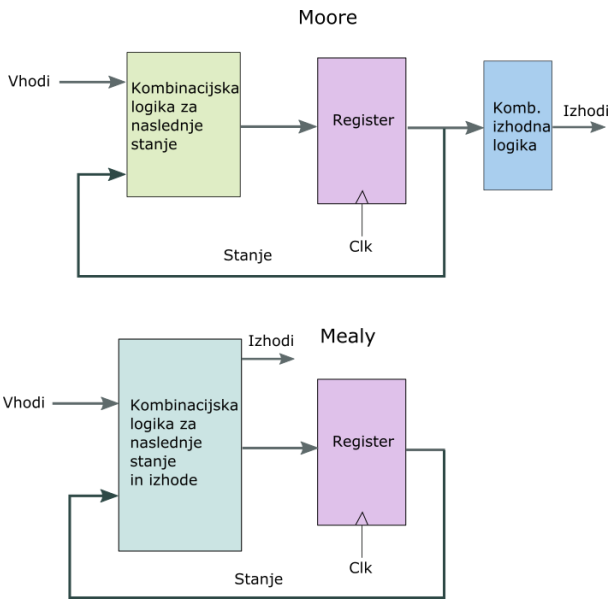
6 Centralna procesna enota

Splošno

- CPE je digitalen sistem.
- Vsebuje **kombinacijska** in **sekvenčna** digitalna vezja
- **Stanje CPE:**
  - stanje sekvenčnih (pomnilnih) elementov
- Delovanje CPE je odvisno od
  - trenutnega stanja in
  - trenutnih vhodov

Konceptualna predstavitev sinhronskih digitalnih vezij



Delovanje CPE

1. **Dobava ukaza iz pomnilnika** (fetch)
2. **Izvrševanje ukaza**
   a) dekodiranje ukaza
   b) prenos operandov v CPE (po potrebi)
   c) izvedba operacije
   d) shranjevanje rezultata (po potrebi)
   e) PC ← PC + 1 (razen pri skokih)

- Ta cikel dveh korakov se ponavlja, dokler računalnik deluje
  - Izjema so prekinitve (interrupt) in pasti (trap)
    - skok na nek drug ukaz

- Vsak od 2 korakov je sestavljen iz bolj elementarnih korakov
  - vsak traja eno ali več period ure CPE
  - urin signal
- Pri sinhronih sekvenčnih vezjih se spremembe stanja prožijo ob aktivni fronti ure (prednja ali zadnja)
  - perioda ure CPE (t_{CPE}) je čas med dvema sosednima frontama

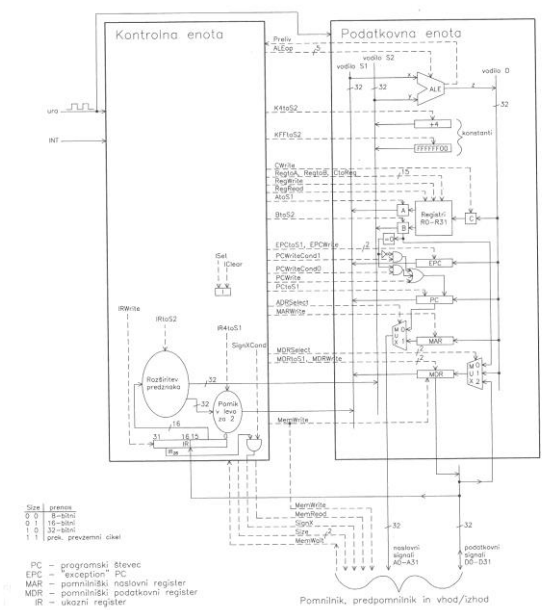
$f_{CPE} = 1/t_{CPE}$

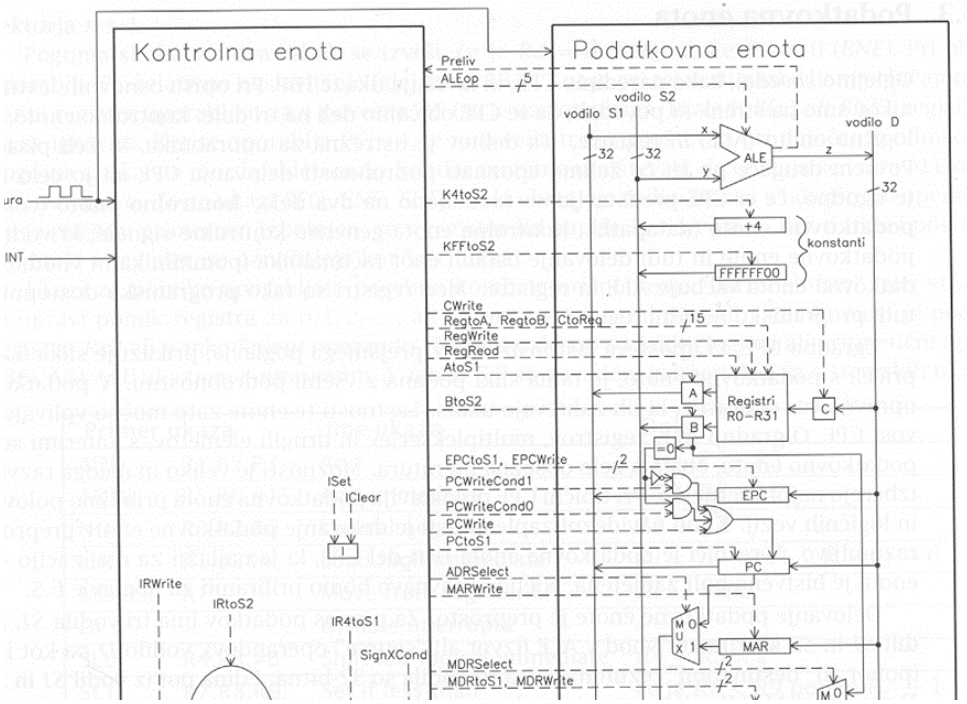
- Perioda je navzdol omejena z zakasnitvami kombinacijskih vezij
  - če bi bila krajša, se novo stanje ne bi imelo časa vzpostaviti
  - zato je frekvenca omejena navzgor
- Opcija so tudi asinhronska sekvenčna vezja
  - hitrejša (ni ure)
    - MIPS R3000 4x hitrejši kot sinhronski
  - težavna za načrtovanje

