Građa računala – Među ispit dva

1. ARM i ARM arhitektura

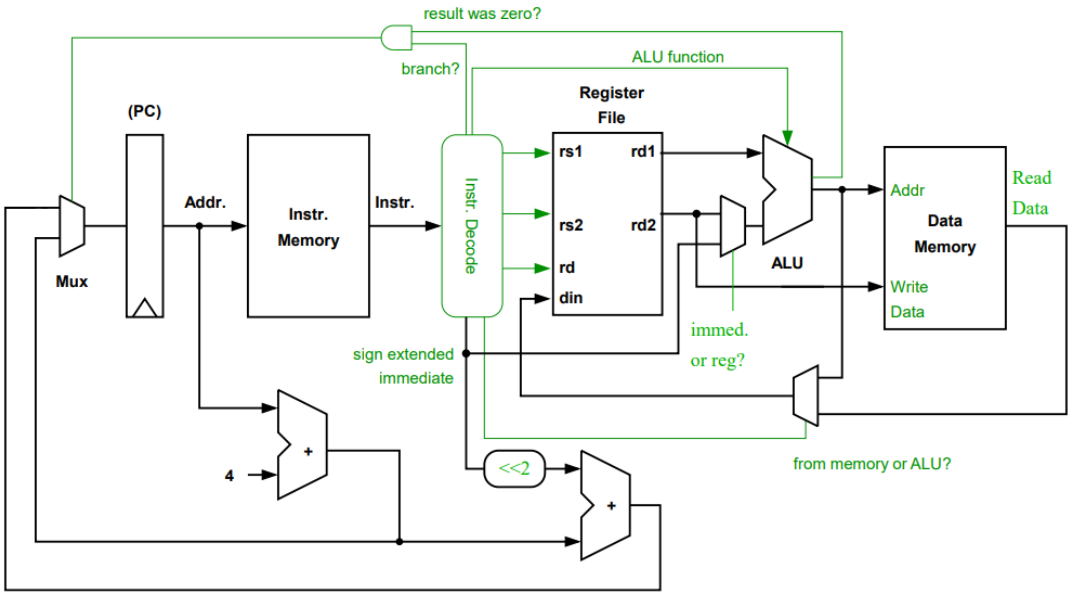
- **Instructoion Set Arhitektura** (*Poredano od high-level interface prema low-level interface*)
    - Izvršavanja viših jezika direktno
    - Izvršavanje kompleksnih instrukcija (CISC)
    - „Složiti“ instruction set za implementaciju visoko-performantnog pipelinea. „Pokazati“ pipeline za instrukcije kompajleru radi optimizacije koda i pojednostavljenja hardvera (RISC)
    - Dodati dodatne eksplicitne informacije o međuovisnostima između instrukcija (VLIW ili slično)
- **ISA**
    - Najbolji **IS** je onaj koji daje najbolju implementaciju
    - Promjene **IS** su teške i ne dešavaju se često
    - Faktori koji utječu na dizajn **IS** se mijenjaju s vremenom; *npr. aplikacije, programski jezici, tehnologije komunikacije, budžeti, proizvodna tehnologija*
    - Ne trebamo uključivati besmislen dodatne mogućnosti
- **RISC pristup**
    - Sve uobičajene primjene napraviti jako brzima kroz odabir najkorisnijih instrukcija, vrsta adresiranja i sl.
    - Instrukcije dizajnirane da dobro koriste register file
    - **RISC ISA** je dizajnirana za jednostavnu implementaciju visokih performansi
- **Uobičajene mogućnosti ISA:**
    - Instrukcije fiksne duljine
    - Svaka instrukcija slijedi slične korake za vrijeme izvršavanja
    - Pristup podatkovnoj memoriji je ograničen na specijalne load/store **instrukcije**
- **ARM 1: Prvi ARM Procesor**
    - **ARM: Advanced RISC Machine**
    - ARM 1:
        - 25 000 tranzistora
        - 3-stage pipline
        - 8 MHz clock
        - No on-chip cache
- **AArch64 – Kako se razlikuje od starijih ARM ISA-a?**
    - Gotovo potpuno eliminirao uvjetno izvršavanje
    - Nema posmicanja
    - PC nije dio registrer set-a

