PROJEKTNI ZADATAK

SISTEMSKI SOFTVER (13S113SS)

Vuk Vuković 0119/17

Opis rešenja

Domaći zadatak realizovan je iz dva dela: jednoprolaznog asemblera i interpretativnog emulatora.

Asembler

Za parsiranje i leksičku analizu ulaznog koda programera korišćeni su alati GNU Flex i GNU Bison. Pravila jezika mogu se pronaći u datoteci *scanner.l*, dok se tokeni mogu pronaći u *parser.y*.

Sve vrste adresiranja su predstavljene korenom klasom *Addressing* iz koje se izvode pojedinačni načini adresiranja i svoje ponašanje ispoljavaju polimorfno.

Klasa Code predstavlja sponu između parsirane ulazne datoteke i samog asemblera. Ova klasa enkapsulira tabelu simbola i prevedene sekcije.

Pored klase koja predstavlja tabelu simbola (*SymbolTable*), postoji i klasa *EquTable* koja sadrži tabelu tj. graf nerazrešenih simbola definisanih *equ* direktivom.

Svaka instrukcija, kao i direktive koje generišu objektni kod, se predstavlja pomoću klase *Encoding* koja sadrži niz bajtova prevedene instrukcije, kao i niz zapisa korišćenih simbola na kojima će biti potrebno izvršiti *backpatching.* Dalje se vrši spajanje ovih objekata u sekcije uz propisno ažuriranje udaljenosti lokacije korišćenog simbola od početka niza bajtova.

Nakon parsiranja, stvaranja simbol tabele i objektnog koda kao i razrešavanja simbola, vrši se backpatching i kreiranje relokacionih zapisa (klasa *Relocations*).

Klasa *BinaryOutFile* omogućava jednostavnu serijalizaciju svih potrebnih tipova u objektnu dadoteku koristeći koncept C++ šablona.

Emulator

Klasa *BinaryInFile* omogućava jednostavnu deserijalizaciju svih potrebnih tipova iz objektne dadoteke.

Klasa *Reader* realizuje čitanje objektnih datoteka, proveru raspoređivanja po memoriji, agregaciju istoimenih sekcija (uz potrebne izmene objektnog koda i relokacija). Takođe, klasa *Reader* u memoriju vrši učitanje programa i razrešavanje relokacija.

Klasa *Memory* predstavlja memoriju veličine 2^16 adresibilnih jedinica od 1B i nudi interfejs za čitanje i upis podataka od 1B i 2B.

Klasa *CPU* jeste centralna klasa emulatora i vrši čitavo izvršavanje učitanog programa koristeći ostale elemente. Sadrži ALU jedinicu koja se koristi za aritmetičke instrukcije. Nakon čitanja, dekodovanja i izvršavanja instrukcije vrši se proera postojanja prekida i njihova obrada.

Svi operandi predstavljeni su korenom apstraktnom klasom *Operand* iz koje se izvode klase za određene tipove adresiranja i koje ispoljavaju svoje ponašanje konceptom polimorfizma. Na ovaj način, *CPU* nije svestan tipa operanda koji koristi, s obzirom da mu je isključivo dovoljan *read/write* interfejs.

Uređaji terminal i tajmer su predstavljeni klasama *Terminal* i *Timer.*
Kako bi se poboljšalo korisničko iskustvo, osnovni terminal je prebačen u presan (*raw*) mod, tj. isključen je automatski ispis unetih karaktera, isključeno je blokirajuće čitanje i potreba za enter tasterom na kraju unetog saržaja.
Timer je implementiran korišćenjem trenutnog sistemskog vremena i provere proteklog vremena između dve provere.

Uputstvo za prevođenje

Projekat je organizovan u folderima *assembler* i *emulator*. Svaki od foldera sadrži odgovarajuće podfoldere *inc* i *src* koji sadrže zaglavlja tipova i C++ izvorni kod.

Kako bi se izvršilo prevođenje, pored Unix baziranog operativnog sistema, potrebno je postojanje C++ kompajlera, kao i GNU alati Flex i Bison (za prevođenje projekta *assembler*).

Radi jednostavnijeg prevođenja kreirane su skripte *compile.sh* u odgovarajućim projektima čijim se pokretanjem stvara izvršna datoteka.

