УНИВЕРЗИТЕТ У БЕОГРАДУ
МАТЕМАТИЧКИ ФАКУЛТЕТ

Александра Караџић

# АЛАТ *VALGRIND* - ИМПЛЕМЕНТАЦИЈА КОНВЕНЦИЈЕ *FPXX* ЗА АРХИТЕКТУРУ *MIPS*

мастер рад

Београд, 2017.

**Ментор**:

др Милена Вујошевић Јаничић, доцент
Универзитет у Београду, Математички факултет

**Чланови комисије**:

др Филип Марић, ванредни професор
Универзитет у Београду, Математички факултет

др Јелена Граовац, доцент
Универзитет у Београду, Математички факултет

**Датум одбране**: 15. јануар 2016.

*Деди*

**Наслов мастер рада**: Алат *Valgrind* - имплементација конвенције *FPXX* за архитектуру *MIPS*

**Резиме**:

**Кључне речи**: анализа, геометрија, алгебра, логика, рачунарство, астрономија

# Садржај

# Глава 1

# Увод

# Глава 2

# Архитектура *MIPS*

### MIPS

MIPS је најелегантнија архитектура међу свим активним RISC архитектура, чак и по мишљењу конкуренције. Елеганција није довољна да би се освојило тржиште, али MIPS микропорцесори су успели бити међу најефикаснији сваком генерацијом остајући међу најједноставнијима. MIPS процесори је један од RISC процесора, рођеног у плодном периоду академских истраживања и развоја. RISC (енг. *Reduced Instruction Set Computing*) је атрактивни акроним, [3]

## 2.1 Регистри у MIPS-у

Регистри представљају малу, веома брзу меморију, која је део процесора. MIPS процесори могу вршити операције само над садржајима регистара и специјалним константама које су део инструкције.

У MIPS архитектури, постоји 32 регистара опште намене. Само два регистара се понашају другачије од осталих регистара:

- **$0** - Увек враћа нулу, без обзира коју му се вредност додели
- **$31** - Увек се користи за адресу повратка из функције на коју се скочи инструкцијом *jal*

Сви ови регистри су идентични и могу се користити за било коју истрикцију (може се чак користити и регистар $0 као дестинација, мада ће резултат да нестане).

Регистри опште намене су описани у наставку:

- **at** - Резервисан за псеудоинструкције које асемблер генерише
- **v0, v1** - Користи се за враћање резултата при повратку из неке функције. Резултат може бити целобројног типа или број записан у фиксном зарезу.
- **a0 - a3** - Користи се за прослеђивање прва 4 аргумената функције која се позива
- **t0 - t9** - по конвенцији која је описана
- **s0 - s7** - по конвенцији која је описана
- **k0, k1** - Резервисано за систем прекида, који након коришћења не враћа садржај ових регистара на почетни. Како се прекид не позива из програма који се тренутно извршава, нема примене позивне конвенције. То значи да се садржај регистара које прекинути програм користи може пореметити. Због тога, систем прекида прво сачува саджаје регистара опште намене, који су важни за програм који се у том тренутку извршавао, и чији садржај планира да мења. У те сврхе се користе ови регистри.
- **gp** - Користи се у различите сврхе. У коду који не зависи од позиције (енг. *Position Independent Code* скраћено PIC), свом коду и подацима се приступа преко табеле показивача, познате као GOT (скраћено од енг. *Global Offset Table*). Регистар *$gp* показује на ту табелу. PIC је код који се може извршавати на било којој меморијској адреси, без модификација. PIC се најчешће користи за дељење библиотеке, при чему се заједнички код библиотеке може учитати у одговарајуће локације адресних простора различитих програма који је користе.
  У регуларном коду који зависи од позиције, регистар **$gp** се користи као показивач на средину у статичкој меморији. То значи да се подацима који се налазе 32 KB лево или десно од адресе која се налази у овом регистру може приступити помоћу једне инструкције. Дакле, инструкције *load* и *store* које се користе за учитавање, односно складиштење података, се могу извршити у само једној инструкцији, а не у две као што је иначе случај. У пракси се на ове локације смештају глобали подаци који не заузимају много меморије. Оно што је битно је да овај регистар не користи сви системи за компилацију и сва окружења за извршавање.