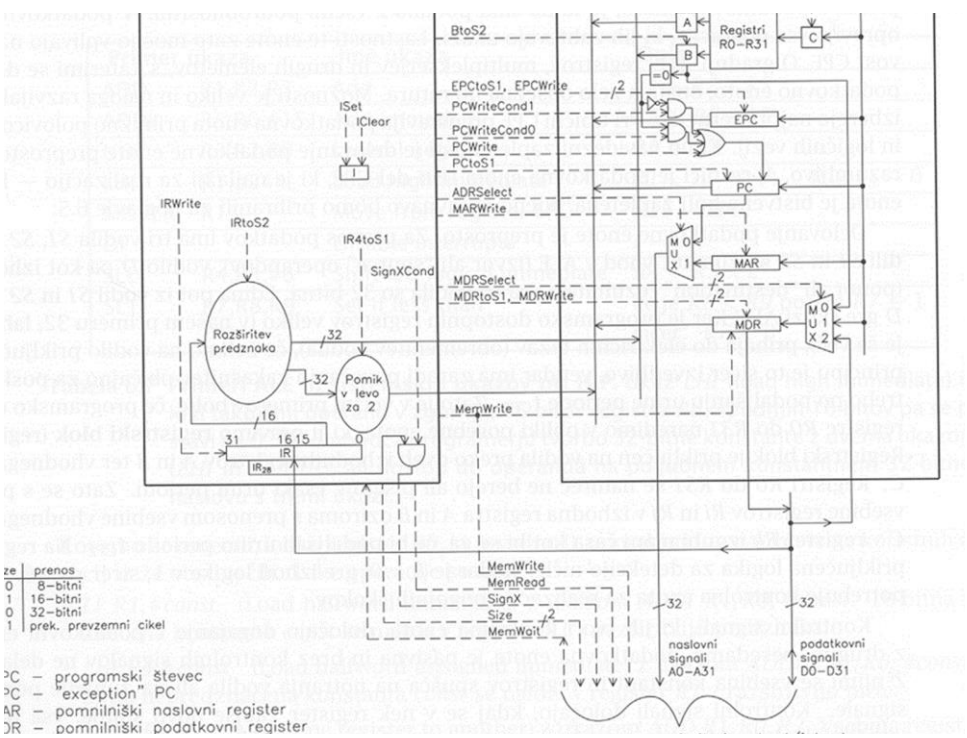
Podatkovna enota

- CPE lahko razdelimo na dva dela:
  - **kontrolna enota** (control unit), KE
    - generira kontrolne signale, ki vodijo delovanje (podatkovne enote, pomnilnika, V/I)
  - **podatkovna enota** (datapath), PE
    - ALE, registri

CPE (HIP)




Kontrolna enota
Podatkovna enota
Preliv
ALEop
5
vodilo S1
vodilo S2
vodilo D
ALE
32
K4toS2
+4
konstanti
KFFtoS2
FFFFFF00
CWrite
RegtoA, RegtoB, CtoReg
15
RegWrite
RegRead
AtoS1
BtoS2
Registri R0–R31
EPCtoS1, EPCWrite
PCWriteCond1
PCWriteCond0
PCWrite
PCtoS1
EPC
PC
ISet
IClear
IRWrite
IRtoS2
IR4toS1
SignXCond
ADRSelect
MARWrite
MAR
MDRSelect
MDRtoS1, MDRWrite
INT


BtoS2
Registri R0–R31
EPCtoS1, EPCWrite
PCWriteCond1
PCWriteCond0
PCWrite
PCtoS1
EPC
PC
ISet
IClear
IRWrite
IRtoS2
IR4toS1
SignXCond
ADRSelect
MARWrite
MAR
MDRSelect
MDRtoS1, MDRWrite
MDR
Razširitev predznaka
Pomik v levo za 2
MemWrite
IR
MemRead
SignX
Size
MemWait
naslovni signali A0–A31
podatkovni signali D0–D31
prenos
8-bitni
16-bitni
32-bitni
prek. prevzemni cikel
– programski števec
– "exception" PC
– pomnilniški naslovni register
– pomnilniški podatkovni register

Podatkovna enota

- 3 (32-bitna) vodila (za prenos podatkov):
  - *S1* in *S2:* vhod v ALE
  - *D:* izhod iz ALE
- Vse ALE operacije se izvršijo v eni urini periodi
- 32-bitni konstanti, priključeni na *S2*:
  - +4 (za povečevanje *PC*) in
  - FFFFFF00 (za izjeme)

Registrski blok

- Registrski blok (register file): registri *R0* do *R31*
  - priključen na vodila preko izhodnih registrov A in B ter vhodnega registra C
  - če bi registre *R0* do *R31* direktno priključili na vodilo, bi dobili večje zakasnitve in s tem daljšo urino periodo t_{CPE}
  - na register B je priključena logika za detekcijo ničle
- Kontrolni signali za registrski blok
  - RegtoA izbere register, ki se bo prebral v A, RegtoB ...
  - CtoReg določa register, kamor se bo vpisala vsebina C
  - prenos sprožita signala
    - RegRead (prenos v A in B) in RegWrite (prenos iz C)
  - pisanje ob aktivni fronti ure
- Signali AtoS1, BtoS2, PCtoS1 prenašajo vsebine registrov A, B in PC 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
- MDRSelect podobno
- 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

PC in EPC

- Za uporabnika sta vidna še programski števec PC in EPC
  - PCtoS1 prepušča vsebino PC na vodilo S1
  - PCWrite sproži pisanje iz vodila D v PC
  - izhod PC je priključen tudi na MUX, preko katerega lahko pride na pomnilniške naslovne signale A0-A31
- EPC shranjuje PC ob prekinitvah in pasteh (izjeme - exceptions)
  - EPCtoS1, EPCWrite
  - 1-bitni register I je v kontrolni enoti (pri I = 0 so prekinitve onemogočene)

MAR in MDR

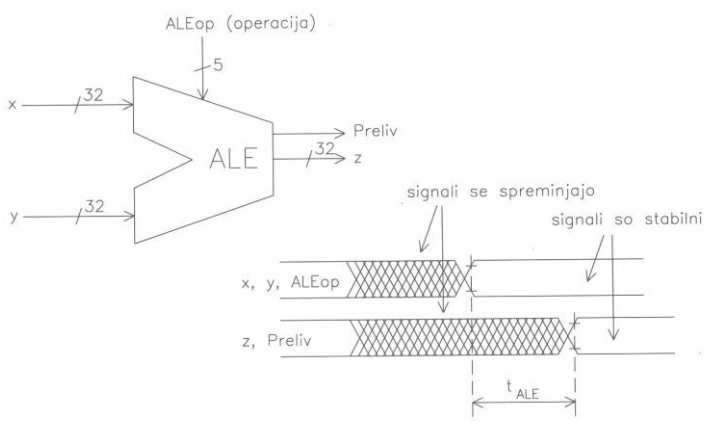
- MAR (Memory address register) in MDR (Memory data register) služita komunikaciji s pomnilnikom
  - MDR je vezan na S1 in na pomnilniške podatkovne signale D0-D31
    - MDRtoS1
    - MemWrite
  - Pisanje v MDR (MDRWrite). Z MDRSelect (2 bita) izberemo vhod
    - pomnilnik (D0-D31)
    - vodilo D
  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

