LABORATORIJSKA VEŽBA 6
Projektovanje procesora

Potrebno predznanje

- Razumevanje kombinacionih mreža, sekvencijalnih mreža, automati, instaciranje modula, ukratko sve do sada naučeno
- Arhitektura računara i asembler

Šta će biti naučeno tokom izrade vežbe?

U ovoj vežbi:

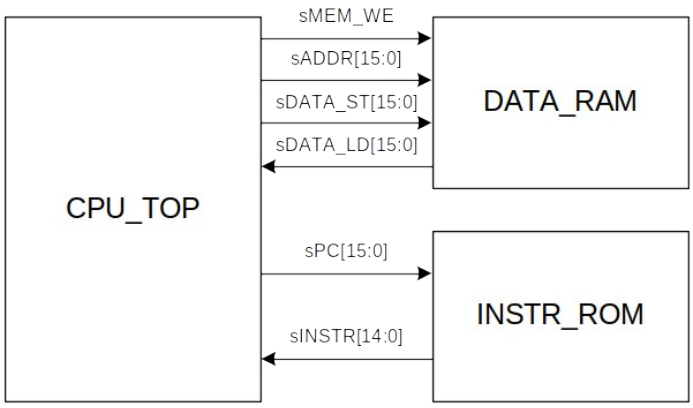
- Kombinujete standardne kombinacione i sekvencijalne komponente, koje su same po sebi veoma jednostavne, u veoma složeni sistem čija primenjivost značajno prevazilazi primenjivost pojedinih komponenti
- Implementiraćete neke komponente procesora
- Vežbaćete projektovanje složenih sistema sastavljenih od nekoliko modula
- Simuliraćete složeni digitalni sistem i vežbati razumevanje simulacionih dijagrama složenih sistema
- Napravite vezu između znanja iz programske podrške i fizičke arhitekture digitalnih sistema
- Razumećete kako funkcioniše procesor

Apstrakt i motivacija

Mozak vašeg ličnog računara je procesor – univerzalna struktura za računanje koja može da izvršava složene aritmetičke i logičke operacije. U ovoj vežbi projektovaćete složenu strukturu za računanje, preteču procesora, koja može da izvršava osnovne operacije – sabiranja, oduzimanja, uvećanja, umanjenja, logičke operacije i pomeranja. Naučićete kako da koristite multipleksere za izbor operanada, kako da projektujete aritmetičko-logičku jedinicu, kako da računane statusne bite (zero, carry, sign) i kako da koristite registre za čuvanje izračunatih vrednosti. Konačno, implementiraćete i upravljačku jedinicu koja izvršava jedan program na vašem procesoru.

