

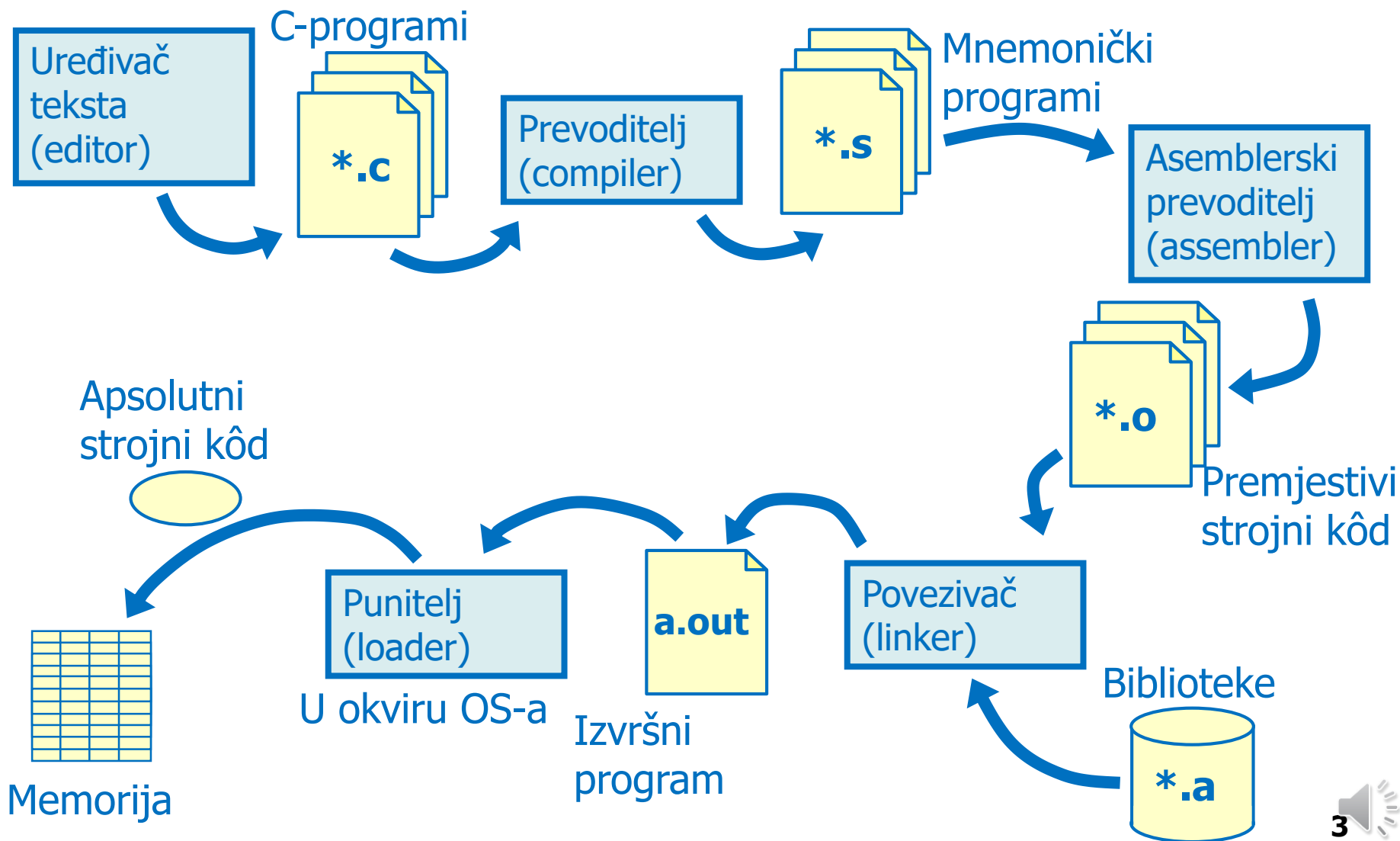
Programiranje procesora FRISC



Asembleri

Asembleri - Uvod

- Tipičan tijek pri prevođenju viših programskih jezika (UNIX):

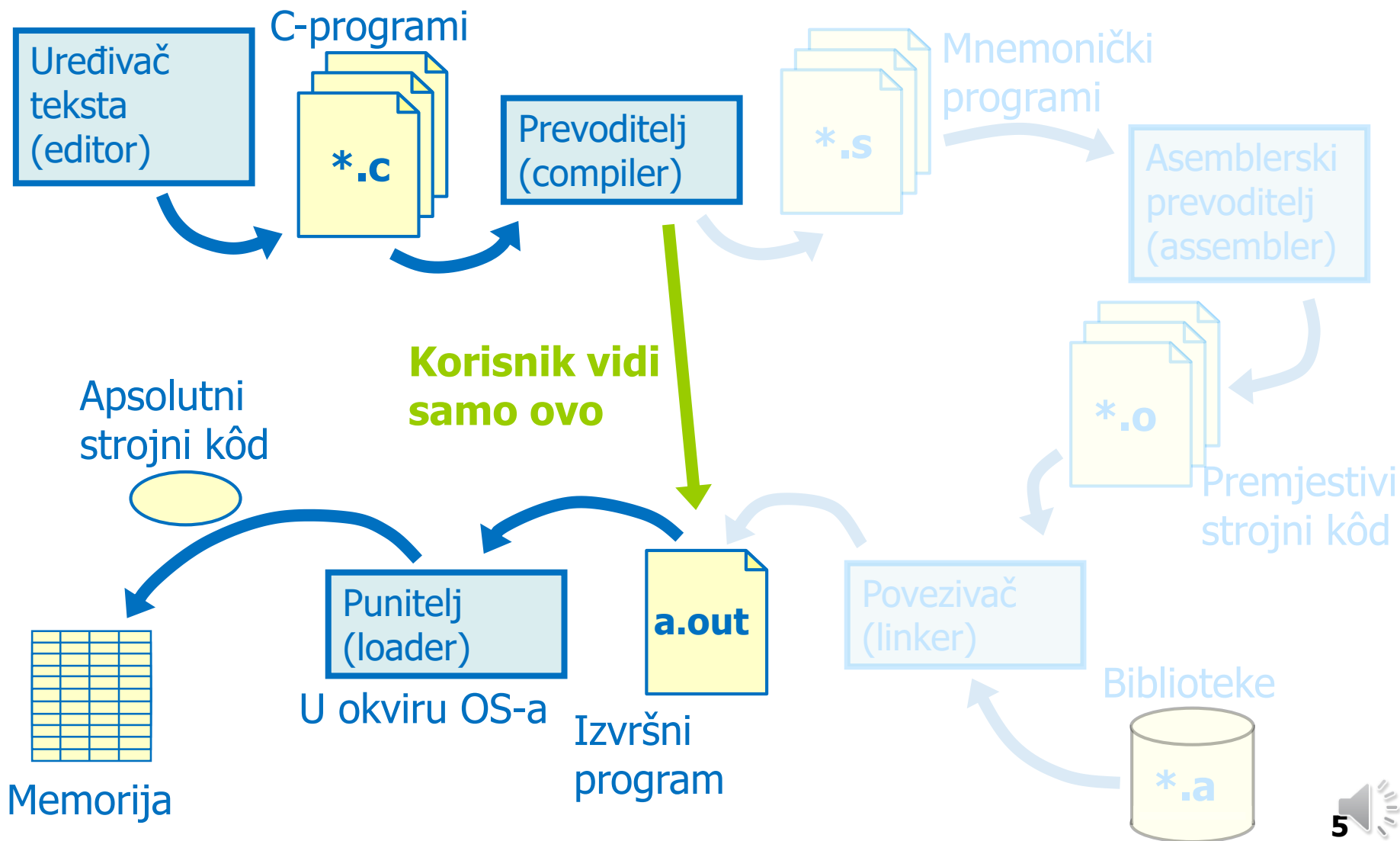


Asembleri - Uvod

- Podjela posla s prethodne slike može biti i drugačija:
 - Povezivanje sa statičkim bibliotekama može obavljati poveziivač, a povezivanje sa dinamičkim bibliotekama može obavljati punitelj
 - Punjenje i povezivanje su zadaće koje može obavljati jedan program.
 - Mnemonički program se stvara samo kao privremena datoteka koja se odmah dalje prevodi asemblerskim prevoditeljem, a ne kao datoteka koja će ostati zapisana na disku (ovo je za korisnika nevidljivo)
 - Izvršni program može biti u apsolutnom ili premjestivom obliku

