Факултет инжењерских наука Универзитета у Крагујевцу

Назив студијског програма: Машинско инжењерство

Ниво студија: Основне академске студије

Модул: Информатика у инжењерству

Предмет: Архитектура рачунарских система

Број индекса: 63/2009

Мирослав Г. Мунић

8-битни микроконтролер PIC16F84

Завршни рад

Комисија за преглед и одбрану:

1. **Проф. др Јасна Радуловић – ментор**
2. _____ Датум одбране:_____
3. _____ Оцена:_____

У оквиру овог завршног рада кандидат треба да проучи препоручену, као и ширу литературу која се односи на микроконтролер PIC16F84, кренувши од основних појмова до примене микроконтролера. На бази усвојеног знања кандидат треба да уради задатак, чиме ће показати да је савладао програмирање на асемблерском језику микроконтролера.

Препоручена литература:

[1] Драган Андрић, Небојша Матић - PIC микроконтролери, Агенција АРС, Београд 2000.

[2] Стојан Трифуновић - Упутство за руковање PIC16F84, Сокобања 2008.

[3] Јелена Бубало, Радослав Илић, Видојко Вељковић - Програмирање PIC микроконтролера PIC16F84, Интер ХИТ, Београд 2001.

Крагујевац, 12.10.2013. Ментор:

Проф. др Јасна Радуловић,
редовни професор

РЕЗИМЕ

У овом раду обрађен је осмобитни микроконтролер PIC16F84, објашњена је његова архитектура, принцип рада и организација меморије са посебним освртом на архитектуру и организацију овог микроконтролера. Такође је урађен задатак који симулира рад показивача правца на аутомобилу са детаљним објашњењем програма.

Кључне речи: Микроконтролер, PIC16F84, показивач правца аутомобила.

ABSTRACT

In this paper has been processed the eight-bit microcontroller PIC16F84, explained its architecture, operating principles and memory organization with a focus on special features of the microcontroller. Also a task was made which simulates operation of direction indicators on the car with a detailed explanation of the program.

Keywords: *Microcontroller*, PIC16F84, auto turn signals.

Садржај

Увод

Микроконтролер је једно од највећих техничких достигнућа које је обележило двадесети век. Са напредком технологије растао је степен интеграције интегрисаних кола, што је довело до појаве првих микропроцесора и микроконтролера. Првенствено због своје цене , микроконтролери налазе све већу примену у свим областима у изради софтверски контролисаних уређаја и система.

Друга велика предност микроконтролера, због које су данас неизбежни, јесте чињеница да се могу програмирати у вишим програмским језицима типа C, Paskal, Basic. Ово омогућава њихово програмирање па чак и без знања асемблерског језика. То повећава број људи који их могу програмирати, па тако постају приступачнији људима који се електроником баве из хобија.

Микроконтролер је мали рачунар, реализован у облику интегрисаног кола који обједињује све потребне компоненте, како би могао самостално да функционише. Ту спадају интегрисани микропроцесор, меморија, дигитални и аналогни улази/излази, тајмери, бројачи, осцилатори и други склопови (у зависности од врсте и намене микроконтролера) који су раније били сваки у свом интегрисаном чипу. Микроконтролер се програмира да нормално ради у бесконачној петљи и за то време очитава улазе и подешава излазе у зависности од програма који му је задат.

Главна разлика између микропроцесора и микроконтролера је да су микроконтролери оптимизовани у правцу интеграције кола, управљања процеса у реалном времену, мале цене и ниске потрошње енергије, док је код микропроцесора акценат стављен на брзину и перформансе. Значи микроконтролери су наменски уређаји који могу обављати одређене функције у границама које одређују сами ресурси микроконтролера, док се микропроцесори користе у опслуживању спољних јединица. *[5]*

1. PIC микроконтролер

PIC микроконтролер (Programmable Intelligent Computer) је оригинални производ компаније Microchip, врло је популаран међу инжењерима и електронским почетницима, а долази у пуно различитих верзија, свака са посебним компонентама и могућностима. Многи електронски пројекти могу се врло лако извести помоћу микроконтролера из PIC породице, попут дигиталних сатова, врло једноставних видео игрица, робота, серво контролера и многих других. Њихова примена је врло широка, а њихове могућности и варијанте су небројене, а могу се набавити по врло приступачним ценама што их чини још популарнијима. Управо због своје разноврсности, али понајвише због прилично једноставног програмирања, PIC микроконтролери имају доста велику примену у едукацији, како средњошколској тако и факултетској, у дигиталној логици, дигиталној електроници, аутоматици и другим сродним областима. Данас постоје 8, 16 и 32 битни модели PIC микроконтролера.

2. Карактеристике микроконтролера PIC16F84

Архитектура је RISC типа (Reduced Instruction Set Computer), то значи има мало инструкција (свега 35), али се извршавају максимално брзо; микроконтролер користи две магистрале и то једна за податке (8-бита) а друга за инстукције (14-бита).

Слика 1: Изглед микроконтролера PIC16F84 *[4]*

- PIC16F84 има 18 пинова од којих чак 13 могу да се користе као I/O линије (I=input - улаз, O=output - излаз);
- FLASH програмску меморију од 1024 речи (свака је дужине 14 битова);
- RAM меморију од 68 бајтова;
- 64 бајта интерног EEPROM-а.

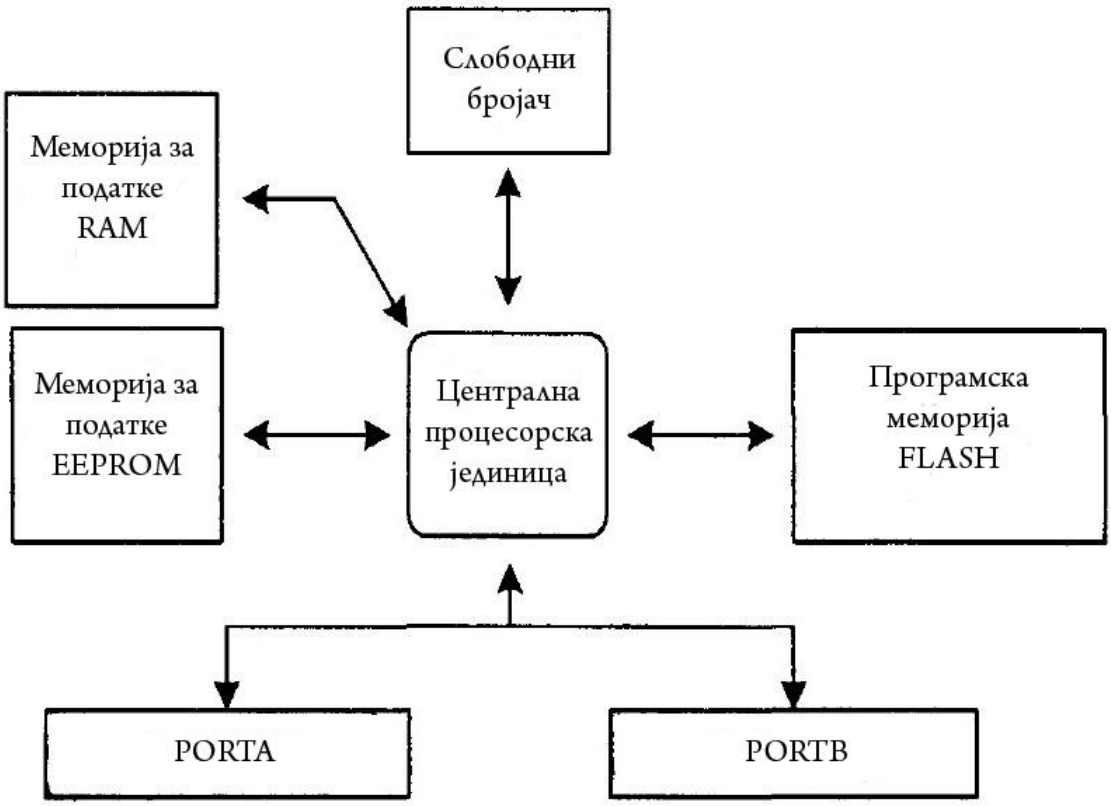
Програмска FLASH меморија може да се репрограмира око 1000 пута и гарантовано "чува" програм бар 40 година.

EEPROM (Electrically Erasable Programmable Read-Only Memory) може да се репрограмира чак 10 милиона пута и такође чува податке бар 40 година.

RAM меморије има више од 68 бајтова, али је један део резервисан за регистре са специјалним функцијама. Такође постоји и STACK од 8 нивоа;

Напон напајања се креће од 2 до 6V, а потрошња је мања од 2 μA при напајању од 5V и такту од 4MHz. Потрошња при напајању од 2V и такту од 32kHz је око 15 μA, а када је у SLEEP моду при 2V пада испод 1 μA.

Што се тиче такта, он може да иде од 0 до 20 MHz; прво су постојале две верзије IC-а, до 4 и до 10MHz, али се појавила и верзија до 20 MHz. *[4]*



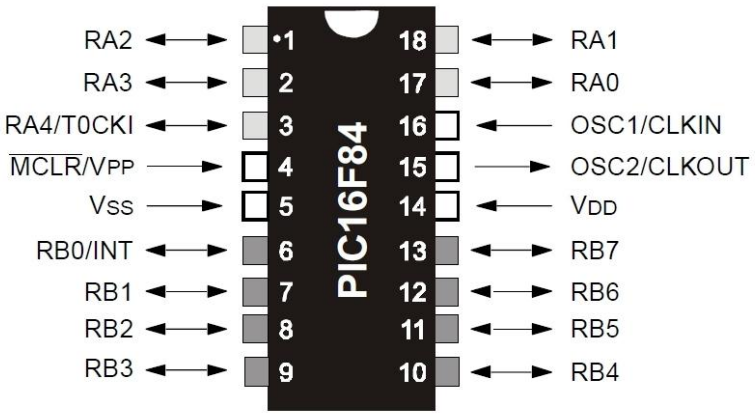
Слика 2: Блок шема микроконтролера PIC16F84

2.1 Улазно/Излазне карактеристике

- 13 улазно-излазних, појединачно управљаних пинова
- максимална улазна струја 25mA по пину
- максимална излазна струја 20мА по пину
- 8- битни тајмер/бројач са програмибилним дељењем фреквенце

2.2 Распоред и функција пинова

На следећој слици (Слика 3) приказан је распоред пинова на микроконтролеру а у даљем тексту и објашњена њихова функција коју обављају.

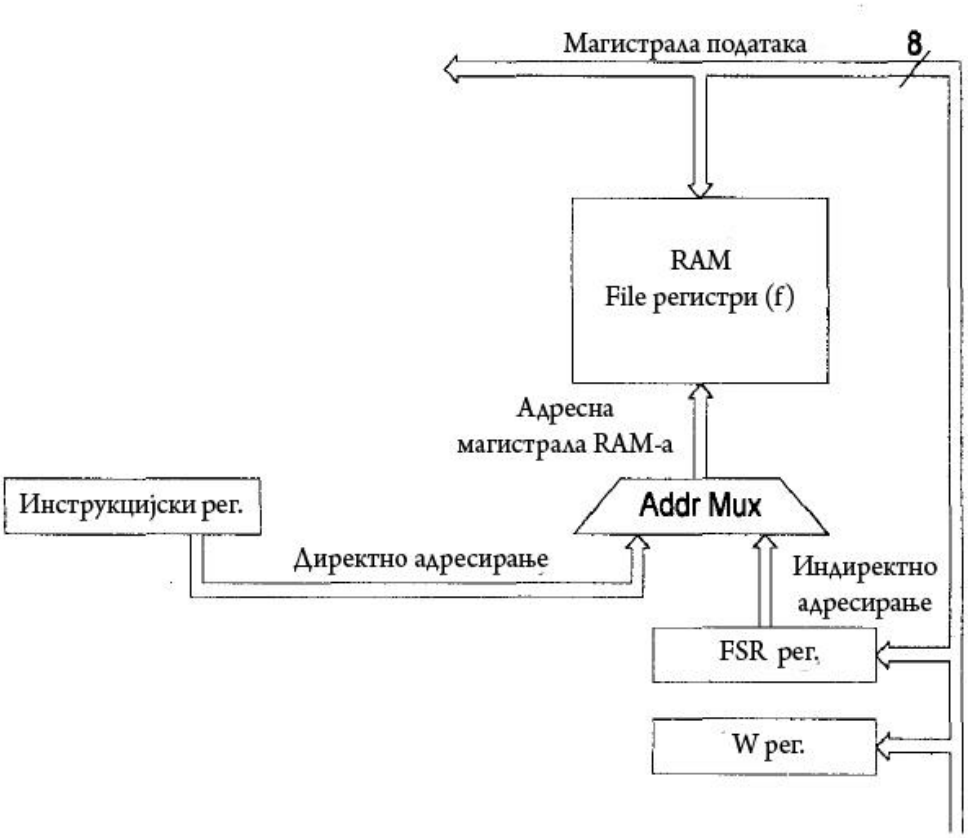


Слика 3: Распоред пинова на микроконтролеру PIC16F84 *[8]*

На пину 1 је RA2, I/O линија порта A, TTL (*transistor-transistor logic*- транзисторско-транзисторска логика) типа; на пину 2 је RA3, I/O линија порта A, TTL типа; на пину 3 је RA4/TOCKI, I/O линија порта A, такође служи као улаз до TMR0 бројача. Када је I - улаз, онда је ST (schmitt trigger) типа, а када је O -излаз, onda је open-drain; на пину 4 је $\overline{\text{MCLR}}$ (инвертован MCLR, обележава се и са -MCLR) је reset пин (ресет када је на ниском нивоу) или се доводи напон програмирања; на пину 5 се налази маса напајања; на пину 6 се налази RB0/INT, I/O линија порта B, или извор спољног интерапта, TTL/ST (ST када је улаз за интерапт) типа; на пину 7 је RB1, I/O линија порта B, TTL типа; на пину 8 је RB2, I/O линија порта B, TTL типа; на пину 9 је RB3, I/O линија порта B, TTL типа; на пину 10 је RB4, I/O линија порта B, такође извор интерапта при промени нивоа, TTL типа; на пину 11 је RB5, I/O линија порта B, извор интерапта при промени нивоа, TTL типа; на пину 12 је RB6, I/O линија порта B, извор интерапта при промени нивоа, такт при програмирању, TTL/ST (ST при програмирању); на пину 13 је RB7, I/O линија порта B, извор интерапта при промени нивоа, подаци при програмирању, TTL/ST (ST при програмирању) типа; на пину 14 се налази позитиван пол напајања; на пину 15 је OSC2/CLKOUT, то је пин намењен спајању са осцилатором; на пину 16 је OSC1/CLKIN, то је пин намењен спајању са осцилатором; на пину 17 је RA0, I/O линија порта A, TTL типа; на пину 18 је RA1, I/O линија порта A, TTL типа.

