

*Za siguran prolaz predmeta preporučam ignorirati prezentacije i pročitati knjigu. Knjiga je dobro napisana i iz nje se stvarno može shvatiti teorijski dio. Ne treba pročitati cijelu knjigu, nego samo **poglavlja 2, 3, 4, 7, 10, 11, 14, 15** i to se dosta brzo prođe (ostala poglavlja nisu vezana za gradivo predmeta). Zadatke je OK vježbati pomoću Ultimate Packa, ali treba sve dobro shvatiti s razumijevanjem, a ne učiti napamet jer tamo ima dosta grešaka.*

U ovoj skripti su samo ispisane najvažnije stvari iz knjige i Ultimate Packa, te tu i tamo shema rješavanja nekih zadataka.

von Neumann (IAS)

- CPU = aritmetičko - logička jedinica (ALU) + upravljačka jedinica
- $A = f(A, M)$
- sve instrukcije su jednodresne
- 2 strojne instrukcije = 1x40b riječ → privremeno se smještaju u MDR
 - desna riječ se smješta u instrukcijski registar (IR)
 - lijeva riječ se smješta u u privremeni registar (CR)
 - izvođenje desne instrukcije
 - lijeva ide iz CR u IR
 - PC++
- PC sadržava adresu sljedeće strojne instrukcije
- IR sadržava adresu instrukcije koja se trenutno izvodi
- adresna sabirnica = 16b
- podatkovna sabirnica = 8b

Fetch

1. $M(PC) \rightarrow IR$
2. $PC + 1 \rightarrow PC$
3. dekodira se operacijski kod instrukcije iz 1. koraka

Execute

1. $M(MAR) \rightarrow MDR$ (dohvat operanda, operand završava u MDR)
2. $AC + MDR \rightarrow AC$
3. korak 1.

Aritmetičko logička jedinica

- izvodi aritmetičke i logičke operacije
- ima:
 - sklop za zbrajanje
 - shifter
 - 40b registar AC (akumulator)
 - 40b registar MQ (proširenje akumulatora AC, 40 značajnih b ide u AC, 40 manje značajnih u MQ)
- 40b operandi = 39b znamenke + 1b predznak

Upravljačka jedinica

- na temelju dekodiranja strojne instrukcije generira sve potrebne upravljačke signale
- format strojne instrukcije IAS računala (ukupno 20b)

b0 - b7	b8 - b19
opkod	adresno polje

von Neumann (MC 6800)

- sve isto kao i za IAS + neke modifikacije
- memorijska jedinica = ROM + RAM

MPU (Microprocessor Unit)

- procesor realiziran u LSI (large scale integration) = ALU + upravljačka jedinica

ROM (Read Only Memory)

- trajno pohranjuje sadržaj
- ne može se mijenjati strojnim instrukcijama tijekom izvođenja programa
- može se čitati

RAM (Random Access Memory)

- za čitanje i pisanje tijekom izvođenja programa

PIA (Peripheral Interface adapter)

- omogućuje paralelni prijenos podataka između računala i perifernih uređaja
- za to se koriste linije PA0 - PA7 i PB0 - PB7
- handshaking: CA1, CA2, CB1, CB2
- programirljiv sklop

ACIA (Asynchronous Communication Interface Adapter)

- omogućuje serijski prijenos podataka
- Rx (Recieve) je prijemna signalna linija
- Tx (Transmit) je predajna linija

CISC

registar/sabirnica	DC	PC	akumulator	interna sabirnica	IR
veličina [b]	16	16	8	8	8

PRAVILA:

- u fazi pribavi PC ide na adresnu sabirnicu
- aktivira se READ
- WRITE - ništa ne radi
- podatak stiže na podatkovnu sabirnicu
- puni se IR u prvom taktu
- kad god nešto dobaviš, podatak moraš negdje staviti
- visoka impedancija dok instrukcija radi svoj execute
- PC se ne mijenja u fazi izvrši, osim ako se ne radi o instrukciji skoka

