Velammal Institute of Technology

DEPARTMENT OF ELECTRICAL AND ELECTRONICS ENGINEERING

Focussed question bank

EE8018 – Microcontroller based system design

Year: IV **Semester : VII**

UNIT – I Introduction to PIC Microcontroller

PART – A

1. What is Microcontroller?

A device which contains the microprocessor with integrated peripherals like memory, serial ports, parallel ports, timer/counter, interrupt controller, data acquisition interfaces like ADC, DAC is called microcontroller.

2. What are the differences between a Microcontroller and Microprocessor?

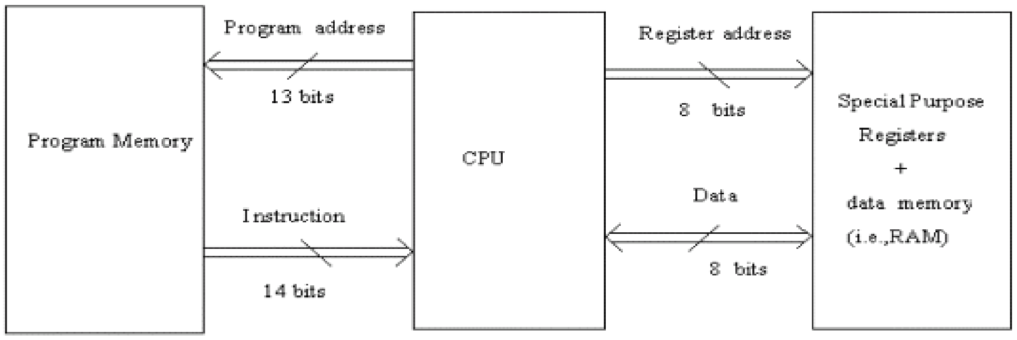
Microprocessor	Microcontroller
It is termed as general purpose digital computer.	It is termed as special purpose digital controller.
It contains the CPU, memory, addressing circuits and interrupt handling circuit.	It possesses all features of microprocessor and additionally it includes timers, parallel and serial I/O and the internal RAM and ROM.
It has one or two types of bit handling instructions.	It has many bit handling instructions.

3. What is PIC Microcontroller?

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.

4. Draw the CPU architecture of PIC Microcontroller.

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.

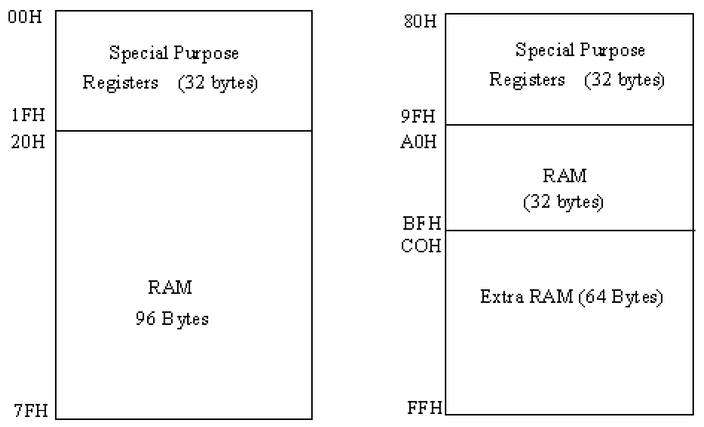


CPU Architecture of PIC microcontroller

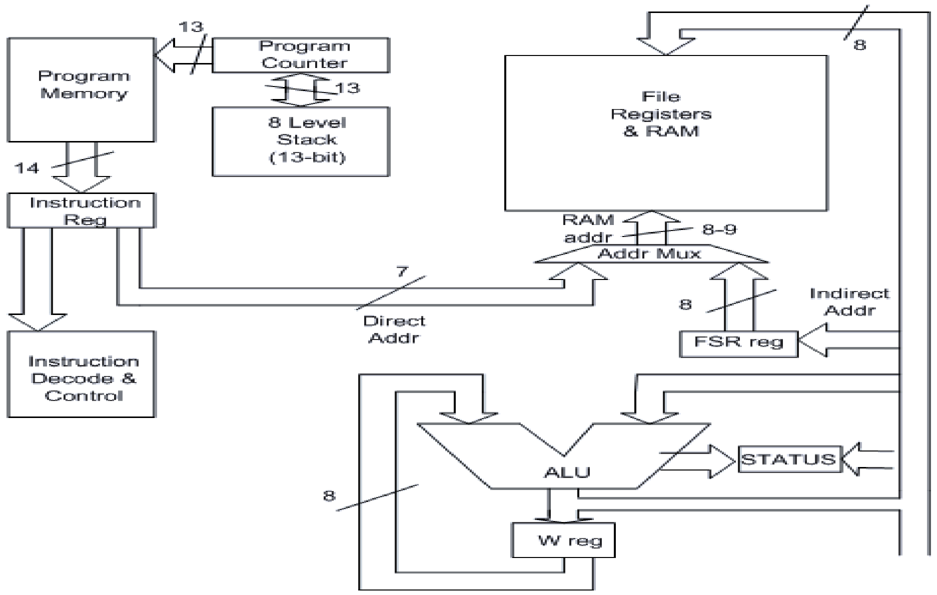
5. What is Special Purpose register file?

The special purpose register file contains input and output ports as well as the control registers used to establish each bit of a port as either an input or an output. It contains registers that provide the data input and data output to the variety of resources on the chip, such as the timers, the serial ports and the ADC. It has registers that contain control bits for selecting the mode of operation of a chip resource as well as enabling or disabling its operation. It has registers containing status bits, which denote the state of one of these chip resources.

6.Give the register file structure of PIC Microcontroller.



6. **Draw the architecture of PIC 16C74A**



7. **What is 'W' register in PIC Microcontroller?**

W, the working register, is used by many instructions as the source of an operand. This is similar to accumulator in 8051. It may also serve as the destination for the result of the instruction execution. It is an 8 - bit register.

W, Working register

8. **Give the status register of PIC Microcontroller. .(Nov/Dec 2016)**

The STATUS register is an 8-bit register that stores the status of the processor. This also stores carry, zero and digit carry bits.

STATUS - address 03H, 83H

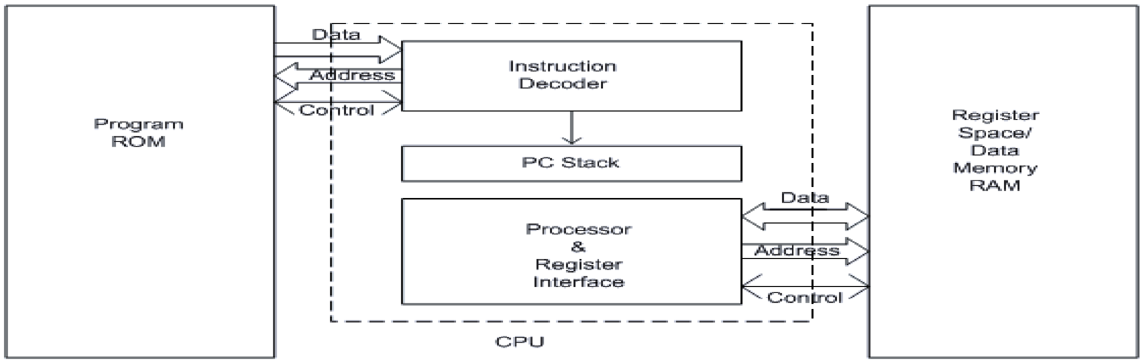
7	6	5	4	3	2	1	0
0	0	RP0	NOT_TO	NOT_PD	Z	DC	C

9. **What is program counter stack?**

Program Counter Stack

An independent 8-level stack is used for the program counter. As the program counter is 13bit, the stack is organized as 8x13bit registers. When an interrupt occurs, the program counter is pushed onto the stack. When the interrupt is being serviced, other interrupts remain disabled. Hence, other 7 registers of the stack can be used for subroutine calls within an interrupt service routine or within the mainline program.

10. Draw the general block diagram of Harvard architecture.



11. What are the types of instruction set used in PIC microcontroller?

There are three types of instruction set used in PIC microcontroller.

1. Bit oriented instruction 2. Byte oriented instruction 3. Literal instructions.

12. What is bit and byte oriented instruction?

The **byte oriented instructions** that require two parameters (For example, movf f, F(W)) expect the f to be replaced by the name of a special purpose register (e.g., PORTA) or the name of a RAM variable (e.g., NUM1), which serves as the source of the operand. 'f' stands for file register. The F(W) parameter is the destination of the result of the operation. It should be replaced by:F, if the destination is to be the source register.
W, if the destination is to be the working register (i.e., Accumulator or W register).
The **bit oriented instructions** also expect parameters (e.g., btfsc f, b). Here 'f' is to be replaced by the name of a special purpose register or the name of a RAM variable. The 'b' parameter is to be replaced by a bit number ranging from 0 to 7.
For example:
Z equ 2
btfsc STATUS, Z

Z has been equated to 2. Here, the instruction will test the Z bit of the STATUS register and will skip the next instruction if Z bit is clear.

13. What are the addressing modes of PIC? .(Nov/Dec 2016)

Addressing is defined as how the operands are specified in the instruction. Direct addressing and indirect addressing mode.
Indirectly addressing the memory used in FSR and INDF instruction. Here the operand is specified indirectly in the instruction.

14.What do you mean by direct addressing mode and indirect addressing mode ?

It uses 7 bits of the instruction and the 8th bit from RP. It directly give the address where the data is present.ie, the address of the operand is given in the instruction.

15.What is instruction pipelining?

It allows the CPU to fetch and execute at the same time while executing one instruction, CPU will fetch next instruction to be executed.

PART-B

1. Explain with neat diagram the architecture of PIC16C6x and PIC16C7x microcontroller. (Nov/Dec 2016)

ARCHITECTURAL OVERVIEW

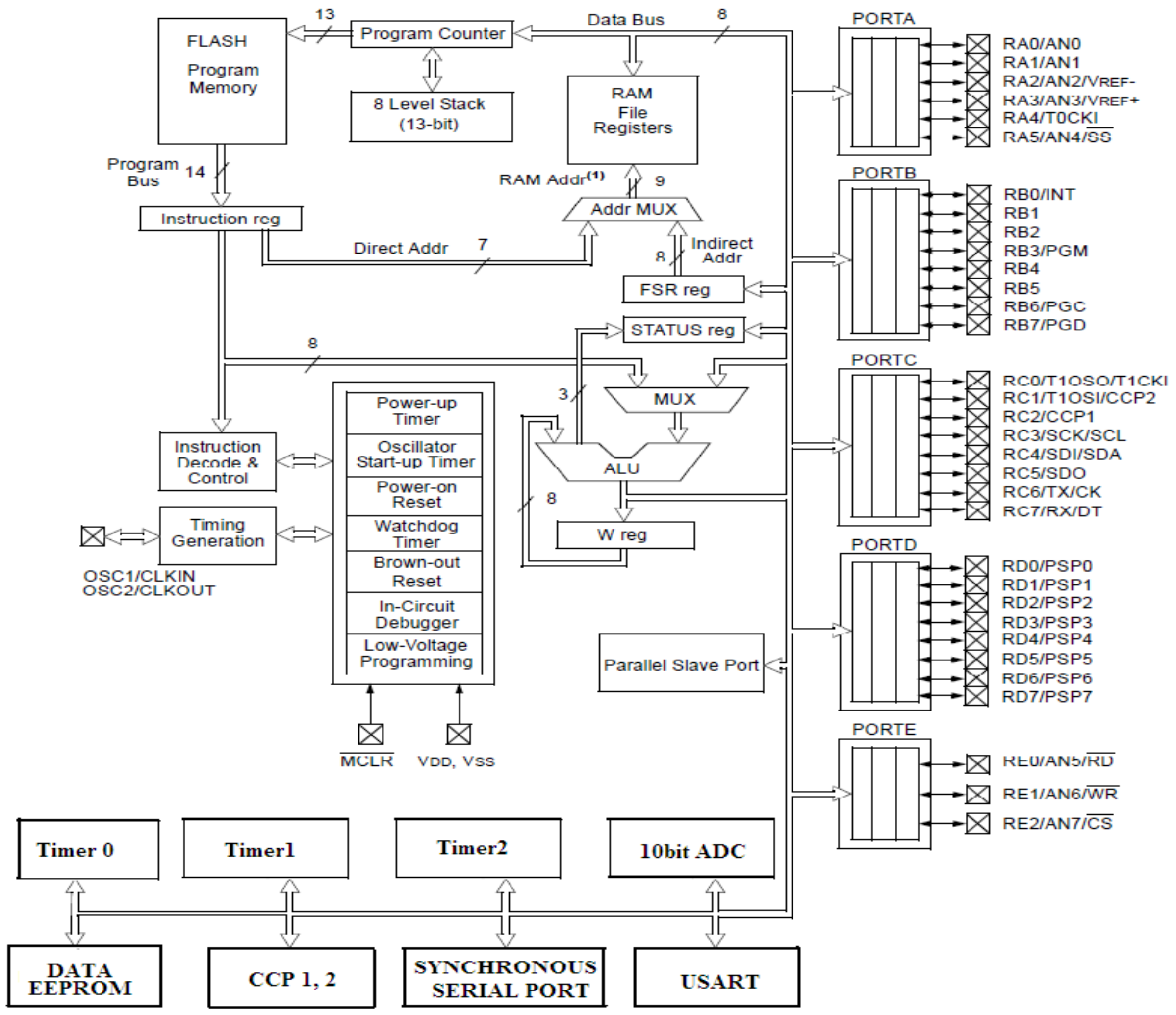
The high performance of the PIC16CXX family can be attributed to a number of architectural features com-monly found in RISC microprocessors. To begin with, the PIC16CXX uses a Harvard architecture, in which, program and data are accessed from separate memories using separate buses. This improves bandwidth over traditional von Neumann architecture where pro-gram and data may be fetched from the same memory using the same bus. Separating program and data bus-ses further allows instructions to be sized differently than 8-bit wide data words. Instruction opcodes are 14-bits wide making it possible to have all single word instructions. A 14-bit wide program memory access bus fetches a 14-bit instruction in a single cycle. A two-stage pipeline overlaps fetch and execution of instruc-tions (Example 3-1). Consequently, all instructions exe-cute in a single cycle (200 ns @ 20 MHz) except for program branches.
The PIC16C61 addresses 1K x 14 of program memory. The PIC16C62/62A/R62/64/64A/R64 address 2K x 14 of program memory, and the PIC16C63/R63/65/65A/R65 devices address 4K x 14 of program memory. The PIC16C66/67 address 8K x 14 program memory. All program memory is internal.

The PIC16CXX can directly or indirectly address its register files or data memory. All special function reg-isters including the program counter are mapped in the data memory. The PIC16CXX has an orthogonal (symmetrical) instruction set that makes it possible to carry out any operation on any register using any addressing mode. This symmetrical nature and lack of “special optimal situations” makes programming with the PIC16CXX simple yet efficient, thus significantly reducing the learning curve.
The PIC16CXX device contains an 8-bit ALU and work-ing register (W). The ALU is a general purpose arithme-tic unit. It performs arithmetic and Boolean functions between data in the working register and any register file.

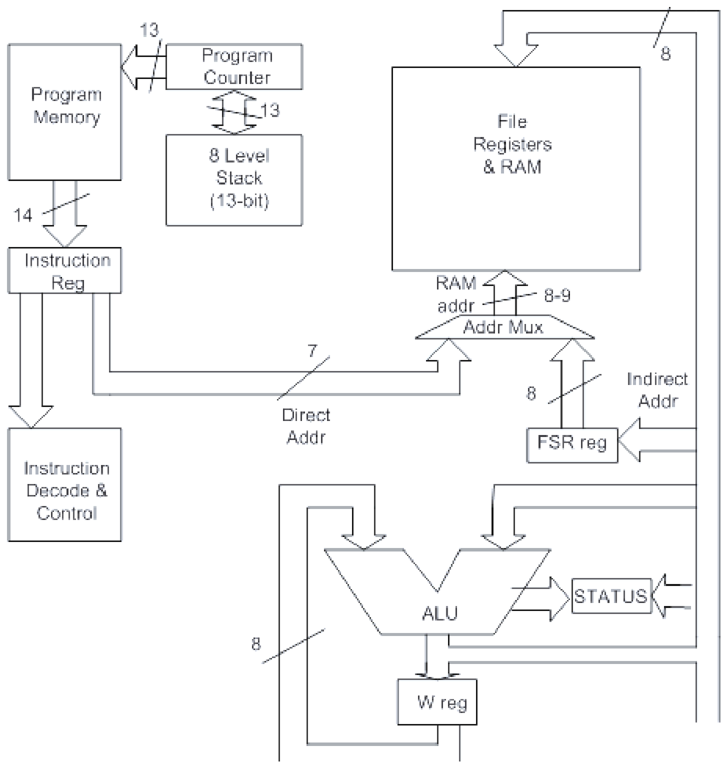
The ALU is 8-bits wide and capable of addition, sub-traction, shift, and logical operations. Unless otherwise mentioned, arithmetic operations are two's comple-ment in nature. In two-operand instructions, typically one operand is the working register (W register), the other operand is a file register or an immediate con-stant. In single operand instructions, the operand is either the W register or a file register.

