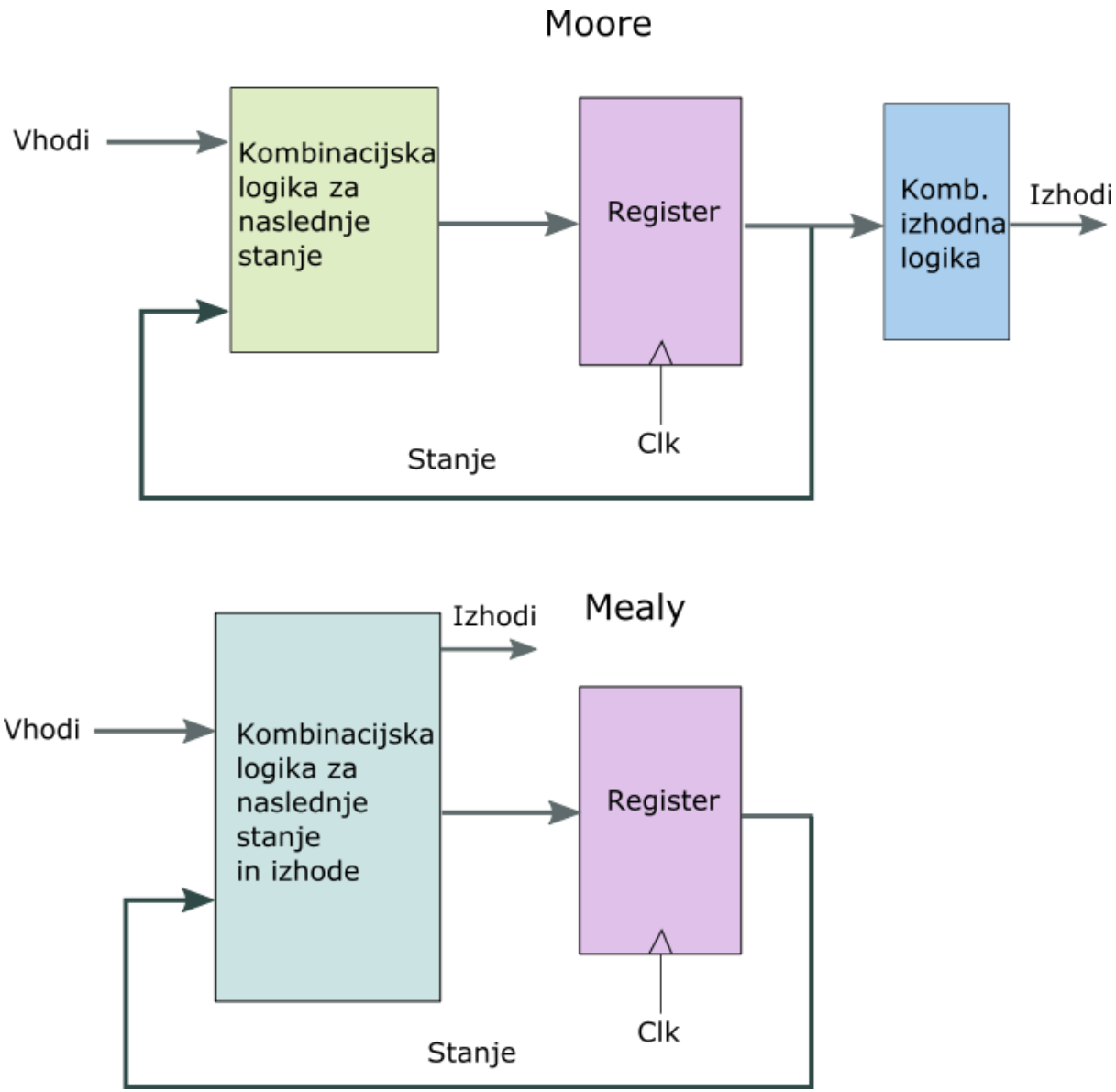
6 CENTRALNA PROCESNA ENOTA

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



- Vzemimo primer dvobitnega števca, ki periodično šteje od 0 do 3.
  - Če vzamemo 2 D-flip-flopa, bo nižji imel na vhodu $D_0 = Q_0'$, višji pa $D_1 = Q_0$ xor Q_1.
  - Imamo torej dvobitni register in kombinacijsko logiko iz enega invertorja in vrat xor.
  - Zakasnitev obojih vrat določa najmanjšo možno periodo
    - Seveda je treba upoštevati še vzpostavitveni in držalni čas
  - Tako je maksimalna frekvenca navzgor omejena

Delovanje CPE

1. **Dobava ukaza iz pomnilnika** (fetch)
2. **Izvrševanje ukaza**
   a) dekodiranje ukaza (in branje registrov)
   b) izvedba ALE operacije (ali izračun naslova)
   c) prenos operandov v CPE iz pomnilnika (po potrebi)
   d) shranjevanje rezultata (po potrebi) – pisanje v register
   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 (načelno) 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

