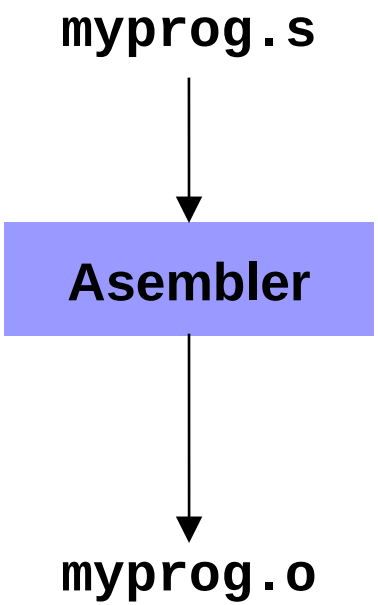
Konstrukcija asemblera

Ulaz i izlaz asemblera

- Asembler (as), prevodi main.s u relokativni objektni fajl main.o:
  as [other arguments] -o myprog.o myprog.s



Čemu služi objektni fajl?

- da se program može pisati u odvojenim celinama – modulima
- da se u programu mogu koristiti biblioteke gotovih, prevedenih modula
- da se program može učitati na lokaciji koja se određuje neposredno pre izvršavanja (zbog postojanja operativnog sistema i drugih programa u izvršavanju).

Šta sadrži objektni program?

- Objektni program sadrži sledeće vrste informacija:
  - *Zaglavlje:* opšte informacije o fajlu, kao što je veličina koda, ime izvornog fajla od koga je nastao prevođenjem, datum kreiranja itd.
  - *Mašinski kod:* Binarne instrukcije i podaci generisani kompajlerom ili asemblerom.
  - *Relokacione informacije:* Lista mesta u objektnom kodu koja treba da se ažuriraju kada linker menja adrese objektnog koda.
  - *Simboli:* Lista globalnih simbola definisanih u posmatranom modulu, simboli koji treba da se uvezu iz drugih modula ili oni koje je definisao linker.
  - *Informacije za debagovanje:* Ostale informacije o objektnom kodu koje nisu potrebne linkeru ali jesu debageru. To uključuje informacije o izvornom fajlu i brojevima linija izvornog koda, lokalne simbole, opise sruktura podataka koje koristi objektni kod, kao što su, na primer, definicije struktura u Cu.

Vrste objektnih fajlova

- Objektni fajlovi javljaju se u tri oblika:
  - **Relokativni objektni fajl**. Sadrži binarni kod i podatke u obliku koji se može kombinovati sa drugim relokativnim objektnim fajlovima u procesu povezivanja da bi se stvorio izvršni objektni fajl (skraćeno izvršni fajl).

    .o na Linuxu, .obj na Windowsu
  - **Izvršni objektni fajl**. Sadrži binarni kod i podatke u obliku koji se može direktno kopirati u memoriju i izvršavati.

    Bez ekstenzije na Linuxu, .exe na Windowsu
  - **Deljeni objektni fajl**.Poseban tip relokativnog objektnog fajla koji se može učitati u memoriju i povezati dinamički, u vreme punjenja ili u vreme izvršavanja.

    .so na Linuxu, .dll na Windowsu
- Kompajleri i asembleri generišu relokativne objektne fajlove (uključujući deljene objekata fajlova). Linkeri generišu izvršne objektne fajlove, kao i deljene.

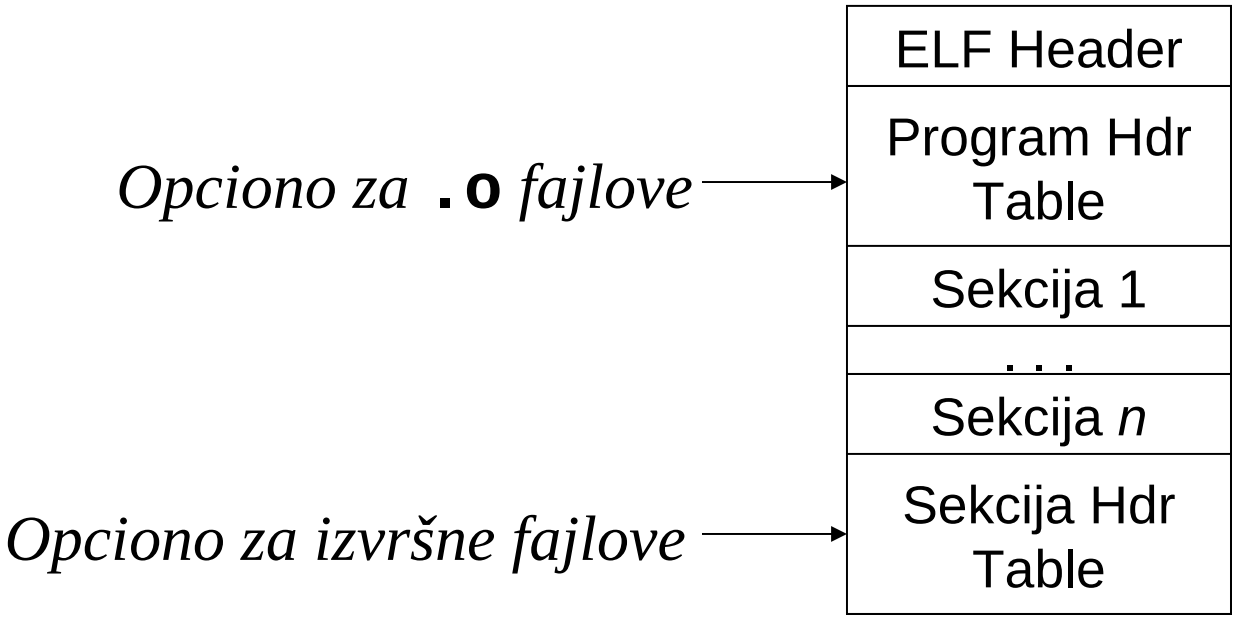
Standardni objektni formati

U današnje vreme, najvažniji standardi za objektne formate su:

- **Portable Executable** (**PE**) (na Microsoft Windows-u). PE je baziran na **Common Object File Formatu** (**COFF**) koji se koristio na Unix-u pre ELF, a još se koristi u nekim embedded sistemima.
- **Executable and Linkable Format** (**ELF**) (na Linux-u i drugim verzijama Unix-a) i
- **Mach-O** (on Mac OS X).

Opšti format ELF fajlova

- ELF je Unix format za objektne i izvršne fajlove
  - Izlaz iz asemblera
  - Ulaz i izlaz linkera

