Pojednostavnjeni modeli CISC I RISC procesora

S. Ribarić, Građa računala /poglavlje 3. str. 59 – 92/
S. Ribarić, Zbirka riješenih zadataka iz GR /poglavlje 3. str. 33 – 70/

- Definicija CISC / RISC
- Komponente modela CISC
- Primjer izvođenja programa
- Stanje registara
- Stanje na sabirnicama
- Komponente modela (S)RISC
- ISA-model procesora

CISC i RISC

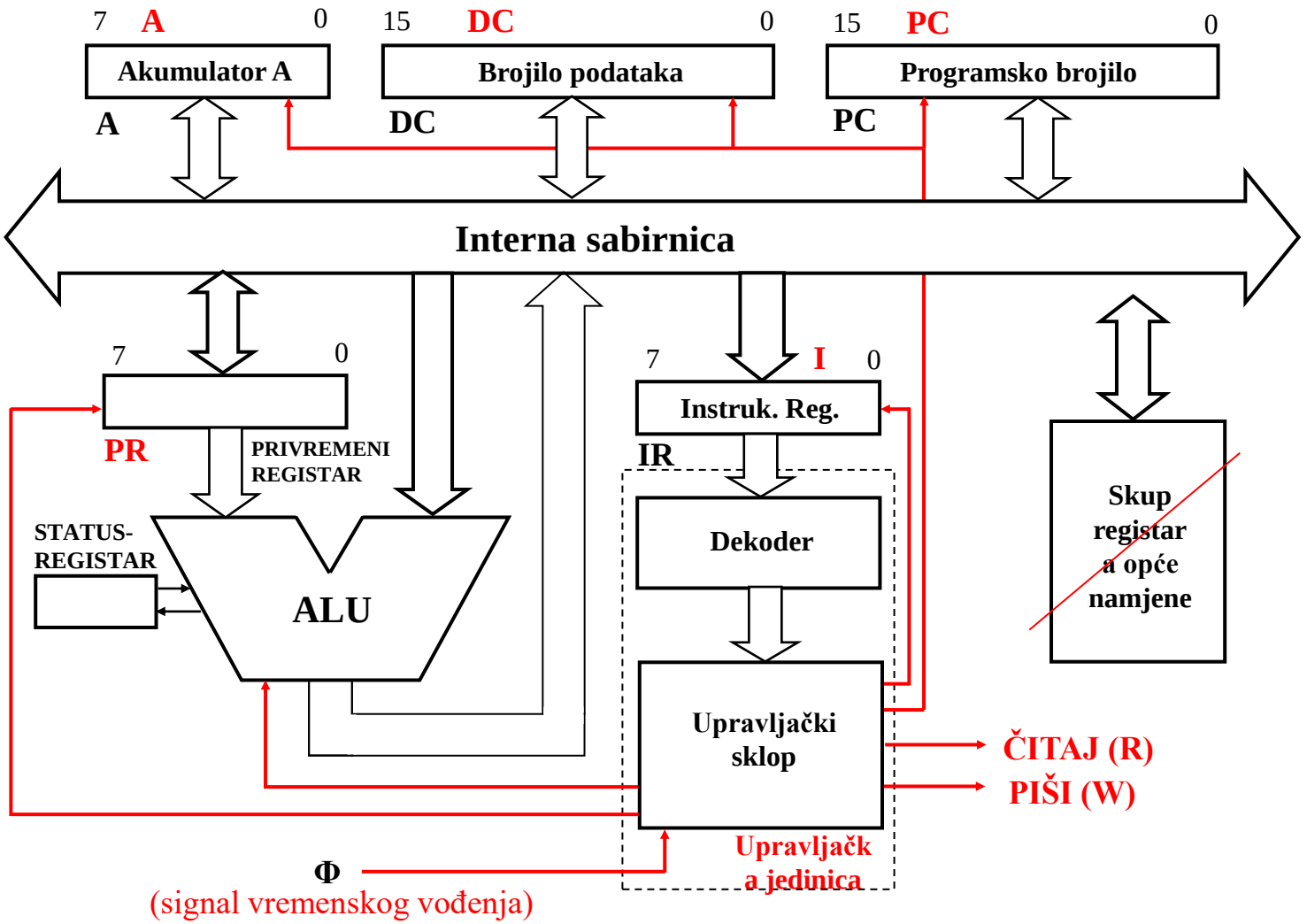
CISC = Complex Instruction Set Architecture

- Velik i složen skup instrukcija
- Velik i „raskošan” skup načina adresiranja
- Instrukcije promjenjive duljine
- Mali broj registara opće namjene
- Memorijski operandi

RISC = Reduced instruction Set Architecture

- Manji skup instrukcija
- Manji skup načina adresiranja
- Instrukcije fiksne duljie
- Veći broj registara opće namjene
- Registarski operandi, LOAD/STORE arhitektura

Model (mikro)procesora CISC arhitekture

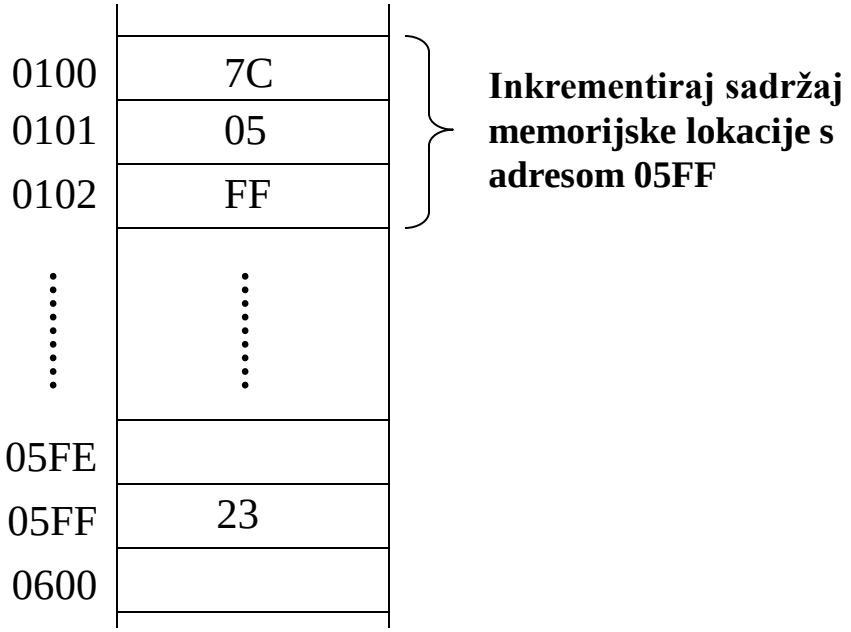


Komponente modela

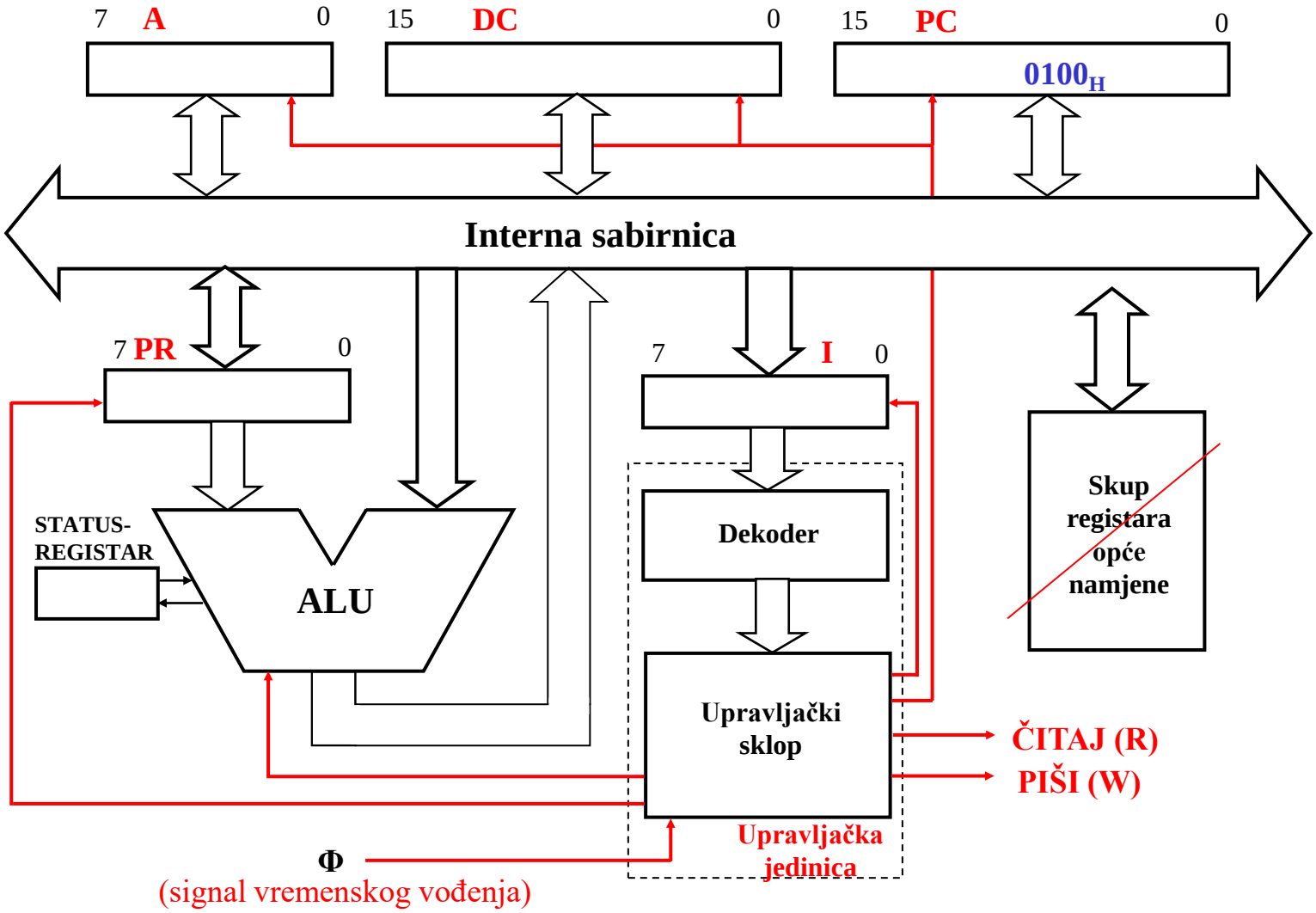
- Akumulator A
- Programsko brojilo PC
- Instrukcijski registar IR
- Brojilo podataka DC
- Privremeni registar PR
- Status-registar (Registar stanja)
- ALU
- (Skup registara opće namjene)
- Interna sabirnica
- Upravljačka jedinica

Primjer programa:

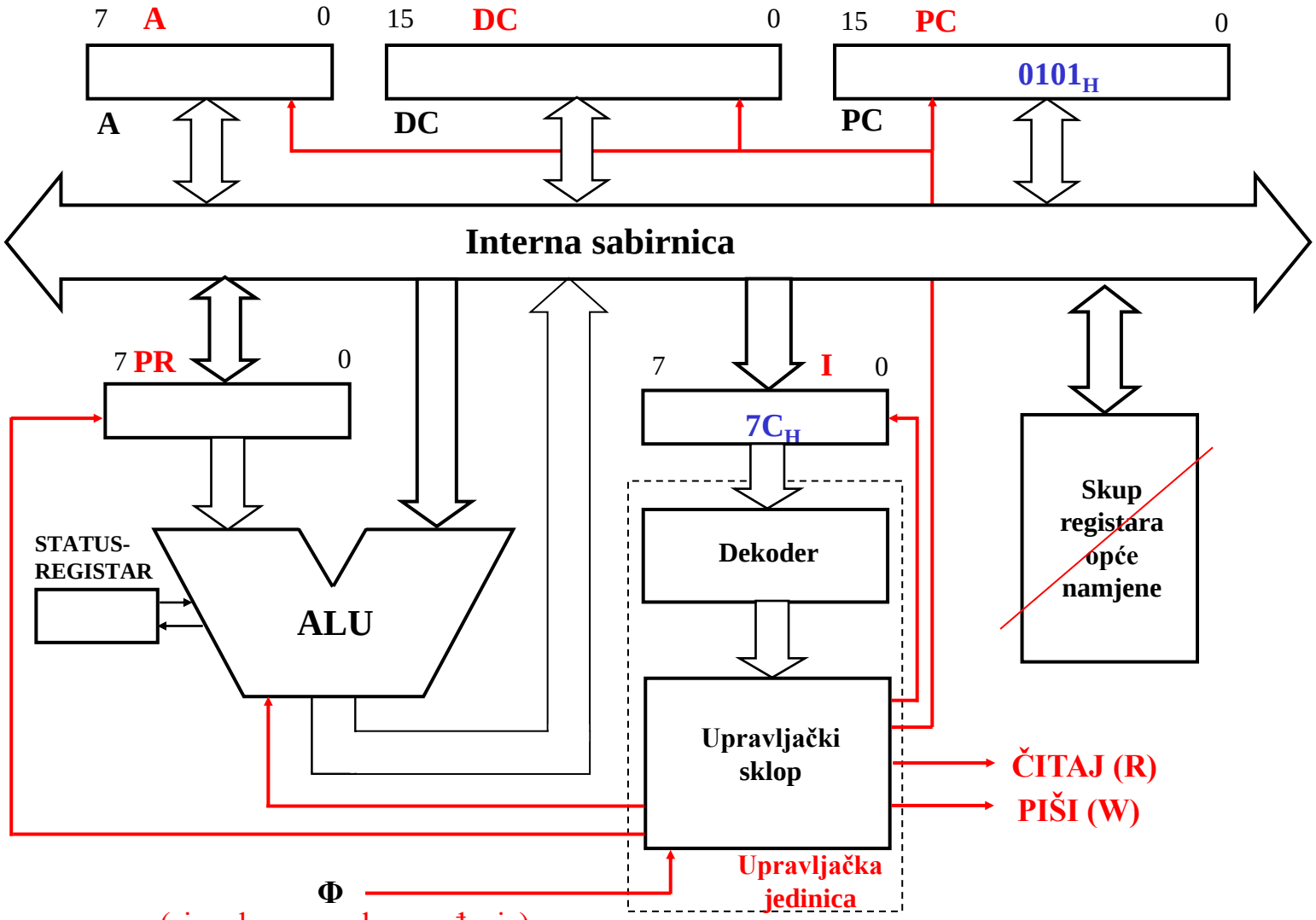
- Program samo od jedne instrukcije INC $05FF

