocessors most important features are:

- Support for up to 16 logical coprocessors.
- Each coprocessor can have up to 16 private registers of any reasonable size; they are not limited to 32 bits.
- Coprocessors use a load-store architecture,with instructions to perform internal operations on registers, instructions to load and save registers from and to memory, and instructions to move data to or from an ARM register.

The simpler ARM cores offer the coprocessor interface at board level, so a co-processor may be introduced as a separate component. High clock speeds make board-level interfacing very difficult, so the higher-performance ARMs restrict the coprocessor interface to on-chip use, in particular for cache and memory management control functions, but other on-chip coprocessors may also be supported.

ARM7TDMI coprocessor interface

The ARM7TDMI coprocessor interface is based on 'bus watching' (other ARM cores use different techniques). The coprocessor is attached to a bus where the ARM instruction stream flows into the ARM, and the coprocessor copies the instructions into an internal pipeline that mimics the behaviour of the ARM instruction pipeline. As each coprocessor instruction begins execution there is a 'hand-shake' between the ARM and the coprocessor to confirm that they are both ready to execute it. The handshake uses three signals:

1. cpi (from ARM to all coprocessors).This signal, which stands for 'Coprocessor Instruction', indicates that the ARM has identified a coprocessor instruction and wishes to execute it.
2. cpa (from the coprocessors to ARM).This is the 'Coprocessor Absent' signal which tells the ARM that there is no coprocessor present that is able to execute the current instruction.
3. cpb (from the coprocessors to ARM).This is the 'CoProcessor Busy' signal which tells the ARM that the coprocessor cannot begin executing the instruction yet.

The timing is such that both the ARM and the coprocessor must generate their respective signals autonomously. The coprocessor cannot wait until it sees cpi before generating cpa and cpb.

Handshake outcomes:

Once a coprocessor instruction has entered the ARM7TDMI and coprocessor pipe-lines, there are four possible ways it may be handled depending on the handshake signals:

1. The ARM may decide not to execute it, either because it falls in a branch shadow or because it fails its condition code test.
2. The ARM may decide to execute it, but no present coprocessor can take it so cpa stays active. ARM will take the undefined instruction trap and use software to recover, possibly by emulating the trapped instruction.
3. ARM decides to execute the instruction and a coprocessor accepts it, but cannot execute it yet. The coprocessor takes cpa low but leaves cpb high. The ARM will 'busy-wait' until the coprocessor takes cpb low, stalling the instruction stream at this point. If an enabled interrupt request arrives while the coprocessor is busy, ARM will break off to handle the interrupt, probably returning to retry the coprocessor instruction later.
4. ARM decides to execute the instruction and a coprocessor accepts it for immediate execution, cpi, cpa and cpb are all taken low and both sides commit to complete the instruction.

Data transfers

If the instruction is a coprocessor data transfer instruction the ARM is responsible for generating an initial memory address but the coprocessor determines the length of the transfer;

ARM will continue incrementing the address until the coprocessor signals completion. The cpa and cpb handshake signals are also used for this purpose. Since the data transfer is not interruptible once it has started, coprocessors should limit the maximum transfer length to 16 words.

Pre-emptive execution

A coprocessor may begin executing an instruction as soon as it enters its pipeline so long as it can recover its state if the handshake does not ultimately complete.

