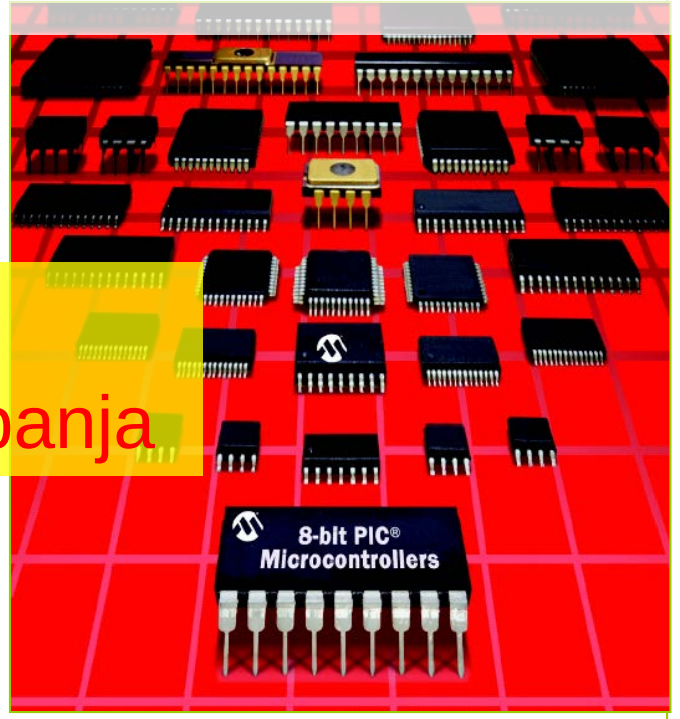
modul a3

integrisani mikroračunarski sistemi



UVOD

kratka priprema za praktična vežbanja

Sadržaj:

Mikroprocesori i mikrokompjuteri

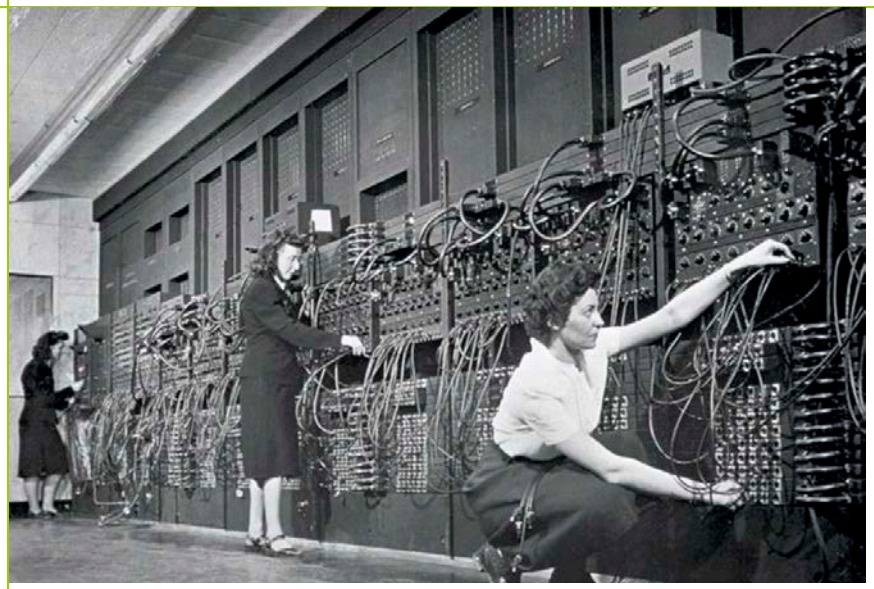
Mikroprocesor je kompleksno elektronsko kolo (VLSI – Very large scale integration) izvedeno u jednom čipu, koje sadrži podsisteme za izvodjenje logičkih, aritmetičkih, komunikacionih i upravljačkih funkcija. Ovi sistemi zbirno/integrisano čine jednu zaokruženu funkcionalnu celinu.

Kada je mikroprocesor ugradjen na jednoj, zajedničkoj štampanoj ploči, zajedno sa memorijskim jedinicama i pratećim interfejsima, onda se takav sklop naziva mikrokompjuter (*single-board computer*). Ukoliko je ovakav sklop smešten na jedan čip, onda se takav kompjuter naziva integrisani mikrokompjuter (*single-chip computer*).

U aplikativno-tehničkom smislu, postoje dve grane razvoja/evolucije mikroprocesora:

- mikroprocesori specijalizovani za primene u personalnim računarima i radnim stanicama (grafičke radne stanice, industrijski računari, CNC upravljački sistemi otvorene arhitekture, ...), gde su osnovni zahtevi brzina rada i širina magistrale (dužina reči 32, 64 i 128 bita)
- integrisani mikroračunarski sistemi – mikrokontroleri (uC) i digitalni procesori signala (DSP), gde je osnovni zahtev multifunkcionalnost ostvarena u minimalnim fizičkim okvirima – visok stepen integracije funkcija.

Ubrzani tehnološki razvoj u ovoj oblasti radikalno menja performanse u prethodno navedenim klasama, tako da i kvantitativna odredjenja i definicije imaju vremensku ograničenost.

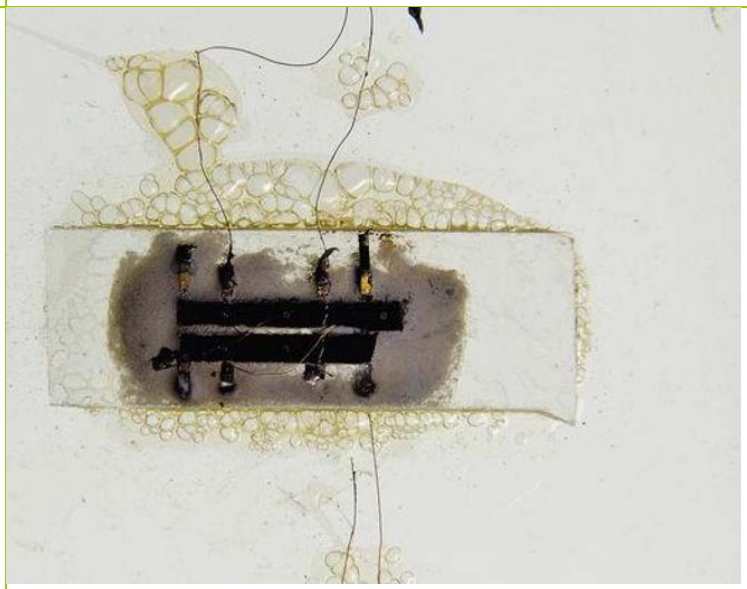


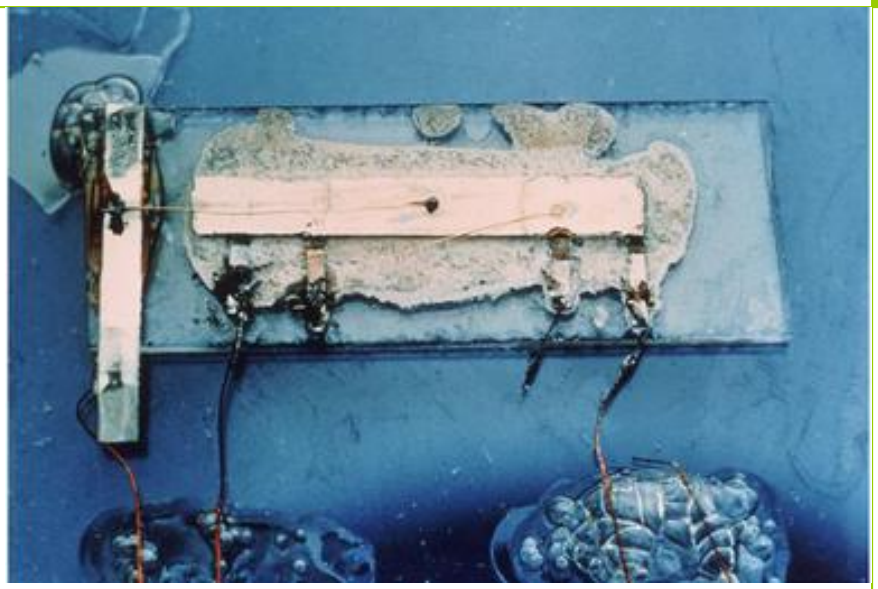
ENIAC

Projekat pokrenut 1943, pušten u rad 1947, radio do 1955 ...

Skraćenica od engleske složenice **Electronic Numerical Integrator And Computer** označava prvi digitalni elektronski računar kog je bilo moguće programirati u cilju rešavanja širokog spektra računarskih problema, mada su i raniji računari pravljeni sa nekim od ovih osobina. ENIAC je dizajniran i izgrađen u svrhu izračunavanja balističkih tablica za američku vojsku. Prvi problemi rešavani na ENIAC-u bili su, pak, povezani sa izradom hidrogenske bombe.

... mogao je da obavlja 385 operacija množenja u sekundi!



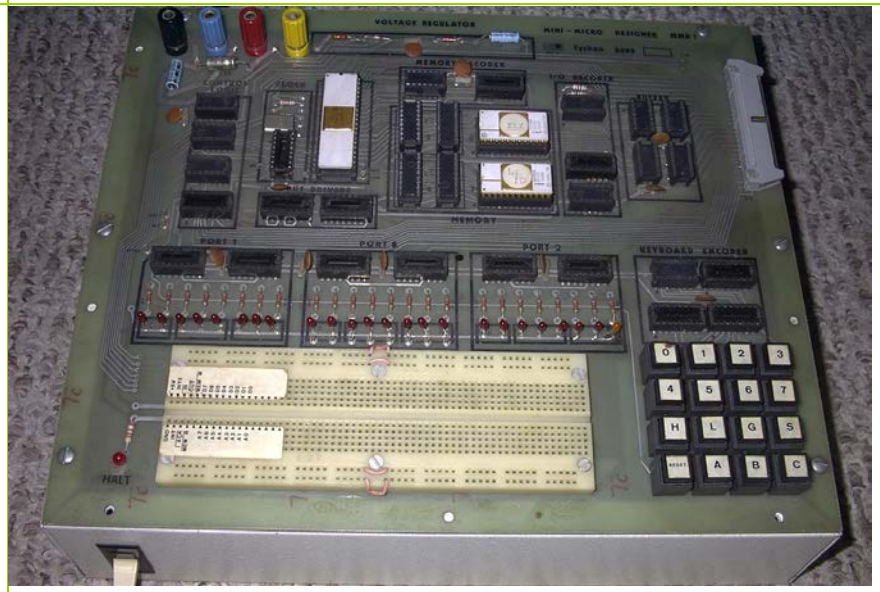


INTEGRISANO KOLO

The first planar monolithic integrated circuit (IC) chip was demonstrated in 1960. These ideas could not be implemented by the industry, until a breakthrough came in late 1958. Three people from three U.S. companies solved three fundamental problems that hindered the production of integrated circuits. **Jack Kilby** of Texas Instruments patented the principle of integration, created the first prototype ICs and commercialized them.

Kilby was awarded the 2000 Nobel Prize in Physics "for his part in the invention of the integrated circuit".

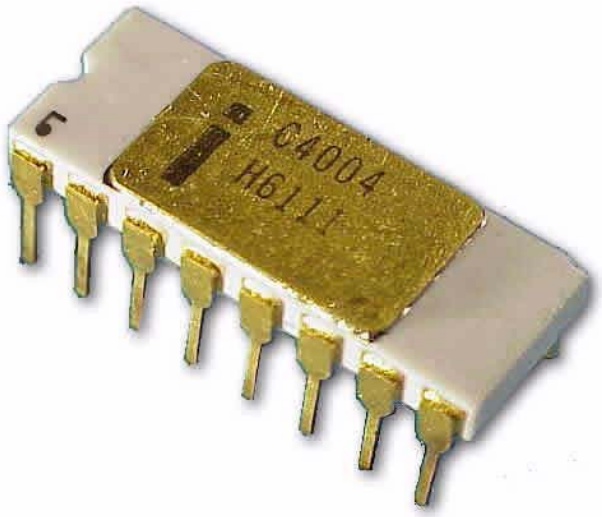
The first monolithic IC chip was invented by Robert Noyce of Fairchild Semiconductor.



Single-board computer (SBC)

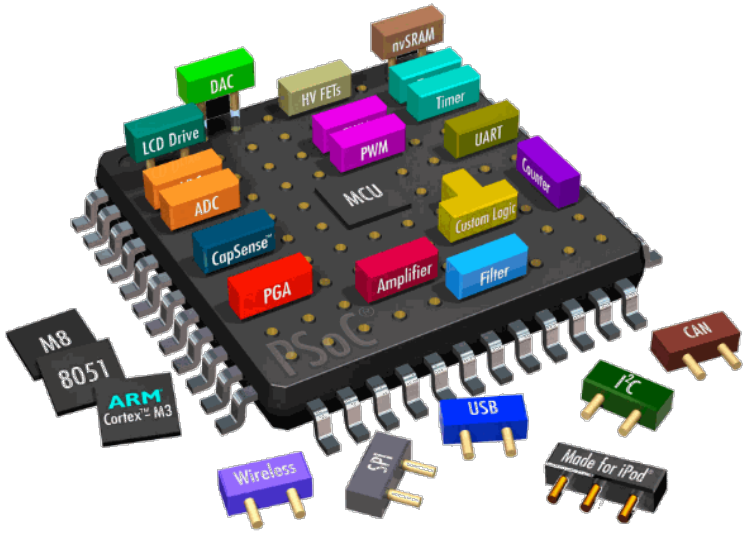
A single-board computer (SBC) is a complete computer built on a single circuit board, with microprocessor(s), memory, input/output (I/O) and other features required of a functional computer.
The first true single-board computer called the "dyna-micro" was based on the Intel C8080A, and also used Intel's first EPROM, the C1702A. The dyna-micro was re-branded by E&L Instruments of Derby, Connecticut, in **1976** as the "MMD-1"

The Raspberry Pi is a low-cost single-board computer used to teach computer science



The Birth of the Microprocessor

Microprocessors were invented by - Ted Hoff, along with a handful of visionary colleagues working at a young Silicon Valley start-up called Intel.
The Intel 4004 is considered the first microprocessor—in other words, the first general-purpose computer on a chip – 1971.



