Unit-3

AVR MICROCONTROLLER

Contents

The AVR RISC microcontroller architecture:

AVR MICROCONTROLLER INTRODUCTION

- The architecture of AVR was developed by **Alf**-Egil Bogen and **Vegard** Wollan. AVR derives its name from its developers and stands for Alf-Egil Bogen Vegard Wollan RISC microcontroller, also known as Advanced Virtual RISC.
- The AT90S8515 was the first microcontroller which was based on AVR architecture.
- However the first microcontroller to hit the commercial market was AT90S1200 in the year 1997.
- **AVR microcontrollers** are available in three categories:

➢ **TinyAVR** – Less memory, small size, suitable only for simpler applications

➢ **MegaAVR** – These are the most popular ones having good amount of memory (upto 256 KB), higher number of inbuilt peripherals and **suitable for moderate to complex applications.**

➢ **XmegaAVR** – Used commercially for complex applications, which require large program memory and high speed.

AVR MICROCONTROLLER INTRODUCTION

- The following table compares the above mentioned AVR series of microcontrollers:

Series Name	Pins	Flash Memory	Special Feature
TinyAVR	6-32	0.5-8 KB	Small in size
MegaAVR	28-100	4-256KB	Extended peripherals
XmegaAVR	44-100	16-384KB	DMA , Event System included

AVR MICROCONTROLLER INTRODUCTION

- **What's special about AVR?**

	8051	PIC	AVR
SPEED	Slow	Moderate	Fast
MEMORY	Small	Large	Large
ARCHITECTURE	CISC	RISC	RISC
ADC	Not Present	Inbuilt	Inbuilt
Timers	Inbuilt	Inbuilt	Inbuilt
PWM Channels	Not Present	Inbuilt	Inbuilt

AVR MICROCONTROLLER INTRODUCTION

- **AVR is an 8-bit microcontroller belonging to the family of Reduced Instruction Set Computer** (**RISC**). In RISC architecture the instruction set of the computer are not only fewer in number but also simpler and faster in operation.
- This microcontroller is capable of transmitting and receiving 8-bit data. **The input/output registers available are of 8-bits.** The AVR family controllers have register based architecture which means that both the operands for an operation are stored in a register and the result of the operation is also stored in a register.
- The specific aspect of RISC is that almost **all instructions execute in one machine cycle**.
- The next RISC principle adopted by AVR is the **load-store architecture** which means that the only instructions that access memory are load and store.
- Load instruction loads data from memory into the register where as store moves the data from register into the memory.
- For an ALU instruction to be executed the operands are brought into the registers using LOAD and the result is stored back in the memory after execution.
- Another aspect of RISC architecture is **larger register file** which is taken care by the AVR design.

AVR MICROCONTROLLER INTRODUCTION

Atmega16 microcontroller, which is a 40-pin IC and belongs to the**megaAVR category of AVR** family. Some of the features of Atmega16 are:

- 16KB of ISP Flash memory
- 1KB of SRAM
- 512 Bytes of EEPROM
- Available in 40-Pin DIP
- 8-Channel 10-bit ADC
- Two 8-bit Timers/Counters
- One 16-bit Timer/Counter
- 4 PWM Channels
- Serial USART
- SPI Interface
- Digital to Analog Converter.

AVR MICROCONTROLLER Architecture

- The Atmel®AVR®ATmega32 is a low-power CMOS 8-bit microcontroller based on the AVR enhanced RISC architecture. By executing powerful instructions in a single clock cycle, the ATmega32 achieves throughputs approaching 1 MIPS per MHz

Atmega32 Features:

- 2 Kilo bytes of internal Static RAM
- 32 X 8bit general working purpose registers
- 32 Kilo bytes of in system self programmable flash program memory.
- 1024 bytes EEPROM
- Programmable serial USART
- 8 Channel, 10 bit ADC
- One 16-bit timer/counter with separate prescaler, compare mode and capture mode.

AVR MICROCONTROLLER Architecture

- Two 8-bit timers/counters with separate prescalers and compare modes
- Available in 40 pin DIP, 44-pad QFN(Quad Flat no lead)/MLF(Micro lead frame) and 44-lead QTFP(thin quad flat package)
- 32 programmable I/O lines
- In system programming by on-chip boot program
- Master/slave SPI serial interface
- 4 PWM channels
- Programmable watch dog timer with separate on-chip oscillator
- a JTAG interface for Boundary scan, On-chip Debugging support and programming
- a serial programmable USART
- A Two Wire Interface(I2C) for serial communication
- six software selectable power saving modes(**Idle mode, Power-down mode, Power-save mode, ADC Noise Reduction mode, Standby mode, Extended Standby mode**)

AVR MICROCONTROLLER Architecture

- The **Idle mode stops the CPU** while allowing the USART, Two-wire interface, A/D Converter, SRAM, Timer/ Counters, SPI port, and interrupt system to continue functioning.
- The **Power-down mode saves the register contents** but freezes the Oscillator, disabling all other chip functions until the next External Interrupt or Hardware Reset.
- In **Power-save mode**, the Asynchronous Timer continues to run allowing the user to maintain a timer base while the rest of the device is sleeping.

Note: *The Asynchronous Timer clock allows the Asynchronous Timer/Counter to be clocked directly from an external 32kHz clock crystal. The dedicated clock domain allows using this Timer/Counter as a real-time counter even when the device is in sleep mode.*

- The **ADC Noise Reduction mode** stops the CPU and all I/O modules except Asynchronous Timer and ADC, to minimize switching noise during ADC conversions.
- In **Standby mode**, the external crystal/resonator clock remains running while the rest of the device is sleeping.
- In **Extended Standby mode**, both the external crystal/resonator clock and the Asynchronous Timer continue to run.

Architecture of AVR

- The architecture of AVR is divided into two parts:

1)**The core**: Computing Engine of the microcontroller.

2)**Peripherals:** No. and type of peripherals vary between different microcontrollers.

- Function of the **core** is to **ensure correct program execution.**
- It has an **8-bit data BUS and the memory is based on Harvard Architecture**(Data memory and program memory and associate BUSES are separate).
- Instructions **in the program memory are executed with single level pipelining** i.e while one instruction is being executed the next instruction is prefetched from the program memory.

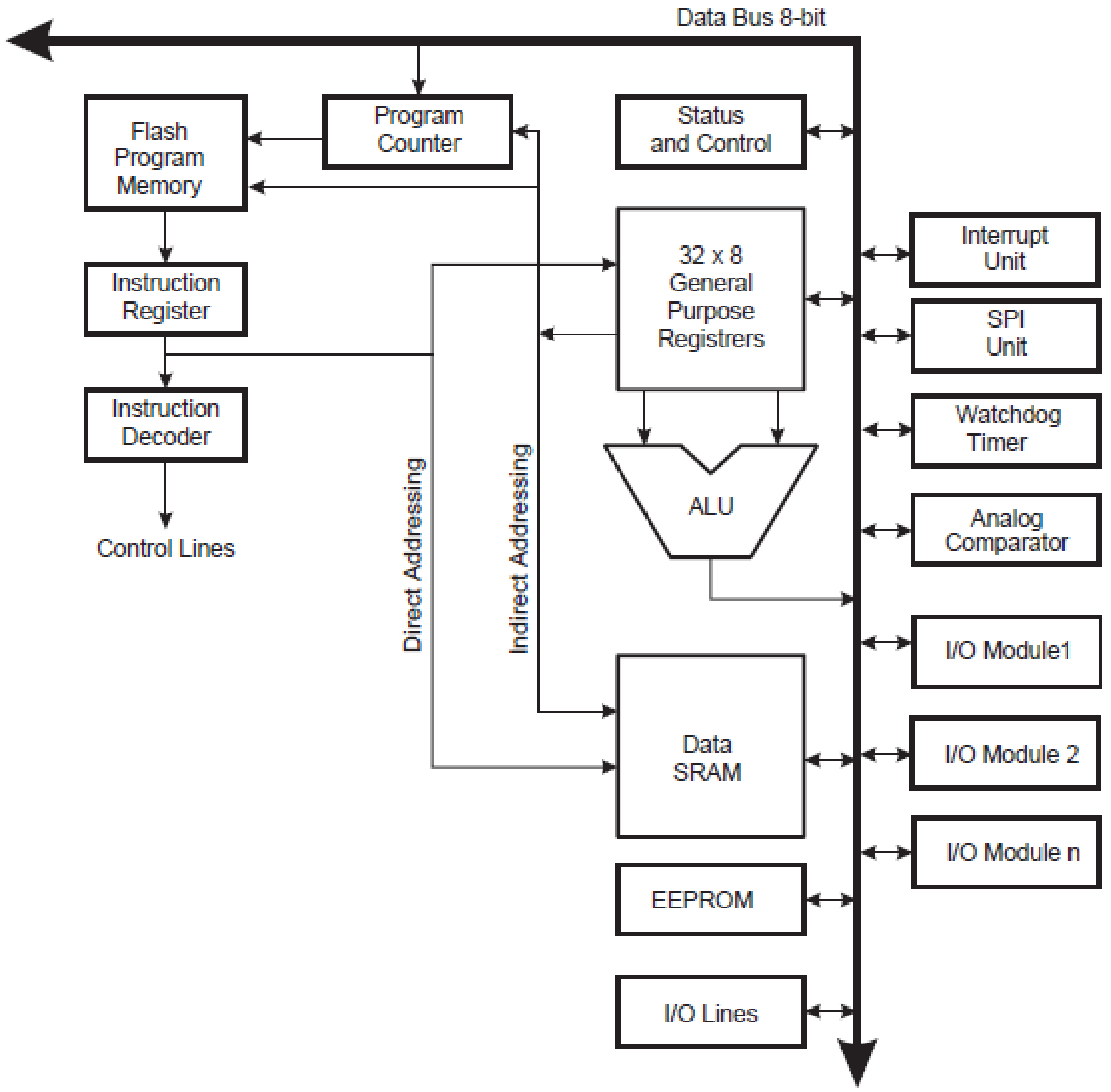
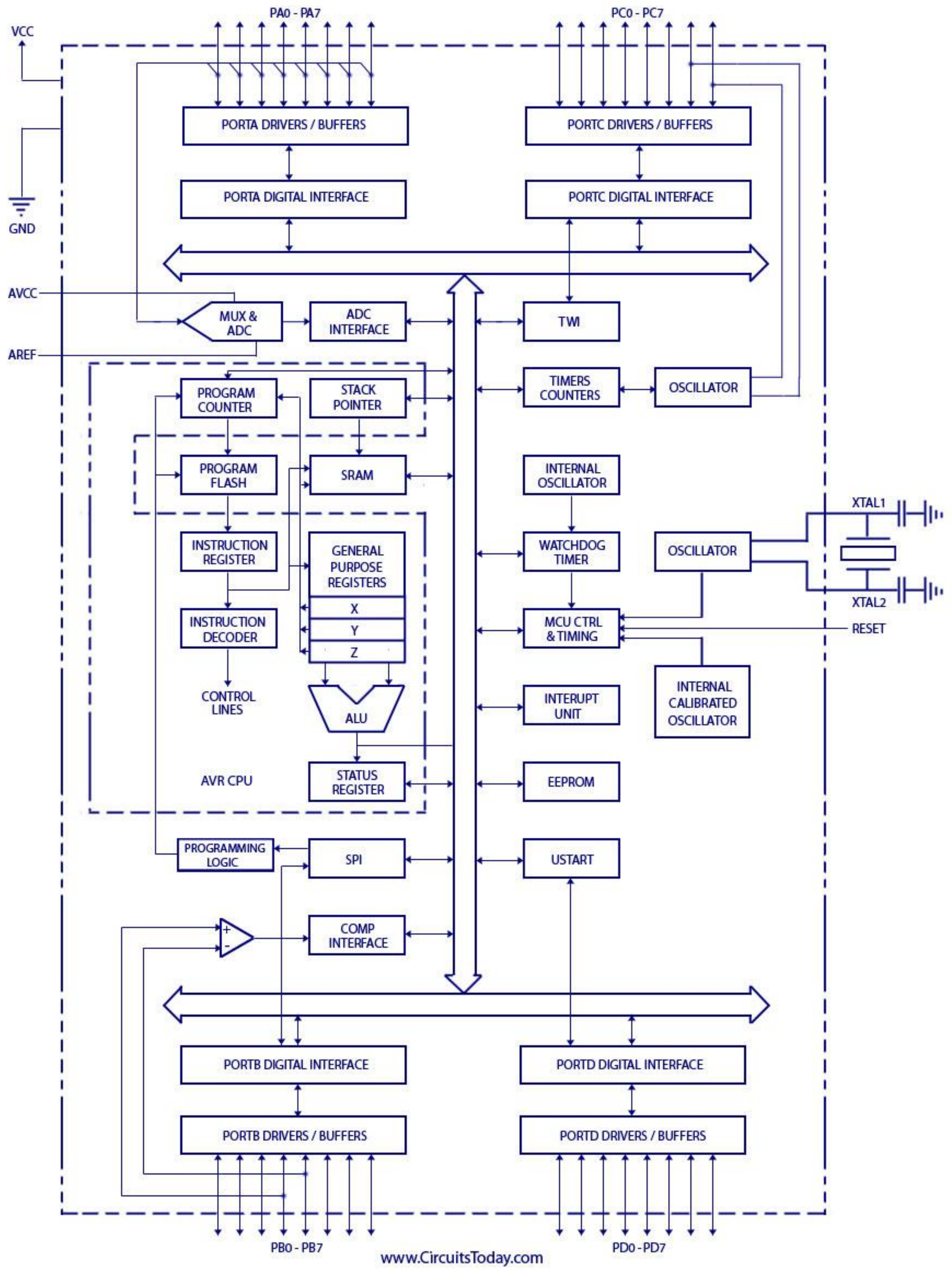


Fig: AVR MCU ARCHITECTURE

Block Diagram - ATmega32(AVR) - 8-bit Microcontroller



www.CircuitsToday.com

Architecture of AVR

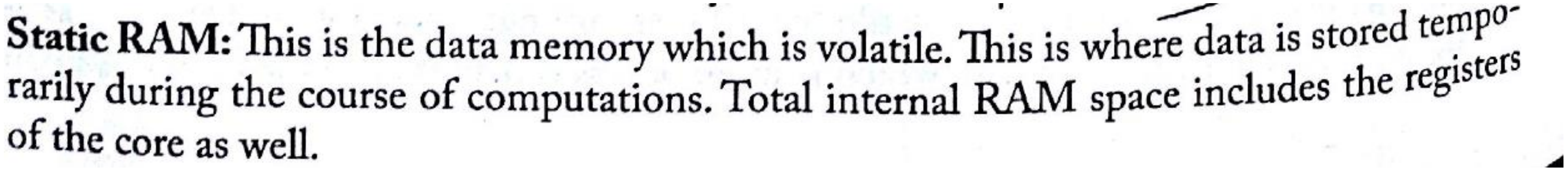
Atmel's ATmega32 is an 8-bit RISC processor, based on Harvard architecture.

On-chip Memory: Atmega32 consist of three different memory sections

1)Flash EEPROM: Flash EEPROM or simple **flash memory is used to store the program dumped or burnt by the user on to the microcontroller**. It can be easily erased in blocks. **Flash memory is non-volatile** i.e., it retains the program even if the power is cut-off. Atmega32 is available with 32KB of in system programmable Flash EEPROM. Program Flash memory space is divided in two sections, the **Boot program section and the Application Program section.** On-chip Boot program running on the AVR core can use any interface to download the application

program in the Application Flash memory.

2) **EEPROM:** This is also a nonvolatile memory used to store data which is more or less a constant once the design is finalized. Atmega32 has 1KB of EEPROM.

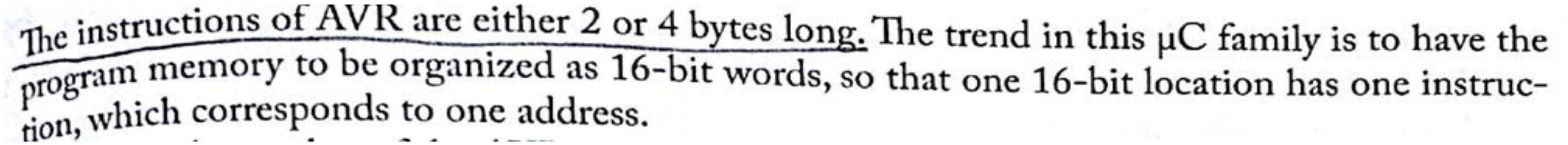
3) 

Static RAM: This is the data memory which is volatile. This is where data is stored temporarily during the course of computations. Total internal RAM space includes the registers of the core as well.

The Size of SRAM in Atmega32 is 2KB.During interrupts and subroutine calls, the return address Program Counter (PC) is stored on the Stack. The Stack is effectively allocated in the general data SRAM

Architecture of AVR

Program Counter:



The instructions of AVR are either 2 or 4 bytes long. The trend in this μC family is to have the program memory to be organized as 16-bit words, so that one 16-bit location has one instruction, which corresponds to one address.

For Each Member of the AVR series ,the bit size of the program counter corresponds to how much of program memory is available. For Atmega32 chip ,there is 32KB of ROM organized as 16K x 16-bits.For address numbering to 16K(2^4 x 2^{10} =2^{14}),14 bits are needed and hence this microcontroller has 14-bit PC.

Register File:

Atmega32 is equipped with 32 general purpose registers of 8-bit size, which are coupled directly with the Arithmetic Logical Unit (ALU) of CPU. These registers are used as operand registers for computation . Of these 32 registers, 6 registers can be used as address registers to act as data address pointers.(Two registers each from X, Y, Z registers)

- The I/O memory space contains 64bytes of space for CPU peripheral functions such as Control Registers, SPI, and other I/O functions. The I/O Memory can be accessed directly, or as the Data Space locations following those of the Register File, \$20 - \$5F.

Note: The registers have names as well as addresses which correspond to the locations in RAM space.

Architecture of AVR

- When using the I/O specific commands IN and OUT, the I/O addresses $00 - $3F must be used. When addressing I/O Registers as data space using LD and ST instructions, $20 must be added to these addresses

Architecture of AVR

- **Status and Control Register:** After an arithmetic operation, the Status Register is updated to reflect information about the result of the operation. The Status Register contains information about the result of the most recently executed arithmetic instruction. This information can be used for altering program flow in order to perform conditional operations.

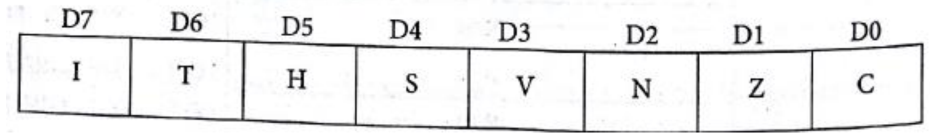


Fig: Status register

Architecture of AVR

I/O Ports: Atmega32 has four (PORTA, PORTB, PORTC and PORTD) **8-bit** input-output ports. In Atmega32, I/O operations are controlled through three types of registers: DDRX (Data Direction Register), PORTX (Port Output Register), and PINX (Port Input Register).

Internal Calibrated Oscillator: Atmega32 is equipped with an internal oscillator for driving its clock. By default **Atmega32 is set to operate at 1 MHz** . The maximum frequency of internal oscillator is 8MHz. Alternatively, ATmega32 can be operated using an **external crystal oscillator with a maximum frequency of 16MHz.**

ADC Interface: Atmega32 is equipped with **an 8 channel ADC (Analog to Digital Converter) with a resolution of 10-bits.**

Timers/Counters: Atmega32 consists of two 8-bit and one 16-bit timer/counter. Timers are useful for generating time delays between two operations.(T0 and T2:8-bit Timers, T1:16-bit timer)

- **Two 8-bit Timer/Counters with Separate Prescalers and Compare Modes**
- **16-bit Timer/Counter with Separate Prescaler, Compare Mode, and Capture Mode**

Watchdog Timer: The Watchdog Timer is clocked from a separate On-chip Oscillator which runs at 1MHz.Watchdog timer **continuously monitors and resets the controller if the code gets stuck at any execution point for more than a defined time interval**.

Architecture of AVR

Interrupts: Atmega32 consists of 21 interrupt sources out of which three are external(INT0,INT1,INT2).Upon activation of these interrupts the Atmega32 controller gets interrupted in whatever task it is doing and jumps to perform **Interrupt Service Routine**. The remaining are internal interrupts which support various operations in the microcontroller like USART, ADC, Timers etc.

USART: Universal Synchronous and Asynchronous Receiver and Transmitter interface is available for interfacing with external device capable of communicating serially by using TXD and RXD pins.

SPI: Serial Peripheral Interface, SPI port is used for serial communication between two devices on a common clock source. The data transmission rate of SPI is more than that of USART.

TWI(I2C): Two Wire Interface (TWI) can be **used to set up a network of devices**, many devices can be connected over TWI interface forming a network, the devices can simultaneously transmit and receive and have their own unique address.

MCU Timing and Control: It is used for the generation of timing and control signals.

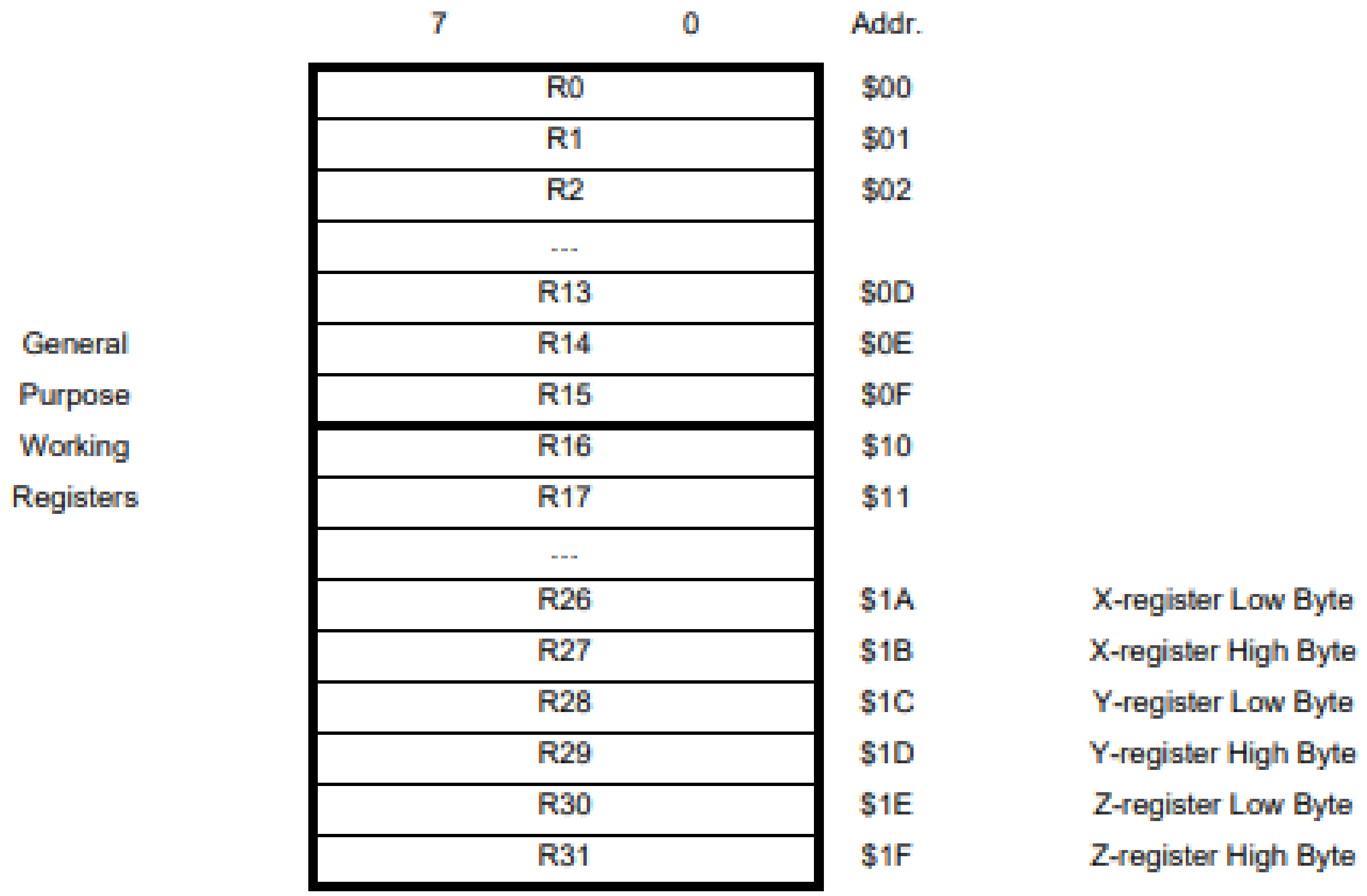
Architecture of AVR

- The **Analog Comparator** compares the input values on the positive pin AIN0 and negative pin AIN1. When the voltage on the positive pin AIN0 is higher than the voltage on the negative pin AIN1, the **Analog Comparator Output, ACO, is set**. The comparator's output can be set to trigger the Timer/Counter1 Input Capture function. In addition, the comparator can trigger a separate interrupt, exclusive to the Analog Comparator.

