CHAPTER 1
A VERY BRIEF HISTORY
OF MICROPROCESSORS & MICROCONTROLLERS

Intel 4004
• Introduced in 1971.
• Federico Faggin design.
• first-ever single-chip microprocessor
• 2300 transistor
• performs 60,000 operations / second.
• approximately the same performance as the 18,000 vacuum tube ENIAC.
• 4-bit Intel C4004 clock Speed is 108 kHz

Intel 8008
• 1972: Faggin begins work on an 8-bit processor, the Intel 8008. The prototype has serious problems with electrical charge leaking out of its memory circuits.
• Intel's 8008 is well-received, but system designers want increased speed, easier interfacing, and more I/O and instructions. The improved version is 8080.

Zilog Z80
• Faggin leaves Intel to start his own company Zilog, who later produce the Z80. 3.5MHz Z80 is a very popular processor taught in many universities)
• ... and, later, a 16-bit Z8000. But INTEL is getting more powerful.

The Zilog Z80
• The Z80 is an 8 bit CPU with a 16 bit address bus capable of direct access of 64K of memory space.
• It was based on the 8080, it has a large instruction set.
• Programming features include an accumulator and six eight bit registers that can be paired as 3-16 bit registers. In addition to the general registers, a stack-pointer, program-counter, and two index (memory pointers) registers are provided.
• It had a 40 pin DIP package manufactured in A, B, and C models, differing only in maximum clock speed. It was also manufactured as a stand-alone microcontroller with various configurations of on-chip RAM and EPROM.
• It proves useful for low cost control applications.

Early Microcontrollers
• 1974: Motorola (originally car radio manufacturers) had introduced transistors in the 1950s and decided to make a late but serious effort in the microprocessor market. They announced their 8-bit 6800 processor.
• 1975: General Motors approach Motorola about a custom-built derivative of the 6800. Motorola's long experience with automobile manufacturers pays off and Ford follow GM's lead.
• 1976: Intel introduce an 8-bit microcontroller, the MCS-48. They ship 251,000 in this year.
• 1980: Intel introduced 8051, an 8-bit microcontroller with on-board EPROM. They ship 22 million and 91 million in 1983.
• Early 90s: PIC is introduced to meet a demand for a cheap, small and practical microcontroller which was both easy to use and program.

CHAPTER 2
PIC
Peripheral Interface Controller
Microcontrollers are small computer on a chip with some special properties:

- CPU, code memory, data memory and IO ports all included on a single chip
- Dedicated to one task
- Small and low cost
- Embedded in many consumer devices

**What is a microcontroller?**

**Why PIC is popular?**

- PICs are popular with developers due to
  - low cost
  - wide availability
  - large user base
  - extensive collection of application notes
  - availability of low cost or free development tools
  - serial programming capability.

PIC is very small and easy to implement for not complex problems and usually accompanies to the microprocessors as an interface. For example:

**Family Core Architectural Differences**

<table>
<thead>
<tr>
<th>Baseline Core Devices (ex:12C50x, 16C5x)</th>
<th>Mid-Range Core Devices (ex:12C50x, 16C5x)</th>
</tr>
</thead>
<tbody>
<tr>
<td>• 12 bit wide code memory,</td>
<td>• 14 bit wide code memory</td>
</tr>
<tr>
<td>• tiny two level deep call stack.</td>
<td>• improved 8 level deep call stack.</td>
</tr>
<tr>
<td>• 33 instructions</td>
<td>• 35 instructions</td>
</tr>
<tr>
<td>• PIC10, PIC12 &amp; PIC16 devices</td>
<td>• increased opcode width allows addressing of more memory</td>
</tr>
<tr>
<td>• 4 pin to 40 pin packages.</td>
<td>• PIC12 and PIC16 devices.</td>
</tr>
</tbody>
</table>

**PIC10, PIC12 & PIC16 Devices**

- 4 pin to 40 pin packages.
- small and low cost
- embedded in many consumer devices

