Arhitektura i organizacija računara

- Karakteristike CISC i RISC arhitektura
- Upravljačka jedinica računara

CISC (Complex Instruction Set Computers)

- Tokom 60-tih i 70-ih god. 20. veka osnovni pravac u razvoju računara ogledao se u povećanju kopleksnosti CPU. Ta kompleksnost se ogledala u:
  - Povećanju broja instrukcija (120-350)
  - Povećanju kompleksnosti instrukcija
  - Povećanju mogućih načina adresiranja (npr. MC 68020 je imao 18 adresnih režima)
  - Veliki broj specijalizovanih a mali broj registara opšte namene
  - Instrukcije su različite dužine (od 1 do 11 reči kod MC 68020)
  - Mikroprogramsko upravljanje (zahteva se velika mikroprogramska memorija zbog složenih instrukcija)
- Razlozi: poboljšanje performansi i pojednostavljenje pisanja kompilatora

CISC – faze u izvršenju instrukcija

- IF – pribavljanje instrukcije (Instruction fetch)
- ID – dekodiranje instrukcije (Instruction Decode)
- EA – izračunavanje efektivne adrese operanada
- OF – pribavljanje operanada (Operand Fetch)
- EXE – izvršenje instrukcije (Execution)

Primer Motorola 680X0

18 addressing modes:

- Data register direct.
- Address register direct.
- Immediate.
- Absolute short.
- Absolute long.
- Address register indirect.
- Address register indirect with postincrement.
- Address register indirect with predecrement.
- Address register indirect with displacement.
- Address register indirect with index (8-bit).
- Address register indirect with index (base).
- Memory inderect postindexed.
- Memory indirect preindexed.
- Program counter indirect with index (8-bit).
- Program counter indirect with index (base).
- Program counter indirect with displacement.
- Program counter memory indirect postindexed.
- Program counter memory indirect preindexed.

Operand size:

- Range from 1 to 32 bits, 1, 2, 4, 8, 10, or 16 bytes.

Instruction Encoding:

- Instructions are stored in 16-bit words.
- the smallest instruction is 2-bytes (one word).
- The longest instruction is 5 words (10 bytes) in length.

Primer Intel X86, 386/486/Pentium

12 adresinh režima:

- Register.
- Immediate.
- Direct.
- Base.
- Base + Displacement.
- Index + Displacement.
- Scaled Index + Displacement.
- Based Index.
- Based Scaled Index.
- Based Index + Displacement.
- Based Scaled Index + Displacement.
- Relative.

veličina operanada:

- Can be 8, 16, 32, 48, 64, or 80 bits long.
- podržava rad sa stringovima.

Instruction Encoding:

- The smallest instruction is one byte.
- The longest instruction is 12 bytes long.
- The first bytes generally contain the opcode, mode specifiers, and register fields.
- The remainder bytes are for address displacement and immediate data.

RISC (Reduced Instruction Set Computers)

- RISC prilaz je upravo suprotan CISC prilazu
- Tokom višegodišnjeg praćenja i merenja programa ustanovljeno je da se samo 25% instrukcija iz kompleksnog skupa instrukcija često koristi (95% vremena)
- 75% instrukcija koje po pravilu zahtevaju velike mikrokodove i veliku mikroprogramsku memoriju se veoma retko koriste
- **IDEJA**: Instrukcije koje se često koriste implementirati hardverski, a instrukcije koje se retko koriste softverski
- Kod CISC upravljačka jedinica zauzima 40-60% oblasti na čipu, a kod RISC približno 10%.
- Ostatak oblasti na čipu se može iskoristiti za druge komponente – registre ili on chip cache

RISC - karakteristike

- Relativno mali broj instrukcija (obično < 100: i860 – 82 instrukcije, MC 88100 – 51 instrkcija)
- Sve instrukcije su iste dužine (32 bita)
- Mali broj načina adresiranja
- Pristup memoriji se obavlja isključivo preko LOAD i STORE instrukcija
- Veliki broj registara opšte namene (32 – 192).
- Najveći broj instrukcija je tipa registar-u-registar
- Hardversko (a ne mikroprogramsko) upravljanje
- Jako izražena protočnost u obradi instrukcija
- Odvojeni keš za instrukcije i za podatke
- Moćni kompilatori

Primer RISC - SPARC

Broj instrukcija: 52

5 addressing modes:

- Register indirect with immediate displacement.
- Register inderect indexed by another register.
- Register direct.
- Immediate.
- PC relative.

Operand sizes:

- Four operand sizes: 1, 2, 4 or 8 bytes.

Instruction Encoding:

- Instruction set has 3 basic instruction formats with 3 minor variations.
- All are 32 bits in length.

Primer RISC ISA:

MIPS

Adresni režimi:

- Register direct.
- Indirektno
- Neposredno (Immediate).
- bazno
- PC-relative.

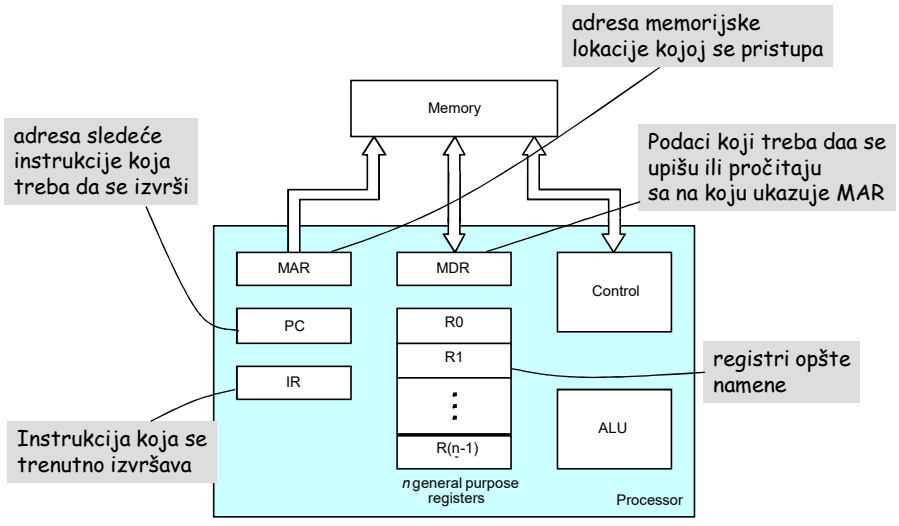
Operand sizes:

- Four operand sizes: 1, 2, 4 or 8 bytes.

Instruction Encoding:

- Instruction set has 3 different formats.
- All are 32 bits in length.

Organizacija procesora

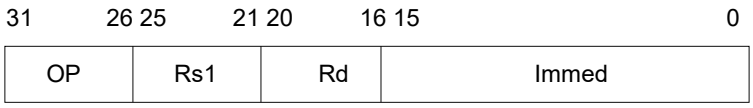


Formati instrukcija RISC procesora – primer MIPS procesora

- Sve instrukcije su 32-bitne
  - tri formata
    - I
    - R
    - J

I format instrukcija

- I format (load, store, branch, ALU operacije sa neposrednim operandom)



OP – kod operacije
Rs1 – izvorni operand za ALU
- bazni registar za generisanje memorijske adrese kod load i store
- registar čiji se sadržaj testira za slučaj barnch

Rd – polje odredišnog registra za slučaj ALU op. i load
- polje regista čiji se sadržaj pamti u mem. za slučaj store

Rd=0 za branch
Immed – neposredni operand u slučaju ALU instrukcije
- Pomeraj koji se dodaje baznom registru Rs1 za slučaj LOAD/STORE instrukcija
- Pomeraj koji se dodaje PC za slučaj naredbe grananja

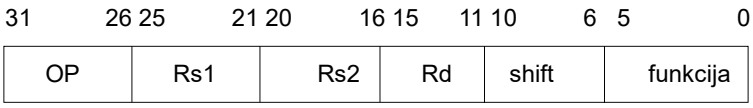
Primer instrukcija I formata

rd Imm Rs1

- LW R1, 30 (R2) dejstvo R1← Mem[30 +[R2]]
- SW 500(R4), R3 dejstvo Mem[500+[R4]]←[R3]
- ADDI R1,R2,#3 dejstvo R1 ←[R2]+3
- BEQZ R1, ime dejstvo if R1=0 then PC← PC+ime

R format instrukcija

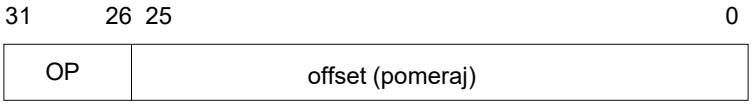
- R format (ALU operacije tipa registar-u registar)



OP+ funkcija – kod operacije
Rs1 – prvi izvorni operand za ALU
Rs2 – drugi izvorni operand
Rd – polje odredišnog registra
Shift –veličina pomeraja kod naredbi pomeranja
PRIMER: ADD R1,R2,R3 dejstvo R1 ← [R2]+[R3]

J format instrukcija

* J format – jump instrukcija

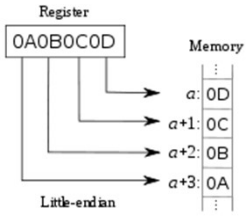


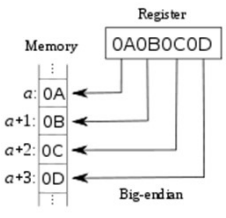
OP– kod operacije
offset – pomeraj koji se dodaje PC

PRIMER: JMP *ime* dejstvo PC ←PC+*ime*

Adresiranje podataka i redosled bajtova u podacima

* Memorija je linearna
  - najmanja adresibilna jedinica memorije je bajt
    - adrese čine niz oblika 0,1,2,...
* kako se u memoriju smeštaju podaci koji su veći od 1 bajta?
  - postoje dve konvencije za redosled smeštanja bajtova unutar većeg objekta (reči, dvostuke reči,...)
    - Little endian –(ili adresiranje repa)
      - bajt najmanje težine nalazi se na najmanjoj adresi
    - Big endian – (ili adresiranje glave)
      - bajt najveće težine nalazi se na najmanjoj adresi
  - redosled smeštanja podataka je bitan ako treba da komuniciraju dva računara koji koriste različite načine smeštanja podataka .