Register File

- The register file is **optimized for the AVR enhanced RISC instruction set**. Most of the instructions that operate on the register file have direct access to all the registers.
- Most of these instructions are just single instruction that run in one clock cycle . Since there are **32 of these registers combined with direct accessibility by the ALU,** programmers can have more registers for storage of values for faster processing.
- Figure in the next slide shows the structure of the 32 general purpose working registers in the CPU:

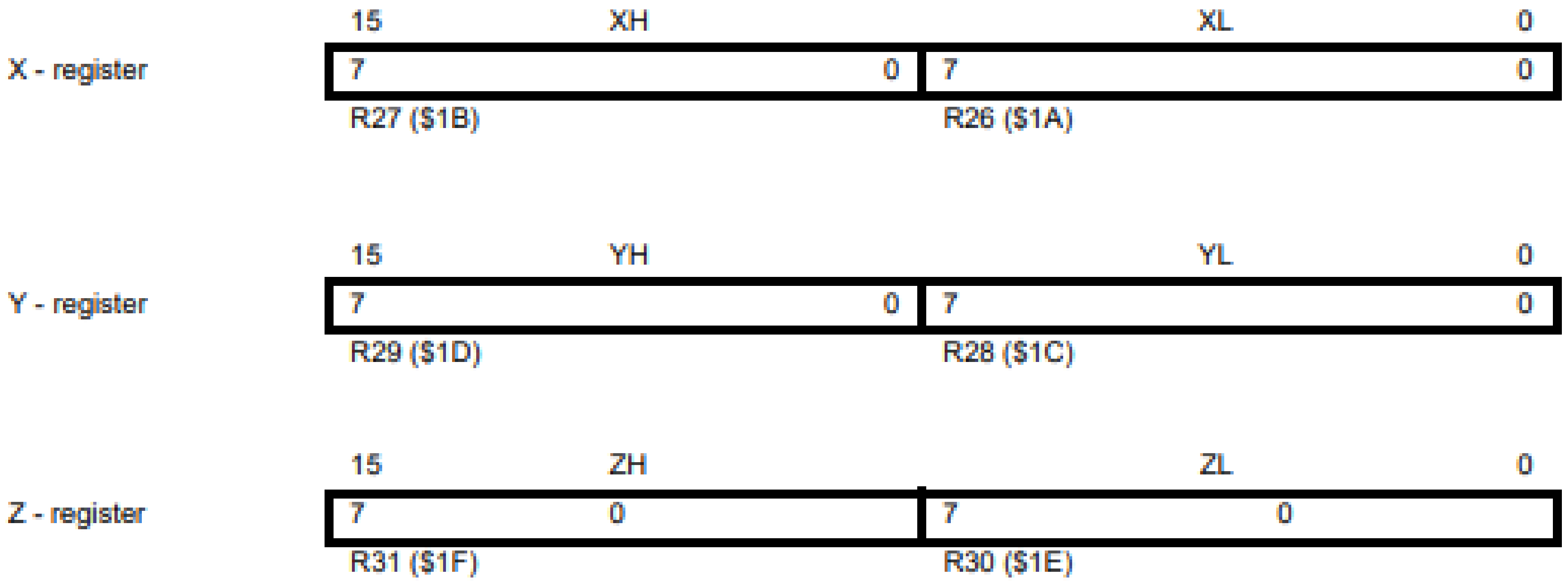
Register File



	7 0	Addr.	
	R0	$00	
	R1	$01	
	R2	$02	
	...		
	R13	$0D	
General	R14	$0E	
Purpose	R15	$0F	
Working	R16	$10	
Registers	R17	$11	
	...		
	R26	$1A	X-register Low Byte
	R27	$1B	X-register High Byte
	R28	$1C	Y-register Low Byte
	R29	$1D	Y-register High Byte
	R30	$1E	Z-register Low Byte
	R31	$1F	Z-register High Byte

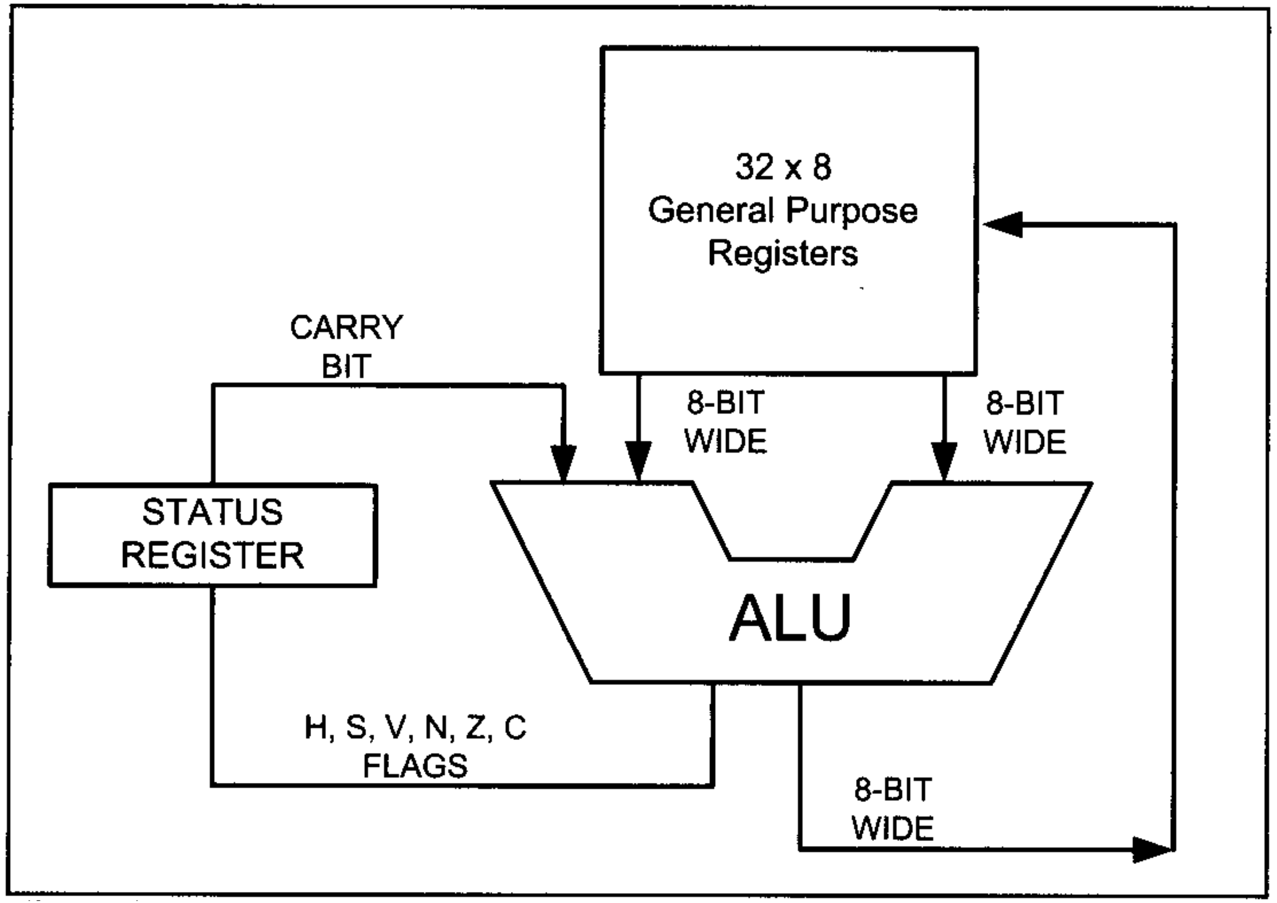
Register File

- Each register is also assigned a data memory address, mapping them directly into the first 32 locations of the user Data Space , although not being physically implemented as SRAM locations.
- The **registers R26..R31 have some added functions to their general purpose usage**.
- These registers are **16-bit address pointers for indirect addressing of the Data Space**. The three indirect address registers X, Y, and Z are defined as described in figure in the next slide:
- The registers have names as well as addresses which correspond to their location in RAM space.
- The registers R27 and R26 are concatenated to form the X-register ,R29 and R28 to form the Y register ,R31 and R30 to form the Z Register



X - register
15
XH
XL
0
7
0
7
0
R27 ($1B)
R26 ($1A)
Y - register
15
YH
YL
0
7
0
7
0
R29 ($1D)
R28 ($1C)
Z - register
15
ZH
ZL
0
7
0
7
0
R31 ($1F)
R30 ($1E)

AVR general purpose Registers



Register File

- **Status and Control Register:** After an arithmetic operation, **the Status Register is updated to reflect information about the result** of the operation. The Status Register contains information about the result of the most recently executed arithmetic instruction. This information can be used for altering program flow in order to perform conditional operations.

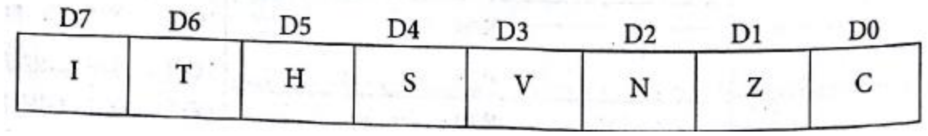


Fig: Status register

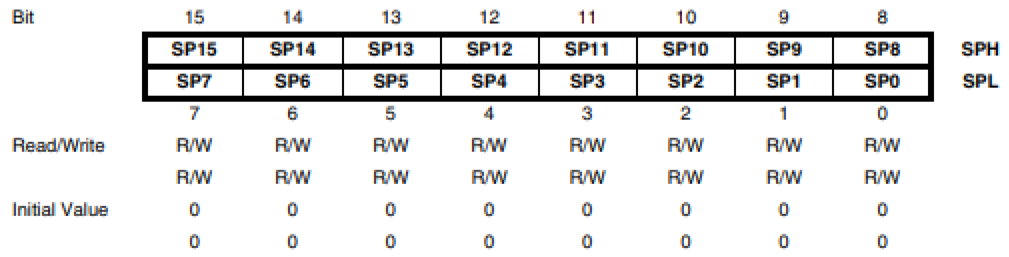
- **Bit 7 – I: Global Interrupt Enable:**The Global Interrupt Enable bit must be set for the interrupts to be enabled.
- **Bit 6 – T: Bit Copy Storage**
- The Bit Copy instructions BLD (Bit LoaD) and BST (Bit STore) use the T-bit as source or destination for the operated bit. A bit from a register in the Register File can be copied into T by the BST instruction, and a bit in T can be copied into a bit in a register in the Register File by the BLD instruction.

Register File

- **Bit 5 – H: Half Carry Flag(Auxiliary Carry):** The Half Carry Flag H indicates a Half Carry in some arithmetic operations. This bit is set to 1 , when a carry is generated after four bits.
- **Bit 4 – S: Sign Bit, S = N $\oplus$ V :**The S-bit is always an exclusive or between the Negative Flag N and the Two's Complement Overflow Flag V.
- **Bit 3 – V: Two's Complement Overflow Flag:** The Two's Complement Overflow Flag V supports two's complement arithmetic. Overflow flag indicates that result is too large to fit in the 8-bit destination operand.
- **Bit 2 – N: Negative Flag:**The Negative Flag N indicates a negative result in an arithmetic or logic operation
- **Bit 1 – Z: Zero Flag:**The Zero Flag 'Z' indicates a zero result in an arithmetic or logic operation
- **Bit 0 – C: Carry Flag:** The Carry Flag C indicates a carry in an arithmetic or logic operation

Register File

- **Stack Pointer:**The Stack is mainly used for storing temporary data, for storing local variables and for storing return addresses after interrupts and subroutine calls. The Stack Pointer Register always points to the top of the Stack. **The Stack Pointer must be set to point above $60.**
- The Stack Pointer is decremented by one when data is pushed onto the Stack with the PUSH instruction, and it is decremented by two when the return address is pushed onto the Stack with subroutine call or interrupt.
- The Stack Pointer is incremented by one when data is popped from the Stack with the POP instruction, and it is incremented by two when data is popped from the Stack with return from subroutine RET or return from interrupt RETI. The AVR Stack Pointer is implemented as two 8-bit registers in the I/O space.
- The number of bits actually used is implementation dependent. Note that the data space in some implementations of the AVR architecture is so small that only SPL is needed. In this case, the SPH Register will not be present.



Bit	15	14	13	12	11	10	9	8	
	SP15	SP14	SP13	SP12	SP11	SP10	SP9	SP8	SPH
	SP7	SP6	SP5	SP4	SP3	SP2	SP1	SP0	SPL
	7	6	5	4	3	2	1	0	
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	
	0	0	0	0	0	0	0	0	

To push a register onto a stack, we use PUSH instruction.
PUSH Rr;
Rr can be any general-purpose register (R0 - R31)

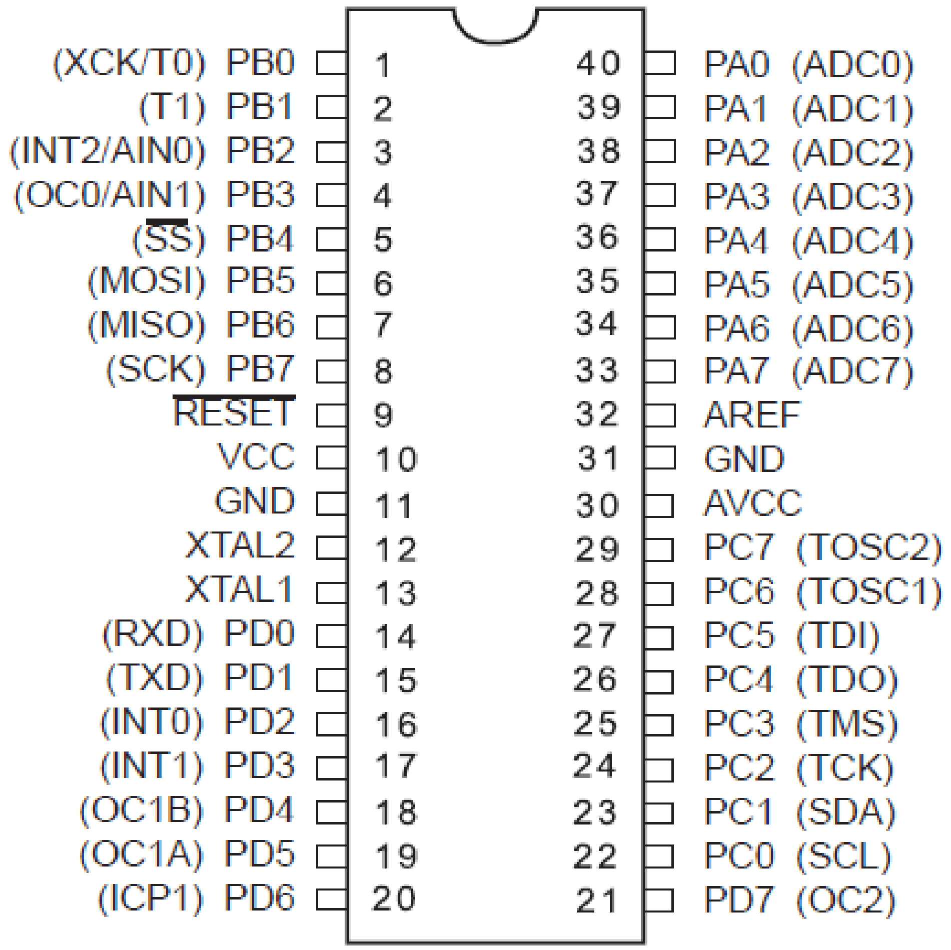
Register File

To retrieve back the data from the stack, we use POP instruction.
POP Rr;
Rr can be any general-purpose register (R0 - R31)

PC(Program Counter):Program Counter contains the address of the next instruction to be executed in the flash program memory. For Atmega32 chip ,there is 32KB of ROM organized as 16K x 16-bits.For address numbering to 16K(2^4 x 2^{10} =2^{14}),14 bits are needed and hence this microcontroller has 14-bit PC.

IR(Instruction Register): The **instruction register (IR) is the part of a CPU's control unit that holds the instruction currently being decoded**. When the CPU fetches an instruction from memory, it is stored in the IR, and the Control Unit decodes the instruction and executes it. The IR is an important part of the CPU's pipeline, which is a technique used to increase the CPU's processing speed by overlapping the execution of multiple instructions. By fetching the next instruction while the current instruction is being executed, the CPU can reduce the amount of time spent waiting for instructions to be fetched, which can improve its overall performance.

PIN DIAGRAM OF AVR(Atmega32)

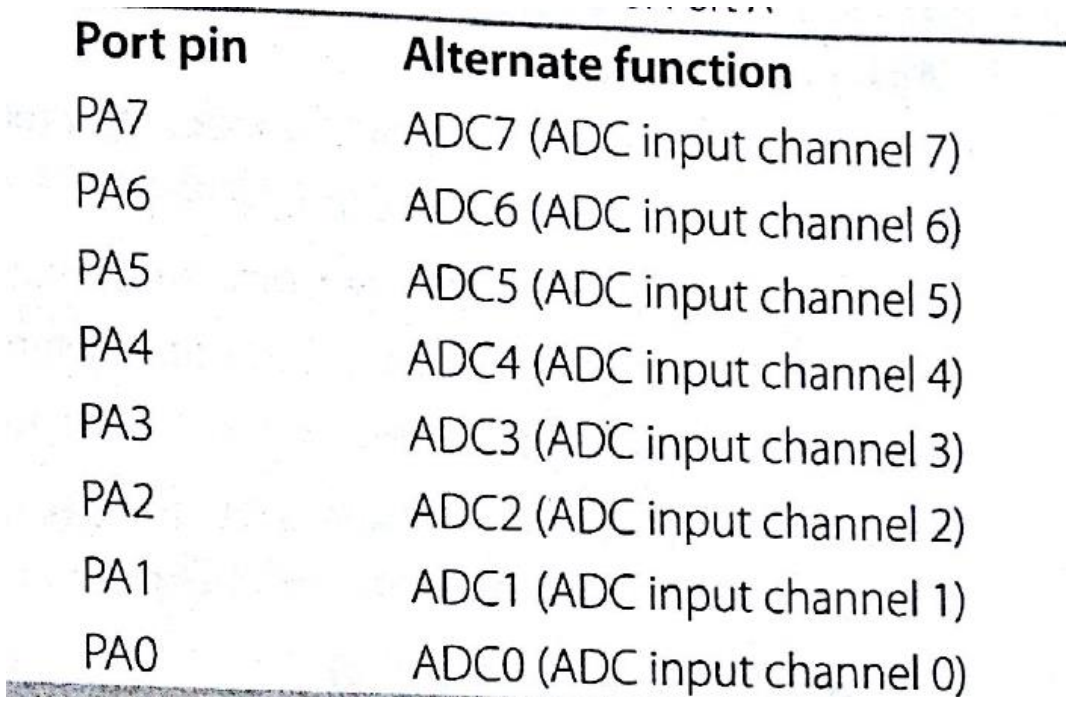


PIN DESCRIPTION OF AVR

Atmega32 is a 40-pin IC with 4 **8-bit ports named as PA,PB,PC,PD constituting 32 GPIO lines.** All port pins have dual functions.

ATMega32 gives four power pins, power input for digital pins and analog power input for ADC and the remaining two are ground pins.

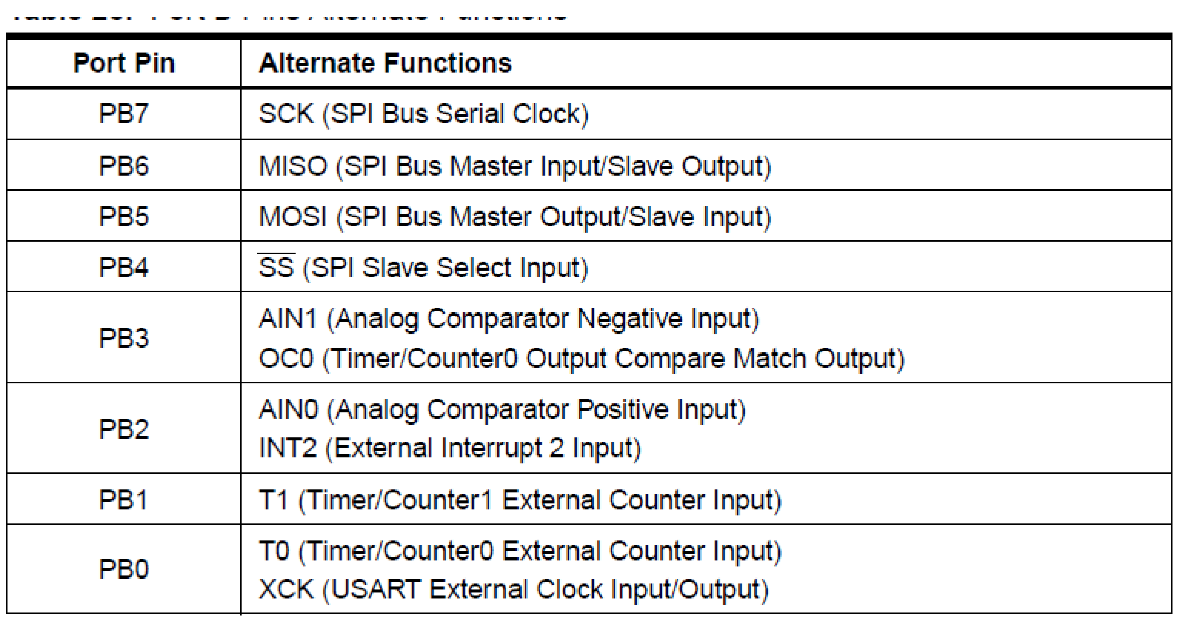
- **VCC:** Digital supply voltage (PIN-10)
- **AVCC(PIN-30):** It should be externally connected to VCC, even if the ADC is not used. If the ADC is used, it should be connected to VCC through a low-pass filter.
- **GND**(PIN11,PIN31)**:** Ground
- **Port A (PA7..PA0):** PIN 33 to 40 come under PORT A. Port A serves as the analog inputs to the A/D Converter. Port A also serves as an 8-bit bi-directional I/O port, if the A/D Converter is not used. Port pins are provided with internal pull-up resistors.