INSTRUKCIJE KOJE SE JAVLJAJU NA ISPITU

- LDA \$A000 - učitaj u akumulator sa \$A000
- INC \$A000 - povećaj sadržaj adrese \$A000
- DEC \$A000 - smanji sadržaj adrese \$A000
- BNE \$A000 - skoči na \$A000 (\$A000 → PC) ako rezultat prethodne operacije $\neq 0$
- JMP \$A000 - skoči na adresu \$A000
- CALL \$A000 - nije kao običan JUMP, jer stavlja na stog!
 - 1 - SP--
 - 2 - PC → SP (ode na tu adresu i radi WRITE dvije periode)
 - 3 - radi execute (JUMP)

Fetch faza

- PC → MAR
- Read
- podatak se ponavljuje na D0-D7
- sprema se u IR

Mikroprogramiranje

Tutorial za zadatke

- svi se zadaci svode na istu logiku -zadan je operacijski kod naredbe koju treba programirati
- to je adresa u CM - tu počnemo
- polja: **CA, CB, COP, CSH, CMB** → što radimo
- polja: **CAB, CBB, CNA, CEM** → kamo skačemo (adresa)
- prvi niz polja određuje instrukcija (ovisi što želimo raditi), a drugi niz polja ovisno gdje se želimo granati

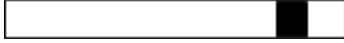

Mikroinstrukcije

- **rj_sel, rk_sel, a_sel, b_sel** - specificiraju koji od registara će biti postavljeni na sabirnice operanada (a bus i b bus – ulazi u ALU, vidi sliku 1).
Ove registre moguće je, naime, specificirati na dva načina:
 - ◆ (i) Izravno iz mikroinstrukcije upravljačkim signalima a_sel i b_sel
 - ◆ (ii) Iz makroinstrukcije (strojne instrukcije), na temelju polja ir_rj (bitovi) i ir_rk u instrukcijskom registru (ovim načinom moguće je specificirati samo registre r0 – r3, jer u makroinstrukciji imamo na raspolaganju samo po dva bita za specifikaciju registara). Koji od ovih dvaju načina će se koristiti pri izvođenju dane mikroinstrukcije, određeno je dodatnim upravljačkim bitovima rj_sel (za prvi operand) i rk_sel (za drugi operand). Ako su ovi bitovi u 0, koristi se način (i); ako su pak postavljeni u 1, koristi se način (ii). O ovoj funkcionalnosti brinu se dva multipleksora koja vidimo na samom vrhu slike 1, iznad skupa registara.
- **alu_sel** - odabire jednu od osam mogućih funkcija ALU.
- **c_in** - postavlja u 0 ili 1 ulaz c in u ALU
- **result_sel** - upravlja multipleksorom koji određuje što će se pojaviti na sabirnici rezultata (result bus) radi upisa u odredišni registar. Na slici 1 uočavamo da se o tome brine multipleksor 4/1 koji, ovisno o vrijednosti polja result_sel predviđa četiri mogućnosti što će se pojaviti na sabirnici rezultata:
 - ◆ (a) rezultat aritmetičke ili logičke operacije iz ALU (koji se nalazi na sabirnici alu bus)
 - ◆ (b) sadržaj memorijskog podatkovnog registra (MDR), odnosno podatak pročitani iz memorije

