УНИВЕРЗИТЕТ У БЕОГРАДУ

ЕЛЕКТРОТЕХНИЧКИ ФАКУЛТЕТ

Урош Р. Раденковић

Спекулативно извршавање инструкција са непрецизно предвиђеним операндима

документ

докторска дисертација

Београд, 2024.

UNIVERSITY OF BELGRADE

SCHOOL OF ELECTRICAL ENGINEERING

Uroš R. Radenković

Speculative execution of instructions with imprecisely predicted operands

Doctoral Dissertation

Belgrade, 2024.

Ментор

др Захарије Радивојевић, ванредни професор
Универзитет у Београду, Електротехнички факултет

Чланови комисије

др Милош Цветановић, ванредни професор
Универзитет у Београду, Електротехнички факултет

др Бошко Николић, редовни професор
Универзитет у Београду, Електротехнички факултет

др Владимир Миловановић, ванредни професор
Универзитет у Крагујевцу, Факултет инжењерских наука

др Драгомир Ел Мезени, доцент
Универзитет у Београду, Електротехнички факултет

др Живојин Шуштран, доцент
Универзитет у Београду, Електротехнички факултет

Датум одбране

_____

Захвалница

Упустивши се у рад на Електротехничком факултету и кренувши на докторске студије, често гледајући друге докторске дисертације и захвалнице у њима, размишљао сам о тренутку када ћу ја писати захвалницу у својој дисертацији. У тренуцима када пишем ову захвалницу осећам се неизмерно срећно, поносно и узбуђено што сам дошао до овог тренутка и желим да се на овом месту захвалим људима који су томе допринели.

Ову докторску дисертацију посвећујем својој породици, оцу, мајци и тетка Доки. Они су ми немерљива и увек присутна подршка у животу. Хвала им на свему.

Велико хвала мом ментору професору Захарију Радивојевићу који ме је зналачки подржавао да истрајем и дођем до краја докторских студија. Хвала му за много сати дискусија из којих су произашле и идеје за ову дисертацију. Још једна ствар за мене изузетно вредна јесте да сам од њега научио много о послу којим се бавимо што ме је обликовало као предавача и човека.

Захваљујем се мојим друговима са којима сам заједно кренуо путем сарадника, а потом и асистента, Мићку и Јоцку. Намерно кажем друговима јер се ми тако ословљавамо иако смо и колеге са исте генерације. Јоцка нисам ни познавао током основних студија јер где човек да га упозна. Мићка сам нешто мало познавао, пошто много прича волео је одмах након испита да дискутује шта је добио од резултата па смо се тада сретали. Данас, када од њих чујем „друже“ знам да то не долази од колеге или друга, већ од правог и искреног пријатеља.

Велику захвалност дугујем нешто старијим колегама Дражи и Жики који су допринели својим саветима да лакше пролазим кроз пут сарадника и асистента. Нарочито им се захваљујем на пруженој помоћи на почетку тог пута као и на увек доброј атмосфери у канцеларији 909.

Захваљујем се професорима Бошку Николићу и Милошу Цветановићу који су увек доступни за разговор и са којима сам сарађивао на неколико пројеката. Они су ме често мотивисали да што пре кренем да радим на докторској дисертацији и на том путу подржавали.

Захваљујем се и свим осталим члановима Катедре за рачунарску технику и информатику Електротехничког факултета. Заиста ми је задовољство радити у таквом колективу.

Велико хвала мојим пријатељима који су употпунили мој живот у различитим периодима, а овде су наведени по редоследу упознавања: Дача, Ђоле, Маркони, Вариоц, Дарко, Мишке, кум Лука, Пара и Рајнпрехт.

На крају бих се захвалио својој девојци Сањи која се појавила у мом животу негде на половини мојих докторских студија. Хвала јој што је заједно са мном прошла кроз све изазове које докторске студије носе.

Наслов докторске дисертације

Спекулативно извршавање инструкција са непрецизно предвиђеним операндима

Сажетак

Спекулативно извршавање је техника на нивоу хардвера коју процесори користе како би побољшали време извршавања. Ова техника представља извршавање неког посла унапред не знајући да ли је заиста потребно извршити га. Механизам предвиђања вредности је техника спекулативног извршавања инструкција која омогућава да се реши проблем зависности по подацима, где нека (зависна) инструкција не може да настави извршавање зато што мора да сачека резултат друге инструкције која још увек није завршена. У таквој ситуацији, вредност која недостаје инструкцији може се предвидети и инструкција може наставити да се извршава спекулативно.

Предмет истраживања ове дисертације је спекулативно извршавање одређених типова инструкција са непрецизно предвиђеним операндима. За разлику од коришћења предиктора вредности за предвиђање резултата инструкције, у склопу дисертације разматра се предвиђање операнада. Након тога се израчунава резултат инструкције на основу предвиђеног операнда и познатог операнда чија је вредност одмах доступна процесору (вредност регистра или непосредна вредност). Операнди који се предвиђају јесу они који нису одмах доступни процесору, а то су операнди који потичу из меморије.

Циљ истраживања јесте приказивање феномена да је могуће доћи до тачног резултата одређених инструкција и у ситуацијама са непрецизно предвиђеним операндима чиме се избегава трошење процесорског времена на опоравак од промашаја. У сврху тога развијена су два аналитичка модела спекулативног извршавања инструкција са предвиђањем вредности операнада. Први модел представља извршавање само са тачно предвиђеним операндима и у случају непрецизно предвиђеног операнда, опоравак од промашаја је обавезан. Други модел дозвољава да се извршавање настави иако је операнд непрецизно предвиђен уколико је резултат инструкције исправан, чиме се избегава опоравак од промашаја.

За потребе евалуације предложених модела развијен је симулатор *VPSim* који извршава инструкције са предвиђањем вредности операнада. Такође, надограђен је симуалтор *Gem5* за генерисања трагова извршавања тестова *SPEC* и *EEMBC* који су улаз симулатора *VPSim*. На основу резултата симулација описани су услови под којима други модел постиже боље време извршавања од првог. Такође, резултати симулација су показали да је прецизност тачног резултата инструкције на основу предвиђеног операнда већа од прецизности тачног предвиђеног операнда између 0,8% до 44% у зависности од предиктора.

Кључне речи

предвиђање вредности, спекулативно извршавање, архитектура рачунара, анализа перформанси

Научна област

Електротехника и рачунарство

Ужа научна област

Рачунарска техника и информатика

Doctoral dissertation title

Speculative execution of instructions with imprecisely predicted operands

Abstract

Speculative execution is a hardware-level technique that processors use to improve execution time. This technique represents doing some work in advance without knowing if it really needs to be done. The value prediction mechanism is a speculative instructions execution technique that allows for solving data dependency problems, where some (dependent) instruction cannot continue execution because it has to wait for the result of another instruction that has not yet been completed. In such a situation, the value missing from the instruction can be predicted, and the instruction can continue to execute speculatively.

This dissertation's research subject is the speculative execution of certain types of instructions with imprecisely predicted operands. In contrast to using value predictors to predict the result of an instruction, the dissertation considers predicting operands. After that, the result of the instruction is calculated based on the predicted operand and a known operand whose value is immediately available to the processor (register value or immediate value). Predicted operands are those that are not immediately available to the processor, i.e. operands that originate from memory.

The goal of the research is to show the phenomenon that it is possible to get the correct result of certain types of instructions even in situations with imprecisely predicted operands, which avoids spending processor time on misprediction recovery. For this purpose, two analytical models of speculative instruction execution with operand value prediction were developed. The first model represents execution only with exactly predicted operands and in the case of an imprecisely predicted operand, recovery from a misprediction is mandatory. The second model allows execution to continue even if the operand is imprecisely predicted if the result of the instruction is true, thus avoiding misprediction recovery.

For the purposes of evaluating the proposed models, a VPSim simulator was developed that executes instructions with the prediction of operand values. Also, the Gem5 simulator has been upgraded to generate execution traces of SPEC and EEMBC tests that represent inputs of the VPSim simulator. Based on the results of the simulations, the conditions under which the second model achieves a better execution time than the first are described. Also, the simulation results showed that the precision of the correct result of the instruction based on the predicted operand is higher than the precision of the correctly predicted operand, between 0.8% and 44%, depending on the predictor.

Key words

value prediction, speculative execution, computer architecture, performance evaluation

Scientific field

Electrical Engineering and Computing

Scientific subfield

Computer Engineering and Informatics

САДРЖАЈ

СПИСАК СЛИКА

СПИСАК ТАБЕЛА

1. Увод

Како би се сагледали тренутни изазови у свету рачунарства и развоју рачунара, односно самих процесора као основог дела сваког рачунарског система, треба поменути неколико битних момената из прошлости развоја рачунарства. На тај начин се може идентификовати значај области архитектуре рачунара (енгл. *computer architecture*) као дела рачунарства. Такође, на основу тога се може разумети шта све област архитектуре рачунара обухвата као и ком делу архитектуре рачунара припада ова докторска дисертација.

Битан искорак у развоју рачунарства је настанак првих механичких калкулатора, чију основу је чинио механизам зупчаника. Први реализован такав калкулатор који је имао могућност уноса два броја и потом могућност да одради неку операцију над њима био је Паскалина [1], а развио га је Блез Паскал у седамнаестом веку. Касније током деветнаестог века Чарлс Бебич је дизајнирао један механички рачунар под називом Аналитичка машина (енгл. *Analytical engine*) који је касније описала Ада Лавлејс. Управо се тај опис сматра првом дефинисаном архитектуром рачунара [2].

Половином двадесетог века, реализовани су први електромеханичким рачунари са вакумским цевима, *ENIAC* (енгл. *Electronic Numerical Integrator And Computer*) [3] и његов наследник *EDVAC* (енгл. *Electronic Discrete Variable Automatic Computer*) [4] који је реализован према Фон Најмановој архитектури [5]. У слично време реализован је и рачунар *Harvard Mark I* [6] према Харвард архитектури. Управо поменуте две архитектуре, Фон Најманова и Харвард, представљају архитектуре које су у основи већине данашњих рачунарских система. Према њима рачунарски систем се састоји од централне процесорске јединице (процесор), оперативне меморије и улазно-излазног подсистема. Ове две архитектуре дефинишу процесор као целину састављену од аритметичко-логичке јединице, контролне јединице и регистара, баш како се и данашњи процесори могу описати.

Велики развитак рачунарства забележен је након Другог светског рата захваљујући настанку електронских рачунара у чијој основи се налазе транзистори. У почетку су транзистори били уграђивани појединачно на заједничкој плочи, а потом повезивани жицама. Изузетан напредак у развоју рачунара у погледу брзине рада, значајно мање величине као и значајно мање потрошње енергије, представља појава интегрисаних кола код којих је више транзистора смештено на истом парчету полупроводника, углавном силицијума. Такође, програми за њих су били писани на асемблеру, али и на вишим програмским језицима који су настали у то време, међу њима су Фортран, Кобол и Алгол.

Од тренутка настанка интегрисаних кола, настаје експанзија рачунарства. Гордон Мур, један од оснивача компаније Интел, инспирисан развијањем интегрисаних чипова 1965. године представља прогнозу да ће се сваке две наредне године број транзистора удвостручити на чипу [7]. Ова прогноза се показала исправном до 2010. године од када број транзистора на чипу спорије расте [8]. Такође, данашњи процесори су достигли одређене фреквенције радног такта и последњих година постоји раст фреквенције од само неколико процената годишње [8] [9]. На пример, процесори компаније Интел су 2003. године достигли фреквенцију од преко 3 GHz, да би 2017. године пребацили границу од 4 GHz, док најновији

и најмоћнији процесори овог произвођача имају максималну фреквенцију 6 GHz [10]. Ове фреквенције су диктиране потребом за ограниченом потрошњом енергије процесора и вођењем рачуна о загревању процесора.

Заједно са напретком технологије израде чипова (процесора), развијала се и област архитектуре рачунара. Нарочит значај ове области огледа се у томе што доноси побољшања перформанси процесора са ограничењима које диктирају технологија израде (спорији раст броја транзистора), максимална фреквенција радног такта и специфична намене рачунара [11]. Област архитектуре рачунара према ауторима књиге [8] данас обухвата неколико ствари за разлику од традиционалног схватања да само опис инструкцијског сета (енгл. *instruction set architecture - ISA*) представља архитектуру рачунара. У наставку су наведене ствари које се данас сматрају деловима области архитектуре рачунара:

- Опис инструкцијског сета – представља скуп доступних асемблерских инструкција и начина адресирања програмеру, односно преводиоцу (енгл. *compliler*) уколико програмер пише програмски код на вишем програмском језику, а не на асемблеру. Инструкцијски сет даје опис доступних инструкција описујући њихов резултат као и значење и број операнада. Он на неки начин представља везу софтвера са хардвером, односно процесором, јер даје опис инструкција које процесор уме извршити, а које су градивни елементи програма, тј. софтвера;
- Организација процесора, у литерати се још користи термин микроархитектура (енгл. *microarchitecture*) – представља интерну организацију процесора којом се постиже могућност извршавања инструкција које чине инструкцијски сет. Дизајн интерних регистара, аритметичко логичке јединице, проточне обраде (енгл. *pipeline*), кеш меморије (енгл. *cache memory*) и спекулативног извршавања инструкција (енгл. *speculative execution*) су неке од ствари који припадају организацији процесора. Заправо, организација процесора описује начин на који се инструкције извршавају на процесору;
- Имплементација (енгл. *implementation*) – израда чипа на основу микроархитектуре, подразумева одабир технологије у којој ће се израдити чип као и верификацију чипа. Под верификацијом се сматра провера дизајна процесора према микроархитектури, тј. да ли се пројектовани процесор понаша на очекивани начин у складу са микроархитектуром. Поред тога, овај део архитектуре рачунара бави се још особинама процесора као што су потрошња енергије и загревање као и њиховом оптимизацијом према специфичним потребама за коришћењем процесора.

У оквиру ове докторске дисертације разматра се део архитектуре рачунара који се односи на микроархитектуру, односно на организацију процесора. Као што је поменуто, овај део архитектуре рачунара се односи на интерну организацију процесора, а дисертација се бави спекулативним извршавањем. Под механизмом спекулативног извршавања подразумева се могућност процесора да неки посао одради унапред. Касније током извршавања процесор треба да провери да ли је заиста било потребно одрадити посао унапред и у складу са провером да преузме одређене акције. Неки од послова које процесор може одрадити унапред јесу спекулативно извршавање инструкција (енгл. *speculative instruction execution*) [12][13] и дохватање података унапред из меморије (енгл. *data prefetching*) [14]. Спекулативно извршавање инструкција подразумева да процесор одради неке инструкције унапред не знајући у тренутку када их извршава да ли је заиста потребно извршити их.

Унапред дохватање података подразумева да процесор може дохватити неке податке из меморије не знајући да ли ће му они заиста бити потребни касније током извршавања.

Ова докторска дисертација се бави спекулативним извршавањем инструкција. Два механизма на нивоу организације процесора која подразумевају спекулативно извршавање инструкција су механизам предвиђања скокова (енгл. *branch prediction*) [12] и механизам предвиђања вредности (енгл. *value prediction*) [13]. На основу механизма предвиђања скокова настао је механизам предвиђања вредности па се стога треба укратко осврнути на механизам предвиђања скокова.

Предвиђање скокова се користи код процесора са проточном обрадом у ситуацији када на извршавање дође инструкција условног скока. Процесори са проточном обрадом [15] извршавање инструкција спроводе у више фаза, односно степени проточне обраде. Сваки степен проточне обраде има одређену намену, па тако постоје степени у којима се дохватају инструкције, врши декодовање инструкције, одрађује операција над операндима и на крају се врши упис резултата у регистре или меморију [16]. Код таквих процесора долази до контролних хазарда (енгл. *control hazards*), а то су ситуације у којима процесор након инструкције условног скока не зна која је наредна инструкција коју треба учитати. То ће процесору бити познато тек када инструкција скока дође у одређен степен проточне обраде у коме се врши утврђивање да ли је услов скока задовољен. Како би се избегло заустављање учитавања нових инструкција док се не утврди исход скока, осмишљен је механизам предвиђања скокова. Овај механизам приликом извршавања условног скока врши предвиђање која је наредна инструкција коју треба процесор да извршава. Постоје два могућа исхода, тј. две инструкције кандидати за извршавање након инструкције условног скока. Први исход јесте да је инструкција коју треба извршити инструкција која непосредно следи у асемблерском коду након инструкције скока, што одговара ситуацији да се скок није десио (енгл. *branch not taken*). Други исход јесте да је инструкција која се налази на одредишној адреси скока инструкција коју треба извршити, што одговара ситуацији да се скок десио (енгл. *branch taken*).

Инструкције које на основу предвиђања буду учитане и започну извршавање сматрају се спекулативним инструкцијама, а њихово извршавање се назива спекулативно извршавање. У тренутку када се сазна коначан исход извршавања условног скока (да ли се скок десио или није), тада се морају предузети акције над спекулативним инструкцијама. Уколико је предвиђање било тачно, спекулативне инструкције се потврђују (енгл. *commit*). Ако предвиђање није било добро, тада се спекулативне инструкције поништавају, врши се рестаурација интерног стања процесора на стање као пре почетка спекулативног извршавања и започиње извршавање нових инструкција према стварном исходу скока.

Примећено је да програми испољавају понашање где приступају некој меморијској локацији, а потом у скоријој будућности приступају истој тој локацији. Такво понашање се назива временском локалношћу (енгл. *temporal locality*). Такође, програми током извршавања уколико су приступили некој меморијској локацији, постоји велика шанса да ће у скоријој будућности приступати и локацијама које се налазе у њеној близини. Такво понашање се назива просторном локалношћу (енгл. *spatial locality*). Поред ове две локалности, уочена је још једна локалност коју испољавају програми приликом извршавања. Та локалност се огледа у томе да постоји велика вероватноћа да ће вредност податка се неке локације бити иста као нека вредност која се у скоријој прошлости налазила на тој локацији. Такво понашање се назива вредносна локалност (енгл. *value locality*) и први пут је објашњена у раду [17]. Због постојања временске и просторне локалности осмишљена је кеш меморија на

процесору, а вредносна локалност је дала повода да се развија механизам предвиђања вредности.

Као што је поменуто на основу идеје о предвиђањима скокова, настали су механизми предвиђања вредности. Њихова идеја јесте да реше проблеме праве зависности по подацима између инструкција (енгл. *true data dependecies*), који се називају још и хазардима података (енгл. *data hazards*). Ови проблеми настају у ситуацији када је резултат једне инструкције потребан некој блиској наредној инструкцији која тај резултат користи као свој операнд. Могуће је да тада инструкција која користи резултат мора да сачека да се заврши инструкција која производи резултат што доводи до заустављања извршавања и учитавања нових инструкција у проточну обраду. Када је некој инструкцији потребна вредност операнда, а његова вредност још увек није доступна, тада је могуће применити предвиђање вредности. Предиктор вредности тада може предвидети вредност која ће се користити уместо праве вредности операнда. Инструкција у том случају може наставити своје извршавање са предвиђеним операндом и на тај начин постаје спекулативна инструкција. Такође, све инструкције које зависе од њеног резултата су спекулативне инструкције.

Слично као и код механизама предвиђања скокова и код предвиђања вредности мора се утврдити исправност предвиђања. Уколико предвиђање није добро, спекулативне инструкције се поништавају и врши се опоравак од лошег предвиђања. Док у случају исправно предвиђене вредности, спекулативно извршавање се сматра исправним и наставља се извршавање.

Механизам предвиђања скокова је у употреби у комерцијалним процесорима већ неколико деценија. Ови механизми су јако добро развијени и постижу прецизност (енгл. *accuracy*) која прелази 99% [12]. За разлику од њих, механизми предвиђања вредности се и даље истражују и нису у употреби у комерцијалним процесорима. Разлог за то је што је њихова прецизност значајно мања и то онда води ка честим опоравцима процесора услед лошег предвиђања. На тај начин много времена процесор губи на честе процесе опоравка. Прецизност предвиђања вредности је мања из разлога што предиктори вредности имају тежак задатак јер треба да погоде вредност која може бити било која вредност, док рецимо предиктори скокова као што је поменуто имају могућа само два исхода.

Ова докторска дисертација се бави спекулативним извршавањем са предвиђањем вредности са посебним акцентом на ситуације када вредност није тачно предвиђена, тј. када је вредност непрецизно предвиђена. Идеја је да се види у којим ситуацијама је могуће успешно наставити извршавање иако вредност није тачно предвиђена на начин да непрецизно предвиђена вредност не учите на коректност програма који се извршава на процесору. Таквим начином извршавања инструкција где би се извршавање наставило и са непрецизно предвиђеном вредношћу, процесор би мање пута морао да врши опоравак услед лоше предвиђене вредности.

Увођење меморијске хијерархије (енгл. *memory hierarchy*) [18] у организацију процесора, пре свега механизма кеш меморије, има за циљ да се ублажи разлика у брзинама рада процесора и меморије, тј. да се сакрије кашњење које уноси меморија (енгл. *memory wall*) [19][20]. Уколико процесор тражи податак који се налази у кеш меморији, тада се избегава приступ оперативној меморији што доводи до бољег времена извршавања. Када се деси промашај у кеш меморији, процесор мора приступити оперативној меморији заустављајући извршавање инструкције док се приступ не заврши.

Ова докторска дисертација представља извршавање инструкција одређеног типа инструкција са превиђањем вредности када им један од операнада није доступан. Операнд

може бити недоступан из разлога што се чека да његову вредност произведе нека друга инструкција или уколико се операнд налази у меморији па је потребно време да се дохвати из меморије. У ситуацијама у којима операнд није доступан одмах услед тога што се налази у меморији може се применити механизам предвиђања вредности. На тај начин инструкција не мора да чека да се заврши приступ меморији, већ може наставити спекулативно извршавање са предвиђеним операндом.

Како би се боље разумела основна идеја ове докторске дисертације, потребно је објаснити шта је резултат извршавања неке инструкције. Резултат инструкције се може посматрати као целина која се састоји из два дела, где у зависности од типа инструкције неки део не мора да постоји. Први део резултата извршавања неке инструкције јесте резултат операције која је спроведена над операндима. Тај резултат се смешта или у неки од регистара процесора или на неко место у меморији и он представља један део резултата – дестинациони операнд. Други део резултата представљају индикатори (енгл. *flags*) у програмској статусној речи – индикатори. У зависности од типа инструкције, неки од индикатора се постављају у складу са вредношћу резултата операције над операндима. Тако да резултат инструкције може бити: само вредност дестинационог операнда, само индикатори или и вредност дестинационог операнда и индикатори.

У оквиру ове докторске дисертације разматра се спекулативно извршавање инструкција са предвиђањем вредности код којих је један операнд одмах доступан нпр. налази се у регистру или је непосредна вредност (уграђена вредност операнда у саму инструкцију). Други операнд потиче из меморије и може зауставити извршавање док се не заврши приступ меморији у случају да није присутан у кеш меморији (енгл. *cache miss*). За разлику од коришћења предиктора вредности за предвиђање резултата инструкције, у склопу дисертације се разматра предвиђање операнада инструкције. Након тога се израчунава резултат инструкције на основу предвиђеног операнда и познатог (одмах доступног) операнда. Како је један операнд доступан, инструкције које су погодне да се врши предвиђање вредности другог операнда су оне код којих резултат не зависи од свих битова операнда чија се вредност предвиђа. Код таквих инструкција је могуће добити тачан резултат иако нису сви битови предвиђеног операнда тачно предвиђени, тј. може се добити тачан резултат и са непрецизно предвиђеним операндом.

На пример, посматра се инструкција логичког *и*, где је први операнд познат, а други потиче из меморије. Други операнд је операнд чија вредност се може предвиђати како инструкција не би чекала да се заврши приступ меморији. Како би се добио исправан резултат инструкције, довољно је само тачно предвидети битове који се налазе на истим позицијама као и битови првог операнда са вредношћу један. На осталим позицијама, тј. на позицијама које одговарају битовима првог операнда са вредношћу нула, свеједно је које ће вредности бити предвиђене за битове на тим позицијама другог операнда. Свакако у складу са операцијом логичког *и* резултујући бит ће имати вредност нула. Слична ствар се може уочити и код инструкције логичког *или*. Код ове инструкције битови другог операнда на позицији битова са вредношћу један првог операнда, могу имати било коју предвиђену вредност и резултат инструкције ће бити тачан. За ове типове инструкција дакле постоји скуп вредности које је могуће предвидети као вредност другог операнда, а да резултат (вредност дестинационог операнда и индикатори) остане исправан.

Други пример инструкција код којих је могуће добити тачан резултат извршавања и са непрецизно предвиђеним операндом јесу инструкције које врше поређење и тестирање операнада, а чији је резултат само постављање индикатора. Може се посматрати инструкција

која пореди два операнда, од којих је један познат (налази се у регистру или је непосредна вредност), а за други операнд се врши предвиђање вредности. Ако је рецимо вредност првог операнда нула, а тачна вредност другог операнда је нека вредност већа од нуле, у оваквом сценарију резултат инструкције јесте тачан са било којом предвиђеном вредношћу која је већа од нуле. Дакле, није потребно предвидети тачну вредност операнда. Довољно је да предвиђена вредност буде само у оном делу бројевне праве где се налази и тачна вредност операнда посматрајући да је тачка која разграничава два дела бројевне праве вредност првог операнда. Инструкција која врши тестирање два операнда заправо врши операцију логичког *и* само што је њен резултат само постављање индикатора, па за њу важи резоновање које је поменуто за инструкцију логичког *и*.

Циљ истраживања у оквиру ове докторске дисертације јесте приказивање феномена да је могуће доћи до тачног резултата инструкције и у ситуацијама са непрецизно предвиђеним операндима чиме се избегава трошење процесорског времена на опоравак од промашаја. У сврху тога биће развијена два аналитичка модела спекулативног извршавања инструкција са предвиђањем вредности операнада. Први модел представља извршавање само са тачно предвиђеним операндима, што сигурно води до тачног резултата инструкције. У случају непрецизно предвиђеног операнда, опоравак од промашаја је обавезан. Други модел дозвољава да се извршавање настави иако је операнд непрецизно предвиђен уколико је резултат инструкције исправан. Опоравак од промашаја код другог модела се спроводи само у ситуацији када резултат инструкције није исправан. Међутим, овај модел укључује додатно извршавање инструкције за коју се вршило предвиђање у тренутку када права вредност операнда постане доступна. На тај начин се врши провера да ли су резултати добијени са предвиђеним и са правим операндом идентични. Циљ истраживања јесте да се на основу стандардизованих тестова перформанси процесора дефинишу услови под којима извршавање према моделу са непрецизно предвиђеним операндима постиже боље време извршавања од извршавања према моделу са само тачно предвиђеним операндима.

Значај овог истраживања огледа се у томе да отвара нове видике на пољу архитектуре и организације рачунара, пре свега у области предвиђања вредности, где се пажња искључиво обраћа на исправан резултат инструкције иако је можда операнд непрецизно предвиђен. Као што је поменуто, у процесорима и даље није имплементирано спекулативно извршавање коришћењем механизама предвиђања вредности највише због потребе за честим опоравцима услед лошег предвиђања. Модел извршавања са непрецизно предвиђеним операндима може послужити за даља разматрања могуће имплементације таквог спекулативног извршавања у постојећим архитектурама. Такође, значај и актуелност ове теме огледа се и у тренутним истраживањима о чему сведоче прикупљене референце као и у постојању светског такмичења у предвиђању вредности [21].

Осим уводног поглавља, дисертација има још осам поглавља. У уводном поглављу је представљен кратак осврт на најбитније моменте у развоју рачунарства, тј. у развоју самих процесора како би се сагледала улога области архитектуре рачунара. Такође, представљене су основне идеје, значај и циљ ове докторске дисертације.

У другом поглављу је дат опис механизама предвиђања вредности, основе рада предиктора вредности и начини на који се предиктори вредности могу користити. Такође, приказана је основна подела предиктора вредности према начину на који врше предвиђање. У другом поглављу су представљени и тренутни изазови и проблеми код механизама предвиђања вредности.

Треће поглавље ове докторске дисертације садржи опис постојећих предиктора вредности. За сваки предиктор приказана је принципска шема функционисања и објашњене су основне идеје на који начин врше предвиђање вредности.

У четвртом поглављу је представљена главна идеја као и мотивација за израду ове докторске дисертације. Такође, представљена су решења којима се ублажавају уочени проблеми код механизама предвиђања вредности. У оквиру четвртог поглавља дате су и полазне хипотезе ове докторске дисертације.

Пето поглавље садржи описе два развијена аналитичка модела спекулативног извршавања. Један модел описује спекулативно извршавање са тачно предвиђеним вредностима операнада. Други модел описује спекулативно извршавање које укључује и тачно и непрецизно предвиђене вредности операнада уколико се са њима добија тачан резултат инструкције.

У шестом поглављу представљена је методологија истраживања које је спроведено у оквиру ове докторске дисертације. У оквиру шестог поглавља описани су спроведени експерименти и коришћени тестови. Такође, описана је надоградња симулатора *Gem5* и развијени симулатор *VPSim*.

У оквиру седмог поглавља представљени су добијени резултати у спроведеним експериментима. На основу добијених резултата дати су коментари у вези са важењем полазних хипотеза.

Осмо поглавље садржи коментаре у вези са могућим правцима даљег истраживања теме ове докторске дисертације. Последње девето поглавље представља закључак ове докторске дисертације. У оквиру њега је дат кратак осврт на спроведено истраживање и представљени су доприноси који су проистекли из ове докторске дисертације.

2. ПРЕДВИЂАЊЕ ВРЕДНОСТИ

У оквиру овог поглавља биће описане основе рада предиктора вредности као и начини на које се могу предиктори вредности користити приликом извршавања инструкција на процесору. Такође, биће представљена подела предиктора према начину на који врше предвиђање. У овом поглављу биће описани изазови и проблеми код механизама предвиђања вредности као што је опоравак процесора услед лошег предвиђања и кашњење које уносе предиктори вредности.

2.1. Основе рада предиктора вредности

Извршавање једне асемблерске инструкције код процесора са проточном обрадом је подељено у више фаза. Свака од фаза (степен проточне обраде) је задужена за извршавање једног дела инструкције. Када се неопходан посао у оквиру једног степена заврши, инструкција прелази у наредни степен. Преласком инструкције у наредни степен, ослобађају се ресурси у том степену за наредну инструкцију која улази у тај степен. На тај начин је извршавање инструкција преклопљено јер се више инструкција извршава на процесору само што се налазе у различитим степенима проточне обраде. Такође, неке организације процесора имају могућност да у оквиру једног степена имају више различитих инструкција. То се постиже тиме што степен има више ресурса које могу користити истовремено различите инструкције.

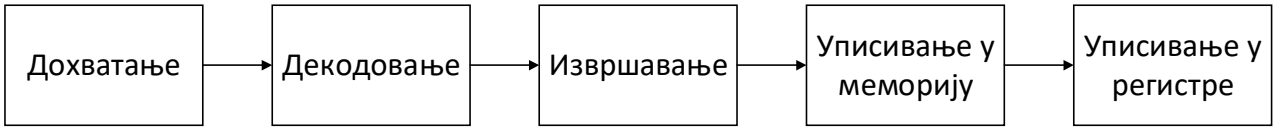
Модел процесора са проточном организацијом под називом *MIPS* (енгл. *Microprocessor without Interlocked Pipelined Stages*) који се најчешће среће у литератури приликом представљања процесора са проточном обрадом описан је у радовима [16], [22] и [23]. На основу овог модела процесора осмишљен је упрошћени модел процесора који се користи у образовне сврхе, а описан је у књизи [8]. Овај процесор организован је у пет степени проточне обраде. Први степен представља дохватање инструкције, тј. учитавање инструкције из меморије. У другом степену врши се декодовање инструкције као и дохватање операнада. Трећи степен је одговоран за извршавање операције над операндима, тј. извршавање инструкције. Четврти и пети степен служе за уписивање резултата у меморију и регистре, респективно.

Предвиђање вредности се може поделити у два корака. Први корак представља само предвиђање вредности када предиктор испоручује предвиђену вредност. Други корак јесте ажурирање предиктора на основу тачне вредности коју је предвиђао у тренутку када она постане доступна. У наставку ће бити објашњено на моделу описаног процесора како се могу користити предиктори вредности, тј. како испоручују предвиђену вредност. Такође, биће објашњено и када и како се врши ажурирање предиктора вредности.

i) Предвиђање вредности

Изнад описан процесор са проточном обрадом може послужити да се прикаже основна идеја извршавања инструкција са коришћењем предиктора вредности. На слици 2.1.1. је приказан такав процесор са проточном обрадом на нивоу фаза проточне обраде, а на слици 2.1.2. исти процесор са додатком механизма предвиђања вредности. У случају када нема

предвиђања вредности, инструкција пролази кроз свих пет фаза проточне обраде: дохватање, декодовање, извршавање, уписивање у меморију и уписивање у регистре. Додатно ће бити усвојено за приказани процесор да фаза извршавања има више ресурса (аритметичко-логичких јединица) чиме је омогућено да више инструкција истовремено буде у овој фази.



Слика 2.1.1. Проточна обрада у пет фаза

Предиктори вредности представљају посебан део процесора чија је улога да врше предвиђање вредности. Уколико се примењује предвиђање вредности, инструкција може да из фазе декодовања настави извршавање коришћењем предиктора вредности. У том случају, вредност која недостаје инструкцији може да се предвиди и са том вредношћу настави спекулативно извршавање. Када инструкција настави спекулативно да се извршава, остале инструкције које зависе од ње могу такође наставити да се извршавају, али су и оне такође спекулативне. У тренутку када тачна вредност предвиђене вредности постане доступна, процесор мора утврдити да ли је предвиђање исправно. Уколико није, процесор мора повратити интерно стање као што је било пре почетка спекулативног извршавања. Вредности које се могу предвиђати за инструкцију биће разматране у потпоглављу 2.2. ове дисертације.

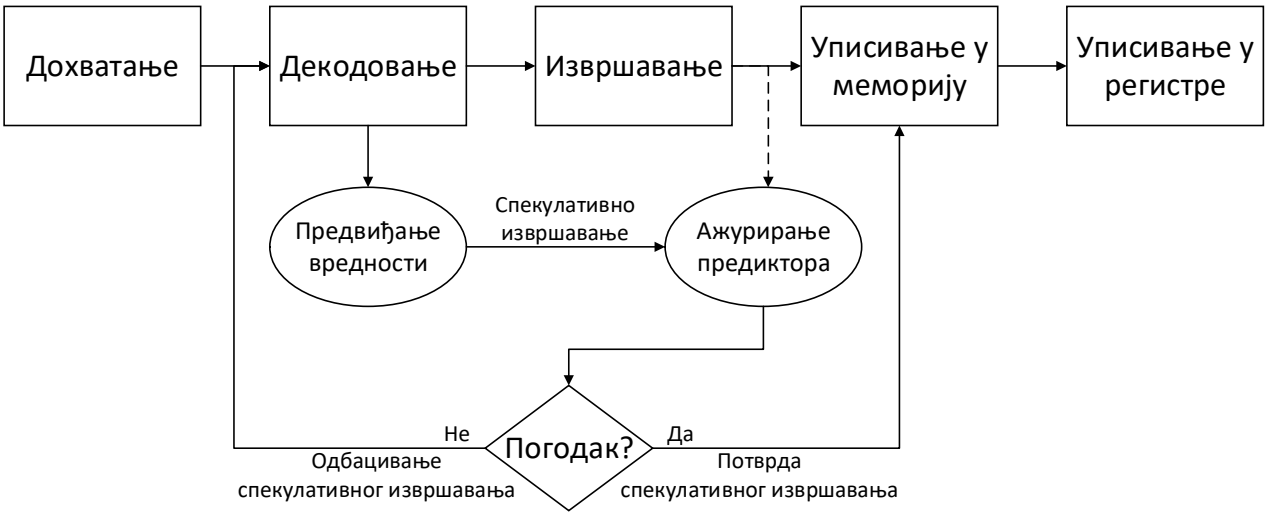
Слика 2.1.2. Проточна обрада у пет фаза са предвиђањем вредности 1

Треба напоменути да спекулативно извршавање подразумева да се инструкције изврше на основу предвиђене вредности. То значи да спекулативна инструкција дође до крајњих фаза проточне обраде у којима се уписује резултат инструкције. Процесор на нивоу своје организације мора имати подршку да информацију о таквим инструкцијама сачува. Процесор таквим инструкцијама не дозвољава да изврше упис у архитектуралне регистре или меморију (оставе траг извршавања), већ их задржава у тим фазама до тренутка када се зна исход предвиђања које је покренуло спекулативно извршавање.

ii) Ажурирање предиктора вредности

На слици 2.1.3. је приказана проширена слика 2.1.2. додатком везаним за ажурирање предиктора. Након што тачна вредност вредности која се предвиђала постане доступна, предиктор је неопходно ажурирати. Ажурирање предиктора се врши на основу тачне и предвиђене вредности као и на основу информација о извршавању (вредност неких регистара) како би предиктор имао ажурно стање за наредна предвиђања. На овај начин се предиктор обучава током извршавања. Такође, треба напоменути да се предиктори ажурирају и онда када за инструкцију нису вршили предвиђање како би се обучили за неко наредно предвиђање вредности, што је на слици 2.1.3. означено испрекиданом везом.

Инструкције које се спекулативно извршавају су инструкције за које се вршило предвиђање и све инструкције које зависе од резултата инструкција за које се вршило предвиђање. Инструкције могу зависити од резултата директно или индиректно преко резултата неке друге инструкције која директно користи резултат инструкције за коју се вршило предвиђање. Све такве зависне инструкције као и сама инструкција за коју се вршило предвиђање остају спекулативне до тренутка када исход предвиђања постане познат.



Слика 2.1.3. Проточна обрада у пет фаза са предвиђањем вредности 2

У тренутку када се зна исход предвиђања, осим што се врши ажурирање предиктора, потребно је још на основу исхода предвиђања одредити даљи пут спекулативних инструкција које зависи од посматраног предвиђања. Уколико је предвиђање било успешно тада се све спекулативне инструкције потврђују, што значи да своје резултате могу уписати у меморију и регистре. У случају да предвиђање није било добро тада се сви резултати спекулативних инструкција одбацују и поново се издају на извршавање, сада само са тачним вредностима.

2.2. Употреба предиктора вредности

Као што је поменуто у уводном поглављу ове докторске дисертације, механизми предвиђања вредности су осмишљени са идејом да се реши проблем праве зависности по подацима на нивоу хардвера [24][25]. У случају да је некој инструкцији потребан податак који треба да произведе нека друга инструкција, тада се може применити предвиђање вредности. Са тим предвиђеним вредностима инструкције могу да наставе спекулативно да се извршавају.

Поред тога што се предиктори вредности користе за предвиђање података како би се ублажио проблем праве зависности по подацима, предиктори се могу користити још и за предвиђање адресе података. У радовима [26] и [27] описан је начин на који се механизам предвиђања вредности може користити за предвиђање адреса података које ће процесор можда у скоријој будућности користити. На тај начин процесор може раније започети дохватање података из меморије, како би се они нашли у кеш меморији у тренутку када буду били заиста потребни. У наведеним радовима је описано понашање инструкција које приступају меморији. Код тих инструкција је примећено да се адресе које се користе за приступ меморији често понављају или мењају по неком обрасцу. На основу тога су

креирани предиктори вредности који служе за предвиђање адресе података, уместо саме вредности података.

У раду [28] је описано како је могуће користити предикторе вредности у комбинацији са предикторима скокова како би се побољшала прецизност предвиђања скокова. Такав предиктор скока се састоји од једног предиктора вредности и једног предиктора скока. Идеја је да предиктор вредности предвиди вредност неког податка на основу кога ће се рачунати да ли је услов за неки условни скок испуњен. Коришћењем те информације предиктор скока врши предвиђање исхода условног скока.

Механизам предвиђања вредности се може користити и код технике којом процесор мења извршавање једне сложене инструкције инструкцијом која је једноставнија у смислу колико времена захтева да се изврши (енгл. *speculative strength reduction*) [29]. Идеја за настанак овог механизма потиче од једне оптимизације коју спроводе преводиоци, а која се заснива на томе да одабере најисплативије инструкције којима се задржава семантика програма написаног на неком вишем програмском језику. Идеја јесте да процесор уместо да извршава неку временски захтевну инструкцију, изврши инструкцију која ће произвести исти резултат али ће бити мање временски захтевна. На пример, посматра се инструкција сабирања која има два изворишна операнда и један дестинациони. Уколико је вредност једног изворишног операнда једнака нули, онда се инструкција сабирања може заменити ефикаснијом инструкцијом као што је инструкција трансфера која ће само вредност изворишног ненултног операнда сместити у дестинациони операнд. Предиктори вредности се овде користе да предвиде да ли неки од операнада инструкције има вредност нула чиме та инструкција постаје кандидат да се замени неком исплативијом инструкцијом.

2.3. Опоравак услед лошег предвиђања

Након што тачна вредност податка чија се вредност предвиђала постане позната, процесор мора извршити проверу да ли је предвиђен податак једнак тачној вредности тог податка. У случају да јесте, процесор наставља извршавање које више није спекулативно јер сада има сигурно тачну вредност податка. У ситуацији да је предвиђање било лоше, процесор мора предузети акције којима повраћа своје интерно стање као што је било пре почетка спекулативног извршавања. Како би се механизам предвиђања вредности искористио што боље, потребно је поред високе прецизности предиктора да постоји и подршка на нивоу организације процесора за ефикасан опоравак услед лошег предвиђања [30].

Предиктори вредности као што је речено нису савршени по питању прецизности као предиктори скокова. Због тога код предвиђања вредности долази често до процеса опоравка. Механизми који се користе за опоравак услед лоше предвиђене вредности су исти механизми који су развијени за потребе опоравка код предиктора скокова. Ти механизми треба да обезбеде да се поврате вредности регистара као што су биле пре спекулативног извршавања. Поред тога, током спекулативног извршавања инструкција, а док се не утврди да ли је њих заиста потребно извршити, не сме се дозволити да оне оставе траг извршавања. Под трагом извршавања подразумевају се промене вредности архитектуралних регистара и меморије.

Два механизма која се могу применити за опоравак процесора са проточном обрадом услед лошег предвиђања описана су у раду [31]. Први механизам јесте механизам испирања проточне обраде (енгл. *pipeline squashing*) који се користи код предвиђања скокова када дође до промашаја. Када се деси промашај у предвиђању исхода скока тада се све инструкције које су ушле у проточну обраду након инструкције скока испирају, тј. поништавају.

Извршавање се након тога наставља учитавањем инструкције у складу са стварним исходом скока. Исти овај механизам се може применити и код предвиђања вредности када се деси погрешно предвиђање.

Други механизам описан у раду [31] јесте механизам селективног поновног извршавања (енгл. *selective reissue*). Овај механизам подразумева да се само инструкције које зависе од предвиђене вредности поништавају и поново учитавају у проточну обраду само са исправном вредношћу. За имплементацију оваквог начина опоравка потребно је да се током извршавања инструкција запамти која је прва инструкција која је користила предвиђену вредност. Затим је потребно памтити и све инструкције које зависе од ње, формирајући на тај начин ланац инструкција које су у спекулативном извршавању. Када се утврди да је предвиђена вредност различита од тачне вредности, тада се поништавају само инструкције које чине формирани ланац, а не све као што је случај код механизма испирања.

Како би се избегло губљење времена на опоравке од лошег предвиђања, у развоју предиктора вредности посебно се обраћа пажња на прецизност предвиђања. Механизми (алгоритми) које су преузели предиктори вредности од предиктора скокова не постижу прецизности које су забележене приликом предвиђања скокова. Због таквог понашања, односно због мање прецизности долази и чешће до потребе да се врши опоравак. Како би се прецизност предиктора повећала, примењена су два приступа у процесу развијања предиктора: смањивање покрвивености (енгл. *coverage*) и увођење механизма поверења (енгл. *confidence mechanisms*).

i) Смањивање покривености

Први приступ од два приступа како би се повећала прецизност предиктора представља смањивање броја различитих типова инструкција за које се врши предвиђање. Владало је мишљење да уколико се повећа број типова инструкција за које се врши предвиђање да ће се перформансе процесора побољшати као и број извршених инструкција по једном циклусу (енгл. *instruction per cycle – IPC*). Такво мишљење је довело до значајног повећања величине предиктора 8-16 KB [32], њихове комплексности и потрошње енергије, што захтева озбиљне промене у организацији процесора.

Због комплексности које сложени и већи предиктори уносе у организацију процесора као и због тога што не постижу очекиване прецизности уколико предвиђају вредности за све типове инструкција, дошло се до идеје да се смањи број типова инструкција за које се врши предвиђање. Примећено је да су неки типови инструкција погоднији за предвиђања од других типова. У радовима [17] и [33] је представљено да су инструкције учитавања из меморије (енгл. *load instructions*) погоднije за предвиђање. То значи да предиктори предвиђају вредности само за те типове инструкција и на тај начин постижу бољу прецизност него стандардни предиктори који врше предвиђања вредности за све типове инструкција. Један предиктор који је представљен у раду [32] врши предвиђање само оних инструкција које могу да изазову „уско грло“ система (енгл. *bottleneck*). Тај предиктор води рачуна о инструкцијама учитавања из меморије јер оне могу да изазову застој у извршавању услед кашњења меморије.

Треба приметити да се за побољшање перформанси процесора мора успоставити компромис између покривености и прецизности предиктора. Уколико се повећава покривеност тада ће се прецизност предиктора смањивати. Такође, важи и обрнуто ако се смањује покривеност тада ће се прецизност предиктора повећавати. Између ове две величине треба успоставити однос који ће довести до побољшања перформанси процесора. Такође, битан чинилац представља и сама намена процесора, односно какви ће се програми

извршавати на њему. То може утицати на одређивање покривености у зависности од тога који типова инструкција су доминантно заступљени у програмима.

ii) Механизми поверења

Други приступ којим се постиже повећање прецизности јесте увођење механизма који одређује када да се врши предвиђање, а када не. Такви механизми се називају механизмима поверења (енгл. *confidence mechanisms*). Ови механизми воде рачуна о успешности претходних предвиђања за неку инструкцију. Уколико су у блиској прошлости предвиђања била успешна, наредни пут када се појави та инструкција овај механизам ће допустити да се врши предвиђање. У ситуацијама, када у блиској прошлости предвиђања нису била успешна, овај механизам неће дозволити да се врши предвиђање. У таквој ситуацији предиктори онда само прате извршавање инструкције како би ажурирали своје интерно стање.

