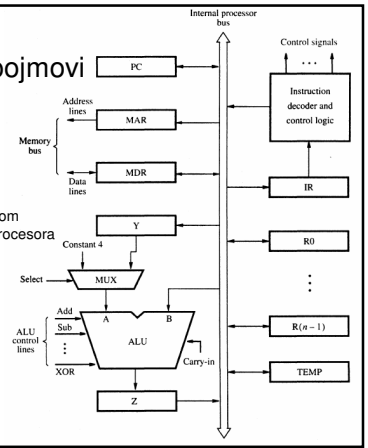
# Centralna procesorska jedinica

Vladimir Filipović
vladaf@matf.bg.ac.yu



## Osnovni pojmovi

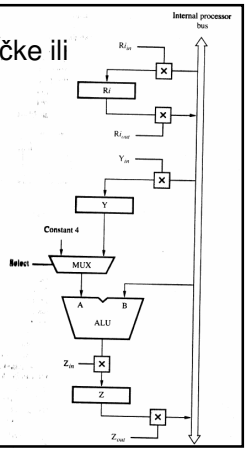
Organizacija sa jedinstvenom magistralom podataka unutar procesora





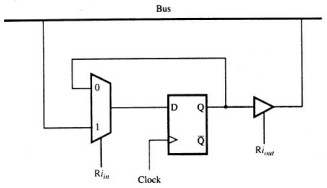

## Izvršavanje aritmetičke ili logičke operacije

Ulazne i izlazne kapije za registre sa prethodnog dijagrama






## Izvršavanje aritmetičke ili logičke operacije



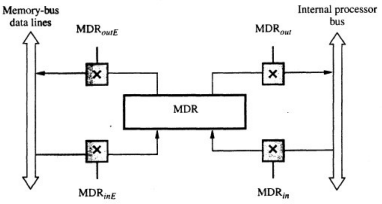

Ulazne i izlazne kapije za jedan registarski bit

1. $R1_{out}$, $Y_{in}$
2. $R2_{out}$, SelectY, Add, $Z_{in}$
3. $Z_{out}$, $R3_{in}$

Sekvenca operacija za dodavanje sadržaja R1 na sadržaj R2 i smeštanje u R3



## Dohvatanje reči iz memorije




Veze i kontrolni signali za registar MDR

1. MAR ← [R1]
2. Start a Read operation on the memory bus
3. Wait for the MFC response from the memory
4. Load MDR from the memory bus
5. R2 ← [MDR]

Primer operacije čitanja, konkretno Move (R1), R2



## Dohvatanje reči iz memorije

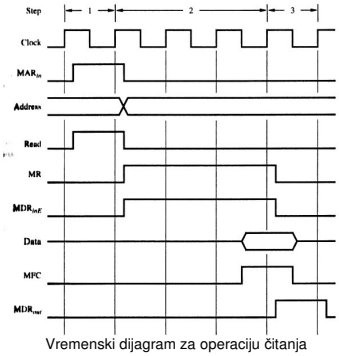
Svaka od akcija sa prethodnog sleda može da bude završena u jednom ciklusu, osim akcije u koraku 3, koja u zavisnosti od brzine uređaja koji je adresiran, može zahtevati jedan ili više ciklusa. Operacija čitanja memorije zahteva tri koraka, koji se mogu opisati kao signali aktivirani na sledeći način:

1. $R1_{out}$, $MAR_{in}$, Read
2. $MDR_{inE}$, WMFC
3. $MDR_{out}$, $R2_{in}$

Pri tome, WMFC je kontrolni signal koji uslovljava da kontrolna kola procesora čekaju na dolazak signala MFC.

Dijagram na sledećem slajdu pokazuje da je $MDR_{inE}$ postavljeno na 1 tačno u onom periodu kada je to slučaj sa signalom komande čitanja MR. Zato se u narednim razmatranjima neće eksplicitno specificirati vrednost MDRinE već će se pretpostavljati da je ta vrednost uvek jednaka sa MR.

