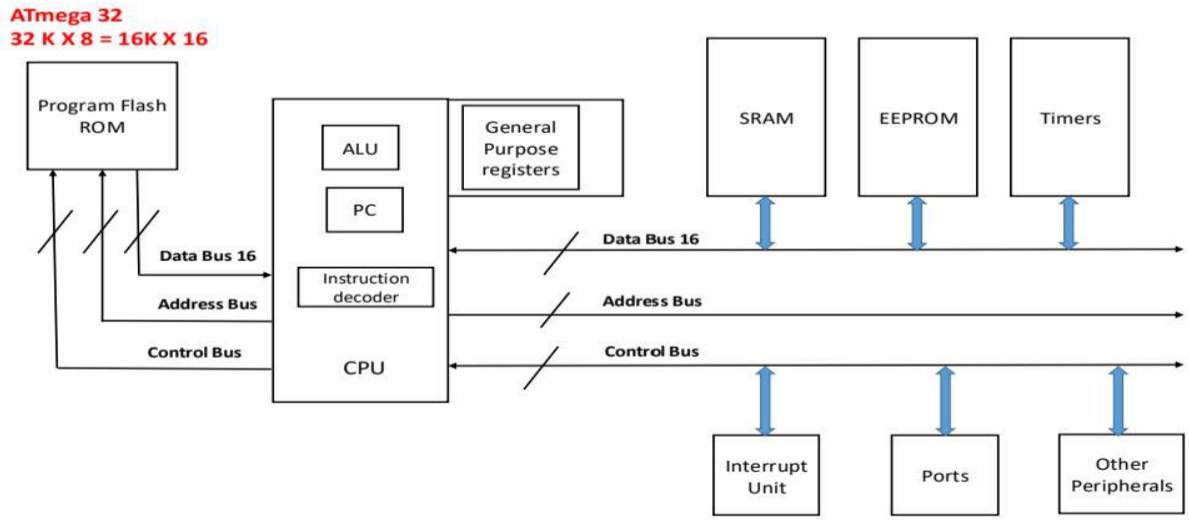
Q1 Compare microcontroller and microprocessor

Sr. No.	Microprocessors	Microcontroller
1	It is only a general purpose computer CPU	It is a microcomputer itself
2	Memory, I/O ports, timers, interrupts are not available inside the chip	All are integrated inside the microcontroller chip
3	Systems become bulkier and expensive.	Make the system simple, economic and compact
4	Microprocessors have many opcodes for moving the data from external memory to CPU.	Microcontrollers have one or two opcodes for moving the data.
5	Higher accessing time required	Low accessing time
6	Very few number of bit handling instructions	Many bit handling instructions
7	Very few pins are programmable	Most of the pins are programmable
8	Widely Used in modern PC and laptops	widely used in small control systems and dedicated applications

Q2 Draw and describe the architecture of AVR

AVR architectural overview :



AVR architectural overview :

Data BUS 8 bit : Its an 8-bit parallel data lines by which the data travels inside the MCU (NOTE: this is the reason why AVR is an 8-bit MCU).

ALU: Arithmetic Logic Unit, the core/heart of the entire system where typically all commands get executed.

Data SRAM: It basically pretty similar to the RAM (Random access memory) we see inside our computers.

EEPROM: (Electrically Erasable Programmable Read Only Memory) its very similar to another component in our computer namely Hard Disk, i.e. a permanent storage.

I/O lines: These are the bunch of registers which is used as a switches or controls for different features of AVR.

32X8 GPR (General Purpose Registers): This are 32 registers each having 8-bit which is a general storage space for data. But, remember, SRAM is also a temporary storage but these registers have some specialty among all.

Status & Control : A couple of registers which are very special to the MCU & to us also.

Program Counter: This is a register which has a responsibility to track the position of the program that is currently executing.

Flash Memory: It is also a permanent storage but its only for storing the program we write to it.

Instruction Register & Decoder: These are import for the MCU but not too much to us.

Q3 List the features of AVR

1. 8-bit RISC core with 32 × 8-bit working registers.
2. Single-cycle execution for most arithmetic/ logical ops.
3. Separate program (Flash) and data (SRAM) memories — Harvard architecture.
4. In-system programmable Flash (ISP), Read-While-Write.
5. EEPROM for nonvolatile byte storage.
6. Multiple timers/counters (8- and 16-bit), PWM capability.
7. On-chip ADC (multi-channel), comparators.
8. Serial interfaces: USART, SPI, TWI(I^2C).
9. Watchdog timer, power-save modes.
10. Interrupt system with vectored interrupts.

Q4. Draw the Family structure of AVR

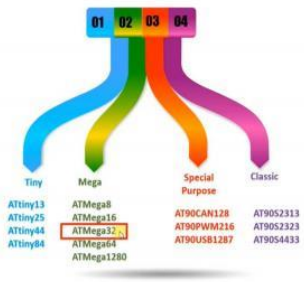
Introduction to AVR

- AVR MCUs are extremely popular MCUs used in various application specially in project prototyping and embedded devices.
- AVR is a **8-bit RISC architecture** (Reduced Instruction Set Computing) microcontroller in market since 1996 which is having on-chip programmable flash memory, SRAM, IO data space & EEPROM.
- AVR is the first MCU in market which has on-chip flash storage.