- Ukazni register IR je v kontrolni enoti
- Zakaj imamo 3 vodila?
  - želimo, da se ALE operacije izvršijo v eni periodi ure
- Zakaj pa imamo sploh vodila in ne kar dvotočkovnih povezav?
  - porabili bi mnogo več elementov in s tem prostora na čipu

ALE

- ALE operacija se izvrši na 32-bitnih vhodnih operandih *x* in *y*, ki sta na vodilih *S1* in *S2*
- Rezultat je 32-bitni izhod *z* (pojavi se na vodilu *D*) in *Preliv*, ki je speljan v kontrolno enoto
- Signali ALEop (5 bitov) pridejo iz kontrolne enote
  - ta jih tvori iz bitov operacijske kode (in njenega podaljška *func*) ukaza
  - pri HIP ukazih za spodnje 4 bite ALEop lahko uporabimo kar spodnje 4 bite operacijske kode (biti 26-29 v registru *IR*)

ALE



ALE operacije (1)

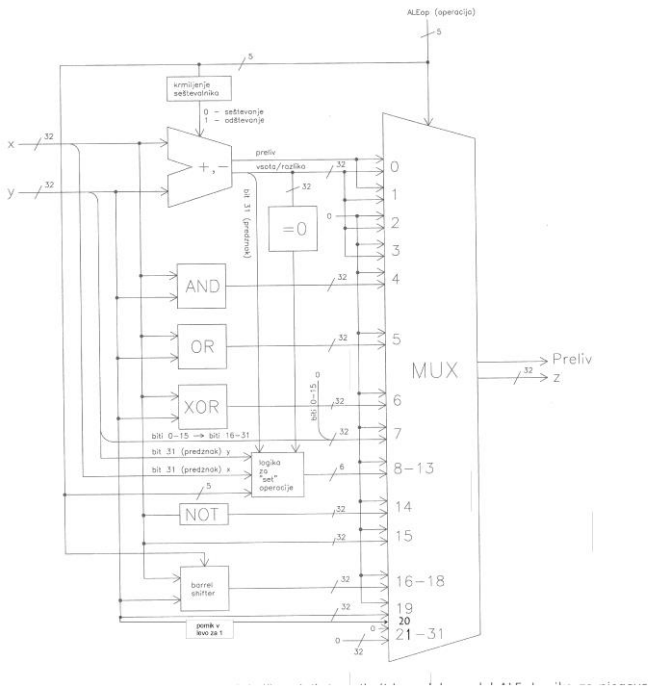
Št.	ALEop	Operacija	Opis
0	00000	ADD	z = x + y in Preliv
1	00001	SUB	z = x – y in Preliv
2	00010	ADDU	z = x + y
3	00011	SUBU	z = x – y
4	00100	AND	z = xy
5	00101	OR	$z = x \vee y$
6	00110	XOR	$z = x \oplus y$
7	00111	LHI	$z = y \times 2^{16}$
8	01000	SEQ	če je x = y, potem z = 1, sicer z = 0
9	01001	SNE	če je $x \neq y$, potem z = 1, sicer z = 0
10	01010	SLT	če je x < y, potem z = 1, sicer z = 0
11	01011	SGT	če je x > y, potem z = 1, sicer z = 0

ALE operacije (2)

Štev.	ALEop	Operacija	Opis
12	01100	SLTU	če je $x \leq y$, potem z = 1, sicer z = 0
13	01101	SGTU	če je $x \geq y$, potem z = 1, sicer z = 0
14	01110	NOT	z = not x
15	01111	S1 → D	z = x (edina pot iz S1 in S2 na D gre skozi ALE)
16	10000	SLL	z = x, logično pomaknjen za y mest v levo
17	10001	SRL	z = x, logično pomaknjen za y mest v desno
18	10010	SRA	z = x, aritmetično pomaknjen za y mest v desno
19	10011	S2 → D	z = y (edina pot iz S1 in S2 na D gre skozi ALE)
20	10100	2*S2 → D	z = 2y
21	10101	ZER	z = 0
22..31	10110 do 11111	rezervirano	z = 0

Najbolj zapleteni operaciji za realizacijo sta seštevanje in odštevanje (še posebno pri Carry Lookahead)

Zgradba ALE pri HIP



Koraki pri izvrševanju ukazov

- Izvrševanje ukazov razdelimo na 5 korakov:
  1. Prevzem ukaza (IF – Instruction Fetch)
  2. Dekodiranje ukaza (ID – Instruction Decode)
  3. Izvrševanje operacije (EX - Execute)
  4. Dostop do pomnilnika (MEM - Memory)
  5. Shranjevanje rezultata (WB – Write Back)
- En korak traja eno ali več urinih period
- Vsi ukazi ne potrebujejo vseh 5 korakov

1. korak: Prevzem

1. Prevzem ukaza

$$IR \leftarrow_{32} M[PC]$$

- Vsebina *PC* se preko MUXa prenese na pomnilniški naslov
- Iz pomnilnika se prebere 32 bitov in prenese v *IR*
  - Vsebina *PC* je vedno mnogokratnik 4
- Prenos traja v primeru zadetka v predpomnilniku 1 periodo, v primeru zgrešitve pa 11 period
  - dokler je aktiven MemWait, kontrolna enota čaka

2. korak: Dekodiranje

2. Dekodiranje ukaza

A ← *Rs1*;
B ← *Rs2* (ali *Rd*);
PC ← *PC* + 4

- to troje se izvede hkrati
- 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

3. korak: Izvrševanje

3. Izvrševanje operacije

- ALE operacija ali računanje dejanskega naslova

3.1 **Ukazi za prenos podatkov** (load/store)

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

- razširjeni $IR_{0..15}$ je odmik (z IRtoS2 gre na *S2*)
- *A* (*Rs1*) gre na *S1* (AtoS1)
- ALE operacija ADDU
- rezultat *D* gre v *MAR*
- hkrati gre *B* v *MDR* (pri branju ni potrebno, vendar ne škodi, obenem pa poenostavlja načrtovanje kontrolne enote)
- porabi se 1 perioda

3.2 **Klic procedure** (ukaz CALL)