# Dohvatanje reči iz memorije




Vremenski dijagram za operaciju čitanja



# Smeštanje reči u memoriju

**R1$_{out}$, MAR$_{in}$**
**R2$_{out}$, MDR$_{in}$, Write**
**MDR$_{outE}$, WMFC**

Primer operacije upisa, konkretno Move R2, (R1)

Slično kao u slučaju operacije čitanja, kontrolni signal za upis prouzrokuje da hardver za interfejs memorijske magistrale izvrši komandu za upis u magistralu. Procesor ostaje u koraku 3 sve dok se ne okonča memorijska operacija i dok se ne dobije odgovor MFC.



# Izvršavanje kompletne instrukcije

Dijagram na ovom slajdu prikazuje sekvencu kontrolnih koraka potrebnih za izvršenje instrukcije

Add (R3), R1

na arhitekturi procesora sa jedinstvenom magistralom podataka.

| Step | Action |
|---|---|
| 1 | PC$_{out}$, MAR$_{in}$, Read, Select4, Add, Z$_{in}$ |
| 2 | Z$_{out}$, PC$_{in}$, Y$_{in}$, WMFC |
| 3 | MDR$_{out}$, IR$_{in}$ |
| 4 | R3$_{out}$, MAR$_{in}$, Read |
| 5 | R1$_{out}$, Y$_{in}$, WMFC |
| 6 | MDR$_{out}$, SelectY, Add, Z$_{in}$ |
| 7 | Z$_{out}$, R1$_{in}$, End |

Kontrolna sekvenca za izvršavanje instrukcije Add (R3), R1



# Instrukcija grananja (skoka)

Instrukcija skoka menja sadržaj PC sa odredištem skoka. Adresa odredišta se obično dobija tako što se ofset X, koji je specificiran instrukcijom skoka, dodaje na ažuriranu vrednost PC.

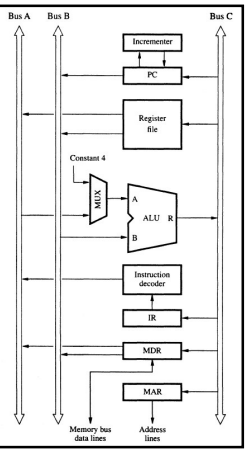
1. **PC$_{out}$, MAR$_{in}$, Read, Select4, Add, Z$_{in}$**
2. **Zout, PCin, Yin, WMFC**
3. **MDR$_{out}$, IR$_{in}$**
4. **Offset-field-of-IR$_{out}$, Add, Z$_{in}$**
5. **Z$_{out}$, PC$_{in}$, End**

Kontrolna sekvenca za izvršavanje instrukcije bezuslovnog skoka



# Organizacija sa više magistrala

Organizacija puta podataka sa tri magistrale






# Organizacija sa više magistrala

Da bi se smanjio broj koraka potrebnih za izvršenje instrukcije, najveći broj komercijalnih procesora koristi višestruke unutrašnje puteve podataka, koji omogućuju da se paralelno izvršava više transfera podataka.

Svi registri opšteg tipa iskombinovani su u jedinstven blok, koji se naziva *registarski fajl.* U VLSI tehnologiji, najefikasniji način za implementaciju većeg broja registara je njihova organizacija na način sličan implementaciji RAM memorije.

U ovoj organizaciji nema potrebe za registrima Y i Z, a uvođenje inkrementera je eliminisalo potrebu da se dodavanje 4 na PC vrši pomoću ALU. Ipak, mogućnost da konstanta 4 bude izvor za ALU multiplekser i je dalje jako korisna. Naime, on može biti korišćen za uvećanje adresa, npr. prilikom izvršenja instrukcija LoadMultiple, odnosno StoreMultiple.