**PIC17 High End Core Devices (Ex:17C4x, 17C7xx)**

- never became popular and superseded by the PIC18 architecture.
- 16 bit wide opcodes (allowing many new instructions) : 58 instructions
- 16 level deep call stack. Packages of 40 to 68 pins.
- a memory mapped accumulator
- read access to code memory (table reads)
- direct register to register moves
- an external program memory interface to expand the code space
- an 8bit x 8bit hardware multiplier
- auto-increment/decrement addressing

**PIC18 High End Core Devices (ex:18Cxxx)**

- new high end pic architecture
- it inherits most of the features and instructions of the 17 series,
- 77 instructions, much deeper call stack (31 levels deep)
- the call stack may be read and written
- offset addressing mode
- a new indexed addressing mode in some devices

**PIC24 and dsPIC 16 bit Microcontrollers**

- architectures differ significantly from prior models.
  - dsPICs are Microchip's newest family (started in 2004)
  - digital signal processing capabilities.
  - Microchip's first inherent 16-bit (data) microcontrollers.
  - hardware MAC (multiply-accumulate)
  - barrel shifting
  - bit reversal
  - 16x16-bit multiplication
  - other digital signal processing operations.
  - can be efficiently programmed in C

**PIC SPEED**

- can use crystals, clock oscillators, or even an RC circuit.
- some PICs have a built in 4MHz RC clock, not very accurate, but requires no external components.
- instruction speed = 1/4 clock speed (Tcyc = 4 * Tclk)
- all PICs can be run from DC to their maximum spec’d speed:
  - 12C50x 4MHz
  - 12C67x 10MHz
  - 16Cxxx 20MHz
  - 17C4x / 17C7xx 33MHz
  - 18Cxxx 40MHz

**Register Addressing Modes**

- Immediate Addressing
  - Movlw H’0F’

- Direct Addressing
  - uses 7 bits of 14 bit instruction to identify a register file address
  - 8th and 9th bit comes from RP0 and RP1
  - bits of STATUS register:
    - Z = equ D’2’
    - bit Set STATUS, Z

- Indirect Addressing
  - Full 8 bit register address is written the special function register FSR
  - IND is used to get the content of the address pointed by FSR
  - Exp : A sample program to clear RAM locations H'20’ – H'2F’
**PIC MEMORY CAPACITY**

- **Device**
  - PIC16F877: 1024 128 224 18
  - PIC16F828: 2048 128 224 18
  - PIC16F84: 512 64 36 16
  - PIC16F844: 1024 64 68 18
  - PIC16F870: 2048 128 240 40
  - PIC16F871: 2048 128 40 20
  - PIC16F878: 4096 128 198 28
  - PIC16F874: 4096 128 102 40
  - PIC16F876: 8192 256 366 40
  - PIC16F877: 8192 256 368 40