Introduction to AVR Family

- Developed by Atmel corporation in 1996 and acquired by Microchip Technology in 2016.
- Advance Virtual Risc or Alf and Vegard Risc
- AVR is an 8 bit RISC based single chip microprocessor with Harvard architecture
- It has Code ROM, Data ROM (EEPROM), RAM, I/O and Timers
- Supports some serial interfaces such as CAN, I2C, SPI, USB, USART.

AVR Family Classification



Q5. Discuss the ROM , RAM , SRAM and EEPROM of AVR

1. **ROM (Read-Only Memory) in AVR**

· This is non-volatile memory used to store the program code (firmware), constant data, and the interrupt vector table. Non-volatile means the content remains even when power is removed.

· In AVR microcontrollers, ROM is implemented as Flash Memory. This is a significant point, as "ROM" often implies one-time programmable, but Flash is re-programmable.

· Non-volatile: Retains data without power.

· The Flash memory can be erased and rewritten thousands of times (typically ~10,000 write/erase cycles)

2. **RAM (Random-Access Memory) in AVR**

· This is volatile memory used for temporary data storage during program execution. Volatile means the content is lost when power is removed.

· The primary RAM in all AVR microcontrollers is SRAM (Static RAM).

· Volatile: Loses all data when power is cycled.

· Data can be read from and written to any location at any time without a special procedure.

3. **SRAM (Static RAM) – The Specific Type of RAM**

SRAM is the specific technology used for the main RAM in AVR MCUs. It's helpful to understand what "Static" means.

· SRAM uses a flip-flop circuit (typically 6 transistors) to store each bit. This circuit will hold its value as long as power is supplied; it does not need to be "refreshed."

· Advantages:

· Much faster than DRAM (Dynamic RAM, which needs refreshing).

· Does not require a refresh controller, simplifying the MCU design.

4. **EEPROM (Electrically Erasable Programmable Read-Only Memory)**

· This is a special non-volatile memory area separate from the program Flash. It is designed for storing data that needs to be saved between power cycles but may need to be updated during program execution.

· Retains data without power.

· Unlike Flash memory, which often requires erasing a large page (e.g., 128 bytes), you can erase and rewrite a single byte in EEPROM without affecting others. This is its biggest advantage.

· Writing to EEPROM is relatively slow (can take several milliseconds) and requires a specific software sequence to initiate.

Q6. List the Application of Microcontroller

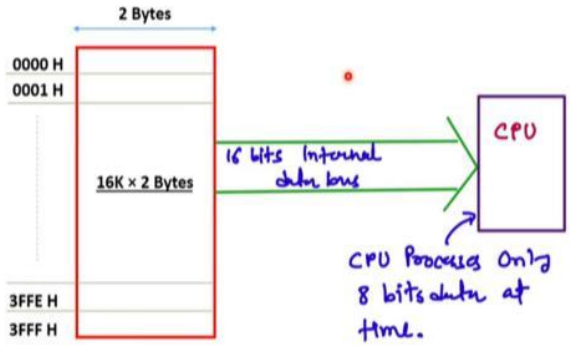
• A microcontroller (MCU) is a small computer on a single integrated circuit. That is designed to control specific tasks within electronic systems. It Combines the functions of a central processing unit (CPU), memory, and Input/output interfaces, all on a single chip.

• Microcontrollers are widely used in embedded systems, such as home Appliances, automotive systems, medical devices, and industrial control Systems.

• Microcontrollers are widely used in various different devices such as –

- Light sensing and controlling devices like LED.
- Temperature sensing and controlling devices like microwave oven, Chimneys.
- Fire detection and safety devices like Fire alarm.
- Measuring devices like Volt Meter.

Q7 Draw and explain the ROM architecture of Atmega32



Memory Organization of Atmega32

1.Flash Memory (Program Memory):
- The ATmega32 typically comes with 32 KB of Flash memory.
- Flash memory is where the program code is stored. It's non-volatile, meaning the code remains even when power is removed.
- Flash memory is organized into 16-bit words.

2.SRAM (Static Random Access Memory):
- The ATmega32 has 2 KB of SRAM.
- SRAM is used for storing variables, intermediate results, and stack data during program execution.
- Unlike Flash memory, SRAM is volatile and loses its content when power is removed.

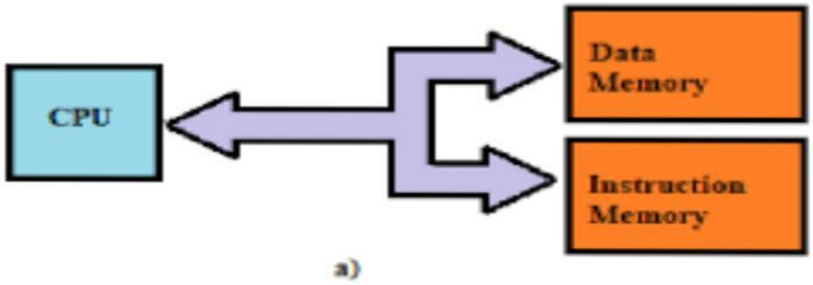
3.EEPROM (Electrically Erasable Programmable Read-Only Memory):
- The ATmega32 features 1 KB of EEPROM.
- EEPROM is non-volatile memory that can be written to and read from by the AVR microcontroller during program execution.
- It's commonly used for storing non-volatile data that needs to be retained even when the power is turned off, such as configuration settings or small amounts of user data.