# Organizacija sa više magistrala

Kontrolna sekvenca za izvršavanje instrukcije sabiranja ima sledeći oblik: u koraku 1, sadržaj registra PC se prosleđuje prema ALU, korišćenjem R=B kontrolnog signala, i učitava se u MAR kako bi se počelo sa operacijom čitanja memorije. Simultano se PC uvećava za 4. Naravno, MAR sadrži originalnu vrednost PC, zato što se uvećana vrednost smestila u PC na kraju ciklusa časovnika; u koraku 2, procesor čeka MFC pa primljene podatke učitava u MDR; u koraku 3 ti podaci bivaju preneseni u IR; na kraju, faza izvršavanja instrukcije zahteva samo jedan korak da bi se uspešno završila.

| Step | Action |
|---|---|
| 1 | $PC_{out}$, R=B, $MAR_{in}$, Read, IncPC |
| 2 | WMFC |
| 3 | $MDR_{outB}$, R=B, $IR_{in}$ |
| 4 | $R4_{outA}$, $R5_{outB}$, SelectA, Add, $R6_{in}$, End |

Kontrolna sekvenca za izvršavanje instrukcije Add R4,R5,R6 na organizaciji opisanoj prethodnim dijagramom



# Ožičena kontrola

Da bi instrukcija bila izvršena, procesor mora na neki način obezbediti da se kontrolni signali generišu u odgovarajućem redosledu. Svi ti raznovrsni pristupi kojima se obezbeđuje zahtevana funkcionalnost spadaju u jednu od dve kategorije: ***ožičena kontrola*** ili ***mikroprogramska kontrola***.

Razmotrimo sekvencu kontrolnih signala za izvršenje instrukcije ‚Add (R3), R1

Svaki korak ove instrukcije se završi tokom jednog perioda časovnika. Kao što dijagram sa sledećeg slajda prikazuje, brojačko kolo se može koristiti da vodi računa o kom se kontrolnom koraku radi - pa svako od stanja brojača odgovara jednom kontrolnom koraku. Kontrolni signali su određeni sledećim informacijama:

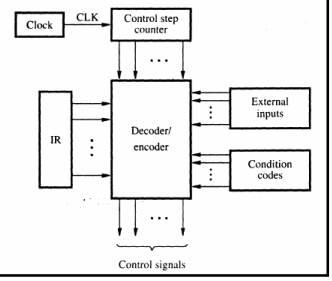
- Sadržaj brojača kontrolnih koraka
- Sadržaj registra instrukcija
- Sadržaj registra uslovnih flegova
- Ulazni signali iz spoljašnosti, kao što su MFC i signali za prekid



# Ožičena kontrola

Kao i obično, razmatranje počinjemo sa uprošćenom verzijom. Dekodersko-enkoderski blok je kombinatorno kolo koje generiše zahtevane kontrolne izlaze, zavisno od stanja svih ulaza.

Organizacija kontrolne jedinice

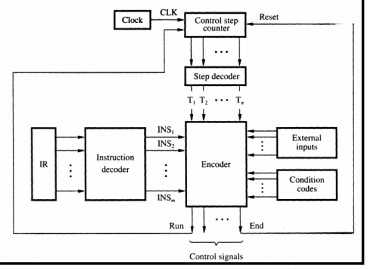





# Ožičena kontrola

Razdvajanjem dekoderskih funkcija od enkoderskih, dobija se detaljniji dijagram. Dekoder koraka obezbeđuje odvojenu signalnu liniju za svaki korak, ili vremenski slot u kontrolnoj sekvenci. Na sličan način, izlaz iz dekodera instrukcija se sastoji od odvojenih linija za svaku mašinsku instrukciju.

Razdvajanje funkcije dekodiranja i funkcije enkodiranja






# Ožičena kontrola

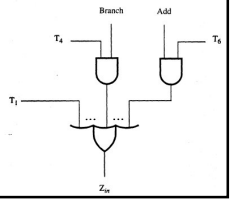
Dakle, za svaku instrukciju koja se učita u IR, tačno jedan od ulaznih signala $INS_1$, $INS_2$, ..., $INS_m$ bude postavljen na 1, a sve ostale linije se postavljaju na 0.

Ulazni signali u enkoderski blok se kombinuju kako bi se generisali pojedinačni kontrolni signali $Z_{in}$, $PC_{out}$, Add, End itd.

