Akademska 2023/24. godina

Semestar: IV - četvrti

Predmet: Arhitektura računara

Predmetni profesor: doc. dr. Admir Midžić

Predmetni asistent: Azra Toroman

Studenti: Hadis Mahmutović (br. index: 1271) i Suad Kucalović (br. index: 1272)

**Materijal za polaganje usmenog/završnog ispita**

1. Navesti osnovne karakteristike obrade podataka kod analogni i digitalnih računara!

* Analogni računari i digitalni računari imaju različite karakteristike obrade podataka, kod analognih računara vrijedi:
  + Koriste kontinuirane fizičke veličine za predstavljanje podataka, poput električnog napona ili struje.
  + Obrada podataka se vrši kontinuirano, što znači da se vrijednosti podataka neprekidno mijenjaju.
  + Analogni računari su pogodni za obradu analognih signala poput zvuka ili slike.
  + Preciznost analognih računara ograničena je rezolucijom komponenti i mogućim šumovima u signalima.
* Dok kod digitalnih računara vrijedi:
  + Koriste diskretne binarne vrijednosti (0 i 1) za predstavljanje podataka.
  + Obrada podataka se vrši diskretno, gdje se podaci obrađuju u koracima ili ciklusima.
  + Digitalni računari su pogodni za precizne matematičke operacije i logičke procese.
  + Preciznost digitalnih računara je visoka zbog jasnog razdvajanja između nula i jedinica.
* Ove karakteristike čine analogne i digitalne računare prikladnim za različite vrste aplikacija i obrade podataka.

1. Osnovna arhitektura digitalnog računara – skicirati model i šta je čini?



* U računaru, memorija sadrži kod u mašinskom jeziku i podatke koji se obrađuju. Procesor mora efikasno donositi instrukcije i podatke iz memorije, te nakon obrade vraćati rezultate nazad. Uobičajeno je da se u memoriji nalazi i stek, koji se koristi za pozivanje funkcija i procedura, te pomaže u organizaciji programskih apstrakcija.
* Osnovne komponente centralne procesne jedinice (CPU) uključuju programski brojač (PC), aritmetičko-logičku jedinicu (ALU), skup registara i registar stanja (CC - Condition Code). ALU obavlja operacije nad operandima iz registara, koji se prethodno moraju učitati iz memorije. Programski brojač pokazuje na instrukciju koja se trenutno izvršava, a zatim se preusmjerava na sljedeću instrukciju kako bi se obezbijedilo sekvenciranje instrukcija. Upravljačka jedinica upravlja svim komponentama CPU-a kako bi obavile odgovarajuće zadatke u pravom trenutku.
* Izvršenje instrukcija ažurira stanje CC registra, koji se često koristi za donošenje odluka u programu, kao što su grananja i petlje. Ove komponente su osnovne za sve procesore, bez obzira na njihovu složenost, i predstavljaju standardnu organizaciju procesora u posljednjim decenijama.

1. Objasni ukratko kroz pregled tipičnih instrukcija Von Neumannovog računara princip rada u ovom modelu!

* Von Neumannov model računara je osnovni konceptualni model digitalnog računara koji se koristi za izvršavanje instrukcija. Pregled tipičnih instrukcija i principa rada u Von Neumannovom modelu:
  + Učitavanje instrukcije (Fetch):
    - Procesor učitava instrukciju iz memorije u kontrolnu jedinicu.
    - Programski brojač (PC) se inkrementira kako bi pokazivao na sljedeću instrukciju.
  + Dekodiranje instrukcije (Decode):
    - Kontrolna jedinica dekodira instrukciju kako bi odredila koja operacija treba biti izvršena.
  + Izvršavanje instrukcije (Execute):
    - Aritmetičko-logička jedinica (ALU) izvršava operaciju navedenu u instrukciji.
    - Podaci se mogu čitati iz memorije, obrađivati u ALU i zatim zapisati nazad u memoriju.
  + Ponavljanje ciklusa:
    - Proces se ponavlja za svaku instrukciju u programu, s PC-om koji se povećava za svaku izvršenu instrukciju.
* Ovaj ciklus učitavanja, dekodiranja i izvršavanja instrukcija omogućava računaru da izvršava programski kod korak po korak. Von Neumannov model obezbjeđuje efikasnu obradu podataka i instrukcija, čineći ga osnovom za modernu digitalnu računarsku arhitekturu.

1. Struktura procesora se može podijeliti u tri osnovna funkcionalna dijela - koja su to tri dijela i šta je uloga/funkcija svakog od njih?

* Struktura procesora se obično dijeli na tri osnovna funkcionalna dijela: kontrolnu jedinicu (Control Unit), aritmetičko-logičku jedinicu (ALU) i registre. Detaljno objašnjenje uloge svakog od ovih dijelova:
  + Kontrolna jedinica (Control Unit):
    - Kontrolna jedinica upravlja izvršavanjem instrukcija i koordinira rad ostalih dijelova procesora.
    - Njena uloga je da dekodira instrukcije, određuje redoslijed operacija i upravlja tokom podataka kroz procesor.
    - Kontrolna jedinica takođe generiše kontrolne signale koji aktiviraju odgovarajuće operacije u procesoru.
  + Aritmetičko-logička jedinica (ALU):
    - ALU obavlja aritmetičke operacije (kao što su sabiranje, oduzimanje, množenje, dijeljenje) i logičke operacije (kao što su AND, OR, NOT) nad podacima.
    - Ova jedinica je odgovorna za izvođenje matematičkih i logičkih operacija koje su neophodne za obradu podataka.
  + Registri:
    - Registri su male, brze memorije unutar procesora koje se koriste za skladištenje privremenih podataka, adresa ili kontrolnih informacija.
    - Postoje različiti tipovi registara, kao što su registar instrukcija (IR), programski brojač (PC), registri opšte namjene (general-purpose registers) za skladištenje podataka, itd.
    - Registri omogućavaju brz pristup podacima i olakšavaju izvršavanje instrukcija.
* Ova tri osnovna funkcionalna dijela procesora zajedno omogućavaju procesoru da izvršava instrukcije, obrađuje podatke i upravlja tokom podataka unutar računarskog sistema. Svaki od ovih dijelova ima ključnu ulogu u radu procesora i doprinosi efikasnosti i performansama računarskog sistema.

1. Šta su to osnovna, a šta izvedena logička kola? Objasniti na primjeru I, ILI i NILI kola!

* Osnovna logička kola su jednostavne logičke funkcije koje obavljaju osnovne operacije kao što su AND, OR i NOT. S druge strane, izvedena logička kola se grade kombinovanjem osnovnih logičkih kola kako bi se izvršile složenije logičke operacije.
* Osnovna logička kola:
  + AND kolo: AND kolo daje izlaz "1" samo kada su oba ulaza "1", inače daje izlaz "0". Na primjer, ako imamo ulaze A i B, izlaz će biti 1 samo ako su A i B oba jednaki 1.
  + ILI kolo: ILI kolo daje izlaz "1" ako je barem jedan od ulaza "1", inače daje izlaz "0". Na primjer, za ulaze A i B, izlaz će biti 1 ako je A ili B jednako 1.
  + NOT kolo: NOT kolo daje inverzni rezultat od ulaza, tj. daje izlaz "0" samo kada je ulaz "1", u suprotnom daje izlaz "1".
* Izvedena logička kola:
  + XOR kolo (isključivo ILI): XOR kolo daje izlaz "1" ako su ulazi različiti, a izlaz "0" ako su ulazi isti. Ovo se može postići kombinovanjem osnovnih kola, poput OR, AND i NOT.
  + NOR kolo (ne-ILI): NOR kolo daje izlaz "1" samo kada su oba ulaza "0", inače daje izlaz "0". Ovo se može postići kombinovanjem osnovnih kola, poput OR i NOT.
  + XNOR kolo (ekskluzivno ILI): XNOR kolo daje izlaz "1" ako su ulazi jednaki, a izlaz "0" ako su ulazi različiti. Ovo se može postići kombinovanjem osnovnih kola, poput XOR i NOT.
* Kombinovanjem osnovnih logičkih kola poput AND, OR i NOT, moguće je konstruisati složenija izvedena logička kola poput XOR, NOR i XNOR, što omogućava izvođenje različitih logičkih operacija u digitalnim sistemima.

1. Šta je logička funkcija? Skicirajte logičku funkciju za izraz: Y (A, B, C, D) = (A + B) · (A + C) · D

* Logička funkcija predstavlja matematički izraz koji opisuje logičku operaciju nad ulaznim podacima kako bi se dobio određeni izlaz. U digitalnoj logici, logička funkcija može biti definisana kao kombinacija logičkih operatora poput AND, OR, NOT, XOR itd.
* Dati izraz predstavlja logičku funkciju sa četiri ulazna signala (A, B, C, D) i jednim izlaznim signalom (Y). Skicirajmo logičku funkciju koristeći logičke operatore OR (+), AND (·) i ulazne signale A, B, C, D:
* U skiciranom dijagramu, ulazni signali A i B se prvo sumiraju koristeći OR operator, rezultat se zatim AND-uje sa A i C, i na kraju se rezultat AND-uje sa ulaznim signalom D kako bi se dobio izlazni signal Y. Ovaj dijagram ilustruje kako se logička funkcija (A + B) · (A + C) · D može realizovati korištenjem osnovnih logičkih operatora i ulaznih signala.

1. Šta je to ASCII kod? Korištenjem ASCII kodne tablice (kodna tablica će biti dostupna, ako se postavi slično pitanje), prikažite ASCII kod za tekst: “NAREDNIH DANA SE PROGNOZIRA LIJEPO VRIJEME” .

* ASCII (American Standard Code for Information Interchange) je standardni karakterni kod koji se koristi za predstavljanje teksta na računarima i drugim uređajima. Svaki karakter (slovo, broj, simbol) ima svoj jedinstveni ASCII kod koji se sastoji od 7 ili 8 bitova.
* 78 65 82 69 68 78 73 72 32 68 65 78 65 32 83 69 32 80 82 79 71 78 79 90 73 82 65 32 76 73 74 69 80 79 32 86 82 73 74 69 77 69

1. Šta je procesor i kako može biti realizovan? Objasni da li postoji razlika između pojmova procesor i mikroprocesor!

