Arhitektura procesora FRISC

Odabir skupa naredaba

Upravljačke naredbe

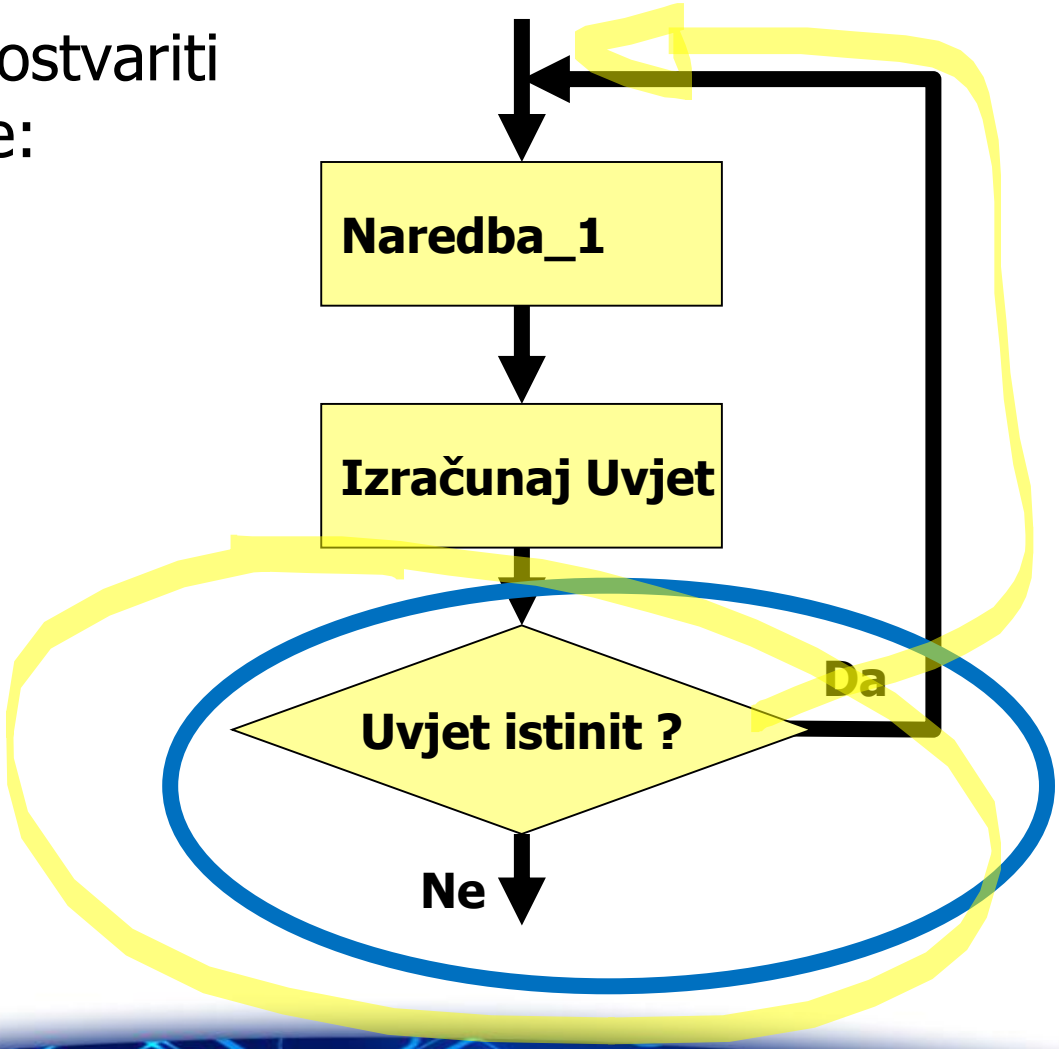
Upravljačke naredbe

- Za sada imamo:
  - Aritmetičko-logičke naredbe (ADD, SUB, AND, OR, XOR)
  - Memorijske naredbe (LOAD, STORE)
- Što **možemo** napraviti s ovim naredbama?
  - Ne puno, ali možemo ostvariti jednostavna izračunavanja pri čemu su podatci i rezultati u memoriji ili u registrima. Na primjer:
    - Izračunavanje aritmetičkih izraza: a+3-4+b+(c-12)+d
    - Izračunavanje operacija s bitovima:
      (a OR 00001111) XOR (b AND 00111100)
    - Izračunavanje logičkih izraza: a AND b XOR true
    - Sve kombinacije gore navedenih izraza gdje se naredbe izvode **slijedno** jedna iza druge

Upravljačke naredbe

- **Ne možemo** ostvariti petlju do-while:

do
Naredba_1
While (Uvjet)



kako ostvariti uvjetni skok

?

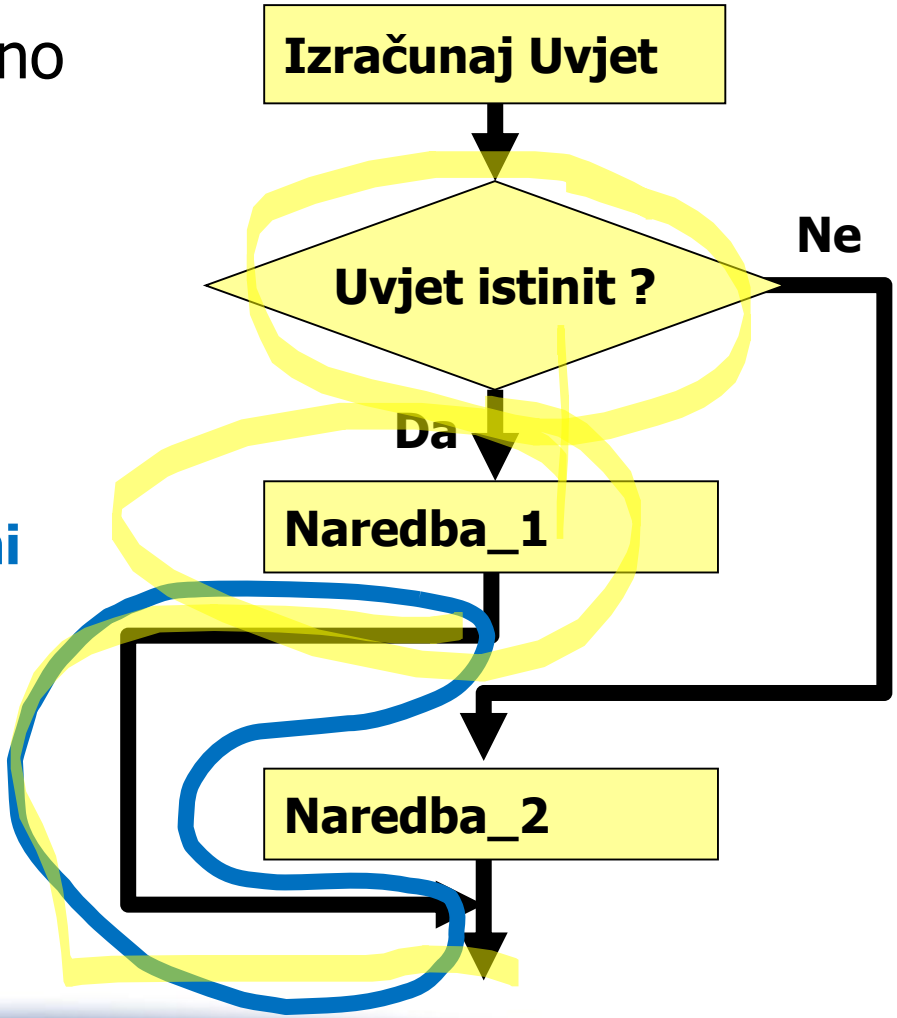
Upravljačke naredbe

- **Ne možemo** ostvariti uvjetno grananje:

```
if (Uvjet) then
    Naredba_1
else
    Naredba_2
endif
```

kako ostvariti bezuvjetni skok

?



Upravljačke naredbe

- Problem je što se AL-naredbe i memorijske naredbe izvode **isključivo slijedno**, tj. jedna iza druge - onim redoslijedom kojim su napisane

- **Zaključak:** nedostaje nam mogućnost mijenjanja redoslijeda normalnog slijednog izvođenja, tj. treba nam **naredba skoka**

- Naredbe skokova svrstavaju se u upravljačke (kontrolne) naredbe jer one upravljaju tijekom izvođenja programa.

Upravljačke naredbe

- Prema prethodnim dijagramima toka, sigurno će nam trebati dvije vrste naredbe skoka i to:
  - Naredba bezuvjetnog skoka (promjena redoslijeda izvođenja)
  - Naredba uvjetnog skoka (grananje na jednu od dvije naredbe u ovisnosti o uvjetu)

  >>>>

- Za obje naredbe, moramo imati operand kojim zadajemo **odredište skoka**, tj. adresu naredbe na koju želimo skočiti

Upravljačke naredbe

- Nazovimo naredbu **bezuvjetnog skoka** JP (od JUMP)

- Definirajmo način pisanja i operand naredbe JP:

```
JP  adresa
```

  - Naredba JP bezuvjetno skače na naredbu sa zadanom *adresom*
  - *Adresa* je zadana običnim brojem kao kod memorijskih naredaba (vidjet ćemo kasnije da se adresa također odnosi na četiri memorijske lokacije, što nećemo posebno naglašavati)
  - Uočite da je moguć skok unaprijed ili unazad
  - Kao i kod memorijskih naredaba, za *adresu* je umjesto broja moguće pisati labelu

Upravljačke naredbe

- Za **uvjetni skok** naredba se izvodi na dva moguća načina
  - Ako je uvjet ispunjen => skok se ostvaruje
  - Ako uvjet nije ispunjen => skok se ne ostvaruje, tj. izvodi se sljedeća naredba (ona koja je "ispod" naredbe skoka)

- Za naredbu uvjetnog skoka upotrijebimo isti naziv JP, ali ćemo mu dometnuti sufiks kojim ćemo označiti **uvjet**. Definirajmo način pisanja i operand naredbe JP:

```
JP_uvjet  adresa
```

  - Naredba *JP_uvjet* skače na naredbu sa zadanom *adresom* samo ako je *uvjet* istinit, a inače nastavlja s izvođenjem sljedeće naredbe

Upravljačke naredbe

- Promotrimo li malo bolje, vidimo da je naredba bezuvjetnog skoka samo **specijalni slučaj** uvjetnog skoka pri čemu je uvjet uvijek istinit

- Zato će postojati **samo jedna naredba za skok JP** koju pišemo na dva načina:

```
JP    100          // bezuvjetno skoči na naredbu na adresi 100
JP_uvjet   100     // skoči na 100 ako je uvjet istinit
```

- Kako se piše **uvjet** ? >>>>

Upravljačke naredbe - uvjeti

- Na primjer, uvjeti mogu biti sljedeći:
  - Jesu li dvije vrijednosti jednake?
  - Je li prva vrijednost veća od druge?
  - Je li prva vrijednost manja ili jednaka od druge?
  - itd. (općenito se uspoređuju dvije numeričke ili logičke vrijednosti)

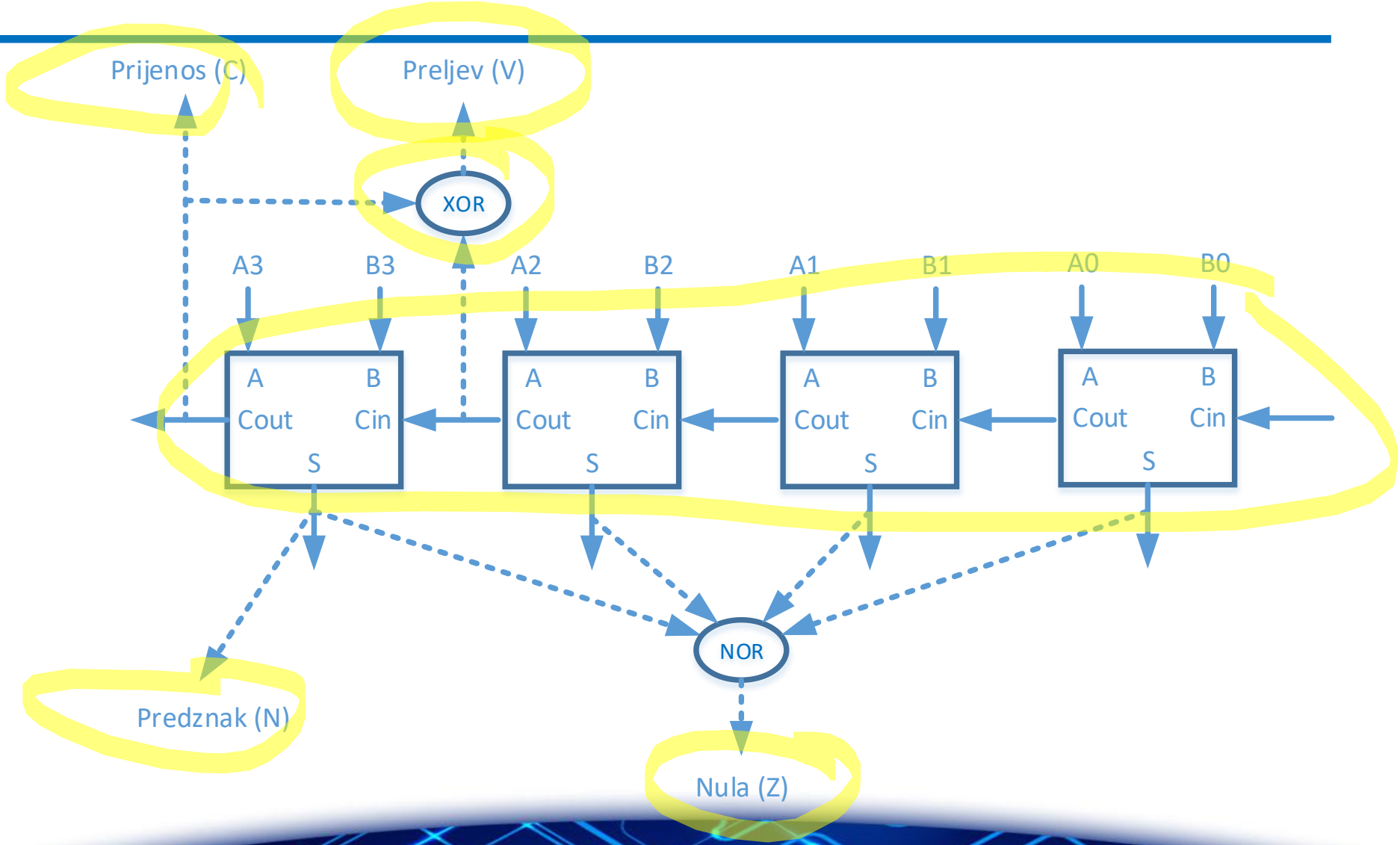
- U uvodnom poglavlju već smo vidjeli kako se mogu usporediti dvije vrijednosti:
  1. prvo se izvede AL-operacija (najčešće je to oduzimanje)
  2. nakon toga se ispitaju zastavice (Podsjetnik: zastavice su bistabili koji se postavljaju na temelju ALU-operacije)