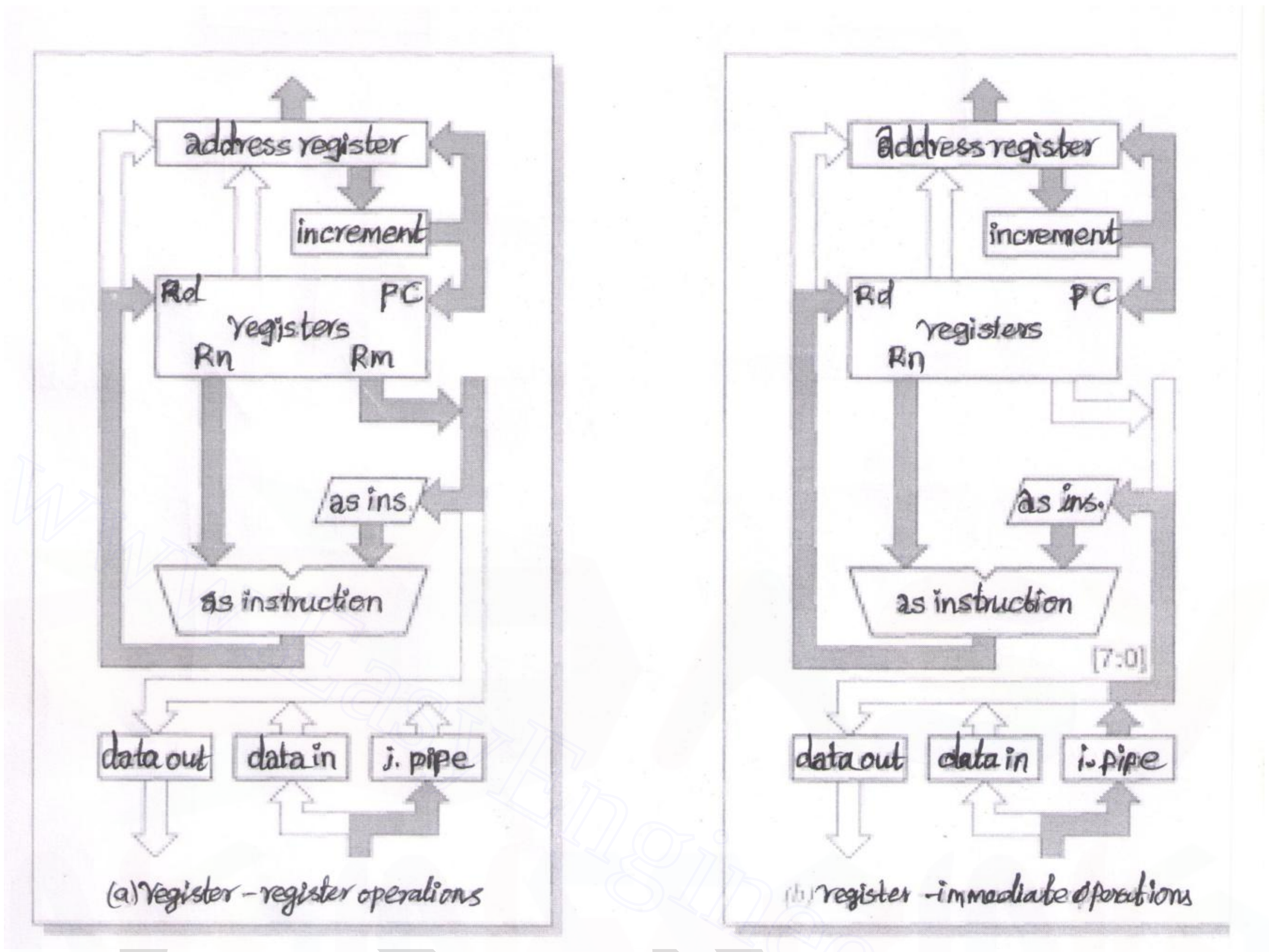
4. (b)Explain how ARM instruction execution done in ARM processor

Data processing instructions:

→A data processing instruction requires two operands, one of which is always a register and the other is either a second register or an immediate value.

→The second operand is passed through the barrel shifter where it is subject to a general shift operation, and then it is combined with the first operand in the ALU using a general ALU operation. Finally, the result from the ALU is written back into the destination register

→All these operations take place in a single clock cycle data processing instructions only the bottom eight bits (bits [7:0]) of the instruction are used in the immediate value.



Data transfer instructions

→A data transfer (load or store) instruction computes a memory address in a manner very similar to the way a data processing instruction computes its result.