Poravnati (aligned) i neporavnati (nonaligned) pristup memoriji

- Drugo važno pitanje kada je u pitanju pristup memoriji je, da li kada se pristupa podacima koji su veći od 1 bajta pristup mora biti poravnat ili ne
  - pristup objektu veličine S bajtova je poravnat (aligned) ako važi da je
    - A mod S=0, gde je A adresa objekta (u suprotnom pristup nije poravnat)
    - npr ako je S=4, ako je reč o poravnatom pristupu onda A može biti 0,4,8,...

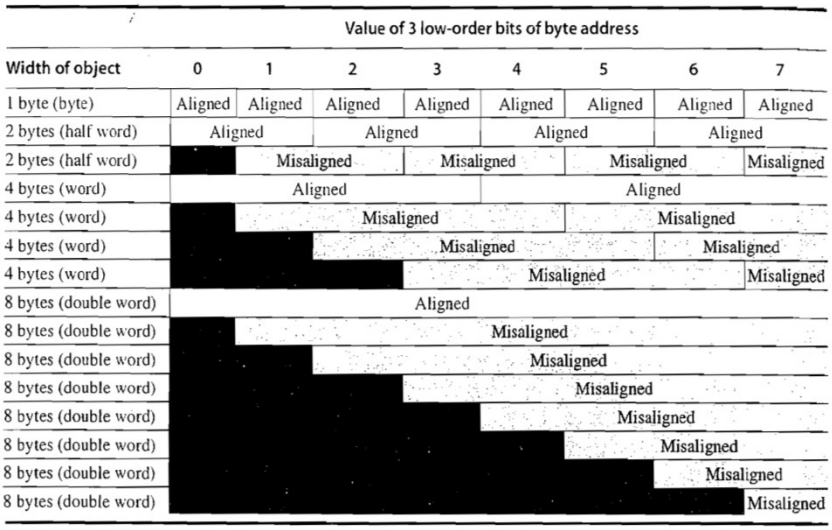
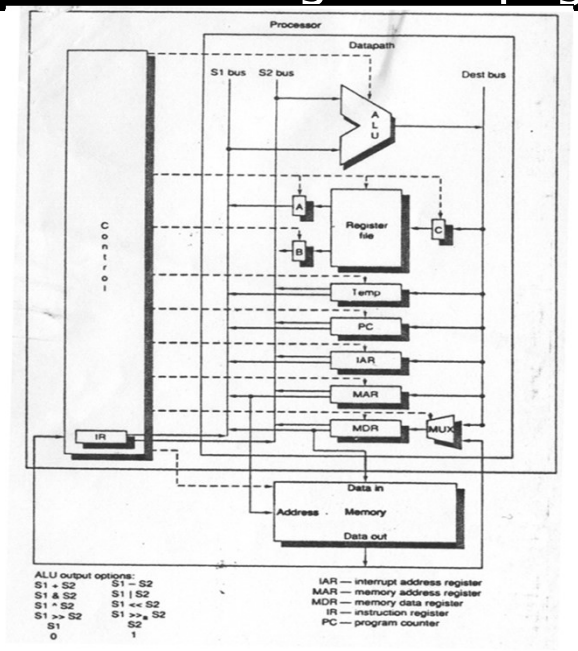


Figure A.5 Aligned and misaligned addresses of byte, half-word, word, and double-word objects for byte-addressed computers. For each misaligned example some objects require two memory accesses to complete. Every aligned object can always complete in one memory access, as long as the memory is as wide as the object. The figure shows the memory organized as 8 bytes wide. The byte offsets that label the columns specify the low-order 3 bits of the address.

izvršenje instrukcije

- Obavlja se u dve faze:
  - Priprema (pribavljanje) instrukcije
  - izvršenje instrukcije
- Pribavljanje instrukcije:
  - instrukcija se pribavlja iz memorije sa adrese na koju ukazuje PC
  - pribavljena instrukcija se smešta u instrukcioni registar (IR)
- Izvršenje instrukcije:
  - Instrukcija koja se nalazi u IR se dekodira da bi se utvrdilo koja operacija treba da se obavi.
  - pribavljaju se operandi iz memorije ili registara.
  - izvršava se operacija
  - rezultat se pamti (u registre ili memoriju)
- Ovaj ciklus pribavljanje/izvršenje se ponavlja za svaku instrukciju

RISC arhitektura – globalni pogled



Staza podataka tipične RISC arhitektura

- 32-192 registra opšte namene
- Sve instrukcije su 32-bitne
  - tri formata (I, R,J)
- Svaka instrukcija se izvršva u najviše 5 koraka
  - Pribavljanje (Instruction fetch – IF)
  - Dekodiranje instrukcije i pribavljanje operanda (Instruction decode – ID)
  - Izvršenje / izračunavanje efektivne adrese (EX)
  - obraćanje memoriji / okončanje grananja (MEM)
  - upis rezultata u registarski fajl (Write Back – WB)

Faze izvršenja

- Svaka integer instrukcija RISC procesora se može obaviti za najviše pet klok ciklusa:

1 Instruction fetch cycle (IF):

```
IR ← Mem[PC]
PC ← PC + 4
```

2 Dekodiranje instrukcije i pribavljanje operanda (ID):

```
A   ← Regs[Rs1];
B   ← Regs[Rs2];
Imm ← ((IR15)16##IR 15..0)
```

sign-extended immediate field of IR

Napomena: IR (instrukcioni registar),
A, B, Imm su privremeni registri

Faze izvršenja- nastavak

3 Execution/Effective address cycle (EX):

- **Memory reference:**

  ALUOutput ← A + Imm;

- **Register-Register ALU instruction:**

  ALUOutput ← A func B;

- **Register-Immediate ALU instruction:**

  ALUOutput ← A op Imm;

- **Branch:**

  ALUOutput ← PC + Imm;
  Cond ← A op 0 (op je relacioni operator definisan kodm instrukcije)

- **Jump:** PC←PC[31..28]||(IR[25..0]<<2)

Faze izvršenja- nastavak

4 Obraćanje memoriji /okončanje grananja (MEM):

- **Obraćanje memoriji:**

  LMD ← Mem[ALUOutput] ili
  Mem[ALUOutput] ← B;

- **Branch:**

  if (cond) PC ← ALUOutput

napomena: LMD (load memory data) register

Faze izvršenja- nastavak

5 Write-back cycle (WB) – upis u registarski fajl:

- **Register-Register ALU instruction:**

  $Reg[IR_{15..11}] \leftarrow ALUOutput;$

- **Register-Immediate ALU instruction:**

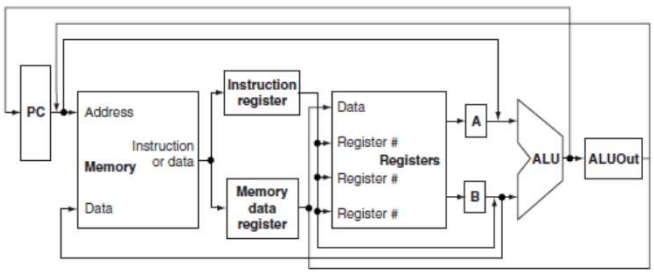
  $Reg[IR_{20..16}] \leftarrow ALUOutput;$

