UNIVERZITET
METROPOLITAN
BEOGRAD

CS120 - ORGANIZACIJA RAČUNARA

Organizacija računara

Lekcija 02

PRIRUČNIK ZA STUDENTE

CS120 - ORGANIZACIJA RAČUNARA

Lekcija 02

ORGANIZACIJA RAČUNARA

UVOD

Uvod u organizaciju računara

U ovoj lekciji najpre biće reči o centralnom procesoru (en. Central Processing Unit, *CPU*), koji se smatra mozgom računara.

Biće reči o tome šta us gradivni blokovi CPU-a, kako CPU zapravo izvršava instrukcije, i kakva je interna organizacija pojedinačnih delova koji čine CPU.

Zatim, biće detaljnije obrađene same instrukcije proceosora, kakve vrsta intrukcija postoje, i koji je tok izvršenja instrukcija. Dat je osvrt na kompleksnost instrukcija, i na razvoj *CISC* i *RISC* arhitekture procesora.

O formatima instrukcija u x86 arhitekture biće reči u posebnoj lekciji.

Osvim osnovnih principa projektovanja procesora koji važe od 60tih godina prošlog veka, u trećem objektu učenja fokus je stavljen na projektovanje savremenih procesora, uz video objašnjenja savremenih arhitektura (x64/ARM/RISC-V).

Četvrti objekat učenja bavi se paralelnim radom, prvenstveno paralelizmom na nivou instrukcija uz osnovni primer protočnog izvršenja instrukcija. Ostali vidovi instrukcijskog paralelizma prevazilaze opseg predmeta, dok različiti tipovi procesorskog paralelizma dobijaju posebnu lekciju.

Konačno, dati su uvodni pojmovi o operativnoj memoriji, razlike između bitova, bajtova, memorijskih reči i memorijskih ćelija.

Spomenuta je keš memorija kao način ubrzavanja rada operativne memorije, kao i problemi koji se prevazilaze korišćenjem arhitekture System-on-a-Chip.

Na pokaznim vežbama obrađuje se predstavljanje negativnih brojeva u računaru koristeći označene amplitude, prvog i drugog komplementa, predstavljanje sa pomerajem i konačno IEEE 754 standard za predstavljanje brojeva sa pokretnom tačkom (zapetom).

Poglavlje 1

Procesori

CENTRALNA PROCESORSKA JEDINICA - CPU

Centrali procesor se tretira kao mozak računara.

Organizacija jednostavnog računara sa magistralama prikazana je na slici desno. Centralna procesorska jedinica (en. Central Processing Unit, *CPU*) je „mozak“ računara.

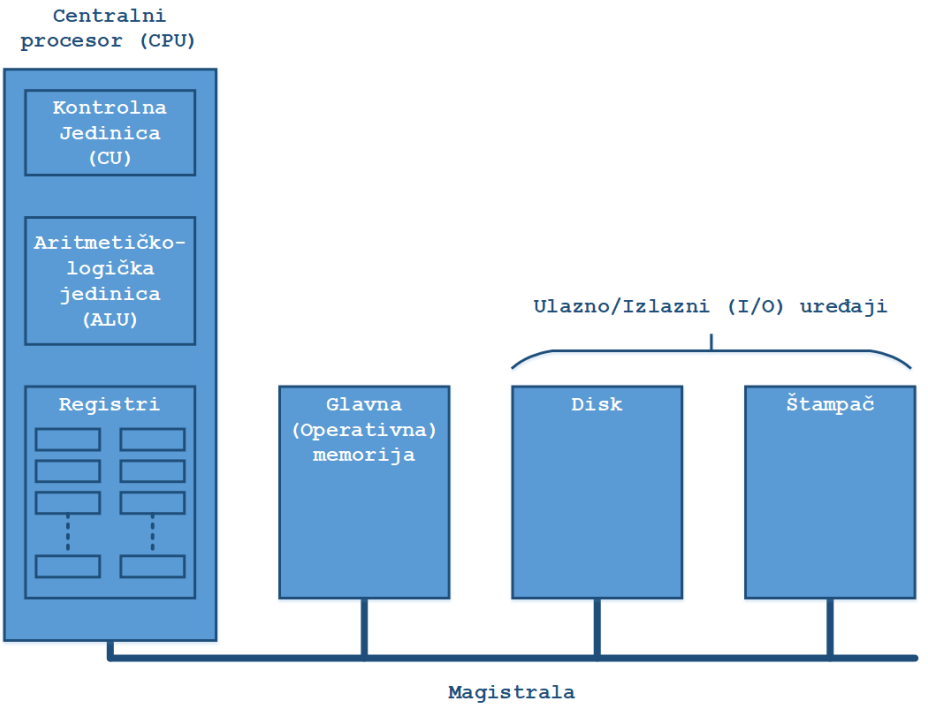
Njegova funkcija je da izvršava programe uskladištene u operativnoj memoriji tako što preuzima njihove instrukcije, ispituje ih tj. dekoduje, a zatim ih izvršava jednu za drugom.

Komponente su povezane *magistralom* ili *sabirnicom* (en. bus), koja je zapravo skup paralelnih provodnika.

Magistrale mogu prenositi adrese, podatke i kontrolne signale.

Magistrale mogu biti eksterne u odnosu na CPU, tj. mogu povezivati CPU sa povezujući sa memorijom i I/O uređajima, ali i takođe može postojati i interna magistrala tj. magistrala unutar CPU-a.

Savremeni računari imaju više magistrala.



Slika 1.1 Orzanizacija jednostnog računara sa jednim CPU-om i dva I/O uređaja. [Izvor: Autor.]

DELOVI CENTRALNOG PROCESORA

Gradivni blokovi savremenih centralnih procesora čine kontrolna jedinica (CU), aritmetičko-logička jedinica (ALU) i registri.

CPU se sastoji od nekoliko različitih delova.Kontrolna jedinica (en. Control Unit, *CU*) je odgovorna za preuzimanje instrukcija iz glavne memorije i određivanje tipa instrukcije.

Ovaj proces naziva se dekodovanje instrukcija.

Aritmetičko-logička jedinica (en. Arithmetical Logical Unit, *ALU*) obavlja operacije kao što su sabiranje, pomeranje, kao i logičke operacije kao što su AND ili OR.

Ovim jednostavnim operacijama se zapravo izvršavaju instrukcije.