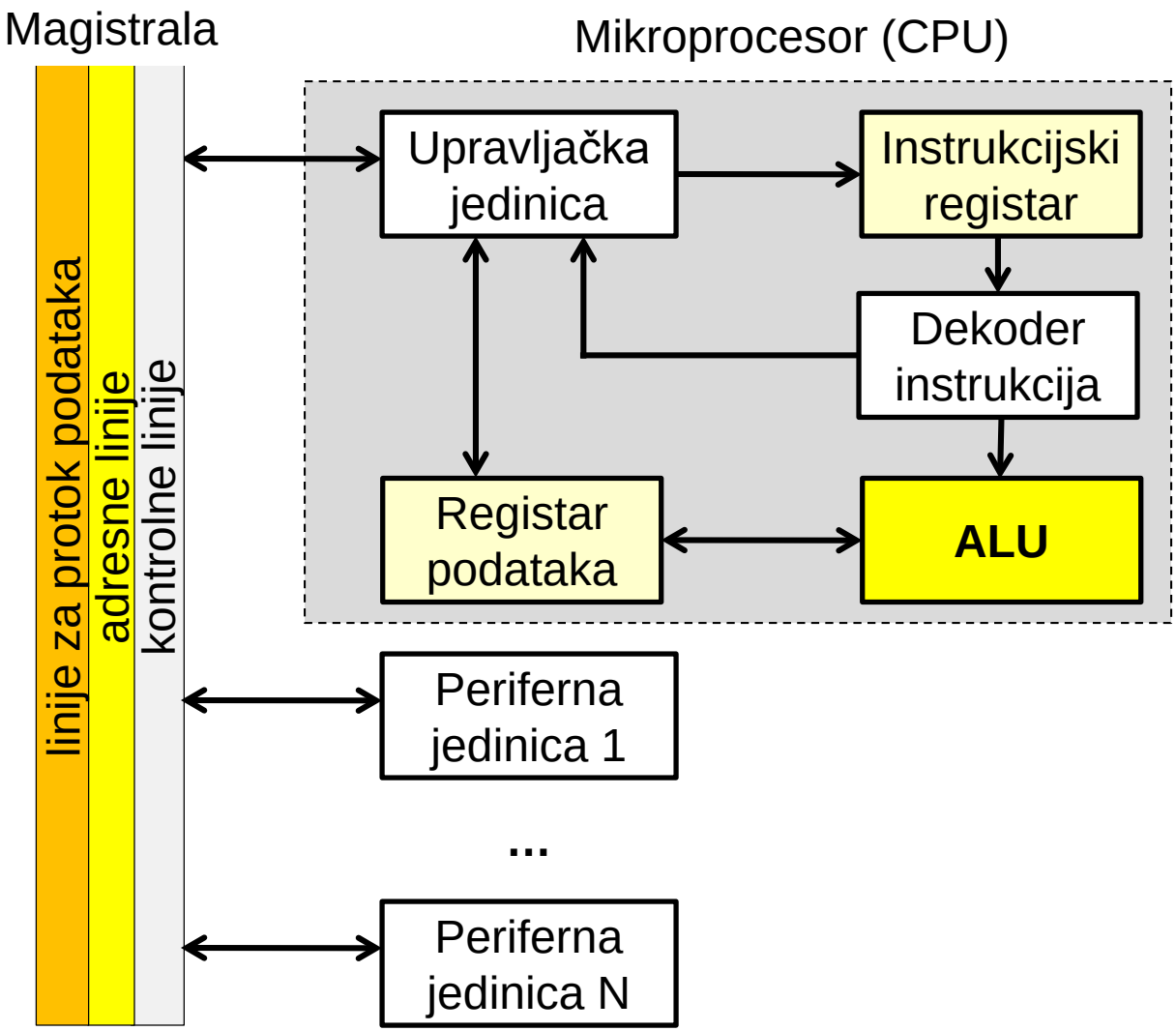
A system on a chip (SoC)

A system on a chip (SoC) is an integrated circuit (also known as a "chip") that integrates all or most components of a computer or other electronic system. These components almost always include a central processing unit (CPU), memory, input/output ports and secondary storage, often alongside other components such as radio modems and a graphics processing unit (GPU) – all on a single substrate or microchip. It may contain digital, analog, mixed-signal, and often radio frequency signal processing functions (otherwise it is considered only an application processor).
SoCs are very common in the mobile computing (such as in smartphones and tablet computers) and edge computing markets. They are also commonly used in embedded systems such as WiFi routers and the **Internet of Things**.

ARM Cortex-M

The ARM Cortex-M is a group of 32-bit RISC ARM processor cores licensed by Arm Holdings. These cores are optimized for low-cost and energy-efficient integrated circuits, which have been embedded in tens of billions of consumer devices. Though they are most often the main component of microcontroller chips, sometimes they are embedded inside other types of chips too.

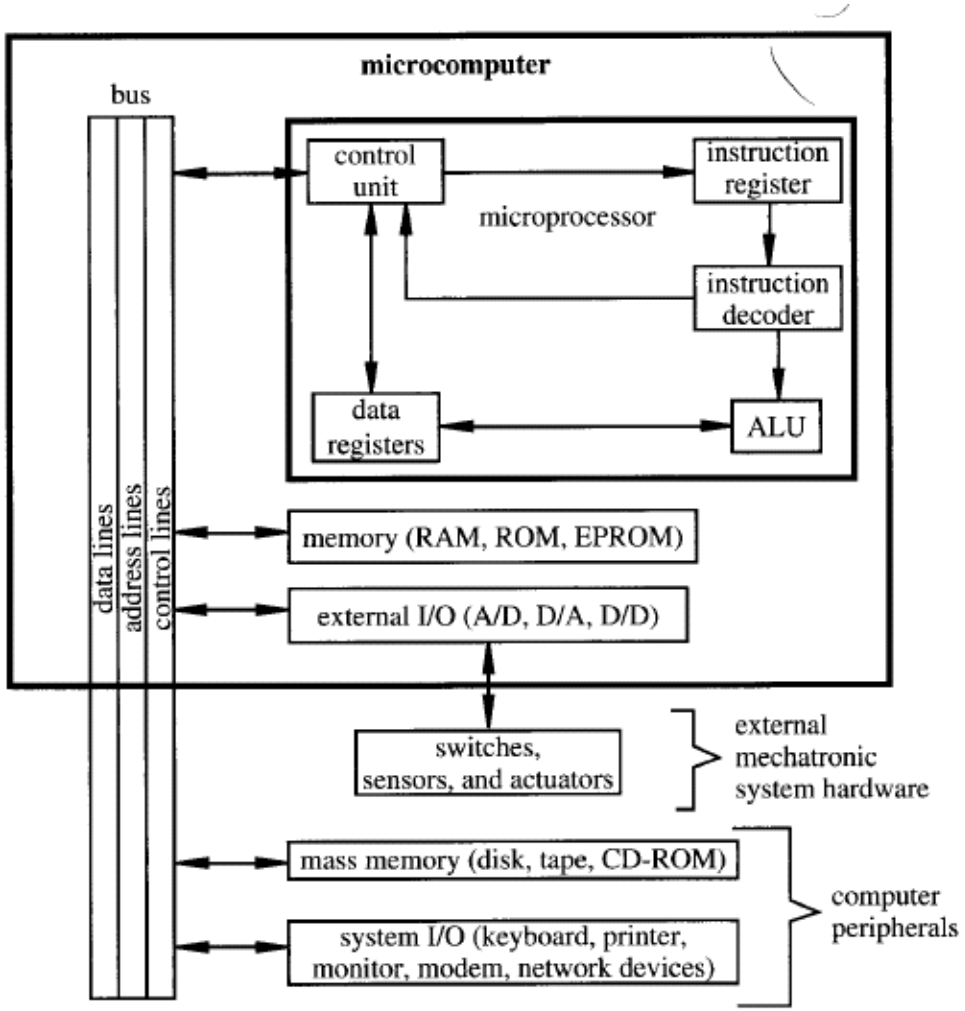
Arhitektura mikrokompjutera (mikroračunarski sistem)

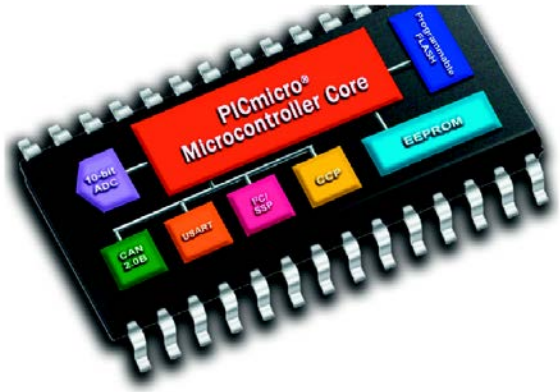


Mikrokompjuter se sastoji iz sledećih funkcionalnih celina: procesor, memorija, ulazno-izlazni interfejsi za komunikaciju sa okruženjem i magistrala.

Procesor, odnosno mikroprocesor se sastoji iz aritmetičko-logičke jedinice, polja registara u kojima se čuva korisnički program i polja registara u kojima se čuvaju radni i sistemski podaci. Upravljačka jedinica podržava rad aritmetičko-logičke jedinice tako što iz polja registra u kome se čuva korisnički program povlači instrukciju po instrukciju, prosledjuje je u sklop za dekodiranje instrukcije i povlači tražene podatke iz polja registara u kojima se čuvaju radni i sistemski podaci.

Magistrala se sastoji iz paralelnih kanala (linija) koji su **upravljačkog**, **adresnog** i **data** tipa. Preko magistrale mikrokoprocesor ostvaruje vezu sa periferijskim uredjajima (memorija, tastatura, monitor, štampač, modem, ...).

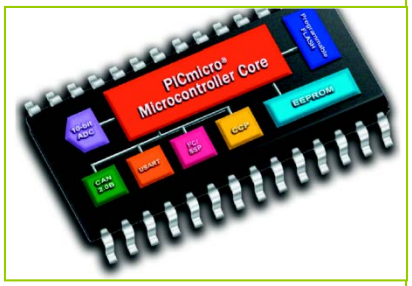




a 31

arhitektura mikrokontrolera

Mikrokontroler

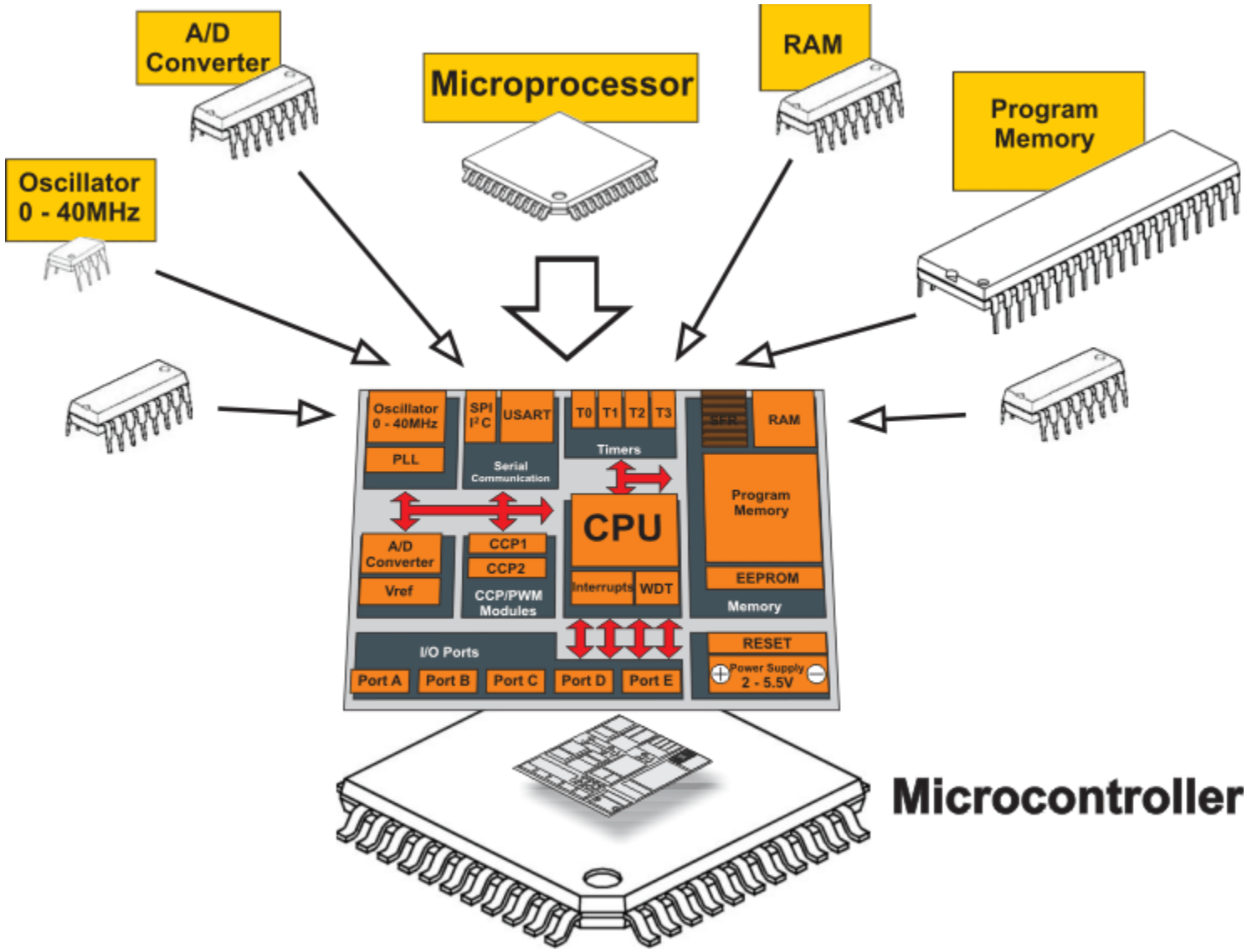


Mikrokontroler je mikroračunar integrisan u jednom čipu.

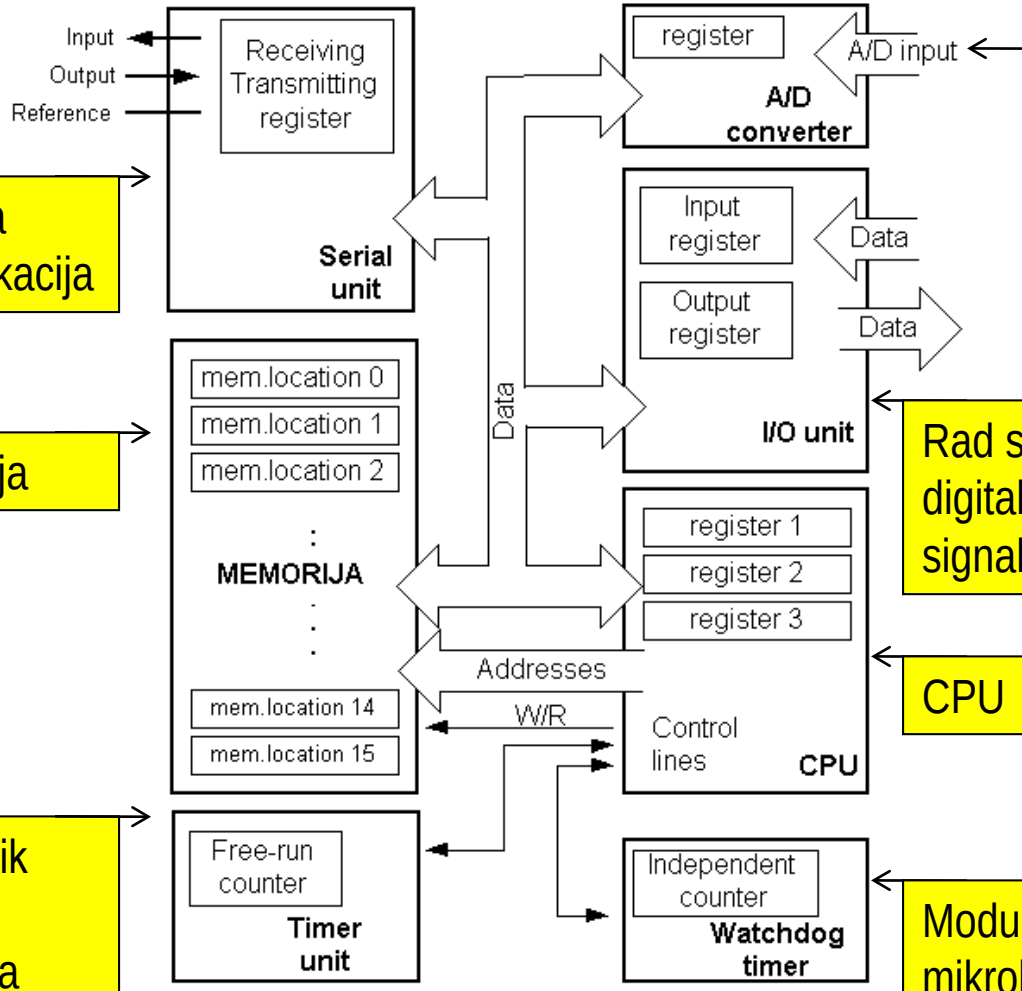
Pored mikroprocesora, ovakav čip u sebi sadrži specijalizovana kola i funkcije koje omogućavaju memorisanje podataka, komunikaciju sa drugim mikrokontrolerima, direktno povezivanje sa različitim ulazno/izlaznim uredjajima (senzori, aktuatori, HMI jedinice) i drugim specijalizovanim resursima za različite aplikacije i realizaciju mehatronskih sistema.

Osnovna odlika mikrokontrolera je **visok stepen integracije funkcija**, kompaktnost, komunikativnost (sposobnost sprezanja sa drugim mikroprocesorskim modulima) i niska cena.

Mikrokontroleri prvenstveno nalaze svoju primenu u oblasti upravljanja različitim procesima i objektima u industriji, kućnim uredjajima, vojnoj opremi i u opremi za istraživanje i razvoj. Specifične tehničke karakteristike mikrokontrolera omogućavaju njihovu neposrednu fizičku ugradnju u mehaničku strukturu sistema, što ih čini idealnom komponentom za gradnju mehatronskih sistema.


