7 PARALELIZEM NA NIVOJU UKAZOV

BRANKO ŠTER

PO KNJIGI - DUŠAN KODEK: ARHITEKTURA IN ORGANIZACIJA RAČUNALNIŠKIH SISTEMOV

Z doslej obravnavanim načinom gradnje CPE je težko doseči CPI < 4

- zaporedno izvrševanje

Število ukazov na sekundo je

$$IPS = f_{CPE} / CPI$$

IPS ... Instructions Per Second

f_{CPE} ... frekvenca ure

CPI ... Clocks Per Instruction

IPS lahko povečamo:

- s povečanjem f_{CPE} (hitrejši logični elementi)
- z drugačno zgradbo CPE, ki bi zmanjšala CPI (več logičnih elementov)

V 20 letih se hitrost elementov poveča ~ 10x, št. elementov na čipu pa ~ 1000x

- zato je druga varianta bolj perspektivna

Z uporabo večjega števila elementov skušamo zmanjšati *CPI* (in s tem povečati *IPS*)

- Skušamo doseči čimvečjo **paralelnost** (istočasnost) operacij

Ena možnost je paralelno programiranje

- programer določi, kaj naj se izvaja paralelno
- dokaj komplicirano: potrebujemo izvorno kodo in znanje, kako to narediti
- večina uporabnikov se s tem ne želi ukvarjati

Enostavneje je izkoristiti **paralelizem na nivoju ukazov** (instruction-level parallelism, ILP)

Najpogostejši način je **cevovod** (pipeline)

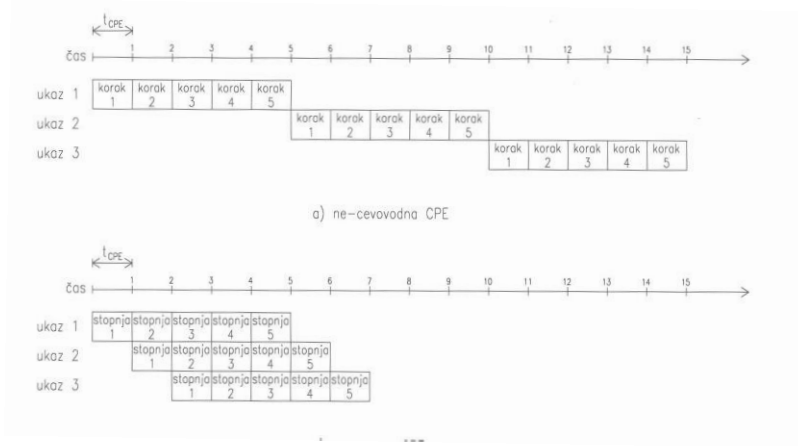
Cevovod - splošno

Cevovod (pipeline)

- Pri cevovodu se naenkrat izvršuje več ukazov tako, da se posamezni koraki izvrševanja prekrivajo
- Podobno tekočemu traku pri proizvodnji
- Vsako podoperacijo opravi določen del cevovoda
  - **stopnja cevovoda** (pipeline stage) ali **segment cevovoda**
- Stopnje tvorijo nekakšno cev
  - ukazi vstopajo v cev, potujejo skoznjo in izstopajo na koncu cevi

Primer: ne-cevovodna CPE in cevovodna CPE

- v drugem primeru se izvrši 5x več ukazov (če zanemarimo začetno zakasnitev)



Čas med dvema pomikoma je praviloma enak urini periodi t_{CPE}

Perioda ne more biti krajše od časa, ki ga za izvršitev svoje podoperacije potrebuje najpočasnejša izmed stopenj cevovoda

- zato je dobro, če so podoperacije časovno uravnotežene
- pri idealno uravnoteženi cevovodni CPE z *N* stopnjami je zmogljivost *N*-krat večja kot pri ne-cevovodni CPE
  - hkrati se obdeluje *N* ukazov
  - na izhodu iz cevovoda jih je zato *N*-krat več
  - CPI *N*-krat manjši
- resnični cevovodi pa nikoli niso idealno uravnoteženi, zato zmogljivost ni *N*-kratna

Število izvršenih ukazov v danem času se poveča zaradi 2 vzrokov:

1. manjši CPI
   - čeprav je trajanje ukaza (**latenca**) enako ($N*t_{CPE}$)
2. krajša t_{CPE}
   - če uspemo narediti enostavne podoperacije
   - $t_{CPE} = t_{shranjevanje} + t_{podoperacija}$

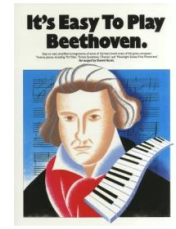
     $t_{shranjevanje}$ ... čas shranjevanja rezultata podoperacije v registre
   - z več stopnjami lahko zmanjšamo $t_{podoperacija}$, $t_{shranjevanje}$ pa ne

Supercevovodni računalniki
- Intel Pentium 4
  - 20-stopenjski, kasneje 31-stopenjski cevovod
    - želeli so doseči f_{CPE} 10GHz, a je bila poraba prevelika (problemi s hlajenjem, tj. odvajanjem toplote)
    - kasnejši CPU (npr. Core) imajo (le) 14-stopenjski cevovod

BTW: najvišja dosežena frekvenca je nekaj nad 8GHz (AMD)
- s pomočjo navijanja frekvence (overclocking), pa tudi polivanja čipa s tekočim dušikom

Zakaj se ne uporabljajo poljubno dolgi cevovodi?

- pač povečujemo N in višamo hitrost CPU …



Cevovodne nevarnosti

Razlog so **cevovodne nevarnosti** (pipeline hazards), zaradi katerih se mora cevovod ustaviti in počakati, da nevarno mine

3 vrste cevovodnih 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**
   - možne pri skokih, klicih in drugih kontrolnih ukazih, ki spreminjajo vsebino PC

Zato zmogljivost z večanjem števila stopenj nekaj časa narašča, nato pa začne padati!

Prednost cevovoda je, da ga je mogoče narediti tako, da je za programerja neviden

- arhitektura računalnika in programiranje ostane enako tudi z razvojem računalnika
  - starejši programi tečejo tudi na novejših računalnikih
- pri drugih vrstah paralelnega procesiranja to pogosto ne velja
- zadnji Intelov necevovodni procesor je bil 80386 (iz leta 1985)

Cevovodna podatkovna enota

Cevovodna realizacija je pri računalnikih RISC enostavnejša kot pri CISC

- preprostejši ukazi

Cevovodno CPE si bomo ogledali na primeru računalnika HIP

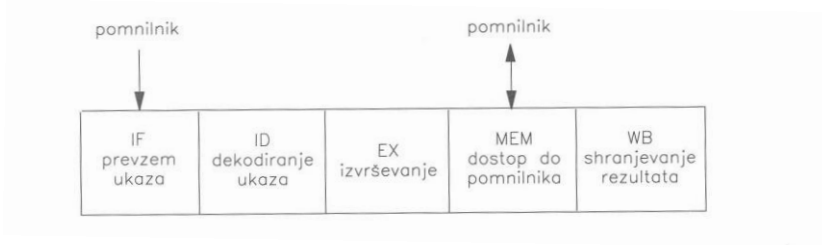
5 korakov izvrševanja ukazov: 5 stopenj cevovoda

1. IF: prevzem ukaza in sprememba PC
2. ID: dekodiranje ukaza in dostop do registrov
3. EX: izvrševanje operacije
4. MEM: dostop do pomnilnika
5. WB: shranjevanje rezultata

Vsaka stopnja mora opraviti svoje delo v eni urini periodi

- prej so nekateri koraki potrebovali 2 periodi

Petstopenjska cevovodna podatkovna enota procesorja HIP:

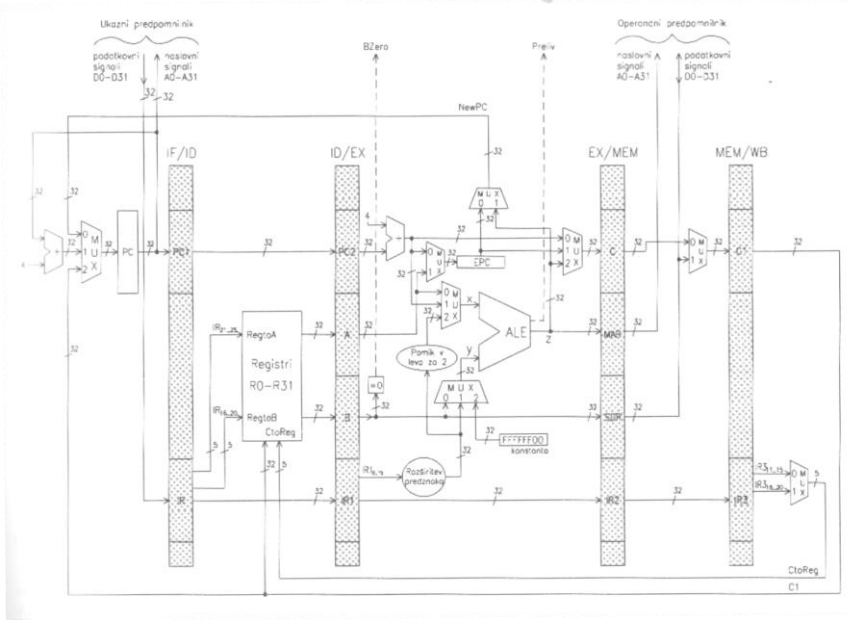


Včasih sta potrebna dva hkratna dostopa do pomnilnika

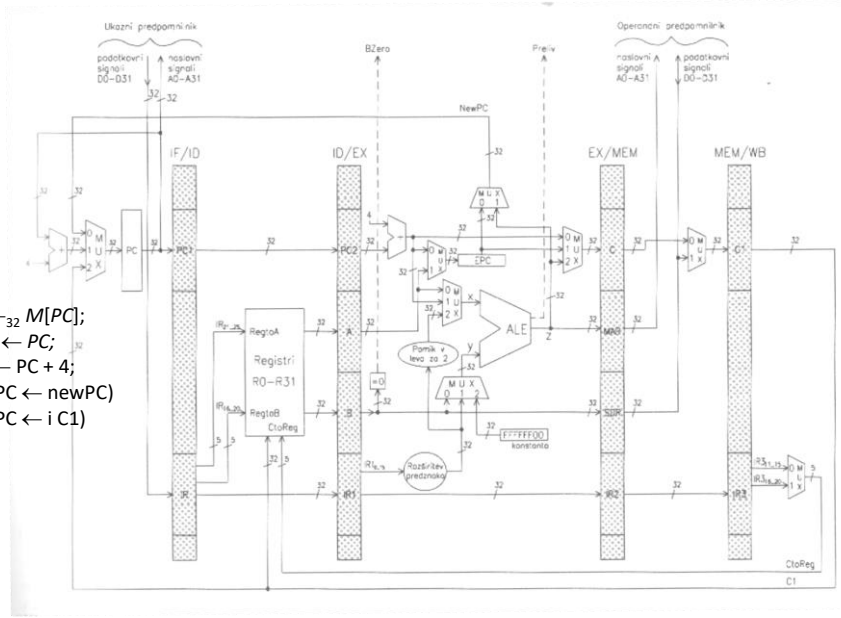
Cevovodni HIP ima zato 2 predpomnilnika:

- ukazni
- operandni

Cevovodna podatkovna enota



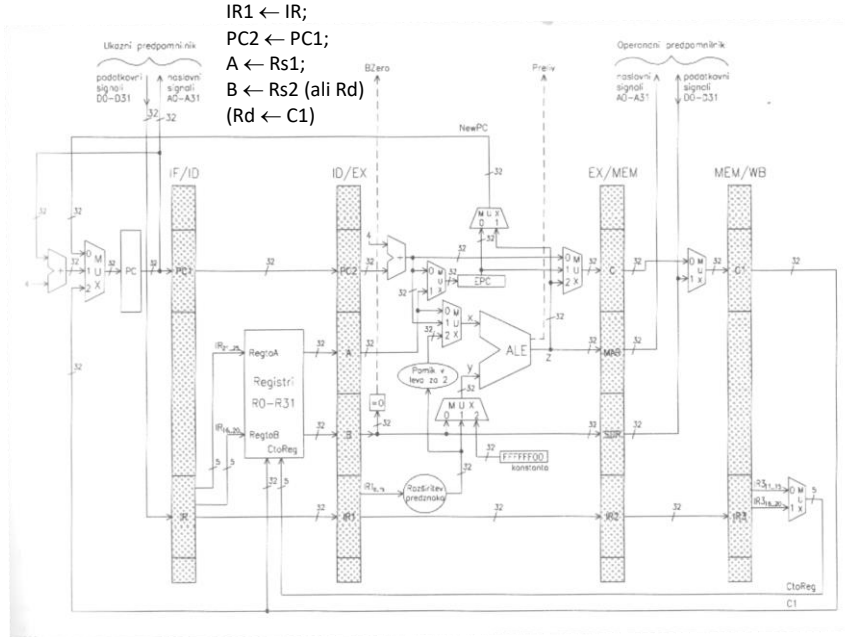
1. Stopnja IF



IR ←$_{32}$ *M*[*PC*];
PC1 ← *PC;*
PC ← PC + 4;
(ali PC ← newPC)
(ali PC ← i C1)

2. Stopnja ID

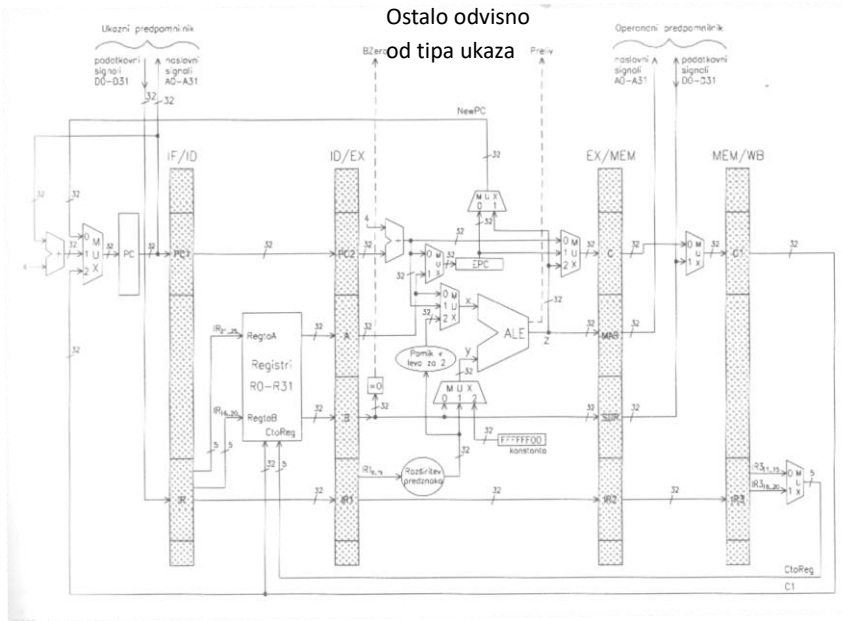
IR1 ← IR;
PC2 ← PC1;
A ← Rs1;
B ← Rs2 (ali Rd)
(Rd ← C1)



3. Stopnja EX

IR2 ← IR1;

Ostalo odvisno od tipa ukaza



→ Stopnja EX

- prehod v to stopnjo se imenuje **izstavitev ukaza** (instruction issue)
  - tukaj se ukaza več ne da na preprost način izničiti (v IF in ID ni problema)
- Delovanje stopnje EX je odvisno od vrste ukaza:

Ukazi za prenos podatkov

IR2 ← IR1

SDR ← B

MAR ← A + raz($IR1_{0..15}$)

ALE ukazi

IR2 ← IR1

SDR ← B

C ← A op B (ali raz($IR1_{0..15}$))

Klic in brezpogojni skok

IR2 ← IR1

C ← PC2 + 4

NewPC ← A + raz($IR1_{0..15}$)

Pogojni skoki

IR2 ← IR1

NewPC ← PC2 + 4 + raz($IR1_{0..15}$)

TRAP

IR2 ← IR1

EPC ← PC2 + 4

MAR ← FFFFFF00 + 4xraz($IR1_{0..15}$)

I ← 0

RFE

IR2 ← IR1

NewPC ← EPC

MOVER in MOVRE

IR2 ← IR1

C ← EPC (pri MOVER)