CPU takođe sadrži male memorijske jedinice velike brzine koji se koriste za privremeno skladištenje rezultata, ili za skladištenje određenih kontrolnih informacija.

Ove brze memorije procesora čine niz *registara* (en. registers), od kojih svaki ima određenu veličinu tj. kapacitet, ali i funkciju. Obično svi registri imaju isti kapacitete skladištenja. Svaki registar može da sadrži npr. jedan broj, od nule do određenog maksimuma, a to je određeno kapacitetom registra.

Budući da se registri nalaze na integrisanom kolu procesora, oni mogu čitati i pisati velikom brzinom.

Najvažniji registar je Programski brojač (en. program counter, *PC*), koji ukazuje na sledeću instrukciju koja se preuzima za izvršenje.

Zanimljivost:
Naziv „programski brojač“ je donekle obmanjujuć jer zapravo nema veze sa bilo kakvim prebrojavanjem, ali se ovaj termin univerzalno koristi.

Osim PC registra, važan je i registar instrukcija ili instrukcijski registar (en. instruction register, *IR*) koji u sebi sadrži instrukciju koja se trenutno izvršava.

Većina računara ima brojne druge registre, neki od njih su opšte namene, a neki služe za posebne svrhe.

Takođe, neke registre koristi operativni sistem za kontrolu računara.

ORGANIZACIJA CPU-A

Putanja podataka sastoji se od registara, ALU i magistrala koje ih povezuju.

Na slici je prikazana unutrašnja organizacija dela jednostavnog von *Njumano*v-og CPU-a, sa fokusom na tzv. *putanji podataka* (en. data path), koja se sastoji se od registara (obično 1 do 32), ALU i nekoliko magistrala koje sve to povezuju.

U ovoj arhitekturi postoje dva ulazna registra koji su direktno povezani na ALU, označena sa **A** i **B** na slici. U ovim registrima smešteni su operandi dok ALU izvršava neko izračunavanje.

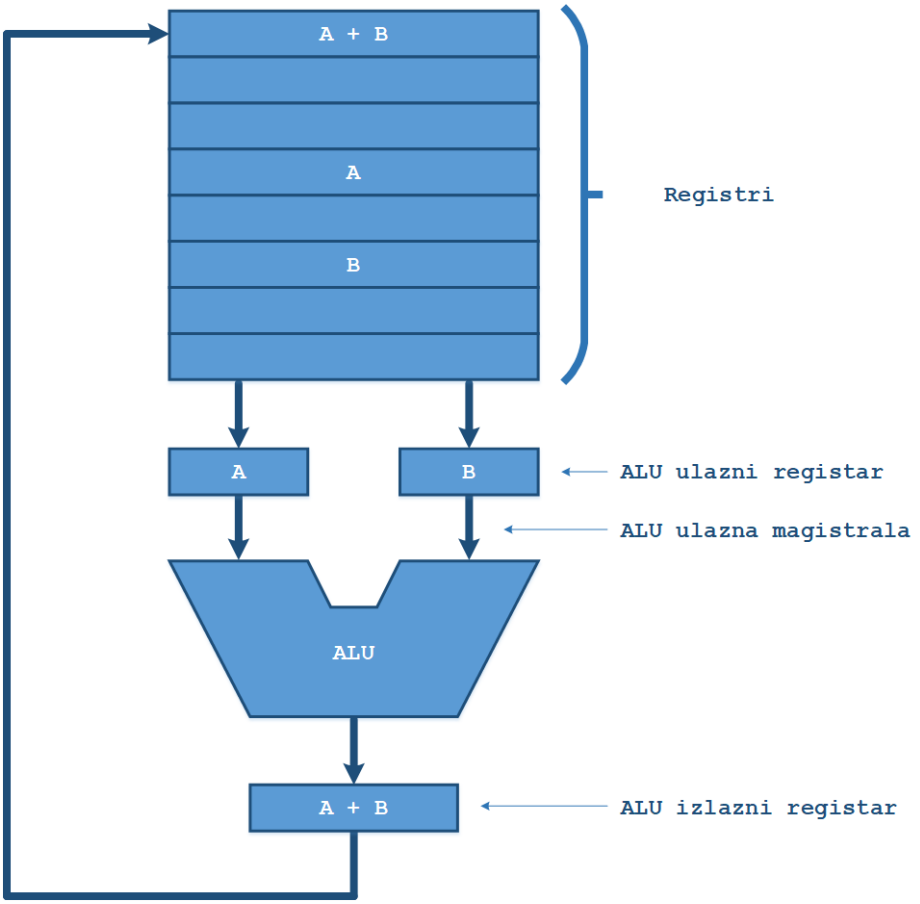
Sam ALU vrši sabiranje, oduzimanje i druge jednostavne operacije nad ***operandima*** koji su na ulazu, a rezultat smešta u izlazni registar.

Podaci sa izlaznog registra mogu biti pohranjeni nazad u neki od registara opšte namene.

Podatak iz registra se može upisati ili uskladištiti u operativnu memoriju.

U primeru na slici ilustrovana je operacija sabiranja, ali ALU može da obavlja i druge operacije.

Napomena:
Neke arhitekture CPU-a nemaju dva ulazna registra za ALU.



Slika 1.2 Putanja podataka u von Neumann CPU-u. [Izvor: Autor]

Poglavlje 2

Instrukcije procesora

VRSTE INSTRUKCIJA

Većina instrukcija se mogu podeliti u instrukcije tipa registar-memorija ili registar-registar.

Većina instrukcija se može podeliti u jednu od dve kategorije:

- registar-memorija;
- registar-registar.

Instrukcije tipa registar-memorija dozvoljavaju da se podaci iz memorije preuzimaju u registre, ili iz registara u memoriju.

Druga vrsta instrukcija je tipa registar-registar. Primer instrukcije registar-registar preuzima dva operanda iz registara, dovodi ih u ALU ulazne registre, izvodi neku operaciju nad njima (kao što je sabiranje ili logička AND operacija) i pohranjuje rezultat nazad u jedan od registara.

Proces pokretanja dva operanda kroz ALU i čuvanje rezultata naziva se *ciklus putanje podataka* (en. data path cycle) i predstavlja „srce“ većine CPU-a. Ovaj ciklus definiše šta računarska mašina može da uradi.

Savremeni računari imaju više ALU jedinica koje rade paralelno i specijalizovani su za različite funkcije.