- **sp** - Показивач на стек. Оно што је битно је да стек расте наниже. Потребне су специјалне инструкције да би се показивач на стек повећао и смањио, тако да *MIPS* ажурира стек само при позиву и повратку из фукције, при чему је за то одговорна функција која је позвана. *sp* се при уласку у функцију прилагођава на најнижу тачку на стеку којој ће да приступати у функцији. Тако ј еомогућено да компилатор може да приступи поменљивама на стеку помоћу константног помераја у односу на *$sp*.

- **fp** - Познат и као *$s8*, показивач на стек оквир. Користи се од стране функције, за праћење стања на стеку, за случај да из неког разлога компилатор или програмер не могу да израчунају померај у односу на *$sp*. То се може догодити уколико програм врши проширење стека, при чему се вредност проширења рачуна у току извршавања. Ако се дно стека не може израчунати у току превођења, не може се приступити променљивама помоћу *$sp*, па се на почетку функције *$fp* иницијализује на константну позицију која одговара стек оквиру функције. Ово је локално за функцију.

- **ra** - Ово је подразумевани регистар за смештање адресе воратка и то је подржано кроз одговарајуће инструкције скока. Ово се разлицкује од конвеција које се корсите на архитеткурама ц86, где инструкција позива функције адресз повратка смешта на стек. При уласку у фукцију регистар *ra* обично садржи адресу повратка фукције, тако да се функције углавном завршавају инструкцијом *jr $ra*, али у принципу, може се користити и неки други регистар. Због неких оптимизација које врши процесор, препоручује се коришћење регистара *$ra*. Функције које позивају друге функције морају сачувати садржај регистара *$ra*.

Постоје два специјална регистра *Hi* и *Lo*, који се користе само при множењу и дељењу. ово нису регистри опште намене, те се не користе при другим инструцијама. Не може им се приступити директно, већ постоје специјалне инструкције *mfhi* и *mflo* за премештање садршаја ових регистара. Инструкција *mfhi* је облика *mfhi rd*, и она премешта садржај регистар *Hi* у регистар *rd*, док инструкција *mflo* премешта садржај регистара *Lo*.

## 2.2 Floating point регистри у MIPS-у

MIPS архитектура користи два формата FP (скр. *Floating Point*) препоручена од стране IEEE 754:

- *Једнострука прецизност* (енг. *Single precision*) - Користи се 32 бита за чување у меморији. MIPS компајлери користе једноструку прецизност за променљиве типа *float*
- *Двострука прецизност* (енг. *Double precision*) - Користи се 64 бита за чување у меморији. C компајлери коисте двоструку прецизност за C *double* типове података.

Начин да се две речи ширине 32 бита се смештају у меморију као једна реч ширине од 64 бита је начин смештања у меморији (виша половина битова прво, или нижа половина битова прво) и зависи од начина смештања у меморији.

Стандартд IEEE 754 је веома захтеван и поставио је два велика проблема. Први, омогућавање детекције неуобичајних резултата доводи проточну обраду (енг. *pipeline*) тешком. Постоји опција да се имплементира IEEE механизам сигнализирања изузетака, али је проблем да се детектују случајеви када хардвер не може да произведе исправан резултат и потребна му је помоћ.

Када се IEEE изузетак деси требало би обевестити и корисника, ово би требало бити синхроно; након заустављања корисник би желио да види све предходно извршене инструкције и све FP регистре који су у preinstruction стању и желе да се увере да ни једна следећа инструкција нема никакав ефекат.

У *MIPS* архитектури, хардверска заустављања су била овако одрађена. Ово заправо ограничава могућности проточне обраде FP операција, јер се не може извршити сљедећа инструкције све док хардвер може бити сигуран да операција FP неће произвести заустављање. Зарад избегавања додавања времена за извршавање, FP операције морају да одлуче да ли ће доћи до заустављања у првој фази.