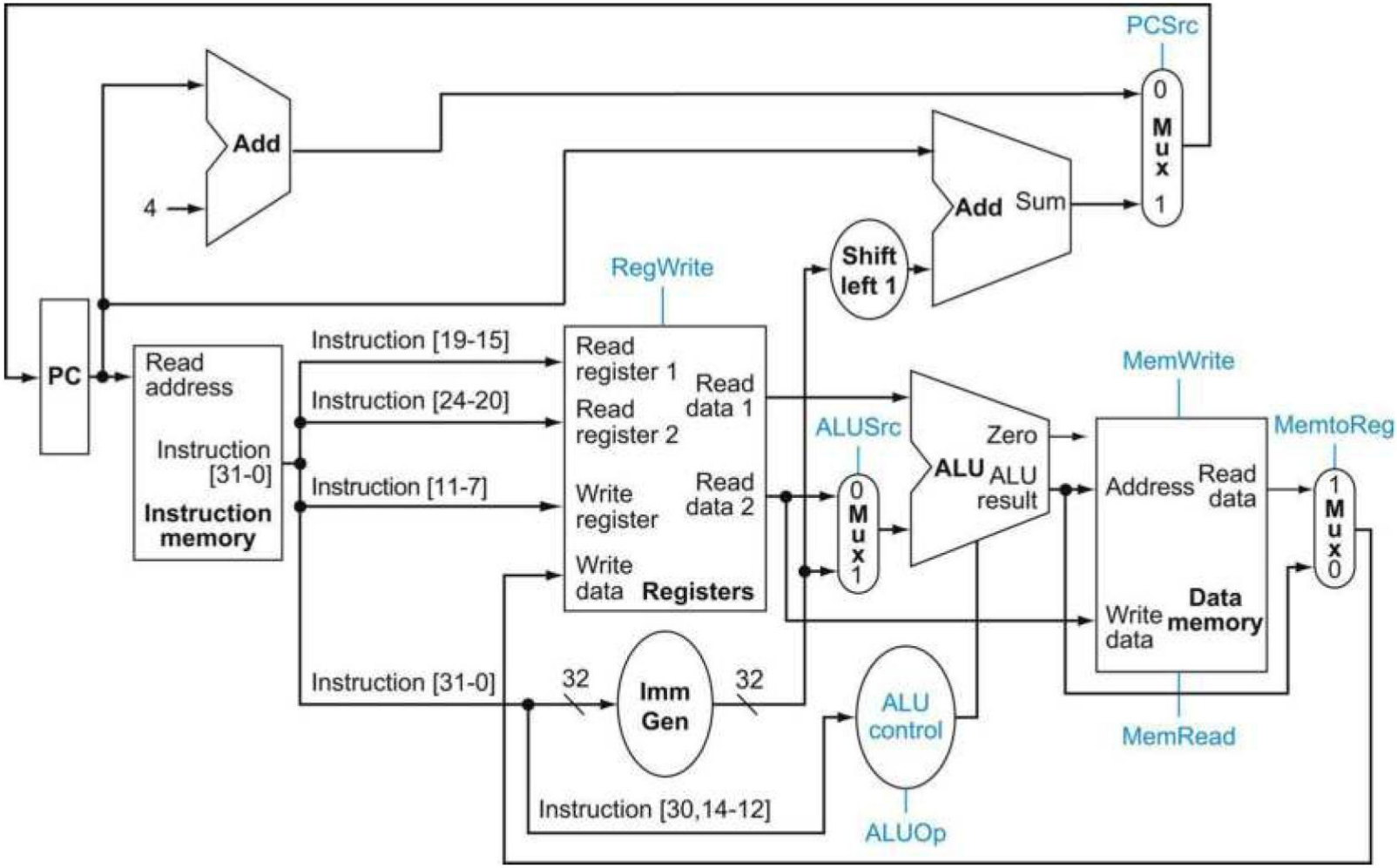
Načini implementacije CPE

- Različni načini (sinhronske) implementacije:
  1. Celoten ukaz v 1 periodi ure ('single-cycle') – ***enocikelni procesor***
     - dolga perioda!
  2. Ukaz v več periodah ure ('multi-cycle') – ***večcikelni procesor***, ukazi pa se ne prekrivajo
     - Kontrolna enota vsebuje *končni avtomat* (FSM – Finite State Machine), ki nadzoruje dogajanje
     - Avtomat prehaja med stanji ob fronti ure
  3. Ukaz v več periodah ure, a se ukazi časovno delno prekrivajo (cevovod - 'pipeline') – ***cevovodni procesor***