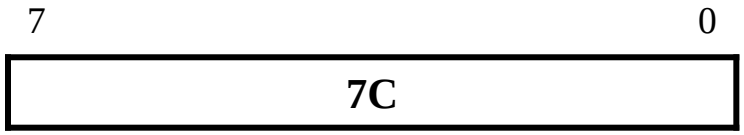


Početni uvjeti



Stanje nakon pribavljanja operacijskog koda instrukcije





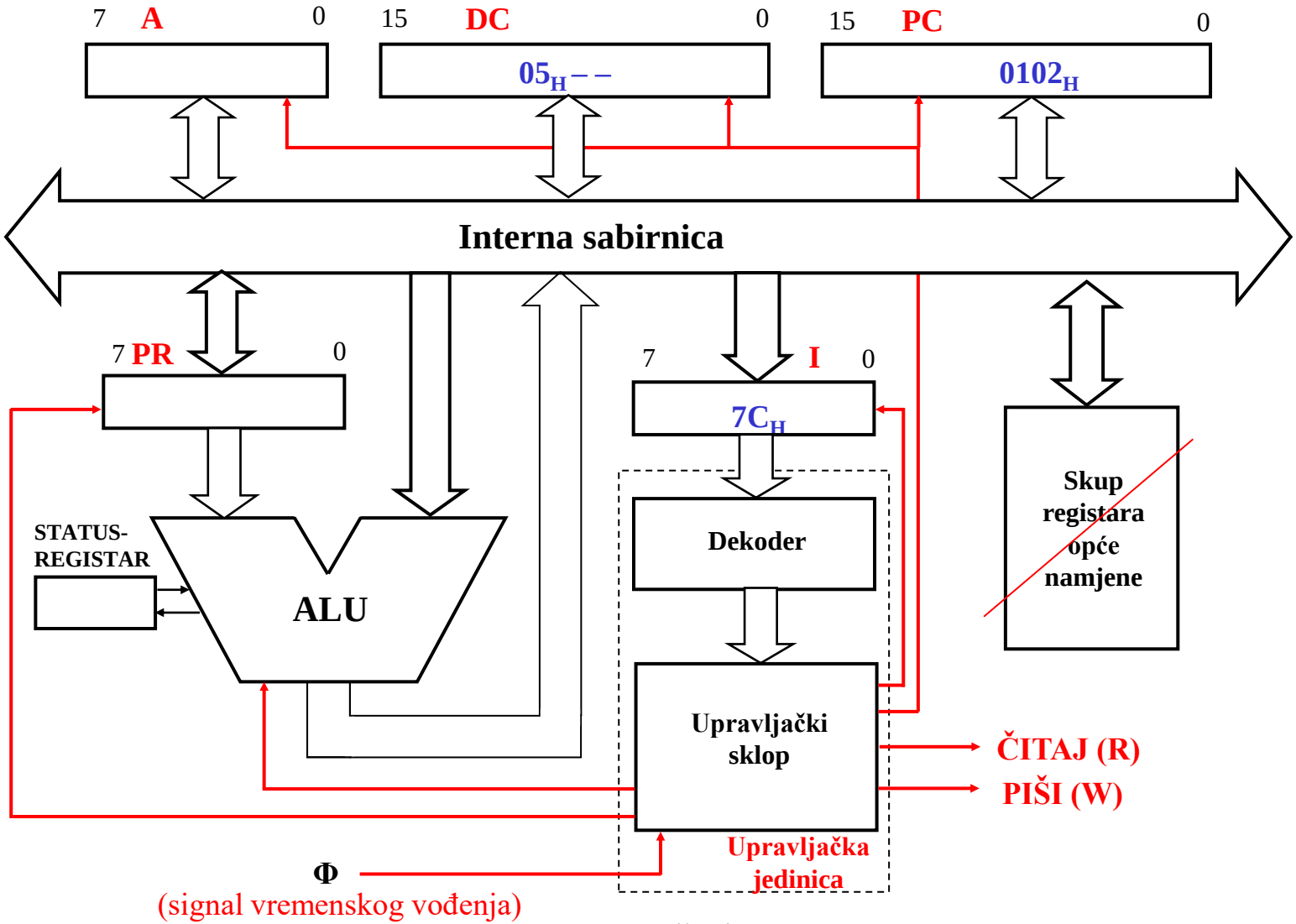
Operacijski kod

7C = 01111100

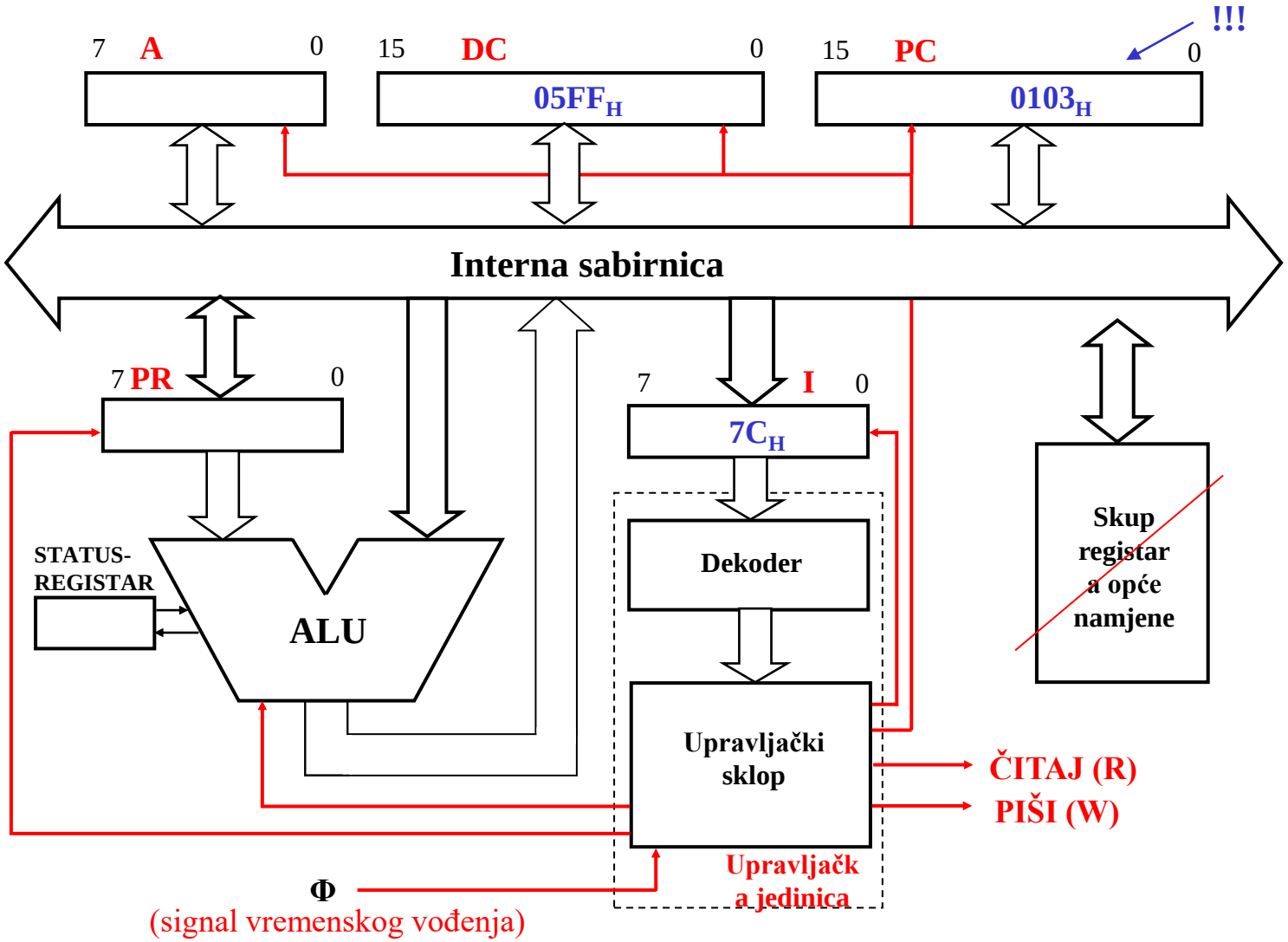
se tumači kao: Povećaj za 1 vrijednost operanda čija je adresa sadržana u dva bajta koja slijede ovom operacijskom kodu.

Stanje nakon pribavljanja značajnijeg bajta adrese operanda

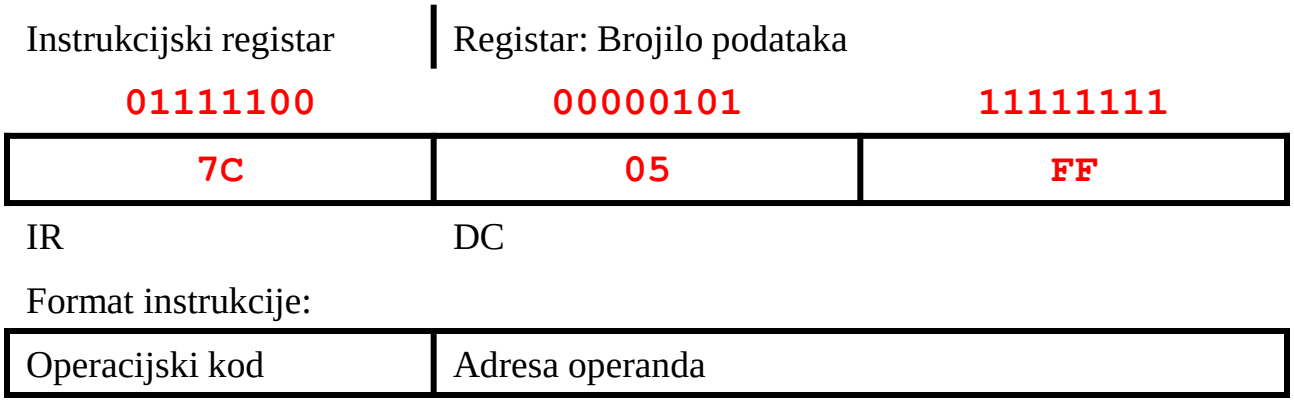
Pazi: Faza Pribavi još uvijek traje!



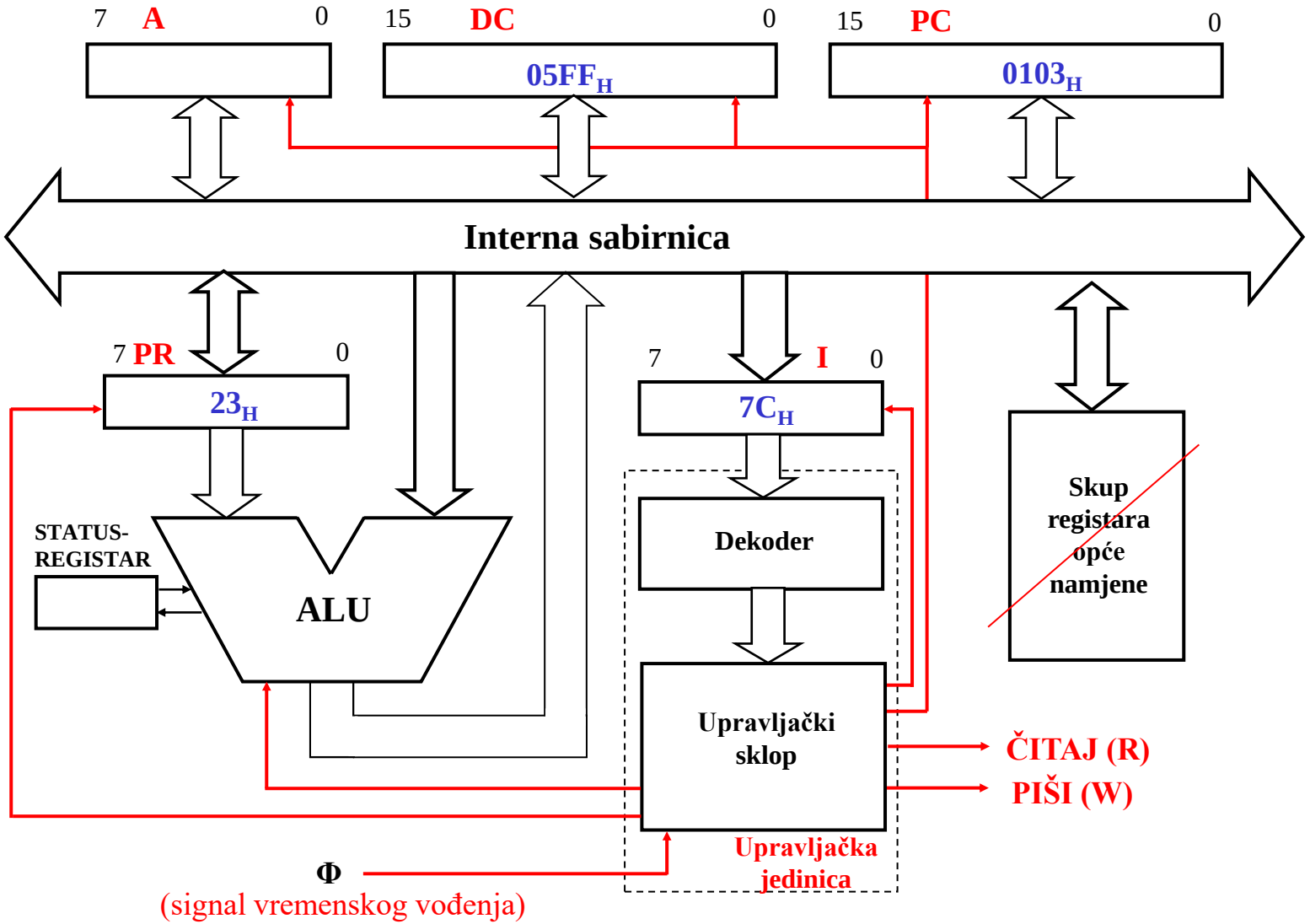
Stanje nakon pribavljanja manje značajnijeg bajta adrese operanda



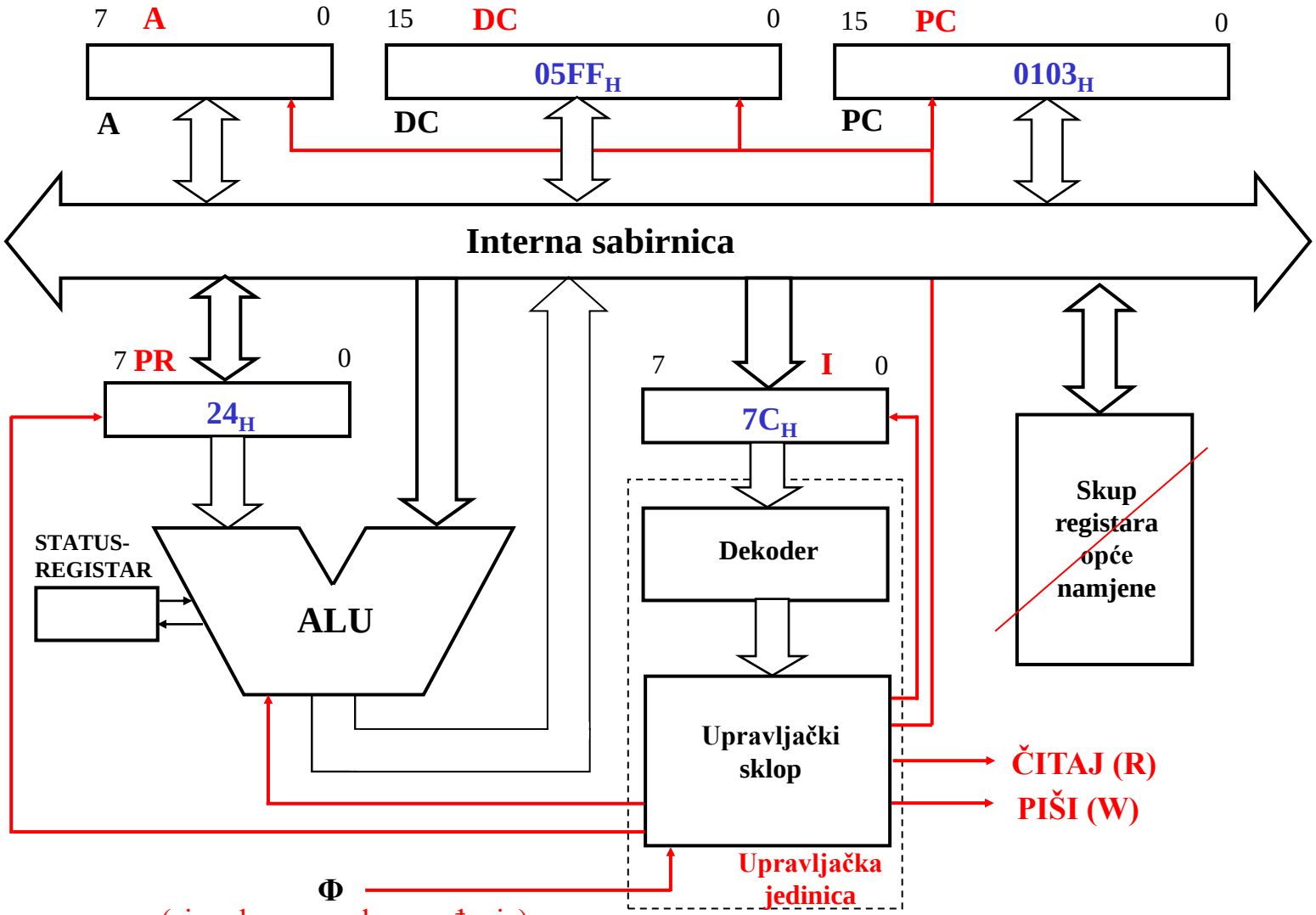
Faza Pribavi je završena!