    - Nema mogućnosti za load-store više instrukcija istovremeno
    - Puno jednostavnije kodiranje instrukcija
- **A64 instrukcije**
    - 64-bit pointeri i registri
    - Fixed-length 32-bit instrukcije
    - Load/store arhitekture
    - Jednostavni način adresiranja
    - 32x64-bit registra opće namjene
    - PC ne može biti specificiran kao odredište procesiranja podataka ili load instrukcija
- **AArch64 – registri**
    - U execution fazi, svaki registri (*X0-X30*) je širine 64 bita (Povećana širina pomaže da se smanji pritisak na register u većini aplikacija)
    - Svaki 64-bit registar opće namjene, također ima i 32-bitnu formu
- **AArch64 načini adresiranja**
    - Base register only
    - Base plus offset
    - Pre-indexed
    - Post-indexed
- **AArch64 – procesiranje podataka**
    - Vrijednost u registrima procesiramo kroz različite instrukcije:
        - Aritmetičke, logičke, seljenje podataka, manipulacije bitovima, posmicanje, uvjetne usporedbe i sl.
    - Ove instrukcije uvijek rade između registara
- **AArch64 – uvjetno izvršavanje**
    - A64 set instrukcija ne uključuje koncept široko rasprostranjenog prediciranog ili uvjetnog izvršenja
    - NZCV registar sadrži kopije zastavica stanja N, Z, C i V
    - Osiguran je mali skup uvjetnih uputa za obradu podataka koje koriste zastavice uvjeta kao dodatni ulaz. Uvjetno se izvršava samo uvjetna grana
        - Uvjetno grananje
        - Dodaj i oduzmi sa prijenosom
        - Uvjetni odabir s povećanjem, negiranjem ili invertiranjem
        - Uvjetna usporedba
- **ARMv9 arhitektura**
    - Predstavljen 2021.
    - Fokus na:
        - Security
        - AI, SVE2
        - Compute dizajn
    - Na neki način, moglo bi se reći da je Armv8 bio pomalo orijentiran no stolno računalo

- Na neki način, moglo bi se reći da je Armv9 više naginje super računalstvu, u arhitekturnom smislu – za SC, za HPC
- Razlog je vrlo jednostavan – zahtjevi i potrebe dizajna

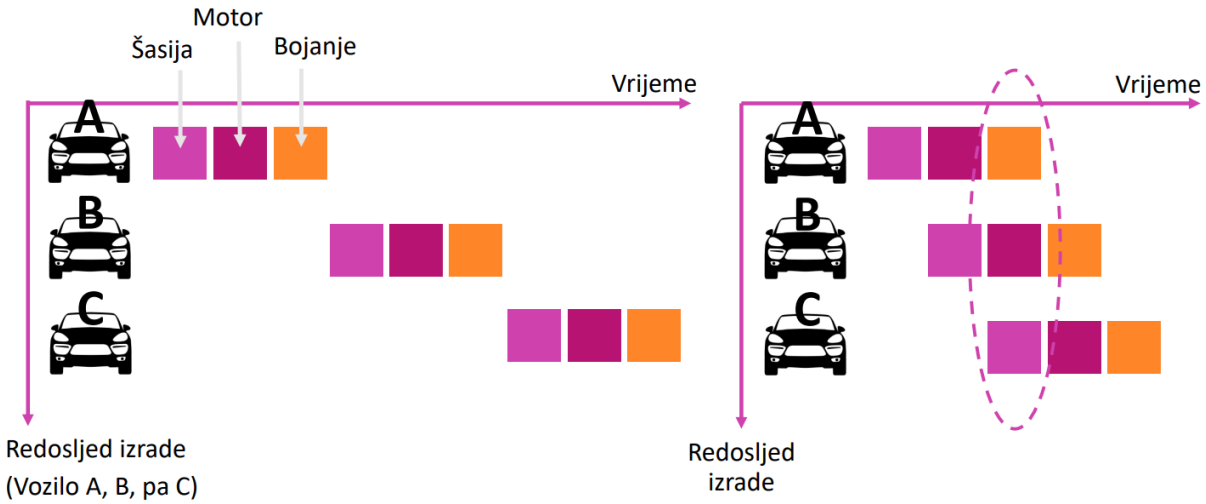
2. Pipeline, rizici u pipeline strukturama i utjecaj na performanse

- **Pojednostavljeni procesor**



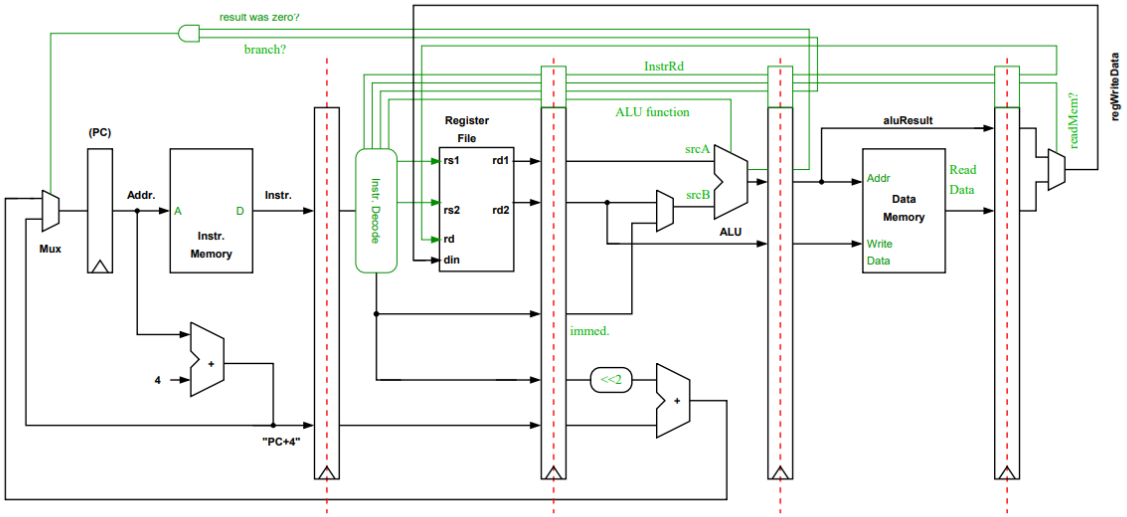
- Izvršava jednu instrukciju po ciklusu (CPI faktor je točno 1)
- Minimalni ciklus definiran je kao najgori put kroz sve logičke sklopove i memoriju, na što su pribrojene oscilacije i mogući pomaci zbog utjecaja procesora, napona i temperature

