Topics

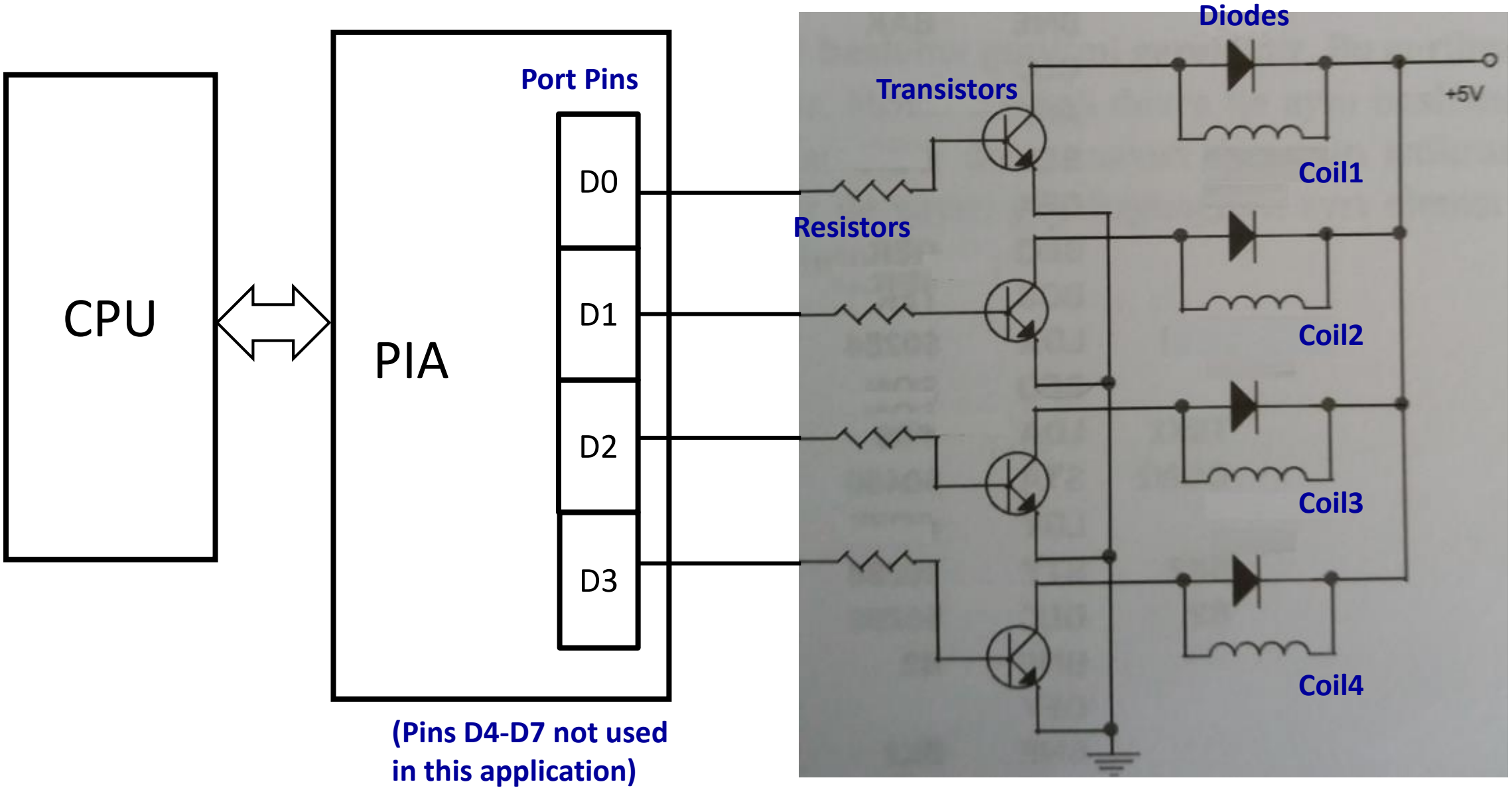
- Step Motor Example
- PIC Microcontrollers
  (LED Application with MikroC)
- CPU Examples
  (Intel, Motorola)

Step Motor Example

Step Motor

- A stepper motor is an electro-mechanical device that can be controlled by a microprocessor.
- A DC motor simply starts running continuously as soon as its operating voltage is large enough.
- But a stepper motor turns in discrete steps, each of which must be initiated by the application.
- Since each step turns the rotor by a constant degree, stepper motors are precise with repeatability of movement.
- Stepper motors are well suited for applications that require precise rotations, like printers, plotters or disk drives.

Design of Example Application



Driving a Step Motor

- In this example, the step motor is connected to the PIA's port pins, and PIA is connected to the microprocessor.

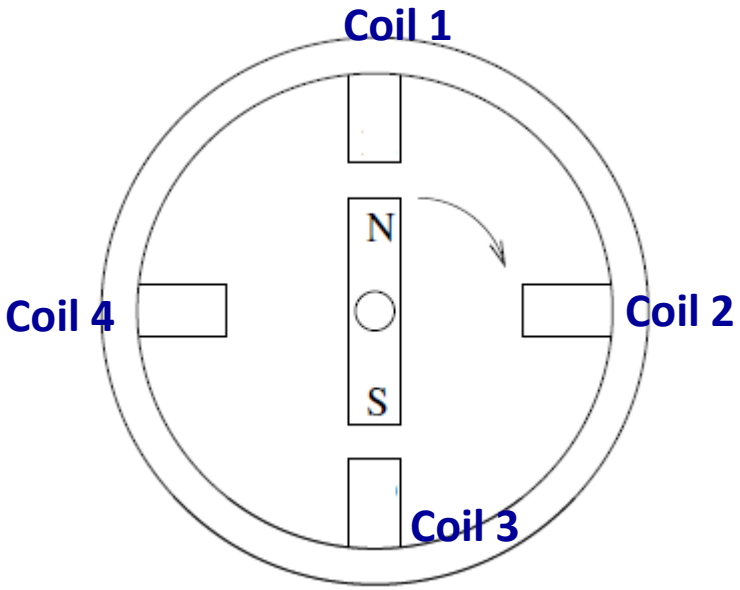
- Purpose of the application:
  - First rotate the motor right (in clockwise direction) for a specified times.
  - Then rotate the motor left (in counter-clockwise direction) for a specified times.
  - Stop the motor completely.

