УНИВЕРЗИТЕТ У БЕОГРАДУ

ЕЛЕКТРОТЕХНИЧКИ ФАКУЛТЕТ

Милан Б. Радуловић

ПРЕДЛОГ ПОДРШКЕ ЗА СПЕКУЛАТИВНО ИЗВРШАВАЊЕ НИТИ У CMP ПРОЦЕСОРИМА

докторска дисертација

Београд, 2014

UNIVERSITY OF BELGRADE

SCHOOL OF ELECTRICAL ENGINEERING

Milan B. Radulović

A PROPOSAL OF SUPPORT FOR THREAD LEVEL SPECULATION IN CMPs

Doctoral Dissertation

Београд, 2014

Ментор:
др Мило В. Томашевић, ванредни професор
Универзитет у Београду, Електротехнички факултет

Чланови комисије:
др Вељко Милутиновић, редовни професор
Универзитет у Београду, Електротехнички факултет

др Душан Старчевић, редовни професор
Универзитет у Београду, Факултет организационих наука

др Јован Ђорђевић, редовни професор
Универзитет у Београду, Електротехнички факултет

др Зоран Јовановић, редовни професор
Универзитет у Београду, Електротехнички факултет

Датум одбране:

Изјава захвалности:

Желио бих да се овим путем захвалим драгим људима који су несебично даровали своје драгоцјено слободно вријеме како би ми пружили значајну подршку током истраживања и рада на овој докторској дисертацији.

На самом почетку истраживања је било неопходно боравити у неком од иностраних истраживачких центара како би се обезбиједио преглед доступне научне литературе из области CMP система са подршком за спекулативно извршавање. Велику захвалност за то дугујем проф. др Предрагу-Мишу Обрадовићу, бившем Ректору Универзитета Црне Горе, који је обезбиједио потребна финансијска средства за мој боравак на Факултету за информатичко инжењерство, Универзитета у Сијени. Такође, велику захвалност дугујем и проф. др Roberto Giorgi-ју, који је омогућио мој боравак у својству гостујућег истраживача на овом факултету.

Велику захвалност дугујем и проф. емеритусу др Радивоју Поповићу, као и проф. др Yusuf Leblebici-ју и проф. др Paolo Ienne-у, са EPFL-а у Лозани, који су омогућили мој боравак на EPFL-у како бих на Катедри за микроелектронске системе и архитектуру процесора изложио резултате мог истраживања и предлоге рјешења подршке за регистарску и меморијску комуникацију у спекулативном CMP систему.

Поред тога, желио бих да искажем велику захвалност проф. др Koen De Bosschere-у са Универзитета у Ghent-у и HiPEAC организацији који су омогућили добијање стипендије за боравак на међунардоним скуповима ACACES (*Advanced Computer Architecture and Compilation for High-Performance and Embedded Systems*). Присуство на овим скуповима ми је омогућило да остварим неопходан увид у савремене резултате других истраживачких центара у пројектовању спекулативних CMP система као и да презентујем постигнуте резултате у ACACES зборницима радова.

Захваљујући учешћима на овим међународним скуповима имао сам прилику и да остварим личне контатке са истраживачима који су радили на дефинисању и пројектовању спекулативних CMP система на водећим универзитетима у свијету. Стога, користим прилику да се овдје најсрдачније захвалим проф. др Gurindar Sohi-ју са University of Wisconsin-Madison, проф. др Kunle Olukotun-у са Stanford University, проф. др Josep Torellas са University of Illinois at Urbana-Champaign, проф. др Pradip Bose-у из IBM T. J. Watson Research Center-а и проф др. Stefanos Kaxiras-у са Uppsala University, који су ми пружили добронамјерне, непроцјењиве и драгоцјене савјете за презентована предложена рјешења регистарске и меморијске комуникације у спекулативном CMP систему. Поред тога, желио бих да се захвалим проф Kaxiras-у и на времену и труду које је уложио за анализу техничких извјештаја у вези са предложеним рјешењима, као и савјетима и предлозима које је дао у циљу дефинисања будућих праваца истраживања.

Велику захвалност дугујем и др. James M. Tuck-у, са North Carolina State University, из истраживачке групе I-ACOMA, чије је дугогодишње искуство у раду са спекулативним CMP системима и симулационим окружењима било од непроцјењиве помоћи у дефинисању симулационог окружења за потребе евалуације предложених рјешења у спекулативном CMP систему.

Такође, веома сам захвалан и проф. др Вељку Милутиновићу на великој помоћи у добијању финасијске подршке за учествовање на међународном скупу, као и драгоцјеним савјетима током публиковања предлога регистарске комуникације у спекулативном CMP систему.

Највећу захвалност за помоћ током дефинисања симулационог окружења као и имплементацији предложеног рјешења спекулативног CMP система у UNISIM симулатору дугујем мом другу др Sylvain Girbal-у из Thales Research & Technologies у Паризу. Са Sylvain-ом сарађујем већ више година и за сво ово вријеме није жалио ни времена ни труда како бисмо на што ефикаснији начин извршили прилагођење симулатора UNISIM чији је он један од пројектаната.

Најсрдачније се захваљујем на помоћи мом другу, мр Александру Самарџићу, чији су савјети током рада на успостављању адекватног симулационог окружења за евалуацију предложеног рјешења спекулативног CMP система били од непроцјењиве користи за успјешност тог пројекта.

Користим прилику да се захвалим и мојим драгим пријатељима, нарочито Татијани и Горану Лучићу, на њиховој неизмјерној подршци коју су ми давали свих ових година.

Тешко је ријечима изразити сву захвалност за свесрдну и безрезервну пружену подршку, као и неизмјерно поштовање, разумјевање, стрпљење и повјерење које ми је указао мој ментор, проф. др. Мило Томашевић, како током свих ових година нашег заједничког истраживачког рада тако и у мојој стручној каријери. Са дужним поштовањем за све што сте учинили и чините за мене једно велико људско ХВАЛА од моје породице и мене лично.

На крају, захвалио бих се својој породици која је увијек била мој главни ослонац у животу, која је увијек била ту за мене да ме охрабри када је тешко и да подијели са мном све радости успјеха, без чије љубави, вјере и подршке не би било могуће постићи и остварити циљеве које сам поставио у животу.

Наслов докторске дисертације: Предлог подршке за спекулативно извршавање нити у CMP процесорима

Резиме: У овом раду се пошло од претпоставке да се ресурси вишејезгарних (CMP) процесора, уз додатну хардверску и/или софтверску подршку за спекулативно извршавање нити (TLS), могу искористити и за извршавање данас преовлађујућих секвенцијалних апликација. Међутим, поједина постојећа рјешења CMP процесора са TLS подршком захтијевају коришћење знатних додатних хардверских и софтверских ресурса за обављање специфичних операција током спекулативног извршавања секвенцијалних апликација. Стога је у овом раду урађена опсежна и веома детаљна компаративна анализа постојећих CMP система са TLS подршком при чему су посебно размотрени елементи хардверске и софтверске подршке за спекулацију на регистарском и меморијском нивоу. Са становишта подршке за регистарску комуникације постојећи системи су класификовани по начину организације скупа регистара и начину повезивања, а након тога су подробно анализирани регистарски комуникациони механизми, технике опоравка након погрешне спекулације, перформансе и скалабилност. Класификација система са становишта спекулативне меморијске комуникације је урађена према организацији меморијске хијерархије, а затим је размотрена сложеност хардверске и софтверске подршке за спекулацију. Уочени су неки потенцијални узроци за деградацију перформанси као што су неуједначен саобраћај при завршетку нити и неприлагођен алгоритам замјене у кеш меморији.

На основу спроведене анализе постојећих спекулативних CMP система, а у циљу остваривања што бољег односа цијена/перформансе и отклањања уочених неефикасности, у овој дисертацији је дефинисан предлог рјешења CMP процесора са једноставнијом TLS подршком, у којем се комуникација између спекулативних нити обавља и на регистарском и на меморијском нивоу. Предложени CMP систем за спекулативно извршавање нити припада групи CMP система са основном генеричком архитектуром надграђеном подршком и за регистарску и за меморијску комуникацију. Он се састоји од четири процесорска језгра са приватним L1 кеш меморијама за податке и инструкције повезаним заједничком магистралом са заједничком L2 кеш меморијом. Спекулативна комуникација на регистарском и меморијском нивоу се обавља преко посебних заједничких магистрала на принципима „*snoopy*“ протокола, док се спекулативна паралелизација спроводи на нивоу петљи тако да спекулативне нити одговарају појединим итерацијама петље. Идентификација нити у предложеном систему се врши уз помоћ бинарног анотатора, посебне софтверске компоненте која ради над секвенцијалним извршним кодом тако да се није потребно поновно превођење изворног кода. Поред тога, бинарни анотатор за потребе протокола који контролише регистарску комуникацију врши класификацију инструкција уписа у регистре чије се вриједности током спекулативног извршавања могу мијењати у петљи. Спекулативно произведене меморијске вриједности се држе у приватној

L1 кеш меморији, док заједничка L2 кеш меморија чува секвенцијално стање апликације. Меморијске вриједности могу бити пренесене у L2 кеш меморију након што нит постане неспекулативна. Спекулативна нит може да чита неспекулативне податке од неспекулативне нити или од претходних или наредних спекулативних нити. Поред тога, у предложеном рјешењу је имплементирана подршка за динамичко препознавање нарушавања зависности по подацима и опоравак након погрешне спекулације.

За регистарску комуникацију су предложена два протокола: основна варијанта - SIC протокол и проширена варијанта – ESIC протокол који је за разлику од SIC-а заснован на спекулативном прослеђивању потенцијално сигурних вриједности регистара. Квалитативно поређење протокола SIC и ESIC је указало на постојање потенцијалног добитка на перформансама у ESIC у односу на SIC. Овај добитак је посљедица смањеног блокирања нити потрошача и директно зависи од вјероватноће да се прослеђена, потенцијално сигурна вриједност регистра испостави као коначна. У супротном, погрешна спекулација изазива поништавање нити што може имати негативан ефекат на перформансе. Због тога се користи информација из профајлера о вјероватноћи успјешне спекулације како би поништавање нити било што рјеђе. У протоколима SIC и ESIC се користи механизам дистрибуираног одлучивања за ефикасније проналажење најближе претходне нити. Осим тога, број инвалидационих промашаја за константне регистре у протоколима SIC и ESIC је смањен коришћењем „*read-snarfing*“ технике. Поред тога, протоколи SIC и ESIC испољавају скалабилност са становишта комплексности зато што величина локалног каталога и хардверска подршка остају скоро константни са повећањем броја процесорских језгара.

Предложене су и три варијанте меморијског протокола које интегришу одржавање кохеренције кеш меморија и подршку за спекулацију. Прва варијанта, протокол SISC-WI, заснован је на стратегији поништавања и користи 13 стања за праћење статуса податка. Он омогућава само комуникацију међу нитима коју је покренуо потрошач. Да би се смањио број инвалидационих промашаја разрађена је и друга верзија протокола са имплементацијом „*read-snarfing*“ технике, SISC-WI-RS, која врши истовремену валидацију поништених копија. Трећа варијанта, SISC-WU протокол, заснован је на стратегији ажурирања и користи 9 стања за праћење статуса податка. Он омогућава комуникацију међу нитима и од стране потрошача и од стране произвођача. Поред тога, свакој ријечи у L1 кеш меморији је додат један бит који означава застарјелост ријечи. У свим варијантама меморијских протокола је предложен и механизам дистрибуиране арбитрације на магистрали за избор одговарајућег достављача података на захтјев неке нити. Такође, све три варијанте меморијских протокола користе прилагођени алгоритам замјене који приликом избора ријечи за замјену узима у обзир њено стање као примарни критеријум, умјесто уобичајеног критеријума историје приступа коришћеног код свих других система. Спекулативној нити је дозвољено да из L1 кеш меморије избаци модификовану потврђену ријеч која је једина копија

приликом операције спекулативног читања односно писања за дату ријеч. Поред тога, све ријечи које нијесу биле спекулативно прочитане или измијењене могу се сматрати кандидатима за замјену, чиме се избјегава заустављање спекулативног извршавања.

За потребе имплементационе и евалуационе анализа предложеног система, развијено је ново симулационо окружење које је засновано на модуларном симулатору UNISIM који подржава симулацију на нивоу циклуса. Као најсложенији дио укупне подршке за TLS у предложеном CMP процесору, за имплементацију је одабран протокол SISC-WI. У том циљу је извршена модификација потребних модула (модул кеш меморије, модул кеш контролера и модул системске магистрале) у библиотеци компоненти симулатора UNISIM како би се задовољили захтјеви SISC-WI протокола. Осим тога, у симулатору UNISIM је имплементиран модул за MIPS спекулативну вишејезгарну архитектуру при чему су додата три нова модула: распоређивач спекулативних нити на процесорска језгра, модул који детектује нарушавање зависности по подацима и врши поништавање нити и модул који бира добављача за тражени податак између више нити. Модул за детекцију и разјешавање погрешне спекулације успјешно разрјешава проблеме који су везани за поништавање нити, а за које у доступној литератури нијесу нађена рјешења. Развијен је и крос-компајлер заснован на радном оквиру *crosstool-NG*. На крају, имплементација протокола SISC-WI у реализованом симулационом окружењу је показала исправност концепта.

Кључне ријечи: CMP процесори, спекулативни паралелизам на нивоу нити, регистарски протокол, меморијски протокол, механизам дистрибуиране арбитрације, алгоритам замјене кеш ријечи, симулационо окружење

Научна област: Електротехника и рачунарство

Ужа научна област: Рачунарска техника и информатика

УДК број: 621.3

Title: A Proposal of Support for Thread Level Speculation in CMPs

Abstract: In this research it was assumed that resources of chip multiprocessors (CMP), with additional hardware and/or software support for Thread Level Speculation (TLS) can be exploited for the execution of sequential applications which prevail today. However, some existing CMPs with TLS support require substantial additional hardware and software resources to perform specific operations during speculative execution of sequential applications. Therefore, an extensive and very detailed comparative analysis of existing speculative CMPs is performed focusing on hardware and software support for the speculation on both register and memory levels. According to the register level communication support, existing systems are classified by the organization of register file and interconnection topology. Then, a comparative analysis of registration communication mechanisms, misspeculation recovery techniques, and performance and scalability, are presented. According to the memory level communication support, existing systems are classified by the organization of the memory hierarchy, and then, the complexity of hardware and software support for speculation is discussed. Some potential causes for performance degradation such as burst traffic on thread commit and inappropriate replacement algorithm for cache memories are considered.

Based on the analysis of existing speculative CMPs, in order to achieve a better cost/performance and to eliminate the observed inefficiencies, a CMP with a simpler TLS support, which enables the communication between speculative threads on both register and the memory level, is proposed in this dissertation. The proposed speculative CMP belongs to the systems with basically generic architecture enhanced with support for register and memory speculative communication. It consists of four processing cores with private data and instruction L1 caches which are connected by a shared bus to a shared L2 cache. The speculative communication at the register and memory level is done through corresponding shared buses following the principles of *snoopy* protocols. Speculative parallelization is implemented at the loop level and speculative threads correspond to the loop iterations. Speculative threads are identified using the binary annotator, the special software component that operates on a sequential executable code without need for source code recompilation. In addition, this tool annotates write instructions to loop-alive registers for the purpose of protocol that controls the register level communication. Speculatively produced memory values are kept in private L1 caches, while shared L2 cache stores the sequential state. Memory values can be transferred to the L2 cache only after a thread becomes non-speculative. The speculative thread can read data from non-speculative thread or some preceding or subsequent speculative thread. The proposed solution implements a support for dynamic identification of data dependence violations and recovery from misspeculation.

Two protocols are proposed for register communication: the basic version – the SIC protocol, and its extended version - the ESIC protocol which is, unlike the SIC, based on speculative forwarding of potentially safe register values. Qualitative comparison of

the SIC and ESIC protocols indicate a potential performance gain in the ESIC compared to the SIC protocol. This gain is a result of reduced blocking of consumer threads and it directly depends on the probability that forwarded, a potentially safe, register value turns out to be final. Otherwise, the misspeculation causes thread squashing which may have hurt the performance. By using profiler provided information on probability of successful speculation it is possible to decrease the frequency of thread squashing. The SIC and ESIC protocols employ a mechanism of distributed arbitration for efficiently finding the data supplier threads. Besides, the number of invalidation misses for non loop-alive registers in SIC and ESIC protocols is reduced by using *read-snarfing* technique. As for the complexity, both protocols are scalable since the size of the local directory and hardware support remain almost constant while increasing the number of processor cores.

Three variants of the memory protocols that integrate the cache coherence with the support for speculation are proposed. The first variant, the SISC-WI protocol, is invalidation-based and uses 13 states. It allows consumer-initiated communication between speculative threads only. The second variant, the SISC-WI-RS protocol, tends to reduce the number of invalidation misses using *read-snarfing* technique for simultaneous validation of invalidated copies. The third variant, the SISC-WU protocol, is update-based and uses 9 states. It allows both producer- and consumer-initiated communication between speculative threads. In addition to state bits, there is a bit per each word in L1 cache which indicates if the word is stale. Again, distributed arbitration on the shared bus is proposed for the selection of appropriate suppliers of data at a thread request. Also, all three memory protocols employ a modified cache memory replacement algorithm which first takes into account a word state, instead of history-based algorithms in all other systems. A speculative thread is allowed to evict a modified committed word from the L1 cache during speculative read or write operations to this word since it is the only copy. In addition, all the words that are not speculatively read or modified can be considered for replacement, thus avoiding the unnecessary stalling of speculative execution.

A new simulation environment based on the UNISIM modular cycle-level simulator is developed for the purpose of implementation and evaluation analysis of the proposed support. Then, the most complex part of the overall proposed TLS support, the SISC-WI protocol, is chosen for implementation in such a simulation environment. In order to make this possible it was necessary to extensively modify some existing standard UNISIM simulator modules: the cache memory, the cache controller and the system bus. On top of this, a speculative multi-core MIPS architecture is simulated and three new modules are included: Scheduler, Squash Arbiter and Supplier Arbiter. The Squash Arbiter module for the detection and resolving of misspeculation successfully solved some problems related to thread squashing that are not considered in the open literature. Besides, the cross-compiler based on the crosstool-NG framework is developed. Finally, the implementation of the SISC-WI protocol in the simulation environment proved the validity of the concept.

Key words: CMP, thread level speculation, register-level protocol, memory-level protocol, mechanism of distributed arbitration, replacement algorithm, simulation environment

Scientific area: Electrical and Computer Engineering

Scientific subarea: Computer Engineering

UDC number: 621.3

САДРЖАЈ

Глава 1 Увод

Све до прије десетак година процесори са једним суперскаларним језгром су били доминантни у технологији процесора. Међутим, ограничења у искоришћењу паралелизма на нивоу инструкције, огромна сложеност дизајна која процесе пројектовања и тестирања чини врло захтјевним, као и проблеми са довођењем потребне енергије и одвођењем дисипиране топлоте, врло су успорили раст перформанси процесора. Све то је у последњој декади условило потпуну промјену развојне филозофије у пројектовању процесора. Средином прошле деценије појавила се нова парадигма у пројектовању процесора којом се превазилазе ови проблеми. Она се заснива на процесорима са више језгара на чипу (CMP – *chip multiprocessors* или *multicore processors*). CMP процесор може ефикасно да искористи и даље растући број транзистора на једном чипу тако што се на чип смјешта више релативно једноставних процесорских језгара умјесто једног сложеног језгра. На тај начин се постиже већа укупна процесна снага. Поред тога, CMP се може користити и као градивни блок у већим скалабилним мултипроцесорским системима до нивоа суперрачунара.

Док је у суперскаларним процесорима преовладавао паралелизам на нивоу инструкције, CMP процесори ефикасно подржавају паралелизам на нивоу нити у оквиру апликације, као и истовремено извршавање више апликација. Зато се данас CMP процесори широко користе као платформа за извршавање апликација као што су базе података, Web сервери, мултимедијални кориснички интерфејси, захтјевне научне и инжењерске апликације са доста инхерентног паралелизма различитог нивоа грануларности.

Међутим, ресурсе CMP процесора треба искористити и приликом извршавања данас, ипак, преовлађујућих секвенцијалних апликација. С обзиром да су преводиоци за паралелизацију успјешни само за ограничену класу апликација,

јавила се идеја да се у циљу убрзавања секвенцијалних програма у CMP процесоре дода подршка за спекулативно извршавање на нивоу нити - TLS (*thread-level speculation*). TLS техника омогућава да се у вријеме превођења или извршавања креирају нити из секвенцијалног изворног или бинарног кода које могу да се спекулативно извршавају паралелно, чак и ако између њих постоје потенцијалне зависности по подацима. Додатна хардверска и/или софтверска подршка надгледа исправност спекулативног извршавања, па уколико се потенцијалне зависности не остваре у вријеме извршавања, нити се нормално завршавају и перформансе се побољшавају због оствареног паралелизма. Међутим, у случају откривања и нарушавања зависности, нити код којих је детектована погрешна спекулација се поништавају заједно са свим ефектима које су произвеле. Затим се оне поново извршавају са исправним расположивим подацима. Све ово намеће потребе за обично прилично сложеном хардверском и софтверском подршком. У спекулативним CMP системима коректно управљање спекулативним извршавањем се обично интегрише са протоколима за кохеренцију. Синхронизација и комуникација између спекулативних нити може бити остварена на регистарском или/и на меморијском нивоу.

CMP системи са подршком за TLS могу се подијелити у три групе. Прву групу чине CMP системи који су у потпуности оријентисани на искоришћење спекулативног паралелизма, гдје спекулативне нити могу комуницирати и на регистарском и на меморијском нивоу. Неки од ових система, као што су Multiscalar, Multiplex, Trace, и SM процесор, имају хардверску подршку која им омогућава да остварују високе перформансе тако што могу обрађивати секвенцијалне извршне програме без потребе за поновним превођењем изворног програмског кода. Међутим, када ови CMP системи извршавају паралелне апликације или више апликација истовремено велики дио хардверске и софтверске подршке за TLS остаје неискоришћен. Другу групу представљају генерички CMP системи, као што су Hydra, STAMPede и SP, које имају минималну подршку за спекулативно извршавање, а комуникација између спекулативних нити се одвија искључиво на меморијском нивоу што упрошћава њихово пројектовање, али потреба за поновним превођењем изворног програмског кода представља њихов главни недостатак. IACOMA, Atlas и NEKO системи су представници треће групе. Они комбинују најбоље карактеристике претходне двије групе као што су: подршка за комуникацију између спекулативних нити и на регистарском и на меморијском нивоу и рад на секвенцијалним извршним програмима без потребе за поновним превођењем изворног програмског кода као код прве групе CMP система, а притом су генеричке CMP архитектуре и имају једноставнију хардверску подршку за спекулативно извршавање као код друге групе.

Међутим, детаљна анализа постојећих решења дата у овом раду показује да ова рјешења захтијевају коришћење знатних додатних хардверских и софтверских ресурса за обављање специфичних операција приликом спекулације на

регистарском и меморијском нивоу, па би од интереса било да се они смање. Поред тога, могу се уочити и неке неефикасности приликом завршавања нити када се генерише велики саобраћај при потврђивању ефеката спекулације. На крају, има мјеста размишљати и о побољшању алгоритама замјене у кеш меморијама оваквих процесора.

Због тога је у овом раду предложено комплетно рјешење подршке за спекулативно извршавање на нивоу нити које ће по свом приступу бити слично рјешењима из треће групе зато што та рјешења имају једноставнију хардверску и софтверску подршку за спекулативно извршавање. Оно је предвиђено за CMP систем са четири језгра повезана заједничком меморијом, али, с обзиром да је засновано на принципима протокола прилагођених заједничкој магистрали, може да се скалира на више језгара. Предлог омогућава комуникацију између спекулативних нити и на регистарском и на меморијском нивоу да би се остварили што боље перформансе. Такав спекулативни CMP процесор, као једноставније и ефикасније рјешење са становишта односа цијена/перформансе, треба да посједује рјешења која ће минимизовати претходно уочене проблеме код CMP процесора и на тај начин постићи побољшање у односу на постојећа рјешења.

За подршку комуникације на нивоу регистара предложена су два протокола. Основна варијанта, SIC протокол, просљеђује само сигурне вриједности, док друга варијанта, ESIC протокол, има агресивнији спекулативни приступ и просљеђује и вриједности које су потенцијално сигурне. Оба протокола се заснивају на заједничкој магистрали и карактерише их мали број стања у којима се податак може наћи.

Комуникација на нивоу меморијског система при спекулативном извршавању је подржана у виду двије алтернативне варијанте протокола који интегришу кохеренцију кеш меморија и спекулацију и који су, такође, засновани на принципима "*snoopy*" протокола за заједничку магистралу. Основна разлика им је начин одржавања кохеренције па SISC-WI протокол користи стратегију инвалидације, док SISC-WU протокол примјењује стратегију ажурирања.

Да би се могле извршити процјене сложености и ефикасности овог предлога било је потребно извршити имплементацију. За те сврхе је развијено погодно симулационо окружење засновано на симулатору UNISIM који омогућава врло прецизну симулацију на нивоу циклуса. MIPS архитектура је искоришћена за процесорско језгро, а затим је имплементирана подршка за SISC-WI протокол модификацијом постојећих и додавањем нових симулаторских модула. Поред тога, коришћењем *crosstool-NG* софтверског алата је направљен крос-компајлер који служи за генерисање бинарних извршних кодова апликација које се спекулативно извршавају у симулатору.

Након ове, уводне главе, у наредној другој глави су размотрене предности CMP процесора у односу на суперскаларне процесоре у складу са садашњим и будућим трендовима у развоју хардвера и софтвера посебно у домену коришћења

паралелизма на нивоу нити. Затим је представљена техника спекулативе паралелизације на нивоу нити као начин бржег извршавања секвенцијалних апликација. Дат је и кратак преглед постојећих система из отворене литературе са подршком за спекулативно извршавање нити по усвојеној класификацији. Ова глава се завршава са поређењем система у погледу начина идентификације нити и домена спекулације.

У трећој глави је прво предложена класификација спекулативних CMP процесора са регистарском комуникацијом. Затим је урађена упоредна анализа ових система са аспеката регистарских комуникационих механизама и техника, опоравка након погрешне спекулације, перформанси ових механизама као и наметнутих ограничења у погледу скалабилности. Излагања су илустрована погодним табелама.

На сличан организациони начин је у четвртој глави дат преглед подршке за меморијску комуникацију у постојећим академским и комерцијалним системима. Након предложене класификације система са овог становишта, представљена је упоредна анализа која разматра сложеност хардверске и софтверске подршке меморијских протокола за кохеренцију и спекулацију. Притом су уочене и истакнуте неке неефикасности које су посљедица неуједначеног саобраћаја при завршавању нити и неадекватне стратегије замјене у кеш меморијама.

Претходна анализа је послужила да се, након поставке проблема и циља истраживања, у петој глави дефинише предлог CMP процесора са подршком за спекулативно извршавање на нивоу нити. Дефинисана је структура система, описан поступак спекулације, а затим укратко представљени и најављени предлози протокола за подршку спекулацији и на регистарском и на меморијском нивоу.

У шестој глави су детаљно представљени предложени протоколи за подршку спекулацији на регистарском нивоу, основна верзија SIC протокол и напредна верзија ESIC протокол, кроз опис хардверске инфраструктуре, софтверске подршке, као и акција оба протокола. Дато је и квалитативно поређење предложених протокола за регистарску комуникацију.

Инфраструктура предложеног SISC-WI протокола за подршку спекулацији на меморијском нивоу који је заснован на стратегији поништавања, као и његова варијанта која имплементира „*read-snarfing*“ технику, су детаљно представљене у седмој глави кроз опис стања, трансакција и сигнала на магистрали, акција и прелаза између стања. Такође, описана је и модификована стратегија замјене кеш ријечи која узима у обзир њено стање ријечи у кеш меморији као примарни критеријум.

На сличан начин је у наредној, осмој глави представљена предложена алтерантивна варијанта подршке за меморијску комуникацију, SISC-WU протокол, који је заснован на стратегији ажурирања, кроз опис стања, трансакција и сигнала на магистрали, акција и прелаза стања.

Симулационо окружење за имплементацију елемената предложене подршке за спекулативно извршавање је описано у деветој глави. Прво је детаљно образложен избор симулационог окружења, а затим су дате и његове основне карактеристике. Након тога је представљено окружење и начин на који се генеришу спекулативни бинарни кодови за предложено CMP рјешење. Размотрене су и потребне функционалности симулатора за подршку спекулативном извршавању.

Имплементациони детаљи у вези са интеграцијом SISC-WI протокола у UNISIM симулатору су представљени у десетој глави. Посебан нагласак је стављен на промјене које је било неопходно урадити на постојећим модулима за процесорско језгро, кеш контролер и магистралу, као и на увођење нових модула (*Scheduler*, *Squash Arbiter*, *Supplier Arbiter*), како би се обезбиједила потребна подршка спекулативном извршавању.

У закључку рада су резимиране активности спроведене у овом истраживању, истакнути остварени доприноси и резултати истраживања, а размотрени су и будући правци за даље истраживање.

Глава 2 Спекулативни CMP процесори

У претходној декади технолошки трендови су проузроковали значајне промјене у парадигмама пројектовања процесора. Тако су вишејезгарни (*multicore*) процесори или мултипроцесори на чипу (*chip multiprocessors* - CMP) преовладали над сложеним суперскаларним процесорима као боље рјешење у коришћењу све већег броја транзистора на чипу и превазилажења технолошких ограничења. Поред нижег нивоа паралелизма на нивоу инструкције (*Instruction Level Parallelism* - ILP) они могу да користе и виши ниво паралелизма на нивоу нити (*Thread Level Parallelism* - TLP). Да би се постигла пуна ефикасност извршавања постојећих секвенцијалних апликација на CMP процесорима користи се техника спекулативног извршавања нити (*Thread Level Speculation* - TLS). Она омогућава да се чак и нити које су потенцијално зависне по подацима извршавају паралелно захваљујући посебној хардверској и софтверској подршци која динамички детектује и контролише остваривање зависности гарантујући коректну семантику секвенцијалног извршавања. Такви системи се називају спекулативни CMP мултипроцесори.

2.1. Промјена принципа пројектовања процесора

Напредак у пројектовању и перформанси рачунарских система је у протеклим деценијама био заснован на устаљеним технолошким трендовима. Показало се да густина транизистора на чипу расте око 35% годишње, док величина чипа расте 10-20% годишње, тако да укупан број транзистора на чипу расте за око 40-55% годишње. Ова уочена правилност да се број транзистора дуплира на сваких 18 до 24 мјесеца назива се *Moore*-ов закон. Данас је број транзистора на чипу већ

прешао милијарду [HENN2012]. Поред тога, добитак у перформансама је, до скоро, у великој мјери остваривaн и повећањем фреквенције рада процесора.

При таквим технолошким трендовима пројектанти процесора су били суочени са изазовом како да их на најбољи начин искористе. Један уобичајен начин је коришћење додатних транзистора да се повећа локалност приступа подацима, па се тако константно повећавао број нивоа и капацитет хијерархије кеш меморија. Други начин је био повећавање паралелизма на нивоу инструкције - ILP, као што је повећање ширине обраде, увођење технике проточне обраде, итд.

Најнапреднији једнојезгарни процесори (нпр., Intel Pentium-Pro, AMD, MIPS R10K, IBM PowerPC, Compaq Alpha) имају суперскаларну архитектуру која дозвољава да се више од једне инструкције издаје на извршавање у једном циклусу и тако повећавају ниво ILP [JOHN1990]. Код статичких (*in-order issue*) суперскалара преводилац генерише оптимални инструкцијски прозор који дозвољава да се више независних инструкција издаје у сваком циклусу. Још виши ниво ILP се може постићи у вријеме извршавања код динамичких (*out-of-order*) суперскалара. Ови процесори користе додатни хардвер да идентификују независне инструкције из динамичког инструкцијског прозора које могу симултано да се извршавају.

Међутим, показало се да паралелизам на нивоу инструкције има своја ограничења и да суперскаларни процесори који издају више од четири инструкције истовремено не могу да остваре иоле веће побољшање (ILP *wall*). С друге стране, сложеност потребне додатне хардверске логике расте са квадратом броја инструкција које се паралелно издају [OLUK2007]. Сложени централизовани суперскаларни дизајни узрокују и већа времена простирања сигнала у чипу као и већу осјетљивост на физичке полупроводничке ефекте. Сами поступци пројектовања и верификације су много захтијевнији и скупљи. Поред свега, показало се да су овакви дизајни изузетно захтијевни у погледу потребне енергије и начина одвођења дисипиране енергије са чипа (*power wall*).

Сви ови комбиновани ефекти су утицали да тренд успоравања *Moore*-овог закона по којем су перформансе расле за око 50% сридином прошле деценије падне на око 20% и указали да екстензивно коришћење ILP није више ефективан начин да се дође до бољих перформанси. То је условило промјену парадигми и довело до новог алтернативног приступа у пројектовању процесора за ефикасно коришћење све већег броја транзистора на чипу – мултијезгарних или CMP процесора. Један CMP се састоји од неколико много простијих суперскаларних језгара интегрисаних са дијелом меморијске хијерархије у истом чипу. Развој CMP процесора је условљен не само технолошким трендовима него и помацима на софтверском плану (апликације са великим степеном паралелизма [WALL1993]). Постојање више језгара омогућује TLP – паралелизам на нивоу нити.

Мултипроцесорски системи су одавно коришћени као рјешење да се превазиђу ограничене процесне снаге једнопроцесорских система. Главни проблеми код

мултипроцесорских система су релативно велике латенције приступа физички дистрибуираном меморијском систему и потреба за међупроцесорском комуникацијом и синхронизацијом. У данашње вријеме, када су мултипроцесори ушли у чип, они нуде неке значајне предности у односу на суперскаларе.

Кашњења по интерконекцијама унутар чипа постају све значајнија како CMOS гејтови постају бржи, а чипови физички већи [CARL2003]. У поређењу са напредним сложеним суперскаларом у CMP процесору се смањују кашњења по жицама и дужине критичног пута на чипу с обзиром на мања и простија језгра која заузимају релативно мало простора на већем чипу. Једино рјеђе коришћене везе између језгара могу имати релативно већа кашњења. На тај начин, CMP дозвољава бржи сигнал такта у сваком језгру.

Поред тога, CMP боље користи простор на чипу јер нема потребе за додатном логиком коју намећу централизоване структуре у суперскалару. Показано је да CMP постиже ефективно већи пропусни опсег издавања инструкција него суперскалар са истом површином чипа. CMP са 8 двоструких (2-*issue*) суперскаларних језгара заузима исти простор као и конвенционални дванаестоструки (12-*issue*) суперскалар [HAMM1997].

Како су језгра код CMP много простија, то такође смањује трошкове јер скраћује потребно вријеме пројектовања и верификације које је код сложених суперскалара сразмјерно веће. Штавише, репликација једног, већ развијеног, процесорског језгра у CMP омогућава погодно скалирање ка већим системима.

С обзиром да се кашњења на чипу брже смањују него кашњења изван чипа, комуникација изван чипа постаје релативно скупља [AGAR2000]. Зато је једна од тежњи пројектаната да ограничи и смањи комуникацију изван чипа. CMP процесор је управо једно такво рјешење с обзиром да се више језгара заједно са дијелом меморијске хијерархије налазе у оквиру чипа [ZAHR2003].

Поред тога, CMP има бољи однос цијене и перформанси него конвенционални SMP мултипроцесори код којих су латенције приступа меморији много веће него у CMP или у суперскалару. Поред времена приступа самој меморији у SMP мултипроцесору, латенцију приступа изван чипа значајно повећава и вријеме кашњења по интерконекционој мрежи, акције протокола за кохеренцију кеш меморија, итд [CULL1999]. Интеграција CMP у скалабилне мултипроцесорске системе има велики потенцијал јер се уз мале измјене може користити и као градивни блок таквих система [STEF2000].

CMP процесори се стандардно користе за извршавање апликација у Web серверима и серверима база података као и за мултипрограмско радно оптерећење, јер се овакве апликације добро извршавају и на конвенционалним мултипроцесорима. У данашњим и будућим мултимедијалним апликацијама веома су заступљени графика, обрада слике и говора. Оне су веома захтјевне у погледу процесне снаге као и апликације за моделовање научних феномена и разне инжењерске апликације. CMP процесори су начелно погодни за извршавање оваквих апликција са крупнијом грануларношћу паралелизма.

Међутим, главни недостатак CMP у односу на сложене суперскаларе се огледа у споријем извршавању секвенцијалних апликација, када се при извршавању користи само једно језгро. Зато се, за шири опсег примјена, морају постићи боља искоришћеност ресурса и перформансе у извршавању секвенцијалног радног оптерећења. Ово се може остварити конвертовањем секвенцијалних програма у вишенитне (*multithreaded*) програме.

2.2 Паралелизам на нивоу нити

Вишенитност (*multithreading*) је техника која дозвољава да се секвенцијални програм разбије у мање инструкцијске токове – нити (*threads*), које се могу извршавати у паралели на различитим језгрима у оквиру CMP процесора. У наставку ће се за овај механизам користити термин TLP (*Thread Level Parallelism*) – паралелизам на нивоу нити. TLP омогућава паралелизам крупне грануларности, за разлику од ILP који представља паралелизам финије грануларности. Како су језгра у CMP простији суперскалари, одређени ниво ILP се и даље може користити, јер су TLP и ILP ортогонални.

Најбољи кандидати за нити су: базични блокови (секвенце инструкција без скокова унутра и напоље осим на почетку и крају), итерације унутрашњих и спољашњих петљи, позиви потпрограма, итд. Основни проблем код TLP лежи у организацији нити, тако да оне могу да се извршавају у паралели, и њиховој синхронизацији, водећи притом рачуна о зависности по подацима.

2.2.1 Зависности по подацима

Нека при секвенцијалном извршавању постоји инструкција која врши упис у неку промјенљиву, а касније у коду нека постоји инструкција која исту промјенљиву чита. Због семантичке коректности неопходно је да се те инструкције и у паралелизованом коду изврше у оригиналном програмском поретку. У супротном, јавља се прерано читање или RAW (*Read after Write*) хазард изазван **правом зависношћу по подацима** између ових инструкција. Детектовање и спречавање овог хазарда је главни проблем у имплементацији TLP-а.

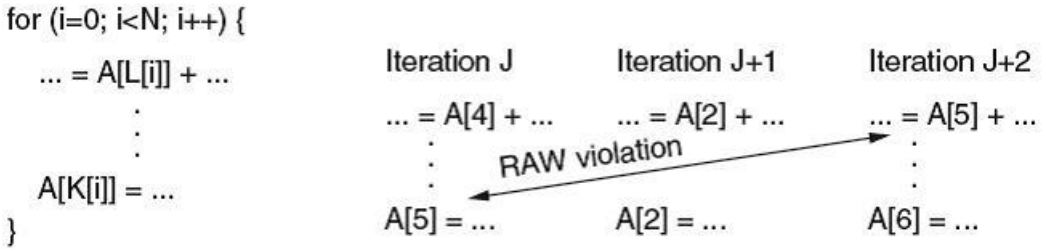
У обрнутом случају када се при секвенцијалном извршавању прво извршава инструкција која чита неку промјенљиву, а затим инструкција које уписује у исту промјенљиву, ове двије инструкције се, такође, морају извршити у истом програмском редосљеду. У супротном, јавља се прерани упис или WAR (*Write after Read*) хазард изазван **антизависношћу по подацима** између ових инструкција. Овај хазард се рјешава преименовањем промјенљиве у коју уписује друга инструкција па се може извршити прије читања јер је коректна вриједност за читање сачувана.

Ако двије операције у програмском поретку врше упис у исту промјењиву, постоји **излазна зависност по подацима** или WAW (*Write after Write*) хазард. Ако се између та два уписа не јавља читање ове промјењиве, први упис се може уклонити. Ако читање постоји, онда се овај случај своди на комбинацију претходне двије поменуте зависности по подацима.

2.2.2 Врсте TLP

Постоје двије врсте TLP: експлицитни и имплицитни. Основне разлике се тичу издавања и извршавања нити, као и комуникационог механизма, док су архитектура процесора и меморијска хијерархија слични. У *експлицитном* (неспекулативном) TLP нити могу бити независне или зависне, али синхронизоване на сваку појаву зависности по подацима тако да могу бити неспекулативно извршене на конкурентан начин у исправном поретку. Зависности између нити се контролишу одговарајућим софтверским примитивама (*fork primitives*).

Експлицитне нити у секвенцијалној апликацији могу бити идентификоване помоћу напредних паралелизујућих преводилаца или мануелном паралелизацијом [TORR2011, OLUK2007, OOI2001]. Међутим, идентификација екплицитних нити није лак посао због проблема са показивачима, с контролним стуктурама селекције и гранања, итд. Чак и у нумеричким апликацијама преводиоци су само у ограниченом обиму ефикасни у препознавању експлицитних нити. Паралелизујући преводиоци примјењују конзервативну стратегију при идентификацији нити јер претпостављају да постоје зависности између нити увијек када не могу да гарантују њихову независност, чак и ако је та зависност мало вјероватна. Као што се види са слике 2-1, ако су вриједности елемената низова L и K зависне од улазних података, преводилац не може одредити да ли различите итерације петље приступају различитим елементима низа, па означавају петљу за секвенцијално извршавање.



Слика 2-1. Примјер могуће зависности између итерација петље [TORR2011]

Према томе, ни напредни преводиоци не могу осигурати да CMP процесори могу ефикасно извршавати секвенцијалне апликације опште намјене као

експлицитне нити. Проблем би се могао ријешити *имплицитним* (спекулативним) нитима које би се могле идентификовати у вријеме превођења програма или при његовом извршавању уз посебну хардверску подршку. Ове нити би могле имати потенцијалне међусобне зависности и извршавале би се спекулативно у CMP процесору уз постојање софтверске или хардверске подршке која би динамички детектовала и разријешавала зависности између нити [TORR2011, OLUK2007, KRIS1999].

У оваквом систему нити се спекулативно извршавају у паралели на различитим језгрима све док зависности нијесу нарушене. Ако се зависност не појави до краја, нити се завршавају и ефекти њиховог рада су важећи. Ако је спекулација неуспјешна, нит која је нарушила зависност и све наредне спекулативне нити прекидају извршавање, њихови ефекти се поништавају и оне се касније извршавају када се добију исправни подаци. Да би се гарантовали исти резултати спекулативног извршавања као код неспекулативног, потребна је додатна хардверска логика за идентификацију нити, предвиђање и детектовање зависности, предвиђање вриједности података, итд. Ова техника се назива спекулација на нивоу нити - TLS (*Thread-Level Speculation*). Пошто се TLS и синхронизација нити не искључују, да би се поправиле перформансе експлицитна синхронизација се може користити када су зависности између нити вјероватне.

2.3 Спекулативна паралелизација на нивоу нити

У [KNIG1986] је предложена прва архитектура која подржава извршавање спекулативних нити за функционалне језике где је коректно извршење паралелног кода са бочним ефектима подржано хардверски. Након тога је TLS техника коришћена код различитих CMP система (нпр., Multiscalar је био један од првих мултипроцесора потпуно оријентисан ка спекулативном извршавању [OLUK2007]). Преглед ових система је дат у секцији 2.4.

2.3.1 Примјенљивост TLS-а

Спекулативни паралелизам се може наћи у многим секвенцијалним апликацијама (нпр., [PACK2009]). TLS омогућава паралелизацију секвенцијалних програма без потребе за сложеном анализом зависности и експлицитном синхронизацијом, као и искоришћење пуног потенцијала CMP процесора у извршавању секвенцијалних апликација опште намјене. Ефекти нити постају важећи у истом поретку као када би се оне извршавале секвенцијално иако се стварно извршавају у паралели.

TLS се најчешће користе у паралелизацији петљи и позива потпрограма. Највећи потенцијал за паралелизацију се налази у петљама јер се свака итерација петље извршава као засебна нит. То се постиже тако што се убацује *spawn*

инструкција, којом се процесору додељује нит на извршавање. Свака нит извршава исти код. Нешто је теже паралелизовати позиве потпрограма. У том случају се обично сам потпрограм одвија у неспекулативној нити, док се наставак кода иза позива одвија у спекулативној нити уз претпостављене резултате рада потпрограма. Ако дође до погрешне спекулације, резултати рада спекулативних нити се поништавају.

2.3.2 TLS механизам

У систему са TLS подршком од више нити које се извршавају паралелно, најраније стартована нит је неспекулативна, док су остале нити спекулативне. Само неспекулативна нит може да мијења перманентно стање у меморији, док спекулативне нити своја стања извршавања држе у локалним баферима. Када се неспекулативна нит заврши, прва наредна спекулативна нит постаје неспекулативна, па се њено стање потврђује и пребацује у меморију (*commit*). Такође, спекулативне нити могу да читају и податке из спекулативних стања који су уписани у бафере претходних (мање спекулативних) нити, али не и оне из бафера наредних (спекулативнијих) нити.

Хардверска подршка за спекулацију у виду идеалног система би укључивала потпуно асоцијативне кеш меморије првог нивоа неограниченог капацитета које су приватне за свако процесорско језгро. Резултати би се чували у овим кеш меморијама током спекулативног извршавања у режиму одложеног уписа (*write-back*), а промјена секвенцијалног стања у меморији би се направила само када се ове нити успјешно заврше. У спекулативном режиму рада, када нит *i* чита неки податак спекулативни хардвер мора да врати најскорију вриједност. Ако се она не нађе у приватној кеш меморији првог нивоа, онда се узима из кеш меморије језгра које извршава нит *j* мање спекулативну од *i*. Ако ниједна мање спекулативна нит није произвела ову вриједност, она се чита из кеш меморије другог нивоа или из меморије [OPLI1997].

Адекватна хардверска подршка за TLS поставља пет захтјева:

1. Прослеђивање података између паралелних нити,
2. Детектовање RAW хазарда,
3. Поништавање резултата спекулативног извршавања када је спекулација погрешна,
4. Извршавање спекулативних уписа у коректном поретку (WAW хазарди)
5. Рјешавање WAR хазарда преименовањем меморијских промјењивих.

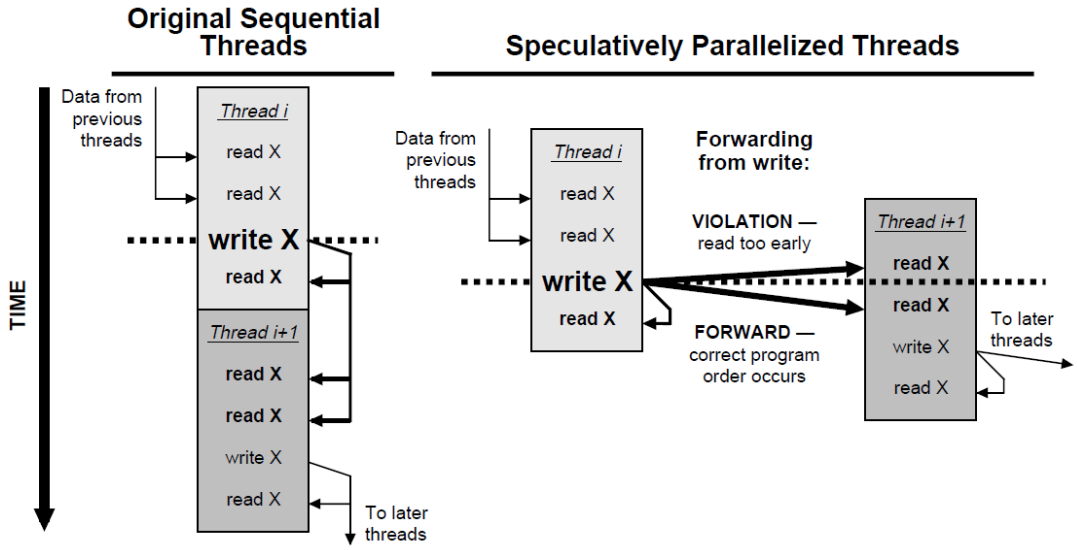
У случају правог дијељења података између нити, када каснијој нити затреба податак, ранија нит треба да јој прослиједи најскорију вриједност. Понекад, да би се скратило чекање, нит која производи податак може да га пошаље наредним нитима унапријед, без захтјева.

RAW хазард се дешава када је нит прерано прочитала податак. Зато ситуација у којој спекулативнија нит прво чита податак који касније нека ранија, мање спекулативна нит уписује мора бити динамички детектована. Ово се обично разрјешава поништавањем (*squashing*) нити која је прочитала некоректан податак, а затим њеним накнадним извршавањем са исправним податком.

У случају нарушавања зависности перманентно стање не смије бити промијењено и сви ефекти те нити морају бити поништени. Обично се спекулативно стање држи само у приватној кеш меморији првог нивоа, а перманентно стање у кеш меморији другог нивоа, тако да се ефекти погрешне спекулације лако поништавају.

Када каснија нит (спекулативнија нит) уписује у исти податак прије него што ранија нит (мање спекулативна нит) упише у њега (WAW хазард), остале нити морају видјети ефекте ових уписа у коректном поретку. Ово се постиже тако што нити ажурирају перманентно стање строго поштујући програмски поредак.

Коначно, ранија нит не смије никада читати податак који је произвела каснија нит (WAR хазард). Ово се осигурава држањем актуелних података у приватном кешу првог нивоа којем не могу приступати раније нити. Илустрација неких описаних ситуација је дата на слици 2-2.



Слика 2-2. Примјер спекулативних нити са зависношћу по подацима [OLUK2007]

CMP систем са TLS подршком је квалитетна и ефективна алтернатива за сложене суперскаларне процесора. Показано је да спекулативни CMP исте површине чипа може постићи на цјелобројним нумеричким апликацијама перформансе као и суперскаларни процесор [OLUK2007]. Међутим, поред предности постоје и потенцијални проблеми код примјене ове технике као што су: а) недостатак инхерентног паралелизма у апликацијама; б) додатна хардверска подршка који се не користи када се извршава чисто паралелна апликација или

мултипрограмско радно оптерећење; в) софтверска подршка за обраду спекулативних нити; г) повећане латенције комуникације међу нитима; д) изгубљено вријеме код погрешне спекулације.

2.4 Преглед спекулативних CMP система

Постоје три главна начина у пројектовању спекулативних CMP-а. Први начин се односи на CMP архитектуре које су у потпуности оријентисане ка искоришћењу спекулативног паралелизма, као што су Multiscalar [SOHI1995], Multiplex [OOI2001], Trace [ROTE1997], SM [MARC1998], MAJC [TREM2000], MP98 (Merlot) [EDAH2000] и Mitosis [MADR2008]. Они се одликују сложеном хардверском подршком за TLS и омогућавају спекулативну комуникацију и на нивоу регистара и на нивоу меморије. Други начин се односи на генеричке CMP архитектуре са минималном додатном подршком за спекулативно извршавање, као што су Hydra [OLUK2007] и STAMPede [STEF2005]. Они садрже само подршку за спекулацију на нивоу меморије. Трећи приступ, који обухвата системе I-ACOMA [KRIS1999], Atlas [SASS2005], NECO [BARL2003] и Pinot [OHSA2005], покушава да комбинује предности прва два приступа. Иако и они омогућавају регистарску и меморијску спекулацију, ови системи су ближи генеричким архитектурама са одговарајућом хардверском подршком за TLS [TOMA2005].

У овој глави ће бити описане опште особине поменутих система, док ће механизми регистарске и меморијске комуникације бити подробно анализирани у наредним главама. На крају ове главе ће бити разматрани и упоређени начини идентификације нити, као и домен спекулације код разматраних система.

2.4.1 Системи потпуно оријентисани на TLS

Ови системи обично имају значајну хардверску и софтверску подршку за ефикасно спекулативно извршавање. Стога, они подржавају комуникацију између нити и на регистарском и меморијском нивоу што им омогућава да постигну високе перформансе при извршавању секвенцијалних апликација без потребе за поновним превођењем изворног кода програма. Међутим, када ови системи извршавају паралелне апликације или више независних апликација истовремено, велика количина подршке за спекулативно извршавање остаје неискоришћена.

Multiscalar. Multiscalar је једно од првих CMP рјешења које је у потпуности оријентисано ка спекулативном извршавању тако што користи агресивну спекулацију на нивоу контроле тока, као и на самим подацима уз подршку намјенског хардвера и сложених механизама просљеђивања. Постоје двије различите варијанте дизајна спекулативног меморијског система које су примјењене у Multiscalar-у: а) ARB (*Address Resolution Buffer*) као централизована кеш меморија првог нивоа (L1) обезбјеђује чување спекулативних података и

детекцију нарушавања меморијских зависности [SOHI1995], и б) SVC (*Speculative Versioning Cache*) који користи дистрибуиране L1 кеш меморије умјесто централизованог решења да елиминише проблеме кашњења и величине пропусног опсега који су уочени код ARB-а [GOPA1998].