- **PIC MEMORY CAPACITY**

- **SPECIAL FUNCTION REGISTERS**

<table>
<thead>
<tr>
<th>Register</th>
<th>Address</th>
<th>Bank</th>
<th>Tutorial</th>
</tr>
</thead>
<tbody>
<tr>
<td>EEADR</td>
<td>09</td>
<td>0</td>
<td>25</td>
</tr>
<tr>
<td>EECOCN</td>
<td>08</td>
<td>1</td>
<td>25</td>
</tr>
<tr>
<td>EECOCN</td>
<td>09</td>
<td>1</td>
<td>25</td>
</tr>
<tr>
<td>EEDATA</td>
<td>00</td>
<td>0</td>
<td>25</td>
</tr>
<tr>
<td>EEDATA</td>
<td>00</td>
<td>0</td>
<td>25</td>
</tr>
<tr>
<td>SFIR</td>
<td>04</td>
<td>0</td>
<td>16</td>
</tr>
<tr>
<td>SFIR</td>
<td>04</td>
<td>0</td>
<td>16</td>
</tr>
<tr>
<td>INTCON</td>
<td>0C</td>
<td>0</td>
<td>18</td>
</tr>
<tr>
<td>INTCON</td>
<td>0C</td>
<td>0</td>
<td>18</td>
</tr>
<tr>
<td>OPTION</td>
<td>02</td>
<td>0</td>
<td>18</td>
</tr>
<tr>
<td>OPTION</td>
<td>02</td>
<td>0</td>
<td>18</td>
</tr>
<tr>
<td>PCL</td>
<td>0E</td>
<td>0</td>
<td>14</td>
</tr>
<tr>
<td>PCL</td>
<td>0E</td>
<td>0</td>
<td>14</td>
</tr>
<tr>
<td>PORTA</td>
<td>06</td>
<td>0</td>
<td>4</td>
</tr>
<tr>
<td>PORTA</td>
<td>06</td>
<td>0</td>
<td>4</td>
</tr>
<tr>
<td>STATUS</td>
<td>03</td>
<td>0</td>
<td>4</td>
</tr>
<tr>
<td>STATUS</td>
<td>03</td>
<td>0</td>
<td>4</td>
</tr>
<tr>
<td>TMRO</td>
<td>04</td>
<td>0</td>
<td>16</td>
</tr>
<tr>
<td>TMRO</td>
<td>04</td>
<td>0</td>
<td>16</td>
</tr>
<tr>
<td>TRISA</td>
<td>06</td>
<td>1</td>
<td>4</td>
</tr>
<tr>
<td>TRISA</td>
<td>06</td>
<td>1</td>
<td>4</td>
</tr>
</tbody>
</table>

**SAMPLE PROGRAM**

<table>
<thead>
<tr>
<th>List coun</th>
<th>Prog coun</th>
<th>Code</th>
<th>Value</th>
<th>Source code</th>
</tr>
</thead>
<tbody>
<tr>
<td>0000</td>
<td>0000</td>
<td>0000</td>
<td>0000</td>
<td></td>
</tr>
<tr>
<td>0001</td>
<td>0000</td>
<td>0000</td>
<td>0000</td>
<td></td>
</tr>
<tr>
<td>0002</td>
<td>0000</td>
<td>0000</td>
<td>0000</td>
<td></td>
</tr>
<tr>
<td>0003</td>
<td>0000</td>
<td>0000</td>
<td>0000</td>
<td></td>
</tr>
<tr>
<td>0004</td>
<td>0000</td>
<td>0000</td>
<td>0000</td>
<td></td>
</tr>
<tr>
<td>0005</td>
<td>0000</td>
<td>0000</td>
<td>0000</td>
<td></td>
</tr>
<tr>
<td>0006</td>
<td>0000</td>
<td>0000</td>
<td>0000</td>
<td></td>
</tr>
<tr>
<td>0007</td>
<td>0000</td>
<td>0000</td>
<td>0000</td>
<td></td>
</tr>
<tr>
<td>0008</td>
<td>0000</td>
<td>0000</td>
<td>0000</td>
<td></td>
</tr>
<tr>
<td>0009</td>
<td>0000</td>
<td>0000</td>
<td>0000</td>
<td></td>
</tr>
<tr>
<td>0010</td>
<td>0000</td>
<td>0000</td>
<td>0000</td>
<td></td>
</tr>
<tr>
<td>0011</td>
<td>0000</td>
<td>0000</td>
<td>0000</td>
<td></td>
</tr>
<tr>
<td>0012</td>
<td>0000</td>
<td>0000</td>
<td>0000</td>
<td></td>
</tr>
<tr>
<td>0013</td>
<td>0000</td>
<td>0000</td>
<td>0000</td>
<td></td>
</tr>
<tr>
<td>0014</td>
<td>0000</td>
<td>0000</td>
<td>0000</td>
<td></td>
</tr>
<tr>
<td>0015</td>
<td>0000</td>
<td>0000</td>
<td>0000</td>
<td></td>
</tr>
<tr>
<td>0016</td>
<td>0000</td>
<td>0000</td>
<td>0000</td>
<td></td>
</tr>
<tr>
<td>0017</td>
<td>0000</td>
<td>0000</td>
<td>0000</td>
<td></td>
</tr>
<tr>
<td>0018</td>
<td>0000</td>
<td>0000</td>
<td>0000</td>
<td></td>
</tr>
<tr>
<td>0019</td>
<td>0000</td>
<td>0000</td>
<td>0000</td>
<td></td>
</tr>
<tr>
<td>0020</td>
<td>0000</td>
<td>0000</td>
<td>0000</td>
<td></td>
</tr>
<tr>
<td>0021</td>
<td>0000</td>
<td>0000</td>
<td>0000</td>
<td></td>
</tr>
<tr>
<td>0022</td>
<td>0000</td>
<td>0000</td>
<td>0000</td>
<td></td>
</tr>
<tr>
<td>0023</td>
<td>0000</td>
<td>0000</td>
<td>0000</td>
<td></td>
</tr>
<tr>
<td>0024</td>
<td>0000</td>
<td>0000</td>
<td>0000</td>
<td></td>
</tr>
<tr>
<td>0025</td>
<td>0000</td>
<td>0000</td>
<td>0000</td>
<td></td>
</tr>
<tr>
<td>0026</td>
<td>0000</td>
<td>0000</td>
<td>0000</td>
<td></td>
</tr>
<tr>
<td>0027</td>
<td>0000</td>
<td>0000</td>
<td>0000</td>
<td></td>
</tr>
</tbody>
</table>