- The followings are the 4 set of components in the step motor device.
  - **Coils:** Logical 1 will be applied to each coil one by one, to make the motor rotate. (Coil = Inductor = Solenoid that generates magnetic field)
  - **Transistors**: They will be relays (electronic switches) to turn a coil ON/OFF.
  - **Diodes:** Helps stable starting and stopping of coils.
  - **Resistors:** Helps the electric current stable on the circuit.

Pulses to Apply for Driving the Step Motor

- The following table shows the steps and the motor coils.
- At each step, only one coil will be driven by applying logical 1 signal.
- If the signals are applied in the 1,2,3,4 step order, then motor will rotate in right direction (clockwise direction).
- If the signals are applied in the 4,3,2,1 step order, then motor will rotate in left direction (counter-clockwise direction).

Step	Coil 1	Coil 2	Coil 3	Coil 4
1	1	0	0	0
2	0	1	0	0
3	0	0	1	0
4	0	0	0	1



Motor rotates 90 degrees of angle at each step.

Main Program

```
            ORG       $1000
RIGHT_COUNTER  RMB 1
            ORG       $1000
            DAT  6    ; Example clockwise count

LEFT_COUNTER RMB 1
            ORG       $1001
            DAT  8     ; Example counter-clockwise count

START       STA  $FF, YÖNLEN.B          ;Conditioning PIA.B as transmitter
            STA  $00, İSKELE.B          ;Reset all coil signals of motor

            LDA  A, <RIGHT_COUNTER>
            CMP  A, 0                   ;Compare with 0

             BEQ  LOOP2                 ;Check if zero?
```

Main Program (continued)

```
LOOP1
      BSR  ROTATE_TO_RIGHT
      DEC  A                    ;Decrement by 1
      BNE  LOOP1                ;Continue until A becomes zero

      LDA  A, <LEFT_COUNTER>
      CMP  A, 0                 ;Compare with 0
      BEQ  FINISH               ;Check if zero?

LOOP2
      BSR  ROTATE_TO_LEFT
      DEC  A                     ;Decrement by 1
      BNE  LOOP2                ;Continue until A becomes zero

FINISH
      STA  $00, İSKELE.B        ;Reset all coil signals of motor
     INT
```

Subroutines

ROTATE_TO_RIGHT

*Rotating the motor 4 steps clockwise

```
STA  $08, İSKELE.B   ;Step1
BSR  WAIT
STA  $04, İSKELE.B   ;Step2
BSR  WAIT
STA  $02, İSKELE.B   ;Step3
BSR  WAIT
STA  $01, İSKELE.B   ;Step4
BSR  WAIT
RTS
```

ROTATE_TO_LEFT

*Rotating the motor 4 steps counter-clockwise

```
STA  $01, İSKELE.B   ;Step1
BSR  WAIT
STA  $02, İSKELE.B   ;Step2
BSR  WAIT
STA  $04, İSKELE.B   ;Step3
BSR  WAIT
STA  $08, İSKELE.B   ;Step4
BSR  WAIT
RTS
```

```
* WAIT Routine
WAIT
    LDA  SK,30000
LOOP   DEC SK
    BNE  LOOP
    RTS
```

Alternative Subroutines (using loops)

```
ROTATE_TO_RIGHT
            LDA  C, 0           ;C will count from 1 to 4
            STA  B, $08         ;B will contain the step motor signals

LOOPR       STA  B, İSKELE.B    ;Rotate the motor one step
            BSR  WAIT
             INC  C             ;Increment by 1
            CMP  C, 4           ;Compare with 4
            BEQ  RETURNR
            LSR  B              ; Logical Shift Right
            BRA  LOOPR

RETURNR   RTS
```

```
ROTATE_TO_LEFT
            LDA  C, 0
            STA  B, $01

LOOPL       STA  B, İSKELE.B
            BSR  WAIT
             INC  C
            CMP  C, 4
            BEQ  RETURNL
            LSL  B      ; Logical Shift Left
            BRA  LOOPL

RETURNL   RTS
```