Upravljačke naredbe - zastavice

- C – prijenos
- V - preljev (od engl. oVerflow, jer je slovo O previše slično znamenki 0)
- Z - nula
- N - predznak (od engl. negative)

- Ove zastavice se postavljaju pri izvođenju aritmetičko-logičkih naredaba

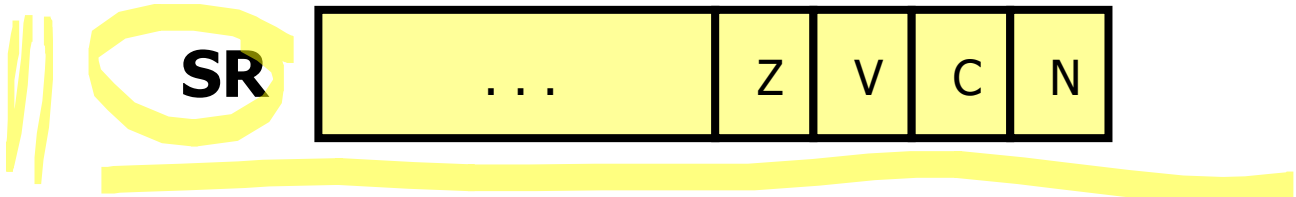
Zastavice



Upravljačke naredbe - registar SR

- Zastavice se nikada ne nalaze u procesoru kao zasebni bistabili, nego su uvijek unutar registra koji čuva i druge zastavice (npr. prekidne zastavice, zastavice koje označuju stanje procesora i sl. - više o tome kasnije)

- Zato ćemo i mi staviti zastavice u jedan registar koji ćemo nazvati registrom stanja SR (engl. status register). Kasnije ćemo mu dodati i druge zastavice koje nam budu trebale:



Upravljačke naredbe - zastavice

- Utjecaj aritmetičko-logičkih naredaba na zastavice:
  - ADD, SUB: C=prijenos, V=preljev, Z=nula, N=predznak
  - AND,OR,XOR: C=0, V=0, Z=nula, N=predznak

- Zastavice Z i N postavljaju se na temelju rezultata AL-naredbe
- Zastavice C i V se u logičkim naredbama brišu jer za logičke operacije prijenos i preljev nemaju smisla

Upravljačke naredbe - uvjeti

- Napravimo popis svih uvjeta koji bi nam mogli trebati u naredbi skoka JP:
  - Uvjeti koji izravno ispituju zastavice
    - Je li zastavica postavljena (set), tj. je li jednaka jedinici
    - Je li zastavica obrisana (clear, reset), tj. je li jednaka nuli
  - Uvjeti koji služe za usporedbu brojeva
    - Usporedba NBC-brojeva
    - Usporedba 2'k brojeva

>>>>

Uvjeti

- Prva skupina - izravno ispitivanje zastavica

Zapis	Značenje (Ispitivani uvjet)
C	C=1
NC	C=0
V	V=1
NV	V=0
Z	Z=1
NZ	Z=0
N	N=1
NN	N=0

Uvjeti

- Druga skupina - usporedbu brojeva (zasebno za NBC i 2'k brojeve)

Zapis	Značenje (engl.)		Ispitivani uvjet
ULE	Unsigned Less or Equal	≤	C=0 or Z=1
UGT	Unsigned Greater Than	>	C=1 and Z=0
ULT	Unsigned Less Than	<	C=0
UGE	Unsigned Greater or Equal	≥	C=1
SLE	Signed Less or Equal	≤	(N xor V)=1 or Z=1
SGT	Signed Greater Than	>	(N xor V)=0 and Z=0
SLT	Signed Less Than	<	(N xor V)=1
SGE	Signed Greater or Equal	≥	(N xor V)=0

Uvjeti

- Još neki praktični uvjeti

Zapis	Značenje (engl.)	Ispitivani uvjet
EQ	Equal	Z=1
NE	Not Equal	Z=0
M	Minus	N=1
P	Plus (Positive)	N=0

Upravljačke naredbe - primjeri

Primjer uvjetnog grananja - naredba if:

Zbrojiti registre R0 i R1 i rezultat staviti u R2. Ako nema prijenosa, onda ne treba napraviti ništa. Ako ima prijenosa (tj. zbroj u NBC-u je izašao iz opsega), onda treba pobrisati R2.

```
1)   R2 = R0 + R1;
     if( R2 nije ispravan) {
         R2 = 0;
     }
```

```
2)   R2 = R0 + R1;
     if( R2 je ispravan) {
     } else {
         R2 = 0;
     }
```

```
3)   R2 = R0 + R1;
     if( R2 je ispravan)
         goto dalje;
     R2 = 0;
  dalje: ...
```

Upravljačke naredbe - primjeri

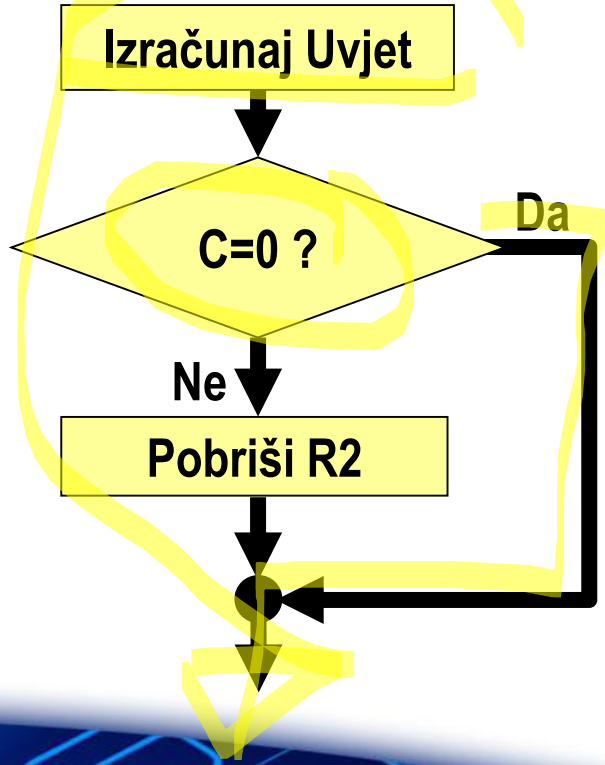
Primjer uvjetnog grananja - naredba if:

Zbrojiti registre R0 i R1 i rezultat staviti u R2. Ako nema prijenosa, onda ne treba napraviti ništa, a ako ima, onda treba pobrisati R2.

Rješenje:

```
        ADD R0,R1,R2    ; AL-operacija

        JP_NC DALJE     ; ispitivanje
                        ; zastavica
                        ; i skok
        SUB R2,R2,R2    ; briši R2
DALJE   ...             ; nastavak
                        ; programa
```



Upravljačke naredbe - primjeri

```
        ADD R0,R1,R2
        JP_NC  DALJE
        SUB R2,R2,R2
DALJE   ...
```

- Svaki uvjet može se napisati i na "obrnut" način. U praksi uvjet "okrećemo" tako da da dobijemo što kraći i razumljiviji program
- Prethodni program napisan s "obrnutim" uvjetom izgledao bi ovako:

```
        ADD    R0,R1,R2
        JP_C   BRISI
        JP     DALJE
BRISI   SUB    R2,R2,R2
DALJE   ...
```

Upravljačke naredbe - primjeri

- **Primjer uvjetnog i bezuvjetnog grananja - naredba if-else:**
- Zbrojiti registre R0 i R1 i rezultat staviti u R2. Ako dođe do prijenosa treba obrisati memorijsku lokaciju REZ, a inače u nju treba upisati R2.
- **Rješenje:**

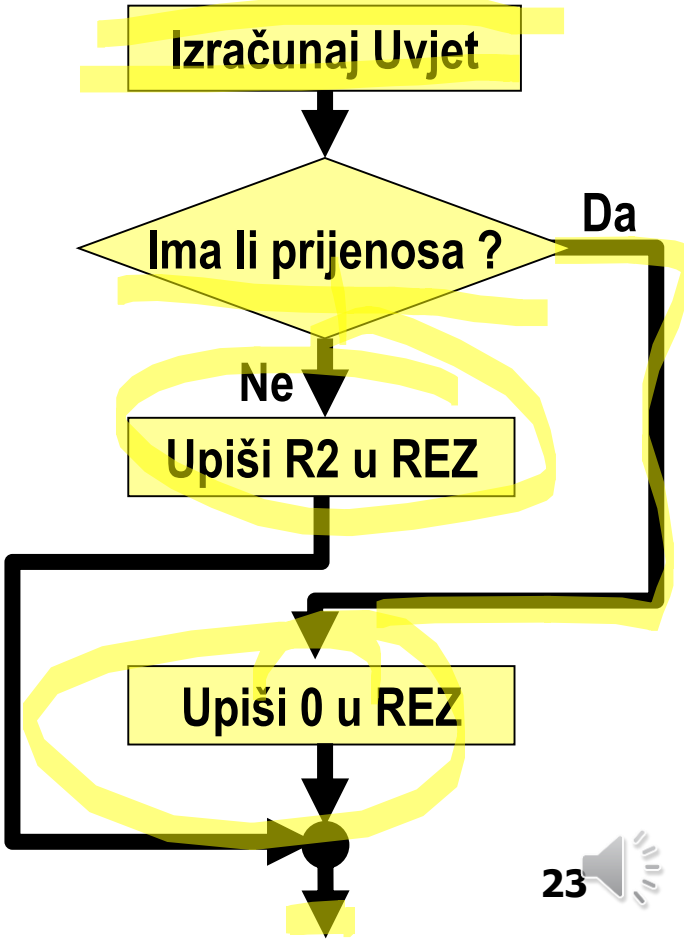
```
        ADD R0,R1,R2

        JP_C  BRISI

PISI    STORE R2,(REZ)  ; upiši R2
                        ; u REZ
        JP  DALJE

BRISI   SUB R3,R3,R3    ; obriši
        STORE R3,(REZ)  ; REZ

DALJE   ...
```



Upravljačke naredbe - primjeri

Primjer petlje u postupku množenja:

Treba pomnožiti dva NBC broja (označimo to kao A*B) koji su smješteni u memoriji na adresama 100 i 200. Rezultat množenja treba spremiti u memoriju na adresu 300.

Rješenje:

Program ćemo temeljiti na dijagramu toka. U programu ćemo vidjeti većinu onoga što smo do sada naučili:

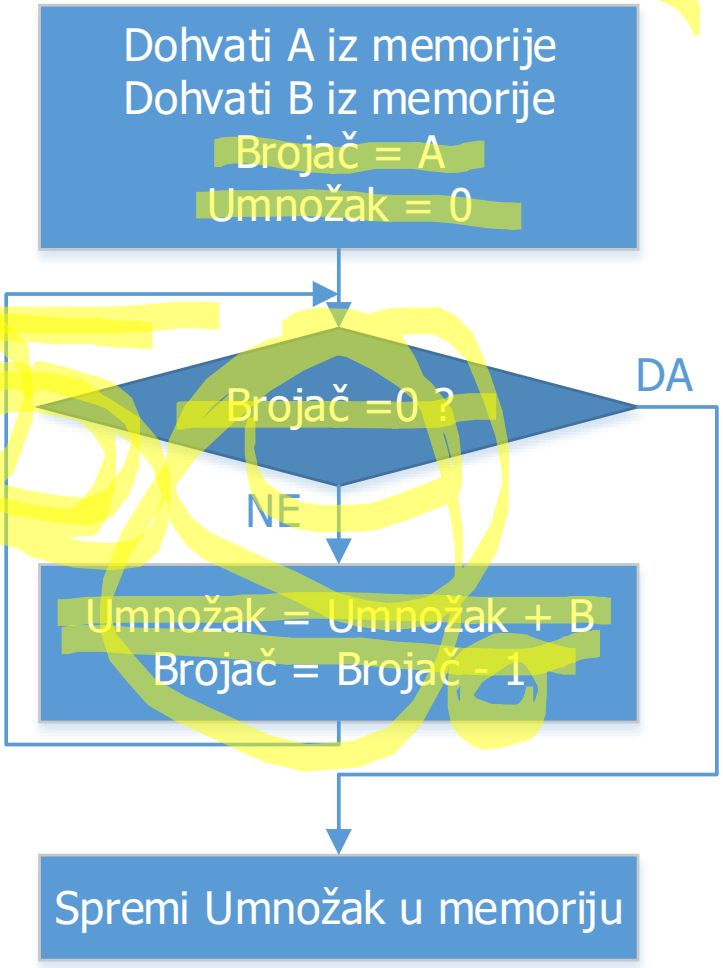
- AL-naredbe,
- memorijske naredbe,
- rad s konstantama,
- naredbu uvjetnog i bezuvjetnog skoka.