* Procesor je elektronska komponenta u računaru koja obavlja aritmetičke, logičke i kontrolne operacije nad podacima prema instrukcijama iz memorije. Procesor je mozak računarskog sistema i odgovoran je za izvršavanje programa i manipulaciju podacima.
* Procesor može biti realizovan na različite načine:
  + Mikroprocesor: Kao što je već pomenuto, mikroprocesor je integrisan u jednom čipu i obuhvata sve osnovne funkcionalnosti procesora. Mikroprocesori se koriste u većini modernih računara, uključujući desktop računare, laptopove, pametne telefone, tablete i druge uređaje.
  + Mikrokontroler: Mikrokontroler je vrsta mikroprocesora koji je specijalizovan za kontrolu ugrađenih sistema. On obično sadrži integrisane periferne uređaje poput analognih-digitalnih konvertora, tajmera, serijskih komunikacionih interfejsa i drugih komponenti koje su potrebne za kontrolu spoljnih uređaja.
  + Procesorski sklop: Procesor se može realizovati kao poseban sklop koji se sastoji od više integrisanih kola. Ovaj pristup se često koristi u specijalizovanim sistemima gde je potrebna visoka performansa ili specifične funkcionalnosti.
  + Mainframe procesori: Ovi procesori su dizajnirani za velike računarske sisteme i mainframe računare, koji su se tradicionalno koristili za obradu velikih količina podataka i transakcija u poslovnim okruženjima.
  + Procesorska jedinica u FPGA: Procesor se može implementirati kao deo Field-Programmable Gate Array (FPGA) čipa. FPGA omogućava programabilnost hardvera, što znači da se procesorska jedinica može konfigurisati prema potrebama aplikacije.
  + Procesor u obliku softvera: Virtualni procesori se mogu realizovati kao softverski emulirani procesori koji se izvršavaju na postojećem hardveru. Ovo se često koristi u razvoju i testiranju novih procesorskih arhitektura.
* Svaka od ovih metoda ima svoje prednosti i mane u pogledu performansi, fleksibilnosti, troškova i složenosti implementacije. Izbor načina realizacije procesora zavisi od specifičnih zahteva aplikacije i okoline u kojoj će procesor biti korišćen.
* Mikroprocesor je tip procesora koji je integrisan u jednom čipu (IC) i obuhvata sve osnovne funkcionalnosti procesora. Mikroprocesor sadrži jedinicu za izvršavanje instrukcija (ALU), registre, kontrolnu jedinicu, keš memoriju i ostale komponente potrebne za rad procesora.
* Razlika između pojmova procesor i mikroprocesor je u obimu funkcionalnosti i integraciji. Procesor se može odnositi na bilo koji tip procesora, uključujući i one koji nisu mikroprocesori, dok se mikroprocesor specifično odnosi na procesor integrisan u jednom čipu. Mikroprocesori su kompaktniji, jeftiniji za proizvodnju i lakši za implementaciju u različite uređaje u poređenju sa tradicionalnim procesorima.
* U suštini, svaki mikroprocesor je procesor, ali ne svaki procesor je mikroprocesor. Mikroprocesori su postali dominantni u računarskoj industriji zbog svoje efikasnosti, performansi i prilagodljivosti za različite primene.

1. Koje operacije podržava većina savremenih (Instruction Set Architecture - ISA) arhitektura? Objasniti!

* Većina savremenih Instruction Set Architecture (ISA) arhitektura podržava različite operacije koje omogućavaju izvršavanje različitih tipova instrukcija. Neke od najčešćih operacija koje podržava većina savremenih ISA arhitektura:
  + Aritmetičke i logičke operacije: Ove operacije obuhvataju osnovne matematičke operacije poput sabiranja, oduzimanja, množenja i deljenja, kao i logičke operacije poput AND, OR, XOR, NOT. Ove operacije su ključne za izvršavanje aritmetičkih i logičkih operacija nad podacima.
  + Prenos podataka: Ove operacije uključuju čitanje i pisanje podataka iz memorije ili u registre. To omogućava procesoru da manipuliše podacima i prenosi ih između različitih lokacija u memoriji ili registrima.
  + Upravljačke operacije: Upravljačke operacije obuhvataju grananje, skokove, pozive procedura i povratke, kao i upravljanje izuzecima i prekidima. Ove operacije omogućavaju kontrolu toka izvršavanja programa.
  + Sistemski pozivi: Ove operacije omogućavaju komunikaciju sa operativnim sistemom i upravljanje resursima kao što su memorija, ulazno/izlazni uređaji i druge sistemske funkcionalnosti.
  + Pomični zarez operacije: Ove operacije obuhvataju manipulaciju podacima u pomičnom zarezu, kao što su dodavanje, oduzimanje, množenje i deljenje brojeva u pomičnom zarezu.
  + String operacije: Ove operacije se koriste za manipulaciju nizovima karaktera ili bajtova, kao što su premještanje stringova, poređenje stringova i pretraživanje stringova.
  + Grafičke operacije: U nekim ISA arhitekturama, podržane su operacije nad pikselima, operacije kompresije/dekompresije slika i druge grafičke operacije.
* Ove operacije su ključne za izvršavanje različitih tipova instrukcija u programima i omogućavaju procesoru da obavlja različite zadatke efikasno i precizno. Različite ISA arhitekture mogu imati različite skupove operacija, ali većina savremenih arhitektura podržava slične osnovne operacije.

1. Šta je to Amdhal-ov zakon o čemu govori i kako se najjednostavnije može interpretirati?

* Amdahl-ov zakon je koncept koji se odnosi na teoretsko ograničenje ubrzanja koje se može postići u poboljšanju performansi sistema kroz paralelizaciju izvršavanja. Ovaj zakon je opisan od strane Gene Amdahla 1967. godine i definiše maksimalno ubrzanje koje se može postići kroz paralelizaciju izvršavanja dijela programa.
* Prema Amdahl-ovom zakonu, ubrzanje koje se može postići paralelizacijom dijela programa je ograničeno udijelom tog dijela programa koji se može paralelizovati. Konkretno, ako je frakcija programa koji se može paralelizovati jednaka F, tada je maksimalno ubrzanje koje se može postići
* gdje je N broj procesora.
* Na primjer, ako je 80% programa paralelizovano, a 20% je serijski dio koji se ne može paralelizovati, tada je maksimalno ubrzanje koje se može postići ograničeno ovim faktorom. To znači da čak i ako se dodaju beskonačno mnogo procesora, ubrzanje će biti ograničeno serijskim dijelom programa.
* Najjednostavnije se Amdahl-ov zakon može interpretirati kao upozorenje da čak i mali dijelovi programa koji se ne mogu paralelizovati mogu značajno ograničiti ukupno ubrzanje sistema kroz paralelizaciju. Stoga je važno identifikovati i optimizovati te delove kako bi se postiglo maksimalno ubrzanje sistema.

1. Koje su faze izvršenja instrukcije i kojim koracima su opisane kod Von Neumannov modela računara?

* U Von Neumannovom modelu računara, faze izvršenja instrukcije obuhvataju sljedeće korake:
  + Pribavljanje (Fetch): U ovoj fazi, računar pribavlja instrukciju iz memorije. Instrukcija se čita iz memorije i smješta u registar za instrukcije.
  + Dekodiranje (Decode): Nakon što je instrukcija pribavljena, računar je dekodira kako bi razumio šta ta instrukcija zahtijeva. Ovaj korak uključuje tumačenje operacije koja treba da se izvrši i identifikaciju operanada.
  + Izvršenje (Execute): U ovoj fazi, računar izvršava operaciju navedenu u instrukciji. To može uključivati matematičke operacije, logičke operacije, premještanje podataka, itd.
  + Pristup memoriji (Memory Access): Ako je potrebno, računar pristupa memoriji kako bi dohvatio ili sačuvao podatke. Ovaj korak se odnosi na čitanje ili pisanje podataka u memoriju.
  + Povratak rezultata (Write Back): Konačno, rezultat izvršene instrukcije se može vratiti nazad u memoriju ili u registre, u zavisnosti od potreba programa.
* Ovi koraci čine osnovnu sekvencu izvršenja instrukcija u Von Neumannovom modelu računara. Ovaj model se zasniva na ideji da se instrukcije i podaci čuvaju u istoj memoriji i da se izvršavaju korak po korak, što omogućava efikasno izvršavanje programa.

1. Šta je to radni takt procesora? Objasniti kako se izračunava vrijeme potrebno za izvršavanje neke instrukcije za koju je potrebno je 5 ciklusa na računaru R. Ako je njegov radni takt 3 GHz - koje je vrijeme potrebno da se instrukcija izvrši?

* Radni takt procesora predstavlja brzinu kojom procesor izvršava instrukcije, odnosno broj ciklusa sata koje procesor može izvršiti za jednu sekundu. U ovom slučaju, radni takt procesora R iznosi 3 GHz, što znači da procesor može izvršiti 3 milijarde ciklusa sata u jednoj sekundi.
* Da bismo izračunali vrijeme potrebno za izvršavanje instrukcije koja zahtijeva 5 ciklusa na računaru R, koristimo formulu:
  + Vrijeme izvršavanja = Broj ciklusa \* (1 / Radni takt)
* U ovom slučaju, broj ciklusa je 5, a radni takt je 3 GHz (što je ekvivalentno 3 \* 10^9 ciklusa u sekundi).
  + Vrijeme izvršavanja = 5 \* (1 / 3 \* 10^9) = 5 / 3 \* 10^9 sekundi
* Dakle, vrijeme potrebno da se instrukcija izvrši na računaru R sa radnim taktom od 3 GHz iznosi 5 / 3 \* 10^9 sekundi.

1. Šta je to MIPS (Million Instructions per Second), šta IF (Instruction Frequency), a šta CPI (Cycles Per Instruction)? Objasniti na primjeru određivanja CPI prosječne instrukcije i MIPS za neki računar za koji su dati sljedeći parametri. Dužina trajanja taktnog intervala nekog računara je 10 ns, dok je skup njegovih instrukcija i njihova učestalost (dobijena na osnovu neke statistike) data u nastavku:

Load i Store 30,4 1,5

Addi i Subbi 1 0,01

Muli i Divi 3,8 10

Addf i Subbf 9,5 7

Mulf i Divf 6,5 15

Logičke 3,0 1

Branch 20,0 1,5

Compare, Shift 16,8 2

* MIPS (Million Instructions per Second) predstavlja mjernu jedinicu koja označava broj miliona instrukcija koje računar može izvršiti u jednoj sekundi. IF (Instruction Frequency) predstavlja frekvenciju određene instrukcije, odnosno koliko često se ta instrukcija pojavljuje u programu. CPI (Cycles Per Instruction) označava broj ciklusa sata potrebnih za izvršenje jedne instrukcije.
* Da bismo odredili CPI prosječne instrukcije i MIPS za dati računar sa navedenim parametrima, koristimo formule:
  + CPI = Σ(IF \* CPI\_instrukcije) / Σ(IF)
  + MIPS = Radni takt / (CPI \* 10^6)
* U datom primjeru, instrukcije su grupisane sa njihovim IF i CPI vrijednostima:
  + Load i Store: IF = 30,4, CPI = 1,5
  + Addi i Subbi: IF = 1, CPI = 0,01
  + Muli i Divi: IF = 3,8, CPI = 10
  + Addf i Subbf: IF = 9,5, CPI = 7
  + Mulf i Divf: IF = 6,5, CPI = 15
  + Logičke: IF = 3,0, CPI = 1
  + Branch: IF = 20,0, CPI = 1,5
  + Compare, Shift: IF = 16,8, CPI = 2
* Prvo ćemo izračunati CPI prosječne instrukcije:
  + CPI = (30,41,5 + 10,01 + 3,810 + 9,57 + 6,515 + 31 + 201,5 + 16,82) / (30,4 + 1 + 3,8 + 9,5 + 6,5 + 3 + 20 + 16,8) CPI = 8,68
* Zatim ćemo izračunati MIPS:
  + Radni takt = 1 / 10 ns = 100 MHz = 100 \* 10^6 Hz MIPS = 100 \* 10^6 / (8,68 \* 10^6) = 11,52 MIPS
* Dakle, za dati računar, CPI prosječne instrukcije iznosi 8,68, a MIPS iznosi 11,52.

1. Šta je to kapacitet diska (hard diska) i kako se računa? Objasniti na primjeru određivanja/izračuna hard diska koji se sastoji od 8 ploča, svaka površina ploče se sastoji od 200 pisti, svaka pista od 64 sektora, dok je obim sektora 4 bajta. Ovaj hard disk je modifikovan tako što su mu dodate još 2 ploče, ali mu je zato broj pisti po površini smanjen na 128 - koji disk je imao veći kapacitet?