Vrh hijerarhije računarskog sistema: CPU + memorija

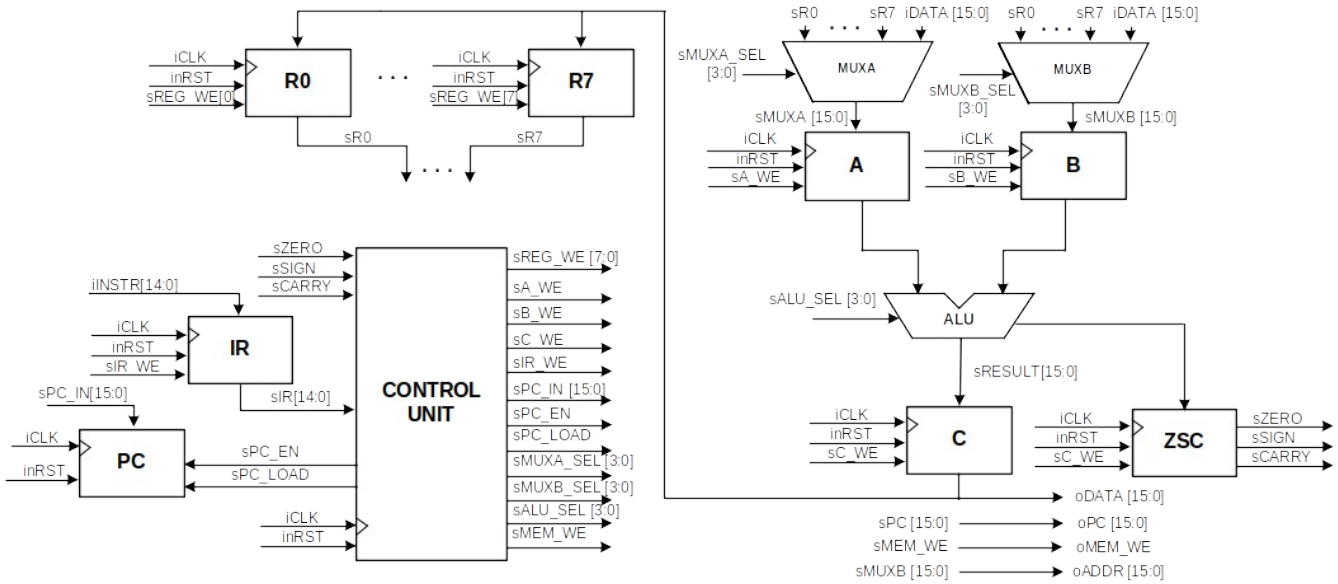
U datoteci *top.vhd* realizovati vrh hijerarhije sistema, u kome je procesor povezan sa memorijama, prikazan na Slici 1. CPU_TOP sadrži procesor, dok je INSTR_ROM memorija za instrukcije, a DATA_RAM memorija za podatke. Ovakva konfiguracija memorija se zove Harvardska arhitektura.



Slika 1. Vrh hijerarhije sistema

Opis procesora

Slika 2 prikazuje izgled procesora koji će biti obrađivan u ovoj vežbi i odgovara *cpu_top.vhd* datoteci i njegovim podmodulima.



Slika 2. Arhitektura procesora

Trenutna instrukcija iINSTR [14:0] se smešta u registar instrukcije (IR), ako je aktivan signal dozvole sIR_WE.

Ulazni podaci se preko 16-bitnog ulaza iDATA i multipleksera upisuju u registre A i B. Izlaz multipleksera su 16-bitne magistrala podataka sMUXA i sMUXB, koje su ujedno i ulazi registara A i B. Aritmetičko logička jedinica izvršava operacije tako što se prvi operand smesti u registar A i drugi operand smesti u registar B u jednom ciklusu takta, a u narednom ciklusu se rezultat operacije smešta u registar C, registar za smeštanje rezultata, a nakon toga rezultat se može proslediti u neki od registara opšte namene R0-R7.

Upravljačka jedinica treba da generiše selekcione i signale dozvole ostalih delova digitalnog sistema kao sto su: sREG_WE[7:0] (dozvole upisa u registre opšte namene R0-R7), sA_WE, sB_WE i sC_WE (dozvole upisa u registre A, B i C), sIR_WE (dozvola upisa u instrukcioni registar), 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).

Na izlaz procesora se prosleđuju: 16-bitni izlazni podatak ka memoriji (oDATA), 16-bitna vrednost programskog brojača (oPC) i dozvola upisa u memoriju (oMEM_WE).

Tabela 1 prikazuje osnovne aritmetičko-logičke instrukcije procesora. U pitanju su operacije koje direktno izračunava aritmetičko-logička jedinica. U levoj koloni tabele su navedena imena instrukcija i oznake operanada, a u sredwoj koloni funkcije pojedinih instrukcija. Sintaksa RZ ← [RX] označava da se sadržaj registra X upisuje u registar Z. Desna kolona sadrži kod instrukcije.