- ◆ (c) 4-bitna usputna konstanta iz instrukcijskog registra, predznačno proširena na 8 bita
- ◆ (d) 8-bitna usputna konstanta iz instrukcijskog registra.
- **ri_sel, r0_write – r7_write** - omogućuju upis podatka sa sabirnice rezultata u registre iz skupa r0 – r7. Ako je aktivan signal ri_sel, određeni registar je određen na temelju dvobitnog polja ri u instrukcijskom registru, odnosno makroinstrukciji (na ovaj način ponovno je moguće pristupiti samo registrima r0 – r3), a ako je signal ri_sel neaktivan, određene registre određuju signali r0 write – r7 write. Pošto je riječ o osam nezavisnih signala koje je moguće aktivirati istovremeno i neovisno jedne o drugima, moguće je isti podatak sa sabirnice rezultata istovremeno upisati i u više registara (iako je to rijetko kada korisno)
- **mar_sel** - aktivira upis podataka sa sabirnice alu_bus (izlaz iz ALU) u memorijski adresni registar (MAR)
- **mdr_sel** - je dvobitni signal koji aktivira upis u memorijski podatkovni registar (MDR). Podatak koji se upisuje u MDR može biti ili podatak sa sabirnice alu_bus ili podatak s vanjske podatkovne sabirnice procesora (podatak iz memorije)
- **ir1_sel, ir0_sel** - aktiviraju upis podatka s vanjske podatkovne sabirnice (iz memorije) u viši, odnosno niži bajt instrukcijskog registra
- **read, write** - upućuju se memoriji i aktiviraju čitanje, odnosno pisanje

Format mikroinstrukcije

CA	CB	COP	CSH	CMB	CAB	CBB	CST	CNA	CEM (EMIT)
2	3	2	2	3	2	2	2	6	8

- **CA** - određuje što propuštamo na lijevu sabirnicu
- **CB** - određuje što propuštamo na desnu sabirnicu
- **COP** - određuje vrstu operacije ALU
- **CSH** - prjenos ALU-S, sa ili bez posmaka
- **CMB** - prijenos sa MB na neko odredište (A, B, PC)
- **CAB** - prenosi na H  vrijednost 0, 1, SR(0) ili MB(0)
- **CBB** - prenosi na H  vrijednost 0, 1, SR(1) ili MB(15)

- **CST** - aktivnost statusnog polja
- **CNA** - proizvoljno polje za programiranje
- **CEM (EMIT)** - proizvoljno polje za programiranje

MB - Main Bus
H - mikroprogramsk adresni registar
F - mikroinstrukcijski registar

CB – veza sa sabirnicom R

000	$R \leftarrow 0$
001	$R \leftarrow B$; prijenos akumulatora B na desnu sabirnicu (R)
010	$R \leftarrow B'$; prijenos jediničnog komplementa sadržaja akumulatora B
011	$R \leftarrow PC$
100	$R \leftarrow STAT$
101	ne koristi se
...	ne koristi se
111	ne koristi se

COP – upravljanje 16-bitnim paralelnim zbrajalom

00	zbroji sa $C_{in} = 0$
01	zbroji sa $C_{in} = 1$
10	ne koristi se
11	ne koristi se

CSH – upravljanje pristupom glavnoj sabirnici

00	$MB \leftarrow Q, Q \leftarrow S$; nema posmaka
01	$MB \leftarrow Q, Q \leftarrow shr S$; posmak udesno
10	$MB \leftarrow Q, Q \leftarrow shl S$; posmak ulijevo
11	$MB \leftarrow IN$

CMB – veza glavne sabirnice (MB) i ostalih komponenti

000	nema prijenosa
001	$A \leftarrow MB$
010	$B \leftarrow MB$
011	$PC \leftarrow MB$
100	$STAT \leftarrow MB$
101	$OUT \leftarrow MB$
110	$PR \leftarrow MB$
111	ne koristi se

CAB – utjecaj na H(6)

00	$H(6) \leftarrow 0$
01	$H(6) \leftarrow 1$
10	$H(6) \leftarrow STAT(0)$
11	$H(6) \leftarrow MB(15)$; MB(15) je najmanje značajni bit 16-bitne riječi na sabirnici MB

STAT(0) = Z (zastavica)

STAT(1) = N (zastavica)

CBB – utjecaj na H(7)

00	$H(7) \leftarrow 0$
01	$H(7) \leftarrow 1$
10	$H(7) \leftarrow \text{STAT}(1)$
11	$H(7) \leftarrow \text{MB}(0)$; MB(0) je najznačajniji bit 16-bitne riječi na sabirnici MB

