MICROCONTROLLERS AND APPLICATIONS

TRUNG-KHANH LE
ltkhanh@fetel.hcmus.edu.vn

GRADING

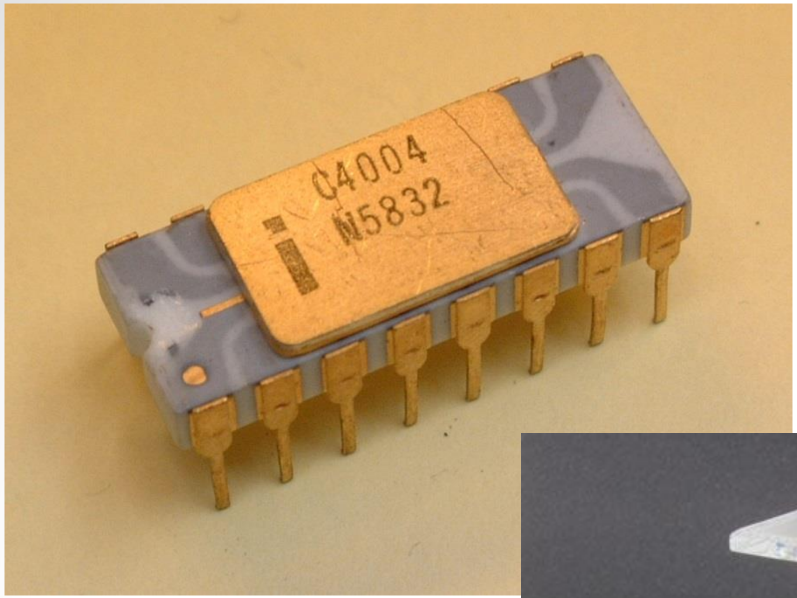
- No final exam!
- Class attendance is not required.
- Weekly project reports:
  - 10 projects total.
  - At least 7 reported projects.
  - 1 point/project.

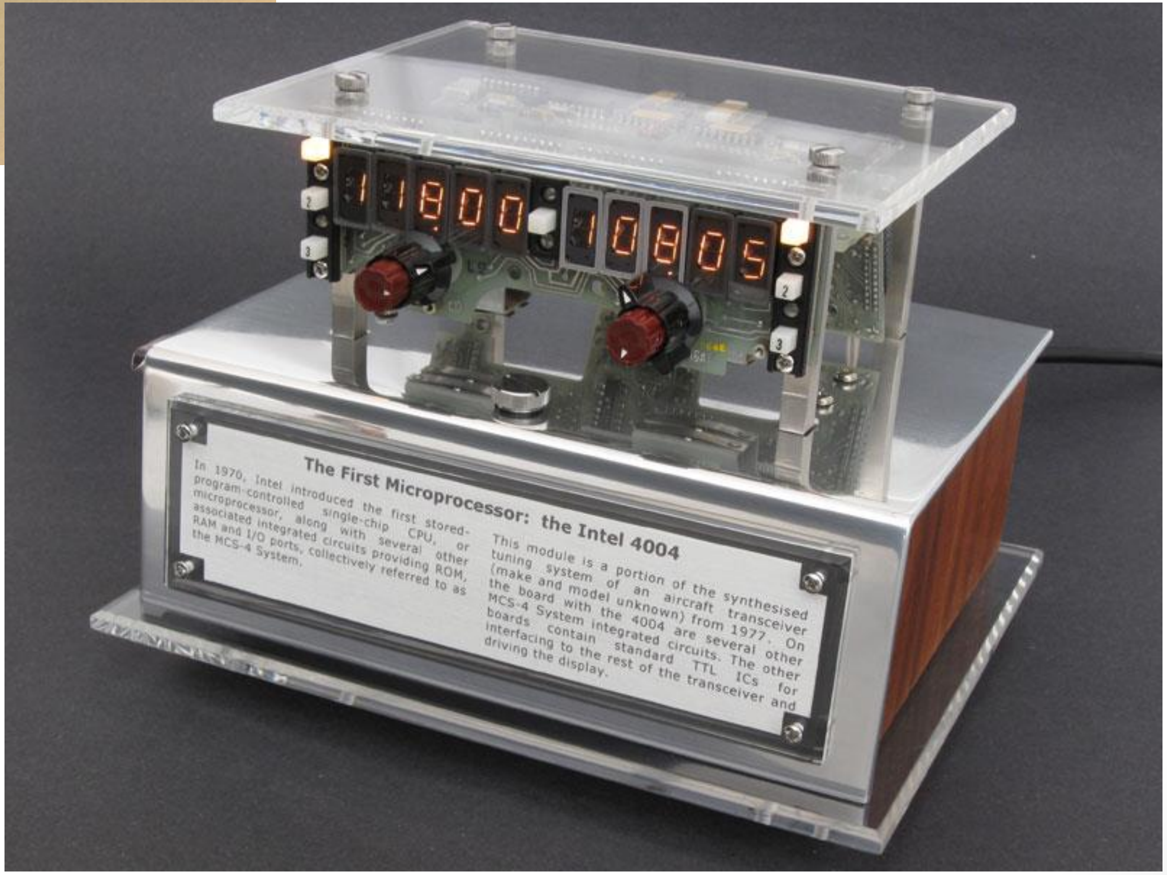
CONTENT

INTRODUCTION

HISTORY

- 1950s - The beginning of the digital era and electronic computing
- 1969 – Intel is a small startup company in Santa Clara with 12 employees
  - Fairchild, Motorola are large semiconductor companies Hp and Busicom make calculators
- 1971 – Intel makes first microprocessor the 4-bit 4004 series for Busicom calculators
- 1972 – Intel makes the 8008 series, an 8-bit microprocessor,
  - ATARI is a startup company
  - Creates a gaming console and releases PONG

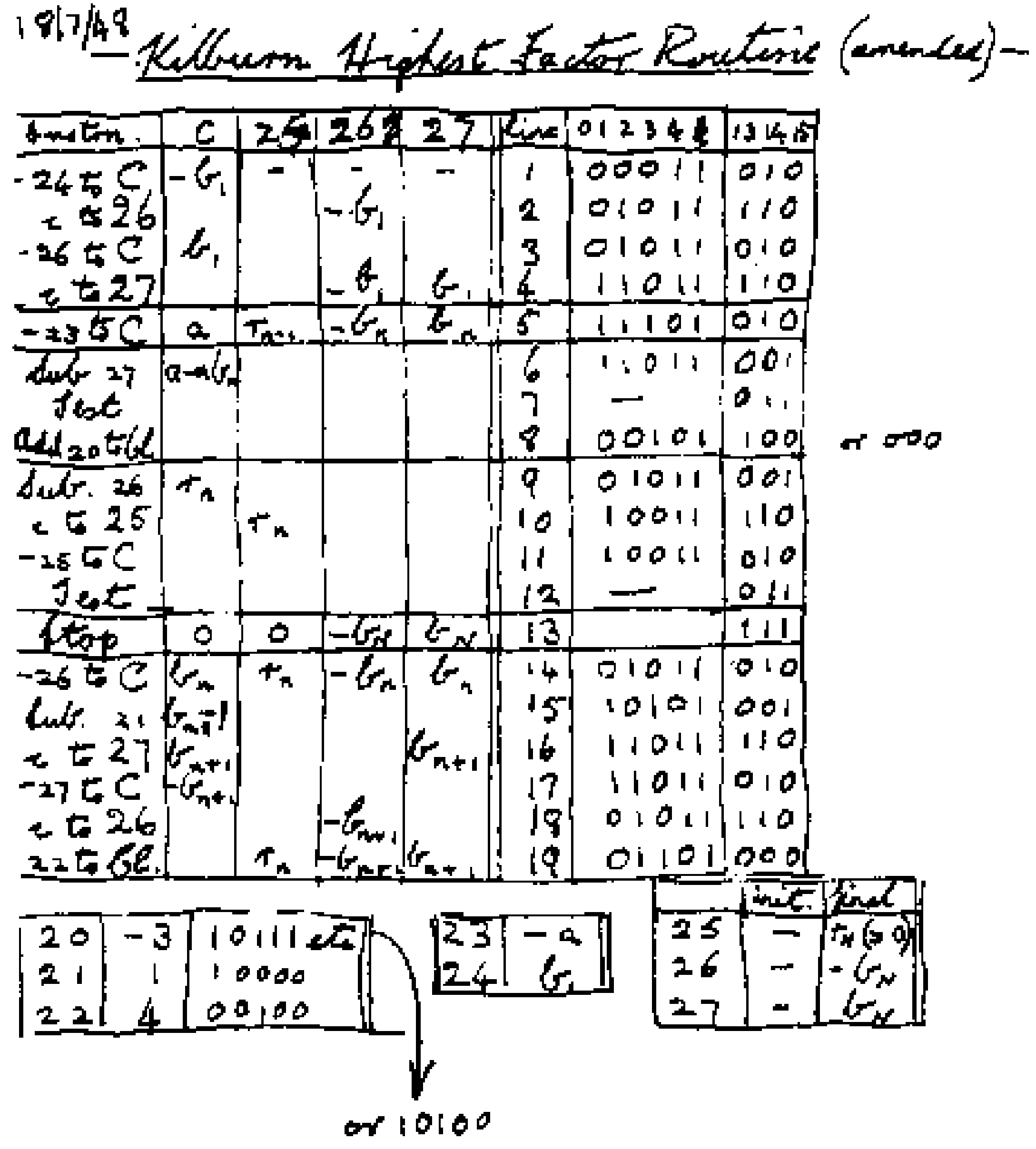




The First Microprocessor: the Intel 4004

In 1970, Intel introduced the first stored-program-controlled single-chip CPU, or microprocessor, along with several other associated integrated circuits providing ROM, RAM and I/O ports, collectively referred to as the MCS-4 System.

This module is a portion of the synthesised tuning system of an aircraft transceiver (make and model unknown) from 1977. On the board with the 4004 are several other MCS-4 System integrated circuits. The other boards contain standard TTL ICs for interfacing to the rest of the transceiver and driving the display.



18/7/48 — Kilburn Highest Factor Routine (amended) —

Instn.	C	25	26	27	Line	01234	13 14 15
-24 to C	$-b_1$	–	–	–	1	00011	010
c to 26			$-b_1$		2	01011	110
-26 to C	b_1				3	01011	010
c to 27			$-b_1$	b_1	4	11011	110
-23 to C	a	r_{n-1}	$-b_n$	b_n	5	11101	010
Subr 27	$a-nb_n$				6	11011	001
Test					7	—	011
Add 20 to Cl					8	00101	100
Subr. 26	r_n				9	01011	001
c to 25		r_n			10	10011	110
-25 to C					11	10011	010
Test					12	—	011
Stop	0	0	$-b_N$	b_N	13		111
-26 to C	b_n	r_n	$-b_n$	b_n	14	01011	010
Subr. 21	b_{n+1}				15	10101	001
c to 27	b_{n+1}			b_{n+1}	16	11011	110
-27 to C	$-b_{n+1}$				17	11011	010
c to 26			$-b_{n+1}$		18	01011	110
22 to Cl.		r_n	$-b_{n+1}$	b_{n+1}	19	01101	000

or 000

20	-3	10111 etc
21	1	10000
22	4	00100

or 10100

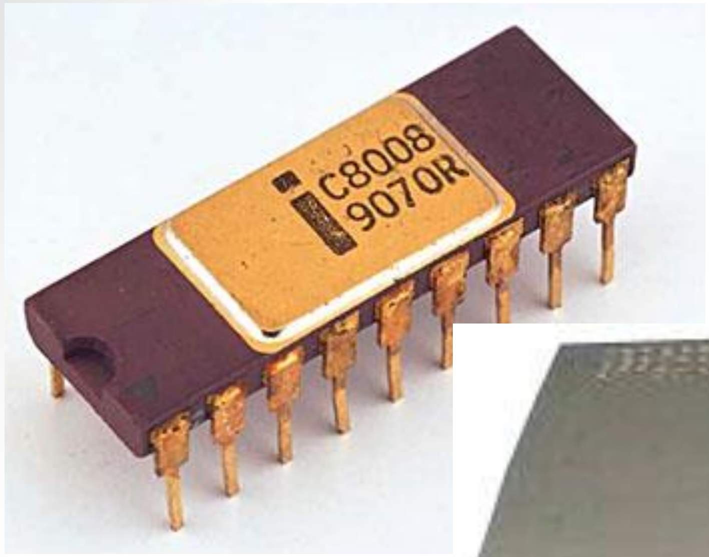
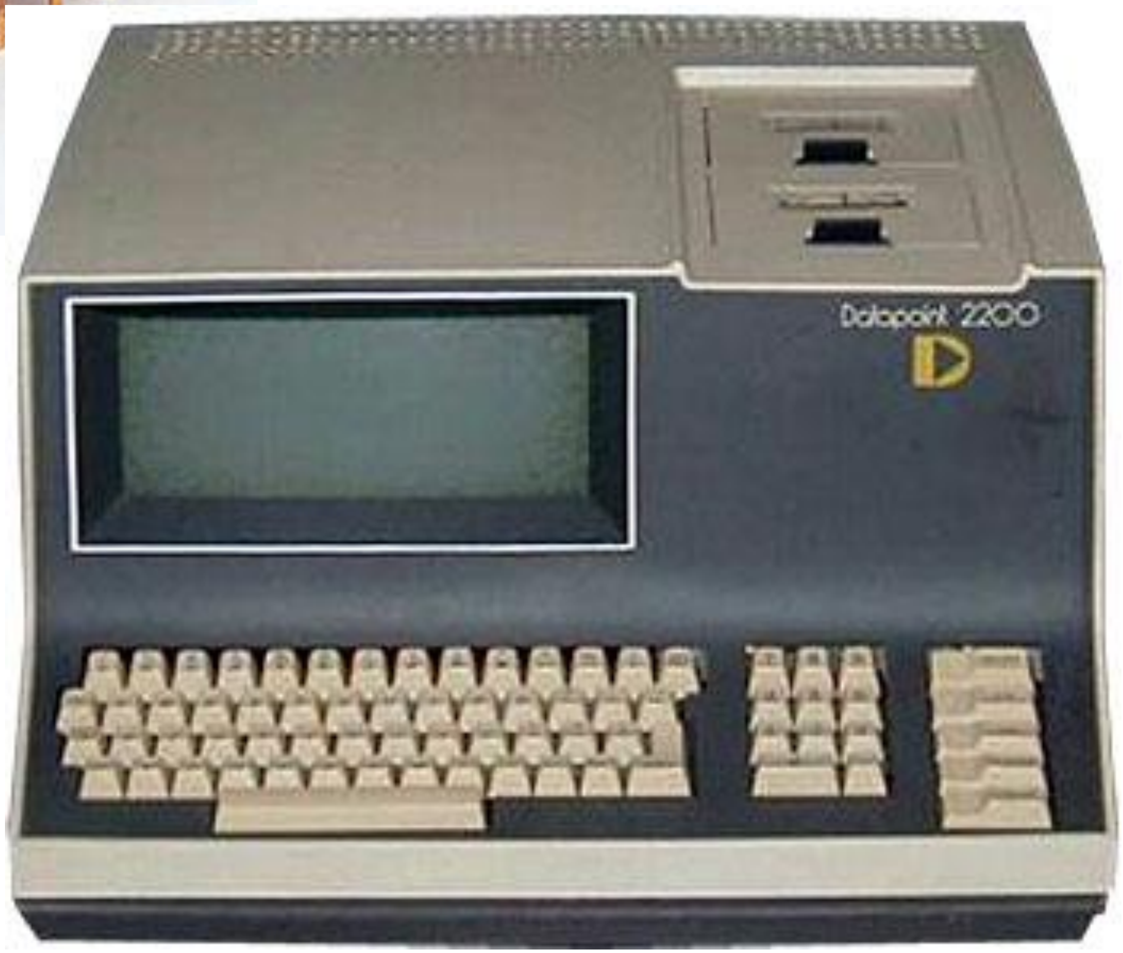
23	$-a$
24	b_1

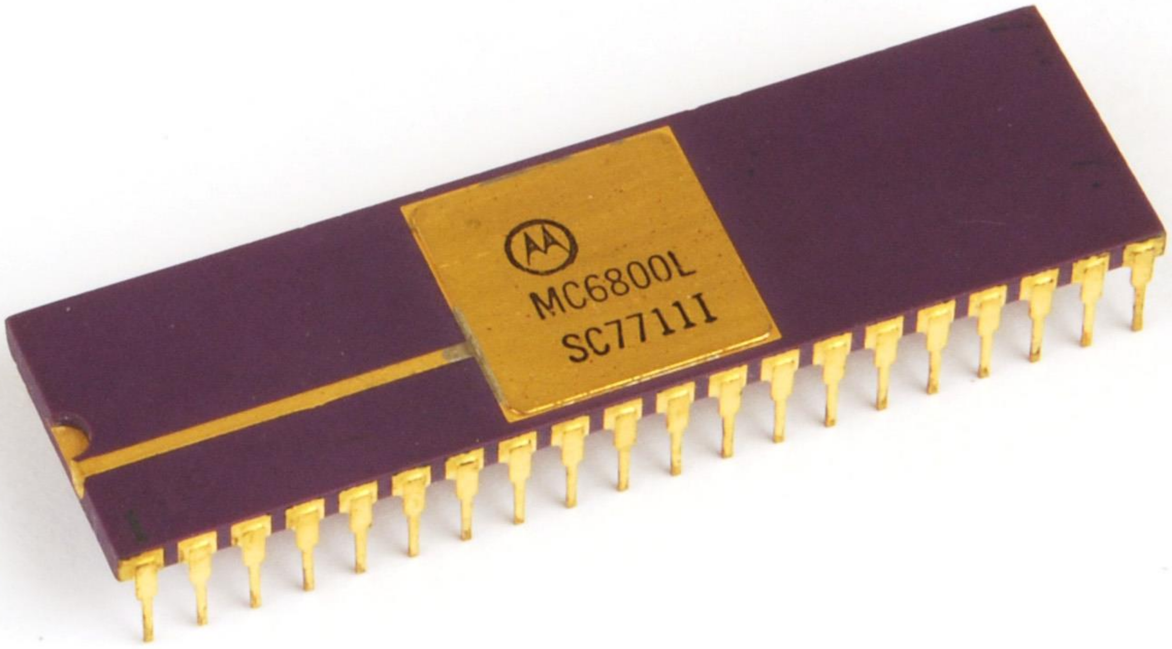
	init.	final
25	—	r_N (=0)
26	—	$-b_N$
27	—	b_N

The First Computer Program

HISTORY

- 1974 – the first real useful 8-bit microprocessor is released by Intel – the 8080
  - Motorola introduces the 6800 series
  - Zilog has the Z80
- 1975 – GM and Ford begin to put microcontrollers in cars
  - Many cars today have over 100 microcontrollers
  - TI gets into the microprocessor business with calculators and digital watches
- 1976 – Intel introduces 8748 MCU, the first MCU of the MCS-48 family. It is created from 17000 transistors.
- 1977 – Apple II is released using MOS 6502 (similar to motorola 6800). Apple II dominated from 1977 to 1983
- 1978 – Intel introduces the first 16-bit processor, the 8086
  - Motorola follows with the 68000 which is ultimately used in the first Apple Macintosh


C8008
9070R

Datapoint 2200


MC6800L
SC7711I

C8748-8
1742C
© INTEL '76
20

HISTORY

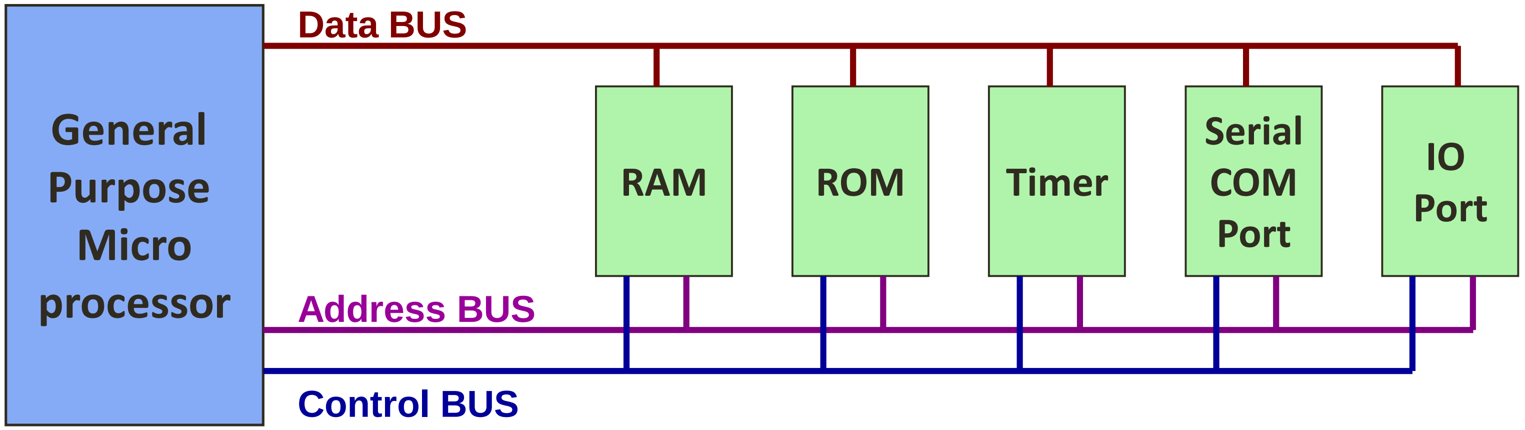
- 1980 – Intel introduces 8051 MCU and MCS-51 family.
- 1981 – IBM enters the PC making market and uses the Intel 8088 – proliferation of the home computer
- 1982-1985 – Intel introduces the 32-bit 80286 and 80386
- 1989 – 80486 is being used in PC’s, able to run Microsoft Windows, 8-bit MCU called PIC is introduces in 1989 by Microchip Technology Corporation.
- 1992 – Apple, IBM and Motorola begin to make PowerMac and PowerPC’s using Motorola chips
- 1993 – Pentium chip is released

Transistors per Processor

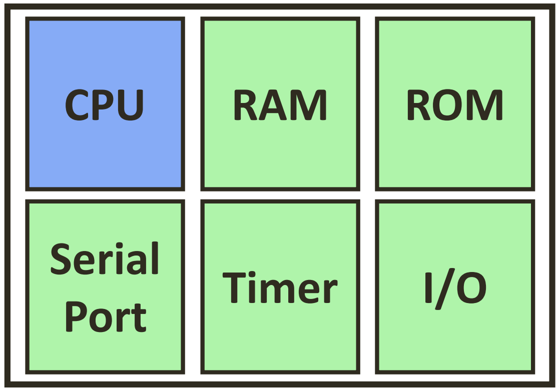
	Year of introduction	Transistors
4004	1971	2,250
8008	1972	2,500
8080	1974	5,000
8086	1978	29,000
286	1982	120,000
386™ processor	1985	275,000
486™ DX processor	1989	1,180,000
Pentium® processor	1993	3,100,000
Pentium II processor	1997	7,500,000
Pentium III processor	1999	24,000,000
Pentium 4 processor	2000	42,000,000

MICROPROCESSOR or MICROCONTROLLER?

- General Purpose Microprocessors



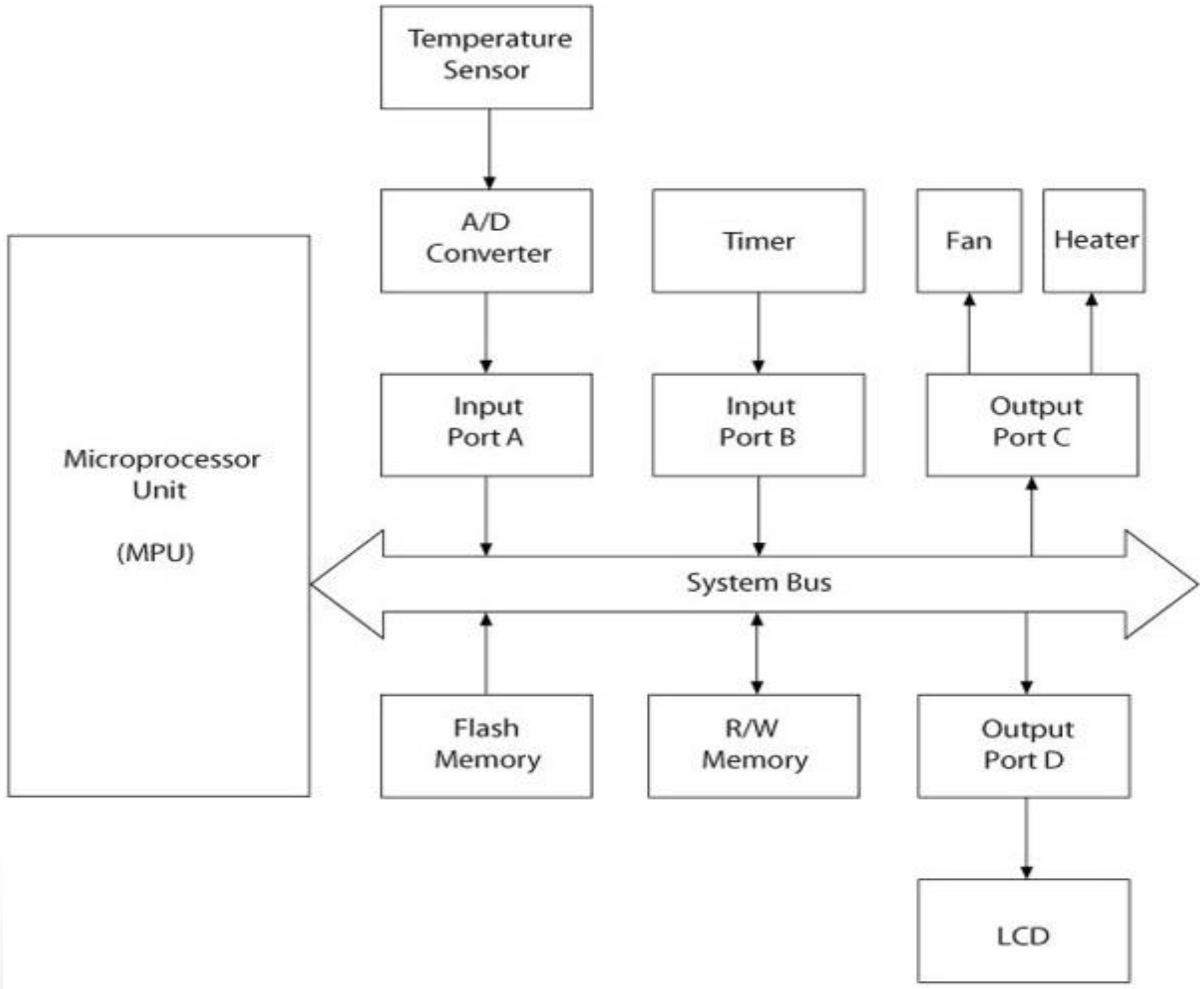
- Microcontrollers



MPU-Based Systems

- System hardware
  - Discrete components
    - Microprocessor, Memory, and I/O
  - Components connected by buses
    - Address, Data, and Control
- System software
  - Group of programs that monitors the functions of the entire system

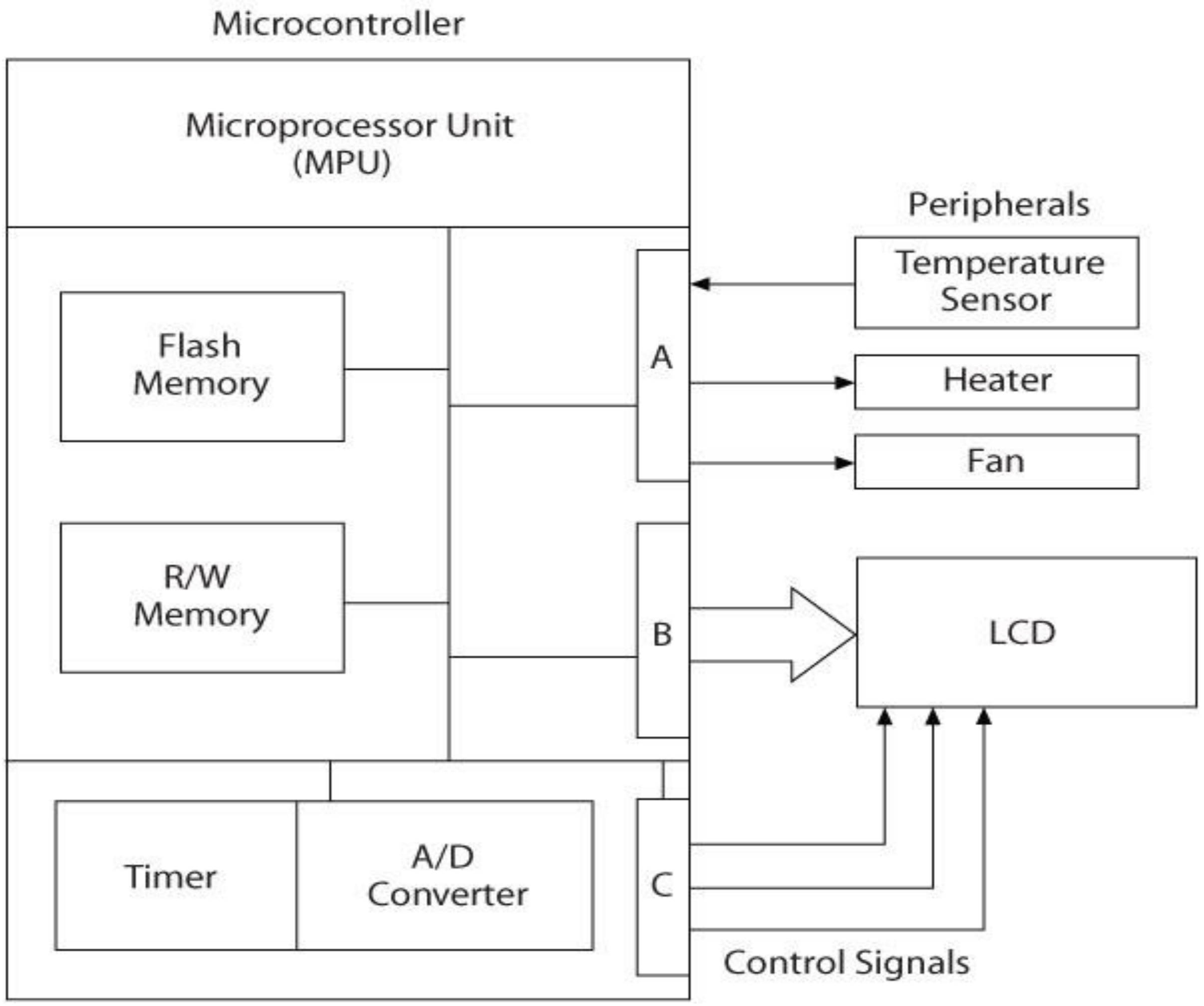
MPU-Based System



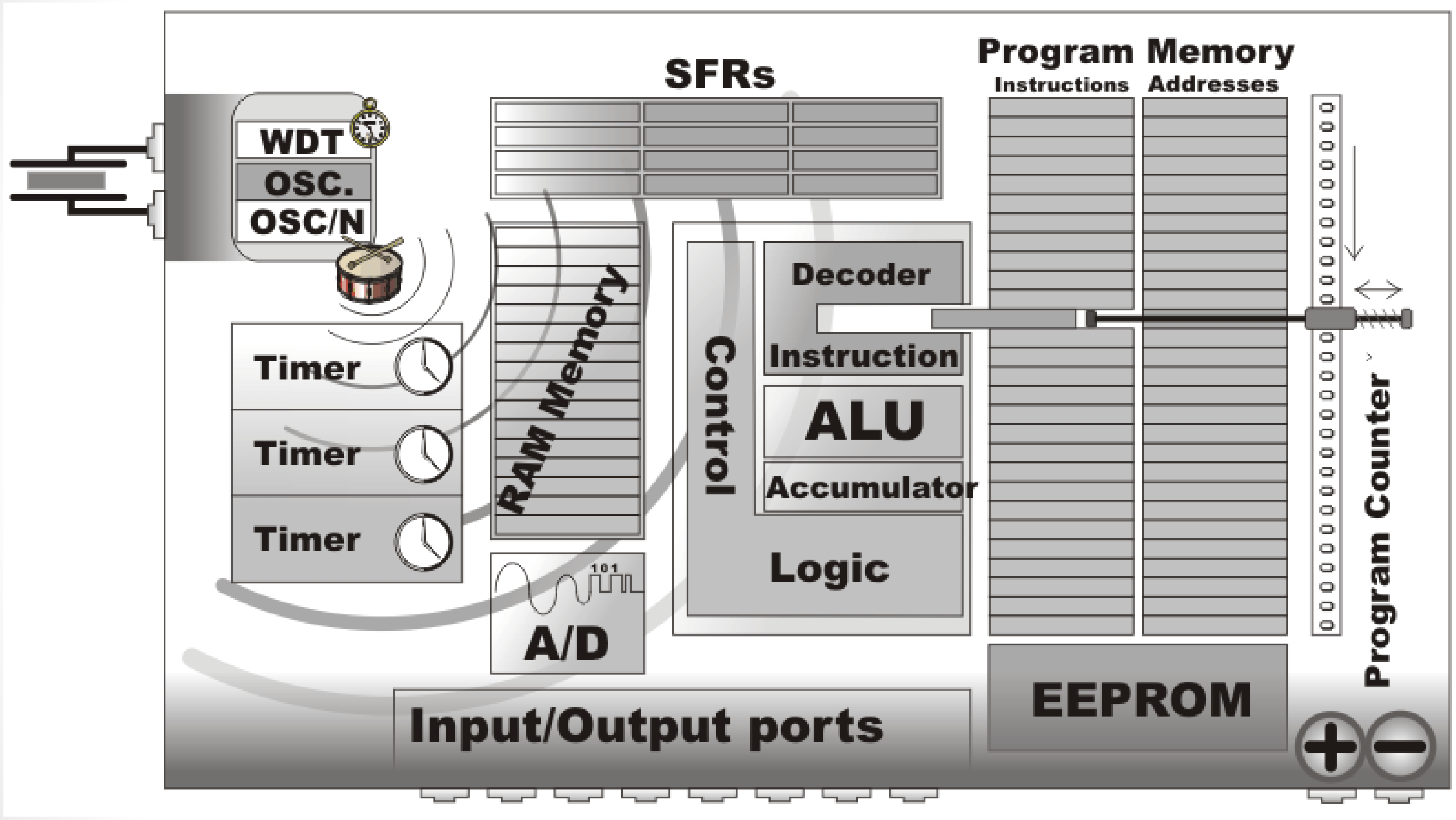
MCU-Based Systems

- Microprocessor, memory, I/O ports, and support devices on a single chip
- Buses generally not available to a system designer
- I/O ports generally multiplexed and can be programmed to perform different functions

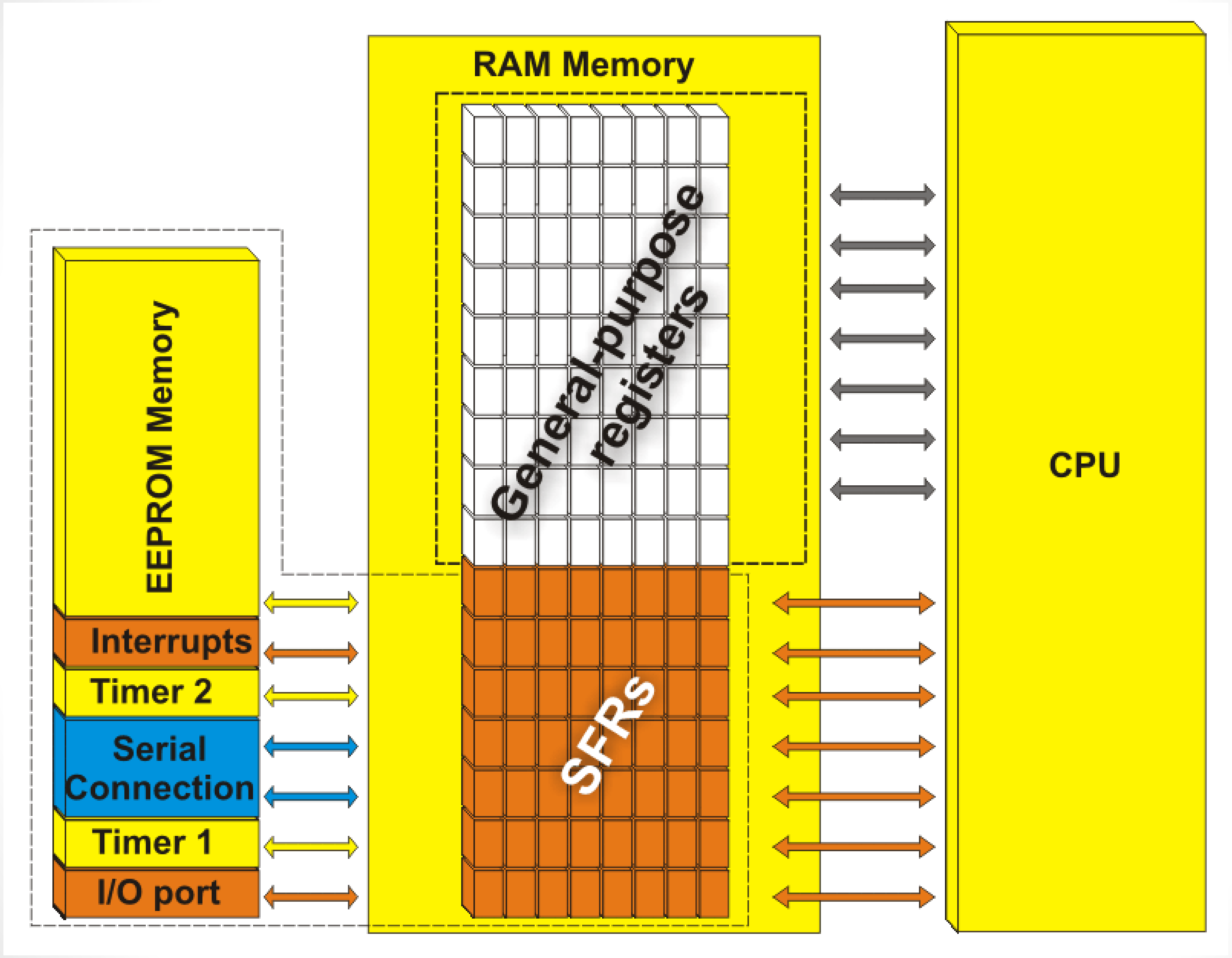
MCU-Based Systems



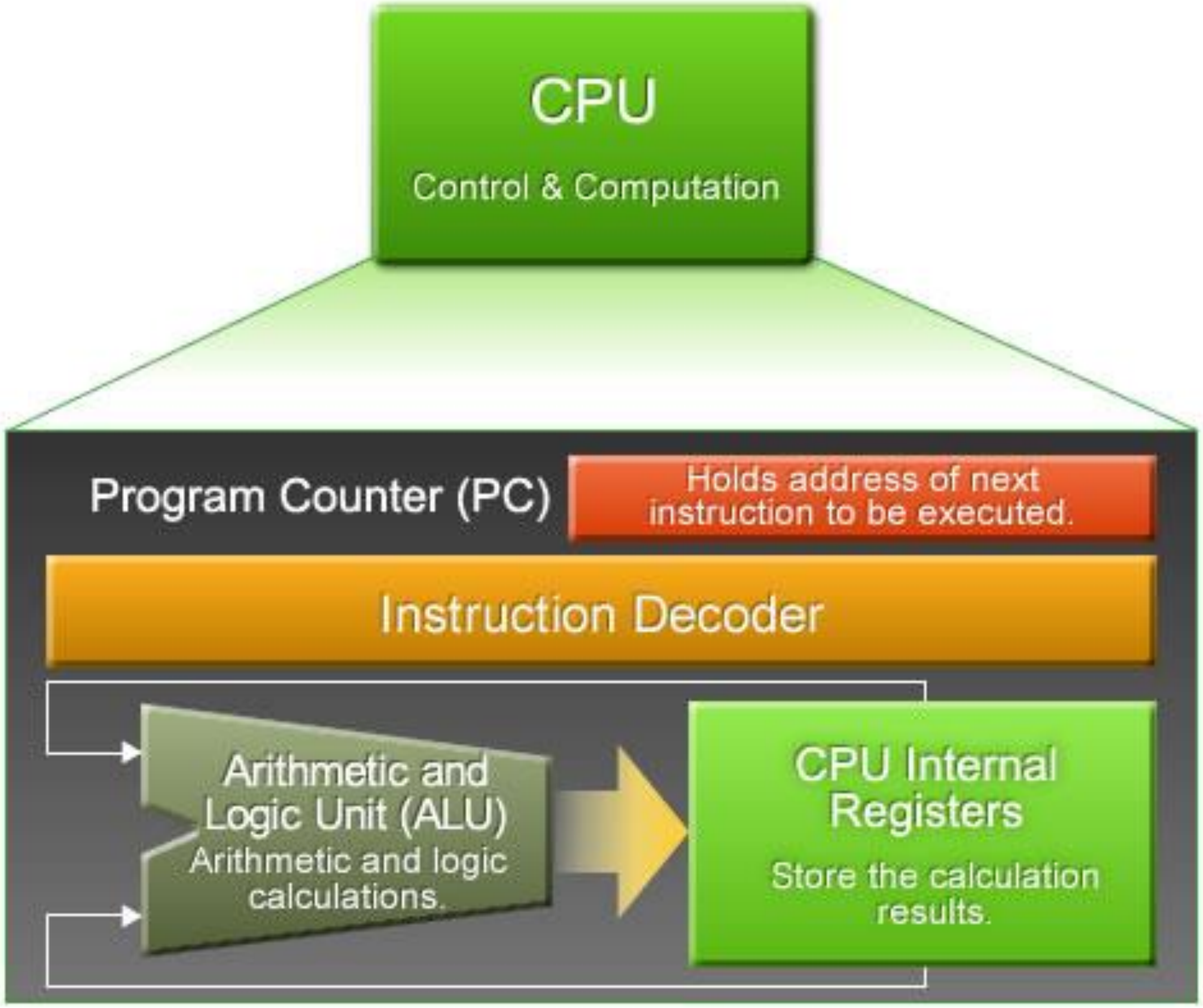
MICROCONTROLLER STRUCTURE



Micro-Controller Structure



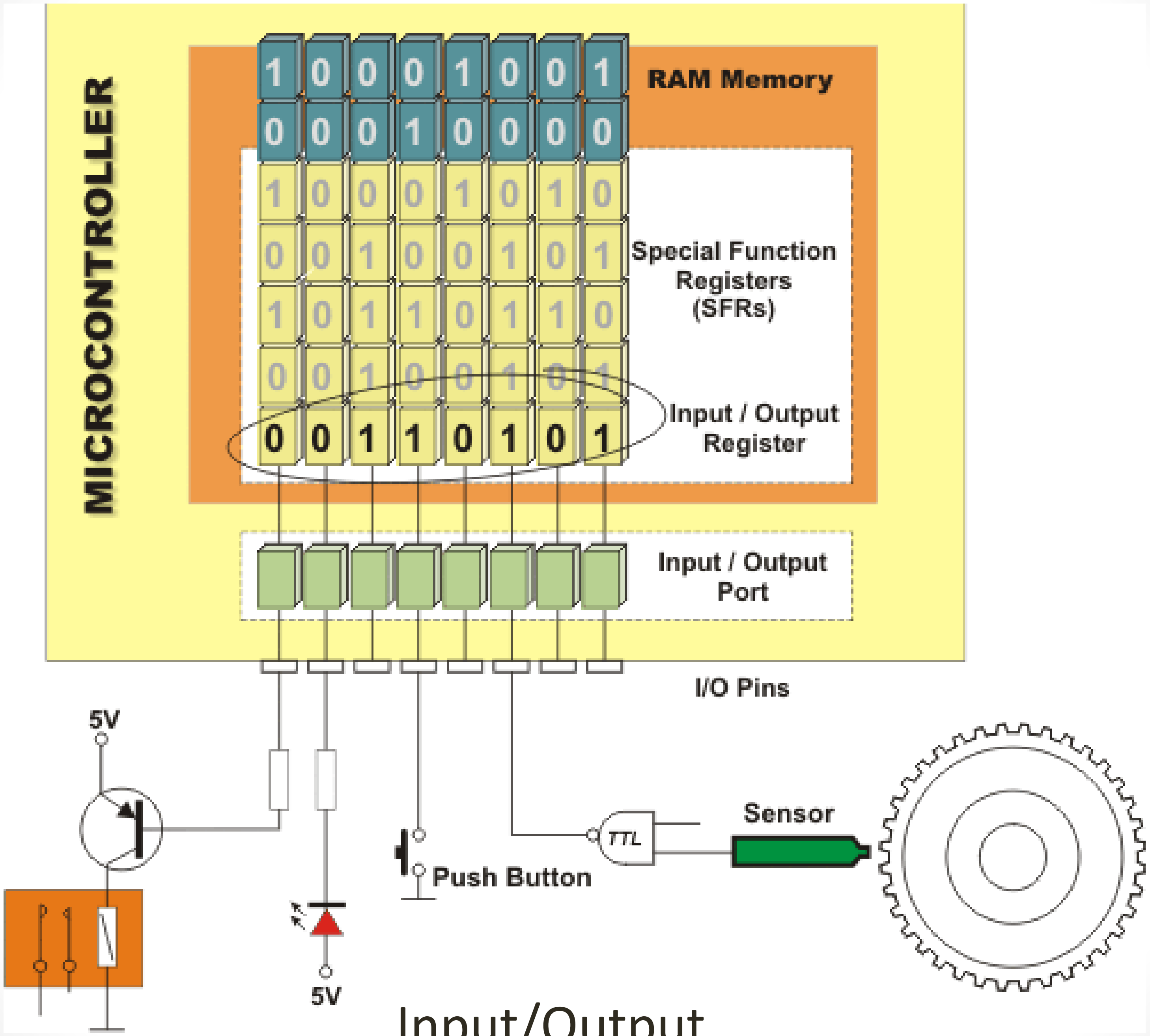
Data traffic


CPU
Control & Computation
Program Counter (PC)
Holds address of next instruction to be executed.
Instruction Decoder
Arithmetic and Logic Unit (ALU)
Arithmetic and logic calculations.
CPU Internal Registers
Store the calculation results.