Port pin	Alternate function
PA7	ADC7 (ADC input channel 7)
PA6	ADC6 (ADC input channel 6)
PA5	ADC5 (ADC input channel 5)
PA4	ADC4 (ADC input channel 4)
PA3	ADC3 (ADC input channel 3)
PA2	ADC2 (ADC input channel 2)
PA1	ADC1 (ADC input channel 1)
PA0	ADC0 (ADC input channel 0)

PIN DESCRIPTION OF AVR

- **Port B (PB7..PB0) :** Port B is an 8-bit bi-directional I/O port with internal pull-up resistors. Port B also serves the functions of various special features of the ATmega16



[illegible]

Port Pin	Alternate Functions
PB7	SCK (SPI Bus Serial Clock)
PB6	MISO (SPI Bus Master Input/Slave Output)
PB5	MOSI (SPI Bus Master Output/Slave Input)
PB4	$\overline{\text{SS}}$ (SPI Slave Select Input)
PB3	AIN1 (Analog Comparator Negative Input) OC0 (Timer/Counter0 Output Compare Match Output)
PB2	AIN0 (Analog Comparator Positive Input) INT2 (External Interrupt 2 Input)
PB1	T1 (Timer/Counter1 External Counter Input)
PB0	T0 (Timer/Counter0 External Counter Input) XCK (USART External Clock Input/Output)

PIN DESCRIPTION OF AVR

(PB4-PB7):

SS’ – GPIO5

MOSI – GPIO6

MISO – GPIO7

SCK – GPIO8

SPI: SPI protocol is one of the best serial communication protocols for communication with multiple modules. It can be used in the case when multiple peripherals have to communicate efficiently with the microcontroller. The **communication wires consist of two wires for data to support full duplex communication(MOSI ,MISO) and one for the clock(SCK)**, and also a wire which is used for device selection. The device selection pin is knowing as Select Slave pin and it is predefined in the micro controller but **any output pin can be made as a select slave pin(SS’) by programming**. SPI protocol is not only used for communication, but it can also be used to program the microcontroller.

PIN DESCRIPTION OF AVR

(PB3-PB2),PD2,PD3:

Analog Comparator: To compare two analog signals an analog comparator is used in the microcontroller.

AN0(Analog Comparator positive input) – PB2

AN1 (Analog Comparator negative input) – PB3

PB3(Timer/Counter0 Output Compare Match Output): The Output Compare Register of timer/counter0 contains an 8-bit value that is continuously compared with the counter value (TCNT0) to generate **an output signal on OC0 pin**. The pin has to be configured as an output (DDB3 set (one)) to serve this function. The **OC0 pin is the output pin for the PWM mode timer function**.

- **INT2(PB2):**External Interrupt 2
- **INT0(PD2) –** GPIO16
- **INT1(PD3) –** GPIO17

In this microcontroller, there are total three interrupt pins which can be used by external modules or by an external button to get the attention of CPU.

PIN DESCRIPTION OF AVR

PB1-PB0:

T0/XCK – Port B, Bit 0

XCK(USART EXTERNAL CLOCK): This pin acts as the clock pin when USART operates in synchronous mode. The Data Direction Register (DDB0) controls whether the clock is output (DDB0 set) or input (DDB0 cleared).

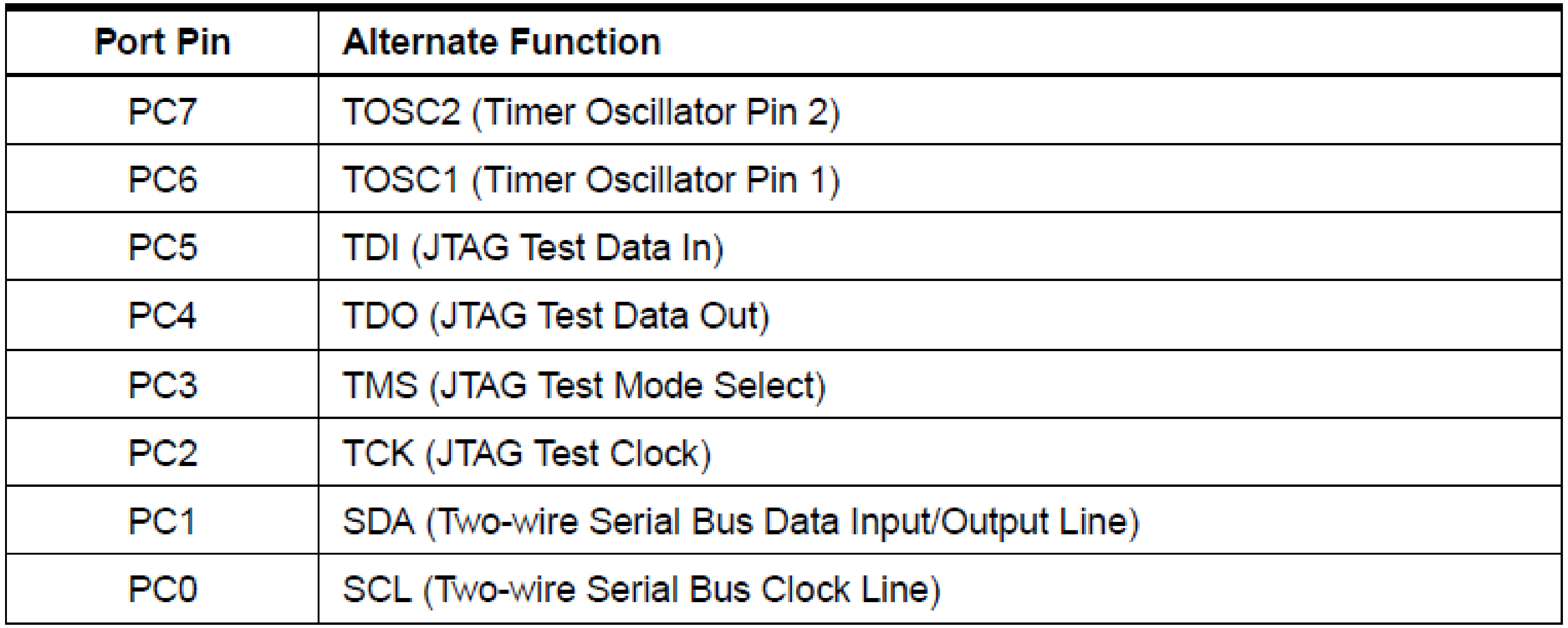
T0(T0, Timer/Counter0 Counter Source**)**

T1 – Port B, Bit 1(T1, Timer/Counter1 Counter Source)

The frequency of external pulses connected to T0 and T1 pin are counted . Timer/counter 0 and Timer/Counter 1 should be configured as a counter to count the external clock pulses.

PIN DESCRIPTION OF AVR

- **Port C (PC7..PC0):Pins 22-29**



Port Pin	Alternate Function
PC7	TOSC2 (Timer Oscillator Pin 2)
PC6	TOSC1 (Timer Oscillator Pin 1)
PC5	TDI (JTAG Test Data In)
PC4	TDO (JTAG Test Data Out)
PC3	TMS (JTAG Test Mode Select)
PC2	TCK (JTAG Test Clock)
PC1	SDA (Two-wire Serial Bus Data Input/Output Line)
PC0	SCL (Two-wire Serial Bus Clock Line)

PIN DESCRIPTION OF AVR

PC7: TOSC2(TOSC2, Timer Oscillator pin 2 **)**

PC6: TOSC1(Timer Oscillator pin 1)

- Timer2 can be operated in the asynchronous mode by connecting an external crystal oscillator betweenTOSC1 AND TOSC2 pins.(***When the AS2 bit in the ASSR Register is written to logic one, the clock source is taken from the Timer/Counter Oscillator connected to TOSC1 and TOSC2***.)

(PC2-PC5):

These pins are present in most of the boards **for testing purposes .** JTAG could be connected to the internal test port , it can also be used for programming the microcontroller and even the bootloader . The JTAG interface is a standard for **testing and debugging electronic devices and systems** and it uses a serial communication protocol to access and control the devices internal registers. JTAG pins in the microcontroller are:

TDI(test data in)-PC5:used to input test data into the device during a JTAG operation

TDO(test data out)- PC4:used to ouput test data from the device during JTAG operation.

TMS(test mode selects)– PC3:shifts the JTAG state machine from one state to the next.

TCK (JTAG Test Clock)– PC2:Used to provide clock signal for JTAG operations. Clock is generated by an external test equipment such as a JTAG debugger.

PIN DESCRIPTION OF AVR

PC1 - SDA

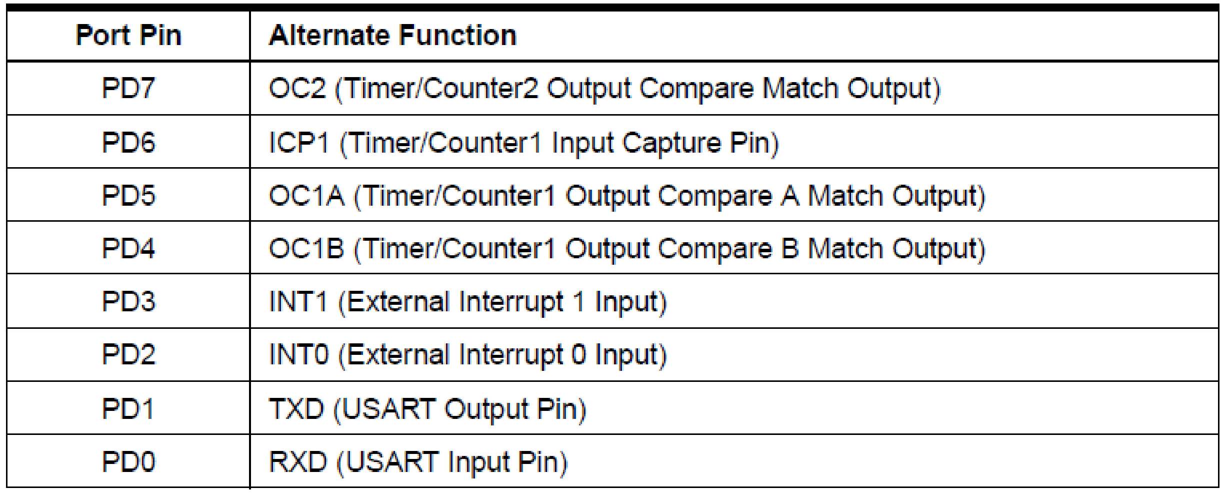
PC0 - SCL

I2C: Some sensors and servos come with serial communication protocol called I2C(Inter Integrated Circuit). To communicate with those peripherals ATMega32 also gives I2C pins interface. One pin is used for data communication and one for a clock. Both pins are listed below:

- SDA(Serial Data) – GPIO23
- SCL(Serial Clock) – GPIO22

PIN DESCRIPTION OF AVR

- **Port D (PD7..PD0)** Port D is an 8-bit bi-directional I/O port with internal pull-up resistors (selected for each bit). Port D also serves the functions of various special features of the ATmega32



Port Pin	Alternate Function
PD7	OC2 (Timer/Counter2 Output Compare Match Output)
PD6	ICP1 (Timer/Counter1 Input Capture Pin)
PD5	OC1A (Timer/Counter1 Output Compare A Match Output)
PD4	OC1B (Timer/Counter1 Output Compare B Match Output)
PD3	INT1 (External Interrupt 1 Input)
PD2	INT0 (External Interrupt 0 Input)
PD1	TXD (USART Output Pin)
PD0	RXD (USART Input Pin)

PIN DESCRIPTION OF AVR

PD7-OC2 (OC2, Timer/Counter2 Output Compare Match output): The Output Compare Register of Timer/counter2 contains an 8-bit value that is continuously compared with the counter value (TCNT2). A **match can be used to generate an output compare interrupt**, or to **generate a waveform output on the OC2 pin**. The pin has to be configured as an output (DDD7 set (one)) to serve this function. The OC2 pin is the output pin for the PWM mode timer function.

PD6-ICP1 (Input Capture Pin): The Timer/Counter value at a given external (edge triggered) event can be captured on the Input Capture Pin.

OC1A – PD5

OC1A, Output Compare Match A output: The PD5 pin can serve as an external output for the Timer/Counter1 Output Compare A. The pin has to be configured as an output (DDD5 set (one)) to serve this function. The OC1A pin is the output pin for the PWM mode timer function.

OC1B – PD4

OC1B, Output Compare Match B output: The PD4 pin can serve as an external output for the Timer/Counter1 Output Compare B. The pin has to be configured as an output (DDD4 set (one)) to serve this function. The OC1B pin is the output pin for the PWM mode timer function.

PIN DESCRIPTION OF AVR

- **TXD – PD1**

TXD, Transmit Data (Data output pin for the USART). When the USART Transmitter is enabled, this pin is configured as an output regardless of the value of DDD1.

- **RXD – PD0**

RXD, Receive Data (Data input pin for the USART). When the USART Receiver is enabled this pin is configured as an input regardless of the value of DDD0.

PIN DESCRIPTION OF AVR

- **RESET(Pin-9)** Reset Input. A low level on this pin for longer than the minimum pulse length will generate a reset.
- **External Oscillator:** ATMega32 has internal oscillator which can be configured up to 8MHz but to extend the clock speed till 16MHz , an external oscillator will be used at the oscillator pins of the microcontroller which are given below:

  XTAL2 – GPIO12

  XTAL1 – GPIO13
- **AREF:** AREF is the analog reference pin for the A/D Converter.(The voltage of the AREF should be in between 0V and AVcc)

Memory Organization

- In Atmega32 there are two kinds of Memory space:

➢ Code Memory(where program is stored)

➢ Data Memory(where data is stored)

- The **Data Memory** space has three components:

➢ *General Purpose Registers*

➢ *I/O Memory*

➢ *Internal SRAM*

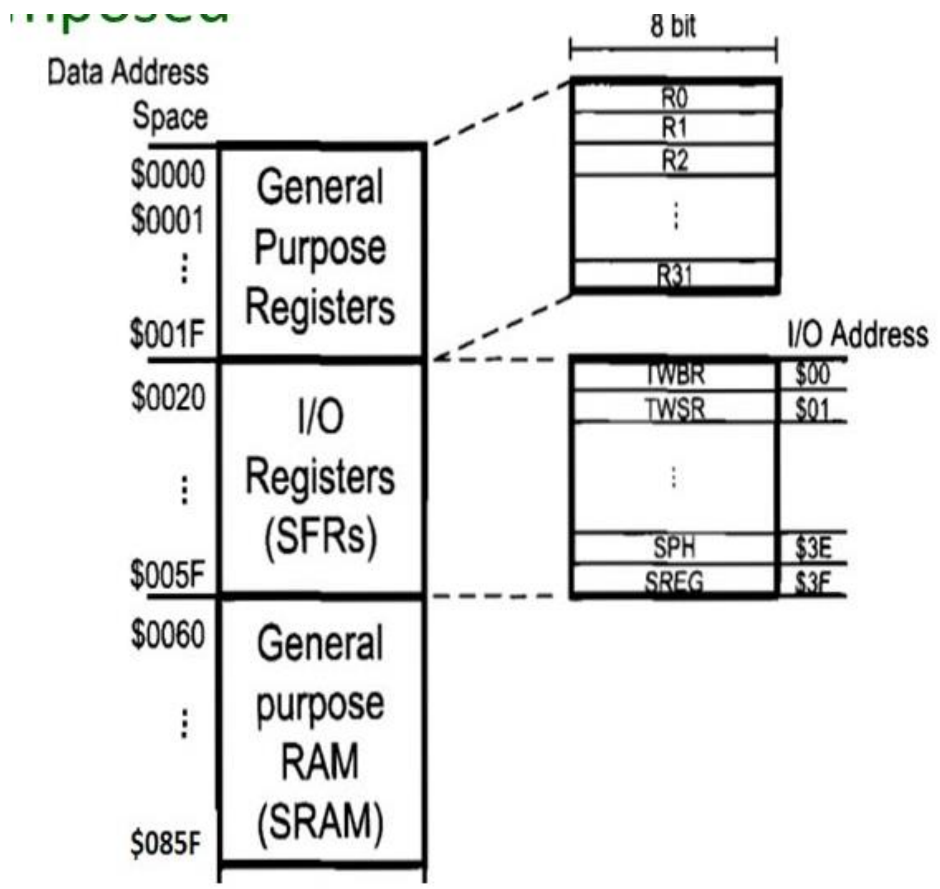
The General Purpose Registers use 32 bytes of the Data Memory space. They take the address location $00 to $1F in the data memory space.

Memory Organization

- **I/O Memory:** The I/O memory **is dedicated to specific functions** such as status register, timers, serial communication ,I/O ports, ADC and so on .
- The function of each of the I/O memory location is fixed at the time of design because it is used for the control of the microcontroller or peripherals . **The AVR I/O memory is made of 8-bit registers** .
- The Atmega32 microcontroller has **64bytes of I/O memory** locations . The 64-byte section is called the **standard I/O Memory** . The **I/O registers are also called as special function registers since each one is dedicated to a special function**.

Memory Organization

- When using the I/O specific commands IN and OUT, the I/O addresses $00 - $3F must be used. When addressing I/O Registers as data space using LD and ST instructions, $20 must be added to these addresses.



Memory Organization

- **Internal Data SRAM:** Size of SRAM in Atmega32 is 2KB.It is widely used for storing data and parameters by AVR programmers and C compilers. It is also called as scratch pad.
- Each location in the SRAM can be directly accessed by its address. Each location is 8-bits wide.
- The three parts of the memory (**GPR'S, SFR'S and Data SRAM**) are made of SRAM.
- **EEPROM:**Atmega32 has an EEPROM of size 1KB. EEPROM is used for storing data that should be rarely changed and should not be lost when the power is off.

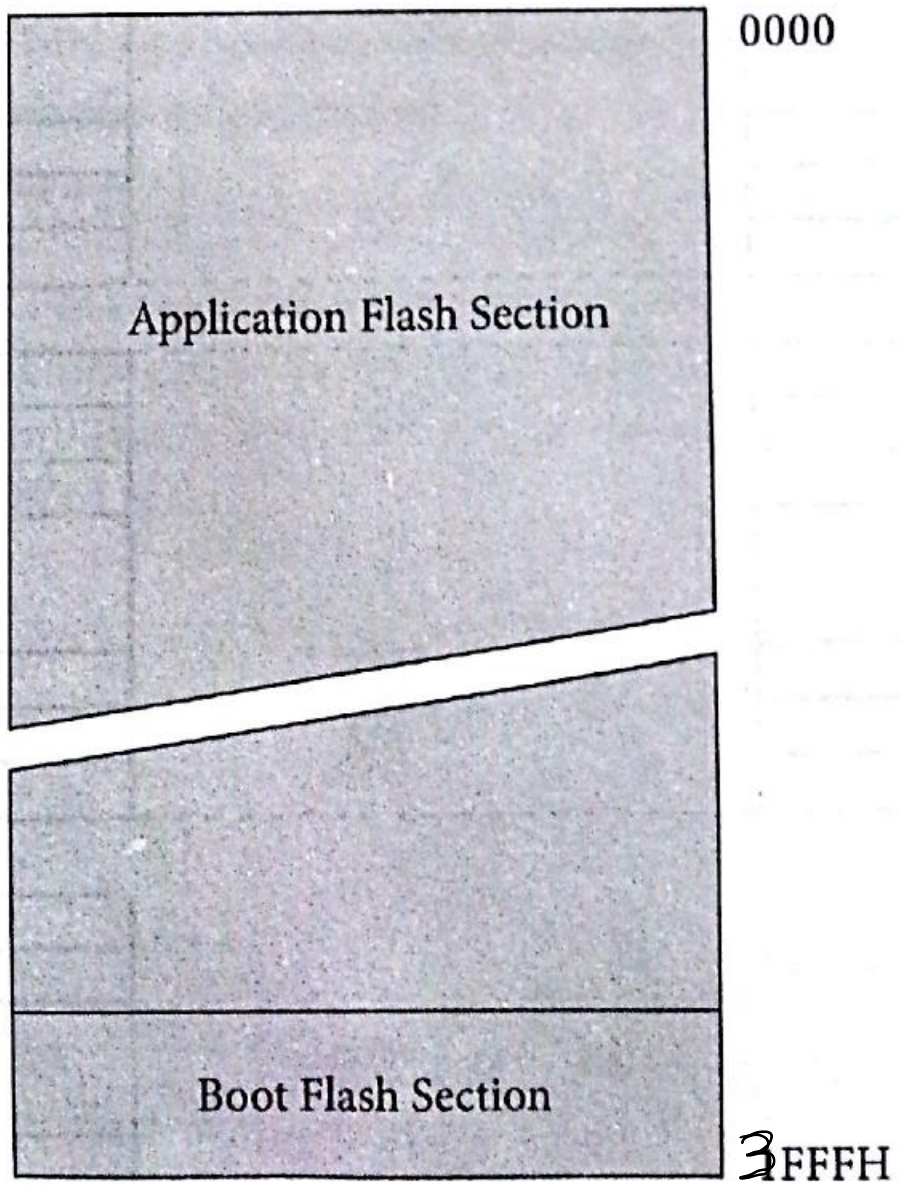
Memory Organization

In AVR microcontrollers each Flash memory location is 2 bytes wide. For example, in ATmega32, whose Flash is 32K bytes, the Flash is organized as 16K× 16, and its program counter is 14 bits wide (2^{14} = 16K memory locations).

The program memory of any AVR microcontroller is word addressable.
In case of a 14-bit program counter the code space is 16K(2^{14}) which occupies the 0000-$3FFF address range.

For software security, the flash program memory space is divided into two sections, the boot program section and the application program section. The user is to use the application section while the start-up code is in the higher area of memory. On reset, the first instruction is taken from the res$\overset{e}{\wedge}$t vector which is 0000. Therein, a jump instruction takes control to the boot sec--tion where the startup code is written.

After the program is burned into ROM of an AVR family member such as ATmega32 or ATtiny11, the opcode and operand are placed in ROM memory loca--tions starting at 0000.



The ROM address space of ATMega32

I/O Ports

- Atmega32 has 4 ports each having 8 pins. PortA, PortB, PortC and PortD.
- To use any of these ports as an input or output port , it must be programmed .
- In addition to being used as a simple I/O port each port has some other functions such as ADC , timers, interrupts and serial communication pins.

I/O Ports

Each port has three I/O registers associated with it, as shown in Table 4-2. They are designated as PORTx, DDRx, and PINx. For example, for Port B we have PORTB, DDRB, and PINB. Notice that DDR stands for Data Direction Register, and PIN stands for Port INput pins. Also notice that each of the I/O registers is 8 bits wide, and each port has a maximum of 8 pins; therefore each bit of the I/O registers affects one of the pins (see Figure 4-2; the content of bit 0 of DDRB represents the direction of PB0 pin and so on.

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

I/O Ports

DDRX Register:

The DDRx I/O register is used solely for the purpose of making a given port an input or output port. For example, to make a port an output, we write 1s to the DDRx register. In other words, to output data to all of the pins of the Port B, we must first put 0b11111111 into the DDRB register to make all of the pins output.

