

UVOD:

ZEMRIS

prof. dr. sc. Slobodan Ribanić

Asistenti:

v.a. dr. sc. Zoran Kalafatić,

doc. dr. sc. Željka Mihajlović

doc. dr. sc. Boško Radej

Tomislav Hrkac, dipl. ing.

ARHITEKTURA I ORGANIZACIJA RAČUNALA

0. Uvodno predavanje (Turingov stroj)
1. Definicija i klasifikacija arhitekture računala
2. Model von Neumannova računala
3. Pojednostavljeni modeli (mikro) procesora
CISC i RISC arhitektura
4. Upravljačka jedinica računala
(sklopovska izvedba)
5. Aritmetsko-logička jedinica
6. Memorijski sustav (virtualna, pribuđena memorija)
7. Izvedba mikroprogramске upravljačke jedinice
8. Ulazno-izlazni sustav računala i načini
komunikacije računalo-vanjski svjet
9. Arhitektura 8-, 16-, 32-, 64-bitnih
CISC/RISC procesora

CISC - Complex Instruction Set Computer
RISC - Reduced Instruction Set Computer

Literatura:

S. Ribanić : Naprednije arhitekture
mikroprocesora

(ELEMENT, Zag. 1996.)

S. Ribanić : RISC i CISC arhitektura
računala

(ŠK, Zag.)

A.S.Tennenbaum : Structured Computer
Organization

(Prentice-Hall)

D.A.Peterson /

J.L.Hennessy : Computer Architecture,
A Quantitative Approach

(Morgan Kaufmann Pub. 1996.)

	Predviđanja godine 1989. za 1996.g.	Stvarno (1996)	Predviđanja godine 1996. za 2000 g.	Predviđanja godine 1996. za 2006.g.
ZNAČAJKE:				
Tranzistori: (10^6)	8	6	40	350
Tehnologija: (microns)	0.35	0.35	0.2 (već (realizirano 0.1!))	0.1
PERFORMANSA:				
MIPS	100	400	2400	20 000
ISPEC 95	2.5	10	60	500
Clock Speed: (MHz)	150	200	300	4000*

MIPS - Million Instructions Per Second

ISPEC - Integer SPEC marks

1 SPEC mark = 1 SPARC station 10/40 (izvor A.Yu 1996.)

SPEC - Standard Performance Evaluation Corporation
udružja proizvođača izabrala je skup
programa na kojima testira brzinu
svojih proizvoda (OBJETIVNOST TESTA !!!)

- * poboljšana sučelja čovjek-procesor
(3D grafika, slike u pokretu, sinteza govora,
raspoznavanje govora, raspoznavanje i
interpretacija slike)
- * procesori za multimedijalne sastave

DANAS: SPEC OFU 2000

Referentni stroj: 300 MHz SunUltra 5-10
(100)

TURINGOV STROJ

OBRADA PODATAKA: svršishodna djelatnost koja ima za cilj da iz raspoloživih podataka dobije traženu informaciju.

- podatak (objekt u obradi)
- algoritam (uputstvo koje opisuje transformaciju početnih podataka u traženi rezultat)
- izvršitelj (stroj; računalo)

ALGORITAM: postupci transformacije grupirani su u korake algoritma.

Svojstva:

1. određenost
2. konačnost
3. širina primjene

1. Određenost se osigurava binjenicom da izvršitelj može izvoditi operacije zadane u koracima i jednoznačno razaznavati pojedine korake.
2. Konačnost: definirati uvjet PREKIDANJA petlje algoritma (alg. ima konačno mnogo koraka)
3. Širina primjene: na koje se važne podatke algoritam odnosi?

Računanje je proces određivanje izlazne supstitucije za zadaniu važnu supstituciju, koja se pokorava svim specifičnim svojstvima problema (GAREY & JOHNSON, 1979.)

PROBLEM NAPRINJAČE

Zadano: Naprinjača nosivosti (kapaciteta) C (kg)
N objekata (predmeta) definirani težinom W_k i vrijednošću V_k ; ($k=1, 2, \dots, N$)

Nadite: udio svakog objekta tako da:

- ne prenabivate naprinjaču
- maksimizirate vrijednost koju nosite

(Opaska: Objekti se ne mogu rastaviti na dijelove)

Parametri (variable) problema: $C, N, W_1, V_1, F_1, \dots, W_N, V_N, F_N$

Primjer važne supstitucije:

$$\begin{array}{lll} C = 14 \text{ kg} & N = 3 \\ W_1 = 4 \text{ kg} & W_2 = 6 \text{ kg} & W_3 = 7 \text{ kg} \\ V_1 = 30 \$ & V_2 = 48 \$ & V_3 = 50 \$ \end{array}$$

Primjer rješenja

(izlazna supstitucija): $\left. \begin{array}{l} 100\% \text{ OBJEKAT } 1: 2 \\ 41\% \text{ OBJEKTA } 3 \end{array} \right\} \text{ukupna vrijednost } 106.57 \$$

ALGORITAM:

1. KORAK: Razvrstaj objekte na temelju omjera vrijednosti i težine

2. KORAK: (Ponavljaj sve dok se naprtnjača ne prenestupa)

Iz skupa objekata uzmi objekt s najvećim omjerom V/W i smjesti ga cijelog u naprtnjaču.

3. KORAK: Uzmi iz naprtnjače posljednji objekt i razdijeli ga tako da upravo njegov dijelovi popune naprtnjaču.

(Što je izvesnito sofistiranoji - Algoritam je jednostavniji)

TURINGOV STROJ:

1. Raščlanjivanje računskog postupka na jednostavne, elementarne operacije kod Turingova stroja (TS) dovedeno grotov do granične mogućnosti
2. Vanjska memorija TS je s obje strane neograničena traka, podijeljena na polja.

Stroj raspolaže konačnim brojem znakova (simbola): $S = \{s_1, s_2, \dots, s_k\}$ koji oblikuju VANJSKU ABECEDU

$$S = \{s_1, s_2, \dots, s_k\}$$

b - prazni (pusti) simbol

- upisivanje praznog znaka u bilo koje polje trake brane znak koji se tamo nalazio

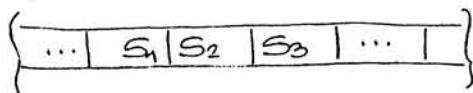
- svako polje sadrži samo 1 znak.

- izraz zapisan na traci predstavljen je konačnim nizom znakova vanjske abecede (\neq od praznog)

$T = \{t_1, t_2, \dots, t_m\}$ - skup simbola koji se pojavljuju na traci BEZ PRAZNIH SIMBOLA

Rad stroja:

- na početku rada stroja na traku zapisujemo početni izraz.



$$T = S/b$$
$$S = \{s_1, s_2, \dots, s_k\} \text{ be } S$$

Rad stroja se odvija u taktovima:

- vrem. diskretan stroj
- odvijanjem taktova stroj pređe informaciju preoblikuje u novu informaciju
- na kraju \neq takta znaci zapisani na traci oblikuju odgovarajuću novu informaciju.

A - početna informacija

Dva slučaja:

1. Nakon konačnog br. taktova stroj staje davši STOP signal i pri tome je na traci zapisana informacija B

→ stroj je primjenjuv na poč. inf. A i prenosi je u informaciju ili rezultat B

2. Stroj nikad ne staje i nikad ne daje STOP signal

→ stroj nije primjenjuv na poč. informaciju A

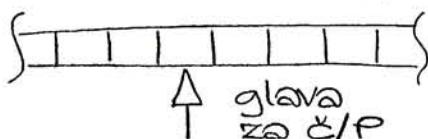
$$A \rightarrow B$$

$Q = \{z_1, z_2, \dots, z_n\}$ - skup unutrašnjih stanja stroja

z_0 - početno stanje stroja

z_f - konačno stanje stroja

Sustav elementarnih stanja operacija i naredbi (jednoadresnih) vrlo je jednostavan: u svakom taktu svaka naredba propisuje samo zamjenju pojedinačnog simbola si upisanog u promatrano polje nekim drugim znakom sj (potrebna je glava za ČITANJE/PISANJE)



- u 1 taktu stroj može i pomaknuti glavu za 1 mjesto (lijево ili udesno) bez mijenjanja simbola.

Ako je $i=j \Rightarrow$ sadržaj promatrano polja se ne mijenja
 $-"1- Si=b \Rightarrow -"1- -"1- -"1-$ se briše

Pri prelazu s 1 taka na 2. address promatrano polje može se promijeniti najviše za jedinicu.

Glava stroja promatra lijevo susjedno ili desno susjedno polje (ili isto polje)

- tri standardne adrese

D - promatraj desno polje

L - -"1- lijevo -"1-

∅ ili N - ostani na istom polju

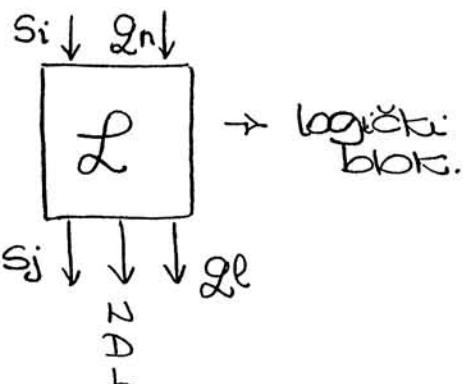
Obrada inf. u TS odvija se u logičkom bloku L koji se može nalaziti u nekom od konačnog broja stanja stroja.

$$Q = \{q_1, q_2, \dots, q_{|Q|}\}$$

$$P = \{N \text{ ili } \phi, D, L\}$$

Wazna dvjeka: s_i, q_n
Izazna trojka: s_j, p, q_e

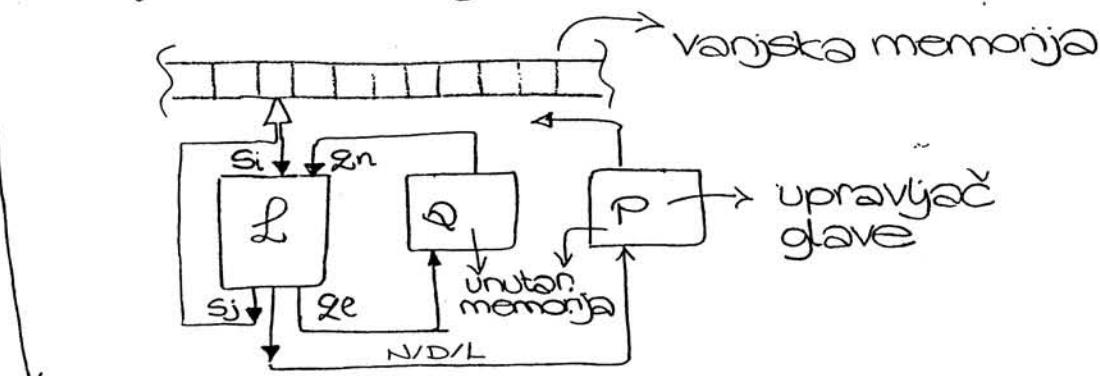
$$p \in P$$



Logički blok realizira funkciju koja svakom paru znakova ul.dvijeke pridružuje izaznu trojku.

δ - logička funk. stroja sastoji se od niza zapisa koji mogu imati 1 od 3 oblike:

- 1) $(q_n, s_i) \rightarrow (q_e, s_i/j, \phi)$
- 2) $(q_n, s_i) \rightarrow (q_e, s_i/j, D)$
- 3) $(q_n, s_i) \rightarrow (q_e, s_i/j, L)$



Log. funk. stroja može se prikazati tablicom:
FUNKCIJALNA SHEMA STROJA:

	q_1	q_2	q_3	...
\wedge	$q_1 \wedge D$	$q_2 \wedge L$	$q_3 \wedge D$	
$ $	$q_1 \wedge N$	$q_2 \wedge N$	$q_3 \wedge L$	
$\&$	$q_1 \wedge L$	$q_2 \wedge D$	$q_3 \wedge L$	
β	$q_1 \beta L$	$q_2 \beta D$	$q_3 \wedge L$	

1 stupac: skup simbola koji se mogu pojaviti + blanc

1 redak: stanja sklopa (TS)

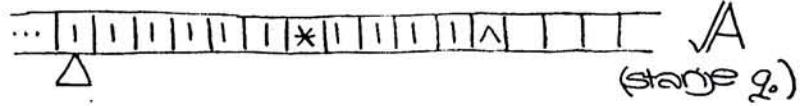
$$TS = \{Q, S, T, b, q_0, q_f, \delta\} \rightarrow \text{definicija T. stroja}$$

Memorija Turingovog stroja:

- Vanjska memorija = traka s mnogo registara
- Unutarnja memorija = polje za sljedeću naredbu
 - ↳ (polje Q za znak stanja)
 - ↳ (polje P za znak komata)

PremjER ZBRAJANJA TS:

Željeni rezultat:



Na vrpoli se nalazi skup artica koji odgovara zbroju artica u početnom stanju.

S - varjska abeceda = { 1, *, b } ^

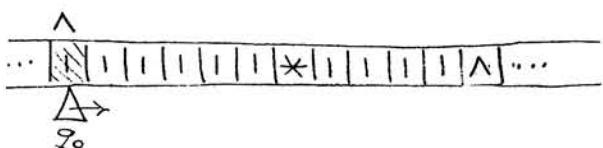
Q - skup konačnih stanja = $\{q_0, \dots, q_f\}$

niz stanja $\hookrightarrow ? = Q_f$

Bessere:

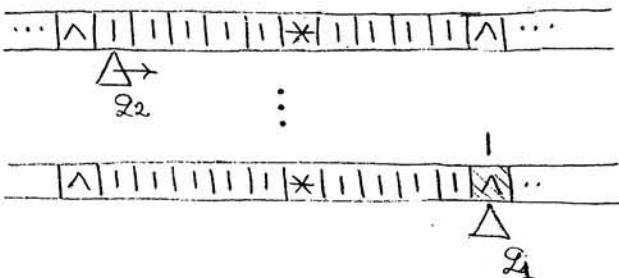
	g_0	g_1	g_2
1	$g_0 \wedge D$	$g_1 \wedge L$	$g_2 \wedge D$
\wedge	$g_0 \wedge D$	$g_0 \wedge D$	$g_1 \wedge N$
$*$	$g_0 \wedge N$	$g_1 \wedge L$	$g_2 \wedge D$

⇒ FUNKCJONALNA SHEMA STROJA



KONFIGURACIJA 1:

TS izlošće prutčić,
prelazi u starje ge
glav pomijeđe za 1 Desno.



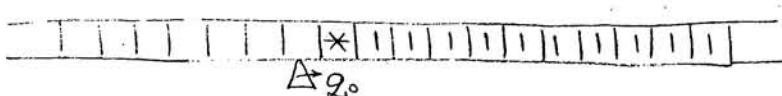
KONFIGURACIJA 2:

pomak udesno zač (D)

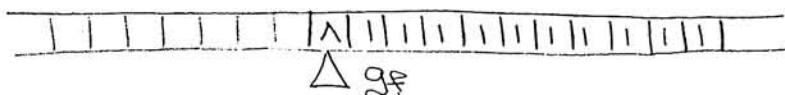
27

KONFIGURACIJA 12:

Ts zapise je žapic
odlazi u stanje gdje
ne pomriće glavu



PREDZADNJA KONFIGURACIJA



KONAČNA KONFIGURACIJA

$$Q = \{q_0, q_1, q_2, q_f = ?\} \quad \rightarrow \text{skup stanja} \\ S = \{1, *, b = \wedge\} \quad \rightarrow \text{varijable abeceda}$$

Pomučamo se čas ujevo, čas udesno do prutida i prenosa ga u cjelav niz. (nadovezan udesno)

k-ta konfiguracija stroja;

slika trake stroja s kompletnom informacijom, koja se na njoj nalazi na početku k-tog taktu, pri čemu je ispod promatranoog polja zapisan znak unut. stajala koje ulazi u log. blok ℓ u početku k-tog taktu.

2. PRIMJER:

Inkrementiranje broja predstavljenog u dekadskom sustavu ($n \rightarrow n+1$)

Zadan je dekadski zapis broja n . Treba nadati dekadski zapis broj $n+1$.

$$S = \{0, 1, 2, 3, 4, 5, 6, 7, 8, 9, \Lambda\} \quad (\Lambda = b)$$

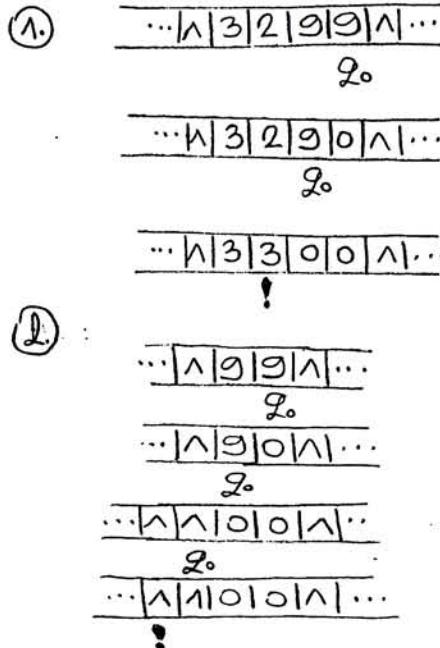
$$Q = \{q_0, q_f\}$$

→ Stroj može biti samo u 2 stanja: q_0 radno, odn početno, te q_f (konačno) stanje. ($q_f = ?$)

FUNK. SHEMA

	q_0	q_f
0	1 ! N	
1	2 ! N	
2	3 ! N	
3	4 ! N	
4	5 ! N	
5	6 ! N	
6	7 ! N	
7	8 ! N	
8	9 ! N	
9	0 ? L	
Λ	1 ? N	

Primjeri konfiguracije:



DEFINICIJA I KLASIFIKACIJA ARHITEKTURE RACUNALA

- izraz "arhitektura računala" (IBM, 60.ih god. last.)

H.H. Enslow

I. Flores

Algoritmi i postupci koji se upotrebljavaju u osnovnim funkcijama jedinicama
(ALU, U/I, upravljačka jedinica, memorija)

E.C. Joseph

- arhitektura funkcionalno usmjerena
(poput arhitekture u modernom građevinarstvu)

CIJEVI:

- povezanje propusnost
- postzoranje prilagodljivost (flexibility)
 - " - pouzdanost (reliability)
 - " - raspoloživost
- niža cijena sustava

TRI SUSTAVNA PODRUČJA ARHITEKTURE:

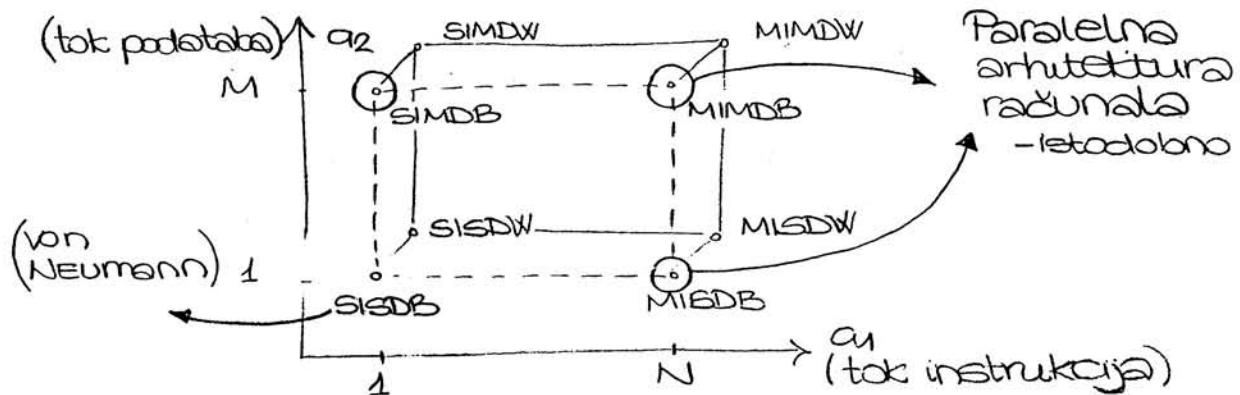
- 1) Sklopovska oprema (HARDWARE)
- 2) Programska -" (SOFTWARE)
- 3) Odnos čovjek-stroj (HUMANWARE) → komunikacija glasom
→ visualno (ikone...)

Arhitektura računala je vještina (znanost) oblikovanja računala s ciljem ostvarenja zahtjeva korisnika. To se postiže nizom tehnika, postupaka i zahvata u svim hijerarhijskim razinama računala (Hardware, Software, Humanware)

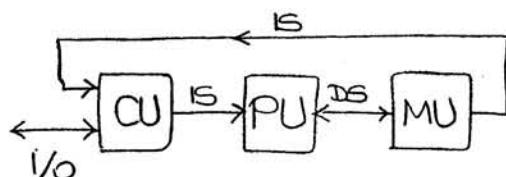
FLYNNova KLASIFIKACIJA ARHITEKTURE:

- SISD [$\vec{A} = (1,1)$] Single Instruction Stream, Single Data Stream
- MISD [$\vec{A} = (M,1)$] Multiple Instruction Stream, Single Data Stream
- SIMD [$\vec{A} = (1,M)$] Single Instruction stream, Multiple Data Stream
- MIMD [$\vec{A} = (M,M)$] Multiple Instruction Stream Multiple Data Stream

Händlerova klasifikacija:



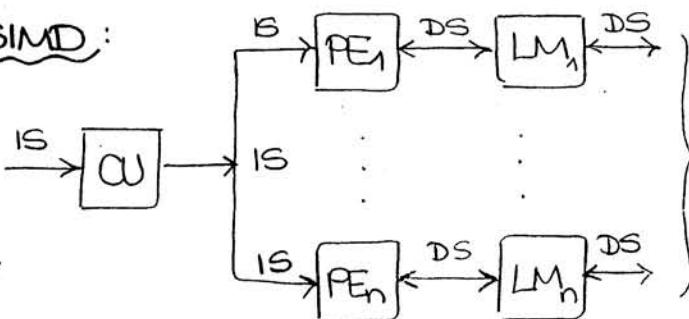
SISD:



Control Unit - CU
Processor - PU
Memory - MU

IS - Instruction Stream
DS - Data Stream
PE - Process Elements
LM - Local Memory

SIMD:



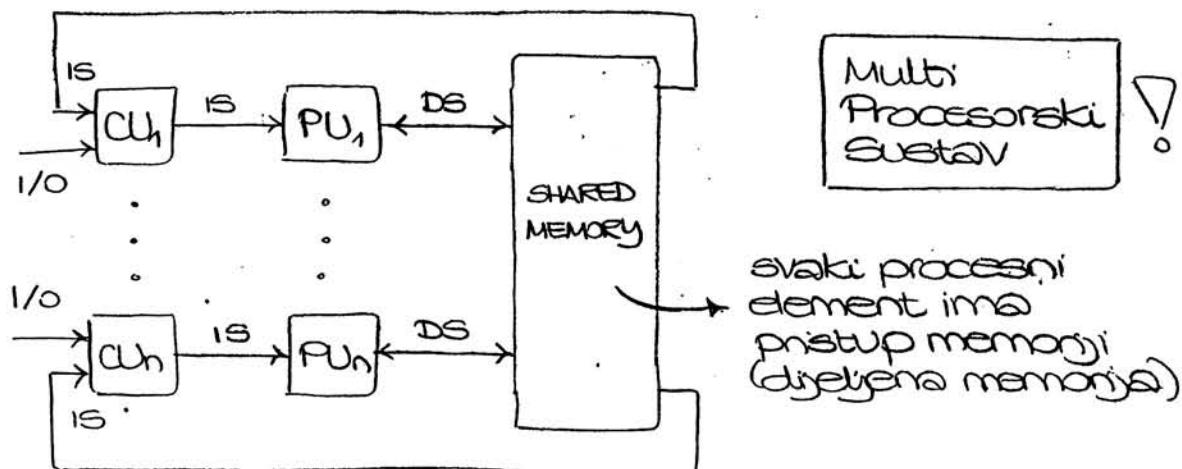
DATA SETS (moduli)
LOADED FROM HOST

SIMD → pogodne za sisteme koji obrađuju mnogo međusobno nezavisnih informacija, operacije nad matricama (ARRAY ARCHITECTURE)

→ n-dimenzionalno uređenje procesnih elemenata (PE)

Tok podataka ima mogućnost dostavljanja podataka natrag u računalo-domadiz (na kraju obrade)

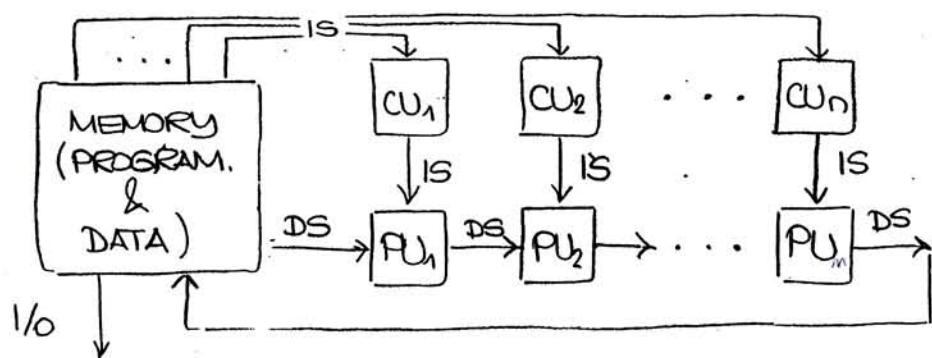
MIMD



- višestručenje SISD-a

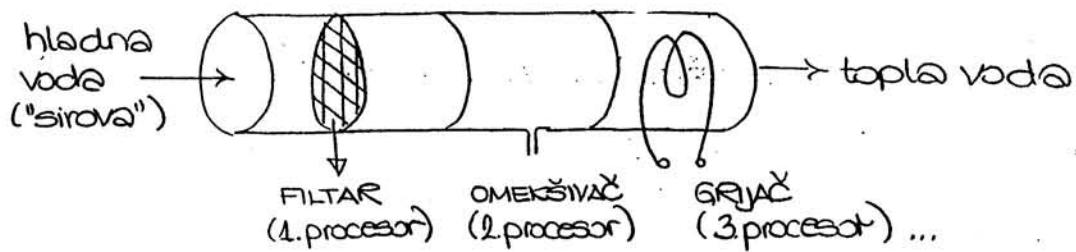
MISD

- strogo gledano, fizički se ne može realizirati
(više instrukcija istodobno "napada" isti podatak)



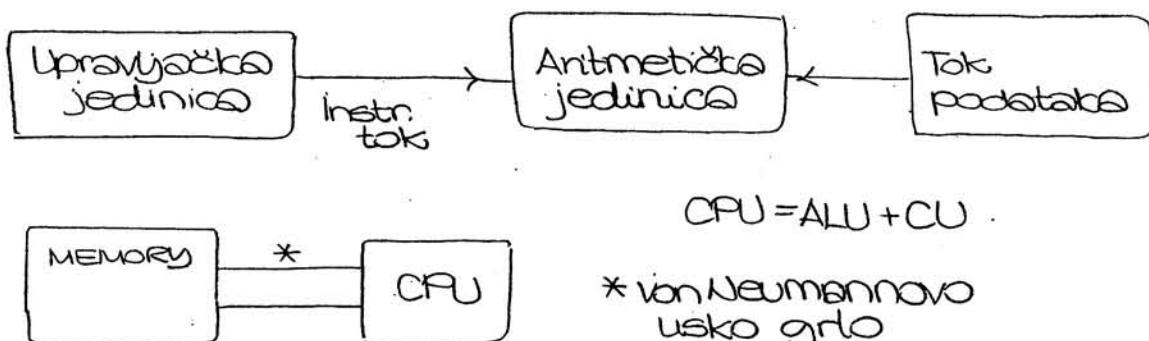
- protočna (PIPELINE) arhitektura svrstava se u MISD klasiifikaciju arhitektura.
- podaci (1 tokom) struje kroz protočne jedinice koje (svakou) modificiraju taj podatak
- zajednički koncept i za RISC i CISC arhitekturu rač.

Analogija: ① grijanje vode (koncept protodnosti)



② proizvodna vraca (u tvornicama) - razlaganje na elementarne radnje
⇒ REZULTAT: ubrzanje rada ??

SISD - von Neumannovo računalo

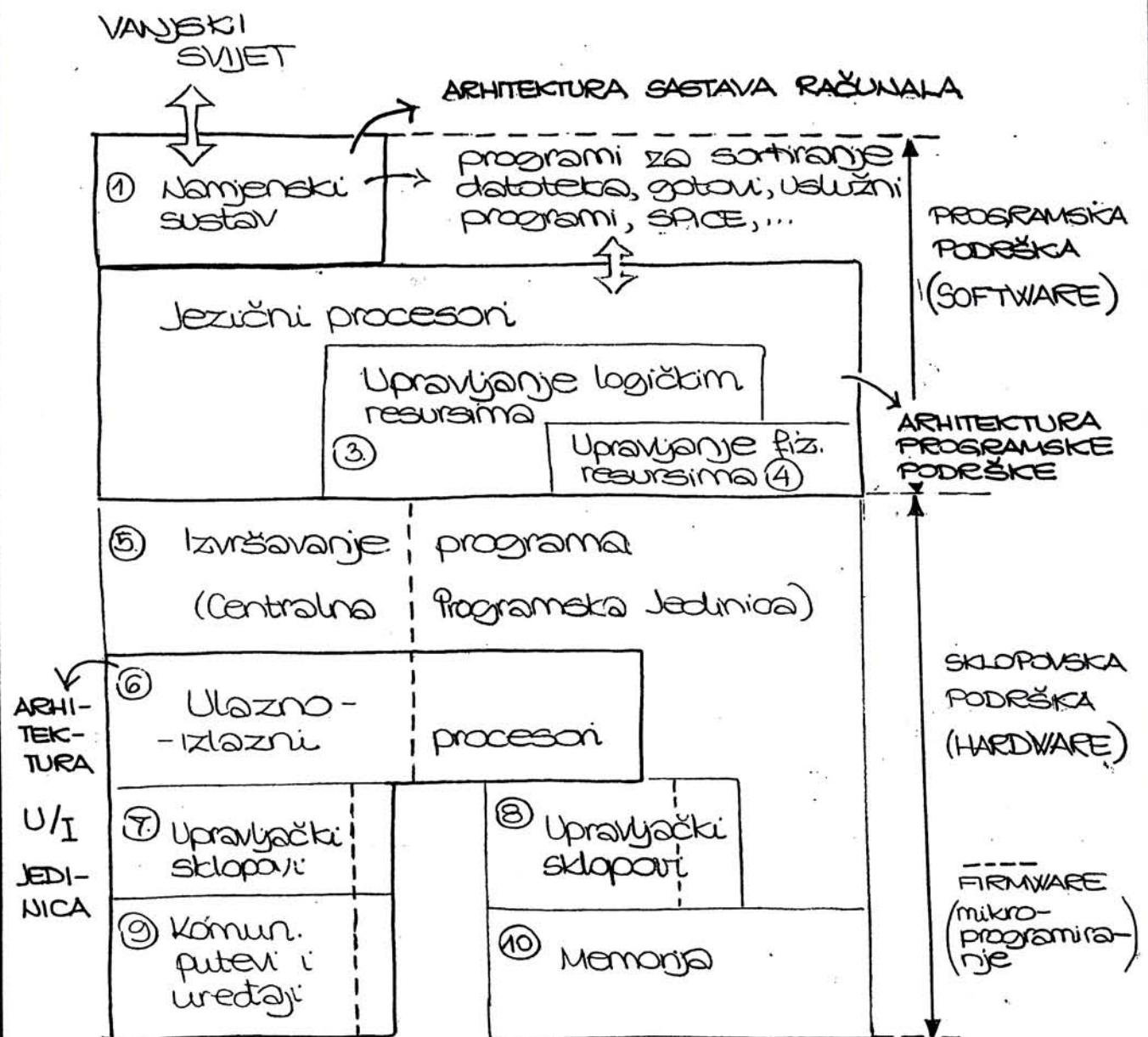


BACKUSOV MODEL von Neumannovog računala:

-smatra da mora postojati drugačiji, bolji način komunikacije $M \leftrightarrow CPU$ od stalnog prenosa (strago sekvenčijalni) inf. $M \leftrightarrow CPU$ koji: gubi prirodni paralelizam, zahtjeva (problema) i time usporava rješavanje problema

SEKVENCIJALNI PRENOS: ne mogu se istodobno prenositi instrukcije i podaci

ARHITEKTURA = distribucija funkcija po razinama



Logički resursi: baze, podataka, datoteke, virtualna memorija, postupci obrade u mrežama računala

Fizički resursi: prim. i sekundarna memorija, vježme procesora, O/I uređaji

Daniel Tabak, 1987. :

ARHITEKTURA → slika računalског sustava je predodžba koju programer stiče dok programira u strognom (zbirnom) jeziku i / ili koju shće pisac prevodioča (COMPILES WRITER)

G.J. Myers 1982:

procesor :

- skup registara
- začetnica / status-registar
- instrukcije / skup instrukcija
- tipovi i formati podataka

ORGANIZACIJA → podrazumijeva detalje vezane za

- konfiguraciju sustava
- međupovezivanje podsustava
(ALU, CU, I-O podustav, ...)
- sabirnički podsustav, ...

REALIZACIJA → (još niža razina)

detalji o sklopovskim strukturama i komponentama kao osnovi za izgradnju sustava, o tehnologiji VLSI, materijalima ...

Von Neumannov MODEL RAČUNALA

Jedan od najznačajnijih radova na području AR:

/ "Uvodna rasprava o logičkom oblikovanju elektronskog računalskog uređaja" /

A.W. Burks, H.H. Goldstein, J. von Neumann,
"Preliminary Discussion of the Logical Design of an Electronic Computing Instrument", 1946. g.