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

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

3.3 **ALE ukazi**

$C \leftarrow A$ op B ali $C \leftarrow A$ op $\text{raz}(IR_{0..15})$

(glede na format (2 ali 1))

- *A* je *Rs1*
- *Rs2* je *B* (format 2) ali pa takojšnji operand $IR_{0..15}$ (format 1)
- Kontrolna enota aktivira BtoS2 oz. IRtoS2
- Traja 1 periodo

3.4 **Ukaz TRAP**

$EPC \leftarrow PC; \qquad I \leftarrow 0$

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

- naslov servisnega programa je na naslovu FFFFFF00 + 4 × *n*
- 2 periodi:
  - 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

3.5 **Brezpogojni skok** (ukaz J)

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

- zapis v *PC* z PCWrite
- 1 perioda

3.6 **Pogojni skoki** (ukaza BEQ in BNE)

BEQ: če je $B = 0$, potem $PC \leftarrow PC + \text{raz}(IR_{0..15})$

BNE: če je $B \neq 0$, potem $PC \leftarrow PC + \text{raz}(IR_{0..15})$

- Kot pogoj se uporablja vsebina *Rd*, ki se je v koraku 2 prenesel v *B*
- Na *B* je priključena logika za ugotavljanje ničle
- 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*:
  - $\text{PCWriteCond0} \cdot (B=0) \vee \text{PCWriteCond1} \cdot (B\neq 0) \vee \text{PCWrite}$
- 1 perioda

3.7 **Ukaz RFE**

za vrnitev iz prekinitve ali pasti

$PC \leftarrow EPC$

- 1 perioda

3.8 **Ukaza MOVER in MOVRE**

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

MOVER: $C \leftarrow EPC$

MOVRE: $EPC \leftarrow A$

3.9 **Ukaza EI in DI**

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

EI: $I \leftarrow 1$

DI: $I \leftarrow 0$

4. korak: Dostop

4. Dostop do pomnilnika

ukazi load in TRAP: $MDR \leftarrow M[MAR]$

ukazi store: $M[MAR] \leftarrow MDR$

- naslov se je v prejšnjem koraku prenesel v *MAR*
- 1 perioda + morebitne čakalne periode

5. korak: Shranjevanje

5. Shranjevanje rezultata

5.1 **ALE ukazi, CALL in MOVER**

$Rd \leftarrow C$

5.2 **Ukaz TRAP**

$PC \leftarrow MDR$

- naslov servisnega programa, ki se je v koraku 4 prebral iz pomnilnika v *MDR*, gre preko ALE ($S2 \rightarrow D$) v *PC*

5.3 **Ukazi load**

$C \leftarrow MDR$ (preko ALE)

$Rd \leftarrow C$

- 2 periodi

Čas izvrševanja ukazov

- Čas izvrševanja različnih ukazov je različen
  - nekateri ukazi ne potrebujejo vseh korakov
  - trajanje nekaterih korakov se od ukaza do ukaza lahko razlikuje
- Čas izvrševanja je odvisen tudi od časa za dostop do pomnilnika, ta pa od pogostosti zgrešitev v predpomnilniku
- Vsi ukazi potrebujejo en dostop do pomnilnika (za prevzem ukaza), ukazi za prenos podatkov (load in store) in TRAP pa še enega

- **Število urinih period na ukaz** (pri HIP)
  - najmanjše
  - povprečno (upošteva tudi zgrešitve v predpomnilniku)

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

Povprečno število urinih period na ukaz

- Povprečno število urinih period pa upošteva še povprečno število čakalnih urinih period, ki so zaradi zgrešitev v predpomnilniku potrebne pri dostopih do pomnilnika
  - Povprečno št. period = najmanjše št. period + povprečno št. čakalnih period (št. čakalnih period × verjetnost zgrešitve × št. pom. dostopov)
    - pri vsaki zgrešitvi je 10 čakalnih urinih period
  - če je verjetnost zadetka v predpomnilniku 95%, je povprečno število čakalnih urinih period enako 10 × 0,05 × število pomnilniških dostopov

Povprečno število urinih period na ukaz za vse ukaze skupaj

- **CPI** (Clocks Per Instruction)

$$CPI = \sum_{i=1}^{n} CPI_i \cdot p_i$$

  - CPI_i je število urinih period za ukaz vrste i
  - p_i je relativna pogostost (verjetnost) posamezne vrste ukaza
  - če za CPI_i vzamemo najmanjše možno število urinih period, dobimo $CPI_{idealni}$, ki ne vključuje izgubljenih urinih period zaradi zgrešitev v predpomnilniku

- Pogostost posameznih skupin ukazov je precej odvisna tudi od programa, ki ga poganjamo
  - Npr. prevajalnik za C uporablja 46% ALE ukazov, 36% ukazov za prenos podatkov in 18% kontrolnih ukazov
  - Če predpostavimo, da je število load ukazov trikrat večje od števila store ukazov in da je 5% od vseh skokov klicev procedur, dobimo:

$CPI_{idealni}$ = 5,5*0,36 + 4*0,46 +
(3*0,95 + 5*0,05)*0,18 = 4,38

- Če upoštevamo še 95% verjetnost zadetka, dobimo

$CPI_{resnični}$ = 6,5*0,36 + 4,5*0,46 + (3,5*0,95 + 5,5*0,05)*0,18
= 5,06
= $CPI_{idealni}$ + 0,68

- Obstaja tudi parameter **MIPS** (Million Instructions Per Second):

$$MIPS = \frac{f_{CPE}}{CPI \cdot 10^6} = \frac{1}{CPI \cdot t_{CPE} \cdot 10^6}$$

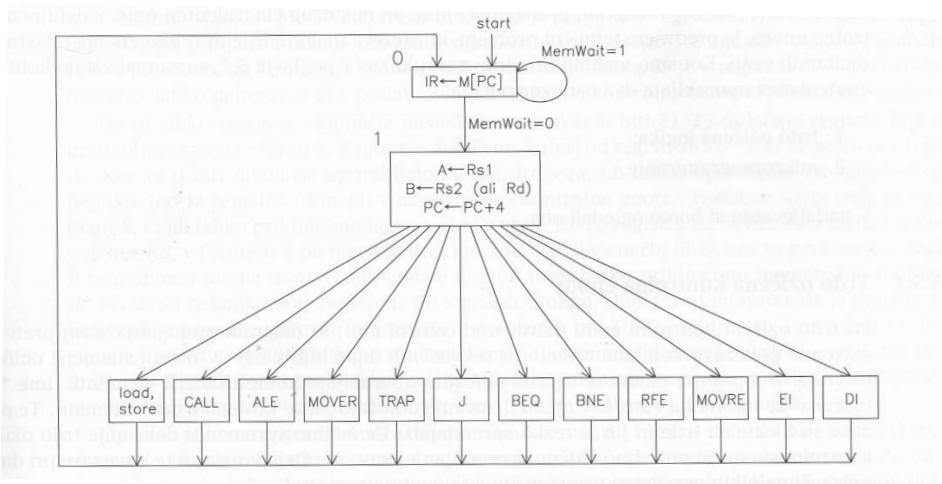
- Na MIPS vpliva frekvenca ure
  - pri 2GHz bi dobili za prevajalnik za C 395,2 MIPS

