УНИВЕРЗИТЕТ У НОВОМ САДУ
ФАКУЛТЕТ ТЕХНИЧКИХ НАУКА

UNIVERZITET U NOVOM SADU
FAKULTET TEHNIČKIH NAUKA
NOVI SAD
Departman za računarstvo i automatiku
Smer računarstvo i automatika

ISPITNI RAD

Kandidat: **Uroš Jevtić**
Broj indeksa: **RA142/2020**

Predmet: **Osnovi paralelnog programiranja i softverski alati**
Tema rada: **MAVN prevodilac**

Novi Sad, jun 2022.

Sadržaj

Spisak slika

1. Uvod

1.1 MAVN Prevodilac

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

2. Analiza problema

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 generiše 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

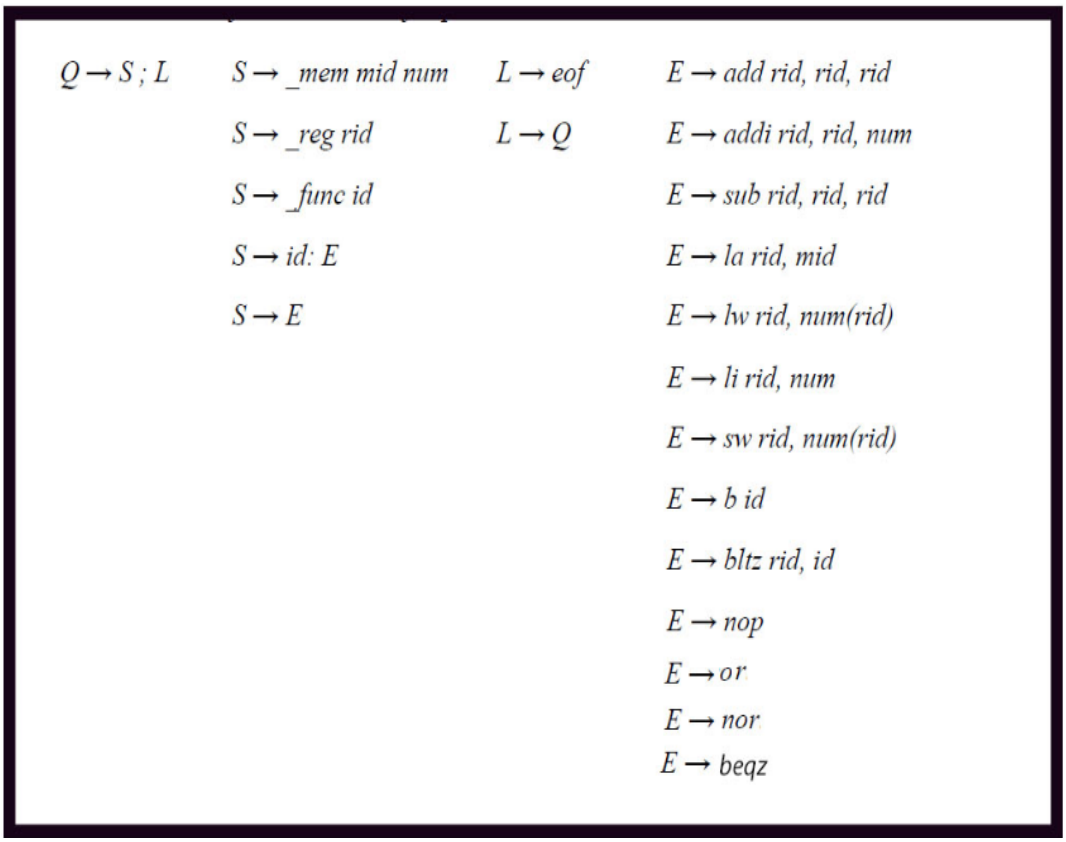
Kako bi se program realizovao potrebno je proći kroz određene faze prevođenja izvornog koda:

1. Leksička analiza
2. Sintaksna analiza
3. Analiza životnog veka promenljivih
4. Dodela resursa

3.Koncept rešenja

Prvi korak u prevođenju sa MAVN jezika je da se odradi leksička analiza. Leksički analizator je realizovan pomoću konačnog determinističkog automata čiji je cilj da prepozna sve simbole u programu.

Drugi korak jeste sintaksna analiza. Zadatak sintaksnog analizatora je da implementira gramatiku jezika (slika 1). Tokom ove analize takođe se vrši popunjavanje listi memorijskih i registarskih promenljivih, kao i poponjavanje liste funkcija i labela. Pored ovoga, vrši se i kreiranje instrukcija.



