PROCESOR DP32

UVOD

Kroz ovu vežbu analiziraćemo princip rada procesora DP32. Ova vežba se sastoji iz dva dela:

I Teorijski deo
II Praktični deo

Prvi deo se odnosi na sagledavanja koja se tiču sledećih detalja:

1. Tipovi podataka
2. Definicija skupa podataka
3. Struktura sistema

U drugom delu biće opisano test kolo i simulator DP32. Kroz zadavanje odgovarajućih test sekvenci analiziraće se rad DP32.

Zadatak studenata se sastoji u sledećem:

- Upoznavanje sa arhitekturom DP32 i davanjem odgovara na odgovarajuće zadatke
- Upoznavanje sa radom simulatora DP32 i korišćenje simulatora DP32 radi izvršavanja odgovarajućih definisanih programskih test sekvenci.

VEŽBA BROJ 1:

TEORIJSKI DEO PROCESORA DP32

1. TIPOVI PODATAKA

Svaka arhitektura može da manipuliše sa različitim brojem tipova podataka. Za mikroprocesor se kaže da podržava određeni tip podataka samo ako je u stanju da sa prihvatljivom efikasnošću manipuliše sa operandima tog tipa. Za efikasnu podršku važna su sledeća dva faktora:

- instrukcije koje se koriste da obave operacije nad formatima podataka sa kojima se manipuliše
- adresni načini rada koji omogućavaju jednostavan pristup operandima sa kojima se manipuliše.

1.1 . KLASIFIKACIJA PODATAKA

Tipove podataka sa kojima računar manipuliše delimo u tri kategorije:

- *korisničko definisani podaci* - korisnik ih eksplicitno specificira u programu. Struktura ovih podataka je uglavnom određena mogućnostima programskog jezika.
- *sistemsko definisani podaci* - u toku izvršenja programa implicitno se generišu od strane računarskog sistema.
- *instrukcije* - program koji se izvršava može se posmatrati kao kompozicija podataka koji imaju svoju sopstvenu strukturu i osobine.

1.2. KORISNIČKO DEFINISANI PODACI

Ove podatke delimo na proste (skalarne), struktuirane i podatke pokazivačkog tipa.

Skalarni tipovi podataka

Svaki tip iz ove grupe podataka ima sledeće tri osobine:

a) skup vrednosti koje podatak može da uzima (domen);
b) operacije koje su nad tim podatkom dozvoljene;
c) način predstavljanja u memoriji računara.

Domen karakterišu dva aspekta: *opseg* - broj vrednosti koje podatak može da uzima; *preciznost* - rastojanje između sukcesivnih vrednosti podataka.

Celobrojne vrednosti

Celobrojne vrednosti (*integers*) se normalno koriste :

a) za brojanje;
b) kao indeksi elemenata tipa polja;
c) za predstavljanje memorijskih adresa i dr.

Kod računarskih sistema njihov opseg (*range*) je ograničen od strane arhitekture računara. Mikro procesori koriste dve prezentacije za memorisanje celobrojnih vrednosti:

a) *notacija dvoičnog komplementa* - vrednost A predstavlja se nizom od n bitova ($a_{n-1},...,a_0$) čija je vrednost

$$A = -a_{n-1}\, 2^{n-1} + \sum_{i=0}^{n-2} a_i\, 2^i$$

Sa ovom prezentacijom opseg je

$$-2^{n-1} \le A \le 2^{n-1}-1$$

Ove brojeve zovemo *označene celobrojne vrednosti.*

b) *čista binarna notacija* - vrednost A predstavlja se kao niz od n bitova ($a_{n-1},...,a_0$) čija je vrednost

$$A=\sum_{i=0}^{n-2} a_i\, 2^i$$

Kod ove prezentacije opseg dozvoljenih vrednosti je

$$0 \le A \le 2^n-1$$

i zbog toga ove brojeve zovemo *neoznačenim celobrojnim vrednostima.*

Kada je rezultat van opsega koji se može predstaviti, generiše se premašaj (*overflow*).

Pomoću 32-bitne reči, koristeći prezentaciju dvoičnog komplementa moguće je predstaviti celobrojne vrednosti u opsegu od -2^{31} do $2^{31}-1$, što iznosi 2^{32} različitih brojeva:

- negativni brojevi između $-(1-2^{-24})*2^{127}$ i $-0.5*2^{-128}$
- pozitivni brojevi između $0.5*2^{-128}$ i $(1-2^{-24})*2^{127}$
- negativni brojevi manji od $-(1-2^{-24})*2^{127}$, nazvani negativni premašaj (*negative overflow*)
- negativni brojevi veći od $-0.5*2^{-128}$, nazvani negativan podbačaj (*negative underflow*)
- nula
- pozitivni brojevi manji od $0.5*2^{-128}$, nazvani pozitivan podbačaj (*positive underflow*)
- pozitivni brojevi veći od $(1-2^{-24}9*2^{127}$, nazvani pozitivni premašaj (*positive overflow*)

2. DEFINICIJA SKUPA INSTRUKCIJE

2.1. PREDSTAVLJANJE INSTRUKCIJA

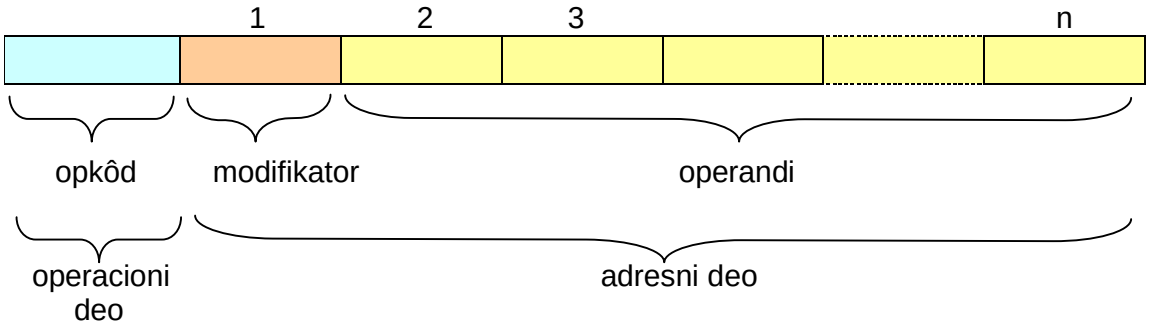
Na konvencionalnom mašinskom nivou svaka instrukcija se predstavlja sekvencom bita. Kodiranjem ovih bitova, na odgovarajući način, formira se informacija koja je potrebna radi upravljanja izvršenjem instrukcije. Ovom informacijom treba da se definiše:

a) tip operacije koja treba da se izvrši,
b) implicitna ili eksplicitna specifikacija jednog ili većeg broja operanada nad kojima se izvršava operacija, tj. specificiraju se izvorni operandi,
c) adresa gde treba da se smesti rezultat obavljene operacije, tj. specificira se odredišni operand,
d) adresa naredne instrukcije koju treba izvršiti nakon što se obavi tekuća.

Instrukciju čine sledeća dva globalna dela: operacioni i adresni, i prikazani su na slici 1.

Slika 1. Globalni delovi instrukcije

Operacioni deo zove se operacioni kod, tj. opkôd. Binarni nizovi kojima se kodiraju pojedine vrste informacija u instrukciji smeštaju se u odgovarajućim delovima koje zovemo polja. Izgled (*layout*) instrukcije zove se format instrukcije. Kod najvećeg broja skupova instrukcija postoji više od jednog formata. Bazični format n-operandskog formata instrukcije prikazan je na slici 2.



Slika 2. Bazični format instrukcije

2.1.1 ADRESNI DEO INSTRUKCIJE

Sastavni delovi adresnog dela instrukcije su polja operandi i modifikator. Modifikator uobičajno sadrži informaciju kojom se :

- opisuje način adresiranja;
- ukazuje o dodatnim uslovima koji se odnose na pristup podacima ili na način izvršenja operacije.

Da bi se smanjio obim instrukcije, a shodno tome i memorijski prostor za njeno čuvanje, neophodno je da se instrukcijom eksplicitno specificira m operanada, gde je $m<n$. Za ostale operande se smatra da su implicitno specificirani. Eksplicitna adresna polja se obično odnose na obraćanje memoriji. Ako je m maksimalan broj eksplicitnih adresa glavne memorije, tada za procesor kažemo je to m-to adresna mašina. Implicitni ulazni operandi moraju se prethodno smestiti na mesta (obično su to registri) koja su procesoru poznata pre nego što se instrukcija koja se obraća njima počne izvršavati.

2.1.2 OPERACIONI DEO INSTRUKCIJE

Neka instrukciju čine (*n*+*k*)-bitova, od kojih se *k*-bitova dodeljuje opkodu a *n*-bitova jedinstvenoj adresi. Sa *k*-bitova dobija se 2^k različitih opkodova. Alternativno, istih (*k*+*n*) bitova je moguće podeliti tako da se (*k*-1) bit dodeli opkodu a (*n*+1) bit adresi, što znači da je broj instrukcija prepolovljen ali je broj memorijskih lokacija udvostručen, ili je za isti iznos memorije, rezolucija koja se odnosi na pristup memoriji, dva puta veća. Sa (*k*+1)-bitnim opkodom i (*n*-1)-bitnom adresom broj opkodova je dva puta veći, ali je broj dostupnih adresibilnih lokacija, prepolovljen. Tehnika kodiranja koja se bazira na proširenju broja bitova opkoda zove se *kodiranje sa proširenjem.*

2.2 . DISKUSIJA U VEZI IZBORA FORMATA

Programeri uvek preferiraju veći broj opkodova, veći broj operanada, veći broj adresnih načina rada i veći adresni opseg. Sve navedene stavke ukazuju da je potrebno koristiti veći broj bitova po instrukciji što znači da će instrukcije biti duže. Duže instrukcije zauzimaju veći memorijski prostor i za duže vreme se pribavljaju iz memorije, a to indirektno zkazuje da ih CPU sporije izvršava.

Za zadatu dužinu instrukcije neophodno je napraviti kompromis između brojeva opkodova i adresnih mogućnosti. Veći broj opkodova zahteva veći broj bitova za opkod polja, što sa druge strane znači smanjenje broja bitova koji su dostupni za adresiranje. Jedno od rešenja je korišćenje opkodova promenljive dužine. Kod ovog pristupa postoji minimalna opkod dužina, a za neke opkodove moraju se specificirati dodatni bitovi. Faktori koji imaju direktan uticaj na dužinu polja za adresiranje su:

a) *broj adresnih načina rada* - kod nekih opkodova korišćenje određenog adresnog načina rada se uvek implicitno podrazumeva, a kod drugih adresni način rada se specificira eksplicitno.
b) *broj operanada* - instrukcije današnjih računara omogućavaju specifikaciju jednog, dva, ili tri operanada.
c) *specifikacija internih registara u odnosu na memoriju* - skoro svi savremeni mikroprocesori imaju ugrađeni interni skup registara. Broj ovih registara se obično kreće od 8 do 32, a kod nekih je i veći. Veći broj registara zahteva duže polje.
d) *adresni opseg* - kod specifikacija adresa pomoću kojih se vrši obraćanje memoriji, veličina adresnog opsega je u direktnoj vezi sa brojem adresnih bitova. Da bi se smanjio ovaj broj, umesto direktnog koristi se adresiranje sa razmeštajem.
e) *adresna granularnost* - sa ciljem da se smanji broj adresnih bitova, kod velikog broja mikroprocesora adresiranje memorije je izvedeno na nivou 8-, 16- ili 32-bitnih podataka.