Kontrolna enota

- Podatkovna enota je pasivna
  - naredi samo tisto, kar od nje zahtevajo kontrolni signali
- Kontrolna enota (KE) mora vedeti za vsak ukaz, kateri koraki so potrebni in katere signale je treba aktivirati v določeni periodi
  - KE je zapletena
    - večina napak pri razvoju novega računalnika je v KE
- Delovanje KE lahko podamo z **diagramom prehajanja stanj** (DPS)
  - med stanji se seveda prehaja ob aktivni fronti ure
- Temu ustreza **končni avtomat** (finite state machine)

- V vsakem stanju sta definirani dve funkciji:
  1. **Funkcija naslednjega stanja.**
     - določa, pri katerih pogojih (vhodni signali) se izvrši prehod v vsako od možnih stanj
  2. **Funkcija izhodnih signalov**
     - določa, kateri izhodni signali so v danem stanju aktivni

Poenostavljen diagram prehajanja stanj

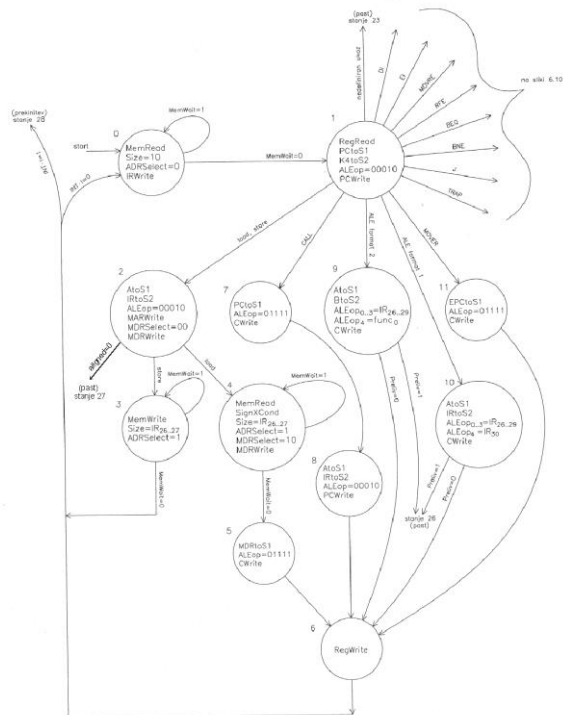


- CPE lahko izvršuje ukaze na 2 načina (to vpliva na realizacijo kontrolne enote):
  1. **Trdo ožičena logika**
     - vezje (logična vrata, pomnilne celice, povezave)
     - spremembe so možne le s fizičnim posegom
  2. **Mikroprogramiranje**
     - pri vsakem ukazu se aktivira ustrezno zaporedje **mikroukazov** (**mikroprogram**)
     - mikroprogrami so shranjeni v kontrolnem pomnilniku CPE
     - mikroukazi so primitivnejši od običajnih in jih izvršuje trdo ožičena logika
     - počasnejše, vendar lahko spreminjamo ali dodajamo ukaze, ne da bi spreminjali vezje
- Uporabnika način izvajanja ukazov v resnici ne zanima

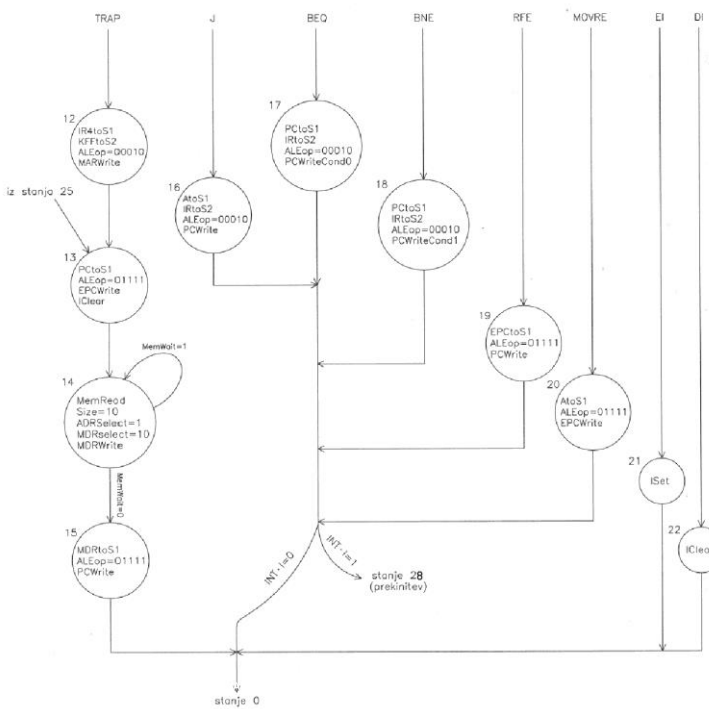
Trdo ožičena kontrolna enota

- Avtomat realiziramo s kombinacijskimi in sekvenčnimi vezji
  - npr. vrata in flip-flopi
- Trdo ožičena pomeni, da so povezave fiksne
  - če hočemo kaj spremeniti, jih je potrebno fizično spremeniti
- Najprej potrebujemo popoln DPS