Један од првих механизама поверења под називом динамички механизам поверења (енгл. *dynamic confidence mechanisms*) представљен је у раду [17]. Овај механизам је настао за потребе коришћења код предиктора скокова. Овај механизам функционише тако што се за одређени условни скок прати успешност предвиђања његовог исхода. У једном регистру се бележи успешност предвиђања тако што се на место најнижег бита у том регистру уписује нула или јединица у зависности да ли је предвиђање било успешно или не, респективно. Након тога се вредност регистра помера улево (енгл. *shift left*). Када наредни пут инструкција условног скока дође на извршавање, предвиђање ће се одрадити само у случају ако је вредност поменутог регистра одговарајућа. Његова вредност се пропушта кроз функцију која броји број битова чија је вредност један. Уколико је тај број изнад неке границе (енгл. *threshold*) тада се предвиђање неће одрадити јер је инструкција означена као инструкција која може покварити прецизност предиктора и изазвати трошење времена на опоравак услед лошег предвиђања.

Други пример механизма поверења представља механизам заснован на бројачима (енгл. *saturated counter*) који је представљен у раду [17]. Овај механизам за сваку инструкцију за коју се врши предвиђање вредности додељује један бројач. Бројач се ажурира на основу успешности последњег предвиђања. Уколико је предвиђање било успешно бројач се инкрементира, а у супротном се декрементира. Само када је вредност бројача изнад неке границе процесор ће користити предвиђање које испоручује предиктор, а у супротном ће само ажурирати стање бројача у складу са исходом предвиђања.

Сличан механизам механизму заснованом на бројачима јесте механизам који одржава машину стања (енгл. *state machine*) [13] за све инструкције које предиктор прати. Машина има четири стања од којих два стања говоре да процесор треба користи предвиђање (јако и слабо), док друга два стања говоре да процесор не треба да користи предвиђање (јако и слабо). Прелазак између стања се реализује тако што се одабере почетно стање, а након тога се на основу исхода предвиђања прелази у наредно стање. Неколико начина транзиција из стања у стање је представљено у раду [34].

Предиктори вредности су комплетно хардверски механизми што значи да су недоступни за програмере. На тај начин је отежана и готово неизводљива оптимизација софтвера према предиктору који се налази у организацији процесора. Због тога су примењене и неке технике које се спроводе током процеса превођења програма. Једно моделовање механизма заснованог на бројачима коришћењем Марковљевог модела представљено је у раду [35]. Ово моделовање служи за аналитичку процену колико ће бити промашаја током извршавања програма што може послужити за оптимизацију преводилаца како би генерисали ефикаснији код према предиктору вредности.

Још један механизам поверења који се реализује на нивоу преводиоца представљена је у раду [36]. Овај механизам се заснива на праћењу извршавања програма и како се предиктор вредности понашао током извршавања. На основу прикупљених информација, преводилац може током превођење да инструкције означи посебним ознакама које би помогле процесору да зна да ли је инструкција погодна за предвиђање вредности или није.

2.4. Подела предиктори вредности

Механизми предвиђања вредности (предиктори вредности) могу се поделити у две традиционалне групе на основу начина на који одређују предвиђену вредност [37]. Прву групе чине предиктори који вредност предвиђају тако што користе неко израчунавање на основу претходне виђене вредности када се претходни пут извршавала инструкција (енгл. *computatinal predictors*). Другу групу предиктора чине предиктори који предвиђају вредност тако што прате извршавање формирајући контекст за који везују неку вредност и управо ту вредност предиктори враћају приликом предвиђања (енгл. *contextual-based predictors*).

i) Предиктори који израчунавају предвиђену вредност (енгл. computatinal predictors)

У ову групу спадају предиктори вредности који предвиђену вредност формирају тако што примењују неку функцију над претходно виђеном вредношћу инструкције за коју се врши предвиђање. Уколико посматрамо исечак програмског кода 2.4.1. можемо видети две променљиве *i* и *j*. Променљива *j* у свакој итерацији петље добија вредност променљиве *i*, а након тога се променљива *i* инкрементира. На основу овога се може закључити да ће променљива *j* у свакој наредној итерацији имати вредност већу за један. Уколико је сада потребно предвидети вредност променљиве *j* за наредну итерацију њена вредност може се добити као збир претходна вредност *j* и јединице. Најпознатији и најједноставнији представници ове групе предиктора су *Last Value Predictor* [13], *Load Value Predictor* [17], *Stride Predictor* [13] и *2-delta Stride Predictor* [17], који ће у наставку ове докторске дисертације бити детаљно описани.

```
for i = 0 to 100
being
     j = i;
     i++;
end
```

Исечак програмског кода 2.4.1. Пример израчунавање предвиђене вредности на основу претходне

ii) Предиктори који користе контекст (енгл. contextual-based predictors)

У ову групу спадају предиктори који током извршавања програма формирају контекст. Контекст представља секвенцу вредности које је нека променљива имала у блиској прошлости, уколико посматрамо програмски код на неком вишем програмском језику. Исечак програмског кода 2.4.2. сличан је претходном исечку, само што сада променљива *j* добија вредности према остатку дељења променљиве *i* бројем четири. То значи да ће променљива *j* узимати вредности 0, 1, 2 и 3, а потом поново исте вредности и тако у круг. У таквој ситуацији могу се формирати контексти и њихове придружене вредности. Један контекст је рецимо (1, 2, 3), а његова придружена вредност је нула. То значи да након што променљива *j* добије вредност 1, па 2 и након тога 3, да је њена наредна вредност коју ће имати једнака нули (придружена вредност). Предиктор вредности када препозна током

извршавања да се формирао контекст (1, 2, 3) може за променљиву *j* да предвиди вредност нула. Најпознатији и најједноставнији представник ове групе предиктора је *Finite Context Method Predictor* [38].

Поред тога што предиктори могу пратити контекст који је сачињен од претходних вредности, постоје предиктори који прате извршавање инструкција скокова [39]. Такви предиктори такође могу да се сврстају у ову групу предиктора. Они контексту сачињеном од претходних вредности придружују још и историју извршавања скокова (енгл. *branch history*). Један такав предиктор је *VTAGE Predictor* [31]. Поменути предиктори из ове групе биће детаљно описани у наставку ове докторске дисертације.

```
for i = 0 to 100
being
      j = i mod 4;
      i++;
end
```

Исечак програмског кода 2.4.2. Пример израчунавање предвиђене вредности на основу поновљеног контекста

Као што је речено, ова подела предиктора је традиционална. Данашњи предиктори који су представљени последњих година заправо не могу да се сврстају у неку од ових група јер се њихов рад заснива и на израчунавању предвиђене вредности и на праћењу контекста. Такви предиктори су обично састављени од неколико различитих предиктора и називају се хибридни предиктори. Представници таквих предиктора су представљени на светском такмичењу у предвиђању вредности [21] који ће такође бити описани у овој докторској дисертацији.

Треба напоменути да су претходни примери дати у виду програмских исечака налик вишим програмским језицима коришћени само како би се лакше представиле две групе предиктора. Предиктори вредности предвиђање вредности врше на нивоу асемблерских инструкција, а не на нивоу променљивих и наредби у вишем програмском језику. Процесор није ни свестан вишег програмског језика програма који извршава, већ познаје само инструкције.

2.5. Кашњење механизама предвиђања

Као што је поменуто предиктори вредности су настали на основу предиктора скокова. Предиктори скокова користе различите алгоритме и структуре како би одредили предвиђање. За спровођење алгоритма, односно за приступ структурама предиктора потребно је неко време. То време се назива кашњење предвиђања које се може посматрати као два одвојена времена [40]. Прво време представља време које је потребно да се одреди само предвиђање. Друго време јесте време које је потребно да се изврши ажурирање предиктора након што се утврди да ли је предвиђање било исправно.

Код предиктора скокова је изузетно битно да време које је потребно да се одреди предвиђање буде што мање како би предвиђање могло да се одреди без заустављања извршавања наредне инструкције. Како би предиктор скокова испоручио предвиђање на време, примењују се технике које су описане у радовима [40], [41] и [42]. Основна идеја ових техника јесте да се унапред покрене одређивање предвиђања и на тај начин сакрије његово

кашњење. Поред тога, ове технике укључују уз главни предиктор још и један мањи (једноставнији) предиктор који довољно брзо одређује предвиђање. Његово предвиђање се користи у оним ситуацијама када главни предиктор не стигне на време да одреди предвиђање.

За разлику од предиктора скокова, где је кашњење изузетно битно да буде довољно мало, јер након инструкције скока треба одмах одредити која је следећа инструкција за извршавање, предиктори вредности немају такво ограничење. Приликом предвиђања вредности треба само водити рачуна да одређивање предвиђања траје мање времена од времена које је потребно да се инструкција за коју се врши предвиђање изврши. Још један пример, ако се посматра инструкција која је изазвала промашај у кеш меморији, тада одређивање предвиђања треба да траје краће времена од приступа главној меморији за обраду насталог промашаја у кеш меморији. Ова запажања везана за кашњење предвиђања вредности су такође представљена и у раду [43].

У раду [43] су описана три традиционална начина према којима се врши предвиђање вредности. Ти начини се односе на фазу извршавања инструкције у којој се започиње одређивање предвиђања:

- Током фазе дохватања инструкције (енгл. *at-fetch*) – ово је први начин где се одређивање предвиђања започиње у оквиру фазе дохватања инструкције, односно у почетним фазама проточне обраде. Адреса инструкције се приликом дохватања инструкције одмах прослеђује и предиктору вредности који може започети одређивање предвиђања. Овај начин има два проблема. Први проблем јесте код процесора који дохватају више инструкција у једном циклусу (енгл. *trace cache proccessors*) јер тада није лако извући појединачне адресе инструкција (инструкције нису исте ширине или адресе нису континуалне јер постоје инструкције скокова). Други проблем јесте што у фази дохватања инструкције није познато о ком типу инструкције се ради, па би тако све инструкције онда приступале структурама предиктора, а неке нису можда погодне за предвиђање. Предност овог начина се огледа у томе што се време потребно за предвиђање потпуно сакрива, јер предвиђена вредност ће бити вероватно формирана пре него што заиста и буде била потребна.
- Након фазе декодовања инструкције (енгл. *post-decode*) – ово је други начин где се одређивање предвиђања започиње након фазе декодовања инструкције, односно у фазама проточне обраде које следе након што се почетне фазе у којима се врши дохватање заврше. Фаза декодовања подразумева да процесор препозна о којој инструкцији се ради, као и да дохвати њене операнде. Оно што су били проблеми код првог начина, код овог начина више не постоје, јер се сада зна адреса инструкције као и њен тип. Проблем код овог начина јесте што сада није сигурно да ће се предвиђена вредност испоручити на време, јер се сада касније (у каснијим фазама проточне обраде) креће са одређивањем вредности.
- Након завршетка инструкције (енгл. *decoupled*) – ово је трећи начин где се одређивање предвиђања започиње одмах након што се извршавање инструкције заврши. Та инструкција се одмах упућује ка предиктору како би предиктор испоручио предвиђену вредност. Предвиђена вредност се потом памти у посебној структури (табели) заједно са адресом те инструкције. Наредни пут када дође на извршавање, предвиђање за ту инструкцију се добија једноставним

приступом тој структури. Проблем који се овде јавља јесте што се предвиђање одређује одмах након завршетка инструкције и на тај начин су предиктори ускраћени о информацијама које би имали да се предвиђање врши када се инструкција тек дохвата. Ово се односи пре свега на то да предиктори неће имати информације о инструкцијама које су се извршиле пре посматране инструкције.

У већини научних радова у којима су представљени предиктори вредности нису разматране чињенице у вези са кашњењем који ови предиктори уносе. Један од разлога због чега је то тако, јесте што прецизности ових предиктора још увек нису савршене у случајевима када је покривеност велика. Самим тим главни фокус током данашњих истраживања и развијања предиктора је усмерен само ка прецизности и покривености.

3. ПОСТОЈЕЋИ ПРЕДИКТОРИ ВРЕДНОСТИ

У оквиру овог поглавља биће описани најчешће помињани постојећи предиктори вредности у научним радовима као и са светског такмичења у предвиђању вредности. Сваки предиктор биће описан кроз три секције: резиме, реализација и евалуација. У секцији резиме биће укратко представљена основна идеја функционисања предиктора као и наведене референце радова где су предиктори први пут били представљени. У секцији реализација биће детаљније описан начин и могуће варијанте функционисања предиктора (слични предиктори). Такође, у овој секцији биће приказане упрошћене принципске шеме предиктора. Секција евалуација описаће резултате и тестове који су коришћени у оквиру радова у којима су представљени предиктори. Такође, секција евалуација садржи информације о типовима инструкција за које предиктори врше предвиђање вредности као и информације о томе које вредности предвиђају (резултат инструкције или адресе којима ће се приступати).

У табели 3.1. дат је сажет преглед описаних постојећих предиктора вредности. У оквиру табеле су наведени називи предиктора и референце радова у којима су представљени предиктори. Затим табела садржи информације ко су аутори предиктора и са које институције су аутори. Табела садржи информацију ком типу предиктора припада предиктор. Типови предиктора су *computational*, *contex-based* и хибридни тип који представља комбинацију прва два типа. Након типа предиктора у табели је наведено за које типове инструкција предиктори предвиђају вредност према информацијама које су доступне у радовима у којима су предиктори представљени. Типови инструкција су означени са *ALU* (аритметичко-логичке инструкције) и *LOAD* (инструкције дохватања података из меморије). Последња колона ове табеле садржи информацију да ли предиктор користи механизам поверења.

Треба прокоментарисати да у табели 3.1. нису дате никакве информације у вези са прецизношћу предиктора као и покривености. Разлог за то јесте што у оригиналним радовима у којима су представљени предиктори, предиктори су евалуирани на различите начине. Коришћени су различити тестови и симулациона окружења, па због тога није релевантно у сажетом прегледу предиктора наводити те информације.

У табели 3.1. у одговарајућој ћелији присуство црне тачке означава да предиктор испуњава неку одлику. Уколико се у ћелији налази знак косе црте то значи да предиктор не испуњава одговарајућу одлику. Табела 3.1. је приказана на наредној страни како не би била преломљена.

Табела 3.1. Приказ постојећих предиктора вредности

НАЗИВ ПРЕДИКТОРА	АУТОРИ [ИНСТИТУЦИЈА]	ТИП ПРЕДИКТОРА			ТИП ИНСТРУКЦИЈЕ		МЕХАНИЗАМ ПОВЕРЕЊА
		COMPUTATIONAL	*CONTEX -BASED*	ХИБРИДНИ	*ALU*	*LOAD*	
Last Value [44]	Mikko Lipasti, John Paul Shen [Carnegie Mellon University]	●	/	/	●	●	/
Stride [13]	Freddy Gabbay, Avi Mendelson [Technion Israel Institute of Technology]	●	/	/	●	●	/
Register-file [13]	Freddy Gabbay, Avi Mendelson [Technion Israel Institute of Technology]	●	/	/	●	●	/
2-delta Stride [15]	Richard J. Eickemeyer, Stamatis Vassiliadis [IBM]	●	/	/	●	●	/
Load Value [17]	Mikko Lipasti, Chris B Wilkerson, John Paul Shen [Carnegie Mellon University]	●	/	/	/	●	/
Finite Context Method [38]	Yiannakis Sazeides, James E. Smith [University of Wisconsin-Madison]	/	●	/	●	●	/
Global Context-Based Value [45]	T. Nakra, R. Gupta, M.L. Soffa [University of Pittsburgh]	/	●	/	●	●	/
EVES [46]	Andre Seznec [INRIA]	/	/	●	●	●	●
CBC-VTAGE [47]	Yasuo Ishii [ARM]	/	/	●	●	●	●
H3VP [48]	Kenichi Koizumi, Kei Hiraki, Mary Inaba [University of Tokyo]	/	/	●	●	●	●

3.1. *Last Value* предиктор

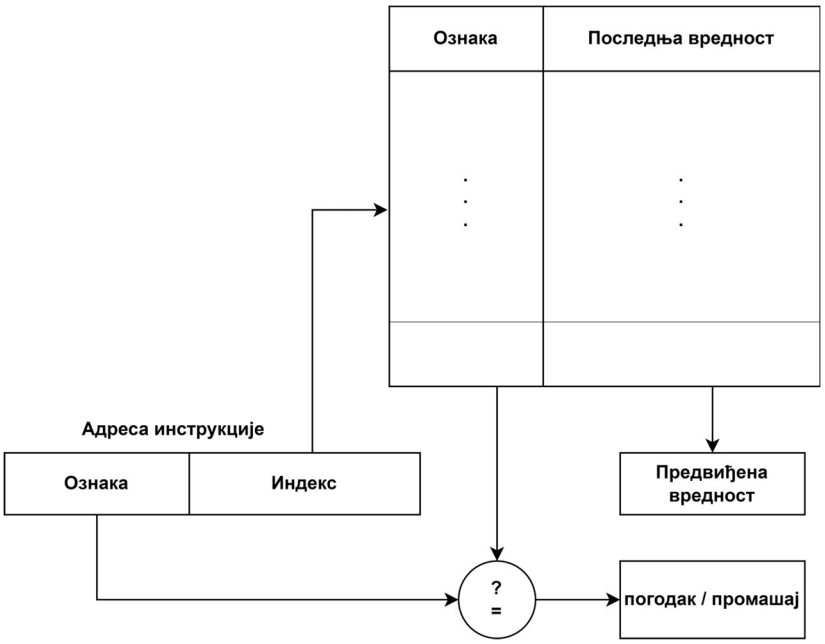
i) Резиме

Last Value предиктор је представљен у радовима [13] и [44]. Овај предиктор је изузетно једноставан по питању начина рада. Предвиђање се врши на основу последње вредности која се појавила (последња виђена вредност). Када на извршавање дође инструкција за коју треба да се врши предвиђање тада се као предвиђена вредност испоручује вредност која је забележена приликом последњег ажурирања предиктора.

ii) Реализација

Овај предиктор се реализује у виду једног регистра који чува последњу виђену вредност. Када треба извршити предвиђање, вредност која се испоручују јесте вредност из тог регистра. Ажурирање се врши када тачна вредност постане доступна и она се само уписује у тај регистар. Та вредност ће управо бити предвиђена вредност за неку наредну инструкцију за коју се врши предвиђање.

Бољa реализација овог предиктора јесте да последња виђена вредност буде везана за појединачне инструкције. Ово се реализује у виду табеле која садржи две колоне: ознаку (енгл. *tag*) и вредност. Табели се приступа на основу најнижих битова адресе инструкције за коју се врши предвиђање, док преостали бити представљају ознаку. Уколико је ознака која је сачувана у улазу (реду) табеле који је адресиран једнака ознаци из адресе инструкције, тада се за предвиђање узима вредност која је сачувана у том улазу табеле. Оваква реализација је погодна за процесоре са проточном обрадом јер се овој табели може приступити у раним фазама проточне обраде. Принципска шема ове реализације *Last Value* предиктора приказана је на слици 3.1.1.



Слика 3.1.1. *Last Value* предиктор [13]

У раду [44] је описана реализација овог предиктора слична изнад описаној, само што табела нема колону за чување ознаке. На тај начин више различитих инструкција може користити исти улаз у табели уколико су им бити адресе на основу којих се приступа табели

исти. На тај начин долази до колизије где једна инструкција може добити предвиђену вредност на основу неке друге инструкције.

iii) Евалуација

Овај предиктор је коришћен за предвиђање вредности резултата аритметичко логичких инструкција и инструкција учитавања из меморије. У раду [13] су представљени резултати на групи тестова (енгл. *benchmarks*) *SPEC95*. На овим тестовима предиктор је забележио прецизност од 11% до 76% за инструкције учитавање из меморије, док је за аритметичко-логичке инструкције прецизност од 39% до 76% у зависности од теста.

3.2. *Stride* предиктор

i) Резиме

Примећено је да програми испољавају понашање такво да производе вредности које се могу представити збиром претходне вредности и корака, где корак представља разлику последње две виђене вредности. Ово понашање није занемарљиво и због тога је било мотивација за настанак *Stride* предиктора. Такође, данашњи најбољи предиктори, који су хибридни, користе *Stride* предиктор као један од својих предиктора.

Предиктор *Stride* је представљен у раду [13]. Основна идеја овог предиктора јесте да предвиђену вредност рачуна на основу претходно виђене вредности и корака (енгл. *stride*). Предвиђена вредност се одређује као збир претходне виђене вредности и корака.

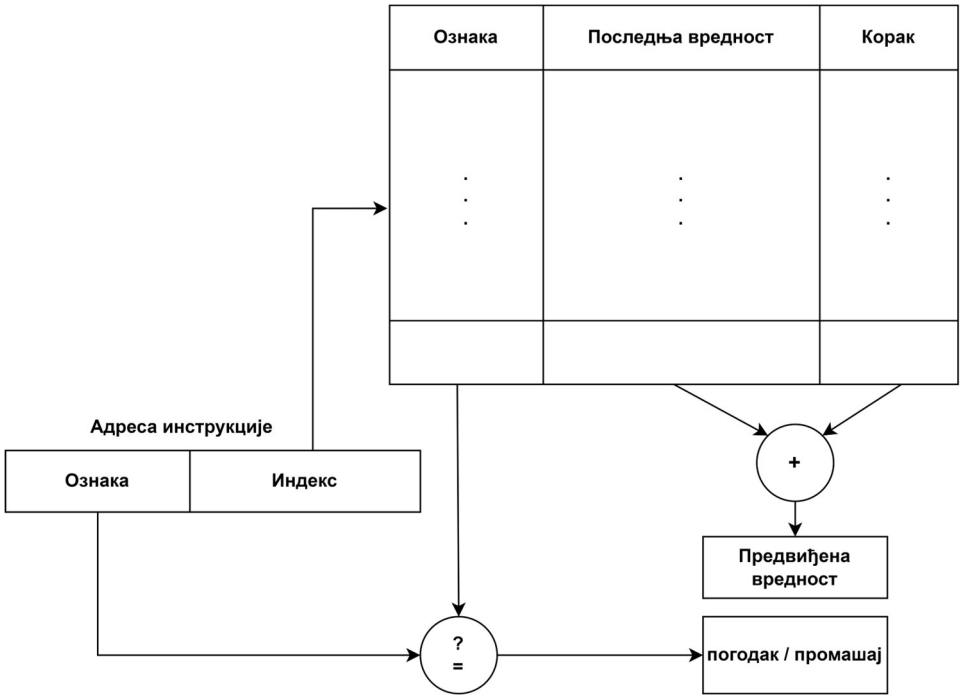
ii) Реализација

Овај предиктор слично као и *Last Value* предиктор је реализован у виду табеле и приказан је на слици 3.1.2. Табела садржи три колоне: ознака, последња вредност и корак. Прве две колоне су идентичне колонама као код *Last Value* предиктора. Трећа колона корак садржи разлику последње две виђене вредности за одређену инструкцију. Предвиђена вредност се добија сабирањем последње вредности са тренутном вредношћу корака. Осим што се врши ажурирање последње вредности, потребно је да се изврши и ажурирање корака када тачна вредност буде доступна.

У раду [46] је представљена побољшана верзија овог предиктора *Enhanced Stride* предиктор. Идеја је да се реши следећи проблем који се јавља код процесора са проточном обрадом. Заправо, предвиђање се врши у неком почетном степену проточне обраде, док се ажурирање предиктора врши у неком каснијем степену. У међувремену, између једног предвиђања и тренутка када се врши ажурирање предиктора, може доћи иста инструкција на извршавање. Ово је случај који се јавља у петљама где рецимо између два извршавања исте инструкције у петљи не постоји велики број инструкција, што води ка томе да се више инстанци исте инструкције нађу истовремено у проточној обради. *Enhanced Stride* предиктор овај проблем ублажава тако што води рачуна колико инстанци инструкције је у проточној обради, а да још увек за њих није извршено ажурирање предиктора. Предвиђена вредност за нову инстанцу се тада рачуна као збир претходне запамћене вредности и производа броја инстанци у проточној обради и тренутне вредности корака.

Још једна варијанта овог предиктора је представљена у раду [49]. У овом раду је објашњен феномен који је уочен да испољавају програми током извршавања, а тиче се вредности које инструкције производе. Раније је још примећено да инструкције производе вредности које се разликују за корак, а у раду [49] је још објашњено да корак може бити константан у неком временском периоду. Реализовани предиктор заправо врши предвиђање да ли ће корак за наредно предвиђање вредности неке инструкције остати исти (енгл. *Stride*

Equality Prediction). Такав предиктор се добро показао за програме који испољавају феномен да у неком временском периоду имају константан корак. Аутори овог предиктора истичу да реализовани предиктор надмашује прецизност *Enhanced Stride* за више од 5%.



Слика 3.2.1. *Stride* предиктор [13]

iii) Евалуација

Stride предиктор је коришћен за предвиђање вредности резултата инструкција и то аритметичко-логичких као и инструкција учитавања из меморије. Тестови који су користили аутори у раду [13] јесу *SPEC95*. Предиктор је постигао прецизност од 35% до 95% за аритметичко-логичке инструкције, док је за инструкције учитавања забележена прецизност од 9% до 75%, у зависности од тестова.

3.3. *Register-file* предиктор

i) Резиме

Register-file предиктор је описан у раду [13] и јако је сличан обичном *Stride* предиктору, на исти начин формирају предвиђену вредност. Разлика је у томе што се сада не памте вредности резултата појединачних инструкција, већ се чувају последње виђене вредности дестинационих регистара.

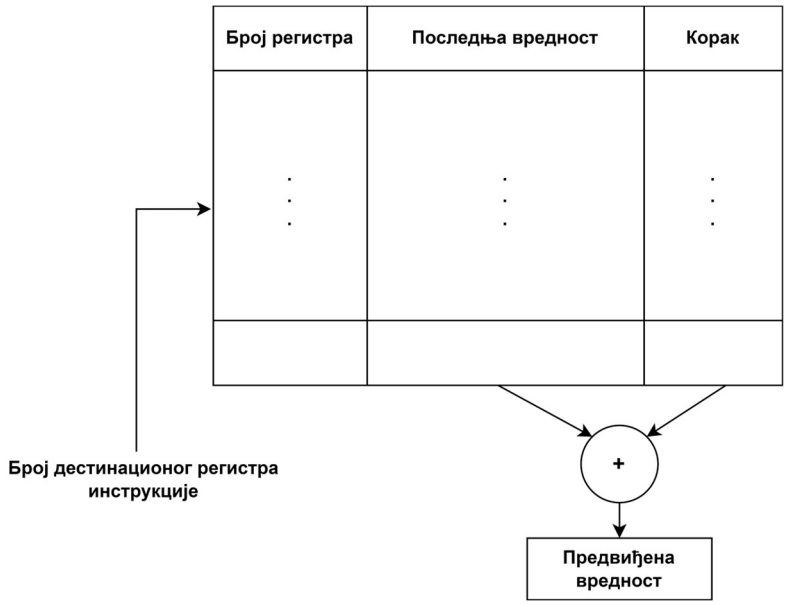
ii) Реализација

Овај предиктор је реализован у виду табеле баш као и *Stride* предиктор и приказан је на слици 3.3.1. За адресирање табеле се користи број дестинационог регистра инструкције за коју је потребно извршити предвиђање. Када се приступи улазу који одговора том регистру, предвиђена вредност се формира сабирањем последње виђене вредности за тај регистар и корака. Корак у овом случају представља разлику између последње две виђене вредности у посматраном регистру. Принципска шема овог предиктора је приказана на слици 3.3.1.

Оно што је интересантно код овог предиктора јесте чињеница да се последња вредност за неки регистар поставља на основу последње инструкције која је изменила тај регистар, тј.

која је извршила упис у њега. Уколико нека друга инструкција буде користила исти тај регистар као свој дестинациони регистар, предвиђање ће се вршити заправо на основу вредности коју је поставила претходна инструкција. Ово није случај код обичног *Stride* предиктора јер се у табели у сваком улазу чува ознака која одређује инструкцију.

Још једна ствар која овај предиктор издваја од *Stride* предиктора јесте његова величина. Овај предиктор је значајно мање величине јер његова табела има само онолико улаза колико има и архитектуралних регистара. Сваки улаз у табели одговара једном архитектуралном регистру.



Слика 3.3.1. *Register-file* предиктор [13]

iii) Евалуација

Као и претходна два описана предиктора и овај предиктор је коришћен за предвиђање вредности резултата аритметичко-логичких инструкција и инструкција учитавања из меморије. Аутори овог предиктора [13] су користили *SPEC95* тестове и на њима је значајно лошије резултате постигао од претходна два предиктора. Прецизност коју је остварио предвиђајући вредност резултата аритметичко-логичких инструкција креће се од 4% до 39%, а за инструкције учитавања из меморије од испод 1% до 13%.

3.4. *2-delta Stride* предиктор

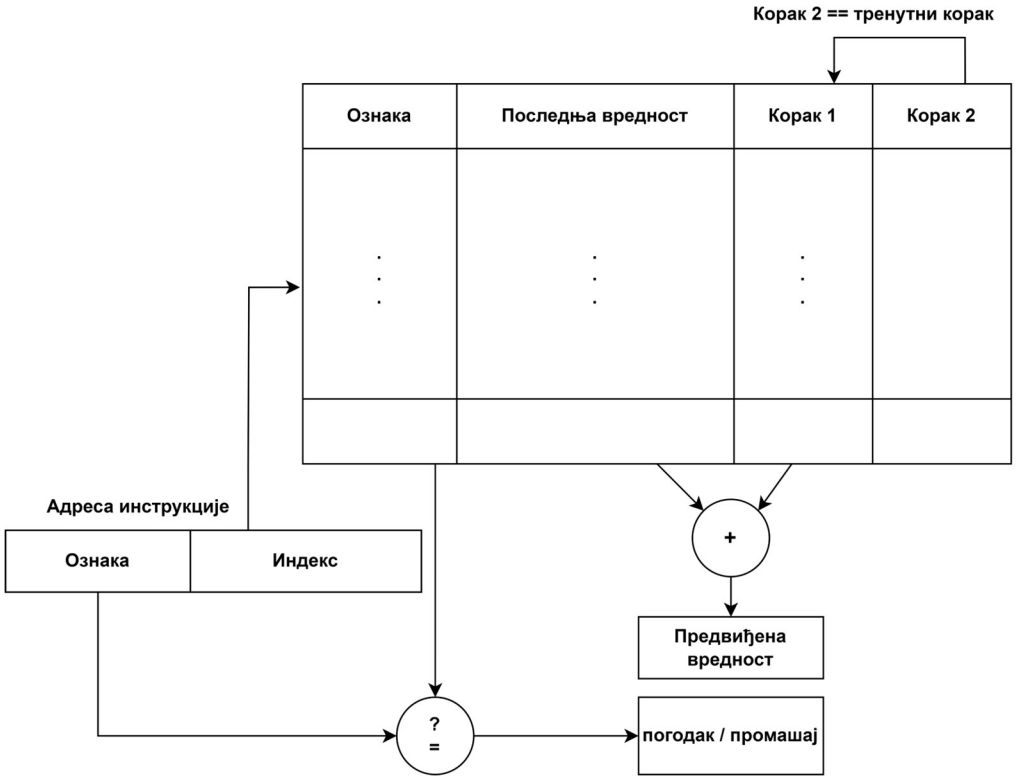
i) Резиме

Овај предиктор је описан у раду [15]. Основна идеја је иста као код *Stride* предиктора са разликом да *2-delta Stride* предиктор има два корака. Корак који се користи за рачунање предвиђене вредности се мења само ако се иста вредност корака појавила два пута заредом. Аутори истичу да је овај предиктор добар за неке уочене секвенце када се неке вредности мењају на основу константног корака. Пример за то је приступ елементима низа. Ако је први елемент низа на адреси 100 и сваки елемент заузима две меморијске локације, тада секвенца изгледа 100, 102, 104, 106… Претпоставимо да је последњи елемент на локацији 110 и да након тога програм опет приступа првом елементу, тада би обичан *Stride* предиктор променио корак, док *2-delta Stride* неће променити корак и задржаће вредност 2 за корак. Ово доводи да ће постојати само један промашај у предвиђању, а то је када се са последњег

елемента прелази поново на први. *Stride* предиктор би у оваквој ситуацији направио два промашаја јер корак мења након сваког предвиђања. Аутори су навели још један пример. Уколико се посматра секвенца вредности 100, 102, 100, 102, … тада би *Stride* предиктор сваки пут правио грешку, док би *2-delta Stride* предиктор сваки други пут правио грешку.

ii) Реализација

Предиктор *2-delta Stride* се може реализовати на исти начин као и *Stride* предиктор у виду табеле која има још једну додатну колону за други корак као што је приказано на слици 3.4.1. Табели се приступа коришћењем највиших битова адресе инструкције. Предвиђена вредност се формира на основу последње виђене вредности и корака из колоне корак 1. Након предвиђања вредност у колони корак 2 се ажурира тако што се поставља на тренутну вредност корака. Уколико је вредност корака 2 једнака тренутној вредности корака тада се вредност из корака 2 уписује у корак 1. Тренутна вредност корака се рачуна у тренутку када се ажурира предиктор и представља разлику између праве (тачне) вредности која се предвиђала и последње запамћене вредности.



Слика 3.4.1. *2-delta Stride* предиктор

iii) Евалуација

Овај предиктор је у раду [15] у коме је представљен коришћен за предвиђање адреса података како би подаци били дохваћени унапред (енгл. *data prefetching*). Идеја је да се предвиде вредности адреса које ће процесор користити. На основу тих адреса се потом ради приступ кеш меморији. Уколико тражених података нема у кеш меморији, биће раније покренут процес дохватања података из главне меморије у кеш. На тај начин уколико заиста касније нека инструкција користи податке са неких од предвиђених адреса, ти подаци ће се налазити већ у кеш меморији.

У спроведеном истраживању [15] акценат није био да се тачно предвиди вредност неког податка, већ да се предвиде адресе које ће се користити. Због тога се овде не може навести прецизност овог предиктора на основу рада у коме је представљен. Међутим, аутори су у раду истакли да је време извршавања инструкција просечно смањено за 0,45 циклуса коришћењем технике дохватања података унапред, што су утврдили спроводећи симулације на *IBM ESA/390* архитектури.

3.5. *Load Value* предиктор

i) Резиме

Load Value предиктор је представљен у раду [17]. Његова основна идеја јесте да врши предвиђање вредности за инструкције учитавања из меморије како би се ублажио проблем кашњења које меморијски систем уноси. Идеја овог предиктора јесте да се врши предвиђање резултата, односно дестинационог операнда инструкција учитавања из меморије. На тај начин би се у неким ситуацијама избегло трошење времена на чекање да се заврши приступ главној меморији.

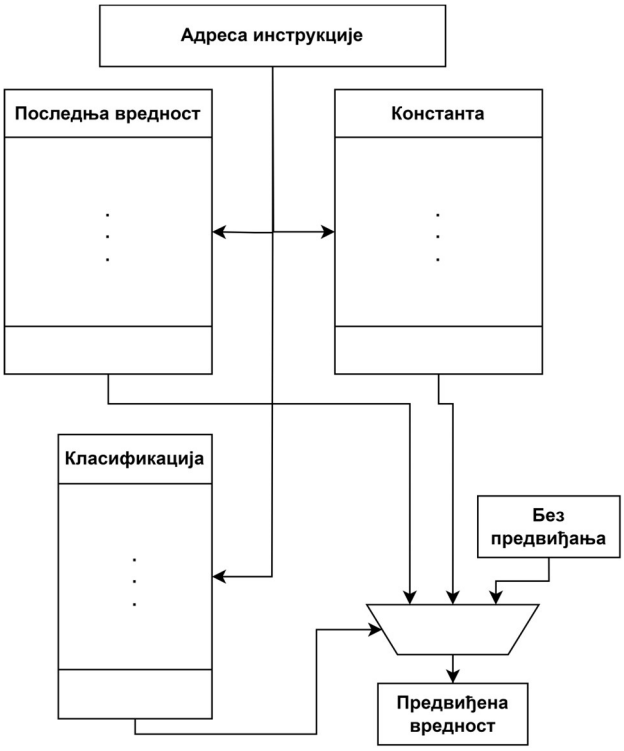
Овај предиктор такође води рачуна и о томе да ли су све инструкције учитавање из меморије погодне за предвиђање. Због тога предиктор има посебан део који врши класификацију инструкција учитавања из меморије у току извршавања програма. Препознате су три групе. Прву групу чине инструкција учитавања које нису погодне за предвиђање (енгл. *unpredictable*) и за ову групу се не врши предвиђање. Другу групу чине инструкције које су погодне за предвиђање (енгл. *predictable*) и за њих се врши предвиђање. Последњу групу чине инструкције које током извршавања често као свој резултат увек имају исту вредност па су означене као константе (енгл. *constant*). Због тога предиктор поседује посебну табелу у којој чува копије из меморије тих константи како би предиктор одмах вратио одређену константу као предвиђену вредност. Када се нека од тих вредности мења од стране инструкције која врши упис у меморију (енгл. *store instruciton*) тада се мења и вредност у тој табели.

ii) Реализација

Овај предиктор је реализован кроз три јединице које су на слици 3.5.1. представљене у виду три табела. Табела у којој се чува последња виђена вредност за неку инструкцију учитавања у раду [17] је названа *Load Value Prediction Table* (*LVTP*). Њој се приступа на основу најнижих битова адресе инструкције. Може се приметити да нема ознаке тако да је присутна колизија, тј. да један улаз користе више различитих инструкција.

Друга јединица из рада [17] названа *Constant Verification Unit* (*CVP*) на слици 3.5.1. је представљена у виду табеле која чува вредности дестинационог операнда оних инструкција учитавања које припадају групи константи. Заправо у тој табели се чувају вредности меморијских локација које током извршавања не мењају често вредност. Ова табела се адресира на исти начин као и *LVTP*.

Трећа јединица у раду [17] названом *Load Clasification Table* (*LCT*) на слици 3.5.1. је представљена као табела која садржи информацију којој групи припада инструкција учитавања које су изнад описане. Овој табели се такође приступа на основу битова адресе инструкција и нема ознаке, тако да је и овде присутна колизија. На основу вредности која се очита из ове табеле доноси се коначна одлука о предвиђању: да ли ће бити предвиђена вредност последње виђена вредност, да ли ће бити константа или се неће вршити предвиђање.



Слика 3.5.1. ***Load value*** **предиктор**

iii) Евалуација

У раду [17] су коришћени тестови који су преузети из скупа тестова *SPEC92* и *SPEC95* као и два програма за обраду слике и два *UNIX* програма (енгл. *unix utilities*). Симулације којим су евалуирани резултати су спровођене над два модела процесора, једним са *in order* и другим са *out of order* извршавањем. У раду [17] није представљена посебно прецизност предиктора, већ је наведено да је добијено просечно убрзање извршавања на коришћеним тестовима 7,7% за најбољу конфигурацију предиктора.

3.6. *Finite Context Method* (*FCM*) предиктор

i) Резиме

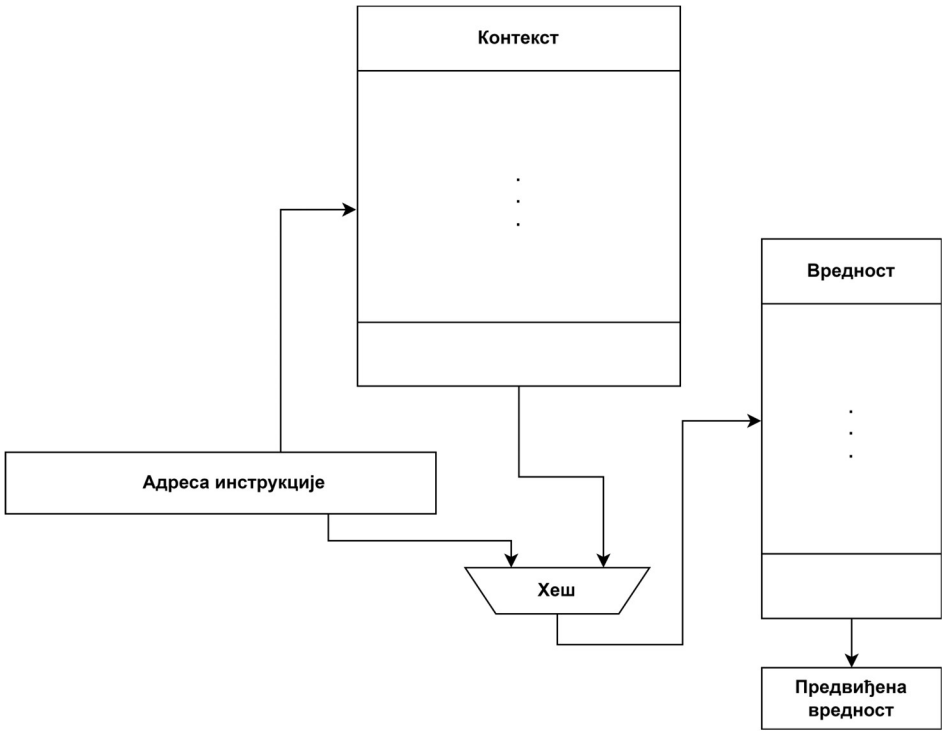
Овај предиктор је описан у раду [38]. Предиктор током извршавања формира контексте које чине вредности које су се појављивале. За сваки контекст се везују и неке вредности, а предвиђање се врши тако што се нека од тих вредности испоручује као предвиђена вредност. У тренутку када се врши предвиђање за неку инструкцију, тада се на основу контекста који је претходно запамћен за ту инструкцију бира једна од његових придружених вредности. У поглављу 2.4. ове докторске дисертације је дато детаљно објашњење везано за контекст.

ii) Реализација

Предиктор је реализован у виду две табеле. У једној табели се чувају претходни контексти који су запамћени током извршавања, у раду [38] у којем је представљен предиктор ова табела је названа *Value History Table* (*Context*). Друга табела служи да чува вредности које су придружене контекстима, а у раду [38] је названа *Value Prediction Table*. Табели са контекстима, како аутори предиктора истичу, може се приступити на основу неких информација које чине тренутно стање процесора као што су: адреса тренутне инструкције,

тип тренутне инструкције, глобална историја скокова или на основу вредности неких други регистара процесора. Како би сви предиктори били униформно приказани у оквиру ове дисертације, на слици 3.6.1. која приказује овај предиктор, усвојено је да се табели са контекстима приступа на основу адресе инструкције. Када се очита контекст онда се он заједно са неким информацијама које представљају стање процесора пропушта кроз хеш функцију чији излаз представља индекс за приступ табели са вредностима.

У самом раду [38] је представљена једна хеш функција коју чине три функције. Прва функција је *Select-Concatenate* која од улазних вредности контекста бира неке битове тих вредности које задржава. Друга функција *Select-Fold-Concatenate* затим спроводи операцију ексклузивног или над одабраним битовима. Трећа хеш функција *Select-Fold-Shift-Xor* резултујуће вредности друге функције додатно још помера улево, а затим над испомераним вредностима спроведе још једном операцију ексклузивног *или*. На тај начин се од почетног контекста који има више вредности добија једна вредност. Та вредност служи као индекс за приступ табели са вредностима. Касније је у раду [50] представљена побољшана варијанта описане хеш функције. Примећено је да оригинална хеш функција често није давала индексе који покривају целу табелу вредности, што је није случај са функцијом из рада [50].



Слика 3.6.1. *Finite Context Method* предиктор [38]

Једна од варијација овог предиктора је представљена у раду [51] и тај предиктор је назван *Differential FCM* (*DFCM*) предиктор. *DFCM* предиктор уместо да предвиђа вредност као *FCM* он предвиђа корак, а предвиђену вредност формира као збир последње виђене вредности из контекста и предвиђеног корака. Аутори овог предиктора кажу да је прецизност овог предиктора боља за 15% од прецизности *FCM* предиктора.

iii) Евалуација

Аутори овог предиктора [38] су за евалуацију користили подскуп тестова из скупа тестова *SPEC95*. Предиктор је коришћен за предвиђање резултата свих инструкција чији је дестинациони операнд специфициран неким регистром опште намене. На њима је овај

предиктор забележио прецизности од нешто испод 30% до нешто преко 90% у зависности од теста.

3.7. *Global Context-Based Value* предиктор

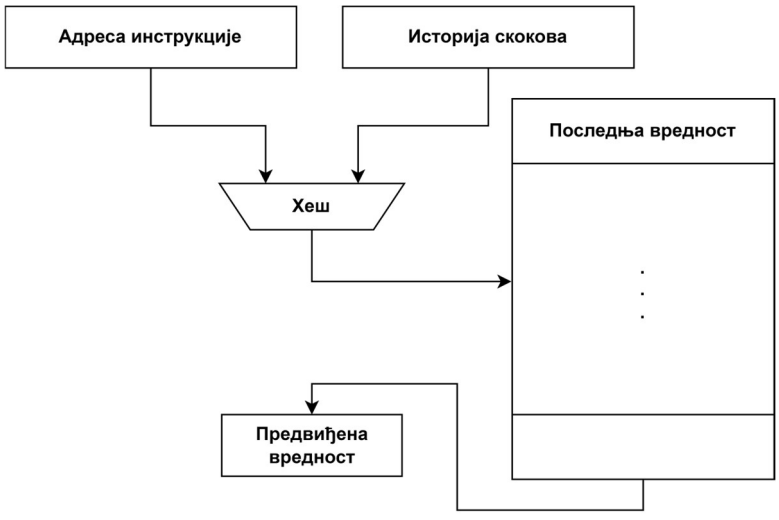
i) Резиме

Овај предиктор је представљен у раду [45]. *Global Context-Based Value* предиктор предвиђање вредности за неку инструкцију врши на основу извршавања осталих инструкција (енгл. *path information*). Уколико се посматра извршавање једне инструкције, њен резултат може зависити од путање извршавања која је довела до ње. Каква ће путања извршавања бити, односно које ће се инструкције извршити пре посматране инструкције, а које неће на то утичу инструкције условних скокова.

Историја како су се извршавали условни скокови (енгл. *branch history*) се памти у оквиру једног померачког регистра [52]. Када се изврши инструкција условног скока регистар се помера. Уколико се последњи извршени условни скок десио (услов је био задовољен) на место бита који се ослободио се уписује јединица. У ситуацији када услов за последњи извршени условни скок није био задовољен (скок се није десио) тада се на место слободног бита уписује нула. На овај начин се памти глобална историја извршавања условних скокова коју овај предиктор користи приликом предвиђања вредности.

ii) Реализација

Аутори овог предиктора су представили три реализације у оквиру рада [45]. На слици 3.7.1. је приказана једна од њих која се заснива на последње виђеној вредности. У овој реализацији постоји једна табела у којој се чувају последње виђене вредности. Овој табели се приступа на основу индекса који формира хеш функција на основу улазних вредности. Улазне вредности за хеш функцију су адресе инструкције и тренутна вредност регистра који прати глобалну историју извршавања условних скокова. Када хеш функција формира индекс, очитана адреса из одговарајућег улаза табеле представља предвиђену вредност.