PIC Microcontrollers

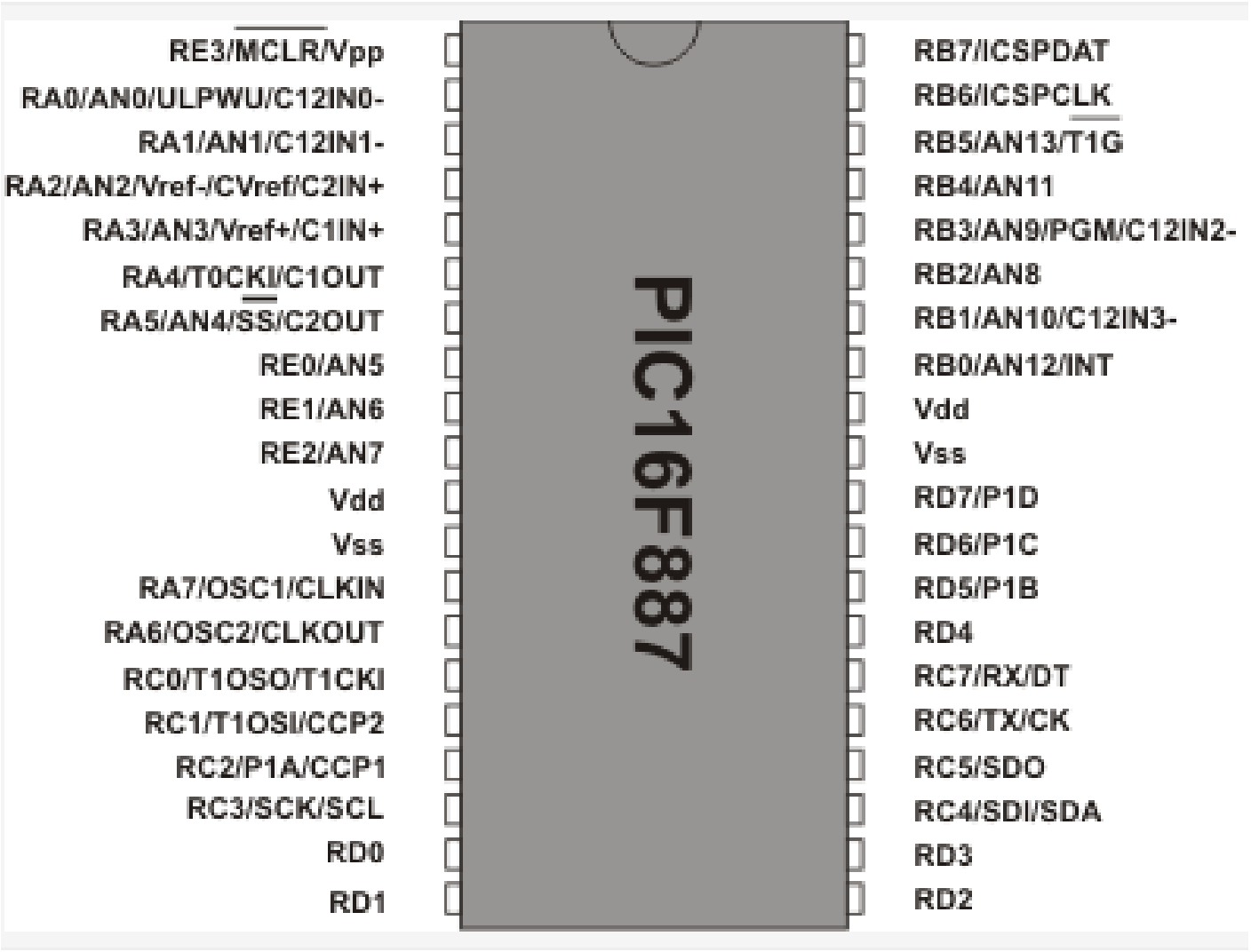
Peripheral Interface Controller

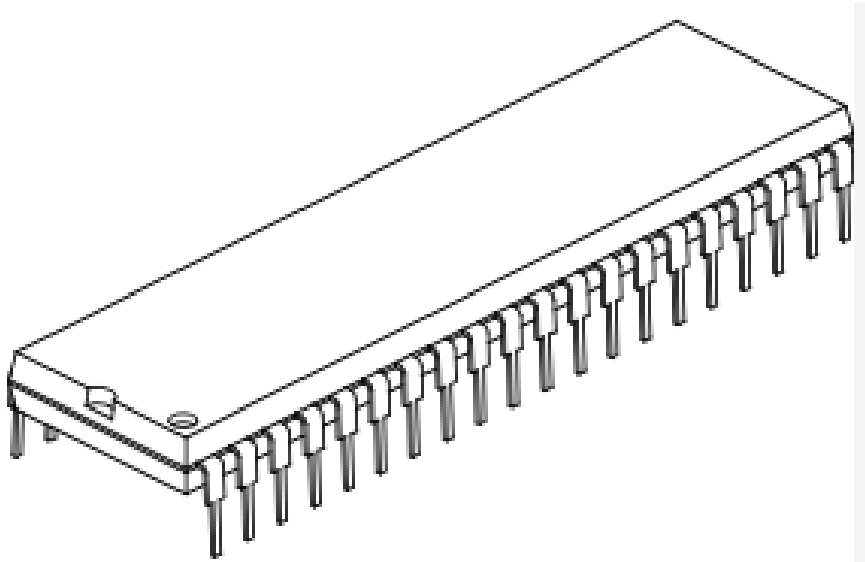
- PIC microcontrollers were first introduced by the Microchip Technology company.
- PIC microcontrollers use Harvard architecture.
- Program memory is connected to the CPU over more than 8 lines.
- Depending on the microcontroller, address bus width can be 12, 14, and 16 bit.
- If 4 MHz oscillator is used, average time for instruction execution is 1 µS.

Basic Features of the PIC Microcontroller (16F887)

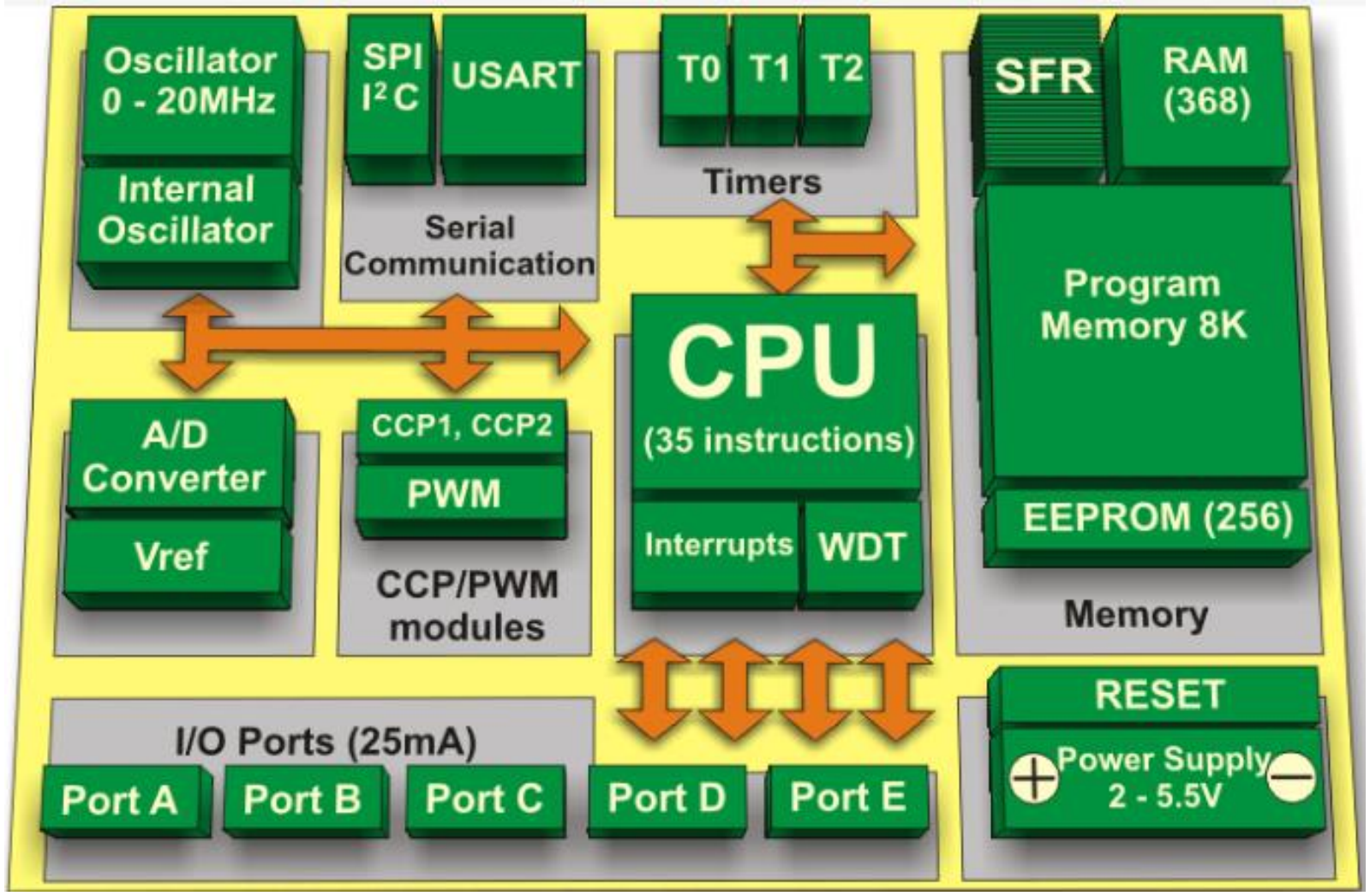
- **RISC architecture** (Only 35 instructions)
- **Operating frequency 0 - 20 MHz**
- **Power supply voltage 2.0 - 5.5 V**
- **35 input/output pins** (Direct LED drive, Interrupt pin)
- **8K ROM memory in FLASH**
- **256 bytes EEPROM memory**
- **368 bytes RAM memory**
- **A/D converter** (10-bit resolution)
- **USART** (Universal Serial Asynchronous Receiver Transmitter)