У обје варијанте брзи једносмјерни прстен повезује процесорска језгра. Веза између процесорских језгара и ARB-а у првој варијанти дизајна је кросбар, док су ARB и главне меморије везани преко магистрале [SOHI1995]. У SVC L1 кеш меморије и главна меморије су на заједничкој магистрали [GOPA1998].

Multiplex. Multiplex CMP [OOI2001] је директни насљедник Multiscalar-а. Осим тога, Multiplex је први спекулативни CMP који обухвата експлицитно и имплицитно вишенитно извршавање груписањем подскупа стања протокола у имплицитном спекулативном CMP-у да би се подржао инвалидациони протокол за експлицитне нити без додатне хардверске подршке. Multiplex се састоји од четири конвенционална процесорска језгра са приватним L1 кеш меморијама за податке и инструкције и заједничком L2 кеш меморијом. Процесорска језгра су, као код Multiscalar-а, повезана у брзи једносмјерни прстен док су дистрибуране L1 кеш меморије преко заједничке магистрале повезане са дијељеном L2 кеш меморијом.

SM. SM процесор истовремено извршава нити на које је издијељен секвенцијални програм коришћењем техника спекулације које не захтијевају било какву подршку преводиоца или програмера [MARC1998, MARC1999]. Поред тога, нијесу потребне никакве промјене у скупу инструкција што значи да стандардни програми који су преведени за извршавање на суперскаларним процесорима такође могу бити извршавани и на SM-у.

SM процесор се састоји од неколико јединица за извршавање нити (*Thread Units* - TU) које су међусобно повезане једносмјерним прстеном. Све TU дијеле исту инструкцијску кеш меморију, као и MVC (*multi-value*) кеш меморију која представља посебан вид L1 кеш меморије који обезбјеђује потребну подршку за чување спекулативних верзија. TU су међусобно повезане са MVC преко кросбара.

Trace. Основна јединица контроле тока извршавања програма и контроле предикције у Trace CMP је траг (*trace*) – динамички низ инструкција који је хардверски генерисан током извршавања, а може обухватати било који број остварених или неостварених инструкција скока. Сличност између трага и нити је препозната у [VAJA1997, SUND1997] гдје се трагови разматрају као динамичке верзије статичких Multiscalar нити иако оне могу бити произвољно велике. Овако се умањује комплексност свих фаза спекулативног извршавања увођењем хијерархије у контроли тока извршавања програма на нивоу трага и унутар њега.

Trace CMP се састоји од четири 4-*issue* суперскаларна процесорска језгра, кеш меморије за чување трагова и диспечера који представља хардверску подршку за генерисање динамичког трага [ROTE1997].

Mitosis. Mitosis је спекулативни CMP који врши предикцију зависности по подацима између нити, као и управљање њима уз софтверску подршку. Предикција и израчунавање улазних вриједности за дату нит се обавља помоћу програмског кода који се додаје на почетку сваке нити – *p-slice* (*precomputation slice*) [MADR2008].

Mitosis се састоји од јединица за обраду нити које су сличне са конвенционалним суперскаларним процесорима, приватних L1 кеш меморија који су са глобалном L2 кеш меморијом повезаним преко дијељене магистрале. Посебна компонента (*Speculation Engine*) управља задацима који се односе на извршавање спекулативне нити: одређивање слободне јединице за обраду нити, иницијализација неких регистара и одржавање поретка између активних нити.

MP98 (Merlot). MP98, који је такође познат и под именом свог првог прототипа Merlot, је комерцијални CMP који подржава спекулацију на нивоу контроле тока програма и на подацима. Његов дизајн са малом потрошњом енергије, као и софтвер за аутоматску паралелизацију, обезбјеђују високе перформансе и омогућава употребу у паметним информационим терминалима. MP98 (Merlot) се састоји од четири процесорска језгра и централизоване кеш меморије за податке која је дијељена између свих процесорских јединица. Поред тога, свако процесорско језгро има свој бафер (SRB) за чување података током спекулативног извршавања. Процесорска језгра су са дијељеним скупом регистара и SRB баферима повезана мрежом директне повезаности (*point-to-point*) [EDAH2000, MATS2000].

MAJC. MAJC (*Multiprocessor Architecture for Java Computing*) CMP је посебно погодан за мултимедијалне апликације јер је пројектован као скалабилно рјешење са добрим перформансама и екстремно великим пропусним опсегом за кохеренцију између процесорских језгара на самом чипу [TREM2000, SUDH2000]. MAJC-5200 је прва имплементација MAJC архитектуре која је намијењена мрежним и комуникационим уређајима, клијентским платформама и апликационим серверима који испоручују дигитални садржај и Java апликације.

MAJC има три нивоа хијерахије: процесорски кластер, процесорску јединицу и функционалну јединицу. Процесорски кластер садржи више процесорских јединица (CPUs) на једном чипу (двије код MAJC-5200). Свака CPU садржи приватну кеш меморију за инструкције док је кохерентна кеш меморија за податке заједничка за све процесорске јединице на чипу. Поред тога, CPU се састоји од четири функционалне јединице, при чему свака од њих има локалне управљачке структуре. Оваква хијерахијска организација MAJC CMP-а омогућава да се изврши њено прилагођење потребама појединачних апликација [SUDH2000].

2.4.2 Генерички системи са TLS подршком

У појединим студијама о утицају комуникационог кашњења на укупне перформансе спекулативног CMP-а сматра се да није потребно увођење брзе

комуникационе шеме између процесорских јединица на чипу зато што је комуникација између нити на меморијском нивоу довољно брза да умањи утицај комуникационог кашњења на перформансе система [PACK2009, OANC2008, ZIAR2009, WARG2006]. Такав систем остварује комуникацију између нити искључиво преко дијељене меморије. Ограничена хардверска подршка у оваквом CMP-у је довољна за исправно спекулативно извршавање пошто се они ослањају на подршку преводиоца и софтверске компоненте за управљање спекулацијом да би прилагодили секвенцијалне програме за спекулативно извршавање. Међутим, потреба за поновним превођењем изворног програмског кода може бити један од озбиљних проблема посебно када изворни код програма није доступан.

Hydra. Hydra CMP комбинује мултипроцесорску архитектуру са дијељеном меморијом, специфичне механизме синхронизације, технологију интегралних кола и технологију паралелних преводилаца да би остварио бољи однос цијена/перформансе и ефикаснији паралелни програмски модел [OLUK2007]. Hydra се састоји од четири процесора заснована на MIPS архитектури (слични су R10000 процесору). Веза између процесора је заснована на дијељеној магистрали док је меморијски систем имплементиран у четири нивоа: приватне L1 кеш меморије и дијељена L2 кеш меморија на самом чипу, L3 кеш меморија и главна меморија ван чипа. Такође, сваки процесор има свој меморијски контролер који управља приступима ка нижим меморијским нивоима, од L2 кеш меморије до главне меморије. Хардверско/софтверски интерфејс који је додат да контролише поредак извршавања нити у Hydra-и је имплементиран коришћењем MIPS копроцесорског CP2 интерфејса и софтверских компоненти [HAMM2000, OLUK2007].

STAMPede. STAMPede CMP са подршком за TLS је веома сличан систему Hydra, али са једноставнијим хардвером [STEF2005]. Пројектанти су жељели да подрже један облик агресивне TLS уводећи само минималне модификације у генерички CMP без умањивања перформанси апликација које не користе TLS како би се избјегле сложене централизоване архитектуре које могу утицати на повећање кашњења код L1 кеш меморија за податке. Општи STAMPede систем може имати већи број нивоа кеш меморије. Међутим, поједностављена верзија STAMPede има приватне кеш меморије за податке и инструкције за сваку процесорску јединицу и дијељену L2 кеш меморију. Дистрибуиране L1 кеш меморије и дијељена L2 кеш меморија су повезане кросбаром.

STAMpede са подршком за TLS је погодан за било коју платформу са паралелним нитима као што су: SMT (*Simultaneous Multithreaded processor*), CMP, вишеструки процесори са дијељеном меморијом или масовни паралелни системи за чији дизајн су као градивни блокови коришћени CMP или SMT процесори. Програм који је једном преведен да искористи TLS може директно бити извршен на било којој од поменутих вишенитних платформи без потребе за поновним превођењем изворног кода.

2.4.3 Хибридни приступ

Системи ове групе покушавају да комбинују најбоље карактеристике претходна два приступа у пројектовању CMP са подршком за TLS:

- основна генеричка CMP архитектура као у другој групи,
- рад са секвенцијалним бинарним кодом без потребе за рекомпилацијом изворног кода као код прве групе,
- подршка за комуникацију између нити и на регистарском и на меморијском нивоу као код прве групе,
- скромнија хардверска подршка за TLS као код друге групе.

Према томе, ови системи имају довољну подршку за извршавање спекулативно паралелизованих апликација, али ефикасније користе ресурсе при извршавању правих паралелних апликација или више независних апликација истовремено него системи из прве групе.

Atlas. Дизајн Atlas процесора је најсличнији Multiscalar-у, али са двије битне разлике: 1) Atlas користи нови алгоритам за подјелу тока инструкција на меморијским референцама умјесто на инструкцијама за гранање тока програма, и 2) Atlas користи агресивну комбиновану предикцију вриједности и контроле тока.

Atlas процесор се састоји од 8 процесорских језгара базираних на Alpha 21164 процесору, који су међусобно повезани двосмјерним прстеном. Такође, свако процесорско језгро је повезано на двије заједничке магистрале, једну за везу према заједничкој L2 кеш меморији и другу према комбинованом предиктору вриједности и контроле тока. Овакав дизајн Atlas процесора је показао да се спекулација нити може комбиновати са предикцијом вриједности и механизмом динамичке подјеле секвенцијалног програма да би се подржало паралелно извршавање међусобно зависних нити [CODR2001, SASS2005].

NEKO. Карактеристике NEKO дизајна су имплементација не само метода за постизање веће тачности спекулације већ и метода чији је задатак да смање утицај погрешних спекулација као и негативне ефекте спекулативног извршавања (нпр. деградација фактора поготка за кеш меморије инструкција и података, велики број инвалидационих промашаја у кешевима података) [HUNG2002, BARL2004].

NEKO се састоји од јединице за контролу нити, четири процесорска језгра, приватних L1 кеш меморија за инструкције и податке и дијељене L2 кеш меморије. Процесорска језгра су повезана једносмјерним прстеном, а приватне L1 кеш меморије су преко дијељене магистрале повезани на заједничку L2 кеш меморију. Централизована јединица контроле нити обрађује контролне зависности између нити и врши распоређивање нити на слободна процесорска језгра. Као подршку спекулативном извршавању NEKO има хардверске јединице за предикцију нити и за валидацију тачности предикције [HUNG2002, BARL2004].

Pinot. Pinot CMP, као и NEKO, такође подржава спекулацију на нивоу података и на нивоу контроле тока извршавања програма. Pinot се састоји од четири процесорске јединице који имају приватне L1 кеш меморије које су преко дијељене магистрале повезане на остатак меморијског система, док су саме процесорске јединице повезане једносмјерним прстеном.

Подршка за спекулацију унутар сваке процесорске јединице обухвата декодер инструкција за паралелизацију (*fork, propagate* и *cancel* инструкције су додате постојећој ISA), контролу генерисања нити следбеника и кеш меморију података у којој се чувају спекулативне вриједности [OHSA2005].

I-ACOMA. I-ACOMA је генеричка CMP архитектура са додатном хардверском подршком за спекулативно извршавање [KRIS1998, KRIS1999]. Састоји се од четири 4-*issue* динамичка суперскаларна процесора (слични са R10K MIPS процесором), приватних L1 кеш меморија и дијељене L2 кеш меморије на чипу. Процесорска језгра су повезана преко магистрале, док су приватне L1 кеш меморије повезани са дијељеном L2 кеш меморијом преко кросбара.

Хардверска подршка за идентификацију меморијских зависности у виду табеле MDT (*Memory Disambiguation Table*), која је аналогна са структуром каталога (*directory*) у конвенционалном мултипроцесору са дијељеном меморијом, зависно од своје величине може бити саставни дио L2 кеш меморије или смјештена посебно на самом чипу [KRIS1998, KRIS1999].

2.5 Идентификација нити и домен спекулације

Прије извршавања секвенцијалних апликација у спекулативном CMP процесору потребно је претходно извршити идентификацију спекулативних нити или прије почетка извршавања (нпр., помоћу преводиоца, коришћењем одређених анотационих алата или алата за паралелизацију секвенцијалних апликација, мануелном паралелизацијом, итд.) или у потпуности током извршавања уз хардверску подршку [TORR2011, OLUK2007, OOI2001]. Домен спекулације (грануларност нити) обухвата широк спектар од основних блокова до читавих потпрограма. У табели 2-1 је дат преглед разматраних спекулативних система у погледу њихове подршке за идентификацију нити и домена спекулације.

2.5.1 Идентификација нити

Већина разматраних спекулативних система користи различите технике превођења за идентификацију спекулативних нити у секвенцијалним апликацијама. Преводиоци код Multiscalar-а и Multiplex-а користе хеуристике које се односе на контролу тока извршавања програма, зависности по подацима и величину нити да би издвојили више паралелизма из цјелобројних апликација као и апликација са покретним зарезом [SOHI1995, OOI2001]. Међутим, поред тога

што оба CMP система подржавају имплицитне спекулативне нити, Multiplex такође подржава и експлицитне спекулативне нити. Multiplex обједињује Polaris преводилац [BLUM1996] са Multiscalar преводиоцем како би обезбиједио експлицитни и имплицитни модел издвајања спекулативних нити из секвенцијалних апликација током додјеле нити процесорским јединицама тако што укључује бит модела у сваком дескриптору нити.

У MP98 (Merlot) преводилац у комбинацији са FOPE (*Fork-once parallel execution*) хардверском подршком врши идентификацију нити из секвенцијалних апликација омогућавајући програмерима да уметну одређене директиве у изворни код да би издвојили експлицитни паралелизам и тиме остварили максималне перформансе [EDAH2000, MATS2000]. MAJC се ослања на подршку преводиоца за додјелу инструкција функционалним јединицама [TREM2000]. NEKO такође користи преводилац да би извршио статичку идентификацију спекулативних нити из секвенцијалне апликације [BARL2003], док Mitosis преводилац статички дијели секвенцијалне програме у спекулативне нити уметањем *spawning* парова у програмски код и генерисањем одговарајућих *p-slice* кодова за сваки пар [MADR2008]. Hydra користи SUIF [AMAR1995] и Hydracat [HAMM2000, OLUK2007] преводиоце како би извршила идентификацију спекулативних нити у секвенцијалним апликацијама, док STAMPede користи инфраструктуру преводиоца која се такође базира на SUIF преводиоцу [STEF2005].

Табела 2-1. Подршка за идентификацију нити и област спекулације

CMP	**Подршка за идентификацију нити**	**Домен спекулације (грануларност нити)**
Multiscalar	Преводилац	Базични блок инструкција, вишеструки базични блокови инструкција, итерација петље, комплетне петље, потпрограми.
Multiplex	Преводилац	Базични блок инструкција, вишеструки базични блокови инструкција, итерација петље, комплетне петље, потпрограми.
SM	Хардвер	Итерација петље
MP98	Преводилац/хардвер	Итерација петље и базични блокови инструкција
MAJC	Преводилац	Комплетне петље и потпрограми
Trace	Хардвер	Трагови (низови динамичких инструкција)
I-ACOMA	Софтверски анотатор	Итерација петље
Atlas	Хардвер	Низови инструкција ограничени меморијским инструкцијама за упис и читање
NEKO	Преводилац	Базични блок инструкција, вишеструки базични блокови инструкција, итерација петље, комплетне петље, потпрограми.
Pinot	Алат за паралелизацију	Итерација петље, потпрограми и базични блокови инструкција
Mitosis	Преводилац	Било који пар базичних блокова инструкција
Hydra	Преводилац	Итерација петље и потпрограми
STAMPede	Преводилац	Итерација петље и потпрограми

I-ACOMA и Pinot користе намјенске софтверске алате за идентификацију спекулативних нити у секвенцијалним апликацијама. I-ACOMA користи анотатор бинарног кода за идентификацију нити као и за идентификацију зависности регистарских вриједности између нити [KRIS1999]. Pinot се ослања на алат за паралелизацију који користи профајлерске информације да би издвојио из секвенцијалних апликација паралелизам на нивоу нити са било којим нивоом грануларности и да би затим извршио конверзију секвенцијалних апликација у спекулативне вишенитне апликације [OHSA2005].

На крају, SM [MARC1998], Trace [ROTE1997] и Atlas [SASS2005] се ослањају у потпуности на хардверску подршку да би динамички издвојили нити из секвенцијалних апликација. Хардверски механизам за идентификацију нити има неколико предности у односу на софтверске технике: а) анализа контроле тока извршавања програма од стране преводиоца је непрецизнија; б) резултати техника за профилисање зависности по подацима које користи преводилац да би побољшао анализу контроле тока извршавања програма се ослањају на конкретан скуп улазних података; в) коришћење механизама за профилисање зависности по подацима има временску и меморијску цијену када се врши профилисање великих апликација, као и апликација чије је вријеме извршавања дуго; г) неки софтверски приступи захтијевају измјене у ISA што не обезбјеђује компатибилност са претходним ISA имплементацијама.

2.5.2 Домен спекулације

Pinot је јединствен у издвајању паралелизма из било које програмске подструктуре у широком распону гранулације нити. Алат за паралелизацију који се користи код Pinot-а може формирати нит од 10K инструкција која укључује читаве петље или потпрограме као и паралелизам фине грануларности [OHSA2005]. С обзиром да Multiscalar, NEKO и Multiplex преводиоци завршавају формирање нити када наиђу на почетак петље или позив потпрограма тако да, за разлику од Pinot-а, не могу у потпуности да издвоје паралелизам крупне грануларности без обзира на величину програмских структура на којима се примјењује спекулација (један или више базичних блокова инструкција, комплетне петље, итерације петљи или позиви потпрограма). MP98, MAJC, Mitosis, Hydra, STAMPede и I-ACOMA спекулишу у мањим областима као што су само итерације петље, само базични блокови инструкција, или и итерације петље и базични блокови инструкција односно петље и потпрограми.

Динамички алгоритми партиционирања који се користе у SM и Trace ограничавају домен спекулације на итерације петљи и границе кеш линије. Међутим, MEM *slicing* алгоритам који се примјењује у Atlas-у представља побољшање у односу на алгоритме партиционирања на фиксном интервалу који се примјењују код SM и Trace јер спекулативне нити издваја на меморијским операцијама за читање или упис што нити чини већима и предвидљивијима.

Глава 3 Регистарска комуникација код спекулативних CMP процесора

Преглед система из претходне главе показује да је подршка за TLS на нивоу регистара веома честа у спекулативним CMP процесорима, као комплементарни механизам уз меморијску комуникацију. Зато се у овој глави предлаже једна класификација и даје подробна упоредна анализа овог механизма из више релевантних аспеката [RADU2014].

3.1 Класификација спекулативних CMP процесора са регистарском комуникацијом

Организација скупа регистара има врло значајан утицај на имплементацију механизама синхронизације и комуникације регистарских вриједности између процесорских језгара у спекулативном CMP. Због тога је она усвојена као главни критеријум класификације. Препозната су три различита приступа у организацији скупа регистара:

- дистрибуирани (локални) скупови регистара (LRF – *Local Register File*),
- дијељени (глобални) скуп регистара (GRF – *Global Register File*),
- хибридни приступ који комбинује локалне скупове регистара са глобалним скупом регистара.

Традиционални микропроцесори углавном имају GRF који међусобно дијели више функционалних јединица што утиче на повећање броја портова за упис и читање и води ка сложенијем ожичењу, промјенама времена циклуса и повећању кашњења. Подршка вишим степенима ILP и TLP паралелизма условљава веома

брз раст сложености хардверске имплементације дизајна са централизованим скупом регистара и великим и комплексним мрежама за размјену регистарских вриједности. У екстремно физички малим дизајнима паралелних процесора, као што је CMP, дијељени (глобални) скуп регистара мора бити замијењен са дистрибураним скуповима регистара са локалним комуникационим путевима како би се ублажили проблеми које изазива велики број дугих интерконекција између скупа регистара и мреже за размјену операнада. Ово је разлог због којег је ексклузивно коришћење дијељеног скупа регистара у CMP дизајну ријетко. Дистрибуиране организације скупа регистара су веома скалабилне у поређењу са традиционалном дијељеном организацијом скупа регистара зато што значајно смањују величину чипа, кашњење и дисипацију. Због тога велика већина спекулативних CMP-а са подршком за регистарску комуникацију користи дистрибуирани приступ организације скупа регистара или га комбинује са малим дијељеним скупом регистара [PATT2007, BELL2006, TANA2009, ALIP2012].

Поред организације скупа регистара, која је главни критеријум класификације, као други класификациони критеријум који значајно утиче на дизајн је узета топологија мреже за размјену вриједности регистара између процесорских језгара на чипу. Прстен код Multiscalar, Multiplex, SM, Atlas, NEKO и Pinot спекулативних CMP процесора представља природан избор топологије мреже зато што током спекулативног извршавања језгра на чипу примарно комуницирају са своја два најближа сусједа – генерисане регистарске вриједности се шаљу ка језгру које извршава наредна спекулативна нит док тражене регистарске вриједности долазе од језгра на којем се извршава претходна спекулативна нит.

Магистрала је сљедећи избор у топологији мреже која је имплементирана код Trace, IACOMA и Mitosis спекулативних процесора. Магистрала би требало да буде довољна да опслужи мали број језгара на чипу (нпр., од 4 до 8 језгара). Међутим, већи број језгара или коришћење бржих језгара неминовно тражи и већи пропусни опсег на магистрали што заузврат захтијева или увођење већег броја магистрала или увођење хијерархије локалних и глобалних магистрала.

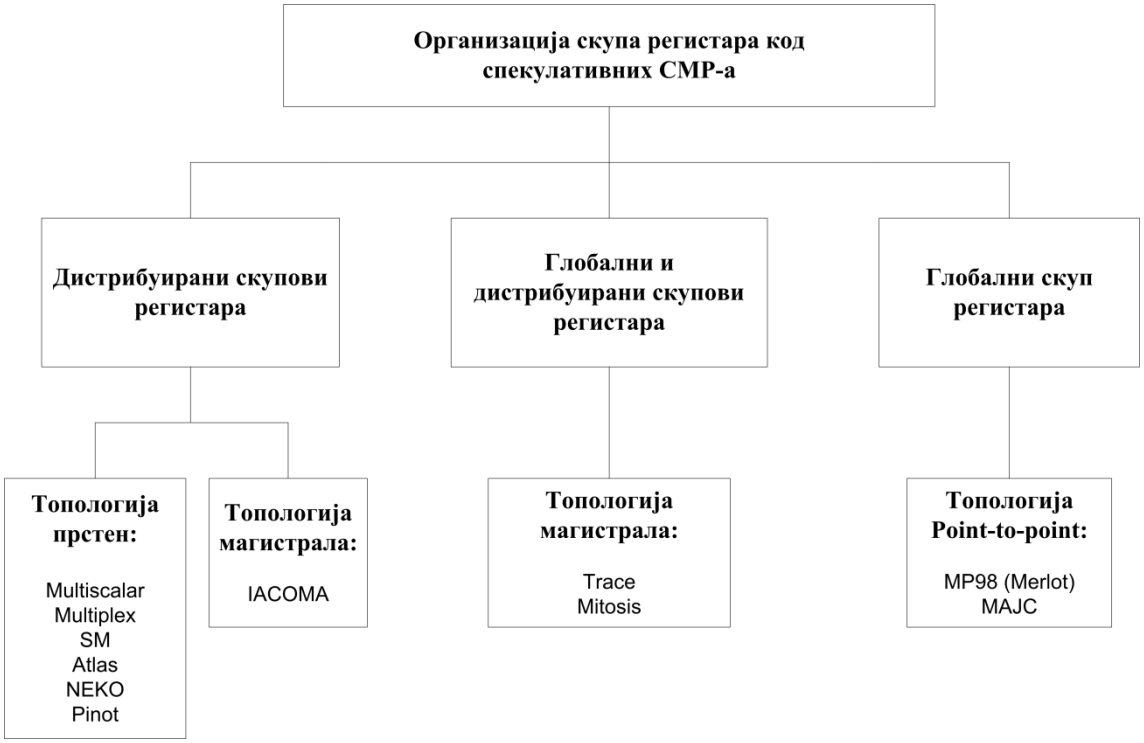
Код спекулативних CMP процесора са GRF као што су MP98 (Merlot) и MAJC, мреже директне повезаности (*point-to-point*) се користе за комуникацију регистарских вриједности између језгара на чипу.

На слици 3-1 је приказана класификација спекулативних CMP-а са подршком за регистарску комуникацију између спекулативних нити на основу претходна два критеријума.

3.2 Упоредна анализа техника регистарске комуникације

Упоредну квантитативну анализу техника регистарске комуникације реализованих у претходно наведеним спекулативним CMP процесорима је веома тешко извести због веома различитих пројектних рјешења. Зато је овдје спроведено квалитативно поређење предлога за подршку регистарске

комуникације са становишта синхронизације и комуникације регистарских вриједности, са становишта технике опоравка након грешке у спекулацији, као и са становишта показаних перформанси и скалабилности система.



Слика 3-1. Класификација спекулативних CMP процесора са подршком за регистарску комуникацију између спекулативних нити

3.2.1 Регистарски комуникациони механизми

Већина спекулативних CMP са подршком за регистарску комуникацију спекулишу на регистарским вриједностима на два начина: коришћењем одређених механизама размјене спекулативних регистарских вриједности (нпр., Multiscalar, Multiplex, NEKO и Pinot) или коришћењем регистарских вриједности које су добијене предикцијом (нпр., SM, Atlas, Trace и Mitosis). С друге стране, системи као што су I-ACOMA и MP98 (Merlot) користе одређене механизме синхронизације да би управљали комуникацијом између спекулативних нити на регистарском нивоу.

Размјена спекулативних регистарских вриједности. Овај начин укључује механизме гдје инструкција коју извршава каснија, више спекулативна нит, а која зависи од резултата извршавања инструкције код раније, мање спекулативне нити, мора да сачека регистарску вриједност коју ће старија спекулативна нит произвести. Ови механизми размјене спекулативних регистарских вриједности уносе дужа кашњења у комуникацији између процесорских језгара спекулативног

CMP-а и узрокују заустављање извршавања због недостатка потребних података. Међутим, коришћењем различитих техника, као што су: а) директна комуникација између регистара у чврсто спрегнутим архитектурама; б) прослеђивање регистарских вриједности, било да су вриједности коначне или не, од нити која их је произвела ближе нити која их тражи; или в) идентификација посљедње нити која је произвела регистарску вриједност, могу се смањити велика кашњења и избјећи заустављање спекулативног извршавања.

Multiscalar и Multiplex користе једносмјерни прстен као интерконекцију између процесора за директну комуникацију између локалних скупова регистара како би смањио кашњења у комуникацији, док је размјена регистарских вриједности контролисана помоћу једноставних логичких операција на каталогу од шест скупова маски битова које се дефинишу за сваки регистар. Проблем посљедње копије је ријешен тако што унутар сваког процесора постоје двије копије скупа регистара. Копије регистарских вриједности које су прослеђене од стране мање спекулативне ка наредној, више спекулативној нити се чувају у *past* скупу регистара док се ново произведене регистарске вриједности чувају у *present* скупу регистара [SOHI1995, OOI2001].

Регистарски синхронизациони механизам који је имплементиран у NEKO CMP-у је сличан Multiscalar-овом, али се разликује у неколико битних детаља. Наиме, NEKO-ов механизам се умногоме ослања на помоћ преводиоца у циљу смањења хардверске подршке за регистарску синхронизацију тако што се у бинарни код умећу информације о регистрима који се могу редефинисати у нити, као и експлицитне инструкције за слање регистарских вриједности. Такође, регистарске вриједности се премјештају ближе нити која их тражи чак иако те регистарске вриједности нијесу коначне, чиме се смањује кашњење у комуникацији између нити. Хардверска подршка у сваком процесору у виду IRB (*Intermediate Register Buffer*) и CRB (*Communication Register Buffer*) обезбјеђује чување регистарских вриједности и рјешава проблем посљедње копије слично *past* и *present* скуповима регистара код Multiscalar-а. Контрола регистарске комуникације се, такође, обавља хардверски на основу праћења статуса сваког регистра у CSC-у (*Communication Scoreboard*) у сваком процесору [BARL2003].

Pinot користи једносмјерни прстен као интерконекцију између процесора за размјену регистарских вриједности без подршке за синхронизацију са експлицитним инструкцијама. Размјена регистарских вриједности је подржана новим инструкцијама (*fork*, *propagate* и *cancel*) које су додате у постојећу ISA и обавља се у тренутку генерисања или након генерисања нове нити [OHSA2005].

Код MAJC-а преводилац обезбјеђује да VLIW пакети не смију имати међузависности по питању података, токова контроле или коришћених ресурса. Глобални дијељени скуп регистара у сваком MAJC процесору омогућава брзу комуникацију дијељених регистарских вриједности између нити помоћу виртуелних канала као и синхронизацију између нити помоћу брзог прекидног механизма [TREM2000, SUDH2000].

Табела 3-1 сумарно приказује подршку за спекулацију на регистарском нивоу и начине размјене регистарских вриједности код датих спекулативних CMP-а.

Табела 3-1. Подршка за спекулацију и начини размјене регистарских вриједности

CMP	**Параметри**	
	Подршка за спекулацију	**Регистарска комуникација**
Multiscalar Multiplex	Двоструки скупови регистара по процесору Шест скупова бит маски по процесору	Инициранa од стране нити која је произвела регистарску вриједност
NEKO	IRB и CRB по процесору CSC за контролу комуникације Раније просљеђивање вриједности Подршка преводиоца	Инициранa од стране нити која је произвела регистарску вриједност
Pinot	*Fork*, *propagate* и *cancel* инструкције	Инициранa од стране нити која је произвела регистарску вриједност
MAJC	Глобални скуп регистара Виртуални канали Механизам брзог интерапта Подршка преводиоца	Синхронизована комуникација између нити која је произвела регистарску вриједност и нити која је потражује

Предикција регистарских вриједности. Предикција регистарских вриједности може бити веома корисна за побољшање укупних перформанси система. Коришћењем предикције могуће је раније обезбиједити регистарске вриједности него што је то био случај у претходно анализираним спекулативним CMP системима са различитим синхронизационим механизмима. Уколико се испостави да је предикција регистарских вриједности исправна, тада се спекулативне нити које имају међузависности извршавају као да су независне. Међутим, успјешност тако агресивне оптимизације спекулативног извршавања зависи од тачности предикције. Механизми предикције који се користе у SM, Atlas и Trace су хардверски подржани, док се код Mitosis-а базирају на софтверској подршци.

SM користи предикцију зависности регистарских вриједности за сваку нову спекулативну нит у односу на претходну, као и предикцију регистарских вриједности које ће се размјењивати између нити. Када се генерише нова спекулативна нит у SM тада се врши копирање регистарских вриједности из локалног скупа регистара претходне нити у локални скуп регистара нове нити. Након тога се сваки активни (*live*) регистар за којег је могуће извршити предикцију иницијализује предвиђеним вриједностима, док се за оне чије вриједности није могуће предвидјети користе вриједности из локалног скупа регистара са пренесеним вриједностима (*live-in*) [MARC1998].

Atlas користи AMA (*Atlas Multi Adaptive*) предиктор, много агресивнију корелисану предикцију регистарских вриједности него што је то случај код SM и Trace, да би се на тај начин избјегло слање регистарских вриједности другим

нитима на захтјев нити потрошача или без њега (*broadcast*). Током процеса завршетка неспекулативне нити врши се верификација предикције регистарских вриједности у односу на оне које је на захтјев иницирао од тренутне неспекулативне нити прослиједила претходна неспекулативна нит [CODR1998].

Зависности по подацима између инструкција код Trace су ублажене коришћењем предикције за глобалне улазне регистарске вриједности за сваки траг. Свако процесорско језгро има скуп регистара у којем се чувају регистарске вриједности добијене предикцијом, као и хардверску подршку за испитивање тачности предикције у односу на регистарске вриједности које су примљене од других трагова (нити). Провјера тачности предикције за глобалне улазне регистарске вриједности се обавља када се израчунате вриједности избаце на глобалне магистрале. Током процеса завршетка неспекулативног трага врши се слање само глобалних излазних регистарских вриједности свим копијама глобалног скупа регистара у сваком процесорском језгру, као и њихово мапирање у физичке регистре [ROTE1997].

Mitosis користи предикциону шему која се заснива на софтверској подршци помоћу које се врши предикција регистарских вриједности директно из изворног кода (*p-slice*) чиме се унапређује тачност предикције у поређењу са предикционим шемама које се заснивају на хардверској подршци као што је то случај код SM, Trace и Atlas. Поред тога, ова предикциона шема енкапсулира вишеструке токове контроле програма који доприносе израчунавању регистарских вриједности без коришћења било какве хардверске подршке за предикцију контроле тока родитељске нити као што је то случај код Trace-a. Провјера тачности регистарских вриједности које је генерисао *p-slice* и оних које је нит на свој захтјев добила од других нити се врши у односу на тренутне одговарајуће регистарске вриједности код претходне нити приликом процеса завршетка неспекулативне нити [MADR2008].

Коришћење предикције регистарских вриједности са великом тачношћу успјешно рјешава регистарске зависности између нити омогућавајући раније извршавање спекулативне нити. Међутим, мора се имати у виду да провјера тачности предикција уноси одређено кашњење. Карактеристике и функционалности шема предикције регистарских вриједности код разматраних спекулативних CMP процесора су представљене у табели 3-2.

Неспекулативна размјена регистарских вриједности. I-ACOMA избјегава потребу за увођењем двоструког скупа регистара за чување регистарских вриједности и рјешавање проблема посљедње копије, као код Multiscalar-a, Multiplex-a и NEKO-a, увођењем једноставнијих хардверских структура за подршку неспекулативне размјене регистарских вриједности између нити. Логика која одређује доступност регистара, као и логика за проналажење посљедње копије за сваки регистар, уз помоћ одређених хардверских компоненти је имплементирана у сваком процесорском језгру како би се омогућила подршка

пропагације сигурних регистарских вриједности у комуникацији која је иницирана или од стране произвођача или од стране потрошача. Међутим, коришћење дијељене магистрале за размјену регистарских вриједности између језгара уноси додатно кашњење узроковано арбитрацијом на магистрали [KRIS1999].

Табела 3-2. Предикција регистарских вриједности

СМР	**Параметри**	
	Карактеристике предикције	**Регистарска комуникација**
SM	Хардверска подршка Предикција тока извршавања и података	Инициран од стране нити која је произвела регистарску вриједност
Atlas	Хардверска подршка Предикција података	Провјера тачности вриједности добијених предикцијом иницирана од стране нити која је произвела регистарску вриједност
Trace	Хардверска подршка Предикција тока извршавања и података	Провјера тачности вриједности добијених предикцијом од стране нити која је произвела дате вриједности
Mitosis	Софтверска подршка Предикција података	Инициран од стране нити која је произвела регистарску вриједност или од нити која потражује регистарску вриједност

MP98 (Merlot) користи механизам насљеђивања регистарских вриједности тако што регистарске вриједности које произведе претходна нит насљеђује нова нит. Умјесто копирања свих регистарских вриједности, што би захтијевало велики пропусни опсег, MP98 (Merlot) користи глобални дијељени скуп регистара да копира само одговарајућу преименовану мапу како би се новој нити омогућило да користи исправне регистарске вриједности [EDAH2000, MATS2000].

У табели 3-3 су дате карактеристике неспекулативне размјене регистарских вриједности код I-ACOMA-е и MP98 (Merlot) процесора.

Табела 3-3. Неспекулативна размјена регистарских вриједности

СМР	**Параметри**	
	Подршка за синхронизацију података	**Регистарска комуникација**
IACOMA	Каталог са информацијом о стању регистара Логика за одређивање доступности регистра у сваком процесору Логика за одређивање посљедње копије регистра у сваком процесору	Инициран од стране нити која је произвела регистарску вриједност или иницирана од стране нити која потражује регистарску вриједност
MP98	Глобални скуп регистара	Механизам насљеђивања регистарских вриједности

3.2.2 Опоравак након погрешне спекулације

Подршка за регистарску комуникацију у спекулативним CMP системима, такође, треба да обезбиједи хардверску и/или софтверску детекцију нарушавања зависности по подацима типа RAW хазарда. Грешка у спекулацији се обично рјешава поништавањем нити која је изазвала нарушавање зависности података као и свих нити које имају већи степен спекулативности од ње. На тај начин се спречава да ниједна спекулативно произведена вриједност од стране нити која је изазвала нарушавање зависности не буде прослеђена до нити које је слиједе. Поништавање комплетног спекулативног рада у случају нарушавања зависности података између нити чини операцију поништавања нити веома скупом.

У спекулативним процесорима који су овдје представљени погрешна спекулација може бити узрокована или нарушавањем регистарских зависности између нити или погрешном предикцијом регистарске вриједности. Подршка за опоравак од погрешне спекулације може бити или чисто хардверска (нпр., NEKO, Pinot) или хардверско/софтверска (нпр., Multiscalar, Mulitplex). Код MP98 (Merlot) и I-ACOMA оваква подршка не постоји пошто у овим системима нема спекулације на регистарским вриједностима.

Нарушавање регистарских зависности. Multiscalar, као и Multiplex, користе двоструке скупове регистара да би чували исправна стања регистара током спекулативног извршавања и на тај начин омогућили опоравак регистарских стања у случају нарушавања регистарских зависности између нити. Уколико дође до грешке у спекулацији *present* маска се ажурира са *past* маском при чему се одбацују спекулативне вриједности које је произвела текућа нит и задржавају претходне произведене регистарске вриједности. Много комплекснији дио процеса опоравка је одбацивање регистарских вриједности које су већ прослеђене нитима са већим степеном спекулативности. За било коју регистарску вриједност која је произведена у спекулативној нити која је дио некоректног спекулативног извршавања се мора извршити њен опоравак. Да би се то обезбједило потребно је консултовати *recover* и *squash* бит маске како би се извршила идентификација прослеђених некоректних регистарских вриједности. *Recover* бит маска означава оне регистарске вриједности које спекулативна нит која је нарушила спекулацију мора прослиједити до спекулативнијих нити да би се извршила замјена некоректних регистарских вриједности. С друге стране, *squash* бит маска је комбинација *recover* бит маски свих спекулативних нити са нижим степеном спекулативности које су биле укључене у процес опоравка и користи се за синхронизацију опорављених регистарских вриједности. Иако је овај механизам у потпуности хардверски имплементиран, његова ефикасност је побољшана извјесном подршком од стране преводиоца [SOHI1995, OOI2001].

У случају грешке у спекулацији код NEKO процес опоравка прво треба да ресетује статус комуникације и затим да поново покрене процес ажурирања и

пропагације регистарских вриједности. Свака процесорска јединица у NEKO-у која треба да се рестартује мора ресетовати битове ажурности и пропагације у свом каталогу да би се рестартовао процес ажурирања и пропагације. Такође, мора се извршити инвалидација статуса регистарских вриједности ресетовањем одговарајућих битова у процесорском језгру које извршава спекулативнију нит [BARL2003].

Код MAJC-а неспекулативна нит или спекулативна нит са нижим нивоом спекулативности прослеђује регистарску вриједност кроз виртуелни канал до спекулативне нити која је узроковала погрешну спекулацију. На овај начин спекулативна нит која узрокује грешку у спекулацији добија актуелну регистарску вриједност произведену од стране мање спекулативних нити што јој помаже да одржи секвенцијалну семантику оригиналног програма [TREM2000].

Родитељска нит у Pinot-у провјерава тачност спекулације и у случају грешке у спекулацији отказује нит коју је генерисала као и све могуће сљедеће више спекулативне нити. Грешка у спекулацији у Pinot-у се детектује уз помоћ хардвера који је подржан минималним измјенама у извршном бинарном коду позивањем нове *fork* инструкције или посебне *cancel* инструкције када је грешка детектована [OHSA2005].

У табели 3-4 су дате сумарне информације о различитим подршкама за опоравак након погрешне спекулације у случају регистарских зависности између нити за разматране спекулативне процесоре.

Табела 3-4. Подршка за опоравак након погрешне спекулације

CMP	**Подршка за опоравак након грешке у спекулацији**
Multiscalar Multiplex	Двоструки скупови регистара по процесору *Recover* и *squash* бит маске по процесору
MAJC	Виртуелни канали
NEKO	Хардверска подршка (CSC и CRB по процесору)
Pinot	Хардверска подршка са модикацијама у ISA

Грешка у предикцији. Сваки пут када се изврши упис у неки од регистара у SM декрементира се његов одговарајући бројач у посебној *Rwrite* табели. Када вриједност бројача у табели за дати регистар постане мања од нуле, тада је дошло до непредвиђеног уписа и погрешне спекулације. Након тога се све сљедеће нити са вишим степеном спекулативности морају одбацити док је све физичке регистре који нијесу дијељени са претходном нити потребно ослободити [MARC1999].

Trace уводи модел селективног извршавања за управљање грешком у спекулацији када се догоди грешка у предикцији изворних регистарских вриједности. Када се израчунате регистарске вриједности пошаљу на глобалне

магистрале, тада се врши провјера предикције улазних глобалних вриједности. Ако се предвиђене вриједности не слажу са израчунатим, долази до погрешне спекулације и инструкције које користе предвиђене вриједности се морају поново извршити са новим вриједностима операнада. Постојећи механизам у Trace-у поново извршава само оне инструкције које су у ланцу зависности [ROTE1997].

Atlas користи модел опоравка фине грануларности у случају грешке у предикцији да би извршио само оне инструкције које зависе од нетачних података. Као посљедица тога, нити са већим степеном спекулативности неће бити одбачене чиме ће се омогућити њихово коректно извршавање и завршетак, чак и у случају грешке у предикцији регистарске вриједности код претходне, мање спекулативне нити [CODR2001].

У случају грешке у спекулацији код Mitosis-а спекулативна нит која је узроковала грешку као и све нити са већим степеном спекулативности од ње ће бити одбачене. Mitosis експлицитно прати коришћене регистарске вриједности помоћу додатне RVS (*Register Validation Store*) структуре која се налази у сваком процесору, а која се користи за провјеру тачности предвиђених регистарских вриједности. Нетачно предвиђене регистарске вриједности које су произведене од стране *p-slice*-а, али нијесу коришћене од стране дате спекулативне нити, неће узроковати грешку у спекулацији. Mitosis је релативно неосјетљив на одбацивање нити зато што је тачност предикције регистарских вриједности велика [MADR2008].

Сумарне информације о различитим подршкама за опоравак усљед погрешне предикције код спекулативних CMP-а који користе различите технике предикције регистарске вриједности да би разријешили регистарске зависности између спекулативних нити су дате у табели 3-5.

Табела 3-5. Подршка за опоравак спекулације након грешке у предикцији

CMP	**Подршка за опоравак спекулације након грешке у предикцији**
SM	*Rwrite* табела
Trace	Механизам селективног извршавања инструкција
Atlas	Модел опоравка фине грануларности
Mitosis	RVS структура за провјеру тачности *p-slice*-а

3.2.3 Перформансе

Евалуације спекулативних CMP процесора који подржавају комуникацију између спекулативних нити на регистарском нивоу су вршене у намјенским симулационим окружењима која вјерно моделују одређени спекулативни CMP уз коришћење различитих апликација. Иако је прилично тешко урадити поређење на овај начин добијених евалуационих резултата за различите спекулативне процесоре ипак је могуће дати одређене коментаре на неке од индикатора

перформанси у овим системима као што су: број инструкција по циклусу (IPC), кашњења и убрзања у односу на процесор са једним језгром или идеалну CMP конфигурацију. Поред тога, представљени су резултати евалуације перформанси за различите системе са становишта хардверске и/или софтверске подршке за комуникацију између нити на регистарском нивоу: организација скупа регистара, тип интерконекције између процесора и подршка за синхронизацију за размјену регистарских вриједности.

Организација скупа регистара. Евалуација MAJC-а са одређеним сигналним и видео апликацијама, као и апликацијама за обраду слике је показала ефикасност имплементације великог јединственог скупа регистара који може садржати било који тип података. На овај начин се обезбјеђује више регистара за апликације које укључују обраду намјенских типова података [TREM2000].

Хијерархијски модел скупа регистара који је имплементиран у Trace је евалуиран у односу на нехијерархијски модел са варијацијама пропусног опсега за регистарску комуникацију или кашњења у регистарској комуникацији. Резултати евалуције су показали да хијерахијски модел значајно побољшава перформансе (број инструкција по такту - IPC до 6.8) због тога што се код њега већи дио регистарске комуникације обавља преко локалних магистрала [ROTE1997].

Перформансе механизма за прослеђивање у I-ACOMA су упоређене са идеалним окружењем у којем не постоји кашњење у комуникацији – било која регистарска вриједност ажурирана од стране нити која ју је произвела је одмах видљива за нит која потражује дату регистарску вриједност. Додатна сложеност је прилично скромна пошто су само по један порт за читање и за упис додати за сваки скуп регистара. Због тога само једна регистарска вриједност може бити прочитана са магистрале или уписана на њу у датом циклусу што заузврат има мањи утицај на перформансе регистарске комуникације. Такође, репликацијом логика за доступност регистра и посљедњу копију регистарске вриједности за сваки регистар понаособ се уноси додатно усложњење. Међутим, резултати евалуације су показали да и поред тога I-ACOMA захтијева незнатно већу површину чипа у односу на 12-*issue* суперскаларни процесор [KRIS1999].

Дизајн Mitosis MRF структуре постиже очекиване перформансе пошто је средње кашњење регистарског приступа једнако кашњењу у појединачном LRF-у, односно 99% регистарских приступа код евалуираних апликација је било задовољено из LRF структуре. Овим се потврђује тачност *p-slice* концепта зато што је већина регистарских вриједности које су потребне током спекулативног извршавања произведена од стране одговарајућег *p-slice*-а [MADR2008].

Тип интерконекције између процесора. Перформансе једноставних конфигурација скупа регистара код Multiscalar-а са малим пропусним комуникационим опсегом између сусједних процесорских јединица од једног регистра по циклусу је сасвим упоредива са идеалном конфигурацијом са

неограниченим пропусним опсегом. Показало се да се мање од четвртине расположивог пропусног опсега стварно користи током спекулативног извршавања док редови за чекање између процесорских јединица садрже не више од једног регистра током већине времена извршавања. Такође, мали пораст кашњења у комуникацији између сусједних процесорских јединица има незнатан утицај на перформансе регистарске комуникације у Multiscalar-у [SOHI1995].

Успјешност подршке за регистарску комуникацију која је имплементирана у NEKO је евалуирана за различите конфигурације интерконекције између процесорских јединица за размјену регистарских вриједности, са промјенљивим кашњењем и пропусним опсегом ажурирања и пропагације. Варијација у кашњењу и пропусном опсегу ажурирања и пропагационог пута је показала да су перформансе више осјетљиве на кашњење и величину пропусног опсега ажурирања него на величину пропагационог пропусног опсега зато што се већина регистарских зависности између нити односи на двије сусједне нити. Међутим, пораст пропусног опсега ажурирања у конфигурацији интерконекције захтијева увођење додатних портова у скупу регистара. С друге стране, конфигурација интерконекције са напреднијим алгоритмом додјеле регистарских вриједности оним нитима које их већ потражују је дала исте резултате као и у случају конфигурације са повећаним пропусним опсегом ажурирања. Стога се може сматрати да је конфигурација интерконекције са напреднијим алгоритмом додјеле регистарских вриједности боље пројектно рјешење јер не захтијева имплементацију додатних портова. Остварене перформансе у погледу IPC-а су у просјеку само 6% ниже у односу на идеалну конфигурацију интерконекције са нултим кашњењем и неограниченим пропусним опсегом [BARL2003].

Осјетљивост организације скупа регистара у Trace-у на комуникационо кашњење глобалне магистрале је мјерена при неограниченом пропусном опсегу глобалне магистрале да би се изоловао утицај кашњења. Резултати су показали да је нехијерархијски модел организације скупа регистара смањио IPC од 20% до 35% јер дати модел и локалним и глобалним регистарским вриједностима додаје један цијели циклус за кашњење током размјене вриједности између нити што у крајњем доприноси увећању комуникационог кашњења за све вриједности. Са друге стране, хијерархијски модел скупа регистара разматра само глобалне вриједности што смањује овај губитак у перформансама на више од пола. Поред тога, евалуациони резултати су показали да захваљујући увођењу хијерархијског модела у контроли тока извршавања и току података Trace успјева да превазиђе комплексност и пројектна ограничења конвенционалних суперскаларних процесора [ROTE1997].

Евалуација пропусног опсега магистрале у I-ACOMA је показала да повећање пропусног опсега магистрале за више од једне ријечи по циклусу има врло незнатан утицај на перформансе система. У случају идеалног окружења са неограниченим пропусним опсегом побољшање је мање од 5% у односу на магистралу са пропусним опсегом од једног регистра по циклусу. Осим тога, чак

се и у случају највећег кашњења на магистрали од 3 циклуса између процесора који су највише удаљени не уочавају деградације перформанси. Ово показује да се комуникација између нити која је произвела дату регистарску вриједност и нити која потражује исту најчешће догађа између сусједних процесорских јединица у I-ACOMA упркос чињеници да је умјесто прстена као интерконекција за размјену регистарских вриједности коришћена дијељена магистрала [KRIS1999].

Мања осјетљивост перформанси на кашњење и величину пропусног опсега пропагационог пута указује на то да је интерконекција у виду прстена или дијељене магистрале довољна за успјешно управљање регистарском комуникацијом у спекулативним CMP-ма.

Убрзање у односу на процесор са једним језгром. Било би веома интересантно извршити међусобно поређење перформанси овдје представљених спекулативних процесора са подршком за регистарску комуникацију. Међутим, таква компаративна евалуација не постоји чак ни за само по два од наведених рјешења зато што је свако од њих евалуирано у различитим симулационим окружењима и са различитим методологијама па је коректна упоредна анализа практично немогућа. При томе, резултати евалуације скоро увијек изражавају комбиноване ефекте спекулације на меморијском и регистарском нивоу тако да је тешко изоловати само ефекте регистарске комуникације. И поред тога, имајући у виду сва наведена ограничења, у табели 3-6 су представљени резултати различитих евалуационих студија перформанси представљених рјешења спекулативних CMP-а са подршком за регистарску комуникацију са нагласком на највише коришћени индикатор перформансе у евалуацији паралелних система – убрзање (*speedup*) у односу на процесор са једним језгром.

Дати резултати обезбјеђују јасан доказ о потенцијалу спекулативних CMP процесора да издвоје паралелизам из секвенцијалних апликација и да их затим брже извршавају него што би се оне извршавале на једном процесору. Убрзање природно зависи од типа апликације. Његова вриједност је уобичајено већа за апликације са бројевима са покретним зарезом него за цјелобројне апликације с обзиром да у цјелобројним апликацијама већина програмских петљи има нижи број итерација и фреквентнију појаву управљачких инструкција унутар нити. Примјери Multiscalar и NEKO процесора указују на то да коришћење процесорских језгара са већим бројем инструкција које се могу истовремено извршити (*issue width*) не доприносе и већем убрзању спекулативног извршавања. С друге стране, код спекулативних CMP код којих се регистарска комуникација заснива на предикцији регистарских вриједности (нпр., Trace), тачност предикције је пресудна за перформансе система.

