ETF - SISTEMSKI SOFTVER (13E113SS)

Domaći zadatak

Nikola Milić
2014/0042

September 2018.

Zadatak

Napisati dvoprolazni asembler za procesor opisan u prilogu. Ulaz asemblera je tekstualni fajl u skladu sa sintaksom opisanom u nastavku. Izlaz asemblera treba da bude predmetni program zapisan u tekstualnom fajlu. Za potrebe učitavanja u emulator dozvoljeno je generisati i binarni format zajedno sa tekstualnim. Format predmetnog programa bazirati na školskoj varijanti elf formata (tekstualni fajl kakav je korišćen u zadatku 9 u prezentaciji V3_Konstrukcija asemblera.ppt) i predložiti izmene u formatu u skladu sa potrebama ciljne arhitekture (nove sekcije, novi tipovi zapisa o relokacijama, dodatna polja u postojećim tipovima zapisa, novi podaci o predmetnom programu i sl.). Prilikom generisanja izlaznog fajla voditi se principima koje koristi GNU asembler, s tim da se sekcije smeštaju jedna za drugom, počev od adrese koja se zadaje iz komandne linije prilikom pokretanja asemblera.

Sintaksa asemblera i ostali zahtevi :

- u jednoj liniji može biti najviše jedna komanda,

- na početku svake linije može da stoji labela koja se završava dvotačkom,

- labela može da stoji i u praznoj liniji, što je ekvivalendno kao da stoji u liniji sa prvim sledećim sadržajem,

- simboli se uvoze i izvoze tako što se na početku fajla navede direktiva:

.global < ime globalnog simbola >,...

- u jednoj direktivi može da se navede i više globalnih naziva koji su odvojeni zapetama,

- izvorni kod je podeljen u sekcije:

    - .text - sekcija koja sadrži mašinski kod,
    - .data - sekcija koja sadrži inicijalizovane podatke,
    - .rodata - sekcija koja sadrži inicijalizovane podatke koje nije dozvoljeno menjati,
    - .bss - sekcija koja sadrži neinicijalizovane podatke,

- svaka sekcija se u fajlu pojavljuje najviše jednom,

- fajl sa izvornim kodom se završava direktivom .end. Ostatak fajla se odbacuje (ne prevodi se),

- pored mnemonika koji su definisani treba obezbediti direktive .char, .word, .long, .align i .skip sa istim funkcionalnostima kao u GNU asembleru,

- ostatak sintakse, ukoliko nije definisan u prilogu, definisati po sopstvenom nahođenju.

Opis procesora

U nastavku je opisan deo 16-bitnog dvoadresnog procesora sa Von-Neuman arhitekturom. Adresibilna jedinica je bajt, a raspored bajtova u reči je little-endian. Veličina adresnog prostora je 2^{16}B. Procesor poseduje 8 opštenamenskih 16-bitnih registara označenih sa r0-r7, pri čemu se r7 koristi kao PC, a r6 ka SP. Pored opštenamenskih registara postoji registar PSW (statusna reč procesora). Stek raste ka nižim adresama, a SP pokazuje na zauzetu lokaciju na vrhu steka. Statusna reč procesora PSW u najniža 4 bita sadrži flegove (redom od najnižeg bita): Z – rezultat prethodne operacije je 0, O – prekoračenje, C – prenos i N – rezultat je negativan. Najviši bit u PSW služi za globalno maskiranje prekida.

Počev od adrese 0 nalazi se IVT sa 8 ulaza. Svaki ulaz zauzima 2 bajta i sadrži adresu odgovarajuće prekidne rutine. Na ulazu 0 se nalazi adresa prekidne rutine koja se izvršava prilikom pokretanja, odnosno prilikom resetovanja procesora. Na ulazu 1 se nalazi adresa prekidne rutine koja se izvršava periodično sa periodom 1 sekund. Prekidna rutina na ulazu 1 se izvršava periodično samo ako je bit 13 u PSW registru postavljen na 1. U suprotnom se ne izvršava. Na ulazu 2 se nalazi adresa prekidne rutine koja se izvršava ako se pokuša izvršavanje nekorektne instrukcije. Na ulazu 3 se nalazi adresa prekidne rutine koja se izvršava kada se pritisne neki taster i tada je potrebno očitati kod pritisnutog tastera (potrebne adrese su prikazane u nastavku). Ostali ulazi su slobodni za korišćenje od strane programera.