The W register is an 8-bit working register used for ALU operations. It is not an addressable register.

Depending upon the instruction executed, the ALU may affect the values of the Carry (C), Digit Carry (DC), and Zero (Z) bits in the STATUS register. Bits C and DC operate as a borrow and digit borrow out bit, respec-tively, in subtraction. See the SUBLW and SUBWF instructions for examples.



2. Explain with neat diagram the block diagram of PIC16C6x and PIC16C7x microcontroller



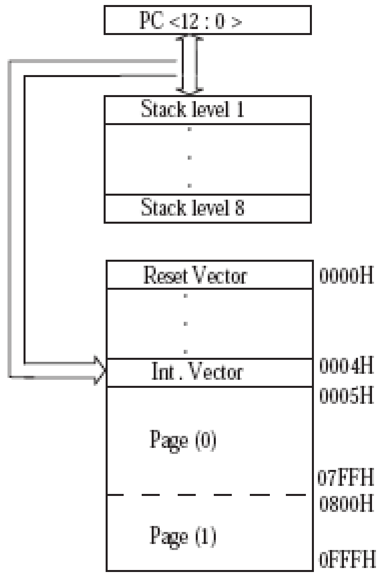
3. **With a neat diagram discuss in detail about memory organization of a PIC microcontroller.**

Memory Organization

The PIC 16C7X family has a 13-bit program counter capable of addressing 8k×14 program memory. PIC16C74A has 4k×14 program memory. For those devices with less than 8k program memory, accessing a location above the physically implemented address will cause a wraparound.

Program memory map and stack

16C74A has 4k program memory. The address range is 0000H - 0FFFH. The reset vector is 0000H and the interrupt vector is 0004H.



4. Explain in detail the register file structure and addressing modes of PIC microcontroller.

REGISTER FILE STRUCTURE

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.

ADDRESSING MODES.

The PIC microcontrollers support only TWO addressing modes .They are

(i) Direct Addressing Mode
(ii) Indirect Addressing mode

Direct Addressing Mode :

In direct addressing mode 7 bits (0-6) of the instruction identify the register file address and the 8 th bit of the register file address register bank select bit(RP0).

The above diagram explains the method of accessing register file address 13H by direct addressing method.
Indirect Addressing Mode
In the indirect addressing mode the 8-bit register file address is first written into a Special Function Register(SFR) which acts as a pointer to any address location in the register file.A subsequent direct access of INDF will actually access the register file using the content of FSR as a pointer to the desired location of the operand.

5. Explain the instruction set of PIC microcontroller. (Nov/Dec 2016)

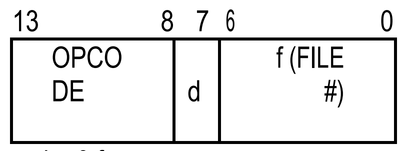
Each PIC16CXX instruction is a 14-bit word divided into an OPCODE which specifies the instruction type and one or more operands which further specify the operation of the instruction. The PIC16CXX instruction set summary lists **byte-oriented**, **bit-ori-ented**, and **literal and control** operations. Table 14-1 shows the opcode field descriptions.
For **byte-oriented** instructions, 'f' represents a file reg-ister designator and 'd' represents a destination desig-nator. The file register designator specifies which file register is to be used by the instruction.
The destination designator specifies where the result of the operation is to be placed. If 'd' is zero, the result is placed in the W register. If 'd' is one, the result is placed in the file register specified in the instruction.
For **bit-oriented** instructions, 'b' represents a bit field designator which selects the number of the bit affected by the operation, while 'f' represents the number of the file in which the bit is located.
For **literal and control** operations, 'k' represents an eight or eleven bit constant or literal value. The instruction set is highly orthogonal and is grouped into three basic categories:

- **Byte-oriented** operations
- **Bit-oriented** operations
- **Literal and control** operations

All instructions are executed within one single instruction cycle, unless a conditional test is true or the pro-gram counter is changed as a result of an instruction. In this case, the execution takes two instruction cycles with the second cycle executed as a NOP. One instruc-tion cycle consists of four oscillator periods. Thus, for an oscillator frequency of 4 MHz, the normal instruction execution time is 1 μs. If a conditional test is true or the program counter is changed as a result of an instruc-tion, the instruction execution time is 2 μs.

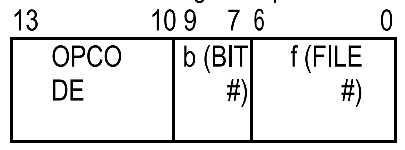
: GENERAL FORMAT FOR INSTRUCTIONS

Byte-oriented file register operations



d = 0 for destination W
d = 1 for destination f
f = 7-bit file register address

Bit-oriented file register operations



b = 3-bit

bit address = 7bit
file register
f Address

Literal and control operations

General

13 — 8	7 — 0
OPCODE	k (literal)

= 8-bit immediate
k value

CALL and GOTO
instructions only

13 — 11	10 — 0
OPCODE	k (literal)

= 11-bit
k immediate value

PIC16CXX INSTRUCTION SET

Mnemonic, Operands		**Description**	**Cycles**	**14-Bit Opcode** MSb … LSb			**Status Affected**	**Notes**
BYTE-ORIENTED FILE REGISTER OPERATIONS								
ADDWF	**f, d**	Add W and f	1	00	0111 dfff	ffff	C,DC,Z	1,2
ANDWF	**f, d**	AND W with f	1	00	0101 dfff	ffff	Z	1,2
CLRF	**f**	Clear f	1	00	0001 lfff	ffff	Z	2
CLRW	**-**	Clear W	1	00	0001 0xxx	xxxx	Z	
COMF	**f, d**	Complement f	1	00	1001 dfff	ffff	Z	1,2
DECF	**f, d**	Decrement f	1	00	0011 dfff	ffff	Z	1,2
DECFSZ	**f, d**	Decrement f, Skip if 0	1(2)	00	1011 dfff	ffff		1,2,3
INCF	**f, d**	Increment f	1	00	1010 dfff	ffff	Z	1,2
INCFSZ	**f, d**	Increment f, Skip if 0	1(2)	00	1111 dfff	ffff		1,2,3
IORWF	**f, d**	Inclusive OR W with f	1	00	0100 dfff	ffff	Z	1,2
MOVF	**f, d**	Move f	1	00	1000 dfff	ffff	Z	1,2
MOVWF	**f**	Move W to f	1	00	0000 lfff	ffff		
NOP	**-**	No Operation	1	00	0000 0xx0	0000		

RLF	**f, d**	Rotate Left f through Carry	1	00 1101 dfff ffff	C	1,2
RRF	**f, d**	Rotate Right f through Carry	1	00 1100 dfff ffff	C	1,2
SUBWF	**f, d**	Subtract W from f	1	00 0010 dfff ffff	C,DC,Z	1,2
SWAPF	**f, d**	Swap nibbles in f	1	00 1110 dfff ffff		1,2
XORWF	**f, d**	Exclusive OR W with f	1	00 0110 dfff ffff	Z	1,2
BIT-ORIENTED FILE REGISTER OPERATIONS						
BCF	**f, b**	Bit Clear f	1	01 00bb bfff ffff		1,2
BSF	**f, b**	Bit Set f	1	01 01bb bfff ffff		1,2
BTFSC	**f, b**	Bit Test f, Skip if Clear	1 (2)	01 10bb bfff ffff		3
BTFSS	**f, b**	Bit Test f, Skip if Set	1 (2)	01 11bb bfff ffff		3
LITERAL AND CONTROL OPERATIONS						
ADDLW	**k**	Add literal and W	1	11 111x kkkk kkkk	C,DC, Z	
ANDLW	**k**	AND literal with W	1	11 1001 kkkk kkkk	Z	
CALL	**k**	Call subroutine	2	10 0kkk kkkk kkkk		
CLRWDT	**-**	Clear Watchdog Timer	1	00 0000 0110 0100	$\overline{TO}$, $\overline{PD}$	
GOTO	**k**	Go to address	2	10 1kkk kkkk kkkk		
IORLW	**k**	Inclusive OR literal with W	1	11 1000 kkkk kkkk	Z	
MOVLW	**k**	Move literal to W	1	11 00xx kkkk kkkk		
RETFIE	**-**	Return from interrupt	2	00 0000 0000 1001		
RETLW	**k**	Return with literal in W	2	11 01xx kkkk kkkk		
RETURN	**-**	Return from Subroutine	2	00 0000 0000 1000		
SLEEP	**-**	Go into standby mode	1	00 0000 0110 0011	$\overline{TO}$, $\overline{PD}$	
SUBLW	**k**	Subtract W from literal	1	11 110x kkkk kkkk	C,DC, Z	
XORLW	**k**	Exclusive OR literal with W	1	11 1010 kkkk kkkk	Z	

Note 1: When an I/O register is modified as a function of itself (e.g., MOVF PORTB, 1), the value used will be that value present on the pins themselves. For example, if the data latch is '1' for a pin configured as input and is driven low by an external device, the data will be written back with a '0'.

2: If this instruction is executed on the TMR0 register (and, where applicable, d = 1), the prescaler will be cleared if assigned to the Timer0 Module.

3: If Program Counter (PC) is modified or a conditional test is true, the instruction requires two cycles. The second cycle is executed as a NOP.

6. Explain in detail the concept of pipelining of instructions in PIC microcontroller.

Instruction Flow/Pipelining

An “Instruction Cycle” consists of four Q cycles (Q1, Q2, Q3, and Q4). The instruction fetch and execute are pipelined such that fetch takes one instruction cycle while decode and execute takes another instruction cycle. However, due to the pipelining, each instruction effectively executes in one cycle. If an instruction causes the program counter to change (e.g. GOTO) then two cycles are required to complete the instruction (Example 3-1).

A fetch cycle begins with the program counter (PC) incrementing in Q1.

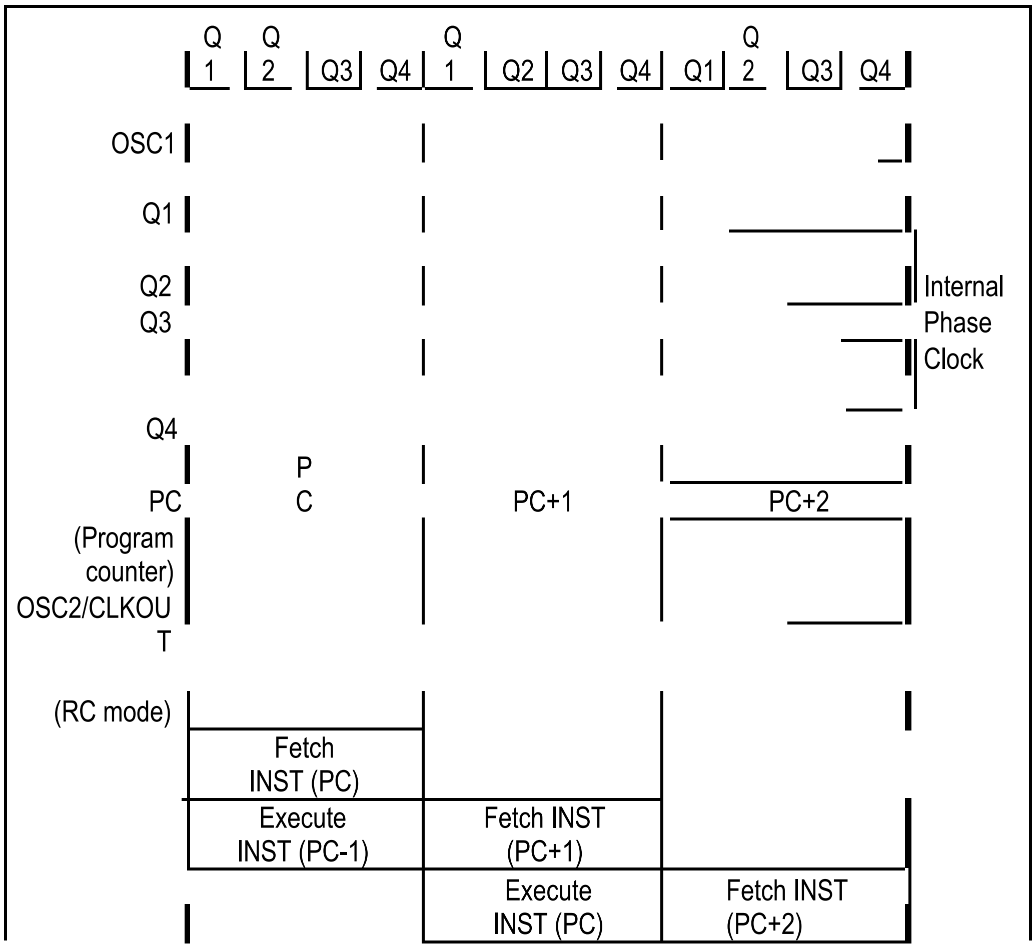
In the execution cycle, the fetched instruction is latched into the “Instruction Register (IR)” in cycle Q1. This instruction is then decoded and executed during the Q2, Q3, and Q4 cycles. Data memory is read during Q2 (operand read) and written during Q4 (destination write).
CLOCK/INSTRUCTION CYCLE

Instruction Flow/Pipelining

An “Instruction Cycle” consists of four Q cycles (Q1, Q2, Q3, and Q4). The instruction fetch and execute are pipelined such that fetch takes one instruction cycle while decode and execute takes another instruction cycle. However, due to the pipelining, each instruction effectively executes in one cycle. If an instruction causes the program counter to change (e.g. GOTO) then two cycles are required to complete the instruction (Example 3-1).

A fetch cycle begins with the program counter (PC) incrementing in Q1.

In the execution cycle, the fetched instruction is latched into the “Instruction Register (IR)” in cycle Q1. This instruction is then decoded and executed during the Q2, Q3, and Q4 cycles. Data memory is read during Q2 (operand read) and written during Q4 (destination write).
CLOCK/INSTRUCTION CYCLE



Execute INST (PC+1)

INSTRUCTION PIPELINE FLOW

	Tcy0	Tcy1	Tcy2	Tcy3	Tcy4	Tcy5
1. MOVLW 55h	Fetch 1	Execute 1				
2. MOVWF PORTB		Fetch 2	Execute 2			
3. CALL SUB_1			Fetch 3	Execute 3		
4. BSF PORTA, BIT3 (Forced NOP)				Fetch 4	Flush	
5. Instruction @ address SUB_1					Fetch SUB_1	Execute SUB_1

All instructions are single cycle, except for any program branches. These take two cycles since the fetch
instruction is “flushed” from the pipeline while the new instruction is being fetched and then executed.

UNIT- II INTERRUPTS AND TIMERS
PART A

1. What are the interrupts available in PIC? (Jan'14)

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	

2.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

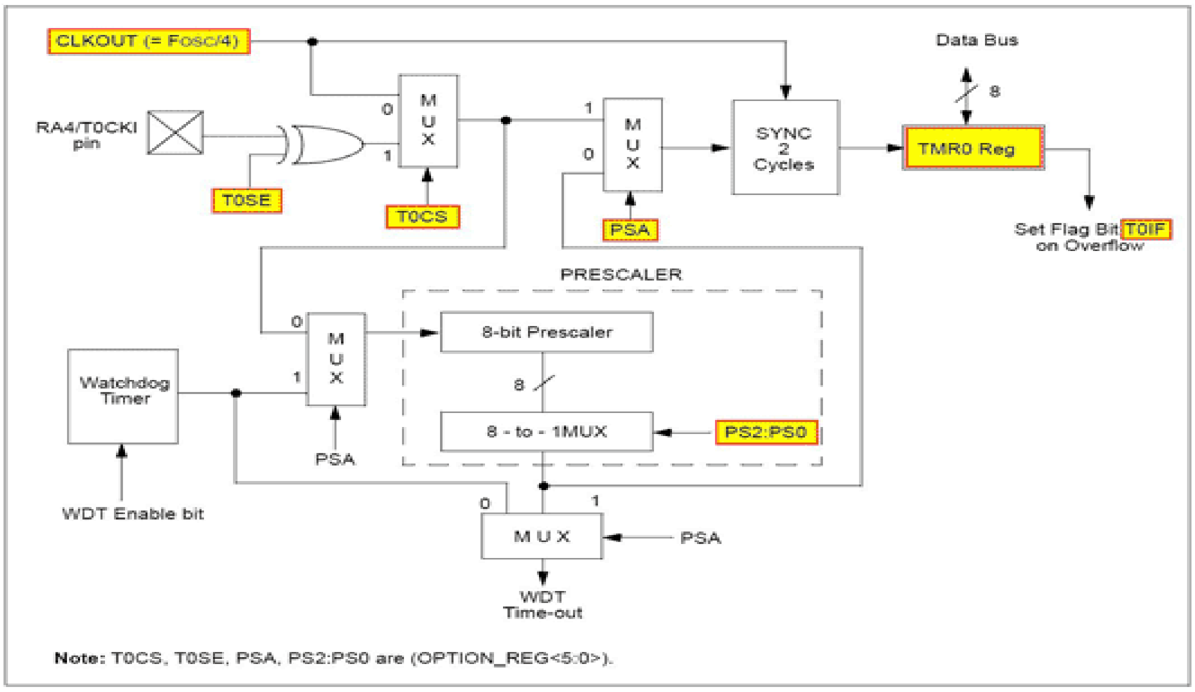
3.What are the features of timer 1?

