ELEKTROTEHNIČKI FAKULTET
BEOGRAD

Implementacija 32-bitnog procesora sa protočnom obradom

student: Stefan Vranković
512/2010
Dejan Golubović
321/2012

profesor: dr Veljko Milutinović
asistent: Živojin Šuštran

broj osvojenih poena:

Beograd
10.06.2016

Implementacija 32-bitnog procesora sa protočnom obradom

Stefan Vranković, Dejan Golubović
email: vs100512d@student.etf.rs, gd120321d@student.etf.rs

1. DEFINISANJE PROJEKTA

1.1 Uvod

Kroz istoriju računarstva razvoj procesora je bio jedna od najvažnijih grana. Počev od prvih, jednostavnijih modela, preko kompleksnijih, sve do današnjih superračunara, sa daljom tendencijom ka usavršavanju i unapređenju, procesori postaju neizostavni deo kako računara, tako i sve većeg broja uređaja za svakodnevnu upotrebu. Procesori su sve brži, kompleksniji, ali njihova osnova je slična. Sa ciljem da bismo napredniji koncepte, počinjemo od jednostavnijih modela.

1.2 Ciljevi projekta

Cilj projekta je implementacija jednog 32-bitnog procesora kako bi se studenti upoznali sa osnovama dizajniranja procesora i radom sa jezicima za opisivanje hardvera.

2. OPIS DIZAJNA

2.1 Zabeleške uz dizajn

Potrebno je projektovati 32-bitni procesor opšte namene. Procesor je povezan sa dve keš memorije, jednom za podatke, drugom za instrukcije.

Interfejs procesora prema okolini sadrži linije za komunikaciju sa keš memorijama, RESET signal i signal kloka. Prilikom startovanja sistema RESET signal se postavlja na aktivnu vrednost i nakon nekog vremena se vraća na neaktivnu vrednost. Smatrati da je trajanje RESET signala dovoljno da se ceo sistem resetuje. Adresibilna jedinica je reč. Reč je veličine 4 bajta.

Jezgro je RISC arhitektura sa sledećim osobinama:

- registarski fajl
- load/store arhitektura
- adresiranje je bazirano na vrednostima u registrima
- uniformna instrukcijska reč, veličine 32 bita

Procesor sadrži protočnu obradu. Broj stepeni je 5. Sve hazarde je potrebno hardverski razrešiti uz poštavanje sledećih zahteva: 1) za instrukcije skoka se koristi prediktor skoka i 2) zaustavljati protočnu obradu samo kad prosleđivanje nije moguće i kada postoji zavistnost po podacima. Pored traženih zahteva težiti ka što boljim performansama. Prediktor skoka treba da bude realizovan po principu keša sa dvobitnom šemom.

Jezgro ima 32 registara opšte namene širine 32-bita. Registri su obeleženi sa R0 – R31. Registri specijalne namene su: 1) PC je pokazivač na sledeću instrukciju i 2) SP pokazivač na vrh steka. SP pokazuje na prvu slobodnu lokaciju i stek raste ka nižim adresama.

2.2 Faze dizajna

Prvo se posmatra funkcionalni nivo dizajna. Tu se posmatra šta procesor treba da radi ali nam se ne govori ništa o implementaciji. Tretira naš proceosor kao crnu kutiju koja bi trebalo da reaguje na svaku kombinaciju ulaznih signala ali nam ne govori ništa o načinu na koji bismo to dizajnirali.

Drugi deo dizajna je strukturalni. Tu počinjemo da dobijamo odgovore na neke od pitanja iz funkcionaln og dela. Predstavljamo povezivanje komponenti i način na koji one funkcionišu zajedno. Transfer memorija-procesor i registar-registar. Veze između faza instrukcija.

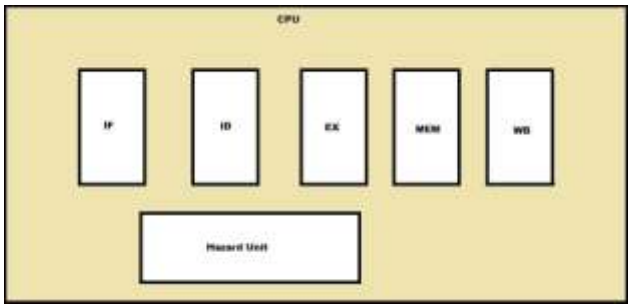
Fizičku reprezentaciju obavlja kompajler.

2.2.1 Interfejs između procesora i memorije

Linije za komunikaciju sa keš memorijom podataka su: 1) 32 adresne linije, 2) 32 linije za čitanje podatka, 3) 32 linije za upis podataka i 4) kontrolne linije. Linije za komunikaciju sa instrukcijskom keš memorijom su: 1) 32 adresne linije, 2) 32 linije za čitanje podataka i 3) kontrolne linije.

2.2.2 Faze izvršavanja instrukcije

Faze izvršavanja instrukcije su Instruction Fetch, Instruction Decode, Instruction Execution, Memory Access i Write Back.



