MEĐIMURSKO VELEUČILIŠTE U ČAKOVCU

STRUČNI STUDIJ RAČUNARSTVA

DANIJEL KORENT

IMPLEMENTACIJA PROTOTIPNOG SIMULATORA MIKROUPRAVLJAČA MICROCHIP PIC16

ZAVRŠNI RAD

ČAKOVEC, 2015.

MEĐIMURSKO VELEUČILIŠTE U ČAKOVCU

STRUČNI STUDIJ RAČUNARSTVA

DANIJEL KORENT

IMPLEMENTACIJA PROTOTIPNOG SIMULATORA MIKROUPRAVLJAČA MICROCHIP PIC16

IMPLEMENTATION OF PROTOTYPE SIMULATOR OF MICROCONTROLLER MICROCHIP PIC16

ZAVRŠNI RAD

Mentor:

mr.sc. Mihael Kukec, dipl.ing.

ČAKOVEC, 2015.

Sažetak:

Zadatak ovog završnog rada je ostvariti prototip simulatora temeljenog na mikroupravljaču tvrtke Microchip arhitekture PIC16. Rad je podijeljen u nekoliko cjelina: povijest mikroupravljača i mikroupravljača familije PIC, opis arhitekture mikroupravljača familije PIC i konkretno mikroupravljača PIC16F84, opis implementacije prototipnog simulatora i parsera asemblerskog koda, te opis grafičkog sučelja koji koristi implementaciju prototipnog simulatora.

U početku rada opisana je općenita povijest mikroupravljača - kako i kada su se pojavili, razvoj tržišta, te povijest 8-bitnih mikroupravljača familije PIC. U nastavku rada navedene su osnovne značajke arhitekture PIC16 mikroupravljača, te prednosti i nedostaci ove arhitekture u odnosu na arhitekture drugih proizvođača mikroupravljača, poput mikroupravljača 8051 tvrtke Intel. Opisane su značajke instrukcijskog skupa, fizička organizacija memorija i sabirnica unutar mikroupravljača, te način na koji se pristupa radnoj memoriji, registrima i sklopovima. Također je opisana podjela familije 8-bitnih mikroupravljača PIC, te kojoj familiji pripadaju mikroupravljači tipa PIC16. Nakon osnovnih značajka arhitekture, opisan je mikroupravljač PIC16F84, njegove nožice, sklopovi kojima raspolaže, instrukcijski skup, te radna i programska memorija.

Implementacija prototipnog simulatora i parsera napravljena je korištenjem objektno orijentirane programske paradigme, te realizirana korištenjem programskog jezika Java. Simulator je implementiran na način da je funkcionalnost mikroupravljača podijeljena na niz podfunkcija mikroupravljača, te se svaka podfunkcija simulira u zasebnom razredu. Razredi su potom hijerarhijski spojeni u smislenu cjelinu korištenjem kompozicije objekata i nasljeđivanjem razreda. Opis implementacije je napravljen na način da je svaki razred pojedinačno opisan. Prvo je opisana funkcija koju taj razred obavlja i tko ga koristi, potom opis implementacije razreda. Na kraju opisa razreda nalazi se popis njegovih javnih metoda i atributa, te opis njihovih funkcija, i tip atributa ili u slučaju metode tip argumenta i povratne vrijednosti.

Zadnje poglavlje prije zaključka sadrži kratak opis grafičkog sučelja koji vizualno prikazuje unutarnja stanja mikroupravljača. Za implementaciju grafičkog sučelja korištena je Swing biblioteka. Poglavlje sadrži popis razreda i sučelja biblioteke Swing

koje implementacija grafičkog sučelje koristi, opis vizualnih elemenata grafičkog sučelja te na koji način je implementirano grafičko sučelje.

Ključne riječi: mikroupravljač, simulator, PIC16, PIC16F84, Java

Sadržaj

1. Uvod

Ubrzo nakon pojave prvih mikroprocesora pokazala se tržišna potreba za procesorski jednostavnijim uređajima koji će uz integriran procesor, na istom čipu također imat i integrirane sve potrebne periferije za potpuno funkcionalno računalo. Četrdeset godina kasnije u prosječnom kućanstvu nalazi se deset do dvadeset takvih uređaja, dok u modernim automobilima taj broj prelazi preko trideset komada. Trenutno 55% svih prodanih procesora čine 8-bitni mikroupravljači.

Zbog specifičnosti mikroupravljača praćenje tijeka izvršavanja programa, stanja registara i memorijskih lokacija nije moguće vršiti softverski na samom uređaju kao što je to slučaj sa stolnim računalima, nego se mora raditi na simulatoru ili koristeći dodatno sklopovlje fizičko povezano na posebne nožice mikroupravljača. Trenutno najpopularniji protokol ove namjene je tzv. JTAG (engl. *Joint Test Action Group*). No kod starijih ili jednostavnijih modela mikroupravljača često ne postoji ni opcija dodatnog sklopovlja, pa jedine opcije ostaju improvizacija sa hardverom za vizualnu indikaciju i komunikacijskim kanalima, ili korištenje simulatora. Koliko je važan simulator u razvoju softvera za mikroupravljače najbolje govore iznosi koje tvrtke plaćaju za korištenje komercijalnih simulatora.

U ovom radu će se istražiti mogućnost izrade jednostavnog simulatora arhitekture Microchip PIC16 u programskom jeziku Java. Analiziranjem arhitekture mikroupravljača pokušat će se odgovoriti kako oblikovati simulator koristeći objektno orijentiranu paradigmu i programski jezik Java.

2. Povijest mikroupravljača

Prvi mikroupravljač počinje se razvijati iste godine kada je napravljen prvi mikroprocesor, 1971. godine. Te godine je dovršen dizajn prvog mikroprocesora od strane Intela za potrebe tvrtke Busicom Corp koja proizvodi kalkulatore, te ga ujedno iste godine komercijalno nudi na tržištu. Radi se o svjetski poznatom 4-bitnom procesoru Intel 4004, koji je uz tri dodatna čipa omogućavao implementaciju potpuno funkcionalnog računala. Iste godine Texas Instruments proizvodi još napredniji integriraniji čip, 4-bitni mikroprocesor sa integriranom radnom memorijom (engl. *Random Access Memory - RAM*), stalnom memorijom (engl. Read-Only Memory) za pohranu programskog koda, te sa ulazno-izlaznim (engl. Input/Output - I/O) funkcijama, što ga čini potpuno funkcionalnim računalom u jednom čipu i prvim mikroupravljačem. Radi se o relativno nepoznatom TMS1000 kojeg Texas Instruments ne nudi na tržištu već počinje koristit u svojim vlastitim kalkulatorima. [1][2]

Tri godine kasnije Texas Instruments svoj mikroupravljač počinje nuditi kao zaseban proizvod, te time TMS1000 ujedno postaje i prvi komercijalno dostupan mikroupravljač. Zbog svoje izuzetno niske cijene i integriranosti TMS1000 zaživio je u potrošačkoj elektronici, te je prodano preko stotinu milijuna komada ovog čipa. Kao odgovor na tržištu se pojavljuju mikroupravljači drugih proizvođača, General Instrument 1975. godine izdaje prvi 8-bitni mikroupravljač i preteča današnje PIC serije - PIC1640, te se 1977. pojavljuju MC6801 tvrtke Motorola i 8048 tvrtke Intel. [3][4]

Ubrzo se tržište procesora specijalizira u dva odvojena smjera. Jedno je tržište procesora opće namjene koji se koriste u stolnim računalima i kompleksnijim sustavima, gdje je potrebna veća računalna snaga te za to vrijeme veliki adresni, poput 8-bitnog Zilog Z80, 16-bitnog Intel 8086 i 32-bitnog Motorola MC68000. Drugo je tržište jeftinijih čipova sa jednostavnijim procesorima koji teže prema što boljim I/O mogućnostima i što većoj integriranosti raznih periferija u jednom čipu, poput brojila (engl. *timer*), prekidnog sustava (engl. *interrupt controller*) ili sklopa za asinkronu komunikaciju (engl. *universal asynchronous receiver/transmitter - UART*), zbog čega postaju idealni u industriji potrošačke elektronike te kao pomoćni čipovi u kompleksnijim računalnim sustavima.

2.1 Povijest arhitekture PIC

General Instrument razvio je tzv. arhitekturu PIC (engl. *Programmable Interface Controller - PIC*), te PIC1640 i PIC1650 mikroupravljače prvenstveno kao pomoćne čipove za svoj 16-bitni mikroprocesor CP1600 koji je imao loše I/O performanse. Kako bi poboljšali ukupne

performanse sustava, PIC mikroupravljači su umjesto procesora obavljali I/O zadatke. Na zalasku CP1600 arhitekture General Instrument svoje poslovanje sa mikroprocesorima i mikroupravljačima izdvaja u zasebnu tvrtku naziva Microchip Technology, te je nekoliko godina kasnije prodaje. Nova tvrtka zaustavlja proizvodnju mikroprocesora i u potpunosti se fokusira na PIC mikroupravljače koje nadograđuje sa EPROM memorijom (engl. *Erasable Programmable Read-Only Memory - EPROM*) kako bi bili višekratno programibilni, te postiže uspjeh sa PIC16C5x serijom mikroupravljača. [4][5]

Microchip 1993. godine ponovo postiže velik uspjeh izdavanjem PIC16C84, prvog mikroupravljača sa EEPROM memorijom (engl. *Electrically Erasable Programmable Read-Only Memory - EEPROM*) i ISP programiranjem (engl. *In System Programming*). Dotadašnji mikroupravljači proizvodili su se u dvije varijante - sa PROM (engl. *Programmable Read-Only Memory*) i EPROM memorijom. U PROM memoriju je bilo moguće pisati samo jednom, zbog čega nisu bili pogodni za eksperimentiranje i razvoj. Za razliku od PROM, čip sa EPROM memorijom je bilo moguće izbrisati izlaganjem UV (engl. *Ultraviolet - UV*) zrakama, zbog čega je kućište imalo poseban kvarcni prozor koji je omogućavao izlaganje čipa UV zrakama. Iako ja ta verzija bila pogodnija za eksperimentiranje, i dalje je bila prilično nespretna zbog potrebe vađenja čipa iz strujnog kruga u sklop za programiranje, a također je bila i skupa zbog opreme za UV zračenje, te su i sami čipovi bili daleko skuplji zbog posebnog kućišta. EEPROM memorija omogućava da se sadržaj ROM-a briše pomoću el. struje, dok ISP omogućava da se čip može programirat unutar strujnog kruga (el. pločice) na kojoj se nalazi pomoću serijskog protokola, čime sklopovi za programiranje postaju daleko jednostavniji i jeftiniji, te nude daleko veću fleksibilnost prilikom razvoja. Nekoliko godina kasnije izdaje PIC16F877, mikroupravljač sa integriranim mehanizmima za praćenje rada i otklanjanje grešaka (engl. *In-Circuit Debugging - ICD*) koji omogućava praćenje rada programa u stvarnom vremenu i unutar planiranog hardverskog okruženja.