Dodatno ćemo vidjeti i petlju s brojačem.

```
        LOAD   R0, (100)      ; R0 ≡ A
        LOAD   R1, (200)      ; R1 ≡ B
        OR     R0, R0, R2     ; R2 ≡ Brojač := A
        LOAD   R3, (400)      ; R3 ≡ Umnožak := 0
        LOAD   R4, (400)      ; R4 ≡ 0
        LOAD   R5, (410)      ; R5 ≡ 1

PETLJA  SUB    R2, R4, R2
        JP_EQ  KRAJ

        ADD    R3, R1, R3
        SUB    R2, R5, R2
        JP     PETLJA
KRAJ    STORE  R3, (300)
...
400     DW     0
410     DW     1
```



Strojni kôd naredaba

Općenito

Strojni kôd naredaba

- Za sada smo definirali 8 naredaba i opisali što te naredbe rade

- Ponovimo:
  - procesor svaku naredbu mora dohvatiti, dekodirati i izvesti

  - dekodiranje je, zapravo, raspoznavanje naredbe kako bi se znalo što točno treba izvesti

  - svaka naredba je zapisana u memoriji (kao niz nula i jedinica) i ima svoj posebni oblik po kojem procesor prepoznaje tu naredbu

- Dakle, svaka naredba mora imati **jedinstveni** zapis, koji će je **razlikovati** od svih drugih naredaba

Strojni kôd naredaba - FRISC

- Za sve naredbe koje smo do sada odabrali moramo definirati strojne kôdove (vodeći računa o budućim proširenjima procesora)

- Pokazat ćemo kako su zamišljeni strojni kôdovi, s time da razlozi za definiciju pojedinih bitova neće biti jasni odmah, nego tek kad budemo imali proširenu verziju procesora

- Zbog jednostavnosti ćemo objašnjenja ovakvih slučajeva odgoditi za kasnije i nećemo sada definirati strojni kôd da bude efikasan za osam dosadašnjih naredaba, jer bi onda kasnije morali mijenjati tako definirane strojne kôdove

Strojni kôd naredaba - FRISC

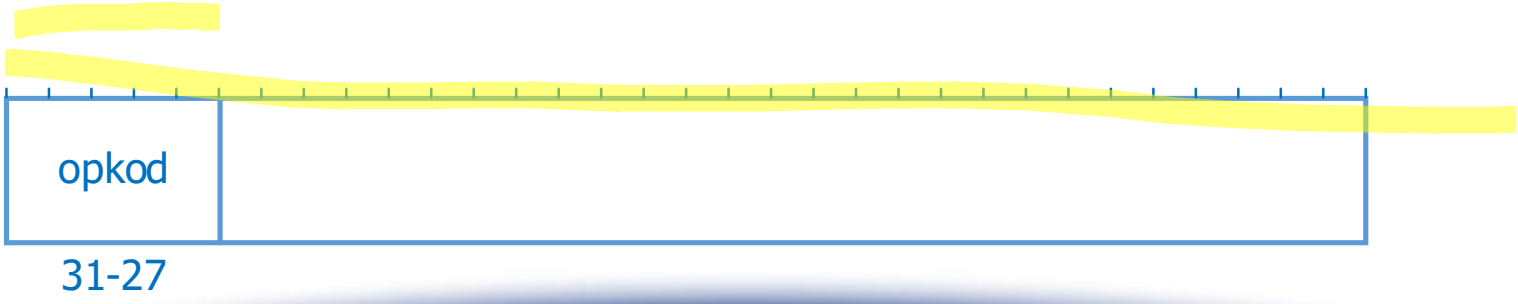
- Mi projektiramo ugradbeni procesor koji:
  - treba biti što jednostavniji
  - treba imati mali broj naredaba
  - ima mali broj registara
  - treba imati mali broj načina adresiranja

- Zato, odabiremo da širina strojnog kôda bude procesorska riječ, odnosno 32 bita za sve naredbe

- Također ćemo se truditi da polja budu raspoređena što pravilnije kako bi se pojednostavnilo i ubrzalo dekodiranje naredaba

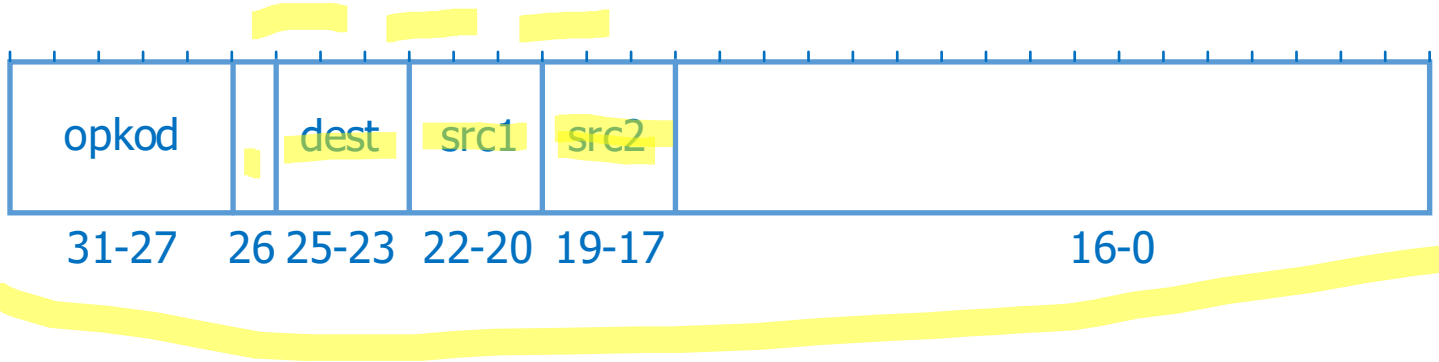
Strojni kôd naredaba

- **Polje operacijskog kôda** jednoznačno definira o kojoj se naredbi radi
- Za operacijski kôd moramo uzeti dovoljno bitova da možemo razlikovati sve potrebne naredbe (ili grupe naredaba). Odabiremo da će polje operacijskog kôda (opkod) zauzimati 5 bitova što nam daje ukupno $2^5 = 32$ kombinacija čime možemo izravno razlikovati 32 naredbe (za sada imamo samo 8 naredaba, ali ćemo ih kasnije proširivati).



Strojni kôd naredaba

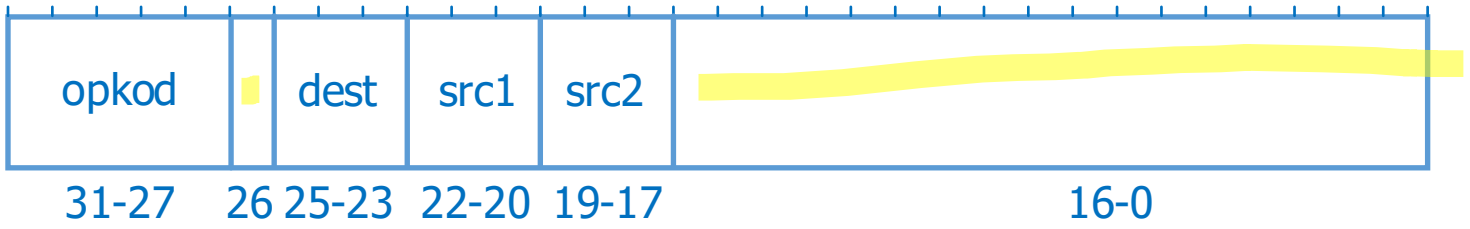
- **Polja operanada** jednoznačno definiraju s kojim operandima naredba obavlja svoju zadaću
- Na primjer, za naredbu ADD:
  - Koji registar će biti prvi operand zbrajanja
  - Koji registar će biti drugi operand zbrajanja
  - U koji registar se sprema rezultat
- Za FRISC je definirano da AL naredbe imaju tri operanda i svi su registri. Kako smo definirali da FRISC ima 8 registara opće namjene svaki operand kôdiramo sa po tri bita.



Strojni kôd naredaba

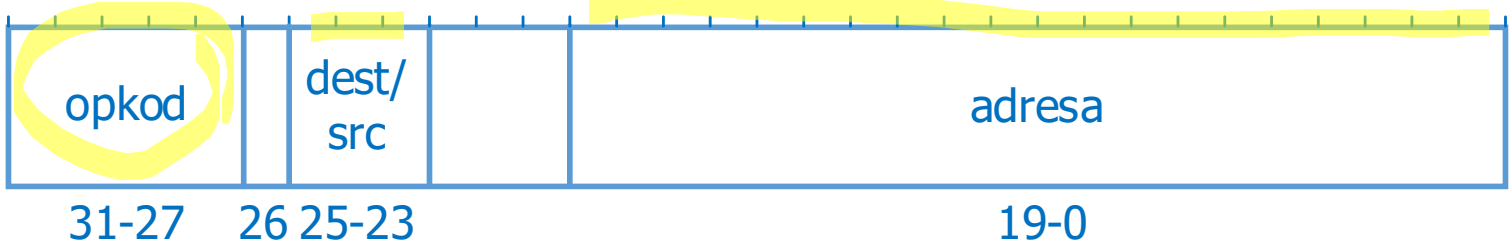
- Iz ovako definiranog strojnog kôda vidimo da imamo neiskorišteno ukupno 18 bitova

- Njih ćemo kasnije upotrijebiti za daljnja proširenja i poboljšanja AL-naredaba



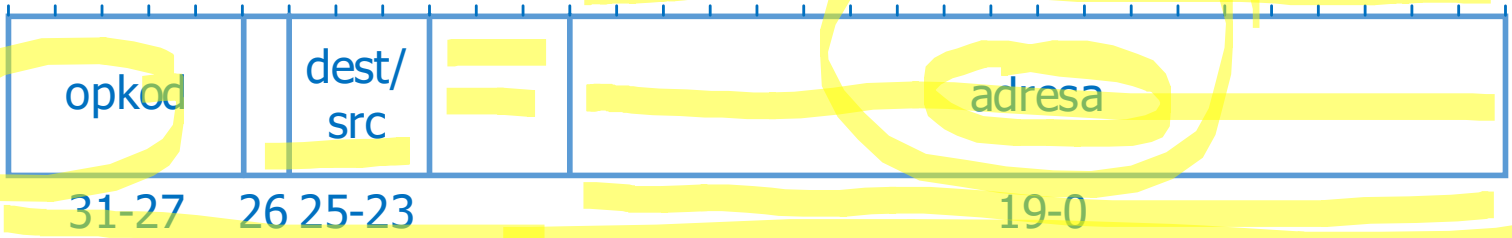
Strojni kôd - memorijske naredbe

- Memorijske naredbe imaju dva operanda:
  - prvi operand je registar čije značenje ovisi o naredbi:
    - za LOAD znači odredišni registar u koji se puni vrijednost (dest)
    - za STORE znači izvorišni registar iz kojeg se čita vrijednost (src)
  - drugi operand je adresa memorijske lokacije s koje se čita/piše podatak
- Nakon operacijskog kôda, kodiramo redom:
  - jedan bit (26) je neiskorišten
  - tri bita (23 do 25) za prvi operand, tj. za registar (dest/src)
  - tri bita (20 do 22) se ne koriste
  - preostalih 20 bitova (0 do 19) sadrže drugi operand, tj. adresu



Strojni kôd - memorijske naredbe

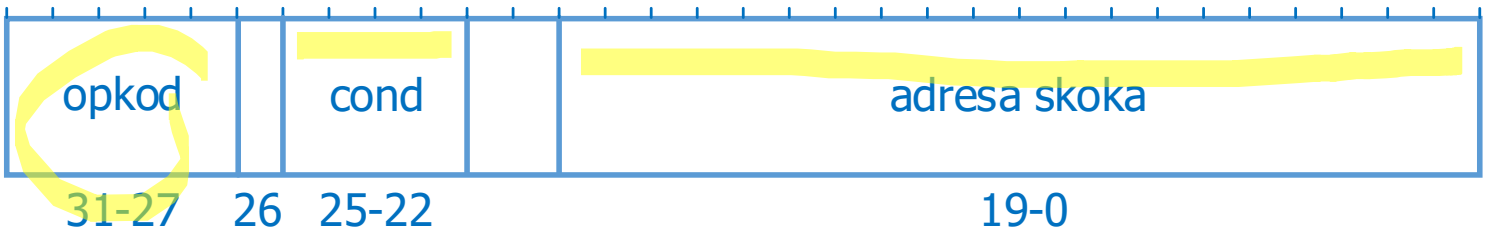
- Vidimo da je adresa ograničena na samo 20 bitova, iako smo definirali da će adresna sabirnica i adresa imati 32 bita
- Na ovom primjeru vidimo kako odluka da svaka naredba ima samo 32 bita ograničava podatke i adrese koje moramo kodirati u naredbi
- Posljedice ovog ograničenja objasnit ćemo kasnije