STAT(0) = Z (zastavica)

STAT(1) = N (zastavica)

CST – utjecaj na STAT

00	nema utjecaja
01	$\text{STAT}(0) \leftarrow \text{ZT}$
10	$\text{STAT}(1) \leftarrow \text{MB}(0)$
11	$\text{STAT}(0) \leftarrow \text{ZT}$ $\text{STAT}(1) \leftarrow \text{MB}(0)$

Tablica vrijednosti bitova ZT i MB(0) u ovisnosti o rezultatu:

ZT (izlaz iz sklopa za ispitivanje nule)	MB(0) Najznačajniji bit sabirnice MB	
0	0	rezultat > 0
0	1	rezultat < 0
1	0	rezultat = 0
1	1	nemoguća kombinacija

Priručna memorija

FORMULE

- kapacitet priručne memorije $s = n \cdot b$
- broj linija PM $n = s / b$
- veličina bloka u liniji $b = s / n$
- struktura adrese:
 $w(\text{adresa}) = w(o) + w(i) + w(p)$
 $w(p) = \log_2(b/z)$
 $w(i) = \log_2(n/a)$

OZNAKE

- p = pomak
- i = indeks
- o = oznaka
- b = veličina bloka (širina linije)
- a = adresa bloka
- n = broj linija
- s = kapacitet (toliko bine adrese)
- z = zrnatost

STRUKTURA ADRESE

servisni bitovi	w(o)	i	p
-----------------	------	---	---

Utjecaj na CPI: $CPI = CPI_{\text{OSNOVNI}} + v(\text{promašaja}) * t(\text{cijena_promašaja})$
 $t_{\text{AVG}} = t(\text{pogodak}) + v(\text{promašaj}) * \text{cijena_promašaja}$

Za zadatke mi je dosta pomogao [ovaj video](#) i slični.

MIPS

Microprocessor without Interlocked Pipeline Stages

- sve instrukcije duljine 32 bita
- 32 32-bitna registra opće namjene, \$0 - \$31

registri

broj registra	mnemoničko ime	funkcija
\$0	\$zero	konstantna vrijednost 0
\$1	\$at	rezerviran za assembler
\$2 - \$3	\$v0 - \$v1	rezultat funkcije/izraza
\$4 - \$7	\$a0 - \$a3	argumenti
\$8 - \$15	\$t0 - \$t7	privremene vrijednosti
\$16 - \$23	\$s0 - \$s7	spremljene priv. vrijednosti
\$24 - \$25	\$t8 - \$t9	privremene vrijednosti
\$26 - \$27	\$k0 - \$k1	rezervirano za jezgru OS
\$28	\$gp	globalno kazalo
\$29	\$sp	kazalo stoga
\$30	\$fp	kazalo okvira
\$31	\$ra	povratna adresa

- 3 osnovne kategorije instrukcija
 - **A** memorijske
 - **B** aritmetičko - logičke
 - **C** instrukcije grananja

A - memorijske instrukcije

- `lw $s1, 20($s2)` `// load word; $s1 ← M[$s2+20]`
- `sw $s1, 20($s2)` `// store word; M[$s2+20] ← $s1`

- **I - tip** instrukcijskog formata

6b	5b	5b	16b
op	rs	rt	konst
operacijski kod	bazni reg.	odredišni/ izvorišni reg.	predznačno proširena konst.