Poslednjih 128 bajtova rezervisano je za periferije. Na adresi 0xFFFE nalazi se registar podataka izlazne periferije. Upisom ASCII koda na adresu 0xFFFE na ekranu se na tekućoj poziciji ispisuje odgovarajući znak, osim u slučaju upisa 0x10, kada se tekuća pozicija prebacuje na početak sledećeg reda. Na adresi 0xFFFC se nalazi registar podataka ulazne periferije. Kada se pritisne neki taster, ASCII kod pritisnutog tastera se ubacuje u ovaj registar, nakon čega sistem generiše prekid.

Sve instrukcije su veličine 2 bajta, izuzev instrukcija koje u sebi sadrže neposredni podatak ili apsolutnu ili PC relativnu adresu, kada je veličina instrukcije 4 bajta. Sve instrukcije se izvršavaju uslovno, pri čemu je uslov određen sa prva dva bita svake instrukcije (tabela sa kodovima data na kraju). Naredna 4 bita rezervisana su za operacioni kod instrukcije. Ostatak instrukcije podeljen je na dva polja po 5 bitova, dst i src. Oba polja se kodiraju na isti način, opisan u nastavku, s tim da neposredno adresiranje nije validan način adresiranja u polju dst.

mnemonik	oc	Efekat	Flegovi koji se menjaju
add	0	dst = dst op src, op je jedna od 4 operacije	zocn
sub	1		
mul	2		zn
div	3		
cmp	4	kao oduzimanje ali se ne menja dst	zocn
and	5	dst = dst & src	zn
or	6	dst = dst | src	
not	7	dst = ~src	
test	8	dst & src	
push	9	sp -= 2; mem[sp] = src	-
pop	10	dst = mem[sp]; sp += 2	
call	11	push pc; pc=src	
iret	12	pop psw; pop pc	
mov	13	dst = src, osim ako je odredište pc, tada važi pc = src	zn
shl	14	dst <<= src	zcn
shr	15	dst >>= src	zcn

Objašnjenja i dodatne napomene:

- Sve aritmetičke operacije se izvode tako da odgovaraju označenim celim brojevima.
- Instrukcije cmp i tst ne čuvaju direktni rezultat odgovrajuće operacije, već samo u skladu sa rezultatom postavljaju nove vrednosti flegova u PSW.
- asembler poseduje i pseudo instrukciju ret koja se prevodi odgovarajućom pop instrukcijom
- asembler poseduje i pseudo instrukciju jmp koja se prevodi instrukcijom mov ili add, u zavisnosti od tipa adresiranja odredišta
- uslovni kodovi su navedeni u narednoj tabeli:

eq	0	==
ne	1	!=
gt	2	> (označeno)
al	3	bezuslovno

Operandi se kodiraju tako što najviša dva bita određuju tip adresiranja. U tipovima adresiranja u kojima je potreban registar, broj registra zapisan je u preostala 3 bita operanda. U slučaju da je za adresiranje potreban i podatak koji predstavlja adresu, pomeraj ili neposrednu vrednost, zapisuje se u 2 bajta neposredno posle instrukcije. U jednoj instrukciji samo jedan operand može da zahteva 2 dodatna bajta. Instrukcije koje zahtevaju dva podatka predstavljaju grešku. Takođe, sve kombinacije instrukcija i operanada, za koje ne postoji razumno tumačenje, smatrati greškom.

00	neposredno ili PSW	ukoliko je u preostala 3 bita 7, koristi se PSW; u suprotnom, u dodatna 2 bajta zapisan je neposredni podatak
01	registarsko direktno	u preostala 3 bita zapisan je redni broj registra koji se koristi
02	memorijsko	preostala 3 bita se ne koriste, u dodatna 2 bajta zapisana je adresa
03	registarsko indirektno sa pomerajem	preostala 3 bita sadrže registar koji se koristi pri adresiranju, a u 2 dodatna bajta zapisan je označeni pomeraj

Sintaksa operanada:

20 – neposredna vrednost 20

&x – vrednost simbola x

x – memorijsko direktno adresiranje

*20 – lokacija sa adrese 20

r3 – direktno registarsko

r4[32] – registarsko indirektno sa pomerajem

r5[x] – registarsko indirektno sa pomerajem

$x – PC relativno adresiranje promenljive x

Opis rešenja

Zadatak je rešen na jeziku C++. Ulazni podaci su: ime ulaznog fajla koji sadrži source kod, ime izlaznog fajla i početna adresa.

