SISTEMSKI SOFTVER

Projketni zadatak

Dokumentacija

Jednoprolazni asembler i interpretativni emulator

Ana Stakić 0287/2016
Elektrotehnički fakultet Univerzitet u Beogradu

1 Opis projekta

1.1 Zadatak 1

Cilj prvog zadatka projekta je izrada jednoprolaznog asemblera za procesor opisan u nastavku. Ulaz asemblera je .s fajl sa izvornim kodom, napisanim u skladu sa sintaksom opisanom u nastvaku. Izlaz asemblera je predmetni program zapisan u .o fajlu, po uzoru na predmetni program prikazan na vežbama. Generisanje izlaznog fajla vrši se principima GNU asemblera, tako da se sve sekcije smeštaju od adrese 0.

1.2 Zadatak 2

Drugi zadatak odnosi se na pisanje interpretativnog emulatora za procesor opisan u nastavku. Ulaz emulatora je izlaz asemblera, sa mogućnošću prosleđivanja većeg broja predmetnih programa koje je potrebno povezati i pokrenuti. Ulazni fajlovi se prosleđuju kao argumenti komandne linije. Početak programa vezuje se za globalni simbol _start.

1.3 Okruženje

Izrada projekta je pod operativnim sistemom Linux, na x86 arhitekturi u C++ programskom jeziku.

1.4 Opis asembelera

Elementi asemblera:

- U jednoj liniji može biti najviše jedna komanda (direktiva ili instrukcija)
- Na početku svake linije može da bude labela, iza koje stoji " : "
- Labela može da stoji sama u liniji, što je ekvivalentno kao da stoji u liniji sa prvim narednim sadržajem
- Simboli se izvoze direktivom .global <ime_simbola>, pri čemu može da se navede više simbola razdvojenih zapetama
- Simbol koji se izvozi mora biti definisan u okviru programa koji se prevodi
- Simboli se uvoze direktivom .extern <ime_simbola>, pri čemu može da se navede više simbola razdvojenih zapetama
- Direktive:
  - .equ <ime_simbola>, <izraz> – gde <izraz> predstavlja sekvencu simbola ili literala razdvojenih operatorima plus i minus
  - .byte <ime_simbola>/<literal> – umeće jedan bajt podataka po argumentu na mesto na kom je dedfinisana
  - .word <ime_simbola>/<literal> – umeće dva bajta podataka po argumentu na mesto na kom je dedfinisana
  - .skip <literal> – preskače onoliko bajtova koliko je navedeno u argumentu i popunjava ih nulama
  - .end – direktiva kojom se završava izvorni kod, ostatak fajla se odbacuje
  - .section <ime_sekcije> – sekcija sa navedenim proizvoljnim imenom
- Komentar je od znaka # do kraja reda

Sintaksa operanada u okviru asemblerskih naredbi koje pristupaju podacima:

- $<literal> - **neposredna vrednost** <literal>
- $<simbol> - **neposredna vrednost** <simbol>
- %r<num> - **vrednost iz registra** r<num>
- (%r<num>) - **vrednost iz memorije na adresi iz registra** %r<num>
- <literal>(%r<num>) - **vrednost iz memorije na adresi** <literal> + r<num>
- <simbol>(%r<num>) - **vrednost iz memorije na adresi** <simbol> + r<num>
- <simbol>(%pc/%r7) - **vrednost iz memorije na adresi** <simbol> (**PC relativno**)
- <literal> - **vrednost iz memorije na adresi** <literal> (**apsolutno**)
- <simbol> - **vrednost iz memorije na adresi** <simbol> (**apsolutno**)

Sintaksa operanada u okviru asemblerskih naredbi koje pristupaju podacima:

- <literal> - **skok na adresu** <literal>
- <simbol> - **skok na adresu** <simbol>
- ***%**r<num> - **skok na adresu iz registra** r<num>
- ***(%**r<num>) - **skok na adresu iz memorije na adresi iz registra** %r<num>
- ***<literal>(%**r<num>) - **skok na adresu iz memorije na adresi** <literal> + r<num>
- ***<simbol>(%**r<num>) - **skok na adresu iz memorije na adresi** <simbol> + r<num>
- ***<simbol>(%pc/%r7**) - **skok na adresu iz memorije na adresi** <simbol> (**PC relativno**)
- ***<literal>** - **skok na adresu iz memorije na adresi** <literal> (**apsolutno**)
- ***<simbol>** - **skok na adresu iz memorije na adresi** <simbol> (**apsolutno**)

Dodatne napomene:

- sve aritmetičke operacije se izvode tako da odgovaraju označenim celim brojevima,
- iza mnemonika asemblerske naredbe, bez belih znakova, može se navesti sufiks **b** ili **w** kako bi se eksplicitno naznačila veličina operanada date instrukcije
- instrukcije **cmp** i **test** nigde ne čuvaju direktni rezultat odgovarajuće operacije, već samo u skladu sa rezultatom postavljaju nove vrednosti flegova u psw registru
- kombinacije instrukcija i operanada, za koje ne postoji razumno tumačenje, proglašava se greškom

1.5 Opis procesora

Procesor je 16-bitni dvoadresni sa Von_Neuman arhitekturom. Adresibilna jedinica je jedan bajt, veličina adresnog prostora 2^{16}B. Od adrese 0xFF00 nalazi se prostor za memorijski mapiirane registre. Od adrese 0x0000 se nalazi IVT tabela:

- ulaz 0 sadrži adresu prekidne rutine koja se izvršava prilikom pokretanja odnosno resetovanja čitavog procesora
- ulaz 1 sadrži adresu prekidne rutine koja se izvršava ukoliko se pokuša izvršavanje nekorektne instrukcije
- ulaz 2 sadrži adresu prekidne rutine koja se izvršava kada stigne zahtev za prekid od tajmera
- ulaz 3 sadrži adresu prekidne rutine koja se izvršava kada stigne zahtev za prekid od terminala
- ostali ulazi su slobodni za korišćenje od strane programera

Procesor poseduje osam opštenamenskih 16-bitnih registara označenih sa r<num> gde <num> može biti vrednost od nula do sedam. Registar r7 se koristi kao pc registar, aregistar r6 kao sp.

Stek raste ka nižim adresama, sp ukazuje na poslednju zauzetu lokaciju. Pored opštenamenskih postoji i psw registar:

- bit 15 I (Interrupt)
- bit 14 Tl (Terminal)
- bit 13 Tr (Timer)
- bit 3 N (Negative)
- bit 2 C (Carry)
- bit 1 O (Overflow)
- bit 0 Z (Zero)

1.6 Periferije

1.6.1 Terminal

Terminal poseduje dva memorijski mapirana registra. Na adresi 0xFF00 memorijskog adresnog prostora nalazi se **data_out** registar izlaznih podataka. Upisom vrednosti u data_out registar na tekućoj poziciji displeja ispisuje se znak koji prema ASCII tabeli odgovara upisanoj vrednosti. Na adresi 0xFF02 memorijskog adresnog prostora nalazi se **data_in** registar ulaznih podataka. Kada se pritisne neki taster upisuje se ASCII kod pritisnutog tastera u data_in registar i terminal, kao periferija posmatranog procesora, generiše zahtev za prekid.

1.6.2 Timer

Tajmer kao periferija periodično generiše zahtev za prekid. Perioda generisanja zahteva za prekid definisana je sadržajem **timer_cfg** konfiguracionog registra tajmera. Registar timer_cfg je memorijski mapiran registar i nalazi se na adresi 0xFF10 memorijskog adresnog prostora. Njegova inicijalna vrednost nakon pokretanja odnosno resetovanja računarskog sistema jeste 0x0000.
Perioda generisanja zahteva za prekid u zavisnosti od $T_2T_1T_0$ vrednosti je sledeća: 0x0 -> 500ms, 0x1 -> 1000ms, 0x2 -> 1500ms, 0x3 -> 2000ms, 0x4 -> 5000ms, 0x5 -> 10s, 0x6 -> 30s i 0x7 -> 60s.

2 Opis rešenja

2.1 Asembler

Jednoprolazni asembler radi tako što se prvo vrši sređivanje ulaznog fajla, uklanjaju se komentari, eventualni tekst nakon .end direktive, uklanja se višak praznih redova, praznih i tab znakova.

Zatim se prolazi kroz sređeni ulazni fajl, i popunjava se tabela simbola, koja sadrži sve deklarisane i/ili definisane simbole (labele, nazive sekcija, equ simbole, simbole koji se uvoze). Ukoliko je nekom simbolu dodeljena vrednost ili se u instrukcijama koristi do tada nedefinisan simbol, taj simbol se u tabelu simbola dodaje sa posebnim flegom – oznakom da je nedefinisan, a na mestu njegovog korišćenja formira se objekat potencijalne relokacije tog simbola sa svim potrebnim informacijama za njenu realizaciju u trenutku definisanja simbola.

