LABORATORIJSKI PROJEKAT

Procesor

Potrebno predznanje

- Urađen laboratorijski zadatak 5
- Arhitekture procesora
- Projektovanje upravljačke jedinice digitalnih sistema
- Osnovno znanje asemblerskog programiranja

Šta će biti naučeno tokom izrade vežbe?

Tokom izrade projekta:

- Proširićete strukturu za računanje koju ste realizovali u prethodnoj vežbi
- Dodaćete instrukcioni registar i programski brojač u vaš procesor
- Implementiraćete dve memorije – za podatke i instrukcije
- Projektovaćete opštu kombinacionu upravljačku jedinicu koja odgovara na instrukcije
- Definisaćete skup instrukcija za procesor
- Implementiraćete aritmetičke instrukcije, skokove i instrukcije za rad sa memorijom
- Programiraćete vaš procesor

Apstrakt i motivacija

Tokom izrade laboratorijskog projekta, primenićete čitavo znanje koje ste stekli tokom semestra. Digitalni sistemi su svoju najveću moć pronašli u domenu računanja, tako da će vaš konačni zadatak biti da napravite univerzalnu strukturu za računanje – procesor. Ovaj procesor će ipak biti dosta jednostavniji od modernih procesora, ali će sadržati sve osnovne elemente koje ima i svaki moderan procesor – aritmetičko-logičku jedinicu, upravljačku jedinicu, registre, pristup memoriji. Vaš procesor će podržavati aritmetičke instrukcije, instrukcije skoka i instrukcije pristupa memoriji. Projektujući ovaj procesor, naučićete principe procesorske arhitekture i primenićete vaše znanje stečeno na ovom predmetu da napravite veoma koristan sistem. I naravno, na kraju ćete moći pisati programe za vaš sopstveni procesor!

ZADACI

1. Pripremni koraci

Ukoliko ste uspešno uradili laboratorijski zadatak 5, otvorite isti projekat. Kao prvi korak, dodaćemo još neke komponente u naš projekat, koje će biti potrebne kako bi naša upravljačka jedinica postala opšta i podržavala rad sa instrukcijama.

1.1. Programski brojač

U datoteci *cnt.vhd* realizovati 16-bitni brojač sa dozvolom upisa i brojanja. Brojač treba da ima mogućnost paralelnog upisa, uvećanja vrednosti za 1 i čuvanja stare vrednosti. Signal dozvole rada brojača iEN ima veći prioritet od signala dozvole upisa.

Tabela 1-1. Prolazi brojača

Prolaz	Smer	Funkcija
iCLK	in	signal takta
inRST	in	signal reseta, aktivan na niskom logičkom nivou
iD [15:0]	in	ulazni podatak
iEN	in	signal dozvole brojača, aktivan na visokom nivou; ako je na niskom nivou brojač zadržava vrednost
iLOAD	in	signal dozvole upisa; ako je na visokom nivou, brojač preuzima vrednost sa ulaza ID; ako je na niskom nivou, brojač uvećava vrednost za 1
oQ [15:0]	out	vrednost brojača

1.2. Memorija za instrukcije

U datoteci *instr_rom.vhd* realizovati kombinacionu ROM memoriju kapaciteta 32 reči širine 15 bita. ROM memorija će se koristiti za čuvanje instrukcija procesora, odn. programa. ROM memorija ima omogućeno samo čitanje, zbog čega se realizuje kombinaciono.

Tabela 1-2. Prolazi memorije

Prolaz	Smer	Funkcija
iA [4:0]	in	adresa
oQ [14:0]	out	izlazni podatak

Vrednosti u ROM memoriji neka za početak budu proizvoljne (npr. sve 0x0000). Ovo ćemo modifikovati kasnije.

1.3. Memorija za podatke

U datoteci *data_ram.vhd* realizovati sinhronu RAM memoriju kapaciteta 32 reči širine 16 bita. RAM memorija će se koristiti za smeštanje podataka. RAM memorija ima omogućeno i čitanje i pisanje, zbog čega je realizovana kao sekvencijalna mreža.

Tabela 1-3. Prolazi memorije

Prolaz	Smer	Funkcija
iCLK	in	signal takta osetljiv na opadajućoj ivici
inRST	in	signal reseta aktivan na niskom logičkom nivou
iA [4:0]	in	adresa
iD [15:0]	in	ulazni podatak
iWE	in	dozvola upisa podatka
oQ [15:0]	out	izlazni podatak

Upis u memoriju kontroliše dozvola upisa, a izlaz uvek ima vrednost lokacije na adresi koja se u datom trenutku nalazi na adresnom ulazu. Početne vrednosti RAM memorije treba da budu postavljene na 0x0000. Ovo ćemo modifikovati kasnije.