Pored main.cpp fajla prisutni su i cpp i header fajlovi nekoliko klasa koje rade odvojen posao u svrhu bolje raspodele poslova i čitljivijeg koda.

- **Structures** - sadrži strukture podataka poput skupova, mapa, nizova, struktura i klasa koje opisuju sve direktive, instrukcije, uslove, greške i tabele koje su potrebne u zadatku.

- **RegexParser** - sadrži metode za proveru ispravnosti labela, instrukcija, izraza, operanada, svih mogućih vrsta adresiranja iz zadatka kao i čišćenje whitespace-ova i komentara

- **FirstPass** - sadrži metode koje upravljaju linijom koja je prosleđena i u zavisnosti od sadržaja linije kreiraju ulaz u tabelu simbola nakon raznih provera i ispitivanja poput provere tipa simbola, izračunavanje koliko ta instrukcija proizvodi offset u memoriji, kreiranje niza operanada, čišćenje komentara i whitespaceova pozivajući parser itd.

  Prvi prolaz samo kreira simbole i dodaje ih u tabelu. Prvi prolaz će javiti sve sintaksne greške koje postoje u source kodu kako bi drugi prolaz krenuo znajući da je što se tiče sintakse i postavke zadatka sve ispoštovano. Ovde se nalazi celokupan posao koji je potrebno uraditi i nakon završetka prolaska kroz fajl prvi put, ovaj objekat ispisuje kreiranu tabelu simbola u izlazni fajl. Brojači se resetuju i tok se dalje nastavlja u klasi SecondPass.

- **SecondPass** - sadrži metode koje detaljno nastavljaju da analiziraju source kod liniju po liniju. Nakon prvog prolaska jedine greške koje mogu postojati a nisu proverene do sada jesu one koje se tiču simbola koji se pojave u instrukcijama a sada znamo da ne postoje u tabeli simbola.

  Drugi prolaz radi opsluživanje direktiva, od globalizacije simbola, kreiranja sekcija, kreiranje ulaza u listu sekcija, preko analiziranja uslovnih i instrukcijskih kodova i tipova operanada i transformisanje u 2B/4B instrukcije, sve do kreiranja relokacionih tabela za svaku sekciju i svako pojavljivanje simbola kome adresa može da se promeni.

  Nakon završetka oba prolaza, na već kreiran izlazni fajl dodaju se tabele sekcija i relokacija. Sekcije sadrže sirove podatke u vidu niza bajtova u little endian formatu nezavisno od funkcije, dakle ukoliko treba pročitati instrukciju veličine duple reči tj 4B, na najnižoj mem. lok. se nalazi najniži bajt itd. Kada se dohvati instrukcija i sastavi u celinu, tumače se prva dva bajta kao kodovi a poslednja dva bajta kao podatak.

Instalacija g++ na Ubuntu 12.04 LTS

sudo apt-get upgrade gcc
sudo apt-get install g++

Primer prevođenja

g++ -std=c++14 *.cpp -o program
./program ulaz.txt izlaz.txt 0
nano izlaz.txt

Testovi

- **ulaz1.txt**

```
.data
        .global x
x:      .word 2, 3, x, y
        .skip 1
        .align 2
a:      .long 1024
        .long x, x+8
.text
        .global y
        moval r2, r3
y:      moval r4, r5

.end
```

- **izlaz1.txt**

#id	#name	#type	#section	#size	#value	#scope
0	.data	section	.data	0x16	0x80	l
3	.text	section	.text	0x84	0x96	l
2	a	symbol	.data	0x0	0x8a	l
1	x	symbol	.data	0x0	0x80	g
4	y	symbol	.text	0x0	0x82	g

```
#text
4B F5 8D F5

#data
02 00 03 00 80 00 82 00 00 00 00 04 00 00 80 00
00 00 88 00 00 00

#rodata
/


#bss
/

#text_relocation
/

#data_relocation
4           R_386_16        0
6           R_386_16        3
e           R_386_32        0
12          R_386_32        0

#rodata_relocation
/

#bss_relocation
/
```