A/D Converter
Microprocessor
RAM
Program Memory
Oscillator 0 - 40MHz
Oscillator 0 - 40MHz
PLL
SPI I²C
USART
Serial Communication
T0
T1
T2
T3
Timers
SFR
RAM
Program Memory
CPU
A/D Converter
Vref
CCP1
CCP2
CCP/PWM Modules
Interrupts
WDT
EEPROM Memory
I/O Ports
Port A
Port B
Port C
Port D
Port E
RESET
Power Supply 2 - 5.5V
Microcontroller

Arhitektura tipičnog mikrokontrolera potpune konfiguracije



Serijska komunikacija

Memorija

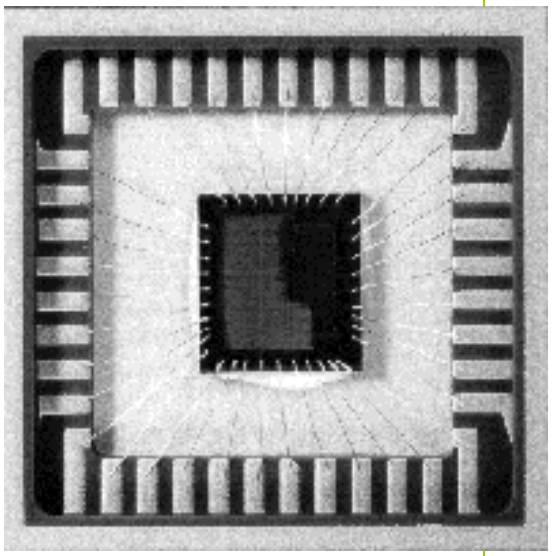
Časovnik realnog vremena

Rad sa analognim signalima

Rad sa digitalnim signalima

CPU

Modul za nadzor funkcionisanja mikrokontrolera



Mikrokontroler PIC16F84A

PIC16F84x je familija osmobitnih RISC mikrokontrolera, izradjen u CMOS tehnologiji, modifikovana Harvard arhitektura, pakovanje sa 18 pinova

(kod CPU Harvard arhitekture odvojene su magistrale za prenos podataka i prenos instrukcija korisničkog programa; kod alternativne, Von Neumann arhitekture, prenos podataka i prenos instrukcija se ostvaruje preko zajedničke magistrale).

Instrukcijska lista od **35 14-bitnih instrukcija**

PIC – *Peripheral Interface Controller*

RISC – *Reduced Instruction Set Computer*

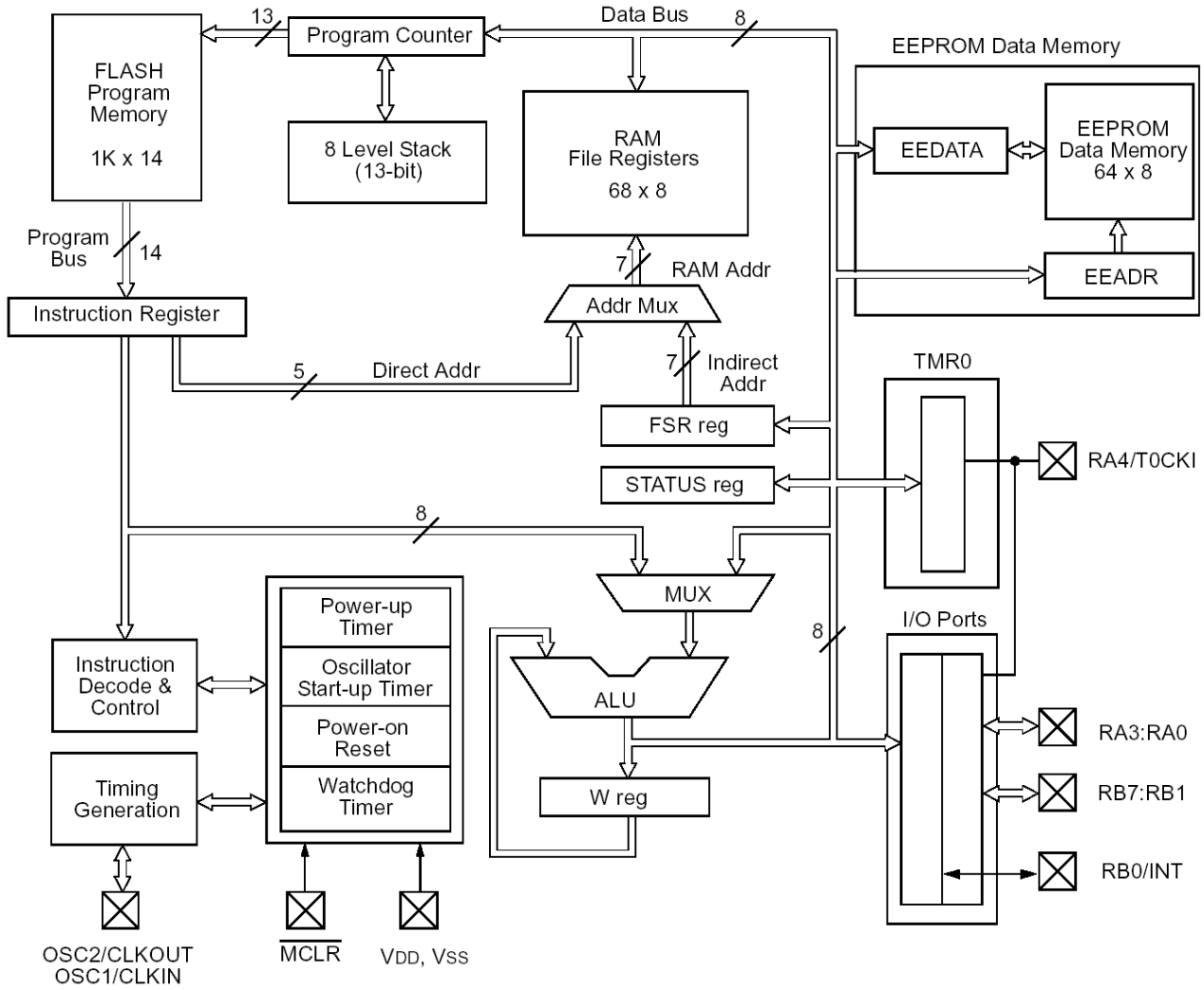
CMOS – Complemetary Metal Oxide Semiconductor

Radna frekvencija: DC - 20 MHz

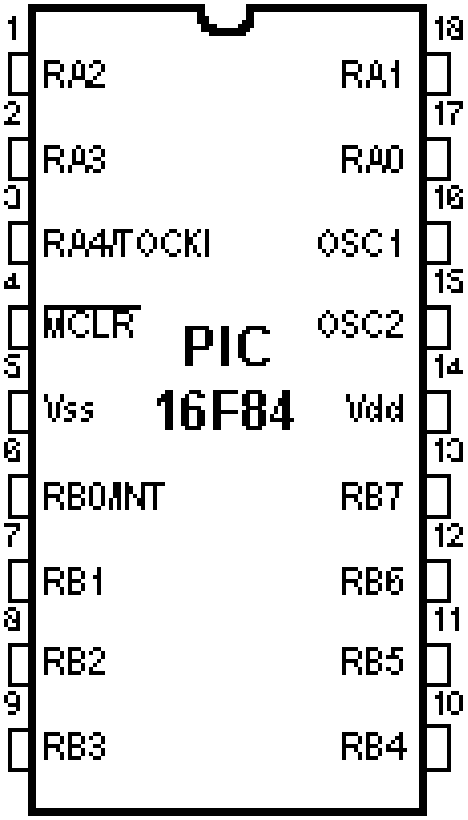
Izvršenje jedne instrukcije: DC – 200ns (maksimalno 5.000.000 instrukcija/sec)

PIC16F84A može da memoriše korisnički program dužine do 1k instrukcija.

Blok dijagram arhitekture mikrokontrolera PIC16F84A

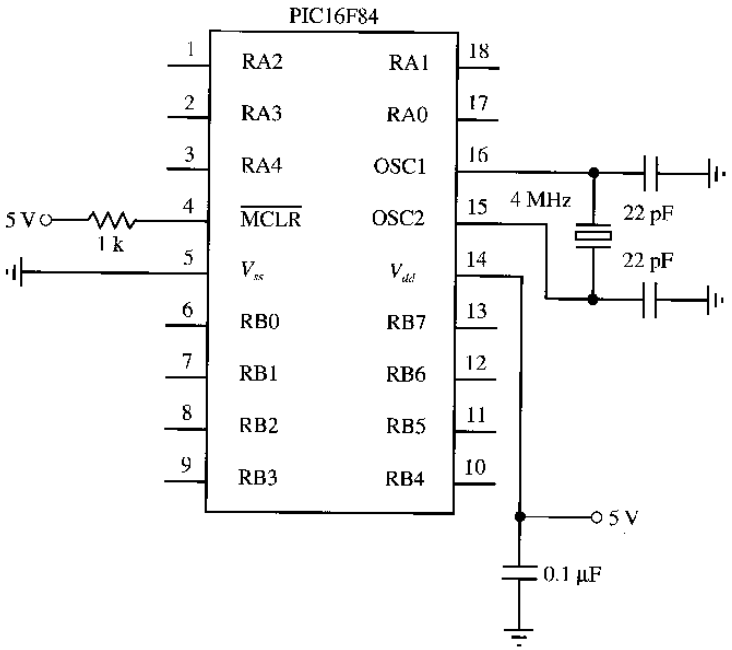


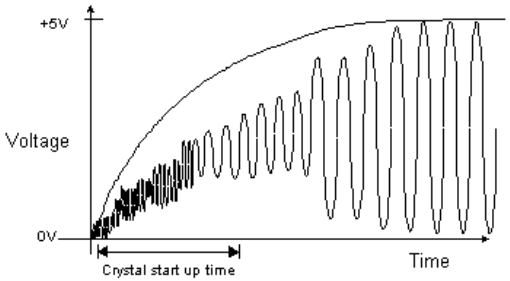
Funkcije pinova:



Pin no.1 **RA2**	Bidirekcioni A port – bit 2
Pin no.2 **RA3**	Bidirekcioni A port – bit 3
Pin no.3 **RA4/TOCK1**	Bidirekcioni A port – bit 4. Ulaz časovnika/brojača TOCK1
Pin no.4 **MCLR**	Master Clear - reset ulaz, Vpp napon programiranja mikrokontrolera
Pin no.5 **Vss**	Napajanje masa (GND).
Pin no.6 **RB0/INT**	Bidirekcioni B port – bit 0, eksterni prekid.
Pin no.7 **RB1**	Bidirekcioni B port – bit 1.
Pin no.8 **RB2**	Bidirekcioni B port – bit 2.
Pin no.9 **RB3**	Bidirekcioni B port – bit 3.
Pin no.10 **RB4**	Bidirekcioni B port – bit 4.
Pin no.11 **RB5**	Bidirekcioni B port – bit 5.
Pin no.12 **RB6**	Bidirekcioni B port – bit 6. Serijski clk za programiranje.
Pin no.13 **RB7**	Bidirekcioni B port – bit 7. Serijski ulaz podataka u modu za programiranje.
Pin no.14 **Vdd**	Napajanje + Vss.
Pin no.15 **OSC2**	Oscilator, takt generatora.
Pin no.16 **OSC1**	Oscilator, takt generatora.
Pin no.17 **RA0**	Bidirekcioni A port – bit 0.
Pin no.18 **RA1**	Bidirekcioni A port – bit 1.

Minimalna električna konfiguracija:

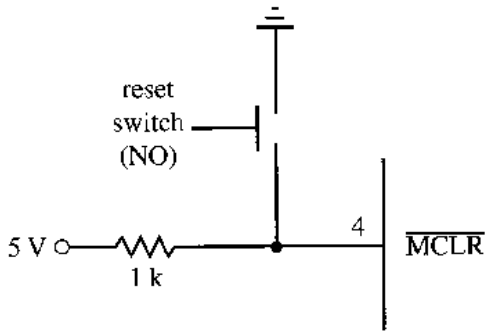




Signal of an oscillator clock after receiving the supply of a microcontroller

Vreme stabilizacije oscilatora kod uključivanja u rad mikrokontrolera. Dok se rad oscilatora ne stabilizuje kontroler treba da bude u reset stanju.

Kontroler se resetuje dovodjenjem nultog naponskog nivoa na pin 4.

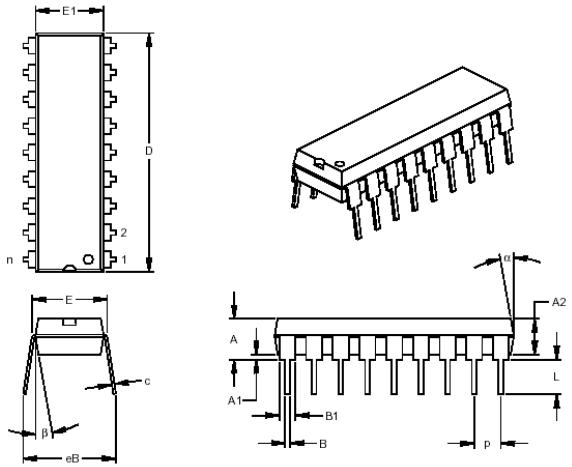


Spoljašnje električno kolo sa kvarcnim kristalom obezbedjuje precizno i stabilno generisanje vremenskog kvanta za taktovanje mikrokontrolera. Spoljašnji oscilator se priključuje na pinove 15 i 16. Da bi se sprečile smetnje, oscilator treba da bude lociran u neposrednoj blizini kontrolera. Postoje tropolni elementi – keramički rezonatori, koji u sebi sadrže kristal i ostale pasivne komponente za rad oscilatora.

Dovodjenjem 5 VDC na pinove 14 (V_{dd} = +5 V) i 5 (V_{ss} = 0 V) mikrokontroler je, posle stabilizacije oscilatora, spreman za rad. Negacija reset funkcije zahteva dovodjenje 5 VDC na pin 4.

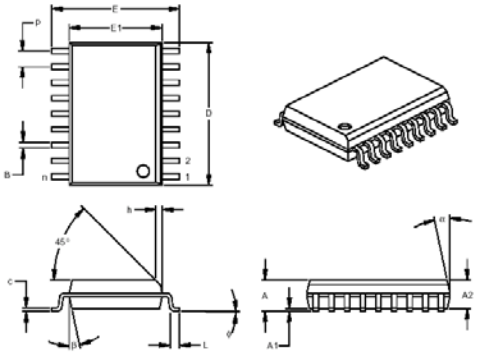
DIP pakovanje

D = 22.8mm E = 7.94mm



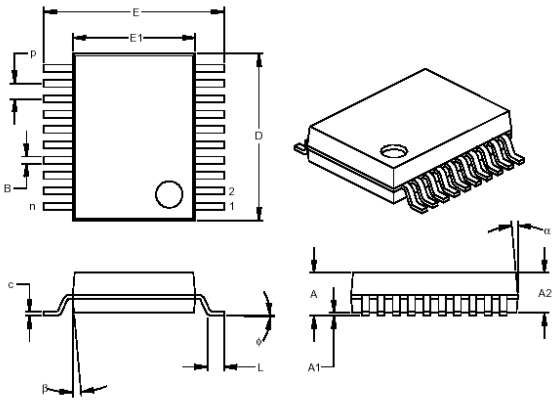
SO pakovanje (SMD tehnologija)

D = 11.53mm E = 10.34mm



SS pakovanje

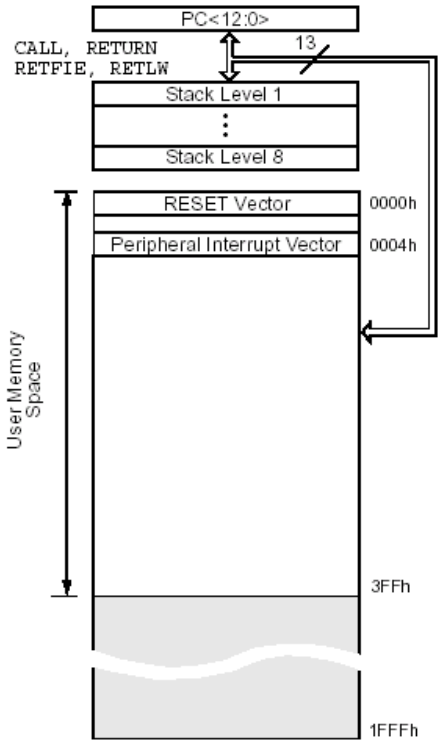
D = 7.2mm E = 7.85mm



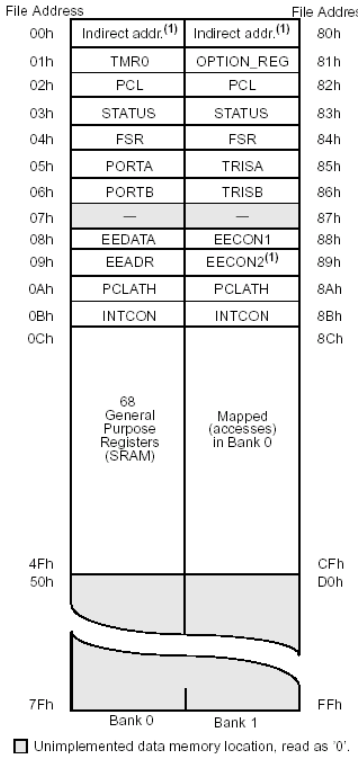
Organizacija memorije:

PIC16F84A poseduje operativnu memoriju koja se sastoji iz dva memorijska bloka: **programska memorija i memorija podataka**. Svaki blok poseduje sopstvenu magistralu, tako da se pristup ovim memorijama može da ostvari paralelno u istom taktu oscilatora (**Harward arhitektura**).