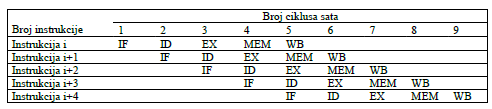
* Kapacitet diska (hard diska) predstavlja ukupnu količinu podataka koja se može smjestiti na disk. Kapacitet diska se računa množenjem broja ploča, broja pista po površini, broja sektora po pisti i obima sektora.
* U datom primjeru, prvobitni hard disk se sastoji od 8 ploča, svaka površina ploče ima 200 pista, svaka pista ima 64 sektora, a obim sektora je 4 bajta.
* Kapacitet prvobitnog hard diska se računa na sljedeći način:
  + Kapacitet = Broj ploča \* Broj pista po površini \* Broj sektora po pisti \* Obim sektora
  + Kapacitet = 8 \* 200 \* 64 \* 4 = 409600 bajta
* Nakon modifikacije, hard disk je dobio dodatne 2 ploče, ali je broj pisti po površini smanjen na 128.
* Kapacitet modifikovanog hard diska se računa na isti način:
  + Kapacitet = (Broj ploča + 2) \* Broj pista po površini \* Broj sektora po pisti \* Obim sektora
  + Kapacitet = 10 \* 128 \* 64 \* 4 = 327680 bajta
* Dakle, modifikovani hard disk ima kapacitet od 327680 bajta, dok je prvobitni hard disk imao kapacitet od 409600 bajta. Prema tome, prvobitni hard disk je imao veći kapacitet u poređenju sa modifikovanim hard diskom.

1. Šta je keš memorija, a šta keš linija odnosno obim keš linije? Objasniti na primjeru izračuna koliko ima linija u kešu čiji je kapacitet 32KB ako je obim linije 128 bajta?

* Keš memorija je vrsta memorije visokih performansi koja se koristi za privremeno skladištenje podataka koji se često koriste kako bi se ubrzao pristup tim podacima. Keš memorija se nalazi između glavne memorije i procesora.
* Keš linija predstavlja osnovnu jedinicu podataka koja se prenosi između glavne memorije i keš memorije. Obim keš linije označava broj bajtova koji se prenose u jednoj keš liniji.
* Da bismo izračunali koliko ima linija u kešu čiji je kapacitet 32KB, a obim keš linije 128 bajta, koristimo formulu:
  + Broj linija = Kapacitet keša / (Obim keš linije)
* U datom primjeru:
  + Kapacitet keša = 32KB = 32 \* 1024 bajta
  + Obim keš linije = 128 bajta
  + Broj linija = 32 \* 1024 / 128 = 256
* Dakle, u kešu čiji je kapacitet 32KB i obim keš linije 128 bajta, ima 256 keš linija.

1. Šta je to protočna struktura i kako se može realizirati? Skicirajte model i opišite!

* Protočna struktura (eng. pipeline) je metoda realizacije centralne procesne jedinice (CPU) kod koje se izvršavanje više instrukcija međusobno vremenski preklapa. Ova tehnika omogućava paralelno izvršavanje različitih faza instrukcija kako bi se povećala efikasnost procesora.
* Model protočne strukture se može skicirati kao niz segmenata ili faza kroz koje prolazi svaka instrukcija.



* Osnovni segmenti u modelu protočne strukture obično uključuju:
  + Pribavljanje instrukcije (IF - Instruction Fetch)
  + Dekodiranje instrukcije (ID - Instruction Decode)
  + Izvršenje instrukcije (EX - Execute)
  + Pristup memoriji (MEM - Memory Access)
  + Upis rezultata nazad u registre (WB - Write Back)
* Svaka faza ili segment predstavlja određenu operaciju koja se izvršava nad instrukcijom. Instrukcije se prenose kroz ove faze u ciklusima sata, pri čemu se svaka faza izvršava u određenom vremenskom intervalu.
* Na primjer, u fazi pribavljanja instrukcije (IF), instrukcija se preuzima iz memorije. Zatim se u fazi dekodiranja instrukcije (ID) instrukcija dekodira i određuje se operacija koja treba biti izvršena. U fazi izvršenja (EX) se obavlja sama operacija instrukcije, dok se u fazi pristupa memoriji (MEM) vrši pristup memoriji ako je potrebno. Na kraju, u fazi upisa rezultata nazad (WB) se rezultat operacije upisuje nazad u registre.
* Realizacija protočne strukture zahtijeva pažljivo planiranje kako bi se osiguralo da nijedan resurs nije zauzet istovremeno za više operacija u istom ciklusu sata. Paralelizam u izvršavanju instrukcija omogućava ubrzanje rada procesora smanjenjem broja ciklusa potrebnih za izvršavanje instrukcija.
* Ukratko, protočna struktura omogućava efikasno izvršavanje instrukcija kroz paralelno preklapanje faza izvršavanja.

1. Šta su hazardi kod protočnih struktura i kako se mogu manifestirati? Objasnite sličnosti i razlike svakog od njih!

* Hazardi kod protočnih struktura predstavljaju situacije koje mogu ometati ili usporiti izvršavanje instrukcija u pipelined procesoru. Postoje tri vrste hazarda kod protočnih struktura:
  + Strukturni hazardi: Nastaju kada se resursi procesora sukobljavaju jer su istovremeno potrebni za izvršavanje više instrukcija. Na primjer, ako jedna instrukcija zahtijeva pristup memoriji dok druga instrukcija koristi ALU, može doći do sukoba resursa. Strukturni hazardi se mogu izbjeći dupliranjem resursa.
  + Hazardi podataka: Nastaju kada jedna instrukcija zavisi od rezultata druge instrukcije koja još nije završena. Na primjer, ako druga instrukcija koristi rezultat prve instrukcije prije nego što je prva instrukcija završena, može doći do hazarda podataka.
  + Hazardi upravljanja: Nastaju kada su promjene u toku izvršavanja instrukcija, kao što su grananja ili skokovi, koje mogu ometati normalan tok izvršavanja instrukcija u pipelined procesoru.
* Sličnosti između ovih hazarda su u tome što svi oni mogu dovesti do usporavanja izvršavanja instrukcija i smanjenja efikasnosti pipelined procesora. Svi ovi hazardi zahtijevaju odgovarajuće tehnike za njihovo rješavanje kako bi se osiguralo nesmetano izvršavanje instrukcija.
* Razlike među ovim hazardima su u uzrocima koji ih izazivaju i načinima na koje se manifestiraju. Strukturni hazardi su uzrokovani sukobljavanjem resursa, dok hazardi podataka proizlaze iz zavisnosti između instrukcija. Hazardi upravljanja se manifestiraju kada se promjene u toku izvršavanja instrukcija ne mogu odmah obraditi u pipelined procesoru.
* Upravljanje hazardima kod protočnih struktura je ključno za održavanje efikasnosti i performansi procesora. Razumijevanje ovih hazarda i primjena odgovarajućih tehnika za njihovo rješavanje omogućava optimalno funkcionisanje pipelined procesora.

1. Šta je to statičko predviđanje grananja? Objasnite na primjeru!

* Statičko predviđanje grananja je tehnika koja se koristi u pipelined procesorima kako bi se predvidjelo ishod grananja prije nego što se stvarno izvrši. Ova tehnika omogućava procesoru da nastavi sa izvršavanjem instrukcija u pipelined strukturi bez čekanja na stvarni rezultat grananja.
* Na primjeru, razmotrimo sljedeći kod:

LW R1, 0(R2)

SUB R1, R1, R2

BEQZ R1, L

OR R4, R5, R6

L: ADD R7, R8, R9

* U ovom kodu, instrukcija BEQZ predstavlja grananje koje zavisi od rezultata prethodne instrukcije SUB. Statičko predviđanje grananja bi se moglo primijeniti tako što se pretpostavi da će se grananje desiti ili neće desiti na osnovu prethodnih informacija.
* Na primjer, ako se pretpostavi da se grananje rijetko dešava, kompajler ili procesor može odlučiti da nastavi sa izvršavanjem instrukcija nakon grananja bez čekanja na stvarni rezultat. U ovom slučaju, instrukcija OR bi se mogla pomjeriti ispred grananja BEQZ kako bi se izbjegao zastoj u pipelined procesoru.
* Statičko predviđanje grananja može se zasnovati na analizi ponašanja programa ili na prethodnim izvršavanjima istog programa. Ova tehnika omogućava procesoru da efikasnije iskoristi pipelined strukturu i smanji zastoje uzrokovane grananjem.
* Ukratko, statičko predviđanje grananja je tehnika koja omogućava procesoru da predvidi ishod grananja na osnovu prethodnih informacija i nastavi sa izvršavanjem instrukcija u pipelined strukturi bez čekanja na stvarni rezultat grananja.

1. Šta su ulazne periferne jednice računara? Nabrojati i objasniti!

* Ulazne periferne jedinice računara su uređaji koji omogućavaju unos podataka ili komunikaciju sa računarom. Ovi uređaji su odgovorni za prijem informacija sa spoljnjeg svijeta i omogućavaju korisnicima da komuniciraju sa računarom.
* Neki primjeri ulaznih perifernih jedinica računara:
  + Tastatura: Tastatura je ulazni uređaj koji omogućava korisnicima unos teksta, komandi i drugih podataka putem pritiskanja tastera. Tastatura je jedan od najosnovnijih ulaznih uređaja računara.
  + Miš: Miš je ulazni uređaj koji omogućava korisnicima da kontrolišu pokret kursora na ekranu računara. Miš se koristi za odabir, pomeranje i manipulaciju objektima na ekranu.
  + Skener: Skener je uređaj koji omogućava digitalizaciju fizičkih dokumenata, slika ili fotografija. Skener pretvara analogne informacije u digitalni format koji se može prikazati ili obrađivati na računaru.
  + Web kamera: Web kamera je ulazni uređaj koji omogućava korisnicima da snimaju video zapise ili fotografije, kao i da vrše video pozive putem interneta. Web kamere se često koriste za video konferencije i online komunikaciju.
  + Mikrofon: Mikrofon je ulazni uređaj koji omogućava korisnicima da unose zvukove ili glasovne komande u računar. Mikrofoni se koriste za snimanje zvuka, vođenje audio razgovora ili za prepoznavanje glasovnih komandi.
  + Čitač kartica: Čitač kartica je uređaj koji omogućava čitanje podataka sa memorijskih kartica, kao što su SD kartice ili SIM kartice. Čitači kartica se često koriste za prenos podataka sa različitih uređaja na računar.
* Ove ulazne periferne jedinice omogućavaju korisnicima da efikasno komuniciraju sa računarom, unose podatke i kontrolišu rad računarskog sistema.

1. Šta su izlazne periferne jedinice računara? Nabrojati i objasniti!

* Izlazne periferne jedinice računara su uređaji koji omogućavaju prikazivanje, reprodukciju ili prenos informacija sa računara ka spoljnjem svijetu. Ovi uređaji omogućavaju korisnicima da vide, čuju ili prenesu rezultate obrade podataka sa računara.
* Neki primjeri izlaznih perifernih jedinica računara:
  + Monitor: Monitor je izlazni uređaj koji prikazuje vizuelne informacije generisane od strane računara. Monitor omogućava korisnicima da vide grafiku, tekst, slike i video zapise koji se prikazuju na ekranu.
  + Štampač: Štampač je uređaj koji omogućava korisnicima da štampaju dokumente, slike ili druge informacije sa računara na papiru. Postoje različite vrste štampača, kao što su laserski, inkjet, ili matrični štampači.
  + Zvučnici: Zvučnici su izlazni uređaji koji omogućavaju reprodukciju zvuka sa računara. Zvučnici omogućavaju korisnicima da čuju zvukove, muziku, ili audio sadržaj koji se reprodukuje na računaru.
  + Projektor: Projektor je uređaj koji omogućava prikazivanje slike ili video zapisa na većoj površini, poput zida ili ekrana. Projektori se često koriste za prezentacije, gledanje filmova ili prikazivanje sadržaja u velikim prostorijama.
  + Ploter: Ploter je uređaj koji omogućava crtanje ili štampanje velikih formata grafika ili crteža. Ploteri se često koriste u dizajnu, arhitekturi ili inženjeringu za štampanje tehničkih crteža.
  + Modem: Modem je uređaj koji omogućava povezivanje računara sa internetom ili drugim mrežama putem telefonske linije ili kabla. Modemi omogućavaju prenos podataka između računara i drugih uređaja putem mreže.
* Ove izlazne periferne jedinice omogućavaju korisnicima da efikasno prikažu, reprodukuju ili prenesu informacije sa računara ka spoljnjem svijetu, što doprinosi širokoj upotrebi računara u različitim oblastima.