- **Što je pipelining?**



- Pokušavamo preklopiti različite faze rada kako bi iskoristili vremenski paralelizam u samom procesu
- Izvršavanje instrukcija možemo razdvojiti na stupnjeve te preklopiti izvršavanje različitih instrukcija
- Moramo osigurati da su rezultati našeg novog procesora koji sadrži pipeline strukture isto kao i rezultat procesora koji ih nema
- Naše izmjene u načinu izvršavanja instrukcija ne smiju utjecati na krajnji rezultat obrade podataka

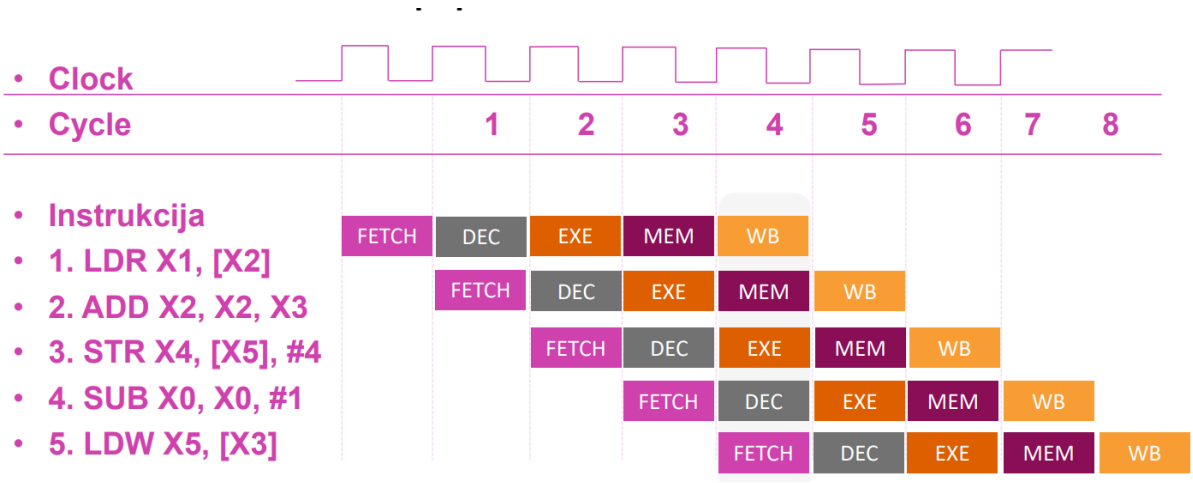
- **Pipelining u našem procesoru**



- U naš procesor možemo ubaciti dodatne registre koji će omogućiti da se logika može podijeliti u stupnjeve pri izvršavanju
- Ako su ovakvi registri ispravno ubace mogu smanjiti kašnjenje uzrokovano najgorim slučajem prebacivanja podataka između registara
- Potrebno je dodati i dodatne kontrolne signale kako bi informacije o upravljanju podacima pratile naše instrukcije kroz sve stupnjeve izvršavanja

- **Pipeline našeg procesora**
  - Uzeli smo osnovni procesor i dodali u njega pipelining registre time mijenjajući tijek podataka. Ovo mijenja rad našeg procesora.
  - Ovime stvaramo pipeline koji ima 5 stupnjeva:
    - **Fetch –** pristup našoj memoriji sa instrukcijama
    - **Decode –** dekodiranje instrukcije i čitanje
    - **Execute –** izvršavanje instrukcija u ALU, izračun memorijskih adresa, po potrebi izračun odredišnih adresav
    - **Memory –** pristup podatcimaa u memoriji
    - **Writeback –** zapisivanje podataka natrag u registar

