Избор инструкција

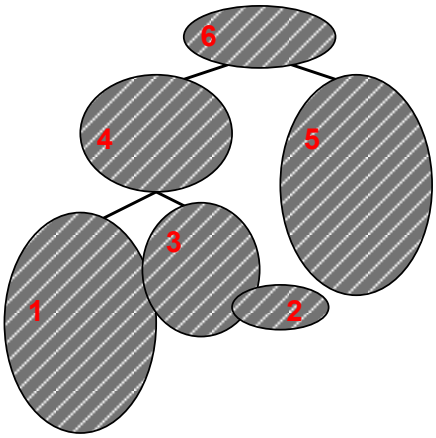
Стабло међукода описује једну основну операцију по чвору, нпр. сабирање, одузимање, писање и читање из меморије, итд. Инструкције на реалној архитектури често могу одједном да изврше више ових основних операција. У последњем реду табеле 1 се налази пример стабла са 3 чвора: move, mem и бинарна операција сабирања. Цело ово стабло је могуће записати помоћу само једне МИПС инструкције: sw r1, c(r2).

Задатак фазе избора инструкција је проналажење одговарајућег скупа инструкција којим се описује цело стабло међукода. Свака инструкција се може записати помоћу малог стабла - шаблона. У табели 1 су приказане основне МИПС инструкције као и одговарајући шаблони. Као што се из табеле може приметити, једна МИПС инструкција описује више чворова стабла међукода.

Табела 1 Основне МИПС инструкције и њихови шаблони у стаблу међурепрезентације

Инструкција	*Шаблон*	*Инстукција*	*Шаблон*	*Инстукција*	*Шаблон*
add ri, rj, rk	+	addi ri, rj, c	+ const, + const	lw ri, c(rj)	mem + const, mem + const, mem const, mem
Инструкција	**Шаблон**	**Инстукција**	**Шаблон**	**Инстукција**	**Шаблон**
sub ri, rj, rk	-	li ri, c	const	sw ri, c(rj)	move mem + const, move mem + const, move mem const, move mem

На следећој слици је приказан пример избора инструкција. Са леве стране је приказано стабло међукода, док је са десне стране списак МИПС инструкција. Овде се може видети да је регистар 1 добио ресурс t1, регистар 2 ресурс t3 и регистар 3 ресурс t2.



Слика 1 Пример поплочавања стабла међукода

1. lw $t1,20($t1)
2. li $t0, 15
3. sub $t0,$t3,$t0
4. add $t0,$t0,$t1
5. lw $t4,10($t2)
6. sw $t4,0($t0)

Алгоритме за избор инструкција упоређујемо на основу цене скупа инструкција које праве. Један критеријум за одређивање цене скупа инструкција је величина скупа: што је скуп већи, већа је и цена. Уколико време извршавања инструкција није једнако, други критеријум би био скуп инструкција који се најкраће извршава. Из овога дефинишемо два појма: најповољније поплочавање и оптимално поплочавање. Под најповољнијим поплочавањем подразумева се скуп инструкција чија је цена најмања, док се под оптималним поплочавањем подразумева скуп инструкциа у коме се суседне инструкције не могу спојити у нову инструкцију и тако добити скуп инструкција са нижом ценом. Један од алгоритама за избор инструкција са најповољнијим поплочавањем је Манч алгоритам.

Манч алгоритам

Манч алгоритам је једноставан алгоритам за избор инструкција. Он креће од почетног чвора стабла међукода и тражи инструкцију која покрива највећи део стабла. На тај начин покриће се почетни чвор и неколико суседних чворова. Затим се алгоритам понавља за сваки преостали чвор. На основу сваког покривања стабла се праве инструкције. Манч алгоритам прави инструкције у супротном редоследу од поплочавања, тј. инструкције које се односе на листове стабла праве се прве, иако се последње поплочавају. Са слике 1 се може видети да је почетни чвор move али да би се направила његова инструкција (6) морају се направити друге инструкције (4, 5); и даље: да би се направила инструкција 4, морају се прво направити инструкције 1 и 3, а инструкција 3 тек после инструкције 2. **Питање:** Да ли у овом примеру инструкције могу бити генерисане у неком другачијем редоследу? Колико различитих могућих ваљаних редоследа генерисања инструкција постоји у овом примеру?

Реализација

Стабло међукода

Стабло међукода је описано у датотеци Tree.h. Стабло је описано преко исказа и израза. Разлика између њих је та да изрази крајњи резултат смештају у неки операнд, а искази не. На програм се гледа као на низ исказа, где су неки искази у ствари изрази (врста исказа EXP_STM у табели 2). На примеру програмског језика Це можемо потврдити овај концепт, јер се може сматрати да свака наредба која се завршава са „;“ представља исказ, па би тако наредба у којој се налази само израз била управо израз који је у ствари исказ (насупрот томе треба приметити да израз може бити део исказа).