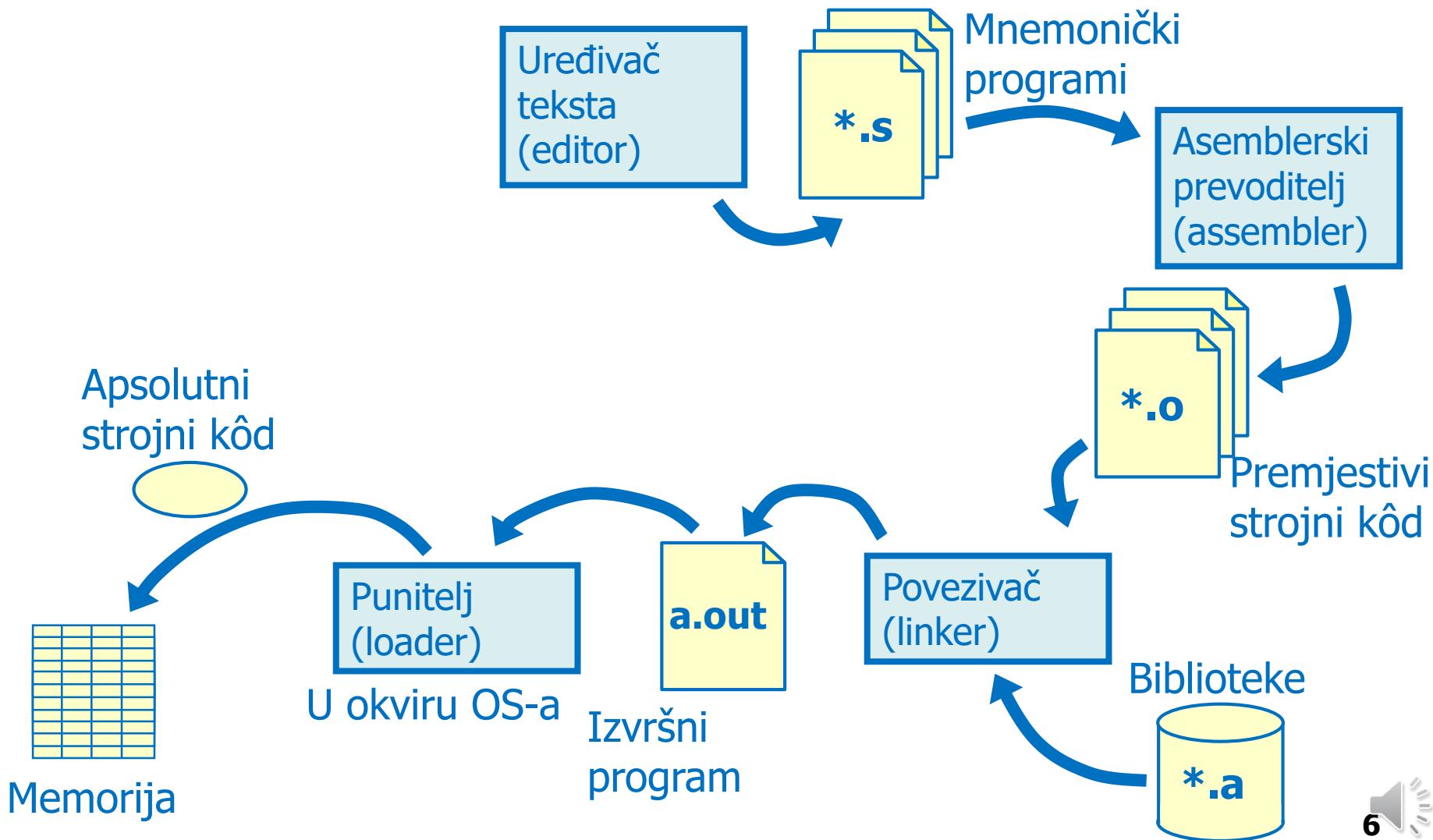
Asembleri - Uvod

- Za korisnika je većina ovog nevidljiva:



Asembleri - Uvod

- Tipičan tijek pri prevođenju mnemoničkih programa:



Asembleri - Uvod

- **Asemblerski prevoditelji**, ili kraće asembleri, su programi koji prevode programe pisane u mnemoničkom jeziku u strojni kôd određenog procesora
 - postupak prevođenja nazivamo asembliranje
 - asembleri su prevoditelji, ali znatno jednostavniji od prevoditelja za više programske jezike
- **Mnemonički jezik** je jezik niske razine i prilagođen je pojedinom procesoru
- Svaki **strojni kôd** ima odgovarajući mnemonik s kojim je u odnosu "jedan na jedan"