Slika 1(Gramatika MAVN jezika)

Odmah nakon sintaksne analize i kreiranja instrukcija, instrukcijama se dodaju naslednici(succ) i prethodnici (pred), dok su proemenljive koje se definišu u instrukciji (def) kao i promenljive koje se koriste u instrukcijama (use) dodate prilikom njenog kreiranja.

Sledeći korak jeste analiza životnog veka. Njen zadatak je da popuni in i out liste svih instrukcija pomoću prethodno definisanih use i def, i to radi preko sledećeg algoritma(slika 2):

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

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

Решење ових једначина се добија следећим алгоритмом, исказаним у псеудокоду:

for each n

    $in[n] \leftarrow \{\}$; $out[n] \leftarrow \{\}$

repeat

    for each n

        $in'[n] \leftarrow in[n]$; $out'[n] \leftarrow out[n]$[1]

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

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

until $in'[n] = in[n]$ and $out'[n] = out[n]$ **for all** n

Slika 2(Liveness analysis algoritam)

Nakon analize životnog veka na red dolazi kreiranje grafa smetnji i to tako što za svaku instrukciju napravimo smetnju izmedju svake promenljive koja je definisana unutar instrukcije i svih promenljivih iz out skupa, i vodi se računa da promenljiva ne bude u smetnji sama sa sobom.

Posle uspešnog kreiranja grafa smetnji vrši se dodeljivanje resursa registarskim promenljivima, i kao poslednji korak vrši se generisanje izlazne datoteke sa ekstenzijom ".s" koja sadrži preveden i korektan MIPS 32bit asemblerski jezik polaznog programa.

4. Programsko rešenje

4.1 Main

- `int main()`

Iz funkcije main() započinje izvršavanje programa. Nakon njenog pokretanja na konzoli će se tražiti da se napiše ime fajla koji se nalazi u folderu examples i koji želimo da prevedemo (npr. simple.mavn). Ukoliko sve prođe u redu poziva funkciju za kreiranje ".s" fajla, u suprotnom na konzoli se ispisuje greška.

4.2 SyntaxAnalysis

4.2.1 Konstruktori

- `SyntaxAnalysis(LexicalAnalysis& lex, Instructions& instructions);`

4.2.2 Funkcije

- `bool Do();` - započinje sintaksnu analizu
- `void createMIPS();` - kreira MIPS 32bit fajl
- `Variables& getRegVariabes();` - vraća listu registarskih promenljivih

4.2.3 Metode

- `void printSyntaxError(Token token);` - štampa poruku o grešci.
- `void printTokenInfo(Token token);` -štampa informacije o prosleđenom tokenu

- `void eat(TokenType t);` - "pojede" trenutni token ukoliko je tipa "t", u suprotnom prijavljuje sintaksnu grešku
- `Token getNextToken();` - vraća sledeći token iz liste tokena
- `void Q();` - neterminalni simbol
- `void S();` - neterminalni simbol
- `void L();` - neterminalni simbol
- `void E();` - neterminalni simbol
- `int findInstructionPosition();` - vraća poziciju prosleđene instrukcije. Koristi se priliko pronalaženja naslednika u slučaju branch instrukcije
- `void fillMemoryVarList(Token& t1, Token& t2);` - popunjava listu memorijskim promenljivima
- `void fillMemoryVarList(Token& t1);` - popunjava listu registarskim promenljivima
- `void fillFunctionList(string& name, int pos);` - popunjava listu funkcijama
- `void fillLabelList(string& label, int pos);` - popunjava listu labelama
- `void fillSuccesors();` - popunjava listu naslednicima(succ)
- `void fillPredecessor();` - popunjava listu prethodnicima(pred)
- `int getRegPosition(const string& regName);` - vraća poziciju registarske promenljive
- `string returnAssignedReg(string r);` - vraća ime dodeljenog registra(t0, t1, t2, t3)

4.3 IR

Ovde su sadržane 2 klase, Variable i Instruction

4.3.1 Klasa Variable

4.3.1.1 Konstruktori:

- `Variable();`
- `Variable(string name, int pos);`
- `Variable(string name, int pos, VariableType v_type, int val = 0);`

4.3.1.2 Atributi:

- `int mem_value;` - vrednost memorijske promenljive
- `VariableType m_type;` - tip `string m_name` – ime