The following code will toggle all 8 bits of Port B forever with some time delay between "on" and "off" states:

```
        LDI    R16,0xFF     ;R16 = 0xFF = 0b11111111
        OUT    DDRB,R16     ;make Port B an output port (1111 1111)
L1:     LDI    R16,0x55     ;R16 = 0x55 = 0b01010101
        OUT    PORTB,R16    ;put 0x55 on port B pins
        CALL   DELAY
        LDI    R16,0xAA     ;R16 = 0xAA = 0b10101010
        OUT    PORTB,R16    ;put 0xAA on port B pins
        CALL   DELAY
        RJMP   L1
```

I/O Ports

It must be noted that unless we set the DDRx bits to one, the data will not go from the port register to the pins of the AVR. This means that if we remove the first two lines of the above code, the 0x55 and 0xAA values will not get to the pins.

To make a port an input port, we must first put 0s into the DDRx register for that port and then bring in the data present at the pins. When DDR contains 0'S the port gets data.

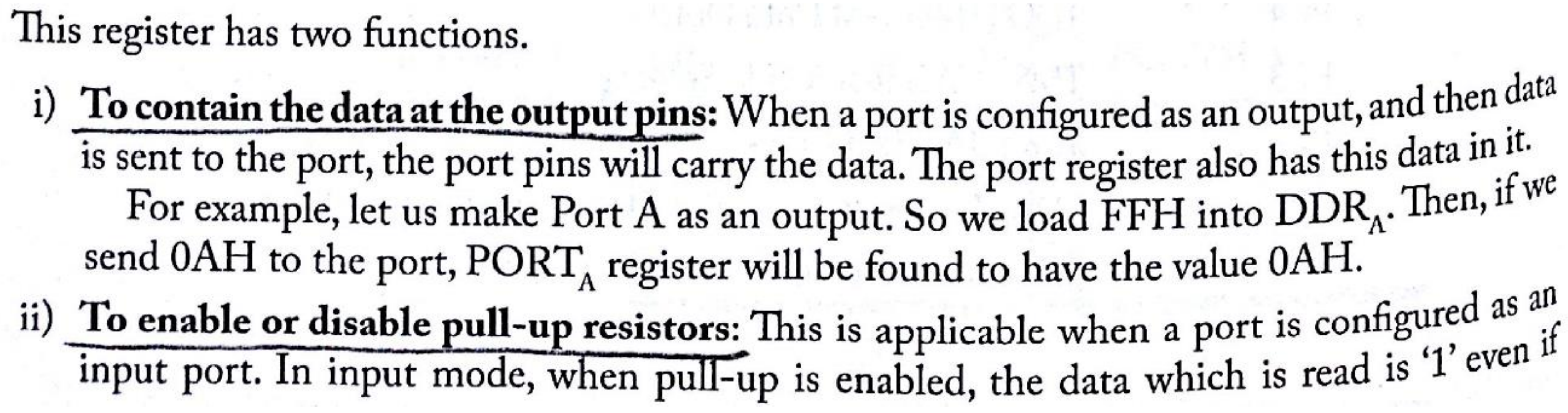
- Upon reset ,all ports have the value 0x00 in their DDR registers . This means all ports are configured as inputs.

I/O Ports

- **PINx and PORTx:**

To read the data present at the pins, we should read the PIN register. It must be noted that to bring data into CPU from pins we read the contents of the PINx register, whereas to send data out to pins we use the PORTx register.

- **PORTX register dual role:**

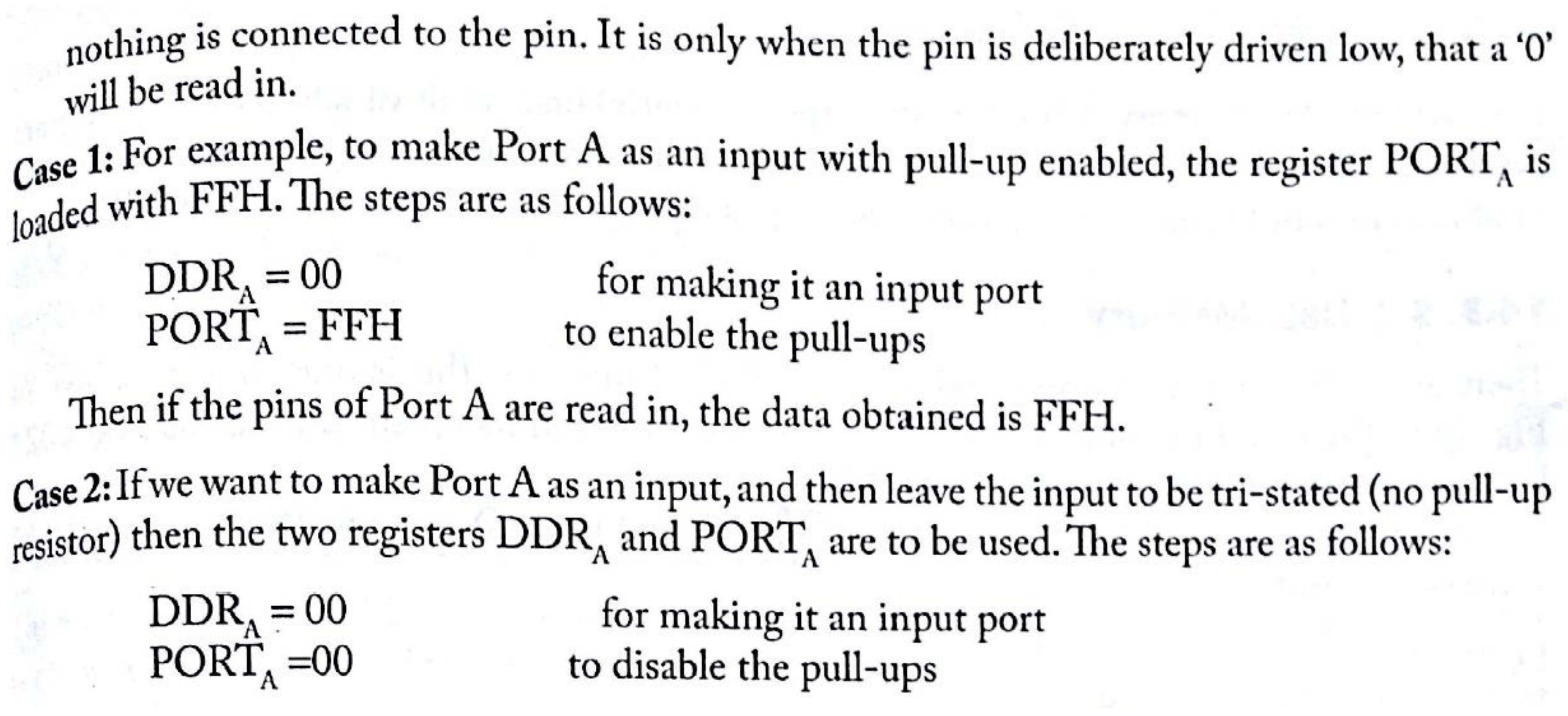


This register has two functions.

i) **To contain the data at the output pins:** When a port is configured as an output, and then data is sent to the port, the port pins will carry the data. The port register also has this data in it.
For example, let us make Port A as an output. So we load FFH into DDR_A. Then, if we send 0AH to the port, $PORT_A$ register will be found to have the value 0AH.

ii) **To enable or disable pull-up resistors:** This is applicable when a port is configured as an input port. In input mode, when pull-up is enabled, the data which is read is '1' even if

I/O Ports



nothing is connected to the pin. It is only when the pin is deliberately driven low, that a '0' will be read in.

Case 1: For example, to make Port A as an input with pull-up enabled, the register $PORT_A$ is loaded with FFH. The steps are as follows:

$DDR_A = 00$	for making it an input port
$PORT_A = FFH$	to enable the pull-ups

Then if the pins of Port A are read in, the data obtained is FFH.

Case 2: If we want to make Port A as an input, and then leave the input to be tri-stated (no pull-up resistor) then the two registers DDR_A and $PORT_A$ are to be used. The steps are as follows:

$DDR_A = 00$	for making it an input port
$PORT_A = 00$	to disable the pull-ups

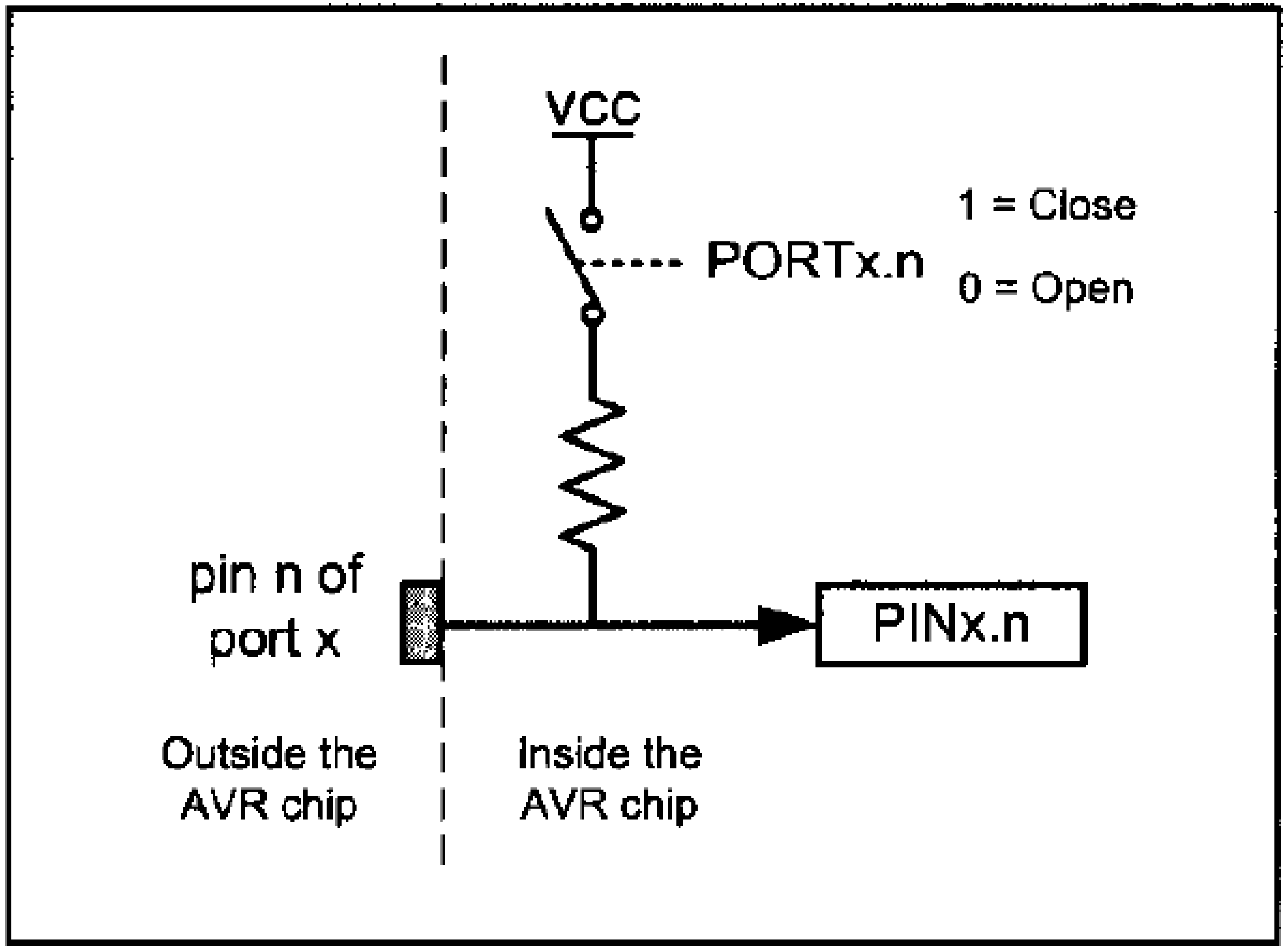


Figure 4-4. The Pull-up Resistor

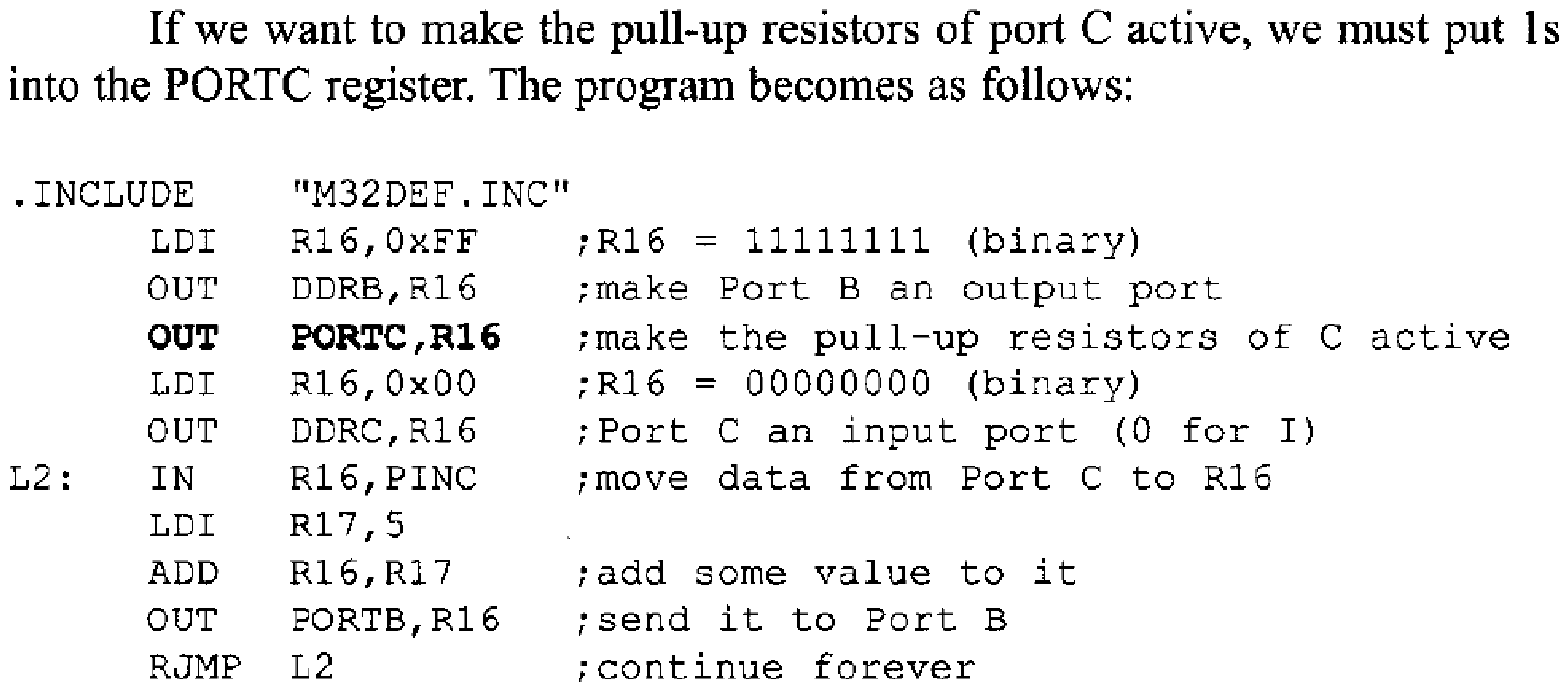
Note: *There is a pull up resistor for each of the AVR pins. If we put 1's into the pins of PORTx register the pull up resistors are activated. If we put 0's into the bits of the PORTx register, the pull-up resistor is inactive.*

I/O Ports

The following code gets the data present at the pins of port C and sends it to port B indefinitely, after adding the value 5 to it:

```
.INCLUDE "M32DEF.INC"
      LDI    R16,0x00     ;R16 = 00000000 (binary)
      OUT    DDRC,R16     ;make Port C an input port
      LDI    R16,0xFF     ;R16 = 11111111 (binary)
      OUT    DDRB,R16     ;make Port B an output port(1 for Out)
L2:   IN     R16,PINC     ;read data from Port C and put in R16
      LDI    R17,5
      ADD    R16,R17      ;add 5 to it
      OUT    PORTB,R16    ;send it to Port B
      RJMP   L2           ;continue forever
```

I/O Ports



If we want to make the pull-up resistors of port C active, we must put 1s into the PORTC register. The program becomes as follows:

```
.INCLUDE    "M32DEF.INC"
      LDI    R16,0xFF      ;R16 = 11111111 (binary)
      OUT    DDRB,R16      ;make Port B an output port
      OUT    PORTC,R16     ;make the pull-up resistors of C active
      LDI    R16,0x00      ;R16 = 00000000 (binary)
      OUT    DDRC,R16      ;Port C an input port (0 for I)
L2:   IN     R16,PINC      ;move data from Port C to R16
      LDI    R17,5
      ADD    R16,R17       ;add some value to it
      OUT    PORTB,R16     ;send it to Port B
      RJMP   L2            ;continue forever
```

In order to make all the bits of Port D an input, DDRD must be cleared by writing 0 to all the bits. In the following code, Port D is configured first as an input port by writing all 0s to register DDRD, and then data is received from Port D and saved in a RAM location:

```
.INCLUDE     "M32DEF.INC"
.EQU  MYTEMP 0x100      ;save it here

      LDI   R16,0x00    ;R16 = 00000000 (binary)
      OUT   DDRD,R16    ;make Port D an input port (0 for In)
      NOP
      IN    R16,PIND    ;move from pins of Port D to R16
      STS   MYTEMP,R16  ;save it in MYTEMP
```

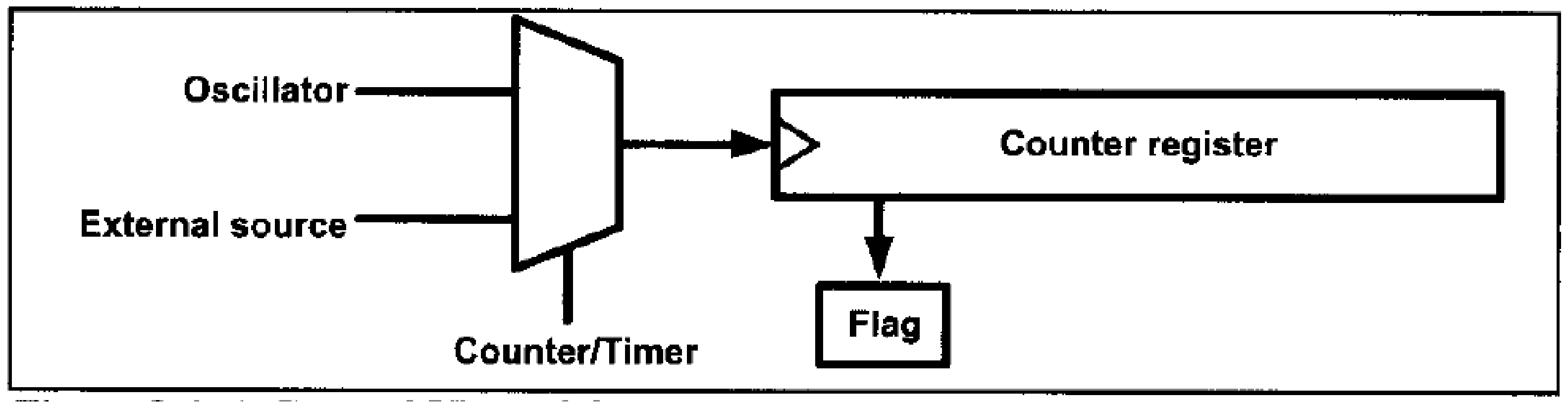
```
;toggle all bits of PORTA
.INCLUDE     "M32DEF.INC"
      LDI    R16,0xFF      ;R16 = 11111111 (binary)
      OUT    DDRA,R16      ;make Port A an output port
L1:   LDI    R16,0x55      ;R16 = 0x55
      OUT    PORTA,R16     ;put 0x55 on Port A pins
      CALL   DELAY
      LDI    R16,0xAA      ;R16 = 0xAA
      OUT    PORTA,R16     ;put 0xAA on Port A pins
      CALL   DELAY
      RJMP   L1
```

Timers

Many applications need to count an event or generate time delays. **Hence there are timer/counter registers for this purpose.**

When we want to count an event, we connect the external event source to the clock pin of the counter register. Then, when an event occurs externally, the content of the counter is incremented; in this way, the content of the counter represents how many times an event has occurred. When we want to generate time delays, we connect the oscillator to the clock pin of the counter. So, when the oscillator ticks, the content of the counter is incremented. As a result, the content of the counter register represents how many ticks have occurred from the time we have cleared the counter. Since the speed of the oscillator in a microcontroller is known, we can calculate the tick period, and from the content of the counter register we will know how much time has elapsed.

A General View of Timers and Counters in Microcontrollers



Timers

So, one way to generate a time delay is to clear the counter at the start time and wait until the counter reaches a certain number. For example, consider a microcontroller with an oscillator with frequency of 1 MHz; in the microcontroller, the content of the counter register increments once per microsecond. So, if we want a time delay of 100 microseconds, we should clear the counter and wait until it becomes equal to 100.

In the microcontrollers, there is a flag for each of the counters. The flag is set when the counter overflows, and it is cleared by software. The second method to generate a time delay is to load the counter register and wait until the counter overflows and the flag is set. For example, in a microcontroller with a frequency of 1 MHz, with an 8-bit counter register, if we want a time delay of 3 microseconds, we can load the counter register with $FD and wait until the flag is set after 3 ticks. After the first tick, the content of the register increments to $FE; after the second tick, it becomes $FF; and after the third tick, it overflows (the content of the register becomes $00) and the flag is set.