Zahvati koji su poslužili kao ishodište za određivanje arhitekture računala:

- računalo treba imati opalu namjeru (potpuno automatsko izvođenje programa)
- računalo mora pored podataka za računanje pohranjivati metadreseliste i rezultate računanja
- računalo mora imati mogućnost pohranjivanja sljedeća instrukcija (programa),

GENERAL PURPOSE STORED PROGRAM COMPUTER

(računalo opće namjene s pohranjivanjem programa)

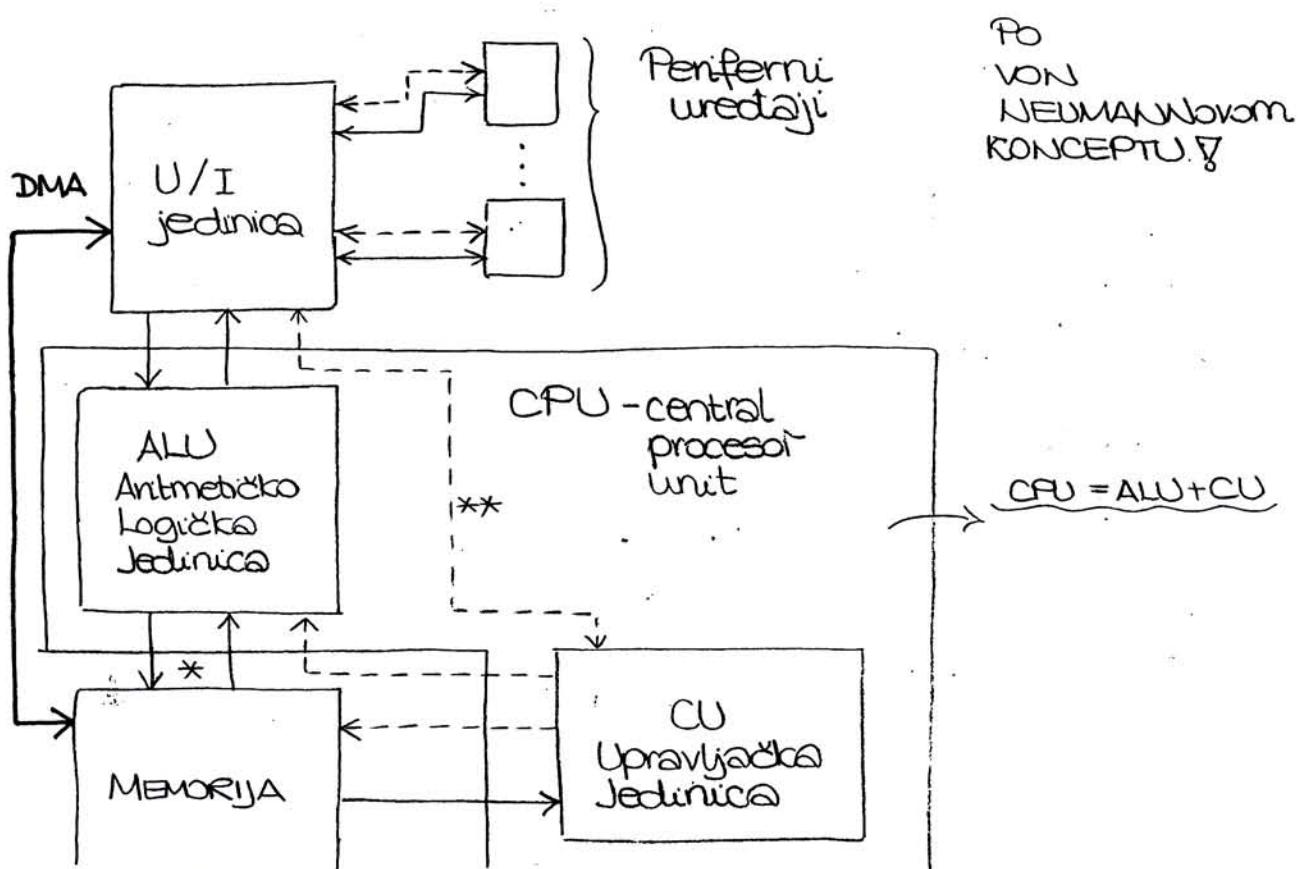
① Instrukcije su u računalu svedene na NUMERIČKI KOD tako da se podaci i instrukcije pohranjuju na jednak način u istoj jedinici (MEMORIJA)

- ② Računalo (stroj za računanje) mora imati jedinicu za izvršavanje osnovnih aritmetičkih operacija (ARITMETIČKA JEDINICA)
- ③ Jedinica koja "razumije" instrukcije i upravlja sljedom izvođenja instrukcija (UPRAVЉАЧКА JEDINICA)
- ④ Računalo mora imati mogućnost komunikacije s raznim sistemom (ULAZNO-IZLAZNA JEDINICA)

Tako dobijamo 5 osnovnih jedinica računala:

1. MEMORIJA
2. ARITMETIČKA JEDINICA
3. UPRAVЉАЧКА - " -
4. ULAZNA JEDINICA
5. IZLAZNA - " -

ULAZNO-IZLAZNA JEDINICA



\longleftrightarrow upravljanje

\longrightarrow tok podataka

DMA - DIRECT MEMORY ACES
(paralelan rad ALU i prijenosa podataka \Rightarrow brži RAD?)

* pohranjivanje
medurezultata
i rezultata,
i SAM PROGRAM
(njegov operacijski kod)

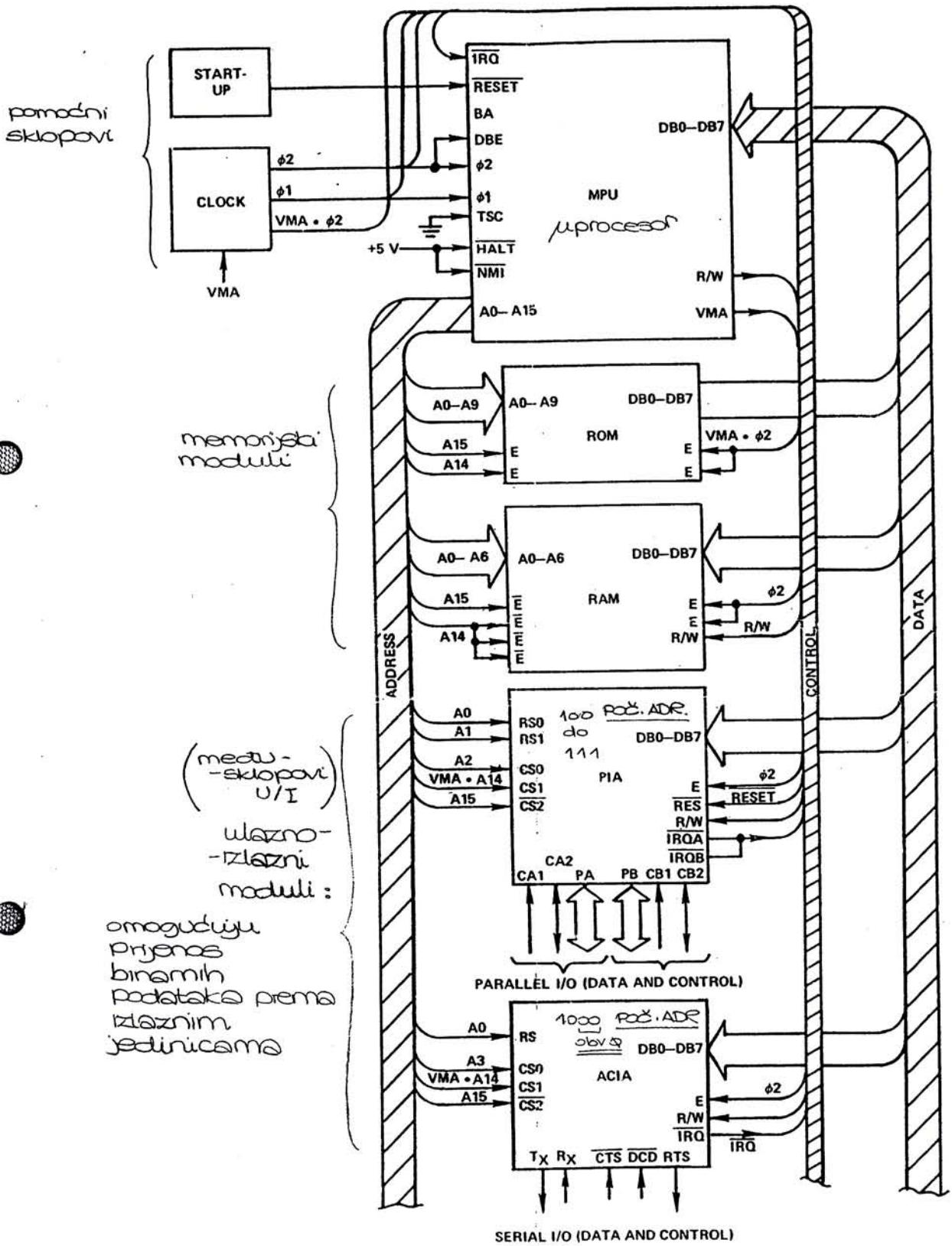


FIGURE 1-1.2-1. MPU Minimum System

Samo je jedan Mali Ivica!

** konznicici su zahtjevali INTERRUPT REQUEST $U/I \rightarrow CU$
 (nije bilo lsmjernih veza upravljanja
 izmedu U/I i CU u prvenstvenom von Neumanovom
 konceptu.)

SLIKA: MPU Minimum Sistem
 (fotokopija)

MPU \rightarrow Micro Processor Unit \Rightarrow ALU + CU
 \hookrightarrow (referentna točka sustava) (fizičko objedinjenje)

③ ROM, RAM \Rightarrow Memory Modul (Read Only M. & Random Acces M.)
 \hookrightarrow omogućava i čitanje i pisanje

④ PIA \rightarrow Parallel Interface } ULAZNO - IZLAZNI
 ⑤ ACIA Serial Interface } podsistav

\rightarrow 5 osnovnih jedinica !!

- Sklop za uključivanje (START-UP & RESET), //
- Generator signala vremenskog vodenja (takta) - CLOCK,
 - kristalno stabiliziran
 - definira diskretne vremenske trenutke
- Sabirnički sustav (BUS):
 - fizička povezanost jedinica

DB_Q \div DB_T (Data Bus - sabirnica od 8 bita $\xrightarrow{\text{Q-LSB}}$ $\xleftarrow{\text{T-MSB}}$)

CB (Control Bus - skup upravljačkih signala)

→ READ / WRITE	: signal (za memoriju)
→ RESET	: (za U/I jedinice)
→ ϕ_2	: signal vrem. vodenja
→ VMA	: "adresa koja je na adresnoj sabirnici je valjana (VALID MEM. ADDRESS)

AB_Q-AB₁₅ (Address Bus - JEDNOSMJERNA sabirnica od 16 bita
 - nijome upravlja isključivo procesor !?)

RAM:

ENABLE	($E=0 : E=1$ za aktivaciju pojedinih modula)
A _Q \div A ₆	(adresni priključci \Rightarrow za interni izbor mem. lokacija $\uparrow\downarrow$ ($2^7 = 128$ nješči))
D _Q \div D _T	(kapacitet je 128 nješči duljine 8 bita)

SABIRNICA - skup šica koje omogućavaju prijenos podataka (eng. BUS)

DATABUS → lsmjerna

ADDRESS BUS → 1smjerna (ADRESU generira procesor, pa ona izlaze iz CPU)

svaka dohvataljiva jedinica (registra, moduli...)

MORA imati jednoznačno određenu adresu!

Inače:

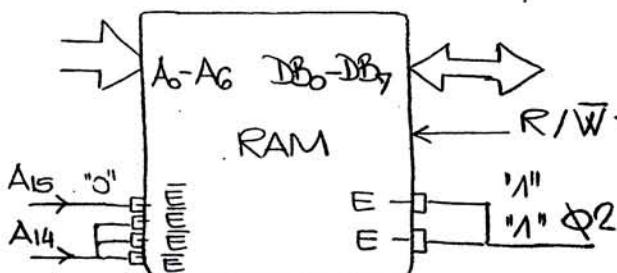
- 1) pogrešan (neodređen) rad (koja će se jed. jasni pri pozivu?)
- 2) moguće izgoranje DRIVERA jedinica (skupovska greška)

(PROGRAMSKA GREŠKA)

ADDRESS BUS: 16 adresnih linija (2^{16} address = 64 kB)
(standard kod 8bitnih registara)

RAM:

Naredba Signal
INPUT → LD A,B R=1 (W=0)
OUTPUT → STORE B R=0 (W=1)



ovom linijom upravlja procesorska jedinica
 $R=0 \rightarrow W=1$ (piši u mem.)
 $R=1 \rightarrow W=0$ (čteš u mem.)

No i PIA ima R/W signal;
jer:
- ovaj procesor konisti
memorijsko U-I preslikavanje*

*procesor ne razlikuje s kojom (RAM, PIA,...) od v.j. komunicira

{ E - enable priključnica (za uključivanje)

Aktivni ulaz je $E = "0"$ (svi ulazi) (do 0.3V)

[Neaktivni - "1" - je $E = "1"$ (od 4.3V-5V)]

E - Aktivni ulaz $E = 1$

→ Za rad RAM-a: E mora biti "0", a E=1"

DB0-DB7 → minimalan dohvat je 1Byte - ni podatak (8 bita)
A0-A6 → manje značajnih bitova

$2^7 \Rightarrow$ adresa = 128 nječi ($\times 8$ bita)

Dakle je KAPACITET RAM-a : 128×1 byte

Gdje se, u adresirivom prostoru od $2^{16} = 64$ K ($K = 2^{10} = 1024$) nalaze 128 adresa RAM-a (odn. nječi RAM-a)?

A₁₅ (adr. linija) je dovedena na prvi E: UVJET: A₁₅ = "0"

A₁₄ (- " -) je dovedena na ostale E: UVJET: A₁₄ = "0"

A₆-A₀ (- " -) : promjene nisu važne; ne služe,

za ozivljavanje chipsa RAM-a, ved da odredi adresu podatka nakon ozivlj. RAM-a.

ϕ_2 mora biti 1 da bi RAM bio aktivan }
 ϕ_2 je jedan od ritmova signala clock-a } RAM ima
 diskretnu
 vremeniku
 komponentu

 RAM "širi" u ritmu signala vodenja!

$A_8 - A_{13}$ → nemaju efekta na RAM.

RAM se ne javlja na fizički jedinstvenom
 dijelu memorije (trik NEFORTUNOŠ ADRESNOG DEKODIRANJA)

Dakle: $A_{15} \ A_{14} | A_{13} \ A_{12} \dots A_8 | A_7 \ A_6 \dots A_0$

RAM nije na jednoznačnoj adresi.

Poštaji li neki drugi modul na istoj adresi? !?

ROM: A_{15} mora biti = A_{14} (znači ROM se ne javlja)
 $A_{15} = A_{14} = "1"$ (kad i RAM)

PIA: $CS_0 = "1" = A_2$
 $CS_1 = "1" = A_{14}$ } (PIA se ne javlja niti kad i RAM)
 $CS_2 = "0" = A_{15}$ } niti kad i ROM
 CHIP SELECT

RS_0, RS_1 (ispada da je samo $2^2 = 4$ registra, ali nije)
 REGISTER SELECT [to je trik, no o tome kasnije] ▶

Aktivan RAM	→	$A_{15} \ A_{14} \ A_{13} \ A_{12} \dots A_8 \ A_7 \ A_6 \dots A_2 \ A_1 \ A_0$
Aktivan ROM	→	0 0 x x ... x x x ... x x x
---	→	1 1 x x ... x x x ... x x x
PIA	→	0 1 x x ... x x x ... 1 x x

$A_{15} \ A_{14} \ \dots$ A_2 A_2 $A_1 \ A_0$
 0 1 x x 1 1 → (PIA aktivna) } Je li mogude da
 → (ACIA je aktivna). } de se oba sklopja
 jave istodobno?
 Da, ali...

Dakle ⇒ PROGRAMER mora paziti da ne adresira ovatu kombinaciju (ako sklop ne bi izbio, desila bi se programatska greska)

Adresa: 0100, Don't Care $A_3 A_2 A_1 A_0$
 HEX: 4 0 0 4 → početna adresa PIA-e
 4 0 0 5
 4 0 0 6 → adresni prostor PIA-e
 4 0 0 7
 4 0 0 8 → poč. adresa ACIA-e
 4 0 0 9 → adresni prostor ACIA-e
 ALI PROGRAMER MORA OSIGURATI $A_7 = A_6 = "0"$

Sbženost adresiranja je problem minimalne konfiguracije računala (jer konisti NEOPTRUNO ADR.DECOD.) i uvelike ovisi o programeru.

IRQ (Interrupt Request) → izlazna linija

generira je ACIA, PIA

↳ linija za zahtjevanje prekida (aktivna u "0")

CPU → obradom zaključi hode li (ili ne) odobrili zahtjevu (ovisno o prioritetu v.j. koja je generirala signal)

RESET : signal za prekid (najvišeg prioriteta)

Aritmetska jedinica: sklopovi za izvršavanje osn. aritm. op.
+ registri za privremenu pohranu podataka

BROJEVNI SUSTAV?

Kandidati:

→ dekadski s. (prirodan čovjek)

→ binarni s. (lakša realizacija sklopa
optimalan bus. je između 2:3)

IZABRAN BINARNI SUSTAV:

Razlog:

Pored lakše tehničke izvedbe i ekonomičnosti predstavlja-
ja brojeva, razlog je i to što računalo:

"... nije samo aritmetski stroj, već je po svojoj
prirodi i logički. Log. sustavi su sustavi DA-NE,
odnosno binarni sustavi." (A.W. Burks et al, 1945.)

(3,7) → prim. brojevi

Neki dekadetski broj možemo predaći kao ostatak
dijeljenja broja s (3,7)

(Rašpon je od 0 do 20) jer $v\ddot{e}stekr. 3 \cdot 7 = 21$

0 → PRIKAZ → (0,0)

1 (1,1)

2 (2,2)

3 (0,3)

4 (1,4)

5 (2,5)

6 (0,6)

7 (1,0)

8 (2,1)

9 (0,2)

10 (1,3)

11 (2,4)

nije POZICIJSKA NOTACIJA - br. sust.
ved RESIDUUM -"-" - br. sust.

4
+ 7

11

CARRY
(zbez poz. notacije)

$$\begin{array}{r} (1,4) \\ + (1,0) \\ \hline (2,4) \end{array}$$

nema CARRYja
(brže zbrajanje)
UBRZANJE RADA ALU
U RES. NOTACIJI

ODNOS: ARITMETIČKE OP. \Leftrightarrow LOGIČKE OPERACIJE

ALU:

Alu-sklopovi u prvom von Neumannovom računalu bili su zbrajalo (FULL ADDER) i sklop za posmaku (SHIFTER).

- oduzimanje se izvršavalo pribrajanjem 2' kompl.
- oper. množenja i dijeljenja (izvedene pod programskim upravljanjem) uzastopnim zbrajanjem, odn. oduzimanjem i posmakom.

OPERANDI von Neumannovog računala = 40 bita DULJINE
šta?

- stroj su dizajnirali matematičari
 - osn. zadaća računala je rešavanje dif. jednadžbi
- 40 bita omogućava preciznost računanja na 12 decimala ($2^{-40} = 0.9 \cdot 10^{-12}$), odn. 12 znamenski.

(Slika iz knjige) CPU = ALU + CU

ALU \rightarrow (FULL ADDER i SHIFTER)

* A, B \rightarrow registri (akumulatori)

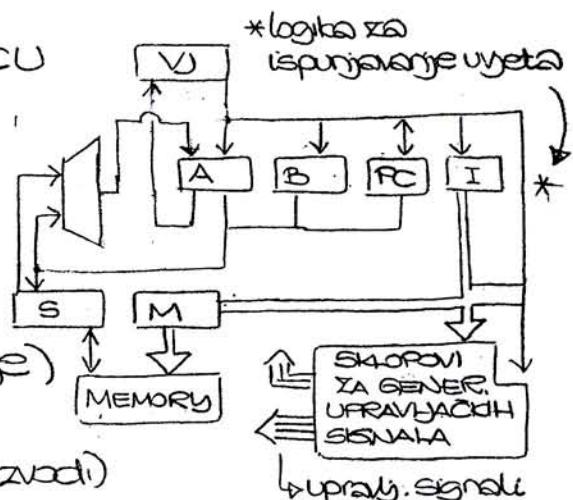
A - vedi prioritet i raspored naredbi, ali uglavnom ravноправni A i B

PC \rightarrow Programm Counter

(adresa sljedeće instrukcije)

I \rightarrow Instrukcijski registar

(pohranjen operac. kod
(instrukcije koja se upravo izvodi))



S - memorijski reg. podataka } sučeljni registri
M - - " - adresni registar } prema memoriju

PC : control counter (zapo se CC) \rightarrow bio je duljine 13 bita

I : functional table reg. (FR)

$$2^{12} = 4K = (4096 \text{ 40 bitnih nječi})$$

1 bit za izbor lijeve ili desne instrukcije

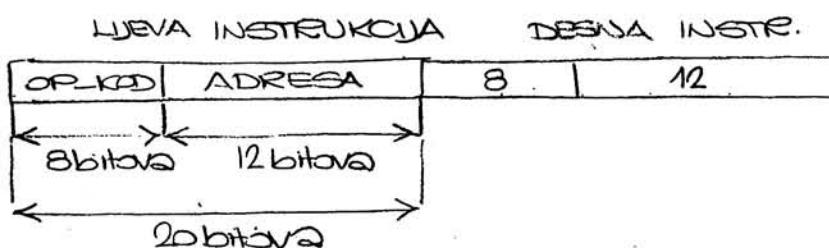
u 40 bita su smješteni 2 instrukcije (više od 20 bita im nije trebalo za instrukciju - bilo koju), pa im je 13. bit određivao koju od 2 instrukcije treba procesor obraditi.

Upravljačka jedinica: daje sve potrebne upravljačke signale za vrem. vodenje i upravljanje ostalim jedinicama računala

- "Svaki korak algoritma predstavljen je 1 ili sljedom instrukcija. Instrukcije def. elementarne operacije koje računalno može izvesti."
- "...postoje kodovi koji in abstracto odgovaraju upravljanju i prouzrokuju izvršavanje nekog sljeda operacija koje su pojedinačno raspoložive u računalu i koje su u svogoj celovitosti razumljive onom koji postavlja problem."
- (A.W. Burks, 1946.)

→ sličnost von Neumannovog rač. s Turingovim strojem

Duljinu nječi od 40 bita smjestiti u 2 instrukcije (LJEVA i DESNA INSTRUKCIJA)



$$2^8 = 256 \quad (\text{broj različitih instrukcija})$$

$$2^{12} = 4096 = 4\text{K} \quad (\text{adresnih lokacija u memoriji})$$

Jednoadresni format instrukcije:

svi dvoadresni (dvoregisterski) operacije
idu preko akumulatora, koji je centralni,
najznačajniji register (opće namjene je)

=AKUMULATORSKI ORIENTIRAN PROSEZOR

$$A = f(A, M) \quad \text{U Akumulatoru je 1 operand}$$

(2. operand je u memoriji), i rješenje.

1. operand je nakon operacije bit nepovratno
izgubljen (prebrisan rezultatom).

Dvoadresni f.i.: nječi bi morala biti duga $8+12+12 = 32$ nječi
(2 instr. ne stanu u 40).

SKUP INSTRUKCIJA:

- AL instrukcije (aritm.-logičke)
- Inst. za prenos podataka
- Inst. za uvjetno / bezuvjetno grananje
- Ulazne / izlazne (U/I) instr.
- Instr. s djelomičnom zamjenom*
- (partial substitution)

*djeluju na 2 instrukcije i menjaju sadržaj u adresnom polju instrukcije (onih 12 bitova)

* današnji princip VIRUSA

Zamisao je bila ušteda memorije \Rightarrow umjesto 2 programa samo je jedan + instrukcije za zamjenu.

LOŠE:

\Rightarrow rizik od nekontroliranih modifikacija programa
(loš rad \Rightarrow virus)

SHEMA IZVODENJA PROGRAMA: (instrukcije)



PRIJAM :

1. korak: MEM (PC) \rightarrow I
2. korak: PC + 1 \rightarrow PC
3. korak: Dekodiranje op. kod.
instrukcije

CPU saznaće:

- o kojoj instrukciji je riječ?
- koje upravljačke signale treba generirati?

IZVRŠI :

4. korak
5. korak
- :

\hookrightarrow broj koraka u fazi izvrši direktno ovisi o složenosti programa (instrukcije)

Numenički kod se izvršava kao INSTRUKCIJA ili
PODATAK ovisno je li CPU u fazi PRIJAM ili IZVRŠI.

Memorija von Neumannovog računala

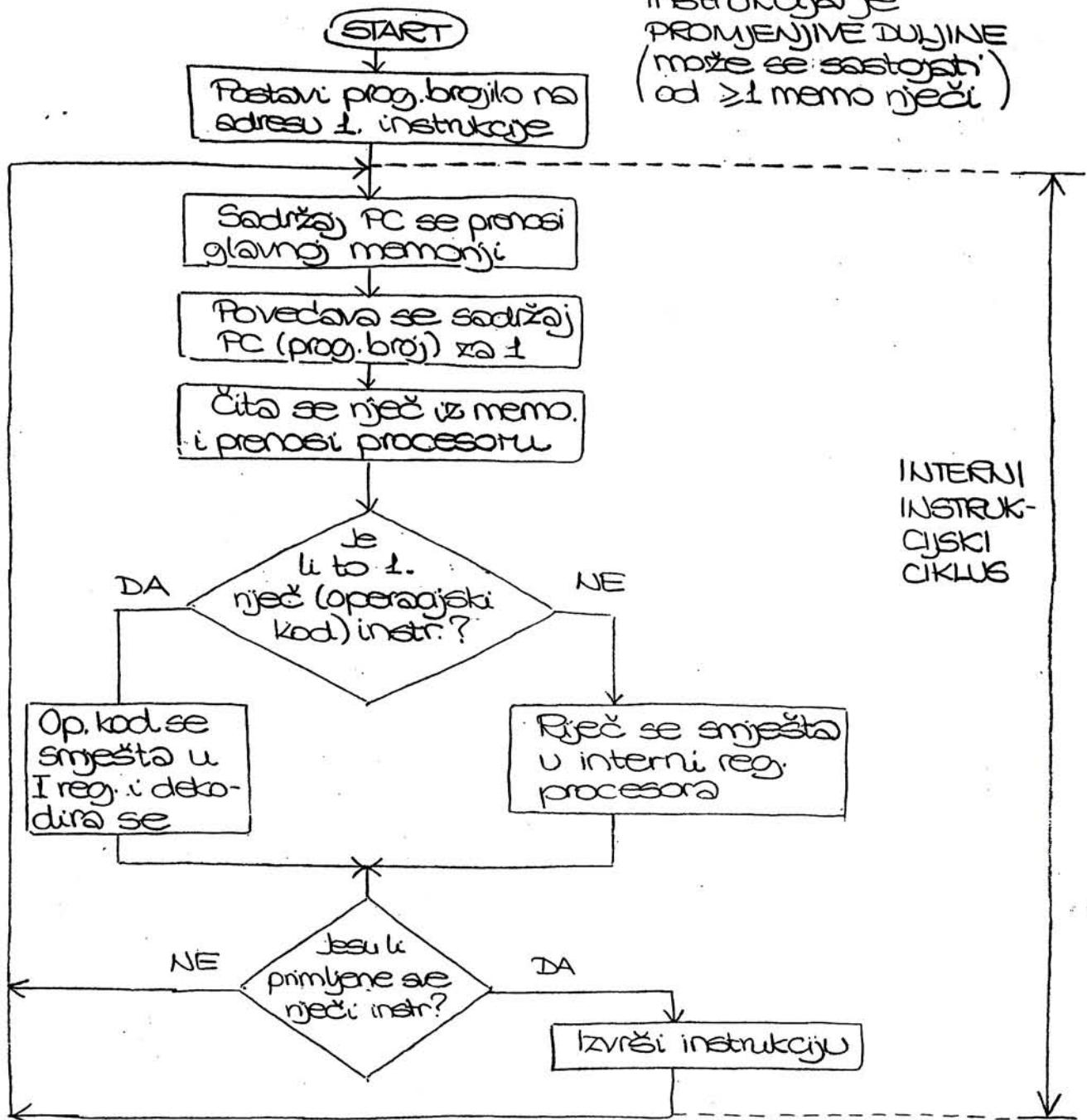
$4096 \times 40 = 163\ 840$. bistabilnih elemenata
realiziranih elektronskim cjevima
(1946.g. ??)

Rješenje: katodna cjev za memoriranje ELECTRON
(Princeton Lab, tvrtka RCA)

40 selectrona: svaki kapaciteta 4096 bita (4K)
vrijeme pristupa $\sim 50\ \mu s$

Von Neumann, Burks i Goldstine razmatraju
hijerarhijsku organizaciju memorije:

- 1.) primarna ili radna memorija (realizirana selektromima)
- 2.) sekundarna memorija (svjetlosno osjetljiv film,
magn. osjetljiva traka ili
mag. žica)
 - \hookrightarrow pod izravnom
kontrolom računala
- 3.) mrtva memorija (pohrana pod. koji se trenutno ne koristi)
 - \hookrightarrow neintegralni dio računala



PRIMJER:

Adresa	Sadržaj memorije	16 bitna adresa
PC → 0100 _H	86	
↓ 0101	03	
↓ 0102	BB	
PC → 0103	86	STA A \$ 03BB
0104	2F	LDA A # \$ 2F

$$(PC) = 0100H$$

U fazi pričekavljivanja PC se (u naredbi STA A) povedao 3 puta

Zamislimo:

Neka (PC) = 0101_H (inicijalno)

PRIJAVA: on pričekavlja 03 i smješta ga u I register.
Sad je 03 shvađen kao instrukcija (op. kod instr.)
a ne kao prvi 8bita adrese podataka?

Tj: Hode li sadržaj memorije biti shvađen kao operacijski kod instrukcije ili kao "goli" podatak
vopće ne ovisi o samom op. kodu, već o trenutnom stanju kontrolnog brojila (PC)
odn. procesne jedinice (FAZA PRIJAVA → kod instr.
-||- IZVRŠI → podatak)

STA A \$ 03BB :

86
03
BB

čim se dekodira op. kod,
procesor (Uprav.Jed) zna
da slijede još točno 2 riječi pod.

PRIJAVA:

86 je u fazi PRIJAVA smješten u instrucijski reg. i dekodiraju se
03 je u fazi PRIJAVA smješten u interni reg. procesora

BB - || - - || - - || - - || - - || - - || - -

Tek sad su primljene sve riječi instrukcije, pa
se IZVRŠAVA instrukcija. (FAZA PRIJAVA je dovršena)

IZVRŠI: sve što se od sada dohvata iz memorije
je "goli" podatak ??

U入azno-izlazna jedinica:

У граfičku izlaznu jedinicu (opet) se koristi gjev
SELEKTRONA:

svjetlo polje → "1" (Preteča graf. U/I
tamno polje → "0" (jedinice visoke rezolucije)

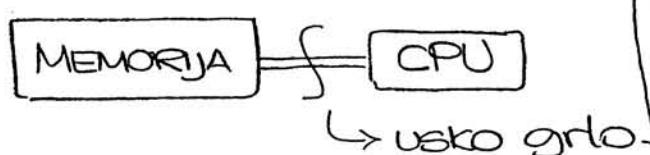
Teleprinter s pomoćnom šiřčnom memorijom
koristio se (također) kao U/I jedinica.

RAČUNALO : jednokonističko (single-used
oriented)

Razmatra se mogućnost SIMULTANOG DJELOVANJA
U/I JEDINICE i CPU
(odustalo se zbog tehnoloških razloga)
(Kasnije je ostvareno DMA!)

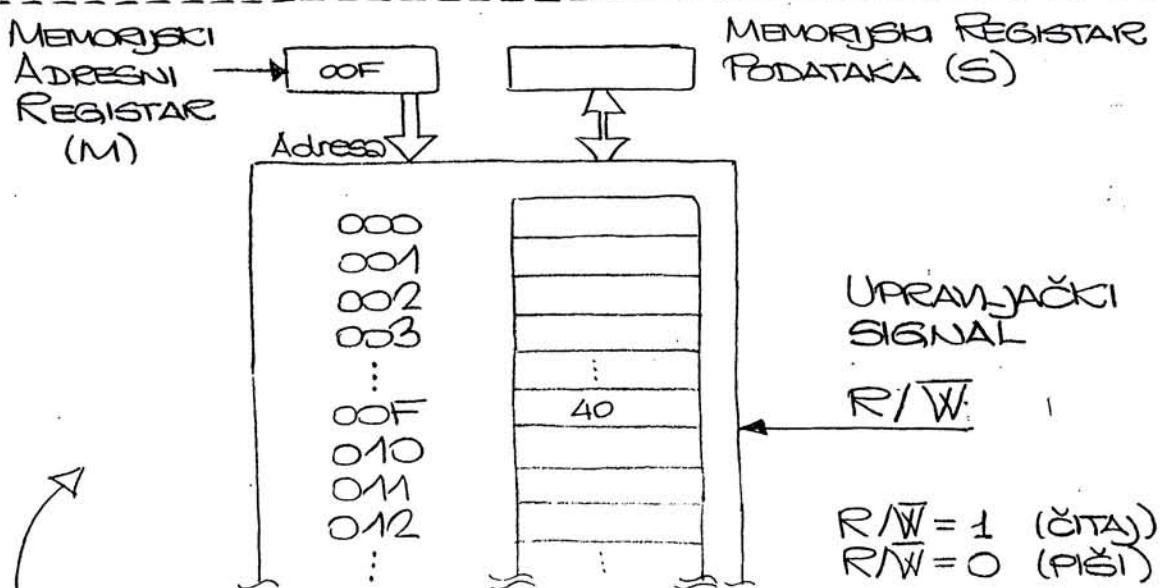
Sve se obavlja preko A : Akumulator \xleftarrow{U} \xrightarrow{I}

Zašto SISD?



u 1 instrukciji min.
1 operacija čitanja
bjesomučna razmjena
podataka samo u
fazi PRIJEM ...