1. Klasifikacija arhitektura računara i raĉunarskih sistema prema funkcionalnosti, raspoloživosti i pouzdanosti rada sistema - objasniti!

* Arhitekture računara i računarskih sistema se mogu klasifikovati prema funkcionalnosti, raspoloživosti i pouzdanosti rada sistema na nekoliko načina:
  + Prema funkcionalnosti:
    - Von Neumann arhitektura: Osnovna arhitektura računara koja se sastoji od jedinstvene memorije za podatke i instrukcije, centralne procesorske jedinice (CPU), ulazno/izlaznih jedinica i sabirnice za komunikaciju između komponenti. Ova arhitektura se koristi u većini modernih računara.
    - Harvard arhitektura: Ova arhitektura razdvaja memoriju za podatke i instrukcije, što omogućava paralelno izvršavanje instrukcija i podataka. Koristi se u specijalizovanim sistemima kao što su mikrokontroleri.
  + Prema raspoloživosti:
    - Single-processor arhitektura: Sistem sa jednim procesorom koji obavlja sve zadatke. Ova arhitektura je jednostavna, ali može biti ograničena u pogledu performansi i skalabilnosti.
    - Multi-processor arhitektura: Sistem sa više procesora koji mogu paralelno izvršavati zadatke. Ova arhitektura omogućava veću brzinu izvršavanja i bolju skalabilnost, ali zahtijeva efikasno upravljanje resursima.
  + Prema pouzdanosti rada sistema:
    - Single-fault tolerant arhitektura: Sistem koji može nastaviti sa radom nakon pojave jedne greške ili kvara. Ova arhitektura obično koristi redundanciju komponenti ili mehanizme za detekciju i oporavak grešaka.
    - Fault-tolerant arhitektura: Sistem koji može nastaviti sa radom čak i nakon više grešaka ili kvarova. Ova arhitektura obično koristi višestruku redundanciju, distribuirane sisteme ili napredne mehanizme za detekciju i oporavak grešaka.
* Klasifikacija arhitektura računara prema funkcionalnosti, raspoloživosti i pouzdanosti rada sistema omogućava inženjerima da odaberu odgovarajuću arhitekturu u skladu sa zahtjevima i potrebama konkretnog sistema ili aplikacije.

1. Jedna linija programa napisanog u asemblerskom jeziku ima ovakav oblik: LABELA: MNEMONIK OPERAND; Objasni značenje svakog od pobrojanih elemenata: LABELA, MNEMONIK, OPERAND!

* Objašnjenje značenja svakog od pobrojanih elemenata:
  + LABELA: Oznaka (labela) je simboličko ime koje se koristi za označavanje određene tačke ili lokacije u programu. Oznake se obično koriste za označavanje početka ili kraja petlji, uslovnih naredbi ili podprograma. Oznake se koriste kao referenca u programu za skakanje na određenu tačku ili za adresiranje određenih instrukcija.
  + MNEMONIK: Mnemonik je skraćeni simbolički naziv instrukcije koja se izvršava u računaru. Svaka instrukcija u asemblerskom jeziku ima svoj odgovarajući mnemonik koji se koristi za predstavljanje te instrukcije. Na primjer, "MOV" se koristi za instrukciju kopiranja podataka, "ADD" za instrukciju sabiranja, "JMP" za instrukciju skoka, itd.
  + OPERAND: Operand je dodatni dio instrukcije koji specificira podatke ili adresu na kojima se instrukcija izvršava. Operand može biti registar, konstanta, adresa memorije ili izračunata vrijednost. Na primjer, u instrukciji "MOV AX, 5", "AX" je registarski operand, a "5" je konstantni operand.
* Ovaj format linije programa u asemblerskom jeziku omogućava programerima da jasno definišu instrukcije, oznake i operande kako bi napisali efikasan i precizan asemblerski kod.

1. U sastavu same riječi i nstrukcije nalazi se i informacija o načinu adresiranja. Ta informacija je obično u okviru operacionog dijela instrukcije, tako da se onda instrukcija praktično sastoji iz tri dijela: operacioni kod, polje načina adresiranja i dio operanda.

* U asemblerskom jeziku, instrukcija se sastoji od operacionog koda, polja načina adresiranja i dijela operanda. Evo detaljnijeg obrazloženja svakog dijela instrukcije:
  + Operacioni kod (mnemonik): Operacioni kod predstavlja skraćeni simbolički naziv instrukcije koja se izvršava. Ovaj deo instrukcije određuje koja se operacija izvršava, kao što su sabiranje, oduzimanje, kopiranje podataka, skakanje na drugu lokaciju, itd. Na primer, "MOV" se koristi za kopiranje podataka, "ADD" za sabiranje, "JMP" za skok, itd. Operacioni kod je esencijalni deo instrukcije koji definiše šta tačno treba da se uradi.
  + Polje načina adresiranja: Polje načina adresiranja u instrukciji sadrži informaciju o tome kako se operandi instrukcije adresiraju ili pristupaju. Način adresiranja može biti direktno, indirektno, registarsko, pomjerajno, itd. Ova informacija pomaže procesoru da odredi gdje se tačno nalaze operandi koje treba da obradi. Na primer, način adresiranja može odrediti da li se operand nalazi direktno u instrukciji, u registru, ili na određenoj adresi u memoriji.
  + Operand: Operand je dio instrukcije koji specificira podatke ili adresu na kojima se instrukcija izvršava. Operand može biti registar, konstanta, adresa memorije ili izračunata vrednost. Ovaj deo instrukcije određuje nad kojim podacima ili adresama se vrši operacija definisana operacionim kodom. Na primer, u instrukciji "MOV AX, 5", "AX" je registarski operand, a "5" je konstantni operand.
* Kombinovanjem ovih dijelova, instrukcija postaje kompletna i precizna, omogućavajući procesoru da izvrši željenu operaciju na odgovarajućim operandima na odgovarajući način. Struktura instrukcije u asemblerskom jeziku omogućava programerima da precizno definišu akcije koje računar treba da izvrši, kao i način na koji treba da pristupi podacima radi efikasnog izvršavanja programa.

1. Tipovi i veličine operanada u instrukciji. Objasniti!

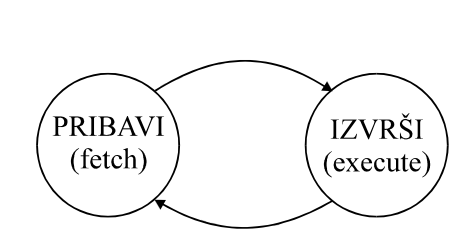
* Tipovi i veličine operanada u instrukcijama igraju ključnu ulogu u računarskim arhitekturama. Ovi elementi određuju kako će se podaci obrađivati i interpretirati unutar računara. Evo nekoliko osnovnih tipova operanada i njihovih veličina:
  + Logički tip (1 bit): Ovaj tip operanda može imati samo dvije vrijednosti - 0 ili 1. Koristi se za logičke operacije poput "I", "ILI", "NEGACIJA".
  + Karakter (1 bajt): Ovaj tip operanda obično predstavlja jedan znak i često se koristi za prikazivanje teksta. Često se koristi ASCII kodiranje za predstavljanje karaktera.
  + Poluriječ (2 bajta): Ovaj tip operanda zauzima 2 bajta memorije i često se koristi za Unicode znakove koji zahtijevaju veći opseg karaktera od ASCII-a.
  + Riječ (4 bajta): Ovaj tip operanda obično predstavlja cjelobrojne vrijednosti i kodira se u 2KK formatu. Koristi se za pohranu većih brojeva.
  + Riječ kao broj sa pomičnim zarezom - jednostruka preciznost: Ovaj tip operanda se koristi za pohranu brojeva sa pomičnim zarezom u jednostrukoj preciznosti.
  + Dupla riječ (8 bajta): Ovaj tip operanda se koristi za pohranu brojeva sa pomičnim zarezom u dvostrukoj preciznosti prema IEEE 754 standardu. Ovo omogućava veću preciznost pri radu s decimalnim brojevima.
* Različiti tipovi operanada omogućavaju računarima da efikasno obrađuju različite vrste podataka i obavljaju različite vrste operacija. Pravilno definisanje tipova i veličina operanada u instrukcijama ključno je za pravilno funkcionisanje računarskih sistema i optimizaciju performansi.

1. Izaberi i opiši ukratko bar četiri načina adresiranja iz liste: implicitno adresiranje, neposredno adresiranje, direktno adresiranje, indirektno adresiranje. indeksno adresiranje, relativno adresiranje.

* Implicitno adresiranje: Kod implicitnog adresiranja, operand se implicitno odnosi na određeni registar ili memorijsku lokaciju. Nema potrebe za eksplicitnim navođenjem adrese operandu, jer se operand podrazumijeva iz konteksta instrukcije.
* Neposredno adresiranje: U neposrednom adresiranju, operand direktno sadrži vrijednost podatka, a ne adresu gdje se podatak nalazi. To znači da se vrijednost operandu direktno proslijeđuje u instrukciji.
* Direktno adresiranje: Kod direktnog adresiranja, operand sadrži adresu memorijske lokacije gdje se željeni podatak nalazi. Pristup podacima se vrši direktno preko te adrese.
* Indirektno adresiranje: U indirektnom adresiranju, operand sadrži adresu memorijske lokacije koja dalje sadrži adresu gdje se nalazi željeni podatak. Pristup podacima se vrši preko posredne adrese.
* Indeksno adresiranje: Indeksno adresiranje uključuje korištenje registra indeksa koji se množi sa određenim faktorom i dodaje na bazu adrese kako bi se dobila konačna adresa operandu.
* Relativno adresiranje: Relativno adresiranje koristi se za adresiranje memorijskih lokacija relativno u odnosu na trenutnu poziciju izvršavanja programa. Adresa se izračunava dodavanjem određenog pomaka (displacement) na trenutnu adresu, što omogućava fleksibilnost u programiranju i olakšava relokaciju programa.

1. Pri izvršavanju instrukcije postoje dvije faze, odnosno dva ciklusa. Navedite koja su to dva ciklusa i šta se dešava u svakom od njih!

* Pri izvršavanju instrukcije, postoje dvije faze ili ciklusa: PRIBAVI (fetch) i IZVRŠI (execute). Evo šta se dešava u svakom od ovih ciklusa:
* Faza PRIBAVI (fetch):
  + Iz memorije se pribavlja (čita) sljedeća instrukcija i prenosi se u instrukcijski registar (IR).
  + Adresa instrukcije se nalazi u registru Programskog brojača (PC).
  + Sadržaj registra PC se povećava za jedan, određujući instrukciju koja slijedi nakon trenutne instrukcije.
  + Dekodira se 8-bitni kôd operacije, aktivirajući odgovarajuću izlaznu liniju dekodera za tu instrukciju.
* Faza IZVRŠI (execute):
  + Pobuđuju se operacije koje se izvršavaju prema dekodiranoj instrukciji (npr. prenos podataka između memorije i registara, aritmetičko-logičke operacije, promjena vrijednosti Programskog brojača u slučaju grananja).
  + Izvršavaju se operacije koje su određene dekodiranom instrukcijom.
  + Nakon završetka izvršavanja instrukcije, upravljačka jedinica se vraća na fazu PRIBAVI kako bi se pribavila i izvršila sljedeća instrukcija.