Табела 3-6. Убрзања спекулативних CMP у односу на процесор са једним језгром

CMP	Бр. ПЈ	Број инст.	Извршавање инструкција	Убрзање (средња вриједност)	Напомена	Тест апликације
Multiscalar	4	1	*in-order*	1.9	Напредна подршка преводиоца побољшава перформансе регистарске комуникације до 67% (*in-order*) и 59% (*out-of-order*)	SPEC92 suite GNU diffutils2.6 GNU textutils1.9
		2		1.7		
	8	1	*in-order*	2.8		
		2		2.6		
	4	1	*out-of-order*	2		
		2		1.6		
	8	1	*out-of-order*	3		
		2		2.4		
Multiplex	4	2	*out-of-order*	2.63	Досеже или надмашује перформансе само имплицитних или само експлицитних TLS CMP за скоро све коришћене тест апликације.	SPECfp95 suite Perfect suite
SM	4	4	*in-order*	1-3	Веће убрзање код апликација са бројевима са покретним зарезом.	SPEC95 suite
MP98	4	2	*in-order*	3	Процесорске јединице које се не користе се искључују у случају мањег апликацијског оптерећења.	Апликације за препознавање говора IDCT code
MAJC	2	4	*in-order*	1.6	Паралелизам на више нивоа (инструкције, подаци, нити и процеси) са брзим комуникационим механизмом.	Апликације за обраду видео записа и слике
Trace	8	4	*out-of-order*	1.1 - 1.25	Без предикције података.	SPECint95 suite
				1.08	Реална предикција података умањује добитак у перформансама.	
				1.45	Идеална предикција података увећава перформансе за скоро све коришћене апликације.	
IACOMA	4	4	*out-of-order*	1-2	За апликације са бројевима са покретним зарезом убрзање је у просјеку скоро два пута веће.	SPEC92 suite SPEC95 suite Media-Bench suite
Atlas	8	1	*in-order*	3.4	Тачност предикције вриједности и контроле тока програма, брзи механизам опоравка од грешке у предикцији и ефикасан начин дефинисања нити.	SPECint95 suite

Табела 3-6. (наставак)

СМР	Бр. ПЈ	Број инст.	Извршавање инструкција	Убрзање (средња вриједност)	Напомена	Тест апликације
NEKO	4	4	*out-of-order*	1.22	Интеграција процесорских јединица са мањим бројем инструкција које се истовремено издају увећава проток и одржава високе перформансе у секвенцијалним апликацијама током спекулативног извршавања.	SPECint95 suite
		2		1.28		
	8	4		1.24		
		2		1.32		
Pinot	4	4	*out-of-order*	3.7	Искоришћавање крупног паралелизма је од суштинског значаја за побољшање перформанси.	SPEC95 suite MiBench suite
Mitosis	4	6	*out-of-order*	2.2	Високо прецизна предикција зависности по подацима и ефикасна размјена података између нити.	Olden suite

3.2.4 Ограничења у скалабилности

Подршка за синхронизацију и комуникацију на регистарском нивоу је такође анализирана и са становишта скалабилности. Значајна питања која су разматрана у вези са скалабилношћу се односе на организацију скупа регистара, интерконекцију за размјену регистарских вриједности између процесорских јединица и подршку за сихнронизацију регистарских вриједности.

Организација скупа регистара. Иако глобална организација скупа регистара (нпр., MP98 и MAJC) штеди пропусни опсег у изузетно малим мултипроцесорима попут СМР, она намеће сложено ожичење, повећање броја портова за читање и упис и утиче на вријеме циклуса и кашњења. Због тога већина спекулативних процесора са подршком за регистарску комуникацију користи дистрибуиране скупове регистара да би се ублажили поменути негативни ефекти који су уочени у дизајнима са дијељеном организацијом скупа регистара.

Подршка за регистарску комуникацију. Хардверска подршка за комуникацију регистарских вриједности у Multiscalar, Multiplex и NEKO захтијева додатне хардверске структуре за чување регистара и решавање проблема посљедње копије регистарске вриједности за свако процесорско језгро што утиче на сложеност њихових дизајна. Подршка за спекулативно извршавање и предикцију у SM је у потпуности заснована на хардверским механизмима што заузврат веома утиче на комплексност свеукупног дизајна. Atlas користи

предиктор вриједности и контроле тока извршавања који је имплементиран на самом чипу при чему је његова комуникација са осталим компонентама преко дијељене магистрале, што такође утиче на ефикасно искоришћење простора и енергије.

С обзиром да су димензија каталога у I-ACOMA, као и хардверска подршка за утврђивање посљедње копије регистарске вриједности и доступности регистарске вриједности која се налази у сваком процесорском језгру, директно зависни од броја имплементираних процесорских језгара њихов дизајн утиче на укупну скалабилност система. Такође, повећање броја процесорских јединица у Mitosis-у ће узроковати повећање величине RVT (*Register Versioning Table*) структуре с обзиром да је њена димензија директно зависна од броја имплементираних процесорских јединица.

Интерконекција за размјену регистарских вриједности. Коришћење прстена, као природног избора интерконекције за размјену регистарских вриједности између процесорских језгара, је рјешење које је имплементирано у Multiscalar, Multiplex, SM, Atlas, NEKO и Pinot. Иако су цијена хардвера и кашњење у топологији прстена мали, она ограничава могућност да се изврши оптимизација локалности података и изведе успјешно истовремено извршавање више независних апликација. Такође, у случају повећања броја процесорских језгара мрежа у виду прстена није скалабилно рјешење. Евалуација скалабилности различитих Atlas дизајна са 4, 8, и 16 процесорских језгара је показала да се може пројектовати један ефикасан систем за извршавање секвенцијалних апликација. Међутим, даље повећање броја процесорских језгара (преко 16) резултује у лошијим перформансама (предикција тока програма и података има мању тачност), лошијем искоришћењу простора и већој потрошњи енергије него што је то случај код процесора са једним језгром [SASS2005].

Trace користи више глобалних магистрала док I-ACOMA и Mitosis користе дијељену магистралу као интерконекцију између процесорских језгара за комуникацију на регистарском нивоу. Укупне перформансе не зависе само од количине података која се преноси већ, такође, и од протокола на магистрали, од параметара магистрале (ширина магистрале, политике приоритета, величина блока података који се преноси, механизма арбитрације итд.), фреквенције такта која зависи од сложености спрежне логике, ожичења и распореда различитих компоненти, као и од активности компоненти на магистрали. Једноставна дијељена магистрала би требало да буде довољна да опслужи мањи број процесорских јединица (4 до 8) у CMP, али више процесорских јединица или чак брже процесорске јединице би могле захтијевати већи пропусни опсег што заузврат намеће имплементацију или више магистрала или хијерархију магистрала. С обзиром да рјешење интерконекције са магистралом може бити скупо рјешење, оно ће вјероватно бити и један од ограничавајућих фактора у

сложености дизајна процесорских језгара као и укупне скалабилности и перформанси CMP-а.

На крају, ограничења у скалабилности уочена код претходно анализираних спекулативних CMP-а који подржавају комуникацију на регистарском нивоу су сажета у табели 3-7.

Табела 3-7. Ограничења у скалабилности спекулативних CMP-а са подршком за регистарску комуникацију

CMP	**Ограничења скалабилности**
Multiscalar	Сложена подршка за синхронизацију и комуникацију регистарских вриједности у свакој процесорској јединици. Топологија прстена за размјену регистарских вриједности.
Multiplex	Сложена подршка за синхронизацију и комуникацију регистарских вриједности у свакој процесорској јединици. Топологија прстена за размјену регистарских вриједности.
SM	Хардверски механизми за спекулацију и предикцију. Топологија прстена за размјену регистарских вриједности.
MP98	Чврсто спрегнута архитектура. Глобални (дијељени) скуп регистара, захтјеви са већим бројем улаза (портова) за упис и читање, дугачка ожичења.
MAJC	Чврсто спрегнута архитектура. Глобални (дијељени) скуп регистара, захтјеви са већим бројем улаза (портова) за упис и читање, дугачка ожичења.
Trace	Глобални (дијељени) скуп регистара, захтјеви са већим бројем улаза (портова) за упис и читање, дугачка ожичења. Глобалне и локалне магистрале за размјену резултата извршавања нити
IACOMA	Логика за утврђивање доступности датог регистра која је имплементирана у свакој процесорској јединици. Логика за утврђивање посљедње регистарске копије која је имплеметирана у свакој процесорској јединици. Дијељена магистрала.
Atlas	Предиктор за регистарске вриједности и контролу тока програма. Дијељена магистрала. Топологија прстена за размјену регистарских вриједности.
NEKO	Сложена подршка за синхронизацију и комуникацију регистарских вриједности у свакој процесорској јединици. Топологија прстена за размјену регистарских вриједности.
Pinot	Топологија прстена за размјену регистарских вриједности.
Mitosis	RVT структура. Топологија прстена за размјену регистарских вриједности.

Глава 4. Меморијска комуникација код спекулативних CMP процесора

Приликом извршавања програма, само мањи дио података може да се држи у регистрима због њиховог ограниченог броја, а већина података се налази у меморијској хијерархији. Зато сваки од постојећих спекулативних CMP система подржава спекулацију на нивоу меморијске комуникације. Ради бољег разумијевања и овдје је, као у претходној глави, прво направљена класификација ових система по критеријуму организације меморијске хијерархије, а затим је спроведена детаљна анализа њихове хардверске и софтверске сложености подршке за спекулацију на меморијском нивоу.

4.1 Класификација спекулативних CMP процесора са меморијском комуникацијом

Организација меморијске хијерархије је један од одлучујућих фактора који утичу на перформансе и сложеност система. Оптимално рјешење се обично добија као компромис ова два индикатора. Перформансе меморијског система у мултипроцесору обично зависе од времена приступа дијељеним подацима, као и од величине дијела меморијске хијерархије приватног за сваки процесор, док сложеност зависи од броја копија једног податка у меморијском систему. У студији [NAYF1996] су евалуиране три архитектуре меморијског система при извршавању паралелних апликација и мултипрограмског оптерећења. Резултати приказани у табели 4-1 показују да су, у општем случају, за паралелне апликације перформансе боље и сложеност мања што је заједнички дио хијерархије ближи

процесорима, док су за мултипрограмско оптерећење перформансе боље што је интерконцекција даља од процесора.

Табела 4-1. Карактеристике архитектуре са дијељеном меморијом

Ниво перформанси	**Перформансе паралелне апликације**	**Перформансе мултипрограмског оптерећења**	**Сложеност**
Најбоље	Дијељена L1 кеш мем.	Дијељена главна мем.	Дијељена L1 кеш мем.
…	Дијељена L2 кеш мем.	Дијељена L2 кеш мем.	Дијељена L2 кеш мем.
Најлошије	Дијељена главна мем.	Дијељена L1 кеш мем.	Дијељена главна мем.

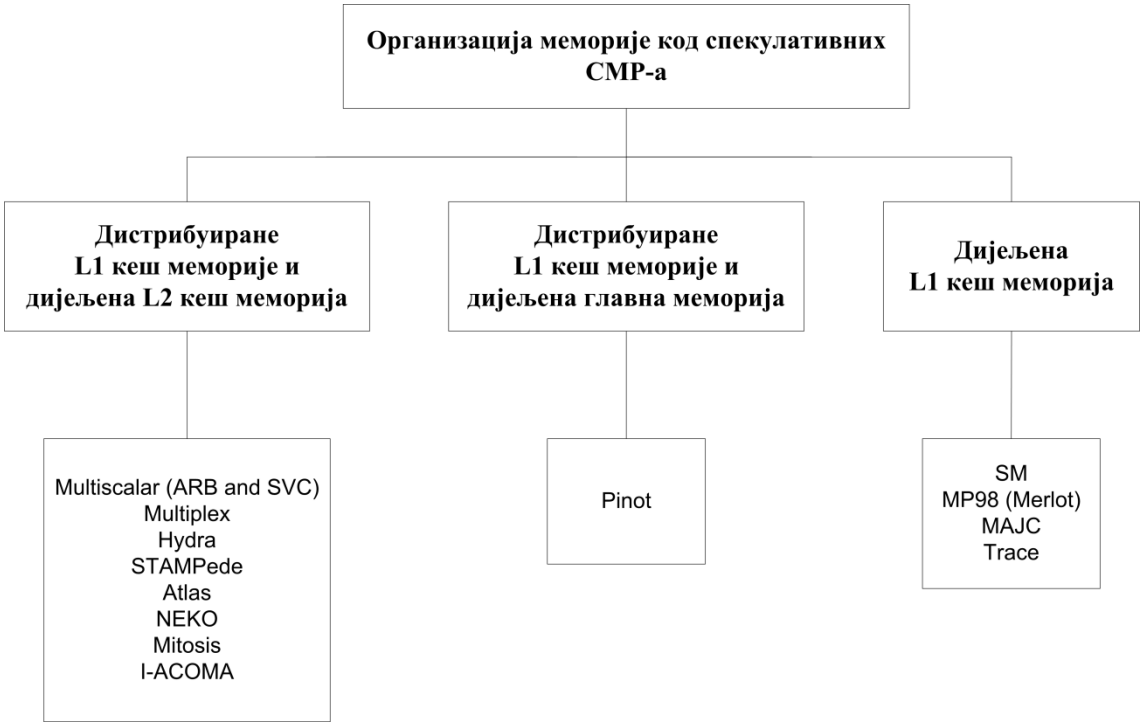
Организација меморијског система има значајан утицај на имплементацију механизама чувања, синхронизације и комуникације меморијских вриједности између процесорских језгара у спекулативним CMP системима. Због тога је она усвојена као главни критеријум класификације у овом прегледу. Постоје три могућа приступа у пројектовању меморијске хијерархије у CMP архитектурама: дијељена L1 кеш меморија, дистрибуиране L1 кеш меморије и дијељена L2 кеш меморија, као и дистрибуиране L1 кеш меморије и дијељена главна меморија.

Мотивација за увођење дистрибуираних L1 кеш меморија лежи у томе што се са овим приступом обезбјеђује велики пропусни опсег и мала латенција у односу на дијељену L1 кеш меморију. С друге стране, потреба за одржавање кохеренције на нивоу L1 кеш меморија лежи у томе што без апстракције дијељене меморије посао преводиоца постаје веома тежак.

Постоји значајна количина дијељених података између спекулативних нити зато што су такве нити добијене из програма који није писан као паралелан од самог почетка. Пошто спекулативни систем мора да прослиједи дијељене податке брзо и ефикасно, поред организације меморијског система која је усвојена као главни критеријум, као други класификациони критеријум који значајно утиче на дизајн спекулативних CMP процесора са хардверском и софтверском подршком за синхронизацију и комуникацију меморијских вриједности између језгара на чипу је топологија повезивања језгара и меморијског система.

Магистрала је главни избор у топологији мреже која је имплементирана код спекулативних процесора Multiscalar, Hydra, STAMPede, Atlas, Trace, NEKO, Pinot и Mitosis. Једноставна архитектура магистрале треба да буде довољна да опслужи мали број језгара на чипу (нпр., од 4 до 8 језгара). Међутим, већи број језгара или коришћење бржих језгара неминовно тражи већи пропусни опсег на магистрали што се може постићи увођењем више магистрала или хијерархије локалних и глобалних магистрала. Код спекулативних CMP процесора као што су SM, MP98 (Merlot), MAJC и I-ACOMA, мреже директне повезаности се користе за комуникацију меморијских вриједности између језгара на чипу.

На слици 4-1 је приказана класификација разматраних система добијена примјеном два поменута критеријума.



Слика 4-1. Класификација спекулативних CMP-ова са подршком за меморијску комуникацију између спекулативних нити

Спекулативни меморијски систем мора да обезбиједи и механизам за праћење читања и уписа дијељене меморије. Када спекулативнија нит прочита податак, а мање спекулативна га након тога упише, дошло је до нарушавања меморијских зависности по подацима. Детекција нарушавања зависности по подацима омогућава систему да утврди када нити нијесу у ствари паралелне, како би се нит која је изазвала нарушавање поново извршила са исправном вриједношћу податка. Такође, потребно је обезбиједити и механизам који омогућава да спекулативна меморија буде ресетована послије детекције нарушавања зависности по подацима; све спекулативне измјене стања меморије морају да буду одбачене, а ниједно стално стање меморије не смије да буде изгубљено у том процесу.

Када спекулативна нит успјешно заврши извршавање, мора се ажурирати стално стање меморије у коректном програмском поретку поштујући секвенцијалну семантику. Ово може да захтијева одлагање уписа од стране спекулативнијих нити који су се заиста догодили прије уписа од стране претходних, мање спекулативних нити. Такође, потребно је обезбиједити да претходна нит, било неспекулативна или мање спекулативна нит, не може да види ниједну промјену начињену од стране спекулативније нити пошто се то не би десило ни у оригиналном секвенцијалном програму.

У наставку ће бити анализирано како постојећи спекулативни CMP системи задовољавају захтјеве за коректно спекулативно извршавање уз помоћ меморијских протокола за спекулацију и кохеренцију са одређеном хардверском

и/или софтверском подршком који уносе додатну сложеност (*overhead*) у односу на генеричке CMP системе. На основу анализе ових уочених додатних трошкова могуће је извести одређене закључке који су од помоћи за дефинисање хардверско/софтверске подршке за меморијску комуникацију у предложеном спекулативном CMP рјешењу [RADU2006, RADU2007, RADU2007a].

4.2 Анализа меморијских протокола за спекулацију и кохеренцију у CMP процесорима

У спекулативним CMP системима подршка за коректно управљање спекулативним извршавањем је обично интегрисана у меморијске протоколе за спекулацију и кохеренцију и, по правилу, је веома сложена. Ова анализа има за циљ да размотри захтјевне елементе ове подршке у постојећим системима како би се на основу тога дошло до предлога ефикасног рјешења које би имало скромну хардверску и софтверску подршку и које би отклонило неке неефикасности.

У овој анализи подршке уочено је неколико аспеката:

а) сложеност хардверске подршке,

б) сложеност софтверске подршке,

в) неуједначен саобраћај (*bursty traffic*) приликом завршетка нити,

г) неприлагођена стратегија замјене.

4.2.1 Сложеност хардверске подршке за спекулацију

Сви спекулативни CMP системи широко користе различите хардверске механизме за рјешавање меморијских зависности по подацима и чување спекулативног стања. Ова подршка намеће потребу за, често, сложеном додатном хардверском логиком.

Централизовани ARB (*Address Resolution Buffer*) блок у Multiscalar-у је сличан конвенционалној кеш меморији за податке. Сваки улаз садржи адресни таг, по једно поље величине кеш ријечи за свако процесорско језгро и битова стања (3 бита) за сваки од података. Они служе за одређивање да ли је верзија валидна и да ли су вршене операције читања и уписа. Код дистрибуираног ARB-а свако процесорско језгро је повезано са приватном копијом података који се састоји од ARB-а и кеша за податке, па верзије података нису одмах доступне као код централизоване шеме. Структура улаза код оба ARB-а је слична, али код дистрибуираног ARB-а подаци нијесу стварне верзије већ дупликати уписа означени додатним битом стања. Нарушавање меморијских зависности по подацима се детектује помоћу битова стања за сваки упис у ARB и одговарајуће контролне логике. За разлику од других, ARB рјешење користи дијељену L1 кеш меморију тако да се шема кеш кохеренције не мијења директно. Главни недостатак ARB-а је у томе што све меморијске операције укључују кашњење

интерконекционе мреже која повезује ARB и процесорска језгра. Ова мрежа мора да омогући велики пропусни опсег за сва процесорска језгра слично организацији дијељеног кеша. Multiscalar, такође, укључује и хардверску подршку у виду секвенцера нити за предикцију контроле тока програма [FRAN1996].

SVC шема је предложена као алтернатива за ARB шему код Multiscalar-а. Она садржи приватне кеш меморије за свако процесорско језгро на чипу организоване слично као у симетричним мултипроцесорима са инвалидационим *snoopy* протоколима за кеш кохеренцију, са додатком посебних битова у кеш линије. SVC шема користи хардверску логику за контролу верзија података, VCL (*Version Control List*), које су организоване у облику VOL (*Version Ordering List*) листе. Поред додатних битова, свака кеш линија садржи идентификатор процесорског језгра чији L1 кеш има наредну верзију у листи за ту линију. Контролна логика управља спекулативним приступима меморији праћењем саобраћаја на магистрали. SVC шема има и механизме детекције нарушавања зависности по подацима и одбацивања нити која ју је узроковала као и свих наредних нити [GOPA1998].

Multiplex обједињује кеш кохеренцију са преименовањем меморијских референци и механизмом разрјешавања меморијских зависности у L1 кеш меморијама кроз MUCS протокол који је изведен из Multiscalar SVC протокола. Хардверска подршка за спекулативно извршавање нити у Multiplex-у је проширена са додатних 6 битова стања за сваку L1 кеш линију. MUCS протокол користи коначни аутомат са два скупа стања. Први обухвата стања које имплементирају кохеренцију између вишеструких копија података и у имплицитном и у експлицитном моду рада, а други обухвата стања која имплементирају спекулацију да би се одржао програмски распоред између више верзија у имплицитном моду рада [OOI2001].

Спекулативна шема кохеренције за сваку L1 кеш линију у Hydra-и користи додатне битове који означавају само оне кеш линије у које је извршен спекулативни упис, па могу да их читају сљедеће нити које треба да буду извршене на том језгру. За детекцију нарушавања зависности по подацима свака L1 кеш линија се проширује и са додатних 16 битова ради спречавања непотребних нарушавања зависности омогућавајући на тај начин преименовање меморијских локација које користе више нити. Осим ових додатних битова Hydra користи L2 секундарне бафере умјесто L1 кеш меморија за чување спекулативних вриједности док не постану сигурне да би се могле сачувати у L2 кешу који садржи само неспекулативне вриједности. Спекулативно стање се одржава на нивоу кеш линије у режиму тренутног уписа, а директно у меморију се уписују само неспекулативне вриједности. По један L2 секундарни бафер се додјељује свакој спекулативној нити па се уписи од различитих нити држе одвојеним. Постоји више L2 секундарних бафера него процесорских језгара да би се омогућила континуалност у спекулативном извршавању када ови бафери пребацују податке у дијељени L2 кеш. Иако увођење ових бафера поједностављује

протокол за кохеренцију, они се могу препунити и зауставити извршавање спекулативних нити. Поред овога, сваком процесорском језгру је придружен копроцесор који врши распоређивање нити на процесорска језгра који захтева додатни простор [OLUK2007].

У STAMPede-у уобичајени MESI протокол је проширен додатним стањима која подржавају спекулативно извршавање. За разлику од Hydra-е, спекулативне вриједности се чувају у L1 кеш меморијама за податке док не постану сигурне за пребацивање у виши ниво меморије. Када нит постаје неспекулативна потребно је осигурати да њене спекулативно модификоване кеш линије буду видљиве за остатак система. За ту сврху STAMPede користи посебан ORB (*Ownership Required Buffer*) бафер тако што у њему чува адресу сваке спекулативно модификоване и дијељене кеш линије за потребе једноставнијег проласка кроз L1 кеш меморију када ове кеш линије треба да буду потврђене у сљедећем меморијском нивоу [STEF2005]. За кодирање стања се користи 5 додатних битова за сваку кеш линију. Од тога се 3 бита користе за кодирање стања кохеренције док се преостала 2 бита користе за разликовање спекулативних од неспекулативних стања. Осим тога, сваком спекулативном стању је придружена структура која се састоји од идентификатора нити и редног броја у поретку меморијских приступа, затим адресе рутина за обраду прекида и погрешне спекулације, ORB бафера, као и бита који означава да ли је дошло до нарушавања зависности између података.

Спекулативне вриједности у I-ACOMA се током извршавања, такође, чувају у приватним L1 кеш меморијама. Свака ријеч у приватном L1 кешу је проширена са додатних 6 битова од којих се 2 бита користе за кохеренцију док остала 4 бита користе спекулативне нити за операције читања, писања, прослеђивања и инвалидације. Одржавање ових битова на нивоу кеш линије могло би да доведе до лажног дијељења, а тиме и до непотребних одбацивања нити, па се они одржавају на нивоу ријечи. Централизовани механизам разрјешавања меморијских зависности је заснован на посебној MDT (*Memory Disambiguation Table*) табели сличној каталогу за одржавање кохеренције у конвенционалном SMP систему. MDT је смјештена између приватних L1 кеш меморија и дијељене L2 кеш меморије и реализована у више банака. Улази у MDT се одржавају на нивоу кеш линије док се информације чувају на нивоу ријечи. Свака ријеч у MDT-у има битове читања и уписа за свако процесорско језгро. Такође, постоји могућност да се током спекулативног извршавања MDT попуни што може проузроковати заустављање спекулативне нити при покушају да уметне нови улаз у MDT. Коришћењем вриједности из MDT додатна логика која провјерава запис прије уметања у MDT одређује да ли је било која спекулативнија нит извршила прерано читање које проузрокује нарушавање меморијских зависности по подацима [KRIS1999].

Хардверска подршка спекулацији у NEKO обухвата динамички предиктор нити, јединицу за провјеру исправности нити и L1 кеш меморије за податке. Спекулативни подаци нити се чувају у L1 кеш меморијама који укључују

подршку за размјену података у случају промашаја при читању, као и подршку за детекцију нарушавања меморијских зависности. Модификација MSI протокола за кохеренцију користи чак 8 додатних битова за сваку ријечи у L1 кеш меморијама за податке као подршка за спекулативно извршавање. За идентификацију стања дате кеш ријечи се користе 4 бита, за избјегавање непотребних детекција нарушавања зависности по подацима три бита, а један бит означава да ли је кеш ријеч застарјела. Сложена је и додатна контролна логика у кеш контролеру за провјеру власништва над подацима и пренос података између процесорских језгара, детекцију нарушавања зависности по подацима и контролу прелаза између стања. С обзиром да се управљање спекулативним стањем обавља на нивоу кеш ријечи, додатни простор је значајно већи него код MSI. Процјена кашњења које уноси контролна логика је показала да она уноси више од пола укупног кашњења за већину операција протокола [YANA2003, BARL2004].

Atlas комбинује спекулацију нити са предикцијом вриједности и контроле тока па веома зависи од перформанси софистицираног предиктора. Међутим, имплементација предиктора, гдје хардвер врши и предикцију вриједности и њихову верификацију, проузрокује додатне просторне и временске трошкове. Предикција вриједности које се размјењују између нити је потребна да би се елиминисала потреба за експлицитном синхронизацијом, као и у циљу повећања искоришћења паралелизма и смањења нарушавања зависности по подацима. При опоравку од погрешне предикције вриједности података поново се извршавају само оне инструкције које зависе од погрешно предвиђених вриједности. Свако процесорско језгро као подршку за спекулацију укључује предиктор зависности између нити и њему придружени асоцијативни кеш у којем се чувају предвиђене вриједности, као и мали потпуно асоцијативни ред чији је задатак да прати активне предикције на процесорском језгру. Она обухвата и модификовану L1 кеш меморију гдје је свакој линији додат спекулативни бит који означава да ли дата кеш линија учествује у спекулацији, бафера у којем се чувају вриједности које су произведене током спекулације и компоненте за опоравак од погрешне спекулације која омогућава поновно извршавање само оних инструкција које зависе од нетачно предвиђених вриједности [CODR2001].

Основна компонента комплетно хардверске подршке за спекулацију у SM-у је вишевриједносни MVC (*Multi-Value Cache*) кеш меморија док се зависности између нити и вриједности које оне размијењују предвиђају на основу табеле итерација петље (*loop iteration table*). MVC кеш за сваку адресу чува онолико различитих ријечи колико има јединица за извршавање нити. За сваку копију ријечи, он садржи два додатна поља: предвиђени број уписа одговарајуће нити и бит који означава да ли је податак ове нити произведен. Сваки улаз садржи један бит присутности који означава да ли ти улази заиста садрже податке или их треба затражити од другог нивоа меморије. MVC кеш може да ради без локалне меморије, као замјена за L1 кеш, али се локална меморија, ипак користи, због побољшања перформанси, како би се искористила локалност у вриједностима

креираним од стране исте нити. Погрешне спекулације обрађује контролна логика тако што провјерава у свом load/store реду да ли постоји одговарајућа операција читања и уколико пронађе такву, поново је извршава као и инструкције које зависе од ње. Механизам за провјеру погрешне спекулације се, такође, активира при операцији уписа када се одговарајућа линија не налази у MVC кешу [MARC1998, MARC2003].

MP98 (Merlot) чува адресе и вриједности спекулативно произведених података у SRB (*Store Reservation Buffer*) баферима за свако процесорско језгро. Када се нит заврши, вриједности које је произвела се шаљу у заједничку L1 кеш меморију за податке, а ако се нит поништи све спекулативно произведене вриједности се поништавају. Да би се што ефикасније искористио мали број улаза у SRB баферу неспекулативни уписи се директно пропагирају у L1 кеш меморију за податке. У циљу смањења саобраћаја за приступ кешу за податке од стране процесорских језгара, када нијесу пуни, SRB бафери се користе као мали, потпуно асоцијативни кешеви. Поред тога, спекулативно извршавање код MP98 (Merlot) има хардверску подршку у виду FOPE (*Fork-Once Parallel Execution*) модела гдје свака нит генерише највише једну нову спекулативну нит што олакшава анализу зависности између нити по контроли тока и по подацима [EDAH2000, MATS2000].

Свако процесорско језгро у MAJC-у функционише као 4-*issue* VLIW машина која посједује хардверску подршку за вертикално вишенитно извршавање. Ово омогућава процесорском језгру да се пребаци на извршавање новог тока инструкција (промјена контекста) у кохерентној дијељеној L1 кеш меморији за податке. Поред тога, у сваком процесорском језгру постоји посебна LSU јединица која управља свим меморијским операцијама омогућавајући вишеструке операције читања и уписа, као и синхронизацију нити током спекулативног извршавања [TREM2000, SUDH2000].

Спекулативно произведене вриједности у Trace-у се чувају у посебном глобалном баферу који може бити организован као ARB у Multiscalar-у или интегрисан као дио L1 кеш меморије за податке код SVC шеме у истом систему [BREA1996]. Поред тога, у сваком процесорском језгру се налазе бафери који се користе приликом операција читања и уписа. Trace користи агресивну спекулацију да би ублажио ограничења узрокована зависностима по контроли тока и по подацима, користећи предикцију и за контролу тока програма и за вриједности које се размјењују између нити. Када нит почне извршавање свим њеним операцијама читања и уписа се придружују редни бројеви којима се одређује поредак меморијских операција током извршавања. Детекцију грешке у спекулацији врши логика дистрибуирана у процесорским језгрима праћењем операција читања на магистралама кеш меморија. Тако се поново извршавају само оне инструкције које су нарушиле спекулацију, као и инструкције које су са њима имале зависности по подацима.

Pinot користи хардверску подршку у виду VRC (*Version Resolution Cache*) кеш меморије који представља замјену за L1 кеш меморију за податке. У њему се

чувају спекулативне вриједности које нити произведу током спекулативног извршавања. Због тога се проширује таг сваке кеш линије додатним бит вектором како би се водило рачуна о позицијама сваке модификоване ријечи у кеш линији. Меморијске инструкције током спекулативног извршавања нити учитавају и уписују меморијске вриједности у различитом поретку што отежава разрјешавање меморијских зависности по подацима, па се зато користи хардверска подршка за праћење поретка извршавања нити [OHSA2005].

Mitosis укључује подршку за извршавање и провјеру исправности *p-slice*-ва као и меморијски подсистем који различитим нитима обезбјеђује подршку за чување више верзија за сваку меморијску локацију. У приватним L1 кеш меморијама за податке се чувају спекулативне вриједности, а свакој линији је додат бит стања за означавање застарјелих вриједности, као и три додатне структуре. Оне се користе за смјештај вриједности које произведе *p-slice* (*Slice Buffer*) и вриједности које буду измјењене од стране нових нити које извршавају *p-slice* (*Old Buffer*), као и репликациони кеш у који се уписују вриједности које произведе претходна нит и чија је намјена да се смањи проценат промашаја приликом операције читања када се он провјерава симултано са L1 кеш меморијом. У глобалном дијељеном L2 кешу се чувају подаци које је учитала неспекулативна нит, док централизована контролна логика VCL (*Version Control Logic*) управља поретком спекулативних верзија меморијске локације као код SVC шеме у Multiscalar-у [MADR2008].

Елементи хардверске подршке за меморијску комуникацију код разматраних спекулативних CMP система су сажето представљени у табели 4-2.

Табела 4-2.Елементи хардверске подршке за меморијску комуникацију

CMP	**Категорије хардверске подршке**			
	Додатни битови стања	**Додатни меморијски бафери**	**Подршка за разрјешавање зависности и одржавање спекулатив. верзија**	**Подршка за предикцију**
Multiscalar (централизовани ARB)	3 бита за сваку ријеч	N/A	Битови стања + централизована контролна логика	Секвенцер нити
Multiscalar (дистрибуирани ARB)	3 бита за сваку ријеч + 3 бита за дупликате уписа	N/A	Битови стања + контролна логика по процесорском језгру	Секвенцер нити
Multiscalar (SVC)	6 битова + показивач за сваку линију	N/A	L1 битови стања + VCL + VOL	Секвенцер нити
Multiplex	6 битова за сваку линију	N/A	L1 битови стања + MUCS аутомати стања	Секвенцер нити
Hydra	18 битова по линији	Секундарни меморијски бафери + *victim* кешеви	L1 битови стања + секундарни меморијски бафери + копроцесор по сваком процесорском језгру	N/A

Табела 4-2. (наставак)

CMP	**Категорије хардверске подршке**			
	Додатни битови стања	**Додатни меморијски бафери**	**Подршка за разрјешавање зависности и одржавање спекулатив. верзија**	**Подршка за предикцију**
STAMPede	5 битова по линији + број нити + флаг за спекулацију	ORB + victim кешеви	L1 битови стања + бројеви нити + логика за њихово поређење	N/A
I-ACOMA	6 битова по ријечи + 8 битова за сваки улаз у MDT	N/A	L1 битови стања + MDT + контролна логика у сваком процесорском језгру за провјеру операције уписа	N/A
NEKO	8 битова по ријечи	N/A	L1 битови стања + јединица за провјеру исправности нити + контролна логика у кеш контролеру	Динамички предиктор нити
Atlas	1 бит по ријечи	Спекулативни кеш за податке + вриједносни кеш по процесорском језгру	Спекулативни кеш за податке + вриједносни кеш по процесорском језгру + глобална логика за контролу нити	Софистицирани глобални предиктор вриједности/контроле тока и локални предиктори вриједности
SM	Бит валидности по ријечи + бит присутности за сваки улаз у кеш	*load*/*store* редови по процесорком језгру	Контролна логика по процесорском језгру	Табела итерација петље
Trace	*Store Buffer* организован као ARB или SVC	*load*/*store* бафери по процесорском језгру	*Store Buffer* организован као ARB или SVC	Динамички предиктор нити и предиктор вриједности
MP98 (Merlot)	N/A	SRB бафери	FOPE	N/A
MAJC	N/A	LSU по процесорском језгру	LSU по процесорском језгру + хардверска подршка за вишенитно извршавање	N/A
Pinot	Бит вектор – 3 бита стања по линији + 8 битова по линији за праћење историје учитавања + 32 бита за праћење историје уписа + 3 бита за број процесорског језгра		VRC	N/A
Mitosis	1 бит по линији	*Slice Buffer* + *Old Buffer* + репликациони кеш	Логика за контролу вишеструких верзија (VCL)	N/A

4.2.2 Сложеност софтверске подршке

За идентификацију спекулативних нити за паралелно извршавање већина разматраних система користи софистициране преводиоце (Multiscalar, Hydra, STAMPede, Atlas, Trace, NEKO, MP98 (Merlot), MAJC, Mitosis) или посебно развијене алате за паралелизацију секвеницијалних апликација (нпр., I-ACOMA, Pinot). Такође, за потребе спекулативног извршавања поједини системи користе и

методе предикције контроле тока програма и/или вриједности које се размјењују између спекулативних нити (нпр., Mitosis, MAJC, STAMPede).

Селекција нити из секвенцијалних апликација значајно утиче на укупне перформансе. У циљу постизања бољих перформанси Multiscalar преводилац користи хеуристике како би се побољшала селекција нити са карактеристикама погодним за спекулативно извршавање. Најважније хеуристике се односе на величину нити, број спекулативних нити које слиједе нит која се извршава, зависности по подацима које треба да буду укључене унутар саме нити како би се избјегла синхронизациона кашњења или грешке у спекулацији [SOHI1995].

Hydra користи копроцесоре који извршавају софтверске рутине за контролу поретка извршавања спекулативних нити, док STAMPede, такође, користи посебне софтверске рутине као подршку за спекулативно извршавање. Осим тога, Hydra и STAMPede захтијевају поновно превођење изворног кода апликације што може представљати озбиљан проблем када изворни код није доступан [OLUK2007, STEF2005]. Поред тога, преводилац у STAMPede-у подржава и предикцију контроле тока програма [STEF2005].

У MP98 (Merlot) преводилац умеће посебне инструкције за генерисање нових спекулативних нити као и за провјеру тачности или поништавање спекулације на нивоу контроле тока програма [EDAH2000, MATS2000].

Одржавање коректних меморијских зависности по подацима између нити током спекулативног извршавања у MAJC-у је засновано на преводиоцу. STC (*Space Time Consuming*) техника, која се користи у вишенитном моделу у MAJC-у омогућава да се спекулативне нити извршавају у њиховом сопственом простору који се зове „*speculative heap*“ и временском домену „*future time“* док се неспекулативна нит извршава у текућем времену и актуелном меморијском простору. MAJC преводилац подржава статичку предикцију контроле тока извршавања програма тако што директно преноси информацију хардверу да ли се очекује гранање програма или не [TREM2000, SUDH2000].

Извлачење спекулативних нити из секвенцијалних апликација се у Atlas-у врши динамички коришћењем новог MEM-*slicing* алгоритма који за границе нити узима меморијске инструкције читања и уписа [CODR1999, CODR2001].

Pinot издваја спекулативне нити широког спектра гранулације из секвенцијалних апликација без помоћи преводиоца користећи, као и I-ACOMA, посебни софтверски алат за паралелизацију [OHSA2005].

Преводилац за Mitosis, поред генерисања спекулативних нити, обавља и генерисање одговарајућих *p-slice* кодова. Они представљају подскупове инструкција за предикцију тачних улазних вриједности које ће спекулативна нит користити током свог извршавања [MADR2008].

Елементи софтверске подршке код разматраних спекулативних CMP система су сажето приказани у табели 4-3.

Табела 4-3. Елементи софтверске подршке за меморијску комуникацију

CMP	Категорије софтверских подршки				
	Рутине за подршку спекулацији	**Поновно превођење изворног кода**	**Идентификација нити помоћу преводиоца**	**Идентификација нити помоћу намјенских алата**	**Подршка за предикцију**
Multiscalar (ARB)	Не	Не	Да	Не	Не
Multiscalar (SVC)	Не	Не	Да	Не	Не
Multiplex	Да	Не	Да	Не	Не
Hydra	Да	Да	Да	Не	Не
STAMPede	Да	Да	Да	Не	Да
IACOMA	Не	Не	Не	Да	Не
NEKO	Не	Не	Да	Не	Не
Atlas	Не	Не	Да	Не	Не
SM	Не	Не	Не	Не	Не
Trace	Не	Не	Да	Не	Не
MP98 (Merlot)	Да	Не	Да	Не	Не
MAJC	Да	Не	Да	Не	Да
Pinot	Не	Не	Не	Да	Не
Mitosis	Да	Не	Да	Не	Да

4.2.3 Неуједначен саобраћај

Протоколи за спекулацију и кохеренцију у спекулативним CMP системима, као што су Multiscalar (ARB), Hydra, I-ACOMA, STAMPede, Atlas, Trace, MP98 (Merlot), MAJC, NEKO, Pinot и Mitosis, омогућавају измјештање неспекулативног садржаја из приватних кеш меморија за податке, односно структура које се користе за чување спекулативних вриједности током спекулативног извршавања у виши ниво меморијске хијерархије (дијељену L1 кеш меморију за податке, дијељену L2 кеш меморију, дијељену главну меморију) само приликом завршетка неспекулативне нити. У том тренутку се јавља неуједначен саобраћај (*bursty traffic*) на интерконекционој мрежи који може увећати вријеме потребно за завршетак неспекулативне нити и прослеђивање неспекулативног статуса сљедећој нити, као и увећати кашњење приликом додјељивања нове спекулативне нити за извршавање што може деградирати перформансе система.

Код Atlas и Mitosis је још карактеристично да се неспекулативне вриједности током извршавања претходно шаљу на магистралу како би се извршила потребна верификација и, у случају погрешне предикције, урадила њихова корекција. Ово, такође, има додатни утицај на саобраћај на заједничкој магистрали и вријеме потребно за завршетак неспекулативне нити [CODR2001, MADR2008].

Током спекулативног извршавања L1 кеш меморије у I-ACOMA раде у ограниченом режиму одложеног уписа (*write-back*) и није им дозвољено да модификоване кеш линије пребацују у виши ниво меморије. Међутим, када

спекулативна нит стекне неспекулативни статус модификоване кеш линије се могу измјестити из L1 кеш меморија и тада оне почињу да раде у режиму директног уписа (*write-through*) [KRIS1999].

Hydra користи L1 кеш меморије са директним уписом тако да се спекулативни подаци уписују директно у L2 секундарне бафере, све док нит не постане неспекулативна и треба да буде завршена. Тада измјене могу да буду укључене у перманентно стање дијељене L2 кеш меморије [OLUK2007].

За разлику од претходно наведених система Multiscalar SVC шема приликом завршетка неспекулативне нити задржава измијењене верзије у L1 кеш меморијама које касније уписује назад у меморију при спекулативном приступу тим линијама кеша. Тиме се спречава проблем са загушењем магистрале током завршетка неспекулативне нити пошто се упис у меморију одлаже док то заиста не буде потребно. Такође, ако се касније појави новија верзија кеш линије могуће је да та линија не буде никада уписана назад у меморију [GOPA1998].

У табели 4-4 су сажето приказани поједини елементи који могу да проузрокују неуједначен саобраћај приликом завршетка неспекулативне нити на интерконекционој мрежи код спекулативних CMP система.

4.2.4 Стратегија замјене

Замјена спекулативно модификованих кеш ријечи или линија у L1 кеш меморијама за податке или хардверским компонентама предвиђеним за чување спекулативних вриједности током спекулативног извршавања у протоколима за кохеренцију и спекулацију код свих разматраних спекулативних CMP система није дозвољена јер би се, на тај начин, промијенило неспекулативно стање у вишем нивоу хијерархије. У том случају се врши привремено заустављање спекулативне нити која покушава да изврши замјену спекулативног податка све док она не добије неспекулативни статус или не буде покренуто њено поновно извршавање (нпр., Multiscalar (ARB), Hydra, I-ACOMA, Atlas). У STAMPede-у, међутим, протокол за кохеренцију и спекулацију ће исту ситуацију посматрати као нарушавање меморијских зависности по подацима што ће у крајњем резултовати поништавањем дате спекулативне нити и свих нити које је слиједе.

Hydra и STAMPede користе *victim* кеш уз сваки L1 кеш за податке као мјеру предострожности да би елиминисали проблем заустављања процесорског језгра односно поништавање спекулације, респективно, ако L1 кеш за податке покуша да избаци спекулативну кеш линију. Међутим, *victim* кеш је коначан ресурс који се током извршавања може препунити што, на крају, ипак може проузроковати заустављање процесорског језгра код Hydra-е [OLUK2007], односно поништавање спекулације код STAMPede-а [STEF2000, STEF2005].

Кеш линија у MVC кешу код SM се сматра кандидатом за замјену само уколико су све спекулативне нити за које је предвиђено да ће вршити упис у дату кеш линију завршиле са својим извршавањем. Такође, уколико се ради о замјени

модификоване кеш линије иста ће бити размотрена као кандидат за замјену само у случају да нема активних спекулативних нити [MARC1998]. MAJC дозвољава само неспекулативној нити да изврши замјене кеш линија из заједничке L1 кеш меморије за податке [TREM2000, SUDH2000].

Табела 4-4. Узроци који могу да проузрокују неуједначен саобраћај

CMP	**Узроци за неуједначен саобраћај**		
	Режим уписа у L1 кеш	**Руковање модификованим подацима на завршетку неспекулативне нити**	**Увећање времена завршетка нити и кашњења у иницијализацији нове нити**
Multiscalar (ARB)	Одложени упис	Модификовани подаци се шаљу у дијељени L2 кеш	Да
Multiscalar (SVC)	Одложени упис	Модификовани подаци се задржавају у приватним L1 кешевима	Не
Hydra	Директни упис	Модификовани подаци се из секундарних бафера шаљу у дијељени L2 кеш	Да
STAMPede	Одложени упис	Модификовани подаци се шаљу у дијељени L2 кеш	Да
IACOMA	Одложени упис у спекулативном моду и директан упис у неспекулативном моду	Модификовани подаци се шаљу у дијељени L2 кеш	Да
NEKO	Одложени упис	Модификовани подаци се шаљу у дијељени L2 кеш	Да
Atlas	Одложени упис	Модификовани подаци се шаљу у дијељени L2 кеш и ка свим нитима сљедбеницама ради провјере предикције	Да
SM	Одложени упис	Модификовани подаци се шаљу у главну меморију	Да
Trace	Одложени упис	Модификовани подаци се шаљу у дијељени L1 кеш	Да
MP98 (Merlot)	Одложени упис	Модификовани подаци се шаљу у главну меморију	Да
MAJC	Одложени упис	Модификовани подаци се шаљу у главну меморију	Да
Pinot	Одложени упис	Модификовани подаци се шаљу у главну меморију	Да
Mitosis	Одложени упис	Модификовани подаци се шаљу у дијељени L2 кеш и ка свим нитима сљедбеницама ради провјере предикције	Да

Уколико је потребно извршити замјену неспекулативних вриједности из одговарајуће L1 кеш меморије или компоненти које подржавају вишеструке верзије за сваку меморијску адресу, које контролише било неспекулативна или спекулативна нит, на основу доступних података у литератури за одређене спекулативне CMP системе, као што су Multiscalar (SVC), Multiplex, Hydra, Trace, Pinot и SM, примјењује се LRU (*Least Recently Used*) стратегија замјене. Према томе, избор кандидата за замјену се ради на основу уобичајеног критеријума историје приступа не узимајући у обзир никакве специфичности спекулативних система.

Глава 5 Предлог подршке за спекулацију на нивоу нити у CMP систему

Претходна анализа подршке за спекулативно извршавање нити у CMP систему показује да је она често врло сложена, посебно у системима из прве групе потпуно оријентисаним ка спекулативном извршавању. Међутим, када се извршава мултипрограмско оптерећење и у правим паралелним апликација, та подршка остаје неискоришћена. Системи претежно генеричке архитектуре из друге групе користе само спекулацију на меморијском нивоу што, ипак, не даје оптималне предуслове за спекулативно извршавање. Због тога је одлучено да предложени систем по свом приступу припада трећој групи хибридних система који имају основну генеричку CMP архитектуру надграђену подршком и за регистарску и за меморијску комуникацију.

Основни циљ је био да се предложи систем са добрим односом цијена/перформансе са једноставнијом хардверско/софтверском подршком за комуникацију и на регистарском и на меморијском нивоу без потребе за поновним превођењем изворног кода секвенцијалних апликација. Поред тога, циљ је био да се отклоне и неке могуће неефикасности препознате у четвртој глави, као што су неуједначен саобраћај при завршавању нити и неприлагођена стратегија замјене у кеш меморијама.

У наставку ће укратко бити представљена структура предложеног CMP рјешења са TLS подршком, затим начин идентификације нити и спекулативног извршавања, као и хардверско/софтверска подршка за спекулацију нити на регистарском и меморијском нивоу.

5.1 Структура предложеног система

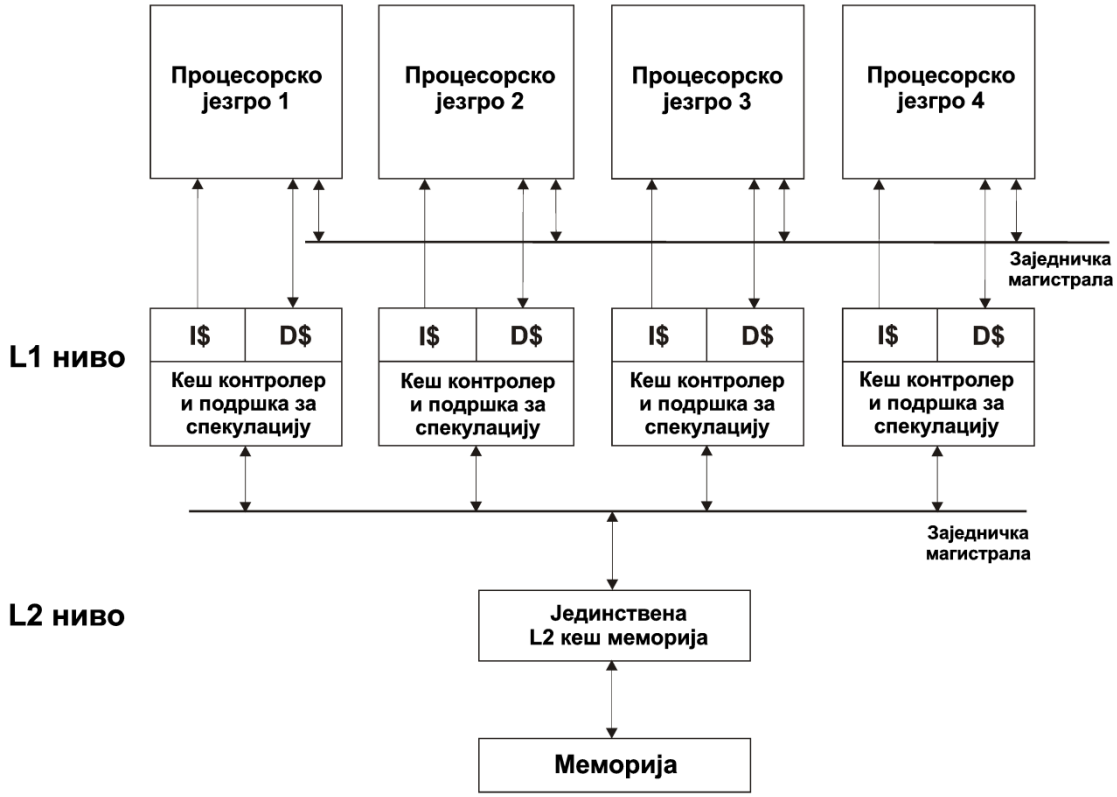
Претпостављена структура CMP система за који се у овом раду предлаже подршка за спекулативно извршавање је слична системима I-ACOMA [KRIS1999], Atlas [CODR2001] и Neko [BARL2003, YANA2003]. Ранија истраживања су показала да се меморијска архитектура са приватним L1 кеш меморијима и заједничком L2 кеш меморијом показала као боља алтернатива у поређењу са заједничком L1 кеш меморијом јер има добру скалабилност и не тражи сложене интерконекционе мреже између језгара и банки L1 кеш меморије. На овај начин се постиже и бржи приступ и већи укупан меморијски капацитет првог нивоа. Такође, показано је да су перформансе секвенцијалне апликације боље са овом архитектуром кеш меморија [MADR2008, STEF2005, SASS2005, BARL2003, OHSA2005, STEF2000, TSAI1999].

Према томе, CMP систем се састоји од четири процесорска језгра од којих свако има приватне L1 кеш меморије за податке и инструкције повезане заједничком магистралом са јединственом L2 кеш меморијом која је повезана са главном меморијом која се налази ван чипа (слика 5-1). Преко ове магистрале се обавља спекулативна комуникација на меморијском нивоу. Оба нивоа кеш меморије користе стратегију одложеног уписа (*write-back*). Иако би за чување спекулативног стања идеалан био потпуно асоцијативни начин мапирања, цијена имплементације налаже коришћење сет-асоцијативног начина мапирања.

Спекулативни CMP систем омогућава међупроцесорску комуникацију и на нивоу регистара и на нивоу меморије. Регистарска комуникација између процесорских језгара током спекулативног извршавања се обавља преко посебне заједничке магистрале (слика 5-1). С обзиром да се и регистарска и меморијска комуникација обављају на принципима „*snoopy*“ протокола, сложеност ове подршке практично не зависи од броја процесорских језгара, па у зависности од пропусности и технолошких параметара обје заједничке магистрале број процесорских језгара са приватним L1 кеш меморијама се може лако повећати.

5.2 Идентификација нити и поступак спекулације

Као и код већине других система описаних у прегледу, и у овом предлогу спекулативна палалелизација се спроводи на нивоу петљи јер у њима постоји највећи потенцијал за паралелизацију. Уколико су петље угњеждене, онда се спекулативно паралелизује унутрашња петља јер она доминантно утиче на вријеме извршавања. Према томе, спекулативне нити одговарају појединим итерацијама ове петље.



Слика 5-1. Структура предложеног система

Спекулативни метод и поступак су слични оном који се примјењује у I-ACOMA [KRIS1999]. За разлику од система који издвајају нити у вријеме извршавања, идентификација нити се врши прије извршавања уз помоћ посебне софтверске компоненте – бинарног анотатора. Он ради над секвенцијалним извршним, бинарним кодом тако да се избјегава потреба за поновним превођењем изворног кода. С обзиром да представљају итерације исте петље, све нити имају исти код, а анотатор означава почетне и завршне тачке нити. Поред тога, за разлику од система I-ACOMA, овдје анотатор има и посебну одговорност да за потребе протокола који контролише регистарску комуникацију изврши класификацију инструкција уписа у промјењиве (*loop-live*) регистре чије се вриједности могу модификовати у петљи, што ће касније бити објашњено у наредној, шестој глави.

Секвенцијална апликација почиње да се извршава на једном процесорском језгру. Када се стигне до улазне тачке петље, почињу да се извршавају четири прве итерације као паралелне нити на четири процесорска језгра. Прва текућа итерација се извршава као неспекулативна нит, док се наредне три извршавају као спекулативне нити. Када се неспекулативна нит заврши, статус спекулативности се ажурира. Наредна, најмање спекулативна нит добија статус неспекулативне нити, а статус спекулативности за остале се помјера. На процесорском језгру које је постало слободно након завршетка претходне неспекулативне нити почиње да

се извршава нова нит која добија најспекулативнији статус. Да би се очувала семантика секвенцијалног извршавања, нити се стартују у њиховом програмском поретку, а и ефекти нити морају да се појављују строго у том поретку. Према томе, ако нека спекулативна нит стигне до краја извршавања, мора да чека да постане неспекулативна да би се њени резулатати потврдили и да би ослободило процесорско језгро за извршавање нове нити.