Implementacija enocikelnega procesorja

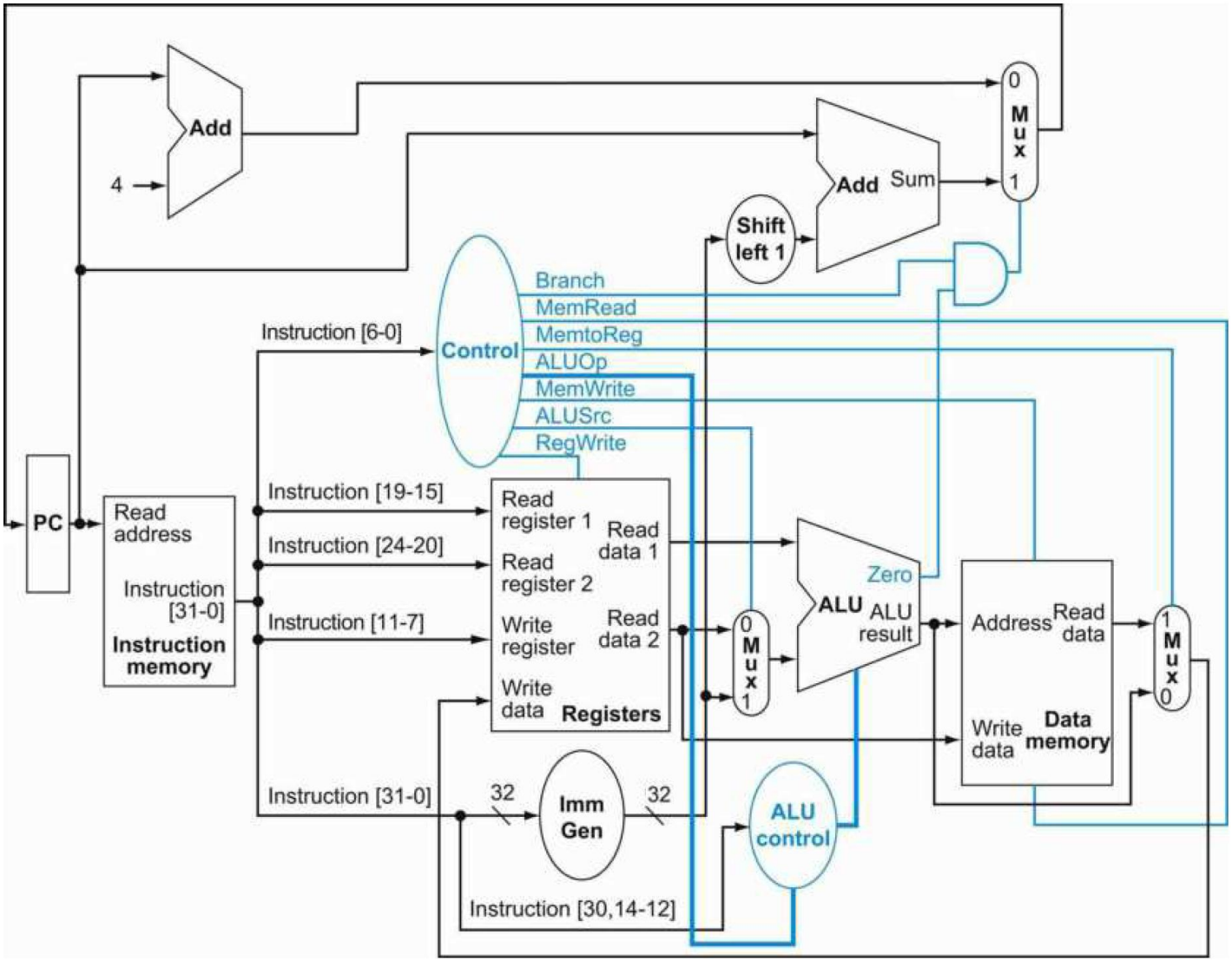
- Preprost primer:
  - implementacija ukazov:
    - lw,
    - sw,
    - beq,
    - add, sub, and, or (format R)
- ALE operacija se izvrši nad dvema 32-bitnima vhodnima operandoma *x* in *y*
  - Rezultat je 32-bitni izhod in bit *Zero*
  - Signali ALEop (4 biti) pridejo iz kontrolne enote
    - ta jih tvori iz bitov operacijske kode (in podaljškov *funct*) ukaza

Podatkovna enota enocikelnega procesorja



Patterson, Hennessy: Computer Organization and Design RISC-V Edition: The Hardware Software Interface (Morgan Kaufmann)

Enocikelni procesor (z dodano kontrolno enoto)



ALE s 4-bitno kontrolno kodo

- majhno kontrolno vezje na osnovi funct7, funct3 in 2-bitne kode ALUOp:
  - 00: add (za load in store)
  - 01: sub and test if zero (za beq)
  - 10: določena glede na funct

Opkoda	ALUOp	Operacija	funct7	funct3	ALE operacija	ALU control
lw	00	load word	xxxxxxx	xxx	+	0010
sw	00	store word	xxxxxxx	xxx	+	0010
beq	01	branch if equal	xxxxxxx	xxx	-	0110
(format R)	10	add	0000000	000	+	0010
(format R)	10	sub	0100000	000	-	0110
(format R)	10	and	0000000	111	and	0000
(format R)	10	or	0000000	110	or	0001