Uputstvo za pokretanje

Nakon prevođenja, u odgovarajućim folderima nastaju izvršne datoteke *assembler* i *emulator*.
Za pokretanje asemblera obavezni argumenti su ulazna i izlazna datoteka (predmetni program). Opciono, može se zadati i parametar *-t* kako bi se na standardni izlaz ispisao tekstualni prikaz predmetnog programa.

Primer poziva asemblera:
asembler *-o izlaz.o ulaz.s -t*

Za pokretanje emulatora potrebno je kao argumente navesti jedan ili više predmetnih programa koji će biti uvezani i izvršeni. Opciono, moguće je i navođenje lokacija na koje će sekcije biti postavljene u obliku *–place=<ime_sekcije>@<adresa>.*

Primer poziva emulatora:
emulator *program1.o program2.o -place=ivtp@0x0000*

Testovi

Test 1

Test 1 prikazuje funkcionalnost terminala, tajmera i prekida. Svaki uneti karakter sa tastature biće ispisan u vidu „CLICKED: [taster]“. Pored toga na svake 2s biće ispisana poruka „TICK“ u okviru prekidne rutine tajmera. Pored ovoga, test prikazuje i funkcionalnost smeštanja sekcije na željenu adresu s obzirom da je prilikom poziva emulatora u vidu argumenata navedeno - *place=ivtp@0x0000 -place=text@0x1000.*

Izgled izlaza testa nakon pritisnutih tastera h, e, l, l, o, s, s, e, t, f sa određenim vremenskim razmakom:

```
HELLO!
CLICK x TO EXIT
CLICKED: h
CLICKED: e
TICK
CLICKED: l
CLICKED: l
CLICKED: o
TICK
CLICKED: s
CLICKED: s
TICK
CLICKED: e
CLICKED: t
CLICKED: f
TICK
TICK
```

Test 2

Test 2 prikazuje rad uređaja tajmera. Prilikom dešavanja prekida od tajmera , u okviru prekidne rutine se ispisuje poruka „TICK x“ gde je x brojač. Klikom na tastere 0-7 menja se brzina otkucaja tajmera.

Izgled izlaza testa nakon nekoliko sekundi i promene brzine tajmera:

```
CHANGE TIMER 0–7
TICK 1
TICK 2
TICK 3
TICK 4
TICK 5
TICK 6
TICK 7
TICK 8
TICK 9
```

Test 3

Test 3 prikazuje rad PC relativnog adresiranja za skokove i podatke u različitnim situacijama i tipa simbola koji se adresira.

mov $hello_label, %r2
call print
mov $hello_label, %r2
*call *print_same_section(%pc)* PC relativno adresiranje simbola u istoj sekciji
mov $hello_label, %r2
*call *print_other_section(%pc)* PC relativno adresiranje simbola u drugoj sekciji
mov $hello_label, %r2
*call *print_global(%pc)* PC relativno adresiranje simbola označeno kao global
*call *print_other_file(%pc)* PC relativno adresiranje simbola iz drugog fajla

Potrebno je primetiti da se za slučaj adresiranja u istoj sekciji ne generiše relokacioni zapis. Takođe, za PC relativno adresiranje simbola označenog kao global se generiše relokacioni zapis za sam taj simbol.

Pored toga, prikazano je i PC relativno adresiranje podataka gde se zapažaju ista svojstva.

```
Relocations for text
-------------------------------------------------------------
Offset    Type                Referenced
-------------------------------------------------------------
4         R_386_16            data
9         R_386_16            text
13        R_386_16            data
22        R_386_16            data
27        R_386_PC16          pointers
31        R_386_16            data
36        R_386_PC16          print_global
40        R_386_PC16          print_other_file
44        R_386_PC16          data
68        R_386_PC16          data
72        R_386_PC16          data
108       R_386_16            text
111       R_386_16            text
```

Izlaz testa su uspešno ispisane poruke i vrednosti sa lokacija koje prikazuju uspešan rad.

```
HELLO
HELLO
HELLO
HELLO
HELLO2!
2
3
```

