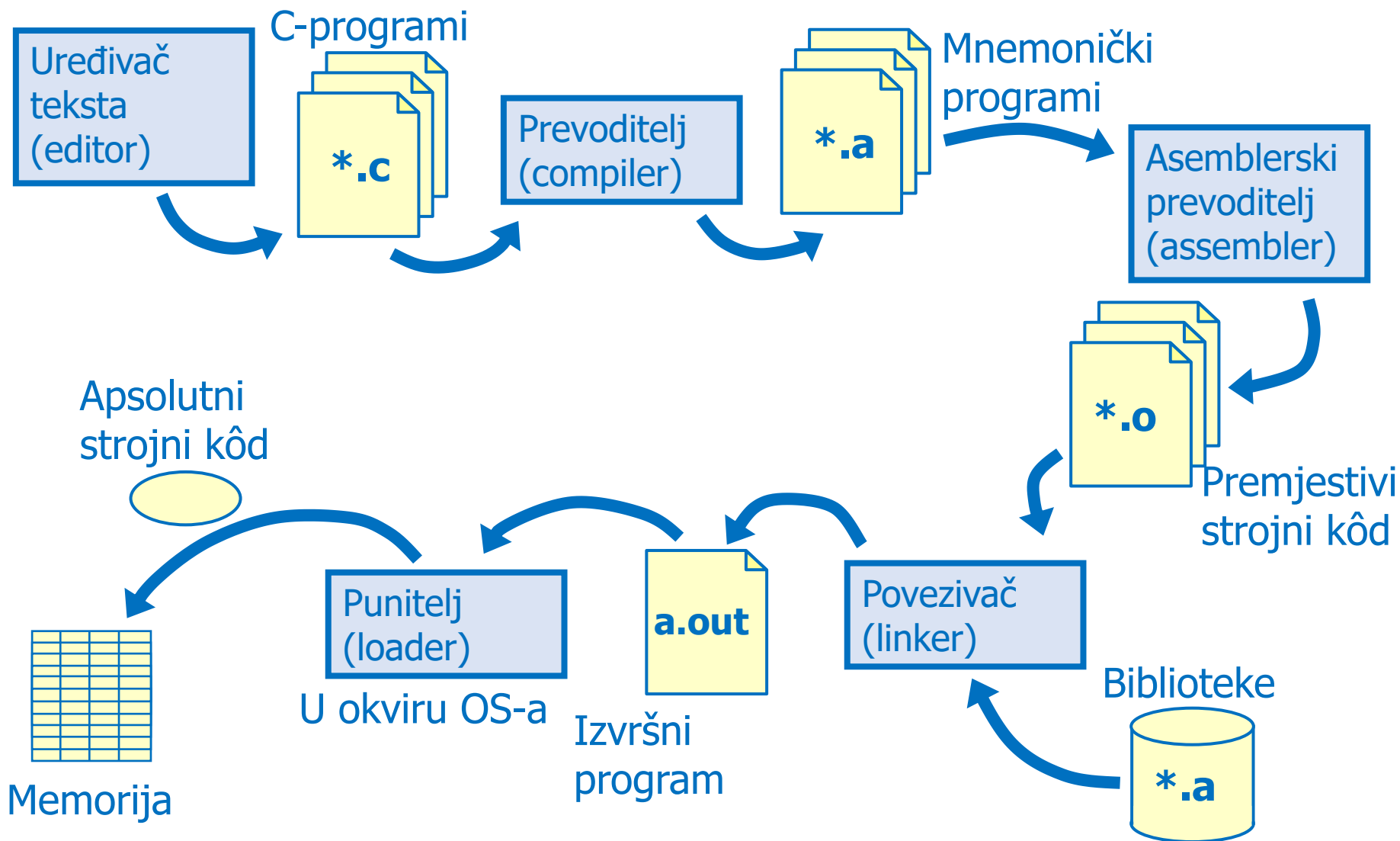


Asembleri

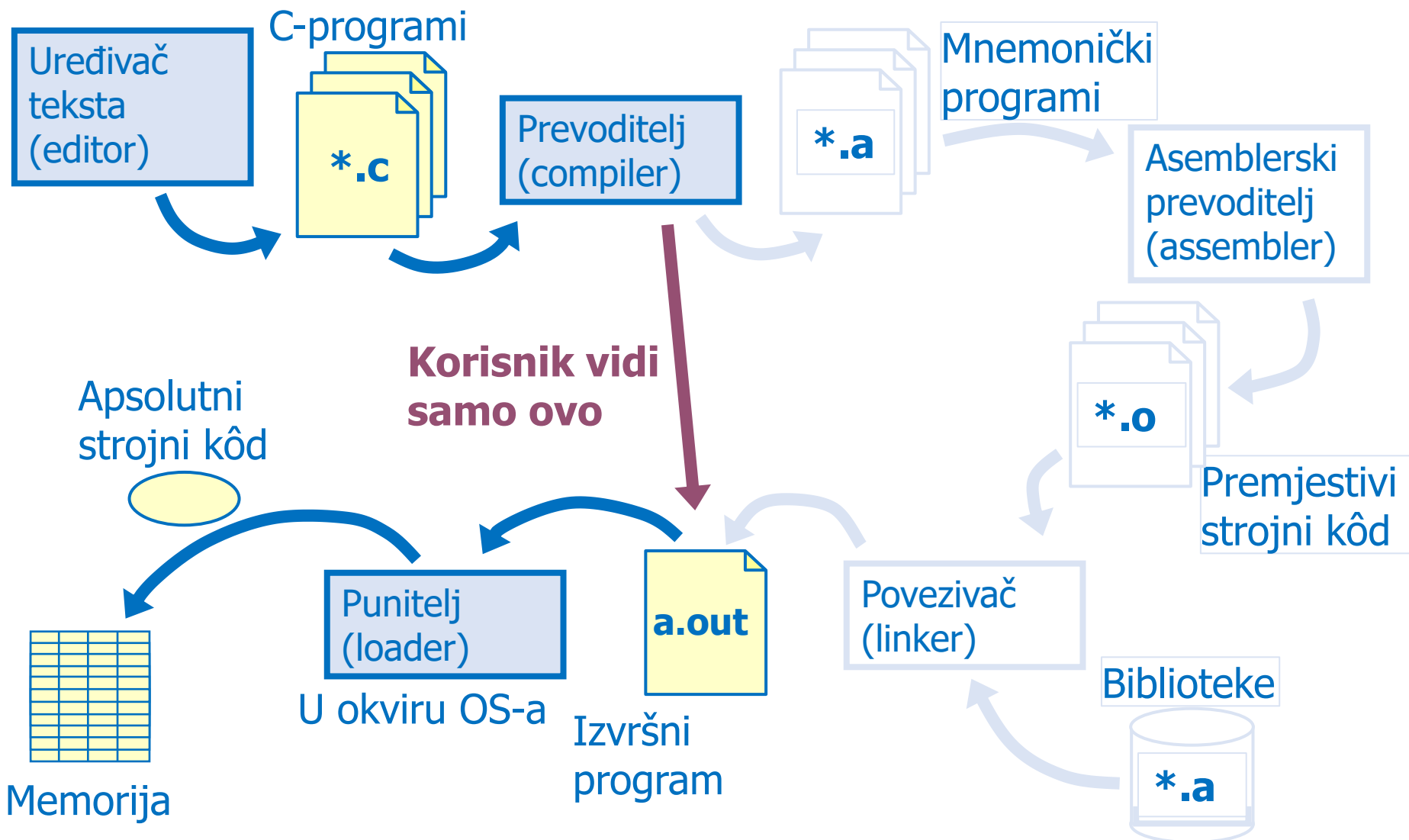
Asembleri - Uvod



- Podjela posla s prethodne slike može biti i drugačija:
 - Povezivanje sa statičkim bibliotekama može obavljati povezač, 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

