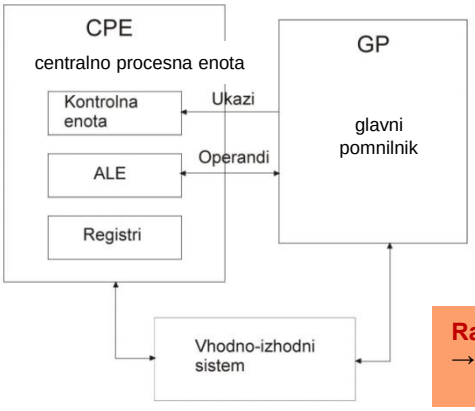
ARHITEKTURA RAČUNALNIŠKIH SISTEMOV

OSNOVE RAČUNALNIKOV



Računanje:
- → določanje vrednosti funkcije **z** = **f**(**x**)
- → x so **vhodni** podatki
- → z so **izhodni** podatki
- → **izračunljivost**: če lahko določimo z za vsak vhodi podatek x
- → ta postopek je **algoritem**
- → **Church Turningova hipoteza:** problem izračunljiv če možno v končnem številu korakov izračunati na **Turingovem stroju**

TURNING MACHINE
- → **procesor**
- → bralno pisalna **glava**
- → neskončno dolg **trak**
- → mehanizem za pomik traku
- → trak razdeljen na **celice**
- → celica je prazna ali pa v njej eden izmed znakov vhodne **abecede** ki so vhodni podatek
- → definiramo **začetno stanje**
- → izvršitev koraka je **ukaz**
- → po končnem številu korakov mora biti na traku z znaki abecede **rezultat**
- → **končni avtomat**

za vsako **stanje** in **vhodno črko** definiramo naslednje stanje torej **pomik glave** in **izhodno črko** ki se piše na trak

Računalniki:
- → **Von Neumanov** model: **Turningov stroj** z končnim pomnilnikom

stanje	prebrani znak	zapisani znak	pomik	naslednje stanje
S0	_	_	D	S0
S0	0	0	D	S1
S0	1	1	D	S1
S1	0	0	D	S1
S1	1	1	D	S1
S1	_	_	L	S2
S2	0	1	L	S3
S2	1	0	L	S2
S2	_	1	L	S3
S3	0	0	L	S3
S3	1	1	L	S3
S3	_	_	*	halt

stroj za inkrement **binarnega** števila

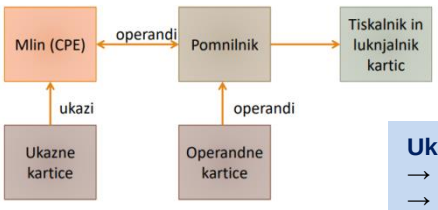
Neizračunljivi problemi:
- → **halting problem**: ni mogoče napisati algoritma da izračuna ali se bo Turningov stroj kdaj ustavil
- → **teoretično** raziskujemo izračunljivost

Digitalni princip:
- **diskretno** predstavljena števila
- **omejeno** število stanj
- natančnost povečamo z **več mesti**

Neobvladljivi problemi:
- → omejen **pomnilnik**
- → omejen **čas**
- → imamo **polinomsko** in **eksponentno** časovno kompleksnost

Analogni princip:
- **zvezno** predstavljena števila
- **omejena** natančnost

Von Neumanov računalnik:
- → centralno procesna enota: **CPE**
- → glavni pomnilnik: **GP**
- → vhodno izhodni sistem: **V / I**
- → **CPE** jemlje **ukaze** programa iz **GP** in jih zaporedoma **izvršuje**

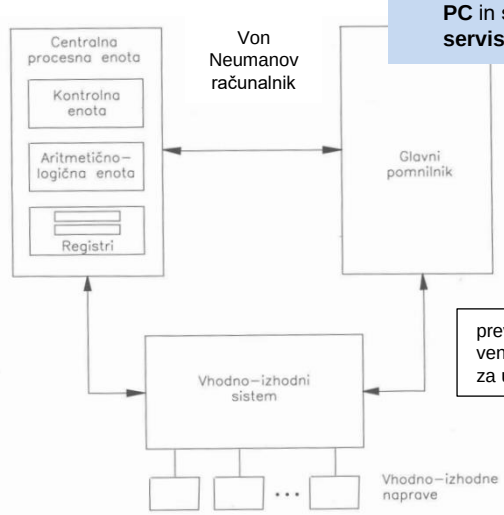


Ukazi:
- → ukaz v eni ali več **pomnilniških besedah**
- → ukaz ima **operacijsko kodo** in informacijo **operandih**
- → format ukaza pove kaj je kaj
- → naslov prvega ukaza pri vklopu računalnika **vnaprej določen**
- → **fetch**: prevzem ukaza iz pomnilnika, strojni ukazi se berejo iz besede v pomnilniku pri **programskem števcu**
- → **execute**: **CPE** izvrši ukaz po navadi **ALE, PC** se spremeni po navadi za 1
- → **prekinitve** in **pasti**: shrani se vrednost **PC** in se zažene **PSP prekinitveni servisni progam**

luknjaste kartice:
- **ukazne**: kartice z programi
- **operandne** kartice

računski del:
- **mlin**: izvedba operacij
- **pomnilnik**: shranjuje operande

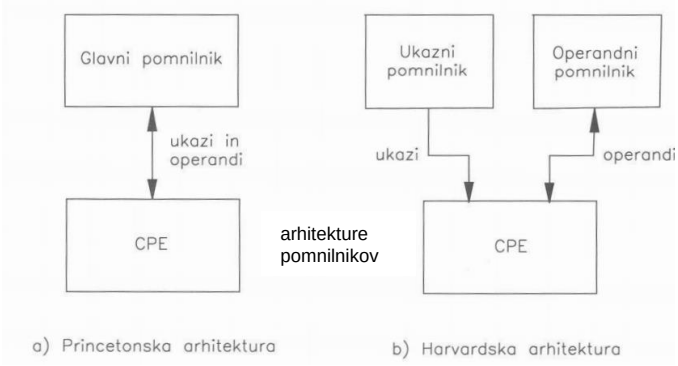
QUICK HISTORY RECAP
- → obdobje **mehanike**: prvi kalkulatorji z osnovnimi operacijami
- → **Charles Babbage**: primitivna tehnologija stroji podobni današnjim računalnikom
- → **diferenčni stroj**: aproksimacija funkcij z polinomi zaporedje fiksnih operacij
- → **analitični stroj**: prvi računalnik ni bil realiziran zaradi stroškov
- → **elektromehanski stroji**: elektromotorji za pogon in električno branje kartic, pridobitev je **rele** električno krmiljeno stikalo
- → **Zuse**: zgradi prvi računalnik, **Z3** prvi delujoči splošnonamenski računalnik
- → zastarelost mehanike uporaba **elektronike**
- → brez releja elektroni so hitrejši, **elektronka** kot vakuumska cev
- → **ENIAC**: dolgo programiranje vsi programi na karticah
- → **EDVAC**: **Von Neumanov** model kjer so programi shranjeni na računalniku, stroj je voden od znotraj
- → dostop do **ukazov** enako hiter kot do **operandov**
- → **tranzistor**: ojačevalnik in stikalo
- → **integrirana** vezja: **Moorov zakon**, podvojitev števila tranzistorjev na čipu vsakih 18 mesecev
- → razvoj **programskih jezikov**
- → jezik **Assembly** in program **Assebler**

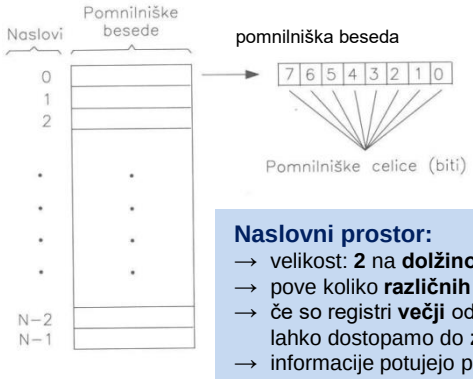


Glavni pomnilnik:
- → **pasiven**: shranjevanje ukazov in operandov
- → **ozko grlo** prenosa podatkov
- → **Harvardska** in **Princetonska** arhitektura

prevladuje **Princetonska** arhitektura, vendar z ločenima **predpomnilnikoma** za **ukaze** in **operande**

Pomnilniške besede:
- → **GP** je zaporedje teh
- → **dolžina**: št bitov
- → vsaka ima svoj **naslov**



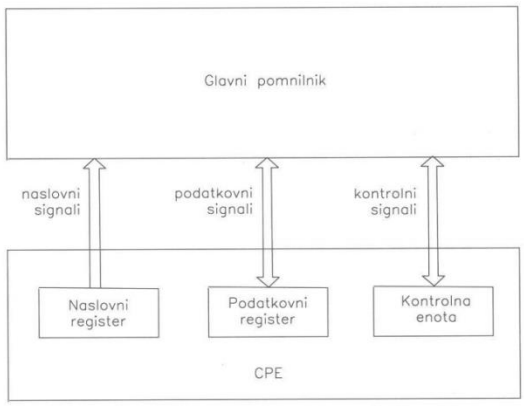


iz pomnilnika ni možno prebrati ali pisati **manj** kot eno besedo

večina besed **1 byte**

Naslovni prostor:
- → velikost: **2** na **dolžino naslova** v bitih
- → pove koliko **različnih** vsebin lahko hranimo
- → če so registri **večji** od pomnilniških besed lahko dostopamo do **zaporednih** besed
- → informacije potujejo po **vodilih**

CPE:
- → vsebuje tudi **naslovni register** oz. **MAR**: vsebuje naslov pomnilniške besede do katere želimo dostopiti
- → vsebuje tudi **podatkovni register** oz. **MDR**: pri branju iz **GP** se zapiše **prebrana** vrednost oz pri pisanju **izračunana** vrednost
- → povezana pomnilnikom preko **vodil**



MAR:
- → dolžina enaka **dolžini naslova**
- → isto **dolžina PC**
- → če naslovni prostor **premajhen** to problem

MDR:
- → število **bitov** ki se lahko naenkrat prenesejo med **CPE** in **GP**
- → enaka **večkratniku** dolžine pomnilniške besede
- → ni problem povečati
- → vpliva na **število dostopov** za operand določene velikosti

ARITMETIKA

Semantični prepad:
- → iz vsebine **pomnilniške besede** ni mogoče vedeti ali gre za **ukaz** ali **operand**
- → **CPE** ne more zaznati nesmiselnih operacij
- → to je razlika med opisom v **višjem** in v **strojnem** jeziku

Nenumerični operandi

- → vsak znak predstavljen z **abecedo**

Abecede:
- → **BCDIC**: **6** bitna
- → **EBCDIC**: **8** bitna razširjena
- → **ASCII**: originalno **7** bitna razširjena **8** bitna
- → **33** kontrolnih znakov in **95** printable znakov
- → **BCD**: 4 bitna, spodnji 4 biti se v vseh abecedah ujemajo
- → **Unicode**: prvih 128 znakov enakih **ASCII**