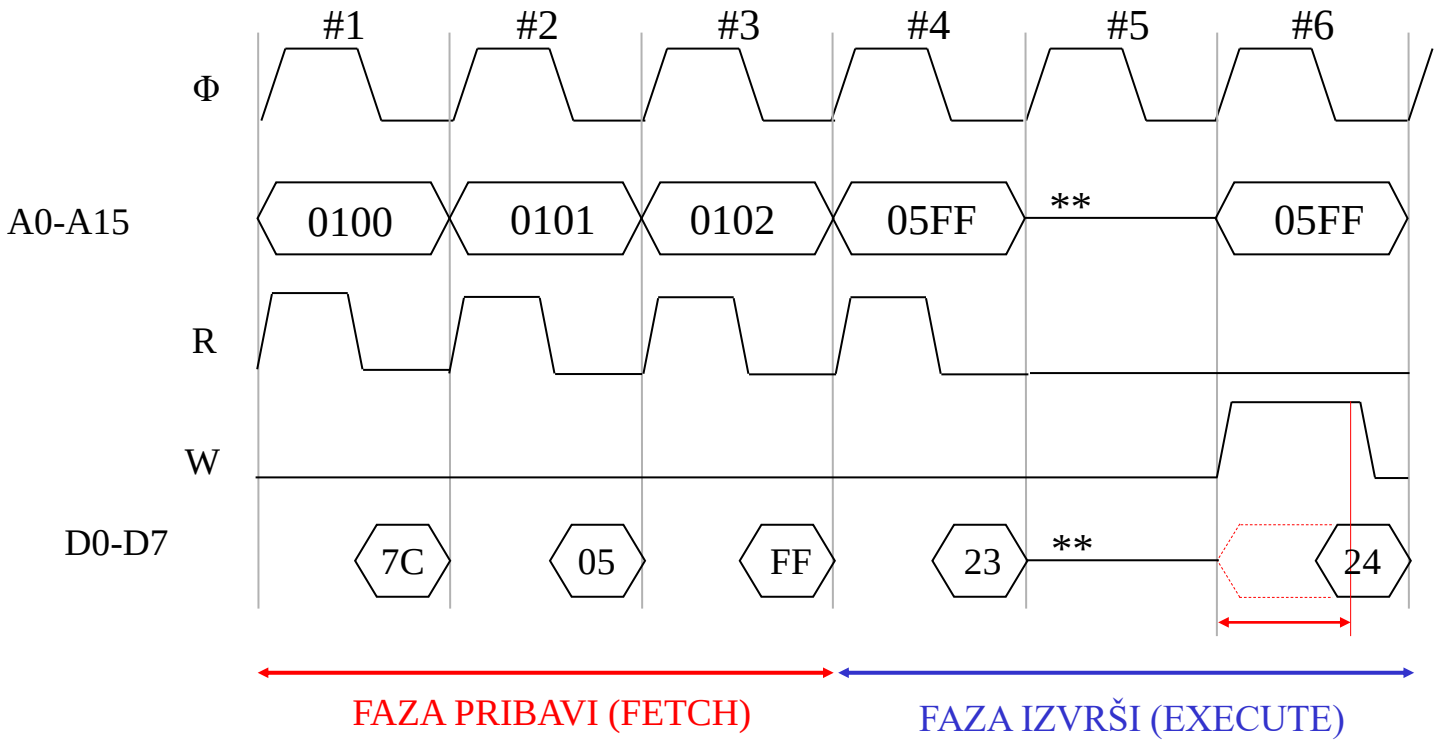
Stanje nakon dohvata operanda (faza IZVRŠI)



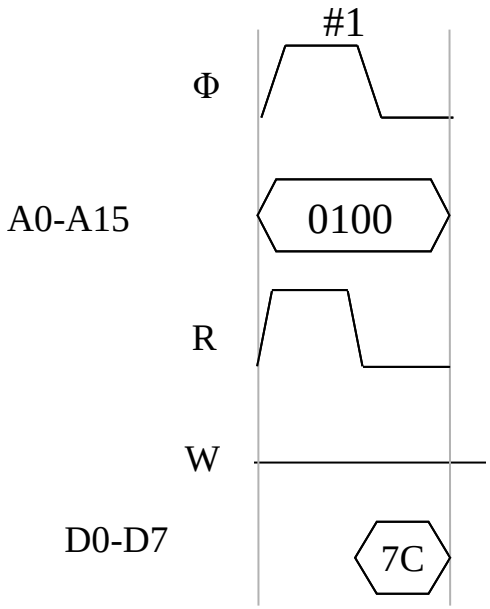
Stanje nakon povećanja operanda za jedan (faza IZVRŠI)



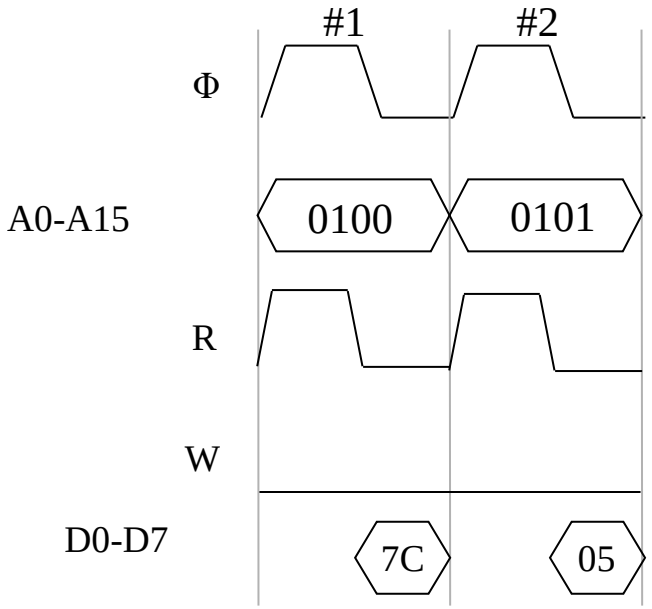
Stanje na vanjskim sabirnicama



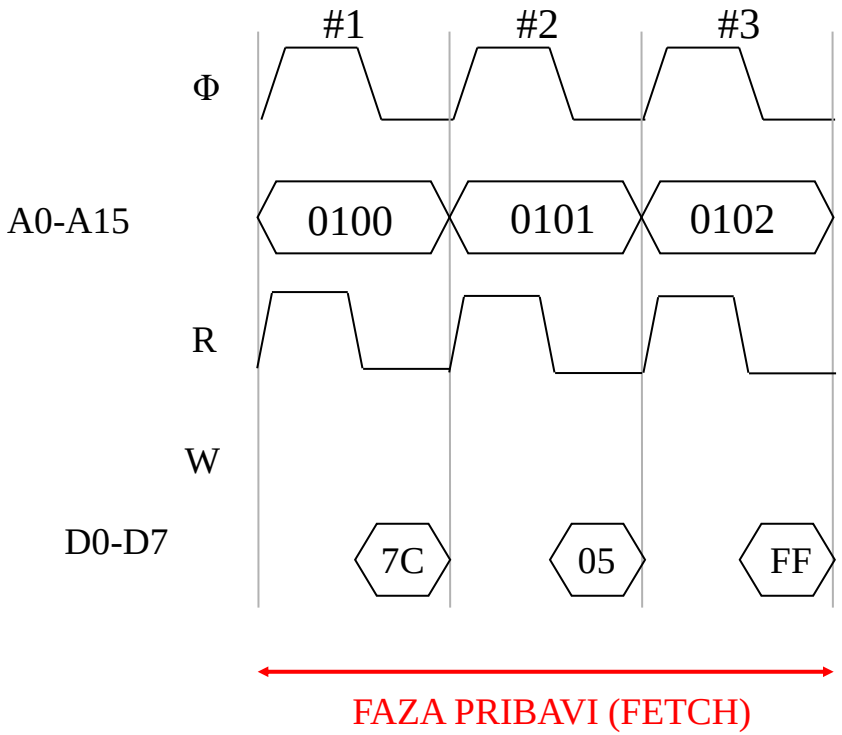
Prva perioda signala vremenskog vođenja



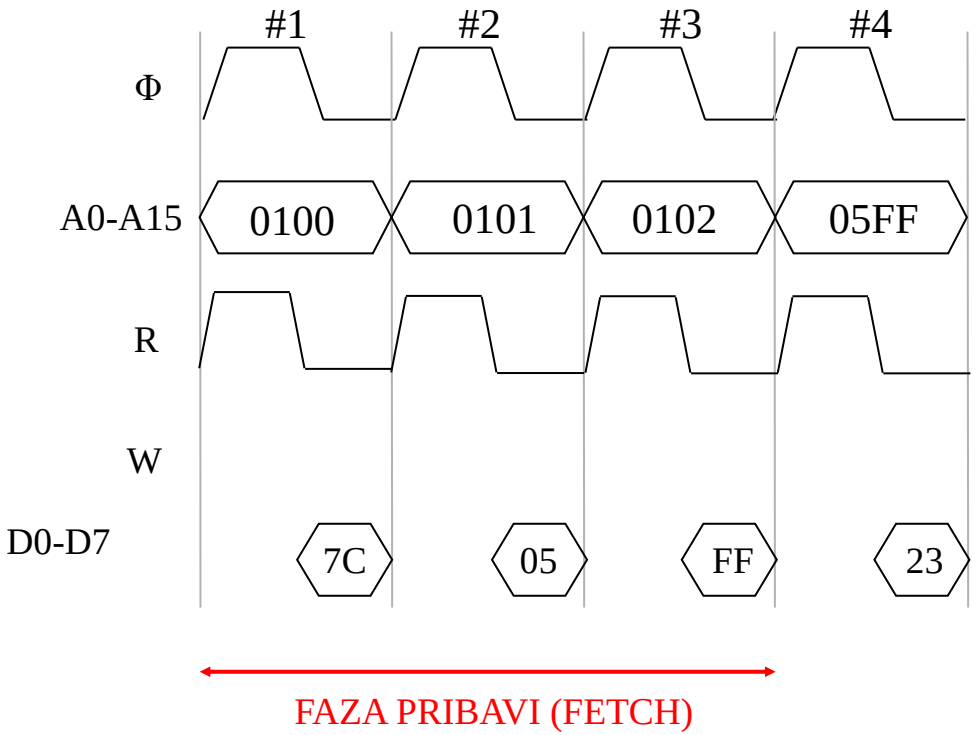
Druga perioda signala vremenskog vođenja



Treća perioda signala vremenskog vođenja



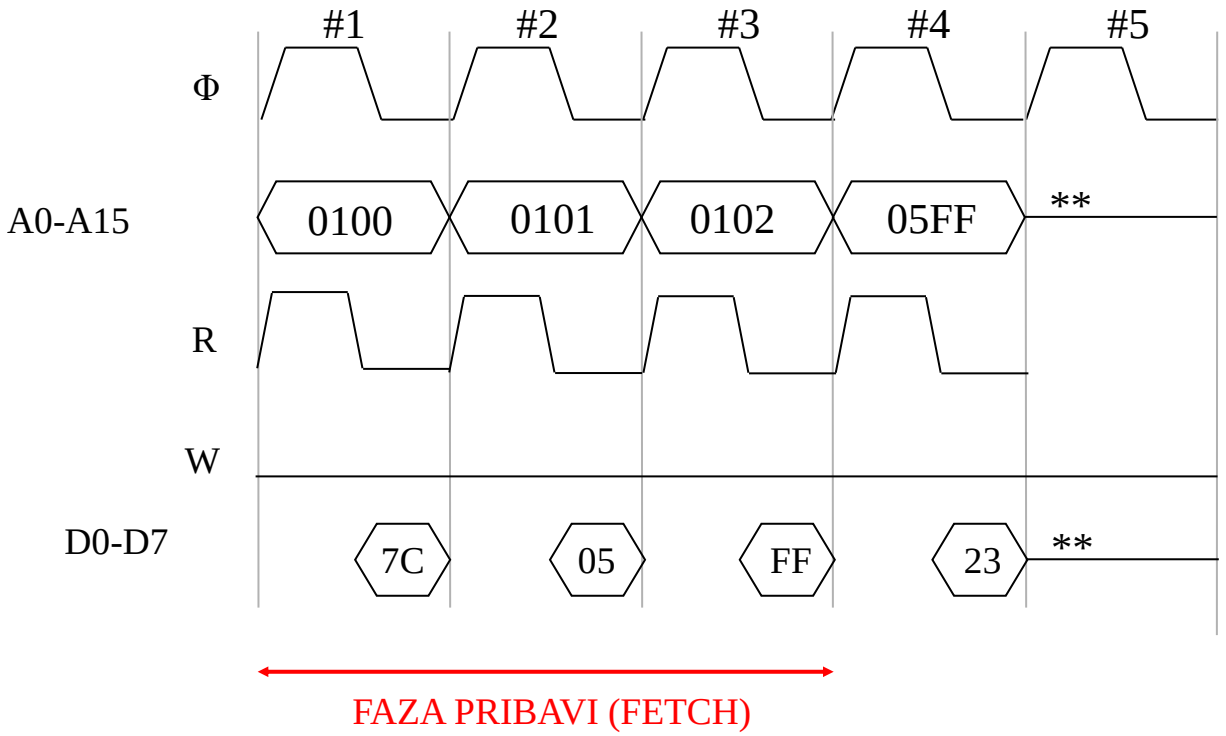
Četvrta perioda signala vremenskog vođenja



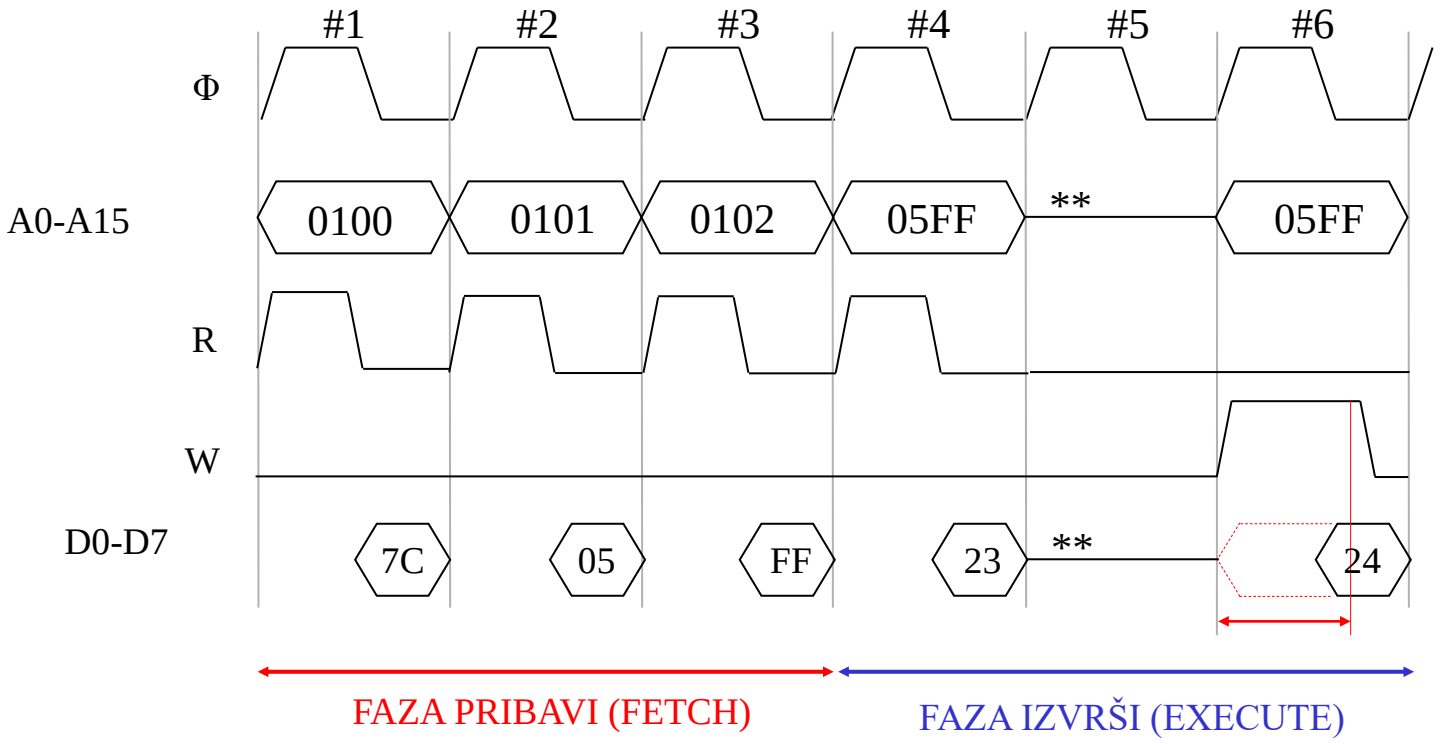
Peta perioda signala vremenskog vođenja

Pozor:

** - Označava stanje visoke impedancije /treće stanje/



Šesta perioda vremenskog vođenja



Motorola MC 6800

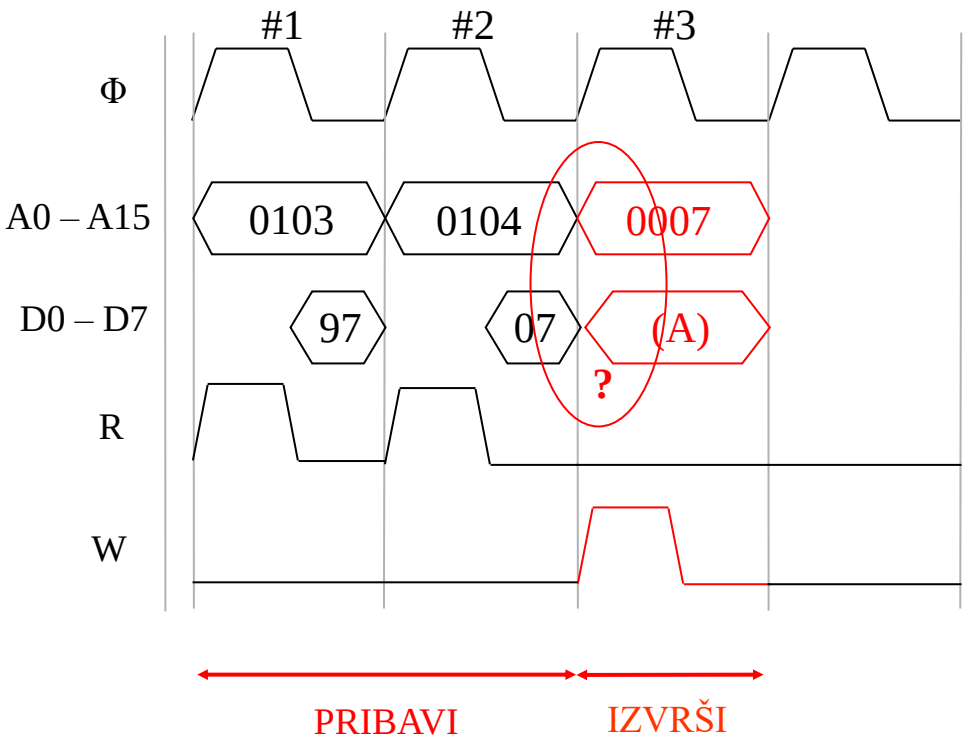
(izravni prošireni način adresiranja)

	OP	~	#
INC	7C	6	3

Zahtijeva 6 periode signala vremenskog vođenja!