**PIC 16F877**

- 8 kbytes of FLASH
- 900 bytes of Data Memory (RAM)
- 256 bytes of EEPROM Data Memory
- 33 input or output pins
- 20 MHz operating speed (200 ns instruction cycle)
- Max. 25 mA current from an output pin

**PIC Program Memory**

- 12C508 512 12bit instructions
- 16C71C 1024 (1k) 14bit instructions
- 16F877 8192 (8k) 14bit instructions
- 17C766 16384(16k) 16bit instructions
PIC Data Memory

PICs use general purpose “file registers” for RAM (each register is 8 bits for all PICs)

Some examples are:

- 12C508: 25 Bytes RAM
- 16C71C: 36 Bytes RAM
- 16F877: 368 Bytes (plus 256 Bytes of nonvolatile EEPROM)
- 17C766: 902 Bytes RAM

Don’t forget, programs are stored in program space (not in data space), so low RAM values are OK.
CHAPTER 3
PIC PROGRAMMING

Programming PIC 16F877
- Assembler (MPLAB)
- Basic (Pic Basic Pro)
- C (HITEC PICC)

```
main()
{
    set_TRIS_B(0x00);
    while(true)
    {
        if (input(PIN_B0))
        {
            output_high(PIN_B0);
        }
        else
        {
            output_low(PIN_B0);
        }
    }
}
```

PIC BASIC PROGRAMMING

Minimum circuitry for PIC16F877

LED (light emitting diode) flasher

```
LOOP:
    HIGH PORTB.0
    PAUSE 500
    LOW PORTB.0
    PAUSE 500
    GOTO LOOP
```
**BUTTON READ**

```plaintext
INPUT PORTD.2

LOOP:
  IF PORTD.2=1 THEN
    HIGH PORTB.0
  ELSE
    LOW PORTB.0
  ENDIF
GOTO LOOP
```

**LCD**

```plaintext
DEFINE OSC 4
DEFINE LCD_DREG PORTB
DEFINE LCD_DBIT 4
DEFINE LCD_RSREG PORTB
DEFINE LCD_RSBIT 0
DEFINE LCD_EBIT 1
DEFINE LCD_COMMAND US 2000
DEFINE LCD_DATA US 50

LCDOUT 254,1, "MERHABA"
LCDOUT 254,192, "2x16 LCD"
END
```