- **Load instruction:**

  $Reg[IR_{20..16}] \leftarrow LMD;$

Napomena: LMD (load memory data) register

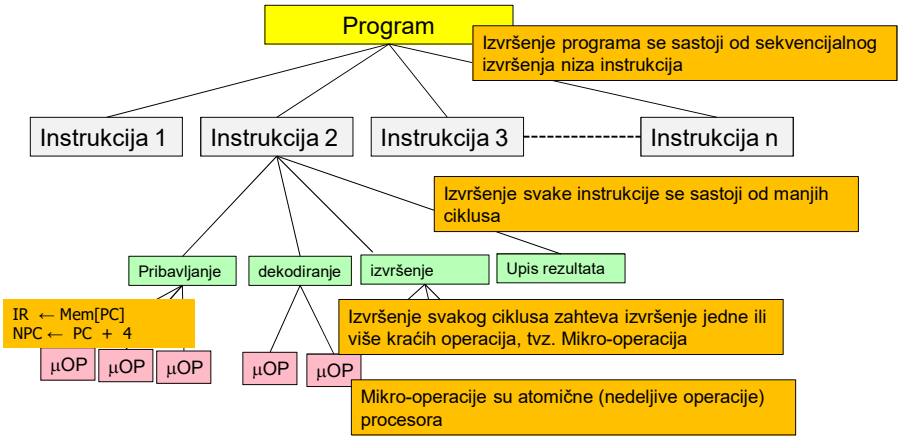
Globalna struktura staze podataka



Izvršenje programa

- Računar izvršava program koji se sastoji od niza instrukcija
- Izvršenje svake instrukcije se sastoji od više faza (ciklusa)
  - pribavljanje, dekodiranje, izvršenje...
- Svaka od ovih faza zahteva izvršenje niza mikro-koraka, pri čemu svaki od njih uključuje korišćenje nekih procesorskih registara.
  - Ove mikro-korake nazivamo mikro-operacijama
    - Prefiks mikro se odnosi na činjenicu da je svaki korak veoma jednostavan

Izvršenje programa



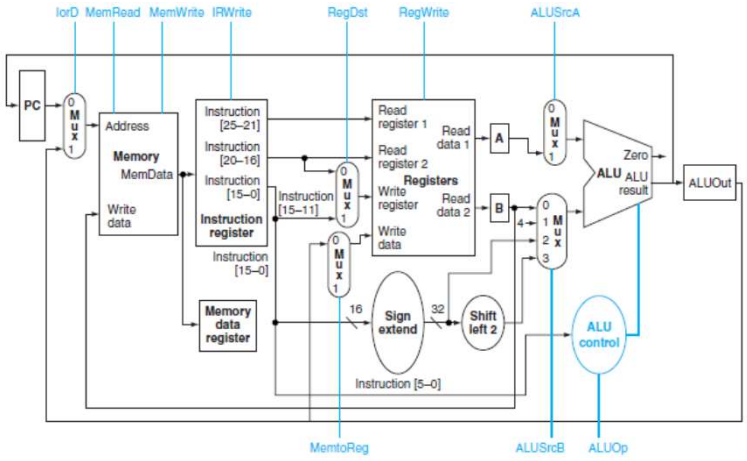
Sve mikro-operacije se mogu svrstati u sledeće kategorije

- Prenos podataka iz jednog registra u drugi
- Prenos podataka iz registra na spoljni interface (npr. Sistemsku magistralu)
- Prenos podataka sa spoljnjeg interfeisa u registar
- Obavljanje aritmetičke ili logičke operacije.

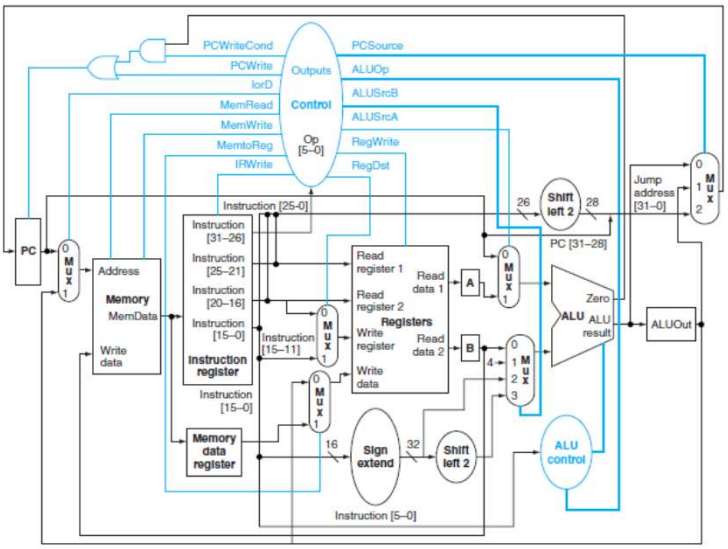
Upravljačka jedinica procesora (Control Unit –CU)

- elementarnih aktivnosti koje obavljaju u svakoj od faza izvršenja instrukcije nazivamo mikrooperacijama
  - Mikroperacija može biti prenos podataka iz jednog u drugi registar CPUa, čitanje podatka iz memorije ili upis podatka u memoriju, elementarne operacije nad podacima i drugo.
    - npr.
      - A ← Regs[Rs1];
      - ALUOutput ← A func B
      - MDR ← Mem[ALUOutput] ili
      - Mem[ALUOutput] ← B;
  - Svaka mikrooperacija zahteva aktiviranje određenih upravljačkih tačkaka elemenata hardvera koji učestvuje u izvršenju određenih mikrooperacija.
- Funkcija upravljačke jedinice je generisanje odgovarajućih upravljačkih signala, u odgovarajućim vremenskim trenutcima, koji su neophodni za rad računara

Staza podataka sa označenim upravljačkim signalima



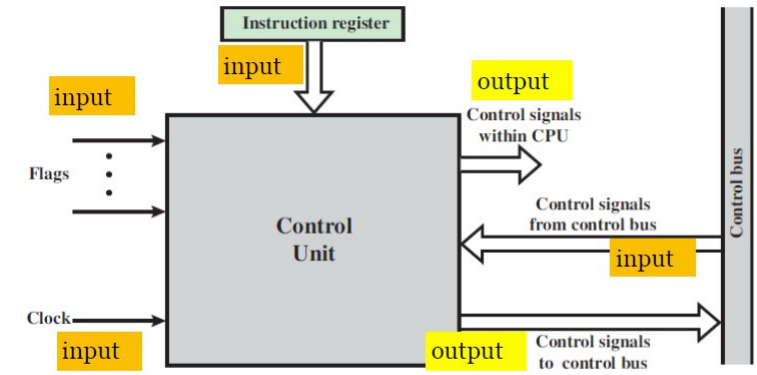
Staza podataka i CU



Upravljačka jedinica

- Da bi CU obavljala svoju funkciju ona mora da ima odgovarajuće ulaze koji joj omogućavaju da odredi stanje sistema
  - Clock
  - Instrukcioni registar (kod operacije)
  - Flegovi (iz registra koda uslova)
  - Upravljački signali sa control bus-a (npr. zahtev za prekidom od U/I uređaja)
- Izlaze koji omogućavaju CU da upravlja radom sistema
  - Upravljački signali unutar procesora
    - uzrokuju prenos podataka
    - aktiviraju određene ALU funkcije
  - Upravljački signali koji se šalju na magistralu
    - ka memoriji ili U/I uređajima

Blok dijagram CU



Upravljačka jedinica

- Upravljački signali se mogu generisati na dva načina
  - hardverski (tvz. upravljačke jedinice sa direktnim upravljanjem)
    - upravljačka jedinica je realizovana kao sekvencijalno logičko kolo
  - mikroprogramski
    - upravljački signali se nalaze u posebnoj memoriji (mikro-programska memorija) koja može biti realizovna kao ROM ili RAM memorija
    - Mikroprogramsko upravaljanje se realizuje pomoću sekvence mikroinstrukcija

Hardversko upravljanje

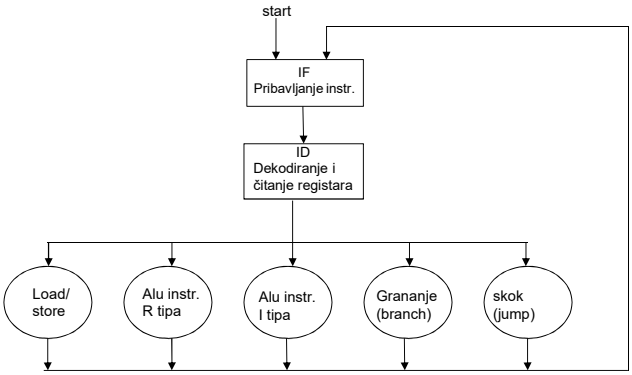
- Postoji nekoliko metoda za sistematsko projektovanje CU
  - Metod baziran na tabeli stanja (state table method)
    - CU se projektuje kao konačni automat koji se može naći u konačnom broju različitih stanja.
  - Metod baziran na kolima za kašnjenje (delay element method)
    - Kola za kašnjenje se koriste za generisanje upravljačkih signala koji aktiviraju određene kontrolne linije
  - Metod baziran na brojaču sekvenci (Sequence counter method)
    - Koristi se kružni brojač da obavi sekvenciranje (definisanje trenutaka kada treba da se aktivraju određene kontrolne linije).