Timers

- In Atmega32 there are three timers

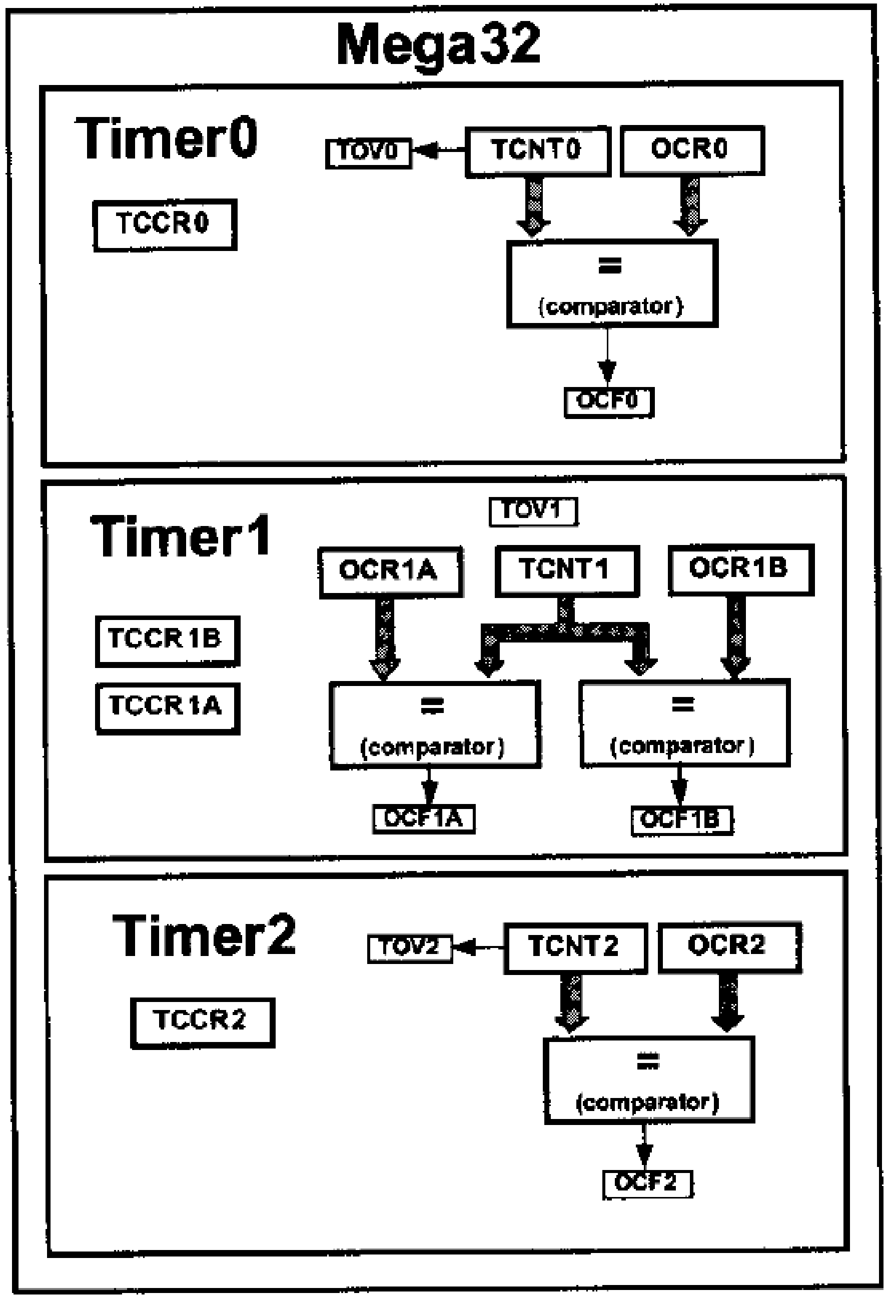
➢Timer0

➢Timer1

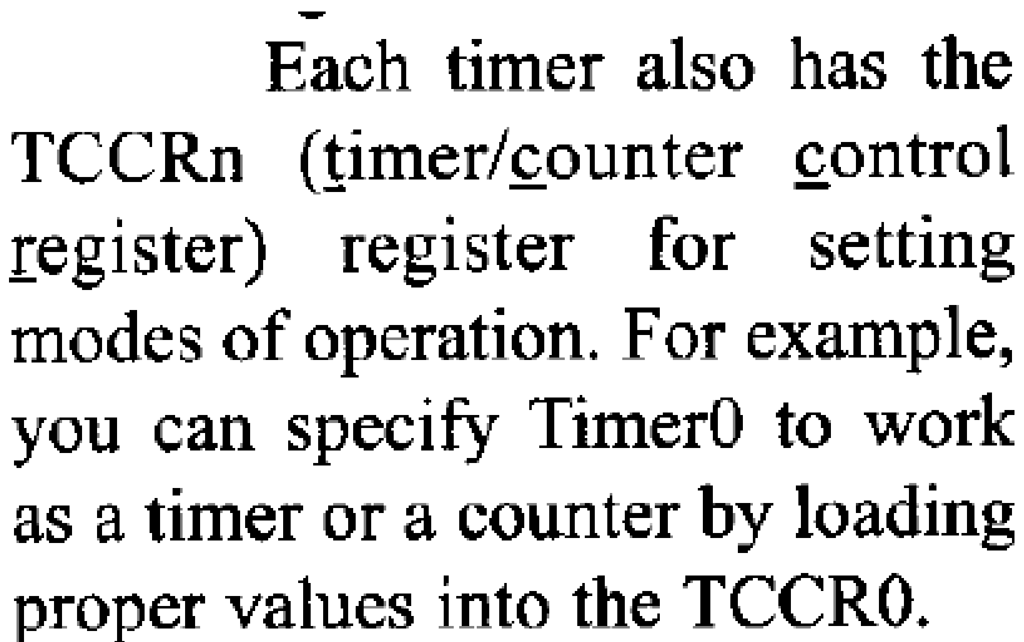
➢Timer2

Timer0 and Timer2 are 8-bit timers while timer1 is a 16-bit timer.

Every timer needs a clock pulse to tick. The clock source can be internal or external. If we use the internal clock source, then the frequency of the crystal oscillator is fed into the timer. Therefore, it is used for time delay generation and consequently is called a *timer*. By choosing the external clock option, we feed pulses through one of the AVR pins . This is called a counter.


Mega32
Timer0
TCCR0
TOV0
TCNT0
OCR0
=
(comparator)
OCF0
Timer1
TOV1
OCR1A
TCNT1
OCR1B
TCCR1B
TCCR1A
=
(comparator)
=
(comparator)
OCF1A
OCF1B
Timer2
TCCR2
TOV2
TCNT2
OCR2
=
(comparator)
OCF2

Timers

- In AVR each timer has TCNTn register. That means in Atmega32 we have TCNT0,TCNT1,TCNT2 registers.
- Upon reset TCNTn contains zero.It counts up(increments) with each pulse.
- The contents of the timers/counters can be accessed using TCNTn register.We can load a value into TCNTn or we can even read its value.
- Each timer has a TOVn(Timer Overflow) flag as well. When a timer overflows its TOVn flag will be set.
- 

Each timer also has the TCCRn (timer/counter control register) register for setting modes of operation. For example, you can specify Timer0 to work as a timer or a counter by loading proper values into the TCCR0.

Timers

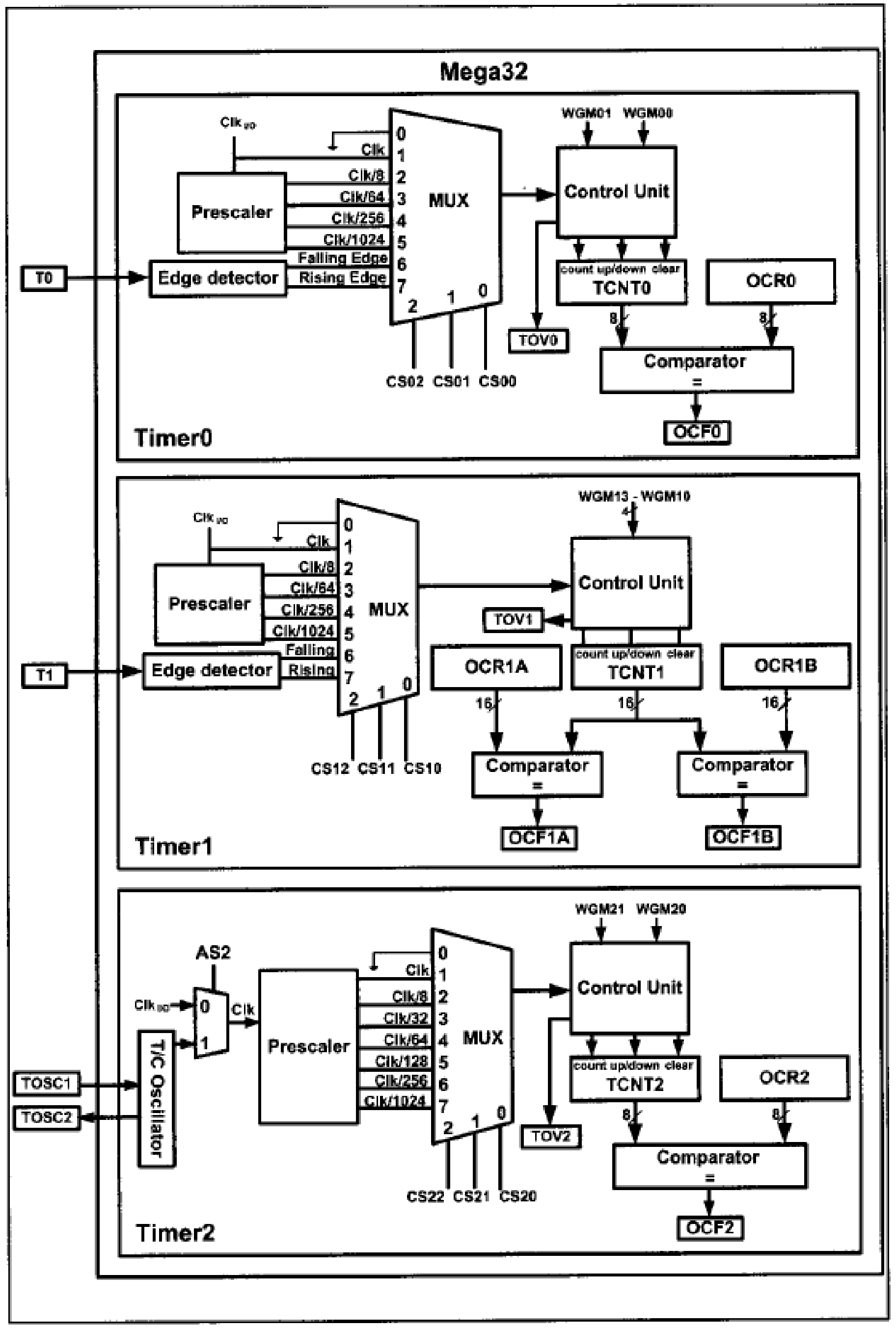
Each timer also has an OCRn (Output Compare Register) register. The content of the OCRn is compared with the content of the TCNTn. When they are equal the OCFn (Output Compare Flag) flag will be set.

The timer registers are located in the I/O register memory. Therefore, you can read or write from timer registers using IN and OUT instructions, like the other I/O registers. For example, the following instructions load TCNT0 with 25:

```
LDI R20,25          ;R20 = 25
OUT TCNT0,R20       ;TCNT0 = R20
```

or "`IN R19,TCNT2`" copies TCNT2 to R19.

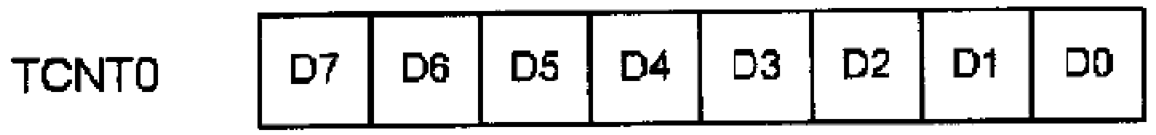
Timers in Atmega32



Timers

- **Timer0:**

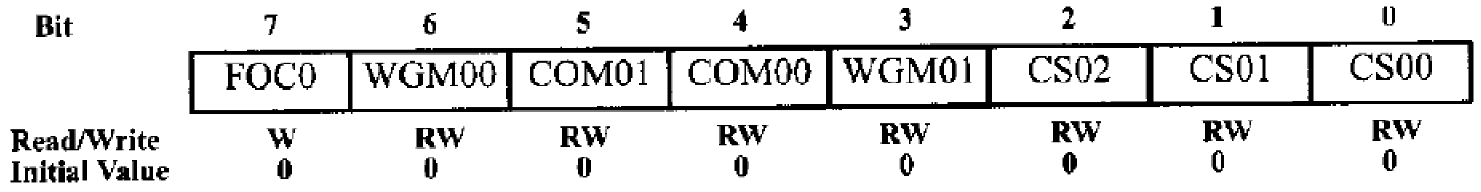
Timer0 is 8-bit in ATmega32; thus, TCNT0 is 8-bit as shown



TCNT0	D7	D6	D5	D4	D3	D2	D1	D0

- ***TCCR0 Register***:

TCCR0 is an 8-bit register and used for the control of timer0.



Bit	7	6	5	4	3	2	1	0
	FOC0	WGM00	COM01	COM00	WGM01	CS02	CS01	CS00
Read/Write	W	RW	RW	RW	RW	RW	RW	RW
Initial Value	0	0	0	0	0	0	0	0

FOC0 D7 Force compare match: This is a write-only bit, which can be used while generating a wave. Writing 1 to it causes the wave generator to act as if a compare match had occurred.

WGM00, WGM01

D6	D3	Timer0 mode selector bits
0	0	Normal
0	1	CTC (Clear Timer on Compare Match)
1	0	PWM, phase correct
1	1	Fast PWM

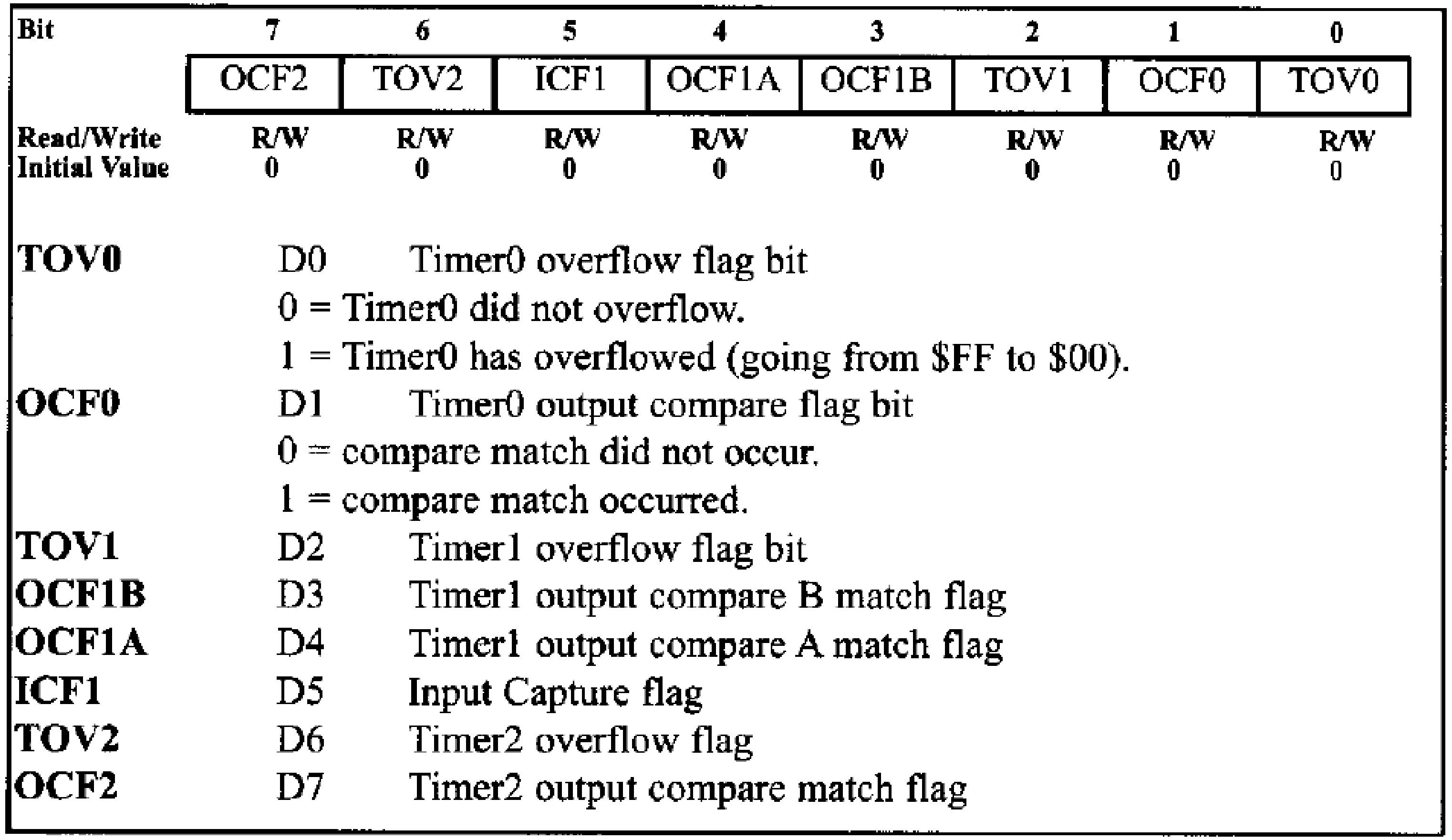
COM01:00 D5 D4 Compare Output Mode:
These bits control the waveform generator (see Chapter 15).

CS02:00

D2	D1	D0	Timer0 clock selector
0	0	0	No clock source (Timer/Counter stopped)
0	0	1	clk (No Prescaling)
0	1	0	clk / 8
0	1	1	clk / 64
1	0	0	clk / 256
1	0	1	clk / 1024
1	1	0	External clock source on T0 pin. Clock on falling edge.
1	1	1	External clock source on T0 pin. Clock on rising edge.

Timers

- Timer/ … ains the flags o

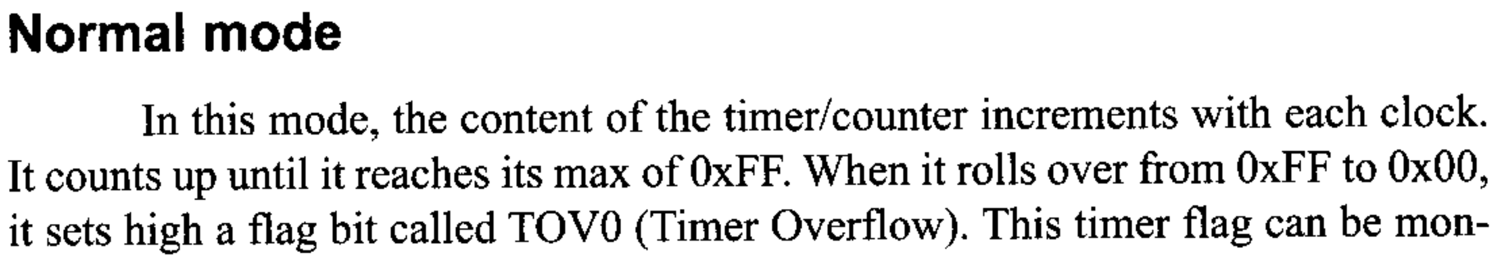


Bit	7	6	5	4	3	2	1	0
	OCF2	TOV2	ICF1	OCF1A	OCF1B	TOV1	OCF0	TOV0
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Initial Value	0	0	0	0	0	0	0	0

TOV0	D0	Timer0 overflow flag bit 0 = Timer0 did not overflow. 1 = Timer0 has overflowed (going from $FF to $00).
OCF0	D1	Timer0 output compare flag bit 0 = compare match did not occur. 1 = compare match occurred.
TOV1	D2	Timer1 overflow flag bit
OCF1B	D3	Timer1 output compare B match flag
OCF1A	D4	Timer1 output compare A match flag
ICF1	D5	Input Capture flag
TOV2	D6	Timer2 overflow flag
OCF2	D7	Timer2 output compare match flag

Timers

- **TOV0 Flag:** The TOV0 flag is set when the timer rolls over from $FF to 00 and it remains set until the software clears it.



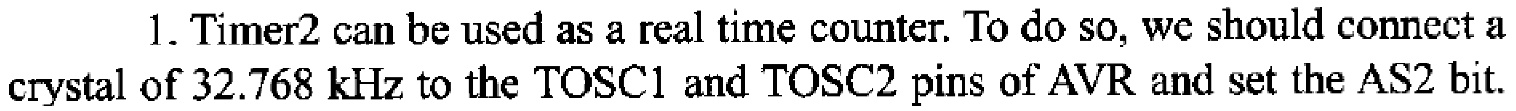
Normal mode

In this mode, the content of the timer/counter increments with each clock. It counts up until it reaches its max of 0xFF. When it rolls over from 0xFF to 0x00, it sets high a flag bit called TOV0 (Timer Overflow). This timer flag can be mon-

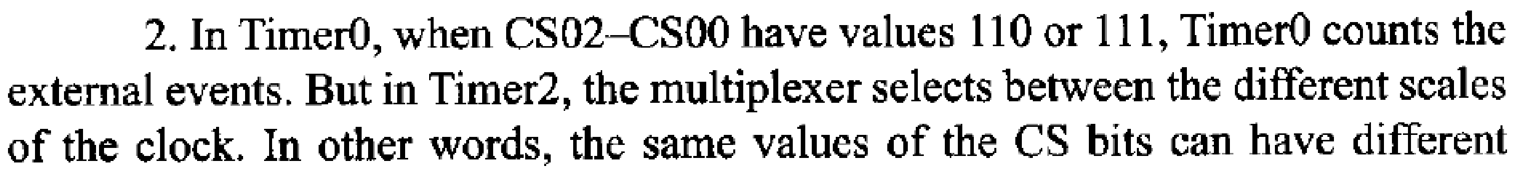
CTC Mode(Clear Timer0 on Compare Match): The OCR0 register is used with CTC mode.In the CTC mode the timer is incremented with the clock. But it counts up until the contents of TCNT0 register becomes equal with the contents of OCR0 register. Then the timer will be cleared and the OCF0 flag will be set.

Timer 2:

- Like timer0 , Timer2 also has the above set of registers and hence the working of timer0 and timer 2 are almost similar with few differences.



1. Timer2 can be used as a real time counter. To do so, we should connect a crystal of 32.768 kHz to the TOSC1 and TOSC2 pins of AVR and set the AS2 bit.

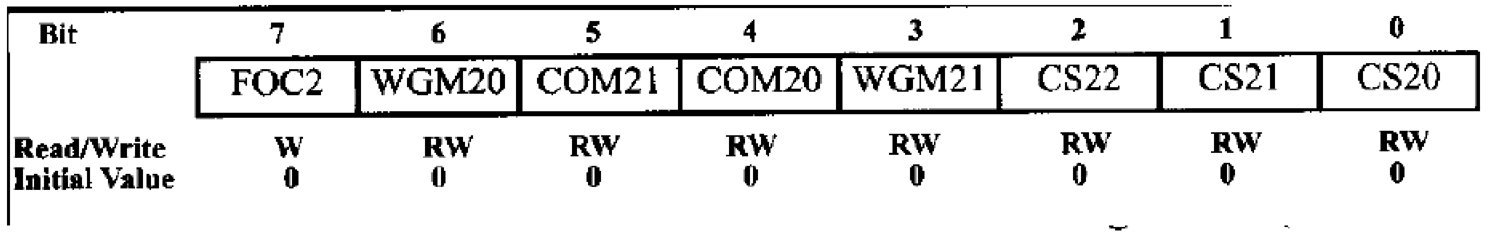


2. In Timer0, when CS02–CS00 have values 110 or 111, Timer0 counts the external events. But in Timer2, the multiplexer selects between the different scales of the clock. In other words, the same values of the CS bits can have different

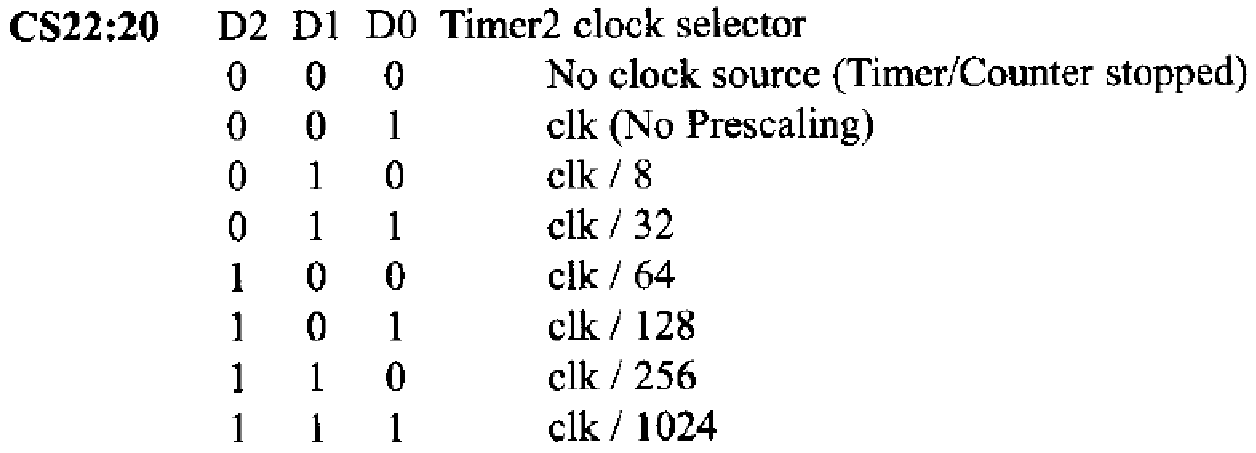
meanings for Timer0 and Timer2.