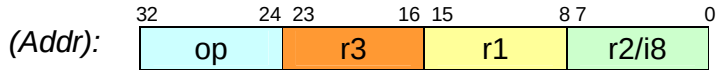
2.3. FORMAT INSTRUKCIJA

DP32 je 32-bitni procesor sa jednostavnim skupom instrukcija.

Skup instrukcija DP32 je podeljen na veliki broj kodiranih podataka koji uključuje: programsko-upravljačke, aritmetičke, logičke i prenos podataka instrukcije.

2.3.1. ARITMETIČKE I LOGIČKE INSTRUKCIJE

Sve aritmetičke i logičke instrukcije su dužine 32-bita.U konkretnom slučaju pojam reč se odnosi na obim podataka od 32-bita. Format je prikazan na slici 3.



Slika 3. Format aritmetičke i logičke instrukcije

gde je:

- op - op-kôd polje,
- r3 - adresa odredišnog registra,
- r1 i r2 - adrese izvorišnih registra,
- i8 - neposredna celobrojna vrednost u prezentaciji dvoičnog komplementa.

Efekat aritmetičke i logičke funkcije prikazane su na Tabeli 1. Logičke funkcije su date u crvenoj boji.

Tabela 1. Aritmetičke i logičke instrukcije DP32

Instruction	Name	Function	opcode
Add	add	r3 ← r1 + r2	X ``00``
Sub	subtract	r3 ← r1 - r2	X ``01``
Mul	multiply	r3 ← r1 × r2	X ``02``
Div	divide	r3 ← r1 ÷ r2	X ``03``
Addq	add quick	r3 ← r1 + i8	X ``10``
Subq	substract quick	r3 ← r1 - i8	X ``11``
Mulq	multiply quick	r3 ← r1 × i8	X ``12``
Divq	divide quick	r3 ← r1 ÷ i8	X ``13``
Land	logical and	r3 ← r1 & r2	X ``04``
Lor	logical or	r3 ← r1 ∣ r2	X ``05``
Lxor	logical exclusive or	r3 ← r1 ⊕ r2	X ``06``
Lmask	logical mask	r3 ← r1 & -r2	X ``07``

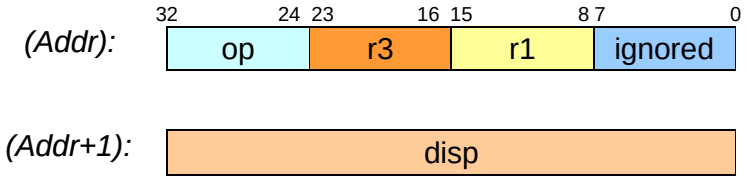
2.3.2. LOAD I STORE INSTRUKCIJE

Instrukcije *Load* i *Store* prikazane su u Tabeli 2.

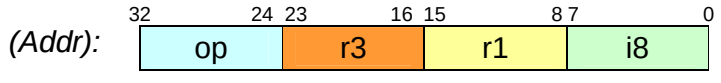
Tabela 2. *Load* i *Store* instrukcije procesora DP32

Instruction	Name	Function	opcode
Ld	load	r3 ← M[r1 + disp32]	X ``20``
St	store	M[r1 + disp32] ← r3	X ``21``
Ldq	load quick	r3 ← M[r1 + i8]	X ``30``
Stq	store quick	M[r1 + i8] ← r3	X ``31``

U zavisnosti od vrednosti razmeštaja instrukcije tipa *Load* i *Store* imaju dva formata prikazanih na slici 4:



a)



b)

Slika 4. Format naredbi *Load* i *Store*

a) format sa većim razmeštajem; b) format sa manjim razmeštajem

gde je:

- op - op-kôd,
- r3- adresa registara za punjenje i pamćenje,
- r1 - indeksni registar,
- disp - veći neposredni razmeštaj,
- i8 - manji neposredni razmeštaj,
- ignore - polje čiji se sadržaj ignoriše

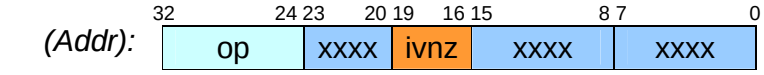
2.3.3. BRANCH INSTRUKCIJE

Postoje četiri *Branch* instrukcije, prikazane u Tabeli 3.

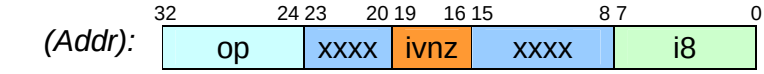
Tabela 3. Branch instrukcije procesora DP32

Instruction	Name	Function	opcode
Br-ivnz	branch	if *cond* then PC ← PC + disp32	X ``40``
Brq-ivnz	branch quick	if *cond* then PC ← PC + i8	X ``50``
Bi -ivnz	branch indexed	if *cond* then PC ← r1 + disp32	X ``41``
Biq-ivnz	branch indexed quick	if *cond* then PC ← r1 + i8	X ``51``

Format osnovne instrukcije *Branch* i *indexed Branch*-a prikazan je na slikama 5 i 6.

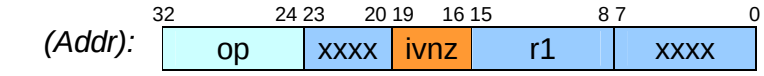


a)

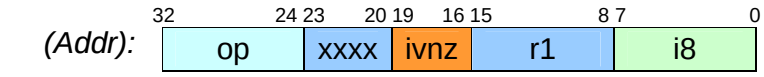


b)

Slika 5. Format *Branch*-a
a) format osnovnog *Branch*-a; b) format *quick Branch*-a



a)



b)

Slika 6. Format *indexed Branch*-a
a) format *indexed Branch*-a; b) format *quick indexed Branch*-a

gde je :

- op - op-kôd,
- disp - veći neposredni razmeštaj,
- i8 - manji neposredni razmeštaj,
- r1 - indeksni registar i
- ivnz - stanje maske.

Branch se izvršava ako je:

cond = ((V & v) | (N & n) | (Z & z)) = i.

2.4. KLASE INSTRUKCIJA

Instrukcije možemo podeliti u sledeće grupe:

1. **prenos-podataka** (Ld, Ld_q, St, St_q) – pomoću ovih instrukcija vrši se kopiranje informacija iz jedne lokacije u drugu. Lokacije mogu pripadati registrima procesora ili memoriji,
2. **aritmetičke** (Add, Sub, Mul, Div, Add_q, Sub_q, Mul_q, Div_q) – obavljaju aritmetičke operacije nad numeričkim podacima,
3. **logičke** (OR, AND, XOR, MASK) – uključuju Booleove i druge nenumeričke operacije.
4. **programsko-upravljačke** (Br, Bi, Br_q, Bi_q) – menjaju sekvencu programskog izvršenja.

3. STRUKTURA SISTEMA

3.1. ARHITEKTURA PROCESORA DP32

DP32 je 32-bitni procesor sa jednostavnim skup instrukcija. On sadrži registre koji su prikazani na slici 7. DP32 ima:

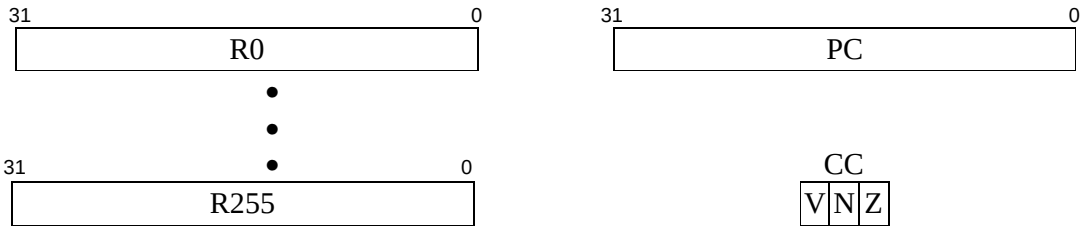
- 256 registara opšte namene (R_0 - R_{255}),
- programski brojač (*Program Counter* - PC) i
- registar marker-uslova (*Condition Code* - CC).

Registrima opšte namene se pristupa programski. Sadržaju registra PC se pristupa instrukcijama grananja. Naredbe za postavljanje i brisanje bitova registra CC ***Set*** bit***_i*** ***Reset*** bit***_i*** ne postoje.

U toku *reset*-a PC je postavljen na nulu, a stanje ostalih registara ostaju nedefinisana. Po konvenciji, u registru R_0 se čuva nula i on se može jedino čitati. Upis vrednosti nule u registar R_0 u konkretnom slučaju se izvodi softverski, određenom naredbom koja sledi odmah nakon *reset*-a sistema, što znači da njegova vrednost nije hardverski definisana.

Obim memorije i obim adrese je 32-bitni. Instrukcije su multipli 32-bitnih reči. Instrukcije su smeštene u memoriji. PC registar sadrži adresu naredne instrukcije koja će se izvršiti. Nakon pribavljanja svake instrukcione reči PC se inkrementira za jedan i ukazuje na narednu reč.

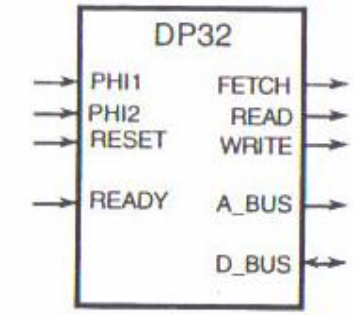
Bitovi trobitnog CC registra (vidi sliku 7.) se ažuriraju nakon izvršenja svake aritmetičke ili logičke instrukcije. Marker bit Z (nula-zero) je set-ovan ako je rezultat operacije nula. Marker bit N (negativan-negative) je set-ovan ako je rezultat aritmetičke operacije negativan, a nedefinisan nakon logičke operacije. Marker bitV (premašaj-overflow) je set-ovan ako je rezultat aritmetičke operacije premašio granicu zadate celobrojne vrednosti u prezentaciji dvoičnog komplementa, a nedefinisan nakon logičke operacije.



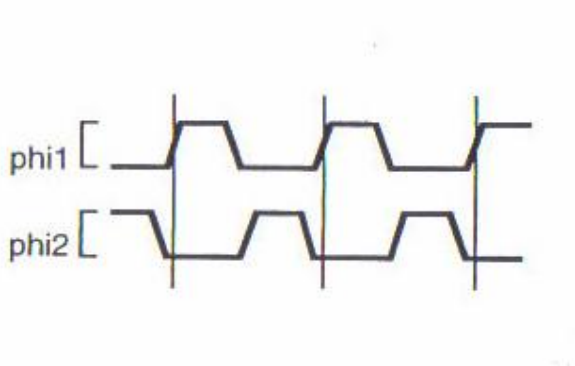
Slika 7.Registri procesora DP32

3.2. ARHITEKTURA MAGISTRALE

Procesor DP32 komunicira sa memorijom preko sinhrone 32-bitne sistemske magistrale. Spoljni signali DP32 prikazani su na slici 8.