2.3 Централна процесорска јединица

Централна процесорска јединица се у стручној литератури означава као CPU. Скраћеница је настала од почетних слова енглеских речи Centrall Procesing Unit па се одатле и у свакодневном говору зове CPU или процесор. CPU се обично схвата као мозак микроконтролера. Тај део је одговоран за проналажење и прибављање одговарајуће инструкције која се треба извршити, декодирање те инструкције и на крају њено извршење.



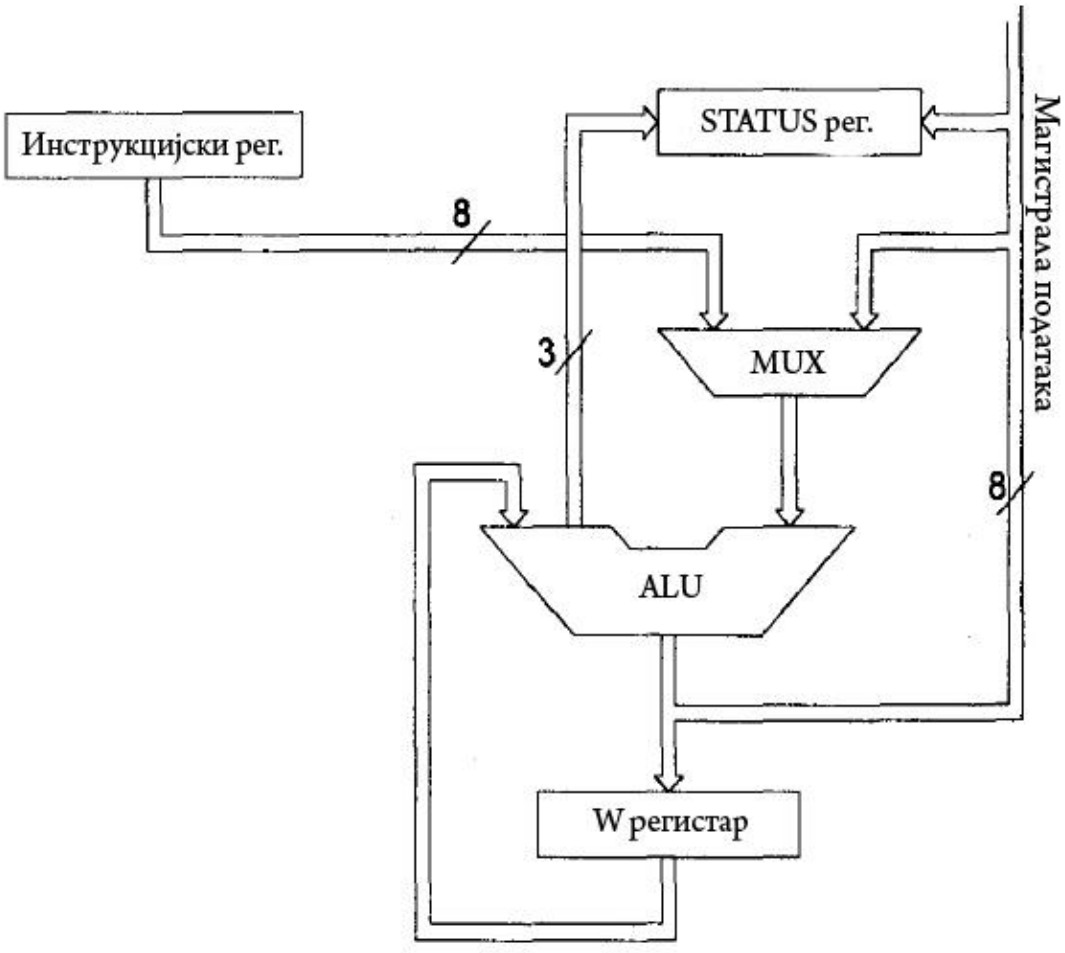
Слика 4: Шема централне процесорске јединице *[1]*

Централна процесорска јединица повезује све делове микроконтролера у једну целину. Најважнија улога је свакако декодирање програмских инструкција. Када програмер напише програм инструкције имају за човека јасан облик нпр. MOVLW 0x20. Међутим, да би микроконтролер то разумео, тај "словни" облик инструкције се мора превести у низ нула и јединица који се назива "опкод" по енглеској речи "opcode". Тај превод из словног у бинарни облик раде преводиоци као што је нпр. асемблерски преводилац(скраћено га сви звоу "асемблер"). Тако добијену инструкцију из

програмске меморије централна процесорска јединица мора да декодује да би се из табеле свих инструкција изабрао скуп акција које извршавају потребан задатак дефинисан у тој инструкцији. Како инструкције у себи могу садржати задатке који захтевају разна пребацивања података из једне меморије у другу, из меморије на портове или нека израчунавања, то онда CPU мора бити повезан са свим деловима микроконтролера. Ово омогућује магистрала података и адресна магистрала.

2.4 Аритметичко логичка јединица

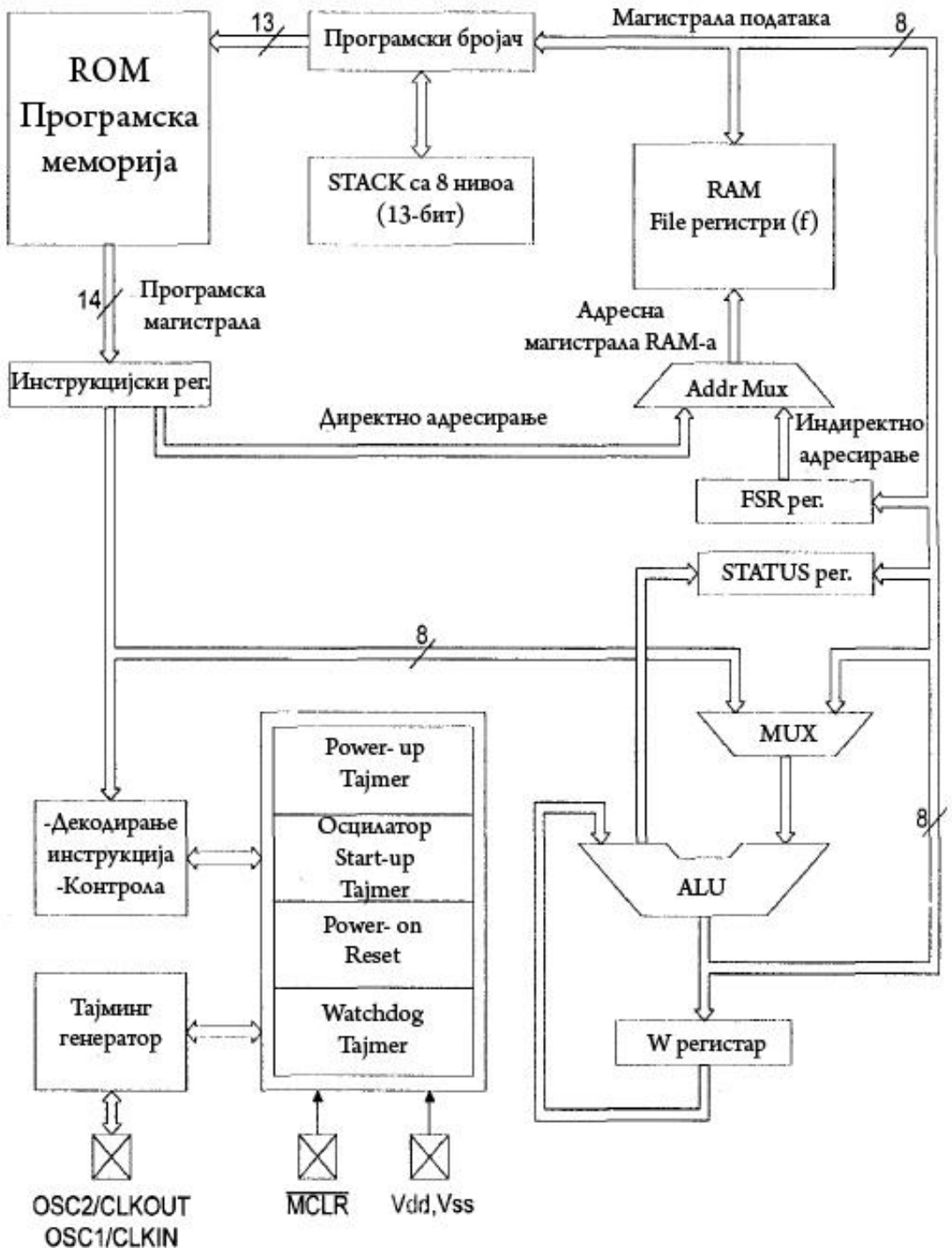
Аритметичко логичка јединица (Arithmetic Logic Unit - ALU) задужена је за обављање операција сабирања, одузимања, померања (лево или десно унутар регистра) и логичких операција. Померање података унутар регистра је познатије и као "шифтовање". Назив је настао од енглеске речи "shift" што значи "померити". PIC16F84 садржи 8-битну аритметичко логичку јединицу и 8-битне радне регистре.



Слика 5: Аритметичко-логичка јединица и начин њеног рада *[1]*

У инструкцијама са два операнда, обично је један операнд у радном регистру (W регистар), а други је неки од регистара или константа. Под "операндом" се подразумева садржај над којим се врши нека од операција а регистар је било који од GPR или SFR регистара. GPR је скраћеница од "General Purposes Registers" а SFR од "Special Function

Registers". У инструкцијама са једним операндом, операнд је или W регистар или неки од регистара. Као додатак за извођење аритметичких и логичких операција, ALU контролише статусне битове (битови који се налазе у STATUS регистру). Извршавање неких инструкција утиче на статус битове, што зависи од стања самог резултата. Зависно од инструкције која се извршава, ALU може утицати на вредност Carry бита (C), Digit Carry (DC), и Zero (Z) битова у STATUS регистру.



Слика 6: Детаљна блок шема микроконтролера PIC16F84 *[1]*

3. Посебне функције микроконтролера

Посебне функције уграђене у овај MCU (микроконтролер) су:

- power-on reset (POR), обезбеђује интерни ресет микроконтролера при укључењу напајања;
- power-up timer (PWRT), обезбеђује фиксно кашњење од 72 ms при укључењу напајања док се напон не стабилизује;
- осцилатор start-up timer (OST), држи IC *(Integrated Circuit*) у ресет стању док се не стабилизује осцилатор;
- watchdog timer (WDT), омогућава ресет IC-а након одређеног времена ако је запао у мртву петљу (dead lock) и сл.;
- заштита од читања садржаја меморије;
- SLEEP функција за минималну потрошњу;
- одабир неколико врста осцилатора;
- серијско програмирање чак и када је IC монтиран на PCB;
- 4 извора интерапта- прекида; интерапт, је у ствари, прекид редовног извршавања програма из неког разлога, одлазак на део програма који се извршава у том случају.

Након што се заврши интерапт рутина, MCU се враћа тамо где је стао у извршавању редовног програма (адресу повратка чува на STACK-у).

Унутар микроконтролера PIC16F84 све је дигитално. Микроконтролер разуме једино логичку 0 и логичку 1, односно напонске нивое од 0V и +5V. Међутим, понекад се догоди да му се уместо ових напона доведу напони који одступају од оних препоручених у његовим спецификацијама. Функција свих ових доле набројаних термина је да се спрече управо такви ефекти.
На примеру укључења и искључења светла на прекидачу може се приметити варничење на контактима. Идеално би било када би прекидач одједном могао пребацити своје стање од укљученог у искључено и обрнуто. Међутим, како се контакти прекидача физички не могу превише брзо кретати, неизбежно је јављање пар варница пре коначне промене стања. Код укључења или искључења светла, ова појава није битна. Али микроконтролер због своје велике брзине услед ове појаве може закључити да је тастер на његовом улазном пину уместо једном притиснут више пута. Да би се ова појава неутралисала потребно је у микроконтролеру реализовати смањење шума на улазном сигналу, односно debouncing.

Debouncing рутина тестира стање пина, сачека мало (колико је потребно да се стабилизује стање прекидача) и онда га опет тестира. Тестирање се, уколико је потребна већа прецизност, може узастопно извршавати и више од два пута. Уколико су у свим случајевима добијена иста логичка стања на пину, то значи да је тастер сигурно притиснут, а у противном да није. Овај принцип стандардно користе тастатуре код рачунара.

Код микроконтролера нема „сивог“ стања. Стање на његовом пину мора бити тачно одређено. Уколико се улазном пину микроконтролера доведе овако недефинисан напон, он може прочитати 0, а може и 1. Нема правила.