Vremenski je kritičan proces komunikacije MEMO-CPU
(obavezan, jer MEMO nema procesnih elemenata)



Model radne memorije:

- lista polja je konačna
- svakom polju je dodatajena adresa

OPERACIJA ČITANJA: ($R/\bar{W}=1$)

addr. $\rightarrow M$
 $R/\bar{W} \rightarrow 1$

\rightarrow u M je adresa traženog pod.
 \rightarrow poslon je signal "čitaj"

„vrijeme pristupa memoriji“ (10 nsek ili 100 nsek)

Memorija (addr.) $\rightarrow S$ \rightarrow traženi podatak je pročitan i kopira se u S

\Rightarrow Nedestruktivna operacija (podatak nakon čitanja ostaje netaknut u memoriji)

Npr. $00F \rightarrow M$
 $R/\bar{W} \rightarrow 1$
„
 $40 \rightarrow S$

OPERACIJA PISANJA: ($R/\bar{W}=0$)

data $\rightarrow S$, addr. $\rightarrow M$
 $R/\bar{W} \rightarrow 0$

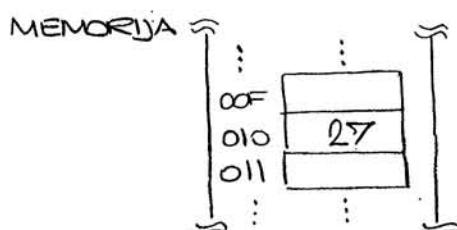
„vrijeme pristupa memoriji“

(S) \rightarrow Memorija (addr.) \rightarrow skup sadržaja S se smješta u memoriju na adresi zadanoj u M

Dakle je:

- za S: operacija čitanja
- za MEMO: oper. pisanja

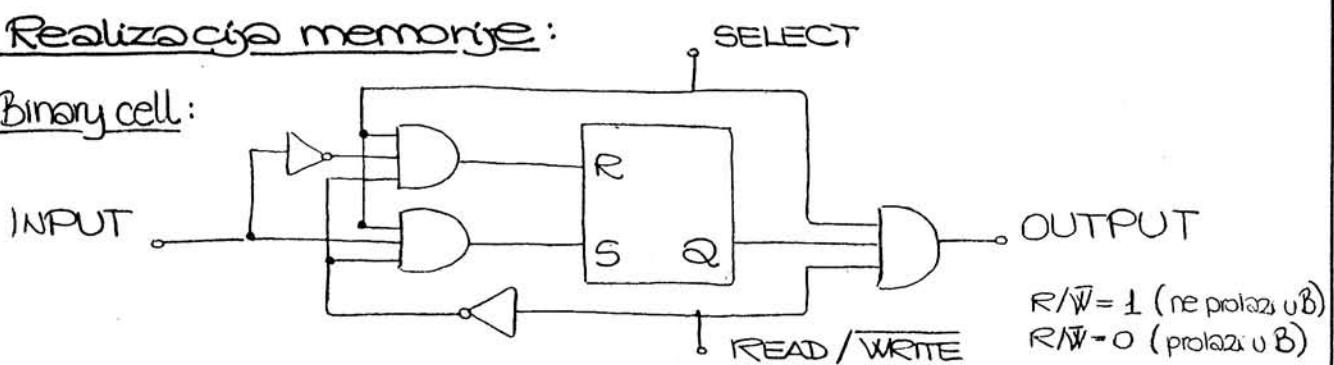
Npr. $27 \rightarrow S$, $010 \rightarrow M$
 $R/\bar{W} \rightarrow 0$



Sadržaj S je ostao nepromjenjen,
 \Rightarrow Destruktivna operacija
 (prethodni zapis u memo na adr. 010 je IZGUBLJEN)

Realizacija memorije:

Binary cell:

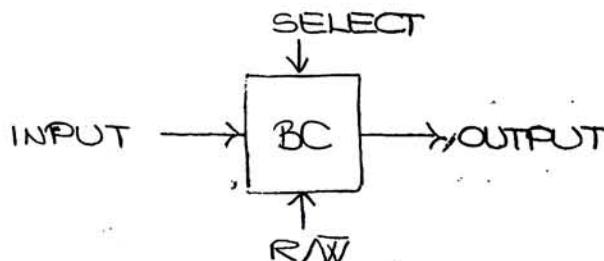


$R/\bar{W}=1$ (ne prolazi u B)
 $R/\bar{W}=0$ (prolazi u B)

Samo je jedan Mali Ivica!

ULAZ SR	00	01	10	11	
stanje Q	0	0	0	1	?
	1	1	0	1	

NEDOVOLJENO
STANJE ??
(Digitalna Elektron.)



ČITAL:

ako je SELECT = 0
sklop je immobiliziran

(Pretpost.) R/W = 1
SELECT = 1 → (oznaka "IZABRANE ČEVIJE") - izlaz iz adres. decoder-a

R/W (invert) = Q ⇒ oba sklopa su Q ⇒ Q ostaje u svom stanju!
I sklop na izlazu je, međutim, SLIKA stanja Q //

PISI :

R/W = Q

SELECT = 1

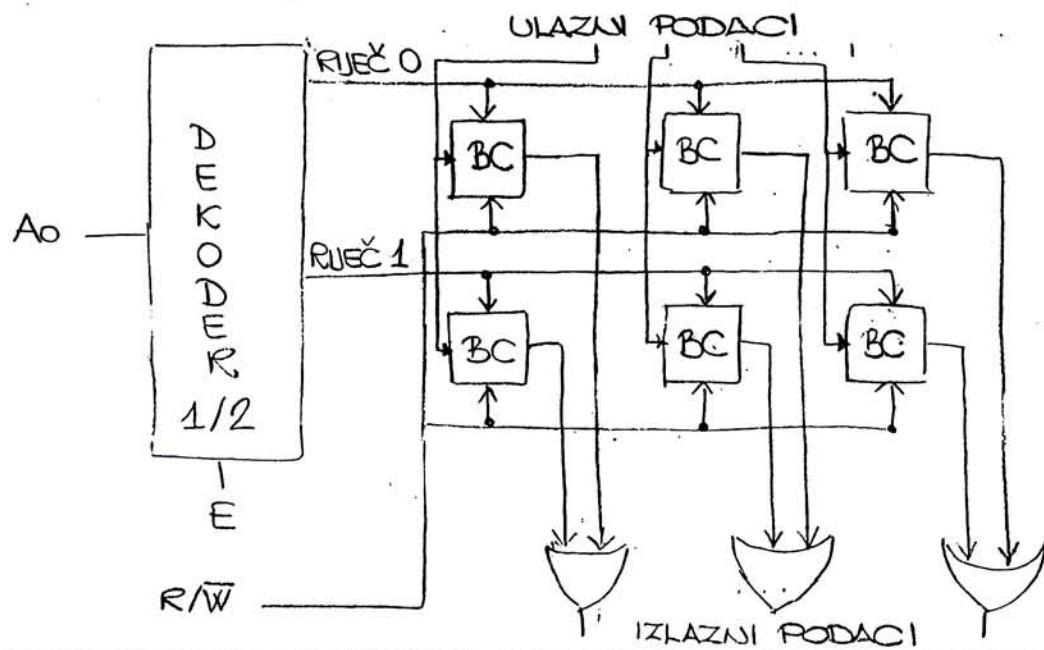
INPUT = Q (želimo upisati Q)

(Pretp.) Q = 1 (odn. Q)

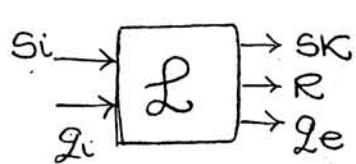
1. I sklop (S) S = (SEL) ∧ (R/W) ∧ (INPUT) = 0

2. I sklop (R) R = (SEL) ∧ (R/W) ∧ (INPUT) = 1

Q prelazi iz 1 u Q (odnosno ostaje u Q)



ALGORITAM OBRADE PODATAKA

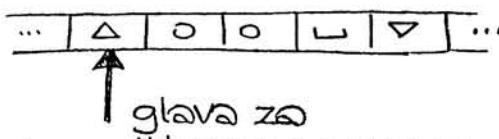


↳ određen izvedbom upravljačke naprave u TS

↳ određen slijedom instrukcija u memoriji u von Neumann rač

MEMORIJSKA JEDINICA

TS: beskonačna traka



- ulazni podaci
- merni rezultati
- rezultati

ALU

↳ objedinjena u uprav. napravi

von Neum.:
konačna memorija

Adr. Sadržaj

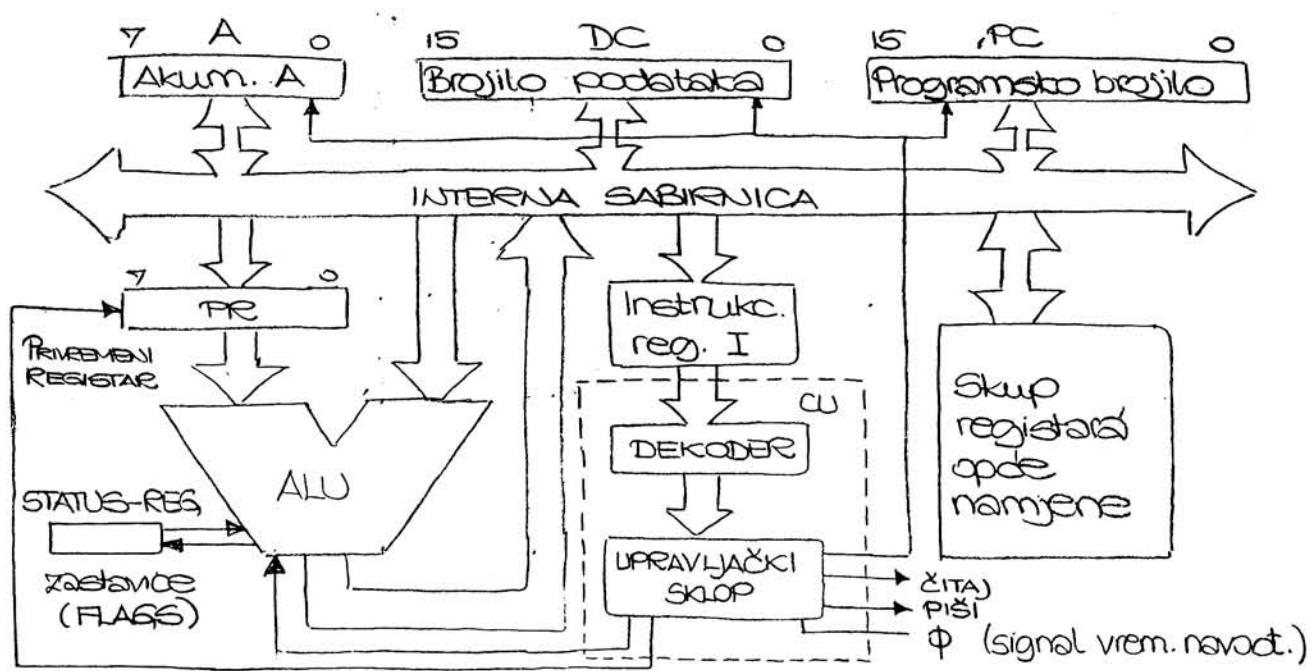
001	
002	
:	⋮
OFF	

* i PROGRAM ??

- posebna jedinica

MODEL MIKROPROCESORA

CPU REALIZIRANA U Large Scale Integration TEHNOLOGIJI



FLAGS → podloga za realizaciju uvjetnih naredbi
(grananja) i processa

PR → privremeni register (korisnik ga ne vidi)
DC → sadrži adresu operanda

Skup reg. opde → privremeno pohranjivanje
namjene rezultata i mrežurezultata da
bi se ubrzao rad procesora
(NIJE to CACHE MEMORIJA)

Φ → clock : signal vremenskog vodjenja
definira takt vrem. izvođenja programa.

PC → 16 bita ($0 - 2^{16} = 64$ k raspon adrese)
↳ kapacitet memorije za
izravno dohvadanje

PRIMJER 1:

Stanje na sabirnici za pogodnostavljeni model μP-a :

INC \$05FF - inkrementiraj sadr. memo. lokacije
s adresom 05FF

	7	0
0100 _H	7C	
0101 _H	05	
0102 _H	FF	
:	:	
05FE _H		
05FF _H	23	
0600 _H		
		:

Memo. nema procesnih
mogućnosti:
Operacija se obavlja u ALU!
μP ne zna što se nalazi u
memo. na lok. 0100...0600

Početni uvjeti: u PC mora biti adresu 1. instrukcije (PC = 0100_H)
u fazi smo PREBAVI (početak je uvek u toj fazi)

Dohvat :

uprav. jed
gen. signal
(memo: čita) (adres) → I
PC + 1 → PC

PC → na internu sabirnicu → adr. sab
(8 bitna je, pa se prenosi)
u 2x: 1. značajniji byte

traži se u
memoriji ta
adresa i stavlja
u I

7C je instrukcija (jer smo u fazi PREBAVI)
(ili njena komponenta)

Sad je: PC = 0101_H I = 7C_H

Dekodiranje: cu saznaće: operacija je INC
instrukcija se sastoji od 3 byte
treba dobiti iz memo još 2 -||-
koji predstavljaju 16 bitnu adresu?

$$7C = 01111100$$

Op. kod se tumači kao: povedaj za 1 vrijednost operanda
daje adresu sadržana u 2 bytea
koja slijede ovom operacijskom kodu

Ponavlja se faza PRIBAVI : $PC \rightarrow \text{adr.sab}$
 $(\text{mem} = \text{čITAJ})$: $\text{adr.sab} \rightarrow DC \leftarrow \mu P$ zna da je NEXT
 $PC+1 \rightarrow PC$ byte PODATAK, pa
ga pohranjuje u DC

$$\begin{array}{ll} \text{Sadje} & \text{PC} = 0102_{11} \\ & I = 7C_H \\ & DC = 05 \end{array}$$

Ponovo: $PC \rightarrow \text{adr. slob}$
 $(\text{adr. slob}) \rightarrow DC$
 $PC + 1 \rightarrow PC$

i končno je: $PC = 0103H$
 $I = 7CH$
 $DC = 05FFH$

(PC sadrži adresu sljedeće instrukcije)
Iz memo. smo dohvatali sva 3 byte oper. koda, μP zaključuje
da je faza PRIJAVI gotova \Rightarrow idemo na fazu IZRŠI

I reg.	Reg: DC (brojilo podataka)
01111100	00000101 11111111
7C	05 FF

Format instrukcije : | OPER.KOD | ADRESA OPERANDA |
(von Neumannov model,)

Neposredno prije faze razrješi poznati su podaci u PC,I,DC
izrješi:

1) Dohvat podatka na adresi (05FF) koja je u DC

DC → no inter. sub. → no addr. sub.

(memo; čita) (adr.s.) → sab. podataka → interna sab. → gdje?

Opravna se smješta u privrem. reg(!!!)-pre

PC se NE MIJENJA (u fazi smo "izvrši") osim ako to ne zahtjeva sam oper.kod (JUMP ili CALL potpr.)

Sadje :

$$PC = 0103H$$

DC-35H-FH
T = 7C

$$PR = \eta B_{11} -$$

PR - 25A

niemo ga smještiti u A jer nam je podatak u A možda potreban, a inc je učinio oper. i nema potrebe za korištenjem A i gubljenjem dotadašnjeg podatka u A.

Izvršava se INC u ALU i rezultat se privremeno vrada u PR
Vrada u rez. u MEMO:

PR → internal sub → sub. podatata

DC → adr.sab (iz DC u adresnu sabirnicu)

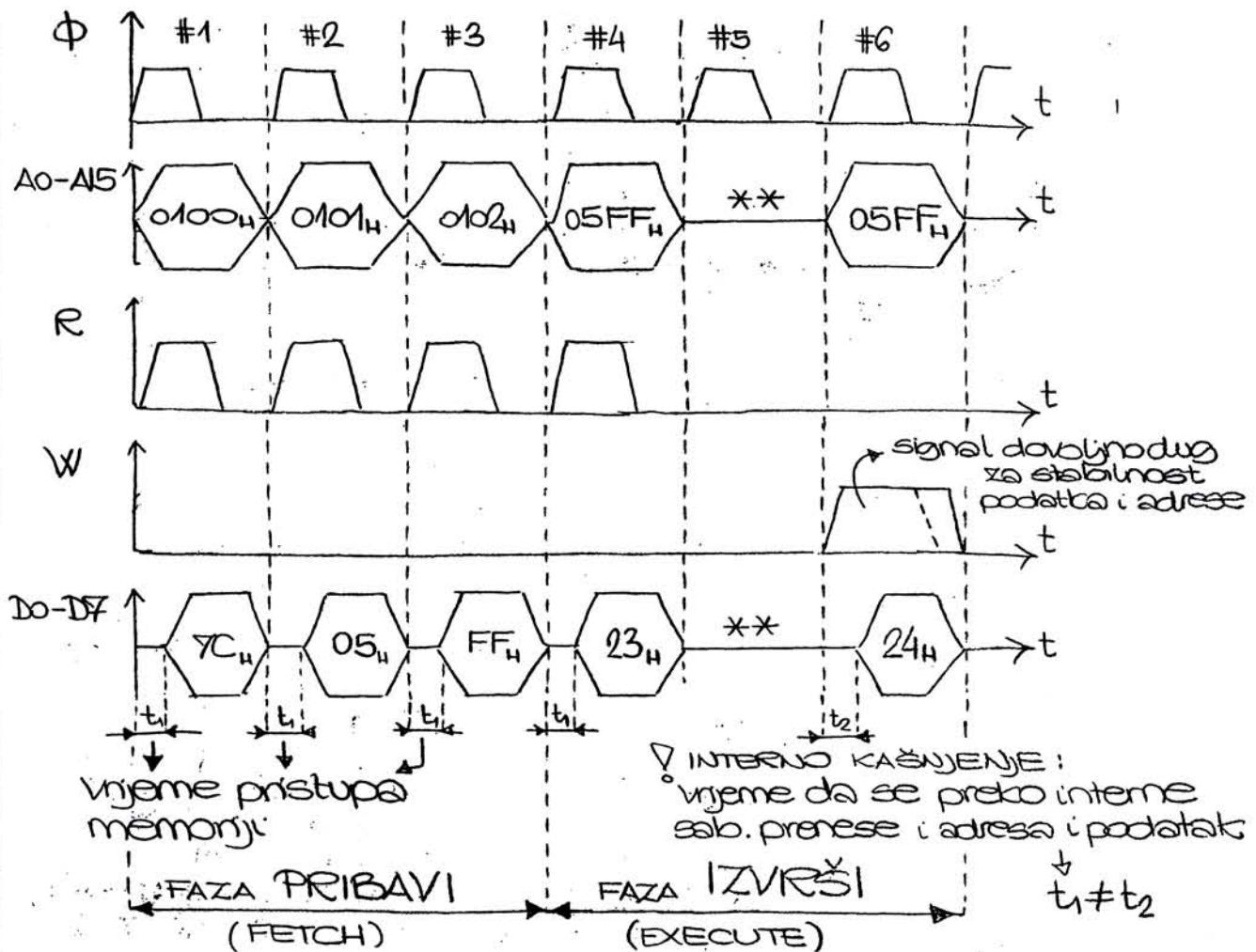
(memo: PIŠI) → PR → (adr; sab)

Faza izvrši završavanje i ponovo smr u fazi PRIBAVI ($PC=0.103_{H_2}$)

PRIBAVI \rightarrow IZRŠI (150 milijuna puta u sec!)

Što se događa na adresi, sablji, sablji podataka i upravljačkoj sabirnici? One su nam dostupne (INTERNE nisu), pa možemo analizirati rad μP i detektirati eventualne pogreške.

Promatramo sablje u vremenu (dobiveno logičkim analizatorom)



** μP se električki odspajao od vanjske sabirnice i na njima nema aktivnosti (izvršava se operacija za koju ne trebamo sabirnice) HIGH IMPEDANCE (HIGH Z)

?? ovde je vrijeme ponudeno U-I modulu i DMA sklopu za "direct-memory-access".

Za vrijeme ** INTERNO (ne vidimo process) se u ALU izvrši 23+1 i generira uprav. signal PIŠI

Vremenski zahtjevna instrukcija! (čak 6 perioda clock-a) Loš

Za 24 ili 32 bitno μP, vrijeme izvođenja je krade jer se u jednoj periodi clocka dohvata cijeli op.kod (FAZA PRIBAVI)

INTERNO KAŠNJENJE:

05FF adresu (16 bit) } sve to prenosi
 24 podatak (8 bit) } interna sab. ??

Interna sabirnica (jedna jedinica) ima 8 bita (usko građena)
 i potrebno je vrijeme da se adr. prenese na A0-A15,
 a podatak na DQ-D7
 (Naravno, da imamo 3 bytevu internu sab, t2 bi bilo 4.)

386, 486 imaju 32bitnu adr. sab. (4GB adresirljivog prostora)
 (nema potrebe za daljnjim širenjem)

PRIMJER 2:

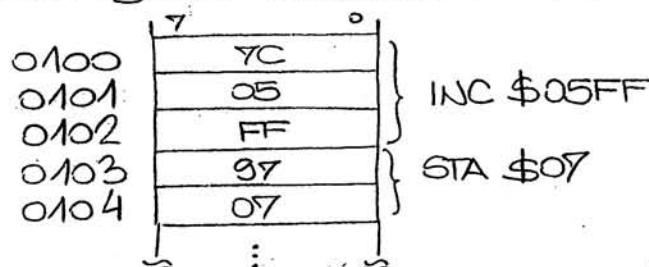
STA \$07 za model procesora

(pohrani sadrž. aturm. A na adresu 0007H)

adresiranje nulte stranice:

kratak način izravnog adresiranja, uštedimo
 na bytu, time i na vremenu.

DC register automatski na 8 važnijih bitova postavljen !



NAPOMENA:

STA \$07

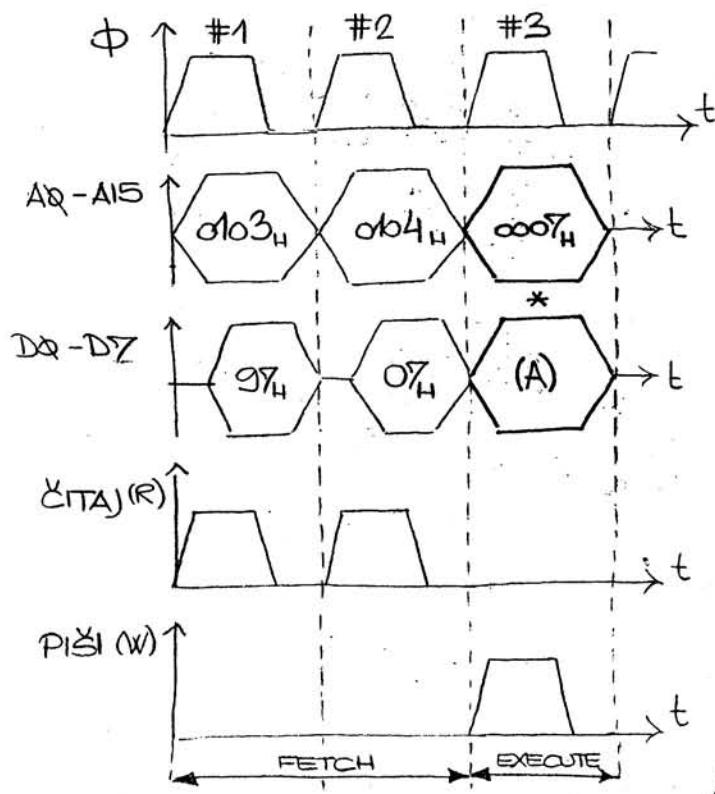
STA \$0745

imaju različite op. kod

STA \$07 → 07.07

STA \$0745 + (npr) 880745

(u op. kodu je NAČIN ADRESIRANJA)



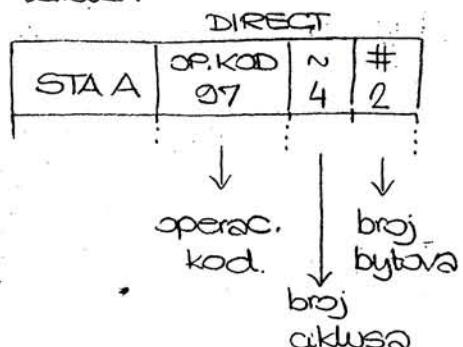
Model zahtjeva 3 perioda takta

Međutim:

stvarni MP

MOTOROLA MC6800

zahtjeva 4 perioda takta!



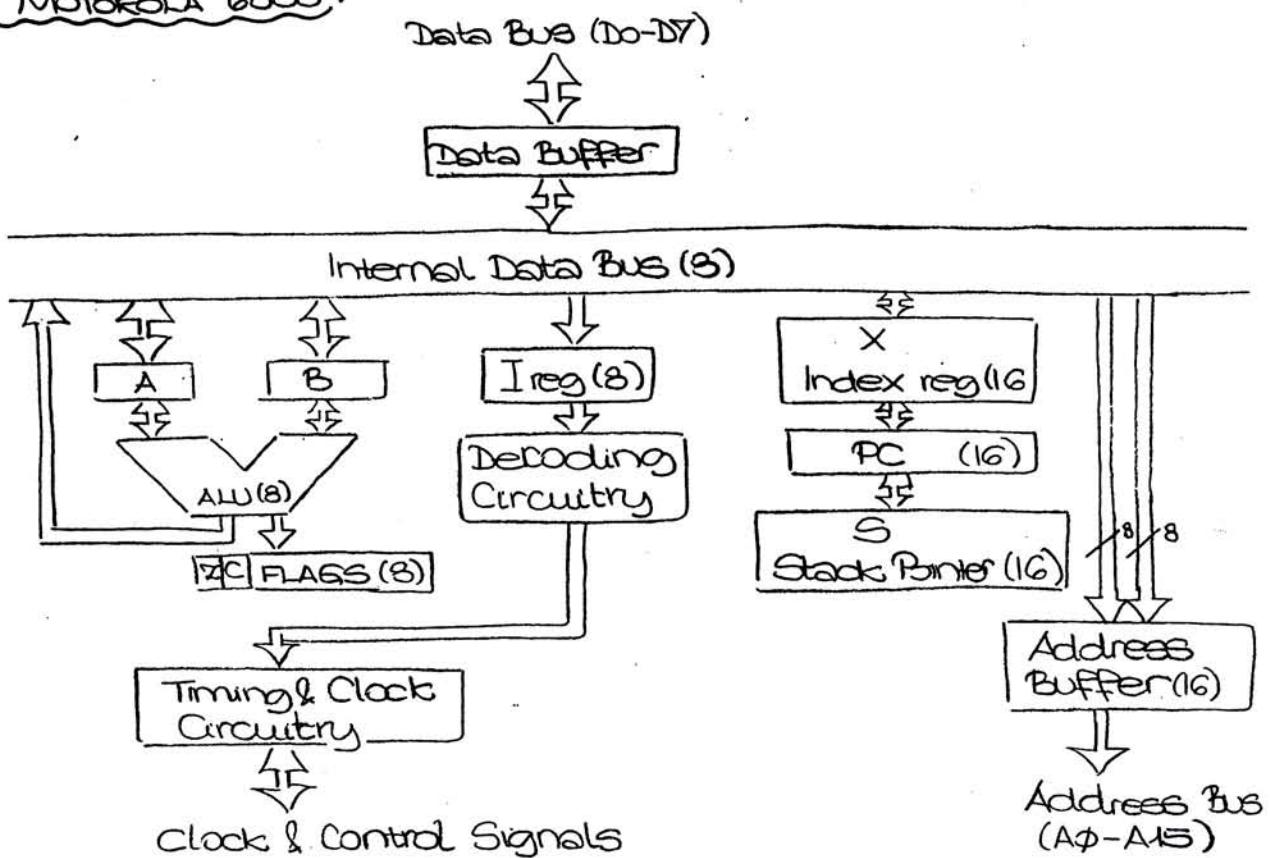
* VRIJINA:

ADRESIRANJE NULTE STR.

→ ušteda 1 byta

→ 1 ciklusa takta

MOTOROLA 6800:



Narzgled nema DC (ne vidi se na slici, ali postoji!)

INTEL 8080A : Reg. parovi $W, Z \rightarrow$ reg. u geni (DC)

$B, C \leftarrow$
 $D, E \leftarrow$
 $H, L \leftarrow (8) (8)$

Ostali registri : Stack Pointer (16)
Program Counter (16)

(zbroj čestog INC, DEC) \leftarrow Incrementer/Decrementer
ubrzanje rada

\downarrow
Address Buffer

RCA 1802: \Rightarrow 1. 8-bitni μ P u CMOS tehnologiji

\Rightarrow von Neumannov koncept, ali ne akum. orjeniran,
ved registrarski orijentirane arhitekture?

3 4 bitni register koji pokazuju na 1 od registrarskih (S)
parova ($8+8=16$) postaje programsko brojilo.

SET % 0001 \Rightarrow Reg par R1 je PC
Izbor registra je 4bitan

Nema PC

NEMA NI X
(indexnog reg)

\Rightarrow Naredboom \Rightarrow
SEX % 0002

biramo reg. par R2
koji postaje X reg

* MOTOROLA 6800: \rightarrow 8 data reg (32bita) opde namjene
 \rightarrow 8 adr. reg (-" -)

- * naprednija inačica \Rightarrow 16 bitni μP , ali sadrži mnoge elemente 32 bitne μP
- \rightarrow 16 bitna interna adresabla i 16 bitna inter. data sab.
- \rightarrow 2 stack pointera (2 stoga \Rightarrow konznički i kontrolni)
- \rightarrow 32 bitna ALU (ili 2 16 bitne)

Logički ANALIZATOR: A₀, A₁, A₂, A₃

$\phi_1, \phi_2, RW, VMA \rightarrow$ kontrolni signali
D₀ - D₇

- \rightarrow nemultipletsirana linija \rightarrow u vremenu 1 linija ima 1 funkciju
 - \rightarrow mult. linija \rightarrow ovise o vremenu, linija može imati funkciju
 - adr. sabirnice
 - podatkovne sab...
 - itd.
- SLOŽENIJA ANALIZA?

INTEL 80286 i 80386:

Clock INPUT : Frekv. waza clock dijeli se sa 2 (ili množi, ovise o porodicu), dobiva se PCLK (Processor Clock)

CLOCK 40 MHz
PCLK 20 MHz } sa 2

INTEL 486 DX	Clock INPUT = Pclock
INTEL 486 DX4	$f(Pclock) = 2x f(Clock)$ ili $3x f(Clock)$

Obično proizvođač referencira na Pclock kada spominje brzinu radnog

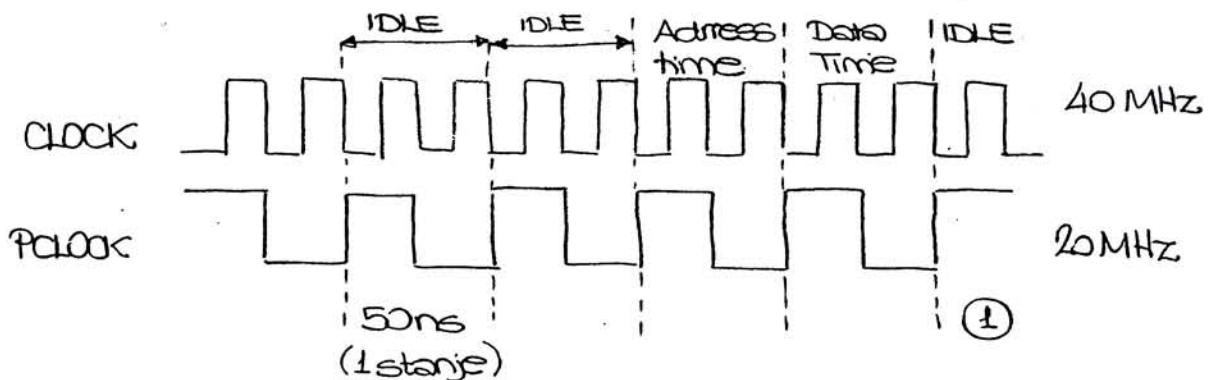
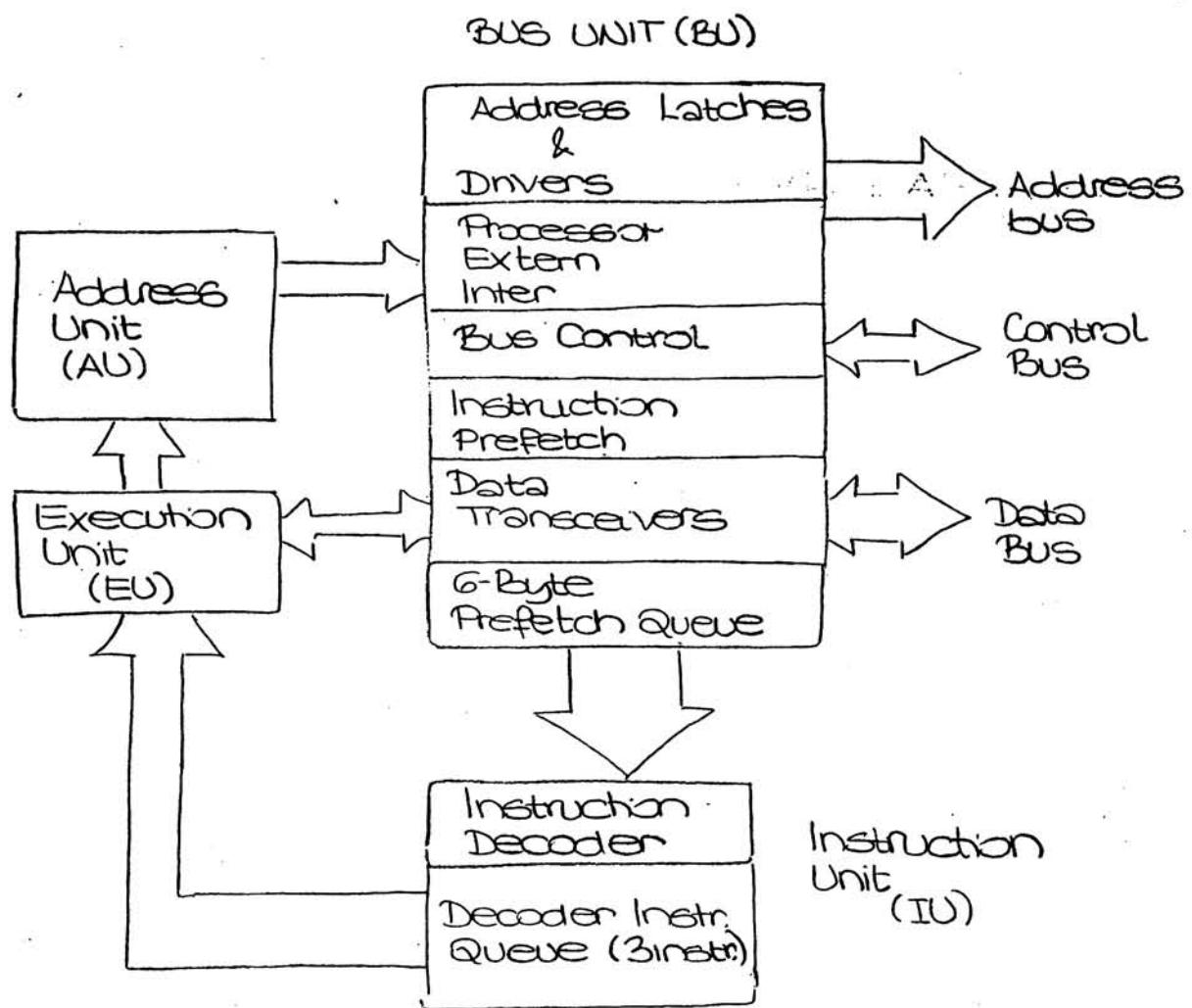
Kad μP izvodi operaciju READ ili WRITE, započinje sljed događaja \rightarrow SABIRNIČKI CIKLUS (bus idle)

μP :

- \rightarrow postavlja adresu na adr. sab.
- uprav. signale na uprav. sab.
- označava vrstu prijenosa (npr. ČITANJE memo ili I/O)
- prenosi podatke između dlynje lokacije μP

X86 μP imaju podsvestav koji se naziva SAB.JED i koja podržava sabirničku jedinicu (BUS UNIT)
SAB.JED = stroj stanja
Vrijeme trajanja stanja = 1 perioda Pclock-a

80286 µP:



IDLE → sabirnicu je neaktivna
 Address time → µP mora obaviti R i W. (1 PCLK), µP postavlja adr. na adr. sab. Na upr. sab. je R i W
 Data time → vrijeme prijenosa podataka
 (također 1 perioda PCLK-a) ⇒ VRJEME PODATAKA

① READY # SE UZORKUJE OD STRANE µP

Na kraju stanja, sa zadnjim brodom signala PCLK, µP, dolje, ispituje stanje READY # ulaza.
Ako je READY # nisko (logička 0), sabirnička jedinica završava sab. ciklus, inače se ne prelazi u IDLE, umesto se još jedno stanje DATA-TIME (*)

O-WAIT-STATE BUS CYCLE:

Najbrži sab. ciklus x86 µP traje 2 periode PCLK-a.
Npr. 100ns (20 MHz) → 50ns za VRJEME ADRESIRANJA
→ 50ns za - " - PODATAKA

→ nema umetanja stanja bekanja
(periferija je iš = brza ili brža od µP pri komunikaciji)

* WAIT-STATE (Data Time)

ISA sabirnica (Industry Standard Architecture)

→ proširenje IBM PC-sabirnice i temelji se na 8088 sustavima

→ ima 62 signalne linije na 8-bitnom dijelu, uključujući:

20 linija za adresiranje memorije

8 linija za podatke

te upravljačke signale (memo READ, memo WRITE)
I/O READ, I/O WRITE

(linije za zahtjevanje/dodjelu & potvrdu prekida)

te za DMA

4.166 MB/sec (8 bitna prenos) → TAKT SABIRNIKE
8.33 MB/sec / 8.33 MB/sec (16 bitna - " -) → određen s ovom freq.

EISA (Extended ISA)

→ proširenje do 32 bita → 8,16-bitni prenos podataka

32 - - " - - " -

32-bitna adres sab.