4.I/O Registers:
- These are memory-mapped registers used for interfacing with peripherals and controlling the microcontroller's operation.
- I/O Registers control various aspects of the microcontroller, including GPIO (General Purpose Input/Output), timers, UART, SPI, and other peripherals.
- They are often used for configuring and controlling the behavior of the microcontroller's peripherals.

5.Special Function Registers (SFRs):
- These registers provide control and status information for the CPU and peripherals.
- SFRs are used to configure the operation of the microcontroller, control interrupts, and manage power-saving modes.
- They also include status flags and control bits for various functions of the microcontroller.

Q8 Draw and explain the Harvard architecture of AVR

The Harvard Architecture is a computer architecture that utilizes separate storage and signal pathways (buses) for instructions and data. This means the CPU can simultaneously fetch instructions from instruction memory and access data from data memory, leading to increased performance compared to architectures like the Von Neumann architecture, which uses a single



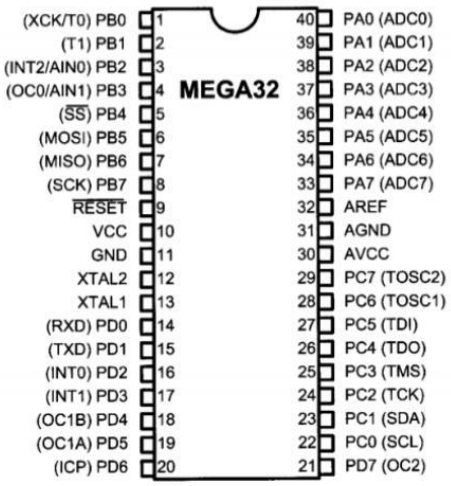
a)

shared memory and bus for both.

Q1 Explain the I/O programming, lists its ports and registers

AVR I/O Port Programming

- In AVR microcontroller family, there are many ports available for I/O operations, depending on which family microcontroller you choose.
- For the ATmega32 40-pin chip 32 Pins are available for I/O operation. The four ports PORTA, PORTB, PORTC, and PORTD are programmed for performing desired operation.



The number of ports in AVR family varies depending on number of pins available on chip.

Pins	8-pin	28-pin	40-pin	64-pin	100-pin
Chip	ATtiny25/45/85	ATmega8/48/88	ATmega32/16	ATmega64/128	ATmega1280
Port A			X	X	X
Port B	6 bits	X	X	X	X
Port C		7 bits	X	X	X
Port D		X	X	X	X
Port E				X	X
Port F				X	X
Port G				5 bits	6 bits
Port H					X
Port J					X
Port K					X
Port L					X

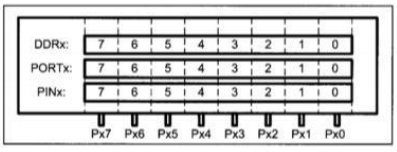
Note: X indicates that the port is available.

- The 40-pin AVR has four ports for using any of the ports as an input or output port, it must be accordingly programmed.
- The Registers Addresses for ATmega32 Ports is given below:
  PORTx,
  DDRx
  PINx
- Each port in AVR microcontroller has three I/O registers associated with it. They are designated as.
- Each of I/O registers is 8 bits wide, and each port has a maximum of 8 pins, therefore each bit of I/O registers affects one of the pins.
- For accessing I/O registers associated with the ports the common relationship between the registers and the pins of AVR microcontroller

Table 4-2: Register Addresses for ATmega32 Ports

Port	Address	Usage
PORTA	$3B	output
DDRA	$3A	direction
PINA	$39	input
PORTB	$38	output
DDRB	$37	direction
PINB	$36	input
PORTC	$35	output
DDRC	$34	direction
PINC	$33	input
PORTD	$32	output
DDRD	$31	direction
PIND	$30	input

The relation between the Registers and the Pins of AVR is shown below:



Q2 Explain the registers for the PORTB and draw its relation with Port pins

DDRx: Data Direction Register

- Before reading or writing the data from the ports, their direction needs to be set. Unless the PORT is configured as output, the data from the registers will not go to controller pins.
- This register is used to configure the PORT pins as Input or Output.
- Writing 1's to DDRx will make the corresponding PORTx pins as output.
- Similarly writing 0's to DDRx will make the corresponding PORTx pins as Input.