Slika 8. Globalni dijagram procesora DP32 na nivou ulazno-izlaznih signala



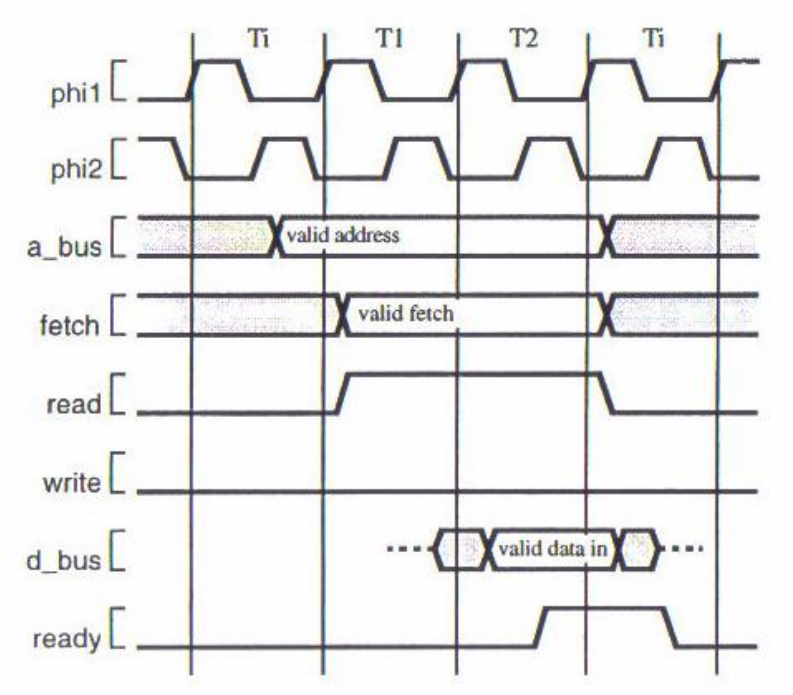
Slika 9. Talasni oblik taktnih impulsa procesora DP32

Pomoću taktnih impulsa, phi1 i phi2 obezbeđeno je dvofazno taktno pobuđivanje procesora bez preklapanja. Talasni oblici taktnih impulsa prikazani su na slici 9.

Svaka perioda taktnog impulsa phi1 definiše jedno od stanja na magistrali. Magistrala se može naći u stanju Ti (pasivno-`idle`), T1 ili T2. Transakcije na magistrali čine: stanje T1, iza koga sledi jedno ili više stanja T2, sa umetnutim Ti stanjima između transakcija (za više detalja vidi sliku 10 ili 11).

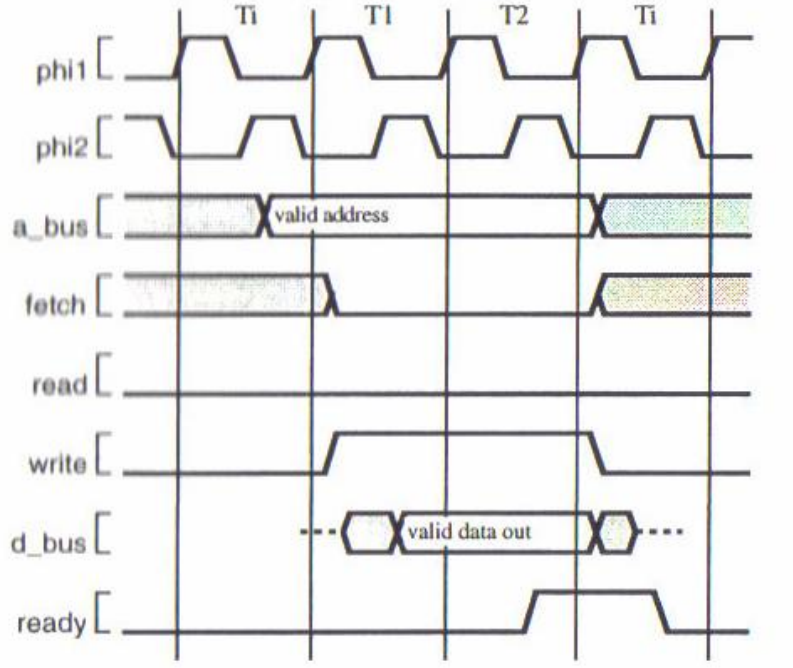
Izlaz (tzv. port) A_bus-a je 32-bitna adresna magistrala. Port D_bus je 32-bitna dvosmerna magistrala podataka. Portovi READ i WRITE definišu transakcije tipa čitanje (*read*) i upisa (*write*) na magistrali. Port FETCH je statusni signal, koji ukazuje da je u toku transakcija čitanja na magistrali, a odnosi se na fazu pribavljanje instrukcije. Stanje na ulazu READY definiše memorija sa ciljem da ukaže da je pročitani podatak dostupan, ili da će se on korektno upisati.

Vremenski dijagram transakcije tipa read na magistrali prikazan je na slici 10. U toku stanja Ti, radi iniciranja transakcije, procesor postavlja memorijsku adresu na adresnu magisralu. Naredno stanje je stanje T1. Nakon rastuće ivice impulsa phi1 procesor aktivira upravljački signal read, ukazujući da je adresa validna i da memorija može da otpočne sa transakcijom *read*. Pored toga, procesor aktivira signal fetch kada se obavlja aktivnost čitanja. Signal write je neaktivan za vreme transakcije *read*. U toku tekućeg stanja T1 i narednog stanja T2, pristupa se specificiranim podacima u memoriji. Kada su oni validni memorija ih postavlja na magistralu podataka. Nakon završetka pristupa podacima (odgovara stanju T2*)* aktivira se signal ready. Procesor prihvata podatke i završava transakciju. Sa druge strane, ako memorija nije još dostavila podatke do kraja stanja T2, ona postavlja signal ready na *false*. Kao odziv, procesor ponavlja stanje T2, sve dok se ne detektuje da je signal ready *true*. Na ovaj način, sporija memorija može vremenski da produži transakciju sve do trenutka kada memorija ne generiše validni podatak i procesor ga prihvati. Na kraju transakcije procesor postavlja upravljačke izlaze na *default*-nu vrednost, dok memorija komplementira signal ready i deaktivira linije za podatke na magistrali podataka. Nakon toga, procesor nastavlja sa pasivnim stanjem Ti sve dok ne otpočne naredna transakcija.



Slika 10.Transakcija *read* magistrale DP32

Vremenski dijagram transakcije *write* na magistrali prikazan je na slici 11. U ovom slučaju transakcija počinje tako što procesor postavlja adresu na adresnu magistralu u toku stanja T_i. Nakon pojave prednje ivice phi1 tokom stanja T1, procesor deaktivira fetch, a aktivira write. Signal read je *false* u toku cele transakcije. U toku stanja T1 procesor postavlja podatke koje treba upisati u memoriju na magistrali podataka. Memorija može prihvatiti ove podatke tokom T1 i narednih nekoliko T2 stanja. Ako se do kraja stanja T2 završava ciklus *write*, memorija aktivira signal ready. Procesor tada završava transakciju tipa *write* i produžava sa generisanjem stanja Ti. Nakon toga linije za podatke na magistrali podataka se deaktiviraju od strane DP32, a memorija komplementira stanje signala ready. Ako memorija nije imala dovoljno vremena da završi sa ciklusom *write* do kraja stanja T2, ona zadržava signal ready u stanje *false*, čime se ciklus write produžava za integer broj stanja T2, tj. procesor će ponavljati stanje T2 sve dok se detektuje ready *true*.



Slika 11.Transakcija *write* magistrale DP32

ILUSTRATIVNI PRIMER

Za instrukciju **ADD r3, r1, r2** odrediti:

a) klasu instrukcije,
b) op-kod instrukcije,
c) funkciju instrukcije i
d) nacrtati format instrukcije

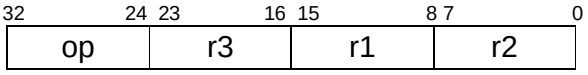
ODGOVOR:

a) ADD instrukcija pripada aritmetičkoj klasi instrukcija - obavlja aritmetičke operacije nad numeričkim podacima.

b) Op-kôd :'00'

c) Funkcija instrukcije je sabiranje: $r_3 \leftarrow r_1 + r_2$

d) Format instrukcije:



ZADACI:

Za datu programsku sekvencu za svaku od instrukcija odrediti:

1. klasu instrukcije,
2. op-kod instrukcije,
3. funkciju instrukcije i
4. nacrtati format instrukcije

Student 1:
Instrukcija: Sub r3, r1, r2
Instrukcija: Land r3, r1, r2
Instrukcija: Ld r3, r1, $disp_{32}$
Instrukcija: Br PC, PC, $disp_{32}$

Student 2:
Instrukcija: Mul r3, r1, r2
Instrukcija: Lor r3, r1, r2
Instrukcija: St r1, $disp_{32}$, r3
Instrukcija: Brq PC, PC, i_8

Student 3:
Instrukcija: Div r3, r1, r2
Instrukcija: Lxor r3, r1, r2
Instrukcija : Ld_q r3, r1, i_8
Instrukcija: Biq PC, r1, i_8

Student 4:
Instrukcija: Add_q r3, r1, i_8
Instrukcija: Lmask r3, r1, r2
Instrukcija: St_q r1, i_8, r3
Instrukcija: Bi PC, r1, disp32

Student 5:
Instrukcija: Sub_q r3, r1, i_8
Instrukcija: Land r3, r1, r2
Instrukcija: Ld r3, r1, $disp_{32}$
Instrukcija: Br PC, PC, $disp_{32}$

Student 6:
Instrukcija: Mul_q r3, r1, i_8
Instrukcija: Lor r3, r1, r2
Instrukcija: Ld_q r3, r1, i_8
Instrukcija: Bi PC, r1, disp32

Student 7:
Instrukcija: Div_q r3, r1, i_8
Instrukcija: Lmask r3, r1, r2
Instrukcija: St r1, $disp_{32}$, r3
Instrukcija: Biq PC, r1, i_8

Student 8:
Instrukcija: Land r3, r1, r2
Instrukcija: Sub_q r3, r1, i_8
Instrukcija: St_q r1, i_8, r3
Instrukcija: Brq PC, PC, i_8

Student 9:
Instrukcija: Lor r3, r1, r2
Instrukcija: Add r3, r1, r2
Instrukcija: St r1, $disp_{32}$, r3
Instrukcija: Br PC, PC, disp32

Student 10:
Instrukcija: Lxor r3, r1, r2
Instrukcija: Add_q r3, r1, i_8
Instrukcija : Ld_q r3, r1, i_8
Instrukcija: Bi PC, r1, disp32

Student 11:
Instrukcija: Ld r3, r1, $disp_{32}$
Instrukcija: Mul r3, r1, r2
Instrukcija: Land r3, r1, r2
Instrukcija: Biq PC, r1, i_8

Student 12:
Instrukcija: St r1, $disp_{32}$, r3
Instrukcija: Div_q r3, r1, i_8
Instrukcija: Lmask r3, r1, r2
Instrukcija: Brq PC, PC, i_8

Student 13:
Instrukcija: Ld_q r3, r1, i_8
Instrukcija: Div r3, r1, r2
Instrukcija: Land r3, r1, r2
Instrukcija: Brq PC, PC, i_8

Student 14:
Instrukcija: St_q r1, i_8, r3
Instrukcija: Lmask r3, r1, r2
Instrukcija: Mul r3, r1, r2
Instrukcija: Br PC, PC, disp32