- 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

4. 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**.**

5.Draw the block diagram of PIC timer 0.

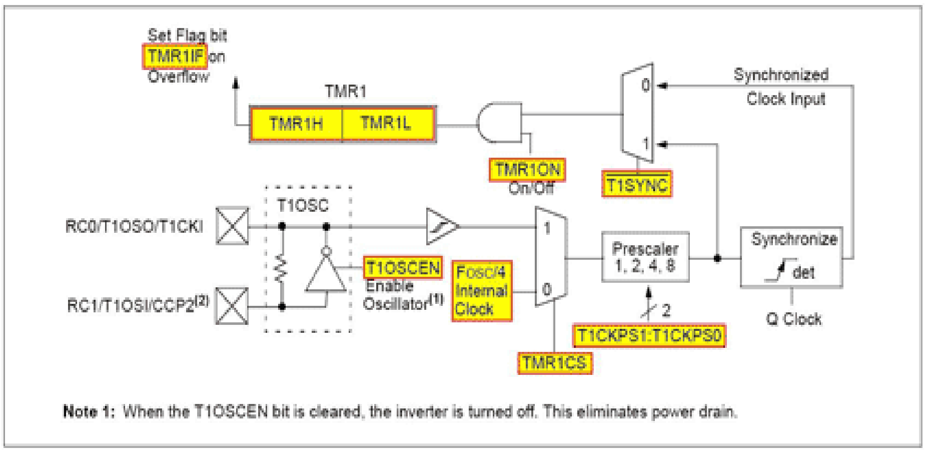


6.What is T1CON register ?

T1CON: TIMER1 CONTROL REGISTER (ADDRESS 10h)

U-0	U-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
—	—	T1CKPS1	T1CKPS0	T1OSCEN	$\overline{\text{T1SYNC}}$	TMR1CS	TMR1ON
bit 7							bit 0

7.Give the block diagram of timer 1.



Note 1: When the T1OSCEN bit is cleared, the inverter is turned off. This eliminates power drain.

8. Give the diagram of a state machine

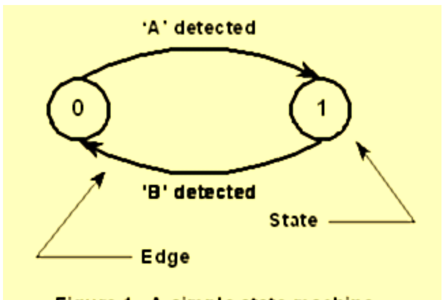


Figure 1. A simple state machine.

9. Brief the 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.

10. 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.

11. What is role of TRISX register? (Nov/Dec 2016)

It is called as data direction register which provides an access to the data flow through the respective ports initialized.

12. What is the minimum and maximum clock frequency of PIC16CXX?(Nov/Dec 2016)

It can operate from 1Mz to 33MHz.

UNIT- II INTERRUPTS AND TIMERS
PART-B

1. **Explain the concepts of interrupts in detail. (Nov/Dec 2016)**

INTERRUPTS :

The PIC family has up to 11 sources of interrupt. The interrupt control register (INTCON) records individual interrupt requests in flag bits. It also has individual and global interrupt enable bits.

Global interrupt enable bit, GIE enables all un-masked interrupts or disables all interrupts. When bit GIE is enabled, and an interrupt flag bit and mask bit are set, the interrupt will vector immediately. Individual interrupts can be disabled through their corresponding enable bits in the INTCON register. GIE is cleared on reset.

The “return from interrupt” instruction, RETFIE, exits the interrupt routine as well as sets the GIE bit, which re-enable interrupts.

The RBO/INT pin interrupt, the RB port change interrupt and the TMR0 overflow interrupt flag bits are contained in the INTCON register.

The peripheral interrupt flag bits are contained in special function registers PIR1 and PIR2. The corresponding interrupt enable bits are contained in special function registers PIE1 and PIE2 and the peripheral interrupt enable bit is contained in special function register INTCON.

When an interrupt is responded to, bit GIE is cleared to disable any further interrupts, the return address is pushed onto the stack and the PC is loaded with 0004h. Once in the interrupt service routine the source(s) of the interrupt can be determined by polling the interrupt flag bits. The interrupt flag bit(s) must be cleared in software before re-enabling interrupts to avoid recursive interrupts.

For external interrupt events, such as the RB0/INT pin or RB port change interrupt, the interrupt latency will be three or four instruction cycles. The exact latency depends when the interrupt event occurs. The latency is the same for one or two cycle instructions. Once in the interrupt service routine the source(s) of the interrupt can be determined by polling the interrupt flag bits. The interrupt flag bit(s) must be cleared in software before re-enabling interrupts to avoid infinite interrupt requests. Individual interrupt flag bits are set regardless of the status of their corresponding mask bit or the GIE bit.

INT INTERRUPT :

External interrupt on RB0/INT pin is edge triggered: either rising if edge select bit INTEDG is set, or falling, if bit INTEDG is clear. When a valid edge appears on the RB0/INT pin, flag bit INTF is set. This interrupt can be disabled by clearing enable bit INTE. The INTF bit must be cleared in software in the interrupt service routine before re-enabling this interrupt. The INT interrupt can wake the processor from SLEEP, if enable bit INTE was set prior to going into SLEEP. The status of global enable bit GIE decides whether or not the processor branches to the interrupt vector following wake-up. See for details on SLEEP mode.

TMR0 INTERRUPT:

An overflow in the TMR0 register will set flag bit T0IF. The interrupt can be enabled/disabled by setting/clearing enable bit T0IE.

PORTB INTERRUPT ON CHANGE :

An input change on PORTB sets flag bit RBIF. The interrupt can be enabled/disabled by setting/clearing enable bit RBIE.

WATCH DOG TIMER (WDT):

The Watchdog Timer is a free running on-chip RC oscillator which does not require any external components. This RC oscillator is separate from the RC oscillator of the OSC1/CLKIN pin. That means that the WDT will run, even if the clock on the OSC1/CLKIN and OSC2/CLKOUT pins of the device has been stopped, for example, by execution of a SLEEP instruction. During normal operation, a WDT time-out generates a device reset. If the device is in SLEEP mode, a WDT time-out causes the device to wake-up and continue with normal operation. The WDT can be permanently disabled by clearing configuration bit WDTE.

WDT PERIOD:

The WDT has a nominal time-out period of 18 ms, (with no prescaler). The time-out periods vary with temperature, VDD and process variations from part to part (see DC specs). If longer time-out periods are desired, a prescaler with a division ratio of up to can be assigned to the WDT under software control by writing to the OPTION register. Thus, time-out periods up to seconds can be realized.

The CLRWDT and SLEEP instructions clear the WDT and the postscaler, if assigned to the WDT, and prevent it from timing out and generating a device RESET condition.

The TO bit in the STATUS register will be cleared upon a WDT time-out.

WDT PROGRAMMING CONSIDERATIONS:

It should also be taken in account that under worst case conditions (VDD = Min., Temperature = Max., max WDT prescaler) it may take several seconds before a WDT time-out occurs.

1. **Explain timer 0 in detail with its registers.**

Timer-0 Overview

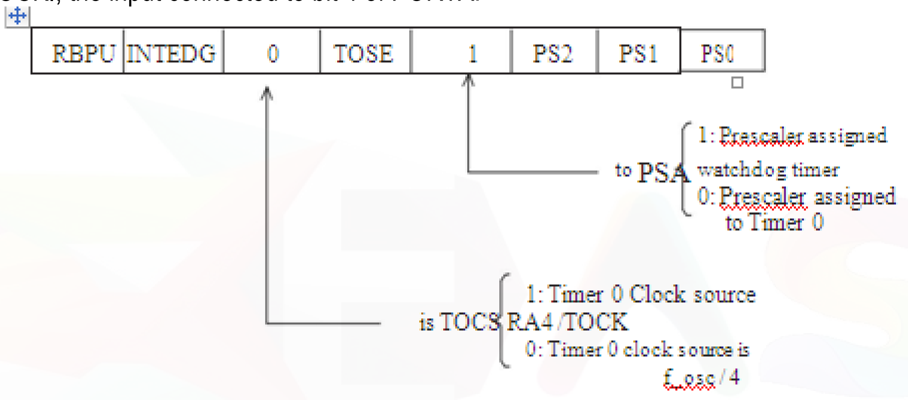
The Timer 0 module is a simple 8-bit overflow counter. The clock source can be either the internal clock ($f_{osc}/4$) or an external clock. When the clock source is an external clock, the Timer-0 module can be selected to increment on either the rising or falling edge.

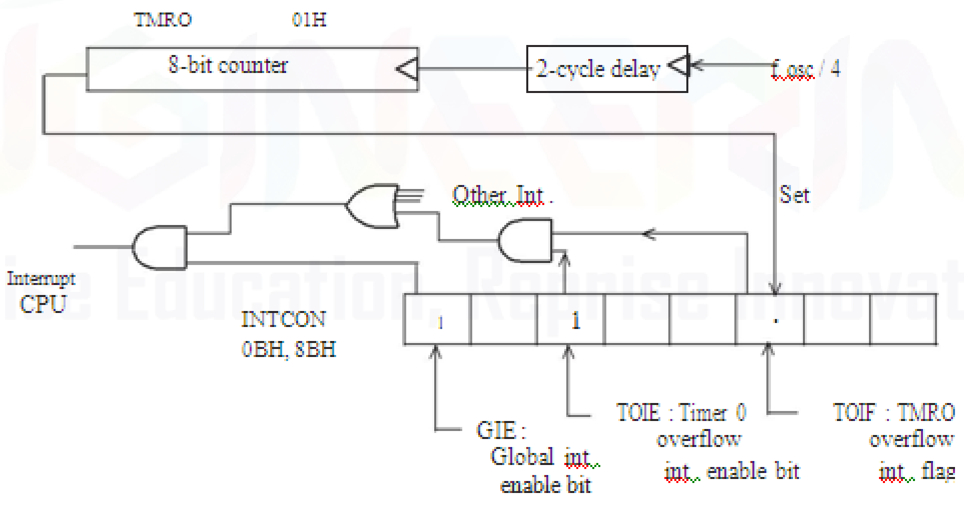
Timer-0 module also has a programmable prescalar option. This prescalar can be assigned either to Timer 0 or the watchdog Timer.

The counter sets a flag TOIF when it overflows and can cause an interrupt at that time if that interrupt source has been enabled (TOIF=1). Timer 0 can be assigned an 8-bit prescalar that can divide the input by 2,4,8,16,...,256. Writing to TMRO resets the prescalar assigned to it.

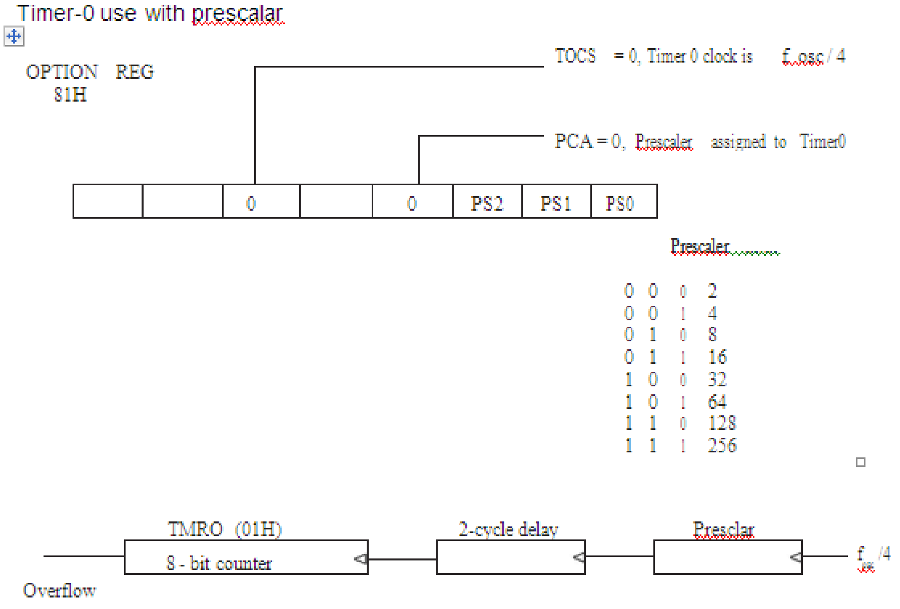
Timer-0, or its prescalar can be connected to either of two input sources

1. $f_{osc}/4$
2. RA4/ TOCKI, the input connected to bit 4 of PORTA.


RBPU
INTEDG
0
TOSE
1
PS2
PS1
PS0
to PSA
1: Prescaler assigned watchdog timer
0: Prescaler assigned to Timer 0
is TOCS
1: Timer 0 Clock source RA4/TOCK
0: Timer 0 clock source is f_osc / 4


TMRO
01H
8-bit counter
2-cycle delay
f_osc / 4
Set
Other Int
Interrupt CPU
INTCON 0BH, 8BH
1
1
GIE : Global int. enable bit
TOIE : Timer 0 overflow int. enable bit
TOIF : TMRO overflow int. flag

Timer-0 use with prescalar

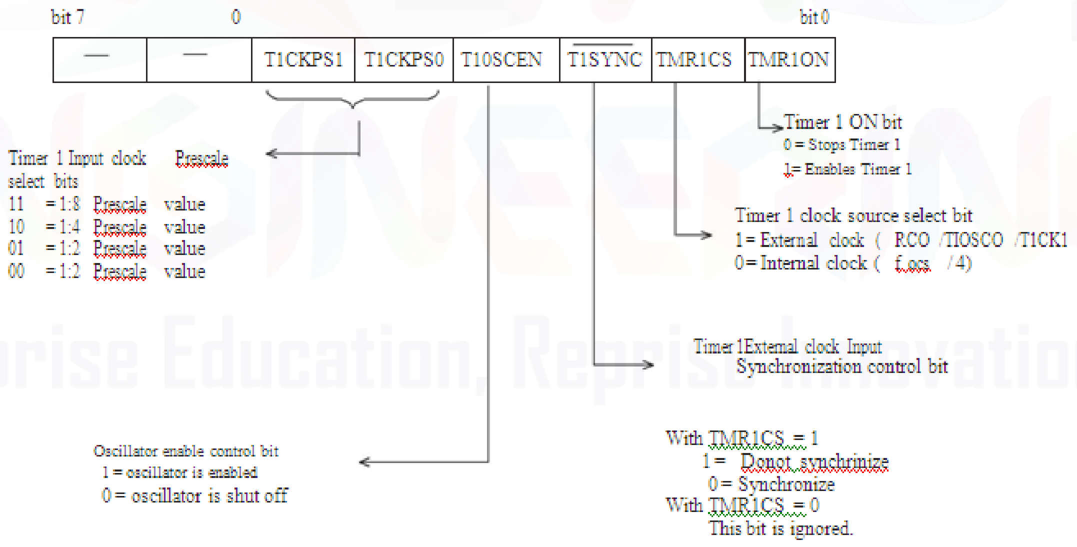


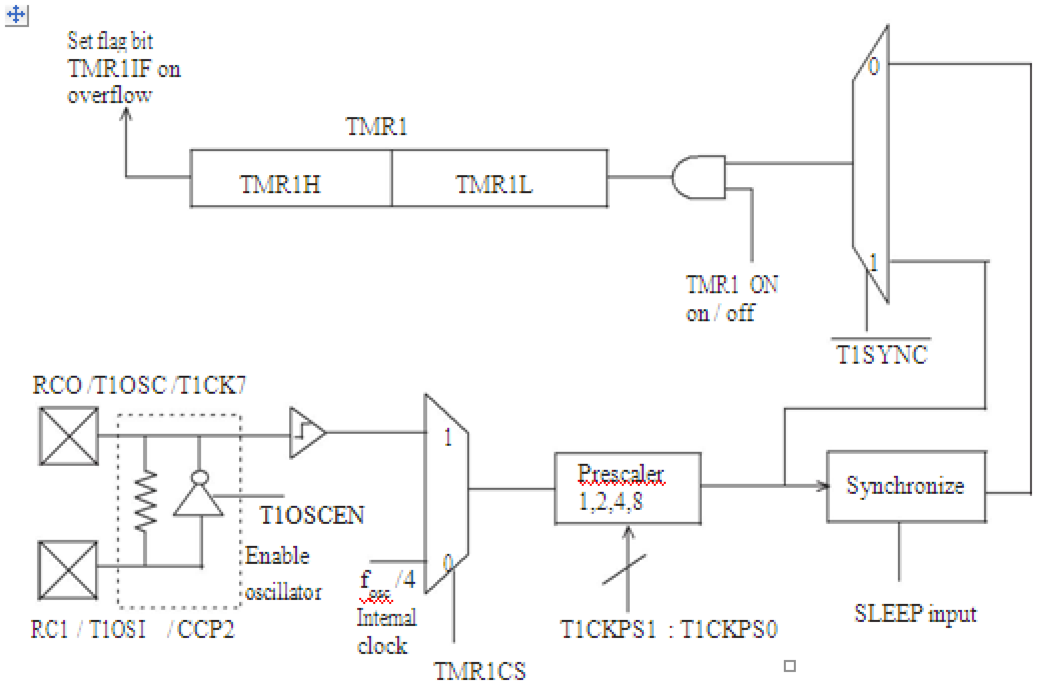
3. Explain in detail, the block diagram of timer 1 and its associated registers.