## 2.3 Системски позиви

## 2.4 FPXX конвеција

# Глава 3

# Valgrind

*Valgrind* је платформа за прављење алата за динамичку бинарну анализу кода. Динамичка анализа обухвата анализу корисничког програма у извршавању, док бинаран анализа обухвата анализу на нивоу машинског кода, снимљеног или као објектни код (неповезан) или као извршни код (повезан). Постоје *Valgrind* алати који могу аутоматски да детектују проблеме са меморијом, процесима као и да изврше оптимизацију самог кода. *Valgrind* се може користити и као алат за прављење нових алата. *Valgrind* дистрибуција тренутно броји следеће алате: детектор меморијских грешака, детектор грешака нити, оптимизатор скривене меморије и скокова, генератор графа скривене меморије и предикције скока и оптимизатор коришћења динамичке меморије. *Valgrind* ради на следећим архитектурама: ***X86/Linux, AMD64/Linux, ARM/Linux, ARM64/Linux, PPC32/Linux, PPC64/Linux, PPC64LE/Linux, S390X/Linux, MIPS32/Linux, MIPS64/Linux, X86/Solaris, AMD64/Solaris, ARM/Android (2.3.x и новије), ARM64/Android, X86/Android (4.0 и новије), MIPS32/Android, X86/Darwin and AMD64/Darwin (Mac OS X 10.12)***.

У наредним поглављима биће детаљно описана структура *Valgrind* и његових алата, као и начин употребе са примерима проблема са којима се програмери свакодневно сусрећу.

## 3.1 О Valgrindu

Алат за динамичку анализу кода се креира као додатак, писан у C програмском језику, на језгро *Valgrind*.

*Језгро Valgrinda + алат који се додаје = Алат Valgrinda*

Језгро *Valgrind*-а омогућава извршавање клијетског програма, као и снимање извештаја који су настали приликом анализе самог програма.

Алати *Valgrind*-а користе методу бојења вредности. Они заправо сваки регистар и меморијску вредност „боје" (замењују) са вредношћу која говори нешто додатно о оригиналној вредности.

Сви *Valgrind* алати раде на истој основи, иако информације које се емитују варирају. Информације које се емитују могу се искористити за отклањање грешака, оптимизацију кода или било коју другу сврху за коју је алат дизајниран.

Сваки *Valgrind*-ов алат је статички повезана извршна датотека која садржи код алата и код језгра. Извршна датаоке valgrind представља програм омотач који је на основу –tool опције бира алат који треба покренути и покреће га помоћу системског позива ***execve***. Извршна датотека алата статички је линкована тако да се учитава почев од неке адресе која је обично доста изнад адресног простора који користе класичан кориснички програм (на ***x86/Linux*** и ***MIPS/Linux*** користи се адреса 0x38000000). У ретким случајевима, када та адреса није потреба, *Valgrind* се може прекомпајлирати да користи неку другу адресу. *Valgrind*-ово језгро прво иницијализује под-систем као што су менаџер адресног простора, и његов унутрашњи алокатр меморије и затим учитава клијентову извршну датотеку. Потом се иницијализују *Valgrind*-ови субсистеми као што су транслациона табела, апарат за обраду сигнала, распоређивач нити и учитавају се информације за дебаговање клијента, уколико постоје. Од тог тренутка *Valgrind* има потпуну контролу и почиње са превођењем и извршавањем клијентског програма. Може се рећи да *Valgrind* врши JIT (*Just In Time*) превођење машинског кода програма у машинкси код програма допуњен инструментацијом. Ниједан део кода клијента се не извршава у свом изворном облику. Алат се умеће у оригинални код на почетку, затим се нови код преводи, сваки основни блок појединачно, који се касније извршава. Процес превођења се састоји из рашчлањивања оригиналног машинског кода у IR (скр. *intermediate representation*) који се касније инструментализује са алтом и поново преводи у нови машински код.

Резултат свега овога се назива транслација, која се чува у меморији и која се извршава по потреби. Језгро троши највише времан на сам процес прављења, проналажења и извршавања транслације. Оригинални код се никада се извршава. Једини проблем који се овде може догодити је ако се врши транслација кода који се мења током извршавања програма.

IR има неке *RISC* одлике као што су *load/store*, свака операција ради само једну ствар, кад се линеаризује све операције раде само на привременим промељивама и литералима. Да би се подржале све целобројне, FP и SIMD операције над различитим величинама потребно је више од 200 примитвних аритметичко-логичких инструкција.

Постоје многе компликације које настају приликом смештања два програма у један процс (клијентски програм и програм алата). Многи ресурси се деле између ова два програма, као што су регистри или меморија. Такође, алат *Valgrind*-а не сме да се одрекне тоталне контроле над извршавањем клијетског програма приликом извршавања системских позива, сигнала и нити.