* Ove dvije faze se izmjenjuju tokom izvršavanja programa, omogućavajući računaru da redom izvršava instrukcije i obavlja operacije definisane tim instrukcijama.

1. Prema funkciji i namjeni skup instrukcija 8086 mikroprocesora se može podijeliti u grupe: instrukcije za prenos podataka, aritmetičke instrukcije, instrukcije za manipulisanje bitima ili logičke instrukcije, instrukcije za rad sa nizovima podataka, instrukcije za prenos upravljanja u programu.

Objasni karakteristike bar četiri grupe od pobrojanih.

* Skup instrukcija 8086 mikroprocesora može se podijeliti u sljedeće grupe prema funkciji i namjeni:
  + Instrukcije za prenos podataka: Ova grupa instrukcija omogućava prenos podataka između registara, memorije i drugih lokacija unutar procesora. To uključuje instrukcije za učitavanje podataka iz memorije u registre, čuvanje rezultata izračunavanja nazad u memoriju, kao i razmjenu podataka između registara. Ove instrukcije su ključne za manipulaciju podacima i izvršavanje aritmetičkih operacija.
  + Aritmetičke instrukcije: Ova grupa instrukcija obuhvata operacije kao što su sabiranje, oduzimanje, množenje i dijeljenje brojeva. Aritmetičke instrukcije omogućavaju procesoru da izvršava matematičke operacije nad podacima, što je bitno za izračunavanje rezultata programa.
  + Instrukcije za manipulisanje bitima ili logičke instrukcije: Ove instrukcije se koriste za manipulaciju pojedinačnim bitovima u registrima ili memoriji. To uključuje operacije kao što su logičko "I", logičko "ILI", negacija bitova, pomjeranje bitova, postavljanje i brisanje određenih bitova. Ove instrukcije su korisne za rad sa flag bitovima, provjeru uslova i implementaciju logičkih operacija.
  + Instrukcije za rad sa nizovima podataka: Ova grupa instrukcija omogućava procesoru da efikasno radi sa nizovima podataka. To uključuje instrukcije za automatsko inkrementiranje/dekrementiranje adresa, pretragu nizova, kopiranje podataka između nizova, kao i druge operacije koje olakšavaju rad sa strukturiranim podacima.
  + Instrukcije za prenos upravljanja u programu: Ove instrukcije se koriste za kontrolu toka izvršavanja programa, uključujući grananje, skakanje na određene adrese, pozive podprograma i povratak iz podprograma. Instrukcije za prenos upravljanja omogućavaju programeru da kontroliše redoslijed izvršavanja instrukcija i implementira uslove grananja i petlje.
* Kombinacija ovih grupa instrukcija omogućava 8086 mikroprocesoru da izvršava različite zadatke i operacije u skladu sa zahtjevima programa koji se izvršava.

1. Objasniti sličnosti i razlike računarskih arhitektura

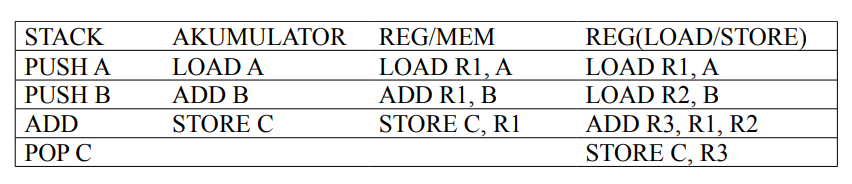
a) stack-orijentisane (0-adresne),

b) akumulatorske,

c) registarske - one sa registrima opšte namjene,

d) memorijske.

* Stack-orijentisane (0-adresne) arhitekture:
  + Sličnosti: U ovim arhitekturama operandi se implicitno nalaze na vrhu stack-a, a operacije se izvršavaju nad vrhom stack-a.
  + Razlike: Ove arhitekture koriste stack kao osnovnu strukturu za čuvanje operanada i rezultata operacija. Operandi se automatski smještaju na vrh stack-a, a operacije se izvršavaju nad tim operandima.
* Akumulatorske arhitekture:
  + Sličnosti: U akumulatorskim arhitekturama, jedan operand je implicitno u akumulatoru, dok se drugi operandi eksplicitno navode u instrukcijama.
  + Razlike: Ove arhitekture koriste akumulator kao glavni registar za izvršavanje aritmetičkih operacija. Rezultati operacija se često čuvaju u akumulatoru.
* Memorijske arhitekture:
  + Sličnosti: U memorijskim arhitekturama, svi operandi se čuvaju u memoriji, a operacije se izvršavaju nad podacima direktno iz memorije.
  + Razlike: Ove arhitekture zahtijevaju češće pristupe memoriji radi čitanja i pisanja podataka, što može rezultirati sporijim izvršavanjem programa u poređenju sa arhitekturama koje koriste registre za čuvanje podataka.
* Registarske arhitekture (sa registrima opšte namjene):
  + Sličnosti: U ovim arhitekturama, operandi se eksplicitno navode u instrukcijama, a operacije se izvršavaju nad sadržajem registara opšte namjene.
  + Razlike: Registarske arhitekture omogućavaju brz pristup podacima i efikasno izvršavanje operacija zahvaljujući upotrebi registara opšte namjene za čuvanje podataka i privremenih rezultata.



* Ukratko, sličnosti između ovih arhitektura uključuju način na koji se operandi koriste u operacijama, dok se razlike odnose na organizaciju registara, pristup podacima i efikasnost izvršavanja operacija. Svaka arhitektura ima svoje prednosti i mane u zavisnosti od specifičnih zahtjeva i primjene.

1. Šta je to kompajler i kakva je njegova uloga za računarsku arhitekturu? Objasniti!

* Kompajler je programska komponenta koja prevodi izvorni kod napisan u programskom jeziku visokog nivoa u mašinski jezik koji može direktno izvršavati računar. Uloga kompajlera je ključna za računarsku arhitekturu jer omogućava izvršavanje programa na hardveru.
* Neke od ključnih uloga kompajlera za računarsku arhitekturu:
  + Prevod izvornog koda: Kompajler prevodi čitljiv i razumljiv izvorni kod napisan na programskom jeziku visokog nivoa (kao što su C, Java, Python) u niz instrukcija mašinskog jezika koje procesor može direktno izvršavati. Ovaj proces se naziva kompilacija.
  + Optimizacija koda: Kompajler ima ulogu da optimizuje generisani mašinski kod kako bi program bio efikasniji u izvršavanju. To uključuje optimizaciju brzine izvršavanja, smanjenje potrošnje memorije i poboljšanje performansi programa.
  + Generisanje izvršnog koda: Kompajler generiše izvršni kod koji je specifičan za određenu arhitekturu računara. To znači da će izvršni kod generisan za jednu arhitekturu biti različit od izvršnog koda generisanog za drugu arhitekturu.
  + Detekcija grešaka: Kompajler detektuje sintaksne i semantičke greške u izvornom kodu tokom faze kompilacije. Ovo pomaže programerima da isprave greške pre nego što program bude izvršen na računaru.
  + Podrška za različite arhitekture: Kompajler mora biti prilagođen specifičnostima određene računarske arhitekture kako bi generisao ispravan izvršni kod. To uključuje podršku za registre, adresiranje memorije, instrukcije i druge karakteristike arhitekture.
* Ukratko, kompajler je ključna komponenta za računarsku arhitekturu jer omogućava programerima da pišu programe na visokom nivou apstrakcije, a zatim ih izvršavaju na hardveru kroz generisanje odgovarajućeg mašinskog koda prilagođenog specifičnoj arhitekturi računara.

1. Kako kompajler utiče na arhitekturu? Objasniti!

* Kompajler ima značajan uticaj na arhitekturu računara na nekoliko načina:
  + Optimizacija koda: Kompajler može optimizovati izvorni kod kako bi poboljšao performanse programa. Ovo može uključivati optimizaciju brzine izvršavanja, smanjenje potrošnje memorije i poboljšanje efikasnosti programa. Kroz optimizaciju koda, kompajler može uticati na to kako se program izvršava na hardveru, što može imati veliki uticaj na performanse sistema.
  + Generisanje mašinskog koda: Kompajler generiše mašinski kod koji je specifičan za određenu arhitekturu računara. Na taj način, kompajler direktno utiče na to kako će program biti izvršen na hardveru. Generisani mašinski kod mora biti prilagođen karakteristikama arhitekture, kao što su registri, instrukcije i način adresiranja memorije.
  + Podrška za specifične karakteristike arhitekture: Kompajler mora biti prilagođen specifičnostima određene arhitekture računara kako bi generisao ispravan izvršni kod. To uključuje podršku za registre, adresiranje memorije, instrukcije i druge karakteristike arhitekture. Kroz podršku za ove karakteristike, kompajler omogućava efikasno izvršavanje programa na određenoj arhitekturi.
  + Detekcija grešaka: Kompajler detektuje sintaksne i semantičke greške u izvornom kodu tokom faze kompilacije. Ovo pomaže programerima da isprave greške pre nego što program bude izvršen na računaru. Na taj način, kompajler može uticati na kvalitet i pouzdanost programa koji se izvršava na određenoj arhitekturi.
* Ukratko, kompajler ima značajan uticaj na arhitekturu računara kroz optimizaciju koda, generisanje mašinskog koda prilagođenog arhitekturi, podršku za specifične karakteristike arhitekture i detekciju grešaka. Kroz ove procese, kompajler direktno utiče na performanse, efikasnost i pouzdanost programa koji se izvršava na određenoj arhitekturi računara.

1. Podrška hardvera u povećanju paralelizma na nivou instrukcija - objasniti!

* Podrška hardvera u povećanju paralelizma na nivou instrukcija odnosi se na tehnike i mehanizme koje se implementiraju u hardveru računara kako bi se omogućilo efikasno izvršavanje instrukcija paralelno. Ovo se često naziva Instruction-Level Parallelism (ILP) i ima za cilj povećanje performansi računarskog sistema kroz paralelno izvršavanje instrukcija.
* Nekoliko načina na koje hardver podržava povećanje paralelizma na nivou instrukcija:
  + Superskalarno izvršavanje: Superskalarni procesori imaju više izvršnih jedinica koje mogu istovremeno izvršavati više instrukcija. Ovo omogućava paralelno izvršavanje nezavisnih instrukcija i povećava brzinu izvršavanja programa.
  + Pipelining: Pipelining je tehnika koja razdvaja izvršavanje instrukcija na nekoliko faza kako bi se omogućilo paralelno izvršavanje više instrukcija u isto vreme. Svaka faza pipelina izvršava dio instrukcije, što omogućava da se više instrukcija izvršava istovremeno.
  + Prediktivno izvršavanje: Prediktivno izvršavanje je tehnika koja se koristi za predviđanje uslovnih skokova u programu kako bi se izbjegle kašnjenja u izvršavanju instrukcija. Ovo omogućava da se instrukcije izvršavaju paralelno i povećava efikasnost izvršavanja programa.
  + Reorder Buffer: Reorder Buffer je mehanizam koji se koristi za preuređivanje redosleda instrukcija radi povećanja paralelizma. Ovaj mehanizam omogućava da se instrukcije izvršavaju u optimalnom redoslijedu bez čekanja na završetak nekih prethodnih instrukcija.
* Kroz ove tehnike i mehanizme, hardver podržava povećanje paralelizma na nivou instrukcija, što rezultira bržim i efikasnijim izvršavanjem programa. Ova podrška omogućava računarskim sistemima da iskoriste više resursa procesora istovremeno i postignu veće performanse u obradi podataka.