Student 15:
Instrukcija: Br PC, PC, $disp_{32}$
Instrukcija: Lor r3, r1, r2
Instrukcija: Sub_q r3, r1, i_8
Instrukcija: St r1, $disp_{32}$, r3

Student 16:
Instrukcija: Bi PC, PC, i_8
Instrukcija: Lxor r3, r1, r2
Instrukcija: Add r3, r1, r2
Instrukcija: Ld r3, r1, $disp_{32}$

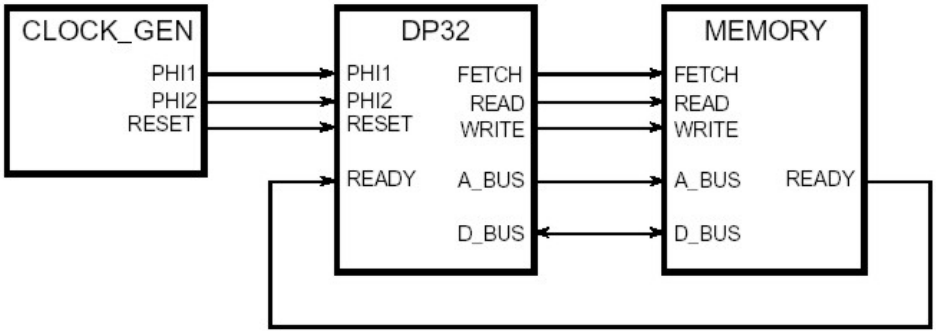
VEŽBA BROJ 2:

SIMULATOR DP32

1. TEST KOLO

Testiranje ponašanja modela procesora DP32 vrši se njegovim povezivanjem na kolo za testiranje, kako je to prikazano na slici 1. Komponenta CLOCK_GEN generise dvo-faznu taktnu pobudu i signal RESET kojim se precesor DP32 postavlja u inicijalno stanje. Memorija se koristi za čuvanje podataka i programa. Naglasimo da se u svakoj memorijskoj lokaciji čuva po jedna 32-bitna reč (četiri bajta).

Zadatak studenta je da kreira modele ponašanja za komponente DP32 i MEMORY, a zatim, za potrebe verifikacije dizajna, poveže ih sa strukturnim opisom *test-bench* programa.



Slika 1. Kolo test bench-a za DP32

Entitet CLOCK_GEN čine dve formalne `generic` konstante. Vremenski period *Tpw* odgovara širini taktnog impulsa za svaki od signala PHI1 i PHI2. To je vremenski period za koji ovi impulsi imaju vrednost '1'. *Tps* je impuls razdvajanja, tj. to je vreme od trenutka promene taknog signala sa '1' na '0', do trenutka promene drugog taknog signala sa '0' na '1'. Na osnovu ovih vrednosti taktna perioda odgovara dvostrukom zbiru vremena kada su impulsi aktivni i kada postoji razdvajanje (T_{clk}=2 *Tpw* +2 *Tps*).

Arhiktekturu taktnog genereratora, opisanu u VHDL-u, čine dva konkurentna iskaza od kojih jedan se koristi za aktiviranje reset signala, a drugi za aktiviranje taktih signala. Drajver reset signala generiše vrednost '1' kada je signal ***reset*** aktiviran (to je proces simulacije u fazi inicijalizacije). Nakon toga ovaj signal se postavlja na vrednost '0' u trajanju od dva taktna perioda. Kada se aktivira proces drajvera takta trenutno generiše impuls PHI1, nakon čega sledi impuls PHI2. Zatim se vrši suspenzija taktnog perioda. Ovaj proces je repetitivan.

2. PRIMER SIMULACIJE

Verifikacija rada DP32 procesora se vrši uz pomoć *test bench* programa. Da bi se ostvario ovaj cilj poziva se program simulacioni monitor a *test bench* program se puni u memoriju.

Na slici 2. prikazan je deo listinga programa nazvan `counter` kreiran na asemblerskom jeziku procesora DP32. Program `counter` inicijalizira registar `r0` na nulu (asemblerska makro naredba `initr0`, linija 8, realizuje se pomoću instrukcije *`Lmask`* `r0, r0, r0`, vidi tabelu 1). Telo petlje čine linije 12, 13, 14. U okviru tela petlje vrši se inkrementiranje brojača petlje tj. registra `r2`, a zatim se od vrednosti ovog registra oduzima 10 (granica petlje). Instrukcijom u liniji 16 granamo se na novu iteraciju u okviru petlje, a instrukcijom u liniji 15 granamo se novi ciklus izvršenja petlje. Vrednosti u zagradama (`counter, start` i `loop`) predstavljaju adrese

instrukcije, a heksadecimalne vrednosti u uglastim zagradama predstavlja binarno izvršive kodove asemblerskih instrukcija.

Prva kolona prikazuje broj linija za unos programa koji generiše editor programa.

U okviru druge, treće i četvrte kolone prikazuje se adresa memorijske lokacije odgovarajuće instrukcije ili makro naredbe.

U petoj, šestoj, sedmoj i osmoj koloni je izvršivi kod u binarnom obliku svake od instrukcija ili makro naredbe, op-kôdovi obima jedne reči (kakav je slučaj linija 13) ili dve reči (linija 12).

Kolona devet je dodeljena direktivama (`include`, `begin` i `end`) i labelama instrukcije (`start`, `loop`, `counter`)

Kolona deset odgovara mnemoniku naredbe ili makro instrukcije.

Kolone 11-15 specificiraju izvorne i odredišne operande ili adresu grananja. Npr. u liniji 13 su izvorni i odredišni operand, a u liniji 15 adresa grananja.

Kolona 16 označava komentar koji počinje znakom '!' i važi do kraja linije. Uočimo da su linije 3 i 4 rezervisane za komentar, a linije 2, 6, 17 su prazne.

I, II, III, IV V, VI, VII, VIII IX, X, XI, XII, XIII, XIV, XV, XVI

```
 1.                                 include dp32.inc $
 2.
 3.                                 !!!   conventions:
 4.                                 !!!       r0 = 0
 5.                                 !!!       r1 scratch
 6.
 7.                                 begin
 8.   (     0) [07000000         ]    initr0
 9.                                 start:
10.   (     1) [10020000         ]    addq(r2, r0, 0)    ! r2 := 0
11.                                 loop:
12.   (     2) [21020000 00000008]    sta(r2, counter)   ! counter := r2
13.   (     4) [10020201         ]    addq(r2, r2, 1)    ! increment r2
14.   (     5) [1101020A         ]    subq(r1, r2, 10)   ! if r2 = 10 then
15.   (     6) [500900FA         ]    brzq(start)        !   restart
16.   (     7) [500000FA         ]    braq(loop)         ! else next loop
17.
18.                                 counter:
19.   (     8) [00000000         ]    data(0)
20.                                 end
```

Slika 2. Izlistani test program asemblera

Napomena: Asembler procesora DP32 nije osetljiv na velika i mala slova, tj. instrukcije `Add` i `add` imaju ekvivalentno dejstvo.

Korak 1

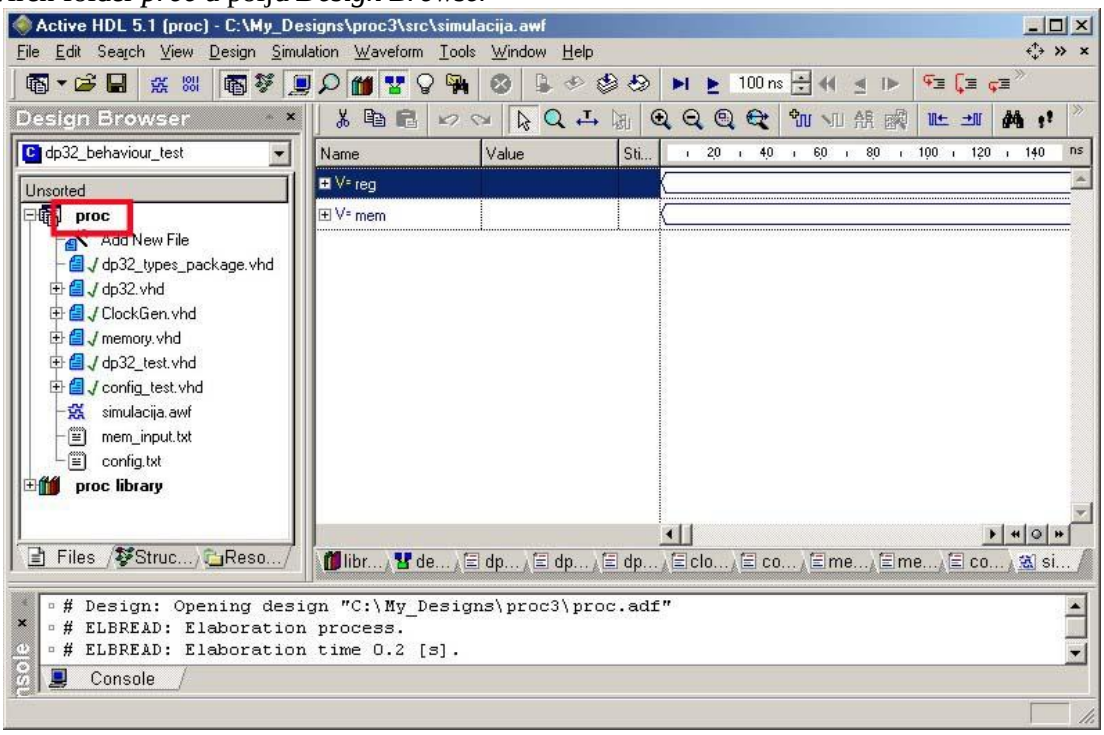
Početno usvajamo da je program za sintezu DP32 procesora prethodno instaliran u memoriju računara. Hardverska sinteza procesora DP32 i simulacija njegovog rada se obavlja pokretanjem programa Active HDL.

