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

ИСПИТНИ РАД

Кандидат: **Марко Драгићевић**
Број индекса: **SW74/2019**

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

Ментор рада: **др Миодраг Ђукић**

Нови Сад, јун, 2021.

SADRŽAJ

1 Uvod

1.1 MAVN - prevodilac

MAVN prevodilac 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.

1.1.1 MAVN

MAVN prevodilac 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 pravih resursa. Ovo znatno olakšava programiranje jer programer ne mora da vodi računa o korišćenim registrima i njihovom sadržaju.

1.1.2 Podržane instrukcije

MAVN jezik podržava 13 MIPS insturkcija, 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) učitavanje jedne memorijske reči
- nop – (no operation) instrukcija bez operacije
- sub – (subtraction) oduzimanje
- sw – (store word) upis jedne memorijske reči
- abs – (absolute value) apsolutna vrednost registra
- or – (logical or) opracija logicko ILI
- seq – upisuje u odredišnji registar 1, ako su dva izvorišna registra jednaka

1.1.3 Sintaksa MAVN jezika

$Q \rightarrow S\ ;\ L$

$S \rightarrow \_mem\ mid\ num$

$S \rightarrow \_reg\ rid$

$S \rightarrow \_func\ id$

$S \rightarrow id:\ E$

$S \rightarrow E$

$L \rightarrow eof$

$L \rightarrow Q$

$E \rightarrow add\ rid,\ rid,\ rid$

$E \rightarrow addi\ rid,\ rid,\ num$

$E \rightarrow sub\ rid,\ rid,\ rid$

$E \rightarrow la\ rid,\ mid$

$E \rightarrow lw\ rid,\ num(rid)$

$E \rightarrow li\ rid,\ num$

$E \rightarrow sw\ rid,\ num(rid)$

$E \rightarrow b\ id$

$E \rightarrow bltz\ rid,\ id$

$E \rightarrow nop$

$E \rightarrow abs\ rid,\ rid$

$E \rightarrow or\ rid,\ rid,\ rid$

$E \rightarrow seq\ rid,\ rid,\ rid$

1.2 Zadatak

Zadatak MAVN prevodioca je da učitanu ulaznu datoteku pisanu na MAVN jeziku prevede na MIPS 32bit asembler. Ograničiti se na jednu ulaznu datoteku. Koristiti ekstenziju „.mavn” za ulaznu datoteku koja sadrži program na MAVN jeziku.

Prevodilac treba da prilikom prevođenja:

1. Dodeli resurse za registarske promenljive – ograničiti se na 4 registra: t0, t1, t2 i t3 iz MIPS arhitekture
2. Sve memorijske promenljive generiše u sekciju za podatke - .data vodeći računa o sintaksi asemblerskog jezika
3. Sve instrukcije smesti u programsku sekciju - .text
4. Ime funkcije generise kao globalni simbol - .globl i kao labelu na prvu njenu instrukciju
5. Generiše izlaznu datoteku sa ekstenzijom „.s” koja sadrži preveden i korektan MIPS 32bit asemblerski jezik polaznog programa