U strukture za čuvanje sadržaja sekcija dodaje se sav mašinski kod koji je u tom trenutku moguće odrediti. Na mestima korišćenja nedefinisanih simbola formiraju se posebne strukture koje su povezane sa samim simbolom i potencijalnom relokacijom tog simbola, i u trenutku definicije tog simbola nepoznati bajtovi iz instrukcije se formiraju.

Na kraju prolaza asemblera, vrši se provera da li su svi simboli definisani, kao i da li prilikom definisanja equ simbola ima kružne zavisnosti simbola.

Prilikom razrešavanja simbola dobijenih equ direktivom, ukoliko je vrednost koja im se dodeljuje izraz, takav simbol može biti relokatibilan. Simboli koji su elemetni izraza učestvuju u stvaranju tabele indeksa klasifikacije, u kojoj jedan ulaz pripada jednoj sekciji, a svaki od elemenata izraza (ukoliko je simbol) dodaje se ulazu one sekcije u kojoj je taj simbol definisan sa apsolutnom vrednošću 1 (jedan) i sa predznakom koji dati simbol ima u izrazu. Svi extern simboli idu u zaseban ulaz za UND sekciju i dodaju se uvek sa predznakom plus (+1) nezavisno od stvarnog predzanaka u izrazu. Literali ne utiču na indeks klasifikacije. Validni izrazi imaju vrednost 1 za ili jedan ili nijedan ulaz (svi ostali imaju vrednost 0) u tabeli indeksa klasifikacije.

2.2 Linker

Linker spaja izlazne fajlove asemblera, koji su ulazni fajlovi za emulator, formirajući sopstvenu tabelu globalnih simbola dobijenih iz svih fajlova koje je dobio kao ulaz. Takođe, spaja sadržaje svih sekcija onim redom kako nailazi na njih, prolazeći kroz ulazne fajlove redosledom njihovog navođenja. Specijalno u slučaju da se prilikom pokretanja emulatora navede komanda
–place=<section>@<address>
onda se prvo smeštaju te sekcije od traženih adresa, a zatim ostale sekcije redom od kraja poslednje smeštene sekcije.

Ukoliko za neku od sekcija nema dovoljno mesta, ili se preklope sa IVT tabelom (0x0000 – 0x0010) ili adresnim prostorom za memorijski mapirane registre (0xFF00 – 0xFFFF), linker će prijaviti grešku i obustaviti izvršavanje. Pomeranjem sekcija, linker dobija tačne vrednosti svih sekcija i globalnih simbola u tabeli simbola i radi prepravljanje objektnog koda na mestima relokacija.

Linker radi proveru da li je definisan globalni simbol _start, i njegovu vrednost zajedno sa kompletnim sadržajem memorije prosleđuje emulatoru na izvršavanje.

2.3 Emulator

Interpretativni emulator obavlja emuliranje rada procesora, memorije, terminala kao ulazno izlazne periferije i hardverskog tajmera. Kako same sekcije nemaju oznake da li se njihov sadržaj može izvršavati, odgovornost progamera je da obezbedi adekvatne vrednosti I ponašanje.

Adresa prve instrukcije koja se upisuje u PC je vrednost _start simbola. Počev od te adrese emulator ciklično radi obradu instrukcija, uz proveravanje timer-a nakon svake izvršene instrukcije i osluškivanja tastature u zasebnoj niti za potrebe terminala.

Kod izvršavanja instrukcija, prvo se radi dekodovanje instrukcije, zatim se dohvataju destinacioni i/ili izvorišni operand, obavlja se operacija nad operandima, setuju se PSW biti ukoliko je potrebno i upisuje rezultat u memoriju ili registar po potrebi. Prilikom upisa podatka u memoriju radi se provera da li je podatak upisan u data_out registar i ukoliko jeste ta vrednost se ispisuje u terminal. Takođe, ukoliko je upisana vrednost u timer_cfg registar, očitava se vrednost najniža tri bita i setuje dužina trajanja između generisanja zahteva za prekid od strane timer-a.

3 Prevođenje i pokretanje programa

3.1 Prevođenje programa

Za prevođenje programa koristi se g++ prevodilac, verzija 5.0. Skripta za instalaciju nalazi se u okviru Assembler/bin/ foldera.

Makefile i src folder moraju biti u istom direktorijumu, i prevođenje se vrši komandom **make**, za asembler iz korenog foldera Assembler/, a za emulator iz foldera Emulator/.

3.2 Pokretanje programa

Program se pokreće iz komandne linije. Neophodno je pozicionirati se u direktorijum u kome se nalazi izvršna verzija programa, dobijena prevođenjem, za asembler Assembler/bin/, za emulator Emulator/bin/.

Komanda za pokretanje asemblera je: **./asembler -o output.o input.s**
Ulazni fajlovi moraju se nalaziti u folderu Assembler/tests i navode se samo njihovi nazivi, ne cela putanja do njih. Takođe, pokretanjem asemblera, izlazni output.o fajl bice formiran i u Emulator/tests/ folderu.

Komanda za pokretanje emulatora je:
./emulator [-place=<section>**@**<address>**] input_1.o input_2.o . . . input_n.o**

4 Testovi

4.1 Testovi za asembler

array.s:

```
.extern data_out, to_digits
.global _start

.equ n, array_end-array_start

.section data
array_start:
        .word 0x0001, 0x0002, 0x0003, 0x0004, 0x0005
        .word 0x0006, 0x0007, 0x0008, 0x0009, 0x000a
        .word 0x000b, 0x000c, 0x000d, 0x000e, 0x000f
        .word 0x0010, 0x0011, 0x0012, 0x0013, 0x0014
        .word 0x0015, 0x0016, 0x0017, 0x0018, 0x0019
        .word 0x001a, 0x001b, 0x001c, 0x001d, 0x001e
        .word 0x001f, 0x0020, 0x0021, 0x0022, 0x0023
        .word 0x0024, 0x0025, 0x0026, 0x0027, 0x0028
array_end: .skip 1

.section text
_start:
        mov $n, %r1
        cmp $0, %r1
        jeq end

loop:
        mov %r1, %r3
        add $array_start, %r3
        sub $2, %r3
        add (%r3), %r2

        push %r1
        push %r2
        mov (%r3), %r2
        call to_digits

        pop %r2
        pop %r1

        sub $2, %r1
        cmp $0, %r1
        jeq jump_plus
        mov $0x2b, data_out
jump_plus:
        cmp $0, %r1
        jne loop

end:
        mov $0xa, data_out
        mov $0x3d, data_out
        mov $0x20, data_out
        call to_digits
        halt
.end
```

=> symbol table

# name	sec	scope	value	num	size
data	1	local	00000000	1	0051
text	2	local	00000000	2	0065
data_out	0	global	00000000	3	
to_digits	0	global	00000000	4	
_start	2	global	00000000	5	
# n	0	local	00000050	6	
array_start	1	local	00000000	7	
array_end	1	local	00000050	8	
loop	2	local	0000000e	9	
jump_plus	2	local	00000042	10	
end	2	local	0000004b	11	

=> classification index table

# symbol	section	index	expression
n:160		1	0

=> .data

```
01 00 02 00 03 00 04 00 05 00 06 00 07 00 08 00 09 00 0a 00
0b 00 0c 00 0d 00 0e 00 0f 00 10 00 11 00 12 00 13 00 14 00
15 00 16 00 17 00 18 00 19 00 1a 00 1b 00 1c 00 1d 00 1e 00
1f 00 20 00 21 00 22 00 23 00 24 00 25 00 26 00 27 00 28 00
00
```

=> .text

```
64 00 50 00 22 8c 00 00 00 22 34 00 4b 00 64 22 26 6c 00 00
00 26 74 00 02 00 26 6c 46 24 4c 22 4c 24 64 46 24 24 00 00
00 54 24 54 22 74 00 02 00 22 8c 00 00 00 22 34 00 42 00 64
00 2b 00 80 00 00 8c 00 00 00 22 3c 00 0e 00 64 00 0a 00 80
00 00 64 00 3d 00 80 00 00 64 00 20 00 80 00 00 24 00 00 00
04
```