Što je ciklus putanje podataka brži, mašina radi brže.

Primeri instrukcija (video objašnjenje):

Ova lekcija sadrži video materijal. Ukoliko želite da pogledate ovaj video morate da otvorite LAMS lekciju.

CIKLUS IZVRŠENJA INSTRUKCIJA

Svi procesori rade na principima ciklusa izvršenja instrukcija fetch-decode-execute.

CPU izvršava instrukcije u nekoliko malih koraka, koje su u opštem slučaju sledeće:

1. Pribavi sledeću instrukciju iz memorije i smesti je u instrukcijski registar,
2. Promeni programski brojač da pokazuje na sledeću instrukciju,
3. Odredi tip pribavljene instrukcije,
4. Ukoliko instrukcija koristi podatke iz memorije, pronađi taj deo memorije,
5. Pribavi deo memorije, ukoliko je potrebno, i smestiti u CPU registar,
6. Izvrši instrukciju,
7. Vrati se na korak 1 i krenuti sa izvršenjem sledeće instrukcije.

Ovaj niz koraka naziva se *pribavi-dekoduj-izvrši* (en. fetch-decode-execute). Svi procesori rade na ovim principima.

Rani računari su imali male, jednostavne skupove instrukcija. Vremenom se javila potreba jače računare, što je dovelo do, između ostalog, do kompleksnijih pojedinačnih instrukcija.

Primer fetch-decode-execute ciklusa (video):

Ova lekcija sadrži video materijal. Ukoliko želite da pogledate ovaj video morate da otvorite LAMS lekciju.

KOMPLEKSNE INSTRUKCIJE

Računari viskoih performansi bi izvršavali instrukcije direkno na hardveru, dok bi jeftiniji modeli interpretirali instrukcije.

Vrlo rano je otkriveno da kompleksnije instrukcije često dovode do bržeg izvršavanja programa, iako bi takve instrukcije pojedinačne trajale duže tj. imale duže izvršenje.

Primeri kompleksnih instrukcija:

- Instrukcija sa pokretnom zapetom,
- Direktan pristup elementu niza/liste,

Vrlo često je kompleksna instrukcija zamenila dve uzastopne jednostavnije instrukcije.

Ovakve složenije instrukcije bile su tretirane kao bolje, jer se izvršenje pojedinačnih instrukcija ponekad može i preklopiti ili na neki drugi način izvršavati paralelno korišćenjem različitog hardvera.

Za skupe računare visokih performansi, cena dodatnog hardvera koji bi podržao kompleksnije instrukcije bi se lako mogao opravdati. Na taj način, skupi računari visokih performansi imali su mnogo više instrukcija od jeftinijih.

Međutim, zahtevi kompatibilnosti instrukcija kada se koristi različit hardver i rastući troškovi razvoja softvera stvorili su potrebu za implementacijom složenih instrukcija čak i na računarima niže klase (npr. personalnih računara), gde je cena bila važnija od brzine.

Ovakvi računari bi imali interpretere za instrukcije, dok bi računari visokih performansi imali direktno izvršavanje kompleksnih instrukcija.

Jednostavniji računari sa interpretiranim instrukcijama su takođe imali i druge prednosti.

Među najvažnijima su bile sledeće:

1. Mogućnost popravljanja pogrešno implementiranih instrukcija „na terenu“, odnosno čak i nadoknaditi nedostatke projektovanja u osnovnom hardveru;
2. Mogućnost dodavanja novih instrukcija uz minimalne troškove, čak i nakon isporuka mašine.
3. Strukturirano projektovanje ovakvih mašina omogućilo je efikasan razvoj, testiranje i dokumentovanje složenih instrukcija.

Poglavlje 3

Projektovanje savremenih procesora

RICS NASPRAM CISC

Savremeni x68/x64 procesori su CISC arhitekture, uz neke primese RICS-a.

Kasnih 70tih godina eksperimentisalo se sa kompleksnim instrukcijama, sa ciljem da se smanji razlika u tom šta mogu mašine izvršiti i šta mogu uraditi programski jezici visokog nivoa.

Ovakvi procesori su nazvani *CISC* (en. Complex Instruction Set Computer).

RICS koncept krenuo je 80tih godina prošlog veka i zasnivao se na jednostavnim instrukcijama koje se mogle jako brzo izvršavati.

Ovakvi procesori koriste *RICS* (en. Reduced Instruction Set Computer) arhitekturu.

Savremeni x68/x64 procesori su CISC arhitekture, uz neke primese RICS-a.

Pravi savremeni RISC procesori koriste ARM arhitekturu ili Open Source RISC-V arhitekturu.

Primeri sličnih instrukcija u RICS i CISC arhitekturama:

Ova lekcija sadrži video materijal. Ukoliko želite da pogledate ovaj video morate da otvorite LAMS lekciju.

PRINCIPI PROJEKTOVANJA SAVREMENIH PROCESORA

Većina projektanta savremenih procesora teži da ispuni tzv. RISC principe projektovanja računara.

Sada, kada je prošlo više od tri decenije od uvođenja prvih RISC mašina, određeni principi projektovanja su postali prihvaćeni kao dobra praksa za projektovanje računara s obzirom na trenutno stanje hardverske tehnologije.

Primer:

Može doći do drastične promene u tehnologiji, tako da novi proizvodni proces procesora iznenada smanjuje vreme memorijskog ciklusa da bude 10 puta manji od vremena ciklusa trenutnih procesora.

Projektanti računarskih mašina treba uvek da obrate pažnju na tehnološke promene koje mogu uticati na ravnotežu između komponenti.

Međutim, postoji skup principa projektovanja, koji se ponekad naziva *RISC principi projektovanja* (en. RISC design principles), koji projektanti novih CPU-a opšte namene prate.

Različita spoljna ograničenja, kao što je zahtev za kompatibilnosti unazad sa nekom postojećom arhitekturom, često zahtevaju kompromise prilikom projektovanja, ali ovi principi daju ciljeve (en. objectives) koje većina projektanata teži da ispuni.

Ovi ciljevi su sledeći:

- Sve instrukcije treba da hardver direktno izvršava;
- Izvršenje instrukcija treba da bude što brže;
- Instrukcije treba da budu lake za dekovodanje;
- Samo instrukcije tipa *LOAD* i *STORE* treba da referenciraju operativnu memoriju;
- Treba imati što više registara.