→A register is used as the base address, to which is added (or from which is subtracted) an offset which again may be another register or an immediate value.

→This time, however, a 12-bit immediate value is used without a shift operation rather than a shifted 8-bit value.

Datapath operation

→ The datapath operation for the two cycles of a data store instruction (SIR) with an immediate offset .Incremented PC value is stored in the register bank at the end of the first cycle so that the address register is free to accept the data transfer address for the second cycle, then at the end of the second cycle the PC is fed back to the address register to allow instruction pre fetching to continue.

→It should, perhaps, be noted at this stage that the value sent to the address register in a cycle is the value used for the memory access in the following cycle.

→ The address register is, in effect, a pipeline register between the processor data path and the external memory.

Branch instructions

→ Branch instructions compute the target address in the first cycle. A 24-bit immediate field is extracted from the instruction and then shifted left two bit positions to give a word-aligned offset which is added to the PC.

→The result is issued as an instruction fetch address, and while the instruction pipeline refills the return address is copied into thelink register (r14) if this is required.

→The third cycle, which is required to complete the pipeline refilling, is also used to make a small correction to the value stored in the link register in order that it points directly at the instruction which follows the branch.

5. Explain any one application of ARM in the field of embedded system.

The Ericsson-VLSI Bluetooth Baseband Controller

→Bluetooth is a de-facto standard for wireless data communication for the 2.4 GHz band developed by a consortium of companies including Ericsson, IBM, Intel, Nokia and Toshiba.

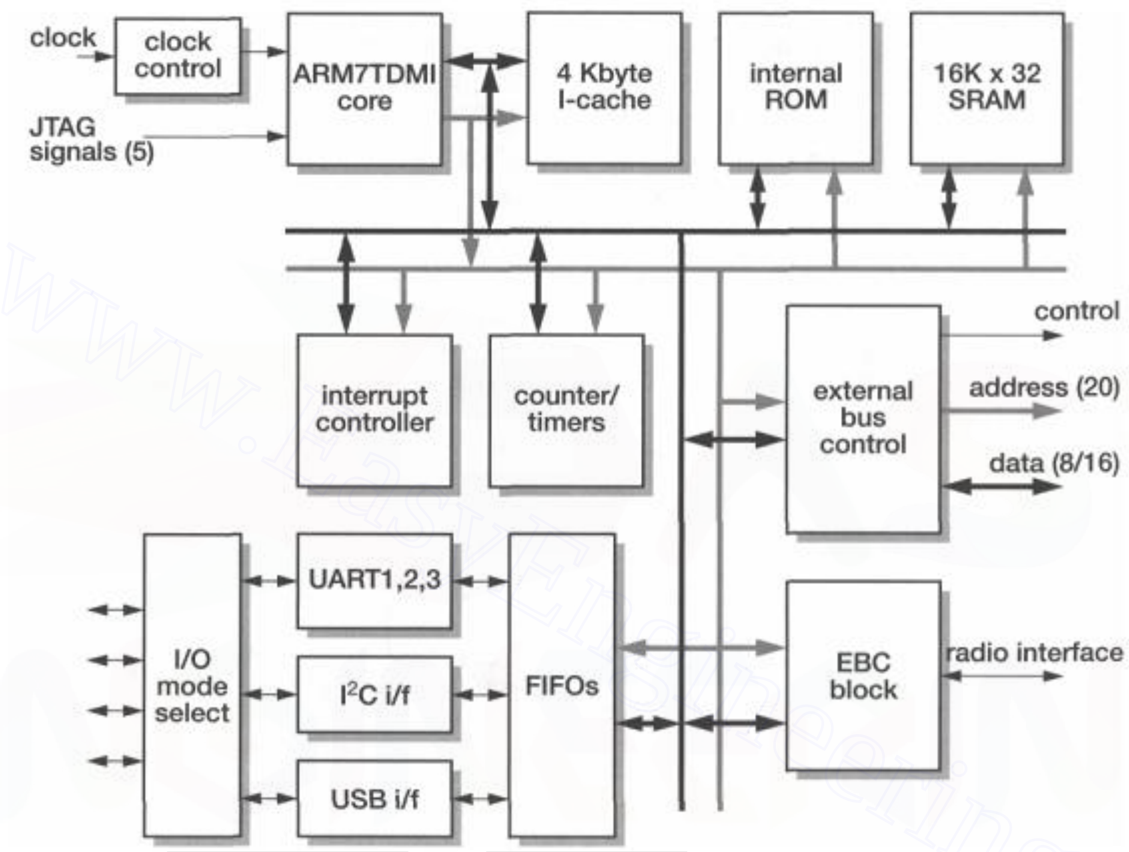
→The standard is intended to support short-range communication (from 10cm to 10m range) in a manner similar to that currentlyachieved with infrared communication using the IrDA standard, but avoiding the line-of-sight, alignment, and mutual interference restrictions of IrDA.