Слика 3.7.1. Global *Context-Based Value* предиктор [45]

Хеш функција која је у раду [45] представљена формира индекс за приступ табели са вредностима на следећи начин. Прво адресу инструкције помера улево за онолико места колика је величина у битовима регистра историје скокова. На ту вредност затим додаје

вредност регистра историје скокова. Последњи корак јесте одређивање остатка при целобројном дељењу добијене вредности са бројем улаза које има табела са вредностима.

Друга реализација овог предиктора поред тога што користи последње виђене вредности за предвиђање додатно има и табелу у којој се чувају кораци. У раду [45] је назван *Per-path Stride Per-path Value* (*PS-PLV*) предиктор. На основу корака који се очита из те табеле и на основу последње виђене вредности која се очита из табеле вредности, формира се предвиђена вредност као збир те две вредности. Ова реализација заправо се може посматрати као *Stride* предиктор само што се за приступ табелама поред адресе инструкције користи и историја скокова, тј. користи се информације о путањи којом се дошло до посматране инструкције за коју се врши предвиђање вредности.

Трећа реализација овог предиктора под називом *Per-Path Stride Per-Instruction* (*PS-PI*) представља заправо један хибридни предиктор који чине предиктор *PS-PLV* и *Last Value* предиктор. Истовремено се врши приступ до оба предиктора приликом предвиђања, тако да оба предиктора дају предвиђање. У сваком улазу табеле *PS-PLV* предиктора која чува кораке постоји двобитни бројач. Бројач у одговарајућем улазу након предвиђања се инкрементира за случај да је *PS-PLV* предиктор дао тачно предвиђање, а декрементира се у случају да је *Last Value* предиктор дао тачно предвиђање. На основу тренутне вредности овог бројача бира се коначно предвиђање, тј. чија предвиђена вредност од два предиктора ће бити испоручена као коначна предвиђена вредност.

iii) Евалуација

Аутори овог предиктора [45] су за потребе евалуације користили подскуп тестова *SPEC95* као и неке *UNIX* програме. На коришћеним тестовима аутори су истакли да трећа реализација, а то је хибридни предиктор *PS-PI*, надмашује конвенционални *Stride* предиктор за нешто више од 10%. Такође, вредно помена у вези са резултатима овог предиктора јесте да је реализацијом овог предиктора показано да коришћењем информација о извршавању других инструкција приликом предвиђања, пре свега инструкција условних скокова, утиче на побољшање прецизности предиктора.

3.8. *EVES* предиктор

i) Резиме

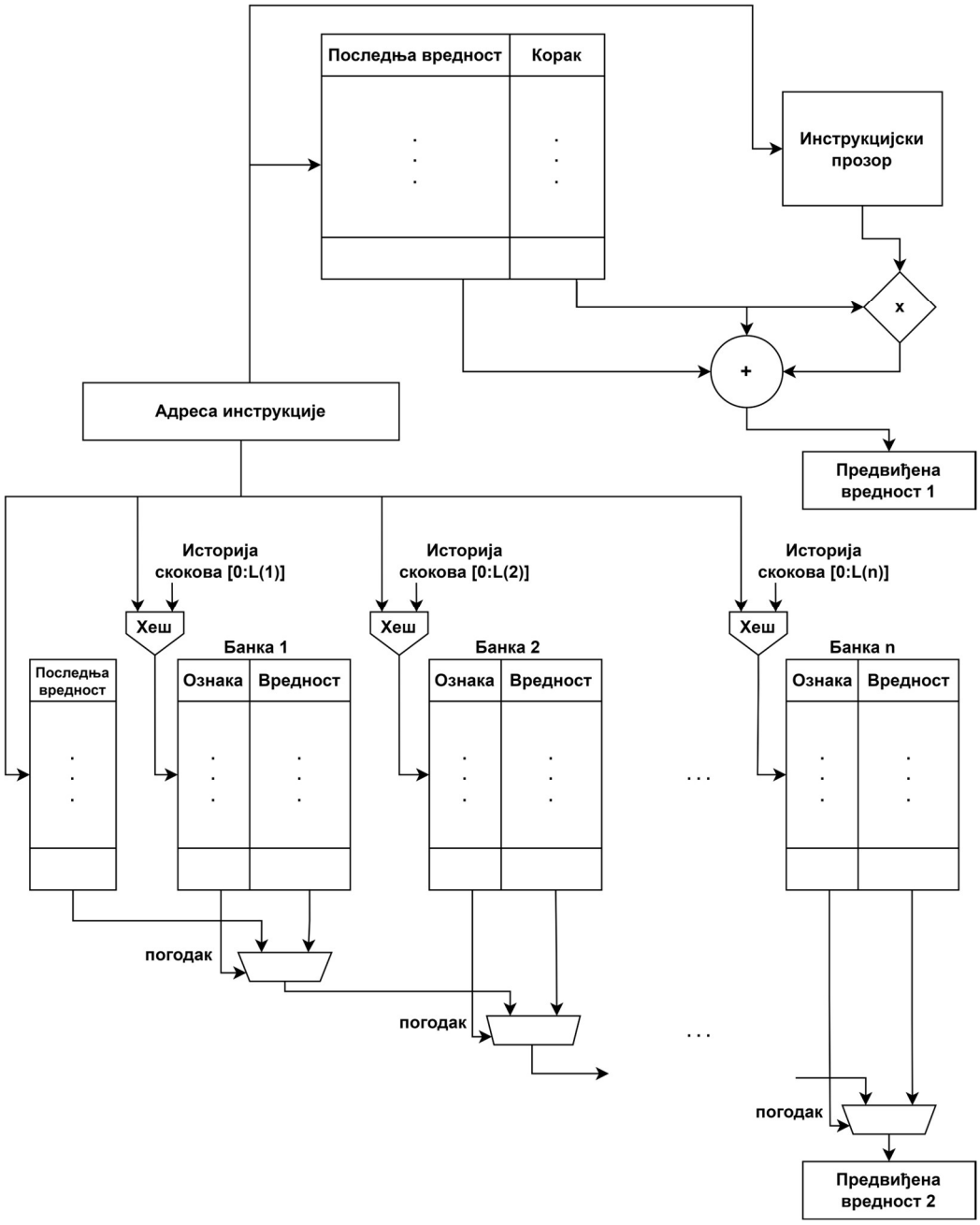
Предиктор *EVES* (*Enhanced VTAGE Enhanced Stride*) спада у групу хибридних предиктора који је представљен у раду [46]. Предиктор се састоји из два предиктора: *E-VTAGE* предиктор (*Enhanced VTAGE*) и *E-Stride* предиктор (*Enhanced Stride*). Основна идеја овог предиктора јесте да предвиђање врши само онда када процени да ће то предвиђање донети побољшање перформанси система.

E-VTAGE предиктор је побољшана верзија предиктора *VTAGE* (*Value TAGE*) [31] који је настао на основу предиктора скокова *TAGE* (*Tagged Geometric*) [53]. Аутори *EVES* предиктора вредности су такође и аутори *TAGE* предиктора који се сматра једним од најбољих предиктора скокова који су представљени научној заједници.

ii) Реализација

На слици 3.8.1. приказана је упрошћена шема овог предиктора. На горњој половини слике приказан је *E-Stride* предиктор. Овај предиктор у својој табели чува корак и последњу вредност. Поред тога чува и ознаку коју представљају виши бити адресе инструкције, али на упрошћеној шеми није приказана. Основна разлика између *Stride* предиктора и *E-Stride* предиктора јесте у формирању предвиђене вредности. Предвиђена вредност се формира као

збир последње вредности, корака и производа корака и броја инстанци посматране инструкције које се још увек извршавају. Информације о томе колико се инстанци посматране инструкције још увек извршавају, добијају се из инструкцијског прозора (енгл. *instruction window*).



Слика 3.8.1. *EVES* предиктор

Други део *EVES* предиктора *VTAGE* предиктор представљен је на другој половини слике 3.8.1. Овај предиктор има неколико табела (банки) које чувају у једном свом улазу ознаку и вредност. Табелама се приступа на основу хеш функције чији су улази адреса инструкције и неки битови регистра који чува историју скокова. Такође, део вредности хеш функције представља и ознаку. Свака банка приликом рачунања хеш функције узима различит број битова из регистра историје скокова. Уколико више банака има погодак

(ознака се поклапа са израчунатом хеш функцијом) за предвиђање се користи очитана вредност банке која је користила највише битова историје скокова. Уколико ни у једној банци нема поготка, тада се користи вредност очитана из посебне табеле којој се приступа на основу адресе инструкције. Та табела чува последње виђене вредности, па је стога то заправо *Last Value* предиктор.

Оба предиктора које чине *EVES* предиктор имају механизам поверења који се води за сваки улаз банке и табеле код *E-Stride* предиктора. Аутори предиктора су запазили да уколико се предвиђа резултат инструкције учитавања из меморије, која изазива промашај у кеш меморији, има значајно више доприноса побољшању перформанси него предвиђање резултата аритметичко-логичке инструкције [46]. Механизам поверења је реализован коришћењем пробабилистичких бројача [54], сваком улазу предиктора је додељен један бројач. Инкрементирање ових бројача се врши са различитим вероватноћама у зависности од типа инструкције као и од величине корака. Аутори су највећу вероватноћу инкрементирања доделили инструкцијама учитавања из меморије док су најмању вероватноћу додели аритметичко-логичким инструкцијама, што је у складу са њиховим закључком да побољшању перформанси највише доприноси предвиђање инструкција учитавања из меморије. Такође, аутори су приметили да на побољшање перформанси више утичу инструкције чији је корак већи од јединице, него инструкције чији је корак једнак јединици. На основу тога бројачи улаза са мањим корацима се инкрементирају са мањом вероватноћом.

Како се *EVES* предиктор састоји из два предиктора, уколико оба предиктора испоруче предвиђање, тада треба одабрати предвиђену вредност. У таквој ситуацију предвиђена вредност која се испоручује јесте вредност коју је испоручио *VTAGE* предиктор. Да ли ће неки предиктор испоручити предвиђену вредност о томе одлучује механизам поверења коју аутори истичу као један од главних делова овог предиктора.

iii) Евалуација

Као што је поменуто овај предиктор врши предвиђање свих типова инструкција, само што механизам поверења фаворизује предвиђања неких инструкција попут инструкција учитавања из меморије. Евалуација овог предиктора је урађена коришћењем радног оквира (енгл. *famework*) који је развијен за потребе светског такмичења у предвиђању вредности [21]. У раду [46] нису дате прецизности *EVES* предиктора, већ су представљени резултати добијени коришћењем поменутог радног оквира који резултате предвиђање представља кроз број извршених инструкција по циклусу (*IPC*). *EVES* предиктор је забележио од 25% до 38% већу вредност *IPC* у односу када радни оквир не користи предвиђање вредности. Распон од 25% до 38% зависи од величине предиктора, тако да побољшање од 25% одговора величини предиктора од 8KB, док је предиктор са неограниченом величином постигао побољшање од 38%.

Овде је још интересантно истаћи и покривеност предиктора *EVES*. На коришћеним тестовима за потребе светског такмичења, аутори предиктора су посматрали покривеност посебно за *E-Stride* компоненту и посебно за *VTAGE* компоненту. *E-Stride* предиктор је имао просечну покривеност 3.4%, док је за *VTAGE* забележена покривеност од 18.7%. Ниске покривености су последица строгог механизма поверења који одлучује да се предвиђање врши само за оне инструкције које означи да ће допринети значајно побољшању перформанси.

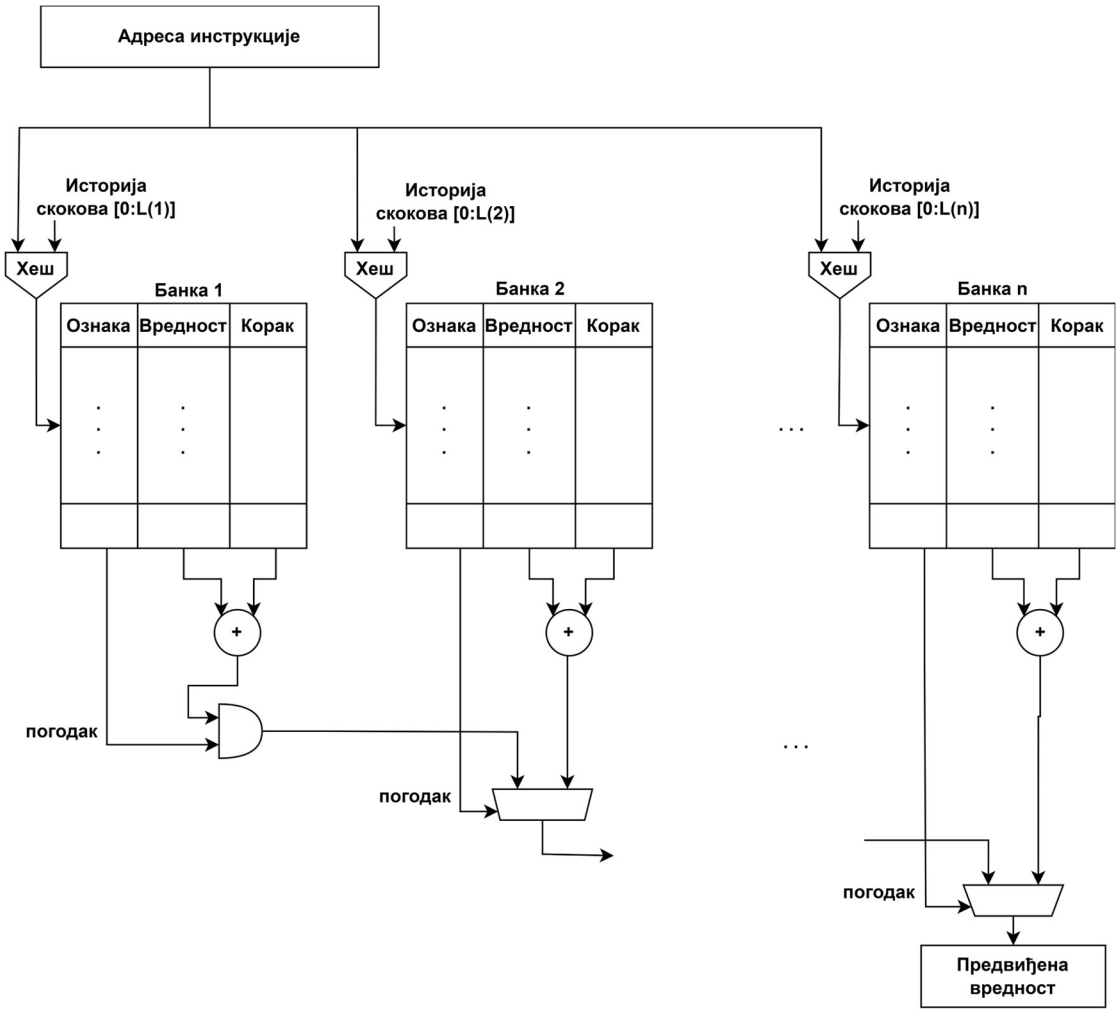
3.9. *CBC-VTAGE* предиктор

i) Резиме

Предиктор *CBC-VTAGE* (*Context-Base Computational VTAGE*) је представљен на светском такмичењу у предвиђању вредности и описан је у раду [47]. У самом називу овог предиктора се налази назив *VTAGE* предиктора који представља његову основу. *CBC-VTAGE* предиктор се разликује од *VTAGE* предиктора по томе што поред последње вредности још у банкама чува и корак. Предвиђена вредност се онда формира као збир последње вредности и корака.

ii) Реализација

На слици 3.9.1. приказана је шема *CBC-VTAGE* предиктора. Предиктор функционише на исти начин као *VTAGE* предиктор који је описан у поглављу 3.8. Разлика је као што је већ поменуто да банке садрже још и кораке, а да се вредност формира сабирањем вредности и корака. Такође, за разлику од *VTAGE* предиктора, овај предиктор нема у свом саставу *Last Value* предиктор.



Слика 3.9.1. *CBC-VTAGE* предиктор

Разлог зашто је овај предиктор проширена варијанта *VTAGE* предиктора кораком се крије у уоченом понашању током извршавања програма. У раду [47] је описано понашање које инструкције испољавају током извршавања. Уочено је да једна инструкција током

извршавања може упоредо формирати различите секвенце вредности (различите кораке) у зависности од путање којом се долази до ње (контекста). Уколико би се предиктору приступало само преко адресе инструкција тада би иста појава инструкције користила исти улаз предиктора, иако можда та инструкција тренутно генерише две различите секвенце вредности. Због свега наведеног улазу предиктора се приступа на основу излаза хеш функције чији су улази адреса инструкције и контекст, а предиктору је додат и корак уз последњу вредност. На тај начин се улазу приступа и на основу тренутног контекста. Ово омогућава да се предвиђање вредности врши за исту инструкцију користећи више улаза предиктора, тј. свака секвенца вредности коју генерише инструкција има свој улаз у предиктору према тренутном контексту.

Пример оваквог понашања где једна инструкција током извршавања упоредо формира различите секвенце вредности у раду [47] је представљен на примеру три функције. Две функције позивају трећу функцију, а унутар треће функције постоји инструкција која формира секвенцу вредности. Уколико се наизменично извршавају прве две функције, инструкција у трећој функцији ће формирати две секвенце вредности које се могу разликовати на основу тога да ли се та инструкција извршава на основу позива прве или друге функције.

Код овог предиктора треба посебно поменути унапређење везано за механизам поверења. Механизам поверења заснован на пробабилистичким бројачима преузет је од *EVES* предиктора. Примењена је оптимизација динамичке промене вероватноћа инкрементирања бројача на основу информација током самог извршавања. За разлику од статичког приступа где су вероватноће унапред постављене према анализи програма који ће се извршавати, динамички приступ подразумева да се у току извршавање вероватноће прилагођавају. Предиктору *CBC-VTAGE* је додата посебна табела која служи да прати прецизност предиктора за различите типове инструкција. Уколико прецизност падне испод неке границе (енгл. *threshold*) за неки тип инструкција, тада се вероватноће за инкрементирање пробабилистичких бројача приликом тачног предвиђања смањују. Стога бројачи спорије долазе до стања која дозвољавају да се инструкције спекулативно изврше и на тај начин предиктор проводи више времена обучавајући се за наредна предвиђања.

Такође, још један додатак овог предиктора је иновативан у односу на претходно описане предикторе. Та новина се огледа у постојању такозване црне листе (енгл. *blacklist filtering*). Посебна табела води евиденцију о прецизности предвиђања за појединачне инструкције. Табели се приступа користећи адресу инструкције и такође има ознаку (енгл. *tag*). Уколико прецизност за неку инструкцију падне испод неке дефинисане границе, тада предиктор престаје да користи предвиђање за ту инструкцију. На тај начин се такве инструкције не извршавају спекулативно док прецизност поново не достигне постављену границу.

Још једна оптимизација је примењена код овог предиктора, а односи се на смањење величине предиктора. Ради се о компресији последње виђених вредности које се чувају у оквиру предиктора. Примећено је да програми често користе вредности из неког скупа вредности где су највиши битови тих вредности једнаке. Пример таквих вредности јесу виртуелне адресе које програм генерише током извршавања. Како не би за сваку вредност чувао све битове, овај предиктор користи посебну табелу у којој чува вредности највиших битова. У оквиру самих банки чува најниже битове вредности и показивач до улаза у табели са највишим битовима. На овај начин је значајно смањена величина предиктора са малим губитком перформанси како је у раду [47] представљено.

iii) Евалуација

Евалуација *CBC-VTAGE* предиктора је урађена коришћењем радног оквира који је развијен за потребе светског такмичења у предвиђању вредности [21]. Убрзање на коришћеним тестовима у оквиру радног оквира јесте 17,1%, тј. за 17,1% је већи број извршених инструкцију по циклусу (*IPC*) за предиктор величине 8KB. Поред убрзања извршавања које је постигао овај предиктор, битно је истаћи као његов резултат и три новине које су примењене код њега: мењање вероватноћа ажурирања пробабилистичких бројача у току извршавања, увођење црне листе инструкција за које је тешко предвиђати тачне вредности и компресију вредности којом се смањује величина предиктора.

3.10. *H3VP* предиктор

i) Резиме

Предиктор *H3VP* (*History Based Highly Reliable Hybrid Value Predictor*) је такође представљен на светском такмичењу у предвиђању вредности, а описан је у раду [48]. Сам назив предиктора открива да се ради о хибридном предиктору. Њега чине три предиктора који врше предвиђање за три групе инструкција које су уочене. На основу групе којој припада инструкција одговарајући предиктор испоручује предвиђену вредност.

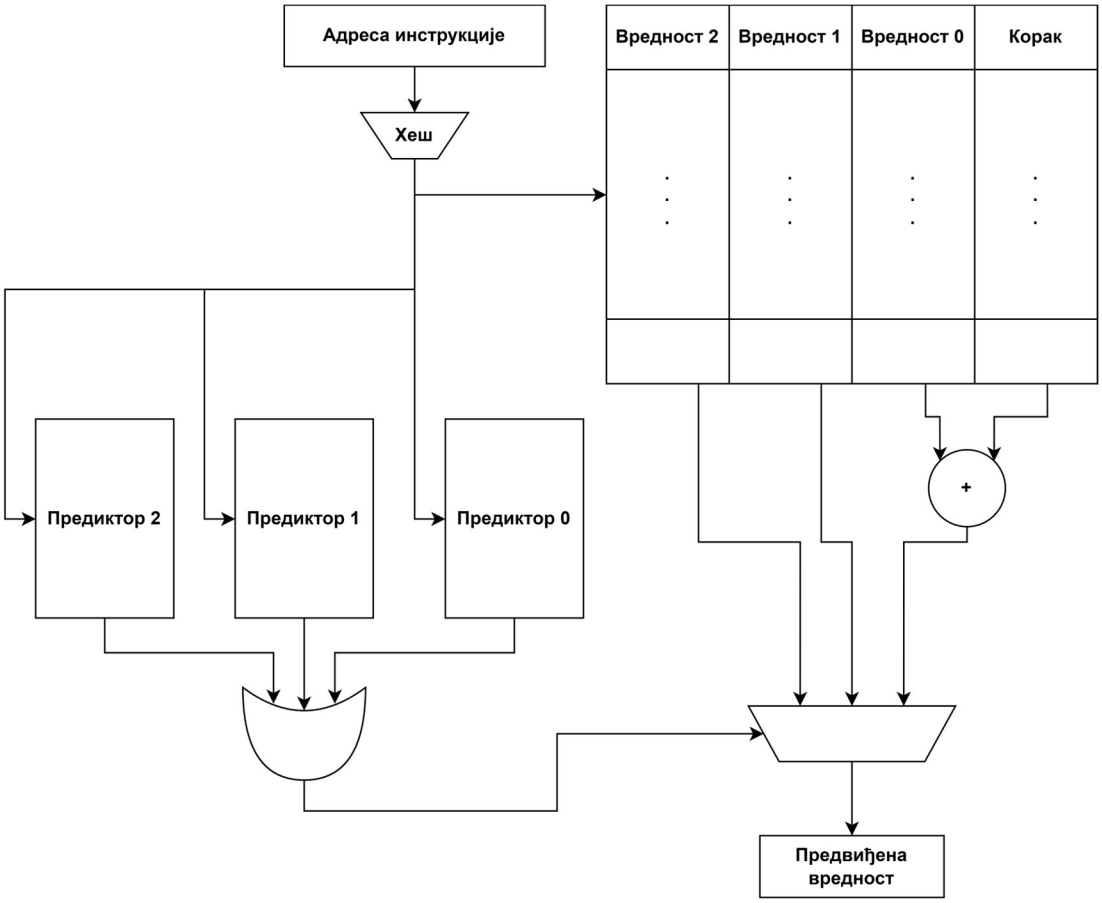
Аутори овог предиктора су инструкције поделили у три групе према томе на који начин се претходна виђена вредност неке инструкције касније опет користи за предвиђање. Према томе они су издвојили прву групу инструкција које производе секвенцу вредности где се на претходну вредност додаје неки корак и формира наредна вредност (енгл. *arithmetic instructions*). У ово групу спадају и константне вредности, тј. када инструкција производи увек исте вредности, корак је у том случају једнак нули. Другу и трећу групу инструкција представљају инструкције које понављају вредност коју су произвеле пре два извршавања или три извршавања, респективно (енгл. *two-periodic and three-periodic instructions*).

ii) Реализација

На слици 3.10.1. је приказана упрошћена шема *H3VP* предиктора. Као што је поменуто овај предиктор се састоји из три предиктора, за сваку групу инструкција по један предиктор. Овим предикторима се приступа на основу адресе инструкције. У оквиру улаза ових предиктора се чувају информације о тренутном стању за посматрану инструкцију. Аутори предиктора су формирали једну машину стања која заправо имплементира механизам поверења. Два најбитнија стања су стање које говори да се предиктор и даље обучава, а друго да се обучио и да се може користити предвиђена вредност за спекулативно извршавање.

Други део овог предиктора поред поменута три предиктора јесте табела која чува претходне виђене вредности као и корак. Овој табели се такође приступа на основу адресе инструкције. У оквиру једног улаза табеле чувају се три вредности, где вредности одговарају вредностима које ја та инструкција произвела пре једног извршавања (последње извршавање), пре два извршавања (претпоследње извршавање) као и пре три извршавања.

На основу стања три предиктора бира се једна вредност која ће бити испоручена као предвиђена вредност. Уколико је случај да предиктор који се односи на инструкције које формирају секвенцу вредности или константне вредности, тада се предвиђена вредност формира као збир последње вредности и корака. У ситуацији када предвиђање треба да испоруче предиктори који се односе на инструкције које понављају своје вредности након два или три извршавања тада се користе вредности 2 и 3 из табеле, респективно.



Слика 3.10.1. *H3VP* предиктор

iii) Евалуација

Овај предиктор је у оквиру радног оквира са светског такмичења у предвиђању вредности постигао за 11,56% већи број извршених инструкција по циклусу (*IPC*) него када се не користи предвиђање вредности. Такође, вредна помена је и анализа коју су аутори изнели, а тиче се понављања већ виђених вредности које производе инструкције. Око 50% инструкција генерише исту вредност као претходно виђену. Око 80% инструкција производи неку од последњих шеснаест виђених вредности, где највећи број инструкција производи исте вредности које су произвеле пре једног, два или три извршавања. Управо је та анализа и мотивисала ауторе да направе овај предиктор.

4. ПОСТАВКА ПРОБЛЕМА

У оквиру овог поглавља биће представљена главна идеја као и мотивација за израду ове докторске дисертације. Такође, биће описани проблеми који се дешавају приликом предвиђања вредности. У овом поглављу биће представљене могућности које предлаже ова докторска дисертација како би се ублажили уочени проблеми. Такође, у оквиру овог поглавља описане су инструкције, као и типови операнада који су погодни за предвиђање вредности у случајевима када је предвиђање непрецизно. На крају овог поглавља представљене су полазне хипотезе ове докторске дисертације.

4.1. Почетна запажања и мотивација

Термин прецизност се може дефинисати као однос између броја тачних предвиђања и укупног броја предвиђања. Постојећи предиктори вредности како би постигли велику прецизност примењују строге механизме поверења. На тај начин предиктори селективно врше предвиђање само за оне инструкције које механизам поверења означи безбедним. То да је инструкција безбедна се односи на процену да ли ће предвиђање бити исправно. Стога механизми поверења прате извршавање и претходна предвиђања за инструкције и на основу тога одређују да ли ће се предвиђање користити или не.

Још једна ствар коју неки механизми поверења имплементирају јесте процена добитка уколико се користи предвиђање у поређењу са губицима перформанси процесора уколико је предвиђање лоше. Када се говори о губицима перформанси пре свега се мисли на утрошено време које је неопходно да би се одрадио опоравак услед лошег предвиђања.

Осим што неки предиктори користе строге механизме поверења, неки предиктори су специјализовани за одређене типове инструкција. Препознато је да су неки типови инструкција погоднији за предвиђање вредности, што је описано у другом поглављу ове дисертације. На тај начин се смањује покривеност јер се не ради предвиђање вредности за све инструкције, већ само за неке.

За постојеће предикторе важи да смањују покривеност како би одржали прецизност на високом нивоу. На тај начин се постојећи предиктори вредности штите од честих промашаја. Даље, самим тим ређе се дешавају опоравци услед лошег предвиђања и избегава се трошење времена за потребе опоравка.

Идеја ове докторске дисертације јесте да прикаже могућност да се настави извршавање инструкције иако предвиђање није било исправно, односно да се извршавање настави исправно са непрецизно предвиђеном вредношћу. На тај начин би се у неким ситуацијама такође избегао опоравак услед лошег предвиђања. Поред тога, предиктори у таквим ситуацијама не би морали да предвиђају комплетно тачну вредност јер ће и неке непрецизне вредности омогућити да се извршавање исправно настави.

Као што је поменуто у другом поглављу предвиђање се врши на нивоу асемблерских инструкција. Да би извршавање могло да се коректно настави и са непрецизно предвиђеним вредностима, инструкције које су погодне за предвиђање вредности се могу описати на

следећи начин. Инструкција има два операнда и њен резултат остаје непромењен за више различитих вредности једног од њених операнада. То значи да постоји скуп вредности једног операнда који са вредношћу другог операнда увек производи идентичан резултат инструкције. Такве инструкције се могу описати формулама 4.1.1. и 4.1.2.

$$R\_A = O\_1 \bullet O\_2 \tag{4.1.1.}$$

$$R\_B = O\_1 \bullet O\_2' \tag{4.1.2.}$$

Резултат инструкције је означен са R_A и R_B, док су први и други операнд означени са O_1 и O_2, респективно. Симболом • је представљена нека операција која се спроводи над операндима. Операнд O_2 је операнд за кога се врши предвиђање вредности. У формули 4.1.1. други операнд O_2 има тачну вредност, док у формули 4.1.2. исти тај операнд има предвиђену вредност O_2'. Предвиђена вредност другог операнда O_2' може да се разликује од његове тачне вредности O_2. Уколико се предвиђена вредност операнда и његова тачна вредност разликују, за даље извршавање инструкције је битно само да је резултат спекулативне извршене инструкције (формула 4.1.2.) идентичан резултату инструкције када се она изврши са тачним вредностима операнада (формула 4.1.1.), што је представљено формулом 4.1.3.

$$R\_A == R\_B, \text{ иако је } O\_2 != O\_2' \tag{4.1.3.}$$

4.2. Инструкције погодне за предвиђање

Инструкције које су описане у претходном потпоглављу могу се поделити у две групе према томе шта је резултат њиховог извршавања. Прву групу чине инструкције чији је резултат извршавања вредност дестинационог операнда и постављање индикатора (енгл. *flags*) у програмској статусној речи (енгл. *program status word*). Док другој групи инструкција припадају оне инструкције чији резултат јесте само постављање индикатора у програмској статусној речи без постављања вредности дестинационом операнду.

Прва група инструкција се може описати формулом 4.2.1. У овој формули ознаком D је представљен дестинациони операнд инструкције. Он добија вредност која се формира на основу операције • која се спроводи над операндима O_1 и O_2. Формирана вредност која се уписује у дестинациони операнд представља први део резултата инструкције. Други део резултата инструкције јесте постављање индикатора у програмској статусној речи. У формули 4.2.1. део резултата инструкције који се односи на постављање индикатора је означен са [*flags*].

$$([D = O\_1 \bullet O\_2] \;\&\&\; [flags]) \tag{4.2.1.}$$

Друга група инструкција се може представити формулом 4.2.2. У овој формули недостаје ознака D која постоји у формули 4.2.1. Разлог зашто не постоји ова ознака јесте што ове инструкције не остављају вредност у дестинационом операнду, тј. оне немају дестинациони операнд. На основу вредности која се формира операцијом • над операндима O_1 и O_2 постављају се индикатори у програмској статусној речи, што представља једини резултат ових инструкција.

$$([O\_1 \bullet O\_2] \;\&\&\; [flags]) \tag{4.2.2.}$$

Може се закључити да описане инструкције као резултат испоручују или дестинациони операнд и промењене индикаторе (прва група) или само промењене индикаторе (друга група). Наредне инструкције, тј. инструкције које следе након описаних

инструкција користе њихове резултате. Један начин је да користе дестинационе операнде описаних инструкција као своје изворишне операнде, што је случај ако резултат производе инструкције из прве групе. Други начин како наредне инструкције могу користити резултат јесте да користе постављане индикаторе у току свог извршавања, што је случај ако резултат производе инструкције и из прве и из друге групе. Такве наредне инструкције су условне инструкције и начин на који ће се извршити зависи од индикатора.

Овде треба напоменути да условне инструкције у неким ситуацијама неће користити све индикаторе који су постављени на основу резултата претходне инструкције. У таквом сценарију је довољно само да буду тачно постављени само индикатори које ће користити условна инструкција. Такође, у неким ситуацијама, условне инструкције услов према коме се извршавају израчунавају као функцију више индикатора. У таквим ситуацијама за исправно извршавање је једино битно да услов који се израчунава има тачну вредност (вредност коју би имао услов када би се израчунавао са тачни вредностима индикатора).

Инструкција која користи резултат неке друге инструкције може на више начина да га искористи. Један од начина подразумева да се користи и вредност дестинационог регистра као и сви постављени индикатори. Док је други начин да се користе само постављени индикатори или само неки од постављених индикатора. Дакле, постоје ситуације у којима ће се користити само неки део резултата инструкције. У оквиру ове докторске дисертације део резултата инструкције који користи нека друга инструкција је назван користан резултат.

У оквиру прве групе инструкција у оквиру ове докторске дисертације посматране су инструкције које спроводе операције логичког *и* (енгл. *and*) и логичког *или* (енгл. *or*). Док су у оквиру друге групе инструкција посматране инструкције које спроводе операције логичког тестирања (енгл. *test*) и упоређивање два операнда (енгл. *compare*). У наставку ће бити описане поменуте инструкције са становишта погодности за спекулативно извршавање са непрецизно предвиђеним операндима.

4.2.1. Инструкција логичког и

Инструкција логичког *и* спада у прву групу инструкција (формула 4.2.1.) чији је резултат вредност која се уписује у дестинациони операнд као и постављање индикатора. Према томе ова инструкција се може описати формулом 4.2.1.1. Инструкција има два изворишна операнда (O_1 и O_2) над којима се спроводи операција логичког *и.* Резултат операције се смешта у дестинациони операнд означен са D што представља први део резултата. Други део резултата ове инструкције представља постављање индикатора у програмској статусној речи на основу добијене вредности операнда D. У зависности од специфичне архитектуре процесора неки индикатори ће добити нове вредности, што представља други део резултата ове инструкције [*flags*].

$$([D = O\_1 \ and \ O\_2] \ \&\& \ [flags]) \qquad (4.2.1.1.)$$

Може се посматрати инструкција логичког *и* где је један њен операнд познат, а други није у тренутку када треба да се примени операција над операндима. Познатим операндом се може сматрати операнд чија се вредност налази у неком од регистара или је непосредна вредност која је укодована у саму инструкцију. Други операнд јесте операнд чија вредност није одмах доступна, може се претпоставити да његова вредност долази из меморије. Како приступ меморији захтева више процесорских циклуса (у ситуацији када постоји промашај у кеш меморији), вредност таквог операнда се може предвидети. Особине операнада који су погодни за предвиђање описане су у наредном потпоглављу 4.3. ове докторске дисертације.

У оквиру програмског исечка кода 4.2.1.1. представљене су две инструкције логичког *и.* Оне имају један дестинациони операнд означен са *dst* и два изворишна операнда. Претпоставка је да су операнди ширине четири бита. Вредност првог изворишног операнда је одмах доступна јер је специфициран непосредним адресирање (вредности 0001b и 1110b). Други операнд је означен са *opr2* и претпоставка је да се његова вредност налази у меморији, специфициран је неким меморијским адресирањем што га чини погодним за предвиђање.

```
AND dst, #0001b, opr2
AND dst, #1110b, opr2
```

Исечак програмског кода 4.2.1.1. Пример инструкције логичког *и*

Уколико се посматра прва инструкција са програмског исечка 4.2.1.1., вредност другог операнда може на позицији три виша бита имати било које вредности. Резултат операције *и* ће зависити само од вредности најнижег бита јер вредност бита познатог операнда (0001b) на тој позицији има вредност један. Према овоме чак осам различитих вредности другог операнда могу довести до тачног резултата инструкције.

Уколико се посматра друга инструкција са програмског исечка 4.2.1.1., вредност другог операнда може на позицији најнижег бита имати било коју вредност. Резултат операције *и* ће зависити од вредности три највиша бита јер је вредност тих битова познатог операнда један (1110b). Према овоме две различите вредности другог операнда могу довести до тачног резултата инструкције.

На основу разматрања програмског исечка 4.2.1.1. може се закључити да више вредности операнда чија се вредност предвиђа могу довести до тачног резултата инструкције. То је због саме природе инструкције логичког *и.* Уколико један операнд има вредност нула на неким позицијама, онда ће и резултат имати на тим позицијама вредност нула без обзира које вредности имају бити на тим позицијама другог операнда. Ово оставља простора да и неке непрецизно предвиђене вредности могу довести до тачног резултата инструкције логичког *и.*

4.2.2. Инструкција логичког или

Инструкција логичког *или* спада у прву групу инструкција (формула 4.2.1.) чији је резултат вредност која се уписује у дестинациони операнд као и постављање индикатора. Према томе ова инструкција се може описати формулом 4.2.2.1. Инструкција има два изворишна операнда (O_1 и O_2) над којима се спроводи операција логичког *или*. Резултат операције се смешта у дестинациони операнд означен са D што представља први део резултата. Други део резултата ове инструкције представља постављање индикатора у програмској статусној речи на основу добијене вредности операнда D. У зависности од специфичне архитектуре процесора неки индикатори ће добити нове вредности, што представља други део резултата ове инструкције.

$$([D = O\_1 \; or \; O\_2] \;\&\&\; [flags]) \qquad (4.2.2.1.)$$

Слично као код инструкције логичког *и*, инструкција логичког *или* може се посматрати као инструкција чије је један операнд познат, а други није у тренутку када треба да се примени операција над операндима. Такође, као код инструкције логичког *и,* овде се може применити предвиђање вредности за операнд чија вредност није доступна.

У оквиру програмског исечка кода 4.2.2.1. представљене су две инструкције логичког *или.* Оне имају један дестинациони операнд означен са *dst* и два изворишна операнда. Претпоставка је да су операнди ширине четири бита. Вредност првог изворишног операнда је одмах доступна јер је специфициран непосредним адресирање (вредности 0001b и 1110b), слично као у примеру за инструкцију логичког *и*. Други операнд је означен са *opr2* и претпоставка је да се његова вредност налази у меморији, специфициран је неким меморијским адресирањем што га чини погодним за предвиђање.

```
OR dst, #0001b, opr2
OR dst, #1110b, opr2
```

Исечак програмског кода 4.2.2.1. Пример инструкције логичког *или*

Уколико се посматра прва инструкција са програмског исечка 4.2.2.1., вредност другог операнда може на позицији најнижег бита имати било коју вредност. Резултат операције *или* ће зависити вредности највиша три бита јер вредност битова познатог операнда (0001b) на тим позицијама имају вредност нула. Према овоме две различите вредности другог операнда могу довести до тачног резултата инструкције.

Уколико се посматра друга инструкција са програмског исечка 4.2.2.1., вредност другог операнда може на позицији три виша бита имати било које вредности. Резултат операције *или* ће зависити само од вредности најнижег бита јер вредност бита познатог операнда (1110b) на тој позицији има вредност нула. Према овоме чак осам различитих вредности другог операнда могу довести до тачног резултата инструкције.

На основу разматрања програмског исечка 4.2.2.1. може се закључити, слично као и за инструкцију логичког *и*, да више вредности операнда чија се вредност предвиђа могу довести до тачног резултата инструкције. То је због саме природе инструкције логичког *или.* Уколико један операнд има вредност један на неким позицијама, онда ће и резултат имати на тим позицијама вредност један без обзира које вредности имају бити на тим позицијама другог операнда. Ово оставља простора да и неке непрецизно предвиђене вредности могу довести до тачног резултата инструкције логичког *или.*

4.2.3. Инструкција логичког тестирања

Инструкција логичког тестирања спада у другу групу инструкција (формула 4.2.2.) чији резултат представља само постављање индикатора у програмској статусној речи. Према томе ова инструкција се може описати формулом 4.2.3.1. Инструкција има два операнда (O_1 и O_2) над којима се спроводи операција логичког *и*, али за разлику од инструкције логичког *и* нема дестинациони операнд. На основу резултата операције *и* само се постављају индикатори у програмској статусној речи, што представља једини резултат инструкције логичког тестирања.

$$([O\_1 \; and \; O\_2] \; \&\& \; [flags]) \qquad (4.2.3.1.)$$

Слично као код инструкције логичког *и*, инструкција логичког тестирања може се посматрати као инструкција чије је један операнд познат, а други није у тренутку када треба да се примени операција над операндима. Такође, као код инструкције логичког *и,* овде се може применити предвиђање вредности за операнд чија вредност није доступна.

Може се посматрати програмски исечак 4.2.3.1. који садржи две инструкције логичког тестирања. Ове инструкције су сличне оним из програмског исечка 4.2.1.1. који садржи

инструкције логичког *и*, само што нема дестинационог операнда. Над операндима инструкције логичког тестирања примењује се операција логичког *и*. Према томе све ствари везане за предвиђање непознатог операнда инструкције логичког *и* важе и за предвиђање непознатог операнда инструкције логичког тестирања.

```
TEST #0001b, opr2
TEST #1110b, opr2
```

Исечак програмског кода 4.2.3.1. Пример инструкције логичког тестирања

На основу разматрања програмског исечка 4.2.3.1. може се закључити, слично као и за инструкцију логичког *и*, да више вредности операнда чија се вредност предвиђа могу довести до тачног резултата инструкције. Дакле, инструкција логичког тестирања такође може бити погодна за предвиђање операнада јер и са непрецизно предвиђеним операндом може се добити тачан резултат извршавања.

4.2.4. Инструкција упоређивања вредности

Инструкција упоређивања вредности спада у другу групу инструкција (формула 4.2.2.) чији резултат представља само постављање индикатора у програмској статусној речи. Према томе ова инструкција се може описати формулом 4.2.4.1. Инструкција има два операнда (O_1 и O_2) над којима се спроводи операција одузимања. Како инструкција нема дестинациони операнд, на основу резултата операције одузимања само се постављају индикатори у програмској статусној речи, што представља једини резултат инструкције упоређивања вредности.

$$([O\_1 - O\_2] \;\&\&\; [flags]) \qquad (4.2.4.1.)$$

Као и код претходно описаних инструкција логичког *и*, *или* и тестирања и код инструкције упоређивање се може применити предвиђање вредности операнда. Уколико вредност једног од операнада није одмах доступна, може се применити предвиђање вредности за тај операнд.

```
CMP #0001b, opr2
CMP #1110b, opr2
```

Исечак програмског кода 4.2.4.1. Пример инструкције упоређивања вредности

Може се посматрати програмски исечак 4.2.4.1. који садржи две инструкције упоређивања вредности. Ове инструкције имају два изворишна операнда. Вредност првог изворишног операнда је одмах доступна, јер је специфициран непосредним адресирање (вредности 0001b и 1110b), слично као у примеру за претходно описане инструкције. Други операнд је означен са *opr2* и претпоставка је да се његова вредност налази у меморији, специфициран је неким меморијским адресирањем што га чини погодним за предвиђање.

Уколико се посматра прва инструкција, у току њеног извршавања ће се применити одузимање вредности другог операнда (*opr2*) од вредности првог операнда (0001b). Може се претпоставити да је други операнд већи од вредности првог. У таквом сценарију било која вредност која је већа од вредности првог операнда, довешће до тачног резултата ове инструкције. Једини резултат ове инструкције јесте постављање индикатора у програмској статусној речи. Дакле, уколико се предвиђа вредност другог операнда, довољно је да у описаном сценарију то буде било која вредност која је већа од 0001b. Само две вредности

неће довести до тачног резултата, а то су вредности 0000b и 0001b које нису веће од вредности првог операнда (0001b).

Може се посматрати друга инструкција и претпоставити да је вредност другог операнда мања од вредности првог операнда (1110b). У овом сценарију само две вредности за предвиђену вредност другог операнда неће довести до тачног резултата инструкције. То су вредности 1111b и 1110b које нису мање од вредности првог операнда (1110b).

На основу разматрања програмског исечка 4.2.4.1. може се закључити, слично као и за претходно описане инструкције, да више вредности операнда чија се вредност предвиђа могу довести до тачног резултата инструкције. Дакле, инструкција упоређивања такође може бити погодна за предвиђање операнада јер и са непрецизно предвиђеним операндом може се добити тачан резултат извршавања.

4.3. Операнди погодни за предвиђање

Понашање инструкције, које је описано формулама 4.1.1., 4.1.2. и 4.1.3., може се искористити у ситуацији када је потребно предвидети вредност једног операнда. Операнд који се сматра познатим операндом је операнд чија је вредност одмах доступна процесору у тренутку извршавања када треба да се одради операција над операндима. Такав операнд је у инструкцији одређен или непосредним адресирањем или регистарским директним адресирање. Уколико је одређен непосредним адресирањем то значи да је сама вредност операнда укодована у запис инструкције која се извршава. Други случај, када је операнд специфициран регистарским директним адресирањем значи да се вредност операнда налази у неком од регистара. Тада је вредност операнда такође одмах доступна процесору. У оваквим ситуацијама нема смисла користити предвиђање вредности јер је вредност одмах доступна процесору.

Операнди за које има смисла предвиђати вредности су они операнди који могу изазвати заустављање извршавања зато што њихове вредности нису доступне у тренутку када треба да се инструкција изврши. Такви операнди потичу из меморије. Уколико се податак не налази у кеш меморији (промашај у кешу), тада процесор мора приступити оперативној меморији. Тај приступ траје неко време и док се не заврши није могуће да инструкција настави своје извршавање.