Signal takta mora biti osetljiv na **opadajućoj ivici** iz razloga navedenih u poglavlju 3.3.

2. Modifikovan vrh hijerarhije procesora

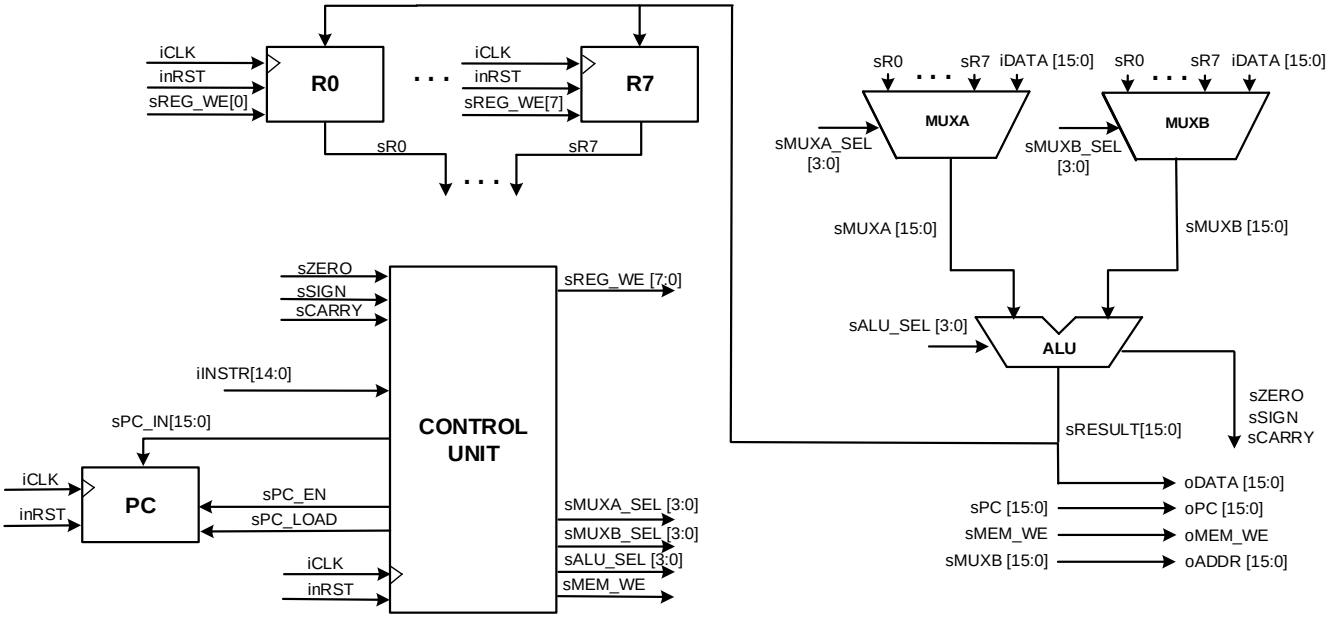
Vrh hijerarhije procesora treba modifikovati prema slici 2-1. Dodatna komponenta, u odnosu na vrh hijerarhije iz prethodne vežbe je:

- Programski brojač PC koji se koristi za čuvanje adrese trenutno izvršavane instrukcije.

Prolazi vrha hijerarhije su sumirani u tabeli 2-1.

Tabela 2-1. Prolazi vrha hijerarhije procesora

Prolaz(i)	Smer	Funkcija
iCLK	in	signal takta
inRST	in	signal reseta, aktivan na niskom logičkom nivou
iINSTR [14:0]	in	instrukcija iz memorije za instrukcije
iDATA [15:0]	in	podatak iz memorije za podatke
oPC [15:0]	out	vrednost programskog brojača, za adresiranje memorije za instrukcije
oDATA [15:0]	out	vrednost za upis u memoriju za podatke
oADDR [15:0]	out	adresa za memoriju za podatke
oMEM_WE	out	dozvola upisa u memoriju za podatke



Slika 2-1. Vrh hijerarhije procesora

3. Opšta upravljačka jedinica

U nastavku, implementiraćemo opštu upravljačku jedinicu koja prihvata instrukcije i izvršava ih, definišući vrednosti kontrolnih signala ostatku procesora.