Tabela 1. Instrukcije procesora

Operacija	Funkcija	Kod instrukcije
mov Rz,Rx	RZ ← [RX]	00 0000
add Rz,Rx,Ry	RZ ← [RX] + [RY]	00 0001
sub Rz,Rx,Ry	RZ ← [RX] - [RY]	00 0010
and Rz,Rx,Ry	RZ ← [RX] & [RY]	00 0011
or Rz,Rx,Ry	RZ ← [RX] | [RY]	00 0100
not Rz,Rx	RZ ← not [RX]	00 0101
inc Rz,Rx	RZ ← [RX] + 1	00 0110
dec Rz,Rx	RZ ← [RX] - 1	00 0111
shl Rz,Rx	RZ ← shl [RX]	00 1000
shr Rz,Rx	RZ ← shr [RX]	00 1001
ashl Rz,Rx	RZ ← ashl [RX]	00 1010
ashr Rz,Rx	RZ ← ashr [RX]	00 1011

Svaka instrukcija je kodovana sa 15 bita. Lista instrukcija se smešta u posebnu, instrukcionu memoriju, odvojenu od memorije za smeštanje podataka. Format instrukcije na bitskom nivou je IIIIIIZZZXXXYYY. Pri tome je sa IIIIII označen kod intrukcije, sa XXX adresa registra RX, sa YYY adresa registra RY, a sa ZZZ adresa registra Z. Kod instrukcije označen sa IIIIII se može predstaviti sa TTOOOO, gde TT označava tip instrukcije, a OOOO označava kod operacije. Kao što se može primetiti iz Tabele 1, aritmetičko-logičke instrukcije imaju tip instrukcije kodovan sa 00.

Izvršenje svih instrukcija je raspoređeno u četiri faze (četiri periode takta):

1. prihvat insntrukcije (*Instruction Fetch)*
2. dekodovanje instrukcije *(Instruction Decode)*,
3. izvršenje instrukcije *(Execute)* i
4. pamćenje rezultata (*Write Back)*.

Ovakvo raspoređivanje izvršenja instrukcija je potrebno zbog arhitekture procesora, koja sadrži četiri registarska stepena:

1. u prvoj fazi se vrednost smešta u registar IR,
2. u drugoj fazi se, na osnovu trenutne instrukcije u IR, vrednost smešta u registre A i B,
3. u trećoj fazi se, na osnovu vrednosti u IR, A i B, vrši računanje i vrednost smešta u registar C,

4. u četvrtoj fazi se rezultat računanja iz registra C smešta u registre R0-R7.

Upravljačka jedinica predstavlja automat koji konstantno kruži između četiri stanja koji predstavljaju prethodno navedene četiri faze izvršenja instrukcije. Funkcija prelaza tog automata je jednostavna, automat bezuslovno prelazi iz stanja FETCH u stanje DECODE, iz stanja DECODE u stanje EXECUTE, iz stanja EXECUTE u stanje WRITE_BACK i iz stanja WRITE_BACK u stanje FETCH.

Funkcija izlaza, odn. vrednosti kontrolnih signala za ostatak procesora, definiše se na osnovu trenutnog stanja automata i trenutno izvršavane instrukcije koja se nalazi u registru IR i predstavlja ulaz upravljačke jedinice (iINSTR).

Pristup memoriji za podatke

Pristup memoriji za podatke je omogućen preko instrukcija LOAD i STORE. Opis instrukcija je dat u Tabeli 2.

Tabela 2. Instrukcije pristupa memoriji

Operacija	Funkcija	Kod instrukcije
`ld` Rz,Ry	RZ ← [[RY]]	10 0000
`st` Rx,Ry	[[RY]] ← [RX]	11 0000

Format ovih instrukcija je sličan kao i kod aritmetičko-logičkih, s tom razlikom da se ignorišu neka polja intrukcije, kao naprimer polje operacije. Polje tipa određuje da li je u pitanju LOAD instrukcija (vrednost 10) ili pak STORE instrukcija (vrednost 11).

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 u sledeće 4 faze:

1. Instrukcija se učitava u IR.
2. 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.
3. ALU propušta vrednost iz registra A koja je jednaka vrednosti iDATA.
4. Vrednost iDATA se, iz registra C, 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 u sledeće 4 faze:

- Instrukcija se učitava u IR.
- Multiplekser MUXB selektuje registar RY i njegova vrednost se prosleđuje na izlaz oADDR. Multiplekser MUXA selektuje registar RX.
- ALU propušta vrednost iz registra A.
- Vrednost iz registra C se upisuje u memoriju (postavlja se signal dozvole MEM_WE).