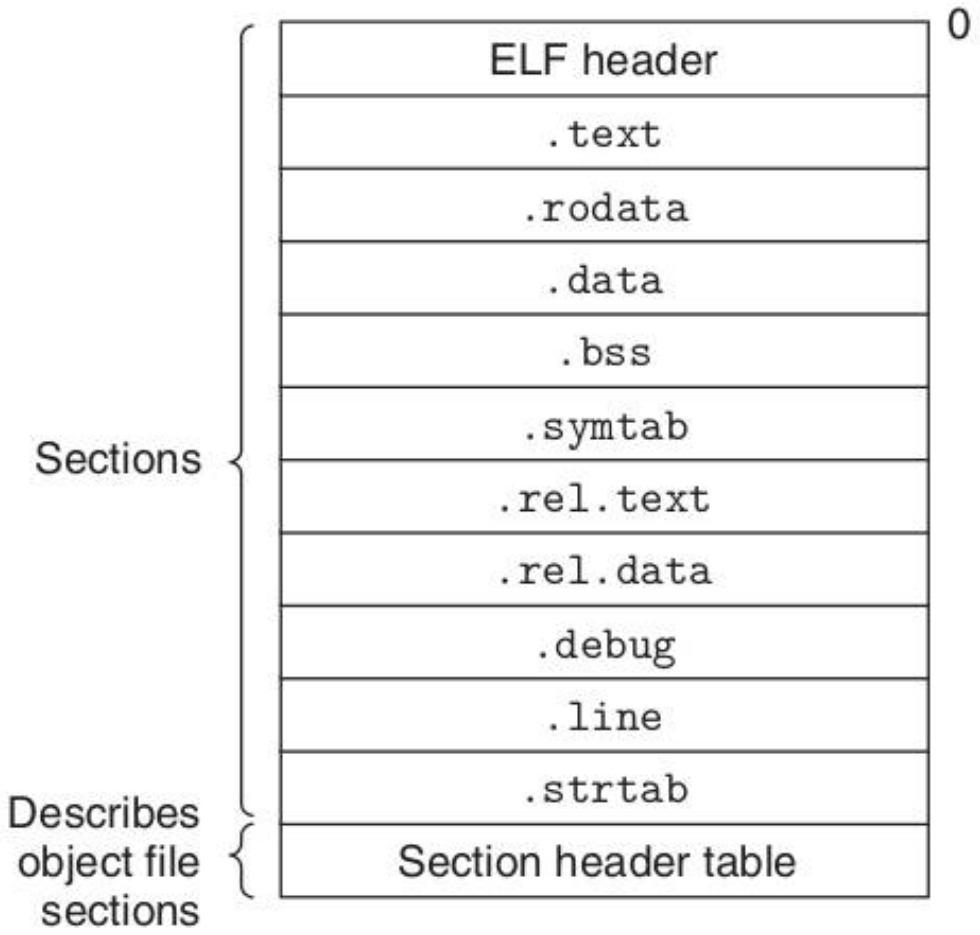


Opšti format ELF fajlova

- **ELF zaglavlje** počinje 16-bajtnim nizom koji opisuje veličinu mem. reči i redosled bajtova u reči sistema koji je generisao fajl. Ostatak ELF zaglavlja sadrži informacije koje omogućavaju linkeru i puniocu da analiziraju i tumače sadržaj fajla.
- Ovo uključuje veličinu ELF zaglavlja, tip objektnog fajla (relokativni, izvršni, odnosno deljeni), tip mašine (npr. IA32), poziciju u fajlu tabele sa zaglavljima sekcija i veličinu zapisa i njihov broj u toj tabeli.
- **Tabela programskih zaglavlja** od značaja je samo za izvršne programe i deljene objekte i sadrži opise segmenata kada se program učita u memoriju za izvršavanje (nema u izlazu asemblera).
- Lokacije i veličine različitih sekcija su opisane u **tabeli zaglavlja sekcija**, koji sadrži zapis za svaku sekciju u objektnom fajlu.
- Između ELF zaglavlja i tabele sa zaglavljima sekcija su smeštene same **sekcije**.
- Sadržaj ELF fajla može se pogledati programima objdump ili readelf

Relokativni objektni fajl po ELF formatu

- Izlaz iz asemblera, ulaz u linker



Relokativni objektni fajl po ELF formatu

- Tipičan ELF relokativni objektni fajl sadrži sledeće sekcije:
- **.text**: Binarni mašinski kod programa.
- **.rodata**: Podaci samo za čitanje, npr. format stringovi u printf
- **.data**: inicijalizovane globalne C promenljive (definisane izvan funkcija). Lokalne C promenljive (unutar funkcija) se pamte na steku u vreme izvršavanja, pa se ne pojavljuju ni u .data ni u .bss sekciji.
- **.bss**: neinicijalizovane globalne C promenljive. Ova sekcija ne zauzima nikakav prostor u objektnom fajlu.

Relokativni objektni fajl po ELF formatu

- **.symtab**: tabela simbola sa informacijama o funkcijama i globalnim promenljivim koje su definisane ili referencirane u programu. Ovaj sadržaj postoji čak i kada se program ne prevodi sa –g opcijom (uključivanje debug informacija u objektni fajl). Jedino se u njoj ne pojavljuju lokalni simboli funkcija.
  - Pažnja: U Cu postoje globalni simboli sa spoljnim (globalnim) povezivanjem (gsim1) i globalni simboli sa internim (lokalnim) povezivanjem (gsim2) i lokalni simboli (lsim).

```
int gsim1;
static float gsim2=0.0;
Int main() {
    char lsim;
}
```

Struktura ulaza .symtab sekcije

code/link/elfstructs.c

```
1   typedef struct {
2       int name;         /* String table offset */
3       int value;        /* Section offset, or VM address */
4       int size;         /* Object size in bytes */
5       char type:4,      /* Data, func, section, or src file name (4 bits)
6            binding:4;   /* Local or global (4 bits) */
7       char reserved;    /* Unused */
8       char section;     /* Section header index, ABS, UNDEF, */
9                         /* Or COMMON  */
10  } Elf_Symbol;
```

code/link/elfstructs.c

Figure 7.4 **ELF symbol table entry.** type and binding are four bits each.

- ABS je za apsolutne simbole kojima ne treba relokacija. UNDEF je za nedefinisane (korišćeni u ovom modulu, definisani u nekom drugom). COMMON je za neinicijalizovane podatke koji još nisu alocirani.

Primer ELF tabele simbola

```
int gsim1;
static float gsim2=0.0;
int main() {
    char lsim;
}
```

```
$ gcc –c lekcija.c –o lekcija.o
$ readelf --symbols  lekcija.o
```

Symbol table '.symtab' contains 11 entries:

Num:	Value	Size	Type	Bind	Vis	Ndx	Name
0:	00000000	0	NOTYPE	LOCAL	DEFAULT	UND	
1:	00000000	0	FILE	LOCAL	DEFAULT	ABS	lekcija.c
2:	00000000	0	SECTION	LOCAL	DEFAULT	1	(.text)
3:	00000000	0	SECTION	LOCAL	DEFAULT	2	(.data)
4:	00000000	0	SECTION	LOCAL	DEFAULT	3	(.bss)
5:	00000000	4	OBJECT	LOCAL	DEFAULT	3	gsim2
6:	00000000	0	SECTION	LOCAL	DEFAULT	5	
7:	00000000	0	SECTION	LOCAL	DEFAULT	6	
8:	00000000	0	SECTION	LOCAL	DEFAULT	4	
9:	00000004	4	OBJECT	GLOBAL	DEFAULT	COM	gsim1
10:	00000000	5	FUNC	GLOBAL	DEFAULT	1	main

Relokativni objektni fajl po ELF formatu

- **.rel.text**: relokacione informacije za .text sekciju. To je spisak lokacija koje će morati da se menjaju kada linker bude spojio ovaj objektni fajl sa drugima. U principu, svaka instrukcija koja poziva eksternu funkciju ili referiše globalnu promenljivu će morati da se menja. Relokacione informacije nisu neophodne u izvršnim fajlovima i obično se izostavljaju ukoliko korisnik izričito ne naloži linkeru da ih uključi.
- **.rel.data**: Relokacione informacije za .data sekciju. U principu, svaki inicijalizovana globalna promenljiva čija je početna vrednost adresa neke globalne promenljive ili eksterno definisane funkcije će morati da se menja.

Relokativni objektni fajl po ELF formatu

- **.debug**: tabela simbola za debagovanje sa stavkama za lokalne promenljive i typedef-ove definisanih programa, globalne promenljive definisane i navedene u programu, kao i ime originalnog C izvornog fajla. Ova sekcija je prisutna samo ako se program prevodi sa -g opcijom.
- **.line**: mapiranje između brojeva linija u izvornom programu i mašinskog koda u .text sekciji. Prisutno samo ako se prevodi sa –g opcijom.
- **.strtab**: tabela stringova za tabele simbola u .symtab i .debug sekcijama i za imena sekcija u zaglavljima sekcija. Radi se o sekvenci stringova terminisanih null .

Glavne funkcije asemblera

- Prevođenje instrukcija iz simboličkog oblika (asemblera) u binarni mašinski kod

