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: **Исидора Станић**
Број индекса: **SW 16/2018**

Предмет: **Системска програмска подршка 1**
Тема рада: **MABH - Преводилац**

Ментор рада: **проф. др Миодраг Ђукић**

Нови Сад, јун, 2020.

Садржај

1. Увод

1.1 Анализа проблема

МАВН (Мипс Асемблер Високог Нивоа) је алат који преводи програм написан на вишем MIPS 32bit асемблерском језику на основни асемблерски језик. Виши MIPS 32bit асемблерски језик служи лакшем асемблерском програмирању, јер уводи концепт **регистарске** променљиве. Регистарске променљиве омогућавају програмерима да приликом писања инструкција користе променљиве уместо правих ресурса. Ово знатно олакшава програмирање јер програмер не мора да води рачуна о коришћеним регистрима и њиховом садржају.

Потребно је реализовати МАВН преводилац који преводи програме са вишег асемблерског језика на основни MIPS 32bit асемблерски језик. Преводилац треба да подржава детекцију лексичких, синтаксних и семантичких грешака, као и генерисање одговарајућих извештаја о евентуалним грешкама. Излаз из преводиоца треба да садржи коректан асемблерски код који је могуће извршавати на MIPS 32bit архитектури (симулатору).

МАВН језик подржава 10 MIPS инструкција, а то су:

add – (addition) сабирање

addi – (addition immediate) сабирање са константом

b – (unconditional branch) безусловни скок

bltz – (branch if less than zero) скок ако је регистар мањи од нуле

la – (load address) учитавање адресе у регистар

li – (load immediate) учитавање константе у регистар

lw – (load word) учитавање једне меморијске речи

nop – (no operation) инструкција без операције

sub – (subtraction) одузимање

sw – (store word) упис једне меморијске речи

and – логичко и

or – логичко или

lb – (load byte) учитавање једног бајта са адресе у регистар

Синтакса МАВН језика описана је граматиком:

$Q \rightarrow S\ ;\ L$

$S \rightarrow \_mem\ mid\ num$
$S \rightarrow \_reg\ rid$
$S \rightarrow \_func\ id$
$S \rightarrow id:\ E$
$S \rightarrow E$

$L \rightarrow eof$
$L \rightarrow Q$

$E \rightarrow add\ rid,\ rid,\ rid$
$E \rightarrow addi\ rid,\ rid,\ num$
$E \rightarrow sub\ rid,\ rid,\ rid$
$E \rightarrow la\ rid,\ mid$
$E \rightarrow lw\ rid,\ num(rid)$
$E \rightarrow li\ rid,\ num$
$E \rightarrow sw\ rid,\ num(rid)$
$E \rightarrow b\ id$
$E \rightarrow and\ rid,\ rid,\ rid$
$E \rightarrow or\ rid,\ rid,\ rid$
$E \rightarrow lb\ rid,\ num(rid)$

Терминални симболи МАВН језика су:

: ; , ( )

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

1.2 Концепт решења

Преводилац треба да прође кроз неколико фаза.

Прва фаза је лексичка анализа. Поштујући дата правила, проверава се исправност сваког унетог симбола и речи у улазној датотеци. Притом се генеришу токени који се користе касније у синтаксној анализи.

Након успешно извршене лексичке анализе реализује се синтаксна анализа. Ту се проверава синтакса MAVH језика која је описана датом граматиком. У оквиру синтаксне анализе врши се и прикупљање променљивих, лабела и функција које програм користи. Лабеле и функције имају своју позицију која се поклапа са позицијом прве инструкције на коју показују.

Затим се генеришу инструкције у структуру са којом је лакше радити, након чега се генерише граф инструкција. Инструкције се увезују оним редом којим су генерисане, а у случају скока увезују се инструкција која „скаче“ и инструкција чија позиција се поклапа са позицијом лабеле.

Након тога врши се анализа животног века променљивих коришћењем претходно дефинисаног графа инструкција.

Затим следи додела ресурса променљивима. Прва фаза доделе ресурса је генерисање матрице сметњи на основу анализе животног века променљивих. На основу матрице сметњи одређују се степени чворова графа, као број чворова са којима је посматрани чвор у сметњи. На основу степена чворова се променљиве смештају на стек у фази упрошћавања. Смештају се тако да се прво са графа скида чвор са највећим степеном мањим од броја регистара, при чему се смањује ранг свим његовим суседима. Поступак се понавља док све променљиве не буду на стеку. Након тога се коришћењем генерисаног стека врши додела регистара променљивама, тзв. бојење графа. Притом се чворови „боје“ (додељују им се регисти) редоследом којим се скидају са стек у претходној фази. Чвору се не сме доделити регистар који је већ додељен неком суседном чвору.