Timers

- **TCCR2(Timer/ counter Control Register 2):**

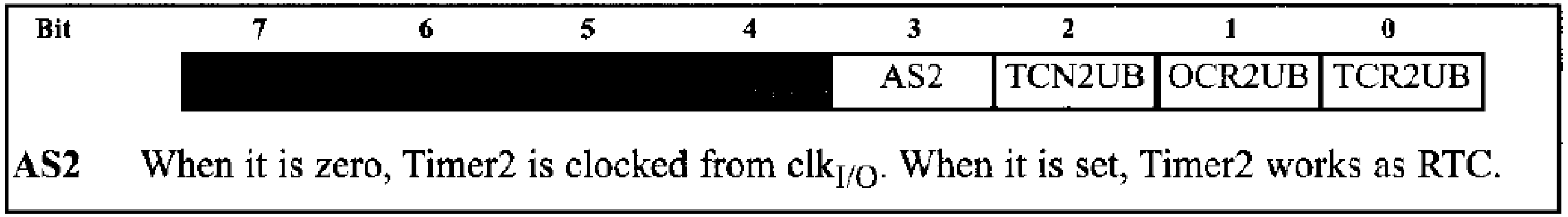


Bit	7	6	5	4	3	2	1	0
	FOC2	WGM20	COM21	COM20	WGM21	CS22	CS21	CS20
Read/Write	W	RW	RW	RW	RW	RW	RW	RW
Initial Value	0	0	0	0	0	0	0	0



CS22:20	D2	D1	D0	Timer2 clock selector
	0	0	0	No clock source (Timer/Counter stopped)
	0	0	1	clk (No Prescaling)
	0	1	0	clk / 8
	0	1	1	clk / 32
	1	0	0	clk / 64
	1	0	1	clk / 128
	1	1	0	clk / 256
	1	1	1	clk / 1024

Asynchronous Status Register:

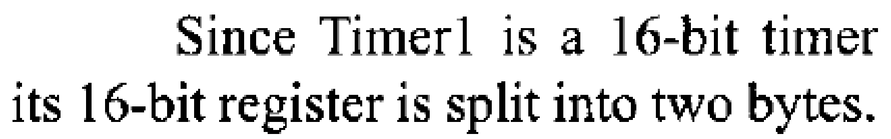


Bit	7	6	5	4	3	2	1	0
					AS2	TCN2UB	OCR2UB	TCR2UB

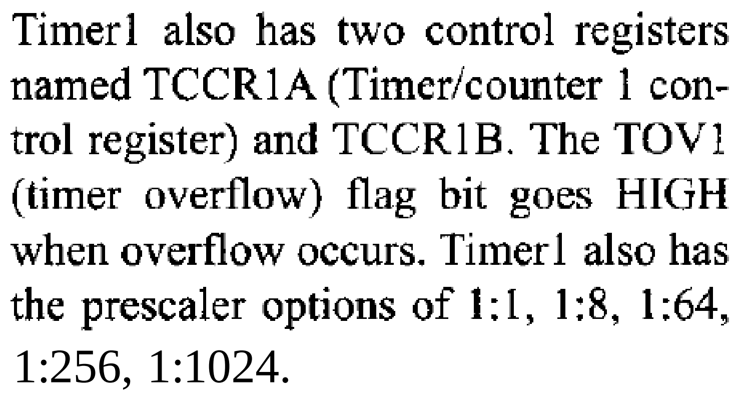
AS2 When it is zero, Timer2 is clocked from $\mathrm{clk}_{I/O}$. When it is set, Timer2 works as RTC.

Timers

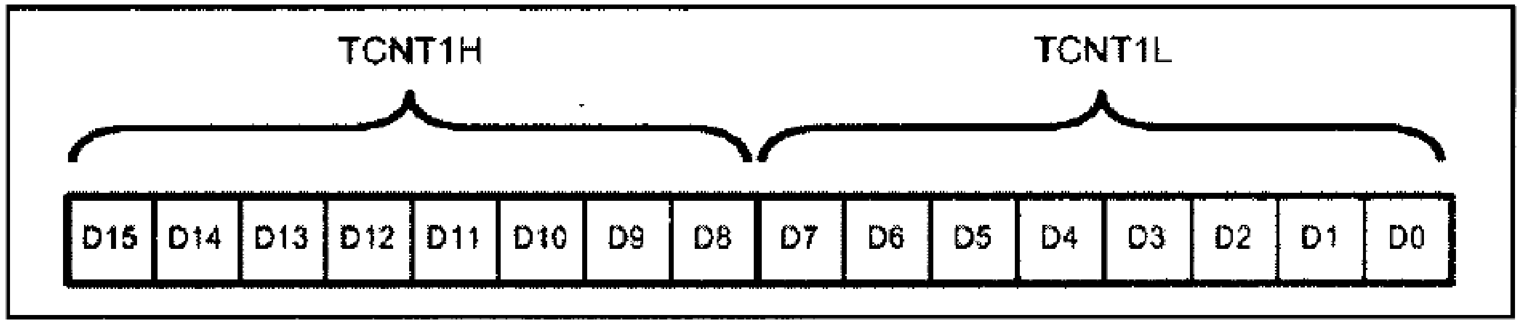
Timer1 : Timer1 is a 16-bit timer.


Since Timer1 is a 16-bit timer
its 16-bit register is split into two bytes.

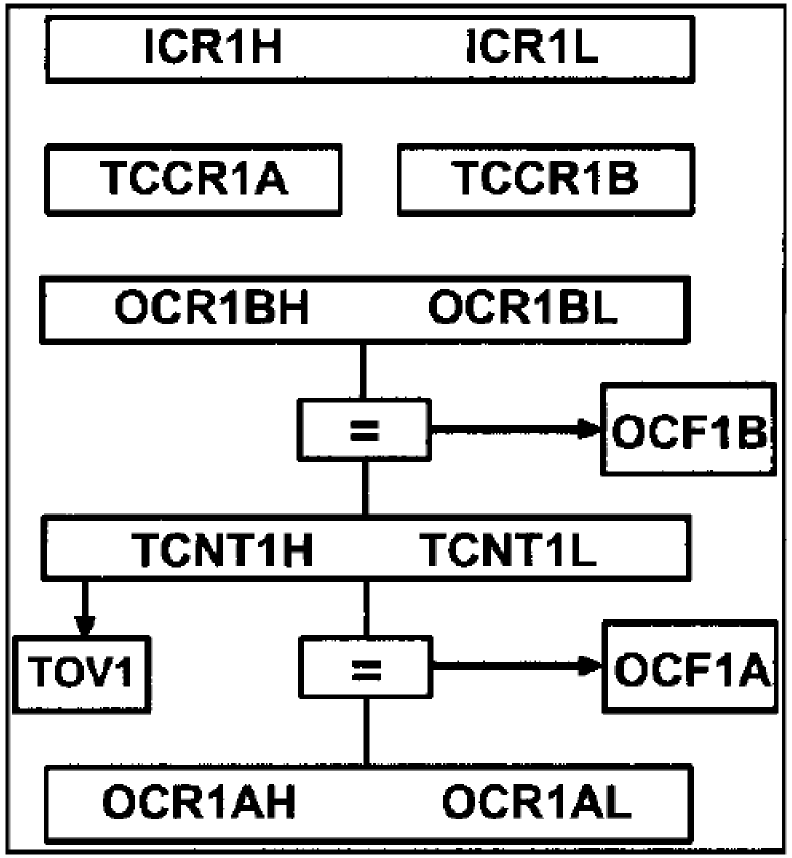
- These are referred to as TCNT1L (Timer1 lower byte) and TCNT1H(Timer1 higher Byte)


Timer1 also has two control registers named TCCR1A (Timer/counter 1 control register) and TCCR1B. The TOV1 (timer overflow) flag bit goes HIGH when overflow occurs. Timer1 also has the prescaler options of 1:1, 1:8, 1:64, 1:256, 1:1024.

- There are two OCR registers each of 16-bit in timer1:OCR1A and OCR1B.There are two separate flags for each of the OCR registers which act independently of each other.
- When TCNT1 equals OCR1A, THE OCF1A flag will be set. When the TCNT1 equal OCR1B the OCF1B flag will be set.
- There is also an auxiliary register named ICR1(Input Capture Register) which is used in operations such as capturing.ICR1 is a 16-bit register made of ICRH and ICRL.



Timer1 High and low Registers



Simplified diagram of timer1

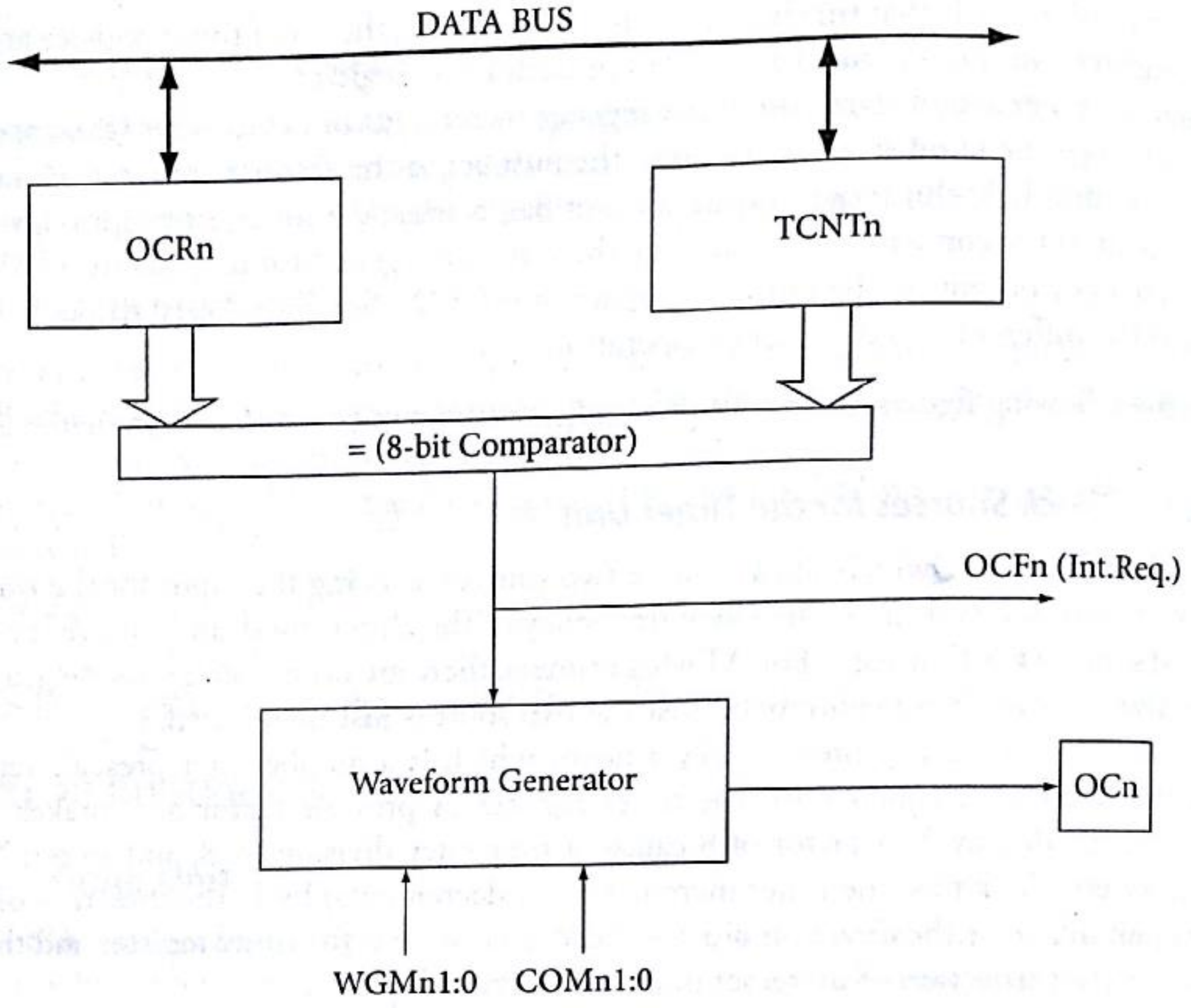


Figure 14.11 | Comparison of count values and waveform generation

USART

Serial data communication uses two methods, asynchronous and synchronous. The *synchronous* method transfers a block of data (characters) at a time, whereas the *asynchronous* method transfers a single byte at a time. It is possible to write software to use either of these methods, but the programs can be tedious and long. For this reason, special IC chips are made by many manufacturers for serial data communications. These chips are commonly referred to as UART (universal asynchronous receiver-transmitter) and USART (universal synchronous-asynchronous receiver-transmitter). The AVR chip has a built-in USART, which is

The USART (universal synchronous asynchronous receiver/transmitter) in the AVR has normal asynchronous, double-speed asynchronous, master synchronous, and slave synchronous mode features. The synchronous mode can be used to transfer data between the AVR and external peripherals such as ADC and EEPROMs. The asynchronous mode is the one we will use to connect the AVR-based system to the x86 PC serial port for the purpose of full-duplex serial data transfer.

USART

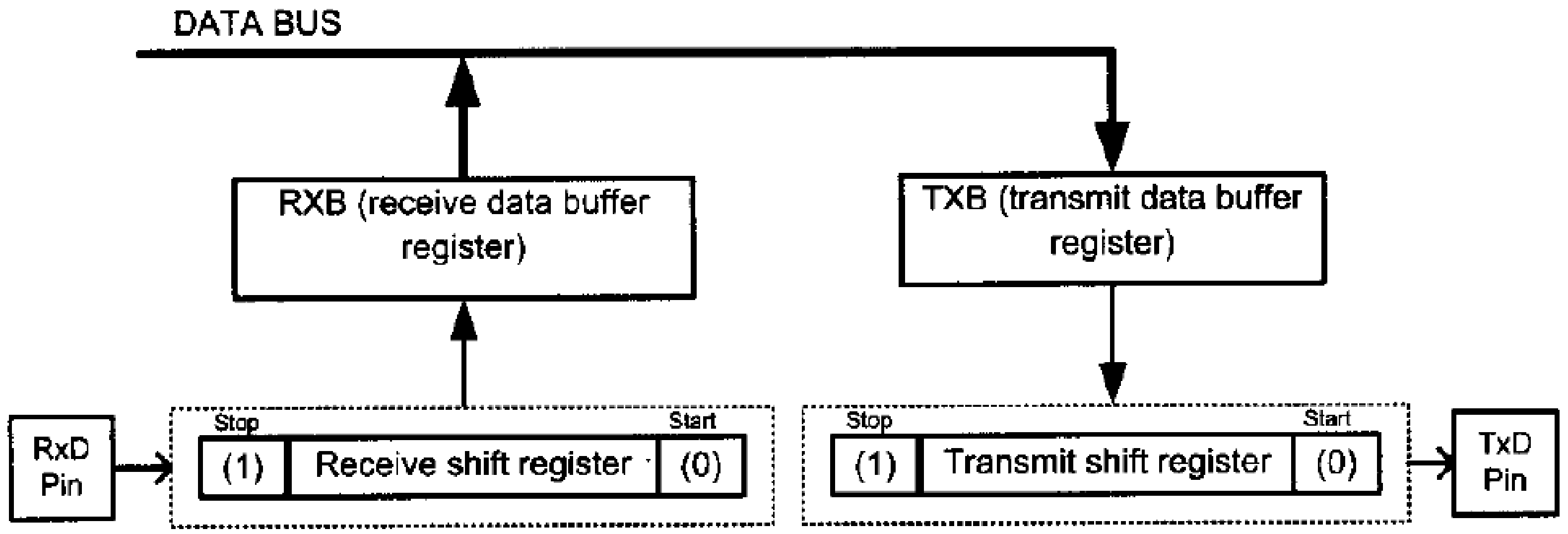
In the AVR microcontroller five registers are associated with the USART

They are UDR (USART Data Register), UCSRA, UCSRB, UCSRC (USART Control Status Register), and UBRR (USART Baud Rate Register).

UDR: Universal Data Register

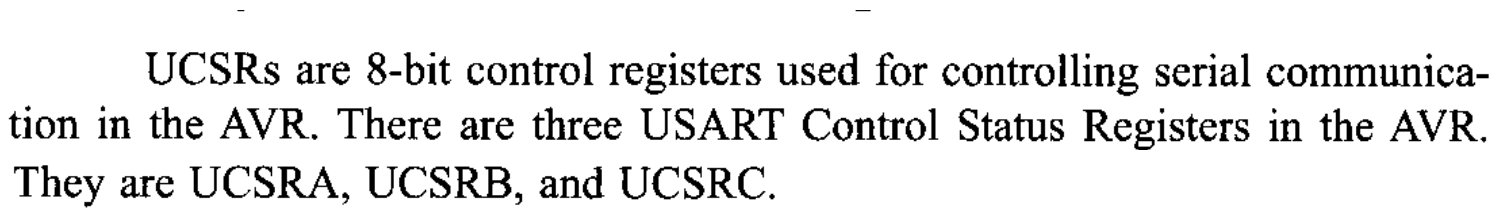
In the AVR, to provide a full-duplex serial communication, there are two shift registers referred to as *Transmit Shift Register* and *Receive Shift Register*. Each shift register has a buffer that is connected to it directly. These buffers are called *Transmit Data Buffer Register* and *Receive Data Buffer Register*. The USART Transmit Data Buffer Register and USART Receive Data Buffer Register share the same I/O address, which is called *USART Data Register* or *UDR*. When you write data to UDR, it will be transferred to the Transmit Data Buffer Register (TXB), and when you read data from UDR, it will return the contents of the Receive Data Buffer Register (RXB).

Simplified USART BLOCK DIAGRAM



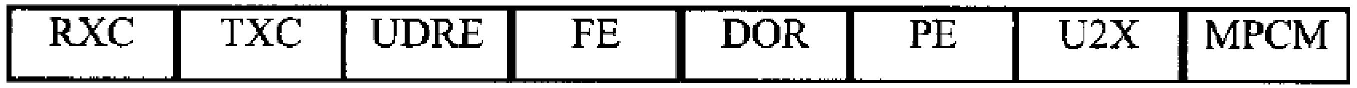
USART

- **UCSR Registers(USART CONTROL STATUS REGISTERS):**



UCSRs are 8-bit control registers used for controlling serial communication in the AVR. There are three USART Control Status Registers in the AVR. They are UCSRA, UCSRB, and UCSRC.

- **UCSRA Register:**



RXC	TXC	UDRE	FE	DOR	PE	U2X	MPCM

- **Bit 7 – RXC:** USART Receive Complete

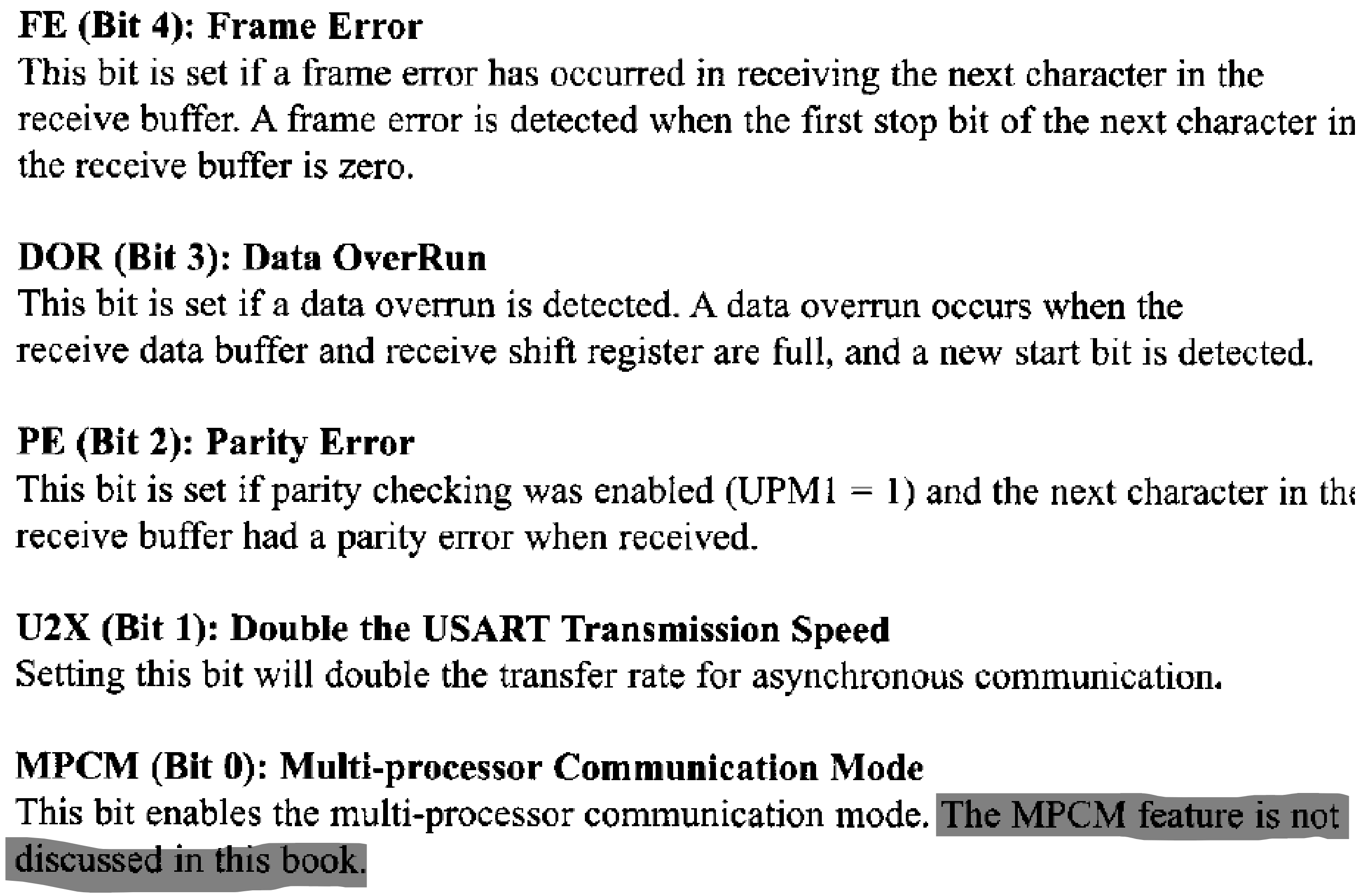
This flag bit is set when there is unread data in the receive buffer. The RXC Flag can be used to generate a Receive Complete interrupt.

- **Bit 6 – TXC:** USART Transmit Complete

This flag bit is set when the entire frame from Tx Buffer is shifted out and there is no new data currently present in the transmit buffer (UDR). The TXC Flag can generate a Transmit Complete interrupt.

USART

- **Bit 5 – UDRE**: USART Data Register Empty
- If UDRE is one, the buffer is empty which indicates the transmit data buffer (UDR) is ready to receive new data. The UDRE Flag can generate a Data Register Empty Interrupt. UDRE is set after a reset to indicate that the transmitter is readv.