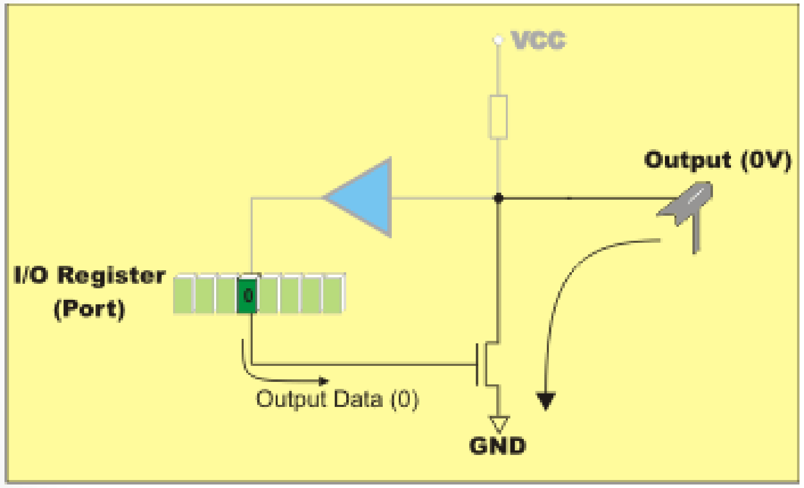
CPU structure

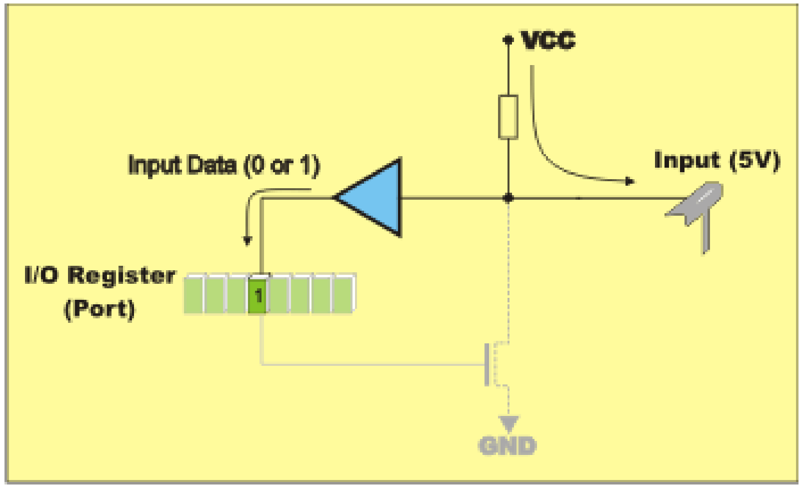
Operation of the CPU
Let calculate 3+4=?

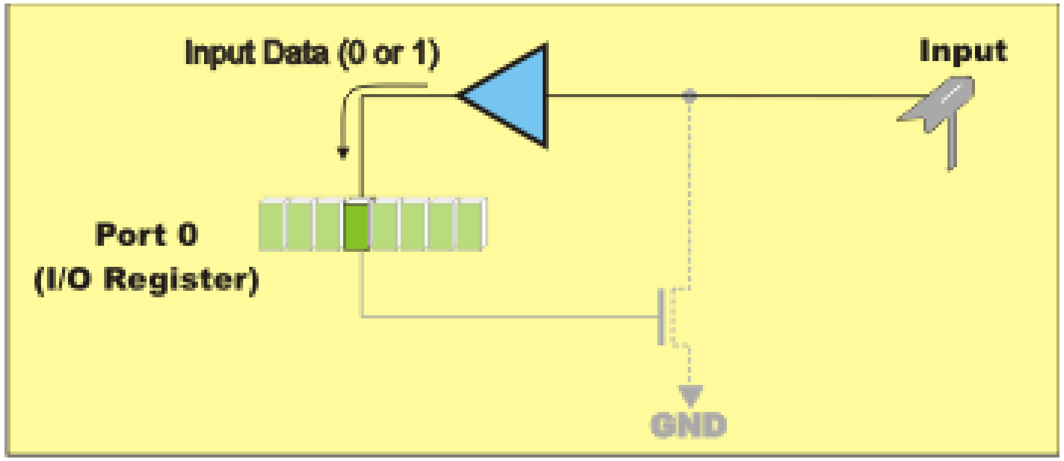
Address	Instruction (a binary code value identifying the action to be taken)
0000	Read the value at memory address 0100, and store it in Register 1.
0001	Read the value at memory address 0101, and store it in Register 2.
0002	Add the value in Register 2 to the value in Register 1, and save the result in Register 1.
~	
Address	**Data**
0100	3
0101	4

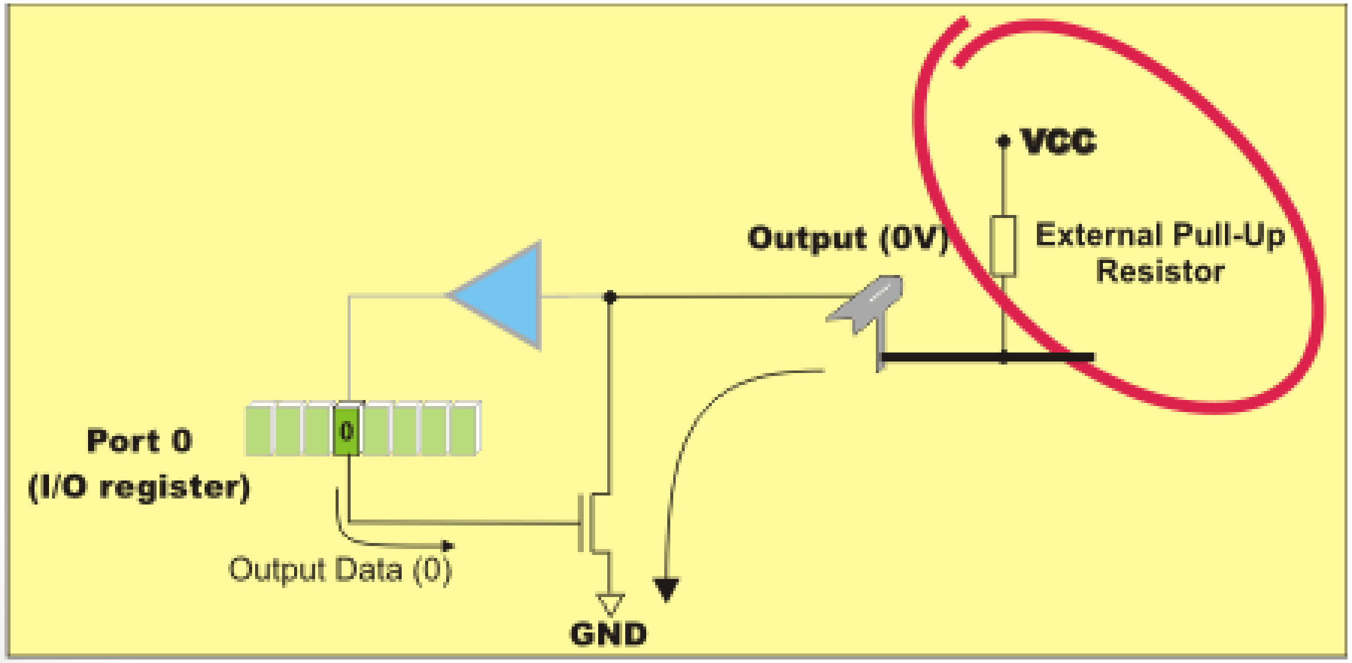


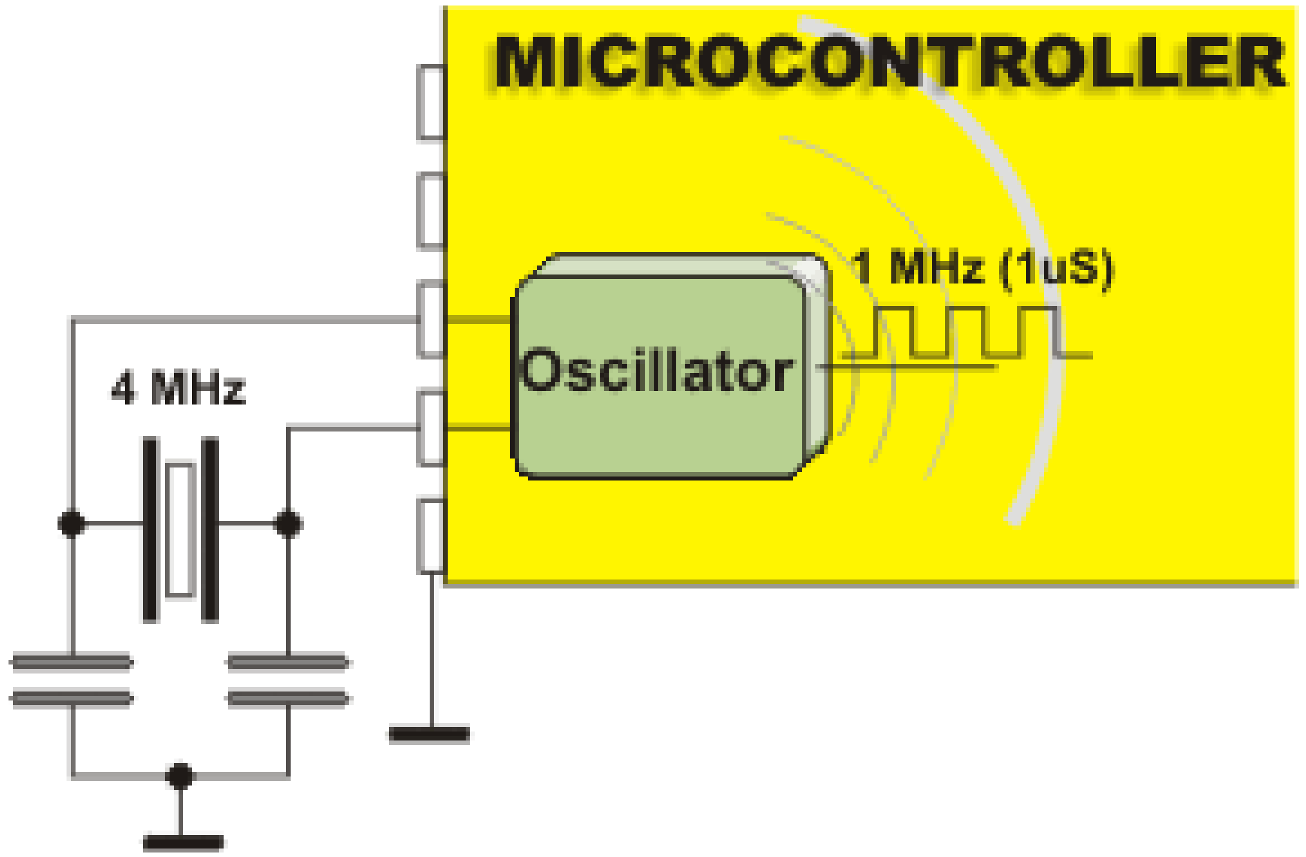
Input/Output


VCC
Output (0V)
I/O Register
(Port)
0
Output Data (0)
GND

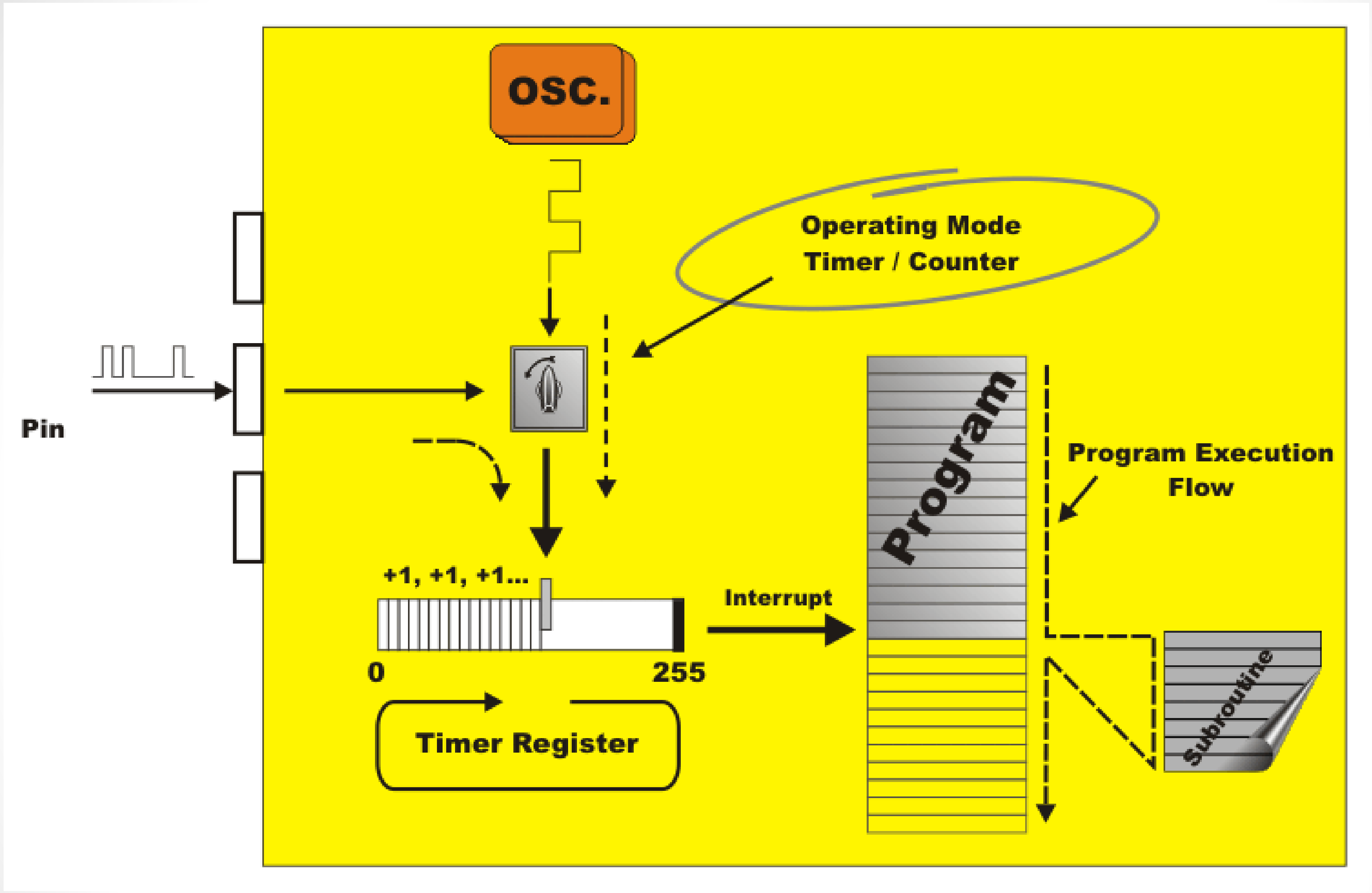

VCC
Input Data (0 or 1)
Input (5V)
I/O Register
(Port)
1
GND


Input Data (0 or 1)
Input
Port 0
(I/O Register)
GND


VCC
Output (0V)
External Pull-Up
Resistor
0
Port 0
(I/O register)
Output Data (0)
GND

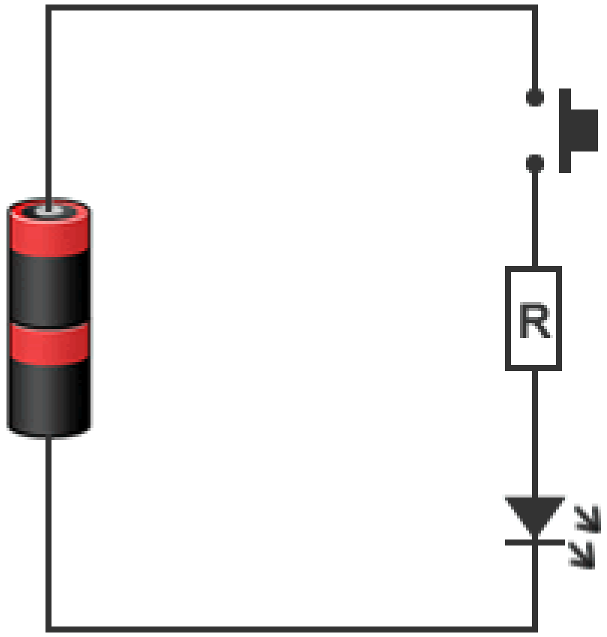


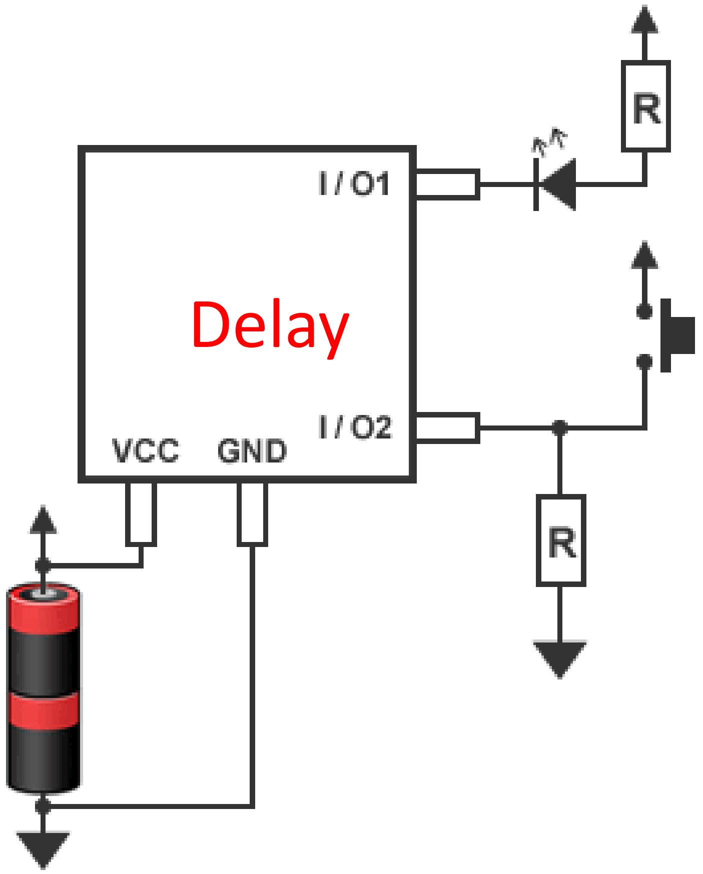
Oscillator



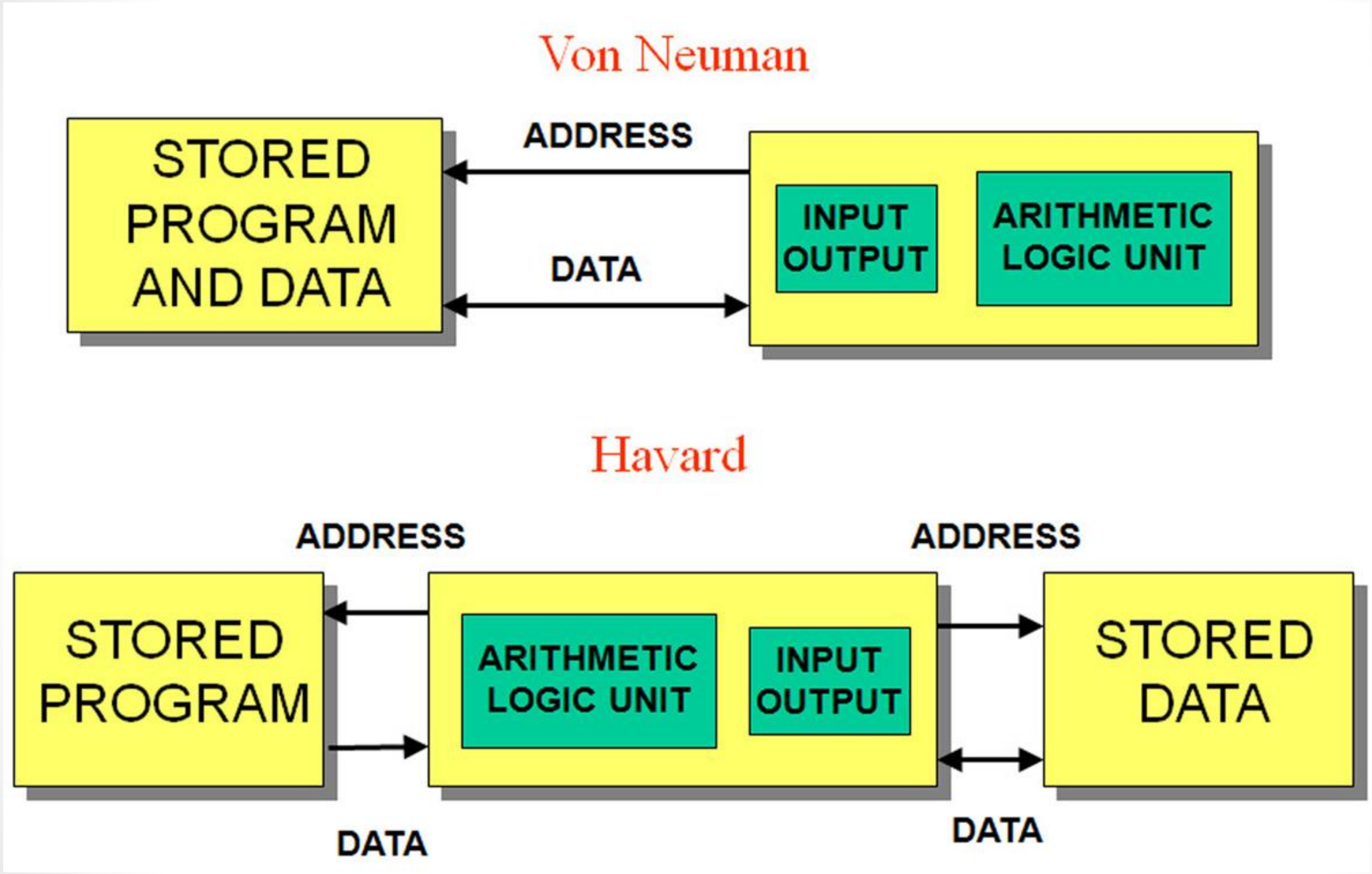
Timer/Counter

Why Do We Use MCUs?





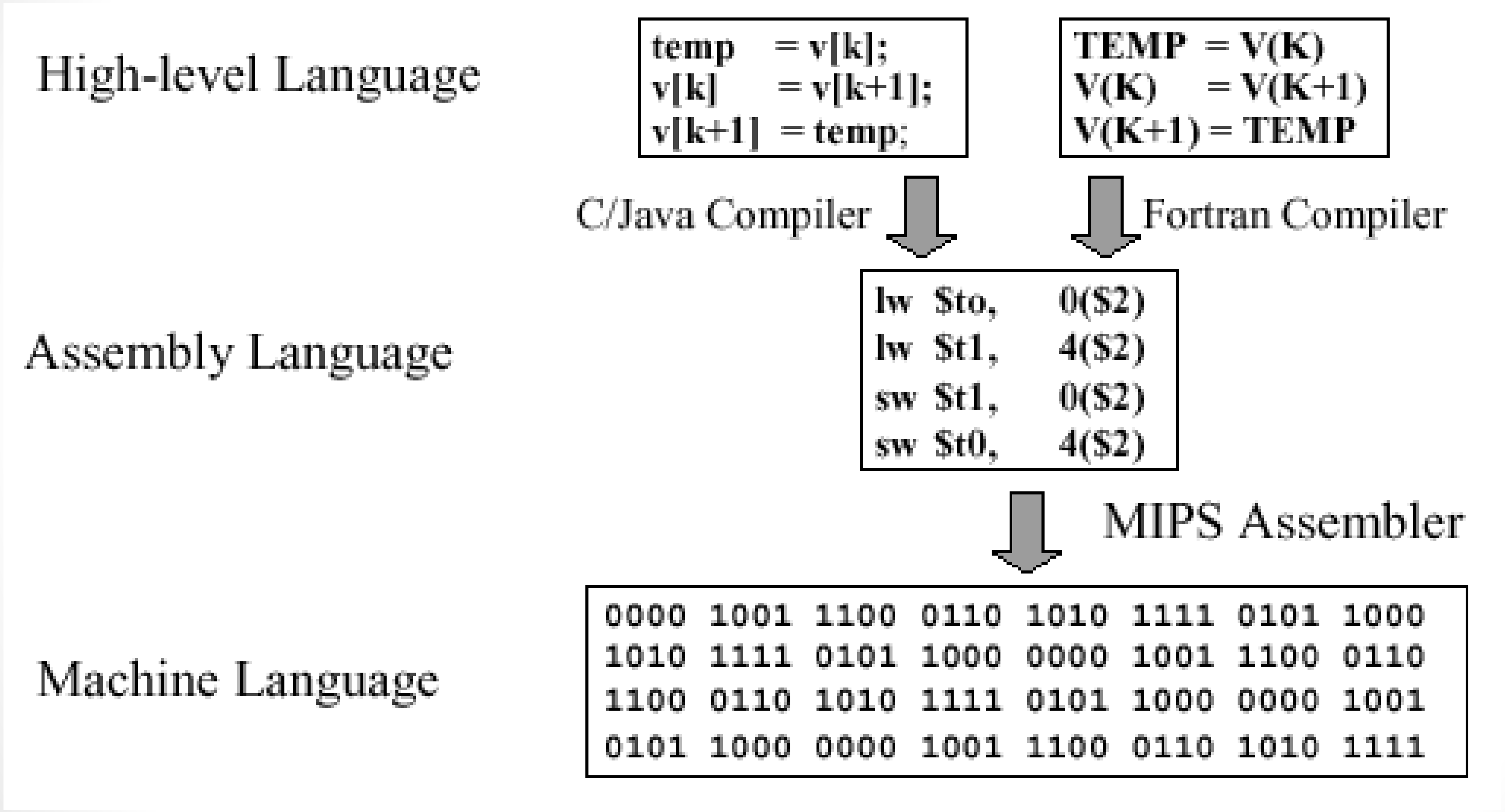
VON NEUMANN vs. HARVARD ARCHITECTURE


Von Neuman
STORED PROGRAM AND DATA
ADDRESS
DATA
INPUT OUTPUT
ARITHMETIC LOGIC UNIT
Havard
ADDRESS
ADDRESS
STORED PROGRAM
ARITHMETIC LOGIC UNIT
INPUT OUTPUT
STORED DATA
DATA
DATA

von Neumann vs. Harvard

- Harvard can’t use self-modifying code.
- Harvard allows two simultaneous memory fetches.
- Most DSPs use Harvard architecture for streaming data:
  - greater memory bandwidth;
  - more predictable bandwidth.

MCU LANGUAGE



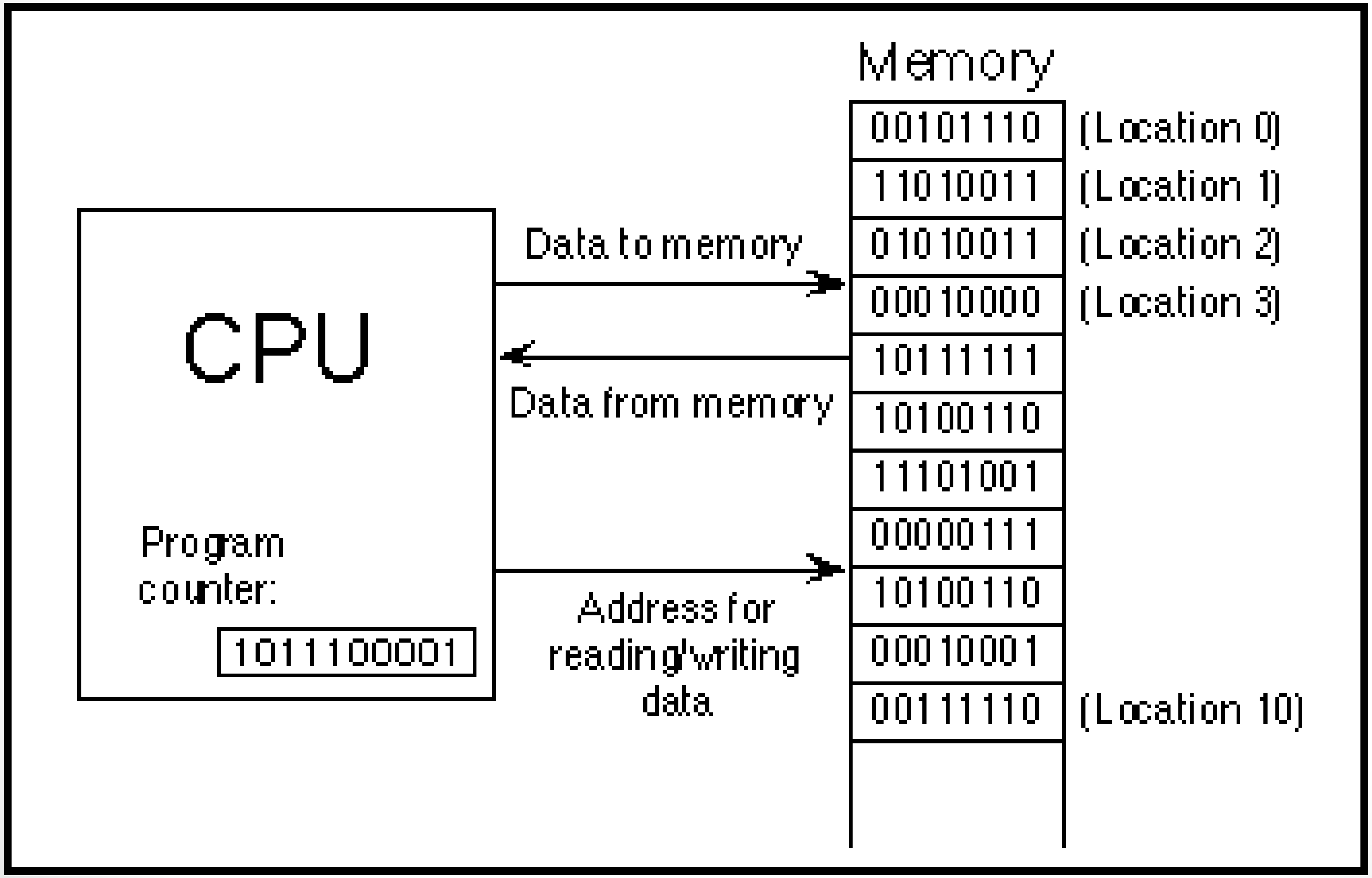
```
temp    = v[k];
v[k]    = v[k+1];
v[k+1]  = temp;
```

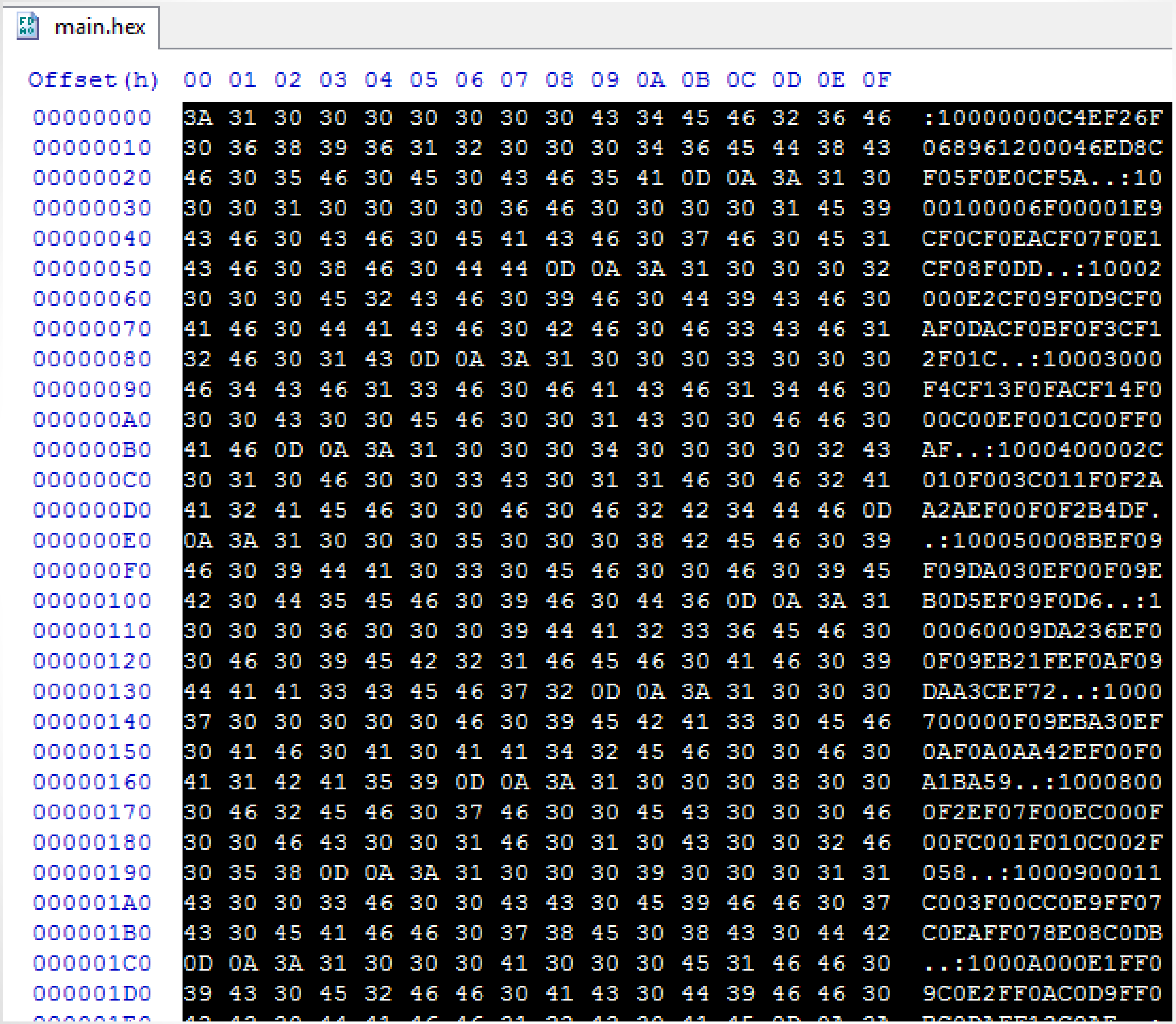
```
TEMP  = V(K)
V(K)    = V(K+1)
V(K+1) = TEMP
```

```
lw  $to,     0($2)
lw  $t1,     4($2)
sw  $t1,     0($2)
sw  $t0,     4($2)
```

```
0000 1001 1100 0110 1010 1111 0101 1000
1010 1111 0101 1000 0000 1001 1100 0110
1100 0110 1010 1111 0101 1000 0000 1001
0101 1000 0000 1001 1100 0110 1010 1111
```