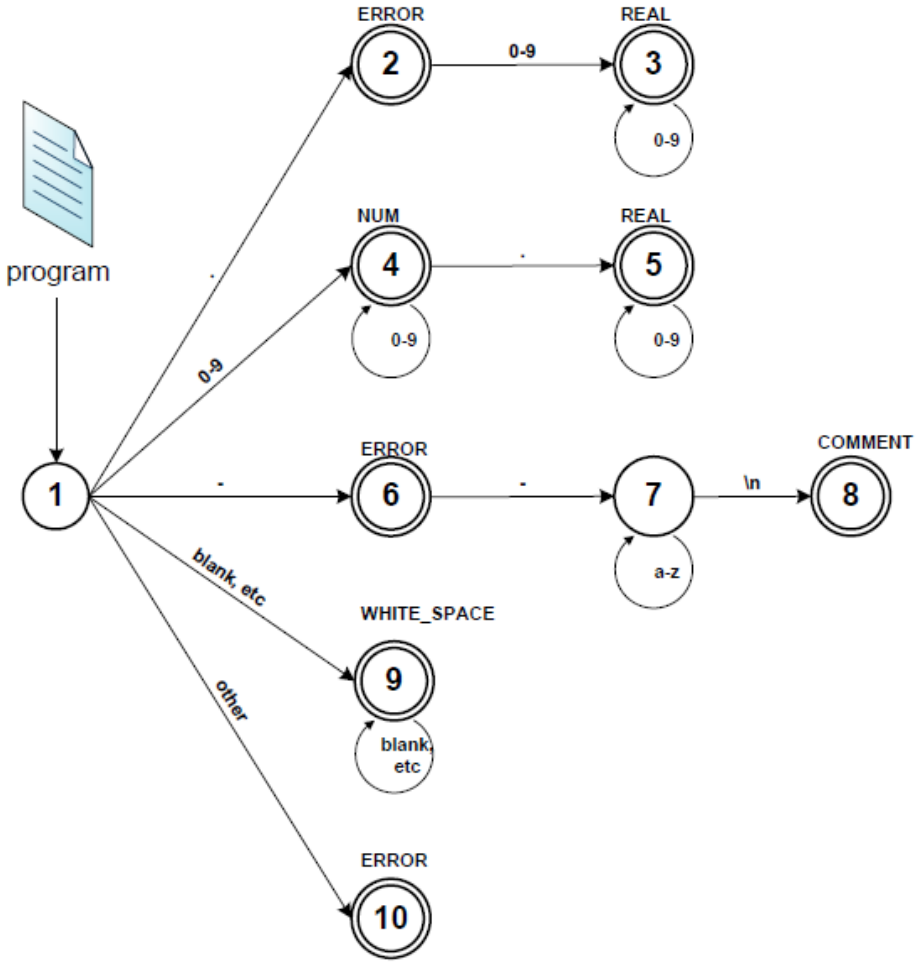
NAPOMENE:

1. Prilikom prevođenja omogućiti detekciju i regaovanje na leksičke, sintaksne i semantičke greške
2. Ukoliko se polazni program uspešno prevede na MIPS 32bit asemblerski jezik, izvršavanje programa je moguće proveriti korišćenjem QtSpim simulatora koji je dostupan na internetu

2 Analiza problema

Problem se može podeliti na sledeće potprobleme:

1. **Leksička analiza** – leksički analizator je alat koji ulazni niz karaktera pretvara u niz tokena – leksičkih simbola koji odgovaraju rečima n ekog programskog jezika. Realizovan je pomoću konačnog determinističkog automata čiji je cilj da prepozna sve simbole u programu. Primer jednog takvog automata dat je na sledećoj slici 1:



2. **Sintaksna analiza** – svaki simbol može biti terminalni ili neterminalni. Terminalni simboli su oni koji su preuzeti iz azbuke jezika. Neterminalni simboli su oni koji se nalaze sa leve strane iste produkcije. Svaka gramatika poseduje jedan neterminalni simbol koji se označava kao početni simbol gramatike.