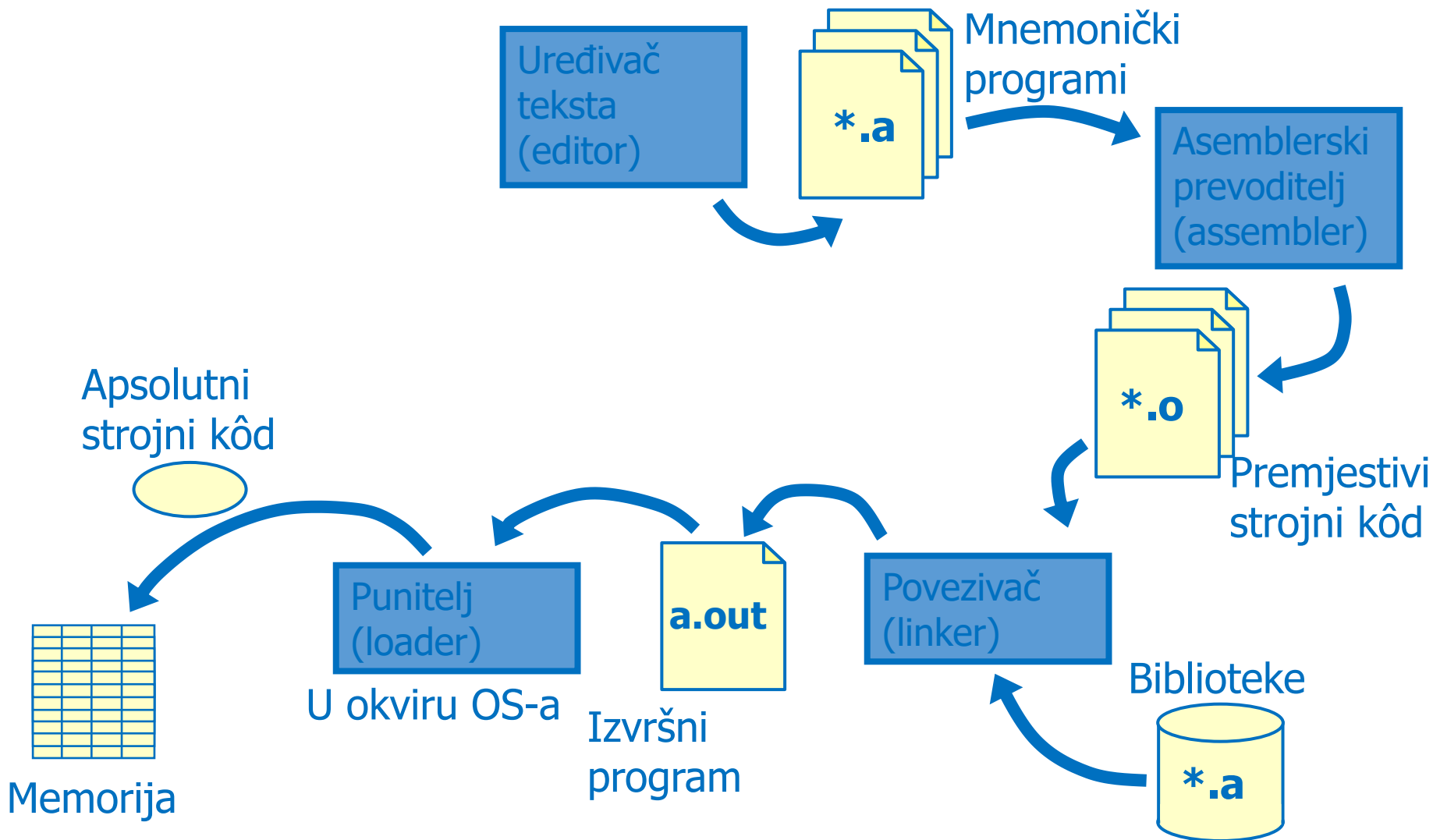
Asembleri - Uvod

Za korisnika je većina ovog nevidljiva