Timer-1 Module

The Timer1 module is a 16-bit timer/counter consisting of two 8-bit registers (TMR1H and TMR1L) which are readable and writable. The TMR1 register pair (TMR1H: TMR1L) increments from 0000H to FFFFH and rolls over to 0000H. The TMR1 interrupt, if enabled, is generated on overflow which sets the interrupt flag bit TMR1IF-(PIR< 0 >). This interrupt can be enabled/disabled by set-ting/clearing TMR1 interrupt enable bit TMR1IE-(PIE < 0 >)

The operating and control modes of Timer 1 is determined by the special purpose register T1CON. T1CON (10H)





Timer 1 can operate in one of the two modes

- As a timer. (TMR1CS = 0)

  In timer mode, Timer 1 increments in every instruction cycle. The Timer 1 clock source is $f_{osc}/4$. Since the internal clock is selected, the timer is always synchronized and there is no further need of synchronization.

- As a counter (TMR1CS = 1)

In counter mode, external clock input from the pin RCO/T1OSC/T1CKI is selected.

3. How timer2 is different from timer 0 and 1. Explain.

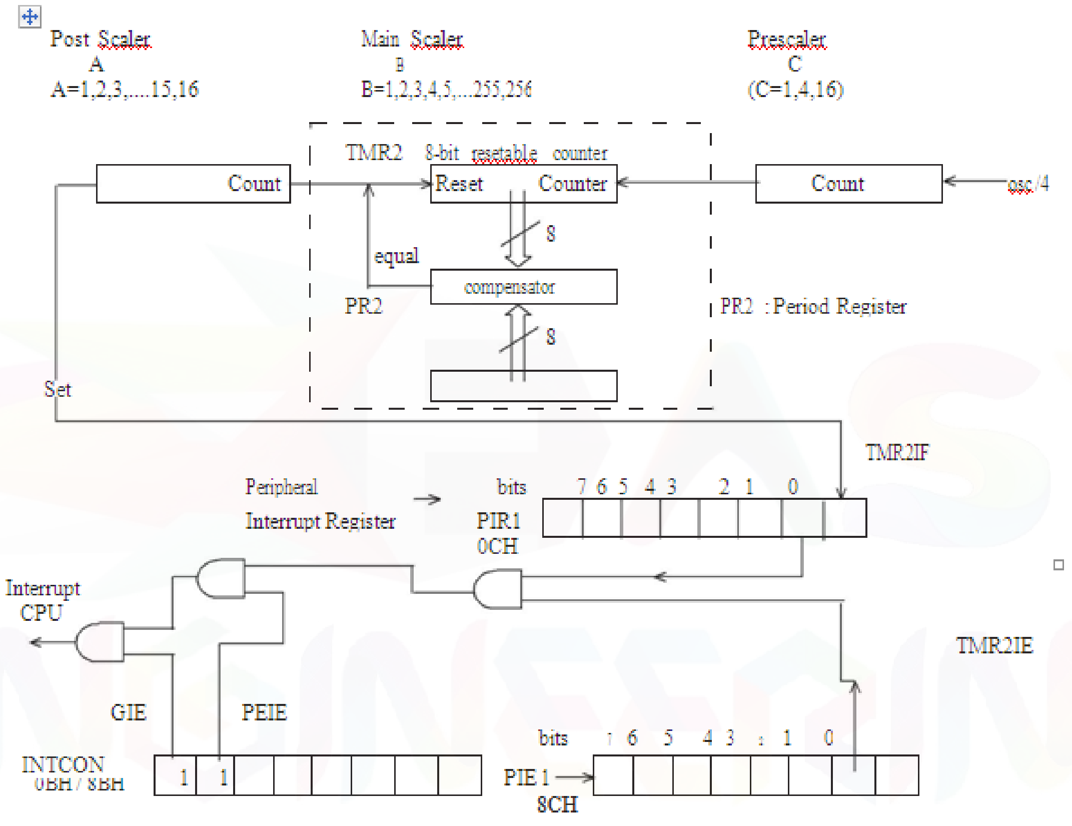
Use of Timer-2

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.

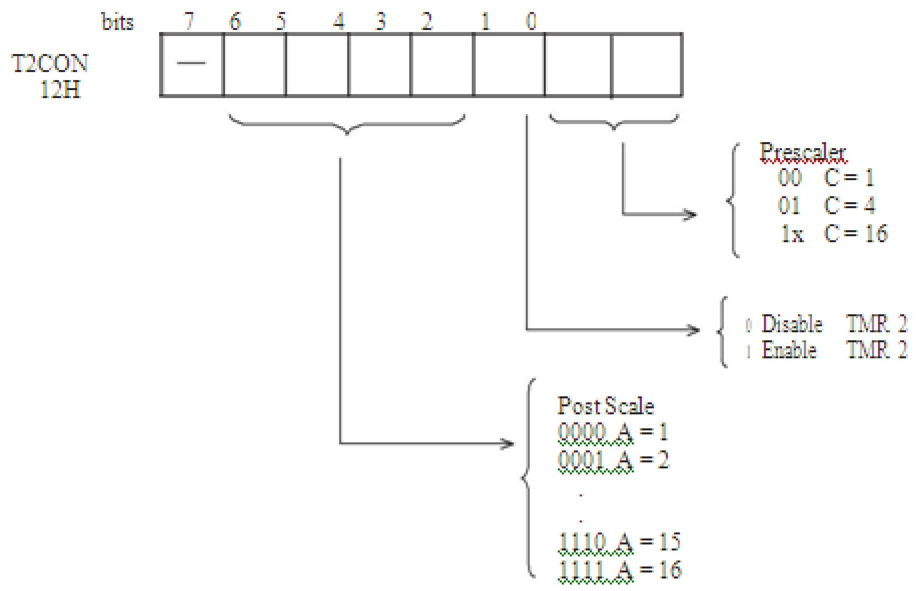
Timer 2 Circuitry



Timer 2 is an 8-bit timer with a prescalar and a port sclar. It can be used on the PWM mode of CCP modules. The TMR2 register is readable and writable and is cleared on any device reset. The input clock (f_{osc}/4) has a prescalar option of 1:1, 1:4 or 1:16 selected by bits 0 and 1 of T2CON register.

The timer 2 module has a 8-bit period register (PR2). timer 2 increments from 00H until it matches PR2 and then resets to 00H on the next increment cycle. PR2 is a readable and a writable register. PR2 is initialized to FFH on reset.

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.



4. With a simple program explain the concept of timer in detail. (Nov/Dec 2016)

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
```

5. What is the value of count for a 0.5 second delay using timer 0?

<u>Timer Delay Calculation for XTAL = 10 MHz with No Prescaler</u>

r General formula for delay calculation

m T = /(10MHz) = 0.4 usecond

T = 10 ms

Time delay = 0.5s/2 = 0.25s.

We need 0.25s/0.4us = 12500 clocks

FFFF - 30D4 +1 =CF2C

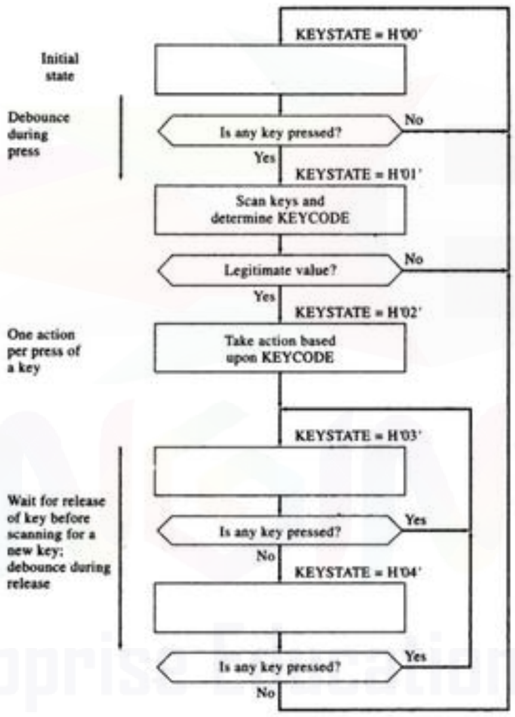
TMR0H = CFH

TMR0L= 2CH

```
BCF     TRISB,3
        MOVLW 0x08
        MOVWF T0CON
HERE
        MOVLW 0xCF
        MOVWF TMR0H
        MOVLW 0x2C
        MOVWF TMR0L
```

```
                BCF     INTCON,TMR0IF
                CALL    DELAY
                BTG     PORTB,3
                BRA     HERE
DELAY
        BSF     T0CON,TMR0ON
AGAIN
        BTFSS   INTCON,TMR0IF
        BRA     AGAIN
        BCF     T0CON,TMR0ON
        RETURN
```

6. Give a detailed note on state machine and key switches with a brief programming concept in PIC microcontroller.



KeySwitch subroutine algorithm.

7. Write a program to display a constant in PIC.

DISPLAY OF CONSTANT STRINGS

Constant strings arise in several ways. The labels associated with softkeys represent one application. The units (e.g., kHz) associated with a variable represent another. In this section, a **DisplayC** subroutine that makes use of display strings stored in program memory will be developed. Each byte of each string is accessed via a **retlw** instruction in the process of returning from a **DisplayC_Table** subroutine. As seen previously, this is the PIC way of storing tables and strings in program memory and subsequently accessing them with a variable pointer.

The source code form of a display string stored in program memory can be illustrated by the following example, used to display the string

```
Row4Col1
```

beginning in the first character position of the fourth row of the LCD display of

```
_Row4Col1
        retlw  H'D4'          ;Cursor-positioning code (Figure 7-8)
        dt     "Row4Col1"     ;Characters to be displayed
        retlw  0              ;End-of-string designator
```

The **dt** assembler directive provides a convenient way of creating ASCII strings. The MPASM assembler converts