Naš model obavlja ovu instrukciju također za 6 perioda!

Stanje na sabirnicama za instrukciju STA $07



Motorola MC 6800

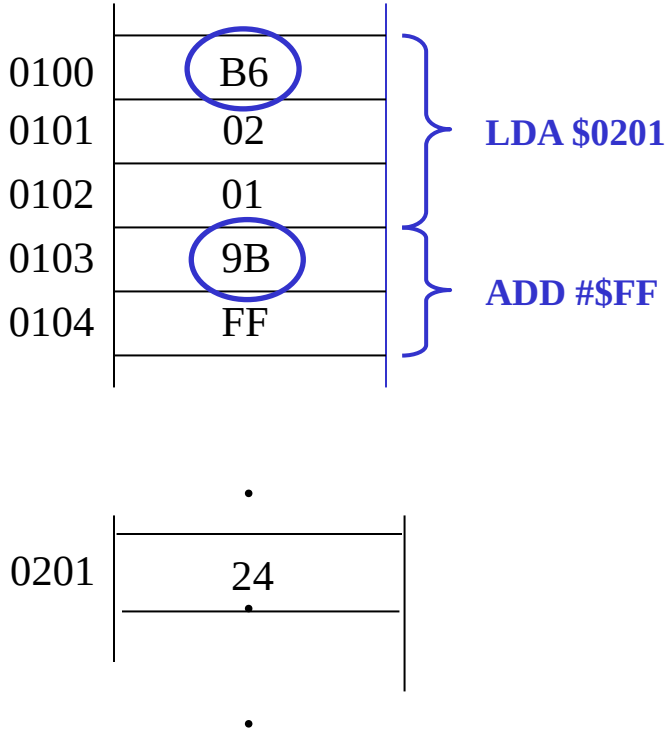
(izravni način adresiranja)

	OP	~	#
STA A	97	4	2

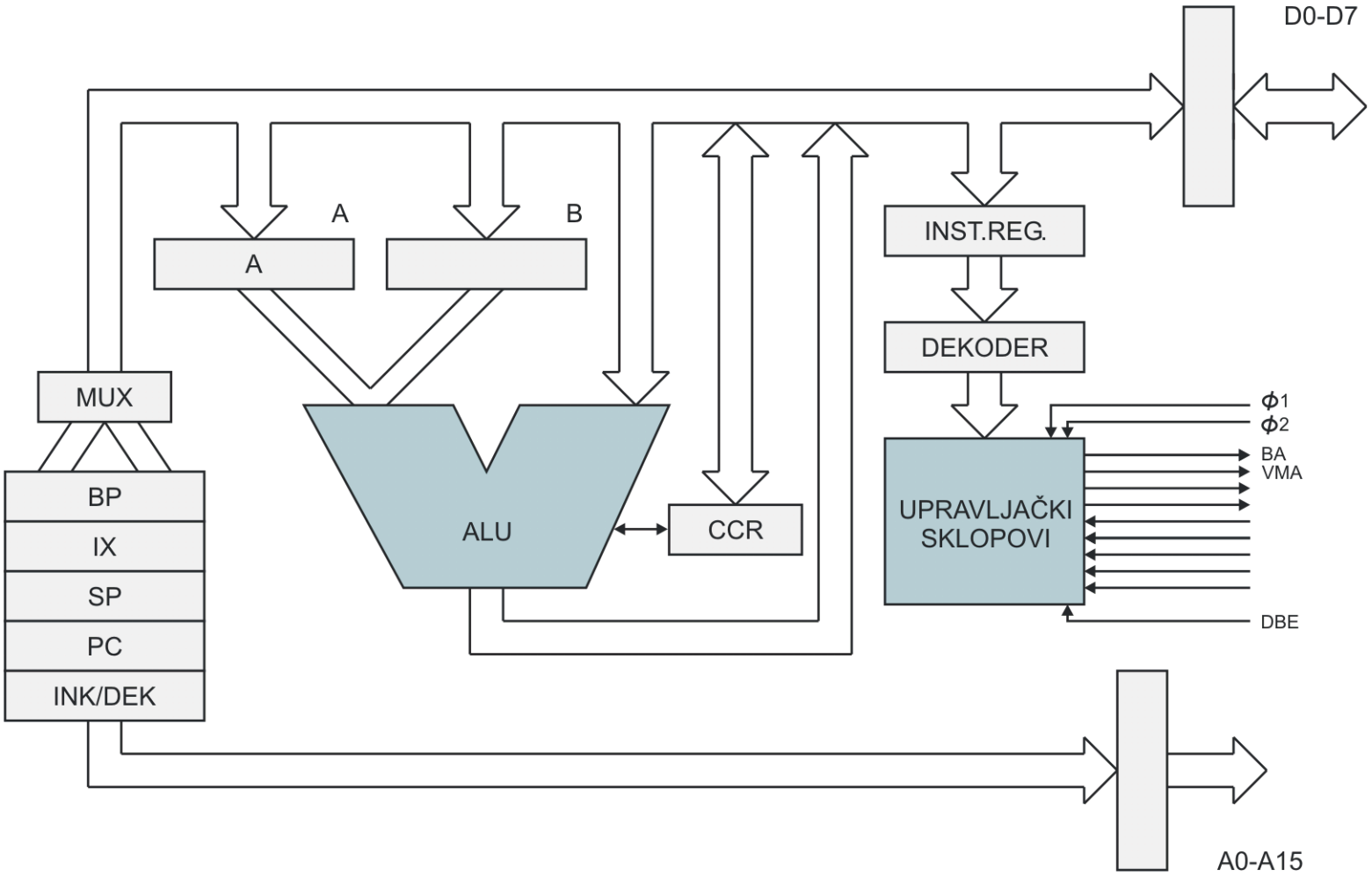
Zahtijeva 4 periode signala vremenskog vođenja!

Naš model obavlja ovu instrukciju tijekom 3 periode?!

Primjer:



Primjer: MC 6800



Status-registar (Registar stanja SR; Registar uvjeta)

- SR je povezan s ALU i sastoji se od bistabila (dojavni bistabili; zastavice)

Prema funkciji zastavice se mogu razvrstati na:

i) zastavice za indikaciju stanja koja su nastala kao posljedica izvođenja neke aritmetičke ili logičke operacije

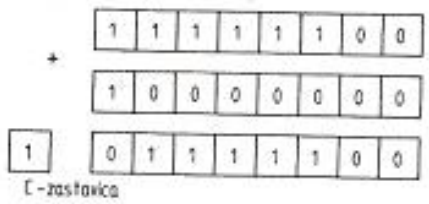
ii) zastavice za indikaciju glavnih stanja procesora

iii) zastavice za rukovanje prekidom i označavanje prekidnih razina

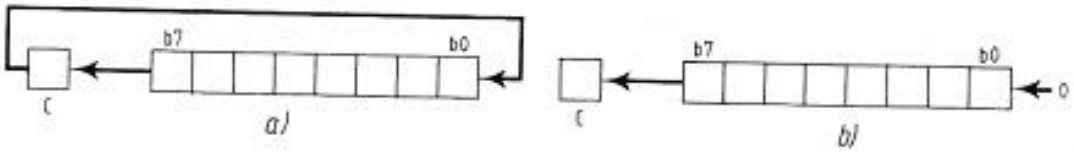
Zastavice iz skupine i) automatski se postavljaju (-> 1) ili brišu (1 -> 0) ovisno o rezultatu aritmetičkih i logičkih operacija

Zastavice iz te skupine se promatraju kao pojedinačni bistabili i imaju oznake npr. C, N, Z , V, H

Zastavice C (Carry) –Zastavica prijenosa pokazuje da se dogodio prijenos iz najznačajnijeg bita rezultata. Na primjer:



Slika 4.4. Primjer zbrajanja sa prijenosom



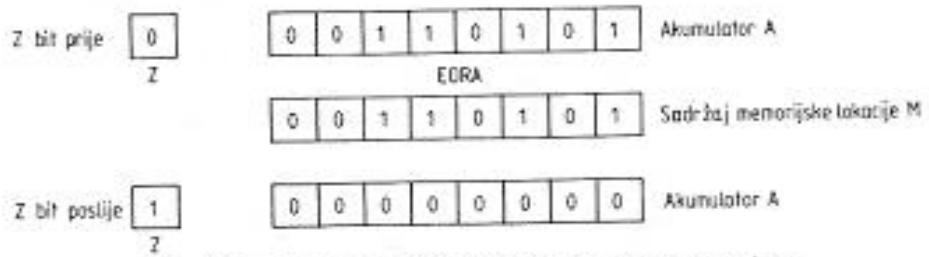
Slika 4.5. Primjer instrukcija: a) kružnog posmaka, b) posmaka ulijevo

S. Ribarić, Arhitektura mikroprocesora, Tehnička knjiga, Zagreb, 1982.

Zastavica N (Negative) upotrebljava se za indikaciju negativnog rezultata aritmetičke ili logičke operacije.

(negativni rezultat ima najznačajniji bit jednak 1)

Zastavica Z (Zero) / zastavica nule/ dojavni bistabil koji se postavlja u 1 (Z-> 1). Npr.



Slika 4.6. Operacije EORA i Z-bit u registru uvjeta

Zastavica V (Overflow) je dojavni bistabil koji označava preliv. U matematičkom smislu pokazuje na grešku u bitu predznaka u aritmetici dvojnog (ili potpunog komplementa):

$$V = C_s \text{ „isključivo ili” } C_p$$

gdje je C_s prijenos u bit predznaka a C_p prijenos iz bita predznaka.

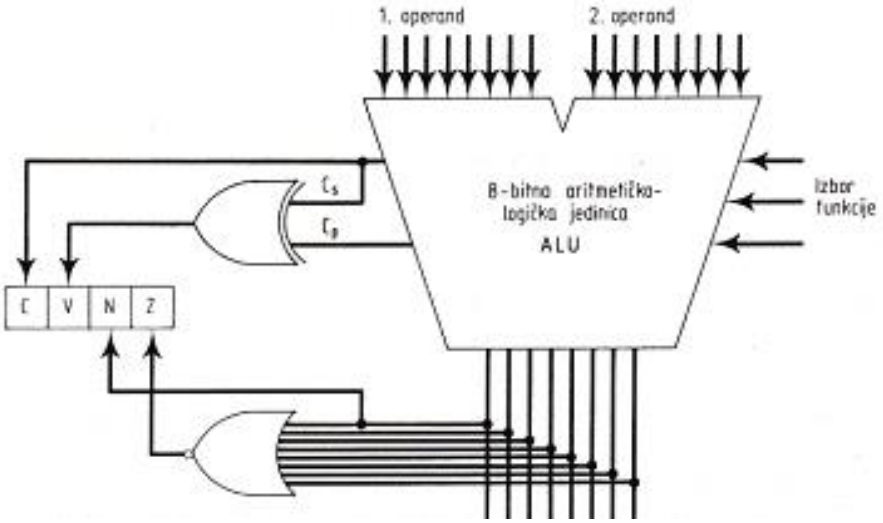
Ako pretpostavimo da je duljina registra (akumulatora) 8 bita, zastavica V će se postaviti u 1 ako je prilikom aritmetičke operacije rezultat prekoračio granicu +/- 127.

Npr. prilikom zbrajanja brojeva -100 i -32

	1	0	0	1	1	1	0	0	(−100)
	1	1	1	0	0	0	0	0	+(−32)
V	1	0	1	1	1	1	0	0	R = (+124)!!!

zastavica

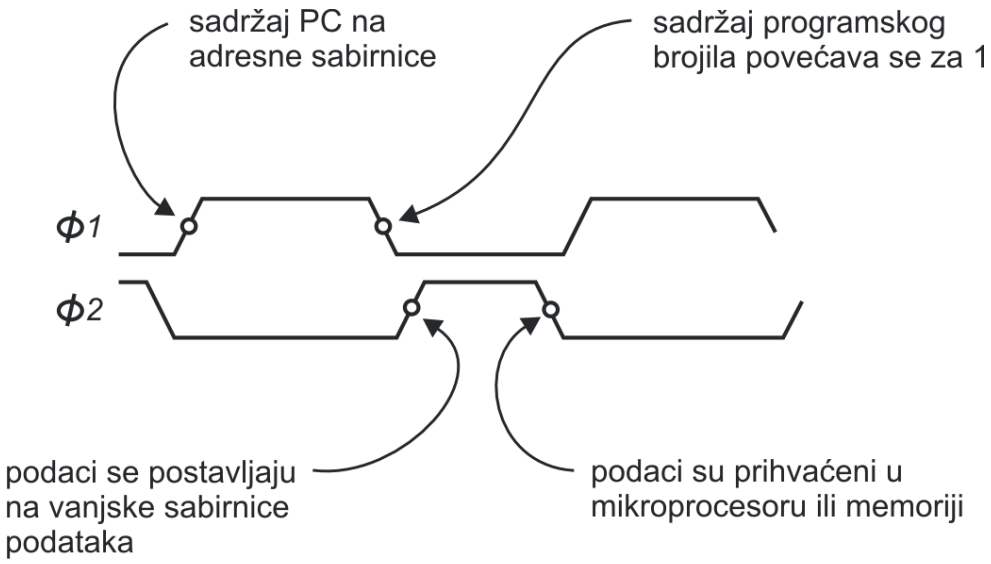
S. Ribarić, Naprednije arhitekture mikroprocesora, Školska knjiga, Zagreb, 1990.