33 MB/sec (vršna brzina prenosa)

DMA; bus master

- također kompatibilna (utorima) sa stanjim karticama

PCI sabirnica (Peripheral Component Interconnect bus)

Npr.

1024 × 768 zaslon / true color

(3 bytes/pix) za sliku u pokretu

→ CRV, PLAVA, ZELENA komp.

1 zaslon = 2.25 MB podataka!

Za 30 zaslona u sec, potrebna je brzina µPod 67.5MB/sec!

Primer: Ako je video zapis sa HARD DISCA, CD-ROM ili DVDA, podaci se prenose preko sabirnice i u memo. U tetc ande se prenose (ponovo preko sab) prema graf. adaptoru $\Rightarrow 135 \text{ MB/sec}$.

(samo za video, ne uzimajući u obzir potrebe CPU-a i drugih uređaja na sabirnicu)

Intel, 1990. \rightarrow PCI

Intel-based računala

Pentium

Sun-Ultra SPARC 11i

Izvorno:

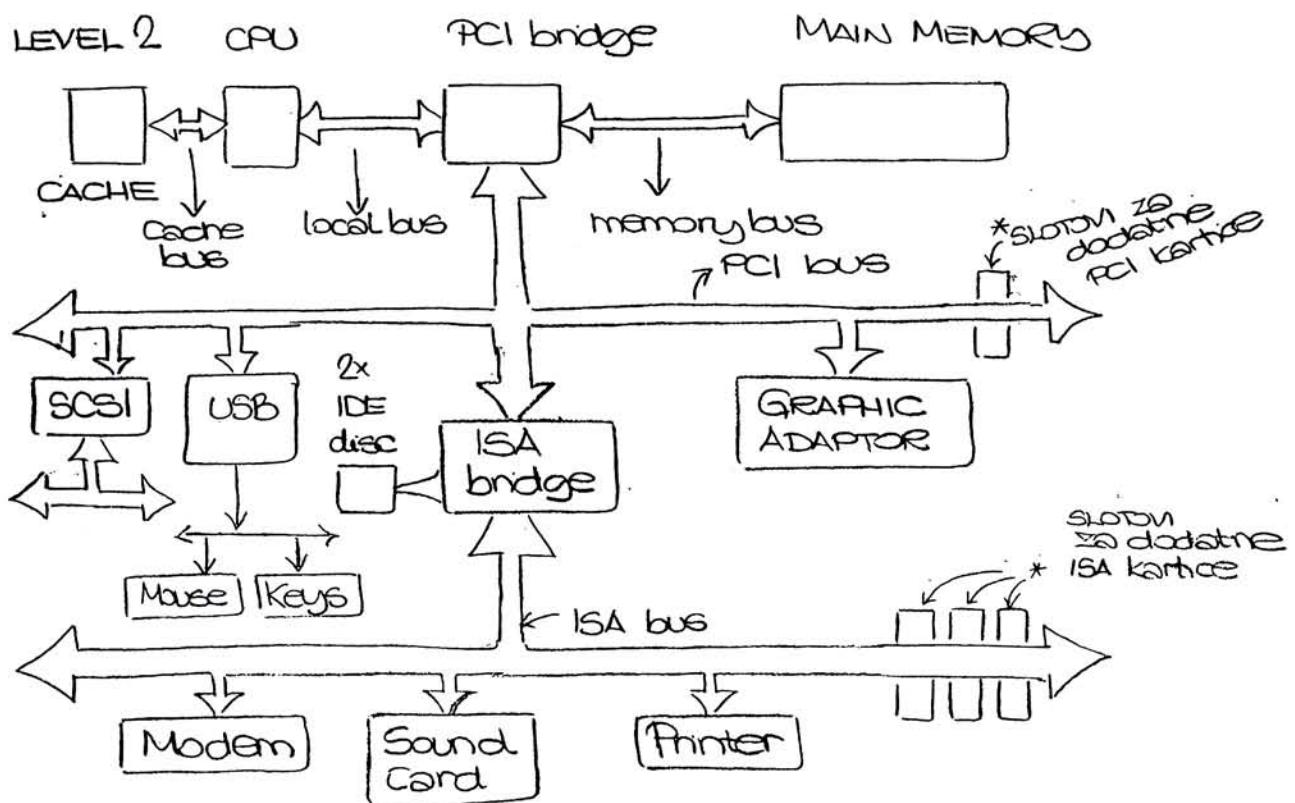
PCI prenosi 32 bita / periodu
na radi na 33 MHz (30 nsek) :
ima 132 MB/sec.

1993. god. \rightarrow PCI 1.0 (66 MHz i rukuje se sa
64-bitnim prijenosom \rightarrow
 $\rightarrow 64 \text{ bit} \times 8 \text{ byte} \Rightarrow 528 \text{ MB/sec}$)

1995. god \rightarrow PCI 2.1
PCI 2.2

još nije dovoljno za
memorijsku sabirnicu

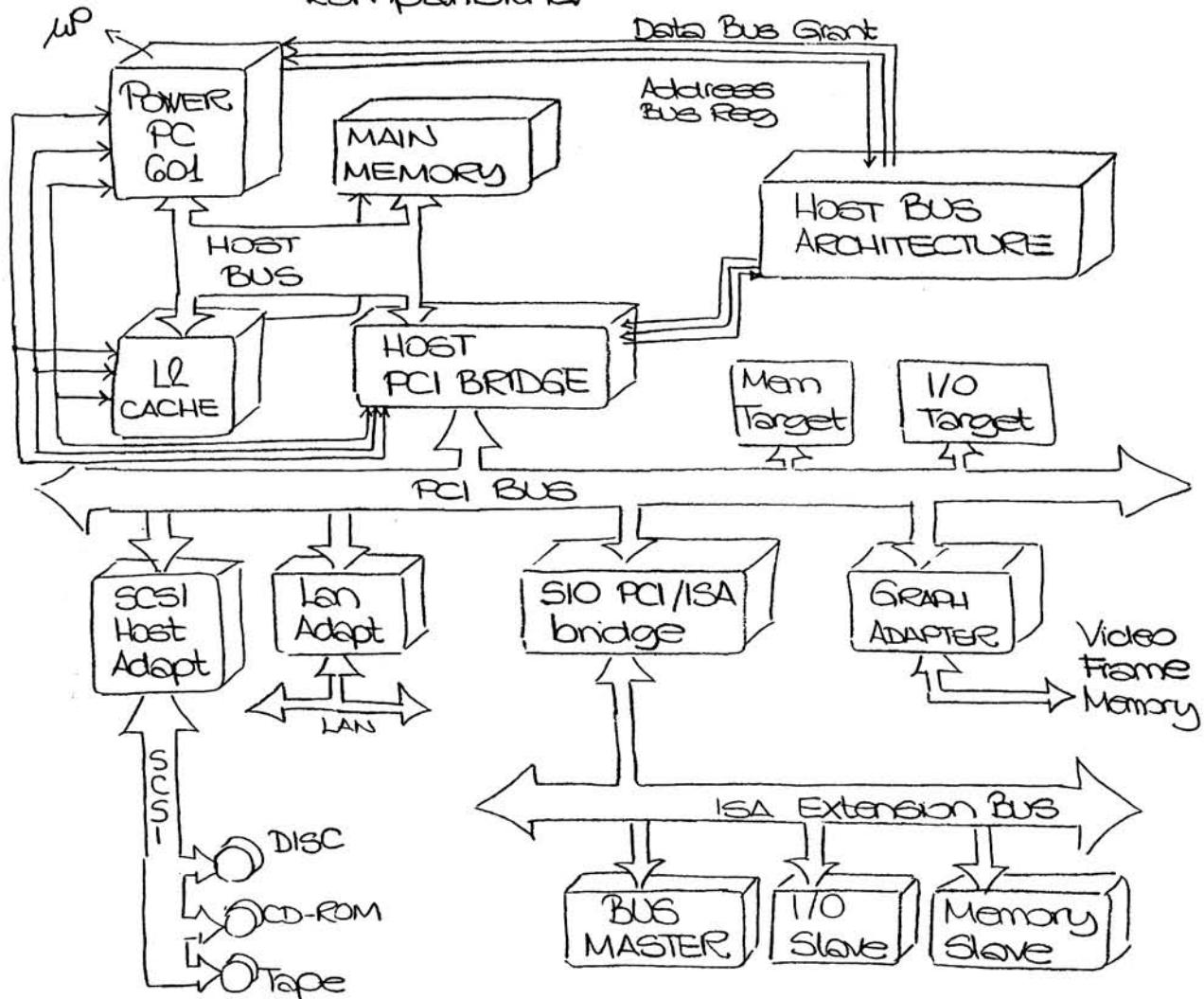
\Rightarrow PCI nije kompatibilna sa prethodnim (ISA, EISA) standardima, pa je Intel ponudio rešenje:



IDE → Integrated Drive Electronics
 USB → Universal Serial Bus

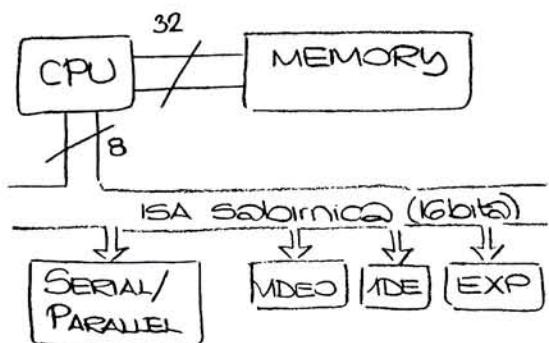
Adresne linije / linije podataka su PCI multiplexirane

ISA bridge → spajanje ISA sabornice jer PCI nije kompatibilna



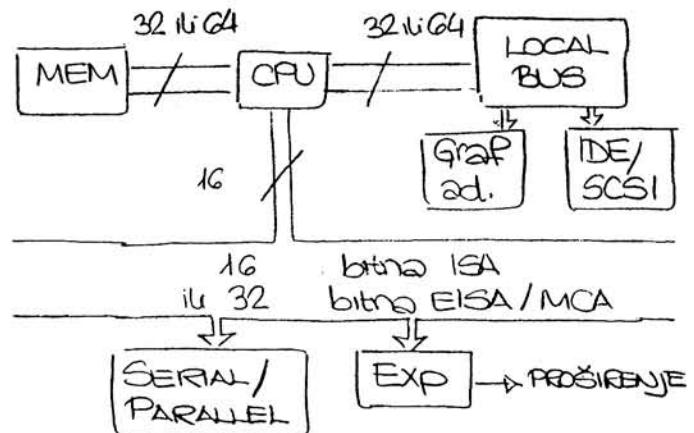
SCSI - Small Computer System Interface

PC 1986. god.



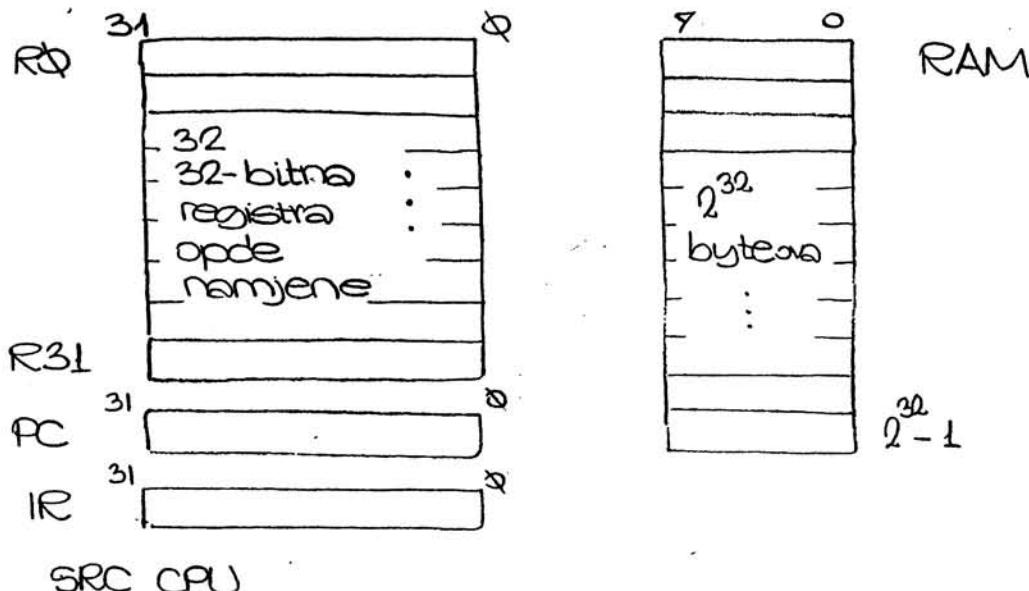
1

PC 1993. god.

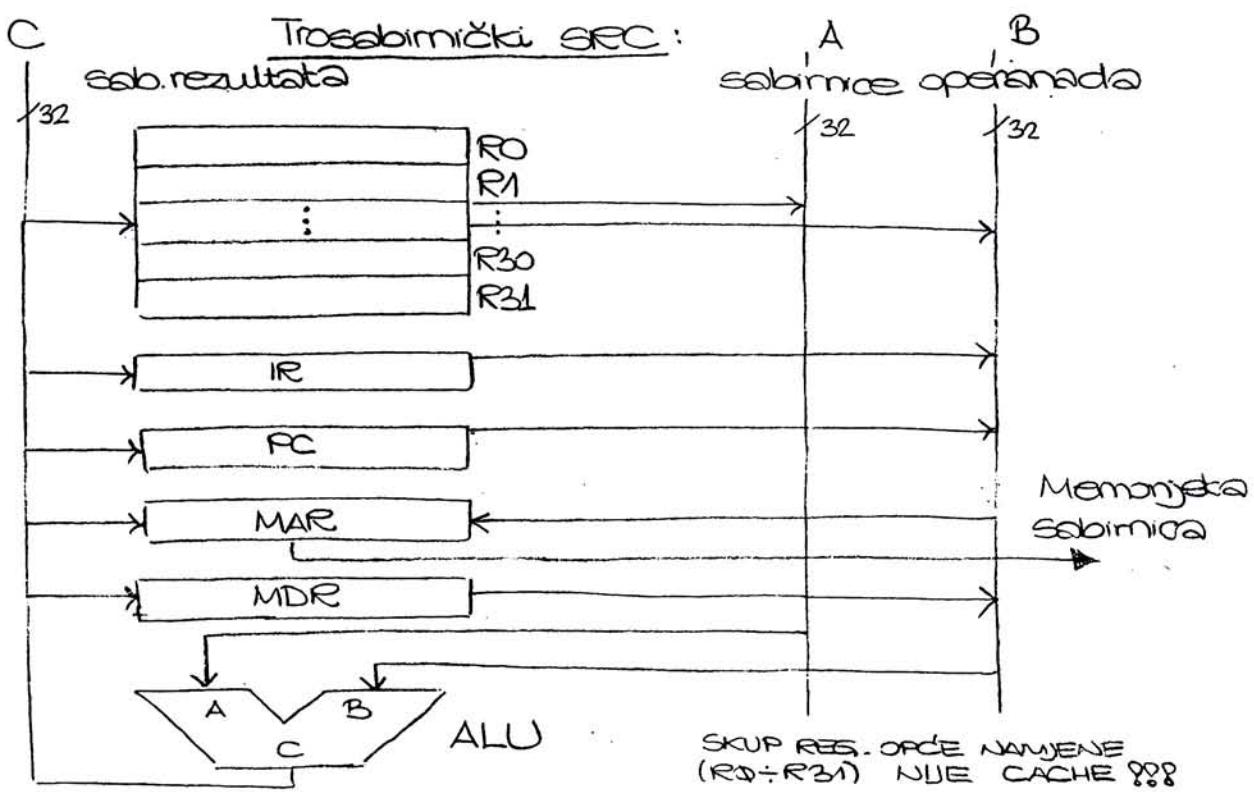


SRISC - SRC (Simple RISC COMPUTER)

PROGRAMSKI MODEL:



- iako je mem. byte organizirana, samo 32 bitna njed može se dohvati/pohraniti u memoriju
- operandi iz memo. mogu se dohvati/lu pohraniti samo instrukcijama tipa LOAD / STORE
- njed na adresi A je definirana kao 4-byte tako da leži na taj (A) i slijednim trima adresama.
- byte na najnižoj adresi sadrži 8 najznačajnijih bitova (Big-Endian Byte Ordering ???)



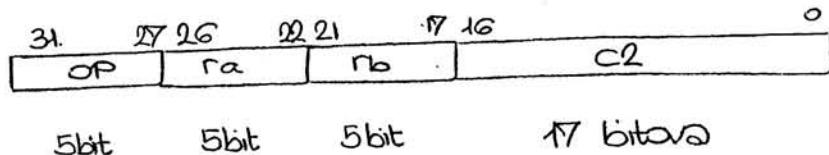
T0 $\text{MAR} \leftarrow \text{PC}$; $\text{MDR} \leftarrow M[\text{MAR}]$; $\text{PC} \leftarrow \text{PC} + 4$
 T1 $\text{IR} \leftarrow \text{MDR}$
 T2 $R[r_a] \leftarrow R[r_b] + R[t_c]$
↳ 4 bytes address
op.koda?

ISA - Instruction Set Architecture

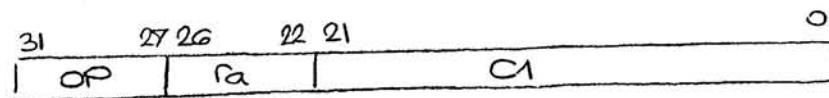
FORMATI INSTRUKCIJA:

src ima SEDAM različitih formatova instrukcija:

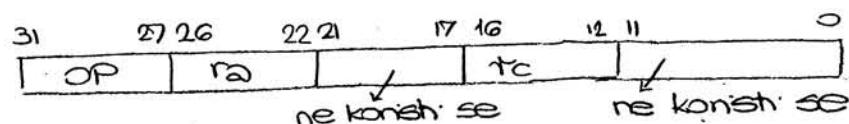
1.
ld, st, la,
add, andi
ori



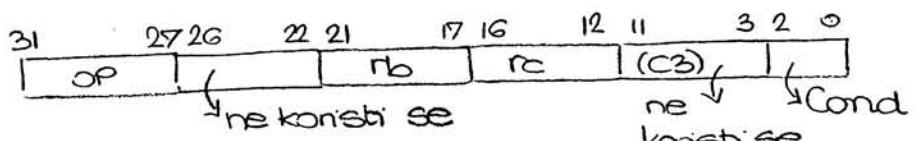
2.
ldr, str, lar



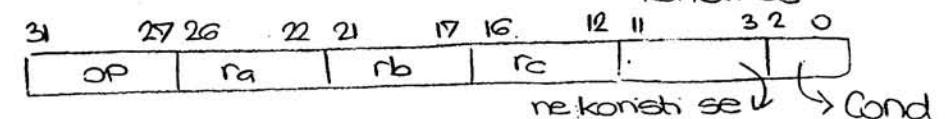
3.
reg, not



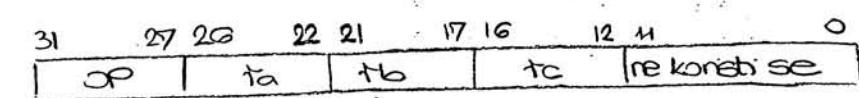
4.
br



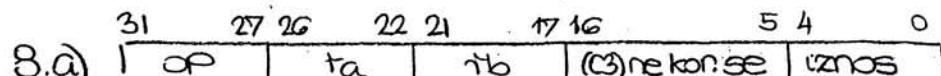
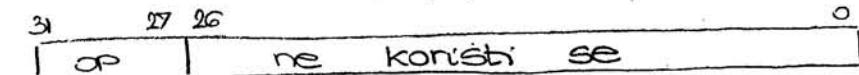
5.
bte



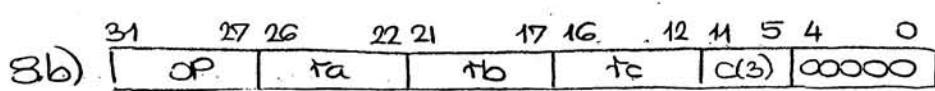
6.
add, sub,
and, or



7.
nop, stop



8.
(POSMACI)
sht, she



INSTRUKCIJE ZA PRISTUP MEMORIJI (LOAD i STORE)

load i store su JEDINE instrukcije za dohvati i pohraniti operanada iz/u memorije.

4. load ins

\Rightarrow ld, ldr, la, lar
 \Rightarrow st, str.

ld ra,C2 : izravno (direktno adresiranje) $R[ra] = M[C2]$
ld,ra,C2(rb) ; indeksno adresiranje ($rb \neq 0$) $R[ra] = M[C2 + R[rb]]$

st,ra,C2 : izravno adresiranje $M[C2] = R[ra]$
st,ra,C2(rb) indeksno adresiranje ($rb \neq 0$) $M[C2 + R[rb]] = R[ra]$

PAZ! Ro ne možeš konistiti u indeksnom adresiranju! (kao rb)
(njegov indeks je 0, a $rb \neq 0$)

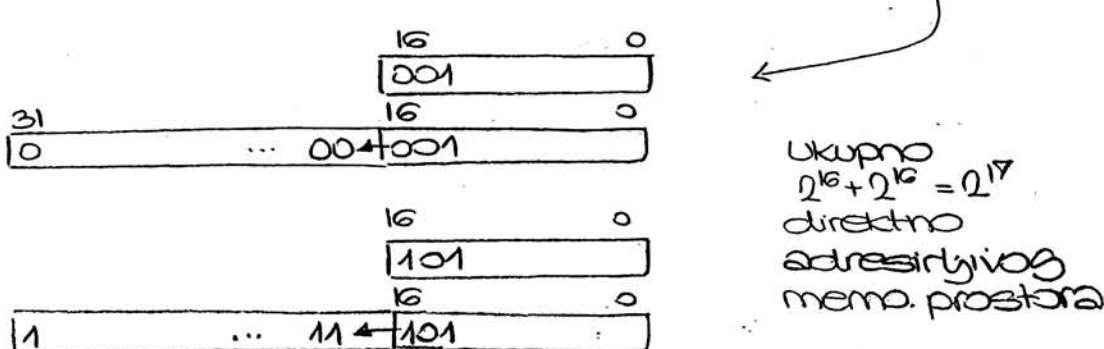
la ra,C2 : puni adresno polje $R[ra] = C2$
la,ra,C2(rb) : - - - - - $R[ra] = C2 + R[rb]$

↳ INSTRUKCIJE RABE 1. FORMAT INSTRUKCIJE?

load

- puni se reg. određen 5-bitnim poljem ra ($2^5=32$)
- adresa izvorišta određena je 17-bitnim vrijednostima u polju C2
 (adresa je 32-bitna \Rightarrow proširenje bita predznaka)
 (SIGN EXTENDED)
- najznačajniji bit se proširuje na ostalih 5 bitova !!
- rb ima dvostruku ulogu:

↓
 Ako $rb=0$ (00000) tada $rb=0$ služi kao signal
 upr. jedinicu da je mem. adresa izvorišta upravo C2
 koja se, u skladu s 2^K , pretvara u 32-bitnu (SIGN EXTEND)



Ako je $rb \neq 0$, tada je mem. adresa : $R[rb] + C2$
 (based, or displacement addressing mode)

Pozore! Izračunavanje $R[rb] + C2$ se obavlja tijekom izvođenja instrukcije (run time)

ld \Rightarrow Puni se reg $R[ra]$ operandom:

- koji je pohranjen na adr C2 ($rb=0$)
- - - - - $C2 + R[rb]$ ($rb \neq 0$)

st \Rightarrow Pohranjuje sadržaj reg. $R[ra]$ na

- adr. C2 $(rb=0)$
- adr. $C2 + R[rb]$ $(rb \neq 0)$

la, (load address)

- računa adresu operanda (kao u gornjim slučajevima)
- ali umjesto dohvata operanda - pohranjuje izračunatu vrijednost u R[ra]
- rabi se za sintezu složenijih načina adresiranja

Ograničenje izravnog adresiranja:

C2 je 17 bitna vrijednost: operandi se mogu nalaziti na li prvi 2^{16} byteva ili zadnjih 2^{16} byteva
(zbog SIGN EXTENDED !!)

32 bitna addressa:

Adresni prostor: $00000000 \div 0000FFFF$ 1. prostor
 $FFFF0000 \div FFFFFFFF$ 2. - " -

Za pristup operandima bilo gdje drugdje:

Konistički adresiranje s pomaknućem: $R[Rb] + C2$

baza + pomaknuće

ili Indirektno adresiranje: C2=0 pa R[Rb] određuje adr.

Pozor! U postupku zbrajanja (računanja) ADRESE, 1. se 17-bitne pomaknuće treba pretvoriti u 32-bitni broj (SIGN EXTENDED)

REGISTARSKO OKNO:

- instrukcije imaju pristup samo nekim registrima (reg. okno)
- registrarska okna 2 procedure mogu se preklapati

Koncept protočnosti dolazi do izražaja upravo kod RISC arhitekture (CISC je po tome lošiji od RISC-a)

Relativno adresiranje: (odnosno adresiranje)

Uporabom rel. adresiranja dobiva se (računa se) addressa operanda kao addressa relativna u odnosu na programsko brojilo PC.

load; store (rel. adresiranje)
↳ RABE 2. FORMAT INSTRUKCIJE :

ldr ta, C1 ⇒ napuni reg. relativnim načinom adresiranja:
 $R[ra] = M[PC+C1]$

str +a, C1 ⇒ pohrani: $M[PC+C1] = R[ra]$

lar ra, C1 ⇒ napuni rel. adresu $R[ra] = PC+C1$ PAZI !!

Efektivna adresa (EA) oblikuje se tijekom izvođenja instrukcije (run-time addition): C1 + PC

"PREMJEŠTIVE" INSTRUKCIJE

(eng. RELOCATABLE INSTRUCTIONS)

- relativne adrese (u odnosu na PC) čine instrukcije premještivim → cijeli se programski moduli i podaci mogu se premještati bez mijenjanja vrijednosti pomaknuda

C1 ima 22 bita → može se specificirati adresu u prostoru od $\pm 2^{21}$ (1 bit za predznak)
(u odnosu na tekudu instrukciju)

PRIMJERI load i store instrukcija:

<u>INSTRUKCIJA:</u>	<u>OP</u>	<u>ra</u>	<u>tb</u>	<u>C1</u>	<u>KOMENTAR:</u>	<u>NAČIN ADRESIRANJA</u>
ld r1,32	1	1	0	32	$R[1] \leftarrow M[32]$	Izravno
ld r2,24(r4)	1	22	4	24	$R[22] \leftarrow M[24+R[4]]$	Pomak
st.t4,Q(r9)	3	4	9	0	$M[R[9]] \leftarrow R[4]$	Reg.ind.
la r7,32	5	7	0	32	$R[7] \leftarrow 32$	Usporno
ldr r12,-48	2	12	-	-48	$R[12] \leftarrow M[PC-48]$	Relativno
lar r3,Q	6	3	-	0	$R[3] \leftarrow PC$	Reg.

→ ovako dobijemo stanje PC-a ???

U fazi izvrši → potrebna komun. s memorijom } KAKO TO
i u fazi prihvati → " -11- -11- -11- } REALIZIRATI?
↓
Koncept protočnosti

ARITMETIČKE I LOGIČKE INSTRUKCIJE

INSTRUKCIJE S 1 OPERANDOM:

→ RABE 3. FORMAT INSTRUKCIJE

neg ra,tc ; $R[ra] = -R[rc]$ (2'K sadržaj registra)
not ra,ic ; $R[ra] = \overline{R[rc]}$ (1'K -11- -11-)

→ OP kód : 24
→ -11- 15

ALU INSTRUKCIJE S 3 OPERANDA: { 2 operanda
+ specificirano određite

add ra,rb,rc ; 2'K zbrojavanje $R[ra] = R[rb] + R[rc]$
sub ra,rb,rc ; 2'K oduzimanje $R[ra] = R[rb] - R[rc]$

and ra,rb,rc ; logičko I
or ra,rb,rc ; - " - ILI

$$R[ra] = R[rb] \wedge R[rc]$$

$$R[ra] = R[rb] \vee R[rc]$$

odd (OP = 12)
sub (OP = 14)
and (OP = 20)
or (OP = 22)

OVE INSTRUKCIJE RABE
6. FORMAT INSTRUKCIJE.

Registersko orijentirana arhitektura } nemoguća instrukcija
Load i store arhitektura } tipa INC m
(direktno povećava
(vrijednost na memo.lok.)

KONCEPT PROTOČNOSTI:

Svaka instrukcija se obavlja u 1 periodu vremenskog vodenja (ili se to bar nastoji ostvariti)

ALU INSTR. KOJE KORISTE USPUTNO ADRESIRANJE

RABE 1. FORMAT

addi ra,rb,c2 ;	$R[ra] = R[rb] + C2 \rightarrow$ usputna konstanta
andi ra,rb,c2 ;	$R[ra] = R[rb] \wedge C2 \rightarrow$ (17 bita \Rightarrow proširuje)
ori ra,rb,c2 ;	$R[ra] = R[rb] \vee C2 \rightarrow$ (sebit predznak do 31)

INSTRUKCIJE ZA POSMATEK (SHIFT INSTRUKCIJE)

Instrukcije posmaju operand iz $R[rb]$ (udesno, uljevo, kružno) i smještavaju rezultat u $R[ra]$. Broj posmatranih mjesto određen je 5-bitnim nepredznačnim brojem.

BAR REGISTAR: omogućava posmatranje sadr. registra za proizvoljan broj bitova sve u jednom taktu vrem. vodenja (BAČVASTI REG.)

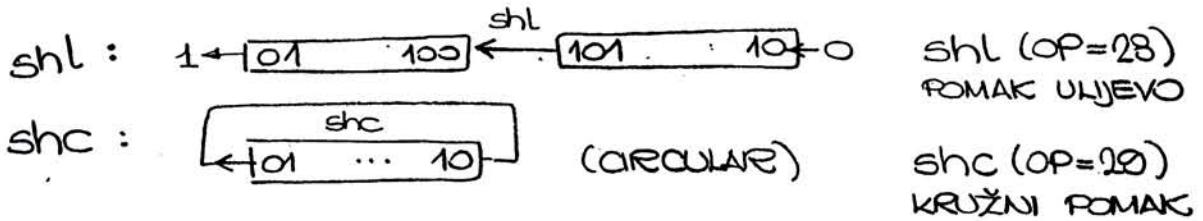
Cjeli broj koji određuje iznos posmataka pohranjen je kao usputna vrijednost u 5 Lsb u instrukciji (RABI SE 8.a FORMAT INSTRUKCIJE), ali ako je ta vrijednost 0, tada se kao iznos posmataka uzima 5 Lsb registra $R[rc]$ (FORMAT 8.b)

5bitna vrijednost \rightarrow iznos posmataka od 0 do 31
RAZLIKUJEMO 2 vrste posmataka UDESNO:

sh+ (OP = 26) ↳ logički posmatak	:	shra (OP = 27) ↳ antimetrički posmatak
----------------------------------------	---	----------------------------------------------

shr : $\xrightarrow{\text{sh+}}$ 0 \rightarrow $\rightarrow 1$

shra : $\xrightarrow{\text{shra}}$ msb \rightarrow $\rightarrow 1$



Instrukcije posmaka:

Shift can be in a register or the constant field of the instruction.

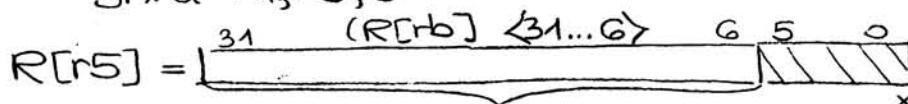
$$n := ((C3 < 4 \dots 0> = 0) \rightarrow R[r_c] < 4 \dots 0>; \\ (C3 < 4 \dots 0> \neq 0) \rightarrow R$$

$$\begin{aligned} shra &(:= OP=27) \rightarrow R[r_a] < 31 \dots 0> \leftarrow n @ R[r_b] < 31 \dots 0> \# R[r_b] < 31 \dots n> \\ shl &(:= OP=28) \rightarrow R[r_a] < 31 \dots 0> \leftarrow R[r_b] < (31-n) \dots 0> \# (n @ 0); \\ shc &(:= OP=29) \rightarrow R[r_a] < 31 \dots 0> \leftarrow R[r_b] < (31-n) \dots 0> \# \\ &\quad R[r_b] < 31 \dots (31-n)>; \end{aligned}$$

Značenje operátora @: $5 @ 0 = 00000$

PR:

shra r4, r5, 6



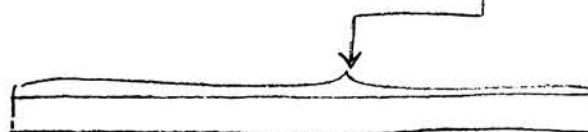
$n = 6$

$$(n @ R[r_5] < 31 >) \# R[r_b] < 31 \dots n >$$

Ako: $R[r_5] < 31 > = 1$

$$n @ 1 = \underline{\underline{6 @ 1}}$$

$$R[r_4] = \overbrace{111111}^{31}$$



INSTRUKCIJE GRANANJA (BRANCH INSTR.)

bt ($OP=8$) ; btl ($OP=9$) RABE FORMATE INSTR. 4, 5

Instrukcija bt grana tako da se mijenja sadržaj PC s ciljnom adresom.