PIC Microcontroller Chip

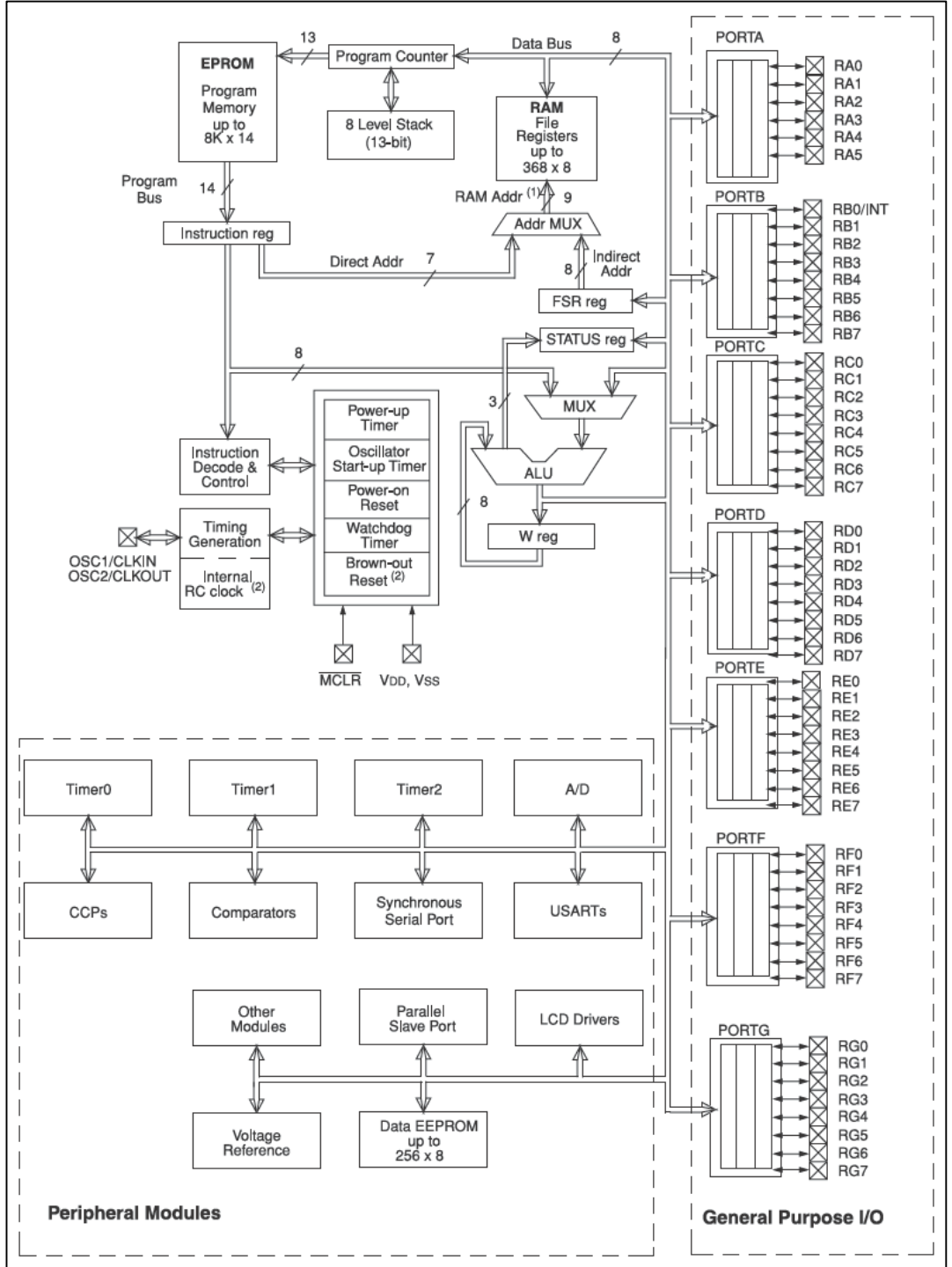




PIC Microcontroller Chip



Architecture of a 8-bit PIC microcontroller



Special Function Registers in PIC

- These registers are also RAM memory locations, but unlike general-purpose registers, their purpose is predetermined during manufacturing process.
- Since their bits are connected to particular circuits on the chip (A/D converter, serial communication module, etc.), any change of their contents directly affects the operation of the microcontroller.
- The memory locations have their names (both registers and their bits), which simplifies the process of writing a program.
- It is enough to specify the name of a register in order to read or change its contents.