=> .rel .text

# offset	relType	ref	sign	size
0000000c	R_386_16	2	+	2
00000013	R_386_16	1	+	2
00000027	R_386_16	4	+	2
00000039	R_386_16	2	+	2
00000040	R_386_16	3	+	2
00000049	R_386_16	2	+	2
00000050	R_386_16	3	+	2
00000057	R_386_16	3	+	2
0000005e	R_386_16	3	+	2
00000062	R_386_16	4	+	2

digits.s:

```
.extern data_out
.global to_digits

.section nums
to_digits:
        mov $0, %r1
        mov $0, %r3
        mov $0, %r4
        mov $0, %r5

        cmp $10000, %r2
        jgt d
        cmp $1000, %r2
        jgt f
        cmp $100, %r2
        jgt h
        cmp $10, %r2
        jgt k
        jeq k
        jmp p

d:      mov %r2, %r1
        div $10000, %r1
        mov %r1, %r3

        add $0x30, %r1
        mov %r1, data_out

f:      mul $10000, %r3
        sub %r3, %r2
        mov %r2, %r3
        div $1000, %r3
        mov %r3, %r4

        add $0x30, %r3
        mov %r3, data_out

h:      mul $1000, %r4
        sub %r4, %r2
        mov %r2, %r4
        div $100, %r4
        mov %r4, %r5

        add $0x30, %r4
        mov %r4, data_out

k:      mul $100, %r5
        sub %r5, %r2
        mov %r2, %r5
        div $10, %r5
        mov %r5, %r1

        add $0x30, %r5
        mov %r5, data_out

p:      mul $10, %r1
        sub %r1, %r2
        add $0x30, %r2
        mov %r2, data_out
        ret
.end
```

=> symbol table

# name	sec	scope	value	num	size
nums	1	local	00000000	1	00bf
data_out	0	global	00000000	2	
to_digits	1	global	00000000	3	
d	1	local	00000040	4	
f	1	local	00000055	5	
h	1	local	00000072	6	
k	1	local	0000008f	7	
p	1	local	000000ac	8	

=> classification index table

# symbol	section	index	expression

=> .nums

```
64 00 00 00 22 64 00 00 00 26 64 00 00 00 28 64 00 00 00 2a
8c 00 10 27 24 44 00 40 00 8c 00 e8 03 24 44 00 55 00 8c 00
64 00 24 44 00 72 00 8c 00 0a 00 24 44 00 8f 00 34 00 8f 00
2c 00 ac 00 64 24 22 84 00 10 27 22 64 22 26 6c 00 30 00 22
64 22 80 00 00 7c 00 10 27 26 74 26 24 64 24 26 84 00 e8 03
26 64 26 28 6c 00 30 00 26 64 26 80 00 00 7c 00 e8 03 28 74
28 24 64 24 28 84 00 64 00 28 64 28 2a 6c 00 30 00 28 64 28
80 00 00 7c 00 64 00 2a 74 2a 24 64 24 2a 84 00 0a 00 2a 64
2a 22 6c 00 30 00 2a 64 2a 80 00 00 7c 00 0a 00 22 74 22 24
6c 00 30 00 24 64 24 80 00 00 14
```

=> .rel .nums

# offset	relType	ref	sign	size
0000001b	R_386_16	1	+	2
00000024	R_386_16	1	+	2
0000002d	R_386_16	1	+	2
00000036	R_386_16	1	+	2
0000003a	R_386_16	1	+	2
0000003e	R_386_16	1	+	2
00000053	R_386_16	2	+	2
00000070	R_386_16	2	+	2
0000008d	R_386_16	2	+	2
000000aa	R_386_16	2	+	2
000000bc	R_386_16	2	+	2

main.s:

```
.extern add_func, sub_func, div_func, mul_func, data_in,
data_out, to_digits
.global _start

.section main_sec
_start:
        mov $0x00, %r3
        mov $0x00, %r4
        mov $0x2000, %psw
        mov $terminal, 0x0006
        mov $0x2000, %sp

        loop: jmp loop
        halt

.section term_sec
terminal:
        cmp $0x2b, data_in
        jeq instr

        cmp $0x2d, data_in
        jeq instr

        cmp $0x2a, data_in
        jeq instr

        cmp $0x2f, data_in
        jeq instr

        cmp $0x3d, data_in
        jeq equal

        # ako nisu cifre ignorisi
        cmp $0x2f, data_in
        jgt ok
return: iret
ok:     cmp $0x39, data_in
        jgt return

        sub $0x30, data_in
        mul $10, %r4
        add data_in, %r4
        iret

equal:  cmp $0x2b, %r3
        jne mm
        call add_func

mm:     cmp $0x2d, %r3
        jne nn
        call sub_func

nn:     cmp $0x2a, %r3
        jne tt
        call mul_func

tt:     cmp $0x2f, %r3
        jne pp
        call div_func
```

=> symbol table

#	name	sec	scope	value	num	size
	main_sec	1	local	00000000	1	0020
	term_sec	2	local	00000000	2	012d
	add_func	0	global	00000000	3	
	sub_func	0	global	00000000	4	
	div_func	0	global	00000000	5	
	mul_func	0	global	00000000	6	
	data_in	0	global	00000000	7	
	data_out	0	global	00000000	8	
	to_digits	0	global	00000000	9	
	_start	1	global	00000000	10	
	loop	1	local	0000001b	11	
	terminal	2	local	00000000	12	
	return	2	local	00000042	13	
	ok	2	local	00000043	14	
	equal	2	local	00000060	15	
	mm	2	local	0000006d	16	
	nn	2	local	0000007a	17	
	tt	2	local	00000087	18	
	pp	2	local	00000094	19	
	negativan	2	local	000000a8	20	
	pozitivan	2	local	000000ba	21	
	instr	2	local	000000bf	22	
	add	2	local	000000f1	23	
	sub	2	local	00000100	24	
	mul	2	local	0000010f	25	
	div	2	local	0000011e	26	

=> classification index table

#	symbol	section	index	expression

=> .main_sec

64 00 00 00 26 64 00 00 00 28 64 00 00 20 3e 64 00 00 00 80 06 00 64 00 00 20 2c 2c 00 1b 00 04

=> .term_sec

8c 00 2b 00 80 00 00 34 00 bf 00 8c 00 2d 00 80 00 00 34 00 bf 00 8c 00 2a 00 80 00 00 34 00 bf 00 8c 00 2f 00 80 00 00 34 00 bf 00 8c 00 3d 00 80 00 00 34 00 60 00 8c 00 2f 00 80 00 00 44 00 43 00 0c 8c 00 39 00 80 00 00 44 00 42 00 74 00 30 00 80 00 00 7c 00 0a 00 28 6c 80 00 00 28 0c 8c 00 2b 00 26 3c 00 6d 00 24 00 00 00 8c 00 2d 00 26 3c 00 7a 00 24 00 00 00 8c 00 2a 00 26 3c 00 87 00 24 00 00 00 8c 00 2f 00 26 3c 00 94 00 24 00 00 00 64 00 20 00 80 00 00 8c 00 00 00 24 34 00 ba 00 44 00 ba 00 64 00 2d 00 80 00 00 64 00 ff ff 26 ac 26 24 74 26 24 24 00 00 00 04 8c 00 2b 00 26 34 00 f1 00 8c 00 2d 00 26 34 00 00 01 8c 00 2a 00 26 34 00 0f 01 8c 00 2f 00 26 34 00 1e 01 64 80 00 00 26 64 28 24 64 00 00 00 28 0c 24 00 00 00 64 00 00 00 28 64 80 00 00 26 0c 24 00 00 00 64 00 00 00 28 64 80 00 00 26 0c 24 00 00 00 64 00 00 00 28 64 80 00 00 26 0c 24 00 00 00 64 00 00 00 28 64 80 00 00 26 0c

```
pp:         mov $0x20, data_out
            cmp $0, %r2
            jeq pozitivan
            jgt pozitivan

negativan:
            mov $0x2d, data_out
            mov $0xffff, %r3
            xor %r3, %r2
            sub %r3, %r2

pozitivan:
            call to_digits
            halt

instr:
            cmp $0x2b, %r3
            jeq add

            cmp $0x2d, %r3
            jeq sub

            cmp $0x2a, %r3
            jeq mul

            cmp $0x2f, %r3
            jeq div

            mov data_in, %r3
            mov %r4, %r2
            mov $0, %r4
            iret

add:        call add_func
            mov $0, %r4
            mov data_in, %r3
            iret

sub:        call sub_func
            mov $0, %r4
            mov data_in, %r3
            iret

mul:        call mul_func
            mov $0, %r4
            mov data_in, %r3
            iret

div:        call div_func
            mov $0, %r4
            mov data_in, %r3
            iret

.end
```