Уколико се посматра процесор са проточном обрадом, заустављање (енгл. *pipeline stall*) [55] значи да ће се инструкција са таквим операндом зауставити у степену проточне обраде у којем се тренутно налази. На тај начин она неће моћи да пређе у наредни степен све док вредност операнда не постане доступна. Тек када вредност операнда постане доступна, процесор може да настави са извршавањем.

Може се замислити пример у којем су инструкција учитавања из меморије (прва) и након ње инструкција која користи њен резултат (друга). Прва инструкција дохвата из меморије вредност коју ће користити друга инструкција као свој операнд. Тада уколико има промашаја у кеш меморији, друга инструкција мора да се заустави док резултат прве инструкције не постане доступан, тј. док прва инструкција не заврши приступ оперативној меморији. Процесор може да убаци одређен број инструкција без дејства (енгл. *NOP – No Operation*) између ове две инструкције. Ове инструкције само пролазе кроз фазе проточне обраде без остављања било каквих резултата, тј. не мењају ни регистре ни меморију. На тај начин инструкција која изазива заустављање ће касније доћи на извршавање. У тренутку

када дође на извршавање вредност операнда ће јој бити доступна, али ће процесор извршавати инструкције без дејства која ће потрошити неко време као и енергију [56].

На основу претходног разматрања уочене су две групе операнада који су погодни за предвиђање. Заједничко за обе групе јесте то што операнди потичу из меморије. Разлика између операнада из ове две групе јесте тренутак када је покренуто дохватање вредности из меморије. На основу тога у наставку следе описи уочених група операнада:

- У прву групу операнада спадају операнди који су специфицирани неким меморијским адресирањем у оквиру саме инструкције. У фази проточне обраде у којој се врши дохватање операнада, приступиће се меморији како би се дохватила вредност операнда. Према меморијској хијерархији прво ће се приступити кеш меморији, а након тога и главној меморији за случај да тражени податак није у кеш меморији. У таквој ситуацији када се приступа главној меморији, може се применити предвиђање вредности. На тај начин процесор може одмах наставити са извршавањем инструкције, само што се мора означити да се инструкција извршава спекулативно јер користи предвиђену вредност операнда. За овај тип операнда увешће се ознака *T_MEM* која ће се користити у наставку ове докторске дисертације.
- Другој групи операнада припадају операнди који такође потиче из меморије, али се у току извршавања неке инструкције они налазе у регистру. То значи да та инструкција специфицира операнд регистарским директним адресирањем и његову вредност користи из регистра. Међутим, могуће је да је нека претходна инструкција одговорна за дохватање те вредности из меморије. Како је извршавање инструкција код процесора са проточном обрадом преклопљено и ако је инструкција која користи ту вредност близу инструкције која дохвата ту вредност, тада је могуће да вредност није дохваћена из меморије на време (промашај у кеш меморији). То значи да ће инструкција која користи ту вредност морати да се заустави док инструкција која дохвата вредност из меморије не заврши приступ меморији. Како би се избегло заустављање извршавања, може се применити предвиђање вредности за овакве операнде чиме се наставља спекулативно извршавање. За овај тип операнда увешће се ознака *R_MEM* која ће се користити у наставку ове докторске дисертације.

У наставку ове докторске дисертације операнди из ове две групе операнада ће се називати меморијским операндима и у наставку ће се користити ознака *ANY_MEM* за ове операнде. Истраживање у оквиру ове докторске дисертације се фокусирало на приказивање феномена да је могуће добити тачан резултат инструкције и са непрецизно предвиђеним операндима. Идеја је да се прикаже максималан потенцијал спекулативног извршавања са непрецизно предвиђеним операндима. Због тога су сви меморијски операнди описаних погодних инструкција тумачени као да изазивају промашај у кешу, тј. сви меморијски операнди су означени као погодни за предвиђање.

4.4. Полазне хипотезе

У оквиру ове докторске дисертације биће приказан феномен спекулативног извршавања инструкција са непрецизно предвиђеним вредностима операнада. Биће представљени развијени аналитички модели извршавања инструкција са непрецизно предвиђеним вредностима операнада. Такође, кроз симулације над стандардизованим

тестовима за процесоре биће одрађена евалуација развијених модела. Полазне хипотезе ове докторске дисертације су дате у наставку:

- Могуће је за погодне типове инструкција добијати исправан резултат иако је непрецизно предвиђен операнд – хипотеза 1;
- Постоје ситуације у којима је прецизност тачног резултата инструкције на основу предвиђеног операнда већа од прецизности тачно предвиђеног операнда – хипотеза 2;
- Могуће је за погодне типове инструкција добијати исправан користан резултат (део резултата инструкције који ће нека наредна да користи) иако је непрецизно предвиђен операнд – хипотеза 3;
- Могуће је користити постојеће предикторе вредности независно од типа погодних инструкција – хипотеза 4;
- Могуће је користити постојеће предикторе вредности независно од тога колико би инструкција, за коју се предвиђа операнд, чекала операнд из меморије у случају да се не користи предвиђање – хипотеза 5;.
- Постоје ситуације у којима извршавање са непрецизно предвиђеним операндима може постизати боље време извршавања од извршавања само са тачно предвиђеним операндима на основу постојећих предиктора вредности – хипотеза 6.

5. АНАЛИТИЧКИ МОДЕЛИ ИЗВРШАВАЊА

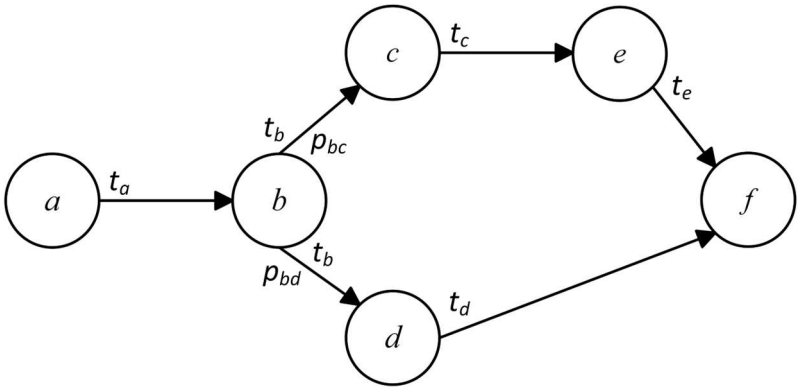
У оквиру ове докторске дисертације описано је спекулативно извршавање инструкција са непрецизно предвиђеним операндима. Идеја је да се предиктори вредности користе за предвиђање вредности операнада погодних инструкција. Такве инструкције могу произвести тачан резултат и са непрецизно предвиђеним операндима избегавајући губљење времена на опоравак услед лошег предвиђања. Као резултат ове докторске дисертације развијена су два аналитичка модела спекулативног извршавања. У оквиру овог поглавља биће представљени аналитички модели спекулативног извршавања инструкција. Један модел описује спекулативно извршавање са тачно предвиђеним вредностима операнада. Други модел описује спекулативно извршавање које укључује и тачно и непрецизно предвиђене вредности операнада уколико се са њима добија тачан резултат инструкције.

5.1. Аналитички модели

Како би се боље разумео неки нови концепт који се уводи у архитектуру рачунара некада је лакше такав концепт аналитички моделовати. На тај начин је могуће лакше и брже сагледати основне идеје тог концепта без разматрања неке специфичне архитектуре процесора. Такође, знатно је брже и ефикасније представити нови концепт аналитичким моделом него имплементирањем таквог концепта у оквиру неког комплексног и робусног симулатора где је потребно познавати детаље симулатора и микроархитектуру коју имплементира симулатор. Још једна предност аналитичког модела се огледа у томе да се приликом развијања таквих модела пажња може сконцентрисати само на најбитније аспекте концепта који се описује. Наравно, аналитичко моделовање не искључује и употребу симулатора за представљање нових концепата, штавише аналитички модел доприноси да се лакше разуме неки концепт који може бити имплементиран и у оквиру неког симулатора [57]. Поред тога аналитичко моделовање је добар избор и приликом развијања нове микроархитектуре процесора за специфичне намене јер се на тај начин брзо долази до процена перформанси које може постићи моделовани процесор [58].

У оквиру истраживања чији је резултат ова докторска дисертација, одлучено је да се извршавање са непрецизним вредностима као феномен представи аналитичким моделима. У сврху тога развијена су три модела која су представљена у научном раду чији је коаутор аутор ове докторске дисертације [19]. Један модел представља само извршавање инструкција без спекулативног извршавања, што значи да не укључује предвиђање вредности. Друга два представљају спекулативно извршавање предвиђањем вредности операнада. Разлика између та два модела се огледа у томе што један прихвата само тачно предвиђене вредности операнада, а други и непрецизно предвиђене вредности операнада ако је резултат инструкције исправан.

У циљу бољег разумевања поменутих модела, прво ће бити представљен један мањи општи модел. На том моделу биће објашњене све ознаке, а потом и дефинисане формуле којима се описује време извршавања инструкција. На слици 5.1.1. је приказан општи аналитички модел који је основа за развијене моделе у оквиру ове докторске дисертације.



Слика 5.1.1. Општи модел извршавања

На слици 5.1.1. је приказан граф извршавања инструкција. Чворови графа представљају инструкције. Чворови су повезани усмереним везама које представљају путању којом тече извршавање. Граф садржи укупно шест инструкција коју су означене са *a*, *b*, *c*, *d*, *e* и *f*. На посматраном примеру извршавање креће од инструкције *a* и завршава се у чвору који одговара инструкцији *f*.

Усмерене везе које постоје на графу као што је поменуто својим смером означавају путању извршавања. На посматраном графу на слици 5.1.1. до последњег чвора је могуће доћи преко две путање. Прва путања је *a-b-c-e-f*, док је друга путања преко чвора *d* и изгледа *a-b-d-f*.

Усмерене везе још имају и придодате ознаке. Ознаке које се налазе изнад везе означавају време које је потребно да се изврши инструкција од које креће веза. Уколико се посматра путања *a-b* потребно је t_a времена да се изврши инструкција *a* и дође до чвора *b*, тј. до инструкције *b*. Поред ознаке времена, усмерене везе имају и ознаку испод везе која представља вероватноћу да се том везом настави извршавање, у случају када је из једног чвора могуће наставити извршавање двема путањама. На приказаном графу је то случај са чвором *b* из ког је могуће наставити извршавање или ка чвору *c* са вероватноћом p_{bc} или ка чвору *d* са вероватноћом p_{bd}. Уколико је из једног чвора могуће отићи само у један наредни чвор, тада је на графу изостављена ознака за вероватноћу јер у таквом случају она износи један.

За овакав модел извршавања инструкција може се дефинисати време које је потребно да се изврше инструкције. На графу са слике 5.1.1. могу се дефинисати два времена која представљају време да се изврше инструкције почевши од инструкције *a* па до чвора *f*. Овде треба напоменути да се не урачунава време за извршавање инструкције *f*, већ само време које протекне док се не дође до чвора *f*. Формуле 5.1.1. и 5.1.2. представљају времена да се дође до чвора f преко чворова *c* и *d*, респективно.

$$t_{bc} = t_a + t_b + t_c + t_e \tag{5.1.1.}$$

$$t_{bd} = t_a + t_b + t_d \tag{5.1.2.}$$

Како време извршавања зависи од вероватноћа које одређују којом путањом се наставља из чвора *b*, могуће је дефинисати математичко очекивање које представља очекивано време извршавања. Како ово извршавање има два могућа исхода (путања преко чвора *c* и преко чвора *d*), могуће одредити очекивано време извршавања као збир сабирака, где је један сабирак време помножено са вероватноћом да се том путањом одвија

извршавање. Формула 5.1.3. представља математичко очекивање за приказани граф на слици 5.1.1.

$$E(t) = \sum pt = p_{bc}t_{bc} + p_{bd}t_{bd} \tag{5.1.3}$$

Даље се формула 5.1.3. може развити према чињеници да збир вероватноћа веза које иду из истог чвора је једнак јединици што је представљено формулом 5.1.4. На основу те формуле може се формирати коначан израз за математичко очекивање из формуле 5.1.3., што је дато формулом 5.1.5.

$$1 = p_{bc} + p_{bd} \tag{5.1.4.}$$

$$E(t) = p_{bc}(t_a + t_b + t_c + t_e) + p_{bd}(t_a + t_b + t_d)$$

$$E(t) = (p_{bc} + p_{bd})(t_a + t_b) + p_{bc}(t_c + t_e) + p_{bd}t_d$$

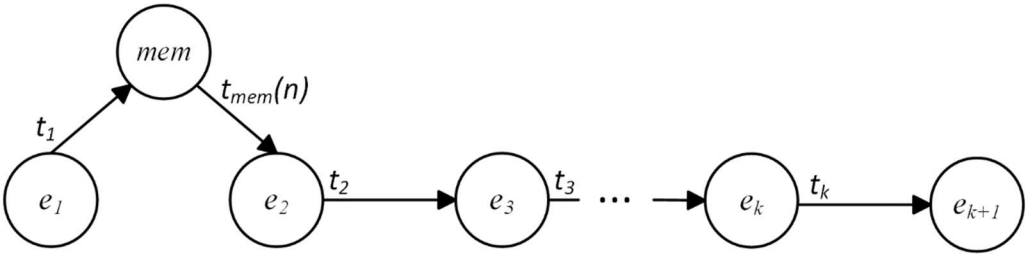
$$E(t) = t_a + t_b + p_{bc}(t_c + t_e) + p_{bd}t_d \tag{5.1.5.}$$

У формули 5.1.5. је представљено математичко очекивање које описује време које је потребно да се изврше инструкције почевши од чвора *a* до чвора *f*. Гледајући граф са слике 5.1.1. може се видети да се током извршавања мора прећи преко чворова *a* и *b*, што је у складу са добијеним изразом у формули 5.1.5. Времена која одговарају везама које иду из тих чворова нису помножена никаквом вероватноћом, док времена која одговарају везама из чворова *c* и *e*, као и чвора *d* јесу помножена одговарајућим вероватноћама p_{bc} и p_{bd}, респективно. Од ове две вероватноће зависиће очекивана вредност времена извршавања.

5.2. Извршавање без предвиђања операнада

Први аналитички модел који ће бити представљен описује извршавање без предвиђања операнада. Као што је већ раније у трећем поглављу ове докторске дисертације описано, предвиђање вредности се примењује над погодним инструкцијама и њиховим погодним операндима, а то су операнди који потичу из меморије (*T_MEM* и *R_MEM*). Ти операнди су препознати као погодни за предвиђање вредности из разлога што процесор мора приступити меморији како би прочитао њихову вредност, а то може изазвати заустављање извршавања. Према меморијској хијерархији уколико траженог податка нема у кеш меморији, процесор тада мора приступити оперативној меморији. Тај приступ траје неко време, а процесор не може да настави извршавање инструкције док се податак не дохвати из оперативне меморије.

Када инструкција има операнд који је специфициран меморијским адресирањем, таква инструкција мора да сачека док се вредност операнда не дохвати из меморије. Као што је већ поменуто како би се приказао феномен извршавања са непрецизно предвиђеним операндима, сви приступи меморији су посматрани као промашаји у кеш меморији. На слици 5.2.1. је приказан модел извршавања без предвиђања вредности операнада.



Слика 5.2.1. Извршавање без предвиђања операнада

На графу са слике 5.2.1. је представљено извршавање инструкција које су означене са e_i где i припада опсегу $[1 \ldots k+1]$. Време које је потребно да се инструкција изврши и пређе у наредни чвор је означено са t_j где j припада опсегу $[1 \ldots k]$. Посматраће се потребно време да се изврше инструкције од e_1 до e_k, укључујући и e_k. Инструкција e_1 представља једну од погодних инструкција за предвиђање вредности операнада, али као што је речено овај модел описује извршавање без предвиђања вредности.

Како у општем случају није познато ког су типа инструкције које следе након инструкције e_1, а такође представљени модел није везан ни за једну конкретну архитектуру процесора, усвојено је да је време трајања сваке инструкције константа. Такође, усвојено је да је време трајања сваке инструкције једнако и означено је са t што је описано формулом 5.2.1. Време t је названо циклусом и у наставку ове докторске дисертације користи се термин циклус. Ако се разматра неки конкретан рачунарски систем (конкретна архитектура процесора), може се емпиријски утврдити просечно време трајања инструкција.

$$t = t_1 = t_2 = \cdots = t_k \tag{5.2.1.}$$

Инструкција означена са e_1 представља инструкцију чији се један операнд налази у меморији. Како је потребно неко време да се вредност операнда дохвати из меморије, уведен је нови чвор са ознаком *mem*. Чвором са ознаком *mem* моделовано је потребно време да се дохвати вредност из меморије. Време које је потребно да се дохвати вредност из меморије је означено са $t_{mem}(n)$ на слици 5.2.1. Параметар n представља колико циклуса инструкција e_1 чека да се заврши приступ меморији, тј. да се из меморије дохвати вредност њеног операнда. Време $t_{mem}(n)$ је описано формулом 5.2.2.

$$t_{mem}(n) = nt \tag{5.2.2.}$$

Извршавање напредује тако што инструкција e_1 прелази у стање (чвор) *mem*, а потом се из тог стања прелази у чвор e_2 након времена $t_{mem}(n)$. Према формули 5.2.2. извршавање се зауставља n циклуса чиме је моделовано кашњење меморије. Када се заврши приступ меморији тада се извршавање наставља инструкцијом e_2.

Може се посматрати нека вредност за параметар n која припада опсегу $[1 \ldots k]$. На основу тога може се дефинисати време које је потребно да се изврше инструкције од e_1 закључно са инструкцијом e_k. То време је означено са $t_e(n)$ и дефинисано је формулом 5.2.3. Параметар k представља број циклуса потребних да се дохвати вредност операнда инструкције уколико инструкција операнд специфицира неким меморијским адресирањем (*T_MEM* тип операнда). То значи да ће инструкција иницирати дохватање вредности операнда из меморије које ће трајати k циклуса.

$$t_e(n) = t_1 + t_{mem}(n) + \sum_{i=2}^{k} t_i = t_{mem}(n) + \sum_{i=1}^{k} t_i \tag{5.2.3}$$

Даље се могу дефинисати две суме S_n и S_k формулама 5.2.4. и 5.2.5., респективно. Како је формулом 5.2.2. дефинисано кашњење меморије као време од n циклуса, оно се може представити и сумом S_n. Сума S_n се може дефинисати као време потребно да се изврши n инструкција. Сума S_k представља суму времена потребног да се изврше инструкције почевши од e_1 закључно са e_k.

$$S_n = \sum_{i=1}^{n} t_i \tag{5.2.4}$$

$$S_k = \sum_{i=1}^{k} t_i \tag{5.2.5.}$$

Сада на основу дефинисаних сума S_n и S_k може се извести скраћени запис формуле 5.2.3. која представља време извршавања инструкција e_1 - e_k. Скраћени запис формуле 5.2.3. је дат формулом 5.2.6.

$$t_e(n) = S_n + S_k \tag{5.2.6.}$$

Треба приметити да вредност параметра n може бити мања од k. То је случај када инструкција e_1 користи операнд чија вредност долази из меморије, али је његово дохватање иницирала нека претходна инструкција нпр. нека инструкција учитавања из меморије. Инструкција e_1 ће ту вредност користити из регистра, али ће ту вредност нека ранија инструкција довући из меморије у тај регистар (*R_MEM* тип операнда). У таквом сценарију инструкција e_1 неће чекати k циклуса већ мање. Колико ће мање чекати вредност инструкција e_1 зависи колико раније је иницирано дохватање вредности, тј. чекаће онолико циклуса мање колико циклуса раније је иницирано дохватање.

У представљеном моделу дефинисано време које је потребно да се заврши приступ меморији зависи од параметра n. Овај параметар је променљив, његова вредност зависи од тога да ли се ради о *T_MEM* операнду када приступ меморији траје тачно k циклуса или траје мање од k циклуса уколико се ради *R_MEM* операнду. На основу тога у оквиру модела уведена је вероватноћа $p(n)$ која говори колика је вероватноћа да вредност операнда која се дохвата из меморије постане доступна након n циклуса. Коришћењем израза који дефинише време из формуле 5.2.6. и вероватноће $p(n)$ може се дефинисати математичко очекивање које представља очекивано време извршавања инструкција почевши од инструкције e_1 закључно са инструкцијом e_k. Математичко очекивање је дато формулом 5.2.7.

$$E\big(t_e(n)\big) = \sum_{n=1}^{k} p(n)t_e(n) = \sum_{n=1}^{k} p(n)(S_n + S_k) \tag{5.2.7.}$$

Како само сума S_n зависи од параметра n, док сума S_k не зависи, формула 5.2.7. се може даље трансформисати до коначног облика који је дат формулом 5.2.9. Такође, како вредност операнда мора сигурно бити дохваћена након неког броја циклуса може се дефинисати сума вероватноће $p(n)$ формулом 5.2.8. Ова формула каже да је сума тих вероватноћа једнака јединици јер ће се вредност операнда добити из меморије сигурно у распону времена од једног циклуса до максимално k циклуса.

$$\sum_{n=1}^{k} p(n) = 1 \tag{5.2.8.}$$

$$E\big(t_e(n)\big) = \sum_{n=1}^{k} p(n)S_n + \sum_{n=1}^{k} p(n)S_k$$

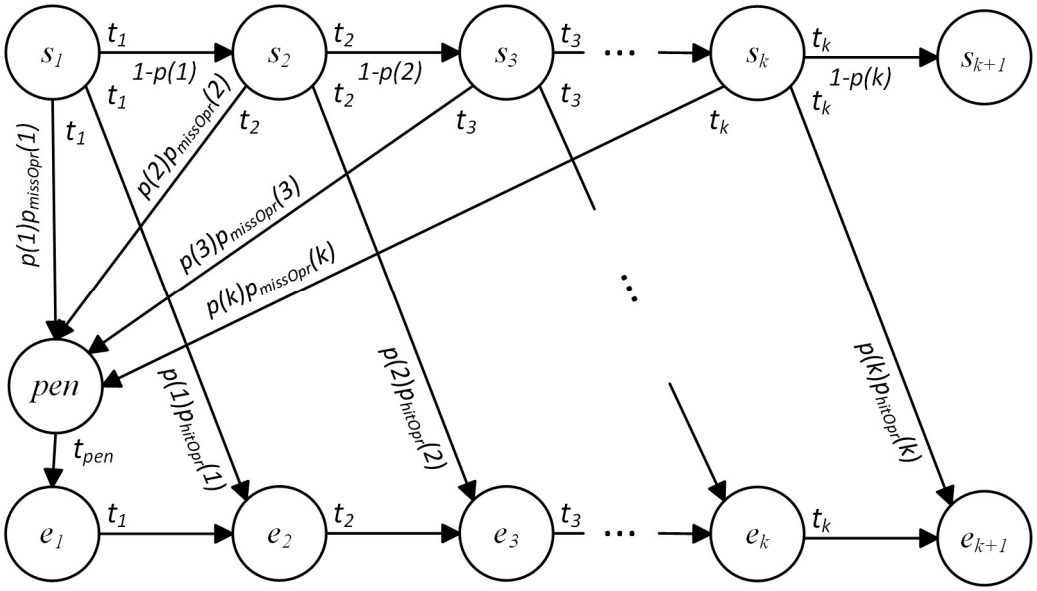
$$E\big(t_e(n)\big) = \sum_{n=1}^{k} p(n)S_n + S_k$$

$$E\big(t_e(n)\big) = S_k + \sum_{n=1}^{k} p(n)S_n \tag{5.2.9.}$$

Математичко очекивање које представља очекивано време извршавања по моделу са слике 5.2.1. је у свом коначном облику представљено формулом 5.2.9. Може се уочити да је то време једнако збиру два сабирка. Први сабирак јесте сума S_k која представља време да се изврше инструкције од e_1 до e_k. Други сабирак јесте сума производа вероватноће $p(n)$ и суме S_n, што представља очекивано време да вредност операнда стигне из меморије.

5.3. Извршавање са тачно предвиђеним операндима

Модел спекулативног извршавања инструкција са предвиђањем вредности погодних операнада погодних инструкција подразумева да се користи предвиђање вредности и да се извршавање наставља само уколико је вредност операнда тачно предвиђена. У случају да није тачно предвиђена вредност операнда, тада се извршавање мора вратити на ону инструкцију која је започела спекулативно извршавање. Овај модел је приказан на слици 5.3.1.



Слика 5.3.1. Извршавање са тачно предвиђеним вредностима операнада

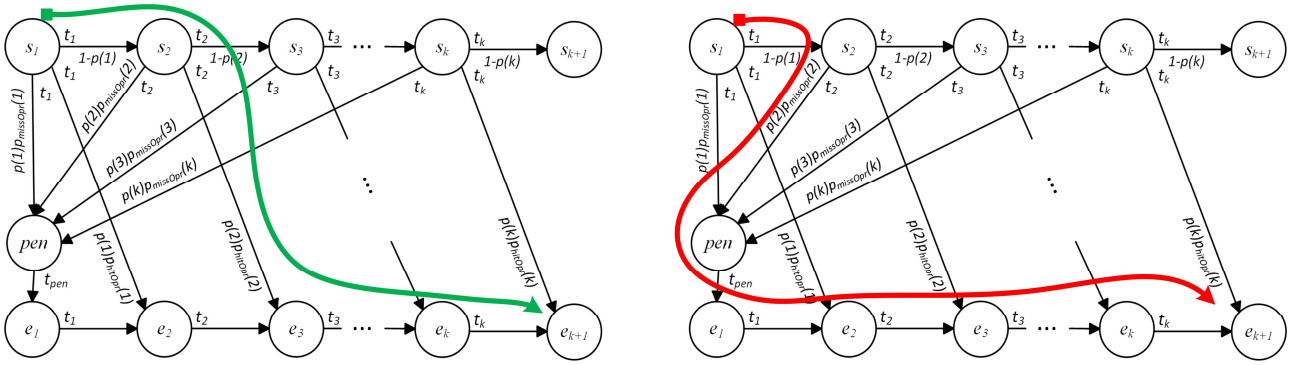
На горњем делу графа извршавања са слике 5.3.1. могу се уочити чворови означени ознакама од s_1 до s_k. Ови чворови представљају спекулативно извршавање инструкција које су означене чворовима e_1 до e_k. Чвор s_1 представља спекулативно извршавање инструкције e_1. У овом моделу извршавања примењено је предвиђање вредности операнда за инструкцију e_1. Инструкција e_1 је инструкција са једним операндом типа *ANY_MEM*. Како инструкција не би чекала, тј. како се не би заустављало извршавање док се не дохвати вредност операнда из меморије, овде се може применити предвиђање вредности тог операнда. Тада се инструкција e_1 спекулативно извршава (s_1) као и остале инструкције које следе након ње (s_2 - s_k).

Као у првом моделу извршавања који је описан у потпоглављу 5.2. и у овом моделу важи да би инструкција e_1 чекала n циклуса да се дохвати вредност операнда из меморије у случају да се не користи предвиђање вредности. На графу са слике 5.3.1. могу се уочити вероватноће $p(n)$ где је n број циклуса након колико стиже тачна вредност операнда из меморије. Уколико тачна вредност операнда не стигне након n циклуса тада се прелази на спекулативно извршавање наредне инструкције што се догађа са вероватноћом $1\text{-}p(n)$.

Уколико након n циклуса стигне тачна вредност операнда, где је параметар n у опсегу од *1* до k, тада се извршавање из одговарајућег чвора s_n, које је до тог тренутка било спекулативно, може наставити двема различитим путањама. Једна путања представља ситуацију у којој је тачно предвиђена вредност операнда. Друга путања којом се може наставити извршавање јесте ситуација у којој вредност операнда није тачно предвиђена.

Уколико је предвиђена вредност операнда једнака тачној вредности операнда, тада се прелази на извршавање инструкције e_{n+1}. Вероватноћа којом се прелази из чвора s_n у чвор e_{n+1} је једнака $p(n)p_{hitOpr}(n)$. Вероватноћа $p_{hitOpr}(n)$ је вероватноћа да је тачно предвиђена вредност операнда (погодак) чија тачна вредност долази из меморије након n циклуса. Путања извршавања која представља описан сценарио, уколико се посматра ситуација у којој је тачна вредност операнда доступна након два циклуса ($n=2$), приказана је на слици 5.3.2. лево.

Уколико је предвиђена вредност операнда није једнака тачној вредности операнда, тада се прелази у чвор који је означен са *pen* на слици 5.3.1. Вероватноћа којом се прелази из чвора s_n у чвор *pen* је једнака $p(n)p_{missOpr}(n)$. Вероватноћа $p_{missOpr}(n)$ је вероватноћа да није тачно предвиђена вредност операнда чија тачна вредност долази из меморије након n циклуса. Путања извршавања која представља описан сценарио, уколико се посматра ситуација у којој је тачна вредност операнда доступна након два циклуса ($n=2$), приказана је на слици 5.3.2. десно.



Слика 5.3.2. Путање извршавања – тачно предвиђена вредност операнда (лево) и нетачно предвиђена вредност операнда (десно)

Чвор који је означен са *pen* представља стање у које прелази извршавање, тј. процесор у ситуацији када се установи да предвиђање вредности операнда није исправно. Процесор тада мора предузети одређене радње како би извршио опоравак услед лошег предвиђања. Те радње су моделоване овим чвором, а време које је потребно да се изврши опоравак је представљено са t_{pen}.

Као што је дефинисано време извршавања за модел извршавања без предвиђања вредности операнада, у формули 5.3.1. представљено је време извршавања за модел са предвиђањем вредности операнда. Формула 5.3.1. представља време потребно да се изврше инструкције од e_1 до e_k за случај да тачна вредност предвиђеног операнда постане доступна након n циклуса где је n у опсегу од *1* до k.

$$t_{pOpr}(n) = \sum_{i=1}^{n} t_i + p_{missOpr}(n)\left(t_{pen} + \sum_{i=1}^{k} t_i\right) + p_{hitOpr}(n)\sum_{i=n+1}^{k} t_i \qquad (5.3.1.)$$

Први сабирак дате формуле представља време потребно да се изврше инструкције до тренутка када тачна вредност предвиђеног операнда постане доступна. Други сабирак представља време које је потребно да се изврше инструкције за случај да је лоше предвиђен операнд. Тада је неопходно одрадити опоравак од грешке за шта је потребно t_{pen}, а потом извршити све инструкције од e_1 до e_k. Трећи сабирак представља остатак потребног времена да се изврше преостале инструкције од e_{n+1} до e_k у случају да је предвиђање вредности операнда било исправно.

Релација између вероватноћа поготка ($p_{hitOpr}(n)$) и вероватноће промашаја ($p_{missOpr}(n)$) предвиђања вредности операнда може се дефинисати на начин да је збир ових вероватноћа једнак јединици. На основу тога може се изразити вероватноћа промашаја предвиђања вредности операнда преко вероватноће поготка предвиђања вредности операнда формулом 5.3.2.

$$p_{missOpr}(n) = 1 - p_{hitOpr}(n) \tag{5.3.2.}$$

Даље, на основу релације из формуле 5.3.2. може се развити израз који је дат формулом 5.3.1. до облика 5.3.3.:

$$t_{pOpr}(n) = \sum_{i=1}^{n} t_i + \Big(1 - p_{hitOpr}(n)\Big)\left(t_{pen} + \sum_{i=1}^{k} t_i\right) + p_{hitOpr}(n) \sum_{i=n+1}^{k} t_i$$

$$t_{pOpr}(n) = \sum_{i=1}^{n} t_i + \Big(1 - p_{hitOpr}(n)\Big) t_{pen} + \Big(1 - p_{hitOpr}(n)\Big) \sum_{i=1}^{k} t_i + p_{hitOpr}(n) \sum_{i=n+1}^{k} t_i$$

$$t_{pOpr}(n) = \sum_{i=1}^{n} t_i + \Big(1 - p_{hitOpr}(n)\Big) t_{pen} + \sum_{i=1}^{k} t_i - p_{hitOpr}(n) \sum_{i=1}^{k} t_i + p_{hitOpr}(n) \sum_{i=n+1}^{k} t_i$$

$$t_{pOpr}(n) = \sum_{i=1}^{n} t_i + \Big(1 - p_{hitOpr}(n)\Big) t_{pen} + \sum_{i=1}^{k} t_i + p_{hitOpr}(n)\left(\sum_{i=n+1}^{k} t_i - \sum_{i=1}^{k} t_i\right)$$

$$t_{pOpr}(n) = \sum_{i=1}^{n} t_i + \Big(1 - p_{hitOpr}(n)\Big) t_{pen} + \sum_{i=1}^{k} t_i - p_{hitOpr}(n) \sum_{i=1}^{n} t_i$$

$$t_{pOpr}(n) = \sum_{i=1}^{k} t_i + \Big(1 - p_{hitOpr}(n)\Big) t_{pen} + \sum_{i=1}^{n} t_i - p_{hitOpr}(n) \sum_{i=1}^{n} t_i$$

$$t_{pOpr}(n) = \sum_{i=1}^{k} t_i + \Big(1 - p_{hitOpr}(n)\Big) t_{pen} + \Big(1 - p_{hitOpr}(n)\Big) \sum_{i=1}^{n} t_i$$

$$t_{pOpr}(n) = \sum_{i=1}^{k} t_i + \Big(1 - p_{hitOpr}(n)\Big)\left(t_{pen} + \sum_{i=1}^{n} t_i\right)$$

$$t_{pOpr}(n) = \sum_{i=1}^{k} t_i + p_{missOpr}(n)\left(t_{pen} + \sum_{i=1}^{n} t_i\right) \tag{5.3.3.}$$

У добијеном изразу 5.3.3. могу се уочити две суме које су већ дефинисане формулама 5.2.4 и 5.2.5. Према тим формулама може се дефинисати коначан израз за време $t_{pOpr}(n)$ који је дат формулом 5.3.4.

$$t_{pOpr}(n) = S_k + p_{missOpr}(n)\left(t_{pen} + S_n\right) \tag{5.3.4.}$$

У изразу 5.3.4. се може уочити да уколико вероватноћа промашаја предвиђања вредности операнда тежи нули, тада се израз своди само на суму S_k, што јесте време потребно да се само изврше све инструкције од e_1 до e_k. Док у случају уколико вероватноћа промашаја предвиђања вредности операнда тежи јединици, тада се време извршавања повећава за време трајања спекулативно извршених инструкција (S_n) и за време потребно да се изврши опоравак услед лошег предвиђања t_{pen}.

Како вредност параметра n зависи од тога да ли се ради о *T_MEM* операнду када приступ меморији траје тачно k циклуса или траје мање од k циклуса уколико се ради *R_MEM* операнду, вероватноћа $p(n)$ говори колика је вероватноћа да вредност операнда која се дохвата из меморије постане доступна након n циклуса. У оквиру формуле 5.3.5. је поновљен дефинисан израз из формуле 5.2.8. који каже да је сума свих вероватноћа $p(n)$ једнака јединици.

$$\sum_{n=1}^{k} p(n) = 1 \tag{5.3.5.}$$

Коришћењем израза који дефинише време из формуле 5.3.4., вероватноће $p(n)$ и израза из формуле 5.3.5. може се дефинисати математичко очекивање које представља очекивано време извршавања инструкција почевши од инструкције e_1 закључно са инструкцијом e_k:

$$E\left(t_{pOpr}(n)\right) = \sum_{n=1}^{k} p(n) t_{pOpr}(n)$$

$$E\left(t_{pOpr}(n)\right) = \sum_{n=1}^{k} p(n)\left(S_k + p_{missOpr}(n)(t_{pen} + S_n)\right)$$

$$E\left(t_{pOpr}(n)\right) = \sum_{n=1}^{k} p(n)\, S_k + \sum_{n=1}^{k} p(n) p_{missOpr}(n)\left(t_{pen} + S_n\right)$$

$$E\left(t_{pOpr}(n)\right) = S_k + \sum_{n=1}^{k} p(n) p_{missOpr}(n)\left(t_{pen} + S_n\right) \tag{5.3.6.}$$

Математичко очекивање које представља очекивано време извршавања по моделу са слике 5.3.1. је у свом коначном облику представљено формулом 5.3.6. Може се уочити да је то време једнако збиру два сабирка. Први сабирак јесте сума S_k која представља време да се изврше инструкције од e_1 до e_k. Други сабирак јесте сума производа вероватноће $p(n)$, вероватноће $p_{missOpr}(n)$ и збира кога чине сума S_n и време t_{pen}. Описани други сабирак заправо представља очекивано време које се изгуби у току извршавања услед погрешно предвиђене вредности операнда. Уколико је вероватноћа промашаја $p_{missOpr}(n)$ мања, тј. ако је блиска вредности нула, тада је и очекивано време опоравка мало и укупно време извршавања тежи ка суми S_k, што је идеалан сценарио. Идеалним сценарионом сматра се сценарио у коме је увек предвиђање вредности операнда било исправно.

Код овог модела треба напоменути да је потребно и неко време да би се утврдила исправност предвиђања. Када се приступ меморији заврши, тј. када тачна вредност операнда постане доступна тада је потребно упоредити је са предвиђеном вредношћу тог операнда. То време није моделовано у представљеном моделу извршавања. Разлог за то је што је за такво поређење две вредности потребно значајно мање времена од времена да се изврши једна инструкција. Због тога је усвојено у моделу да се у оквиру времена извршавања инструкције врши и поређење тачне вредности и предвиђене вредности операнда. На овај начин је модел извршавања упрошћен, а то време не учествује у формулама које описују модел.

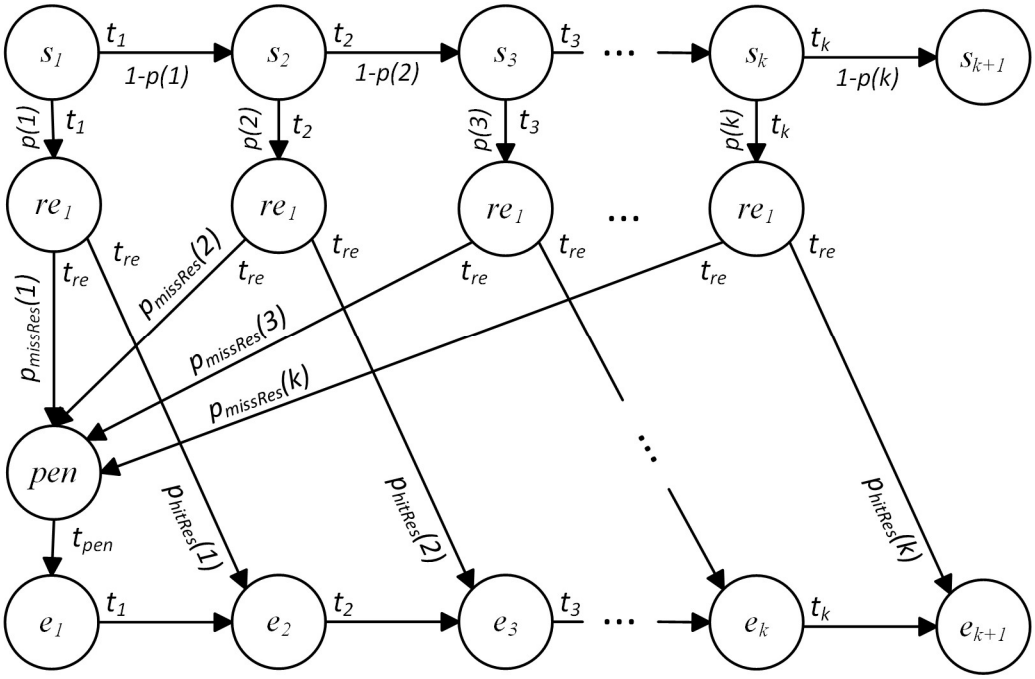
5.4. Извршавање са непрецизно предвиђеним операндима

Попут модела који је описан у претходном потпоглављу, а који користи предвиђање вредности операнда типа *ANY_MEM*, и модел који се описује у овом потпоглављу користи предвиђање вредности операнда. Разлика између претходног модела и модела са непрецизно предвиђеним операндима јесте у томе што модел са непрецизно предвиђеним операндима може да настави извршавање у одређеним ситуацијама иако вредност операнда није исправно предвиђена.

У моделу са непрецизно предвиђеним операндима битно је да је резултат инструкције исправан. То може код погодних типова инструкција, које су описане у поглављу 3. ове докторске дисертације, бити у два случају. Наравно, први случај је у ситуацији када је предвиђена вредност операнда једнака тачној вредности операнда, тј. када је предвиђање исправно. Тада је резултат инструкције са тачном вредношћу операнда идентичан резултату инструкције са предвиђеном вредношћу операнда. Други случај када резултат инструкције може бити исправан, јесте када и непрецизно предвиђена вредност операнда доводи до тачног резултата инструкције.

На слици 5.4.1. представљен је модел извршавања са непрецизно предвиђеним операндима. Слично као код претходног модела могу се уочити чворови означени ознакама од s_1 до s_k као и чворови од e_1 до e_k. Чворови од s_1 до s_k представљају спекулативно извршавање одговарајућих инструкција означене ознакама од e_1 до e_k. Чвором s_1 је представљено спекулативно извршавање инструкције e_1. Инструкција e_1 је инструкција са једним операндом типа *ANY_MEM*. Како инструкција не би чекала, тј. како се не би заустављало извршавање док се не дохвати вредност операнда из меморије, овде се може применити предвиђање вредности тог операнда као и у претходном моделу.

Као у првом моделу извршавања који је описан у потпоглављу 5.2. и у овом моделу важи да би инструкција e_1 чекала n циклуса да се дохвати вредност операнда из меморије у случају да се не користи предвиђање вредности. На графу са слике 5.4.1. могу се уочити вероватноће $p(n)$ где је n број циклуса након колико стиже тачна вредност операнда из меморије. Уколико тачна вредност операнда не стигне након n циклуса тада се прелази на спекулативно извршавање наредне инструкције што се догађа са вероватноћом $1-p(n)$.



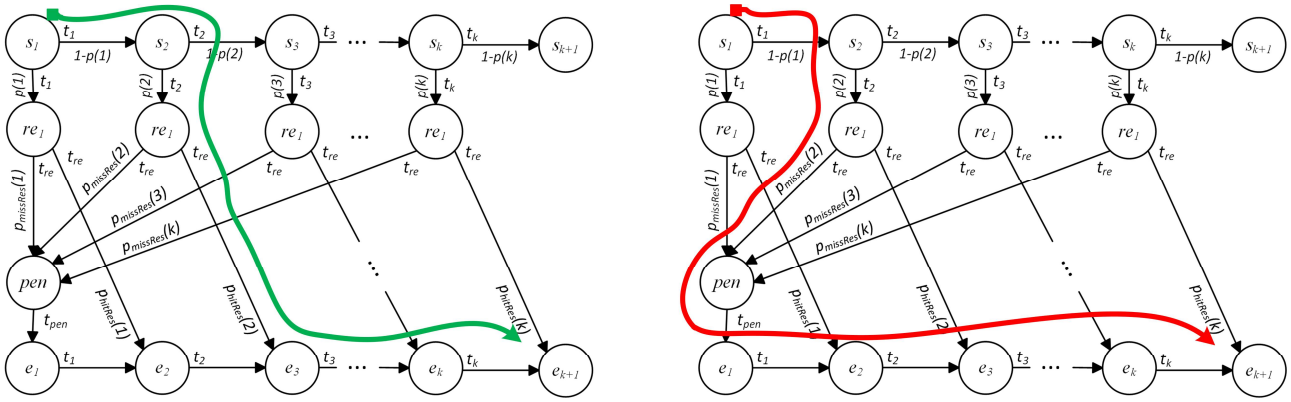
Слика 5.4.1. Извршавање са непрецизно предвиђеним вредностима операнада

Уколико након *n* циклуса стигне тачна вредност операнда, где је параметар *n* у опсегу од *1* до *k*, тада се извршавање из одговарајућег чвора s_n, које је до тог тренутка било спекулативно, наставља према чвору означеном са re_1. Како овај модел дозвољава да се извршавање настави и са непрецизно предвиђеним операндима ако је резултат исправан, онда се мора извршити додатна провера исправности резултата која је моделована овим чвором. У оквиру овог чвора када тачна вредност операнда постане доступна поново се извршава инструкција e_1 али сада са тачном вредношћу предвиђеног операнда. Такође, у оквиру овог чвора се врши упоређивање резултата инструкције добијеног са тачном вредношћу операнда и резултата инструкције добијеног са предвиђеном вредношћу операнда. Време потребно да се одраде описане радње моделовано је временом t_{re}.

Из чвора re_1 могуће је наставити извршавање двема различитим путањама у зависности од исхода поређења резултата инструкције добијеног на основу тачне и на основу предвиђене вредности операнда. Једна путања представља ситуацију у којој је резултат инструкције на основу тачне вредности операнда идентичан резултату на основу предвиђене вредности операнда. Друга путања којом се може наставити извршавање јесте ситуација у којој резултат на основу тачне вредности операнда није идентичан резултату на основу предвиђене вредности операнда.

Уколико је резултат инструкције на основу тачне вредности операнда идентичан резултату инструкције на основу предвиђене вредности операнда, тада се прелази на извршавање инструкције e_{n+1}. Вероватноћа којом се прелази из чвора re_1 у чвор e_{n+1} је једнака $p_{hitRes}(n)$. Вероватноћа $p_{hitRes}(n)$ је вероватноћа да је резултат инструкције исправан на основу предвиђене вредности операнда. За разлику од модела који прихвата само тачно предвиђену вредност операнда, овај модел може наставити извршавање и са непрецизно предвиђеним операндом, само је важно да је резултат инструкције исправан. Зато у овом моделу учествује вероватноћа која говори да ли је резултат инструкције исправан, а не вероватноћа која говори да ли је тачно предвиђена вредност операнда. Путања извршавања која представља

описан сценарио, уколико се посматра ситуација у којој је тачна вредност операнда доступна након два циклуса ($n=2$), приказана је на слици 5.4.2. лево.



Слика 5.4.2. Путање извршавања – тачан резултат инструкције на основу предвиђеног операнда (лево) и нетачан резултат инструкције на основу предвиђеног операнда (десно)

Уколико резултат инструкције на основу тачне вредности операнда није идентичан резултату инструкције на основу предвиђене вредности операнда, тада се прелази у чвор означен са *pen*. Вероватноћа којом се прелази из чвора re_1 у чвор *pen* је једнака $p_{missRes}(n)$. Вероватноћа $p_{missRes}(n)$ је вероватноћа да резултат инструкције није исправан на основу предвиђене вредности операнда. У таквој ситуацији извршавање пролази кроз чвор *pen* којим је моделован опоравак услед лошег предвиђања, а време потребно да се изврши опоравак је t_{pen}. Путања извршавања која представља описан сценарио, уколико се посматра ситуација у којој је тачна вредност операнда доступна након два циклуса ($n=2$), приказана је на слици 5.4.2. десно.