DPS,
1.del

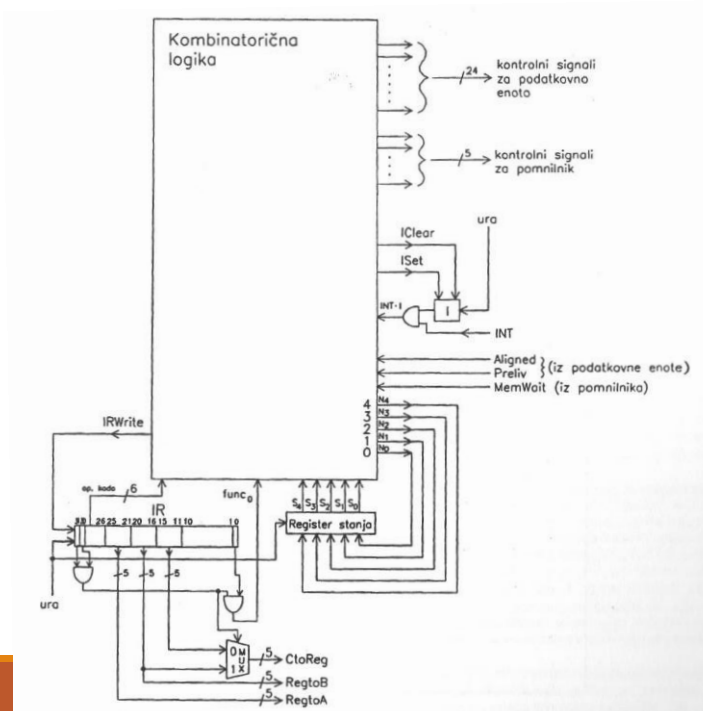


DPS,
2.del



- DPS ima 28 stanj
  - 5-bitni register stanja
  - kombinacijsko logiko, ki iz stanja, registra IR in vhodnih signalov MemWait, Preliv in INT (pogojen z *I*) tvori naslednje stanje in izhodne signale
- Zaradi preproste zgradbe ukazov lahko precejšen del bitov IR peljemo mimo KE v PE
  - npr. biti 21-25 določajo register *Rs1* in se uporabljajo samo v koraku 1
    - določajo, kateri od registrov *R0-R31* se prebere v *A*
    - lahko jih uporabimo za signale RegtoA neposredno iz *IR*
  - bite 16-20 registra *IR* lahko peljemo direktno na RegtoB
  - pri CtoReg potrebujemo še dodaten MUX, kajti
    - register *Rd* (kamor se piše iz *C*) določajo v formatu 1 biti 16-20, v formatu 2 pa biti 11-15
    - zato je treba izbirati med dvema 5-bitnima signaloma $IR_{11..15}$ (če sta $IR_{30..31}$ oba 1) in $IR_{16..20}$ (sicer)
- Kontrolna enota ima 39 izhodnih signalov

Realizacija KE



- Primer: izhodni signal IRWrite je aktiven samo v stanju 0, zato ga definira funkcija

  $$IRWrite = S_4'S_3'S_2'S_1'S_0' ,$$

  kjer so $S_{0..4}$ biti stanja

- Primer: izhodni signal PCtoS1 je aktiven v stanjih 1, 7, 13 in 24, zato ga definira funkcija

  $$PCtoS1 = S_4'S_3'S_2'S_1'S_0 + S_4'S_3'S_2S_1S_0 + S_4'S_3S_2S_1'S_0 + S_4S_3S_2'S_1'S_0'$$

- Za realizacijo kombinacijske logike se pogosto uporabljajo kar bralni pomnilniki (ROM) ali PAL

Mikroprogramska kontrolna enota

- Pri nekaterih računalnikih je število ukazov, formatov in načinov naslavljanja lahko zelo veliko
  - za Intelove procesorje 80x86 bi potrebovali več tisoč stanj
- **Mikroprogramska kontrolna enota** je narejena kot majhen računalnik
  - diagram prehajanja stanj se pretvori v **mikroprogram**
    - vsako stanje je en **mikroukaz**
  - mikroprogram je shranjen v bralnem pomnilniku (ROM)
    - **firmware**
    - pri HIP zadostuje ROM s 512 lokacijami (9-bitni naslov)
  - za vsak ukaz obstaja majhen mikroprogram (iz mikroukazov, ki so bolj primitivni)
  - pri potencialnem spreminjanju ukazov je mnogo lažje spremeniti mikroprogram, kot pa vezje

- Format mikroukazov pri HIP:

48 0

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

- Pri mikroprog. KE vsak ukaz traja 1 urino periodo
- Mikroukazi aktivirajo kontrolne signale
  - pri HIP večina bitov mikroukaza predstavlja signale (vsak enega)
- Mikroprog. števec μPC je 9-biten, mikroprog. pomnilnik velikosti 512 x 49
  - ukazov nekaj čez 50, povprečen ukaz rabi ≈ 2 mikroukaza

- Polja v mikroukazu (vsak bit predstavlja en signal):
  - 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

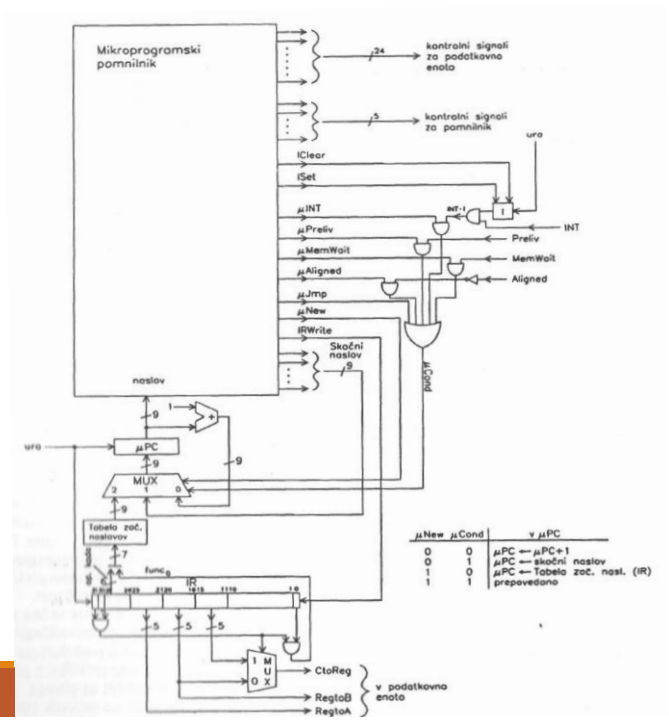
- R:
  - določa, ali se bere ali piše v registre R0-R31
  - Bita: RegWrite, RegRead
- Mux:
  - določa krmiljenje obeh mux
  - Biti: ADRSelect, MDRSelect
- Pomnilnik:
  - določa, kaj se dogaja s pomnilnikom
  - Biti: MemRead, MemWrite, SignXCond, Size
- Next:
  - določa način izbire naslova naslednjega 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

- Določanje naslova naslednjega mikroukaza:
  1. Če μNew = 1, se v μPC iz tabele začetnih naslovov (dispatch table) prenese naslov prvega mikroukaza novega mikroprograma
     - tabela preslika $IR_{26..31}$ in IR_0 v 9-bitni mikroprogramski naslov
     - zaradi 7 bitov je lokacij v tabeli precej več kot ukazov
       - nedefinirani ukazi se v tabeli preslikajo v mikroprogram, ki sproži past
  2. Če μJmp = 1, brezpogojni skok na Skočni naslov
     - sicer se izračuna logična funkcija μINT · INT + μPreliv · Preliv + μAligned · Aligned' + μMemWait · MemWait
       - če 1, skok na Skočni naslov
  3. Sicer μPC ← μPC + 1