Addr.	Name
00h	INDF
01h	TMR0
02h	PCL
03h	STATUS
04h	FSR
05h	PORTA
06h	PORTB
07h	PORTC
08h	PORTD
09h	PORTE
0Ah	PCLATH
0Bh	INTCON
0Ch	PIR1
0Dh	PIR2
0Eh	TMR1L
0Fh	TMR1H
10h	T1CON
	....

Assembly Instruction Set for PIC

Data Transfer Instructions

INSTRUCTION	DESCRIPTION
MOVLW k	Move constant to W
MOVWF f	Move W to f
MOVF f,d	Move f to d
CLRW	Clear W
CLRF f	Clear f
SWAPF f,d	Swap four bits in f

Program Control Instructions

BTFSC f,b	Test bit b of f. Skip the following instruction if clear.
BTFSS f,b	Test bit b of f. Skip the following instruction if set.
DECFSZ f,d	Decrement f. Skip the following instruction if clear.
INCFSZ f,d	Increment f. Skip the following instruction if set.
GOTO k	Go to address
CALL k	Call subroutine
RETURN	Return from subroutine
RETLW k	Return with constant in W

Bit-oriented Instructions

BCF f,b	Clear bit b in f
BSF f,b	Clear bit b in f

Arithmetic / Logic Instructions

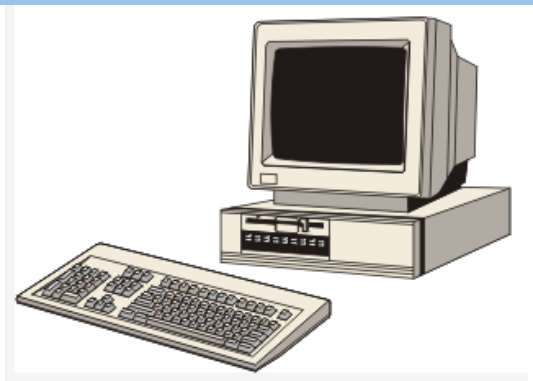
ADDLW k	Add W and constant
ADDWF f,d	Add W and f
SUBLW k	Subtract W from constant
SUBWF f,d	Subtract W from f
ANDLW k	Logical AND with W with constant
ANDWF f,d	Logical AND with W with f
ANDWF f,d	Logical AND with W with f
IORLW k	Logical OR with W with constant

IORWF f,d	Logical OR with W with f
XORWF f,d	Logical exclusive OR with W with constant
XORLW k	Logical exclusive OR with W with f
INCF f,d	Increment f by 1
DECF f,d	Decrement f by 1
RLF f,d	Rotate left f through CARRY bit
RRF f,d	Rotate right f through CARRY bit
COMF f,d	Complement f

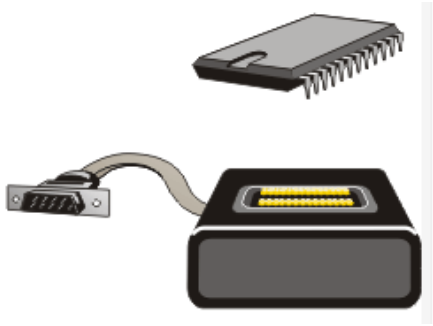
Using MikroC for PCI Programming

- In adddition to assembly language, it is possible to use the C language for PCI programming.
- The **MikroC** is a special C compiler with its own IDE (Interactive Development Environment).

1) Write and compile the program (Assembly or C).



2) Copy binary program to ROM memory



IDE Components of MikroC

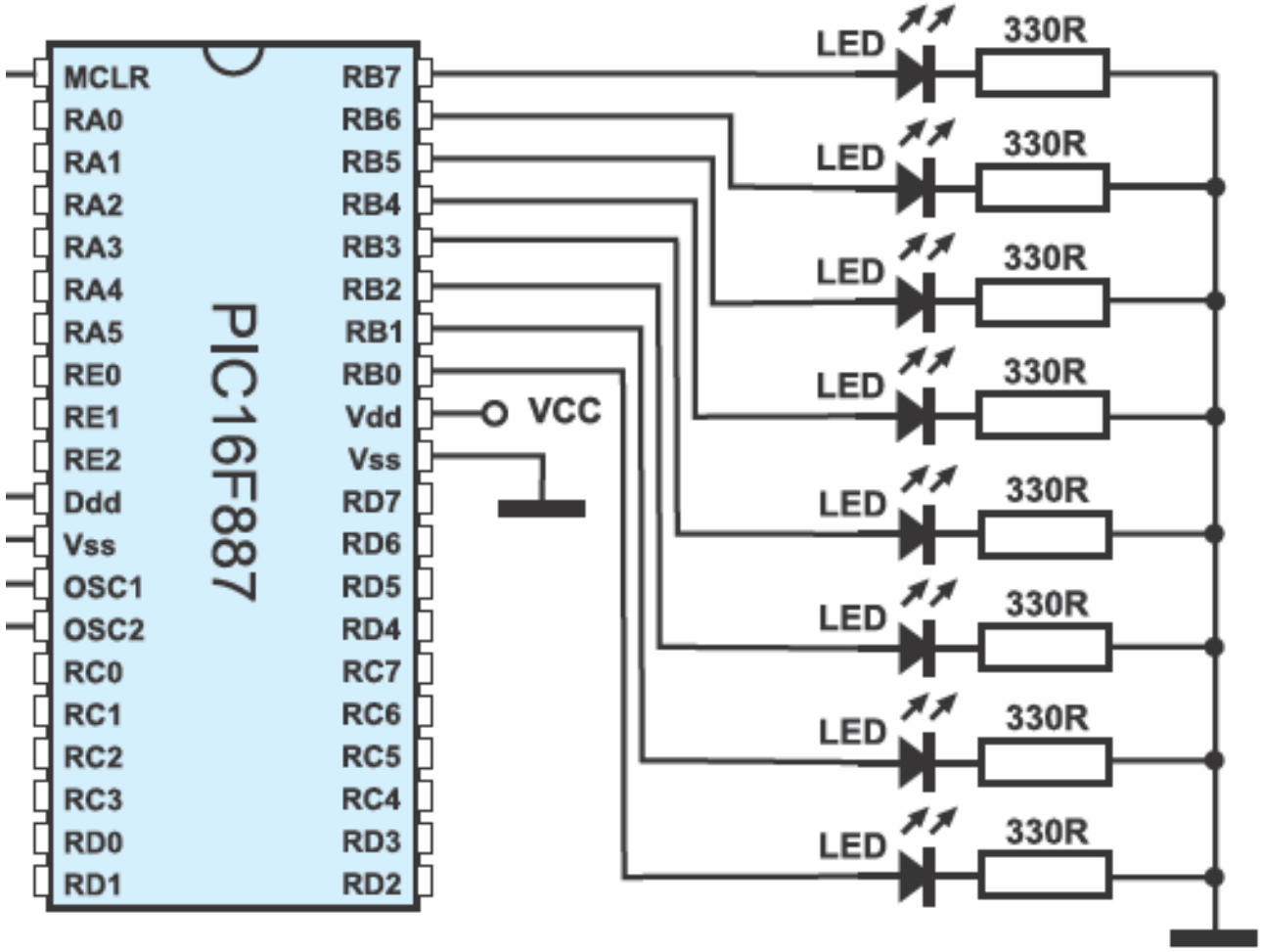
- Code Editor
- Software Simulator
- USART Terminal
- EEPROM Editor
- Seven Segment LED Editor
- LCD Editor