Као што је дефинисано време извршавања за претходно описане моделе извршавања у формули 5.4.1. представљено је време извршавања за модел са непрецизно предвиђеним вредностима операнада. Формула 5.4.1. представља време потребно да се изврше инструкције од e_1 до e_k за случај да тачна вредност предвиђеног операнда постане доступна након n циклуса где је n у опсегу од *1* до *k*.

$$t_{pRes}(n) = \sum_{i=1}^{n} t_i + p_{missRes}(n)\left(t_{re} + t_{pen} + \sum_{i=1}^{k} t_i\right) + p_{hitR}\quad(n)\left(t_{re} + \sum_{i=n+1}^{k} t_i\right) \qquad (5.4.1.)$$

Први сабирак дате формуле 5.4.1. представља време потребно да се изврше инструкције до тренутка када тачна вредност предвиђеног операнда постане доступна. Други сабирак представља време које је потребно да се изврше инструкције за случај да је лоше предвиђен операнд. Тада је прво неопходно проверити да ли је резултат инструкције са тачним операндом идентичан резултату са предвиђеним операндом за шта је потребно t_{re} времена. Потом треба одрадити опоравак од грешке за шта је потребно t_{pen} времена и на крају извршити све инструкције од e_1 до e_k. Трећи сабирак представља остатак потребног времена да се изврше преостале инструкције од e_{n+1} до e_k у случају да је резултат инструкције са тачним операндом и резултат инструкције са предвиђеним операндом идентичан. Заправо трећи сабирак представља време које је потребно да се изврше инструкције за случај да је операнд тачно предвиђен или је непрецизно предвиђен, али је резултат инструкције исправан.

Слично као у претходном моделу (потпоглавље 5.3.) може се дефинисати релација између вероватноћа тачног ($p_{hitRes}(n)$) и вероватноће нетачног ($p_{missRes}(n)$) резултата

инструкције на основу предвиђене вредности операнда. Збир ових вероватноћа једнак јединици. Према томе може се развити формула 5.4.1. до облика 5.4.2. на следећи начин:

$$t_{pRes}(n) = \sum_{i=1}^{n} t_i + p_{missRes}(n)t_{re} + p_{hitR} \quad (n)t_{re} + p_{missRes}(n)\left(t_{pen} + \sum_{i=1}^{k} t_i\right) + p_{hitR} \quad (n) \sum_{i=n+1}^{k} t_i$$

$$t_{pRes}(n) = \sum_{i=1}^{n} t_i + t_{re} + p_{missRes}(n)\left(t_{pen} + \sum_{i=1}^{k} t_i\right) + p_{hitRes}(n) \sum_{i=n+1}^{k} t_i \qquad (5.4.2.)$$

Може се изразити вероватноћа нетачног резултата инструкције на основу предвиђене вредности операнда формулом 5.4.3. Даље, на основу релације из формуле 5.4.3. може се развити израз који је дат формулом 5.4.2. до облика 5.4.4.:

$$p_{missRes}(n) = 1 - p_{hitRes}(n) \qquad (5.4.3.)$$

$$t_{pRes}(n) = \sum_{i=1}^{n} t_i + t_{re} + \big(1 - p_{hitRes}(n)\big)\left(t_{pen} + \sum_{i=1}^{k} t_i\right) + p_{hi} \qquad (n) \sum_{i=n+1}^{k} t_i$$

$$t_{pRes}(n) = \sum_{i=1}^{n} t_i + t_{re} + \big(1 - p_{hitRes}(n)\big)t_{pen} + \big(1 - p_{hitRes}(n)\big) \sum_{i=1}^{k} t_i + p_{hitRes}(n) \sum_{i=n+1}^{k} t_i$$

$$t_{pRes}(n) = \sum_{i=1}^{n} t_i + t_{re} + \big(1 - p_{hitRes}(n)\big)t_{pen} + \sum_{i=1}^{k} t_i - p_{hitRes}(n) \sum_{i=1}^{k} t_i + p_{hitRes}(n) \sum_{i=n+1}^{k} t_i$$

$$t_{pRes}(n) = \sum_{i=1}^{n} t_i + t_{re} + \big(1 - p_{hitRes}(n)\big)t_{pen} + \sum_{i=1}^{k} t_i + p_{hitRes}(n)\left(\sum_{i=n+1}^{k} t_i - \sum_{i=1}^{k} t_i\right)$$

$$t_{pRes}(n) = \sum_{i=1}^{n} t_i + t_{re} + \big(1 - p_{hitRes}(n)\big)t_{pen} + \sum_{i=1}^{k} t_i - p_{hitRes}(n) \sum_{i=1}^{n} t_i$$

$$t_{pRes}(n) = \sum_{i=1}^{k} t_i + t_{re} + \big(1 - p_{hitRes}(n)\big)t_{pen} + \sum_{i=1}^{n} t_i - p_{hitRes}(n) \sum_{i=1}^{n} t_i$$

$$t_{pRes}(n) = \sum_{i=1}^{k} t_i + t_{re} + \big(1 - p_{hitRes}(n)\big)t_{pen} + \big(1 - p_{hitRes}(n)\big) \sum_{i=1}^{n} t_i$$

$$t_{pRes}(n) = \sum_{i=1}^{k} t_i + t_{re} + \big(1 - p_{hitRes}(n)\big)\left(t_{pen} + \sum_{i=1}^{n} t_i\right)$$

$$t_{pRes}(n) = \sum_{i=1}^{k} t_i + t_{re} + p_{missRes}(n)\left(t_{pen} + \sum_{i=1}^{n} t_i\right) \qquad (5.4.4.)$$

У добијеном изразу 5.4.4. могу се уочити две суме које су већ дефинисане формулама 5.2.4 и 5.2.5. Према тим формулама може се дефинисати коначан израз за време $t_{pRes}(n)$ који је дат формулом 5.4.5.

$$t_{pRes}(n) = S_k + t_{re} + p_{missRes}(n)\big(t_{pen} + S_n\big) \qquad (5.4.5.)$$

Израз 5.4.5. се разликује од израза 5.3.4. (описује време извршавања само са тачно предвиђеним операндима) у додатном времену t_{re} које је потребно за поновно извршавање инструкције за коју се вршило предвиђање операнда. У изразу 5.4.5. се може уочити да уколико вероватноћа нетачног резултата инструкције на основу предвиђене вредности операнда тежи нули, тада се израз своди само на збир суме S_k и времена t_{re}, што јесте време потребно да се изврше све инструкције од e_1 до e_k увећано за време које је потребно да се поново изврши инструкција e_1. Док у случају уколико вероватноћа нетачног резултата инструкције на основу предвиђене вредности операнда тежи јединици, тада се време извршавања повећава за време трајања спекулативно извршених инструкција (S_n) и за време потребно да се изврши опоравак услед лошег предвиђања t_{pen}.

Исто као у претходно описаном моделу вредност параметра n зависи од тога да ли се ради о *T_MEM* операнду када приступ меморији траје тачно k циклуса или траје мање од k циклуса уколико се ради *R_MEM* операнду, уведена је вероватноћа $p(n)$. Она говори колика је вероватноћа да вредност операнда која се дохвата из меморије постане доступна након n циклуса. Као у претходном моделу (потпоглавље 5.3.) и за овај модел важи израз из формуле 5.3.5. који каже да је сума вероватноћа $p(n)$ једнака јединици. Коришћењем израза који дефинише време из формуле 5.4.5. и вероватноће $p(n)$ може се дефинисати математичко очекивање које представља очекивано време извршавања инструкција почевши од инструкције e_1 закључно са инструкцијом e_k:

$$E\left(t_{pRes}(n)\right) = \sum_{n=1}^{k} p(n) t_{pRes}(n)$$

$$E\left(t_{pRes}(n)\right) = \sum_{n=1}^{k} p(n)\left(S_k + t_{re} + p_{missRes}(n)(t_{pen} + S_n)\right)$$

$$E\left(t_{pRes}(n)\right) = \sum_{n=1}^{k} p(n)\, S_k + \sum_{n=1}^{k} p(n)\, t_{re} + \sum_{n=1}^{k} p(n) p_{missRes}(n)\left(t_{pen} + S_n\right)$$

$$E\left(t_{pRes}(n)\right) = S_k + t_{re} + \sum_{n=1}^{k} p(n) p_{missRes}(n)\left(t_{pen} + S_n\right) \qquad (5.4.6.)$$

Математичко очекивање које представља очекивано време извршавања по моделу са слике 5.4.1. је у свом коначном облику представљено формулом 5.4.6. Може се уочити да је то време једнако збиру три сабирака. Први сабирак јесте сума S_k која представља време да се изврше инструкције од e_1 до e_k. Трећи сабирак јесте сума производа вероватноће $p(n)$, вероватноће $p_{missRes}(n)$ и збира кога чине сума S_n и време t_{pen}. Описани трећи сабирак заправо представља очекивано време које се изгуби у току извршавања уколико резултат инструкције није исправан на основу предвиђене вредности операнда. Уколико је вероватноћа $p_{missRes}(n)$ мања, тј. ако је блиска вредности нула, тада је и очекивано време опоравка мало и укупно време извршавања тежи ка збиру суме S_k и времена t_{re}, што је идеалан сценарио. Идеалним сценарием за овај модел извршавања сматра се сценарио у коме је увек резултат инструкције исправан на основу предвиђене вредности операнда.

Код овог модела треба напоменути да је потребно и неко време да би се утврдило да ли је резултат инструкције на основу тачне вредности операнда идентичан резултату инструкције на основу предвиђене вредности операнда. Када се приступ меморији заврши, тј. када тачна вредност операнда постане доступна тада се поново извршава инструкција за

коју је вршено предвиђање што је моделовано чвором re_1. Време које је потребно да се поново изврши инструкција моделовано је са t_{re}. У претходном моделу је време које је потребно да се утврди исправност предвиђања занемарено јер је потребно само упоредити две вредности. У овом моделу је неопходно извршити поново инструкцију како би се формирао комплетан резултат инструкције на основу тачних вредности операнада и потом упоредити тај резултат са резултатом који је добијен на основу предвиђене вредности операнда.

Према дефинисаним моделима извршавања који користе предвиђање вредности операнада може се прокоментарисати уз које претпоставке се може очекивати да модел који прихвата и непрецизно предвиђене операнде уколико је резултат тачан, постиже боље време извршавања. Другим речима може се прокоментарисати када је очекивати да је исплативо користити такав модел. Као што је речено време извршавања модела са непрецизно предвиђеним операндима се увећава за време потребно да се поново изврши инструкција за коју се вршило предвиђање (t_{re}) без обзира на то какав је исход предвиђања, док тог времена нема код модела који прихвата само тачно предвиђене вредности операнада. Ово значи да модел са непрецизно предвиђеним операндима треба то време на неки начин надокнадити. Дефинисана очекивана времена извршавања ових модела имају један исти сабирак суму S_k, што значи да време које треба надокнадити модел са непрецизно предвиђеним операндима једино може бити у оквиру сабирка који представља време које се утроши на погрешно спекулативно извршене инструкције и опоравак услед лошег предвиђања. Вредност тог сабирка код модела са тачно предвиђеним операндима зависи од вероватноће промашаја тачне вредности операнда ($p_{missOpr}$), а код модела са непрецизно предвиђеним операндима од вероватноће промашаја тачног резултата инструкције на основу предвиђеног операнда ($p_{missRes}$). Под претпоставком да је вероватноћа $p_{missRes}$ довољно мања од вероватноће $p_{missOpr}$, модел са непрецизно предвиђеним вредности операнда сакрива (надокнађује) време t_{re} и може постићи и боље време извршавања. Другим речима уколико се довољно чешће добијају тачни резултати инструкција на основу предвиђених вредности операнада него што се добијају тачно предвиђене вредности операнада, тада је исплативо користити модел са непрецизно предвиђеним операндима. Још једна претпоствка јесте да се у оквиру реалних програма јављају погодне иснтрукције (описане у претходном поглављу) за непрецизно предвиђање вредности како би модел са непрецизно предвиђеним вредностима операнада био применљив.

5.5. Резиме аналитичких модела извршавања

У оквиру овог потпоглавља је дат кратак осврт на описане моделе како би се уочиле разлике између њих. На основу тога су аналитички описани услови када модел који наставља извршавање уколико је исправан резултат инструкције чак и у случају непрецизно предвиђеног операнда постиже боље време извршавања у односу на модел који извршавање наставља само са тачно предвиђеним операндом. Такође, поновљене су формуле које описују очекивано време извршавања за сва три модела:

- формула 5.2.9. одговара моделу извршавања који не укључује предвиђање вредности операнада. У наставку ове докторске дисертације за овај модел користиће се назив модел 1;
- формула 5.3.6. одговара моделу који укључује предвиђање вредности операнада, а извршавање наставља само у случају да је комплетно тачно

предвиђена вредност операнда. У наставку ове докторске дисертације за овај модел користиће се назив модел 2;

- формула 5.4.6. одговара моделу који такође укључује предвиђање вредности операнда али за разлику од модела 2 извршавање наставља чак и са непрецизно предвиђеном вредношћу операнда уколико је резултат инструкције исправан. У наставку ове докторске дисертације за овај модел користиће се назив модел 3.

$$E\big(t_e(n)\big) = S_k + \sum_{n=1}^{k} p(n) S_n \tag{5.2.9.}$$

$$E\left(t_{pOpr}(n)\right) = S_k + \sum_{n=1}^{k} p(n) p_{missOpr}(n)\big(t_{pen} + S_n\big) \tag{5.3.6.}$$

$$E\left(t_{pRes}(n)\right) = S_k + t_{re} + \sum_{n=1}^{k} p(n) p_{missRes}(n)\big(t_{pen} + S_n\big) \tag{5.4.6.}$$

У дефинисаним формулама које описују очекивано време извршавања може се уочити да сва три модела имају као сабирак суму S_k. Као што је раније дефинисано ова сума представља време да се изврше инструкције са тачним вредностима операнада. Модел 1 додатно укључује и време које је потребно да се заврши приступ меморији за операнд који недостаје инструкцији, што је представљено другим сабирком у оквиру формуле 5.2.9. Модел 2 укључује поред суме S_k и време које неопходно да се изврши опоравак услед лошег предвиђања као и утрошено време на спекулативно извршене инструкције на основу погрешно предвиђене вредности операнда. Модел 3 поред суме S_k и времена потребног за опоравак и утрошеног времена на извршавање спекулативних инструкција чији резултат није исправан, укључује још и додатно време t_{re} за поновно извршавање инструкције чији операнд се предвиђао.

Уколико су вероватноће $p_{missOpr}$ и $p_{missRes}$ мале, тада се очекивано време извршавања модела 2 своди само на суму S_k, а модела 3 на суму S_k увећану за време t_{re}. Таква очекивана времена извршавања су свакако мања него код модела 1 где сигурно извршавање мора да се заустави док се не дохвати вредност операнда из меморије.

Према једној од хипотеза ове докторске дисертације очекивано време извршавања модела 3 је мање од очекиваног времена извршавања модела 2 (хипотеза 6). Како би се упоредила очекивана времена извршавања модела 2 и модела 3 потребно је извршити одузимање очекиваног времена извршавања модела 3 од очекиваног времена извршавања модела 2, што је дато у наставку:

$$E\left(t_{pOpr}(n)\right) - E\left(t_{pRes}(n)\right) =$$

$$S_k + \sum_{n=1}^{k} p(n) p_{missOpr}(n)\big(t_{pen} + S_n\big) - S_k - t_{re} - \sum_{n=1}^{k} p(n) p_{missRes}(n)\big(t_{pen} + S_n\big) =$$

$$\sum_{n=1}^{k} p(n)\left(p_{missOpr}(n) - p_{missRes}(n)\right)\big(t_{pen} + S_n\big) - t_{re}$$

У добијеном изразу један од чинилаца производа у суми јесте разлика вероватноћа $p_{missOpr}$ и $p_{missRes}$ које зависе од параметра n. Уколико се претпостави да вероватноћа

промашаја предвиђања вредности операнда ($p_{missOpr}$) и вероватноћа да је резултат нетачан на основу предвиђеног операнда ($p_{missRes}$) не зависе од тога колико се циклуса чека вредност операнда из меморије (параметар n), добијени израз се може даље трансформисати до облика датог у формули 5.5.1.

$$E\left(t_{pOpr}(n)\right) - E\left(t_{pRes}(n)\right) = \left(p_{missOpr} - p_{missRes}\right)\sum_{n=1}^{k} p(n)\left(t_{pen} + S_n\right) - t_{re} \quad (5.5.1.)$$

Како би очекивано време извршавања модела 3 било мање од очекиваног времена извршавања модела 2 израз из формуле 5.5.1. треба бити већи од нуле. Може се на основу претходног запажања дефинисати неједначина која је представљена формулом 5.5.2.

$$\left(p_{missOpr} - p_{missRes}\right)\sum_{n=1}^{k} p(n)\left(t_{pen} + S_n\right) - t_{re} > 0 \quad (5.5.2.)$$

Даље се може моделовати сума S_n која представља време потребно да се тачна вредност дохвати из меморије као производ броја инструкција n и просечног трајања једне инструкције (трајање једног циклуса t), што је представљено формулом 5.5.3. Следећа ствар која се може моделовати као просечно трајање једне инструкције (циклус t) јесте време потребно да се поново изврши инструкција за коју се вршило предвиђање t_{re} и упореди тај резултат инструкције са резултатом добијеним на основу предвиђене вредности операнда, што је дато формулом 5.5.4.

$$S_n = \sum_{i=1}^{n} t_i = \sum_{i=1}^{n} t = nt \quad (5.5.3.)$$

$$t_{re} = t \quad (5.5.4.)$$

Неједначина из формуле 5.5.2. може се даље развити на основу формула 5.5.3. и 5.5.4. на следећи начин:

$$\left(p_{missOpr} - p_{missRes}\right)\left(\sum_{n=1}^{k} p(n)S_n + t_{pen}\right) - t_{re} > 0$$

$$\left(p_{missOpr} - p_{missRes}\right)\left(\sum_{n=1}^{k} p(n)nt + t_{pen}\right) - t > 0$$

$$\left(p_{missOpr} - p_{missRes}\right) > t/\left(\sum_{n=1}^{k} p(n)nt + t_{pen}\right)$$

Уколико се десна страна добијене неједначине сада подели са временом које означава циклус t даље се неједначина може развити на следећи начин:

$$\left(p_{missOpr} - p_{missRes}\right) > 1/\left(\sum_{n=1}^{k} p(n)n + t_{pen}/t\right)$$

У добијеном облику неједначине се могу увести нове ознаке према формулама 5.5.5. и 5.5.6. и на основу њих се може написати израз неједначине 5.5.2. у виду формуле 5.5.7.

$$p_{missOpr} - p_{missRes} = \Delta \tag{5.5.5.}$$

$$\sum_{n=1}^{k} p(n)n = s(k) \tag{5.5.6.}$$

$$\Delta > \frac{1}{s(k) + \frac{t_{pen}}{t}} \tag{5.5.7.}$$

У добијеној формули 5.5.7. учествују разлика вероватноћа Δ, функција *s(k)* где је *k* кашњење меморије и однос t_{pen}/t који представља однос времена потребног да се изврши опоравак услед лошег предвиђања и просечног трајања инструкције. Представљени модели извршавања нису везани ни за једну специфичну архитектуру процесора. Потребно је одредити услове под којима модел 3 постиже боље очекивано време извршавања од модела 2. Према формули 5.5.7. да би модел 3 постизао боље очекивано време извршавања неопходно је да важи добијена неједнакост. Даље се формула 5.5.7. може трансформисати до облика 5.5.8. на следећи начин:

$$\frac{t_{pen}}{t} > \frac{1}{\Delta} - s(k)$$

$$\frac{t_{pen}}{t} > f\big(\Delta, s(k)\big) \tag{5.5.8.}$$

Према коначном облику неједнакости која је представљена формулом 5.5.8. може се дефинисати услов када модел 3 постиже боље очекивано време извршавања од модела 2. Са леве стране неједнакости стоји однос времена које је потребно да се изврши опоравак и времена просечног трајања једне инструкције. Док се на десној страни неједнакости налази функција *f* која зависи од разлике вероватноћа Δ и функције *s(k)*. Све док важи неједнакост у формули 5.5.8. модел 3 постиже боље време извршавања од модела 2.

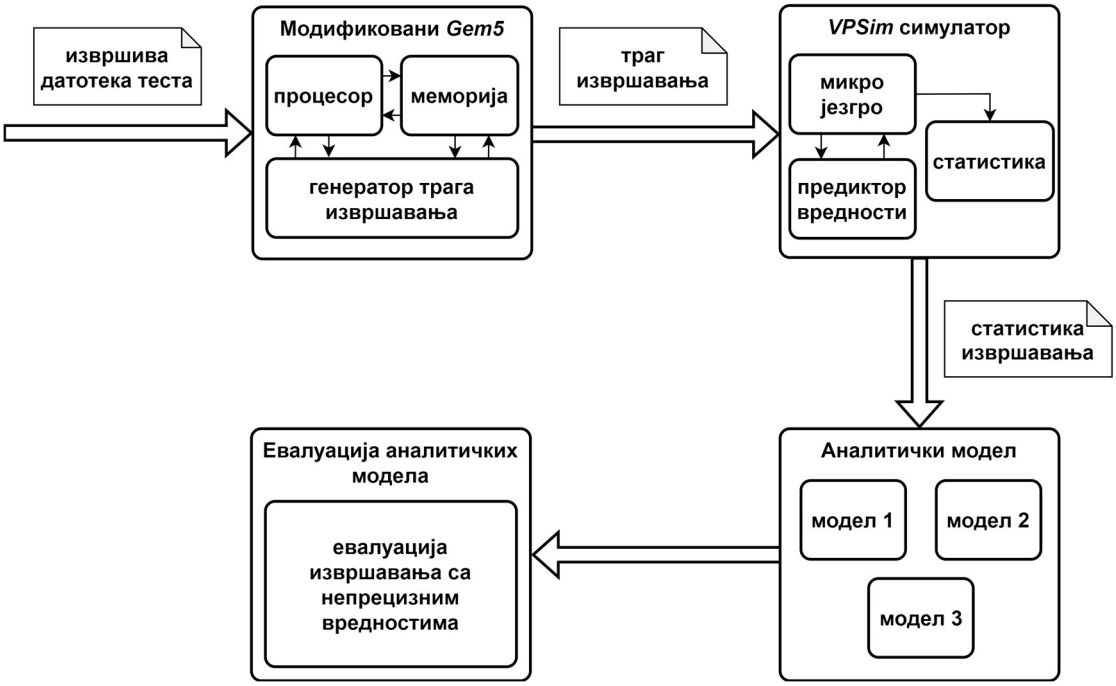
Уколико се посматра функција *f*, може се приметити да њена вредност може бити и негативна. Ситуација када вредност функције *f* може бити негативна јесте када је разлика вероватноћа Δ довољно велика па је њена реципрочна вредност мала. Одузимањем вредности функције *s(k)* од реципрочне вредности разлике вероватноћа Δ у таквој ситуацији се може добити и негативна вредност. То даље значи да модел 3 постиже боље очекивано време извршавања од модела 2 кад год функција *f* има негативну вредност. Разлог за то је што неједнакост из формуле 5.5.8. тада важи јер је однос времена које је потребно да се изврши опоравак услед промашаја и времена просечног трајања једне инструкције (t_{pen}/t) сигурно већи од нуле пошто су и времена t_{pen} и *t* већа од нуле.

6. МЕТОДОЛОГИЈА И ОПИС СИМУЛАЦИЈА

У оквиру овог поглавља биће детаљно описана методологија истраживања током рада на овој докторској дисертацији. Биће описани извршени експерименти, тј. симулације извршавања инструкција са предвиђањем вредности. Такође, ово поглавље садржи преглед коришћених тестова. Поред тога, ово поглавље садржи и опис коришћених симулатора. Један симулатор је симулатор отвореног кода под називом *Gem5* који је модификован за потребе прикупљања трага извршавања програма, а други симулатор је развијен од стране аутора ове докторске дисертације за потребе симулације рада самих предиктора вредности.

6.1. Методологија

У петом поглављу ове докторске дисертације представљени су аналитички модели извршавања инструкција са предвиђањем вредности операнада. На основу модела изведене су математичке формуле које описују очекивано време извршавања, а потом су и аналитички упоређена времена модела 2 и модела 3. У добијеним формулама фигуришу неке величине које је могуће добити путем симулације. Неке од величина које је могуће добити симулацијом јесу прецизност различитих предиктора, учесталост погодних инструкција за непрецизно предвиђање вредности као и учесталост *ANY_MEM* операнада и њихова расподела према времену колико дуго се чекају да стигну из меморије. На основу аналитичких модела и величина које су добијене симулацијама може се описати под којим условима модел 3 постиже боље време извршавања од модела 2. На слици 6.1.1. је скицирана примењена методологија у оквиру истраживања током израде ове докторске дисертације.



Слика 6.1.1. Методологија истраживања

Као прва ствар са слике 6.1.1. може се уочити блок који представља извршиву (енгл. *executable*) датотеку теста. У оквиру истраживања коришћене су две групе тестова перформанси процесора (енгл. *benchmarks*). Једну групу тестова представља подскуп тестова који припадају групи тестова *SPEC* [19], док другу групу тестова чине тестови који припадају групи тестова *EEMBC* [59]. Коришћени тестови су написани на програмским језицима *C* и *C++*. За њихово превођење до извршивих датотека коришћен је преводилац *gcc* за архитектуру *x86* на којој је спроведено истраживање. Добијене извршиве датотеке су потом коришћене као улаз симулатора *Gem5*. Коришћени тестови су детаљно описани у наставку ове докторске дисертације у оквиру потпоглавља 6.2.

Како је за извршавање тестова (извршиве датотеке) потребна и подршка оперативног система због тога што тестови користе системске услуге, одабран је симулатор *Gem5* [60] који има могућност емулирања системских услуга. Овај симулатор је симулатор отвореног кода и користи се у истраживањима везаним за архитектуру рачунара, а такође има и активну заједницу која га одржава. Подржава више архитектура, а међу њима и архитектуру *x86*. На слици 6.1.1. у оквиру модификованог симулатора *Gem5* представљени су посебним блоковима процесор, меморија и генератор трага извршавања. Симулатор *Gem5* је надограђен тако да прати извршавање инструкција, приступ меморији и генерише траг извршавања са неопходним информацијама за рад симулатора *VPSim*. Извршавање теста у оквиру симулатора *Gem5* захтева неколико сати по тесту што га за велики број понављања извршавања не чини погодним. Поред тога у оквиру симулатора *Gem5* укључене су многе ствари које се симулирају, а нису од интереса за предвиђање вредности. Због тога је одлучено да се *Gem5* симулатор користи како би произвео трагове извршавања тестова, док се симулација предвиђања вредности операнада обавља на основу трага извршавања у оквиру *VPSim* симулатора. Надоградња симулатора *Gem5* је детаљно описано у потпоглављу 6.3. у наставку ове докторске дисертације.

Симулатор *VPSim* је развијен да спроводи трагом вођену симулацију извршавања инструкција (енгл. *trace-driven simulation*) са предвиђањем вредности операнада. Како је циљ истраживања био да се представи феномен извршавања инструкција са непрецизним вредностима операнада развијен је овај симулатор. На слици 6.1.1. се могу видети његови главни делови, микро језгро и предиктори вредности. Микро језгро је део симулатора који садржи архитектуралне регистре архитектуре *x86* као и део који је одговоран за постављање индикатора у програмској статусној речи. Названо је микро језгро јер имплементира само неопходне делове за спровођење симулације извршавања инструкција са предвиђањем вредности. У оквиру овог симулатора имплементиран је механизам који извршава инструкције са предвиђањем вредности операнада и у оквиру њега је имплементирано укупно седам различитих предиктора вредности. Спроведеним симулацијама коришћењем *VPSim* симулатора прикупљена је статистика која је потом искоришћена како би се упоредили дефинисани аналитички модели извршавања. Имплементација и детаљи *VPSim* симулатора су приказани у потпоглављу 6.4. у наставку ове докторске дисертације.

У оквиру петог поглавља су представљени аналитички модели извршавања инструкција. Моделом 1 је моделовано извршавање без предвиђања вредности док су моделима 2 и 3 моделована извршавања инструкција са предвиђањем вредности операнада. Модел 3 за разлику од модела 2 може да настави извршавање и уколико је непрецизно предвиђена вредност операнда у ситуацији када је резултат инструкције исправан. За све моделе је дефинисано математичко очекивање које представља очекивано време извршавања одређеног броја инструкција. У тим формулама учествују неки параметри које је могуће добити емпиријски, што је учињено симулацијама у симулатору *VPSim*.

Последњи блок на слици 6.1.1. представља евалуацију дефинисаних модела извршавања. У оквиру тог дела истраживања представљене су прецизности предиктора и то прецизност тачно предвиђеног операнда и прецизност тачног резултата инструкције који је добијен на основу предвиђене вредности операнда. Такође, у том делу коначно су објашњени услови када модел 3 постиже боље очекивано време извршавања у односу на модел 2.

6.2. Коришћени тестови

Како би се истражило извршавање инструкција са непрецизно предвиђеним вредностима операнада било је неопходно посматрати извршавање неких реалних програма. У научним радовима, тј. у истраживањима која припадају архитектури рачунара доста често се користе стандардизовани тестови *SPEC* (енгл. *Standard Performance Evaluation Corporation*)[61] и *EEMBC* (енгл. *Embedded Microprocessor Benchmark Consortium*)[59], па су стога одабрани да буду коришћени у оквиру овог истраживања. У наставку овог потпоглавља следи њихов опис.

6.2.1. *SPEC* тестови

SPEC је организација која се бави формирањем и одржавањем тестова за различите рачунарске системе и његове делове. *SPEC* организација има различите скупове тестова који тестирају перформансе процесора, серверских машина или процесора специфичне намене, као што су графичке картице. Током година свог постојања, а основана је 1988. године, ова организација је дошла на водеће место организација које се баве стандардизацијом тестова рачунарских система. Интересантно је да су током израде ове докторске дисертације тестови који припадају организацији *EEMBC* постали део организације *SPEC*.

У оквиру истраживања која су спроведена током израде ове докторске дисертације коришћен је подскуп тестова који припадају тестовима *SPEC CPU2006* [62] [63]. Овај скуп тестова садржи две групе тестова *Integer Workloads* и *Floating-point Workloads*. За све тестове је доступан изворни програмски код као и скуп података које ови тестови користе као улаз. Скуп тестова *SPEC CPU2006* чине заиста постојеће апликације (програми) и у оквиру скупа нема вештачки формираних тестова, што овај скуп чини реалним.

Како за сваки тест постоји изворни програмски код, да би се тест могао извршити потребно је превести тест у облик извршиве датотеке. За ту потребу коришћен је преводилац *gcc*, а из овог скупа тестова одабрани су тестови који су написани на програмским језицима *C* и *C++* како би било могуће превести их са новијим верзијама *gcc* преводилаца (верзија 9.3+ [64]). У наставку следи кратак опис коришћених тестова из скупа *SPEC CPU2006*:

- *429.mcf* – овај тест је програм за распоређивање линија у великим системима јавног превоза за транспорт робе и људи (енгл. *single-depot vehicle-scheduling problem*);
- *458.sjeng* – овај тест је програм који представља игру шах покушавајући да пронађе најбољи потез на основу различитих алгоритама;
- *462.libquantum* – овај тест је симулатор квантног рачунара заснованог на квантној механици;
- *464.h264ref* – овај тест је програм за компресију садржаја који је у видео формату;

- *473.astar* – овај тест представља библиотеку која имплементира алгоритме претраживања заснованих на алгоритму A* (енгл. *A-star*);
- *433.milc* – овај тест представља програм који симулира интеракције између кваркова (енгл. *quark*) према теорији квантне физике;
- *470.lbm* – овај тест представља део програма за исцртавање флуида у тродимензионом простору према Болцмановом методу (енгл. *Lattice Boltzmann Method*).

6.2.2. *EEMBC* тестови

EEMBC је организација која је настала 1997. и бави се формирањем тестова хардвера и софтвера за уграђене системе (енгл. *embedded systems*). Као и *SPEC* организација, и организација *EEMBC* има више стандардизованих скупова тестова који су намењени за тестирање перформанси процесора у различитим рачунарским системима попут мобилних уређаја, аутономних возила, интернет ствари и других система.

У оквиру истраживања која су спроведена током израде ове докторске дисертације коришћен је скуп тестова који припадају тестовима *CoreMark-Pro* [65]. Овај скуп тестова служи за тестирање комплетног процесора пружајући могућност да се тестира појединачно језгро процесора или више језгара процесора. Укупно садржи девет тестова написаних на програмском језику *C* који су подељени у две групе (исто као и код *SPEC CPU2006*): *Integer Workloads* и *Floating-point Workloads*. Коришћен је исти преводилац као и за тестове *SPEC CPU2006* да се добију извршиве датотеке. У наставку је дат кратак опис тестова из скупа *CoreMark-Pro*:

- *JPEG compression* – овај тест представља програм који обавља *JPEG* (енгл. *Joint Photographic Experts Group*) компресију дигиталних фотографија;
- *ZIP compression* – овај тест представља програм који обавља *ZIP* компресију једне или више датотека формирајући на такав начин нову датотеку (архиву);
- *XML parsing* – овај тест представља програм који врши парсирање датотека које су *XML* (енгл. *Extensible Markup Language*) формата;
- *SHA-256* – овај тест представља имплементацију криптографског алгоритма *SHA* (енгл. *Secure Hash Algorithm*);
- *CoreMark* – овај тест представља комплетан тест раније верзије скупа тестова *CoreMark-Pro* под називом *CoreMark* [66];
- *Radix-2 Fast Fourier Transform* – овај тест представља имплементацију алгоритма за израчунавање дискретне Фуријеове трансформације;
- *Gaussian elimination* – овај тест представља имплементацију Гаусовог метода елиминације за решавање система линеарних једначина;
- *Neural-net* – овај тест представља имплементацију неуралне мреже;
- *Livermore loops* – овај тест представља скуп тестова који је намењен за тестирање паралелизма рачунарских система и сачињен је од 24 различитих програмских петљи [67].

6.2.3. *Резиме коришћених* тестови

У табели 6.2.3.1. дат је сажет преглед коришћених тестова у спроведеним симулацијама. За сваки тест су наведене следеће информације:

- потпун назив теста;
- скраћен назив теста који ће се користити у наставку ове докторске дисертације;
- група којој тест припада (*SPEC* или *EEMBC*);
- тип теста (*Integer* или *Floating-point*);

У табели 6.2.3.1. у одговарајућој ћелији присуство црне тачке означава да тест испуњава неку одлику. Уколико се у ћелији налази знак косе црте то значи да тест не испуњава одговарајућу одлику.

Табела 6.2.3.1. Резиме коришћених тестова

НАЗИВ ТЕСТА	СКРАЋЕН НАЗИВ ТЕСТА	ГРУПА ТЕСТА		ТИП ТЕСТА	
		SPEC	*EEMBC*	*INTEGER*	*FLOATING-POINT*
429.mcf	spec.mcf	●	/	●	/
458.sjeng	spec.sjeng	●	/	●	/
462.libquantum	spec.libquantum	●	/	●	/
464.h264ref	spec.h264ref	●	/	●	/
473.astar	spec.astar	●	/	●	/
433.milc	spec.milc	●	/	/	●
470.lbm	spec.lbm	●	/	/	●
JPEG compression	embc.cjpeg	/	●	●	/
ZIP compression	embc.zip	/	●	●	/
XML parsing	embc.parser	/	●	●	/
SHA-256	embc.sha	/	●	●	/
CoreMark	embc.core	/	●	●	/
Radix-2 FFT	embc.radix2	/	●	/	●
Gaussian elimination	embc.linear	/	●	/	●
Neural-net	embc.nnet	/	●	/	●
Livermore loops	embc.loops	/	●	/	●

6.3. Модификација *Gem5* симулатора и трагови извршавања

У оквиру овог потпоглавља биће укратко описан симулатор *Gem5*, његове основне карактеристике и начини на које се може употребити. Поред тога биће приказан и формат трагова извршавања који је генерисан коришћењем надограђеног симулатора *Gem5*. Такође, биће описана и имплементација надоградње симулатора *Gem5*.

6.3.1. Основе симулатора *Gem5*

Симулатор *Gem5* настао је 2011. године као резултат спајања два до тада изузетно коришћена симулатора у научним истраживањима *Gems* и *M5*. Симулатор *Gem5* представља симулационо окружење које дају подршку за симулирање рада процесора и меморијског система уз подршку за неколико различитих архитектура процесора као што су *x86*, *ARM* и *MIPS* [68] [69] [70]. Основна идеја овог симулатора јесте да пружи флексибилност пружајући могућност да се конфигурише ниво детаља симулације и према томе одређује ниво прецизности и брзине извршавања симулације. Такође, симулатор је отвореног кода и пружа могућности надоградње [71].

Симулатор *Gem5* је написан на језику *C++* и имплементиран је на начин да се рачунарски систем који се симулира састоји из више модула. Неки од модула су процесор, оперативна меморија и улазно-излазни уређаји. Како би се конфигурисао (саставио) рачунарски систем симулатор *Gem5* користи скрипт језик *Python*.

Симулатор подржава неколико модела процесора и неколико модела меморијског система. Подржани модели процесора су:

- *Atomic CPU* – овај модел процесора подразумева *in order* извршавање инструкција, а приступ меморији симулира на начин да се не узима потребно време за приступ оперативној меморији већ се приступ меморији завршава одмах;
- *Timing CPU* – овај модел процесора подразумева *in order* извршавање инструкција, али за разлику од *Atomic CPU* модела приступ меморији симулира детаљно заустављањем извршавања приликом промашаја у кешу и чекањем одређеног времена да се заврши приступ оперативној меморији;
- *Minor CPU* – овај модел процесора подразумева *in order* извршавање инструкција са проточном обрадом;
- *O3 CPU* – овај модел процесора подразумева *out of order* извршавање инструкција са проточном обрадом;

За потребе истраживања које је спроведено у оквиру ове докторске дисертације одабран је модел *Atomic CPU*. Овај модел пружа симулацију извршавања инструкција и симулацију оперативне меморије на начин да се приступ меморији било за читање било за упис истог тренутка завршава (нема симулирања кашњења). Таква симулација садржи довољно информација како би се формирали трагови извршавања који су улаз за *VPSim* симулатор.

Друга ствар коју је било потребно одабрати тиче се мода рада симулатора *Gem5*. Симулатор *Gem5* познаје два мода рада према томе на који начин се симулира присуство оперативног система:

- *SE (System-call Emulation) mode* – овај мод рада подразумева да се системски позиви емулирају. То значи да не постоји заиста покренут оперативни систем у симулатору, већ су основне системске услуге имплементиране у виду функција које се позивају када програм захтева системску услугу. Неке од имплементираних системских услуга су системске услуге за рад са датотекама и алокацију меморије;

- *FS (Full-System*) *mode* – овај мод рада симулатора омогућава да се покрене оперативни систем, а да се онда у оквиру оперативног система извршавају различити програми и на тај начин пружа више детаља који се могу пратити него код *SE* мода. Како је у овом моду присутан комплетан оперативни систем, могуће је покренути било који програм без измена јер су све системске услуге доступне.

У оквиру истраживања приликом израде ове докторске дисертације одлучено је да се користи *SE* мод јер је идеја да се испита колико често се извршавају погодне инструкције за непрецизно предвиђање вредности у оквиру реалних програма. Стога је довољно да се само врши емулација системских позива и на тај начин се у оквиру симулације извршавају само инструкције које припадају тестовима (програмима).

6.3.2. Трагови извршавања тестова

Када је одлучено који тип процесора (*Atomic CPU*) и који мод извршавања (*SE mode*) се користи, било је неопходно надоградити симулатор модулом чија је улога да генерише траг извршавања. Траг извршавања се током симулације формира и на крају симулације се добија текстуална датотека која садржи траг извршавања. На самом крају симулације траг извршавања који је у облику текстуалне датотеке се одмах компресује како би његова величина била мања. Оригинална величина текстуалних датотека које садрже траг извршавања је величине неколико гигабајта, док је компримована датотека величине неколико стотина мегабајта.

Тестови који су коришћени су пропуштени кроз симулатор *Gem5* како би се формирали трагови извршавања. Приликом покретања теста, симулатор извршава програмски код који је неопходан за иницијализацију система и онда прелази на извршавање главне функције (енгл. *main*). Од главне функције креће извршавање самог теста па је стога одлучено да се траг извршавања формира почевши од прве инструкције унутар главне функције.

Приликом покретања симулатора *Gem5* могуће је проследити неке аргументе који утичу на начин како се симулација одвија. За потребе овог истраживања додат је један аргумент који представља адресу инструкције од које треба започети формирање трага извршавања. На овај начин је омогућено да се од тачно неке тачке у програму формира траг извршавања. Увидом у извршиве датотеке тестова коришћењем алата *objdump* [72] могуће је утврдити адресу прве инструкције главне функције која се потом прослеђује као аргумент приликом покретања симулације у *Gem5* симулатору. За све тестове формирани су трагови извршавања који садрже сто милиона инструкција уз изузетак за неке тестове који током свог извршавања не изврше сто милиона инструкција (заврше своје извршавање са мање од сто милиона извршених инструкција).

Трагови извршавања који су формирани садрже у сваком реду запис који описује извршавање једне инструкције. У оквиру записа налазе се следеће информације:

- адреса на којој се у меморији налази инструкција тј. вредност регистра *PC* (енгл. *program counter*);
- назив инструкције који се састоји од мнемоника инструкције и ознака које говоре ког типа су операнди;

- изворишни регистри где се за сваки регистар чува његово име, ширина у битовима и вредност коју је имао на почетку извршавања посматране инструкције;
- дестинациони регистри где се за сваки регистар чува његово име, ширина у битовима и вредност коју добија након извршавања посматране инструкције;
- изворишне меморијске локације где се за сваку меморијску локацију којој се приступало ради операције читања чува адреса локације и вредност податка која је очитана са те адресе;
- дестинационе меморијске локације где се за сваку меморијску локацију којој се приступало ради операције уписа чува адреса локације и вредност податка која је уписана на ту локацију;
- непосредна вредност уколико инструкција има специфицирану непосредну вредност у оквиру саме инструкције што може бити непосредна вредност операнда или померај код инструкција скокова;
- вредност програмске статусне речи након завршетка посматране инструкције.

Треба напоменути да нема свака инструкције све описане информације у свом запису у трагу извршавања. То зависи од типа инструкције и начина адресирања којима се специфицирају операнди инструкције. На посматраној архитектури *x86* инструкције могу имати операнде означене следећим ознакама:

- *R* – овом ознаком су означени операнди који се налазе у регистрима;
- *M* – овом ознаком су означени операнди који се налазе у меморији;
- *I* – овом ознаком су означени операнди који су специфицирани непосредном вредношћу у оквиру саме инструкције.

У наставку је дат исечак трага извршавања 6.3.1. са четири инструкције које су у оквиру истраживања означене да су погодне за непрецизно извршавање инструкција. То су инструкције логичког *и* (*AND*), логичког *или* (*OR*), инструкција упоређивања (*CMP*) и инструкција тестирања (*TEST*).

```
7ffff8003097-AND_R_M-RS[64RAX,2]-RD[64RAX,2]-MS[7fffffffdfd0,6]-F[2]
...
7ffff8011f6b-OR_M_I-MS[7ffff7fed31c,1]-MD[7ffff7fed31c,11]-IMM[10]-F[6]
...
7ffff8012422-CMP_R_M-RS[32RBX,9691a75]-MS[7ffff7ea9d10,905f4e6]-F[12]
...
7ffff801881d-TEST_M_I-MS[7ffff7cc51ea,1]-IMM[1]-F[2]
```

Исечак формираног трага извршавања 6.3.2.1. Пример записа инструкција *and*, *or*, *cmp* и *test*

У оквиру исечка трага извршавања 6.3.2.1. могу се видети записи четири инструкција и то баш оних погодних инструкција за предвиђање вредности операнада. На почетку сваког записа налази се адресе инструкције, док се на крају записа налази вредност програмске статусне речи након извршене посматране инструкције и означена је са *F*. Све вредности у запису су записане у хексадецималном систему.

Први запис одговара инструкцији логичког *и*, где је један изворишни операнд регистар *RAX* што је означено делом записа *RS*. Други изворишни операнд се налази у меморији на локацији *7fffffffdfd0* и има вредност *6* што је означено делом записа *MS*. Такође, дестинациони операнд је специфициран регистром *RAX* и тај део записа је означен са *RD*. Број *64* (децимално) испред назива регистра означава ширину у битовима тог регистра који се користе пошто архитектура *x86* пружа могућност парцијалног коришћења битова регистра.

Други запис одговара инструкцији логичког *или*, где вредност једног изворишног операнда долази са једне меморијске локације, а такође та локација представља и дестинациони операнд, тј. место у меморији где се смешта резултат што је у запису означено са *MD*. Други изворишни операнд је специфициран непосредном вредношћу *6*, што је означено делом записа *IMM*.

Трећи запис одговара инструкције упоређивања *cmp*. Ова инструкција има два изворишна операнда и нема дестинациони операнд јер је њен резултат само постављање индикатора у програмској статусној речи. Један изворишни операнд је специфициран регистром, док се вредност другог изворишног операнда налази у меморији.