Sa cjenovno pristupačnim i jednostavnim sklopovima za programiranje te jednostavnom arhitekturom, PIC mikroupravljači ubrzo postaju popularni u profesionalnoj i hobi zajednici. Mikroupravljači PIC16x84 i PIC16F877 su jedni od najprodavanijih i najviše korištenih 8-bitnih mikroupravljača, dok je tvrtka Microchip postala vodeći proizvođač 8-bitnih mikroupravljača u svijetu, sa prodajom od preko milijardu komada godišnje. [6]

3. Značajke 8-bitnih mikroupravljača arhitekture PIC

Glavna odlika 8-bitne PIC familije mikroupravljača je njihova jednostavnost i niska cijena. PIC mikroupravljači imaju vrlo mali broj procesorskih instrukcija, samo 33 instrukcije kod osnovnih modela, do 83 instrukcije kod najnaprednijih modela. Uz mali instrukcijski skup, same instrukcije su također izrazito ortogonalne, tj. nema mnogo instrukcija koje mogu koristiti samo određene registre ili načine adresiranja, što dodatno pojednostavljuje učenje i pisanje programskog koda. Većina registara posebne namjene koji kontroliraju procesor i periferije memorijski mapirani su u procesorski adresni prostor, te im se pristupa jednako kao i ostalim registrima u memoriji, što omogućava jednostavnu kontrolu CPU-u i ostale periferije bez potrebe za dodatnim instrukcijama. Nabrojane značajke znatno pojednostavljuju i olakšavaju učenje ove arhitekture, što je ujedno i jedan od glavnih razloga njihove velike popularnosti. [7]

Mane PIC arhitekture su mala nelinearan pristup radnoj memoriji, tj. podjela memorije u tzv. „banke“, malen stog, te samo jedna prekidna rutina. Također jedna od mana tako minimalnog instrukcijskog skupa i jednostavne arhitekture je neprilagođenost kompajlerima viših programskih jezika, poput programskog jezika C. Ovaj nedostatak se u novijim PIC mikroupravljačima donekle ublažio proširenjima instrukcijskog skupa i dodatnim načinima adresiranja memorije.

3.1 Arhitektura PIC mikroupravljača

PIC arhitekturu obilježavaju sljedeće značajke:

- RISC arhitektura
- Harvard arhitektura
- Akumulatorska arhitektura
- Nelinearan pristup memoriji
- Izrazito ortogonalan instrukcijski skup

3.1.1 RISC arhitektura

PIC mikroupravljači koriste procesor koji ima reducirani instrukcijski set (engl. *Reduced Instruction Set Computing - RISC*), što znači da imaju mali broj jednostavnih instrukcija. Glavna prednost arhitekture RISC naspram arhitekture sa kompleksnim skupom instrukcija

(engl. *Complex Instruction Set Computing - CISC*) je što jednostavnije instrukcije omogućuju jednostavniji dizajn procesora i zahtijevaju manji broj taktova za izvršenje pojedinačne instrukcije. Ove značajke omogućile su da se strojni ciklus izvršava u 4 radna takt-a signala vremenskog vođenja, što je za to vrijeme bilo brzo, od čega se instrukcije grananja izvršavaju u dva strojna ciklusa, dok se sve ostale instrukcije izvršavaju u jednom strojnom ciklusu.

Uslijed ovih značajki, PIC mikroupravljači su u trenutku pojave na tržištu bili iznimno brzi naspram ostale 8-bitne konkurencije. Prvi službeni mikroupravljač arhitekture PIC (PIC16C54) je imao maksimalan radni takt od 40MHz, vrlo visok radni takt čak i za tadašnje procesore za stolna računala, što mu je omogućavalo postizanje iznimno niskih latencija odaziva. Uz četiri radna takta po strojnom ciklusu, ovaj mikroupravljač je postizao brzine do teoretskih 10 MIPS-a (milijuna instrukcija u sekundi). Za usporedbu tadašnji najveći konkurenti, Motorola 68HC11 postiže maksimalno 5Mhz sa svojim CISC procesorom, dok je Intelova 8051 CISC arhitektura radila na 12MHz, sa jednim strojnim ciklusom na 12 radna takta. Time se postizala brzina do teoretskih 1MIPS, dok se u praksi radilo o oko 0.6 MIPS-a zbog velikog broja instrukcija koje koriste dva strojna ciklusa, a neke čak i četiri strojna ciklusa. No treba imati na umu da ove broje nisu direktno usporedive, jer je u mnogo slučajeva potrebno izvršiti od nekoliko pa cak do desetak PIC instrukcija za obavljanje zadaće koja se na 8051 arhitekturi može obaviti jednom instrukcijom. Brzinska prednost PIC arhitekture izgubila se pojavom mnogih derivata 8051 arhitekture sa jednim radnim taktom po bajtu instrukcije, kao i pojavom Atmelove AVR arhitekture koja radi na jednom radnom taktu po strojnom ciklusu.

3.1.2 Harvard arhitektura

Kod harvard arhitekture programska i radna memorija fizički su odvojene sa vlastitim adresnim i podatkovnim linijama. Konkretno kod arhitekture PIC, programska i radna memorija nalaze se na potpuno odvojenim sabirnicama, koje su neovisne jedna o drugoj.

Ovaj odabir arhitekture omogućava dvije bitne stvari za ostale značajke PIC arhitekture:

- Programskoj i radnoj memoriji moguće je pristupiti istodobno
- Zbog odvojenih sabirnica programska i radna memorija mogu biti različite širine

Istovremen pristup programskoj i radnoj memoriji iskorišten je implementacijom dvostupanjskog instrukcijskog cjevovoda (engl. *instruction pipelining*). Dvostupanjski cjevovod omogućuje da se izvršavanje trenutne instrukcije i dohvat iduće izvode paralelno, u prvom stupnju se obavlja dohvaćanje iduće instrukcije, dok se u drugom stupnju istodobno

izvodi izvršavanje trenutne instrukcije. Ovo ubrzava izvođenje svih instrukcija koje ne mijenjaju programsko brojilo. Promjenom programskog brojila dohvaćena instrukcija u prvom stupnju cjevovoda više nije valjana, te se u idućem strojnom ciklusu ne izvršava već se samo dohvaća nova instrukcija. [8]

Kod širene sabirnice programske memorije dolazi do još jedne bitne razlike naspram 8051 arhitekture. Iako obje arhitekture koriste instrukcije duže od svoje podatkovne sabirnice, PIC radi samo jedno čitanje programske memorije po instrukciji, dok 8051 mora raditi i do tri čitanja po instrukciji. Kao i većini CISC arhitektura, instrukcije u arhitekturi 8051 variraju dužinom, od 8 bitova pa sve do 24 bitova dužine, ali se prenose sabirnicom fiksne širine, u slučaju arhitekture 8051 dužina iznosi samo 8 bitova. Za razliku od toga, u arhitekturi PIC sve instrukcije su jednake širine (12, 14 ili 16 bitova), te programska memorija i sabirnica imaju isti broj bitova kao i instrukcije, što omogućava da se cijela instrukcija uvijek prenese u jednom čitanju. Ove razlike u arhitekturi nisu rezultat dobrih ili loših inženjerskih odluka, nego su rezultat potpuno različitih ciljeva u mogućnostima i namjeni sklopa. Naime, cilj arhitekture PIC je da bude što jednostavnija kako bi proizvodnja bila što jeftinija, te jednaka dužina svih instrukcija i samo nekoliko vrsta formata instrukcija omogućuju vrlo jednostavnu implementaciju upravljačke jedinice mikroupravljača. Sa druge strane arhitektura 8051 dizajnirana je da bude što svestranija, te između ostalog omogućava dohvaćanje programskog koda sa vanjske memorijske jedinice kojima je u to vrijeme standardna širina podatkovne sabirnice bila 8 bitova. Budući da je interna programska sabirnica direktno spojena na vanjsku sabirnicu, interna sabirnica je morala biti jednake dužine kao i vanjska. U slučaju različitih veličina bilo bi potrebno posebno sklopovlje za dohvat instrukcije za svaku od sabirnica. [9]

3.1.3 Akumulatorska arhitektura

Kod svih aritmetičkih i logičkih operacija koje se izvode između dva operanda, implicitno se podrazumijeva da je jedan od operanda uvijek radni registar "W". Drugi operand može biti dvije vrste, konstanta "k" koji se u dokumentima naziva "literal" ili vrijednost sa memorijske adrese "f". Rezultat operacije se uvijek sprema u jedan od dva korištena operanda, ili u radni registar "W" ili u memorijsku lokaciju "f" ukoliko se adresiralo radnu memoriju. Akumulatorsku arhitekturu također koristi i 8051 arhitektura.

U usporedbi sa drugim arhitekturama, prednost je što omogućava relativno jednostavnu fizičku izvedbu procesora, što znači i nižu cijenu konačnog uređaja. Prednost drugih arhitektura je što dva ulazna operanda i izlazni operand mogu biti bilo koji od seta registara

poput registar-registar arhitekture (npr. MIPS i ARM), ili bilo koji registar i memorijska lokacija poput registar-memory arhitekture (npr. x86). Mana ove arhitekture je što često potrebno izvršiti nekoliko instrukcija da se izvede operacija koju register-register i register-memory arhitekture mogu izvršiti jednom instrukcijom.

3.1.4 Nelinearan pristup memoriji

Jedna od mana PIC mikroupravljača je što nemaju linearni pristup memoriji, već je memorija podijeljena u tzv. "banke", te se moraju koristi određeni bitovi u registrima posebne namjene da bi se mogao adresirati cijeli adresni prostor radne ili programske memorije. Prilikom direktnog pristupa radnoj memoriji, memorijska adresa je ukodirana u instrukciju koja je uvijek fiksne dužine. Npr. kod PIC mikroupravljača sa 12-bitnim instrukcijama, aritmetičko-logičke instrukcije imaju 5 bitova namijenjeno za adresu radne memorije. To znači da instrukcija može direktno adresirati samo 32 memorijske adrese. Kako bi se moglo adresirati više memorije, koristi se poseban registar koji kontrolira gornje bitove memorijske adrese prilikom pristupa memoriji. Kod programiranja treba obratiti posebnu pozornost na podjelu memorije jer nepravilno korištenje može uzrokovati greške koje je vrlo teško pronaći. Radna memorija PIC arhitekture je podijeljena na područje registara posebne namjene (engl. *Special Function Registers - SFR*) i te područje općih registara (engl. *General Purpose Registers - GPR*). U području posebnih registara mapirani su specijalni registri procesora i periferija. Neovisno o kojem području radne memorije se radi, podatku se pristupa istom vrstom instrukcija na jednak način.

PIC arhitektura također podržava i indirektno adresiranje, te adresiranje relativno na programski brojač. Kod naprednijih modela mikroupravljača indirektno adresiranje ima linearan pristup radnoj i programskoj memoriji.

3.1.5 Izrazito ortogonalni instrukcijski skup

Ortogonalni instrukcijski skup je onaj kod kojeg nema posebnih instrukcija koje mogu koristiti samo određene registre ili načine adresiranja. Neortogonalni instrukcijski skup je teže naučit i koristit jer od programera zahtjeva da pamti i pazi koje operacije može izvršiti nad kojim registrima. Primjer neortogonalnog instrukcijskog skupa je 8051 arhitektura.