Zbog potrebe instrukcije STORE, a da ne bi morali menjati arhitekturu procesora, signal selekcije multipleksera MUXB ne samo da ima vrednost definisanu trenutnom instrukcijom u fazi DECODE, već takođe i fazi

WRITE_BACK, kada je potrebno aktivirati signal MEM_WE radi upisa u memoriju. Takođe, signal REG_WE u WRITE_BACK fazi tokom instrukcije STORE ne upisuje vrednost ni u jedan registar.

Bezuslovni skok

Instrukcija bezuslovnog skoka je prikazana u Tabeli 3. Ova instrukcija ima sledeći format: IIIIIAAAAAAAAA. Polje tipa instrukcije TT je kodovano sa 01. Instrukcija se sastoji iz 2 dela:

- iINSTR (13 downto 9) – kod instrukcije,
- iINSTR (8 downto 0) – adresa na koju se skače.

Tabela 3. Instrukcija bezuslovnog skoka

Operacija	Funkcija	Kod instrukcije
jmp ADDR	PC ← ADDR	01 0000

Adresa na koju se skače treba u fazi WRITE_BACK da se upiše u programski brojač. U tu svrhu se koriste signali sPC_IN i sPC_LOAD. Pošto se rezultat operacije ne upisuje u registre opšte namene, signal oREG_WE u WRITE_BACK fazi, kao i signal oC_WE u EXECUTE fazi su neaktivni tokom instrukcije JMP, pa se vrednost ne upisuje ni u jedan registar procesora, kao ni u registre rezultata i statusnih bita.

Uslovni skok

Instrukcije uslovnog skoka su date u Tabeli 4. Ove instrukcije imaju isti format kao instrukcija bezuslovnog skoka, sa drugačijim kodom instrukcije. Polje operacije definiše Statusni biti ZERO, SIGN i CARRY so označeni sa Z, S i C respektivno.

Tabela 4. Instrukcije uslovnog skoka

Operacija	Funkcija	Kod instrukcije
jmpz ADDR	if Z=1 PC ← ADDR	01 0001
jmps ADDR	if S=1 PC ← ADDR	01 0010
jmpc ADDR	if C=1 PC ← ADDR	01 0011
jmpnz ADDR	if Z=0 PC ← ADDR	01 0101
jmpns ADDR	if S=0 PC ← ADDR	01 0110
jmpnc ADDR	if C=0 PC ← ADDR	01 0111

Kao i kod instrukcije bezuslovnog skoka, ponašanje oREG_WE u WRITE_BACK fazi, kao i signal oC_WE u EXECUTE fazi je isto kako se ne bi upisivalo ni u jedan registar, kao ni u registre rezultata i statusnih bita.

ZADATAK

Za početak, otvorite projekat u kome ćete realizovati delove procesora. Upoznajte se sa strukturom projekta.

Usled trivijalnosti registari i multiplekser su već realizovani.

U datoteci *alu.vhd* je potrebno realizovati aritmetičko-logičku jedinicu (ALU) procesora, dok u datoteci *control_unit.vhd* je potrebno relaizovati kontrolnu jedinicu (CU).

ALU treba da računa rezultat operacije nad svojim ulaznim operandima, u zavisnosti od selekcionih bita. Podržane operacije su sumirane u Tabeli 5.

ALU treba da bude implementirana kao kombinaciona mreža. Pretpostaviti da sve operacije rade sa označenim brojevima u II komplementu.

Tabela 5. Operacije ALU

Kod operacije	Operacija
0000	A
0001	A + B
0010	A – B
0011	A and B
0100	A or B
0101	not (A)
0110	A + 1
0111	A – 1
1000	shl (A)
1001	shr (A)
1010	ashl (A)
1011	ashr (A)

Tabela 6. Prolazi ALU

Prolaz	Smer	Funkcija
iA [15:0]	in	prvi ulazni operand
iB [15:0]	in	drugi ulazni operand
iSEL [3:0]	in	selekcija operacije
oC [15:0]	out	rezultat operacije
oZERO	out	statusni bit, rezultat jednak nuli
oSIGN	out	statusni bit, rezultat negativan
oCARRY	out	statusni bit, rezultat ima prenos

Statusni biti takođe treba da budu realizovani kombinaciono, odn. kao funkcije rezultata iz ALU. Aktivna vrednost statusnih bita treba da bude na visokom nivou.