```
        dt     "Row4Col1"
```

to

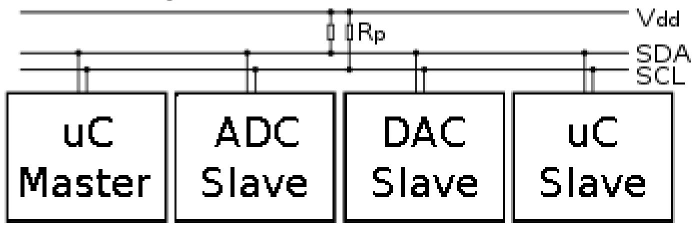
```
        retlw  A'R'
        retlw  A'o'
        retlw  A'w'
        retlw  A'4'
        retlw  A'C
        retlw  A'o'
        retlw  A'l'
        retlw  A'1'
```

where A'R' represents the ASCII code for the letter R. The label for this sequence of **retlw** instructions,

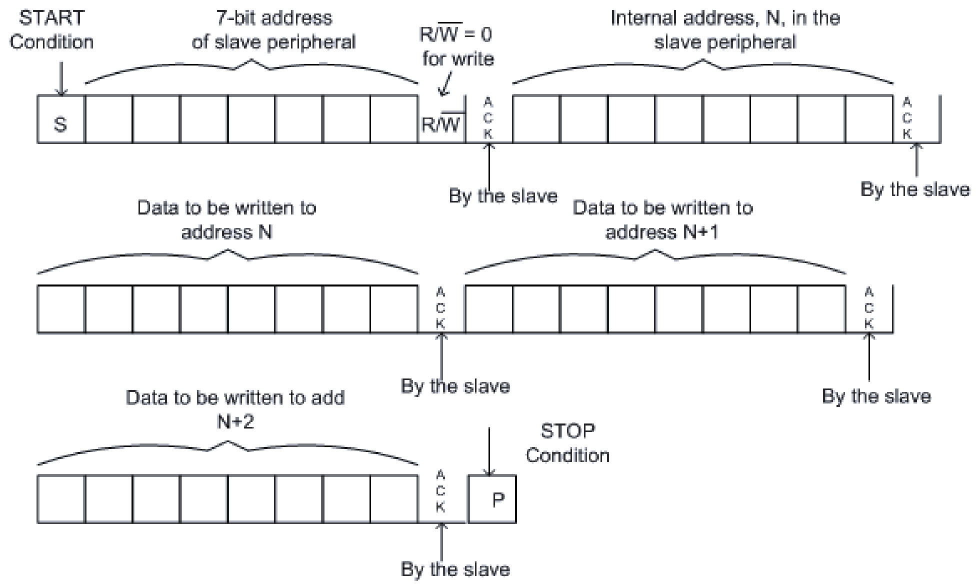
```
        _Row4Col1
```

UNIT – III PERIPHERALS AND INTERFACING
PART – A

1. **What is an I²C bus?**
   I²C is a serial protocol for two-wire interface to connect low-speed devices like microcontrollers, EEPROMs, A/D and D/A converters, I/O interfaces and other similar peripherals in embedded systems. It was invented by Philips and now it is used by almost all major IC manufacturers. I2C bus is popular because it is simple to use, there can be more than one master, only upper bus speed is defined and only two wires with pull-up resistors are needed to connect almost unlimited number of I2C devices. I2C can use even slower microcontrollers with general-purpose I/O pins since they only need to generate correct Start and Stop conditions in addition to functions for reading and writing a byte.
2. **Draw the Block diagram of I²C bus.**



3. **What are the modes of operation in I²C bus?**
   There are three data transfer speeds for the I2C bus: standard, fast-mode, and high-speed mode. Standard is 100 Kbps. Fast-mode is 400 Kbps, and high-speed mode supports speeds up to 3.4 Mbps. All are backward compatible. The I2C bus supports 7-bit and 10-bit address space devices and devices that operate under different voltages.
4. **How does PIC write data through I²C bus?**
   - A peripheral chip address and a read/write bit designating that the peripheral chip is to read successive bytes.
   - A peripheral internal register or address byte.
   - Data to write into one or more consecutive internal addresses.
5. **How PIC does reads data through I²C bus?**
   - PIC sends out a peripheral chip address and a read/write bit designating that the peripheral chip is to send one or more successive bytes beginning at a previously selected internal register or address.
   - Reads back one or more bytes of data.
6. **Draw the format of I²C bus to read and write from several peripheral interfaces.**


START Condition
7-bit address of slave peripheral
R/W = 0 for write
Internal address, N, in the slave peripheral
S
R/W
ACK
By the slave
By the slave
Data to be written to address N
Data to be written to address N+1
ACK
By the slave
By the slave
Data to be written to add N+2
STOP Condition
ACK
P
By the slave

7. **Write a note on temperature sensor used for interfacing with I²C bus.**
National Semiconductor's LM 75 chip combines an analog temperature transducer, an analog-to-digital convertor (9-bit), and an I²C bus interface, all in a tiny S)-8 surface mount package. The temperature range covered is -25°C to +100°C with ±2°C accuracy. The two's complement form of the temperature is available from the 9-bit ADC. The resolution of the ADC is about 0.5°C.

8. **What is the function of TRISA pin?**
Setting TRISA bit will configure portA as input and resetting will configure as output port.

9. **What is synchronous and asynchronous transmission**?
Asynchronous – start and stop bit allowed for transmission of data. Synchronous – no start and stop bit only block header data.

10. **What is baud rate in asynchronous mode?**
The baud rate in asynchronous mode is given by B.R = Fosc/64.(x+1) for low speed, and Fosc/16(x+1) for high speed.

11. **How data is transmitted serially using UART?**
To transmit a byte of data serially from the TX pin, the byte is written to the TXREG register. Assuming there is not already data in the TSR, the content of TXREG will be automatically transferred to TSR, making TXREG available for a second byte even as the first byte is being shifted out of the TX pin, framed by START and STOP bits.

12. **What is key debouncing?**
Key bouncing may cause multiple entries made for the same key. To overcome this problem after a key press is sensed the device is made to wait for few milliseconds. Then the key is checked again to ensure it is still pressed. If it is still pressed it is taken as a valid key press. This process is called keyboard debouncing.

13. **Name any two types of ADCs.**
The different types of ADC are successive approximation ADC, Counter type ADC flash type ADC, integrator converters and voltage to-frequency converters.

14. **What is the difference between A/D and D/A converters?**
Digital-to-analog conversion is to pull the samples from memory and convert them into an impulse train. An ADC is attempting to capture and convert a largely unknown signal into a known representation. In contrast, a DAC is taking a fully known, well-understood representation and "simply" generating an equivalent analog value. The challenge for an ADC is much greater than it is for a DAC.

15. **What is the value to be loaded into SPBRG register if we want 19200 baud rate with 10MHZ clock source. (Nov/Dec 2016)**
01101010 is the value to be loaded into SPBRG register

UNIT – III PERIPHERALS AND INTERFACING

PART – B

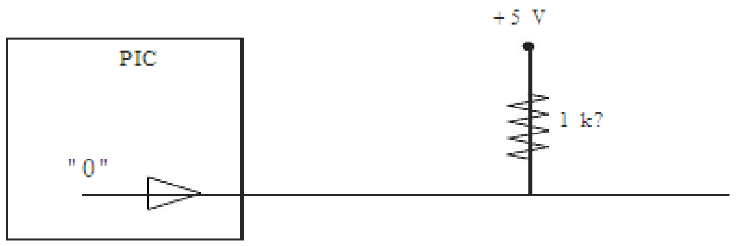
1. What is meant by I²C module? Explain how I²C is interfaced with PIC microcontroller. Nov/Dec 2016

I^2C Bus for Peripheral Chip Access

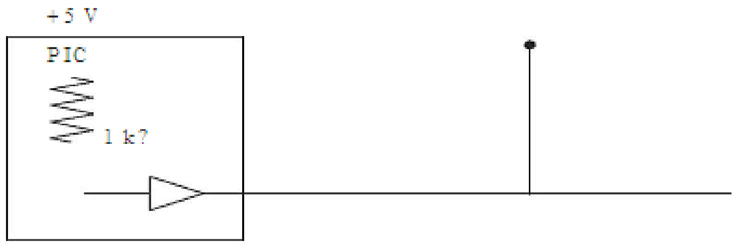
Requires two open-drain I/O pins.

Port-C of PIC IC can be used for I^2C communication.

SCL (Serial Clock)	RC3/SCK/SCL
SDA (Serial Data)	RC4/SDI/SDA

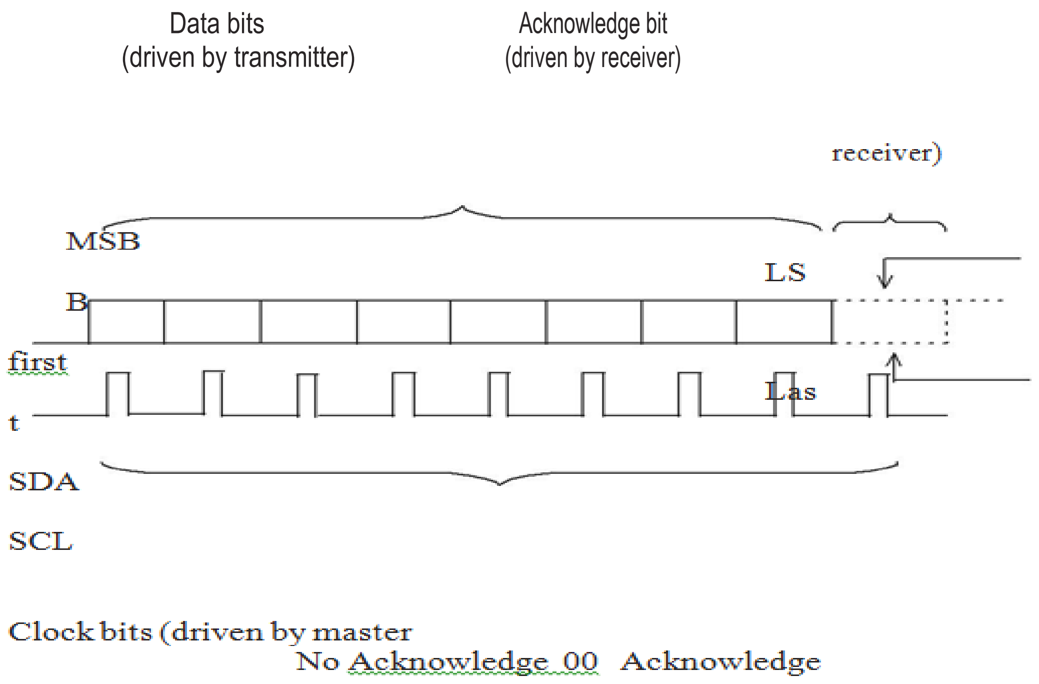


Low output on SCL or SDA I/O pin set to be an output with "0" written to it.



High output on SCL or SDA I/O pin set to be an input.

Transfers on the I^2C bus take place a bit at a time.



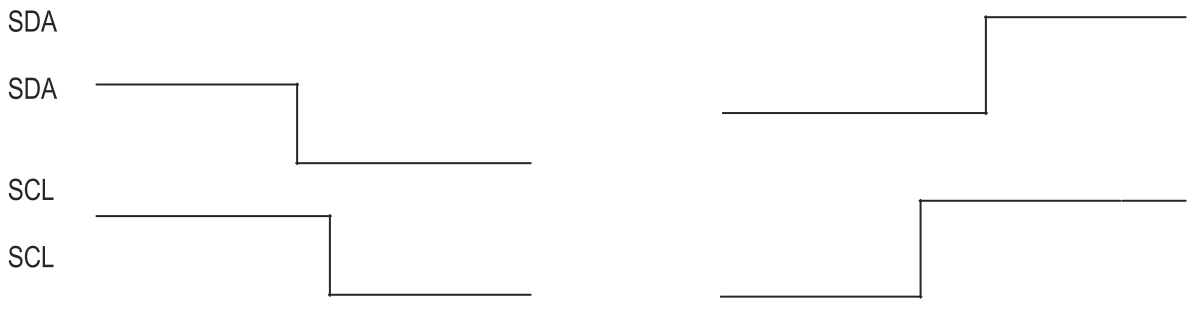
The clock line, SCL, is driven by the PIC chip, which server as *bus master*. The open drain feature of every chip's bus driver can be used by the receiver to hold the clock line low, there by signalling the transmitter to pause until the clock line is released by the receiver. The open drain feature is also needed if this PIC will ever become an

I^{2}C slave to another PIC, in which it must relinquish control of the SCL line.

The previous figure illustrates that the first eight bits on the SDA line are sent by the transmitter whereas the ninth bit is the acknowledgment bit which is sent by the receiver in response to the byte sent by the transmitter. For instance, when the PIC sends out a chip address, it is the transmitter,
while every other chip on the I^{2}C bus is a receiver. During the acknowledgment bit time, the addressed chip is the only one that drives the SDA line, pulling it low in response to the masters pulse on SCL, acknowledging the reception of its chip address.
When the data transfer direction is reversed that is form a peripheral chip to the PIC, which is the master , the peripheral chip drives the eight daa bits in response to the clock pulse from PIC. In this case, the acknowledge bit is driven in a special way by the PIC, which is serving as receive but also as bus master. If the peripheral chip is one that can send the contents of successive internal address back to the PIC, then PIC completes the reception of each byte and signals a request for the next byte by pulling *SDA line low* in acknowledgment. After any number of bytes have been received by the master from the peripheral, the PIC can signal the peripheral to stop any further transfers by not pulling the SDA line low in acknowledgment.

SDA line should be *stable during high period of the clock (SCL)*. When the slave peripheral is driving SDA line , either as transmiter or acknowledge, it initiates the new bit in response to the falling edge of SCL, after a specified time. It maintains that bit on SDA line until the next falling edge of SCL, again afte r a specified hold time.

I^{2}C bus transfers consist of a number of byte transfers framed between a START condition and either another START condition or a STOP condition. Both SDA and SCL lines are released by all drives and *float high* when bus transfers are not taking place. The PIC (I^{2}C bus controller) initiates a transfer with a START condition by first pulling SDA low and then pulling SCL as shown in the figure.



START Condition STOP Condition

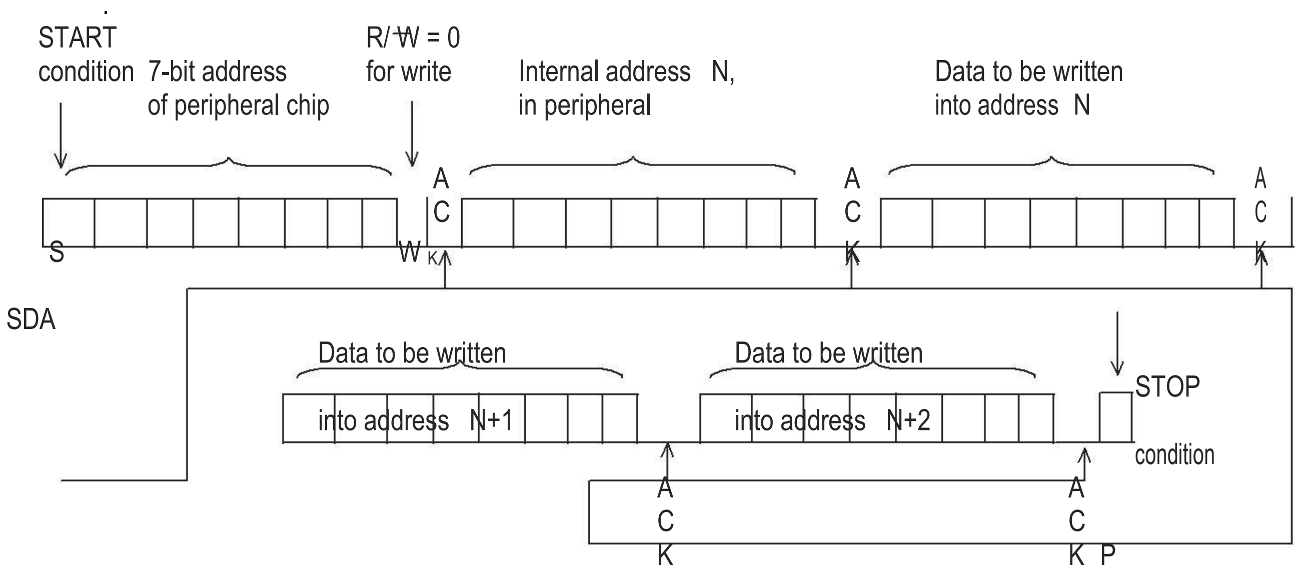
Similarly, the PIC terminates a multiple byte transfer with the STOP condition. With both SDA and SCL initially low, it first releases SCL and then SDA. Both then occurrences are easily recognized by I^{2}C hardware in each peripheral chip since they both consist of a chage in SDA line which SCL is high, a condition that never happens in the middle of a byte transfer.

Data Communication protocol

In I^{2}C communication standard, there is one bus master and several slaves. It can be assumed here that the PIC microcontroller is the bus master and several peripheral devices connected to SDA and SCL bus are slaves.

Following a start condition, the master sends a 7-bit address of the slave on SDA line. The MSB is sent first. After sending 7 bit address of the slave peripheral a R/*W* bit (8*th* bit) is sent by the master. If R/*W* bit is 0 the following byte (after the acknowledgment) is written by the master to the addressed slave peripheral. If R/*W* bit is 1, the following byte after the acknowledgment bit has to be read from the slave by the master. After sending the 7-bit address of the slave, the master sends the address of the internal register of the salve where from the data has to be used or written to. The subsegment access is automatically directed to the next address of the internal register.

The following diagrams give the general format to write and read from several peripheral internal registers


START condition
7-bit address of peripheral chip
R/ W = 0 for write
Internal address N, in peripheral
Data to be written into address N
S
W
ACK
ACK
ACK
SDA
Data to be written into address N+1
Data to be written into address N+2
STOP condition
ACK
ACK
P

2. Explain with example the concept of I²C subroutine in PIC microcontroller.

I² C Subroutine

SDA equ 4
SCL equ 3

The following subroutine DATA_OUT transfers out three bytes, i.e., ADDRDEV, ADDR8, and DATAWRTE

```
DATA_OUT:        call START           ; Generate start condition
                  movf ADDRDEV, W     ; Sends 7-bit peripheral address with R/ W=0
                  call TRBYTE          ; Transmit
                  movf ADDR8, W        ; Send 8-bit internal address
                  call TRBYTE
                  movf DATAWRTE, W    ; Send data to be written
                  call TRBYTE
                  call STOP            ; Generate Stop condition
                  return
```

The DATA_IN subroutine, which is given below transfers out ADDRDEV (with R/ W=0) and ADDR8, restarts and transfers out ADDRDEV (with R/ W =1) and read one byte back into RAM variable DATARD.

```
DATA_IN:         call START
                  movf ADDRDEV, W          ;  Send 7-bit peripheral address R/ W=0
                  call TRBYTE
                  movf ADDR8, W            ;  Send int. address
                  call TRBYTE
                  call START1              ; Restart
                  movf ADDRDEV, W          ;  Send 7-bit peripheral address R/W=1
                  call TRBYTE
                  bsf  TRBUF, 7            ; Generate NO ACK
                  call RCVBYTE
                  movwf DATARD
                  call STOP
                  return
```

The 'START' subroutine initializes I²C bus and then generates START condition on the I ²C bus. START1 bypasses the initialization of I ²C.

```
START:           movlw 3BH                ;enables I²C  master mode by programming SSPCON
                  movwf SSPCON
                  bcf PORTC, SDA           ; drive SDA low when it is an o/p
                  movlw TRISC              ;set indirect pointer to TRISC
```

```
                              movwf FSR
START1:
                              bsf INDF, SDA                ; SDA=1
                              bsf INDF , SCL               ; SCL=1
                              call DELAY                   ; Generates a suitable delay
                              bcf INDF, SDA                ; SDA=0
                              call DELAY                   ; Generate a suitable delay
                              bcf INDF, SCL                ;SCL=0
                              return
STOP:
                              bcf INDF, SDA                ;SDA=0
                              bsf INDF, SCL                ; SCL=1
                              call DELAY                   ; Generate a suitable delay
                              bsf INDF, SDA                ;SDA=1
                              return
```

The subroutine 'TRBYTE' send out the byte available in w. It returns with Z=1 if ACK occurs. It returns with Z=0 if NOACK occurs. TRBUF is an 8-bit RAM variable used for temporary storage. The bits are shifted to carry flag (C) and the carry bit transmitted successively. Data transfer is complete when all 8-bits are transmitted. Setting C = 1 initially sets an index for 8-bits to be transferred. C is rotated through TRBUF. After transmitting C, C-bit is cleared. When TRBUF is completely cleared, all 8-btis are transmitted.

```
TRBYTE:
movwf    TRBUF
bsf      STATUS,C
TR_1:


rlf      TRBUF, F
movf     RBUF,F
btfss    STATUS, Z
call     out_bit          ; Send a bit available in C
btfss    STATUS, Z
goto     TR_1
call     in_bit           ; Get the ACK bit in RCBUF<0>
movlw    01H              ;
andwf    RCBUF, W         ; Store the complement of ACK bit in Z flag
return
```

The RCVBYTE subroutine receives a byte from I^2 C into W using a RAM variable RCBUF buffer.

Call RCVBYTE with bit 7 of TRBUF clear for ACK
Call RCVBYTE with bit 7 of TRBUF set for NOACK
RCBUF is an 8-bit RAM variable used for recieving the data. the bit is recieved in the RCBUF<0> and is rotated successively through RCBUF as shown. The reception ends when all 8-bits are recieved.