- Kakšen je namen vrat IN (AND)?
- Kontrolna enota mora za vsak ukaz določiti operacije, označeno na sliki z modro:
  - ALUsrc, Branch, MemRead, MemWrite, ...
- Kontrolna enota je v primeru enocikelnega procesorja kar preprosta tabela
  - zato jo lahko realiziramo s kombinacijskim vezjem (ali bralnim pomnilnikom)

Ukaz	ALUsrc	MemToReg	Reg-Write	Mem-Read	Mem-Write	Branch	ALUOp
format R: add, sub, and, or	0	0	1	0	0	0	10
lw	1	1	1	1	0	0	00
sw	1	X	0	0	1	0	00
beq	0	X	0	0	0	1	01

Primer programa:

Naslov	Ukaz	Format	Polja (f3 je funct3)	Strojna koda
0x1000 L7:	lw x6, -4(x9)	I	$imm_{11:0}$ rs1 f3 rd op 111111111100 01001 010 00110 0000011	FFC4A303
0x1004	sw x6, 8(x9)	S	$imm_{11:5}$ rs2 rs1 f3 $imm_{4:0}$ op 0000000 00110 01001 010 01000 0100011	0064A423
0x1008	or x4, x5, x6	R	funct7 rs2 rs1 f3 rd op 0000000 00110 00101 110 00100 0110011	0062E233
0x100C	beq x4, x4, L7	B	$imm_{12,10:5}$ rs2 rs1 f3 $imm_{4:1,11}$ op 1111111 00100 00100 000 10101 1100011	FE420AE3

Zmogljivost enocikelnega procesorja

- Najdalj traja ukaz LW:
  - PC se naloži po uri (t_{pcq} = propagation delay clock-to-q),
  - Branje ukaza iz pomnilnika (t_{mem}),
  - RF bere ReadData1 (t_{RFread}),
    - obenem se razširja predznak odmika (se: sign-extension) in gre preko mux-a na ALE (t_{se}+ t_{mux}),
  - ALE sešteje bazo in odmik (t_{ALE}),
  - Podatkovni pomnilnik bere iz tega naslova (t_{mem}),
  - Mux MemtoReg izbere ReadData (t_{mux}),
  - Result se mora pojaviti malo (vzpostavitveni čas, setup time, $t_{RFsetup}$) pred fronto ure

$$t_{CPE} = t_{pcq} + t_{mem} + \max(t_{RFread}, t_{se} + t_{mux}) + t_{ALE} + t_{mem} + t_{mux} + t_{RFsetup}$$

    - označeni z rdečo so običajno večji od ostalih - po grobi oceni je perioda malo večja kot

$$2t_{mem} + t_{RFread} + t_{ALE}$$

    - CPI (clocks per instruction) = 1

- Primer: Določi najmanjšo možno periodo ure pri zakasnitvah elementov, podanih v Tabeli.
  - Koliko časa traja program z 10^9 ukazi?

Parameter	Element	Zakasnitev [ps]
t_{pcq}	register propagation clock-to-q	30
t_{se}	razširitev predznaka (sign-extension)	20
t_{mux}	multipleksor	25
t_{ALE}	ALE	200
t_{mem}	branje iz pomnilnika	250
t_{RFread}	branje registrov (register file)	150
$t_{RFsetup}$	vzpostavitev RF	20