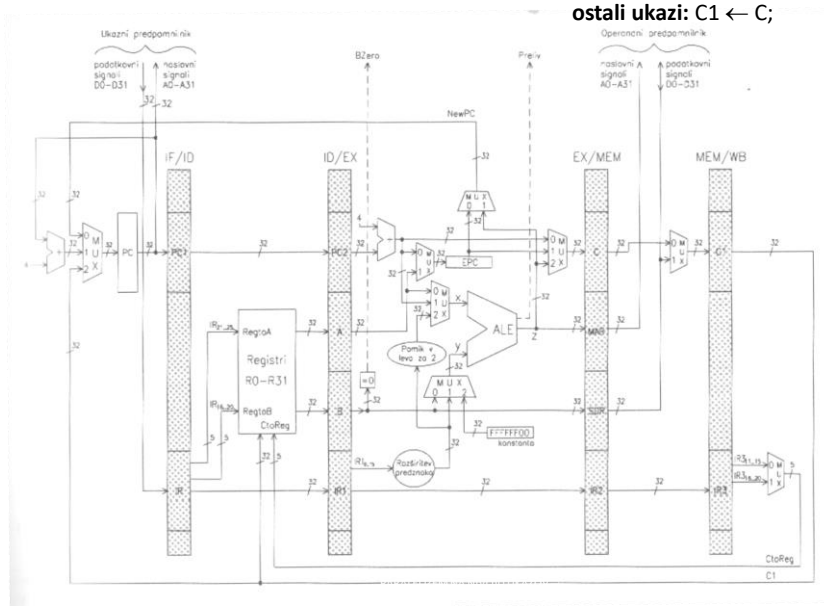
EPC ← A (pri MOVRE)

EI in DI

IR2 ← IR1

I ← 1 (pri EI) ali I ← 0 (pri DI)

4. Stopnja MEM IR3 ← IR2;
load in TRAP: C1 ← M[MAR]; (OPP)
store: M[MAR] ← SDR; (OPP)
ostali ukazi: C1 ← C;



→ Stopnja MEM

load in TRAP

IR3 ← IR2; C1 ← M[MAR];
branje iz operandnega PP

store

IR3 ← IR2; M[MAR] ← SDR;
SDR se shrani v operandni PP

Ostali ukazi

IR3 ← IR2; C1 ← C; (zaradi WB)

pri zgrešitvah se ustavijo vse stopnje cevovoda (čakanje na MemWait)

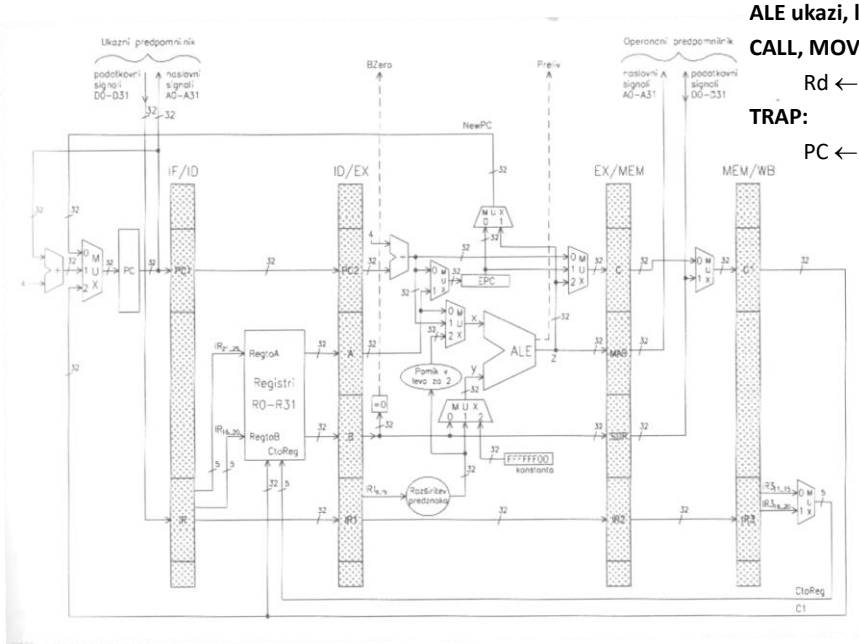
5. Stopnja WB

ALE ukazi, load, CALL, MOVER:

Rd ← C1;

TRAP:

PC ← C1;



Idealen potek izvrševanja ukazov v cevovodu

	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

Cevovodne nevarnosti

Ob pojavu nevarnosti se mora cevovod ustaviti in počakati, da nevarnost mine

Programsko odpravljanje cevovodnih nevarnosti

- pri prvih RISC (80. leta)
- vstavljanje ukazov NOP za ukaze, ki lahko povzročijo nevarnost
  - NOP ne spremeni stanja registrov
  - ekvivalentno čakanju eno urino periodo
  - pri višjih programskih jezikih jih vstavlja kar prevajalnik
- 2 slabosti:
  - potrebno je drugačno programiranje
  - upočasnjeno delovanje
- ni več posebno aktualno

Strukturne nevarnosti

- SN: kadar več stopenj cevovoda v neki urini periodi potrebuje isto enoto (reg., ALE, GP, PP)
- SN tudi na računalnikih, kjer nekateri ukazi trajajo več urinih period
  - Množenje, deljenje, FP operacije
- Izguba zaradi SN običajno bistveno manjša kot zaradi drugih nevarnosti
- Popolno odpravljanje SN je drago (večje število enot)
  - Na manjših računalnikih se pač dovoli, da do njih občasno pride
- Pri HIP do SN ne prihaja
  - Če PP ne bi bil razdeljen, bi lahko prihajalo (load, store)

Podatkovne nevarnosti

- PN (data hazard): kadar ukaz potrebuje kot vhodni operand rezultat še ne dokončanega ukaza
  - medsebojna odvisnost ukazov, ki so blizu skupaj
- Npr.

ADDI	R20, R0, #0	R20 ← 0
SUB	R3, R4, R5	R3 ← R4 – R5
ADD	R1, R3, R6	R1 ← R3 + R6
AND	R2, R3, R7	R2 ← R3 ∧ R7
XOR	R8, R3, R9	R8 ← R3 ⊕ R9
OR	R10, R3, R12	R10 ← R3 ∨ R12

Rešitev 1:
Cevovodna zaklenitev (pipeline interlock)

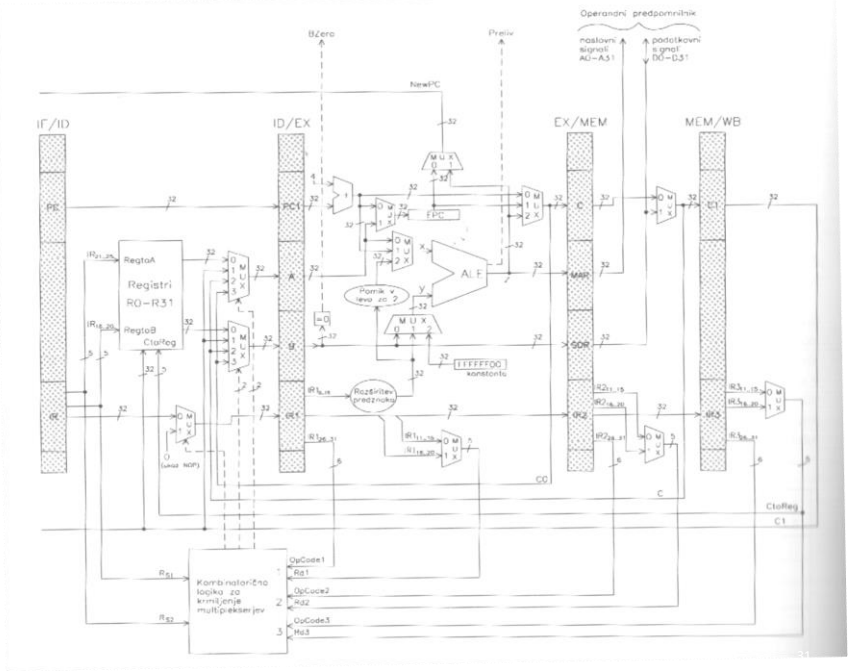
- Stopnja ID se ustavi za 3 periode (zato se mora tudi IF, ker bi se sicer izgubil ukaz, ki je v njej). EX,MEM in WB morajo delovati naprej (sicer se vzrok za nevarnost ne bo odstranil).
- Vsaka stopnja, kjer lahko pride do napake, mora imeti vgrajeno logiko za cev. zaklenitev
- mehurček (bubble) je strojni ekvivalent operacije NOP
  - mehurček naredimo z "ukazom" NOP (32 ničel, ADDI ...)
  - v tem primeru ga "izvaja" stopnja EX
  - mehurček potuje od stopnje EX naprej
- Pri HIP lahko pride do PN le v stopnji ID
  - prisotnost nevarnosti se ugotovi s primerjavo bitov $IR_{16..20}$ in $IR_{21..25}$ (Rs1, Rs2) z biti $IRx_{16..20}$ in $IRx_{21..25}$ (format 1, 2), x=1..3
  - to velja le za ukaze, ki shranjujejo v Rd (v stopnjah EX, MEM in WB)