Уколико улазни пин (у најгорем могућем случају) није повезан на екстерно коло, напон на њему може заосциловати, па чак проузроковати уништење микроконтролера.
У пракси се некад јави ситуација да је потребно на микроконтролер довести управо овакав „сиви“ напон. У том случају пожељно је користити узлазе са шмитовим окидачем (Schmitt trigger).
У PIC16F84 шмитов окидач налази се на RA4 пину. На RB0 пину налази се само када је овај дефинисан као извор интерапта (више о њима касније).
Код улаза са шмитовим окидачем нема недефинисаних стања улазног пина. Стање на пину промениће се тек када се сигнал довољно приближи његовој дефинисаној области. Ако се након тога врати ка средњој (сивој) области, улаз ће и даље јављати његово последње стабилно дефинисано стање.
Очигледно је да ће при повезивању тастер прекидача на улазе PIC16F84 микроконтролера бити потребно да његови улази буду у сваком тренутку на тачно дефинисаном логичком нивоу. Ово се хардверски реализује Pull-up (повући нагоре) или Pull-down (повући надоле) отпорницима.

Pull up отпорник је отпорник повезан између напона напајања микроконтролера и улазног пина, при чему се тастер прекидач везује између пина и масе. Када је тастер отпуштен, pull up отпорник доводи напон напајања директно на пин, постављајући га тако у стање логичке јединице, све док се не притисне тастер. Без њега би напон на пину вероватно зашао у недефинисано стање. Овај отпорник не треба бити превише мале вредности због мање потрошње струје при притиснутом тастер прекидачу. Препоручене вредности су од 4,7kΩ do 10kΩ.

Све инструкције са којима ради PIC16F84 су четрнаестобитне. Међутим, унутар самих инструкција налази се код инструкције (по коме се нпр. инструкција CALL разликује од MOVLW) и операнд, који представља податак са којим радимо (бит, бајт или оба у нпр. bsf PORTA,2 инструкцији). Постоје и инструкције које немају операнд (нпр. CLRW).
Пре извршења сваке инструкције PIC гледа садрежај регистра PCL и PCLATH. Они (тачније битови 5 PCLATH + 8 PCL) чине јединствен тринаестобитни PC (**P**rogram **C**ounter) регистар. У њему је смештена адреса извршавања текуће инструкције.
По извршавању инструкције, PC се увећава за 1, и прелази на следећу инструкцију.
Уколико је текућа инструкција инструкција условног скока (DECFSZ, INCFSZ, BTFSC или BTFSS), резултат инструкције (тачно или нетачно – 1 или 0) додаје се на PC, чиме је омогућен скок преко следеће инструкције. Међутим уколико је текућа инструкција GOTO, она ће у PC поставити нову адресу, и извршење програма наставиће се одатле.
Специјалан случај представља коришћење CALL инструкције. PIC мора на неки начин упамтити место са кога је скочио на потпрограм. Та информација чува се у стеку.

WDT (Watchdog)- То је, у ствари, независтан RC осцилатор уграђен у сам микроконтролер. Намена му је да под извесним условима изазове интерни ресет, како би спречио упадање програма у мртву петљу и повећао сигурност рада. Ако је WDT омогућен (то се ради или селектовањем одређене опције при програмирању или у самом програму), програм треба с времена на време да ресетује тај бројач применом инструкције CLRWDT, чиме циклус бројања почиње испочетка; ако програм из неког разлога западне у мртву петљу, вероватно се неће извршити ова инструкција (мада не мора да значи, нарочито ако се у петљи нађе и CLRWDT) и након истека времена WDT-а извршиће се ресет и микроконтолер "извући" из мртве петље. Основни период WDT-а износи приближно 18 ms (мада варира у зависности од разних фактора - толеранција,

температура...), а применом прескалера максимално време може да се продужи до 2,3 секунде.

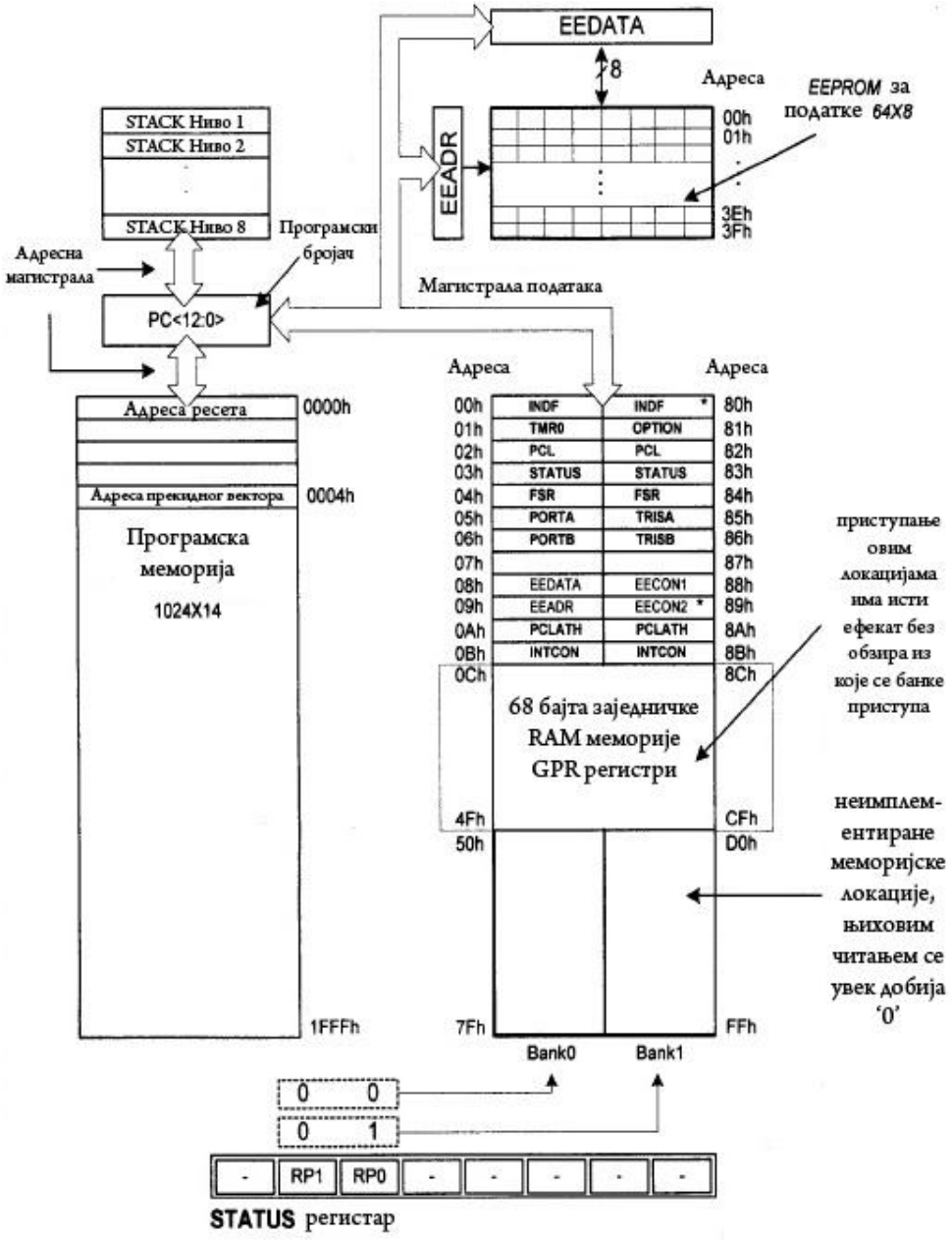
SLEEP „спавајући“ мод представља стање у којем се микроконтролер налази у режиму смањене потрошње енергије. У SLEEP моду блокира се рад главног осцилатора. Инструкције се престају извршавати, тајмер престаје са радом, а WDT се (уколико је укључен) ресетује (али наставља са радом због сопственог осцилатора). Портови задржавају своја стања. За најмању могућу потрошњу потребно је пројектовати спољна електронска кола тако да не вуку струју из излазних пинова портова, и да улазне пинове поставе на стабилан логички ниво (0 или 1). RA4/T0CKI пин би такође требао бити на стабилном нивоу (не би смео бити излазни са логичком 1 јер је не може дати). Напон напајања микроконтролера може се спустити и до 1,5V без губитка података. Потрошња микроконтролера пада са око 2mA на око 5 μ A .
У SLEEP мод улази се SLEEP инструкцијом, а из њега се микроконтролер може „пробудити“ на три начина.

1. Довођењем логичке нуле на MCLR пин микроконтролера, што проузрокује ресет микроконтролера. Више о ресету, у наредном делу текста.
2. Ресетом преко истека времена WDT.
3. Интераптом на RB0/INT пину, интераптом при промени стања на PORTB регистру, битовима 4 до 7, или интераптом изазваним завршетком уписа у EEPROM.

Интерапт изазван тајмером не може се користити за буђење из SLEEP мода, јер је у SLEEP моду и тајмер успаван.
Буђење из SLEEP мода интераптима разликује се од обичних интерапта по томе што је могуће да се уместо скока на интерапт рутину, извршавање програма настави иза SLEEP инструкције. Наиме, уколико је у SLEEP моду GIE сетован, при интерапту ће се микроконтролер пробудити и скочити на интерапт рутину, као у обичном програму. Али, уколико је GIE ресетован, при интерапту (дозвољеним одговарајућим интерапт флагом) ће се пробудити и наставити даље извршење програма, без скока на интерапт рутину. У зависности од тренутка стицања услова за појаву интерапта могуће је да се интерапт догоди непосредно пре SLEEP инструкције. У том случају, SLEEP инструкција се неће извршити, па неће бити ресетован WDT и његов прескалер, ресетован $\overline{PD}$ бит и сетован $\overline{TO}$ бит. Због тога је потребно ручно ресетовати WDT (уколико се користи) непосредно пре SLEEP инструкције. Ако је потребно сазнати да ли се извршила SLEEP инструкција, то се може проверити тестирањем $\overline{PD}$ бита STATUS регистра.
У Sleep моду осцилатор је нестабилан приликом буђења. Наиме, пошто је осцилатор за време спавања блокиран, није му баш лако да одједном почне производити чисте тактове. Буђење неће бити могуће док се осцилатор не стабилизује. Зато се при коришћењу SLEEP мода не може вршити мерење времена бројањем инструкцијских циклуса инструкција у програму, чак иако се буђење интераптом дешава тачно одређено време након уласка у Sleep мод. Како RC осцилатор најлакше почиње са осцилацијама, време за његову стабилизацију је скоро тренутно, док је код употребе осталих врста осцилатора потребно 1024 такта осцилатора за комплетну стабилизацију. Све док се стабилизација такта не заврши, микроконтролер се неће моћи пробудити из SLEEP мода. *[2]*

4. Организација меморије

Меморија микроконтролера PIC 16F84 подељена је у два блока (Слика 7): програмску и меморију података.



Слика 7: Организација меморије

- **Програмска меморија** је величине 1024 локација ширине 14 бита. Адреса 0x00 резервисана је за ресет вектор, а локација 0x04 за прекидни вектор.

- **Меморија података** састоји се од 64 осмобитних локација EEPROM (Electrically Erasable Programmable Read-Only Memory) меморије и 68 локација RAM меморије од адресе 0x0C до 0x4F. Може се приметити специфична подела меморије "по ширини", чиме се формирају два сегмента, "банке"–BANK0 и BANK1. Првих дванаест локација у обе "банке" заузимају регистри специјалне намене – SFR (Special Function Register).

Стек чине осам тринаестобитних локација, а програмски бројач један тринаестобитни регистар.

Стек (Stack) је интерни део микроконтролера (као уосталом и W регистар) коме могу приступити једино инструкције скока и повратка из потпрограма или интерапт рутине (више о њој касније). Њега можете замислити као цев затворену са једне стране у коју редом убацујете кликере. Црвени, зелени, жути и плави. Очигледно, не можете извући зелени кликер уколико претходно не извучете плави па жути.
При наиласку на CALL инструкцију, адреса наредне инструкције се ставља на стек, и врши се измена PC. Микроконтролер скаче на потпрограм, и тамо наставља извршење програма све до инструкције повратка. По наиласку на инструкцију повратка, микроконтролер преузима адресу повратка са стека, и смешта је у PC. Даље извршење програма наставља се одатле. Међутим, стек није бесконачан. У њега се може уписати максимално 8 адреса за повратак (још мање уколико се користе интерапти). Након тога нове адресе повратка пребрисаће почетне (црвени па зелени кликер...), што ће проузроковати неправилан рад програма.

4.1 EEPROM меморија

Унутар PIC16F84 налази се 64 бајта EEPROM меморије (0x00–0x3F). За разлику од регистара који су обична RAM меморија, ова меморија задржава своје стање и након нестанка напона напајања микроконтролера. Из EEPROM меморије читамо податак тако да сетујемо RD (EECON1 регистар) бит који иницира пренос података с адресе која се налази у регистру EEADR у EEDATA регистар. За читање податка није потребно неко време као за упис па се тај податак може користити већ у следећој инструкцији.