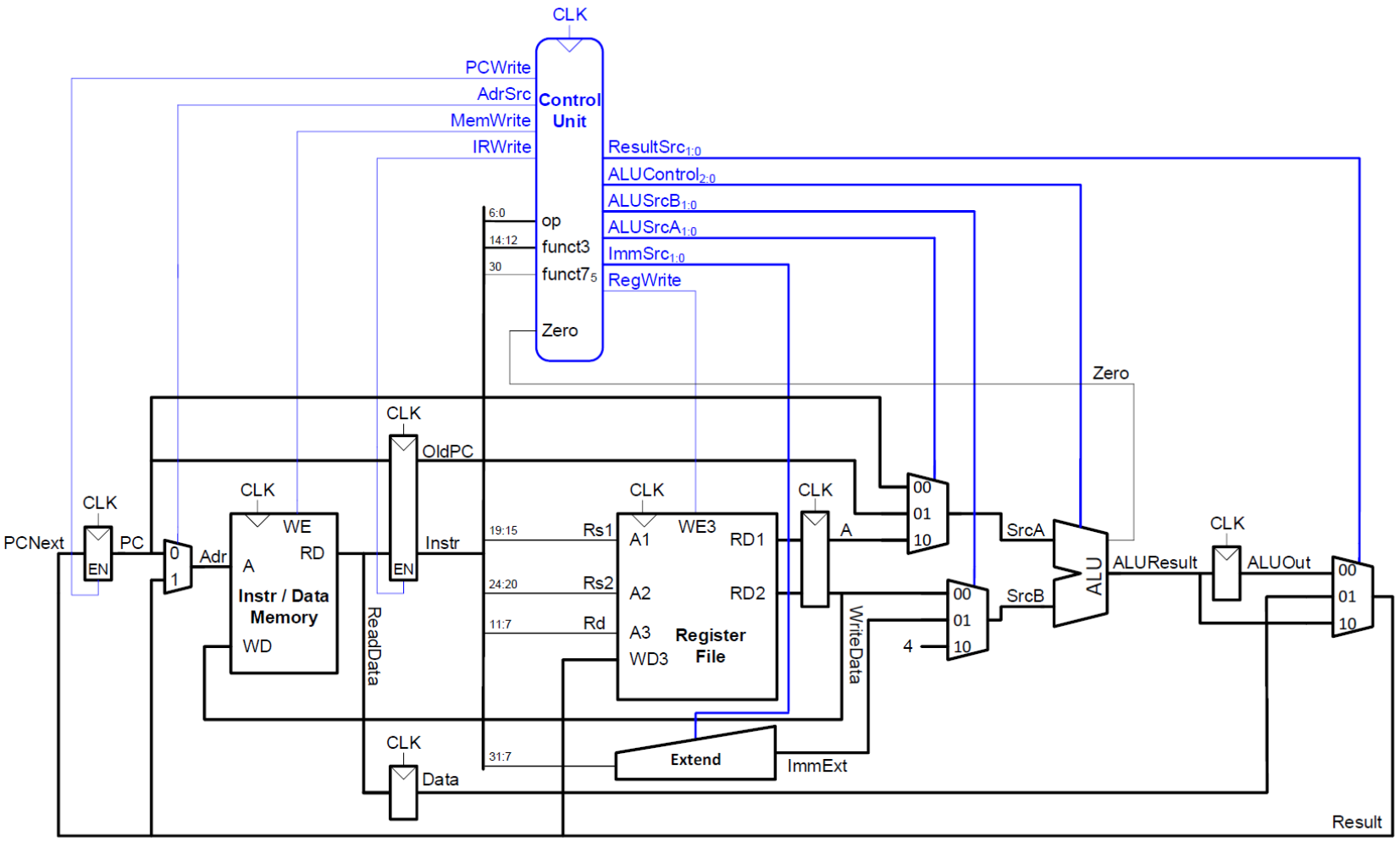
$t_{CPE,min}$ = 30+250+150+200+250+25+20 = 925 ps

Program traja 925 ms.

Implementacija večcikelnega procesorja

- Vsak ukaz potrebuje za izvedbo nekaj period ure
  - v vsaki periodi se izvrši ena podoperacija
- Preprost primer:
  - implementacija ukazov: lw, sw, beq, add, sub, and, or
- V tem primeru dodamo nekaj registrov:
  - Instr (ukazni reg.), Data, ALUOut, OldPC, A, ...

Večcikelni procesor



Harris & Harris: Digital Design and Computer Architecture (Morgan Kaufmann)