PIC arhitektura postiže visoku ortogonalnost zahvaljujući činjenici da koristi samo jedan radni registar W, koji se implicitno podrazumijeva u instrukcijama, dok su skoro svi ostali specijalni registri mapirani u adresni prostor radne memorije, te im se pristupa isključivo putem radne memorije, sa istim instrukcijama kojima se pristupa i ostalim memorijskim

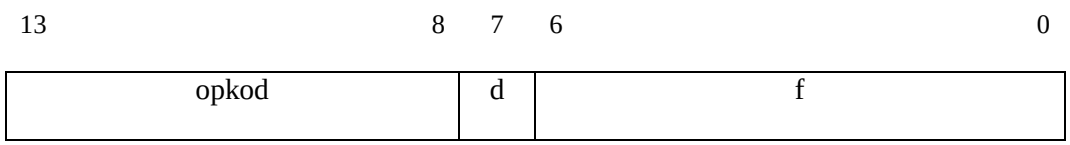
lokacijama. Iznimke su instrukcija WDT koja jedina ima pristup sigurnosnom brojilu (engl. *watchdog timer*), te instrukcija SLEEP koja mikroupravljač postavlja u stanje spavanja.

Instrukcijski skup PIC16 mikroupravljača iznimno je jednostavan i simetričan, te se prema formatu može podijeliti u četiri generalne kategorije:

- Instrukcije za 8-bitne operacije
- Instrukcije za 1-bitne operacije
- Instrukcije sa 8-bitnim neposrednim vrijednostima
- Instrukcije za bezuvjetni skok

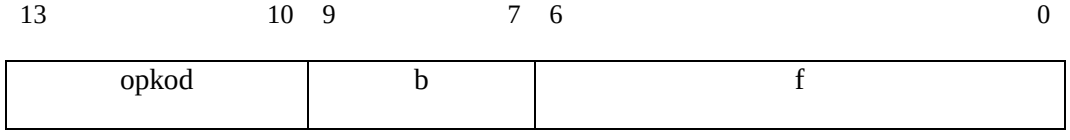
Instrukcije koje izvršavaju 8-bitne operacije imaju tri polja:

- opkod (6b) – određuje vrstu instrukcije
- f (7b)– ovdje je zapisana 7-bitna adresa radne memorije
- d (1b) – određuje da li će se rezultat operacije zapisati u registar w ili u lokaciju f



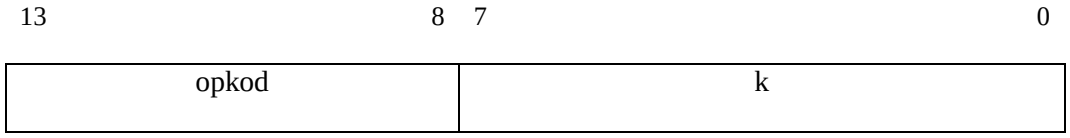
Instrukcije za 1-bitne operacije imaju također tri polja:

- opkod (4b) – određuje vrstu instrukcije
- f (7b) – ovdje je zapisana 7-bitna adresa radne memorije
- b(3b) – ovdje je zapisana 3-bitna adresa bita kojem se pristupa



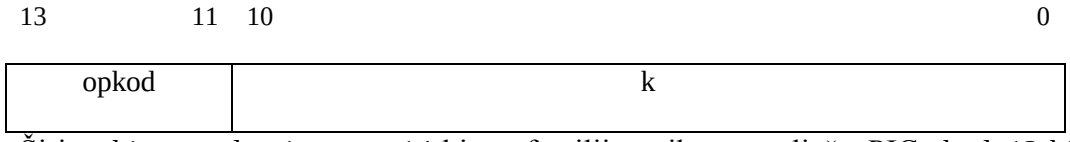
Instrukcije sa 8-bitnim neposrednim vrijednostima imaju dva polja

- opkod (6b) - određuje vrstu instrukcije
- k (8b) – ovdje je zapisana 8-bitna vrijednost



Istrukcije za bezuvjetni skok

- opkod (3b) - određuje vrstu instrukcije
- k (11b) – ovdje je zapisana adresa skoka



Širina bitova odnosi se na 14-bitnu familiju mikroupravljača PIC, kod 12-bitne familije mikroupravljača polja f su kraća za dva bita, polje k kod skokova dva do tri bita kraće.

3.2 Podjela familije 8-bitnih mikroupravljača PIC

Microchip svoj skup proizvoda kategorizira prema broju nožica, koji je naznačen u samom nazivu mikroupravljača, i prema arhitekturi jezgre. [10][11]

Prema broju nožica, Microchip svoje mikroupravljače dijeli u četiri kategorije:

- PIC10 - 6 nožica
- PIC12 - 8 nožica
- PIC16 - 14-64 nožica
- PIC18 - 18-100 nožica

Prema arhitekturi se dijelu također u četiri kategorije:

- Osnovna familija proizvoda – 12 bitne instrukcije
- Srednja familija proizvoda – 14 bitne instrukcije
- Nadograđena srednja familija proizvoda – 14 bitne instrukcije
- Familija PIC18 – 16 bitne instrukcije (ali i dalje 8-bitni CPU).

3.2.1 Osnovna familija proizvoda

Osnovna familija proizvoda je prva familija mikroupravljača pod službenom markom "PIC", nastala na temelju mikroupravljača PIC1640 tvrtke General Instruments. To je ujedno najjednostavnija Microchipova porodica, namijenjena za manje zahtjevne zadaće i niskobudžetno tržište. Ova arhitektura koristi 12-bitne instrukcije.

Ovoj porodici pripadaju mikroupravljači oznaka: PIC10F2xx, PIC12F5xx, PIC16Fxx

3.2.2 Srednja familija proizvoda

Arhitektura srednje familije koristi 14-bitne instrukcije te ima vrlo sličan instrukcijski skup kao arhitektura osnovne familije, uz nekolicinu novih instrukcija, te većim adresnim opsegom samih instrukcija. Konkretno instrukcije koje adresiraju „file“ registre sada koriste 7-bitne memorijske adrese, naspram 5-bitnih adresa u arhitekturi osnovne familije, što znači da je moguće direktno adresiranje 128 memorijskih lokacija. Također GOTO i CALL instrukcije mogu primati 11-bitne adrese, dok u arhitekturi osnovne familije instrukcija GOTO prima 9-

bitne adrese, a CALL 8-bitne. Uz dodatne opcije periferija, najvažnija novost je mogućnost vanjskog prekida i povećanje stog memorije sa 2 na 8 nivoa. Vanjski prekid podržava jednu prekidnu rutinu (engl. *Interrupt rutine*) sa nekoliko različitih izvora.

Ovoj porodici pripadaju mikroupravljači oznaka:

PIC10F3xx, PIC12F6xx, PIC16F6xx, PIC16F7xx, PIC16F8xx, PIC16F9xx

3.2.3 Nadograđena srednja familija proizvoda

Ova arhitektura uvodi prve veće promjenu u instrukcijskom skupu ali zadržava kompatibilnost za mid-core arhitekturom. Uz postojeće instrukcije dodaje 14 novih instrukcija. Nove instrukcije omogućuju bolju optimiziranost za više programske jezike, poput jezika C, te ujedno omogućuju bolju gustoću programskog koda. Jedna od važnijih novosti je automatska pohrana važnih registara prilikom prekida te povrat vrijednosti nakon završetka rutine (engl. *context save*). Također je omogućen indirektni linearni pristup programskoj i radnoj memoriji pomoću dva 8-bitna adresna registra.

Ovoj porodici pripadaju mikroupravljači oznaka: PIC12F1xxx iPIC16F1xxx

3.2.4 Familija mikroupravljača PIC18

Porodica sa najboljim performansama i najboljom optimiziranošću za više programske jezike. Od najvažnijih novina izdvaja se više prekidnih rutina sa prioritetima, 8-bitni hardverski množitelj, indirektni linearni pristup memoriji sa pojedinačnim 12-bitnim adresnim registrom, ograničeni direktni linearni pristup memoriji (jedna instrukcija za prijenos podataka), te softverski pristup stogu.

Ovoj porodici pripadaju samo mikroupravljači oznake PIC18Fx

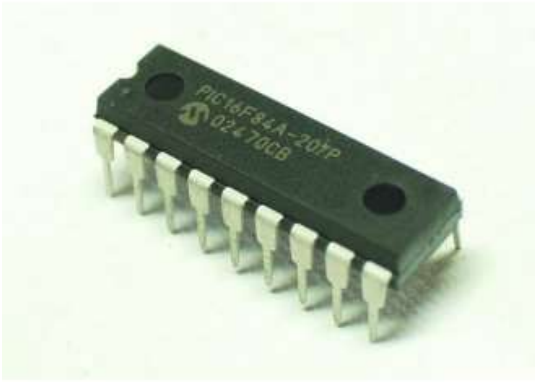
	Baseline Architecture	Mid-Range Architecture	Enhanced Mid-Range Architecture
Pin Count	6-40	8-64	8-64
Interrupts	No	Single interrupt capability	Single interrupt capability with hardware context save
Performance	5 MIPS	5 MIPS	8 MIPS
Instructions	33, 12-bit	35, 14-bit	49, 14-bit
Program Memory	Up to 3 KB	Up to 14 KB	Up to 28 KB
Data Memory	Up to 138 Bytes	Up to 368 Bytes	Up to 1,5 KB
Hardware Stack	2 level	8 level	16 level
Features	▪ Comparator ▪ 8-bit ADC ▪ Data Memory ▪ Internal Oscillator	In addition to Baseline: ▪ SPI/I²C™ ▪ UART ▪ PWMs ▪ LCD ▪ 10-bit ADC ▪ Op Amp	In addition to Mid-Range: ▪ Multiple Communication Peripherals ▪ Linear Programming Space ▪ PWMs with Independent Time Base
Highlights	Lowest cost in the smallest form factor	Optimal cost to performance ratio	Cost effective with more performance and memory

Slika 1. Usporedba PIC arhitektura

Izvor: http://www.microchip.com

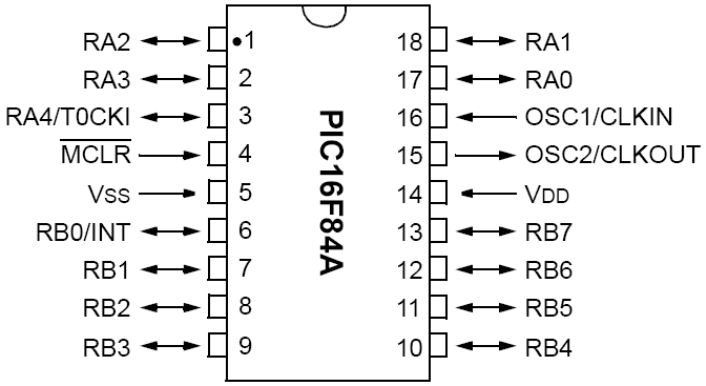
4. Mikroupravljač PIC16F84A

Kao referenca za oblikovanje aplikacije odabran je mikroupravljač PIC16F84A. Ovaj model pripada sredini skupa familija (engl. *mid-range*) sa 35 instrukcija, te je unutar te familije jedan od najjednostavnijih modela. Maksimalni radni takt mu je 20MHz, ima 1024x14bit programske memorije, 68B radne memorije, te 64B EEPROM memorije. Podržava prekide sa jednom prekidnom rutinom i četiri moguća izvora, te ima ugrađen 8-bitni brojač. Također ima ugrađen hardverski stog koji podržava do osam adresa, te sigurnosno brojilo (engl. *watchdog timer*) koji pruža zaštitu od beskonačnih petlja, ukoliko brojilo odbroji do nule procesor se resetira. [12]



Slika 2. PIC16F84A u DIP pakiranju

Izvor: http://www.bits4bots.com



Slika 3. Raspored nožica na PIC16F84A

Izvor: http://www.rapidsignalph.com

4.1 Nožice mikroupravljača

Čip u DIP (engl. *Dual In-line Package*) kućištu ima 18 nožica, od čega ih se 13 može koristi kao ulazne ili izlazne nožice. Nožicama se čita ili mijenja izlazno stanje pomoću specijalnih 8-bitnih registara PORTA i PORTB, svaka nožica mapirana je na jedan od bitova ovih registara. Mijenjanjem sadržaja registara TRISA i TRISB na jednak se način svakoj nožici određuje da li je ulazni ili izlazni. Ako je bit koji kontrolira nožicu postavljen na '0' tada je nožica izlazna, u suprotnom nožica je u ulaznom stanju. Prilikom pokretanja mikroupravljača svi TRIS bitovi postavljeni su na '1', dakle sve nožice postavljene su kao ulazne. Ovaj model također ima i nožicu za vanjski prekid – nožica INT, nožicu za vanjsko resetiranje mikroupravljača – MCLR, te jednu nožicu za vanjsku kontrolu 8-bitnog brojača – T0CKI. Od

ostalih nožica dvoje služe za prihvat radnog takta mikroupravljača, te dvoje za napajanje. Neke nožicu su multipleksirane, tj. dvije različite funkcije nalaze se na jednoj nožici, stoga nije moguće istovremeno koristiti svih 13 I/O nožica i sve opisane mogućnosti. Pomoću registara posebne namjene određuje se funkcija pojedinih nožica.