Programska memorija



Memorija podataka



Sadržaj SFR registra:

Addr	Name	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	Value on Power-on RESET	Details on page
Bank 0											
00h	INDF	Uses contents of FSR to address Data Memory (not a physical register)								---- ----	11
01h	TMR0	8-bit Real-Time Clock/Counter								xxxx xxxx	20
02h	PCL	Low Order 8 bits of the Program Counter (PC)								0000 0000	11
03h	STATUS(2)	IRP	RP1	RP0	$\overline{TO}$	$\overline{PD}$	Z	DC	C	0001 1xxx	8
04h	FSR	Indirect Data Memory Address Pointer 0								xxxx xxxx	11
05h	PORTA(4)	—	—	—	RA4/T0CKI	RA3	RA2	RA1	RA0	---x xxxx	16
06h	PORTB(5)	RB7	RB6	RB5	RB4	RB3	RB2	RB1	RB0/INT	xxxx xxxx	18
07h	—	Unimplemented location, read as '0'								—	—
08h	EEDATA	EEPROM Data Register								xxxx xxxx	13,14
09h	EEADR	EEPROM Address Register								xxxx xxxx	13,14
0Ah	PCLATH	—	—	—	Write Buffer for upper 5 bits of the PC(1)					---0 0000	11
0Bh	INTCON	GIE	EEIE	T0IE	INTE	RBIE	T0IF	INTF	RBIF	0000 000x	10
Bank 1											
80h	INDF	Uses Contents of FSR to address Data Memory (not a physical register)								---- ----	11
81h	OPTION_REG	RBPU	INTEDG	T0CS	T0SE	PSA	PS2	PS1	PS0	1111 1111	9
82h	PCL	Low order 8 bits of Program Counter (PC)								0000 0000	11
83h	STATUS (2)	IRP	RP1	RP0	$\overline{TO}$	$\overline{PD}$	Z	DC	C	0001 1xxx	8
84h	FSR	Indirect data memory address pointer 0								xxxx xxxx	11
85h	TRISA	—	—	—	PORTA Data Direction Register					---1 1111	16
86h	TRISB	PORTB Data Direction Register								1111 1111	18
87h	—	Unimplemented location, read as '0'								—	—
88h	EECON1	—	—	—	EEIF	WRERR	WREN	WR	RD	---0 x000	13
89h	EECON2	EEPROM Control Register 2 (not a physical register)								---- ----	14
0Ah	PCLATH	—	—	—	Write buffer for upper 5 bits of the PC(1)					---0 0000	11
0Bh	INTCON	GIE	EEIE	T0IE	INTE	RBIE	T0IF	INTF	RBIF	0000 000x	10

Legend: x = unknown, u = unchanged, - = unimplemented, read as '0', q = value depends on condition

Note 1: The upper byte of the program counter is not directly accessible. PCLATH is a slave register for PC<12:8>. The contents of PCLATH can be transferred to the upper byte of the program counter, but the contents of PC<12:8> are never transferred to PCLATH.

2: The $\overline{TO}$ and $\overline{PD}$ status bits in the STATUS register are not affected by a $\overline{MCLR}$ Reset.

3: Other (non power-up) RESETS include: external RESET through $\overline{MCLR}$ and the Watchdog Timer Reset.

4: On any device RESET, these pins are configured as inputs.

5: This is the value that will be in the port output latch.

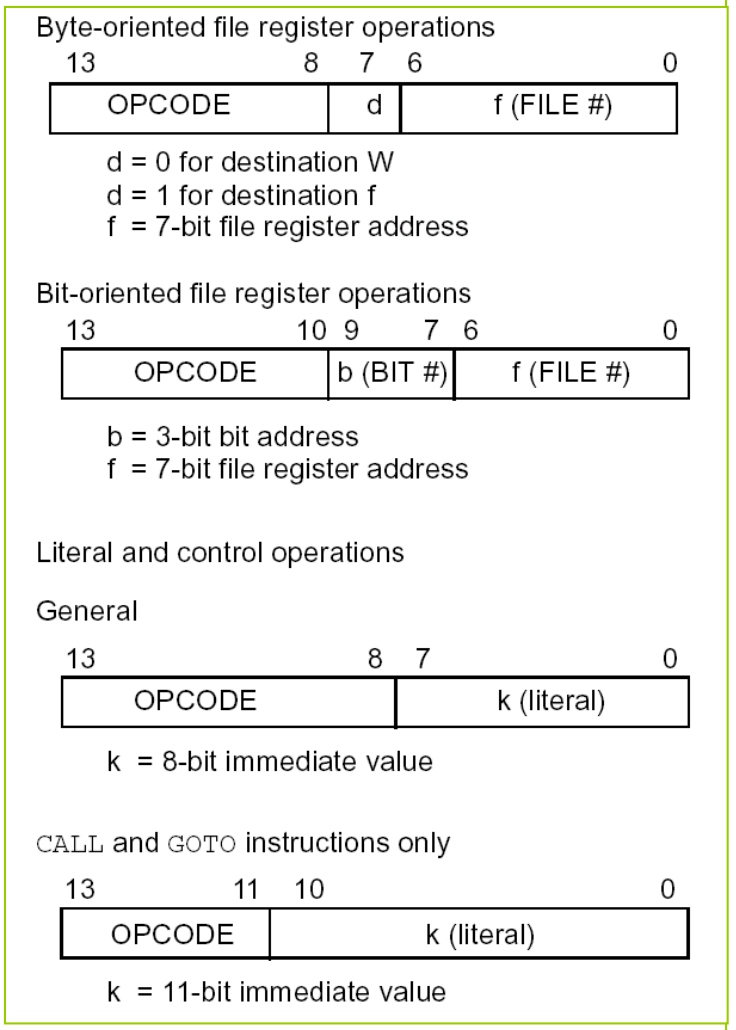
Lista instrukcija

PIC16F84 je mikrokontroler baziran na RISC tehnologiji (*Reduced Instruction Set Computer*). RISC tehnologija omogućava kompresiju programa 2:1 i ubrzavanje rada za 4:1 u odnosu na ekvivalentne CISC (*Complex Instruction Set Computer*) mikrokontrolere.

PIC16F84 poseduje 35 instrukcija. Instrukcije se klasifikuju u tri grupe:

1. bajt orijentisane,
2. bit orijentisane i
3. literal/kontrolne instrukcije.

Instrukcija je kodirana 14 bitnom reči i njen format uključuje OPCODE polje (binarni kod operacije) i adresna polja u kojima se naznačuje destinacija i adresa operanda, ili direktna vrednost operanda (kombinovan dvoadresni i nulaadresni format).



```
bsf     PORTA,0   ;setovanje RA0
decfzs  brojac, 1 ;dekrementiranje brojaca
goto    petlja    ;povratak na pocetak glavne
                  ;petlje ako brojac > 0
```

a 32

lista instrukcija i programiranje u asembleru

Lista instrukcija PIC16F84 mikrokontrolera

18 Byte orijentisanih instrukcija

4 Bit orijetnisane instrukcije

13 kontrolnih i memorijski orijentisanih instrukcija

Mnemonic, Operands		Description	Cycles	14-Bit Opcode MSb			LSb	Status Affected	Notes
BYTE-ORIENTED FILE REGISTER OPERATIONS									
ADDWF	f, d	Add W and f	1	00	0111	dfff	ffff	C,DC,Z	1,2
ANDWF	f, d	AND W with f	1	00	0101	dfff	ffff	Z	1,2
CLRF	f	Clear f	1	00	0001	lfff	ffff	Z	2
CLRW	-	Clear W	1	00	0001	0xxx	xxxx	Z	
COMF	f, d	Complement f	1	00	1001	dfff	ffff	Z	1,2
DECF	f, d	Decrement f	1	00	0011	dfff	ffff	Z	1,2
DECFSZ	f, d	Decrement f, Skip if 0	1 (2)	00	1011	dfff	ffff		1,2,3
INCF	f, d	Increment f	1	00	1010	dfff	ffff	Z	1,2
INCFSZ	f, d	Increment f, Skip if 0	1 (2)	00	1111	dfff	ffff		1,2,3
IORWF	f, d	Inclusive OR W with f	1	00	0100	dfff	ffff	Z	1,2
MOVF	f, d	Move f	1	00	1000	dfff	ffff	Z	1,2
MOVWF	f	Move W to f	1	00	0000	lfff	ffff		
NOP	-	No Operation	1	00	0000	0xx0	0000		
RLF	f, d	Rotate Left f through Carry	1	00	1101	dfff	ffff	C	1,2
RRF	f, d	Rotate Right f through Carry	1	00	1100	dfff	ffff	C	1,2
SUBWF	f, d	Subtract W from f	1	00	0010	dfff	ffff	C,DC,Z	1,2
SWAPF	f, d	Swap nibbles in f	1	00	1110	dfff	ffff		1,2
XORWF	f, d	Exclusive OR W with f	1	00	0110	dfff	ffff	Z	1,2
BIT-ORIENTED FILE REGISTER OPERATIONS									
BCF	f, b	Bit Clear f	1	01	00bb	bfff	ffff		1,2
BSF	f, b	Bit Set f	1	01	01bb	bfff	ffff		1,2
BTFSC	f, b	Bit Test f, Skip if Clear	1 (2)	01	10bb	bfff	ffff		3
BTFSS	f, b	Bit Test f, Skip if Set	1 (2)	01	11bb	bfff	ffff		3
LITERAL AND CONTROL OPERATIONS									
ADDLW	k	Add literal and W	1	11	111x	kkkk	kkkk	C,DC,Z	
ANDLW	k	AND literal with W	1	11	1001	kkkk	kkkk	Z	
CALL	k	Call subroutine	2	10	0kkk	kkkk	kkkk		
CLRWDT	-	Clear Watchdog Timer	1	00	0000	0110	0100	$\overline{TO}$,$\overline{PD}$	
GOTO	k	Go to address	2	10	1kkk	kkkk	kkkk		
IORLW	k	Inclusive OR literal with W	1	11	1000	kkkk	kkkk	Z	
MOVLW	k	Move literal to W	1	11	00xx	kkkk	kkkk		
RETFIE	-	Return from interrupt	2	00	0000	0000	1001		
RETLW	k	Return with literal in W	2	11	01xx	kkkk	kkkk		
RETURN	-	Return from Subroutine	2	00	0000	0000	1000		
SLEEP	-	Go into standby mode	1	00	0000	0110	0011	$\overline{TO}$,$\overline{PD}$	
SUBLW	k	Subtract W from literal	1	11	110x	kkkk	kkkk	C,DC,Z	
XORLW	k	Exclusive OR literal with W	1	11	1010	kkkk	kkkk	Z	

ADDLW — Add Literal and W

Syntax: [*label*] ADDLW k

Operands: $0 \le k \le 255$

Operation: (W) + k → W

Status Affected: C, DC, Z

Encoding:

11	111x	kkkk	kkkk

Description: The contents of the W register are added to the eight bit literal 'k' and the result is placed in the W register.

Words: 1

Cycles: 1

Q Cycle Activity:

Q1	Q2	Q3	Q4
Decode	Read literal 'k'	Process data	Write to W register

Example1

```
ADDLW   0x15
```

Before Instruction
W = 0x10
After Instruction
W = 0x25

Example 2

```
ADDLW   MYREG
```

Before Instruction
W = 0x10
Address of MYREG † = 0x37
† MYREG is a symbol for a data memory location
After Instruction
W = 0x47

Example 3

```
ADDLW   HIGH (LU_TABLE)
```

Before Instruction
W = 0x10
Address of LU_TABLE † = 0x9375
† LU_TABLE is a label for an address in program memory
After Instruction
W = 0xA3

Example 4

```
ADDLW   MYREG
```

Before Instruction
W = 0x10
Address of PCL † = 0x02
† PCL is the symbol for the Program Counter low byte location
After Instruction
W = 0x12

ADDWF — Add W and f

Syntax: [*label*] ADDWF f,d

Operands: $0 \le f \le 127$
$d \in [0,1]$

Operation: (W) + (f) → destination

Status Affected: C, DC, Z

Encoding:

00	0111	dfff	ffff

Description: Add the contents of the W register with register 'f'. If 'd' is 0 the result is stored in the W register. If 'd' is 1 the result is stored back in register 'f'.

Words: 1

Cycles: 1

Q Cycle Activity:

Q1	Q2	Q3	Q4
Decode	Read register 'f'	Process data	Write to destination

Example 1

```
ADDWF   FSR, 0
```

Before Instruction
W = 0x17
FSR = 0xC2
After Instruction
W = 0xD9
FSR = 0xC2

Example 2

```
ADDWF   INDF, 1
```

Before Instruction
W = 0x17
FSR = 0xC2
Contents of Address (FSR) = 0x20
After Instruction
W = 0x17
FSR = 0xC2
Contents of Address (FSR) = 0x37

Example 3

```
ADDWF   PCL
```

Case 1:
Before Instruction
W = 0x10
PCL = 0x37
C = x
After Instruction
PCL = 0x47
C = 0

Case 2:
Before Instruction
W = 0x10
PCL = 0xF7
PCH = 0x08
C = x
After Instruction
PCL = 0x07
PCH = 0x08
C = 1

ANDLW And Literal with W

Syntax: [*label*] ANDLW k

Operands: $0 \le k \le 255$

Operation: (W).AND. (k) → W

Status Affected: Z

Encoding:

11	1001	kkkk	kkkk

Description: The contents of W register are AND'ed with the eight bit literal 'k'. The result is placed in the W register.

Words: 1

Cycles: 1

Q Cycle Activity:

Q1	Q2	Q3	Q4
Decode	Read literal 'k'	Process data	Write to W register

Example 1

```
ANDLW   0x5F
Before Instruction                  ; 0101 1111  (0x5F)
        W  =  0xA3                  ; 1010 0011  (0xA3)
After Instruction                   ;----------  ------
        W  =  0x03                  ; 0000 0011  (0x03)
```

Example 2

```
ANDLW   MYREG
Before Instruction
        W  =  0xA3
        Address of MYREG † = 0x37
        † MYREG is a symbol for a data memory location
After Instruction
        W  =  0x23
```

Example 3

```
ANDLW   HIGH (LU_TABLE)
Before Instruction
        W  =  0xA3
        Address of LU_TABLE † = 0x9375
        † LU_TABLE is a label for an address in program memory
After Instruction
        W  =  0x83
```

ANDWF AND W with f

Syntax: [*label*] ANDWF f,d

Operands: $0 \le f \le 127$
$d \in [0,1]$

Operation: (W).AND. (f) → destination

Status Affected: Z

Encoding:

00	0101	dfff	ffff

Description: AND the W register with register 'f'. If 'd' is 0 the result is stored in the W register. If 'd' is 1 the result is stored back in register 'f'.