Za pokretanje programa Active HDL potrebno je obaviti sledeće aktivnosti:

- izabrati programsku ikonu *My Computer* lociranu na Desktop-u.
- aktivirati *My Computer* – aktiviranje se izvodi dvostrukim klikom levog tastera miša na odabranu ikonu
- na C: particiji aktivirati *My _Designs*, gde se nalazi folder *proc*

- otvoriti folder i aktivirati ikonu *proc.adf* (Active HDL.Design)

Startovanjem ikone pojavljuje se aplikacioni prozor prikazan na slici 3. Crvenom bojom je uokviren folder *proc* u polju *Design Browser*



Slika3. Izgled prozora programa za simulaciju *Active HDL 5.1*

U ovom prozoru može se posmatrati:

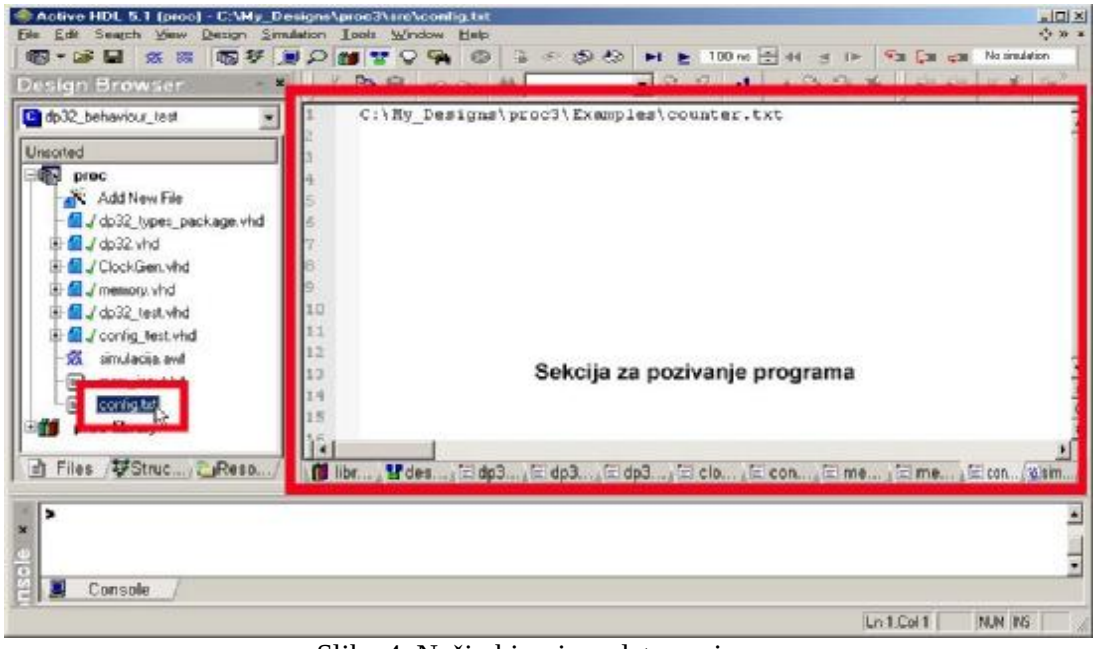
a) program kreiran za potrebe simulacije,

b) listanje sadržaja memorije i

c) tok simulacije (tj.generisane vrednosti i vremenski dijagram izabrane simulacije).

- Selektovati i kliknuti levim tasterom miša na *confing.txt* da bi otvorili sekciju pomoću koje biramo zadati primer (slika 4.)

U izabranoj simulaciji prikazano je izvršenje programa `counter`.



Slika 4. Način biranja zadatog primera

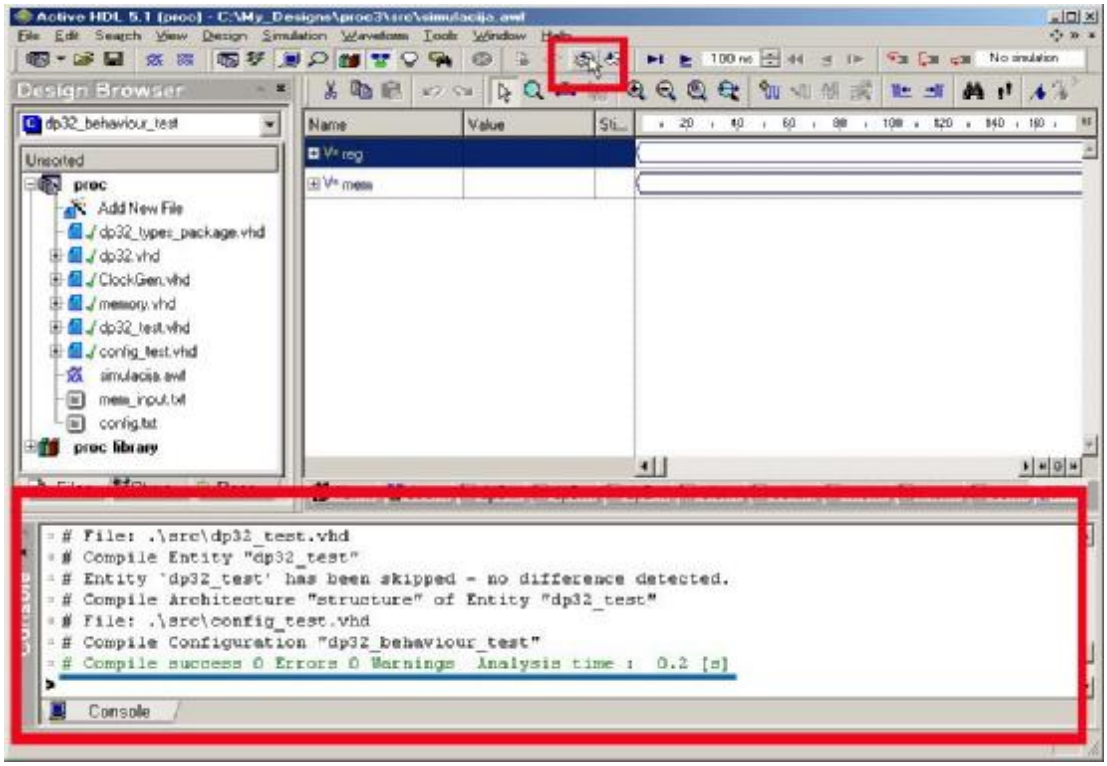
Ovim smo završili učitavanje zahtevane simulacije programa `counter`.

Korak 2

Ovaj korak se odnosi na fazu pripreme izvršenja simulacije koja uključuje sledeće aktivnosti:

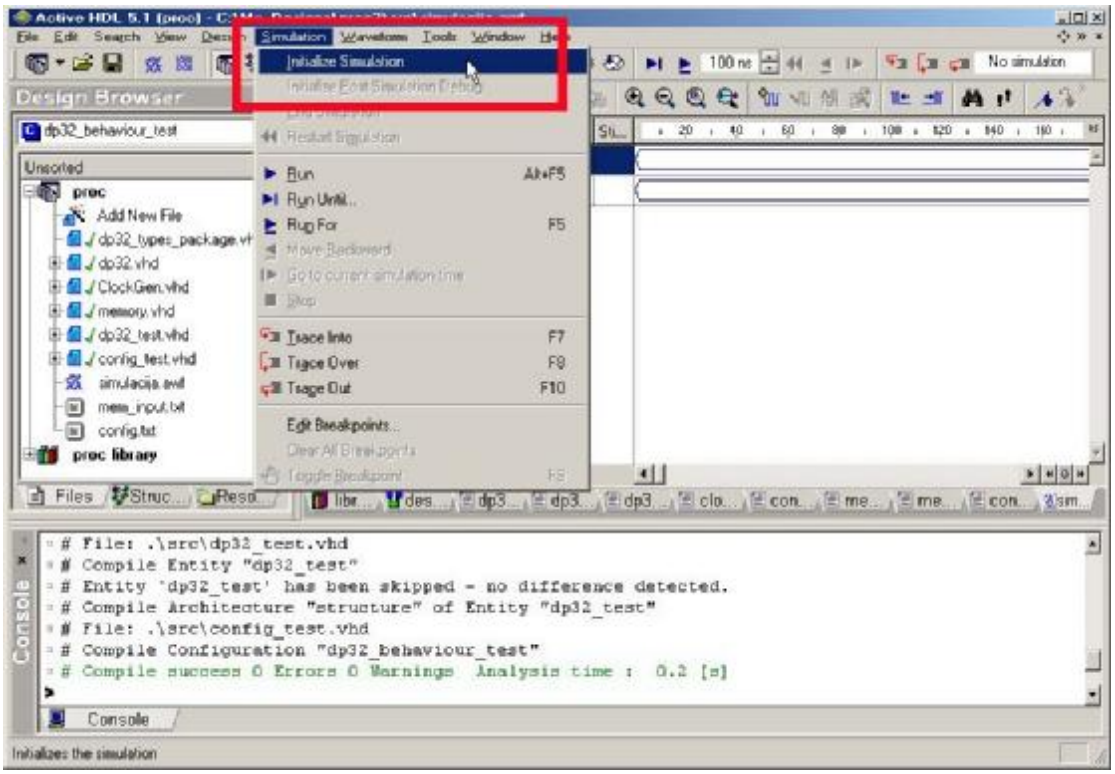
1. **Kompajliranje**: aktivira se klikom na ikonicu *Compile All* koji je locirana u Tool Bar-u (obeleženo na slici 5.). Izveštaj da je kompajliranje uspešno završeno ili nije, broj sintaksnih grešaka detektovanih u toku kompajliranja, i vreme potrebno za analizu (podvučeno plavom linijom na slici 5.) prikazuje se u sekciji *Console* (slika 5.).

Napomena: *Console* je sekcija prikaza koja se koristi za izveštaj bilo koje operacije koja se vrši nad tekućom simulacijom. (Pri radu obratiti pažnju na izveštaje u *Console*.)



Slika 5. Izvršenje komande *Compile All*

2. **Inicijalizacija simulacije**: Nakon što je utvrđeno da je kompajliranje uspešno završeno i da nema grešaka u programu aktivira se komanda *Initialize Simulation.* U konkretnom slučaju prvo se selektuje *Simulation* na Menu Bar-u, a nakon otvaranja padajućeg menija bira se *Initialize Simulation* (slika 6.).

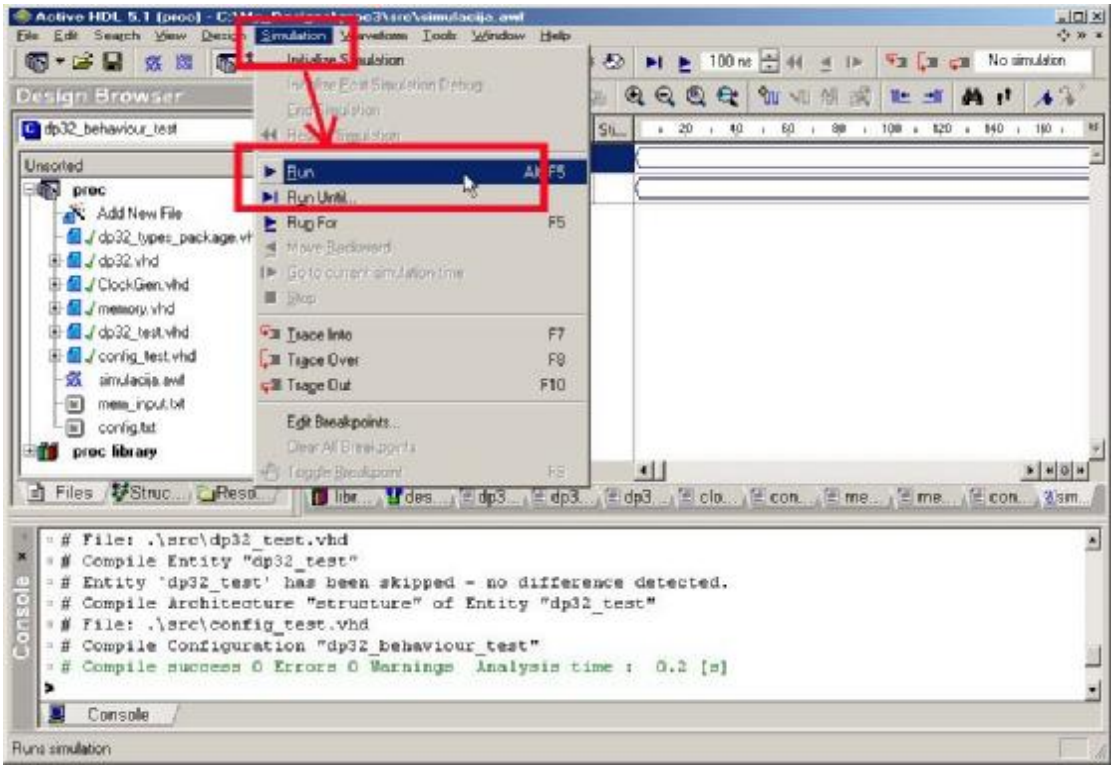


Slika 6. Prikaz načina inicijalizacije simulacije

Korak 3

Nakon inicijalizacije simulacije sledi pokretanje koje uključuje sledeću aktivnost.