4.2 Instrukcije

Mid-range porodica raspolaže sa 35 instrukcija širine 14 bita. Instrukcije se prema namjeni mogu podijeliti u pet grupa:

- Prijenos podataka
- Aritmetičke operacije
- Logičke operacije
- Uvjetne instrukcije
- Bezuvjetna kontrola toka programa

4.2.1 Prijenos podataka

U instrukcijskom skupu postoje četiri instrukcije za prijenos podataka:

MOVF f, d - prijenos vrijednosti sa adrese f u registar W ili u adresu f

MOVWF f - prijenos vrijednosti sa adrese f u registar W

MOVLW k - prijenos neposredne vrijednosti k u registar W

SWAPF f, d - zamjena pozicije gornjih i donjih četiri bitova te spremanje u registar W ili u adresu f

Argument d određuje odredište operacije, tj. da li će se rezultat zapisati u registar W ili u registar na adresi f .

Iako prve dvije instrukcije djeluju redundantno, postoji vrlo važna razlika između njih. MOVF utječe na statusnu zastavicu Z, dok MOVWF ne utječe.

Varijanta MOVF f, f je korisna za provjeru da li je neka memorijska lokacija sadrži vrijednost '0' bez da potrebe mijenjanja registra W

Instrukcija MOVWF f je krucijalna ukoliko želimo sačuvati stanje mikroupravljača prilikom prekida, odnosno ukoliko želimo postići da nakon izlaska iz prekidne rutine važni registri budu jednakih vrijednosti kao što su bile prilikom ulaska u rutinu. Bez ove instrukcije ne

možemo sačuvati vrijednosti registra W bez da time možebitno ne promijenimo STATUS registar. Instrukcija SWAPF koja isto ne utječe na zastavicu Z također igra bitnu ulogu. Bez nje ne možemo povratiti vrijednost registra W a da pri tome ne utječemo na registar STATUS koji se vratili instrukciju prije.

4.2.2 Aritmetičke operacije

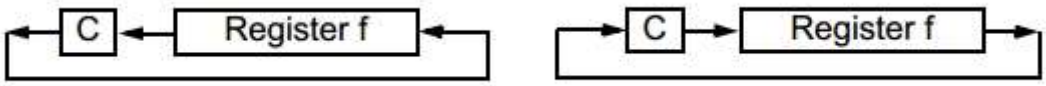
Procesor posjeduje instrukcije za zbrajanje (ADDWF, ADDLW), i oduzimanje (SUBWF, SUBLW). Operacije se ne izvode uzimajući u obzir zastavicu za prijenos bita (engl. *carry bit*) stoga programer sam mora uvažiti zastavicu ukoliko to želi.

Također podržava instrukcije za povećanje vrijednosti memorijske lokacije f za jedan (INCF), kao i umanjivanje za jedan (DECF). Sa instrukcijama CLRF i CLRW može se memorijska lokacija f ili registar W postaviti na vrijednost '0'.

4.2.3 Logičke operacije

Procesor podržava standardne logičke operacije ILI (ANDLW, ANDWF), I (IORLW, IORWF), NE (COMF) i isključivo ILI (XORLW, XORWF)

Uz navedene instrukcije, sa instrukcijama BCF i BSF omogućeno je postavljanje ili brisanje pojedinačnih bitova ram memorije. Instrukcije RLF i RRF rotiraju vrijednost memorijske lokacije lijevo ili desno, rotacija se odvija kroz zastavicu prijenosa koja služi kao privremeni bit.



Slika 4. Operacije rotacije biteva

Izvor: Microchip PIC16F84A Data Sheet

4.2.4 Uvjetne instrukcije

U arhitekturi PIC uvjetne instrukcije testiraju uvjet, te ukoliko je uvjet zadovoljen, procesor preskače iduću instrukciju. Ovime se može postići uvjetovan skok ukoliko je iduća instrukcija skok ili poziv potprograma. Procesor raspolaže sa četiri uvjetne instrukcije:

- BTFSC f, b - preskače ukoliko je zadani bit postavljen na '1'
- BTFSS f, b - preskače ukoliko je zadani bit postavljen na '0'
- DECFSZ f, d - umanjuje f, te preskače ukoliko je f jednak '0'

- INCFSZ f, d - uvećava f, te preskače ukoliko je f jednak '0'

4.2.5 Bezuvjetna kontrola toka programa

- GOTO k - skok programa na adresu k
- CALL k - skok programa na adresu k te povratak mjesto skoka sa RETURN
- RETLW k - isto kao i RETURN, samo što pri povratku u W učita vrijednost k
- RETFIE - povratak iz rutine prekida

4.2.6 Ostale instrukcije

- CLRWDT - resetira sigurnosnom brojilu na nulu
- SLEEP - zaustavlja procesor i ulazi u stanje niske potrošnje el. energije

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	

Slika 5. Instrukcijski skup mikroupravljača PIC16F84

Izvor: Microchip PIC16F84A Data Sheet

4.3 Programska memorija

Kao dio srednje familije proizvoda, ovaj mikroupravljač koristi 13-bitni programski brojač i može adresirati do 8K x 14bit memorije, no samo 1K je fizički implementirano. Mikroupravljač ignorira troje bitova najviše vrijednosti u programskom brojaču. Npr. ako se napravi skok na adresu 1025, procesor će nastavi izvađanje programa sa adrese br. 2. Na lokaciji 04h nalazi se rutina prekida.

4.4 Radna memorija

Memorijski prostor je podijeljen na dvije banke od 128 lokacija. PIC16F84 ima 16 registra posebne namjene i 68 registara opće namjene. Iako se tih 84 registara moglo smjestiti u samo jednu banku, to nije napravljeno zbog kompatibilnosti sa ostatkom "mid-range" familije.

Address	Bank 0	Bank 1	Address
00h	INDF	←	80h
01h	TMR0	OPTION_REG	81h
02h	PCL	←	82h
03h	STATUS	←	83h
04h	FSR	←	84h
05h	PORTA	TRISA	85h
06h	PORTB	TRISB	86h
07h	Unimplemented	←	87h
08h	EEDATA	EECON1	88h
09h	EEADR	EECON2	89h
0Ah	PCLATH	←	8Ah
0Bh	INTCON	←	8Bh
0Ch - 4Fh	GPR	←	8Ch - CFh

Slika 6. Adresni prostor radne memorije

Izvor: http://www.piclist.com

Na slici 6. prikazan je adresni prostor radne memorije za memorijske banke 0 i 1. Većina registara mapirano je u obje banke, te se u tom slučaju pristupa istom fizičkom registru neovisno aktivnoj adresnoj banci, uz pet iznimaka – registri TMR0/OPTION, PORTA/TRISA, PORTB/TRISB, EEDATA/EECON1, EEADR/EECON2. Na prvih 12 adresa nalaze se mapirani registri posebne namjene, kojima se upravljaju procesor i periferije. Na adresama od 12 do 79 nalaze se registri opće namjene, na slici žuto obojano. Ostatak

dostupnog adresnog prostora, od adresa 80 do 127, nije fizički implementiran te čitanje tih registara uvijek vraća nulu.

4.5 Registri posebne namjene

PIC16F84 ima 16 registara posebne namjene mapiranih u adresni prostor radne memorije, pomoću kojih se kontrolira rad procesora i dodatne periferije. Registri PORT i TRIS u ranijem su poglavlju objašnjeni kao registri vezani uz kontrolu fizičkih nožica mikroupravljača. Registri koji upravljaju integriranom EEPROM memorijom su EEDATA, EEADR, EECON1 i EECON2. Registra TMR0 sadrži trenutnu vrijednost integriranog 8-bitnog brojača, dok se u OPTION registru može podesiti djelitelj kako bi se povećao vremenski opseg brojača. U registru INTCON se može uključiti ili isključiti mogućnost prekida, te svaki izvor prekida pojedinačno kontrolirati.

4.5.1 Adresni registri

Osim direktnog adresiranja radne memorije, procesor ima mogućnost indirektnog adresiranja. Adresa kojoj se želi indirektno pristupit upiše se u registar FSR te se čitanjem i pisanjem u registar INDF čita ili mijenja lokacija na koju pokazuje registar FSR. Budući da je FSR registar 8-bitni, a adresni prostor ovog mikroupravljača sastoji se od dvije banke po 128 adresa, indirektnim adresiranjem može se linearno pristupiti cijelom adresnom prostoru mikroupravljača.

Druga dva para adresnih registara su PCL i PCLATH. U registru PCL mapirano je donjih osam bitova programskog brojila, u koji se može čitati i pisati, čime se mijenja tok programa. PCLATH predstavlja gornje bitove programa, ni nije dio direktno adresiranog programskog brojila, već je zasebni registar koji služi kao međuspremnik. Kada se u PCL zapisuje nekad vrijednost, istodobno se sadržaj međuspremnika PCLATH upisuje u gornje biteve programskog brojila. Iz tog se razloga prilikom svakog upisivanja u PCL registar treba pobrinuti da se PCLATH registru nalazi ispravna vrijednost. Čitanje i pisanje PCL registra omogućava adresiranje relativno s obzirom na programsko brojilo, što znači da je moguće unutar koda konstruirati tablicu podataka iz koje se uzimaju podaci na temelju indeksa. Upravo za ovu svrhu dodana je instrukcija RETLW koja prilikom povratka iz potprograma u W registar upiše ukodiranu vrijednost k, te time omogućuje efikasniju implementaciju tablica sa manje potrebnog koda.

No ovdje se i krije potencijalna greška koju je lako napravit. Prilikom planiranja tablice treba imati na umu početnu poziciju tablice u programskom adresnom prostoru, jer ukoliko dođe do

preliva registra PCL prilikom računanja adrese, procesor će napravit skok na krivu adresu i izvršit potpuno različite instrukcije naspram željenih.

4.5.2 Registar STATUS

U status registru kontrolira se podjela memorije, te provjeravaju određena stanja procesora.