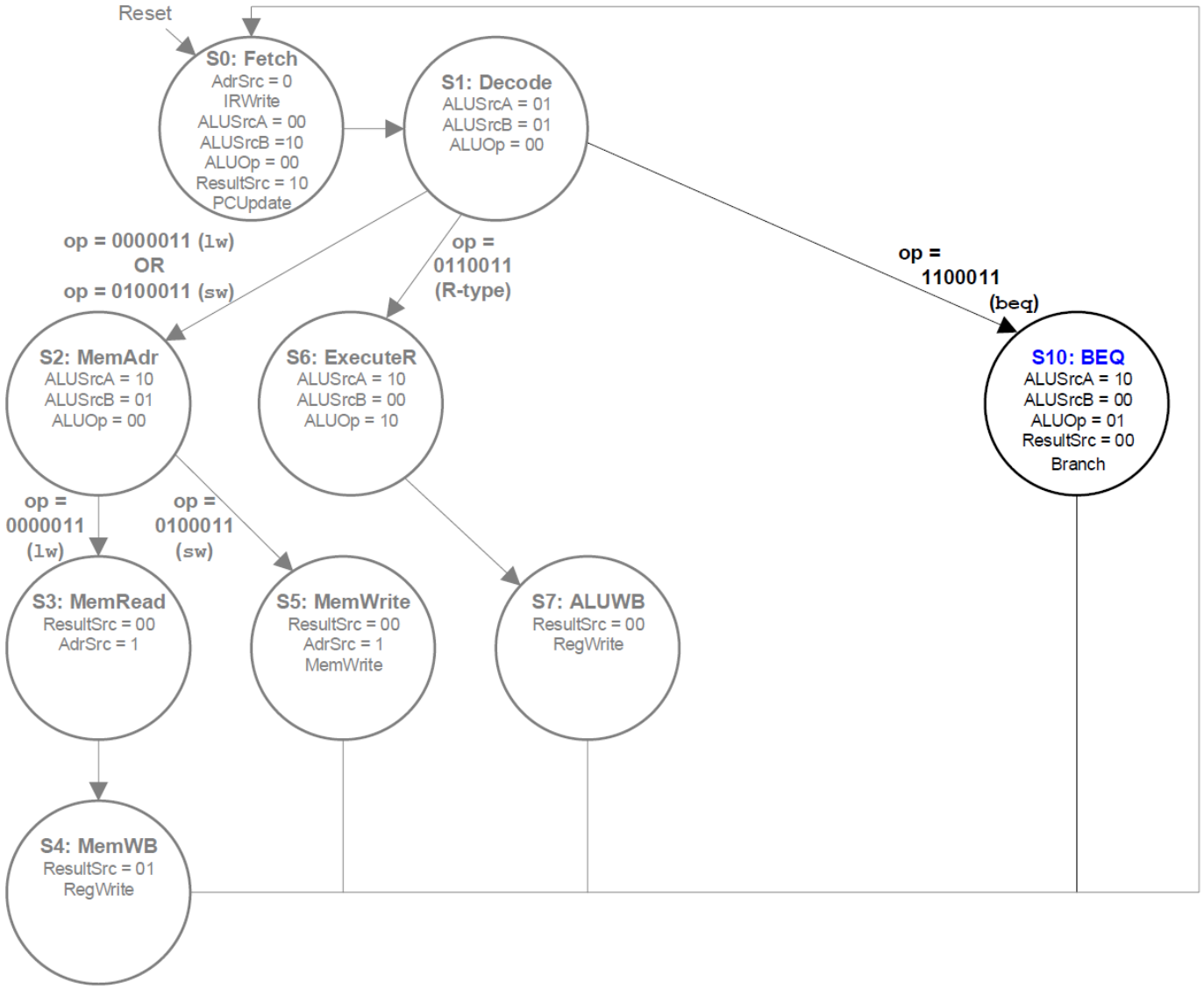
Kontrolna enota večcikelnega procesorja

- 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, FSM)

- 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

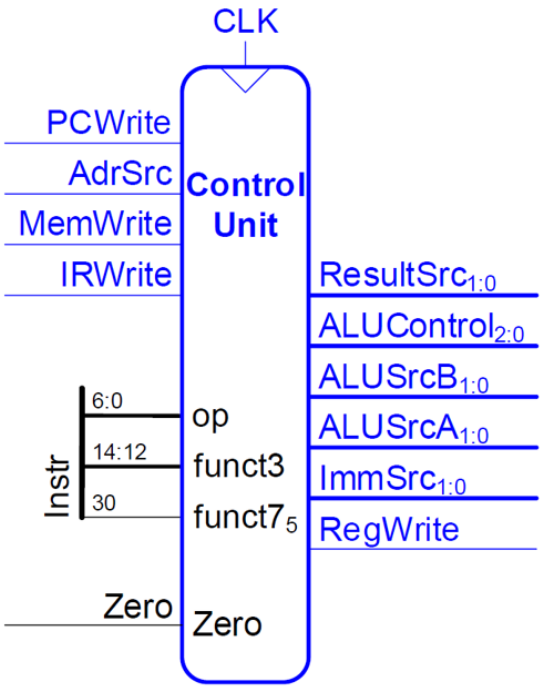
- Končni avtomat realiziramo s kombinacijskimi in sekvenčnimi vezji
  - npr. vrata in flip-flopi
- Najprej potrebujemo popoln DPS

DPS končnega avtomata, ki nadzoruje delovanje večcikelne CPE:

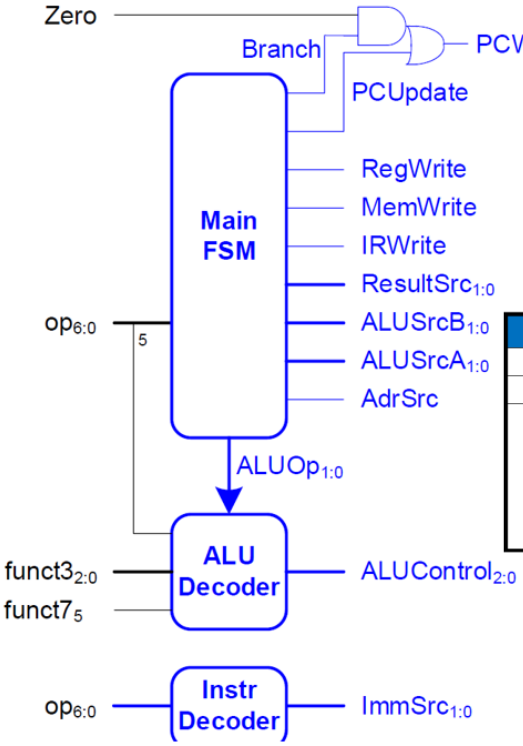


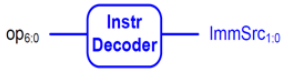
Kontrolna enota

High-Level View



Low-Level View





op	Instruction	ImmSrc
3	`lw`	00
35	`sw`	01
51	R-type	XX
99	`beq`	10

ALUOp	funct3	op5, funct75	Instruction	ALUControl2:0
00	x	x	`lw, sw`	000 (add)
01	x	x	`beq`	001 (subtract)
10	000	00, 01, 10	`add`	000 (add)
	000	11	`sub`	001 (subtract)
	010	x	`slt`	101 (set less than)
	110	x	`or`	011 (or)
	111	x	`and`	010 (and)