FE (Bit 4): Frame Error
This bit is set if a frame error has occurred in receiving the next character in the receive buffer. A frame error is detected when the first stop bit of the next character in the receive buffer is zero.

DOR (Bit 3): Data OverRun
This bit is set if a data overrun is detected. A data overrun occurs when the receive data buffer and receive shift register are full, and a new start bit is detected.

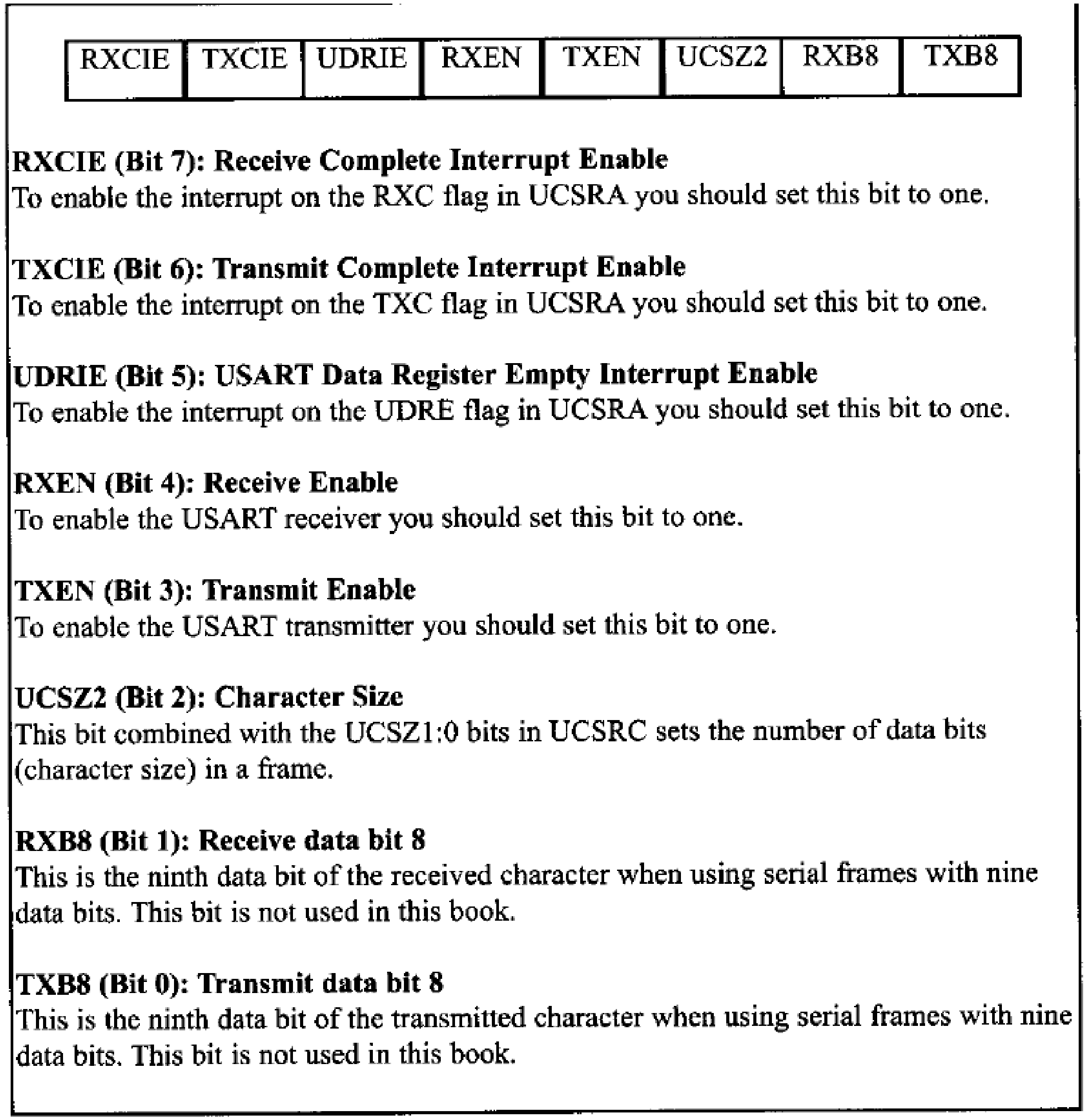
PE (Bit 2): Parity Error
This bit is set if parity checking was enabled (UPM1 = 1) and the next character in the receive buffer had a parity error when received.

U2X (Bit 1): Double the USART Transmission Speed
Setting this bit will double the transfer rate for asynchronous communication.

MPCM (Bit 0): Multi-processor Communication Mode
This bit enables the multi-processor communication mode. The MPCM feature is not discussed in this book.

USART

- **UCSRB Register:**



RXCIE	TXCIE	UDRIE	RXEN	TXEN	UCSZ2	RXB8	TXB8

RXCIE (Bit 7): Receive Complete Interrupt Enable
To enable the interrupt on the RXC flag in UCSRA you should set this bit to one.

TXCIE (Bit 6): Transmit Complete Interrupt Enable
To enable the interrupt on the TXC flag in UCSRA you should set this bit to one.

UDRIE (Bit 5): USART Data Register Empty Interrupt Enable
To enable the interrupt on the UDRE flag in UCSRA you should set this bit to one.

RXEN (Bit 4): Receive Enable
To enable the USART receiver you should set this bit to one.

TXEN (Bit 3): Transmit Enable
To enable the USART transmitter you should set this bit to one.

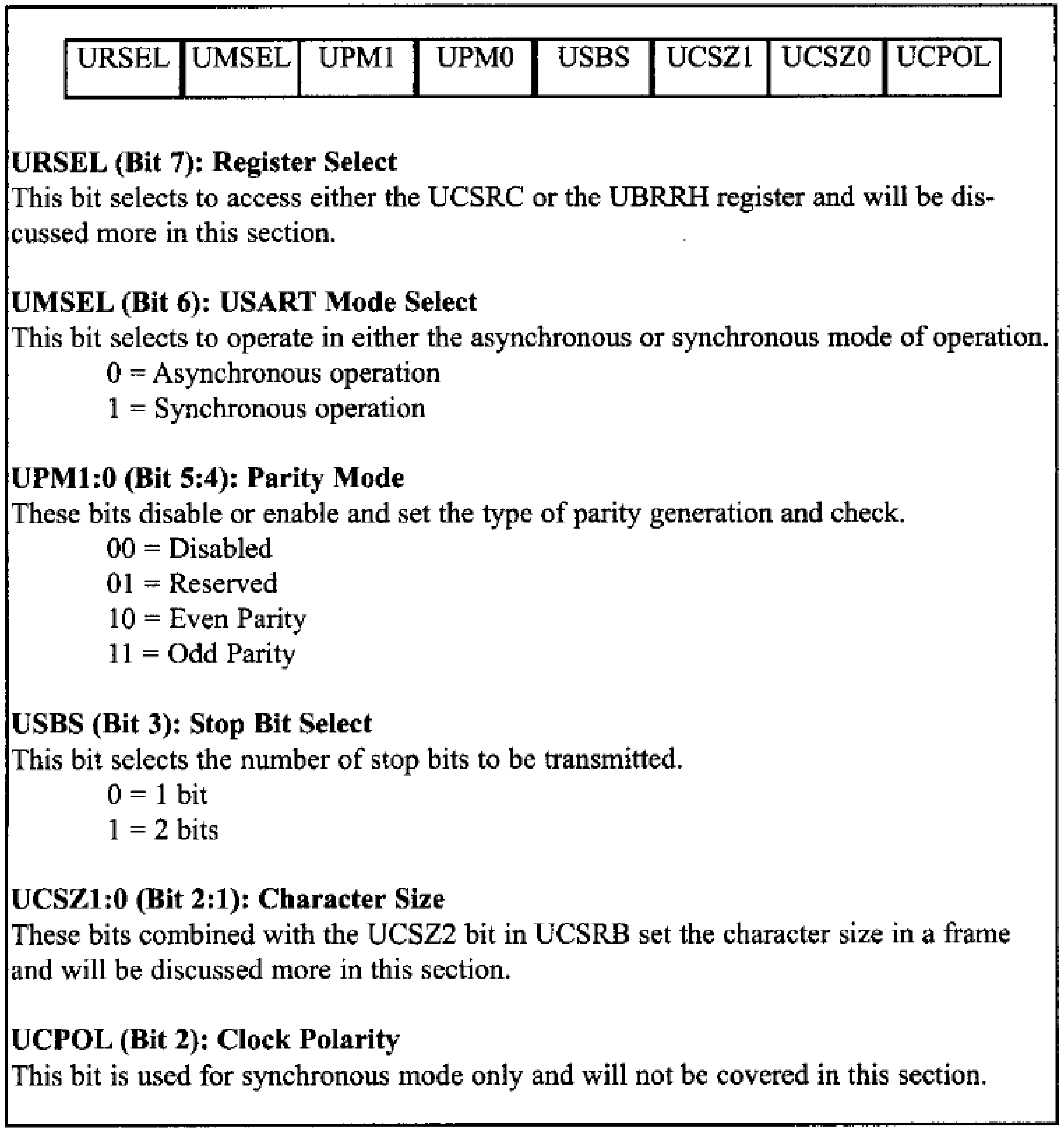
UCSZ2 (Bit 2): Character Size
This bit combined with the UCSZ1:0 bits in UCSRC sets the number of data bits (character size) in a frame.

RXB8 (Bit 1): Receive data bit 8
This is the ninth data bit of the received character when using serial frames with nine data bits. This bit is not used in this book.

TXB8 (Bit 0): Transmit data bit 8
This is the ninth data bit of the transmitted character when using serial frames with nine data bits. This bit is not used in this book.

USART

- **UCSRC Register:**



URSEL	UMSEL	UPM1	UPM0	USBS	UCSZ1	UCSZ0	UCPOL

URSEL (Bit 7): Register Select
This bit selects to access either the UCSRC or the UBRRH register and will be discussed more in this section.

UMSEL (Bit 6): USART Mode Select
This bit selects to operate in either the asynchronous or synchronous mode of operation.

0 =	Asynchronous operation
1 =	Synchronous operation

UPM1:0 (Bit 5:4): Parity Mode
These bits disable or enable and set the type of parity generation and check.

00 =	Disabled
01 =	Reserved
10 =	Even Parity
11 =	Odd Parity

USBS (Bit 3): Stop Bit Select
This bit selects the number of stop bits to be transmitted.

0 =	1 bit
1 =	2 bits

UCSZ1:0 (Bit 2:1): Character Size
These bits combined with the UCSZ2 bit in UCSRB set the character size in a frame and will be discussed more in this section.

UCPOL (Bit 2): Clock Polarity
This bit is used for synchronous mode only and will not be covered in this section.

USART

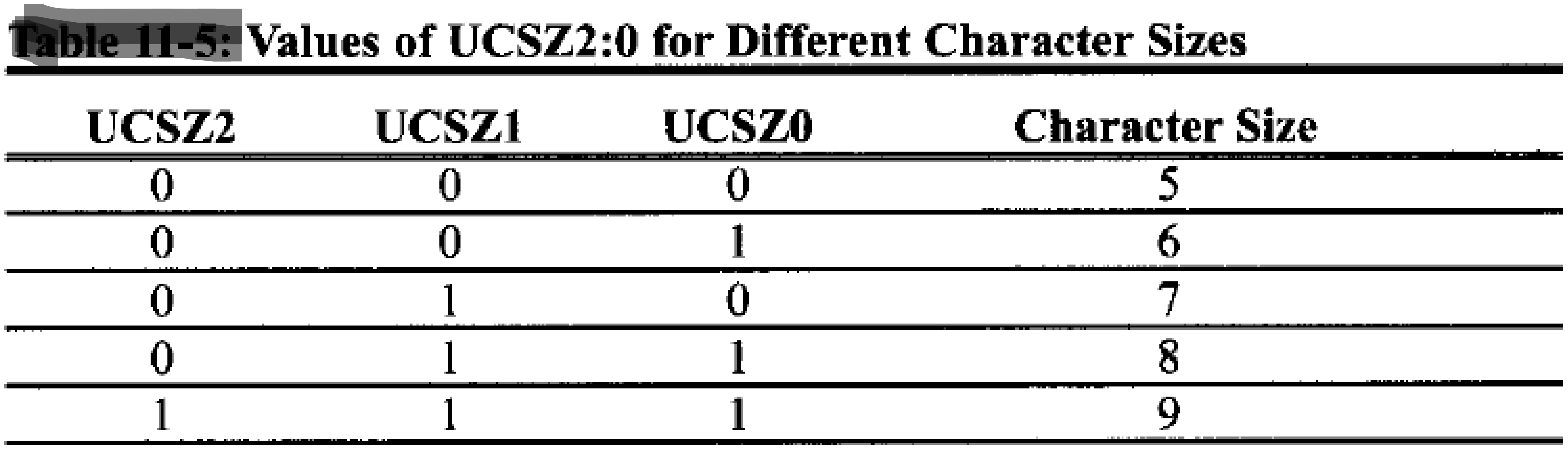


Table 11-5: Values of UCSZ2:0 for Different Character Sizes

UCSZ2	UCSZ1	UCSZ0	Character Size
0	0	0	5
0	0	1	6
0	1	0	7
0	1	1	8
1	1	1	9

USART

- **Universal Baud Rate Register:**

AVR transfers and receives data serially at many different baud rates. The baud rate in the AVR is programmable. This is done with the help of the 8-bit register called UBRR. For a given crystal frequency, the value loaded into the UBRR decides the baud rate. The relation between the value loaded into UBBR and the Fosc (frequency of oscillator connected to the XTAL1 and XTAL2 pins) is dictated by the following formula:

Desired Baud Rate = Fosc/ (16(X + 1))

where X is the value we load into the UBRR register. To get the X value for different baud rates we can solve the equation as follows:

X = (Fosc/ (16(Desired Baud Rate))) – 1

Assuming that Fosc = 8 MHz, we have the following:

Desired Baud Rate = Fosc/ (16(X + 1)) = 8 MHz/16(X + 1) = 500 kHz/(X + 1)

X = (500 kHz/ Desired Baud Rate) – 1

Table 11-3: Some PC Baud Rates in HyperTerminal

1,200
2,400
4,800
9,600
19,200
38,400
57,600
115,200

USART

Baud Rate	UBRR (Decimal Value)	UBRR (Hex Value)
38400	12	C
19200	25	19
9600	51	33
4800	103	67
2400	207	CF
1200	415	19F

Note: For Fosc = 8 MHz we have UBRR = (500000/BaudRate) – 1

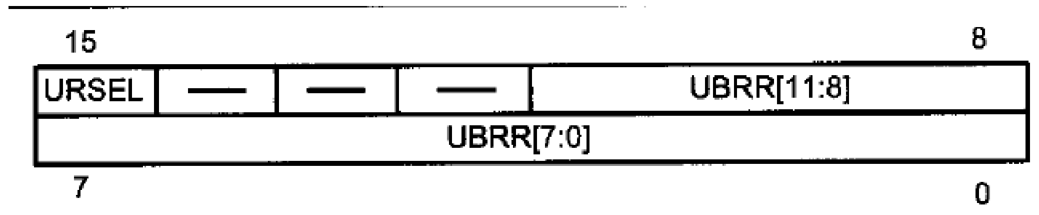
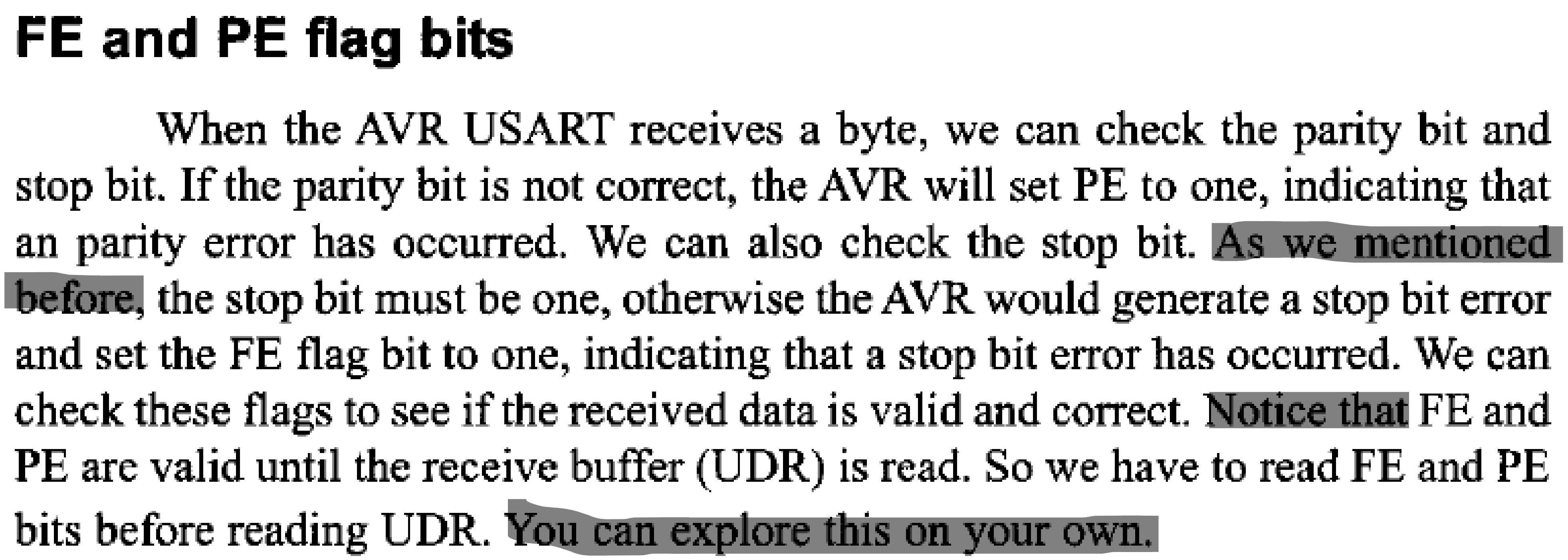


FIG:UBRR REGISTER

Before you start serial communication you have to enable the USART receiver or USART transmitter by writing one to the RXEN or TXEN bit of UCSRB. As we mentioned before, in the AVR you can use either synchronous or asynchronous operating mode. The UMSEL bit of the UCSRC register selects the USART operating mode. Since we want to use synchronous USART operating mode, we have to set the UMSEL bit to one. Also you have to set an identical character size for both transmitter and the receiver. If the character size of the receiver does not match the character size of the transmitter, data transfer would fail. Parity mode and number of stop bits are other factors that the receiver and transmitter must agree on before starting USART communication.

USART



FE and PE flag bits

When the AVR USART receives a byte, we can check the parity bit and stop bit. If the parity bit is not correct, the AVR will set PE to one, indicating that an parity error has occurred. We can also check the stop bit. As we mentioned before, the stop bit must be one, otherwise the AVR would generate a stop bit error and set the FE flag bit to one, indicating that a stop bit error has occurred. We can check these flags to see if the received data is valid and correct. Notice that FE and PE are valid until the receive buffer (UDR) is read. So we have to read FE and PE bits before reading UDR. You can explore this on your own.

Interrupt Structure

- A single microcontroller can serve several devices. There are two methods by which devices receive service from the microcontroller.

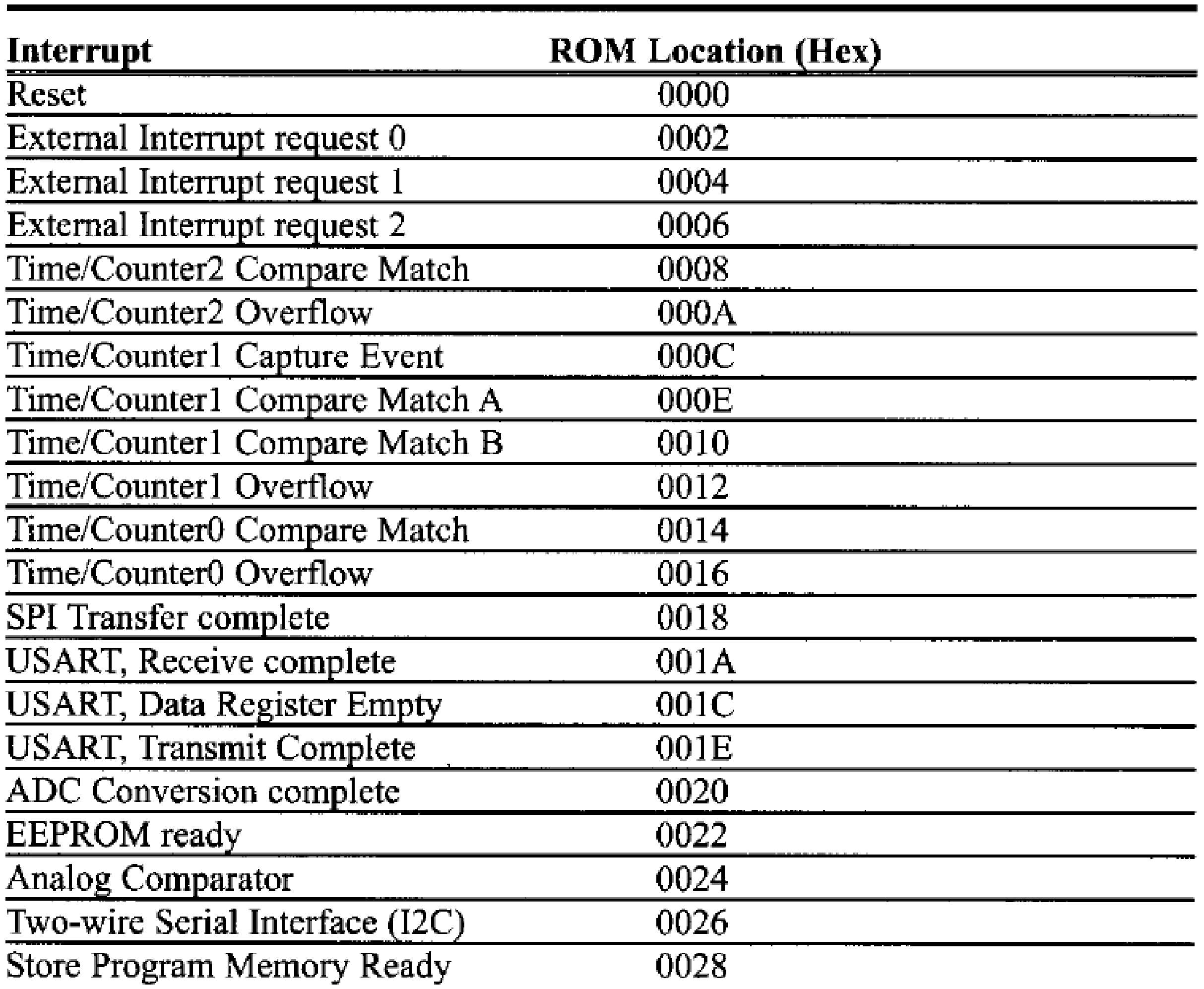
1. Interrupts
2. Polling

In the interrupt method whenever any device needs microcontroller’s service , the device notifies it by sending an interrupt signal. Upon receiving the interrupt signal the microcontroller stops whatever it is doing and serves the device.

The program associated with the interrupt is called Interrupt service Routine or Interrupt Handler.

The group of memory locations set aside to hold the addresses of ISRs is called the interrupt vector table.