7 0

IRP	RP1	RP0	TO	PD	Z	DC	C

- C - *carry bit*, operacije zbrajanja i oduzimanja utječu na ovaj bit
- DC - *digital carry bit*, postavljen ukoliko je gornjih četiri bitova uvećano sa nule
- Z - *zero bit*, ukoliko je rezultat operacije '0' ovaj bit je postavljen
- PD - *power-down*, postavljen na '0' ukoliko je procesor probuđen iz stanja spavanja
- TO - time-out, postavljen na '0' ukoliko je došlo do preliva u sigurnosnom brojilu
- RP0 - kontrolira podjelu memorije
- RP1 - u ovom modelu mikroupravljača bit nije primjenjiv
- IRP - u ovom modelu mikroupravljača bit nije primjenjiv

5. Programska implementacija simulatora

Za implementaciju simulatora odabran je programski jezik Java, te je biblioteka *Swing* odabrana za implementaciju grafičkog sučelja simulatora. Programska implementacija sadrži simulator PIC1684F mikroupravljača, parser tekstualnog asemblerskog programskog koda u strojni opkod mikroupravljača, te grafičko sučelje koje vizualno prikazuje unutarnja stanja mikroupravljača.

5.1 Implementacija simulatora PIC16F84 mikroupravljača

Implementacija je fokusirana oko dvije glavne apstrakcije, registri mikroupravljača - realizirani sa apstraktnim razredom *Registar8_Base* koji implementira 8-bitne registre i ALU, te instrukcije mikroupravljača - realizirani sa razredom "Instruction" te razredima koji nasljeđuju ovaj razred, koji implementiraju instrukcije i upravljačku jedinicu. Iako su registri i aritmetičko-logička jedinica (engl. *Arithmetic Logic Unit*) unutar dizajna mikroupravljača jasno odvojeni konstrukti, kao također i instrukcije i upravljačka jedinica, zadani simulator neće simulirati hardversku arhitekturu mikroupravljača, već jednu apstrakciju više - programski model mikroupravljača (engl. *Instruction Set Architecture - ISA*), koji ima samo dvije bitne apstrakcije - instrukcija i registar.

Radna memorija mikroupravljača implementirana kao polje tog razreda. Iako se u tu svrhu moglo koristiti polje nekog primitivnog podatkovnog tipa, poput int ili byte, apstraktni razred koristi se da bi se mogla prema potrebi pozvati dodatna logika prilikom operacije sa tim vrijednostima. Razloga tome je što se kod određenih operacija moraju postavljati razne zastavice mikroupravljača, te također što zadani mikroupravljač ima nekoliko vrsta registara, koje instrukcije tretiraju potpuno identično, iako su oni interno potpuno različiti. Iz tog razloga potrebno je nekoliko razreda koji imaju zajedničko sučelje, ali djelomično različitu implementaciju. Takvim dizajnom omogućeno je da razred "Instruction" i naslijeđeni ne moraju pratiti kakvom vrstom registra manipuliraju i da li moraju pokrenuti kakve dodatne radnje, nego sve registre tretirana na potpuno jednak način, dok se implementacija tih registara brine o dodatnim radnjama, baš kao i u fizičkoj implementaciji mikroupravljača.

Razred "Instruction" čini bazu za četiri dodatna razreda, modelirana prema četiri binarna formata instrukcija mikroupravljača iz familije PIC16. Razred "Instruction" ima implementiran samo "kustur" razreda, osnovne metode, dok je u naslijeđenim razredima implementirana logika instrukcija. Da bi ovi razredi mogli pristupiti registrima, koristi se

sučelje "InternalCpuInterface" koje daje pristup svim registrima, stogu i potrebnim informacijama za uspješno izvršavanje svih instrukcija.

Kako bi razredi na bazi "Instrukcija" mogli izvršiti sve zadaće koje mogu izvršitit mikroupravljači PIC16, potrebna su još dva dodatna razreda - "RegisterPC" i "StackMemory". Razred "RegisterPC" predstavlja 13-bitni programski brojač mikroupravljača, koji je djelomično memorijski mapiran. Razred "StackMemory" predstavlja hardversku stog memoriju koju mikroupravljač koristi za proceduralne pozive i prekide.

5.1.1 Razred „Cpu“

Razred "Cpu" je kompozicija svih objekata potrebnih za simulaciju mikroupravljača. To uključuje razrede koji simuliraju programsku memoriju, radnu memoriju, hardverski stog, module poput brojača i prekidni sustav, te njima pripadajuće registre posebne namjene. Ovaj razred implementira dva sučelja, jedno koje razred *Instruction* koristi za pristup svim podacima i metodama potrebnim za izvršavanje instrukcija mikroupravljača, dok drugo sučelje koriste razredi grafičkog sučelja za dohvat svih podataka koje grafičko sučelje prikazuje.

Programska memorija je implementirana kao polje instanci razreda „Instruction“. Razred također instancira "StackMemory" za simulaciju hardverskog stoga, "RegistarFile" za simulaciju radne memorije, te razrede InterruptController, TimerController i EepromController za simulaciju modula mikroupravljača.

Pozivanjem metode *executeInstruction* izvršava se jedna instrukcija. Ova metoda ažurira module mikroupravljača, provjerava da li je došlo do prekida, te izvršava instrukciju koja se nalazi na adresi na koju pokazuje programsko brojilo.

Tablica 1. Javne metode razreda „Cpu“

Povratna vrijednost	Naziv	Opis
(konstruktor)	Cpu()	Instancira objekt
void	reset()	Postavlja Cpu u početno stanje
void	clearRom()	Briše programsku memoriju
void	executeInstruction()	Izvršava jednu instrukciju
String	parseAssemblerCode(String text)	Parsira kod u programsku memoriju

Izvor: programski kod PIC16 simulatora

Primjer implementacije metode „executeInstruction“:

```
public void executeInstruction()
{
        timerModule.onTick();
        eepromController.onTick();

        if( interruptController.isInterrupted())
        {
            HwStack.push( RegisterFile.PC.get());
            RegisterFile.PC.set( INT_VECTOR );
            interruptController.setGlobalEnable( false );
            isInIsr = true;
        }

        rom[ RegisterFile.PC.get()].execute();
}
```

5.1.1.1 Sučelje "CpuInternalInterface"

Ovo sučelje daje pristup svim potrebnim resursima za izvršavanje cijelog instrukcijskog skupa mikroupravljača.

Tablica 2. Javne metode sučelja „CpuInternalInterface“

Povratna vrijednost	Naziv	Opis
void	pushStack(int val)	Postavlja vrijednost na stog
int	popStack()	Dohvaća vrijednost sa stoga
Register8b_Base	getRam(int address)	Dohvaća registar sa zadane adrese
Register8b_Base	getW()	Dohvaća radni registar
RegisterStatus	getStatus()	Dohvaća registar sa zastavicama
RegisterPC	getPc()	Dohvaća programsko brojilo
void	enableGlobalInterrupts()	Aktivira prekide

Izvor: programski kod PIC16 simulatora

5.1.1.2 Sučelje "CpuExternalInterface"

Uz metode koje nudi sučelje "CpuInternalInterface", grafičkom sučelju potrebne su dodatne metode za prikaz svih unutarnjih stanja simuliranog mikroupravljača. Uz nove metode, sučelje nasljeđuje metode definirane u sučelju "CpuInternalInterface".

Tablica 3. Javne metode sučelja „CpuExternalInterface“

Povratna vrijednost	Naziv	Opis
Instruction	getRom(int address)	Dohvaća instrukciju sa zadane adrese
int	getActiveBank()	Vraća informaciju trenutno aktivnoj banki radne memorije
boolean	isIsr()	Vraća informaciju da li mikroupravljač trenutno obrađuje prekid

Izvor: programski kod PIC16 simulatora

5.1.2 Razred "RegistarFile"

Radna memorija mikroupravljača simulirana je sa razredom "RegistarFile" . Razred ima jednu javnu metodu pomoću koje se dohvaća registar sa bilo koje adrese, pri čemu automatski provjerava sa koje banke treba dohvatiti registar, te odrađuje funkcionalnost indirektnog adresiranja. Iz jednu javnu metodu, instance važnih registara posebne namjene dostupne su pomoću javnih varijabli poput radnog registra *W*, programskog brojila, registra *STATUS*, i ostalih važnih registara. Javne varijable su konstantnog tipa, te nije moguće promijeniti koju instancu varijabla referencira.

Implementiran je pomoću dva polja instanci razreda *Register8b_Base,* svako polje predstavlja jednu banku memorije. U polja se instanciraju svi potrebni opći registri i registri posebne namjene, dok se fizički neimplementirani adresni prostor popunjava sa posebnim razredom koji implementira ponašanje tog prostora. Konstruktor ovog razreda prima instance razreda svih modula budući da su potrebne za instanciranje registara posebne namjene.

Tablica 4. Javne metode razreda „RegistarFile“

Povratna vrijednost	Naziv	Opis
(konstruktor)	RegisterFileMemory(TimerController timerModule, InterruptController interruptController)	Instancira objekt
Register8b_Base	getRam(int adr)	Dohvaća registar sa zadane adrese

Tablica 5. Javne varijable razreda „RegistarFile“

Tip	Naziv	Opis
Register8b_Base	W	Instanca radnog registra
RegisterPC	PC	Instanca programskog brojila
RegisterStatus	STATUS	Instanca registra STATUS
Register8b_Base	PORTA	Instanca registra PORTA
Register8b_Base	PORTB	Instanca registra PORTB

Izvor: programski kod PIC16 simulatora

5.1.3 Razred "StackMemory"

Ovaj razred implementira stog memorijsku strukturu. Struktura prima 13-bitne vrijednosti budući da služi isključivo za pamćenje memorijskih adresa iz programskog brojila. Ovaj razred je nužan simuliranje instrukcija "call" i "ret", te mehanizma prekida.

Memorija je implementirana kao polje varijabli *int* tipa, te sa jednom *int* varijablom koja se koristi kao kružni pokazivač na stog.

Tablica 6. Javne metode razreda „StackMemory“

Povratna vrijednost	Naziv	Opis
(konstruktor)	StackMemory(int size)	Instacira stog zadane veličine
void	push(int val)	Postavlja vrijednost na stog
int	pop()	Uzima vrijednost sa stoga

int	getData(int address)	Vraća vrijednost sa određene adrese stoga
int	getStackPointer()	Vraća vrijednost pokazivača na stog

Izvor: programski kod PIC16 simulatora

5.1.4 Razred Register8b_Base

Ovaj razred simulira rad registra i aritmetičko-logičke jedinice. Implementiran je kao apstraktni razred sa djelomičnom implementacijom, stoga ovaj razred nije moguće instancirati već služi kao baza za razrede koji nasljeđuju ovaj razred i u potpunosti implementiraju metode. Sve aritmetičke i logičke operacije implementirane su u ovom razredu, ali metode za dohvaćanje i postavljanje same vrijednosti tog registra nisu implementirane, već su apstraktne. Ovime se na jednostavan način postiže da registri svih vrsta imaju zajedničko sučelje, te ih instrukcije tretiraju na jednak način.

Postoje nekoliko vrsta registara u adresnom prostoru PIC16F84: standardni koji služe za pamćenje općih podataka, neimplementirani - memorijske adrese koje instrukcije mogu adresirati ali se na njima ne nalaze fizički implementirani registri, te memorijski-mapirani registri modula poput brojila i prekidnog sustava. Svi navedeni registri implementirani su nasljeđivanjem i implementiranjem ovog razreda.