Numerični operandi

- → zapis v **fiksni** vejici
- → zapis v **plavajoči** vejici

FIKSNA VEJICA:
- → vsaka pozicija ima svojo **težo**
- → **utež** v obliki $\mathbf{r}^{\mathbf{i}}$
- → **r** je **baza** oziroma **radix**
- → v računalnikih baza **2**
- → **algoritem** za pretvorbo **!**
- → uporabljamo tudi **šestnajstiški** in **osmiški**
- → **sorodne baze** lažja pretvorba

$$V = \sum_{i=-m}^{n-1} b_i r^i \qquad V(b) = \sum_{i=-m}^{n-1} b_i 2^i$$

Napaka pri rezanju decimalk:
- → napaka **N' – N**
- → absolutna napaka | **N' – N** |
- → število **N** odrežemo na **k** decimalk
- → ne more biti večja od $\mathbf{r^{-k}}$
- → iščemo **k**
- → če ne znamo izračunati pretvorimo na bazo **10** ali pa **e**

$$r^{-k} \le E_{max} \qquad \log_b(a) = \frac{\log_x(a)}{\log_x(b)}$$

Nepredznačena števila:
- → z **n biti** lahko zapišemo **nepredznačena** števila od **0** do $\mathbf{2^n - 1}$ v kateremkoli formatu
- → kadar dosežemo obseg: **carry** oz **prenos**
- → rezultat **nepravilen**

$$V(b) = \sum_{i=0}^{n-1} b_i 2^i - b_{n-1} 2^n$$

Preliv:
- → ni isto kot **carry**
- → pri **2'K** se carry **ignorira**
- → zgodi se samo kadar **števili istega predznaka**
- → če predznak drugačen **OF**

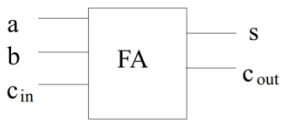
$$-2^{n-1} \le x \le 2^{n-1} - 1$$

Predznačena števila:
- → zapišemo na več načinov
- → **PV**: prvi **bit predznak** ostali število, problem ker dve ničli in predznak moramo obravnavati posebej
- → **PO**: zapis z **odmikom**, pri seštevanju moramo odšteti odmik pri odštevanju pa prišteti
- → **1'K**: prvi bit je **predznak**, negativno število dobimo z negiranjem vseh bitov, problem ker imamo dve ničli in kadar carry moramo na najnižjem mestu prišteti 1, **end around carry**
- → **2'K**: isto nima slabosti kot prej

b_2	b_1	b_0	PV	PO	1'K	2'K
0	0	0	+0	-4	+0	0
0	0	1	1	-3	1	1
0	1	0	2	-2	2	2
0	1	1	3	-1	3	3
1	0	0	-0	0	-3	-4
1	0	1	-1	1	-2	-3
1	1	0	-2	2	-1	-2
1	1	1	-3	3	-0	-1

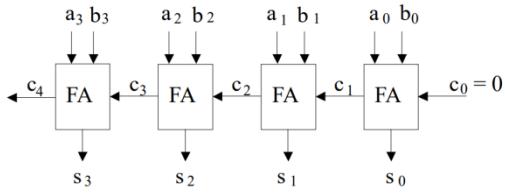
ARITMETIČNA VEZJA:
1. **polovični** seštevalnik
2. **polni** seštevalnik
3. **večbitni** seštevalnik
4. seštevalnik z **vnaprejšnjim prenosom**
5. matrični **množilnik**

Polni seštevalnik (Full Adder, FA)
- sešteva 3 bite, izračuna vsoto in (izhodni) prenos



$$s = a \oplus b \oplus c_{in} \; (= a'b'c_{in} \vee a'b\,c_{in}' \vee a\,b'c_{in}' \vee a\,b\,c_{in})$$
$$c_{out} = a\,b \vee a\,c_{in} \vee b\,c_{in}$$

Večbitni seštevalnik



Seštevalnik z razširjanjem prenosa: zaporedna vezava 1 bitnih **polnih seštevalnikov**. Problem je zakasnitev kjer maksimalna zakasnitev **narašča linearno.**

$$s = a \oplus b \oplus c_{in}$$
$$c_{out} = a\,b \vee a\,c_{in} \vee b\,c_{in}$$

Seštevalnik z vnaprejšnjim prenosom: izračuna **samo prenose** na osnovi a b in prvim c.

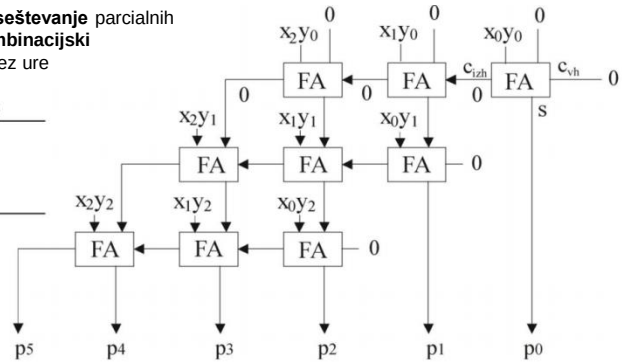
nekateri seštevalniki so odveč. Množenje v **2'K** je po **Boothovem algoritmu**. Delimo z odštevanjem in pomikanjem.

problemi pri aritmetiki v računalniškem sistemu:
- → **preliv**: postavitev posebnega bita in sprožitev pasti
- → **dolžina produkta**
- → **množenje** in **deljenje** zahtevnejši operaciji

Binarno množenje

imamo 2 metodi **pomikanje** in **seštevanje** parcialnih vsot kjer je 1 bit cikel ure in **kombinacijski množilniki** ki so dragi hitri in brez ure

```
x2   x1   x0   ×   y2   y1   y0
x2y2 x1y2 x0y2
     x2y1 x1y1 x0y1
          x2y0 x1y0 x0y0
```



	P	B	
0	0000	0110	začetek
1	0000	0110	P ← P + 0
	0000	0011	P,B >> 1
2	0101	0011	P ← P + A
	0010	1001	P,B >> 1
3	0111	1001	P ← P + A
	0011	1100	P,B >> 1
4	0011	1100	P ← P + 0
	0001	1110	P,B >> 1

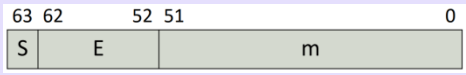
PLAVAJOČA VEJICA:
- → potrebujemo zelo velika in zelo majhna števila
- → zapišemo z **mantiso bazo** in **eksponentom**
- → s spreminjanjem eksponenta se vejica premika po mantisi
- → možnih števil še vedno **2** na **n**
- → **eksponent** predstavljen z odmikom
- → **normirana števila** prvi bit je 1
- → **denormirana števila** prvi bit je 0
- → standard **IEEE 754**

Denormirana števila:
- → kadar je **E** = **0**
- → eksponent v enojni natančnosti je -**126**
- → eksponent v dvojni natančnosti je -**1022**
- → **neskončnost**: ko je **E** = **255** oz **2047** in mantisa enaka **0**
- → **NaN**: ko **mantisa** ni **0**

ENOJNA NATANČNOST: 32 bitov

- → predznak **S**
- → **8** biten eksponent **E** = **e** + **127**
- → **23** bitna **mantisa**
- → normirana vrednost odmika od **1** do **254**

DVOJNA NATANČNOST: 64 bitov



- → predznak **S**
- → **11** biten eksponent **E** = **e** + **1023**
- → **52** bitna **mantisa**
- → normirana vrednost odmika od **1** do **2046**

Seštevanje:
- → prvo število z večjim **eksponentom**
- → **pomik mantise** drugega števila
- → če izpadejo **enice** se shranijo v **lepljivem** bitu
- → **seštejemo** oz **odštejemo** mantise
- → če se pojavi prenos naprej **zmanjšamo mantiso** in povečamo **eksponent** za **1**
- → **zaokrožimo mantiso**: če varovalni bit **1** in zaokroževalni in lepljivi **0** potem k **sodemu** številu, če **zadnji** bit mantise **0** ga pustimo če **1** pa **zaokrožimo navzgor**

Zaokroževanje:
- → zakroži se k **sodemu** številu
- → mantiso podaljšamo za **3** dodatne bite
- → **varovalni** bit: če vsota za eno mesto daljša od operandov
- → **zaokroževalni** bit: natančno zaokroževanje
- → **lepljivi** bit: da se iz izpadlih bitov vidi če je kakšen različen od **0** da zaokrožimo **navzgor** in ne k sodemu številu

Množenje:
- → eksponenta seštejemo
- → **mantisi zmnožimo** v fiksni vejici
- → po potrebi **normiramo** rezultat
- → predznak rezultata je **xor** obeh predznakov
- → deljenje je **odštevanje** eksponentov in **deljenje mantis**

UKAZI

- → informacija o operaciji: **operacijska koda**
- → **informacija** o **operandih**
- → shranjen v eni ali več **pomnilniških besed**
- → **format** pove kako so razdeljene operacije

Dimenzija
1. Način shranjevanja operandov v CPE
2. Število eksplicitnih operandov v ukazu
3. Lokacija operandov in načini naslavljanja
4. Operacije
5. Vrsta in dolžina operandov

Programsko dostopni:
- → majhen pomnilnik
- → shranimo operande
- → hitrejši od GP
- → krajši naslovi
- → več registrov dostopnih na enkrat

načini shranjevanja podatkov:

1. **akumulator**
- **najstarejši** način
- **edini** register
- v ukazih ni treba navajati **registra**
- ukaza **LOAD** in **STORE**
- veliko prometa v **GP** zato zelo **počasen**

2. **sklad**
- v trenutku **dostopna** samo **najvišja** lokacija
- **PUSH POP** ali **PULL**
- prostor za **več operandov**
- naenkrat dostopen samo **eden**

3. **register set**
- danes **edina** rešitev
- skupina **pomnilniških** celic s skupnimi **krmilnimi signali**
- vsak register **svoj naslov**
- shranjevanje **vmesnih rezultatov**
- splošnonamenski registri ali pa posebej za naslove in operande
- programsko **dostopni** in **nedostopni**

število ukazov:
- → m **operandni** računalnik
- → danes največ **3**

OP3 ← OP2 + OP1
OP2 ← OP2 + OP1
AC ← AC + OP1

operandi v **registrih**

enostavnejši in **počasnejši**

$Sklad_{VRH} \leftarrow Sklad_{VRH} + Sklad_{VRH-1}$

3 operandni, 2 operandni, 1 operandni in brez operandni

lokacija operandov:
→ registri **CPE** in **GP**
→ registri vhodno izhodnih sistemov
→ **registrsko registrski** računalniki
→ **registrsko pomnilniški** računalniki
→ **pomnilniško pomnilniški** računalniki

→ najbolj **razširjeni**
→ vsi **operandi** v registrih **CPE**
→ **load** in **store** ukazi

→ en **operand** v registru
→ drugi lahko v **pomnilniku**

→ vsi lahko v **pomnilniku**
→ **kompliciran**i ukazi

načini naslavljanja:
→ kako podana **informacija** o **podatkih**
→ pomembno pri **pomnilniških** operandih

takojšnje naslavljanje: operand v ukazu podan z vrednostjo

neposredno naslavljanje: operand je podan z naslovom, če naslov registra je **registrsko** naslavljanje, če naslov v **GP** je to **pomnilniško** naslavljanje

težave: velik naslovni prostor pomeni dolge naslove in dolge ukaze če povečamo pomnilniški prostor ukazi nezdružljivi

relativno naslavljanje:
→ **bazno**: **A = R2 + D**

- naslavljanje z **odmikom**
- najpogostejše
- **R2** je bazni register **D** je odmik
- **A** je dejanski naslov

→ **indeksno**: **A = R2 + R3 + D**

- R3 je **indeksni register**
- polja strukture in **seznami**

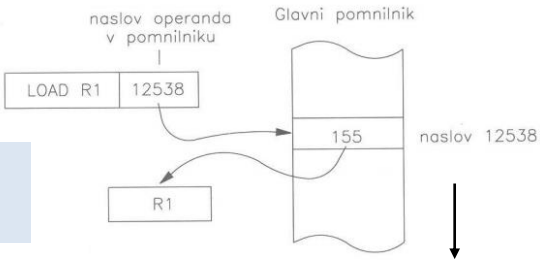
→ **pred dekramentno**: **R3 ← R3 – Δ**
→ **po inkeremntno**: **R3 ← R3 + Δ**
→ **velikostno indeksno**:

A = R2 + R3 · Δ + D

→ **skladovno**: v paru pred in po !

pozicijsko naslavljanje:
→ **pozicijsko neodvisni** programi

PC relativno naslavljanje:
→ **bazni** register služi kot **programski števec**



operacije:
→ **računalnik** lahko samo **en ukaz**:

M[A] ← M[A] - M[B]; če M[A] < 0, skoči na C

→ operacij manj kot ukazov
→ imena ukazov: **mnemoniki**

Skupine operacij

aritmetične in logične operacije:
→ izvajajo se v **ALE**
→ operandi v **fiksni vejici**
→ **aritmetične operacije**: seštevanje, odštevanje, množenje, deljenje, negacija, absolutna vrednost…
→ **logične operacije**: and, or, not, xor, pomiki

posredno naslavljanje: v ukazu naslov lokacije na kateri je naslov operanda**, pomnilniško** se ne uporablja pogosto in **registrsko** kjer se uporablja **odmik** rečemo tudi **relativno naslavljanje**.