Искази (енгл. statement) су описани у класи `Statement` док су изрази (енгл. expression) описани у класи `Expression`. За све описане подтипове израза и исказа су реализоване изведене класе (такође описане у датотеци Tree.h).

Табела 2 Типови исказа и израза

Тип исказа	Објашњење	Тип израза	Објашњење
`MOVE_STM`	Садржи два операнда и користи се за пребацивање садржаја из једног у други операнд.	`BINOP_EXP`	Садржи два операнда и операцију плус, минус, итд.
`SEQ_STM`	Користи се уколико је потребно описати два исказа у низу.	`ESEQ_EXP`	Користи се за опис израза који садржи по један израз и исказ.
`EXP_STM`	Користи се уколико исказ треба да опише само један израз.	`MEM_EXP`	Користи се за операцију приступа меморији. У зависности од тога са које стране исказа се израз налази користи се за: писање у меморију (одредишни операнд у исказу) или читање из меморије (изворни операнд у исказу).
		`REG_EXP`	Користи се за опис израза који садржи само један регистар.
		`CONST_EXP`	Користи се за опис израза који садржи само једну константу.

Сваки тип исказа и израза поседује одговарајуће конструкторе уз помоћ којих се на једноставан начин може изградити стабло међукода. Имајући ово у виду, стабло међукода програма са слике 1 се може формирати на следећи начин:

```
Statement* program = new S_Move(

        new E_Mem(new E_Binop(E_Binop::PLUS_OP,

                new E_Mem(new E_Binop(E_Binop::PLUS_OP, new E_Const(20), new E_Reg(reg1))),

                new E_Binop(E_Binop::MINUS_OP, new E_Const(15), new E_Reg(reg2)))),

        new E_Mem(new E_Binop(E_Binop::PLUS_OP, new E_Const(10), new E_Reg(reg3)))

);
```

Инструкције

МИПС Инструкције су описане у датотекама Instruction.h и Instruction.cpp. Свака инструкција има листе одредишних и изворишних операнада (регистара) које могу бити и празне у зависности од типа инструкције (учитавање константе нема изворишне операнде). Поред тога инструкције садрже и симболички запис асемблерске инструкције у ком фигуришу одредишни и изворишни операнди у облику:

```
add `d, `s, `s
```

где се накнадно изврши замена симбола које означавају одредишне операнде „d“ (енгл. destination) и изворишне операнде „s“ (енгл. source) са стварним операндима из поменутих листа. Због тога је неопходно да листа одредишних регистара има тачно онолико елемената колико у формату инструкције има специјалних симбола „`d“. Исто важи и за листу са изворишним регистрима. Класа такође располаже функцијом `toString` која добавља текстуални облик целе инструкције.

Манч алгоритам

Манч алгоритам је описан у датотекама Munch.h и Munch.cpp. Ту се налазе функције `munchStm` и `munchExp` које се користе за поплочавање исказа и израза. У овим инструкцијама обавља се поплочавање међукода МИПС инструкцијама. Инструкције се праве помоћу конструктора класе `Instruction` и функције `emit` која инструкције смешта у излазну листу инструкција. Следи пример за инструкцију addi.

```
string instructionString = "addi `d,`s, " + toString(constExp->getValue());

Reg srcReg = munchExp(binopRightExp);
Reg dstReg = getNewReg();

Instruction* instr = new Instruction(instructionString);
instr->getDst().push_back(dstReg);
instr->getSrc().push_back(srcReg);

emit(instr);
```

Прво се формира симболички запис инструкције где се за одредишне и изворишне операнде, које тек треба одредити, поставе одговарајући симболи *`d* и *`s* који ће бити накнадно замењени. Затим се одређују изворишни и одредишни операнди. Ова инструкција има један изворишни (srcReg) и један одредишни (dstReg) операнд. Функција *getNewReg()* се користи за добављање новог јединственог регистра и користи се за одредишни операнд - операнд у који се резултат ове инструкције смешта. За изворишни операнд (цео израз) је потребно поново позвати функцију `munchExp` ради његове евалуације.

Позивом функције `emit` додаје се нова инструкција у листу инструкција `selectedInstructions`. Овој листи инструкција могуће је приступити преко следеће функције:

```
InstructionList& getInstructionList()
```

ЗАДАТАК

1. У пропратном пројекту урађени су случајеви сабирања константе са изразом и израза са константом (горњи део табеле 1, други шаблон, наранџасте боје). Покрити случај сабирања два израза (горњи део табеле 1, први шаблон, плаве боје).

2. Додати lw МИПС инструкцију (горњи део табеле 1, трећи шаблон, зелене боје) у Манч алгоритам и тестирати је. Место на које треба додати инструкцију је означено коментаром у коду.