Tablica 7. Javne metode razreda „Register8_Base“

Povratna vrijednost	**Naziv**	**Opis**
(konstruktor)	Register8b_Base()	Instancira objekt
int	get()	Dohvaća vrijednost registra
void	set(int newValue) set(Register8b_Base reg)	Postavlja vrijednost registra
boolean	getBit(int b)	Dohvaća vrijednost adresiranog bita
void	setBit(int b)	Postavlja adresirani bit na „1“
void	clearBit(int b)	Postavlja adresirani bit na „0“
Flags	add(Register8b_Base reg) add(int k)	Zbraja vrijednost registra i argumenta, te sprema rezultat

Flags	sub(Register8b_Base reg) sub(int k)	Oduzima vrijednost registra i argumenta, te sprema rezultat
boolean	logAnd(Register8b_Base reg) logAnd(int k)	Radi logički „I“ između vrijednosti registra i argumenta, te sprema rezultat
boolean	logOr(Register8b_Base reg) logOr(int k)	Radi logički „ILI“ između vrijednosti registra i argumenta, te sprema rezultat
boolean	logXor(Register8b_Base reg) logXor(int k)	Radi isključivi „ILI“ između vrijednosti registra i argumenta, te sprema rezultat
boolean	rotateLeft(boolean carryBit)	Rotira u lijevo bitove registra + carry zastavicu
boolean	retateRight(boolean carryBit)	Rotira u desno bitove registra + carry zastavicu
boolean	complement()	Radi logički „NE“ nad vrijednošću registra, te sprema rezultat
void	swapNibbles()	Izmjenjuje mjesta četiri gornja i donja bitova vrijednosti 8-bitnog registra

Izvor: programski kod PIC16 simulatora

Sve metode implementirane su na način prvo dohvate trenutnu vrijednost registra sa apstraktnom metodom "get", izvrše potrebnu operaciju, te postave novu vrijednost sa apstraktnom metodom "set". Za svaku operaciju postoje dvije metode, jedna koja kao argument prima cjelobrojni broj (engl. *integer*), te jedna koja kao argument prima instancu razreda "Register8b_Base". Ukoliko operacija može promijeniti neku od zastavica, te informacija se vraća kao rezultat poziva metode. Ukoliko se radi o jednoj zastaviti, informacija se vraća kao podatak tipa „boolean“, a ukoliko se radi o više zastavica, informacija se vraća kao instanca razreda "Flags". Sve metode koje primaju instancu razreda "Register8b_Base" implementirane su tako da samo dohvate vrijednost registra proslijeđenog kroz argument, te sa tom vrijednošću pozovu istoimenu metodu koja prima cjelobrojnu vrijednost.

Primjer implementacije metode:

```
public Flags add(int k)
{
    // check argument range
    if(k <0|| k >255)
        thrownew IllegalArgumentException("Nedopustena vrijednost"+k);

    Flags cpuFlags =new Flags();

    // do the operation
    int result =this.get()+ k;

    // check for overflow
    if(result >0xFF)
    {
        result &=0xFF;// simulate overflow
        cpuFlags.c =true;
    }

    // check cpu flags
    if(result ==0)
        cpuFlags.z =true;

    if( ((this.get()&0xF)+(k &0xF)) > 0xF )
        cpuFlags.dc =true;


    this.set(result);
    return cpuFlags;
}
```

5.1.4.1 Razred Flags

Nekoliko instrukcija PIC16 instrukcijskog skupa može modificirat sve tri aritmetičke zastavice, te kako bi se lakše razmijenila informacija o sve tri zastavice napravljen je ovaj pomoćni razred. Razred nema nikakvih metoda, smo tri javno dostupne varijable koji predstavljaju tri zastavice, te ga razred "Register8b_Base" u nekim metodama koristi kao vrstu povratne vrijednosti.

Tablica 8. Javne metode razreda „Flags“

Tip varijable	Naziv	Opis
boolean	c	Vrijednost „Carry“ zastavice
boolean	dc	Vrijednost „Digit Carry“ zastavice
boolean	z	Vrijednost „Zero“ zastavice

Izvor: programski kod PIC16 simulatora

5.1.4.2 Razred Register8b_Standard

Registri koji pripadaju GPR dijelu radne memorije implementirani su razredom "Register8b_Standard". Ovaj razred nasljeđuje "Register8b_Base" te implementira apstraktnu metodu *set* koja sprema vrijednost u varijablu tipa int, te apstraktnu metodu *get* koja čita vrijednost iz te varijable.

5.1.4.3 Razred Register8b_Unimplemented

Ovaj razred predstavlja registre tj. adresni prostor kojeg instrukcije mogu adresirati ali na čijim adresama nema fizički implementiranih registara. Čitanje tih adresa uvijek vraća "0", dok zapisivanje ništa ne radi (iako ALU zastavice svejedno mogu biti promijenjene tokom pristupa ovim registrima). Na taj način je implementiran i ovaj razred, "get" metoda samo vraća 0, dok "set" metoda ne radi ništa. Budući da su sve aritmetičke i logičke operacije i dalje implementirane, tj. naslijeđene iz baznog razreda, razred Instrukcija može ove registre tretirani jednako kao i GPR registre, te također iz istog razloga nakon operacija potrebne zastavice budu promijenjene jednako kao i na stvarnom mikroupravljaču.

5.1.5 Implementacija registara posebne namjene

Registri posebne namjene koriste se kako bi se dohvatile neke posebne informacije o stanju procesora ili modula mikroupravljača, ili pokrenule određene akcije u modulima. Iz tog razloga implementirani su dodatnim razredima kako bi im se implementirala potrebna dodatna logika.

5.1.5.1 Razred RegisterStatus

Razred "RegisterStatus" predstavlja registar STATUS, te nasljeđuje razred "Register8b_Standard". Uz postojeće metode, implementira dodatne metode za dohvaćanje aritmetičko-logičkih zastavica "Carry", "Digit carry" i "Zero". Također je implementira specifičnost ovog registra u tome što se u bitove tri i četiri ne može zapisivat instrukcijama, samo čitat.

Tablica 9. Javne metode razreda „RegisterStatus“

Povratna vrijednost	**Naziv**	**Opis**
void	setCpuFlags(Flags flags)	Postavlja sve aritmetičke zastavice
boolean	getC()	Dohvaća vrijednost „Carry“ zastavice
void	setC()	Postavlja „Carry“ zastavicu
void	clearC()	Briše „Carry“ zastavicu
boolean	getDC()	Dohvaća vrijednost „Digital Carry“ zastavice
void	setDC()	Postavlja „Digital Carry“ zastavicu
void	clearDC()	Briše „Digital Carry“ zastavicu
boolean	getZ()	Dohvaća vrijednost „Zero“ zastavice
void	setZ()	Postavlja „Zero“ zastavicu
void	clearZ()	Briše „Zero“ zastavicu
boolean	getRP0()	Dohvaća vrijednost „RP0“ zastavice
void	setRP0()	Postavlja „RP0“ zastavicu
void	clearRP0()	Briše „RP0“ zastavicu

Izvor: programski kod PIC16 simulatora

5.1.5.2 Razred RegisterOption

Simulacija registra OPTION implementira sve bitove koji nisu namijenjeni konfiguraciji nožica. To uključuje postavke prekida za brojilo, te postavke predbrojila (engl. *prescaler*). Prilikom promjene vrijednosti registra, pozivaju se metode razreda *InterruptController*.

5.1.5.3 Razred RegisterTmr0

Registar TMR0 implementiran je razredom "RegisterTmr0". Razred u metodama *get* i *set* poziva *get* i *set* metode razreda *TimerController*.

5.1.5.4 Razred RegisterIntcon

Razred "RegisterIntcon" predstavlja implementaciju registra INTCOM. Razred nasljeđuje "Register8b_Base", te dohvaćanje i zapisivanje vrijednosti registra implementira pozivima metodama razreda *InterruptController*.

5.1.6 Razred RegisterPC

RegistarPC simulira 13-bitno programsko brojilo mikroupravljača. Ovaj razred ima tek nekoliko metoda, za razliku od implementacije 8-bitnih registara, iz razloga što se nad ovim registrom ne vrše nikakve logičke ili aritmetičke operacije osim uvećanja varijable za jedan.

Iako je programsko brojilo moglo biti realizirano pomoću jedne cjelobrojne varijable, sastavljen je od dvije instance razreda *Register8b_Base*. Razlog tome je što je donjih 8 bitova programskog brojača (PCL) mapirano u adresni prostor mikroupravljača, te je moguće i čitati i mijenjati te bitove brojača. Umjesto provjere prilikom svakog pisanja u radnoj memoriji da li je odabrana lokaciji PCL registra i eventualno raditi bitovne operacije nad programskim brojilom, brojilo je realizirano pomoću dvije instance razreda *Register8b_Base* te razred pruža metodu za dohvaćanje instance registra koji služi kao PCL. U metodama postavljanja i uvećanja vrijednosti implementirana je logika koja brine da je 13-bitna vrijednost registra ispravno mapirana između dvije instance 8-bitnog registra.

Razred se također i brine da adresa koju sadrži ne izađe izvan granica adresnog programskog prostora PIC16F84, te da bude unutar granica indeksa polja koje simulira taj programski prostor.

Tablica 10. Javne metode razreda „RegisterPC“

Povratna vrijednost	**Naziv**	**Opis**
(konstruktor)	RegisterPC()	Instancira objekt
int	get()	Dohvaća vrijednost registra
void	set(int k)	Postavlja vrijednost registra
void	inc()	Uvećanje vrijednost registra za jedan
Register8b_Base	getPCL()	Dohvaća registar koji služi kao PCL

Izvor: programski kod PIC16 simulatora

5.1.7 Razred „Instruction“

Logika upravljačke jedinice implementirana je u ovom razredu. Svaka instanca ovog razreda predstavlja jednu instrukciju u programskoj memoriji mikroupravljača. Razred prilikom instanciranja prima strojni kod instrukcije, te ju u konstruktoru dekodira na tip instrukcije i operande prema određenom formatu. Četiri dodatna razreda nasljeđuju ovaj razred, svaki odgovoran za određeni format instrukcije.

Razredi imaju nekoliko javnih metoda, uključujući metodu "execute" koja izvršava logiku procesorske instrukcije. Metoda "execute" je implementirana na način da sve potrebne informacije i objekte dohvaća pomoću sučelja "CpuInternalInterface". Nakon što dohvati potrebne registre, npr. registar čija je adresa ukodirana u instrukciju, radni registar W, programsko brojilo, metoda uspoređuje tip instrukcije i na temelju vrste odlučuje koje metode poziva nad dohvaćenim registrima. Također ukoliko je potrebno radi operacije nad stog memorijom ili prekidnim sustavom.

Za instanciranje pravilnog razreda na temelju opkoda instrukcije, koristi se statička metoda *getInstance*. Metoda provjerom prvih nekoliko bitova instrukcije određuje kojem formatu pripada instrukcija. Potom alocira novu instancu odabranog razreda te referencu na instancu vrati kao povratnu vrijednost.