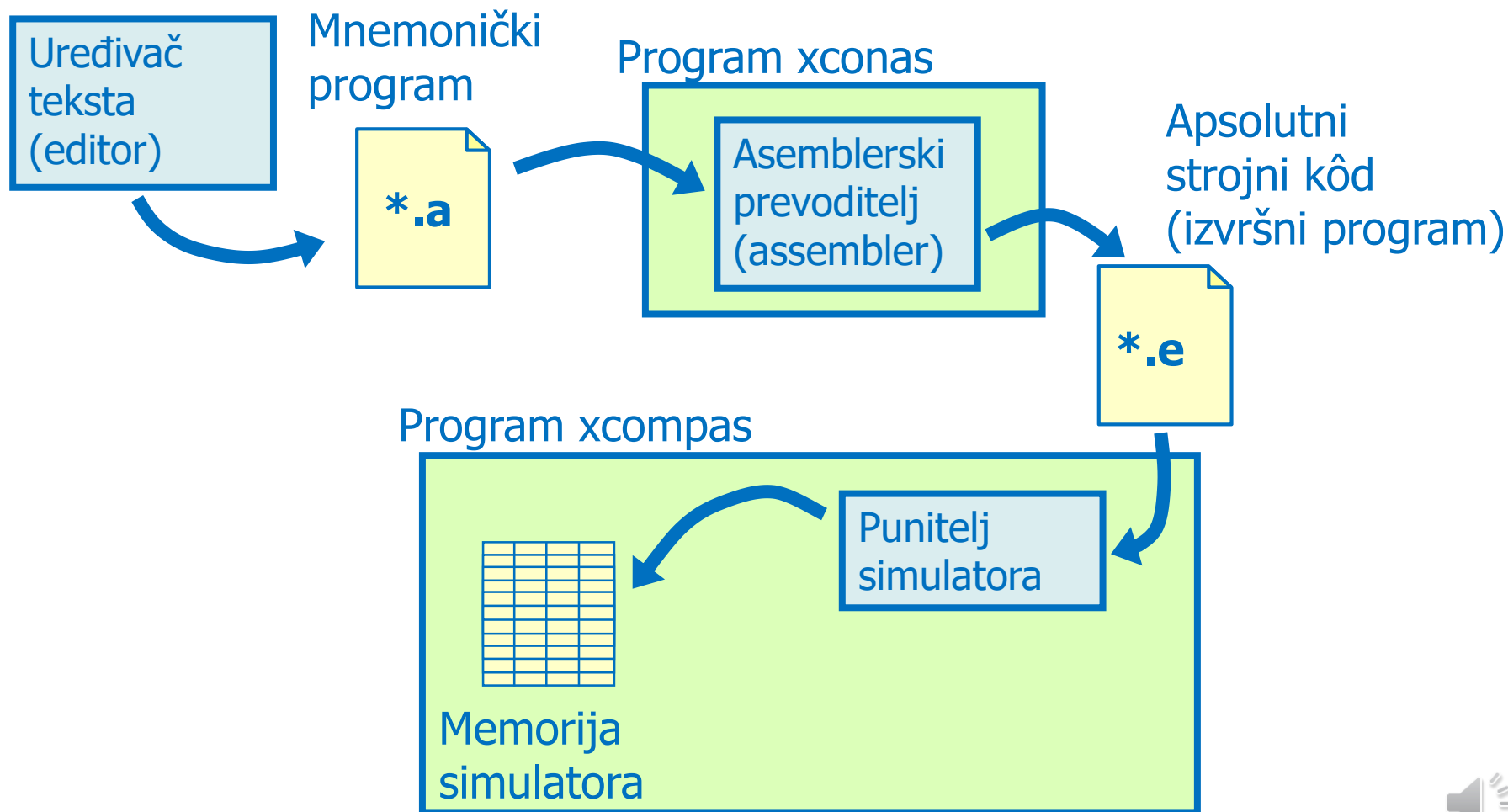


Asembleri - Uvod

- Nakon povezivanja može se dobiti program koji je još uvijek premjestiv ili je u apsolutnom obliku
 - Program u apsolutnom obliku ima određene sve adrese podatka i potprograma i spreman je za izravno punjenje u memoriju računala i izvođenje
 - Za program u premjestivom obliku mora se prilikom punjenja odrediti početna adresa i na temelju toga preračunati sve adrese koje se u programu koriste.
- Nakon punjenja u memoriju računala, program se pokreće
 - Punjenje se tipično odvija pod upravljanjem operacijskog sustava (OS-a)
 - Punjenje se ne zadaje izravno, nego se podrazumijeva kad pokrenemo neki program
 - Korisnik pokreće program pomoću ljuske (npr. tcsh/bash na UNIX-u) ili grafičkog sučelja (Microsoft Windows, X Window na UNIX-u)

Asembleri - Uvod

- Prevođenje mnemoničkih programa u **ATLAS-u**:



Asembleri - Uvod

- U ATLAS-u se može koristiti **samo jedna datoteka** s mnemoničkim programom
- Ona se odmah prevodi **u izvršnu datoteku u apsolutnom obliku**, tj. sadrži strojni kôd koji ima zadanu adresu punjenja u memoriju
- ATLAS je simulator računala na niskoj razini i u njemu ne postoji operacijski sustav - njegove najosnovnije zadaće preuzima korisničko sučelje simulatora u kojem se programi mogu puniti i izvoditi

Asembleri - Mnemonički jezik

- U ovom poglavlju naučit ćemo programirati procesor u **mnemoničkom** ili **asemblerskom jeziku** (assembly language)
 - Mnemonički jezik ovisi o procesoru za kojega je namijenjen, za razliku od viših programskih jezika koji ne ovise o računalu i/ili operacijskom sustavu na kojem će se izvoditi
 - Proizvođač procesora propisuje simbolička imena (mnemonike) za naredbe svog procesora
 - Dok smo objašnjavali arhitekturu i naredbe FRISC-a, već smo vidjeli primjere manjih programa i djelomično pravila pisanja programa u mnemoničkom jeziku

Asembleri - Mnemonički jezik

- Osim propisanih mnemonika, asemblerski prevoditelji dodaju svoja pravila pisanja, ograničenja ili dopunske mogućnosti
- Datoteke u mnemoničkom jeziku su obične tekstovne datoteke pisane prema pravilima pojedinog procesora i asemblerskog prevoditelja*
- Ovdje ćemo koristiti pravila za program CONAS (CONfigurable ASsembler), koji je asemblerski prevoditelj programskog sustava ATLAS

* Napomena: i asemblerski jezik i asemblerski prevoditelj često se nazivaju skraćeno ***assembler***

Asembleri - Pravila pisanja

- Mnemoničke datoteke nemaju slobodan format pisanja kao viši programski jezici, nego su **retkovno orijentirane**:
 - Naredba se **ne može** protezati kroz više redaka
 - U jednom retku može biti **najviše jedna** naredba
 - Smije se pisati prazan redak (zbog bolje čitljivosti)
- Svaki redak sastoji se od sljedećih polja:

POLJE_LABELE	POLJE_NAREDBE	POLJE_KOMENTARA
---------------------	----------------------	------------------------

Asembleri - Pravila pisanja

- Polja imaju sljedeća značenja i pravila pisanja:
- **Polje labele:**
 - Obavezno počinje od **prvog stupca** datoteke, ali se smije ispustiti
 - Labela je simboličko ime za adresu
 - Labela se sastoji od niza slova i znamenaka te znaka podvlake _, a prvi znak mora biti slovo
 - Duljina labele nije ograničena, ali se razlikuje samo prvih deset znakova

Asembleri - Pravila pisanja

- Polja imaju sljedeća značenja i pravila pisanja:
- **Polje naredbe:**
 - Polje naredbe ispred sebe obavezno mora imati **prazninu** (znak razmaka ili tabulatora), bez obzira stoji li ispred labela ili ne
 - Polje naredbe se smije ispustiti (tada naravno nije potrebno stavljati praznine)
 - Naredba se piše prema pravilima definiranim za pojedini procesor
 - U polju naredbe umjesto naredbe smije stajati i pseudonaredba (bit će objašnjene kasnije)

Asembleri - Pravila pisanja

- Polja imaju sljedeća značenja i pravila pisanja:
- **Polje komentara:**
 - Polje komentara počinje znakom komentara i proteže se do kraja tekućeg retka
 - Znak komentara ovisi o procesoru
 - za FRISC i ARM to je znak **točka-zarez ;**
 - Polje komentara se također može ispustiti
 - Polje komentara se zanemaruje prilikom prevođenja

Asembleri - Pravila pisanja

- Primjeri:

POLJE_LABELE	POLJE_NAREDBE	POLJE_KOMENTARA
--------------	---------------	-----------------

PETLJA	ADD R0, R1, R2	;naredba ADD
	SUB R3, R2, R3	
PODATCI	ORG 200	;pseudonaredba ORG
LABELA_3		; labela smije stajati bez naredbe
		; komentar smije početi od prvog stupca
		; ovaj bi red bio prazan da nema komentar :)

Asembleri - Vrste asemblera

- Asembleri se mogu podijeliti po broju prolaza na:
 - jednoprolazne ili apsolutne asemblere
 - dvoprolazne ili simboličke asemblere
 - troprolazne asemblere
 - četveroprolazne asemblere
- } tzv. makro-asembleri
- Ovisno o broju prolaza, asembleri imaju različite mogućnosti
 - što više prolaza, to više mogućnosti


Proučite za domaću zadaću objašnjenje načina prevođenja (pogledajte datoteku nazvanu "3. tjedan - Način asembliranja")

Asembleri - Labele

- U assembleru se **labele** koriste kao **odredište skoka** ili **adrese podataka**
 - Za razliku od viših programskih jezika, u assembleru je korištenje labela i naredbe skoka (npr. JUMP) jedini način za upravljanje tokom programa
 - Kao odredište skoka može se pomoću broja zadati i stvarna adresa skoka, ali tada moramo **točno znati na koju adresu želimo skočiti**, tj. moramo tu adresu "ručno izračunati".