Words: 1

Cycles: 1

Q Cycle Activity:

Q1	Q2	Q3	Q4
Decode	Read register 'f'	Process data	Write to destination

Example 1

```
ANDWF   FSR, 1
Before Instruction                  ; 0001 0111  (0x17)
        W  =  0x17                  ; 1100 0010  (0xC2)
        FSR =  0xC2                 ;----------  ------
After Instruction                   ; 0000 0010  (0x02)
        W  =  0x17
        FSR =  0x02
```

Example 2

```
ANDWF   FSR, 0
Before Instruction                  ; 0001 0111  (0x17)
        W  =  0x17                  ; 1100 0010  (0xC2)
        FSR =  0xC2                 ;----------  ------
After Instruction                   ; 0000 0010  (0x02)
        W  =  0x02
        FSR =  0xC2
```

Example 3

```
ANDWF   INDF, 1
Before Instruction
        W  =  0x17
        FSR =  0xC2
        Contents of Address (FSR) = 0x5A
After Instruction
        W  =  0x17
        FSR =  0xC2
        Contents of Address (FSR) = 0x15
```

BCF — Bit Clear f

Syntax: [*label*] BCF f,b

Operands: $0 \le f \le 127$
$0 \le b \le 7$

Operation: $0 \rightarrow f<b>$

Status Affected: None

Encoding:

01	00bb	bfff	ffff

Description: Bit 'b' in register 'f' is cleared.

Words: 1

Cycles: 1

Q Cycle Activity:

Q1	Q2	Q3	Q4
Decode	Read register 'f'	Process data	Write register 'f'

Example 1

```
BCF     FLAG_REG, 7
```

Before Instruction
FLAG_REG = 0xC7 ; 1100 0111
After Instruction
FLAG_REG = 0x47 ; 0100 0111

Example 2

```
BCF     INDF, 3
```

Before Instruction
W = 0x17
FSR = 0xC2
Contents of Address (FSR) = 0x2F
After Instruction
W = 0x17
FSR = 0xC2
Contents of Address (FSR) = 0x27

BSF — Bit Set f

Syntax: [*label*] BSF f,b

Operands: $0 \le f \le 127$
$0 \le b \le 7$

Operation: $1 \rightarrow f<b>$

Status Affected: None

Encoding:

01	01bb	bfff	ffff

Description: Bit 'b' in register 'f' is set.

Words: 1

Cycles: 1

Q Cycle Activity:

Q1	Q2	Q3	Q4
Decode	Read register 'f'	Process data	Write register 'f'

Example 1

```
BSF     FLAG_REG, 7
```

Before Instruction
FLAG_REG =0x0A ; 0000 1010
After Instruction
FLAG_REG =0x8A ; 1000 1010

Example 2

```
BSF     INDF, 3
```

Before Instruction
W = 0x17
FSR = 0xC2
Contents of Address (FSR) = 0x20
After Instruction
W = 0x17
FSR = 0xC2
Contents of Address (FSR) = 0x28

BTFSC — Bit Test, Skip if Clear

Syntax: [*label*] BTFSC f,b

Operands: $0 \le f \le 127$
$0 \le b \le 7$

Operation: skip if (f<b>) = 0

Status Affected: None

Encoding:

01	10bb	bfff	ffff

Description: If bit 'b' in register 'f' is '0' then the next instruction is skipped.
If bit 'b' is '0' then the next instruction (fetched during the current instruction execution) is discarded, and a NOP is executed instead, making this a 2 cycle instruction.

Words: 1

Cycles: 1(2)

Q Cycle Activity:

Q1	Q2	Q3	Q4
Decode	Read register 'f'	Process data	No operation

If skip (2nd cycle):

Q1	Q2	Q3	Q4
No operation	No operation	No operation	No operation

Example 1

```
HERE    BTFSC   FLAG, 4
FALSE   GOTO    PROCESS_CODE
TRUE    •
        •
        •
```

Case 1: Before Instruction
PC = address HERE
FLAG= xxx0 xxxx
After Instruction
Since FLAG<4>= 0,
PC = address TRUE

Case 2: Before Instruction
PC = address HERE
FLAG= xxx1 xxxx
After Instruction
Since FLAG<4>=1,
PC = address FALSE

BTFSS — Bit Test f, Skip if Set

Syntax: [*label*] BTFSS f,b

Operands: $0 \le f \le 127$
$0 \le b < 7$

Operation: skip if (f<b>) = 1

Status Affected: None

Encoding:

01	11bb	bfff	ffff

Description: If bit 'b' in register 'f' is '1' then the next instruction is skipped.
If bit 'b' is '1', then the next instruction (fetched during the current instruction execution) is discarded and a NOP is executed instead, making this a 2 cycle instruction.

Words: 1

Cycles: 1(2)

Q Cycle Activity:

Q1	Q2	Q3	Q4
Decode	Read register 'f'	Process data	No operation

If skip (2nd cycle):

Q1	Q2	Q3	Q4
No operation	No operation	No operation	No operation

Example 1

```
HERE    BTFSS   FLAG, 4
FALSE   GOTO    PROCESS_CODE
TRUE    •
        •
        •
```

Case 1: Before Instruction
PC = address HERE
FLAG= xxx0 xxxx
After Instruction
Since FLAG<4>= 0,
PC = address FALSE

Case 2: Before Instruction
PC = address HERE
FLAG= xxx1 xxxx
After Instruction
Since FLAG<4>=1,
PC = address TRUE

CALL — Call Subroutine

Syntax: [*label*] CALL k

Operands: $0 \le k \le 2047$

Operation: (PC)+ 1→ TOS,
k → PC<10:0>,
(PCLATH<4:3>) → PC<12:11>

Status Affected: None

Encoding:

10	0kkk	kkkk	kkkk

Description: Call Subroutine. First, the 13-bit return address (PC+1) is pushed onto the stack. The eleven bit immediate address is loaded into PC bits <10:0>. The upper bits of the PC are loaded from PCLATH<4:3>. CALL is a two cycle instruction.

Words: 1

Cycles: 2

Q Cycle Activity:

1st cycle:

Q1	Q2	Q3	Q4
Decode	Read literal 'k'	Process data	No operation

2nd cycle:

Q1	Q2	Q3	Q4
No operation	No operation	No operation	No operation

Example 1

```
HERE    CALL   THERE
```

Before Instruction
PC = Address HERE
After Instruction
TOS = Address HERE+1
PC = Address THERE

CLRF — Clear f

Syntax: [*label*] CLRF f

Operands: $0 \le f \le 127$

Operation: 00h → f
1 → Z

Status Affected: Z

Encoding:

00	0001	lfff	ffff

Description: The contents of register 'f' are cleared and the Z bit is set.

Words: 1

Cycles: 1

Q Cycle Activity:

Q1	Q2	Q3	Q4
Decode	Read register 'f'	Process data	Write register 'f'

Example 1

```
CLRF    FLAG_REG
```

Before Instruction
FLAG_REG=0x5A
After Instruction
FLAG_REG=0x00
Z = 1

Example 2

```
CLRF    INDF
```

Before Instruction
FSR = 0xC2
Contents of Address (FSR)=0xAA
After Instruction
FSR = 0xC2
Contents of Address (FSR)=0x00
Z = 1

CLRW — Clear W

Syntax: [*label*] CLRW

Operands: None

Operation: 00h → W
1 → Z

Status Affected: Z

Encoding:

00	0001	0xxx	xxxx

Description: W register is cleared. Zero bit (Z) is set.

Words: 1

Cycles: 1

Q Cycle Activity:

Q1	Q2	Q3	Q4
Decode	Read register 'f'	Process data	Write register 'W'

Example 1 `CLRW`

Before Instruction
W = 0x5A
After Instruction
W = 0x00
Z = 1

CLRWDT — Clear Watchdog Timer

Syntax: [*label*] CLRWDT

Operands: None

Operation: 00h → WDT
0 → WDT prescaler count,
1 → $\overline{TO}$
1 → $\overline{PD}$

Status Affected: $\overline{TO}$, $\overline{PD}$

Encoding:

00	0000	0110	0100

Description: `CLRWDT` instruction clears the Watchdog Timer. It also clears the prescaler count of the WDT. Status bits $\overline{TO}$ and $\overline{PD}$ are set.

Words: 1

Cycles: 1

Q Cycle Activity:

Q1	Q2	Q3	Q4
Decode	No operation	Process data	Clear WDT Counter

Example 1 `CLRWDT`

Before Instruction
WDT counter= x
WDT prescaler =1:128
After Instruction
WDT counter=0x00
WDT prescaler count=0
$\overline{TO}$ = 1
$\overline{PD}$ = 1
WDT prescaler =1:128

Note: The `CLRWDT` instruction does not affect the assignment of the WDT prescaler.

COMF — Complement f

Syntax: [*label*] COMF f,d

Operands: $0 \le f \le 127$
$d \in [0,1]$

Operation: $(\bar{f}) \rightarrow$ destination

Status Affected: Z

Encoding:

00	1001	dfff	ffff

Description: The contents of register 'f' are 1's complemented. If 'd' is 0 the result is stored in W. If 'd' is 1 the result is stored back in register 'f'.

Words: 1

Cycles: 1

Q Cycle Activity:

Q1	Q2	Q3	Q4
Decode	Read register 'f'	Process data	Write to destination

Example 1

```
COMF    REG1, 0
```

Before Instruction
REG1= 0x13
After Instruction
REG1= 0x13
W = 0xEC

Example 2

```
COMF    INDF, 1
```

Before Instruction
FSR = 0xC2
Contents of Address (FSR)=0xAA
After Instruction
FSR = 0xC2
Contents of Address (FSR)=0x55

Example 3

```
COMF    REG1, 1
```

Before Instruction
REG1= 0xFF
After Instruction
REG1= 0x00
Z = 1

DECF — Decrement f

Syntax: [*label*] DECF f,d

Operands: $0 \le f \le 127$
$d \in [0,1]$

Operation: (f) - 1 → destination

Status Affected: Z

Encoding:

00	0011	dfff	ffff

Description: Decrement register 'f'. If 'd' is 0 the result is stored in the W register. If 'd' is 1 the result is stored back in register 'f'.

Words: 1

Cycles: 1

Q Cycle Activity:

Q1	Q2	Q3	Q4
Decode	Read register 'f'	Process data	Write to destination

Example 1

```
DECF    CNT, 1
```

Before Instruction
CNT = 0x01
Z = 0
After Instruction
CNT = 0x00
Z = 1

Example 2

```
DECF    INDF, 1
```

Before Instruction
FSR = 0xC2
Contents of Address (FSR) = 0x01
Z = 0
After Instruction
FSR = 0xC2
Contents of Address (FSR) = 0x00
Z = 1

Example 3

```
DECF    CNT, 0
```

Before Instruction
CNT = 0x10
W = x
Z = 0
After Instruction
CNT = 0x10
W = 0x0F
Z = 0

GOTO — Unconditional Branch

Syntax: [*label*] GOTO k

Operands: $0 \le k \le 2047$

Operation: $k \rightarrow PC<10:0>$
$PCLATH<4:3> \rightarrow PC<12:11>$

Status Affected: None

Encoding:

10	1kkk	kkkk	kkkk

Description: GOTO is an unconditional branch. The eleven bit immediate value is loaded into PC bits <10:0>. The upper bits of PC are loaded from PCLATH<4:3>. GOTO is a two cycle instruction.

Words: 1

Cycles: 2

Q Cycle Activity:

1st cycle:

Q1	Q2	Q3	Q4
Decode	Read literal 'k'<7:0>	Process data	No operation

2nd cycle:

Q1	Q2	Q3	Q4
No operation	No operation	No operation	No operation

Example

```
GOTO THERE
```

After Instruction
PC =Address THERE

INCF — Increment f

Syntax: [*label*] INCF f,d

Operands: $0 \le f \le 127$
$d \in [0,1]$

Operation: (f) + 1 → destination

Status Affected: Z

Encoding:

00	1010	dfff	ffff

Description: The contents of register 'f' are incremented. If 'd' is 0 the result is placed in the W register. If 'd' is 1 the result is placed back in register 'f'.

Words: 1

Cycles: 1

Q Cycle Activity:

Q1	Q2	Q3	Q4
Decode	Read register 'f'	Process data	Write to destination

Example 1

```
INCF    CNT, 1
```

Before Instruction
CNT = 0xFF
Z = 0
After Instruction
CNT = 0x00
Z = 1

Example 2

```
INCF    INDF, 1
```

Before Instruction
FSR = 0xC2
Contents of Address (FSR) = 0xFF
Z = 0
After Instruction
FSR = 0xC2
Contents of Address (FSR) = 0x00
Z = 1

Example 3

```
INCF    CNT, 0
```

Before Instruction
CNT = 0x10
W = x
Z = 0
After Instruction
CNT = 0x10
W = 0x11
Z = 0

INCFSZ — Increment f, Skip if 0

Syntax: [*label*] INCFSZ f,d

Operands: $0 \le f \le 127$
$d \in [0,1]$

Operation: (f) + 1 → destination, skip if result = 0

Status Affected: None

Encoding:

00	1111	dfff	ffff

Description: The contents of register 'f' are incremented. If 'd' is 0 the result is placed in the W register. If 'd' is 1 the result is placed back in register 'f'.
If the result is 0, then the next instruction (fetched during the current instruction execution) is discarded and a NOP is executed instead, making this a 2 cycle instruction.

Words: 1

Cycles: 1(2)

Q Cycle Activity:

Q1	Q2	Q3	Q4
Decode	Read register 'f'	Process data	Write to destination

If skip (2nd cycle):

Q1	Q2	Q3	Q4
No operation	No operation	No operation	No operation

Example

```
HERE      INCFSZ   CNT, 1
          GOTO     LOOP
CONTINUE  •
          •
          •
```

Case 1: Before Instruction
PC = address HERE
CNT = 0xFF
After Instruction
CNT = 0x00
PC = address CONTINUE

Case 2: Before Instruction
PC = address HERE
CNT = 0x00
After Instruction
CNT = 0x01
PC = address HERE + 1

IORWF — Inclusive OR W with f

Syntax: [*label*] IORWF f,d

Operands: $0 \le f \le 127$
$d \in [0,1]$

Operation: (W).OR. (f) → destination

Status Affected: Z

Encoding:

00	0100	dfff	ffff

Description: Inclusive OR the W register with register 'f'. If 'd' is 0 the result is placed in the W register. If 'd' is 1 the result is placed back in register 'f'.

Words: 1

Cycles: 1

Q Cycle Activity:

Q1	Q2	Q3	Q4
Decode	Read register 'f'	Process data	Write to destination

Example 1

```
IORWF   RESULT, 0
```

Before Instruction
RESULT=0x13
W = 0x91
After Instruction
RESULT=0x13
W = 0x93
Z = 0

Example 2

```
IORWF   INDF, 1
```

Before Instruction
W = 0x17
FSR = 0xC2
Contents of Address (FSR) = 0x30
After Instruction
W = 0x17
FSR = 0xC2
Contents of Address (FSR) = 0x37
Z = 0

Example 3

```
IORWF   RESULT, 1
```

Case 1:
Before Instruction
RESULT=0x13
W = 0x91
After Instruction
RESULT=0x93
W = 0x91
Z = 0

Case 2:
Before Instruction
RESULT=0x00
W = 0x00
After Instruction
RESULT=0x00
W = 0x00
Z = 1

MOVLW — Move Literal to W

Syntax: [*label*] MOVLW k

Operands: $0 \le k \le 255$

Operation: k → W

Status Affected: None

Encoding:

11	00xx	kkkk	kkkk

Description: The eight bit literal 'k' is loaded into W register. The don't cares will assemble as 0's.