Metod baziran na tablici stanja

- Upravljačku jedinicu procesora možemo posmatrati kao konačni automat, zadat sledećim elementima:
  - •skupom stanja upravljačke jedinice
    - S={S0, S1, ... , Sn-1},
  - skupom ulaznih upravljačkih signala
    - X={x1, x2, ... , xm},
  - skupom izlaznih upravljačkih signala
    - Y={y1, y2, ... , yk},
  - funkcijom promene stanja FS: (X,S) ⇒ S, i
  - funkcijom izlaza FY: (X,S) ⇒ Y.
- Postoje dva tipa konačnih automata: Mealy-ev i Moore-ov automat.
  - Mi ćemo ovde koristiti Moore-ov automat, čija funkcija izlaza ima oblik FY: S ⇒ Y.
    - To znači da su izlazni signali određeni samo stanjem u kome se automat nalazi, nezavisno od ulaznih signala.

CU kao konačni automat

- Na osnovu aktivnosti po fazama možemo formirati polazni graf konačnog automata koji prikazuje promene stanja



Detaljan prikaz
promene stanja
sa označenim uprvljačkim signalima

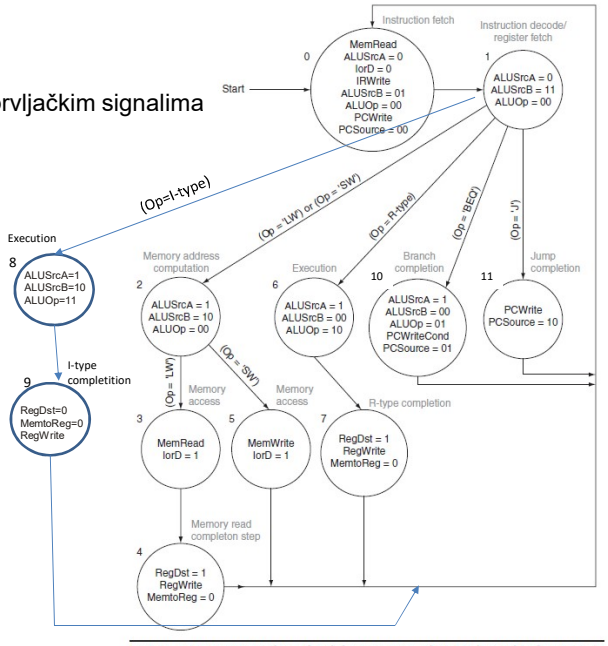
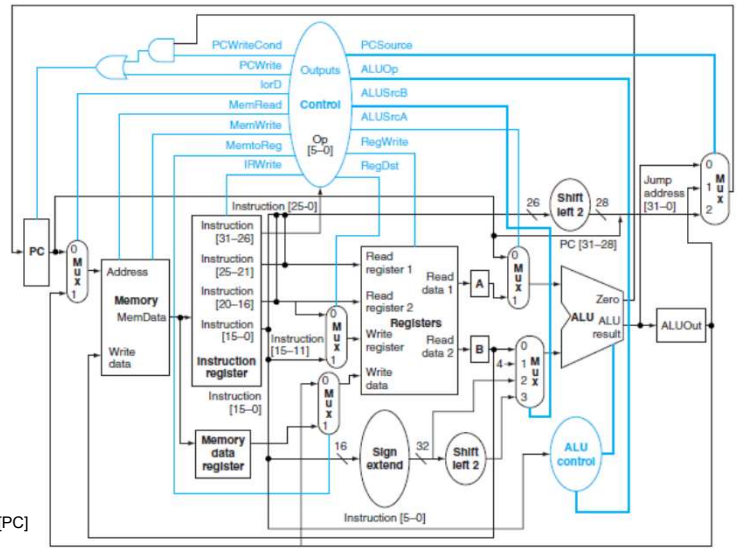


FIGURE 5.38 The complete finite state machine control for the datapath shown in Figure 5.28. The labels on the arcs are conditions that are tested to determine which state is the next state; when the next state is unconditional, no label is given. The labels inside the nodes indicate the output sig-



IorD = 0
MemRead
IRWrite
IR←Mem[PC]

ALUSrcA = 0
ALUSrcB = 01
ALUOp = 00
PCSource = 00
PCWrite
PC←PC+4

Blok šema konačnog automata

- Konačni automat može se implementirati sekvencijalnom prekidačkom mrežom, koja se sastoji od kombinacione prekidačke mreže i memorijskih elemenata (registar stanja)
  - Memorijski elementi mogu biti flip-flopovi tipa D, RS, JK ili T
  - Kombinaciona mreža na osnovu koda operacije i trenutnog stanja generiše upravljačke signale za stazu podataka i upravljačke signale koji izazivaju promenu stanja konačnog automata

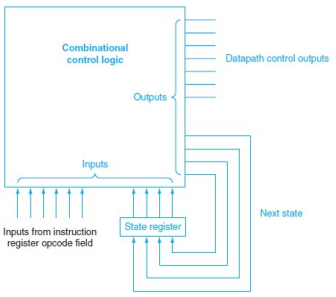


FIGURE 5.37 Finite state machine controllers are typically implemented using a block of combinational logic and a register to hold the current state. The outputs of the combinational

Projektovanje CU metodom tablice stanja

- Na osnovu grafa promene stanja može se napraviti tabilca prelaza/izlaza konačnog automata (Moore-ov)

Tablica prelaza/izlaza konačnog automata

Stanje/ izlazni signali	Ulazni upravljački signali			
	x_1	x_2	...	x_m
S_0/ Y_0	S_{01}	S_{02}	...	S_{0m}
S_1/ Y_1	S_{11}	S_{12}	...	S_{1m}
...	...	...	...	...
S_{n-1}/ Y_{n-1}	$S_{(n-1)1}$	$S_{(n-1)2}$	...	$S_{(n-1)m}$

- Koristićemo D flip-flopove za realizaciju registra stanja
  - Broj potrebnih f-f određujemo na osnovu broja stanja konačnog automata Ns, kao $m = \lceil \log_2 N_s \rceil$
  - Tablica pobude D flip-flopa

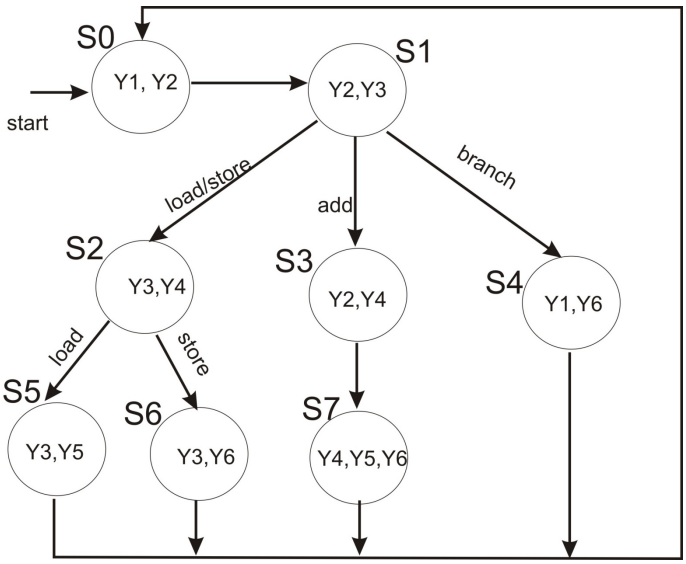
Q^i	Q^{i+1}	D
0	0	0
0	1	1
1	0	0
1	1	1

Primer 1

- Neka je skup mogućih stanja sistema
  - S={S0,S1,…,S7}
- Neka postoje samo 4 instrukcije
  - Load, store, add, branch
    - To znači da kod operacije ima samo 2 bita pomoću kojih kodiramo ove 4 instrukcije
- Neka je potrebno generisati 6 različitih upravljačkih signala
  - Y ={Y1, Y2,…,Y6}

Primer 1 (nast.)