   Da bi se saznalo da li je neka rečenica gramatički ispravna određuje se njeno poreklo – obavlja se izvođenje. Izvođenje se obavlja tako što se kreće od početnog simbola, nakon čega se u iteracijama umesto neterminalnih simbola smenjuju njihove produkcije.
3. **Analiza životnog veka** - služi da bi se omogućilo da različite promenljive, koje nisu istovremeno u upotrebi, dele isti resurs. Životni vek promenljive započinje definicijom promenljive, a završava se poslednjom upotrebom. Analiza životnog veka se sastoji iz sledećih koraka:
   a. ***Graf toka upravljanja*** – konstruiše se tako što se svaka instrukcija predstavi čvorom grafa, a zatim se za svaka dva čvora x i y, takve da x prethodi y, povlači strelica od x ka y. Životni vek promenljivih teče preko strelica grafa, pa se iz tog razloga određivanje opsega života svodi na problem toka podataka.
   b. ***Graf smetnji*** – Problem koji onemogućava da se isti registar dodeli dvema privremenim promenljivama naziva se smetnja. Najčešći uzrok smetnje je preklapanje opsega životnog veka promenljivih. Graf smetnji se često u računaru predstavlja matricom smetnji gde kolone i redovi predstavljaju sve promenljive u programu. Na preseku odgovarajuće kolone i reda biće upisan indikator da li postoji smetnja između promenljive u koloni i promenljive u redu.
4. **Dodela resursa:**
   a. ***Faza formiranja*** – formira se graf smetnji na osnovu životnog veka promenljivih.
   b. ***Faza uprošćavanja***
   c. ***Faza prelivanja*** – u toku faze uprošćavanja može se desiti da se graf smetnji ne može više uprostiti, a da još uvek postoje čvorovi u njemu. To znači da se dati graf ne može obojiti sa K boja, Tada se pribegava rasterećivanju grafa tako što će nekim promenljivama biti promenjen resurs, tj. biće prebačene iz registara u, na primer, memoriju.
   d. ***Faza izbora*** – nakon što je graf uspešno uprošćen, potrebno je obojiti čvorove. Redosled čvorova je određen stekom na koji su se čvorovi

postavljali prilikom uprošćavanja grafa. Jedino pravilo koje se koristi prilikom bojenja je da se čvoru ne dodeli boja koja je već dodeljena nekom od susednih čvorova.

Poslednja stavka koja preostaje jeste pravljenje izlazne datoteke i upisivanje prikupljenih podataka u nju, tako kreirajući .s fajl.

3 Koncept rešenja

Rešenje je koncipirano na analizi problema.

Program je podeljen u faze:

1. Učitavanje .mavn datoteke iz ulaznog fajla
2. Leksička analiza
3. Sintaksna analiza i prikupljanje svih instrukcija i promenljvih
4. Pravljenje prethodnika i naslednika za svaku instrukciju
5. Analiza životnog veka promenljivih
6. Kreiranje grafa smetnji (matrica smetnji)
7. Pojednostavljenje grafa – pravljenje steka sa promenljivim
8. Konačna dodela resursa – bojenje čvorova
9. Kreiranje izlaznog .S fajla

4 Opis rešenja

4.1 Moduli i osnovne metode

4.1.1 Modul glavnog programa (Main)

```
int main();
```

Glavna funkcija programa. U njoj se redom pozivaju sve faze programa, počevši od učitavanja .mavn datoteke, sve do kreiranja izlaznog .s fajla.

4.1.2 Modul za obavljanje leksičke analize (LexicalAnalysis)

Tri glavne metode u ovom modulu:

1. bool readInputFile(string fileName); - metoda služi za učitavanje ulazne .mavn datoteke
2. void initialize(); - pokreće funkciju initStateMachine() iz modula FiniteStateMachine
3. bool Do(); - obavlja leksičku analizu

4.1.3 Modul FiniteStateMachine

- **Automat prelaza stanja, mapa koje čuva stanja kao ključ, i matricu prelaza stanja kao vrednost:**

STATE_MACHINE stateMachine;

// map<StateNumber, map<TransitionCharacter, NextStateNumber>>

- **Stanja konačnog determinističkog automata**:

static const TokenType stateToTokenTable[D_NUM_STATE];

- **Karakteri koje leksički analizator podržava:**

static const char supportedCharacters[D_NUM_OF_CHARACTERS];

- **Matrica prelaza stanja:**

static const int stateMatrix[D_NUM_STATE][D_NUM_OF_CHARACTERS];

Vrste u matrici prelaza stanja predstavljaju trenutno stanje automata, a kolone matrice predstavljaju podržane karaktere leksičkog analizatora. U preseku vrste i kolone nalazi se identifikator stanja u koje automat prelazi kada naiđe na određeni karakter.