Strojni kôd - upravljačke naredbe

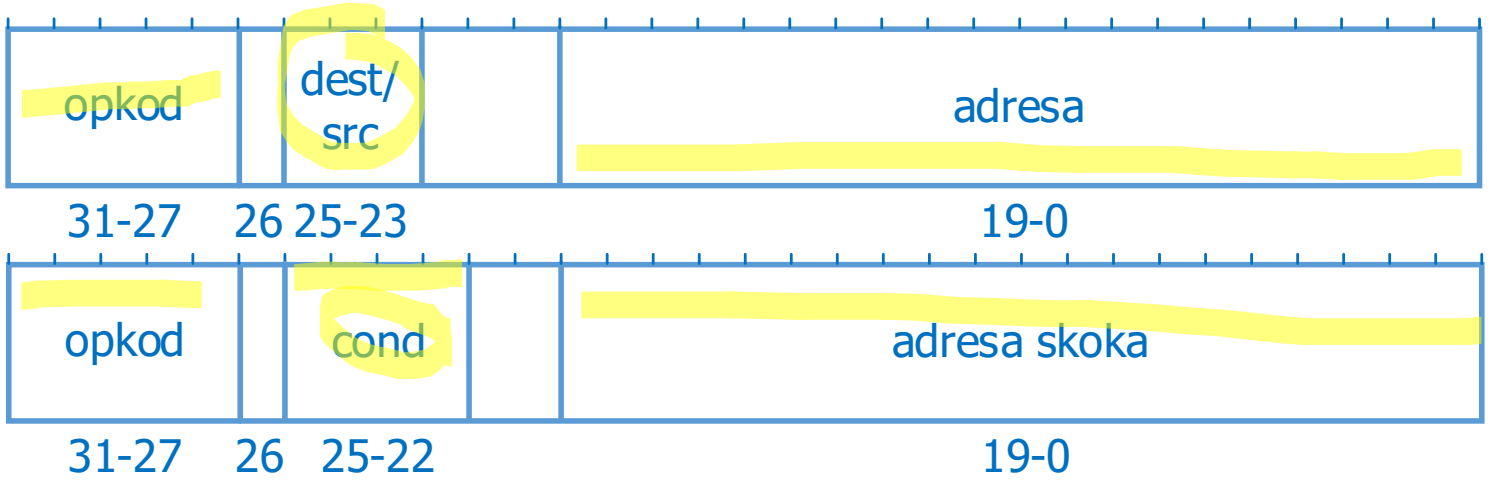
- Upravljačke naredbe imaju uvjet i jedan operand:
  - uvjet se piše kao dio naredbe (sufiks)
  - operand je adresa skoka (20 bita)

- Nakon operacijskog kôda, kodiramo redom:
  - jedan bit (26) je neiskorišten
  - četiri bita (22 do 25) za uvjet (cond)
  - dva bita (20 do 21) se ne koriste
  - preostalih 20 bitova (0 do 19) sadrže operand, tj. adresu skoka



Strojni kôd - upravljačke naredbe

- Usporedimo polje adrese s istim poljem u memorijskim naredbama:
  - i ovdje adresa ima samo 20 bitova od potrebnih 32
  - razlika u značenju adrese je da ovdje adresa predstavlja odredište skoka, a ne položaj podatka za čitanje/pisanje
  - budući da obje naredbe imaju jednako polje za adresu, strojni kôd je pravilniji pa će i dekodiranje i izvođenje naredaba biti jednostavnije i brže (barem tako očekujemo)



Strojni kôd - upravljačke naredbe

- Pogledajmo još je li za kodiranje uvjeta dovoljno 4 bita kojima se može kodirati 16 različitih uvjeta?

- Iz popisa uvjeta imamo sljedeće različite uvjete:
  - 8 uvjeta za izravno ispitivanje zastavica
  - 8 uvjeta za usporedbu brojeva
  - 2 uvjeta za ispitivanje jednakosti brojeva: EQ, NE
  - 2 uvjeta za ispitivanje predznaka: M i P
  - 1 uvjet koji je uvijek istinit (za bezuvjetni JP)

- Ukupno imamo 21 različitih uvjeta što je više od 16 mogućih pa na prvi pogled izgleda da imamo premalo bitova u polju

Strojni kôd - upravljačke naredbe

<<<<

- Ali, ovdje se radi samo o različitim načinima **pisanja** uvjeta

- Bitno je koliko postoji različitih načina **ispitivanja zastavica**, tj. koliko postoji različitih načina izvođenja naredbe, jer se samo to mora razlikovati u strojnom kôdu

- Iz tablica uvjeta vidimo da dio uvjeta ispituje zastavice na jednak način i da postoji samo **15 različitih** ispitivanja zastavica
- Dakle: 4 bita su dovoljna

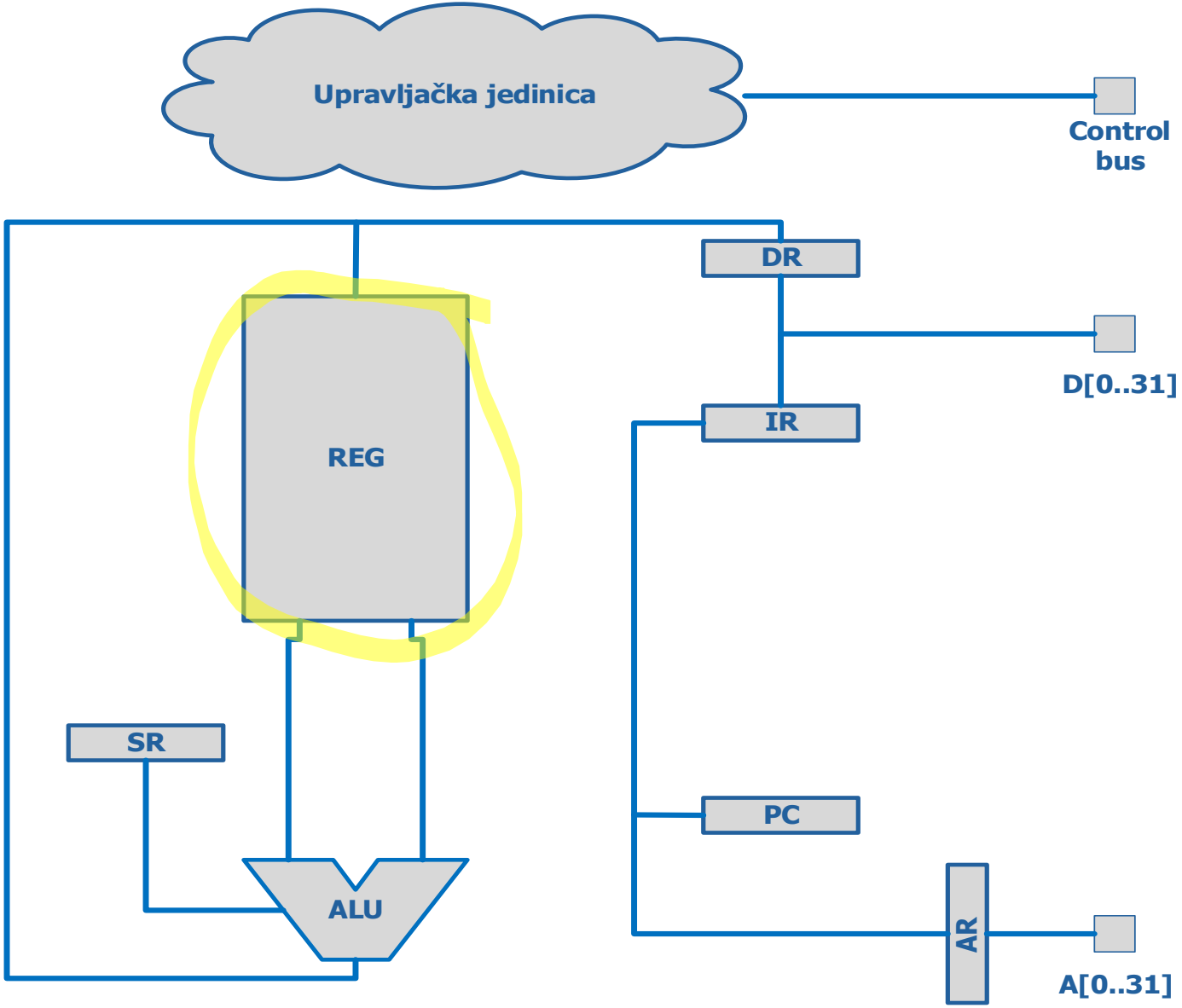
Načelna mikroarhitektura

Načelna mikroarhitektura

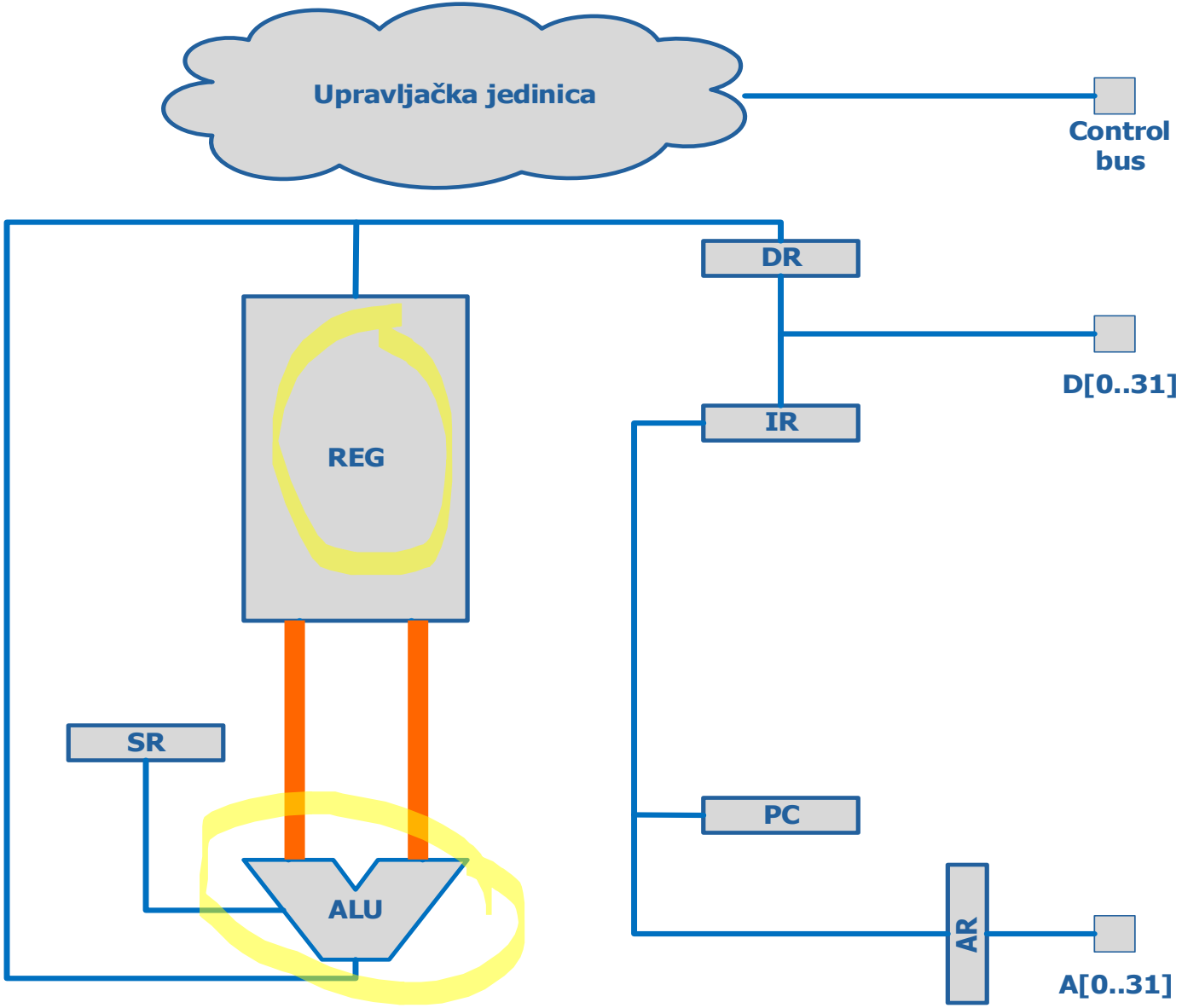
- Do sada smo se koncentrirali na naredbe bez objašnjavanja mikroarhitekture koja bi te naredbe mogla izvoditi
- Pogledajmo načelnu mikroarhitekturu našeg procesora:
  - koje dijelove sadrži
  - kako su ti dijelovi povezani

- Slika će za početak biti shematska i pojednostavljena (u budućim predavanjima ćemo je prikazivati sve detaljnije)