Спекулативни статус нити која се тренутно извршава на језгру се означава двобитним кодом у посебном регистру. Неспекулативна нит (код 00) извршава текућу итерацију петље, док спекулативни сљедбеници извршавају прву (маска 01), другу (маска 10) и трећу (маска 11) сукцесивну спекулативну итерацију, респективно. Према томе, када се неспекулативна нит заврши њена наредна нит постаје неспекулативна док се маске осталих нити које се извршавају декрементирају. Нова нит почиње извршавање на ослобођеном процесорском језгру и добија највећи спекулативни статус (маска 11). Када се посљедња итерација петље заврши као неспекулативна, све наредне спекулативно започете нити се поништавају.

Спекулативно стање нити се за вријеме њеног извршавања држи у приватној L1 кеш меморији док заједничка L2 кеш меморија чува секвенцијално стање апликације (слика 5-1). Спекулативно стање може бити пренесено у L2 кеш меморију само када нит постане неспекулативна. Спекулативна нит може да чита податке од неспекулативне нити или од претходних спекулативних нити. Такође, спекулативна нит може да чита неспекулативне податке и од наредних нити уколико исти постоје у њиховим кеш меморијама. Уколико се динамички препозна погрешна спекулација (тј., ако тако прочитана вриједност буде касније промијењена од стране неспекулативне или неке раније мање спекулативне нити – RAW хазард), тада се спекулативна нит која је прочитала и користила погрешну вриједност поништава и касније поново извршава са исправном вриједношћу. Поред тога, поништавају се и све наредне спекулативне нити јер су могле користити податке произведене од стране спекулативне нити која је детектовала погрешну спекулацију.

5.3 Подршка за спекулацију на нивоу регистара

Иако подршка за комуникацију на нивоу регистара повећава сложеност система (брза интерконекција за размјену вриједности, додатна логика, итд.), као што се види и из прегледа у трећој глави, ранија истраживања су закључила да се исплати избјећи захтјевну меморијску комуникацију која намеће инструкције за читање и упис вриједности које се размјењују преко меморије, као и синхронизацију нити. Показано је у [KECK1998] да је регистарска комуникација 10 пута бржа, а синхронизација 60 пута бржа од одговарајућих механизама на нивоу меморије. Утицај изостанка регистарске комуникације на укупне перформансе система је евалуиран у [KRIS1998]. Показано је да комуникација

само кроз L2 кеш меморију (а не и кроз регистре) у CMP са четири суперскаларна 4-*issue* језгра доводи до деградације перформанси чак и до 50%. Према томе, ово је била јака мотивација да се у предложено рјешење укључи и подршка спекулацији на нивоу регистара.

Циљ ове подршке је био да омогући минималну додатну хардверску сложеност у односу на сличне приступе (нпр., I-ACOMA, NEKO) одржавајући латенцију регистарске комуникације што мањом. У том смислу предложено рјешење избјегава заједничке скупове регистара са више портова, додатне скупове за чување коректних вриједности, предикционе технике, захтјевне комуникационе механизме и сложене табеле статуса. За комуникацију на регистарском нивоу су предложена два алтернативна протокола: а) основна варијанта - SIC протокол [RADU2005] и б) проширена варијанта – ESIC протокол [RADU2005a]. Ови протоколи имају за циљ да омогуће једноставност, скалабилност, као и бољи однос цијена/перформансе у поређењу са сличним приступима из литературе (нпр., I-ACOMA, NEKO). Они су намијењени за генеричку CMP архитектуру и зато укључују само неопходну подршку за спекулативно извршавање.

Како се комуникација регистарских вриједности врши се преко посебне заједничке магистрале, ови протоколи су природно засновани на принципима „*snoopy*“ протокола за кохеренцију кеш меморија [TOMA1993]. Акције протокола се имплементирају у хардверу. Међутим, функционисање се ослања и на софтверску подршку у виду бинарног анотатора који, поред идентификације нити, треба да класификује и на одговарајући начин означи инструкције уписа у промјенљиве регистре чије се вриједности могу модификовати у петљи у односу на то да ли формирају финалне вриједности или не. Протокол води рачуна и о регистрима чије се вриједности не мијењају у петљи (константни регистри). Тренутни статус вриједности регистра у протоколу се чува у виду битова стања у локалном каталогу у сваком процесорском језгру посебно.

SIC протокол има конзервативнији приступ и прослеђује само коначне регистарске вриједности. Он користи четири стања за праћење статуса промјенљивих регистара: INV (*Invalid*), VU (*Valid-Unsafe*), VS (*Valid-Safe*) и LC (*Last Copy*), док константни регистри могу бити у INV (*Invalid*) или у VS (*Valid-Safe*) стању.

ESIC протокол примјењује агресивнији спекулативни приступ па прослеђује и потенцијално коначне регистарске вриједности. Он има осам стања за промјенљиве регистре: INV (*Invalid*), VU (*Valid-Unsafe*), VS (*Valid-Safe*), VPS (*Valid-Possibly-Safe*), VPSF (*Valid-Possibly-Safe-Forwarded*) и LC (*Last Copy*), док константни регистри могу бити у INVO (*Invalid-Others*) или у VSO (*Valid-Safe-Others*). Оба протокола омогућавају и комуникацију иницирану од корисника (*consumer initiated*) и од произвођача (*producer initiated*). Такође, механизам дистрибуиране арбитрације је предложен код оба протокола за бирање најближе, најскорије претходне нити којој је дозвољено да пошаље валидну вриједност регистра на магистралу при задовољавању захтјева за подацима од стране

спекулативних нити. Поред тога, оба протокола користе „*read-snarfing*“ технику како би смањили број инвалидационих промашаја при операцији читања код константних регистара.

Комплетна, детаљна спецификација оба предложена протокола за регистарску комуникацију је дата у наредној, шестој глави.

5.4 Подршка за спекулацију на нивоу меморије

У поређењу са регистарским зависностима по подацима, веома је тешко пронаћи меморијске зависности по подацима анализирајући извршни бинарни код. Зато је одлучено да се у овом предлогу обезбиједи хардверска подршка која је у потпуности одговорна за проналажење и обраду меморијских зависности по подацима, као и опоравак од погрешне спекулације приликом извршавања. Она гарантује да се уписи и читања од различитих језгара одвијају у програмском поретку. Овај предлог за спекулативно извршавање поштује стриктну секвенцијалну семантику приликом завршетка нити и приликом иницирања нове нити.

Меморијска комуникација, чување спекулативних података и детекција зависности по подацима у предложеном CMP систему се омогућава помоћу дистрибуираних L1 кеш меморија података са одложеним уписом и са додатом подршком за спекулативно извршавање. L1 кеш меморије су, такође, повезане заједничком магистралом. За ову намјену функционалност неопходна за контролу спекулативног извршавања је имплементирана у оквиру кеш контолера приватних L1 кеш меморија. Зато и заједничка магистрала, поред стандардних линија података, адреса и контроле, садржи и посебне линије за подршку интегрисаној контроли кохеренције и спекулације. За потребе арбитације при задовољавању захтјева за подацима од стране спекулативних нити предложен је механизам дистрибуиране арбитрације. Он отклања потребу за централним арбитратором и омогућава ефикасну арбитрацију на основу маски спекулативних нити.

С обзиром на коришћење магистрале, и TLS подршка за меморијску комуникацију се заснива на принципима „*snoopy*“ протокола за кохеренцију кеш меморија. Због тога што обједињује одржавање кеш кохеренцију и контролу спекулације предложени протокол је назван SISC (*Speculation Integrated with Snoopy Coherence*) протокол. Протокол користи скуп стања који омогућава кеш кохеренцију и спекулацију тако да задржи програмски поредак међу различитим верзијама података. Стање меморије се одржава на нивоу једне ријечи, јер су претходни радови показали да грануларност објекта спекулације на нивоу једне кеш линије може проузроковати изражену појаву лажне зависности по подацима, која захтијева непотребно поништавање нити, што смањује перформансе [GOPA1998, TSAI1999, KRIS1999, CINT2000, ZAHR2003, YANA2003].

Протоколи за очување кохеренције првенствено користе двије стратегије за очување кохеренције: поништавање (*invalidation*) и ажурирање (*update*). Многе

студије су евалуирале ове стратегије, али се ниједна није показала супериорнијом у свим условима, јер њихове перформансе веома зависе од преовлађујућег начина дијељења података у апликацији. Зато су овдје предложене двије варијанте меморијског протокола.

Прва варијанта, SISC-WI протокол, је заснован на стратегији поништавања. Као основа за предложени протокол коришћен је стандарни MSI (*Modified-Shared-Invalidation*) протокол са одложеним уписом код којег један блок у кеш меморији може бити у једном од три стања: I (*Invalid*), S (*Shared*) или M (*Modified*). Ради детекције зависности по подацима и ради чувања спекулативних уписа у меморију овај стандардни скуп стања мора да се прошири, као што је урађено у SVC [GOPA1998], Multiplex [OOI2001], STAMPede [STEF2005] или NEKO [BARL2004].

SISC WI протокола користи 13 стања за праћење статуса података: INV (*Invalid*), S (*Shared*), SSp (*Shared-Speculative*), StM (*Stale-Modified*), StS (*Stale-Shared*), StSp (*Stale-Speculative*), StSpM (*Stale-Speculative-Modified*), StMSp (Stale-*Modified-Speculatively*), M (*Modified*), MSp (*Modified-Speculative*), MSpF (*Modified-Speculative-Forwarded*), MSSp (*Modified-Shared-Speculative*), MSSpF (*Modified-Shared-Speculative-Forwarded*). SISC-WI протокол омогућава само комуникацију међу нитима коју је покренуо потрошач. Коректну вриједност доставља најближа претходна спекулативна нит, неспекулативна нит или дијељена L2 кеш меморија.

За разлику од неких других рјешења, када се неспекулативна нит заврши њени подаци могу да остану у L1 кеш меморији и не морају одмах да се шаљу у L2 кеш меморију након што је нит која је уписала податке завршена. На овај начин се избјегава могуће повећање саобраћаја на магистрали које може повећати вријеме потребно за завршетак нити као и вријеме за иницирање нове нити на истом процесору. Модификована и потврђена ријеч се уписује из L1 кеш меморије у L2 кеш меморију када јој се приступи наредни пут, било од стране локалне или неке друге спекулативне нити. Ријечи у кеш меморији које су у неком од *stale* стања, St или StS, затим у S стању или M стању, остају у истом стању када спекулативна нит постане неспекулативна. Тада, нит може да се заврши и да се иницира нова нит на истом језгру. Детаљан приказ SISC-WI протокола је дат у седмој глави.

Код протокола заснованих на поништавању један од акутних проблема су инвалидациони промашаји који настају као незаобилазна посљедица ове стратегије. Код већег броја дијељених копија података ово може значајно деградирати перформансе. Зато је у ранијим радовима предложена „*read-snarfing*“ техника која омогућава да, када једна кеш меморија задовољава инвалидациони промашај дохватањем података, све остале које имају поништену копију могу да их истовремено ажурурају узимањем доступних података са магистрале [EGGE1989, CULL1999]. С обзиром да то само незнатно повећава сложеност постојећег кеш контролера одлучено је да се предложи и верзија SISC-WI протокола која имплементира акције „*read-snarfing*“ технике. Ова верзија је

названа SISC-WI-RS. Детаљан приказ ове верзије протокола је, такође, дат у седмој глави.

Предложена је и алтернативна варијанта SISC протокола која је заснована на стратегији ажурирања, SISC-WU протокол. Мотивација за развој овог протокола долази из чињенице да стратегија ажурирања више одговара неким апликацијама са мањом дужином уписног низа, као и гранулацији објекта кохеренције на нивоу ријечи. Код ове стратегије, процесор врши упис у локалну кеш меморију и шаље информацију свим осталим процесорима који имају копију податка. Други процесори читају копије података из локалне кеш меморије, без иницирања комуникације од стране нити потрошача избјегавајући скупе инвалидационе промашаје. SISC-WU протокол користи 9 стања: M (*Modified*), S (*Shared*), SU (*Shared-Unsafe*), SUSp (*Shared-Unsafe-Speculative*), SSp (*Shared-Speculative*), MSp (*Modified-Speculative*), MSpF (*Modified-Speculative-Forwarded*), MSSp (*Modified-Shared-Speculative*), MSSpF (*Modified-Shared-Speculative-Forwarded*). Свакој ријечи у L1 кеш меморији се додаје и још један бит назван St (*Stale*) бит. Помоћу овог бита се означава ријеч у кеш меморији која припада некој од претходних нити и коју може користити текућа нит или која се може прослиједити некој наредној нити на захтјев, али се та ријеч не може користити у нитима које ће бити иницирane у будућности на истом процесорском језгру.

Међунитна меморијска комуникација у овој варијанти протокола може бити иницирана и од стране потрошача и од стране произвођача. Када се дијељена реч у SISC-WU протоколу упише у одговарајућу L1 кеш меморију, њена вриједност се ажурира у L1 кеш меморијама свих следбеника који такође имају ту ријеч у истом дијељеном стању. Када касније спекулативне нити захтијевају ту ријеч, она може бити дохваћена из њихових локалних L1 кеш меморија са малом латенцијом. Поред тога, L1 кеш меморије наредних нити ће бити ажуриране само једном трансакцијом на магистрали чиме се смањује саобраћај на магистрали. Комплетан приказ SISC-WU протокола за меморијску комуникацију је дат у осмој глави.

У све три варијанте протокола кохеренције и спекулације се користи прилагођени алгоритам замјене (*replacement policy*). Приликом избора ријечи за замјену алгоритам узима у обзир стање ријечи као примарни критеријум умјесто уобичајеног критеријума историје приступа (слично као у [TOMA2005a]). Први кандидати за замјену су ријечи у неспекулативним стањима, јер би замјена спекулативне ријечи проузроковала поништавање и поновно покретање спекулативне нити. Алгоритам дозвољава спекулативној нити да избаци из L1 кеш меморије модификовану потврђену ријеч која је једина копија и која је била спекулативно прочитана или измијењена. Такође, ријечи које нијесу биле спекулативно прочитане или измијењене могу бити кандидати за замјену. Тиме се избјегава заустављање процесора и спекулативно извршавање се наставља. Међутим, ако се више ријечи налази у истом стању, за избор између њих користи се као секундарни критеријум стандардни начин који је заснован на временској локалности (као што је FIFO или LRU) или алгоритам случајног избора.

Глава 6 Предлог протокола за регистарску комуникацију

Као што је најављено у претходној глави, предложене су двије варијанте подршке за регистарску комуникацију између нити у спекулативном CMP систему: а) основна верзија - SIC протокол [RADU2005]; и б) напредна верзија – ESIC протокол [RADU2005a]. У овој глави ће детаљно бити описана софтверска подршка, хардверска инфраструктура, као и акције оба протокола, а на крају ће бити дато њихово квалитативно поређење [RADU2014].

6.1 SIC протокол

SIC[1] (*Snoopy Inter-register Communication*) протокол је заснован на конзервативном приступу јер прослеђује само сигурне вриједности регистара наредним спекулативним нитима.

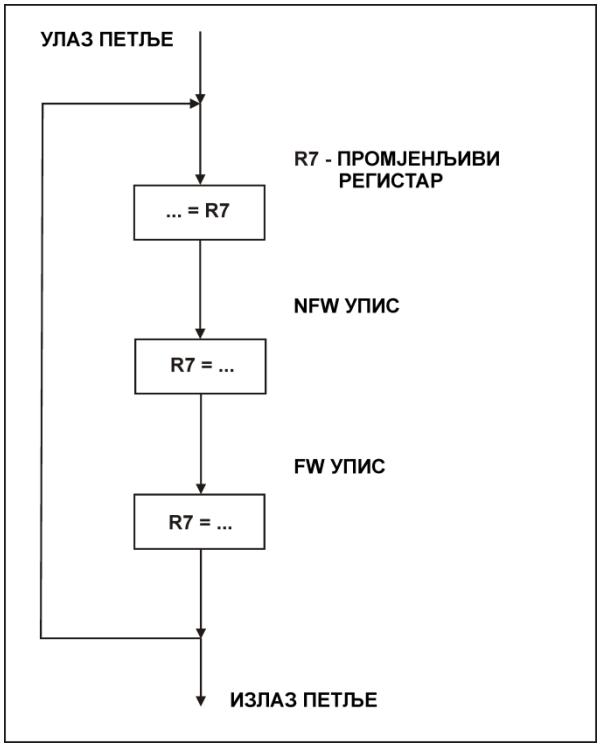
6.1.1 Софтверска подршка

Метод спекулације и приступ је сличан као код I-ACOMA процесора. Софтверска подршка у виду бинарног анотатора ради на извршном, бинарном коду секвенцијалне апликације. Тако се избјегава потреба за изворним кодом и за његовим поновним превођењем што је значајна предност. Као и у I-ACOMA, бинарни анотатор идентификује нити, проналази регистарске инструкције за упис

1 "*sic"* (латински) – тако, овако, на овај начин

које генеришу тачне вриједности за прослеђивање наредним нитима и у складу са тим анотира бинарни код.

Идентификација спекулативних нити је релативно једноставна с обзиром да су оне ограничене на итерације петљи и, у складу са тим, програмски код нити које се извршавају паралелно је исти. Једино улазне и излазне тачке за сваку петљу морају бити означене. Анализа уписа у регистре је много захтјевнија. Прво, анотатор препознаје регистре чије се вриједности уносе у петљу и износе из ње и у које може бити уписивано у оквиру петље – промјенљиви (*loop-live*) регистри. Регистри из којих се може само читати у оквиру петље се називају константни (*non loop-live*) регистри. На крају, инструкције уписа у промјенљиве регистре морају бити означене. Ови уписи могу бити класификовани као текући (NFW - *non-final write*) или коначни (FW – *final write*). Текући уписи генеришу вриједности регистара које могу бити преписане у току итерације петље, док коначни уписи генеришу вриједности регистара које не могу бити промијењене у текућој итерацији петље. На слици 6-1 је дат примјер за ове двије врсте уписа.

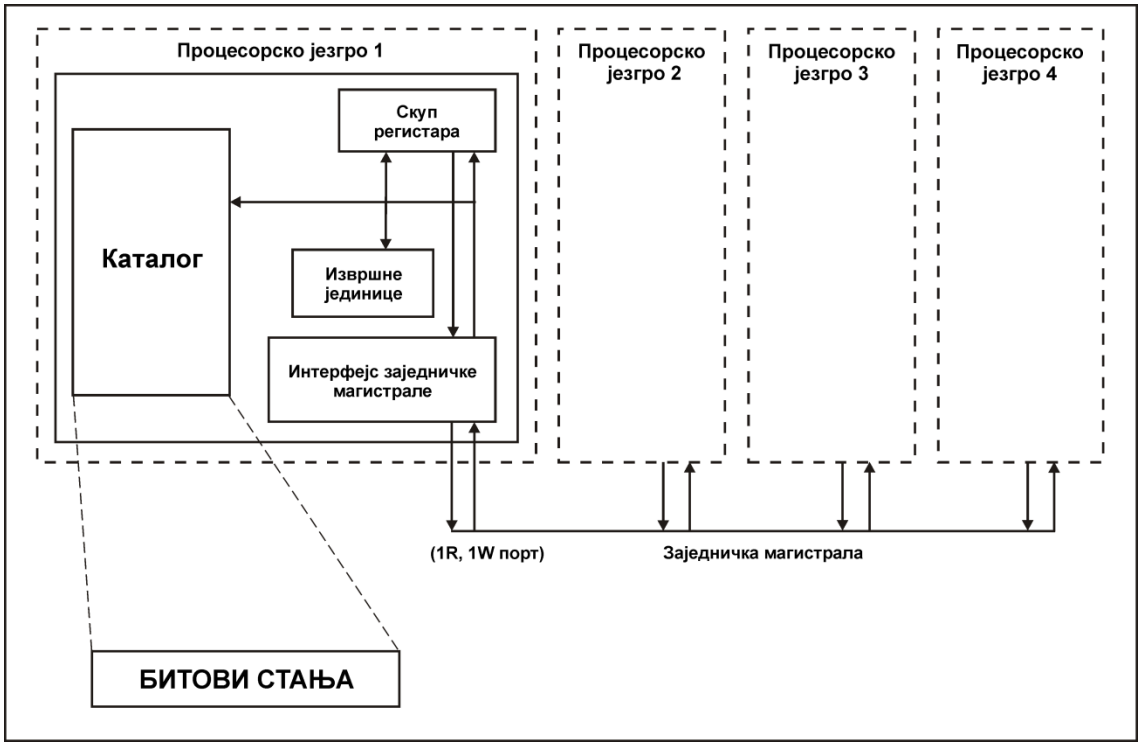


Слика 6-1. Примјер класификације уписа у промјенљиве регистре у SIC протоколу

6.1.2 Хардверска подршка

Посебна дијељена магистрала се користи за пренос вриједности регистара између језгара. Хардверска инфраструктура у сваком процесорском језгру за подршку SIC протокола се састоји од интерфејса дијељене магистрале, логике за

чување и промјену спекулативног статуса нити, контролне логике за остваривање прелаза између стања протокола и локалног каталога за чување стања сваког регистра (слика 6-2). Улаз у локалном каталогу за сваки регистар је величине само неколико бита и садржи информације о валидности, доступности за прослеђивање, итд. Спекулативни статус за нит која се тренутно извршава на процесорском језгру се чува у посебном регистру. С обзиром да се четири нити могу извршавати паралелно, статус нити се представља са два бита.



Слика 6-2. Хардверска подршка за регистарску комуникацију

Регистарски каталог има улаз за сваки регистар за чување његовог тренутног стања. Промјенљиви регистри могу бити у једном од четири стања: INV (*Invalid*), VU (*Valid-Unsafe*), VS (*Valid-Safe*) и LC (*Last Copy*). Константни регистри могу бити или у INV (*Invalid*) или у VS (*Valid-Safe*) стању. Због тога се у локалном каталогу налазе по три бита за сваки регистар, један бит за разликовање промјенљивих и константних регистара и два бита за кодирање највише четири стања.

Стање INV указује да вриједност регистра није валидна и да не може бити коришћена у локалној нити или прослеђена другим нитима. Стање VU показује да је вриједност регистра валидна за текућу нит, али није коначна вриједност коју генерише ова нит и не може бити прослеђена наредним нитима. Стање VS указује да је вриједност регистра валидна за текућу нит, да је она коначна и да се може

просљедити наредним нитима. Стање LC указује да је то једина копија валидне вриједности регистра.

Комуникација на нивоу регистара се одвија преко посебне дијељене магистрале. Она обухвата адресне линије регистара, линије података за вриједности регистара, као и неке контролне линије. Контролни дио укључује *BusR, BusW, Mask* и *Shared* сигнале.

Сигнал *BusR* означава трансакцију читања на магистрали коју генерише захтјев за читање од стране процесора за регистар који се налази у стању INV. Захтјев за читање се задаје на магистрали заједно са маском нити која се извршава на датом процесору.

Сигнал *BusW* означава трансакцију уписа на магистрали коју генерише захтјев за упис процесора када спекулативна нит достигне стање VS за дати регистар или када је вриједност регистра у стању LC послата преко магистрале. Вриједност регистра је послата преко магистрале тако да нит којој је потребна може да је учита. У оба случаја, произвођач иницира комуникацију јер се, у овом случају, вриједност регистра не преноси на захтјев корисника.

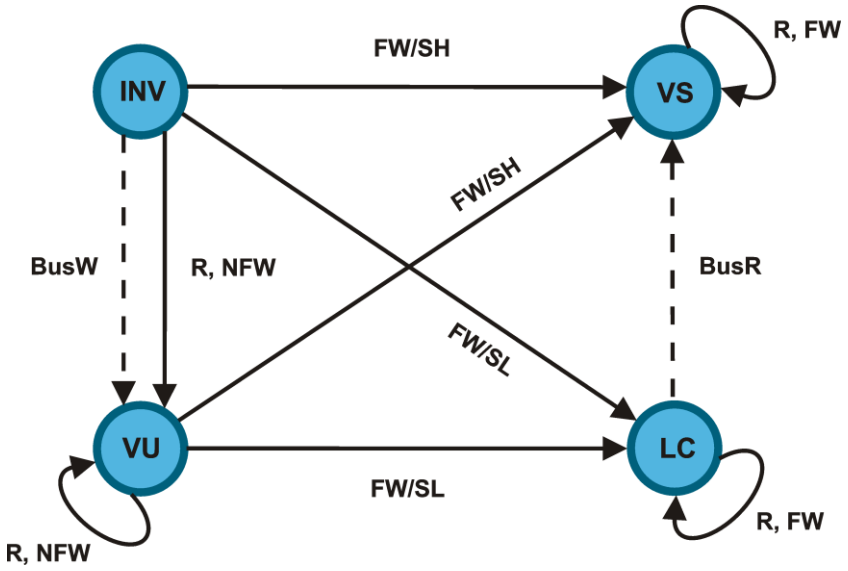
Сигнали *Mask* се постављају на магистралу током трансакције читања на магистрали, када нит тражи вриједност регистра од раније нити. Свака претходна нит са траженом вриједношћу регистра у стању VS одговара тако што просљеђује своју маску на одговарајуће линије магистрале, које су имплементиране као „ожичено-ИЛИ”. Логика интерфејса магистрале у сваком процесорском језгру може одредити да ли је неки друга ранија нит постављала своју маску на магистралу. Механизам дистрибуиране арбитрације бира најближу, најскорију претходну нит којој је дозвољено да пошаље валидну вриједност регистра на магистралу.

Сигнал *Shared*, који се такође шаље преко „ожичено-ИЛИ” линије магистрале, поставља нит потрошач одговарајући на трансакцију коју иницира произвођач када шаље вриједности регистра у стању VS или LC. Када је ова линија сетована од стране нити потрошача, произвођач је обавијештен да је вриједност регистра која је послата на магистралу такође учитана од стране неког другог процесора и да она није посљедња и једина копија.

6.1.3 Опис акција SIC протокола

Да би приступио регистру, процесор може издати два типа захтјева: читање (R - *read*) или упис. Уписи могу бити коначни (FW) или текући (NFW) како их означава бинарни анотатор. SIC протокол укључује регистарску комуникацију иницирану и од стране произвођача и од стране потрошача. Овај протокол је базиран на претходно дефинисаним стањима и трансакцијама међу њима. Протокол различито управља приступом константним и промјенљивим регистрима. Прелази између стања које изазивају инструкције процесора

(*processor-induced*) и прелази стања које изазивају трансакције на магистрали (*bus-induced*) за промјењиве регистре су представљени на слици 6-3.



Слика 6-3. Трансакције стања које изазива процесор (пуне линије) и трансакције стања које изазива магистрала (испрекидане линије) за промјењиве регистре у SIC протоколу. Читања процесора су означена са R. Нотација A/B значи да су процесорски уписи (FW или NFW) посматрани у релацији са сигналом *Shared* који је или активан (SH) или неактиван (SL).

Механизам протокола за промјењиве регистре ради на сљедећи начин:

(1) **Погодак при читању.** Ако спекулативна или неспекулативна нит изда захтјев за читањем регистра у валидном стању (VU, VS или LC) захтјев је задовољен локално и регистар остаје у истом стању.

(2) **Промашај при читању.** Захтјев за читањем регистра у стању INV изазива промашај при читању и иницира трансакцију *BusR*. Захтјев за читањем овог регистра се шаље на магистралу заједно са кодом маске спекулативне нити која је издала захтјев. Посљедица тога је да промашај читања генерише комуникацију међу нитима коју иницира потрошач. Сви могући достављачи односно раније нити (неспекулативна нит и/или раније спекулативне нити) које имају тражени регистар или у стању VS или у LC, одговарају постављајући своје кодове маске на магистралу и дистрибуирани арбитар бира најближу претходну нит. Након тога изабрана претходна нит обезбјеђује тражену вриједност регистра на магистрали и она се у стању VU учитава од спекулативне нити која је издала захтјев за читањем. Ако је захтјевани регистар био у стању LC на страни достављача, његово стање се мијења из стања LC у VS. Ако нема доступног достављача када

нит потрошач изда захтјев за читањем на магистралу, нит потрошач се блокира и чека док произвођач не генерише и пошаље тражену вриједност.

(3) **Погодак при упису.** Ако је вриједност регистра у стању VU када нит, било неспекулативна или спекулативна, обавља NFW упис, регистар се ажурира локално и остаје у стању VU. Међутим, ако је вриједност регистра у стању VU када нит обавља FW упис, регистар се ажурира и стање се мијења из стања VU у VS. Такође, FW упис изазива комуникацију иницирану од стране произвођача шаљући вриједност регистра на магистралу за најближу каснију нит. Ако је вриједност учитана у регистар најближе касније нити, стање регистра се мијења из стања INV у VU и каснија нит поставља линију *Shared* обавјештавајући произвођача да је вриједност учитана. Такође, каснија нит која је била блокирана чекајући баш ову вриједност регистра, наставља извршавање. Ако је линија *Shared* остала неактивна, произвођач мијења стање овог регистра из стања VS у LC, означавајући да је то једина валидна копија.

(4) **Промашај при упису**. NFW или FW упис у регистар у стању INV изазива промашај при упису. Акције протокола за обје врсте уписа су исте као у претходно описаним случајевима поготка при упису. Крајња стања при прелазу стања су, такође, иста, док је почетно стање INV (умјесто стања VU у случају поготка при упису).

С обзиром да константни регистри могу бити само читани од стране нити, механизам протокола за њих ради на сљедећи начин:

(1) **Погодак при читању.** Ако је вриједност регистра у стању VS када нит, било неспекулативна или спекулативна, изда захтјев за читањем, он се задовољава локално и регистар остаје у стању VS.

(2) **Промашај при читању.** Овај случај може да се догоди само када се на процесору спекулативна нит извршава по први пут након неког секвенцијалног дијела апликације. Захтјев за читањем регистра у стању INV иницира трансакцију *BusR*. Затим, неспекулативна нит или нека од спекулативних нити обезбјеђује захтијевану вриједност регистра која се учитава у стању VS од нити која је издала захтјев. Истовремено друге, више спекулативне нити које такође имају исти регистар у стању INV ослушкују магистралу и учитавају послату вриједност у стању VS у складу са „*read-snarfing*” техником [TOMA1993].

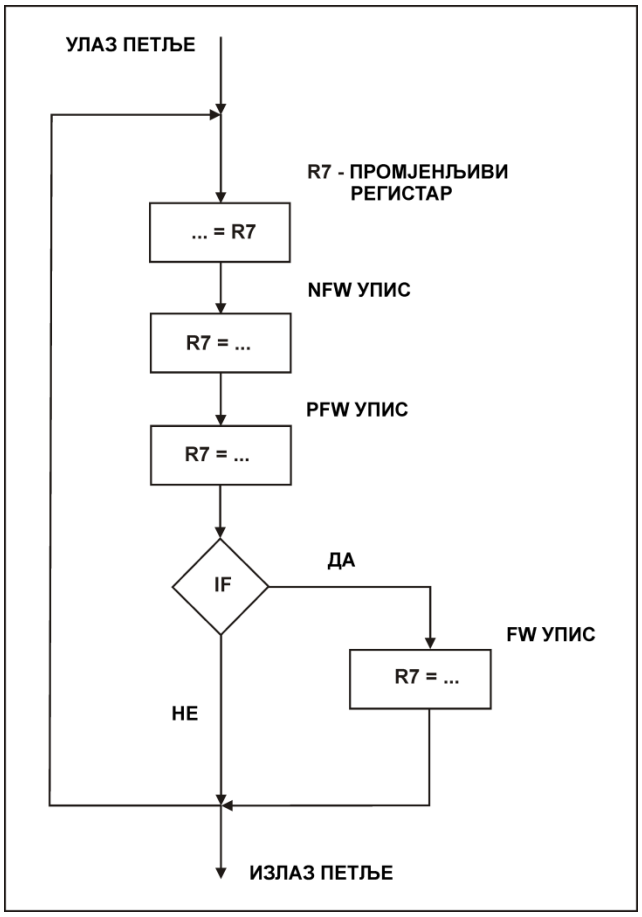
6.1.4 Иницијализација и завршетак нити

Када процесор иницира нову спекулативну нит (најспекулативнија нит) неопходно је извршити инвалидацију свих промјенљивих регистара постављајући њихово стање на INV, док константни регистри остају у свом текућем стању. Када је нит на крају свог извршавања, тада се врши претрага за регистрима у локалном каталогу који су посљедње копије (стање LC). Нити је дозвољено да заврши своје

извршавање само ако нема регистарских вриједности у стању LC у локалном каталогу. У супротном, свака вриједност регистра у стању LC мора бити сачувана тако што их нит пошаље преко магистрале наредним нитима, као у случају обраде FW уписа у комуникацији иницираној од стране произвођача.

6.2 ESIC протокол

У тежњи да се побољшају перформансе SIC протокола, блокирање нити потрошача док чека на вриједност регистра произвођача је препознато као главна препрека. SIC протокол је спекулативно конзервативан јер су током регистарске комуникације инициране и од стране произвођача и од стране потрошача размјењују само сигурне вриједности. Међутим, постоје случајеви (посебно када се појави условно извршавање) гдје коначне вриједности не могу бити недвосмислено идентификоване од стране бинарног анотатора. Упис у регистар, који претходи *If*-исказу, се при извршавању може испоставити као коначан (FW) ако услов није испуњен, али и као текући (NFW) ако је услов испуњен, као што је показано на слици 6-4.



Слика 6-4. Примјери класификације уписа у ESIC анотатору

Зато SIC анотатор овај упис конзервативно класификује као текући. Ако услов током извршавања постане испуњен, то може проузроковати непотребно блокирање неке наредне спекулативне нити која захтијева овај регистар. Због тога је у циљу потенцијалног побољшања перформанси у оваквим ситуацијама предложен ESIC (*Enhanced SIC*) протокол. За разлику од SIC-а, он је заснован на спекулативном прослеђивању потенцијално сигурних вриједности регистара. У сљедећим секцијама је представљена надградња ESIC-а у односу на SIC протокол.

6.2.1 Софтверска подршка

ESIC протокол захтијева неке модификације бинарног анотатора. Класификација уписа је употпуњена додавањем препознавања потенцијално коначног уписа (PFW – *Possibly Final Write*) (слика 6-4). Као и раније, текући упис је праћен бар још једним уписом у исти регистар, док је коначан упис посљедњи извршени упис у нити. Потенцијално коначан упис може у извршавању бити или текући или коначан у зависности од тока контроле. У случају када нит потрошач захтијева вриједност регистра која још увек није коначно генерисана, потенцијално коначна вриједност регистра може бити прослеђена нити потрошачу да би се задовољио захтјев и избјегло њено блокирање. Касније, ако нит произвођач изврши коначан упис за раније прослеђену потенцијално коначну вриједност регистра, она шаље сигнал *Squash* потрошачкој нити, која ју је већ искористила, да би поништила ефекте погрешне спекулације.

Добитак на перформансама овако агресивне спекулације директно зависи од вјероватноће да се упис означен са PFW од стране анотатора испостави као коначан током спекулативног извршавања и да спекулација буде успјешна. Иначе, погрешна спекулација изазива одређене трошкове. Ова вјероватноћа се може процијенити профајлирањем апликације. Због тога, у циљу постизања најбољих перформанси препроцесирање апликације, такође, обухвата и сљедеће кораке:

- Профајлирањем апликације за сваки упис означен као PWF од стране анотатора, статистика сакупља колико се пута (*fcnt – final counter*) овај упис појавио као коначан у свих *n* итерација за вријеме извршавања.
- Ако је вјероватноћа успјеха спекулације (*fcnt/n*) изнад предвиђеног прага за поједини PFW упис, спекулација може бити корисна и овај упис остаје означен као PFW. У супротном, овај упис се означава као NFW, с обзиром да превише агресивна спекулација може бити чак и контрапродуктивна.

Конкретна вриједност прага зависи од система и апликације, а може се одредити симулацијама.

6.2.2 Хардверска подршка

Промјенљиви регистри у ESIC протоколу се могу наћи у једном од сљедећих стања: INV (*Invalid*), VU (*Valid-Unsafe*), VS (*Valid-Safe*), VPS (*Valid-Possibly-Safe*), VPSF (*Valid-Possibly-Safe-Forwarded*) и LC (*Last Copy*). Константни регистри могу бити или у INVO (*Invalid-Others*) или у VSO (*Valid-Safe-Others*) стању.

Стања INV, VU, VS, LC, INVO и VSO су наслијеђена из SIC протокола, док су стања VPS и VPSF уведена само у ESIC протоколу. Стање VPS означава да је вриједност регистра валидна за текућу нит и као потенцијално коначна вриједност може бити спекулативно прослеђена наредним нитима на њихов захтјев. Стање VPSF указује да је дата вредност регистра валидна за текућу нит и да је она већ прослеђена наредним нитима на њихов захтјев. Ово стање омогућава да се прате могуће погрешне спекулације и детектују нарушавања.

Као додатак сигналима *BusR, BusW, Mask* и *Shared*, контролни дио дијељене магистрале такође укључује сигнал *Squash* (*Sq*). Овај сигнал се издаје на магистрали као посљедица уписа процесора у регистар у VPSF стању. Потрошачка нит која је превремено учитала дати регистар се поништава када се детектује сигнал *Squash* за овај регистар на магистрали.

6.2.3 Опис акција ESIC протокола

Регистарска комуникација између нити у ESIC протоколу може бити иницирана од стране произвођача или од стране потрошача, као и у SIC протоколу. Прелази стања иницирани од стране процесора или магистрале за промјенљиве регистре у ESIC протоколу су представљени на слици 6-5.

Протокол за промјенљиве регистре ради на сљедећи начин:

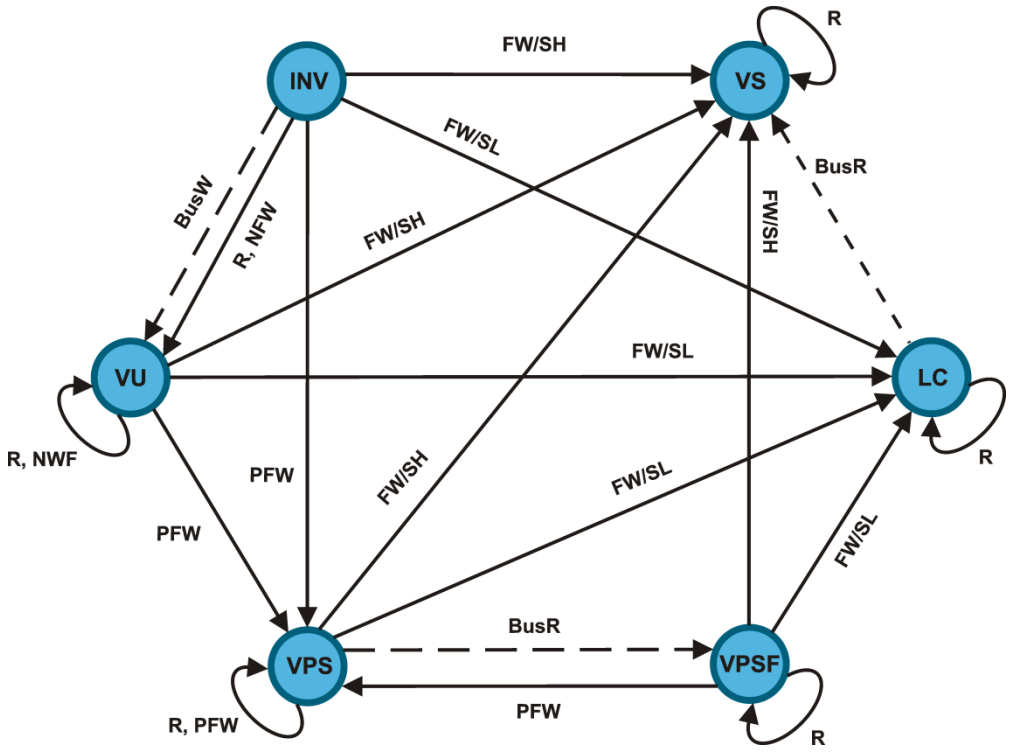
(1) **Погодак при читању.** Као у SIC протоколу, захтјев за читањем регистра у валидном стању (овде су укључена стања и VPS и VPSF) се задовољава локално без промјене стања.

(2) **Промашај при читању.** Захтјев за читањем регистра у стању INV иницира трансакцију *BusR*, као и код SIC протокола. Процедура је иста и једина разлика је што потенијални достављач може, такође, бити нека претходна нит са траженим регистром у стању VPS. Ако је тражени регистар био у стању VPS на страни достављача, он прелази из стања VPS у VPSF стање да би се сачувала информација о спекулативном прослеђивању. Тражени регистар се учитава у стању VU на страни потрошача.

У SIC протоколу нит потрошач се блокира, када изда захтјев за читањем на магистрали, ако нема достављача који ће јој прослеђити коначну вриједност. Међутим, у истој ситуацији ESIC протокол избјегава блокирање ако постоји достављач потенцијално коначне вриједности. Нит потрошач у ESIC протоколу се

блокира само у случају када нема достављача или коначне или потенцијално коначне вриједности.

(3) **Погодак при упису.** Реакција на NFW упис у стању VU и на FW упис у стању VU или VPS су исте као у SIC протоколу. Међутим, ако FW упис затекне вриједност регистра у стању VPSF, нит прво шаље сигнал *Squash* на магистралу наредним нитима које су раније учитале дату регистарску вредност, да би поништила ефекте погрешне спекулације. Након тога, понавља се иста процедура као у случају FW уписа у стању VPS да би се наредним нитима обезбиједила коректна вриједност.



Слика 6-5. Прелази између стања иницирани од стране процесора или магистрале за промјењиве регистре у ESIC протоколу. Нотација *A/B* значи да је при упису процесора (FW, PFW и NFW) детектовано активно (SH) или неактивно (SL) стање сигнала *Shared*. R означава читања процесора. Испрекиданом линијом су приказани прелази које изазива магистрала, док су прелази које изазива процесор приказани пуном линијом.

У случају PFW уписа у стању VU, регистар се ажурира локално и његово стање се мијења из стања VU у стање VPS. Ако PFW упис затекне вриједност регистра у стању VPS, регистар се ажурира локално и остаје у стању VPS. На крају, када PFW затекне вриједност регистра у стању VPSF, нит издаје сигнал *Squash* на магистрали за наредне нити које су раније учитале дату вриједност. У том случају се регистар ажурира локално и прелази из стања VPSF у стањеVPS.

(4) **Промашај при упису.** NFW упис у стању INV само ажурира вриједност регистра избјегавајући трансакцију *BusW*. Стање регистра се мијења из стања INV у стање VU. Иста реакција се јавља у случају PFW захтјева за вриједношћу у стању INV, али је стање у које регистар прелази VPS. FW упис у стању INV ажурира вриједност регистра и мијења његово стање у VS. Такође, трансакцијом *BusW* се обавља и комуникација између нити иницирана од стране произвођача. Нит произвођач поставља вриједност регистра на магистралу и, ако нека наредна нит учита послату вриједност, она поставља сигнал *Shared* и стање регистра у свом локалном каталогу на VU. Крајње стање у нити произвођачу је VS или LC, у зависности од детектованог сигнала *Shared*.

Протокол за константне регистре функционише на исти начин као у SIC протоколу.

6.2.4 Иницијализација, завршетак и поништавање нити

Када процесор иницира нову спекулативну нит, неопходно је поништити вриједности свих промјенљивих регистара у стањима LC, VU, VPS или VPSF у локалном каталогу постављајући њихова стања на INV. Међутим, константни регистри остају у својим стањима током иницијализације нове нити.

Нити је дозвољено да заврши своје извршавање само у случају да нема регистара у стању VPS или LC у њеном локалном каталогу. У супротном, свака вриједност регистра у стању VPS или LC се мора сачувати тако што се шаље наредним нитима преко магистрале, на исти начин као код коначног уписа у комуникацији иницираној од стране произвођача.

Када нит произвођач уписује вриједност регистра у стању VPSF она врши постављање сигнала *Squash* на магистралу у циљу поништавања нити потрошача која је већ користила дату вриједност регистра. За вријеме поништавања нити потрошача константни регистри у локалном каталогу остају у текућим стањима, док се промјенљиви регистри, изузев оних у стању LC, поништавају.

6.3 Квалитативно поређење SIC и ESIC протокола

Поређење SIC и ESIC протокола је урађено са аспекта комплексности и перформанси. Софтверска подршка за ESIC протокол захтијева мање модификације бинарног анотатора за идентификацију потенцијално коначног уписа. Ово не представља проблем с обзиром да је праћење низова уписа у регистре једна од основних активности бинарног анотатора. ESIC протокол не захтијева више битова по улазу у локалном каталогу у поређењу са SIC протоколом. SIC користи 3 бита (2 бита за кодирање 4 стања и 1 бит за означавање промјенљивих или константних регистара). ESIC такође користи 3 бита за кодирање 6 стања за промјенљиве и 2 стања за константне регистре. Што

се тиче структуре магистрале, потребан је само један додатни контролни сигнал (*Squash* сигнал). Дијаграм стања у ESIC контролеру је сложенији него у SIC контролеру, али искуство из сличних дизајна говори да додатна логика није значајна.

Иако се крајњи закључак о перформансама може извући једино евалуацијом са реалним апликацијама, постоји потенцијални добитак на перформансама у ESIC у односу на SIC. Ово је посљедица смањеног блокирања нити потрошача током чекања на коректне крајње вриједности од нити произвођача. С друге стране, поништавање нити може имати негативан ефекат на перформансе система. Да би се спријечила деградација перформанси, користи се информација из профајлера у циљу примјене спекулације само када постоји вјероватноћа њеног успјешног извршавања.

Прецизно поређење са неким другим механизмима регистарске комуникације је тешко због различитих система на којима је примијењена и начина реализације. Ипак се може направити кратко квалитативно поређење са I-ACOMA [KRIS1999] као постојећег рјешења истог проблема. Док је софтверска подршка практично иста, што се тиче хардверске подршке за регистарску комуникацију, SIC и ESIC протоколи значајно смањују величину локалног каталога као и додатну логику за провјеру доступности регистра и статуса посљедње копије. Поред тога, у поређењу са I-ACOMA, механизам дистрибуираног одлучивања омогућава да се много брже и несеквенцијално пронађе најближа претходна нит. На крају, у SIC и ESIC протоколима коришћење „*read-snarfing*" технике смањује број инвалидационих промашаја у константним регистрима.

Такође се примјећује да су SIC и ESIC протоколи много скалабилнији када је у питању комплексност. Док се величина локалног каталога и хардверска логика код I-ACOMA протокола повећава са повећањем броја процесорских језгара, у овим протоколима они остају скоро константни.

Глава 7 Предлог SISC-WI протокола за меморијску комуникацију

У петој глави је начелно представљен предлог подршке за меморијску комуникацију који обухвата два протокола. У овој глави је детаљно представљен први од њих - SISC-WI, који је заснован на стратегији поништавања, као и његова верзија која имплементира „*read-snarfing*“ технику. У наставку је детаљно представљена основна инфраструктура кроз опис стања, трансакција и сигнала на магистрали, а затим су прецизно описане акције и прелази између стања у овом протоколу.

7.1 Инфраструктура SISC-WI протокола

У овој секцији је представљена инфраструктура SISC-WI протокола која обухвата стања у којима се може наћи ријеч у кеш меморији, као и контролне сигнале и трансакције на магистрали током спекулативног извршавања. Осим тога ће бити представљен и механизам дистрибуиране арбитрације на магистрали који се активира у случају када је потребно одговорити на захтјев за податком који је настао усљед промашаја у кеш меморији.

7.1.1 Стања

За вријеме спекулативног извршавања, ријеч у L1 кеш меморији сваког процесорског језгра у предложеном CMP систему може да се нађе у неком од сљедећих 13 стања:

- INV (*Invalid*),
- S (*Shared*),
- SSp (*Shared-Speculative*),
- StS (*Stale-Shared*),
- StM (*Stale-Modified*),
- StSp (*Stale-Speculative*),
- StSpM (*Stale-Speculative-Modified*),
- StMSp (*Stale-Modified-Speculatively*),
- M (*Modified*),
- MSp (*Modified-Speculative*),
- MSpF (*Modified-Speculative-Forwarded*),
- MSSp (*Modified-Shared-Speculative*),
- MSSpF (*Modified-Shared-Speculative-Forwarded*).

Стање INV указује да ријеч у L1 кеш меморији не постоји или нема валидан садржај тако да не може бити коришћена од стране неспекулативне нити ни од стране спекулативне нити.

Стање S указује да је ријеч у L1 кеш меморији немодификована у односу на дијељену L2 кеш меморију. Друге L1 кеш меморије могу, такође, имати дијељену копију исте ријечи.

Стање SSp означава да је нека спекулативна нит прочитала ријеч из кеш меморије.

Стање StS за ријеч у L1 кеш меморији процесора који извршава неспекулативну или спекулативну нит означава да је каснија спекулативна нит извршила упис у исту меморијску адресу. Ријеч која је била у стању S у L1 кеш меморији раније неспекулативне или спекулативне нити постаје застарјела (*stale*) и прелази у стање StS. Ако је ријеч у L1 кеш меморији раније спекулативне нити, била у стању M када каснија спекулативна нит изврши упис у одговарајући улаз у кеш меморији, ријеч мора да се упише у L2 кеш меморију и мијења јој се стање у StS. Ријеч у стању StS може да користи текућа неспекулативна или спекулативна нит и може да се прослиједи каснијој спекулативној нити на њен захтјев, али нити које ће бити инициране на истом процесору у будућности не могу је више користити.

Стање StM за ријеч у L1 кеш меморији процесора који извршава неспекулативну нит означава да је неспекулативна нит извршила упис у ријеч док је она била у стању StS, што је проузроковало прелаз у стање StM. Ријеч у стању StM може користити текућа неспекулативна нит или може бити прослијеђена каснијој спекулативној нити на њен захтјев, али нити које ће бити покренуте на истом процесору у будућности не могу је користити.

Стање StSp за ријеч у L1 кеш меморији процесора који извршава спекулативну нит означава или да је текућа спекулативна нит спекулативно прочитала ријеч док је била у стању St или је, за вријеме извршавања текуће спекулативне нити, ријеч била у стању SSp, када је каснија спекулативна нит уписала на исту меморијску

адресу. Такође, ријеч у кеш меморији која је у стању INV ће прећи у стање StSp приликом читања или уписа, када постоји каснија спекулативна нит која је уписала у исту меморијску адресу. Ријеч у стању StSp може користити текућа спекулативна нит или може бити прослијеђена каснијој спекулативној нити на њен захтјев.

Стање StSpM за ријеч у L1 кеш меморији процесора који извршава спекулативну нит означава да је ријеч била у стању StSp када је текућа нит извршила упис на исту меморијску адресу. Ријеч у стању StSpM може користити текућа спекулативна нит или може бити прослијеђена каснијој спекулативној нити на њен захтјев. Такође, ако је ријеч у некој ранијој спекулативној нити била у стању MSSpF када је каснија нит извршила упис у одговарајући улаз у својој кеш меморији, без обзира да ли је био погодак или промашај, ријеч у претходној нити прелази у стање StSpM да би се сачувала информација да је ријеч претходно спекулативно модификована.

Стање StMSp за ријеч у L1 кеш меморији процесора који извршава спекулативну нит означава да је ријеч била у стању StS када је текућа нит извршила упис на исту меморијску адресу. Ријеч у стању StMSp може користити текућа спекулативна нит или може бити прослијеђена каснијој спекулативној нити на њен захтјев. Такође, ако је ријеч у некој ранијој спекулативној нити била у стању MSpF када је каснија нит извршила упис у одговарајући улаз у својој кеш меморији, без обзира да ли је био погодак или промашај, ријеч у ранијој нити прелази у стање StMSp да би се сачувала информација да је ријеч претходно спекулативно модификована.

Стање M означава да је неспекулативна нит модификовала ријеч у кеш меморији и сада се та ријеч разликује од копије у L2 кеш меморији. Ријеч у стању M може да се прослиједи каснијој спекулативној нити на њен захтјев. Ријеч у стању M може да се чува у L1 кеш меморији за вријеме спекулативног извршавања, чиме се избјегавају многобројни одложени уписи на дијељеној магистрали за вријеме комитовања нити. У супротном, када се неспекулативна нит завршава све модификоване ријечи из одговарајуће L1 кеш меморије се уписују у дијељену L2 кеш меморију. Пошто ријеч у стању M може да се чува као једина копија у L1 кеш меморији за вријеме спекулативног извршавања, ријеч се може наћи у L1 кеш меморији спекулативне нити која ће се иницирати у будућности на истом процесору. Тада, ријеч у стању M може да се прослиједи било којој спекулативној нити на њен захтјев. Ријеч ће бити уписана у дијељену L2 кеш меморију сљедећи пут када локална нит постави захтјев за упис или када удаљена нит постави захтјев за упис или читање за дату ријеч.