- Izborom *Simulation* na Menu Bar-u otvara se padajući meni (slika 7.) koji sadrži opciju *Run*. Selekcijom i aktiviranjem opcije *Run* startuje se simulacija.



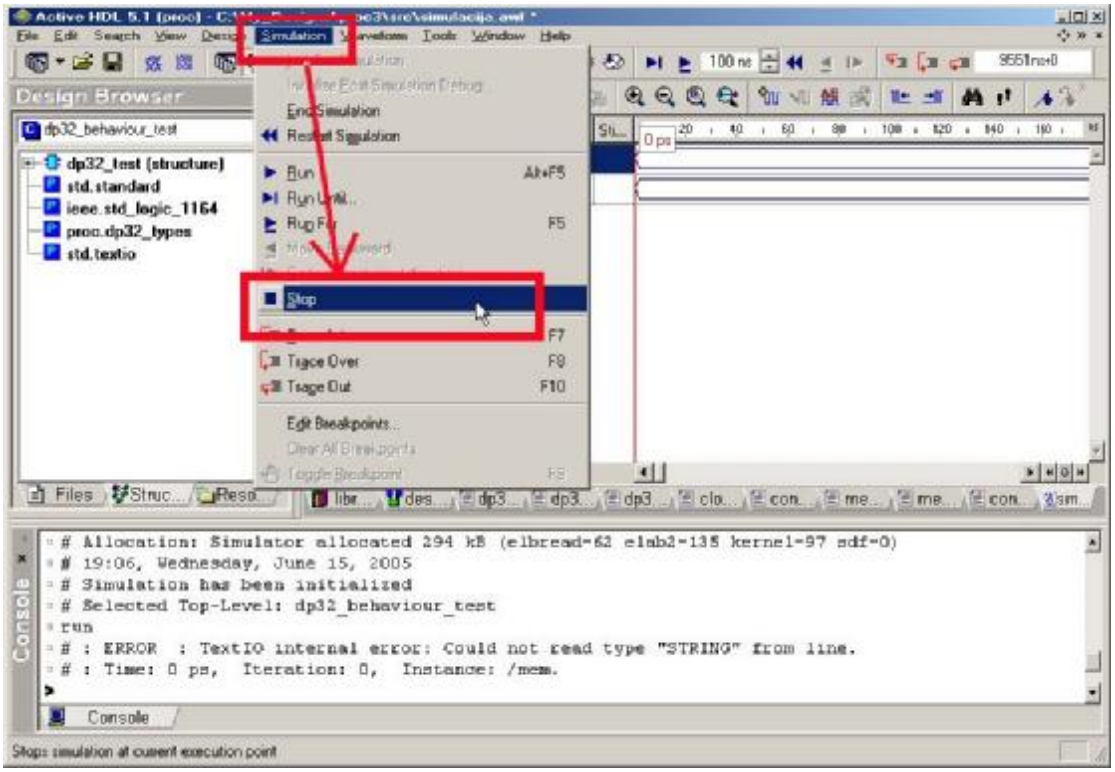
Slika 7. Prikaz opcije *Run*

U prozoru koji je markiran kao *No simulation*, može se pratiti vreme trajanja simulacije (slika 7). Istovremeno u sekciji *Console* promenjen je sadržaj, odnosno sekcija sada daje izveštaj da je u toku proces simulacije.

Korak 4

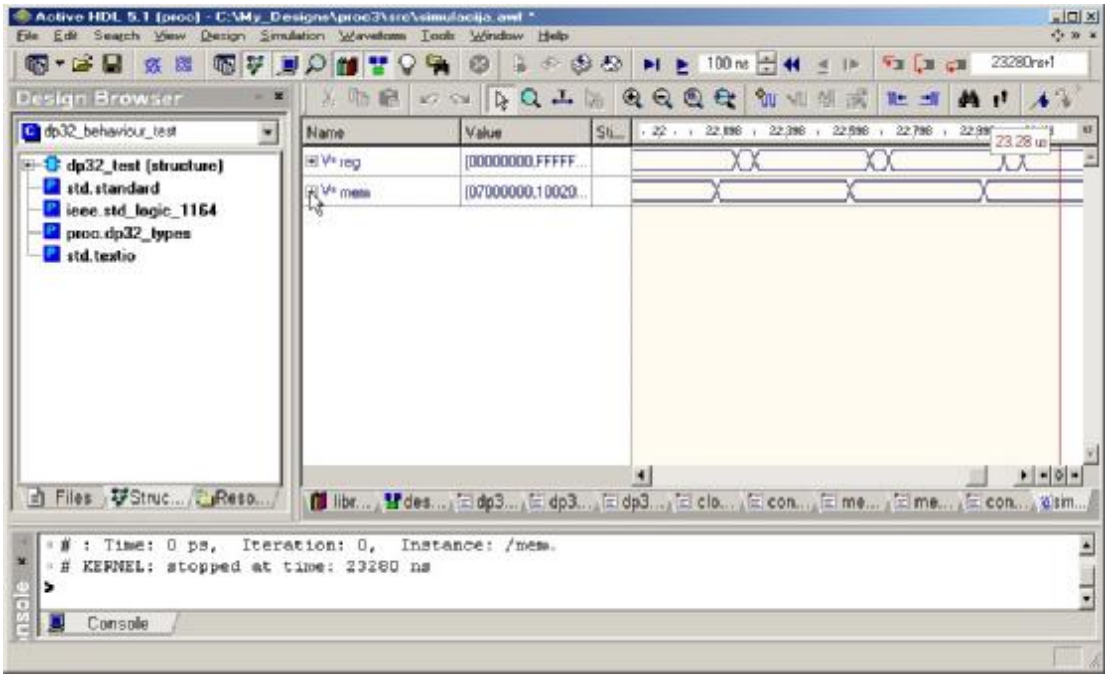
Proces simulacije se može zaustaviti sa ciljem da se proveri stanje internih registara procesora i memorije. Na osnovu pročitanih stanja internih registara i memorije može se proveriti da li program ispravno funkciniše ili ne. Aktivnost koja prati ovaj korak je sledeća

- **Zaustavljanje simulacije:** selekcijom polja *Simulation* na Menu Bar-u otvara se padajući meni (slika 7.) koji sadrži opciju *Stop*. Aktiviranjem opcije *Stop* zaustavlja se simulacija.



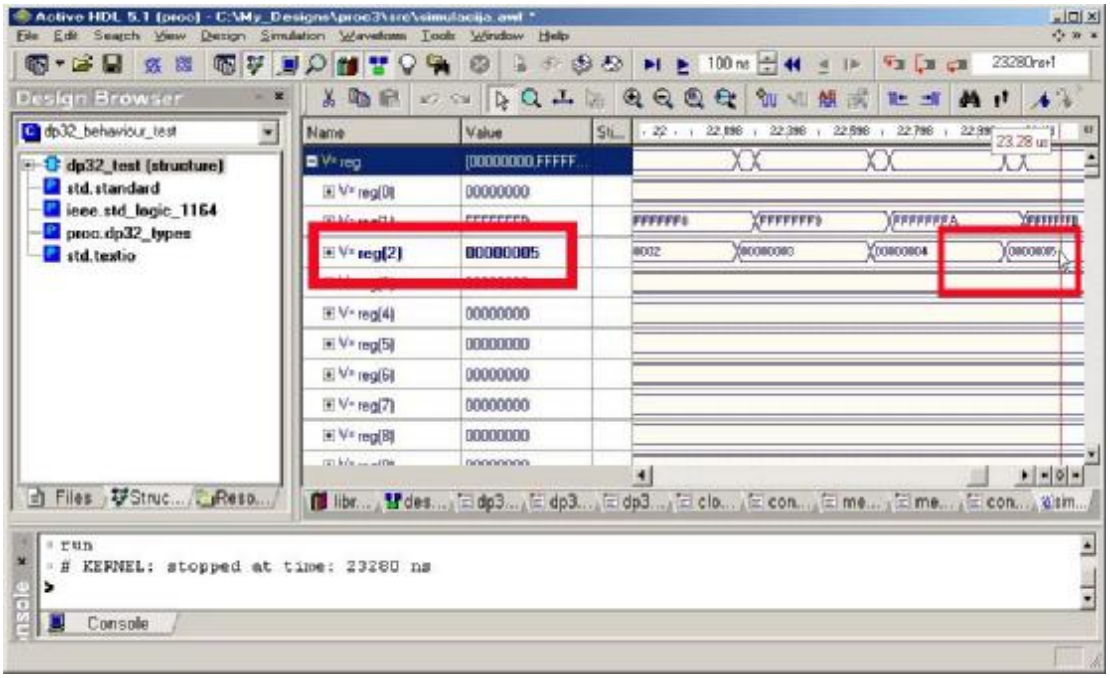
Slika 8. Prikaz opcije *Stop*

Na slici 9. prikazan je izgled prozora nakon zaustavljanja simulacije. (U sekciji *Console* treba uočiti izveštaj o vremenu trajanja simulacije do trenutka zaustavljanja)



Slika 9. Izgled prozora nakon što je izvršena opcija *Stop*

Takođe treba uočiti i vrste V=reg i V=mem u polju *Tree*. Klikom na simbol '+' koji prethodi V=reg prikazuje se listing sadržaja registara procesora DP32, a klikom na simbol '+' koji prethodi V=mem prikazuje se listing sadržaja memorije (slika 9).



Slika 10. Stanje registra

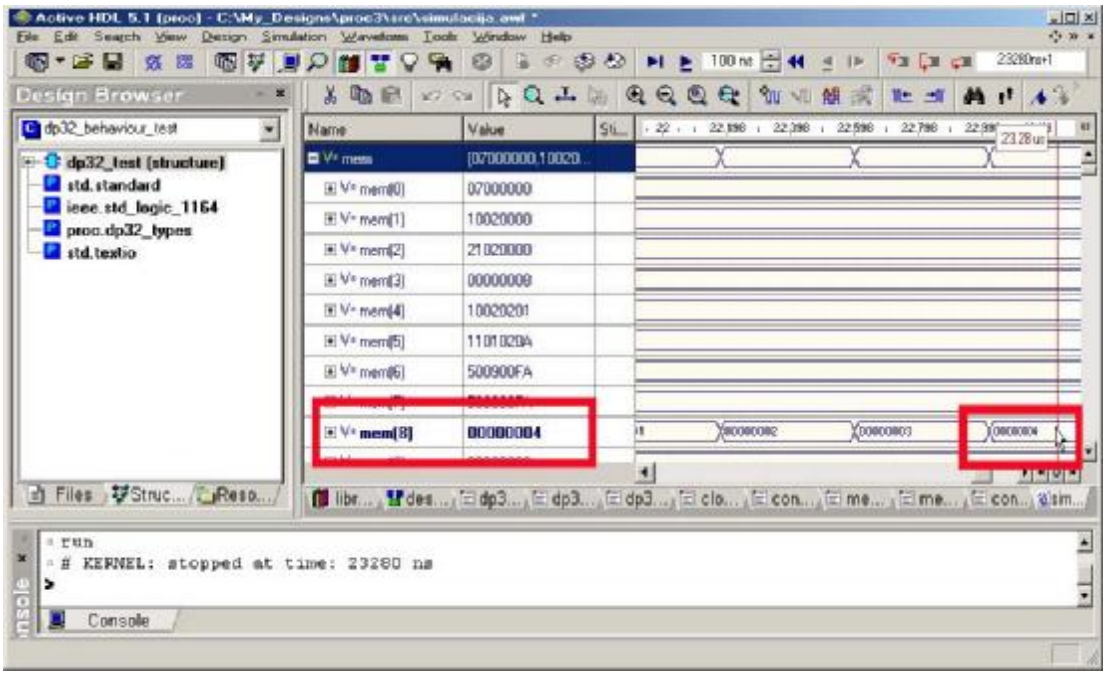
U programu `counter` u registru reg(2) privremeno se pamti promenljiva `v_counter`, a stanje memorijske lokacije mem(8) odgovara sadržaju reg(2) na početku svake iteracije (slika 1).