Prenosi podatkov:
→ **load**, **store**, **push**, **pull**, **pop**

kontrolne operacije:
→ spreminjajo **vrstni red** ukazov
→ **pogojni** skoki
→ **brezpogojni** skoki
→ klici in vrnitve iz **podprogramov**

- **pogojni register**: gledamo če njegova vsebina enaka 0
- **pogojni bit**: rezultat določenih operacij npr overflow
- **primerjaj** in **skoči**: skok če primerjava izpolnjena

delimo tudi na **skalarne** in **vektorske** pri katerih se ukaz izvrši na **več operandih**. Pri skalarnih računalnikih za to uporabimo **zanko**.

Vrste operandov

operacije v plavajoči vejici:
→ **FPU** ni del **ALE**
→ **koren**, **logaritem**, **eksponentna** funkcija in **trigonometrične** funkcije

sistemske operacije:
→ vplivajo na **način delovanja** računalnika
→ **privilegirani** ukazi

znak:
→ običajno **8 bitov**
→ običajno **ASCII**
→ več **znakov** v niz

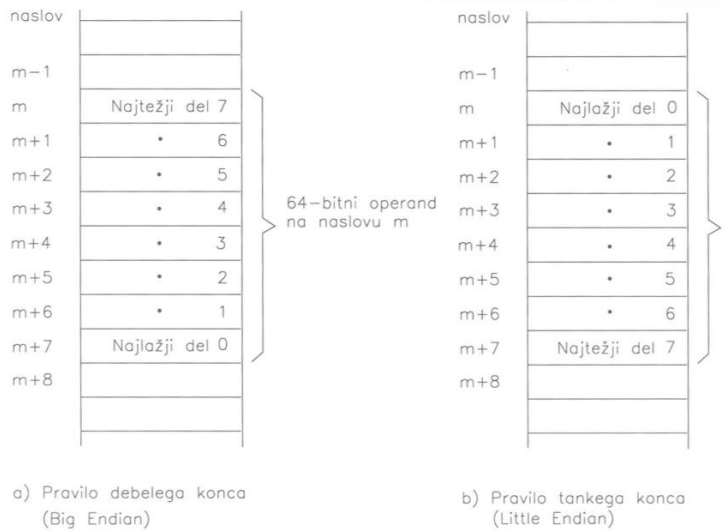
bit:
→ ni v **višjih** jezikih
→ koristno v **sistemskih** operacijah

vhodno izhodne operacije:
→ obstajajo ponekod
→ prenosi med **GP** in **V / I**
→ prenosi med **CPE** in **V / I**

celo število:
→ **predznačeno** ali **nepredznačeno**
→ dolžine **8**, **16**, **32**, **64** bitov

realno število:
→ v **plavajoči** vejici

desetiško število:
→ v **8** bitih
→ **2 BCD** števili ali **1 ASCII znak**



→ **8** bitov: **byte**
→ **16** bitov: **halfword**
→ **32** bitov: **word**
→ **64** bitov: **double word**
→ **128** bitov: **quad** word

problem poravnanosti: pomnilnik omogoča dostop do **8** osem bitnih besed naenkrat je narejen ko 8 **paralelnih pomnilnikov**

istočasen dostop do **s** besed **dolgega operanda** na naslovu **A** je možen le če je **A** deljiv z **s**.

→ **sestavljeni pomnilniški operandi**: iz več pomnilniških besed ki so v pomnilniku na zaporednih lokacijah
→ **big endian rule**: **najtežji** del operanda na **najnižjem** naslovu
→ **little endian rule**: **najlažji** del operanda na **najnižjem** naslovu

HIP

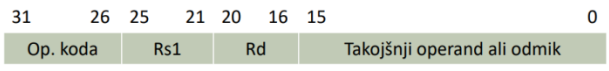
Ukazna arhitektura:

→ Instruction Set Architecture: **ISA**
→ definira vse **ukaze procesorja**
→ ne govori o **implementaciji**
→ **HIP** poenostavljen procesor **MIPS**

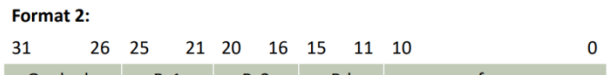
→ **8** bitna **pomnilniška beseda**
→ **32** bitni **pomnilniški** naslov
→ dostop do predpomnilnika pri **zadetku** traja **en cikel ure**
→ dostop do predpomnilnika pri **zgrešitvi** traja **11 ciklov**
→ pomnilniško preslikan vhod / izhod

→ način **shranjevanja** operandov v **CPE**: **32** 32 bitnih splošno namenskih **registrov**
→ **r0** rezerviran za **vrednost 0**
→ vsi ukazi imajo **3 operande**
→ pri dveh se tretji ignorira **!**
→ **load** / **store** računalnik: dostop do operandov v pomnilniku
→ pri **ALE** ukazih **2 operanda** v **registrih**
→ tretji v registru ali takojšnji
→ **takojšnje** ali **bazno** naslavljanje
→ takojšnji operand **16** biten
→ bazno: **16** bitni **2'K** odmik
→ **big endian rule**
→ neporavnanost sproži **past**
→ vse operacije so **32 bitne**
→ razširitev pri **load** ukazih **signed** / **unsigned**
→ vse **ALE** operacije v eni urini periodi
→ **2 formata** ukazov

Format 1:



Format 2:



če razširimo besedo **unsigned** jo razširimo z ničlami in če **signed** z najtežjim bitom

- če **30** bit enak **1** potem **bazno** naslavljanje drugače **takojšnje**
- v drugem formatu func **razširja** operacijsko kodo
- ni ukazov za množenje in deljenje
- ni ukazov v plavajoči vejici

ukazi za prenos podatkov:

Format	Op. koda	Ukaz	Opis
1	100000	LBU	Load byte unsigned
1	100001	LHU	Load halfword unsigned
1	100100	LB	Load byte
1	100101	LH	Load halfword
1	100110	LW	Load word
1	101000	SB	Store byte
1	101001	SH	Store halfword
1	101010	SW	Store word

psevdo ukazi:

.data	– začetek podatkovnega segmenta
.text, .code	– začetek ukaznega segmenta
.org <n>	– določen začetni naslov
.space <n>	– rezerviraj n bajtov prostora (naključne vred.)
.word <n1>,<n2>..	– določi zaporedna 32-bitna števila
.word16 <n1>,<n2>..	– določi zaporedna 16-bitna števila
.byte <n1>,<n2>..	– določi zaporedna 8-bitna števila
.align <n>	– poravnaj naslov, da bo deljiv z n

load high immediate: če hočemo v register naložiti **32** bitno **vrednost** predolga za 32 bitni ukaz moramo z **lhu**

kontrolni ukazi:
spremembo vrstnega reda izvajanja ukazov tem ukazom rečemo **skoki**. Imamo **brezpogojne** in **pogojne** skoke. Omogočajo **if** stavke in **zanke**.

Format	Op. koda	Ukaz	Opis
1	100011	BNE	Branch if Rd not equal to zero
1	100111	BEQ	Branch if Rd equal to zero
1	101100	J	Jump
1	101101	CALL	Jump to subroutine
1	101110	TRAP	Jump to vector address
1	101111	RFE	Return from exception

ALE ukazi:

Format	Op. koda	func	Ukaz	Opis	Tip operacije
2	110000	0	ADD	Add	Aritmetične
2	110001	0	SUB	Subtract	
2	110010	0	ADDU	Add unsigned	
2	110011	0	SUBU	Subtract unsigned	
2	110100	0	AND	And	Logične
2	110101	0	OR	Or	
2	110110	0	XOR	Exclusive or	
2	111110	0	NOT	Not (1's complement)	
1	000000	x	ADDI	Add immediate	Aritmetične takojšnje
1	000001	x	SUBI	Subtract immediate	
1	000010	x	ADDUI	Add unsigned imm.	
1	000011	x	SUBUI	Sub. unsigned imm.	
1	000100	x	ANDI	And immediate	Logične takojšnje
1	000101	x	ORI	Or immediate	
1	000110	x	XORI	Exclusive or imm.	
2	111000	0	SEQ	Set if equal	set
2	111001	0	SNE	Set if not equal	set
2	111010	0	SLT	Set if less than	set
2	111011	0	SGT	Set if greater than	set
2	111100	0	SLTU	Set if less than unsigned	set
2	111101	0	SGTU	Set if greater than unsigned	set
1	001000	x	SEQI	Set if equal immediate	set imm.
1	001001	x	SNEI	Set if not equal immediate	set imm.
1	001010	x	SLTI	Set if less than immediate	set imm.
1	001011	x	SGTI	Set if greater than imm.	set imm.
1	001100	x	SLTUI	Set if less than unsig. imm.	set imm.
1	001101	x	SGTUI	Set if greater than uns. imm.	set imm.
2	110000	1	SLL	Shift left logical	shift
2	110001	1	SRL	Shift right logical	shift
2	110010	1	SRA	Shift right arithmetic	shift
1	010000	x	SLLI	Shift left logical immediate	shift imm.
1	010001	x	SRLI	Shift right logical immediate	shift imm.
1	010010	x	SRAI	Shift right arithmetic imm.	shift imm.
1	000111	x	LHI	Load high immediate	

sistemski ukazi:

Format	Op. koda	func	Ukaz	Opis
2	110011	1	EI	Enable interrupts
2	110100	1	DI	Disable interrupts
2	110101	1	MOVER	Move from EPC to register
2	110110	1	MOVRE	Move from register to EPC

Zgradba ukazov:

→ na **zgradbo** ukaza **vplivajo**:

- **dolžina pomnilniške besede**: večkratnik
- pri dolgih besedah polovica ali četrtina besede
- pri osem bitnih besedah večkratnik 8
- **število eksplicitnih operandov**
- vrsta in število **registrov** v CPE
- **dolžina pomnilniškega naslova**
- število **operacij**

ortogonalnost ukazov: informacija o operaciji oz. operandu neodvisna od informaciji o operandih

→ **optimalne rešitve** za format ukazov ni imamo **3 načine**

- **spremenljiva** dolžina ukazov
- **fiksna** dolžina ukazov
- **hibridni** način

računalniki:
CISC računalniki imajo **veliko** ukazov **RISC** računalniki **majhno** število ukazov npr MIPS. Drugi so **hitrejši**. Zmanjšanje zaradi pojava **predpomnilnikov**, težav prevajalnikov in uvajanje paralelizma v **CPE** npr. **cevovod**

RISC:
večina ukazov v enem **ciklu CPE**. **Load** / **store** zasnova. Trdo ožičena logika, ne **mikroprogramerska**. Malo ukazov in načinov naslavljanja. Enaka **dolžina** ukazov. Dobri prevajalniki.