4.1.4 Modul za obavljanje sintaksne analize (SyntaxAnalysis)

U ovom modulu se vrši sintaksna analiza na osnovu date gramatike. Jedan od načina za realizaciju sintaksnog analizatora je pomoću algoritma sa rekurzivnim spuštanjem. Svaka produkcija pretvara se u rečenicu pozivima rekurzivnih funkcija. Algoritam sa rekurzivnim spuštanjem poseduje po jednu funkciju za svaki neterminalni simbol i po jedan uslov za svaku produkciju, tako da će za datu gramatiku biti potrebne funkcije Q, S, L, E u kojima će se ispitivati postojanje terminalnih simbola: : ; , () kao i _mem, _reg, _func, num id, rid, mid, eof, addi, sub, la, lw, li ,sw, b, bltz, nop, abs, or, seq.

4.1.4.1 Opis algoritma sintaksne analize

Na početku algoritam dobavlja sledeći (prvi) simbol, pomoću funkcije **Token getNextToken()** i stavlja ga u globalnu promenljivu *token.* Zatim se poziva funkcija Q. U funkciji Q se radi izvođenje tako što se prvo poziva funkcija S, koja dalje vrši svoje izvodjenje tako što se poziva funkcija **eat(TokenType t)** za sve terminalne simbole i odgovarajuća rekurzivna funkcija za neterminalne simbole, u skladu sa gramatikom.

Funkciji **eat** prosleđuje se simbol koji se očekuje da će biti sledeći u programu. Ukoliko se očekivani simbol ne poklapa sa novootkrivenim simbolom potrebno je prijaviti sintaksnu grešku.

4.1.4.2 Metoda za kreiranje instrukcija (makeInstruciton)

```
void makeInstruction(InstructionType instrType, vector<string>& destVars, vector<string>& srcVars, int instrPos);
```

Kreira jednu instrukciju i dodaje je vektoru svih instrukcija u programu.

Parametri:

- instrType – tip instrukcije (I_ADD, I_BLTZ...)
- destVars – vektor popunjen nazivima odredišnih promenljivih,
- srcVars – vektor popunjen nazivima izvorišnih promenljivih
- instPos – pozicija instrukcije (tj. redni broj).

Pri sinatksnoj analizi, kada dodjemo do simbola T_REG ili T_MEM, znamo da se radi o deklaraciji promenljivih. Tada kreiramo objekat **Variable** i njega ubacjujemo u listu svih varijabli koje su do tada deklarisane. Pre samog ubacivanja, proverava se da li je promenljiva sa tim imenom vec deklarisana.

U svakom ulasku u rekurzivnu funkciju E se definišu dva vektora stringova (nazivi promenljvihi). Oni si pune tako što u njih ubacujemo odgovarajuće vrednosti simbole T_R_ID, T_M_ID ili T_NUM (T_NUM koristim kao konstante).
Prosleđeni elementi vektora se pomoću funkcije **findVar(**string varName, Variables& vars**)** pretvaraju iz imena promenljive u sam objekat klase Variable, ako je promenljiva sa tim imenom prethodno definisana.

4.1.4.3 Metoda za pravljenje prethnodika i naslednika instrukcija (makePredAndSucc)

```
void makePredAndSucc();
```

Prolazi kroz sve instrukcije i dodaje odgovarajuce instrukcije naslednice i prethodnike za svaku instrukciju. U slučaju instrukcija za skokove, vrši se posebna obrada:

- Za tip instrukcije I_BLTZ naslednice će biti instrukcija „ispod" BLTZ, ali I instrukcija ispod labele na koju BLTZ skače.
- Za tip instrukcije I_B naslednica će biti samo instrukcija „ispod" labele na koju B skače, jer je to bezuslovni skok

Takođe, za instrukcije na koje pokazuje labela dodaju se, kao prethodnici, sve instrukcije skoka koje skaču na tu labelu.

4.1.5 Modul za analizu životnog veka (LivenessAnalysis)

Na osnovu prethodno napravljenih instrukcija – radi se analiza životnog veka promenljvih

4.1.5.1 Metoda za analizu životnog veka (livenessAnalysis)

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

Popunjava **in** i **out** skupove za svaku instrukciju.

Parametri:

- instructions – lista svih kreiranih instrukcija.