MACHINE LANGUAGE

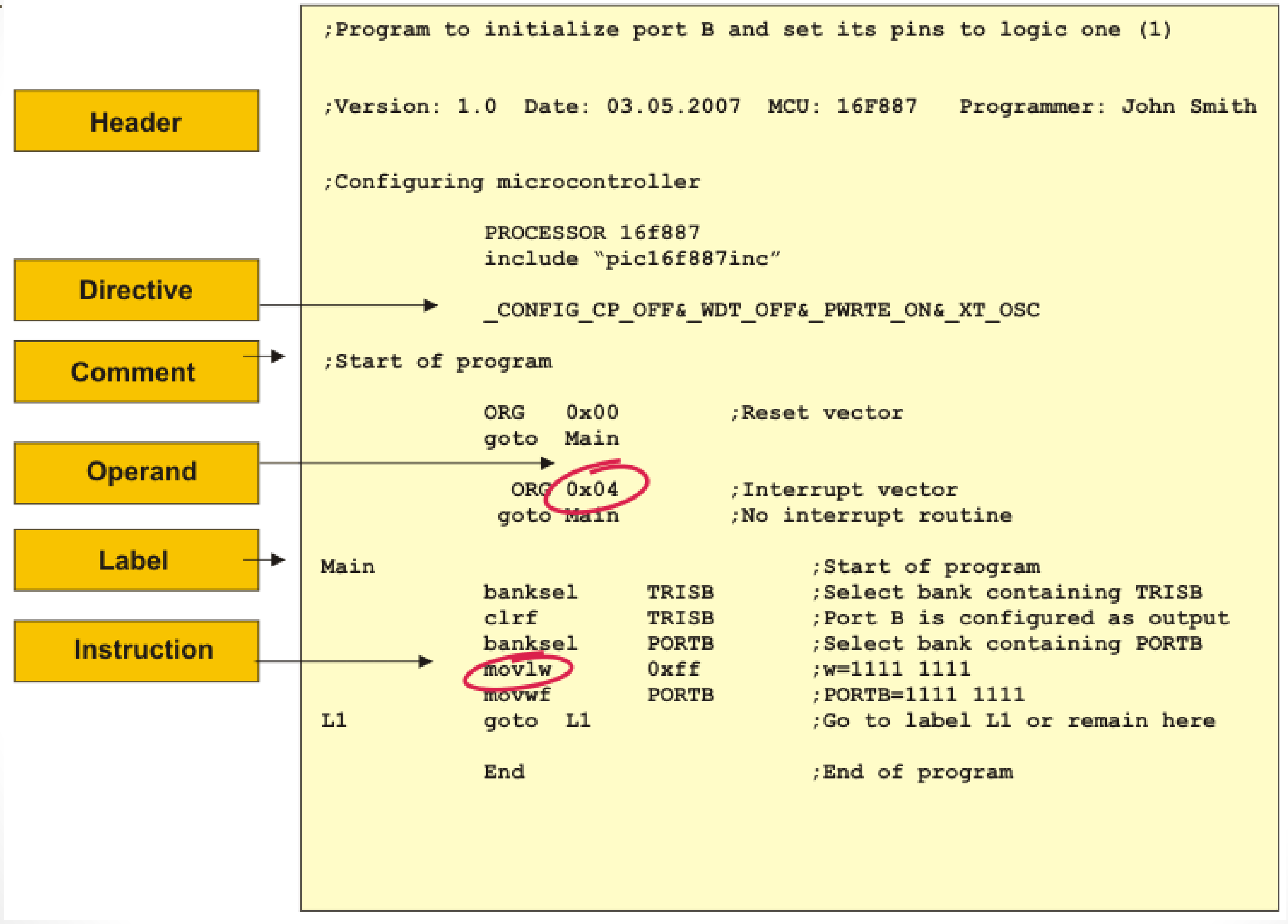




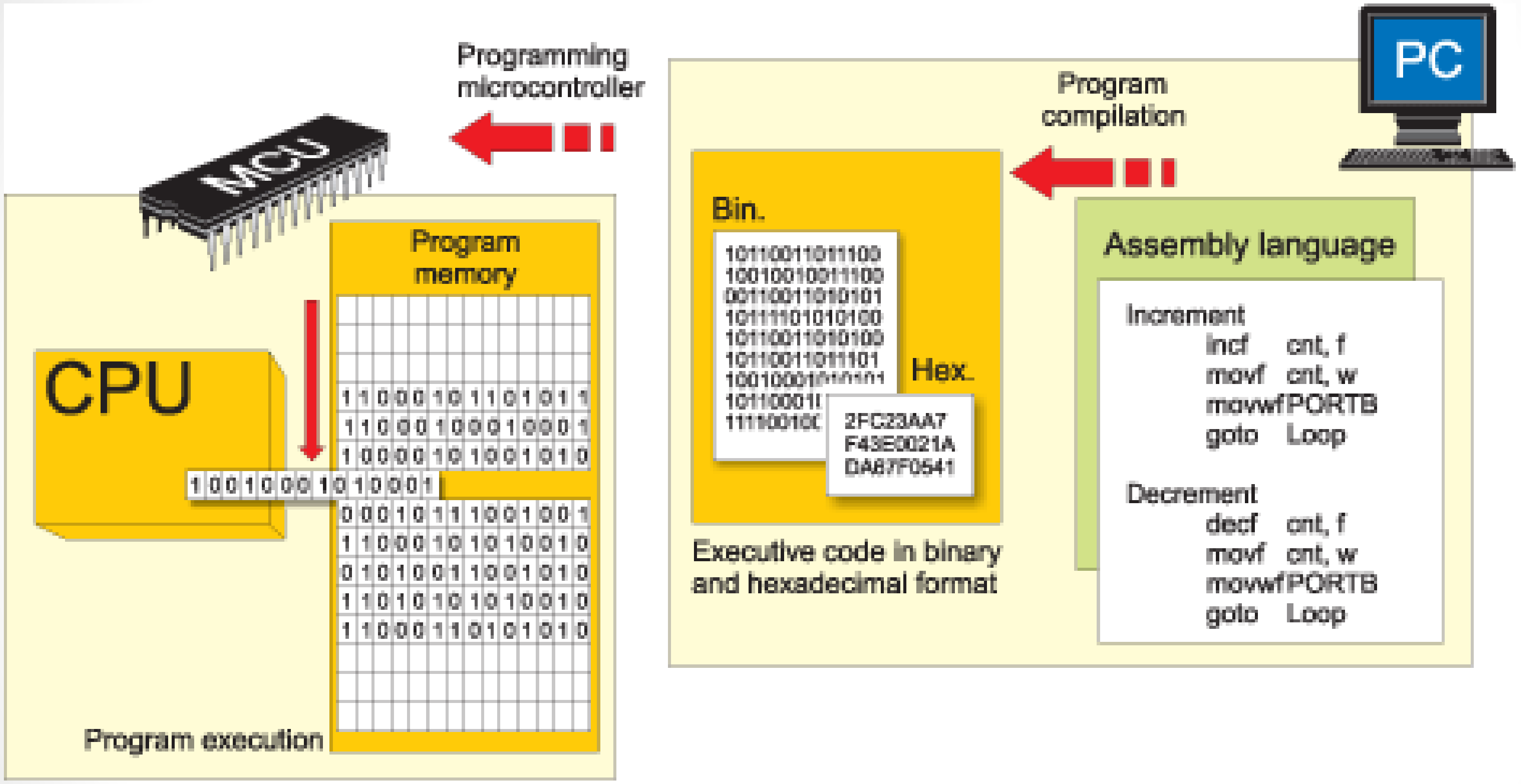
main.hex

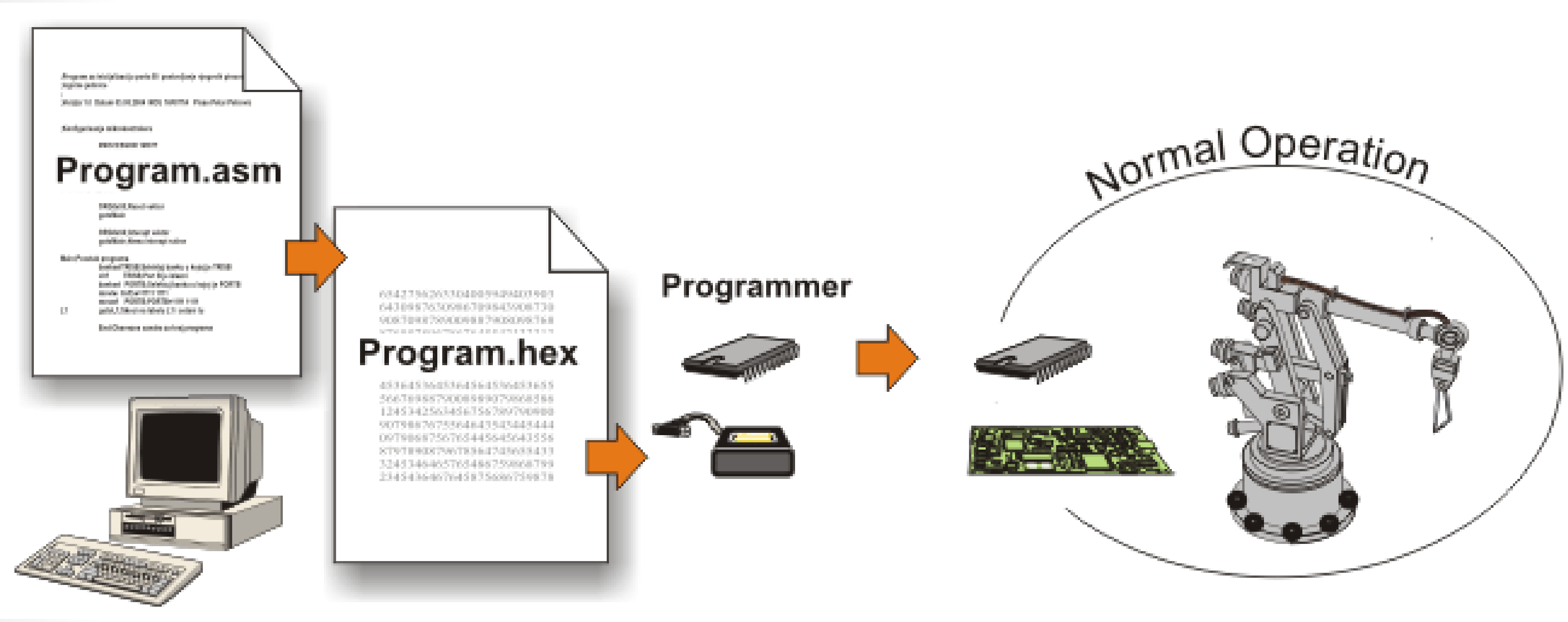
```
Offset(h) 00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F

00000000  3A 31 30 30 30 30 30 30 30 43 34 45 46 32 36 46  :10000000C4EF26F
00000010  30 36 38 39 36 31 32 30 30 30 34 36 45 44 38 43  068961200046ED8C
00000020  46 30 35 46 30 45 30 43 46 35 41 0D 0A 3A 31 30  F05F0E0CF5A..:10
00000030  30 30 31 30 30 30 30 36 46 30 30 30 30 31 45 39  00100006F00001E9
00000040  43 46 30 43 46 30 45 41 43 46 30 37 46 30 45 31  CF0CF0EACF07F0E1
00000050  43 46 30 38 46 30 44 44 0D 0A 3A 31 30 30 30 32  CF08F0DD..:10002
00000060  30 30 30 45 32 43 46 30 39 46 30 44 39 43 46 30  000E2CF09F0D9CF0
00000070  41 46 30 44 41 43 46 30 42 46 30 46 33 43 46 31  AF0DACF0BF0F3CF1
00000080  32 46 30 31 43 0D 0A 3A 31 30 30 30 33 30 30 30  2F01C..:10003000
00000090  46 34 43 46 31 33 46 30 46 41 43 46 31 34 46 30  F4CF13F0FACF14F0
000000A0  30 30 43 30 30 45 46 30 30 31 43 30 30 46 46 30  00C00EF001C00FF0
000000B0  41 46 0D 0A 3A 31 30 30 30 34 30 30 30 30 32 43  AF..:1000400002C
000000C0  30 31 30 46 30 30 33 43 30 31 31 46 30 46 32 41  010F003C011F0F2A
000000D0  41 32 41 45 46 30 30 46 30 46 32 42 34 44 46 0D  A2AEF00F0F2B4DF.
000000E0  0A 3A 31 30 30 30 35 30 30 30 38 42 45 46 30 39  .:100050008BEF09
000000F0  46 30 39 44 41 30 33 30 45 46 30 30 46 30 39 45  F09DA030EF00F09E
00000100  42 30 44 35 45 46 30 39 46 30 44 36 0D 0A 3A 31  B0D5EF09F0D6..:1
00000110  30 30 30 36 30 30 30 39 44 41 32 33 36 45 46 30  00060009DA236EF0
00000120  30 46 30 39 45 42 32 31 46 45 46 30 41 46 30 39  0F09EB21FEF0AF09
00000130  44 41 41 33 43 45 46 37 32 0D 0A 3A 31 30 30 30  DAA3CEF72..:1000
00000140  37 30 30 30 30 30 46 30 39 45 42 41 33 30 45 46  700000F09EBA30EF
00000150  30 41 46 30 41 30 41 41 34 32 45 46 30 30 46 30  0AF0A0AA42EF00F0
00000160  41 31 42 41 35 39 0D 0A 3A 31 30 30 30 38 30 30  A1BA59..:1000800
00000170  30 46 32 45 46 30 37 46 30 30 45 43 30 30 30 46  0F2EF07F00EC000F
00000180  30 30 46 43 30 30 31 46 30 31 30 43 30 30 32 46  00FC001F010C002F
00000190  30 35 38 0D 0A 3A 31 30 30 30 39 30 30 30 31 31  058..:1000900011
000001A0  43 30 30 33 46 30 30 43 43 30 45 39 46 46 30 37  C003F00CC0E9FF07
000001B0  43 30 45 41 46 46 30 37 38 45 30 38 43 30 44 42  C0EAFF078E08C0DB
000001C0  0D 0A 3A 31 30 30 30 41 30 30 30 45 31 46 46 30  ..:1000A000E1FF0
000001D0  39 43 30 45 32 46 46 30 41 43 30 44 39 46 46 30  9C0E2FF0AC0D9FF0
```

ASSEMBLY

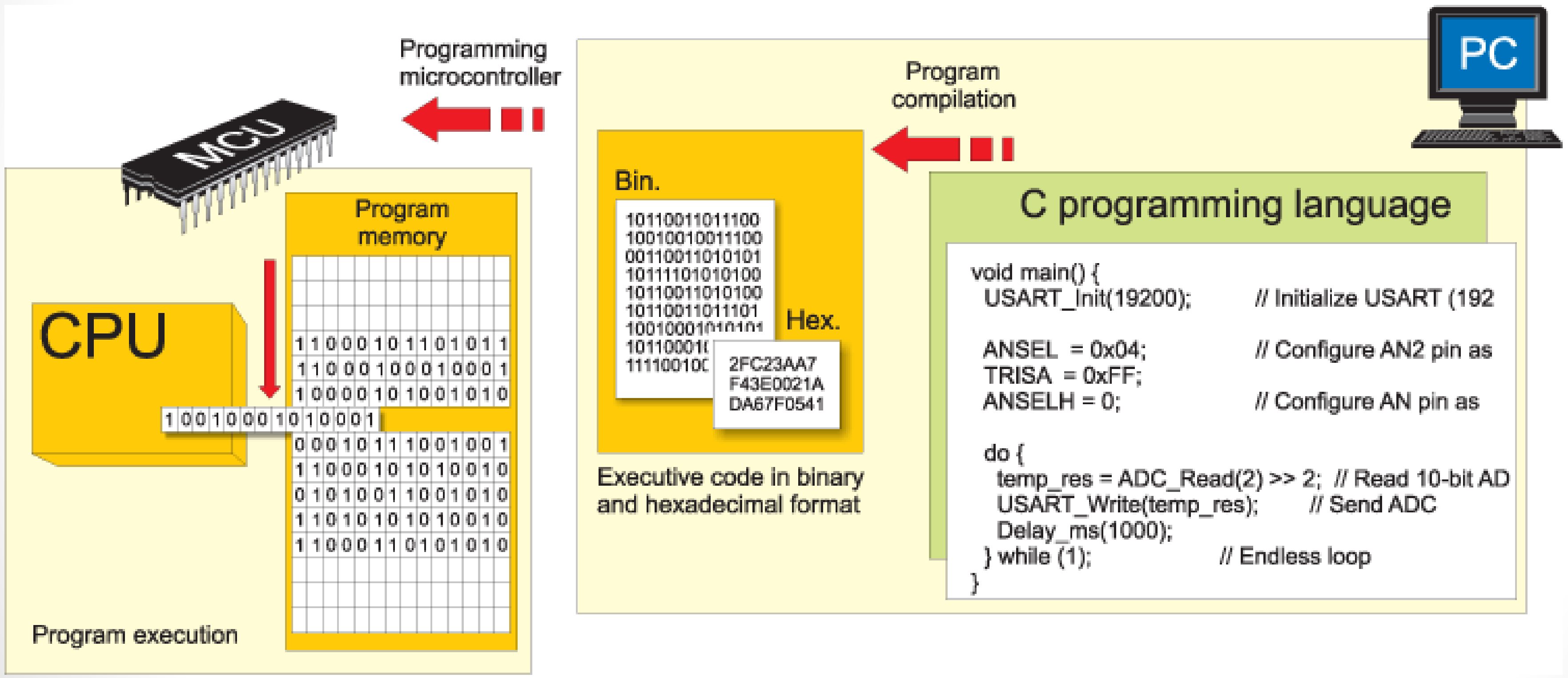


```
;Program to initialize port B and set its pins to logic one (1)


;Version: 1.0  Date: 03.05.2007  MCU: 16F887   Programmer: John Smith


;Configuring microcontroller

            PROCESSOR 16f887
            include "pic16f887inc"

            _CONFIG_CP_OFF&_WDT_OFF&_PWRTE_ON&_XT_OSC

;Start of program

            ORG    0x00          ;Reset vector
            goto   Main

              ORG  0x04          ;Interrupt vector
             goto Main           ;No interrupt routine

Main                                   ;Start of program
            banksel     TRISB          ;Select bank containing TRISB
            clrf        TRISB          ;Port B is configured as output
            banksel     PORTB          ;Select bank containing PORTB
            movlw       0xff           ;w=1111 1111
            movwf       PORTB          ;PORTB=1111 1111
L1          goto   L1                  ;Go to label L1 or remain here

            End                        ;End of program
```

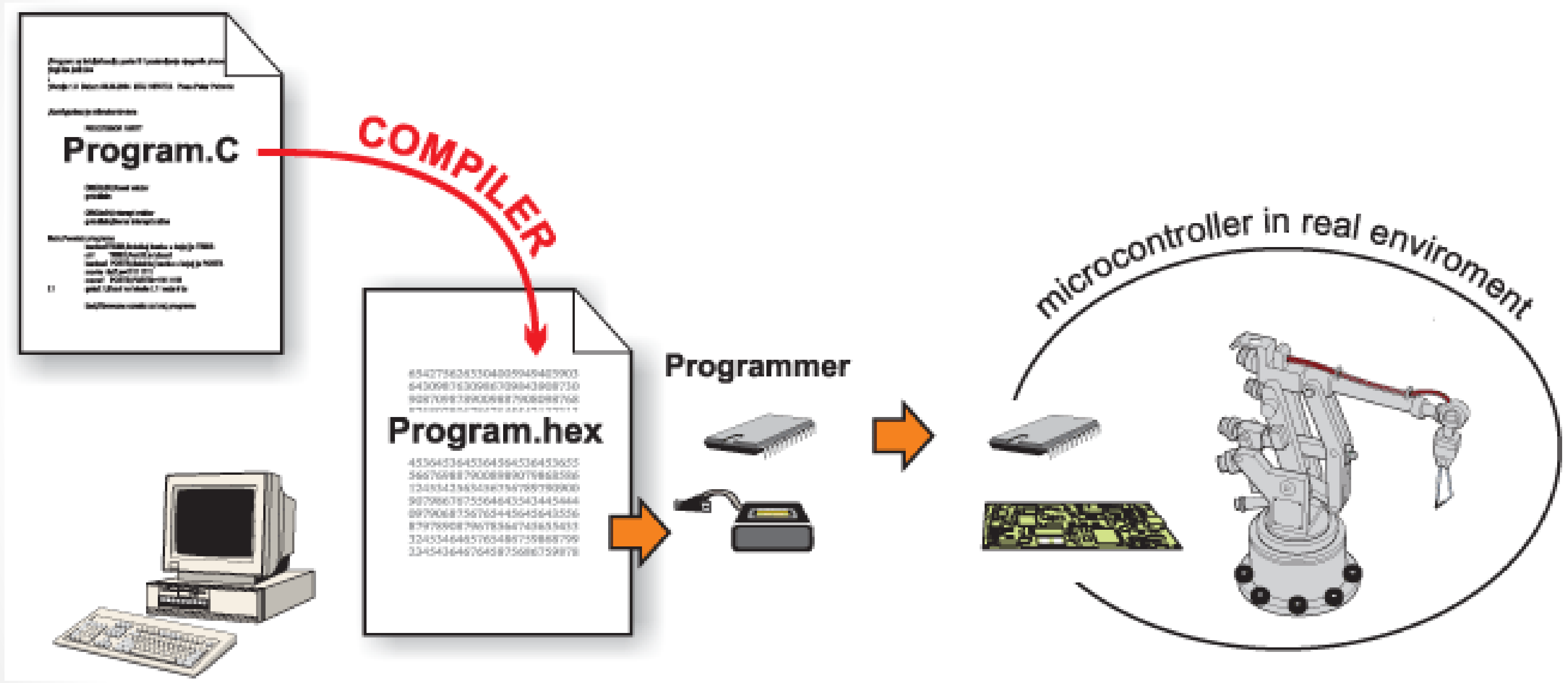

Programming microcontroller
Program compilation
PC
MCU
CPU
Program memory
Program execution
Bin.
10110011011100
10010010011100
00110011010101
10111101010100
10110011010100
10110011011101
Hex.
2FC23AA7
F43E0021A
DA67F0541
Executive code in binary and hexadecimal format
Assembly language
Increment
incf cnt, f
movf cnt, w
movwf PORTB
goto Loop
Decrement
decf cnt, f
movf cnt, w
movwf PORTB
goto Loop


Program.asm
Program.hex
Programmer
Normal Operation

C/C++



```
void main() {
  USART_Init(19200);          // Initialize USART (192

  ANSEL  = 0x04;              // Configure AN2 pin as
  TRISA  = 0xFF;
  ANSELH = 0;                 // Configure AN pin as

  do {
    temp_res = ADC_Read(2) >> 2;  // Read 10-bit AD
    USART_Write(temp_res);        // Send ADC
    Delay_ms(1000);
  } while (1);               // Endless loop
}
```

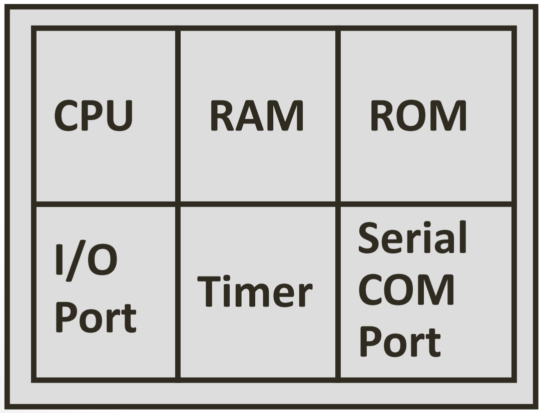

Program.C
COMPILER
Program.hex
Programmer
microcontroller in real enviroment

Assembly vs. C/C++?

8051 MICROCONTROLLER

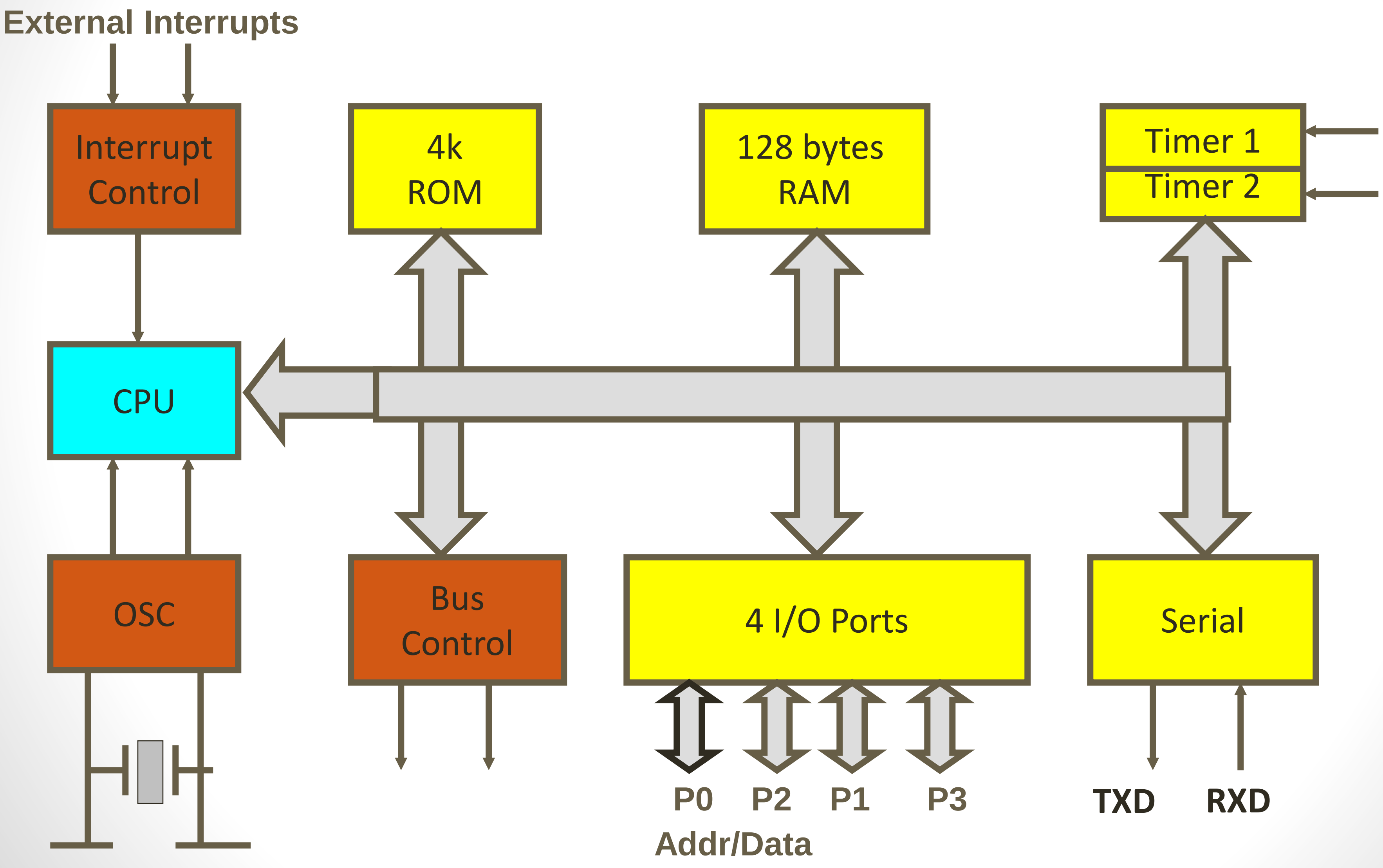
8051 Basic Component

- 4K bytes internal ROM
- 128 bytes internal RAM
- Four 8-bit I/O ports (P0 - P3).
- Two 16-bit timers/counters
- One serial interface



← A single chip Microcontroller

Block Diagram



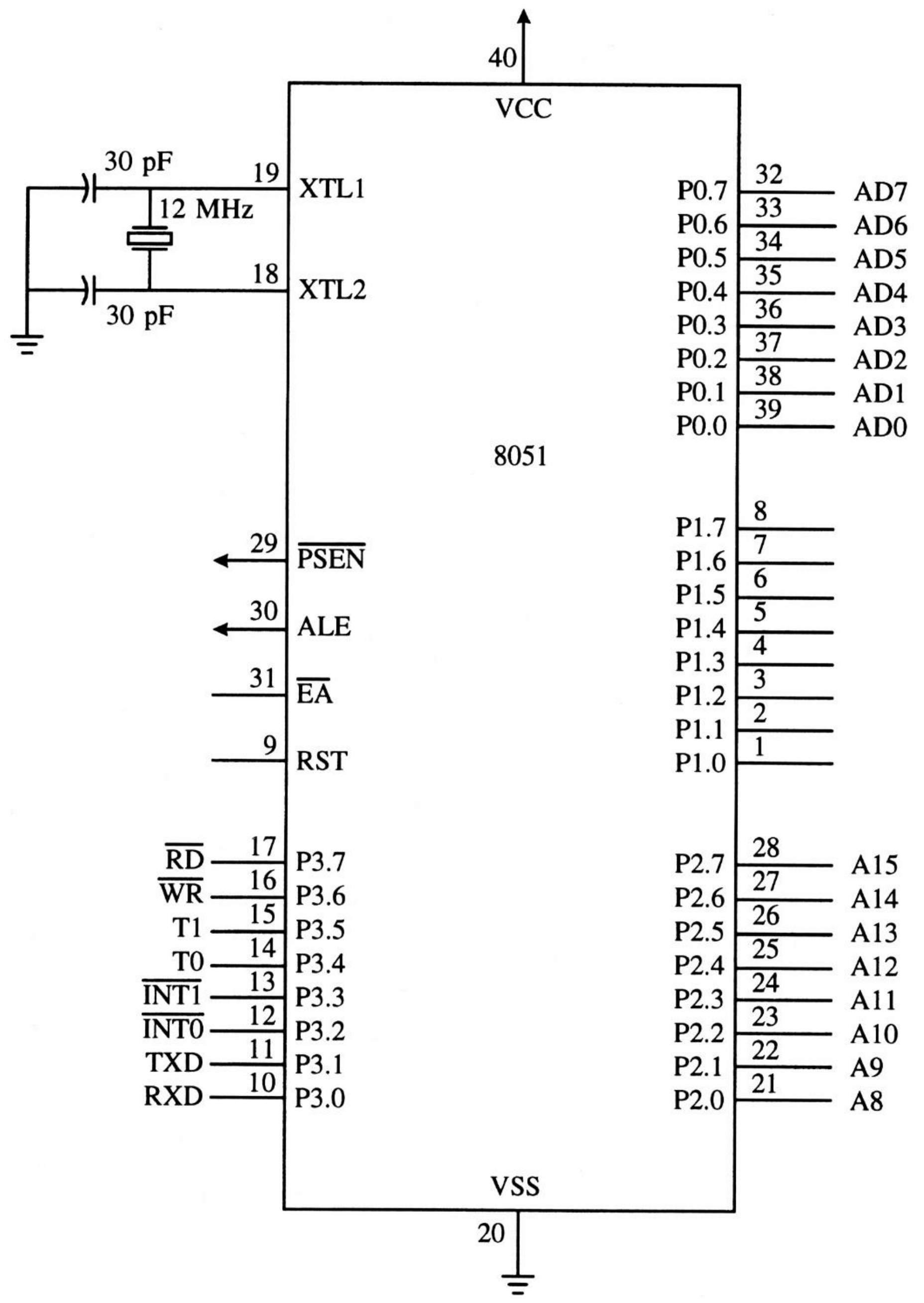
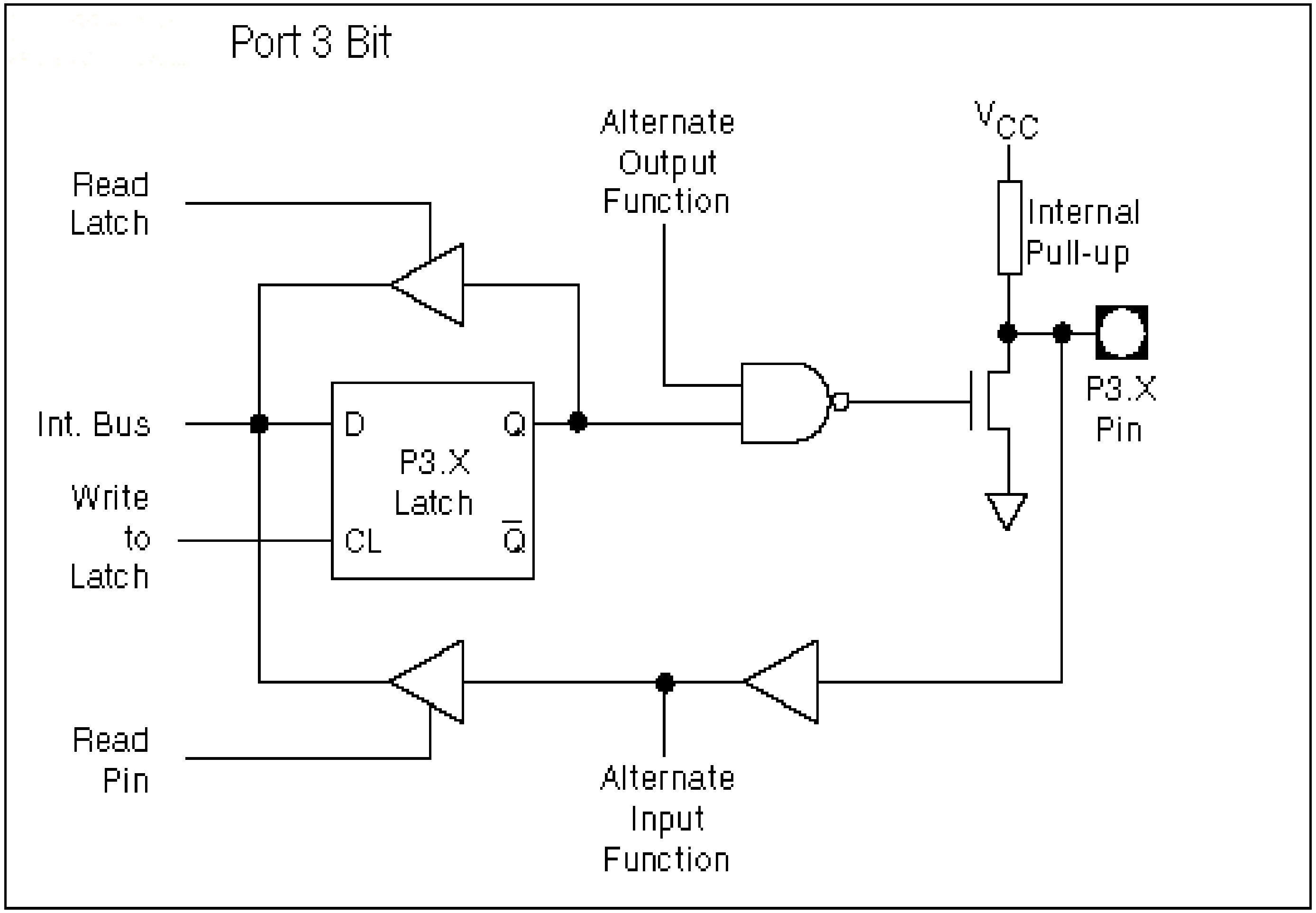
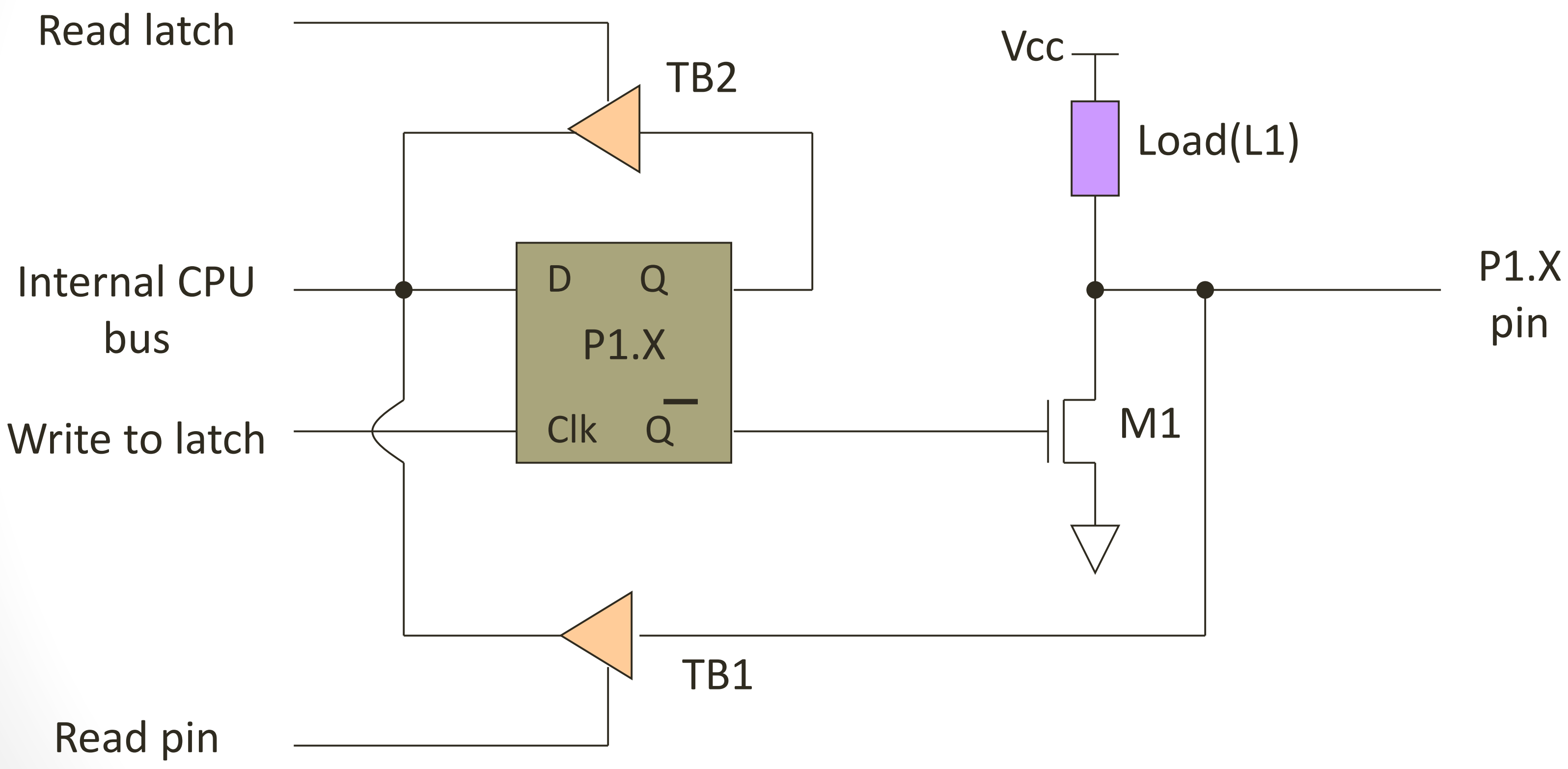