	Urina perioda												
Ukaz	1	2	3	4	5	6	7	8	9	10	11	12	13
ADDI R20,R0,#0	IF	ID	EX	MEM	WB								
SUB R3,R4,R5		IF	ID	EX	MEM	WB							
ADD R1,R3,R6			IF	○	○	○	ID	EX	MEM	WB			
AND R2,R3,R7							IF	ID	EX	MEM	WB		
XOR R8,R3,R9								IF	ID	EX	MEM	WB	
OR R10,R3,R12									IF	ID	EX	MEM	WB

Rešitev 2:
Premoščanje (bypassing, data forwarding)

- rezultat ukaza ADD iz EX prenesemo v ID in ustavljanje ni potrebno
- vendar premoščanje le iz EX v ID ni dovolj:
  - tudi AND in XOR rabita rezultat ukaza ADD (v R3 ga še ni, v EX pa ga ni več)
- logika za premoščanje mora omogočati prenos iz stopenj EX, MEM in WB v stopnjo ID
- PN se ugotavljajo s primerjavo Rs1 in Rs2 v stopnji ID z Rd, ki ga uporabljajo ukazi v stopnjah EX do WB (če gre za ukaze, ki pišejo v Rd)
- če PN ni mogoče odpraviti (pri HIP možno le pri load), logika ustavi IF, v ID pa vstavi mehurček v IR1

Cevovodna PE z logiko za ugotavljanje podatkovnih nevarnosti in premoščanjem



Tako se bistveno zmanjša izguba zaradi PN, ni pa popolnoma odpravljena

- Včasih premoščanje ni možno, ker operanda ni v cevovodu
- Npr.

LW	R3, 56(R4)	R3 $\leftarrow_{32}$ M[56 + R4]
SUB	R1, R3, R6	R1 $\leftarrow$ R3 – R6
ADD	R2, R3, R7	R2 $\leftarrow$ R3 + R7
XOR	R8, R3, R9	R8 $\leftarrow$ R3 $\oplus$ R9

- LW dobi operand šele v stopnji MEM
- Premoščanje v ID ni možno, ker operanda ni v CPE
- Čakanje je nujno (se pa da zmanjšati za eno periodo)
- Pri ADD pa čakanje ni potrebno (zaradi premoščanja iz WB)

Potek izvajanja ukazov pri premoščanju

	Urina perioda								
Ukaz	1	2	3	4	5	6	7	8	9
LW R3,56(R4)	IF	ID	EX	MEM	WB				
SUB R1,R3,R6		IF	○	ID	EX	MEM	WB		
ADD R2,R3,R7				IF	ID	EX	MEM	WB	
XOR R8,R3,R9					IF	ID	EX	MEM	WB

Rešitev 3: Cevovodno razvrščanje (pipeline scheduling)

- Prevajalnik lahko s spreminjanjem vrstnega reda ukazov pogosto odpravi nevarnost
- Npr.:

a = b + c (naslovi naj bodo v registrih Ra, ..., Rf)
d = e – f

LW	R2,0(Rb)	R2 ← b
LW	R3,0(Rc)	R3 ← c
LW	R4,0(Re)	R4 ← e ukaz prestavljen naprej
ADD	R5,R2,R3	R5 ← b + c
LW	R6,0(Rf)	R6 ← f ukaz prestavljen naprej
SW	0(Ra),R5	a ← b + c
SUB	R7,R4,R6	R7 ← e – f
SW	0(Rd),R7	d ← e – f

Večina prevajalnikov danes uporablja cev. razvrščanje

- čakanja pa se vedno ne da odpraviti
- delež ukazov load, kjer se kljub temu pojavi PN:
  - 4 – 40% (odvisno od programa), povprečno pa 19%
  - ukazov load je v povp. 24%
  - 0,19*0,24=0,0456
  - CPI = (1 – 0,0456)*1 + 0,0456*2 = 1,0456
  - zaradi PN pri load je cevovod za 4,6% počasnejši

Kontrolne nevarnosti

KN: pri ukazih, ki spremenijo PC drugače kot PC ← PC + 4

- to so kontrolni ukazi oz. skoki:
  - brezpogojni skoki, pogojni skoki, klici (procedur), vrnitve
  - J, BEQ, BNE, CALL, TRAP, RFE
- skočni naslov se prenese v PC (v stopnji EX, razen pri TRAP (WB))
- KN: Kadar se v stopnji EX spremeni PC, je vsebina stopenj IF in ID neveljavna!
  - v njiju sta ukaza, ki sledita skočnemu ukazu
  - ne smeta se izvršiti
  - najenostavnejša rešitev je vstavljanje mehurčka v IF in ID

Odpravljanje kontrolnih nevarnosti

Prvi korak je zmanjšanje skočne zakasnitve:

1. *Preverjanje pogoja za skok* naj se izvaja čim bližje prvi stopnji cevovoda
2. *Izračun skočnega naslova* naj se izvaja čim bližje prvi stopnji cevovoda

- HIP: preverjanje skočnega pogoja je v BEQ in BNE
  - računanje skočnega naslova je možno v že stopnji ID
    - BEQ in BNE uporabljata PC-relativno naslavljanje, vrednost PC pa je že v reg. PC1
  - preverjanje pogoja šele v stopnji EX (komparator za B=0)

Vstavljanje mehurčkov

- Najenostavnejša (a bolj slaba) rešitev je vstavljanje mehurčka v IF in ID
- V tem primeru se v primeru skoka razveljavi ukaza v IF in ID (z mehurčki), če pa se skok ne izvede, pa deluje normalno naprej brez ustavljanja
  - **skočna zakasnitev** (branch delay), čakanje 2 periodi (pri TRAP izjemoma 5 period)
- Vsak skočni ukaz (z izjemo TRAP) povzroči 2 čakalni periodi

	Urina perioda									
Št. ukaza	1	2	3	4	5	6	7	8	9	10
Skočni ukaz	IF	ID	EX	MEM	WB					
Skočni ukaz + 1		IF	○	IF	ID	EX	MEM	WB		
Skočni ukaz + 2					IF	ID	EX	MEM	WB	
Skočni ukaz + 3						IF	ID	EX	MEM	WB
Skočni ukaz + 4							IF	ID	EX	MEM

- če pogoj za skok ni izpolnjen, ni čakanja
- cevovod predpostavi, da skoka ne bo

- V povprečju:
  - pogojnih skokov 12,5%
    - pogoj izpolnjen pri ~ 2/3 primerov
  - brezpogojnih skokov 2,5%
- Sprememba PC v stopnji EX:
  - 0,125*2/3 + 0,025 = 0,109
  - pri 10,9% ukazov je CPI = 3, sicer 1
  - CPI = 3*(0,109) + 1*(1 - 0,109) = 1,218
    - tj. več kot 20% izguba (daljši čas računanja)
  - Pri rač. z dolgimi cevovodi in pri CISC so izgube še večje
- KN so najhujše od 3 vrst nevarnosti

Drug način je predikcija izpolnitve skočnega pogoja in napoved skočnega naslova (če se skok izvede)

- branch predictor je vezje, ki napoveduje (ne)izpolnjenost pogoja

2 skupini:

- s statično predikcijo
- z dinamično predikcijo

Statična predikcija z zakasnjenimi skoki

Prevajalnik skuša napovedati bolj verjeten rezultat preverjanja skočnega pogoja

- med izvrševanjem programa se zato ne spreminja → statična predikcija
- tudi že omenjeni primer (ki predpostavi neizpolnjenost pogoja) je preprost primer statične predikcije
- **skočne reže** (branch slots)
  - v njih so ukazi, ki so v programu za skokom
  - enako številu stopenj cevovoda, ki so pred stopnjo, v kateri se v PC zapiše skočni naslov
    - pri HIP je to EX, pred njo sta 2 stopnji (zato 2 skočni reži)