- **ulaz2.txt**

```
 .text
        .global a, d
        addeq r0, r5
        shlal r1, 9
b:          subgt r4, d
            movne r3, r0[p]
        mulal r0, 3
t:      moval *20, r0
        callal $a

a:       .long a-t, b-14
        .global b
        moval e, r4
        calleq r3[t]
.data
        .word a, 16, a-b
e:
            .long 99
.rodata
p:       .char 24, 2, 25
        .word 202
        .long .data

.end
```

- **izlaz2.txt**

#id	#name	#type	#section	#size	#value	#scope
4	.data	section	.data	0xa	0xaa	l
6	.rodata	section	.rodata	0x89	0x8a	l
0	.text	section	.text	0x2a	0x80	l
3	a	symbol	.text	0x0	0x9a	g
1	b	symbol	.text	0x0	0x86	g
8	d	symbol	/	0x0	0x0	g
5	e	symbol	.data	0x0	0x86	l
7	p	symbol	.rodata	0x0	0x80	l
2	t	symbol	.text	0x0	0x92	l

```
#text
0D 01 09 00 20 F9 00 00 90 85 80 00 78 75 03 00
00 C9 14 00 08 F6 30 01 00 EE 08 00 00 00 78 00
00 00 86 00 0C F6 92 00 60 2F

#data
9A 00 10 00 14 00 63 00 00 00

#rodata
18 02 19 CA 00 AA 00 00 00

#bss
/

#text_relocation
6          R_386_16        8
a          R_386_16        6
```

```
1e          R_386_32        0
22          R_386_16        4
26          R_386_16        0

#data_relocation
0           R_386_16        3

#rodata_relocation
/

#bss_relocation
/
```

- **ulaz3.txt**

```
.text
        .global a, b, c
a:      addal r1, r2
        subeq r1, 4
        mulne r2, a
        divgt a, r3
        cmpal r1[2], r2
        andeq r4[a], r3
        orne  $a, r1
        notgt r2, $a
b:      testal r1, *4
        pusheq r2
        popne r3
        callgt b
        iretal
        moveq r4, &a
        shlne r3, b+1
        shrgt r4, a+b
c:      retal
        jmpeq a
        .char 100
        .word 200, a, b
        .long a+20, b-40
        .skip 5
        .align 8
.bss
        .global x, y, z
.data
        .word 255, a, b+3
        .long b+c
.rodata
        .skip 3
        .align 8
.end
```

- **izlaz3.txt**

#id	#name	#type	#section	#size	#value	#scope
4	.bss	section	.bss	0x0	0xd8	l
5	.data	section	.data	0xa	0x80	l

6	.rodata	section	.rodata	0x88	0x8a	l
0	.text	section	.text	0x58	0x80	l
1	a	symbol	.text	0x0	0x80	g
2	b	symbol	.text	0x0	0x9e	g
3	c	symbol	.text	0x0	0xb8	g
7	x	symbol	/	0x0	0x0	g
8	y	symbol	/	0x0	0x0	g
9	z	symbol	/	0x0	0x0	g

#text
2A C1 04 00 20 05 80 00 50 49 80 00 0B 8E 02 00
2A D3 80 00 8B 17 16 01 09 5A 1A 01 50 9D 04 00
30 E1 40 25 60 69 9E 00 00 AE 00 F0 80 00 80 35
9F 00 70 79 1E 01 90 BD 00 C0 80 00 E0 35 64 C8
00 80 00 9E 00 94 00 00 00 76 00 00 00 00 00 00
00 00 00 00 00 00 00 00

#data
FF 00 80 00 A1 00 56 01 00 00

#rodata
00 00 00 00 00 00 00 00

#bss
/

#text_relocation
6	R_386_16	0
a	R_386_16	0
12	R_386_16	0
26	R_386_16	0
2c	R_386_16	0
30	R_386_16	0
34	R_386_16	0
34	R_386_16	0
41	R_386_16	0
43	R_386_16	0
45	R_386_32	0
49	R_386_32	0

#data_relocation
2	R_386_16	1
4	R_386_16	2

#rodata_relocation
/

#bss_relocation
/