1. Objasnite na primjeru dinamičko raspoređivanje instrukcija sa semaforom kod protočne strukture!

* Dinamičko raspoređivanje instrukcija sa semaforom je tehnika koja se koristi u protočnim strukturama procesora kako bi se povećao paralelizam između instrukcija. Semafor se koristi za upravljanje redoslijedom izvršavanja instrukcija i riješavanje zavisnosti između njih.
* Uzmimo za primjer jednostavnu protočnu strukturu procesora sa semaforom koji podržava paralelno izvršavanje instrukcija. U ovom primjeru, imamo dva množača, jedan sabirač, jedan djelitelj i jednu cjelobrojnu jedinicu za sve pristupe memoriji, grananja i cjelobrojne operacije.
  + Svaka instrukcija prolazi kroz semafor: Kada instrukcija stigne na izvršavanje, prolazi kroz semafor gde se zapisuju zavisnosti podataka. Semafor određuje kada instrukcija može pročitati svoje operande i započeti izvršavanje.
  + Upravljanje zavisnostima: Semafor nadgleda zavisnosti između instrukcija i osigurava da se instrukcije izvršavaju u odgovarajućem redosledu kako bi se izbegle konflikte i greške.
  + Paralelno izvršavanje: Kroz semafor, instrukcije koje nemaju zavisnosti mogu se izvršavati paralelno. Na primer, dok se jedna instrukcija množenja izvršava u jednom delu procesora, druga instrukcija sabiranja može se izvršavati istovremeno u drugom delu procesora.
  + Efikasno upravljanje resursima: Semafor omogućava efikasno upravljanje resursima procesora tako što omogućava paralelno izvršavanje instrukcija bez konflikata i čekanja.
* Kroz dinamičko raspoređivanje instrukcija sa semaforom, protočna struktura procesora može efikasno iskoristiti paralelizam na nivou instrukcija i povećati brzinu izvršavanja programa. Ova tehnika omogućava da se više instrukcija izvršava istovremeno, što dovodi do poboljšanja performansi procesora.

1. Kako se izračunava vrijeme izvršavanja jedne instrukcije kod protočne strukture u idealnim uslovima? Objasniti!

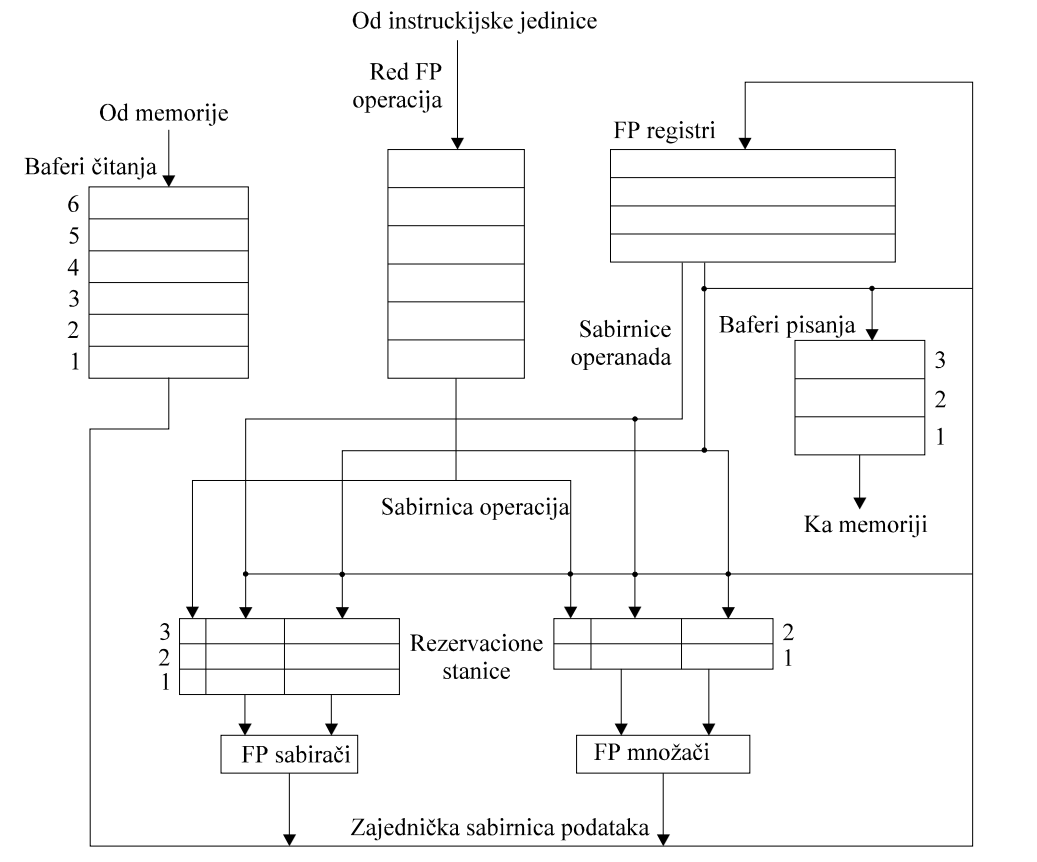
* Vrijeme izvršavanja jedne instrukcije kod protočne strukture procesora u idealnim uslovima može se izračunati kroz koncept poznat kao "CPI" (Cycles Per Instruction) ili broj ciklusa po instrukciji. U idealnim uslovima, kada nema zastoja ili prekida u izvršavanju instrukcija, vrijeme izvršavanja jedne instrukcije može se izračunati kao zbir vremena provedenog u svakoj fazi protočne strukture.
* Koraci za izračunavanje vremena izvršavanja jedne instrukcije u idealnim uslovima kod protočne strukture:
  + Identifikacija faza protočne strukture: Prvo je potrebno identifikovati faze kroz koje instrukcija prolazi u procesu izvršavanja. Tipične faze uključuju preuzimanje instrukcije (IF), dekodiranje instrukcije (ID), izvršavanje instrukcije (EX), pristup memoriji (MEM) i zapisivanje rezultata (WB).
  + Određivanje broja ciklusa po fazi: Za svaku fazu, određuje se broj ciklusa potrebnih za izvršavanje te faze. Na primer, IF faza može zahtevati 1 ciklus, dok EX faza može zahtevati 2 ciklusa.
  + Izračunavanje CPI: Zatim se sumiraju ciklusi potrebni za svaku fazu kako bi se dobio ukupan broj ciklusa potrebnih za izvršavanje jedne instrukcije. Ovo je CPI.
  + Vrijeme izvršavanja jedne instrukcije: Konačno, vrijeme izvršavanja jedne instrukcije može se izračunati kao proizvod CPI i vremena jednog ciklusa procesora.
* U idealnim uslovima, kada nema zastoja ili prekida u izvršavanju instrukcija, ovaj proračun daje procjenu vremena potrebnog za izvršavanje jedne instrukcije u protočnoj strukturi procesora. Međutim, u stvarnim uslovima, faktori poput zastoja zbog zavisnosti podataka ili skokova mogu uticati na stvarno vrijeme izvršavanja instrukcija.

1. Šta je to Tomasulov algoritam i kako se koristi? Objasnite na primjeru!

* Tomasulov algoritam je tehnika dinamičkog raspoređivanja instrukcija koja se koristi u super-skalarnim procesorima za efikasno upravljanje resursima i izbjegavanje zastoja usled zavisnosti podataka. Ovaj algoritam omogućava paralelno izvršavanje instrukcija koje nemaju zavisnosti, čime se povećava efikasnost procesora.
* Evo kako Tomasulov algoritam funkcioniše na primjeru jednostavnog super-skalarnog procesora sa jednom FP jedinicom i jednom jedinicom za pristup memoriji:
  + Rezervacione stanice: U Tomasulovom algoritmu, instrukcije se ne izvršavaju direktno, već se prvo smeštaju u rezervacione stanice. Svaka rezervaciona stanica čuva instrukciju, operande i informacije o zavisnostima.
  + Izvršavanje instrukcija: Kada su svi operandi dostupni, instrukcija se šalje na izvršavanje u odgovarajuću jedinicu (npr. FP jedinicu). Rezultati izvršavanja se smeštaju u bafer za pisanje.
  + Otklanjanje zavisnosti: Ukoliko se pojavi zavisnost između instrukcija, Tomasulov algoritam koristi mehanizam prenosa rezultata (Common Data Bus - CDB) kako bi se otklonile zavisnosti. Na taj način, instrukcije koje čekaju na određene operande mogu nastaviti sa izvršavanjem čim operandi postanu dostupni.
  + Paralelno izvršavanje: Zahvaljujući rezervacionim stanicama i mehanizmu otklanjanja zavisnosti, Tomasulov algoritam omogućava paralelno izvršavanje instrukcija koje nemaju zavisnosti, čime se povećava efikasnost procesora.
* Na primjeru, ako imamo instrukcije za sabiranje, množenje i pristup memoriji, Tomasulov algoritam omogućava da se ove instrukcije izvršavaju paralelno ukoliko nemaju zavisnosti. Na taj način, procesor može efikasno iskoristiti resurse i ubrzati izvršavanje programa.
* Tomasulov algoritam je ključna tehnika u super-skalarnim procesorima za postizanje visokog stepena paralelizma i efikasnog upravljanja resursima.

1. Skicirajte i objasnite osnovnu struktura FP jedinice ogledne arhitekture sa Tomasulo-vim algoritmom!

* Slika daje osnovnu strukturu FP-jedinice ogledne arhitekture bazirane na Tomasulovom algoritmu.
* Nije data nijedna tabela za upravljanje izvršenjem.
* Rezervacione stanice sadrže instrukcije koje su već bile pokrenute i čekaju na izvršenje na FJ-i, operande za tu instrukciju (ako su već izračunati) ili izvor operanada, kao i podatke neophodne za upravljanje u toku izvršenja na FJ-i.
* Ulazni (load) i izlazni (store) baferi sadrže podatke ili adrese koji stižu od, ili odlaze ka memoriji. FP registri su povezani dvjema sabirnicama sa FJ-ama i jednom sa izlaznim (store) baferima.
* Svi rezultati iz FJ-a i memorije se šalju na zajedničku sabirnicu podataka, kojom su vezani za sva odredišta osim ulaznih (load) bafera.
* Svi baferi i rezervacione stanice imaju tag-polja koja se koriste za kontrolu hazarda