→Using radio communica-tion, Bluetooth is intended to support laptop to cellular telephone, printer, PDA, desktop, fax machines, keyboards, and so on,and it can also provide a bridge to existing data networks.

→It serves as a cable replacement technology for per-sonal networks.The standard supports a gross data rate of1 Mbit/s, and uses a frequency hopping scheme and forward error correction to give robust communication in a noisy and uncoordinated environment.

→The Ericsson-VLSI Bluetooth Baseband Controller chip is a jointly developed stand-ard part which is intended for use in portable Bluetooth-based communication devices.

Bluetooth 'piconet':



→Bluetooth units dynamically form ad hoc 'piconets', which are groups of two to eight units that operate the same frequency-hopping scheme.

→All of the units are equal peers with identical implementations, though one of the units will operate as master when the piconet is established.

→The master defines the clock and hopping sequence that synchronize the piconet. Multiple piconets can be linked to form a 'scatternet'.

Bluetooth controller organization:

→The chip is based around a synthesized ARM7TDMI core and includes 64 Kbytes of fast (zero wait state) on-chip SRAM and a 4K byte instruction cache. Critical routines can be loaded into the RAM to get the best performance.

→The cache improves the performance and power-efficiency of code resident in the off-chip memory. There is a set of peripheral modules which share a number of pins,

Fig: Ericsson Blue tooth core

Including three UARTs, a USB interface and an I2C-bus interface. FIFO buffers decouple the processor from having to respond to every byte which is transferred through these interfaces. The external bus interface supports devices with 8- and 16-bit data buses and has flexible wait state generation.

→The counter timer block has three 8-bit counters connected to a 24-bit prescaler, and an interrupt controller gives control of all on- and off-chip interrupt sources.

Ericsson Bluetooth Core:

→The Bluetooth Baseband Controller includes a power-optimized hardware block, the Ericsson Bluetooth Core (EBC), which handles all the Link Controller functionality within the Bluetooth specification and includes the interface logic to a Bluetooth radio implementation.

→The EBC performs all the packet-handling functions for point-to-point, multislot and point-to-multipoint communications. The baseband protocol uses a combination of circuit and packet switching. Slots can be reserved for synchronous channels, for example to support voice transmission.

Power Management:

The chip has four power management modes:

1. On-line: all blocks are clocked at their normal speed. The ARM7TDMI core clock is between 13 and 40 MHz, depending on the application. At the maximum data transfer rate the current consumption is around 30 mA.