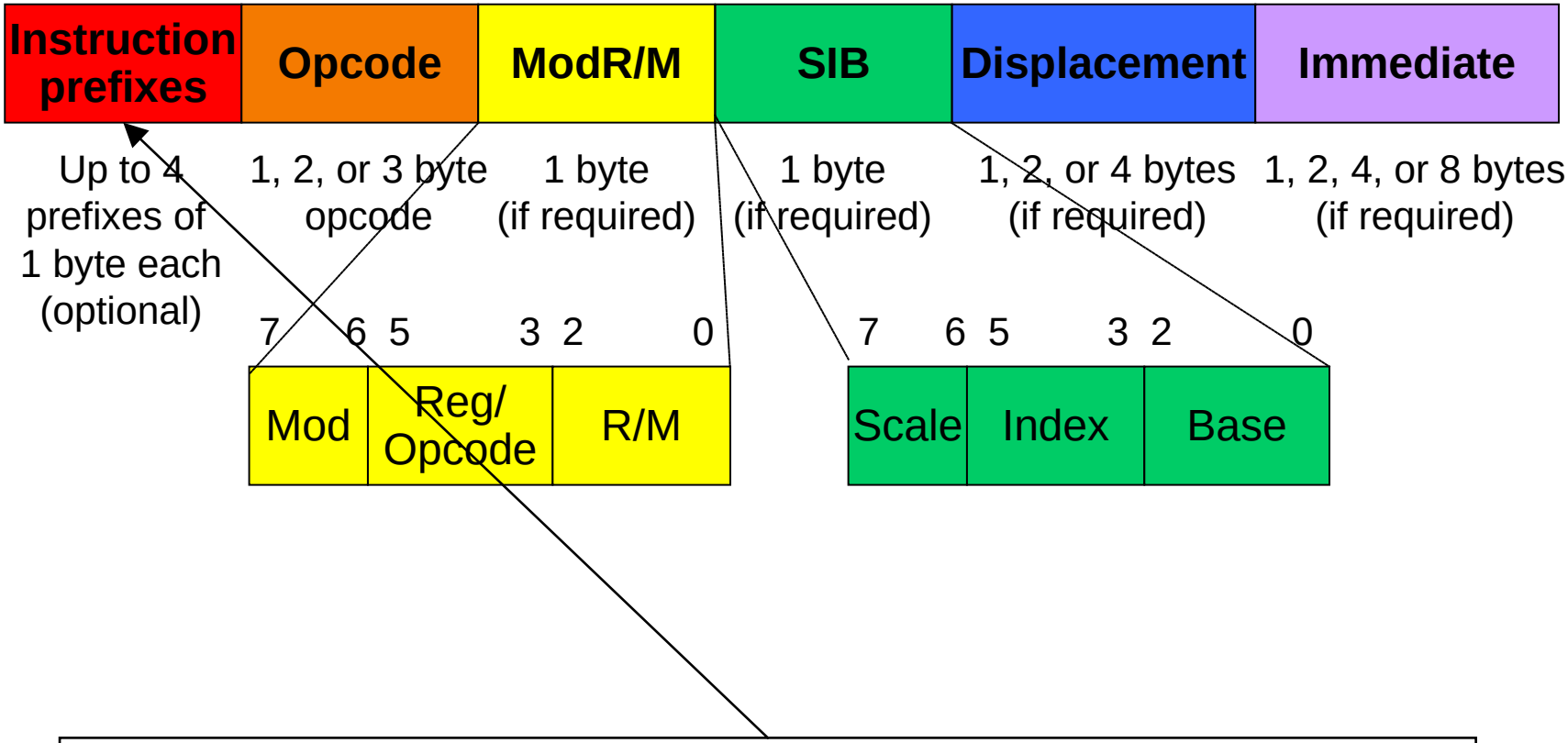
- Obrada referenci u instrukciji na druge instrukcije i podatke:
  - Skokovi na druge lokacije u kodu
  - Pristup globalnim promenljivama preko simboličkih imena dodeljenih njihovim memorijskim lokacijama
  - Pozivanje i povratak iz funkcija definisanim na drugim mestima u kodu

```
main:
        push   rbp
        mov    rbp, rsp
        call    getchar
        cmp    eax, 'A'
        jne     skip
        mov edi, offset msg
        call    printf
skip:
        mov    eax, 0
        mov    rsp, rbp
        pop    rbp
        ret
```

Prevođenje instrukcija

- Primer x86_64 mašinskog jezika
  - Teško je generalizovati format instrukcije
    - Ima puno izuzetaka u odnosu na opšte pravilo kodiranja instrukcija
- Navešćemo nešto što se smatra kao opšte pravilo kodiranja x86_64 instrukcije i ilustrovati sa dva konkretna primera…

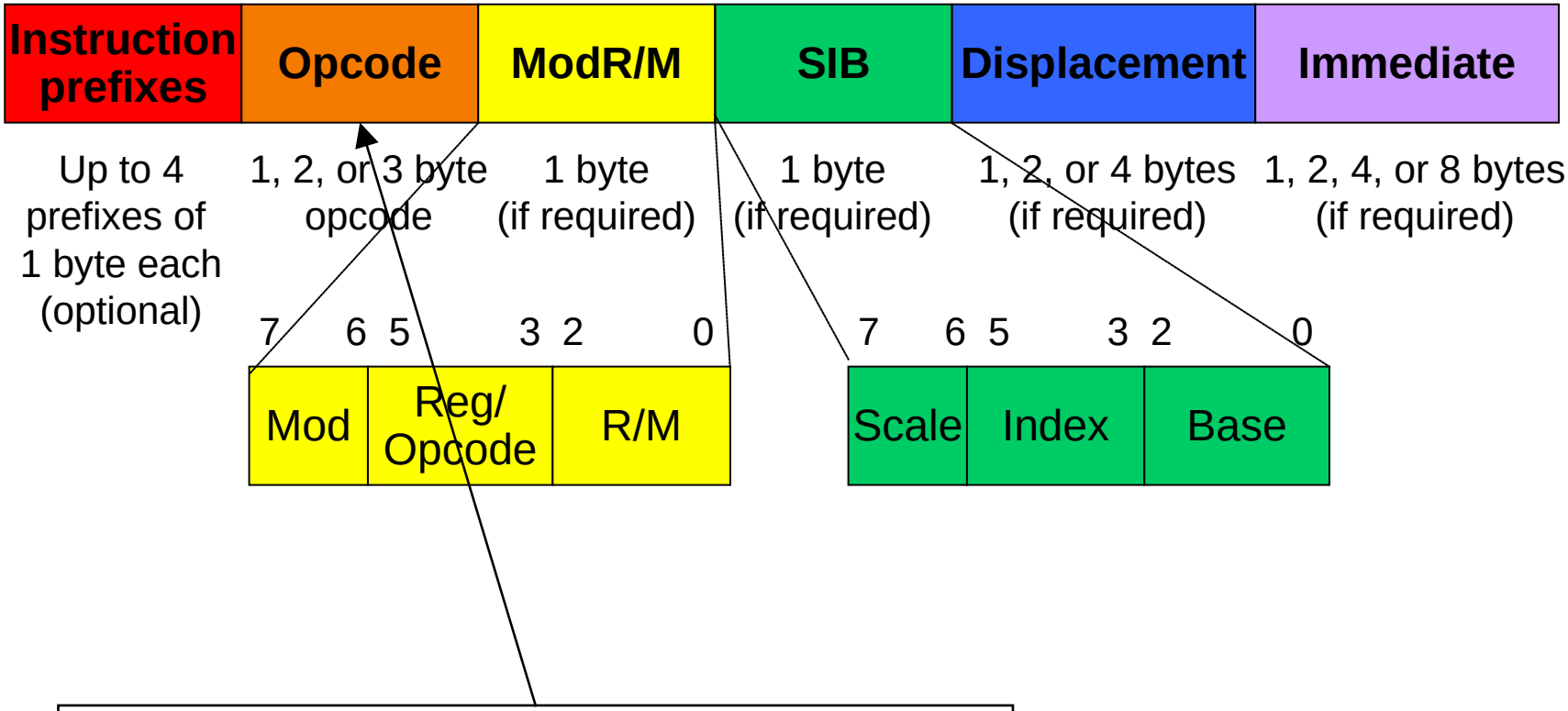
Opšti format x86_64 instrukcije



Opcioni instrukcijski prefiks

- Prefiks ponavljanja string instrukcje, prefiks veličine operanda

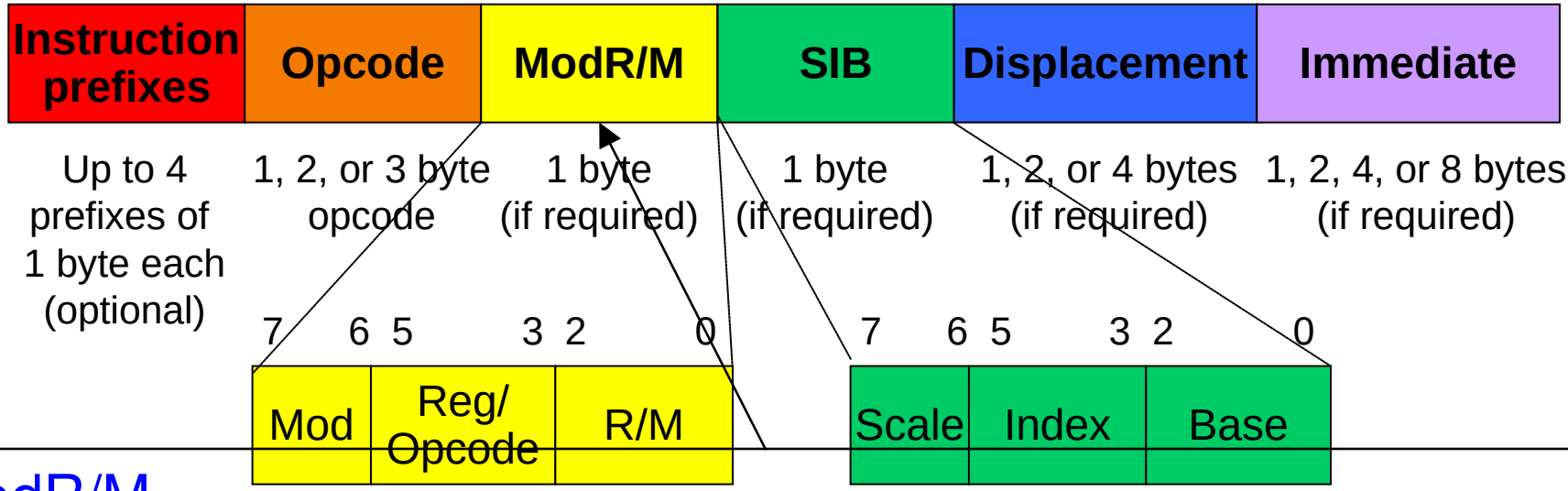
Opšti format x86_64 instrukcije



Operacioni kod

- Određuje koja operacija treb da se izvrši
- Npr. add, move, call, ...