Some Hardware Libraries of MikroC

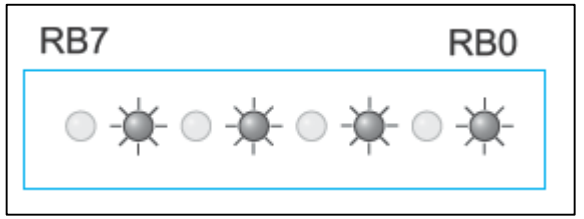
- Keypad (4×4 push buttons)
- PS/2 (Standard keyboard)
- LCD display (2×16 characters)
- Graphic LCD (128×64 resolution)
- Serial communication
- UART (Universal Asynchronous Receiver Transmitter)
- USB (Universal Serial Bus)
- A/D converter
- EEPROM
- Flash Memory
- Ethernet
- PWM (Pulse Wave Modulation)
- Sound (audio signal generation)
- Multi Media Card

Example: LED Application with PIC and MikroC

- The purpose of this application is to turn on a few LED diodes on port B.
- Figure below shows PIC pin connections to LEDs.



Example LED lights



Description of Application

- **PHASE-1:**
  - As soon as the microcontroller is turned on, all LEDs will emit light for a second. The Delay function will be called to wait in milliseconds.
  - After one second, the program enters the for loop, and iterates until k counter reaches 20.
    The variable is incremented by 1 after each iteration.
    Within the for loop, the if command monitors port B logic state.
    If PORTB=0xFF, then its state is inverted into 0x00, and vice versa.

- **PHASE-2:**
  - When the program exits the for loop, the port B logic state changes (0xb 01010101) and the program enters the endless while loop.
    The port B logic state is inverted each 200 mS.

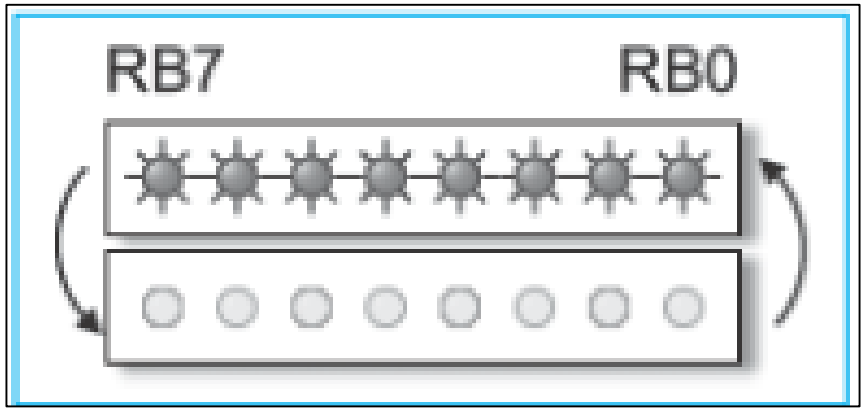
MikroC Program

```
void main()
{
  int k;

  ANSEL = 0;              // All pins of I/O port are configured as digital
  PORTB = 0xFF;           // Reset port B (all LEDs light OFF)
  TRISB = 0;              // Port B pins are configured as outputs

  PORTB = 0x00;           // All LEDs light ON

  Delay_ms(1000);         // 1 second delay
```

MikroC Program (continued)

```
// PHASE-1
 for (k=1; k <20; k++)
  {
    if (PORTB == 0x00)
    {
      PORTB = 0xFF;    // All LEDs light OFF
      Delay_ms(100);  // 100 mili seconds delay
    }
    else
    {
      PORTB = 0x00;   // All LEDs light ON
      Delay_ms(100);  // 100 mili seconds delay
    }

  } // end of for
```

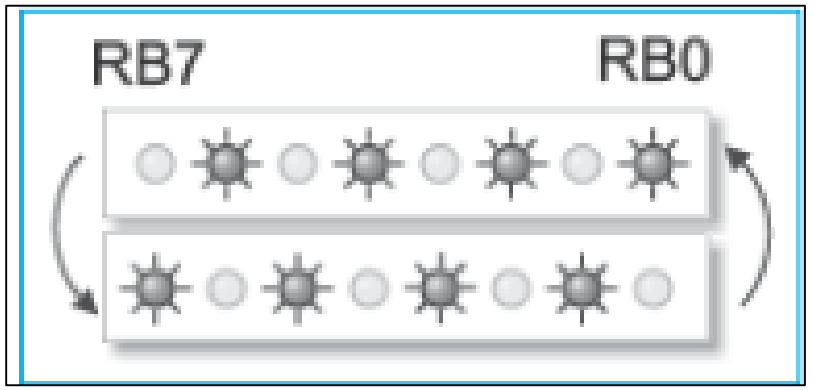
All LEDs ON , then all LEDs OFF



MikroC Program (continued)