Sklad in delo z podprogrami:

→ linearni seznam organiziran v smislu **LIFO**
→ potrebujemo skladovni kazalec: **SP**
→ uporabljamo operaciji **PUSH** in **POP**
→ shranjevanje začasnih **spremenljivk**
→ prenos parametrov v **podprograme**
→ shranjevanje **registrov** v podprogramih
→ shranjevanje **povratnega naslova** pri podprogramih

→ **SP** kaže na **prvo** prosto mesto
→ **sklad** v smeri **naraščajočih** naslovov

```
PUSH reg :  M[SP] <- reg; SP <- SP + n;
POP reg :   SP <- SP - n; reg <- M[SP];
```

→ **SP** kaže na **zadnji** podatek na skladu
→ **sklad** v smeri **naraščajočih** naslovov

```
PUSH reg :  SP <- SP + n; M[SP] <- reg;
POP reg :   reg <- M[SP]; SP <- SP - n;
```

→ **SP** kaže na **prvo** prosto mesto
→ **sklad** v smeri **padajočih** naslovov

```
PUSH reg :  M[SP] <- reg; SP <- SP - n;
POP reg :   SP <- SP + n; reg <- M[SP];
```

→ **SP** kaže na **zadnji** podatek na skladu
→ **sklad** v smeri **padajočih** naslovov

```
PUSH reg :  SP <- SP - n; M[SP] <- reg;
POP reg :   reg <- M[SP]; SP <- SP + n;
```

Klicani podprogram ob vstopu:

1. na sklad porine povratni naslov - register `R31`;
2. na sklad porine kazalec na okvir - register `R29`;
3. v register `R29` (kazalec na okvir) prepiše skladovni kazalec;
4. po potrebi rezervira prostor na skladu za lokalne spremenljivke (spreminja skladovni kazalec!);
5. na sklad **shrani vse registre, ki jih spreminja**;
6. do prenešenih parametrov in lokalnih spremenljivk dostopamo preko kazalca na okvir (register `R29`): zadnji parameter na skladu je na naslovu `R29+12`, prva lokalna spremenljivka je na naslovu `R29`.

Klicani podprogram pred izstopom:

1. v register `R28` zapiše vrednost, ki jo vrača;
2. s sklada obnovi vse shranjene registre;
3. s sklada 'odstrani' lokalne spremenljivke tako, da register `R29` prepiše v skladovni kazalec `R30`;
4. s sklada obnovi (prebere) staro vrednost registra `R29`;
5. s sklada obnovi (prebere) povratni naslov v register `R31`;
6. z ukazom `jmp` se vrne na naslov `R31+0`.

→ **HIP** nima skladovnega kazalca po dogovoru r30
→ **sklad** v smeri **padajočih** naslovov
→ **SP** kaže na **prvo** prosto mesto
→ sklad naj se zacne **na dnu** podatkovnega segmenta
→ samo operaciji **PUSH** in **POP** ki nista zares lol

```
PUSH reg :  sw 0(r30), reg
            subui r30,r30,#4
POP reg :
            addui r30,r30,#4
            lw reg, 0(r30)
```

klic **podprograma**

```
call rp, odmik(rb)
```

→ prenos parametrov: prva dva **r24**, **r25**, ostale na sklad
→ parametri se prenašajo z **leve proti desni**
→ povratni naslov shrani v register **rb** (**r31**)
→ v **PC** se vpiše naslov podprograma **rb** + **odmik**
→ za **kratke** pod klice register **r0**
→ za **dolge** pod klice register **r27**
→ za nazaj rabimo **j odmik** (**r31**) oz **r26** za dolg skok

```
lhi   r26, #SKOK
addui r26, r26, #SKOK
j     0(r26)
```

```
lhi   r27, #PODPROG
addui r27, r27, #PODPROG
call  r31, 0(r27)
```

→ do parametrov dostopamo preko **frame pointerja r29**
→ tako vemo kje so parametri na skladu
→ iz podprograma **vračamo** 1 vrednost v registru **r28**

Register	Namen uporabe
R0	Ničla
R1-R23	Splošno namenski registri
R24	Prvi parameter z leve, ki se prenese v podprogram
R25	Drugi parameter z leve, ki se prenese v podprogram
R26	Bazni register za dolge skoke
R27	Bazni register za dolge klice
R28	Vrednost, ki jo vrača podprogram
R29	Kazalec na okvir
R30	Skladovni kazalec
R31	Povratni naslov

CENTRALNO PROCESNA ENOTA

→ **CPE** je **digitalen** sistem
→ **kombinacijska** in **sekvenčna** digitalna vezja
→ odvisno od trenutnega **stanja** in **vhodov**

delovanje CPE:

→ dobava ukaza iz pomnilnika: **fetch**
→ potem **izvrševanje** ukaza:

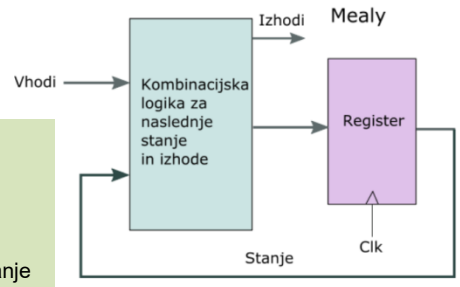
- **dekodiranje** ukaza
- **prenos** operandov v CPE
- **izvedba** operacije
- po potrebi **shranjevanje** rezultata
- **PC** gre v **PC** + **1**
- pri **skokih** zadnje ne velja

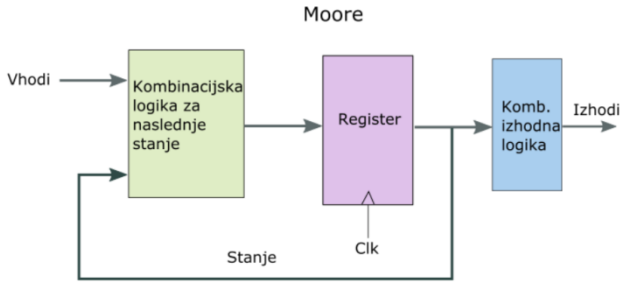
→ ta cikel se **ponavlja** cel čas
→ izjema so **pasti** in **prekinitve**
→ vsak od korakov iz bolj elementarnih
→ pri **sinhronih** sekvenčnih vezjih se spremembe stanja prožijo ob **aktivni fronti** ure
→ vsak traja eno ali več **period ure** CPE, perioda ure je čas med **sosednima frontama** t_{CPE} $f_{CPE} = 1/t_{CPE}$
→ perioda je **navzdol omejena** z zakasnitvami kombinacijskih vezij
→ frekvenca **omejena navzgor**
→ **asinhronska** sekvenčna vezja: ni ure

podatkovna enota:

→ **kontrolna** enota: **KE**
→ **podatkovna** enota: **PE**

→ **KE** vodi delovanje, kontrolni signali
→ **PE** vodi **ALE** in registre
→ ima **3** **32** bitna **vodila**
→ **S1** in **S2** vhod v **ALE** in **D** izhod
→ vse **ALE** operacije v **1** urini **periodi**
→ **32** bitni konstanti na **S2** za povečevanje **PC** in **FFFFFF00** za izjeme





Registrski blok

→ **register file**: registri od **r0** do **r31**
→ priključen na vodila preko izhodnih registrov **A** in **B** ter vhodnega registra **C**
→ če bi priključili direktno bi dobili velike **zakasnitve**
→ na register **B** je priključena **logika** za **detekcijo ničle**

→ **kontrolni** signali za registrski blok:
→ **RegtoA** izbere register ki se bo **prebral** v **A**
→ **RegtoB** izbere register ki se bo **prebral** v **B**
→ **CtoReg** določa register kamor se bo **vpisala** vsebina **C**
→ pisanje ob **aktivni fronti** ure

MAR (Memory address register) in MDR (Memory data register) služita komunikaciji s pomnilnikom

- MDR je vezan na S1 in na pomnilniške podatkovne signale D0-D31
  - MDRtoS1
  - MemWrite
- Pisanje v MDR (MDRWrite). Z MDRSelect (2 bita) izberemo vhod
  - pomnilnik (D0-D31)
  - vodilo D
  - register B
- MAR je vezan preko MUXa na pomnilniške naslove A0-A31
- MARWrite: D v MAR

prenos sprožita signala **RegRead** in **RegWrite**

ukazni register **IR** je v **kontrolni** enoti.

ALE

→ operacija se izvrši na 32 bitnih **vhodnih operandih x** in **y**
→ operanda sta na **vodilih S1** in **S2**

- Signali **AtoS1**, **BtoS2**, **PCtoS1**
- prenašajo vsebine registrov **A**, **B** in **PC**
- prenašajo na vodili **S1** in **S2**
- signali za pisanje v registre
- **Cwrite**, **PCWrite**, **MDRWrite**,…

ADRSelect:

- ADRSelect = 0: PC na naslovno vodilo
- ADRSelect = 1: MAR na naslovno vodilo

Za uporabnika sta vidna še programski števec PC in EPC

- PCtoS1 prepušča vsebino PC na vodilo S1
- PCWrite sproži pisanje iz vodi
- izhod PC je priključen tudi na pomnilniške naslovne signale

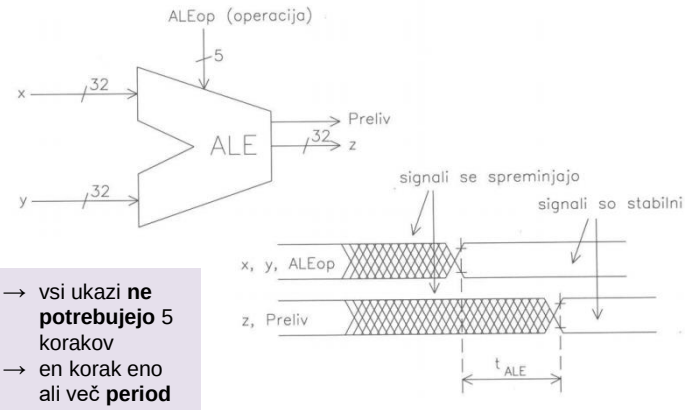
EPC shranjuje PC ob prekin exceptions)

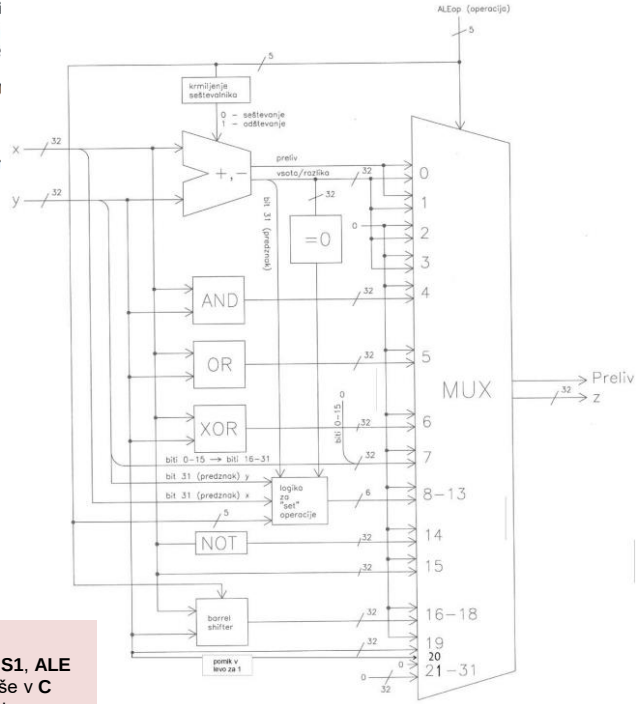
- EPCtoS1, EPCWrite
- 1-bitni register I je v kontrolr onemogočene)