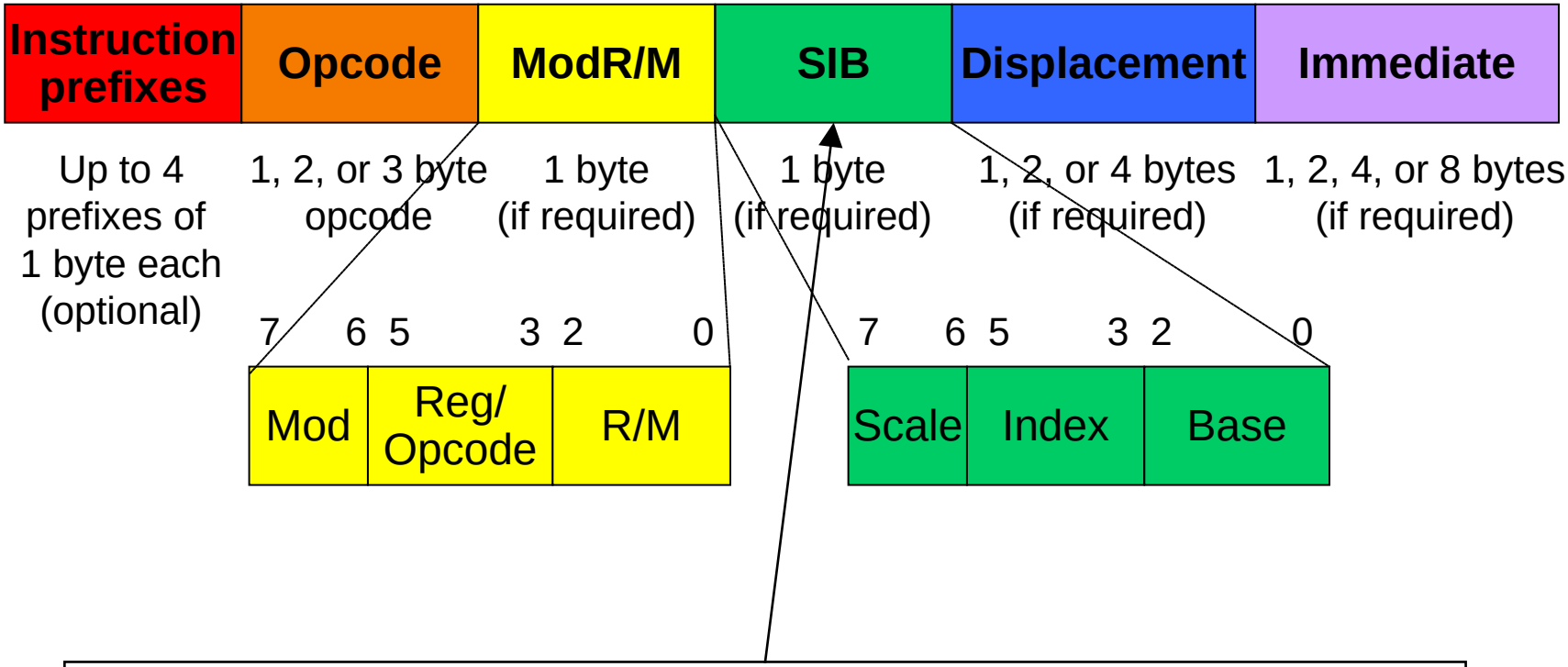
Opšti format x86_64 instrukcije



ModR/M

- Određuje vrste operanada (neposredni, registarski, memorijski)
- Određuje veličine operanada (byte, word, long)
- Ponekad označava određeni registar (za ostale registre u instr. prefiksu postoji dodatan bit):
  000 = RAX /EAX /AL; 011 = RBX /EBX /BL; 001 = RCX /ECX /CL; 010 = RDX /EDX /DL;
  110 = RSI /ESI /DH; 111 = RDI /EDI /BH; 101 = RBP /EBP /CH; 100 = RSP /ESP/ AH

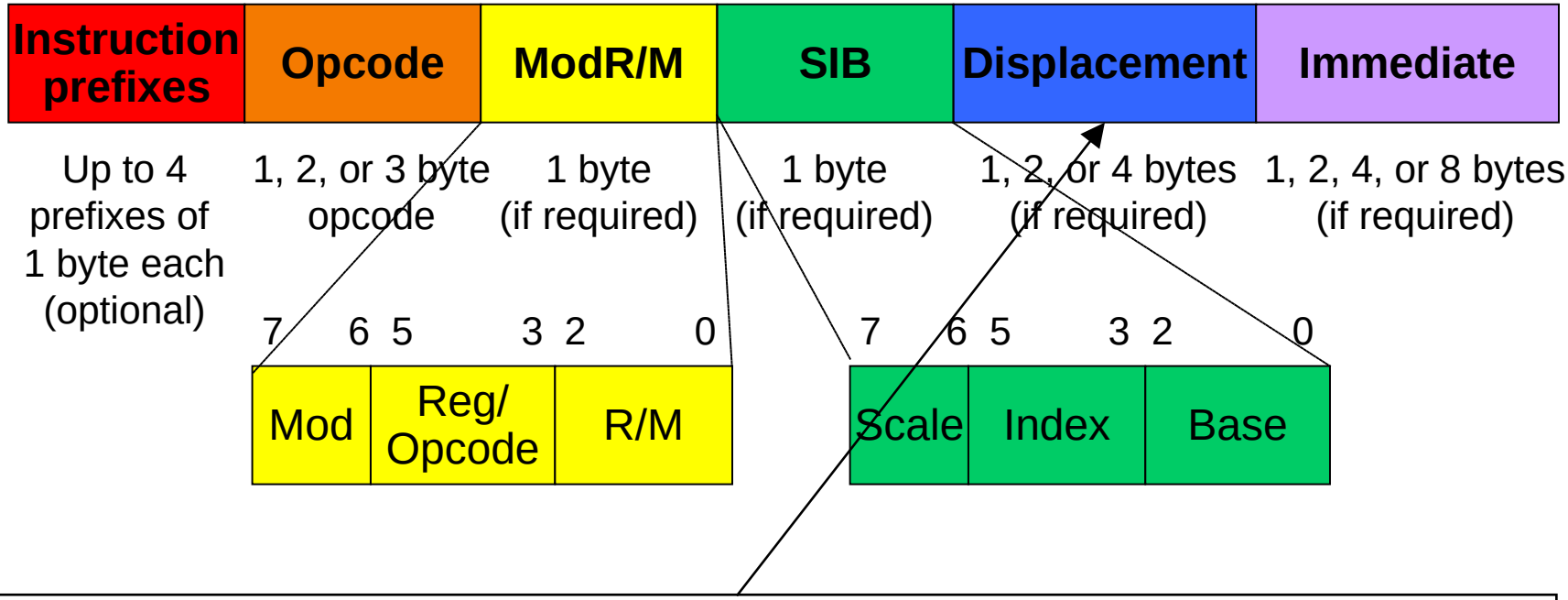
Opšti format x86_64 instrukcije



SIB

- Koristi se kada jedan od operanada i koristi faktor skaliranja, indeksni registar, i/ili bazni registar