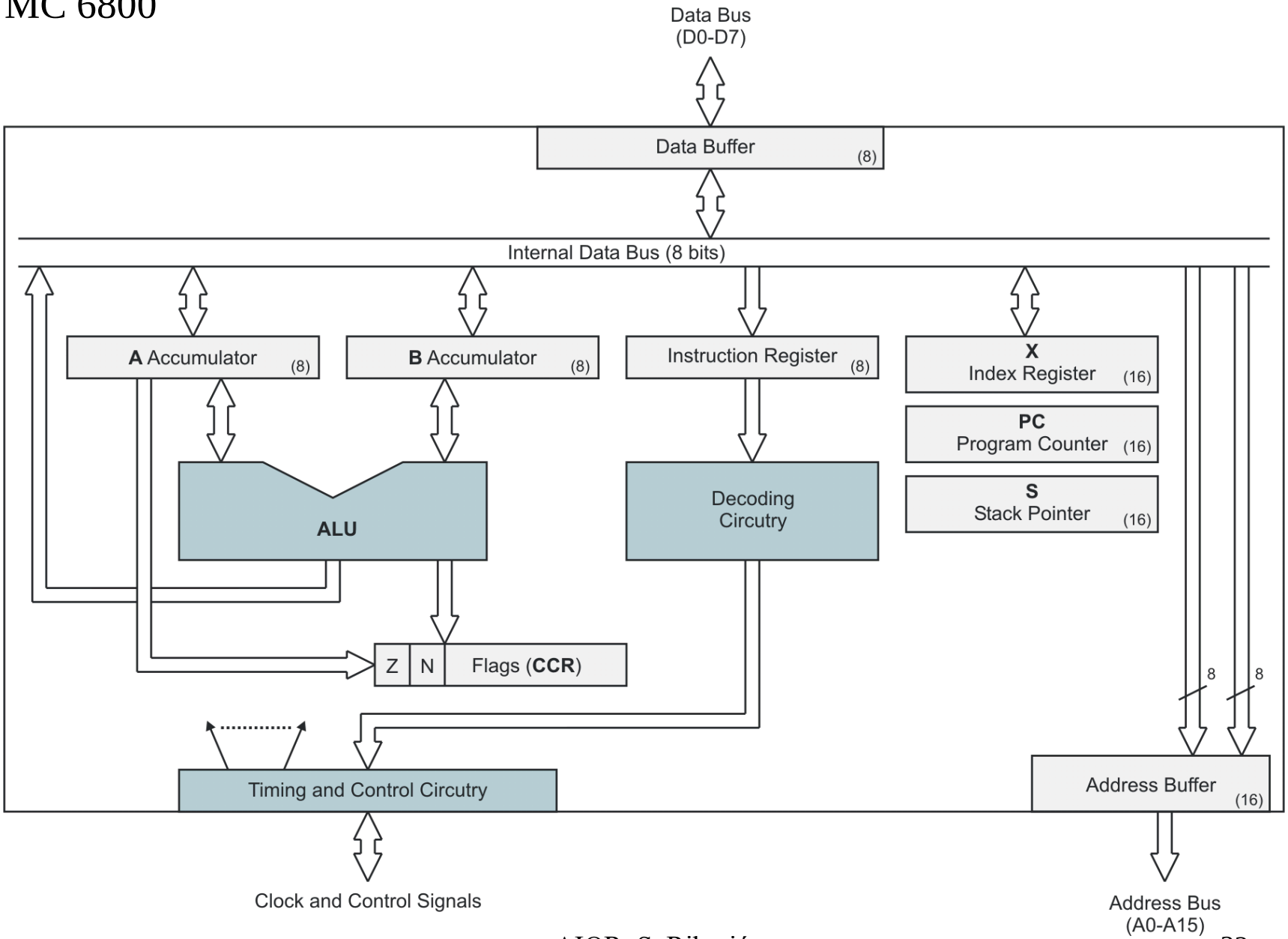
Instrukcije	1	1	H	I	N	Z	V	C	CCR
STAA			•	•	↕	↕	R	•	
EORA			•	•	↕	↕	R	•	
ADDA			↕	•	↕	↕	↕	↕	
CLRA			•	•	R	S	R	R	
COMA			•	•	↕	↕	R	S	
PSHA			•	•	•	•	•	•	
LDAA			•	•	↕	↕	R	•	

• ne utječe
S postavlja
R briše
↕ postavlja ili briše u ovisnosti od rezultata

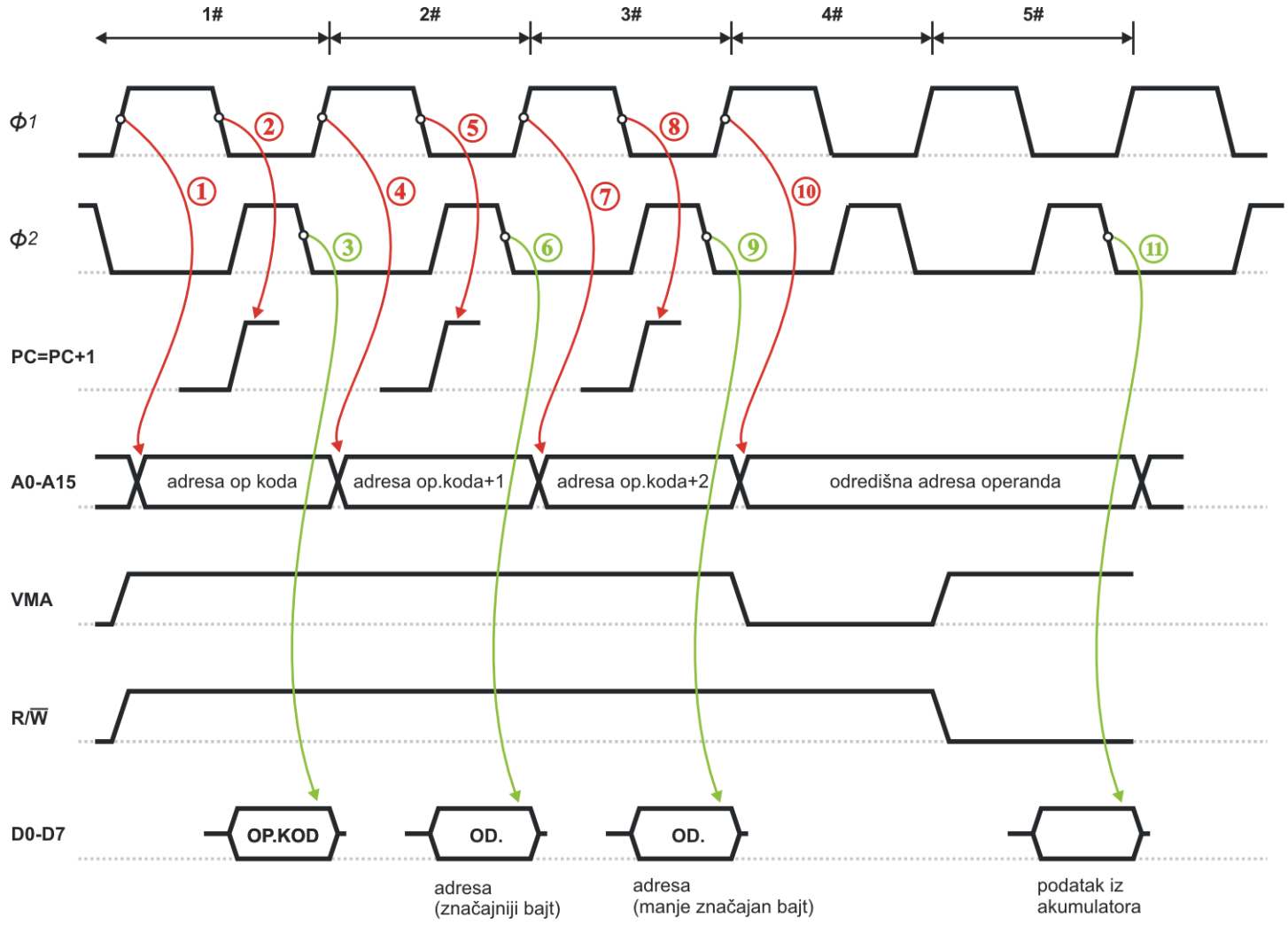
Signali vremenskog vođenja Φ 1 i Φ2



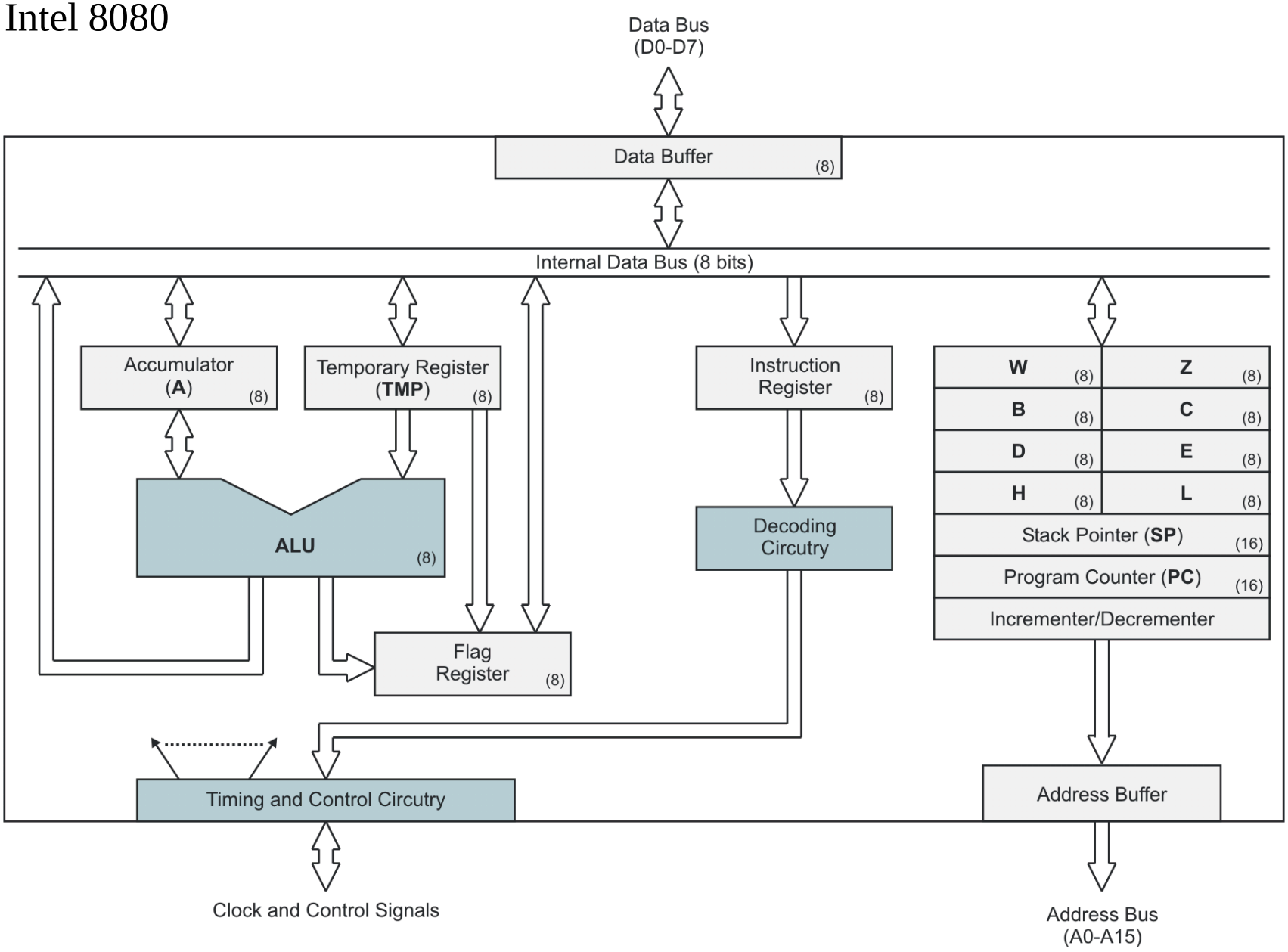
MC 6800

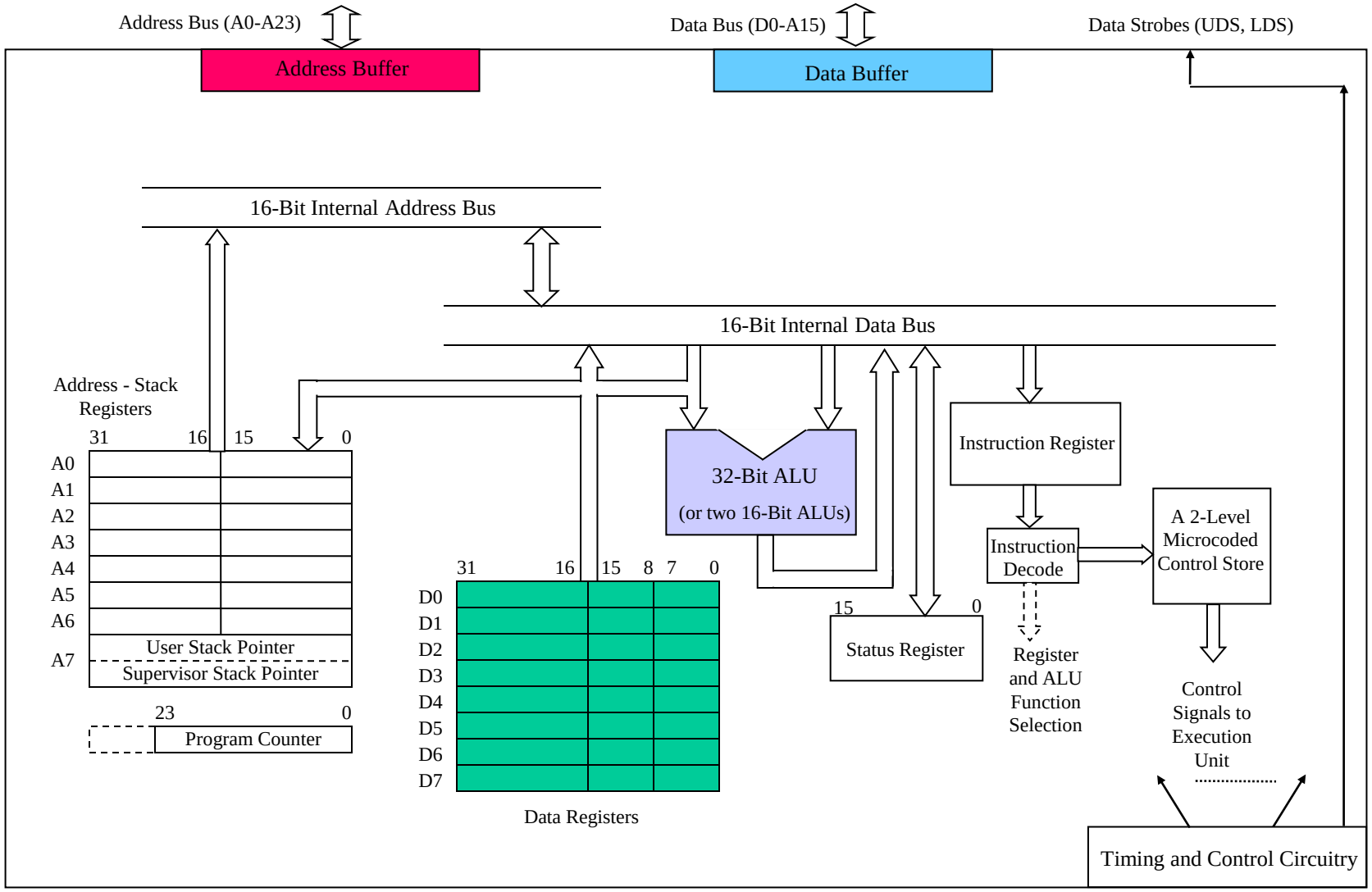

Data Bus
(D0-D7)
Data Buffer
(8)
Internal Data Bus (8 bits)
A Accumulator
(8)
B Accumulator
(8)
Instruction Register
(8)
X
Index Register
(16)
PC
Program Counter
(16)
S
Stack Pointer
(16)
ALU
Decoding
Circutry
Z
N
Flags (CCR)
8
8
Timing and Control Circutry
Address Buffer
(16)
Clock and Control Signals
Address Bus
(A0-A15)

Stanje na sabirnicama za STA A $010F (MC 6800)



Intel 8080


Data Bus
(D0-D7)
Data Buffer
(8)
Internal Data Bus (8 bits)
Accumulator
(A)
(8)
Temporary Register
(TMP)
(8)
Instruction
Register
(8)
W
(8)
Z
(8)
B
(8)
C
(8)
D
(8)
E
(8)
H
(8)
L
(8)
ALU
(8)
Decoding
Circuitry
Stack Pointer (SP)
(16)
Program Counter (PC)
(16)
Incrementer/Decrementer
Flag
Register
(8)
Timing and Control Circuitry
Address Buffer
Clock and Control Signals
Address Bus
(A0-A15)


Address Bus (A0-A23)
Data Bus (D0-A15)
Data Strobes (UDS, LDS)
Address Buffer
Data Buffer
16-Bit Internal Address Bus
16-Bit Internal Data Bus
Address - Stack Registers
31
16
15
0
A0
A1
A2
A3
A4
A5
A6
A7
User Stack Pointer
Supervisor Stack Pointer
23
0
Program Counter
32-Bit ALU
(or two 16-Bit ALUs)
Instruction Register
Instruction Decode
A 2-Level Microcoded Control Store
31
16
15
8
7
0
D0
D1
D2
D3
D4
D5
D6
D7
Data Registers
15
0
Status Register
Register and ALU Function Selection
Control Signals to Execution Unit
Timing and Control Circuitry