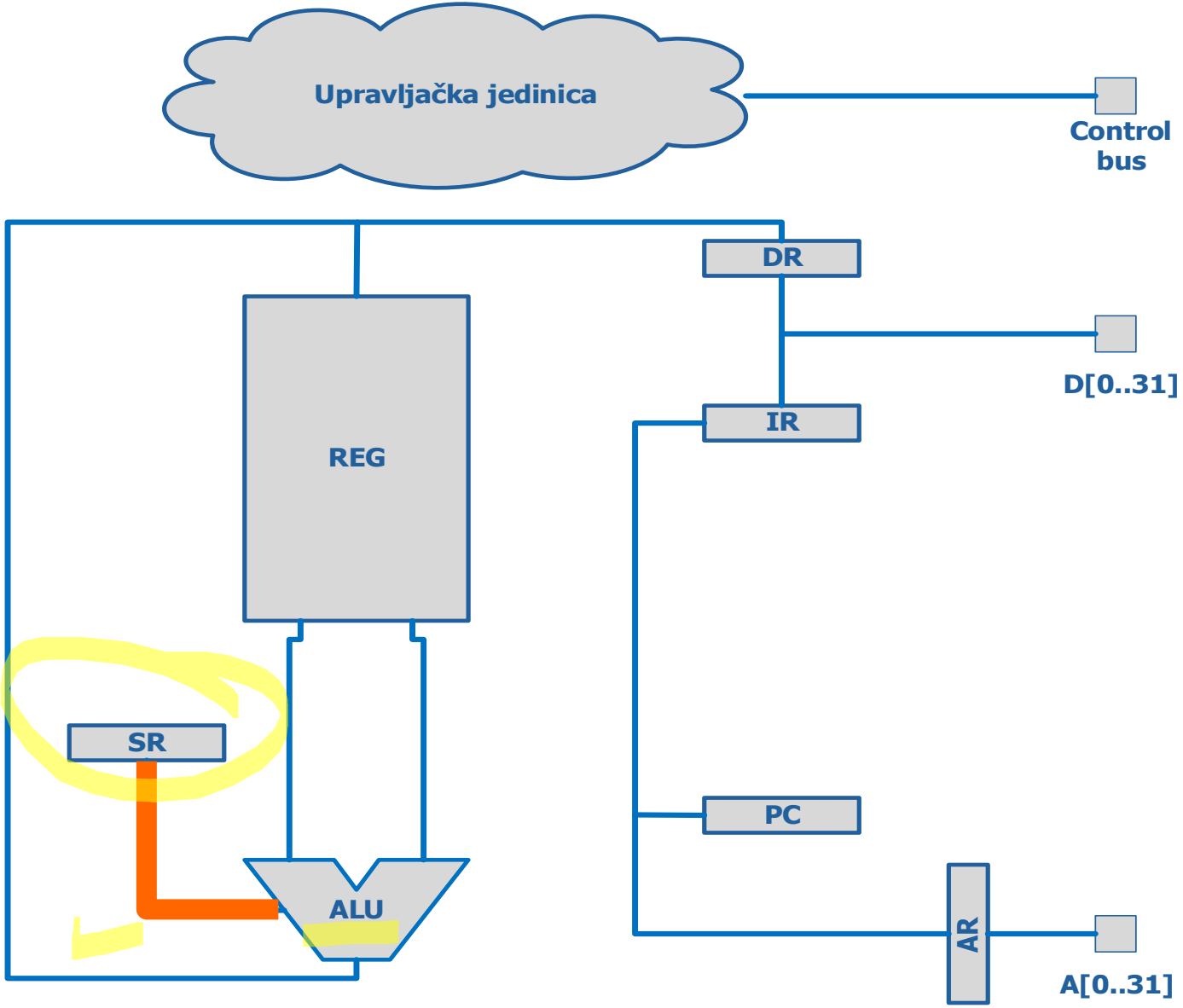
>>>>



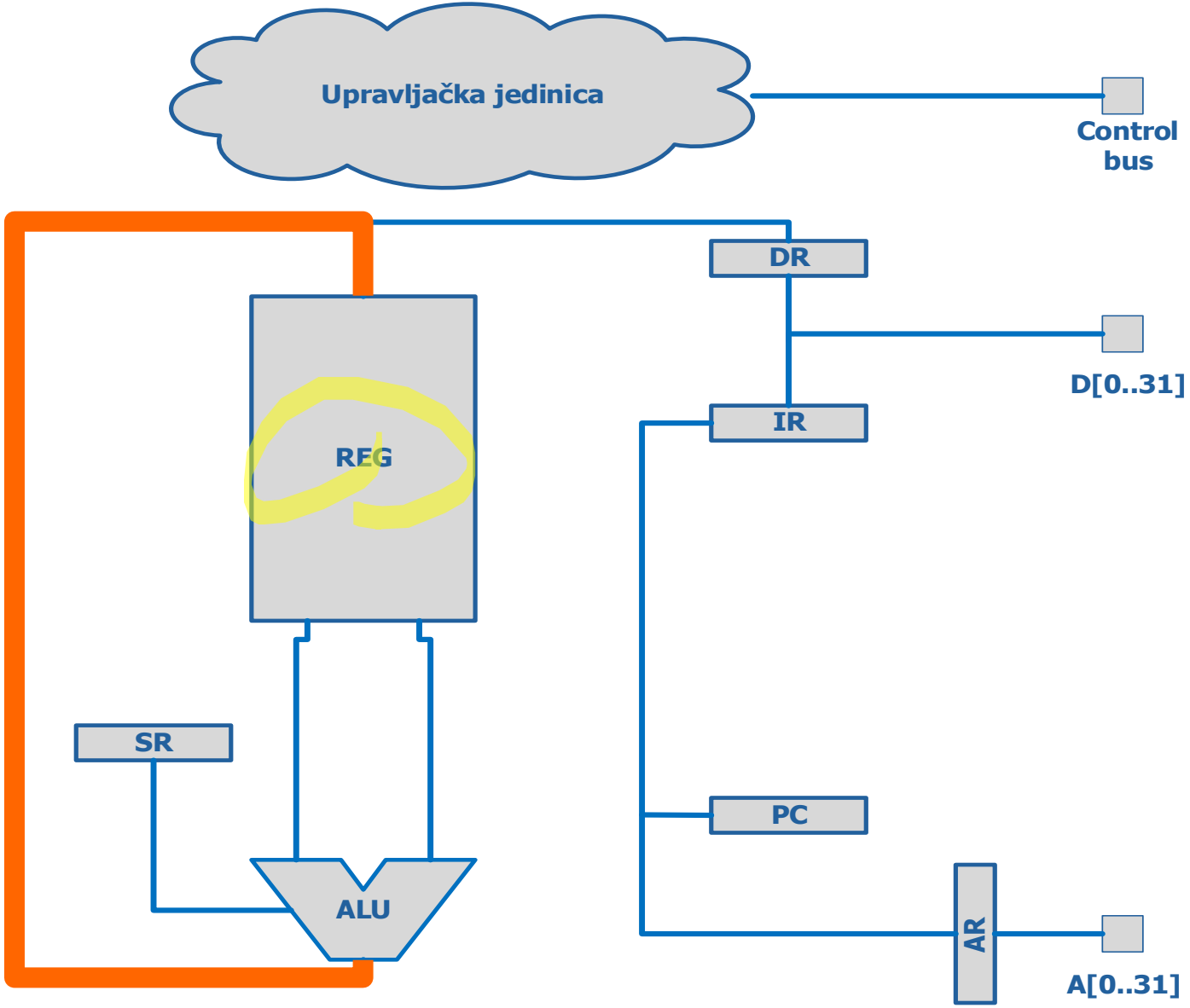
Na slici vidimo skup registara opće namjene (REG) i ostale registre procesora kao i način na koji su načelno spojeni >>>>



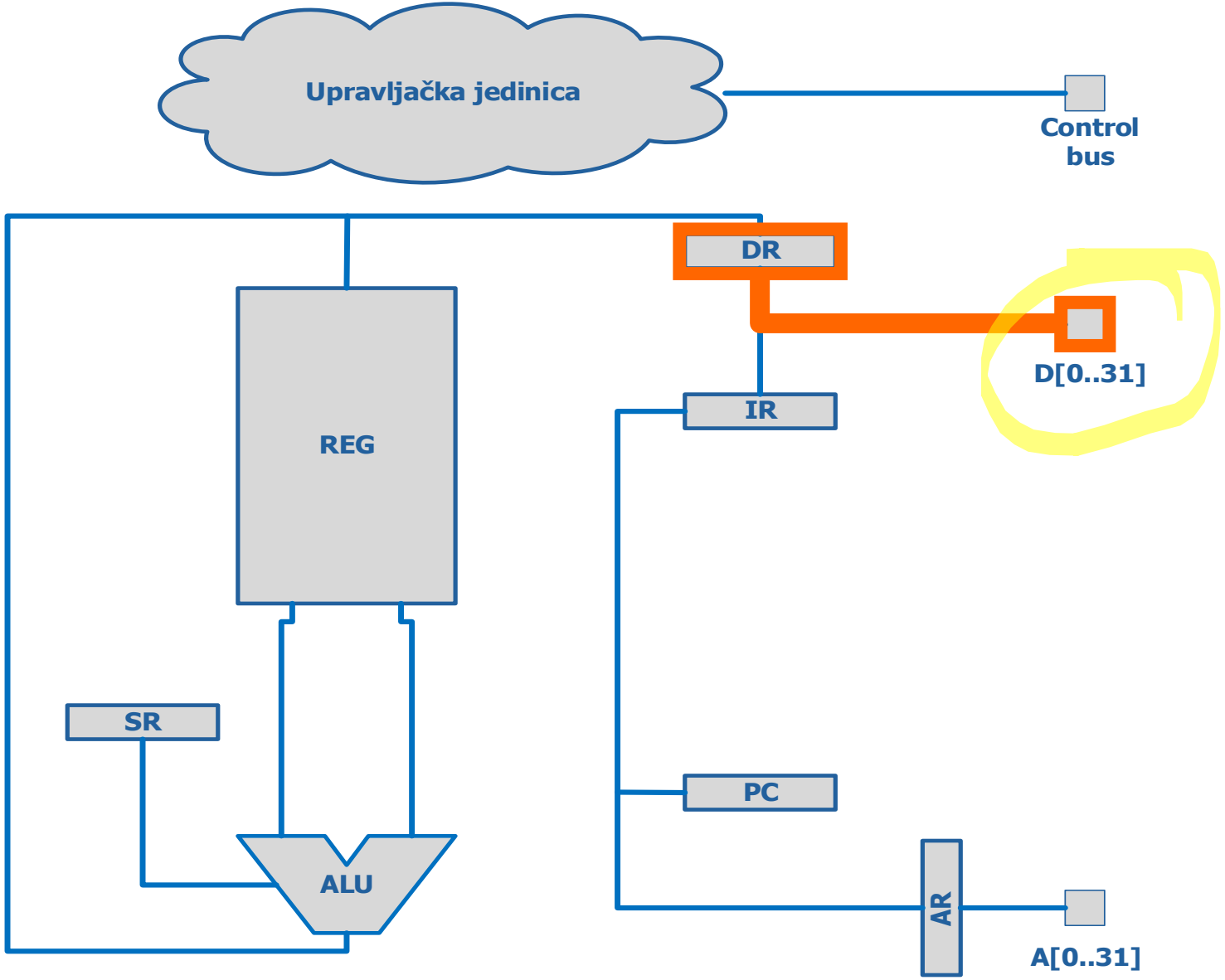
Iz registara opće namjene postoje dva spojna puta do ALU za slanje operanada u AL-naredbama