Trenutna instrukcija iINSTR[14:0] se prihvata kao ulaz procesora.

Izvršavanje instrukcije počinje izborom operanada pomoću multipleksera MUXA i MUXB. Izlazi multipleksera su 16-bitni signali sMUXA i sMUXB, koji su ujedno i ulazi aritmetičko-logičke jedinice. Aritmetičko-logička jedinica izvršava izabranu operaciju i formira rezultat i statusne bite, a nakon toga se rezultat prosleđuje u neki od registara opšte namene R0-R7.

Upravljačka jedinica treba da generiše selekcione i signale dozvole ostalih delova digitalnog sistema: sREG_WE[7:0] (dozvole upisa u registre opšte namene R0-R7), sPC_IN, sPC_EN i sPC_LOAD (kontrolni signali i ulaz programskog brojača), sMUXA_SEL[3:0] i sMUXB_SEL[3:0] (odabir prvog i drugog operanda), sALU_SEL (kod operacije aritmetičko-logičke jedinice) i sMEM_WE (dozvola upisa u memoriju za podatke).

Na izlaz procesora se prosleđuju: 16-bitni izlazni podatak ka memoriji za podatke (oDATA), 16-bitna adresa memorije za podatke (oADDR), 16-bitna vrednost programskog brojača kao adresa memorije za instrukcije (oPC) i dozvola upisa u memoriju za podatke (oMEM_WE). Prolazi su sumirani u tabeli 3-1.

Tabela 3-1. Prolazi upravljačke jedinice

Prolaz(i)	Smer	Funkcija
iCLK	in	signal takta
inRST	in	signal reseta, aktivan na niskom logičkom nivou
iZERO	in	statusni bit zero
iSIGN	in	statusni bit sign
iCARRY	in	statusni bit carry
iINSTR [14:0]	in	instrukcija iz instrukcione memorije
oPC_EN	out	signal dozvole programskog brojača
oPC_LOAD	out	signal dozvole upisa programskog brojača
oPC_IN [15:0]	out	adresa skoka
oREG_WE [7:0]	out	dozvole upisa registara opšte namene
oMUXA_SEL [3:0]	out	selekcija za multiplekser prvog operanda
oMUXB_SEL [3:0]	out	selekcija za multiplekser drugog operanda
oALU_SEL [3:0]	out	kontrola aritmetičko-logičke jedinice
oMEM_WE	out	dozvola upisa u memoriju za podatke

3.1. Osnovni skup instrukcija

Procesor za početak treba da obavlja operacije koje prikazuje Tabela 3-2. U pitanju su operacije koje direktno izračunava aritmetičko-logička jedinica. U levoj koloni tabele su navedena imena instrukcija i oznake operanada, a u desnoj koloni funkcije pojedinih instrukcija. Sintaksa RZ←[RX] označava da se sadržaj registra X upisuje u registar Z.

Tabela 3-2. Aritmetičko-logičke instrukcije procesora

Operacija	Funkcija	Kod instrukcije
`mov Rz,Rx`	RZ ← [RX]	000000
`add Rz,Rx,Ry`	RZ ← [RX] + [RY]	000001
`sub Rz,Rx,Ry`	RZ ← [RX] - [RY]	000010
`and Rz,Rx,Ry`	RZ ← [RX] & [RY]	000011
`or Rz,Rx,Ry`	RZ ← [RX] | [RY]	000100
`not Rz,Rx`	RZ ← not [RX]	000101
`inc Rz,Rx`	RZ ← [RX] + 1	000110
`dec Rz,Rx`	RZ ← [RX] - 1	000111
`shl Rz,Rx`	RZ ← shl [RX]	001000
`shr Rz,Rx`	RZ ← shr [RX]	001001
`neg Rz,Rx`	RZ ← - [RX]	001010
`ashr Rz,Rx`	RZ ← ashr [RX]	001011

Svaka instrukcija je kodovana sa 15 bita. Format instrukcije je sledeći:

```
IIIIIIZZZXXXYYY
```

Prvih 6 bita predstavlja kod instrukcije (iz tabele 3-2), a sledećih 9 bita predstavljaju registre koji učestvuju u realizaciji instrukcije. Registar rezultata RZ se navodi prvi, a nakon njega se navode dva registra operanada (RX i RY). Ukoliko instrukcija koristi samo jedan registar operanda, drugi se može definisati proizvoljno.