Na dijagramu u dnu slajda dat je primer kako enkoder generiše kontrolni signal $Z_{in}$ kada je procesorska organizacija sa jednostrukom magistralom. Kolo sa dijagrama implementira logičku funkciju

$$Z_{in} = T_1 + T_6 * ADD + T_4 * BR + \ldots.$$

Generisanje $Z_{in}$ signala za procesor sa jednostrukom magistralom

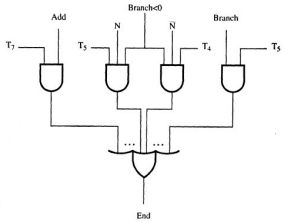





# Ožičena kontrola

Još jedan primer dijagrama koji opisuje kako se generiše kontrolni signal End kod procesora sa jednostrukom internom magistralom. U ovom slučaju, radi se o logičkoj funkciji:

$$End = T_7 * ADD + T_5 * BR + (T_5 * N + T_4 * \overline{N}) * BRN + \ldots$$




Generisanje kontrolnog signala End za procesor opisan dijagramom na slajdu 2

# Ožičena kontrola

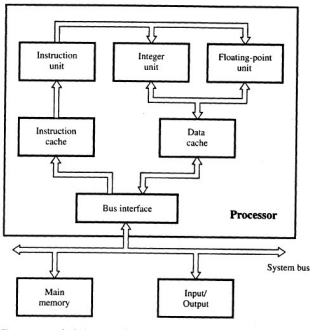
Signal End otpočinje novi instrukcijski ciklus, tako što brojač kontrolnih koraka resetuje na njegovu početnu vrednost. Dijagram koji opisuje kontrolnu jedinicu sadrži i signal označen sa RUN. Kada je taj signal postavljen na 1, to dovodi da brojač kontrolnih koraka bude inkrementiran na kraju svakog ciklusa časovnika. Kada je RUN signal jednak 0, brojač prestaje sa odbrojavanjem. Ovakvo ponašanje je potrebno kad god se aktivira WMFC signal, i na taj način se obezbeđuje da procesor čeka na odgovor iz memorije.

Kontrolni hardver sa prethodnih dijagrama može da se posmatra kao konačni automat, gde se prelaz iz jednog stanja u drugo vrši pri svakom ciklusu časovnika, zavisno od sadržaja registra instrukcija, uslovnih kodova i ulaza iz spoljašnosti. Izlaz iz tog konačnog automata su kontrolni signali. Redosled operacija koje izvršava takav automat je određen načinom na koji su logički elementi povezani, pa stoga ime "ožičen".

Kontroler koji koristi ovakav pristup može da radi velikom brzinom. Međutim, on ima malu fleksibilnost, a složenost skupa instrukcija koje se tako mogu implementirati je ograničena.



# Ožičena kontrola




Blok dijagram kompletnog procesora



# Mikroprogramska kontrola

Kod mikroprogramske kontrole, kontrolni signali su generisani programom koji je sličan programima na mašinskom jeziku.

Prvo uvodimo osnovne pojmove.

*Kontrolna reč* (*CW*) je reč čiji pojedinačni bitovi predstavljaju različite kontrolne signale. Svaki od kontrolnih koraka u kontrolnoj sekvenci instrukcije predstavlja jedinstvenu kombinaciju jedinica i nula koje se nalaze u kontrolnoj reči.

| Micro - instruction | .. | $PC_{in}$ | $PC_{out}$ | $MAR_{in}$ | Read | $MDR_{out}$ | $IR_{in}$ | $Y_{in}$ | Select | Add | $Z_{in}$ | $Z_{out}$ | $R1_{out}$ | $R1_{in}$ | $R3_{out}$ | WMFC | End | .. |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | | 0 | 1 | 1 | 1 | 0 | 0 | 0 | 1 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | |
| 2 | | 1 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 1 | 0 | |
| 3 | | 0 | 0 | 0 | 0 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | |
| 4 | | 0 | 0 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | |
| 5 | | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 1 | 0 | |
| 6 | | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | |
| 7 | | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 1 | 0 | 0 | 1 | |

Primer mikroinstrukcija za instrukciju Add (R3), R1 – ovde je pretpostavljeno da je SelectY Select=0, a da je Select4 Select=1

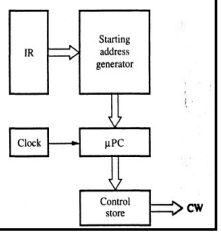


# Mikroprogramska kontrola

Sekvenca kontrolnih reči koje odgovaraju kontrolnoj sekvenci za mašinsku instrukciju obrazuju *mikrorutinu* za tu instrukciju, a pojedinačne kontrolne reči iz mikrorutine se označavaju kao *mikroinstrukcije*.