**Analog to Digital Conversion**

Format: ADCIN Channel, Var

Sample Program:

```plaintext
ABC VAR BYTE
ADCON1=2 'PORTA is analog
INPUT PORTA.0
LOOP:
  ADCIN PORTA.0,ABC
  LCDOUT 254,1,#ABC
  PAUSE 100
GOTO LOOP
```

**Pulse Width Modulation**

Format: HPWM Channel, Dutycycle, Frequency

Sample Program:

```plaintext
DEFINE CCP1_REG PORTC 'Hpwm 1 pin port
DEFINE CCP1_BIT 2 'Hpwm 1 pin bit

DUTY VAR BYTE

DONGU:
  FOR i=0 TO 255
    HPWM 1, DUTY,1000
    DUTY=DUTY+1
    PAUSE 50
  NEXT
GOTO DONGU
```

**Programming in C**

- Programming the PIC in C offers several advantages:
  - Higher level language – developer is insulated from details of the chip
  - Library support for common tasks (string manipulation, serial communication)
- We use the CCS compiler ([http://www.ccsinfo.com/](http://www.ccsinfo.com/)) which don’t suck. All examples will use CCS code
PIC – Common Tasks with CCS

• Program Template. Starting point for just about everything

```c
#define <16F877.h> // Define the type of chip you're using.
// Makes it easier to switch chips
#use delay(clock=20000000) // 20Mhz oscillator

void main()
{
    /* Initialization Code goes here */
    while (TRUE)
    {
        /* Program Code goes here */
    }
}
```

• Digital I/O

– Standard I/O vs. Fast I/O
– Using (standard I/O):

```c
// Output a high on PIN_D1, low on PIN_D2
// Wait 50 us and invert
output_high(PIN_D1);
output_low(PIN_D2);
delay_us(50);
output_low(PIN_D1);
output_high(PIN_D2);
```

PIC – Common Tasks with CCS

• Analog Input

– Initialization:

```c
setup_adc_ports(ALL_ANALOG);
setup_adc(ADC_CLOCK_DIV_2);
```

– Picking a channel:

```c
set_adc_channel(0); // Note: must wait between changing
                    // input channels (~ 10us)
```

– Inputting Data:

```c
unsigned int16 data; // Declare a 16-bit integer
data = read_adc(); // Read a 10-bit value from the // selected channel
```

• Using PWM

– Initialization:

```c
setup_timer_2(T2_DIV_BY_1,249,1); // Setup the PWM period
setup_ccp1(CCP_PWM); // Set CCP1 for PWM
```

– Setting the Duty Cycle:

```c
set_pwm1_duty(500); // See the CCS examples for the formula // for setting the PWM period and duty // cycle
```

PIC – Tips for Software Design

• Design the program as a state machine

– A main() loop, with:
  • A switch() statement that jumps to a function() which represents the actions that occur in that state
  • Each state function() has an output section and a transition section (which can change the current state variable)

– Interrupts are very useful (for example: interrupt when data received on serial port), but can cause problems.
  • i.e. if you change state during an interrupt (such as an E-stop), return from the interrupt service routine, then change the state variable again (during the transition section) the interrupt change is lost.
– Design with tuning and debugging in mind
  • Programmer time is more important than machine time – the PIC16F877 is plenty fast

PIC – Tips for Debugging

• Use a protoboard with RS232 support and lots of print statements. Example:

– program waits for a switch press
– reads an analog voltage
– changes the PWM cycle accordingly
PIC – Tips for Debugging

```c
void Some_function()
{
    int1 pushed = FALSE, last_pushed = FALSE;
    int16 analog_value;
    float volts;
    pushed = input(PIN_D3);
    if (pushed && !last_pushed) {
        puts("Button Pushed!");
        analog_value = read_adc(); /* 10-bit analog input value is * between 0-1023 0-5V range */
        volts = 5.0 * (analog_value / 1024.0);
        printf("Button pushed! Analog value is %f volts, PWM to %i
", volts, analog_value);
        set_pwm1_duty(analog_value);
        /* We've pre-configured PWM channel 1 - the set_pwm1_duty cycle function accepts * a 10-bit number and adjusts the cycle accordingly */
    }
}
```