=> .rel .main_sec

# offset	relType	ref	sign	size
00000011	R_386_16	2	+	2
0000001d	R_386_16	1	+	2

=> .rel .term_sec

# offset	relType	ref	sign	size
00000005	R_386_16	7	+	2
00000009	R_386_16	2	+	2
00000010	R_386_16	7	+	2
00000014	R_386_16	2	+	2
0000001b	R_386_16	7	+	2
0000001f	R_386_16	2	+	2
00000026	R_386_16	7	+	2
0000002a	R_386_16	2	+	2
00000031	R_386_16	7	+	2
00000035	R_386_16	2	+	2
0000003c	R_386_16	7	+	2
00000040	R_386_16	2	+	2
00000048	R_386_16	7	+	2
0000004c	R_386_16	2	+	2
00000053	R_386_16	7	+	2
0000005c	R_386_16	7	+	2
00000067	R_386_16	2	+	2
0000006b	R_386_16	3	+	2
00000074	R_386_16	2	+	2
00000078	R_386_16	4	+	2
00000081	R_386_16	2	+	2
00000085	R_386_16	6	+	2
0000008e	R_386_16	2	+	2
00000092	R_386_16	5	+	2
00000099	R_386_16	8	+	2
000000a2	R_386_16	2	+	2
000000a6	R_386_16	2	+	2
000000ad	R_386_16	8	+	2
000000bc	R_386_16	9	+	2
000000c6	R_386_16	2	+	2
000000cf	R_386_16	2	+	2
000000d8	R_386_16	2	+	2
000000e1	R_386_16	2	+	2
000000e5	R_386_16	7	+	2
000000f3	R_386_16	3	+	2
000000fc	R_386_16	7	+	2
00000102	R_386_16	4	+	2
0000010b	R_386_16	7	+	2
00000111	R_386_16	6	+	2
0000011a	R_386_16	7	+	2
00000120	R_386_16	5	+	2
00000129	R_386_16	7	+	2

op.s:

```
.global add_func, sub_func, div_func, mul_func
.section text

add_func:
        add %r4, %r2
        ret

sub_func:
        sub %r4, %r2
        ret

div_func:
        cmp $0, %r4
        jeq end
        div %r4, %r2
        ret
end: int 1

mul_func:
        mul %r4, %r2
        ret



.end
```

```
=> symbol table
# name          sec  scope   value     num   size
  text            1    local   00000000  1     001d
  add_func        1    global  00000000  2
  sub_func        1    global  00000004  3
  div_func        1    global  00000008  4
  mul_func        1    global  00000019  5
  end             1    local   00000015  6


=> classification index table
# symbol          section   index   expression


=> .text
6c 28 24 14 74 28 24 14 8c 00 00 00 28 34 00 15 00 84 28 24
14 1c 00 01 00 7c 28 24 14


=> .rel .text
#  offset       relType      ref   sign   size
   0000000f  R_386_16      1     +     2
```

print_num.s:

```
.extern data_out
.global print_num

.section nums
print_num:
        push %r1
        push %r2
        push %r3
        push %r4
        push %r5
        mov %sp, %r1
        add $13, %r1                    #adresa value
        mov (%r1), %r2

        mov $0, %r1
        mov $0, %r3
        mov $0, %r4
        mov $0, %r5

        cmp $10000, %r2
        jgt d
        cmp $1000, %r2
        jgt f
        cmp $100, %r2
        jgt h
        cmp $10, %r2
        jgt k
        jeq k
        jmp p
```

```
=> symbol table
# name          sec  scope   value     num   size
  nums            1    local   00000000  1     00de
  data_out        0    global  00000000  2
  print_num       1    global  00000000  3
  d               1    local   00000055  4
  f               1    local   0000006a  5
  h               1    local   00000087  6
  k               1    local   000000a4  7
  p               1    local   000000c1  8


=> classification index table
# symbol          section   index   expression


=> .nums
4c 22 4c 24 4c 26 4c 28 4c 2a 64 2c 22 6c 00 0d 00 22 64 42
24 64 00 00 00 22 64 00 00 00 26 64 00 00 00 28 64 00 00 00
2a 8c 00 10 27 24 44 00 55 00 8c 00 e8 03 24 44 00 6a 00 8c
00 64 00 24 44 00 87 00 8c 00 0a 00 24 44 00 a4 00 34 00 a4
00 2c 00 c1 00 64 24 22 84 00 10 27 22 64 22 26 6c 00 30 00
22 64 22 80 00 00 7c 00 10 27 26 74 26 24 64 24 26 84 00 e8
03 26 64 26 28 6c 00 30 00 26 64 26 80 00 00 7c 00 e8 03 28
74 28 24 64 24 28 84 00 64 00 28 64 28 2a 6c 00 30 00 28 64
28 80 00 00 7c 00 64 00 2a 74 2a 24 64 24 2a 84 00 0a 00 2a
64 2a 22 6c 00 30 00 2a 64 2a 80 00 00 7c 00 0a 00 22 74 22
24 6c 00 30 00 24 64 24 80 00 00 54 2a 54 28 54 26 54 24 54
22 14
```

```
d:      mov %r2, %r1
        div $10000, %r1
        mov %r1, %r3

        add $0x30, %r1
        mov %r1, data_out

f:      mul $10000, %r3
        sub %r3, %r2
        mov %r2, %r3
        div $1000, %r3
        mov %r3, %r4

        add $0x30, %r3
        mov %r3, data_out

h:      mul $1000, %r4
        sub %r4, %r2
        mov %r2, %r4
        div $100, %r4
        mov %r4, %r5

        add $0x30, %r4
        mov %r4, data_out

k:      mul $100, %r5
        sub %r5, %r2
        mov %r2, %r5
        div $10, %r5
        mov %r5, %r1

        add $0x30, %r5
        mov %r5, data_out

p:      mul $10, %r1
        sub %r1, %r2

        add $0x30, %r2
        mov %r2, data_out

        pop %r5
        pop %r4
        pop %r3
        pop %r2
        pop %r1

        ret
.end
```

```
=> .rel .nums
# offset      relType      ref  sign  size
  00000030 R_386_16     1    +    2
  00000039 R_386_16     1    +    2
  00000042 R_386_16     1    +    2
  0000004b R_386_16     1    +    2
  0000004f R_386_16     1    +    2
  00000053 R_386_16     1    +    2
  00000068 R_386_16     2    +    2
  00000085 R_386_16     2    +    2
  000000a2 R_386_16     2    +    2
  000000bf R_386_16     2    +    2
  000000d1 R_386_16     2    +    2
```

sort.s:

```
.extern print_num, data_in, data_out
.global _start
.equ array, 0x0500

.section text
_start:
        mov $0x2000, %psw          #maskiran timer
        mov $terminal, 0x0006      #interrupt za terminal
        mov $0x2000, %sp
        mov $0, %r0                #r0 za size
        loop: jmp loop
        halt

.section term_sec
terminal:
        cmp $0x20, data_in
        jeq save_num

        cmp $0x3d, data_in
        jeq save_num

        cmp $0x2f, data_in
        jgt ok
return: int 1

ok:     cmp $0x39, data_in
        jgt return

        sub $0x30, data_in
        mul $10, %r2
        add data_in, %r2
        iret

save_num:
        cmp $0, %r2
        jeq ignore
        mov %r2, array(%r0)
        mov $0, %r2
        add $2, %r0
        cmp $0x3d, data_in
        jeq sort

ignore: cmp $0x3d, data_in
        jeq sort
        iret

sort:   mov %r0, %r1
        mov %r0, %r2
        add $array, %r1     #u r1 pocetak sortiranog niza
        mov %r1, %r5
        mov %r0, %r4
        sub $2, %r0         #r0 na poslednji elem niza

for:    call find_min
        mov %r3, (%r1)
        cmp $0, %r0
        jeq print
        sub $2, %r0
        add $2, %r1
        jmp for
```

=> symbol table

# name	sec	scope	value	num	size
text	1	local	00000000	1	001b
term_sec	2	local	00000000	2	0131
print_num	0	global	00000000	3	
data_in	0	global	00000000	4	
data_out	0	global	00000000	5	
_start	1	global	00000000	6	
# array	0	local	00000500	7	
loop	1	local	00000016	8	
terminal	2	local	00000000	9	
return	2	local	00000021	10	
ok	2	local	00000025	11	
save_num	2	local	00000042	12	
ignore	2	local	00000065	13	
sort	2	local	00000071	14	
for	2	local	00000087	15	
print	2	local	000000a5	16	
printloop	2	local	000000cb	17	
end_func	2	local	000000ef	18	
find_min	2	local	000000f0	19	
doloop	2	local	000000fe	20	
new_min	2	local	0000010b	21	
cond	2	local	00000113	22	
end	2	local	00000125	23	

=> classification index table

# symbol	section	index	expression

=> .text

```
64 00 00 20 3e 64 00 00 00 80 06 00 64 00 00 20 2c 64 00 00
00 20 2c 00 16 00 04
```