Пример дела програма који врши читање података из EEPROM-а можемо видети на следећој слици (Слика 8):

```
bcf     STATUS,RP0      ; bank0 jep je EEADR на адреси 09h
movlw   0x00            ; адреса локације која се чита
movwf   EEADR           ; адреса се пребацује у EEADR
bsf     STATUS,RP0      ; bank1 jep je EECON1 на адреси 88h
bsf     EECON,RD        ; читање из EEPROM -а
bcf     STATUS,RP0      ; bank0 jep je EEDATA на адреси 08h
movf    EEDATA,W        ; W←EEDATA
```

Слика 8: Пример кода који врши читање из EEPROM-а

Да би уписали податак у EEPROM меморију треба прво уписати адресу жељене меморијске локације у EEADR регистар а податак у EEDATA регистар. Након тога сетујемо WR бит који покреће уписивање податка на жељену локацију. Након уписа WR бит ће бити ресетован, а EEIF сетован, што се може искористити за обраду прекида. Вредности 55h и AAh су први и други кључ који онемогућују да дође до случајног уписа у EEPROM. Те две вредности се уписују у EECON2, који служи само за прихватање ове две вредности и тиме спречи случајан упис. Програмске линије у доле наведеном примеру означене бројевима од 1. до 5. морају бити извршене тим редоследом и у правилним временским размацима, тако да је врло важно онемогућити прекиде за време извршавања тих инструкција. Након уписа прекиди се могу омогућити.
Пример дела програма који врши упис податка 0xEE у прву локацију EEPROM меморије.

```
    bcf     STATUS,RP0      ;bank0 jep je EEADR на адреси 09h
    movlw   0x00            ;адреса локације у коју се пише
    movwf   EEADR           ;адреса се пребацује у EEADR
    movlw   0xEE            ;уписујемо вредност 0xEE
    movwf   EEDATA          ;податак иде у EEDATA регистар
    bsf     STATUS,RP0      ;bank1 jep je EEADR на адреси 09h
    bcf     INTCON,GIE      ;сви прекиди се онемогућују
    bsf     EECON1,WREN     ;омогућује се упис
1.  movlw   55h
2.  movwf   EECON2          ;први кључ 55h →EECON2
3.  movlw   AAh
4.  movwf   EECON2          ;други кључ AAh→EECON2
5.  bsf     EECON1,WR       ;иницира упис
    bsf     INTCON,GIE      ;прекиди се омогућују
```

Слика 9: Део кода који врши упис податка 0xEE у прву локацију EEPROM меморије

5. Начини адресирања

Приликом извођења програма мора се знати где се налазе инструкције и операнди над којима ће се изводити операције. Начини на које се прибављају инструкције и одређени операнди за време извођења инструкцијског циклуса називају се начини адресирања. Локацијама у RAM меморији може се приступати директно или индиректно.

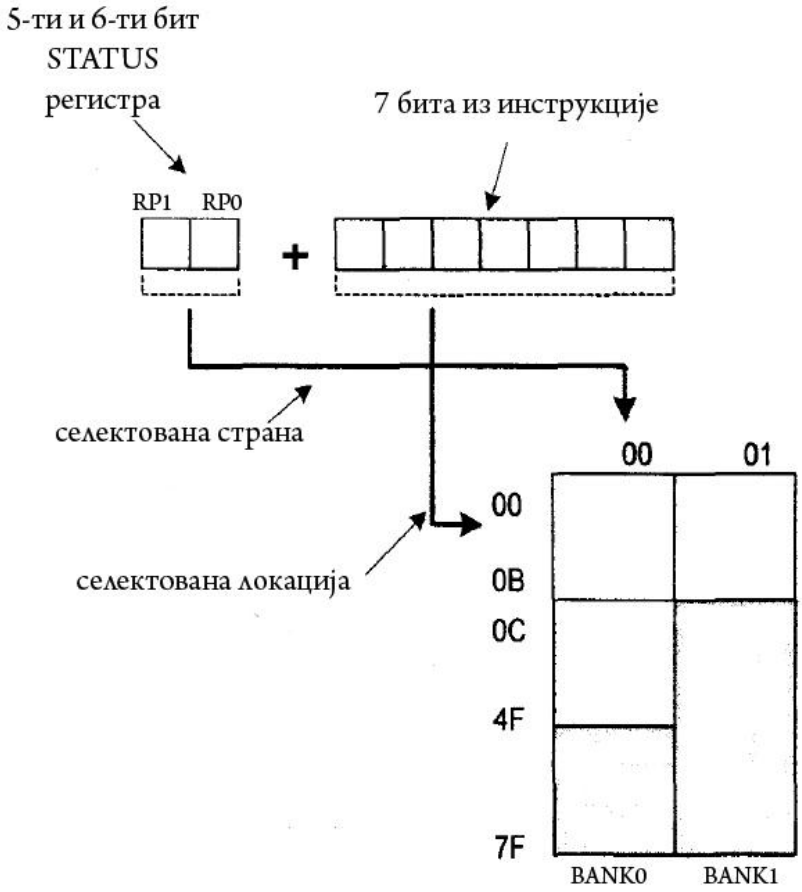
5.1 Директно адресирање

Директно адресирање иде преко 9-битне адресе која се добија повезивањем 7-ог бита директне адресе из инструкције и два бита (RP1,RP0) из регистра STATUS као што је приказано на слици (Слика 10).

Као пример директног адресирања може се узети сваки приступ SFR регистрима.

```
bsf     STATUS, RP0     ; банка 1
movlw   0xFF            ; w= 0xFF
movwf   TRISA           ; адреса TRISA регистра се узима из инструкције
                        ; movwf
```

Слика 10: Пример кода директног адресирања

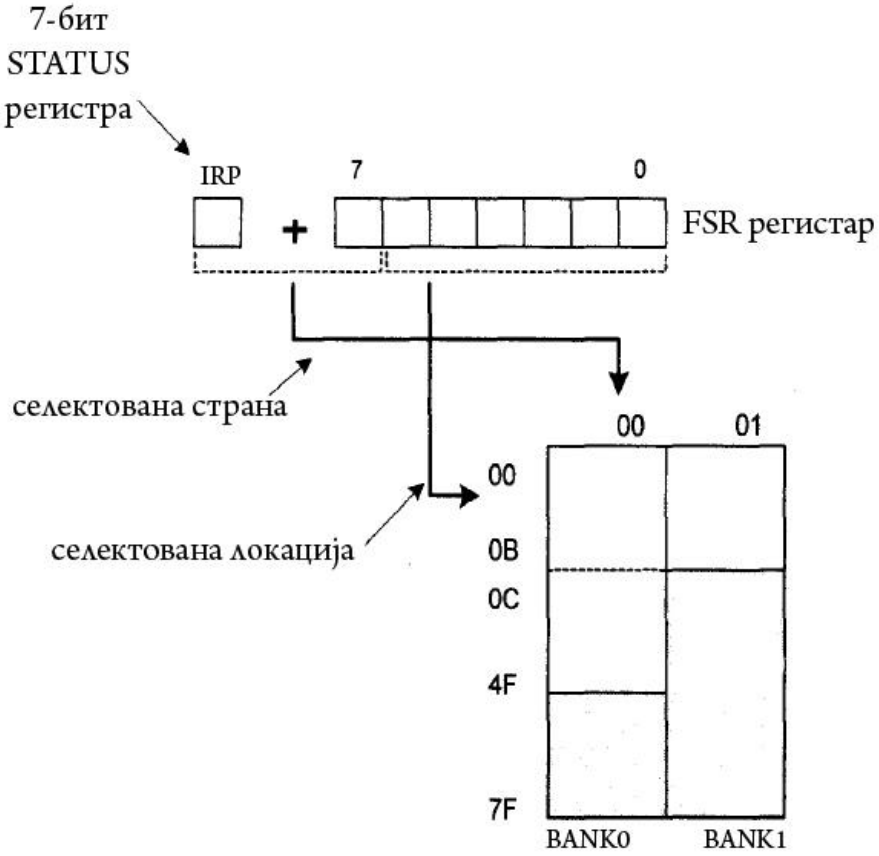


Слика 11: Директно адресирање *[1]*

5.2 Индиректно адресирање

Индиректно адресирање за разлику од директног, адресу не узима из инструкције већ је прави помоћу IRP бита STATUS регистра и FSR регистра. Адресираној локацији се приступа преко INDF регистра који у сттвари садржи адресу на коју FSR показује. Другим речима, било која инструкција која користи INDF као регистар у ствари приступа подацима на које показује FSR регистар. Нека на пример један регистар опште намене (GPR) на адреси 0Fh садржи вредност 20. Уписом вредности 0Fh у регистар FSR добијамо показивач на регистар на адреси 0Fh, а читањем из регистра INDF добијамо вредност 20, што значи да смо из првог регистра прочитали његову

вредност а да му нисмо директно приступили(већ преко FSR и INDF). Наизглед, овај начин адресирања нема никаквих предности над директним, али постоје одређене потребе при програмирању које се елегантно решавају једино индиректним адресирањем.



Слика 12: Индиректно адресирање *[1]*

Један од таквих примера је слање низа података серијском комуникацијом (Слика 13), рад са баферима и показивачима или брисање дела RAM меморије (16 локације).

```
                movlw  0x0C       ; иницијализација почетне адресе
                movwf  FSR        ; FSR показује на адресу 0x0C
                clrf   INDF       ; INDF= 0
PETLJA          incf   FSR        ; адреса = почетна адреса +1
                btfss  FSR,4      ; да ли су све локације избрисане?
                goto PETLJA       ; нису, поново пролази кроз петљу

CONTINUE
                       .
                       .          ; јесу, настави са програмом
                       .
```

Слика 13: Пример кода индиректног адресирања

Читање података из регистра INDF када је садржај FSR регистра једнак нули враћа вредност нула, а упис у њега резултира NOP (No Operation- нема операције) операцијом. *[1]*

6. Регистри

Регистар је RAM меморијска локација унутар микроконтролера у коју се може уписивати, са које се може читати, или обављати обе ове функције.
Следећа слика (Слика 5) приказује распоред регистара унутар PIC16F84 микроконтролера. Слика приказује адресе на којима је смештема RAM меморија (у облику осмобитних регистара) унутар микроконтролера, што ће Вам помоћи при разумевању њиховог адресирања. На слици можемо видети знак / који означава неимплементовану меморијску локацију (чита се као 0) и ознаку (1)- није физички регистар.

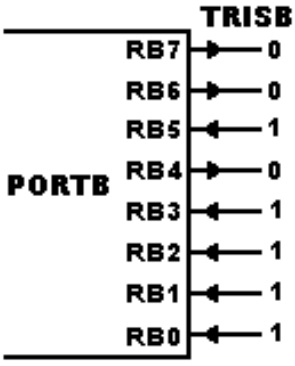
Адреса	BANK0	BANK1	Адреса
0x00	INDF(1)	INDF(1)	***0x80***
0x01	TMR0	OPTION_REG	***0x81***
0x02	PCL	PCL	***0x82***
0x03	STATUS	STATUS	***0x83***
0x04	FSR	FSR	***0x84***
0x05	PORTA	TRISA	***0x85***
0x06	PORTB	TRISB	***0x86***
0x07	/	/	***0x87***
0x08	EEDATA	EECON1	***0x88***
0x09	EEADR	EECON2(1)	***0x89***
0x0A	PCLATH	PCLATH	***0x8A***
0x0B	INTCON	INTCON	***0x8B***
0x0C ***0x4F***	68 регистара опште намене (Статичка RAM меморија)	Мапирани у BANK0	***0x8C*** ***0xCF***
0x50 . . . ***0x7F***	/	/	***0xD0*** . . . ***0xFF***

Слика 14: Распоред регистара унутар PIC16F84 микроконтролера *[6]*

Примећује се да су регистри подељени у две групе (Слика 14), означене са BANK0 и BANK1. BANK1 се углавном користи за конфигурисање тренутног хардверског стања микроконтролера. Углавном му је (за једноставније програме) довољно приступити само приликом почетне иницијализације микроконтролера. Како се најбитнији регистри (PORTA i PORTB) налазе у BANK0, програм ће углавном бити у њој. Следећу специфичност представљају првих 12 регистара у банкама (0x00 – 0x0B). Њима се директно управља радом микроконтролера што их чини повлашћеним (Special Function Registers - регистри специјалне намене), док је обична RAM меморија (General Purpose Registers - регистри опште намене) доступна у регистрима од 0x0C до 0x4F, а истовремено јој се може приступити и преко 0x8C до 0xCF адреса.

Може се узети пример рампи за пролазнике у супермаркетима или аутобуским станицама. Рампе пропуштају пролазнике само у једном смеру. Унутар микроконтролера TRIS (TRISA i TRISB) регистри извршавају ову функцију. Они се налазе на адресама 0x85 и 0x86, респективно. Да бисте програмирали жељени пин тако да буде улазни или излазни (пропушта пролазнике унутра или напоље), пошаљете 1 или 0 на његов бит у одговарајућем ТРИС регистру. При томе можете користити било који (бинарни, децимални или хексадецимални) формат бројева, међутим бинарним се сликовитије представља стање жељеног пина.

Сви RA (RA0 до RA4) и RB (RB0 до RB7) пинови микроконтролера имају двојаку фукнцију. Они могу бити улазни или излазни. Да ли ће бити улазни или излазни подешава се у одговарајућем TRIS регистру.



Пример: уколико се у TRISB налази следећа бинарна вредност: B'00101111' пинови ће бити конфигурисани као на слици (Слика 15). Примећујете да бит 7 (MSB – *Most Single Bit* - бит највеће тежине) одговара пину РБ7 и да се он приликом бинарног означавања налази са леве стране. Насупрот њему бит 0, бит најмање тежине (LSB – *Low Single Bit*) одговара пину RB0. Потпуно иста ситуација је и са ТРИСА регистром, са том разликом што су њему горњи битови неискоришћени (јер немају припадајућих пинова).

Слика 15: Пример подешавања који ће портови бити улазни или излазни

На слици (Слика 15) пинови су конфигурисани по следећем редоследу који можемо видети (Табела 1):

Пин	RB7	RB6	RB5	RB4	RB3	RB2	RB1	RB0
Број бита	7 (**MSB**)	6	5	4	3	2	1	0 (**LSB**)
Стање	0	0	1	0	1	1	1	1
Улаз/ Излаз	Излаз	Излаз	Улаз	Излаз	Улаз	Улаз	Улаз	Улаз

Табела 1: Пример конфигурисаних пинова

Како се TRISB налази у банци 1 (BANK1) потребно је пребацити се тамо. Да би се пребацили из банке у банку користи се бит 5 STATUS registra. За пребацивање из банке 0 у банку 1 потребно га је сетовати, а за враћање у банку 0, ресетовати.