B.1 - aritmetičko-logičke instrukcije s registarskim operandima

- `add $s1, $s2, $s3` `// add; $s1 ← $s2 + $s3`
- `sub $s1, $s2, $s3` `// subtract; $s1 ← $s2 - $s3`
- `and $s1, $s2, $s3` `// and; $s1 ← $s2 & $s3`
- `or $s1, $s2, $s3` `// or; $s1 ← $s2 | $s3`

- **R - tip** instrukcijskog formata

6b	5b	5b	5b	5b	6b
op	rs	rt	rd	shamt	funct
operacijski kod	izvorišni reg.	izvorišni reg.	odredišni reg.	shift amount	

B.2 - aritmetičko-logičke instrukcije s neposrednim operandom

- `addi $s1, $s2, 1` `// add; $s1 ← $s2 + 1`
- `andi $s1, $s2, ff` `// and, $s1 ← $s2 & ff`
- `ori $s1, $s2, ff` `// or, $s1 ← $s2 | ff`

- **I - tip** instrukcijskog formata

6b	5b	5b	16b
op	rs	rt	konst
operacijski kod	operand	odredišni reg.	predznačno proširena konst.

C.1 - instrukcije grananja s apsolutnom adresom (bezuvjetno apsolutno grananje)

- `j 2500` // jump to target address; $PC \leftarrow 2500$
- `jal 2500` // jump and link; $\$ra \leftarrow PC+4$, $PC \leftarrow 2500$

- **J - tip** instrukcijskog formata

6b	26b
op	adr
operacijski kod	od ove adrese se nastavlja izvođenje programa

C.2 - instrukcije grananja s relativnom adresom

- **bezuvjetno registarsko grananje**

- `jr $ra` // jump register; $PC \leftarrow \$ra$

- **uvjetno grananje**

- `beq $t1, $t2, 25` // branch if equal;
(if $\$t1 == \$t2$ onda $PC \leftarrow (PC+4) + 25$)
- `bne $t1, $t2, 25` // branch if not equal;
(if $\$t1 \neq \$t2$ onda $PC \leftarrow (PC+4) + 25$)
- nema statusnog registra: uvjet se ispituje nad **2 registra opće namjene**
- grananje je relativno u odnosu na **PC+4**

- **I - tip** instrukcijskog formata

6b	5b	5b	16b
op	rs	rt	konst
operacijski kod	operand	odredišni reg.	predznačno proširena konst.

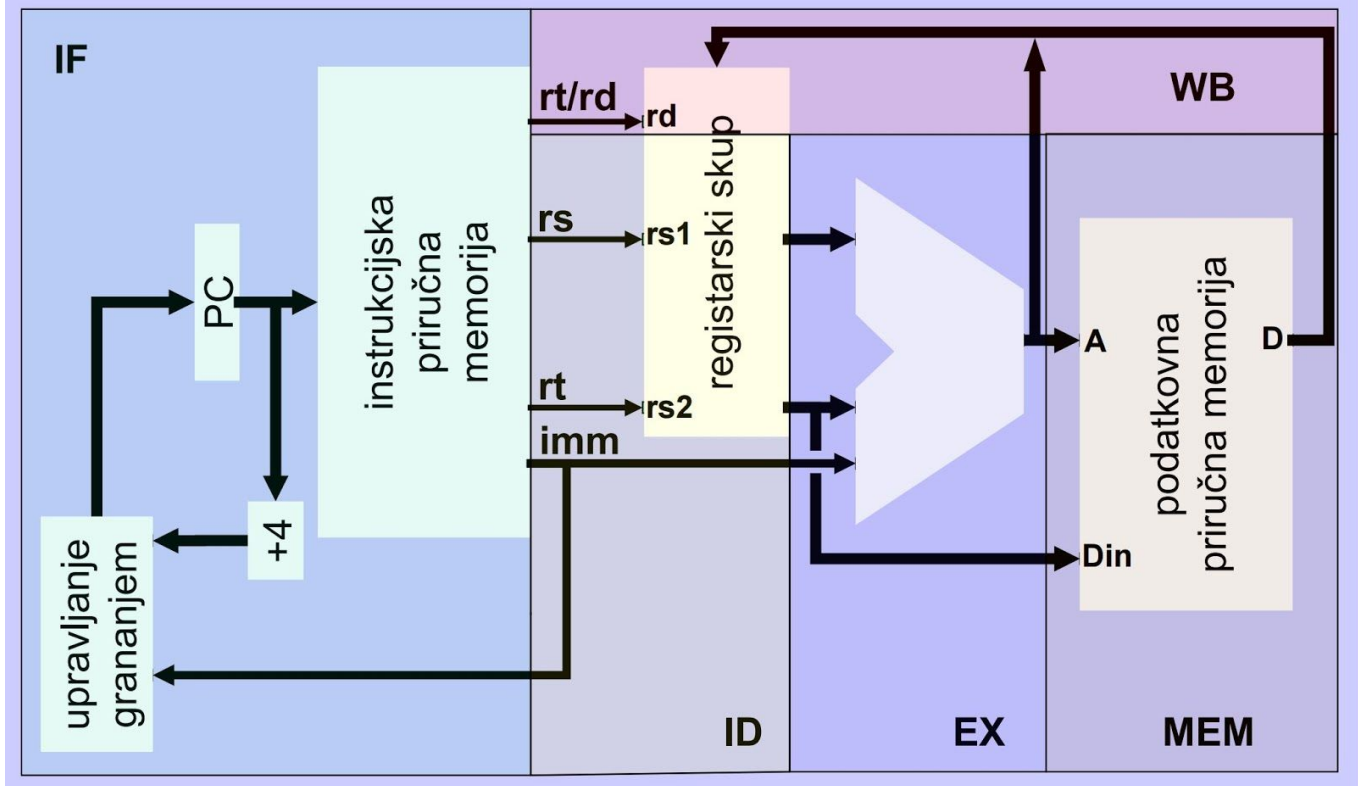
Tijek izvođenja instrukcija iz 3 najčešće grupe