=> .term_sec

```
8c 00 20 00 80 00 00 34 00 42 00 8c 00 3d 00 80 00 00 34 00
42 00 8c 00 2f 00 80 00 00 44 00 25 00 1c 00 01 00 8c 00 39
00 80 00 00 44 00 21 00 74 00 30 00 80 00 00 7c 00 0a 00 24
6c 80 00 00 24 0c 8c 00 00 00 24 34 00 65 00 64 24 60 00 05
64 00 00 00 24 6c 00 02 00 20 8c 00 3d 00 80 00 00 34 00 71
00 8c 00 3d 00 80 00 00 34 00 71 00 0c 64 20 22 64 20 24 6c
00 00 05 22 64 22 2a 64 20 28 74 00 02 00 20 24 00 f0 00 64
26 42 8c 00 00 00 20 34 00 a5 00 74 00 02 00 20 6c 00 02 00
22 2c 00 87 00 64 00 53 00 80 00 00 64 00 4f 00 80 00 00 64
00 52 00 80 00 00 64 00 54 00 80 00 00 64 00 20 00 80 00 00
6c 2a 24 4c 4a 64 4a 22 4c 22 24 00 00 00 54 22 64 00 20 00
80 00 00 6c 00 02 00 2a 8c 2a 24 34 00 ef 00 2c 00 cb 00 04
4c 22 4c 28 64 00 fe 7f 26 74 00 02 00 28 8c 68 00 05 26 44
00 0b 01 2c 00 13 01 64 68 00 05 26 64 28 22 8c 00 00 00 28
34 00 25 01 74 00 02 00 28 2c 00 fe 00 64 00 ff 7f 62 00 05
54 28 54 22 14
```

=> .rel .text

# offset	relType	ref	sign	size
00000007	R_386_16	2	+	2
00000018	R_386_16	1	+	2

```
print:      #mov $0x20, data_out
            mov $0x53, data_out
            mov $0x4f, data_out
            mov $0x52, data_out
            mov $0x54, data_out
            mov $0x20, data_out
            #od r5 do r2
            add %r5, %r2

printloop:push (%r5)
            mov (%r5), %r1
            push %r1
            call print_num
            pop %r1
            mov $0x20, data_out
            add $2, %r5
            cmp %r5, %r2
            jeq end_func
            jmp printloop
end_func: halt

find_min: push %r1
            push %r4
            mov $0x7ffe, %r3    #u r3 min
            sub $2, %r4

doloop:     cmp array(%r4), %r3
            jgt new_min
            jmp cond

new_min: mov array(%r4), %r3            #u r3 min
            mov %r4, %r1                #u r1 index min

cond:       cmp $0, %r4
            jeq end
            sub $2, %r4
            jmp doloop

end:        mov $0x7fff, array(%r1)
            pop %r4
            pop %r1
            ret
.end
```

```
=> .rel .term_sec
# offset      relType      ref  sign  size
  00000005 R_386_16     4    +    2
  00000009 R_386_16     2    +    2
  00000010 R_386_16     4    +    2
  00000014 R_386_16     2    +    2
  0000001b R_386_16     4    +    2
  0000001f R_386_16     2    +    2
  0000002a R_386_16     4    +    2
  0000002e R_386_16     2    +    2
  00000035 R_386_16     4    +    2
  0000003e R_386_16     4    +    2
  00000049 R_386_16     2    +    2
  0000005f R_386_16     4    +    2
  00000063 R_386_16     2    +    2
  0000006a R_386_16     4    +    2
  0000006e R_386_16     2    +    2
  00000089 R_386_16     2    +    2
  00000095 R_386_16     2    +    2
  000000a3 R_386_16     2    +    2
  000000aa R_386_16     5    +    2
  000000b1 R_386_16     5    +    2
  000000b8 R_386_16     5    +    2
  000000bf R_386_16     5    +    2
  000000c6 R_386_16     5    +    2
  000000d4 R_386_16     3    +    2
  000000dd R_386_16     5    +    2
  000000e9 R_386_16     2    +    2
  000000ed R_386_16     2    +    2
  00000105 R_386_16     2    +    2
  00000109 R_386_16     2    +    2
  0000011a R_386_16     2    +    2
  00000123 R_386_16     2    +    2
```

timer.s:

```
.global _start
.extern data_out, data_in, timer_cfg, print_num

.section timer
_start:
.equ nula, 0
        mov $0x2155, %sp
        mov $ivt_terminal, 0x0006
        mov $ivt_timer, 0x0004
        mov $0x0005, timer_cfg
        mov $0, %r0
for:    jmp for

ivt_timer:
        add $1, %r0
        mov $0xa, data_out
        push %r0
        call print_num
        pop %r0
        iret

ivt_terminal:
        halt
        push $5
        call print_num
        pop %r5
        iret
.end
```

=> symbol table

# name	sec	scope	value	num	size
timer	1	local	00000000	1	0044
_start	1	global	00000000	2	
data_out	0	global	00000000	3	
data_in	0	global	00000000	4	
timer_cfg	0	global	00000000	5	
print_num	0	global	00000000	6	
# nula	1	local	00000000	7	
for	1	local	0000001f	8	
ivt_timer	1	local	00000023	9	
ivt_terminal	1	local	00000038	10	

=> classification index table

# symbol	section	index	expression

=> .timer

```
64 00 55 21 2c 64 00 38 00 80 06 00 64 00 23 00 80 04 00 64
00 05 00 80 00 00 64 00 00 00 20 2c 00 1f 00 6c 00 01 00 20
64 00 0a 00 80 00 00 4c 20 24 00 00 00 54 20 0c 04 4c 00 05
00 24 00 00 00 54 2a 0c
```

=> .rel .timer

# offset	relType	ref	sign	size
00000007	R_386_16	1	+	2
0000000e	R_386_16	1	+	2
00000018	R_386_16	5	+	2
00000021	R_386_16	1	+	2
0000002d	R_386_16	3	+	2
00000033	R_386_16	6	+	2
0000003f	R_386_16	6	+	2

print_num.s:

```
.extern _start
.global ivt_0, ivt_1, ivt_2, ivt_3, ivt_4, ivt_5, ivt_6, ivt_7

.global timer_cfg, data_out, data_in
.equ data_out, 0xff00
.equ data_in, 0xff02
.equ timer_cfg, 0xff10

.section ivt

ivt_0:  push $_start
        iret

ivt_1:  mov $0x45, 0xff00
        mov $0x52, 0xff00
        mov $0x52, 0xff00
        mov $0x4f, 0xff00
        mov $0x52, 0xff00
        mov $0x5f, 0xff00
        mov $0x69, 0xff00
        mov $0x76, 0xff00
        mov $0x74, 0xff00
        mov $0x31, 0xff00
        mov $0x0a, 0xff00
        halt

ivt_2:  iret
ivt_3:  iret
ivt_4:  iret
ivt_5:  iret
ivt_6:  iret
ivt_7:  iret

.end
```

=> symbol table

#	name	sec	scope	value	num	size
	ivt	1	local	00000000	1	0059
	_start	0	global	00000000	2	
	ivt_0	1	global	00000000	3	
	ivt_1	1	global	00000005	4	
	ivt_2	1	global	00000053	5	
	ivt_3	1	global	00000054	6	
	ivt_4	1	global	00000055	7	
	ivt_5	1	global	00000056	8	
	ivt_6	1	global	00000057	9	
	ivt_7	1	global	00000058	10	
~	timer_cfg	0	global	0000ff10	11	
~	data_out	0	global	0000ff00	12	
~	data_in	0	global	0000ff02	13	

=> classification index table

#	symbol	section	index	expression

=> .ivt

```
4c 00 00 00 0c 64 00 45 00 80 00 ff 64 00 52 00 80 00 ff 64
00 52 00 80 00 ff 64 00 4f 00 80 00 ff 64 00 52 00 80 00 ff
64 00 5f 00 80 00 ff 64 00 69 00 80 00 ff 64 00 76 00 80 00
ff 64 00 74 00 80 00 ff 64 00 31 00 80 00 ff 64 00 0a 00 80
00 ff 04 0c 0c 0c 0c 0c 0c
```

=> .rel .ivt

#	offset	relType	ref	sign	size
	00000002	R_386_16	2	+	2

bomb.s:

```
.extern data_in, data_out, timer_cfg, to_digits
.global _start

.section text

_start:
        mov $0x00, %r0
        mov $0x2000, %psw
        mov $term, 0x0006
        mov $timer, 0x0004
        mov $0x2000, %sp
loop:
        cmp $1, %r4
        jne loop

        mov $0x0001, 0xff10
        mov $0x4000, %psw
        mov %r0, %r1
        div $60,  %r0
        mov $60, %r2
        mul %r0, %r2
        sub %r2, %r1

wait: jmp wait
        halt

.section terminal
term:
        cmp $0x3d, data_in
        jeq count

        # ako nisu cifre greska
        cmp $0x2f, data_in
        jgt ok
return: int 1
ok:     cmp $0x39, data_in
        jgt return

        sub $0x30, data_in
        mul $10, %r0
        add data_in, %r0
        iret
count:  mov $1, %r4
        iret
timer:
        push %r0
        push %r1
        mov %r0, %r2
        call to_digits
        pop %r1
        pop %r0

        mov $0x3a, data_out(%pc)

        push %r0
        push %r1
        mov %r1, %r2
        call to_digits
        pop %r1
        pop %r0
```