	ADD . . .
NATRAG	XOR . . .
	SUB . . .
	JP NATRAG

Labela



0	ADD . . .
4	XOR . . .
8	SUB . . .
C	JP 4

Stvarna adresa



Asembleri - Labele

- Labele su simbolički nazivi za adrese, a glavne prednosti su:
 - jednostavnije i brže programiranje
 - bolja čitljivost i lakše održavanje programa
 - izračunavanje adresa obavlja asemblerski prevoditelj što ujedno smanjuje mogućnost pogreške
- Asembler "izračunava" stvarne vrijednosti labela (tj. adrese) točno onako kako ih i mi "ručno izračunavamo"
- Korištenje labela naziva se **simboličko adresiranje**, a korištenje stvarnih adresa zadanih brojem naziva se **apsolutno adresiranje**
- **POZOR:** ovo su **asemblerska adresiranja** i ne treba ih miješati s **procesorskim adresiranjima**, naročito ne s istoimenim apsolutnim procesorskim adresiranjem

Asembleri - Pseudonaredbe

- Pseudonaredbe:
 - nemaju veze s procesorom
 - to su naredbe za asemblerski prevoditelj: one upravljaju njegovim radom govoreći mu približe kako treba obavljati prevođenje
 - "izvodi" ih asemblerski prevoditelj tijekom prevođenja
- U okviru ARH1 koristit će se pseudonaredbe ORG, EQU, DW, DH, DB, DS, MACRO i ENDMACRO.

xconas - Pseudonaredba ORG

- Pseudonaredba ORG (origin) zadaje asembleru adresu punjenja strojnog kôda ili podataka, a piše se ovako:

ORG adresa

- Adresa mora biti zadana brojem, a ne labelom
 - Podaci će biti smješteni od zadane adrese
 - Strojni kôdovi dobiveni prevođenjem sljedećih redaka datoteke smjestit će se u memoriji od zadane adrese (ako je djeljiva s 4) ili prve sljedeće adrese koja je djeljiva s 4
- ATLAS-ov asembler daje apsolutni strojni kôd pa se mora znati početna adresa punjenja programa:
 - zadaje se pomoću ORG u prvom retku datoteke
 - ako se ORG ispusti, onda se pretpostavlja početna adresa 0

xconas - Pseudonaredba ORG

- Moguće je u datoteci navesti više pseudonaredba ORG čije adrese:
 - moraju biti u rastućem redoslijedu
 - ne smiju biti manje od adrese zadnjeg prevedenog strojnog kôda

program:

```
ORG 0
ADD ...
STORE ...

ORG 14
LOAD ...
HALT
```

brojevi su heksadekadski

memorija:

adresa	sadržaj
0:	ADD
4:	STORE
8:	0
C:	0
10:	0
14:	LOAD
18:	HALT

zapravo ADD
zauzima adrese 0-3,
STORE 4-7, itd.

} "preskočeno"
do adrese 14

xconas - Pseudonaredba ORG

- Primjer:

program:

```
ORG 20
ADD ...
STORE ...
LOAD ...
```

```
ORG 0
XOR ...
HALT
```

memorija:

adresa	sadržaj
20:	ADD
24:	STORE
28:	LOAD
XXXXXXXXXXXXXXXXXXXX	

greška: 0 je manje od adrese prethodnog ORG-a (iako bi na adresama 0 i 4 bilo mjesta za strojni kod naredaba XOR i HALT)

xconas - Pseudonaredba ORG

- Primjer:

program:

```
ORG 0
ADD ...
STORE ...
LOAD ...
```

```
ORG 4
XOR ...
HALT
```

memorija:

adresa	sadržaj
0:	ADD
4:	STORE
8:	LOAD
XXXXXXXXXXXXXXXXXXXX	

greška: 4 je manje od adrese prethodne naredbe LOAD (veći je od prethodnog ORG-a, ali ne "dovoljno")

VAŽNO: Poravnanje naredaba za FRISC

- **Poravnanje naredaba za FRISC**

- Memorijske lokacije široke su jedan bajt (tj. najmanja količina memorije koja se može adresirati je jedan bajt)
- Podatkovna sabirnica je širine 32 bita, što znači da se može odjednom pročitati sadržaj 4 memorijske lokacije

– Naredbe su široke 32 bita pa su u memoriji uvijek **spremljene na adresama djeljivima s 4**

- Kažemo da su naredbe poravnate na adresu dijeljivu s 4 (memory aligned).

VAŽNO: Poravnanje naredaba i ORG

- Čak ako sa ORG zadamo adresu koja nije djeljiva s 4, prevoditelj će automatski poravnati naredbe:

program:

```
ORG 0
ADD ...
STORE ...
...

ORG 25
LOAD ...
HALT
```

memorija:

adresa	sadržaj
0-3:	ADD
4-7:	STORE
...	...
25:	0
26:	0
27:	0
28-2B:	LOAD
2C-2F:	HALT



asemblerški prevoditelj automatski
"poravnava" naredbe na adresu djeljivu s 4

xconas - Pseudonaredba EQU

- Pseudonaredba EQU (equal) služi za "ručno" definiranje vrijednosti labele (kao da definiramo imenovanu konstantu):

LABELA **EQU** **podatak**

- Labela i podatak se obavezno pišu, pri čemu podatak mora biti zadan numerički, a ne nekom drugom labelom
- Inače, kad nema pseudonaredbe EQU, asemblerski prevoditelj samostalno određuje vrijednost labele na temelju trenutačne adrese (spremljene u lokacijskom brojilu) i ubacuje labelu u tablicu labela
- Pri nailasku na pseudonaredbu EQU, prevoditelj će zanemariti vrijednost lokacijskog brojila. Umjesto toga jednostavno će uzeti labelu i podatak i staviti ih zajedno u tablicu labela

xconas - Pseudonaredba DW

- Pseudonaredba DW (define word) služi za izravan upis riječi (4 bajta) u memoriju (bez prevođenja):

DW podatci

- Podatci moraju biti zadani numerički, a ne labelom
- Ispred pseudonaredbe DW može stajati labela
- Prevoditelj jednostavno uzima podatke i stavlja ih od sljedeće memorijske riječi na dalje, čime se zauzima i inicijalizira memorija

xconas - Pseudonaredba DH

- Pseudonaredba DH (define half-word) služi za izravan upis poluriječi (2 bajta) u memoriju (bez prevođenja):

DH podatci

- Podatci moraju biti zadani numerički, a ne labelom
- Ispred pseudonaredbe DH može stajati labela
- Prevoditelj jednostavno uzima podatke i stavlja ih od sljedeće memorijske riječi na dalje, čime se zauzima i inicijalizira memorija

xconas - Pseudonaredba DB

- Pseudonaredba DB (define byte) služi za izravan upis bajta u memoriju (bez prevođenja):

DB podatci

- Podatci moraju biti zadani numerički, a ne labelom
- Ispred pseudonaredbe DB može stajati labela
- Prevoditelj jednostavno uzima podatke i stavlja ih od sljedeće memorijske riječi na dalje, čime se zauzima i inicijalizira memorija

xconas - Pseudonaredba DS

- Pseudonaredba DS (define space) služi za zauzimanje većeg broja memorijskih lokacija (bajtova) i njihovu inicijalizaciju u nulu:

DS podatak

- Podatak se obavezno piše te mora biti zadan numerički, a ne labelom
- Podatak zadaje koliko memorijskih lokacija treba zauzeti
- Ispred pseudonaredbe DH može stajati labela
 - Labela će biti adresa prve lokacije u nizu koji je zauzela pseudonaredba DS

xconas - Pisanje brojeva

- Brojevi se pišu u podrazumijevanoj bazi koja je heksadekadska
- To se odnosi na sve brojeve koji se pišu u pseudonaredbama i naredbama bez obzira predstavljaju li adresu, podatak, broj podataka ili bilo što drugo
- za brojeve u drugim bazama, svaki se broj pojedinačno može napisati u željenoj bazi ako se napiše sa jednim od prefiksa %B za binarnu bazu, %D za dekadsku i %H za heksadekadsku. Na primjer: **%B 101010110** ili **%D 1239** ili **%H 12AB04**
- Kako bi assembler razlikovao brojeve od labela, brojevi će uvijek počinjati znamenkom, a labela slovom: svi heksadekadski brojevi koji počinju slovom na početku imaju dodanu nulu (npr. 0A38C)