- Promene stanja i upravljački signali koji se aktiviraju u određenim stanjima



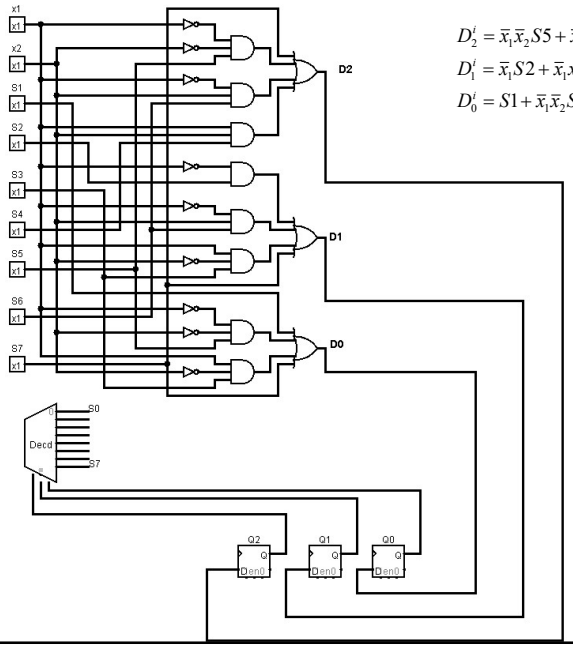
Tablica promene stanja i pobude f-f

- Pošto postoji 8 različitih stanja u kojima se sistem može naći u toku rada, potrebna su nam 3 D f-f za kodiranje stanja

S_i	x1x2	S_{i+1}	$Q_2^{i+1}Q_1^{i+1}Q_0^{i+1}$	$D_2^iD_1^iD_0^i$
S0	x x	S1	0 0 1	0 0 1
S1	0 x	S2	0 1 0	0 1 0
S2	0 0	S5	1 0 1	1 0 1
S2	0 1	S6	1 1 0	1 1 0
S1	1 0	S3	0 0 1	0 0 1
S1	1 1	S4	1 0 0	1 0 0
S3	x x	S7	1 1 1	1 1 1
S5	x x	S0	0 0 0	0 0 0
S6	x x	S0	0 0 0	0 0 0
S4	x x	S0	0 0 0	0 0 0
S7	x x	S0	0 0 0	0 0 0

$$D_2^i = \bar{x}_1\bar{x}_2S5 + \bar{x}_1x_2S6 + x_1x_2S4 + S7$$
$$D_1^i = \bar{x}_1S2 + \bar{x}_1x_2S6 + x_1\bar{x}_2S3 + S7$$
$$D_0^i = S1 + \bar{x}_1\bar{x}_2S5 + x_1\bar{x}_2S3 + S7$$

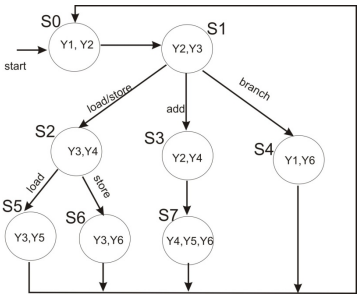
Logika koja upravlja promenom stanja



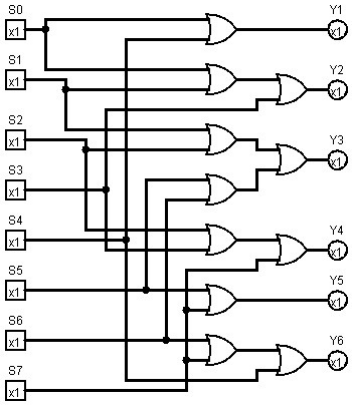
$$D_2^i = \bar{x}_1\bar{x}_2S5 + \bar{x}_1x_2S6 + x_1x_2S4 + S7$$
$$D_1^i = \bar{x}_1S2 + \bar{x}_1x_2S6 + x_1\bar{x}_2S3 + S7$$
$$D_0^i = S1 + \bar{x}_1\bar{x}_2S5 + x_1\bar{x}_2S3 + S7$$

Primer 1 (nast.)

- Upravljački signali Y1 do Y6
  - Dobijaju se direktno iz grafa promene stanja jer je reč o Moore-ovom automatu



$$Y1 = S0 + S4$$
$$Y2 = S0 + S1 + S3$$
$$Y3 = S1 + S2 + S5 + S6$$
$$Y4 = S2 + S3 + S7$$
$$Y5 = S5 + S7$$
$$Y6 = S4 + S6 + S7$$



PRIMER 2

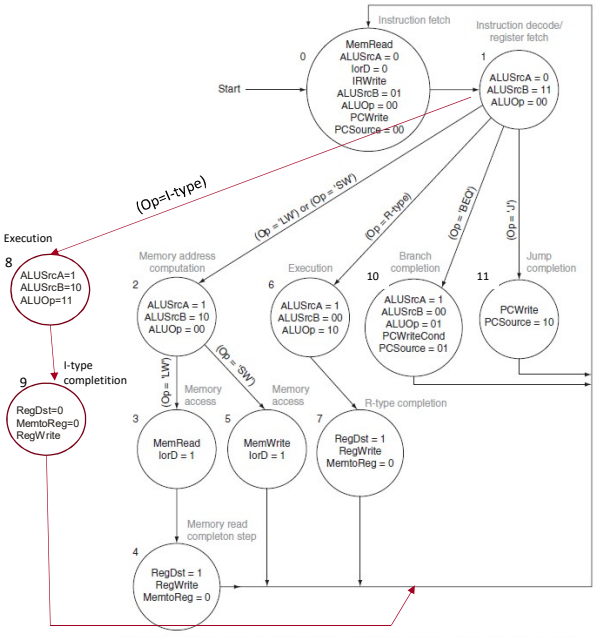


FIGURE 5.38 The complete finite state machine control for the datapath shown in Figure 5.28. The labels on the arcs are conditions that are tested to determine which state is the next state; when the next state is unconditional, no label is given. The labels inside the nodes indicate the output sig-

Projektovanje CU metodom tablice stanja

Tablica pobude flip-flopova

Stanja S0-S11
kodiramo sa 4 bita

Si	$Q_3^i Q_2^i Q_1^i Q_0^i$	a5a4a3a2a1a0	$Q_3^{i+1} Q_2^{i+1} Q_1^{i+1} Q_0^{i+1}$	$D_3^i D_2^i D_1^i D_0^i$
S0	0000	xxxxxx	0001	0001
S1	0001	100011-LW	0010	0010
S1	0001	101011-SW	0010	0010
S1	0001	000000-Rft	0110	0110
S1	0001	001100-ANDI	1000	1000
S1	0001	000100-BEQ	1010	1010
S1	0001	000010-J	1011	1011
S2	0010	100011	0011	0011
S2	0010	101011	0101	0101
S3	0011	xxxxxx	0100	0100
S4	0100	xxxxxx	0000	0000
S5	0101	xxxxxx	0000	0000
S6	0110	xxxxxx	0111	0111
S7	0111	xxxxxx	0000	0000
S8	1000	xxxxxx	1001	1001
S9	1001	xxxxxx	0000	0000
S10	1010	xxxxxx	0000	0000
S11	1011	xxxxxx	0000	0000

Projektovanje CU metodom tablice stanja

- Na osnovu tablice pobude flip-flopova određujemo logičke izraze za signale pobude flip-flopova, D_0-D_3

Logički izrazi pobude flip-flopova

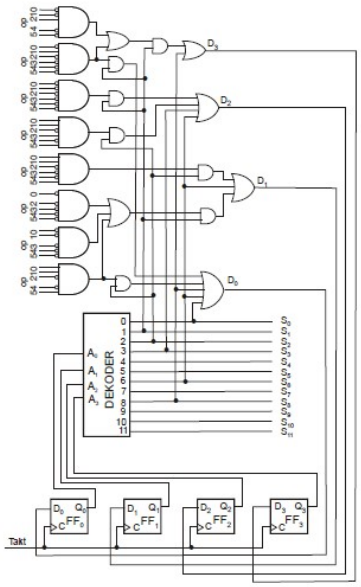
$$D_3 = \overline{Q}_3\overline{Q}_2\overline{Q}_1Q_0(\overline{a}_5\overline{a}_4a_2\overline{a}_1\overline{a}_0 \vee \overline{a}_5\overline{a}_4\overline{a}_3\overline{a}_2a_1\overline{a}_0) \vee Q_3\overline{Q}_2\overline{Q}_1\overline{Q}_0$$

$$D_2 = \overline{Q}_3\overline{Q}_2\overline{Q}_1Q_0 \cdot \overline{a}_5\overline{a}_4\overline{a}_3\overline{a}_2\overline{a}_1\overline{a}_0 \vee \overline{Q}_3\overline{Q}_2Q_1\overline{Q}_0 \cdot a_5\overline{a}_4a_3\overline{a}_2a_1a_0 \vee$$
$$\vee \overline{Q}_3\overline{Q}_2Q_1Q_0 \vee \overline{Q}_3Q_2Q_1\overline{Q}_0$$

$$D_1 = \overline{Q}_3\overline{Q}_2\overline{Q}_1Q_0(a_5\overline{a}_4\overline{a}_2a_1a_0 \vee \overline{a}_5\overline{a}_4\overline{a}_3\overline{a}_1\overline{a}_0 \vee \overline{a}_5\overline{a}_4\overline{a}_3\overline{a}_2\overline{a}_0) \vee$$
$$\vee \overline{Q}_3\overline{Q}_2Q_1\overline{Q}_0 \cdot a_5\overline{a}_4\overline{a}_3\overline{a}_2a_1a_0 \vee \overline{Q}_3Q_2Q_1\overline{Q}_0$$

$$D_0 = \overline{Q}_3\overline{Q}_2\overline{Q}_1\overline{Q}_0 \vee \overline{Q}_3\overline{Q}_2\overline{Q}_1Q_0 \cdot \overline{a}_5\overline{a}_4\overline{a}_3\overline{a}_2a_1\overline{a}_0 \vee$$
$$\vee \overline{Q}_3\overline{Q}_2Q_1\overline{Q}_0 \cdot a_5\overline{a}_4\overline{a}_2a_1a_0 \vee \overline{Q}_3Q_2Q_1\overline{Q}_0 \vee Q_3\overline{Q}_2\overline{Q}_1\overline{Q}_0$$