## Основни блок

*Valgrind* дели оригинални код у секвенце које се називају основни блокови. Основни блок је праволинијска секвенца машинског кода, на чији се почетак скаче, а која се завршава са скоком, позивом функције или повратком. Сваки код програма који се анализира поново се преводи на захтев, појединачно по основним блоковима, непосредно пре самог извршавања самог блока. Ако узмемо да су основни блокови клијетског кода *BB1, BB2, ...* онда преведене основне блокове обележавамо са *t(BB1), t(BB2), ...* Величина основног блока је ограничена на максимално 60 машинских инструкција. На процесорима *MIPS*, инструкције скока и гранања имају такозвано „одложено извршавање". То значи да се приликом извршавања тих инструкција извршава и инструкција која се налази непосредно иза инструкције гранања или скока. У случају да је последња шездесета инструкција основног блока инструкција гранања, *Valgrind* учитава и инструкцију која се налази непосредно иза ње, односно шездесет и прва инструкција. Тиме се омогућава конзистентно извршавање програма који се анализира, као и у случају да се програм извршава без посредства *Valgrind*-а. Уколико након извршених 60 инструкција *Valgrind* није наишао на инструкцију гранања, секвенца инструкција се дели на два или више основних блокова, који се извршавају један за другим.

## Системски позиви

Апликациони програми комуницирају са оперативним системом помоћу системских позива (енг. **system calls**), тј. преко операција (функција) које дефинише оперативни систем. Системски позиви се реализују помоћу система прекида: кориснички програм поставља параметре системског позива на одређене меморијске локације или регистре процесора, иницира прекид, оперативни систем преузима контролу, узима парамтре, извршава тражене радње, резултат ставља на одређене меморијске локацић еили у регистре и враћа контролу корисчком програму. Апликација која жели да користи неке од ресурса, као што су меморија, процесор или улазно/излазни уређаји, комуницира са језгром опративног система користећи системске позиве. Језгро оператвиног система дели виртуелну маморију на корисничку меморију и системску меморију. Системска меморија је одређена за само језгро оператвиног система, његова проширања, као и за урављачке програме. Кориснички прогстор је део меморије где се налазе све корсничке апликације приликом њиховог изврђавања. Корисничке апликације могу да приступе улазно/излазним уређајима, виртуелној меморији, датотекама ид ругим реусрсисам језгра оператвино система користећи само системске позиве. Системски позиви обезбеђују спрегу између програма који се извршава и оператвиног система. Генерално, реализују се на асемблерском језику, али новији виши програсмки језици, попут језика C и C++, такође омогућавају реализацију системског позива. Пграом кои се извршава може проседити параметре опративном систему на три начина:

- прослеђивање параметара у регистрима процесора;
- постављањем параметара у меморијској табели, при чему се адреса табеле прослеђује у регистру процесора;
- постављањем параметара на врх стека (енг. *push*), које оперативни систем „скида” (енг. *pop*).

Системски позиви се извршавају без посредства *Valgrind*-а, зато што језгро *Valgrind*-а не може да прати њихово извршавање у самом језгру оперативног система.

## Транслација

У наставку су описани кораци које *Valgrind* извршава приликом анализе програма. Постоји осам фаза транслације. Све фазе осим инструментације коју обавља алат *Valgrind*-а, обавља језгро *Valgrind*-а.

- **Дисасемблирање** - процес превођења машинског кода у еквивалетни асемблерски код. *Valgrind* врши превођење машинског кода у интерни скуп инструкција која се називају међукод инструкције. Међукод представља редуковани скуп инструкција (скр. енг. *RISC*). Ова фаза је зависна од архитетктуре на којој се извршава.

- **Оптимизација 1** - Прва фаза оптимизације линеаризује *IR* репрезентацију. Примењују се неке стандардне оптимизације програмских преводилаца као што су уклањање редудантног кода, елиминација подизраза, једноставно одмотавање петљи и сл.

- **Инструментација** - Блок кода у *IR* репрезентацији се прослеђује алату, који може произвољно да га трансформише. Приликом инструментације алат у задати блок додаје додатне *IR* операције, кјима проверава исправност рада програма.