=> symbol table

# name	sec	scope	value	num	size
text	1	local	00000000	1	004a
terminal	2	local	00000000	2	00be
data_in	0	global	00000000	3	
data_out	0	global	00000000	4	
timer_cfg	0	global	00000000	5	
to_digits	0	global	00000000	6	
_start	1	global	00000000	7	
loop	1	local	0000001d	8	
wait	1	local	00000045	9	
term	2	local	00000000	10	
return	2	local	00000016	11	
ok	2	local	0000001a	12	
count	2	local	00000037	13	
timer	2	local	0000003d	14	
mins	2	local	00000078	15	
end	2	local	00000093	16	

=> classification index table

# symbol	section	index	expression

=> .text

```
64 00 00 00 20 64 00 00 20 3e 64 00 00 00 80 06 00 64 00 3d00 80
04 00 64 00 00 20 2c 8c 00 01 00 28 3c 00 1d 00 64 00 01 00 80 10
ff 64 00 00 40 3e 64 20 22 84 00 3c 00 20 64 00 3c 00 24 7c 20 24
74 24 22 2c 00 45 00 04
```

=> .terminal

```
8c 00 3d 00 80 00 00 34 00 37 00 8c 00 2f 00 80 00 00 44 00 1a 00
1c 00 01 00 8c 00 39 00 80 00 00 44 00 16 00 74 00 30 00 80 00 00
7c 00 0a 00 20 6c 80 00 00 20 0c 64 00 01 00 28 0c 4c 20 4c 22 64
20 24 24 00 00 00 54 22 54 20 64 00 3a 00 6e fe ff 4c 20 4c 22 64
22 24 24 00 00 00 54 22 54 20 8c 00 00 00 22 34 00 78 00 74 00 01
00 22 64 00 0a 00 6e fe ff 0c 8c 00 00 00 20 34 00 93 00 64 00 3b
00 22 74 00 01 00 20 64 00 0a 00 6e fe ff 0c 64 00 0a 00 6e fe ff 64
00 42 00 6e fe ff 64 00 4f 00 6e fe ff 64 00 4f 00 6e fe ff 64 00 4d
00 6e fe ff 64 00 0a 00 6e fe ff 04
```

=> .rel .text

# offset	relType	ref	sign	size
0000000c	R_386_16	2	+	2
00000013	R_386_16	2	+	2
00000024	R_386_16	1	+	2
00000047	R_386_16	1	+	2

=> .rel .terminal

# offset	relType	ref	sign	size
00000005	R_386_16	3	+	2
00000009	R_386_16	2	+	2
00000010	R_386_16	3	+	2
00000014	R_386_16	2	+	2
0000001f	R_386_16	3	+	2
00000023	R_386_16	2	+	2
0000002a	R_386_16	3	+	2
00000033	R_386_16	3	+	2
00000046	R_386_16	6	+	2
00000051	R_386_PC16	4	+	2

```
            cmp $0, %r1
            jeq mins
            sub $1, %r1
            mov $0x0a, data_out(%pc)
            iret

mins:
            cmp $0, %r0
            jeq end
            mov $59, %r1
            sub $1, %r0


            mov $0x0a, data_out(%pc)
            iret

end:
            mov $0x0a, data_out(%pc)
            mov $0x42, data_out(%pc)
            mov $0x4f, data_out(%pc)
            mov $0x4f, data_out(%pc)
            mov $0x4d, data_out(%pc)
            mov $0x0a, data_out(%pc)
            halt
.end
```

```
0000005c R_386_16     6    +    2
00000069 R_386_16     2    +    2
00000075 R_386_PC16   4    +    2
0000007f R_386_16     2    +    2
00000090 R_386_PC16   4    +    2
00000098 R_386_PC16   4    +    2
0000009f R_386_PC16   4    +    2
000000a6 R_386_PC16   4    +    2
000000ad R_386_PC16   4    +    2
000000b4 R_386_PC16   4    +    2
000000bb R_386_PC16   4    +    2
```

4.1 Testovi za emulator

***Svi testovi su pokretani bez –place komande

<table>
<tr><td colspan="2">test 1</td><td>Sortiranje niza koji zadaje korisnik</td></tr>
<tr><td colspan="2">perifrerija</td><td>Terminal</td></tr>
<tr><td colspan="2">ulazni fajlovi</td><td>sort.o print_num.o</td></tr>
<tr><td rowspan="3">izlaz emulatora</td><td>linker</td><td># symbol table
name sec scope value num
ivt_0 1 global 00000010 4
ivt_1 1 global 00000015 5
ivt_2 1 global 00000063 6
ivt_3 1 global 00000064 7
ivt_4 1 global 00000065 8
ivt_5 1 global 00000066 9
ivt_6 1 global 00000067 10
ivt_7 1 global 00000068 11
timer_cfg 0 global 0000ff10 12
data_out 0 global 0000ff00 13
data_in 0 global 0000ff02 14
_start 1 global 00000069 15
print_num 1 global 000001b5 16

header table
num segment_name value size
0001 ivt 0010 0059
0002 text 0069 0074
0003 term_sec 0084 01a5
0004 nums 01b5 0283

segment 0001
4c 00 69 00 0c 64 00 45 00 80 00 ff 64 00 52 00 80 00 ff 64 00 52 00 80 00 ff 64 00 4f 00 80 00 ff 64 00 52 00 80 00 ff 64 00 5f 00 80 00 ff 64 00 69 00 80 00 ff 64 00 76 00 80 00 ff 64 00 74 00 80 00 ff 64 00 31 00 80 00 ff 64 00 0a 00 80 00 ff 04 0c 0c 0c 0c 0c

segment 0002
64 00 00 20 3e 64 00 84 00 80 06 00 64 00 00 20 2c 64 00 00 00 20 2c 00 7f 00 04

segment 0003
8c 00 20 00 80 02 ff 34 00 c6 00 8c 00 3d 00 80 02 ff 34 00 c6 00 8c 00 2f 00 80 02 ff 44 00 a9 00 1c 00 01 00 8c 00 39 00 80 02 ff 44 00 a5 00 74 00 30 00 80 02 ff 7c 00 0a 00 24 6c 80 02 ff 24 0c 8c 00 00 00 24 34 00 e9 00 64 24 60 00 05 64 00 00 00 24 6c 00 02 00 20 8c 00 3d 00 80 02 ff 34 00 f5 00 8c 00 3d 00 80 02 ff 34 00 f5 00 0c 64 20 22 64 20 24 6c 00 00 05 22 64 22 2a 64 20 28 74 00 02 00 24 00 74 01 64 26 42 8c 00 00 00 20 34 00 29 01 74 00 02 00 20 6c 00 02 00 22 2c 00 0b 01 64 00 53 00 80 00 ff 64 00 4f 00 80 00 ff 64 00 52 00 80 00 ff 64 00 54 00 80 00 ff 64 00 20 00 80 00 ff 6c 2a 24 4c 4a 64 4a 22 4c 22 24 00 b5 01 54 22 64 00 20 00 80 00 ff 6c 00 02 00 2a 8c 2a 24 34 00 01 2c 00 4f 01 04 4c 22 4c 28 64 00 fe 7f 26 74 00 02 00 28 8c 68 00 05 26 44 00 8f 01 2c 00 97 01 64 68 00 05 26 64 28 22 8c 00 00 00 28 34 00 a9 01 74 00 02 00 28 2c 00 82 01 64 00 ff 7f 62 00 05 54 28 54 22 14

segment 0004
4c 22 4c 24 4c 26 4c 28 4c 2a 64 2c 22 6c 00 0d 00 22 64 42 24 64 00 00 00 22 64 00 00 00 26 64 00 00 00 28 64 00 00 00 2a 8c 00 10 27 24 44 00 0a 02 8c 00 e8 03 24 44 00 1f 02 8c 00 64 00 24 44 00 3c 02 8c 00 0a 00 24 44 00 59 02 34 00 59 02 2c 00 76 02 64 24 22 84 00 10 27 22 64 22 26 6c 00 30 00 22 64 22 80 00 ff 7c 00 10 27 26 74 26 24 64 24 26 84 00 e8 03 26 64 26 28 6c 00 30 00 26 64 26 80 00 ff 7c 00 e8 03 28 74 28 24 64 24 28 84 00 64 00 28 64 28 2a 6c 00 30 00 28 64 28 80 00 ff 7c 00 64 00 2a 74 2a 24 64 24 2a 84 00 0a 00 2a 64 2a 22 6c 00 30 00 2a 64 2a 80 00 ff 7c 00 0a 00 22 74 22 24 6c 00 30 00 24 64 24 80 00 ff 54 2a 54 28 54 26 54 24 54 22 14</td></tr>
<tr><td>ulaz korisnika</td><td>14 6 8 54 21 36 57= [enter]</td></tr>
<tr><td>Izlaz programa</td><td>SORT 6 8 14 21 36 54 57</td></tr>
</table>