Стање MSp означава да је ријеч у кеш меморији спекулативно модификована и може да се прослиједи каснијим спекулативним нитима на њихов захтјев.

Стање MSpF означава да је спекулативно модификована ријеч у кеш меморији прослијеђена некој каснијој спекулативној нити на њен захтјев. Сврха уведеног

стања је да се омогући памћење информације код произвођача да је спекулативно модификована ријеч у кеш меморији прослијеђена каснијој нити на њен захтјев.

Стање MSSp означава да је ријеч у кеш меморији спекулативно модификована, а при томе је претходно била у стању SSp када је спекулативна нит извршила упис у дату ријеч.

Стање MSSpF означава да је спекулативно модификована ријеч у кеш меморији у стању MSSp прослијеђена једној или више каснијим спекулативним нитима на њихов захтјев. Сврха уведеног стања је да се памти информација о спекулативно модификованој ријечи у кеш меморији произвођача, који је претходно прочитао ту ријеч, да је та ријеч прослијеђена каснијој спекулативној нити на њен захтјев.

Четири бита у локалној L1 кеш меморији се користе за кодирање ових 13 стања у којима се може наћи једна ријеч у кеш меморији током спекулативног извршавања у предложеном CMP систему.

7.1.2 Контролни сигнали и трансакције на магистрали

Процесор може да постави двије врсте захтјева: читање (*PrRd*) и упис (*PrWr*). Дијељена магистрала која повезује локалне L1 кеш меморије са дијељеном L2 кеш меморијом садржи контролне сигнале *BusRd* (*Bus-Read*), *BusRdX (Bus-Read Exclusive)*, *BusWB (Bus-Write Back)*, *WB (Write-Back)*, као групе контролних сигнала *Mask, Stale* и *Speculative-Modified*.

Сигнал *BusRd* означава трансакцију која је посљедица постављања захтјева за читање од стране процесора (*PrRd*) када је дошло до промашаја у локалној L1 кеш меморији. Кеш контролер поставља захтјев за читање на дијељену магистралу заједно са маском нити која је генерисала захтјев да би добио тражену ријеч.

Сигнал *BusRdX* означава трансакцију која је посљедица постављања захтјева за упис од стране процесора (*PrWr*), без обзира да ли се ради о неспекулативном или спекулативном упису. Када кеш контролер постави сигнал *BusRdX* на дијељену магистралу, свака ранија било неспекулативна или спекулативна нит, ослушкује магистралу и ако има исту ријеч у валидном стању означава је као застарјелу копију. У исто вријеме, свака каснија нит ослушкује магистралу да би детектовала RAW нарушавање зависности, ако је претходно спекулативно прочитала дату ријеч. Та нит, као и све касније нити (ако их има), која је прочитала одговарајућу ријеч биће поништена и поново покренута.

Сигнал *BusWB* означава трансакцију коју генерише кеш контролер као одговор на трансакцију *BusRd* или *BusRdX* коју иницира неко друго процесорско језгро.

Сигнал *WB* означава трансакцију коју иницира само процесорско језгро приликом одложеног уписа ријечи у стању M који је посљедица или замјене или захтјева за читање/упис те ријечи од стране процесора/магистрале. Кеш контролер поставља адресу и садржај ријечи на дијељену магистралу како би се уписала у дијељену L2 кеш меморију.

Сигнали *Mask* се постављају на дијељену магистралу у случају промашаја приликом читања у локалној L1 кеш меморији када нит тражи ријеч од своје најближе раније (претходне) односно касније (наредне) нити. Дијељена магистрала има двије линије за маску које су реализоване као "ожичено ИЛИ" коло. Свака ранија нит, са траженом ријечи у неком од стања S, SSp, StS, StM, StSp, StSpM, StMSp, M, MSp, MSpF, MSSp или MSSpF, одговара постављањем маске на одговарајуће линије магистрале. Такође, свака каснија нит који има тражену ријеч у стању M или S одговара постављањем своје маске на ове линије. Кеш контролер у свакој L1 кеш меморији може да детектује да ли је било која ранија или каснија нит поставила своју маску на магистралу. Вриједности маске ни на који начин нијесу повезане са идентификатором процесора. Оне зависе само од статуса спекулације нити која се тренутно извршава на процесору.

Излазни сигнали из групе контролних сигнала *Stale* на дијељеној магистрали су:

- **pHStWO** (*predecessor Has Stale Word Output*) – Овај сигнал се шаље до касније нити.
  Значење: Ранија нит има тражену ријеч у једном од *Stale* стања и то оглашава свим каснијим нитима да би обавијестила спекулативну нит која је поставила захтјев за читање те ријечи да тражену ријеч постави у стање *StSp* приликом учитавања у своју локалну L1 кеш меморију.
- **sHStWO** (*successor Has Stale Word Output*) – Овај сигнал се шаље до раније нити.
  Значење: Каснија нит има тражену ријеч у једном од *Stale* стања и то оглашава свим ранијим нитима, да би обавијестила оне раније нити које су поставиле захтјев за читање да тражену ријеч поставе у стање StSp приликом учитавања у своју локалну L1 кеш меморију.

Улазни сигнали, који су реализовани као „ожичено ИЛИ" коло, за претходно наведене *Stale* сигнале су:

- **pHStWI** – Овај сигнал је „ожичено ИЛИ" коло свих pHStWO сигнала.
- **sHStWI** – Овај сигнал је „ожичено ИЛИ" коло свих sHStWO сигнала.

Неактивне вриједности свих претходно поменутих улазних и излазних сигнала из групе контролних сигнала *Stale* су означене као: **/pHStWO**, **/sHStWO**, **/pHStWI**, **/sHStWI**.

Излазни сигнали из групе контролних сигнала *Speculative Modified* су:

- **pHMSpWO** (*predecessor Has Modified Speculative Word Output*) – Овај сигнал се шаље до касније нити.
  Значење: Ранија нит има тражену ријеч у једном од спекулативно модификованих стања које није *Stale* стање и то ће огласити свим каснијим спекулативним нитима да би их обавијестила да тражену ријеч поставе у

стање SSp приликом учитавања у локалну L1 кеш меморију уколико су издале захтјев за читање, или да промијене стање тражене ријечи у стање MSp ако су издале захтјев за упис.

- **sHMSpWO** (*successor Has Modified Speculative Word Output*) – Овај сигнал се шаље до раније нити.
  Значење: Каснија нит има захтијевану ријеч у једном од спекулативно модификованих стања које није *Stale* стање и то оглашава свим ранијим нитима да би их обавијестила да тражену ријеч поставе у стање StSp приликом учитавања у локалну L1 кеш меморију ако су издале захтјев за читање.

Улазни сигнали, који су реализовани као „ожичено ИЛИ“ коло, за претходно поменуте S*peculative Modified* сигнале су означени као:

- **pHMSpWI** – Овај сигнал је „ожичено ИЛИ“ коло свих pHMSpWO сигнала.
- **sHMSpWI** – Овај сигнал је „ожичено ИЛИ“ коло свих sHMSpWO сигнала.

Неактивне вриједности свих горе поменутих улазних и излазних сигнала из групе контролних сигнала *Speculative Modified* су означене као: **/pHMSpWO**, **/sHMSpWO**, **/pHMSpWI**, **/sHMSpWI**.

7.1.3 Механизам дистрибуиране арбитрације

У неким случевима на захтјев за податком, који је настао усљед промашаја у кеш меморији, може да одговори више нити могућих достављача (*suppliers*), неспекулативна нит (вриједност маске: 00) или спекулативне нити (вриједности маске: 01, 10 и 11). У таквој ситуацији треба изабрати достављача као ранију нит са најближим спекулативним статусом. Да би се избјегла потреба за посебним арбитром, усвојен је механизам дистрибуиране арбитрације гдје нити у два корака између себе дистрибуирано одлучују која ће бити изабрана као добављач и поставити податак на магистралу.

Овај механизам захтијева да магистрала буде проширена са двије додатне линије за маску нити. Ове двије линије су имплементиране као „ожичена ИЛИ“ кола и обиљежавају се као M_0 (виши бит)) и M_1 (нижи бит). Сваки процесор је повезан на ове двије линије и када је потребно, према протоколу, поставља на њих двобитну вриједност маске нити коју извршава. Стање „ожиченог ИЛИ“ кола ће бити високо (1) ако је бар један процесор поставио јединицу на линију, а ниско (0) ако ниједан процесор није поставио јединицу на ову линију.

Дистрибуирана арбитрација се обавља у два корака (један за сваки од битова). Након што су могући достављачи поставили своје маске нити које извршавају на линије магистрале, сваки од њих консултује стање M_0 сигнала у првом кораку и ако је стање овог сигнала исто као и виши бит маске његове нити, наставља да приступа магистрали. Међутим, ако су ова два стања различита, процесор

одустаје (другим ријечима уклања маску нити са обје линије на магистрали). Непосредно након тога, у другом кораку, исти процес арбитрације се понавља, али на M_1 линији магистрале (нижи бит). Коначно, процесор који је поставио маску нити са највећом вриједношћу се бира као достављач. Након тога, изабрани достављач може да постави тражену ријеч на магистралу.

На примјер, нека постоје три могућа достављача који постављају на линије маске на магистрали вриједности 00, 01, и 10. Зато ће обје линије маске на магистрали имати високу вриједност. Како M_0 има вриједност 1, процесори који су поставили маске 00 и 01 се повлаче, а само процесор са маском 10 остаје. Сада, стање M_0 је 1 и M_1 је 0. У другом кораку поступак се понавља са M_1 и као достављач се бира нит са маском 10.

7.2 Акције и прелази између стања у SISC-WI протоколу

У овој секцији ће бити представљене све акције које настају приликом поготка и промашаја при читању и упису, замјене у кеш меморији, као и одговарајући прелази између стања у SISC-WI протоколу који се одвијају код неспекулативне односно спекулативних нити током наведених акција. Опис свих ових акција је илустрован одговарајућим дијаграмима стања за које важи сљедећи начин означавања прелаза:

- Нотација *A/B сигнал* означава да, ако кеш контролер детектује догађај A од стране процесора, поред промјене стања, он генерише трансакцију на магистрали или акцију B и поставља *сигнал*.
- Нотација *A магистрала-сигнал/B,* односно *A магистрала-сигнал,* означава да ако кеш контролер детектује догађај A на магистрали и *сигнал* на магистрали (L-: самоизазвана промјена – локална; P-: промјену проузрокује ранија нит; S-: промјену проузрокује каснија нит), тада поред промјене стања, он генерише трансакцију на магистрали или акцију B ("-" означава да нема акције).
- Пуна линија означава да је процесор изазвао промјену стања, док испрекидана линија означава да је промјену изазвала акција на магистрали.

7.2.1 Погодак приликом читања

Неспекулативна нит. Ако неспекулативна нит постави захтјев за читање (*PrRd*) ријечи у кеш меморији која се налази у стању StS, StM или M, захтјев се обрађује локално и ријеч остаје у непромијењеном стању. Дијаграм прелаза стања за неспекулативну нит је приказан на слици 7-1.

Спекулативна нит. Ако спекулативна нит постави захтјев за читање (*PrRd*) ријечи у кеш меморији која се налази у стању SSp, StMSp, StSpM, StSp, MSp, MSpF, MSSp или MSSpF, захтјев се обрађује локално и ријеч остаје у непромијењеном стању.

Ако је ријеч у стању S, стање ће јој бити промијењено у стање SSp да би се сачувала информација да је ријеч спекулативно прочитана ради детекције нарушавања RAW зависности у будућности.

Ако је ријеч у стању StS када спекулативна нит постави захтјев за читање, захтјев се обрађује локално и стање ријечи се мијења у стање StSp да би се сачувала информација да је застарјела ријеч спекулативно прочитана.

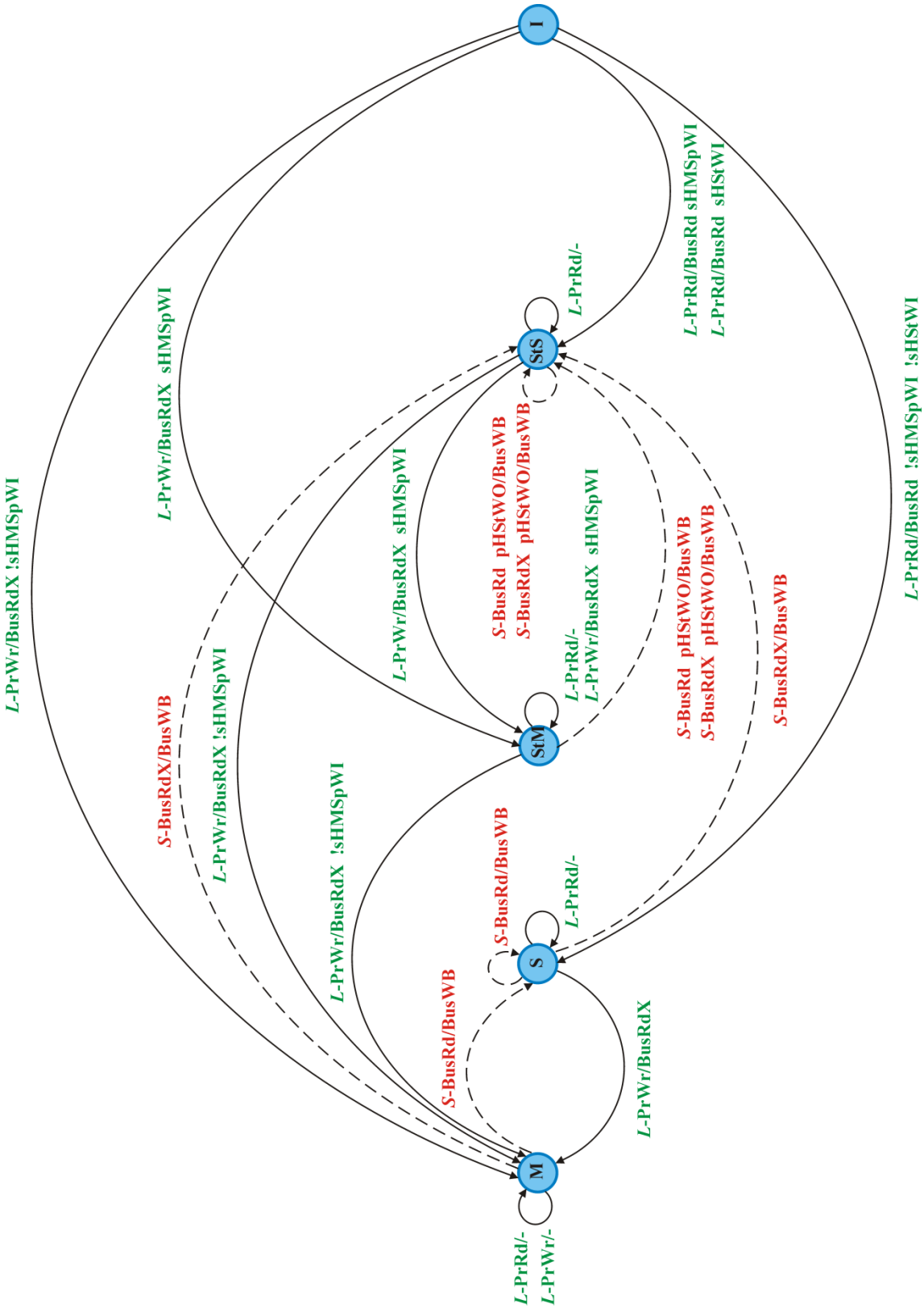
Ако је спекулативна нит поставила захтјев за читање ријечи у кеш меморији која је у стању M, ријеч мора да буде прослијеђена дијељеној L2 кеш меморији да би се сачувала једина кеширана копија, а након тога стање ријечи се мијења у стање SSp да би се сачувала информација да је ријеч спекулативно прочитана ради детекције нарушавања RAW зависности у будућности.

Дијаграм прелаза стања за спекулативну нит за акције иницране од стране процесора је приказан на слици 7-2.

7.2.2 Промашај приликом читања

Неспекулативна нит. Захтјев за читање (*PrRd*) ријечи која се налази у стању INV или која се уопште не налази у локалној L1 кеш меморији изазива промашај приликом читања. Овај догађај проузрокује међунитну комуникацију иницирану од стране потрошача који извршава трансакцију *BusRd* на дијељеној магистрали. Такође, поред захтјева за читање на магистралу се поставља маска неспекулативне нити. Кеш контролери каснијих спекулативних нити ослушкују магистралу. Спекулативне нити које можда имају тражену ријеч у стању M или S одговарају постављањем своје маске на дијељену магистралу.

Тада механизам дистрибуиране арбитрације бира достављача и дозвољава му да постави тражену ријеч на дијељену магистралу. Тражена ријеч се учитава у стању S у L1 кеш меморију процесора који извршава неспекулативну нит. Уколико је тражена ријеч код достављача била у стању M потребно је да се та ријеч упише у L2 кеш меморију да би се сачувала једина најскорија кеширана копија, а након тога се стање ријечи мијења у стање S на страни достављача. У случају да је ријеч код достављача била у стању S тада се њено стање не мијења.



Слика 7-1. Дијаграм прелаза стања за неспекулативну нит - акције иницирane од стране процесора и магистрале

Уколико ниједна каснија спекулативна нит, која може бити достављач, нема тражену ријеч у стању M или S, и ако каснија спекулативна нит има ријеч у стању StS, каснија нит поставља свој код маске на дијељену магистралу. Механизам арбитрације бира достављача и дозвољава му да постави тражену ријеч на дијељену магистралу. Поред тога, достављач поставља сигнал *sHStWO* да обавијести неспекулативну нит да постави стање тражене ријечи у стање StS у L1 кеш меморији процесора који извршава неспекулативну нит.

Ако ниједна каснија спекулативна нит, која може бити достављач, нема ријеч у стању M, S или StS, на неспекулативан захтјев за читање одговара дијељена L2 кеш меморија. Тражена ријеч добија стање S када се учита у L1 кеш меморију процесора који извршава неспекулативну нит, под условом да ниједна каснија спекулативна нит нема тражену ријеч у стању MSp, MSpF, MSSp или MSSpF. У супротном, каснија нит поставља сигнал *sHMSpWO* да би обавијестила неспекулативну нит да тражена ријеч треба да добије стање StS приликом учитавања у њену L1 кеш меморију.

Дијаграм прелаза стања за неспекулативну нит и акције иницране од стране процесора и одговарајуће акције на магистрали је приказан на слици 7-1.

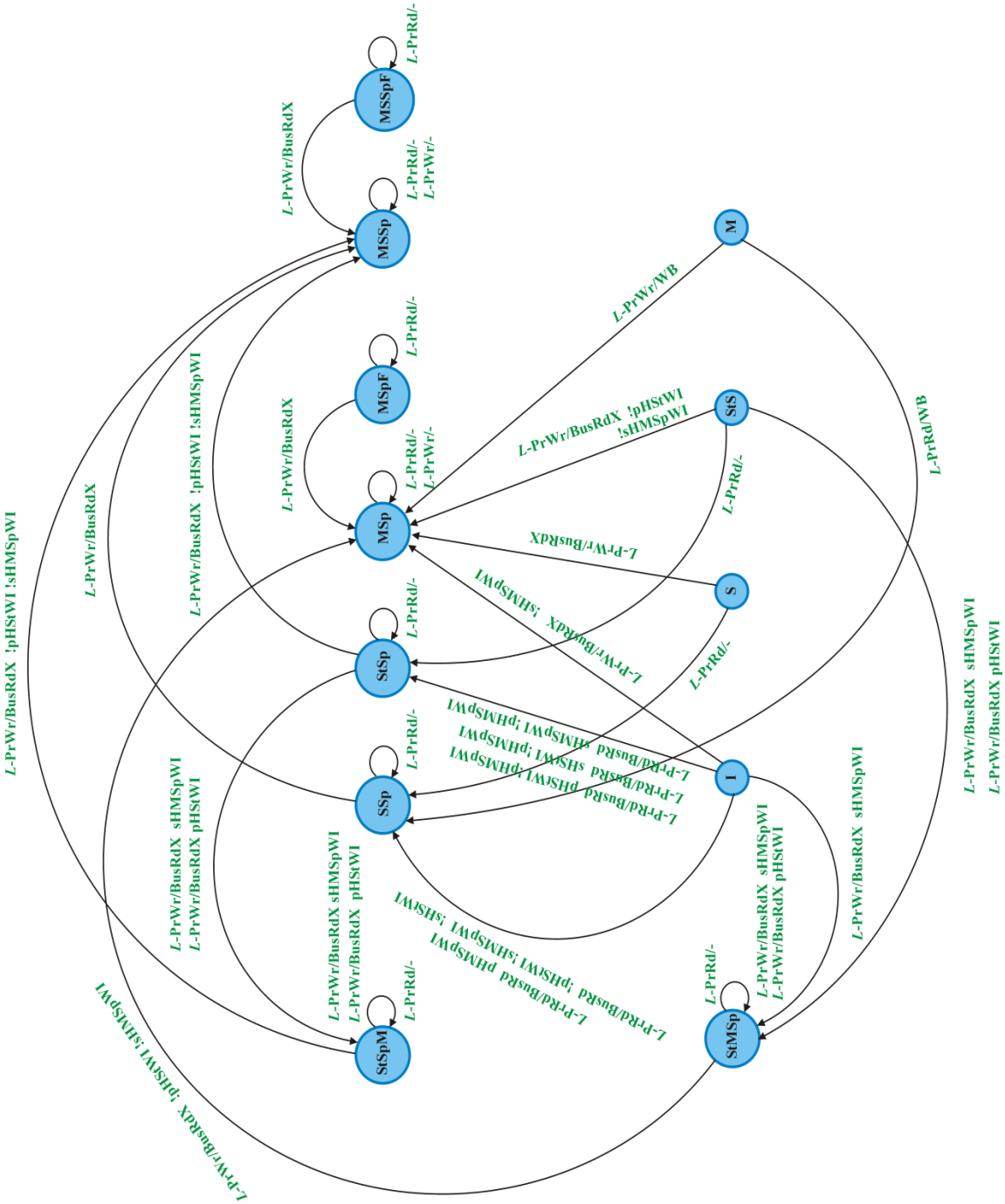
Спекулативна нит. Ако спекулативна нит постави захтјев за читање (*PrRd*) за који се догоди промашај у локалној L1 кеш меморији, нит покреће трансакцију *BusRd* на дијељеној магистрали и поставља свој код маске на магистралу. Додатно, промашај приликом читања покреће међунитну комуникацију коју иницира потрошач. Уколико нека неспекулативна нит или најближа спекулативна нит имају тражену ријеч у стању M, достављача међу њима бира механизам дистрибуиране арбитрације.

У супротном, најближа нит, било неспекулативна или спекулативна, која има тражену ријеч у стању S, се бира као достављач. У оба случаја, тражена ријеч се поставља у стање SSp приликом учитавања у L1 кеш меморију процесора који извршава спекулативну нит.

Уколико не постоји тражена ријеч у стању M или S у било којој L1 кеш меморији раније или касније нити, ранија спекулативна нит која има тражену ријеч у стању MSp, MSpF, MSSp, или MSSpF се бира као достављач. Тада се изабраном достављачу дозвољава да тражену ријеч постави на дијељену магистралу. Истовремено, достављач поставља сигнал *pHMSpWO* да би обавијестио спекулативну нит, која је поставила захтјев за трансакцију *BusRd*, да стање тражене ријечи постави у стање SSp приликом учитавања у своју L1 кеш меморију.

Уколико се не ради ни о једном од претходно наведених случајева, тада се ранија спекулативна нита која има тражену ријеч у стању SSp бира као достављач. Стање тражене ријечи се поставља у стање SSp приликом учитавања у L1 кеш меморију процесора који извршава спекулативну нит.

Уколико не постоји тражена ријеч ни у једном од претходно наведених стања у било којој L1 кеш меморији раније или касније нити, неспекулативна нит која има тражену ријеч у стању StM се бира као достављач. Неспекулативна нит поставља сигнал *pHStWO* да обавијести спекулативну нит да постави стање тражене ријечи у стање StSp приликом њеног учитавања у припадајућу L1 кеш меморију.



Слика 7-2. Дијаграм прелаза стања за спекулативну нит - акције иницране од стране процесора

Уколико не постоји тражена ријеч у стању StM у било којој L1 кеш меморији раније или касније нити, тада се најближа ранија неспекулативна или спекулативна нит, или најближа каснија спекулативна нит, која има тражену ријеч у стању StS бира као достављач. Ако је достављач ранија, било неспекулативна или спекулативна нит, она поставља сигнал pHStWO да обавијести спекулативну нит која је поставила захтјев за трансакцију *BusRd*, да постави стање тражене ријечи у стање StSp приликом учитавања у L1 кеш меморију процесора који извршава спекулативну нит. Ако је достављач каснија спекулативна нит, она поставља сигнал *sHStWO* да обавијести спекулативну нит која је поставила захтјев за трансакцију *BusRd*, да постави стање тражене ријечи у стање StSp приликом учитавања у своју L1 кеш меморију

Уколико се не ради ни о једном од претходно наведених случајева, тада се најближа ранија спекулативна нит која има тражену ријеч у стању StMSp, StSpM или StSp бира као достављач. Тада се дозвољава изабраном достављачу да постави тражену ријеч на дијељену магистралу. Достављач поставља сигнал *pHStWO* да обавијести спекулативну нит која је поставила захтјев за трансакцију *BusRd*, да стање тражене ријечи постави у стање StSp приликом учитавања у своју L1 кеш меморију.

Ако не постоји достављач у тренутку када је постављен захтјев за читање на дијељеној магистрали, на захтјев ће одговорити дијељена L2 кеш меморија. Стање тражене ријечи ће бити постављено у стање SSp приликом учитавања у L1 кеш меморију спекулативне нити која је издала захтјев за читање, само ако не постоји тражена ријеч код неке касније нити у стању StMSp, StSpM, MSp, MSpF, MSSp или MSSpF. У супротном, каснија нит која има тражену ријеч у MSp, MSpF, MSSp или MSSpF стању, поставља сигнал *sHMSpWO*, док каснија нит која има тражену ријеч у стању StMSp или StSpM, поставља сигнал *sHStWO* да би обавијестио спекулативну нит која је издала захтјев за читање, да постави стање тражене ријечи у стање StSp приликом учитавања у своју L1 кеш меморију.

Промјене стања тражене ријечи код достављача обављају се на сљедећи начин:

- ако је тражена ријеч у стању M, ријеч се уписује у дијељену L2 кеш меморију да би се избјегао губитак валидне копије. Траженој ријечи се стање M мијења у стање S на страни достављача.
- ако је тражена ријеч у стању MSp код достављача, стање ријечи се мијења у стање MSpF.
- ако је тражена ријеч у стању MSSp код достављача, стање ријечи се мијења у стање MSSpF.
- ако је тражена ријеч у стању StM код достављача, ријеч се уписује у дијељену L2 кеш меморију да би се избијегао губитак валидне копије, а затим се ријеч прослијеђује спекулативној нити која је поставила захтјев за читање. Траженој ријечи се стање мијења у стање StS на страни достављача.
- ако је тражена ријеч у стању S, SSp, MSpF, MSSpF, StS, StSp, StSpM или StMSp код достављача, стање ријечи остаје исто.

Дијаграм прелаза стања за спекулативну нит за акције иницране од стране процесора је приказан на слици 7-2, док су акције иницране са магистрале приказане на слици 7-3.

7.2.3 Погодак приликом уписа

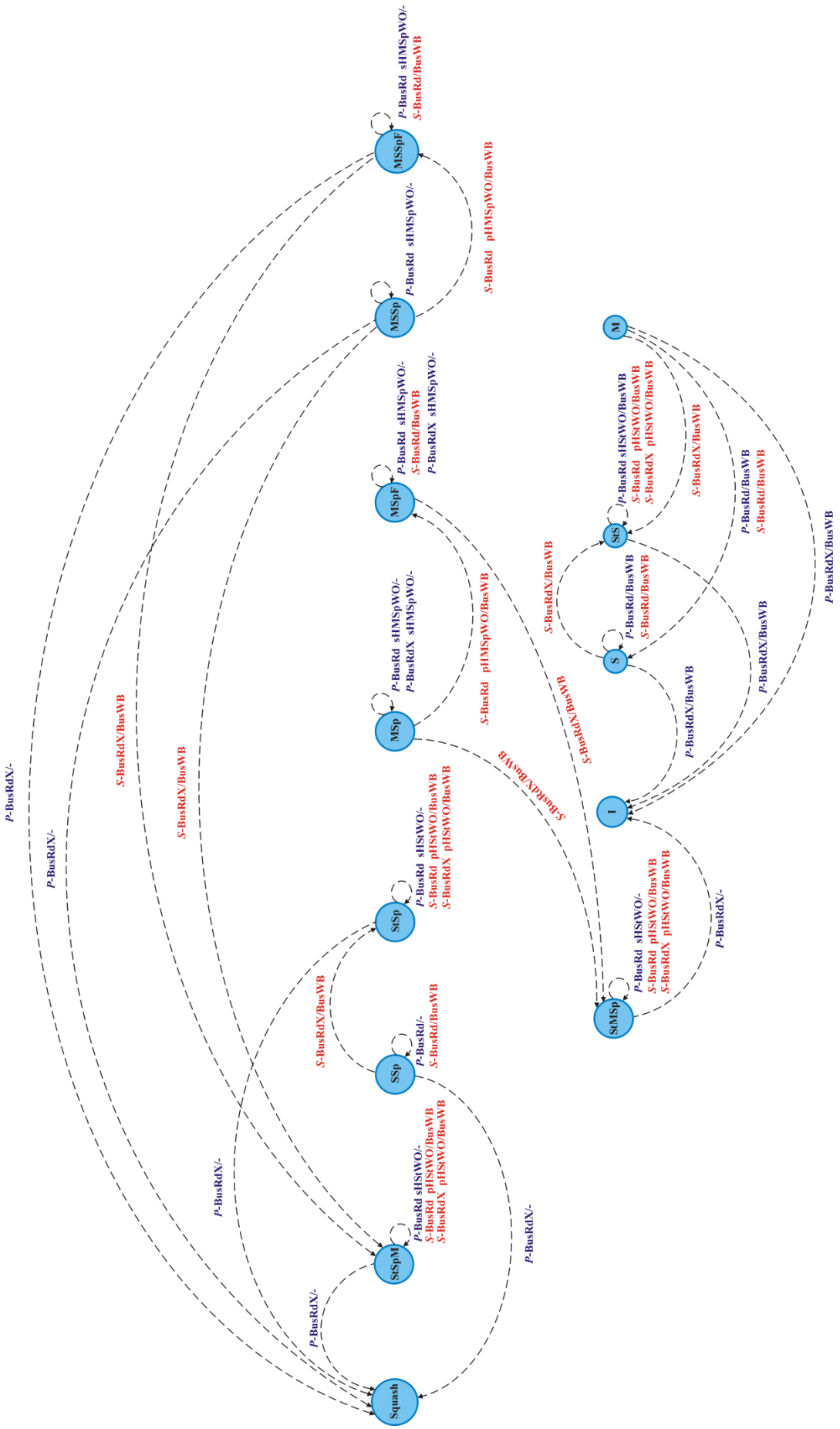
Неспекулативна нит. Ако је ријеч у кеш меморији у стању M када неспекулативна нит постави захтјев за упис (*PrWr*), ријеч се уписује у локалну L1 кеш меморију и остаје у стању M, без трансакције на дијељеној магистрали.

Ако је ријеч у стању S када неспекулативна нит постави захтјев за читање, трансакција *BusRdX* се обавља на дијељеној магистрали. Сви кеш контролери процесорских језгара који извршавају касније спекулативне нити ослушкују дијељену магистралу и провјеравају одговарајући улаз у својој L1 кеш меморији. Нарушавање RAW зависности се детектује ако је одговарајући улаз у L1 кеш меморији процесорског језгра које извршава каснију спекулативну нит, у стању SSp.

Из тог разлога, та спекулативна нит, као и све касније нити бивају поништене и рестартоване. На другој страни, ако је одговарајући улаз у L1 кеш меморији процесорског језгра које извршава каснију спекулативну нит, у стању S, одговарајући улаз ће бити поништен. Тада, неспекулативна нит врши упис у ријеч и мијења њено стање у M.

Ако је ријеч у стању StS када неспекулативна нит изда захтјев за упис, обавља се трансакција *BusRdX* на дијељеној магистрали. Сви кеш контролери процесорских језгара који извршавају касније спекулативне нити, ослушкују дијељену магистралу и провјеравају одговарајући улаз у својој L1 кеш меморији. Нарушавање RAW зависности се детектује ако је одговарајући улаз у L1 кеш меморији процесорског језгра које извршава каснију спекулативну нит, у неком од стања StSp, StSpM, MSSp, MSSpF или SSp. Из тог разлога, та спекулативна нит, као и све касније нити се поништавају и рестартују. Ипак, ако је одговарајућа ријеч у неком од стања StS или StMSp у L1 кеш меморији касније нити, иста ће бити само поништена. Ако је одговарајућа ријеч у неком од стања MSp или MSpF у L1 кеш меморији касније нити, њено стање остаје исто и та спекулативна нит поставља сигнал *sHMSpWO* да обавијести неспекулативну нит да стање одговарајуће ријечи промијени у стање StM, у супротном стање дате ријечи се мијења у стање M.

Ако је ријеч у стању StM када неспекулативна нит постави захтјев за упис, покреће се *BusRdX* трансакција на дијељеној магистрали као и у претходном случају. Сви кеш контролери процесорских језгара који извршавају касније спекулативне нити, ослушкују дијељену магистралу и провјеравају одговарајући улаз у својој L1 кеш меморији. Нарушавање RAW зависности се детектује ако је одговарајући улаз у L1 кеш меморији процесорског језгра који извршава каснију спекулативну нит, у стању SSp.



Слика 7-3. Дијаграм прелаза стања за спекулативну нит - акције инициране од стране магистрале

Из тог разлога, та спекулативна нит се поништава и рестартује. Међутим, ако је одговарајућа ријеч у стању StMSp у L1 кеш меморији касније нит, она ће бити поништена. Ако је одговарајућа ријеч у стању MSp или MSpF у L1 кеш меморији касније нити, њено стање остаје исто и та спекулативна нит поставља сигнал *sHMSpWO* да обавијести неспекулативну нит да остави стање ријечи непромијењено, у супротном стање ријечи се мијења у стање M.

Дијаграм прелаза стања за неспекулативну нит и акције инициране од стране процесора и одговарајуће акције на магистрали је приказан на слици 7-1.

Спекулативна нит. Ако је ријеч у кеш меморији у стању S када спекулативна нит постави захтјев за упис, покреће се трансакција *BusRdX* на дијељеној магистрали. Сви кеш контролери процесорских језгара који извршавају касније спекулативне нити, ослушкују магистралу и провјеравају одговарајући улаз у својој L1 кеш меморији. Ако је одговарајући улаз у L1 кеш меморији касније спекулативне нити, у стању SSp, детектује се зависност међу подацима и та касније нит, као и све наредне, се поништавају и рестартују. У случају када је ријеч у L1 кеш меморији касније спекулативне нити, у стању S, она се поништава. Тада спекулативна нит врши упис и мијења стање ријечи у стање MSp. Такође, сви кеш контролери процесорских језгара који извршавају претходне нити, неспекулативне и спекулативне, ослушкују дијељену магистралу за вријеме трансакције *BusRdX* да би означили у својим кеш меморијама податак који не смије бити коришћен од стране нити која ће се у будућности извршавати на истом процесорском језгру. Из тог разлога, ранија нит мијења стање одговарајућег улаза у својој L1 кеш меморији на сљедећи начин:

- ако је ријеч у стању S, мијења јој се стање у стање StS.
- ако је ријеч у стању SSp, мијења јој се стање у стање StSp.

Ако је ријеч у кеш меморији у стању SSp када спекулативна нит постави захтјев за упис, покреће се трансакција *BusRdX* на дијељеној магистрали, као и у претходном случају. Врши се упис у ријеч и њено стање се мијења у стање MSSp. Ако је одговарајући улаз у стању SSp у L1 кеш меморији касније нити, нарушавање RAW зависности се детектује и та спекулативна нит, као и све наредне нити, се поништавају и рестартују. У случају да је ријеч у L1 кеш меморији касније нити у стању S, она се поништава. Такође, сви кеш контролери процесорских језгара који извршавају претходне нити, неспекулативне и спекулативне, ослушкују магистралу током трансакције *BusRdX* да би означили у својим кеш меморијама податак који не смије бити коришћен од стране нити која ће се извршавати у будућности на истом процесорском језгру. Из тог разлога, раније нити мијењају стање одговарајућег улаза у својим L1 кеш меморијама на сљедећи начин:

- ако је ријеч у стању S, мијења јој се стање у стање StS.
- ако је ријеч у стању SSp, мијења јој се стање у стање StSp.

- ако је ријеч у стању MSpF, мијења јој се стање у стање StMSp.
- ако је ријеч у стању MSSpF, мијења јој се стање у стање StSpM.

Ако је ријеч у кеш меморији у стању StS или StSp када спекулативна нит постави захтјев за упис, покреће се *BusRdX* трансакција на дијељеној магистрали. Ако је одговарајући улаз у L1 кеш меморији касније нити, у неком од стања StSp, StSpM, SSp, MSSp или MSSpF детектује се нарушавање RAW зависности, и тада та спекулативна нит, као и све наредне, се поништавају и рестартују. Међутим, ако је одговарајућа ријеч у неком од стања StS или StMSp у L1 кеш меморији касније нити, она се поништава. Ако је одговарајућа ријеч у неком од стања MSp или MSpF у L1 кеш меморији касније нити, њено стање остаје исто и та каснија нит поставља сигнал *sHMSpWO* да би обавијестила спекулативну нит да промијени стање ријечи у стање StMSp, у супротном она мијења стање ријечи у стање MSp. Поред тога, ако је одговарајућа ријеч у неком од стања StSp, StMSp, или StSpM у L1 кеш меморији раније нити, њено стање остаје исто и ранија нит поставља сигнал *pHStWO* да би обавијестила спекулативну нит која је поставила захтјев за упис да промијени стање ријечи у стање StMSp.

Ако је ријеч у кеш меморији у стању StSpM или StMSp када спекулативна нит постави захтјев за упис, покреће се трансакција *BusRdX* на дијељеној магистрали. Ако је одговарајући улаз у L1 кеш меморији касније нити у неком од стања StSp или SSp, та нит, као и све наредне, се поништавају и рестартују. Ако је одговарајућа ријеч у неком од стања MSp или MSpF у L1 кеш меморији касније нити, њено стање остаје исто и та каснија нит поставља сигнал *sHMSpWO* да би обавијестила спекулативну нит да стање ријечи треба да остане исто, у супротном стање ријечи се мијења у стање MSSp. Као и у претходном случају, ако је одговарајућа ријеч у неком од стања StS или StSp у L1 кеш меморији раније нити, њено стање остаје исто и ранија нит поставља сигнал *pHStWO* да обавијести спекулативну нит која је поставила захтјев за упис да стање ријечи треба да остане исто.

Ако је ријеч у кеш меморији у неком од стања MSp или MSSp када спекулативна нит постави захтјев за упис, вриједност ријечи се ажурира и њено стање остаје исто. У овом случају не покреће се трансакција *BusRdX* на дијељеној магистрали.

Ако је ријеч у кеш меморији у стању MSpF када спекулативна нит постави захтјев за упис, вриједност ријечи се ажурира и мијења јој се стање у стање MSp. Ако је ријеч у кеш меморији у стању MSSpF када спекулативна нит постави захтјев за упис, врши се упис и мијења јој се стање у стање MSSp да би се сачувала информација да је ријеч спекулативно прочитана раније. Такође, у оба случаја се покреће трансакција *BusRdX* на дијељеној магистрали да би се детектовала могућа нарушавања RAW зависности ако је одговарајући улаз у L1 кеш меморији касније нити у стању SSp. У случају нарушавања зависности по подацима та каснија нит, као и све наредне нити, се поништавају и рестартују.

Ако је одговарајућа ријеч у L1 кеш меморији раније нити, у неком од стања StS, StMSp, StSp или StSpM, стање јој остаје исто и та ранија нит поставља сигнал *pHStWO*.

Ако је ријеч у кеш меморији у стању M када спекулативна нит постави захтјев за упис, покреће се трансакција *WB* да би се осигурало да једина исправна копија ријечи не буде уништена приликом спекулативног модификовања. Након тога се стање ријечи мијења у стање MSp.

Дијаграм прелаза стања за спекулативну нит приликом поготка при упису за акције инициране од стране процесора је приказан на слици 7-2, док су акције инициране са магистрале приказане на слици 7-3.

7.2.4 Промашај приликом уписа

Неспекулативна нит. Захтјев за упис (*PrWr*) неспекулативне нити у ријеч која је у стању INV у локалној L1 кеш меморији изазива промашај приликом уписа. Овај догађај проузрокује упис у ријеч, промјену њеног стања и покретање трансакције *BusRdX* на дијељеној магистрали.

Касније нити ослушкују дијељену магистралу и уколико ниједна нема одговарајућу ријеч у својој L1 кеш меморији у неком од стања MSp или MSpF, неспекулативна нит уписује у ријеч и мијења јој стање у стање M. У супротном, каснија нит која има ријеч у неком од стања MSp или MSpF поставља сигнал *sHMSpWO* да би обавијестила неспекулативну нит да промијени стање ријечи у стање StM. Ако је одговарајући улаз у L1 кеш меморији неке касније нити у неком од стања StS или StMSp, он се поништава. У случају да је одговарајући улаз у стању M у L1 кеш меморији касније нити, исти мора да се упише у дијељену L2 кеш меморију да би се очувала једина најновија копија, а затим се поништава. Уз то, ако је одговарајући улаз у L1 кеш меморији касније нити у стању S, он ће бити поништен.

Нарушавање RAW зависности се детектује ако је одговарајући улаз у L1 кеш меморији процесорског језгра које извршава каснију нит у неком од стања SSp, StSp, StSpM, MSSp или MSSpF. Тада се та спекулативна нит, као и све наредне нити, поништавају и рестартују.

Дијаграм прелаза стања за неспекулативну нит и акције иницране од стране процесора и одговарајуће акције на магистрали је приказан на слици 7-1.

Спекулативна нит. Када спекулативна нит постави захтјев за упис (*PrWr*) за ријеч која је у стању INV у локалној L1 кеш меморији долази до промашаја приликом уписа. Овај спекулативни промашај приликом уписа проузрокује покретање трансакције *BusRdX* на дијељеној магистрали. Сви кеш контролери процесорских језгара који извршавају нит на вишем нивоу спекулације, ослушкују дијељену магистралу и провјеравају одговарајући улаз у својој L1 кеш меморији. Ако каснија нит има одговарајућу ријеч у неком од стања MSp или

MSpF у својој L1 кеш меморији, тада она поставља сигнал *sHMSpWO* да би обавијестила спекулативну нит која је проузроковала промашај приликом уписа, да промијени стање ријечи у стање StMSp. При томе, ако је одговарајући улаз у L1 кеш меморији касније нити у неком од стања StS или StMSp, исти се поништава.

Међутим, уколико ниједна каснија нит нема ријеч у неком од стања MSp или MSpF, дата ријеч се ажурира и мијења јој се стање у стање MSp. У случају да је одговарајући улаз у стању M у L1 кеш меморији касније нити, он се уписује у дијељену L2 кеш меморију да би се спријечио губитак једине најскорије копије података, а затим се поништава. Уз то, ако је одговарајући улаз у L1 кеш меморији касније нити у стању S, он ће бити поништен. Такође, сви кеш контролери процесорских језгара који извршавају раније нити, неспекулативна и спекулативне нити, ослушкују магистралу током трансакације *BusW* да би сачували информацију да податак у њиховим кеш меморијама не могу да користе нити које ће се у будућности извршавати на истом процесорском језгру тако што мијењају стање одговарујућег улаза у својим локалним L1 кеш меморијама у неко од *stale* стања. Ако је одговарајући улаз у L1 кеш меморији раније нити у стању S, мијења му се стање у стање StS, а ако је у стању SSp његово стање се мијења у стање StSp. У случају да је одговарајући улаз у L1 кеш меморији раније нити у стању StM, он се уписује у дијељену L2 кеш меморију, а затим му се стање мијења у стање StS. Међутим, ако је у неком од стања StS, StSp, StSpM или StMSp, стање му остаје исто. Ако се налази у неком од стања MSp или MSpF, мијења му се стање у стање StMSp, док ако се налази у неком од стања MSSp или MSSpF мијења му се стање у стање StSpM. У случају да је одговарајући улаз у L1 кеш меморији претходника у стању M, он се уписује у дијељену L2 кеш меморију да би се спријечио губитак једине најскорије копије, а након тога му се мијења стање у стање StS.

Нарушавање RAW зависности се детектује ако је одговарајући улаз у L1 кеш меморији процесорског језгра које извршава каснију нит у неком од стања SSp, MSSp или MSSpF, тада се та спекулативна нит, као и сви наредне нити, поништавају и рестартују. Такође, нарушавање RAW зависности се детектује ако је одговарајући улаз у L1 кеш меморији касније нити у неком од стања StSp или StSpM. Тада се та нит, као и свака каснија нит која има тај улаз у неком од стања StSp или StSpM, поништавају и рестартују.

Дијаграм прелаза стања за спекулативну нит приликом промашаја при упису за акције иницране од стране процесора је приказан на слици 7-2, док су акције иницране са магистрале приказане на слици 7-3.

7.2.5 Замјена ријечи у кеш меморији

Као што је најављено у петој глави, предложен је алгоритам замјене који узима у обзир стање ријечи у кеш меморији као примарни критеријум приликом

избора кандидата за замјену умјесто историје претходних приступа ријечима. Он дозвољава спекулативним нитима да у својим L1 кеш меморијама изврше замјену ријечи приликом операција читања и ажурирања ријечи које су или ријечи у модификованом стању M или ријечи које су у немодификованим стањима S или StS. На тај начин се спречава заустављање спекулативне нити која се извршава и спекулативно извршавање може бити настављено. Међутим, међу ријечима које су у истом стању и које су кандидати за избацивање из L1 кеш меморије, било неспекулативне или спекулативне нити, може се примијенити FIFO, LRU или алгоритам случајног избора као секундарни критеријум за бирање ријечи која ће бити замијењена.

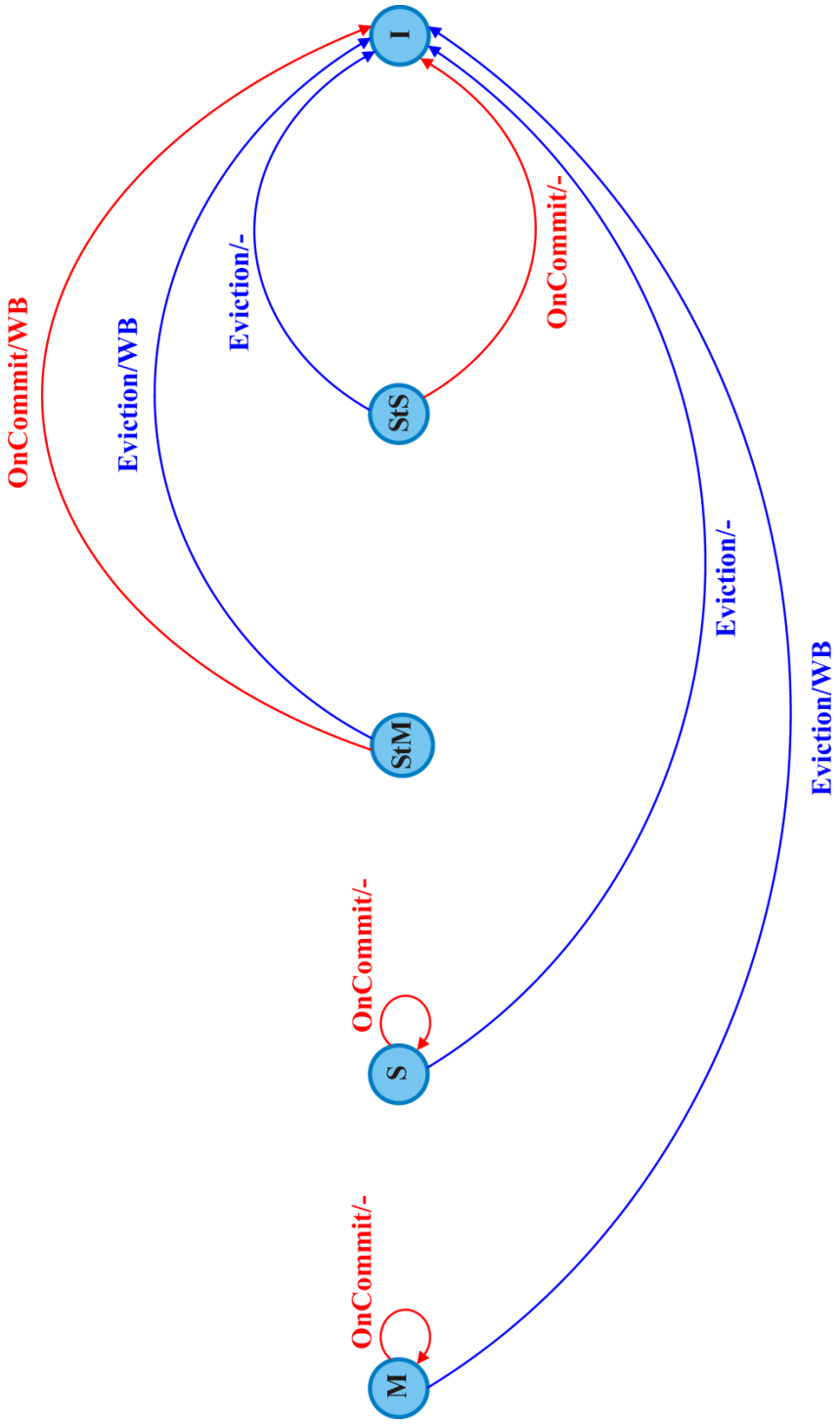
Када се појави потреба за избацивањем неке ријечи у одговарајућем сету кеш меморије, први кандидати су ријечи које се налазе у стању INV. Уколико не постоји ријеч у стању INV, алгоритам замјене за избацивање бира ријеч у стању StS чија вриједност постоји и у главној меморији и/или у кеш меморији неке од других нити. Ако не постоји ријеч у стању StS у кеш меморији, тада се ријеч у стању S, чија вриједност постоји и у главној меморији и/или у кеш меморији неке од других нити, бира за избацивање. Уколико нема ријечи у поменутим стањима, тада се за избацивање из кеш меморије бира ријеч у стању StM која постоји само у кешу неспекулативне нити. На крају, ако не постоји ниједна ријеч у наведеним стањима, за избацивање се бира ријеч која се налази у стању M.

Ако нема ријечи у поменутим стањима и спекулативна ријеч у неком од стања SSp, StSp, StSpM, StMSp, MSp, MSpF, MSSp или MSSpF мора бити замијењена у L1 кеш меморији спекулативне нити, SISC-WI протокол третира ову ситуацију као недозвољену, а затим поништава спекулацију и рестартује нит. Дозвола за замјену спекулативне ријечи у кеш меморији може проузроковати проблем у спекулативном извршавању зато што се на тај начин онемогућава протоколу да прати могућа нарушавања зависности по подацима између нити.