FIGURE 2–2
8051 pinouts

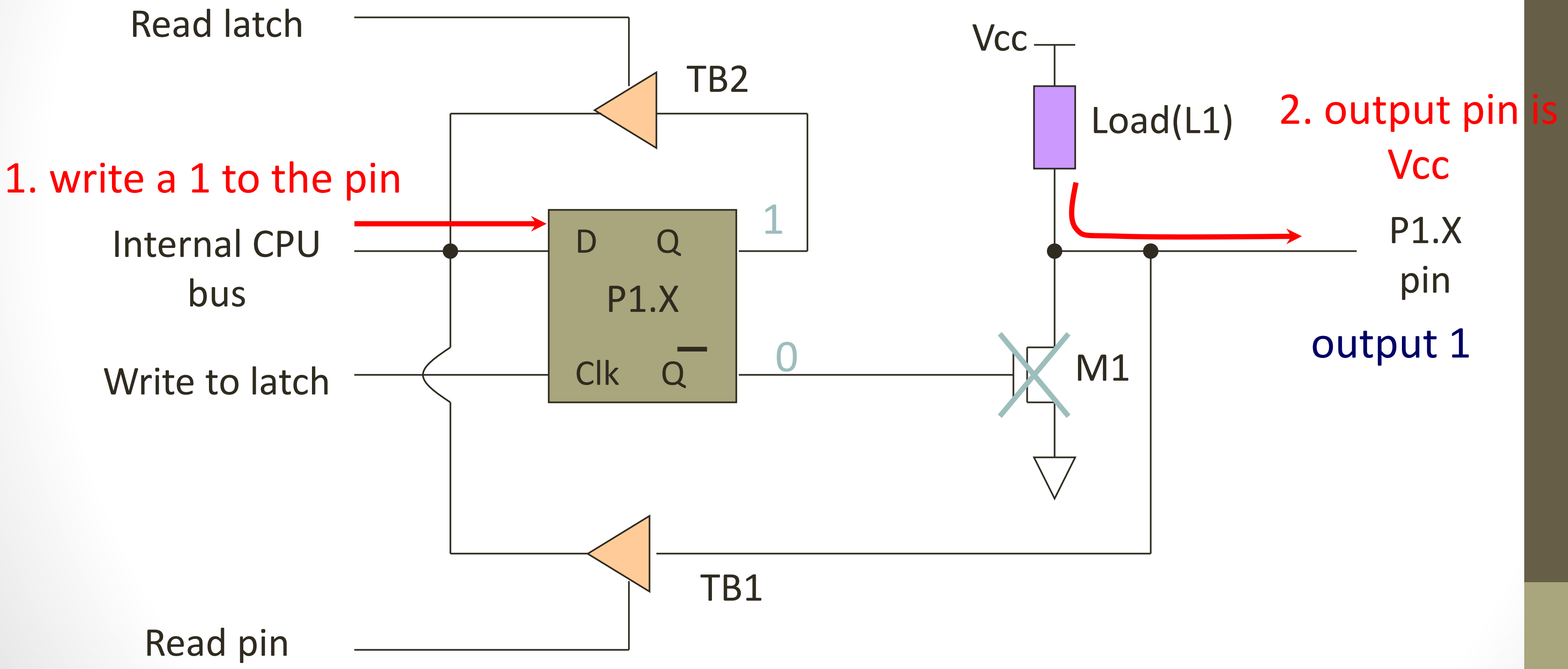
8051 Port 3 Bit Latches and I/O Buffers



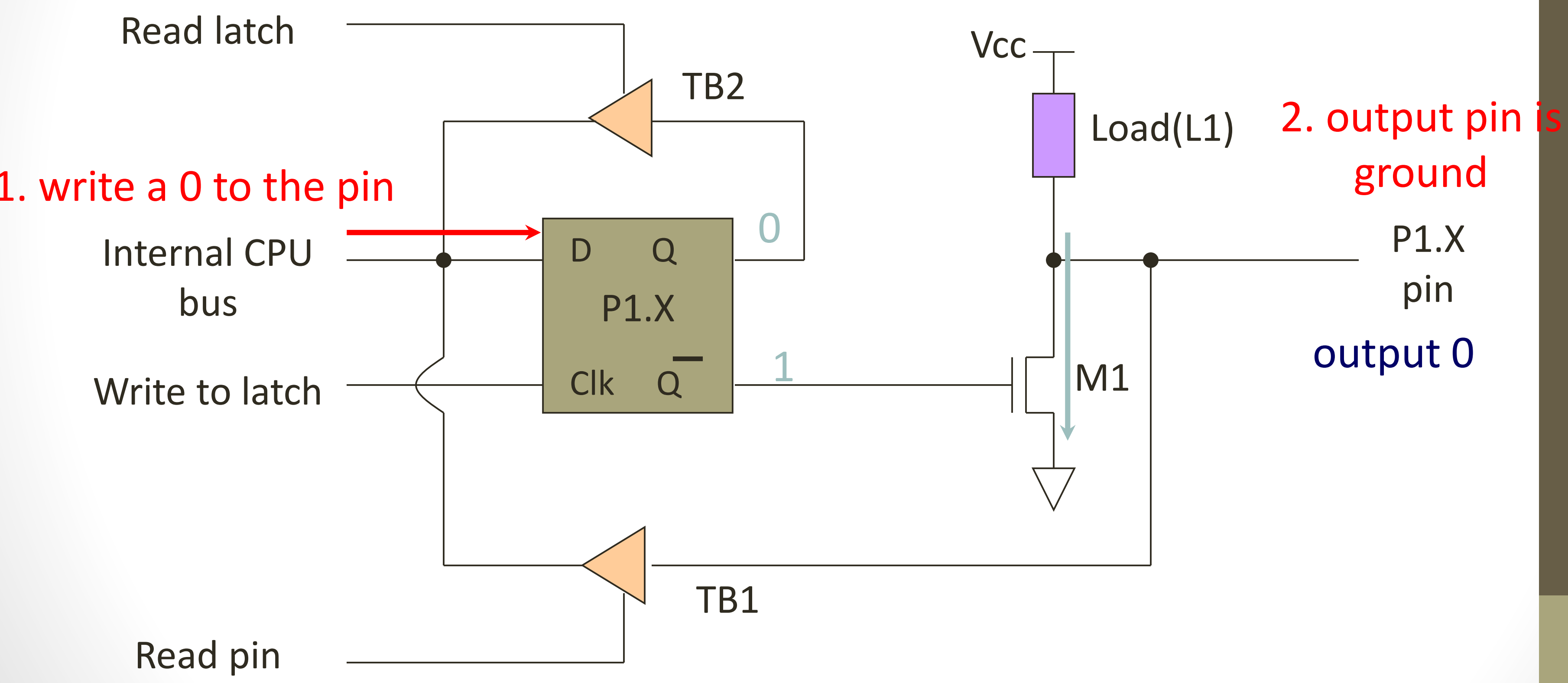
Hardware Structure of I/O Pin



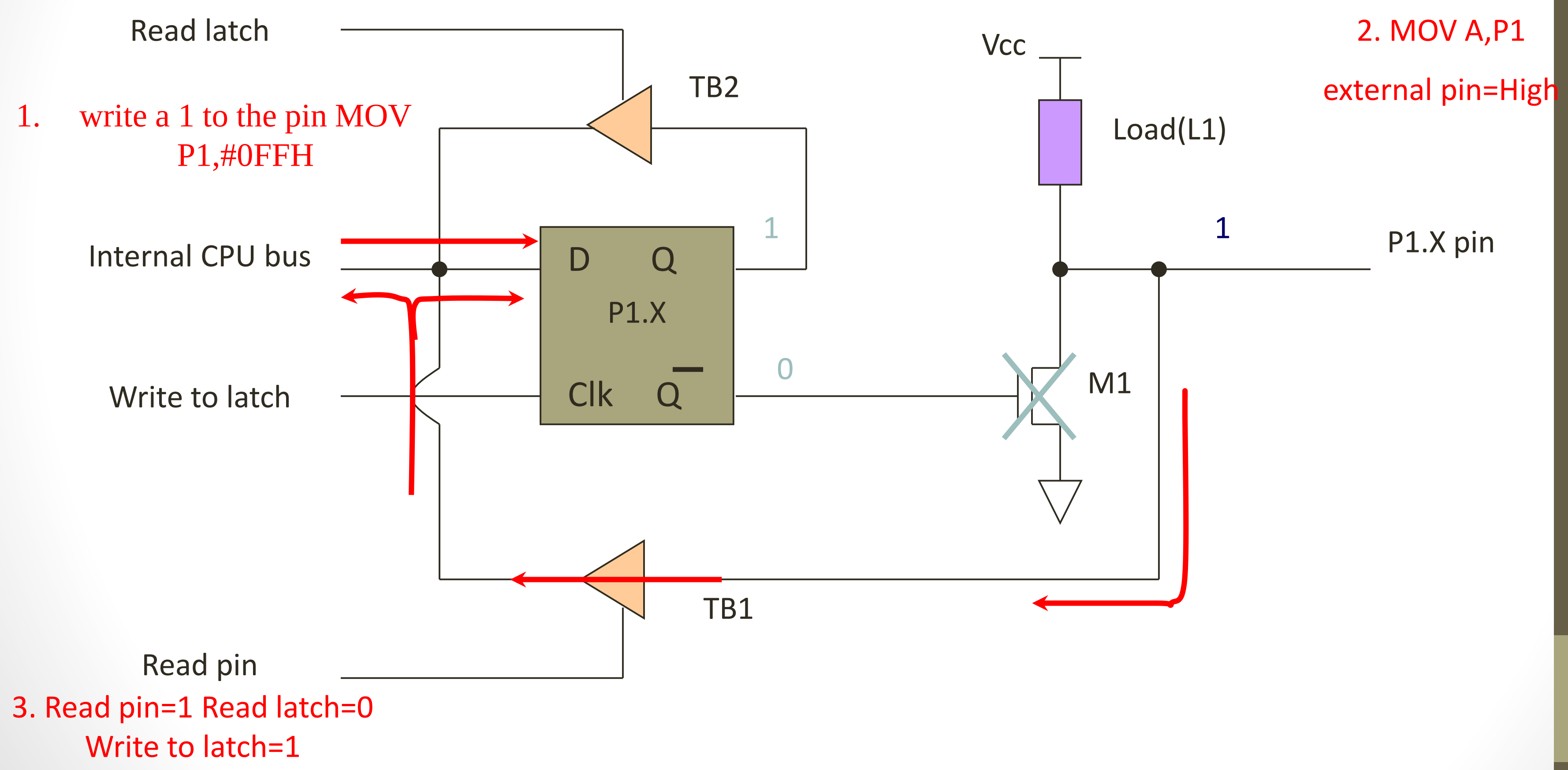
Writing "1" to Output Pin P1.X



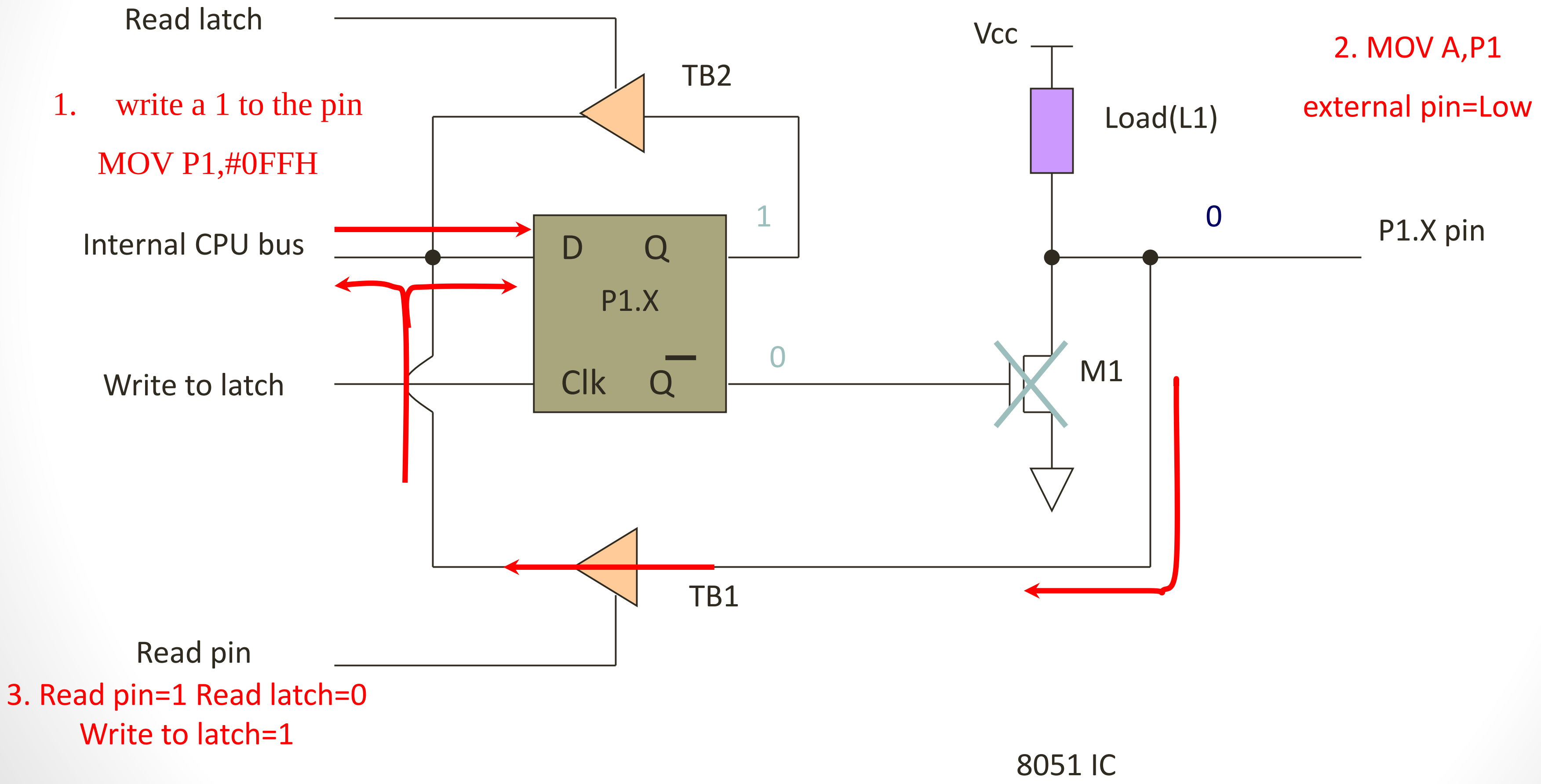
Writing "0" to Output Pin P1.X



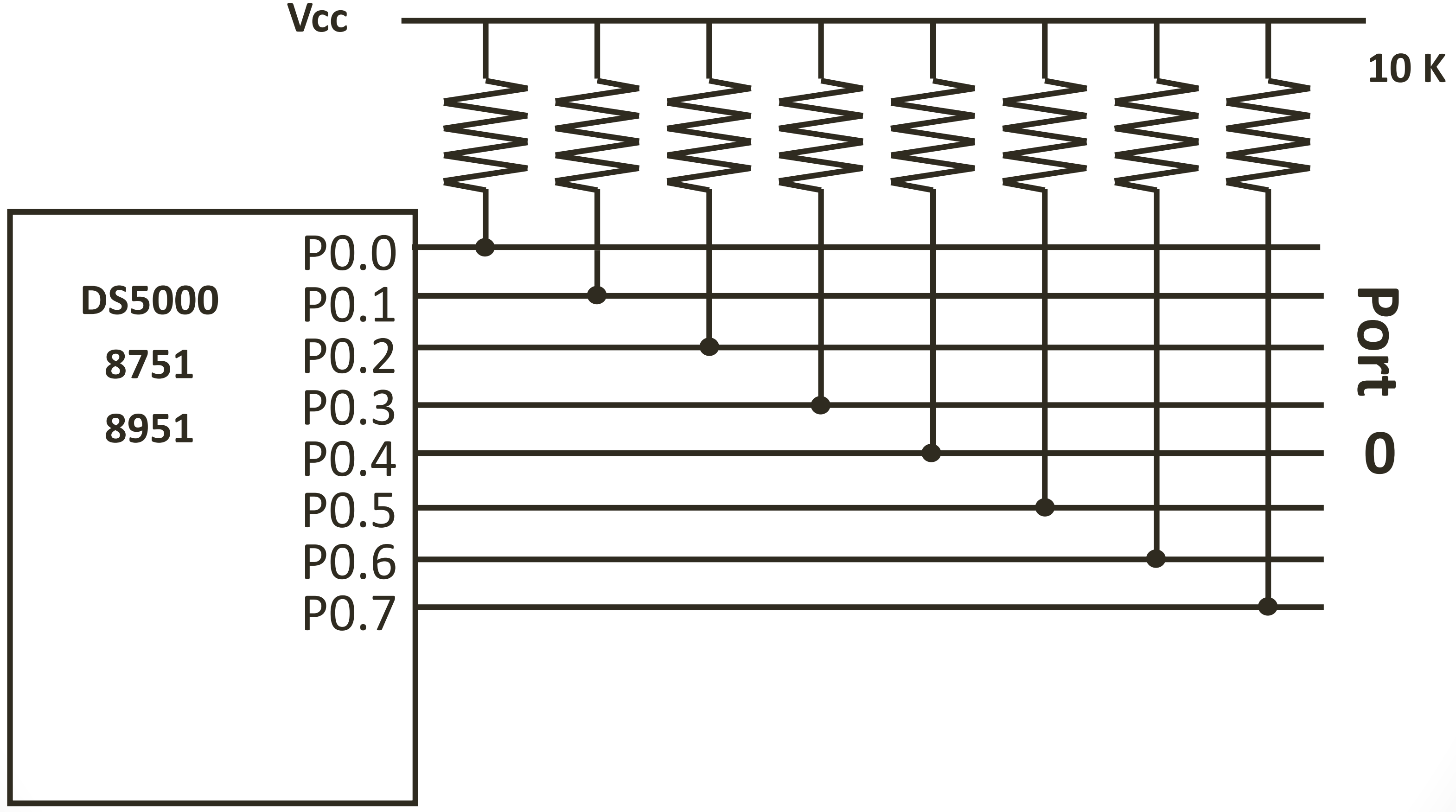
Reading "High" at Input Pin



Reading “Low” at Input Pin



Port 0 with Pull-Up Resistors



IMPORTANT PINS

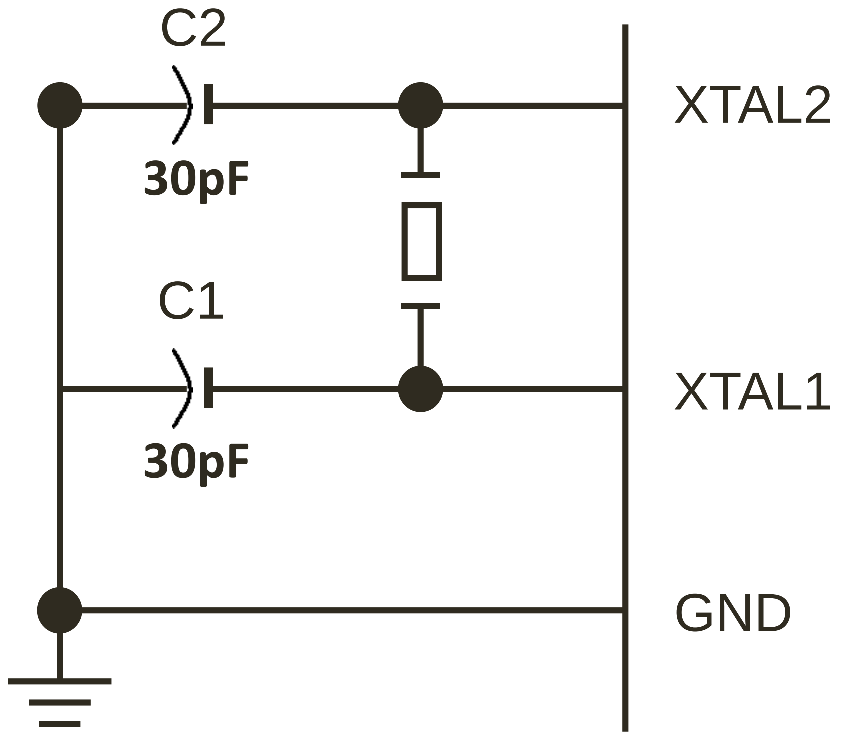
- **PSEN** (out): Program Store Enable, the read signal for external program memory (active low).
- **ALE** (out): Address Latch Enable, to latch address outputs at Port0 and Port2
- **EA** (in): External Access Enable, active low to access external program memory locations 0 to 4K
- **RXD,TXD**: UART pins for serial I/O on Port 3
- **XTAL1 & XTAL2**: Crystal inputs for internal oscillator.

Pins of 8051

- Vcc（pin 40）：
  - Vcc provides supply voltage to the chip.
  - The voltage source is +5V.
- GND（pin 20）：ground
- XTAL1 and XTAL2（pins 19,18）：
  - These 2 pins provide external clock.
  - Way 1：using a quartz crystal oscillator
  - Way 2：using a TTL oscillator
  - Example 4-1 shows the relationship between XTAL and the machine cycle.

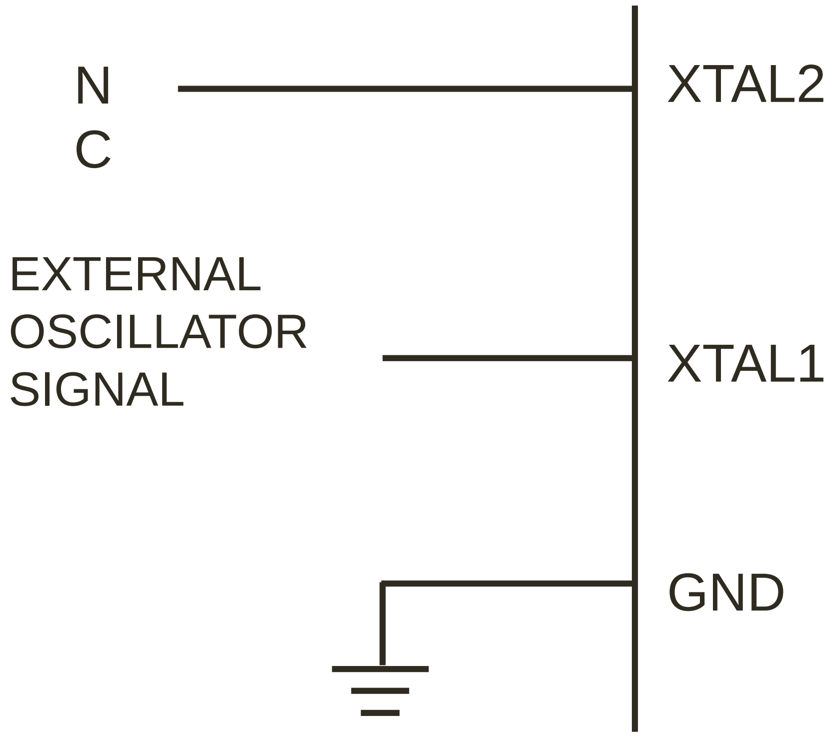
XTAL Connection to 8051

- Using a quartz crystal oscillator
- We can observe the frequency on the XTAL2 pin.



XTAL Connection to an External Clock Source

- Using a TTL oscillator
- XTAL2 is unconnected.



Machine cycle

- Find the machine cycle for
- (a) XTAL = 11.0592 MHz
- (b) XTAL = 16 MHz.

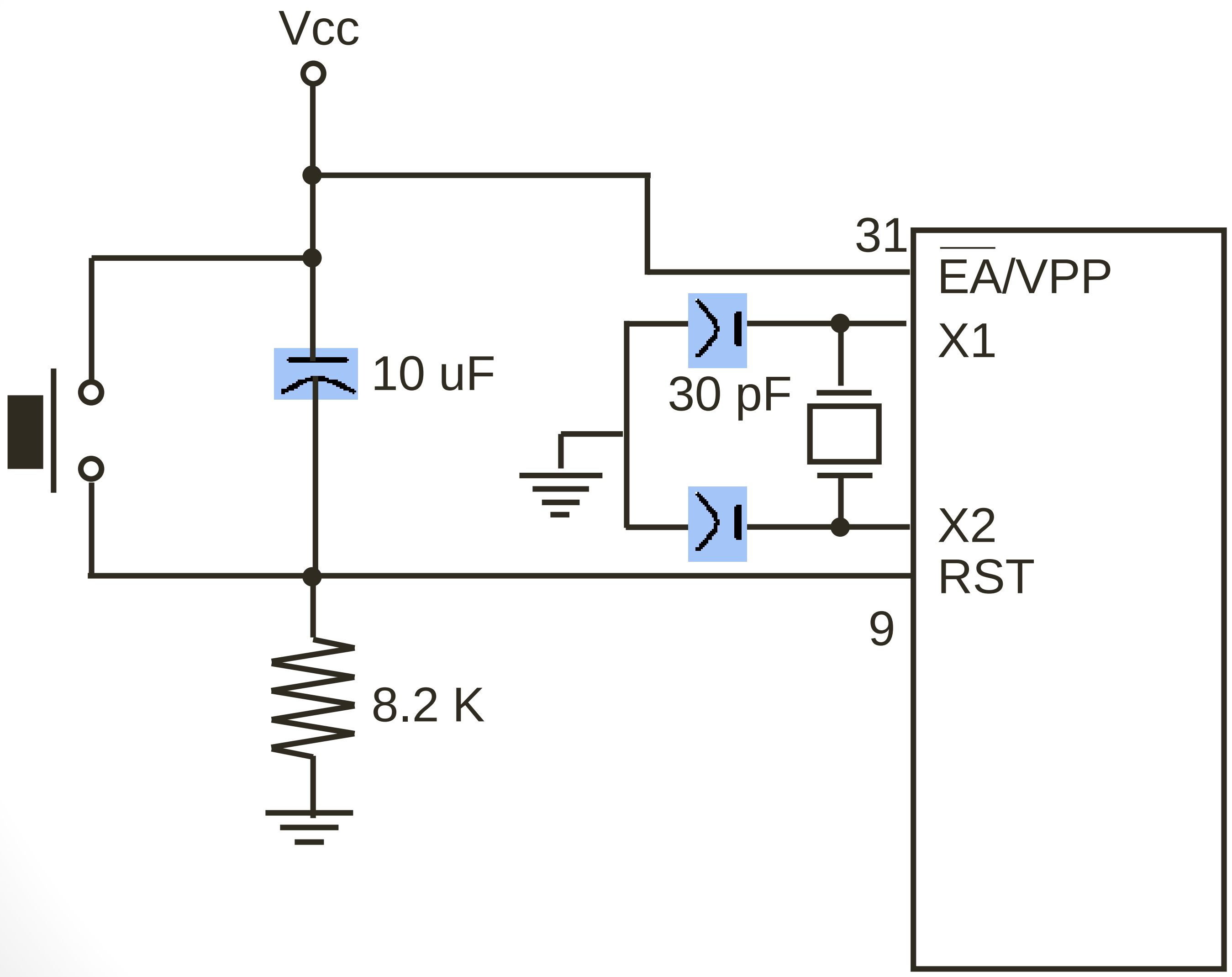
- Solution:

  - (a) 11.0592 MHz / 12 = 921.6 kHz;
  - machine cycle = 1 / 921.6 kHz = 1.085 us
  - (b) 16 MHz / 12 = 1.333 MHz;
  - machine cycle = 1 / 1.333 MHz = 0.75 us

Pins of 8051

- RST（pin 9）：reset
  - input pin and active high（normally low）.
    - The high pulse must be high at least 2 machine cycles.
  - power-on reset.
    - Upon applying a high pulse to RST, the microcontroller will reset and all values in registers will be lost.
    - Reset values of some 8051 registers
  - power-on reset circuit

Power-On RESET



RESET Value of Some 8051 Registers

Register	Reset Value
PC	0000
ACC	0000
B	0000
PSW	0000
SP	0007
DPTR	0000

RAM are all zero

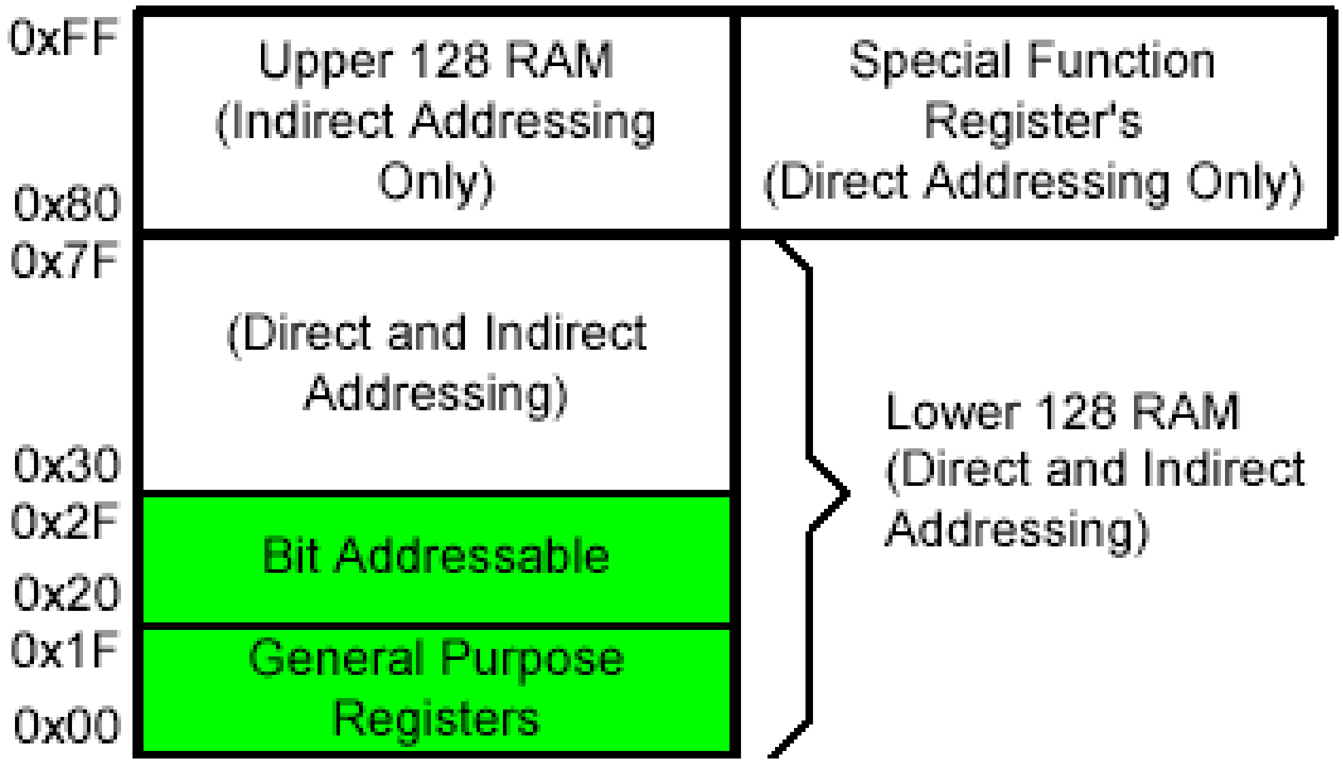
Pins of 8051

- /EA（pin 31）：external access
  - There is no on-chip ROM in 8031 and 8032 .
  - The /EA pin is connected to GND to indicate the code is stored externally.
  - /PSEN ＆ ALE are used for external ROM.
  - For 8051, /EA pin is connected to Vcc.
  - “/” means active low.
- /PSEN（pin 29）：program store enable
  - This is an output pin and is connected to the OE pin of the ROM.
  - See Chapter 14.

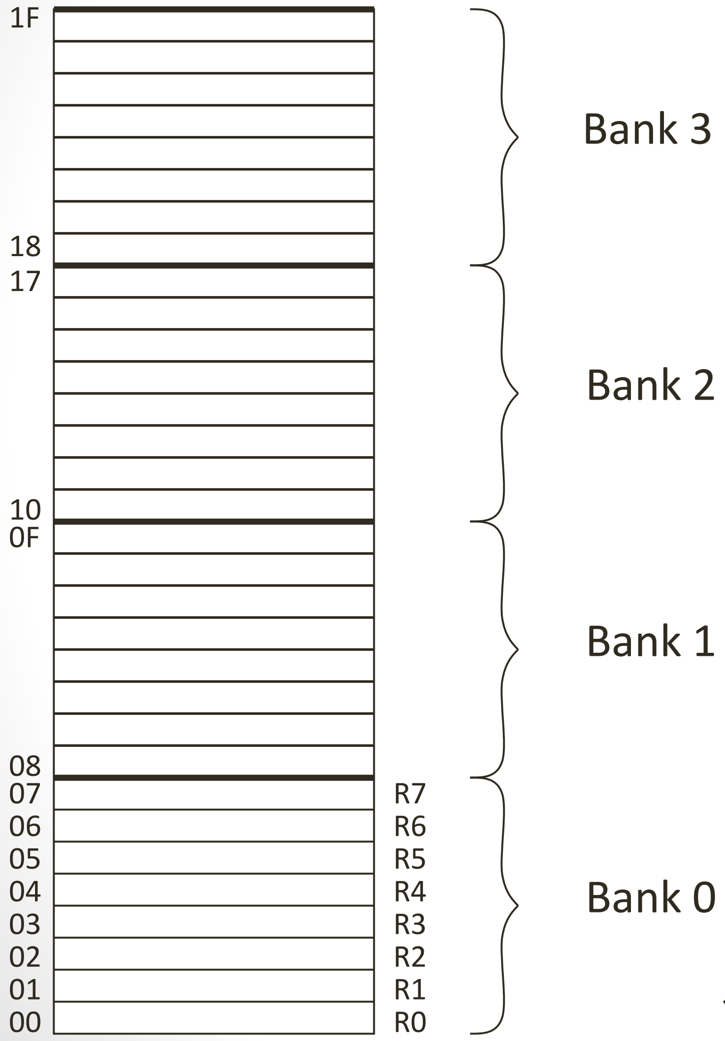
Pins of 8051

- ALE（pin 30）：address latch enable
  - It is an output pin and is active high.
  - 8051 port 0 provides both address and data.
  - The ALE pin is used for de-multiplexing the address and data by connecting to the G pin of the 74LS373 latch.

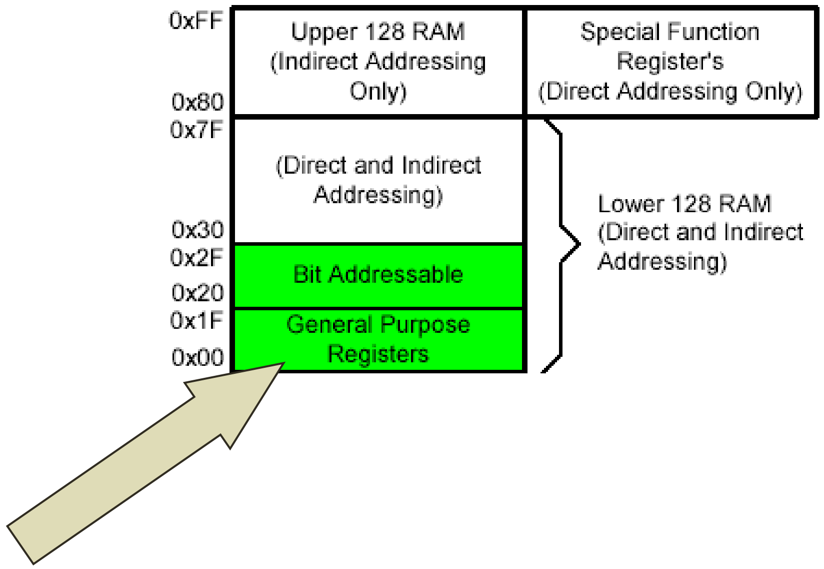
On-Chip Memory
Internal RAM


0xFF
0x80
0x7F
0x30
0x2F
0x20
0x1F
0x00
Upper 128 RAM (Indirect Addressing Only)
Special Function Register's (Direct Addressing Only)
(Direct and Indirect Addressing)
Bit Addressable
General Purpose Registers
Lower 128 RAM (Direct and Indirect Addressing)

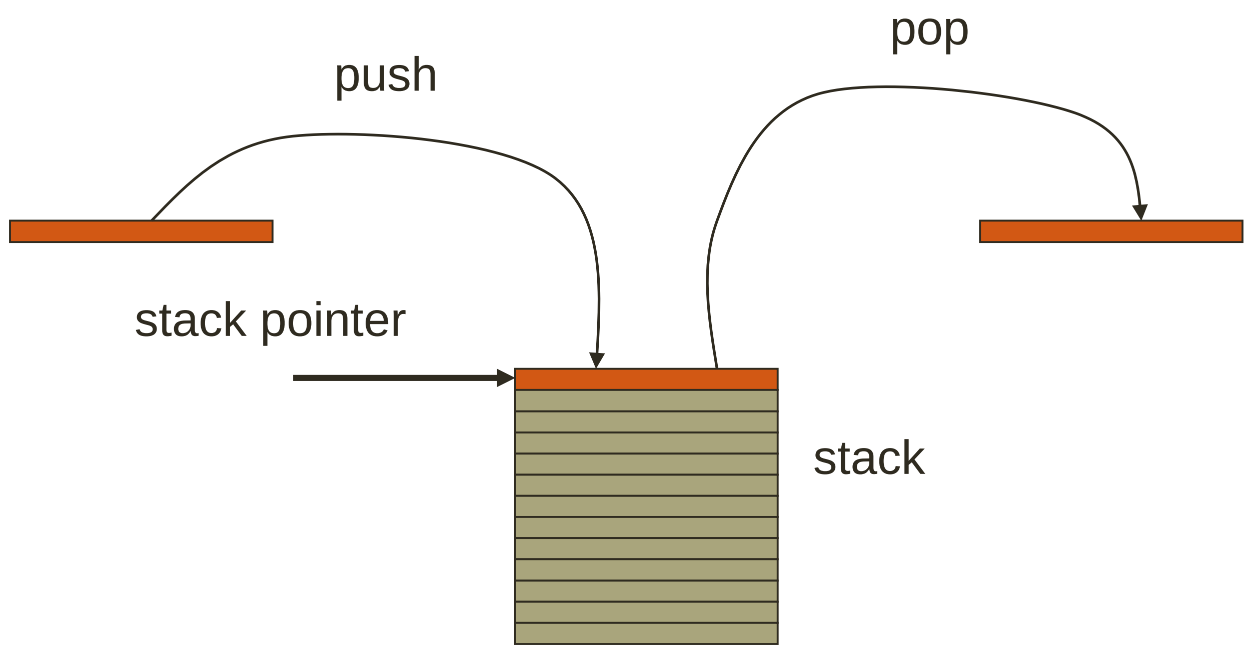
Registers



Four Register Banks
Each bank has R0-R7
Selectable by psw.2,3


0xFF
0x80
0x7F
0x30
0x2F
0x20
0x1F
0x00
Upper 128 RAM (Indirect Addressing Only)
Special Function Register's (Direct Addressing Only)
(Direct and Indirect Addressing)
Bit Addressable
General Purpose Registers
Lower 128 RAM (Direct and Indirect Addressing)

Stacks



The 8051
Assembly Language

Data Transfer Instructions

- MOV dest, source   dest ← source
- Stack instructions

```
PUSH byte ;increment stack pointer,
               ;move byte on stack
POP byte       ;move from stack to byte,
                     ;decrement stack pointer
```

- Exchange instructions

```
XCH a, byte    ;exchange accumulator and byte
XCHD a, byte   ;exchange low nibbles of
                     ;accumulator and byte
```

Arithmetic Instructions

Mnemonic	Description
`ADD A, byte`	add A to byte, put result in A
`ADDC A, byte`	add with carry
`SUBB A, byte`	subtract with borrow
`INC A`	increment A
`INC byte`	increment byte in memory
`INC DPTR`	increment data pointer
`DEC A`	decrement accumulator
`DEC byte`	decrement byte
`MUL AB`	multiply accumulator by b register
`DIV AB`	divide accumulator by b register
`DA A`	decimal adjust the accumulator

8051 Examples

AVR MICROCONTROLLER ATmega8

INTRODUCTION

- 130 Instructions – Most Single-clock Cycle Execution
- 32 x 8 General Purpose Working Registers
- 64 x 8 Special Function Registers (I/O Registers)
- Up to 16 MIPS Throughput at 16 MHz
- On-chip 2-cycle Multiplier

Nonvolatile Program and Data Memories

- 8K Bytes of In-System Self-Programmable Flash
  10,000 Write/Erase Cycles
- Optional Boot Code Section with Independent Lock Bits
- 512 Bytes EEPROM (100,000 Write/Erase Cycles)
- 1K Byte Internal SRAM
- Programming Lock for Software Security

Peripheral Features

- Two 8-bit Timer/Counters
- One 16-bit Timer/Counter with Capture Mode
- Real Time Counter with Separate Oscillator
- Three PWM Channels
- 6-channel ADC with 10 resp 8 Bit resolution (TQFP: 8 channels)
- Two-wire Serial Interface (TWI)
- Programmable Serial USART
- Master/Slave SPI Serial Interface
- Programmable Watchdog Timer with On-chip Oscillator
- On-chip Analog Comparator

II: Atmega

Special Microcontroller Features

- Programmable Brown-out Detection
- Internal Calibrated RC Oscillator
- External and Internal Interrupt Sources
- Five Sleep Modes

I/O and Packages

- 23 Programmable I/O Lines
- 28-lead PDIP, 32-lead TQFP, and 32-pad MLF

II: Atmega

Operating Voltages

- 2.7 - 5.5V (ATmega8L)
- 4.5 - 5.5V (ATmega8)

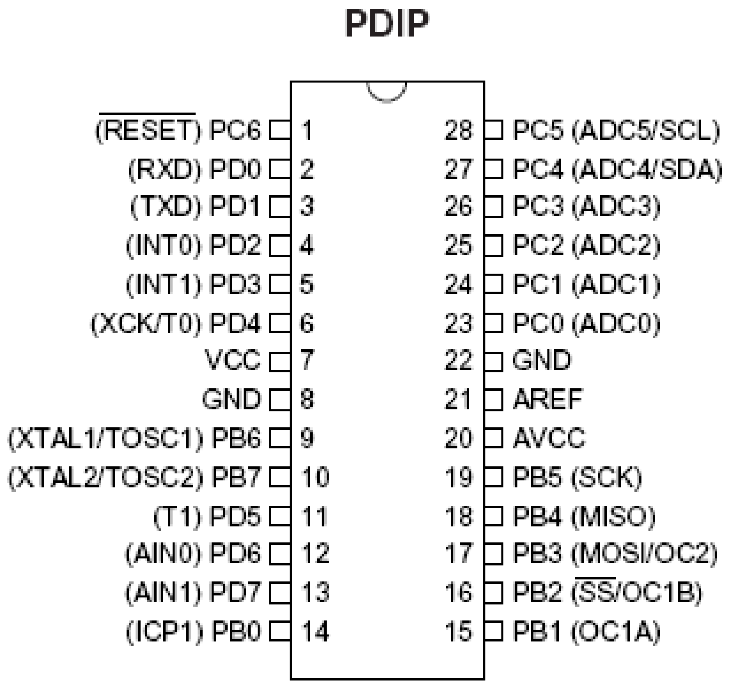
Speed Grades

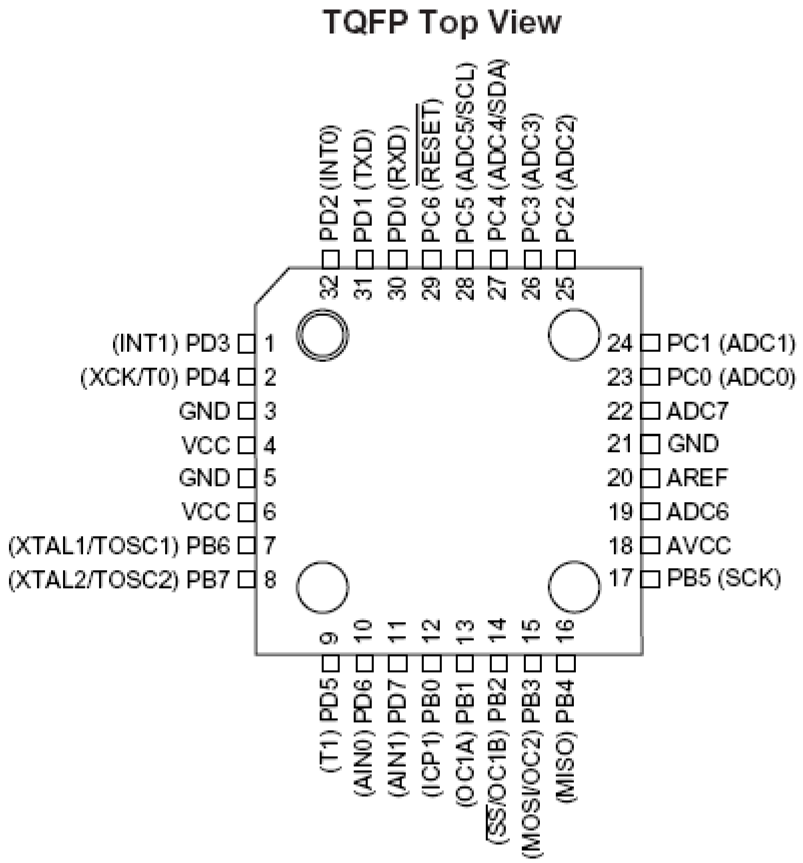
- 0 - 8 MHz (ATmega8L)
- 0 - 16 MHz (ATmega8)

Power Consumption at 4 Mhz, 3V, 25°C

- Active: 3.6 mA
- Idle Mode: 1.0 mA
- Power-down Mode: 0.5 µA

II: Atmega



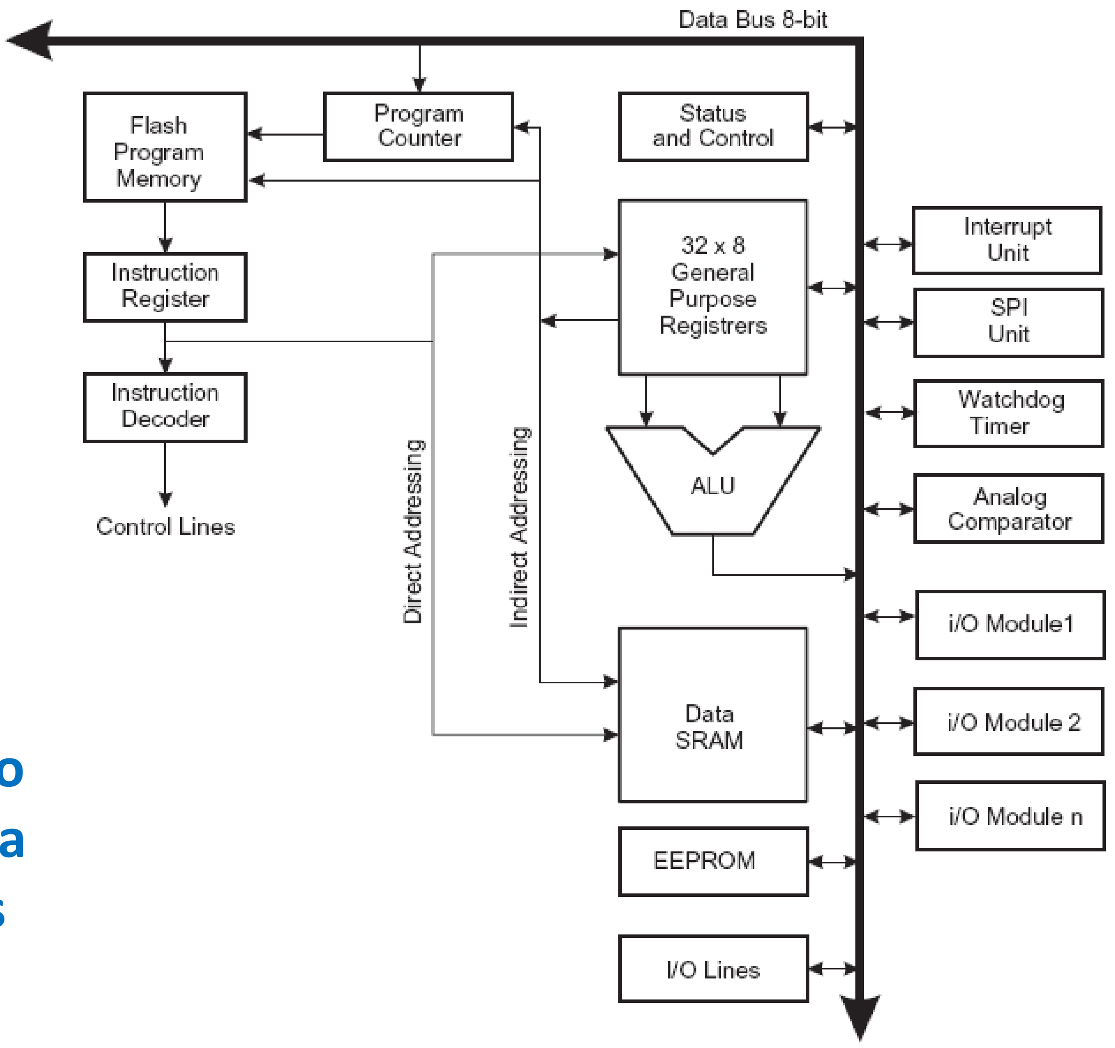


ATmega8 Pinout and Packages (DIP and TQFP)

II: Atmega

Mega8 CPU Core

- Seperate Instruction and Data Memories (Harvard)
- all 32 General Purpose Registers connected to ALU
- I/O Modules connected to Data Bus and accessible via Special Function Registers



II: Atmega

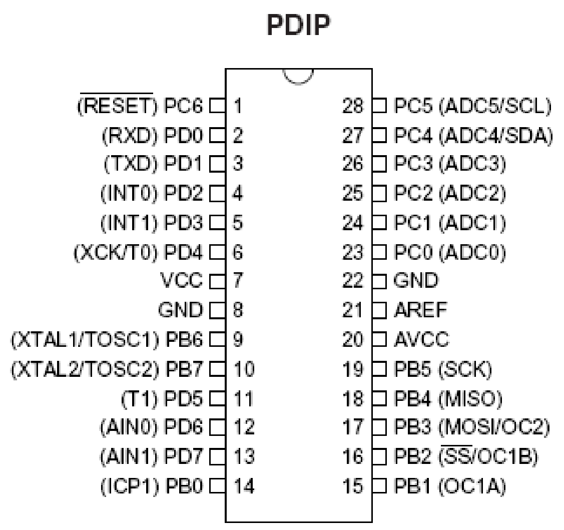
Pin and Port Overview:

GND: Ground (0V)
VCC: Digital Supply Voltage (2,7 – 5,5V)
AVCC: Analog Supply Voltage
  connect to low-pass filtered VCC
AREF: Analog Reference Voltage, usually AVCC
/Reset: Low level on this pin will generate a reset

Port B, Port C, Port D:
- General Purpose 8 Bit bidirectional I/O - Ports,
- optional internal pullup-resistors when configured as input
- output source capability: 20mA

Special Functions of the Ports available as configured using the SFRs:

Port D: Uart, external Interrupts, Analog Comparator
Port B: External Oscillator/Crystal, SPI
Port C: A/D converters, TWI



Memory organization

AVR Memory organization:

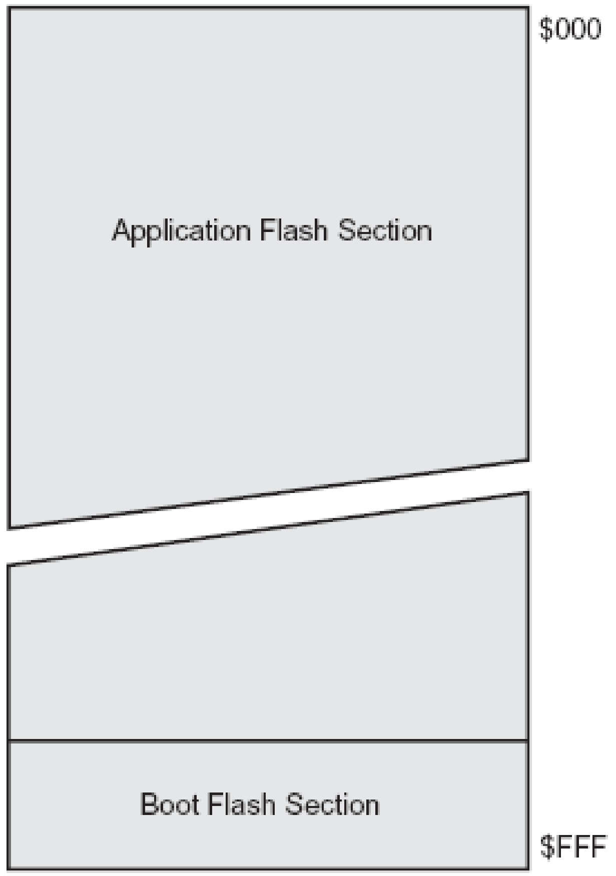
● Program Flash Memory:

On-chip, in system programmable

8 Kbytes, organized in 4K 16 bit words
Program Counter (PC) = 12 bits

Accessible via special instructions: LPM, SPM

Boot Loader support: Boot Flash Section,
SPM can be executed only from Boot Flash



II: Atmega

AVR Memory organization:

● **EEPROM - Memory:**

512 Bytes, single Bytes can be read and written

Special EEPROM read and write procedure using SFRs:

EEPROM Address Register, EEPROM Data Register,
EEPROM Control Register
C – Library Functions available

Precautions to prevent EEPROM memory corruption:

- **no flash memory or interrupt operations**
- **stable power supply**

AVR Memory organization:

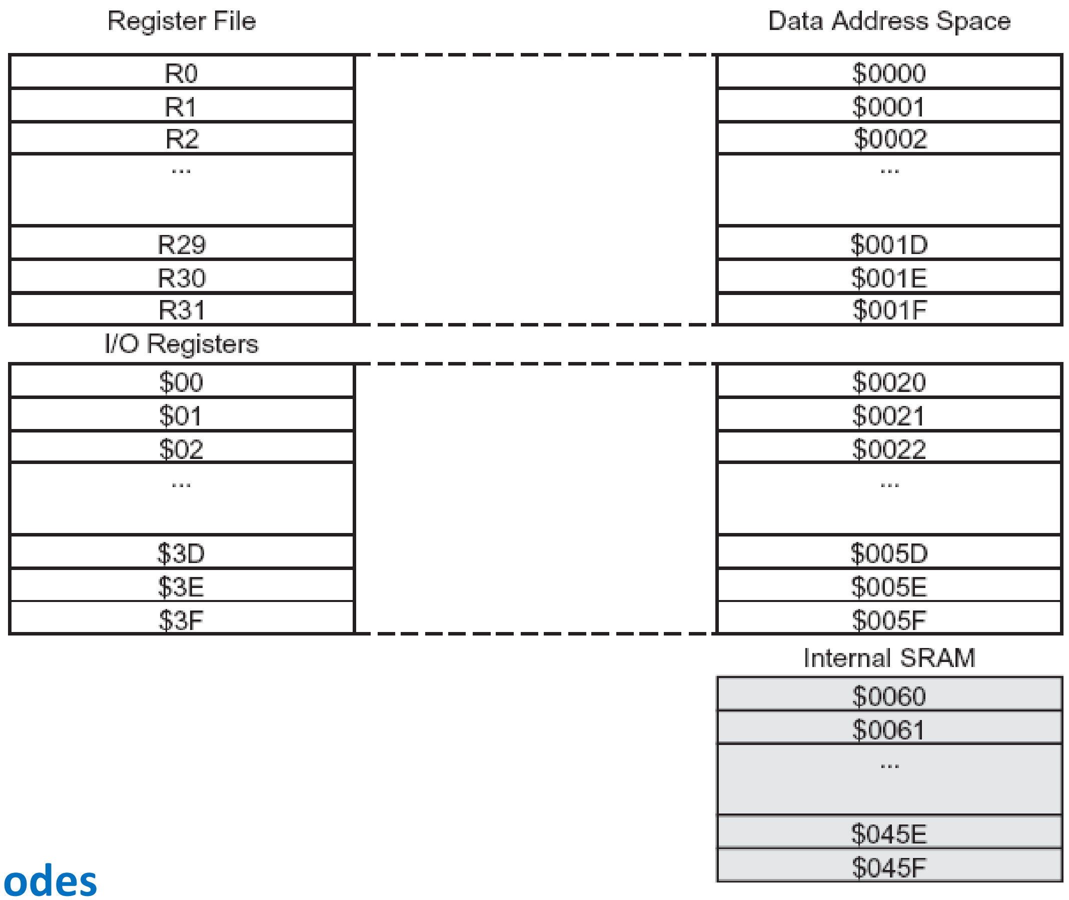
- **SRAM Data Memory:**

32 GPR's and
64 SFR's mapped
to SRAM memory space

SFR's accessed
via in / out instructions
(I/O-registers)

1 Kbytes of internal
SRAM can be accessed
from address 0x060
to address 0x45f

5 Direct and indirect addressing modes



II: Atmega

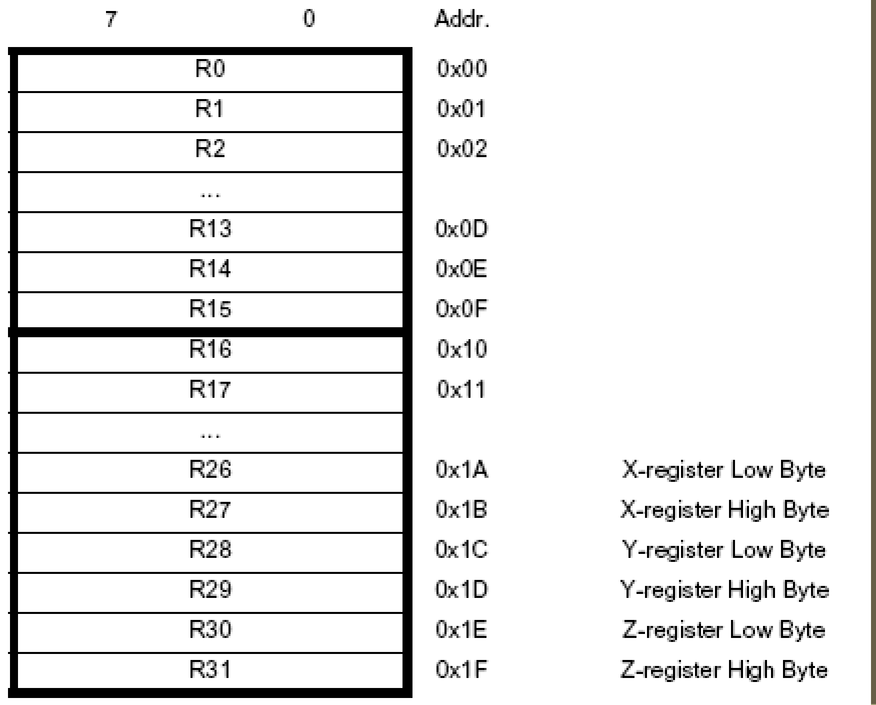
AVR Memory organization:

● **General Purpose Registers:**

Although not being physically
implemented as SRAM locations,
GPR's can be accessed
by SRAM locations

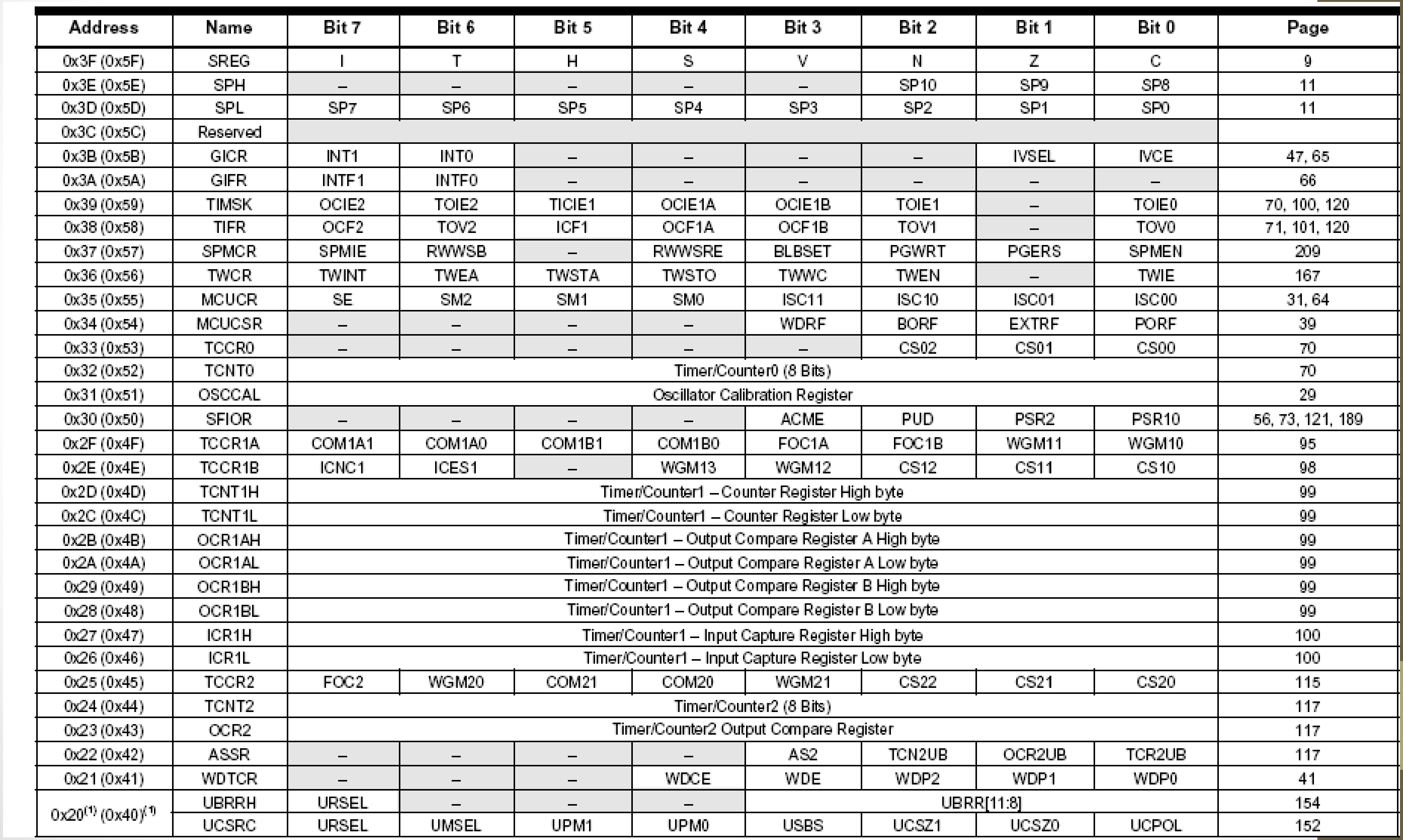
X, Y and Z 16-bit registers
can be used for indirect addressing

ALU - Input / output schemes:
one 8-bit operand, 8-bit result
two 8-bit operands, 8-bit result
two 8-bit operands, 16-bit result
one 16-bit operand, 16-bit result



7 0	Addr.	
R0	0x00	
R1	0x01	
R2	0x02	
...		
R13	0x0D	
R14	0x0E	
R15	0x0F	
R16	0x10	
R17	0x11	
...		
R26	0x1A	X-register Low Byte
R27	0x1B	X-register High Byte
R28	0x1C	Y-register Low Byte
R29	0x1D	Y-register High Byte
R30	0x1E	Z-register Low Byte
R31	0x1F	Z-register High Byte

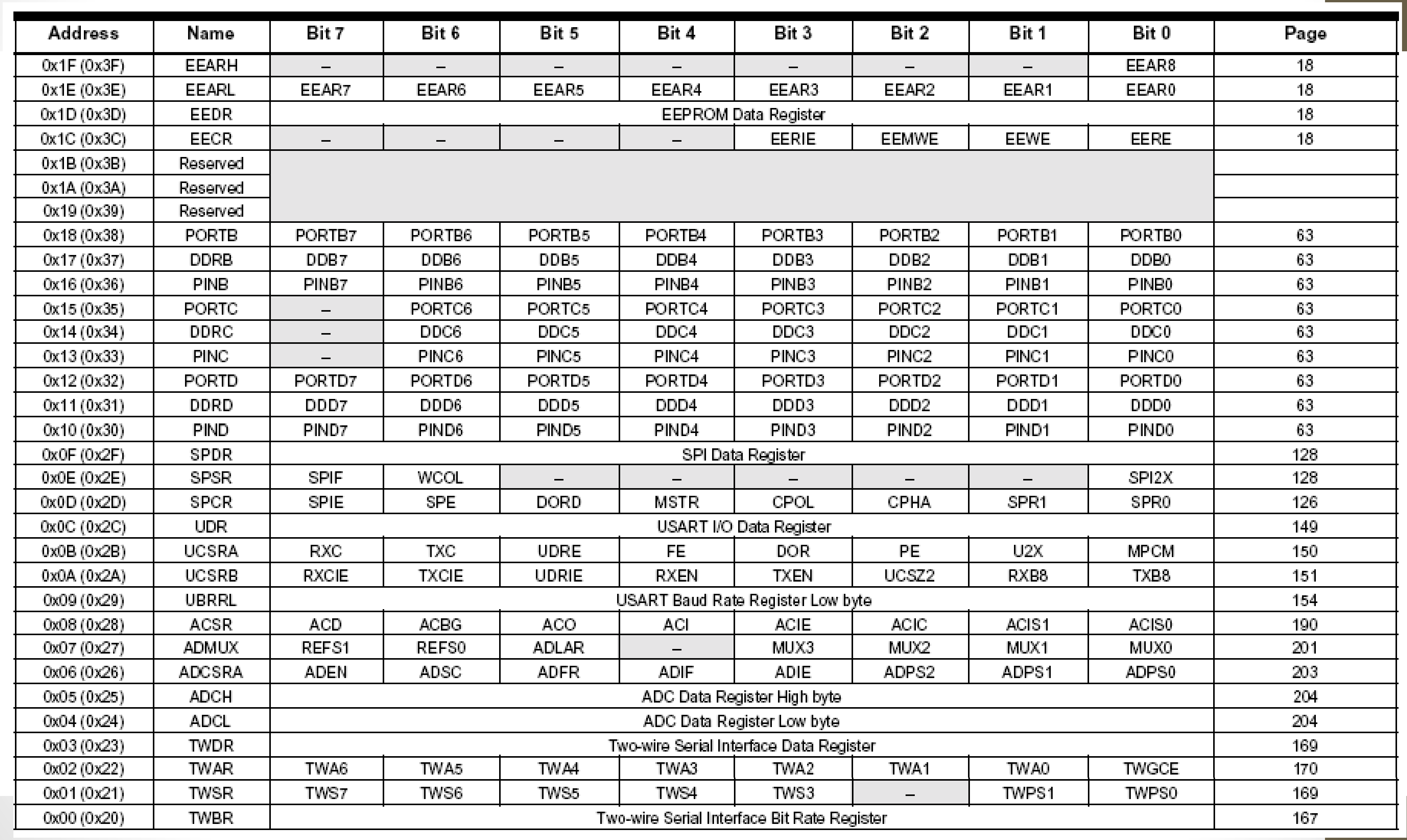
I/O Memory (SFR) Overview



Address	Name	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	Page
0x3F (0x5F)	SREG	I	T	H	S	V	N	Z	C	9
0x3E (0x5E)	SPH	–	–	–	–	–	SP10	SP9	SP8	11
0x3D (0x5D)	SPL	SP7	SP6	SP5	SP4	SP3	SP2	SP1	SP0	11
0x3C (0x5C)	Reserved									
0x3B (0x5B)	GICR	INT1	INT0	–	–	–	–	IVSEL	IVCE	47, 65
0x3A (0x5A)	GIFR	INTF1	INTF0	–	–	–	–	–	–	66
0x39 (0x59)	TIMSK	OCIE2	TOIE2	TICIE1	OCIE1A	OCIE1B	TOIE1	–	TOIE0	70, 100, 120
0x38 (0x58)	TIFR	OCF2	TOV2	ICF1	OCF1A	OCF1B	TOV1	–	TOV0	71, 101, 120
0x37 (0x57)	SPMCR	SPMIE	RWWSB	–	RWWSRE	BLBSET	PGWRT	PGERS	SPMEN	209
0x36 (0x56)	TWCR	TWINT	TWEA	TWSTA	TWSTO	TWWC	TWEN	–	TWIE	167
0x35 (0x55)	MCUCR	SE	SM2	SM1	SM0	ISC11	ISC10	ISC01	ISC00	31, 64
0x34 (0x54)	MCUCSR	–	–	–	–	WDRF	BORF	EXTRF	PORF	39
0x33 (0x53)	TCCR0	–	–	–	–	–	CS02	CS01	CS00	70
0x32 (0x52)	TCNT0	Timer/Counter0 (8 Bits)								70
0x31 (0x51)	OSCCAL	Oscillator Calibration Register								29
0x30 (0x50)	SFIOR	–	–	–	–	ACME	PUD	PSR2	PSR10	56, 73, 121, 189
0x2F (0x4F)	TCCR1A	COM1A1	COM1A0	COM1B1	COM1B0	FOC1A	FOC1B	WGM11	WGM10	95
0x2E (0x4E)	TCCR1B	ICNC1	ICES1	–	WGM13	WGM12	CS12	CS11	CS10	98
0x2D (0x4D)	TCNT1H	Timer/Counter1 – Counter Register High byte								99
0x2C (0x4C)	TCNT1L	Timer/Counter1 – Counter Register Low byte								99
0x2B (0x4B)	OCR1AH	Timer/Counter1 – Output Compare Register A High byte								99
0x2A (0x4A)	OCR1AL	Timer/Counter1 – Output Compare Register A Low byte								99
0x29 (0x49)	OCR1BH	Timer/Counter1 – Output Compare Register B High byte								99
0x28 (0x48)	OCR1BL	Timer/Counter1 – Output Compare Register B Low byte								99
0x27 (0x47)	ICR1H	Timer/Counter1 – Input Capture Register High byte								100
0x26 (0x46)	ICR1L	Timer/Counter1 – Input Capture Register Low byte								100
0x25 (0x45)	TCCR2	FOC2	WGM20	COM21	COM20	WGM21	CS22	CS21	CS20	115
0x24 (0x44)	TCNT2	Timer/Counter2 (8 Bits)								117
0x23 (0x43)	OCR2	Timer/Counter2 Output Compare Register								117
0x22 (0x42)	ASSR	–	–	–	–	AS2	TCN2UB	OCR2UB	TCR2UB	117
0x21 (0x41)	WDTCR	–	–	–	WDCE	WDE	WDP2	WDP1	WDP0	41
0x20[(1)] (0x40)[(1)]	UBRRH	URSEL	–	–	–	UBRR[11:8]				154
	UCSRC	URSEL	UMSEL	UPM1	UPM0	USBS	UCSZ1	UCSZ0	UCPOL	152

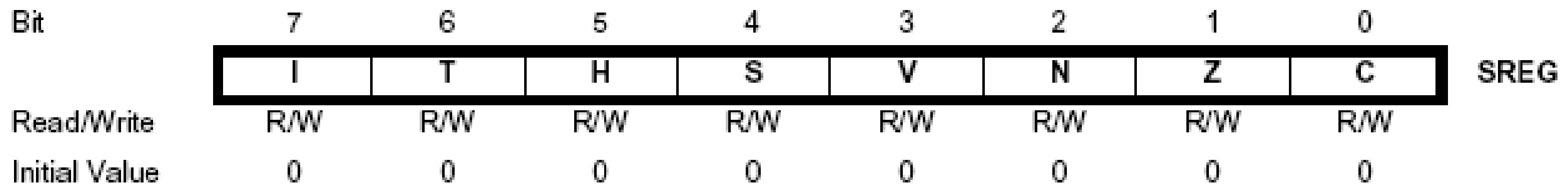
II: Atmega

I/O Memory (SFR) Overview



Address	Name	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	Page
0x1F (0x3F)	EEARH	–	–	–	–	–	–	–	EEAR8	18
0x1E (0x3E)	EEARL	EEAR7	EEAR6	EEAR5	EEAR4	EEAR3	EEAR2	EEAR1	EEAR0	18
0x1D (0x3D)	EEDR	EEPROM Data Register								18
0x1C (0x3C)	EECR	–	–	–	–	EERIE	EEMWE	EEWE	EERE	18
0x1B (0x3B)	Reserved									
0x1A (0x3A)	Reserved									
0x19 (0x39)	Reserved									
0x18 (0x38)	PORTB	PORTB7	PORTB6	PORTB5	PORTB4	PORTB3	PORTB2	PORTB1	PORTB0	63
0x17 (0x37)	DDRB	DDB7	DDB6	DDB5	DDB4	DDB3	DDB2	DDB1	DDB0	63
0x16 (0x36)	PINB	PINB7	PINB6	PINB5	PINB4	PINB3	PINB2	PINB1	PINB0	63
0x15 (0x35)	PORTC	–	PORTC6	PORTC5	PORTC4	PORTC3	PORTC2	PORTC1	PORTC0	63
0x14 (0x34)	DDRC	–	DDC6	DDC5	DDC4	DDC3	DDC2	DDC1	DDC0	63
0x13 (0x33)	PINC	–	PINC6	PINC5	PINC4	PINC3	PINC2	PINC1	PINC0	63
0x12 (0x32)	PORTD	PORTD7	PORTD6	PORTD5	PORTD4	PORTD3	PORTD2	PORTD1	PORTD0	63
0x11 (0x31)	DDRD	DDD7	DDD6	DDD5	DDD4	DDD3	DDD2	DDD1	DDD0	63
0x10 (0x30)	PIND	PIND7	PIND6	PIND5	PIND4	PIND3	PIND2	PIND1	PIND0	63
0x0F (0x2F)	SPDR	SPI Data Register								128
0x0E (0x2E)	SPSR	SPIF	WCOL	–	–	–	–	–	SPI2X	128
0x0D (0x2D)	SPCR	SPIE	SPE	DORD	MSTR	CPOL	CPHA	SPR1	SPR0	126
0x0C (0x2C)	UDR	USART I/O Data Register								149
0x0B (0x2B)	UCSRA	RXC	TXC	UDRE	FE	DOR	PE	U2X	MPCM	150
0x0A (0x2A)	UCSRB	RXCIE	TXCIE	UDRIE	RXEN	TXEN	UCSZ2	RXB8	TXB8	151
0x09 (0x29)	UBRRL	USART Baud Rate Register Low byte								154
0x08 (0x28)	ACSR	ACD	ACBG	ACO	ACI	ACIE	ACIC	ACIS1	ACIS0	190
0x07 (0x27)	ADMUX	REFS1	REFS0	ADLAR	–	MUX3	MUX2	MUX1	MUX0	201
0x06 (0x26)	ADCSRA	ADEN	ADSC	ADFR	ADIF	ADIE	ADPS2	ADPS1	ADPS0	203
0x05 (0x25)	ADCH	ADC Data Register High byte								204
0x04 (0x24)	ADCL	ADC Data Register Low byte								204
0x03 (0x23)	TWDR	Two-wire Serial Interface Data Register								169
0x02 (0x22)	TWAR	TWA6	TWA5	TWA4	TWA3	TWA2	TWA1	TWA0	TWGCE	170
0x01 (0x21)	TWSR	TWS7	TWS6	TWS5	TWS4	TWS3	–	TWPS1	TWPS0	169
0x00 (0x20)	TWBR	Two-wire Serial Interface Bit Rate Register								167

Important I/O Registers:
SREG – Status Register



Bit	7	6	5	4	3	2	1	0	
	I	T	H	S	V	N	Z	C	SREG
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	

Bit 7 – I: Global Interrupt Enable

Bit 6 – T: Bit Copy Storage

Bit 5 – H: Half Carry Flag

Bit 4 – S: Sign Bit

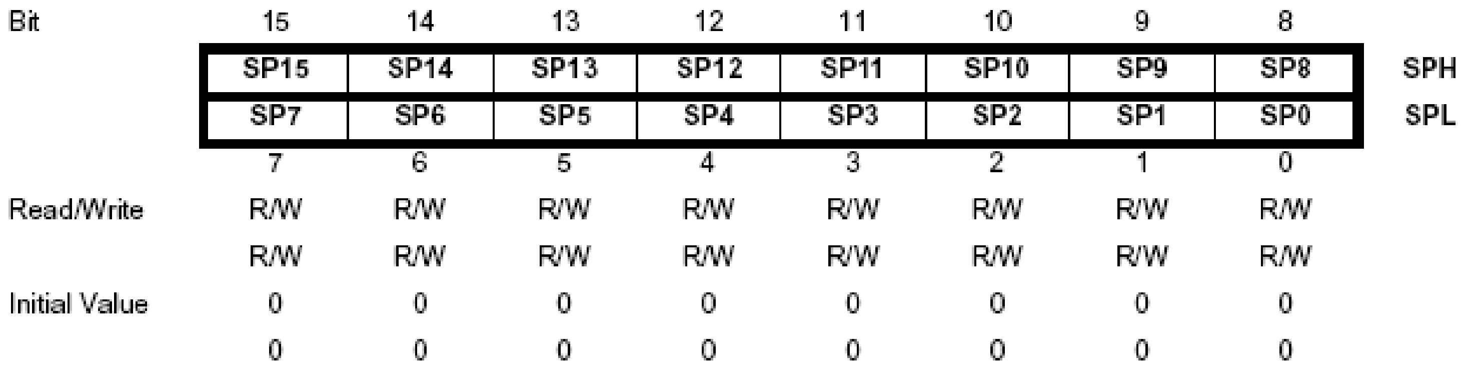
Bit 3 – V: Two’s Complement Overflow Flag

Bit 2 – N: Negative Flag

Bit 1 – Z: Zero Flag

Bit 0 – C: Carry Flag

Important I/O Registers:
Stack Pointer (SPH and SPL)



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	

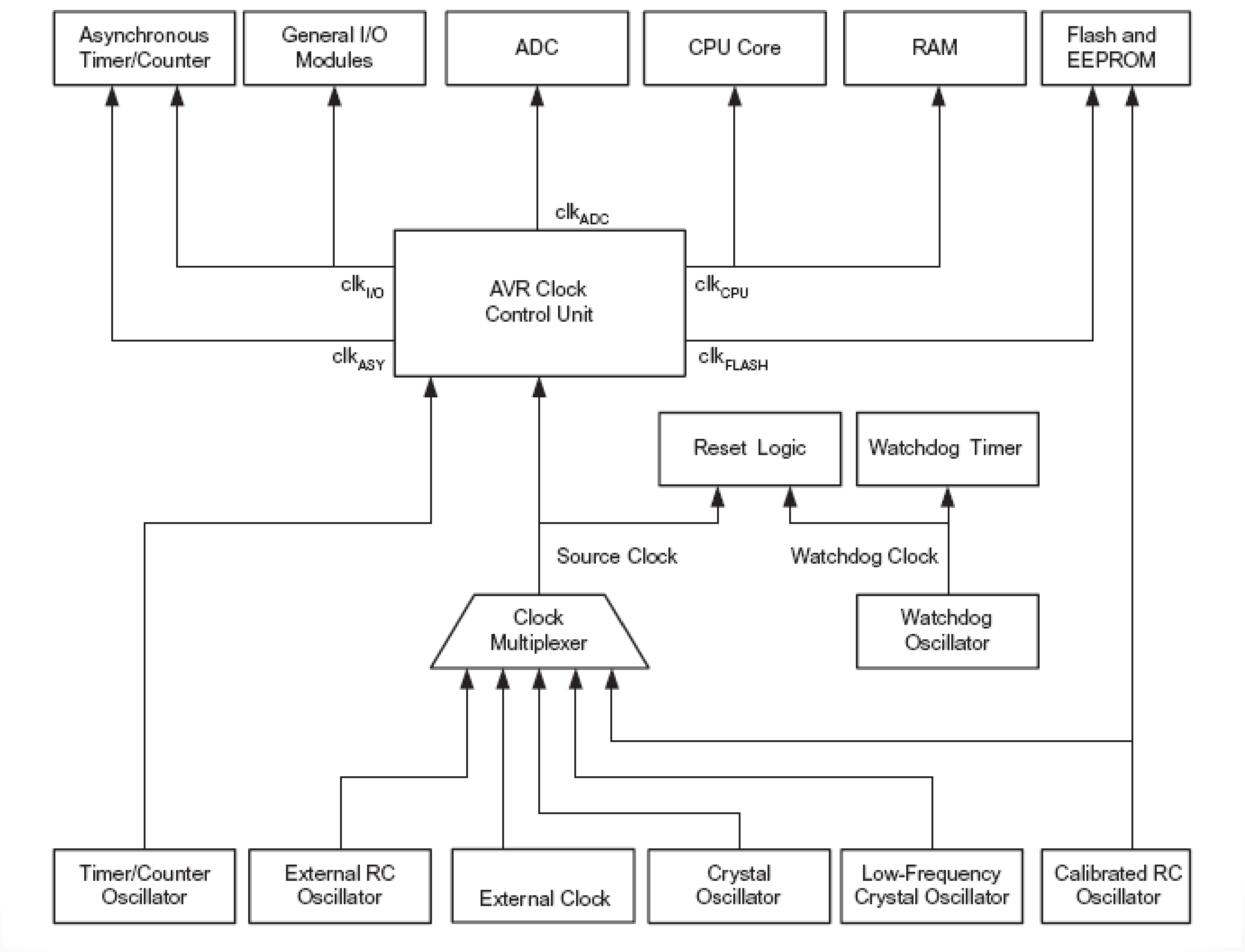
- Stack is a LIFO buffer located in SRAM
- Stack Pointer addresses the current location
- Push and pop instructions write / read from Stack
- Enter or return from subroutines / interrupt routines:

  Address and Parameters transferred via Stack

Clock Options

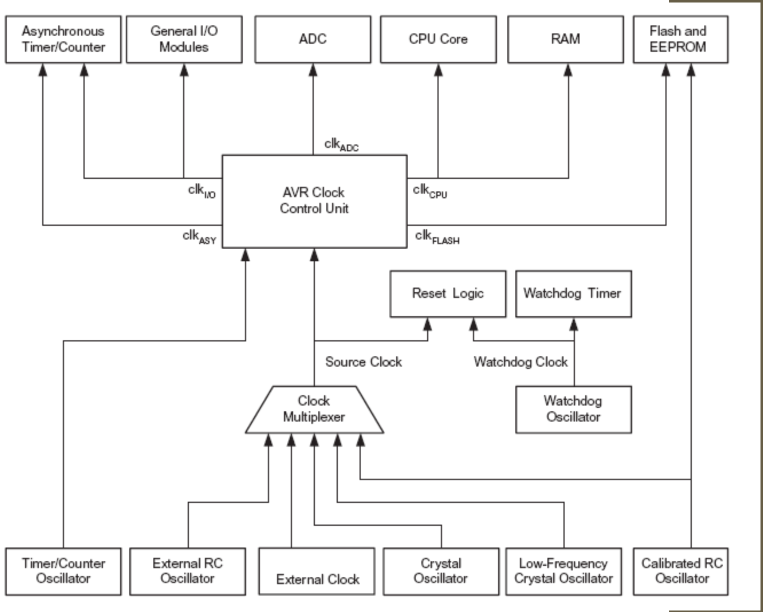
II: Atmega

System Clock Options:

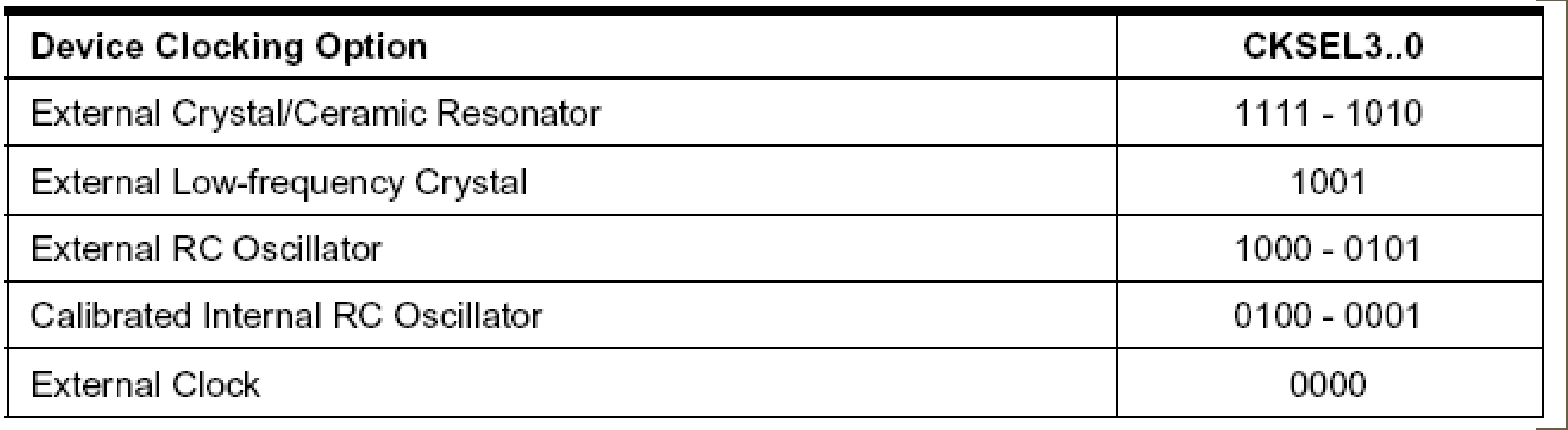


System Clock Options:

- Clock Muliplexer selects the clock source according to FUSE settings
- Clock Control Unit distributes clocks clocks can be halted to reduce power consumption
- CPU Clock: CPU, ALU, GPRs
- I/O Clock: Ports, Timers, SPI, UART
- ADC Clock: seperate cock for ADC noise reduction in sleep mode
- Asynchronous Timer Clock: external 32kHz Crystal for realtime clock, keeps timer module running during sleep mode



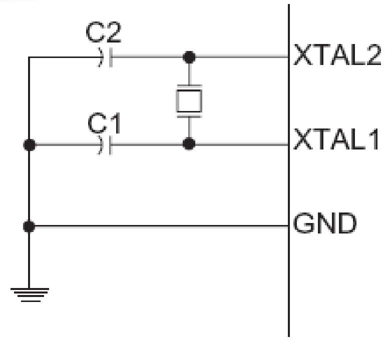
System Clock Options - FUSE bits:

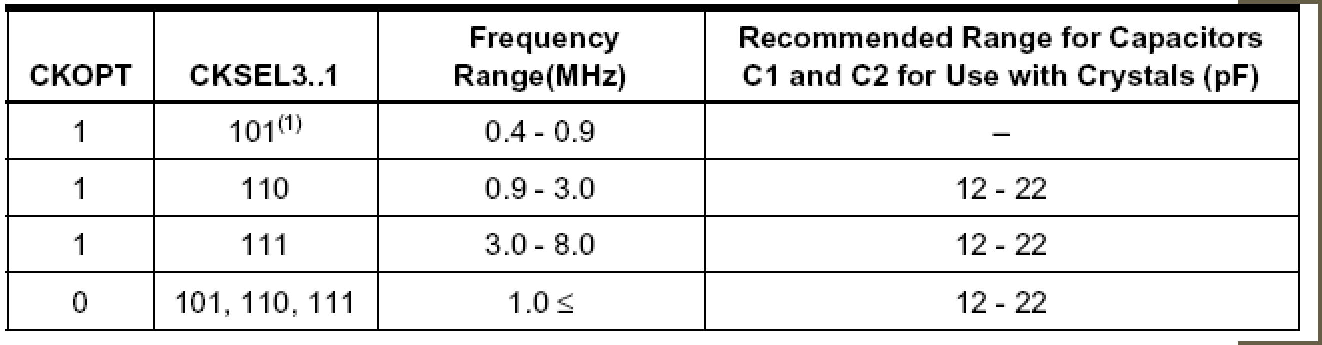


Device Clocking Option	CKSEL3..0
External Crystal/Ceramic Resonator	1111 - 1010
External Low-frequency Crystal	1001
External RC Oscillator	1000 - 0101
Calibrated Internal RC Oscillator	0100 - 0001
External Clock	0000

- The four CKSEL Bits of the FUSE – Byte select the main Clock Source
- The startup time to stabilize power supply and oscillator can be changed with the SUT fuses
- The device is shipped with CKSEL = 0001 (1 MHZ internal RC oscillator) and SUT = 10 (slowly rising power, 65ms)

System Clock Options - using an external crystal:





CKOPT	CKSEL3..1	Frequency Range(MHz)	Recommended Range for Capacitors C1 and C2 for Use with Crystals (pF)
1	101[1]	0.4 - 0.9	–
1	110	0.9 - 3.0	12 - 22
1	111	3.0 - 8.0	12 - 22
0	101, 110, 111	1.0 ≤	12 - 22

- CKOPT influences the output swing of the inverting oscillator amplifier (1 = full rail to rail swing, 0 = power save mode)
- For crystals from 3 – 8 MHz set CKOPT = 1 and CKSEL3..1 = 111

System Clock Options - using the internal RC oscillator

CKSEL3..0	Nominal Frequency (MHz)
0001[1]	1.0
0010	2.0
0011	4.0
0100	8.0

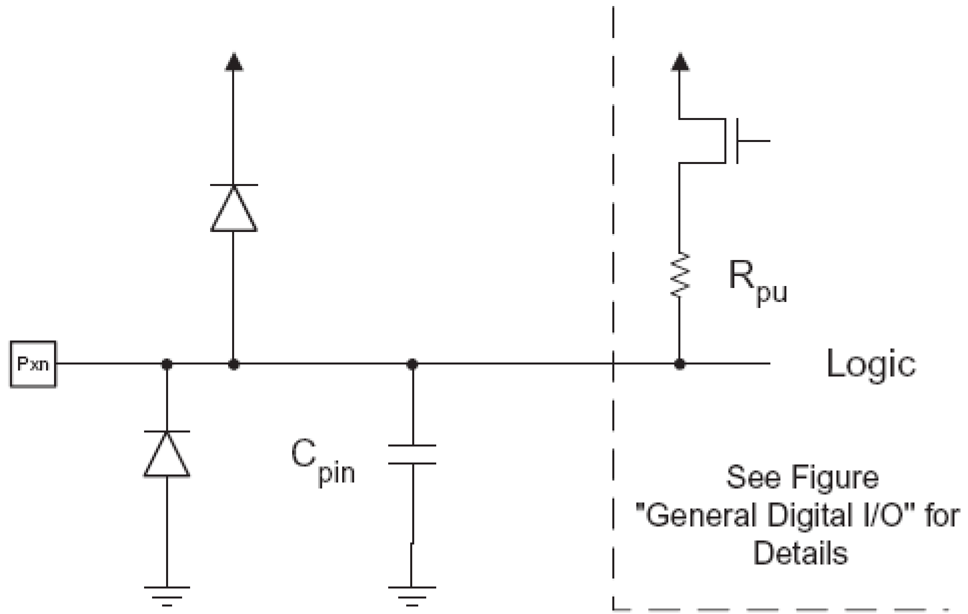
- Fixed 1, 2, 4 or 8 MHz clock
- works without external components
- changes with temperature and operating voltage

detailed information on other clock options, startup times, calibration is found in the ATmega8 data sheet, pp. 23

I/O Ports

II: Atmega

I/O Ports



- General Purpose IO : Data Direction Input or Output
- Internal Pullup can be used for Input Pins
- Output driver can source 20mA current
- protection diodes to GND and VCC

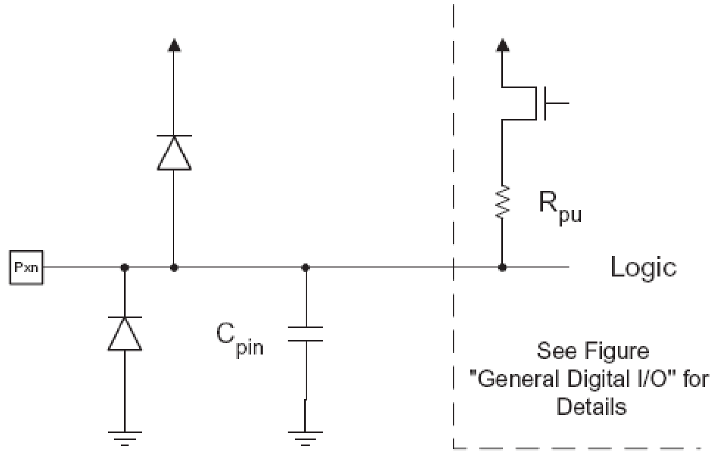
I/O Ports

- 3 I/O-Registers for each port:

  Data Register (r/w):
  PORTB, PORTC, PORTD

  Data Direction Register (r/w):
  DDRB, DDRC, DDRD

  Port Input Pin Register (r):
  PINB, PINC, PIND



The Bits of these registers set the configuration for one Port Pin.