Note: DDRx (X – any port from (PORT A ,PORT B PORT C, PORT D)

DDRx	Port Type
1's	O/P
0's	I/P

Example :
1. if we want to make port B as Output : DDRB = 0xFF;
2. If we want to make 6th pin of port A is 0 : DDRA.5 = 0x40;

Example :

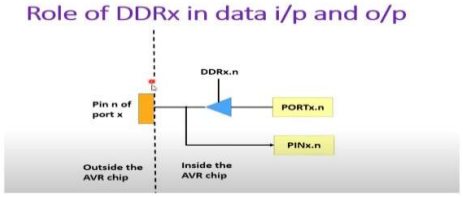
```
DDRB = 0xff; // Configure PORTB as Output.
DDRC = 0x00; // Configure PORTC as Input.
DDRD = 0x0F; // Configure lower nibble of PORTD as
               Output and higher nibble as Input
```

PORT A

8th	7th	6th	5th	4th	3rd	2nd	1st
0	0	1	0	0	0	0	0
7	6	5	4	3	2	1	0

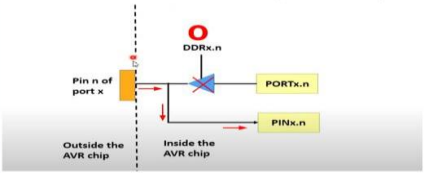
Pins of Port A

Role of DDRx in data i/p and o/p



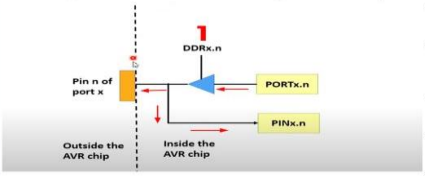
- DDRx register is 8 bit internal register of AVR. The role of DDRx register is to set physical pin of port input or output.
- 1 = O/P Pin
- 0 = I/P Pin
- PORT register is 8 bit internal register of AVR. The role of PORT register is to send the data on pin n of port x.
- Any output device i.e. LED or LCD is connected with physical pin will turn ON or OFF by sending data through PORTx register.
- PINx is also 8 bit internal register. The role of this register is to read data from input device like switch PUSH button which is connected with physical pin of AVR.

I/P port working :



- The role of DDRx register is to set physical pin of port input or output. 1 = O/P Pin 0 = I/P Pin
- When I/P device is connected with physical pin of AVR, put 0 in DDRx register to make that port as an input port.
- Tristate buffer which is connected with DDRx register will get disable as soon as it received 0.
- Once it get disable the data from input device will come in to the PINx register.
- Hence we can read the status of an input device.

O/P port working :



- The role of DDRx register is to set physical pin of port input or output. 1 = O/P Pin 0 = I/P Pin
- When O/P device is connected with physical pin of AVR, put 1 in DDRx register to make that port as an output port.
- Tristate buffer which is connected with DDRx register will get enable as soon as it received 1.
- Once it get enable the data from an output device will transfer on the device through PINx register.
- Hence we can send the data to an output device.

PORTx: PORTx register can be used for two purposes:

1. To output data:
- when port is configured as output then PORTx register is used.
- When we set bits in DDRx to 1, corresponding pins becomes output pins.
- Now we can write the data into respective bits in PORTx register.
- This will immediately change the output state of pins according to data we have written on the ports.

For example:

1. To output data in variable x on port A

```
1. DDRA = 0xFF;          //make port A as outputs
2. PORTA = x;            //output variable on port
```

2. To output 0xFF data on port B

```
1. DDRB = 0b11111111;    //set all the pins of port B as outputs
2. PORTB = 0xFF;          //write the data on port
```

3. To output data on only 0th bit of port C

```
1. DDRC.0 = 1;     //set only 0th pin of port C as an output
2. PORTC.0 = 1;    //make it high signal.
```

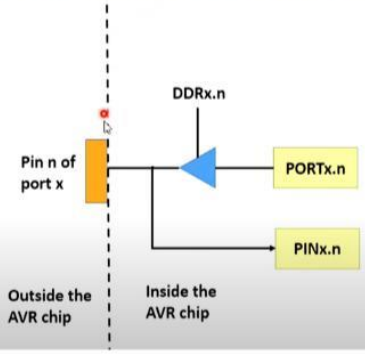
PINx register:

- PINx register used to read the data from port pins. In order to read the data from port pin, first we have to change the portxs data direction to input.
- This is done by setting bits in DDRx to zero. If port is made output, then reading PINx register will give a data that has been output on port pins.
- There are two input modes.
- Either we can use port pins as internal pull up or as tri stated inputs. It will be explained as shown below:
- For reading the data from port A.

```
1. DDRA = 0x00;   //Set port A as input
2. x = PINA;      //Read contents of port a
```

Q3. Explain with diagram the role of DDRxn in inputting and Outputting the data

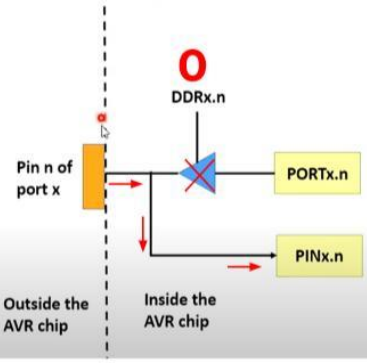
Role of DDRx in data i/p and o/p



- DDRx register is 8 bit internal register of AVR. The role of DDRx register is to set physical pin of port input or output.
- 1 = O/P Pin
- 0 = I/P Pin

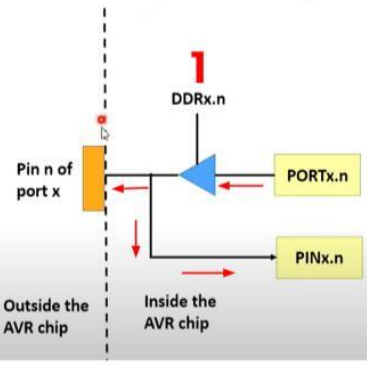
- PORT register is 8 bit internal register of AVR. The role of PORT register is to send the data on pin n of port x.
- Any output device i.e. LED or LCD is connected with physical pin will turn ON or OFF by sending data through PORTx register.
- PINx is also 8 bit internal register. The role of this register is to read data from input device like switch PUSH button which is connected with physical pin of AVR.

