Savremeni mikroprocesori (2)

Literatura: Hennessy & Patterson, C.1–C.2

Model rada (klasična fon Nojmanova arhitektura)

- Memorija je linearna i sekvencijalno adresirana (veličina najmanje adresibilne jedinice nije bitna, danas je to 8-bitni bajt/oktet)
- Instrukcije se izvršavaju strogo sekvencijalno
  - izvršavanje naredne instrukcije počinje tek kad je prethodna završena
  - naredna instrukcija vidi sve efekte prethodne
- Integrisana kola su sinhrona
  - ovo nema neposredne veze sa načinom izvršavanja instrukcija, ali ima presudan uticaj na arhitekturu

Instrukcijski ciklus (1)

- Koraci potrebni za izvršavanje jedne instrukcije, apstraktno gledano; realni procesori imaju *mašinski* ciklus
- (1) Prihvatanje instrukcije
  - iz memorijske lokacije na koju pokazuje **programski brojač**, registar koji prati izvršavanje instrukcija
  - po završetku ovog koraka programski brojač će pokazivati na sledeću instrukciju u nizu
- (2) Dekodiranje instrukcije

Instrukcijski ciklus (2)

- (3) Čitanje efektivne adrese (može se posmatrati i kao potkorak drugog koraka)
  - ako instrukcija koristi indirektno adresiranje, efektivna adresa se čita iz memorije
  - ako je instrukcija memorjska i direktna, u ovom koraku se ništa ne dešava
  - isto važi i za instrukcije koje ne operišu s memorijom
- (4) Izvršavanje
  - ukoliko je u pitanju izvršena instrukcija grananja, postaviće novu vrednost u programski brojač

Petostepeni transport

- „5-stage pipeline“, jedan od načina da se instrukcijski ciklus mapira na hardver
- Istorijski, prvobitno upotrebljen (za mikroprocesore) u RISC arhitekturama 1980-ih

IF	Instruction fetch	Prihvatanje instrukcije
ID	Instruction decode	Dekodiranje instruckije
EX	Execute	Izvršavanje
MEM	Memory access	Pristup memoriji
WB	Register writeback	Zapis registara

Primarni zahtevi „pipelined“ ahitekture

- Instrukcije su uniformne dužine, najčešće 32 bita
- Koriste se proste instrukcije za memorijski pristup
  - „Load/store architecture“, bez komplikovanih režima adresiranja
- Sve ostale instrukcije rade s registrima
- Široko korišćeno u RISC arhitekturama, kanonički primer: MIPS, moderna iteracija: RISC-V

Problemi u „pipelined“ arhitekturi

- Dve osnovne kategorije problema
- Prva, veće zauzeće memorije za instrukcije nego što je neophodno
  - ovo znači i indirektan pritisak na interne procesorske memorijske strukture, što ćemo obraditi kasnije
- Druga, narušavanje sekvencijalne semantike izvršavanja:
  - prilikom rada s podacima
  - prilikom grananja

Zauzeće memorije za instrukcije

- Uobičajen izbor je bio veličina mašinske reči, 32 bita
- Ovo je dovoljno da se nedvosmisleno kodiraju troadresne instrukcije sa skupom od 32 registra
- Značajan podskup programa ne zahteva ovako veliki registarski prostor
- Uvode se kompaktne, 16-bitne instrukcije koje rade na smanjenom skupu registara i ograničenom skupu instrukcija
- Kanonički primer: ARM Thumb

Varijante kompaktne reprezentacije

- Prva varijanta (Thumb u prvobitnoj realizaciji) zahteva da se procesor prebaci u poseban režim rada da bi izvršavao ovakve instrukcije
- Ovo je problematično jer se često dešava da je u nizu kompaktnih instrukcija potrebno izvršiti nekoliko instrukcija standardne veličine, pa treba preskakati između režima rada
- Modernija izvedba (RISC-V, ARM Thumb2) omogućava da se standardne i kompaktne instrukcije slobodno mešaju
- Rezultat (RISC-V): gustina koda vrlo slična onoj koju ima Intel x86_64 arhitektura

Semantika pristupa podacima

- Uopšteno: problem nastaje ako paralelno izvršavanje delova instrukcija ili čitavih instrukcija izazove efekat nemoguć u sekvencijalnom kodu
- Uvek je moguće sprečiti semantičke probleme ako se u izvršavanje unese čekanje, čija je krajnja granica ekvivalent sekvencijalnog izvršavanja
- Ovo obara performanse, pa se, ako je moguće, traže rešenja koja će raditi bez čekanja