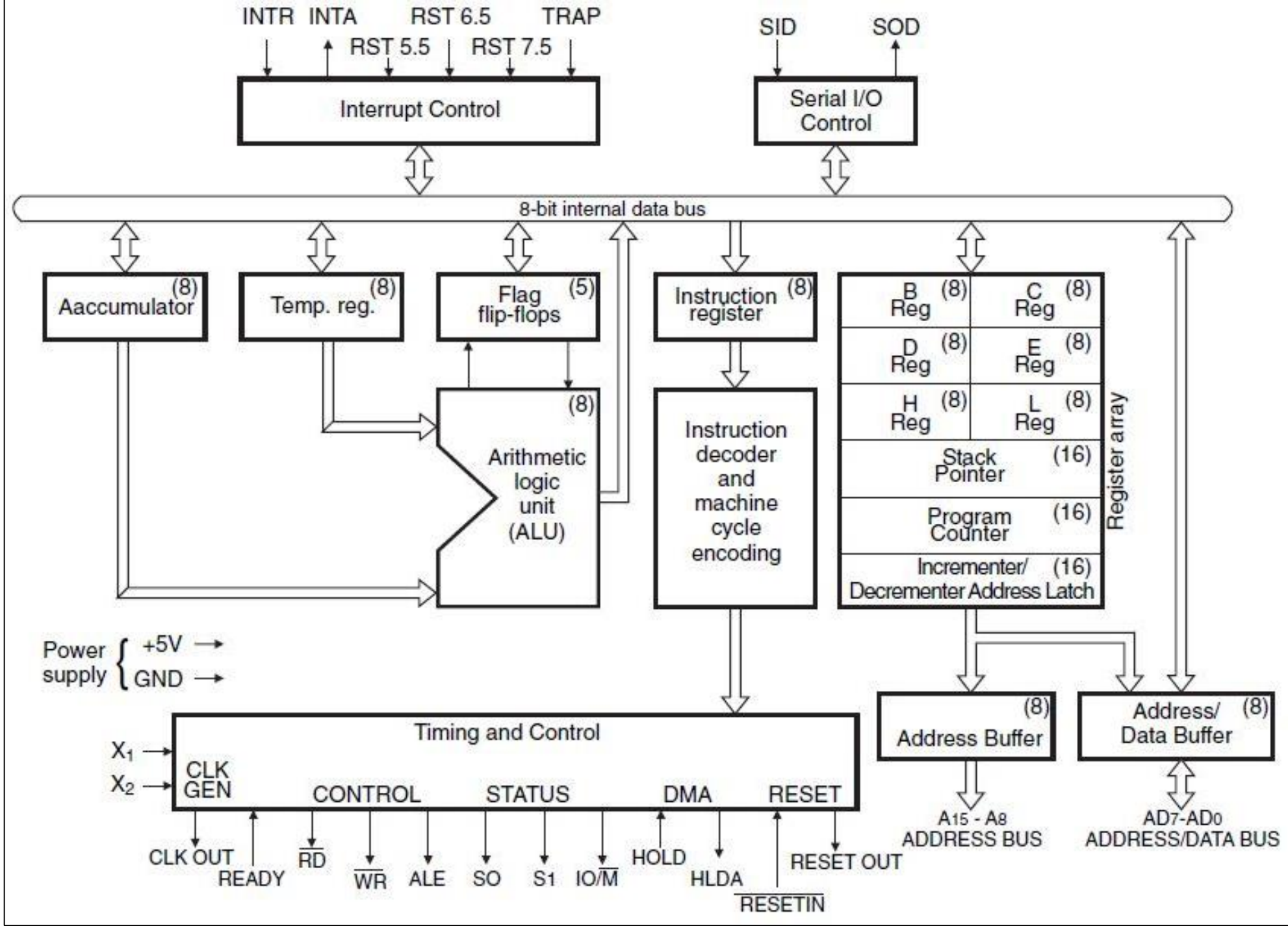
```
// PHASE-2

 PORTB = 0b01010101;
 // Binary combination on port B

  while (1)   //endless loop
  {
     PORTB = ~PORTB;   // Invert port B logic state
     Delay_ms(200);       // 200 mili seconds delay
  } // end of while

} // end of main
```