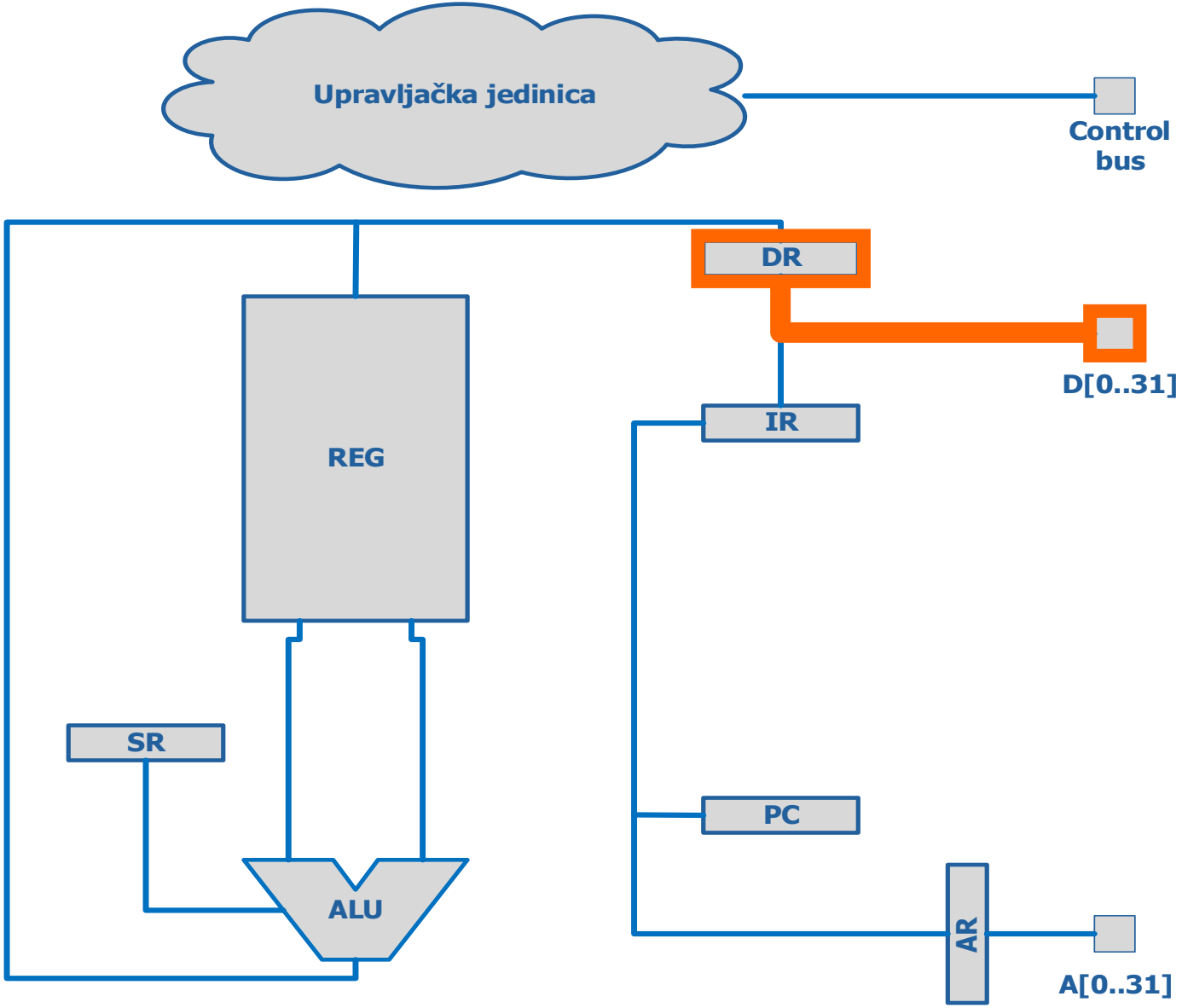
Iz ALU postoji veza do registra stanja SR, kako bi AL-operacija mogla utjecati na stanje zastavica



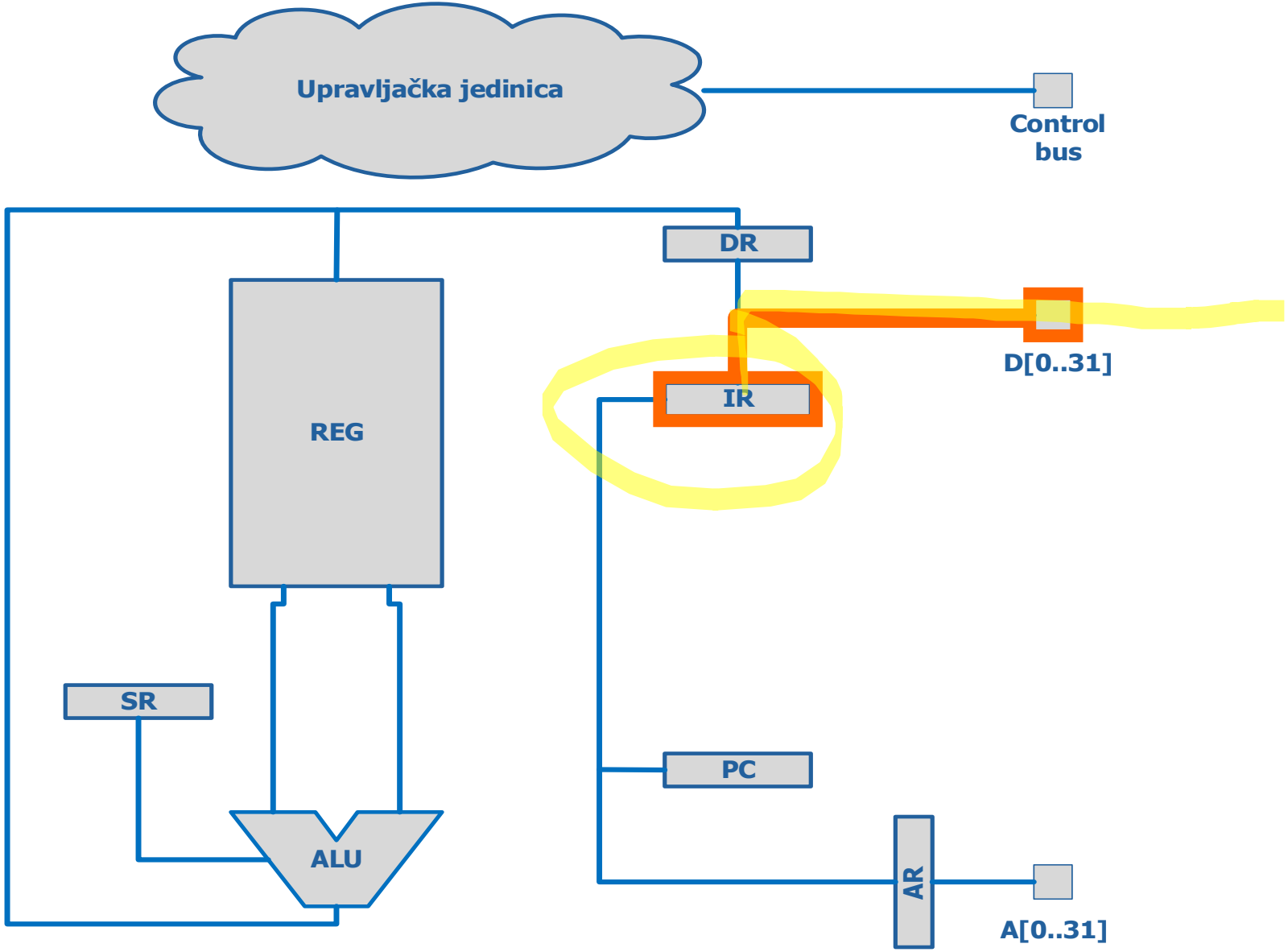
Postoji spojni put od izlaza ALU do skupa općih registara, kako bi u AL-naredbama rezultat mogao biti zapisan u jedan od općih registara



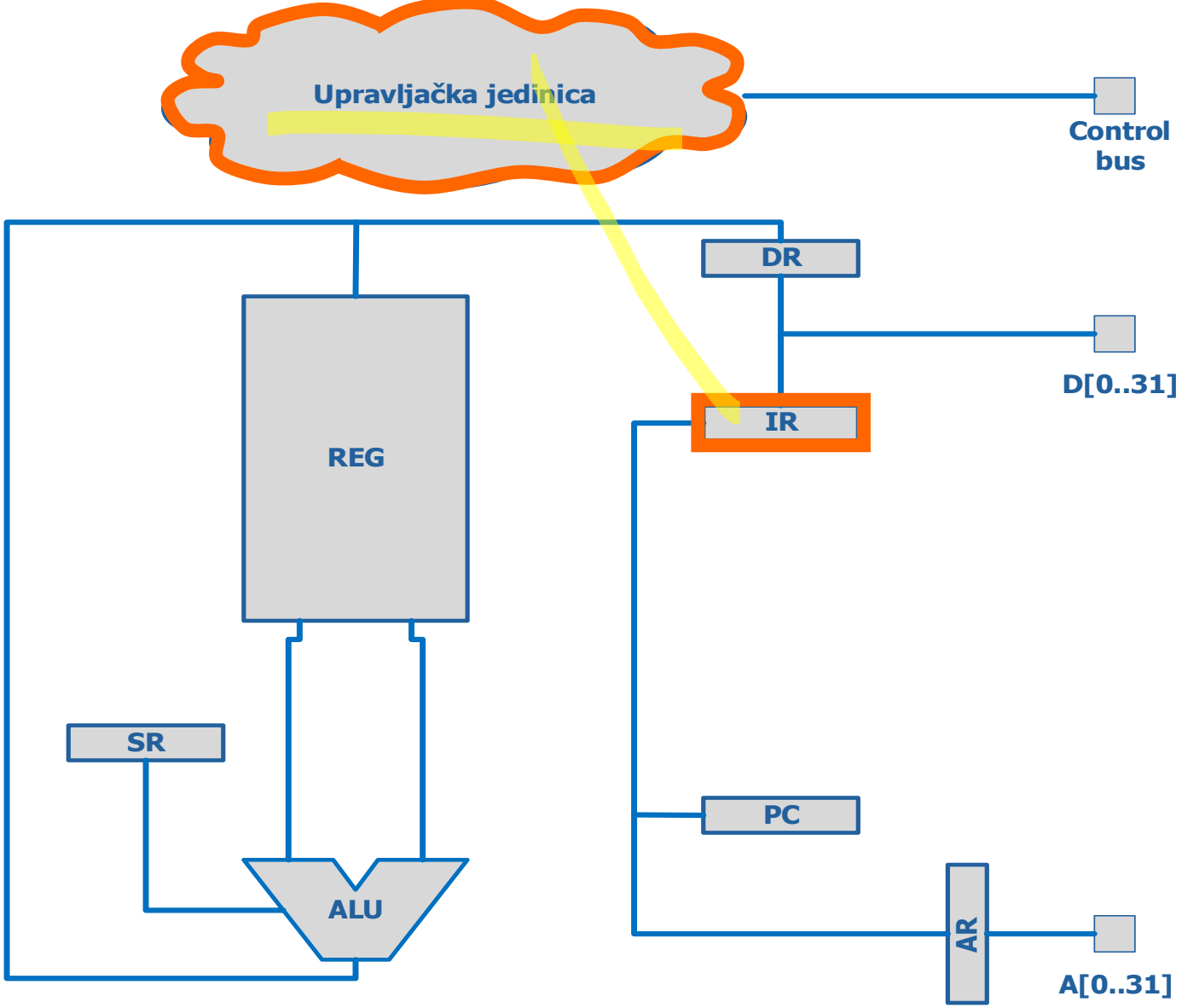
Registar podataka DR (Data Register) služi kao međuspremnik između unutrašnjosti procesora i sabirnice podataka, koja je spojena na procesor preko podatkovnih priključaka >>>>



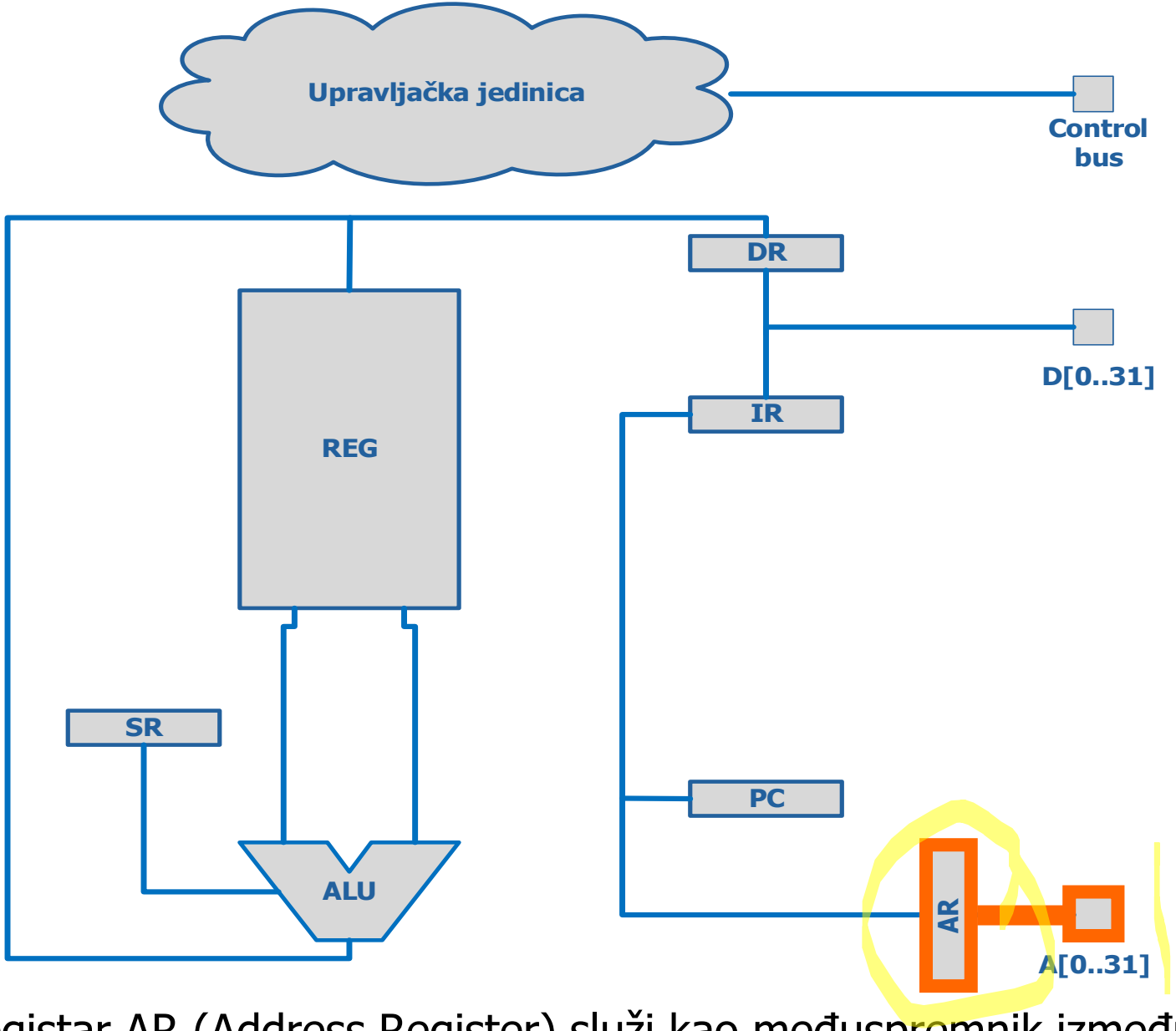
Podatci koji se čitaju i pišu preko sabirnice podataka uvijek prolaze kroz registar DR (uz jednu iznimku) >>>>