Mikrorutine za sve instrukcije su smeštene u posebnu memoriju, koja se naziva *kontrolna memorija*. Kontrolna jedinica može generisati kontrolne signale za ma koju instrukciju tako što sekvencijalno čita kontrolne reči odgovarajuće mikrorutine iz kontrolne memorije.

Ovo sugeriše sledeću organizaciju kontrolne jedinice:




Osnovna organizacija mikroprogramirane kontrolne jedinice



# Mikroprogramska kontrola

Da bi se pročitale kontrolne reči sekvencijalno iz kontrolne memorije, koristi se *brojač mikro-naredbi* (*μPC*). Svaki put kada se nova instrukcija učita u IR, izlaz iz bloka nazvanog "generator početne adrese" se prosledi u μPC. Registar μPC se potom automatski inkrementira od strane časovnika, i tako se postiže da se sukcesivne mikroinstrukcije čitaju iz kontrolne memorije. Na taj način se kontrolni signali prosleđuju do različitih delova procesora u korektnom redosledu.

Važna funkcija koju kontrolna jedinica sa prethodnog dijagrama ne može implementirati je vezana za uslovne skokove. U takvim slučajevima, kontrolna jedinica mora da proveri status uslovnog koda ili ulaz iz spoljašnosti, kako bi izabrala između alternativnih pravaca akcije. Stoga je potrebno postojanje mikroinstrukcija uslovnog grananja. Ove mikroinstrukcije, pored adresa koja predstavljaju odredišta skoka, specificiraju i to koji od spoljašnjih ulaza, uslovnih kodova, ili možda bitova registra instrukcija, treba da budu provereni da bi uslov za skok bio ispunjen.

Tako na primer, instrukcija Branch < 0 može da se implementira na način koji je opisan sledećim pseudokodom:



# Mikroprogramska kontrola

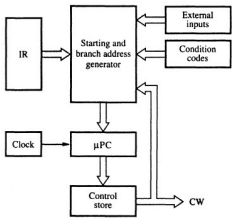
| Address | Microinstruction |
|---|---|
| 0 | $PC_{out}$, $MAR_{in}$, Read, Select4, Add, $Z_{in}$ |
| 1 | $Z_{out}$, $PC_{in}$, $Y_{in}$, WMFC |
| 2 | $MDR_{out}$, $IR_{in}$ |
| 3 | Branch to starting address of appropriate microroutine |
| ... | ... |
| 25 | If N=0, then branch to microinstruction 0 |
| 26 | Offset-field-of-$IR_{out}$, SelectY, Add, $Z_{in}$ |
| 27 | $Z_{out}$, $PC_{in}$, End |

Mikrorutina za instrukciju Branch < 0

Po učitavanju te instrukcije u IR, dolazi do skoka tj. prenosa kontrole na mikrorutinu koja se odnosi na grananje tj. baš na ovu vrstu instrukcije (u našem slučaju to je adresa 25 u kontrolnoj memoriji). Mikroinstrukcija na lokaciji 25 proverava fleg N. Ako je on 0, skače se na lokaciju 0 u kontrolnoj memoriji, kako bi se dohvatila nova mašinska instrukcija. Inače, izvršava se mikroinstrukcija sa lokacije 26, čime se odredište skoka smešta u registar Z, a potom sledeća mikroinstrukcija tu adresu učitava u PC.

