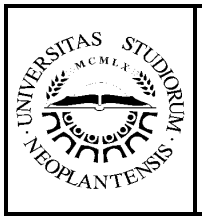


УНИВЕРЗИТЕТ У НОВОМ САДУ
ФАКУЛТЕТ ТЕХНИЧКИХ НАУКА

УНИВЕРЗИТЕТ У НОВОМ САДУ
ФАКУЛТЕТ ТЕХНИЧКИХ НАУКА
НОВИ САД
Департман за рачунарство и аутоматику
Одсек за рачунарску технику и рачунарске комуникације

ПРОЈЕКТНИ ЗАДАТАК

Кандидат: **Михал Сабадош**
Број индекса: **SW 20/2019**

Предмет: **Системска програмска подршка I**
Тема рада: **MABH - преводилац**

Ментор рада: **др Миодраг Ђукић**

Нови Сад, мај, 2021.

Sadržaj

SLIKE

1. Analiza problema

MAVN (Mips Asembler Visokog Nivoa) je alat koji prevodi program napisan na višem MIPS 32bit asemblerskom jeziku na osnovni asemblerski jezik. Viši MIPS 32bit asemblerski jezik služi lakšem asemblerskom programiranju jer uvodi koncept registarske promenljive. Registarske promenljive omogućavaju programerima da prilikom pisanja instrukcija koriste promenljive umesto ravih resursa. Ovo znatno olakšava programiranje jer programer ne mora da vodi računa o korišćenim registrima i njihovom sadržaju.

Potrebno je realizovati MAVN prevodilac koji prevodi programe sa višeg asemblerskog jezika na osnovni MIPS 32bit asemblerski jezik. Prevodilac treba da podržava detekciju leksičkih, sintaksnih i semantičkih grešaka kao i generisanje odgovarajućih izveštaja o eventualnim greškama. Izlaz iz prevodioca treba da sadrži korektan asemblerski kod koji je moguće izvršavati na MIPS 32bit arhitekturi (simulatoru).

MAVN jezik podržava 10 MIPS instrukcija, a to su:

- add – (addition) sabiranje
- addi – (addition immediate) sabiranje sa konstantom
- b – (unconditional branch) bezuslovni skok
- bltz – (branch on less than zero) skok ako je registar manji od nule
- la – (load address) učitavanje adrese u registar
- li – (load immediate) učitavanje konstante u registar
- lw – (load word) čitavanje jedne memorijske reči
- nop – (no operation) instrukcija bez operacije
- sub – (subtraction) oduzimanje
- sw – (store word) upis jedne memorijske reči
- mult – (multiply) množenje
- or – (or) binarna operacija ili
- seq – (set equal) postavljane jednakosti

Sintaksa MAVN jezika opisana je gramatikom:

Q → S ; L	S → _mem mid num	L → eof	E → add rid, rid, rid
	S → _reg rid	L → Q	E → addi rid, rid, num
	S → _func id		E → sub rid, rid, rid
	S → id: E		E → la rid, mid
	S → E		E → lw rid, num(rid)
			E → li rid, num
			E → sw rid, num(rid)
			E → b id
			E → bltz rid, id
			E → nop
			E → mult rid, rid
			E → or rid, rid, rid
			E → seq rid, rid, rid

Terminalni simboli MAVN jezika su:

: ; , ( )

_mem, _reg, _func, num id, rid, mid, eof, add, addi, sub, la, lw, li, sw, b, bltz, nop

- Deklaracija funkcije:

  _func *funcName*

  funcName –mora početi slovom, u nastavku može biti bilo koji niz slova i brojeva.
- Deklaracija memorijske promenljive:

  _mem *varName value*

  varName – mora početi malim slovom m u nastavku može biti bilo koji broj.
- Deklaracija registarske promenljive:

  reg *varName*

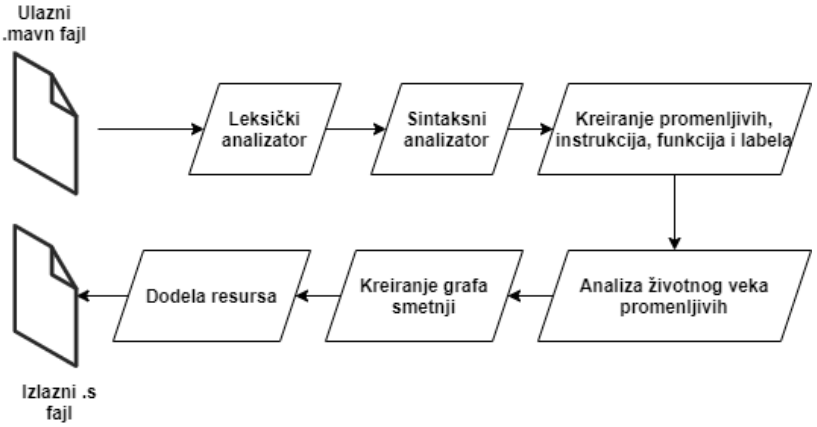
  varName – mora početi malim slovom r u nastavku može biti bilo koji broj.

2. Koncept rešenja

Kako bi realizovali MAVN prevodilac potrebno je da implementiramo sledeće alate odnosno faze:

1. Leksički analizator
2. Sintaksni analizator
3. Kreiranje promenljivih, instrukcija, funkcija i labela
4. Analizu životnog veka promenljivih
5. Kreiranje grafa smetnji
6. Dodela resursa

Grafički predstavljen proces prevođenja prikazuje slika 1.



Slika 1 Grafički prikaz procesa prevođenja

2.1 Leksički analizator

Ovaj alat pretvara ulazni niz karaktera iz .mavn datoteke u niz tokena odnosno leksičkih simbola koji odgovaraju rečima programskog jezika MAVN. Pretvaranje se vrši tako što se čita karakter po karakter iz niza karaktera koji su učitani iz .mavn datoteke i formiraju se tokeni koji su potrebni za sledeći alat.

2.2 Sintaksni analizator

Sintaksni analizator proverava ispravnost napisanih tokena koje smo dobili iz leksičke analize sa gramatikom programskog jezika MAVN. U ovome alatu se razlikuju terminalne simbole koji su preuzeti iz azbuke jezika MAVN kao na primer id, add, la, :, _reg... i neterminalne simbole koji se nalaze sa leve strane produkcije na primer simbol E, Q, L...

Ukoliko se desi sintaksna greška, ovaj alat nam pomaže da je otkrijemo. Ukoliko je uspešna sintaksna analiza, tada možemo da pređemo na pravljenje promenljivih, instrukcija, funkcija i labela.

2.3 Kreiranje promenljivih, instrukcija, funkcija i labela

U ovoj fazi kreiramo prvo promenljive kako memorijske tako i registarske, funkcije i labele prolaskom kroz listu tokena. Promenljive, čuvamo i upisujemo ih u listu promenljivih, funkcije u listu funkcija i labele u listu labela.

Nakon ovoga kreiraju se instrukcije i čuvaju se u listi instrukcija kroz koju prolazimo u trećoj fazi pravljenja prethodnika i sledbenika svake funkcije.

2.4 Analiza životnog veka promenljivih

U prethodnoj fazi su dobijene liste instrukcija kojima su dodeljeni skupovi promenljivih dest, src, use i def, međutim treba još popuniti i skupove in i out kod svake instrukcije kako bi mogli da pređemo u sledeću fazu. Ove skupove popunjavamo sledećim algoritmom:

Skup out[n] je unija „in“ skupova svih naslednika čvora n, odnosno unija svih promenljivih s koje pripadaju skupu „in“ svakog naslednika čvora n. Skup in[n] čine sve promenljive iz skupa use[n] plus.