Interrupt Structure

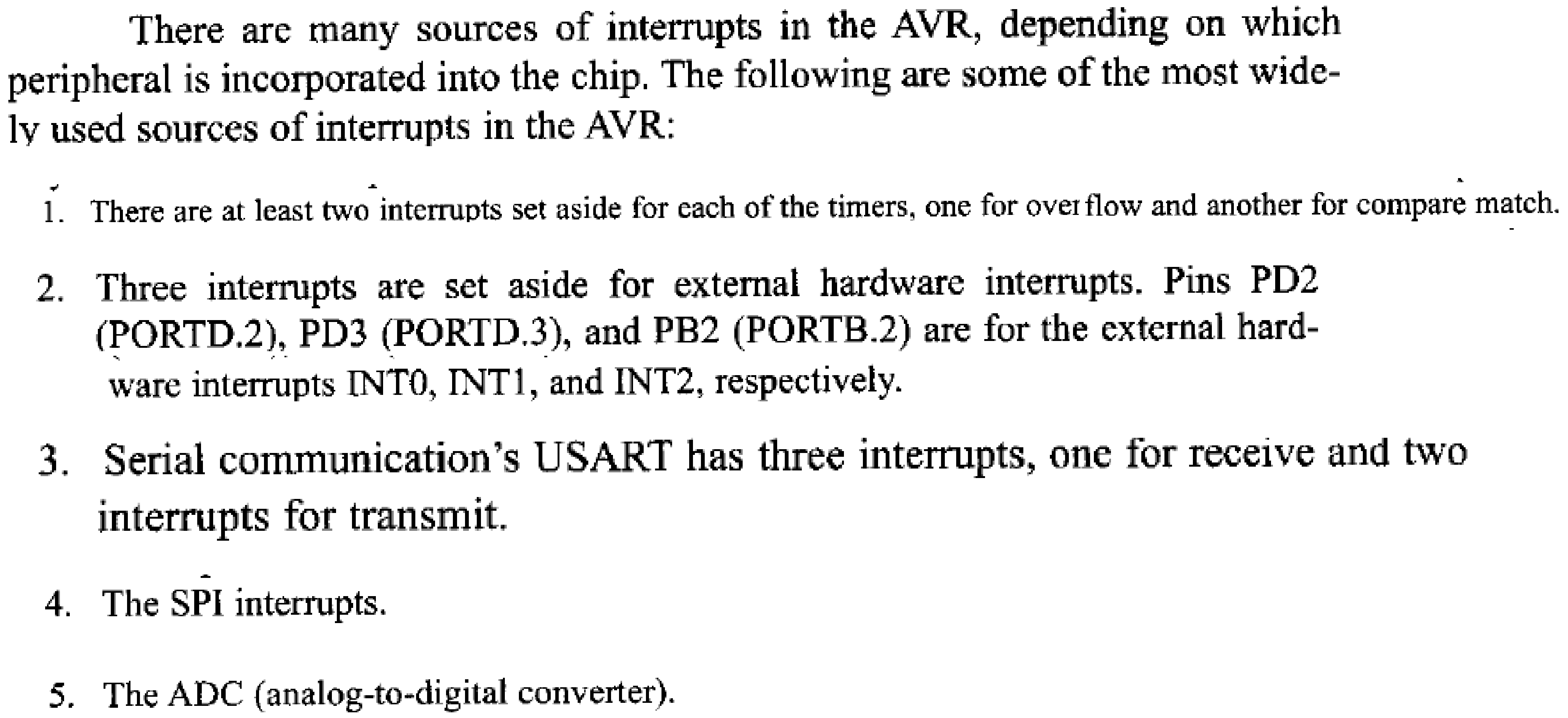


Interrupt	ROM Location (Hex)
Reset	0000
External Interrupt request 0	0002
External Interrupt request 1	0004
External Interrupt request 2	0006
Time/Counter2 Compare Match	0008
Time/Counter2 Overflow	000A
Time/Counter1 Capture Event	000C
Time/Counter1 Compare Match A	000E
Time/Counter1 Compare Match B	0010
Time/Counter1 Overflow	0012
Time/Counter0 Compare Match	0014
Time/Counter0 Overflow	0016
SPI Transfer complete	0018
USART, Receive complete	001A
USART, Data Register Empty	001C
USART, Transmit Complete	001E
ADC Conversion complete	0020
EEPROM ready	0022
Analog Comparator	0024
Two-wire Serial Interface (I2C)	0026
Store Program Memory Ready	0028

Interrupt Vector Table of Atmega32

Interrupt Structure

- Sources of Interrupts in AVR:



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

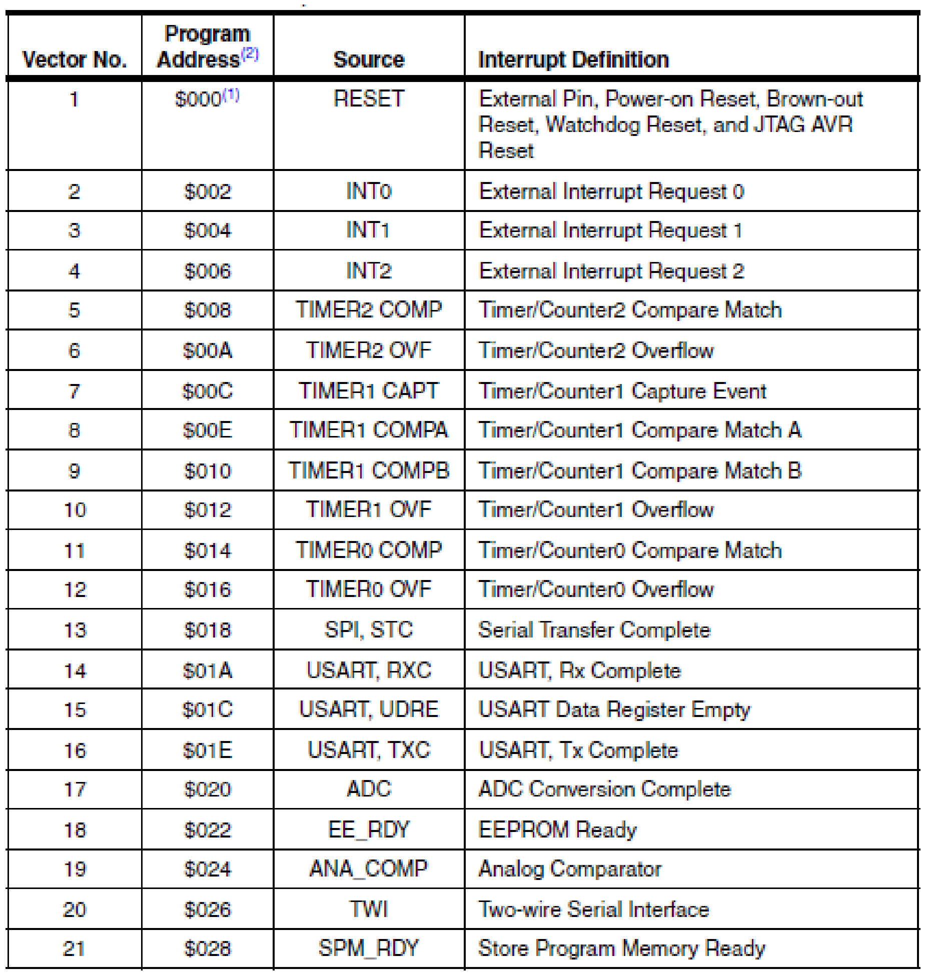
1. There are at least two interrupts set aside for each of the timers, one for overflow and another for compare match.
2. Three interrupts are set aside for external hardware interrupts. Pins PD2 (PORTD.2), PD3 (PORTD.3), and PB2 (PORTB.2) are for the external hardware interrupts INT0, INT1, and INT2, respectively.
3. Serial communication's USART has three interrupts, one for receive and two interrupts for transmit.
4. The SPI interrupts.
5. The ADC (analog-to-digital converter).

Interrupt Structure

- Upon reset all the interrupts are disabled(masked). The interrupts must be enabled by software in order for the microcontroller to respond to them.
- The **D7 bit(I-bit) of the SREG(Status Register)** is responsible for enabling and disabling interrupts globally . With a single instruction “CLI” (Clear Interrupt) we can make I=0 during the operation of a critical task.
- All interrupts are assigned individual enable bits which must be written logic one together with the Global Interrupt Enable bit in the Status Register in order to enable the interrupt

Interrupt Structure

- The lowest addresses in the program memory space are by default defined as the Reset and Interrupt Vectors. The complete list of vectors is shown below. The list also determines the priority levels of the different interrupts. The lower the address the higher is the priority level. RESET has the highest priority, and next is INT0 – the External Interrupt Request0. The Interrupt Vectors can be moved to the start of the Boot Flash section by setting the IVSEL bit in the General Interrupt Control Register (GICR).



Vector No.	Program Address[(2)]	Source	Interrupt Definition
1	$000[(1)]	RESET	External Pin, Power-on Reset, Brown-out Reset, Watchdog Reset, and JTAG AVR Reset
2	$002	INT0	External Interrupt Request 0
3	$004	INT1	External Interrupt Request 1
4	$006	INT2	External Interrupt Request 2
5	$008	TIMER2 COMP	Timer/Counter2 Compare Match
6	$00A	TIMER2 OVF	Timer/Counter2 Overflow
7	$00C	TIMER1 CAPT	Timer/Counter1 Capture Event
8	$00E	TIMER1 COMPA	Timer/Counter1 Compare Match A
9	$010	TIMER1 COMPB	Timer/Counter1 Compare Match B
10	$012	TIMER1 OVF	Timer/Counter1 Overflow
11	$014	TIMER0 COMP	Timer/Counter0 Compare Match
12	$016	TIMER0 OVF	Timer/Counter0 Overflow
13	$018	SPI, STC	Serial Transfer Complete
14	$01A	USART, RXC	USART, Rx Complete
15	$01C	USART, UDRE	USART Data Register Empty
16	$01E	USART, TXC	USART, Tx Complete
17	$020	ADC	ADC Conversion Complete
18	$022	EE_RDY	EEPROM Ready
19	$024	ANA_COMP	Analog Comparator
20	$026	TWI	Two-wire Serial Interface
21	$028	SPM_RDY	Store Program Memory Ready

Interrupt Structure

- The Reset Vector can also be moved to the start of the boot Flash section by programming the BOOTRST fuse.
- There are basically two types of interrupts.

1. The first type is triggered by an event that sets the Interrupt Flag. For these interrupts, the Program Counter is vectored to the actual Interrupt Vector in order to execute the interrupt handling routine, and hardware clears the corresponding Interrupt Flag. Interrupt Flags can also be cleared by writing a logic one to the flag bit position(s) to be cleared.
2. The second type of interrupts will trigger as long as the interrupt condition is present. These interrupts do not necessarily have Interrupt Flags. If the interrupt condition disappears before the interrupt is enabled, the interrupt will not be triggered.

<u>**External Interrupts:**</u> There are three external hardware interrupts in Atmega32.They are INT0,INT1 and INT2.

- The hardware interrupts must be enabled before they can take effect. These interrupts are controlled by the following registers:

1. GICR
2. GIFR
3. MCUCR
4. MCUCSR

Interrupt Structure

- **GICR(General Interrupt Control Register):** The General Interrupt Control Register controls the placement of the Interrupt Vector table.

Bit	7	6	5	4	3	2	1	0	
	INT1	INT0	INT2	–	–	–	IVSEL	IVCE	GICR
Read/Write	R/W	R/W	R/W	R	R	R	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

- **INT0:**When this bit is ‘1’ and global interrupt bit in SREG is ‘1’ the external interrupt INT0 is enabled. The ISC01 and ISC00 of MCUCR register control when the interrupt is to be activated(Rising or falling edge or level triggered)
- **INT1:**When this bit is ‘1’ and global interrupt bit in SREG is ‘1’ the external interrupt INT1 is enabled. The ISC11 and ISC10 of MCUCR register control when the interrupt is to be activated(Rising or falling edge or level triggered)
- **INT2:**When this bit is ‘1’ and global interrupt bit in SREG is ‘1’ the external interrupt INT2 is enabled. The ISC2 of MCUCSR register control when the interrupt is to be activated(Rising or falling edge)

Interrupt Structure

- INT0 and INT1 can be edge triggered or level triggered but INT2 can only be edge triggered.
- The MCUCR and MCUCSR registers decide the triggering options of the external hardware interrupts INT0,INT1 and INT2.

Bit 1 – IVSEL: Interrupt Vector Select

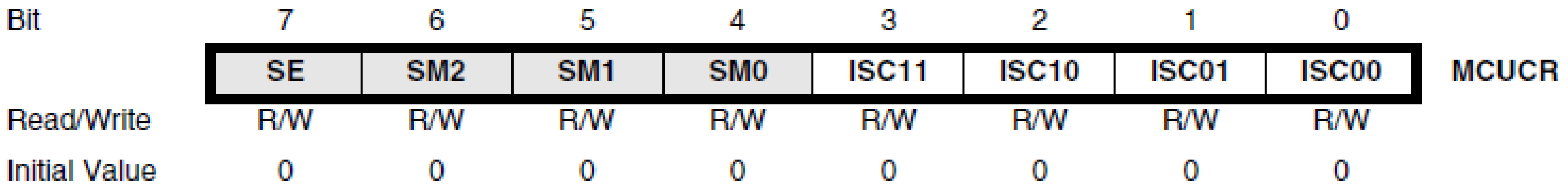
- When the IVSEL bit is cleared (zero), the Interrupt Vectors are placed at the start of the Flash memory. When this bit is set (one), the interrupt vectors are moved to the beginning of the Boot Loader section of the Flash.

Bit 0 – IVCE: Interrupt Vector Change Enable

- The IVCE bit must be written to logic one to enable change of the IVSEL bit

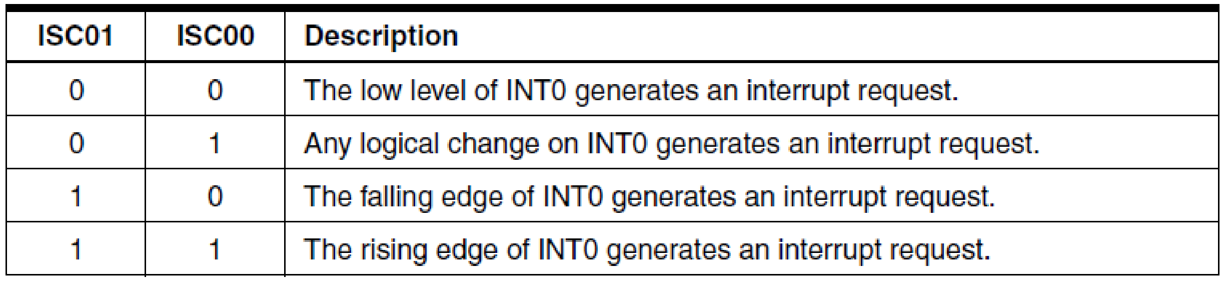
Interrupt Structure

- **MCUCR(MCU Control Register):**The MCU Control Register contains control bits for interrupt sense control and general MCU functions.



Bit	7	6	5	4	3	2	1	0	
	SE	SM2	SM1	SM0	ISC11	ISC10	ISC01	ISC00	MCUCR
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

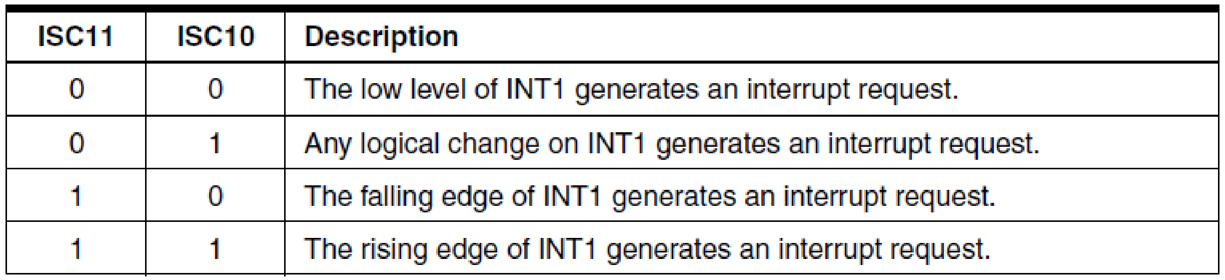
- **ISC01 and ISC00(Interrupt Sense Control Bits):**These bits define whether the signal should be level or edge triggered at INT0 pin.



ISC01	ISC00	Description
0	0	The low level of INT0 generates an interrupt request.
0	1	Any logical change on INT0 generates an interrupt request.
1	0	The falling edge of INT0 generates an interrupt request.
1	1	The rising edge of INT0 generates an interrupt request.

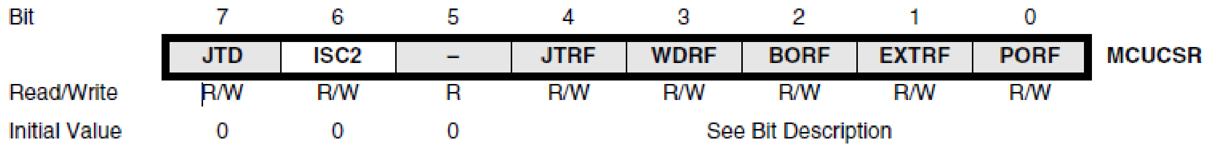
Interrupt Structure

- ISC11 and ISC10: These bits define whether the signal should be level or edge triggered at INT1 pin.



ISC11	ISC10	Description
0	0	The low level of INT1 generates an interrupt request.
0	1	Any logical change on INT1 generates an interrupt request.
1	0	The falling edge of INT1 generates an interrupt request.
1	1	The rising edge of INT1 generates an interrupt request.

- **MCUCSR(MCU Control and Status Register):**



Bit	7	6	5	4	3	2	1	0	
	JTD	ISC2	–	JTRF	WDRF	BORF	EXTRF	PORF	MCUCSR
Read/Write	R/W	R/W	R	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	See Bit Description					

ISC2:This bit controls the INT2 interrupt trigger condition

ISC2 =0:The interrupt is detected on falling edge

ISC2 =1: The interrupt is detected on rising edge

Interrupt Structure

- **General Interrupt Flag Register(GIFR):**

Bit	7	6	5	4	3	2	1	0	
	INTF1	INTF0	INTF2	–	–	–	–	–	GIFR
Read/Write	R/W	R/W	R/W	R	R	R	R	R	
Initial Value	0	0	0	0	0	0	0	0	

Bit 7 – INTF1: External Interrupt Flag 1

When an edge or logic change on the INT1 pin triggers an interrupt request, INTF1 becomes set (one). If the I-bit in SREG and the INT1 bit in GICR are set (one), the MCU will jump to the corresponding Interrupt Vector. The flag is cleared when the interrupt routine is executed.

Bit 6 – INTF0: External Interrupt Flag 0

When an edge or logic change on the INT0 pin triggers an interrupt request, INTF0 becomes set(one). If the I-bit in SREG and the INT0 bit in GICR are set (one), the MCU will jump to the corresponding interrupt vector. The flag is cleared when the interrupt routine is executed.

Bit 5 – INTF2: External Interrupt Flag 2

- When an event on the INT2 pin triggers an interrupt request, INTF2 becomes set (one). If the I-bit in SREG and the INT2 bit in GICR are set (one), the MCU will jump to the corresponding Interrupt Vector. The flag is cleared when the interrupt routine is executed

Interrupt Structure(CHAT GPT)

The ATMega16 microcontroller has a flexible interrupt system that allows the user to handle interrupt requests from various peripherals and external devices.
The interrupt structure of the ATMega16 consists of the following components:
1.Interrupt Vector Table: This is a table of memory addresses that correspond to each of the interrupt sources. When an interrupt occurs, the microcontroller jumps to the corresponding interrupt vector to handle the interrupt.
2.Interrupt Request (IRQ) Pin: The microcontroller has several interrupt request (IRQ) pins that can be used to generate interrupt requests from external devices.
3.Interrupt Mask Register (IMR): This register allows the user to enable or disable specific interrupt sources.

Interrupt Structure(CHAT GPT)

4.Interrupt Flag Register (IFR): This register holds the interrupt flags, which indicate the status of the interrupt sources.
5.Interrupt Service Routine (ISR): This is a function that handles the interrupt request. The ISR is executed when an interrupt occurs, and it performs the necessary operations to process the interrupt.
To use the interrupt system of the ATMega16, the user must configure the interrupt sources by setting the appropriate bits in the IMR and IFR registers. The user must also define the ISR functions that correspond to each interrupt source. When an interrupt occurs, the microcontroller jumps to the corresponding ISR to handle the interrupt. After the ISR is executed, the microcontroller returns to the main program.

Architecture(Chat GPT)

- The architecture of an AVR microcontroller is based on the Harvard architecture, which separates the data memory and the program memory. The AVR architecture consists of the following components:

1. Central Processing Unit (CPU): The CPU is responsible for executing instructions, performing arithmetic and logic operations, and managing the flow of the program.
2. Program Memory: The program memory stores the executable instructions that the CPU executes. It is usually flash memory, which can be reprogrammed multiple times.
3. Data Memory: The data memory stores the variables and data used by the program. It includes both static and dynamic RAM.
4. Input/Output (I/O) Ports: The AVR microcontroller has a number of I/O ports that can be used to interface with external devices, such as sensors and actuators.
5. Timers/Counters: The AVR microcontroller has several timers and counters that can be used for timing and counting purposes.
6. Analog-to-Digital Converter (ADC): The ADC converts analog signals to digital signals, which can be processed by the CPU.
7. Serial Communication Interfaces: The AVR microcontroller has several serial communication interfaces, such as USART, SPI, and I2C, that can be used for communication with other devices.
8. Interrupt Controller: The interrupt controller handles interrupt requests from external devices and peripherals, allowing the CPU to respond to events in a timely manner.
9. Clock Generator: The clock generator generates the clock signal used by the microcontroller to synchronize its operations.

- Overall, the architecture of an AVR microcontroller is designed to be simple, flexible, and efficient, making it a popular choice for a wide range of applications.

PIN DESCRIPTION OF ATMEGA16

The ATMega16 is a popular AVR microcontroller that has the following pin description:

1. Power Pins: The ATMega16 has two power pins: VCC (positive power supply, typically 5V) and GND (ground).
2. I/O Pins: The ATMega16 has up to 31 I/O pins that can be used as inputs or outputs. These pins are labeled as PORT A, PORT B, PORT C, and PORT D.
3. Reset Pin: The reset pin (labeled RESET) is used to reset the microcontroller and is typically connected to a push-button switch.
4. Oscillator Pins: The ATMega16 has two oscillator pins (XTAL1 and XTAL2) that are used to connect an external crystal oscillator. This oscillator is used to generate the clock signal that drives the microcontroller.

PIN DESCRIPTION OF ATMEGA16

5. Analog Input Pins: The ATMega16 has up to 6 analog input pins (labeled ADC0 to ADC5) that can be used with the built-in analog-to-digital converter (ADC).
6. Serial Communication Pins: The ATMega16 has several serial communication pins that can be used for communication with other devices, including USART (Universal Synchronous/Asynchronous Receiver/Transmitter), SPI (Serial Peripheral Interface), and I2C (Inter-Integrated Circuit).
7. Interrupt Pins: The ATMega16 has several interrupt pins that can be used to handle interrupt requests from external devices and peripherals.
8. Other Pins: The ATMega16 has several other pins, including a programming pin (labeled MISO), a clock pin (labeled SCK), and a data pin (labeled MOSI), which are used for programming the microcontroller using an in-system programmer.