```
RCVBYTE:
movlw    01H
movwf    RCBUF            ; Keep an index for 8-bits to be recieved.



rlf      RCBUF, F
call     In_bit
btfss    STATUS, C
goto     RCV_1
rlf      TRBUF, F
call     Out_bit
movf     RCBUF,w
```

```
return
```

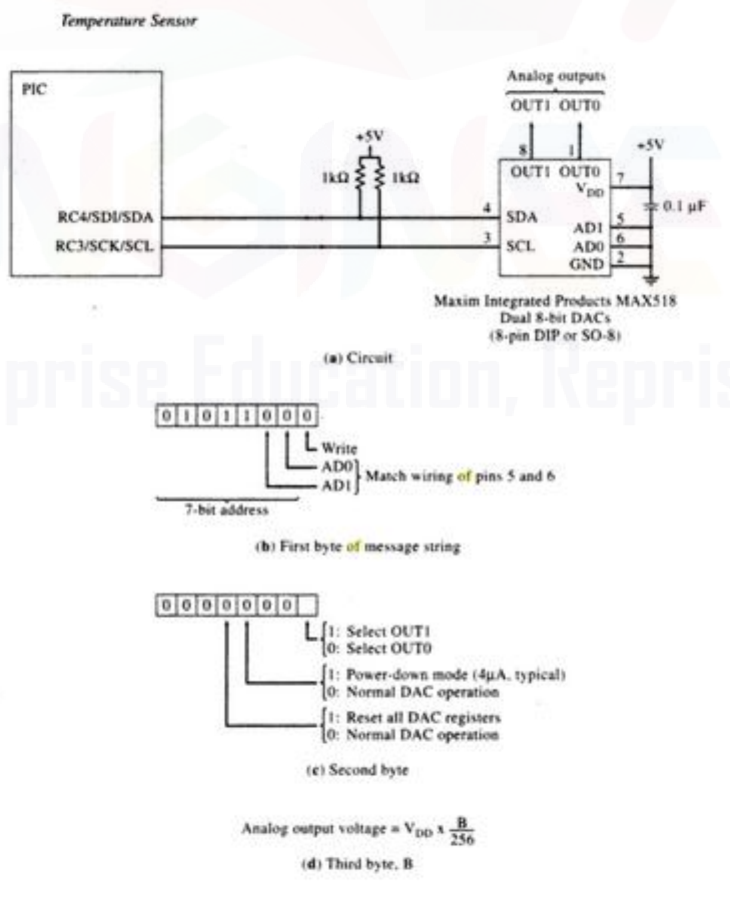
The out_bit subroutine transmits carry bit, then clears the carry bit.

```
Out_bit:
bcf      INDF, SDA
btfsc  STATUS, C
bsf     INDF, SDA            ; Send carry bit
bsf     INDF, SCL
call     DELAY
bcf      INDF, SCL
bcf      STATUS,C            ; Clear carry bit
return
```

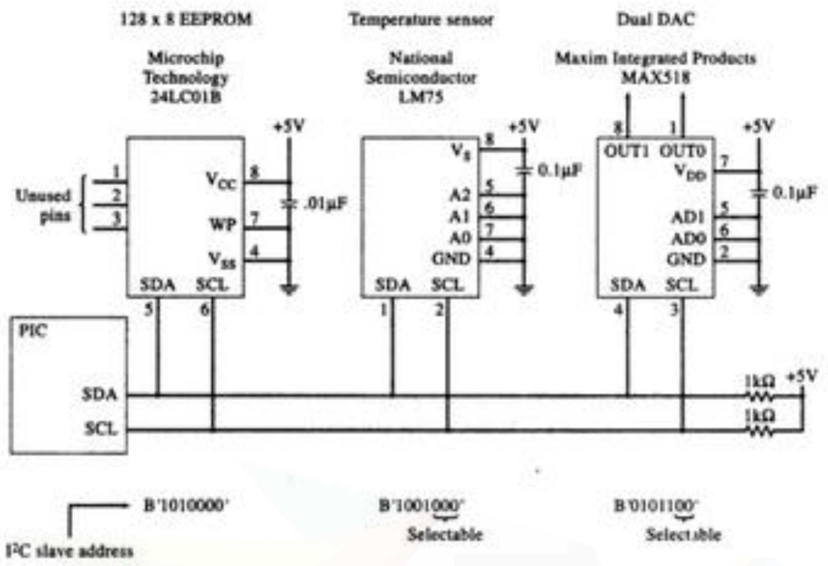
The in_bit subroutine receives one bit into bit-0 of RCBUF.

```
In_bit:
bsf      INDF,SDA
bsf      INDF, SCL
bcf      RCBUF, 0
btfsc   PORTC, SDA          ; Check SDA line for data bit
bsf      RCBUF, 0
bcf      INDF, SCL
return
```

3. Explain in detail the interfacing of temperature sensor using I²C bus.



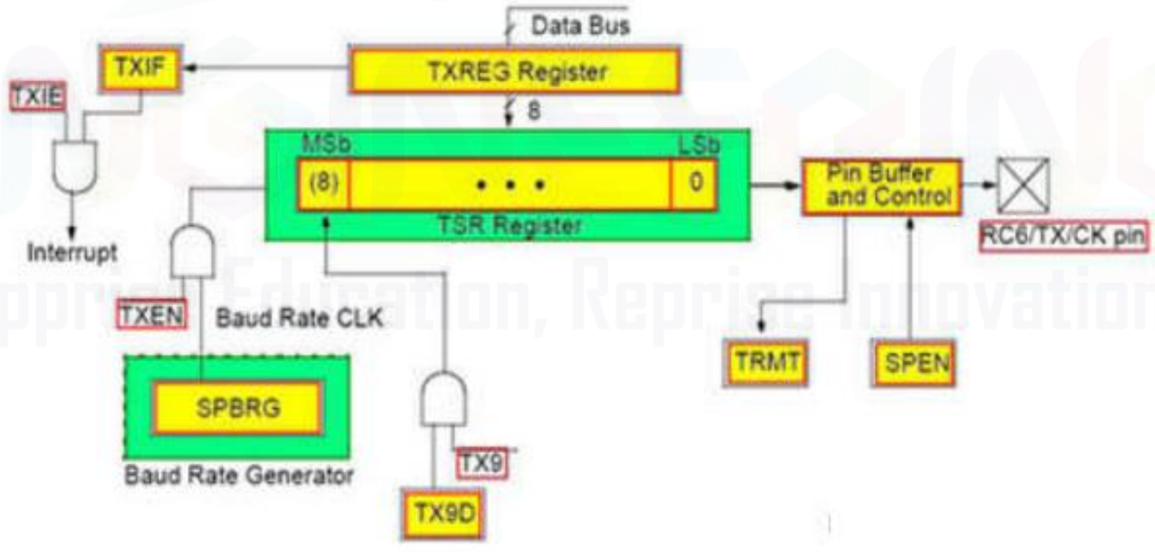
4. **Explain with neat diagram interfacing of serial EEPROM using I²C bus.**



I²C bus connection to EEPROM chip and to other devices.

5. **Explain with neat diagram the use of UART to interface two PIC resources.**

USART transmit block diagram



USART transmit block diagram

DEPARTMENT OF ELECTRICAL AND ELECTRONICS ENGINEERING

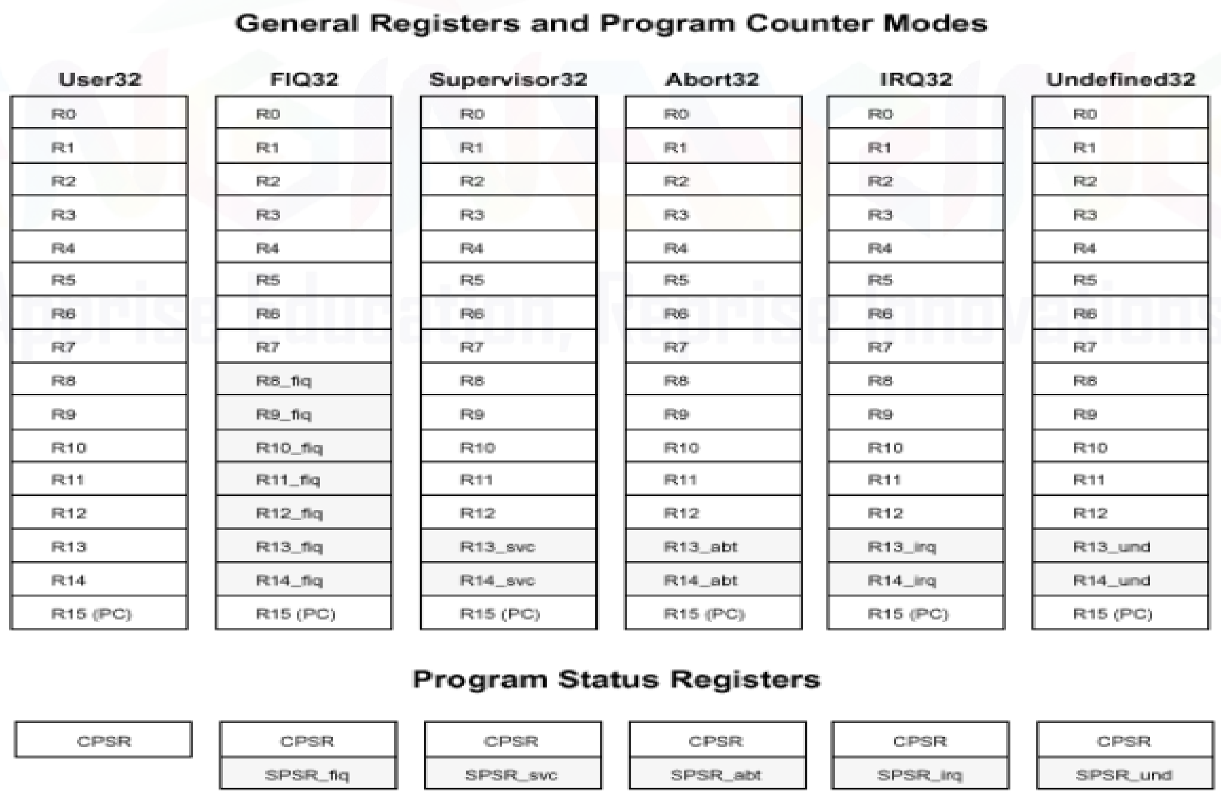
EE8018 – Microcontroller based system design

Year: IV **Semester : VII**

UNIT – IV INTRODUCTION TO ARM PROCESSOR

PART-A

1. **What are the features of RISC architecture that were used in ARM architecture?**
   - A load store architecture
   - Fixed length 32 bit instructions
   - 3- Address instruction formats.
2. **What is CPSR?**
   The CPSR (Current Program Status Register) is used in user level programs to store the condition code bits. These bits are used for example, to record the result of a comparison operation and to control whether or not a condition branch is taken.
3. **What are the registers available in ARM processor?**
   - R0 to R15 directly accessible
   - R0-R14 general purpose
   - R13 stack pointer
   - R14 Linked register
   - R15 holds Program Counter
   - CPSR- Current Program Status Register contains condition code flags and current mode bits.
   - 5 SPSRs (Saved Program Status Registers) which are loaded with CPSR when exception occurs.
4. **Draw the ARM programmer's model.**



General Registers and Program Counter Modes

User32	FIQ32	Supervisor32	Abort32	IRQ32	Undefined32
R0	R0	R0	R0	R0	R0
R1	R1	R1	R1	R1	R1
R2	R2	R2	R2	R2	R2
R3	R3	R3	R3	R3	R3
R4	R4	R4	R4	R4	R4
R5	R5	R5	R5	R5	R5
R6	R6	R6	R6	R6	R6
R7	R7	R7	R7	R7	R7
R8	R8_fiq	R8	R8	R8	R8
R9	R9_fiq	R9	R9	R9	R9
R10	R10_fiq	R10	R10	R10	R10
R11	R11_fiq	R11	R11	R11	R11
R12	R12_fiq	R12	R12	R12	R12
R13	R13_fiq	R13_svc	R13_abt	R13_irq	R13_und
R14	R14_fiq	R14_svc	R14_abt	R14_irq	R14_und
R15 (PC)	R15 (PC)	R15 (PC)	R15 (PC)	R15 (PC)	R15 (PC)

Program Status Registers

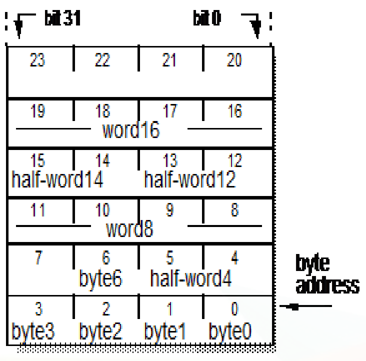
CPSR	CPSR	CPSR	CPSR	CPSR	CPSR
	SPSR_fiq	SPSR_svc	SPSR_abt	SPSR_irq	SPSR_und

4.List out some features of ARM architecture.
- A large set of registers, all of which can be used for most purposes.
- 3- address instructions(that is, the two source operand registers and the result register are all independently specified)

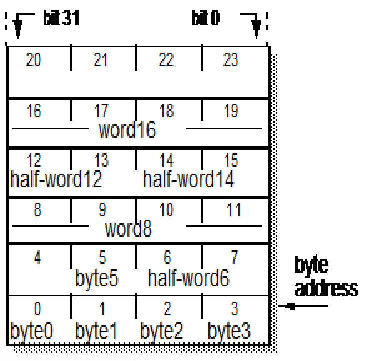
- Conditional execution of every instruction.
- Inclusions of very powerful load and store multiple register instructions.

5. **List out some ARM development tools.**
   ARM C compiler, ARM assembler, Linker, ARMsd, ARMulator, ARM development board.
6. **What is ARMulator?**
   The ARMulator is a suite of programs that models the behavior of various ARM processor cores in software on a host system.

7. **Draw Memory organization in ARM processor.**



(a) Little-endian memory organization



(b) Big-endian memory organization

8. **What are the seven modes of operations in ARM processor?**
   User mode(usr),Fast interrupt mode(fiq), Interrupt mode(irq), supervisor mode(svc),abort(abt),,system(sys), undefined mode(und).
9. **List out the types of instructions used in ARM processor.**
   i. Data processing instructions
   ii. Data Transfer instructions.
   iii. Control flow instructions.
10. **What is register indirect addressing? Give example.**
    Register indirect addressing uses a value in one register (the base register)as a memory address and either loads the value from that address into another register or stores the value from another register into that memory address.
    e.g. LDR r0, [r1] ; r0 : = mem$_{32}$[r1]
    STR r0, [r1] ; mem$_{32}$[r1] : =r0
11. **What is pre indexed and post indexed addressing modes?**
    The preindexed addressing mode is one which allows one base register to be used to access a number of memory locations which are in the same area of memory.
    The post indexed addressing, allows the base to be used without an offset as the transfer address, after which it is auto indexed.
12. **What is cache memory?**
    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.
13. **Define hit and miss rate.**
    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.
14. **What are the two memory management approaches?**
    The two memory management approaches are segmentation and paging.
15. **What is paging?**
    In 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.
16. **What is the purpose of Program Counter? Nov/Dec 2016.**
    1. Store the address of the instruction to be executed.

2. All instructions are 32 bit wide
3. Thus, the last 2 bits of PC are undefined.

17. **List out some ARM Development tools. Nov/Dec 2016.**
Keil-C, IAR bench, mbed, ARM DS-5 development studio are some of the ARM development tools.

UNIT – IV INTRODUCTION TO ARM PROCESSOR

Part-B

1. With Neat sketch, explain the functional block diagram of ARM architecture. Nov//Dec 2016.

ARM architecture forms the basis for every ARM processor. Over time, the ARM architecture has evolved to include architectural features to meet the growing demand for new functionality, integrated security features, high performance and the needs of new and emerging markets. There are currently 3 ARMv8 profiles, the ARMv8-A architecture profile for high performance markets such as mobile and enterprise, the ARMv8-R architecture profile for embedded applications in automotive and industrial control, and the ARMv8-M architecture profile for embedded and IoT applications.

The ARM architecture supports implementations across a wide range of performance points, establishing it as the leading architecture in many market segments. The ARM architecture supports a very broad range of performance points leading to very small implementations of ARM processors, and very efficient implementations of advanced designs using state of the art micro-architecture techniques. Implementation size, performance, and low power consumption are key attributes of the ARM architecture.

The ARM architecture is similar to a Reduced Instruction Set Computer (RISC) architecture, as it incorporates these typical RISC architecture features:

A uniform register file load/store architecture, where data processing operates only on register contents, not directly on memory contents.

Simple addressing modes, with all load/store addresses determined from register contents and instruction fields only.

Enhancements to a basic RISC architecture enable ARM processors to achieve a good balance of high performance, small code size, low power consumption and small silicon area.

A64 is a new 32-bit fixed length instruction set to support the AArch64 execution state. The following is a summary of the A64 ISA features.

Clean decode table based on 5-bit register specifies
Instruction semantics broadly the same as in AArch32
31 general purpose 64-bit registers accessible at all times
No modal banking of GP registers - Improved performance and energy
Program counter (PC) and Stack pointer (SP) not general purpose registers
Dedicated zero register available for most instructions
Key differences from A32 are:
New instructions to support 64-bit operands. Most instructions can have 32-bit or 64-bit arguments
Addresses assumed to be 64-bits in size. LP64 and LLP64 are the primary data models targeted
Far fewer conditional instructions than in AArch32 conditional {branches, compares, selects}
No arbitrary length load/store multiple instructions LD/ST 'P' for handling pairs of registers added A64.

2. Explain the various operating modes programmers model in ARM processor. Nov//Dec 2016.

Introduction

ARM has a 32-bit data bus and a 32-bit address bus. The data types the processor supports are Words (32 bits), where words must be aligned to four byte boundaries. Instructions are exactly one word, and data operations
(e.g. ADD) are only performed on word quantities. Load and store operations
can transfer words.

ARM supports six modes of operation:

(1) User mode (usr): the normal program execution state

(2) FIQ mode (fiq): designed to support a data transfer or channel process
(3) IRQ mode (irq): used for general-purpose interrupt handling
(4) Supervisor mode (svc): a protected mode for the operating system
(5) Abort mode (abt): entered after a data or instruction prefetch abort
(6) Undefined mode (und): entered when an undefined instruction is executed

Mode changes may be made under software control or may be brought about by external interrupts or exception processing. Most application programs will execute in User mode. The other modes, known as privileged modes, will be entered to service interrupts or exceptions or to access protected resources.

Registers

The processor has a total of 37 registers made up of 31 general 32 bit registers and 6 status registers. At any one time 16 general registers (R0 to R15) and one or two status registers are visible to the programmer. The visible registers depend on the processor mode and the other registers (the banked registers) are switched in to support IRQ, FIQ, Supervisor, Abort and Undefined mode processing. The register bank organization is shown in Figure (3-1). The banked registers are shaded in the diagram. In all modes 16 registers, R0 to R15, are directly accessible. All registers except R15 are general purpose and may be used to hold data or address values. Register R15 holds the Program Counter (PC). When R15 is read, bits [1:0] are zero and bits [31:2] contain the PC. A seventeenth register (the CPSR - Current Program Status Register) is also accessible.
It contains condition code flags and the current mode bits and may be thought of as an extension to the PC. R14 is used as the subroutine link register and receives a copy of R15 when a Branch and Link instruction is executed. It may be treated as a general- purpose register at all other times. R14_svc, R14_irq, R14_fiq, R14_abt and R14_und are used similarly to hold the return values of R15 when interrupts and exceptions arise, or when Branch and Link instructions are executed within interrupt or exception routines.

Exceptions arise whenever there is a need for the normal flow of program execution to be broken, so that (for example) the processor can be diverted to handle an interrupt from a peripheral. The processor state just prior to handling the exception must be preserved so that the original program can be resumed when the exception routine has completed. Many exceptions may arise at the same time.
ARM handles exceptions by making use of the banked registers to save state. The old PC and CPSR contents are copied into the appropriate R14 and SPSR and the PC and mode bits in the CPSR bits are forced to a value that depends on the exception. Interrupt disable flags are set where required to prevent otherwise unmanageable nestings of exceptions. In the case of a re-entrant interrupt handler, R14 and the SPSR should be saved onto a stack in main memory before re-enabling the interrupt; when transferring the SPSR register to and from a stack, it is important to transfer the whole 32-bit value, and not just the flag or control fields. When multiple exceptions arise simultaneously, a fixed priority determines the order in which they are handled. The priorities are listed later in this chapter.

The FIQ (Fast Interrupt reQuest) exception is externally generated by taking the nFIQ input LOW. This input can accept asynchronous transitions, and is delayed by one clock cycle for synchronization before it can affect the processor execution flow. It is designed to support a data transfer or channel process, and has sufficient private registers to remove the need for register saving in such applications (thus minimizing the overhead of context switching). The FIQ exception may be disabled by setting the F flag in the CPSR (but note that this is not possible from User mode). If the F flag is clear, Our ARM checks for a LOW level on the output of the FIQ synchronizer at the end of each instruction.When a FIQ is detected, ARM performs the following:

(1) Saves the address of the next instruction to be executed plus 4 in R14_fiq; saves CPSR in SPSR_fiq
(2) Forces M[4:0]=10001 (FIQ mode) and sets the F and I bits in the CPSR
(3) Forces the PC to fetch the next instruction from address 0x1C

To return normally from FIQ, use SUBS PC, R14_fiq,#4 which will restore both the PC (from R14) and the CPSR (from SPSR_fiq) and resume execution of the interrupted code.

The IRQ (Interrupt ReQuest) exception is a normal interrupt caused by a LOW level on the nIRQ input. It as a lower priority than FIQ, and is masked out when a FIQ sequence is entered. Its effect may be masked out at any time by setting the I bit in the CPSR (but note that this is not possible from User mode). If the I lag is clear, Our ARM checks for a LOW level on the output of the IRQ synchronizer at the end of each instruction. When an IRQ is detected, Our ARM performs the following:

(1) Saves the address of the next instruction to be executed plus 4 in R14_irq; saves CPSR in SPSR_irq
(2) Forces M[4:0]=10010 (IRQ mode) and sets the I bit in the CPSR
(3) Forces the PC to fetch the next instruction from address 0x18

To return normally from IRQ, use SUBS PC,R14_irq,#4 which will restore both the PC and the CPSR and resume execution of the interrupted code.

3. Explain in detail ARM development tools.

Software Tools
The ARM tools range offers two software development families that provide you with all the necessary tools for every stage of your software development workflow:
ARM DS-5 Development Studio provides best-in-class tools for the broadest range of ARM processor-based platforms, including application processors and multicore SoCs
Keil MDK-ARM™ is a complete software development toolkit for ARM processor-based microcontrollers. It is the right choice for embedded applications based on the ARM Cortex™-M series, ARM7™, ARM9™, and Cortex-R4 processors

DS-5
DS-5 is a suite of professional software development tools for all ARM processors.

Keil MDK-ARM
Keil MDK-ARM™ supports more than 1200 devices based on the ARM Cortex™-M series, ARM7™, ARM9™ and Cortex-R4 processors. It includes numerous examples, project templates and middleware libraries, with extensive TCP/IP software stack, flash file system, USB Hose and Device stack, CAN access, and a comfortable solution for graphical user interfaces.

The easy-to-use IDE and the fully featured debugger with advances analysis functionality help developers to get started quickly with their project and concentrate on the differentiating features of their application.

ARM Software Development Tools
Documentation set for ARM software development tools, including the following product families:
ARM DS-5 Development Studio is a suite of tools for embedded development on all ARM processor-based devices. It includes:
DS-5 Debugger: An operating system aware debugger for ARM software, for debugging bare-metal systems, the Linux kernel, and Linux and native Android applications. Also supports CoreSight trace.
Streamline Performance Analyzer: ARM's profiling tool, presenting a system-wide dashboard of hardware and software performance counters for ARM processors and Mali GPUs.
ARM C/C++ compilation tools: ARM Compiler 5 and ARM Compiler 6 generate size and performance optimized code for ARM processors.
Fixed Virtual Platforms: Models of ARM systems to test software execution without a hardware target.
Eclipse IDE: Extensible platform for code authoring and project management.
Keil MDK-ARM supports embedded microcontroller software development on Cortex-M, Cortex-R and classic ARM devices. It includes:
µVision IDE: Includes project management, code editing and RTOS-aware debug tools.
ARM C/C++ compilation tools: ARM Compiler 5 generates small-footprint code for microcontrollers.
RTX: Deterministic real-time operating system with source code.
Middleware: Drivers and software stacks for embedded development.
ARM mbed provides tools that simplify software development for the Internet of Things (IoT), such as an online compiler, SDK, libraries, and code sharing tools.
Mali GPU Development Tools for software development on Mali GPUs, including Mali Graphics Debugger.

ARM also provides documentation for a range of debug probes and adapters:
DSTREAM: High performance JTAG debug and parallel trace unit, capable of supporting custom ARM SoCs.
HSSTP: High speed serial trace probe, replacing the parallel trace probe on the DSTREAM.
ULINKpro Family: JTAG Debug unit compatible with Keil MDK-ARM and DS-5, supporting a range of off-the-shelf ARM-based devices.
ULINK2: Entry level debug probe for use with Keil MDK-ARM and a limited set of devices in DS-5.
ULINK-ME: Basic debug adapter for use with Keil MDK-ARM.

4. Explain with examples in detail the Data processing instructions of ARM processor.

Arithmetic Instructions
arithmetic instructions are very basic and frequently used in your ARM programming.
Here is a table that demonstrates the usage of the ARM processor's arithmetic instructions with examples.

Instruction	Mnemonic	Meaning
Addition	ADD R0, R1, R2	; R0 = R1 + R2
Addition	ADDS R1, R2, R3	; R1 = R2 + R3, ; and FLAGs are updated
Subtraction	SUB R0, R1, R2	; R0 = R1 - R2
Subtraction	SUBS R1, R2, R3	; R1 = R2 - R3, ; and FLAGs are updated
	SUBS R7, R6, #20	; R7 = R6 - 20 ; Sets the flags on the result
Reverse Subtraction	RSB R4, R4, #120	; R4 = 120 - R4
Multiply	MUL R0, R1, R2	; R0 = R1 * R2
	UMULL R0, R4, R5, R6	; Unsigned (R4,R0) = R5 * R6
	SMLAL R4, R5, R3, R8	; Signed (R5,R4) = (R5,R4) + R3 * R8
Division	SDIV R0, R2, R4	; Signed divide, R0 = R2/R4
	UDIV R8, R8, R1	; Unsigned divide, R8 = R8/R1.

Examples of Move Instructions

Mnemonic	Meaning
MOVS R11, #0x000B	; Write value of 0x000B to R11, flags get updated
MOV R1, #0xFA05	; Write value of 0xFA05 to R1, flags are not updated
MOVS R10, R12	; Write value in R12 to R10, flags get updated
MOV R3, #23	; Write value of 23 to R3
MOV R8, SP	; Write value of stack pointer to R8
MVNS R2, #0xF	; Write value of 0xFFFFFFF0 (bitwise inverse of 0xF) ; to the R2 and update flags.

Logical Operation Instructions
AND R9, R2, R1 ; R9 = R2 AND R1