I/P port working :



- The role of DDRx register is to set physical pin of port input or output. 1 = O/P Pin 0 = I/P Pin
- When I/P device is connected with physical pin of AVR, put 0 in DDRx register to make that port as an input port.
- Tristate buffer which is connected with DDRx register will get disable as soon as it received 0.
- Once it get disable the data from input device will come in to the PINx register.
- Hence we can read the status of an input device.

O/P port working :



- The role of DDRx register is to set physical pin of port input or output. 1 = O/P Pin 0 = I/P Pin
- When O/P device is connected with physical pin of AVR, put 1 in DDRx register to make that port as an output port.
- Tristate buffer which is connected with DDRx register will get enable as soon as it received 1.
- Once it get enable the data from an output device will transfer on the device through PINx register.
- Hence we can send the data to an output device.

Q4. Write C program to send data 25H to port A(similar Questions)

```
#include <avr/io.h>
Int main(void) {
   DDRA = 0xFF;     // configure PORTA pins as outputs
   PORTA = 0x25;    // send 0x25 to PORTA
   While (1) {
      // idle
   }
   Return 0;
}
```

Q5Write the value of the registers for I/O programing for the

Following the statements (data may vary)

1) To output data in variable x on port A
   DDRA = 0xFF; // all PORTA pins outputs
   PORTA = x;   // write value of x

2) To output 0xFF data on port B
   DDRB = 0xFF; // PORTB as outputs
   PORTB = 0xFF; // set all pins high

3) To read the data from port A
   DDRA = 0x00; // PORTA as inputs
   Uint8_t val = PINA; // read pins
   // if you want pull-ups: PORTA = 0xFF; (with DDRA=0x00)

4) To output data on only 0^{th} bit of port C
   DDRC |= (1<<PC0);   // set PC0 as output (others unchanged)
   If (value & 0x01) PORTC |= (1<<PC0); // set PC0
   Else PORTC &= ~(1<<PC0);   // clear PC0

Q6. Define timer and Interrupts, list the timer and interrupts in AVR

Definition of Timer :- A timer is an essential part of any modern Microcontroller Unit (MCU)1 and shares the same internal unit as a counter.

Definition of Interrupts :- Interrupts are one of two primary methods (the other being polling) a microcontroller uses to serve several connected devices.

The timers in the AVR ATmega16/ATmega32 family are:

Timer0: An 8-bit timer.

Timer1: A 16-bit timer.

Timer2: An 8-bit timer.

Sources of Interrupts in the AVR :-The AVR has many sources of interrupts, which depend on the peripherals incorporated into the chip10. The following are listed as some of the most widely used sources of interrupts in the AVR:

Timer Interrupts: Each timer has at least two associated interrupts.

One interrupt for overflow10.

Another interrupt for a compare match.

External Hardware Interrupts: Three interrupts are set aside for external hardware interrupts:

INT0 (associated with PORTD.2).

INT1 (associated with PORTD.3).

INT2 (associated with PORTB.2).

Serial Communication (USART) Interrupts: The USART has three interrupts:

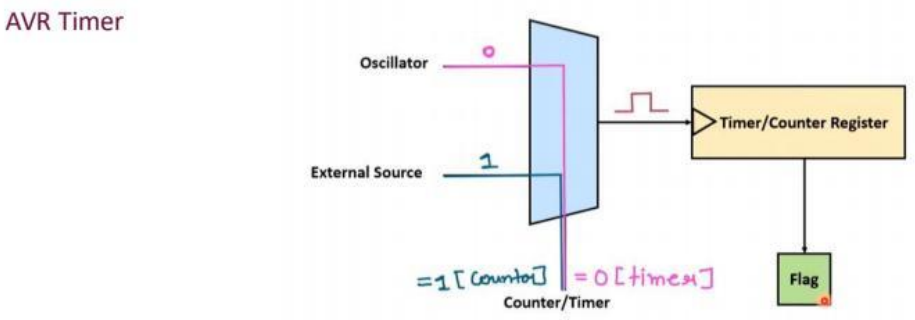
One interrupt for receiving.

Two interrupts for transmit.

The SPI interrupts.

The ADC interrupts

Q7. Define timers and explain types of timers



- Timers come in handy when you want to set some time interval like your alarm. This can be very precise to a few microseconds.
- Timers/Counters are essential part of any modern MCU. Remember it is the same hardware unit inside the MCU that is used either as Timers or Counter. Timers/counters are an independent unit inside a micro-controller. They basically run independently of what task CPU is performing. Hence they come in very handy, and are primarily used for the following:
  1. **Internal Timer**: As an internal timer the unit, ticks on the oscillator frequency. The oscillator frequency can be directly feed to the timer or it can be pre-scaled. In this mode it used generate precise delays. Or as precise time counting machine.
  2. **External Counter**: In this mode the unit is used to count events on a specific external pin on a MCU.
  3. **Pulse width Modulation(PWM) Generator**: PWM is used in speed control of motors and various other applications.
- Atmega32 has 3 timer units, timer 0, timer 1 and timer 2 respectively.