Iako su za navedene instrukcije dovoljna četiri bita, ovde se rezerviše šest iz razloga proširenja skupa instrukcija koji će se raditi u nastavku.

Upravljačka jedinica predstavlja kombinacionu mrežu koja, na osnovu trenutne instrukcije koju dobija na ulazu iINSTR[15:0], generiše upravljačke signale ostatku procesora. Implementirajte upravljačku jedinicu koja podržava instrukcije iz tabele 3-2 i realizuje ih u zavisnosti od vrednosti pojedinih delova instrukcije.

Programski brojač se za sada stalno uvećava za 1, bez paralelnog upisa. Memoriji za podatke se ne pristupa.

Za sada ne koristimo sledeće upravljačke signale: `sPC_IN`, `sPC_LOAD`, `sMEM_WE`.

3.2. Instrukcije skoka

Skokovi su fundamentalni deo svakog programskog jezika. U nastavku ćemo ih podržati i u našem procesoru.

Dopuniti skup instrukcija instrukcijom bezuslovnog skoka iz Tabele 3-3. Ova instrukcija ima sledeći format:

```
IIIIIIAAAAAAAAA
```

Instrukcija se sastoji iz 2 dela:

- iINSTR (14 downto 9) – kod instrukcije,
- iINSTR (8 downto 0) – adresa na koju se skače.

Tabela 3-3. Instrukcija bezuslovnog skoka

Operacija	Funkcija	Kod instrukcije
`jmp ADDR`	PC ← ADDR	010000

Adresa na koju se skače treba da se upiše u programski brojač. U tu svrhu se koriste signali `sPC_IN` i `sPC_LOAD`.

Dopuniti skup instrukcija i instrukcijama uslovnog skoka iz Tabele 3-4. Ove instrukcije imaju isti format kao instrukcija bezuslovnog skoka, sa drugačijim kodom instrukcije. Statusni biti ZERO, SIGN i CARRY so označeni sa Z, S i C respektivno.

Uslovni skokovi su korisni za grananje unutar programa – ukoliko je uslov ispunjen izvršava je jedan skup instrukcija, u suprotnom drugi.

Tabela 3-4. Instrukcije uslovnog skoka

Operacija	Funkcija	Kod instrukcije
`jmpz ADDR`	if Z=1 PC ← ADDR	010001
`jmps ADDR`	if S=1 PC ← ADDR	010010
`jmpc ADDR`	if C=1 PC ← ADDR	010011
`jmpnz ADDR`	if Z=0 PC ← ADDR	010101
`jmpns ADDR`	if S=0 PC ← ADDR	010110
`jmpnc ADDR`	if C=0 PC ← ADDR	010111

3.3. Pristup memoriji za podatke

Konačno, omogućićemo pristup memoriji za podatke preko instrukcija LOAD i STORE.

Tabela 3-6. Instrukcije pristupa memoriji za podatke

Operacija	Funkcija	Kod instrukcije
`ld Rz,Ry`	RZ ← [[RY]]	100000
`st Ry,Rx`	[RY] ← [RX]	110000

Instrukcija LOAD upisuje u registar RZ sadržaj koji se nalazi u memoriji podataka na adresi koja piše u registru RY (zbog toga je u opisu funkcije stavljeno dvostruko dereferenciranje registra RY). Izvršavanje instrukcije se odvija na sledeći način:

1. Multiplekser MUXB selektuje registar RY i njegova vrednost se prosleđuje na izlaz oADDR; u istom taktu vrednost iz memorije se pojavljuje na ulazu iDATA, pošto se čitanje memorije vrši kombinaciono; multiplekser MUXA selektuje vrednost iDATA.
2. ALU propušta vrednost prvog operanda koja je jednaka vrednosti iDATA.
3. Vrednost iDATA se upisuje u registar RZ.

Instrukcija STORE upisuje u memoriju, na adresu koja piše u registru RY, vrednost koja piše u registru RX. Izvršavanje instrukcije se odvija na sledeći način:

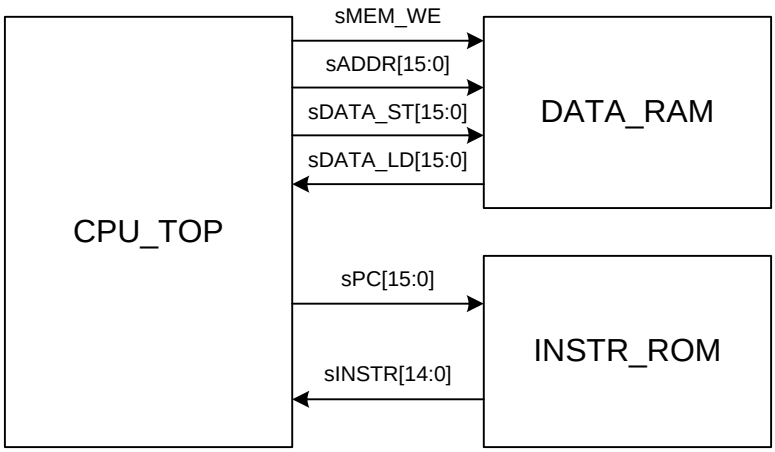
1. Multiplekser MUXB selektuje registar RY i njegova vrednost se prosleđuje na izlaz oADDR. Multiplekser MUXA selektuje registar RX.
2. ALU propušta vrednost prvog operanda.
3. Vrednost iz ALU se upisuje u memoriju (aktivira se signal dozvole `sMEM_WE`).

Kako bi memorija za podatke upisala vrednost prilikom izvršavanja instrukcije STORE, neophodno je da dobije rastuću ivicu takta **tokom trajanja takta procesora** u kome se dešava instrukcija STORE. Zbog toga je memorija za podatke osetljiva na **opadajuću ivicu** takt signala.

4. Vrh hijerarhije računarskog sistema

Nakon realizacije procesora i dve memorije, sve komponente ćemo spojiti u jedan računarski sistem. Ovo je primer Harvard arhitekture računara, koja koristi odvojene memorije za instrukcije i podatke, a veoma je popularna u procesorima za obradu signala. Von Neumann-ova arhitektura, tipična za personalne računare, koristi zajedničku memoriju i za podatke i za instrukcije.

U datoteci *top.vhd* realizovati vrh hijerarhije sistema, u kome je procesor povezan sa memorijama, prikazan na Slici 4-1. Jedini prolazi ovog sistema su ulazni takt (iCLK) i reset (inRST). Sistem bez izlaza i dalje nije sintetizabilan, ali se može proveriti u simulaciji.



Slika 4-1. Vrh hijerarhije računarskog sistema CPU + ROM + RAM

5. Programiranje procesora

Došao je red na oživljavanje našeg računarskog sistema. Procesor nije veoma koristan bez programa koji će da izvršava, pa ćemo sada napisati dva programa u instrukcionoj memoriji procesora (odn. dve verzije instrukcionog ROM-a).

5.1. Prvi program

Da bi oživeli naš procesor, a i igrali se malo sa matematikom, napišite program koji računa vrednost funkcije f(N) ukoliko je ona definisana na sledeći način:

$$f(0) = 0$$

$$f(N) = (f(N-1) + 1) * 2^N \qquad (N > 0)$$

Ulazni parametar N možemo lako menjati promenom konstante u samom programu, pa ga izaberite proizvoljno, ali neka bude veći od 3.

5.2. Slobodni program

Konačno, impresionirajte sebe i druge oko vas programom koji računa ... šta god želite! Napišite neki interesantan program, težak za ručno izračunavanje, izračunajte nešto zanimljivo i pokažite pravu moć vašeg procesora!

Ukoliko bude potrebno, možete i proširiti ROM i RAM ako ste ga pravili, kako bi vaš program stao u memoriju. Pošto simulacija nema ograničenja (teorijski), možete širiti memorije koliko god vam treba.

Ako vam trebaju brojevi koji zahtevaju više od 16 bita, proširite vaš procesor da bude 32-bitni ili 64-bitni. Videćete da je takvo proširenje veoma jednostavno na našem procesoru. Širinu instrukcije ne morate povećavati u tom slučaju.

ZAKLJUČAK

Šta reći na kraju, budite veoma ponosni na ono što ste uradili – napravili ste vaš prvi kompletan procesor! Iako ovaj procesor nema neke napredne mogućnosti o kojima ćete učiti u narednim predmetima (čudne reči kao što su protočna struktura, predviđanje grananja, eliminacija hazarda), on sadrži osnovne elemente koje imaju svi procesori i može da izvrši bilo koji algoritam koji obuhvata podržane operacije i staje u njegov ROM. Ovim projektom zaokružili ste osnovno znanje projektovanja digitalnih sistema, naročito onih koji imaju za cilj neko računanje, i napravili ste veoma važan most između softvera i hardvera. Čestitamo na uspešno završenim vežbama!