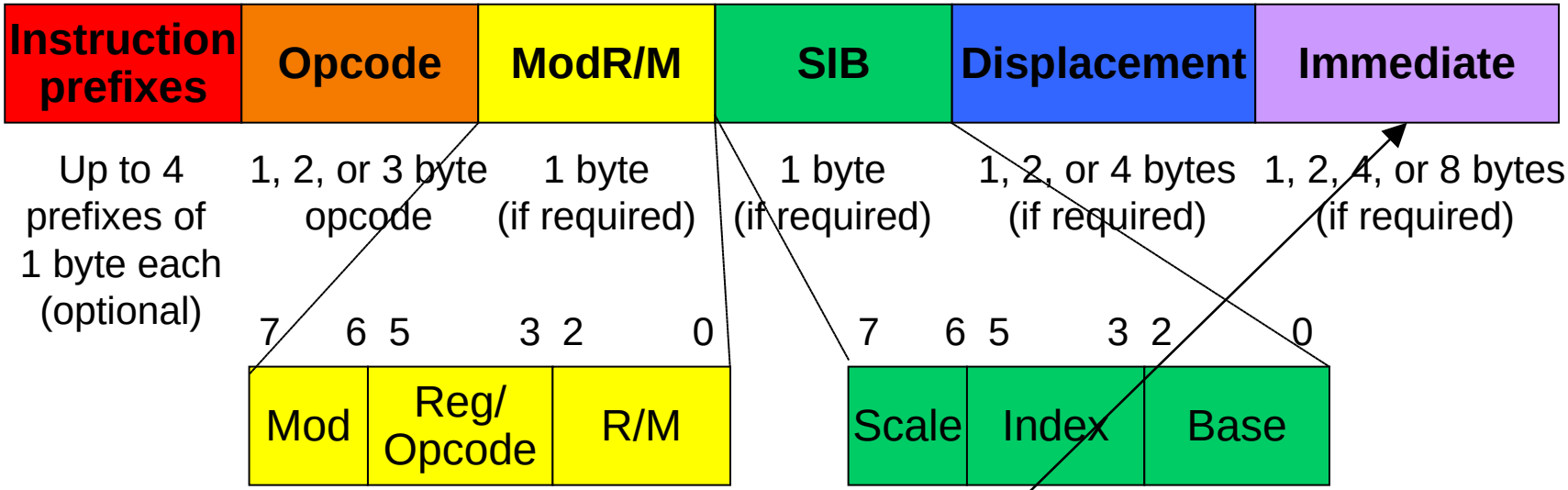
Opšti format x86_64 instrukcije



Ofset

- Koristi se u instrukcijama jump i call
- Označava Ofset između odredišta skoka i instrukcije koja sledi jump/call
- [adr odredišne instr] – [adr instr koja sledi jump/call]
- Koristi little-endian redosled

Opšti format x86_64 instrukcije



Neposredna vrednost

- Specifikuje neposredan operand
- Koristi se little-endian redosled

Primer: Potiskivanje na Stek

- Asemblerska oznaka

```
push rdx
```

- Mašinski kod:
  - X86_64 ima posedne opkodove za svaki registarski operand
    - 0x50: push eax
    - 0x51: push ecx
    - 0x52: push edx → 0101 0010
    - …
  - Rezultuje *jednobajtnom* instrukcijom
- Primetimo da ponekad jedan asemblerski mnemonik može da se mapira na grupu različitih opkodova (malo komplikuje konstrukciju asemblera)

Drugi primer: Load Effective Address

- Asemblerska oznaka:

```
lea eax, [rax+rax*4]
```

U registar **eax** ide vrednost izraza **rax + 4 * rax**

- Mašinski kod:
  - 1. bajt: 0x8D (opkod za “load effective address”)
  - 2. bajt: 0x04 (odredište eax, sa SIB bajtom)
  - 3. bajt: 0x80 (scale=4, index=eax, base=eax)

1000 1101

0000 0100

1000 0000

- Napomena: instrukcija **`lea rax, [rax+rax*4]`** ima isto kodiranje samo sa dodatim tzv. REX.W prefiksom (0x48) koji menja veličinu operanda na 64b sa podrazumevanih 32b

Reference na druge instrukcije/podatke

- Mnoge instrukcije mogu se prevesti izolovano od drugih
  - push edx
  - lea eax, [eax+eax*4]
  - mov eax, 0
  - add ecx, ebx
- Ali, neke se referišu na druge delove programa ili podatke
  - jne skip
  - push offset msg
  - call printf
- Potrebno je razrešiti (odrediti vrednost) ovih referenci da bi se generisao mašinski kod

Problem referisanja unapred

- Problem

```
    …
    jmp myLabela
    …
myLabela:
    …
```

Simbol myLabela
se koristi pre definisanja

- U trenutku kada asembler treba da generiše mašinski kod za “jmp myLabela”, nije još obradio definiciju simbola myLabela
- Instrukcija skoka sadrži **referencu unapred** na myLabela

- **Rešenje** ovog problema je da asembler napravi dva sekvencijalna prolaza kroz programski kod: u prvom prolazu gleda definicije simbola, a u drugom prolazu (kada poznaje vrednost myLabela) izvrši generisanje mašinskog koda

Konstrukcija dvoprolaznog asemblera

- Prvi prolaz
  - Asembler prolazi kroz asemblerski kod programa da napravi:
  - Tabelu simbola
    - Ključ: naziv simbola
    - Ostali atributi simbola
      - Tip Labele, u kojoj sekciji, koji pomeraj unutar sekcije, …
- Drugi prolaz
  - Asembler prolazi kroz asemblerski kod ponovo da napravi:
  - RODATA sekciju
  - DATA sekciju
  - BSS sekciju
  - TEXT sekciju
  - Sekciju relokacionih zapisa
    - Svaki relokacioni zapis označava deo koda koji linker mora da “zakrpi”
  - Kreira objektni fajl

Primer asemblerske obrade

- Programski primer

- C ekvivalent:

```
#include <stdio.h>
int main(void) {
   if (getchar() == 'A')
      printf("Hi\n");
   return 0;
}
```

(gcc –static –no-pie –masm=intel…)

- Razmotrićemo obradu ovog programa od strane asemblera…

```
        .intel_syntax noprefix
        .section .rodata
msg:
        .string "Hi\n"
        .text
        .globl  main
main:
        push    rbp
        mov     rbp, rsp
        call    getchar
        cmp     eax, 'A'
        jne     skip
        mov     edi, OFFSET msg
        call    printf
skip:
        mov     eax, 0
        mov     rsp, rbp
        pop     rbp
        ret
```

Strukture podataka asemblera (1)

- Tabela simbola

Labela	Sekcija	Ofset	Lokal?	R.Br

- Relokacioni zapisi

Sekcija	Ofset	Tip relokacije	Simbol	addend

- RODATA Sekcija (location counter: 0)

Ofset	Sadržaj	Objašnjenje

- Nema DATA ni BSS sekcija u primeru
- Inicijalno sve sekcije su prazne

- TEXT Sekcija (location counter: 0)

Ofset (decimalno)	Sadržaj (heksa)	Objašnjenje

Prvi prolaz asemblera

```
        .section .rodata
msg:
        .string "Hi\n"
        .text
        .globl  main
main:
        push    rbp
        mov     rbp, rsp
        call    getchar
        cmp     eax, 'A'
        jne     skip
        mov     edi, OFFSET msg
        call    printf
skip:
        mov     eax, 0
        mov     rsp, rbp
        pop     rbp
        ret
```

Asembler evidentira da je tekuća sekcija RODATA i ubacuje .rodata u tabelu simbola

Asembler ubacuje msg u tabelu simbola…

Strukture podataka asemblera (2)

- Tabela simbola

Labela	Sekcija	Ofset	Lokal?	R.Br
.rodata	RODATA	0	local	0
msg	RODATA	0	local	1

- msg je lokacija u RODATA sekciji sa pomerajem 0
- msg je lokalna labela
- Redni broj msg je 1

- Relokacioni zapisi
  - (bez promene)
- RODATA sekcija (location counter: 0)
  - (bez promene)
- TEXT sekcija (location counter: 0)
  - (bez promene)

Prvi prolaz asemblera (nastavak)

```
 .section .rodata
msg:
        .string "Hi\n"
        .text
        .globl  main
main:
        push    rbp
        mov     rbp, rsp
        call    getchar
        cmp     eax, 'A'
        jne     skip
        mov     edi, OFFSET msg
        call    printf
skip:
        mov     eax, 0
        mov     rsp, rbp
        pop     rbp
        ret
```

Asembler uvećava ***location counter*** RODATA sekcije za 4 (broj bajtova u stringu)…

Strukture podataka asemblera (3)

- Tabela simbola

Labela	Sekcija	Ofset	Lokal?	R.Br
.rodata	RODATA	0	local	0
msg	RODATA	0	local	1

- Relokacioni zapisi
  - (bez promene)
- RODATA sekcija (location counter: 4)
  - (bez promene)
- TEXT sekcija (location counter: 0)
  - (bez promene)

- RODATA location counter sada iznosi 4
- Da je bila još neka labela u ovoj tački programa, označavala bi adresu u RODATA sekciji sa pomerajem 4