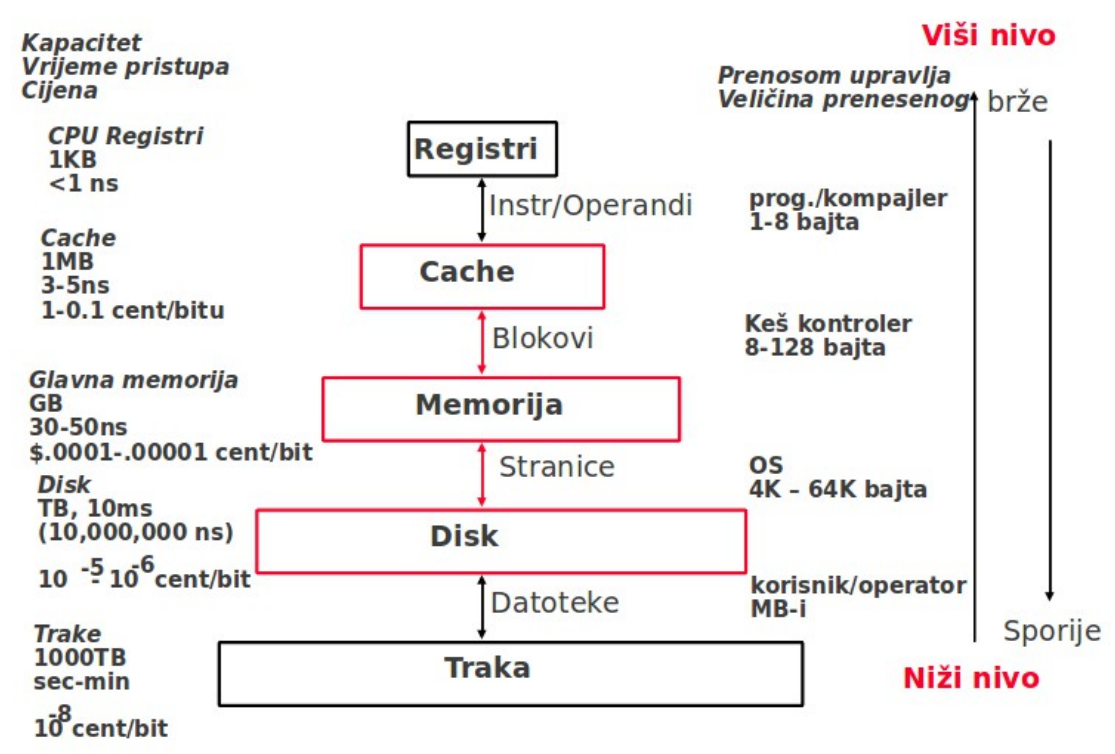
* Kada se pokrenu, FP operacije se šalju iz instrukcijske jedinice u red čekanja.
* U rezervacionim stanicama su operacije i operandi, kao i informacije potrebne za otkrivanje i rješavanje hazarda.
* Baferi čitanja sadrže rezultate tekućih čitanja iz memorije, a baferi pisanja sadrže adrese tekućih upisa u memoriju koji čekaju na svoje operande.
* Svi rezultati iz FP jedinica ili jedinice čitanja iz memorije se šalju na zajednočku sabirnicu podataka (CDB), koja vodi do FP registara, kao i rezervacionih stanica i bafera upisa u memoriju.
* FP sabirači odrađuju sabiranje i oduzimanje, dok FP množači rade množenje i dijeljenje.
* Pošto se operandi prenose drugačije nego kod semafora, postoje samo tri koraka/faze kroz koje prolazi instrukcija:
  + Pokretanje - uzima instrukciju iz reda FP-instrukcija. Ako je uzeta FP-instrukcija, pokreće se ako postoji slobodna/prazna rezervaciona stanica, i, ako su operandi u registrima, šalju se u tu rezervacionu stanicu. Ako je uzeta instrukcija čitanje (load) ili pisanja (store), pokreće se ako postoji prazan odgovarajući bafer. Ako ne postoji prazna RS ili prazan bafer, postoji strukturni hazard i instrukcija se zadržava dok se RS ili bafer ne oslobodi. Ovaj korak vrši i reimenovanje registara.
  + Izvršenje - Ako jedan ili više operanada još nisu raspoloživi, prati se CDB dok se čeka da se izračuna sadržaj registra. Kada je operand raspoloživ, šalje se u odgovarajuću RS-u. Kada su oba raspoloživa izvrši se operacija. Ovaj korak provjerava RAW hazarde.
  + Pisanje rezultata - kada je rezultat raspoloživ, piše se na CDB, a odatle u registre i FJ-e koje taj rezultat očekuju

1. Kako se realizira prenos podataka između nivoa memorijske hijerarhije? Objasniti!

* Prenos podataka između nivoa memorijske hijerarhije se realizira kroz različite mehanizme kako bi se postigla efikasnost i brzina pristupa memoriji. Postoje različiti pristupi za prenos podataka između keš memorije, radne memorije i drugih nivoa memorije.
  + Pristup riječ-po-riječ: Jedan od načina je sukcesivni prenos riječi od glavne memorije prema kešu i obratno. Ovaj pristup koristi sabirnice iste širine za prenos riječi.
  + Proširena sabirnica: Drugi pristup koristi sabirnicu od keša ka procesoru širine jedne riječi, ali za prenose između keša i glavne memorije koristi proširenu sabirnicu, što omogućava prenos bloka u jednom memorijskom ciklusu.
  + Preplitanje memorijskih modula: Treći pristup zadržava sabirnice širine jedne riječi, ali se preplitanjem memorijskih modula "sakrije" kašnjenje prilikom prenosa bloka. Ovo omogućava slično vrijeme prenosa bloka kao kod proširene sabirnice, uz zadržavanje jednostavnog ožičenja.
* Uz ove pristupe, važno je uzeti u obzir i lokalnost prilikom pristupa memoriji, što može pomoći u smanjenju učestanosti promašaja i poboljšanju performansi memorijske hijerarhije.

1. Opišite nivoe u memorijskoj hijerarhiji, sa cijenama, kapacitetima inicijatorima prenosa i veličinama prenesenih blokova i skicom prikažite njihov odnos!

* Memorijska hijerarhija obično uključuje nekoliko nivoa memorije, svaki sa svojim karakteristikama. Evo opisa tipičnih nivoa u memorijskoj hijerarhiji:
  + Registri: Registri su najbrži nivo memorije koji se nalazi direktno u procesoru. Imaju najmanju latenciju i najveću brzinu pristupa. Kapacitet registara je obično mali, ali se koriste za čuvanje trenutnih podataka i instrukcija koje se trenutno izvršavaju.
  + Keš memorija: Keš memorija je brža od glavne memorije i koristi se za čuvanje kopija nedavno korištenih podataka iz glavne memorije. Postoje različiti nivoi keš memorije (L1, L2, L3) sa različitim kapacitetima i brzinama pristupa. L1 keš je najbliži procesoru i obično ima najmanji kapacitet, dok L3 keš može imati veći kapacitet ali sporiji pristup.
  + Glavna memorija (RAM): Glavna memorija je većeg kapaciteta od keš memorije, ali ima veću latenciju pristupa. Koristi se za čuvanje podataka i instrukcija koje se trenutno ne koriste u kešu.
  + Sekundarna memorija: Sekundarna memorija, kao što su tvrdi diskovi ili SSD-ovi, ima najveći kapacitet ali najveću latenciju pristupa. Koristi se za dugoročno čuvanje podataka koji nisu trenutno potrebni u radnoj memoriji.
* Cijene, kapaciteti, inicijatori prenosa i veličine prenesenih blokova variraju između ovih nivoa memorije. Registri imaju najmanju cijenu po bitu, ali i najmanji kapacitet. Keš memorija ima veću cijenu po bitu u odnosu na registre, ali i veći kapacitet. Glavna memorija ima nižu cijenu po bitu od keš memorije, ali i veću latenciju pristupa. Sekundarna memorija ima najnižu cijenu po bitu, ali i najveću latenciju pristupa.
* Skica prikazuje ovaj odnos sa registrima na vrhu (najbrži ali najmanji kapacitet) i sekundarnom memorijom na dnu (najveći kapacitet ali najveća latencija pristupa), sa keš memorijom i glavnom memorijom između njih po brzini i kapacitetu.

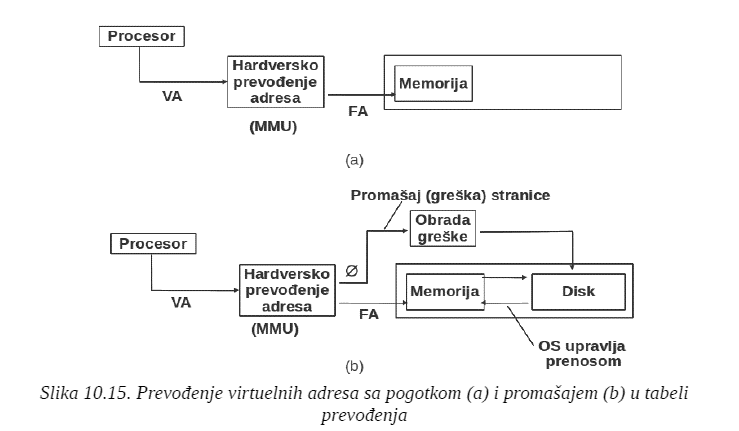


1. Skiciraj i opiši korake prilikom izvršenja instrukcije sa Branch-Target-Buffer-om!

* Koraci prilikom izvršenja instrukcije sa Branch-Target-Buffer-om obuhvataju proces predviđanja grananja kako bi se smanjilo kašnjenje prilikom izvršenja programa. Evo skice i opisa koraka:
  + Pretraga BTB-a: Prvi korak je pretraga Branch-Target-Buffer-a (BTB) kako bi se pronašao odgovarajući unos za trenutnu instrukciju koja se izvršava.
  + Predviđanje grananja: Ako se pronađe odgovarajući unos u BTB-u, to znači da se radi o instrukciji grananja. BTB sadrži informacije o ciljnoj adresi grananja kako bi se predvidjelo gdje će se program nastaviti nakon grananja.
  + Pribavljanje instrukcija: Ako je predviđanje tačno, instrukcije sa ciljne adrese grananja se pribavljaju i ubacuju u pipeline za daljnje izvršenje.
  + Nastavak izvršenja: Nakon što su instrukcije sa ciljne adrese grananja pribavljene, program nastavlja sa izvršavanjem bez zastoja.
  + Netačno predviđanje: U slučaju netačnog predviđanja grananja, dodatni ciklus se troši za ponovno pribavljanje korektne instrukcije sa tačne adrese grananja.
* Skica prikazuje ove korake u procesu izvršenja instrukcije sa BTB-om, uključujući pretragu BTB-a, predviđanje grananja, pribavljanje instrukcija sa ciljne adrese, i eventualno ponovno pribavljanje u slučaju netačnog predviđanja.

1. Objasnite mehanizam prevođenja adresa kod virtuelne memorije! Skicirajte dijagram toka mogućih događaja prilikom pristupa virtuelnoj memorji!

* Mehanizam prevođenja adresa kod virtuelne memorije omogućava da se programi izvršavaju kao da imaju pristup velikom bloku memorije, iako fizički ne postoji dovoljno memorije za sve programe istovremeno. Ovaj mehanizam koristi tabele stranica kako bi se mapirale virtuelne adrese programa na fizičke adrese u memoriji.



* Dijagram toka mogućih događaja prilikom pristupa virtuelnoj memoriji:
  + Program izdaje zahtev za pristupom određenoj virtuelnoj adresi.
  + Procesor koristi Page Table (PT) registar da pronađe početak tabele stranica aktivnog programa.
  + Gornji deo virtuelne adrese (broj virtuelne stranice) se sabira sa baznom adresom tabele stranica kako bi se pronašao odgovarajući red u tabeli.
  + Ako je tražena stranica prisutna u fizičkoj memoriji, V bit u redu tabele stranica će biti postavljen.
  + Ako je stranica prisutna, procesor dobija odgovarajući redni broj fizičke stranice u memoriji.
  + Preostali deo prevođenja uključuje dodavanje ofseta virtuelne adrese bitima rednog broja fizičke stranice kako bi se dobila konačna fizička adresa.
  + Procesor pristupa fizičkoj memoriji na dobijenoj adresi radi čitanja ili pisanja podataka.
* Ovaj mehanizam omogućava efikasno upravljanje memorijom i omogućava da se programi izvršavaju bez potrebe za kompletnim fizičkim adresnim prostorom. Virtuelna memorija omogućava da se programi izvršavaju efikasno i da se resursi memorije optimalno koriste.