Asembleri - Uvod

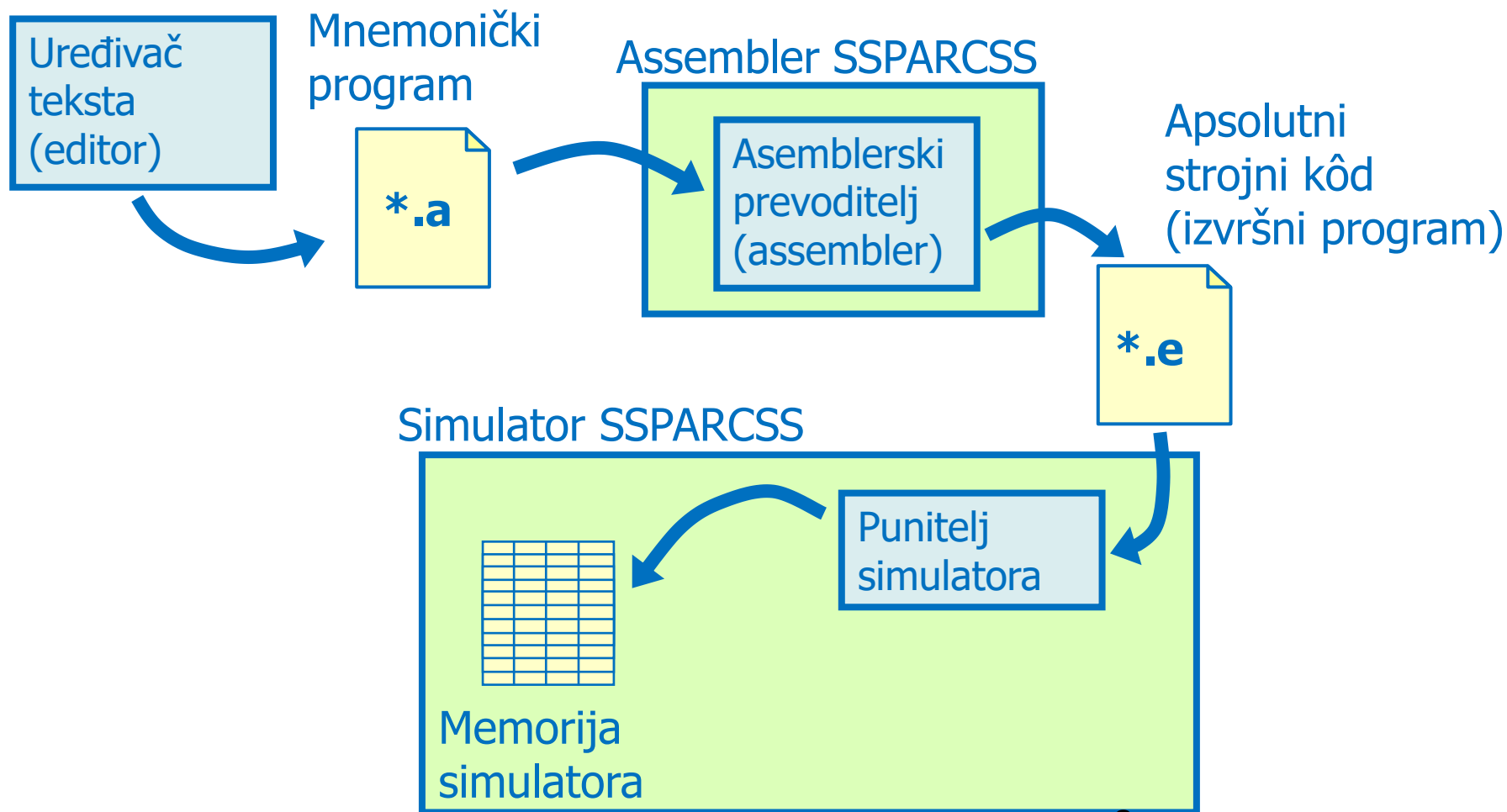
- Tipičan tijek pri prevođenju mnemoničkih programa:



- **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"

- 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 ili grafičkog sučelja

- Prevođenje mnemoničkih programa u **SSPARCSSu**:



- U SSPARCSS-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
- SSPARCSS 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

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*
- U okviru ovog predmeta ćemo koristiti pravila sustava SSPARCSS

* 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
-------	--------	-------	---------	-------	-----------

- 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

- 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)

- Polja imaju sljedeća značenja i pravila pisanja:
- **Polje komentara:**
 - Polje komentara počinje znakom komentara i proteže se do kraja tekućeg retka
 - Znak komentara je **točka-zarez ;**
 - Polje komentara se također može ispustiti
 - Polje komentara se zanemaruje prilikom prevođenja

- 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
- Postoje i četveroprolazni asembleri (omogućuju MACRO)

- 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 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 . . .

0

ADD . . .

NATRAG

LDR . . .

4

LDR . . .

SUB . . .

8

SUB . . .

B NATRAG

C

B 4

Labela



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 **assemblerska adresiranja** i ne treba ih miješati s **procesorskim adresiranjima**, naročito ne s istoimenim apsolutnim procesorskim adresiranjem

- Pseudonaredbe:
 - nemaju veze s procesorom
 - to su naredbe za asemblerski prevoditelj: one upravljaju njegovim radom govoreći mu поближе kako treba obavljati prevođenje
 - "izvodi" ih asemblerski prevoditelj tijekom prevođenja
- U okviru AR1R koristit će se pseudonaredbe
 - **ORG, EQU, DW, DH, DB, DS**

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
- SSPARCSS-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

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 ...
STR ...

ORG 0x14
LDR ...
SUB
```

memorija:

adresa	sadržaj
0:	ADD
4:	STR
8:	0
C:	0
10:	0
14:	LDR
18:	SUB

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

"preskočeno"
do adrese 14

- Pseudonaredba ORG

- Primjer:

program:

```
ORG 20
ADD ...
STR ...
LDR ...

ORG 0
EOR ...
SUB
```

memorija:

adresa	sadržaj
20:	ADD
24:	STR
28:	LDR
XXXXXXXXXXXXXXXXXXXX	

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

- Pseudonaredba ORG

- Primjer:

program:

```
ORG 0
ADD ...
STR ...
LDR ...
```

```
ORG 4
EOR ...
SUB
```

memorija:

adresa	sadržaj
0:	ADD
4:	STR
8:	LDR
XXXXXXXXXXXXXXXXXXXX	

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

VAŽNO: Poravnanje naredaba

- **Poravnanje naredaba**
- 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 s ORG zadamo adresu koja nije djeljiva s 4, prevoditelj će automatski poravnati naredbe:


program:

```
ORG 0
ADD ...
STR ...
...

ORG 0x25
LDR ...
SUB
```

memorija:

adresa	sadržaj
0-3:	ADD
4-7:	STR
...	...
25:	0
26:	0
27:	0
28-2B:	LDR
2C-2F:	SUB



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

- 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 trenutne 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

- 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

- 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

- 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

- 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 DS može stajati labela
 - Labela će biti adresa prve lokacije u nizu koji je zauzela pseudonaredba DS

- Pseudonaredba DSTR

DSTR je pseudonaredba slična kao DB, ali njome se definira string. String će biti upisan u memoriju, svaki ASCII-znak u jedan bajt, i bit će automatski terminiran znakom \0

STRING1 DSTR "fgafasdf" ; upis stringa u memoriju

- Pisanje brojeva

- Brojevi se pišu u podrazumijevanoj bazi koja je **dekadska**
- 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 s jednim od prefiksa
 - 0x za heksadekadsku
 - 0b za binarnu
- Na primjer: **0b101010110** ili **1239** ili **0x12AB04**
- Kako bi assembler razlikovao brojeve od labela, brojevi će uvijek počinjati znamenkom, a labela slovom