II: Atmega

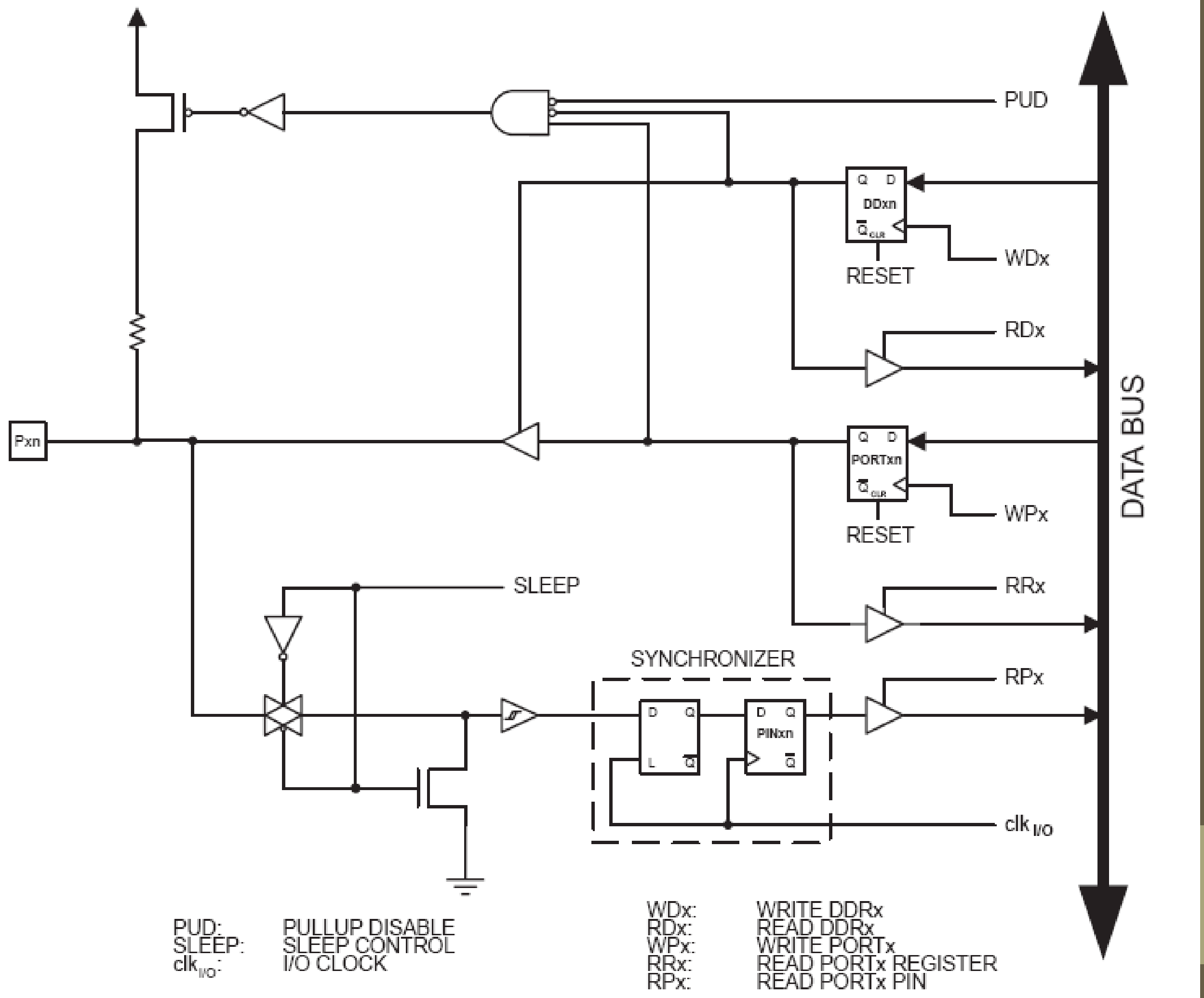
I/O Ports

General Digital IO

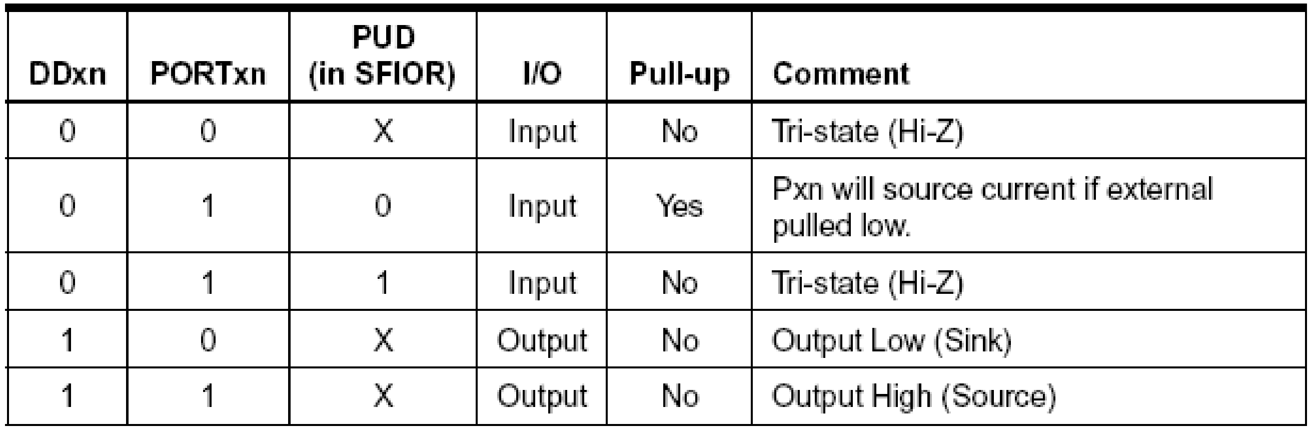
Logic of GPIO-Ports:

DDx
PORTx
PINx

Common to all Ports:
Pullup disable (PUD),
SLEEP



I/O Ports – Configuration and usage



DDxn	PORTxn	PUD (in SFIOR)	I/O	Pull-up	Comment
0	0	X	Input	No	Tri-state (Hi-Z)
0	1	0	Input	Yes	Pxn will source current if external pulled low.
0	1	1	Input	No	Tri-state (Hi-Z)
1	0	X	Output	No	Output Low (Sink)
1	1	X	Output	No	Output High (Source)

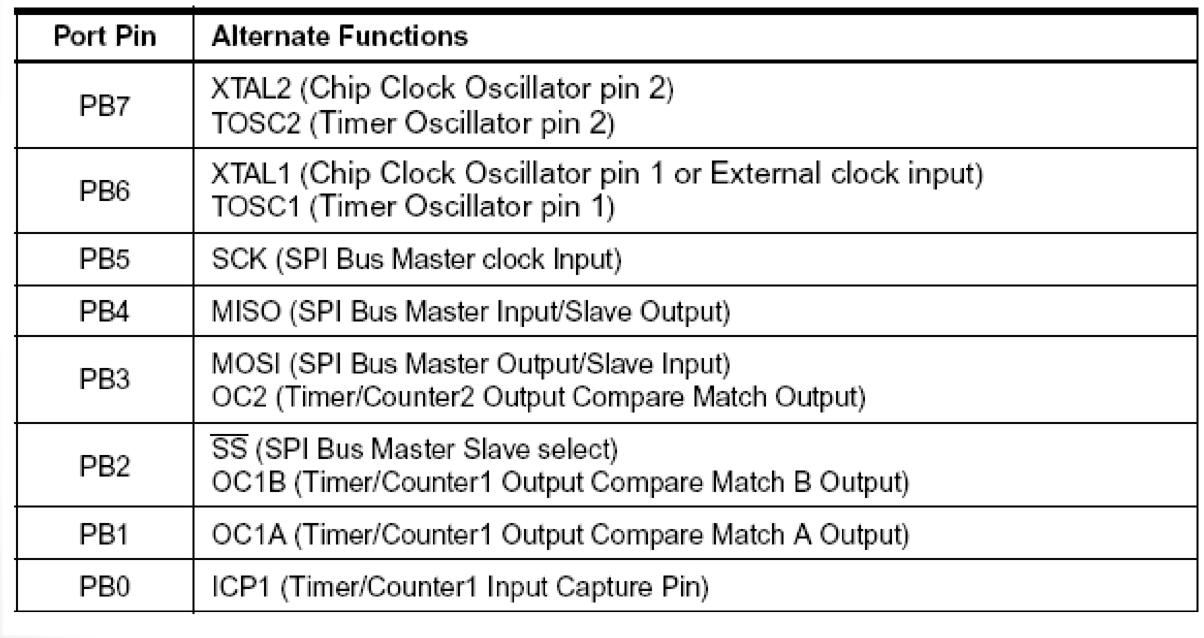
C-Example 1 - Configure Pin B3 as output, set output level to VCC:

```
DDRB |= (1<<3);  PORTB |= (1<<3);
```

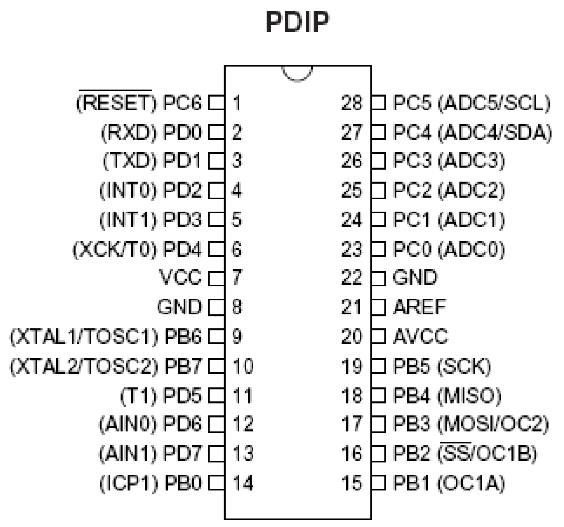
C-Example 2 - Configure Pin D2 as input with pullup, read pin value:

```
DDRD &= ~(1<<2);  PORTD |= (1<<2); uint8_t x = PIND & (1<<2);
```

Alternate Port functions Port B

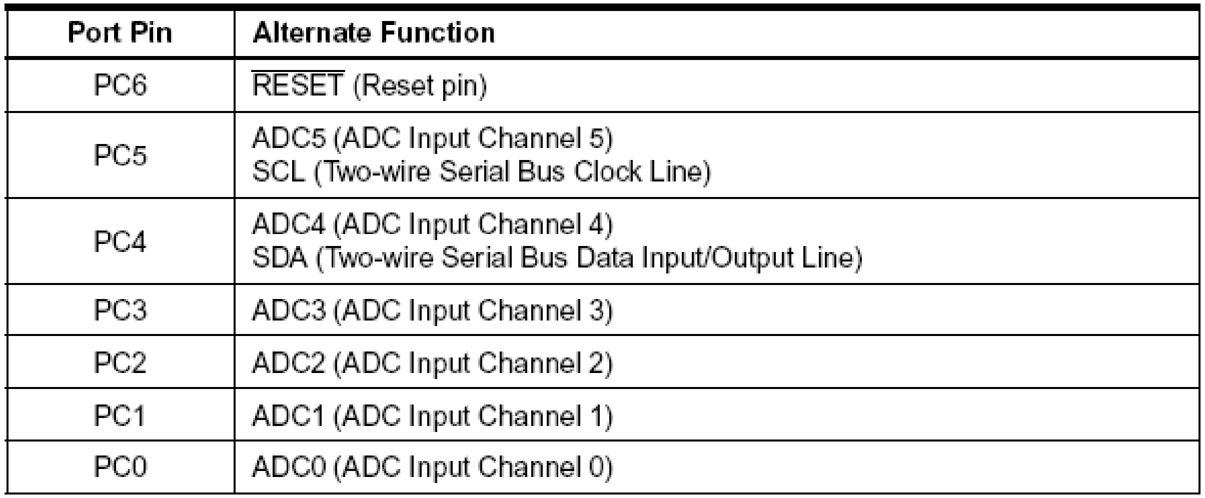


Port Pin	Alternate Functions
PB7	XTAL2 (Chip Clock Oscillator pin 2) TOSC2 (Timer Oscillator pin 2)
PB6	XTAL1 (Chip Clock Oscillator pin 1 or External clock input) TOSC1 (Timer Oscillator pin 1)
PB5	SCK (SPI Bus Master clock Input)
PB4	MISO (SPI Bus Master Input/Slave Output)
PB3	MOSI (SPI Bus Master Output/Slave Input) OC2 (Timer/Counter2 Output Compare Match Output)
PB2	$\overline{\text{SS}}$ (SPI Bus Master Slave select) OC1B (Timer/Counter1 Output Compare Match B Output)
PB1	OC1A (Timer/Counter1 Output Compare Match A Output)
PB0	ICP1 (Timer/Counter1 Input Capture Pin)

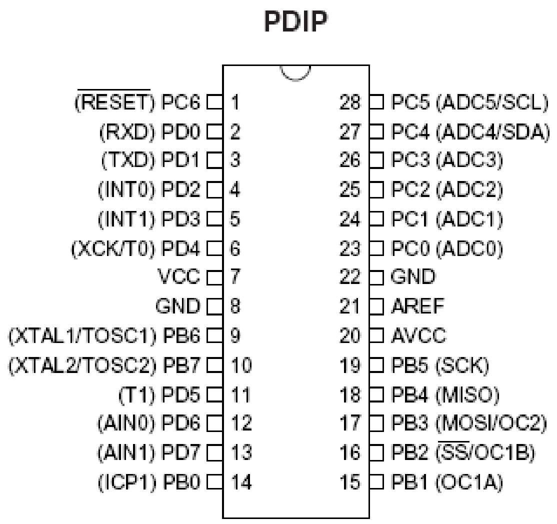


II: Atmega

Alternate Port functions Port C

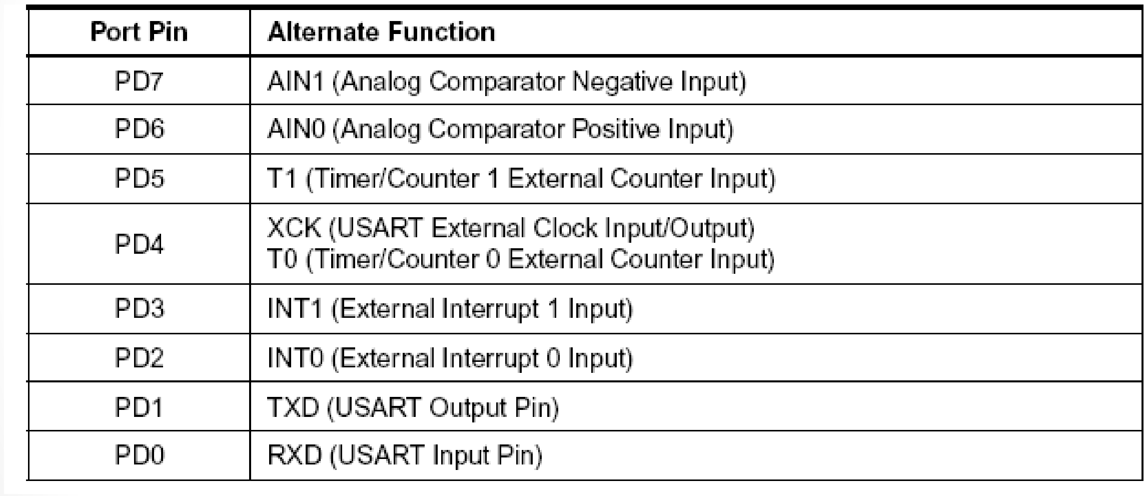


Port Pin	Alternate Function
PC6	$\overline{\text{RESET}}$ (Reset pin)
PC5	ADC5 (ADC Input Channel 5) SCL (Two-wire Serial Bus Clock Line)
PC4	ADC4 (ADC Input Channel 4) SDA (Two-wire Serial Bus Data Input/Output Line)
PC3	ADC3 (ADC Input Channel 3)
PC2	ADC2 (ADC Input Channel 2)
PC1	ADC1 (ADC Input Channel 1)
PC0	ADC0 (ADC Input Channel 0)

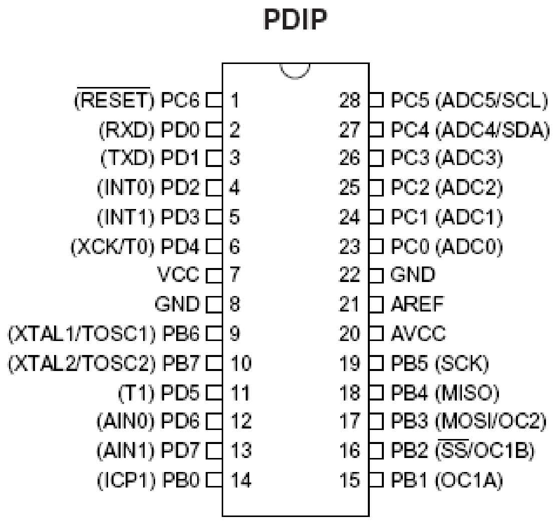


II: Atmega

Alternate Port functions Port D



Port Pin	Alternate Function
PD7	AIN1 (Analog Comparator Negative Input)
PD6	AIN0 (Analog Comparator Positive Input)
PD5	T1 (Timer/Counter 1 External Counter Input)
PD4	XCK (USART External Clock Input/Output) T0 (Timer/Counter 0 External Counter Input)
PD3	INT1 (External Interrupt 1 Input)
PD2	INT0 (External Interrupt 0 Input)
PD1	TXD (USART Output Pin)
PD0	RXD (USART Input Pin)



Reset- and Interrupt Handling

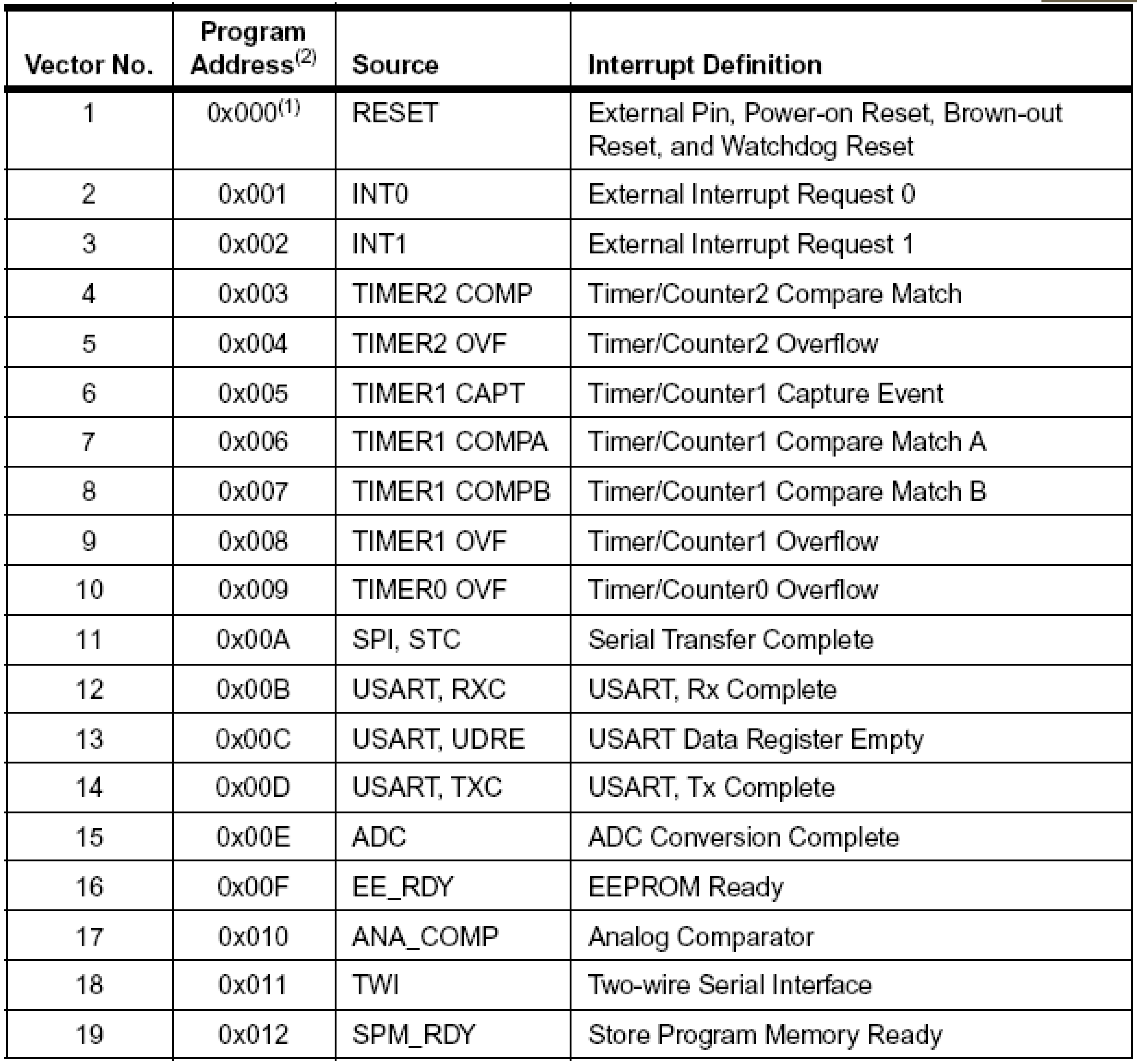
Interrupt Processing

- several Interrupt Sources:
  External Interrupts, Timer, Bus-Peripherals,
  ADC, EEPROM
- individual Interrupt-Enable bits in the SFR‘s
- global interrupt enable Bit in SREG,
  set with sei() and clear with cli() instruction
- flagged (remembered) and non-flagged interrupt sources
- lowest addresses in program memory reserved
  for the interrupt vector table
- higher priority interrupts have lower addresses

II: Atmega

Reset-Vector and Interrupt-Vectors

- Word addresses 0, 1 – 19 in Flash Ram
- When a reset or interrupt occurs, the CPU calls the address
- Install an Interrupt Handler: modify the vector table to jump to your user-handler
- return from interrupt: reti

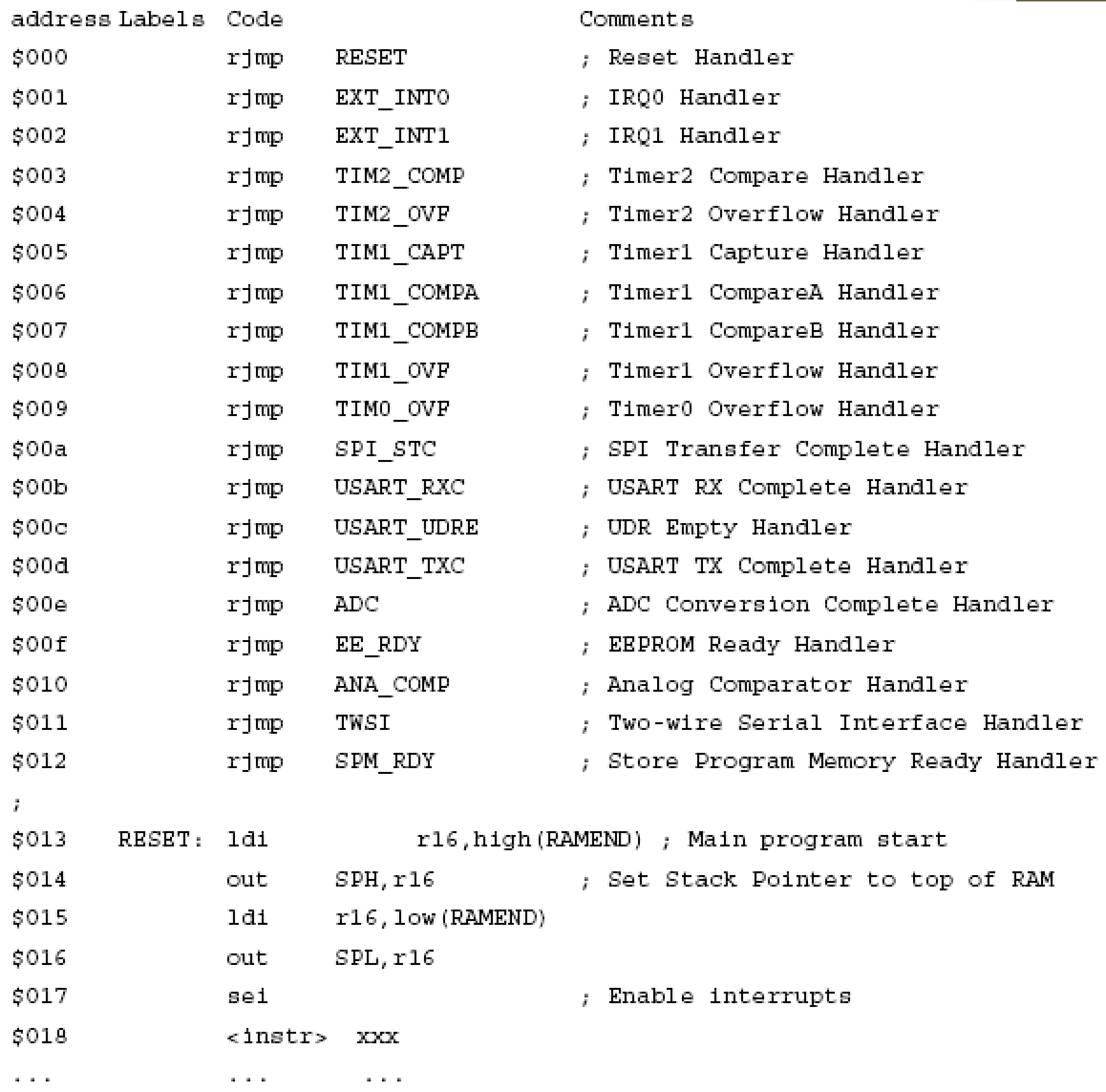


Vector No.	Program Address[2]	Source	Interrupt Definition
1	0x000[1]	RESET	External Pin, Power-on Reset, Brown-out Reset, and Watchdog Reset
2	0x001	INT0	External Interrupt Request 0
3	0x002	INT1	External Interrupt Request 1
4	0x003	TIMER2 COMP	Timer/Counter2 Compare Match
5	0x004	TIMER2 OVF	Timer/Counter2 Overflow
6	0x005	TIMER1 CAPT	Timer/Counter1 Capture Event
7	0x006	TIMER1 COMPA	Timer/Counter1 Compare Match A
8	0x007	TIMER1 COMPB	Timer/Counter1 Compare Match B
9	0x008	TIMER1 OVF	Timer/Counter1 Overflow
10	0x009	TIMER0 OVF	Timer/Counter0 Overflow
11	0x00A	SPI, STC	Serial Transfer Complete
12	0x00B	USART, RXC	USART, Rx Complete
13	0x00C	USART, UDRE	USART Data Register Empty
14	0x00D	USART, TXC	USART, Tx Complete
15	0x00E	ADC	ADC Conversion Complete
16	0x00F	EE_RDY	EEPROM Ready
17	0x010	ANA_COMP	Analog Comparator
18	0x011	TWI	Two-wire Serial Interface
19	0x012	SPM_RDY	Store Program Memory Ready

II: Atmega

Reset-Vector and Interrupt-Vectors

- example shows full featured vector table
- 19 handlers installed
- program execution after reset: jmp RESET ($013)
- Main program is located at $013, beyond the vectors



```
address Labels  Code                         Comments
$000            rjmp   RESET                 ; Reset Handler
$001            rjmp   EXT_INT0              ; IRQ0 Handler
$002            rjmp   EXT_INT1              ; IRQ1 Handler
$003            rjmp   TIM2_COMP             ; Timer2 Compare Handler
$004            rjmp   TIM2_OVF              ; Timer2 Overflow Handler
$005            rjmp   TIM1_CAPT             ; Timer1 Capture Handler
$006            rjmp   TIM1_COMPA            ; Timer1 CompareA Handler
$007            rjmp   TIM1_COMPB            ; Timer1 CompareB Handler
$008            rjmp   TIM1_OVF              ; Timer1 Overflow Handler
$009            rjmp   TIM0_OVF              ; Timer0 Overflow Handler
$00a            rjmp   SPI_STC               ; SPI Transfer Complete Handler
$00b            rjmp   USART_RXC             ; USART RX Complete Handler
$00c            rjmp   USART_UDRE            ; UDR Empty Handler
$00d            rjmp   USART_TXC             ; USART TX Complete Handler
$00e            rjmp   ADC                   ; ADC Conversion Complete Handler
$00f            rjmp   EE_RDY                ; EEPROM Ready Handler
$010            rjmp   ANA_COMP              ; Analog Comparator Handler
$011            rjmp   TWSI                  ; Two-wire Serial Interface Handler
$012            rjmp   SPM_RDY               ; Store Program Memory Ready Handler
;
$013    RESET:  ldi          r16,high(RAMEND) ; Main program start
$014            out    SPH,r16               ; Set Stack Pointer to top of RAM
$015            ldi    r16,low(RAMEND)
$016            out    SPL,r16
$017            sei                          ; Enable interrupts
$018            <instr>  xxx
...             ...        ...
```

II: Atmega

Reset- and Interrupt- Vectors

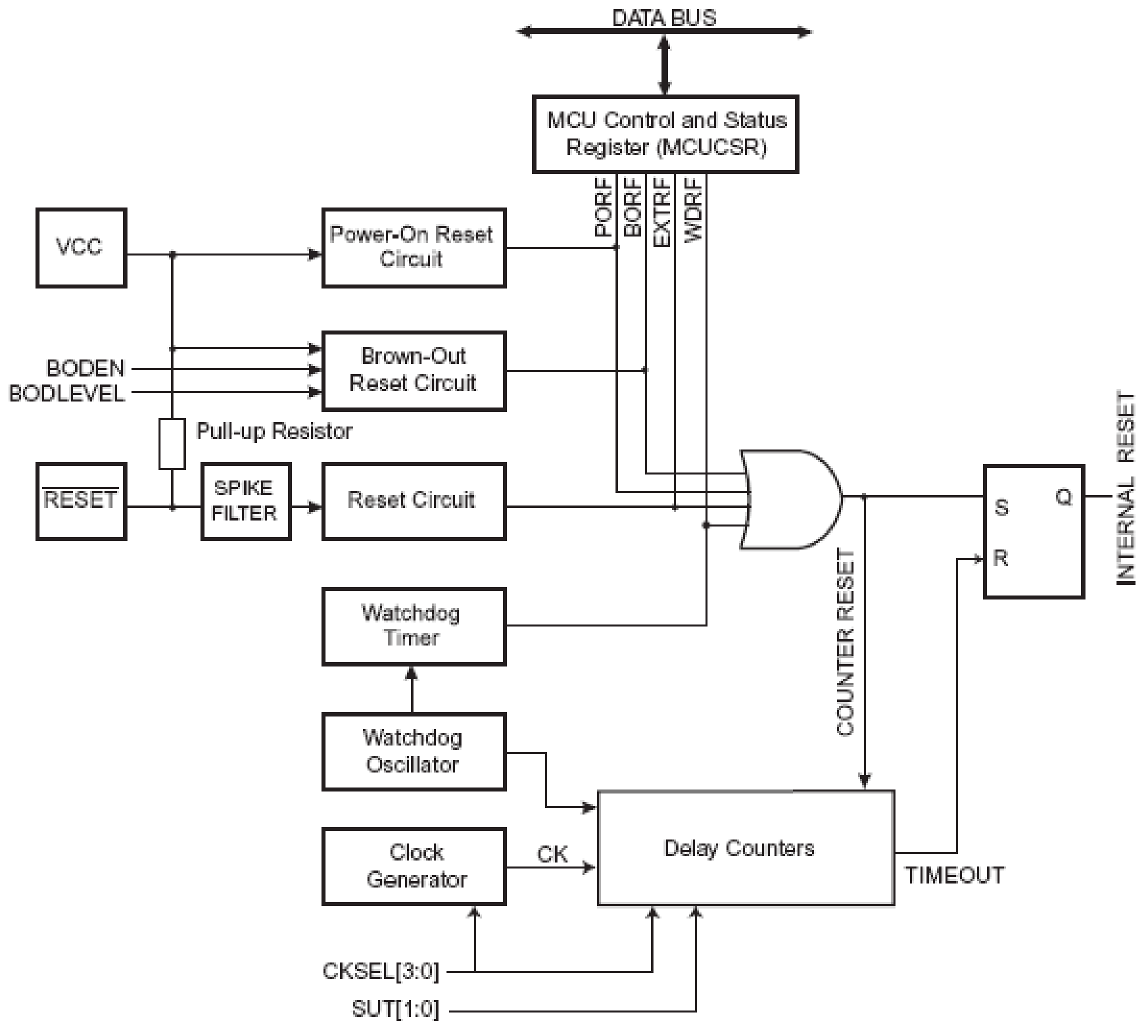
BOOTRST[1]	IVSEL	Reset Address	Interrupt Vectors Start Address
1	0	0x000	0x001
1	1	0x000	Boot Reset Address + 0x001
0	0	Boot Reset Address	0x001
0	1	Boot Reset Address	Boot Reset Address + 0x001

- Reset vector can be set to the Bootloader section using the BOOTRST fuse bit

- Interrupt vectors can be set to the Bootloader section using the IVSEL bit of the General Interrupt Contol Register

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

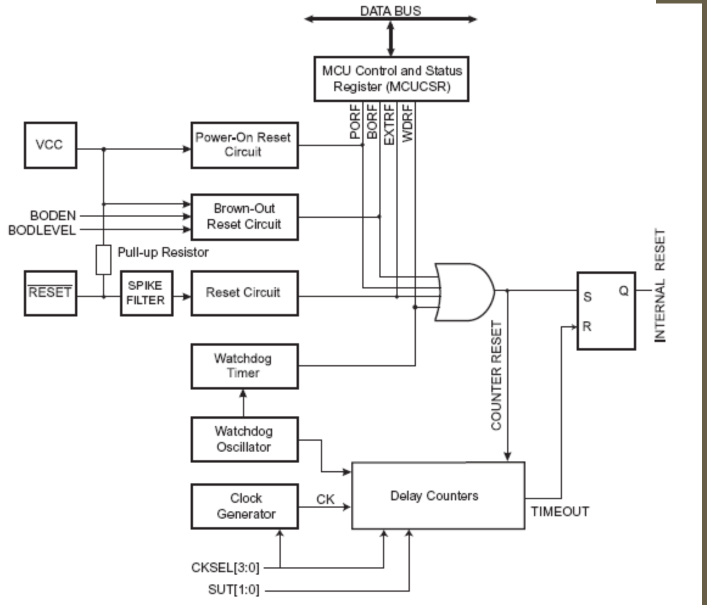
AVR Reset Sources:



II: Atmega

Reset Sources:

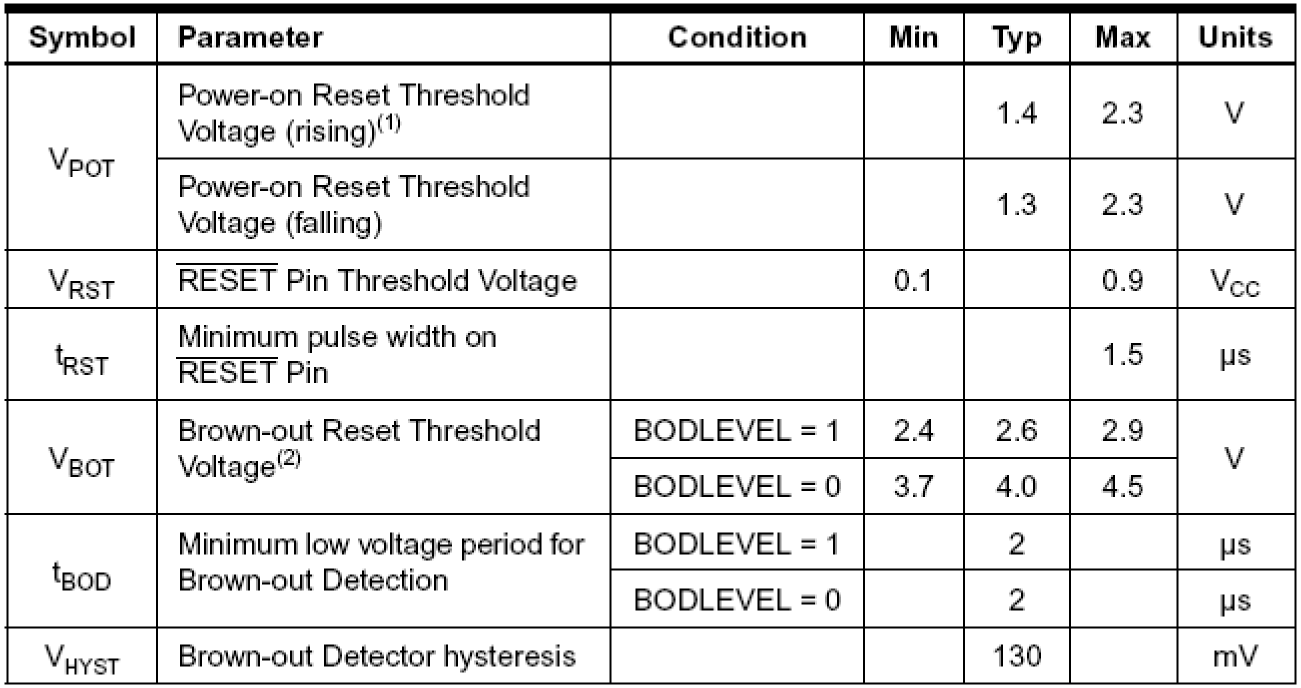
- **Power-on Reset: supply voltage is below the Power-on Reset threshold**
- **External Reset: low level is present on /RESET – input pin**
- **Watchdog Reset: Watchdog Timer enabled and period expires**
- **Brown-out Reset: Brown-out Detector enabled and supply voltage below threshold**



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

MCUCSR provides information on which reset source caused a CPU reset

Reset Voltage Thresholds

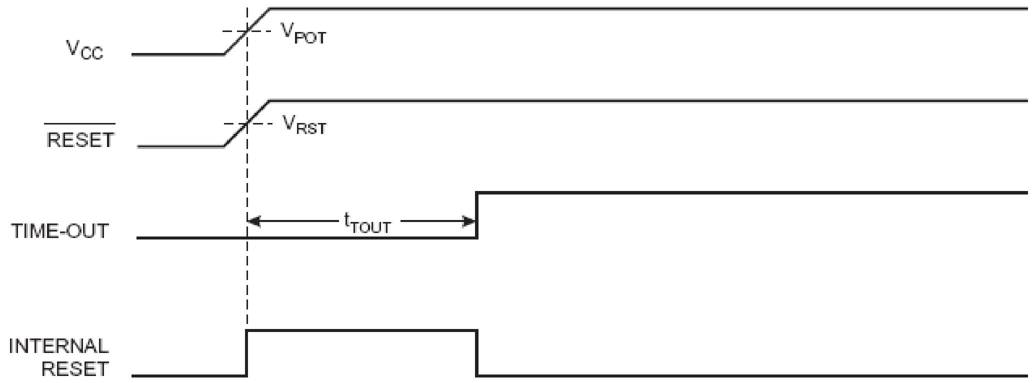


Symbol	Parameter	Condition	Min	Typ	Max	Units
V_{POT}	Power-on Reset Threshold Voltage (rising)(1)			1.4	2.3	V
	Power-on Reset Threshold Voltage (falling)			1.3	2.3	V
V_{RST}	$\overline{\text{RESET}}$ Pin Threshold Voltage		0.1		0.9	V_{CC}
t_{RST}	Minimum pulse width on $\overline{\text{RESET}}$ Pin				1.5	µs
V_{BOT}	Brown-out Reset Threshold Voltage(2)	BODLEVEL = 1	2.4	2.6	2.9	V
		BODLEVEL = 0	3.7	4.0	4.5	
t_{BOD}	Minimum low voltage period for Brown-out Detection	BODLEVEL = 1		2		µs
		BODLEVEL = 0		2		µs
V_{HYST}	Brown-out Detector hysteresis			130		mV

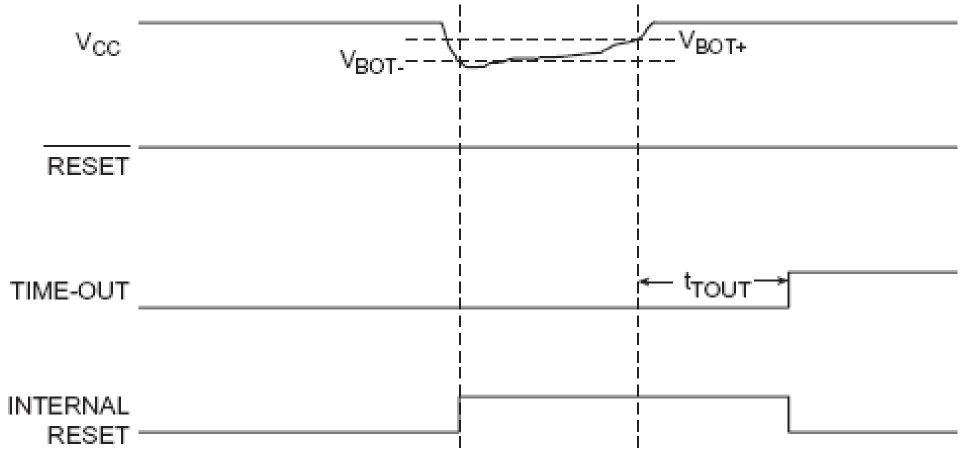
II: Atmega

Reset Voltage Thresholds:

Example:
Power-on Reset



Example:
Brown Out Reset



II: Atmega

External Interrupts Int0 and Int1:

- Int0 connected to PD2
- Int1 connected to PD3
- asynchronous operation: can wake up CPU
- rising/falling edge or low level can trigger interrupt, defined by Interrupt Sense control – bits of MCUCR SFU

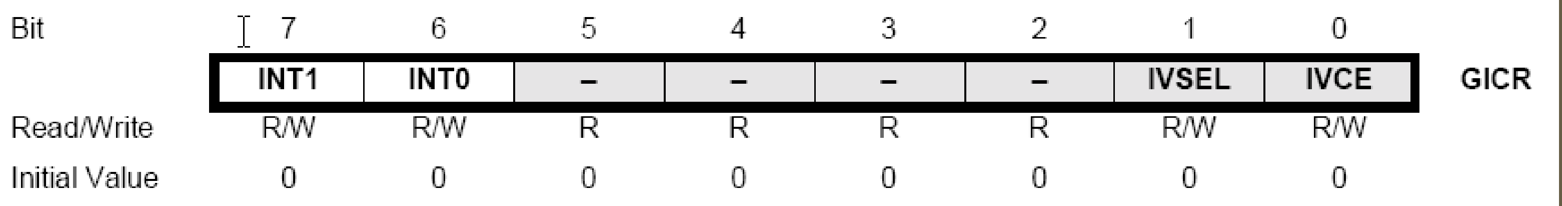
Bit	7	6	5	4	3	2	1	0	
	SE	SM2	SM1	SM0	ISC11	ISC10	ISC01	ISC00	MCUCR
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

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.