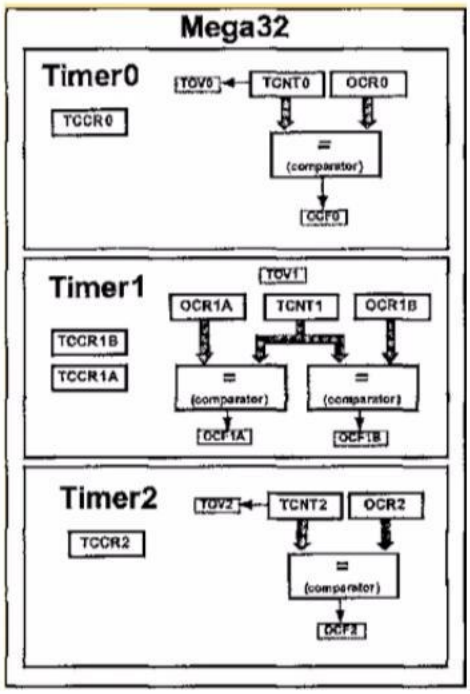
Q8. Explain with neat diagram the timer0/1/2 and its registers

Basic Registers of Timers

In AVR ATmega16 / ATmega32, there are three timers:
- **Timer0**: 8-bit timer
- **Timer1**: 16-bit timer
- **Timer2**: 8-bit timer

Basic registers and flags of the Timers :

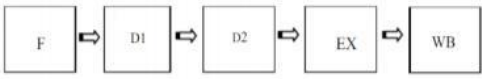
TCNTn: Timer / Counter Register :Every timer has a timer/counter register. It is zero upon reset. We can access value or write a value to this register. It counts up with each clock pulse.
TOVn: Timer Overflow Flag : Each timer has a Timer Overflow flag. When the timer overflows, this flag will get set.
TCCRn: Timer Counter Control Register: This register is used for setting the modes of timer/counter.
OCRn: Output Compare Register: The value in this register is compared with the content of the TCNTn register. When they are equal, the OCFn flag will get set.



Q1. List any Six/eight features of Pentium processor

Features of Pentium Processor :

- Pentium Processor is **fifth generation** microprocessor evolved in 1993.
- It has a **32-bit address** bus
- Address bus is 32-bit hence can access **2^{32} = 4GB memory**.
- It has **64-bit data** bus
- Pentium Processor designed to operate from **60MHZ to 233 MHZ**
- **It has 5 stage pipelining .**

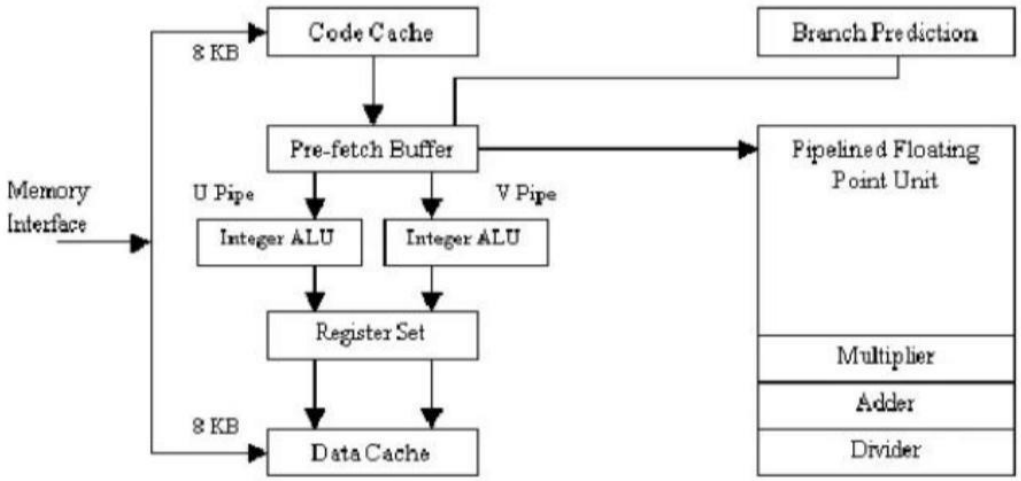


- **It is 2 way superscalar : U pipe , V pipe**
- **On chip floating point unit**
- **Branch prediction logic : 256 entry BTB**
- **On chip cache memory**
- **Instruction cache** has **8K bytes** memory
- **Data cache** has **8K bytes** memory

- **Cache is** a small amount of memory which **is** a part of the **CPU** - closer to the **CPU** than RAM .
- It **is** used to temporarily hold **instructions** and **data** that the **CPU is** likely to **reuse**.