Instrukcija btl (BRANCH & LINK) se izvodi tako da KOPIRA PC u tzv. LINKAGE REGISTAR i to preje grananja (povezni reg) \downarrow ($R_0 \div R_{31}$)

Povezni reg. dopušta povratak iz potprogramma (rabiti se za implementaciju procedura i funkcija u više prog. jezike (HLL)).

Operacije nad podacima:

→ aritmetička, li logička operacija (specificirano op. kodom)

Računanje adrese rezultata:

→ koristi se jedinica za rač. adr. rezultata
(može biti složena operacija)

→ za nečasne, jednostavne tipove podataka, ved za
npr. VEKTORE, dolazi do petlje: RAČUNAJ ADR. REZ.

\uparrow \downarrow
POHRANI ADR. REZ

→ faza pribavi se pojavlja samo 1,

faza izvrši se ponavlja koliko ima komponenti vektora
(npr. $\vec{a} = (a_1, a_2, a_3)$ ima 3 faze izvrši ??)

VEKTORSKI PROSECI:

(SUPER PROTČNA)
podržavaju najznačajnija računanja RADUNALA ??

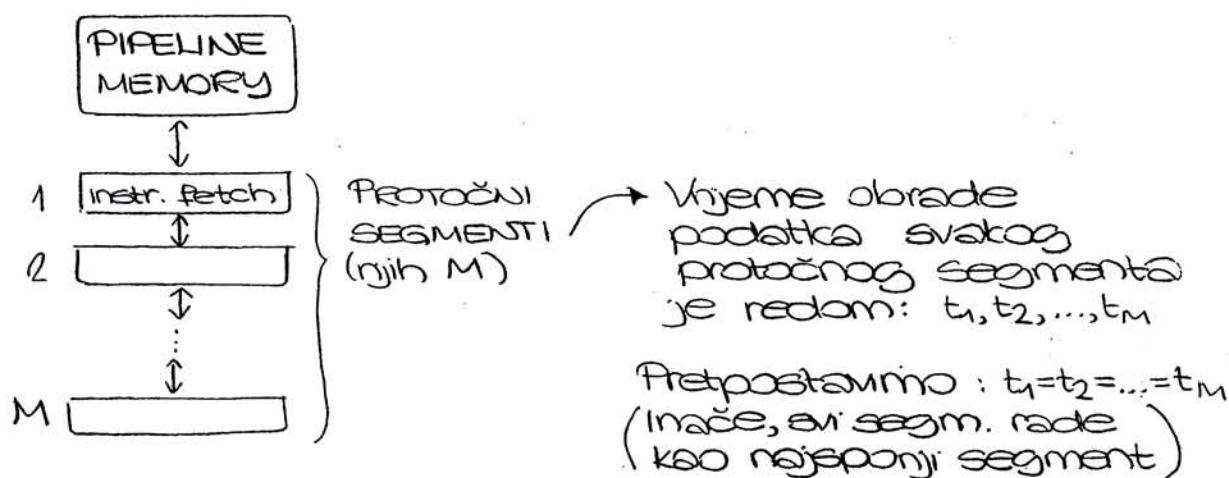
Pohranja rezultata: upiši rez. u memoriju ili U/I podsustav

Računanje adrese instrukcije:

→ ako je operacija grananja \Rightarrow izračunaj vrijednost PC
(inache PC ved sadrži adr. sljed. instrukcije)

PIPELINE MEMORIJA? Kako je ostvareni?

↳ davanjem brzinom šalje podatke procesoru



N - broj instrukcija koji se treba izvršiti (program)

Pretpostavimo: $N \gg 1$

Prvi rez. se pojavlja nakon $M \cdot t_m$ vremena, a slijedeći
odmah nakon njega.

Ukupna obrada traje: $T_{obrade} = M \cdot t_m + (N-1) \cdot t_m$

Obrada 1 instrukcije: $t_{one} = \frac{T_{obrade}}{N}$

$$t_{ef} = \lim_{N \rightarrow \infty} \frac{T_{BRADE}}{N} \Rightarrow \text{za } N \gg 1$$

$$t_{ef} = \lim_{N \rightarrow \infty} \frac{M \cdot t_M + (N-1)t_M}{N} = t_M$$

Vrijeme obavljanja
1 instrukcije nije =
 $\sum t$ svih segmenta,
ved = t jednog
segmenta?

RISC \rightarrow 1 instrukcija u 1 perioda vrem. izvođenja?
u načelu neizvedivo...

No:

Za skup jednostavnih instr. = jednake kompleksnosti i istih vremena izvođenja (npr. 5 perioda).

Ako želimo da $t_{ef} = t_M$, onda uzmemos 5 protičnih segmenta.)

TEORETSKI MOGUĆE, NO ŠTO AKO SE GRANA PROGRAMA?

Za 1 instr. po 1 vrem. taktu:

broj protičnih segmenta = broj vrem. taktova svake instrukcije

Za >1 instr. po 1 vrem. taktu:

Više paralelnih, protičnih struktura:

- za Antim-log. operacije
- za dohvati/spremanje itd...

Tzv. 2. razina paralelizma.

Npr. za 1 instr. po 1/5 vrem. taktu:

- fukt. nezavisne instrukcije? Ne treba. U arhitekturi 2. i 3. generacije RISC računala

OUT OF ORDER	EXECUTION	(izdavanje izvan red.) (izvođenje - - - - -)
- - -	- - -	- - - - -

\Rightarrow procesor izvršava instrukcije različitim redoslijedom od zadanih, sekvenčnog

UVJET:
sve protične strukture moraju biti zapoštene da bi performance bila maksimalna.

Iako obje (CISC i RISC) imaju protičnost kao svojstvo, RISC je pogodniji jer mu sve instr. imaju = veličinu, i = vrijeme izvođenja instrukcija.

PROTIČNI SEGMENTI

IF (Instruction Fetch): Pribavljanje instrukcije

Upotrebljava se PC za dohvati sljedeće instrukcije, koje se obično nalaze u priručnoj (eng. CACHE) memoriji koja se čita tijekom faze "pribavi".

ID (Instruction Decoding and operand fetching)

Istodobno se dekodira operac. kod i dohvadaju operandi

iz skupa registara. Operandi su (zbog load/store arhitekture) već u registrima, pa za antm. logičke operacije moguće je istodobno i dekodirati i dohvadati operande. No, za load/store operacije, u reg. je adresa izvornja/odredišta, pa je opet potreban paralelizam dekodiranja i dohvata operanada.

EX (Instruction execution)

Obavlja se operacija specificirana oper. kodom.
Za load/store instrukcije u ovom protičnom segmentu računa se efektivna adresa.

Load/store arhitektura:

nema potrebe za 2 jedinice (antmetičke), jer su instr. ili load/store (onda EX računa $\langle ea \rangle$) ili antm./log. (onda ID dohvada operande)
CISC, međutim, ima posebnu jedinicu koja računa ef. adresu ($\langle ea \rangle$)

ME (Memory access)

Izvode se load/store instrukcije (obično se upotrebljava priručna memorija) \Rightarrow nekonisti se ovaj segment za ostale instr.

WB (Result write-back)

Rezultat operacije se upisuje natrag u skup registara

\longrightarrow vijest

1 instr.	IF	ID	EX	ME	WB		
2 instr.	IF	ID	EX	ME	WB		
3 instr.	IF	ID	EX	ME	WB		
4 instr.	IF	ID	EX	ME	WB		
5 instr.		IF	ID	EX	ME	WB	
6 instr.			IF	ID	EX	ME	WB
:						...	

U A periodi pristupa se memoriji (IF), pa je mem. stalno okupirana instr. dohvata sljedećeg op. koda. Ako je sama instr. tipa load/store, u ME će biti dohvati (konflikt aktivnosti protičnih segmenta IF i ME)?
Zato je dobro fizički odvojiti INSTRUKCIJSKU (programsku) memoriju, od one za podatke (PODATKOVNA MEMORIJA) \rightarrow 2 CACHE MEMORY

UPRAVЉАЧКА JEDINICA

- 1 verzija RISC procesora podržavala je C prog. jezik (upravo preko upravl. jedinice)
- teže je dizajnirati up da efektivno podržava više programskih jezika.

FUNKCIJA UPRAV. JEDINICE:

- pribavljanje instrukcije
- interpretacija
- generiranje upravl. signala koji se šalju ALU i ostalim jedinicama)

A Upravljanje slijedom instrukcija (INSTR. SEQUENCING)

- odabir instrukcija
- prijenos upravljanja s 1 instruk. na drugu.

B Interpretacija instrukcije - metode koje se upotrebljavaju za skriviranje signala

A) Najjednostavnija metoda upravljanja slijedom instrukcija: svaka instr. određuje adresu svog naslednika (prva računala)
NPR. EDVAC (Electronic Discrete Variable Computer)

↳ oblik sintetičke instrukcije:
 $A_1 A_2 A_3 A_4 \text{ OP}$
adresa operanda rezultat adresa sljed. instrukcije

NEDOSTATAK:
povećanje duljine nječi

Danas: DATA FLOW arhitektura
up de izvršiti operaciju kad za nju ima sve operande
a ne kad dobije instrukciju za nju
(arhitektura upravljanja tokom podataka)
- visoki stupanj paralelizma

DRUGA METODA:

Instr. I na adresi (lokaciji) A

I ima jedinstvenog naslednika \rightarrow instr. I'
i ona je na lokaciji A+1

PC - progr. brojilo (instrukcijski adresni register)
sadrži adresu A instrukcije I

Adresa I' određuje se: $PC \leftarrow PC + k$ (za RISC $\rightarrow k=1$)
 $k \rightarrow$ duljina u nječima instrukcije I' (za CISC $\rightarrow k$ varira)

- Instrukcije koje prenose upravljanje IZMEDU INSTRUKCIJA
(ali tako da one nisu slijedne)
- Instrukcije grananja ili skoka (no one su uzrok
hazarda u prototnosti?)

↓
1) U fazu izvrši modificira se PC
($PC \leftarrow X$) (bezuvjetno grananje)

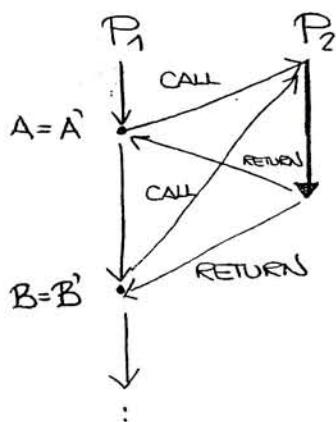
2) Prvo se ispituje UVJET C (cje posljedica neke
ranije instrukcije). Ako je C ispunjen, tada $PC \leftarrow X$
inade $PC \leftarrow PC + k$
(uvjetno grananje)

- Instrukcije za prijenos IZMEDU PROGRAMA:

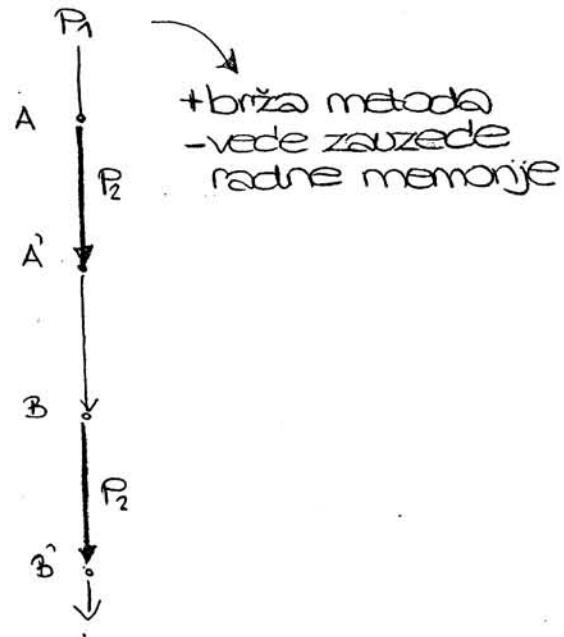


- ① $P_1 \rightarrow P_2$ (instrukcije za pozivanje potprograma)
call ili jump to subroutine

CALL X (X-adresa, ili neka komponenta za izračun adr.
prve instrukcije programa P_2)



\Leftrightarrow



CALL se izvodi u 2 KORAKA:

1 KORAK: Sadržaj PC koji pokazuje na sljedeću instr.
programske strukture P_1 se pohranjuje
na za to predodređenu lokaciju S

2 KORAK: X se prenosi u PC

Povratak (instr. RETURN) → posljednja instrukcija u potprogramu P₂ //

- GNJEŽDENJE potprograma??
↳ ne podržava se sa ovakvom instr. CALL

PRIMJER: PDP-8 (kako se nekad radilo)

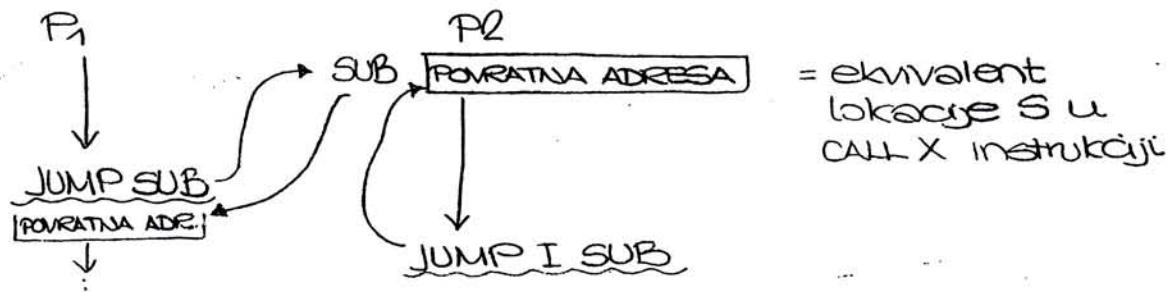
JUMP SUB (jump to subroutine)

PC se pohranjuje na lokaciji SUB

PC se automatski INKREMENTIRA podrazumjevajući
da se 1. instrukcija potprograma SUB nalazi
na PC+1

Upravljanje se prenosi natrag (s podprograma na glavni program) izvodenjem instrukcije:

JUMP I SUB (INDIREKTNÍ SKOK NA LOKACIJI SUB ??)



Ovakva struktura podržava gnezdenje nekog drugog potprogramma P3 unutar P2, pa analogijom P4 u P3 ... itd.
NO NE PODRŽAVA REKURZIJU !!!

Rekursivni program P može se prikazati kao kompozicija P osnovnih instrukcija si (koje ne sadrže P) i samog programa P:

Kpr. u PROLOG -u (progr. jezik)

```
* predak-od (X,Y) :- roditeľ-od (X,B)  
** predak-od (X,Y) :- roditeľ-od (Z,Y)  
                           predak-od (X,Z)
```

- * X je predak od Y ako je X rodotely od Y
- ** X je predak od Y ako je Σ rodotely od Y
 X je predak od Σ

$$\text{Ili } n! = 1 \times 2 \times 3 \times \dots \times n \\ n! = n \times (n-1)! \Rightarrow \text{rekurzivni način}$$

Rješenje: Upotreba LIFO struktura (STOS)
služi za pohranjivanje povratnih adresa



Pr:

Begin

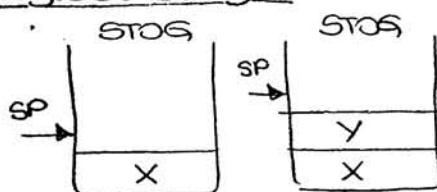
X: CALL SUB
⋮
⋮

end

⋮
Y: CALL SUB
⋮
RETURN

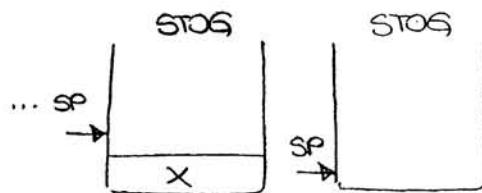
ovde nedostaje
ujet prekuda
rekurzije
(inade činimo)
⇒ petlja

Izgled stoga:



1. pozivanje

2. pozivanje



predzadnji
povratak



zadnji povratak

SP → stack pointer

Ugradnja kontrolnih mehanizama za returniju

Npr:

Begin

N=3
CALL SUB
X: ⋮
⋮

end

SUB:
⋮
N=N-1
IF(N>0) THEN CALL SUB
Y: ⋮
RETURN

N=3 → prvi poziv SUB (N postaje 2)

N=2 → drugi - - - - (N - - - 1)

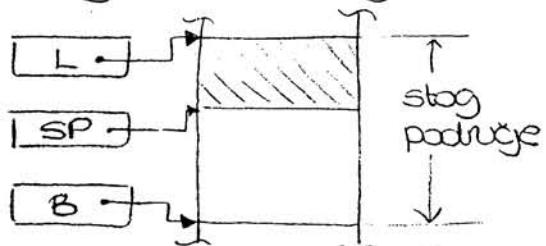
N=1 → treci - - - - (N - - - 0)

N=0 → ne poziva se subjer nije zadovoljen uvjet N>0
prije povrataka iz SUB!

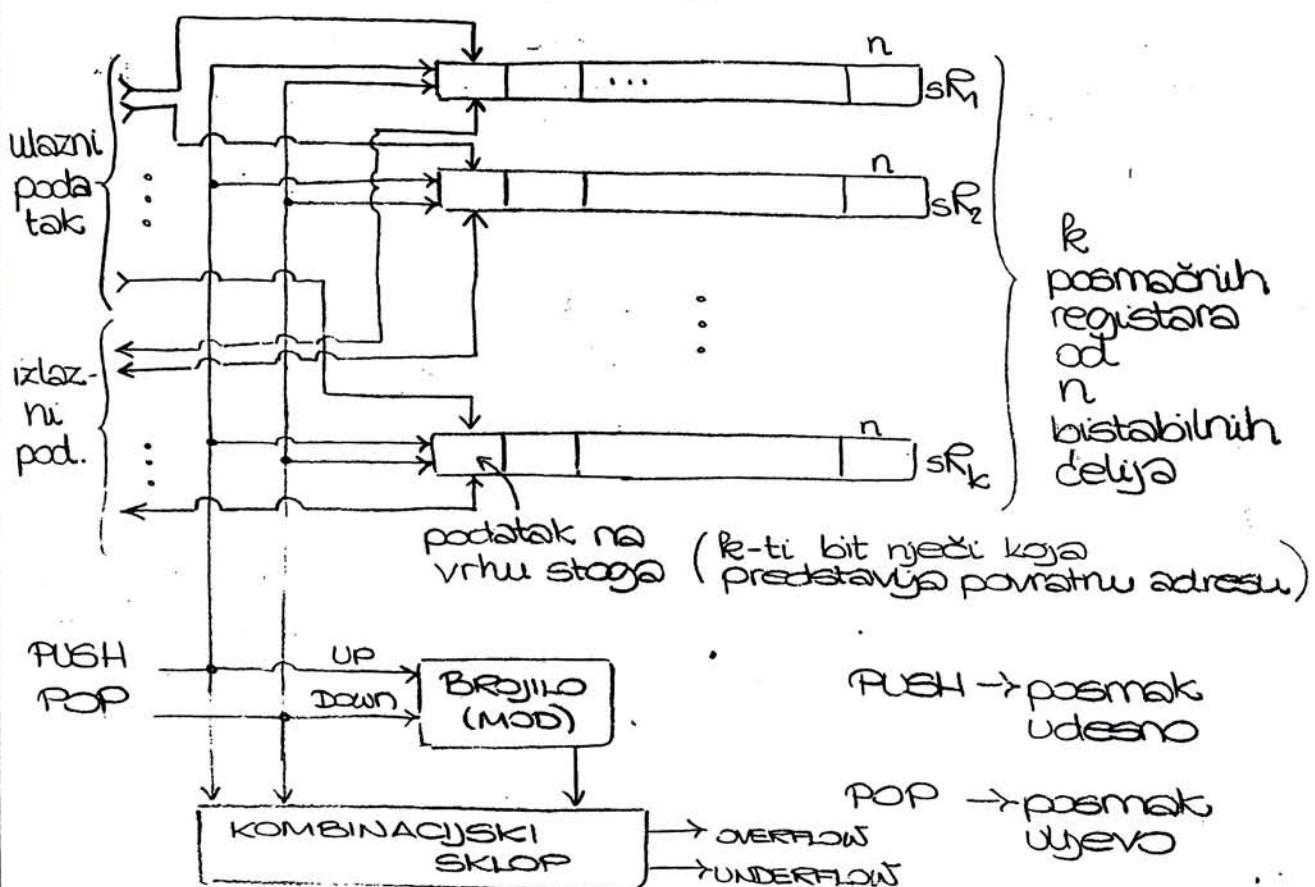
STOS može biti realiziran

1.) programski (u RAM-u treba osigurati memoriju za STOS)
2.) sklopovski

3.) uporaba RAM memorije
kao područje
STOS-a:

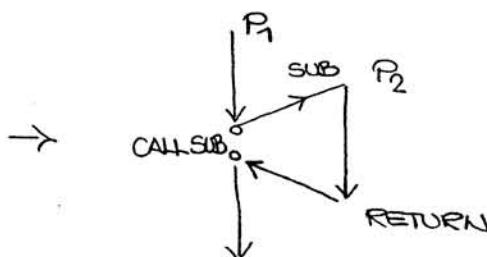


2.) Izvedba stoga od n k -bitnih nješki izveden pomodu posmračnih registara

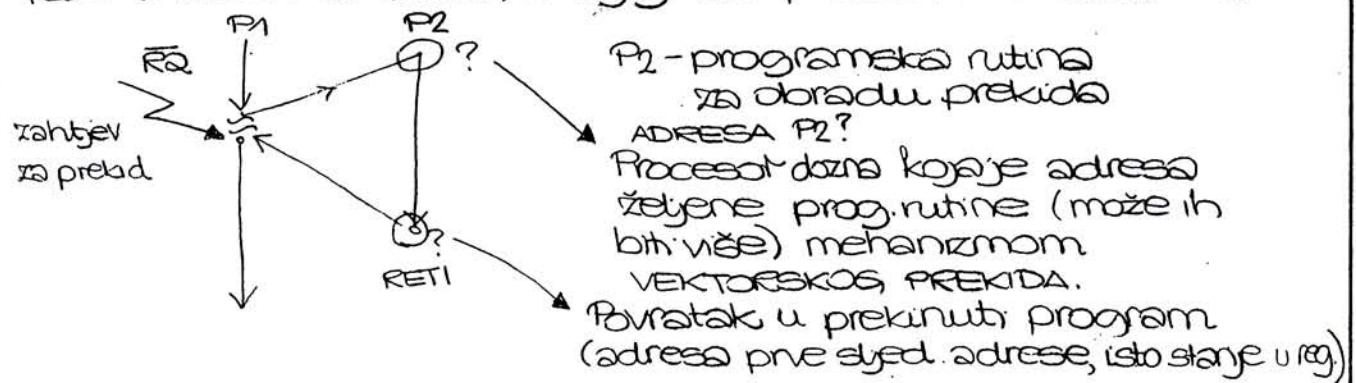


② Prenos upravljanja u slučaju PREKIDA:

(Dosad smo stogom rješili prijenos upravljanja kod pozivanja potprograma, koji se aktivira nakon definirane naredbe CALL SUB, odn. RETURN FROMSUB)



Prekidi su neodšekivani : nema za to definiranog trenutka ili instrukcije. Zahtjev za prekidom javlja se neovisno o fazi (PRIŠAVI ili IZVRŠI) u kojoj se procesor nalazi. (RQ)



Pored povratne adrese, pri prekidu se mora pohraniti i STATUS-registar, koji (bar minimalno) određuje stanje μP u trenutku zahtjeva za prekid (koji je ASINKRON)

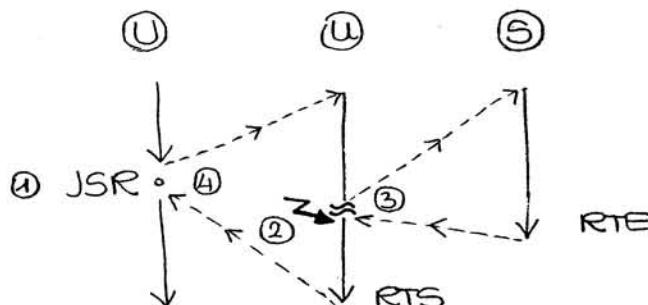
Ipak, μP ne odgovara trenutno na zahtjev, već završava tekuću instrukciju, pa pohrani sadržaj PC i STATUS-reg. Tek onda udovoljava zahtjevu. [minimalni kontekst]

Pri povratku: instr. RET I (return from interrupt) opet se obavljaju "kudanski poslovi" (housekeeping) (dohvat PC i SR)

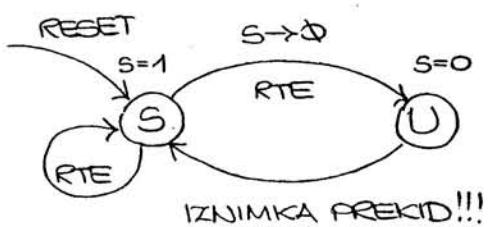
Također, potrebno je i pohranići sadržaje SVH REGISTARA prije prelaska u P₂

PRIMER UPOTREBE STOSA: (MC 68000) → iz knjige
"Napr. arhit. rač."

- procesor je u konzničkom načinu rada
1. • poziva se potprogram
- program se nastavlja izvođenjem potprograma
2. • obrada prekida
3. • vraćanje u potprogram
4. • vraćanje iz potprograma



RTE - povlaštena instrukcija



Povlašteni instr. se mogu izvesti samo u nadglednom načinu rada. Ako je konzničnik pokuša konzničiti, μP de je detektirati i generirati iznimku visoke razine pronteta ("POVREDA PRAVLJENJA")

④ - user mode
(konznički način rada)

⑤ - supervisor mode
(nadgledni način rada)

Programma za obradu prekida obavlja se u modu koji je "viš" od "običnog" (povlaštenje)

U kon. načinu rada
μP je raspoloživ uMANJEN skup instrukcija
(nema instr. koji mogu utjecati na opće stanje μP (RESET) kao i stanje processa u programima drugih konzničkih)

RESET → također iznimka NRP (najviše razine pronteta)

$s=1 \Rightarrow$ zastavica za "supervisor mode"
"MP je u nadglednom načinu rada"
MP u tom modu može konistiti povlaštenu
instr. STOP i prelazi u konisnički način rada.

Ako je μP u konisničkom n.r ne može, konštenjem povlaštenih instr. utjecati na s zaštitom i predi u supervisor mode. Jedini način da pređe tamo je zahvat za prekidom.

Nakon obrade zahtjeva za prekid, vradamo se u user mode instr. (povlaštenom) RTE, no i ne moramo? → Možemo ostati u supervisor modu (ako je zahtjev za prekid došao dok je µP bio u supervisor modu).

KORISNIČKI MODEL UP:

- svi registri vidljivi programeru na razini assembler koda (DataReg, AdrReg, PC, SE, SP→stack pointer)

2 stoga → jedan za user mode
→ jedan za supervisor mode } A7 registr je fizički odvojen

DataReg → reg apde namjene (za Byte - B (8bita)
(D0 - D7) Word - W (16 --) Long - L (32 --))

Adr Reg → nemaju mogućnost pristupa 1 Bytu (8 bita)?
(AD - AT) imaju za W (16 bita) : L (svih 32 bita)

→ AT konzničko kazalo stoga } no μP može u 1 trenutku
 AT^(prim) sistemsko -||- -||- } pristupiti samo 1 od njih ??

PC → 32 bitno progr. br. (24 bita za adresirajuću mem.)

SR → 16 bitni register → předznak (Extend)

- 0 - 7 → konstrukční byte (Z, P, Overflow...) očr
- 8 - 15 → systémski byte (S, I₀, I₁, I₂ zastavice)

SR se cijeli spremi u stog pri pohrani PC i SR-Q.

Pohrana na stog iznosi, da je 6 byteova \rightarrow 4 za PC i 2 za SE pri obradi prekuda (a ne 4, kao pri pozivu potprograma)

MC68000 pohyba 7 razina prekida (zastavice I_0, I_1, I_2)

→ ujet prihvadanja zahtjeva za prekid je da u njemu kod zahtjev za prekid bude vedu od koda zastavica I₁,I₂

→ npr. zahtjev NRP (najviše razinie prioriteta) u ma
kod 111 i uvjek se prihvaca?

Slijedi sluka iz knjige: "Naprednija arhitekturna računala"
STANJE STOGA → str.

SP → počinju na zadnjem punu lokaciju stoga,
= a ne na 1. praznu (to varira o izvedbi μP ff)
Stog (memorija) je **BJEDNO ORIENTIRANA** (pozračjuje 8 po 8 bitova!!)

NAPOMENA:

Instrukcija RTE umjesto RTS u user modu nede se dopustiti i desit će se iznimka?

Instrukcija RTS umjesto RTE u supervisor modu

dopustiti će se i umjesto 6, skinuti 4 byta sa stoga:

Rezultat: skidemo na "adr. instrukcije" koja je načinjena od 2 byta (SR) i 2 viša byta PC \Rightarrow RAZITI NA TO??? *

IZVEDBA UPRAVЉАЧКЕ JEDINICE ZA μP SA SKUPOM OD 8 INSTRUKCIJA

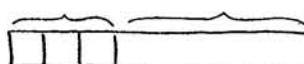
MNE MONIK.

LOAD X
STORE X
* ADD X
* AND X
JUMP X
JUMPZ X
COMP
RSHIFT

OPIS

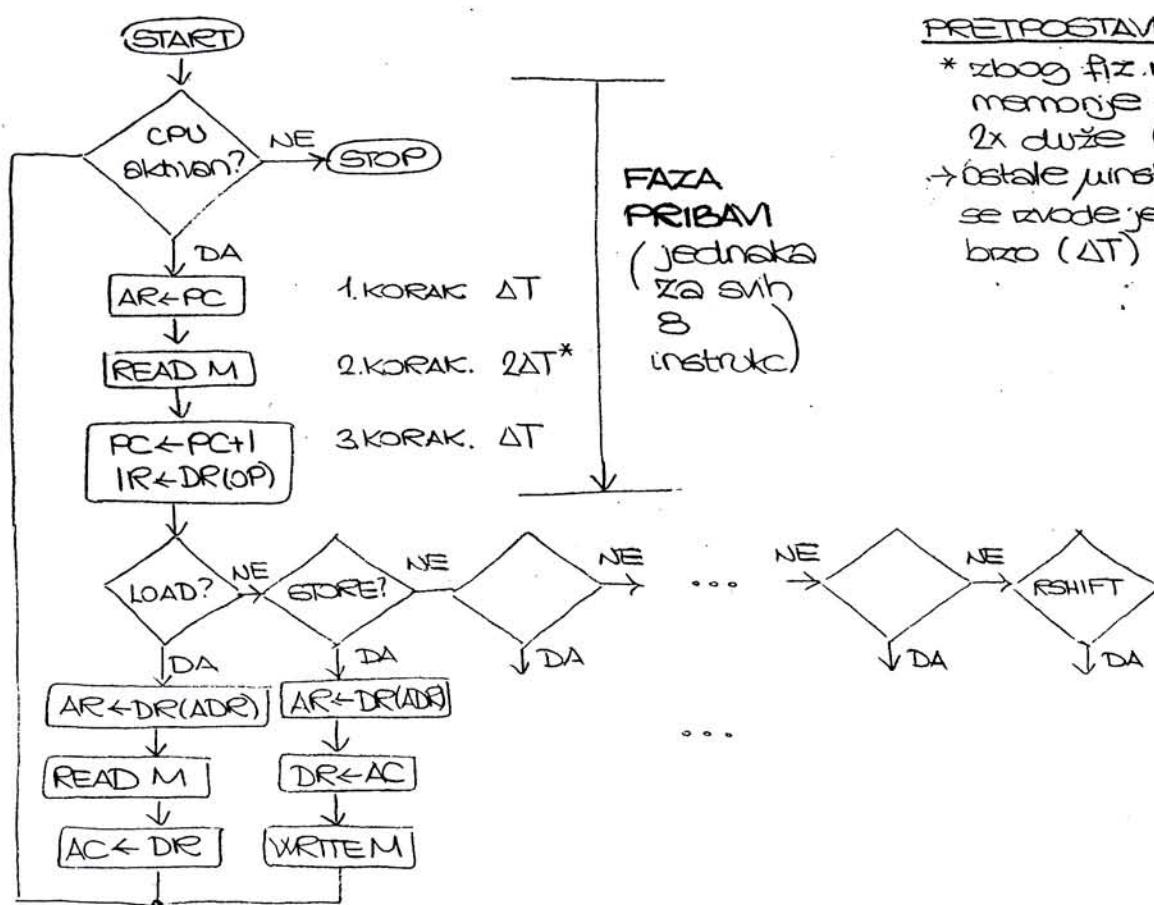
AC \leftarrow M(X)
M(X) \leftarrow AC
AC \leftarrow AC + M(X)
AC \leftarrow AC \wedge M(X)
PC \leftarrow X
If AC = 0 then PC \leftarrow X
AC $\leftarrow \overline{AC}$
Poznati udesno

OP. KOD ADR. POLJE



3bita
su dovoljna za 8 instr?

* RISC zahtjeva da sv. operandi budu dohvadeni iz memo i spremjeni u registre. Ova instr. zbroja operand iz memo \Rightarrow grubo narušavanje RISC načina rada, pa ovo MORA BITI CISC procesor??



PRETPOSTAVKA:

- * zbog fiz. razdvojenosti memorije traje 2x duže ($2\Delta T$)
- \rightarrow ostale instrukcije se izvode jedinstveno brzo (ΔT)

SLIKA 1 : [5.4]

C_3 -čita } radi jednokratnosti, pretpostavljamo 2
 C_4 -piši } distretne linije za RW signal

Upravljački signali

C_0
 C_1
 C_2
 $\frac{C_3}{C_4}$
 C_5
 C_6
 $\frac{C_7}{C_8}$
 C_9
 C_{10}
 C_{11}
 C_{12}

(Mikro)operacije

$AC \leftarrow AC + DR$	
$AC \leftarrow AC \wedge DR$	
$AC \leftarrow \bar{AC}$	
<hr/>	
$DR \leftarrow M(AR)$	(Read M)
$M(AR) \leftarrow DR$	(Write M)
$DR \leftarrow AC$	
$AC \leftarrow DR$	
<hr/>	
$AR \leftarrow DR(ADR)$	
$PC \leftarrow DR(ADR)$	
$PC \leftarrow PC + 1$	
$AR \leftarrow PC$	
$IR \leftarrow DR(OP)$	
$RIGHT-SHIFT AC$	

→ sve mikrooperacije osim Read & Write izvode se u $1 \Delta T$ vremenu,
 a R & W se izvode u $2 \Delta T$.