Signali za dostop do pomnilnika

- z MemWrite in MemRead se izbere pisanje ali branje
- Size (2-bitni) pove, ali se prenaša 8, 16 ali 32 bitov
- SignX pri pretvorbah v 32 bitov:
  - SignX = 1: razširitev predznaka
  - SignX = 0: razširitev ničle
- MemWait: čakanje na pomnilnik pri zgrešitvi v predpomnilniku

→ **rezultat** se shrani na **z** pri vodilu **D**
→ **preliv** speljan na **kontrolno enoto**
→ signali **ALEop** iz





→ vsi ukazi **ne potrebujejo** 5 korakov
→ en korak eno ali več **period**

UKAZNI KORAKI:

1. **prevzem**: instruction fetch (**IF**)
2. **dekodiranje**: instruction decode (**ID**)
3. **izvrševanje** operacije: execute (**EX**)
4. **dostop** do pomnilnika: memory (**MEM**)
5. **shranjevanje** rezultata: write back (**WB**)

PREVZEM UKAZA

→ vsebina **PC** se preko **MUX** prenese na **pomnilniški** naslov
→ z pomnilnika se prebere **32 bitov** in prenese v **IR**
→ $IR \leftarrow_{32} M[PC]$

DEKODIRANJE UKAZA

→ povečanje **PC** za **4** izvaja **ALE**
→ na **S1** gre vsebina **PC**
→ na **S2** gre **konstanta** + **4**
→ **ALE** operacija **ADDU**
→ vsebina **D** gre v **PC**
→ korak traja **eno periodo**

$A \leftarrow Rs1;$
$B \leftarrow Rs2$ (ali Rd);
$PC \leftarrow PC + 4$

IZVRŠEVANJE

1. **load** / **store** ukazi
2. klic procedure **call**
3. **ALE** ukazi
4. ukaz **trap**
5. **brezpogojni** skok
6. **pogojni** skok
7. ukaz **RFE**
8. ukaza **MOVER** in **MOVRE**
9. ukaza **EI in DI**

1.

- z **IRtoS2** gre na **S2**.
- **A** (**Rs1**) gre na **S1** (**AtoS1**)
- **ALE** operacija **ADDU**
- rezultat **D** gre v **MAR**
- hkrati gre **B** v **MDR**
- porabi se **1 perioda**

$MAR \leftarrow A + \text{raz}(IR_{0..15})$;
$MDR \leftarrow B$

2.

- **2** periodi
- v prvi se **PC** preko **S1**, **ALE** (**S1** → **D**) in **D** zapiše v **C**
- v drugi se **vsota A** in odmika zapiše v **PC**

$C \leftarrow PC.$
$PC \leftarrow A + \text{raz}(IR_{0..15})$

3.

- **A** je **Rs1**
- **Rs2** je **B** (**format 2**) ali pa takojšnji operand IR (**format 1**)

$C \leftarrow A$ op B ali $C \leftarrow A$ op $\text{raz}(IR_{0..15})$
(glede na format (2 ali 1))

4.

- naslov **servisnega programa** je na **FFFFFF00** + **4** · **n**
- v prvi gre **PC** preko **S1**, **ALE** (**S1** → **D**) in **D** v **EPC**
- **0** pa gre v **I** da se onemogočijo **prekinitve**
- v drugi se sešteje **konstanta** in **IR** (številka vektorja **n**), pomaknjen za **2** v **levo**

$EPC \leftarrow PC;$ $I \leftarrow 0$
$MAR \leftarrow FFFFFF00 + 4 \times \text{raz}(IR_{0..15})$

5.

- zapis v **PC** z **PCWrite**
- **1** perioda

$PC \leftarrow A + \text{raz}(IR_{0..15})$

6.

- kot **pogoj** se uporablja vsebina **Rd** ki se je v koraku **2** prenesel v **B**
- na **B** je priključena **logika** za ugotavljanje **ničle**
- **1** perioda

BEQ: če je $B = 0$, potem $PC \leftarrow PC + \text{raz}(IR_{0..15})$
BNE: če je $B \neq 0$, potem $PC \leftarrow PC + \text{raz}(IR_{0..15})$
PCWriteCond0 piše v PC, če je $B = 0$
PCWriteCond1 piše v PC, če je $B \neq 0$
Logika z vrati AND in OR (na sliki) določa vpis v PC:

- PCWriteCond0 · (B=0) ∨ PCWriteCond1 · (B≠0) ∨ PCWrite

7.

- **1** perioda $PC \leftarrow EPC$
- za vrnitev iz **prekinitve** ali **pasti**

8.

- **MOVER** prenese vsebino **EPC** v **Rd**
- **MOVRE** prenese vsebino **Rs1** v **EPC**

MOVER: $C \leftarrow EPC$
MOVRE: $EPC \leftarrow A$

EI: $I \leftarrow 1$
DI: $I \leftarrow 0$

- omogočanje / onemogočanje **prekinitev**
- signala **Iset** in **Iclear** postavita register **I** na **1** oz. na **0**

9.

DOSTOP
- → naslov v **prejšnjem** koraku v **MAR**
- → 1 perioda + morebitne **čakalne** periode

ukazi store:	*M[MAR] ← MDR*
ukazi load in TRAP:	*MDR ← M[MAR]*

SHRANJEVANJE
- → **ALE** ukazi **CALL** in **MOVER**: **Rd ← C**
- → ukaz **TRAP**: **PC ← MDR**
- → ukaz **load**: **C ← MDR** in **Rd ← C**

Časi izvrševanja

- → odvisen od tega **katere korake** potrebujemo in koliko **zgrešitev** v pomnilniku
- → vsi ukazi potrebujejo **en dostop** do pomnilnika
- → **load store** in **trap** potrebujejo še enega
- → povprečno **število urinih period** je vsota **najmanjše število period** in povprečno število **čakalnih urinih period** ki nastanejo zaradi **zgrešitev**
- → **čakalne periode**: vsaka dolga 10
- → izračunamo: št. **čakalnih** period · **verjetnost** zgrešitve · št. **pom**. **dostopov**
- → **CPI** je **clocks per instruction**

Vrsta ukazov	Število pomnilniških dostopov	Najmanjše število urinih period na ukaz	Povprečno število urinih period na ukaz
Load	2	6	7
Store	2	4	5
TRAP	2	6	7
ALE	1	4	4,5
J, BEQ, BNE	1	3	3,5
CALL	1	5	5,5
MOVER	1	4	4,5
MOVRE, EI, DI	1	3	3,5

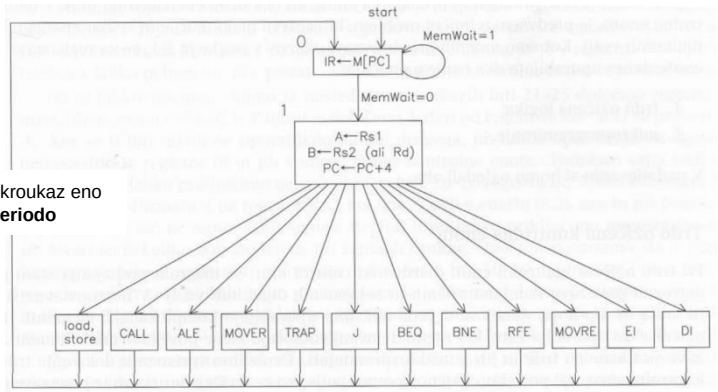
- → **CPI_i** je število **urinih period** za ukaz vrste **i**
- → **p_i** je **relativna verjetnost** posamezne vrste ukaza
- → če za **CPI_i** vzamemo najmanjše možno število urinih period dobimo idealni **CPI** ki ne vključuje zgrešitev
- → **MIPS**: Million Instructions Per Second

$$MIPS = \frac{f_{CPE}}{CPI \cdot 10^6} = \frac{1}{CPI \cdot t_{CPE} \cdot 10^6} \qquad CPI = \sum_{i=1}^{n} CPI_i \cdot p_i$$

- → na **MIPS** vpliva **frekvenca ure**

- → **kontrolna enota** je kompliciran
- → za vsak ukaz mora vedeti katere **signale aktivirati**
- → diagram prehajanja stanj: **DPS**
- → temu ustreza **končni aparat**
- → **CPE** lahko izvršuje ukaze na 2 načina

vsak mikroukaz eno **urino periodo**



48 0

ALEop	S1	S2	Wreg	R	Mux	Pom-nilnik	Next	Skočni naslov
5	5	4	10	2	3	5	6	9

format **mikroukaza** v HIPU

- → **mikroprogramiranje**
- → pri vsakem ukazu se aktivira ustrezno zaporedje **mikroukazov** oz. **mikroprogram**
- → shranjeni v **kontrolnem pomnilniku CPE**
- → mikroukazi so **primitivnejši** od običajnih
- → izvršuje **trdo ožičena logika**
- → počasnejše
- → bolj fleksibilno
- → kot majhen **računalnik**
- → shranjen v **ROM**
- → aktivirajo **kontrolne programe**

- → **horizontalno**: malo ali nič kodiranja, dolgi ukazi, drag
- → **vertikalno**: veliko kodiranja, veliko ukazov kratki, počasnejše, cenejše

- → **trdo ožičena logika**
- → **vezje**: logična vrata, pomnilne celice, povezave
- → spremembe so možne le s **fizičnim posegom**
- → vrata in flip flop
- → potrebujemo popoln **DPS**

ALEop:
- ALE operacija v tej urini periodi
  - tudi pri vseh ostalih poljih gre za trenutno urino periodo, zato tega ne bomo posebej omenjali

S1:
- določa, kateri register gre na vodilo S1
- Biti: AtoS1, EPCtoS1, PCtoS1, MDRtoS1, IR4toS1 (le en bit lahko 1)

S2:
- določa, kateri register ali konstanta gre na vodilo S2
- Biti: K4toS2, KFFtoS2, BtoS2, IRtoS2 (le en bit lahko 1)

Wreg:
- delovni register, v katerega se piše
- Biti: Cwrite, EPCWrite, PCWrite, PCWriteCond0, PCWriteCond1, MARWrite, MDRWrite, IRWrite, ISet, IClear

R:
- določa, ali se bere ali piše v registre R0-R31
- Bita: RegWrite, RegRead

Mux:
- določa krmiljenje obeh mux
- Biti: ADRSelect, MDRSelect

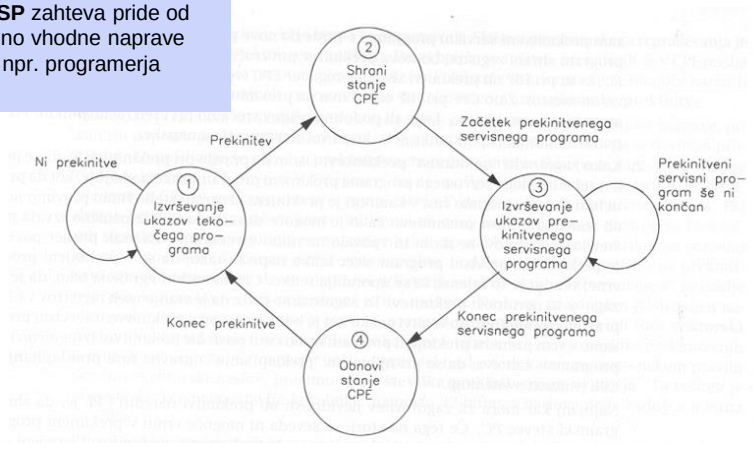
Pomnilnik:
- določa, kaj se dogaja s pomnilnikom
- Biti: MemRead, MemWrite, SignXCond, Size