ALUControl2:0	Function
000	add
001	subtract
010	and
011	or
101	SLT

- DPS ima 9 stanj
  - 4-bitni register stanja
  - kombinacijsko logiko, ki iz stanja, registra IR in vhodnih signalov tvori naslednje stanje in izhodne signale
- Zaradi preproste zgradbe ukazov lahko precejšen del bitov IR peljemo mimo KE v PE
- Kontrolna enota ima precej izhodnih signalov (do 20)

Realizacija kontrolne enote

- CPE lahko izvršuje ukaze na 2 načina (to vpliva na realizacijo kontrolne enote):
  1. **Trdo ožičena logika (hard-wired logic)**
     - vezje (logična vrata, pomnilne celice, povezave)
     - trdo ožičena pomeni, da so povezave fiksne
     - 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

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**
    - za majhen primer 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

CPI

- Čas izvrševanja različnih ukazov na večcikelnem procesorju je različen
  - vsak ukaz rabi določeno celo št. urinih period, vendar nekateri ukazi ne potrebujejo vseh korakov
  - pri enocikelnem procesorju je trajanje periode ure določeno z najpočasnejšim ukazom, pri večcikelnem pa z najpočasnejšim *korakom ukaza*
    - pri enocikelnem procesorju je CPI vsakega ukaza seveda 1, pri večcikelnem pa več (tipično od 3 do 6)
- Povprečen **CPI** (Clocks Per Instruction) za nek program:

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

  - p_i je relativna pogostost (verjetnost) posamezne vrste ukaza
  - CPI_i je število urinih period za ukaz vrste *i*

- Pogostost posameznih skupin ukazov je precej odvisna tudi od programa, ki ga poganjamo
  - Npr. nek prevajalnik za C uporablja 46% ALE ukazov, 36% ukazov za prenos podatkov in 18% kontrolnih ukazov
  - Če upoštevamo CPI ukazov v spodnji tabeli in predpostavimo, da je število load ukazov trikrat večje od števila store ukazov, dobimo:

$$CPI_{idealni} = 0{,}46*4 + 0{,}36*(0{,}75*5+0{,}25*4) + 0{,}18*3 =$$
$$= 1{,}84 + 1{,}71 + 0{,}54 =$$
$$= \mathbf{4{,}09}$$

Vrsta ukaza	**CPI_i** (število urinih period na ukaz)
Ukazi load	5
Ukazi store	4
Ukazi ALE	4
Skoki	3

- Če za CPI_i vzamemo najmanjše možno število urinih period, dobimo $CPI_{idealni}$, ki ne vključuje izgubljenih urinih period zaradi zgrešitev v predpomnilniku
- Čas izvrševanja pa je v splošnem odvisen tudi od časa za dostop do pomnilnika, ta pa od pogostosti zgrešitev v predpomnilniku (PP)
  - Vsi ukazi potrebujejo en dostop do pomnilnika (za prevzem ukaza), ukazi za prenos podatkov (load in store) pa še enega
  - Povprečno število urinih period pri upoštevanju PP upošteva še povprečno število čakalnih urinih period ($CPEperiode_{čak}$), ki so zaradi zgrešitev v predpomnilniku potrebne pri dostopih do pomnilnika
    - pri vsaki zgrešitvi je določeno število čakalnih urinih period (npr. 10) – temu se reče tudi *zgrešitvena kazen* (K_Z)
  - Povprečno št. period = najmanjše št. period + povprečno št. čakalnih period:

$$CPI_{resnični} = CPI_{idealni} + CPEperiode_{čak}$$

- Povprečno št. čakalnih period = št. pom. dostopov (N) × verjetnost zgrešitve × št. čakalnih period:

$$CPEperiode_{čak} = N*(1-H)*K_Z$$

- Če je verjetnost zadetka (H) v predpomnilniku npr. 95% in $K_Z = 10$, je povprečno število čakalnih urinih period enako $N \times 0{,}05 \times 10$

Število urinih period na ukaz

- 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	5	6
store	2	4	5
ALE (format R)	1	4	4,5
BEQ	1	3	3,5

- Če upoštevamo torej 95% verjetnost zadetka (H=0,95), dobimo:

$CPI_{resnični}$ = 0,46*4,5 + 0,36*(0,75*6+0,25*5) + 0,18*3,5 =

= 2,07 + 2,07 + 0,63 =

= 4,77 = $CPI_{idealni}$ + **0,68**

- Lahko določimo 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
  - npr., pri 2 GHz bi dobili za ta C-prevajalnik 2000/4,77 = 419 MIPS