1. pribavljanje instrukcije iz instrukcijske memorije (PC)
2. proslijeđivanje kodova registara registarskom skupu + čitanje registara
3. ALU određuje:
 - a. rezultat aritmetičke operacije
 - b. efektivnu adresu
 - c. odredište relativnog grananja (MIPS: ne!)
4. memorijske instrukcije pristupaju memoriji
5. upisivanje rezultata u odredišni registar

Segmenti puta podataka arhitekture MIPS:

1. **IF:** pribavljanje 32-bitne instrukcije, $PC=PC+4$
2. **ID:** dekodiranje operacijskog koda, pribavljanje registarskih operanada (0, 1, 2)
3. **EX:** zbrajanje ili posmak (aritmetika, efektivna adresa)
4. **MEM:** pristup podatkovnoj memoriji (samo memorijske instrukcije)
5. **WB:** upis odredišnog registarskog operanda (aritmetika, load)

Put podataka za računalu arhitekture MIPS - pojednostavnjeno



Hazard

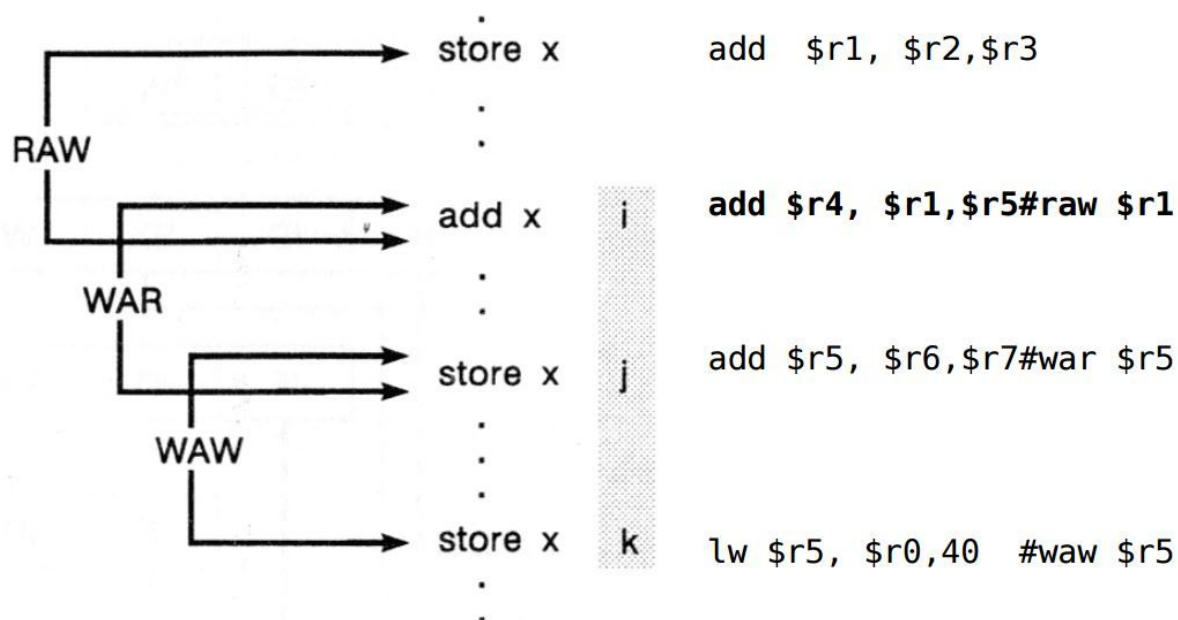
- situacija koja izaziva poremećaje i kašnjenje u “glatkom” protoku podataka kroz protočnu strukturu
- hazardi sprečavaju da sljedeća instrukcija u nizu bude izvedena u za nju predviđenom periodu signala takta
- 3 tipa:
 - **strukturni hazard**
 - **podatkovni hazard**
 - **upravljački hazard**

Strukturni hazard

- ako **IF** neke instrukcije ide u istom taktu kao i **MEM** neke druge instrukcije
 - sukob oko resursa

Podatkovni hazard

- nastupa zbog *međuviznosti podataka*
- nastaje kad dvije ili više instrukcija pristupaju istom podatku (sa load ili store)
- 3 vrste:
 - RAW - read after write / čitanje poslije upisa
 - WAR - write after read / pisanje poslije čitanja