```
------------------------------------------------------------------------
        AND R9, R2, #0xFF00              ; R9 = R2 AND #0xFF00
------------------------------------------------------------------------
        ORR R9, R2, R1                   ; R9 = R2 OR R1

        ORR R9, R2, #0xFF00
------------------------------------------------------------------------
        ORREQ R2, R0, R5
------------------------------------------------------------------------
        ANDS R9, R8, #0x19
------------------------------------------------------------------------
        EOR  R7, R11, R10                ; R7 = R11 XOR R10
------------------------------------------------------------------------
        EORS R7, R11, #0x18181818
------------------------------------------------------------------------
        BIC R0, R1, #0xab                ; R0 = R1 AND (NOT(#0xab))
------------------------------------------------------------------------
        ORN R7, R11, R14, ROR #4         ; R7 = R11 OR (NOT(R14 ROR #4))
------------------------------------------------------------------------
        ORNS R7, R11, R14, ROR #2        ; update the flags
```

5. Explain with examples in detail the Data transfer instructions of ARM processor.

Load and store multiple register instructions

The ARM, Thumb-2, and pre-Thumb-2 Thumb instruction sets include instructions that load and store multiple registers to and from memory.

Multiple register transfer instructions provide an efficient way of moving the contents of several registers to and from memory. They are most often used for block copy and for stack operations at subroutine entry and exit. The advantages of using a multiple register transfer instruction instead of a series of single data transfer instructions include:

Smaller code size.

A single instruction fetch overhead, rather than many instruction fetches.

On uncached ARM processors, the first word of data transferred by a load or store multiple is always a nonsequential memory cycle, but all subsequent words transferred can be sequential memory cycles. Sequential memory cycles are faster in most systems.

Block data transfer instructions load/store a set of general purpose register values from/to main memory; the instruction format is shown in Figure 2. These instructions are used for procedure entry and return (saving and restoring workspace registers), and in writing memory block copy routines.

The set of registers to be transferred is encoded using a 16-bit value; the program counter may be included in the list (bit fifteen). The memory block is determined by the base register Rn, and the bits P and U. The W flag enables base register write back (auto-indexing). There are also special forms of the instruction for accessing the user mode registers (when not in user mode) and for restoring the CPSR when returning from an exception – these options are controlled by the S flag and bit fifteen.

The instruction syntax is illustrated below:

```
LDM|STM{<cond>}<add mode> Rn{!}, <registers>
LDM{<cond>}<add mode> Rn{!}, <registers + pc>^
LDM|STM{<cond>}<add mode> Rn, <registers - pc>^
```

Here <cond> is a condition code, <addr mode> is the address mode and <registers> is a list of registers. The block copying address modes are IA, IB, DA and DB – as indicated these increment/decrement the address register after/before each memory access.2 An ! is used for base register write-back, and the suffix ^ is used to set the S flag.

As an example, if the processor is in supervisor mode with the Z flag set, the instruction

LDMEQDB r0!, {r1,r2,pc}^

will perform the following assignments:

r0 ⟵ r0-12; r1 ⟵ mem[r0-12]; r2 ⟵ mem[r0-8]; r15 ⟵ mem[r0-4]; CPSR ⟵ SPSR_svc .

All transfers are ordered: registers with lower indices are mapped to lower memory addresses. The register list should not be empty i.e. the lowest sixteen bits of the op-code should not all be clear. This restriction will be enforced by any sensible compiler and/or assembler, but this does not guarantee that such instructions can never be executed (it is trivial to write an assembly program that generates and then executes such an instruction). The ARM6 has an unfortunate load multiple behaviour when the register list is empty – a load to the program counter occurs. Rather than specify this at the programmer's model level, the hol model of the ARM6 has been modified to give a more sensible behaviour i.e. no load occurs.
With block stores, if the program counter is in the list then the value stored is implementation dependent. If the base register is in the list then write-back should not be specified because the result is unpredictable. The hol programmer's model specification has been tailored to conform with ARM6 behaviour for these cases.

6. Explain with examples in detail the Control flow instructions of ARM processor
Branch and Control Instructions:

Branch instructions are very useful for selection control and looping control.

Here is a list of the ARM processor's Branch and Control instructions.

```
B loopA  ; Branch to label loopA unconditioally
---------------------------------------------------------------------------
        B.W target        ; Branch to target within 16MB range
---------------------------------------------------------------------------
        BEQ target        ; Conditionally branch to target, when Z = 1
---------------------------------------------------------------------------
        BEQ.W target      ; Conditionally branch to target within 1MB when Z = 1
---------------------------------------------------------------------------
        BNE AAA                   ; branch to AAA when Z = 0
---------------------------------------------------------------------------
        BMI BBB                   ; branch to BBB when N = 1
---------------------------------------------------------------------------
        BPL  CCC          ; branch to CCC when N = 0
---------------------------------------------------------------------------
        BLT  labelAA      ; Conditionally branch to label labelAA,
                          ; N set and V clear  or  N clear and V set
                          ; i.e.  N != V
---------------------------------------------------------------------------
        BLE labelA        ; Conditionally branch to label labelA,
                          ; when less than or equal, Z set or N set and V clear
                          ; or N clear and V set
                          ; i.e.  Z = 1 or N != V
---------------------------------------------------------------------------
        BGT  labelAA      ; Conditionally branch to label labelAA,
                          ; Z clear and either N set and V set
                          ;               or  N clear and V clear
                          ; i.e.   Z = 0 and N = V
---------------------------------------------------------------------------
        BGE labelA        ; Conditionally branch to label labelA,
                          ; when Greater than or equal to zero,
                          ; Z set or N set and V clear
                          ; or N clear and V set
                          ; i.e.  Z = 1 or N !=V
---------------------------------------------------------------------------
        BL funC ; Branch with link (Call) to function funC,
                          ; return address stored in LR, the register R14
---------------------------------------------------------------------------
```

```
        BX LR           ; Return from function call
-----------------------------------------------------------------------
        BXNE R0         ; Conditionally branch to address stored in R0
-----------------------------------------------------------------------
        BLX R0          ; Branch with link and exchange (Call)
                        ; to a address stored in R0.
```

7. Explain with examples different types of addressing used in ARM processor.

There are different ways to specify the address of the operands for any given operations such as load, add or branch. The different ways of determining the address of the operands are called addressing modes. In this lab, we are going to explore different addressing modes of ARM processor and learn how all instructions can fit into a single word (32 bits).

Name	Alternative Name	ARM Examples
Register to register	Register direct	MOV R0, R1
Absolute	Direct	LDR R0, MEM
Literal	Immediate	MOV R0, #15 ADD R1, R2, #12
Indexed, base	Register indirect	LDR R0, [R1]
Pre-indexed, base with displacement	Register indirect with offset	LDR R0, [R1, #4]
Pre-indexed, autoindexing	Register indirect pre-incrementing	LDR R0, [R1, #4]!
Post-indexing, autoindexed	Register indirect post-increment	LDR R0, [R1], #4
Double Reg indirect	Register indirect Register indexed	LDR R0, [R1, R2]
Double Reg indirect with scaling	Register indirect indexed with scaling	LDR R0, [R1, r2, LSL #2]
Program counter relative		LDR R0, [PC, #offset]

8. i. Write and ARM ALP to display a text “Hello World” .
ii. Write and ARM ALP which dumps a register to the display in hexadecimal notation.