Next:
- določa način izbire naslova naslednjega ukaza
- Biti: µNew, µJmp, µINT, µPreliv, µAligned, µMemWait
  - največ eden na 1

Skočni naslov:
- naslov, ki se bo zapisal v µPC, če tako določajo biti v Next
- Biti: naslov v mikroprog. pomnilniku

Prekinitve in pasti

- → **prekinitev**: zaženemo **PSP** zahteva pride od **zunaj** npr. od neke izhodno vhodne naprave
- → **pasti**: pridejo od **znotraj** npr. programerja
- → razlikujemo **4 stanja**



primeri uporabe:
- **zahteve** V / I naprav ob različnih dogodkih
- napaka v delovanju nekega **dela računalnika**
- **aritmetični preliv**
- **napaka** strani ali segmenta
- dostop do **zaščitene pomnilniške besede**
- dostop do **neporavnane** pomnilniške besede
- uporaba **nedefiniranega ukaza**
- klic programov operacijskega sistema

5 dejavnikov prekinitve:

→ kdaj **CPE** reagira na prekinitveno zahtevo: najenostavneje po izvrševanju **tekočega** ukaza, v tem primeru se mora **ohraniti** samo **stanje** programsko **dostopnih registrov**, programer lahko **onemogoči** odziv CPE na prekinitvene zahteve

→ **kako** zagotoviti nevidnost **nevidnost** prekinitev: treba je zagotoviti, da je stanje **registrov CPE enako** kot prej

→ **kje** dobimo naslov **PSP**: pomembno pri prekinitvah od zunaj, ugotoviti treba katera **naprava** je zahtevala prekinitev, **programsko izpraševanje** preverja **bit** ki pove če naprava sprožila prekinitev. obstajajo **vektorske prekinitve** izračuna iz številke **prekinitvenega vektorja** po nekem pravilu

→ **prioriteta**: vgnezdene prekinitve pri katerih zahteve z **višjo** prioriteto prekinejo prek. servisne programe z **nižjo** prioriteto. **prekinitveni krmilnik** omogoča računalnikom ki imajo CPE z enim samim bitom za omogočanje prekinitev fleksibilno obravnavo prioritete. lahko tudi z **marjetično verigo** kjer naprave ki **niso** sprožile prekinitve spustijo signal tista ki jo je pa ga ustavi.

→ **potrjevanje** prekinitve: potrebno da ne servisiramo prekinitve 2x. **programsko** tako da napišemo v register ali pa **strojno** z nekim signalom.

→ **HIP** ima en sam prekinitveni vhod **INT** in uporablja **vektorske** prekinitve
→ **CPE** se odzove na prekinitveno zahtevo s prekinitveno prevzemnim ciklom
→ vektorski naslovi deljivi z 4 od **FFFFFF00** naprej so **PSP**
→ **nedefiniran** ukaz, **preliv** ali pa **neporavnan** operand
→ ukaz **TRAP** ki je programska past
→ pri prekinitvi ali pasti se v **PC** naloži 32-bitni naslov, ki je shranjen v pomnilniku na vektorskem naslovu

Številka vektorja n	Vrsta pasti ali prekinitve	Vektorski naslov
0	nedefiniran ukaz	FFFF FF00$_H$
1	preliv	FFFF FF04$_H$
2	neporavnan operand	FFFF FF08$_H$
3-63	V/I naprave	FFFF FF00$_H$ + 4 × n

Zmogljivost CPE

→ ni enako kot zmogljivost računalnika
→ enako če **CPE** ne čaka **V / I** naprav in **pomnilnika**

$$CPE\ čas = Število\ ukazov\ programa \times CPI \times t_{CPE}$$

→ **CPE čas** odvisen od hitrosti in števila **digitalnih vezij**, pa tudi od **zgradbe** CPE
→ **CPI** odvisen od **zgradbe** CPE in **ukazov**
→ **MIFLOPS** ocena dobra za floating point

$$CPE\ čas = \frac{Število\ ukazov}{MIPS \cdot 10^6}$$

Vhodno izhodni sistem

→ pretvorba **informacije** iz ene oblike v drugo
→ izjema so naprave za **shranjevanje** informacije

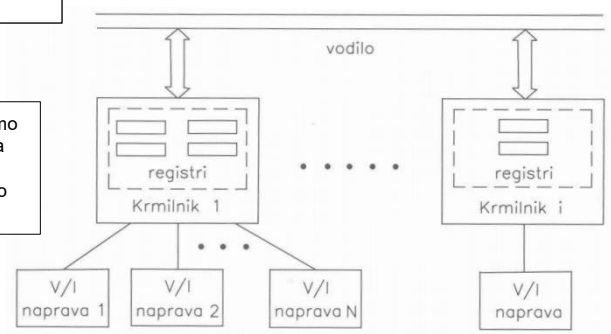
Programski vhod / izhod
- **programmed I / O**
- z **V / I** komunicira **CPE**
- vsak podatek iz **GP** v **CPE**
- nato v **napravo** ali **obratno**
- prenos z zaporedjem ukazov
- hiba **počasnost** in **zasedenost** CPE

Neposredni dostop do pomnilnika:
- **direct memory access DMA**
- naprava komunicira neposredno z **GP**
- rabimo **DMA krmilnik** namesto **CPE**
- **vhodno / izhodni** procesorji

→ vsaka **V / I** priključena preko **krmilnika** naprave
→ vezje za **prenos podatkov** v napravo in iz nje
→ nekateri za več naprav naenkrat
→ s krmilnikom komuniciramo preko **registrov**
→ registri **krmilnikov** so lahko v istem naslovnem prostoru kot **GP** ali v **posebnem**

1. **Pomnilniško preslikan vhod/izhod (memory mapped I/O)**
   - registri krmilnikov so v pomnilniškem naslovnem prostoru
   - iz CPE so videti kot pomnilniške lokacije
   - iz njih bere in vanje piše z ukazi za dostop do pom.
   - ni posebnih V/I ukazov
2. **Ločen vhodno/izhodni prostor**
   - registri krmilnikov so v posebnem naslovnem prostoru
   - za dostop do registrov so potrebni *posebni V/I ukazi*
   - pri tem CPE aktivira tudi določen(e) signal(e), ki pove(jo), da se naslavlja V/I naslovni prostor
3. **Posredno preko vhodno/izhodnih procesorjev**
   - tudi tu so registri krmilnikov v posebnem naslovnem prostoru, ki pa iz CPE ni neposredno dostopen
   - vmes so še vhodno/izhodni procesorji (razbremenijo CPE)
   - pri velikih računalnikih

pogosto srečamo oba načina za počasne računalnike po navadi prvi.



PARALELIZEM

→ zaporedno izvajanje ukazov
→ **Flynnova** klasifikacija: koliko **ukazov** se izvršuje naenkrat in koliko **ponovitev operandov** en ukaz obdeluje nankrat
→ **SISD**: naenkrat en ukaz na eni zbirki operandov najbolj zmogljivi so vektorski
→ **SIMD**: izvajajo en ukaz na **n** zbirkah oprenadov imajo eno kontrolno enoto **n ALE** enot in **n** množic **registrov**
→ **MISD**: več ukazov na eni enoti operandov ne obstajajo
→ **MIMD**: zvajajo **več ukazov** na **več zbirkah** operandov multiprocesorji in multiračunalniki

→ **MIMD tesno** povezani z skupnim pomnilnikom in **rahlo** povezani preko V / I sistemov
→ **večjedrni** procesorji so tesno povezani
→ **SIMD** in **MIMD** so paralelni računalniki

Gustafsonov zakon:
→ če **povečujemo** problem se zaporedni del f **zmanjšuje** in pohitritev postane skoraj **linearna**

Amdahlov zakon:
→ pohitrimo delovanje dela operacij
→ **f** je **zaporedni delež** programa
→ **1 - f** je **vzporedni delež** programa
→ delovanje **n krat** hitrejše

$$S(N) = \frac{1}{f + (1-f)/N} = \frac{N}{1 + (N-1)f}$$

Nivo 5: Višji prog. jezik
- prevajanje ali interpretiranje

Nivo 4: Zbirni jezik
- prevajanje

Nivo 3: Operacijski sistem
- interpretiranje

Nivo 2: Strojni jezik
- interpretiranje

Nivo 1: Mikroprogramski jezik
- interpretiranje

Nivo 0: Digitalna logika

prevajanje:
- izvorni program v **enem** jeziku
- ciljni program v **nižjem** jeziku

interpretacija:
- izvorni program se prevaja **sproti**
- manjša **hitrost** ampak bolj **fleksibilno**

delno prevajanje:
- prevajanje v vmesno kodo ki se jo interpretira npr **Java**

hardware
software
firmware

$IPS = f_{CPE} / CPI$

IPS ... Instructions Per Second
f_{CPE} ... frekvenca ure
CPI ... Clocks Per Instruction

IPS lahko povečamo s povečanjem **frekvence** ali drugačno **zgradbo** CPE

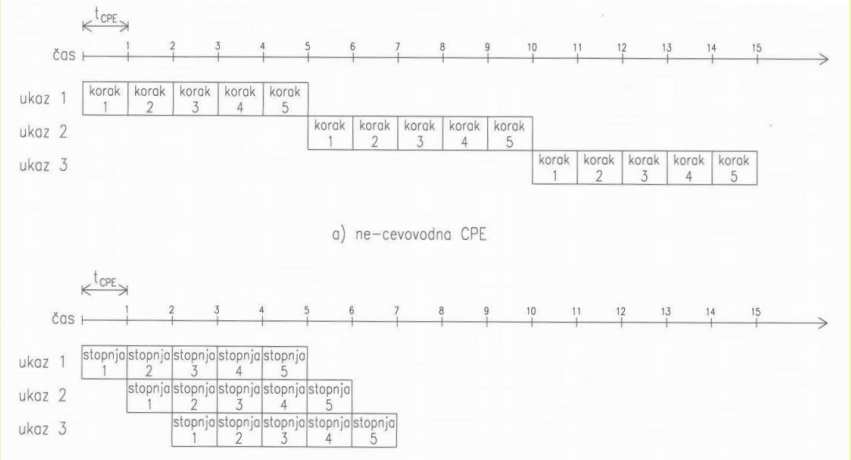
- število izvršenih ukazov v danem času se **poveča** zaradi 2 vzrokov
- manjši CPI
- krajši CPE čas: če uspemo narediti **enostavne** podoperacije

$$t_{CPE} = t_{shranjevanje} + t_{podoperacija}$$

zmogljivost z večanjem števila stopenj nekaj časa **narašča** nato pa začne **padati**. Mogoče narediti cevovod tako da ga programer ne vidi pri drugih vrstah paralelnega procesiranja to pogosto ne velja. bolj enostavna realizacija pri **RISC** kot pri **CISC**

CEVOVOD:

→ več ukazov se izvršuje tako da se **koraki** izvrševanja prekrivajo
→ vsako podoperacijo opravi del cevovoda: **stopnja cevovoda**



→ čas **med dvema pomikoma** je praviloma enak **urini periodi** CPE časa
→ perioda ne more biti **krajša** od časa **najpočasnejše** stopnje cevovoda
→ dobro če operacije časovno **uravnotežene**
→ pri **idealno** uravnoteženi **cevovodni** CPE z **n** stopnjami je zmogljivost **n** krat **večja** kot pri **ne cevovodni** CPE
→ trajanje ukaza: **latenca**

Cevovodne nevarnosti:

1. **Strukturne nevarnosti**
   - kadar več stopenj cevovoda v neki urini periodi potrebuje isto enoto
2. **Podatkovne nevarnosti**
   - kadar ukaz potrebuje kot vhodni operand rezultat prejšnjega, še ne dokončanega ukaza
3. **Kontrolne nevarnosti** **PC**
   - možne pri skokih, klicih in drugih kontrolnih ukazih, ki spreminjajo vsebino

HIP CEVOVOD:

→ ima dva predpomnilnika **ukazni** in **operandni**
→ vstavljanje ukazov **NOP** za **nevarne** ukaze
→ **NOP** ne spremeni stanja registrov
→ ekvivalentno čakanju **eno urino** periodo
→ pri **višjih** programskih jezikih jih vstavlja **prevajalnik**

	Urina perioda										
Št. ukaza	1	2	3	4	5	6	7	8	9	10	11
ukaz i	IF	ID	EX	ME	WB						
ukaz i+1		IF	ID	EX	ME	WB					
ukaz i+2			IF	ID	EX	ME	WB				
ukaz i+3				IF	ID	EX	ME	WB			
ukaz i+4					IF	ID	EX	ME	WB		
ukaz i+5						IF	ID	EX	ME	WB	
ukaz i+6							IF	ID	EX	ME	WB

Strukturne nevarnosti:

→ **več stopenj** cevovoda v urini periodi potrebuje **isto enoto**
→ kjer nekateri ukazi trajajo več urinih period
→ izguba **manjša** kot ostale nevarnosti
→ popolno odpravljanje drago
→ **nimamo** v HIP

Podatkovne nevarnosti:

→ kadar ukaz potrebuje kot **vhodni** operand rezultat še ne **dokončanega** ukaza
→ **rešitev**: cevovodna zaklenitev mehurček ki v bistvu **NOP**
→ pri **HIP** samo v **ID** koraku
→ **rešitev**: **premoščanje** prenesemo rezultat iz drugih korakov v **ID** možno le pri **load**
→ primerjamo registra
→ **rešitev**: drugače razporedimo ukaze

Kontrolne nevarnosti:

→ kadar imamo **skoke**
→ preverjanje pogoja za skok naj se izvaja čim bližje **prvi stopnji** cevovoda
→ **izračun** skočnega naslova naj se izvaja čim bližje prvi stopnji cevovoda
→ **rešitev**: vstavljanje **mehurčka** v **IF** in **ID**
→ skočni ukaz tako izgubi **2 periodi**
→ cevovod predpostavi da skoka ne bo
→ **statična** in **dinamična predikcija**

- **HIP** skočno zakasnitev 2 urini periodi
- skočni predpomnilnik **SPP** z enim **prediktorskim** bitom

Statična predikcija:

→ skuša napovedati **bolj verjeten** rezultat preverjanja skočnega pogoja
→ med izvrševanjem programa se ne spreminja
→ **skočne reže**
→ enako številu stopenj cevovoda pred stopnjo v kateri se v **PC** zapiše **skočni naslov**
→ v **HIP** to pomeni **EX** torej **dve reži**
→ pri zakasnjenih skokih ne glede na izpolnjenost pogoja vsi ukazi

Dinamična predikcija:

→ prilagaja se dogajanju v **programu**
→ **1** in **2 bitna prediktorska tabela**
→ **korelacijski** prediktor
→ **turnirski** prediktor
→ **skočni predpomnilnik**
→ **vrnitveni** prediktor
→ enota za prevzem podatkov

Stopnja cevovoda	Problematične pasti pri HIP
IF	napaka strani (pri branju ukaza), zaščita pomnilnika
ID	nedefiniran ukaz
EX	preliv
MEM	napaka strani (pri dostopu do operanda), zaščita pomnilnika neporavnan operand,
WB	nobena

Zadetek v SPP	Napoved izpolnjenosti pogoja	Dejanska izpolnjenost pogoja	Število čakalnih period
da	izpolnjen	izpolnjen	0
da	izpolnjen	neizpolnjen	2
ne	neizpolnjen	izpolnjen	2
ne	neizpolnjen	neizpolnjen	0

Prekinitve in pasti cevovoda:

→ **vhodno / izhodne** prekinitve
→ **programske** pasti
→ pasti med **izvrševanjem** ukaza

Operacije več urinih period:

- **Celoštevilska enota** (integer unit)
  - celošt. ALE ukazi, skoki, load, store
  - pri HIP je le ta
- **Enota za operacije v plavajoči vejici** (floating-point unit)
  - seštevanje, odštevanje, pretvorbe
- **Enota za množenje**
  - celoštevilsko in v FP
- **Enota za deljenje**
  - celoštevilsko in v FP

cevovod prepočasen uvedemo funkcijske enote ki imajo tri vrste **podatkovnih nevarnosti**. **HIP** samo **RAW**

podatkovne nevarnosti:
→ **read** after **write**
→ **write** after **read**
→ **read** after **read**

špekulativno izvajanje ukazov: predpostavimo da dinamična predikcija pravilna potrebujemo pa mehanizem ki odstrani nepravilne ukaze za to **preureditveni izravnalnik.**

poskušamo približevati **CPI** vrednosti **1**. dinamična **predikcija** skokov, dinamično **razvrščanje** in **špekulativno** izvrševanje ukazov. Če želimo manj od 1 **večizstavitveni procesorji.**

1. Prevzem ukazov
   - izstavitev *n* ukazov zahteva, da je ukazni PP sposoben dostavljati *n* ukazov v periodi
   - treba je povečati širino dostopa do čakalne vrste in zmogljivost pomnilnika
2. Izstavljanje ukazov
   - če je med (*n*) ukazi skok z napovedanim skočnim pogojem, se preostali ukazi ne izstavijo
     - prevzem ukazov v naslednji periodi pa se začne z napovedanega skočnega naslova
   - potrebno je tudi preveriti medsebojne odvisnosti med operandi
     - pri *n* ukazih s 3 reg. operandi je potrebnih $n(n-1)$ primerjav $(2(n-1) + 2(n-2) \ldots)$

omejitve paralelizma:
→ količina paralelnosti v programih je **omejena**
→ s povečevanjem količine **logike** omejeno
→ paralelizem na višjem nivoju: **niti**
→ **niti** si delijo **FE** enega procesorja
→ vsaka nit ima svoje **stanje**
→ nit ima svojo **kopijo registrov** in svoj **PC**
→ niti si delijo **GP** in **PP**
→ nit vidi procesor kakor da je namenjen le njej
→ **časovna**: drobno in grobo zrnata
→ **istočasna**

superskalarni procesorji in po novem **eksplicitno poimenovanje** registrov

$H = N_p/N = N_p/(N_g+N_p)$
N_p ... število zadetkov
N_g ... število zgrešitev $(=N-N_p)$

PREDPOMNILNIKI

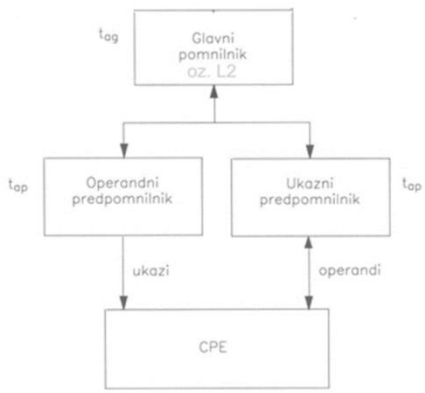
→ programi večkrat uporabijo iste **ukaze** in **operande**
→ **GP** zamenjamo z **pomnilniško hierarhijo**
→ **prostorska** in **časovna** lokalnost
→ **DRAM** se povečuje z leti ampak ne tako hitro kot **CPE**
→ **CPE** ne sme čakati na pomnilnik zato uporabljamo **predpomnilnik**
→ **pomožni pomnilnik**: posreden dostop CPE preko **V** / **I** ukazov
→ **GP** so polprevodniški iz integriranih vezij
→ množica podatkov v **PP** je podmnožica v **GP**
→ pri **cevovodnih** CPE je predpomnilnik razdeljen na **ukazni** in **operandni** del

Štirinivojska pomnilniška hierarhija
- M_1: predpomnilnik 1. nivoja (SRAM)
- M_2: predpomnilnik 2. nivoja (SRAM)
- M_3: GP (DRAM)
- M_4: pomožni pomnilnik (magnetni disk)

nivoji **predpomnilnika**

L1 (level 1) je manjši in hitrejši in je kar na čipu CPE
L2 je malo večji in malo počasnejši (danes običajno tudi na CPE)
L3 je večji in počasnejši (običajno ni na CPE)
- še vedno pa hitrejši od DRAMa

a) Homogen predpomnilnik



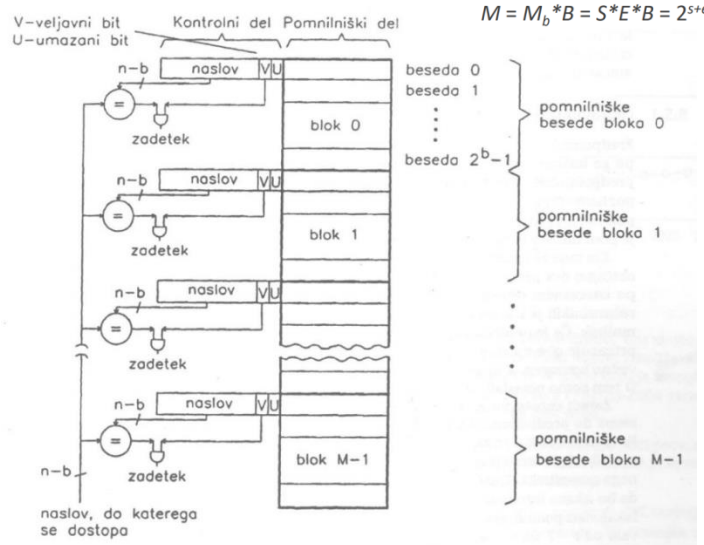
b) Nehomogen predpomnilnik

→ **zadetek**: ko je naslov že v **PP**
→ **zgrešitev**: ko naslov ni še v **PP**
→ kadar zgrešitev moramo blok podatkov prenesti iz **GP**
→ uspešnost delovanja PP merimo z **verjetnostjo** zadetka
→ **H** je verjetnost zadetka
→ imamo **čas dostopa**
→ t_b je miss penalty
→ nekateri deli GP vedno zgrešitev
→ ima **kontrolni** in **pomnilniški** del
→ 2^b pomnilniških besed v pomnilniškem delu
→ če pomnilniški naslov n biten potem je **b** bitov **beseda** in **n** – **b** bitov **naslov** bloka
→ kontrolni del informacijo ki enolično piše vsak blok: **naslov** bloka v **GP**, **veljavni** in **umazani** bit
→ veljavni bit določi zgrešitev ali zadetek
→ umazani bit **1** ko pišemo **0** ob prenosu bloka
→ število blokov je $\mathbf{M_b}$