Након што су све ове фазе извршене превођење се завршава уписом у одабрану датотеку са екстензијом *.s*, са одговарајућим заглављима и поштујући правила записа инструкција.

2. Програмско решење

2.1 LexicalAnalysis

Модул за извршавање лексичке анализе програма.

2.2 SyntaxAnalysis

Модул за извршавање синтаксне анализе програма.

Функције:

`void printVariables(), void printLabels(), void printFunctions()` – штампају листе променљивих, лабела и функција, по њиховим називима.

`Variable* findVariable(string name)` – тражи променљиву у листи променљивих.

Параметри:

string name – име променљиве

Повратна вредност:

*Variable** – показивач на нађену променљиву или NULL ако није пронађена

`Label* findLabel(string name)` – тражи лабелу у листи лабела.

Параметри:

string name – име лабеле

Повратна вредност:

*Label** – показивач на нађену лабелу или NULL ако није пронађена

`Func* findFunc(string name)` – тражи функцију у листи функција..

Параметри:

string name – име функције

Повратна вредност:

*Func** – показивач на нађену променљиву или NULL ако није пронађена

`void printSyntaxError(Token token)` – исписује синтаксну грешку и токен који је ту грешку проузроковао.

Параметри:

Token token – токен на којем је дошло до синтаксне грешке

`void eat(TokenType t)` – проверава да ли је тренутни токен прослеђеног типа и пријављује грешку уколико није. Такође у њој се генерише функција, лабела или променљива и смешта у одговарајући привремени показивач ако се наиђе на дефиницију. Проверава да ли је променљива додата у листу ако се користи у било којој инструкцији која није дефиниција. Ако не постоји, пријављује грешку.

Параметри:

TokenType t – очекивани тип токена

`Token getNextToken()` – враћа следећи токен из листе токена генерисаних у лексичкој анализи.

`bool Do()` – покреће проверу синтаксе. Унутар ње се рекурзивно позивају следеће 4 функције.

Повратна вредност:

true ако није дошло до синтаксне грешке, *false* ако јесте

`void Q()` – дефинише граматику.

`void S()` – дефинише граматику.

`void L()` – дефинише граматику.

`void E()` – дефинише граматику.

2.3 Program

Модул за генерисање инструкција, њихово увезивање, креирње графа сметњи, анализе животног века и доделу ресурса.

Функције:

`void makeInstructions()` – поново пролази кроз токене и генерише листу инструкција.

`void managmentFlowGraph()` – пролази кроз генерисане инструкције и увезује их по редоследу генерисања и по скоковима на лабеле, стварајући граф инструкција.

`void livenessAnalysis()` – одређује животни век сваке регистарске променљиве на основу графа инструкција.

`void buildInterferenceGraph()` – попуњава матрицу(граф) сметњи на основу животног века променљивих.

`bool haveInterference(Variable& v1, Variable& v2)` – помоћна функција која проверава да ли две променљиве имају сметњу међу собом.

Параметри:

Variable& v1, *Variable& v2* – променљиве за које се тражи сметња

Повратна вредност:

true ако постоји сметња, *false* ако не постоји

`stack<Variable*>* doSimplification(int degree)` – „скида“ чворове са графа сметњи по степену и смешта их на стек (Фаза упрошћавања).

Параметри:

int degree – број доступних регистара за доделу

Повратна вредност:

stack<Variable>** – показивач на стек попуњен променљивама

`bool doResourceAllocation(stack<Variable*>* simplificationStack)` – додељује регистре променљивама скидајући их редом са стека, при чему не сме да додели исти регистар који је додељен променљивој у суседном чвору. Ако не може да додели ниједан регистар некој променљивој, прекида извршавање које се касније тумачи као преливање.

Параметри:

stack<Variable>* simplificationStack* – упрошћени стек са променљивама довијен у оретходној функцији

Повратна вредност:

true ако су успешно додељени регистри свим променљивама, *false* ако је дошло до преливања(нека променљива није могла да добије ниједан регистар)

2.4 Print

Модул за штампање у конзолу и упис преведеног кода у излазну датотеку.

`string getRegister(Variable *v)` – помоћна функција која прави string који представља додељени регистар прослеђеној променљивој.

Параметри:

*Variable *v* – променљива која се замењује додељеним регистром

Повратна вредност:

string – назив додељеног регистра

`string getInstruction(Instruction *i)` – помоћна функција која прави string који представља инструкцију након превођења и доделе ресурса.

Параметри:

*Instruction *i* – инструкција која се преводи

Повратна вредност:

string – преведена инструкција

`void printProgram(Program& p, string filename)` – уписује преведен код у излазну датотеку.

Параметри:

Program& p – преведени програм

string filename – назив датотеке(релативна адреса)

`void printInstructionsPos(Instructions& instrs)` – штампа позицију инструкције.