Kritična pot, ki navzdol omejuje periodo ure, je vedno **med dvema registroma**:

- S_0 (PC+4):

$$t_{CPE,min} = t_{pcq(PC)} + t_{mux} + t_{ALE} + t_{mux} + t_{setup(PC)},$$

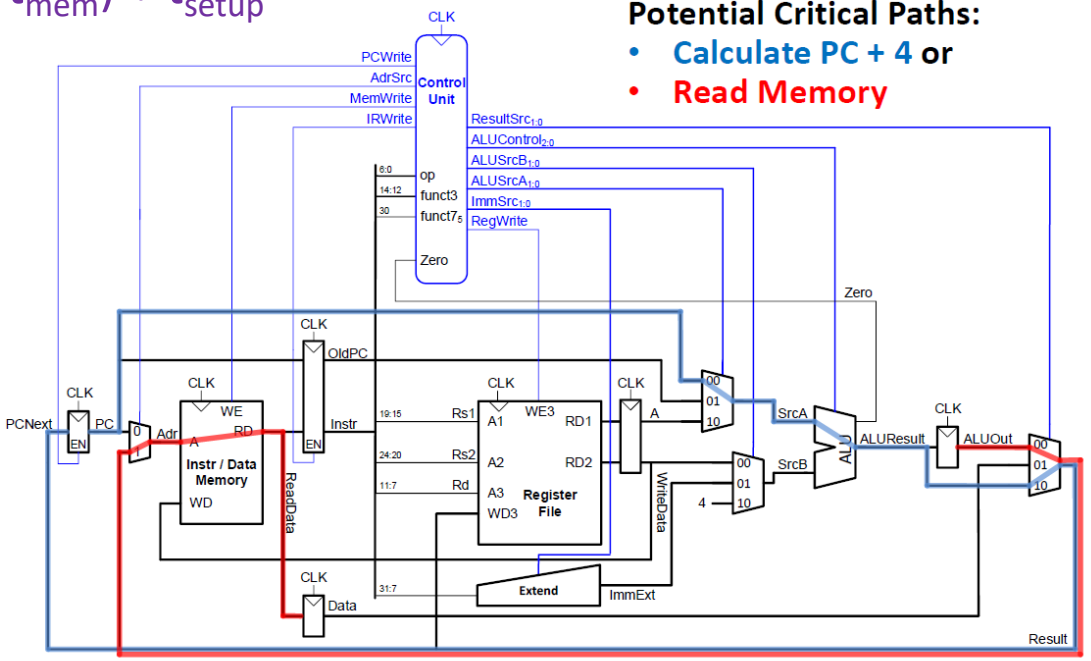
ali

- S_3 (MemRead):

$$t_{CPE,min} = t_{pcq(ALUOut)} + 2t_{mux} + t_{mem} + t_{setup(Data)},$$

Torej:

$$t_{CPE,min} = t_{pcq} + 2t_{mux} + \max(t_{ALE}, t_{mem}) + t_{setup}$$



- Primer:
  a) Kolikšna je najmanjša možna perioda ure, če uporabimo enake elemente (glej Tabelo spodaj), kot pri primeru enocikelnega procesorja?
  b) Koliko časa traja v povprečju en ukaz, če vzamemo primer C-prevajalnika?
  c) Koliko časa se izvaja program z 10^9 ukazi, če vpliva predpomnilnika ne upoštevamo?

- Izračuni:

a) $t_{CPE,min} = 30 + 2*25 + 250 + 20 = 350$ ps

b) $CPI_{idealni} * t_{CPE} = 4{,}09 * 350$ ps $= 1{,}432$ ns

c) Program z $I = 10^9$ ukazi traja:

$I * CPI_{idealni} * t_{CPE} = 1{,}432$ s

Parameter	Element	Zakasnitev [ps]
$t_{pcq(PC)}$	register propagation clock-to-q	30
t_{setup}	vzpostavitev registra	20
t_{mux}	multipleksor	25
t_{ALE}	ALE	200
t_{mem}	branje iz pomnilnika	250
t_{RFread}	branje registrov (register file)	150
$t_{RFsetup}$	vzpostavitev RF	20

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
- Zmogljivost (Performance) je obratno sorazmerna s časom izvajanja programa:

$$Zmog \propto \frac{1}{CPEčas}$$

- Pohitritev (Speed-up) pri primerjavi 2 procesorjev (ali pa stare in nove verzije enega procesorja):

$$S = \frac{Zmog_A}{Zmog_B} = \frac{CPEčas_B}{CPEčas_A}$$

- 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:

$$CPE\ \check{c}as = \frac{\check{S}tevilo\ \ ukazov}{MIPS \cdot 10^6}$$

  - Tudi MIPS ni popolnoma merodajen:
    - odvisen od števila in vrste ukazov
      - pri enostavnejših ukazih je MIPS večji (čeprav jih potrebujemo več)
    - odvisen od programa
    - celo *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