# Mikroprogramska kontrola

Organizacija kontrolne jedinice koja dozvoljava uslovno grananje u mikroprogramu




Blok za generisanje početne adrese je sada postao blok za generisanje početne adrese i adrese grananja. Ovaj blok učitava novu adresu u μPC kad god ga mikroinstrukcija instruiše da to uradi. Da bi se omogućila implementacija uslovnih prelazaka, ulazi u ovaj blok se sastoje od spoljašnjih ulaza uslovnih flegova, kao i od sadržaja IR. Ovde se μPC ne inkrementira uvek kada se mikroinstrukcija dohvata iz mikroprogramske memorije.



# Mikroprogramska kontrola

Do inkrementacije μPC ne dolazi u sledećim slučajevima:

- Kada se nova instrukcija učitava u IR – tada se u μPC smešta početna adresa mikrorutine za tu instrukciju
- Kada se radi o mikroinstrukciji grananja, a uslov grananja je zadovoljen – tada se u μPC smešta odredišta grananja
- Kada se radi o mikroinstrukciji End – tada se u μPC učitava adresa prve kontrolne reči u mikrorutini za ciklus dohvatanja (to je u primeru koji smo analizirali bila adresa 0)



# Mikroinstrukcije

Sada možemo pažljivije ispitati format pojedinačnih mikroinstrukcija.

Direktan pristup za strukturisanja mikroinstrukcija je dodela jedinstvene pozicije bita svakom od kontrolnih signala (ranije smo imali primer takvog podešavanja). Međutim sa ovakvom shemom ima problema – mikroinstrukcije su tada veoma duge. Nadalje, kod svake takve mikroinstrukcije je samo mali broj bitova postavljen na 1 – što ukazuje da je slabo korišćen bitovni prostor koji je na raspolaganju.

Na sreću, dužina mikroinstrukcija može lako da se skrati. Na primer, samo jedna funkcija ALU može biti aktivirana u datom trenutku. Dalje, izvor za prenos podataka mora biti jedinstven. Signali Read i Write za čitanje iz i upis u memoriju ne mogu biti simultano aktivni. Sve ovo sugeriše da treba grupisati signale tako da su uzajamno isključujući signali smešteni u istoj grupi. Tako je u svakoj mikroinstrukciji moguće specificirati najviše jednu *mikrooperaciju* iz grupe. Tada je moguće koristiti binarno kodiranje kako bi se reprezentovali signali unutar grupe. Na primer, dovoljna su 4 bita da predstave 16 logičkih operacija ALU, i 4 bita za predstavljanje signala za izlaz: $PC_{out}$, $MDR_{out}$, $Offset_{out}$, $R0_{out}$, $R1_{out}$, $R2_{out}$, $R3_{out}$, $Temp_{out}$.



# Mikroinstrukcije

Preostali signali se mogu dalje prirodno grupisati. Dijagram na sledećem slajdu prikazuje primer delimičnog formata mikroinstrukcija, u kome svaka grupa zauzima polje dovoljno veliko za smeštaj zahtevanih kodova. Grupisanje kontrolnih signala u polja zahteva malo više hardvera, jer se moraju koristiti dekoderska kola za dekodiranje bitovnih obrazaca u svakom polju.

Zasad smo samo razmatrali grupisanje uzajamno isključujućih kontrolnih signala. Ova ideja se može proširiti tako što se enumerišu obrasci zahtevanih signala u svakoj mogućoj mikroinstrukciji. Svakoj smislenoj kombinaciji aktivnih kontrolnih signala može biti dodeljen kod koji je predstavlja. Takvo, puno kodiranje bi verovatno još smanjilo dužinu mikroreči, ali bi uvećalo složenost neophodnih dekoderskih kola.

Sheme sa visokim nivoom enkodiranja se obično označavaju kao *vertikalna organizacija*. Na drugom polu, sheme sa minimalnim enkodiranjem se nazivaju *horizontalna organizacija*.