Words: 1

Cycles: 1

Q Cycle Activity:

Q1	Q2	Q3	Q4
Decode	Read literal 'k'	Process data	Write to W register

Example 1

```
MOVLW   0x5A
```

After Instruction
W = 0x5A

Example 2

```
MOVLW   MYREG
```

Before Instruction
W = 0x10
Address of MYREG † = 0x37
† MYREG is a symbol for a data memory location
After Instruction
W = 0x37

Example 3

```
MOVLW   HIGH (LU_TABLE)
```

Before Instruction
W = 0x10
Address of LU_TABLE † = 0x9375
† LU_TABLE is a label for an address in program memory
After Instruction
W = 0x93

MOVF — Move f

Syntax: [*label*] MOVF f,d

Operands: $0 \le f \le 127$
$d \in [0,1]$

Operation: (f) → destination

Status Affected: Z

Encoding:

00	1000	dfff	ffff

Description: The contents of register 'f' is moved to a destination dependent upon the status of 'd'. If 'd' = 0, destination is W register. If 'd' = 1, the destination is file register 'f' itself. 'd' = 1 is useful to test a file register since status flag Z is affected.

Words: 1

Cycles: 1

Q Cycle Activity:

Q1	Q2	Q3	Q4
Decode	Read register 'f'	Process data	Write to destination

Example 1

```
MOVF    FSR, 0
```

Before Instruction
W = 0x00
FSR = 0xC2
After Instruction
W = 0xC2
Z = 0

Example 2

```
MOVF    INDF, 0
```

Before Instruction
W = 0x17
FSR = 0xC2
Contents of Address (FSR) = 0x00
After Instruction
W = 0x17
FSR = 0xC2
Contents of Address (FSR) = 0x00
Z = 1

Example 3

```
MOVF    FSR, 1
```

Case 1:
Before Instruction
FSR = 0x43
After Instruction
FSR = 0x43
Z = 0

Case 2:
Before Instruction
FSR = 0x00
After Instruction
FSR = 0x00
Z = 1

MOVWF — Move W to f

Syntax: [*label*] MOVWF f

Operands: $0 \le f \le 127$

Operation: (W) → f

Status Affected: None

Encoding:

00	0000	lfff	ffff

Description: Move data from W register to register 'f'.

Words: 1

Cycles: 1

Q Cycle Activity:

Q1	Q2	Q3	Q4
Decode	Read register 'f'	Process data	Write register 'f'

Example 1

```
MOVWF   OPTION_REG
```

Before Instruction
OPTION_REG=0xFF
W = 0x4F
After Instruction
OPTION_REG=0x4F
W = 0x4F

Example 2

```
MOVWF   INDF
```

Before Instruction
W = 0x17
FSR = 0xC2
Contents of Address (FSR) = 0x00
After Instruction
W = 0x17
FSR = 0xC2
Contents of Address (FSR) = 0x17

NOP — No Operation

Syntax:	[*label*] NOP
Operands:	None
Operation:	No operation
Status Affected:	None
Encoding:	00 | 0000 | 0xx0 | 0000
Description:	No operation.
Words:	1
Cycles:	1

Q Cycle Activity:

Q1	Q2	Q3	Q4
Decode	No operation	No operation	No operation

Example

```
HERE    NOP
```

Before Instruction
PC = address HERE
After Instruction
PC = address HERE + 1

OPTION — Load Option Register

Syntax:	[*label*] OPTION
Operands:	None
Operation:	(W) → OPTION
Status Affected:	None
Encoding:	00 | 0000 | 0110 | 0010
Description:	The contents of the W register are loaded in the OPTION register. This instruction is supported for code compatibility with PIC16C5X products. Since OPTION is a readable/writable register, the user can directly address it.
Words:	1
Cycles:	1

To maintain upward compatibility with future PIC16CXX products, do not use this instruction.

RETFIE — Return from Interrupt

Syntax: [*label*] RETFIE

Operands: None

Operation: TOS → PC,
1 → GIE

Status Affected: None

Encoding:

00	0000	0000	1001

Description: Return from Interrupt. The 13-bit address at the Top of Stack (TOS) is loaded in the PC. The Global Interrupt Enable bit, GIE (INTCON<7>), is automatically set, enabling Interrupts. This is a two cycle instruction.

Words: 1

Cycles: 2

Q Cycle Activity:

1st cycle:

Q1	Q2	Q3	Q4
Decode	No operation	Process data	No operation

2nd cycle:

Q1	Q2	Q3	Q4
No operation	No operation	No operation	No operation

Example

```
RETFIE
```

After Instruction
PC = TOS
GIE = 1

RETLW — Return with Literal in W

Syntax: [*label*] RETLW k

Operands: 0 ≤ k ≤ 255

Operation: k → W;
TOS → PC

Status Affected: None

Encoding:

11	01xx	kkkk	kkkk

Description: The W register is loaded with the eight bit literal 'k'. The program counter is loaded 13-bit address at the Top of Stack (the return address). This is a two cycle instruction.

Words: 1

Cycles: 2

Q Cycle Activity:

1st cycle:

Q1	Q2	Q3	Q4
Decode	Read literal 'k'	Process data	Write to W register

2nd cycle:

Q1	Q2	Q3	Q4
No operation	No operation	No operation	No operation

Example

```
HERE    CALL TABLE   ; W contains table
                     ; offset value
        •            ; W now has table value
        •
        •
TABLE   ADDWF PC     ;W = offset
        RETLW k1     ;Begin table
        RETLW k2     ;
        •
        •
        •
        RETLW kn     ; End of table
```

Before Instruction
W = 0x07
After Instruction
W = value of k8
PC = TOS = Address `Here + 1`

RETURN — Return from Subroutine

Syntax: [*label*] RETURN

Operands: None

Operation: TOS → PC

Status Affected: None

Encoding:

00	0000	0000	1000

Description: Return from subroutine. The stack is POPed and the top of the stack (TOS) is loaded into the program counter. This is a two cycle instruction.

Words: 1

Cycles: 2

Q Cycle Activity:

1st cycle:

Q1	Q2	Q3	Q4
Decode	No operation	Process data	No operation

2nd cycle:

Q1	Q2	Q3	Q4
No operation	No operation	No operation	No operation

Example

```
HERE    RETURN
```

After Instruction

PC = TOS

RLF — Rotate Left f through Carry

Syntax: [*label*] RLF f,d

Operands: $0 \le f \le 127$
$d \in [0,1]$

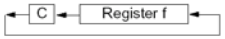
Operation: See description below

Status Affected: C

Encoding:

00	1101	dfff	ffff

Description: The contents of register 'f' are rotated one bit to the left through the Carry Flag. If 'd' is 0 the result is placed in the W register. If 'd' is 1 the result is stored back in register 'f'.



Words: 1

Cycles: 1

Q Cycle Activity:

Q1	Q2	Q3	Q4
Decode	Read register 'f'	Process data	Write to destination

Example 1

```
RLF     REG1,0
```

Before Instruction
REG1= 1110 0110
C = 0
After Instruction
REG1=1110 0110
W =1100 1100
C =1

Example 2

```
RLF     INDF, 1
```

Case 1:
Before Instruction
W = xxxx xxxx
FSR = 0xC2
Contents of Address (FSR) = 0011 1010
C = 1
After Instruction
W = 0x17
FSR = 0xC2
Contents of Address (FSR) = 0111 0101
C = 0

Case 2:
Before Instruction
W = xxxx xxxx
FSR = 0xC2
Contents of Address (FSR) = 1011 1001
C = 0
After Instruction
W = 0x17
FSR = 0xC2
Contents of Address (FSR) = 0111 0010
C = 1

RRF — Rotate Right f through Carry

Syntax: [*label*] RRF f,d

Operands: $0 \le f \le 127$
$d \in [0,1]$

Operation: See description below

Status Affected: C

Encoding:

00	1100	dfff	ffff

Description: The contents of register 'f' are rotated one bit to the right through the Carry Flag. If 'd' is 0 the result is placed in the W register. If 'd' is 1 the result is placed back in register 'f'.

Words: 1

Cycles: 1

Q Cycle Activity:

Q1	Q2	Q3	Q4
Decode	Read register 'f'	Process data	Write to destination

Example 1

```
RRF     REG1,0
Before Instruction
    REG1= 1110 0110
    W   = xxxx xxxx
    C   = 0
After Instruction
    REG1= 1110 0110
    W   = 0111 0011
    C   = 0
```

Example 2

```
RRF   INDF, 1
```

Case 1:

```
Before Instruction
    W   = xxxx xxxx
    FSR = 0xC2
    Contents of Address (FSR) = 0011 1010
    C   = 1
After Instruction
    W   = 0x17
    FSR = 0xC2
    Contents of Address (FSR) = 1001 1101
    C   = 0
```

Case 2:

```
Before Instruction
    W   = xxxx xxxx
    FSR = 0xC2
    Contents of Address (FSR) = 0011 1001
    C   = 0
After Instruction
    W   = 0x17
    FSR = 0xC2
    Contents of Address (FSR) = 0001 1100
    C   = 1
```

SLEEP

Syntax: [*label*] SLEEP

Operands: None

Operation: 00h → WDT,
0 → WDT prescaler count,
1 → $\overline{TO}$,
0 → $\overline{PD}$

Status Affected: $\overline{TO}$, $\overline{PD}$

Encoding:

00	0000	0110	0011

Description: The power-down status bit, $\overline{PD}$ is cleared. Time-out status bit, $\overline{TO}$ is set. Watchdog Timer and its prescaler count are cleared.
The processor is put into SLEEP mode with the oscillator stopped.

Words: 1

Cycles: 1

Q Cycle Activity:

Q1	Q2	Q3	Q4
Decode	No operation	No operation	Go to sleep

Example:

```
SLEEP
```

Note: The SLEEP instruction does not affect the assignment of the WDT prescaler

SUBLW — Subtract W from Literal

Syntax: [*label*] SUBLW k

Operands: $0 \le k \le 255$

Operation: k - (W) → W

Status Affected: C, DC, Z

Encoding:

11	110x	kkkk	kkkk

Description: The W register is subtracted (2's complement method) from the eight bit literal 'k'. The result is placed in the W register.

Words: 1

Cycles: 1

Q Cycle Activity:

Q1	Q2	Q3	Q4
Decode	Read literal 'k'	Process data	Write to W register

Example 1: `SUBLW 0x02`

Case 1: Before Instruction
```
W = 0x01
C = x
Z = x
```
After Instruction
```
W = 0x01
C = 1      ; result is positive
Z = 0
```
Case 2: Before Instruction
```
W = 0x02
C = x
Z = x
```
After Instruction
```
W = 0x00
C = 1      ; result is zero
Z = 1
```
Case 3: Before Instruction
```
W = 0x03
C = x
Z = x
```
After Instruction
```
W = 0xFF
C = 0      ; result is negative
Z = 0
```

Example 2 `SUBLW MYREG`

Before Instruction
```
W = 0x10
Address of MYREG † = 0x37
```
† `MYREG` is a symbol for a data memory location

After Instruction
```
W = 0x27
C = 1      ; result is positive
```

SUBWF — Subtract W from f

Syntax: [*label*] SUBWF f,d

Operands: $0 \le f \le 127$, $d \in [0,1]$

Operation: (f) - (W) → destination

Status Affected: C, DC, Z

Encoding:

00	0010	dfff	ffff

Description: Subtract (2's complement method) W register from register 'f'. If 'd' is 0 the result is stored in the W register. If 'd' is 1 the result is stored back in register 'f'.

Words: 1

Cycles: 1

Q Cycle Activity:

Q1	Q2	Q3	Q4
Decode	Read register 'f'	Process data	Write to destination

Example 1: `SUBWF REG1,1`

Case 1: Before Instruction
```
REG1= 3
W   = 2
C   = x
Z   = x
```
After Instruction
```
REG1= 1
W   = 2
C   = 1      ; result is positive
Z   = 0
```
Case 2: Before Instruction
```
REG1= 2
W   = 2
C   = x
Z   = x
```
After Instruction
```
REG1= 0
W   = 2
C   = 1      ; result is zero
Z   = 1
```
Case 3: Before Instruction
```
REG1= 1
W   = 2
C   = x
Z   = x
```
After Instruction
```
REG1= 0xFF
W   = 2
C   = 0      ; result is negative
Z   = 0
```

SWAPF — Swap Nibbles in f

Syntax: [*label*] SWAPF f,d

Operands: $0 \le f \le 127$
$d \in [0,1]$

Operation: (f<3:0>) → destination<7:4>,
(f<7:4>) → destination<3:0>

Status Affected: None

Encoding:

00	1110	dfff	ffff

Description: The upper and lower nibbles of register 'f' are exchanged. If 'd' is 0 the result is placed in W register. If 'd' is 1 the result is placed in register 'f'.

Words: 1

Cycles: 1

Q Cycle Activity:

Q1	Q2	Q3	Q4
Decode	Read register 'f'	Process data	Write to destination

Example 1

```
SWAPF   REG, 0
```

Before Instruction
REG1= 0xA5

After Instruction
REG1= 0xA5
W = 0x5A

Example 2

```
SWAPF   INDF, 1
```

Before Instruction
W = 0x17
FSR = 0xC2
Contents of Address (FSR) = 0x20

After Instruction
W = 0x17
FSR = 0xC2
Contents of Address (FSR) = 0x02

Example 3

```
SWAPF   REG, 1
```

Before Instruction
REG1= 0xA5

After Instruction
REG1= 0x5A

TRIS — Load TRIS Register

Syntax: [*label*] TRIS f

Operands: $5 \le f \le 7$

Operation: (W) → TRIS register f;

Status Affected: None

Encoding:

00	0000	0110	0fff

Description: The instruction is supported for code compatibility with the PIC16C5X products. Since TRIS registers are readable and writable, the user can directly address them.

Words: 1

Cycles: 1

Example

To maintain upward compatibility with future PIC16CXX products, do not use this instruction.

XORLW — Exclusive OR Literal with W

Syntax: [*label*] XORLW k

Operands: $0 \le k \le 255$

Operation: (W).XOR. k → W

Status Affected: Z

Encoding:

11	1010	kkkk	kkkk

Description: The contents of the W register are XOR'ed with the eight bit literal 'k'. The result is placed in the W register.

Words: 1

Cycles: 1

Q Cycle Activity:

Q1	Q2	Q3	Q4
Decode	Read literal 'k'	Process data	Write to W register

Example 1

```
XORLW   0xAF            ; 1010 1111   (0xAF)
```
Before Instruction
```
                        ; 1011 0101   (0xB5)
    W = 0xB5            ; ---------   ------
```
After Instruction
```
                        ; 0001 1010   (0x1A)
    W = 0x1A
    Z = 0
```

Example 2

```
XORLW   MYREG
```
Before Instruction
```
    W = 0xAF
    Address of MYREG † = 0x37
```
† MYREG is a symbol for a data memory location

After Instruction
```
    W = 0x18
    Z = 0
```

Example 3

```
XORLW   HIGH (LU_TABLE)
```
Before Instruction
```
    W = 0xAF
    Address of LU_TABLE † = 0x9375
```
† LU_TABLE is a label for an address in program memory

After Instruction
```
    W = 0x3C
    Z = 0
```

XORWF — Exclusive OR W with f

Syntax: [*label*] XORWF f,d

Operands: $0 \le f \le 127$, $d \in [0,1]$

Operation: (W).XOR. (f) → destination

Status Affected: Z

Encoding:

00	0110	dfff	ffff

Description: Exclusive OR the contents of the W register with register 'f'. If 'd' is 0 the result is stored in the W register. If 'd' is 1 the result is stored back in register 'f'.