Slika 1 - Faze izvršavanja instrukcije

2.2.2.1. Instruction Fetch

U ovoj fazi se računa naredna vrednost PC-a i šalje se adresa instrukcijskoj memoriji sa koje se čita instrukcija. Fazi ID se šalje vrednost PC, PC+1, i prediktovana vrednost PC-a za instrukcije skoka.

U ovoj fazi se nalazi prediktor skoka.

2.2.2.2. Instruction Decode

U ovoj fazi se dekoduje instrukcija dobijena od instrukcijske memorije. Generišu se kontrolni signali i šalju se narednim fazama. Računa se i koji su registri potrebni za računanje te instrukcije i šalju se hazard jedinici kako bi se izračunalo da li je došlo do prosleđivanja ili treba eventualno generisati stall.

Tu se nalazi i registarski fajl pa ova faza njemu šalje signale za upis i čitanje. Registarskom fajlu se pristupa asinhrono unutar ove faze. Registarskom fajlu se šalju kontrolni signali rd, wr, adresa za čitanje/upis i eventualno podatak za upis.

Pravi se record u koji se smeštaju svi kontrolni signali i podaci od interesa o jednoj instrukciji. Taj record se prenosi kroz ceo pajplajn.

2.2.2.3. Instruction Execution

U ovoj fazi se obavljaju sva izračunavanja. Za aritmetičke i pomeračke instrukcije računa se rezultat operacije i to se prosleđuje mem fazi, dok za load i store računamo adresu sa koje se čita i to upisujemo u zapis o instrukciji. Za instrukcije skoka računamo da li je do skoka došlo ili ne, osim za rts.

Šaljemo hazard jedinici koji registar nam je ovde aktivan, u koji treba da upišemo i vrednost koju upisujemo. Hazard jedinici šaljemo tačno:

- Adresu registra u koji se upisuje u datoj instrukciji
- Da li stvarno upisujemo u taj registar
- Podatak koji se upisuje
- Da li je podatak validan - ako je aritmetička instrukcija - jeste, ako je load - nije.
- Da li je instrukcija flush-ovana.

Ovo je format koje hazard jedinici šalje i mem i wb faza.

2.2.2.4. Memory Access

U ovoj fazi pristupamo memoriji i steku.

Ako je u pitanju memorijska instrukcija, tada pristupamo memoriji. Šaljemo signale rd, wr, adresa za čitanje i eventualno podatak za upis. Instrukcije load i store.

Ako je pitanju instrukcija koja radi sa stekom, hardverskom steku šaljemo kontrolne signale, push, pop, i eventualno podatak za push. To radimo kod instrukcija push, pop, jsr, rts.

Ako nisu ni memorijske ni stek instrukcije, samo šaljemo podatke o instrukciji (record) narednoj fazi.

Ovu fazu stopiramo, ne izvršavamo, ako je došlo do stall ili flush signala. Tako sprečavamo upis u memorijske lokacije ili stek ako je instrukcija nevalidna, iako Exec faza obavi izračunavanja.

Ova faza šalje isti format podataka hazard jedinici, kao i exec.

2.2.2.5. Write back

U ovoj fazi obavljamo slanje podataka ID fazi za upis u registre. Tu imamo podatak i u slučaju aritmetičkih operacija, i u slučaju instrukcija sa memorijom i stekom. Šaljemo kontrolne signale wr, adr i data za upis u registre. ID faza ih onda prosleđuje registarskom fajlu.

3. IMPLEMENTACIJA

Implementacija je rađena u jeziku VHDL. Osnovna ideja je bila podeliti procesor na faze koje bi predstavljale entitete. Zatim specifične delove faza podeliti ponovo na entitete. Za svaki entitet arhitektura i procesi. Ukupan broj entiteta - 12. Entiteti:

- InstrFetch - Faza čitanja instrukcije
- InstrDecode - Faza dekodovanja instrukcije
- InstrExec - Faza izračunavanja
- InstrMem - Faza pristupa memoriji
- InstrWB - Faza upisa u registre
- RegisterFile - Registarski fajl u ID fazi
- Predictor - Prediktor skoka u IF
- DataMemory - Memorija sa podacima
- InstructionMemory - Instrukcijska memorija
- Stack - Hardverski stek
- HazardUnit - Jedinica za hazarde
- CPU - Procesor

3.1 Hazardi

Hazard koji se ovde javlja je read after write. Rešen je prosleđivanjem, a ako ne može prosleđivanje - onda stall.

3.2 Oporavak od hazarda

Prosleđivanje imamo kada nam aritmetičko-logička ili pomeračka instrukcija generiše rezultat koji koristi naredna instrukcija. Tu podatak imamo spreman u EX fazi pa može da se prosledi u ID naredne instrukcije. Takođe isti podatak se prosleđuje i iz MEM i WB faze.

Instrukcije LOAD i POP generišu stall. Ako npr. instrukcija posle instrukcije LOAD koristi kao argument rezultat LOAD instrukcije, onda nam hazard jedinica generiše stall koji stopira procesor na dva takta dok podatak ne bude spreman, dok se ne pročita iz memorije.

3.3 Primeri hazarda

Primer 1: Instrukcija SUB koristi rezultat instrukcije ADD, pa se rezultat prosleđuje.