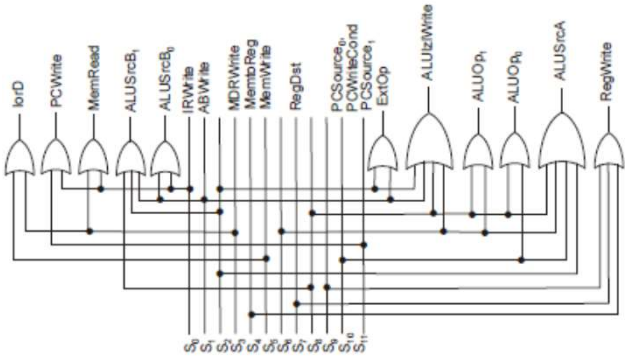
Deo logike koji upravlja promenom stanja



Zavisnost upravljačkih signala od stanja CU

PCWriteCond = S10	RegWrite = S4+S7+S9
PCWrite = S0+S11	ALUSrcA = S2+S6+S8+S10
IorD = S3+S5	ALUSrcB1 = S1+S2+S8
MemRead = S0+S3	ALUSrcB0 = S0+S1
MemWrite =S5	ALUOp1 = S6+S8
MDRWrite =S3	ALUOp0 = S8+S10
IRWrite =S0	PCSrc1 = S11
RegDst =S7	PCSrc0 = S10
MemtoReg =S4	

Logika za generisanje upravljačkih signala na osnovu stanja CU



Projektovanje CU metodom elemenata za kašnjenje

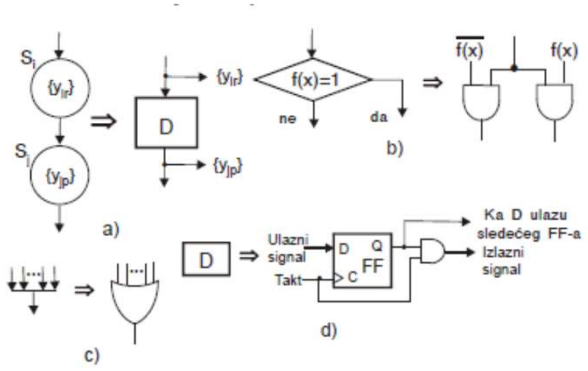
- Polazi se opet od dijagrama toka promene stanja
- Potrebno je obezbediti da se grupa upravljačkih signala aktivira u određenom redosledu
  - Između promene stanja mora da postoji određeno kašnjenje između aktiviranja signala
    - Niz elemenata za kašnjenje (D flip-flopovi) može se iskoristiti da generiše upravljačke signale jedan za drugim
- Deo sekvencijalne mreže koji je zadužen za promenu stanja CU se može implementirati na osnovu dijagrama toka

Projektovanje CU metodom elemenata za kašnjenje

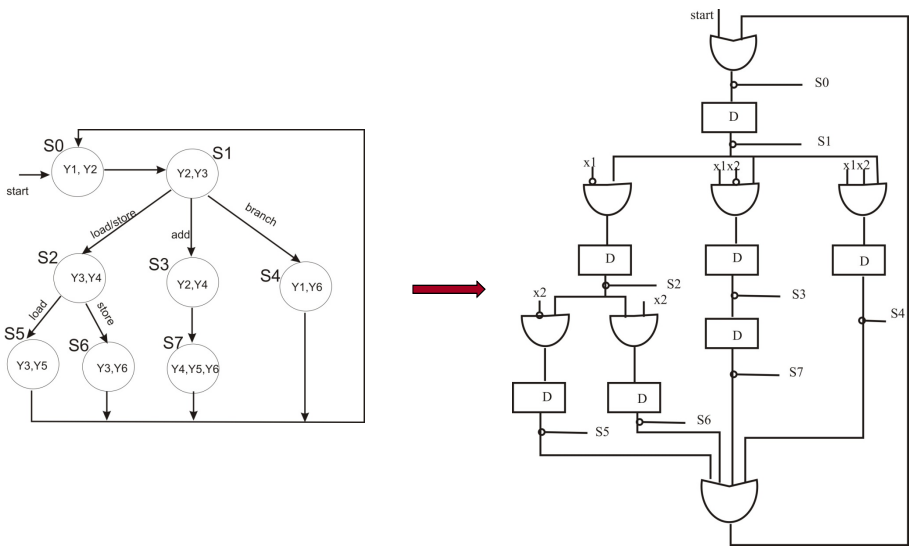
- Obavlja se preslikavanje grafa promene stanja u prekidačku mrežu
  - Pravila za projektovanje CU na osnovu dijagrama toka
    - Između svaka dva uzastopna stanja neophodno je ubaciti element za kašnjenje
    - Signali koji aktiviraju upravljačke linije ili učestvuju u njihovom formiranju uzimaju se direktno sa ulaza i izlaza elementa za kašnjenje (slučaj pod a) sa slike)
    - Blok odluke (uslovno grananje) se implementira parom AND kola (sl. b))
    - Slivanje n dolaznih grana u jednu odlaznu granu implementira se ILI logičkim elementom sa n ulaza (sl. c))

Pravlila konstrukcije

Preslikavanje elemenata dijagrama toka upravljanja u elemente prekidačke mreže koja implementira UJ



Za primer1:



Primer 2

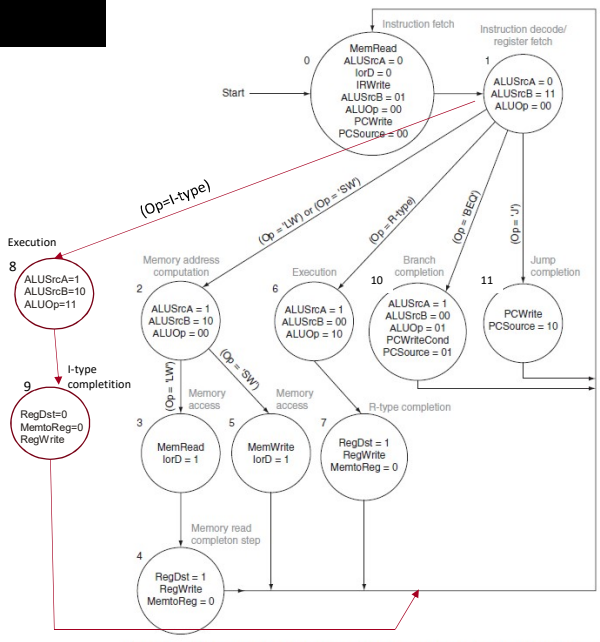
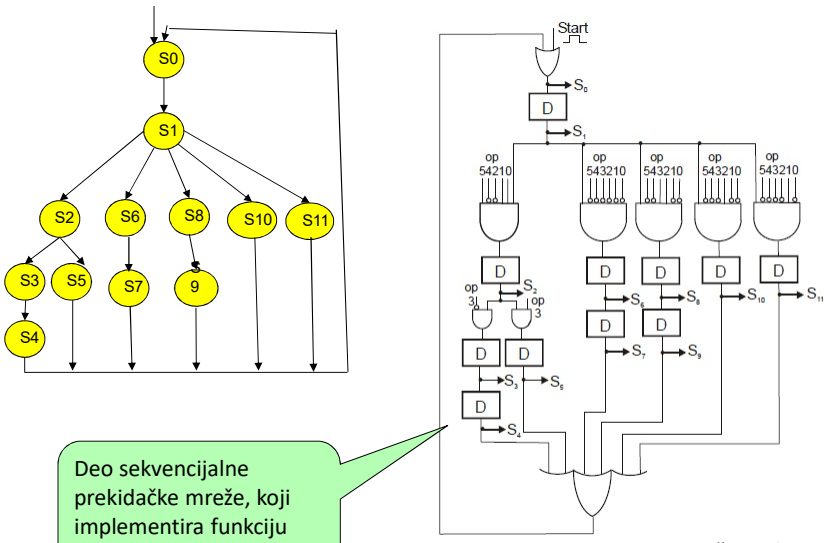


FIGURE 5.38 The complete finite state machine control for the datapath shown in Figure 5.28. The labels on the arcs are conditions that are tested to determine which state is the next state; when the next state is unconditional, no label is given. The labels inside the nodes indicate the output sig-

Primer2: Izgled mreže koja generiše signale S0-S11

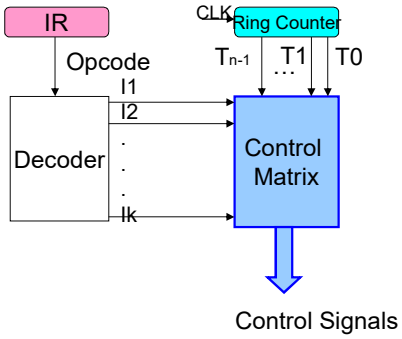


Metod baziran na brojaču sekvenci (sequence counter metod)