W или **WREG** (*Working Register*- радни регистар) нама познатији као акумулатор је регистар опште намене у који можете сместити било коју бројчану вредност (у опсегу од .0 до .255) са којом желите радити. Након што сте придружили одређену вредност регистру W, можете му придружити неку другу вредност (опрез – тиме бисте аутоматски обрисали претходну) или запамћену вредност из њега преместити на неку другу меморијску локацију (регистар). За разлику од осталих регистара он нема своју адресу, већ је интегрисан у склопу самог микроконтролера.

6.1 Регистри са специјалним функцијама

Део RAM меморије одвојен је за ове регистре који служе за контролу рада. Пошто у инструкцијама за рад са RAM-ом има седам битова за адресирање локације, то значи да можемо да адресирамо 128 локација RAM-а, од 00h до 7Fh (од 0 до 127). Међутим, као што смо претходно видели, један део регистара налази се изнад тих адреса, па је питање како њих адресирати. За то су задужена два бита у STATUS регистру (то су битови 5 и 6) који чине битове 7 и 8 адресе, чиме добијамо могућност да адресирамо чак 512 локација. Пошто нам је довољно 8 битова (0-7), нећемо морати да користимо бит 8; он је остављен од стране произвођача за евентуална проширења и не препоручује се његово коришћење. Значи, када приступамо локацијама од 80h, потребно је једном инструкцијом сетовати бит 5 (зове се RP0), у супротном га треба ресетовати.

6.2 Опис регистара са специјалним функцијама

На адреси 00 налази се регистар са именом INDF; то није физички регистар већ се користи за индиректно адресирање у спрези са регистром SFR за индиректно адресирање, на следећим адресама налази се:
- 01 - TMR0 (слободни бројач), 8-битни бројач (сат) реалног времена,
- 02 - PCL, нижих 8 битова програмског бројача PCLATH + PCL = PC,
- 03 - STATUS, садржи битове који означавају стање аритметичко-логичке јединице (ALU), RESET статус, и битове за адресирање виших локација RAM-а,
- 04 - FSR, поинтер за индиректно адресирање RAM меморије,
- 05 - PORTA, то су, у ствари, пинови порта A,
- 06 - PORTB, пинови порта B,
- 07 - не користи се,
- 08 - EEDATA, регистар података за EEPROM,
- 09 - EEADR, регистар адресе за EEPROM,
- 0A - PCLATH, виших 5 битова програмског бројача PCLATH + PCL = PC,
- 0B - INTCON, регистар за контролу интерапта,
- 80 - INDF, исто као 00,
- 81 - OPTION, управљачки регистар за прескалер, спољни интерапт, TMR0 i pull- up - отпорнике на порту B,
- 82 - PCL, као 02,
- 83 - STATUS, као 03,
- 84 - FSR, као 04,
- 85 - TRISA, регистар за дефинисање смера (улаз или излаз) пинова на порту A,

- 86 - TRISB, регистар за дефинисање смера пинова на порту B,
- 87 - не користи се,
- 88 - EECON1, управљачки регистар за рад са EEPROM-ом,
- 89 - EECON2, други управљачки регистар (није физички регистар),
- 8A - PCLATH, као 0A,
- 8B - INTCON, као 0B.

Као што се види, неки регистри су дуплирани у меморијској мапи. *[7]*

Adresa	***Naziv***	***Bit 7***	***Bit 6***	***Bit 5***	***Bit 4***	***Bit 3***	***Bit 2***	***Bit 1***	***Bit 0***	Bank
0x00	INDF									obe
0x01	TMR0									0
0x02	PCL	PC	PC	PC	PC	PC	PC	PC	PC	obe
0x03	STATUS	IRP	RP1	RP0	$\overline{\textbf{TO}}$	$\overline{\textbf{PD}}$	Z	DC	C	obe
0x04	FSR									obe
0x05	PORTA	-	-	-	RA4/T0CKI	RA3	RA2	RA1	RA0	0
0x06	PORTB	RB7	RB6	RB5	RB4	RB3	RB2	RB1	RB0	0
0x07	/									/
0x08	EEDATA									0
0x09	EEADR									()0
0x0A	PCLATH	-	-	-	PC	PC	PC	PC	PC	obe
0x0B	INTCON	GIE	EEIE	T0IE	INTE	RBIE	T0IF	INTF	RBIF	obe
0x80	INDF									obe
0x81	OPTION	RBPU	INTEDG	T0CS	T0SE	PSA	PS2	PS1	PS0	1
0x82	PCL	PC	PC	PC	PC	PC	PC	PC	PC	obe
0x83	STATUS	IRP	RP1	RP0	$\overline{\textbf{TO}}$	$\overline{\textbf{PD}}$	Z	DC	C	obe
0x84	FSR									obe
0x85	TRISA	-	-	-						1
0x86	TRISB									1
0x87	/	-	-	-	-	-	-	-	-	/
0x88	EECON1	-	-	-	EEIF	WRERR	WREN	WR	RD	1
0x89	EECON2									1
0x8A	PCLATH	-	-	-	PC	PC	PC	PC	PC	obe
0x8B	INTCON	GIE	EEIE	T0IE	INTE	RBIE	T0IF	INTF	RBIF	obe

Табела 2: Регистри са специјалним функцијама и њихове адресе

6.3 Детаљан опис регистара са специјалним функцијама

INDF - Овај регистар, како је већ речено, и није физички регистар, већ само "канал" помоћу кога се врши индиректно адресирање RAM локација или математичког регистра (познатијег као "акумулатор") који се овде зове W регистар. Пошто не постоје посебне инструкције за индиректно адресирање, коришћење овог регистра заједно са FSR регистром представља једини начин за остваривање тог начина адресирања.

TMR0- Слободни бројач - Ово је бројач који може да броји или спољне импулсе или импулсе који долазе са осцилатора, али сваки четврти (ако је прескалер додељен WDT). Зашто баш сваки четврти? Свака инструкција (осим оних које мењају садржај PC-a) изврши се за 4 такта осцилатора, што значи да се при фреквенци од 4 MHz, инструкције извршавају брзином од 1 MIPS-a (MIPS=milion instruction per second). Ако је бит 5 (T0CS) у OPTION регистру ресетован, до бројача стижу импулси са осцилатора (f osc/4), а ако је сетован, до њега стижу импулси са RA4/T0CKI; у том случају бит 4 (T0SE) OPTION регистра одређује да ли ће стање бројача да се увећава на растућу или опадајућу ивицу. У микроконтролер је уграђен и један 8-битни прескалер који може да се додели или бројачу TMR0 или WDT-u (о томе одлучује бит 3 (PSA) OPTION регистра), а фактор дељења прескалера зависи и од тога коме је додељен, и од стања прва три бита (PS0, PS1, PS2) у OPTION регистру, што може да се види у следећој табели (Табела 3):

ПРЕСКАЛЕР			**ФАКТОР ДЕЉЕЊА**	
PS2	PS1	PS0	TMR0	WDT
0	0	0	1:2	1:1
0	0	1	1:4	1:2
0	1	0	1:8	1:4
0	1	1	1:16	1:8
1	0	0	1:32	1:16
1	0	1	1:64	1:32
1	1	0	1:128	1:64
1	1	1	1:256	1:128

Табела 3: Фактор дељења прескалера *[7]*

Прекорачење стања овог бројача (када са FFh прелази на 00h) може да изазове интерапт ако је дозвољен интерапт у глобалу (бит 7 (GIE) INTCON registra) и ако је омогућен интерапт на прекорачење стања TMP0 (bit 5 (T0IE) истог регистра) и управо та особина може да се искористи за мерење реалног времена.

PCL i PCLATH - Program counter (PC)- Програмски бројач је регистар који чува адресу следеће инструкције која треба да се изврши. С обзиром да је он ширине 13 битова (нама је за адресирање 1 Kb довољно 10 - и овде је произвођач предвидео каснија могућа проширења), подељен је на два дела од којих су првих 8 битова смештени у PCL регистар и помоћу кога може да се врши директан приступ нижем делу адресе. Овај регистар се најчешће користи за читање садржаја табела, али треба пазити да табела

буде смештена у блоку од 256 бајтова који адресира виши део PC адресе (PCH). Вишем делу програмске адресе (PCH) не може да се приступа директно, али може да се приступа регистру PCLATH (Program counter latch high), а стање овог регистра се аутоматски преписује у PCH када се у програму директно приступа PCL-u или када се извршавају GOTO и CALL инструкције. У случају директног приступа PCL-у, у PC се преноси свих 5 битова из PCLATH у PCH, а у случају GOTO и CALL инструкција преносе се само битови 3 и 4, а прва 3 се игноришу. Разлог за ово лежи у чињеници да у инструкцијама GOTO и CALL има места за 11 адресних битова који се преносе у PC (што значи да можемо да адресирамо 2K) па, једноставно, нема места за комплетну адресу. Обзиром да PIC16F84 има само 1 Kb програмске меморије, не морамо да водимо рачуна о овоме, сем ако произвођач не направи верзију овог MCU-а са више од 2Kb програмске меморије.

STATUS - Све битове овог регистра, сем битова 3 и 4, можемо директно да мењамо, мада остаје питање има ли смисла мењати битове који показују стање ALU јединице, јер може да се деси да их нека инструкција промени. Објаснићемо редом појединачне битове:

0: C, carry/inv. borrow - користи се код сабирања, одузимања или ротирања, а означава да ли је сабирање или одузимање могуће или је извршен пренос једног бита. Обратите пажњу да је при одузимању бит инвертован: сетован је ако је могуће одузимање без преноса, у супротном је ресетован. С обзиром да су сви регистри 8-мо битни, бит означава прекорачење у опсегу од 256.
Пример: ако саберемо бројеве 220 и 40, резултат ће бити 4 и биће сетован C бит.

1: DC (digit carry/inv. digit borrow) - као и C (бит 0), али означава пренос са бита 3 на бит 4.

2: Z, zero - ако је резултат неке аритметичке или логичке операције 0, овај бит ће бити сетован, у супротном ће бити ресетован.

3: inv. PD, power down - бит је сетован при укључењу микроконтролера, након извршења CLRWDT инструкције или после ресета MCU-а. Ресетује га SLEEP инструкција када микроконтролер прелази у POWER DOWN мод и тада је интерни осцилатор заустављен, а потрошња струје минимална. Директан упис у овај бит није могућ, а поновно сетовање овог бита обавља се интерно након ресета (спајањем MCLR на низак ниво или искључењем и поновним укључењем напајања), затим интераптом са RB0/INT пина, променом на порту B, завршетком циклуса уписа у EEPROM, као и истеком времена WDT-а (ако је WDT омогућен при програмирању).

4: inv. TO, time out - као и претходни бит, ни овај није могуће мењати директним уписом. Сетован је аутоматски након ресета и након инструкција CLRWDT и SLEEP. Ресетује га WDT када истекне време.

5, 6: RP0, RP1, избор групе регистара - као што је раније речено, инструкције за рад са RAM-ом имају само седам битова у адресном пољу, па је могуће адресирати само 127 бајтова. Ова два бита представљају дво-битно проширење те адресе, чиме добијемо адресни простор од 512 бајтова. За PIC16F84 потребан је само један од та два бита (RP0- бит 5) и то због регистара специјалне намене који се налазе на адресама од 80h до

8Bh. Значи, када приступамо тим регистрима морамо "ручно" инструкцијом у програму да сетујемо бит RP0, у противном, тај бит треба да ресетујемо. Бит RP1 је остављен од стране произвођача за евентуална будућа проширења овог MCU-а.

7: IRP, избор групе регистара - за њега важи исто као и за RP1, сем што се он користи за дефинисање бита 8 адресе код индиректног адресирања. За PIC16F84 није нам потребан и не треба да водимо рачуна о њему.

OPTION - Као што је већ речено, ово је управљачки регистар. Ево описа појединачних битова:

0, 1, 2: PS0, PS1, PS2,одређују фактор дељења прескалера (Табела 3).

3: PSA, одређује коме је додељен прескалер - ако је PSA=0 прескалер је додељен TMR0, у супротном је додељен WDT-у.

4: T0SE, избор ивице за окидање бројача TMR0 - ако је TMR0 конфигурисан да се окида импулсима са RA4/T0CKI пина, онда ће се, ако је овај бит=0, бројач окидати растућом ивицом, а ако је 1 онда ће се окидати опадајућом ивицом.

5: T0CS, избор извора такта за TMR0 - ако је постављен на лог. 0, на бројач се доводи интерни CLKOUT (једна четвртина фреквенце осцилатора), а ако је постављен на лог. 1, на бројач се доводе импулси са RA4/T0CKI пина. У оба случаја ће ићи и преко прескалера ако је прескалер додељен бројачу TMR0.

6: INTEDG, избор ивице сигнала за генерисање спољног интерапта - ако је овај интерапт омогућен, биће генерисан опадајућом ивицом када је овај бит=0, или растућом ако је овај бит=1.

7: inv. RBPU, pull-up otpornici na portu B - ако је овај бит сетован (=1) отпорници су искључени и улази су високо импедансни, а ако је ресетован, укључени су ови отпорници, али само на пиновима који су улазни.

FSR - У овај регистар се смешта адреса RAM локације којој хоћемо да приступимо индиректним адресирањем. Навешћемо један пример: у локацији 0xCh налази се вредност 23h, а у локацији 0xDh вредност 8Ah. Убацивањем броја 0xCh у FSR регистар и читањем INDF регистра очитаћемо вредност 23h. Ако увећамо FSR за један и поново очитамо регистар INDF, добићемо вредност 8Ah.