II: Atmega

External Interrupts Int0 and Int1:

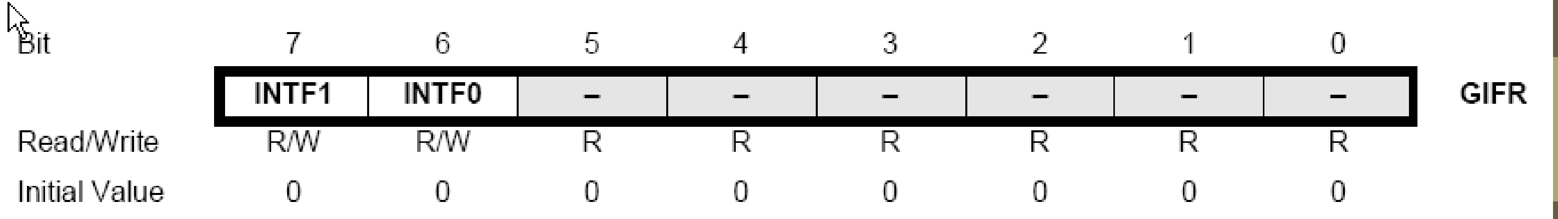
- Int0 and Int1 have to be enabled by the GICR (+ I-bit in SREG)



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

- flagged interrupts: General Interrupt Flag Register (GIFR) indicates when an interrupt request happened

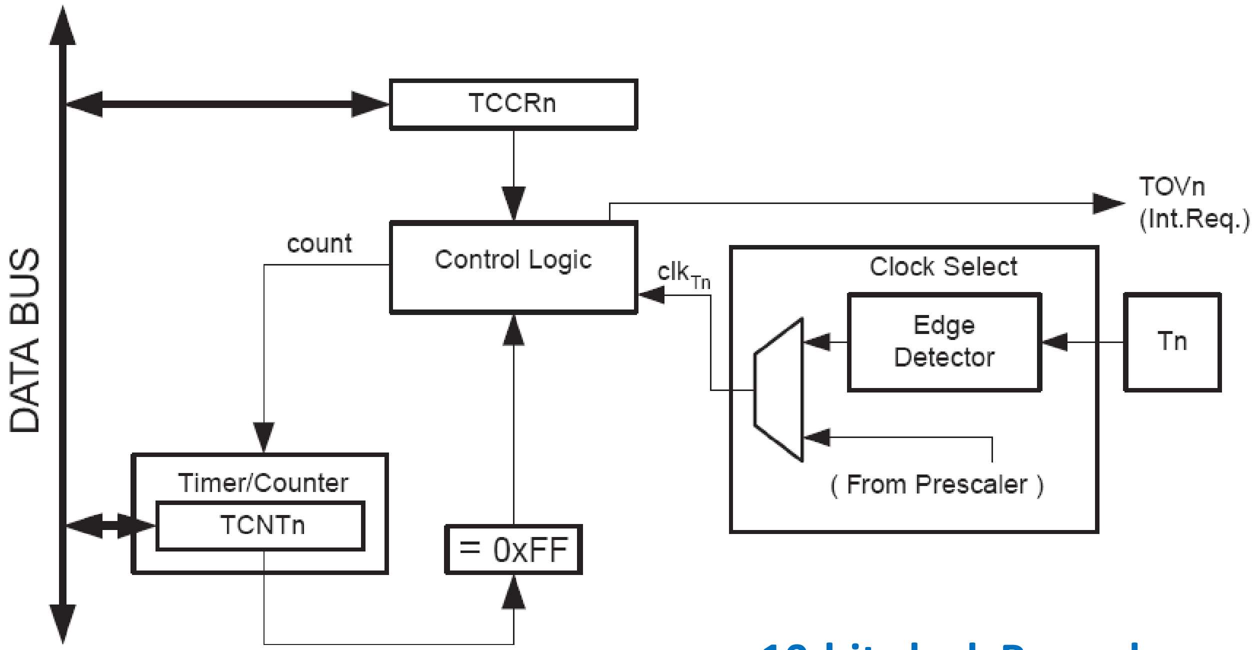
- flags are cleared by executing the interrupt service routine (ISR) or by writing 1 to the flag bit of GIFR



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

AVR TIMER/COUNTER

8-bit Timer / Counter0

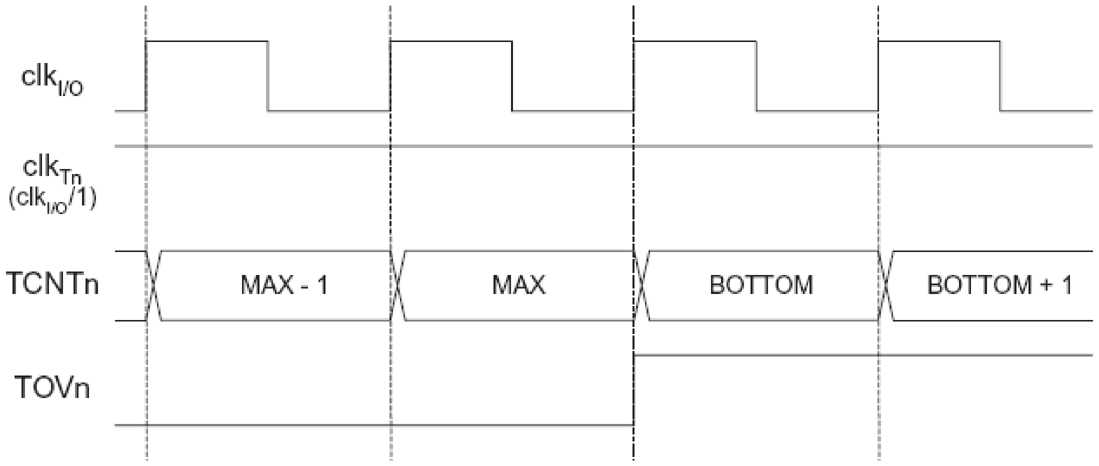


- 10-bit clock Prescaler
  timer-clk (t0) = clk (IO) / prescaler

- External clock source T0 connected to PD4
  cannot be prescaled, clk(ext) <= clk (IO) / 2.5

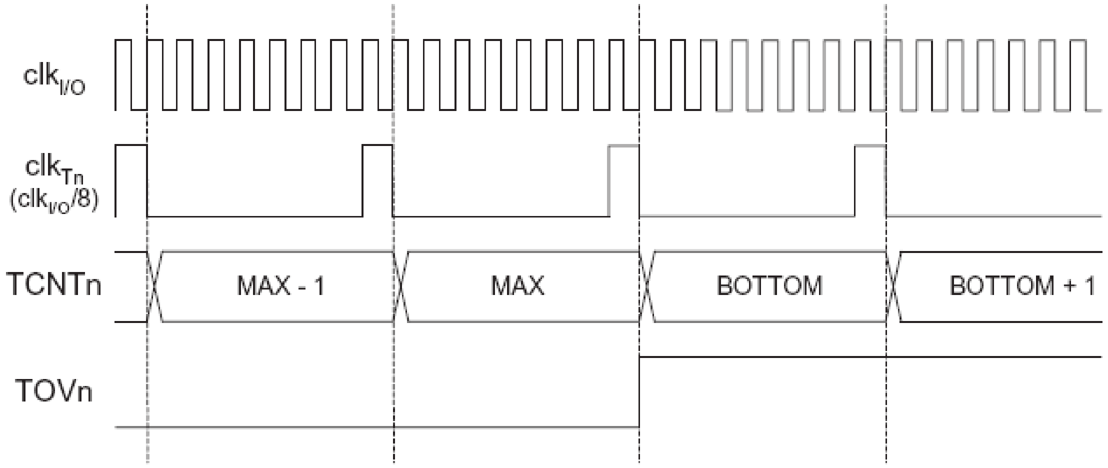
8-bit Timer / Counter0 - prescaler operation

No prescaler



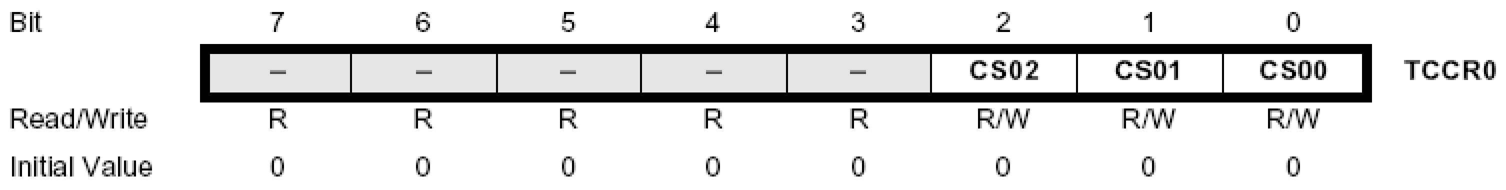
MAX=0xff
BOTTOM=0

Prescaler = 8

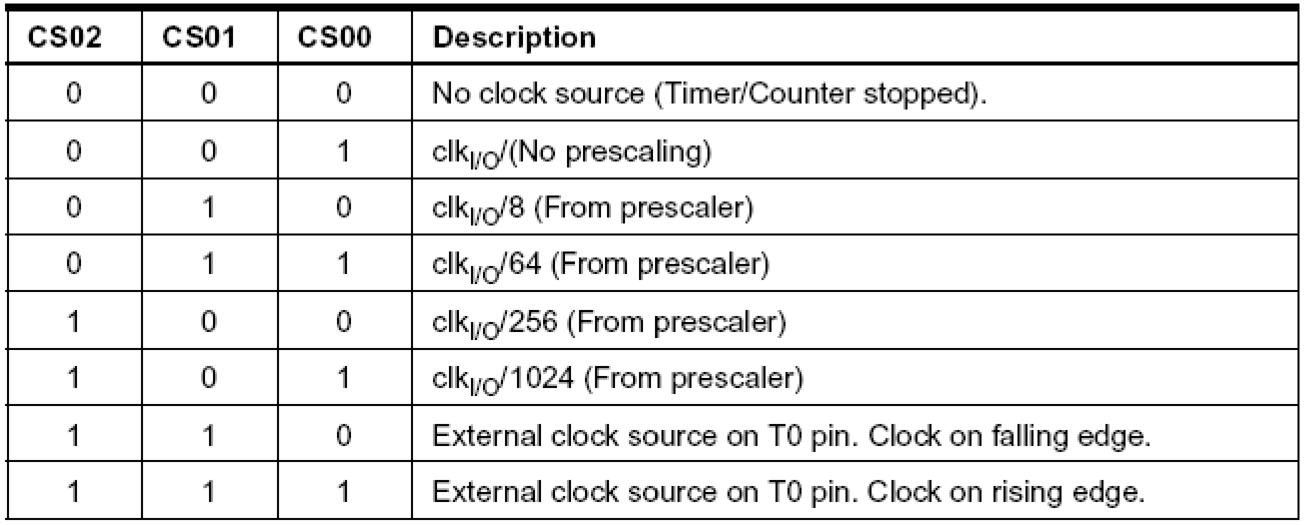


8-bit Timer / Counter0 usage

Timer/Counter0 Control Register (TCCR0), Bits CS02-CS00 select Clock Source and Prescaler Value :



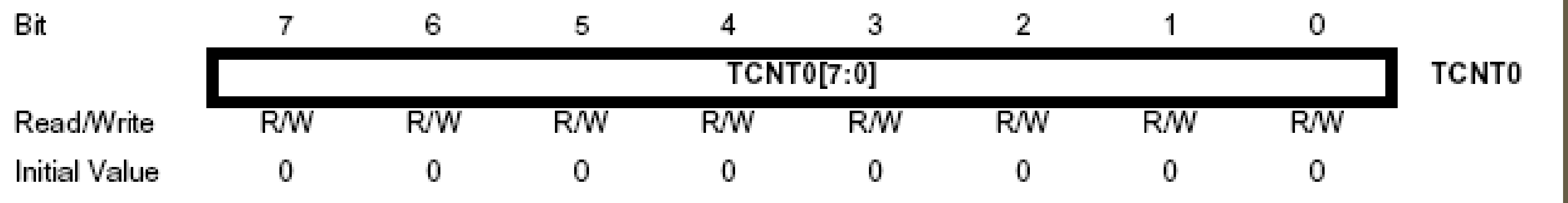
Bit	7	6	5	4	3	2	1	0	
	–	–	–	–	–	CS02	CS01	CS00	TCCR0
Read/Write	R	R	R	R	R	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	



CS02	CS01	CS00	Description
0	0	0	No clock source (Timer/Counter stopped).
0	0	1	$clk_{I/O}$/(No prescaling)
0	1	0	$clk_{I/O}$/8 (From prescaler)
0	1	1	$clk_{I/O}$/64 (From prescaler)
1	0	0	$clk_{I/O}$/256 (From prescaler)
1	0	1	$clk_{I/O}$/1024 (From prescaler)
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.

8-bit Timer / Counter0 usage

Timer/Counter0 Register (TCNT0) :
read/write, incremented per CLK cycle, overflow: 0xff

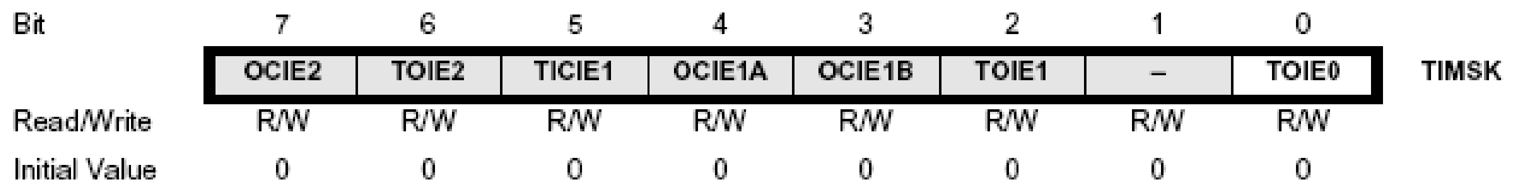


Bit	7	6	5	4	3	2	1	0	
	TCNT0[7:0]								TCNT0
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

- **A Reload-Value is used to fine-tune the interrupt interval**
- **write Reload-Value to TCNT0 in the ISR**

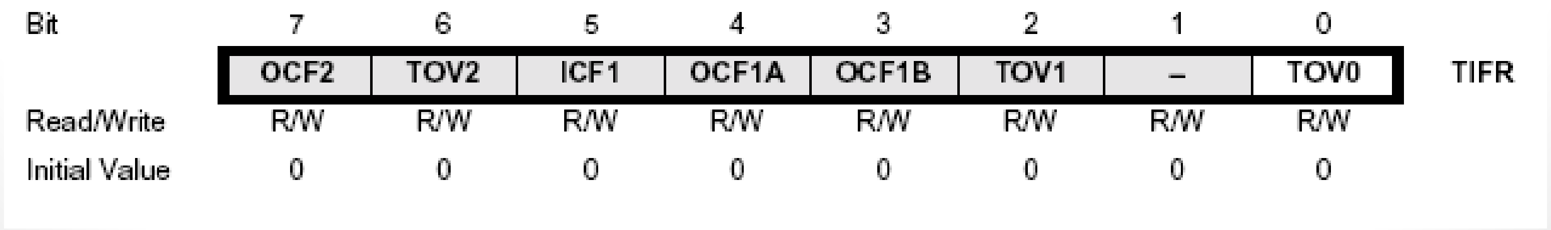
8-bit Timer / Counter0 usage

Timer/Counter Interrupt Mask Register (TIMSK) :
Bit 0 : Timer 0 interrupt enable
set 1 to enable timer 0 overflow interrupt (+ I-Bit in SREG)



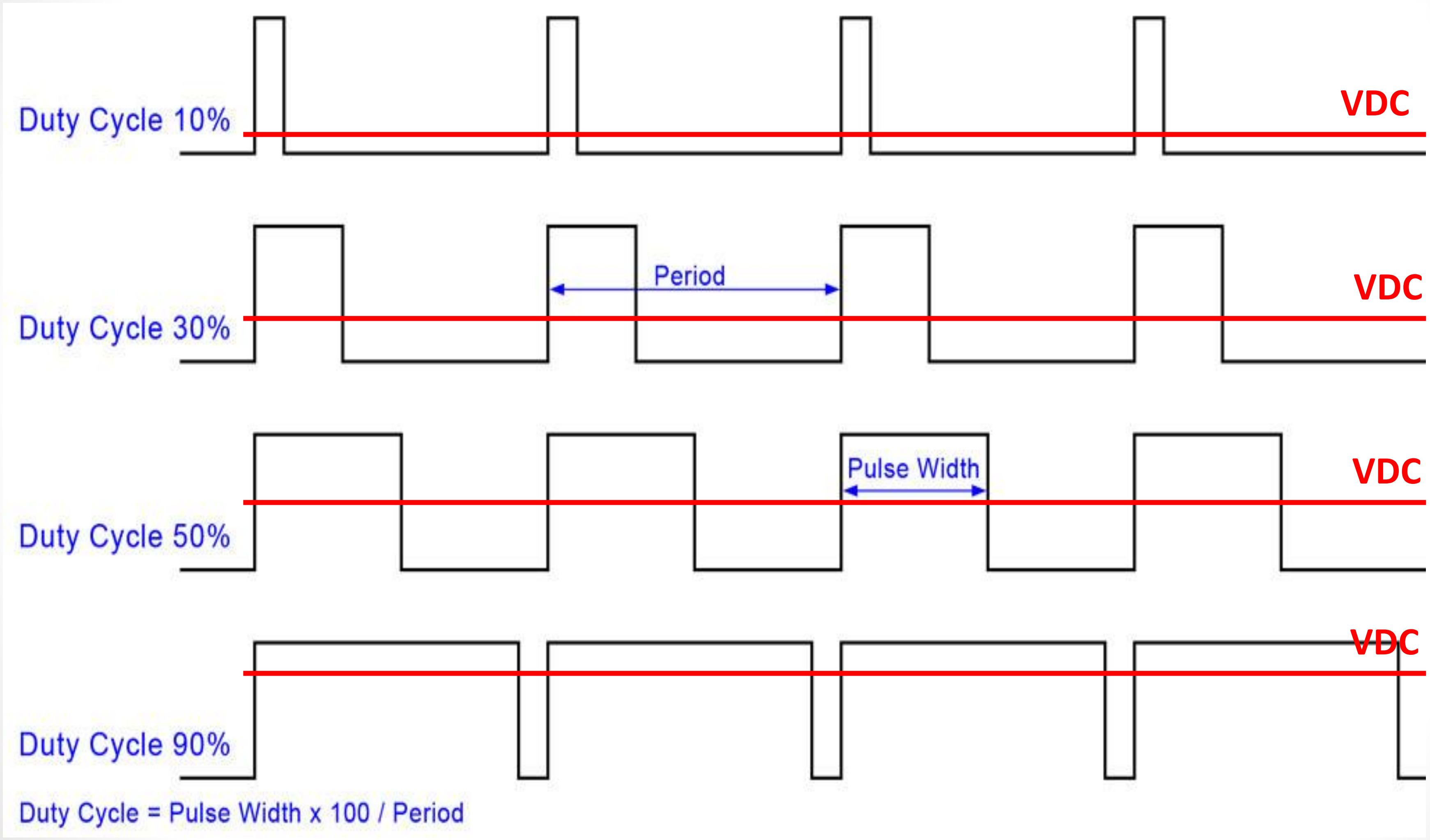
Bit	7	6	5	4	3	2	1	0	
	OCIE2	TOIE2	TICIE1	OCIE1A	OCIE1B	TOIE1	–	TOIE0	TIMSK
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

Timer Interrupt Flag Register (TIFR) :
TOV0 indicates a Timer0 overflow, cleared by hardware when the ISR is executed or by writing 1 to the flag



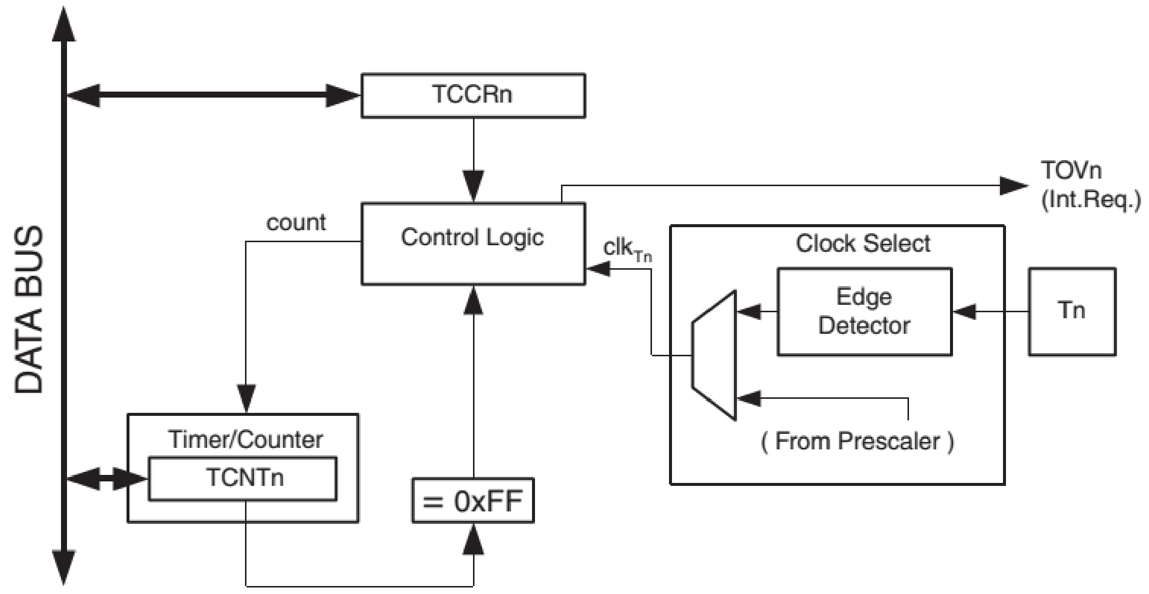
Bit	7	6	5	4	3	2	1	0	
	OCF2	TOV2	ICF1	OCF1A	OCF1B	TOV1	–	TOV0	TIFR
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

PWM


Duty Cycle 10%
VDC
Period
Duty Cycle 30%
VDC
Pulse Width
VDC
Duty Cycle 50%
VDC
Duty Cycle 90%
Duty Cycle = Pulse Width x 100 / Period

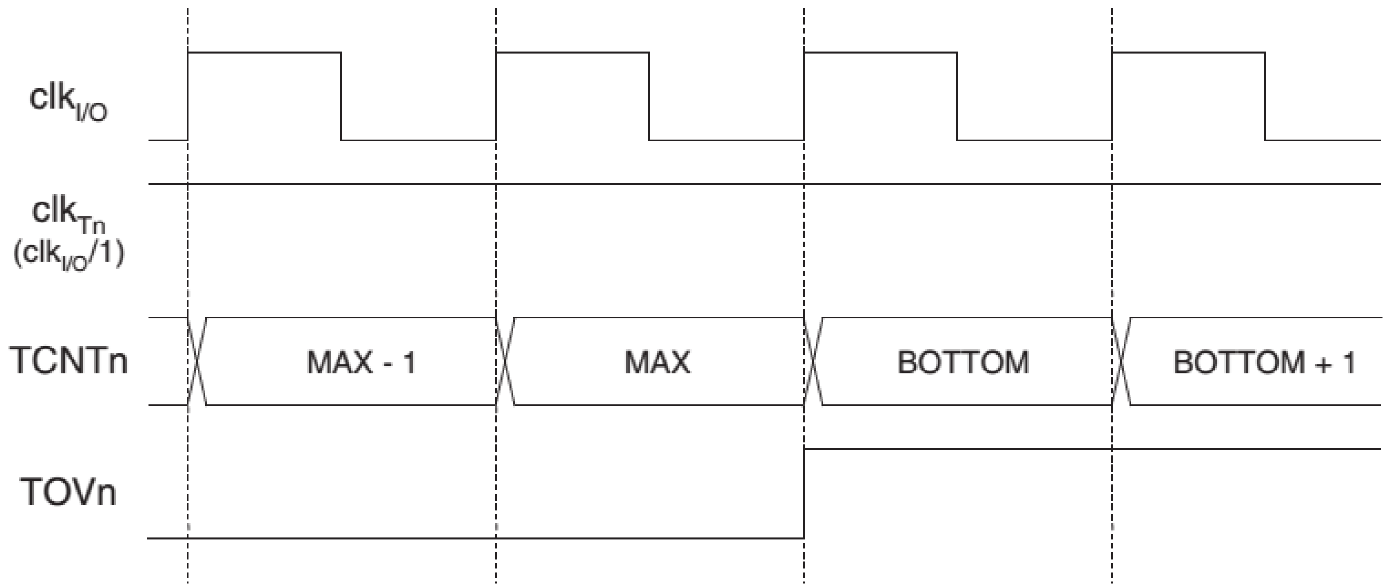
8-bit Timer/Counter 0

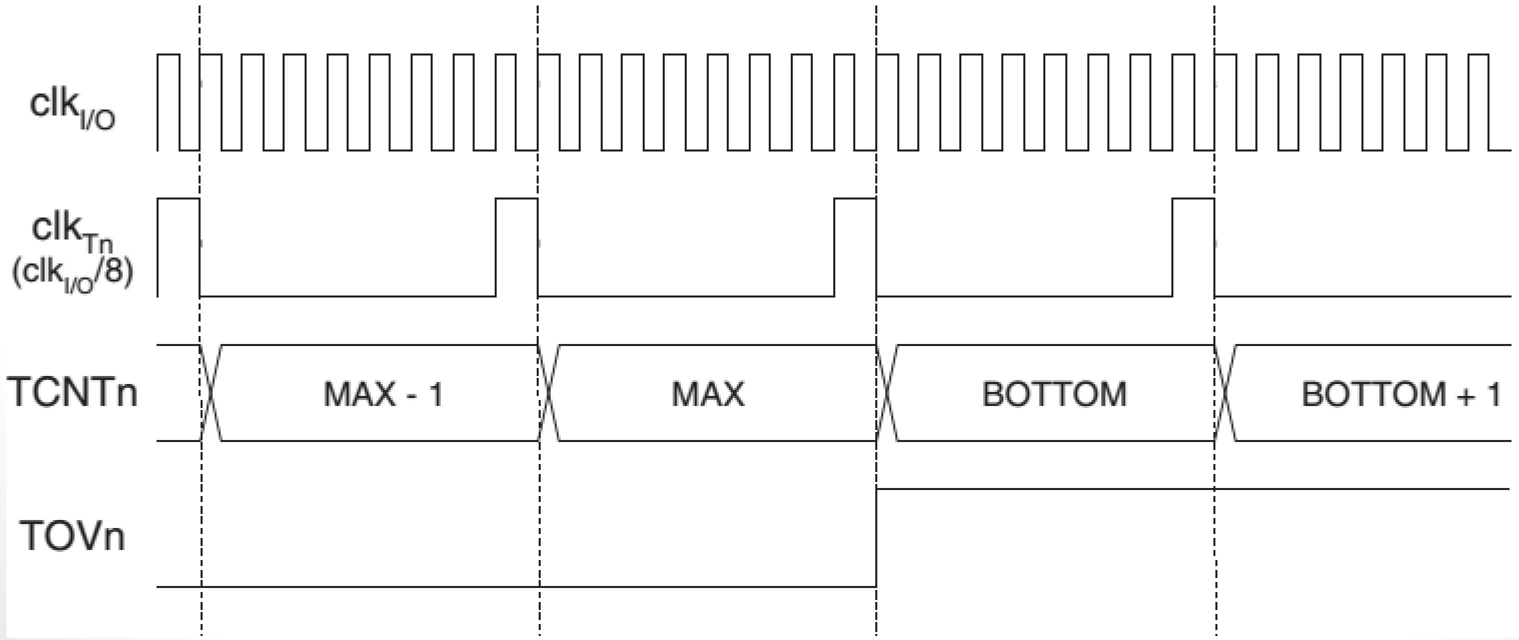
Figure 26. 8-bit Timer/Counter Block Diagram



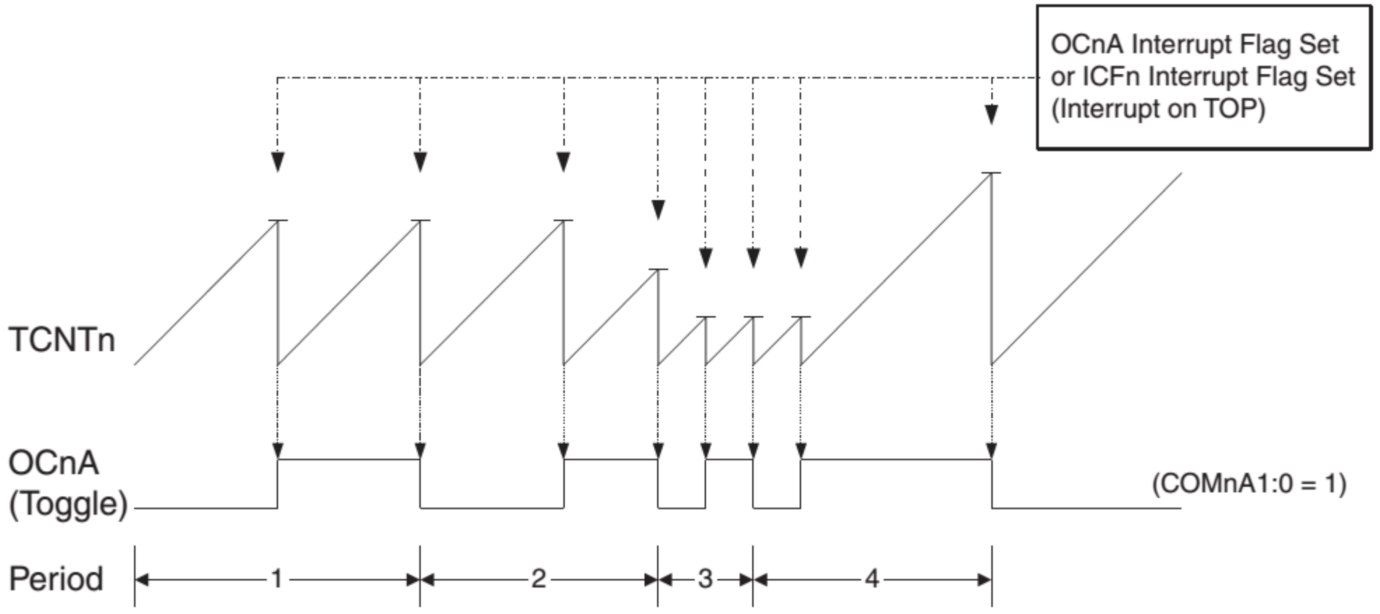
BOTTOM	The counter reaches the BOTTOM when it becomes 0x00
MAX	The counter reaches its MAXimum when it becomes 0xFF (decimal 255)

Figure 28. Timer/Counter Timing Diagram, No Prescaling





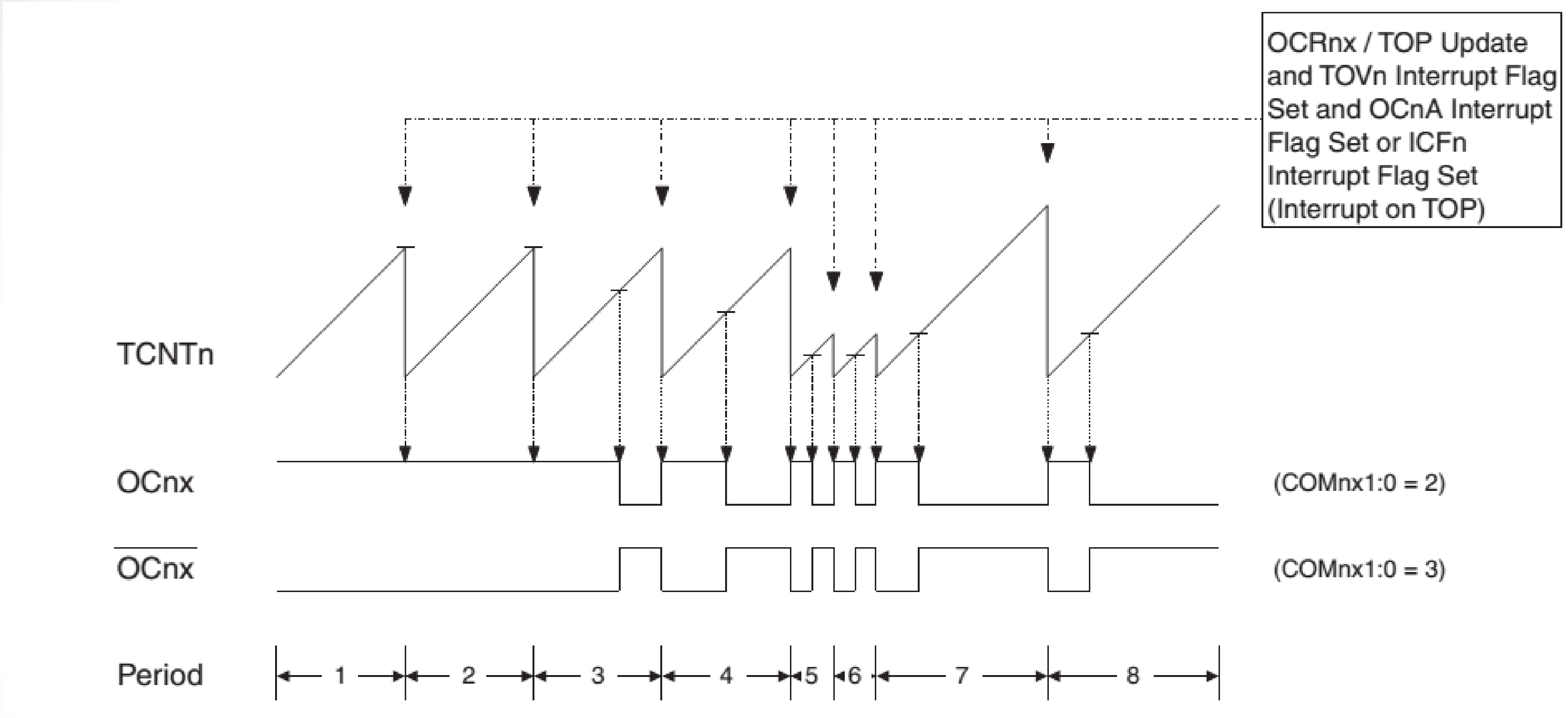
16-bit Timer/Counter 1



$$f_{OCnA} = \frac{f_{\text{clk\_I/O}}}{2 \cdot N \cdot (1 + OCRnA)}$$

CTC Mode, Timing Diagram

Fast PWM Mode, Timing Diagram


OCRnx / TOP Update and TOVn Interrupt Flag Set and OCnA Interrupt Flag Set or ICFn Interrupt Flag Set (Interrupt on TOP)
TCNTn
OCnx
OCnx
(COMnx1:0 = 2)
(COMnx1:0 = 3)
Period
1
2
3
4
5
6
7
8

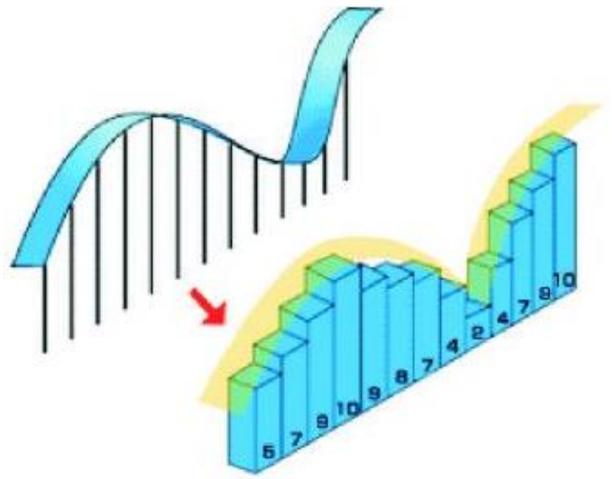
$$f_{OCnxPWM} = \frac{f_{\text{clk\_I/O}}}{N \cdot (1 + TOP)}$$

AVR Examples

AVR ADC

The Analog Digital Converter

- 10-bit Resolution (8-bit Accuracy on ADC4 and ADC5)
- Up to 15 kSPS at Maximum Resolution
- 6 Multiplexed Single Ended Input Channels
- 8 Multiplexed Single Ended Input Channels
  in TQFP / MLF Package
- 0 - VCC ADC Input Voltage Range
- Selectable internal 2.56V Reference Voltage
- Free Running or Single Conversion Mode
- Interrupt on ADC Conversion Complete
- Sleep Mode Noise Canceler



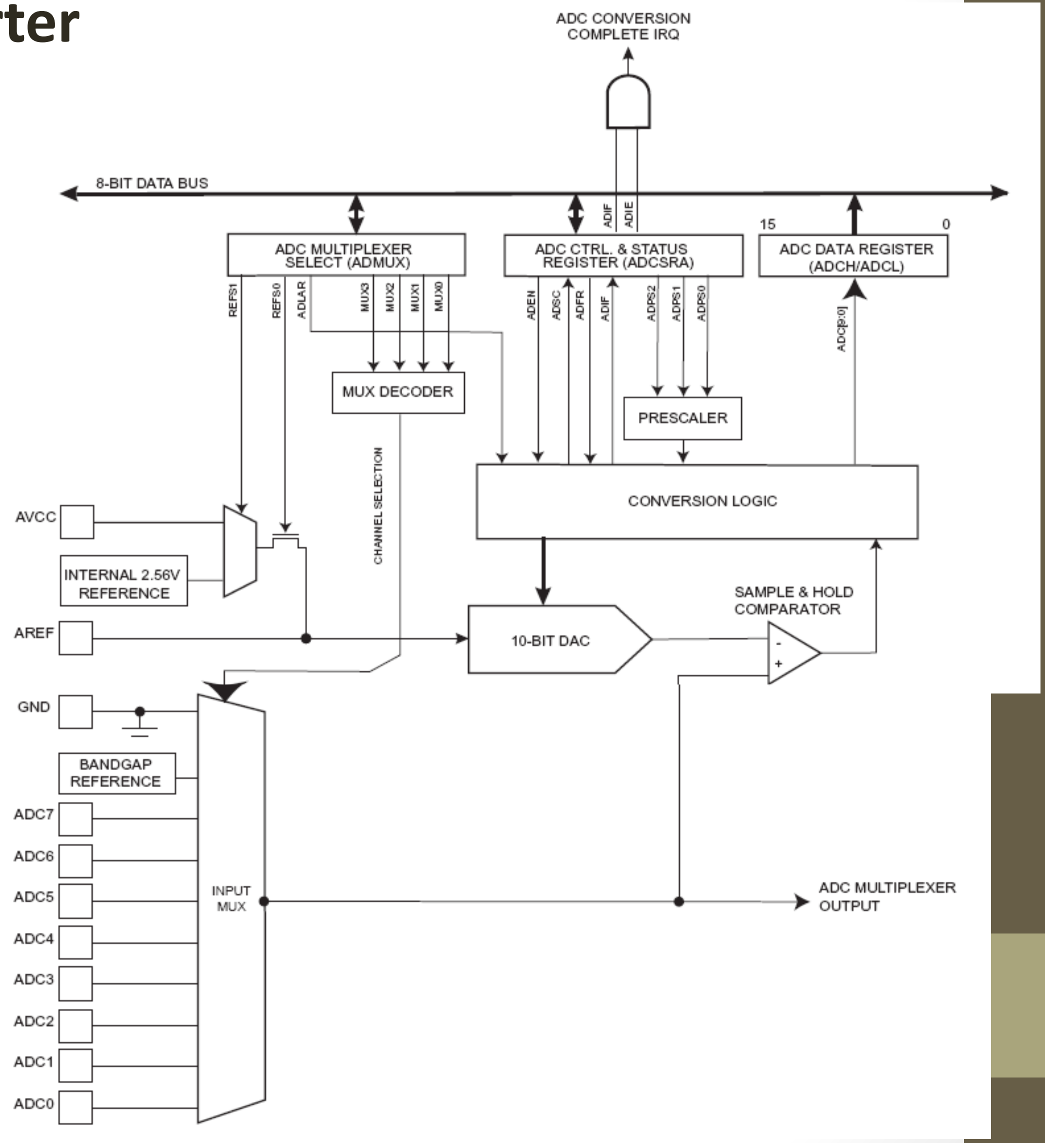
The Analog Digital Converter

● A/D conversion by successive approximation: DAC and comparator

Min: 0x0000 = GND
Max: 0x03FF = AREF-1LSB

● Internal 2,56V reference: do not connect external voltages to AREF if internal reference is used

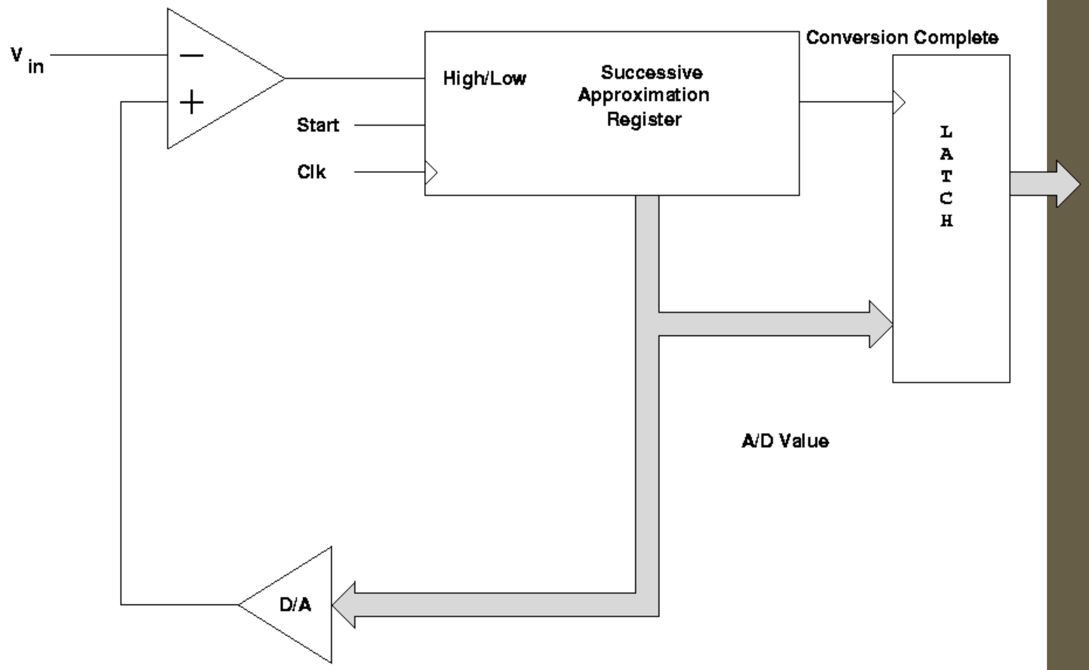
● Channel multiplexer select input channel before conversion starts

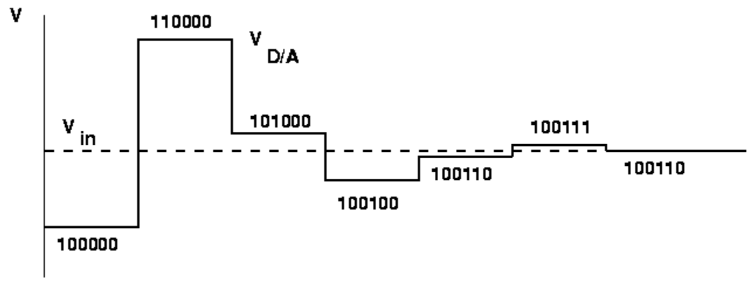


The Analog Digital Converter

Successive approximation:

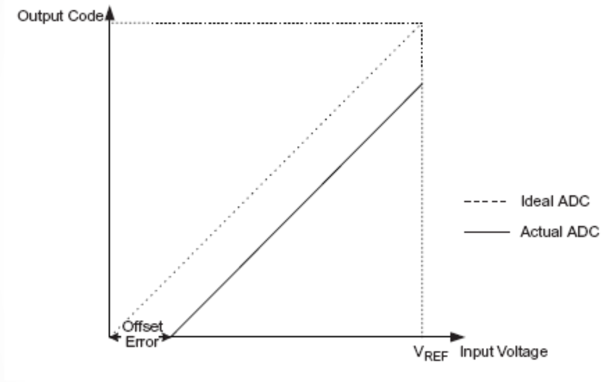
- **Starting with the MSB, test values are generated converted to analog (DAC)**
- **Result of comparation influences current bit in approximation register**
- **After n bits, the result is latched out**