Words: 1

Cycles: 1

Q Cycle Activity:

Q1	Q2	Q3	Q4
Decode	Read register 'f'	Process data	Write to destination

Example 1

```
XORWF   REG, 1          ; 1010 1111   (0xAF)
```
Before Instruction
```
                        ; 1011 0101   (0xB5)
    REG= 0xAF           ; ---------   ------
    W  = 0xB5           ; 0001 1010   (0x1A)
```
After Instruction
```
    REG= 0x1A
    W  = 0xB5
```

Example 2

```
XORWF   REG, 0          ; 1010 1111   (0xAF)
```
Before Instruction
```
                        ; 1011 0101   (0xB5)
    REG= 0xAF           ; ---------   ------
    W  = 0xB5           ; 0001 1010   (0x1A)
```
After Instruction
```
    REG= 0xAF
    W  = 0x1A
```

Example 3

```
XORWF   INDF, 1
```
Before Instruction
```
    W   = 0xB5
    FSR = 0xC2
    Contents of Address (FSR) = 0xAF
```
After Instruction
```
    W   = 0xB5
    FSR = 0xC2
    Contents of Address (FSR) = 0x1A
```

a 33

ulazi i izlazni kanali

U/I port

PIC16F84 raspolaže sa dva ulazno/izlazna porta – **port A** i **port B**. Ovi portovi predstavljaju fizičke kanale za vezu mikrokontrolera sa okruženjem. Oba porta su bidirekciona i ukupno poseduju 13 ulaznih ili izlaznih binarnih kanala. Prema potrebi moguće su različite kombinacije, odnosno slobodne dodele ulazne ili izlazne funkcije svakom od 13 fizičkih kanala. Pojedini kanali su multipleksirani i njima mogu da budu dodeljene neke posebne funkcije. Pored toga, kanali se u električnom smislu izvode na različit način, tako da se korisniku nude odredjene funkcionalne specifičnosti koje se kao pogodnost mogu iskoristiti kod sinteze upravljanja nekog mehatronskog sistema.

Port A je petobitni port, a njemu pripadajući konfiguracioni registar nosi oznaku **TRISA**. Setovanjem pojedinih bitova na logičku jedinicu pripadajućem pinu se dodeljuje funkcija ulaza. U suprotnom, kada je TRISA bit setovan na nulu, pripadajući pin dobija funkciju izlaza. TRISA registar služi za selekciju smera toka signala na portu A. Sadržaj na portu A se čuva u sistemskom registru **PORTA**.

Primer setovanja porta A:

```
CLRF    PORTA           ;Inicijalizacija registra PORTA upisom
                        ;nula u sve bitove memorijskog leča
BSF     STATUS, RP0     ;Izbor memorijske banke 1
MOVLW   0xCF            ;Upis konstante CFh kojom se definise
                        ;smer toka signala u W radni registar
                        ;CFh = 11001111b
MOVWF   TRISA           ;Prepisivanje W u TRISA kontrolni
                        ;registar,PORTA<3:0> = ulazi,
                        ;PORTA<5:4> = izlazi,
                        ;TRISA<7:6> uvek se setuje kao '0'
```

PORT A

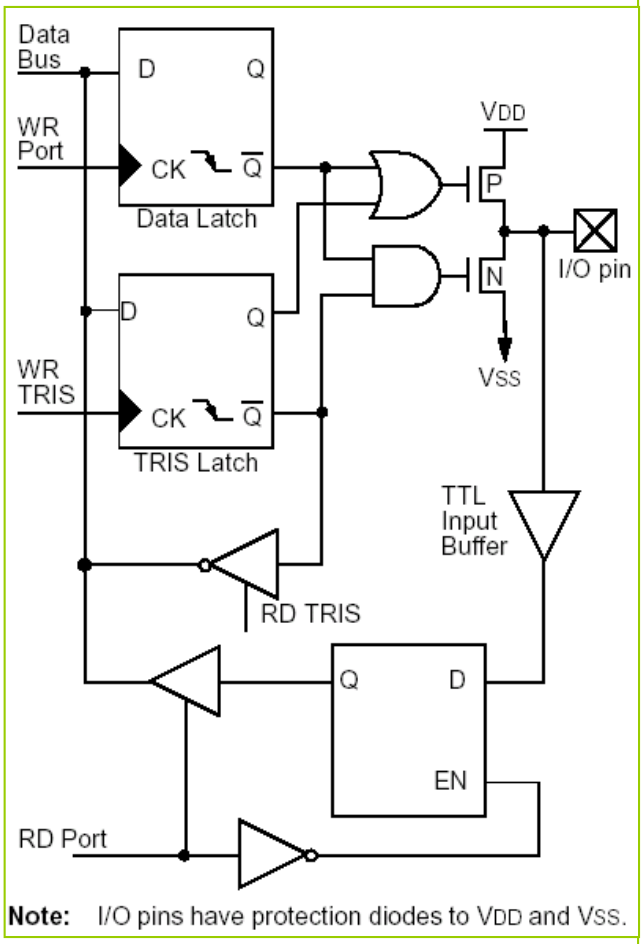
Kod porta A za korisnika su dostupna samo 5 kanala (pina) RA4:RA0 (kanali RA7:RA5 nisu raspoloživi).

Hardversko izvodjenje kanala **RA3: RA0**

Svi kanali su hardverski izvedeni kao TTL ulazi i kao potpuni CMOS izlazi.

WR TRIS kontroliše TRIS leč a RD TRIS kontroliše trostatički NI element i tako odredjuju smer toka informacija – ulaz/izlaz.

Kod resetovanja napajanja mikrokontrolera, svi bitovi **TRISA** registra su setovani na **1** (ulaz) a svi bitovi **PORTA** registra su setovni na **0**.

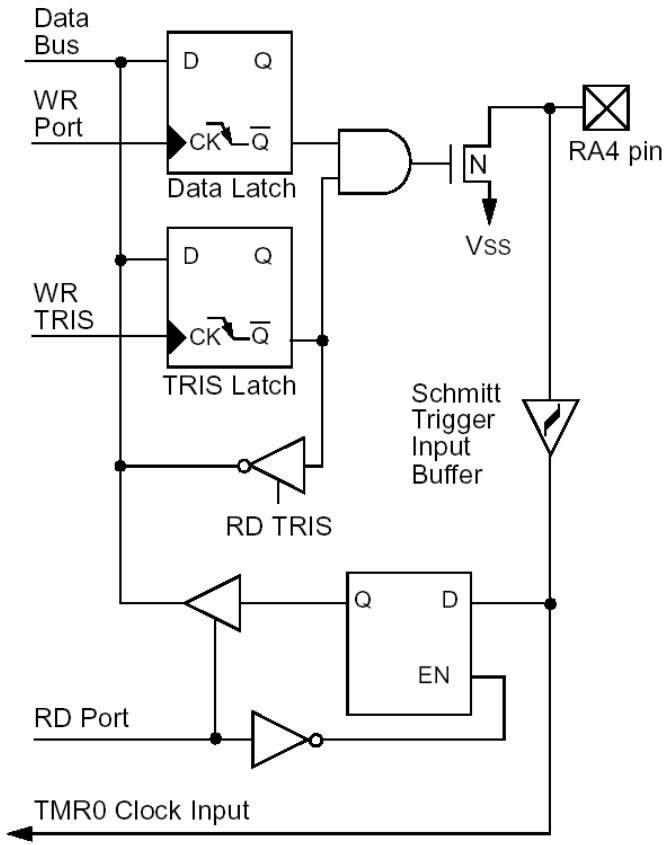


PORT A

Hardversko izvodjenje kanala **RA4**

RA4 kanal je izveden kao Šmitov okidač za uobličavanje ulaznog signala ili kao otvoreni drejn izlaz.

RA4 je multipleksiran sa modulom **TIMER0** kod njegove pobude spoljašnjim oscilatorom.



Note: I/O pins have protection diodes to VDD and VSS.

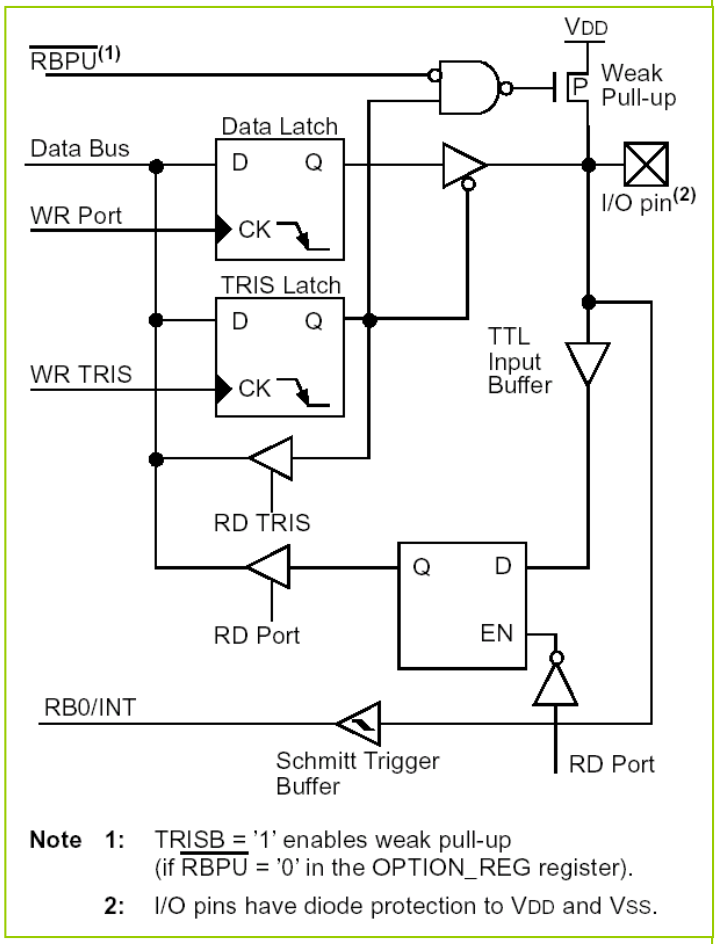
Pregled SFR registara povezanih sa portom A:

Address	Name	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	Value on Power-on Reset	Value on all other RESETS
05h	PORTA	—	—	—	RA4/T0CKI	RA3	RA2	RA1	RA0	---x xxxx	---u uuuu
85h	TRISA	—	—	—	TRISA4	TRISA3	TRISA2	TRISA1	TRISA0	---1 1111	---1 1111

Legend: x = unknown, u = unchanged, - = unimplemented, read as '0'. Shaded cells are unimplemented, read as '0'.

PORT B

Port B je osmobitni port, a njemu pripadajući konfiguracioni registar nosi oznaku TRISB. Setovanjem pojedinih bitova na logičku jedinicu pipadajućem pinu se dodeljuje funkcija ulaza. U suprotnom, kada je TRISB bit setovan na nulu, pripadajući pin dobija funkciju izlaza. TRISB registar služi za selekciju smera toka signala na portu B.



Note 1: TRISB = '1' enables weak pull-up (if RBPU = '0' in the OPTION_REG register).

2: I/O pins have diode protection to VDD and VSS.

Hardversko izvodjenje kanala **RB3: RB0**

Svi kanali su hardverski izvedeni kao TTL ulazi i kao polu CMOS izlazi. Svi kanali imaju interni slabi pul-up ulaz koji se aktivira kontrolnim bitom RBPU (OPTION<7>) FSR registra. Kod resetovanja napajanja slabi pul-up se deaktivira. Kada je konfigurisan kao eksterni prekid RB0 poseduje Šmitov okidač za uobličavanje signala.

WR TRIS kontroliše TRIS leč a RD TRIS kontroliše trostatički NI element i tako odredjuju smer toka informacija – ulaz/izlaz.

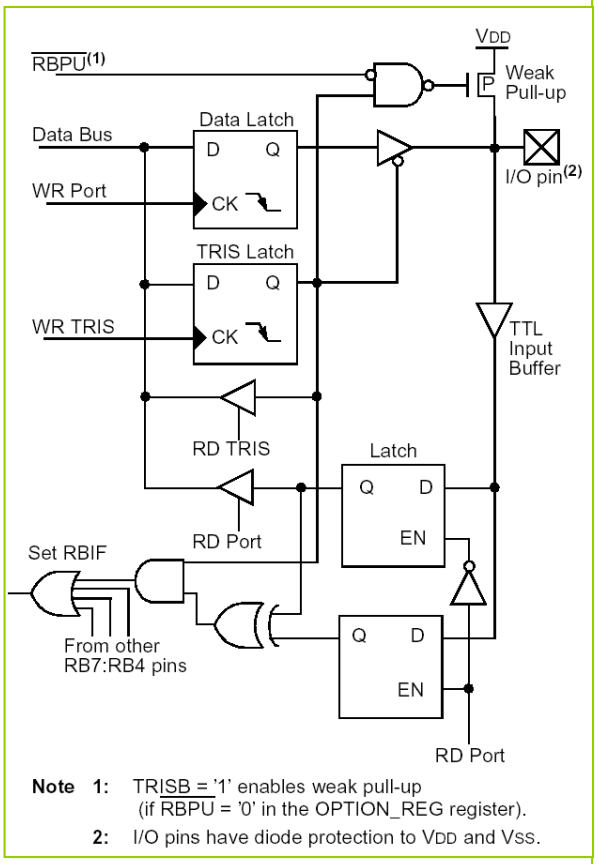
PORT B

Hardversko izvodjenje kanala **RB7: RB4**

Svi kanali su hardverski izvedeni kao TTL ulazi i kao polu CMOS izlazi. Svi kanali imaju interni slabi pul-up ulaz koji se aktivira kontrolnim bitom RBPU (OPTION<7>) FSR registra. Kod resetovanja napajanja slabi pul-up se deaktivira.

Kada su konfigurisani kao ulazi svim ulazima mogu da budu dodeljene **funkcije eksternog prekida** preko bita RBIF (INTCON<0>) FSR registra. Ovi prekidi mogu da se iskoriste za aktiviranje mikrokontolera kada se on nalazi u SLEEP modu.

Kada je konfigurisan kao ulaz i koristi se za serijsko programiranje RB7: RB6 poseduju Šmitov okidač za uobličavanje signala.



Note 1: TRISB = '1' enables weak pull-up (if RBPU = '0' in the OPTION_REG register).
2: I/O pins have diode protection to VDD and VSS.

Pregled SFR registara povezanih sa portom B:

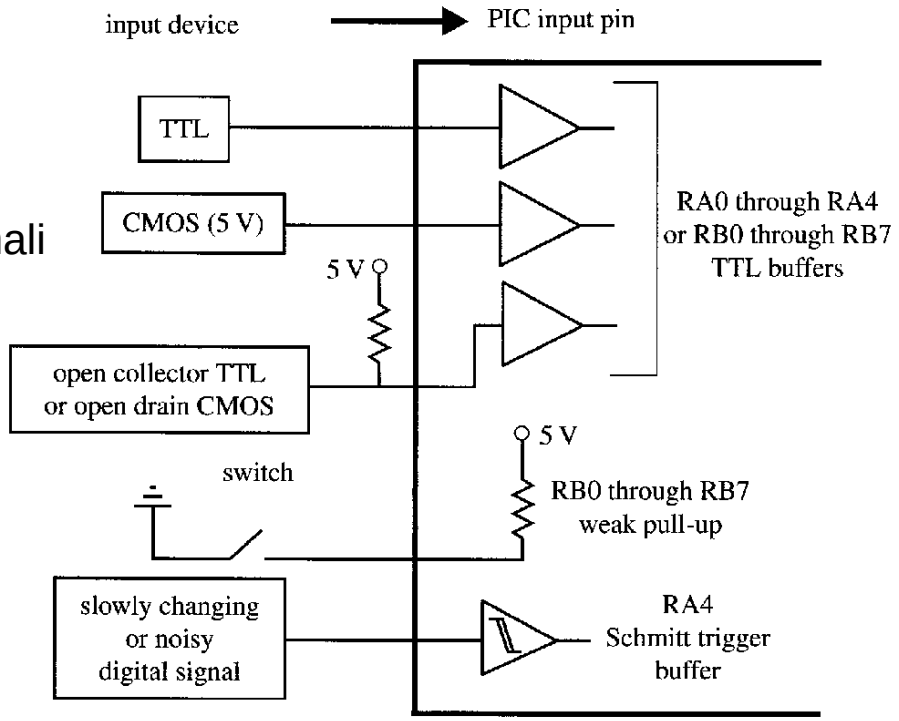
Address	Name	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	Value on Power-on Reset	Value on all other RESETS
06h	PORTB	RB7	RB6	RB5	RB4	RB3	RB2	RB1	RB0/INT	xxxx xxxx	uuuu uuuu
86h	TRISB	TRISB7	TRISB6	TRISB5	TRISB4	TRISB3	TRISB2	TRISB1	TRISB0	1111 1111	1111 1111
81h	OPTION_REG	RBPU	INTEDG	T0CS	T0SE	PSA	PS2	PS1	PS0	1111 1111	1111 1111
0Bh,8Bh	INTCON	GIE	EEIE	T0IE	INTE	RBIE	T0IF	INTF	RBIF	0000 000x	0000 000u

Legend: x = unknown, u = unchanged. Shaded cells are not used by PORTB.