- `int m_position;` - pozicija
- `Regs m_assignment;` - dodeljni registar

4.3.1.3 Metode:

- `string GetName();` - vraća ime promenljive
- `int getValue();` - vraća vrednost promenljive
- `int getPosition();` - vraća poziciju promenljive
- `Regs getAssignment();` - vraća dodeljeni registar
- `void setAssigment(Regs reg);` - dodeljuje registar promenljivoj

4.3.2 Klasa Instruction

4.3.2.1 Konstruktori:

- `Instruction();`
- `Instruction(int pos, InstructionType type, Variables& dst, Variables& src)`

4.3.2.2 Atributi:

- `int m_position;` - pozicija instrukcije
- `InstructionType m_type;` - tip instrukcije
- `Variables m_dst;`
- `Variables m_src;`
- `Variables m_use;`
- `Variables m_def;`
- `Variables m_in;`
- `Variables m_out;`
- `list<Instruction*> m_succ;`
- `list<Instruction*> m_pred;`

4.3.2.3 Metode:

- `InstructionType getType();` - vraća tip instrukcije
- `Int getPosition();` - vraća poziciju instrukcije;
- `Variables getDst();` - vraća dst listu
- `Variables getSrc();` - vraća src listu
- `Variables getUse();` - vraća use listu
- `Variables getDef();` - vraća def listu

- `Variables getIn();` - vraća in listu
- `Variables getOut();` - vraća out listu
- `list<Instruction*> getSucc();` - vraća succ listu
- `list<Insturction*> getPred();` - vraća pred listu
- `void setSucc(Instruction* succ);` - postavlja vrednost za succ
- `void setPred(Instruction* pred);` - postavlja vrednost za pred
- `void fillUseDefVariables();` - popunjava def i use listu

4.4 Livenes analysis

4.4.1 Funkcije

- `Void livenessAnalysis(Instructions instructions);` - vrši analizu životnog veka nad poslatim instrukcijama
- `Bool variableExists(Variable* variable, Variables variables);` - proverava da li se određena promenljiva već nalazi u skupu (ova funkcija se koristi za izračunavanje (out[n]-def[n]))

4.5 Interference graph

4.5.1 Klasa InterferenceGraph

4.5.1.1 Konstruktori:

- `InterferenceGraph(Variables& vars);`

4.5.1.2 Metode:

- `void ApplyRegToVariable(int varPos, Regs reg);` - dodeljuje pravi registar registarskoj promenljivoj
- `void buildInterferenceGraph(Instructions& instructions);` - kreira matricu smetnji
- `void buildVariableStack();` - kreira stek koji se puni svim registarskim promenljivima
- `void resourceAllocation();` - vrši dodelu resursa na osnovu matrice smetnji
- `void printInterferenceMatrix();` - štampa matricu smetnji

4.6 Lexical Analysis

4.6.1 Klasa LexicalAnalysis

4.6.1.1 Atributi:

- `ifstream inputFile;` - ulazni fajl koji sadrži tekst koji treba da se analizira
- `vector<char> programBuffer;` - ovde se nalazi sadržaj ulaznog fajla
- `unsigned int programBufferPosition;` - trenutna pozicija program buffera
- `FiniteStateMachine fsm;` - objekat automata sa konačnim brojem stanja
- `TokenList tokenList` – lista tokena
- `Token errorToken` – ukoliko dodje do greške ovde se nalazi token koji ju je izazvao

4.6.1.2 Metode:

- `void initialize();` - inicijalizuje leksičku analizu i automat sa konačnim brojem stanja
- `bool Do();` - metoda koja vrši leksičku analizu
- `bool readInputFile(string fileName);` - metoda koja čita iz ulaznog fajla
- `Token getNextTokenLex();` - vraća sledeći token
- `TokenList& getTokenList();` - vraća listu tokena
- `void printTokens();` - štampa listu tokena

5. Verifikacija

Nakon uspešnog kompajliranja i pokretanja, biće traženo od korisnika da u konzoli unese ime jednog od fajlova koji se nalaz u folderu examples\mavn_files (npr. simple.mavn). Nakon uspešno završenog prevođenja, ukoliko je to moguće, u folderu examples\mips_file pojaviće se “MIPSfile.s”. Ovaj MIPS 32bit fajl se može testirati emulatorom QtSpim.

U slučajevima da MAVN fajl nije moguće prevesti (iz razloga kao što su pogrešan unos imena fajla koji treba da se prevodi, korišćenje promenljivih koje nisu definisane, sintaksne greške...), na konzoli se ispisuje grška.