- **Оптимизација 2** - Друга фаза оптимизације је једноставније од прве, укључује множење констати и уклањање мртвог кода.

- **Градња стабла** - Линеаризована *IR* репрезентација се конвертује натраг у стабло ради лакшег избора инструкција.

- **Одабир инструкција** - Стабло *IR* репрезентације се конвертује у листу инструкција које користе виртуалне регистре. Ова фаза се такође разликује у зависности од архитеткуре на којој се извршава.

- **Алокација регистара** - Виртуални регистри се замењују стварним. По потреби се уводе пребацивања у меморију. Независна је за платформу, користи позив функција које налазе из који се регистара врши читање и у које се врши упис.

- **Асемблирање** - Изабране инструкције се енкодују на одговарајући начин и смештају у блок мемроји. Ова фаза се такође разликује у зависноти од архитектуре на који се изршава. [3]

## 3.2 Memcheck

## 3.3 Callgrind

## 3.4 Cachgrind

## 3.5 Massif

## 3.6 Helgrind

## 3.7 DRD

# Глава 4

# FPXX

# Глава 5

# Закључак

Фијуче ветар у шибљу, леди пасаже и куће иза њих и гунђа у оџацима. Ниџо, чежњиво гледаш фотељу, а Ђура и Мика хоће позицију себи. Људи, јазавац Џеф трчи по шуми глођући неко сухо жбуње. Љубави, Олга, хајде пођи у Фуџи и чут ћеш њежну музику срца. Боја ваше хаљине, госпођице Џафић, тражи да за њу кулучим. Хаџи Ђера је заћутао и бацио чежњив поглед на шољу с кафом. Џабе се зец по Хомољу шуња, чувар Јожеф лако ће и ту да га нађе. Оџачар Филип шаље осмехе туђој жени, а његова кућа без деце. Бутић Ђуро из Фоче има пун џак идеја о слагању ваших жељица. Џајић одскочи у аут и избеже ђон халфа Пецеља и његов шамар. Пламте оџаци фабрика а чађаве гује се из њих дижу и шаљу ноћ. Ајшо, лепото и чежњо, за љубав срца мога, дођи у Хаџиће на кафу. Хучи шума, а иза жутог џбуна и пања ђак у цвећу деље сеји фрулу. Гоци и Јаћиму из Бање Ковиљаче, флаша џина и жеђ падаху у исту уру. Џаба што Феђа чупа за косу Миљу, она јури Живу, али њега хоће и Даца. Док је Фехим у џипу журно љубио Загу Чађевић, Циле се ушуњао у ауто. Фијуче кошава над оџацима а Иља у гуњу журећи уђе у суху и топлу избу. Боже, џентлмени осећају физичко гађење од прљавих шољица! Дочепаће њега јака шефица, вођена љутом срџбом злих жена. Пази, геџо, брже однеси шефу тај ђавољи чек: њим плаћа цех. Фине џукце озлеђује бич: одгој их пажњом, стрпљивошћу. Замишљао би кафеџију влажних прстића, црњег од чађи. Ђаче, уштеду плаћај жаљењем због џиновских цифара. Џикљаће жалфија између тог бусења и пешчаних двораца. Зашто гђа Хаџић лечи живце: њена љубав пред фијаском? Јеж хоће пецкањем да вређа љубичастог џина из флаше. Џеј, љубичаст зец, лаже: гађаће одмах поквашен фењер. Плашљив зец хоће јефтину дињу: грожђе искамчи џабе. Џак је пун жица: чућеш тад свађу због ломљења

харфе. Чуј, џукац Флоп без даха с гађењем жваће стршљена. Ох, задњи шраф на џипу слаб: муж гђе Цвијић љут кочи. Шеф џабе звиждуће: млађи хрт јаче кљуца њеног пса. Одбациће кавгаџија плаштом чађ у жељезни фењер. Дебљи кројач: згужвах смеђ филц у тањушни џепић. Џо, згужваћеш тихо смеђ филц најдебље крпењаче. Штеф, бацих сломљен дечји зврк у џеп гђе Жуњић. Дебљој згужвах смеђ филц — њен шкрт џепчић.