Tablica 11. Javne metode razreda „Instruction“

Povratna vrijednost	**Naziv**	**Opis**
(konstruktor)	Instruction(int op)	Instancira objekt
void	execute()	Izvršava instrukciju
String	getAsmCode()	Dohvaća naziv instrukcije
Instruction	getInstance(int op)	Vraća instancu instrukcije na temelju strojnog koda u numeričkom obliku
void	setCpuInterface(CpuInternalInterface c)	Dohvaća referencu na objekt sa sučeljem „CpuInternalInterface“

Izvor: programski kod PIC16 simulatora

Primjer implementacije instrukcije „execute“ u razredu „InstructionControl“:

```
@Override
publicvoid execute()
{
    super.execute();

    if(type == OPCODE_GOTO)
    {
        cpu.getPc().set( value );
    }
    else if(type == OPCODE_CALL)
    {
        cpu.pushStack( cpu.getPc().get());
        cpu.getPc().set(value );
    }
    elseif(type == OPCODE_RETURN)
    {
        cpu.getPc().set( cpu.popStack());
    }
    elseif(type == OPCODE_RETFIE)
    {
        cpu.getPc().set( cpu.popStack());
        cpu.enableGlobalInterrupts();
    }
}
```

5.1.7.1 InstructionBit

Instrukcije koje koriste operand za adresiranje bitova dekodirane su u ovome razredu. Četiri instrukcije koriste ovaj format. Razred u konstruktoru dekodira tri informacije: vrstu instrukcije, pozicija bita koji se adresira, te adresa registra nad kojim se adresira bit.

Format instrukcije: ii iibb bfff ffff

- i – vrsta instrukcije
- b – adresa bita
- f – adresa registra

5.1.7.2 InstructionByte

Instrukcije za operacije nad radnim registrom w te registrima iz ram memorije dekodirane su u razredu "InstructionByte". Konstruktor dekodira tri informacije iz opkoda: vrstu instrukcije,

bit odredišta, te adresa registra nad kojim se vrše operacije. Registar w koristi se implicitno te nije ukodiran u instrukciju.

Format instrukcije: ii iiii dfff ffff

- i – vrsta instrukcije
- d – bit odredišta
- f – adresa registra

5.1.7.3 InstructionLiteral

Instrukcije za operacije nad radnim registrom i ukodiranom vrijednošću dekodirane su u razredu "InstructionLiteral". Konstruktor dekorira dvije informacije iz instrukcije: vrstu instrukcije te ukodiranu neposrednu 8-bitnu vrijednost. Registar w koristi se implicitno te nije ukodiran u instrukciju.

Format instrukcije: ii iiii llll llll

- i – vrsta instrukcije
- l – ukodirana neposredna vrijednost

5.1.7.4 InstructionControl

Razred "InstructionControl" obuhvaća dva formata instrukcija. Jedan je bez operanda, drugi kao operand sadrži 11-bitnu adresu za skok programskog brojila.

Format instrukcije bez argumenta: ii iiii iiii iiii

Format instrukcije sa argumentom: ii iaaa aaaa aaaa

- i – vrsta instrukcije
- l – adresa programske memorije

5.1.8 Implementacija modula mikroupravljača

Moduli su implementirani na način da im je osnovna funkcionalnost implementirana u jednom razredu, te svi specijalni registri koji su povezani sa modulom primaju referencu na instancu određenog modula i pozivaju potrebne metode modula.

5.1.8.1 Razred „InterruptController“

Ovaj razred simulira funkcionalnosti prekidnog sustava. Sučelje je dizajnirano na nacin da modul pruža generičke metode za postavljanje prekidnih zastavica i maska za prekidne zastavice sa indeks brojem koji predstavlja određeni izvor prekida, te globalno uključivanje i isključivanje prekida. Moduli i registri potom pozivaju ove metode sa indeksom izvora prekida, dok razred „Cpu“ prilikom izvršenja instrukcije, prije samog izvršenja provjerava da li je mikroupravljač u prekidu te u tom slučaju radi potrebne operacije da se prekid izvrši.

Razred je implementiran sva dva polja tipa *boolean*. Jedno polje služi za pamćenje postavljenih zastavica koji predstavljaju izvor prekida, a drugo za omogućene ili onemogućene izvore prekida. Također se koristi i dodatna *boolean* varijabla za pamćenje da li su prekidi globalno omogućeni.

Tablica 12. Javne metode razreda „InterruptController“

Povratna vrijednost	**Naziv**	**Opis**
(konstruktor)	InterruptController()	Instancira objekt
boolean	isInterrupted()	Vraća informaciju da li je mikroupravljač u prekidu.
boolean	getGlobalEnable()	Dohvaća „Global Enable“ zastavicu
void	setGlobalEnable(boolean globalMask)	Postavlja „Global Enable“ zastavicu
void	setInterruptFlag(int index, boolean value)	Postavlja zastavicu prekida za određeni izvor
boolean	getInterruptFlag(int index)	Dohvaća zastavicu prekida za određeni izvor
void	setInterruptEnable(int index, boolean value)	Postavlja zastavicu maske za određeni izvor
boolean	getInterruptEnable(int index)	Dohvaća zastavicu maske za

		određeni izvor
int	getRegIntcon()	Dohvaća vrijednost „INTCON“ registra
void	setRegIntcon(int newValue)	Postavlja vrijednost „INTCON“ registra

Izvor: programski kod PIC16 simulatora

5.1.8.2 Razred „TimerController“

Razred "Timer" simulira 8-bitni hardverski brojač mikroupravljača, te funkcionalnost predbrojila koji mu se može pridružit. Javne metode ovog razreda pozivaju se iz razreda RegisterOption, RegisterTmr0, dok se metoda *onTick* poziva prilikom svakog izvršenja instrukcije. Pozivom metode onTick povećava se interna vrijednost brojača, odnosno predbrojila ukoliko je aktivirano. Također, nakon što brojač odbroji do maksimalne vrijednosti, razred poziva *InterruptController* te postavlja prekidnu zastavicu.

Tablica 13. Javne metode razreda „TimerController“

Povratna vrijednost	**Naziv**	**Opis**
(konstruktor)	Timer(InterruptController int)	
void	onTick()	Ažurira vrijednost brojača
void	setPrescalerActive(boolean active)	Aktivira „prescaler“ brojača
void	setPrescalerSetting(int setting)	Postavlja „prescaler“ brojača
int	get()	Dohvaća vrijednost brojača
void	set(int val)	Postavlja vrijednsot brojača

Izvor: programski kod PIC16 simulatora

5.2 Implementacija parsera asemblerskog programskog koda

Radi lakšeg korištenja i testiranja simulatora implementiran je parser za kod u asemblerskom jeziku. Razred parsira proizvoljan tekst, te ukoliko je asemblerski kod valjan, kao izlaz vraća strojni kod koji se učita u simulator. Format koda koji parser prima na ulazu je podskup

službenog formata od strane Microchip-a. Parser podržava sve instrukcije familije mikroupravljača PIC16 i nekoliko operatera.

5.2.1 Razred „Parser“

Razred "Parser" sadrži jednu javnu metodu koja prima asemblerski kod u obliku tipa *String* te kao povratnu vrijednost vraća listu strojnog kod u numeričkom obliku. Implementacija je podijeljena na nekoliko koraka implementiranih kao privatne metode koje poziva javna metoda *parse*.

U prvom koraku tekst se rastavlja na listu *stringova* koji predstavljaju riječi i operatere. Drugom koraku radi se identifikacija svakog *string* iz liste, te ih se pretvara u objekte razreda "Token" koji sadrže dodatne informacije o značenju *stringa*. U trećoj fazi *tokeni* se obrađuju te pretvaraju u instrukcije, argumente, labele i sl. U ovom koraku instancira se lista objekata AsmInstruction razreda koja predstavlja listu instrukcija, te koja sadrži opcode vrijednosti i vrijednosti argumenta. Provjerom tokena parsiraju se i popunjavaju vrijednosti AsmInstuction razreda, te "equ" i label lista. A zadnjoj fazi popunjene instance razreda AsmInstruction izračunavaju svoju opkod vrijednost, sa kojima se generira lista opkod vrijednosti koja je povratni argument.

Tablica 14. Javne metode razreda „Parser“

Povratna vrijednost	Naziv	Opis
ArrayList<AsmInstruction>	Parse(String text)	Parsira asemblerski kod u tekstualnom obliku, te vraća strojni kod u numeričkom obliku.

Izvor: programski kod PIC16 simulatora

5.2.2 Razred „Token“

Zadaća ovog razreda je da obradi ulazni argument tipa *String*, te prepozna da li se radi i nazivu instrukcije, operateru ili nepoznatoj riječi, te ukoliko se radi o instrukciji ili operateru, da prepozna o kojoj se instrukciji ili operateru radi.

Ulazni argument uspoređuje se sa definiranim riječima i operaterima, te pretvara u *tokene* - strukturu koja sadrži informaciju o tipu riječi, konkretno da li se radi o instrukciji, operateru, te koja instrukcija ili operater, ili da li se radi i nepoznatoj riječi. Razred ima tri javno

dostupne varijable, te jednu metodu koja kao argument prima objekt tipa "String", identificira riječ zapisanu u argumentu, te vraća objekt tipa "Token" sa popunjenim varijablama.

Tablica 15. Javne varijable razreda „Token“

Tip varijable	Naziv	Opis
int	type	Sadrži informaciju da li je parsirana riječ instrukcija, operater ili nepoznata
int	value	Sadrži informaciju o vrsti instrukcije ili operatera
String	word	Naziv riječi koja se parsira

Izvor: programski kod PIC16 simulatora

Tablica 16. Javne metode razreda „Token“

Povratna vrijednost	Naziv	Opis
Token	Parse(String word)	Parsira objekt tipa "String" te vraća popunjeni objekt tipa „Token“

Izvor: programski kod PIC16 simulatora

5.2.3 Razred „AsmInstruction“

Objekt razreda "AsmInstruction" predstavlja jednu instrukciju, te sadrži javne varijable za postavljanje tipa instrukcije i njihovih argumenata, pomoću kojih se u metodi "GetOpcode" izračunava konačna strojna numerička vrijednost pojedinačne instrukcije.