PORTA i PORTB - PORTA је петобитни, а PORTB је осмобитни регистар и они представљају, фактички, пут до самих пинова. Уписом у ове регистре мењамо стања на пиновима тих портова, наравно ако су одређени пинови дефинисани као излазни. У ствари, врши се упис у latch-еве тих регистара. Читањем ових регистара очитавамо стање директно на пиновима код оних који су дефинисани као улазни; код оних који су дефинисани као излазни, очитаће се стање које је последње било уписано. Свака наредба која мења стање излазних пинова је "read-modify-write" (прочитај-измени-упиши) типа, што значи да се прво врши очитавање стања, затим измена и поновни упис у регистар. Ово треба узети у обзир приликом оптерећивања излазних пинова јер, ако се неком пину присилно промени стање (нпр. ако излазни пин на коме је логичка

јединица оптеретимо са више од 20mA може доћи до пада напона на том пину), биће прочитано погрешно стање које ће затим бити уписано као ново и ето проблема. Такође треба обратити пажњу на ситуацију у којој две узастопне наредбе мењају стања два различита бита истог порта; под одређеним условима може да се деси (због "read-modify-write" принципа) да на једном пину не буде жељено стање. Произвођач препоручује да се у том случају између те две наредбе убаци једна NOP наредба. Пин RA4 води и до TMR0, што се одређује битом T0CS у OPTION регистру. Сваки пин порта B има уграђен слаби pull-up отпорник (отпорник који спаја пин и позитиван пол напајања) који се укључује ресетовањем бита inv.RBPU (бит 7) OPTION регистра. Ако се неки од пинова постави као излазни, аутоматски се тај pull-уп отпорник искључује. Пинови RB4 до RB7 имају уграђену "интерапт при промени стања" функцију и то само ако је пин дефинисан као улазни, што значи да промена логичког стања на неком од тих пинова може да изазове интерапт.

TRISA i TRISB - Стања битова ових регистара одређују да ли ће одређени пин на порту бити улазни или излазни. Постављање одређеног бита на 1 дефинише одговарајући бит као улазни, а постављање на 0 дефинише одговарајући бит као излазни. Приликом ресета сви пинови су постављени као улазни (сви битови су на лог. 1). Бит 0 регистра TRISA утиче на RA0, бит 1 на RA1 итд. Мењање стања битова 5, 6 и 7 нема смисла јер порт A има само 5 пинова. Бит 0 регистра TRISB утиче на RB0, бит 1 на RB1 и тако све до бита 7 који утиче на RB7.

EEDATA - При упису у EEPROM, у овај регистар се убацује податак који желимо да програмирамо, а при читању, у овом регистру се налази садржај прочитане адресе EEPROM-а.

EEADR - У овај регистар се уписује адреса са које желимо да читамо или у коју уписујемо у EEPROM.

EECON1 - Овај регистар се користи при упису и читању EEPROM-а и има пет битова:

0: RD, старт читања EEPROM-а - сетовањем овог бита стартује се секвенца читања EEPROM-а и то са адресе која је унета у EEADR, а прочитани податак ће се наћи у регистру EEDATA и тај податак може да се прочита већ у следећој инструкцији. Овако изгледа типична секвенца читања (Слика 16):

```
bcf STATUS, RP0      ; ресетује се RP0 јер је EEADR на адреси 09h
movlw adresa         ; припрема се адреса са које се чита
movwf EEADR          ; и ставља у EEADR регистар
bsf STATUS, RP0      ; сетује се RP0, јер је EECON1 на адреси 88h
bsf EECON1, RD       ; стартује се читање EEPROM-а
bcf STATUS, RP0      ; ресетује се RP0 јер је EEDATA на адреси 08h
movf EEDATA, W       ; пребацује се прочитани податак из EEDATA у W
                     ; W регистар (радни регистар)
```

Слика 16: Читање из EEPROM-а

Када се препис из EEPROM-а у EEDATA заврши, RD бит ће аутоматски бити ресетован; није га могуће ресетовати директним приступом.

1: WR, старт уписа у EEPROM - сетовањем овог бита стартује се секвенца уписа у EEPROM података смештеног у EEDATA на адресу смештену у EEADR. Секвенца уписа изгледа овако (Слика 17):

```
bcf     STATUS, RPO         ; ресетује се RP0 јер је EEADR на адреси 09h
movlw   adresa              ; препирема се адреса на коју се уписује
movwf   EEADR               ; и ставља у EEADR регистар
bsf     STATUS, RP0         ; сетује се RP0 јер су INTCON, EECON1 и EECON2
                            ; на адресама већим од 7Fh
bcf     INTCON, GIE         ; спречи интерапт (ако је омогућен)
bsf     EECON1, WREN        ; омогући упис у EEPROM
movlw   55h                 ; припреми први кључ
movwf   EECON2              ; и убаци га у регистар EECON2
movlw   0AAh                ; припреми други кључ
movwf   EECON2              ; и убаци га у EECON2
bsf     EECON1, WR          ; стартуј упис у EEPROM
bcf     EECON1, WREN        ; онемогући упис у EEPROM
    bsf     INTCON, GIE     ; дозволи интерапт (ако је потребно)
```

Слика 17: Упис у EEPROM-а

Секвенца мора да буде оваква, иначе може да дође до погрешног уписа или да уопште не дође до уписа. Кључеви 55h и AAh су предвиђени од стране произвођача и постоје (као и EECON2 регистар) да би се смањила могућност случајног уписа у EEPROM. Тајминзи морају да буду тачно такви и због тога се пре иницирања уписа забрањују интерапти; ово је потребно урадити само ако су у програму интерапти омогућени. Уписом нове вредности стара се аутоматски брише, а цела секвенца уписа траје 10 ms и када се једном стартује не може се зауставити и даље је потпуно независна. Док траје, може да се извршава неки други програм, а по завршетку биће генерисан интерапт ако је омогућен, или може да се тестира стање битова WR и EEIF (EECON1 регистар) - када се заврши упис, WR ће аутоматски бити ресетован, а EEIF сетован.

2: WREN, дозвола уписа - ако је овај бит сетован, упис у EEPROM је омогућен, а ако је ресетован није могуће чак ни сетовање бита WR. Ово је урађено због веће сигурности и препоручује се да се тај бит сетује непосредно пред упис и затим ресетује. Ресетовање је могуће чак и ако није истекло 10 ms, јер је, као што је речено, након иницирања уписа даљи ток независтан.

3: WRERR, грешка при упису - ако у току уписа у EEPROM дође до ресета (спуштањем MCLR на лог. 0 или истеком времена за WDT), сетоваће се овај бит. С обзиром да ресет не мења стање овог бита, могуће је његовим тестирањем при иницијализацији утврдити да ли је ресет прекинуо упис. Ако је бит ресетован то још увек не гарантује да је упис исправан, већ само да упис није прекинут ресетом.

4: EEIF, интерапт циклуса уписа - након завршеног уписа у EEPROM, овај бит се аутоматски сетује што може да се испита у програму како би се уверили да је упис готов. Пошто се ресет не обавља аутоматски, треба га након тога ресетовати програмски.

INTCON - Овај регистар управља интераптима (о њима ће бити речи нешто касније) и сваком биту може програмски да се промени или очита његово стање. Ово су његови битови:
0: RBIF, промена на улазима 4 до 7 порта B - када дође до промене стања на битовима 4 до 7 порта B, овај бит се аутоматски сетује. Потребно га је после евентуалне обраде интерапта програмски ресетовати.
1: INTF, дошло је до спољног интерапта - ако се на пину RB0/INT појави одговарајућа ивица импулса (дефинисана битом INTEDG у OPTION регистру), овај бит се аутоматски сетује. Потребно га је, као и RBIF, програмски ресетовати.
2: T0IF, прекорачење бројача TMR0 - када бројач TMR0 пређе са FFh на 00h (прекорачење бројача), овај бит се аутоматски сетује. Као и претходна два, после евентуалне обраде интерапта, треба га софтверски ресетовати.
3: RBIE - када се овај бит сетује, омогућен је интерапт на промену стања на пиновима RB4 до RB7, у противном до интерапта неће доћи.
4: INTE - када се овај бит сетује, омогућен је спољни интерапт са улаза RB0/INT.
5: T0IE - када је овај бит сетован, омогућен је интерапт на прекорачење бројача TMR0.
6: EEIE - када је овај бит сетован, омогућен је интерапт на крај секвенце уписа у EEPROM.
7: GIE, дозвољени интерапти у глобалу - ако је овај бит ресетован, не може да наступи ни један интерапт, а ако је сетован, омогућени су они интерапти који су појединачно омогућени.

Конфигурациони регистар - Овај регистар се налази у програмској меморији на адреси 2007h, али му се не може приступити у току извршавања програма јер се налази у посебном меморијском блоку за тест и конфигурисање (од 2000h до 3FFFh); овом регистру може да се приступи само приликом програмирања. Његов садржај може да се одреди у корисничком програму, али и пре самог програмирања микроконтролера; сваки програм за програмирање требало би да омогући промену стања његових битова. Следи опис појединачних битова:

0, 1: FOSC0, FOSC1 - ова два бита одређују врсту осцилатора (Табела 4).

bit 1	bit 0	oscilator
0	0	LP
0	1	XT
1	0	HS
1	1	RC

Табела 4: Дефинисање који се осцилатор користи

2: WDTE - ако је овај бит једнак нули, WDT је онемогућен, у супротном је омогућен (укључен WATCHDOG тајмер).

3: $\overline{\text{PWRTE}}$ - ако је овај бит једнак нули, Power-up Timer је искључен, у супротном је укључен.

4-13: CP - ако су ови битови сви на 0, меморија је заштићена од читања, у противном није. Једини начин да се ови битови промене са 0 на 1 (да се искључи заштита од читања) је брисање целокупне програмске меморије.

7. Осцилатори

Осцилатор је електрично коло које ствара излазни сигнал одређене фреквенције. Као што се види из претходне табеле, код PIC16F84 могу да се употребе 4 врсте осцилатора и то:

- LP (Low Power) представља кристални осцилатор са фреквенцом до 200 kHz и у овом режиму микроконтролер има најмању потрошњу (у односу на остале врсте осцилатора).
- XT се користи за осцилатор са кристалом/керамичким резонатором и то од 100 kHz до 4 MHz за кристал или 455 kHz до 4 MHz за керамички резонатор.
- за HS (High Speed) важи све као и за XT, али се користи за фреквенце преко 4 MHz.
- RC је варијанта осцилатора када нам није потребан прецизан такт и потребни су нам само један отпорник и један кондензатор. У овом случају, на OSC2 пину појављује се 1/4 фреквенце осцилатора. Могуће је користити и спољни осцилатор, у ком случају се PIC конфигурише за LP, XT или HS режим, а фреквенца се доводи на OSC1 пин. *[2]*

8. Ресет

Код PIC16F84 постоји неколико врста ресета:

- ресет при укључењу (POR, power-on reset),
- довођењем ножице inv.MCLR на низак логички ниво при нормалном раду,
- довођењем inv. MCLR на низак ниво за време SLEEP наредбе,
- WDT reset (watchdog tajmer) ,
- WDT буђење за време SLEEP наредбе.

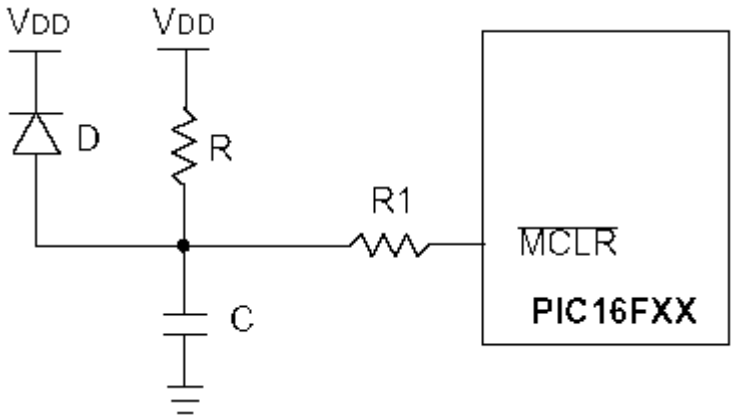
Ресет при укључењу (POR – Power-on Reset). Унутар микроконтролера налази се мало електрично коло које детектује пораст напона напајања до напона довољног за рад микроконтролера. У тренутку достизања номиналног напона, микроконтролер се ресетује. Овим се спречава рад микроконтролера при прениском напону напајања.
Уколико је потребно, може се укључити PWRTE (Power-up Timer Enable) конфигурациони бит који држи микроконтролер у стању ресета око 72mS од довођења напона напајања. Ово кашњење реализовано је интерним RC осцилатором независним од такта осцилатора. То је довољно да се напон напајања подигне на номиналну вредност и да осцилатор микроконтролера почне производити чисте непригушене осцилације. Након што се ово кашњење заврши, микроконтролер чека још 1024 такта осцилатора пре него што почне извршавати програм.