U kontrolnoj jedinici je potrebno realizovati automat za 4 gorepomenute faze i u izlaznoj funkciji kombinaciono odrediti kontrolne signale. U Tabeli 7 sledi spisak upravljačkih signala čiju vrednost treba postaviti u

datoj fazi izvršenja instrukcije. U fazama u kojima signal nije naveden, on treba da bude neaktivan. Najjednostavnije način za realizaciju ovoga je na početku kombinacionog procesa izlazne funkcije svim kontrolnim signalima dodeliti neutralne vrednosti (sve 0), a posle u case-u za određenu fazu dodeljivati vrednosti potrebnim kontrolnim. Na ovaj način se će svakom signalu biti dodeljena vrednost i kombinaciona mreža je potpuna i neće se generisati leč.

Tabela 7. Aktivni upravljački signali po fazama izvršenja instrukcije

Instruction Fetch	Instruction Decode	Execute	Write-Back
oIR_WE	oMUXA_SEL oMUXB_SEL oA_WE oB_WE	oALU_SEL oC_WE	oREG_WE oMEM_WE oMUXB_SEL sPC_EN oPC_IN oPC_LOAD

U prvoj fazi se, signalom sIR_WE omogućuje prihvatanje instrukcije u registar IR.

U drugoj fazi se vrši odabir prvog i drugog operanda signalima sMUXA_SEL i sMUXB_SEL, a odabrani operandi upisuju u registre A i B postavljanjem signala dozvole sA_WE i sB_WE na 1.

U trećoj fazi se odabira operacija aritmetičko-logičke jedinice i vrednost upisuje u registar C, kao i statusne flip-flopove.

U poslednjoj fazi se rezultat operacije upisuje u registar opšte namene (samo jedan bit u signalu sREG_WE treba da bude aktivan) i programskom brojaču se, pomoću signala sPC_EN, dozvoljava uvećavanje adrese za 1, kako bi se u narednom taktu preuzela naredna instrukcija.

Radi lakše realizacije ALU i CU, instrukcije će biti dodavano postepeno i biće proveravane putem simulacije. Radi testiranja se koristi algoritam kumulativnog sabiranja datog u datoteci *cumsum.asm*. Za početak je potrebno implementirati prvu instrukciju **inc** u ALU, kao i dodelu vrednosti sledećim kontrolnim signalima u CU: IR_WE, MUXA_SEL, A_WE, ALU_SEL, C_WE, REG_WE. Nakon toga je potrebno pokrenuti simulaciju. Uvidom u talasiće da li se u registru R0 nakon WRITE_BACK faze 0. intrukcije nalazi vrednost 1. Ukoliko se ne nalazi, proveriti prethodne signale da li su kako bi trebali. Na primer, registar C bi trebao imati vrednost 1 posle EXECUTE faze, selekcioni signal ALU jedinice treba da ima kod operacije 0110 u EXECUTE fazi, registar A i B vrednosti 0 nakon DECODE faze, i tako dalje.

Dodatni zadaci

1. Neki kontrolni signali se postavljaju u određenim fazama, međutim ako bi bili postavljeni u svim fazama to ne bi promenilo rezultat izvršavanja instrukcija od strane procesora. Primer jednog takvog signala oPC_IN. Probajte razumeti zašto je to tako. Pronaći ostale takve signale i optimizovati tako što neće zavisiti od faze i samim tim neće morati biti unutar izlazne funkcije automata CU.