# Mikroinstrukcije

Delimični primer formata za mikroinstrukcije sa kodiranim poljima

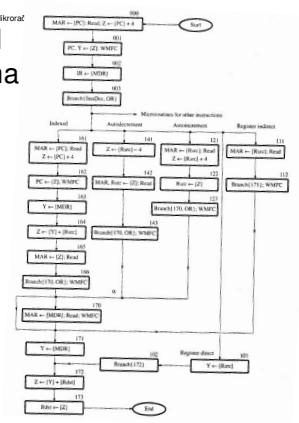
| F1 | F2 | F3 | F4 | F5 |
|---|---|---|---|---|
| F1 (4 bits) | F2 (3 bits) | F3 (3 bits) | F4 (4 bits) | F5 (2 bits) |
| 0000: No transfer<br>0001: $PC_{out}$<br>0010: $MDR_{out}$<br>0011: $Z_{out}$<br>0100: $R0_{out}$<br>0101: $R1_{out}$<br>0110: $R2_{out}$<br>0111: $R3_{out}$<br>1010: $TEMP_{out}$<br>1011: $Offset_{out}$ | 000: No transfer<br>001: $PC_{in}$<br>010: $IR_{in}$<br>011: $Z_{in}$<br>100: $R0_{in}$<br>101: $R1_{in}$<br>110: $R2_{in}$<br>111: $R3_{in}$ | 000: No transfer<br>001: $MAR_{in}$<br>010: $MDR_{in}$<br>011: $TEMP_{in}$<br>100: $Y_{in}$ | 0000: Add<br>0001: Sub<br>⋮<br>1111: XOR<br>16 ALU functions | 00: No action<br>01: Read<br>10: Write |

| F6 | F7 | F8 | ⋯ |
|---|---|---|---|
| F6 (1 bit) | F7 (1 bit) | F8 (1 bit) | |
| 0: SelectY<br>1: Select4 | 0: No action<br>1: WMFC | 0: Continue<br>1: End | |

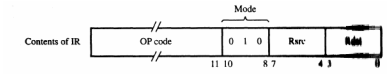


# Redosled kod mikroprograma

Dijagram toka za instrukciju
Add src, Rdst



# Redosled kod mikroprograma



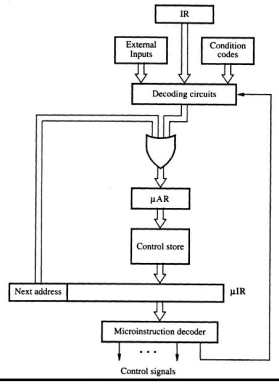

Mikroinstrukcija
Add [Rsrc]+, Rdst

| Address (octal) | Microinstruction |
|---|---|
| 000 | $PC_{out}$, $MAR_{in}$, Read, Select4, Add, $Z_{in}$ |
| 001 | $Z_{out}$, $PC_{in}$, $Y_{in}$, WMFC |
| 002 | $MDR_{out}$, $IR_{in}$ |
| 003 | μBranch {μPC ← 101 (from Instruction decoder)}; $μPC_{5,4} ← [IR_{10,9}]$; $μPC_3 ← [\overline{IR_{10}} · [IR_8] · [IR_8]]$ |
| 121 | $Rsrc_{out}$, $MAR_{in}$, Read, Select4, Add, $Z_{in}$ |
| 122 | $Z_{out}$, $Rsrc_{in}$ |
| 123 | μBranch {μPC ← 170; $μPC_0 ← [IR_8]$}, WMFC |
| 170 | $MDR_{out}$, $MAR_{in}$, Read, WMFC |
| 171 | $MDR_{out}$, $Y_{in}$ |
| 172 | $Rdst_{out}$, SelectY, Add, $Z_{in}$ |
| 173 | $Z_{out}$, $Rdst_{in}$, End |