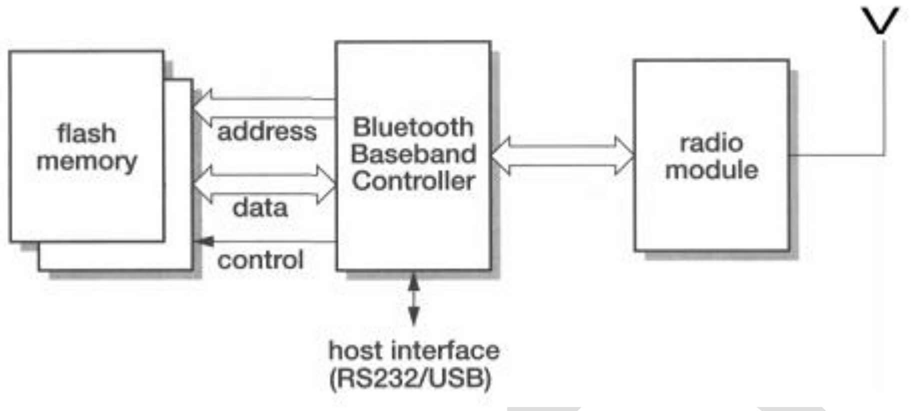
2. Command: The ARM7TDMI clock is slowed by the insertion of wait states.

3. Sleep: The ARM7TDMI clock is stopped, as are the clocks to a programmable subset of the other blocks. The current drawn in this mode is around 0.3 mA.

4. Stopped: The clock oscillator is turned off.

Bluetooth system:

The baseband controller chip requires an external radio module and program ROM to complete the system. The high level of integration leads to a very compact and economic implementation of a sophisticated and highly functional radio communication