Skup **out**[n] je unija „**in**" skupova svih naslednika čvora n, odnosno unija svih promenljivih koje pripadaju skupu „**in**" svakog naslednika čvora n. Skup **in**[n] čine sve promenljive iz skupa **use**[n] plus promenljive koje jesu u out[n], ali nisu u def[n].

4.1.6 Modul za kreiranje grafa smetnji (InterferenceGraph)

```
InterferenceGraph& buildInterferenceGraph(Instructions& instructions,
Variables& vars);
```

Kreira graf smetnji zasnovan na listi instrukcija analiziranog programa.

Parametri:

- instructions – lista instrukcija
- vars – lista promenljivih

Povratna vrednost:

- Referenca na kreiran graf smetnji (klasa **InterferenceGraph&** koja se sastoji od **InterferenceMatrix** i **Varables*)**

Da bi se iz rezultata analize životnog veka promenljivih napravio graf smetnji, koristi se sledeće pravilo:

- Za svaku **definiciju** promenljive A, dodaj u grafu smetnji novu smetnju između promenljive A i svake promenljive B_i koja živi na izlazu čvora

4.1.7 Modul za fazu uprošćavanja grafa (Simplification)

```
SimplificationStack& doSimplification(InterferenceGraph ig, int degree);
```

Kreira stek postupnim uklanjanjem čvorova iz grafa smetnji.

Parametri:

- ig – graf smetnji
- degree – broj dostupnih registara

Povratna vrednost:

- Referenca na kreiran stack – std::stack<Variable*>

Na stek se stavlja čvor sa najvećim rangom, pod uslovom da je njegov rang manji od broja dostupnih registara.

4.1.8 Modul za fazu dodele resursa(ResourceAllocation)

```
bool doResourceAllocation(SimplificationStack& ss,InterferenceGraph& ig);
```

„Bojenje" čvorova – dodela realnih registara.

Parametri:

- ss – pojednostavljeni stek

- ig – graf smetnji

Povratna vrednost:

- true – uspešno završena dodela resursa
- false – nije moguće izvršiti dodelu resursa

Skidaju se promenljive sa steka sve dok se on ne isprazni. Zatim se za tu skinutu promenljivu proverava sa kojim je promenljivim u smetnji i za te sa kojima jeste se belezi da se ne mogu dodeliti istom registru.

4.1.9 Modul za kreiranje izlaznog fajla (Output)

```
void createOutFile(string fileName, Instructions& instrs, Variables vars, Labels labels, string funcs);
```

Kreiranje izlazne .s datoteke.

Parametri:

- fileName – ime izlazne datoteke
- instrs – vektor svih instrukcija
- vars – lista svih promenljivih
- labels - labele // unordered_map<string, int>
- funcs – naziv glavne funkcije

Funkcija **createOutFile** se sastoji od sledećih delova:

- ime funkcije generiše kao globalni simbol - .globl
- sve memorijske promenljive generiše u sekciju za podatke - .data
- Sve instrukcije smesti u programsku sekciju - .text

5 Zaključak

Napravljen je jedan koncept za realizaciju datog problema. Napravljen je i verifikovan MAVN prevodilac koji ispunjava zahteve opisane u zadatku.

Verifikacija programa urađena je kroz pokretanje raznih primera .mavn datoteka. Time je potvrđena funkcionalnost svakog pojedinačnog bloka, njihovih veza i programa u celini.

Na sledećoj slici je prikazan ulazni .mavn fajl:

```
_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;
```

A na sledećoj slici je prikazan izlazni .S fajl dobijen pokretanjem MAVN prevodioca:

```
.globl main

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

.text
main:
        la      $t0, m1
        lw      $t2, 0($t0)
        la      $t0, m2
        lw      $t1, 0($t0)
        add     $t0, $t2, $t1
```

6 Literatura

Implementacija projekta se zasniva na delovima prevodioca koji su radjeni na terminima vežbi.

Takođe, informacije koje su priložene u analizi problema su uzete sa materija za vežbe iz predmeta Sistemska programska podrška 1.