Primjeri programa

Uspoređivanje brojeva

- FRISC ima sve potrebne uvjete u upravljačkim naredbama za jednostavno uspoređivanje brojeva (u formatima NBC i 2'k), tako da ne treba "ručno" ispitivati pojedine zastavice
- Usporedbe se (najčešće) obavljaju na sljedeći način:
 - Prvo se dva broja oduzmu naredbom CMP (ili SUB ako je potrebna i njihova razlika)
 - Naredbom uvjetnog skoka usporede se brojevi: ako je uvjet istinit, onda se skok izvodi, a inače se nastavlja s izvođenjem sljedeće naredbe
 - U naredbi skoka, uvjet se interpretira kao da je operator usporedbe stavljen "između" operandata koji su oduzimani

Uspoređivanje brojeva

- Na primjer, želimo li usporediti je li broj u R5 veći ili jednak od broja u R2 (uz pretpostavku da su to NBC-brojevi):
- Želimo postaviti uvjet $R5 \geq R2$ pa upotrijebimo sufiks UGE:
 - U: unsigned - jer uspoređujemo NBC-brojeve
 - G: greater - jer ispitujemo je li R5 veći od R2
 - E: equal - jer ispitujemo je li R5 jednak R2
- U naredbi CMP pišemo brojeve u istom redoslijedu kao u uvjetu $R5 \geq R2$:

```
CMP      R5, R2
JR_UGE   UVJET_ISTINIT
```

>>>>

Pisanje uvjeta..

- Na primjer: ako je $R5 \geq R2$, onda treba uvećati R7 za 7, a inače ne treba napraviti ništa. Izravnim pisanjem uvjeta dobivamo:

```
                CMP      R5, R2
                JR_UGE    UVECAJ_R7
NISTA           JR       DALJE
UVECAJ_R7       ADD      R7, 7, R7
DALJE           ...
```

- S obrnutim uvjetom program je nešto razumljiviji, kraći i brži:

```
                CMP      R5, R2
                JR_ULT    DALJE
UVECAJ_R7       ADD      R7, 7, R7
DALJE           ...
```

Uspoređivanje brojeva

Usporediti dva 2^k-broja spremljena u registrima R0 i R1. Manji od njih treba staviti u R2. Drugim riječima treba napraviti:

$$R2 = \min (R0, R1)$$

Rješenje:

	CMP	R0 , R1	
	JR_SLT	R0_MANJI	
R0_VECI_JEDNAK	MOVE	R1 , R2	; R1 je manji
	JR	KRAJ	
R0_MANJI	MOVE	R0 , R2	; R0 je manji
KRAJ	HALT		

Uspoređivanje brojeva

Ispitati 2^k-broj u registru R6. Ako je negativan, u R0 treba staviti broj -1. Ako je jednak nuli, u R0 treba upisati 0. Ako je pozitivan, treba u R0 upisati 1. Drugim riječima treba napraviti:

$$R0 = \text{signum} (R6)$$

Rješenje:

```
OR      R6,R6,R0 ; broj ispitaj i stavi u R0
JR_Z    KRAJ      ; ako je 0, u R0 je rezultat
JR_N    NEG        ; ispitaj predznak
POZ     MOVE      1, R0
        JR        KRAJ
NEG     MOVE      -1, R0

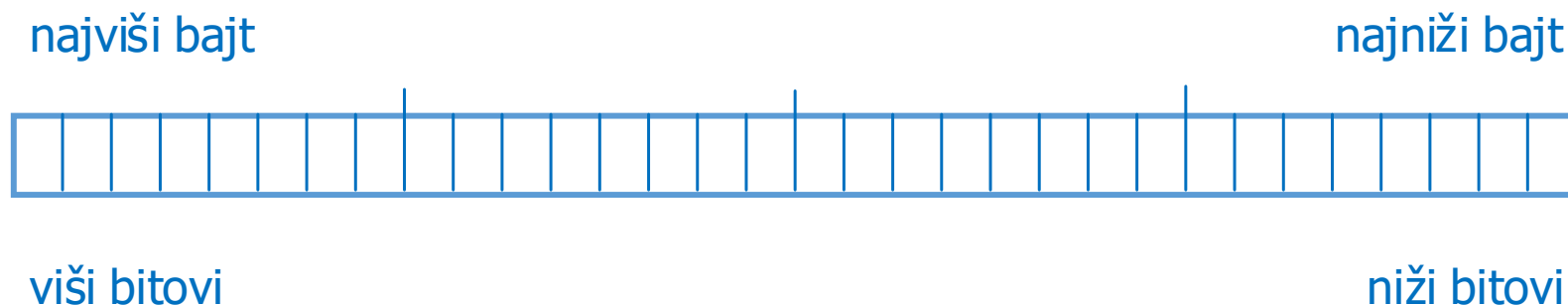
KRAJ    HALT
```

Rad s bitovima

- Čest zadatak u asemblerskom programiranju
- Potrebno je mijenjati ili ispitivati bitove u registru ili memorijskoj lokaciji
 - Rad s bitovima u registru jednostavno se ostvaruje kombinacijama aritmetičko-logičkih naredaba
 - Rad s bitovima u memorijskoj lokaciji nije moguć pa se zato podatak prvo prebaci u jedan od registara, radi se s bitovima te se podatak vrati u memorijsku lokaciju
- Osnovne operacije s bitovima:
 - postavljanje (set)
 - brisanje (reset)
 - komplementiranje (complement)
 - ispitivanje (test)

Rad s bitovima

- Nekoliko pojmova vezanih za bitove u podatku:



- Paritet / Parnost
- Maska

Rad s bitovima - Postavljanje bitova

U registru R0 treba **postaviti** najniža 4 bita, a ostali se ne smiju promijeniti.

```
LOAD    R1, (MASKA)
OR       R0, R1, R0
HALT
```

```
MASKA    DW    %B 1111    ; ostali bitovi su 0
```

Ili jednostavnije (i bolje):

```
OR       R0, %B 1111, R0
HALT
```

Rad s bitovima - Ispitivanje bitova

Treba **ispitati** je li broj u registru R0 paran ili neparan. Ako je paran, treba ga upisati u R2, a inače u R2 treba upisati broj 1.

Rješenje:

Dakle, zadatak se svodi na **ispitivanje stanja jednog bita** što se ostvaruje brisanjem bitova koji se ne ispituju i testiranjem zastavice Z.

>>>>

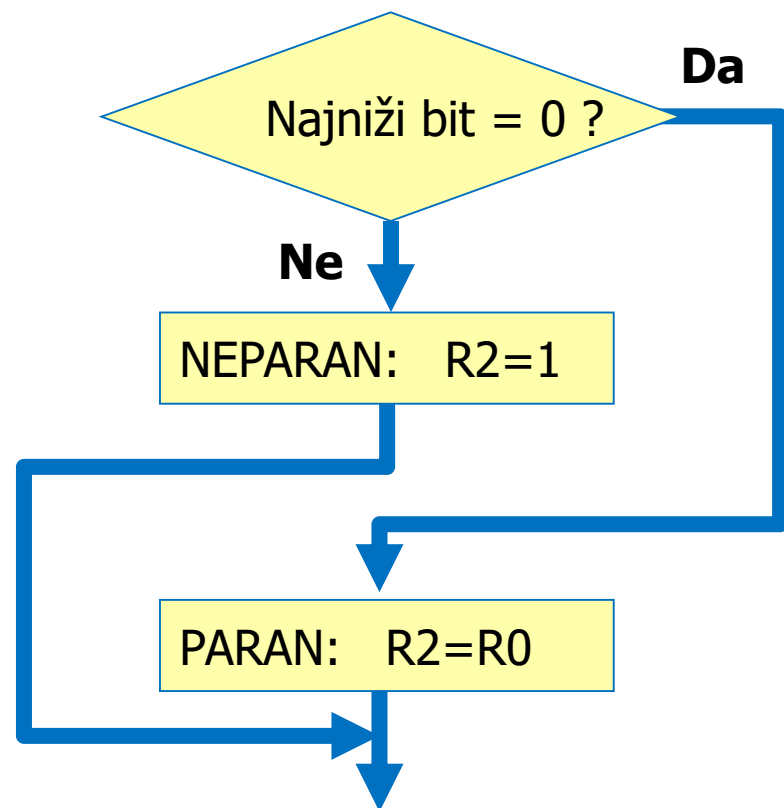
Rad s bitovima - Ispitivanje bitova

```
AND    R0, 1, R1    ; ispitaj najniži bit
JR_Z   PARAN
```

```
NEPARAN MOVE 1, R2
JR        KRAJ
```

```
PARAN   MOVE R0, R2
```

```
KRAJ    HALT
```