Primeri mikroprogramov za ukaze LW, SH, SUB, ADDI in ADDU

št.	Oznaka	ALEop	S1	S2	Wreg	R	Mux	Pomnilnik	Next	Skočni naslov
1	**LW:**	ADDU	AtoS1	IRtoS2	MARWrite					
2	LW1:				MDRWrite		ADRSelect=1 MDRSelect=10	MemRead Size=10	µMemWait	LW1
3		S1 → D	MDRtoS1		Cwrite					
4	regw:					RegWrite				
5	fetch:				IRWrite		ADRSelect=0	MemRead Size=10	µMemWait	fetch
6		ADDU	PCtoS1	K4toS2	PCWrite	RegRead			µNew	
1	**SH:**	ADDU	AtoS1	IRtoS2	MARWrite MDRWrite		MDRSelect=00			
2	SH1:						ADRSelect=1	MemWrite Size=01	µMemWait	SH1
3	SH2:				IRWrite		ADRSelect=0	MemRead Size=10	µMemWait	SH2
4		ADDU	PCtoS1	K4toS2	PCWrite	RegRead			µNew	
1	**SUB:**	SUB	AtoS1	BtoS2	CWrite				µJmp	regw
1	**ADDI:**	ADD	AtoS1	IRtoS2	CWrite				µJmp	regw
1	**ADDU:**	ADDU	AtoS1	BtoS2	CWrite				µJmp	regw

Mikroprog-ramska KE



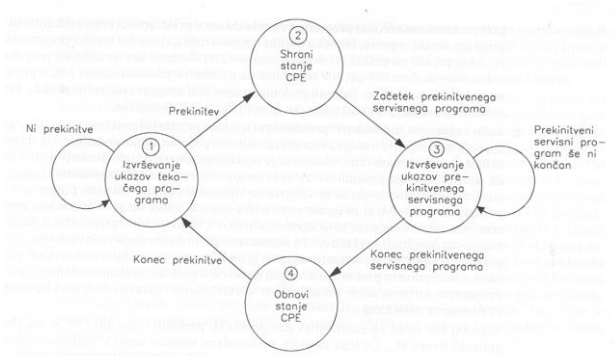
- Ker je mikroprog. pomnilnik najdražji del mikroprog. KE, so želeli zmanjšati
  - širino mikroukazov
    - kodiranje (namesto 1-od-N)
    - poleg tega:
      - določena polja niso vedno aktivna,
      - določena niso aktivna naenkrat, ...
  - število mikroukazov
- Delitev glede na širino ukazov oz. stopnjo kodiranja:
  1. **Horizontalno mikroprogramiranje**
     - malo (ali nič) kodiranja
     - dolgi mikroukazi
     - hitrejše, dražje
  2. **Vertikalno mikroprogramiranje**
     - veliko kodiranja
     - kratki mikroukazi (a bolj številni)
     - počasnejše, cenejše

Prekinitve in pasti

- **Prekinitev** (interrupt) je dogodek, ki povzroči, da CPE začasno preneha izvajati tekoči program ter prične izvajati t.i. **prekinitveni servisni program** (PSP)
  - Zahteva za prekinitev pride v CPE od zunaj, npr. od neke vhodno/izhodne naprave
- **Past** (trap) je posebna vrsta prekinitve, ki jo zahteva sama CPE ob nekem nenavadnem dogodku ali celo na zahtevo programerja
  - pasti pridejo od znotraj
- Če ne bi bilo prekinitev in pasti, bi morala CPE stalno preverjati stanje mnogih naprav

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

- Pri prekinitvah razlikujemo 4 stanja:
  - Normalno izvrševanje ukazov programa
  - Shranjevanje stanja CPE ob pojavu zahteve za prekinitev
  - Skok na prekinitveni servisni program in njegovo izvajanje
  - Vrnitev iz prekinitvenega servisnega programa in obnovitev stanja CPE



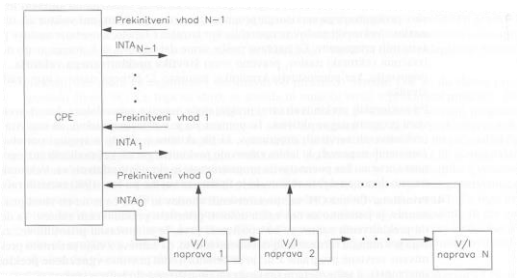
- 5 dejavnikov:

1. **Kdaj CPE reagira na prekinitveno zahtevo**
   - najenostavneje je po izvrševanju tekočega ukaza
     - v tem primeru se mora ohraniti samo stanje programsko dostopnih registrov (*R0-R31, PC, EPC, I*)
   - programer lahko onemogoči odziv CPE na prekinitvene zahteve (bit *I*, ukaza DI in EI)
     - po vklopu so V/I prekinitve onemogočene, dokler se V/I naprave ne inicializirajo
     - če pride do nove prekinitve, preden prekinitveni servisni program shrani registre, lahko pride do izgube *PC*, ki se ob prekinitvi shrani v *EPC*
2. **Kako zagotoviti "nevidnost" prekinitev**
   - treba je zagotoviti, da je stanje (registrov) CPE enako kot prej

2. **Kje se dobi naslov prekinitvenega servisnega programa**
   - to je pomembno pri prekinitvah, ki prihajajo od zunaj
   - najprej je treba ugotoviti, katera naprava je zahtevala prekinitev
     - če je na vsakem prekinitvenem vhodu samo ena naprava, je problem trivialen
     - drugače je, če je na enem prek. vhodu več naprav, ali če ima ista naprava več PSP
   - najpreprostejše je **programsko izpraševanje** (polling)
     - CPE bere registre vsake V/I naprave, v katerih je bit, ki pove, ali je naprava zahtevala prekinitev
     - če je, izvrši skok na njen prek. servisni program
     - polling je zamuden
   - običajen način pa so **vektorske prekinitve**
     - naprava pošlje v CPE naslov njenega PSP v **prekinitvenem prevzemnem ciklu**, s katerim CPE obvesti V/I naprave, naj pošljejo informacijo o izvoru prekinitve
     - ta naslov se imenuje **prekinitveni vektor** ali **vektorski naslov**, ki se običajno izračuna iz **številke prekinitvenega vektorja** po nekem pravilu (tu zadošča npr. že 8-bitno število)
     - možno je tudi, da ima ena naprava več PSP