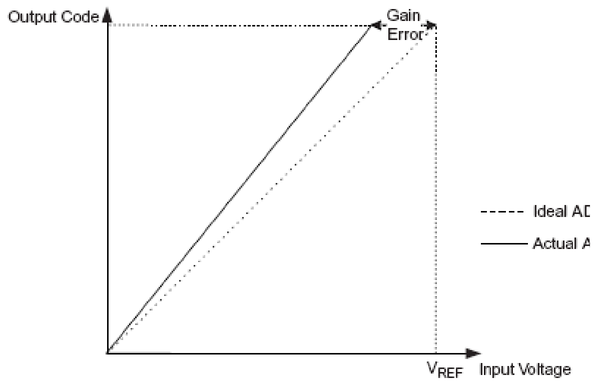


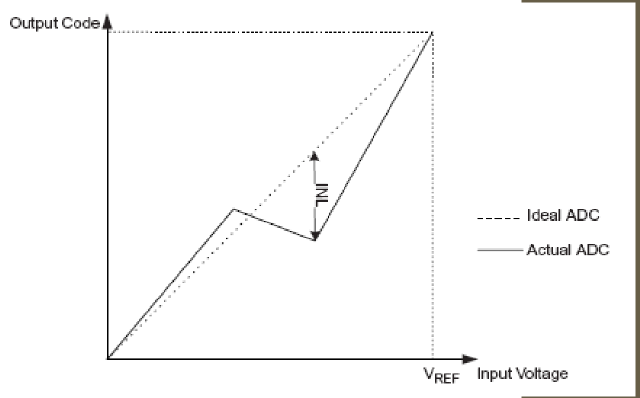


The Analog Digital Converter

Types of Conversion Errors:







Offset error

Gain error

Non-linearity

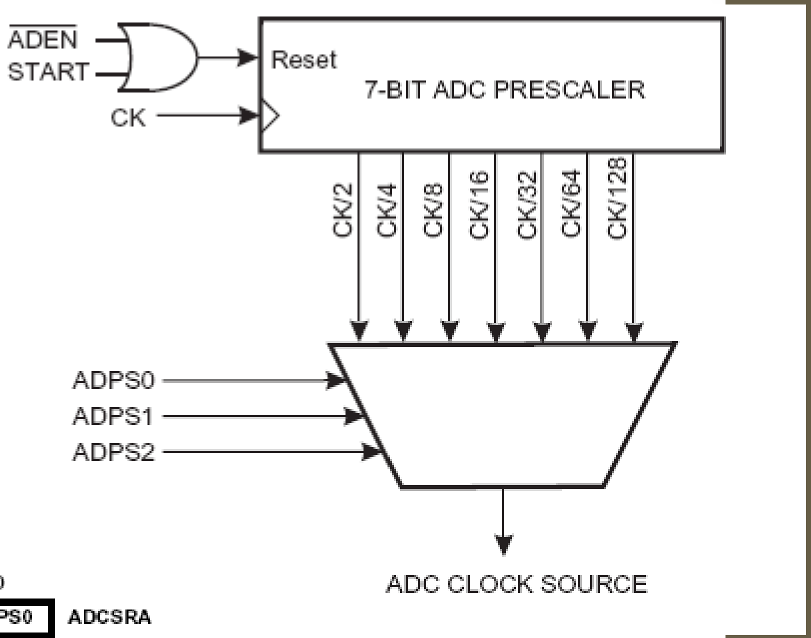
ATmega8 – ADC:

- 0.5 LSB Integral Non-linearity
- ± 2 LSB Absolute Accuracy

The Analog Digital Converter

- ADC Clock should run at 50kHz – 200kHz to give full resolution

- Prescaler generates ADC clock from CPU clocks >= 100 kHz



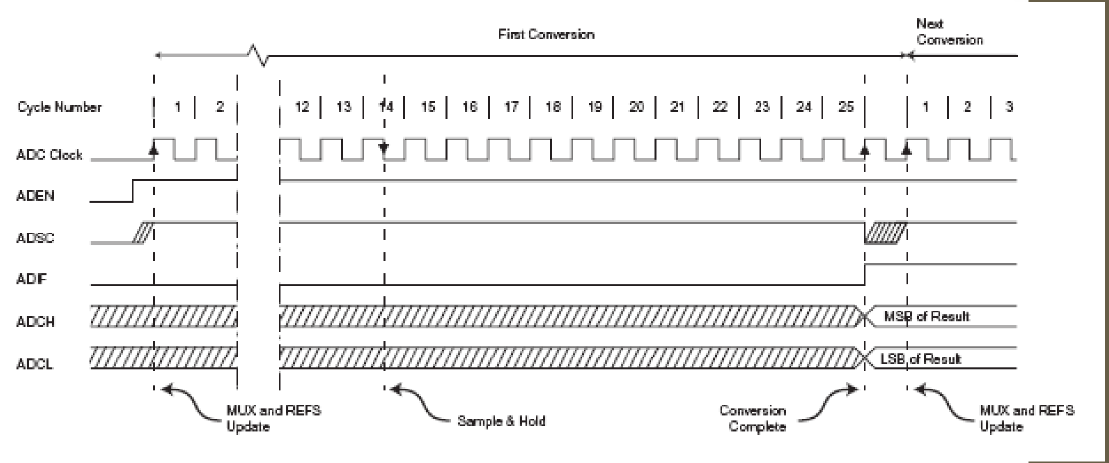
Bit	7	6	5	4	3	2	1	0	
	ADEN	ADSC	ADFR	ADIF	ADIE	ADPS2	ADPS1	ADPS0	ADCSRA
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

- ADPS2:0 bits of ADCSRA Register select clock prescaler value

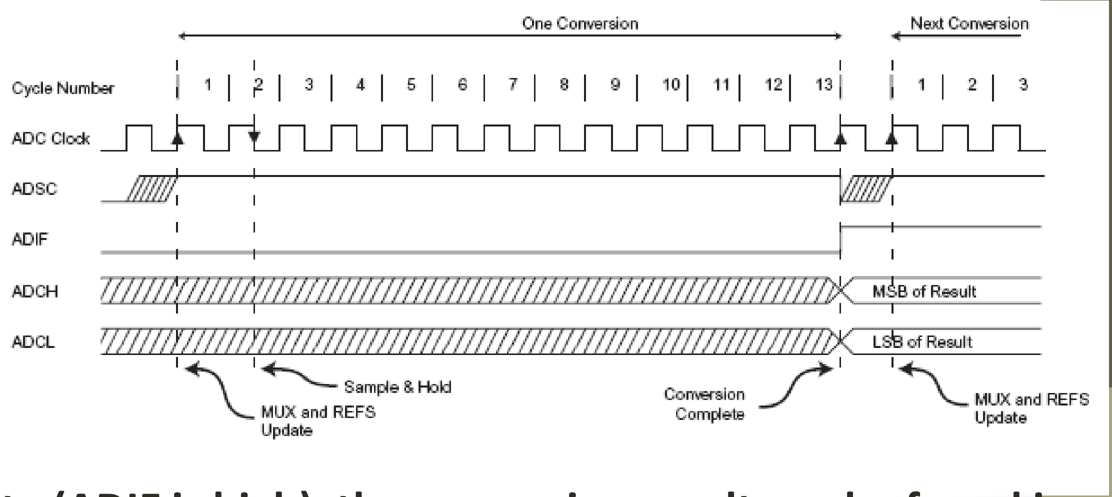
ADPS2	ADPS1	ADPS0	Division Factor
0	0	0	2
0	0	1	2
0	1	0	4
0	1	1	8
1	0	0	16
1	0	1	32
1	1	0	64
1	1	1	128

The Analog Digital Converter

Single / First conversion takes 25 ADC clock cycles

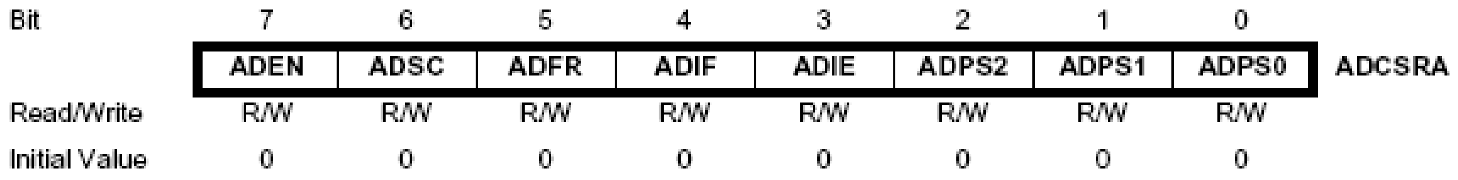


Free running conversions take 13 ADC clock cycles



After the conversion is complete (ADIF is high), the conversion result can be found in the ADC Result Registers (ADCL, ADCH)

The Analog Digital Converter



Bit	7	6	5	4	3	2	1	0	
	ADEN	ADSC	ADFR	ADIF	ADIE	ADPS2	ADPS1	ADPS0	ADCSRA
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	

ADEN: ADC Enable

ADSC: ADC Start Conversion

start a single conversion or free running mode

ADFR: ADC Free Running Select

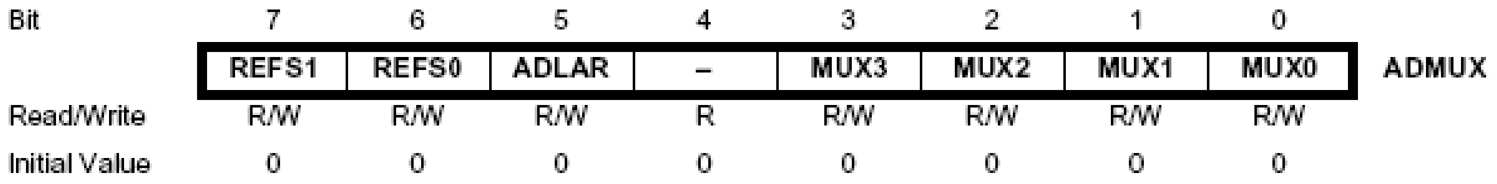
1: continuous sampling and update of data registers

ADIF: ADC Interrupt Flag

ADIE: ADC Interrupt Enable

Reset pending interrupts by writing 1 to the ADIF-Bit !

The Analog Digital Converter



Bit	7	6	5	4	3	2	1	0	
	REFS1	REFS0	ADLAR	–	MUX3	MUX2	MUX1	MUX0	ADMUX
Read/Write	R/W	R/W	R/W	R	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

REFS1:0 Reference Selection Bits
00: external AREF, 01: AVCC, 11: internal 2,56V reference

ADLAR: ADC Left Adjust Result
1: result is left adjusted, 0: result is right adjusted

MUX3:0: select the A/D input channel:
0000: ADC0 – 0111: ADC7, 1110: 1,23V, 1111: GND

The Analog Digital Converter

Handling of the ADC Data Registers (ADCL, ADCH):

ADLAR=0

Bit	15	14	13	12	11	10	9	8	
	–	–	–	–	–	–	ADC9	ADC8	ADCH
	ADC7	ADC6	ADC5	ADC4	ADC3	ADC2	ADC1	ADC0	ADCL
	7	6	5	4	3	2	1	0	

ADLAR=1

Bit	15	14	13	12	11	10	9	8	
	ADC9	ADC8	ADC7	ADC6	ADC5	ADC4	ADC3	ADC2	ADCH
	ADC1	ADC0	–	–	–	–	–	–	ADCL
	7	6	5	4	3	2	1	0	

- read Low Byte (ADCL) first, then high byte (ADCH)
- If only 8 bits resolution are needed, use left adjustment and read only high byte

The Analog Digital Converter

ADC Noise canceler:

- conversion during sleep mode to reduce noise induced
  from the CPU core and other I/O peripherals

- select single conversion mode and enable ADC interrupts
  Enter ADC Noise Reduction mode (select sleep mode „Idle“)

- ADC conversion be will start automatically after CPU
  is put to sleep using the sleep () command

- ADC interrupt (or other interrupt) will wake up the CPU

- next conversion will be issued after next sleep - command

The Analog Digital Converter

General issues for using the A/D converter

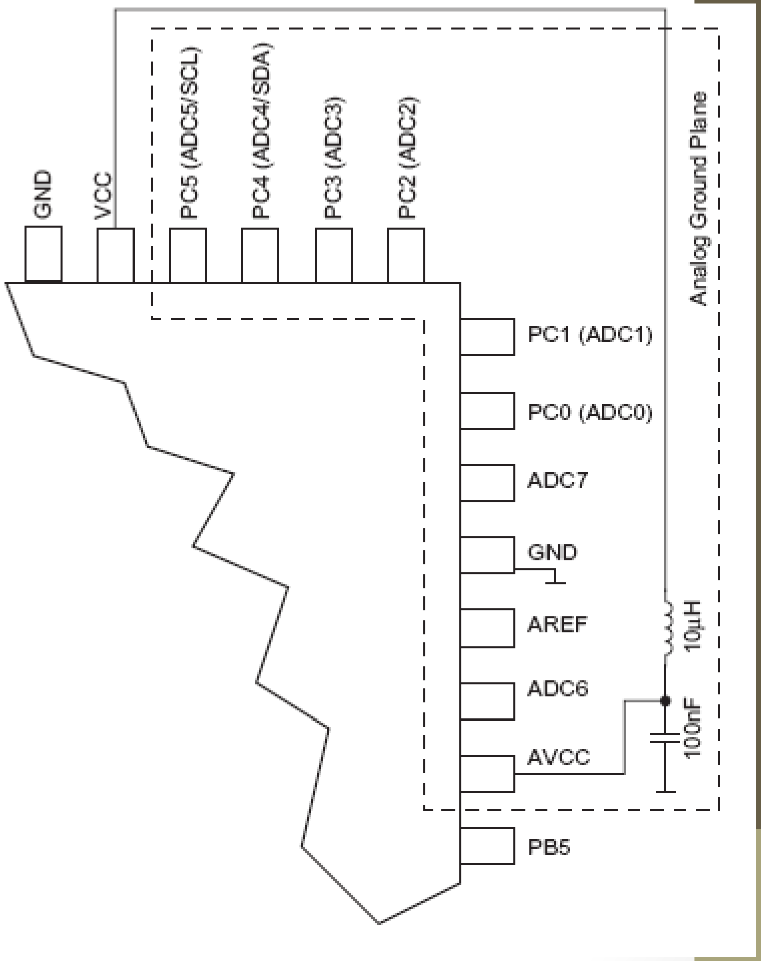
- connect low impedance sources (< 10kOhm) to achieve fast sampling rates
- do not connect signals with frequencies higher than the Nyqist frequency (half the sampling frequency)
- Use a low-pass filter to remove higher frequency components to prevent aliasing
- Keep analog signal paths as short as possible

The Analog Digital Converter

General issues for using the A/D converter

- Make sure analog tracks run over the analog ground plane, keep them well away from high-speed switching digital tracks

- Use an LC network to connect AVCC (low pass filter the supply voltage)



The Analog Digital Converter

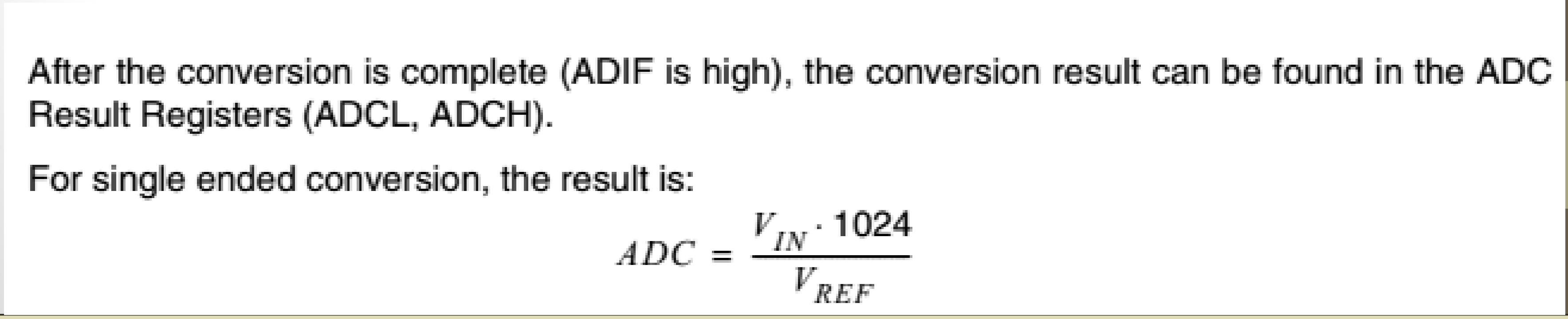
Example: Initialisation for Interrupt driven AD Conversion

```
 ADMUX = 0;              // Select ADC channel 0, external Vref
 ADCSRA =  (1<<ADPS2) | (1<<ADPS1);
// Prescaler = 64, free running mode = off, interrupts off
// prescaler = 64 (ADPS2 = 1, ADPS1 = 1, ADPS0 = 0)
// ADCYCLE = 7,3728Mhz / prescaler = 115200Hz or 8.68 us/cycle
// 14 (single conversion) cycles = 121.5 us (8230 samples/sec)
// 26 (1st conversion) cycles = 225.69 us

ADCSRA |= (1<<ADIF);     // Reset any pending ADC interrupts
ADCSRA |= (1<<ADEN);     // Enable the ADC
ADCSRA |= (1<<ADIE);     // Enable ADC interrupts
ADCSRA |= (1<<ADSC);    // single conversion: Start the ADC
```

The Analog Digital Converter

Example: ADC Interrupt Service Routine

```
ISR(ADC_vect)  // AD-conversion-complete interrupt service routine
{
    unsigned char low,high;
    unsigned int value;

    low = ADCL;                    // read ADC value (low byte first !)
    high = ADCH;                   // read ADC value high byte
    value = (high<<8) + low;  // calculate integer value
    //or use value = ADCW;
    ADCSRA |= (1<<ADSC);    // single conversion: Start the ADC
}
```



After the conversion is complete (ADIF is high), the conversion result can be found in the ADC Result Registers (ADCL, ADCH).

For single ended conversion, the result is:

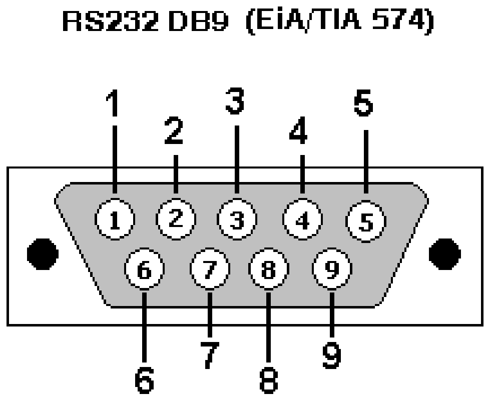
$$ADC = \frac{V_{IN} \cdot 1024}{V_{REF}}$$

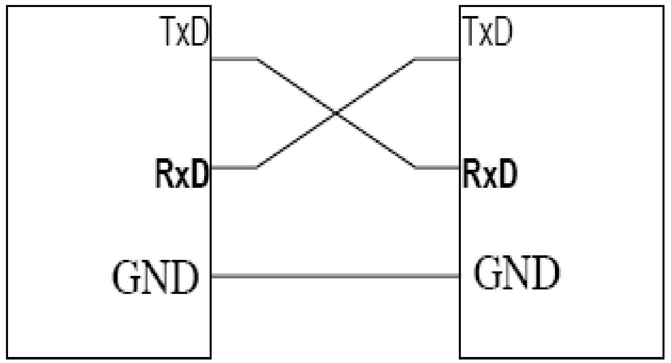
The UART Serial Interface

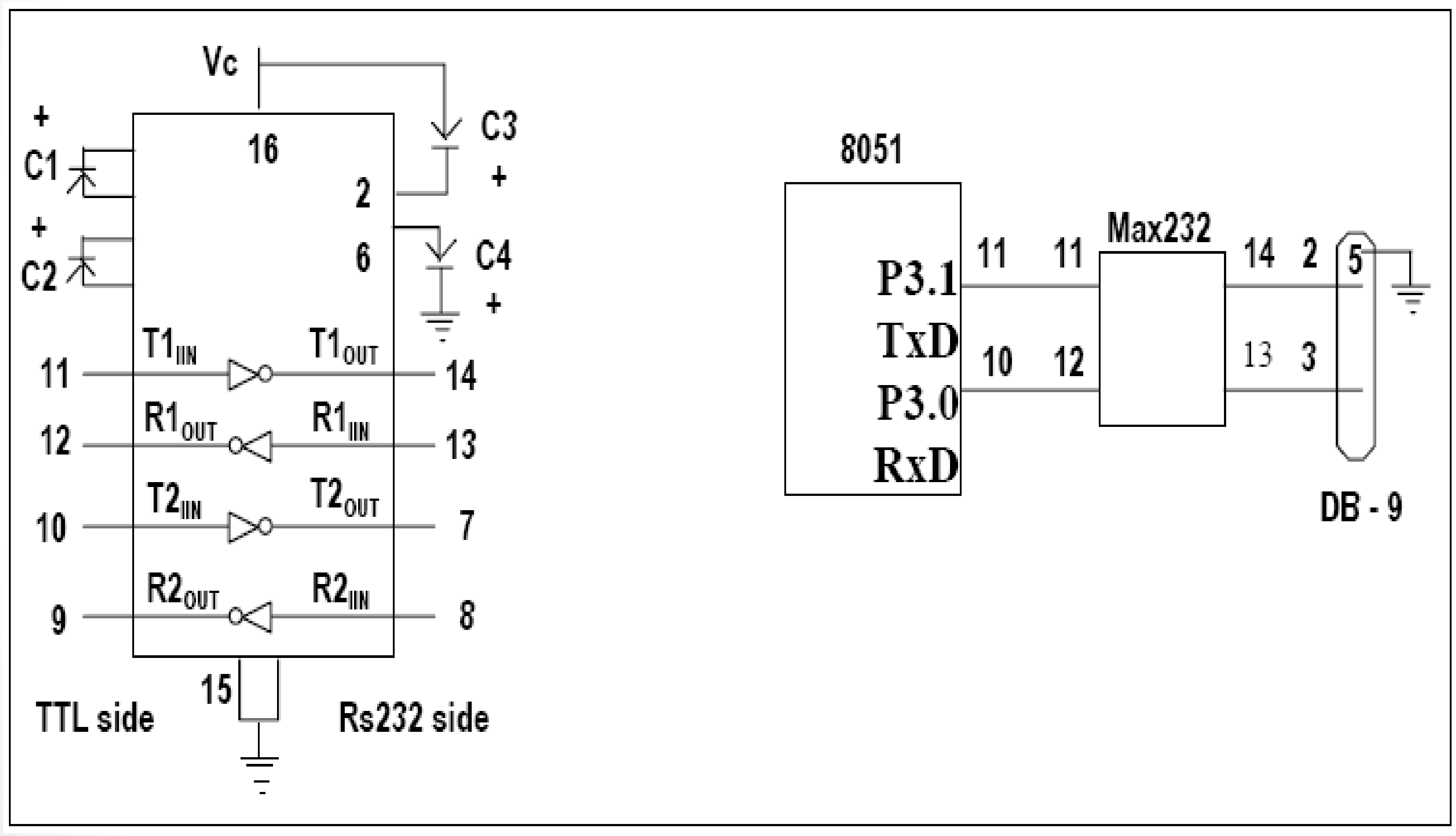
USART interface

- universal synchronous / asynchronous receiver / transmitter
- Full Duplex Operation: Receive and Transmit Registers
- Asynchronous or Synchronous Operation
- Frames with 5, 6, 7, 8, or 9 Databits and 1 or 2 Stop Bits
- Odd or Even Parity Generation and Parity Check
- Framing Error Detection, Noise Filtering
- Interrupts possible on TX Complete, TX Data Register Empty and RX Complete

RS232 Standard





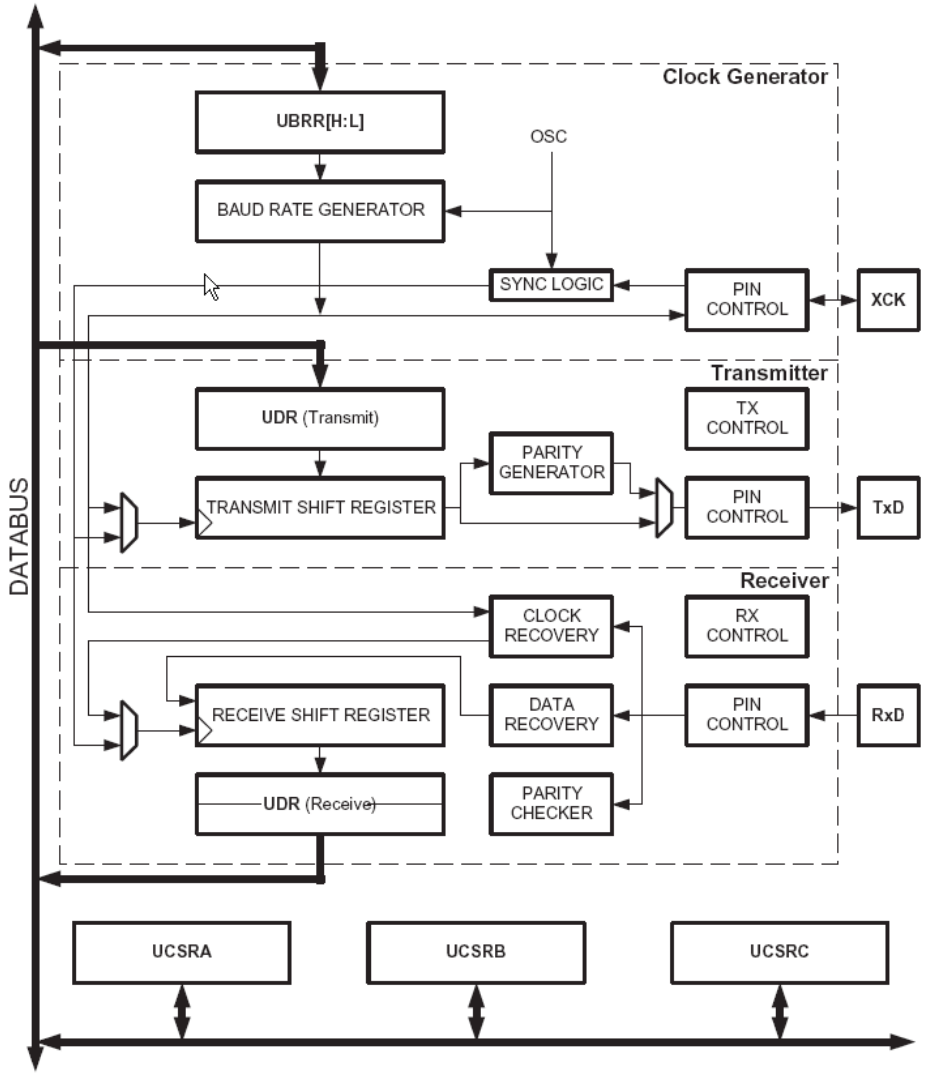


Vc
16
+
C1
+
C2
2
6
C3
+
C4
+
T1IIN
T1OUT
11
14
R1OUT
R1IIN
12
13
T2IIN
T2OUT
10
7
R2OUT
R2IIN
9
8
15
TTL side
Rs232 side
8051
P3.1
TxD
P3.0
RxD
11
11
10
12
Max232
14
2
13
3
5
DB - 9

USART interface

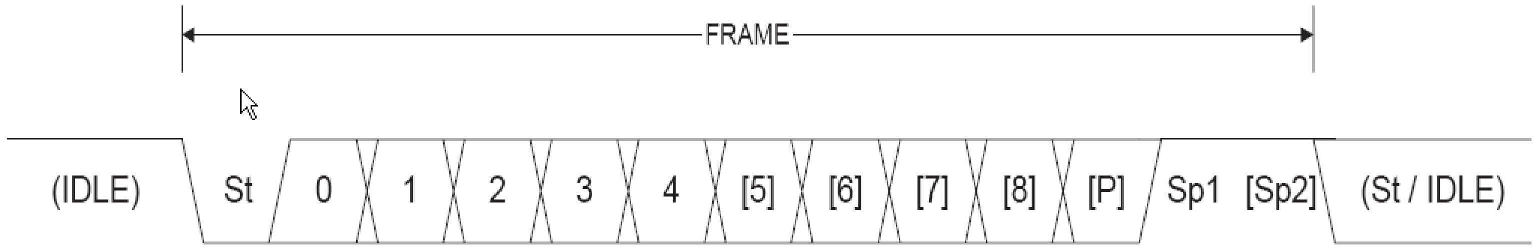
- synchronous mode:
  Pin XCK is used
  as clock Input (slave)
  or clock output (master)

- asynchronous mode:
  receiver and transmitter
  are clocked independently



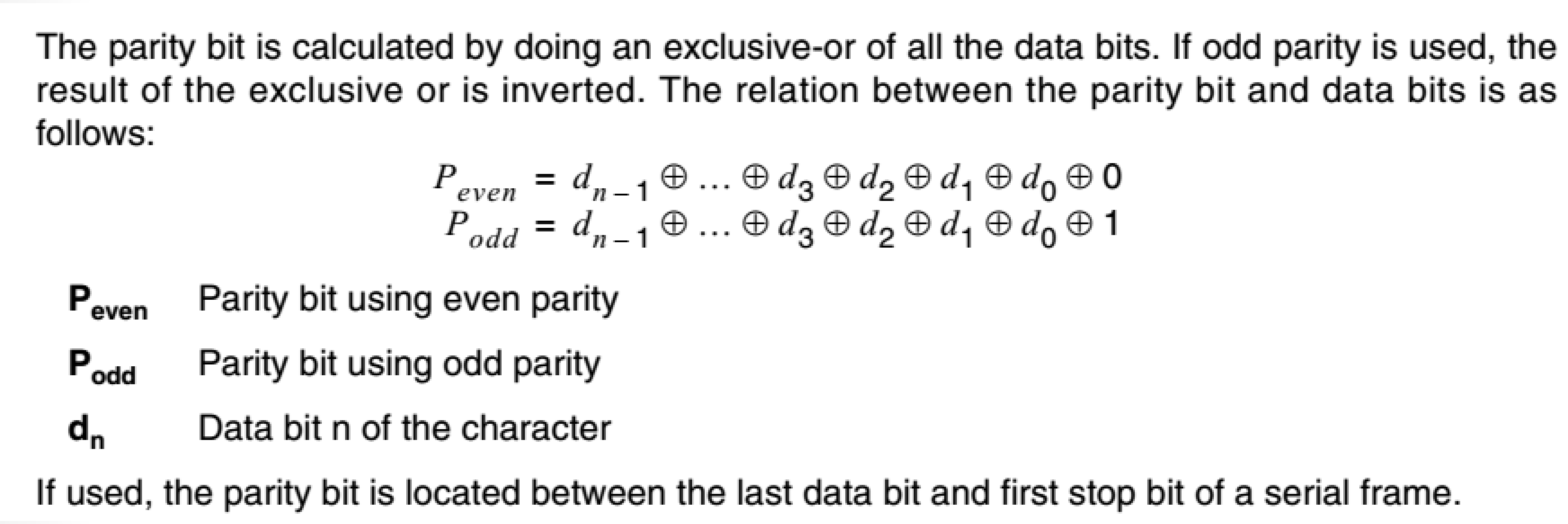
USART interface

- **Frame formats:**



St	Start bit, always low.
(n)	Data bits (0 to 8).
P	Parity bit. Can be odd or even.

Parity Bit Calculation



The parity bit is calculated by doing an exclusive-or of all the data bits. If odd parity is used, the result of the exclusive or is inverted. The relation between the parity bit and data bits is as follows:

$$P_{even} = d_{n-1} \oplus \dots \oplus d_3 \oplus d_2 \oplus d_1 \oplus d_0 \oplus 0$$
$$P_{odd} = d_{n-1} \oplus \dots \oplus d_3 \oplus d_2 \oplus d_1 \oplus d_0 \oplus 1$$

$\mathbf{P_{even}}$	Parity bit using even parity
$\mathbf{P_{odd}}$	Parity bit using odd parity
$\mathbf{d_n}$	Data bit n of the character

If used, the parity bit is located between the last data bit and first stop bit of a serial frame.

USART interface

- **calculating the baud rate register value:**

Operating Mode	Equation for Calculating Baud Rate[1]	Equation for Calculating UBRR Value
Asynchronous Normal mode (U2X = 0)	$BAUD = \frac{f_{OSC}}{16(UBRR + 1)}$	$UBRR = \frac{f_{OSC}}{16BAUD} - 1$
Asynchronous Double Speed Mode (U2X = 1)	$BAUD = \frac{f_{OSC}}{8(UBRR + 1)}$	$UBRR = \frac{f_{OSC}}{8BAUD} - 1$
Synchronous Master Mode	$BAUD = \frac{f_{OSC}}{2(UBRR + 1)}$	$UBRR = \frac{f_{OSC}}{2BAUD} - 1$

low and high byte of ubrr are written into the UBRRL and UBRRH registers

Accuracy depends on System clock source !

(see table in ATmega8 data sheet, pp 155)

USART interface

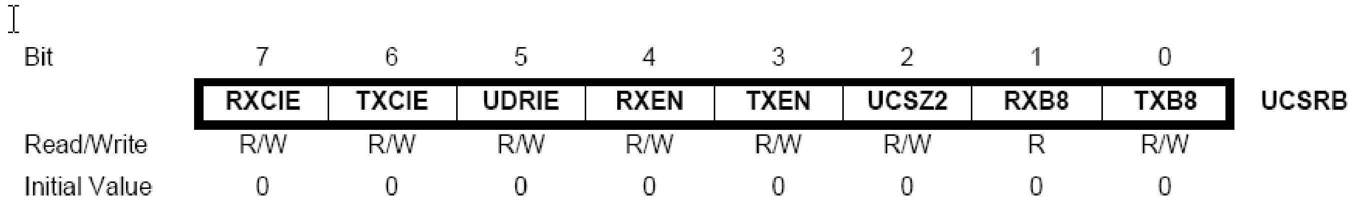
- **Initialisation: set baud rate, frame format, enable TX and RX**

Bit	7	6	5	4	3	2	1	0	
	RXC	TXC	UDRE	FE	DOR	PE	U2X	MPCM	UCSRA
Read/Write	R	R/W	R	R	R	R	R/W	R/W	
Initial Value	0	0	1	0	0	0	0	0	

RXC: RX complete
TXC: TX complete
UDR: Uart Data Register empty
FE: Frame Error
DOR: Data OverRun
PE: Parity Error
U2X: Double the USART speed

USART interface

- **Initialisation: set baud rate, frame format, enable TX and RX**

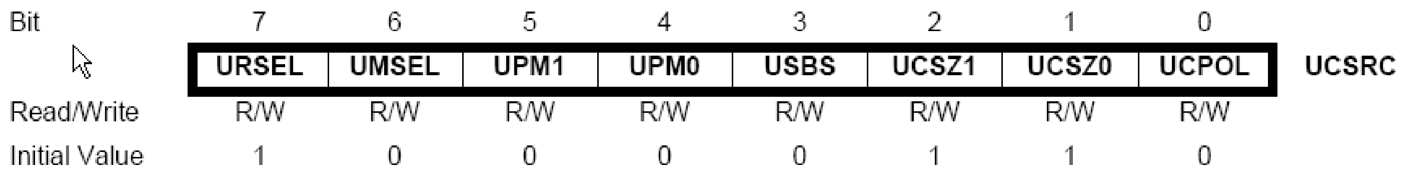


Bit	7	6	5	4	3	2	1	0	
	RXCIE	TXCIE	UDRIE	RXEN	TXEN	UCSZ2	RXB8	TXB8	UCSRB
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R	R/W	
Initial Value	0	0	0	0	0	0	0	0	

RXCIE: RX complete interrupt enable
TXCIE: TX complete interrupt enable
UDRIE: Uart Data Register empty interrupt enable
RXEN: Receiver Enable
TXEN: Transmitter Enable
UCSZ2: Character Size
RXB8, TXB8: Bit 8 for receive and transmit

USART interface

- Initialisation: set baud rate, frame format, enable TX and RX

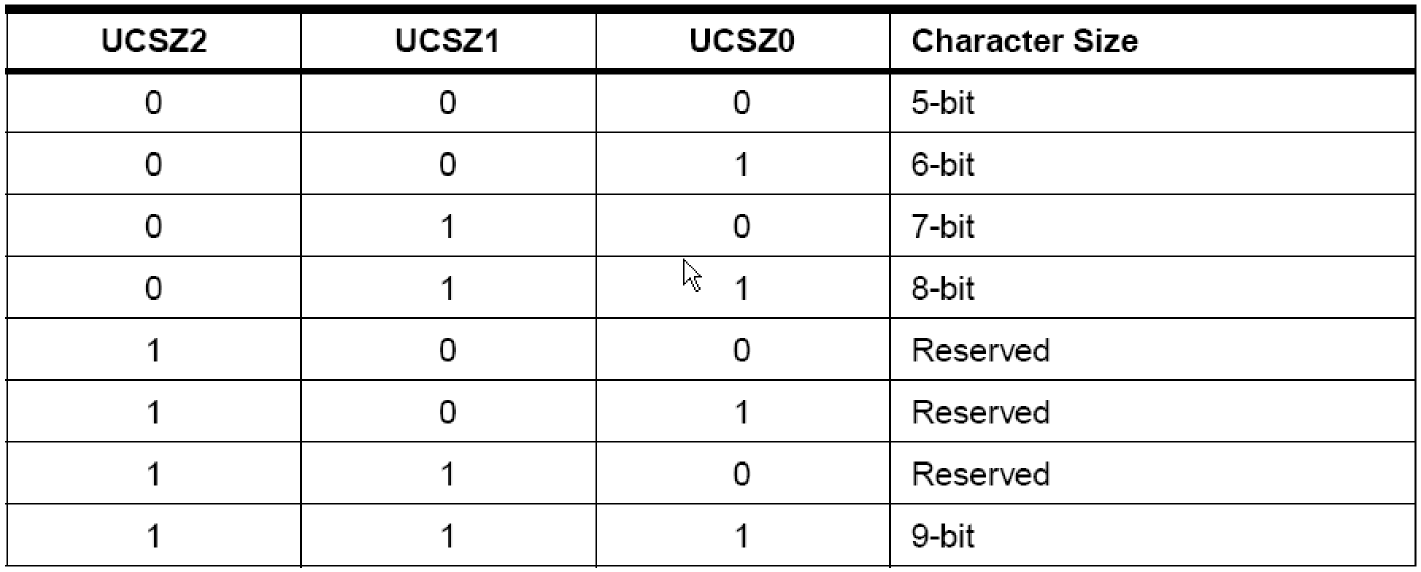


Bit	7	6	5	4	3	2	1	0	
	URSEL	UMSEL	UPM1	UPM0	USBS	UCSZ1	UCSZ0	UCPOL	UCSRC
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	1	0	0	0	0	1	1	0	

URSEL: Register Select (1=UCSR/0=UBRRH)
UMSEL: 0=async. mode, 1=sync. Mode
UMP1, UMP0: Parity mode:
00 = disabled, 10 = even, 11=odd
USBS: Stop Bits: 0=1 Stop Bit, 1= 2 Stop bits
UCSZ2,1, 0 : character size

USART interface

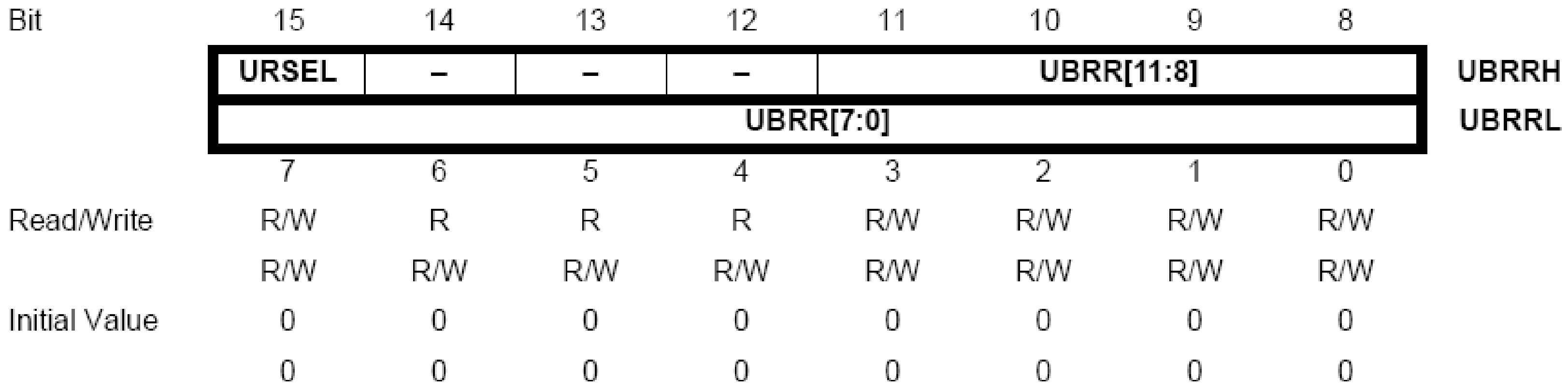
character size selection using the UCSZ bits



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

USART interface

Selection of the baud rate using the UBRRH and UBRRL SFRs:



Bit	15	14	13	12	11	10	9	8	
	URSEL	–	–	–	UBRR[11:8]				UBRRH
	UBRR[7:0]								UBRRL
	7	6	5	4	3	2	1	0	
Read/Write	R/W	R	R	R	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	

URSEL has to be 0 when writing to the UBRRH register

USART interface

- **Initialisation: set baud rate, frame format, enable TX and RX**

```
void USART_Init( unsigned int baud )
{
   /* Set baud rate */
   UBRRH = (unsigned char)(baud>>8);
   UBRRL = (unsigned char)baud;

  /* Set frame format: 8data, 2stop bit */
   UCSRC =    (1<<URSEL)|(1<<USBS)|(3<<UCSZ0);

   /* Enable Receiver and Transmitter */
   UCSRB = (1<<RXEN)|(1<<TXEN);

}
```

USART interface

- **Sending bytes in polling mode**

```
void USART_Transmit( unsigned char data )
{
   /* Wait for empty transmit buffer */
   while ( !( UCSRA & (1<<UDRE)) );

   /* Put data into buffer, sends the data */
   UDR = data;
}
```

USART interface

- **Receiving bytes in polling mode**

```
unsigned char USART_Receive( void )
{
   /* Wait for data to be received */
   while ( !(UCSRA & (1<<RXC)) ) ;

   /* Get and return received data from buffer */
   return UDR;
}
```