Alternating the LEDs

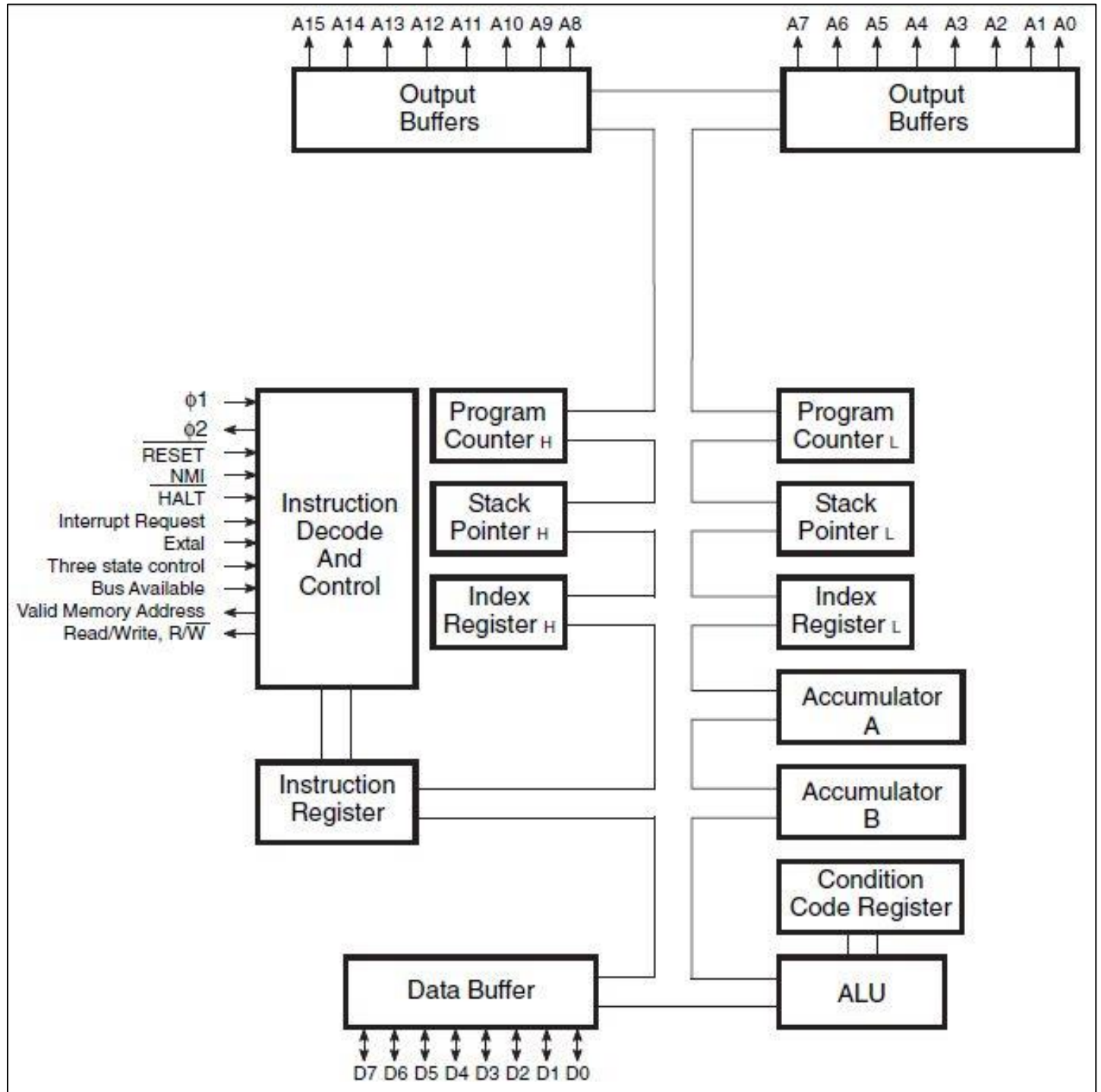


CPU Examples

Intel 8085 8-bit



Motorola 6800 8-bit



Example: Using the MASM32 Assembler (32 bit Intel CPU)

Assembly Source File : ornek.asm

```
; This program adds two numbers, and shows the result on screen.
   .486                          ; Build 32 bit code
   .model flat, stdcall          ; 32 bit memory model
   option casemap :none   ; case sensitive

  ; The header files
   include \masm32\include\masm32.inc
   include \masm32\include\kernel32.inc
   include \masm32\macros\macros.asm      ; MASM supported macros

  ; The library files
   includelib \masm32\lib\masm32.lib
   includelib \masm32\lib\kernel32.lib
```

(contined)

```
   .code                ; Tell MASM where the code starts
start:                  ; The CODE entry point to the program
   mov eax, 100         ; Copy the immediate number 100 into the EAX register
   mov ecx, 250         ; Copy the immediate number 250 into the ECX register
   add ecx, eax         ; ADD EAX to ECX

   print str$(ecx)       ; Show the result at the console (The print is a MASM32 macro.)
   print chr$(13,10)   ; Send ASCII Carriage Return and Line Feed at the console
   print "Program finished."
   exit
end start               ; Tell MASM where the program ends
```

Batch File : ornek.bat
(Batch File for MASM32 Assembler and Linker)

```
c:\masm32\bin\ml   /c /Zd /coff         ornek.asm
c:\masm32\bin\Link   /SUBSYSTEM:CONSOLE   ornek.obj
ornek.exe
pause
```

Example: Using the EASy68K Assembler/Simulator (16 bit Motorola 68000 CPU)

Assembly Source File : ornek.X68

```
; This program adds two numbers, and shows the result on screen.
START   ORG $1000       the program will load into address $1000
    move.b  #3,d1       put 3 in low byte of data register D1
    add.b   #5,d1       add 5 to low byte of data register D1
* D1 contains 8 in its low byte

* Display the results in D1
    move    #14,d0      load task number into D0
* task number 14 is used to display string at (A1)
    lea     textD1,a1   load address of string to display into A1
    trap    #15         trap #15 activates input/output task
```

```
*(continued)

    move    #3,d0       task number 3 into D0
* task number 3 is used to display  contents of D1.L as a number
    trap    #15         display number in D1.l

* Stop execution
    SIMHALT

* dc (define constant) is used to place data in memory
textD1  dc.b    'Resulting D1 contains: ',0   null terminated string

    END     START   end of program
```

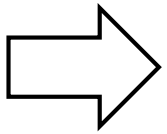
Example: Using Dev-C++ to generate Assembly code from C code

C Source File : ornek.c

```
#include <stdio.h>
int main() {
  int sayi1=10, sayi2=20, sonuc;
  sonuc = sayi1+sayi2;
  printf("Sonuc = %d \n", sonuc);
  return 0;
}
```

Batch File : ornek_disassembly.bat

```
gcc -S -masm=intel ornek.c
gcc -c ornek.c
gcc -m32 -o ornek.exe ornek.c
objdump -d ornek.o > ornek_asm_listing.txt
pause
```



```
ornek.o:     file format pe-x86-64
Disassembly of section .text:
0000000000000000 <main>:
   0:   55                      push   %rbp
   1:   48 89 e5                mov    %rsp,%rbp
   4:   48 83 ec 30             sub    $0x30,%rsp
   8:   e8 00 00 00 00          callq  d <main+0xd>
   d:   c7 45 fc 0a 00 00 00    movl   $0xa,-0x4(%rbp)
  14:   c7 45 f8 14 00 00 00    movl   $0x14,-0x8(%rbp)
  1b:   8b 55 fc                mov    -0x4(%rbp),%edx
  1e:   8b 45 f8                mov    -0x8(%rbp),%eax
  21:   01 d0                   add    %edx,%eax
  23:   89 45 f4                mov    %eax,-0xc(%rbp)
  26:   8b 45 f4                mov    -0xc(%rbp),%eax
  29:   89 c2                   mov    %eax,%edx
  2b:   48 8d 0d 00 00 00 00    lea    0x0(%rip),%rcx        # 32 <main+0x32>
  32:   e8 00 00 00 00          callq  37 <main+0x37>
  37:   b8 00 00 00 00          mov    $0x0,%eax
  3c:   48 83 c4 30             add    $0x30,%rsp
  40:   5d                      pop    %rbp
  41:   c3                      retq
```