CHAPTER 4

PIC PROGRAMMERS

PIC-PG2 - SERIAL PORT PROGRAMMER FOR 8/18/28/40 PIN PIC MICROCONTROLLERS AND I2C EEPROMS + ICSP connector and cable

http://www.olimex.com/dev/pic-pg2.html

IC-Prog Prototype Programmer
Programs:
- 12Cxx, 16Cxx, 16Fxxx, 16F87x,
- 18Fxxx, 16F7x, 24Cxx, 93Cxx,
- 90Sxxx, 59Cxx, 89Cx051,
- 89S53, 250x0, PIC, AVR, 80C51 etc.

http://www.ic-prog.com/

BOOTLOADER

A bootloader is a program that stays in the microcontroller and communicates with the PC (usually through the serial interface). The bootloader receives a user program from the PC and writes it in the flash memory, then launches this program in execution.
• The PIC16F877 has on-board FLASH memory
  – No burner needed to reprogram the PIC
  – No need to remove PIC from circuit
• Using a bootloader on the PIC, and a bootloader utility on the PC the PIC can be reprogrammed in seconds over a serial link.
  – Burn the bootloader code onto the PIC
  – When writing your program in C tell the compiler not to use the top 255 bytes of flash memory
  – Connect the PIC circuit to the PC via a serial link. Run the bootloader code from the PC and download your code to the circuit in seconds.

http://www.microchip.com/PIC16bootload/

CHAPTER 5
PIC ISIS
SIMULATION
APPLICATIONS

APPLICATION 1
PIC COLOR SENSOR

ISIS CIRCUIT SCHEME
DETAILS OF THE CIRCUIT

• Reads the color of the surface from the distance between 5-40mm. Three colors (red, blue, green) can be detected and shown on red, blue, green indicator LEDs.

• Red, Blue, Green LED diodes emit lights, and the reflected light read with LDR. LDR output is converted to a digital value using ADC channel 0 of PIC.

DEFINE OSC 4
DEFINE ADC_BITS 8
DEFINE ADC_CLOCK 3
DEFINE ADC_SAMPLEUS 50
ADCON1 = 2
INPUT PORTA.0
TRISB = 0
SYMBOL VERI = PORTA.0
SYMBOL RED_IND = PORTB.3
SYMBOL GREEN_IND = PORTB.5
SYMBOL BLUE_IND = PORTB.4
SYMBOL RED = PORTB.0
SYMBOL GREEN = PORTB.1
SYMBOL BLUE = PORTB.2
DAT_VAR BYTE
DAT_R VAR BYTE
DAT_G VAR BYTE
DAT_B VAR BYTE

CLEAR
MAIN:
cALL read_val
GOTO MAIN
readVal:
HIGH RED
PAUSE 5
CALL ADC_RD
DAT_R = DAT
LOW RED
PAUSE 5
HIGH GREEN
PAUSE 5
CALL ADC_RD
DAT_G = DAT
LOW GREEN
PAUSE 5
HIGH BLUE
PAUSE 5
IF DAT_R < BILGI_B && DAT_R < DAT_Y THEN
HIGH KIRMLED
LOW MAVILED
LOW YESILED
ENDIF
IF DAT_G < DAT_B && DAT_G < DAT_R THEN
HIGH YESILED
LOW MAVILED
LOW KIRMLED
ENDIF
IF DAT_B < DAT_G && DAT_B < DAT_R THEN
HIGH MAVILED
LOW KIRMLED
LOW YESILED
PAUSE 20
ENDIF
RETURN
ADC_READ:
ADCIN VERI, BILGI
RETURN