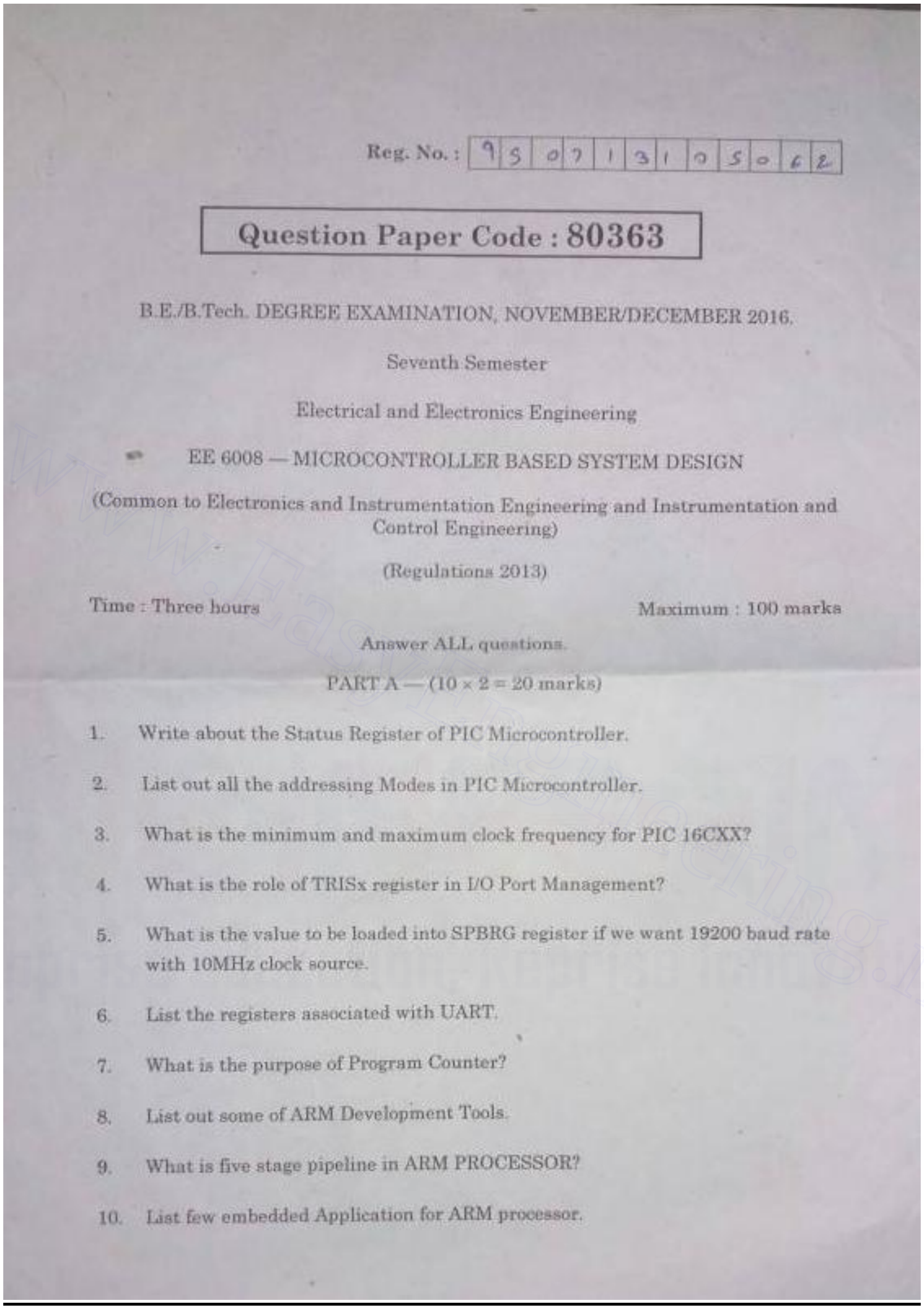


system.

Fig: Bluetooth

Bluetooth silicon:

The synthesized ARM7TDMI core in the top-right corner of the chip has far less visible structure than the ARM7TDMI hard macro cell. The processor is capable of operating at up to 39 MHz, but the data shown in the table are representative of a typical GSM application. The chip I/Os operate at 3.3 V, but the core logic typically operates at 2.5



Reg. No. : 9 5 0 7 1 3 1 0 5 0 6 2

Question Paper Code : 80363

B.E./B.Tech. DEGREE EXAMINATION, NOVEMBER/DECEMBER 2016.

Seventh Semester

Electrical and Electronics Engineering

EE 6008 — MICROCONTROLLER BASED SYSTEM DESIGN

(Common to Electronics and Instrumentation Engineering and Instrumentation and Control Engineering)