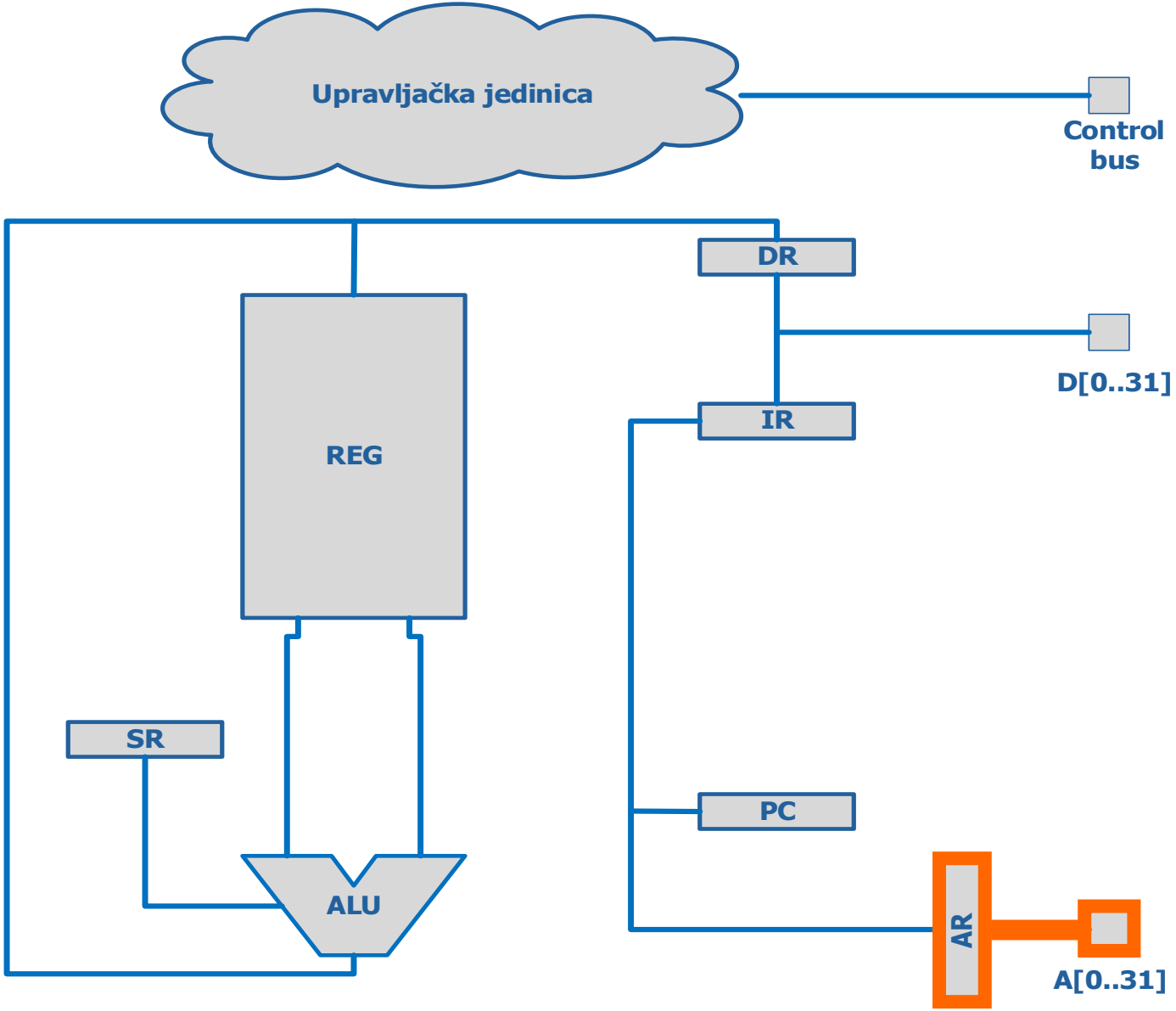
Ta iznimka je dohvaćanje naredbe iz memorije. Tada se ona ne sprema u DR, nego izravno u naredbeni registar IR (Instruction Register)



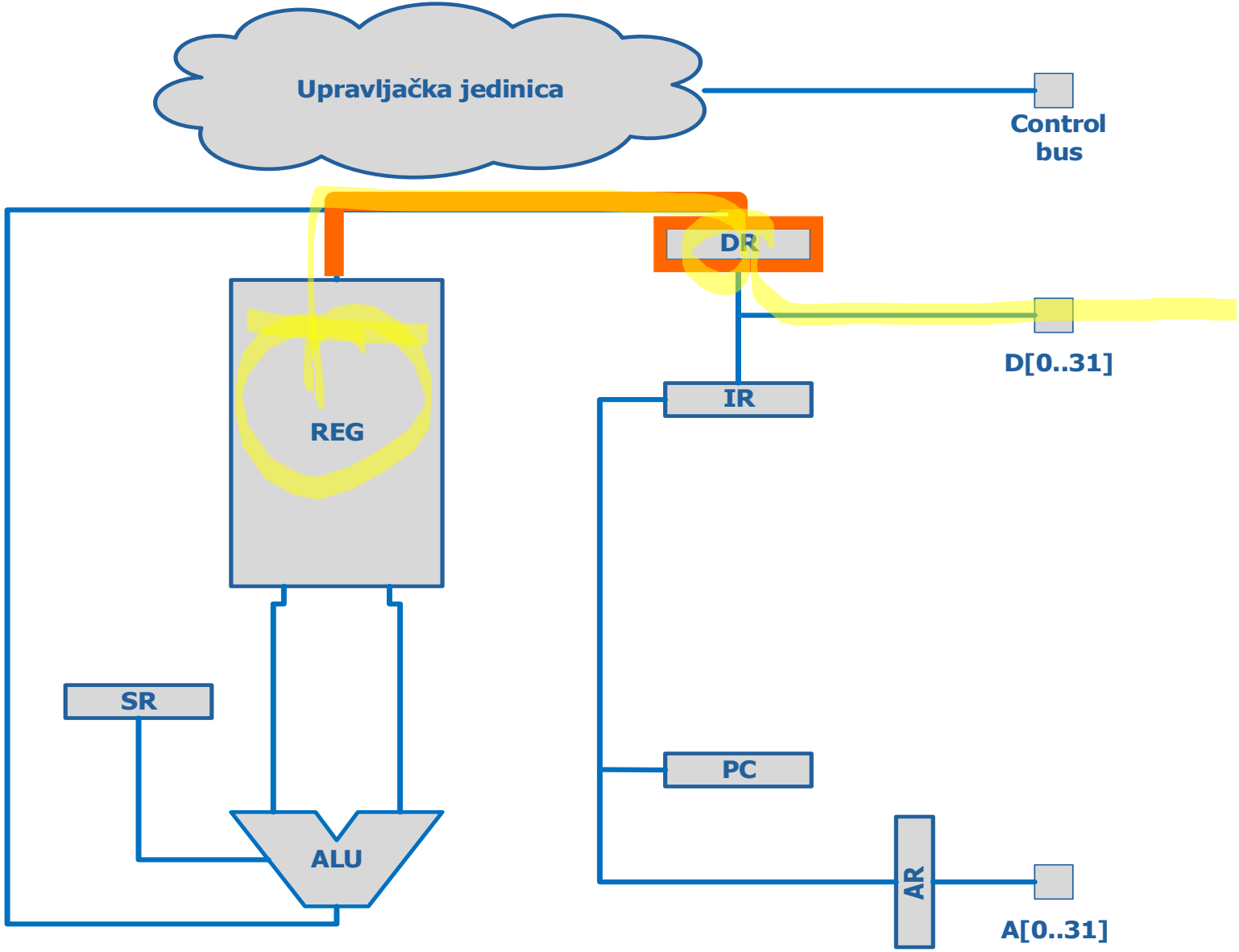
Za vrijeme dekodiranja naredbe, njezin strojni kôd je spremljen u registru IR i čita ga upravljačka jedinica (veza od IR do upravljačke jedinice nije prikazana na slici).



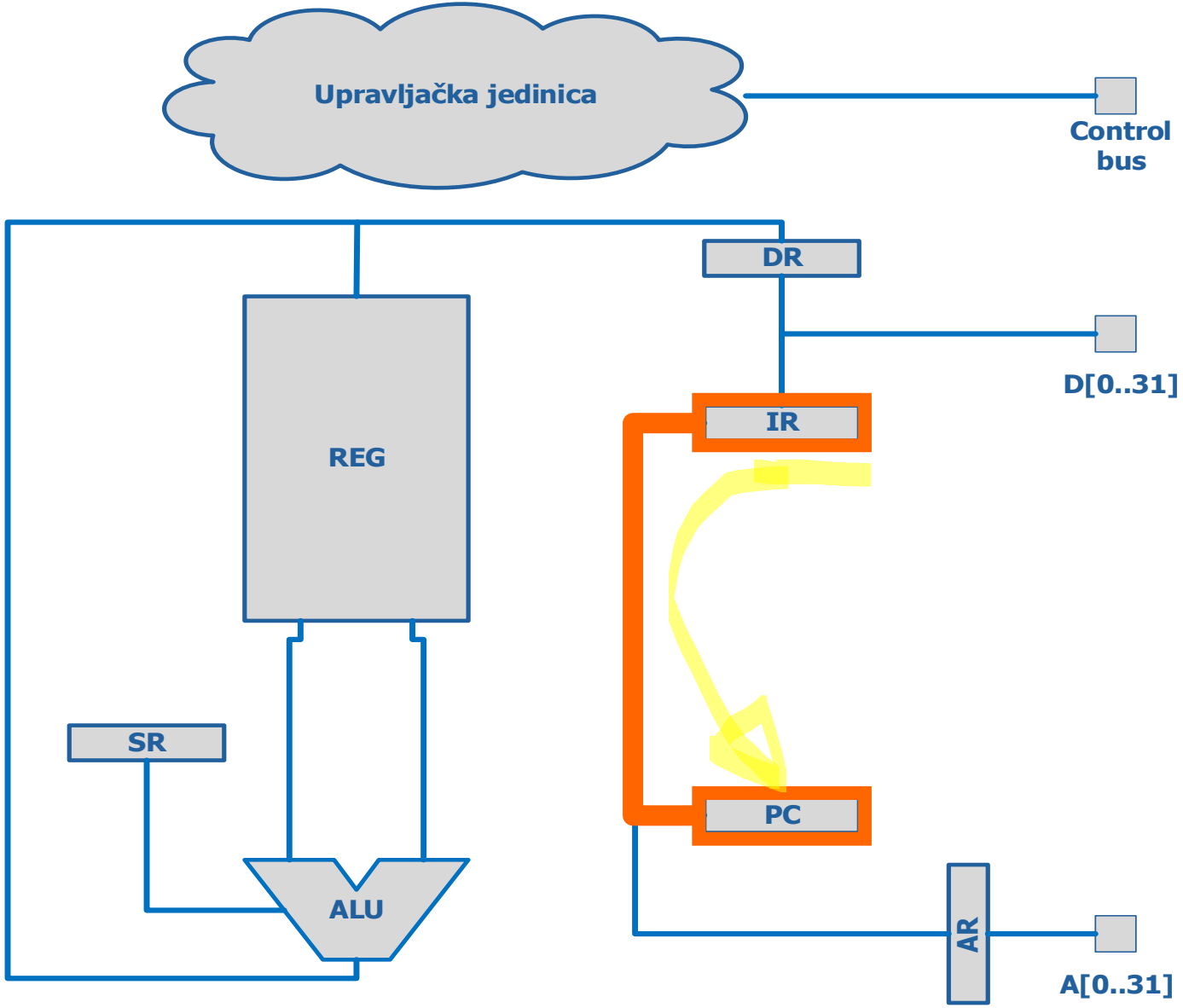
Adresni registar AR (Address Register) služi kao međuspremnik između unutrašnjosti procesora i adresne sabirnice, koja je spojena na procesor preko adresnih priključaka. >>>>