Mikrooperacija → primitivna, osnovna instrukcija
 realizirana sklopovljem

SLIKA 2 : [1.1]

Jedinstvena faza PRIBAVI (još jedna sličnost s RISC-om,
 tako je ovo CISC !!)

Pretp. Jednaka vel. instrukcije (OP kod + 1adr. instrukcije)

• Izvesti LOAD X:

PRIBAVI :	(jedinstven za sve)	C_{10}, C_3, C_9, C_{11}	$4 \Delta T$
IZVRŠI :	$MAR \leftarrow MDR(X)$	C_7	ΔT
	READ	C_3, C_3	$2 \Delta T$
	$AC \leftarrow MDR$	C_6	ΔT

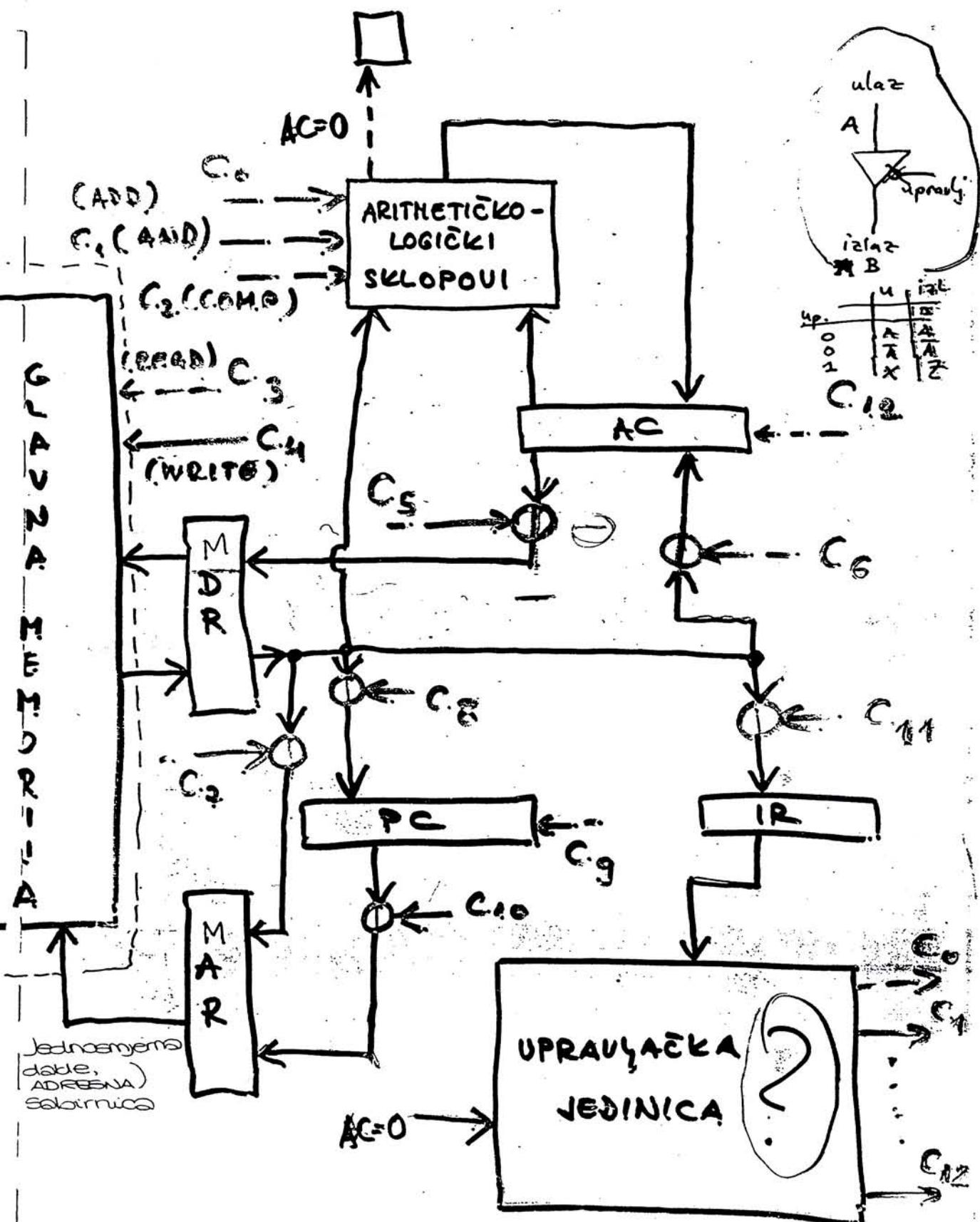
• STORE X:

PRIBAVI :	(jedinstven)	C_{10}, C_3, C_9, C_{11}	$4 \Delta T$
IZVRŠI :	$MAR \leftarrow MDR(X)$	C_7	ΔT
	$MDR \leftarrow AC$	C_5	ΔT
	WRITE	C_4, C_4	$2 \Delta T$

• ADD X

PRIBAVI :	(jedinstven)	C_{10}, C_3, C_9, C_{11}	$4 \Delta T$
	$MAR \leftarrow MDR(X)$	C_7	ΔT
	READ	C_3, C_3	$2 \Delta T$
	$AC \leftarrow AC + MDR$	C_6	ΔT

Svaka od ovih instrukcija (uz AND X) traje $8 \Delta T$

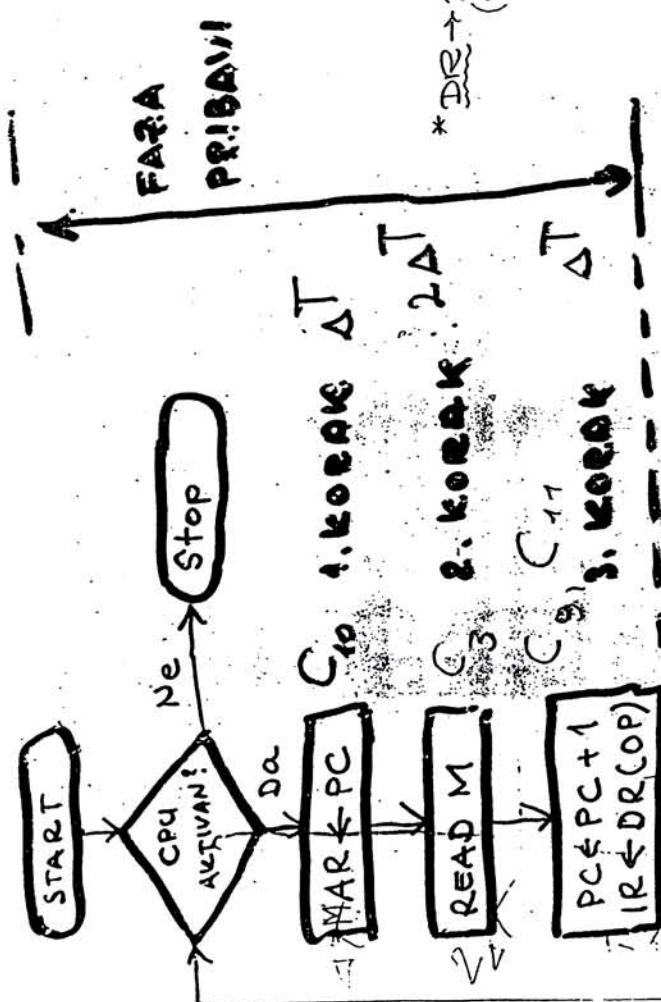


Samo je jedan Mali Ivica!

Prepostavka: SVAKA MIKROINSTRUKCIJA
 CIJENEV READ M i WRITE M
 SE IZVODI U JEDNE
 VREMENSKOU JEDINICI AT.

• READ M i WRITE M
24 HATIJEVaju DIVJE VREMENSKU
JEDINICE (u mode)

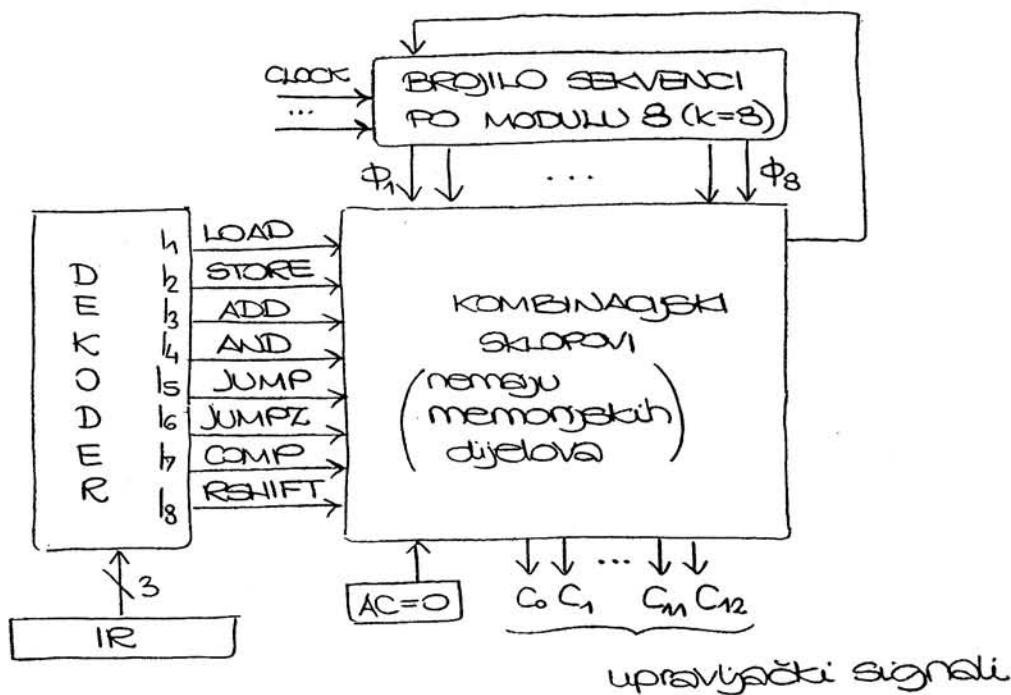
* DR \rightarrow Blistni podneregister
 (prenos operatoda u IR)



<u>JUMP</u>	PRIJAVI: (jedinstven)	C ₁₀ , C ₃ , C ₉ , C ₁₁	4ΔT
	IZVRSI: PC ← MDR(X)	C ₈	ΔT
→ ne zahtjeva komunikaciju s mem., pa trajektorija: 5ΔT (krade su)			zato je brojilo

"Spore" instrukcije (npr. ADD, STORE, ...) trebaju 8, a "brze" - " " (npr. JUMP) zahtjevaju samo 5 vremenskih jedinica
(Brojilo sekvenci po modulu 8 vodeno signalom vremena, vodenja daje perioda ΔT broj instrukcije)

STRUKTURA UPRAVILJAČKE JEDINICE



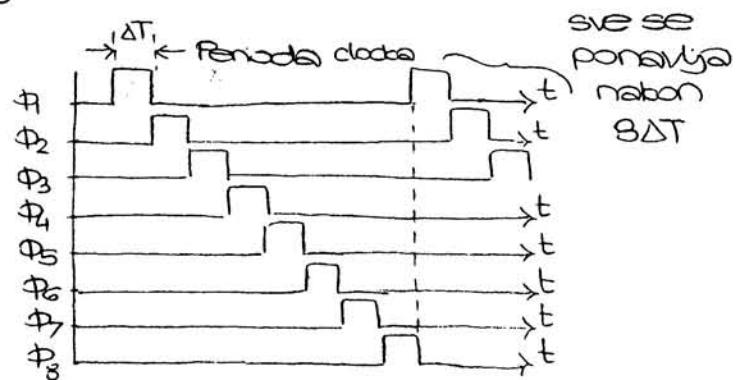
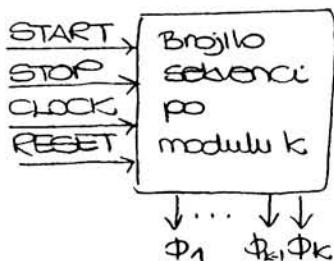
upravljački signali

Upravljački signali mogu se opisati LOGIČKOM JEDNADŽBOM odlike:

$$C_i = \sum_j^k (\phi_j \cdot \sum_m I_m) \quad i = 0, \dots, 12 \quad I_m - \text{izlaz iz dekoder-a}$$

Ako instrukcija zahtjeva $j < k$; ($k=8$) koraka, brojilo se može resetirati nakon j (odmah prekid faze PRIJAVI i kreni na izvršenje)

BROJILO PO MODULU k:



$$C_i = \sum_j (\phi_j \cdot \sum_m l_m) \rightarrow \text{log. jedn.} \Rightarrow \sum = \text{ILI operacija}$$

$\cdot = \text{I operacija}$

Faza PRIJAVI:

$$\begin{array}{ll} \phi_1 & : \text{MAR} \leftarrow \text{PC} \\ \phi_2, \phi_3 & : \text{READ M} \\ \phi_4 & : \text{PC} \leftarrow \text{PC} + 1 \\ & \quad \text{IR} \leftarrow \text{MDR(OP)} \\ \downarrow & \uparrow \end{array}$$

$$\begin{array}{l} C_0 \\ C_3 \\ C_9, C_{11} \end{array}$$

Veza diskretnih trenutaka ϕ_1, \dots, ϕ_8 s operacijama.

$$\begin{aligned} l_i \cdot \phi_5 &= C_7 \\ C_7 &= \phi_5 \cdot l_1 + \phi_5 \cdot l_2 = \phi_5(l_1 + l_2) \end{aligned} \quad (l_i \rightarrow \text{izlaz i-te instr. iz dekodera})$$

Faza IZVRŠI:

① Instr. LOAD:

$$\begin{aligned} l_i &= l_1 \\ C_7 &= \phi_5 \cdot l_1 \\ C_3 &= \phi_6 \cdot l_1 + \phi_7 \cdot l_1 \Rightarrow \text{za READ } (C_3 \text{ traje LAT}) \\ C_6 &= \phi_8 \cdot l_1 \end{aligned}$$

$\phi_1, \dots, \phi_8 \rightarrow \text{izlaz iz brojila setvencija}$

$l_1, \dots, l_8 \rightarrow \text{izlaz iz dekodera}$

$C_0, \dots, C_{12} \rightarrow \text{izlaz iz upravljič. jedinice} = f(\phi_1, \dots, \phi_8, l_1, \dots, l_8)$

② Instr. STORE:

$$\begin{aligned} l_i &= l_2 \\ C_7 &= \phi_5 \cdot l_2 \\ C_5 &= \phi_6 \cdot l_2 \\ C_4 &= \phi_7 \cdot l_2 + \phi_8 \cdot l_2 \end{aligned}$$

③ Instr. ADD

$$\begin{aligned} l_i &= l_3 \\ C_7 &= \phi_5 \cdot l_3 \\ C_3 &= \phi_6 \cdot l_3 + \phi_7 \cdot l_3 \\ C_0 &= \phi_8 \cdot l_3 \end{aligned}$$

④ Instr. AND

$$\begin{aligned} l_i &= l_4 \\ C_7 &= \phi_5 \cdot l_4 \\ C_3 &= \phi_6 \cdot l_4 + \phi_7 \cdot l_4 \\ C_1 &= \phi_8 \cdot l_4 \end{aligned}$$

⑤ Instr. JUMP

$$\begin{aligned} l_i &= l_5 \\ C_8 &= \phi_5 \cdot l_5 \\ j = 5 &\rightarrow \text{nakon toga se računa brojilo} \\ &\quad (\ubrzoje izvodenja instrukcija) \end{aligned}$$

C_3 je aktivan za ①, ③ i ④ instr.

Faza izvrši:

$$C_3 = \phi_6 \cdot l_1 + \phi_7 \cdot l_1 + \phi_6 \cdot l_3 + \phi_7 \cdot l_3 + \phi_6 \cdot l_4 + \phi_7 \cdot l_4 + \dots \quad (\text{do } l_8)$$

→ Faza PRIJAVI i IZVRŠI:

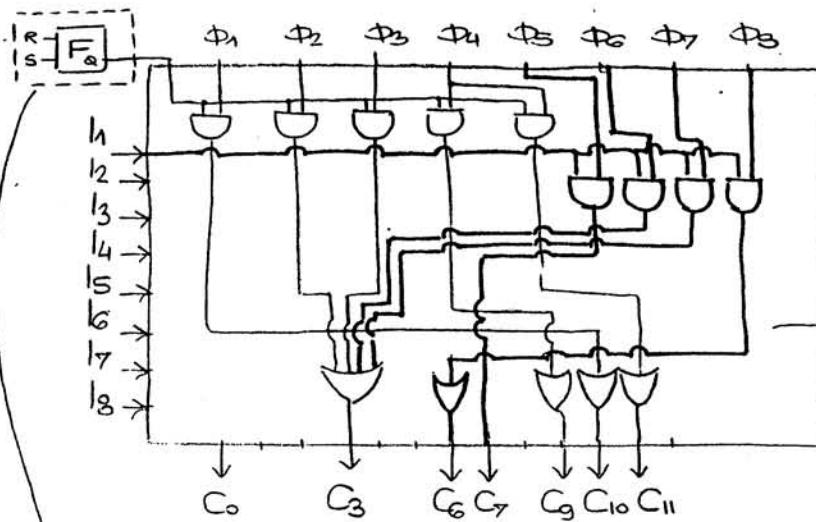
$$C_3 = \phi_2 + \phi_3 + \phi_6(l_1 + l_3 + l_4) + \phi_7(l_1 + l_3 + l_4) \Rightarrow \begin{array}{l} \text{OBNIK} \\ \text{KOŠICE} \\ \text{JEDNADŽBE} \end{array}$$

Kombinacijski sklop je jednostavna realizacija logičke jednadžbe ??

Kakva je to realizacija?

Npr. za fazu PRIJAVI:
(olovkom)

Samo je jedan Multi Input!



FAZA PRIJAVI:

$$\begin{aligned} C_{10} &\Rightarrow \phi_1 \\ C_3 &\Rightarrow \phi_2 + \phi_3 \\ C_9 &\Rightarrow C_{11} \Rightarrow \phi_4 \end{aligned}$$

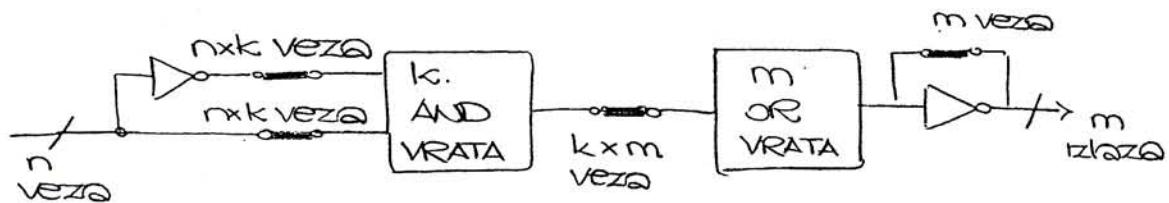
Vrlo pravilna struktura
I i III sklopova
(I-ravnina
III-ravnina)

FETCH bistabil: Σ FAZU PRIJAVI $Q = 1$
 Σ FAZU IZVRŠI $Q = Q$

Σ LOAD Instrukciju: nadopuni komb. sklop (+ izlaz l1)
(kemijskom) resetira se F bistabil $\Sigma\Sigma$ ($Q = 0$)

Komb. sklop: veze izmedu I RAVNINE
III RAVNINE } PLA ??

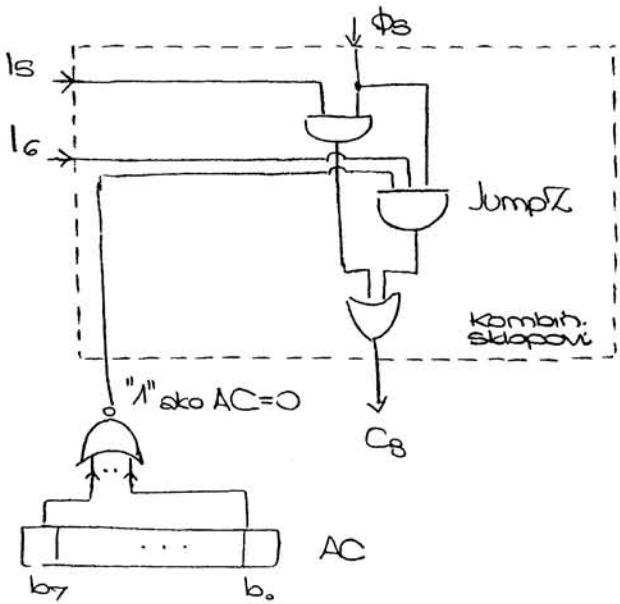
PLA - Programmable Logic Array



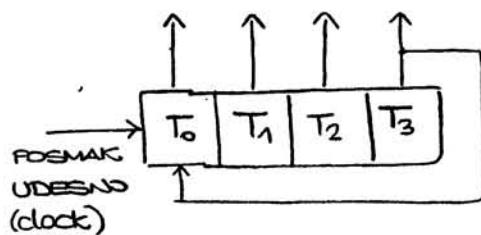
JUMP (ls) } Faza IZVRŠI:
JUMPE (le) }

! Isprtni zadatok:

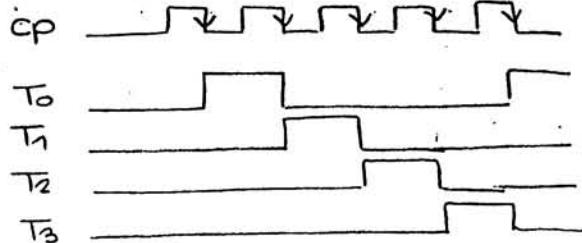
Σ dodan atom. B
(def. novi instr.) ili
wedemo novi instr.:
→ promjeni broj sekvenci
→ - - Inst. dekoder



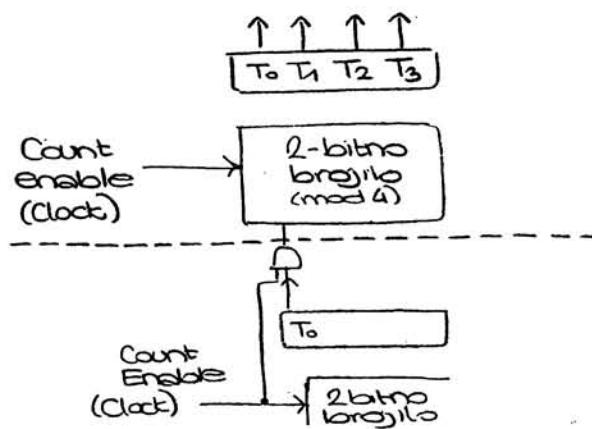
1) PRESTENASTO BROJILLO:



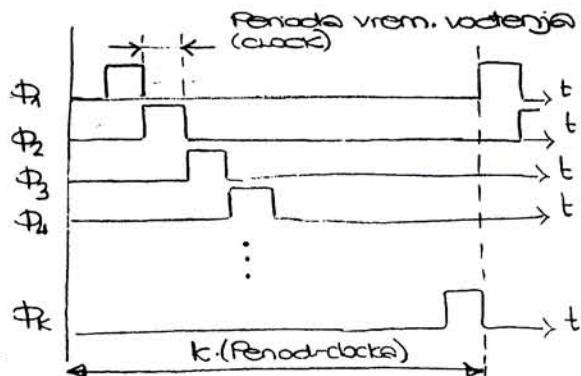
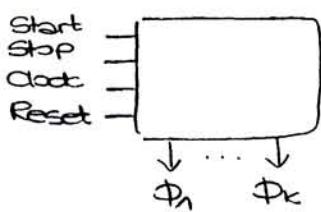
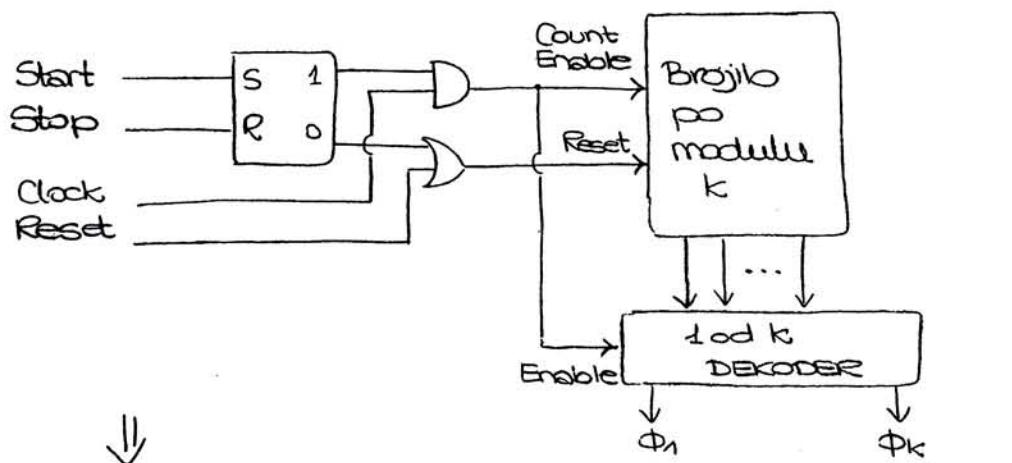
Prestavka: poč.vrij. 100
Vrem. dijagram:



2) BROJILLO I DEKODER:

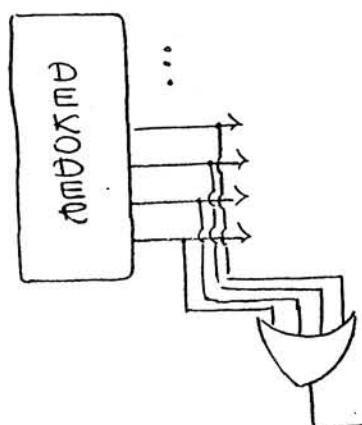


BROJILLO SEKVENCI PO MODULU k (Modulo-k sequence counter)



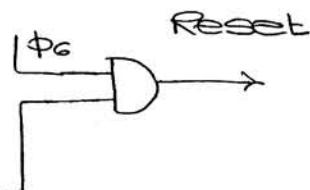
Sklop za resetiranje

$J=5$

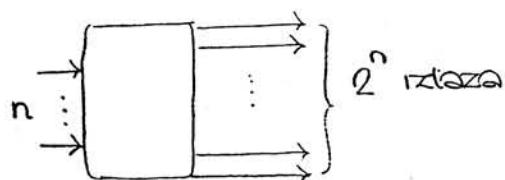


Objašnjenje:

ako su dekodirane instrukcije
koje tražu $<8\Delta T$, odnosno
zahtevaju RESET, njihovi izlazi
spajaju se u 1H sklop (bit koji od
njih, uč aktivan je RESETIRAN brojilo)
Posljedica: Brojilo broji samo $5\Delta T$



DEKODER: ("1 out of 2^n ")



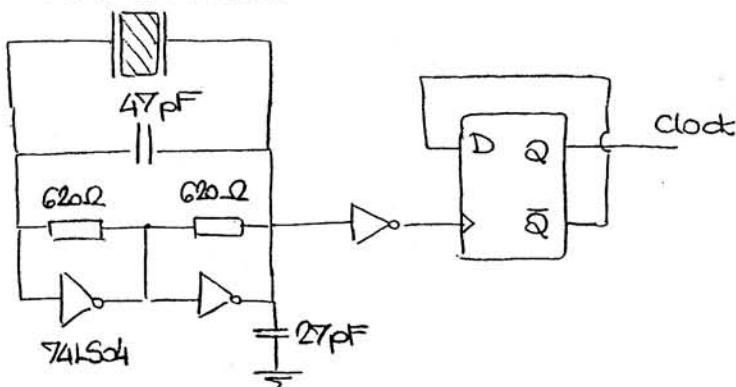
Npr. $n=3$

x	y	Σ	D ₀	D ₁	D ₂	D ₃	D ₄	D ₅	D ₆	D ₇
0	0	0	1	0	0	0	0	0	0	0
0	0	1	0	1	0	0	0	0	0	0
0	1	0	0	0	1	0	0	0	0	0
0	1	1	0	0	0	1	0	0	0	0
1	0	0	0	0	0	0	1	0	0	0
1	0	1	0	0	0	0	0	1	0	0
1	1	0	0	0	0	0	0	0	1	0
1	1	1	0	0	0	0	0	0	0	1

REALIZACIJA:

2MHz signal nizam/takt

4.0 MHz kristal



MIKROPROGRAMIRANJE → programska izvedba UPRAVlj.JED.

CPU = ALU + UPRAVLJAČKA JEDINICA

Mikrooperacija: elementarna (nedjeljiva) operacija
u potpunosti sklopovski podržana

Npr:

- prenos sadržaja između 2 registra
- aktiviranje sklopa u ALU
- resetiranje bistakala

Izvođenje instrukcije u strognom jeziku → niz mikrooperacija

$$\begin{array}{l} \text{ADD} \rightarrow MDR \leftarrow M(\text{MAR}) \\ \quad \quad \quad \text{IR} \leftarrow \text{MDR} \\ \quad \quad \quad \text{PC} \leftarrow \text{PC} + 1 \\ \quad \quad \quad D \leftarrow \text{opa} \\ \quad \quad \quad \text{MAR} \leftarrow \text{MDR} \\ \quad \quad \quad : \end{array}$$

Mikroinstrukcija: kodirano predstavljena 1 ili više mikrooperacija

Mikroprogram: sklad (niz) mikroinstrukcija kojem su pohranjene u upravljačkoj (uprogramskoj) memoriji.

Često se mikroinstrukcija pohranjena u uprav. memoriji naziva mikronjed (eng. microword)

↓

ili upravljačka njed

DODATAK 1: Wilkesova shema uprav.jed.

FAZE MIKROPROGRAMIRANJA:

1.FAZA: upotreba diodnih matrica

- vrijeme pristupa diod. matrica $\sim 0.5\mu\text{s}$
- memorjski ciklus $> 10\mu\text{s}$

2.FAZA: kasnih 50.-tih godina

gl. memorija → FERITNE JEZGRICE

diodne matrice nisu dovoljno brze

/uprogramiranje samo za emulaciju (opravljavanje)/
2. radnata (ne u realnom vremenu)

3.FAZA: krajem 60.-tih godina

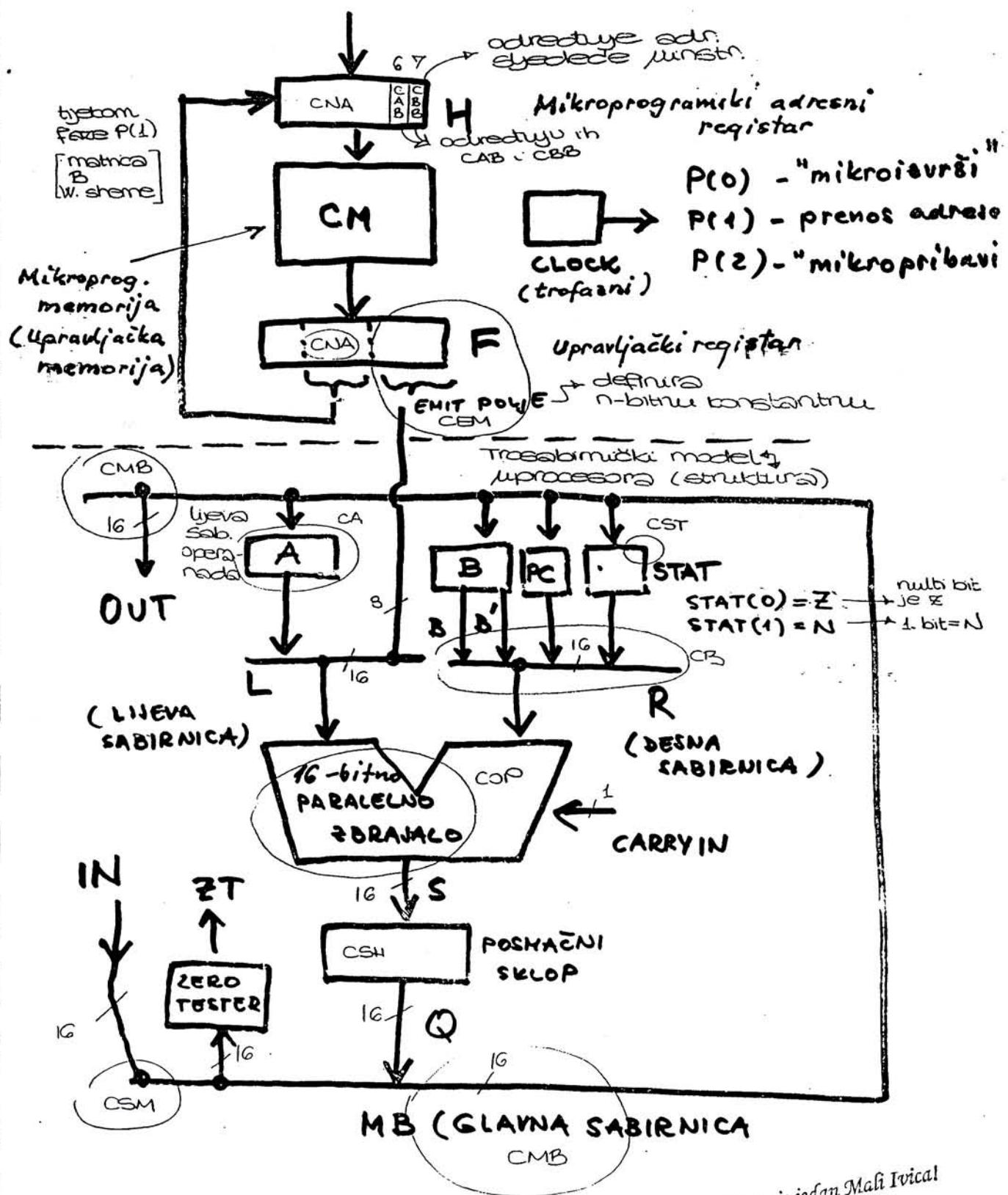
pojava brzih bipolarnih memorija za izvedbu
upravljačke jed (FAZA OPRAVAKA uprog. u primjeni)

Mikroprogramiranje → najbolja metoda realizacije
upravl.jedinice (LSI, VLSI, ...)