Pri uporabi zakasnjenih skokov se (ne glede na izpolnjenost pogoja) izvršijo vsi prevzeti ukazi

- ukaza (oz. ukazi) v skočnih režah se ne nadomestita z mehurčki
- ker se vedno izvršita (izvršijo), izgleda kakor da se skok izvede kasneje

Primer:

Prvotno zaporedje ukazov		**Spremenjeno zaporedje ukazov pri zakasnjenih skokih**		
XOR	R1,R2,R3	XOR	R1,R2,R3	
ADD	R4,R6,R7	JMP	88(R20)	
SUB	R8,R5,R10	ADD	R4,R6,R7	skočna reža 1
JMP	88(R20)	SUB	R8,R5,R10	skočna reža 2
AND	R2,R2,R3	AND	R2,R2,R3	
OR	R7,R4,R9	OR	R7,R4,R9	

- Ukaza ADD in SUB se prestavita v skočni reži
- Pri desnem zaporedju ni čakalnih period

Pri pogojnih skokih je težje:

- ukaza, ki vpliva na pogoj, ne smemo dati v režo!
- namesto njega damo ukaz NOP (to dela prevajalnik)
  - prebere se iz ukaznega PP
  - možno je tudi, da bi bila oba ukaza NOP (lahko pa tudi nobeden)

Prvotno zaporedje ukazov	**Spremenjeno zaporedje ukazov pri zakasnjenih skokih**	
L1: XOR R1,R2,R3	L1: XOR R1,R2,R3	
ADD R4,R6,R7	SUB R8,R5,R10	
SUB R8,R5,R10	BEQ R8,L1	
BEQ R8,L1	ADD R4,R6,R7	skočna reža 1
AND R2,R2,R3	NOP	skočna reža 2
OR R7,R4,R9	AND R2,R2,R3	
	OR R7,R4,R9	

Meritve (na cevovodih z eno skočno režo):

- pri pogojnih skokih se koristno zapolni ~ 70% rež
- če upoštevamo še brezpogojne skoke (kjer so reže vedno koristno zasedene), se št. čakalnih urinih period zmanjša na 25%

Ugotovljeno je bilo, da se druga reža koristno zapolni 2x redkeje kot prva

- pri HIP bi pričakovali zmanjšanje čakalnih period na ~ 40%
- pri skokih se namesto 2 izgubi 0,8 periode

CPI:

$CPI_{idealni} = (1 – 0{,}109)*1 + 0{,}109*1{,}8 = 1{,}087$

- dosti bolje od 1,218
- če bi uporabili še razvelj. skoke, bi bilo še bolje
- pri dolgih cevovodih pa je koristno zapolniti reže težko

Statična predikcija

- prednost: večino dela opravi prevajalnik
- hiba: večino dela opravi prevajalnik
  - zahteva drugačno programiranje → problemi s kompatibilnostjo za nazaj

Danes se bolj uporabljajo strojni načini za dinamično predikcijo skokov

Dinamična predikcija skokov

Prilagaja se dogajanju v programu

Več vrst dinamične predikcije:

1. **1-bitna prediktorska tabela**
   - *prediktorska tabela* (branch prediction table, branch history table)
   - to je majhen pomnilnik, iz katerega se v stopnji IF bere vrednost (1 bit pri 1-bitni tabeli)
   - naslov določajo spodnji biti naslova ukaza
   - če je pogoj izpolnjen, se vpiše 1, sicer 0
     - v stopnji EX, ko je to znano

- služi kot napoved izpolnjenosti pogoja
- če je napovedan izpolnjen pogoj, potrebujemo še skočni naslov
  - ta je dostopen šele v stopnji ID
  - zato privarčujemo le en urino periodo (pri HIPu)
  - če je bila napoved napačna (izvemo v EX), je treba v IF in ID vstaviti mehurčke
- metoda ni posebno zanesljiva
  - npr. pri vgnezdenih zankah bo napoved tipično napačna dvakrat

2. 2-bitna prediktorska tabela

- 4 vrednosti (0..3)
- Povečanje ali zmanjšanje za 1
- 0 in 1: neizpolnjen pogoj, 2 in 3: izpolnjen
- Pri vgnezdenih zankah le 1 napačna napoved

Možna tudi n-bitna prediktorska tabela, n > 2

- Vendar ni dosti boljša kot 2-bitna
- Tabele so velikosti največ 4096 (12 bitov naslova)

3. Korelacijski prediktor

Correlating branch prediction table

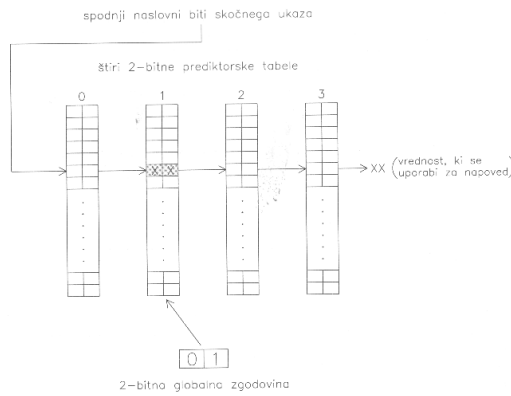
Primer:

```
if ( a == 2)
        a = 0;
if ( b == 2)
        b = 0;
if ( a != b) {
```

```
        SUBI    R3,R1,#2
        BNE     R3,L1           ; skok s1
        ADD     R1,R0,R0  ; a ← 0
L1:     SUBI    R3,R2,#2
        BNE     R3,L2           ; skok s2
        ADD     R2,R0,R0  ; b ← 0
L2:     SUBI    R3,R1,R2  ; R3 ← a – b
        BEQ     R3,L3           ; skok s3
```

- Skok s3 odvisen od s1 in s2
- Običajna prediktorska tabela tega ne more zajeti

- Korelacijski prediktor (m,n) uporablja obnašanje prejšnjih m skokov (t.i. *globalna* zgodovina), da izbere eno od 2^m n-bitnih predliktorskih tabel
  - Navadna 2-bitna tabela bi bila k.p. (0,2)
    - Imenuje se tudi *lokalni* prediktor
- Primer: korelacijski prediktor (2,2)
  - Za globalno zgodovino lahko uporablja 2-bitni pomikalni register (pomika v levo)



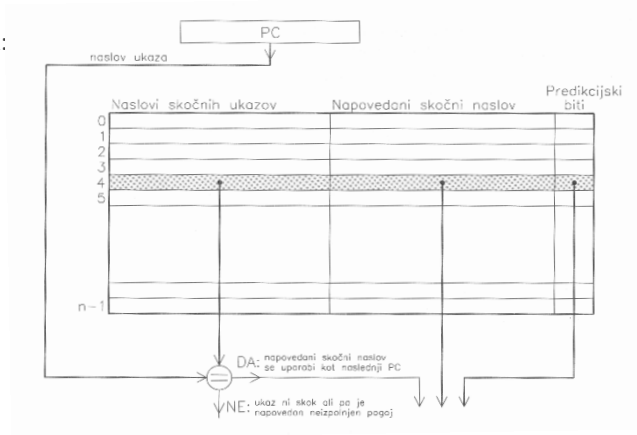
4. Turnirski prediktor

- tournament branch predictor
- Upošteva dejstvo, da globalni prediktor ni vedno boljši od lokalnega
- Paralelno delojoča lokalni in globalni prediktor tekmujeta
- Selektor določa, kateri bo uporabljen (glede na prejšnji uspeh)

5. Skočni predpomnilnik

- branch target buffer
- Tudi pri pravilni napovedi pogoja se vedno izgubi ena perioda
  - V stopnji IF ne poznamo skočnega naslova (ne poznamo niti ukaza, ker še ni dekodiran)
- Skočni PP vsebuje skočne naslove zadnjih skokov, pri katerih je bil pogoj izpolnjen
  - Naslovi se vanj shranijo v stopnji EX
- V IF se poleg ukaza bere tudi skočni PP
  - V primeru zadetka (in potencialnih prediktorskih bitov) se skočni naslov takoj vpiše v PC
  - Pri pravilni napovedi ni treba čakati 1 periodo
  - Pri napačni napovedi (ali skočnem naslovu) je treba vstavljati mehurčke

Skočni predpomnilnik:



Skočni PP je bolj zapleten od prediktorjev na osnovi tabel

- Npr. PP 1024x32 potrebuje 1024 32-bitnih komparatorjev (primerjalnikov)

V skočni PP se shrani skočni naslov le, kadar je pogoj izpolnjen

- Sicer je naslov poznan (naslednji po vrsti)

6. Vrnitveni prediktor

- return address predictor
- Težavna vrsta posrednih skokov
- Ista procedura se lahko kliče z zelo različnih mest v programu (npr. funkcija printf v C-ju)
  - Težko napovedati
- Običajno majhen sklad (npr. 16 naslovov)

7. Enota za prevzem ukazov

- integrated instruction fetch unit
- Današnji računalniki lahko istočasno prevzemajo in izvršujejo več ukazov (superskalarnost)
  - Prevzem ukazov bolj zapleten
- Enota deluje samostojno in dostavlja ukaze ostalim stopnjam
  - Dela tudi predikcijo, dostopa do PP, pri zgrešitvah menjava bloke v PP, ...