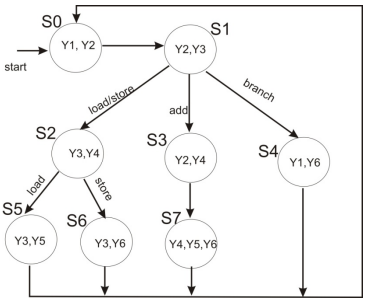
- Potrebno je identifikovati maksimalni broj ciklusa potrebnih za izvrešenje instrukcija
  - analizirali smo primer u kojem izvršenje svake instrukcije traje najviše 5 ciklusa (IF, ID, EX, MEM, WB)
  - Da bi se identifikovali ciklusi koristi se kružni brojač
- Projektovati kombinacionu prekidačku mrežu za generisanje sekvence upravljačkih signala
  - Upravljački signali zavise od dekodirane isntrukcije i faze izvršenja

CU bazirana na kružnom brojaču

- CU se sastoji od sledećih funkcionalnih jedinica:
  - kružni brojač
  - dekoder instrukcija
  - upravljačka matrica (control matrix)
- Ulaz u CU je kod operacie instrukcije koja se trenutno nalazi u instrukcionom registru
- Upravljačke signale generiše upravljačka matrica

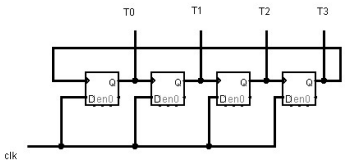


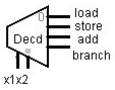
Primer 1



$$Y1 = T0 + T2 * Branch$$
$$Y2 = T0 + T1 + T2 * Add$$
$$Y3 = T1 + T2 * (Load + Store) + T3 * (Load + Store)$$
$$Y4 = T2 * (Load + Store) + T2 * Add + T3 * Add$$
$$Y5 = T3 * Load + T3 * Add$$
$$Y6 = T1 * Branch + T2 * Store + T2 * Add$$

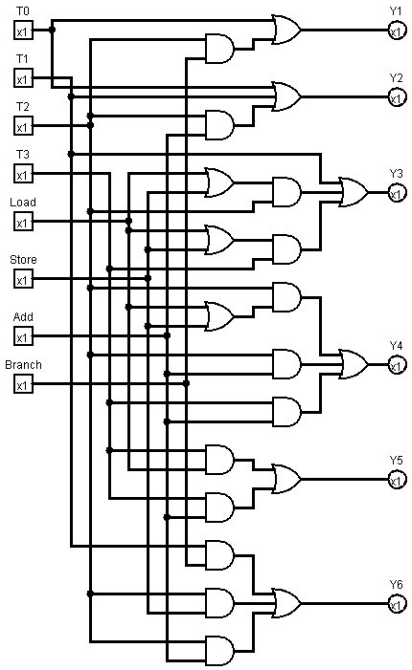
Kružni brojač modula 4





Dekoder sa 4 izlaza

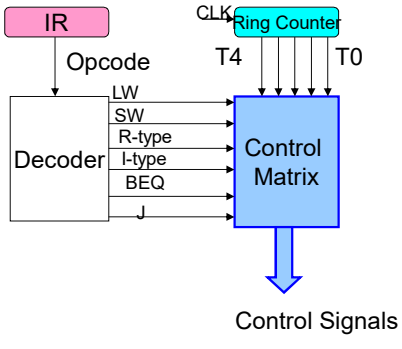
Primer1: upravljačka matrica



$$Y1 = T0 + T2 * Branch$$
$$Y2 = T0 + T1 + T2 * Add$$
$$Y3 = T1 + T2 * (Load + Store) + T3 * (Load + Store)$$
$$Y4 = T2 * (Load + Store) + T2 * Add + T3 * Add$$
$$Y5 = T3 * Load + T3 * Add$$
$$Y6 = T1 * Branch + T2 * Store + T2 * Add$$

Primer2: CU bazirana na kružnom brojaču

- CU se sastoji od sledećih funkcionalnih jedinica:
  - kružni brojač modula 5
  - dekoder instrukcija
  - upravljačka matrica (control matrix)
- Ulaz u CU je kod operacie instrukcije koja se trenutno nalazi u instrukcionom registru
- Upravljačke signale generiše upravljačka matrica



implementacija kružnog brojača

- Kružni brojač generiše sekvencu od 5 uzastopnih aktivnih signala koja se stalno ponavlja.
  - Kružni brojač je sinhronizovan sa sistemskim klokom (taktom) i on prvo aktivira liniju T0, zatim T1, itd.
    - Nakon aktiviranja T4, sekvenca se vraća na početak, tj počinje se od T0.

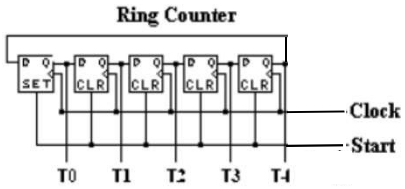


Figure 3. The Internal Organization of the Ring Counter

Dekoder

- Dekoder instrukcija ima ulaz koji potiče od opcode polja instrukcije, i na osnovu njega aktivira jednu i samo jednu od svojih izlaznih linija.
  - Svaka linija odgovara jednoj instrukciji iz seta instrukcija računara.
- Najvažniji deo hardverske CU je upravljačka matrica (control matrix).
  - Ona dobija ulaze sa kružnog brojača i dekodera instrukcija i obezbeđuje korektnu sekvencu upravljačkih signala neophodnih za izvršenje svake instrukcije

Detaljan prikaz
promene stanja
sa označenim uprvljačkim signalima

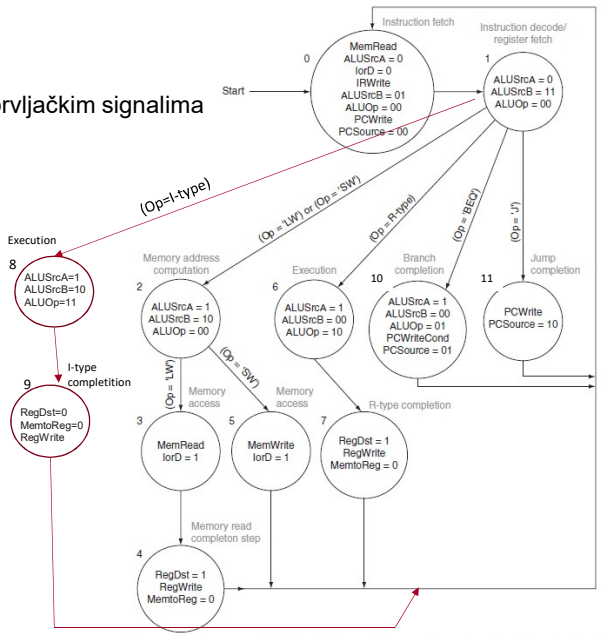
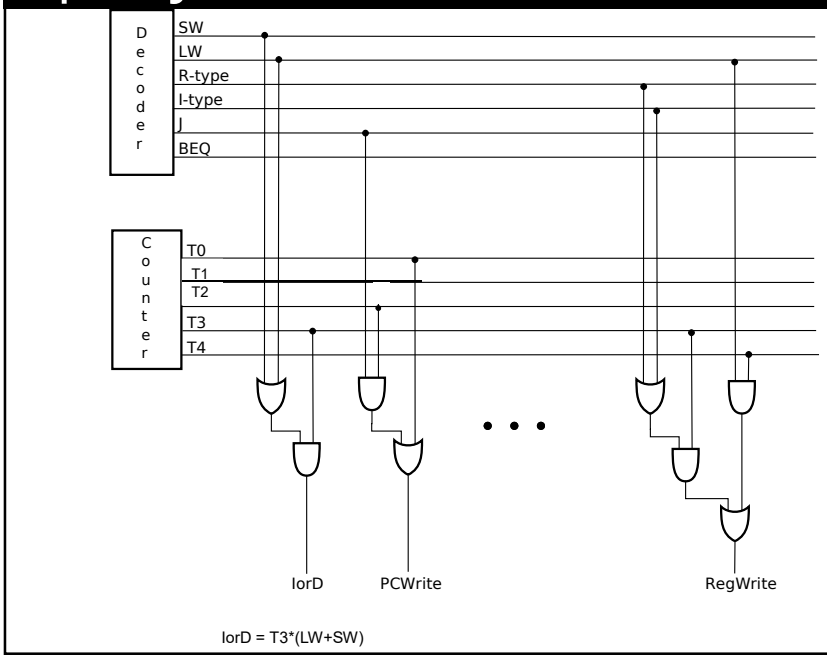


FIGURE 5.38 The complete finite state machine control for the datapath shown in Figure 5.28. The labels on the arcs are conditions that are tested to determine which state is the next state; when the next state is unconditional, no label is given. The labels inside the nodes indicate the output sig-