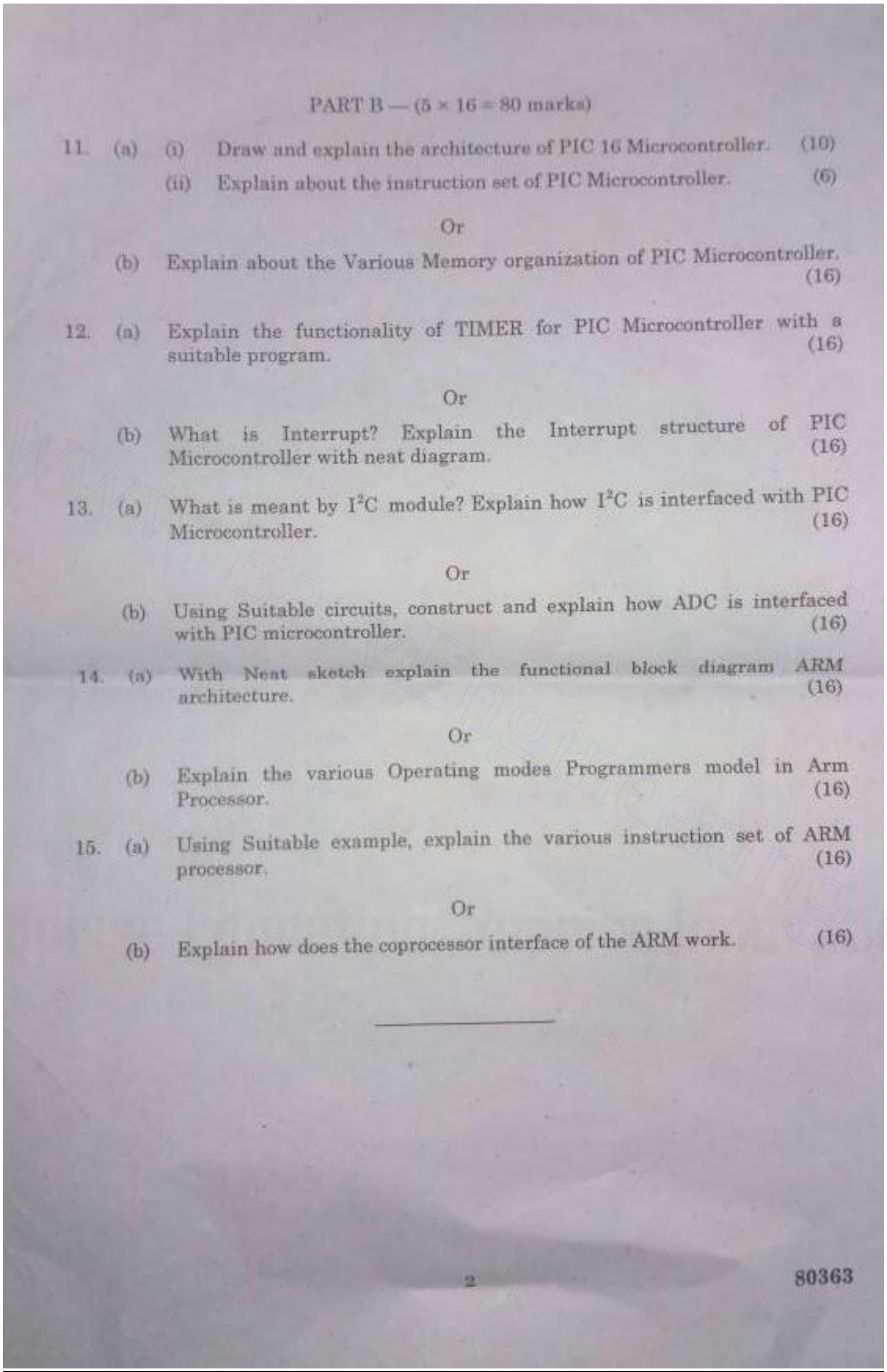
(Regulations 2013)

Time : Three hours Maximum : 100 marks

Answer ALL questions.

PART A — (10 × 2 = 20 marks)

1. Write about the Status Register of PIC Microcontroller.
2. List out all the addressing Modes in PIC Microcontroller.
3. What is the minimum and maximum clock frequency for PIC 16CXX?
4. What is the role of TRISx register in I/O Port Management?
5. What is the value to be loaded into SPBRG register if we want 19200 baud rate with 10MHz clock source.
6. List the registers associated with UART.
7. What is the purpose of Program Counter?
8. List out some of ARM Development Tools.
9. What is five stage pipeline in ARM PROCESSOR?
10. List few embedded Application for ARM processor.



PART B — (5 × 16 = 80 marks)

11. (a) (i) Draw and explain the architecture of PIC 16 Microcontroller. (10)

(ii) Explain about the instruction set of PIC Microcontroller. (6)

Or

(b) Explain about the Various Memory organization of PIC Microcontroller. (16)

12. (a) Explain the functionality of TIMER for PIC Microcontroller with a suitable program. (16)

Or

(b) What is Interrupt? Explain the Interrupt structure of PIC Microcontroller with neat diagram. (16)

13. (a) What is meant by I^2C module? Explain how I^2C is interfaced with PIC Microcontroller. (16)

Or

(b) Using Suitable circuits, construct and explain how ADC is interfaced with PIC microcontroller. (16)

14. (a) With Neat sketch explain the functional block diagram ARM architecture. (16)

Or

(b) Explain the various Operating modes Programmers model in Arm Processor. (16)

15. (a) Using Suitable example, explain the various instruction set of ARM processor. (16)

Or

(b) Explain how does the coprocessor interface of the ARM work. (16)

———————

2 80363