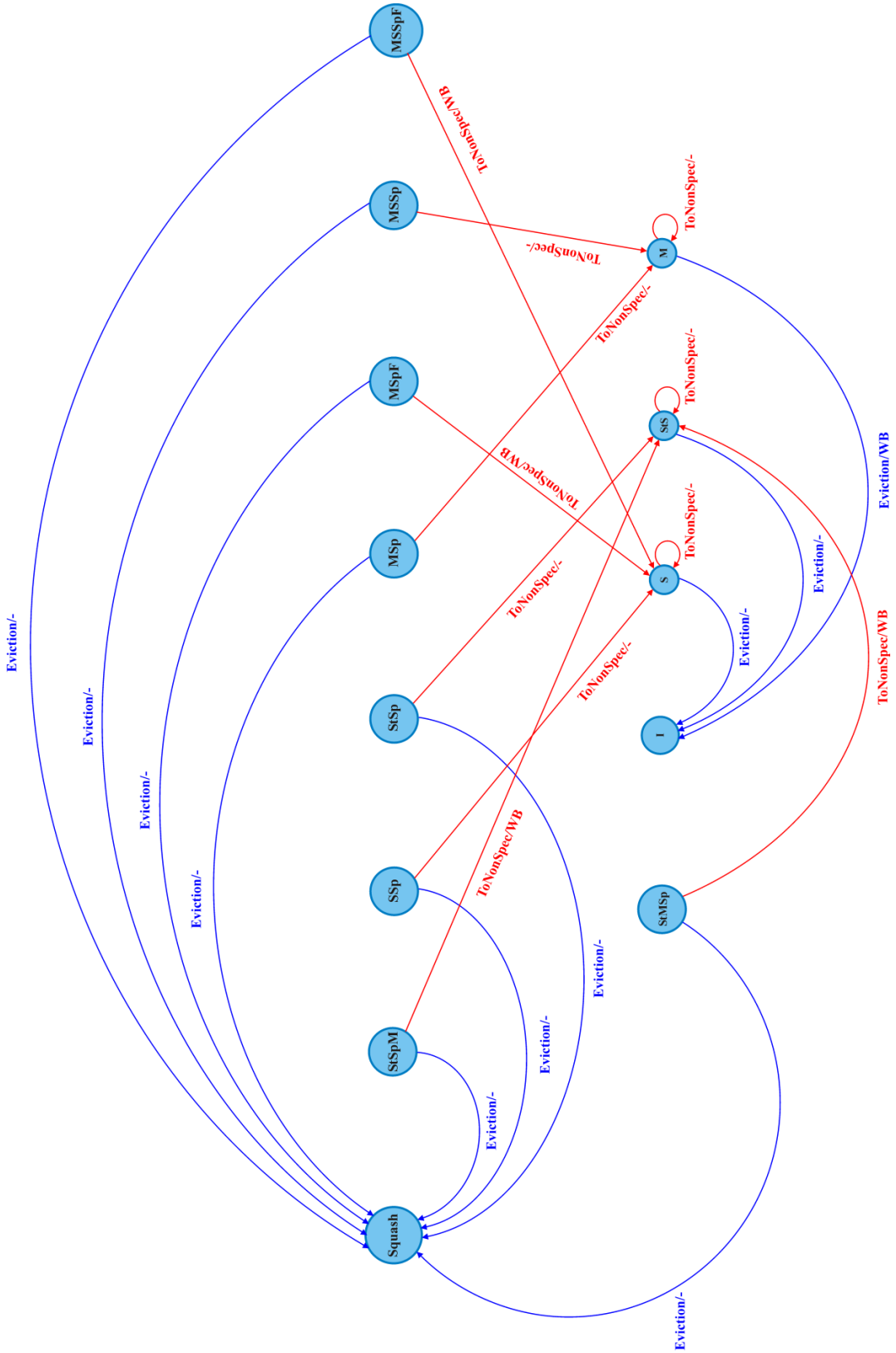
Ако постоји више ријечи у истом изабраном стању, S, StS или StM, стандардни FIFO, LRU или алгоритам случајног избора могу да се користе као секундарни критеријум за бирање која ће ријеч између осталих бити замијењена. У случају да је ријеч у стању S или StS, тада није потребан одложени упис у L2 кеш меморију, а у случају да је ријеч у стању StM она се уписује у L2 кеш меморију да би се очувала једина, најскорије промијењена кеширана копија, а затим се може замијенити у одговарајућој L1 кеш меморији.

Ако ријеч у стању M мора да буде замијењена у L1 кеш меморији неспекулативне или спекулативне нити, та ријеч мора да буде уписана у L2 кеш меморију да би се сачувала једина најскорије промијењена кеширана копија, а након тога се може замијенити у одговарајућој L1 кеш меморији. Такође, као и претходном случају, ако постоји више ријечи у стању M у локалној L1 кеш меморији, секундардним критеријум се бира која ће ријеч бити замијењена међу оним које су у стању M.

Акције приликом замјене ријечи у кеш меморији неспекулативне и спекулативне нити су приказане на сликама 7-4 и 7-5, респективно.



Слика 7-4. Дијаграм прелаза стања за неспекулативну нит током замјене ријечи и приликом завршетка нити: *Eviction* – замјена ријечи у кеш меморији; *OnCommit* – комитовање нити



Слика 7-5. Дијаграм прелаза стања за спекулативну нит током замјене ријечи и приликом преласка у неспекулативно извршавање: *Eviction* – замјена ријечи у кеш меморији; *ToNonSpec* – нит постаје неспекулативна

7.2.6 Завршавање, поништавање и иницијализовање нити

Нити се завршавају у секвенцијалном поретку, па нит мора да постане неспекулативна прије него што се заврши и прије него што се нова нит иницира на истом процесору.

Завршетак неспекулативне нити. Када се неспекулативна нит завршава свака ријеч у стању M у локалној L1 кеш меморији процесорског језгра које извршава нит остаје у стању M. Ова операција је у потпуности локална на нивоу L1 кеш меморије и не иницира постављање захтјева на магистрали. Стога, модификоване и потврђене верзије могу да остану у кеш меморији након што се заврши нит која их је генерисала. На овај начин се избјегава нагло повећање саобраћаја (*bursty traffic*) на магистрали која може повећати вријеме потребно да се нит заврши као и вријеме потребно да се иницијализује нова нит. Модификована и потврђена ријеч се уписује у дијељену L2 кеш меморију када јој се следећи пут приступи на захтјев за читање/упис од стране локалне или неке удаљене нити. Све ријечи у стању StM морају да се упишу у дијељену L2 кеш меморију након чега се поништавају, док се све ријечи у стању StS само поништавају.

Активности које се одвијају приликом завршетка неспекулативне нити су приказане на слици 7-4.

Завршетак спекулативне нити. Када спекулативна нит постане неспекулативна долази до промјена стања ријечи у кеш меморији на сљедећи начин:

- ако је ријеч у стању SSp, стање јој се мијења у стање S,
- ако је ријеч у стању StSp, стање јој се мијења у стање StS,
- ако је ријеч у стању StSpM или StMSp мора да се упише у меморију и стање јој се затим мијења у стање StS у локалној кеш меморији,
- ако је ријеч у стању MSp или MSSp стање јој се мијења у стање M,
- ако је ријеч у неком од стања MSpF или MSSpF мора да се упише у меморију и стање јој се затим мијења у стање S.

Ријеч која се затекне у неком од стања S или M остаје у истом стању када спекулативна нит постане неспекулативна. Нит наставља са својим извршавањем до тренутка када се заврши и када се нова нит иницијализује на истом процесорском језгру.

Активности које се одвијају приликом завршетка спекулативне нити су приказане на слици 7-5.

Иницирање нове спекулативне нити. У тренутку иницирања нове спекулативне нити на слободном процесорском језгру, ријечи у кеш меморији које су у неком од стања M или S остају у одговарајућој L1 кеш меморији. На овај начин протокол елиминише проблем празне L1 кеш меморије на почетку извршавања

нити тако што се модификоване и потврђене ријечи, као и непромијењене ријечи, задржавају у L1 кеш меморији.

Поништавање спекулативне нити. Када се поништава спекулативна нит, све спекулативно модификоване ријечи у неком од стања MSp, MSSp, MSpF или MSSpF, затим све спекулативно учитане ријечи у стању SSp, као и све застарјеле ријечи у неком од стања StSp, StMSp или StSpM, морају да се пониште у одговарајућим L1 кеш меморијама. Остале ријечи у неком од стања M, S или StS се задржавају у кеш меморијама приликом поништавања нити.

7.3 SISC-WI-RS протокол

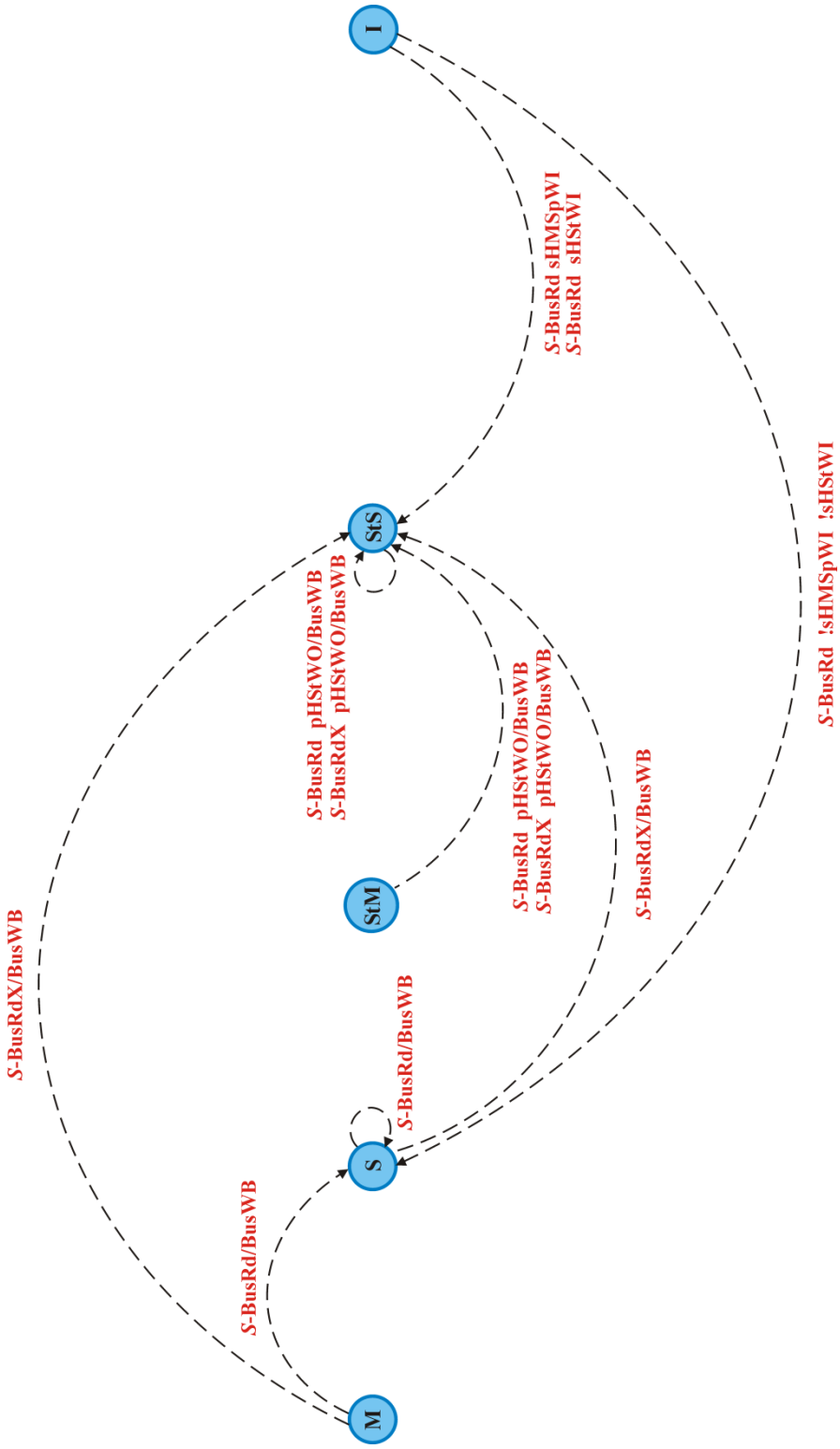
Основни узрок деградације перформанси код протокола заснованих на стратегији поништавања представљају инвалидациони промашаји. Ако има *n* дијељених копија у кеш меморијама, па се упише у једну од њих, осталих *n-1* ће бити поништено. Да би оне поново постале ажурне, потребно је *n-1* инвалидационих промашаја, што за веће *n* изазива велики саобраћај на магистрали. За превазилажење овог проблема, предложена је „*read-snarfing*“ техника код које се једним инвалидационим промашајем ажурирају све копије одједном [EGGE1989, CULL1999]. Већ код првог инвалидационог промашаја након уписа на захтјев једног процесора, када се тражена ријеч појави на магистрали, остале кеш меморије ослушкују магистралу и без захтјева свог процесора поново ажурирају своје копије, па остали инвалидациони промашаји изостају.

Имплементација „*read-snarfing*“ подршке додаје малу комплексност кеш контролеру, а може да побољша перформансе, па је предложена верзија SISC-WI протокола са имплементацијом „*read-snarfing*“ подршке која се назива SISC-WI-RS протокол. У наставку су описане акције овог протокола приликом промашаја током операције читања за вријеме извршавања неспекулативне или спекулативне нити. „*Read-snarfing*“ подршка се имплементира за кеш ријеч у стању INV у L1 кеш меморији сљедбеника, спекулативне нити.

Дијаграми прелаза стања за неспекулативну и спекулативну нит са „*read-snarfing*“ подршком за акције иницране од стране магистрале су приказани на сликама 7-6 и 7-7, респективно.

Неспекулативна нит. Захтјев за читање ријечи (*PrRd*) у стању INV или ријечи која није присутна у локалној L1 кеш меморији изазива промашај приликом читања. Овим се проузрокује међунитна комуникација инициран од стране потрошача и та комуникација се обавља извршавањем *BusRd* трансакције на дијељеној магистрали. Такође, заједно са захтјевом за читање, код маске неспекулативне нити се поставља на дијељену магистралу. Сви кеш контролери сљедбеника, спекулативне нити, ослушкују магистралу и спекулативна нит која

има тражену ријеч у неком од стања M или S, одговара постављањем свог кода маске на дијељену магистралу.



Слика 7-6. Дијаграм прелаза стања за неспекулативну нит - акције иницране од стране магистрале

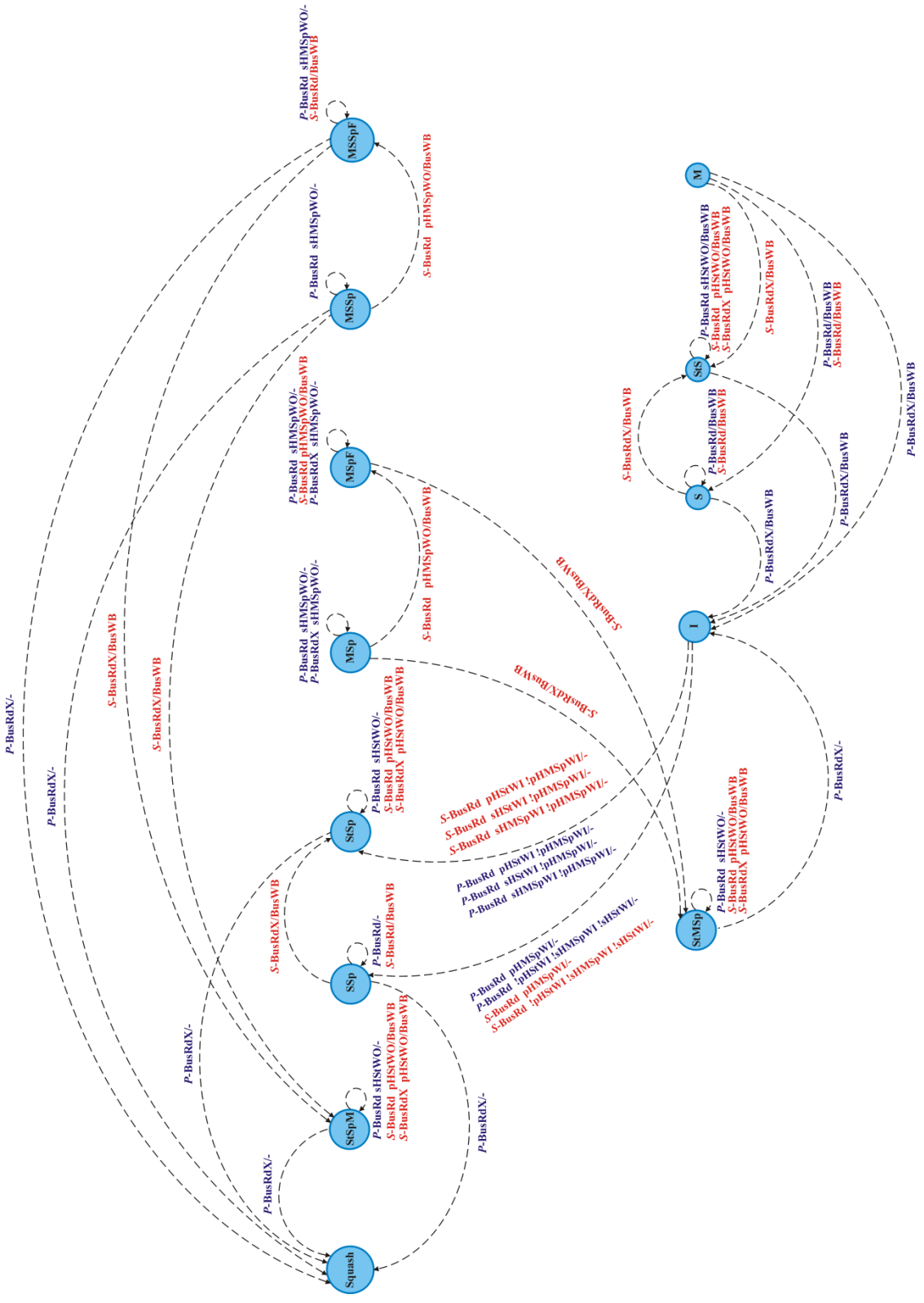
Тада механизам дистрибуиране арбитрације бира достављача и дозвољава му да постави тражену ријеч на дијељену магистралу. Тражена ријеч се учитава у стању S у L1 кеш меморију процесора који извршава неспекулативну нит. Поред тога, одговарајућа ријеч која је у стању INV у L1 кеш меморији неке од каснијих нити се ажурира и њено стање се мијења у стање SSp. Уколико је ријеч код достављача била у стању M уписује се у L2 кеш меморију да би се сачувала једина најскорије модификована кеширана копија, а затим се њено стање мијења у стање S. Уколико је ријеч код достављача у стању S, стање јој остаје исто.

Ако не постоји ниједна каснија нит која има тражену ријеч у неком од стања M или S, тада, каснија нит, која има тражену ријеч у стању StS одговара постављањем свог кода маске на дијељену магистралу. Механизам дистрибуиране арбитрације бира достављача и дозвољава му да постави тражену ријеч на дијељену магистралу. Такође, достављач поставља сигнал *sHStWO* да обавијести неспекулативну нит да приликом учитавања тражене ријечи, њено стање постави у стање StS у L1 кеш меморији процесора који извршава неспекулативну нит. Поред тога, одговарајућа ријеч, која је у стању INV у L1 кеш меморији неке од каснијих нити, се ажурира и стање јој се мијења у стање StSp.

Ако ниједна каснија нит нема тражену ријеч у неком од стања M, S или StS, ријеч се учитава из дијељене L2 кеш меморије. Траженој ријечи стање се поставља у стање S у L1 кеш меморији процесора који извршава неспекулативну нит под условом да ниједна спекулативна нит нема ту ријеч у неком од спекулативно модификованих стања MSp, MSpF, MSSp или MSSpF. У супротном, спекулативна нит поставља сигнал *sHMSpWO* да обавијести неспекулативну нит да постави стање тражене ријечи у стање StS приликом учитавања исте у своју L1 кеш меморију. Такође, одговарајућа ријеч, која је у стању INV у L1 кеш меморији неке од каснијих нити, се ажурира и стање јој се мијења у стање StSp.

Спекулативна нит. Ако спекулативна нит постави захтјев за читање и догоди се промашај у локалној L1 кеш меморији, поставља се *BusRd* захтјев на дијељену магистралу заједно са кодом маске спекулативне нити. Промашај приликом читања проузрокује међунитну комуникацију покренуту од стране потрошача. Могући достављач, кога бира механизам дистрибуиране арбитрације, може бити неспекулативна нит или најближа спекулативна нит, која има тражену ријеч у стању M.

У супротном, неспекулативна нит или најближа спекулативна нит, која има тражену ријеч у стању S, се бира као достављач. У оба случаја, тражена ријеч се поставља у стање SSp приликом учитавања у L1 кеш меморију процесора који извршава спекулативну нит која је поставила *BusRd* захтјев. При томе, одговарајућа ријеч, која је у стању INV у L1 кеш меморији сљедбеника, спекулативне нити, се ажурира и стање јој се мијења у стање SSp, док се одговарајућа ријеч, која је у стању INV у L1 кеш меморији неспекулативне нити ажурира и њено стање се мијења у стање S.



Слика 7-7. Дијаграм прелаза стања за спекулативну нит - акције иницране од стране магистрале

Уколико тражена ријеч не постоји у неком од стања M или S у било којој L1 кеш меморији раније или касније нити, тада се неспекулативна нит која има тражену ријеч у стању StM бира као достављач. Неспекулативна нит поставља сигнал *pHStWO* да би обавијестила спекулативну нит да учита тражену ријеч у своју L1 кеш меморију у стању StSp. Такође, одговарајућа ријеч, која је у стању INV у L1 кеш меморији касније нити се ажурира и њено стање се поставља у стање StSp.

Ако нема тражене ријечи у неком од стања M, S или StM у било којој L1 кеш меморији раније или касније нити, тада се најближа ранија или најближа каснија нит, која има тражену ријеч у стању StS, бира као достављач. Такође, ако је достављач ранија нит, он поставља сигнал *pHStWO* да обавијести спекулативну нит која је поставила *BusRd* захтјев да учита тражену ријеч у стање StSp у L1 кеш меморију процесора који извршава спекулативну нит. Ако је достављач каснија нит, она поставља сигнал *sHStWO* да обавијести спекулативну нит да учита тражену ријеч у стање StSp. Одговарајућа ријеч, која је у стању INV у L1 кеш меморији раније спекулативне нити или касније спекулативне нити, се ажурира и стање јој се мења у стање StSp у зависности да ли је неки од сигнала *pHStWO*, *sHStWO* или *sHMSpWO* постављен, док неспекулативна нит мијења стање ријечи у StS у зависности да ли је постављен неки од сигнала *sHStWO* или *sHMSpWO*.

Ако тражена ријеч не постоји ни у једном од стања M, S, StM или StS у било којој L1 кеш меморији раније или касније нити, тада се ранија спекулативна нит, која има тражену ријеч у неком од спекулативно модификованих стања MSp, MSpF, MSSp или MSSpF, бира као достављач. Након тога се изабраном достављачу дозвољава да постави тражену ријеч на дијељену магистралу. Достављач поставља сигнал *pHMSpWO* да обавијести спекулативну нит која је поставила *BusRd* захтјев да учита тражену ријеч и да јој постави стање SSp у њеној локалној L1 кеш меморији. Поред тога, одговарајућа ријеч, која је у стању INV у кеш меморији касније нити се ажурира и стање јој се мења у стање SSp.

Ако се није догодио ниједан од претходних случајева, ранија спекулативна нит, који има тражену ријеч у стању SSp, се бира као достављач. Тражена ријеч се учитава у стању SSp у L1 кеш меморију процесора који извршава спекулативну нит. Одговарајућа ријеч у стању INV свих L1 кеш меморија каснијих нити, се ажурира и стање јој се мијења у стање SSp.

Ако се није догодио ниједан од горе поменутих случајева, тада се најближа ранија спекулативна нит која има тражену ријеч у неком од стања StMSp, StSpM, или StSp, бира као достављач. Тада се изабраном достављачу дозвољава да постави тражену ријеч на дијељену магистралу. Достављач поставља сигнал *pHStWO* да обавијести спекулативну нит која је поставила *BusRd* захтјев, да учита тражену ријеч и постави јој стање StSp у својој L1 кеш меморији. Одговарајућа ријеч која је у стању INV у L1 кеш меморији касније нити се ажурира и стање јој се мијења у стање StSp.

Ако не постоји достављач у тренутку када спекулативна нит постави захтјев за читање на дијељену магистралу, ријеч се учитава из дијељене L2 кеш меморије. Траженој ријечи стање се поставља у стање SSp у L1 кеш меморији спекулативне нити која је поставила захтјев за читање само ако ниједна каснија нит нема одговарајућу ријеч у неком од стања MSp, MSpF, MSSp или MSSpF. У супротном, каснија нит поставља сигнал *sHMSpWO* да обавијести спекулативну нит која је поставила захтјев за читање да постави стање тражене ријечи у стање StSp. Одговарајућа ријеч у стању INV у L1 кеш меморији касније нити, се ажурира и њено стање се мијења у стање SSp (сигнал *sHMSpWO* није постављен) или StSp (сигнал *sHMSpWO* је постављен).

Глава 8 Предлог SISC-WU протокола за меморијску комуникацију

Поред SISC-WI протокола заснованог на стратегији поништавања, у петој глави је начелно предложена и алтерантивна варијанта подршке за меморијску комуникацију, SISC-WU протокол, који је заснован на стратегији ажурирања. У овој глави ће бити дата прецизна спецификација овог протокола кроз опис стања, трансакција и сигнала на магистрали, акција и прелаза стања.

8.1 Инфраструктура SISC-WU протокола

У овој секцији ће бити представљена инфраструктура SISC-WU протокола која обухвата стања у којима се може наћи ријеч у кеш меморији, као и контролне сигнале и трансакције на магистрали током спекулативног извршавања.

8.1.1 Стања

У SISC-WU протоколу ријеч у L1 кеш меморији процесорских језгара током спекулативног извршавања може да се нађе у неком од сљедећих стања:

- M (*Modified*),
- S (*Shared*),
- SU (*Shared-Unsafe*),
- SUSp (*Shared-Unsafe-Speculative*),
- SSp (*Shared-Speculative*),
- MSp (*Modified-Speculative*),
- MSpF (*Modified-Speculative-Forwarded*),
- MSSp (*Modified-Shared-Speculative*),

- MSSpF (*Modified-Shared-Speculative-Forwarded*).

Значења стања M, S, SSp, MSp и MSpF у SISC-WU протоколу су иста као и значења ових стања у SISC-WI протоколу која су већ описана у секцији 7.1.1, па су овдје описана само значења стања специфичних за овај протокол.

Стање SU означава да је ријеч у одређеној L1 кеш меморији спекулативне нити модификована у односу на дијељену L2 кеш меморију. Ријеч није сигурна јер је произведена од стране спекулативне нити. Такође, друге L1 кеш меморије могу имати исту дијељену копију. Ријеч постаје сигурна, односно, стање ријечи се мијења у стање S када спекулативна нит која је произвела ту ријеч постане неспекулативна.

Стање SUSp означава да је ријеч претходно била у стању SSp у некој L1 кеш меморији спекулативне нити када је та нит ажурирала ријеч у односу на дијељену L2 кеш меморију. Ријеч није сигурна јер ју је произвела спекулативна нит. Такође, друге L1 кеш меморије могу имати исту дијељену копију. Ријеч постаје сигурна, односно стање ријечи се мијења у стање S, када спекулативна нит која је произвела ту ријеч постане неспекулативна.

Стање MSSp означава да је спекулативно модификована ријеч била у стању SSp, SU или SUSp када је спекулативна нит извршила упис у њу.

Стање MSSpF означава да је спекулативно модификована ријеч била у стању MSSp када је прослијеђена каснијој нити, на њен захтијев. Стање служи да се чува информација да је спекулативно модификована ријеч прослијеђена каснијој нити на њен захтјев, а притом је претходно била прочитана код нити произвођача или спекулативно ажурирана.

Четири бита по ријечи је потребно у локалној L1 кеш меморији за кодирање горе поменутих 9 стања у којима ријеч може да се нађе приликом спекулативног извршавања. Такође, пети бит назван *Stale* (St) је додат свакој ријечи у L1 кеш меморији. Он означава ријеч коју може користити текућа нит и која може бити прослијеђена каснијој нити на њен захтјев, али ту ријеч не смије да користи нит која ће бити иницирана у будућности на истом процесорском језгру.

8.1.2 Контролни сигнали и трансакције на магистрали

Процесорско језгро поставља два типа захтјева: читање (*PrRd*) и упис (*PrWr*). Дијељена магистрала која повезује L1 кеш меморије са дијељеном L2 кеш меморијом садржи контролне сигнале *BusRd* (*Bus-Read*), *BusUpd* (*Bus-Update*), *BusWB* (*Bus-Write Back*), *WB* (*Write-Back*), *ShAck* (*Shared-Acknowledge*), *Mask, Stale* и *Speculative-Modified* сигнале. Дефиниције контролних сигнала *BusRd, WB, Stale* и *Speculative-Modified* сигнала у SISC-WU протоколу су исте као и дефиниције ових сигнала у SISC-WI протоколу које су дате у секцији 7.1.2, па су овдје описани само сигнали специфични за овај протокол.

Сигнал *BusUpd* означава трансакцију која је посљедица неспекулативног или спекулативног уписа од стране процесора (*PrWr*) у меморијску ријеч. Када кеш контролер постави сигнал *BusUpd* на дијељену магистралу, сваки ранија нит, неспекулативна или спекулативна нит, ослушкује магистралу. Ако има одговарајућу ријеч у валидном стању, тај улаз се означава као застарјела копија тако што му се постави бит *St* на јединицу. У исто вријеме, свака касније нит ослушкује магистралу да би ажурирала одговарајућу ријеч ако ју је прочитала раније.

Сигнал *BusWB* означава трансакцију која генерише локални кеш контролер као одговор на трансакцију *BusRd* коју је иницирало неки други кеш контролер.

Сигнал *ShAck*, који је реализован као „ожичено ИЛИ“ коло, поставља спекулативна нит када чита тражену ријеч или када одговара на комуникацију иницирану од стране произвођача, претходне нити.

Сигнали *Mask* се постављају на дијељену магистралу када се догоди промашај приликом читања и када нит захтјева одговарајућу ријеч од своје најближе раније нити. Линије маске на дијељеној магистрали су реализоване као „ожичено ИЛИ“ коло. Свака ранија нит, са траженом ријечи у неком од стања S (St=0, St=1), M, MSp, MSpF, MSSp, MSSpF, SU (St=0, St=1), SUSp (St=1) или SSp (St=0, St=1) у својој L1 кеш меморији, одговара постављањем своје маске на одговарајуће линије магистрале. Такође, свака касније нит која има тражену ријеч у неком од стања M или S (St=0), одговара постављањем своје маске на магистралине линије за маску. Кеш контролер у свакој L1 кеш меморији може да детектује да ли је нека од ранијих или каснијих нити поставила своју маску на магистралу. Тада, механизам дистрибуиране арбитрације бира достављача за нит која је издала захтјев за читање. Изабрани достављач добија дозволу да постави тражену ријеч на магистралу. Дијељена L2 кеш меморија доставља податак нити која је поставила захтјев за читање, ако се тражена ријеч не налази ни у једној L1 кеш меморији.

Механизам дистрибуиране арбитрације примијењен код SISC-WU варијанте протокола је исти као и механизам примијењен код SISC-WI варијанте протокола.

8.2 Акције и прелази између стања у SISC-WU протоколу

У овој секцији ће бити представљене све акције које настају приликом поготка и промашаја при читању и упису, замјене у кеш меморији, као и одговарајући прелази између стања у SISC-WU протоколу који се одвијају код неспекулативне односно спекулативних нити током наведених акција. Поред тога ће бити представљене и акције које се одвијају током завршавања, поништавања и инијализовања нити током спекулативног извршавања. Као и у глави 7, све акције су илустроване одговарајућим дијаграмима стања, за које важе исте напомене у погледу нотације које су дате на почетку секције 7.2.

8.2.1 Погодак приликом читања

Неспекулативна нит. Ако неспекулативна нит постави захтјев за читање (*PrRd*) ријечи која је у неком од стања M или S (St=0 или St=1), захтјев се обрађује локално и ријеч остаје у истом стању. Дијаграм прелаза стања за неспекулативну нит је приказан на слици 8-1.

Спекулативна нит. Ако спекулативна нит постави захтјев за читање (*PrRd*) ријечи у неком од стања SSp (St=0 или St=1), MSp, MSSp, MSpF или MSSpF, захтјев се обрађује локално и ријеч остаје у истом стању.

Ако је ријеч у неком од стања S (St=0 или St=1) или SU (St=0 or St=1), мијења јој се стање у стање SSp (St=0 или St=1) приликом захтјева за читање, да би се означило да је ријеч прочитана спекулативно и да би се касније могло детектовати RAW нарушавање зависности. Ако је ријеч у стању SUSp (St=1) приликом спекулативног захтјева за читање, стање јој остаје исто.

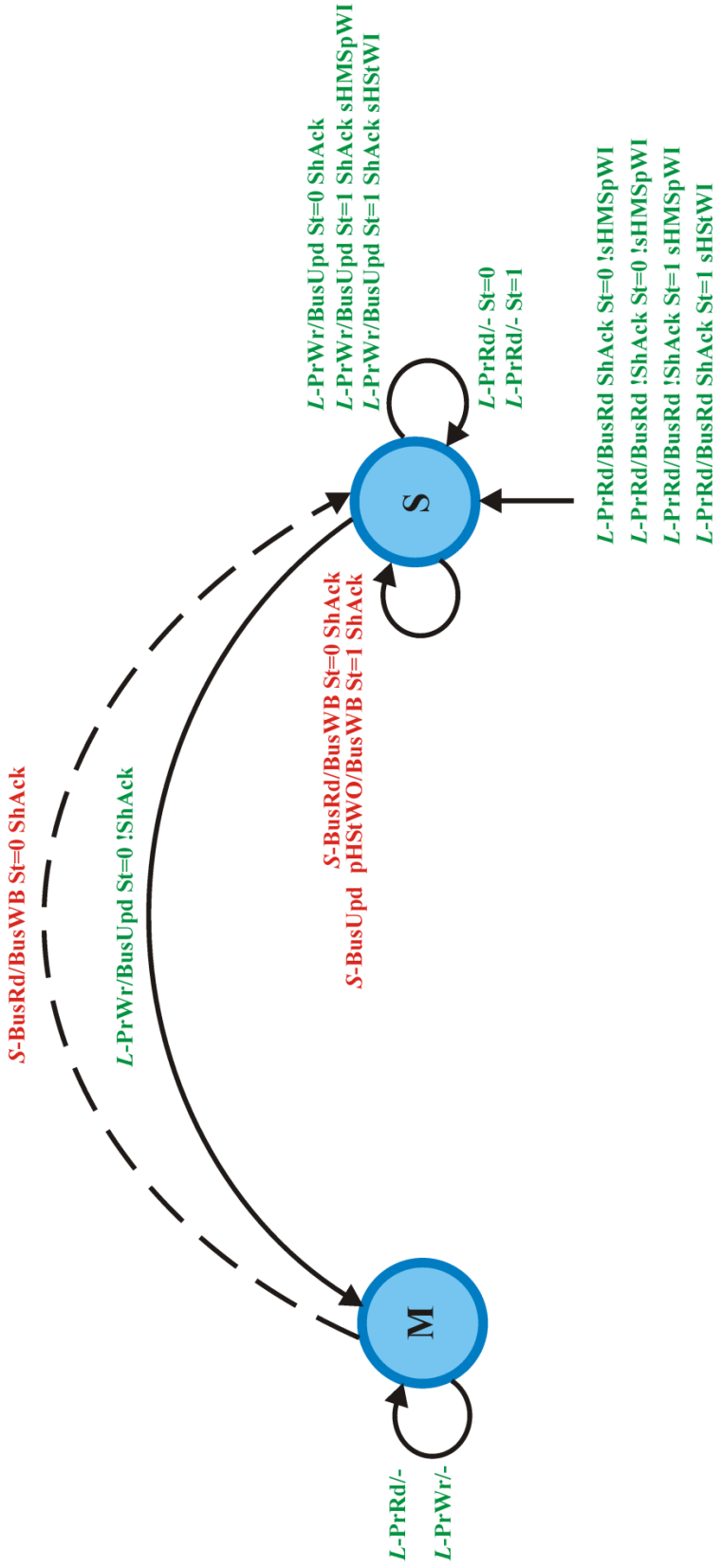
Даље, ако спекулативна нит изда захтјев за читање ријечи која је у стању M, ријеч мора да се упише у дијељену L2 кеш меморију да би се сачувала једина кеширана копија и, након тога, стање јој се мијења у стање SSp (St=0) да би се означило да је ријеч прочитана спекулативно.

Дијаграм прелаза стања за спекулативну нит за акције иницране од стране процесора је приказан на слици 8-2.

8.2.2 Промашај приликом читања

Неспекулативна нит. Захтјев за читање (*PrRd*) ријечи која није присутна у локалној L1 кеш меморији проузрокује промашај приликом читања. Овај догађај проузрокује међунитну комуникацију инициране од стране потрошача, неспекулативне нити, која извршава *BusRd* трансакцију на дијељеној магистрали. Такође, поред захтјева за читање на магистралу се поставља маска неспекулативне нити. Каснија нит који има тражену ријеч у неком од стања M или S (St=0) одговара постављањем своје маске на дијељену магистралу. Тада, механизам дистрибуиране арбитрације бира најближег достављача и дозвољава му да постави тражену ријеч на дијељену магистралу. Изабрани достављач поставља сигнал *ShAck* да обавијести неспекулативну нит да постави стање тражене ријечи у стање S (St=0) приликом учитавања у L1 кеш меморију. На страни достављача, ако је ријеч била у стању M, ријеч се уписује у L2 кеш меморију да би се сачувала једина најскорије промијењена кеширана копија и, након тога, стање јој се мијења у стање S (St=0). У случају да је тражена ријеч била у стању S (St=0) на страни достављача остаје у истом стању.

Ако не постоји ниједна каснија нит која може бити достављач са траженом ријечи у неком од стања M или S (St=0) и ако каснија нит има ријеч у стању S (St=1), она одговара постављањем своје маске на дијељену магистралу.



Слика 8-1. Дијаграм прелаза стања за неспекулативну нит - акције иницране од стране процесора и магистрале

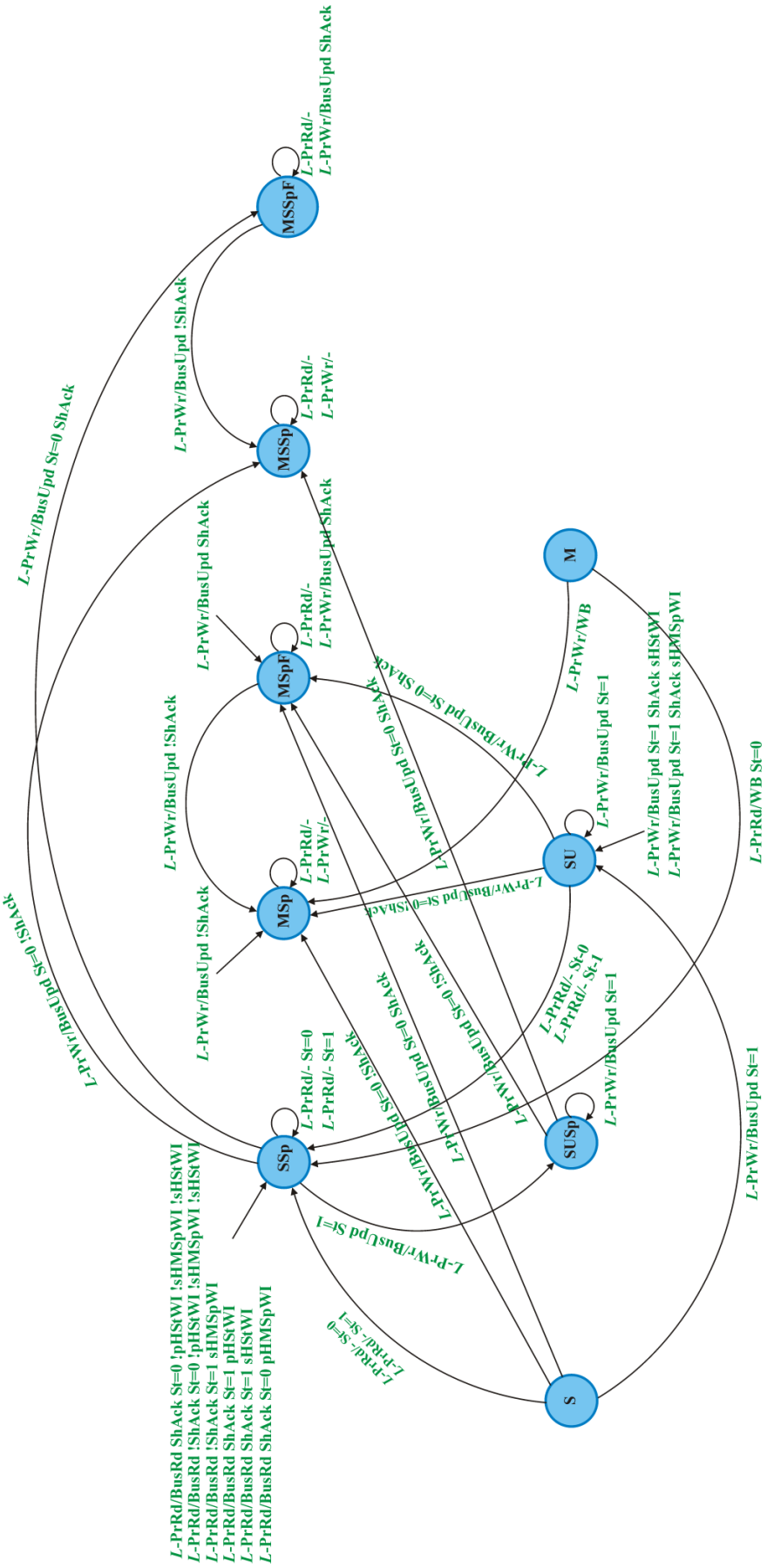
Механизам дистрибуиране арбитрације бира достављача и дозвољава му да постави тражену ријеч на дијељену магистралу. Изабрани достављач поставља сигнале *ShAck* и *sHStWO* да би обавијестио потрошача, неспекулативну нит, да учита тражену ријеч у стање S и да постави St бит тражене ријечи на јединицу у својој L1 кеш меморији.

Ако не постоји ниједна каснија нит која може бити достављач са траженом ријечи у неком од стања M или S (St=0 или St=1), неспекулативни захтјев за читање задовољава дијељена L2 кеш меморија. Стање тражене ријечи се поставља у S (St=0) приликом учитавања у L1 кеш меморију процесора који извршава неспекулативну нит под условом да не постоји ниједна спекулативна нит која посједује тражену ријеч у неком од стања MSp, MSpF, MSSp или MSSpF. У супротном, каснија нит поставља сигнал *sHMSpWO* да обавијести неспекулативну нит да постави стање тражене ријечи у стање S (St=1). Неспекулативна нит може да користи локално ријеч са постављеним битом St на јединицу и може да је прослиједи сљедбенику на његов захтјев, али нити инициране на истом процесорском језгру у будућности не могу да је користе. Пошто се тражена ријеч чита из дијељене L2 кеш меморије, сигнал *ShAck* се не поставља.

Дијаграм прелаза стања за неспекулативну нит и акције инициране од стране процесора и одговарајуће акције на магистрали је приказан на слици 8-1.

Спекулативна нит. Ако спекулативна нит постави захтјев за читање и догоди се промашај у локалној L1 кеш меморији, захтјев се поставља на дијељену магистралу заједно са маском спекулативне нити. Промашај приликом читања проузрокује међунитну комуникацију иницирану од стране нити потрошача. Могућег достављача бира механизам дистрибуиране арбитрације између неспекулативне нити или најближе спекулативне нити које имају тражену ријеч у стању M. У супротном, најближа нит, било неспекулативна или спекулативна, која има тражену ријеч у стању S (St=0) се бира као достављач. У оба случаја, изабрани достављач поставља сигнал *ShAck* да обавијести потрошача, спекулативну нит, да постави стање ријечи у стање SSp (St=0) приликом учитавања у своју L1 кеш меморију.

Ако не постоји ријеч у неком од стања M или S (St=0) у било којој L1 кеш меморији претходника или сљедбеника, тада се најближи претходник, неспекулативна или спекулативна нит, или најближи сљедбеник, спекулативна нит, који има тражену ријеч у стању S (St=1) бира као достављач. Такође, ако је достављач, ранија нит, неспекулативна или спекулативна нит, она поставља сигнал *pHStWO* да обавијести спекулативну нит која је покренула трансакацију *BusRd* да постави *stale* бит на јединицу (St=1) за тражену ријеч у својој L1 кеш меморији. Ако је достављач каснија спекулативна нит, она поставља сигнал *sHStWO* да обавијести спекулативну нит која је покренула трансакцију *BusRd* да постави *stale* бит на јединицу (St=1) за тражену ријеч у својој L1 кеш меморији.



Слика 8-2. Дијаграм прелаза стања за неспекулативну нит - акције иницране од стране процесора

Такође, у оба случаја, изабрани достављач поставља сигнал ShAck да обавијести нит потрошача да постави стање ријечи у SSp (St=1) приликом учитавања у L1 кеш меморију.

Ако не постоји тражена ријеч у неком од стања M или S (St=0 или St=1) у било којој L1 кеш меморији раније или касније нити, тада се као достављач бира ранија спекулативна нит која има тражену ријеч у неком од спекулативно модификованих стања MSp, MSSp, MSpF или MSSpF. Поменути могући достављачи одговарају постављањем маске на дијељену магистралу и тада механизам дистрибуиране арбитрације бира најближег одговарајућег достављача. Изабрани достављач добија дозволу да постави тражену ријеч на дијељену магистралу и поставља сигнале pHMSpWO и ShAck да би обавијестио нит потрошача да постави стање тражене ријечи у стање SSp (St=0) приликом учитавања у L1 кеш меморију.

Ако не постоји тражена ријеч у неком од претходно поменутих стања, тада се ранија спекулативна нит бира као достављач ако има тражену ријеч у неком од стања SU (St=0 или St=1), SUSp (St=0 или St=1) или SSp (St=0 или St=1). Изабрани достављач добија дозволу да постави тражену ријеч на дијељену магистралу и поставља сигнал *ShAck* да обавијести нит потрошача да постави стање ријечи у стање SSp (St=0 или St=1) приликом учитавања у своју L1 кеш меморију. У случају када тражена ријеч на страни достављача има постављен St бит на јединицу, достављач поставља сигнал *pHStWO* да обавијести нит потрошача да учита ријеч у своју L1 кеш меморију са битом St постављеним на јединицу.

Ако не постоји достављач у тренутку када нит потрошач постави захтјев за читање на дијељену магистралу, тада дијељена L2 кеш меморија доставља тражену ријеч. Њено стање се поставља у стање SSp (St=0) приликом учитавања у L1 кеш меморију спекулативне нити која је поставила захтјев за читање, само ако ниједна каснија нит нема одговарајућу ријеч у неком од стања SUSp (St=1), SU (St=1), MSp, MSpF, MSSp или MSSpF. У супротном, каснија нит која има тражену ријеч у неком од стања MSp, MSpF, MSSp или MSSpF, поставља сигнал *sHMSpWO*, док каснија нит која има тражену ријеч у неком од стања SUSp (St=1) или SU (St=1) поставља сигнал *sHStWO* да би обавијестио спекулативну нит која је поставила захтјев за читање да постави стање ријечи SSp (St=1) приликом учитавања у локалну кеш меморију.

Промјене стања тражене ријечи на страни достављача обављају се на сљедећи начин:

- ако је тражена ријеч у стању M, податак се уписује у дијељену L2 кеш меморију да би се сачувала једина најскорије промијењена копија. Траженој ријечи се стање мијења у стање S на страни достављача.
- ако је тражена ријеч у стању S (St=0, St=1), SU (St=0, St=1), SUSp (St=0, St=1), SSp (St=0, St=1), MSpF или MSSpF на страни достављача, стање јој остаје исто.

- ако је тражена ријеч у стању MSp на страни достављача, стање јој се мијења у стање MSpF.
- ако је тражена ријеч у стању MSSp на страни достављача, стање јој се мијења у стање MSSpF.

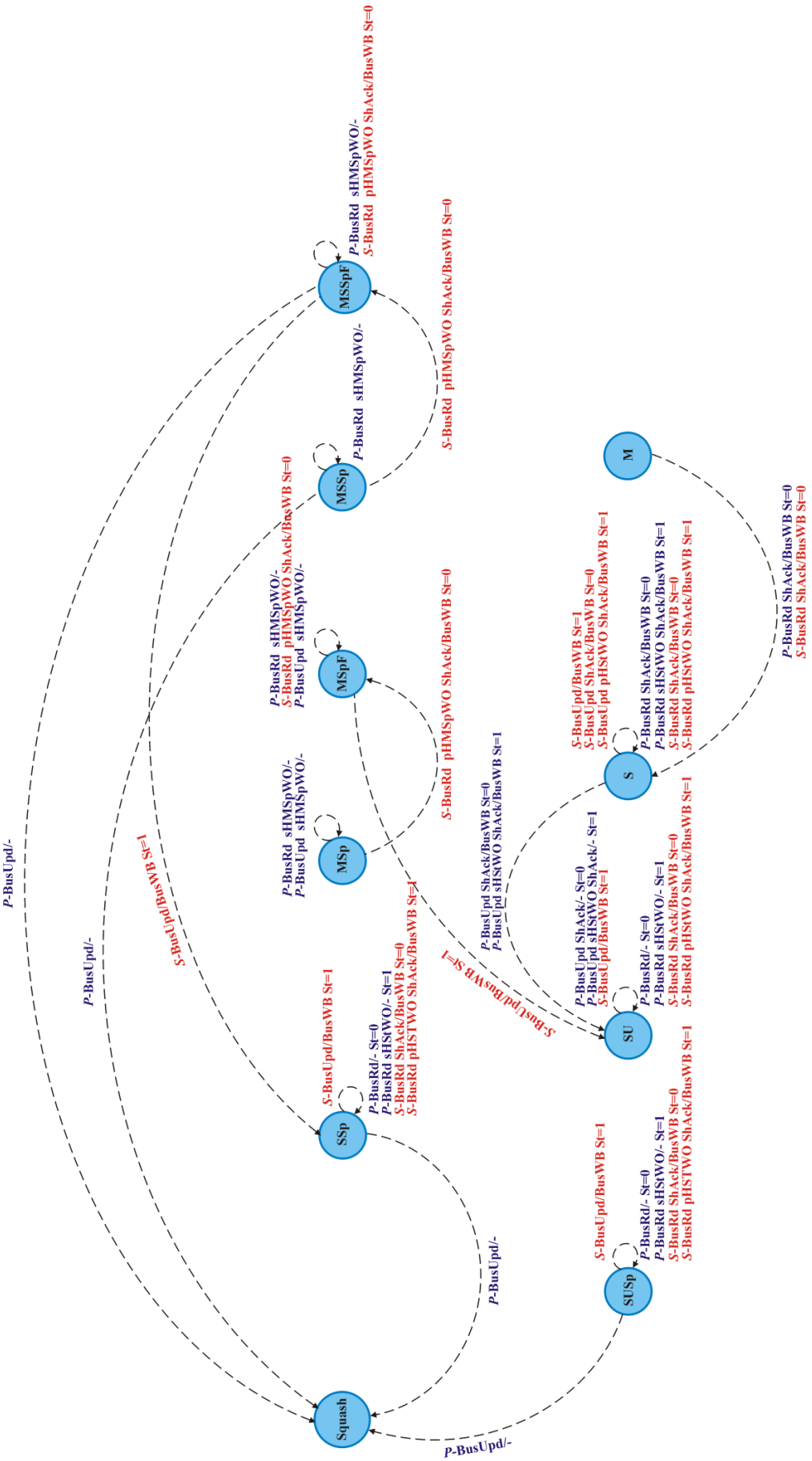
Дијаграм прелаза стања за спекулативну нит за акције иницране од стране процесора је приказан на слици 8-2, док су акције иницране са магистрале приказане на слици 8-3.

8.2.3 Погодак приликом уписа

Неспекулативна нит. Ако је ријеч у стању M када неспекулативна нит постави захтјев за упис (*PrWr*), ријеч се ажурира и стање јој остаје исто. Нема потребе за извршавањем одложеног уписа, пошто је неспекулативна нит поставила захтјев за упис. Модификована ријеч ће бити потврђена касније и податак неће бити изгубљен. У овом случају трансакција *BusUpd* се не издаје на дијељеној магистрали.

Ако је ријеч у стању S (St=0 или St=1) када неспекулативна нит постави захтјев за упис, трансакцијом *BusUpd* на магистрали се иницира комуникација од стране произвођача. Сваки кеш контролер који извршава каснију нит ослушкује дијељену магистралу и провјерава одговарајући улаз у својој L1 кеш меморији. Ако је ријеч код касније нити у стању S (St=0 или St=1), ријеч се ажурира и стање је јој се мијења у стање SU (St=0 или St=1). Ако је ријеч код касније нити у стању SU (St=0 или St=1), ријеч се ажурира и стање јој остаје исто. Поред тога, нит потрошач поставља сигнал *ShAck* да обавијести нит произвођача о успјешном ажурирању и модификована ријеч на страни произвођача остаје у истом стању S (St=0 или St=1). Такође, ажурира се и одговарајући улаз у дијељеној L2 кеш меморији. Нарушавање RAW зависности се детектује ако је одговарајући улаз у L1 кеш меморији процесорског језгра које извршава каснију нит у неком од стања SSp (St=0 или St=1), SUSp (St=0 или St=1), MSSp, MSSpF или SSp (St=0). Тада се каснија нит, као и све наредне нити, поништавају и рестартују. Ако каснија нит има тражену ријеч у неком од стања S или SU са битом St постављеним на јединицу, она поставља сигнал *sHStWO* да обавијести произвођача да постави бит St тражене ријечи на јединицу у својој L1 кеш меморији.