- **Kako funkcionira pipeline**



- **Idealni pipeline**
  - U idealnom slučaju naš CPI je 1, dakle nikada se ne događaju usporavanja u pipeline-u
  - Bilo kakvo usporavanje pipeline-a znači povećanje CPI
  - *Napomena: Ako želimo maksimalno iskoristiti pipeline moramo izbjegavati bilo kakva čekanja, bez povećavanja trajanja jednog ciklusa*
- **Čekanje zbog pristupa memoriji**
  - Latencija zbog pristupa memoriji koja nije u procesoru je 10 – 100 puta veća od trajanja jednog ciklusa našeg procesora
  - Da bi izbjegli kašnjenje i čekanje moramo koristiti cache memoriju u procesoru
- **Rizici pipeline strukture**
  - Struktura pipeline-a omogućava novim instrukcijama da se krenu izvršavati dok su prethodne instrukcije još u fazi izvršavanja, dakle izvršavanje se preklapa
  - Postoje situacije u kojima instrukcija mora čekati na rezultat neke od prethodnih instrukcija kako bi se osigurala ispravnost izvršavanja
  - Ove situacije nazivamo rizikom te ih dijelimo na nekoliko slučajeva:
    - **Structural hazard –** strukturni rizici zbog pristupa dijeljenim resursima
    - **Data hazard –** podatkovni rizici zbog potrebe da se osigura međuovisnost podataka u različitim instrukcijama
    - **Control hazard –** kontrolni rizici koje uzrokuju instrukcije koje mijenjanju PC registar, dakle grananja i skokovi u programu
- **Podatkovna ovisnost; prava međuovisnost podataka**
  - **Prava podatkovna ovisnost – poznata kao i Read-After-Write (RAW)** ovisnost
  - Zamislimo dvije instrukcije, instrukciju *i* nakon koje slijedi instrukcija *j*
  - Ako *j* koristi rezultat instrukcije *i* onda kažemo da je *j* **podatkovno ovisan** o *i*

- **Podatkovna ovisnost; ovisnost o imenu**
  - **Izlazna ovisnost – poznata i kao Write-After-Write (WAW)**
    - Moramo se pobrinuti da ne posložimo pisanje u isti registar odnosno da ne mijenjamo red kojim se sadržaj registra mijenja
    - To bi značilo da bi naknadne upute mogle dobiti pogrešnu vrijednost podataka
  - **Anti-ovisnost – poznata i kao ovisnost o pisanju nakon čitanja (WAR)**
    - Moramo biti oprezni da ne prebrišemo registar čija je trenutna vrijednost još uvijek potrebna ranijim instrukcijama
- **Kontrolni rizici – procjena grananja u fazi dekodiranja**
  - Da bismo smanjili troškove grananja, mogli bismo procijeniti grananje u fazi dekodiranja
  - Neizvršeno grananje uključuje samo jedan „mrtvi“ ciklus
  - Potencijalni rizici za podatke:
    - Ako instrukcije neposredno prije grananja piše u registar koji se testira pri grananju, moramo pričekati jedan ciklus
    - Također će nam trebati staze za prosljeđivanje od faza EXE i MEM do faze dekodiranja
- **Pipeline CPI**
  - Čekanja se mogu smanjiti kombinacijom kompajlera i hardverskih tehnika
  - Hardverske tehnike obično povećavaju površinu i složenost našeg procesora
- **Tipične duljine pipeline-a**
  - Jezgre optimizirane za manju površinu mogu imati 2-3 faze
  - Jednostavni, učinkoviti skalarni pipeline-ovi obično provode 5-6 faza
  - Jezgre sa višim performansama mogu imati 8-16 faza
  - Procesori opće namjene su dosegli vrhunac od 31 faze sa Intelovim Pentiumom 4

3. Napredni pipeline: Predviđanje grananja, iznimke i ograničenja u pipeline konceptu

- **Upravljački hazardi**
  - Kada dohvatimo instrukciju sa uvjetom grananja moramo odlučiti događa li se grananje ili ne te ako se grananje događa moramo izračunati i prebaciti odredišnu adresu
- **Kako želimo rukovati grananjem**
  - Opcija 1: Pretpostavimo da se grananje nije dogodilo
  - Opcija 2: Izračunamo grananje što je ranije moguće
  - Opcija 3: Pričekamo na grananje
  - Opcija 4: Predviđamo grananje

- **Pretpostavimo da se grananje nije dogodilo**
    - Ako procjenjujemo grananje u fazi izvršenja, izgubili bismo dva ciklusa svaki put kada bismo naišli na grananje koje treba izvršiti
- **Izračunamo grananje što je ranije moguće**
    - Premjestimo test grananja i izračun i ciljne adrese grananja u fazu dekodiranja
    - Time bi smo kazana grananja smanjila na jedan ciklus u slučaju izvršenog grananja
        - i.e., u slučaju grananja moramo odbaciti iduću instrukciju u pipeline-u
    - Da bi ova tehnika djelovala:
        - Stanje grananja mora biti jednostavno procijeniti
        - Moramo se pobrinuti za potencijalno podatkovne rizike
- **Pričekamo grananje**
    - Mogli bismo odlučiti uvijek izvršiti instrukcije nakon grananja, bez obzira na to je li grananje izvršeno
    - Ova instrukcija iza grananja se sad zove „**branch delay slot**“
    - Kompajler obično može popuniti branch delay slot 60-70% ukupnog vremena izvršavanja
    - Ako imamo više faza u pipeline-u možemo kreirati dodatne delay slotove no time ih ima previše pa ih je teško popuniti
    - Ako je instrukciju nemoguće pronaći onda na njezino mjesto ide NOP instrukcija
- **Predviđanje grananja**
    - Za procesor visokih performansi, s dubokim pipeline strukturama, do sada opisane tehnike nisu dovoljne
    - Primjerice ARM Cortex-A157
        - 15 stupnjeva pipeline
        - Grananja se računaju u kasnijim fazama izvršavanja
        - Svaka greška u procijeni grananja je oko 14 ciklusa kašnjenja
        - Procesor dohvaća do 4 instrukcije po ciklusu i dekodira po 3 istovremeno
        - Rezultat svega ovoga je da krivo procijenjeno grananje može značiti ponovno izvršavanje više od 40 instrukcija
- **Statičko predviđanje grananja**
    - Metode statičkog predviđanja iskorištavaju opažanje da će određeno grananje vjerojatno biti vrlo pristrano u jednom smjeru
    - Sheme se često temelje na tome grana li se grana naprijed ili natrag u kodu ili alternativno ovise o op-kodu instrukcija u grananju
    - **Predviđanje temeljeno na pomaku**
        - Možemo iskoristiti jednostavno opažanje da su grane koje se granaju unazad obično grane petlje i da će vjerojatno biti aktivirane