Код прва четири ресета PC (program counter) ће садржати адресу 0, што значи да ће програм почети да се извршава од почетка, од прве локације програмске меморије, док ће код WDT буђења за време SLEEP наредбе програм да настави извршењем следеће наредбе. Који ресет је извршен, може да се одреди тестирањем битова $\overline{TO}$ и $\overline{PD}$ према следећој табели (Табела 5):

-TO	-PD	врста ресета
1	1	ресет при укључењу напајања
1	1	довођење ниског нивоа на -MCLR при нормалном раду
1	0	довођење ниског нивоа на -MCLR за време SLEEP наредбе
0	1	WDT reset
0	0	WDT буђење за време SLEEP наредбе

Табела 5: Одређивање ресета који је извршен *[2]*

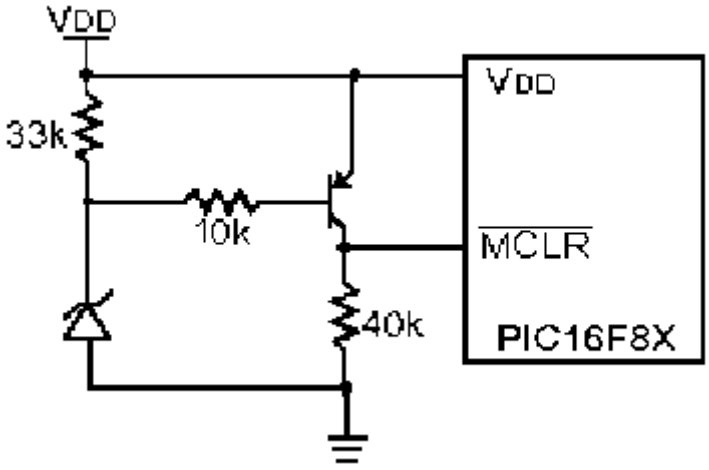
У већини случајева ће за ресет при укључењу бити довољно да се ножица inv.MCLR веже за плус напона напајања, јер се посебне функције (POR, PWRT, OST) брину за правилну и сигурну иницијализацију микроконтролера и то код LP, XT или HS осцилатора, док се код RC може користити само PWRT. Ако је пораст напона при укључењу мањи од потребног, неопходно је додати спољно ресет коло према следећој шеми (Слика 18):



Слика 18: Спољно ресет коло *[4]*

Проблем може настати и када напон падне испод дозвољене толеранције, а не падне на нулу, па се врати на нормалну вредност, јер у том случају микроконтролер нема потребан напон за нормалан рад одређено време и може да се деси да PC (program counter) узме неку прозвољну вредност и да програм "залута" и не ради онако како је замишљено. Та ситуација се зове brown-out и у том случају потребно је ресетовати микроконтролер, а једна од шема којима може да се изврши ресет изгледа као на слици (Слика 19).

Напон при којем се врши ресет једнак је напону зенер диоде увећаном за 0,7V (због пада напона на транзистору).



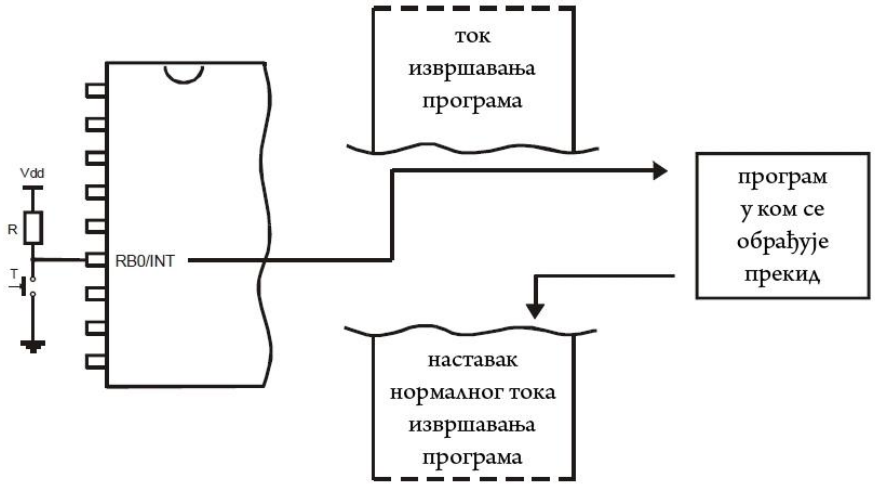
Слика 19: Једна од шема којима може да се изврши ресет у случају brown-out-a *[4]*

9. Прекиди (Интерапти)

Интерапт (прекид) је ситуација када због одређених услова програм прекида да извршава редован програм и почиње да извршава интерапт програм, а по његовом завршетку враћа се тамо где је прекинут.

Те услове смо већ напоменули, а то су:

1. спољни интерапт са RB0/INT пина,
2. прекорачење бројача TMR0,
3. промена стања на пиновима RB4 до RB7
4. завршетак уписа у интерни EEPROM.



Слика 20: Спољни интерапт

Сваки прекид мења ток извршавања програма, прекида га, а након извршења прекидног потпрограма наставља на истом месту (Слика 20). Контролни регистар прекида назива

се INTCON и налази се на адреси 0B. Његова улога је да омогући или забрани прекиде, а у случају да су забрањени, региструје појединачне захтеве прекида преко својих битова.

Да би се неки од тих интерапта реализовао потребно је да бит GIE буде сетован (дозвољени интерапти у глобалу) и да појединачни интерапти буду дозвољени. Када наступи интерапт, микроконтролер ресетује GIE бит, уписује садржај PC регистра на стек и "скаче" на адресу 04 јер је то фабрички предвиђена адреса (тзв. интерапт вектор). На ту адресу можемо да сместимо GOTO наредбу која ће да преусмери програм на неку другу адресу, или да од те адресе сместимо комплетну интерапт рутину. Пошто се код интерапта не чувају стања регистара, потребно је да то сами на почетку урадимо и да им садржаје сместимо на неку локацију у RAM-у, а на крају да им вратимо садржај. Довољно је да сачувамо садржаје W и STATUS регистра и то се ради овако (Слика 21):

```
movwf   w_temp          ; смести садржај W регистра у w_temp
swapf   STATUS, W       ; пребаци садржај STATUS регистра у W
movwf   status_temp     ; смести садржај W регистра у status_temp
                        ; овде се убацује интерапт рутина
                        ; а на крају се враћају регистри
swapf   status_temp, W  ; пребаци садржај status_temp u W регистар
movwf   STATUS          ; пребаци садржај W регистра у STATUS регистар
swapf   w_temp, f       ;
swapf   w_temp, W       ; пребаци садржај w_temp у W регистар
```

Слика 21: Пример кода за чување садржаја W и STATUS регистра

На крају интерапт рутине бит GIE се аутоматски сетује, са стека се узима адреса повратка и уписује у PC регистар, и микроконтролер наставља даљи рад управо тамо где је био прекинут. Аутоматско сетовање GIE бита може да буде и недостатак у једном случају, тј. може да се деси да ресетовање овог бита уопште и не буде извршено: ако интерапт наступи баш у моменту извршавања инструкције која ресетује GIE бит, прво ће инструкција бити завршена до краја (ресетоваће се GIE бит), а затим ће да крене извршавање интерапт рутине која ће на крају да аутоматски сетује бит GIE. Тако ће GIE бит бити сетован иако смо га управо пре интерапта ресетовали. Да би се заштитили од ове ситуације, произвођач препоручује да се након ресетовања GIE бита провери да ли је бит заиста ресетован и, ако није, да се операција понови.

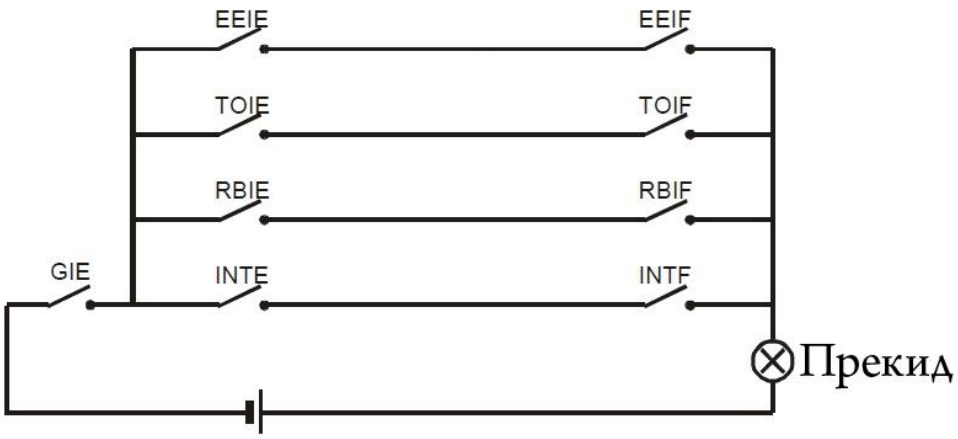
Спољни интерапт са RB0/INT пина - Ако је овај интерапт омогућен (сетован бит 4 (INTE) у INTCON регистру) и на RB0/INT пину се појави одговарајућа ивица (дефинисана INTEDG битом OPTION регистра) наступиће интерапт. На крају интерапт рутине потребно је у програму ресетовати INTF бит у INTCON регистру, како би наредни интерапт био могућ.

Интерпат на прекорачење бројача TMR0 - Ако је сетован бит 5 (T0IE) INTCON регистра и бројач TMP0 прекорачио стање, наступиће интерапт. На крају интерапт рутине потребно је ресетовати бит T0IF.

Интерапт на промену стања на RB4-RB7 - Ако је сетован бит 3 (RBIE) INTCON регистра и промени се стање на неком од улаза RB4 до RB7, наступиће интерапт. На крају интерапт рутине потребно је ресетовати бит RBIF.

Завршетак уписа у EEPROM - Када се заврши упис у EEPROM и ако је сетован бит 6 (EEIE) INTCON регистра наступиће интерапт. На крају интерапт рутине потребно је ресетовати бит EEIF у EECON1 регистру.

На слици **(**Слика 22**)** приказана је општа шема прекида. Помоћу GIE бита омогућујемо било који од четири могућа прекида. Користећи битове EEIE, TOIE, RBIF, INTE одређујемо врсте прекида, док нам битови са десне стране EEIF, TOIF, RBIF, INTF показују који се прекид догодио неком тренутку.



Слика 22: Општа шема прекида

10. Сет инструкција

Овај микроконтролер има само 35 наредби (инструкција) ширине 14 битова, што може да буде предност, али и мана. Предност зато што је лако запамтити мали број наредби, а мана зато што не постоји довољан комфор у раду са тако мало наредби. С друге стране, свака наредба заузима само једну меморијску локацију и извршава се у једном циклусу (који траје 4 такта осцилатора), сем оних који мењају садржај PC регистра (скокови, позиви потпрограма, повратак из потпрограма) које се извршавају за два циклуса.

MICROCHIP је поделио све наредбе у 3 групе:

а) оне које раде са бајтовима,
б) оне које раде са битовима,
ц) оне које раде са константама и врше контролу.

Оваква подела је условљена форматом наредби:

a) - битови 0-6 садрже адресу регистра (обележава се словом f од речи File)
- бит 7 одређује циљ, тј. одредиште (обележава се словом d од речи Destination)
- битови 8-13 садрже код (шифру) саме наредбе

b) - битови 0-6 садрже адресу регистра (f)
- битови 7-9 садрже адресу бита унутар регистра
- битови 10-13 садрже код наредбе

ц) овде постоје два случаја:
генерални - битови 0-7 садрже константу (број од 0 - 256), а остали код наредбе
CALL и GOTO - битови 0-10 садрже константу тј. адресу.

Словом f обележава се локација у RAM-у и, с обзиром да у инструкцији постоји само 7 битова за ту адресу, можемо да адресирамо само 128 локација. Зато је потребно обратити пажњу на којој адреси се налази регистар коме приступамо и одговарајућим постављањем бита RP0 усмерити се на одговарајућу банку регистара. Словом d је обележено одредиште, тј. место на које ће бити усмерен резултат операције. То није неки нови регистар већ једноставно означава да ли ће резултат бити смештен у акумулатор (W регистар) или у RAM чија адреса се налази у самој инструкцији, што у том случају знаћи да се операција извршава над неком RAM локацијом (f) и да се резултат смешта на исто место. Ако се за d стави 0, резултат се уписује у W, а за d=1 резултат се смешта у регистар (RAM локацију). У случају да се у наредби изостави ова нула или јединица, асемблер ће подразумевати да резултат иде у f (значи као да је уписана јединица) и истовремено ће генерисати упозорење. Боље је, а и прегледније, на почетку програма дефинисати неки симбол који ће заменити јединицу; често ћете на почетку програма сретати наредбу SAME EQU 1 (SAME има значење "исто") која ће ово урадити, па ће нпр. инструкција увећања изгледати овако: `INCF brojac, SAME` која значи да се садржај регистра "бројач" увећава за један и резултат смешта на исто место, тј. назад у регистар. Оваким начином добија се на прегледности. Акумулатор тј. W није смештен у RAM и једини је регистар који служи за пренос података између RAM локација и који врши математичко-логичке операције са RAM локацијама.

То конкретно значи да, ако хоћемо нпр. да упишемо неку вредност у RAM, морамо прво ту вредност да сместимо у W, а затим да је другом наредбом пребацимо из W у жељену RAM локацију (регистар).

Напомене за следећу табелу (Табела 6):

1-Уколико је излазни пин очитан па модификован од стране једне инструкције (нпр. MOVF PORTB,f), вредност која је узета за почетну биће вредност која је очитана са самих пинова. Ово може изазвати проблеме због RMW (Read Modify Write) циклуса. RMW циклус значи да микроконтролер приликом промене стања неког од битова одговарајучег регистра најпре очита цео регистар (свих 8 битова), затим изврши жељену операцију над њим, и на крају врати резултат у тај исти регистар.

2- Уколико се ова инструкција извршава над TMR0 регистром (и уколико је одредиште d=1 ако се користи), стање прескалера ће бити обрисано уколико је придружено тајмеру.

3- Уколико је тест исправан, инструкција ће се извршити за 2 инструкцијска циклуса.