Mogući nedostatak: ispitivanje uništava sadržaj R1

Rad s bitovima - Ispitivanje bitova

- **Ispitivanje stanja više bitova**
 - **jesu li svi ispitivani bitovi jednaki nulama?**
- Obrisati sve bitove koje ne ispitujemo (), a bitove koje ispitujemo (?) ostavimo nepromijenjene
- Ispitamo zastavicu Z, tj. ispitamo je li rezultat jednak nuli:
 - Ako rezultat=0, onda su svi ispitivani bitovi u nulama
 - Ako rezultat≠0, onda nisu svi ispitivani bitovi u nulama

Početni broj: ?? ? ??? ?=ispitivani bit

Nakon maskiranja: 00??0000?00???

Rad s bitovima - Ispitivanje bitova

Ispitati jesu li u registru R0 u bitovima 0, 1, 30, 31 **sve nule**. Ako jesu, treba obrisati R0, a inače ga ne treba mijenjati.

```
LOAD    R1, (MASKA)
AND      R0, R1, R1
JR_NZ   IMA_1
```

```
SVE_0    MOVE    0, R0      ; u isp. bitovima su sve 0
```

```
IMA_1    HALT                ; u isp. bitovima ima 1
```

```
MASKA    DW  %B 110000000000000000000000000000000000000011
```

Rad s bitovima - Ispitivanje bitova

- **Ispitivanje stanja više bitova:**
 - **jesu li svi ispitivani bitovi jednaki jedinicama?**
- Postaviti sve bitove koje ne ispitujemo (), a bitove koje ispitujemo (?) ostavimo nepromijenjene
- Komplementirati sve bitove
- Ispitamo zastavicu Z, tj. ispitamo je li rezultat jednak nuli:
 - Ako rezultat=0, onda su svi ispitivani bitovi u jedinicama
 - Ako rezultat≠0, onda nisu svi ispitivani bitovi u jedinicama

Početni broj:

 ?? ? ???

Nakon maskiranja:

11??1111?11???

Nakon komplementa:

00??0000?00???

Rad s bitovima - Ispitivanje bitova

Ispitati jesu li u registru R0 u bitovima od **1 do 4** i bitovima od **12 do 17** te u bitu **30 sve jedinice**. Ako jesu, treba obrisati R0, a inače ga ne treba mijenjati.

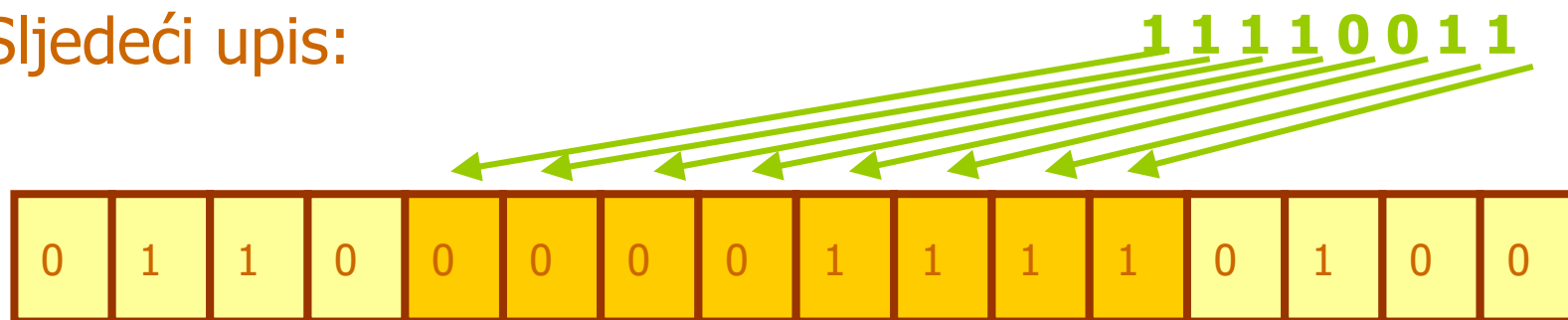
```
LOAD    R1, (MASKA)
OR       R0, R1, R1
XOR      R1, -1, R1    ; komplementiranje
JR_NZ    IMA_0
```

```
SVE_1    MOVE    0, R0    ; u isp. bitovima su sve 1
IMA_0    HALT          ; u isp. bitovima ima 0
```

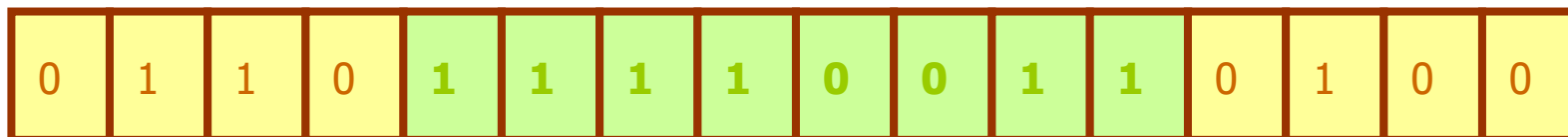
```
MASKA    DW %B 10111111111111110000001111111100001
```

Rad s bitovima – Upis bitova

Sljedeći upis:



Daje rezultat:



Rad s bitovima – Upis bitova

- Postupak upisa je sljedeći:
 - Bitovi podatka koji se žele upisati (**y**) se ne mijenjaju, a ostali se brišu (**_**)
 - Bitovi registra koji se žele mijenjati (**_**) se obrišu, a ostali se ne mijenjaju (**x**)
 - "Poravna" se podatak "iznad" registra
 - Napravi se operacija OR između poravnatog registra i podatka

Podatak:

Maskirani podatak:

Poravnati podatak:

OR:

_____yyyy
0000yyyy
00yyyy00

Registar:

Maskirani registar:

xx_____xx
xx0000xx

xxyyyyxx

Primjer

U bitove **2 do 10** registra R0 treba **upisati** bitove **20 do 28** iz registra R7 (brisanje+OR).

```

; brisanje bitova 2 do 10 u R0
LOAD  R1, (MASKA0)
AND   R0, R1, R0
; brisanje svih bitova osim 20 do 28 u R7
LOAD  R1, (MASKA7)
AND   R7, R1, R7

ROTR  R7, %D 18, R7      ; poravnavanje

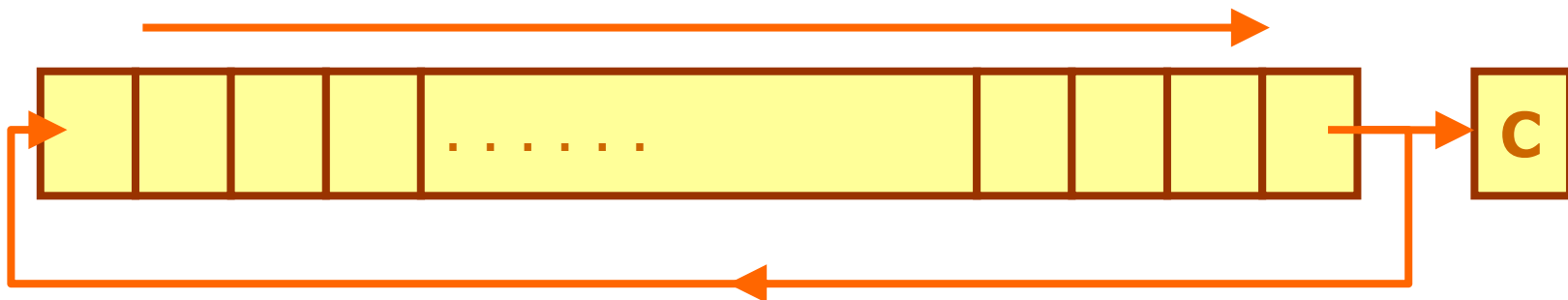
OR    R0, R7, R0        ; upis u R0
HALT