        - Ako ciljna adresa grananja < PC, predviđamo da će grananje biti izvršeno
        - Točnost ove vrste sheme je oko 65%
- **Optimizacije**
    - Tournament Prediktori
    - Problemi s pseudonimom
- **Ograničenja dinamičkog predviđanja**
    - Neke grane su nepredvidive
    - Trebate „trenirati“ prediktore za neko razdoblje prije nego što predviđanja budu točna
    - Točnost prediktora biti će ograničena područjem, vremenom ciklusa ili snagom hardvera
    - Pseudonimi i smetnje
- **Grananje bez zastoja**
    - Kako bi izbjegli usporavanja u grananju trebamo znati koja instrukcija koje „trenutno“ donosimo je grananje
        - Moram predvidjeti hoće li se grananje dogoditi
        - Ako se grananje događa moramo odrediti odredišnu adresu
    - Da bi smo pružili informacije za rješavanje ovih problema, navedena grananja pohranjujemo zajedno s njihovim ciljnim adresama u Branch Target Buffer (BTB)
- **Preklapanje grana**
    - Možemo pohraniti ciljne instrukcije u BTB
    - Nemamo potrebe za dohvaćanjem sljedećih uputa
    - Također di mogli omogućiti da odvojio više vremena za pristup BTB
    - Grananje je sada uklonjeno iz slijeda instrukcija koji je predstavljen pipeline-u za izvršavanje (zamjenjuje se ciljnom instrukcijom grane)
- **Prediktor povratnih adresa**
    - Funkcije se mogu pozvati s više mjesta u programu
    - Točnost cilja grane pohranjen u BTB-u može biti vrlo niska
    - Rješenje: za pohranu tih adresa koristite mali hardverski stog
- **Iznimke**
    - Vrste iznimki za ARM:
        - Interrupts
        - Aborts
        - Reset
        - Exception generating instructions

4. Paralelizam – single i multi-core procesori

- **Tipovi paralelizama**
  - Paralelizam u hardveru
    - Single-core – pipelining, superscalar, VLIW
    - SIMD instrukcije, Vector processors, GPU
    - Multi-core – SMP, distributed-memory MP
    - Multicomputer (klasteri)
  - Paralelizam u softveru
    - Instruction-level parallelisam
    - Task-level parallelisam
    - Dana parallelisam
    - Transaction level parallelisam
- **Instruction-level parallelisam**
  - Više instrukcija iz istog toka instrukcija se izvršavaju u isto vrijeme
  - Generiran i upravljan od strane hardvera ili kompajlera
  - Pristup koji je u praksi limitiran podatkovnim i kontrolnim međuovisnostima
- **Thread-level ili task-level parallelisam**
  - Više thread-ova ili sekvenci instrukcija iz iste aplikacije izvršavaju se konkurentno
  - Generiran od strane kompajlera, upravljan od strane kompajlera ili hardvera
  - U praksi, limitiran zbog overhead-a u komunikaciji i sinkronizaciji, kao i karakteristika različitih algoritama
- **Data-level parallelisam**
  - Instrukcije iz jednog toka operiraju konkurentno na više podataka
  - U praksi, limitiran od strane nepravilnih uzoraka manipulacija podataka i bandwidth-a memorije
- **Transaction-level parallelisam**
  - Više procesa iz različitih transakcija se izvršavaju konkurentno
  - U praksi limitiran zbog overhead-a konkurentnosti
- **Superscalar i Superpipelined procesori**
  - Superscalar se proizvode češće od superpipelined procesora zbog:
    - Problemi sa povećanim frekvencijama rada procesora
    - Neke operacije ili moduli se teško razbijaju na pipeline operacije
    - Potreba za balansiranjem logike u različitim fazama pipeline-a
  - Ali:
    - Scheduling instrukcija je vrlo kompleksan
    - Postoje i druge metode – unutrašnji paralelizam u instrukcijskom toku, kompleksnost, grananje kroz VLIW (*very long instruction word*)