Logički izrazi za upravljačke signale

PCWriteCond [= S10]= T2*BEQ
PCWrite [= S0+S11]= T0+T2*J
IorD [= S3+S5] = T3*(LW+SW)
MemRead [= S0+S3]= T0+T3*LW
MemWrite [=S5] = T3*SW
MDRWrite [=S3]= T3*LW
IRWrite [=S0] = T0
RegDst [=S7]= T4*R-type
MemtoReg [=S4]= T4*LW
RegWrite [= S4+S7+S9] = T4*LW+T3*(R-type +I-type)
ALUSrcA [= S2+S6+S8+S10] = T2*(LW+SW+R-type+I-type+BEQ)
ALUSrcB1 [= S1+S2+S8] = T1 +T2*(SW+LW+I-type)
ALUSrcB0 [= S0+S1]= T0+T1
ALUOp1 [= S6+S8]= T2*(R-type+I-type)
ALUOp0 [= S8+S10] = T2*(I-type+BEQ)
PCSrc1 [= S11]= T2*J
PCSrc0 [= S10]= T2*BEQ

Upravljačka matrica



IorD = T3*(LW+SW)

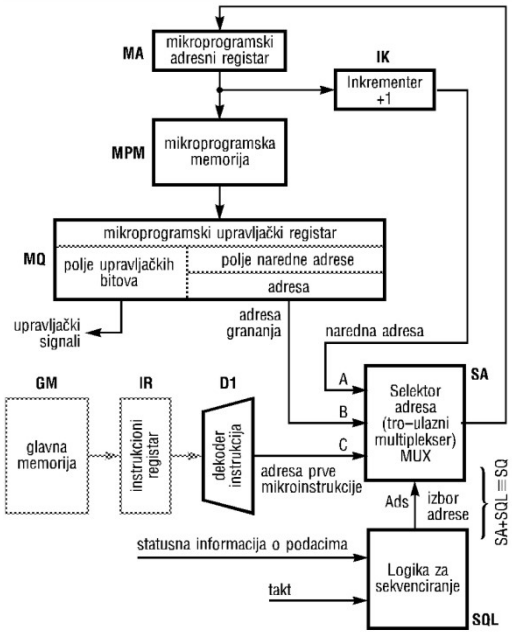
Osobine hardverske implementacije

- kompleksnost sekvenciranja i logike, ako je veliki skup instrukcija
- otežano projektovanje i testiranje
- nefleksibilan dizajn – teško je dodati nove instrukcije u već projektovanu CU
  - Kada je jedanom projektovana, male promene zahtevaju potpuni redizajn kola
- Glavna prednost je brzina rada i to je razlog što se ova tehnika koristi za realizaciju brzih procesora

Mikroprogramsko upravljanje

- Skup upravljačkih signala zajedno sa sekvencirajućom informacijom smešta se u memoriju
  - Memorija može biti ROM ili RAM tipa i naziva se mikroprogramska memorija (microprogramme memory ili control memory)
  - Mikroprogramsko upravaljanje se realizuje pomoću sekvence mikroinstrukcija
  - Za svaku instrukciju iz skupa instrukcija date arhitekture, postoji poseban mikro-program zapamćen u mikro-programskoj memoriji.
  - Svaka mikro-instrukcija definiše upravljačke signale koji se aktiviraju u datom trenutku.
    - Mikroprogram može imati linijsku strukturu, ali može imati i naredbe grananja i skoka.
    - Adresa sledeće mikro-instukcije se određuje logikom za sekvenciranje:
      - To može biti startna adresa mikro-programa, koja se dobija na osnovu koda operacije instrukcije
      - naredna instrukcija u nizu (adresa se dobija inkrementiranjem sadržaja mikroprogramskog adresnog registra)
      - ili adresa iz polja naredne adrese (adresa grananja)

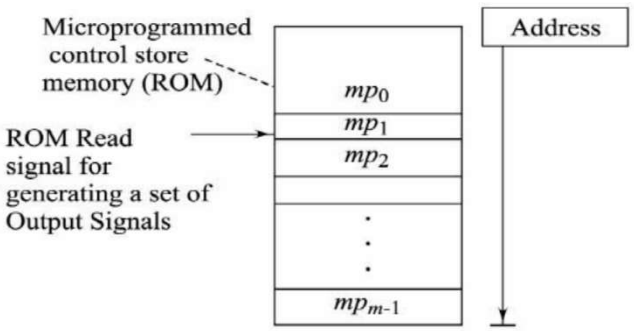
Mikroprogramsko upravljanje



Organizacija mikroprogramske memorije

- Ako imamo m instrukcija u ISA, I_0 do I_{m-1}, onda postoji m različitih mikro-programa, mp_0 do mp_{m-1}, zapamćenih u memoriji
  - Svaki mikro-program ima različitu adresu u mikroprogramskoj memoriji.
  - Da bi se izvršila instrukcija, mikroprogramski upravljani procesor pristupa ovoj memoriji da bi pronašao skup mikroinstrukcija potrebnih za implementaciju instrukcije I_i
  - startna adresa se dobija na osnovu koda operacije pribavljene instrukcije (nakon okončanja faze dekodiranja instrukcije).

Izgled mikroprogramske memorije: za m instrukcija, postoji m mikroprograma, pri čemu je svaki zapamćen na različitoj adresi u mikroprogramskoj memoriji



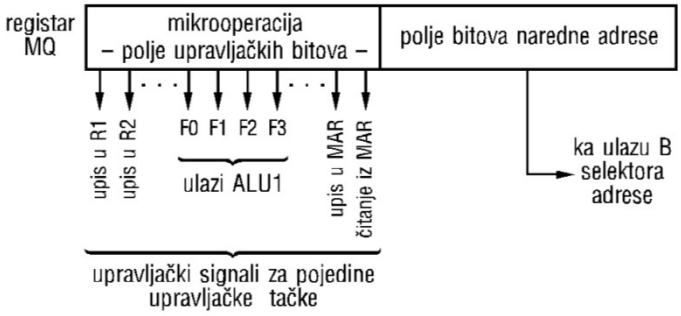
Format mikroinstrukcija

- polje upravljačkih signala
- polje adrese sledeće mikroinstrukcije

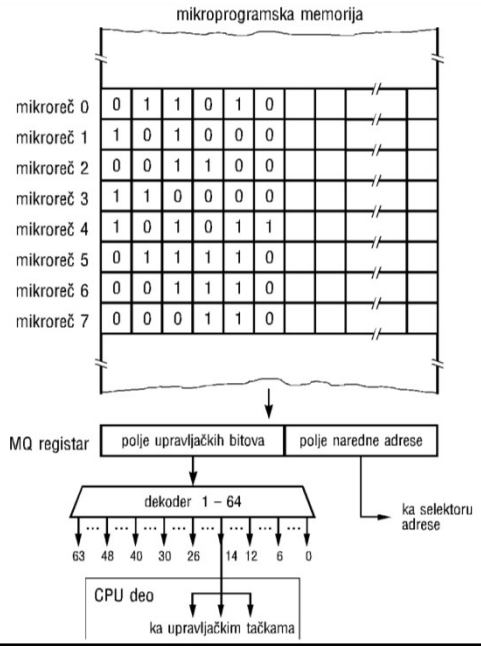
✱ Horizontalni format
- koristi se po jedna binarna pozicija za svaki upravljački signal
  - mogu biti veoma duge instrukcije, tipično 150-200 bitova
  - visok stepen paralelizma u radu (jednom instrukcijom se može definisati ALU operacija, prenos ka/iz memorije, prenos između registara)
- upravljački signali sa izlaza registra mikroinstrukcija direktno se vode do upravljačkih tačaka u stazi podataka procesora

✱ Vertikalni format (ili maksimalno kodirani format)
- mikroinstrukcijom se kodira jedna operacija (akcija)
  - npr prenos iz registra u registar (R1←R2)
  - obavlja operacija (R1←R2+R3)
- neophodno je definisati skup svih mikrooperacija i svaku kodirati
  - sa n bitova može se kodirati 2^n mikroisntrukcija
- mnogo su kraće instrukcije (12 do 32 bita)
- zahteva se korišćenje dekodera
- manje fleksibilno
  - dodavanje nove instrukcije zahteva promenu dekodera

Horizontalni format



vertikalni format



Mikroprogramsko upravljanje - osobine

- prednosti
  - Izmene se izvode brže
  - Greške se lakše otklanjaju
  - Projektovanje i održavanje računara je pojednostavljeno
- Nedostatak
  - sporije izvršenje instrukcija

Poređenje hardverskog i mikroprogramskog upravljanja

atribut	Hardversko upravljanje	Mikroprogramsko upravljanje
brzina	brzo	sporo
Cena implementacije	skuplje	jeftinije
fleksibilnost	Nefleksibilan dizajn; izmena ili dodavanje nove instrukcije zahteva kompletan re-dizajn	fleksibilnije; nove instrukcije se lakše mogu dodati
dekodiranje	složeno	lako
Primena	RISC procesori	CISC procesori
Veličina skupa instrukcija	mala	velika
Upravljačka memorija	Ne postoji	postoji
Zahtevana površina oblasti na čipu	manja	veća
Mogućnost upravljanja kompleksnim instrukcijama	teško	lakše