Ovaj algoritam je matematički prikazan na slici 2.

$$out[n] \leftarrow \bigcup_{s \in succ[n]} in[s]$$

$$in[n] \leftarrow use[n] \cup (out[n] - \text{def}[n])$$

Slika 2 matematički definisan algoritam analize životnog veka

2.5 Kreiranje grafa smetnji

Graf smetnji prikazujemo matricom smetnji gde su kolone i redovi matrice isti i predstavljaju sve registarske promenljive u programu. Promenljivim koje su u smetnji ne mogu se dodeliti isti resursi odnosno isti registri, zbog toga kreiramo matricu smetnji pomoću koje vidimo smetnje između promenljivih. Analizom životnog veka smo popunili sve podatke koje su potrebne za kreiranje matrice smetnji prolaskom kroz sve instrukcije pomoću pravila:

- Za svaku promenljivu A iz skupa promenljivih **def**, dodati novu smetnju između promenljive A i svake promenljive B_i iz skupa **out** iste instrukcije.

2.6 Dodela resursa

Nakon kreiranja matrice smetnji potrebno je preći na fazu uprošćavanja (simplify) gde se sve registarske promenljive stavljaju na stek (simplification stack). Ovaj stek se popunjava tako što se iz grafa smetnji uzme promenljiva sa najviše broja smetnji, ali ne većim od broja registara umanjen za jedan (__REG_NUMBER__-1).

Ukoliko se dođe do trenutka da sve promenljive u grafu smetnji imaju veći broj smetnji od maksimalnog, tada se dešava prelivanje (spill) i to znači da ne postoji dovoljno registara koje treba dodeliti promenljivim.

Ukoliko se uspešno popuni stek, tada se prelazi u fazu izbora odnosno same dodele resursa promanljivim. Prolazi se kroz stek promenljih i dodeljuje se svakoj promenljivoj prvi slobodan registar koji nije dodeljen njegovim susedima, odnosno promenljivim sa kojima ima smetnje.

Nakon ove faze, potrebno je upisati u izlaznu .s datoteku preveden kod na asemblerski jezik.

3. Opis rešenja

3.1 Moduli i osnovne metode

3.1.1 Modul glavnog programa (main)

```
int main();
```

Glavna funkcija programa. Prikazuje osnovne informacije o programu.

3.2 LexicalAnalysis

3.2.1 Modul za čitanje ulazne datoteke

```
bool LexicalAnalysis::readInputFile(string fileName);
```

Modul za čitanje ulazne datoteke *fileName*. Vraća true ukoliko je čitanje uspešno, false u suprotnom.

3.2.2 Metoda za inicijalizaciju leksičke analize

```
void LexicalAnalysis::initialize();
```

3.2.3 Metoda za izvršavanje leksičke analize

```
bool LexicalAnalysis::Do();
```

Prolazi kroz Tokene i dodaje ih u listu tokena tokenList.

Vraća true ako je lexička analiza uspešno izvršena, false u suprotnom.

3.3 SyntaxAnalysis

3.3.1 Metoda za izvršavanje sintaksne analize

```
bool SyntaxAnalysis::Do();
```

Proverava ispravnost tokena iz liste tokena u gramatici jezika pozivanjem neterminalnih simbola. Vraća **true** ako je sintaksna analiza uspešno izvršena, **false** u suprotnom**.**

3.4 VarInstrBuilder

3.4.1 Metoda za generisanje promenljivih, funkcija i labela

```
void VarInstrBuilder::generateVariables();
```

Prolazi kroz listu tokena, kreira memorijske i registarske promenljive, funkcije i labele i čuva ih u listi promenljivih, funkcija i labela.

3.4.2 Metoda za generisanje instrukcija

```
void VarInstrBuilder::generateInstructions();
```

Prolazi kroz listu tokena, kreira instrukcije i čuva ih u listi instrukcija.

3.4.3 Metoda za generisanje sledbenika i prethodnika

```
void VarInstrBuilder::generateSuccPred();
```

Prolazi kroz listu instrukcija i dodaje svakoj instrukciji njegovog prethodnika i sledbenika.

3.5 LivenessAnalysis

```
void livenessAnalysis(Instructions& instructions);
```

Parametar *instructions* je lista instrukcija.