Vključitev dinamične predikcije v HIP

HIP ima skočno zakasnitev 2 urini periodi

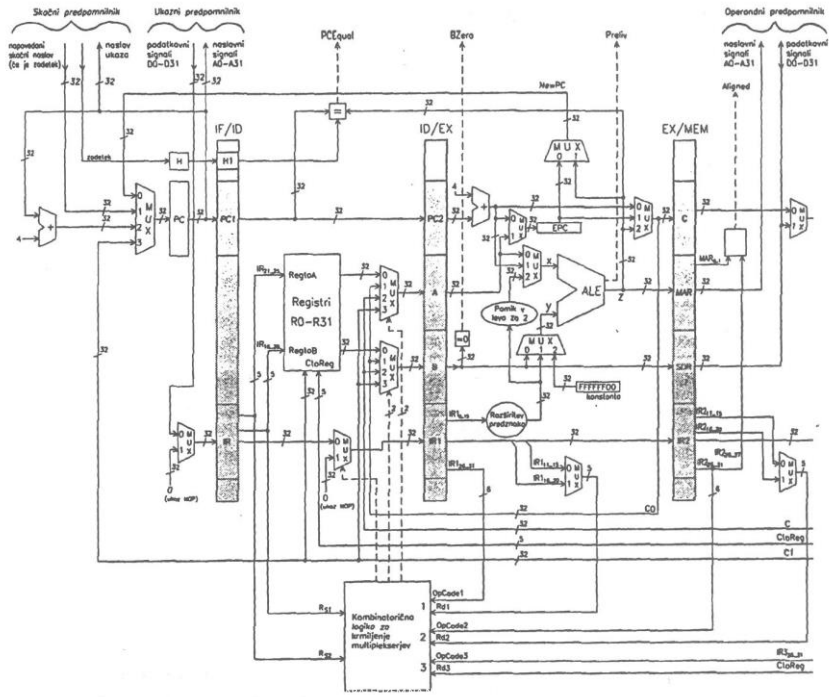
Odločimo se npr. za skočni predpomnilnik (SPP) z enim prediktorskim bitom

- ta bit prepreči, da bi se informacija o skoku izbrisala iz PP že ob prvi napačni napovedi

V stopnji IF gre vsebina PC v ukazni in v SPP

Če je v SPP zadetek, se napovedani naslov iz njega prenese v PC

- s tega naslova naj bi stopnja IF prevzela naslednji ukaz
- skočni ukaz pa se izvršuje naprej (v ID in EX), ker napoved morda ni pravilna



Modifikacije podatkovne enote:

- mux 4/1 (namesto 3/1) zaradi SPP
- 1-bitni register H
  - ob zadetku se vanj vpiše 1, ob zgrešitvi 0
  - H je potreben zato, ker se mora v EX preveriti pravilnost skočnega naslova (ob zadetku se je vpisal v PC)
- komparator v ID
  - primerja napovedani (v PC1, stopnja ID) in izračunani (na izhodu ALE, stopnja EX) skočni naslov
  - njegov izhod je PCEqual
    - 0 pri neenakosti, če je hkrati H1=1
    - 1 sicer
  - kadar je PCEqual=1, je bila napoved naslova pravilna
    - skočnemu ukazu v EX se ne dovoli pisanje v PC
  - sicer je bila napačna
    - tedaj sta napačna tudi ukaza v IF in ID
    - v IF in ID se vstavita mehurčka
    - v PC se prenese NewPC = PC2 + 4 + $razIR1_{0..15}$
  - v stopnji EX se mora osvežiti informacija v skočnem PP (v primeru skoka)
    - način odvisen od
      - izpolnitve pogoja
      - zadetka v skočnem PP (H1)
      - prediktorskega bita

1. Skočni pogoj je izpolnjen
   - v primeru zadetka v SPP se prediktorski bit postavi na 1
   - sicer se naslov skočnega ukaza in izračunani skočni naslov shranita v SPP (predikt. bit 1)
2. Skočni pogoj ni izpolnjen
   - v primeru zadetka v SPP
     - in predikt. bita 0, se info. o tem skočnem ukazu izbriše iz SPP
     - in predikt. bita 1, gre le-ta v 0, info. pa ostane v SPP
   - sicer nič

- stanje predikt. bita se torej uporablja le pri odločitvi o brisanju info. iz SPP
- zadetek v SPP pri HIPu pomeni, da je napovedan izpolnjen pogoj
  - ne glede na stanje predikt. bita

Uspešnost dinamične predikcije pri HIP

- verjetnost zadetka v SPP H_c naj bo 90%
  - za to zadošča že majhen SPP, velikosti 64
- verjetnost, da zadetek v SPP pomeni pravilno napoved, H_p naj bo 92%
- pogoj za skok naj bo izpolnjen z verjetnostjo 67,3%

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

Čakalne periode $_{\text{pogojni skok}}$

$= H_c * (1-H_p) * 2 + (1-H_c) *$ verj. za izpolnjen skočni pogoj $* 2$

$= 0,9*0,08*2 + (1-0,9)*0,673*2$

$= 0,279$

Čakalne periode $_{\text{brezpogojni skok}} = (1-H_c)*2 = (1-0,9)*2 = 0,2$

Čakalne periode $_{\text{skok}} = \frac{0,125 \times 0,279 + 0,025 \times 0,2}{0,125 + 0,025} = 0,266$

$CPI_{\text{idealni}} = (1 - 0,15)*1 + 0,15*1,266 = 1,04$

4% je približno 2x manj kot pri statični predikciji

Škoda zaradi zgrešitev v PP je znatno večja (v gornjih enačbah ni upoštevana)

Prekinitve in pasti pri cevovodu

Kdaj skočiti na servisni program?

- istočasno se izvaja več ukazov
- delno izvršeni ukazi lahko povzročijo napake

3 primeri

1. **Vhodno/izhodne prekinitve**
   - običajno je, da cevovod izvrši ukaze (ki so že v njem) do konca
     - V/I prekinitve so razmeroma redki dogodki, zato izguba ni velika
   - pri HIP je prekinitveno-prevzemni cikel treba izvesti izven cevovoda (sicer bi rabili 6 stopenj v cevovodu)

2. **Programske pasti**
   - TRAP je v bistvu kot klic procedure
     - poseben brezpogojni skok

3. **Pasti, do katerih pride med izvrševanjem ukaza**
   - najtežje
   - zgodijo se na sredi ukaza
     - ukaz se ne more dokončati
     - potrebno ga je ustaviti, izvršiti servisni program in ga ponovno začeti
       - treba je tudi paziti, da del ukaza, ki se je (bil) že izvršil, ne povzroči napake

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

- napaka strani: pri navideznem (virtualnem) pomnilniku, kadar stran ni (fizično) v GP (ne gre za resnično napako)
- zaščita pomnilnika: dostop do naslova, ki ne pripada programu
- pri napaki strani se po servisiranju program nadaljuje na prekinjenem mestu
- pri ostalih pasteh se običajno zaključi z diagnostičnim sporočilom

Operacije, ki trajajo več urinih period

Ko ukaz s tako operacijo pride v stopnjo EX, se cevovod ustavi in čaka, da se operacija izvrši

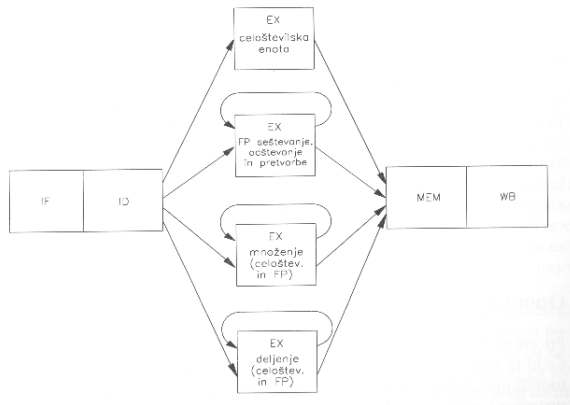
- cevovod bi pri mnogih programih postal prepočasen