Четврти запис одговара инструкцији тестирања *test* и исто као и претходна инструкција нема дестинациони операнд већ само два изворишна. Први изворишни операнд је одређен меморијском локацијом, тј. налази се у меморији, а други је непосредна вредност која је у запису означена са *IMM.*

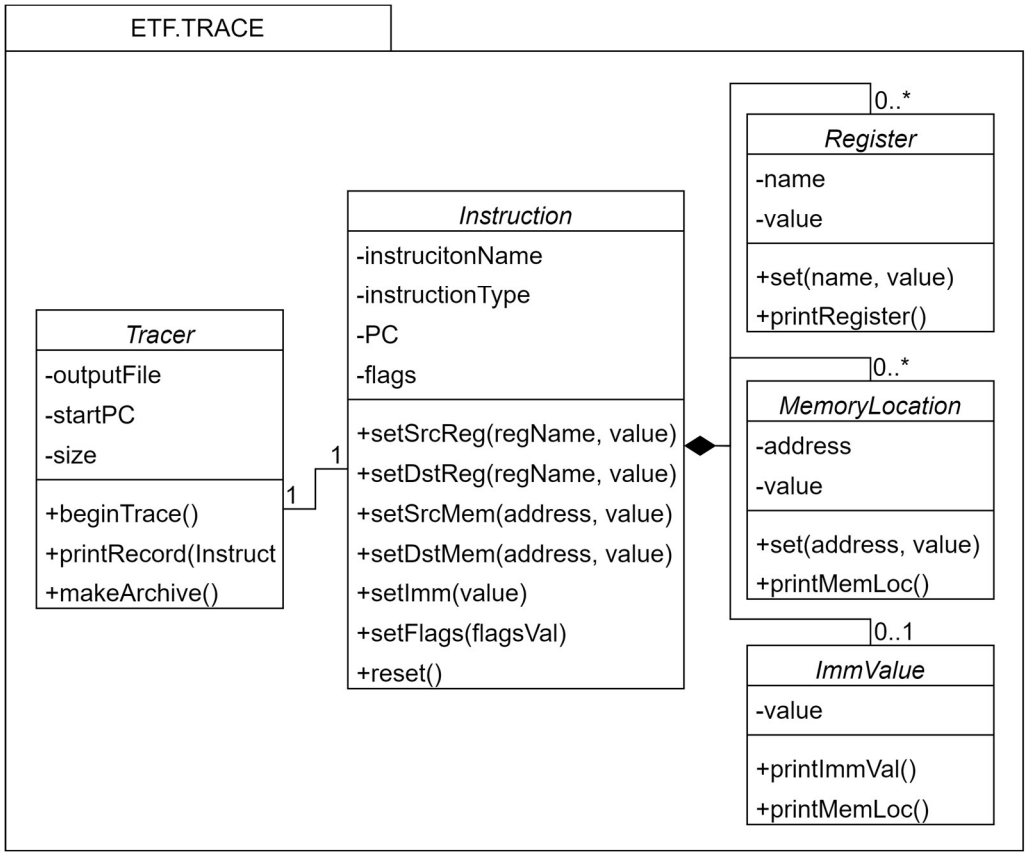
6.3.3. Имплементација надоградње симулатора *Gem5*

Како би било могуће формирати описани траг извршавања било је потребно имплементирати нови модул у оквиру симулатора *Gem5*. Као што је већ поменуто симулатор *Gem5* је имплементиран коришћењем програмског језика *C++* па је и надоградња имплементирана на истом језику за потребе истраживања у оквиру ове докторске дисертације. На слици 6.3.3.1. приказан је упрошћен дијаграм најбитнијих класа које су реализоване за потребе генерисања трагова извршавања у оквиру пакета (модула) *ETF.TRACE*.

Класа *Trace* је одговорна за креирање трага извршавања. У току извршавања симулације ова класа формира траг извршавања уписивањем записа инструкција у излазну датотеку *outputFile*. Ова класа садржи и информацију од које инструкције, тј. на којој адреси се налази инструкција од које треба започети формирање трага извршавања као и максималан број инструкција које може траг садржати. Такође, одговорност ове класе јесте и формирање компримоване излазне датотеке како би се потребан простор за чување трага извршавања смањио. Током извршавања симулације објекат класе *Trace* приступа једном објекту класе *Instruciton* који чува информације о инструкцији чије се извршавање тренутно симулира. Када се заврши симулација њеног извршавања тада објекат класе *Trace* уписује информације из објекта класе *Instruction* у излазну датотеку.

Класа *Instruction* је одговорна за прикупљање информације о инструкцији чије се извршавање тренутно симулира. У оквиру објекта ове класе се налазе све информације које се после уписују у излазну датотеку и које чине запис о извршеној инструкцији. За инструкцију се чува њено име (мнемоник), адреса на којој се налази, тип инструкције, вредност програмске статусне речи након завршетка инструкције као и сви њени операнди. Постоје три типа операнада према томе где се налази њихова вредност. Вредност операнада може бити у регистру, меморији или може бити записана у самој инструкцији. Такође,

операнди су изворишни или дестинациони у зависности на који начин их инструкција користи.



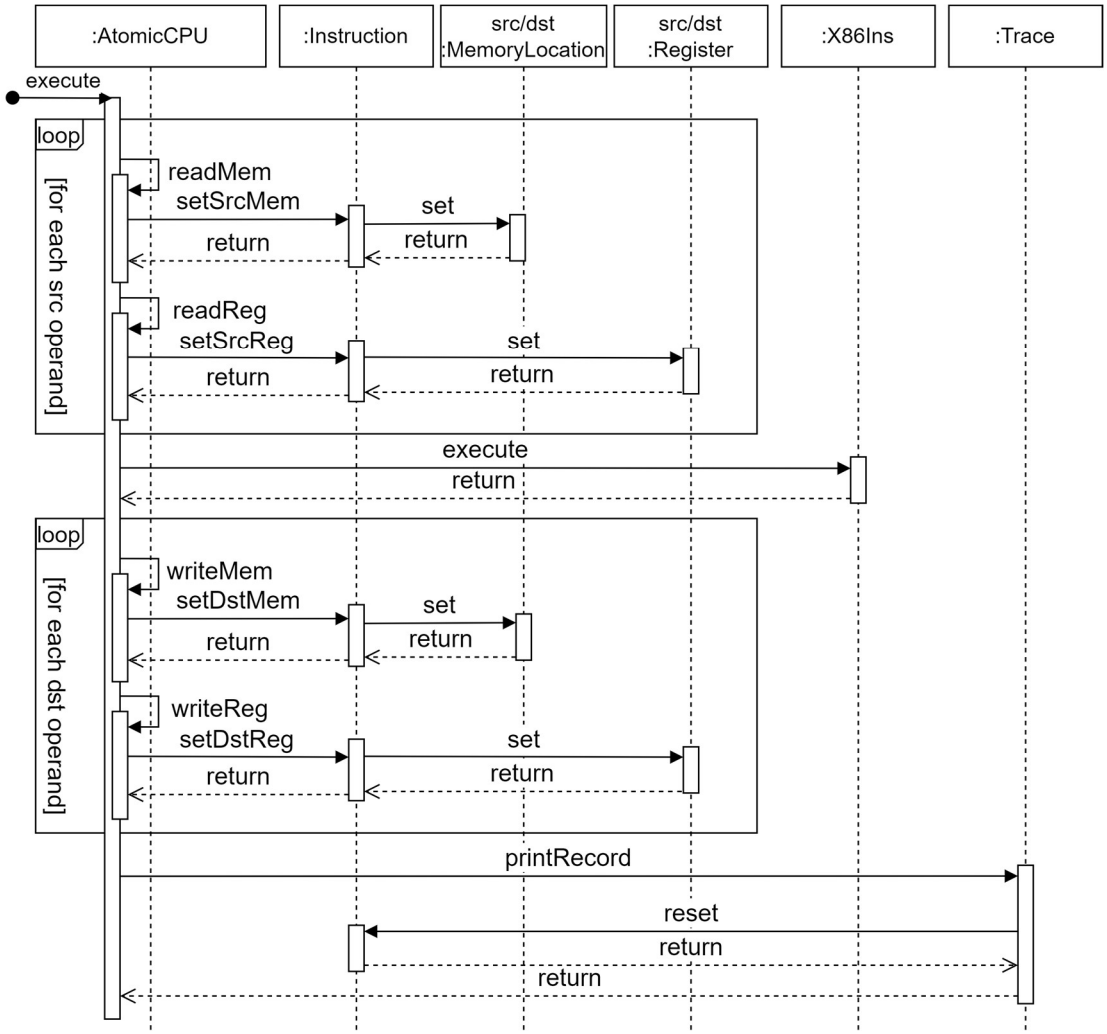
Слика 6.3.3.1. Класни дијаграм надоградње симулатора *Gem5*

За сваки тип изворишног и дестинационог операнада постоји метода која служи да сачува вредност коришћеног операнда. За регистарске операнде чува се назив регистра и његова вредност која је искоришћена. Уколико се ради о регистарском изворишном операнду тада се чува вредност која је прочитана из тог регистра, док ако се ради о дестинационом регистарском операнду тада се чува вредност која је уписана у регистар у току извршавања посматране инструкције. Операнди који су у меморији, за њих се чува вредност и адреса у меморији где се налази та вредност. Уколико се ради о изворишном меморијском операнду тада се чува вредност која је прочитана са те локације, док ако се ради о дестинационом меморијском операнду тада се чува вредност која је уписана на ту локацију у току извршавања посматране инструкције.

Током извршавања симулације постоји само по једна инстанца класа *Trace* и *Instruction*. Инстанца класе *Trace* креира се пре почетка симулације извршавања инструкција и живи до краја завршетка симулације. Инстанца класе *Instruction* креира се такође пре почетка симулације извршавања инструкција. Та инстанца служи да чува информације о тренутној инструкцији чије се извршавање симулира. Када се симулација извршавања једне инструкције заврши тада се стање те инстанце ресетује чиме иста инстанца постаје спремна да прикупља информације о наредној инструкцији.

Информације о регистарским операндима се чувају у оквиру објеката класе *Register*, док се у оквиру објеката класе *MemoryLocation* чувају информације о меморијским операндима. Објекти класе *ImmValue* служе за чување вредности операнда чија је вредност

записана у самој инструкцији (непосредни операнд). Како инструкције на архитектури *x86* могу користити само један непосредни операнд довољно је креирати само један објекат класе *ImmValue* у току симулације извршавања инструкције. Ове три класе које описују операнде су опремљене и методама за испис операнада у формату који је представљен у претходном потпоглављу.



Слика 6.3.3.2. Дијаграм секвенце прикупљања информација о извршеној инструкцији на симулатору *Gem5*

На слици 6.3.3.2. приказан је дијаграм секвенце на коме је представљено прикупљање информација у току извршавања о једној инструкцији. Назив (мнемоник) инструкције, тип инструкције и адреса на којој се инструкција налази су доступни у оквиру симулатора *Gem5* пре почетка симулације извршавања инструкције. Те три информације су сачуване у оквиру класе *Instruction* и тај део није приказан на дијаграму секвенце. На дијаграму секвенце је приказан део извршавања симулације који се односи само на извршавање инструкције. У оквиру *Gem5* симулатора извршавање сваке инструкције се одвија у методи *execute* класе *AtomicCPU* јер је коришћен *Atomic CPU* модел процесора. Позивом поменуте методе започиње симулација извршавања инструкције.

У првом делу дијаграма врши се дохватање свих потребних вредности операнада, а то могу бити вредности из меморије, регистара или непосредна вредност из самог записа инструкције. На дијаграму није приказано уколико инструкција користи непосредну

вредност, већ само ако користи вредности из регистара или меморије. Позивом одговарајуће методе класе *AtomicCPU* за читање вредности из меморије *readMem* приступа се меморији. У оквиру те методе подметнут је позив методе *setSrcMem* класе *Instruction* чији је задатак да покупи вредност која је очитана као и адресу са које је очитана вредност. Те информације се потом смештају у објекат класе *MemoryLocation*. Слично се приступа и вредностима регистара. Позивом методе *readReg* класе *AtomicCPU* приступа се траженом регистру. У оквиру те методе подметнут је позив методе *setSrcReg* класе *Instruction* чији је задатак да покупи вредност која је очитана као и назив регистра чија је вредност очитана. Описани поступак којим се прикупљају изворишни операнди се понавља за сваки изворишни операнд, па је то на дијаграму представљено постојањем петље.

Након што се заврши дохватање вредности свих операнада, тада се приступа извршавању саме инструкције. То се обавља позивом методе *execute* која припада класи *X86Ins*. У симулатору *Gem5* је овде тек тачка где се долази до специфичног дела који одговара одређеној архитектури. Симулатор има посебан развијен језик (енгл. *ISA description language*) за опис архитектуре процесора на основу кога се потом генеришу класе на језику *C++* које су одговорне за извршавање инструкција према специфичној архитектури. Оваква имплементација симулатора омогућава да се остали нивои апстракције у њему мењају независно од специфичне архитектуре.

Након што се заврши извршавање саме инструкције према архитектури која се користи, прелази се на део симулације који је одговоран за прикупљање вредности дестинационих операнада. Слично као и приликом прикупљања вредности изворишних операнада, додати су позиви метода у оквиру постојећих метода које служе за приступ регистрима и меморијским локацијама ради операција уписа. Позивом методе *writeMem* класе *AtomicCPU* врши се упис вредности на неку меморијску локацију. У оквиру ове методе подметнут је позив методе *setDstMem* класе *Instruction* чиме се бележи адреса меморијске локације и вредност која је уписана на ту локацију. Ове информације се чувају у оквиру објекта класе *MemoryLocation*. Уколико инструкција мења вредности регистара, тада се позива метода *writeReg* класе *AtomicCPU*. У оквиру ове методе подметнут је позив методе *setDstReg* класе *Instruciton* чиме се бележи назив регистра и вредност која је уписана у њега. Ове информације се чувају у оквиру објекта класе *Register*. Такође, исто као и за прикупљање информација о изворишним операндима, поступак којим се прикупљају дестинациони операнди понавља се за сваки дестинациони операнд, што је на дијаграму представљено постојањем петље.

У оквиру методе *execute* класе *AtomicCPU* на крају се још врши уписивање прикупљених информација о извршавању инструкције у излазну датотеку. Позива се метода *printRecord* која запис о инструкцији уписује у излазну датотеку која представља траг извршавања. У оквиру ове методе се врши и припрема за праћење извршавања наредне инструкције. Позива се метода *reset* класе *Instruction* која чисти објекат класе *Instruciton* од прикупљених информација инструкције која је управо уписана у излазну датотеку.

6.4. Имплементација *VPSim* симулатора

Као што је већ поменуто за истраживање извршавања инструкција са непрецизно предвиђеним операндима имплементиран је у оквиру овог истраживања симулатор *VPSim* (енгл. *Value Prediction Simulator*). На овај корак се прешло из разлога што симулације у оквиру симулатора *Gem5* трају много времена (неколико сати по тесту) и симулирају се детаљи који нису били од интереса за истраживања у оквиру ове докторске дисертације.

VPSim је трагом вођен симулатор (енгл. *trace-driven simulation*) попут симулатора који је аутор ове докторске дисертације представио у свом раду [73]. Трагом вођена симулација јесте симулација која као улаз користи датотеку са трагом извршавања програма. У оквиру такве датотеке се налазе записи о инструкцијама које симулатор чита и на основу њих спроводи симулацију. Предност оваквих симулације јесте што је могуће ставити акценат на одређени део процесора, што доводи до значајног бржег времена извршавања него у ситуацијама када се симулирају сви делови процесора. Уколико се посматрају потенцијални проблеми оваквих симулација може се поменути величина датотека која садржи трагове извршавања. Те датотеке могу захтевати доста меморијског простора за складиштење, али и оперативне меморије током извршавања јер је потребно читати такве датотеке.

Симулатор *VPSim* је комплетно имплементиран у *Java* програмском језику. Покреће се као алат из командне линије коме се прослеђује једна конфигурациона текстуална датотека. У тој датотеци се наводи предиктор вредности који ће се користити, параметри за његову конфигурацију као и путање до датотека које садрже трагове извршавања. Излаз симулатора представљају текстуалне датотеке за сваки траг извршавања по једна излазна датотека. Оне садрже прикупљене информације о извршеној симулацији над одговарајућим трагом извршавања.

У оквиру *VPSim* симулатора је имплементирано укупно седам различитих предиктора вредности. Одлучено је да се имплементира четири опште позната предиктора из литературе и три хибридна предиктора са светског такмичења у предвиђању вредности. Имплеметирани предиктори су (за прва три наведена предиктора у оквиру заграда наведене су скраћенице које ће се користити у наставку):

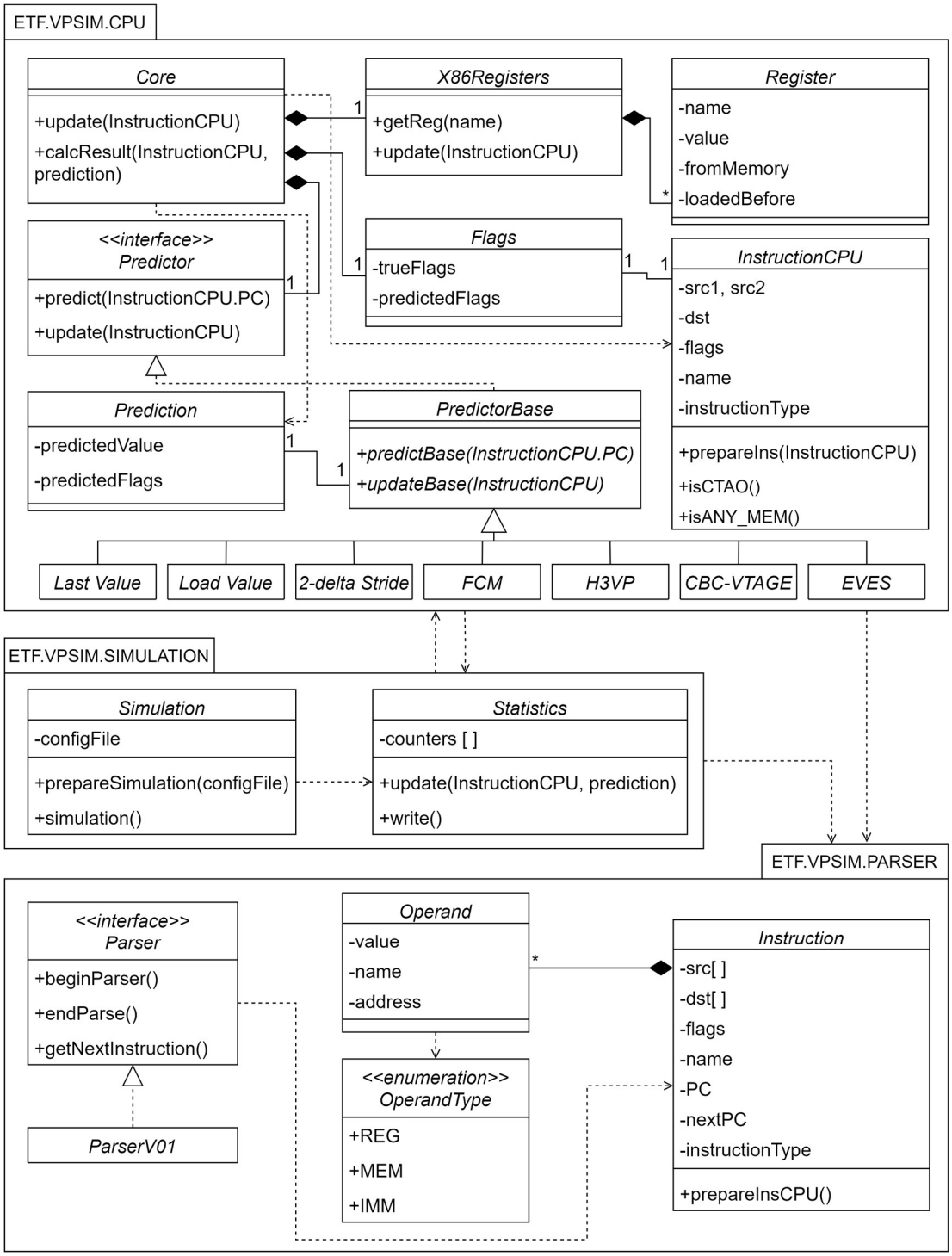
- *Last Value* (LAVP) предиктор;
- *Load Value* (LOVP) предиктор;
- *2-delta Stride* (TDS) предиктор;
- *FCM* предиктор;
- *EVES* предиктор;
- *CBC-VTAGE* предиктор;
- *H3VP* предиктор.

Прва четири набројана предиктора сваки пут када наиђе погодна инструкција за предвиђање вредности операнда врше предвиђање. Разлог за то је што ови предиктори немају механизам поверења и предвиђање врше сваки пут када се појави погодна инструкција. Они су одабрани јер се најчешће срећу у литератури. Последња три набројана предиктора су хибридни предиктори који су постигли најбоље резултате на светском такмичењу у предвиђању вредности. Имају знатно већу прецизност од прва четири предиктора, али знатно смањену покривеност.

На слици 6.4.1. је приказан упрошћени класни дијаграм имплементације симулатора *VPSim*. На дијаграму су представљене најбитније класе са својим најбитнијим атрибутима и методама. Симулатор је имплементиран кроз три пакета која се могу уочити на дијаграму:

- *ETF.VPSIM.CPU* – овај пакет садржи класе које имплементирају ствари везане за само језгро процесора и предикторе;

- *ETF.VPSIM.SIMULATION* – овај пакет је одговоран за спровођење симулације и прикупљање статистике о извршавању;
- *ETF.VPSIM.PARSER* – овај пакет је одговоран за читање улазне датотеке које садржи траг извршавања и формирање објеката инструкције који се касније прослеђују до језгра процесора.



Слика 6.4.1. Класни дијаграм симулатора *VPSim*

У оквиру пакета *ETF.VPSIM.CPU* постоји класа *Core* која представља језгро процесора. Садржи архитектуралне регистре архитектуре *x86*, предиктор вредности који користи у току симулације извршавања инструкција и индикаторе програмске статусне речи. Једна од две најбитније методе ове класе је метода којом се врши ажурирање процесора на основу извршавања једне инструкције. Након сваке инструкције врши се ажурирање архитектуралних регистара као и предиктора. Друга метода је одговорна за рачунање резултата инструкције и индикатора у оквиру програмске статусне речи на основу предвиђене вредности операнда инструкције. Ова метода се користи у ситуацији када на извршавање дође погодна инструкција за предвиђање вредности операнда. На овај начин се добија вредност индикатора на основу резултата инструкције који је добијен на основу предвиђене вредности операнда. Самим тим овако израчунати индикатори јесу спекулативни јер су добијени на основу спекулативно извршене инструкције. Ово је неопходно како би се касније у току симулације утврдило да ли су спекулативно израчунати индикатори једнаки индикаторима који су добијени на основу тачног резултата инструкције.

Класа *X86Registers* садржи објекте класе *Register* и представља архитектуралне регистре процесора. За сваки регистар чува се његово име, вредност и информација да ли из меморије потиче његова тренутна вредност. Уколико његова вредност потиче из меморије тада је валидан и атрибут који говори колико циклуса (инструкција) је прошло од тренутка када је постављена та вредност из меморије. Након сваке извршене инструкције ажурирају се вредности регистара на основу вредности дестинационих регистара извршене инструкције.

Класа *Flags* представља имплементацију индикатора програмске статусне речи. Садржи тачну вредност индикатора као и вредност индикатора која је добијена на основу резултата инструкције за коју се вршило предвиђање вредности операнда. Такође, садржи и информацију која је инструкција последња поставила вредност индикатора.

Објектима класе *InstructionCPU* имплементиране су инструкције које су формиране на основу записа из трага извршавања. Ови објекти су прилагођени за рад са објектом класе *Core* и настају на основу објеката класе *Instruction* из пакета *ETF.VPSIM.PARSER*. Они садрже изворишне операнде као и дестинациони операнд који су неопходни да се изврши операција неке инструкције (*dst = src1 operation src2*). Поред тога садржи назив инструкције као и ког је типа инструкција (аритметичко-логичка или инструкција скока). Објекти ове класе садрже још и вредност индикатора програмске статусне речи након извршене посматране инструкције. Такође, ова класа има и две помоћне методе које утврђују да ли је инструкција погодна за предвиђање. Једном методом (*isCTAO* метода) се утврђује да ли је инструкција нека од инструкција упоређивања, тестирања, логичког *и* или логичког *или*, а другом методом (*isANY_MEM* метода) се утврђује да ли је један операнд типа *ANY_MEM* што га чини погодним за предвиђање његове вредности.

Имплементација предиктора је урађена на начин да постоји интерфејс *Predictor* који има две методе. Прва метода је одговорна за предвиђање вредности, док друга метода служи да се изврши ажурирање предиктора вредности. Овај интерфејс је потом имплементиран у виду апстрактне класе *PredictorBase* која заправо имплементира претходно поменуте две методе на начин да постану шаблонске методе по узору на пројектни узорак шаблонски метод (енгл. *template method*). У оквиру тих шаблонских метода врши се препознавање типа инструкције и њених операнада што је потребно урадити без обзира на конкретан предиктор вредности. Пошто је идеја да се предвиђање врши само за погодне инструкције са одговарајућим типовима операнада (*ANY_MEM* тип) те провере су спроведене у оквиру шаблонских метода. Унутар њих су постављени позиви апстрактних метода *predictBase* и

updateBase које треба да имплементира конкретан предиктор, а које врше предвиђање вредности и ажурирање предиктора, респективно.

Конкретни предиктори су изведени из апстрактне класе *PredictorBase*. На дијаграму су приказани само називи класа које представљају имплементацију одговарајућих предиктора. Изостављене су помоћне методе као и атрибути ових класа због поједностављења самог дијаграма. Такође, имплементација сложенијих предиктора укључује више различитих класа које такође нису приказане на дијаграму. Објекти класе *Prediction* представљају заправо резултат предвиђања. У оквиру њих се чува предвиђена вредност као и индикатори који су добијени на основу спекулативног резултата инструкције, а који је добијен на основу предвиђене вредности операнда.

Пакет *ETF.VPSIM.SIMULATION* садржи две класе чији објекти су одговорни за управљање симулацијом као и за прикупљање статистике извршавања. Класа *Simulation* спроводи трагом вођену симулацију. Она је одговорна за покретање симулације, тј. за конфигурисање процесорског језгра задатим предиктором. Такође, она руководи симулацијом извршавајући све трагове извршавања који су задати конфигурационом датотеком. За сваки траг извршавања се формира излазна датотека са статистиком извршавања. Током симулације објекат класе *Statistics* је задужен за праћење извршавања инструкција као и предвиђања вредности која се дешавају. У оквиру ове класе постоје више параметара који су праћени, тј. више параметара је бројано што је на дијаграму означено атрибутом *counters*. Након сваке извршене инструкције врши се ажурирање статистике. Ажурирање се обавља на основу извршене инструкције и предвиђања уколико га је било. Такође, објекат класе *Statistics* је задужен и за формирање излазних текстуалних датотека које садрже резултате симулације.

Последњи пакет класа са слике 6.4.1. је пакет *ETF.VPSIM.PARSER*. Овај пакет је задужен за само читање трага извршавања као и формирање инструкција на основу прочитаног записа инструкције из трага извршавања. Објекат класе *ParserV01* користи се за читање улазних датотека које садрже траг извршавања. На основу сваког прочитаног реда, објекат класе *ParserV01* формира инструкцију у виду објекта класе *Instruction*. У оквиру ове класе имплементирани су регуларни изрази који служе за парсирање једног реда (записа) из улазне датотеке чији је формат приказан на слици 6.3.1. У оквиру класе *Instruction* чувају се све прочитане информације о инструкцији из трага извршавања. Такође, ова класа је одговорна и за формирање класе *InstructionCPU* која је прилагођена за рад са језгром процесора. Објекти класе *Instruction* имају листе свих операнада, изворишних и одредишних. Операнди могу бити у регистру или у меморији, а такође могу бити специфицирани непосредном вредношћу. Као што је поменуто за сваки операнд се чува вредност, а у зависности од типа операнда чува се још и име уколико се ради о регистру или адреса меморијске локације уколико се ради о операнду из меморије.

6.5. Симулације

У оквиру овог потпоглавља биће дат опис извршавања симулације која је имплементирана у оквиру симулатора *VPSim.* Биће представљен псеудо код симулације који истиче њене најбитније делове. Такође, ово потпоглавље садржи и опис излаза саме симулације као и начин на који се користи имплементирани симулатор *VPSim*.

6.5.1. Опис извршавања симулације

Након формирања трагова извршавања коришћењем симулатора *Gem5*, формирани трагови извршавања коришћени су као улаз за симулатор *VPSim*. Симулатор *VPSim* спроводи трагом вођену симулацију и прикупља статистику. Програмски исечак 6.5.1.1. представља псеудо код извршавања инструкција у оквиру симулатора *VPSim*.

На почетку извршавања симулатора *VPSim* креира се објекат класе *Simulation*. Као што је поменуто овај објекат је одговоран за креирање свих других објеката који су неопходни за извођење симулације. Приликом креирања објекта класе *Simulation* врши се очитавање улазне конфигурационе датотеке. На основу прочитаних информације из конфигурационе датотеке формирају се сви неопходни објекти за извођење симулације, а то су парсер, језгро процесора, предиктор и објекат који прикупља резултате симулације.

Приликом креирања предиктора, позива се одговарајући конструктор конкретног предиктора који је задат унутар конфигурационе датотеке. Поред имена предиктора, један ред у оквиру конфигурационе датотеке садржи информације за креирање задатог предиктора. На основу тих информација у оквиру конструктора предиктора врши се креирање осталих објеката који чине задати предиктор.

Симулација започиње извршавањем инструкција из првог задатог трага извршавања. Чита се ред по ред, где се након прочитаног реда (записа инструкције) формира објекат инструкције који се користи у наставку симулације. За формирану инструкцију проверава се да ли је погодна инструкција за предвиђање вредности операнда. Инструкција је погодна уколико су испуњена два услова. Први услов јесте да је инструкција једна од четири инструкција: упоређивања (енгл. *cmp*), тестирања (енгл. *test*), логичког *и* (енгл. *and*) или логичког *или* (енгл. *or*). Овај услов се проверава позивом методе *isCTAO*. Други услов који треба бити испуњен јесте да инструкција има један операнд који је типа *ANY_MEM* (овај тип операнда је описан у потпоглављу 4.3.). Овај услов се проверава позивом методе *isANY_MEM*. У случају да су оба услова испуњена тада предиктор врши предвиђање вредности за операнд који је типа *ANY_MEM*.

Предиктор врши предвиђање вредности на основу свог интерног стања (информација које су прикупљене током извршавања) и на основу информација које има о самој инструкцији. Према литератури у којој су описани постојећи предиктори вредности, информација о инструкцији која се користи као улаз предиктора приликом предвиђања јесте адреса те инструкције. У оквиру симулација које су спроведене у оквиру ове докторске дисертације, предикторима су као улаз прослеђиване адресе инструкција. Поред тога, као улаз предиктора коришћене су и неке друге информације које су изведене из информација о инструкцији о чему ће бити више речи у наредном поглављу.

Након што се изврши предвиђање вредности операнда, потребно је инструкцију извршити на језгру са предвиђеном вредношћу операнда, што је представљено методом *caclResutl* у псеудо коду приказаном у оквиру програмског исечка 6.5.1.1. Изводи се операција над познатим операндом и операндом чија је вредност предвиђена. На тај начин се добија резултат на основу предвиђене вредности операнда. Поред тога што се спроводи операција над операндима, неопходно је још одредити и индикаторе у програмској статусној речи. Тако постављени индикатори су постављени на основу резултата инструкције који је добијен на основу предвиђене вредности операнда. На овај начин се формира спекулативан резултат инструкције на основу предвиђене вредности операнда. Ово је битан део симулације јер се спекулативан резултат касније упоређује са исправним резултатом ради прикупљања статистике.

```
simulation = Simulation(configFile)
begin
    parser = ParserV01();
    core = Core();
    predictor = Predictor();
    stats = Statistics();
end;

while(parser.hasTrace()) do
    parser.beginParse();
    while(parser.hasInstruction()) do

          instruction = parser.getNextInstruction().prepareInsCPU();
          prediction = predictor.predict(instruction.pc)
          begin
                if(instruction.isCTAO() and instruction.isANY_MEM())
                then
                      return predictBase(instruction.PC);
                else
                      return null;
                end if;
          end;

          if(prediction != null)
          then
                core.calcResult(instruction, prediction)
                begin
                 execute the instruction using predicted and known operand;
                 determine flags according to the instruction's resutl;
                end;
          end if;

          predictor.update(instruction);
          core.update();
          stats.update();

    end while;
    parser.endParse();
    stats.write();
end while;
```

Програмски исечак 6.5.1.1. Псеудо код симулације у оквиру симулатора *VPSim*

Након предвиђања вредности, које може да се одради или не у зависности од тога да ли је инструкција погодна, прелази се на део симулације који се за сваку инструкцију извршава. У оквиру тог дела прво се врши ажурирање предиктора вредности. Предиктор вредности на овај начин се током извршавања обучава за наредна предвиђања ажурирањем свог интерног стања.

Наредни корак у симулацији односи се на ажурирање стања језгра. У оквиру ажурирања стања језгра, на основу инструкције врши се ажурирање архитектуралних регистара. Тада се поставља нова вредност регистара чију вредност је посматрана инструкција променила. Такође, води се још и евиденција да ли нова вредност у регистру потиче из меморије. За све регистре чија је вредност дошла из меморије прати се још и колико циклуса (инструкција) је протекло од тренутка када је вредност дошла у регистар. Ове информације су неопходне како би се могло препознати да ли је операнд неке

инструкције *R_MEM* типа (операнд је специфициран регистром чија је вредност претходно дошла из меморије, описано у потпоглављу 4.3.).

Последњи корак у оквиру извршавања једне инструкције јесте ажурирање објекта који води евиденцију о статистици. На основу инструкције која се обрађивала у посматраној итерацији као и на основу предвиђања врши се ажурирања статистике. У оквиру објекта статистике броје се различите ствари везане за извршавање инструкција. Неке од најбитнијих ствари које су се пратиле јесу:

- број извршених инструкција;
- број извршених инструкција упоређивања, тестирања, логичког *и* и логичког *или*;
- број извршених погодних инструкција за предвиђање вредности операнада – инструкције упоређивања, тестирања, логичког *и* и логичког *или*;
- број погодних инструкција за које се вршило предвиђање;
- број тачно предвиђених вредности операнада;
- број тачно добијених резултата инструкција на основу предвиђених вредности операнада;
- број тачно добијених вредности програмске статусне речи на основу предвиђених вредности операнада;
- број пута коришћења одређених индикатора програмске статусне речи;
- број пута постављања одређених индикатора од стране погодних инструкција за које се вршило предвиђање;
- број пута тачно постављених одређених индикатора од стране погодних инструкција за које се вршило предвиђање;
- број погодних инструкција за предвиђање вредности операнада са *R_MEM* типом операнда и са *T_MEM* типом операнда;
- број предвиђања вредности операнада који су *R_MEM* типа и *T_MEM* типа;
- број тачно предвиђених вредности операнада који су *R_MEM* типа и *T_MEM* типа.

Као што је поменуто у претходном потпоглављу, у оквиру симулатора *VPSim* прати се тачна вредност индикатора у оквиру програмске статусне речи, али се прати и спекулативна вредност индикатора који су постављени од стране неке спекулативно извршене инструкције (инструкције за коју се вршило предвиђање вредности операнада). Тачна вредност индикатора се добија из трага извршавања, док се спекулативна вредност индикатора формира на основу резултата инструкције који је добијен на основу предвиђене вредности операнда. Поред тога што се одређује спекулативна вредност индикатора, додатно се памти и која је инструкција поставила индикаторе. Када на извршавање дође инструкција која користи индикаторе, а то су најчешће условне инструкције, тада се прво проверава да ли су индикатори спекулативно одређени. Уколико то јесте случај тада се одређује услов на основу праве и на основу спекулативне вредности индикатора. Ово се све обавља у симулацији у оквиру ажурирања статистике. На овај начин се у оквиру симулације, тј. у статистици бележе три ситуације које омогућавају коректно извршавање програма:

- прва ситуација јесте у случају да је спекулативна вредност програмске статусне речи идентична тачној вредности индикатора, то значи да су сви спекулативни индикатори једнаки тачним индикаторима. До овога долази уколико је вредност операнда инструкције на основу чијег резултата су постављени индикатори потпуно тачно предвиђена или делимично тачно предвиђена;
- друга ситуације јесте да су спекулативни индикатори, који се користе за одређивање услова условне инструкције, добили исправне вредности. До овога долази уколико је вредност операнда инструкције на основу чијег резултата су постављени индикатори делимично тачно предвиђена. Треба напоменути да нису сви индикатори можда искоришћени и онда вредности оних индикатора који нису коришћени за одређивање услова условне инструкције не морају бити тачно постављене;
- трећа ситуација јесте да нису сви спекулативни индикатори који се користе за одређивања услова условне инструкције исправно постављени, али је функција којом се одређује услов дала исти резултат као и када се примени над тачним индикаторима. Једино битно за коректно извршавање условне инструкције јесте да је резултат услова исправно одређен.

Када се изврше све инструкције из једног трага извршавања, тада симулатор формира излазну датотеку у којој се налазе резултати извршавања. Након тога се прелази на наредни траг извршавања чија је путања задата у оквиру конфигурационе датотеке. Поново се улази у унутрашњу петљу и креће се у извршавање инструкција новог трага извршавања.

Описана симулација псеудо кодом који дат у оквиру програмског исечка 6.5.1.1. има за циљ да прати извршавање инструкција са предвиђањем вредности операнада без улажења у детаље организације и архитектуре конкретног процесора. На крају описа симулације коју спроводи имплементирани симулатор *VPSim* могу се издвојити његова најбитнија својства у циљу прикупљања статистике како би се приказао феномен коректног извршавања програма са непрецизно предвиђеним вредностима операнада:

- симулатор бележи ситуације у којима је и непрецизно предвиђен операнд прихватљив уколико је резултат инструкције исправан или искоришћени део резултата исправан;
- симулатор води евиденцију о томе да ли је неки операнд који је специфициран регистром заправо *R_MEM* типа, тј. да ли вредност регистра потиче из меморије и колико циклуса раније је дошла у регистар из меморије;
- симулатор уводи још једну инстанцу програмске статусне речи у којој се чува вредност индикатора који су постављени на основу резултата спекулативне инструкције која је извршена са предвиђеном вредношћу операнда.

6.5.2. Провера исправности и покретање симулације

i) Исправност симулације

На крају описа симулације треба прокоментарисати и начин на који је утврђена исправност симулације. Исправност симулације се утврдила тако што су се погодне инструкције за предвиђање вредности операнада у оквиру *VPSim* извршавале и без предвиђања. Коришћене су тачне вредности операнада из трага извршавања и са тим вредностима извршене су инструкције. Тако добијен резултат се упоредио са резултатом који

је записан у оквиру трага извршавања, а који је добијен извршавањем у оквиру симулатора *Gem5*. Како је ту долазило до поклапања резултата (вредност дестинационог операнда и индикатора у програмској статусној речи) добијених коришћењем *VPSim* симулатора и резултата добијених коришћењем *Gem5* симулатора, утврђена је исправност симулације.

ii) Покретање симулације

Као што је поменуто симулатор *VPSim* је имплементиран у програмском језику *Java*. Симулатор је запакован у виду *JAR* (енгл. *Java Archive*) датотеке који се покреће из команде линије уз задавање путање до конфигурационе датотеке. На овај начин је могуће покренути више симулација истовремено из различитих командних линија. Ово омогућава да се симулације спроводе за више различитих предиктора истовремено покретајући истовремено више симулација са различитим конфигурационим датотекама. Треба напоменути да рачунар на коме се покрећу симулације мора имати довољно расположиве оперативне меморије јер симулатор ради са великим датотекама које садрже трагове извршавања, као и одговарајућу *JVM* (енгл. *Java Virtual Machine*) која подржава рад са датотекама већим од четири гигабајта. Током извршавања симулације у командној линији се исписује који траг извршавања се тренутно обрађује и на сваких десет милиона инструкција се исписује порука колико је укупно инструкција обрађено. Као што је речено трагови извршавања садрже до сто милиона инструкција.

```
etf.vpsim.cpu.LoadValuePredictor
ANY_MEM
CTAO
1024 1024 1
END_PREDICTOR_CONFIG
\VPCPU\Traces\cjpeg-100M.zip
\VPCPU\Traces\core-100M.zip
...
```

Исечак конфигурационе датотеке 6.5.2.1. Део конфигурационе датотеке симулатора *VPSim* за предиктор *Load Value*

Исечак конфигурационе датотеке 6.5.2.1. приказује део конфигурационе датотеке за покретање симулације. У оквиру датотеке у првом реду се налази име предиктора који ће бити коришћен. У другом реду се наводи тип операнда за који ће се вршити предвиђање вредности. У трећем реду се налази ознака погодних инструкција за које ће се вршити предвиђање вредности операнада. У четвртом реду се налазе аргументи којим се конструише задати предиктор у првом реду. Аргументи су међусобно раздвојени знаком белине. Након тога један ред садржи натпис *END_PREDICTOR_CONFIG* који означава да се у наредним редовима налазе путање до датотека које садрже трагове извршавања.

7. РЕЗУЛТАТИ

У оквиру овог поглавља биће представљени добијени резултати у спроведеним експериментима током истраживања у оквиру израде ове докторске дисертације. Прво ће бити представљена почетна запажања и анализа вредности операнада погодних инструкција за предвиђање. Поред тога биће представљена анализа учесталости погодних инструкција за предвиђање вредности у коришћеним тестовима. Након тога биће представљени резултати симулације на основу којих ће се извести закључци о томе да ли постављене хипотезе у оквиру потпоглавља 4.4. важе. Такође, биће детаљно коментарисани добијени резултати извршених симулација. Део резултата који ће бити представљен у оквиру овог поглавља аутор ове докторске дисертације је представио у свом раду [19].

7.1. Почетна анализа

У оквиру истраживања након што су креирани трагови извршавања приступило се анализи добијених трагова извршавања са циљем да се утврди учесталост погодних инструкција за предвиђање вредности. Трагови извршавања као што је већ описано су формирани извршавањем тестова на архитектури *x86*. Још једном треба поновити инструкције које су у оквиру овог истраживања одређене као погодне за предвиђање:

- инструкција упоређивања вредности два операнда – *CMP*;
- инструкција тестирања вредности два операнда – *TEST*;
- инструкција логичког *и* – AND;
- инструкција логичког *или* – *OR*.

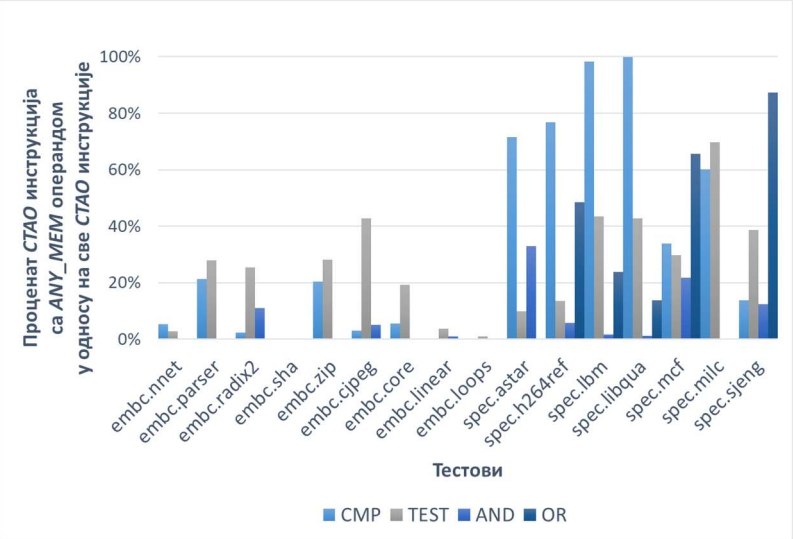
Набројане инструкције у наставку ове докторске дисертације скраћено ће се називати *CTAO* инструкције. *CTAO* инструкције су погодне уколико је један од њених операнада *ANY_MEM* типа. То значи да је операнд специфициран неким меморијским адресирањем или регистром, али вредност у регистру потиче из меморије.

Табела 7.1.1. приказује процентуално заступљеност свих *CTAO* инструкција на коришћеним тестовима. Највећи удео на неком тесту јесте 22%, док је најмањи удео 1%. Просечни удео на свим коришћеним тестовима је 10%, а такође и медијана износи 10%. Како тестови садрже сто милиона инструкција, то значи да су просечно десет милиона инструкција по тесту заправо инструкције *CTAO*.

Табела 7.1.1. Заступљеност *CTAO* инструкција на коришћеним тестовима

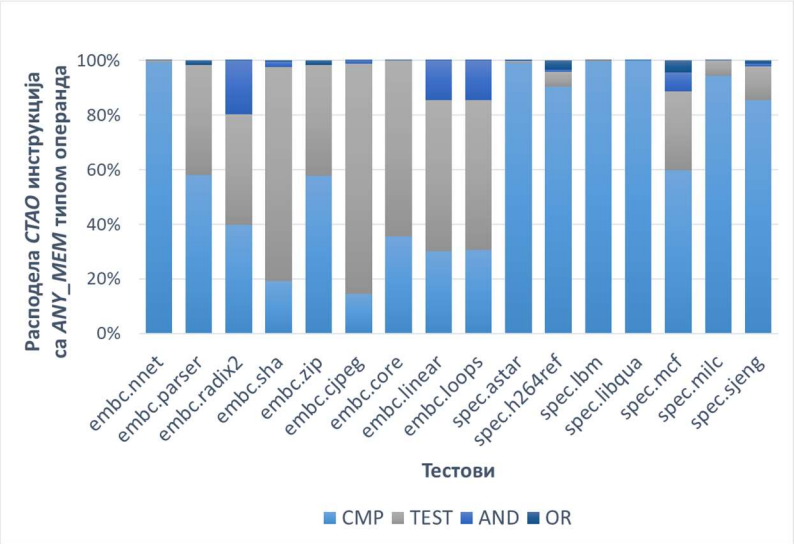
СКРАЋЕН НАЗИВ ТЕСТА	ЗАСТУПЉЕНОСТ *CTAO* ИНСТРУКЦИЈА
embc.nnet	11%
embc.parser	11%
embc.radix2	7%
embc.sha	8%
embc.zip	11%
embc.cjpeg	10%
embc.core	22%
embc.linear	15%
embc.loops	16%
spec.astar	7%
spec.h264ref	7%
spec.lbm	1%
spec.libqua	6%
spec.mcf	19%
spec.milc	4%
spec.sjeng	12%

На слици 7.1.1. приказан је дијаграм који приказује проценат погодних *CTAO* инструкција за предвиђање вредности операнада у односу на све *CTAO* инструкције, појединачно за инструкције *CMP, TEST, AND* и *OR*. Са дијаграма се може видети да тестови из групе *spec* тестова имају знатно већи удео погодних *CTAO* инструкција у односу на групу тестова *embc*. Разлог за то је што су тестови из групе *spec* меморијски интензивнији. Такође, са дијаграма се може видети да на неким тестовима удео погодних инструкција прелази 60%, док код неких достиже чак нешто мало испод 100%.