Q2. Draw and explain Pentium superscalar architecture

Pentium superscalar architecture :



- Processors capable of parallel instruction execution of multiple instructions are known as Superscalar.
- The Pentium processor is a superscalar machine, capable of executing two instructions in parallel.
- The process of issuing two instructions in parallel is called as dual pairing.
- Given figure presents a block diagram overview of the Pentium Processor including two instruction pipelines because of dual pipelining feature.
  1. "U" pipe : The U-pipe can execute all integer and floating point instructions.
  2. "V" pipe : The V-pipe can execute simple integer instructions and floating point instructions.
- There are two separate cashes are there in the block diagram :
  1. Instruction cache (code cache)
  2. Data cache
- There are 2 separate 32 bit ALU to perform 64 bit data operation.
- 5-stage or dual pipelining and branch prediction are two advance and important features of Pentium processor.

Q3. State need of branch prediction logic of Pentium processor

Instruction Branch Prediction:

Why do we need branch prediction?

1. The gain produced by Pipelining can be reduced by the presence of program transfer instructions eg JMP, CALL, RET etc .
2. They change the sequence causing all the instructions that entered the pipeline after program transfer instructions invalid.
3. Thus no work is done as the pipeline stages are reloaded.

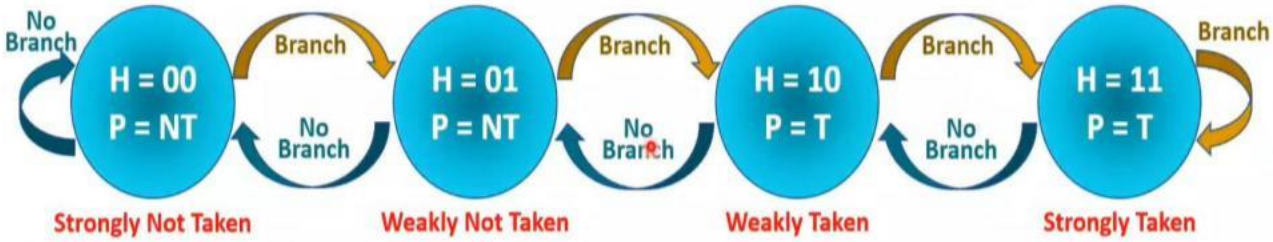
Branch prediction logic:

- To avoid this problem, Pentium uses a scheme called Dynamic Branch Prediction.
- In this scheme, a prediction is made for the branch instruction currently in the pipeline. The prediction will either be taken or not taken.
- If the prediction is true then the pipeline will not be flushed and no clock cycles will be lost.
- If the prediction is false then the pipeline is flushed and starts over with the current instruction.
- It is implemented using 4 way set associated cache with 256 entries.
- This is called **Branch Target Buffer (BTB)**.
- The directory entry for each line consists of:
  Valid bit: Indicates whether the entry is valid or not.
  History bit: Track how often bit has been taken.
- Source memory address is from where the branch instruction was fetched. If the directory entry is valid then the target address of the branch is stored in corresponding data entry in BTB.

Q5. Explain Branch prediction logic of Pentium processor

Branch Prediction Logic

- Pentium uses a scheme called Dynamic Branch Prediction. In this scheme, a prediction is made for the branch instruction currently in the pipeline.
- If the prediction is true then the pipeline will not be flushed and no clock cycles will be lost. If the prediction is false then the pipeline is flushed and starts over with the current instruction.
- It is implemented using 4 way set associated memory with 256 entries. This is called Branch Target Buffer (BTB).
- Here, 4 way set is defined by history bits. Based on history bits prediction is fixed.

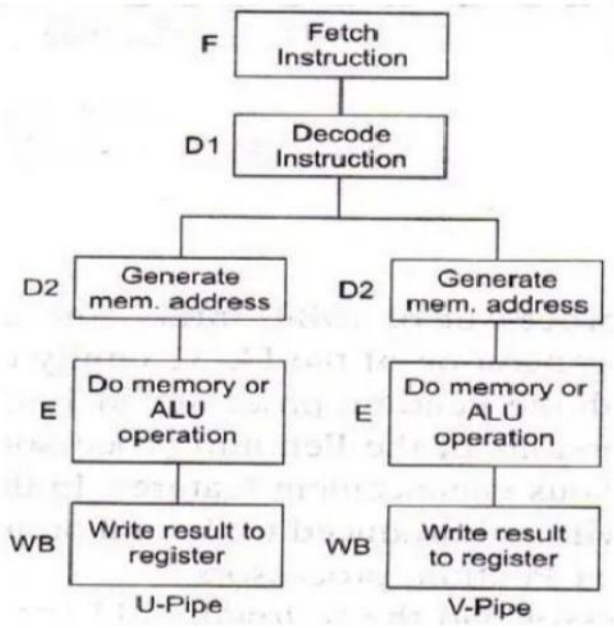


History Bits	Branch Prediction
00	Not Taken
01	Not Taken
10	Taken
11	Taken

- For H bits = 00 or 01 branch prediction is not taken and For H bits = 10 and 11 branch prediction is taken.
- For false prediction, Pipeline will gets flushed.
- If branch is executed then H bits data will be incremented by one, it means branch prediction priority will increase.
- If branch is not executed then H bits data will be decremented by one, it means branch prediction priority will decrease.

