UNIVERZITET U NOVOM SADU
FAKULTET TEHNIČKIH NAUKA

UNIVERZITET U NOVOM SADU
FAKULTET TEHNIČKIH NAUKA
NOVI SAD
Departman za računarstvo i automatiku
Odsek za računarsku tehniku i računarske komunikacije

Projekat

Kandidat: **Mihailo Dikanović**
Broj indeksa: **RA 84/2021**

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

Asistent: **MSc Milorad Marković**
Profesor: **dr Miodrag Đukić**

Novi Sad, jun, 2023.

Sadržaj

1 Uvod

1.1 MAVN prevodilac

MAVN prevodilac prevodi sa višeg asemblerskog jezika na osnovni asemblerski jezik-MIPS 32bit. Viši asemblerski jezik je spečifican po tome što može da koristi registarske promenjive što omogućava korišćenje promenjivih umesto pravih resursa.

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 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

NAPOMENE:
1. Prilikom prevođenja omogućiti detekciju i reagovanje 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.

1.3 Rešenje problema

Da bi se datoteka pisana na MAVN jeziku prevede na MIPS 32bit asembler, potrebno je izvršiti sledeće korake:

1. Lexical analysis
2. Syntax analysis
3. Liveness analysis
4. Kreiranje grafa smetnji
5. Faza uprošćavanja
6. Dodela resursa
7. Kreiranje MIPS fajla

2 Analiza problema

Sam MAVN prevodilac nudi izbor deset podržanih 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

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 promeljive:

_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.

Sintaksa MAVN jezika opisana je gramatikom:

$Q \rightarrow S\ ;\ L$

$S \rightarrow \_mem\ mid\ num$

$S \rightarrow \_reg\ rid$

$S \rightarrow \_func\ id$

$S \rightarrow id:\ E$

$S \rightarrow E$

$L \rightarrow eof$

$L \rightarrow Q$

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

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

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

$E \rightarrow la\ rid,\ mid$

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

$E \rightarrow li\ rid,\ num$

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

$E \rightarrow b\ id$

$E \rightarrow bltz\ rid,\ id$

$E \rightarrow nop$

Da bi se traženi zahtev programa realizovao potrebno je implementirati odgvarajuće faze u prevođenju izvornog koda i to:

- leksičku analizu
- sintaksnu analizu
- analizu životnog veka promenljivih
- dodelu resursa

3 Koncept rešenja

3.1 Sistem

Main modul sadrži funkcije main(), analysis() i ispis(). Pozivom funkcije analysis() u mainu, izvršavamo redom sve korake, počevši od leksičke analize zaključno sa dodelom resursa. *LexicalAnalysis* modul, kao što mu naziv kaže, vrši samu leksičku proveru ispravnosti programa. Potom prepušta kontrolu *SyntaxAnalysis* modulu koji vrši sintaksnu proveru. Nakon toga, poziva se *LivenessAnalysis* u kom se prvenstveno formira graf toka upravljanja, tačnije, popunjavaju se odgovarajući skupovi instrukcija (*pred* i *succ*) čija se semantika ogleda upravo u čvorovima grafa tj. granama između tih čvorova, a nakon toga se vrši analiza životng veka promenljivih na osnovu koje se kreiraju odgovarajući skupovi instrukcija (*in* i *out*) pomoću kojih se dalje u *InterferenceGraph*-u kreira graf smetnji kao prva faza dodele resursa, nakon koje će se obaviti i uprošćavanje grafa smetnji putem modula *SimplificationStack*, kao druga faza dodele resursa. Nakon toga vrši se dodela resursa promenljivim putem modula *ResourceAllocation*. Nakon pozivanja funkcije analysis() u mainu, pozivamo funkciju ispis() koja će izvršiti ispis instrukcija i formirati izlaznu datoteku sa određenom ekstenzijom “.s”.

3.2 Način pokretanja programa

Kada pokrenemo program, konzola će od nas tražiti da unesemo naziv fajla zajedno sa ekstenzijom (.mavn) koji želimo da prevodimo. Nakon što unesemo naziv i pritisnemo enter program počinje sa svojim izvršavanjem.

4 Opis rešenja

4.1 Main

```
int main();
void Analysis(in, Variables& v, Labels& l, Instructions& i);
void Ispis(out, Variables& v, Labels& l, Instructions& i);
```

Parametri:

- Variables& v – referenca na varijable koje ce biti napravljene u main funkciji
- Labels& l – referenca na labele koje ce biti napravljene u main funkciji
- Instructions& i – referenca na instrukcije koje ce biti napravljene u main funkciji
- in – naziv ulazne datoteke
- out – naziv izlazne datoteke

4.2 LexicalAnalysis

Metode:

- `bool readInputFile(string fileName);`
- `void initialize();`
- `bool Do();`

4.3 SyntaxAnalysis

Atributi:

- `int varPozicija;`
- `LexicalAnalysis& lexicalAnalysis;`

Metode:

- `SyntaxAnalysis(LexicalAnalysis& lex);`
- `bool Do();`
- `void printSyntaxError(Token& token);`
- `void eat(TokenType t);`
- `void Q();`
- `void S();`
- `void L();`
- `void E();`
- `Variable* getVariable(string name);`
- `void printVariables(Variables& varijable);`

4.4 LivenessAnalysis

Metode:

- `void fillSuccPred();`
- `void fillDefUse ();`
- `bool isEqual(Variables first, Variables second);`
- `void livenessAnalysis();`

4.5 InterferenceGraph

Atributi:

- `Variables* variables;`
- `InterferenceMatrix matrix;`

Metode:

- `InterferenceGraph(){}`
- `InterferenceGraph(Variables* v): variables(v){}`
- `void printInterferenceMatrix();`
- `makeInterferenceGraph(Variables& v, Instructions& i)`

4.6 SimplificationStack

Metode:

- `SimplificationStack* doSimplification(InterferenceGraph ig, int degree);`

Strukture:

- `struct NotEnughRegisters: std::runtime_error`

4.7 ResourceAllocation

Metode:

- `bool doResourceAllocation(SimplificationStack ss, InterferenceGraph ig);`
- `bool checkResourceAllocation(InterferenceGraph ig);`

4.8 IR

4.8.1 Variable

Atributi:

- `int m_position;`
- `VariableType m_type;`
- `string m_name;`
- `string m_value;`
- `Regs m_assignment;`

Metode:

- `string typeToS(VariableType vt);`
- `string regsToS(Regs r);`

4.8.2 Label

Atributi:

- `string name;`

4.8.3 Instruction

Atributi:

- `int m_position;`
- `int m_konstanta;`
- `string m_labela;`
- `string m_jump`
- `InstructionType m_type;`
- `Variables m_dst;`
- `Variables m_src;`
- `Variables m_use;`
- `Variables m_def;`
- `Variables m_in;`
- `Variables m_out;`
- `std::list<Instruction*> m_succ;`
- `std::list<Instruction*> m_succ;`

Metode:

- `Instruction () : m_position(0), m_type(I_NO_TYPE) {}`
- `Instruction(string lab, int pos, InstructionType type, Variables& dst, Variables& src) : m_position(pos), m_labela(lab), m_type(type), m_dst(dst), m_src(src) {}`
- `Instruction (string lab, int pos, int c, InstructionType type, Variables& dst, Variables& src): m_position(pos), m_labela(lab), m_type(type), m_dst(dst), m_src(src), m_konstanta(c) {}`
- `Instruction(string lab, int pos, string s, InstructionType type, Variables& dst, Variables& src) : m_position(pos), m_labela(lab), m_type(type), m_dst(dst), m_src(src), m_jump(s) {}`
- `string printInstruction();`

5 Testiranje

simple.mavn ==>

```
_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;
```

Izvršavanje programa ==>

```
Microsoft Visual Studio Debug Console
Interference matrix:
=========================================
       r1      r2      r3      r4      r5
r1      0       1       0       0       1
r2      1       0       0       0       0
r3      0       0       0       0       0
r4      0       0       0       0       0
r5      1       0       0       0       0
=========================================


Resource allocation is successfull!


=========================================
Variables after resource allocation:
=========================================
[NAME] m1 [TYPE] MEM_VAR [ASSIGNMENT] no_assign
[NAME] m2 [TYPE] MEM_VAR [ASSIGNMENT] no_assign
[NAME] r1 [TYPE] REG_VAR [ASSIGNMENT] $t0
[NAME] r2 [TYPE] REG_VAR [ASSIGNMENT] $t1
[NAME] r3 [TYPE] REG_VAR [ASSIGNMENT] $t0
[NAME] r4 [TYPE] REG_VAR [ASSIGNMENT] $t0
[NAME] r5 [TYPE] REG_VAR [ASSIGNMENT] $t1
=========================================


C:\Users\Mihailo\Desktop\OPPiSA projekat\MihailoDikanovicRA084\Debug\LexicalAnalysis.exe (process 8840) exited with code
 0.
Press any key to close this window . . .
```

simple.s ==>

```
.globl main

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

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

multiply.mavn ==>

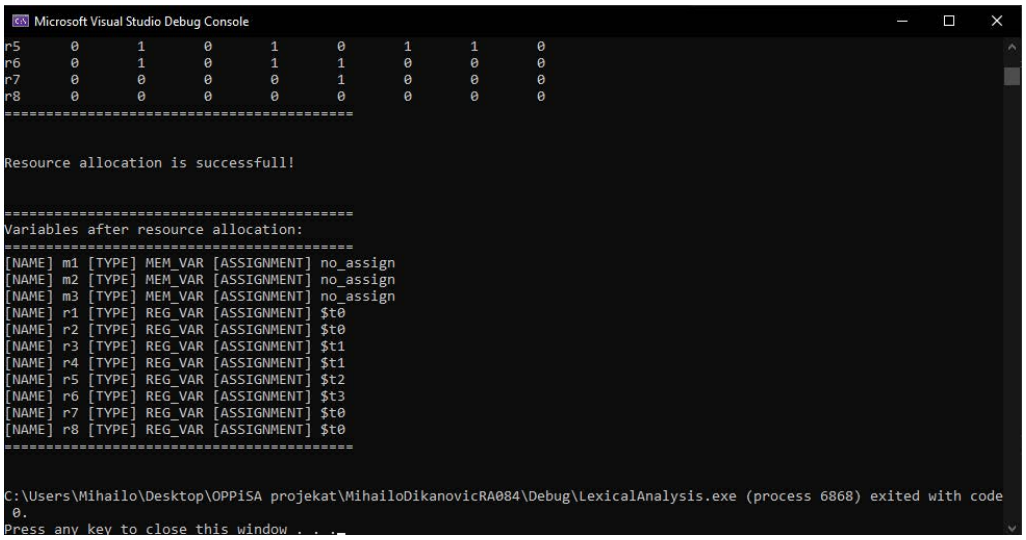
```
_mem m1 6;
_mem m2 5;
_mem m3 0;

_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;
```

Izvršavanje programa ==>



multiply.s ==>

```
.globl main

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

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