# Mikroinstrukcije sa poljem sledeće adrese

Mikroinstrukcijski – sekvencirana organizacija






# Mikroinstrukcije sa poljem sledeće adrese

Microinstruction

| F0 | F1 | F2 | F3 |
|---|---|---|---|
| F0 (8 bits) | F1 (3 bits) | F2 (3 bits) | F3 (3 bits) |
| Address of next microinstruction | 000: No transfer<br>001: $PC_{out}$<br>010: $MDR_{out}$<br>011: $Z_{out}$<br>100: $Rsrc_{out}$<br>101: $Rdst_{out}$<br>110: $TEMP_{out}$ | 000: No transfer<br>001: $PC_{in}$<br>010: $IR_{in}$<br>011: $Z_{in}$<br>100: $Rsrc_{in}$<br>101: $Rdst_{in}$ | 000: No transfer<br>001: $MAR_{in}$<br>010: $MDR_{in}$<br>011: $TEMP_{in}$<br>100: $Y_{in}$ |

Format mikroinstrukcija sa adresiranjem sa širokim grananjem



# Mikroinstrukcije sa poljem sledeće adrese

| F4 | F5 | F6 | F7 |
|---|---|---|---|
| F4 (4 bits) | F5 (2 bits) | F6 (1 bit) | F7 (1 bit) |
| 0000: Add<br>0001: Sub<br>⋮<br>1111: XOR | 00: No action<br>01: Read<br>10: Write | 0: SelectY<br>1: Select4 | 0: No action<br>1: WMFC |

| F8 | F9 | F10 |
|---|---|---|
| F8 (1 bit) | F9 (1 bit) | F10 (1 bit) |
| 0: NextAdrs<br>1: InstDec | 0: No action<br>1: $OR_{mode}$ | 0: No action<br>1: $OR_{indsrc}$ |

Format mikroinstrukcija sa adresiranjem sa širokim grananjem



# Mikroinstrukcije sa poljem sledeće adrese

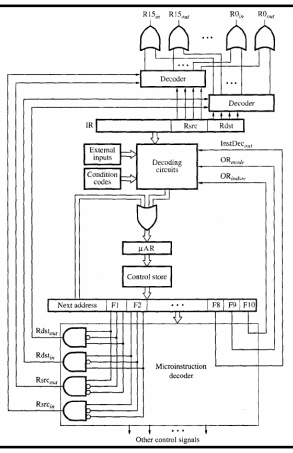
| Octal address | F0 | F1 | F2 | F3 | F4 | F5 | F6 | F7 | F8 | F9 | F10 |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 000 | 00000001 | 001 | 011 | 001 | 0000 | 01 | 1 | 0 | 0 | 0 | 0 |
| 001 | 00000010 | 011 | 001 | 100 | 0000 | 00 | 0 | 1 | 0 | 0 | 0 |
| 002 | 00000011 | 010 | 010 | 000 | 0000 | 00 | 0 | 0 | 0 | 0 | 0 |
| 003 | 00000000 | 000 | 000 | 000 | 0000 | 00 | 0 | 0 | 1 | 1 | 0 |
| 121 | 01010010 | 100 | 011 | 001 | 0000 | 01 | 1 | 0 | 0 | 0 | 0 |
| 122 | 01111000 | 011 | 100 | 000 | 0000 | 00 | 0 | 1 | 0 | 0 | 1 |
| 170 | 01111001 | 010 | 000 | 001 | 0000 | 01 | 0 | 1 | 0 | 0 | 0 |
| 171 | 01111010 | 010 | 000 | 100 | 0000 | 00 | 0 | 0 | 0 | 0 | 0 |
| 172 | 01111011 | 101 | 011 | 000 | 0000 | 00 | 0 | 0 | 0 | 0 | 0 |
| 173 | 00000000 | 011 | 101 | 000 | 0000 | 00 | 0 | 0 | 0 | 0 | 0 |

Implementacija mikrorutine za instrukciju Add(Rsrc)+, Rdst

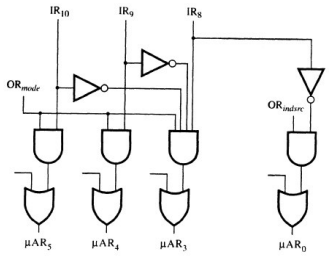


# Emulacija

Neki detalji elektronskih kola za generisanje kontrolnih signala



## Emulacija




Kontrolna kola za bitovno OR-ovanje (deo dekoderskih kola sa slike na prethodnom slajdu)



## Zadaci

-
-
-