RGB VALUES

COLOR          RED        GREEN        BLUE
GLOSSY RED     3.14 V    1.23 V       1.11 V
DARK RED       2.99 V    1.45 V       1.36 V
DARK MATTE GREEN 2.24 V  2.89 V       3.11 V
LIGHT GLOSSY GREEN 1.27 V  3.78 V       3.37 V
MATTE GREEN    1.51 V    3.44 V       2.39 V
LIGHT MATTE BLUE 1.11 V  2.84 V       3.24 V
MATTE BLUE     1.23 V    2.87 V       3.17 V
DARK GLOSSY BLUE 2.22 V  2.57 V       3.36 V

APPLICATION 2
RF CONTROLLED MOTOR

433,92 MHZ
CMCON=07 'PortA
INCLUDE "modedefs.bas"
OPTION_REG.7 = 1 'PortB Pull_Up Active
TRISB = %11110000
TRISA = %00000000
K VAR BYTE
K=0
PAUSE 500
Serout2 PORTA,0,16780,[REP$AA5,REP$0000$FF05]
; preamble + Synchron Sending
MAINLOOP:
IF PORTB.4=1 then K.BIT0=1
IF PORTB.5=1 then K.BIT1=1
IF PORTB.6=1 then K.BIT2=1
IF PORTB.7=1 then K.BIT3=1
SEROUT PORTB.7,N2400, [254]
SEROUT PORTB.7,N2400, [K]
SEROUT PORTB.7,N2400, [192]
PAUSE 16
K=0
GOTO MAINLOOP

CMCON=07 'PortA
TRISB = %00000010
DEFINE HSER_RCSTA 90h
DEFINE HSER_TXSTA 20h
DEFINE HSER_BAUD 2400
DEFINE HSER_CLROERR 1
LEFTFWD VAR PORTB.3
LEFTREV VAR PORTB.4
RIGHTFWD VAR PORTB.5
RIGHTREV VAR PORTB.6
K VAR BYTE
ERRCHK VAR BYTE
PAUSE 250
MAINLOOP:
HSERIN [WAIT(254),K,ERRCHK]
LEFTREV=0; RIGHTREV=0;
RIGHTFWD=0; LEFTFWD=0;
IF ERRCHK=192
THEN
IF K.BIT0=1 THEN
LEFTFWD=1
ELSE
LEFTFWD=0
ENDIF
ELSE
RIGHTFWD=0
ENDIF
IF K.BIT1=1 THEN
LEFTREV=1
ELSE
LEFTREV=0
ENDIF
ELSE
RIGHTREV=0
ENDIF
IF K.BIT2=1 THEN
RIGHTFWD=1
ELSE
RIGHTFWD=0
ENDIF
ELSE
RIGHTREV=0
ENDIF
PAUSE 10
GOTO ANADONGU

REFERENCES
- www.microchip.com (Official website of the PIC manufacturer, PIC16F877 datasheet & some application notes are available)
- http://www.ipixton диркон.co.uk/pic/
- http://www.copic.com/
- www.antrak.org (Ankara amateur radio society website)
- www.picproje.net (A discussion forum on PIC in Turkish)
- www.elektroda.pl (A discussion forum on PIC)
- robot.metu.edu.tr (METU Robot Society website)
- http://www.0m0r.cn/-msmmms/pic/ (code archive)
- http://www.workingtext.com/hpic/ (examples)

REFERENCES-2
- http://www.0limex.com/dev/pic-pg2.html
C and bootloader links

• CCS PIC C Compiler
  – http://www.ccsinfo.com/
• CCS PIC C Compiler Manual
  – http://www.ccsinfo.com/piccmanual3.zip
• Bootloader Code (that resides on the PIC)
  – http://www.workingtex.com/htpic/PIC16F87x_and_PIC16F7x_bootloader_v7-40.zip
• Bootloader Program (that resides on the PC)
  – http://www.ehl.cz/pic/