$t_a = t_{ap} + (1\text{-}H)t_B$

$M_b = S*E = 2^{s+e}$ blokov

$M = M_b*B = S*E*B = 2^{s+e+b}$ pomnilniških besed

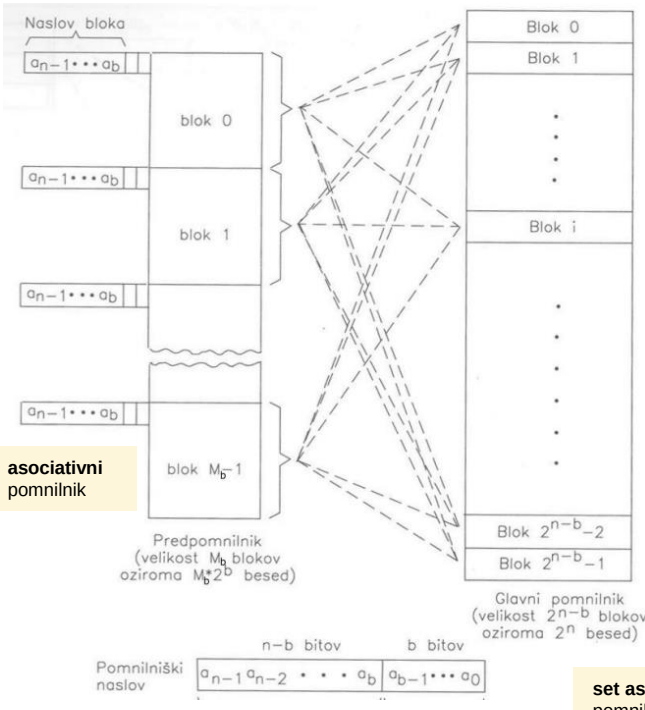


→ če so vsi veljavni bloki zasedeni **menjava bloka**
→ imamo **asociativni**, **set asociativni** in **direktni** PP

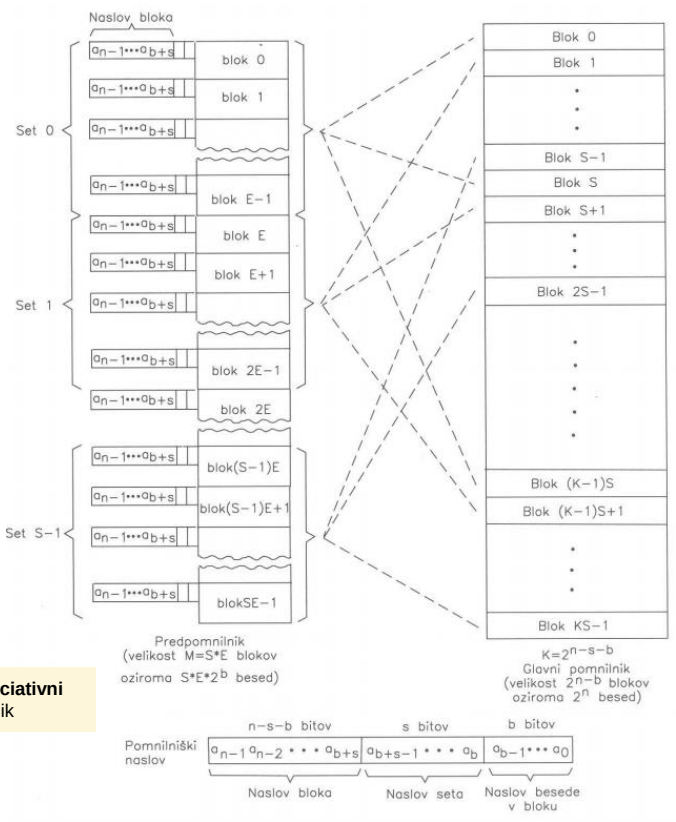
SET ASOCIATIVNI
→ **znotraj seta** shranjevanje na poljubno mesto
→ sestavljen iz več majhnih **AP**
→ 2^s setov
→ stopnja **asociativnosti** število blokov v setu 2^e
→ **omejitev**: v **GP** v naprej določeno v kateri set se preslika
→ naslov seta je **predpomnilniški indeks**

ASOCIATIVNI
→ ima največjo verjetnost zadetka
→ so najboljši in najbolj dragi
→ omogoča **poljubno pozicijo** bloka

DIREKTNI
→ blok je enak setu
→ v naprej **določeno** kam se kaj preslika iz GP



asociativni pomnilnik



set asociativni pomnilnik

→ dva načina da **zamenjamo** blok ob zgrešitvi
→ **naključno** in **LRU** oz. **least recently used**
→ več kot je blokov v setu težje
→ **pisanje**: se začne ko potrjen zadetek
→ **write through**: pišemo v **PP** in **GP**
→ **write back**: piše v **PP** in spremenjeni blok prenesemo nazaj v **GP** zato **umazani bit** po pisanju nastavimo na **1**
→ **write buffer**: CPE shrani podatek za v GP
→ pri pisanju tudi pri zadetku **2 periodi**
→ problem **skladnosti**

- pri bralnih zgrešitvah se blok vedno prenese v PP (zamenja enega od obstoječih)
- pri pisalnih zgrešitvah 2 možnosti:
  1. **Pisalna zamenjava** (write allocate)
     - prenos novega bloka v PP (podobno kot bri branju)
     - bolj običajno pri pisanju nazaj
     - bolj razširjena
  2. **Pisanje naokrog** (write around, no write allocate)
     - zamenjava bloka samo v GP (ne v PP)
     - bolj običajno pri pisanju skozi

Vrste zgrešitev

1. **Obvezne zgrešitve** (compulsory misses)
   - reče se tudi zgrešitve prvega dostopa
2. **Velikostne zgrešitve** (capacity misses)
   - zaradi končne velikosti PP običajno ne more vsebovati vseh blokov, ki jih program potrebuje
   - zato prihaja do zamenjav blokov, ki so kmalu spet potrebni
3. **Konfliktne zgrešitve** (conflict misses)
   - zamenjava blokov, ki se preslikajo v isti set
   - pri čistem APP jih ni

$CPEčas = (CPEperiode_{izvrš.} + CPEperiode_{čak.})*t_{CPE}$
$CPEperiode_{izvrš} = I*CPI_{idealni}$

$CPEperiode_{čak} = N*(1-H)*K_Z$
$N = I*(1+M_I)$

$CPEčas = I*(CPI_{idealni} + M_I*(1-H)*K_Z)*t_{CPE}$
Če $H_R \neq H_W$:
$CPEperiode_{čak} = N_R*(1-H_R)*K_{Z,R} + N_W*(1-H_W)*K_{Z,W}$

Če $H_{UPP} \neq H_{OPP}$:
Število zgr.: $N*(1-H) \rightarrow I*(1-H_{UPP}) + I*M_I*(1-H_{OPP})$
Verj. zgr.: $1-H = 1-H_{UPP} + M_I*(1-H_{OPP})$
$CPEperiode_{čak} = (I*(1-H_{UPP}) + I*M_I*(1-H_{OPP}))*K_z$

N_R ... število bralnih dostopov
N_W ... število pisalnih dostopov
N ... število vseh pom. dostopov
I ... število ukazov
H ... povprečna verjetnost zadetka
K_Z ... povprečna zgrešitvena kazen
M_I ... povprečno število operandnih pomnilniških dostopov na ukaz
$CPI_{idealni}$ predpostavi, da ni zgrešitev

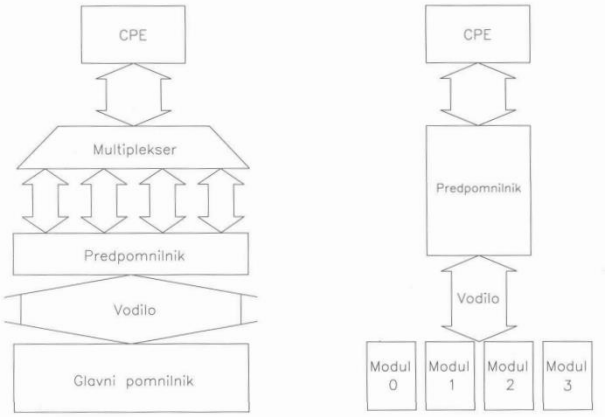
zmanjševanje zgrešitvene kazni:
→ **vnaprejšnji prevzem bloka**: pri prenosu bloka k v PP se prebereta še npr. bloka k+1 in k+2 in shranita v read buffer
→ **neblokirajoči PP**: med zamenjavo bloka PP deluje naprej in CPE lahko špekulativno izvršuje naslednje ukaze

POMNILNIKI

→ **cena:** SRAM, DRAM, bliskovni, magnetni
→ **hitrost**: čas dostopa in hitrost dostopa
→ **način dostopa**: naključni, zaporedni, krožni… **RAM** je random access memory
→ **asociativni**: zelo hitri, primerjava!
→ **spremenljivost**: ROM, PROM, bralno pisalni
→ **PROM**: EPROM, EEPROM, flash
→ **obstojnost**
→ **zanesljivost**

izguba informacije:
→ **destruktivno** branje
→ **dinamično** shranjevanje
→ **izpad** napajanja

zaščita GP
OS ščiti **GP** in del OS je vedno gor. razpad sistema če spremenimo vsebino lokacij kjer OS. vsaka stran ima svoj **zaščitni ključ.**



a) Glavni pomnilnik s široko podatkovno potjo.

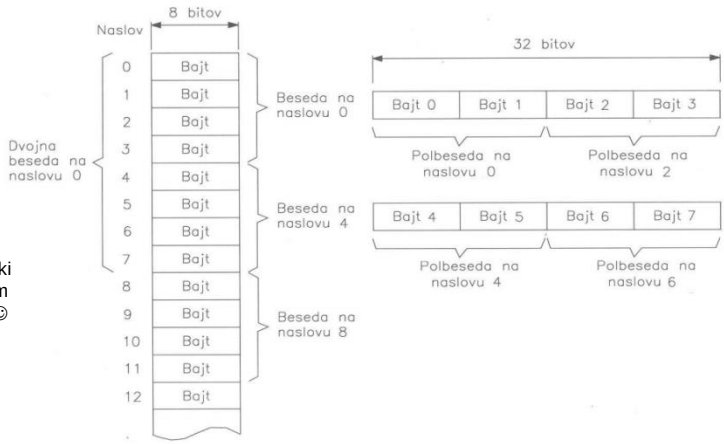
b) Glavni pomnilnik z ozko podatkovno potjo in pomnilniškim prepletanjem.

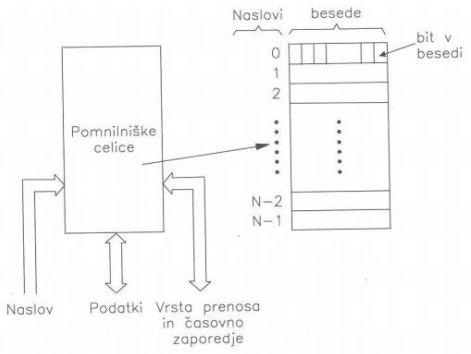
Pomnilniška beseda

- to je najmanjše število bitov s svojim naslovom
  - **dolžina besede** (običajno 1B oz. 8 bitov)
- običajno je možen dostop do več besed

Pomnilniški naslov

- binarno število
- **dolžina naslova** določa velikost pomnilniškega prostora
  - pri *m*-bitnem naslovu $a_{m-1}...a_1a_0$ je lahko največ 2^m besed





pomnilniki in predpomnilniki so trash in sm obupala kle ☺