Ако се ријеч налази у неком од стања MSp или MSpF у L1 кеш меморији касније нити, остаје у истом стању и каснија нит поставља сигнал *sHMSpWO* да обавијести неспекулативну нит да промијени стање ријечи у S (St=1).



Слика 8-3. Дијаграм прелаза стања за спекулативну нит - акције иницране од стране магистрале

Ако ниједна каснија нит нема одговарајућу ријеч у својој L1 кеш меморији, неспекулативна нит ажурира ријеч и мијења јој стање у стање M пошто сигнал *ShAck* није постављен.

Дијаграм прелаза стања за неспекулативну нит и акције иницране од стране процесора и одговарајуће акције на магистрали је приказан на слици 8-1.

Спекулативна нит. Ако је ријеч у стању M када спекулативна нит постави захтјев за упис, трансакција *WB* се покреће да би се осигурало да се једина најскорија модификована копија сачува приликом спекулативне модификације. Стање ријечи се код произвођача мијења у MSp.

Ако је ријеч у стању S (St=0) или SU (St=0) када спекулативна нит постави захтјев за упис, трансакцијом *BusUpd* се на дијељеној магистрали покреће комуникација иницирана од стране произвођача. Сваки кеш контролер процесорског језгра које извршава каснију нит ослушкује магистралу и провјерава одговарајући улаз у својој L1 кеш меморији. Ако је одговарајући улаз у L1 кеш меморији касније нити у неком од стања SSp (St=0) или SUSp (St=0), зависност међу подацима се детектује и та каснија нит, као и све наредне нити, се поништавају и рестартују. У случају да је ријеч у L1 кеш меморији касније нити у стању S (St=0), ријеч се ажурира и стање јој се мијења у стање SU (St=0). При томе, нит потрошач поставља сигнал *ShAck* да би обавијестио произвођача да је прихватио податак, а након тога се одговарајућа ријеч на страни произвођача ажурира и мијења јој се стање у стање MSpF. Ако се не постави сигнал *ShAck* произвођач ажурира ријеч и стање јој се мијења у стање MSp.

Такође, сваки кеш контролер процесорског језгра које извршава претходника, неспекулативна или спекулативна нит, ослушкује дијељену магистралу приликом трансакације *BusUpd* да би означио податак у својој кеш меморији који више не могу да користе нити које ће у будућности бити стартоване на истом процесорском језгру. Из тог разлога, ако је ријеч у стању S (St=0), SSp (St=0) или SUSp (St=0) стање јој остаје исто и бит St јој се поставља на јединицу.

Ако је ријеч у стању SSp (St=0) или SUSp (St=0) када спекулативна нит постави захтјев за упис, *BusUpd* трансакција на дијељеној магистрали покреће комуникацију иницирану од стране произвођача. Сваки кеш контролер процесорских језгара која извршавају касније нити ослушкује дијељену магистралу и провјерава одговарајући улаз у својој L1 кеш меморији. Ако је ријеч у неком од стања SSp (St=0) или SUSp (St=0) у L1 кеш меморији касније нити, детектује се RAW нарушавање зависности и та каснија нит, као и све наредне нити, се поништавају и рестартују. У случају када је ријеч у L1 кеш меморији касније нити у стању S (St=0), ријеч се ажурира и стање јој се мијења у стање SU (St=0), а ако је у стању SU (St=0) остаје у истом стању. Поред тога, нит потрошач поставља сигнал *ShAck* да обавијести нит произвођача да је прихватила податак. Тада се код нити произвођача одговарајући улаз ажурира и стање му се мијења у стање MSSpF. У супротном, сигнал *ShAck* се не поставља и нит произвођач стање

одговарајуће ријечи мијења у стање MSSp. Сваки кеш контролер процесорских језгара која извршавају раније нити ослушкује дијељену магистралу приликом *BusUpd* трансакције да би означило податак у својој кеш меморији који не смију да користе нити које ће у будућности бити стартоване на истом процесорском језгру. Из тог разлога, ранија нит може поставити бит St на јединицу за одговарајућу ријеч у својој L1 кеш меморији на сљедећи начин:

- ако је ријеч у стању S (St=0), SSp (St=0), SU (St=0) или SUSp (St=0), стање јој остаје непромијењено и поставља јој се бит St на јединицу.
- ако је ријеч у стању MSpF, стање јој се мијења у стање SU (St=1).
- ако је ријеч у стању MSSpF, стање јој се мијења у стање SSp (St=1).

Ако је ријеч у стању MSp или MSSp када спекулативна нит постави захтјев за упис, ријеч се ажурира и остаје у истом стању. Трансакција *BusUpd* се не покреће на магистрали као ни комуникација иницирана од стране произвођача.

Ако је ријеч у стању MSpF или MSSpF када спекулативна нит постави захтјев за упис, трансакцијом *BusUpd* на магистрали се покреће комуникација иницирана од стране произвођача. У случају да је ријеч у L1 кеш меморији касније нити у неком од стања SSp (St=0) или SUSp (St=0) детектује се RAW нарушавање зависности међу подацима. Та спекулативна нит, као и све наредне нити, се поништавају и рестартују, док се стање ријечи код произвођача мијења у стање MSp ако се не постави сигнал *ShAck*. Ако је одговарајућа ријеч у стању SU (St=0) код касније нити, ријеч ће бити ажурирана и стање јој остаје исто. Такође, нит потрошач поставља сигнал *ShAck* да би обавијестила нит произвођача да стање ажуриране ријечи, MSpF, остави исто у својој L1 кеш меморији. Ријеч у кеш L1 меморији процесорског језгра које извршава ранију нит остаје у истом стању ако је у неком од стања S (St=1), SU (St=1), SUSp (St=1) или SSp (St=1).

Ако је ријеч у стању S (St=1) или SU (St=1) када спекулативна нит постави захтјев за упис, ријеч се ажурира и стање ријечи се мијења у SU (St=1) односно остаје непромијењено, респективно. Трансакцијом *BusUpd* се на магистрали покреће комуникација иницирана од стране произвођача. Сваки кеш контролер процесорског језгра које извршава каснију нит ослушкује магистралу и провјерава одговарајући улаз у својој L1 кеш меморији. Ако се ријеч налази у стању S (St=1), она ће бити ажурирана и стање јој се мијења у стање SU (St=1), а ако је у стању SU (St=1), стање јој остаје исто. Нит потрошач поставља сигнал *ShAck* да би обавијестила нит произвођача да је прихватила податак. У случају да је ријеч у неком од стања SSp (St=1), SUSp (St=1), MSSp, MSSpF или SSp (St=0), та спекулативна нит, као и све наредне нити, се поништавају и рестартују. Ако је одговарајућа ријеч у неком од стања MSp или MSpF у L1 кеш меморији касније нити, ријеч остаје у истом стању и та спекулативна нит поставља сигнал *sHMSpWO* да обавијести спекулативну нит која је поставила захтјев за упис да промијени стање ријечи у SU (St=1). Додатно, ако је ријеч у L1 кеш меморији процесорских језгара која извршавају раније нити у неком од стања S (St=1), SU

(St=1), SUSp (St=1) или SSp (St=1), стање јој остаје исто. Ранија и каснија нит са одговарајућом ријечи која има постављен St бит на јединицу, постављају сигнале *pHStWO* и *sHStWO*, респективно, да би обавијестили произвођача да постави бит St на јединицу за ажурирану ријеч. Ако не постоји одговарајућа ријеч у L1 кеш меморији раније или касније нити, стање ажуриране ријечи код произвођача остаје у SU стању са St битом постављеним на јединицу.

Ако је ријеч у стању SSp (St=1) или SUSp (St=1) када спекулативна нит постави захтјев за упис, ријеч се ажурира и стање ријечи се мијења у стање SUSp (St=1) односно остаје непромијењено, респективно. Трансакцијом *BusUpd* се на магистрали покреће комуникација иницирана од стране произвођача. Сваки кеш контролер процесорског језгра које извршава каснију нит ослушкује магистралу и провјерава одговарајући улаз у својој L1 кеш меморији. Ако се ријеч налази у стању S (St=1), она ће бити ажурирана и стање јој се мијења у SU (St=1), а ако је у стању SU (St=1), стање јој остаје исто. Нит потрошач поставља сигнал *ShAck* да би обавијестила нит произвођача да је прихватила податак. У случају да је ријеч у неком од стања SSp (St=1), SUSp (St=1), MSSp, MSSpF или SSp (St=0), та спекулативна нит, као и све наредне нити, се поништавају и рестартују. Ако је одговарајућа ријеч у неком од стања MSp или MSpF у L1 кеш меморији касније нити, ријеч остаје у истом стању и та спекулативна нит поставља сигнал *sHMSpWO* да обавијести спекулативну нит која је поставила захтјев за упис да промијени стање ријечи у стање SU (St=1). Додатно, ако је ријеч у L1 кеш меморији процесорских језгара која извршавају претходнике у неком од стања S (St=1), SU (St=1), SUSp (St=1) или SSp (St=1), стање јој остаје исто. Ранија и каснија нит са одговарајућом ријечи која има постављен St бит на јединицу, постављају сигнале *pHStWO* и *sHStWO*, респективно, да би обавијестили произвођача да постави бит St на јединицу за ажурирану ријеч. Ако не постоји одговарајућа ријеч у L1 кеш меморији раније или касније нити, стање ажуриране ријечи код произвођача остаје у стању SUSp са St битом постављеним на јединицу.

Дијаграм прелаза стања за спекулативну нит приликом поготка при упису за акције инициране од стране процесора је приказан на слици 8-2, док су акције инициране са магистрале приказане на слици 8-3.

8.2.4 Промашај приликом уписа

Неспекулативна нит. Захтјев за упис (*PrWr*) у ријеч која нија присутна у локалној L1 кеш меморији процесорског језгра које извршава неспекулативну нит третира се као промашај приликом уписа. Прво, мора да се провјери да ли постоји слободан улаз у одговарајућој L1 кеш меморији. Ако не постоји слободан улаз, тада мора да се изврши замјена. Када постоји слободан улаз податак се уписује у L1 кеш меморију, а трансакција *BusUpd* покреће на дијељеној магистрали комуникацију иницирану од стране произвођача. Сваки кеш контролер

процесорског језгра које извршава каснију нит ослушкује магистралу и провјерава одговарајући улаз у својој L1 кеш меморији.

Ако је одговарајућа ријеч у стању S (St=0) код касније нити, ријеч се ажурира и мијења јој се стање у SU (St=0). Поред тога, нит потрошач поставља сигнал *ShAck* да обавијести нит произвођача да је ажурирала ријеч. Такође, ријеч се уписује у L2 кеш меморију, док се стање ријечи код произвођача мијења у стање S (St=0). Међутим, ако је одговарајући улаз у L1 кеш меморији процесорског језгра које извршава каснију нит у неком од стања SSp (St=0), SUSp (St=0), MSSp или MSSpF, детектује се RAW нарушавање зависности. Тада се та спекулативна нит, као и све наредне нити, поништавају и рестартују.

Ако је одговарајућа ријеч у стању S (St=1) код касније нити, ријеч се ажурира и стање јој се мијења у стање SU (St=1). Поред тога, нит потрошач поставља сигнал *sHStWO* да обавијести нит произвођача да промијени стање ријечи у стање S (St=1). Такође, ако је одговарајућа ријеч у L1 кеш меморији процесорског језгра које извршава каснију нит, у неком од стања SSp (St=1), SUSp (St=1), MSSp, MSSpF или SSp (St=0), та нит, као и све наредне нити, се поништавају и рестартују.

Ако је одговарајућа ријеч у неком од стања MSp или MSpF код касније нити, та нит поставља сигнал *sHMSpWO* да обавијести нит произвођача да промијени стање ријечи у локалној L1 кеш меморији у S са битом St постављеним на јединицу да означи застарјелу вриједност. Ако одговарајућа ријеч не постоји ни у једној L1 кеш меморији касније нити и сигнали *ShAck, Speculative Modified* и *Stale* нијесу постављени, тада неспекулативна нит поставља стање нове ријечи у својој L1 кеш меморији у M.

Дијаграм прелаза стања за неспекулативну нит и акције иницране од стране процесора и одговарајуће акције на магистрали је приказан на слици 8-1.

Спекулативна нит. Захтјев за упис спекулативне нити (*PrWr*) у ријеч која није присутна у локалној L1 кеш меморији одговарајућег процесорског језгра проузрокује промашај приликом уписа. Као и у претходном случају, прво мора да се пронађе слободан улаз у одговарајућој L1 кеш меморији. Тада се нови податак уписује у L1 кеш меморију, а трансакција *BusUpd* се покреће на магистрали као комуникација иницирана од стране нити произвођача.

Сваки кеш контролер процесорског језгра које извршава каснију нит ослушкује дијељену магистралу и провјерава одговарајући улаз у својој L1 кеш меморији. Ако је одговарајућа ријеч у стању S (St=0), ријеч се ажурира и стање јој се мијења у SU (St=0). Поред тога, нит потрошач поставља сигнал *ShAck* да обавијести нит произвођача о успјешном ажурирању и тада се одговарајућа ријеч код произвођача ажурира и стање јој се мијења у стање MSpF. Ако је одговарајући улаз у L1 кеш меморији процесорског језгра које извршава каснију нит, у стању SSp (St=0), детектује се RAW нарушавање зависности. Тада се та спекулативна нит, као и све наредне нити, поништавају и рестартују.

Ако је одговарајућа ријеч у стању S (St=1) код касније нити, ријеч се ажурира и стање јој се мијења у SU (St=1). Поред тога, нит потрошач поставља сигнале *ShAck* и *sHStWO* да обавијести нит произвођача да постави стање нове ријечи у стање SU (St=1). Ако је одговарајући улаз у L1 кеш меморији процесорског језгра које извршава каснију нит, у неком од стања SSp (St=1), SUSp (St=1), MSSp, MSSpF или SSp (St=0), детектује се нарушавање RAW зависности и та нит, као и све наредне нити, се поништавају и рестартују.

Ако је одговарајућа ријеч у неком од стања MSp или MSpF код касније нити, та нит поставља сигнал *sHMSpWO* да обавијести нит произвођача да промијени стање ријечи у својој локалној L1 кеш меморији у стање SU са битом St постављеним на јединицу, да би се означила застарјела вриједност.

Ако не постоји одговарајући улаз у било којој L1 кеш меморији касније нити, тада спекулативни произвођач мијења стање ријечи у стање MSp у својој L1 кеш меморији.

Сваки кеш контролер процесорског језгра које извршава ранију нит, неспекулативну или спекулативну нит, ослушкује дијељену магистралу и провјерава одговарајући улаз у својој L1 кеш меморији. Стање ријечи у L1 кеш меморији раније нити се мијења на следеће начине:

- ако је ријеч у стању M, она мора да се упише у дијељену L2 кеш меморију да би се сачувала једина најскорије промијењена копија и стање јој се мијења у стање S (St=1).
- ако је ријеч у стању S (St=0), стање јој се мијења у стање S (St=1).
- ако је ријеч у стању S (St=1), SSp (St=1), SU (St=1), SUSp (St=0), SUSp (St=1), стање јој остаје непромијењено.
- ако је ријеч у стању SSp (St=0), стање јој се мијења у стање SSp (St=1).
- ако је ријеч у стању SU (St=0), стање јој се мијења у стање SU (St=1).
- ако је ријеч у стању MSp, стање јој се мијења у стање SU (St=1).
- ако је ријеч у стању MSpF, стање јој се мијења у стање SU (St=1).
- ако је ријеч у стању MSSp, стање јој се мијења у стање SUSp (St=1).
- ако је ријеч у стању MSSpF, стање јој се мијења у стање SUSp (St=1).

Дијаграм прелаза стања за спекулативну нит приликом промашаја при упису за акције иницране од стране процесора је приказан на слици 8-2, док су акције иницране са магистрале приказане на слици 8-3.

8.2.5 Замјена ријечи у кеш меморији

Алгоритам замјене у SISC-WU протоколу је заснован на истим принципима као и код SISC-WI протокола. Алгоритам прво узима у обзир стање у којем се ријеч налази, умјесто временске историје приступа. Такође, он дозвољава спекулативној нити да замијени из њене L1 кеш меморије модификовану и потврђену ријеч као једину најскорије промијењену копију која је спекулативно била прочитана или промијењена. Чисте ријечи које још нису биле спекулативно

прочитане или промјењене могу, такође, да се замијене. Из тог разлога, заустављање процесора ће бити избјегнуто и спекулативно извршавање ће се наставити. Међу ријечима које су у истом стању може да се примијени неки други алгоритам замјене, нпр., FIFO, LRU или случајни избор, као секундарни критеријум за одабир ријечи за замјену.

Први кандидат за избацивање ријечи из кеш меморије је ријеч која се налази у стању S (St=1) чија вриједност постоји и у главној меморији и/или у кеш меморији неке од других нити. Ако не постоји ријеч у стању S (St=1) у кеш меморији, тада се ријеч у стању S (St=0) бира за избацивање из кеш меморије. На крају, ако ниједан од наведених случајева није задовољен, алгоритам замјене за избацивање бира ријеч која се налази у стању M, ако постоји.

Ако спекулативна ријеч у неком од стања SSp, SU, SUSp, MSp, MSpF, MSSp или MSSpF мора бити замијењена у L1 кеш меморији спекулативне нити, SISC-WU протокол третира ово као недозвољену ситуацију која проузрокује прекид спекулације и поновно извршавање нити. Дозвољавање замјене спекулативне ријечи може утицати на исправност спекулације јер уклањање спекулативне ријечи проузрокује да SISC-WU протокол више не може да прати могућа нарушавања зависности по подацима између нити.

Ако ријеч у стању S (St=0 или St=1) треба да буде замјењена у L1 кеш меморији било неспекулативне било спекулативне нити, FIFO, LRU или случајни избор могу да се користе као секундарни критеријум за избор ријечи за замјену међу свим ријечима у стању S (St=0 или St=1).

Ако ријеч у стању M треба да се замијени у L1 кеш меморији неспекулативне нити, та ријеч се уписује у L2 кеш меморију да се сачува једина најскорије промијењена копија, а након тога може да се замјени у L1 кеш меморији. Такође, као и у претходним случајевима, ако постоји више ријечи у стању M у локалној L1 кеш меморији, тада, FIFO, LRU или случајан избор могу да се користе као секундарни критеријум за избор ријечи за замјену међу ријечима у стању M.

Акције приликом замјене ријечи код неспекулативних и спекулативних нити су приказане на сликама 8-4 и 8-5, респективно.

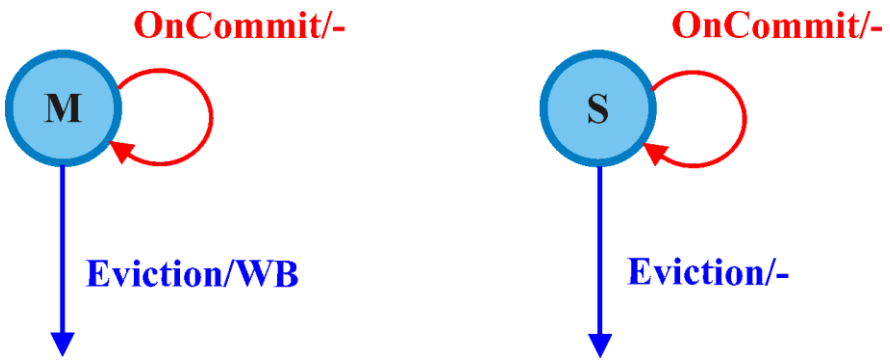
8.2.6 Завршавање, поништавање и иницијализовање нити

Нити се завршавају у секвенцијалном поретку, па нит прво мора да постане неспекулативна прије него што се заврши, а затим се иницира нова нит на истом процесору.

Завршетак неспекулативне нити. Када се неспекулативна нит завршава, свака ријеч у стању M у локалној L1 кеш меморији процесора који извршава ту нит, остаје у истом стању. Због тога, модификоване и потврђене верзије могу да остану у кеш меморији након што се заврши нит која је посљедња уписала у ту ријеч. На тај начин се избјегава већа количина саобраћаја на магистрали приликом

завршетка нити што може проузроковати повећање времена потребног да се нит заврши, као и повећање времена потребног да се иницијализује нова нит. Потврђена ријеч се уписује у дијељену L2 кеш меморију када јој се приступи сљедећи пут било због локалног читања или уписа, било због удаљеног читања или уписа од стране других нити.

Акције приликом завршетка неспекулативне нити су приказане на слици 8-4.



Слика 8-4. Дијаграм прелаза стања за неспекулативну нит током замјене ријечи и приликом завршетка нити: *Eviction* – замјена ријечи у кеш меморији; *OnCommit* – комитовање нити

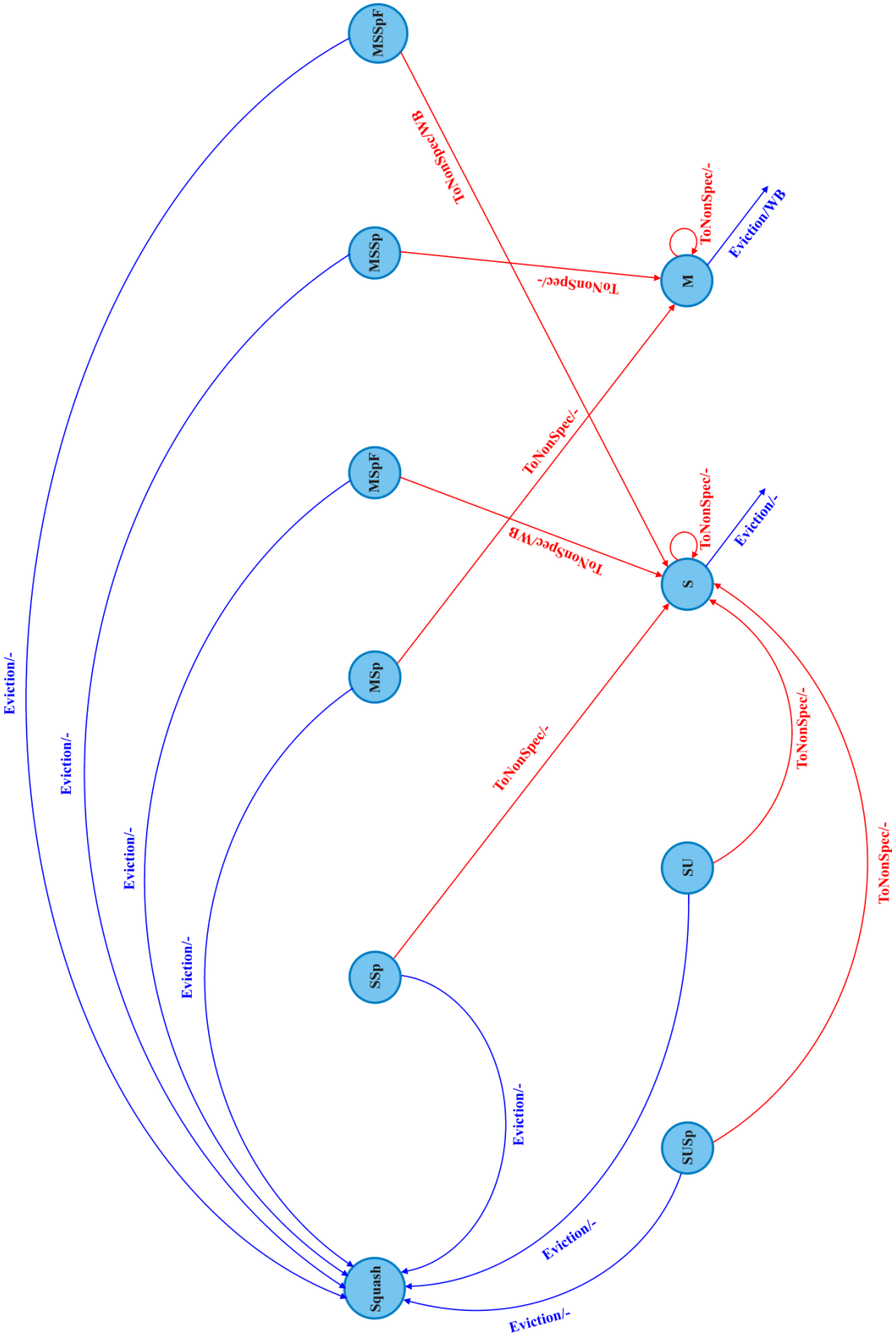
Завршетак спекулативне нити. Када спекулативна нит постане неспекулативна сљедеће промјене стања ријечи се дешавају:

- ако је ријеч у стању MSp или MSSp, стање јој се мијења у стање M.
- ако је ријеч у стању SSp (St=0 или St=1), SU (St=0 или St=1) или SUSp (St=0 или St=1), стање јој се мијења у стање S (St=0 или St=1).
- ако је ријеч у стању MSpF или MSSpF, стање јој се мијења у стање S (St=0).

У супротном, ако је ријеч у неком од стања S или M, ријеч остаје у истом стању када спекулативна нит постане неспекулативна. Тада та нит може да настави са својим завршетком и нова нит може да се стартује на истом процесору.

Акције приликом завршетка спекулативне нити су приказане на слици 8-5.

Иницирање нове спекулативне нити. Приликом иницијализације нове спекулативне нити, све ријечи које су у стању S (St=1) у локалној L1 кеш меморији процесорског језгра где се нова нит иницира, морају да се пониште. Ријечи у неком од стања M или S (St=0) остају у L1 кеш меморији приликом иницијализације нове нити.



Слика 8-5. Дијаграм прелаза стања за спекулативну нит током замјене ријечи и приликом преласка у неспекулативно извршавање: *Eviction* – замјена ријечи у кеш меморији; *ToNonSpec* – нит постаје неспекулативна

Поништавање спекулативне нити. Када се поништава спекулативна нит, све ријечи у неком од стања MSp, MSpF, MSSp или MSSpF, спекулативно учитане ријечи у неком од стања SSp (St=0) или SUSp (St=0), ријечи у стању SU (St=0), или застарјеле ријечи у неком од стања S (St=1), SU (St=1), SUSp (St=1) или SSp (St=1), морају да се пониште у одговарајућој L1 кеш меморији. Остале ријечи у неком од стања M или S (St=0) се задржавају у кеш меморији приликом поништавања нити. На овај начин SISC-WU протокол елиминише проблем празне L1 кеш меморије приликом почетка извршавања нити, јер задржава потврђене и непромијењене ријечи у L1 кеш меморији, што смањује саобраћај на магистрали.

Глава 9 Симулационо окружење

У претходним главама је показано да TLS механизам захтијева сложене протоколе који обично комбинују одржавање кохеренције кеш меморија и подршку спекулацији, као што је предложени SISC протокол. Према томе, може се очекивати и да је његова имплементација и евалуација такође сложена. У овој глави је описано како приступ структурне симулације може олакшати имплементацију и евалуацију перформанси предложеног протокола и додатног хардвера који је потребан за TLS подршку. У наставку је детаљно образложен избор симулационог окружења, као и његове основне карактеристике. Посебно је представљено окружење и начин за генерисање извршног бинарног кода спекулативних нити за предложени CMP систем, као и потребне функционалности симулатора [RADU2008, RADU2009].

9.1 Избор симулационог окружења

Симулатори који су коришћени у евалуационим студијама за постојеће CMP системе су модификоване верзије већ постојећих симулатора опште намјене (нпр., MINT [VEEN1994], SimpleScalar [BURG1997], SIMCA [HUAN1998], Simics/GEMS [MART2005], итд.) или намјенски симулатори (нпр., Multiscalar [SOHI1995], SimOS [ROSE1995], SESC [ORTE2004], SystemC [OSCI2003], итд.) комплетно развијени да би вјерно моделовали конкретне CMP системе. На примјер, Simics/GEMS симулира мултипроцесорски систем Sparc без подршке за спекулативно извршавање. Он омогућава симулацију комерцијалног софтвера као што су базе података које се извршавају на Solaris оперативном систему, али је вријеме извршавања симулације веома велико. Са друге стране, SESC симулатор који симулира спекулативни мултипроцесорски систем базиран на MIPS

процесору има кратко вријеме извршавања, али модел мреже на самом чипу (NoC) није довољно флексибилан да би омогућио даља истраживања.

Евалуација постојећих CMP система са подршком за спекулативно извршавање је претежно рађена на поједностављеним моделима, док реална евалуација оваквих система захтијева извршавање на реалном хардверу или веома детаљне симулационе моделе. Поједностављени хардверски модели се суочавају са проблемима нереално процијењене цијене комуникације и кашњења, подцијењеног ефекта такта, и на крају, са нереалном пропагацијом информација, као што је синхронизација процесора и кеш меморије.

Послије периода развоја сложених монолитних симулатора, све више су препознате предности модуларних симулатора у којима су системи организовани од модела појединачних хардверских компонената налик класичним хардверским блок дијаграмима [AUGU2007, PERE2006, VACH2007, EMER2002]. Њихова модуларност омогућава поновну употребљивост и дијељење хардверских компоненти и омогућава лакшу израду симулатора CMP система који су намијењени за искоришћавање предности спекулације на нивоу нити.

Нека од јавно доступних модуларних симулационих окружења за CMP системе (нпр., Simics/GEMS, SuperScalar, MINT) су разматрана као могући избор за основу симулатора у овом раду. Међутим, она не омогућавају прецизно и детаљно моделовање на најнижем нивоу или не подржавају спекулативно извршавање. Поред тога, имплементација специфичног „*snoopy*“ протокола за кеш кохеренцију са посебним механизмима за надгледање трансакција и синхрони одговор, као и остали аспекти протокола заснованих на заједничкој магистрали који се односе на строге временске захтјеве, нијесу подржани у овим симулационим окружењима (нпр., Simics/GEMS, MINT). Такође, радна оптерећења (*workloads*) за неке од њих (нпр., Simics/GEMS) не могу бити дистрибуирана због власничких права [MART2005].

Са друге стране, модуларна окружења намијењена за симулацију специфичних спекулативних CMP система (нпр., Multiscalar, Hydra, STAMPede, IACOMA, итд.) нијесу могла да буду искоришћена за израду симулатора за предложено CMP рјешење, без обзира да ли су доступни као отворена (*open-source*) рјешења или не, зато што су стриктно пројектовани да моделују одређени CMP систем и тешко се могу искористити за даљу модификацију и прилагођавање за друге спекулативне CMP системе. Међутим, радно оптерећење прилагођено за спекулативно извршавање, које је коришћено за евалуацију у оваквим намјенским симулаторима (нпр., SESC симулатор за I-ACOMA) може бити добра почетна тачка за развој специфичног софтверског алата за евалуацију предложеног спекулативног CMP система.

С обзиром да није било доступног, потпуно адекватног симулационог окружења за евалуацију предложеног спекулативног CMP рјешења, одлучено је да се оно развије у оквиру неког од постојећих генералних симулатора. На тај начин се ствара могућност да се ураде не само неопходна тестирања предложеног

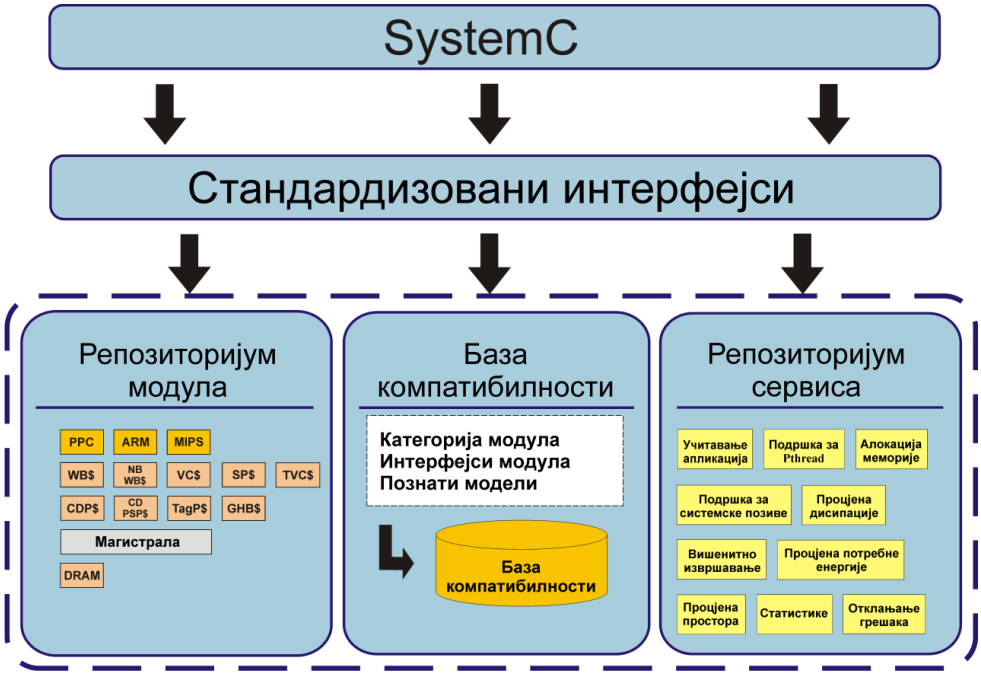
рјешења већ и да се дефинише отворено развојно окружење за тестирање и поређење других спекулативних CMP система. Један од главних критеријума при избору, осим доступности, била је и могућност симулације на нивоу циклуса и детаљног моделовања сваког елемента предложене TLS подршке што се може постићи само у модуларном симулационом окружењу. Због тога је за потребе овог истраживања као основа изабран модуларни симулатор UNISIM [AUGU2007].

9.2 Опис симулационог окружења

Структура симулатора UNISIM је представљена на слици 9-1. Он је имплементиран као слој на врху SystemC индустријског стандарда [OSCI2003] што омогућава да се склопе системи од блокова из репозиторијума модула као што су регуларне PowerPC, ARM или MIPS проточне обраде, затим различите врсте кеш меморија: блокирајуће или неблокирајуће кеш меморије са одложеним уписом, *victim* кеш меморије, разне врсте кеш меморија са дохватањем унапријед (на основу помјераја, садржаја, тагова), бафера са глобалном историјом, итд. Сви ови модули комуницирају преко стандардизованог комуникационог интерфејса архитектуре (ACI) што обезбјеђује својство модуларности неопходно за аутоматско склапање компатибилних модула у жељену архитектуру за симулацију.

У репозиторијуму доступних компоненти за овај рад су поготово важни модули флексибилне кеш меморије са одложеним уписом и магистрале као интерконекције између кеш меморија јер олакшавају имплементацију одговарајућег протокола кохеренције који користи заједничку магистралу. Модуларност омогућава да се лакше симулира предложени протокол за интегрисану кохеренцију и спекулацију модификацијом постојећих коначних аутомата стања. Такође, имплементација других рјешења на сличан начин у истом окружењу би дозволила коректно поређење предложеног CMP рјешења са неким другим CMP предлозима.

Поред модуларности, главне предности симулатора UNISIM су то што је имплементиран као слој на врху индустријског стандарда SystemC, као и то што обезбјеђује експлицитне и стандардизоване комуникационе интерфејсе између модула. У симулатору UNISIM принципи повезивања модула су стандардизовани да обезбиједе њихово аутоматско уклапање. На примјер, сваки модул кеш меморије који подржава дати интерфејс између процесора и меморије се може лако уклопити, без унутрашњих модификација било у процесорском или у меморијском модулу. Према томе, могу се испробати разни модули кеш меморије уколико подржавају дати интерфејс. За надградњу протокола кохеренције додавањем спекулативних функционалности потребно је највише пажње усмјерити на прилагођавање модула кеш меморије подешавајући његов кеш контролер и евентуално урадити његово поређење са стандардним протоколом немодификоване кеш меморије.



Слика 9-1. Принципи UNISIM симулационог окружења

Једна од кључних предности UNISIM окружења је и могућност поновне употребе контролне логике што је обично често занемарено код симулационих окружења, а у крајњем омогућава поновно коришћење изворног кода симулатора. На примјер, у модулима кеш меморије већина простора (*area*) је намијењена меморијским структурама, а само мањи дио припада контролној логици. Међутим, у одговарајућем изворном коду, меморијске структуре су описане са пар линија изворног кода (као низови или мапе), док контролном дијелу одговарају стотине, па чак и хиљаде линија изворног кода. Стога, могућност поновног коришћења кода симулатора се једино може постићи ако то укључује и контролни дио.

У циљу подржавања нове парадигме у пројектовању CMP процесора, UNISIM симулационо окружење подразумијева моделовање на нивоу трансакције TLM **(***Transaction-level Modeling***)** [CAI2003] као додатак уобичајеном моделовању на нивоу циклуса CLM **(***Cycle-level Modeling***)** [FRAB2004, KHAN2012]. У TLM симулаторима већи нагласак је стављен на функционалност преноса података (који подаци су пренешени ка којим и са којих локација), него на сам протокол који је коришћен у преносу података. С друге стране, CLM симулација је заснована на моделима чија је главна карактеристика висока прецизност током евалуације перформанси у поређењу са реалним хардвером. Ово повећава комуникацију између CLM симулационих модула што резултује успоравањем цјелокупне симулације. Међутим, TLM симулатори су мање прецизни, али много бржи од CLM симулатора. UNISIM проширује TLM додајући временске анотације на захтјеве (указује на вријеме између трансакција као што су меморијски

захтјеви или комуникација са осталим језгрима/IP блоковима) и ажурирајући ове захтјеве на основу цјелокупног понашања система (нпр., кашњење захтјева за учитавање због ранијих контенција на мрежи). Иако је TLM добар за моделирање комуникације између модула, он није довољно ефикасан за имплементацију предложеног протокола за кохеренцију и комуникацију. За моделовање прецизног утицаја протокола који се користи за пренос података потребна је финија гранулација симулације која се може постићи само са CLM симулатором.

Предности UNISIM-а такође произилазе из дефиниције његових сервиса. То су специјализовани API интерфејси, који омогућавају да се, поред евалуације перформанси, симулатор прошири неким новим функционалностима (нпр., алати за процјену дисипације, утрошка простора на чипу и времена приступа кеш меморији, статистика, чување података о сесијама, отклањање грешака у симулираним програмима, опонашање оперативног система, итд.). Било који модул који имплементира овај скуп стандардизованих позива аутоматски користи одговарајуће сервисе. Ови сервиси су независни од врсте симулатора јер су доступни на нивоу симулационе машине, што их чини једноставним за модификацију или замјену. Поред тога, ови сервиси могу бити коришћени и за прикупљање статистика о дисипацији, потрошњи енергије и искоришћењу простора за било који компатибилни архитектонски модул, укључујући и рјешења представљена у овој тези.

Сервис процјене снаге је базиран на CACTI моделу који је развијен у HP лабораторијама [TARJ2006]. Он процјењује утрошену снагу у погледу динамичке и статичке снаге. У овом случају он омогућава да се процијени утицај додатних битова потребних за смјештање стања кеш ријечи, као и утицај додатне комуникације изазване предложеним SISC протоколом. Сервис је потпуно одвојен од модела понашања, па се неизмијењен може користити са прилагођеним модулом кеш меморије који подржава SISC протокол.

Поред процјене снаге, UNISIM омогућава да се процијени и захтјевност имплементације протокола у погледу броја транзистора рачунајући простор који захтијевају меморијске структуре (*Area Estimator Tool*). Такође, постоји и могућност да се процијени и реално вријеме приступа за структуре кеш меорије у предложеном спекулативном CMP систему (*Access Estimator Tool*).

9.3 Крос-компајлер

Израда одговарајућег система развојних алата (*toolchain*) представља важну фазу у пројекту развоја софтвера. Ови алати треба да врше превођење и повезивање програмског кода који се развија за одређену архитектуру. Систем се обично састоји од сложених софтверских компоненти са индивидуалним опцијама, али и потребом лаке координације рада. Посебна тешкоћа развоја овог скупа алата произлази из чињенице да он треба да произведе извршни код за другу платформу од оне на којој се извршава (*cross-compilation*).

Неки доступни системи алата се могу користити за општи развој, али имају и много ограничења. Они су обично конфигурисани да подрже већину доступних процесора и стога нијесу оптимизовани за специфична рјешења. Такође, чак и ако адресирају специфична рјешења, тада обично нијесу лаки за коришћење или нијесу оптимизовани за жељено специфично рјешење. Поред тога, често користе застарјеле компоненте које не подржавају функционалности потребне за нови процесор. Зато се врло често израђује сопствени систем алата да би се искористиле све могућности предложеног процесора.

У спекулативном вишенитном моделу извршавања спекулативне нити се издвајају из секвенцијалних апликација и спекулативно извршавају у паралели без нарушавања секвенцијалне програмске семантике. Преводилац креира спекулативне нити постављањем одређених граничних тачака у секвенцијалним бинарним кодовима да би се означио дио који ће се спекулативно извршити. У случају да се детектује нарушавање зависности по подацима, спекулативни CMP систем мора обезбиједити коректно извршавање, па се тако спекулативна нит која је проузроковала нарушавање, као и све касније нити, поништавају и поново извршавају са тачним подацима. За генерисање бинарног кода за спекулативно извршавање у предложеном CMP систему су коришћене двије инфраструктуре преводилаца: POSH преводилац [WEI2006] и *crosstool-NG* алат [CROS2014].

9.3.1 POSH преводилац

POSH је јавно доступни преводилац који је развијен као надградња над *gcc* [REHM2003]. Он генерише бинарни код за 32-битни MIPS II процесор и намијењен је за SESC симулатор [ORTE2004] који моделује I-ACOMA систем [KRIS1999]. С обзиром да предложено CMP рјешење, као и I-ACOMA, има домен спекулације на нивоу итерација петље, POSH преводилац је коришћен за идентификацију и генерисање бинарних извршних кодова спекулативних нити. Спекулативне нити су генерисане на основу структуре кода, итерација петљи и потпрограма било ког нивоа угњеждавања. POSH преводилац умеће *fork* и *commit* инструкције које се користе за иницијализацију и успјешно завршавање спекулативне нити. Након идентификације нити, он користи профајлер да би одстранио бескорисне нити.

POSH преводилац аутоматски одлучује који дио кода ће се извршавати спекулативно, али то није потпуно транспарентно за корисника. За прецизну провјеру исправности спекулације потребно је да се имплементира експлицитна спекулација и из тог разлога је донесена одлука да се развије инфраструктура за крос-компајлер заснована на радном оквиру (*framework*) *crosstool-NG*.

9.3.2 Инфраструктура за крос-компајлер заснована на алату *crosstool-NG*

Прије извршавања неке апликације у симулационом окружењу прво је потребно генерисати статичку бинарну извршну датотеку те апликације за жељени (*targeted*) симулирани процесор. Да би се то урадило потребно је извршити превођење апликације на таквом процесору или за њега израдити крос-компајлер на основном (*host*) рачунару гдје ће се изводити симулације. У овом раду је коришћен други приступ, па је било неоходно направити крос-компајлер.

У ту сврху је коришћен јавно доступни софтверски алат *crosstool-NG* који је специјализован за израду *gcc/glibc* крос-компајлера. Неке од најважнијих предности *crosstool-NG*-а су сљедеће:

1) једноставан кориснички конфигурациони интерфејс,
2) стални раст броја подржаних процесора,
3) подршка за алтернативне компоненте у систему алата као што су *uClibc*, *glibc* и *eglibc* библиотеке,
4) подршка за различите оперативне системе,
5) подршка за различите моделе нити,
5) подршка за отклањање грешака,
6) примјери једноставнијих конфигурација које могу послужити као полазна тачка за развој сопственог система алата.

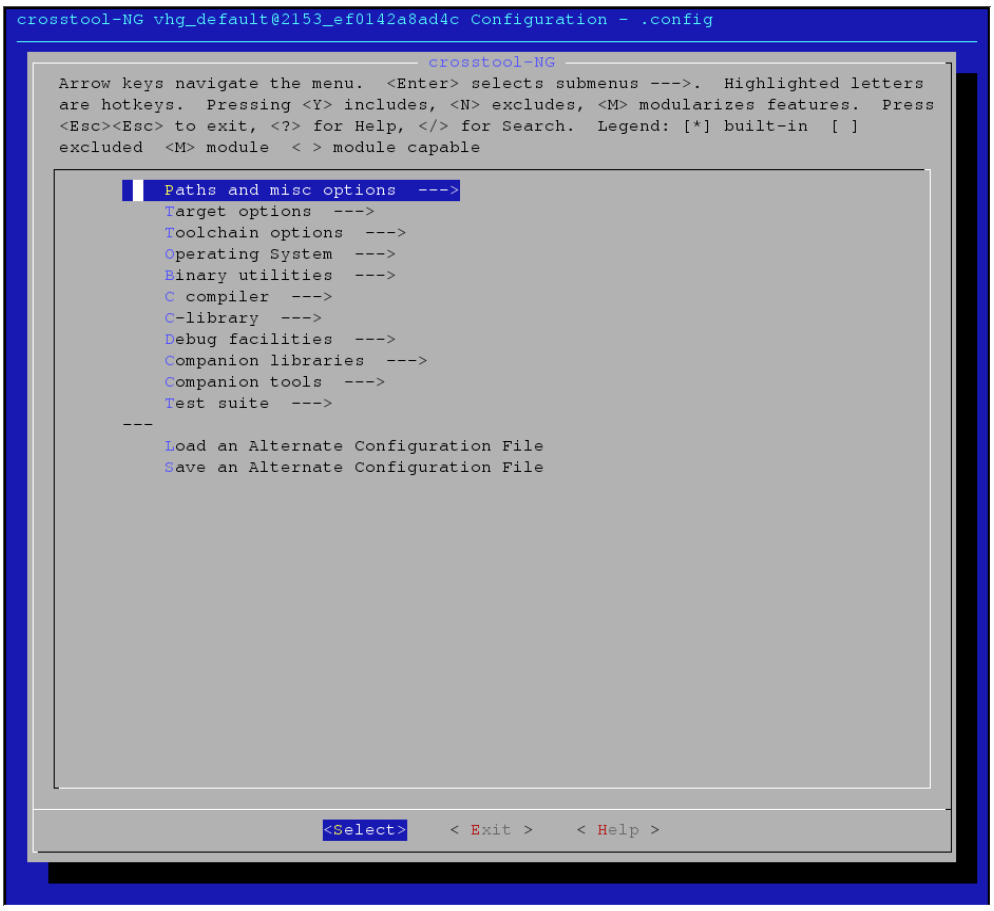
Помоћу *crosstool-NG* скрипти је могуће израдити и тестирати неколико верзија *gcc*-а (од *gcc*-2.95.3 до *gcc*-4.0.0) и *glibc*-а (од *glibc*-2.1.3 до *glibc*-2.3.5) за већину процесора које подржава *glibc* (нпр., Alpha, Arm, AVR32, Blackfin, Microblaze, MIPS, OpenRISC, PowerPC, s390, Sparc, SuperH и x86), односно израдити преводиоце за Linux циљно окружење (Linux, Mac OS X, Solaris и Cygwin).

Развој сопственог крос-компајлера помоћу *crosstool-NG* софтвера много доприноси искоришћењу свих предности предложеног CMP система са подршком за TLS. Добре стране оваквог приступа су: могућност избора различитих верзија компоненти у *crosstool-NG* за израду сопственог алата, оптимизација сопственог алата за изабрано процесорско језгро у CMP систему, познат скуп софтверских закрпа (*patchset*) за одабране верзије компоненти у *crosstool-NG*-у, добра подршка тима који одржава *crosstool-NG* у виду директне комуникације путем електронске поште и интернета (IRC канал) за рјешавање проблема током развоја сопственог алата и једноставност коришћења, одржавања и надградње сопственог алата развијеног помоћу *crosstool-NG*-а.

Развој *gcc* крос-компајлера за коришћење и тестирање је захтјеван посао с обзиром да треба уложити много труда како би се подесили параметри који одговарају предложеном CMP систему са подршком за TLS. Током развоја крос-компајлера помоћу *crosstool-NG* појавило се више проблема који су успјешно ријешени. Тако је, у сарадњи са тимом који одржава *crosstool-NG,* било

потребно дефинисати и тестирати нове закрпе за верзију *crosstool-NG*-а која је коришћена у изради жељеног крос-компајлера. Затим је било потребно извршити избор одговарајућих опција за превођење изворног кода тестних програма, избор одговарајућег модела нити и погодне библиотеке за потребе имплементације TLS-а, као и избор опција за укључење подршке за отклањање грешака.

Crosstool-NG софтвер је инсталиран са неопходним закрпама које су добијене од тима који га одржава и помоћу *menuconfig* конфигурационог интерфејса (слика 9-2) су подешени параметри у *.config* датотеци као што је приказано у табели 9-1.



Слика 9-2. Конфигурациони интерфејс *menuconfig* у алату *crosstool-NG*

У *General target options* подменију је изабран MIPS као циљна архитектура оптимизована за MIPSII (32-bit, опције за превођење: “-mno-abicalls“ како компајлер не би генерисао позиционо независан код). Постоје три разлога због којих је MIPS изабран као процесорско језгро у симулацији предложеног спекулативног CMP рјешења: 1) MIPS је најчешће коришћено процесорско језгро у постојећим спекулативним CMP системима као што су Multiscalar, Hydra, Multiplex, I-ACOMA, STAMPede и Trace; 2) POSH преводилац аутоматски генерише спекулативни бинарни код за 32-битни MIPS II процесор; 3) спекулативни бинарни кодови за MIPS процесор чије коришћење је омогућила

истраживачка група I-ACOMA омогућавају брзу евалуацију предложеног спекулативног CMP рјешења.

Табела 9-1. Подешавање параметара у *.config* датотеци

Опције избора	**Вриједности**
Paths and misc options*	
	Подразумијеване вриједности
Target options*	
*General target options**	Target Architecture: mips*
	Endianess: Bid endian*
	Bitness: 32 bit*
*Target optimizations**	Architecture level: mips2*
	Target CFLAGS: -mno-abicalls*
	Target LDFLAGS: -mno-abicalls*
	Остале вриједности су подразумијеване
Toolchain options*	
	Подразумијеване вриједности
Operating System*	
	Target OS: linux*
	Linux kernel version: 2.6.31.14* - (подразумијевана вриједност)
	Остале вриједности су подразумијеване
Binary utilities*	
	Подразумијеване вриједности
C compiler*	
	C compiler: gcc*
	gcc version: 4.3.2*
	Остале вриједности су подразумијеване
C-library*	
	C-library: uClibc*
	uClibc version: 0.9.30.1*
	Threading implementation to use: none*
	Остале вриједности су подразумијеване
Companion libraries*	
	Подразумијеване вриједности

Легенда: * - термини су преузети у оригиналу из *menuconfig* интерфејса

Важно је нагласити да је реализација крос-компајлера морала бити изведена без имплементације нити с обзиром да постоје конфликти између TLS *fork* и *commit* техника и класичног POSIX (*Portable Operating System Interface*) интерфејса за вишенитно извршавање [PRAS1996]. Такође, потребно је да *LibC* библиотека буде без подршке нитима да би се омогућило додавање предложене спекулативне подршке, па је због тога избор пао на *uClibc* библиотеку која задовољава тај услов. Поред тога, коришћењем *uClibc* библиотеке се жељело постићи генерисање бинарних кодова једноставним системским позивима како би се на тај начин добило да симулатор највећи дио времена током симулације буде фокусиран на кернеле спекулативног извршавања умјесто симулирања комплексних системских библиотека. С обзиром да није планирана имплементација специфичних функционалности за отклањање грешака, искључене су *gdb* и *duma* функционалности, док су сви остали параметри

crosstool-NG алата остављени са њиховим подразумијеваним вриједностима да би се успјешно генерисао MIPS крос-компајлер.

Предложени полуаутоматски приступ уводи *SISC_fork()* и *SISC_commit()* функције на нивоу изворног кода, слично као код POSH преводиоца. *SISC_fork()* функција иницира нову спекулативну нит на слободном процесору, док извршавање *SISC_commit()* функције показује да је нит завршила са извршавањем. Превођење изворног кода се, такође, врши као код POSH преводиоца, помоћу *gcc* крос-компајлера генерисаног од стране *crosstool-NG* алата. Он преводи функцијске позиве за *SISC_fork()* и *SISC_commit(),* који се додају у сваки спекулативни програмски код, као скокове на рутине у симулатору које их одрађују. Симулатор се ослања на ове позиве функција да би на тај начин одредио који дјелови програмског кода треба да буду спекулативно извршени.