```

[illegible]

Rad s bitovima – Prebrajanje bitova

- Pod prebrajanjem bitova misli se na prebrajanje nula ili jedinica u određenom nizu bitova u podatku
- Najlakše se ostvaruje naredbama rotacije (ulijevo ili udesno) i ispitivanjem zastavice C
- Rotacija radi tako da izlazni bit odlazi u zastavicu C:



Rad s bitovima – Prebrajanje bitova

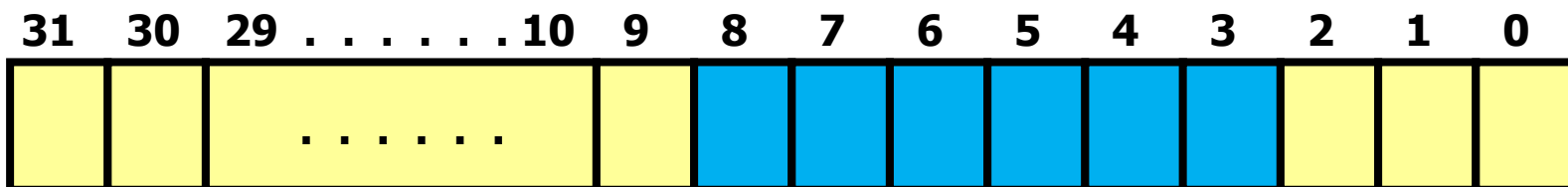
- Ako se rotacija obavlja za više bitova, onda u C odlazi samo izlazni bit od "zadnjeg koraka rotacije"
- Zato treba rotirati registar jedan po jedan bit (u petlji) i ispitivati zastavicu C*
- Prebrajanje bitova koristi se, npr. kod određivanja pariteta podatka

*Drugi način je ispitivanje zastavice N, jer se u njoj nalazi najviši bit registra nakon rotacije (nije toliko uobičajeno rješenje)

Primjer

Koliko nula ima u bitovima **3 do 8** registra R0. Broj nula treba spremiti u memorijsku lokaciju NULE.

```
      MOVE    0, R1          ; R1 = brojač nula
      MOVE    6, R2          ; R2 = brojač za petlju
      ROTR    R0, 3, R0      ; "izbaci" bitove 0 do 2
LOOP  ROTR    R0, 1, R0
      JR_C    JEDAN
      ADD     R1, 1, R1
JEDAN SUB     R2, 1, R2
      JR_NZ   LOOP
      STORE   R1, (NULE)
      HALT
```



Višestruka preciznost

Višestruka preciznost

- Dijelovi računala (memorijske lokacije, registri, ALU, sabirnice) su ograničeni na određen broj bita
- Npr. FRISC ima 32-bitnu arhitekturu (tj. riječ mu ima 32 bita) pa može normalno raditi s podacima te širine
- Ako treba raditi s brojevima većeg opsega ili kakvim drugim podacima širima nego što stanu u riječ procesora ili memorijsku lokaciju, onda koristimo **višestruku preciznost**
- Ovisno koliko procesorskih riječi se koristi za zapis podatka, govorimo o dvostrukoj, trostrukoj, itd. preciznosti

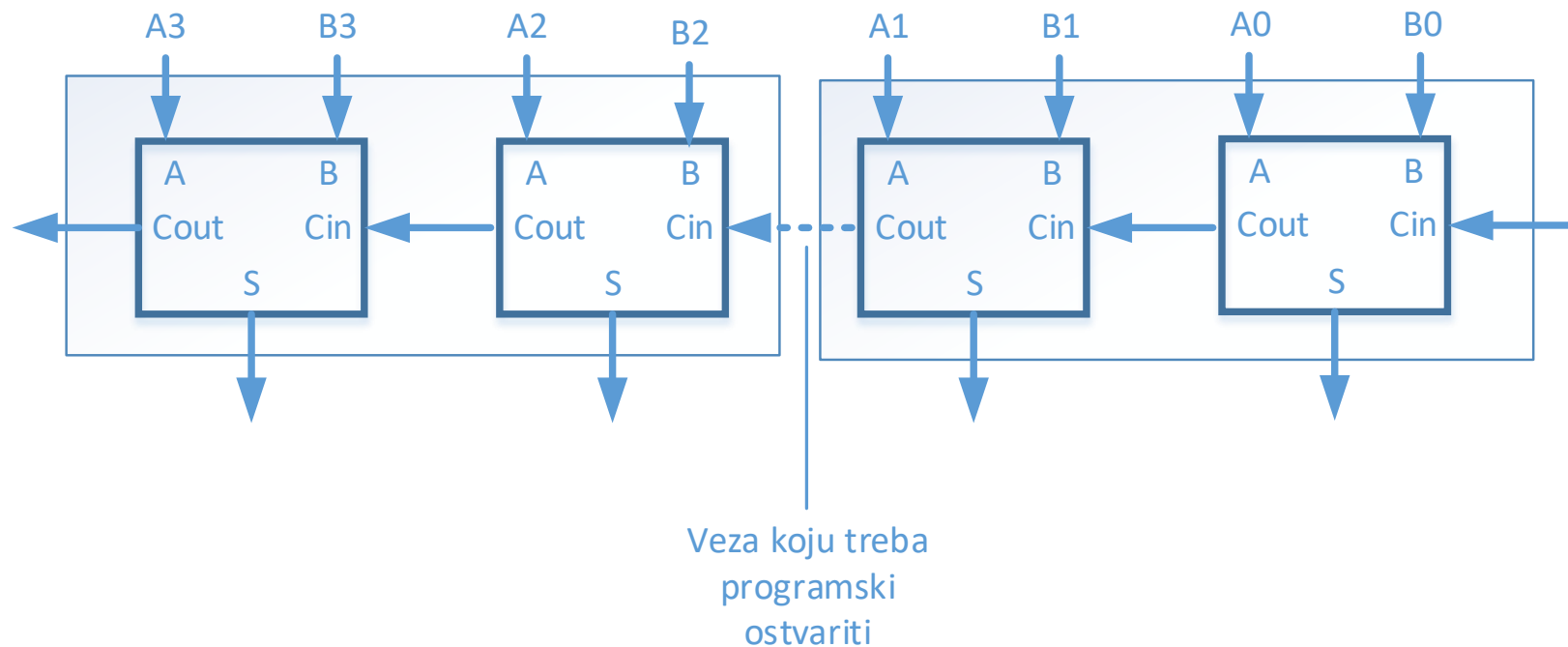
Višestruka preciznost

- **Načelno** se operacije u višestrukoj preciznosti uvijek obavljaju jednako, bez obzira koristimo li dvostruku, trostruku ili neku veću preciznost
 - Zato ćemo pokazati kako se koristi dvostruka preciznost
- Kod zapisivanja podataka u višestrukoj preciznosti u memoriji, treba podatak zapisivati u više uzastopnih lokacija
 - slično zapisivanju 32-bitnih riječi u bajtnoj memoriji, treba voditi računa o rasporedu zapisivanja pojedinih dijelova podatka unutar memorijskih lokacija
 - Budući da FRISC koristi little-endian za zapis 32-bitnih riječi unutar memorije, onda možemo koristiti isti redoslijed i kod višestruke preciznosti (iako to nije nužno)