Tablica 17. Javne varijable razreda „AsmIntruction“

Tip varijable	Naziv	Opis
int	type	Informacija o vrsti instrukcije
int	arg1	Vrijednost prvog argumenta
int	arg2	Vrijednost drugog argumenta
String	label	Naziv labela ukoliko se koristi

Izvor: programski kod PIC16 simulatora

Tablica 18. Javne metode razreda „AsmInstruction“

Povratna vrijednost	Naziv	Opis
int	GetOpcode()	Izračunava strojni kod instrukcije u numeričkom obliku

Izvor: programski kod PIC16 simulatora

5.3 Implementacija grafičkog sučelja

Grafičko sučelje aplikacije koristi biblioteku *Swing*, te sljedeće elemente ovo biblioteke:

Tablica 19. Korišteni razredi i sučelja biblioteke Swing

Vrsta	Naziv	Opis
razred	*JPanel*	grafički element koji omogućuje grupiranje elementa *Swing* biblioteke
razred	*JTextArea*	grafički element za unos teksta
razred	*JLabel*	grafički element za prikaz teksta
sučelje	*AbstractTableModel*	Sučelje za definiranje izvora podataka tablice
sučelje	*TableCellRenderer*	Sučelje za definiranje izgleda tablice

Izvor: programski kod PIC16 simulatora

Grafičko sučelje aplikacije sastoji se od četiri glavna panela implementirana pomoću razreda *JPanel* iz biblioteke *Swing*:

- panel sa gumbima za upravljanje
- panel sa "JTextArea" za unos asemblerskog koda
- panel sa prikazom rom, ram, i instrukcijama simuliranog mikroupravljača
- panel sa JLabel za prikaz grešaka parsera

Sučelje ima gumbe za tri operacije:

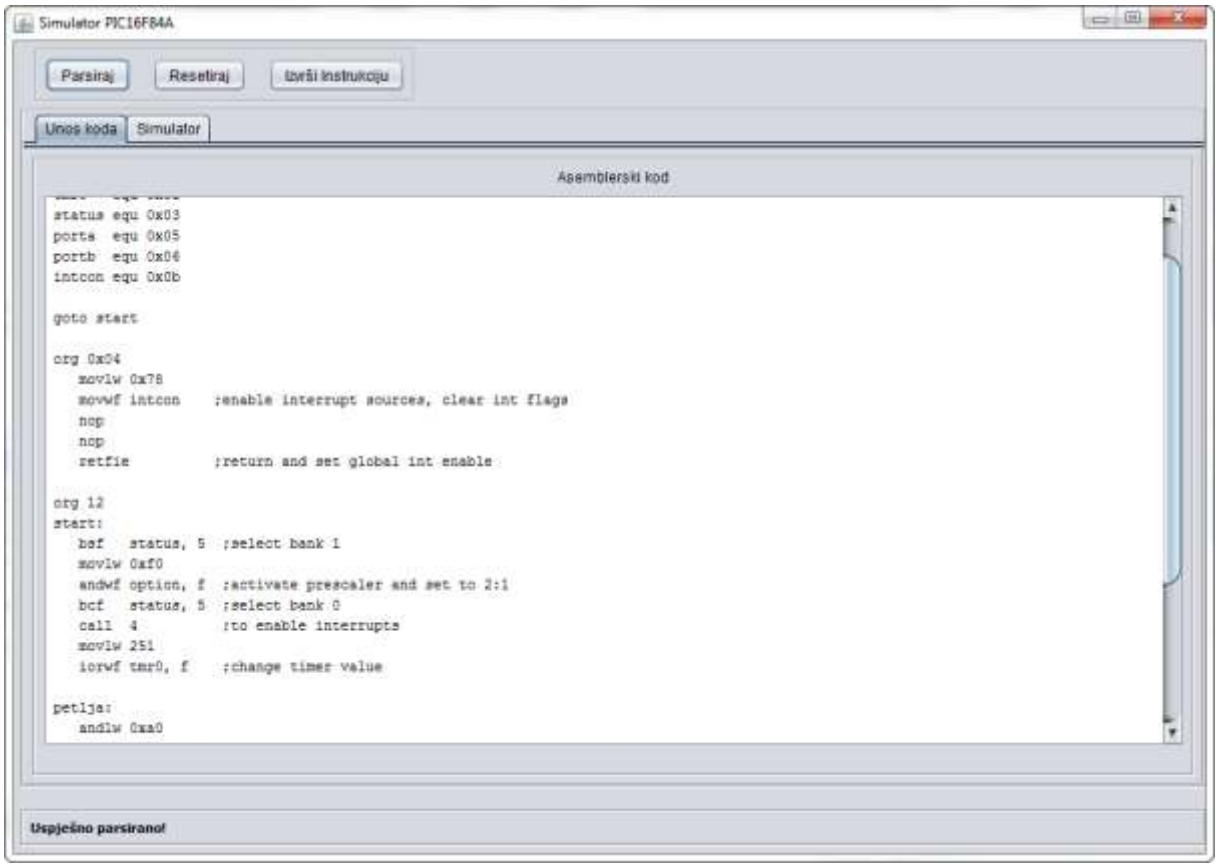
- Gumb „Parsiraj“ za parsiranje asemblerskog koda unesenog u prozoru za unos koda

- Gumb „Izvrši instrukciju“ za izvršavanje instrukcije mikroupravljača
- Gumb „Resetiraj“ za postavljanje mikroupravljača u početno stanje

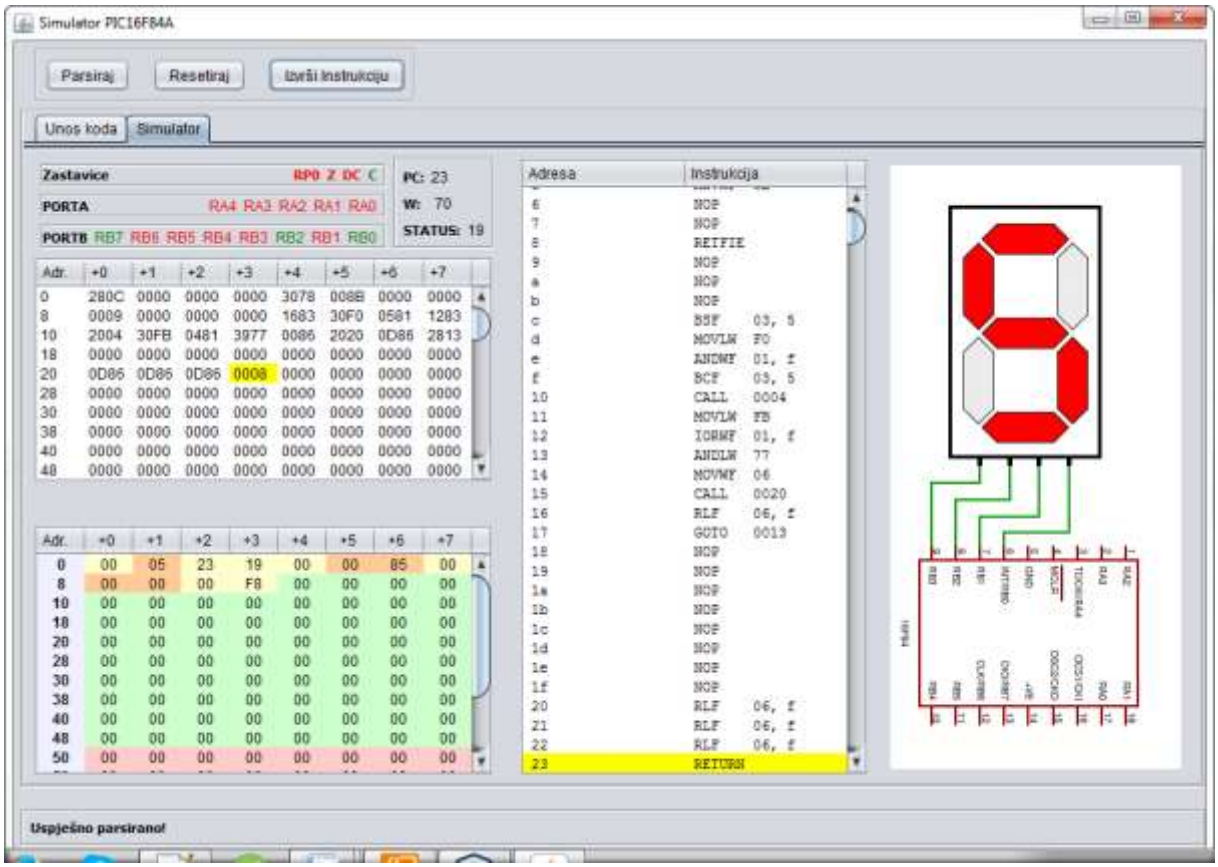
Središnji panel aplikacije podijeljen je na *jPanel* za unos asemblerskog teksta, te panel koji sadrži simulator mikroupravljača. U panelu simulatora mikroupravljača nalaze se tri tablice koje prikazuju numerički sadržaj radne memorije, programske memorije, te tekstualni opis sadržaja programske memorije. Sadržaj simulatora ažurira se prilikom svakog pritiska

Panel za prikaz greška parsera sadrži jedan "JLabel" koji ažurira svoj tekst prilikom svakog pritiska na gumb "Parsiraj".

U glavnoj dretvi grafičkog sučelja instanciran je objekt razreda *Cpu,* te gumbi *Parsiraj*, *Izvrši instrukciju* i *Resetiraj* pozivaju metode razreda *Cpu*. Da bi tablice dobile podatke o stanju mikroupravljača, za svaku tablicu napisan je razred koji implementiraju sučelje *AbstractTableModel,* te su tablicama ti razredi dodijeljeni kao zadani razredi za dohvat sadržaja tablica. Svaki od tih razreda prima objekt razreda *Cpu* kao tip sučelje *CpuInternalInterface,* te u implementacija poziva metode tog sučelja za dohvat podataka.



Slika 7. Grafičko sučelje simulatora



Slika 8. Grafičko sučelje simulatora

6. Zaključak

Kao što je u literaturi navedeno iznimno je malo vremena potrebno za savladavanje ove arhitekture, počevši od jednostavnog i instrukcijskog skupa, do iznimno jednostavne registarske i memorijske arhitekture. Mikroupravljači PIC arhitekture su jedni od najjednostavnijih i najkorištenijih mikroupravljača u svijetu, te time čine idealni izbor za simuliranje i kao i dobru priliku za učenje popularne arhitekture.

Iako je ovaj zadatak zbog same svoje prirode izrazito jasno i detaljno specificiran, zadatak ima mnogo elemenata koji se mogu na podosta različitih načina implementirati, i najveći izazov je u biti predstavljao izbor najboljeg rješenja za svaki takav element zadatka.

7. Popis korištene literature

[1] General-Purpose Microcontroller Family is Announced. http://www.computerhistory.org/semiconductor/timeline/1974-MCU.html (18.10.2014)

[2] Texas Instruments. (1976). TMS 1000 Series Data Manual.

[3] The Most Widely Used Computer on a Chip The TMS 1000. http://smithsonianchips.si.edu/augarten/p38.htm (18.10.2014)

[4] Examining The First Microprocessor Chip Devices. http://www.ukessays.com/essays/information-technology/examining-the-first-microprocessor-chip-devices-information-technology-essay.php (18.10.2014)

[5] History of pic microcontrollers . http://roboengineers.blogspot.com/2008/04/history-of-pic-microcontrollers.html (18.10.2014)

[6] Microchip Technology Delivers 10 Billionth PIC® Microcontroller to Samsung Electronics Co. http://www.microchip.com/pagehandler/en-us/press-release/microchip-technology-delivers-10-billionth-pic-mic.html (18.10.2014)

[7] PICmicro™ Mid-Range MCU Family Reference Manual. http://ww1.microchip.com/downloads/en/devicedoc/33023a.pdf (10.12.2014)

[8] Getting Started with On-chip Memory. http://ww1.microchip.com/downloads/en/DeviceDoc/ramrom.pdf (14.12.2014)

[9] Heffernan, D. 8051 TUTORIAL. http://galia.fc.uaslp.mx/~cantocar/microprocesadores/EL_Z80_PDF_S/8051.PDF (14.01.2015)

[10] Compare 8-bit PIC® MCU Architectures. http://www.microchip.com/pagehandler/en-us/family/8bit/architecture/home.html (25.11.2014)

[11] Microchip 8-bit PIC® Microcontrolers. http://www.microchip.com/stellent/groups/SiteComm_sg/documents/DeviceDoc/en557096.pdf (26.11.2014)

[12] Microchip PIC16F84A Data Sheet. http://ww1.microchip.com/downloads/en/devicedoc/35007b.pdf (25.11.2014)