UVOD U RISC-V ARHITEKTURU (VIDEO OBJAŠNJENJE)

Sledi video objašnjenje o RISC-V arhitekturi

Ova lekcija sadrži video materijal. Ukoliko želite da pogledate ovaj video morate da otvorite LAMS lekciju.

SPECIFIČNOSTI RAZLIČITIH ARHITEKTURA SAVREMENIH PROCESORA

Slede video objašnjenja o specifičnostima x86, ARM i RISC-V arhitektura savremenih procesora.

x86 naspram ARM arhitekture:

Ova lekcija sadrži video materijal. Ukoliko želite da pogledate ovaj video morate da otvorite LAMS lekciju.

ARM naspram RISC-V arhitekture:

Ova lekcija sadrži video materijal. Ukoliko želite da pogledate ovaj video morate da otvorite LAMS lekciju.

Poglavlje 4

Uvod u paralelni rad računarskih sistema

POTREBA ZA PARALELNIM IZVRŠENJEM INSTRUKCIJA

Pri projektovanju savremenih procesora, stalno se teži poboljšanju performansi.

Pri projektovanju savremenih procesora, stalno se teži poboljšanju performansi. Ubrzanjem radnog takta procesora je samo jedan način ubrzanja rada, ali postoji ograničenje koliko je moguće ostvariti brzinu procesora na ovaj način.

Danas se ubrzanje procesora ogleda u paralelizmu, tj. izvršenje više instrukcija odjednom.

U opštem slučaju, paralelizam (en. parallelism) se može ostvariti na nivou instrukcije i na nivou procesora.

- U prvom slučaju, paralelizam se ostvaruje izvršenjem više instrukcija po sekundi.
- U drugom slučaju, paralelizam se ostvaruje kada više CPU-a rade na istom problemu.

Paralelizam na nivou instrukcija (en. instruction-level parallelism, *ILP*) se može ostvariti na više načina.

U predmetu CS120 spomenućemo tzv. protočno izvršenje instrukcija, dok su ostali oblici instrukcijskog paralelizma spomenuti u sledećem video objašnjenju:

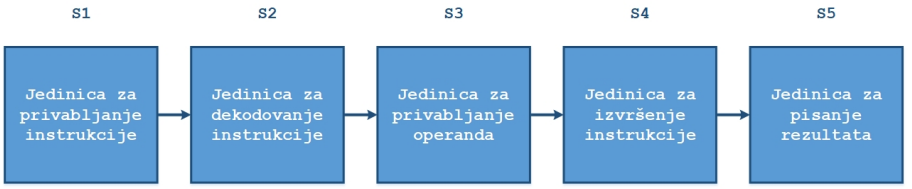
Ova lekcija sadrži video materijal. Ukoliko želite da pogledate ovaj video morate da otvorite LAMS lekciju.

PROTOČNO IZVRŠENJE INSTRUKCIJA

Koncept protočnog izvršenja instrukcija radi na principu podela instrukcijskog ciklusa na više nezavisnih delova.

Koncept protočnog izvršenja instrukcija (en. pipelining) radi na principu podela instrukcijskog ciklusa na više delova, od kojih svaka je faza (en. stage) dodeljena jedinstvenom hardveru, od kojih svaki mogu raditi paralelno.

Na slici je prikazano protočno izvršenje instrukcija.



Slika 4.1 Protočno izvršenje instrukcija u pet faza. [Izvor: Autor]

Faza 1 preuzima instrukciju iz memorije i smešta je u bafer dok ne bude potrebna.

Faza 2 dekoduje instrukciju, određujući njen tip i operande koje su potrebni toj instrukciji.

Faza 3 pronalazi i preuzima operande, bilo iz registara ili iz operativne memorije.

Faza 4 zapravo obavlja posao izvršavanja instrukcije kroz prethodno objašnjenu putanju podataka.

Konačno, faza 5 upisuje rezultat nazad u odgovarajući registar.

Protočno izvršenje instrukcija (video objašnjenje):

Ova lekcija sadrži video materijal. Ukoliko želite da pogledate ovaj video morate da otvorite LAMS lekciju.

PRIMER: PROTOČNO IZVRŠENJE INSTRUKCIJA U FUNKCIJI VREMENA

Dok se odvijaju faze S1-S5 za instrukciju 1, faza S1 može odmah da radi sa sledećim instrukcijama koje nailaze.

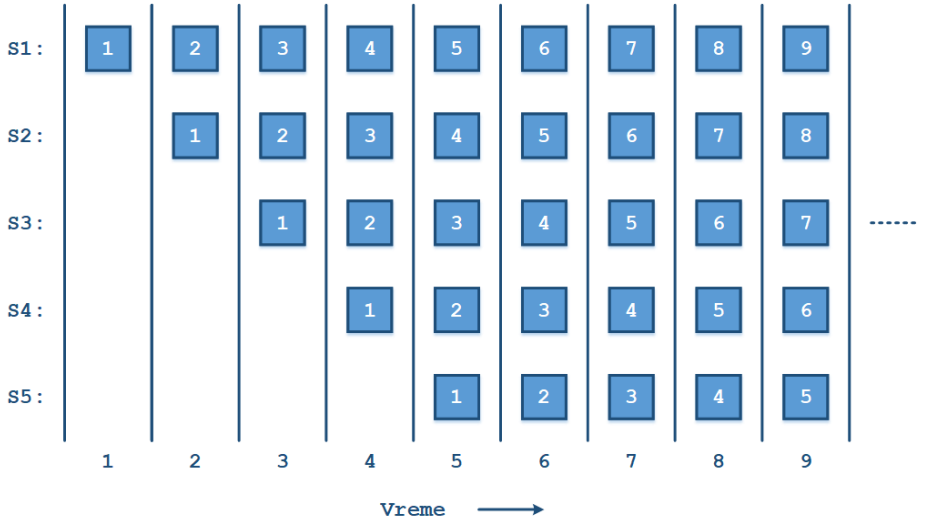
Na slici vidimo kako protočno izvršenje instrukcija funkcioniše kao u funkciji vremena.

U toku takt ciklusa 1, faza S1 radi na instrukciji 1, preuzimajući je iz memorije.

U toku ciklusa 2, faza S2 dekodira instrukciju 1, dok faza S1 preuzima instrukciju 2.

Tokom ciklusa 3, faza S3 preuzima operande za instrukciju 1, faza S2 dekodira instrukcija 2, a faza S1 preuzima treću instrukciju.