Test 4

Test 4 oslikava rad INT instrukcije za programsko izazivanje prekida. Na početku programa pozivaju se int $2 i int $4 što izaziva prekid tajmera (i ispis poruke TICK), kao i ulazak u prekidnu rutinu 4 koju je programer definisano (i ispis poruke INT ROUTINE). Ovaj test takođe oslikava rad I bita koji zabranjuje prekide dok je program u nekoj od prekidnih rutina. Ovo je oslikano time što prekidna rutina za unos sa terminala u sebi sadrži beskonačnu petlju. Izazivanjem ovog prekida, program ulazi u beskonačnu petlju dok su isključeni prekidi i sistem prestaje da reaguje na prekide (ne ispisuju se TICK poruke).

```
TICK
INT ROUTINE
TICK
TICK
TICK
TICK
TICK
```

Izgled izlaza nakon nekoliko seknudi i klika nekog od tastera što izaziva prestanak reagovanja sistema na prekide.

Test 5

Test 5 oslikava rad razrešavanja definisanih simbola različitog tipa. Izlaz testa je simbol tabela sa definisanim simbolima, njihovim vrednostima, sekcijama, kao i vezivanjem.

```
.equ A, 5
.equ B, A+1
.equ C, A+B-4
.extern D
.equ E, D+C
.equ F, label1-3
.equ G, label1-main
.equ H, label1-main+E
.equ I, label1+3
.global I
.global C

.section text:
 main:
   halt
 label1:
   halt
```

Name	Value	Reference	Bind
text	0	text	l
A	5	ABS	l
B	6	ABS	l
C	7	ABS	g
D	0	UND	g
E	7	D	l
F	-2	text	l
G	1	ABS	l
H	8	D	l
I	4	text	g
label1	1	text	l
main	0	text	l

Test 6

Test 6 prikazuje rad linkera (razrešavanje simbola i agregacija sekcija) i asemblera (globalni i eksterni simboli). Istomene sekcije se nalaze u različitim fajlovima . Takođe, simboli koji se koriste su u različitim fajlovima. Test prikazuje uspešno pozivanje „potprograma“ definisanog u drugom fajlu, kao i učitavanje vrednosti simbola upisanih u drugom fajlu i drugoj sekciji.

Potprogram print, lokacija *A_location* sa vrednošću 4 i *B_text* sa vrednošću „B VALUE:“ su definisani u fajlu test6_1.s, a koriste se u fajlu test6.s. Pored toga, u fajlu test6.s definisan je apsolutni simbol *PRINT_REG* koji se koristi u fajlu test6_1s.

Izlaz testa prikazuje uspešno pročitane i ispisane promenute vrednosti:

```
A VALUE: 4
B VALUE: 5
```

Test 7

Test 7 prikazuje rad uslovnih skokova i aritmetičkih bitova (N, C, Z, O).

Vrše se sledeća poređenja i ispisuje ogovaraćua poruka:

20 > 10 ?
-10 > 20 ?
-200 = -200 ?
-200 = 482 ?
3 >> 1 C set?
2 >> 1 C set?
0xF0 << 1 C set? (prikazuje i rad korišćenja dela registra %r3h)

```
GREATER
NOT GREATER
EQUALS
NOT EQUALS
C SET
C NOT SET
C SET
```

Test 8

Test 8 oslikava rad aritmetičkih operacija ispisujući operande, operaciju i rezultat.

```
142+427=569
325-124=201
21*13=273
25>>1=12
25<<1=50
11^5=14
```

Test 9

Test 9 oslikava rad određenih tipova adresiranja i instrukcija različitih veličina.

mov $1, %r1 neposredno (literal)
mov $A, %r1 neposredno (symbol)
mov %r2, %r1 registarsko direktno
mov (%r2), %r1 registarsko indirektno
mov 2(%r2), %r1 registarsko indirektno sa pomerajem (literal)
mov B(%r2), %r1 registarsko indirektno sa pomerajem (simbol)

mov C(%pc), %r1 PC relativno
mov 0x200, %r1 Memorijsko direktno (literal)
mov F, %r1 Memorijsko direktno (simbol)

mov %r2h, %r1l Prenos viših 8 bitova registra r2 u 8 nižih bitova registra r1 (automatski zaključena veličina)
movb %r2l, %r1l Prenos nižih 8 bitova registra r2 u 8 nižih bitova registra r1 (eksplicitno naveena veličina)

Izlaz testa su validno upisane vrednosti različitim načinima adresiranja:

```
1
2
3
4
5
6
7
8
9
4
3
```