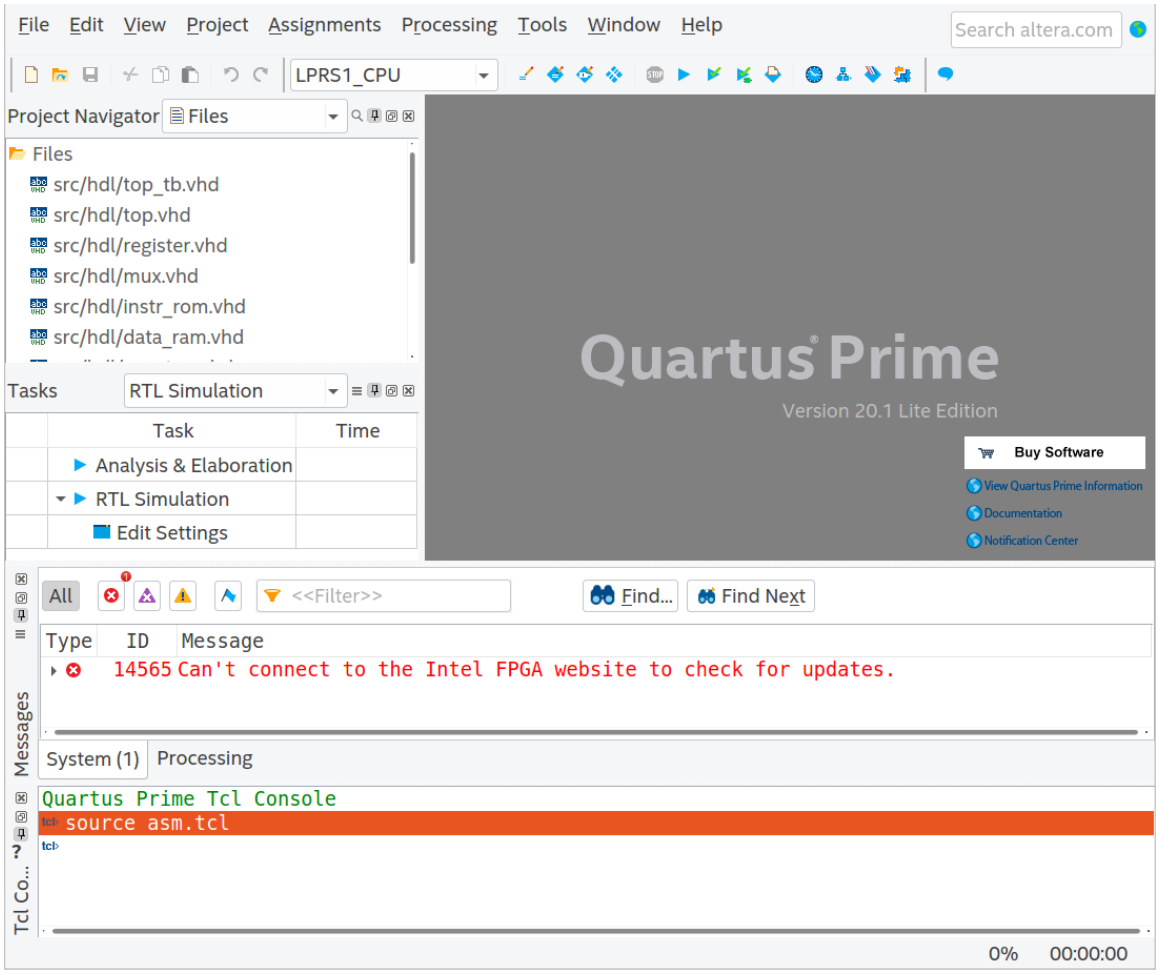
2. Optimizovati sDO_JMP signal u CU.

Dodatak: Asembliranje

U slučaju izmene programa u datoteci cumsum.asm, potrebno je ponovno izvršiti asembliranje. Radi pokretanja asemblera, otvoriti Tcl Console preko prečice **Alt+2**. U toj istok konzolici upisati komandu praćenu enterom:

source asm.tcl

Slika 3 prikazuje izgled Quartus-a pri asembliranju.



Slika 3. Asembliranje