```
.global _start
_start:
  MOV R7, #4
  MOV R0, #1
  MOV R2, #12
  LDR R1, =string
  SWI 0
  MOV R7, #1
  SWI 0
  .data
string:
  .ascii "Hello Worldn"
```

ii) load a register :

```
LDR rn [pc, #offset_to_literal_pool]
                        ; load register n with one word
                        ; from the address [pc + offset]
```

DEPARTMENT OF ELECTRICAL AND ELECTRONICS ENGINEERING

EE8018 – Microcontroller based system design

UNIT – V ARM ORGANIZATION
PART-A

1. What is instruction pipelining?

Instruction pipelining is a technique that implements a form of parallelism called instruction-level parallelism within a single processor. It therefore 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. Rather than processing each instruction sequentially (finishing one instruction before starting the next), each instruction is split up into a sequence of steps so different steps can be executed in parallel and instructions can be processed concurrently (starting one instruction before finishing the previous one).

2. What are the three processes used by pipelining technique?

Pipelined processors commonly use three techniques to work as expected when the programmer assumes that each instruction completes before the next one begins:

Processors that can compute the presence of a hazard may stall, delaying processing of the second instruction (and subsequent instructions) until the values it requires as input are ready. This creates a bubble in the pipeline (see below), also partly negating the advantages of pipelining.

Some processors can not only compute the presence of a hazard but can compensate by having additional data paths that provide needed inputs to a computation step before a subsequent instruction would otherwise compute them, an attribute called operand forwarding.

Some processors can determine that instructions other than the next sequential one are not dependent on the current ones and can be executed without hazards. Such processors may perform out-of-order execution.

3.What are the 3-stage ARM pipelines?

Fetch:

• The instruction is fetched from memory decode

• The instruction is decoded and the datapath control signals prepared for the next cycle\

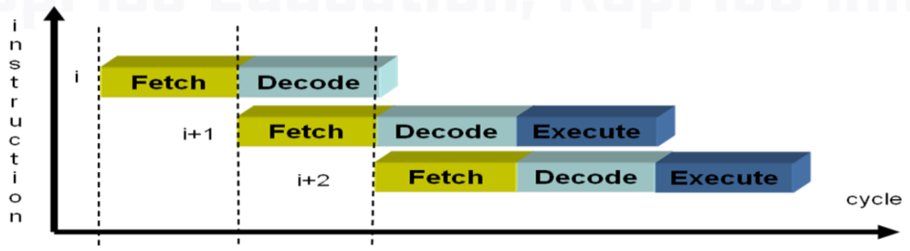
Decode:

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.

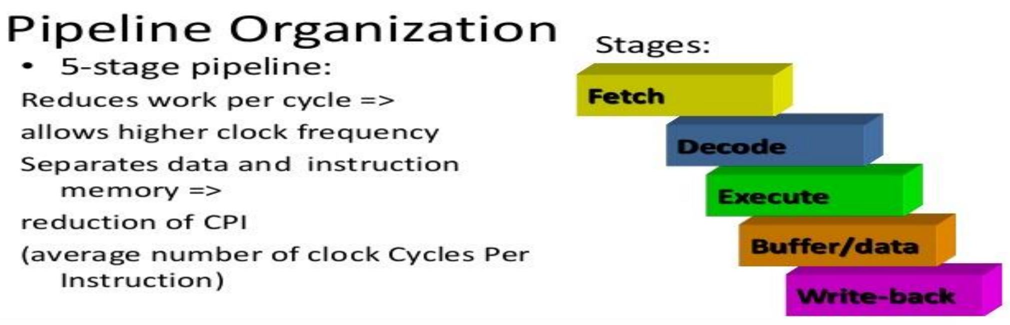
Execute:

• The operands are read from the register bank, shifted, combined in the ALU and the result written back

4.Draw the diagrammatic view of 3-stage pipeline.



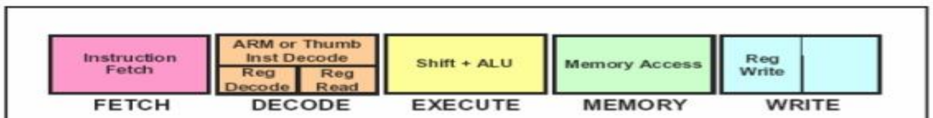
5.Draw the diagram of 5-stage pipeline organization.


Pipeline Organization

• 5-stage pipeline:

Reduces work per cycle =>
allows higher clock frequency
Separates data and instruction
memory =>
reduction of CPI
(average number of clock Cycles Per
Instruction)

Stages:
Fetch
Decode
Execute
Buffer/data
Write-back



6. What is five stage pipeline in ARM processor? (Nov/Dec 2016)

A typical 5-stage ARM pipeline is that employed in the ARM9TDMI.

The ARM processors which use a 5-stage pipeline have the following pipelinestages:

• 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 buf-fered 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.

7. 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.

A base pointer ,used to point to the “bottom” of the stack (the first location).

A stack pointer, used to point the current “top” of the stack.

8. 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. In ARM7TDMI cores, the "D" represented JTAG debug support, and the "I" represented presence of an "Embedded ICE" debug module. For ARM7 and ARM9 core generations, Embedded ICE over JTAG was a de facto debug standard, though not architecturally guaranteed.

9. Give an assembly language module in ARM.

```
AREA    ARMex, CODE, READONLY
                    ; Name this block of code ARMex
    ENTRY           ; Mark first instruction to execute
start
    MOV     r0, #10        ; Set up parameters
    MOV     r1, #3
    ADD     r0, r0, r1     ; r0 = r0 + r1
stop
    MOV     r0, #0x18      ; angel_SWIreason_ReportException
    LDR     r1, =0x20026   ; ADP_Stopped_ApplicationExit
    SVC     #0x123456      ; ARM semihosting (formerly SWI)
    END                    ; Mark end of file
```

10. What is the need of Thumb instruction set in ARM processor?

The Thumb instruction set is a subset of the most commonly used 32-bit ARM instructions. Thumb instructions are each 16 bits long, and have a corresponding 32-bit ARM instruction that has the same effect on the processor model. Thumb instructions operate with the standard ARM register configuration, allowing excellent interoperability between ARM and Thumb states

UNIT – V ARM ORGANIZATION

PART-B

1. **Explain in detail, the 3-state ARM pipeline organization. Show the difference between a single cycle and a multi-cycle instruction**

3-stage pipeline ARM organization

Fig. 3-stage pipeline ARM organization.

The principal components of an ARM organization with a 3-stage pipeline 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 (pc).
- 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 incrementer, which select and hold all memory addresses and generate sequential addresses when required.
- The data register, which hold data passing to and from memory.
- The instruction decoder and associated control logic.

1) The 3-stage pipeline

ARM processors up to the ARM7 employ a simple 3-stage pipeline with the following 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 datapath:

j The register bank is read.

k An operand is shifted.

l The ALU result is generated, and written back into a destination register.

Fig. ARM single-cycle instruction 3-stage pipeline operation.

At any one time, three different instructions may occupy each of these stages, so the hardware in each stage has to be capable of independent operation.

F When the processor is executing simple data processing instructions the pipeline enables one instruction to be completed every clock cycle (that is, the throughput is one instruction per cycle), although an individual instruction takes three clock cycles to complete, that is, it has a three-cycle latency

F When a multi-cycle instruction is executed, as illustrated in Fig. 4.3, the pipeline flow is less regular.

Fig. ARM multi-cycle instruction 3-stage pipeline operation.

- The cycle colored in YELLOW is accessing main memory, so it can be seen that memory is used in every cycle.
- The DATAPATH 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 datapath 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.

2. **Explain 5-stage ARM pipeline organization in detail. Explain the concept of data forwarding in this architecture.**

5-stage pipeline ARM organization

1) Execution performance

(11)

- : the time required to execute a given program.
- : the number of ARM instructions executed in the program.
- : the average number of clock cycles per instruction.

- : the processor's clock frequency.

There are some ways to increase performance:

- Increase the clock rate, .

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, .

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.

1) 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 ahigher clock frequency to be used (provided that other system components, and particularly the instruction memory, are also redesigned to operate at this higher clock rate

2) The separate instruction and data memories (which may be separate caches connected to a unified instruction and data main memory) allow a significant reduction in the core's CPI.

2) The 5-stage pipeline

The 5-stage pipeline has stages:

- Fetch

The instruction is fetched from memory and placed in the instruction pipeline.

- Decode

The instruction is decoded.

- Execute

An operand is shifted.

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 (that is, for LOAD or STORE instructions). 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.

This 5-stage pipeline has been used for many RISC processors and considered to be the classic way to design such a processor.

3) Data forwarding in the 5-stage pipeline

- Instruction execution in the 3-stage pipeline is spread across 3 pipeline stages (Execute, Buffer/Data, and Write-back) in the 5-stage pipeline. The only one way to solvedata dependencies without stalling the 5-stage 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. The 5-stage ARM pipeline requires each of the 3 source operands to be forwarded from any of 3 intermediate result registers.

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

```
        LDR     rN, [ . . ]        ; load rN from somewhere, N: 0~15
        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 (rN) is needed by the following instruction at the start of the execute stage.
- The only way to avoid this stall is to encourage the C compiler (or assembly language programmer) not to put a dependent instruction immediately after a load instruction.
- Since the 3-stage pipeline ARM cores are not adversely affected by this code sequence, existing ARM programs will often use it. Such programs will run correctly on5-stage ARM cores, but could probably be rewritten

to run faster by simply reordering the instructions to remove these dependencies.

3. Compare 3-stage and 5-stage ARM pipeline organization

The principal components of an ARM organization with a 3-stage pipeline 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 (pc).
- 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 incrementer, which select and hold all memory addresses and generate sequential addresses when required.
- The data register, which hold data passing to and from memory.
- The instruction decoder and associated control logic.

The 5-stage pipeline has stages:

- Fetch

The instruction is fetched from memory and placed in the instruction pipeline.

- Decode

The instruction is decoded.

- Execute

An operand is shifted.
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 (that is, for LOAD or STORE instructions). 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.

4. Explain in detail, the SMART CARD system.

PROCESSOR USED FOR SMART CARD

The ARM Cortex-M3 processor, which forms the foundation of the Secure Core SC300 processor, was specifically developed to target the low-cost requirements of a broad range of markets and applications where memory and processor size significantly impact device costs. The Cortex-M3 processor brings together multiple technologies to reduce memory size while delivering industry-leading performance in a small power efficient RISC core and delivers an ideal platform to accelerate the migration of thousands of applications around the globe from legacy components to 32-bit microcontrollers.

The ARM Secure Core SC300 processor is designed specifically for high performance smartcard and embedded security applications. The SC300 combines the benefits of the industry standard Cortex-M3 processor with the proven security features of ARM Secure Core processors; currently the most widely licensed 32-bit CPU for smartcards worldwide.

Key Benefits

High performance. The processor executes the Thumb®-2 instruction set, including hardware division, single cycle multiply, and bit-field manipulation with low dynamic power.
Fast time to market. Based on the same architecture as Cortex-M3 the industry standard making software development straightforward.
Developers also benefit from the ARM partnership's extensive ecosystem of embedded tools, software, and knowledgebase.

Applications

The SC300 embeds counter measures against side channels attacks and fault injections. The SC300 processor's high performance makes it an ideal choice for a range of secure applications.

5. **Explain the history of ARM implementation.**

ARM, originally Acorn RISC Machine, later Advanced RISC Machine, is a family of reduced instruction set computing (RISC) architectures for computer processors, configured for various environments. British company ARM Holdings develops the architecture and licenses it to other companies, who design their own products that implement one of those architectures—including systems-on-chips (SoC) that incorporate memory, interfaces, radios, etc. It also designs cores that implement this instruction set and licenses these designs to a number of companies that incorporate those core designs into their own products.

A RISC-based computer design approach means processors require fewer transistors than typical complex instruction set computing (CISC) x86 processors in most personal computers. This approach reduces costs, heat and power use. Such reductions are desirable traits for light, portable, battery-powered devices—including smartphones, laptops, tablet and notepad computers, and other embedded systems.

The official Acorn RISC Machine project started in October 1983. The first samples of ARM silicon worked properly when first received and tested on 26 April 1985.
The first ARM application was as a second processor for the BBC Micro, where it helped in developing simulation software to finish development of the support chips (VIDC, IOC, MEMC), and sped up the CAD software used in ARM2 development. Wilson subsequently rewrote BBC BASIC in ARM assembly language. The in-depth knowledge gained from designing the instruction set enabled the code to be very dense, making ARM BBC BASIC an extremely good test for any ARM emulator. The original aim of a principally ARM-based computer was achieved in 1987 with the release of the Acorn Archimedes.In 1992, Acorn once more won the Queen's Award for Technology for the ARM.
The ARM2 featured a 32-bit data bus, 26-bit address space and 27 32-bit registers. Eight bits from the program counter register were available for other purposes; the top six bits (available because of the 26-bit address space) served as status flags, and the bottom two bits (available because the program counter was always word-aligned) were used for setting modes. The address bus was extended to 32 bits in the ARM6, but program code still had to lie within the first 64 MB of memory in 26-bit compatibility mode, due to the reserved bits for the status flags.The ARM2 had a transistor count of just 30,000, compared to Motorola's six-year-older 68000 model with around 40,000. Much of this simplicity came from the lack of microcode (which represents about one-quarter to one-third of the 68000) and from (like most CPUs of the day) not including any cache. This simplicity enabled low power consumption, yet better performance than the Intel 80286. A successor, ARM3, was produced with a 4 KB cache, which further improved performance.

6.**Using suitable examples, explain the various instruction sets of ARM processor (Nov/Dec 2016)**
Processor Modes
* The ARM has six operating modes: • User (unprivileged mode under which most tasks run) • FIQ (entered when a high priority (fast) interrupt is raised) • IRQ (entered when a low priority (normal) interrupt is raised) • Supervisor (entered on reset and when a Software Interrupt instruction is executed) • Abort (used to handle memory access violations) • Undef (used to handle undefined instructions) * ARM Architecture Version 4 adds a seventh mode: • System (privileged mode using the same registers as user mode)
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 * However these are arranged into several banks, with the accessible bank being governed by the processor mode. Each mode can access • a particular set of r0-r12 registers • a particular r13 (the stack pointer) and r14 (link register) • r15 (the program counter) • cpsr (the current program status register) and privileged modes can also access • a particular spsr (saved program status registers.
Register Organisation General registers and Program Counter Program Status Registers
Accessing Registers using ARM Instructions * No breakdown of currently accessible registers. • All instructions can access r0-r14 directly. • Most instructions also allow use of the PC. * Specific instructions to allow access to CPSR and SPSR.

The Program Status Registers (CPSR and SPSRs) Copies of the ALU status flags (latched if the instruction has the "S" bit set). N = Negative result from ALU flag. Z = Zero result from ALU flag. C = ALU operation Carried out V = ALU operation oVerflowed * Interrupt Disable bits. I = 1, disables the IRQ. F = 1, disables the FIQ. * T Bit (Architecture v4T only) T = 0, Processor in ARM state T = 1, Processor in Thumb state * Condition Code Flags N Z C V Mode 31 28 8 4 0 I F T * Mode Bits M[4:0] define the processor mode.
The Instruction Pipeline * The ARM uses a pipeline in order to increase the speed of the flow of instructions to

the processor. • Allows several operations to be undertaken simultaneously, rather than serially. * Rather than pointing to the instruction being executed, the PC points to the instruction being fetched. FETCH DECODE EXECUTE Instruction fetched from memory Decoding of registers used in instruction Register(s) read from Register Bank Shift and ALU operation Write register(s) back to Register.

7.**Explain the co-processor interface of ARM in detail. (Nov/Dec 2016)**

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.

1) Coprocessor architecture

- 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 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.

2) 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 behavior 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. (from ARM to all coprocessors): This signal indicates that the ARM has ***identified*** a coprocessor instruction and wishes to execute it.
2. (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. (from the coprocessors to ARM): This is the '***CoProcessor Busy***' signal which tells the ARM that the coprocessor cannot begin executing the instruction yet.

2-1) Handshake outcomes

Once a coprocessor instruction has entered the ARM7TDMI and coprocessor pipelines, there are ***4 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. (All ARM instructions are**conditionally executed**, including coprocessor instructions.) **ARM will not assert , and the instruction will be discarded by all parties**.
2. The ARM may decide to execute it (and signal this by asserting cpi, but no coprocessor can take it so 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.
4. ARM decides to execute the instruction and a coprocessor accepts it for immediate execution. and are all taken low and both sides commit to complete the instruction.

2-2) Data transfers

If the instruction is a coprocessor data transfer instruction, the ARM is responsible for generating an initial memory address (the coprocessor does not require any connection to the address bus) but the coprocessor determines the length of the transfer; ARM will continue incrementing the address until the coprocessor signals completion.