Uočiti da je sadržaj reg(2)=00000005, a da je sadržaj mem(8)=00000004 (slika 10 i slika 11). Razlika koja se u ovom slučaju detektuje između sadržaja reg(2) i mem(8) nastaje kao posledica trenutka zaustavljanja procesa simulacije. Konkretno proces simulacije je prekinut posle naredbe `add` a pre naredbe `store`. Za slučaj da je proces simulacije zaustavljen posle naredbe `store` a pre naredbe `add` do nekonzistentnosti sadržaja reg(2) i mem(8) ne dolazi.

Postupak zaustavljanja programa u proizvoljnom trenutku izvršenja odgovara režimu rada prekidne tačke (breakpoint) i koristi se za potrebe debagiranja.

Prekinutu simulaciju je moguće ponovo akitivirati opcijom *Run.* Program `counter` će nastaviti tamo gde je stao. Ponovno zaustavljanje se obavlja na isti način.
Crvena vertikalna linija je granična linija koja pokazuje na trenutak zaustavljanja procesa simulacije (slike 10 i 11).

Pozicioniranjem kursora u polje vremenskog-dijagrama (slike 10 i 11) prati se stanje internih registara procesora ili stanje memorije u zavisnosti od izbora.



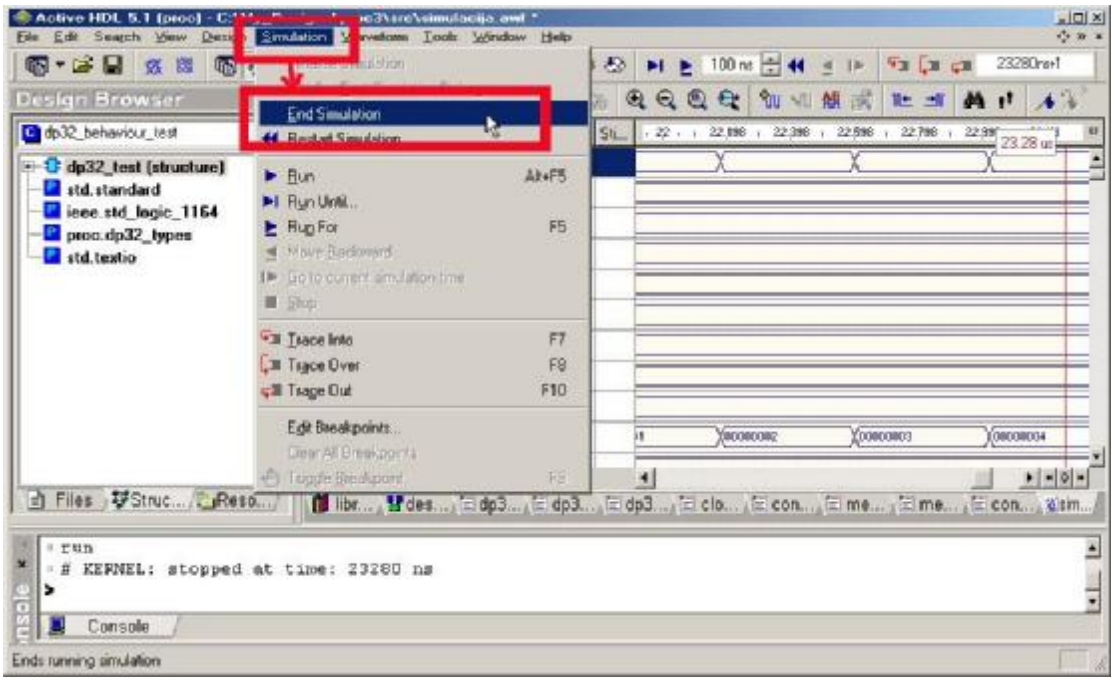
Slika 11. Stanje memorije

Korak 5

Ukoliko želimo da završimo simulaciju postupak je sledeći:

- Selekcijom *Simulation* na Menu Bar-u otvara se padajući meni (slika 12) koji sadrži opciju *End Simulation.* Klikom na polje *End Simulation* završava se tekuća simulacija.

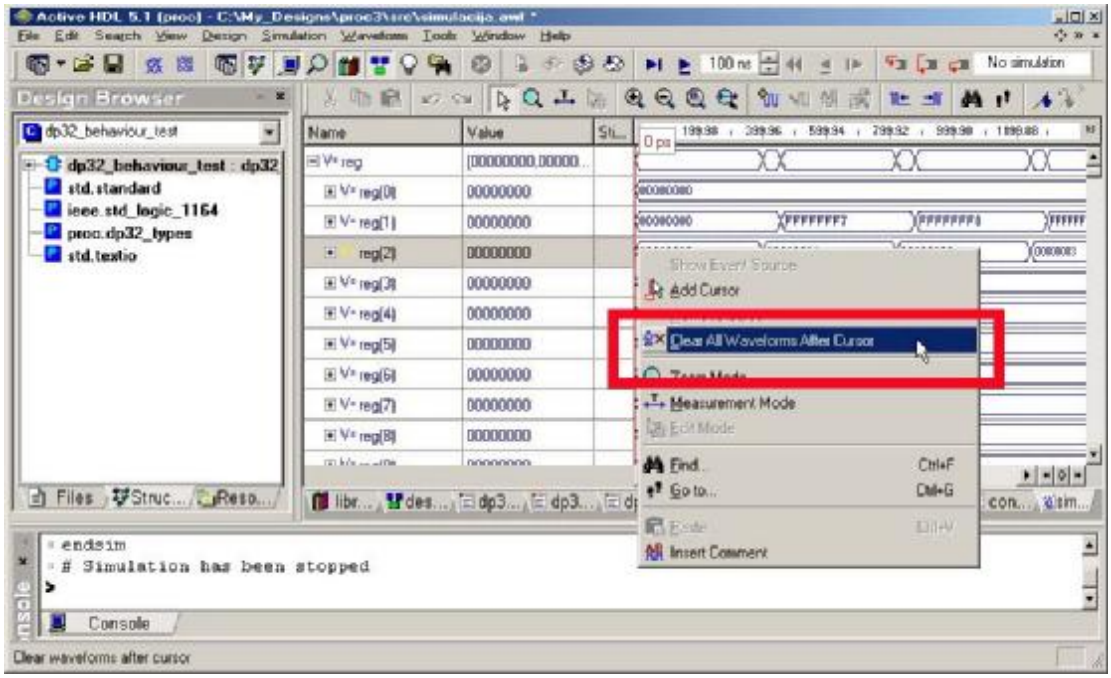
Indikacija o završetku procesa simulacije se raportira u prozoru *Console* porukom # *Simulation has been stopped*



Slika 12. Prikaz opcije *End Simulation*

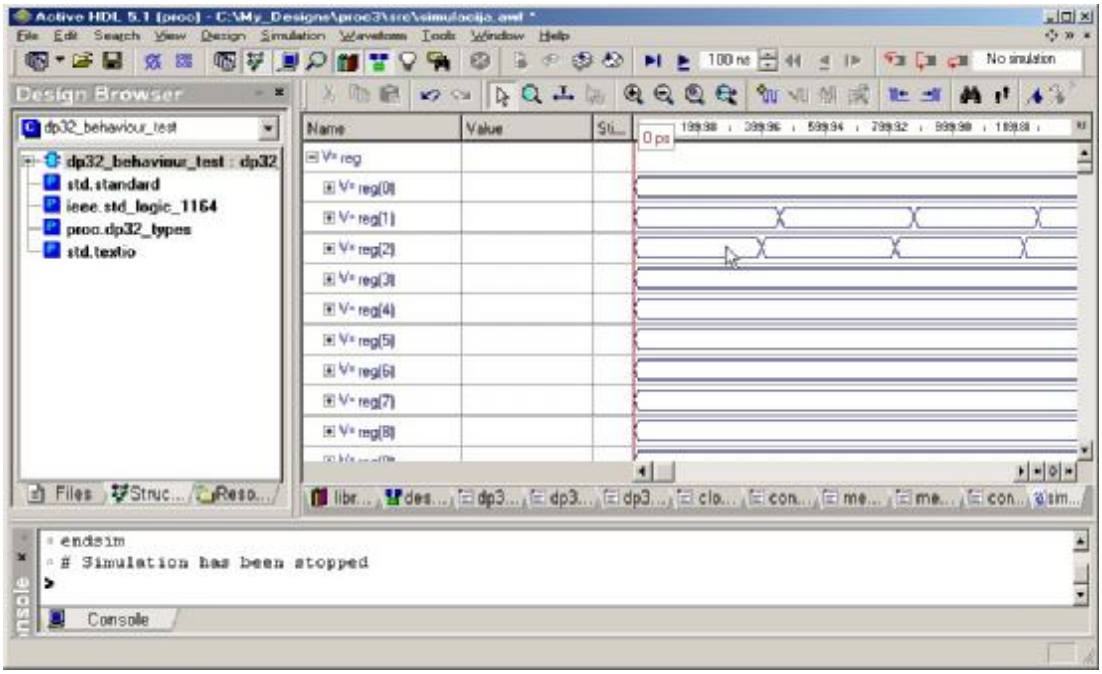
Da bi postavili sistem u inicijalno stanje neophodno je preduzeti sledeće korake:

- privući crvenu graničnu liniju na početak simulacije (0 ns)
- klikom desnim tasterom miša na crvenu graničnu liniju pojavljuje se padajući meni.
- aktiviranjem opcije *Clear All Waveforms After Cursor* (slika 13) sistem se postavlja u inicijalno stanje.



Slika 13. Način brisanja prethodnih stanja u brojaču

Na slici 14. prikazan je izgled prozora nakon postavljanja sistema u inicijalno stanje (svi interni registri su postavljeni na vrednost ‘0’, sve memorijske lokacije su izbrisane).