Tokom ciklusa 4, faza S4 izvršava instrukciju 1, S3 preuzima operande za instrukciju 2, S2 dekodira instrukciju 3, a S1 preuzima instrukciju 4. Konačno, u ciklusu 5, S5 upisuje rezultat instrukcija 1 nazad u registar (ili memoriju), dok ostale faze rade prema svojim instrukcijskim ciklusima.



Slika 4.2 Stanja svake faze u funkciji vremena. Ilustrovano je devet ciklusa. [Izvor: Autor]

PARALELIZAM NA NIVOU PROCESORA

O paralelnim arhitekturama biće detaljnije rečeno u posebnoj lekciji.

Instrukcijski paralelizam može poboljšati performanse procesora najviše 5 do 10 puta, što za mnoge primene nije dovoljno.

Jedino se projektovanjem računara koji koriste više CPU-a može postići poboljšanje od 50, 100 ili više puta u odnosu na jednoprocesorski računar.

Sledi video objašnjenje o nedostacima ILP-a:

Ova lekcija sadrži video materijal. Ukoliko želite da pogledate ovaj video morate da otvorite LAMS lekciju.

Napomena:

O paralelizmu na nivou procesira (en. procesor-level parallelism) biće detaljno reči u posebnoj lekciji.

Poglavlje 5

Operativna memorija

UVOD U OPERATIVNU MEMORIJU

Binarni brojevni sistem zahteva samo dve vrednosti koje se trebaju razlikovati. Stoga je to najpouzdanija metoda kodiranja digitalnih informacija.

Memorija (en. memory) je deo računara u kome se čuvaju programi i podaci. Neki naučnici iz oblasti računarske tehnike (posebno britanski naučnici) koriste termin skladište ili skladište umesto memorije, iako se sve više i više termin „skladištenje“ koristi za označavanje diskova za skladištenje.

Bez memorije iz koje procesori mogu da upisuju i čitaju informacije, ne bi bilo digitalnih računara koje danas poznajemo.

Osnovna jedinica memorije jebinarna cifra (en. binary digit), nazvan bit. Bit može sadržati 0 ili 1. To je najjednostavnija moguća jedinica.

Ljudi često kažu da računari koriste binarnu aritmetiku jer je "efikasna". Ono što misle (iako to retko shvataju) jeste da se digitalne informacije mogu skladištiti tako da se razlikuju različite vrednosti neke kontinuirane fizičke količine, kao što je napon ili struja. Što se mora razlikovati između više vrednosti, manje je razlike između susednih vrednosti, a memorija je manje pouzdana.
Binarni brojevni sistem zahteva samo dve vrednosti koje se trebaju razlikovati. Stoga je to najpouzdanija metoda kodiranja digitalnih informacija.

Bitovi i bajtovi (video objašnjenje):

Ova lekcija sadrži video materijal. Ukoliko želite da pogledate ovaj video morate da otvorite LAMS lekciju.

MEMORIJSKA ĆELIJA

Bitovi se grupišu u bajtove, a bajtovi u memorijske reči.

Memorija se sastoji od ćelija (ili lokacija), kod kojih svaka možete skladištiti neku informaciju. Svaka ćelija sadrži broj, koja se naziva adresa (tzv. *adresa memorijske lokacije*) kojoj pristupaju programi.

Ukoliko memorija ima n ćelija, imaće adrese od 0 do n-1.

Sve ćelije u memoriji mogu sadržati jednak broj bitova.

Ukoliko se ćelija sastoji od ***k*** bitova, onda može sadržati bilo koju informaciju od $\mathbf{2^k}$ kombinacija.

Računari koji koriste binarni brojni sistem (uključujući i heksadecimalnu i oktalnu reprezentaciju binarnih brojeva) izražavaju memorijske lokacije kao binarne brojeve.

Ako adresa ima **m** bitova, onda je maksimalni broj mogućih ćelija koje se mogu adresirati $\mathbf{2^m}$.
Značaj ćelije jeste da je to najmanja jedinica memorija koja se može adresirati.
Danas računari koriste standardizovanu 8-bitnu ćeliju koja se naziva bajt (en. byte), a bajtovi se grupišu u memorijske reči (en. word).

Računar sa 32-bitnom rečju ima 4 bajta po reči, dok računar sa 64-bitnom rečju ima 8 bajtova po reči.

Značaj reči jeste da mnoge instrukcije koriste više reči, npr. da sabiraju više reči.
Na ovaj način, 32-bitna mašina imaće 32-bitne registre za manipulisanje 32-bitnih reči, dok će 64-bitna mašina imati 64-bitne registre sa instrukcijama za manipulisanje 64-bitnih reči.

UBRZAVANJE OPERATIVNE MEMORIJE

Operativna memorija se može ubrzati korišćenjem keš memorije kao svojsvenog bafera u komunikaciji sa procesorom.

Procesori računara, tj. registri u procesorima su istorijski uvek bili brži nego operativna memorija. Vremenom se brzina memorije povećavala, ali se povećavala i brzina procesira, samim tim i registara, pa se održavala razlika u brzinama.

Moguće je projektovati operativnu memoriju koja je brza kao i CPU, ali postoje dva problema:

- Operativna memorija bi morala biti na čipu CPU-a, jer je prolazak kroz magistralu od CPU-a ka memoriji i obrnuto sporo,
- Izrada ovakve operativna memorije bi bilo skupo u odnosu na konvencijalne, uz nemogućnost nadogradnje.

Ukoliko memoriju malog kapaciteta postavimo na CPU, i ako je kombinujemo sa memorijom velikog kapaciteta koja je sporija, možemo relativno jeftino dobiti performanse koje nisu identične kao kada bi celu operativnu memoriju smestili na procesor, ali bi svakako dobili znatno ubrzanje.
Ovaj koncept doveo je do keš memorije (en. cache, od fr. cacher, što znači sakriti).

Ideja je jednostavna – najšešće korišćene reči se smeštaju i drže u keš memoriji.

CPU najpre traži reč u keš memoriji, a tek ako ta reč nije u keš memoriji, onda se traži od glavne memorije.