Višestruka preciznost - Pohrana podataka

- Kod zapisivanja podatka u dvostrukoj preciznosti u registrima, također se moraju koristiti dva registra
- Kod označavanja podataka obično se koriste sufiksi L i H koji označavaju:
 - niži dio podatka (L - low)
 - viši dio podatka (H - high)
- Npr. podatak A u dvostrukoj preciznosti označava se (po dijelovima) oznakama AL i AH

Višestruka preciznost - Zbrajanje



Višestruka preciznost - Zbrajanje

- Kako uračunati međuprijenos? Pomoću **naredbe ADC**.
 - Podsjetnik: naredba ADC, osim dva pribrojnika, pribraja i vrijednost prijenosa iz prethodne operacije zbrajanja
- Budući da je prijenos od zbrajanja spremljen u zastavici C, onda naredba ADC zapravo radi ovako:

$$\text{ADC } X,Y,R \quad \equiv \quad X+Y+\text{prijenos} \rightarrow R \quad \equiv \quad X+Y+C \rightarrow R$$

- Sklopovski se naredba ADC izvodi tako da se na ulaz Cin od najnižeg potpunog zbrajala, dovede stanje iz zastavice C (podsjetnik: kod običnog zbrajanja dovodi se 0)

Višestruka preciznost - Primjer

Zbrojiti NBC ili 2'k brojeve u dvostrukoj preciznosti. Prvi operand smješten je na memorijskim lokacijama AL (niži dio) i AH (viši dio), a drugi na lokacijama BL i BH. Rezultat se sprema na RL i RH.

; ZBROJI NIŽE DIJELOVE

LOAD R0, (AL)

LOAD R1, (BL)

ADD R0, R1, R2

STORE R2, (RL)

; ZBROJI VIŠE DIJELOVE

LOAD R0, (AH)

LOAD R1, (BH)

ADC R0, R1, R2

STORE R2, (RH)

HALT

; OPERANDI

AL DW 0A3541E21

AH DW 942F075F

BL DW 936104A7

BH DW 017F3784

; MJESTO ZA REZULTAT

RL DW 0

RH DW 0

Višestruka preciznost - Logičke operacije

- Logičke operacije AND, OR, XOR, NOT rade neovisno na pojedinim bitovima podataka
- Zato nije bitan redoslijed obavljanja operacija na pojedinim riječima podatka - jedino treba obaviti operacije na svim riječima
- Također, ne postoji nikakvi podatci, kao međuprijenosi, koje bi trebalo prenositi između viših i nižih riječi podataka

Višestruka preciznost - Pomaci i rotacije

- Sklopovski ostvareni pomaci i rotacije prenose bitove između pojedinih riječi pa to također treba napraviti i u programu
- Redoslijed operacije na pojedinim riječima podatka nije bitan, ali ovisno o operaciji može biti praktičniji jedan ili drugi redoslijed. Npr. pomak u lijevo za 1 bit:



- Prvo pomaknemo ulijevo RL. Izlazni bit je u zastavici C. Pomaknemo RH ulijevo i upišemo C u najniži bit od RH.
- Prvo pomaknemo ulijevo RH. Zatim pomaknemo ulijevo RL. Izlazni bit je u zastavici C. Upišemo C u najniži bit od RH.
- Oba redoslijeda izgledaju podjednako komplicirano...

Višestruka preciznost - Pomaci i rotacije

- Prvo pomaknemo ulijevo RL. Izlazni bit je u zastavici C. Pomaknemo RH ulijevo i upišemo C u najniži bit od RH:

```
      SHL    RL, 1, RL
      JR_C    JEDAN
NULA  SHL    RH, 1, RH
      JR      DALJE
JEDAN SHL    RH, 1, RH
      OR     RH, 1, RH
```

- Prvo pomaknemo ulijevo RH. Zatim pomaknemo ulijevo RL. Izlazni bit je u zastavici C. Upišemo C u najniži bit od RH:

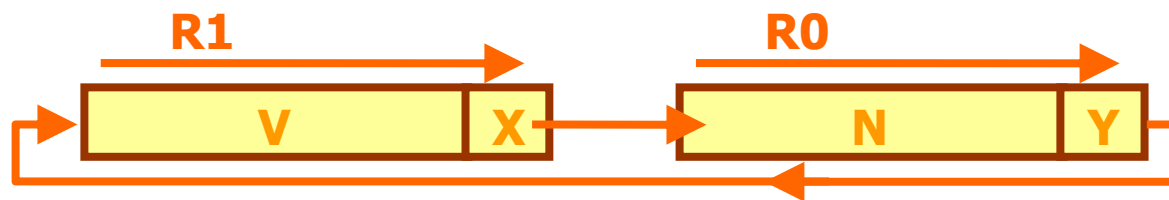
```
      SHL    RH, 1, RH
      SHL    RL, 1, RL
      ADC    RH, 0, RH
```

**Ipak je ovo
jednostavnija varijanta**

Primjer:

Rotirati u desno za 1 mjesto podatak u dvostrukoj preciznosti zapisan u registrima R0 (niža riječ) i R1 (viša riječ).

Početno stanje prije rotacije i način rotacije:



Željeno stanje nakon rotacije u dvostrukoj preciznosti:



Nakon neovisnih rotacija izvedenih na R0 i R1:

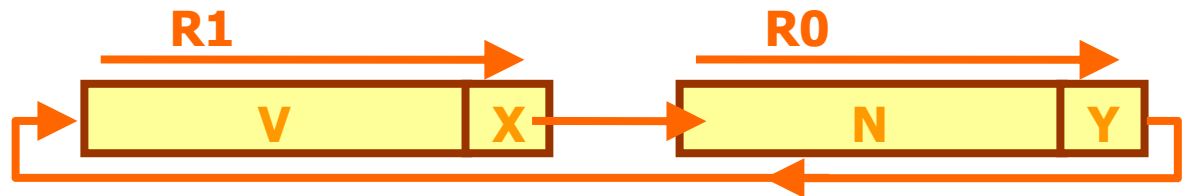


Vidimo da su bitovi V i N na ispravnom mjestu, ali bitovi X i Y moraju zamijeniti mjesta

Rješenje:

1. varijanta: napraviti dvije obične rotacije na R0 i R1, a zatim im zamijeniti vrijednosti najviših bitova X i Y.
2. varijanta: prvo zamijeniti najniže bitove X i Y, pa tek onda napraviti obje rotacije. **Ovo će biti programski lakše.**

Početno stanje prije rotacije i način rotacije:



Nakon zamjene bitova X i Y:



Nakon neovisnih rotacija izvedenih na R0 i R1:



Kako najjednostavnije zamijeniti bitove X i Y? Oni mogu imati sljedeće vrijednosti:

X	Y	operacija
0	0	-
0	1	treba ih zamijeniti
1	0	treba ih zamijeniti
1	1	-

zamjenu bitova koji su
različiti jednostavno
ostvarimo tako da ih
komplementiramo

- Komplementiranje znamo napraviti od prije: XOR s maskom
- Takvu masku dobivamo ako napravimo XOR između R0 i R1 i zatim obrišemo sve bitove osim bita na poziciji X i Y (najniži bit)

```
; stvaranje maske u R3 (za zamjenu X i Y)
; ako su bitovi X i Y jednaki      => maska=0
; ako su bitovi X i Y različiti   => maska=1
```

```
XOR    R0, R1, R3    ; Usporedi najniže bitove
AND     R3, 1, R3     ; u R0 i R1, tj. bitove X i Y.
```

```
; zamjena najnižih bitova R0 i R1 (tj. X i Y)
```

```
XOR    R3, R0, R0
XOR    R3, R1, R1
```

```
; Nezavisna rotacija R0 i R1
```

```
ROTR   R0, 1, R0
ROTR   R1, 1, R1
```

```
HALT
```