Zato so uvedli funkcijske enote:

- **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

Predpostavimo, da FE niso cevovodne

- naslednji ukaz lahko uporabi neko enoto šele, ko jo prejšnji zapusti (strukturne nevarnosti)
- samo celošt. enota rabi 1 periodo, ostale več

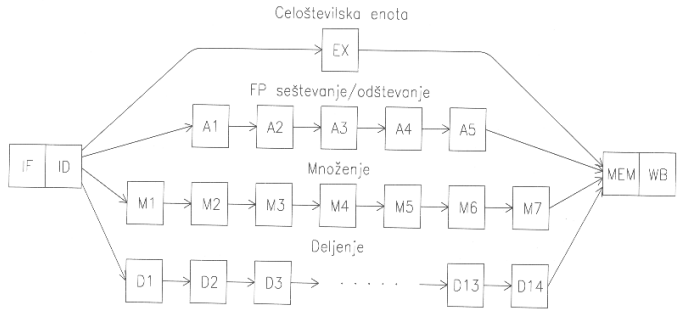


Če so FE cevovodne (danes običajno), lahko odpravimo strukturne nevarnosti v ID in EX

- lahko pa se SN pojavijo v MEM in WB

Dodatni problemi:

- V MEM in WB pride hkrati lahko več rezultatov
  - reg. blok mora omogočati več pisanj vanj hkrati
- poveča se tudi verjetnost podatkovnih nevarnosti
  - v MEM in WB prihajajo ukazi v spremenjenem vrstnem redu
  - pojavijo se PN tipov WAW in WAR



3 vrste PN:

- **RAW** (read after write): ukaz *j* bere operand, preden ga ukaz *i* shrani
- **WAR** (write after read): ukaz j piše v reg., še preden ga *i* prebere
- **WAW** (write after write): ukaz j piše v reg., preden vanj piše *i*
- RAR ne more povzročiti PN

Pri HIP je edina možnost RAW

- s premoščanjem jo običajno odpravimo (razen pri load)

Primer: zaporedje ukazov v plavajoči vejici

- enota (FPU) ima kar svojo množico registrov
  - poenostavi ugotavljanje nevarnosti
  - to rešitev uporablja večina CPE

	Urina perioda																
Ukaz	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18
FLD F4,0(R2)	IF	ID	EX	MEM	WB													
FMUL F0,F4,F6		IF	ID	○	M1	M2	M3	M4	M5	M6	M7	MEM	WB					
FADD F2,F0,F8			IF	○	ID	○	○	○	○	○	○	A1	A2	A3	A4	A5	MEM	WB
FST 0(R2),F2					IF	○	○	○	○	○	○	ID	EX	○	○	○	○	MEM

Odpravljanje podatkovnih nevarnosti z dinamičnim razvrščanjem

Dinamično razvrščanje:

- strojna sprememba vrstnega reda izvrševanja ukazov (da se zmanjša št. čakalnih period)

Primer PN:

- čakanje na "počasen" ukaz, ki se izvršuje v neki FE
- npr.:

```
FDIV  F0,F5,F6        ; F0 ← F5/F6
FADD F4,F0,F2         ; F4 ← F0 + F2
FSUB  F8,F2,F1        ; F8 ← F2 – F1
```

  - ukaz FSUB mora čakati (cevovod se ustavi zaradi odvisnosti med FDIV in FADD)
  - ker pa je FSUB neodvisen od prejšnjih ukazov, ga lahko pomaknemo gor (da se izogne čakanju)

ID moramo razdeliti na 2 stopnji:

1. Izstavljanje (issue)
   - dekodiranje
   - ugotavljanje SN
     - pri SN izvrševanje ukaza ni možno (ne glede na PN)
2. Branje operandov
   - ugotavljanje PN
     - v primeru nevarnosti se čaka
     - v tej stopnji lahko pride do spremembe vrstnega reda ukazov

Spremenjen vrstni red izvrševanja lahko pripelje do PN tipa WAR in WAW

Tomasulov algoritem (1967)

Podatkovne odvisnosti lahko delimo na
- prave podatkovne odvisnosti
  - ukaz potrebuje kot vhodni operand rezultat enega od prejšnjih ukazov
- imenske odvisnosti


```
FDIV    F0,F5,F6        ; F0 ← F5/F6
FADD    F4,F0,F2        ; F4 ← F0+F2
FST     0(R1),F4        ; M[R1] ← F4
FSUB    F2,F3,F7        ; F2 ← F3-F7
FMUL    F4,F3,F2        ; F4 ← F3*F2
```

- imenske odvisnosti
  - med FADD in FSUB zaradi R2
    - nevarnost WAR (*antiodvisnost*)
  - med FADD in FMUL zaradi F4
    - nevarnost WAW (*izhodna odvisnost*)
- prave podatkovne odvisnosti
  - med FDIV in FADD
  - med FADD in FST
  - med FSUB in FMUL

Imenske odvisnosti lahko vedno odpravimo s preimenovanjem registrov (če imamo na voljo dodatne registre)

```
FDIV   F0,F5,F6      F0 ← F5/F6
FADD   FT2,F0,F2     FT2 ← F0+F2
FST    0(R1),FT2     M[R1] ← FT2
FSUB   FT1,F3,F7     FT1 ← F3-F7
FMUL   F4,F3,FT1     F4 ← F3*FT1
```

Tomasulov algoritem pa odpravi nevarnosti, ki izvirajo iz imenskih odvisnosti (WAR in WAW), brez preimenovanja registrov

Špekulativno izvajanje ukazov

Pri dinamičnem razvrščanju ukazov se problemi zaradi KN zelo povečajo

- ker se v vsaki periodi izvršuje več ukazov, je v primeru napačne predikcije težko ugotoviti, kateri se morajo razveljaviti
- cevovod se mora ustavljati

Špekulativno izvajanje ukazov (speculative execution)

- predpostavi se, da je napoved skokov z dinamično predikcijo pravilna
- potreben pa je mehanizem, ki v primeru napačne napovedi odstrani vse, kar so naredili napačno napovedani ukazi
  - izvršitev ukaza ne sme vplivati na registre, dokler ni potrjena pravilnost napovedi skoka
  - **preureditveni izravnalnik** (reorder buffer, ROB)
    - začasno hrani rezultate ukazov

Preureditveni izravnalnik je realiziran kot FIFO vrsta v obliki krožnega bufferja
- ukazi so v njem v pravilnem vrstnem redu

Vsako polje v njem ima 4 parametre:
1. vrsta ukaza
   - skoki, store ali registrski ukazi
2. ponor
   - register ali pomnilniška beseda
3. vrednost
   - rezultat ukaza, ki naj se shrani
4. veljavnost
   - 1, če je v parametru vrednost že rezultat ukaza
   - 0, če se na rezultat ukaza še čaka

Velikost preureditvenega izravnalnika se imenuje *ukazno okno* (instruction window)
- določa, koliko ukazov se lahko izvede špekulativno
- če je velika, se porabi več energije za izbris vsega izračunanega

Večizstavitveni procesorji

Približevanje CPI vrednosti 1
- dinamična predikcija skokov
- dinamično razvrščanje
- špekulativno izvrševanje ukazov

CPI < 1
- v vsaki urini periodi se mora prevzeti in izstaviti več kot 1 ukaz
- običajno se uporablja IPC = 1 / CPI
- **večizstavitveni procesorji** (multiple issue processors)

Vidiki prevzema in izstavljanja ukazov

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 $(2(n-1) + 2(n-2) ...)$

Strojno ugotavljanje podatkovnih odvisnosti je zahtevno za realizacijo, zato sta se pojavili 2 rešitvi:

1. Superskalarnost
2. VLIW

Superskalarni procesorji

Dinamično določanje, kateri ukazi se v dani periodi ure izstavijo
- če se jih lahko izstavi največ n, je to n-kratni superskalarni procesor (n-way superscalar processor)

n(n-1) primerjav je težko izvesti v eni periodi
- pri superskalarnih procesorjih se primerjave razdeli med več stopenj cevovoda