Klasifikacija opasnosti (1)

- Nepoželjni efekti se zovu *opasnosti* (*hazards*) i označavaju se skraćenicama
- Oznake mogu da deluju zbunjujuće, jer ukazuju na *poželjan* efekat
- Svaka od opasnosti uključuje zapisivanje vrednosti u nekoj od kombinacija; samo čitanje nikad ne može narušiti semantiku
- Kad budemo opisivali opasnosti, navešćemo prvo poželjan, pa nepoželjan efekat, koji treba sprečiti

Klasifikacija opasnosti (2)

- RAW (*Read After Write*):
  - želimo da čitanje posle pisanja vrati upravo zapisanu vrednost
  - opasnost: naredna instrukcija ne vidi rezultat izvršavanje prethodne
- WAR (*Write After Read*):
  - želimo da čitanje pre pisanja vrati originalnu vrednost
  - opasnost: prethodna instrukcija vidi rezultat izvršavanja naredne
  - nemoguće u klasičnoj petostepenoj arhitekturi, može da se desi ako se instrukcije izvršavaju preko reda

Klasifikacija opasnosti (3)

- WAW (*Write After Write*):
  - želimo da konačan rezultat ove sekvence bude druga zapisana vrednost
  - opasnost: ukupan efekat je rezultat izvršavanja prethodne instrukcije, a ne naredne
  - takođe nemoguće u standardnoj petostepenoj arhitekturi
- Ilustrovaćemo samo RAW opasnost jer je najčešća, relativno lako shvatljiva, i često ima tehničko rešenje

RAW problem (rešiv bez čekanja)

ADDU **R2**,R1,R3
SUBU R4,**R2**,R5

ADDU R2,R1,R3	IF	ID	EX	MEM	**WB**	
SUBU R4,R2,R5		IF	ID	EX	MEM	WB

- Instrukcija ADDU izračunava rezultat u stepenu EX, ali ga zapisuje u R2 tek u stepenu WB
- Instrukcija SUBU čita registar R2 u stepenu ID, tako da vidi raniju vrednost
- Rešenje: hardverska indikacija promene tako da rezultat postaje vidljiv već u stepenu ID

RAW problem (nerešiv bez čekanja)

LW **R1**,0(R2)
SUBU R4,**R1**,R5

LW R1,0(R2)	IF	ID	EX	MEM	WB	
SUBU R4,R1,R5		IF	ID	EX	MEM	WB

- Instrukcija LW (Load Word) ima vrednost na raspolaganju tek u stepenu MEM, kada se instrukcija SUBU već izvršava
- Mora se čekati bar jedan mašinski takt da bi učitana vrednost mogla da se pročita u stepenu ID od strane instrukcije SUBU

Grananje (1)

- Dva osnovna problema s grananjem
- U principu se ne može pouzdano znati koja će se instrukcija semantički izvršiti posle instrukcije grananja
- Kod petostepene arhitekture, u trenutku dekodiranja instrukcije grananja naredna instrukcija (u memoriji) već je pročitana, tako da je trošak njenog čitanja i dekodiranja straćen ukoliko se grananje izvrši

Grananje (2)

- Za prvi problem, bez dodatnih komplikacija nema rešenja sem čekanja
- Za drugi, neki RISC procesori čine vidljivom (i izvršavaju) instrukciju iza grananja bez obzira da li je do grananja došlo: „branch delay slot“
- Ovo je zbunjujuće za programere i nezgodno za kompajlere, tako da su kasniji procesori ovo izbegavali

Dodatno unapređenje paralelizma (1)

- Prekoredno izvršavanje (*Out of order execution*)
  - izvršava instrukcije u zavisnosti od dostupnih podataka
- Superskalarni procesori
  - izvršavaju više od jedne instrukcije istovremeno
  - koriste postojanje više izvršnih jedinica u okviru procesora
  - (ovo još uvek nije višeprocesorski sistem)

Dodatno unapređenje paralelizma (2)

- Spekulativno izvršavanje (*Speculative execution*)
  - izvršavanje unapred da bi se amortizovali troškovi grananja
- Predviđanje grananja (*Branch prediction*), sklop se zove *prediktor*
  - prosti statički (uslovno grananje se neće izvršiti)
  - statički (→ se neće izvršiti, ← će se izvršiti)
  - primena spekulativnog izvršavanja