Napomena:

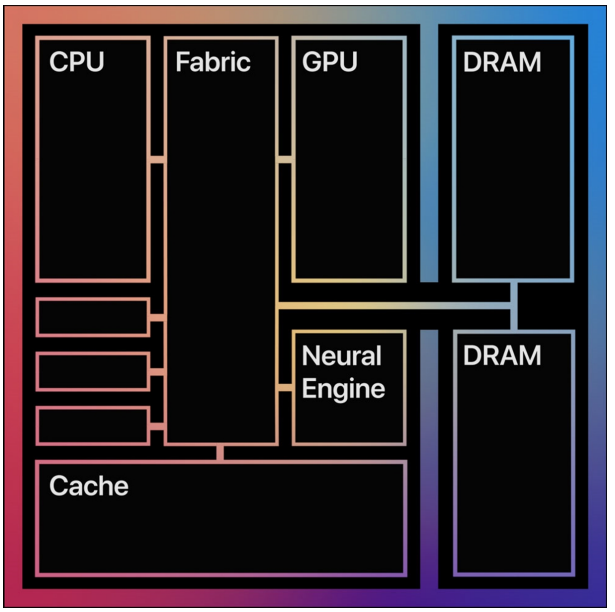
Detaljni funkcionisanja keš memorije obradiće seu posebnoj lekciji.

SISTEM NA ČIPU

Savremeni mobilni uređaji koriste operativnu memoriju koja se nalazi direktno na čipo procesora. Takvi sistemi nazivaju se sistemi na čipu - SoC

Procesori na savremenim mobilnim uređajima (pametni telefoni, tableti i sl.) uglavnom imaju objedinjene procesore i operativnu memoriju, pa se ova arhitektura naziva sistem na čipu (en. system-on-a-chip, *SoC*).

Apple M1 i M2 procesori bazirani na ARM arhitekturi su takođe SoC.



Slika 5.1 Izgled arhitekture Apple M1 SoC. [Izvor: https://arstechnica.com/gadgets/2020/11/the-first-arm-based-mac-with-apple-silicon-is-tk-name/]

Slika 5.2 Fizički izgled Apple M1 SoC. [Izvor: https://en.wikipedia.org/wiki/Apple_M1]

Poglavlje 6

Pokazne vežbe

PREDSTAVLJANJE NEGATIVNIH BROJEVA

Negativni binarni brojevi se mogu prestaviti na četiri načina.

Negativni brojevi se mogu binarno predstaviti na četiri načina:

- **Označena amplituda**
  Podrazumeva da se negativnim brojevima dodaje 1 kao dodatna najteža cifra. Za pozitivne brojeve dodaje se 0.
- **Prvi (jedinični) komplement**
  Za znak se takođe koristi 0 za pozitivne brojeve i 1 za negativne brojeve. Kod negativnih brojeva, se zamenjuje svaka 1 sa 0 i svaka 0 sa 1.
- **Drugi (dvojčani) komplement**
  Ovde je takođe cifra znaka 0 za pozitivne brojeve i 1 za negativne. Za negativne brojeve se uvode dodatna dva koraka. Prvo se svaka 1 zamenjuje sa 0 i svaka 0 sa 1, kao u predhodnom slučaju. Drugo, svakom ovako dobijenom broju se dodatno dodaje 1.
- **Binarno predstavljanje sa pomerajem**

OZNAČENA AMPLITUDA

Kod predstavljanje negativnih brojeva označenom amplitudom, uglavnom dodajemo jedan bit koji označava znak.

Kod predstavljanje negativnih brojeva označenom amplitudom, uglavnom dodajemo jedan bit koji označava znak.

Pa pozitivne brojeve dodaje se nula, a za negativne brojeve dodaje se jedinica.

Primer #1 (5 minuta):

Pokazati predstavljanje binarnih brojeva od 0 do 9 koristeći označenu amplitudu:

Rešenje:

```
| DEC  |   BIN    | Signed BIN  |
|   0  | 00000000 |    00000000 |
|   1  | 00000001 |    00000001 |
|   2  | 00000010 |    00000010 |
|   3  | 00000011 |    00000011 |
|   4  | 00000100 |    00000100 |
```

```
|   5  | 00000101 |    00000101 |
|   6  | 00000110 |    00000110 |
|   7  | 00000111 |    00000111 |
|   8  | 00001000 |    00001000 |
|   9  | 00001001 |    00001001 |
```

Primer #2 (5 minuta):

Pokazati predstavljanje binarnih brojeva od -1 do -9 koristeći označenu amplitudu:

Rešenje:

```
| DEC  | Signed BIN  |
|      |             |
|  -1  |    10000001 |
|  -2  |    10000010 |
|  -3  |    10000011 |
|  -4  |    10000100 |
|  -5  |    10000101 |
|  -6  |    10000110 |
|  -7  |    10000111 |
|  -8  |    10001000 |
|  -9  |    10001001 |
```

PRVI KOMPLEMENT

Kod prvog komplementa negativni brojevi su inverzovani pozivitni - nule i jedinice menjaju mesta.

Kod prvog komplementa ili komplementa jedinice (en. one's complement), za znak se takođe koristi 0 za pozitivne brojeve i 1 za negativne brojeve.

Međutim, kod negativnih brojeva se zamenjuje svaka 1 sa 0 i svaka 0 sa 1.

Primer #3 (5 minuta):

Napisati prvi komplement za brojeve od 0 do 9.

Rešenje:

```
| DEC  |   BIN    | One's comp. |
|   1  | 00000001 |    11111110 |
|   2  | 00000010 |    11111101 |
|   3  | 00000011 |    11111100 |
|   4  | 00000100 |    11111011 |
|   5  | 00000101 |    11111010 |
|   6  | 00000110 |    11111001 |
|   7  | 00000111 |    11111000 |
|   8  | 00001000 |    11110111 |
|   9  | 00001001 |    11110110 |
```

DRUGI KOMPLEMENT

Drugi komplement se nadovezuje na prvi komplement sabiranjem jedinice na prvi komplement.

Kod drugog komplementa ili komplementa dvojke (en. two's complement), takođe je cifra znaka 0 za pozitivne brojeve i 1 za negativne.

Za negativne brojeve se uvode dodatna dva koraka.

Prvo se svaka 1 zamenjuje sa 0 i svaka 0 sa 1, kao u prethodnom slučaju. Zatim, svakom ovako dobijenom broju se dodatno dodaje 1.

Primer #4 (5 minuta)

Napisati drugi komplement za brojeve od 0 do 9.