1. Kako se vrši adresiranje U/I uređaja (dva načina)? Objasnite!

* Postoje dva osnovna načina adresiranja U/I uređaja.
* Njihove adrese mogu biti dio memorijskog prostora (eng. memory mapped I/O), odnosno dio memorijskog prostora se odvaja za U/I uređaje. Tako se gubi (mali) dio memorijskog adresnog prostora, a procesor pristupa U/I uređajima i njihovim veznim registrima istim instrukcijama (ali različitim adresama!) kao i svim ostalim memorijskim lokacijama. Naravno, fizička memorija ignoriše pristupe dijelu memorijskog prostora odvojenog za U/I uređaje.
* Drugi način adresiranja je da se koristi odvojeni U/I adresni prostor (eng. U/I mapped I/O), mnogo manji od memorijskog (dovoljno je 10 do 15-bitni adresni prostor). Razlika u pristupu U/I uređajima u odnosu na memorijski prostor sada nije samo u odabranim adresama već i u različitim upravljačkim signalima. Isti opseg adresa se može koristiti u oba adresna prostora.
* Kod odvojenog U/I adresnog prostora, kariste se posebne instrukcije za pristup veznim registrima kontrolera uređaja – npr. umjesto load i store za pristup memorijskim lokacijama, koriste se in i out instrukcije.

1. Kako se može realizirati paralelizam na nivou instrukcija? Objasniti na dva primjera!

* Paralelizam u izvršenju instrukcija se odvija u PS-i ako instrukcije nisu međusobno zavisne. Statistički 15% instrukcija u cjelobrojnim operacijama su grananja (dinamička), pa se 6 do 7 instrukcija obavi u osnovnim blokovima između dva grananja. To umanjuje mogućnost preklapanja u izvršenju instrukcija. Da bi se postigle veće performanse, neophodno je iskoristiti paralelizam na nivou instrukcija (Instruction-Level Parallelism - ILP) u više osnovnih blokova istovremeno.

1. Superskalarni i VLIW procesori - sličnosti i razlike? Objasniti na po jednom primjeru!

* Sličnosti između superskalarnih i VLIW procesora:
  + Oba pristupa omogućavaju paralelno izvršavanje više instrukcija istovremeno radi poboljšanja performansi sistema.
  + Cilj oba pristupa je povećanje brzine izvršavanja instrukcija i efikasnije korišćenje resursa procesora.
  + Zahtevaju odgovarajuću podršku kompajlera koji može generisati odgovarajuće instrukcije za paralelno izvršavanje.
* Razlike između superskalarnih i VLIW procesora:
  + Superskalarni procesori:
    - Superskalarni procesori mogu pokrenuti različit broj instrukcija u jednom ciklusu, u zavisnosti od dostupnih resursa.
    - Instrukcije se dinamički raspoređuju i izvršavaju u zavisnosti od dostupnih resursa i zavisnosti između instrukcija.
    - Primer: U superskalarnom procesoru, dok se jedna instrukcija izvršava, druga instrukcija može pristupiti memoriji, dok treća izvršava aritmetičku operaciju.
  + VLIW procesori:
    - VLIW procesori izvršavaju fiksni broj instrukcija istovremeno, koje su grupisane u jednu dugu instrukciju ili paket instrukcija.
    - Instrukcije se statički raspoređuju i izvršavaju istovremeno, bez potrebe za dinamičkim raspoređivanjem.
    - Primer: U VLIW procesoru, jedna duga instrukcija može sadržati više operacija poput pristupa memoriji, aritmetičkih operacija i grananja, koje se izvršavaju istovremeno.
* Na primeru dodavanja skalara nizu u memoriji, superskalarni procesor bi istovremeno izvršavao nezavisne instrukcije poput učitavanja, dodavanja i čuvanja rezultata, dok bi VLIW procesor zahtevao da kompajler unapred pripremi paket instrukcija koje se mogu izvršiti istovremeno. Oba pristupa imaju svoje specifičnosti u pogledu upravljanja instrukcijama i resursima, što ih čini pogodnim za različite vrste aplikacija i zahteve sistema.

1. Objasnite sličnosti i razlike između dva pristupa za protočnu strukturu kod superskalarnih i VLIW (eng. Very Long Instruction Word) procesora na proizvoljno odabranom primjeru.

* Superskalarni i VLIW (Very Long Instruction Word) procesori su oba oblici paralelnih računarskih arhitektura koji omogućavaju izvršavanje više instrukcija istovremeno radi poboljšanja performansi sistema. Evo sličnosti i razlika između ova dva pristupa na primeru dodavanja skalara nizu u memoriji:
* Sličnosti:
  + Oba pristupa omogućavaju paralelno izvršavanje više instrukcija istovremeno.
  + Cilj oba pristupa je povećanje brzine izvršavanja instrukcija i poboljšanje performansi sistema.
  + Oba pristupa zahtevaju određenu vrstu kompajlera koji može generisati odgovarajuće instrukcije za paralelno izvršavanje.
* Razlike:
  + Superskalarni procesori:
    - Superskalarni procesori pokreću različit broj instrukcija po ciklusu i koriste statičko ili dinamičko raspoređivanje instrukcija.
    - Instrukcije koje se istovremeno pokreću moraju biti nezavisne i zadovoljiti određene preduslove, kao što je dozvoljen samo jedan pristup memoriji po ciklusu.
    - Broj pokrenutih instrukcija po ciklusu može varirati u zavisnosti od nezavisnosti instrukcija.
  + VLIW procesori:
    - VLIW procesori pokreću određen broj instrukcija formatiranih kao jedna velika instrukcija ili kao paket instrukcija.
    - Odgovornost za kreiranje paketa instrukcija koje se mogu istovremeno pokrenuti je na kompajleru, dok hardver ne može naknadno uticati na to.
    - VLIW procesori zahtevaju precizno planiranje i raspoređivanje instrukcija od strane kompajlera.
* Na primeru dodavanja skalara nizu u memoriji, superskalarni procesor bi istovremeno izvršavao nezavisne instrukcije poput učitavanja, dodavanja i čuvanja rezultata, dok bi VLIW procesor zahtevao da kompajler unapred pripremi paket instrukcija koje se mogu izvršiti istovremeno. Oba pristupa imaju svoje prednosti i mane u zavisnosti od zahteva sistema i aplikacija koje se izvršavaju.

1. Koje se performanse prate (mjere) kod paralelnih računarskih arhitekutra? Na koji način se mogu poboljšati? Objasniti!

* Kod paralelnih računarskih arhitektura, prate se različite performanse kako bi se procenila efikasnost sistema. Neke od ključnih performansi koje se prate i mere u paralelnim računarskim arhitekturama su:
  + Performanse procesora (CPU): Ove performanse uključuju brzinu izvršavanja instrukcija, broj instrukcija koje se mogu izvršiti u jedinici vremena, kao i efikasnost korišćenja resursa procesora.
  + Performanse međuveza: Ove performanse obuhvataju kašnjenje (latenciju) i propusnost (bandwidth) komunikacionih kanala između procesora i memorijskih jedinica. Brže i efikasnije međuveze mogu poboljšati performanse sistema.
  + Performanse memorije: Ove performanse uključuju brzinu pristupa memoriji, kapacitet memorije i efikasnost korišćenja keš memorije. Brza i efikasna memorija može značajno poboljšati performanse sistema.
  + Performanse komunikacije: Ove performanse obuhvataju brzinu i pouzdanost komunikacije između procesora i drugih komponenti sistema. Efikasna komunikacija može smanjiti kašnjenja i poboljšati ukupne performanse sistema.
* Da bi se poboljšale performanse paralelnih računarskih arhitektura, mogu se primeniti različite strategije, uključujući:
  + Optimizacija algoritama: Korišćenje efikasnijih algoritama može smanjiti opterećenje sistema i poboljšati performanse.
  + Paralelizacija zadatka: Razbijanje zadatka na manje delove koji se mogu izvršavati paralelno na više procesora može ubrzati izvršavanje.
  + Optimizacija komunikacije: Efikasno upravljanje komunikacijom između procesora i minimiziranje kašnjenja u prenosu podataka može poboljšati performanse sistema.
  + Poboljšanje hardverskih komponenti: Nadogradnja procesora, memorije i međuveza sa bržim i efikasnijim komponentama može značajno unaprediti performanse sistema.
* Kombinacija ovih strategija može doprineti poboljšanju performansi paralelnih računarskih arhitektura i omogućiti efikasnije izvršavanje različitih zadataka.

1. Šta su to paralelne računarske arhitekture i kako se projektuju? Objasniti!

* Paralelne računarske arhitekture su arhitekture računarskih sistema koje koriste više procesorskih elemenata koji istovremeno rade na izvršavanju zadatka. Ovi sistemi omogućavaju paralelno izvršavanje instrukcija i podataka radi poboljšanja performansi i efikasnosti računarskih sistema.
* Prilikom projektovanja paralelnih računarskih sistema, osnovni koraci uključuju:
  + Odabir procesorskih elemenata: Potrebno je odabrati odgovarajući broj i tip procesorskih elemenata koji će biti korišteni u sistemu. To mogu biti standardni mikroprocesori ili specijalizovane strukture poput aritmetičko-logičkih jedinica.
  + Odabir memorijskih elemenata: Važno je odabrati odgovarajuće memorijske elemente koji će podržati rad procesorskih elemenata i omogućiti efikasno deljenje podataka između procesora.
  + Povezivanje elemenata: Nakon odabira procesorskih i memorijskih elemenata, potrebno je razmotriti kako će se ovi elementi međusobno povezati. To uključuje dizajn mreže za komunikaciju između procesora i memorijskih jedinica.
* Paralelne računarske arhitekture mogu biti multiprocesorske ili multiračunarske. U multiprocesorskim arhitekturama, svi procesori dele zajedničku memoriju, dok u multiračunarskim arhitekturama svaki procesor ima sopstvenu memoriju. Ovi sistemi se mogu primeniti za rešavanje različitih problema, a programski modeli se mogu prilagoditi tako da omoguće efikasnu komunikaciju među procesima.
* U suštini, projektovanje paralelnih računarskih arhitektura zahteva pažljiv odabir procesorskih i memorijskih elemenata, kao i efikasno povezivanje ovih elemenata radi postizanja visokih performansi i efikasnosti sistema.

1. Objasni koje metode "prikrivanja" kašnjenja postoje kod paralelnih računarskih arhitektura!

* Postoje različite metode "prikrivanja" kašnjenja kod paralelnih računarskih arhitektura. Neke od tih metoda su:
  + Višenitnost: Ova tehnika omogućava izvršavanje više lakih procesa ili niti istovremeno ili pseudo-istovremeno kroz vremensko dijeljenje procesora. Brzo prebacivanje konteksta među nitima omogućava neprekidno izvršavanje bez gubitka vremena na čekanje, što pomaže u prikrivanju kašnjenja.
  + Neprekidni upis u memoriju: Ova metoda omogućava procesoru da nastavi izvršavanje nakon pokretanja upisa u memoriju, dok se završetak memorijskog ciklusa prepušta dodatnom hardveru. Na taj način se štedi vrijeme i smanjuje kašnjenje.
  + Paralelno procesiranje: Korištenje više procesora istovremeno može pomoći u prikrivanju kašnjenja. Ako se niz podataka dovoljno dugo obrađuje, svi procesori mogu biti istovremeno zauzeti, što pomaže u efikasnijem izvršavanju.
  + Paralelizam na nivou instrukcija: Korištenje paralelizma na nivou instrukcija u protočnim i superskalarnim strukturama može pomoći u postizanju ubrzanja u odnosu na klasično sekvencijalno izvršavanje instrukcija. Korištenje velikog broja procesora istovremeno može dodatno ubrzati izvršavanje.
* Ove metode omogućavaju paralelnim računarskim arhitekturama da efikasnije upravljaju kašnjenjima i optimiziraju performanse sistema.