Opšte o ulazima

PIC16F87 omogućava povezivanje sa ulaznim periferijskim jedinicama koje generišu sledeće vrste signala:

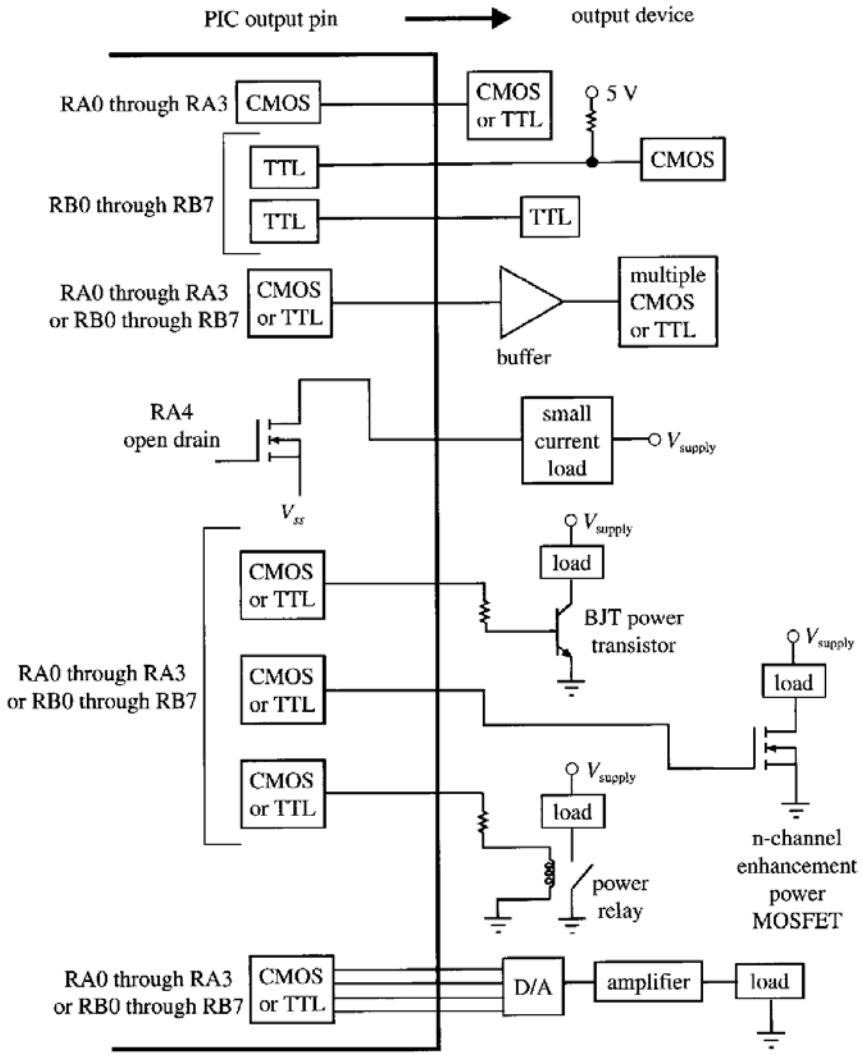
- TTL
- CMOS 5V
- TTL otvoreni kolektor
- CMOS otvoreni drejn
- Mehanički prekidač
- Nestabilni ili šumom kontaminirani signali



Opšte o izlazima

PIC16F87 omogućava povezivanje sa izlaznim periferijskim jedinicama koje na svom ulazu prihvataju sledeće vrste signala:

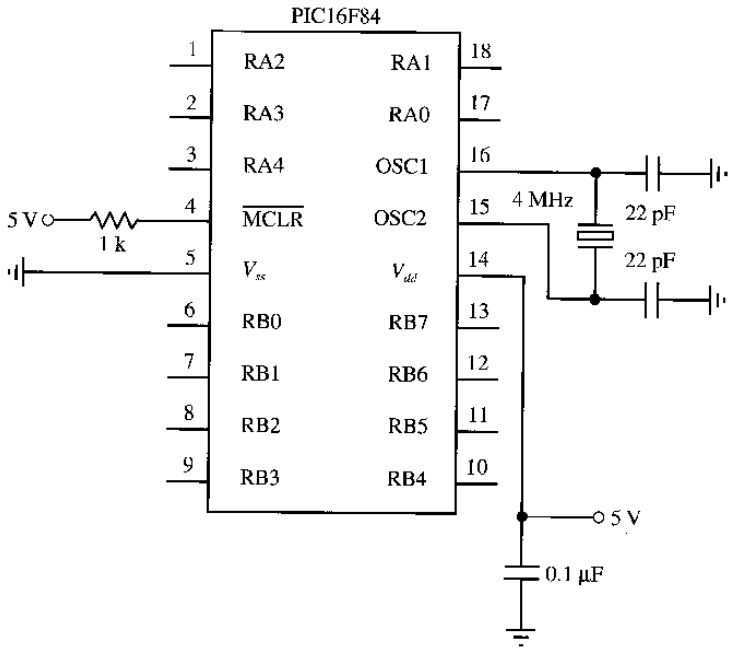
- TTL
- CMOS 5V
- TTL otvoreni kolektor
- CMOS otvoreni drejn
- Mehanički releji
- Bipolarni tranzistori
- MOSFET
- D/A konvertor



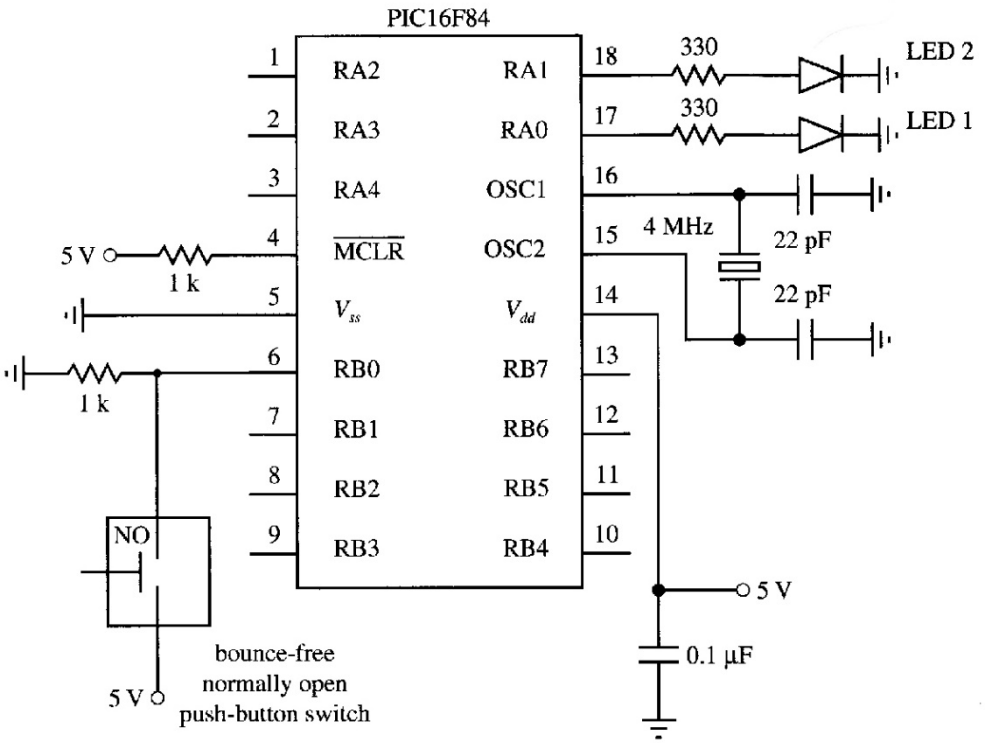
Primer 1:

Zadatak je da se pritiskom na taster na ulazu RB0 aktivira izlaz RA1 (LED 2 svetli) koji ostaje aktiviran onoliko dugo koliko je aktiviran RB0. Kada je RB0 neaktivan, deaktivira se izlaz RA1 i aktivira se izlaz RA0 (LED 1 trajno svetli).

Minimalna HW konfiguracija



HW konfiguracija za postavljeni zadatak



```
;taster.asm (naziv fajla u kome se cuva asemblerski kod)
;Bgd., 11.2005. c PBPetrovic KIBERNETIKA / Masinski fakultet univerziteta u Beogradu

;definicija procesora koji se koristi
list      p=16f84
include<p16f84.inc>

;konfigurisanje porta A, port B je uvek inicijalno setovan kao ulazni
          bcf       STATUS, RP0         ;selekcija banke 0
          clrf      PORTA               ;inicijalizacija registra PORTA
          bsf       STATUS, RP0         ;selekcija banke 1
          movlw     0xFC                ;upis konstante FCh za setovanje porta A
                                        ;FCh = 11111100b
          movwf     TRISA               ;PORTA<1:0> = izlaz, PORTA<7:3> = ulaz
;inicijalizacija portova zavrsena

          bsf       PORTA,0             ;setovanje RA0 (LED1 svetli)

petlja                                  ;glavna petlja
          btfss     PORTB,0             ;cekanje da se RB0 taster aktivira !!!!!
          goto      petlja              ;NE
          bcf       PORTA,0             ;resetovanje RA0 (LED1 NE svetli)
          bsf       PORTA,1             ;setovanje RA1 (LED2 svetli)

cekaj
          btfsc     PORTB,0             ;cekanje da se taster RB0 dekativira !!!!!
          goto      cekaj               ;NE
          bsf       PORTA,0             ;setovanje RA0 (LED1 svetli)
          bcf       PORTA,1             ;setovanje RA1 (LED2 NE svetli)
          goto      petlja              ;povratak na pocetak glavne petlje
          end                           ;kraj instrukcijske liste
```

Svi karakteri koji slede iza znaka ‘ ; ‘ se ne razmatraju kod prevodjenja asemblerskog u mašinski kod.

U zaglavlju se navode osnovni podaci vezani za konkretnu aplikaciju. Zatim se navodi definicija mikrokontrolera za koji se razvija aplikacija. list, including, ..., target su asemblerske instrukcije koje su namenjene za definisanje mikrokontrolera.

Sledeći korak je konfigurisanje portova. Port B je inicijalno uvek konfigurisan kao ulazni i zato se on ostavlja u stanju u kome se nalazi. Kod porta B neophodno je setovati kanal RA0 i RA1 kao izlazne, dok ostali kanali mogu da ostanu u inicijalnom stanju (ulazi). Ovim se omogućava da ulazna električna kola portova A i B učitaju signal sa tastera na RB0 i posalju TTL signal na izlaze RA0 i RA1 za aktiviranje LED dioda priključenih na njih.

Setovanje porta A zahteva rad sa dva sistemska registra (SFR): PORTA koji se nalazi u banci 0 (adresa 05h) i TRISA koji se nalazi u banci 1 (adresa 85h). Zato se prvo instrukcijom `bcf STATUS`, RP0 bira banka 0 a zatim instrukcijom `clrf PORTA` resetuje kompletan sadržaj. Zatim se instrukcijom `bsf STATUS, RP0` setuje bit RP0 čime se otvara pristup banci 1. Ovim je omogućeno da se pristupi konfiguracionom bajtu TRISA u koji treba da se upiše odgovarajući sadržaj. Sadržaj koji se upisuje mora prethodno da se upiše u radni registar W i to se čini instrukcijom `movlw 0xFC` – upis heksadecimalnog broja FC (0x koji prethodi označava heksadecimalni format). Heksadecimalno FC u binarnom obliku glasi 11111100, što znači da će bit koji korespondira RA0 i RA1 biti resetovan, što dalje znači da će smer toka podataka na tim pinovima biti od mikrokontrolera ka okruženju – izlaz. Ovo se ostvaruje prenosom sadržaja radnog registra W u TRISA registar instrukcijom `movwf TRISA`. Ovim su svi portovi konfigurisani prema potrebama konkretne aplikacije.

Instrukcijom `bsf PORTA, 0` setuje se bit (*bit-set-file*) bit 0 na portu A, što za posledicu ima aktiviranje LED 1.

Dalje se zahteva realizacija postavljenog zadatka koja se može ostvariti primenom beskonačne petlje koja neprekidno skenira promenu stanja na RB0 pinu porta B.

Početak petlje se označava labelom kojoj se dodeljuje proizvoljno alfanumeički ime, u ovom slučaju to je `petlja`. Instrukcijom `btfss PORTB,0` se ispituje stanje na RB0. Instrukcija `btfss` ima značenje: *bit-test-file-skip-if-set*. Dakle, ukoliko je bit RB0 resetovan (taster nije pritisnut) izvršiće se naredna instrukcija: `goto petlja`, u suprotnom ona se preskače i izvršava se instrukcija `bsf PORTA,1` čime se setuje RA1 odnosno, bit 1 porta A, što za posledicu ima da LED 2 počne da svetli. Istovremeno se instrukcijom `bcf PORTA,0` resetuje RA0 i time isključuje LED 1.

Zatim se ulazi u narednu pod petlju kojom se proverava da li je taster isključen. Za ovo se koristi instrukcija `btfsc PORTB,0` odnosno, *bit-test-file-skip-if-clear*. Ukoliko je RB0 setovan izvršava se naredna instrukcija `goto cekaj`, koja (clear) vraća programski brojač na labelu `čekaj`. U suprotnom, kada je RB0 resetovan (clear) izvršava se instrukcija `bsf PORTA,0` koja uključuje LED 1 i instrukcija `bcf PORTA,1` koja isključuje LED 2.

Prethodnim se završava glavna programska petlja što se ostvaruje instrukcijom `goto petlja`.

Kompletan programski kod se završava instrukcijom `end`.

Program napisan u asembleru treba prevesti u mašinski kod i zatim ga preneti u programsku memoriju mikrokontrolera.

MICROCHIP nudi korisnicima svojih mikrokontrolera **MPASM Assembler** univerzalni programski paket za generisanje mašinskog koda za sve PIC mikrokontrolere (cross-assembler) koji je namenjen za rad u IBM PC kompatibilnim računarima.

Programiranje u asembleru je vrlo delikatan zadatak. Rad u asembleru podrazumeva način razmišljanja koji je vrlo blizak radu aritmetičke jedinice i periferijskih modula, što se u potpunosti razlikuje od rada sa višim programskim jezicima. Realizacija skoro svih logičkih funkcija zahteva primenu primitivnih mikrologičkih operacija direktno uslovljenih konstrukcijom aritmetičko logičke jedinice. Viši programski jezici su značajno bliži praktičnom razmišljanju koje ljudski mozak koristi u svakodnevnim aktivnostima. Ipak, pored očiglednih teškoća, rad u asembleru pruža i očigledne prednosti kroz potpun uvid u sve resurse mikroprocesora i modula koji ga okružuju, čime se omogućava optimalno korišćenje raspoloživog mikroračunarskog HW i kodiranje kompaktnih i računski efikasnih programa.

Postupak kreiranja asemblerskog koda može se značajno pojednostaviti primenom viših programskih jezika, koji pre svega vode računa o zadovoljenju funkcija definisanih projektnim zadatkom, a manje o HW aspektima. Prevodjenjem ovakvog koda u njemu ekvivalentan asemblerski kod i dalje, generisanje mašinskog koda iz tako nastalog asemblerskog koda, završava se proces razvoja programa kojim se programira rad konkretnog mikrokontrolera. Primer ovakvog jezika je **MicroPASCAL** ili **MicroC** firme Mikroelektronika iz Beograda. Ovakvi programski paketi predstavljaju jedno kompletno razvojno okruženje, koje sem kodiranja, omogućava proveru (debagiranje), emulaciju, dokumentovanje i upisivane proizvedenog mašinskog koda u memoriju mikrokontrolera.