DODATAK 2:

9

MIKROPROGRAMIRANI CPU (MODEL)



三

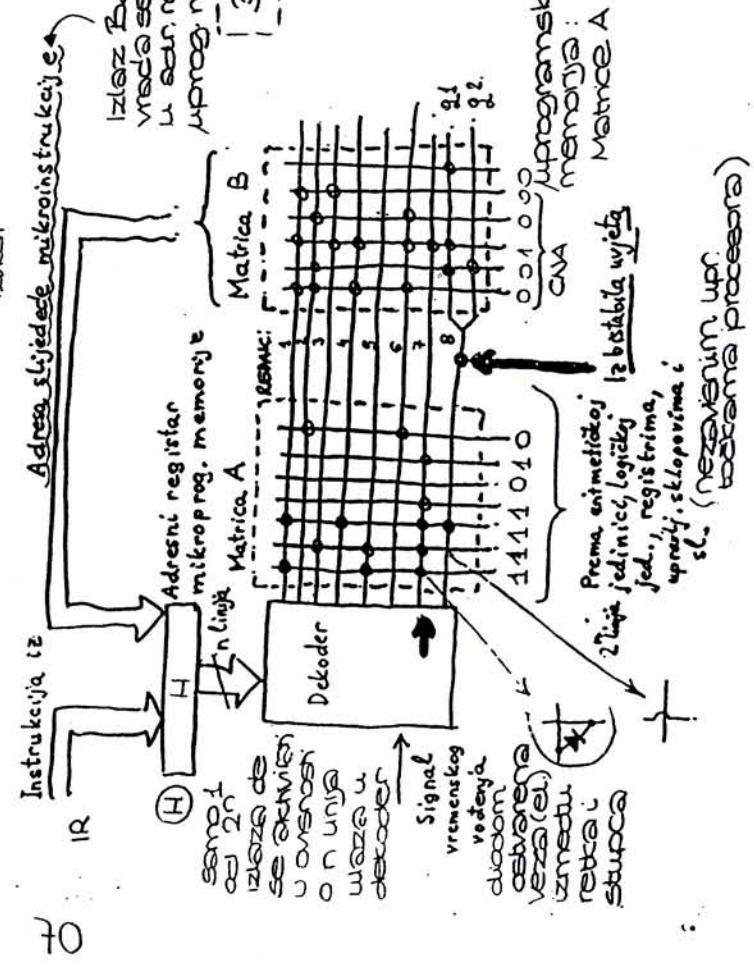
DODATAK 1
M.V. Wilkes (1956)

elied kuperacio povezova kombinaciem a matice B

metropag ramirouje:

Primer Instrucción ADD

1. mikrooperacija : $MDR \leftarrow M(MAR)$
2. mikrooperacija : $IR \leftarrow MDR, P \leftarrow P+1$
3. mikrooperacija : $D \leftarrow \text{opc}$
4. mikrooperacija : $MAR \leftarrow AP, F \leftarrow 0, E \leftarrow 1$
5. mikrooperacija : $TR1 \leftarrow MDR, TR2 \leftarrow ACC$
6. mikrooperacija : $ACC \leftarrow ADDER$
7. mikrooperacija : $E \leftarrow 0, F \leftarrow 1, MAP \leftarrow P$



CHORAL HYMNS AND HYMNEA (MUSICAL EDITION)

(3) @ mitroinststrukcija / format :

2bito CA - Vera sa sabirnicom L.

00 01 $\frac{1}{(0-3)} \frac{8-15}{\sqrt{5}} = 0-EEM$; *hema pijnosa podatika*

neg → Sbitne konst.
(CEN) se ; konstante se smještava

$$10 \cdot L(0.7, 8-15) = F(\text{GMAZ}) \text{ konstant}$$

se encontra na MS bait

44 $L = A \rightarrow$ ~~secular~~ A proposal re L
; Sabirnice L

C6 - Vena sa řešením B

000 6 : name name name

R = B, *Weltwirtschaftszeitung* 1904, 1.

Sa
040 R = B
041 D = BC

mo je je
R = C
R = STAT
100

104 } ne boriste se!

卷之二

nestavak na dozvuku 3(b)

Samo je jedan Mali Ivica!

nestavak na dodatku 3(b)

COP - microprocessor Paralleling -

30

CM - upravljačka memorija, **MC - mitskočna memorija**

00	zbroji sa $C_{in} = 0$
01	zbroji sa $C_{in} = 1$
10	ne koristi se
11	

<u>CSH</u>	- sprawdzanie sklejek na poimaki
00	$M_B = Q$, $Q = S$; nema poimaka
01	$M_B = Q$, $Q = shr S$
10	$M_B = Q$, $Q = sh \emptyset S$
11	$M_B = N$, przynosząc wzorzec substytucji na elanum sarmaticum rodziców
<u>CMB</u>	- vera planice sabinice (<u>NB</u>) ostatnia komponent; precyza

```

    A → MB
    B → MB
    PC ← MB
    STAT ← MB
    OUT = MB

```

110 } ne konisti se
 } 111

CM
256 rijetki
čakavina riječi i 32. b.

Cm

256 rijeci
četvrtina rijeci 82-1)

255

Mikroprogramski adresni register H

MB = IN, Q = 147,5
Bemerkung: nur der Bereich zwischen den beiden ersten Säulen ist ausgebaut.

CMB - vera gleame sabirnice (MS)
ostalih komponenti prenosa

(Ls 8) ॥ ४

A → MB
 B → MB
 PC → MB
 STAT → MB
 OUT = MB

Namijenjeno polju CNA (sljedeća
adresa)

Bitteri H (6,7) određuju se poljima CAB : CDA

CABA : CDA

CAB - utjecaj na H(C)

(3c)

00	$H(C) \leftarrow 0$
01	$H(C) \leftarrow 1$
10	$H(C) \leftarrow STAT(0)$
11	$H(C) \leftarrow STAT(1)$
	$STAT(0) = Z$
	$STAT(1) = N$

utjecaj na
bitove realne
veličine

(3c)

CST - utjecaj na H(C)

(3c)

00	ϕ ; nema utjecaj [x]
01	$IF(ZT=0) THEN (STAT(0) \leftarrow 0)$ <small>ZTOZ TESTNO!</small>
10	$STAT(0) \leftarrow 1$ <small>ELSE</small>
11	$STAT(1) \leftarrow MB(0)$ <small>[x] TESTNO!</small>

Zastavice

Z

N

CBB - utjecaj na H(P)

(3c)

00	$H(P) \leftarrow 0$
01	$H(P) \leftarrow 1$
10	$H(P) \leftarrow STAT(1)$
11	$H(P) \leftarrow MB(0); MB(0) je$; najmanji vrijiji bit ; 16-bitne vrijednosti ; uvođe se u "MB" ; sabirnici ; rezultat je > 0 ; rezultat je < 0 ; MB rezultata smještan u H(P)

00	$IF(ZT=0) THEN (STAT(0) \leftarrow 0)$ <small>ZTOZ TESTNO!</small>
10	$STAT(0) \leftarrow 1$ <small>ELSE</small>

Tablica adresa ZT; MB(0)

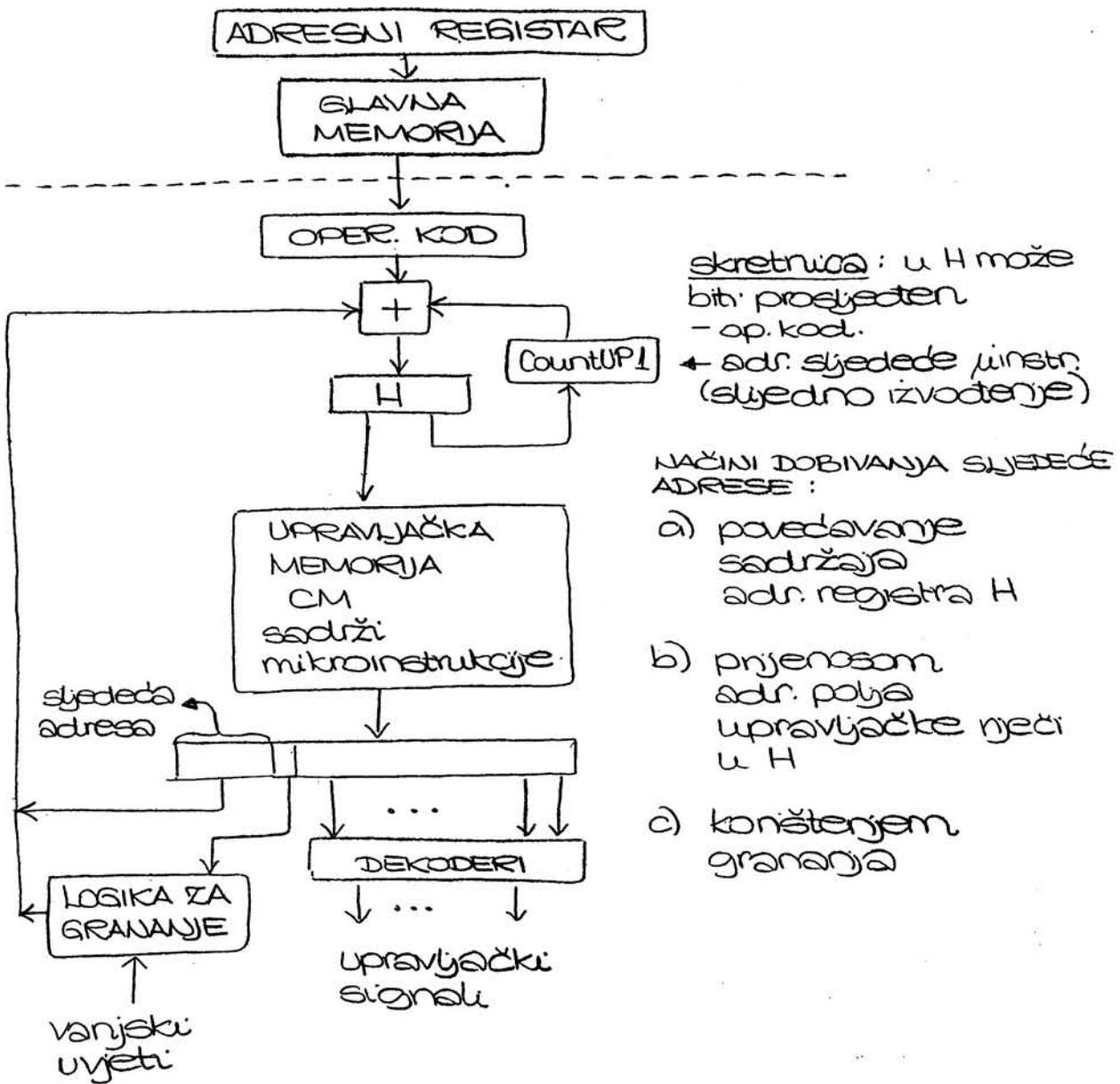
ZT'	MB(0)	rezultat na H(B) = 0	rezultat na H(B) = 1
0	0	0	1
1	0	1	0
0	1	1	1
1	1	0	1

CNA = polje sljedeće adrese
matrice B
u V. smjeru
CMB → H(0-5)
CEM - 8-bitna konstanta CEMIT POLJE

2 razinska realizacija uprav. jed.
 - μprogramiranje } (2 koncepta)
 - nano - "

Npr: Motorola 68020 //

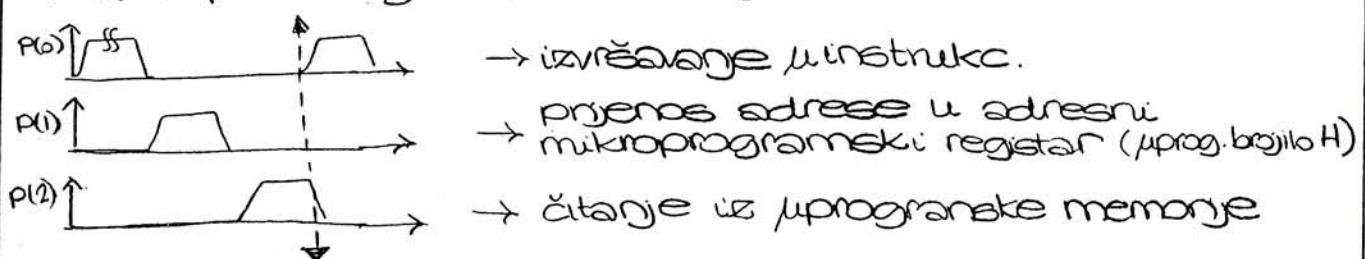
OPIS μPROGRAMIRANE UPRAV. JED.



DODATAK 2: model CPU

DODATAK 3: format μinstrukcije (a,b : c)

μprogram. uprav. jed. koja koristi uvjete grananja u μprogramima ima 3 fazni signal vrem. vodenja



1. RAZRED : Neprogramirajući skup instrukcija (fixni skup instr.)
2. -- : Mikroprogramirajući procesori
sam konstrukcija programira skup instrukcija
3. -- : kombinacija 1. i 2. razreda

Zadak :

Napiši program (za instr. CBR : Conditional Branch) za uspoređivanja 2 brojeva (jedan u reg A, drugi u reg B) koji su predstavljeni u 2^k :

→ Ako :

- A > B : nema promjene u PC registru
- A = B : PC se uvedava $\rightarrow 1$ (sadržaj PC)
- A < B : PC se $\rightarrow 11$ $\rightarrow 2$

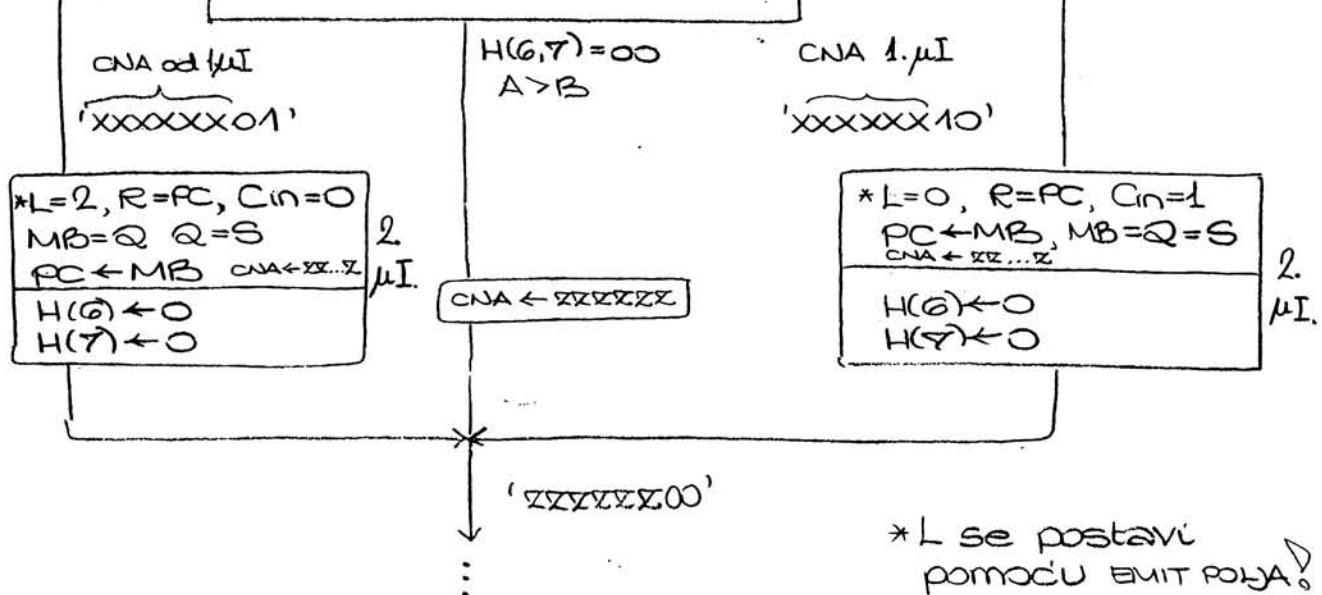
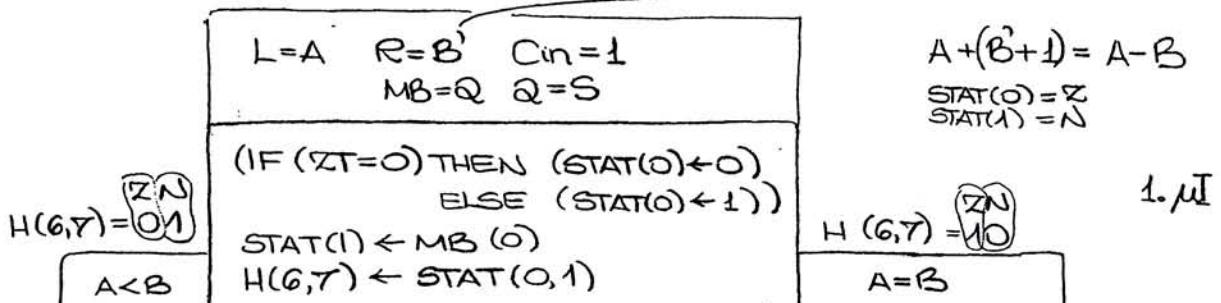
MNEMONIK : CBR

```
IF A > B THEN (NOP) PC=PC
IF A = B THEN PC=PC+1
IF A < B THEN PC=PC+2
```

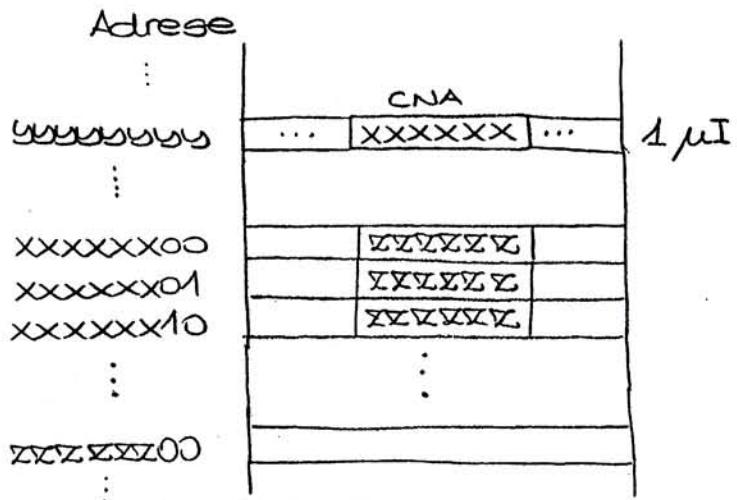
Pretp. : početna adresa instrukcije je predstavljena izravno strojnim kodom instrukcije (u praksi to nije slučaj)

'yyyyyyyyyy' = OP

→ Jedinični kompl. 1'



'yyyyyyyy'
op. kod. instr. CSE



Po FAZAMA:

/START(0)*P(0)/ $P_p \leftarrow 0, G \leftarrow 0$; G = upravljački reg.
koji osigurava pravilan početak sekvence s odgovarajućim adresom H

/G'*P(1)/ $H \leftarrow 'yyyyyyyy', G \leftarrow 1$
/G *P(2)/ $F \leftarrow CM(H)$

upribavljene i instrukcije
miznisi:

/CA(3)*P(0)/	$L=A$
/CB(2)*P(0)/	$R=B$
/COP(1)*P(0)/	$Cin=1$
/CSH(0)*P(0)/	$MB=Q, Q=S$
/CMB(0)*P(0)/	NOP
/CST(3)*P(0)/	(IF ($STAT=0$) THEN ($STAT(0) \leftarrow 0$) ELSE ($STAT(0) \leftarrow 1$))
	$STAT(1) \leftarrow MB(0)$

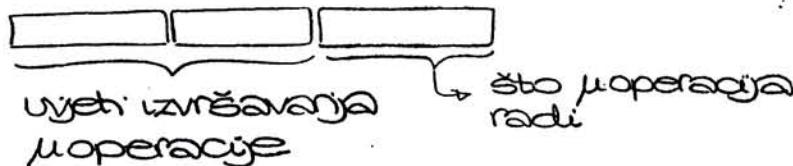
Dosad, u fazi P(0), definiran je format:

broj bitova: 2 3 2 2 3 2 2 2 6 8
 CA CB COP CHS CMB CAB CBB CST CNA CEM
 FORMAT 1. μI : 11 010 01 00 000 10 10 11 ,

/CAB(2)*P(1)/	$H(G) \leftarrow STAT(0)$	$\begin{cases} stat(0)=Z \\ stat(1)=N \end{cases}$
/CBB(2)*P(1)/	$H(Y) \leftarrow STAT(1)$	
/G * P(1)/	$H(O-5) \leftarrow F(CNA)$	
/G * P(2)/	$F \leftarrow CM(H)$	\downarrow $xxxxxx$

ovim je definirana adresa sljedeće instrukcije
i prelazimo na sljedeću.
(FAZA UPRIJAVI sljeduća instr. je već obavljena u P(1); P(2)
PRETHODNE instrukcije ??)

CDL → Computer Design Language



Za slučaj $A = B$, instrukcija je:
 Njena adresa je $001000\boxed{10}$ → z: N zastavice
 CNA preth. instr.

$$\underline{PC = PC + 1}$$

/KA(1)*P(0)/
 /KB(3)*P(0)/
 /KOP(1)*P(0)/
 /KSH(0)*P(0)/
 /KMB(3)*P(0)/
 /KST(0)*P(0)/
 /KAB(0)*P(1)/
 /KBB(0)*P(1)/
 /G*P(1)/
 /G*P(2)/

L=0 L (0-7,8-15) = 0-F(EM) (CEM)
 R=PC
 Cn=1
 MB=Q Q=S
 PC ← MB
 NOP
 H(6) ← 0
 H(7) ← 0
 H(0-5) ← F(CNA) ; F(CNA) = 'XXXXXX'
 F ← CM(H)

bilo nego KA(0) jer ako je L neaktiviran, ne mora znati da je Q=0

Za slučaj $A < B$

, adresa je $001000\boxed{01}$ $\underline{PC = PC + 2}$

/KA(1)*P(0)/
 /KB(3)*P(0)/
 /KOP(0)*P(0)/
 /KSH(0)*P(0)/
 /KMB(3)*P(0)/
 /KST(0)*P(0)/
 /KAB(0)*P(1)/
 /KBB(0)*P(1)/
 /G*P(1)/
 /G*P(2)/

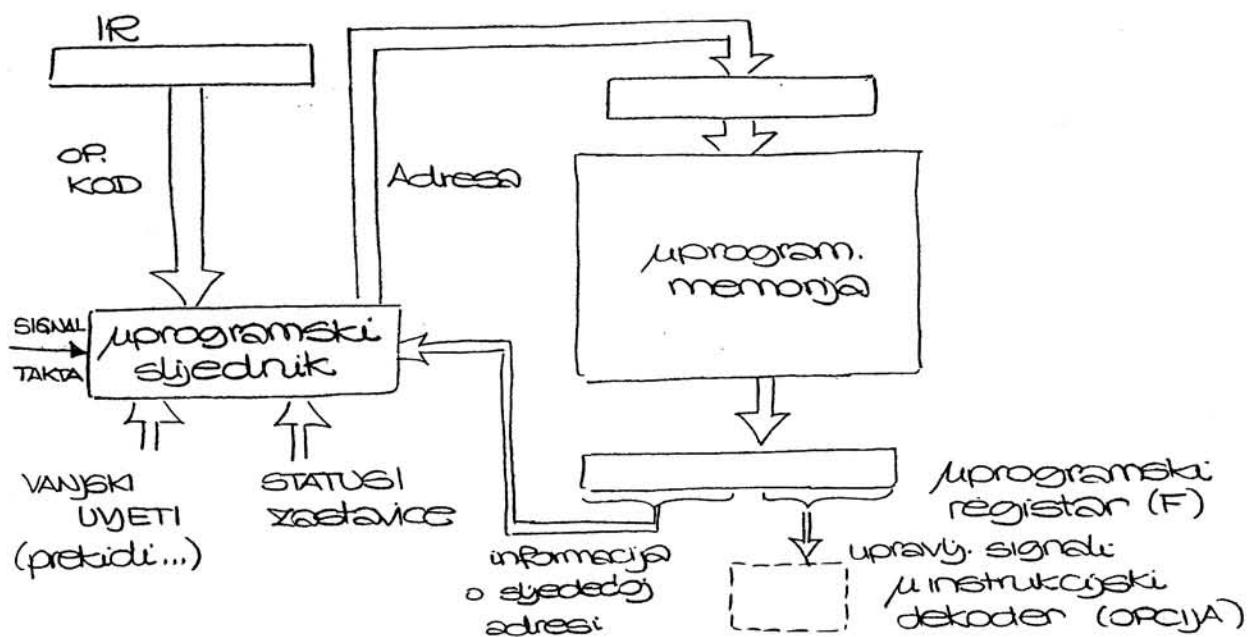
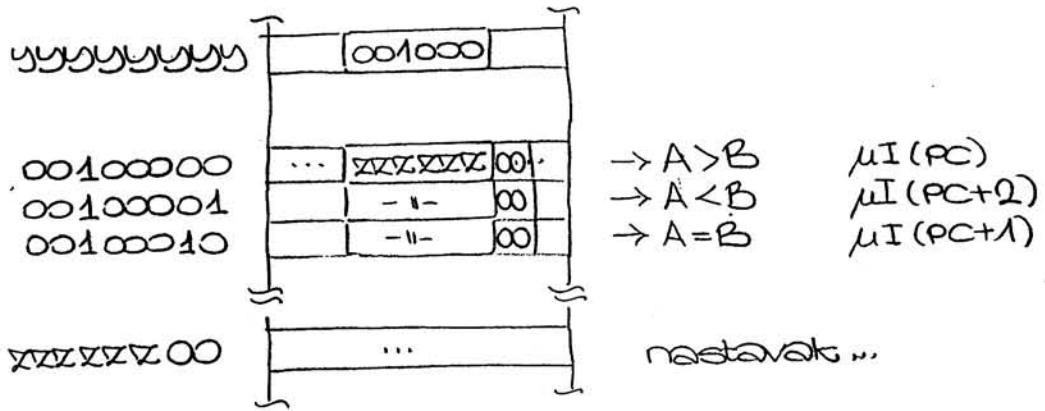
L (0-7,8-15) = 0-F(EM) ; F(EM)=2 !!!
 R=PC
 Cn=0
 MB=Q Q=S
 PC ← MB
 NOP
 H(6) ← 0
 H(7) ← 0
 H(0-5) ← F(CNA) ; F(CNA) = 'XXXXXX'
 F ← CM(H)

Za slučaj $A > B$, adresa je $001000\boxed{00}$

$$\underline{PC = PC}$$

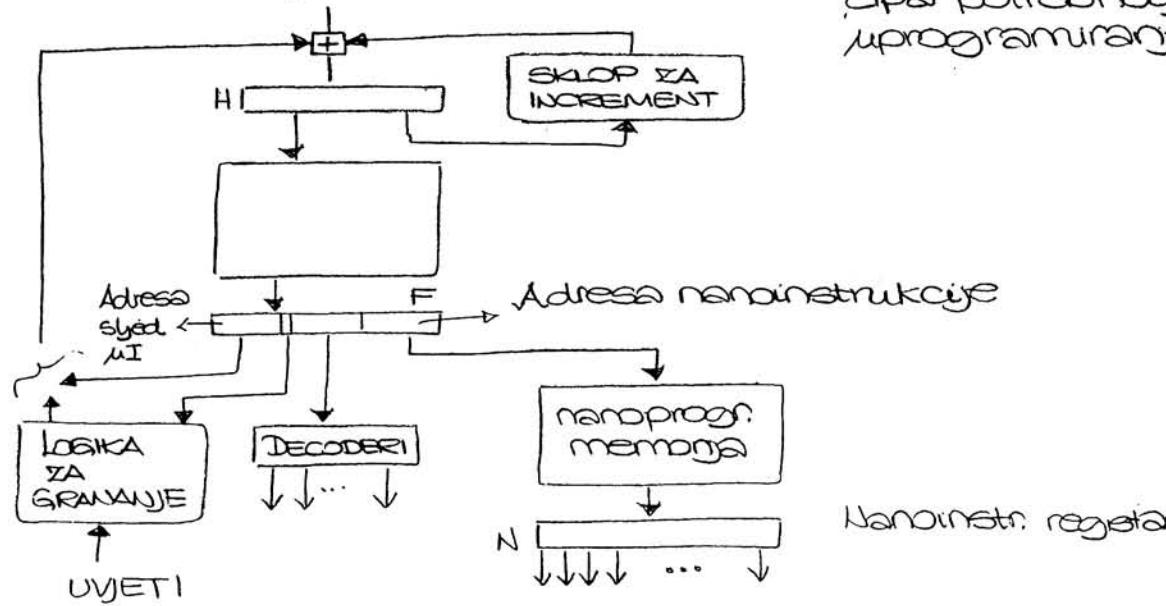
/KA(0)*P(0)/
 /KB(0)*P(0)/
 /KOP(0)*P(0)/
 /KSH(0)*P(0)/
 /KMB(0)*P(0)/
 /KST(0)*P(0)/
 /KAB(0)*P(1)/
 /KBB(0)*P(1)/
 /G*P(1)/
 /G*P(2)/

} nema prijenosa podataka
 H(6) ← 0 odnosno STAT(0) jer Z=0
 H(7) ← 0 -- STAT(1) jer N=0
 H(0-5) ← F(CNA) ; F(CNA) = 'XXXXXX'
 F ← CM(H)



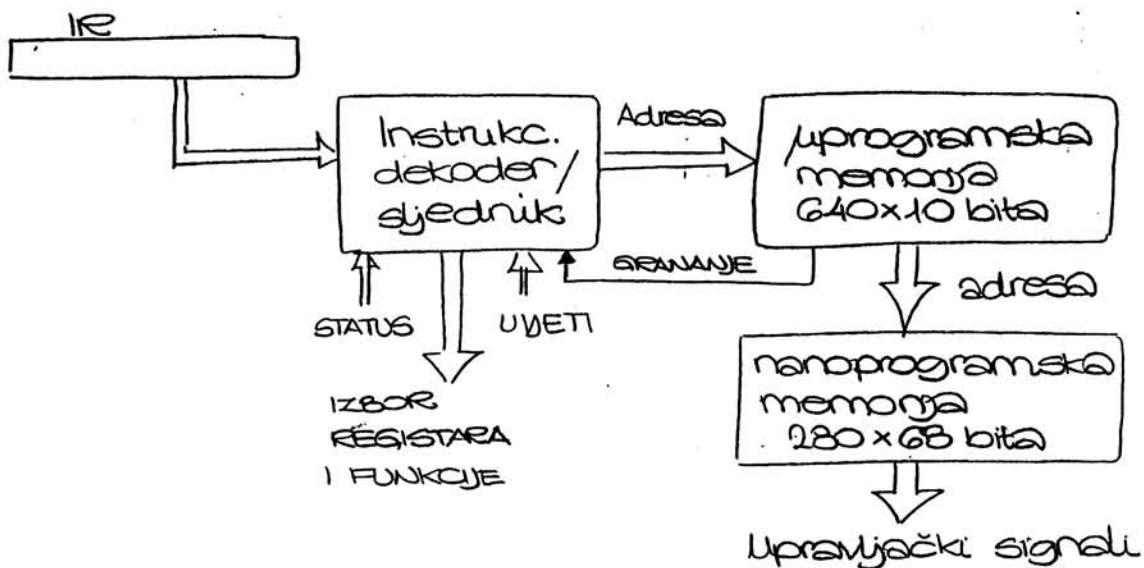
"Programski slijednik": određuje adresu NEXT mikroinstrukcije.

Dvorazinsko programiranje:



⊕ reduciranje površine čipa potrebnog za programiranje

IZVEDBA UPRAVlj.JEDINICE (μprogramirane) ZA MOTOROLU 68020:



FORMATI UPRAVLJAČKE RJEČI (μinstrukcije)

Određivanje oblika uprav. rječi nije jednostavan zato: Uprav. rječ mora izvoditi sljedeće funkcije:

- 1) Grupirati i dodjeljivati upravljačke bitove iz upravljačke rječi
- 2) Sekvenuirati - vezati u niz uprav. rječi (MIKROPROGRAM)
- 3) Sadržati nešto fleksibilnosti za re-programiranje

Te funkc. bi trebale biti izvršene sa:

- a) Minimalnim brojem bitova u uprav. rječi
- b) Min. br. rječi u uprav. memoriji
- c) -" vremenom izvođenja μprograma.

Tehnike za doznačavanje i utvrđivanje uprav. bitova:

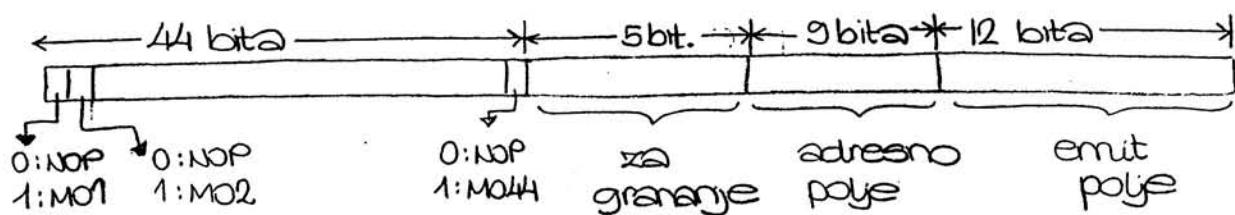
- a) Izravno upravljanje
- b) Grupiranje bitova
- c) Višestruki formati
- d) Vertikalno / horizontalno μprogramiranje

⇒ Veri Long Inner W [za multimedijiske μproc] se bazira na @

I. U.

(engl. direct control)

- unijec je organiziran tako da svakoj moperaciji odgovara 1 bit u unjcu



MO → moperacija

→ u računalu može biti >100 različitih moperacija → duće unjci

GR.

(engl. minimal encoding ; optimal m.e.)

- Grupiranje moperac. koje se UNJEK izvode istodobno → → 1 upravlј. bit upravlja s tom grupom moperacija
- Grupiranje onih moperac. koje se izvode isključivo 1 u vremenu i to tako da se moperacije kodiraju poljem bitova određene duljine
→ manja fleksibilnost reprogram.
→ isključeno istodobno izvođenje moperac.

Npr: ALU → podržava 16 operacija

umjesto 16 moper (koje se isključuju) ⇒ 4 bita $2^4 = 16$ //

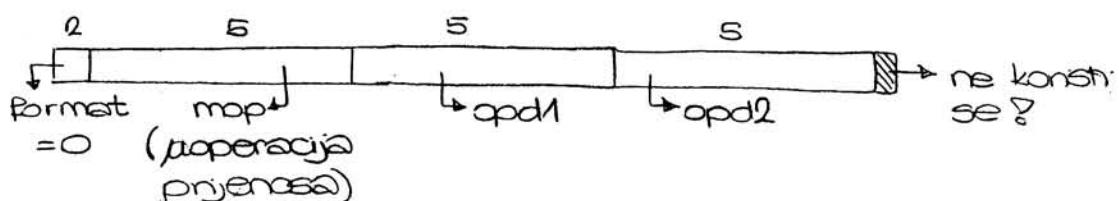
VIES

(engl.