- **VLIW**
  - Instruction-level paralelizam – programi kontroliraju paralelno izvršavanje instrukcija
  - Koristi kompajler za razrješavanje svih problema
  - Potprogrami odlučuju paralelni tijek instrukcija i rješavaju konflikte
  - Povećava se kompleksnost kompajlera, ali u isto vrijeme se smanjuje kompleksnost hardvera
- **SIMD (Single-Instruction, Multiple-Data)**
  - Višeprocesorska mašina sposobna izvršavati istu instrukciju na procesoru ali raditi na različitim procesorima
  - Ovaj se pristup tipično koristi u znanstvenim izračunima pošto takva vrsta računarstva obično ima puno vektorskih i matričnih operacija
- **Vektorski procesori**
  - Sastoji se od:
    - Vektorski registri
    - Vektorizirane i pipeline jedinice
    - Interleaved memorija
    - Pristup memoriji „u koracima“, hardverski mehanizam za skupljanje i komadanje
  - Prednosti i mane u prezentaciji ili na netu
- **Grafičke kartice**
  - Koriste točke, linije i trokute za predstavljanje površine fizičkog objekta
  - Koriste pipeline grafičkog procesa za pretvaranje interne reprezentaciji u niz piksela
  - Više faza pipeline-a == programibilne faze
  - Shader funkcije definiraju ponašanje programibilnih faza
  - Shader funkcije su kratke
  - Implicitno su paralelne
  - SIMD procesor je osnova grafičkog procesiranja pošto se primjena shader funkcija na elemente grafike odvija u više faza (Postiže se kroz primjenu velikih količina ALU-a na svakoj jezgri GPU procesora)
  - Koristi se hardverski multitherading kako bi se izbejgli zastoji
  - Broj threadova ovisi o količini resursa
  - Negativna strana – potrebna velika količina threadova za efikasno korištenja
- **Ubrzanje, skalabilnost**
  - Skalabilnost – dodavanje *X* puta više resursa da bismo dobili blizu *X* puta bolje performanse
  - Postoji nekoliko modela skalabilnosti:
    - Ograničeni problemom
    - Ograničeni vremenom
  - Adekvatni model ovisi o potrebama korisnika
- **Multi-core**

    - Jezgre povezane i mogu surađivati
- **UMA/SMP**
    - Fizički centralizirana memorija, uniformni pristup
    - Sva memorija je na jednakoj "udaljenosti" od svih procesora - zbog toga se često zove i simetrično multiprocesiranje (SMP)
    - Brzina rada memorije je fiksna i mora zadovoljiti potrebe svih procesora - problem skaliranja na veći broj procesora
    - Koristi se danas - npr.u sustavima sa jednim fizičkim podnožjem za procesore
- **NUMA**
    - Fizički distribuirana memorija, nije uniformna
    - Dio memorije je alociran za svaki procesor
    - Pokušava se napraviti sustav kod kojeg je većina zahtjeva prema memoriji lokalnog tipa
    - Pristup lokalnoj memoriji je brži od pristupa remote memoriji
    - Zajedno, to znači da ako imamo većinom lokalne pristupe memoriji, brzina rada memorije se povećava linearno sa brojem procesora
    - Koristi se kod multi-socket sustava (serverski sustavi)

5. Paralelizam – paralelizam i heterogenost

- **Heterogenost (Asimetričnost)**
    - Generalni koncept dizajna sustava
    - Ideja: umjesto više instanci istog resursa dizajniramo neke instance da budu drugačije
    - Različite instance mogu biti optimizirane za veću efikasnost
- **Zašto asimetričnost**
    - Različiti workload-i se drugačije ponašaju
    - Sustavi su dizajnirani za različite metrike u isto vrijeme
    - Simetrični dizajn pokušava se prilagoditi svim metrikama i aplikacija (vrlo teško ostvarivo)
    - Asimetričnost omogućava prilagodbu
- **General vs Special (purpose)**
    - Asimetričnost omogućava specijalizaciju
    - Cilj asimetričnog dizajna je dobiti najbolje od GP (dizajn za sve) i SP (jedan dizajn po aplikaciji) svjetova
- **Prednosti i mane asimetričnosti**
    - Prednosti
        - Omogućava optimizaciju za više metrika
        - Može omogućiti bolju prilagodbu aplikaciji
        - Daje SP benefite sa GP fleksibilnosti
    - Mane
        - Veći overhead i kompleksnost dizajna i verifikacije

        - Veći overhead u upravljanju
        - Overhead u prebacivanju između komponenti može dovesti do degradacije u performansi
- **Tri ključna problema budućih sustava**
    - Memorijski sustav
        - Aplikacije su sve više intenzivne s obzirom na podatke
        - Seljenje podataka limitira performanse i efikasnost
    - Efikasnost (performanse i energija) -> skalabilnost
        - Omogućava razvoj skalabilnijih sustava - nove aplikacije
        - Omogućava bolje korisničko iskustvo - novi usage modeli
    - Predvidljivost i robusnost
        - Dijeljenje resursa i nepouzdanog hardvera = QoS problem
        - Predvidljivost performansi i QoS su primarna ograničenja
- **Dizajn multi-core sustava (heterogeni princip)**
    - Jednostavniji dizajn, troši manje struje nego jedna velika jezgra
    - Široki paralelizam na čipu
    - Što želimo:
        - N puta veću performansu ako koristimo N puta više jezgri
    - Što dobivamo:
        - Amdahl-ov zakon (usko grlo u serijskom izvršavanju)
        - Uska grla u paralelnom izvršavanju
    - Problemi paralelizma:
        - Amdah-lov zakon
        - Serijski dio koda (ne može se paralizirati)
    - Sve se zapravo svodi na to da li trebamo koristiti velike ili male jezgre
        - Velike jezgre
            - Visoke single-thread performanse, tj.serijske performanse
            - Loše performanse u paralelnom procesiranju
        - Male jezgre
            - Velika brzina izvršavanja paralelnog dijela koda
            - Loše performanse serijskog dijela koda
- **Vrste asimetričnosti**
    - Asimetričnost za energetsku efikasnost
    - Asimetričnost kroz podizanje frekvencije