Rešenje:

```
| DEC  |   BIN    | One’s comp. | Two’s comp. |
|   1  | 00000001 |    11111110 |    11111111 |
|   2  | 00000010 |    11111101 |    11111110 |
|   3  | 00000011 |    11111100 |    11111101 |
|   4  | 00000100 |    11111011 |    11111110 |
|   5  | 00000101 |    11111010 |    11111011 |
|   6  | 00000110 |    11111001 |    11111010 |
|   7  | 00000111 |    11111000 |    11111001 |
|   8  | 00001000 |    11110111 |    11111000 |
|   9  | 00001001 |    11110110 |    11110111 |
```

BINARNO PREDSTAVLJENJE SA POMERAJEM

Brojevi se uvek predstavljaju kao pomeraj (offset) od određenog referentnog broja.

Bonarno predstavljanje brojeva sa pomerajem (en. offset binary) tretira brojeve sa pomerajem od određene vrednosti.

Kôd četvrtog sistema za **m**-bitne brojeve (brojeve sa **m** cifara), koji se nazivaju *excess* $2^{(m-1)}$, predstavljaju se brojevi koji se dobijaju kao sabirci samog binarnog broja i vrednosti broja i $2^{(m-1)}$.

Primer:

Pomeraj sa 8-bitnim brojevima.

Svakom 8-bitnom broju dodaje se vrednost 128 (odnosno polovina maksimalne vrednosti broja).

$$m = 8 \Rightarrow 2^{m-1} = 2^7 = 128$$

Primer #5 (5 minuta):

Napisati broj -3 sa u offset notaciji ako se zna da se koriste 8-bitni brojevi.

Broj -3 postaje -3+128=125 i piše se 01111101.

PREDSTAVLJANJE RAZLOMLJENIH BROJEVA

Različiti proizvođači su koristili različite algoritme u zaokruživanju vrednosti, izračunavanju vrednosti elementarnih funkcija i izvođenju osnovnih aritmetičkih operacija.

Do sada je bilo reči o predstavljanju samo celih brojeva.

Međutim, potrebno je naći uniformni način predstavljanja razlomljenih brojeva tj. brojeva sa pokretnom tačkom (en. floating point)

Broj **n** sa pokretnom tačkom se predstavlja kao proizvod *mantise* **f** i stepena broja na *eksponent* **e.**

$$n = f \times 10^e$$

Primer #6 (5 minuta):
Neka je dat model sa 14-bita, sa 1-bitnim znakom (en. sign bit), 5-bitnim eksponentom i 8-bitnom mantisom.

Broj se može onda predstaviti na sledeći način:

```
|0|01100|1000000|
```

Kod ovakvog predstavlja brojeva, dolazi do sledećeg problema:
Opseg vrednosti brojeva je determinisan brojem cifara u eksponentu, dok je preciznost određena brojem cifara u mantisi.

Zbog toga treba predstaviti brojeve na uniformni način.

Različiti proizvođači su koristili različite algoritme u zaokruživanju vrednosti, izračunavanju vrednosti elementarnih funkcija i izvođenju osnovnih aritmetičkih operacija.

Prenosivost numerički intenzivnih programa između različitih računarskih sistema bila vrlo slaba, često uz razliku u rezultatima izvršavanja takvih programa.

IEEE (en. Institute of Electrical and Electronics Engineers) je uveo standard 754 koji služi za prikaz realnih brojeva.

99% procesora koristi IEEE 754 standard.

IEEE 754 STANDARD ZA PREDSTAVLJANJE RAZLOMLJENIH BROJEVA

IEEE 754 standard poseduje tri formata za predstavljanje realnih brojeva: standardni format, format dvostruke tačnosti, i format proširene tačnosti.

Formati IEEE 754 standarda

Do pojave Standarda IEEE 754, svaki proizvođač računara je imao svoju formu zapisa cifara u pokretnom zarezu. Pored toga, proizvođači su koristili različite algoritme u zaokruživanju vrednosti, izračunavanju vrednosti elementarnih funkcija i izvođenju osnovnih aritmetičkih operacija.

Zbog toga je prenosivost numerički intenzivnih programa, između različitih računarskih sistema – bila vrlo slaba, često uz razliku u rezultatima izvršavanja takvih programa.

Postoje tri formata u ovom standardu:

- Standardni format sa 32 bita (en. single precision)
- Format dvostruke preciznosti sa 64 bita (en. double precision):
- Format proširene tačnosti sa 80 bitova (en. extended precision). Ovaj format je predviđen da umanji greške, koje nastaju zaokruživanjem.

STANDARDNI FORMAT IEEE 754 STANDARDA

Primer standardnog formata IEEE 754 standarda

Standardni format poseduje 32 bita (4 bajta) za predstavljanje realnih brojeva.

Tih 32 bita dele se na tri dela:

- *Bit znaka* (en. sign bit) - nula predstavlja pozitivan broj, a jedinica negativan broj.
- *Eksponent* (en. exponent) - 8 bitova za eksponent koji je pomeren za offset of 127
- Razlomljeni deo - *frakcija* ili *mantisa* (en. fraction) - 23 bita koji označavaju deo nakon pokretne tačke.

Računanje decimalnog ekvivalenta broja vrši se po sledećoj formuli:

$$number = (-1)^{signbit} \times 2^{exponent - bias} \times (1 + fraction)$$

Računanje frakcije biće objašnjeno na primeru.

Primer #7 (10 minuta):

Dat je binarni broj 11110100110101000101001101010101 koji koristi format IEEE 754. Pronaći decimalni ekvivalent broja i zapisati u formatu.

Rešenje:

Sign bit = 1 (negativan je broj u pitanju)
Bias = 127 (broj bitova u delu za eksponent – 1)
Eksponent se najpre računa predvaranjem bitova za eksponent u decimalnu vrednost, nakon čeka od tog broja oduzimamo bias:

$(11101001)_2 = (233)_{10}$

233 – 127 = **106**

Kod frakcije, prvi bit f_1 predstavljanje $f_1 \times 2^{-1}$, drugi $f_2 \times 2^{-2}$ i sve do $f_{23} \times 2^{-23}$

Bitovi f_1 do f_{23} imaju vrednosti 0 ili 1

U primeru, vrednost frakcije je sledeća:
$2^{-1} + 2^{-3} + 2^{-5} + 2^{-9} + 2^{-11} + 2^{-14} + 2^{-15} + 2^{-17} + 2^{-19} + 2^{-21} + 2^{-23}$

Konačno, broj pišemo u sledećem formatu:

$$n = (-1) \times 2^{126} \times (1 + 2^{-1} + 2^{-3} + 2^{-5} + 2^{-9} + 2^{-11} + 2^{-14} + 2^{-15} + 2^{-17} + 2^{-19} + 2^{-21} + 2^{-23})$$

PREDSTAVLJANJE SPECIJALNIH VREDNOSTI

Postoje normalnizovani i denormalizovani brojevi u IEEE 754 standardu.