Prvi prolaz asemblera (nastavak)

```
        .section .rodata
msg:
        .string "Hi\n"
        .text
        .globl  main
main:
        push    rbp
        mov     rbp, rsp
        call    getchar
        cmp     eax, 'A'
        jne     skip
        mov     edi, OFFSET msg
        call    printf
skip:
        mov     eax, 0
        mov     rsp, rbp
        pop     rbp
        ret
```

Asembler evidentira da je tekuća sekcija TEXT i ubacuje .text u tabelu simbola

Asembler ne radi ništa u prvom prolazu

Asembler ubacuje simbol main u tabelu simbola…

Strukture podataka asemblera (4)

- Tabela simbola

Labela	Sekcija	Ofset	Lokal?	R.Br
.rodata	RODATA	0	local	0
msg	RODATA	0	local	1
.text	TEXT	0	local	2
main	TEXT	0	local	3

- main označava adresu u TEXT sekciji sa pomerajem 0
- main je lokalna labela (asembler će to ažurirati tek u 2. prolazu)
- main ima redni broj 3

- Relokacioni zapisi
  - (bez promene)
- RODATA sekcija (location counter: 4)
  - (bez promene)
- TEXT sekcija (location counter: 0)
  - (bez promene)

Prvi prolaz asemblera (nastavak)

```
        .section .rodata
msg:
        .string "Hi\n"
        .text
        .globl  main
main:
        push    rbp
        mov     rbp, rsp
        call    getchar
        cmp     eax, 'A'
        jne     skip
        mov     edi, OFFSET msg
        call    printf
skip:
        mov     eax, 0
        mov     rsp, rbp
        pop     rbp
        ret
```

Asembler uvećava brojač lokacija TEXT sekcije dužinom svake instrukcije (van toga ne obrađuje instrukcije…)

Strukture podataka asemblera (5)

- Tabela simbola

Labela	Sekcija	Ofset	Lokal?	R.Br
.rodata	RODATA	0	local	0
msg	RODATA	0	local	1
.text	TEXT	0	local	2
main	TEXT	0	local	3

- Relokacioni zapisi
  - (bez promene)
- RODATA sekcija (location counter: 4)
  - (bez promene)
- TEXT sekcija (location counter: 24)
  - (bez promene)

- TEXT location counter sada iznosi 24

Prvi prolaz asemblera (nastavak)

```
        .section ".rodata"
msg:
        .asciz  "Hi\n"
        .section ".text"
        .globl  main
main:
        push    ebp
        mov     ebp, esp
        call    getchar
        cmp     eax, 'A'
        jne     skip
        push    offset msg
        call    printf
        add     esp, 4
skip:
        mov     eax, 0
        mov     esp, ebp
        pop     ebp
        ret
```

Asembler ubacuje skip u tabelu simbola…

Strukture podataka asemblera (6)

- Tabela simbola

Labela	Sekcija	Ofset	Lokal?	R.Br
.rodata	RODATA	**0**	local	0
msg	RODATA	0	local	1
.text	TEXT	0	local	2
main	TEXT	0	local	3
skip	TEXT	24	local	4

- skip označava lokaciju u TEXT sekciji sa pomerajem 24
- skip je lokalna labela
- skip ima redni broj 4

- Relokacioni zapisi
  - (bez promene)
- RODATA sekcija (location counter: 4)
  - (bez promene)
- TEXT sekcija (location counter: 24)
  - (bez promene)

Prvi prolaz asemblera (nastavak)

```
        .section .rodata
msg:
        .string "Hi\n"
        .text
        .globl  main
main:
        push    rbp
        mov     rbp, rsp
        call    getchar
        cmp     eax, 'A'
        jne     skip
        mov     edi, OFFSET msg
        call    printf
skip:
        mov     eax, 0
        mov     rsp, rbp
        pop     rbp
        ret
```

Asembler uvećava brojač lokacija TEXT sekcije dužinom svake instrukcije…

Strukture podataka asemblera (7)

- Tabela simbola

Labela	Sekcija	Ofset	Lokal?	R.Br
.rodata	RODATA	**0**	local	0
msg	RODATA	0	local	1
.text	TEXT	0	local	2
main	TEXT	0	local	3
skip	TEXT	24	local	4

- Relokacioni zapisi
  - (bez promene)
- RODATA sekcija (location counter: 4)
  - (bez promene)
- TEXT sekcija (location counter: 34)
  - (bez promene)

- TEXT location counter sada iznosi 35

Kraj prvog prolaza asm, počinje drugi

- Kraj prvog prolaza
  - Assembler je (delimično) kreirao tabelu simbola
  - Asembler sada zna vrednosti svakog lokalno definisanog simbola

- Početak drugog prolaza
  - Asembler postavlja na 0 sve brojače lokacija svih sekcija…

Strukture podataka asemblera (8)

- Tabela simbola

Labela	Sekcija	Ofset	Lokal?	R.Br
.rodata	RODATA	**0**	local	0
msg	RODATA	0	local	1
.text	TEXT	0	local	2
main	TEXT	0	local	3
skip	TEXT	24	local	4

- Relokacioni zapisi
  - (bez promene)
- RODATA sekcija (location counter: 0)
  - (bez promene)
- TEXT sekcija (location counter: 0)
  - (bez promene)

- Brojači lokacija postavljeni na 0

Drugi prolaz asemblera

```
        .section .rodata
msg:
        .string "Hi\n"
        .text
        .globl  main
main:
        push    rbp
        mov     rbp, rsp
        call    getchar
        cmp     eax, 'A'
        jne     skip
        mov     edi, OFFSET msg
        call    printf
skip:
        mov     eax, 0
        mov     rsp, rbp
        pop     rbp
        ret
```

Asembler evidentira da je tekuća sekcija RODATA

Asembler ne radi ništa

Asembler generiše bajtove u RODATA sekciji i uvećava location counter…

Strukture podataka asemblera (9)

- Tabela simbola
  - (bez promene)
- Relokacioni zapisi
  - (bez promene)
- RODATA sekcija (location counter: 4)

Ofset	Sadržaj (hex)	Objašnjenje
0	48	ASCII kod za ‘H’
1	69	ASCII kod za ‘i’
2	0A	ASCII kod za ‘\n’
3	00	ASCII kod za null znak

- TEXT sekcija (location counter: 0)
  - (bez promene)

- Location counter uvećan za 4

- RODATA sekcija sadrži bajtove datog stringa

Drugi prolaz asemblera (nastavak)

```
        .section .rodata
msg:
        .string "Hi\n"
        .text
        .globl  main
main:
        push    rbp
        mov     rbp, rsp
        call    getchar
        cmp     eax, 'A'
        jne     skip
        mov     edi, OFFSET msg
        call    printf
skip:
        mov     eax, 0
        mov     rsp, rbp
        pop     rbp
        ret
```

Asembler evidentira da je tekuća sekcija TEXT

Asembler ažurira atribut Lokal? za main u tabeli simbola

Strukture podataka asemblera (10)

- Tabela simbola

Labela	Sekcija	Ofset	Lokal?	R.Br
.rodata	RODATA	**0**	local	0
msg	RODATA	0	local	1
.text	TEXT	0	local	2
main	TEXT	0	global	3
skip	TEXT	24	local	4

- Relokacioni zapisi
  - (bez promene)
- RODATA sekcija (location counter: 4)
  - (bez promene)
- TEXT sekcija (location counter: 0)
  - (bez promene)

- main je globalni simbol

Drugi prolaz asemblera (nastavak)

```
        .section .rodata
msg:
        .string "Hi\n"
        .text
        .globl  main
main:
        push    rbp
        mov     rbp, rsp
        call    getchar
        cmp     eax, 'A'
        jne     skip
        mov     edi, OFFSET msg
        call    printf
skip:
        mov     eax, 0
        mov     rsp, rbp
        pop     rbp
        ret
```

Asembler ne radi ništa

Asembler generiše mašinski kod u tekućoj (TEXT) sekciji…

Strukture podataka asemblera (11)

- Tabela simbola
  - (bez promene)
- Relokacioni zapisi
  - (bez promene)
- RODATA sekcija (location counter: 4)
  - (bez promene)
- TEXT sekcija (location counter: 1)

Ofset	Sadržaj	Objašnjenje
0	55	push rbp 01010101 Mašinski kod “push rbp” instrukcije

Drugi prolaz asemblera (nastavak)

```
        .section .rodata
msg:
        .string "Hi\n"
        .text
        .globl  main
main:
        push    rbp
        mov     rbp, rsp
        call    getchar
        cmp     eax, 'A'
        jne     skip
        mov     edi, OFFSET msg
        call    printf
skip:
        mov     eax, 0
        mov     rsp, rbp
        pop     rbp
        ret
```

Assembler generiše mašinski kod code u tekućoj (TEXT) sekciji…

Strukture podataka asemblera (12)