Ukazi se sicer izvajajo špekulativno
- le da jih je več hkrati
- Št. FE običajno > n, da se zmanjšajo SN

Potrebujejo pa večjo zmogljivost:
- prenosnih poti,
- preureditvenega izravnalnika,
- dostopa do registrov

Najbolj zapleteni del superskalarnega procesorja je ROB

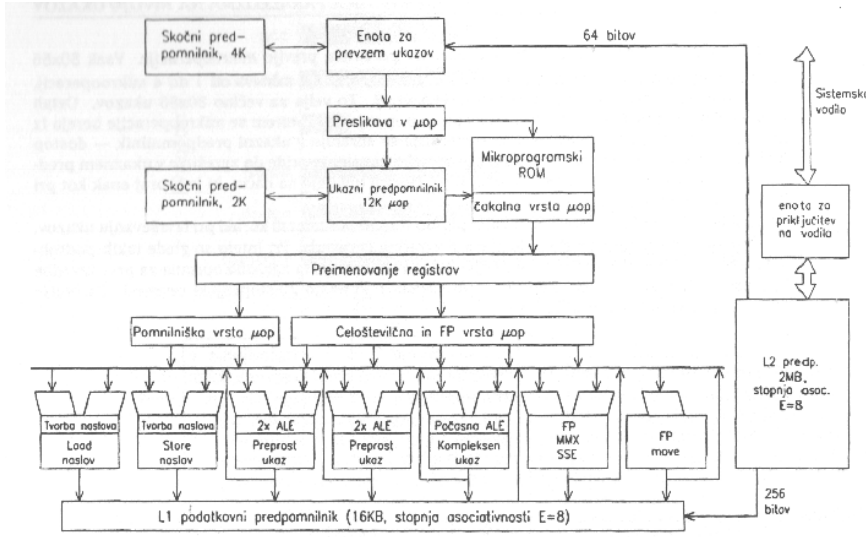
Zato procesorji po letu 2000 uporabljajo **eksplicitno preimenovanje registrov**

- preureditveni izravnalnik je preprostejši
  - skrbi le za vrstni red ukazov, ne pa tudi za operande iz registrov
- procesor ima še dodatne registre
  - *razširjena množica registrov* (lahko tudi nekaj sto)
  - *preimenovalna tabela* določa, kateri so v neki periodi programsko dostopni
- korak izstavljanja je drugačen:
  - Iz čakalne vrste se vzame *n* ukazov
  - Izhodni register vsakega ukaza se preimenuje v enega od prostih registrov
    - S tem se odpravijo nevarnosti WAW in WAR, ki izvirajo iz imenskih odvisnosti
  - Preveri se medsebojna odvisnost operandov (kot že prej opisano)
    - Po potrebi se popravijo številke vhodnih registrov
  - Ukazi se prenesejo v ROB
    - Globina se določi na osnovi števila registrov
  - Ukazi se prenesejo v FE

Primer superskalarnega procesorja: Pentium 4

- mikroarhitektura NetBurst
- 7 FE:
  - load
  - store
  - preproste celoštevilske operacije (x2)
  - zahtevne celoštevilske operacije
  - FP operacije
  - prenosi FP operandov iz/v pomnilnik

Pentium 4



Procesorji VLIW

Procesorji VLIW (very long instruction word) imajo dolge ukaze

- Vsebujejo več običajnih ukazov, ki se lahko izvršujejo paralelno
  - Npr. da vsak zaposli eno FE
- Tipičen ukaz:
  - 3 celošt. ukazi
  - 2 FP ukaza
  - 2 pomnilniška dostopa
  - 1 skok
- CPE ne ugotavlja odvisnosti in nevarnosti
  - To je delo prevejalnika
  - Če ne uspe najti dovolj neodvisnih ukazov za vse enote, se nekaterim FE da ukaz NOP

Potencialne prednosti VLIW:
- Prevajalnik vidi celoten program
  - Zato lahko odkrije več paralelnosti kot logika v procesorju, ki vidi le ukazno okno
  - Odkrivanje paralelnosti se izvede samo enkrat
- Procesor je lahko preprostejši
  - Ne rabi logike za odkrivanje paralelnosti
  - Zato je frekvenca ure lahko višja

Digitalno procesiranje signalov
- Veliko paralelnosti

EPIC (explicitly parallel instruction computing)
- Intel 1997
- *predikatni ukazi*
- Itanium 1 (2000), 2 (2002)

Omejitve paralelizma na nivoju ukazov

Količina paralelnosti v programih je omejena
- S povečevanjem količine logike lahko pridobimo le do neke meje

Koliko paralelnosti na nivoju ukazov je v nekem programu?
- zamislimo si idealni superskalarni procesor
- lastnosti:
  1. ni strukturnih nevarnosti
     - neomejeno število registrov za preimenovanje
     - neomejeno število FE, vse izvršijo operacijo v 1 periodi
     - torej se v 1 periodi lahko izstavi in izvrši neomejeno število ukazov
  2. ni kontrolnih nevarnosti
     - popolno napovedovanje skokov (vsi napovedani 100%)
     - neomejeno ukazno okno
     - do izbrisa zaradi napačne špekulacije nikoli ne pride

3. naslovi vseh pomnilniških operandov znani vnaprej
   - ukazi load se lahko prestavijo pred store (če ne gre za isti naslov)
4. predpomnilniki nimajo zgrešitev
   - vsi pomnilniški dostopi trajajo 1 periodo

- ostanejo le prave podatkovne nevarnosti

- izvajamo različne programe in merimo dosegljivi IPC
  - na 6 programih iz SPEC92
    - IPC od 18 do 150
    - povprečni IPC okrog 80
  - z upoštevanjem bolj realnih lastnosti dosegljivi IPC pade na okrog 5
  - realni IPC pa je manjši

Paralelizem na nivoju niti

Paralelizem na višjem nivoju, ki ga na nivoju ukazov ni mogoče izkoristiti
- izvrševanje se razdeli v več neodvisnih poti (niti)
  - thread-level parallelism
- Pri večnitnosti (multithreading) si niti delijo FE enega procesorja
- vsaka nit ima svoje stanje
  - ločeno in neodvisno od drugih niti
  - nit ima svojo kopijo registrov, svoj PC, svoje tabele strani in nekatere programsko nevidne registre
- niti pa si delijo GP in PP
- nit vidi procesor, kakor da je namenjen le njej sami
  - en fizični procesor je videti kot več *logičnih procesorjev*
- problem: niti je treba definirati (paralelno programiranje)
  - eksplicitni paralelizem
  - obstoječe programe je (bi bilo) potrebno predelati!

Več oblik večnitnosti:

1. **Časovna večnitnost** (temporal multithreading)
   - preklapljanje, niti se izmenjujejo
   a. *Drobno-zrnata večnitnost*
      - preklop med nitmi vsako urino periodo
      - treba je shraniti celotno stanje cevovoda
      - če bi posamezna nit morala čakati, se jo v tem ciklu izpusti (da se ne izgublja časa)
      - hiba je upočasnitev posameznih niti
   b. *Grobo-zrnata večnitnost*
      - preklop samo, kadar pride pri niti do daljšega čakanja
      - ni treba shraniti stanja cevovoda (čakamo, da se izprazni)
2. **Istočasna večnitnost** (simultaneous multithreading, SMT)
   - pri večizstavitvenih procesorjih
     - Intel Pentium 4: Hyper-threading (običajno 2 niti)
   - ni potrebno veliko sprememb
   - prednost: ni medsebojnih odvisnosti
   - hiba: v določenih primerih se zmogljivost tudi poslabša
     - programerji morajo preverjati, ali se pri neki aplikaciji SMT obnese, ali ne

Večjedrni procesorji (multicore)

- več CPE (jeder) na istem čipu
- pogosto imajo CPE svoje PP L1, L2 in višje pa si delijo
  - zato CPE niso čisto neodvisne
  - jedra so običajno tudi večnitna (pogosto dvonitna)
- množična proizvodnja večjedrnih procesorjev
  - predvsem v interesu proizvajalcev
    - ceneje kot vlagati v razvoj novih rešitev
  - uporabniki redko lahko uporabijo veliko število jeder
  - Npr., procesor z IPC = 4 bi bil verjetno bolj koristen kot 8-jedrni
  - “uporabniki se bodo pač morali naučiti pisanja večnitnih programov” ?!

Primer:

- Intel Core 2 Quad