Intel 80286 i 80386

• CLOCK input

Frekvencija signala na ulazu CLOCK interno se dijeli s 2:

fPCLOCK(ProcessorCLOCK) = ½ fCLOCK

Intel 486 DX

fPCLOCK(ProcessorCLOCK) = fCLOCK

Proizvođači se obično referenciraju na PCLOCK

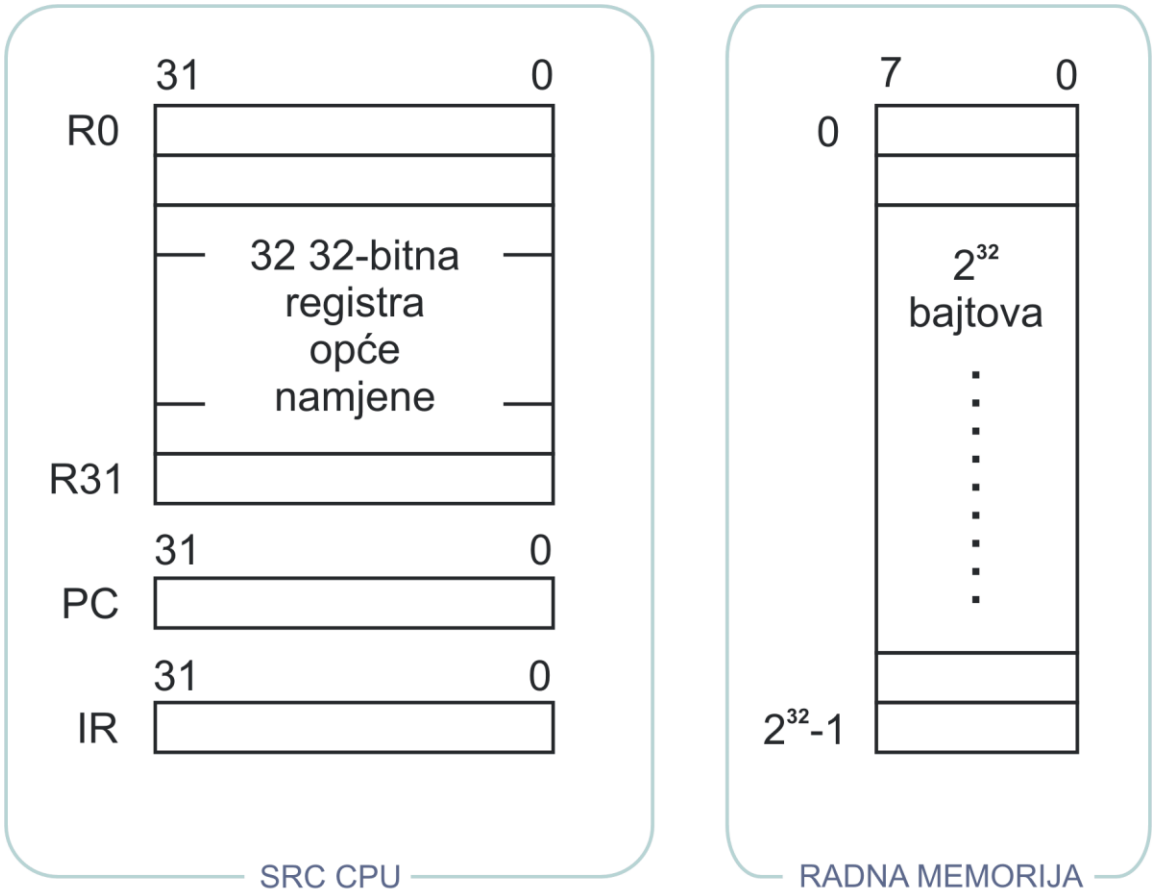
Intel 486 DX4

fPCLOCK(ProcessorCLOCK) = 2fCLOCK
3fCLOCK

Izvor: T. Shanley, D. Anderson, ISA System Architecture, Addison-Wesley, 1995.

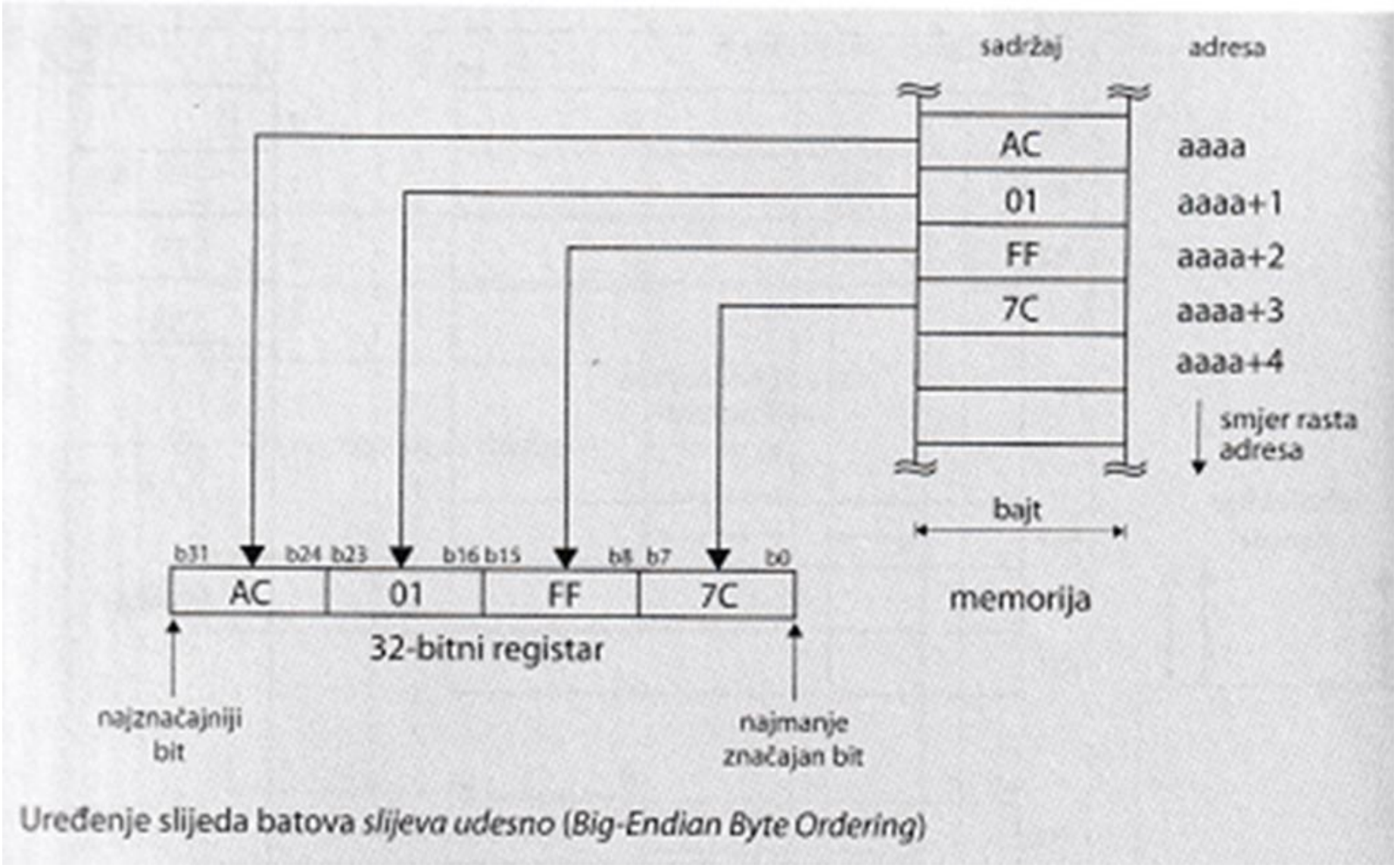
SRISC – Simple RISC

- Programski model



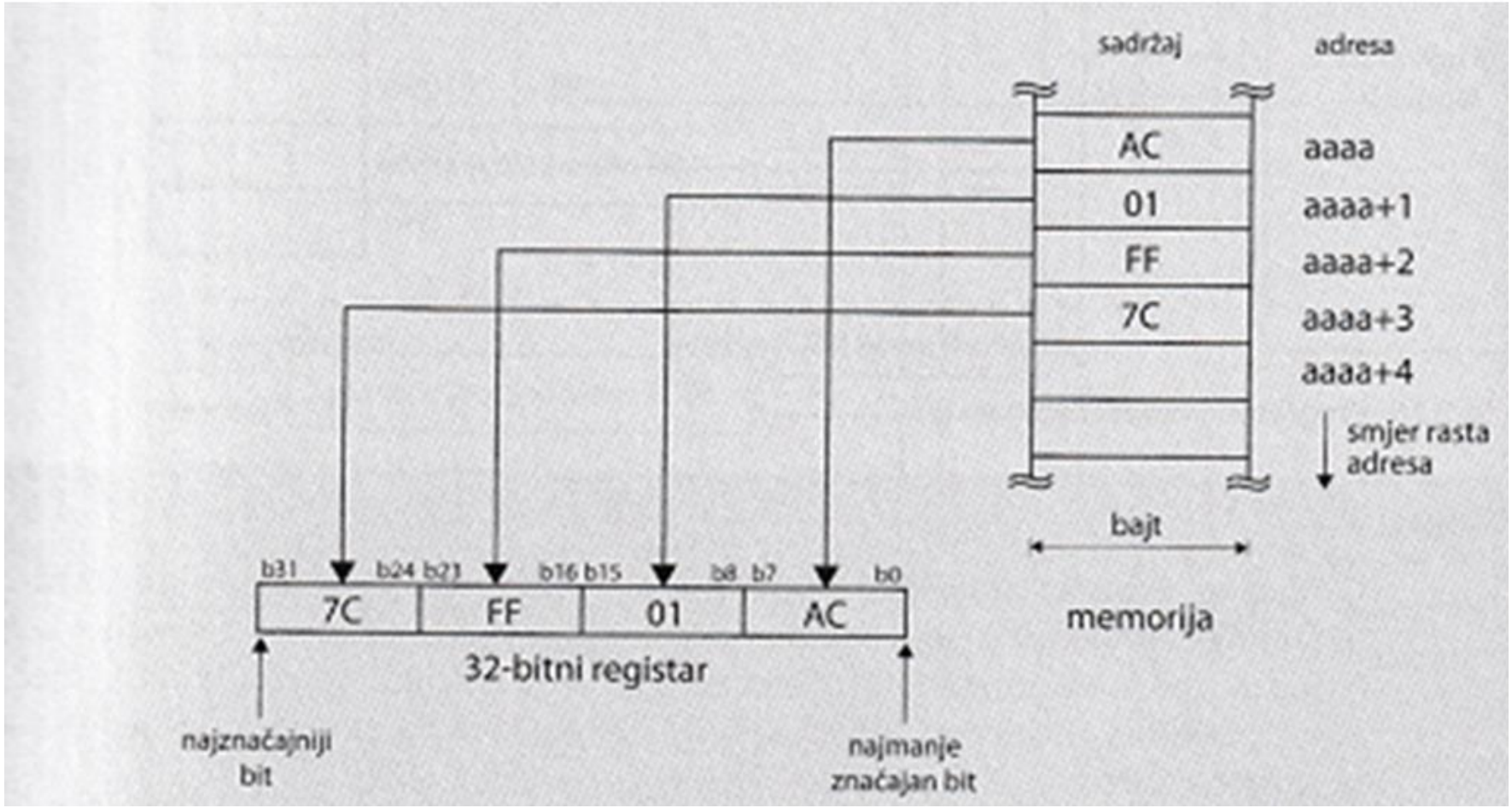
Značajke:

- Memorija bajtno organizirana – dohvat samo 32-bitne riječi
- *load/store* arhitektura
- Big – Endian Byte Ordering

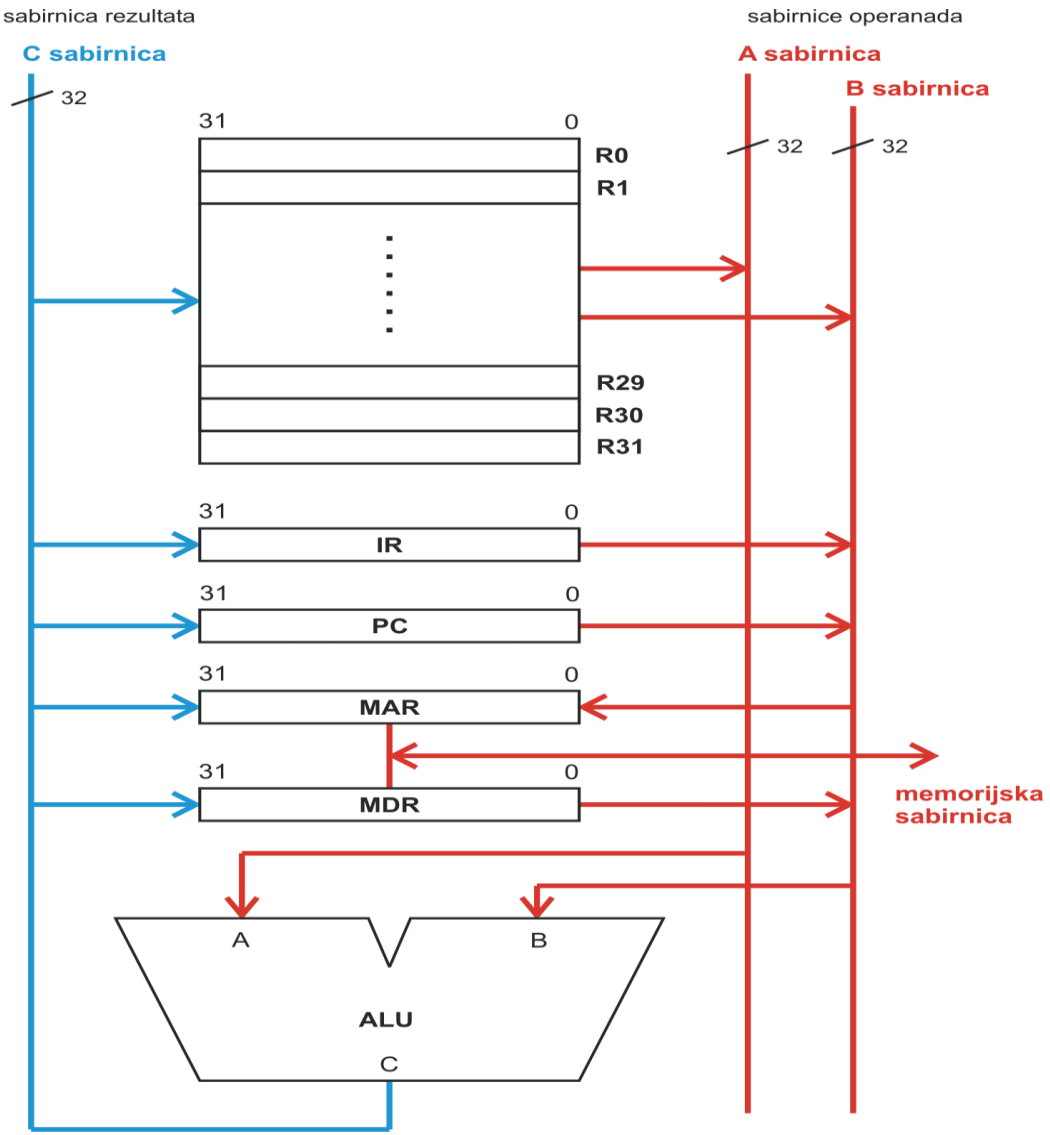


Uređenje slijeda batova *slijeva udesno (Big-Endian Byte Ordering)*

Little-Endian Byte Ordering



Trosabirnički SRISC

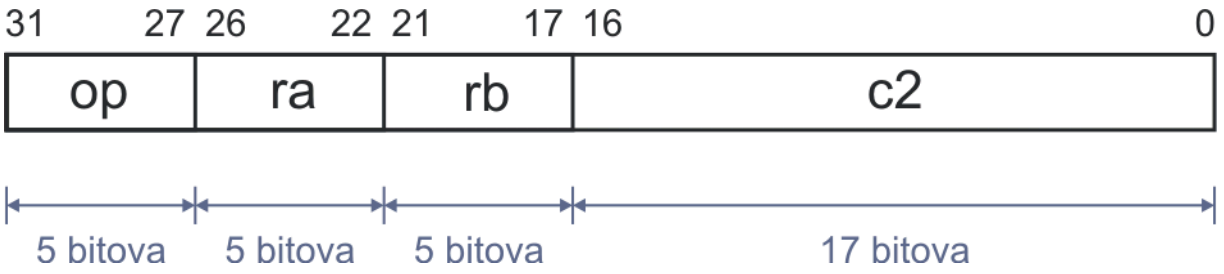


ISA – Instruction Set Architecture

SRISC ima osam različitih formata instrukcija:

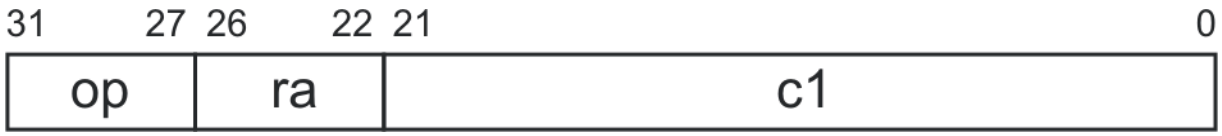
1. format

Instrukcije: *ld*, *st*, *la*, *addi*, *andi*, *ori*



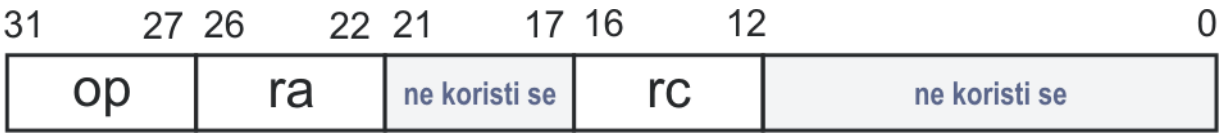
2. format

Instrukcije: *ldr, str, lar*



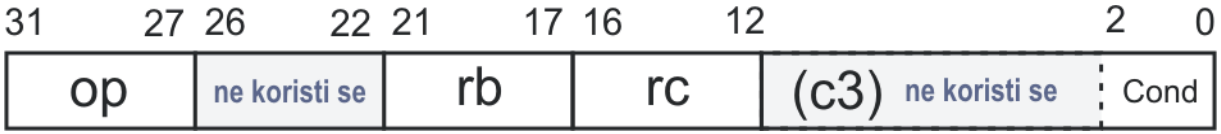
3. format

Instrukcije: *neg*, *not*



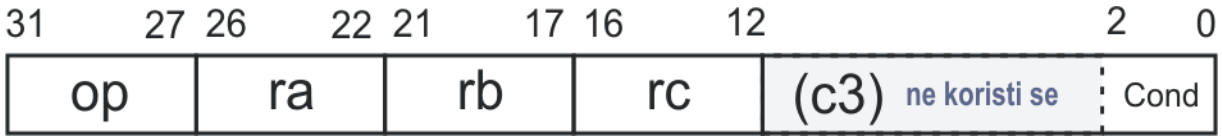
4. format

Instrukcija: *br*



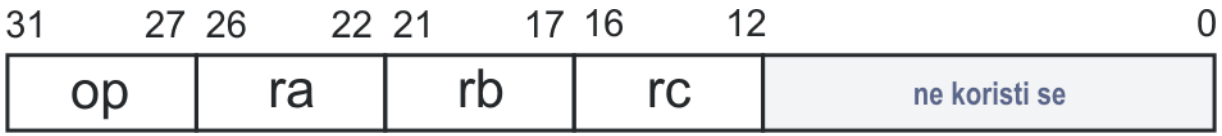
5. format

Instrukcija: *brl*



6. format

Instrukcije: *add*, *sub*, *and*, *or*



Troadresni format instrukcije!

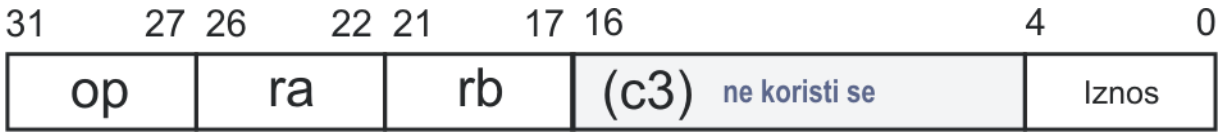
7. format

Instrukcije: *nop*, *stop*

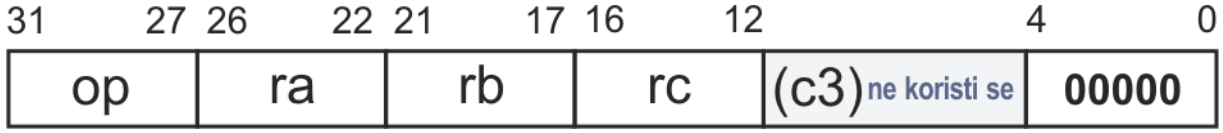
8. format

Instrukcije: *shr, shl*

a)



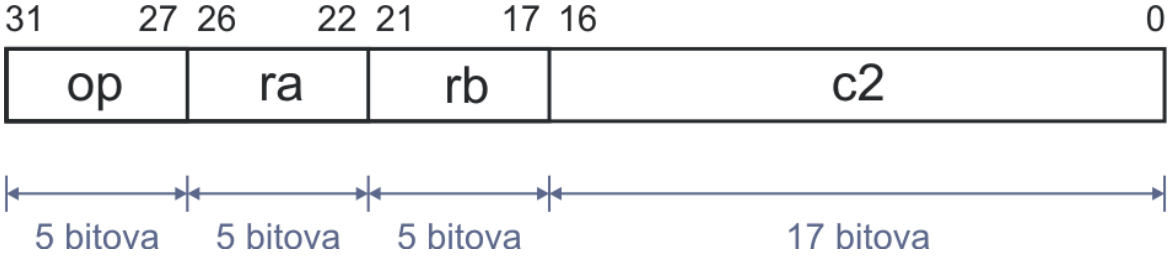
b)



Instrukcije za pristup memoriji (load i store tip instrukcija)

- *load* i *store* **jedine** su instrukcije za dohvat i pohranu operanada iz/u memorije (i):

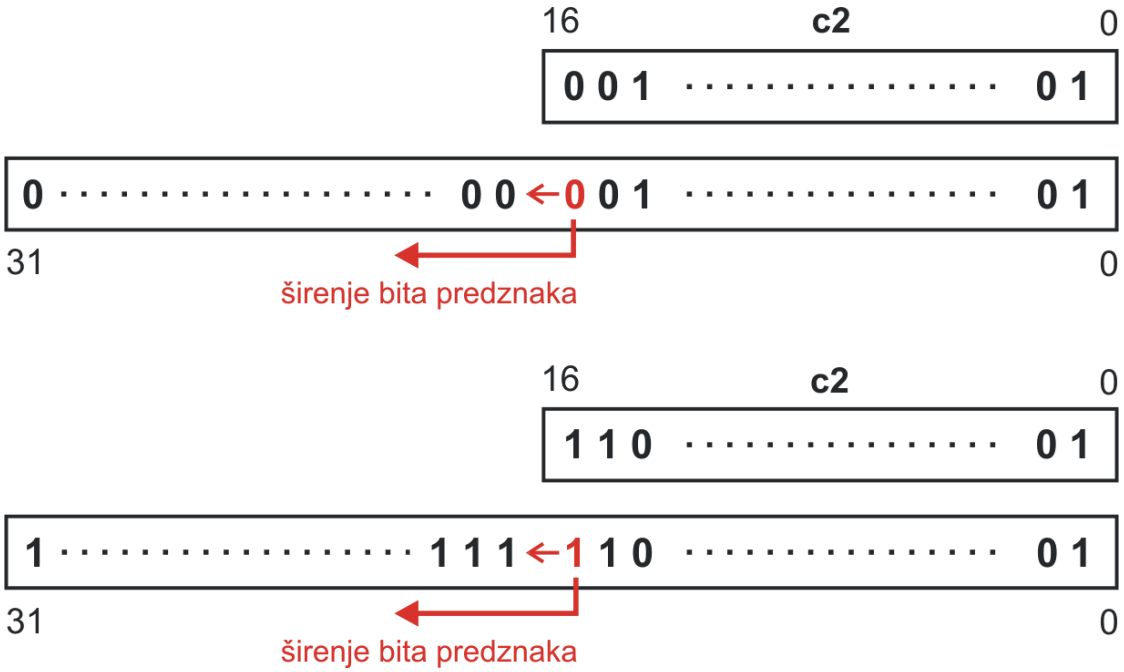
4 *load* instrukcije: *ld, ldr, la, lar*
2 *store* instrukcije: *st, str*



load: /mnemonik ld/

- puni registar određen 5-bitnim poljem *ra* (1. format instrukcije)
- adresa izvorišta određena je 17-bitnom vrijednosti u polju *c2*
- polje *rb* ima dvostruku ulogu:
    1. Ako je *rb* = 0 (00000_B) tada (*rb* = 0) služi kao "signal" upravljačkoj jedinci da je adresa memorijske lokacije (izvorišta) vrijednost *c2* koja se pretvara u 32-bitnu i to **širenjem bita predznaka (engl. sign-extended)**

Širenje bita predznaka



2. Ako je $rb \neq 0$ tada se adresa memorije oblikuje kao $c2$ + R[rb] (based, displacement addressing mode)

POZOR: zbrajanje $c2$ + R[rb] obavlja se TIJEKOM izvođenja instrukcije (engl. run time)

Instrukcije koje koriste 1. format instrukcije:

ld ra, c2 ; izravno (direktno) adresiranje: R[*ra*] = M[*c2*]

ld ra, c2(rb) ; indeksno adresiranje (*rb* $\neq 0$): R[*ra*] = M[*c2* + R[*rb*]]

store: /mnemonik st/

-Instrukcija *st* obavlja "obrnutu" (u odnosu na *ld*) operaciju:

1. Ako *rb* = 0 (00000_B) tada se R[*ra*] pohranjuje na memorijskoj lokaciji s adresom *c2*

2. Ako je *rb* $\neq 0$ tada se R[*ra*] pohranjuje na memorijskoj lokaciji s adresom *c2* + R[*rb*]

Instrukcije koje koriste 1. format instrukcije:

st ra, c2 ; izravno adresiranje M[*c2*] = R[*ra*]

st ra, c2(rb) ; indeksno adresiranje (*rb* $\neq$ 0): M[*c2* + R[*rb*]] = R[*ra*]

Instrukcije koje koriste 1. format instrukcije (nastavak):

la – load address

- računa adresu operanda (kao u prethodnim slučajevima) ali umjesto dohvata operanda pohranjuje se izračunata adresa u R[*ra*]

la ra, c2 ; napuni R[*ra*] s adresnim pomaknućem: R[*ra*] = *c2*

la ra, c2(rb) ; napuni R[*ra*] s adresnim pomaknućem: R[*ra*] = *c2* + R[*rb*]

la se koristi za sintezu složenijih načina adresiranja!

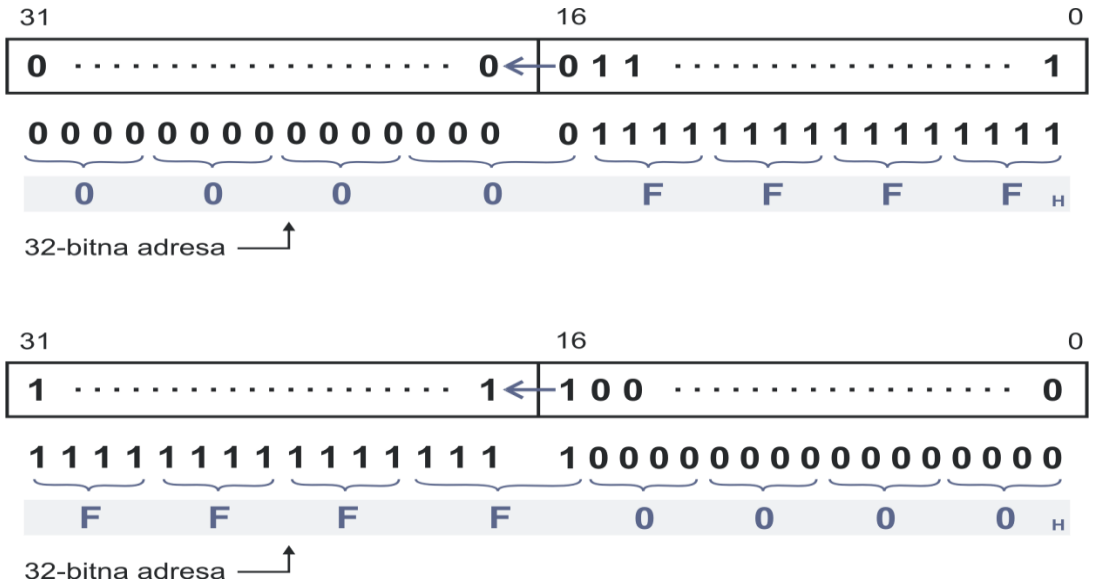
Ograničenje izravnog adresiranja

Pazi: *c2* je 17-bitna konstanta

Operandi se nalaze:

1) Na prvih 2^{16} bajtova:

2)



Adresni prostori:

00000000 - 0000FFFF	1. prostor
FFFF0000 – FFFFFFF	2. prostor

Kako ostvariti pristup operandima drugdje u memoriji?

Za pristup operandima drugdje u memorijskom prostoru:

• Adresiranje pomaknućem: R[*rb*] – baza
c2 – pomaknuće

efektivna adresa = R[*rb*] + *c2*

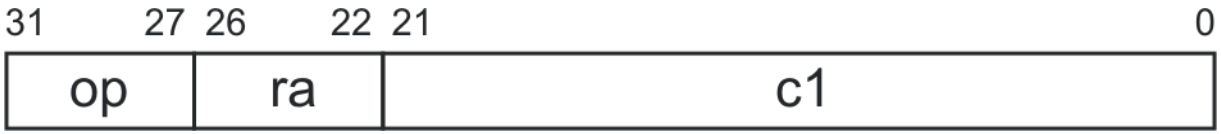
• Indirektno adresiranje: uporaba R[*rb*] pri *c2* = 0

efektivna adresa = R[*rb*] + 0

POZOR: U postupku računanja (zbrajanja) adrese prvo se 17-bitno pomaknuće treba pretvoriti u 32-bitni broj

Instrukcije: *ldr, str, lar*

Koriste 2. format instrukcije



ldr ra, c1 ; napuni registar R[*ra*] = M [PC + *c1*]
; relativni način adresiranja

str ra, c1 ; pohrani M [PC + *c1*] = R[*ra*]

lar ra, c1 ; napuni relativnu adresu: R[*ra*] = PC + *C1*

POZOR: Efektivna se adresa
oblikuje tijekom izvođenja instrukcije (run-time addition)

Premjestive instrukcije (engl. Relocatable instructions)

- relativne adrese (u odnosu na PC) čine instrukcije premjestivim – cijeli se programski moduli mogu premještati bez mijenjanja vrijednosti pomaknuća

- *c1* je duljine 22 bita – može se specificirati adresa u prostoru od $\pm 2^{21}$ u odnosu na tekuću instrukciju

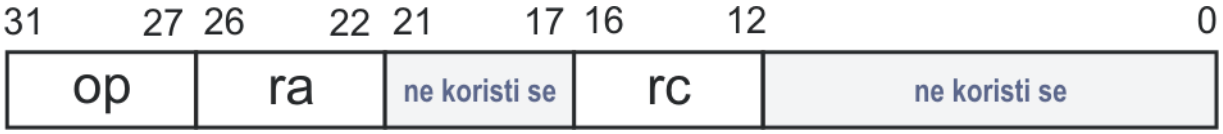
Primjeri instrukcija *load* i *store*

Instrukcija	op	ra	rb	c1	Komentar	Način adresiranja
`ld r1,32`	1	1	0	32	R[1] ← M[32]	Izravno
`ld r22,24(r4)`	1	22	4	24	R[22] ← M[24+R[4]]	Pomaknuće
`st r4,0(r9)`	3	4	9	0	M[R[9]] ← R[4]	Reg.Indirektno
`la r7,32`	5	7	0	32	R[7] ← 32	Usputno
`ldr r12,-48`	2	12	-	-48	R[12] ← M[PC-48]	Relativno
`lar r3,0`	6	3	-	0	R[3] ← PC	Registar

Aritmetičke i logičke instrukcije

• Instrukcije s jednim operandom

- Koriste 3. format instrukcije:

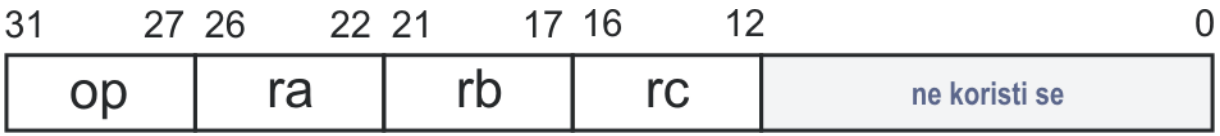


neg ra, rc ; R[*ra*] = - R[*rc*] / dvojni komplement/

not ra, rc ; R[*ra*] = $\overline{\text{R[}rc\text{]}}$ /jedinični komplement/

• ALU instrukcije (binarne operacije)

- koriste 6. format instrukcije (troadresni!):



add ra, rb, rc ; R[*ra*] = R[*rb*] + R[*rc*]