- Tabela simbola
  - (bez promene)
- Relokacioni zapisi
  - (bez promene)
- RODATA sekcija (location counter: 4)
  - (bez promene)
- TEXT sekcija (location counter: 4)

Ofset	Sadržaj	Objašnjenje
…	…	…
1-3	48 89 E5	mov ebp,esp REX.W 10001001 11 100 101 Ovo je “mov” instrukcija čiji izvorišni operand je registar Polje M označava registar Izvorišni registar je RSP Odredišni registar je RBP

Drugi prolaz asemblera (nastavak)

```
        .section .rodata
msg:
        .string "Hi\n"
        .text
        .globl  main
main:
        push    rbp
        mov     rbp, rsp
        call    getchar
        cmp     eax, 'A'
        jne     skip
        mov     edi, OFFSET msg
        call    printf
skip:
        mov     eax, 0
        mov     rsp, rbp
        pop     rbp
        ret
```

Asembler generiše mašinski kod u tekućoj (TEXT) sekciji…

Strukture podataka asemblera (12)

- Tabela simbola
  - (bez promene)
- Relokacioni zapisi
  - (bez promene)
- RODATA sekcija (location counter: 4)
  - (bez promene)
- TEXT sekcija (location counter: 9)

- Asembler gleda u tabelu simbola da nađe ofset getchar-a
- getchar nije u tabeli simbola
- Asembler ne može da izračuna pomeraj koji treba smestiti na ofsetu 5 TEXT sekcije
- …

Ofset	Sadržaj	Objašnjenje
…	…	…
4-8	E8 ????????	call getchar 11101000 ???????????????????????????? Ovo je “call” instrukcija sa 4-bajtnim displacement operandom (relativna adresa skoka)

Strukture podataka asemblera (13)

- Tabela simbola

Labela	Sekcija	Ofset	Lokal?	R.Br
.rodata	RODATA	0	local	0
msg	RODATA	0	local	1
.text	TEXT	0	local	2
main	TEXT	0	global	3
skip	TEXT	24	local	4
getchar	?	?	global	5

- Relokacioni zapisi
  - (bez promene)
- RODATA sekcija (location counter: 4)
  - (bez promene)
- TEXT sekcija (location counter: 9)
  - (bez promene)

- Asembler dodaje getchar u tabelu simbola
- Potom…

Strukture podataka asemblera (14)

- Tabela simbola
  - (bez promene)
- Relokacioni zapisi

Sekcija	Ofset	Tip relokacije	R.Br	addend
TEXT	5	R_X86_64_PC32	5	-4

- Asembler generiše relokacioni zapis, čime nalaže linkeru da "zakrpi" kod

Linkeru se nalaže da izmeni sadržaj TEXT sekcije na ofsetu 5. Radi se o "PC relativnoj" 32 bitnoj adresi simbola rednog broja 5 (getchar).

Ofset-addend treba da bude jednako vrednosti RIP u vreme izvršavanja tj. adresi instrukcije iza call

- RODATA sekcija (location counter: 4)
  - (bez promene)
- TEXT sekcija (location counter: 9)
  - (bez promene)

Drugi prolaz asemblera (nastavak)

```
        .section .rodata
msg:
        .string "Hi\n"
        .text
        .globl  main
main:
        push    rbp
        mov     rbp, rsp
        call    getchar
        cmp     eax, 'A'
        jne     skip
        mov     edi, OFFSET msg
        call    printf
skip:
        mov     eax, 0
        mov     rsp, rbp
        pop     rbp
        ret
```

Asembler generiše mašinski kod u tekućoj (TEXT) sekciji…

Strukture podataka asemblera (15)

- Tabela simbola
  - (bez promene)
- Relokacioni zapisi
  - (bez promene)
- RODATA sekcija (location counter: 4)
  - (bez promene)
- TEXT sekcija (location counter: 12)

Ofset	Sadržaj	Objašnjenje
…	…	…
9-11	83 F8 41	cmp eax, 'A' 10000011 11 111 000 01000001 Instrukcija ima jednobajtni neposredni operand Polje M označava registar Radi se o “cmp” instrukciji Odredišni registar je EAX Neposredni operand je ‘A’

Drugi prolaz asemblera (nastavak)

```
        .section .rodata
msg:
        .string "Hi\n"
        .text
        .globl  main
main:
        push    rbp
        mov     rbp, rsp
        call    getchar
        cmp     eax, 'A'
        jne     skip
        mov     edi, OFFSET msg
        call    printf
skip:
        mov     eax, 0
        mov     rsp, rbp
        pop     rbp
        ret
```

Asembler generiše mašinski kod u tekućoj (TEXT) sekciji…

Strukture podataka asemblera (16)

- Tabela simbola
  - (bez promene)
- Relokacioni zapisi
  - (bez promene)
- RODATA sekcija (location counter: 4)
  - (bez promene)
- TEXT sekcija (location counter: 14)

- Asembler gleda u tabelu simbola da nađe ofset skip (24)
- Asembler oduzima ofset sledeće instrukcije (14)
- Rezultujući pomeraj je 24-14=10 (decimalno)

Ofset	Sadržaj	Objašnjenje
…	…	…
12-13	75 0A	jne skip 01110101 00001010 Ovo je jne instrukcija koja ima 1 bajtni operand Pomeraj između odredišne instrukcije i sledeće instrukcije je 13

Drugi prolaz asemblera (nastavak)

```
        .section .rodata
msg:
        .string "Hi\n"
        .text
        .globl  main
main:
        push    rbp
        mov     rbp, rsp
        call    getchar
        cmp     eax, 'A'
        jne     skip
        mov     edi, OFFSET msg
        call    printf
skip:
        mov     eax, 0
        mov     rsp, rbp
        pop     rbp
        ret
```

Asembler generiše mašinski kod u tekućoj (TEXT) sekciji…

Strukture podataka asemblera (16)

- Tabela simbola
  - (bez promene)
- Relokacioni zapisi
  - (bez promene)
- RODATA sekcija (location counter: 4)
  - (bez promene)
- TEXT sekcija (location counter: 19)

- Asembler zna ofset za msg (0) u okviru RODATA sekcije
- Ali ne zna lokaciju RODATA sekcije, tako da ne zna flat adresu za msg
- Potom…

Ofset	Sadržaj	Objašnjenje
…	…	…
14-18	BF 00000000	mov edi, OFFSET msg 10111111 00000000000000000000000000000000 Ovo je push instrukcija sa 4 bajtnim immediate operandom

Strukture podataka asemblera (17)

- Tabela simbola
  - (bez promene)
- Relokacioni zapisi

Sekcija	Ofset	Tip relokacije	R.Br	addend
…	…	…	…	
TEXT	15	R_X86_64_32	0	0

- Asembler generiše relokacioni zapis, nalažući linkeru da popravi kod

Linker treba da popravi TEXT sekciju na ofsetu 15 Radi se o apsolutnoj 32bitnoj adresi simbola rednog broja 0 (.rodata).

- RODATA sekcija (location counter: 4)
  - (bez promene)
- TEXT sekcija (location counter: 19)
  - (bez promene)

Drugi prolaz asemblera (nastavak)

```
        .section .rodata
msg:
        .string "Hi\n"
        .text
        .globl  main
main:
        push    rbp
        mov     rbp, rsp
        call    getchar
        cmp     eax, 'A'
        jne     skip
        mov     edi, OFFSET msg
        call    printf
skip:
        mov     eax, 0
        mov     rsp, rbp
        pop     rbp
        ret
```

Asembler generiše mašinski kod u tekućoj (TEXT) sekciji…

Strukture podataka asemblera (18)

- Tabela simbola
  - (bez promene)
- Relokacioni zapisi
  - (bez promene)
- RODATA sekcija (location counter: 4)
  - (bez promene)
- TEXT sekcija (location counter: 24)

- Asembler gleda u tabelu simbola da nađe ofset za printf
- printf nije u tabeli simbola
- Asembler ne može da izračuna pomeraj koji treba upisati na ofsetu 20 TEXT sekcije
- Potom…

Ofset	Sadržaj	Objašnjenje
…	…	…
19-23	E8 ????????	call printf 11101000 ???????????????????????????? Ovo je “call” instrukcija sa 4-bajtnim operadnom Ovo je displacement (relat. Adr. Skoka)

Strukture podataka asemblera (19)

- Tabela simbola

Labela	Sekcija	Ofset	Lokal?	R.Br
.rodata	RODATA	0	local	0
msg	RODATA	0	local	1
.text	TEXT	0	local	2
main	TEXT	0	global	3
skip	TEXT	24	local	4
getchar	?	?	global	5
printf	?	?	global	6