 - WAW - write after write / pisanje poslije pisanja
- protočne arhitekture upisuju rezultat u određeni registar pri kraju instrukcijskog ciklusa pa nemaju hazarde WAW i WAR
- WAW i WAR nastaju isključivo u superskalarnim arhitekturama s dinamičkim raspoređivanjem i izvođenjem izvan redosljeda
- **RAW** - postoji opasnost da instrukcija *add x* dohvati operand s lokacije x prije nego instrukcija *store x* upiše novu vrijednost na lokaciji x
 - najviše izražen tijekom izvršenja instrukcije **load**
 - sanira se:
 - usporavanjem protočnosti (dodavanjem mjehurića)
 - prosljeđivanjem, npr. $MEM[i] \rightarrow EX[i+1]$ (jedan mjehurić)
 - zakašnjelim čitanjem
- **WAR** – instrukcija j (store x) koja logički slijedi instrukciji i mijenja podatak na lokaciji x koju čita instrukcija i (ovaj problem se ne javlja kod jednostavnih protočnih arhitektura)
- **WAW** – obje instrukcije j i k žele upisati podatak na memorijskoj lokaciji x (problem nastaje ako se instrukcija j izvede poslije instrukcije k, ovaj problem se ne javlja kod jednostavnih protočnih arhitektura)



Upravljački hazard

- nastupa kad adresu sljedeće instrukcije nije moguće izračunati prije njenog dohvata (kod instrukcija grananja)
- smanjenje kašnjenja se može postići tako da se u računanje i upis ciljne adrese grananja obavi u segmentu ID (umjesto u W ili EX)
- uz protočni mjehurić, koristi se prosljeđivanje $ID[i] \rightarrow IF[i+1]$: relativno odredište računa se u zasebnom zbrajalu, u okviru sklopa za upravljanje grananjem (glavno zbrajalo u to vrijeme računa rezultat prethodne operacije!)