6. Primjer Među ispita dva

- [I4_M, 3 boda] Objasnite što je prekidni sustav i kako radi.
    - Interrupt Koji hardverski zaustavlja program i pokrece kod koji rijesava interrupt. Postponing maskble interrupt koji se Moze softwareski Ugasiti i nonmaskble interrupt koji she nemoze zaustaviti kad se pokrene.
- [I4_M, 3 boda] Koje vrste pipeline hazarda poznajete?

        - **Structural hazard –** strukturni rizici zbog pristupa dijeljenim resursima
        - **Data hazard –** podatkovni rizici zbog potrebe da se osigura međuovisnost podataka u različitim instrukcijama
        - **Control hazard –** kontrolni rizici koje uzrokuju instrukcije koje mijenjanju PC registar, dakle grananja i skokovi u programu
- [I4_M, 4 boda] Objasnite što se događa sa procesorom ako se dogodi iznimka (exception).
    - Iznimka će uzrokovati da procesor izvrši sljedeće:
    - Spremite stanje procesora (PSTATE), npr. zastavice procesora, bitove maske prekida, razinu iznimke itd.
    - Spremanje povratne adrese (trenutni PC).
        - 1. Skok na dio za rukovanje iznimkom definiran vektorom u memoriji
        - 2. Spremite registre, izvršite kod rukovatelja iznimke, vratite registre.
        - 3. Povratak iz iznimke (instrukcija ERET).
- [I4_M, 5 boda] Koristeći sedam segmentni display ispišite samo parne brojeve s kratkom petljom koja će omogućiti da između brojki postoji vremenski razmak.
    - .global _start
      _start:
      ldr r2,=table
      mov r3,#0
      ldr r4,=SSD
      MOV R6,#0

```
loop:
        ldrb r10,[r2,r3]
        str r10,[r4]
        add r3,r3,#2
        b delay
delay:
        add r6,r6,#1
        cmp r6,#3276800
        bne delay
        mov r6,#0
        cmp r3,#10
        blt loop
        b end
end:
        b end
.data
.equ SSD,0xff200020
table:
  .byte 0x3F                    // 0
```

```
.byte 0x06          // 1
.byte 0x5B          // 2
.byte 0x4F          // 3
.byte 0x66          // 4
.byte 0x6D          // 5
.byte 0x7D          // 6
.byte 0x07          // 7
.byte 0x7F          // 8
.byte 0x6F          // 9
```

- [I5_M, 2 boda] Je li slijedeća izjava točna: „Procesori imaju ogromne količine brze cache memorije dok grafičke kartice imaju malo cache memorije. Procesori imaju vrlo brz pristup ogromnim količinama radne memorije (RAM), dok grafičke kartice imaju spor pristup grafičkoj memoriji.” Objasnite svoj odgovor.
    - Ova tvrdanja nije točna. Brzina pristupa cache memoriji varira od uređaja do uređaja, za procesore i grafičke kartice podjednako. Što se tiče tvrdnje o pristupu RAM-u, on je točna. Procesori generalno imaju brži pristup RAM-u nego grafičke kartice imaj pristup grafičkoj memoriji.
- [I5_M, 2 boda] Je li slijedeća izjava točna: I kod SMP-a i kod NUMA-e postoji problem latencije memorije, koja je značajno sporija od cache memorije. Ako se pravilno dizajnira, sustav baziran na NUMA arhitekturi će u ogromnoj količini slučajeva ostvarivati bolje performanse i manje probleme sa kvalitetom usluge (latencije pristupa memoriji, brzine pristupa memoriji). Objasnite svoj odgovor.
    - Tvrdnja je točna pod uvjetom da pokušavamo napraviti sustav kod kojeg je većina zahtjeva prema memoriji lokalnog tipa jer je pristup lokalnoj memoriji brži od pristupa remote memoriji. Ako to je što pokušavamo napraviti onda je NUMA brža, a ako se većina zahtjeva nije prema memoriji lokalnog tipa onda bi SMP bio brži.
- [i5_M, 3 boda] Kod heterogenih i asimetričnih arhitektura (primjer: CPU + GPU), koje vrste jezgri postoje i za kakve poslove bismo ih mogli idealno iskoristiti (1 bod)?
    - male CPU jezgre (akceleracija serijskih dijelova koda) i velike CPU jezgre (akceleracija paralelnih dijelova koda), uz kvalitetnu predikciju koji kod spada u koju grupu (serijski ili paralelni). Također, GPU-ove za masovno paralelne izračune.
    - male CPU jezgre (akceleracija paralelnih dijelova koda) i velike CPU jezgre (akceleracija serijskih dijelova koda), uz kvalitetnu predikciju koji kod spada u koju grupu (serijski ili paralelni). Također, GPU-ove za masovno paralelne izračune.
    - možemo sve izvršiti na paralelan način pošto je sav izvorni kod tretiran kroz kompajler i organiziran pri kompajliranju tako da se može izvoditi paralelno
    - najbolje bi bilo napraviti optimalan heterogeni OpenCL kod korištenjem heterogene platforme (CPU+GPU) jer ćemo tako uvijek dobiti najbolje performanse.