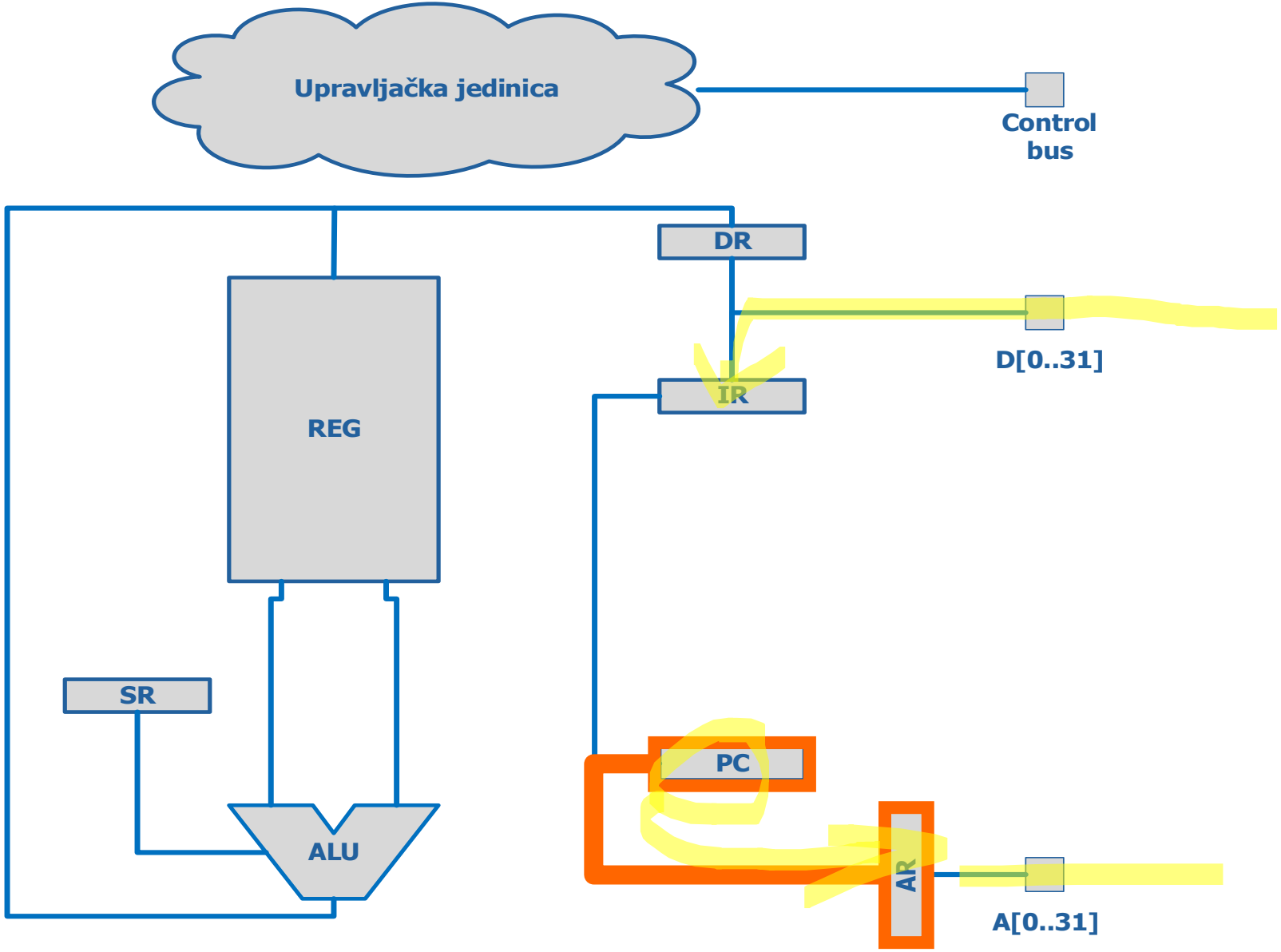
Registar AR uvijek se koristi kad procesor pristupa memoriji. U AR se sprema adresa memorijske lokacije koju procesor želi čitati ili pisati.



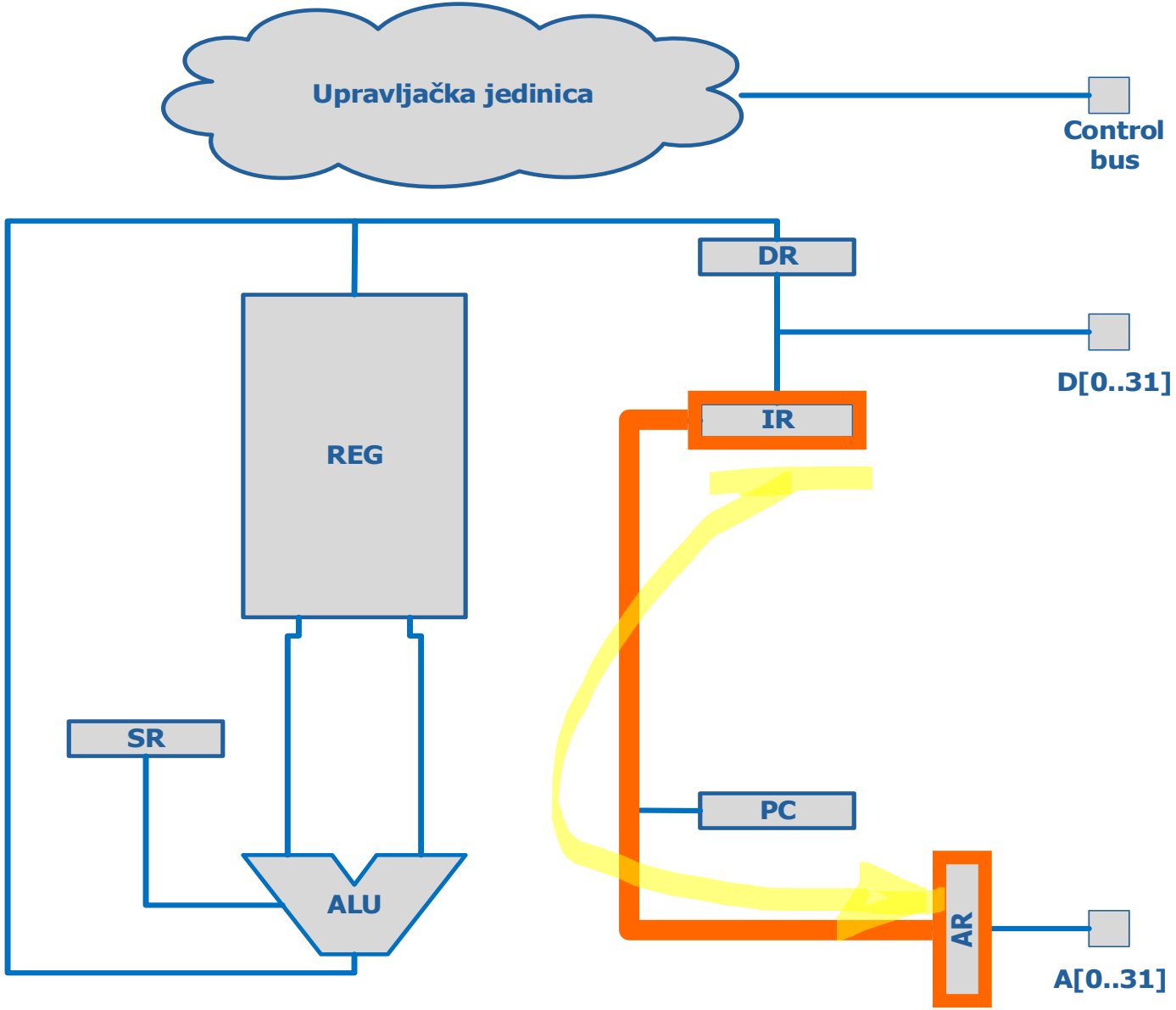
Postoji spojni put između registra DR i općih registara jer tim putom prolaze podatci prilikom izvođenja naredaba LOAD i STORE



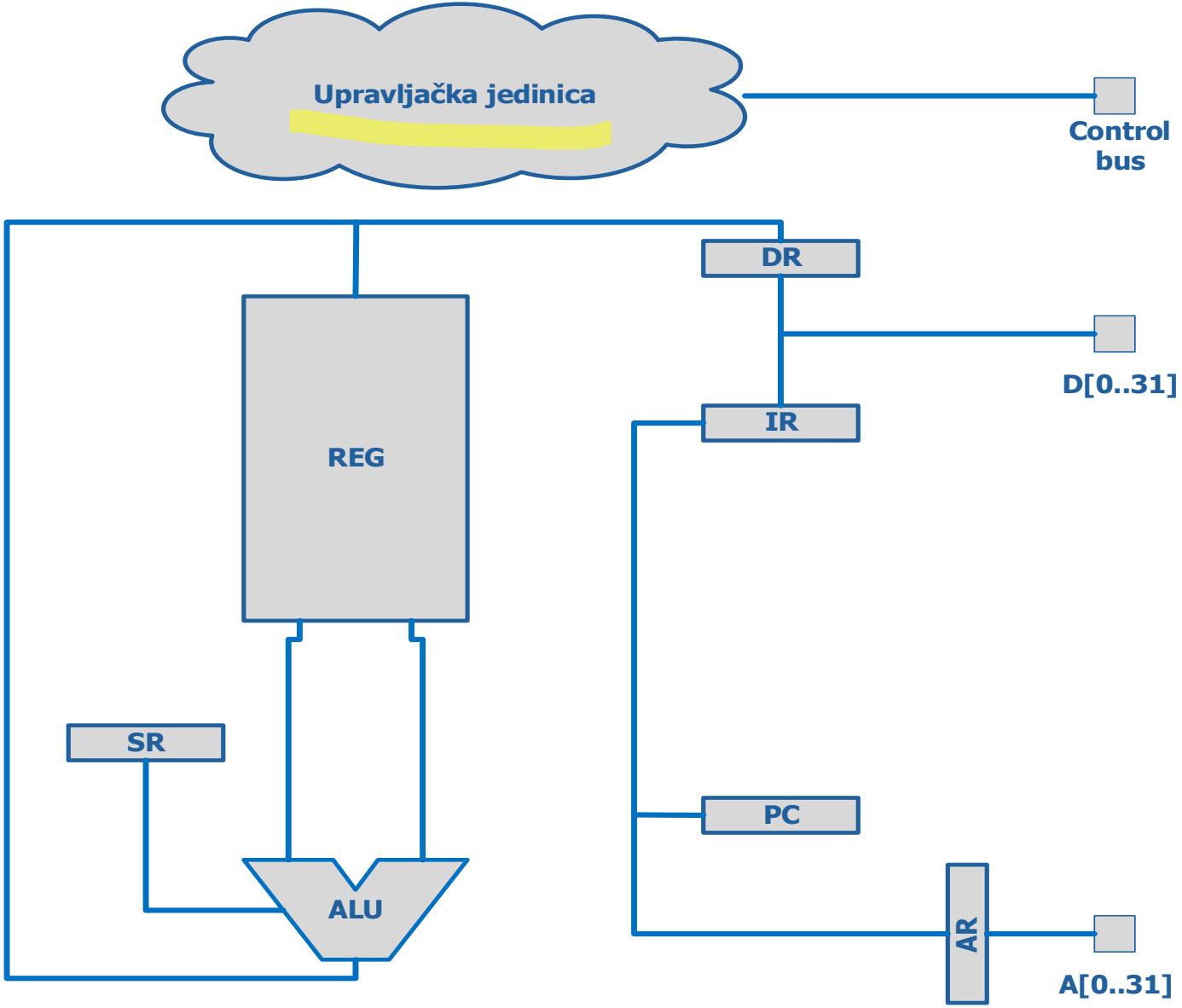
Registar IR u sebi čuva strojni kôd u kojem može biti adresa za naredbu skoka JP. Ta adresa šalje se u PC.



Postoji spojni put između registra PC i AR jer se tim putom šalje adresa sljedeće naredbe koju želimo dohvatiti iz memorije



Registar IR u sebi čuva strojni kôd u kojem može biti adresa za naredbe LOAD/STORE. Ta adresa šalje se u adresni registar AR.



Veze od upravljačke jedinice do ostalih dijelova procesora nisu prikazane jer ih ima previše - svaki dio upravljan je s jednim ili više signala koji dolaze iz upravljačke jedinice

Načelna mikroarhitektura

- Za sada će nam ovo objašnjenje mikroarhitekture biti dovoljno
- Novi važni dijelovi procesora koje smo upravo uveli su:
  - **DR** - podatkovni registar koji služi kao međusklop između unutrašnjosti procesora i podatkovne sabirnice
  - **AR** - adresni registar koji služi kao međusklop između unutrašnjosti procesora i adresne sabirnice
  - **IR** - naredbeni registar koji čuva strojni kôd naredbe za vrijeme njenog dekodiranja
  - Ovi registri su interni, u smislu da programer nema izravnog pristupa do njih

Osnovna inačica procesora

Pregled arhitekture - Rekapitulacija

OSNOVNA INAČICA PROCESORA

- Do sada donesenim odlukama o izboru dijelova arhitekture i do sada opisanim naredbama, definirali smo osnovnu inačicu arhitekture našeg jednostavnog procesora:
  - Von Neumannova arhitektura
  - Load-store arhitektura
  - Osam 32-bitnih registara opće namjene (R0 do R7)
  - 32-bitni registar PC
  - Registar stanja SR (za sada ima definirana 4 bita)
  - Širina podataka unutar procesora je 32 bita
  - Adresna i podatkovna sabirnica su širine 32 bita
  - Memorija s adresiranjem okteta
  - RISC arhitektura
  - Početni skup od osam naredaba

OSNOVNI i POBOLJŠANI FRISC

- U daljnjim predavanjima ćemo definirati poboljšanja i proširenja funkcionalnosti procesora FRISC pri čemu ćemo:
  - proširiti skup naredaba s novim mogućnostima koje omogućuju lakše i efikasnije programiranje
  - posvetiti više pažnje mikroarhitekturi
  - pokazati kako se FRISC povezuje s ostatkom računalnog sustava