Параметри:

Instruction& i – инструкција

`void printVariablesNames(Variables& variables)` – штампа назив променљиве.

Параметри:

Variable& v – променљива

`string stringType(InstructionType t)` – враћа назив типа инструкције.

Параметри:

InstructionType t – тип инструкције

`void printInstruction(Instruction* instr)` – штампа позицију, тип, листе предака, наследника, дефинисаних и коришћених променљивих, живих на улазу и излазу инструкције.

Параметри:

Instruction i* – инструкција

`void printInterferenceMatrix(InterferenceGraph& ig)` – штампа граф сметњи.

Параметри:

InterferenceGraph& ig – граф сметњи

3. Верификација

Исправност превођења је проверена на неколико улазних датотека. Прва улазна датотека за коју је преводилац тестиран је *simple.mavn*, датотека која је дата као пример при поставци задатка.

```
1   _mem m1 6;
2   _mem m2 5;
3
4   _reg r1;
5   _reg r2;
6   _reg r3;
7   _reg r4;
8   _reg r5;
9
10  _func main;
11      la      r4, m1;
12      lw      r1, 0(r4);
13      la      r5, m2;
14      lw      r2, 0(r5);
15      add     r3, r1, r2;
16
```

Слика улазне датотеке *simple.mavn*

```
C:\WINDOWS\system32\cmd.exe
Lexical analysis finished successfully!
Syntax analysis finished successfully!
Instructions made successfully!
Managment flow graph made successfully!
Liveness analysis finished successfully!
Interference graph made successfully!
Simplification stack made successfully!
Resource allocation finished successfully!
Press any key to continue . . .
```

Слика исписа у конзоли

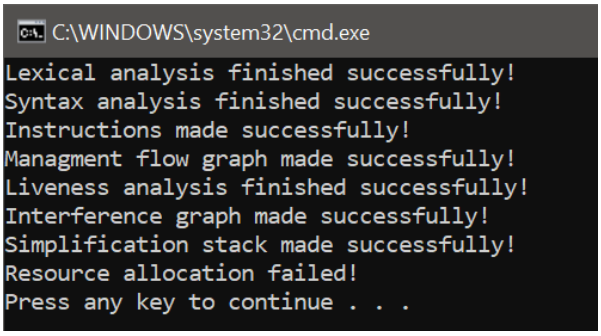
```
.globl main

.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

```

Слика преведеног кода

Друга улазна датотека је *multiply.mavn*, датотека која је такође дата као пример при поставци задатка.

```
_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;


```

Слика улазне датотеке *multiply.mavn*

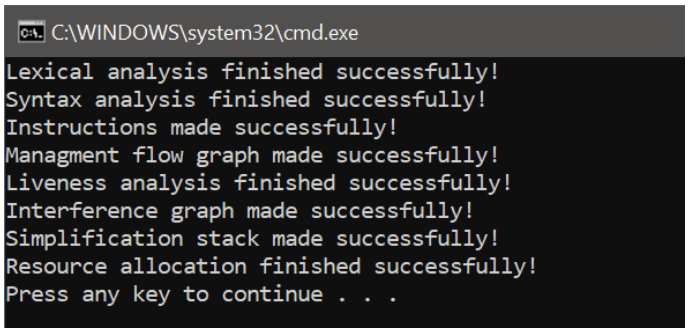


Слика исписа у конзоли

Напомена: Десило се преливање у овом случају, јер преводилац није могао да додели ниједан регистар некој променљивој. Ово може да се превазиђе уколико се повећа број регистара који могу да се доделе(__*REG_NUM*__ се дефинише као 5). Резултат превођења у том случају је:

```
1   .globl main
2
3   .data
4   m1:     .word  6
5   m2:     .word  5
6   m3:     .word  0
7
8   .text
9   main:
10      la      $t0, m1
11      lw      $t3, 0($t0)
12      la      $t0, m2
13      lw      $t4, 0($t0)
14      li      $t1, 1
15      li      $t2, 0
16  lab:
17      add     $t2, $t2, $t3
18      sub     $t0, $t1, $t4
19      addi    $t1, $t1, 1
20      bltz    $t0, lab
21      la      $t0, m3
22      sw      $t2, 0($t0)
23      nop
24
```

Слика преведеног кода



Слика исписа у конзоли

Тестови на које преводилац треба да баци грешку:

TestInstrNotComplete.mavn – инструкција није дефинисана до краја

TestMemVarNotM.mavn – меморијска променљива није у формату m + број

TestRegVarNotR.mavn – регистарска променљива није у формату r + број

TestUndefMemVar.mavn – меморијска променљива није дефинисана

TestUndefRegVar.mavn – регистарска променљива није дефинисана

TestUndefLabel.mavn – скаче се на лабелу која није дефинисана

TestNoMain.mavn – функција main није дефинисана