- Zašto (2 boda)?
    - Točan odgovor je B jer…
- [I5_M, 3 boda] Koje su prednosti i mane privatnog vs dijeljenog pristupa priručnoj (cache) memoriji kod višejezgrenih procesora?
    - Kod dijeljenog pristupa svi procesori dijele zajedničku memoriju te zbog toga svi imaju identično vrijeme pristupa toj memoriji, problem je to što je memorija fiksna i mora zadovoljavati potrebe svih procesora t zbog toga nastaje problem kod skaliranja na veći broj procesora.
    - Kod privatnog pristupa je dio memorije alociran za svaki procesor radi toga su procesori brži ako se zahtjev odnosi na taj dio alocirane memorije. Problem nastaje kada se zahtjev odnosi na neki drugi dio memorije te se u toj situaciji sve dramatično usporava.
- [I5_Ž, 2 boda] Iz aspekta paralelizma, zašto nam je bitno da paralelno procesiranje ima kvalitetno dizajniran memorijski sustav?
    - Ako je memorijski sustav loše dizajniran onda je rad cijelog sustava komprimiran. Ne samo da je sustav drastično sporiji, nego mogu i nastati problemi pri samom pristupu memoriji što može još više pogoršati situaciju. Također se može desiti da procesori ne mogu doći do nekog dijela memorije iz bilo kojeg razloga te zbog toga ne mogu raditi svoj posao.
- [I5_Ž, 3 boda] Koristeći Amdahl-ov zakon izračunajte koliko je ubrzanje u postocima ako su vrijednosti za petlje u primjeru u vježbama redom 2048, 1024, 5120 i 4096, a maksimalno ubrzanje koje možete postići za petlju je 3 puta.
    - A
- [I6_M, 3 boda] Da li SIMD instrukcije procesora rade različitom brzinom na SMP ili NUMA arhitekturi (1 bod). Zašto? (2 boda)
    - Da jer kod SMP arhitekture svaki procesor ima jednako vrijeme pristupa memoriji dok kod NUMA arhitekture svaki procesor ima manju lokalnu memoriji te ako se nešto nalazi u toj lokalnoj memoriji procesor do toga može doći znatno brže nego li procesor u SMP arhitekturi.
- [I6_M, 4 boda] Koja je osnovna razlika između SMP i NUMA modela organizacije sustava sa jednim ili više procesorskih utora i jednom ili više procesorskih jezgri?
    - Osnovna razlika je u memoriji. U SMP-u se sva memorija nalazi na jednakoj udaljenosti od svih procesora, dok kod NUMA-e se memorija dijeli na više dijelova i svaki dio je dodijeljen jednom procesoru.
- [I6_M, 2 boda] Koja je cijena koju kod NUMA arhitekture moramo „platiti“ ukoliko koristimo remote memoriju (memoriju drugog fizičkog procesora)?
    - Plaćamo cijenu brzine. Ako procesora mora dohvatiti nešto što se ne nalazi u njegovoj lokalnoj memoriji automatski je sporiji.
- [I6_Ž, 2 boda] Kako bi u NUMA arhitekturi izgledala optimalna memorijska hijerarhija, ako na izboru imate sve komponente koje na tržištu možete pronaći?

    - Procesor sa što više memorije, najveća memorija uz svaki procesor, NVME memorija za sve ostalo. Cilj je imati dovoljno memorije da svaki procesor nikada ne mora iči van svoje lokalne memorije.
- [I6_Ž, 3 boda] Na raspolaganju su vam dvije serverske SMP arhitekture sa dva i četiri socketa (podnožja za procesor na matičnoj ploči). Aplikacija koju trebate instalirati na jedan od ta dva servera je jako osjetljiva na brzinu rada i latenciju memorije. Uz uvjet da obje arhitekture imaju dovoljno procesorske snage za izvršavanje aplikacije čak i na jednom socketu, koja arhitektura bi bila optimalniji izbor za izvršavanje aplikacije (1 bod)? Zašto (2 boda)?
    - Arhitektura sa dva socketa bi bila optimalnija jer je, pod zadanim uvjetima, lakša za napraviti i održavati. Pošto se nalazimo u SMP arhitekturi to znači da svi procesori moraju imati jednako vrijeme pristupa memoriji te je puno lakše to napraviti sa dva procesora nego sa četiri jer to vrijeme pristupa mora zadovoljavati potrebe svih procesora.