Mikronjec je kratka $16 \div 32$ bita

→ svaka MI u μprogramu specificira 1 ili 2 (max. 4) moperacije

Npr:



if format = 0

then $mop \in \{ \text{TRANSFER} \}$

and interpret (opd1 as source,
opd2 as result)

Specifikacija
samo + MO ?

Specifikacija do 3 μop. (koje se mogu istodobno izvoditi)

if format=1

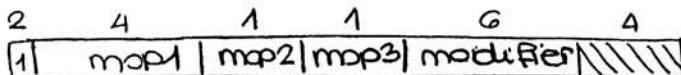
then $mop1 \in \{ADD, SUB, OR, AND, EX, XOR, NOT, SHR, SHL\}$

$mop2 \in \{READ_M, WRITE_M\}$

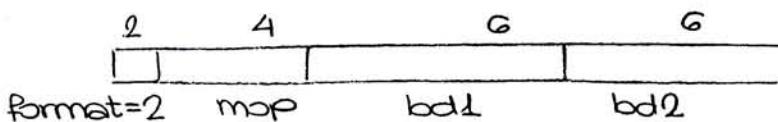
$mop3 \in \{INC_PC\}$

& if $mop1 \in \{SHL, SHR\}$

then interpret (modifier as shift amount)



Spec. za μop. maskiranja:



if format=2

then $mop \in \{MASK\}$

& interpret (bd1 as bound i_1 ,
bd2 as bound i_2)

$\Rightarrow AIR$

Left Input Buffer in ALU $\Rightarrow AIL$

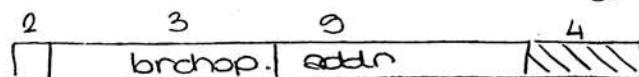
AOUT := AIL mask AIR [$i_1 \dots i_2$]

Output Buffer in ALU $\Rightarrow AOUT$

ad i_1 ujedno do i_2 (udeo)

bez izmjene,

a $[i_1, i_2]$ postaje sadržaj AIR



if format=3

then $brchop \in \{BN, BPZ, BU, BISI, BNISI\}$

& interpret (addr as branch address)

H.M.P

\rightarrow horizontalna MI:

1) MI omogućava različite resurse (npr. funkcionalne jedinice, puteve podataka) u računalu tako da se oni upravljaju nezavisno.

Tj. hor. MI određuje nekoliko paralelnih izvodljivih μop.

2) MI je dug je [64 do >100] bita

3) Hor. MI dopušta programeru upravljanje na razini pojedinačne MI

\rightarrow verticalno MI:

1) MI opisano specifikaciju 1 ili 2 μoperac, ne dopušta iskorištavanje potencijalnog parallelizma u μprocessoru.

- 2.) μI [16 ÷ 32] bita
- 3.) "Vertikalno" μ program ne dopušta upravljanje na razini pojedinačne μ oper.
- ! vert. μI prouzrokuje (pobudiye) skup (skup) μ operacija

PUT PODATAKA (eng. DATA PATH)

- registri (opće namjene, PC, posebni uprav. reg.)
- ALU (arit. i log. sklopovi, posmračni sklop)
- sabirnica procesora (internal bus)

Vrijeme potrebno za dohvat operanada iz reg., za operaciju nad operandima u ALU, te ups rezultata natrag u reg. \Rightarrow VREM. CIKLUS PUTA PODATAKA

↳ što je moguće kradi

Zlatno pravilo arh. RISC:

Štavuj se da bi vremenski ciklus puta podataka bio što kradi?

MIPS procesor:

Data Path : 39.2% ukupne površ. čipa!

Skup od 16 32 bitnih reg. nalazi se na putu podataka

- Tijekom 1 perioda signala:
 - čitanje 2 registra (Φ_1)
 - UPS u 2 - " - (u fazi Φ_2) } DataPath
} kradi ciklus
- ALU - za računanje operandima (i za računanje $\langle ea \rangle$)
 - i ma zbrajalo s CARRY LOOK AHEAD sklopom
- Dodatni registri (Hi i Lo) i bačvasti sklop
 - dijelyenje i množenje ostvareno je Boothovim algoritmom
- PC, PC-3, PC-2, PC-1 su komponente
- Jedinica za maskiranje \rightarrow pretvorba stringne adrese u logičku
 - (- utvrđuje i neovlašteni, pristup memoriji)

glavni problem
kad priučne
i virtualne
memorije

ARITMETIČKO-LOGIČKA JEDINICA MIKROPROCESORA

ALU → nesfunkcijski sklop

izvodi osnovne aritm.-logičke operacije
(upotrebljava se i računaje ef. adrese operanada)

Kombinacijski skloovi ⇒ nemaju mem. elemenata, r.
(brži od sekvenčnih) izlaz ovisi samo o trenutnim
vrijednostima ulaza

ALU → ZBRAJALO + SKLOP ZA POSMACK (bin Neumannovo rač.)

posmak za prizvoljen broj delija u \leq taktu signala → realiziramo samo kombinac. sklopovima (NE s posmacknim registrima)

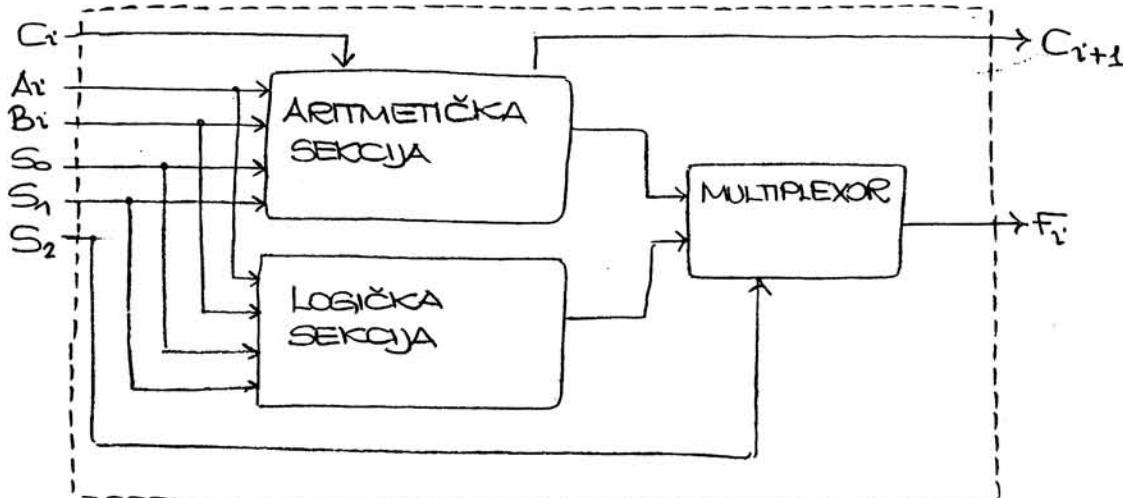
ODUZIMANJE: zbrajanje negativnih brojeva (u notaciji 2^k)

MNOŽENJE: ponavljajući sljed zbrajanja i posmaka

DIJELJENJE: - - - - - oduzimanja - - -

→ knjiga: "Naprednije arh. računala"

ALU: i-ti stupanj

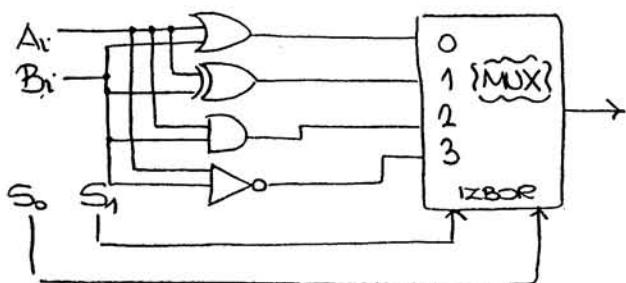


Aritm. i log. sekcije u realizaciji su neodvojive! (jer svaka se aritm. operacija svodi na logičku)

Postupak oblikovanja ALU:

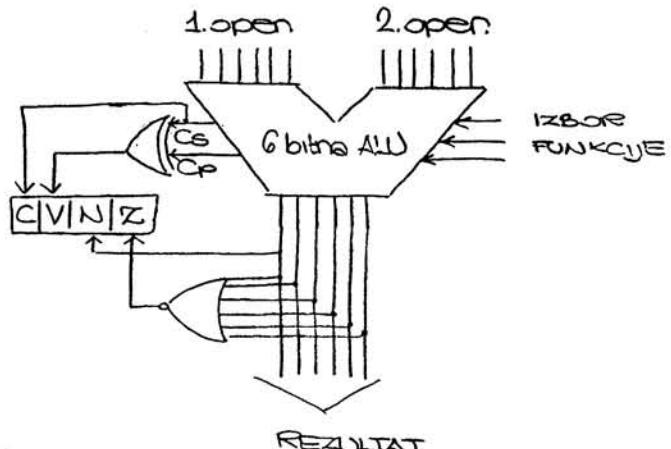
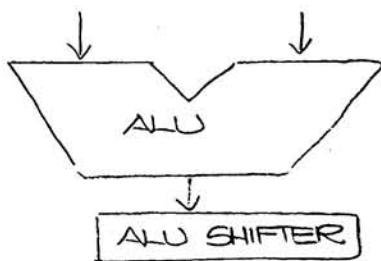
- 1) Oblikuje se aritm. sekcija nezavisno od log.
- 2) Određuju se log. oper. koje se mogu izvesti sa sklopovima iz aritm. sekcije
- 3) Modificiraju se aritm. sklopovi da bi se dozvile željene logičke operacije

LOG. SEKCIJA



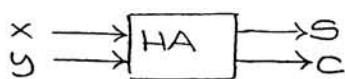
(ti stupanj log. sekcijs na opstrukcionej razini)

ALU:



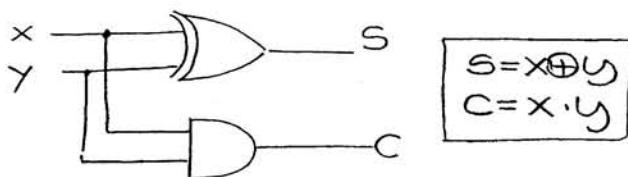
ZBRAJALA:

POLUZBRAJALO (half-adder)



x	y	C	S
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0

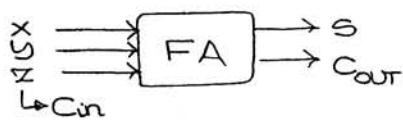
$S = x'y + xy$
 $C = xy$



$$S = x \oplus y$$

$$C = x \cdot y$$

POTFUNKO ZBRAJALO (full-adder)



$$\rightarrow C = xy + xz + yz$$

$$S = \underbrace{z \oplus (x \oplus y)}_{A \quad B} = A \oplus B$$

$$\rightarrow S = z'x'y + z'xy' + z'y'x + xy'z$$

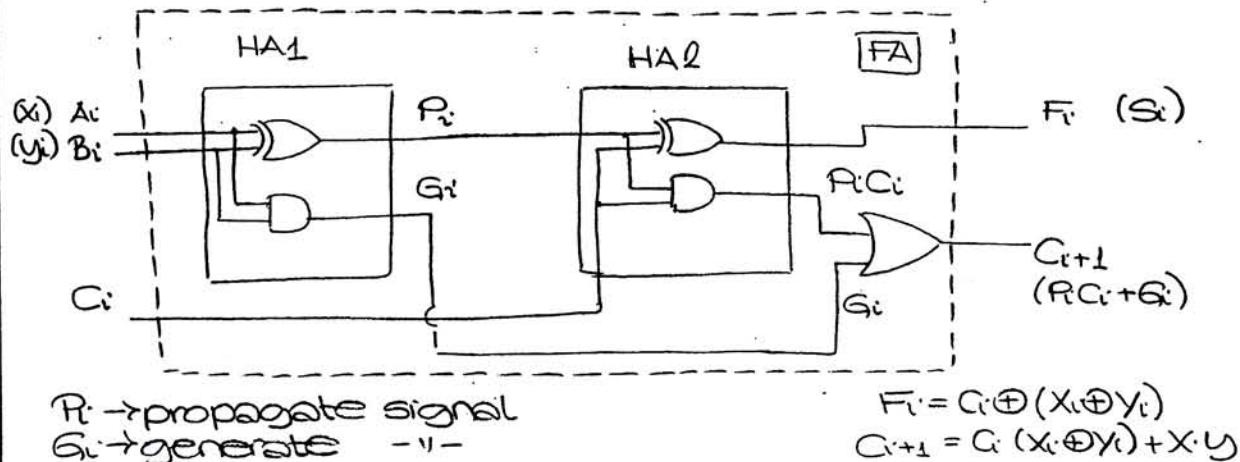
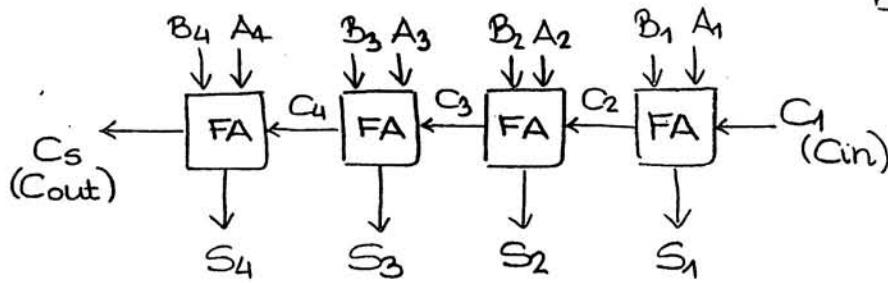
x	y	z	C	S
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
1	0	0	0	1
0	1	1	1	0
1	0	1	1	0
1	1	0	1	1
1	1	1	1	1

$$C = \Sigma(x \oplus y) + xy$$

$$S = \Sigma(x \oplus y)$$

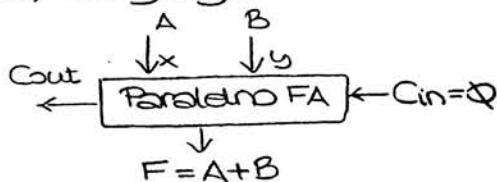
Paral. bin. zbrojalo
(4-bitni operandi)

A_1 - LSB (A)
 B_1 - LSB (B)

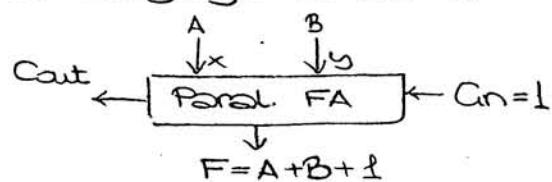


OBNIKOVANJE ARITM. SKLOPOVA

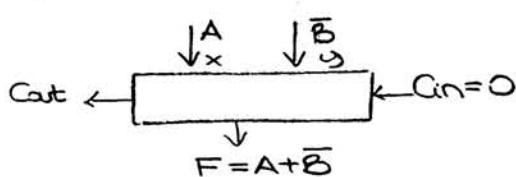
a) zbrojalo



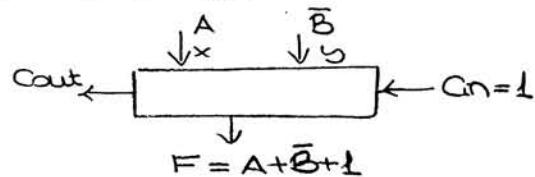
b) zbrojalo s $Cin=1$



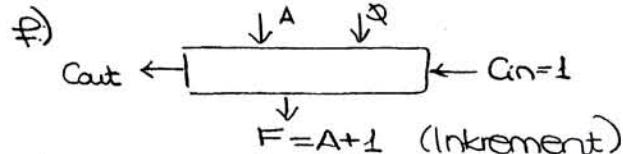
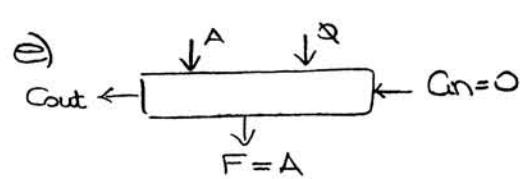
c)



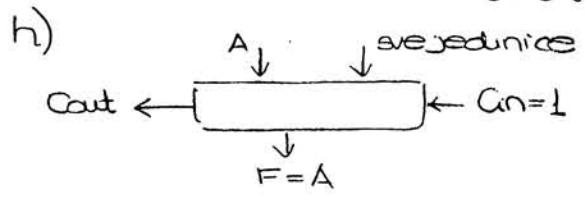
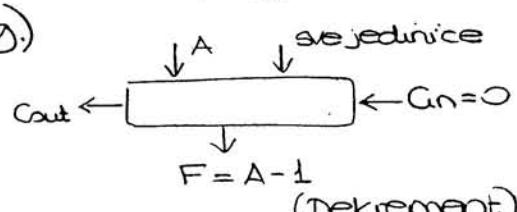
d) oduzimanje



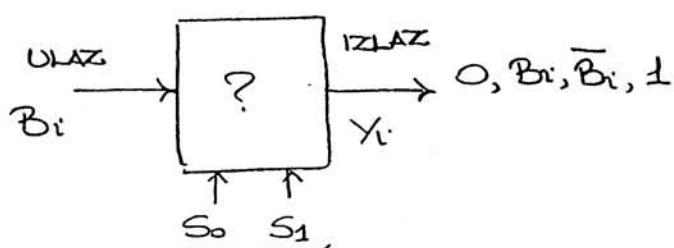
e)



f)



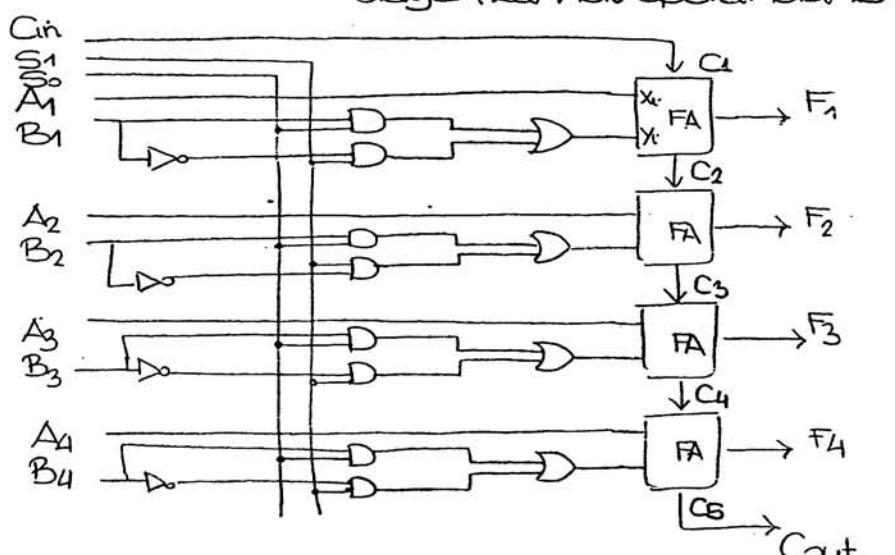
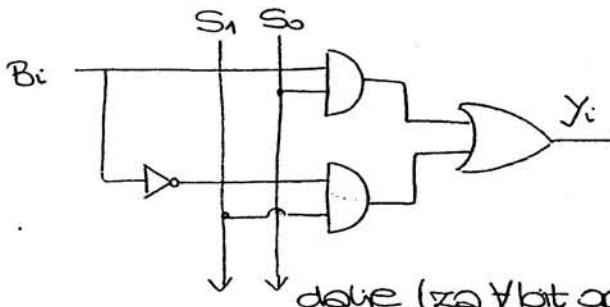
Sklop za pribavljanje operanda B?
 (time zbrajajalom ostvarjujemo $A+B$, $A-B$, $A+1$, $A-1$)



S_1	S_0	B_i	Y_i
0	0	X	0
0	1	\bar{B}_i	\bar{B}_i
1	0	\bar{B}_i	\bar{B}_i
1	1	X	1

$X \rightarrow \text{Don't Care}$

upravljački signali



S_1	S_0	C_{in}	Y	$\text{razz } F$
0	0	0	0	$F = A$
0	0	1	0	$F = A+1$
0	1	0	\bar{B}	$F = A+\bar{B}$
0	1	1	\bar{B}	$F = A+\bar{B}+1$
1	0	0	\bar{B}	$F = A+\bar{B}$
1	0	1	\bar{B}	$F = A+\bar{B}+1$
1	1	0	sve 1	$F = A-1$
1	1	1	sve 1	$F = A$

Prijenos A
 Inkrement A
 Zbrajanje

Oduzimanje
 Dekrement

$$F_i = X_i \oplus Y_i \oplus C_i \quad (\text{za } C_i=Q) \Rightarrow F_i = X_i \oplus Y_i$$

Logička sekocija:

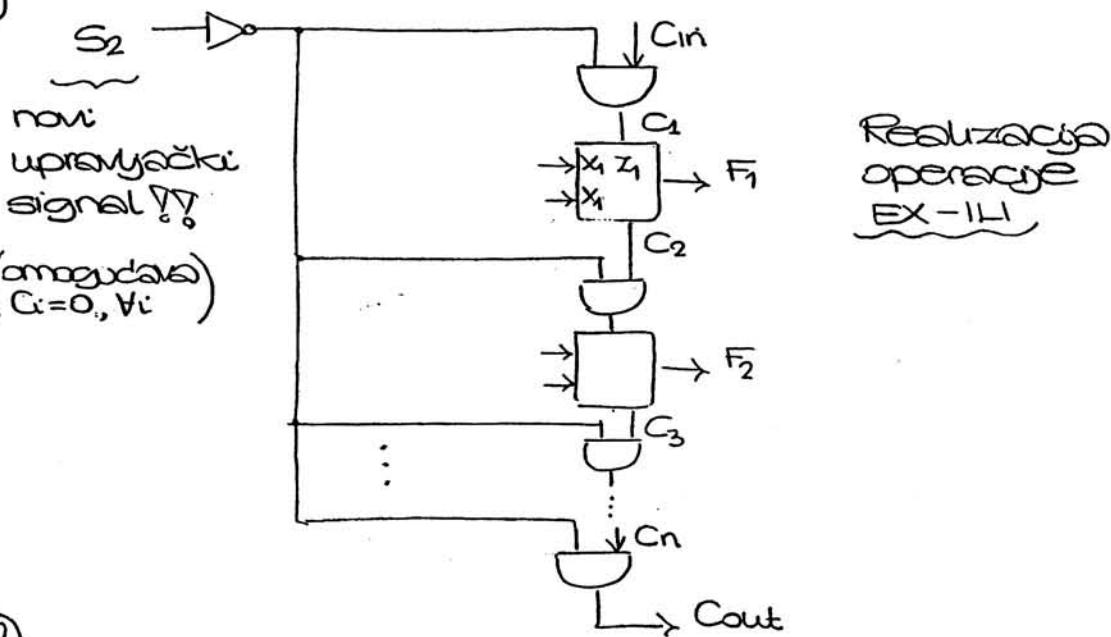
$$FA : F_i = x_i \oplus y_i \oplus c_i$$

$$\begin{array}{c} C \\ \downarrow \\ A \rightarrow \boxed{x_c \frac{z_c}{F_A}} \\ B \rightarrow x \end{array}$$

$$\text{Def. } C_i = Q \Rightarrow F_i = X_i \oplus Y_i$$

LOS. DR. EX-111

1.



2

$S_2 \quad S_1 \quad S_0$ \Rightarrow KOMPLEMENT $[S_1=S_0=1 \text{ (Bsv \& 1)}; S_2=1 \text{ (Cn = \emptyset)}]$

$$F_i = x_i \oplus y_i \Rightarrow F_i = x_i \oplus 1 = x_i^1 = A_i \quad \text{LOGIČKO NE}$$

3

$$\begin{array}{ccccc} S_2 & S_1 & S_0 & \rightarrow \text{EKVIVALENCJA} & \\ 1 & \underbrace{1 \quad 0}_{Y_i = \bar{B}_i} & & & (x \oplus y = \bar{x}y + x\bar{y}) \\ F_i = x_i \oplus y_i = A_i \oplus \bar{B}_i = A_i B_i + \bar{A}_i \bar{B}_i = A_i \odot \bar{B}_i & & & & \end{array}$$

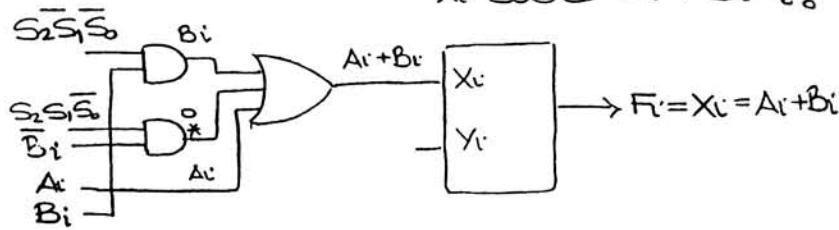
(4) KAKO REALIZIRATI LOG. FUNK. ILI?
(5) -"-. . -"-. -"-. |?

⑤ -1- . -1- -1- | ?

$$A. \quad \begin{matrix} S_2 \\ 1 \end{matrix} \quad \underbrace{\begin{matrix} S_1 \\ 0 \end{matrix}}_{\text{ }} \quad \underbrace{\begin{matrix} S_0 \\ 0 \end{matrix}}_{\text{ }}$$

$F_i = A_i = \text{proyectos}$

Ako je: $F_i = A_i$ \Rightarrow promjenimo uaz X_i svakog FA da X_i bude $A_i + B_i$??



$$* \begin{array}{r} S_2 \\ 1 \underbrace{10} \\ y_i = \overline{B_2} \end{array}$$

5. I:

$$\begin{array}{c} S_2 \\ S_1 \\ 1 \end{array} \quad \begin{array}{c} S_0 \\ 1 \\ 0 \end{array}$$

$$F_i = A_i \odot B_i = A_i' B_i + \bar{A}_i \bar{B}_i$$

šezdeset samo ovaj dio rezultata //

Uvedimo varijablu K_i kojom izvedemo logičko ILI sa svakim ulazom A_i !! Tj. $X_i = A_i + K_i$

$$\begin{array}{c} S_2 \\ S_1 \\ S_0 \\ Y_i = B_i \end{array}$$

$$F_i = X_i \oplus Y_i = (A_i + K_i) \oplus \bar{B}_i = A_i B_i + K_i B_i + \bar{A}_i \bar{K}_i \bar{B}_i = A_i B_i //$$

K_i mora biti $= \bar{B}_i$ tako da neželjene članove "uaye" //

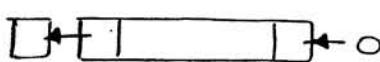
IZVEDBA i-tog STUPNJA ALU JEDINICE // (kopija)

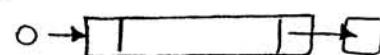
SKLOPOVI ZA POSMAK

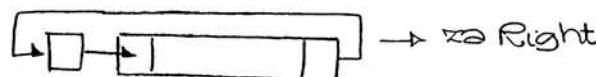
- dvozmjerni posmačni registri s paralelnom vezom
- kombinacijski skloovi

H ₀	H ₁	
0	0	bez posmaka
0	1	posmak uljevo
1	0	-"- udesno
1	1	"0"

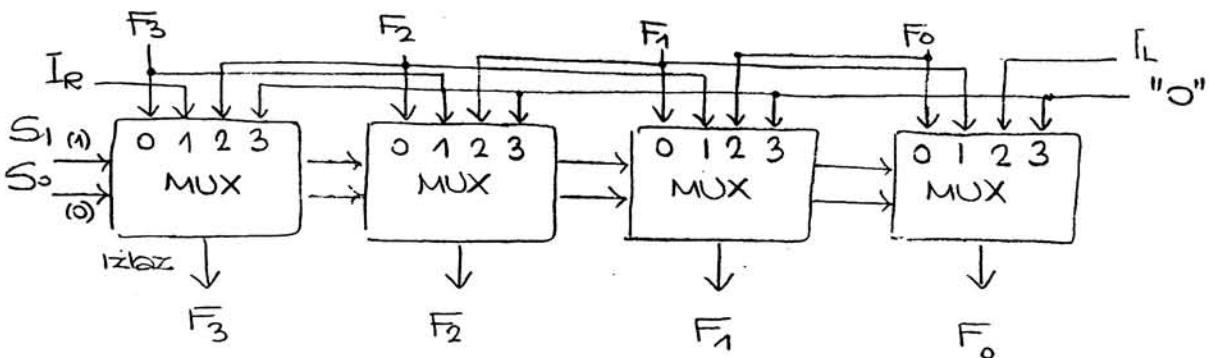
I_R, I_L - senjski ulazi

Arith. Shift Left 

Logic Shift Right 

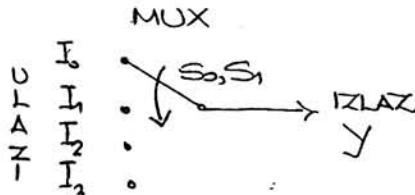
ROtate R, L  → \Rightarrow Right

Kombinac. posmačni sklop:



S ₀	S ₁	
0	0	bez posm.
0	1	posm. L
1	0	posm. R
1	1	"0"

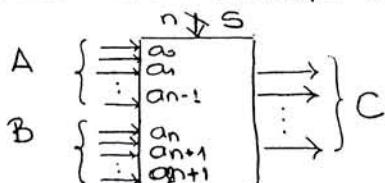
I_R, I_L - senjski ulazi



Definiranje "0" ⇒ VAŽNO! (Npr. RISC μP obično Ro ili R3L ima kao generator "Q" (poniranje podataka))

Matematički koprocesori; μP naprednije arhitekture (32 bitni) imaju SHIFTERE (brže) za proizv. br. bitova u 1 taktu:

Baćvasti sklop za posmak (BARREL SHIFTER):



S → za koliko mjesto pomaknuti operand na ulazu?

$$\overbrace{a_{n-1} a_{n-2} \dots a_0}^A \quad \overbrace{a_{2n-1} \dots a_{n+1} a_n}^B$$

$$s=0 \Rightarrow C=A$$

$$s=i \Rightarrow C_{n-1} \dots C_2 C_1 C_0 = a_{n-1+i} + \dots + a_{n+i} a_0 + \dots + a_i$$

$$s=n \Rightarrow C_{n-1} \dots C_2 C_1 C_0 = a_{2n-1} \dots a_n = B$$

??

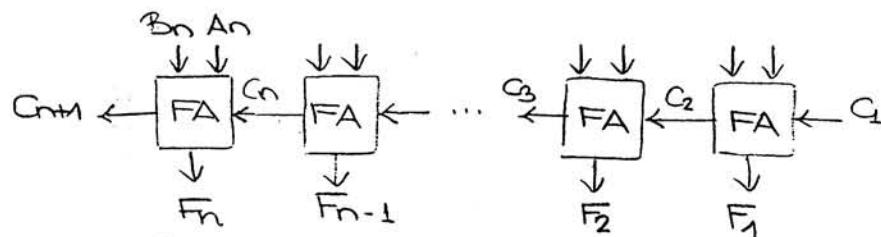
$\forall A = B \Rightarrow$ kružni posmaci

$\forall B=0 \Rightarrow$ posmaci udesno } A, B mogu biti i 1 (su bitovi)
 $\forall A=0 \Rightarrow$ posmaci uljevo } (proizvodno)

Sklopovi za predviđanje bita prijenosa

HA

HA



Kašnjenje : širenje bita prijenosa

F_n stabilan tek kad je C_n stabilan (C_n ovisi o sumi prethodnog stupnjevima)

Kašnjenje C_i signala: $(2 \times n) t_g + t_h$

tg - kašnjenje na pojedinim vratima
 t_h - inicijalno kašnjenje

• Signal generiranja bita prijenosa : $g_i = A_i B_i$

• Signal propagacije bita : $p_i = A_i \oplus B_i$ (A_i modulo 2 B_i)

Izlazni signal bita prijenosa : $C_{i+1} = g_i + p_i C_i$

Suma : $F_i = p_i \oplus C_i$

Analogno za : $C_{i+2} = g_{i+1} + p_{i+1}(C_{i+1}) = g_{i+1} + p_{i+1}(g_i + p_i C_i)$

Odnosno :

$$C_{i+3} = g_{i+2} + p_{i+2}(C_{i+2}) = g_{i+2} + p_{i+2}[g_{i+1} + p_{i+1}(g_i + p_i C_i)]$$

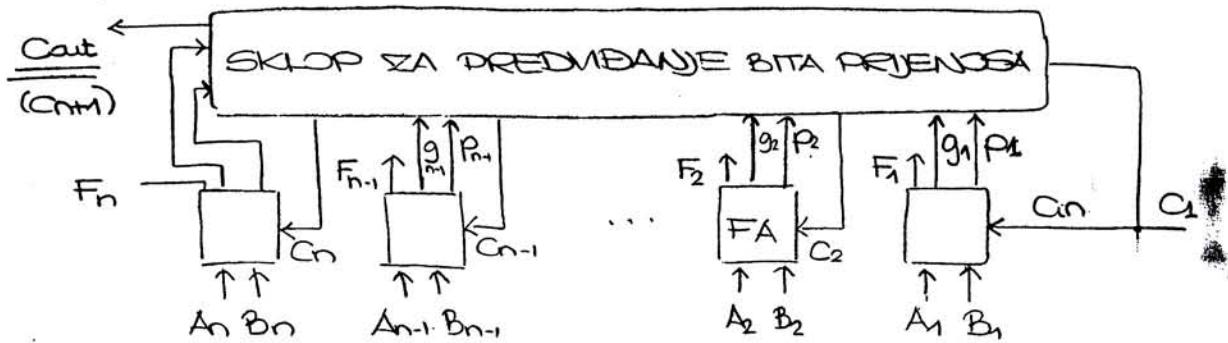
$$C_{i+3} = g_{i+2} + p_{i+2} \cdot g_{i+1} + p_{i+2} p_{i+1} g_i + p_{i+2} p_{i+1} p_i C_i$$

Nastavljamo tako do izraza za $C_{out} = C_{n+1}$ kao sumu produkata p : g ??

Npr. za 4bitno zbrojalo:

$$C_5 = g_4 + p_4 c_4 = g_4 \\ \dots = g_4 + p_4 g_3 + p_4 p_3 g_2 + p_4 p_3 p_2 g_1 + p_4 p_3 p_2 p_1 c_1$$

Sada C_n ovisi o p, g (odnosno o BITOVIMA OPERANADA A_i B_i), i ne mora ovisiti o C prethodnog stupnja & (ne mora na njega dezati?)



1 razina: log.vrata I
2 razina: -||- -||- ||I | } samo 2tg } DVORAZINSKI LOGIČKI SKLOP