Prema kombinaciji vrednosti eksponenta i frakcije, podaci u IEEE 754 zapisu se dele u klase koje uključuju numeričke i odgovarajuće ne-numeričke entitete.

1. Normalizovani brojevi

- poseduju eksponent 0<e<max,
- implicitni bit je 1,
- frakcija može imati bilo koju vrednost.

2. Denormalizovani brojevi

- frakcija ne mora da bude normalizovana,
- Eksponent je jednak nuli.

Nula se predstavlja sa eksponentom i frakcijom koja se sastoji iz svih nula

Beskonačno se predstavlja tako što je eksponent takođe sastavljen iz svih jedinica, dok je frakcija jednaka nulama.

NaN (en. not-a-number)

Tradicionalno, izračunavanje kao što je deljenje sa nulom (0/0) ili kvadratni koren iz -1 (sqr(-1)), se smatralo nepopravljivom greškom koja je dovodila do zaustavljanja izračunavanja.

Međutim u nekim situacijama ima smisla nastaviti sa izračunavanjem. Ovakvi problemi se izbegavaju uvođenjem specijalnih vrednosti koje nisu brojevi.
Označavaju se na taj način što se **eksponent sastoji iz svih jedinica**, dok frakcija ***može imati bilo koju vrednost različitu od nule***.

Poglavlje 7

Zadaci za samostalni rad

IEEE 754 STANDARD - ZADACI ZA SAMOSTALNI RAD

Zadaci za samostalni rad mogu se uraditi za okvirno 30 minuta

Dati su brojevi u binarnom formatu koji su napisani koristeći IEEE 754 standard. Prebaciti brojeve u dekadni ekvivalent za zapisom:

$$number = (-1)^{signbit} \times 2^{exponent - bias} \times (1 + fraction)$$

Napomena:

Delovi za bit znaka, eksponent i frakciju su odvojeni uspravnim crtama.

```
|1|11101011|10001001011100101010100|
|0|01001101|01001101001001101011011|
|1|11001011|10101001011000001010110|
|1|00101011|01001111001101001010110|
|0|10101001|10001001011001101110110|
|0|10101111|01001001001011101011110|
|1|01001010|10101101011000101010111|
|0|11101101|10001001101001001110110|
|1|10101011|00101001001011001010100|
|0|10101110|00101011011011101000101|
|1|10101010|10110011011010101010001|
|0|10101110|00101011011011100000101|
|1|00101101|10100011111011001010001|
```

Rešenja zadataka za samostalni rad mogu se proveriti koristeći online konverter:

https://www.h-schmidt.net/FloatConverter/IEEE754.html

Zadaci za samostalni rad u Python programskom jeziku

Napisati program koji će izvršiti konverziju binarnog broja od 32 bita u decimalni broj koristeći IEEE 754 standard. Program treba da prikaže broj u zapisu kroz bit znaka, eksponenta i frakcije, ali i pravi decimalni ekvivalent broja.

Poglavlje 8

Domaći zadatak

DOMAĆI ZADATAK #2

Domaći zadatak #2 okvirno se radi 30 minuta.

Uputstvo za odabir zadataka
Student bira zadatke tako što radi zadatke koji se nalaze u broju indeksa, uzima zadnje dve cifre svog indeksa.
Primeri:

- Broj indeksa 2318 – student radi 1. i 8. zadatak,
- Ukoliko se indeks završava sa dve iste cifre, npr.1244, student radi 4. i 5. zadatak,
- Ukoliko je neka od cifara 0, npr 1230 student radi zadatak 3. i 10 zadatak,
- Ukoliko su obe cifre nule, student radi 1. i 10. zadatak.

Domaći zadatak #2:

Dat je floating point broj u IEEE 754 formatu od 32 bita, 1 bit za znak, 8 bita za eksponent i 23 bita za frakciju (mantisu).

Odrediti decimalni ekvivalent broja, u formatu:

$$number = (-1)^{signbit} \times 2^{\,exponent\,-bias} \times (1 + fraction)$$

```
Zadatak #01:  |0|00010100|01110110100011010101011|
Zadatak #02:  |1|10110010|10110010110110010100100|
Zadatak #03:  |0|00110100|01010110100111110101001|
Zadatak #04:  |0|11010100|10110000110010110101001|
Zadatak #05:  |1|01010110|01110110100110010001001|
Zadatak #06:  |1|01010000|10110110110100010100001|
Zadatak #07:  |0|10110101|01010010100111010101000|
Zadatak #08:  |1|00010010|01110110010110110001001|
Zadatak #09:  |0|01010100|11010110110100110101011|
Zadatak #10:  |1|01010001|11010100100100010111010|
```

Domaće zadatke slati odgovarajućem predmetnom asistentu, sa predmetnim profesorom u CC. Predati domaći zadatak koristeći .doc/docx uputstvo dato u prošloj lekciji.

Poglavlje 9

Zaključak

ZAKLJUČAK

Rezime lekcije #2

U lekciji #2 najpre je bilo reči o procesorima i gradivnim blokovima procesora.

Zatim, bilo je reči o instrukcijama, kategorijama instrukcija i ciklusu izvršenja instrukcija.

Dat je osvrt na projektovanje savremenih računara koristeći kako CISC, tako i RISC arhitekture.

Tadi su uvodni pojmovi o paralelnom radu, prvenstveno na protočno izvršenje instrukcija.

Konačno, dati su uvodni pojmovi o operativnoj memoriji, bitovima, bajtovima, rečima i ćelijama, keš memoriji i načinima ubrzavanja operativne memorije.

Na vežbama obrađeno je mašinsko predstavljanje negativnih brojeva, kao i predstavljanje brojeva sa pokretnom tačkom.

Literatura:

A. Tanenbaum, Structured Computer Organization, Chapter 02, pp. 26 – 85, Appendix A, pp. 681 – 691