<table>
<tr><td colspan="2">test 2</td><td>Štoperica – timer podešen da uvećava brojač na 1 sekundu</td></tr>
<tr><td colspan="2">perifrerije</td><td>Terminal, timer</td></tr>
<tr><td colspan="2">ulazni fajlovi</td><td>timer.o print_num.o</td></tr>
<tr><td rowspan="3">izlaz emulatora</td><td>linker</td><td># symbol table
name sec scope value num
ivt_0 1 global 00000010 3
ivt_1 1 global 00000015 4
ivt_2 1 global 00000063 5
ivt_3 1 global 00000064 6
ivt_4 1 global 00000065 7
ivt_5 1 global 00000066 8
ivt_6 1 global 00000067 9
ivt_7 1 global 00000068 10
timer_cfg 0 global 0000ff10 11
data_out 0 global 0000ff00 12
data_in 0 global 0000ff02 13
_start 1 global 00000069 14
print_num 1 global 000000ad 15

header table
num segment_name value size
0001 ivt 0010 0059
0002 timer 0069 009d
0003 nums 00ad 017b

segment 0001
4c 00 69 00 0c 64 00 45 00 80 00 ff 64 00 52 00 80 00 ff 64 00 52 00 80 00 ff 64 00 4f 00 80 00 ff 64 00 52 00 80 00 ff 64 00 5f 00 80 00 ff 64 00 69 00 80 00 ff 64 00 76 00 80 00 ff 64 00 74 00 80 00 ff 64 00 31 00 80 00 ff 64 00 0a 00 80 00 ff 04 0c 0c 0c 0c 0c 0c

segment 0002
64 00 55 21 2c 64 00 a1 00 80 06 00 64 00 8c 00 80 04 00 64 00 01 00 80 10 ff 64 00 00 00 20 2c 00 88 00 6c 00 01 00 20 64 00 0a 00 80 00 ff 4c 20 24 00 ad 00 54 20 0c 04 4c 00 05 00 24 00 ad 00 54 2a 0c

segment 0003
4c 22 4c 24 4c 26 4c 28 4c 2a 64 2c 22 6c 00 0d 00 22 64 42 24 64 00 00 00 22 64 00 00 00 26 64 00 00 00 28 64 00 00 00 2a 8c 00 10 27 24 44 00 02 01 8c 00 e8 03 24 44 00 17 01 8c 00 64 00 24 44 00 34 01 8c 00 0a 00 24 44 00 51 01 34 00 51 01 2c 00 6e 01 64 24 22 84 00 10 27 22 64 22 26 6c 00 30 00 22 64 22 80 00 ff 7c 00 10 27 26 74 26 24 64 24 26 84 00 e8 03 26 64 26 28 6c 00 30 00 26 64 26 80 00 ff 7c 00 e8 03 28 74 28 24 64 24 28 84 00 64 00 28 64 28 2a 6c 00 30 00 28 64 28 80 00 ff 7c 00 64 00 2a 74 2a 24 64 24 2a 84 00 0a 00 2a 64 2a 22 6c 00 30 00 2a 64 2a 80 00 ff 7c 00 0a 00 22 74 22 24 6c 00 30 00 24 64 24 80 00 ff 54 2a 54 28 54 26 54 24 54 22 14</td></tr>
<tr><td>ulaz korisnika</td><td>Klikom na enter prekida se rad programa</td></tr>
<tr><td>Izlaz programa</td><td>1
2
3
5
6
. . .</td></tr>
</table>

<table>
<tr><td colspan="2">test 3</td><td>Kalkulator</td></tr>
<tr><td colspan="2">perifrerije</td><td>Terminal</td></tr>
<tr><td colspan="2">ulazni fajlovi</td><td>digits.o op.o main.o</td></tr>
<tr><td rowspan="2">izlaz emulatora</td><td>linker</td><td># symbol table
name sec scope value num
ivt_0 1 global 00000010 5
ivt_1 1 global 00000015 6
ivt_2 1 global 00000063 7
ivt_3 1 global 00000064 8
ivt_4 1 global 00000065 9
ivt_5 1 global 00000066 10
ivt_6 1 global 00000067 11
ivt_7 1 global 00000068 12
timer_cfg 0 global 0000ff10 13
data_out 0 global 0000ff00 14
data_in 0 global 0000ff02 15
to_digits 1 global 00000069 16
add_func 1 global 00000128 17
sub_func 1 global 0000012c 18
div_func 1 global 00000130 19
mul_func 1 global 00000141 20
_start 1 global 00000145 21

header table
num segment_name value size
0001 ivt 0010 0059
0002 nums 0069 0118
0003 text 0128 0135
0004 main_sec 0145 0155
0005 term_sec 0165 0282

segment 0001
4c 00 45 01 0c 64 00 45 00 80 00 ff 64 00 52 00 80 00 ff 64 00 52 00 80 00 ff 64 00 4f 00 80 00 ff 64 00 52 00 80 00 ff 64 00 5f 00 80 00 ff 64 00 69 00 80 00 ff 64 00 76 00 80 00 ff 64 00 74 00 80 00 ff 64 00 31 00 80 00 ff 64 00 0a 00 80 00 ff 04 0c 0c 0c 0c 0c 0c

segment 0002
64 00 00 00 22 64 00 00 00 26 64 00 00 00 28 64 00 00 00 2a 8c 00 10 27 24 44 00 a9 00 8c 00 e8 03 24 44 00 be 00 8c 00 64 00 24 44 00 db 00 8c 00 0a 00 24 44 00 f8 00 34 00 f8 00 2c 00 15 01 64 24 22 84 00 10 27 22 64 22 26 6c 00 30 00 22 64 22 80 00 ff 7c 00 10 27 26 74 26 24 64 24 26 84 00 e8 03 26 64 26 28 6c 00 30 00 26 64 26 80 00 ff 7c 00 e8 03 28 74 28 24 64 24 28 84 00 64 00 28 64 28 2a 6c 00 30 00 28 64 28 80 00 ff 7c 00 64 00 2a 74 2a 24 64 24 2a 84 00 0a 00 2a 64 2a 22 6c 00 30 00 2a 64 2a 80 00 ff 7c 00 0a 00 22 74 22 24 6c 00 30 00 24 64 24 80 00 ff 14

segment 0003
6c 28 24 14 74 28 24 14 8c 00 00 00 28 34 00 3d 01 84 28 24 14 1c 00 01 00 7c 28 24 14

segment 0004
64 00 00 00 26 64 00 00 00 28 64 00 00 20 3e 64 00 65 01 80 06 00 64 00 00 20 2c 2c 00 60 01 04

segment 0005
8c 00 2b 00 80 02 ff 34 00 24 02 8c 00 2d 00 80 02 ff 34 00 24 02 8c 00 2a 00 80 02 ff 34 00 24 02 8c 00 2f 00 80 02 ff 34 00 24 02 8c 00 3d 00 80 02 ff 34 00 c5 01 8c 00 2f 00 80 02 ff 44 00 a8 01 0c 8c 00 39 00 80 02 ff 44 00 a7 01 74 00 30 00 80 02 ff 7c 00 0a 00 28 6c 80 02 ff 28 0c 8c 00 2b 00 26 3c 00 d2 01 24 00 28 01 8c 00 2d 00 26 3c 00 df 01 24 00 2c 01 8c 00 2a 00 26 3c 00 ec 01 24 00 41 01 8c 00 2f 00 26 3c 00 f9 01 24 00 30 01 64 00 20 00 80 00 ff 8c 00 00 00 24 34 00 1f 02 44 00 1f 02 64 00 2d 00 80 00 ff 64 00 ff ff 26 ac 26 24 74 26 24 24 00 69 00 04 8c 00 2b 00 26 34 00 56 02 8c 00 2d 00 26 34 00 65 02 8c 00 2a 00 26 34 00 74 02 8c 00 2f 00 26 34 00 83 02 64 80 02 ff 26 64 28 24 64 00 00 00 28 0c 24 00 28 01 64 00 00 00 28 64 80 02 ff 26 0c 24 00 2c 01 64 00 00 00 28 64 80 02 ff 26 0c 24 00 41 01 64 00 00 00 28 64 80 02 ff 26 0c 24 00 30 01 64 00 00 00 28 64 80 02 ff 26</td></tr>
<tr><td>ulaz korisnika</td><td>14 + 5 + 18 - 9 * 2 / 4 = [enter]</td></tr>
</table>