Q6. Draw and explain all phases of Pentium processor pipelining

Pipelining :



The Pipeline Stages

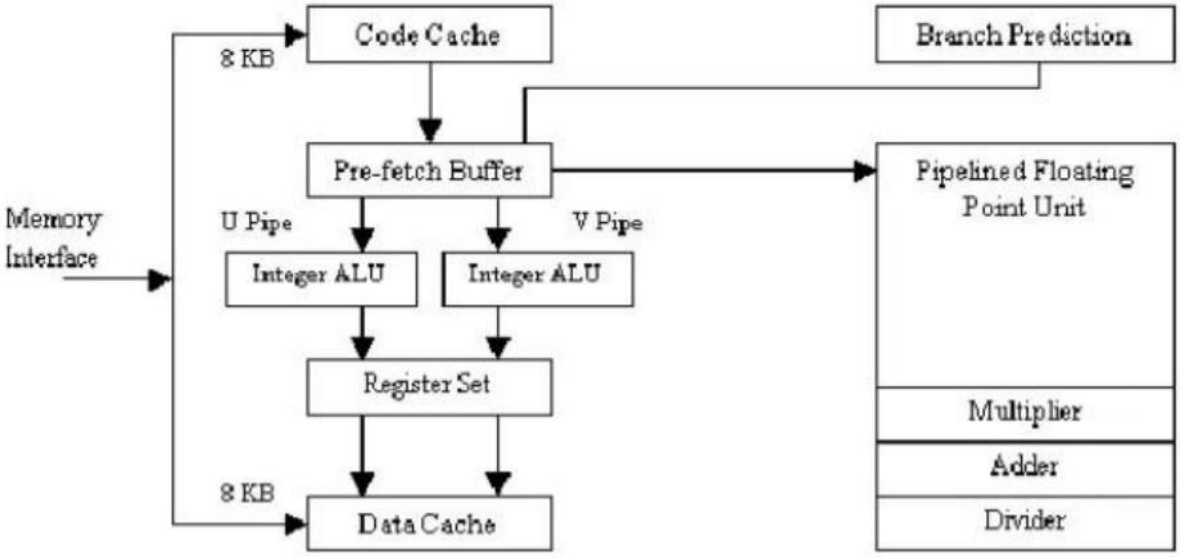
F - Instruction Fetch Stage
D1 - Decode Stage 1
D2 - Decode Stage 2
E - Execute Memory or ALU
WB - Write back to destination

Pipelining :

Each of the two pipelines have five stages:

1. **Prefetch (F) :** Instructions are prefetched from instruction cache or memory.
2. **D1 instruction decode :** In the D1 stage, the processor decodes the instruction to generate a control word. A single control word executes instruction directly.
3. **D2 Address generate :** In the D2 stage, the processor decodes the control word from D1 for use in the EX stage.
4. **Execute (EX) :** In this stage, the processor either accesses the data cache or calculates results in the ALU.
5. **Write Back (WB) :** In the WB stage, the processor updates the registers and flags with the instructions result.

Q7. Draw architecture of Pentium processor and explain following



1. Pipelining

Each of the two pipelines have five stages:

1. **Prefetch (F) :** Instructions are prefetched from instruction cache or memory.
2. **D1 instruction decode :** In the D1 stage, the processor decodes the instruction to generate a control word. A single control word executes instruction directly.
3. **D2 Address generate :** In the D2 stage, the processor decodes the control word from D1 for use in the EX stage.
4. **Execute (EX) :** In this stage, the processor either accesses the data cache or calculates results in the ALU.
5. **Write Back (WB) :** In the WB stage, the processor updates the registers and flags with the instructions result.

2. Branch Prediction Logic

Branch prediction logic:

- To avoid this problem, Pentium uses a scheme called Dynamic Branch Prediction.
- In this scheme, a prediction is made for the branch instruction currently in the pipeline. The prediction will either be taken or not taken.
- If the prediction is true then the pipeline will not be flushed and no clock cycles will be lost.
- If the prediction is false then the pipeline is flushed and starts over with the current instruction.
- It is implemented using 4 way set associated cache with 256 entries.
- This is called **Branch Target Buffer (BTB)**.
- The directory entry for each line consists of:
  Valid bit: Indicates whether the entry is valid or not.
  History bit: Track how often bit has been taken.
- Source memory address is from where the branch instruction was fetched. If the directory entry is valid then the target address of the branch is stored in corresponding data entry in BTB.

Working of Branch Prediction:

- BTB is a lookaside cache that sits to the side of Decode Instruction(DI) stage of 2 pipelines and monitors for branch instructions.
- The first time that a branch instruction enters the pipeline, the BTB uses its source memory to perform a lookup in the cache.
- Since the instruction was never seen before, it is BTB miss. It predicts that the branch will not be taken even though it is unconditional jump instruction.
- When the instruction reaches the EU(execution unit), the branch will either be taken or not taken.
- If taken, the next instruction to be executed will be fetched from the branch target address.
- If not taken, there will be a sequential fetch of instructions.
- When a branch is taken for the first time, the execution unit provides feedback to the branch prediction.
- The branch target address is sent back which is recorded in BTB.
- A directory entry is made containing the source memory address and history bit is set as strongly taken.