Фијуче ветар у шибљу, леди пасаже и куће иза њих и гунђа у оџацима. Ниџо, чежњиво гледаш фотељу, а Ђура и Мика хоће позицију себи. Људи, јазавац Џеф трчи по шуми глођући неко сухо жбуње. Љубави, Олга, хајде пођи у Фуџи и чут ћеш њежну музику срца. Боја ваше хаљине, госпођице Џафић, тражи да за њу кулучим. Хаџи Ђера је заћутао и бацио чежњив поглед на шољу с кафом. Џабе се зец по Хомољу шуња, чувар Јожеф лако ће и ту да га нађе. Оџачар Филип шаље осмехе туђој жени, а његова кућа без деце. Бутић Ђуро из Фоче има пун џак идеја о слагању ваших жељица. Џајић одскочи у аут и избеже ђон халфа Пецеља и његов шамар. Пламте оџаци фабрика а чађаве гује се из њих дижу и шаљу ноћ. Ајшо, лепото и чежњо, за љубав срца мога, дођи у Хаџиће на кафу. Хучи шума, а иза жутог џбуна и пања ђак у цвећу деље сеји фрулу. Гоци и Јаћиму из Бање Ковиљаче, флаша џина и жеђ падаху у исту уру. Џаба што Феђа чупа за косу Миљу, она јури Живу, али њега хоће и Даца. Док је Фехим у џипу журно љубио Загу Чађевић, Циле се ушуњао у ауто. Фијуче кошава над оџацима а Иља у гуњу журећи уђе у суху и топлу избу. Боже, џентлмени осећају физичко гађење од прљавих шољица! Дочепаће њега јака шефица, вођена љутом срџбом злих жена. Пази, геџо, брже однеси шефу тај ђавољи чек: њим плаћа цех. Фине џукце озлеђује бич: одгој их пажњом, стрпљивошћу. Замишљао би кафеџију влажних прстића, црњег од чађи. Ђаче, уштеду плаћај жаљењем због џиновских цифара. Џикљаће жалфија између тог бусења и пешчаних двораца. Зашто гђа Хаџић лечи живце: њена љубав пред фијаском? Јеж хоће пецкањем да вређа љубичастог џина из флаше. Џеј, љубичаст зец, лаже: гађаће одмах поквашен фењер. Плашљив зец хоће јефтину динју: грожђе искамчи џабе. Џак је пун жица: чућеш тад свађу због ломљења харфе. Чуј, џукац Флоп без даха с гађењем жваће стршљена. Ох, задњи шраф на џипу слаб: муж гђе Цвијић љут кочи. Шеф џабе звиждуће: млађи хрт јаче кљуца њеног пса. Одбациће кавгаџија плаштом чађ у жељезни фењер. Дебљи кројач: згужвах смеђ филц у тањушни џепић. Џо, згужваћеш тихо смеђ филц најдебље крпењаче. Штеф, бацих сломљен дечји зврк у џеп гђе

Жуњић. Дебљој згужвах смеђ филц — њен шкрт џепчић.

# Библиографија

[1] Yuri Gurevich and Saharon Shelah. Expected computation time for Hamiltonian path problem. *SIAM Journal on Computing*, 16:486–502, 1987.

[2] Petar Petrović and Mika Mikić. Naučni rad. In Miloje Milojević, editor, *Konferencija iz matematike i računarstva*, 2015.

[3] Dominic Sweetman. *See MIPS Run.* Wiley, 2007.

# Биографија аутора

**Вук Стефановић Караџић** (*Тршић, 26. октобар/6. новембар 1787. — Беч, 7. фебруар 1864.*) био је српски филолог, реформатор српског језика, сакупљач народних умотворина и писац првог речника српског језика. Вук је најзначајнија личност српске књижевности прве половине XIX века. Стекао је и неколико почасних доктората. Учествовао је у Првом српском устанку као писар и чиновник у Неготинској крајини, а након слома устанка преселио се у Беч, 1813. године. Ту је упознао Јернеја Копитара, цензора словенских књига, на чији је подстицај кренуо у прикупљање српских народних песама, реформу ћирилице и борбу за увођење народног језика у српску књижевност. Вуковим реформама у српски језик је уведен фонетски правопис, а српски језик је потиснуо славеносрпски језик који је у то време био језик образованих људи. Тако се као најважније године Вукове реформе истичу 1818., 1836., 1839., 1847. и 1852.