- Asembler dodaje printf u tabelu simbola
- Potom…

- Relokacioni zapisi
  - (bez promene)
- RODATA sekcija (location counter: 4)
  - (bez promene)
- TEXT sekcija (location counter: 24)
  - (bez promene)

Strukture podataka asemblera (20)

- Tabela simbola
  - (bez promene)
- Relokacioni zapisi

Sekcija	Ofset	Tip relokacije	R.Br	adend
…	…	…	…	
TEXT	20	R_X86_64_PC32	6	-4

- Asembler generiše relokacioni zapis, nalažući linkeru da ispravi kod

Linker treba da popravi TEXT sekciju na ofsetu 20. Radi se o PC relativnoj 32bitnoj adresi simbola rednog broja 6 (printf).
Ofset-addend treba da bude jednako vrednosti RIP u vreme izvršavanja (24)

- RODATA sekcija (location counter: 4)
  - (bez promene)
- TEXT sekcija (location counter: 24)
  - (bez promene)

Drugi prolaz asemblera (nastavak)

```
        .section .rodata
msg:
        .string "Hi\n"
        .text
        .globl  main
main:
        push    rbp
        mov     rbp, rsp
        call    getchar
        cmp     eax, 'A'
        jne     skip
        mov     edi, OFFSET msg
        call    printf
skip:
        mov     eax, 0
        mov     rsp, rbp
        pop     rbp
        ret
```

Asembler ne radi ništa

Asembler generiše mašinski kod u tekućoj (TEXT) sekciji…

Strukture podataka asemblera (21)

- Tabela simbola, Relokacioni zapisi, RODATA sekcija
  - (bez promene)
- TEXT sekcija (location counter: 34)

Ofset	Sadržaj	Objašnjenje
…	…	…
24-28	B8 00000000	mov eax, 0 10111000 00000000000000000000000000000000 Ovo je mov instrukcija sa 4-bajtnim operadnom Neposredni operand je 0
29-31	48 89 EC	mov rsp,rbp REX.W 10001001 11 101 100 Ovo je “mov” instr. sa registrom kao izvorištem Polje M označava registar Izvorišni registar je EBP Odredišni registar je ESP
32	5D	pop rbp 01011101 Ovo je “pop ebp” instrukcija
33	C3	ret 11000011 Ovo je “ret” instrukcija

Kako videti generisane relokacije?

```
$ objdump -d -r -M intel main2.o
0000000000000000 <main>:
   0:    55                       push   rbp
   1:    48 89 e5                 mov    rbp,rsp
   4:    e8 00 00 00 00           call   9 <main+0x9>
   5: R_X86_64_PC32  getchar-0x4
   9:    83 f8 41                 cmp    eax,0x41
   c:    75 0a                    jne    18 <skip>
   e:    bf 00 00 00 00           mov    edi,0x0
   f: R_X86_64_32       .rodata
  13:    e8 00 00 00 00           call   18 <skip>
   14: R_X86_64_PC32          printf-0x4

0000000000000018 <skip>:
  18:    b8 00 00 00 00           mov    eax,0x0
  1d:    48 89 ec                 mov    rsp,rbp
  20:    5d                       pop    rbp
  21:    c3                  ret
```

Konstrukcija jednoprolaznog asemblera

- glavni razlog postojanja drugog prolaza je potreba da se razreši obraćanje unapred, to jest, korišćenje labela pre njihovog definisanja
- moguće je napraviti i jednoprolazni asembler, uz modifikaciju u strukturama podataka
- modifikujemo tabelu simbola (asemblera, a bez izmena u ELF fajlu) uvođenjem polja **defined** (da li je labela definisana) i **flink** koje će ukazivati na jedan ulaz novouvedene tabele obraćanja unapred (engl. forward reference table):

Konstrukcija jednoprolaznog asemblera

- **Promene u tabeli simbola (opis jednog ulaza tabele simbola)**

```
struct ST_entries {
  char *name;
   int size;
  int value;
. . .
   bool defined;        // true kada se naiđe na definiciju
    ST_forwardrefs *flink;// početak liste obraćanja unapred
};
```

- **Lista obraćanja unapred (struktura jednog ulaza liste)**

```
struct ST_forwardrefs { // obraćanja identifikatorima
                                    // pre njihove definicije
    int patch;                  // adresa u programu za izmenu
    ST_forwardrefs *nlink;// pokazivač na sledeći zapis
};
```

Algoritam obrade labela u jednom prolazu

- Kada se u adresnom polju instrukcije naiđe na neku labelu, pretražuje se tabela simbola za taj identifikator. Nekoliko ishoda pretrage su mogući:
  - Ako je labela već definisana, biće nađena u tabeli simbola, polje **defined** biće **true,** pa se odgovarajuća vrednost može odmah pročitati iz polja **value.**
  - Ako labela nije nađena u tabeli simbola, biće kreiran novi ulaz za nju, postavljeno polje **name** i polje **defined = false.** Polje **flink** se postavlja da pokazuje na novo kreirani ulaz u tabeli obraćanja unapred**.** U tom novom ulazu u polje byte upisuje se tekuća vrednost ***brojača lokacija LC*** *tj. adresa u mašinskom kodu koja odgovara adresnom polju tekuće instrukcije***.**

Algoritam obrade labela u jednom prolazu

- Za nedefinisani simbol usvaja se vrednost 0 pri računanju vrednosti adresnog polja instrukcije. Ako je u pitanju relativno adresiranje (skokovi), to znači da se u adresno polje razlika 0 – vrednost Location brojača za sledeću instrukciju.

- Ako se labela nalazi u tabeli i pri tome je **defined == false,** onda se u tabelu obraćanja unapred dodaje novi ulaz, isto kao u prethodnoj tački**.**

- na kraju prolaza, kada su sve labele (u normalnoj situaciji kada nema grešaka) definisane, potrebno je proći kroz sve neprazne liste obraćanja unapred i dodati poznatu vrednost simbola na adrese navedene u polju byte u zapisu ST_forwardrefs. Taj postupak se naziva ***backpatching koda***

Primer (nije u gnu sintaksi)

```
0000                                BEG    ; count the bits in a number
0000 E4 00                          IN   AL,      0    ; Read(A)
0002                        LOP                       ; REPEAT
0002 C0 E8 01                       SHR  AL,      1    ;  A := A DIV 2
0005 0F 83 F5 FF FF FF              JNC  EVN           ;  IF A MOD 2 # 0
000B 88 05 00 00 00 00              MOV  TEMP,    AL   ;  TEMP := A
0011 8A 05 00 00 00 00              MOV  AL,      BITS
0017 FE C0                          INC  AL
0019 88 05 00 00 00 00              MOV  BITS,    AL   ;  BITS:=BITS+1
001F 8A 05 00 00 00 00              MOV  AL,      TEMP ;    A := TEMP
0025 84 C0                          TEST AL,      AL   ;  Z := (A == 0)
0027 0F 85 D5 FF FF FF  EVN         JNZ  LOP           ; UNTIL A == 0
002D 8A 05 00 00 00 00              MOV  AL,      BITS ;
0033 E6 00                          OUT  0,       AL   ; Write(BITS)
0035 F4                             HLT                ; terminate prog.
0036 ??                     TEMP    DS   5             ; VAR TEMP : BYTE
003B 00                     BITS    DB   0             ;     BITS : BYTE
003C                                END
```

Primer

- izgled tabele simbola i tabele obraćanja unapred posle obrade instrukcije na adresi 19:

```
Symbol Table
BITS        --- undefined            FLINK: 001b 0013
TEMP        --- undefined            FLINK: 000d
EVN         --- undefined            FLINK: 0007
LOP        DWORD PTR 00000002        FLINK:
```

- izgled ovih tabela posle prolaska kroz ceo program:

```
Symbol Table
BITS         BYTE PTR 0000003b FLINK: 002f 001b 0013
TEMP         BYTE PTR 00000036 FLINK: 0021 000d
EVN         DWORD PTR 00000027 FLINK: 0007
LOP         DWORD PTR 00000002 FLINK:
```

Zaključak

- Asembler: čita asemblerski program
  1. prolaz: Generiše tabelu simbola
     - TS sadrži informacije o labelama u programu
  2. prolaz: Koristi tabelu simbola da generiše kod
     - TEXT, RODATA, DATA, BSS Sekcije
     - relokacioni zapisi
  - Kreira objektni fajl (po ELF formatu)
- Jednoprolazni asembler radi backpatching tehniku
- Linkeru je ostavljeno:
  - Razrešavanje simbola: Razrešava reference na uvezene simbole
  - Relokacija: Uses Tabela simbola and Relokacioni zapisi to patch code
  - Pravljenje izvršnog fajla (po ELF formatu)