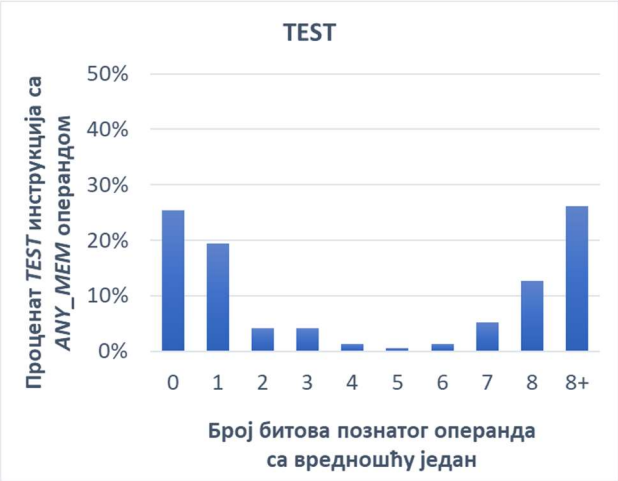
Слика 7.1.1. Проценат погодних *CTAO* инструкција у односу на све *CTAO* инструкције по тестовима

На дијаграму са слике 7.1.2. приказана је расподела појединачних инструкција *CMP*, *TEST*, *AND* и *OR* у оквиру погодних *CTAO* инструкција за предвиђање вредности операнада. Са овог дијаграма може се уочити да су најзаступљеније инструкције *CMP* и *TEST*. Након њих следи инструкција *AND*, док је најмање заступљена инструкција *OR*. Ово је очекивана расподела инструкција уколико се сагледа природа архитектуре *x86*. Заправо инструкције *CMP* и *TEST* служе за постављање индикатора у програмској статусној речи и њихово извршавање претходи условним инструкцијама које користе постављене индикаторе за одређивање услова. Како су условне инструкције у програмима доста заступљене, нарочито инструкције условних скокова, отуда и највећи удео у оквиру погодних *CTAO* инструкција имају инструкције *CMP* и *TEST*.

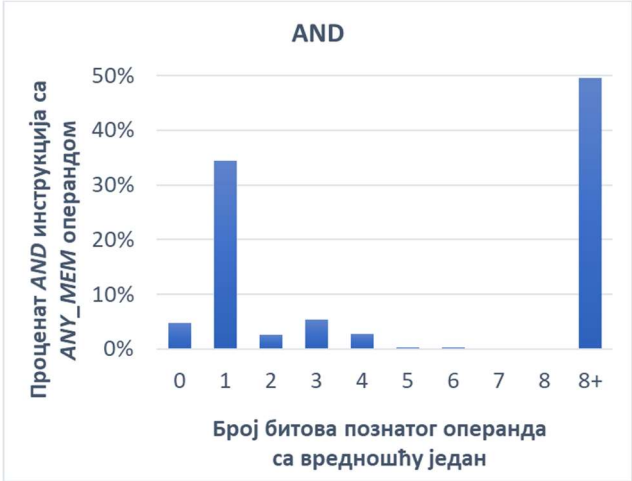


Слика 7.1.2. Дијаграм расподеле погодних *CTAO* инструкција по тестовима

У оквиру истраживања које је спроведено у оквиру ове докторске дисертације урађена је анализа вредности познатог операнда чија се вредност не предвиђа код погодних инструкција. Такве вредности као што је раније поменуто су одмах доступне, нпр. вредност операнда се налази у регистру. На сликама 7.1.3. и 7.1.4. приказани су хистограми који приказују колико битова са вредношћу један имају познати операнди погодних инструкција *TEST* и *AND* чији је један операнд *ANY_MEM* типа (операнд чија се вредност предвиђа). Као што је раније поменуто за ове две инструкције битно је исправно предвидети само оне битове који се налазе на позицијама битова познатог операнда са вредношћу један. Вредност битова на преосталим позицијама познатог операнда јесте нула што значи да одговарајући битови операнда чија се вредност предвиђа не утичу на резултат. Са хистограма за инструкцију *TEST* може се видети да преко 70% појава инструкција *TEST* имају вредност познатог операнда која садржи до осам битова са вредношћу један. Вредност познатог операнда инструкција *AND* има до осам битова са вредношћу један у преко 50% појава инструкције *AND*. Такође, за обе инструкције са дијаграма се може уочити да око 40% појава ових инструкција има један или ниједан бит са вредношћу један. То значи да у тим ситуацијама је потребно предвидети тачно само један или чак ниједан бит како би резултат био исправан. Наравно, битно је за предвиђање знати и позицију битова чије је вредности потребно тачно предвидети. Међутим, значајно се смањује опсег могућих вредности које се могу формирати са одређеним бројем битова чија је вредност један.

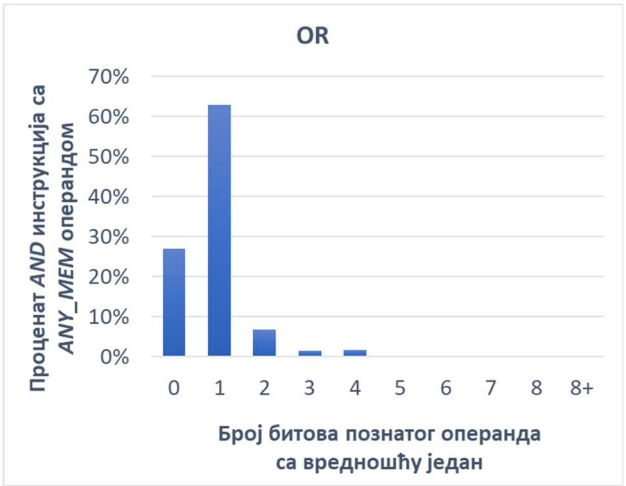


Слика 7.1.3. Хистограм броја битова са вредношћу један познатог операнда инструкција *TEST*



Слика 7.1.4. Хистограм броја битова са вредношћу један познатог операнда инструкција *AND*

На исти начин је спроведена анализа и вредности познатог операнда инструкција *OR*. Као што је поменуто код инструкције *OR* битно је исправно предвидети само оне битове који се налазе на позицијама битова познатог операнда са вредношћу нула. Вредност битова на преосталим позицијама познатог операнда јесте један што значи да одговарајући битови операнда чија се вредност предвиђа не утичу на резултат. На слици 7.1.5. приказан је хистограм који приказује колико битова са вредношћу један имају познати операнди погодних инструкција *OR* чији је један операнд *ANY_MEM* типа (операнд чија се вредност предвиђа).



Слика 7.1.5. Хистограм броја битова са вредношћу један познатог операнда инструкција *OR*

Са хистограма за инструкције *OR* може се уочити да приближно у 90% појава ове инструкције познати операнд има само један или ниједан бит са вредношћу један. Ово указује да остали битови познатог операнда имају вредност нула, што даље указује да је потребно онда тачно предвидети све битове са тих позиција у операнду који се предвиђа. Ово је у супротности у односу на претходне две инструкције где се опсег могућих вредности операнда чија се вредност предвиђа знатно смањио на основу вредности познатог операнда. Разлог за то се крије у програмима написаним на вишим програмским језицима код којих се операција логичког *или* користи приликом постављања вредности одређених битова неке

променљиве (постављање вредности један неких битова и то најчешће тачно једног бита). Преводилац тада операцију написану на вишем програмском језику преводи на инструкцију *OR*. Такође, како је употреба ове инструкције готово само у таквим ситуацијама стога је и најмање заступљена у оквиру групе *CTAO* инструкција.

7.2. Евалуација

У оквиру овог потпоглавља биће разматране постављене хипотезе у потпоглављу 4.4. на основу добијених резултата спроведених симулација. Свака хипотеза биће поновљена и за сваку хипотезу биће укратко описана поставка спроведених симулација као и коментари у вези са добијеним резултатима. На крају ће се за сваку хипотезу дати коментар у вези са важењем постављене хипотезе на освнову спроведених симулација. Такође, у оквиру овог потпоглавља биће и описане ствари које су занемарене приликом спровођена симулације.

7.2.1. Хипотезе 1 и 2 - тачан резултат инструкције са непрецизно предвиђеним операндима

i) Хипотезе

Хипотеза 1: Могуће је за погодне типове инструкција добијати исправан резултат иако је непрецизно предвиђен операнд.

Хипотеза 2: Постоје ситуације у којима је прецизност тачног резултата инструкције на основу предвиђеног операнда већа од прецизности тачно предвиђеног операнда.

ii) Резултати симулације

Циљ је да се испита да ли резултат *CTAO* инструкција са непрецизно предвиђеним вредностима операнада типа *ANY_MEM* може бити исправан. Као што је раније објашњено, није неопходно да се предвиди потпуно тачна вредност операнда како би се добио исправан резултат. Та особина *CTAO* инструкција их раздваја од осталих типова инструкција код којих је неопходно да се предвиди увек тачна вредност операнда.

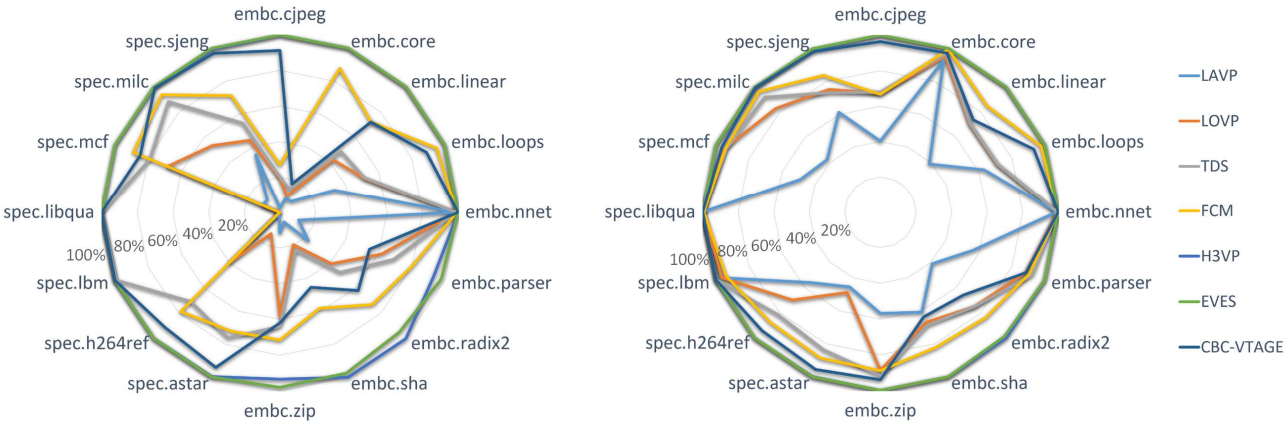
У циљу да се испита описана ситуација спроведене су трагом вођене симулације у симулатору *VPSim* над траговима извршавања од којих сваки садржи сто милиона инструкција, уз изузетак неких тестова који заврше своје извршавање са мање извршених инструкција. Предиктори који су коришћени су били подешени да користе максимално до осам килобајта простора за своје интерне структуре. Током симулације су праћени бројни параметри који су описани у претходном поглављу. Најбитнији параметри за испитивање ове ситуације су број предвиђања, број тачно предвиђених операнада и број тачних резултата инструкција на основу предвиђених вредности операнада.

На слици 7.2.1.1. приказан је дијаграм прецизности одабраних предиктора на коришћеним тестовима. Предиктори су вршили предвиђање вредности погодних *CTAO* инструкција са *ANY_MEM* типом операнда. На дијаграму а) са слике 7.2.1.1. приказана је прецизност тачно предвиђеног операнда, док је на дијаграму б) са исте слике приказана прецизност тачног резултата инструкције добијеног на основу предвиђеног операнда. Прецизност (вероватноћа) тачно предвиђеног операнда (P_{hitOpr}) и прецизност тачно добијеног резултата на основу предвиђене вредности (P_{hitRes}) могу се дефинисати формулама 7.2.1.1. и 7.2.1.2. У овим формулама промењљива *predicted* представља укупан број извршених предвиђања, док промењљиве *hitOpr* и *hitRes* представљају број тачно предвиђених операнада и број тачно добијених резултата инструкције на основу предвиђених операнада, респективно.

$$P_{hitOpr} = \frac{hitOpr}{predicted} \quad (7.2.1.1.)$$

$$P_{hitRes} = \frac{hitRes}{predicted} \quad (7.2.1.2.)$$

На ободу дијаграма са слике 7.2.1.1. налазе се имена тестова, док су различитим бојама обојене контуре које представљају прецизности које су предиктори постигли. Може се уочити да су контуре на дијаграму б) свих предиктора знатно ближе ободу дијаграма него на дијаграму а) са слике 7.2.1.1., што значи да је прецизност тачно добијеног резултата на основу предвиђене вредности операнда већа од прецизности тачно погођеног операнда. Како је у оба случаја исти број предвиђања био (*predicted*), може се коначно закључити да је у неким ситуацијама и када није тачно предвиђена вредност операнда, резултат инструкције био потпуно тачан.



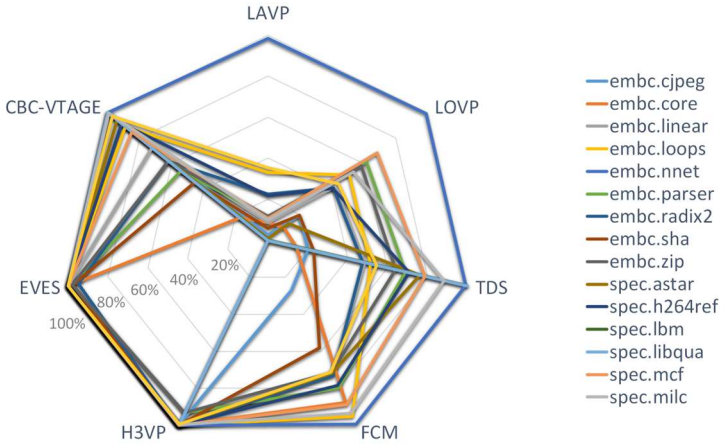
а) б)

Слика 7.2.1.1. Прецизност тачно предвиђеног операнда *CTAO* инструкција а) и прецизност тачног резултата *CTAO* инструкције на основу предвиђене вредности операнда б)

iii) Важење хипотеза

Број тачно добијених резултата инструкција (*hitRes*) је једнак суми броја тачно предвиђених вредности операнада (*hitOpr*) и броја ситуација у којима је резултат инструкције био исправан, а операнд није био тачно предвиђен (*hitResMissOpr*). Како се са дијаграма уочава да је $P_{hitRes} > P_{hitOpr}$ може се закључити да је $hitResMissOpr > 0$ што значи да постоје ситуације када се добија тачан резултат са непрецизно предвиђеним операндом. Овим се потврђује полазна **хипотеза 1** која каже да је могуће за погодне типове инструкција добијати исправан резултат иако је непрецизно предвиђен операнд. Такође, на основу претходног разматрања ($P_{hitRes} > P_{hitOpr}$) може се закључити да и полазна **хипотеза 2** важи која каже да постоје ситуације у којима је прецизност тачног резултата инструкције на основу предвиђеног операнда (P_{hitRes}) већа од прецизности тачно предвиђеног операнда (P_{hitOpr}).

Прецизност предиктора зависи и од самог програма који се извршава на процесору. Неки програми су погоднији за предвиђање вредности, тј. предиктори лакше науче исправно да предвиђају вредности током извршавања тих програма. Једноставан пример јесте уколико програм често користи само неки мали скуп вредности током свог извршавања.



Слика 7.2.1.2. Прецизност тачно предвиђеног операнда према коришћеним тестовима

На слици 7.2.1.2. приказане су постигнуте прецизности тачно предвиђеног операнда као на слици 7.2.1.1. а). Разлика је у томе што је промењен начин приказа (замењене осе). На слици 7.2.1.2. на ободу дијаграма се налазе имена предиктора док су контурама представљене остварене прецизности по тестовима. Са ове слике може се уочити да су сви предиктори, па чак и најпростији *LAVP* постигли прецизност близу 100% на тесту *embc.nnet*, што сведочи претходном опажању да прецизност предиктора зависи и од самих тестова.

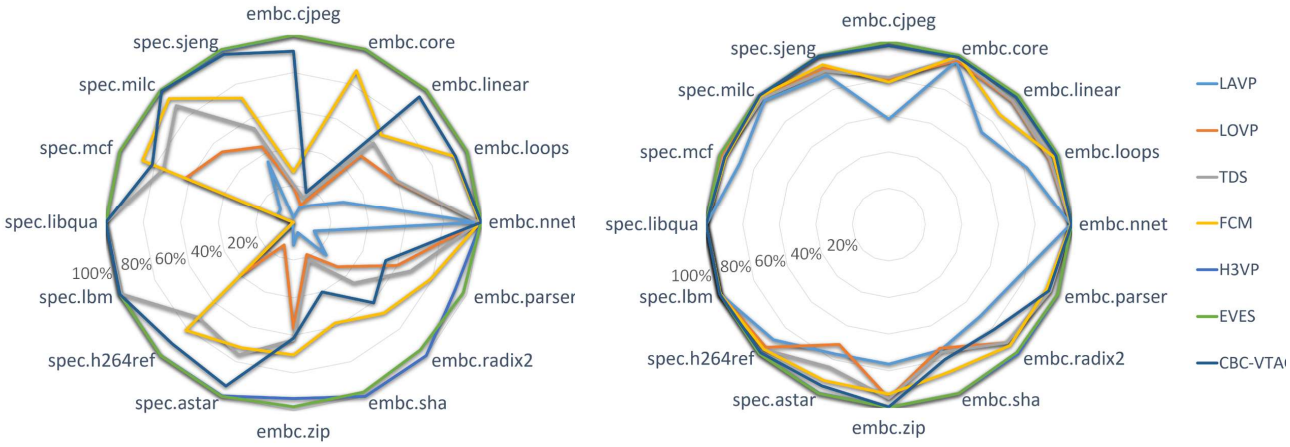
7.2.2. Хипотеза 3 – тачан користан резултат инструкције са непрецизно предвиђеним операндима

i) Хипотеза

Хипотеза 3: Могуће је за погодне типове инструкција добијати исправан користан резултат (део резултата инструкције који ће нека наредна да користи) иако је непрецизно предвиђен операнд.

ii) Резултати симулација

Током симулација посебан акценат је био на извршавању погодних инструкција *CMP* и *TEST* са *ANY_MEM* операндом (*CT* инструкције). У ту сврху праћено је извршавање условних инструкција које користе индикаторе постављене од стране *CT* инструкција. У оквиру симулације води се евиденција о томе која је инструкција последња поставила одређени индикатор у оквиру програмске статусне речи. Током симулације праћено је да ли је коришћен резултат *CT* исправан, тј. да ли је довео до исправног извршавања инструкције која га је користила. Као што је раније објашњено користан резултат неке инструкције представља део њеног резултата која нека наредна инструкција користи. Условне инструкције користе индикаторе како би се одредио услов на основу кога се оне извршавају. Над индикаторима се примењује одређена функција која формира услов. Управо резултат који даје та функција је означен као користан резултат *CT* инструкције.



а) б)

Слика 7.2.2.1. Прецизност тачно предвиђеног операнда *CT* инструкција а) и прецизност тачног корисног резултата на основу резултата *CT* инструкције са предвиђеним операндом б)

На слици 7.2.2.1. приказана је прецизност предиктора забележена на коришћеним тестовима приликом предвиђања вредности операнада *CT* инструкција. Слично као на слици 7.2.1.1. а) и на слици 7.2.2.1. је приказана прецизност тачно предвиђеног операнда инструкција *CT*, док је на слици 7.2.2.1. б) представљена прецизност тачног корисног резултата одређеног на основу резултата *CT* инструкције. Може се приметити да су дијаграми са слика 7.2.1.1. а) и 7.2.2.1. а) јако слични, разлог за то је што су *CT* инструкције најзаступљеније у оквиру *CTAO* инструкција, тј. инструкције *CMP* и *TEST* су доста више заступљеније од инструкција *AND* и *OR*.

iii) Важење хипотеза

Са слике 7.2.2.1. се може приметити да је прецизност тачно предвиђеног операнда знатно мања у односу на прецизност тачног корисног резултата. Како је постигнута прецизност тачног корисног резултата већа од прецизности тачно предвиђене вредности операнда то значи да се у неким ситуацијама и са непрецизно предвиђеним вредностима операнада добија исправан користан резултат. На основу претходног се закључује да полазна **хипотеза 3** важи, а она каже да је могуће у неким ситуацијама добијати исправан користан резултат и на основу непрецизно предвиђених вредности операнада.

Такође, може се приметити да су контуре које представљају прецизност предиктора још више ближе ободу дијаграма (ближе прецизности од 100%) на дијаграму б) са слике 7.2.2.1. него на дијаграму б) са слике 7.2.1.1. Овакав изглед дијаграма б) са слике 7.2.2.1. говори да је прецизност тачног корисног резултата инструкције изузетно велика, а који се добија на основу предвиђених вредности операнада *CT* инструкција. Интересантно је приметити да чак и једноставни предиктори постижу добре прецизности тачног корисног резултата.

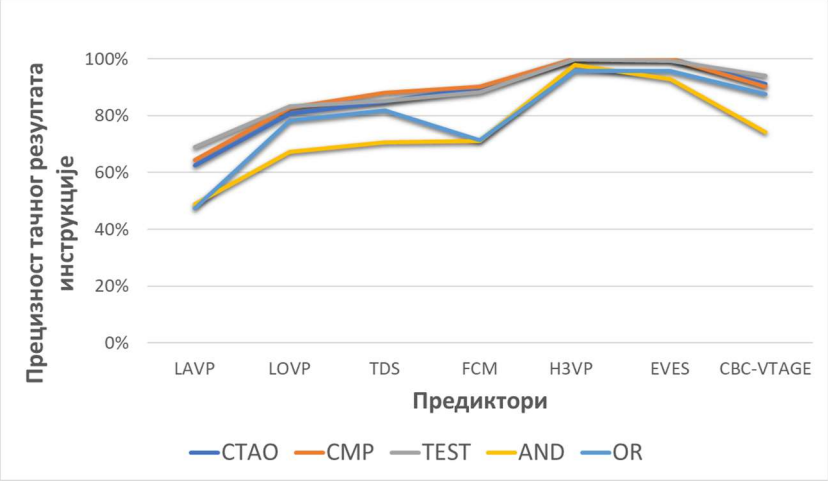
7.2.3. Хипотеза 4 - прецизност предиктора у зависности од типа инструкције

i) Хипотеза

Хипотеза 4: Могуће је користити постојеће предикторе вредности независно од типа погодних инструкција.

ii) Резултати симулација

У оквиру истраживања извршено је више симулација са различитим конфигурацијама. Како би се утврдило да ли предиктори вредности постижу боље прецизности уколико предвиђања врше само за један тип инструкција у односу на то када предвиђају све *CTAO* инструкције, извршене су симулације у којима су предиктори били конфигурисани да врше предвиђање за само један тип инструкција. На тај начин су добијене прецизности појединачно за сваку инструкцију *CMP*, *TEST*, *AND* и *OR* са *ANY_MEM* типом операнда. На слици 7.2.3.1. приказане су постигнуте прецизности добијеног тачног резултата инструкција на основу предвиђене вредности операнда и то појединачно за сваку инструкцију и збирно за *CTAO* инструкције. Приказане прецизности су просечне прецизности које су предиктори постигли на коришћеним тестовима.



Слика 7.2.3.1. Прецизност тачног резултата инструкције на основу предвиђеног операнда – појединачно и збирно за инструкције *CTAO*

iii) Важење хипотезе

Са слике 7.2.3.1. може се уочити да постоји одступање код неких предиктора између прецизности која је забележена приликом предвиђања свих *CTAO* инструкција и инструкција *AND* и *OR*. Ова одступања се јављају због тога што нема много појављивања инструкција *AND* и *OR* на основу којих би предиктори могли да се обуче за исправно предвиђање. Осим поменутих одступања може се закључити да прецизност добијеног тачног резултата на основу предвиђене вредности операнда генерално не зависи од типа погодне инструкције. Претходно излагање је у складу са **хипотезом 4** која каже да је могуће користити постојеће предикторе вредности независно од типа погодних инструкција.

7.2.4. Хипотеза 5 - прецизност предиктора у зависности од типа операнда

i) Хипотеза

Хипотеза 5: Могуће је користити постојеће предикторе вредности независно од тога колико би инструкција, за коју се предвиђа операнд, чекала операнд из меморије у случају да се не користи предвиђање.

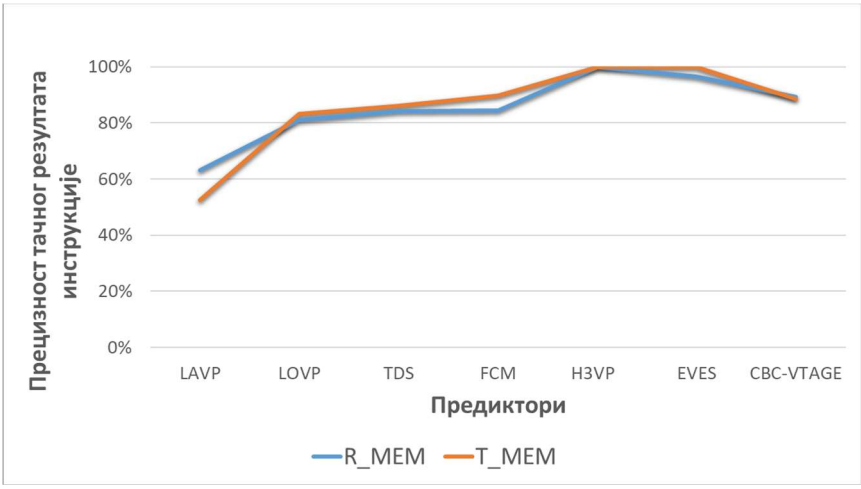
ii) Резултати симулација

Као што је раније поменуто у поглављу 4. ове докторске дисертације дефинисана су два типа погодних операнада за предвиђање вредности. Први тип је *T_MEM* тип операнда који представља операнде који су специфицирани неким меморијским адресирањем. Други

тип је *R_MEM* тип операнда који представља операнде који су одређени регистром, али вредност у регистру потиче из меморије. Оба ова типа, као што је дефинисано у четвртом поглављу, формирају тип *ANY_MEM*.

Слично претходним симулацијама спроведене су симулације које су послужиле да се утврди да ли прецизност предиктора зависи од типа операнда чија се вредност предвиђа. Инструкције чији операнди су се посматрали су инструкције *CTAO*. Симулације које су спроведене су биле тако конфигурисане да предиктори предвиђају вредности појединачно само за један од два дефинисана типа операнда. Другим речима симулације су извршаване тако да су предиктори предвиђали вредности или само за *T_MEM* тип операнда или само за *R_MEM* тип операнда.

На слици 7.2.4.1. приказане су прецизности тачног резултата *CTAO* инструкције на основу предвиђених вредности операнада и то за *R_MEM* тип и за *T_MEM* тип. Приказане прецизности су просечне прецизности које су предиктори постигли на коришћеним тестовима.



Слика 7.2.4.1. Прецизност тачног резултата *CTAO* инструкције на основу предвиђеног операнда – појединачно за *R_MEM* и *T_MEM* тип операнда

iii) Важење хипотезе

Са слике 7.2.4.1. може се уочити да постоји мало одступање само код два предиктора, *LAVP* и *FCM*. То одступање износи од 5% до 10%, док код осталих предиктора је оно знатно мање и износи мање од 1%. На основу овога се може закључити да прецизност тачног резултата инструкције не зависи значајно од типа операнда за који се предвиђа вредност. На основу овог закључка може се такође закључити да важи и **хипотеза 5** која каже да је могуће користити постојеће предикторе вредности независно од тога колико би инструкција, за коју се предвиђа операнд, чекала операнд из меморије у случају да се не користи предвиђање. Заправо ако се ради о типу операнда *T_MEM* тада инструкција мора да сачека онолико циклуса колико је кашњење меморије. У случају да се ради о *R_MEM* типу, инструкција мора да сачека онолико циклуса колико је још преостало некој претходној инструкцији која је иницирала дохватање вредности да заврши приступ меморији. Дакле, хипотезом 5 се другим речима каже да прецизност предиктора неће зависити од тога за који тип операнда се предвиђа вредност, тј. колико циклуса се чека да права вредност операнда стигне из меморије.

7.2.5. Хипотеза 6 - евалуација модела извршавања

i) Хипотеза

Хипотеза 6: Постоје ситуације у којима извршавање са непрецизно предвиђеним операндима може постизати боље време извршавања од извршавања само са тачно предвиђеним операндима на основу постојећих предиктора вредности.

ii) Резултати симулације

У потпоглављу 5.5. ове докторске дисертације одређена је неједнакост која говори под којим условима модел 3 извршавања постиже боље очекивано време од модела 2. Та неједнакост 5.5.8. је и овде поновљена. Све док важи неједнакост у формули 5.5.8. модел 3 постиже боље време извршавања од модела 2.

Приликом извођења неједнакости 5.5.8. у потпоглављу 5.5. усвојено је да вероватноћа промашаја предвиђања вредности операнда ($p_{missOpr}(n)$) и вероватноћа да је резултат нетачан на основу предвиђеног операнда ($p_{missRes}(n)$) не зависе од тога колико се циклуса чека вредност операнда из меморије (параметар n). Како је симулацијама показано да заиста прецизност предиктора не зависи од типа операнда за који се врши предвиђање (хипотеза 5), усвојена претпоставка приликом извођења неједнакости је исправна.

$$\frac{t_{pen}}{t} > f\big(\Delta, s(k)\big) \tag{5.5.8.}$$

Према неједнакости 5.5.8. може се дефинисати услов када модел 3 постиже боље очекивано време извршавања од модела 2. Са леве стране неједнакости стоји однос времена које је потребно да се изврши опоравак и времена просечног трајања једне инструкције. Док се на десној страни неједнакости налази функција f која зависи од разлике вероватноћа Δ и функције $s(k)$. Функција f је дефинисана у потпоглављу 5.5. и овде је дата у оквиру формуле 7.2.5.1. Разлика вероватноћа Δ представља разлику између вероватноће промашаја вредности предвиђеног операнда ($p_{missOpr}$) и вероватноће промашаја тачног резултата инструкције на основу предвиђеног операнда ($p_{missRes}$). Ова разлика је дефинисана формулом 5.5.5. у потпоглављу 5.5., а овде је такође поновљена.

$$f\big(\Delta, s(k)\big) = \frac{1}{\Delta} - s(k) \tag{7.2.5.1.}$$

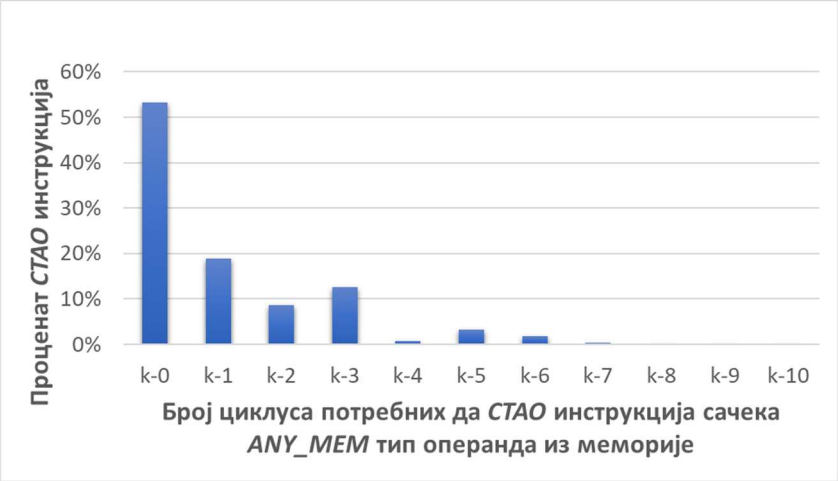
$$p_{missOpr} - p_{missRes} = \Delta \tag{5.5.5.}$$

Вредност функције $s(k)$ представља суму производа вероватноће да ће тачна вредност операнда постати доступна тачно за n циклуса и самог броја циклуса n. Као што је раније објашњено n може имати вредности из скупа $[1 \ldots k]$. Уколико се ради о операнду *R_MEM* типа тада је n мање од k, док уколико се ради о операнду *T_MEM* типа тада је n једнако параметру k. Функција $s(k)$ је дефинисана формулом 5.5.6. у оквиру потпоглавља 5.5., а сада је и овде поновљена.

$$\sum_{n=1}^{k} p(n)n = s(k) \tag{5.5.6.}$$

Током симулација праћено је колико често се јављају *CTAO* инструкције са *ANY_MEM* типом операнда и то појединачно за типове *R_MEM* и *T_MEM*. За *R_MEM* тип је праћено и колико раније циклуса је иницирано његово дохватање из меморије. На слици 7.2.5.1. је приказан хистограм који представља колико циклуса треба да чека *CTAO* инструкција да би вредност њеног операнда типа *ANY_MEM* постала доступна за случај да се не користи

предвиђање вредности операнда. Заправо овај хистограм представља расподелу *R_MEM* и *T_MEM* типа операнда. Описано праћење је урађено над свим тестовима, а на хистограму су приказане просечне вредности на основу свих тестова. На хоризонталној оси су представљени бројеви који означавају колико циклуса је потребно да вредност операнда стигне из меморије. Вредност *k-0* одговара операндима који су типа *T_MEM*, то значи да је сама *CTAO* инструкција иницирала дохватање из меморије и онда мора сачекати тачно *k* циклуса колико износи кашњење меморије. Све остале вредности на хоризонталној оси хистограма одговарају операндима типа *R_MEM*. То су вредности *k-x* ($x > 0$). Ово значи да је дохватање вредности операнда који треба да користи *CTAO* инструкција покренула нека претходна инструкција и то *x* циклуса раније. Са хистограма се може видети да преко 50% *CTAO* инструкција има *T_MEM* тип операнда што значи да те инструкције морају да чекају тачно *k* циклуса вредност операнда из меморије. Остале *CTAO* инструкције имају *R_MEM* тип операнда и оне морају да чекају вредност операнда из меморије *k-x* циклуса. На основу хистограма може се закључити да око 99.5% *CTAO* инструкција са *ANY_MEM* типом операнда чека до *k*-10 циклуса вредност операнда из меморије.



Слика 7.2.5.1. Број циклуса потребних да вредност операнда постане доступна из меморије

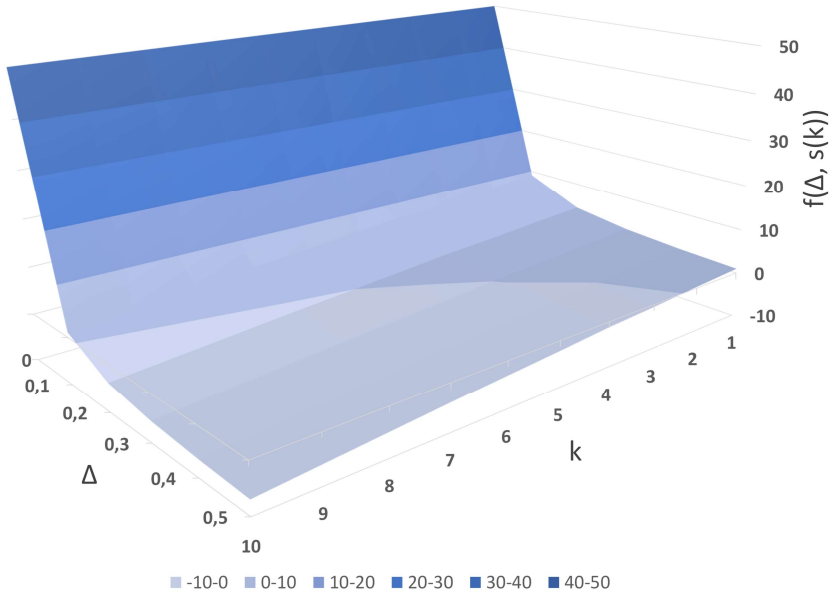
На основу прикупљене статистике извршавања симулација одређене су вредности функције *s* у зависности од параметра *k*. Параметар *k* као што је раније објашњено представља кашњење меморије, тј. број циклуса потребних да се дохвати вредност из меморије. У табели 7.2.5.1. дате су вредности функције *s(k)*. За сваки тест су израчунате вредности функције *s(k)*, а потом су одређене просечне вредности *s(k)* на основу свих тестова. На основу чињенице да 99.5% *CTAO* инструкција са *ANY_MEM* типом операнда чека до *k*-10 циклуса, за параметар *k* у табели 7.2.5.1. узете су вредности у опсегу од 1 до 10. Према формули 5.5.6. приказане вредности у овој табели представљају заправо математичко очекивање након колико циклуса ће вредност операнда постати доступна инструкцији уколико је *k* кашњење меморије.

Табела 7.2.5.1. Вредност функције *s(k)*

S(1)	*S(2)*	*S(3)*	*S(4)*	*S(5)*	*S(6)*	*S(7)*	*S(8)*	*S(9)*	*S(10)*
1,00	1,73	2,55	3,20	4,18	5,06	5,96	6,94	7,93	8,91

На основу усвојене вредности за параметар *k* може се представити изглед функције *f*. На слици 7.2.5.2. представљена је површ која одговара овој функцији. Уколико се посматра изглед функције *f*, може се приметити да је њена вредност у неким деловима површи

негативна. Ситуација када је вредност функције *f* негативна јесте када је разлика вероватноћа Δ довољно велика и тиме је њена реципрочна вредност мала. Одузимањем вредности функције *s(k)* од реципрочне вредности разлике вероватноћа Δ у таквој ситуацији се добија негативна вредност. На слици 7.2.5.2. најсветлијом плавом бојом је обојена површ где функција *f* има негативну вредност. Када је вредност функције негативна то онда значи да модел 3 постиже боље очекивано време извршавања од модела 2, што је у складу са хипотезом 6. Разлог за то је што неједнакост из формуле 5.5.8. тада важи јер је однос времена које је потребно да се изврши опоравак услед промашаја и времена просечног трајања једне инструкције (t_{pen}/t) сигурно већи од нуле пошто су и времена t_{pen} и t већа од нуле, што је објашњено и у потпоглављу 4.5.



Слика 7.2.5.2. График функције *f*

У табели 7.2.5.2. дате су дискретне вредности функције *f* за вредности параметра *k* од 1 до 10 и за вредности разлике вероватноћа Δ у опсегу од 0 до 0,5. У табели су зеленом бојом означене све негативне вредности функције *f*. У тим ситуацијама модел 3 са непрецизно предвиђеним операндима постиже увек боље очекивано време извршавања од модела 2, што је исто у складу са хипотезом 6.

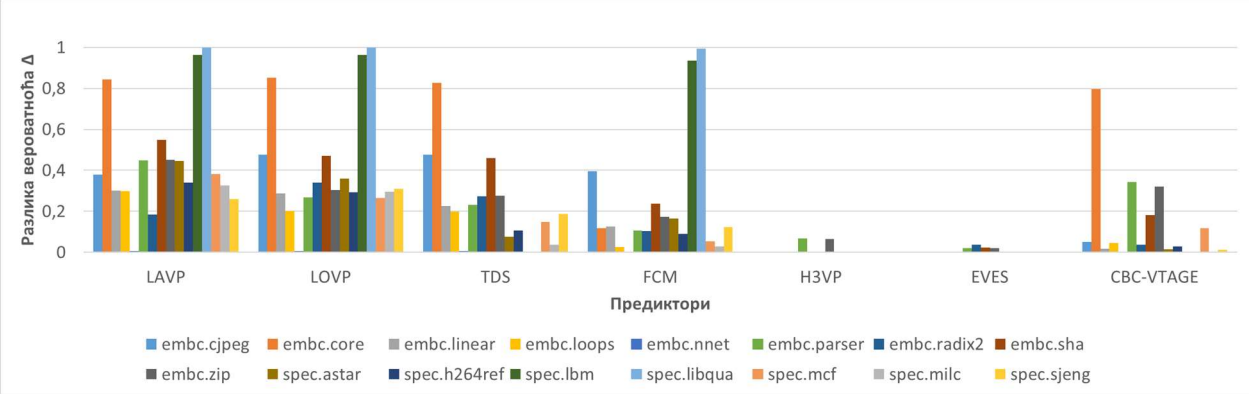
Табела 7.2.5.2. Вредност функције *f*

Δ \ k	1	2	3	4	5	6	7	8	9	10
0,05	19,00	18,27	17,45	16,80	15,82	14,94	14,04	13,06	12,07	11,09
0,1	9,00	8,27	7,45	6,80	5,82	4,94	4,04	3,06	2,07	1,09
0,2	4,00	3,27	2,45	1,80	0,82	-0,06	-0,96	-1,94	-2,93	-3,91
0,3	2,33	1,61	0,79	0,13	-0,84	-1,73	-2,63	-3,61	-4,59	-5,57
0,4	1,50	0,77	-0,05	-0,70	-1,68	-2,56	-3,46	-4,44	-5,43	-6,41
0,5	1,00	0,27	-0,55	-1,20	-2,18	-3,06	-3,96	-4,94	-5,93	-6,91

Такође, вредности функција *f* у ситуацијама када је разлика вероватноћа Δ једнака или већа од 10%, крећу се у опсегу од 0,13 до 9,00. То значи да однос t_{pen}/t треба да буде већи од

ових вредности да би модел 3 имао боље очекивано време извршавања према формули 5.5.8. Потпуно реално је очекивати да је потребно време за опоравак услед промашаја једнако времену потребном да се изврши неколико инструкција ($t_{pen} = mt,\ m > 0$), што значи да и у овим ситуацијама модел 3 може постићи боље време извршавања од модела 2. Такође, са порастом кашњења меморије k вредност функције f опада што иде у прилог моделу 3.

На основу спроведених симулација добијене су вероватноће промашаја вредности операнда ($p_{missOpr}$) и вероватноће промашаја резултата инструкције на основу предвиђене вредности операнда ($p_{missRes}$). На слици 7.2.5.3. приказане су разлике ових вероватноћа које су забележене код предикторима по тестовима. Може се приметити да простији предиктори (*LAVP, LOVP, TDS* и *FCM*) имају доста већу разлику. Разлог за то је што је њихова прецизност предвиђања потпуно тачне вредности операнда доста мања од сложених предиктора. Треба напоменути да они такође немају механизам поверења па предвиђање врше за сваку појаву погодне инструкције чиме смањују своју прецизност.



Слика 7.2.5.3. Разлика вероватноћа $p_{missOpr}$ и $p_{missRes}$

У табели 7.2.5.3. дате су просечне разлике (Δ) ових вероватноћа на коришћеним тестовима појединачно за сваки предиктор. Такође, у овој табели су дате и просечене забележене прецизности на свим тестовима и то прецизност тачног предвиђеног операнда (p_{hitOpr}) и прецизност тачног резултата на основу предвиђене вредности операнда (p_{hitRes}).

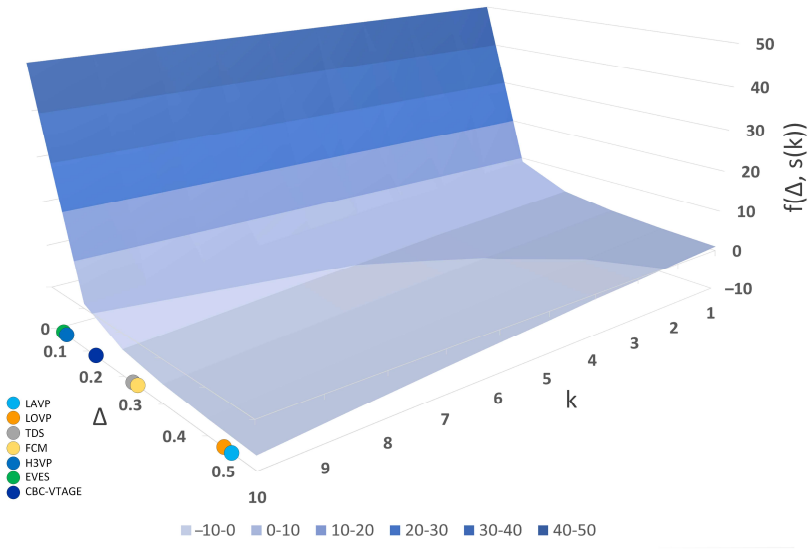
Табела 7.2.5.3. Просечне прецизности предиктора и разлика вероватноћа Δ

ПРЕДИКТОР	Δ	p_{hitOpr}	p_{hitRes}
LAVP	0,449	0,176	0,625
LOVP	0,418	0,389	0,807
TDS	0,221	0,628	0,849
FCM	0,230	0,669	0,899
H3VP	0,011	0,988	0,999
EVES	0,008	0,991	0,999
CBC-VTAGE	0,123	0,788	0,911

На основу табеле 7.2.5.3. може се уочити да су се прецизности тачног резултата једноставнијих предиктора приближили прецизностима сложенијих предиктора. Разлог за ово управо лежи у особинама погодних инструкција за које није неопходно потпуно тачно предвидети вредност операнда па стога и једноставнији предиктори могу послужити за извршавање са непрецизно предвиђеним вредностима.

iii) Важење хипотезе

На слици 7.2.5.4. је приказана иста површ функције f као на слици 7.2.5.2. уз означавање вредности Δ за предикторе. Сви предиктори, осим предиктора *H3VP* и *EVES*, имају вредност функције f у опсегу који не прелази вредност 10. На основу тога и формуле 5.5.8. може се закључити да ће модел 3 постизати боље време извршавања од модела 2 уколико је однос t_{pen}/t већи од 10. Колики је однос између t_{pen}/t зависи од конкретне архитектуре процесора на којој би се имплементирало непрецизно предвиђање вредности. За очекивати да је то време буде бар неколико пута веће од (просечног) трајања једне инструкције ако се у обзир узме да је у оквиру опоравка потребно повратити стање процесора што укључује или испирање комплетне проточне обраде или селективно поништавање само спекулативних инструкција (потпоглавље 2.3.). На основу овог излагања може се закључити да **хипотеза 6** важи, а која каже да модел са непрецизно предвиђеним операндима (модел 3) може постизати боље време извршавања од модела са само тачно предвиђеним операндима (модел 2).