		***Program računa operacije redom kojim se navode **14 + 5** = 19 19 **+ 18** = 37 27 **– 9** = 28 28 *** 2** = 56 56 **/ 4** = **14**
	Izlaz programa	= 14

test 4		Suma elemenata niza (vrednosti elementa od 1 do 40)
ulazni fajlovi		digits.o array.o
izlaz emulatora	linker	# symbol table # name sec scope value num ivt_0 1 global 00000010 4 ivt_1 1 global 00000015 5 ivt_2 1 global 00000063 6 ivt_3 1 global 00000064 7 ivt_4 1 global 00000065 8 ivt_5 1 global 00000066 9 ivt_6 1 global 00000067 10 ivt_7 1 global 00000068 11 timer_cfg 0 global 0000ff10 12 data_out 0 global 0000ff00 13 data_in 0 global 0000ff02 14 to_digits 1 global 00000069 15 _start 2 global 00000179 16 # header table # num segment_name value size 0001 ivt 0010 0059 0002 nums 0069 0118 0003 data 0128 0169 0004 text 0179 01ce # segment 0001 4c 00 79 01 0c 64 00 45 00 80 00 ff 64 00 52 00 80 00 ff 64 00 52 00 80 00 ff 64 00 4f 00 80 00 ff 64 00 52 00 80 00 ff 64 00 5f 00 80 00 ff 64 00 69 00 80 00 ff 64 00 76 00 80 00 ff 64 00 74 00 80 00 ff 64 00 31 00 80 00 ff 64 00 0a 00 80 00 ff 04 0c 0c 0c 0c 0c # segment 0002 64 00 00 00 22 64 00 00 00 26 64 00 00 00 28 64 00 00 00 2a 8c 00 10 27 24 44 00 a9 00 8c 00 e8 03 24 44 00 be 00 8c 00 64 00 24 44 00 db 00 8c 00 0a 00 24 44 00 f8 00 34 00 f8 00 2c 00 15 01 64 24 22 84 00 10 27 22 64 22 26 6c 00 30 00 22 64 22 80 00 ff 7c 00 10 27 26 74 26 24 64 24 26 84 00 e8 03 26 64 26 28 6c 00 30 00 26 64 26 80 00 ff 7c 00 e8 03 28 74 28 24 64 24 28 84 00 64 00 28 64 28 2a 6c 00 30 00 28 64 28 80 00 ff 7c 00 64 00 2a 74 2a 24 64 24 2a 84 00 0a 00 2a 64 2a 22 6c 00 30 00 2a 64 2a 80 00 ff 7c 00 0a 00 22 74 22 24 6c 00 30 00 24 64 24 80 00 ff 14 # segment 0003 01 00 02 00 03 00 04 00 05 00 06 00 07 00 08 00 09 00 0a 00 0b 00 0c 00 0d 00 0e 00 0f 00 10 00 11 00 12 00 13 00 14 00 15 00 16 00 17 00 18 00 19 00 1a 00 1b 00 1c 00 1d 00 1e 00 1f 00 20 00 21 00 22 00 23 00 24 00 25 00 26 00 27 00 28 00 00 # segment 0004 64 00 50 00 22 8c 00 00 00 22 34 00 c4 01 64 22 26 6c 00 28 01 26 74 00 02 00 26 6c 46 24 4c 22 4c 24 64 46 24 24 00 69 00 54 24 54 22 74 00 02 00 22 8c 00 00 00 22 34 00 bb 01 64 00 2b 00 80 00 ff 8c 00 00 00 22 3c 00 87 01 64 00 0a 00 80 00 ff 64 00 3d 00 80 00 ff 64 00 20 00 80 00 ff 24 00 69 00
	Izlaz programa	40+39+38+37+36+35+34+33+32+31+30+29+28+27+26+25+24+23+22+21+20+19+18+17+16+15+14+13+12+11+10+9+8+7+6+5+4+3+2+1 = 820

<table>
<tr><td colspan="2">test 1</td><td>Bomba</td></tr>
<tr><td colspan="2">perifrerije</td><td>Terminal, timer</td></tr>
<tr><td colspan="2">ulazni fajlovi</td><td>bomb.o digit_two.o</td></tr>
<tr><td rowspan="3">izlaz emulatora</td><td>linker</td><td># symbol table
name sec scope value num
ivt_0 1 global 00000010 4
ivt_1 1 global 00000015 5
ivt_2 1 global 00000063 6
ivt_3 1 global 00000064 7
ivt_4 1 global 00000065 8
ivt_5 1 global 00000066 9
ivt_6 1 global 00000067 10
ivt_7 1 global 00000068 11
timer_cfg 0 global 0000ff10 12
data_out 0 global 0000ff00 13
data_in 0 global 0000ff02 14
_start 1 global 00000069 15
to_digits 1 global 00000171 16

header table
num segment_name value size
0001 ivt 0010 0059
0002 text 0069 00a3
0003 terminal 00b3 0161
0004 nums 0171 0220

segment 0001
4c 00 69 00 0c 64 00 45 00 80 00 ff 64 00 52 00 80 00 ff 64 00 52 00 80 00 ff 64 00 52 00 80 00 ff 64 00 4f 00 80 00 ff 64 00 52 00 80 00 ff 64 00 5f 00 80 00 ff 64 00 69 00 80 00 ff 64 00 76 00 80 00 ff 64 00 74 00 80 00 ff 64 00 31 00 80 00 ff 64 00 0a 00 80 00 ff 04 0c 0c 0c 0c 0c 0c

segment 0002
64 00 00 00 20 64 00 00 20 3e 64 00 b3 00 80 06 00 64 00 f0 00 80 04 00 64 00 00 20 2c 8c 00 01 00 28 3c 00 86 00 64 00 02 00 80 10 ff 64 00 00 40 3e 64 20 22 84 00 3c 00 20 64 00 3c 00 24 7c 20 24 74 24 22 2c 00 ae 00 04

segment 0003
8c 00 3d 00 80 02 ff 34 00 ea 00 8c 00 2f 00 80 02 ff 44 00 cd 00 1c 00 01 00 8c 00 39 00 80 02 ff 44 00 c9 00 74 00 30 00 80 02 ff 7c 00 0a 00 20 6c 80 02 ff 20 0c 64 00 01 00 28 0c 4c 20 4c 22 64 20 24 24 00 71 01 54 22 54 20 64 00 3a 00 6e fa fd 4c 20 4c 22 64 22 24 24 00 71 01 54 22 54 20 8c 00 00 00 22 34 00 2b 01 74 00 01 00 22 64 00 0a 00 6e d6 fd 0c 8c 00 00 00 20 34 00 46 01 64 00 3b 00 22 74 00 01 00 20 64 00 0a 00 6e bb fd 0c 64 00 0a 00 6e b3 fd 64 00 42 00 6e ac fd 64 00 4f 00 6e a5 fd 64 00 4f 00 6e 9e fd 64 00 4d 00 6e 97 fd 64 00 0a 00 6e 90 fd 04

segment 0004
64 00 00 00 22 64 00 00 00 26 64 00 00 00 28 64 00 00 00 2a 8c 00 10 27 24 44 00 b1 01 8c 00 e8 03 24 44 00 c6 01 8c 00 64 00 24 44 00 e3 01 8c 00 0a 00 24 44 00 00 02 2c 00 00 02 2c 00 1d 02 64 24 22 84 00 10 27 22 64 22 26 6c 00 30 00 22 64 22 80 00 ff 7c 00 10 27 26 74 26 24 64 24 26 84 00 e8 03 26 64 26 28 6c 00 30 00 26 64 26 80 00 ff 7c 00 e8 03 28 74 28 24 64 24 28 84 00 64 00 28 64 28 2a 6c 00 30 00 28 64 28 80 00 ff 7c 00 64 00 2a 74 2a 24 64 24 2a 84 00 0a 00 2a 64 2a 22 6c 00 30 00 2a 64 2a 80 00 ff 7c 00 0a 00 22 74 22 24 6c 00 30 00 24 64 24 80 00 ff 14</td></tr>
<tr><td>ulaz korisnika</td><td>5= [enter]</td></tr>
<tr><td>Izlaz programa</td><td>00:05
00:04
00:03
00:02
00:01
00:00
BOOM</td></tr>
</table>