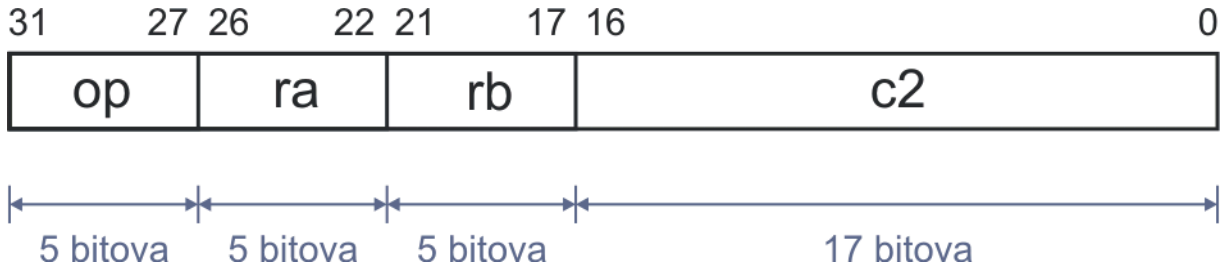
sub ra, rb, rc ; R[*ra*] = R[*rb*] - R[*rc*]

and ra, rb, rc ; R[*ra*] = R[*rb*] $\wedge$ R[*rc*]

or ra, rb, rc ; R[*ra*] = R[*rb*] $\vee$ R[*rc*]

• ALU instrukcije (usputno adresiranje)

- koriste 1. format instrukcije:



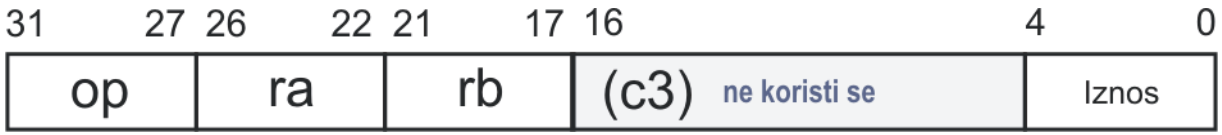
addi ra, rb, c2 ; R[*ra*] = R[*rb*] + *c2* : *c2 – usputna konstanta*

andi ra, rb, c2 ; R[*ra*] = R[*rb*] $\wedge$ *c2*

ori ra, rb, c2 ; R[*ra*] = R[*rb*] $\vee$ *c2*

• Posmačne instrukcije

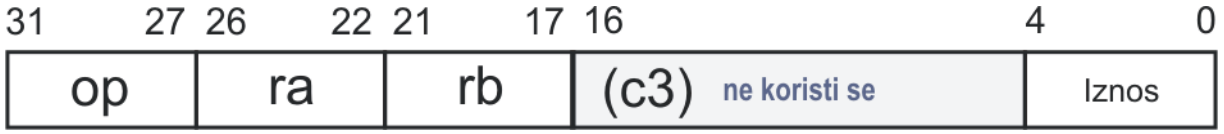
- koriste 8a) format instrukcije:



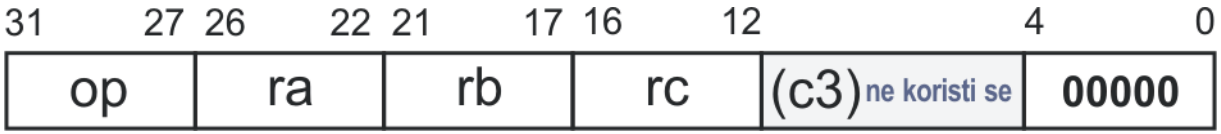
Instrukcije posmiču operand iz R[*rb*] (udesno, ulijevo ili kružno) i smještavaju rezultat u R[*ra*].

Broj posmaknutih mjesta određen je 5-bitnim nepredznačenim brojem

5-bitna vrijednost → iznos posmaka od 0 do 31



Ako je ta vrijednost 0 onda se kao iznos posmaka uzima 5 LSb registra R[*rc*] (format 8b))

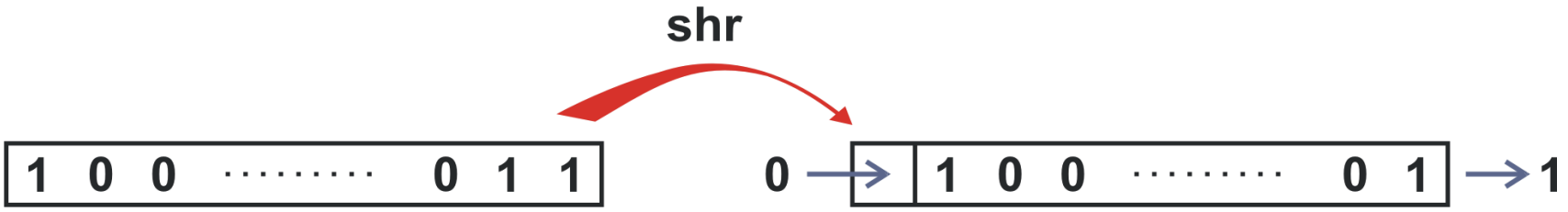


-Razlikujemo dvije vrste posmaka:

1) logički posmak
2) aritmetički posmak

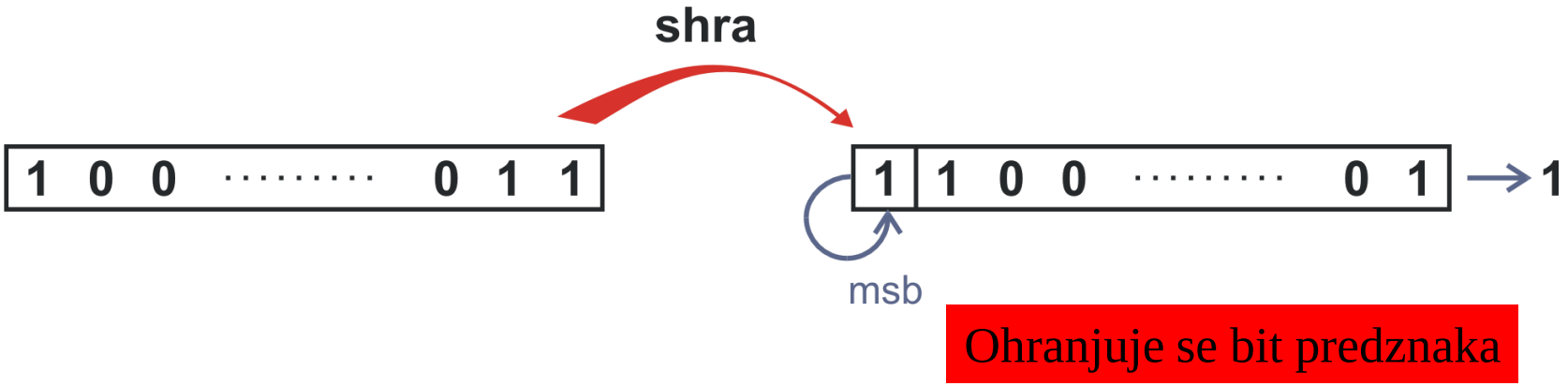
Primjer:

shr logički posmak udesno



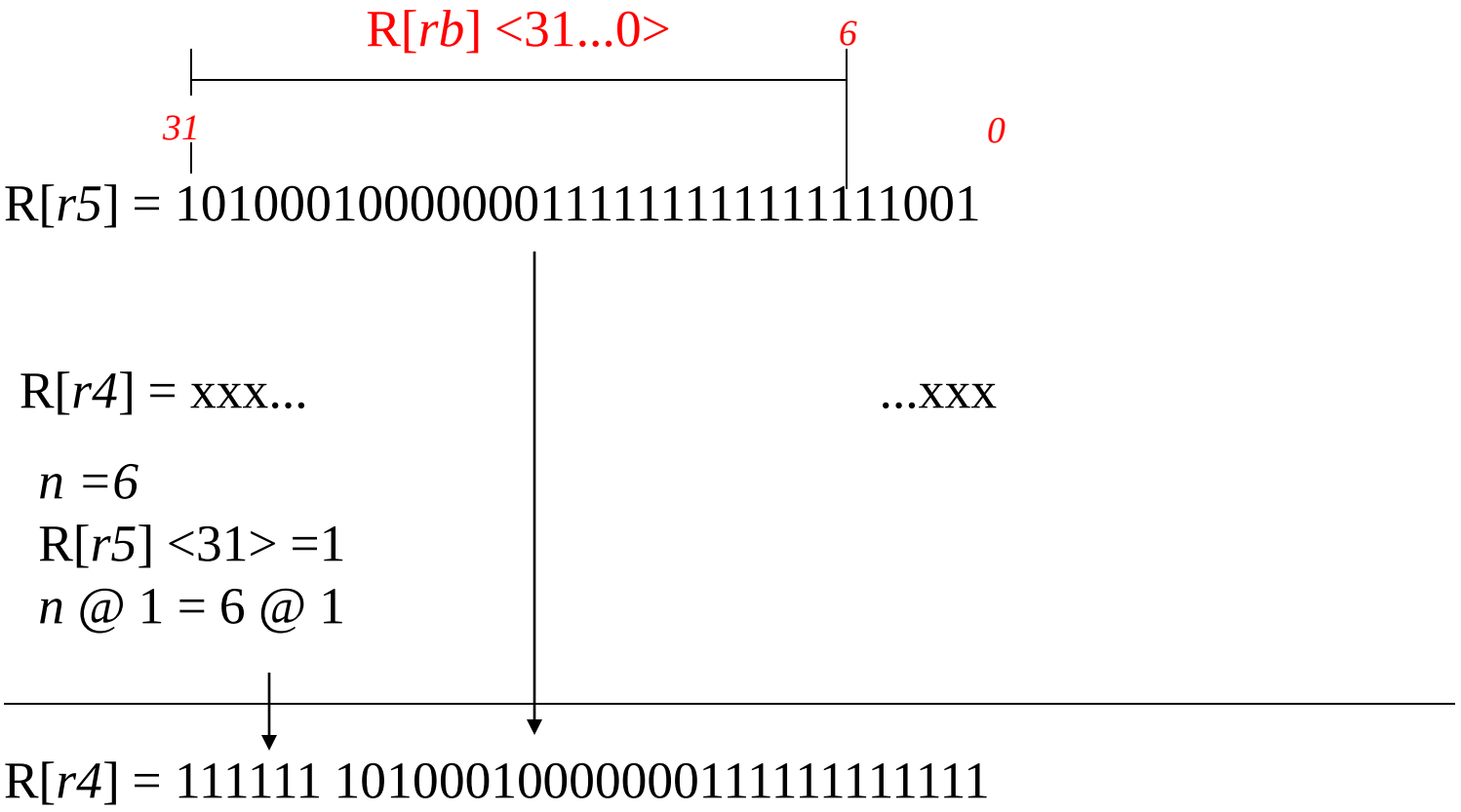
Primjer:

shra aritmetički posmak udesno



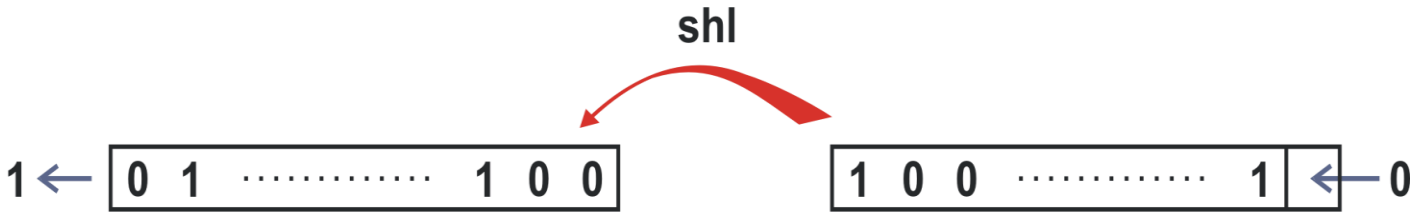
shra (:=op = 27) ⟶ R[*ra*] <31...0> ⟵ (*n*@R[*rb*] <31>) # R[*rb*] <31...n>

shra r4, r5, 6



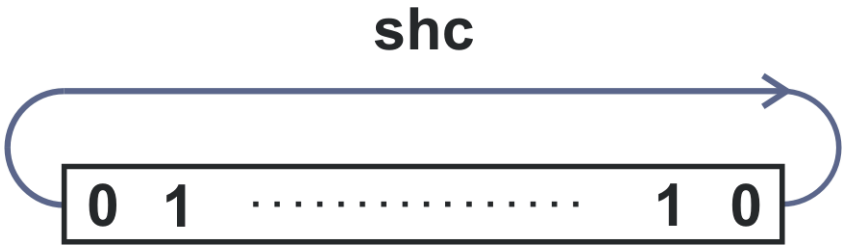
Posmak ulijevo

shl



Kružni posmak

shc



Posmačne instrukcije – primjeri:

shr ra, rb, rc ; posmakni R[*rb*] (za broj mjesta koji je određen s R[*rc*]) i pohrani ga u R[*ra*]

shr ra, rb, count ; posmakni R[*ra*] udesno i pohrani ga u R[*ra*] (za broj mjesta određen u 0-4 polju c3)

shra ra, rb, rc

shl ra, rb, rc

shl ra, rb, count

shc ra, rb, rc

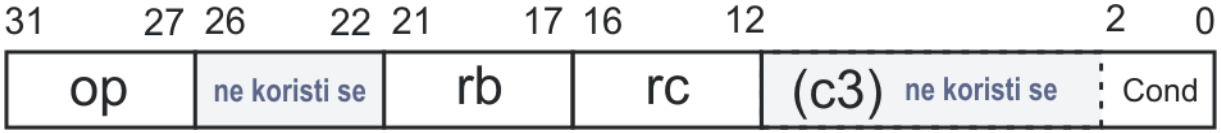
shc ra, rb, count

*A*ko je *count* polje u instrukciji 0 (format 8b) tada se za broj mjesta *posmaka* uzima iz registra koji je određen bitovima instrukcije na poziciji 12 - 16

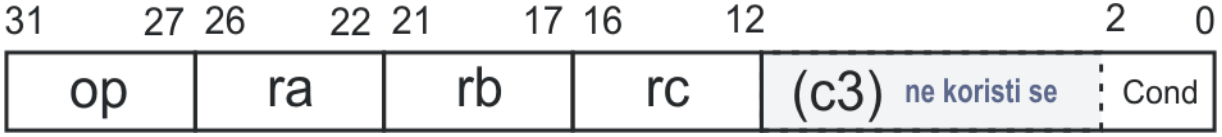
Instrukcije grananja

br i *brl*

Instrukcije koriste 4. i 5. format:



rb – određuje registar R[rb] koji sadržava 32-bitnu adresu ciljne instrukcije
rc – određuje registar R[rc] čiji će sadržaj biti ispitan
Instrukcija *br* se izvodi tako da se mijenja sadržaj PC s ciljnom adresom



brlpl ra, rb, rc → sadržaj PC-a smještava se u R[ra], zatim se ispituje uvjet je li sadržaj registra R[rc] pozitivan, i ako je, onda se sadržaj registra R[rb] smješta u PC. Ako uvjet grananja nije zadovoljen grananje se neće dogoditi ali će sadržaj PC-a biti pohranjen u R[ra]

Instrukcija *brl* (branch&link) se izvodi tako da kopira PC u povezni (engl. linkage) registar i to prije grananja

Povezni registar dopušta povratak iz potprograma i rabi se za implementaciju procedura i funkcija u HLL

Pozor: PC se kopira u povezni registar bez obzira da li će se grananje izvesti (ili ne)

Instrukcije grananja (uvjeti!):

zbirni jezik	*c3*<2...0>	Uvjeti grananja
brnv, brlnv	*0*	*Nikada*
br, brl	*1*	*Bezuvjetno*
brzr, brlzr	*2*	*Ako je* R[*rc*] = 0
brnz, brlnz	*3*	*Ako je* R[*rc*] ≠ 0
brpl, brlpl	*4*	*Ako je* R[*rc*]<31> = 0
brmi, brlmi	*5*	*Ako je* R[*rc*] < 0

Primjer:

brl ra, rb, rc, c3 ; R[*ra*] ← PC, ispituje uvjet (c3) za podatak
; smješten u R[c] i granaj na ciljnu
; adresu određenu s R[*rb*] ako je
; zadovoljen uvjet *c3*

Mješovite instrukcije

nop ; ne čini ništa

stop ; zaustavi stroj

nop ??? ⟶ *važna instrukcija u protočnoj izvedbi procesora!!!*

ARM 6 procesor - Acorn RISC Machine