Инструкција	*Опис*	*Цик.*	*Код*	*Флагови*	*Напомена*
Препис података					
MOVF f,d	Препиши f у d	1	001000 dfffffff	Z	1 2
MOVWF f	Препиши W у f	1	000000 1fffffff		
MOVLW k	Упиши константу у W	1	1100xx kkkkkkkk		
CLRF f	Упиши 0 у f	1	000001 1fffffff	Z	2
CLRW	Упиши 0 у W	1	000001 0xxxxxxx	Z	
SWAPF f,d	Препиши унакрсно ниблове из f у d	1	001110 dfffffff		1 2
Аритметичко логичке операције					
ADDWF f,d	Сабери W и f	1	000111 dfffffff	C DC Z	1 2
ADDLW k	Сабери W са константом	1	11111x kkkkkkkk	C DC Z	
SUBWF f,d	Одузми W од f	1	000010 dfffffff	C DC Z	1 2
SUBLW k	Одузми W од константе	1	11110x kkkkkkkk	C DC Z	
INCF f,d	Увећај f	1	001010 dfffffff	Z	1 2
DECF f,d	Смањи f	1	001011 dfffffff	Z	1 2
IORWF f,d	Логичко ILI W sa f	1	000100 dfffffff	Z	1 2
IORLW k	Логичко ILI W са константом	1	111000 kkkkkkkk	Z	
ANDWF f,d	Логичко I W са f	1	000101 dfffffff	Z	1 2
ANDLW k	Логичко I W са константом	1	11111x kkkkkkkk	Z	
XORWF f,d	Логичко искључиво ILI W са f	1	000110 dfffffff	Z	1 2
XORLW k	Логичко искључиво ILI W са констан.	1	111010 kkkkkkkk	Z	
COMF f,d	Инвертуј f	1	001001 dfffffff	Z	1 2
Операције са битовима					
BCF f,b	Ресетуј бит b у f	1	0100bb bfffffff		1 2
BSF f,b	Сетуј бит b у f	1	0101bb bfffffff		1 2
RLF f,d	Ротирај f налево кроз CARRY	1	001101 dfffffff	C	1 2
RRF f,d	Ротирај f надесно кроз CARRY	1	001100 dfffffff	C	1 2
Управљање током програма					
BTFSC f,b	Тест бита b у f, прескочи ако је бит =0	1 (2)	0110bb bfffffff		3
BTFSS f,b	Тест бита b у f, прескочи ако је бит =1	1 (2)	0111bb bfffffff		3
INCFSZ f,d	Увећај f, прескочи ако је = 0	1 (2)	001111 dfffffff		1 2 3
DECFSZ f,d	Смањи f, прескочи ако је = 0	1 (2)	001011 dfffffff		1 2 3
GOTO k	Иди на адресу	2	101kkk kkkkkkkk		
CALL k	Позови потпрограм	2	100kkk kkkkkkkk		
RETURN	Врати се из потпрограма	2	000000 00001000		
RETLW k	Врати се са константом у W	2	1101xx kkkkkkkk		
RETFIE	Врати се из интерапта	2	000000 00001001		
Остало					
NOP	Без операције	1	000000 0xx00000		
CLRWDT	Иницијализуј watchdog tajmer	1	000000 01100100	TO PD	
SLEEP	Прелазак у sleep мод	1	000000 01100011	TO PD	

Табела 6: Табела сета инструкција микроконтролера PIC16F84 *[6]*

Легенда (Табела 6):

f – адреса регистра; W – радни регистар; b – адреса бита унутар осмобитног регистра; k – константна (непроменљива) вредност или лабела; d – одредиште: d=0 резултат у W,

d=1 резултат у f; C – Carry flag STATUS регистра; DC – Digit Carry flag STATUS регистра; Z – Zero flag STATUS регистра; TO – Time out бит STATUS регистра
PD – Power down бит STATUS регистра.

11. Задатак

11.1 Поставка задатка

- ✓ Реализовати симулатор жмигаваца за аутомобил. Учестаност жмигаваца је 90/min. Задатак реализовати помоћу микроконтролера PIC16F84 на асемблеру. Уколико је сијалица неисправна учестаност осциловања је 120/min.

- PORTB.0 Лево предње светло
- PORTB.1 Симулација неисправности левог предњег светла
- PORTB.2 Лево задње светло
- PORTB.3 Симулација неисправности левог задњег светла
- PORTB.4 Десно предње светло
- PORTB.5 Симулација неисправности десног предњег светла
- PORTB.6 Десно задње светло
- PORTB.7 Симулација неисправности десног задњег светла

- PORTA.0 Укључење левог показивача
- PORTA.1 Укључење десног показивача
- PORTA.2 Укључење четири жмигавца
- PORTA.4 Сигнализација на инструмент табли

11.2 Решење задатка

```
#include  p16f84.inc          ; иницијализација процесора који ће се користити
        processor 16f84
        brojac1         equ     0x20            ;ознака за меморијску локацију бројача1(0.32)
        brojac2         equ     0x21            ;ознака за меморијску локацију бројача2(0.24)
        W_TEMP          equ     0x22
        STATUS_TEMP equ 0x23
        goto init
        org 4
        goto    prekid
        org     10                      ; програм почиње од позиције 5 у меморији
init                                    ;иницијализација
        clrf    PORTA
```

```
        clrf    PORTB
        bsf     STATUS,RP0      ; пребацити се у банку 1 меморије
        movlw   B'01111'        ; дефинисање да је порт А улазни
        movwf   TRISA
        movlw   B'10101010'     ; дефинисање да су В портови 1,3,5,7  улазни
        movwf   TRISB           ; вратити се из банке 1 меморије
        movlw   B'00000111'     ; поставити TMR0 на интерне импулсе
        movwf   OPTION_REG      ; поставити прескалер TMR0 на 255,
      bcf     STATUS,RP0        ; вратити се из банке 1 меморије
      movlw   0xD8              ; вредност која дефинише време између интерапта
      movwf   TMR0              ; поставити почетну вредност за TMR0
      movlw   B'10100000'       ; дозволити интерапте и
      movwf   INTCON            ; поставити интерапт TMR0
start                           ; лабела за почетак испитивања
      btfsc   PORTA,0           ; испитај да ли је притиснут тастер порта  A.0
        goto    levi            ; ако јесте-скочи на леви жмигавац-ако није прескочи
                                ; на следеће испитивање
      btfsc   PORTA,1           ; испитај да ли је притиснут тастер порта A.1
      goto    desni             ; ако јесте-скочи на десни жмигавац
      btfsc   PORTA,2           ; испитај да ли је притиснут тастер порта A.2
      goto    cetiri            ; ако јесте-скочи на четири жмигавца
      btfsc   PORTB,1           ; испитај да ли је притиснут тастер порта B.1
      goto    nelevi1           ; ако јесте-скочи на леви предњи жмигавац
      btfsc   PORTB,3           ; испитај да ли је притиснут тастер порта B.3
      goto    nelevi2           ; ако јесте-скочи на леви задњи
      btfsc   PORTB,5           ; испитај да ли је притиснут тастер порта B.5
      goto    nedesni1          ; ако јесте-скочи на десни предњи
      btfsc   PORTB,7           ; испитај да ли је притиснут тастер порта B.7
      goto    nedesni2          ; ако јесте-скочи на десни задњи
      goto    start             ; врати се на почетак испитивања

                    ;програми за кашњење
kasnjenje1
clrf    brojac1                 ;постављање бројача1 на 0
brojanje1                       ;потпрограм који врши бројање од 0 до 32
      movf    brojac1,0
      sublw   0x20              ; одузимање W од константе 32
      btfsc   STATUS,0
      goto    brojanje1
      return
kasnjenje2
      clrf    brojac2           ;постављање бројача2 на 0
brojanje2                       ;потпрограм који врши бројање од 0 до 24
```

```
        movf    brojac2,0
        sublw   0x18            ; одузимање W од константе 24
        btfsc   STATUS,0
        goto    brojanje2
        return
prekid                          ; интерапт иде од 216 до 255 => 39 пута и прави 0.01 sec
        bcf     INTCON,7
; снимање тренутног стања регистара
        MOVWF   W_TEMP
        SWAPF   STATUS, W
        MOVWF   STATUS_TEMP
          bcf   INTCON,2        ; обриши маркер прекорачења интерапта TMR0
        movlw   0xd8            ;постави 216 као почетну вредност
        movwf   TMR0            ; поставити почетну вредност за TMR0
        incf    brojac1,1       ;brojac1 повећај за 1
        incf    brojac2,1       ;brojac2 повећај за 1
        SWAPF   STATUS_TEMP, W
        MOVWF   STATUS
        SWAPF   W_TEMP, F
        SWAPF   W_TEMP, W
        bsf     INTCON,7        ;дозвољени интерапти у глобалу
        retfie                  ;повратак из интерапта
;крај дела везаног за кашњење
levi                    ; пале се лед диоде B0,B2 а гасе све остале, 90/60sec
        movlw   B'00000101'  ; у порт B упиши леви жмигавац
        movwf   PORTB        ;
        bsf     PORTA,4      ; у порт A4 упиши 1
        call    kasnjenje1
        movlw   B'00000000'  ; у порт B упиши нуле
        movwf   PORTB        ;
        bcf     PORTA,4      ; у порт A4 упиши нулу
        call    kasnjenje1
        goto    start        ; врати се на почетак испитивања
desni                   ; пале се лед диоде B4,6 и A4, 90/60sec
        movlw   B'01010000'  ; у порт B упиши B'01010000'
        movwf   PORTB        ;
        bsf     PORTA,4      ; у порт A4 упиши 1
        call    kasnjenje1
        movlw   B'00000000'  ; у порт B упиши нуле
        movwf   PORTB        ;
        bcf     PORTA,4      ; у порт A4 упиши нулу
        call    kasnjenje1
```

```
        goto    start           ; врати се на почетак испитивања
cetiri                  ; пали се А4 заједно са B0,2,4,6, 90/60sec
        movlw   B'01010101'     ; у порт B упиши B'01010101'
        movwf   PORTB           ;
        bsf     PORTA,4         ; ; у порт А4 упиши 1
        call    kasnjenje1
        movlw   B'00000000'     ; у порт B упиши нуле
        movwf   PORTB           ;
        bcf     PORTA,4         ; у порт А4 упиши нулу
        call    kasnjenje1
        goto    start           ; врати се на почетак испитивања
nelevi1                         ; пале се лед диоде B2 и А4 а гасе све остале, 120/60sec
        movlw   B'00000100'     ; у порт B упиши леви жмигавац
        movwf   PORTB           ;
        bsf     PORTA,4         ;  у порт А4 упиши 1
        call    kasnjenje2
        movlw   B'00000000'     ; у порт B упиши нуле
        movwf   PORTB           ;
        bcf     PORTA,4         ; у порт А4 упиши нулу
        call    kasnjenje2
        goto    start           ; врати се на почетак испитивања
nelevi2                         ; пале се лед диоде B0 и А4, 120/60sec
        movlw   B'00000001'     ; у порт B упиши леви жмигавац
        movwf   PORTB           ;
        bsf     PORTA,4         ; у порт А4 упиши 1
        call    kasnjenje2
        movlw   B'00000000'     ; у порт B упиши нуле
        movwf   PORTB           ;
        bcf     PORTA,4         ; у порт А4 упиши нулу
        call    kasnjenje2
        goto    start           ; врати се на почетак испитивања
nedesni1                        ; пале се лед диоде А4 иB4
        movlw   B'00010000'     ; у порт B упиши B'00010000'
        movwf   PORTB           ;
        bsf     PORTA,4         ; у порт А4 упиши 1
        call    kasnjenje2
        movlw   B'00000000'     ; у порт B упиши нуле
        movwf   PORTB           ;
        bcf     PORTA,4         ; у порт А4 упиши нулу
        call    kasnjenje2
        goto    start           ; врати се на почетак испитивања
nedesni2                        ; пале се лед diode А4 и B6, 120/60sec
        movlw   B'01000000'     ; у порт B упиши B'01000000'
```

```
movwf   PORTB                   ;
bsf     PORTA,4     ; у порт A4 упиши 1
call    kasnjenje2
movlw   B'00000000' ; у порт B упиши нуле
movwf   PORTB                   ;
bcf     PORTA,4     ; у порт A4 упиши нулу
call    kasnjenje2
goto    start                   ; врати се на почетак испитивања
__config B'11111111110001'  ; нема заштите кода, powerup дозвољено,
                                ; wd искључен, xt осцилатор
end                             ; крај програма
```

11.3 Прекид (Интерапт) у задатку

За одређивање колико ће интерапт трајати користимо следећу једначину:

$$K * T * 1\mu S = t$$

У мом случају најлакше је било узети t = 10 ms, па је:

$$K * T * 1\mu S = 10ms$$

K = 256 узимамо из табеле TMR0 за интерне импулсе 00000111 = 256

Добијамо следећу једначину:

$$T = \frac{10 * 10^{-3}}{256 * 10^{-6}} = 39$$

$$255 - 39 = 216$$

$$216 = D8$$

Тиме добијамо да нам TMR0 креће од D8 (216)- добијамо импулс интерапта од 0.01s Бројање се врши тако што се бројачи инкреметирају на сваких 0.01s потребан број пута.

Закључак

Данас не постоји више ниједан савременији апарат у којем микроконтролери не врше неку функцију. Захваљујући њиховој компактној и универзалној структури идеални су за мање или више интелигентне управљачке функције у свакодневици. Од њиховог настанка микроконтролери све више преузимају логичке задатке у управљачкој техници. Посебна карактеристика ових контролера је то што могу да функционишу и без додатних периферијских елемената. Код њих је на једном чипу интегрисано све: од меморије, преко централне процесорске јединице до улазно/излазних портова, а њихова предност лежи у томе што су мале, једноставне, лаке за програмирање и врло повољне цене. *[3]*

Референце

[1] Драган Андрић, Небојша Матић - PIC микроконтролери, Агенција АРС, Београд 2000.

[2] Стојан Трифуновић - Упутство за руковање PIC16F84, Сокобања 2008.

[3] Јелена Бубало, Радослав Илић, Видојко Вељковић - Програмирање PIC микроконтролера PIC16F84, Интер ХИТ, Београд 2001.

[4] Microchip Technology Inc, *PIC16F8X Datasheet*, http://www.microchip.com/ (октобар 2013.)

[5] http://www.automatika.rs/ (октобар 2013.)

[6] мр Аца Ђокић – Роботика и мехатроника http://www.sites.google.com/site/robotickragujevac034/home (октобар 2013.)

[7] http://sinel.freehostia.com/pic/pic_kontroleri.htm (октобар 2013.)

[8] http://www.ke4nyv.com (октобар 2013.)