Слика 7.2.5.4. График функције f са означеним разликама вероватноћа Δ

7.3. Коментари у вези спроведених симулација

i) Постављање индикатора у оквиру програмске статусне речи

У спроведеним симулацијама за сваки индикатор у оквиру програмске статусне речи водила се евиденција о томе која инструкција је последња поставила тај индикатор. Оно што се кроз симулације касније утврдило јесте да за условне инструкције на архитектури *x86* која је коришћена, не долази никада до ситуације да се услов формира на основу индикатора који су постављени од стране различитих инструкција. Такође, утврђено је да увек непосредно пред условну инструкцију се извршава нека инструкција која поставља индикаторе, најчешће инструкције *CMP* и *TEST*. Другим речима, непосредно након сваке *CMP* или *TEST* инструкције извршава се нека условна инструкција.

ii) Улази предиктора и вредности које су предвиђане

Као што је већ поменуто за улазе предиктора су коришћене адресе инструкција за које се врши предвиђање и ти резултати су представљени у оквиру овог поглавља. На основу

адресе инструкције предиктори приступају својим интерним структурама приликом предвиђања и ажурирања. Ово је стандардан начин који се користи и који користе описани предиктори у поглављу 3. У оквиру симулација које су спроведене, спроведене су и симулације у којима су предиктори користили улазе који укључују вредност познатог операнда погодних инструкција за предвиђање:

- вредност познатог операнда – на улаз предиктора се прослеђивала неизмењена вредност познатог операнда;
- хеш функција 1 – на улаз предиктора се прослеђивао резултат функције која као параметре добија адресу и вредност познатог операнда инструкције и која над њима примењује операцију логичког *или*;
- хеш функција 2 – на улаз предиктора се прослеђивао резултат функције која као параметре добија адресу и вредност познатог операнда инструкције. Функција прво рачуна број јединица у оквиру познатог операнда, а потом ту вредност записану на шест битова поставља на место најнижих битова адресе, што представља резултат функције који се користи као улаз предиктора.

Такође, представљени резултати су резултати симулација у оквиру којих су предиктори били упошљени да предвиђају вредност операнда типа *ANY_MEM* инструкција *CTAO*. Поред овога извршене су и симулације у оквиру којих су предиктори били упошљени да предвиђају вредности различитих података:

- директно резултат инструкције;
- само вредност програмске статусне речи.

Варијације које су примењене у вези са различитим улазима који су прослеђивани предикторима као и различити подаци које су предиктори предвиђали нису дали генерално боље резултате прецизности у односу на оне представљене у овом поглављу. На неким тестовима су резултати били занемарљиво бољи, а на неким лошији па се може закључити да предиктори постижу најбољу прецизност када предвиђају директно вредност операнда, а као улаз добијају адресу инструкције за коју се врши предвиђање.

iii) Ажурирање предиктора у симулацији и механизми поверења

У спроведеним симулација процес ажурирања предиктора се одвија непосредно након извршеног предвиђања, а пре наредне инструкције. У том тренутку, како се ради о симулацији, позната је одмах и тачна вредност и могуће је одмах извршити ажурирање предиктора. У процесорима са проточном обрадом постоји шанса да су појаве две *CTAO* инструкције близу једна друге. У таквој ситуацији процес ажурирања предиктора након прве појаве *CTAO* инструкције може да буде још увек у току када треба да се одреди предвиђање за другу појаву. Последица овога јесте да друга појава инструкције неће имати ажурирано стање предиктора.

Два решења се могу применити да се описана ситуација реши. Прво решење јесте да се користи тренутно стање предиктора приликом одређивања предвиђања за другу појаву. Друго решење подразумева да се изврши спекулативно ажурирање предиктора одмах након предвиђања за прву појаву на основу предвиђене вредности. У спроведеним симулацијама број инструкција између две појаве *CTAO* инструкција за које се врши предвиђање је просечно између 40 и 900 инструкција у зависности од теста. Како су *CTAO* инструкције довољне размакнуте, ситуација у вези ажурирања није разматрана већ је као што је речено примењено ажурирање одмах након извршеног предвиђања.

Предиктори са светског такмичења у предвиђању вредности који су коришћени у оквиру симулација имају изузетно јаке механизме поверења. Механизми поверења им омогућавају да селективно врше предвиђање само у ситуацијама у којима механизми процене да су велике шансе да се на исправан начин предвиди вредност. Сходно томе, ови предиктори постижу изузетну велику прецизност али са знатно мањом покривеношћу. Предиктори са светског такмичења *H3VP* и *EVES* у спроведеним симулацијама имају минималне разлике између вероватноће тачно погођеног резултата инструкције на основу предвиђене вредности операнда и вероватноће тачно погођене вредности операнда. Разлог је као што је наведено коришћење јаког механизма поверења тако да ови предиктори када су вршили предвиђање у већини случајева су тачно предвидели вредност операнда. У оваквој ситуацији вредност функције f се приближава бесконачности чинећи примену модела 3 извршавања неизводљивом, тј. неисплативом. Смањивањем утицаја механизма поверења, може се повећати покривеност, али и разлика поменутих вероватноћа, чиме се отварају могућности за примену модела 3.

8. СМЕРНИЦЕ ЗА ДАЉА ИСТРАЖИВАЊА

У оквиру овог поглавља биће дати коментари у вези са могућим правцима даљег истраживања теме ове докторске дисертације. Биће дата анализа могућности развијања новог предиктора вредности који би користио феномен да није увек неопходно предвидети комплетно тачну вредност. Такође, биће представљен још један модел извршавања који би укључивао још и информацију ког типа је инструкција која користи резултат инструкције за коју је извршено (непрецизно) предвиђање вредности операнда. Такође, биће описан још један тип операнада који могу бити погодни за предвиђање вредности. Последња ствар која ће бити разматрана у оквиру овог поглавља јесте могућност коришћења представљених модела извршавања код процесора са *in order* и *out of order* извршавањем.

8.1. Развијање новог предиктора

Резултати спроведеног истраживања у оквиру ове докторске дисертације је показало да је могуће добијати тачан резултат погодних инструкције и на основу непрецизно предвиђене вредности операнда. Код погодних инструкција није неопходно да сваки бит вредности који се предвиђа има тачну вредност. Ово отвара могућност да се развије предиктор који би користио овај феномен на начин да предвиђену вредност формира предвиђајући вредност бит по бит. Оно што је још интересантно јесте да није неопходно за неке погодне инструкције предвиђати ни све битове пошто неки битови неће ни утицати на резултат инструкције. Позиције битове чије би вредности требало предвиђати се могу одредити на основу типа инструкције и другог операнда чија је вредност позната.

Сличан концепт предвиђања је већ представљен код предиктора скокова у раду [74]. Представљени предиктор врши предвиђање одредишне адресе скока на начин да сваки бит адресе предвиђа посебно користиће неуралне мреже. Тако предвиђену вредност адресе затим упоређује са претходним адресама које су се појављивале и за коначно предвиђање бира ону адресу која се највише подудара са предвиђеном адресом.

8.2. Проширење модела извршавања са непрецизним предвиђањем

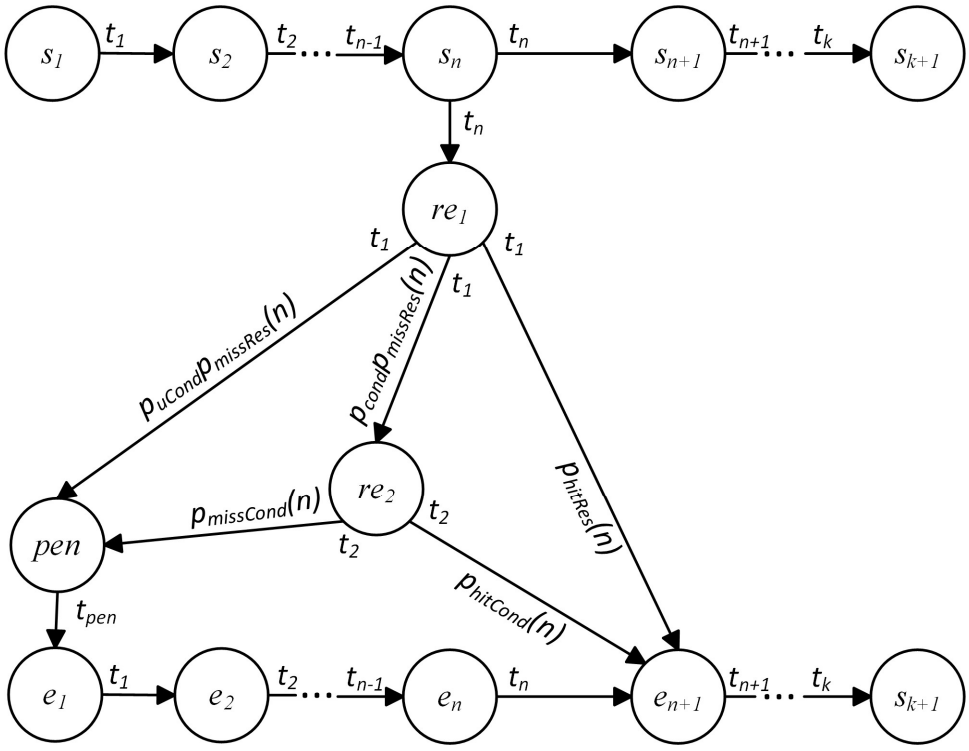
i) Проширења модела 3 извршавања

У петом поглављу ове докторске дисертације представљен је модел 3 извршавања инструкција који описује извршавање са непрецизно предвиђеним вредностима операнда. Према том моделу уколико је резултат инструкције за коју се вршило предвиђање исправан, извршавање се наставља без обзира на то што је предвиђена вредност операнда можда различита од тачне вредности операнда. У оквиру модела 3 није разматрано ког типа је инструкција која користи резултат инструкције за коју се вршило предвиђање.

Модел 3 се може проширити на начин да се у обзир узима и тип инструкције који користи резултат инструкције за коју се вршило предвиђање. На слици 8.2.1. приказан је проширени модел 3 извршавања. Приказани модел узима у обзир да ли условна инструкција користи резултат инструкције за коју се предвиђала вредност операнда. Инструкција

означена са s_1 представља инструкцију за коју се врши предвиђање, док је инструкција s_2 инструкција која користи резултат инструкције s_1.

Као и код модела 3, када тачна вредност операнда чија се вредност предвиђала постане доступна, неопходно је поново извршити инструкцију s_1 са тачном вредношћу тог операнда. Уколико су резултати инструкције са предвиђеном и са тачном вредношћу операнда идентични, тада се извршавања наставља успешно. У ситуацији када ови резултати нису идентични, тада постоје два сценарија.



Слика 8.2.1. Проширење модела извршавања са непрецизним предвиђањем

Први сценарио односи се на ситуацију када инструкција s_2 није условна инструкција. Тада се извршавање наставља као и код модела 3. Прелази се у стање *pen* где се ради опоравак услед лошег предвиђања, а потом се наставља поновно извршавање инструкција почевши од *e1*.

Други сценарио односи се на ситуацију када је инструкција s_2 условна инструкција. У овом сценарију није неопходно одмах прећи у стање *pen* већ се прелази у чвор означен са re_2. У оквиру овог чвора поново се израчунава услов условне инструкције, али сада са тачном вредношћу резултата инструкције s_1. Како није неопходно да су сви индикатори програмске статусне речи исправно постављени, могуће је добити исправан исход услова и на основу спекулативног резултата инструкције s_1. Поред тога, могуће је добити и тачан исход услова иако нису сви индикатори који учествују у формирању услова исправно постављени. Услов који користи инструкција s_2 се дефинише као користан резултат инструкције s_1, као што је објашњено у седмом поглављу. Дакле, за условне инструкције тек уколико услов који користе није исправно постављен, а што се проверава у оквиру чвора re_2, прелази се у стање *pen*. У чвору re_2 се заправо врши провера да ли је користан резултат инструкције *s1* исправан. У овом сценарију се избегава трошење времена на опоравак и поновно извршавање свих инструкција уколико је користан резултат исправан и поред непрецизно предвиђеног операнда инструкције s_1.

ii) Погодни типови операнада за предвиђање вредности

У четвртом поглављу ове докторске дисертације описани су погодни типови операнада за предвиђање вредности. Као погодни операнди за предвиђање означени су они чија вредност потиче из меморије. Разлог је тај што инструкције у случају промашаја у оквиру кеш меморије приликом дохватања вредности операнда, морају да сачекају да тражена вредност стигне из оперативне меморије. Ово уноси кашњење у извршавање програма па су стога ови операнди идентификовани као погодни за предвиђање.

Још један тип операнда се може уочити сличним резоновањем везаним за кашњење. Уколико инструкција користи као свој операнд резултат неке друге инструкције, тада се може применити предвиђање вредности тог недостајућег операнда. Ово се заправо своди на то да се предвиди резултат извршавања инструкције која производи вредност тог операнда. Како би се препознали такви операнди потребно је препознати која инструкција производи његову вредност и колико кашњење уноси та инструкција. Уколико та инструкција захтева више времена од просечног трајања једне инструкције (ова граница се може дотеривати) тада би се могао применити механизам предвиђања вредности који би дао предвиђену вредност за недостајући операнд. Дакле, могу се предвиђати резултати оних инструкција које дуго трају како на њихове резултате не би чекале друге инструкције.

8.3. Могућност примене представљених модела код *in order* и *out of order* процесора

Код *out of order* процесора извршавање инструкција се одвија не према оригиналном редоследу инструкција из програма, већ према доступности операнада инструкција и слободним јединицама извршавања (енгл. *execution unit*). Ово значи да процесор може преко реда извршавати инструкције које имају доступне операнде док чека да операнди за неке друге постану доступни, водећи рачуна притом о зависности по подацима. Пример таквих процесора су они чије су архитектуре засноване на Томасуловом алгоритму [75]. Инструкција чије вредности операнада нису доступне, а погодне су за предвиђање вредности, могу се одмах издати на извршавање. На тај начин се инструкција (И1) може одмах извршити са предвиђеном вредношћу операнда чим је слободна одговарајућа јединица извршавања. Таква инструкција је спекулативно извршена па ју је потребно означити да је спекулативно извршена. Такође, све инструкције које директно или индиректно (преко неке друге инструкције) користе њен резултат морају бити означене као спекулативне. На такав начин формира се ланац спекулативних инструкција који се заправо може представити као предложени модел 3 извршавања. Када тачна вредност операнда инструкције И1 постане доступна тада се она поново издаје на извршавање како би се добио тачан резултат. Уколико је спекулативан резултат једнак тачном резултату тада се зависне инструкције из ланца могу потврдити (енгл. *commit*). Треба приметити да се у оваквом ланцу могу наћи зависне инструкције за које се такође вршило предвиђање вредности, нпр. вредност једног операнда је резултат инструкције И1, а вредност другог операнд је добијена предвиђањем. Такве инструкције остају у ланцу, тј. не могу се потврдити док се не утврди исправност предвиђања другог операнда. У случају да предвиђање за инструкцију И1 није исправно тада се морају поново издати на извршавање све зависне инструкције, тј. цео формирани ланац инструкција. Овде се заправо примењује селективно поновно извршавање инструкција, описано у другом поглављу, јер се могу поново издати на извршавање само инструкције које су се налазиле у ланцу. Независне инструкције се према Томасуловом алгоритму несметано извршавају и потврђују без обзира на то да ли се неке инструкције извршавају спекулативно.

Код *in order* процесора извршавање инструкција се одвија према редоследу инструкција какав је у оквиру програма. Код ових процесора могуће је применити модел 3 извршавања само што се у ланцу инструкција налазе све инструкције, а не само зависне као код *out of order* процесора. У ланцу *out of order* процесора налазе се само зависне инструкције од инструкције за коју се вршило предвиђање јер се остале могу извршавати на слободним јединицама за извршавање према Томасуловом алгоритму. Код *in order* процесора у ланцу инструкција налазе се све инструкције које следе након инструкције за коју се вршило предвиђање (И1) према редоследу инструкција какав је у програму. Међу тим инструкцијама налазе се инструкције које су зависне од резултата инструкције И1, али и независне инструкције. Како се код *in order* процесора извршавају инструкције баш по редоследу какав је у програму, онда се приликом опоравка услед лошег предвиђања понављају све инструкције (комплетно испирање процесора јер се све инструкције налазе у ланцу).

9.Закључак

Сваки искорак у истраживању области спекулативног извршавања коришћењем механизма предвиђања вредности је драгоцен из разлога што још увек није примењен код процесора. Главни изазов приликом примене механизма предвиђања вредности јесте постизање високе прецизности како би се ређе дешавали скупи опоравци услед лошег предвиђања. Сагледавајући претходно поменуте ствари ова докторска дисертација предлаже решење за искоришћење механизма предвиђања вредности при предвиђању вредности операнада инструкција чији резултати могу бити тачни чак и када предвиђени операнди имају непрецизне вредности. На тај начин у ситуацијама када је резултат инструкције на основу непрецизно предвиђене вредности операнда тачан, избегава се опоравак услед лошег предвиђања јер је резултат инструкције исправан.

У оквиру истраживања током израде ове докторске дисертације механизам предвиђања вредности је коришћен на начин да се предвиђа вредност операнада погодних инструкција. Притом су уочена четири типа погодних инструкција (логичко *и*, логичко *или*, инструкција тестирања и инструкција упоређивања) код којих је могуће добити исправан резултат иако један операнд има непрецизну вредност. Описани су и операнди који су погодни за предвиђање, а то су они чије вредности потичу из меморије. Представљена су два аналитичка модела извршавања инструкција са предвиђањем вредности операнада. Један модел извршавање наставља само у случају тачно предвиђене вредности операнда, док други омогућава да се извршавање настави и у случају непрецизно предвиђене вредности операнда, уколико је резултат инструкције исправан. Представљени модели су евалуирани и упоређени коришћењем стандардних тестова процесора *SPEC* и *EEMBC*. За потребе евалуације коришћена су два симулатора. Симулатор отвореног кода *Gem5* је надограђен како би се формирали трагови извршавања поменутих тестова. Други симулатор који је коришћен јесте *VPSim* симулатор, а који је развијен у оквиру ове докторске дисертације. Он служи за прикупљање статистике извршавања са предвиђањем вредности операнада спроводећи траговима вођене симулације. Симулатор *VPSim* имплементира седам предиктора вредности, од чега су четири опште позната и три најбоље пласирана са светског такмичења у предвиђању вредности. На основу прикупљене статистике извршавања, извршена је евалуација представљених аналитичких модела извршавања и потврђене су полазне хипотезе ове докторске дисертације.

Резултати симулација су показали да је прецизност тачно погођеног резултата инструкције на основу предвиђене вредности операнда већа од прецизности тачно погођене вредности операнда. Та разлика се креће од 0,8% до 44% у зависности од предиктора. Евалуацијом представљених аналитичких модела извршавања и на основу резултата симулација одређени су услови под којима модел који укључује и непрецизно предвиђене вредности може постизати боље очекивано време извршавања од модела са тачно предвиђеним вредностима. Услови су дефинисани у виду неједнакости која мора бити задовољена, а која укључује просечно време извршавања инструкције, време потребно за опоравак услед лошег предвиђања, време потребно да се дохвати податак из меморије (кашњење меморије) и поменуту разлику прецизности. За неке вредности поменутих

параметара неједнакост увек важи, што значи да модел са непрецизним вредностима постиже боље очекивано време извршавања. Такође, на основу резултата симулација закључено је да модел са непрецизно предвиђеним вредностима постиже боље очекивано време извршавања уколико је однос времена потребног за опоравак услед лошег предвиђања и просечног трајања једне инструкције већи од десет.

На крају се могу сумирати главни доприноси ове докторске дисертације који су остварени у следећим правцима. Преглед и исцрпна анализа постојећих механизама предвиђања вредности који обухвата начине њихових функционисања као и начине опоравка процесора услед лошег предвиђања. Опис карактеристика инструкција и њихових операнада који су погодни за предвиђање у случајевима непрецизно предвиђених вредности. Дефинисана два аналитичка модела извршавања са предвиђањем операнада, једног који представља извршавање само са тачно предвиђеним операндима и другог који представља извршавање и са тачно и са непрецизно предвиђеним операндима уколико је резултат инструкције исправан. Надоградња симулатора *Gem5* модулом за генерисање трагова извршавања као и скуп трагова извршавања који су формирани. Имплементација новог симулатора за потребе извршавања инструкција на основу дефинисаних аналитичких модела. У склопу симулатора су имплементирани постојећи механизми предвиђања. Евалуација дефинисаних модела извршавања и дефинисање услова када извршавање према моделу са непрецизно предвиђеним операндима постиже боље време извршавања од извршавања према моделу са тачно предвиђеним операндима на основу прецизности постојећих предиктора вредности.

Спроведено истраживање у оквиру ове докторске дисертације је приказало начин на који се могу искористити механизми за предвиђање вредности, а који омогућава да се у неким ситуацијама извршавање настави и у случају непрецизно предвиђених вредности. Резултати истраживања истичу да је могуће доћи до тачног резултата инструкције у неким ситуацијама и са непрецизно предвиђеним вредностима операнада чиме се избегава опоравак услед лошег предвиђања. На основу тога отвара се нови пут за истраживање механизама предвиђања вредности, а који ће се усредсредити на добијање тачног резултата инструкције, без обзира на то да ли је вредност операнда можда непрецизно предвиђена. У оквиру ове дисертације коришћени су постојећи предиктори вредности који предвиђају комплетну вредност. На основу представљених резултата ове дисертације даљи пут истраживања може водити ка прављењу новог предиктора вредности, али који не би предвиђао комплетну вредност, већ само неке њене делове који утичу на резултат извршавања. Такође, даљи правци истраживања се могу оријентисати на анализу могућности имплементације представљених модела извршавања на конкретним процесорима.

Литература

[1] J. I. Rojas-Sola, G. Del Río-Cidoncha, A. Fernández-De La, P. Sarriá, and V. Galiano-Delgado, "Blaise pascal's mechanical calculator: Geometric modelling and virtual reconstruction," *mdpi.comJI Rojas-Sola, G del Río-Cidoncha, A Fernández-de la Puente Sarriá, V Galiano-DelgadoMachines, 2021•mdpi.com*, 2021, doi: 10.3390/machines9070136.

[2] E. Kim, B. T.-S. American, and undefined 1999, "Ada and the first computer," *JSTOR*, Accessed: Oct. 04, 2023. [Online]. Available: https://www.jstor.org/stable/26058246

[3] H. H. Goldstine and A. Goldstine, "The Electronic Numerical Integrator and Computer (ENIAC)," *The Origins of Digital Computers*, pp. 359–373, 1982, doi: 10.1007/978-3-642-61812-3_29.

[4] J. Von Neumann and M. D. Godfrey, "First Draft of a Report on the EDVAC," *IEEE Annals of the History of Computing*, vol. 15, no. 4, pp. 27–75, 1993, doi: 10.1109/85.238389.

[5] R. Eigenmann, D. L.-W. E. of Electrical, and undefined 1998, "Von neumann computers," *public.callutheran.edu*, 1998, Accessed: Oct. 03, 2023. [Online]. Available: http://public.callutheran.edu/~reinhart/CSC521/Week3/VonNeumann.pdf

[6] G. O'Regan, "Harvard Mark 1 Computer," *The Innovation in Computing Companion*, pp. 147–149, 2018, doi: 10.1007/978-3-030-02619-6_30.

[7] G. E. Moore, "Cramming more components onto integrated circuits, Reprinted from Electronics, volume 38, number 8, April 19, 1965, pp.114 ff.," *IEEE Solid-State Circuits Society Newsletter*, vol. 11, no. 3, pp. 33–35, Feb. 2009, doi: 10.1109/N-SSC.2006.4785860.

[8] J. Hennessy and D. Patterson, "Computer Architecture - 6th Edition," p. 936, 2017, Accessed: Oct. 06, 2023. [Online]. Available: https://www.elsevier.com/books/computer-architecture/hennessy/978-0-12-811905-1

[9] P. Ross, "Why CPU Frequency Stalled," *IEEE Spectr*, vol. 45, no. 4, p. 72, 2008, doi: 10.1109/MSPEC.2008.4476447.

[10] "Intel® Core™ i9-13900KS Processor." Accessed: Oct. 06, 2023. [Online]. Available: https://www.intel.com/content/www/us/en/products/sku/232167/intel-core-i913900ks-processor-36m-cache-up-to-6-00-ghz/specifications.html

[11] J. Shalf, "The future of computing beyond Moores Law," *Philosophical Transactions of the Royal Society A*, vol. 378, no. 2166, Mar. 2020, doi: 10.1098/RSTA.2019.0061.

[12] S. Mittal, "A survey of techniques for dynamic branch prediction," *Concurr Comput*, vol. 31, no. 1, p. e4666, Jan. 2019, doi: 10.1002/CPE.4666.

[13] F. Gabbay and A. Mendelson, "Using value prediction to increase the power of speculative execution hardware," *ACM Transactions on Computer Systems (TOCS)*, vol. 16, no. 3, pp. 234–270, Aug. 1998, doi: 10.1145/290409.290411.

[14] T. F. Chen and J. L. Baer, "Effective Hardware-Based Data Prefetching for High-Performance Processors," *IEEE Transactions on Computers*, vol. 44, no. 5, pp. 609–623, 1995, doi: 10.1109/12.381947.

[15] R. J. Eickemeyer and S. Vassiliadis, "Load-instruction unit for pipelined processors," *IBM J Res Dev*, vol. 37, no. 4, pp. 547–564, 1993, doi: 10.1147/RD.374.0547.

[16] J. Hennessy, N. Jouppi, F. Baskett, and J. Gill, "MIPS: A VLSI Processor Architecture," *VLSI Systems and Computations*, pp. 337–346, 1981, doi: 10.1007/978-3-642-68402-9_37.

[17] M. H. Lipasti, C. B. Wilkerson, and J. P. Shen, “Value locality and load value prediction,” *Comput Archit News*, vol. 24, no. Special Issu, pp. 138–147, 1996, doi: 10.1145/237090.237173.
[18] S. A. Przybylski, *Cache and memory hierarchy design : a performance-directed approach*. Morgan Kaufmann Publishers, 1990. Accessed: Nov. 20, 2023. [Online]. Available: http://www.sciencedirect.com:5070/book/9780080500591/cache-and-memory-hierarchy-design
[19] U. Radenković, M. Mićović, and Z. Radivojević, “Evaluation and Benefit of Imprecise Value Prediction for Certain Types of Instructions,” *Electronics 2023, Vol. 12, Page 3568*, vol. 12, no. 17, p. 3568, Aug. 2023, doi: 10.3390/ELECTRONICS12173568.
[20] S. A. McKee, “Reflections on the memory wall,” *2004 Computing Frontiers Conference*, pp. 162–167, 2004, doi: 10.1145/977091.977115.
[21] “Championship Value Prediction (CVP).” Accessed: Jul. 04, 2023. [Online]. Available: https://microarch.org/cvp1/
[22] HennessyJohn *et al.*, “MIPS,” *ACM SIGMICRO Newsletter*, Oct. 1982, doi: 10.1145/1014194.800930.
[23] I. Pantazi-Mytarelli, “The history and use of pipelining computer architecture: MIPS pipelining implementation,” *9th Annual Conference on Long Island Systems, Applications and Technology, LISAT 2013*, 2013, doi: 10.1109/LISAT.2013.6578243.
[24] S. Del Pino, L. Piñuel, R. A. Moreno, and F. Tirado, “Value prediction as a cost-effective solution to improve embedded processors performance,” *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, vol. 1981, pp. 181–195, 2001, doi: 10.1007/3-540-44942-6_14/COVER.
[25] F. Gabbay, “Speculative Execution based on Value Prediction Research Proposal towards the Degree of Doctor of Sciences,” 1996.
[26] R. Sheikh, H. W. Cain, and R. Damodaran, “Load value prediction via path-based address prediction: Avoiding mispredictions due to conflicting stores,” *Proceedings of the Annual International Symposium on Microarchitecture, MICRO*, vol. Part F131207, pp. 423–435, Oct. 2017, doi: 10.1145/3123939.3123951.
[27] J. González and A. González, “Memory address prediction for data speculation,” *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, vol. 1300 LNCS, pp. 1084–1091, 1997, doi: 10.1007/BFB0002857/COVER.
[28] J. L. Aragón, J. Gonzalez, J. M. García, and A. González, “Selective Branch Prediction Reversal by correlating with data values and control flow,” *Proceedings - IEEE International Conference on Computer Design: VLSI in Computers and Processors*, pp. 228–233, 2001, doi: 10.1109/ICCD.2001.955033.
[29] A. Perais, “A case for speculative strength reduction,” *IEEE Computer Architecture Letters*, vol. 20, no. 1, pp. 22–25, Jan. 2021, doi: 10.1109/LCA.2020.3048694.
[30] H. Zhou, C. Fu, E. Rotenberg, and T. Conte, “A study of value speculative execution and misspeculation recovery in superscalar microprocessors,” 2000.
[31] A. Perais and A. Seznec, “Practical data value speculation for future high-end processors,” *Proceedings - International Symposium on High-Performance Computer Architecture*, pp. 428–439, 2014, doi: 10.1109/HPCA.2014.6835952.
[32] S. Bandishte, J. Gaur, Z. Sperber, L. Rappoport, A. Yoaz, and S. Subramoney, “Focused Value Prediction,” *Proc Int Symp Comput Archit*, vol. 2020-May, pp. 79–91, May 2020, doi: 10.1109/ISCA45697.2020.00018.

[33] R. Sheikh and D. Hower, “Efficient load value prediction using multiple predictors and filters,” *Proceedings - 25th IEEE International Symposium on High Performance Computer Architecture, HPCA 2019*, pp. 454–465, Mar. 2019, doi: 10.1109/HPCA.2019.00057.
[34] T. Y. Yeh and Y. N. Patt, “Two-level adaptive training branch prediction,” *Proceedings of the Annual International Symposium on Microarchitecture, MICRO*, pp. 51–61, Sep. 1991, doi: 10.1145/123465.123475.
[35] L. Yang, L. Huang, and Z. Zheng, “Confidence Counter Modelling for Value Predictor,” *Proceedings of the Great Lakes Symposium on VLSI 2023*, pp. 221–222, Jun. 2023, doi: 10.1145/3583781.3590319.
[36] F. Gabbay and A. Mendelson, “Can program profiling support value prediction?,” *Proceedings of the Annual International Symposium on Microarchitecture*, pp. 270–280, 1997, doi: 10.1109/MICRO.1997.645817.
[37] Y. Sazeides and J. E. Smith, “The predictability of data values,” pp. 248–258, Nov. 2002, doi: 10.1109/MICRO.1997.645815.
[38] Y. Sazeides and J. E. Smith, “Implementations of context based value predictors,” 1997, Accessed: Jul. 04, 2023. [Online]. Available: https://citeseerx.ist.psu.edu/document?repid=rep1&type=pdf&doi=3a95620ade477b089ae392395a0d19653e822c38
[39] K. Kalaitzidis and A. Seznec, “Leveraging Value Equality Prediction for Value Speculation,” *ACM Transactions on Architecture and Code Optimization (TACO)*, vol. 18, no. 1, Dec. 2020, doi: 10.1145/3436821.
[40] U. Radenković, M. Mićović, and Z. Radivojević, “A mechanism for mitigation of branch prediction calculation latency,” 2020, Accessed: Nov. 14, 2023. [Online]. Available: http://www.eventiotic.com/eventiotic/files/Papers/URL/ffe03f4d-8944-4eaf-a21c-4d9ab1c0d735.pdf
[41] D. A. Jimenez, S. W. Keckler, and C. Lin, “The impact of delay on the design of branch predictors,” pp. 67–76, Nov. 2002, doi: 10.1109/MICRO.2000.898059.
[42] W. Jin, F. Shi, Q. Song, and Y. Zhang, “A novel architecture for ahead branch prediction,” *Front Comput Sci*, vol. 7, no. 6, pp. 914–923, Dec. 2013, doi: 10.1007/S11704-013-2260-X/METRICS.
[43] R. Bhargava and L. K. John, “Latency and energy aware value prediction for high-frequency processors,” pp. 45–56, Jun. 2002, doi: 10.1145/514191.514201.
[44] M. H. Lipasti and J. P. Shen, “Exceeding the dataflow limit via value prediction,” *Proceedings of the Annual International Symposium on Microarchitecture*, pp. 226–237, 1996, doi: 10.1109/MICRO.1996.566464.
[45] T. Nakra, R. Gupta, and M. Lou Soffa, “Global context-based value prediction,” *IEEE High-Performance Computer Architecture Symposium Proceedings*, pp. 4–12, 1999, doi: 10.1109/HPCA.1999.744311.
[46] A. Seznec, “Exploring value prediction with the eves predictor,” in *1st Championship Value Prediction*, 2018.
[47] Y. Ishii, “Context-base computational value prediction with value compression,” in *1st Championship Value Prediction*, 2018.
[48] K. Koizumi, K. Hiraki, and M. Inaba, “H3VP: History based highly reliable hybrid value predictor,” in *1st Championship Value Prediction*, 2018.
[49] L. Yang *et al.*, “Stride Equality Prediction for Value Speculation,” *IEEE Computer Architecture Letters*, vol. 21, no. 2, pp. 57–60, 2022, doi: 10.1109/LCA.2022.3195411.

[50] BurtscherMartin, “An improved index function for (D)FCM predictors,” *ACM SIGARCH Computer Architecture News*, vol. 30, no. 3, pp. 19–24, Jun. 2002, doi: 10.1145/571666.571677.
[51] B. Goeman, H. Vandierendonck, and K. De Bosschere, “Differential FCM: Increasing value prediction accuracy by improving table usage efficiency,” *IEEE High-Performance Computer Architecture Symposium Proceedings*, pp. 207–216, 2001, doi: 10.1109/HPCA.2001.903264.
[52] S. M.-W. T. N. TN-36, D. Equipment, and undefined 1993, “Combining Branch Predictors,” *cir.nii.ac.jp*, Accessed: Nov. 04, 2023. [Online]. Available: https://cir.nii.ac.jp/crid/1573105975379042304
[53] A. Seznec and P. Michaud, “A case for (partially) tagged geometric history length branch prediction,” *The Journal of Instruction-Level Parallelism*, vol. 8, p. 23, 2006, Accessed: Jul. 04, 2023. [Online]. Available: https://inria.hal.science/hal-03408381
[54] N. Riley and C. Zilles, “Probabilistic counter updates for predictor hysteresis and stratification,” *Proceedings - International Symposium on High-Performance Computer Architecture*, vol. 2006, pp. 111–121, 2006, doi: 10.1109/HPCA.2006.1598118.
[55] V. Saravanan, K. D. Pralhaddas, D. P. Kothari, and I. Woungang, “An optimizing pipeline stall reduction algorithm for power and performance on multi-core CPUs,” *Human-centric Computing and Information Sciences*, vol. 5, no. 1, pp. 1–13, Jan. 2015, doi: 10.1186/S13673-014-0016-8/FIGURES/4.
[56] P. Lotfi-Kamran, A. M. Rahmani, A. A. Salehpour, A. Afzali-Kusha, and Z. Navabi, “Stall power reduction in pipelined architecture processors,” *Proceedings of the IEEE International Frequency Control Symposium and Exposition*, pp. 541–546, 2008, doi: 10.1109/VLSI.2008.34.
[57] K. Skadron, M. Martonosi, D. I. August, M. D. Hill, D. J. Lilja, and V. S. Pai, “Challenges in computer architecture evaluation,” *Computer (Long Beach Calif)*, vol. 36, no. 8, pp. 30–36, Aug. 2003, doi: 10.1109/MC.2003.1220579.
[58] S. Van Den Steen *et al.*, “Analytical Processor Performance and Power Modeling Using Micro-Architecture Independent Characteristics,” *IEEE Transactions on Computers*, vol. 65, no. 12, pp. 3537–3551, Dec. 2016, doi: 10.1109/TC.2016.2547387.
[59] “Embedded Microprocessor Benchmark Consortium.” Accessed: Dec. 11, 2023. [Online]. Available: https://www.eembc.org/
[60] “gem5: The gem5 simulator system.” Accessed: Jul. 04, 2023. [Online]. Available: https://www.gem5.org/
[61] “SPEC - Standard Performance Evaluation Corporation.” Accessed: Dec. 11, 2023. [Online]. Available: https://www.spec.org/
[62] “SPEC CPU® 2006.” Accessed: Jul. 04, 2023. [Online]. Available: https://www.spec.org/cpu2006/
[63] J. L. Henning, “SPEC CPU2006 benchmark descriptions,” *ACM SIGARCH Computer Architecture News*, vol. 34, no. 4, pp. 1–17, Sep. 2006, doi: 10.1145/1186736.1186737.
[64] “GCC 9 Release Series - GNU Project.” Accessed: Dec. 11, 2023. [Online]. Available: https://gcc.gnu.org/gcc-9/
[65] “GitHub - eembc/coremark-pro: Containing dozens of real-world and synthetic tests, CoreMark®-PRO (2015) is an industry-standard benchmark that measures the multi-processor performance of central processing units (CPU) and embedded microcrontrollers (MCU).” Accessed: Jul. 04, 2023. [Online]. Available: https://github.com/eembc/coremark-pro

[66] “GitHub - eembc/coremark: CoreMark® is an industry-standard benchmark that measures the performance of central processing units (CPU) and embedded microcrontrollers (MCU).” Accessed: Dec. 11, 2023. [Online]. Available: https://github.com/eembc/coremark
[67] J. Dongarra *et al.*, “Livermore Loops,” *Encyclopedia of Parallel Computing*, pp. 1041–1043, 2011, doi: 10.1007/978-0-387-09766-4_161.
[68] N. Binkert *et al.*, “The gem5 simulator,” *ACM SIGARCH Computer Architecture News*, vol. 39, no. 2, pp. 1–7, Aug. 2011, doi: 10.1145/2024716.2024718.
[69] J. Umeike, N. Patel, A. Manley, A. Mamandipoor, H. Yun, and M. Alian, “Profiling gem5 Simulator,” *2023 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*, pp. 103–113, Apr. 2023, doi: 10.1109/ISPASS57527.2023.00019.
[70] J. Lowe-Power *et al.*, “The gem5 Simulator: Version 20.0+,” Jul. 2020, doi: https://doi.org/10.48550/arXiv.2007.03152.
[71] Z. Sustran and J. Protic, “Migration in Hardware Transactional Memory on Asymmetric Multiprocessor,” *IEEE Access*, vol. 9, pp. 69346–69364, 2021, doi: 10.1109/ACCESS.2021.3077539.
[72] “objdump (GNU Binary Utilities).” Accessed: Dec. 13, 2023. [Online]. Available: https://sourceware.org/binutils/docs/binutils/objdump.html
[73] U. Radenković, M. Mićović, F. Hadžić, and Z. Radivojević, “A Tool for Testing Branch Predictors,” in *YUINFO 2019*, Kopaonik: Društvo za informacione sisteme i računarske mreže, 2019, pp. 77–82.
[74] E. Garza, S. Mirbagher-Ajorpaz, T. A. Khan, and D. A. Jiménez, “Bit-level perceptron prediction for indirect branches,” *Proc Int Symp Comput Archit*, pp. 27–38, Jun. 2019, doi: 10.1145/3307650.3322217.
[75] R. M. Tomasulo, “An Efficient Algorithm for Exploiting Multiple Arithmetic Units,” *IBM J Res Dev*, vol. 11, no. 1, pp. 25–33, Apr. 1967, doi: 10.1147/RD.111.0025.

Биографија аутора

Урош Раденковић рођен је 20. августа 1993. године у Јагодини, Република Србија. Основну школу „Бошко Ђуричић” у Јагодини завршио је као ђак генерације и носилац Вукове дипломе. Гимназију у Јагодини „Светозар Марковић”, природно-математичког смера, завршио је као носилац Вукове дипломе 2012. године. Током основне и средње школе учествовао је на републичким такмичењима из физике и хемије. За успехе током школовања добио је Октобарску награду града Јагодине 2007. године.

Након завршене средње школе, уписао се на Електротехнички факултет у Београду, студијски програм Електротехника и рачунарство, а дипломирао на Модулу за рачунарску технику и информатику 2016. године са просечном оценом 9,33. Дипломски рад „Унапређење визуелног дебагера за језик Capsules за развој графичких корисничких интерфејса Веб апликација” под менторством проф. др Драгана Милићева, одбранио је са оценом 10. Током основних студија био је демонстратор на неколико предмета при Катедри за рачунарску технику и информатику.

Мастер академске студије на Електротехничком факултету у Београду, на Модулу за рачунарску технику и информатику уписао је у октобру 2016. године. Мастер рад „Алат за тестирање система за предвиђање скокова код процесора са проточном обрадом” под менторством проф. др Захарија Радивојевића, одбранио је 2018. са оценом 10. Просечна оцена на мастер студијама је 10,00.

Докторске академске студије на Електротехничком факултету у Београду уписао је 2018. године и положио је све испите са просечном оценом 10,00. У свом истраживачком раду показао је изузетно интересовање за области архитектуре и организације рачунара, са посебним акцентом на механизме предвиђања скокова и предвиђања вредности. Похађао је летњу школу ACACES (*Advanced Computer Architecture and Compilation for High-performance Embedded Systems*) 2019. године. Током мастер и докторских студија објавио је као аутор или коаутор 9 научних радова на међународним и домаћим конференцијама, као и 2 рада у истакнутим међународним часописима. Био је рецензент научних радова на конференцији ETRAN и у часопису Facta Universitatis, Series: Electronics and Energetics.

Од децембра 2016. запослен је био као сарадник у настави на Електротехничком факултету у Београду. Од децембра 2018. је запослен као асистент на истом факултету и тренутно је ангажован на предметима: Објектно-оријентисано програмирање 1, Архитектура рачунара, Оперативни системи 1, Практикум из оперативних система, Архитектура и организација рачунара 1 и Микропроцесорски системи.

Током периода у ком је запослен на Електротехничком факултету учествује на више научних и комерцијалних пројеката. На пројектима је сарађивао са неколико домаћих и страних фирми као и са колегама са других катедри Електротехничког факултета и других факултета.

Изјава о ауторству

Име и презиме аутора: Урош Раденковић

Број индекса: 2018/5014

Изјављујем

да је докторска дисертација под насловом

Спекулативно извршавање инструкција

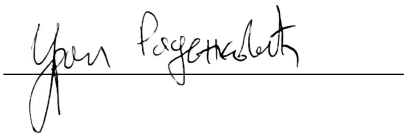
са непрецизно предвиђеним операндима

- резултат сопственог истраживачког рада;
- да дисертација ни у целини ни у деловима није била предложена за стицање друге дипломе према студијским програмима других високошколских установа;
- да су резултати коректно наведени и
- да нисам кршио ауторска права и користио интелектуалну својину других лица.

У Београду,

06.02.2024.

Потпис аутора



Изјава о истоветности штампане и електронске верзије докторског рада

Име и презиме аутора: Урош Раденковић

Број индекса: 2018/5014

Студијски програм: Рачунарска техника и информатика

Наслов рада: Спекулативно извршавање инструкција са непрецизно предвиђеним операндима

Ментор: проф. др Захарије Радивојевић

Изјављујем да је штампана верзија мог докторског рада истоветна електронској верзији коју сам предао ради похрањивања у **Дигиталном репозиторијуму Универзитета у Београду**.

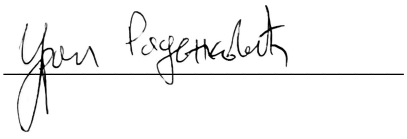
Дозвољавам да се објаве моји лични подаци везани за добијање академског назива доктора наука, као што су име и презиме, година и место рођења и датум одбране рада.

Ови лични подаци могу се објавити на мрежним станицама дигиталне библиотеке, у електронском каталогу и у публикацијама Универзитета у Београду.

У Београду,

06.02.2024.

Потпис аутора



Изјава о коришћењу

Овлашћујем Универзитетску библиотеку „Светозар Марковић“ да у Дигитални репозиторијум Универзитета у Београду унесе моју докторску дисертацију под насловом:

Спекулативно извршавање инструкција

са непрецизно предвиђеним операндима

која је моје ауторско дело.

Дисертацију са свим прилозима предао сам у електронском формату погодном за трајно архивирање.

Моју докторску дисертацију похрањену у Дигиталном репозиторијуму Универзитета у Београду и доступну у отвореном приступу могу да користе сви који поштују одредбе садржане у одабраном типу лиценце Креативне заједнице (*Creative Commons*) за коју сам се одлучио.

1. Ауторство (CC BY)
2. Ауторство – некомерцијално (CC BY-NC)
3. Ауторство – некомерцијално – без прерада (CC BY-NC-ND)
4. Ауторство – некомерцијално – делити под истим условима (CC BY-NC-SA)
5. Ауторство – без прерада (CC BY-ND)
6. Ауторство – делити под истим условима (CC BY-SA)

(Молимо да заокружите само једну од шест понуђених лиценци.

Кратак опис лиценци је саставни део ове изјаве.)

У Београду,

06.02.2024.

Потпис аутора

Урош Пајгенчевић

1. **Ауторство**. Дозвољавате умножавање, дистрибуцију и јавно саопштавање дела, и прераде, ако се наведе име аутора на начин одређен од стране аутора или даваоца лиценце, чак и у комерцијалне сврхе. Ово је најслободнија од свих лиценци.

2. **Ауторство – некомерцијално**. Дозвољавате умножавање, дистрибуцију и јавно саопштавање дела, и прераде, ако се наведе име аутора на начин одређен од стране аутора или даваоца лиценце. Ова лиценца не дозвољава комерцијалну употребу дела.

3. **Ауторство – некомерцијално – без прерада**. Дозвољавате умножавање, дистрибуцију и јавно саопштавање дела, без промена, преобликовања или употребе дела у свом делу, ако се наведе име аутора на начин одређен од стране аутора или даваоца лиценце. Ова лиценца не дозвољава комерцијалну употребу дела. У односу на све остале лиценце, овом лиценцом се ограничава највећи обим права коришћења дела.

4. **Ауторство – некомерцијално – делити под истим условима**. Дозвољавате умножавање, дистрибуцију и јавно саопштавање дела, и прераде, ако се наведе име аутора на начин одређен од стране аутора или даваоца лиценце и ако се прерада дистрибуира под истом или сличном лиценцом. Ова лиценца не дозвољава комерцијалну употребу дела и прерада.

5. **Ауторство – без прерада**. Дозвољавате умножавање, дистрибуцију и јавно саопштавање дела, без промена, преобликовања или употребе дела у свом делу, ако се наведе име аутора на начин одређен од стране аутора или даваоца лиценце. Ова лиценца дозвољава комерцијалну употребу дела.

6. **Ауторство – делити под истим условима**. Дозвољавате умножавање, дистрибуцију и јавно саопштавање дела, и прераде, ако се наведе име аутора на начин одређен од стране аутора или даваоца лиценце и ако се прерада дистрибуира под истом или сличном лиценцом. Ова лиценца дозвољава комерцијалну употребу дела и прерада. Слична је софтверским лиценцама, односно лиценцама отвореног кода.