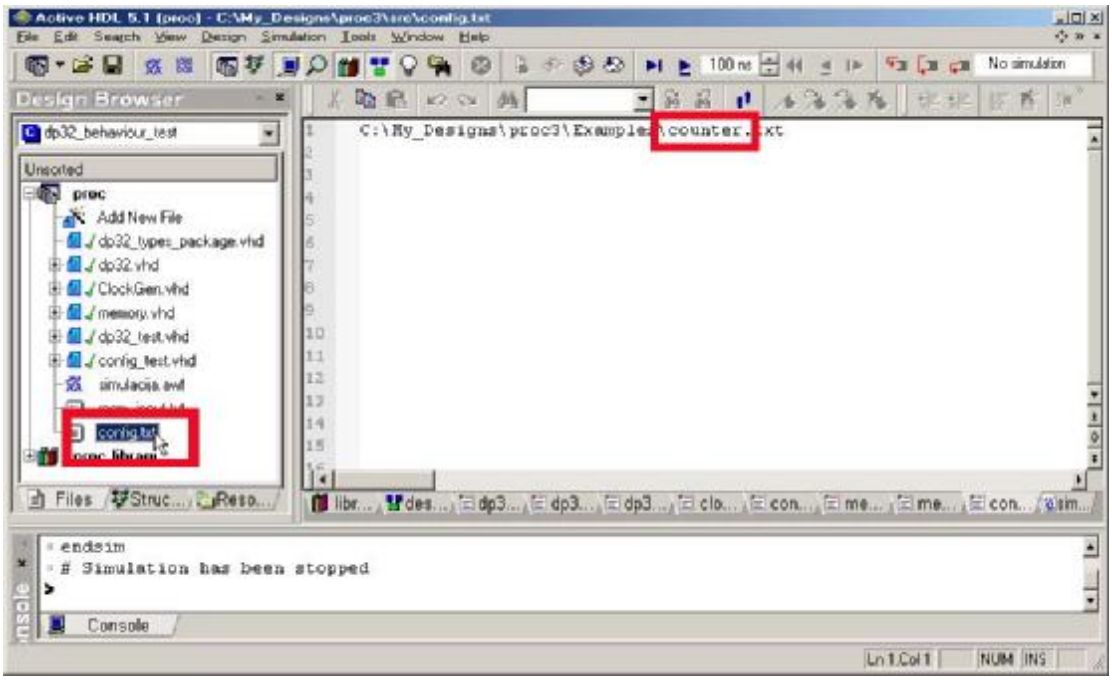


Slika 14. Izgled prozora nakon brisanja stanja brojača

Dodatak: Listanje sadržaja programske memorije

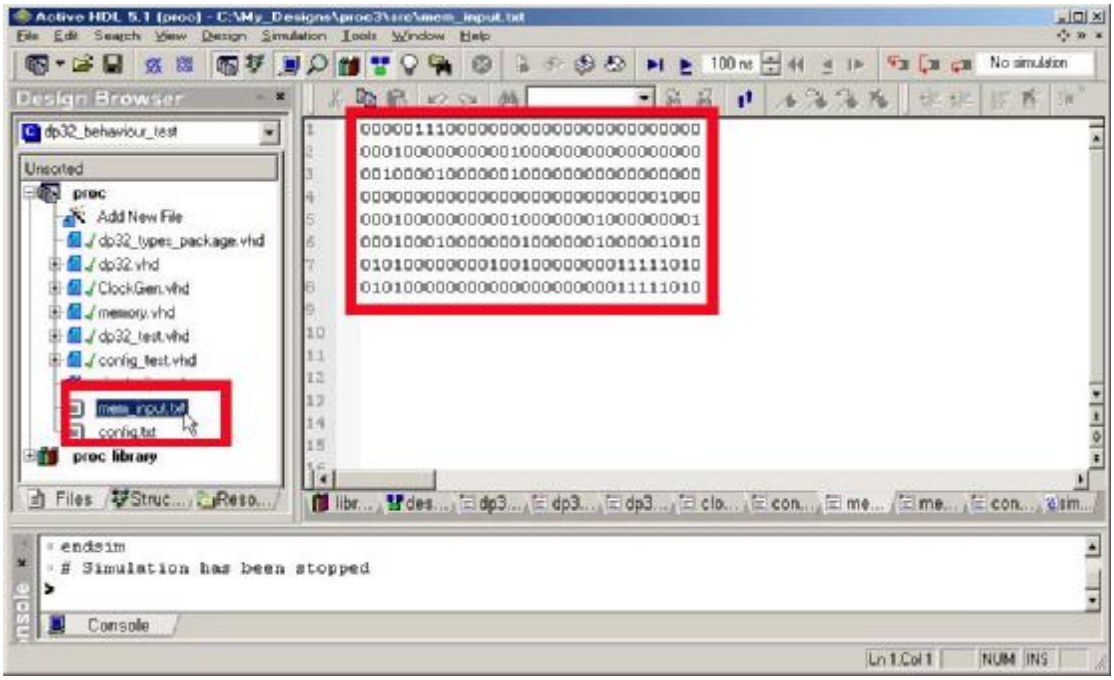
Često se javlja potreba za listanjem sadržaja programske memorije nakon što je program *load*-ovan. Listanje se obavlja na sledeći način:

1. Pre kompajliranja treba upisati zadatak odgovarajuće grupe (1-16). Svaki student treba da promeni samo naziv programa obeležen crvenom bojom, dvostrukim klikom na `config.txt` u Design Browser-u (slika 15).



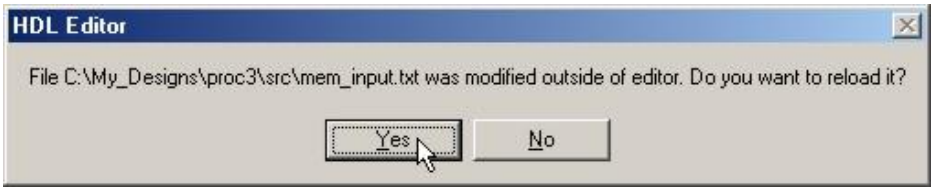
Slika 15. Način biranja zadatog programa

2. Aktiviranjem `mem_input.txt` (Slika 16.) otvara se prozor u kome je prikazan *layout* memorije u binarnom obliku (Slika16.), i prozor koji nas obaveštava da je modifikovan sadržaj memorije kao i da li želimo da prihvatimo tu promenu (Slika 17.)



Slika 16. *Layout* memorije u binarnom obliku

Napomena: U konkretnom slučaju za program `counter` layout memorije u binarnom obliku je prikazan na Slici 16.



Slika 17. Prozor: potvrda-promene sadržaja memorije

ZADATAK:

Programska sekvenca XXX.YYY odgovara sadržaju programske memorije dat u heksadecimalnom formatu. Prva kolona odgovara adresi memorijske lokacije a u drugoj koloni je dat sadržaj memorijske lokacije. Svaku memorijsku lokaciju čine 32 bita.

Za dodeljenu programsku sekvencu:

a) napisati asemblerski program u koloni 3, a u polju komentar u koloni 4 ukazati koja se aktivnost obavlja od strane svake instrukcije (vidi sliku 2.)

b) proveriti izvršenje dodeljene programske sekvence aktiviranjem simulatora procesora DP32. Postupak rada je identičan kao u Sekciji 2. PRIMER SIMULACIJE i čine ga sledeće aktivnosti:

1. **pokretanje programa** *Active HDL*: aktivirati *My Computer* ® na C: particiji aktivirati *My _Designs* ® otvoriti folder *proc* i aktivirati ikonu *proc.adf (Active HDL.Design)*. (za više detalja videti Korak 1 u Sekciji 2. PRIMER SIMULACIJE)
2. **izbor programske sekvence XXX.YYY koju treba simulirati**: dvostrukim klikom na `config.txt` u Design Browser-u otvoriti prozor u kome se menja naziv programa ® aktivirati `mem_input.txt` radi prikaza *layout*-a memorije ®potvrditi promenu sadržaja memorije (za više detalja videti Dodatak u Koraku 5 u Sekciji 2. PRIMER SIMULACIJE)
3. **kompajliranje i inicijalizacija simulacije:** kompajliranje pokrenuti klikom na ikonicu *Compile All* ® selektovati *Simulation* na Menu Bar-u i aktivirati komandu *Initialize Simulation* (za više detalja videti Korak 2 u Sekciji 2. PRIMER SIMULACIJE)
4. **pokretanje simulacije:** selektovati *Simulation* na Menu Bar-u i aktivirati komandu *Run*. (za više detalja videti Korak 3 u Sekciji 2. PRIMER SIMULACIJE)
5. **zaustavljanje simulacije:** selektovati *Simulation* na Menu Bar-u i aktivirati komandu *Stop*® izlistati sadržaj registra i/ili memorije (za više detalja videti Korak 4 u Sekciji 2. PRIMER SIMULACIJE)
6. **završetak simulacije:** selektovati *Simulation* na Menu Bar-u i aktivirati komandu *End Simulation* ® postaviti sistem u inicijalno stanje aktiviranjem opcije *Clear All Waveforms After Cursor* (za više detalja videti Korak 5 u Sekciji 2. PRIMER SIMULACIJE)

c) Napisati proizvoljnu programsku sekvencu na sledeći način:

1. aktivirati *Start* / *Programs* / *Accessories* / *Notepad* ® upisati programsku sekvencu u heksadecimalnom formatu maksimalne dužine do 8 linija® selektovati *File* na Menu Bar-u, aktivirati komandu *Save As* i snimiti fajl pod željenim imenom na lokaciji C:\My_Design\lab_vezbe
2. proveriti izvršenje programske sekvence aktiviranjem simulatora procesora DP32. Postupak rada je identičan kao pod stavkom b) ove sekcije.

STUDENT 1:

sekvenca: ADD.TXT

0 07010000
1 07020000
2 07030000
4 10020002
5 10010001
6 00030102

STUDENT 2:

sekvenca: MULTIPLY.TXT
0 07010000
1 07020000
2 07030000
2 10010002
3 1002010A
4 02030201

STUDENT 3:

sekvenca: DIVIDE.TXT
0 07010000
1 07020000
2 07030000
3 10010006
4 10020024
5 03030201

STUDENT 4:

sekvenca: SUBTRACT.TXT
0 07010000
1 07020000
3 07030000
3 10010001
4 1002000A
5 01030201

STUDENT 5:

sekvenca: ADDQ.TXT
0 07010000
1 10010101

STUDENT 6:

sekvenca: SUBQ.TXT

0 07010000
1 1101010A

STUDENT 7:

sekvenca: MULQ.TXT
0 07010000
1 1001010A
2 1201010A

STUDENT 8:

sekvenca: DIVQ.TXT

0 07010000
1 1001010A
2 13010105

STUDENT 9:

sekvenca: LD.TXT

0 07020000
1 20020000
2 00000009
9 AAAAAAAA

STUDENT 10:

sekvenca: LDQ.TXT

0 07030000
1 30030009
9 AAAAAAAA

STUDENT 11:

sekvenca: ST.TXT

0 07020000
1 1002026A
2 21020000
3 00000008

STUDENT 12:

sekvenca: STQ.TXT

0 07020000
1 1002026A
2 31020008

STUDENT 13:

sekvenca: BR.TXT

0 07010000
1 07020000
2 1001010A
3 01020201
4 400A0000
5 00000001
7 01010101
8 40090000
9 FFFFFFF8

STUDENT 14:

sekvenca: BRQ.TXT

0 07010000
1 07020000
2 1001010A
3 01020201
4 500A0002
7 01010101
8 500900F9

STUDENT 15:

sekvenca: BI.TXT

0 07010000
1 07020000
2 07030000
3 1001010A
4 01020201
5 410A0300
6 00000008
8 01010101
9 41090300
10 00000000

STUDENT 16:

sekvenca: BIQ.TXT

0 07010000
1 07020000
2 07030000
3 1001010A
4 01020201
5 510A0308
8 01010101
9 51090300