4. Prioriteta

- če ima CPE več prek. vhodov in več naprav na posameznem vhodu, potrebujemo neko prioriteto.
- **vgnezdene prekinitve** (nested interrupts), 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, bolj fleksibilno obravnavo prioritete
- določanje prioritete je možno izvesti tudi z **marjetično verigo** (daisy chain): naprava, ki ni zahtevala prekinitve, spusti določen signal v naslednjo napravo, tista pa, ki jo je, zapre signalu pot in vrne CPE ustrezno informacijo, da jo CPE lahko prepozna



5. Potrjevanje prekinitve

- potrebno zato, da naprava spusti prekinitveni vhod (da se prekinitev ne servisira večkrat)
- dva načina:
  - programsko: prekinitveni servisni program piše v nek register krmilnika naprave
  - strojno: z nekim signalom (ali kombinacijo večih) se obvesti napravo

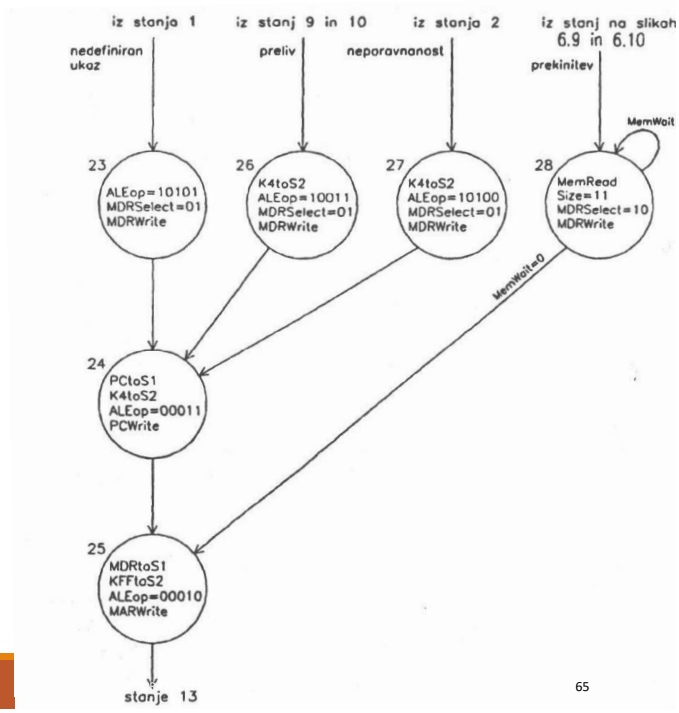
Prekinitve in pasti pri HIP

- HIP ima en sam prekinitveni vhod INT in uporablja vektorske prekinitve
- CPE se odzove na prekinitveno zahtevo s prekinitveno prevzemnim ciklom
  - podobno branju iz pomnilnika, le da signal Size=11 povzroči, da se pomnilnik ne odzove
  - V/I naprave pa se odzovejo tako, da tista z najvišjo prioriteto da na podatkovne signale D0-D7 s 4 pomnoženo številko vektorja *n*
    - to je 6-bitno število
    - *n* gre na D2-D7, ker je množenje s 4 enako pomiku za 2 bita, D0 in D1 pa gresta na 0
    - s tem so pomnilniški naslovi FFFFFF00 + 4 x n (vektorski naslovi), na katerih so shranjeni začetni naslovi PSP, vedno večkratniki 4 (poravnanost)

- Poleg prekinitev ima HIP 3 pasti:
  1. Nedefiniran ukaz (*n*=0)
  2. Preliv (pri ADD, ADDI, SUB, SUBI) (*n*=1)
  3. Neporavnan operand (pri 16- ali 32-bitnih operandih pri load oz. store) (*n*=2)
- Poleg tega ima ukaz *TRAP*, ki je programska past
  - program skoči na naslov, ki je shranjen na enem od vektorskih naslovov
- 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 $FF00_H + 4 \times n$

- Vključitev prekinitev in pasti v DPS



Merjenje zmogljivosti CPE

- Zmogljivost CPE ni isto kot zmogljivost računalnika!
  - vplivata tudi zmogljivost pomnilniškega in V/I sistema
  - zmog. CPE in zmog. računalnika lahko enačimo le, če sta pomnilniški in V/I sistem dovolj zmogljiva (da CPE ne čaka), kar pa je problemsko odvisno
- Za zmogljivost CPE je merodajen čas izvrševanja programa
- Če zanemarimo V/I, je čas izvrševanja programa enak času, ki ga potrebuje CPE (CPE čas)

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

CPI ... povprečno št. urinih period na ukaz (Clocks Per Instruction)

- Te tri lastnosti so medsebojno odvisne (pa tudi sredstva za njihovo izboljšanje):
  - t_{CPE} (f_{CPE}): odvisna od hitrosti in števila digitalnih vezij, pa tudi od zgradbe CPE
  - CPI: zgradba CPE in ukazi
  - Število ukazov, v katere se prevede program: ukazi in lastnosti prevajalnika
- Posamezna od teh lastnosti ni merilo!
- Čas je seveda odvisen tudi od programa, vhodnih podatkov in velikosti problema

- Marsikdo primerja različne CPE kar na osnovi frekvence ure (f_{CPE})
  - slabo, ker je lahko zelo zavajajoče
  - neka CPE nižje frekvence ima lahko krajše CPE čase kot neka druga CPE višje frekvence
- Pogosto se uporablja **MIPS** (Million Instructions Per Second):

$$MIPS = \frac{1}{CPI \cdot t_{CPE} \cdot 10^6} = \frac{f_{CPE}}{CPI \cdot 10^6}$$

  - Z njim se CPE čas izrazi takole:

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

  - Tudi MIPS nezanesljiv
    - odvisen od števila in vrste ukazov
      - pri enostavnejših ukazih je MIPS večji (čeprav jih potrebujemo več)
    - odvisen od programa
    - Meaningless Indication of Processor Speed

- **MFLOPS** (Million FLoating point Operations Per Second)
  - operacije v plavajoči vejici so si (na različnih računalnikih) bolj podobne kot ukazi
  - ima smisel samo za programe, ki uporabljajo operacije v plavajoči vejici
  - proizvajalci so začeli navajati maksimalno (teoretično) zmogljivost
    - dosti večja kot na realnih programih
- **Sintetični "benchmarki"**
  - 1976: Whetstone, Linpack
  - 1984: Dhrystone (brez FP)
  - Quicksort, Sieve, Puzzle, ...
  - proizvajalci tudi tu niso stali križem rok ... ☺
    - npr. "optimizacija prevajalnikov"
- **SPEC** (Standard Performance Evaluation Corporation)
  - več programov, pogosto spreminjanje