Prolazak kroz instrukcije i dodavanje skupova promenljivih **in** i **out** svakoj instrukciji.

3.6 ResourceAllocation

3.6.1 Metoda za kreiranje grafa smetnji

```
InterferenceGraph* buildInterferenceGraph(Instructions& instructions);
```

Parametar *instructions* je lista instrukcija.

Kreira matricu smetnji.

Vraća objekat grafa smetnji u kom se nalazi matrica smetnji.

3.6.2 Metoda za popunjavanje steka uprošćavanja

```
stack<Variable*>* doSimplification(InterferenceGraph* ig);
```

Parametar *ig* je prethodno kreiran graf smetnji.

Prolazi kroz matricu smetnji i popunjava stek varijabli.Baca grešku ukoliko je došlo do spilla. Vraća stek.

3.6.3 Metoda za dodelu resursa promenljivim

```
bool doResourceAllocation(stack<Variable*>* simplificationStack, InterferenceGraph* ig);
```

Parametri *simplificationStack* je prethodno popunjen stack a *ig* popunjen graf smetnji.

Dodeljuje registre promenljivim iz steka.

Vraća true ako je uspešno izvšena dodela, false u suprotnom.

3.7 Metoda za upis u izlazni fajl

```
void writeFile(string fileName, Variables vars, FuncLabs funcLabs, Instructions instrs);
```

Parameti: ime izlaznog fajla *fileName*, lista varijabli *vars*, lista funkcija i labela *funcLabs* i lista instrukcija *instrs*.

Upisuje u izlazni fajl rešenje prevedenog MAVN koda.

4. Verifikacija

4.1 Primer 1

Program na MAVN programskom jeziku prikazan na slici 3

```
_mem m1 6;
_mem m2 5;

_reg r1;
_reg r2;
_reg r3;
_reg r4;
_reg r5;

_func main;
    la      r4,m1;
    lw      r1, 0(r4);
    la      r5, m2;
    lw      r2, 0(r5);
    add     r3, r1, r2;
```

Slika 3 prikaz ulaznog fajla simple.mavn

Slika 4 pokazuje izlazni fajl nakon prevođenja programa na MIPS 32bit asemblerski jezik.

```
.globl main

.data
m1: .word 6
m2: .word 5

.text
main:
    la  $t0, m1
    lw  $t1, 0($t0)
    la  $t0, m2
    lw  $t0, 0($t0)
    add $t0, $t1, $t0
```

Slika 4 prikaz izlaznog fajla resultSimple.s

4.2 Primer 2

Program na MAVN programskom jeziku prikazan na slici 5

```
_mem m1 6;
_mem m2 5;
_mem m3 0;

_reg r1;
_reg r2;
_reg r3;
_reg r4;
_reg r5;
_reg r6;
_reg r7;
_reg r8;

_func main;
    la      r1, m1;
    lw      r2, 0(r1);
    la      r3, m2;
    lw      r4, 0(r3);
    li      r5, 1;
    li      r6, 0;
lab:
    add     r6, r6, r2;
    sub     r7, r5, r4;
    addi    r5, r5, 1;
    bltz    r7, lab;

    la      r8, m3;
    sw      r6, 0(r8);
    nop;
```

Slika 5 prikaz ulaznog fajla multiply.mavn

Slika 6 pokazuje izlazni fajl nakon prevođenja programa na MIPS 32bit asemblerski jezik.

```
.globl main

.data
m1: .word 6
m2: .word 5
m3: .word 0

.text
main:
    la   $t0, m1
    lw   $t3, 0($t0)
    la   $t0, m2
    lw   $t4, 0($t0)
    li   $t2, 1
    li   $t1, 0
lab:
    add $t1, $t1, $t3
    sub $t0, $t2, $t4
    addi    $t2, $t2, 1
    bltz     $t0, lab
    la   $t0, m3
    sw   $t1, 0($t0)
    nop
```

Slika 6 prikaz izlaznog fajla result.s