ADD R1 R2 R3	IF	ID	EX	M	WB	
SUB R4 R1 R5		IF	ID	EX	M	WB

Primer 2: Instrukcija SUB koristi rezultat instrukcije LOAD, pa procesor mora da se zaustavi dok podatak ne bude spreman.

LOAD R1 R2 #1	IF	ID	EX	M	WB			
SUB R4 R1 R5		IF	ID	S	S	EX	M	WB

3.4 Prediktor skoka

Prediktor je realizovan kao keš jedinica za prediktovanje.

U tag delu je adresa instrukcije skoka.

U data delu je adresa skoka.

State deo je automat sa 4 stanja, da li dolazi do skoka ili ne. 00 - Not Taken, 01 - Weekly not Taken, 10 - Weekly Taken, 11 - Taken.

LRU - algoritam zamene. Svaki ulaz u kešu ima svoj brojač od 0 do 2^n - 1. U svakom trenutku svi ovi brojači za sve ulaze moraju da budu različiti. Ako imamo 4 ulaza u kešu, vrednosti su 3, 2, 1, 0. Kada koristimo neki ulaz iz keša, postavljamo njegov brojač na 0. One brojače koji su bili veći od njega ne diramo, manje povećavamo. Kada ubaciujemo u keš, ubacujemo na mesto najvećeg brojača.

TAG	DATA	STATE	LRU
0001h	0100h	11b	11
0005h	0105h	10b	10
0009h	0109h	10b	00
0011h	0102h	11b	01

Opis iz tabele, prvi red. Na adresi 0001h imamo skok na adresu 0100h. State je 11 znači skok treba da se desi. Predviđamo da skačemo na 0100h.

Ako u ovaj prediktor dođe na primer adresa:
tag:0100 data:0103 state:10,
tada neku treba izbaciti iz tabele i udpatovati red. Izbacićemo prvi red, jer je LRU=11, tj maksimum.

4. TESTIRANJE I VERIFIKACIJA

Testirano je sa pojedninačnim instrukcijama, random instrukcijama, i sa programima koji proizvode rezultat. Posebno testirane specifične situacije gde se generišu stall i flush signali, gde je predikcija uspešna/neuspešna, gde dolazi do prosleđivanja, ili neke neregularne situacije.

3.1. Javni test

```
PC = 00001000
SUB R6, R6, R6
SUB R4, R4, R4
LOAD R6,R6
LOAD R5,R6
SUB R5,R5,#1
BGT R5, R4, -2
STORE R5,R6
HALT
```

Specifično:
Prosleđivanje SUB do LOAD, R6
STALL između LOAD i LOAD, R6
STALL između LOAD i SUB
Prosleđivanje SUB do BGT, R5
STORE upis u memoriju, upisano 0.

3.2. Rekurzivno sabiranje

```
1000    MOVI R1 #10 BROJ DO KOG SE SABIRA
1001    MOVI R2 #1
1002    MOVI R3 #0 REZULTAT
1003    JSR 100A
1004    JMP 1013

100A    BNE R1 R2 #2
100B    MOVI R3 #1
100C    RTS
100D    PUSH R1
100E    SUBI R1 R1 #1
100F    JSR 100A
1010    POP R1
1011    ADD R3 R3 R1
1012    RTS
1013    HALT
```

Specifično: Dosta skokova, jsr, rts, jmp, rekurzija.

3.3. Aritmetička sredina niza

```
1000
movi r1 #0x08    // r1=8  broj brojeva koji se sabiraju
movi r2 #0x10    // r2=16 adresa u data memoriji
xor r3 r3 r3     // r3=0  cuvanje trenutnog broja
xor r4 r4 r4     // r4=0  konacan rezultat
movi r5 #1       // r5=1 brojac
jsr 100A
halt

100A
add  r2, r2, r5   // uvecavamo adresu za brojac
load r3, r2, #0   // r3=mem[r2] citamo podatak u r3
add  r4 r4 r3          // r4=r4+r3 inc sume
addi r5 r5 #1     // inc brojaca
ble  r5 r1 -5     // ako je r5>r1 onda smo stigli do kraja, ne vracamo se
shr r4, #3        // pomeranje r4 za 3 mesta udesno
rts
```

Specifično: Dosta skokova i load operacija, push, pop.

5. ZAKLJUČAK

U ovom projektu smo imali prilike da se susretnemo sa mnogim izazovima prilikom dizajna i implementacije procesora. Pajplajn, hazardi, predikcija skoka su bili ključni elementi za implementaciju. Ovim projektom studenti su stekli kompletan uvid u dizajn svih faza pajplajn procesora i mogli da primete koliko pajplajn organizacija utiče na brzinu i performanse procesora. Kako je ipak ovo uvod, postoji delovi projekta koji bi mogli da se optimizuju. Pre svega prediktor skoka, gde se može unaprediti procenat pogođenih skokova. Ideja za dalji rad ima puno, ali ovim projektom je uspešno završen uvod u dizajn i implementaciju procesora.

6. LITERATURA

Spisak literature korišćen u izradi domaćeg zadatka:
[1] http://home.etf.rs/~vm/os/vlsi/index.html