Algoritam za identifikaciju hazarda

1. **strukturni hazard:** preklapanje **IF** i **MEM** faze
 2. **podatkovni hazard:** gledamo naredbu po naredbu i ako dvije uzastopne naredbe koriste isti resurs, npr. naredba 2 piše u R_1 , a naredba 3 čita iz R_1 , onda **može** doći do greške
 3. **upravljački hazard:** ako imamo naredbu grananja, njenu adresu skoka nećemo znati do **WB**, dakle **IF** iduće instrukcije promašuje
- također:
1. vidi preklapa li se nešto sa IF
 2. ako se radi o memorijskoj naredbi \rightarrow strukturni
 3. vidi koriste li 2 naredbe isti resurs \rightarrow podatkovni
 4. ako je naredba grananja \rightarrow odmah stavi NOP
 5. čitaj unatrag, npr. naredba 1 piše R_1 W
 naredba 2 piše R_1 R
- dakle radi se o RAW hazardu

Virtualne memorije

Fizički Adresni Prostor (FAP) - adrese dostupne u RAM-u

Logički Adresni Prostor (LAP) - sve (moguće) dostupne adrese

- skoro uvijek $|LAP| > FAP$
- programu moramo dodijeliti dio FAP-a, pomoću neke funkcije

Najbolje pročitati taj dio u knjizi iako je to gradivo govno kako god okreneš.

Višeprocorski sustavi

- dvije vrste paralelizma
 - raspoloživi
 - funkcijski (priroda problema)
 - podatkovni (struktura podataka)
 - iskorišteni
- raspoloživi funkcijski
 - ILP (finozrnati)
 - razina petlji (srednjezrnati)
 - razina procedura (srednjetrnati)
 - razina programa (grubo zrnati)
- iskorišteni funkcijski
 - ILP (arhitektura procesora)
 - razina dretvi (arhitektura proc. + OS)
 - razina procesa (arhitektura proc. + OS)
 - razina korisnika (OS)
- VLIW - u samoj strukturi instrukcije ubačen paralelizam
 - bazira se na horizontalnom mikroprogramiranju

Podatkovni paralelizam - SIMD

- SIMD računala ga dobro iskorištavaju (vektorski CPU)
- pogodna za multimedijску primjenu

Vrste vektorskih instrukcija

1. vektor - vektor
2. vektor - skalar
3. vektor - memorija
4. redukcija vektora (traženje min/max vrijednosti u vektoru)
5. okupljanje/raspršivanje
6. maskirajuće instrukcije

Traka (lane)

- protočna jedinica kod vektorskog procesora
- ubrzanje programa vektorskim procesorom ovisi o:
 1. razini moguće vektorizacije
 2. broju komponenti vektora
 3. razini ulančanosti vektora
 4. razini preklapanja vektorskih/skalarnih operacija

Podatkovni i multimedijски paralelizam - MIMD

- ako MIMD sustav ima više procesora → “multiprocesorski” u kojem svaki procesor može obavljati svoj proces
- ako je još i višedretveni MIMD → istodobno više procesa i više dretvi
- klasifikacija:
 - UMA - procesori dijele zajedničku memoriju
 - NUMA - procesori imaju deduciranu memoriju
 - COMA -
- UMA - pogodan za <100 procesora (koji mogu razmjenjivati load/store)
- NUMA - svaki procesor ima svoju memoriju, ali svi procesori mogu pristupiti svim djelovima memorije
 - a) brzo - proc. pita svoju memoriju
 - b) sporo - proc pita tuđu memoriju
 - pogodna za >100 procesora
 - čvorovi: CPU + ULI + pristup prospojnoj mreži (procesorska nakupina)
- COMA - poseban slučaj NUMA-e, gdje se dijeli PM

PROBLEM KOHERENCIJE MEMORIJE (UMA)

- procesor ima privatnu PM, zajednički RAM → problem
- rješenje:
 - programski - prevodioc kaže koji podaci smiju u cache
 - sklopovski - npr. snooping, protokol temeljen na prospojnoj mreži (zajedničkoj sabirnici)