У наставку је дат примјер једноставног програма који сабира два цјелобројна вектора `A` и `B` кода у вектор `C`. Прво је приказана секвенцијална верзија за један процесор, а затим и верзија за спекулативни CMP процесор са додатим позивима функција *SISC_fork()* и *SISC_commit().*

Секвенцијална верзија за један процесор:

```
#define N 10000
int A[N], B[N], C[N];

void vecadd(){
 int i, j;
 for(i=0; i<N; i++)
   C[i] = A[i] + B[i];

}
```

Верзија за спекулативни CMP процесор – секвенцијални код је подијељен на скупове од по 100 нити, при чему се свака нит састоји од 100 итерација и извршава на једном процесору по принципу кружног опслуживања:

```
#define N 10000
int A[N], B[N], C[N];

void spec_vecadd(){
int i, j, index=0;
 for(i=0; i<N/100; i++){
   if(SISC_fork() == 0) goto end_i_loop; //нова спекулативна нит
   index = i*100;
   for(j=0;j<100;j++,index++)    //издвајање скупа итерација
                                 //за обраду
     C[index] = A[index] + B[index];
   SISC_commit(); //завршетак нити након краја обраде
                  //скупа итерација
  end_i_loop:
 }
}
```

Након што крос-компајлер, који је генерисан помоћу *crosstool-NG*, изврши превођење спекулативне верзије програмског кода са додатим позивима функција *SISC_fork()* и *SISC_commit()*, добија се сљедећи асемблерски код:

```
<spec_vecadd>:
          ...
      sw  $0,32($30)                                    //i=0
    i_loop:nop   <----------------------+
      jal SISC_fork                     |
      nop                               |
      beqz  $2,end_i_loop -----------+  |
      nop                            |  |
      lw  $3,32($30)                 |  |                  //$3=i
      move  $4,$0                    |  |                   //j=0
      sll  $2,$3,0x2                 |  |                 //$2=i*4
      lui  $3,0x1001                 |  |
      addiu  $3,$3,18592             |  |                  //$3=&C
      addu  $7,$3,$2                 |  |              //$7=&C[i]
      lui  $3,0x1000                 |  |
      addiu  $3,$3,4128              |  |                  //$3=&A
      addu  $6,$3,$2                 |  |              //$3=&A[i]
      lui  $3,0x1001                 |  |
      addiu  $3,$3,-21408            |  |                  //$3=&B
      addu  $5,$3,$2                 |  |              //$3=&B[i]
    j_loop:lw  $2,0($5)   <-------+  |  |           //$2=A[index]
      lw  $3,0($6)                |  |  |           //$3=B[index]
      addiu  $4,$4,1              |  |  |                  // j++
      addiu  $6,$6,4              |  |  |
      addu  $2,$2,$3              |  |  | //$2=A[index]+B[index]
      sw  $2,0($7)                |  |  |           //C[index]=$2
      li  $2,100                  |  |  |
      addiu  $5,$5,4              |  |  |
      addiu  $7,$7,4              |  |  |
      bne  $4,$2,j_loop ----------+  |  |
      nop                            |  |
      jal SISC_commit                |  |
end_i_loop:nop       <---------------+  |
      lw  $3,32($30)                    |                  //$3=i
      li  $7,10000                      |
      addiu  $3,$3,100                  |              //$3=i+100
      sw  $3,32($30)                    |                  //i=$3
      bne  $3,$7,i_loop ----------------+
      nop
      ...
```

Позиви функција *fork* и *commit* се сада појављују као `jal` *(jump and link)* инструкције у асемблерском коду. Сада симулатор треба да обради ове позиве и да на основу њих изврши одговарајуће акције да би ефективно раздијелио нити на доступне процесоре, као и да рукује операцијама извршавања нити и

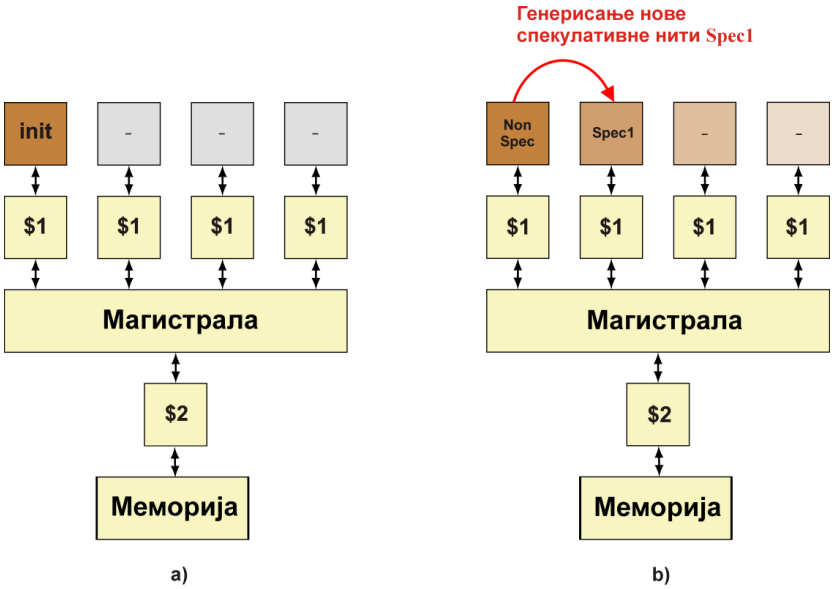
просљеђивања неспекулативног статуса сљедећој нити након што се неспекулативна нит заврши.

9.4 Управљање спекулативним извршавањем у симулатору

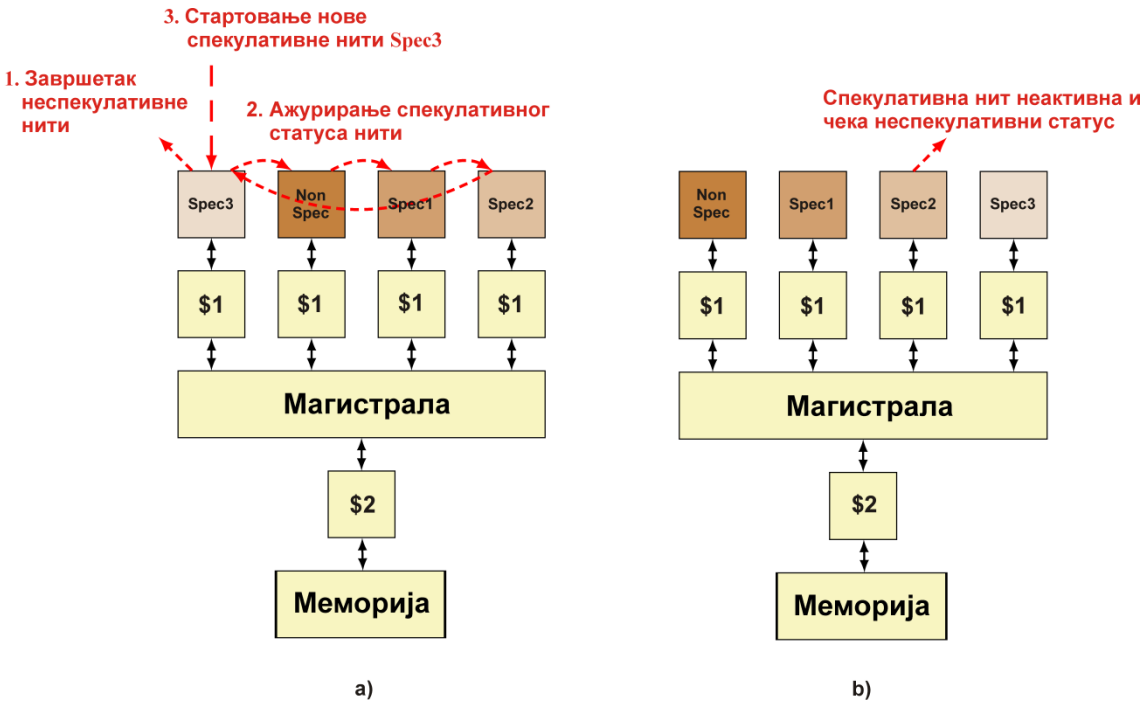
Симулатор извршава спекулативне и неспекулативне дјелове кода на тај начин што обрађује *fork* и *commit* позиве које је уметнуо преводилац. Он има могућност да распоређује нове спекулативне нити на доступним процесорским језгрима када дође до извршавања *fork* инструкције, као и да врши ослобађање ресурса када дође до извршавања *commit* инструкције. Поред тога, симулатор мора да води рачуна о различитим нивоима спекулације, укључујући и операцију просљеђивања неспекулативног статуса када се неспекулативна нит заврши.

Претходно поменуте `jal` инструкције, настале превођењем *fork* и *commit* инструкција у крос-компајлеру, скачу на лабеле на којима се налазе симулаторске рутине за спекулативно извршавање програма. Начин на који симулатор реагује на *fork* инструкције илустрован је на слици 9-3. На почетку извршавања само је прво процесорско језгро активно и оно извршава главну нит прије било какве спекулације (слика 9-3а). Када дође до извршавања *fork* инструкције симулатор треба да креира нову спекулативну нит (Spec1) на сљедећем процесорском језгру (слика 9-3б). Умјесто извршавања функцијског позива симулатор копира регистарске вриједности из скупа регистара иницирајуће нити у скуп регистара нове нити, поставља вриједност програмског бројача за нову нит на инструкцију која слиједи након функцијског позива и тако започиње спекулативно извршавање.

Начин на који симулатор треба да обрађује *commit* инструкције зависи од спекулативности саме нити. Када неспекулативна нит дође до *commit* инструкције, симулатор треба да ажурира спекулативност свих нити, док процесор на којем се извршавала текућа нит постаје доступан за сљедећу најспекулативнију нит (Spec3), уколико таква нит постоји (слика 9-4а). Када спекулативна нит дође до *commit* инструкције, она постаје неактивна и чека на статус неспекулативне нити (слика 9-4б).



Слика 9-3. Очекивано понашање симулатора код извршавања *fork* инструкције: а) прије спекулације само први процесор извршава главну нит; б) након извршавања *fork* инструкције први процесор наставља са извршавањем првог блока итерација док други процесор спекулативно извршава други блок итерација (примјер спекулативног кода у секцији 9.3.2).



Слика 9-4. Очекивано понашање симулатора у случају извршавања *commit* инструкције: а) стање након завршавања неспекулативне нити; б) стање након завршавања спекулативне нити.

Глава 10 Имплементација SISC-WI протокола у симулационом окружењу

Имплементација комплетне предложене подршке за TLS је врло сложен и захтјеван посао, па је за имплементацију у симулатору одабран њен најсложенији дио – SISC-WI протокол. У овом поглављу су представљени имплементациони детаљи у вези са интеграцијом SISC-WI протокола у UNISIM симулатору при чему је нагласак стављен на промјене које је било неопходно урадити на постојећим модулима, као и на увођењу нових модула, а све у циљу имплементације подршке спекулацији на нивоу нити [RADU2008, RADU2009].

10.1 Прилагођење симулационог окружења

Због разлога изнесених у претходној глави, за потребе имплементације подршке за TLS на меморијском нивоу предложене у овом раду одабрано је симулационо окружење UNISIM. Библиотека компоненти овог симулатора садржи неколико модула који су морали бити модификовани како би задовољили захтјеве SISC-WI протокола који су представљени у седмој глави. Модул кеш меморије са одложеним уписом је параметризован тако да симулира L1 кеш меморију за податке са величином линије која одговара једној ријечи.

Постојећи MESI протокол за кохеренцију је искоришћен за имплементацију SISC-WI протокола тако што је извршена модификација одговарајућег коначног аутомата стања. Посебан изазов је био то што се SISC-WI протокол, који се имплементира у кеш контролеру сваког процесорског језгра, понаша различито током симулације у складу са спекулативним статусом нити која се извршава на датом процесорском језгру. Поред тога, с обзиром да SISC-WI протокол укључује доста више стања у односу на постојећи MESI протокол, потребно је било

модификовати и кеш контролер како би он могао реаговати на нове догађаје који одговарају новим прелазима између стања у измијењеном аутомату стања. Кеш контролер посебно укључује догађаје који су посљедица спекулативне природе SISC-WI протокола, као када спекулативна нит постаје неспекулативна па се врши прелаз из сваког спекулативног стања у аутомату у одговарајуће неспекулативно стање (нпр., прелаз из *Modified Speculative* стања у *Modified* стање). Такође, извршена је и модификација постојеће системске магистрале да би се додали потребни сигнали за TLS као што су: спекулативни контролни сигнали, сигнали арбитрационог механизма и сигнали маски спекулативних нити.

С обзиром да UNISIM није подржавао MIPS процесор, потребно је било да се изврши његова имплементација. Прво је имплементиран симулаторски модул за основну MIPS архитектуру која одговара MIPS R4400 32-битном процесору, а који садржи пар 64KB L1 директно мапираних или сет-асоцијативних кеш меморија са одложеним уписом за инструкције и податке, дијељену 512KB L2 директно мапирану или сет-асоцијативну кеш меморију са одложеним уписом и системску магистралу која је повезана на DRAM меморију. Након тога је извршено проширење MIPS симулатора са једног на четири процесорска језгра. Са становишта симулатора ово значи да је потребно само реплицирати неколико процесорских модула (процесорски модул садржи процесорско језгро и L1 кеш меморије за инструкције и податке), повезати их путем системске магистрале и додати одређени хардверски механизам како би се осигурала кохеренција меморије. На крају, у овакав MIPS симулатор за CMP систем је додата подршка за TLS чиме је добијен коначни MIPS симулатор за CMP TLS систем. У наредној секцији ће бити детаљно представљена подршка за спекулативно извршавање која је додата у MIPS симулатор за CMP систем.

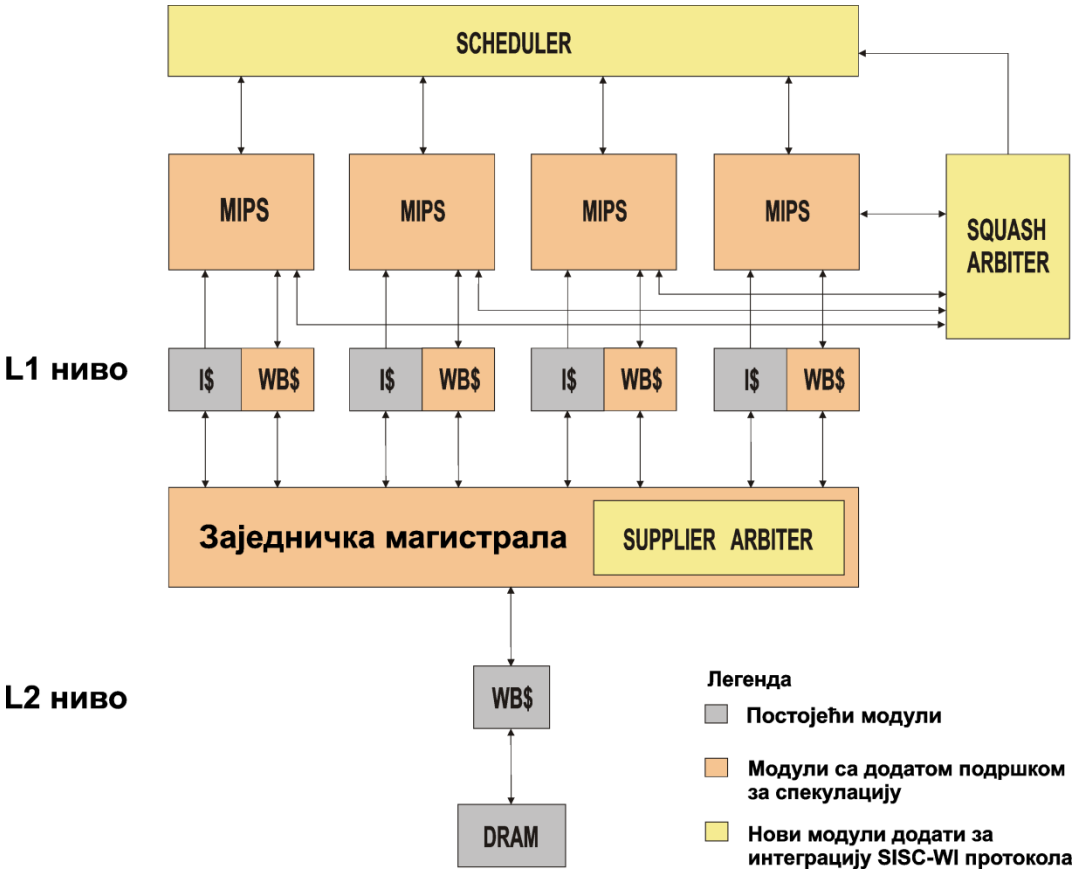
Поред тога, додата су и три нова модула како би се у потпуности подржао механизам TLS и интеграција SISC-WI протокола у UNISIM: 1) модул *Scheduler* чија је улога да врши распоређивање спекулативних нити на процесорска језгра, 2) модул *Squash Arbiter* чија је улога да реагује на нарушавање RAW зависности по подацима током спекулативног извршавања и да изврши потребне акције за поништавање нити и 3) модул *Supplier Arbiter* чија је улога да изабере добављача између више нити које су потенцијални добављачи за тражени податак.

Структура симулираног система који подржава TLS на меморијском нивоу (SISC-WI протокол) са главним компонентама и међусобним везама у UNISIM симулатору је представљена на слици 10-1.

10.2 Интеграција модула процесорског језгра

Свако од четири процесорска језгра у предложеном CMP TLS рјешењу може извршавати или неспекулативну или спекулативну нит током спекулативног извршавања. Нит се додјељује датом процесорском језгру након успјешно

извршене *fork* операције која је иницирана од стране претходне (родитељске) нити.



Слика 10-1 Структура симулираног система

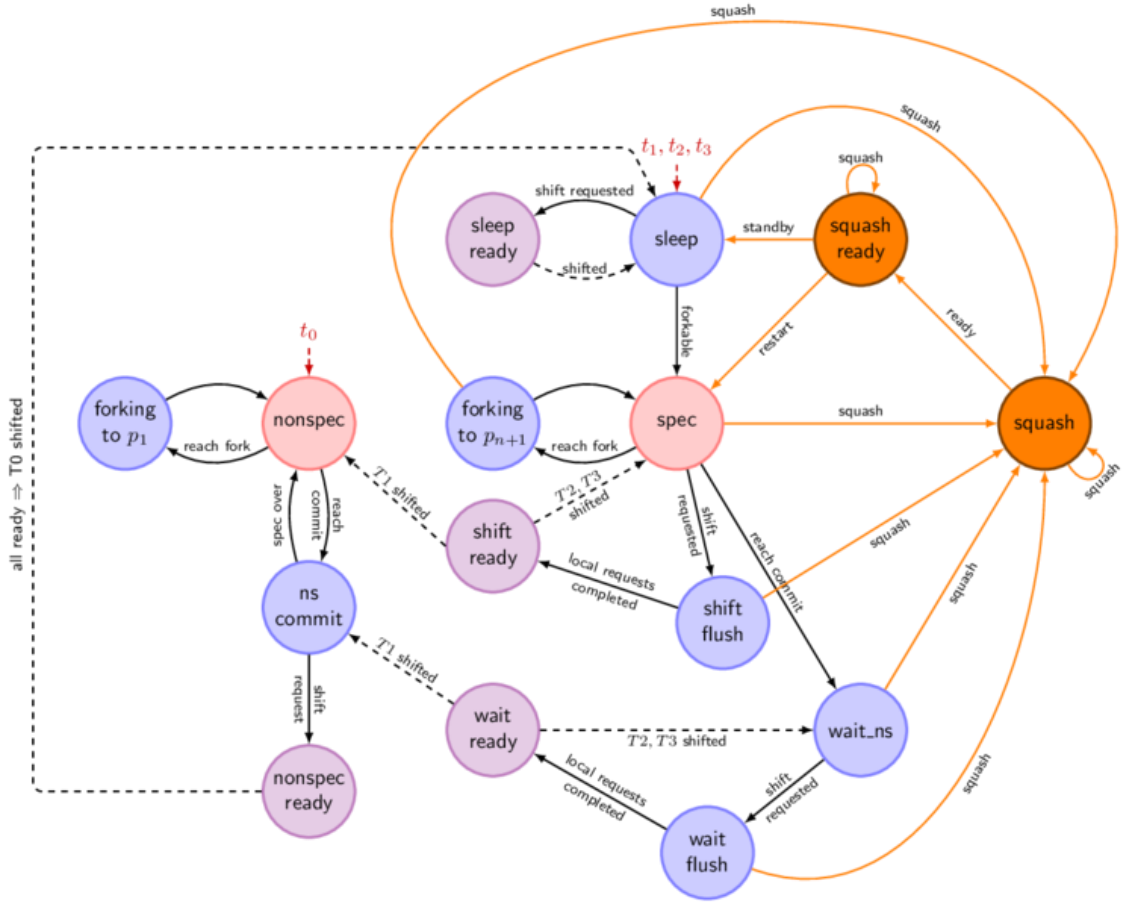
Понашање процесорских језгара током спекулативног извршавања у предложеном CMP рјешењу са имплементираним SISC-WI протоколом за меморијску комуникацију је представљено кроз дијаграм прелаза процесорских стања који је приказан на слици 10-2. Црвеном бојом су означена стања у којима се у режиму проточне обраде инструкције прихватају, извршавају и потврђују њихови резултати, док је у стањима са плавом бојом режим проточне обраде суспендован. Наранџастом бојом су означени прелази и стања када се врши поништавање (*squashing*) нити која се извршава на датом језгру. Љубичастом бојом су означена стања која одговарају синхронизационој баријери прије промјене стања спекулативног статуса нити, док придружене испрекидане линије одговарају прелазима које се догађају након завршетка синхронизационе баријере.

Постоји петнаест стања у којима се може наћи процесорско језгро током спекулативног извршавања: **nonspec**, **forking_to_P1**, **ns_commit**, **nonspec_ready**, **sleep**, **sleep_ready**, **spec**, **forking_to_Pn+1**, **shift_flush**, **shift_ready**, **wait_ns**,

wait_flush, **wait_ready**, **squash** и **squash_ready**. Процесорска стања се могу груписати на сљедећи начин:

а) Активна (*running*) стања:

- **sleep**: Ово је почетно стање за сва процесорска језгра осим оног који извршава неспекулативну нит *t0*. Када је процесорско језгро у овом стању, оно не извршава нит већ само одговара на захтјеве. Процесорско језгро се налази у овом стању прије него што му распоређивач додијели спекулативну нит која ће се на њему извршавати.
- **nonspec:** Ово је активно стање за процесорско језгро које извршава неспекулативну нит. У овом стању се у режиму проточне обраде инструкције прихватају, извршавају и потврђују њихови резултати.
- **spec**: Ово је активно стање за процесорска језгра која извршавају спекулативне нити (нпр., нити *t1*, *t2* и *t3*). У овом стању се у режиму проточне обраде инструкције прихватају, извршавају и потврђују њихови резултати.



Слика 10-2. Дијаграм прелаза процесорских стања у предложеном систему са имплементираним SISC-WI протоколом

б) Стања гранања (*forking*) у које родитељска нит прелази након извршавања *fork* инструкције. Намјена ових стања је да процесорско језгро сачека да се

заврше сви процесорски захтјеви који су у фази извршавања да би могло да постави акцију *fork* у ред за чекање у распоређивачу. Нова спекулативна нит ће бити извршена на сљедећем језгру које прелази из стања **sleep** у стање **spec**.

- **forking_to_P1:** Означава да је неспекулативна нит у свом извршавању дошла до *fork* инструкције.
- **forking_to_Pn+1**: Означава да је спекулативна нит у свом извршавању дошла до *fork* инструкције.

в) Завршна (*commit*) стања означавају да су неспекулативна или спекулативна нит у свом извршавању дошле до *commit* инструкције.

- **ns_commit**: Намјена овог стања је да процесорско језгро сачека да се заврше сви процесорски захтјеви који су у фази извршавања и да у зависности од статуса спекулативног извршавања или заврши са њим или да иницира припремну синхронизациону фазу за промјену спекулативног статуса нити.
- **wait_ns**: Спекулативна нит чека да прихвати неспекулативни статус од претходне нити. Намјена овог стања је да процесорско језгро на којем се извршава дата спекулативна нит сачека да почне припремна синхронизациона фаза за промјену спекулативног статуса нити.

г) Стања промјене спекулативног статуса (*shift*) које се односе на извршавање припремне (*preshifting*) синхронизационе фазе за промјену спекулативног статуса нити. У њима се зауставља локално извршавање и чека се да се заврше сви процесорски захтјеви чије извршавање је започето.

- **shift_flush**: Спекулативна нит која се извршава на локалном процесорском језгру чека промјену спекулативног статуса.
- **wait_flush**: Спекулативна нит која се извршава на локалном процесорском језгру чека неспекулативни статус.

д) Стања спремности (*ready*) – свако процесорско језгро је спремно да изврши акцију промјене спекулативног статуса нити коју извршава. Прелази у ова стања се извршавају секвенцијално како би се осигурало да она не могу бити поништена:

- **nonspec_ready**: Намјена овог стања да процесорско језгро на којем се завршила неспекулативна нит сачека да се заврши припремна синхронизациона фаза за промјену спекулативног статуса нити што ће указати да су тада сва процесорска језгра спремна да синхроно изврше акцију измјене спекулативног статуса нити које извршавају.
- **sleep_ready**: Процесорско језгро је спремно за синхронизовану акцију промјене спекулативног статуса нити.
- **shift_ready**: Процесорско језгро је завршило припремну синхронизациону фазу за промјену спекулативног статуса нити. Спекулативна нит која се извршава на процесорском језгру чека нови спекулативни статус.
- **wait_ready**: Процесорско језгро је завршило припремну синхронизациону фазу за промјену спекулативног статуса нити.

Спекулативна нит која се извршава на процесорском језгру чека неспекулативни статус.

е) Стања поништавања (*squash***)** - постоје два стања за *squash* акцију с обзиром да је потребно прво завршити текуће процесорске захтјеве, а затим их поништити након њиховог завршетка:

- **squash**: Претходно детектовано нарушавање зависности по подацима код нити узрокује акцију поништавања те нити као и свих каснијих нити. Намјена овог стања је да поништи све текуће процесорске захтјеве и обавијести *Squash Arbiter* да су процесорско језгро и спекулативна нит која се на њему извршава спремни за акцију поништавања текућег стања и свих спекулативно произведених вриједности.
- **squash_ready**: Текуће стање процесорског језгра и нити која се на њему извршава је поништено и процесорско језгро остаје у стању **squash_ready** све док *Squash Arbiter* не укаже да ли ће процесорско језгро бити или рестартовано у стању **spec** и почети са поновним извршавањем придружене спекулативне нити или ће бити постављено у стање **sleep**.

Прелазе између стања изазивају сљедеће акције:

а) Акција *reach_fork* је посљедица извршавања *fork* инструкције чиме се захтјев за *fork* поставља у ред за чекање у распоређивачу.

б) Акцију *forkable* издаје распоређивач чиме се сљедећој нити дозвољава да почне са спекулативним извршавањем.

в) Акција r*each_commit* је посљедица извршавања *commit* инструкције. Спекулативне нити тада чекају неспекулативни статус док неспекулативна нит иницира сихнронизацију за промјену спекулативног статуса нити која је представљена акцијом *shift_request*.

г) Током фазе синхронизације за промјену спекулативног статуса нити све спекулативне нити извршавају акцију *shift_requested* и чекају да се заврше сви меморијски захтјеви локалног процесорског језгра који су у фази извршавања. Након њиховог завршетка извршава се акција *local_requests_completed* при чему спекулативна нит који се још увијек извршава прелази у стање **shift_ready**, спекулативна нит која је завршила са извршавањем прелази у стање **wait_ready**, док нити које се налазе у стању **sleep** прелазе у стање **sleep_ready**. Важно је нагласити да се извршавање акција *local_requests_completed* обавља у растућем *threadID* поретку како би се осигурало да се не захтијева поништавање нити које се налазе у неком од претходно поменутих *ready* стања.

д) Када све нити достигну неко од стања *ready*, зависно од тога да ли се ради о неспекулативној нити (**nonspec_ready**), о спекулативној нити која се још увијек извршава (**shift_ready**), о спекулативној нити која је завршила са извршавањем (**wait_ready**) или о нити која је била у стању **sleep**

(**sleep_ready**), долази до синхронизоване промјене спекулативног статуса нити извршавањем акција *shifted*.

е) Када спекулативна нит детектује нарушавање меморијске зависности по подацима, извршава се акција *squash* за нит која је извршила детекцију и све наредне нити, што изазива прелаз у стање **squash**. Спекулативна нит остаје у стању **squash** док се не заврше сви меморијски захтјеви локалног процесорског језгра који су у фази извршавања, када је нити дозвољено да акцијом *ready* пређе у стање **squash_ready**. Извршавањем акције *ready* се обавјештава *Squash Arbiter* да су завршене све акције које треба да се изврше током поништавања нити.

ф) Акције *standby* и *restart* су омогућене од стране *Squash Arbiter*-а чиме се наредним нитима дозвољава да пређу у стање **sleep**, док нит која је била иницијатор акције *squash* прелази у стање **spec**.

г) Акцију *spec_over* издаје распоређивач након завршетка спекулативног кода

Да би се осигурало да не постоје два прелаза стања активна у истом тренутку, уведени су нивои приоритета између акција *squash*, *shift*, *fork* и *commit*. Поредак приоритета од највећег до најмањег по акцији је сљедећи: *squash*, *shift*, *fork* и *commit*.

Акција *squash* има највећи приоритет између свих наведених акција, при чему се врши поништавање нити која је изазвала грешку у зависности по подацима као и свих наредних нити. Том приликом се врши поништавање свих спекулативно произведених вриједности у њиховим одговарајућим L1 кеш меморијама за податке. Акције *shift*, које се догоде током акција *squash*, ће бити обустављене док се не заврше акције *squash*, док се акције *fork* и *commit* поништавају од стране акција *squash*. Оне ће бити поново извршене када се рестартује одговарајућа спекулативна нит на процесору.

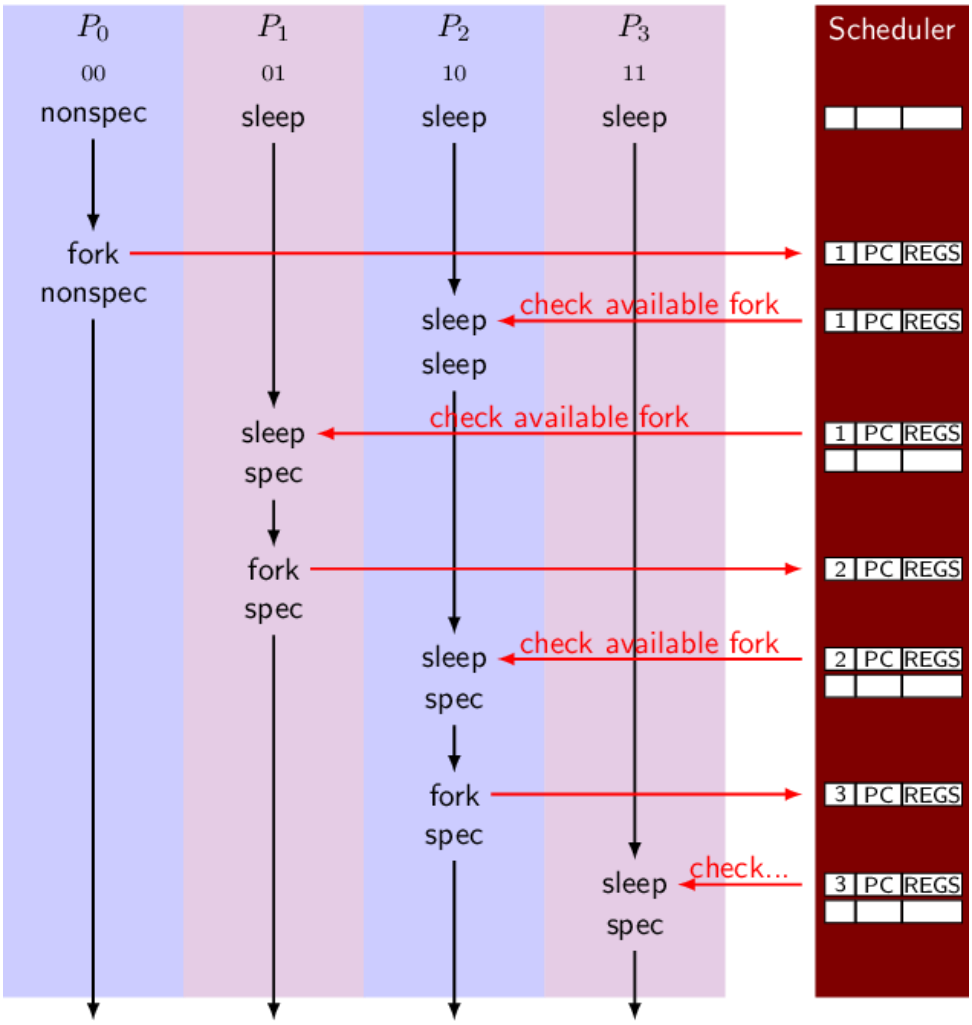
10.3 Имплементација модула *Scheduler*

Scheduler је нови модул који је имплементиран у симулатору UNISIM да би се подржао TLS. Његова намјена је да имитира акције које би иначе радио оперативни систем, јер је симулација оперативног система на нивоу циклуса временски изузетно захтјевна па се од тога одустало. Овај модул је централизована компонента која води рачуна о распоређивању нити за извршавање на процесорским језгрима.

Интеракција између процесорских језгара и модула *Scheduler* је представљена на слици 10-3. Свако процесорско језгро је повезано са модулом *Scheduler* преко једног намјенског порта.

Када нит током извршавања дође до инструкције *fork* модул *Scheduler* поставља у ред за чекање триплет који се састоји од: ID процесора који ће извршавати нову спекулативну нит, адресу кода који ће се извршавати на том

процесору (*PC - Program Counter*) и регистарске вриједности (*REGS*). Док је процесорско језгро у стању **sleep** оно сваког циклуса пропитује *Scheduler* да ли постоји расположива нит која би се на њему извршавала. Када постоји, избацује се одговарајући триплет из реда за чекање и почиње се са извршавањем спекулативне нити на процесорском језгру од адресе *PC*. Уз то, *Scheduler* је одговоран да реагује на акције *squash* и *shift* и да информише систем о броју спекулативних нити које се тренутно извршавају.



Слика 10-3 Интеракција између процесорских језгара и модула *Scheduler*

Улазни сигнали у модул *Scheduler*, који долазе од стране процесорског језгра, су сљедећи:

- сигнал *fork* – прима се од стране процесорског језгра које извршава родитељску нит када током извршавања та нит дође до инструкције *fork*,

- сигнал *check* – прима се од стране сваког процесорског језгра које се налази у стању **sleep** да би се провјерило да ли постоји нова спекулативна нит расположива за извршавање,
- сигнал *shifted* – прима се од стране сваког процесорског језгра када се акција *shift* догоди током циклуса.

Улазни сигнали у модул *Scheduler*, који долазе од стране *Squash Arbiter*-а, су:

- сигнал *squash* – прима се од стране *Squash Arbiter*-а када се догоди *squash* током циклуса,
- сигнал *standby* – прима се од стране *Squash Arbiter*-а када процесорско језгро прелази у стање **sleep**.

Излазни сигнал из модула *Scheduler*, који се шаље ка процесорском језгру, је:

- сигнал *forkable* – шаље се као одговор на сигнал *check* да би се обавијестило процесорско језгро које се налази у стању **sleep** да постоји нова спекулативна нит расположива за извршавање.
- сигнал *spec_over* – шаље се ка процесорском језгру које извршава неспекулативну нит када се заврши спекулативни дио кода.

Са становишта родитељске нити све акције *fork* су успјешне и неблокирајуће пошто се све постављају у ред за чекање у модулу *Scheduler*.

Псеудокод који објашњава понашање модула *Scheduler* током спекулативног извршавања је сљедећи:

```
if настао squash током циклуса then
  игнорисати долазне сигнале fork, check и standby и
  испразнити триплет
else if постоји расположива нит за извршавање then
   испразнити триплет
else if захтјев за генерисањем нити је примљен током циклуса
then
   поставити у ред за чекање одговарајући триплет
else if акција shift се догоди током циклуса then
   ажурирати ID-ве процесора у триплетима
else if процесорско језгро прелази у стање sleep then
   ажурирати број спекулативних нити које се извршавају
```

10.4 Имплементација модула *Squash Arbiter*

Squash Arbiter (SA*)* модул је, такође, нови модул који је развијен у симулатору UNISIM да би се подржао TLS. Намјена овог модула је да генерише акције *squash* када спекулативна нит детектује нарушавање RAW зависности по подацима, при чему се врши поништавање те нити као и свих наредних нити. SA разрјешава два проблема који су везани за поништавање нити: 1) када се двије акције *squash* догоде у истом тренутку и 2) када се акција *squash* догоди током акције *squash*.

Треба нагласити да у доступној литератури за било који од постојећих спекулативних CMP система није разматрано рјешење за ова два поменута проблема који се јављају код поништавања нити. Постојећи спекулативни CMP системи сматрају да се акција *squash* обави тренутно док у реалном хардверу било којој акцији *squash* треба времена да испразни структуре што доводи до могуће ситуације да се током текуће акције *squash* догоди нова акција *squash*.

У модулу SA имплементиране су сљедеће пројектне одлуке: 1) задржава се *squash* најмање спекулативне нити као актуелни *squash* када се више акција *squash* догоди у истом тренутку; 2) игноришу се нове акције *squash* током постојеће акције *squash* ако је нит која узрокује поништавање има већи спекулативни статус; 3) текућа акција *squash* се замјењује новом рестартујући при томе акцију *squash* када мање спекулативна нит детектује нарушавање RAW зависности по подацима.

Акције *squash* се завршавају у обрнутом поретку да би се избјегло да се родитељска нит рестартује и изврши акцију *fork* прије него што наредна нит заврши са својом акцијом *squash*. Након завршетка акције *squash*, све наредне нити примају сигнал *standby* од SA како би се процесорска језгра на којима се оне извршавају пребацила у стање **sleep**, док нит која је била иницијатор акције *squash* прима сигнал *restart* од SA како би се процесорско језгро на којем се она извршава пребацило у стање *spec* (слика 10-2).

Током операције *squash* акције које је потребно извести на нивоу кеш меморије су сљедеће: 1) поништити све спекулативно произведене ријечи; 2) отказати све текуће процесорске захтјеве. Модификације стања кеш ријечи може бити урађена помоћу комбинационе логике. Међутим, отказивање свих текућих процесорских меморијских захтјева не може бити урађено помоћу комбинационе логике, а да то нема значајан утицај на кашњење. Класична имплементација ће да сачека да се захтјеви заврше, а затим их након тога све поништава. Са процесорског становишта акција *squash* треба да рестартује процесорско језгро тако што ће испразнити степене проточне обраде, извршити обнављање (*restoring*) регистарских вриједности за 32 MIPS регистра и програмски бројач *PC*. Ове активности не могу бити урађене помоћу комбинационе логике.

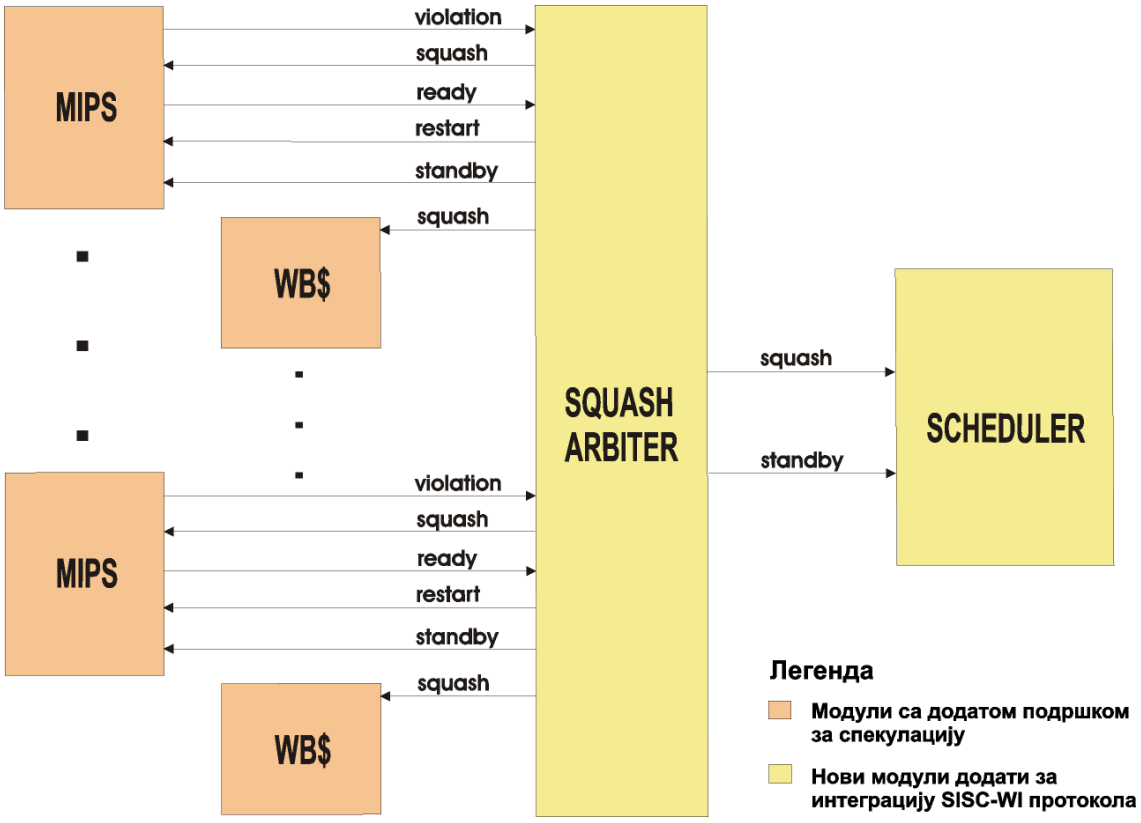
Улазни сигнали за модул SA су:

- сигнали *violation* – шаљу се од стране процесорског језгра ка SA када нит која се извршава на датом процесорском језгру детектује нарушавање RAW зависности по подацима,
- сигнали *ready* – шаљу се од стране процесорског језгра ка SA како би га обавијестили да су завршене акције које је потребно извршити током *squash* акције.

Излазни сигнали за модул SA су:

- сигнали *squash* – шаљу се ка процесорском језгру које извршава спекулативну нит која је детектовала нарушавање RAW зависности по подацима, као и ка процесорским језгрима која извршавају све наредне нити, као и према *Scheduler*-у како би он био информисан да се догодио *squash* током циклуса,
- сигнали *standby* – шаљу се ка процесорском језгру како би оно добило информацију да треба да пређе у стање **sleep**, као и према *Scheduler*-у који треба да чува информацију о броју спекулативних нити које се тренутно извршавају на систему,
- сигнали *restart* – шаљу се ка процесорском језгру које извршава спекулативну нит која је детектовала нарушавање RAW зависности по подацима.

Интеракције између процесорских језгара и *Scheduler*-а са *Squash Arbiter*-ом су представљене на слици 10-4.



Слика 10-4. Везе *Squash Arbiter*-а са *Scheduler*-ом и процесорским језгрима

Псеудокод који објашњава понашање модула *Squash Arbiter* током спекулативног извршавања је сљедећи:

```
if нарушавање зависности по подацима детектовано током циклуса then
   if каснија нит од нити која је изазвала текући squash then
     игнорисати
   else
     започети нови squash са новом нити која га је узроковала
   endif
endif

while текући squash then
   послати сигнал squash од најспекулативније нити
   ка нити која је узроковала squash
endwhile

once завршен squash од нити која га је узроковала
   послати сигнале standby ка наредним нитима
   послати сигнал restart ка нити која је узроковала squash
   squash је сада завршен
endonce
```

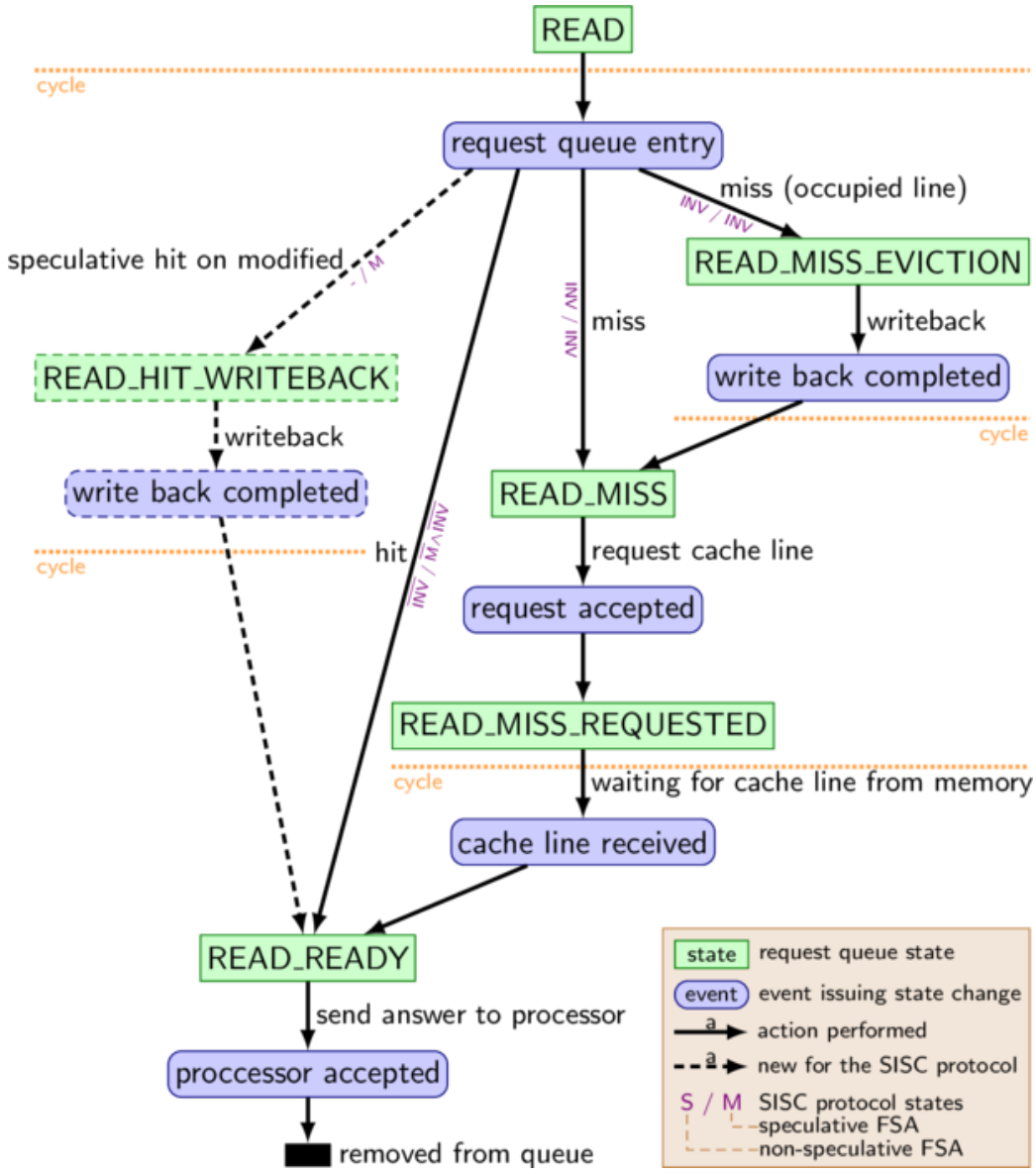
10.5 Имплементација модула *Cache Controller*

У TLM симулаторима и једноставним CLM симулаторима понашање кеш контролера може се једноставно представити додавањем фиксног кашњења (мањег за поготке у кеш меморији и већег за промашаје у кеш меморији). У симулатору UNISIM постојећи модул кеш меморије са одложеним уписом имплементира MESI протокол за кеш кохеренцију и има много комплексније понашање са тачношћу на нивоу циклуса. Након пријема захтјева за упис у кеш меморију или читање из ње од стране процесора током једног циклуса, овај захтјев се поставља у посебан ред за чекање унутар модула кеш меморије. Улази у редовима за захтјеве ће бити провјеравани и прогресивно рјешавани током наредних циклуса. Слике 10-5 и 10-6 приказују како се задовољавају захтјев за читање из кеш меморије и захтјев за упис у кеш меморију, респективно, у модификованом модулу кеш меморије са имплементираним SISC-WI протоколом за спекулацију и кохеренцију.

Када обрађује нови захтјев за читањем из кеш меморије који је постављен у ред за чекање, немодификовани симулаторски модул кеш контролера има неколико опција на основу стања кеш линије као што је приказано на слици 10-5. У случају поготка у кеш меморији приликом операције читања захтјев прелази у стање READ_READY. Ако је дошло до промашаја приликом операције читања и ако је потребно извршити избацивање кеш линије како би се могла учитати одговарајућа кеш линија, захтјев прелази у READ_MISS_EVICTION стање. У супротном, кеш линија је доступна и тада захтјев за читањем из кеш меморије прелази директно у READ_MISS стање.

У односу на кашњење, најгори случај одговара промашају при читању који изазива избацивање постојеће модификоване ријечи из кеш меморије (слика 10-5).

Током циклуса, када се појави захтјев за читање, овај захтјев се само поставља у ред за чекање. У сљедећем циклусу кеш контролер ће провјеравати захтјеве у реду за чекање и, под претпоставком да се ради о промашају током читања који налаже избацивање ријечи из кеш меморије, стање одговарајућег улаза у кеш меморији ће бити постављено у READ_MISS_EVICTION, па се започиње са уписом модификоване ријечи у сљедећи ниво меморије.

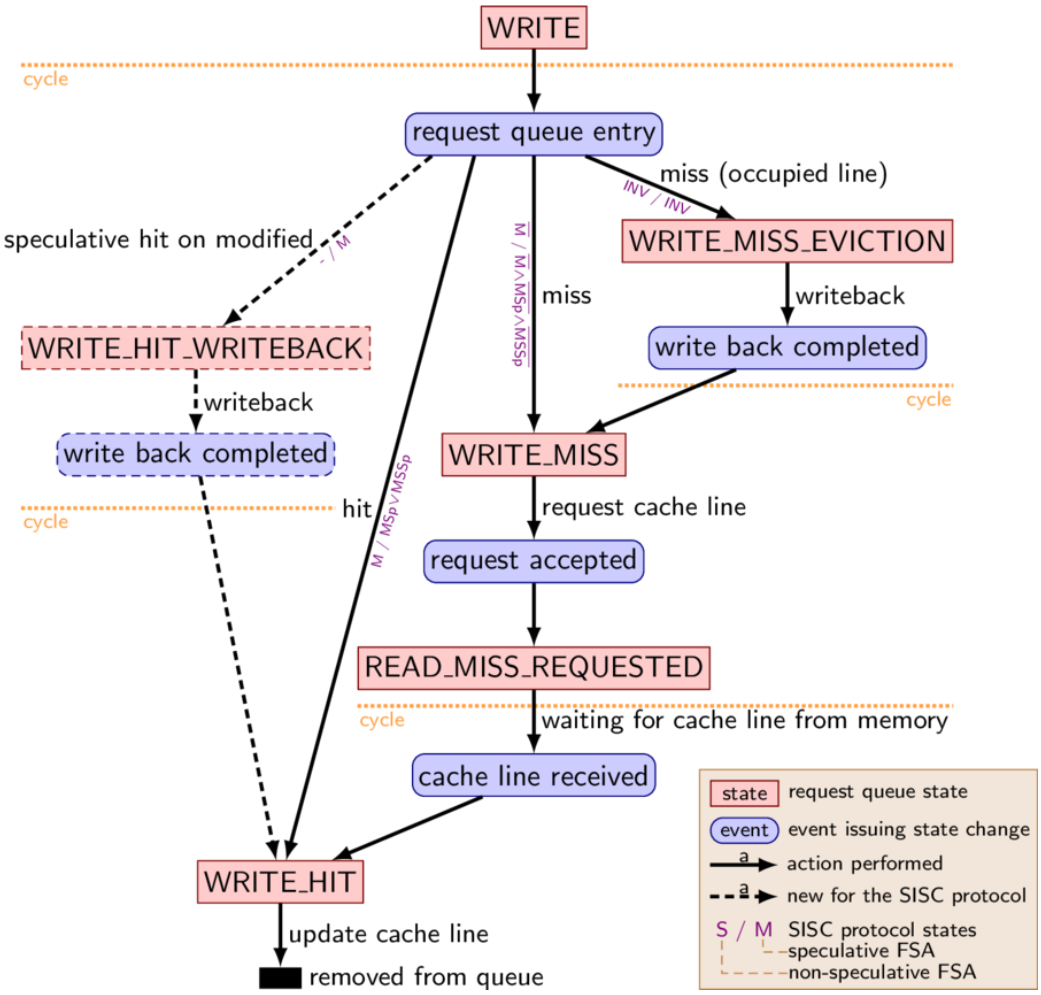


Слика 10-5. Извршавање захтјева за читање ријечи из кеш меморије

Неколико циклуса касније, када се упис у меморију заврши, стање улаза у кеш меморију ће постати READ_MISS (као и да није било претходног избацивања ријечи). Док је у овом стању, кеш контолер ће покушати да од меморије тражи потребну ријеч док се тај захтјев не прихвати, а затим прелази у стање

READ_MISS_REQUESTED. Касније, након већег броја циклуса, кеш меморија ће примити тражену ријеч из меморије, интерно ће је смјестити и прећи ће у стање READ_READY (као што би било да је имао тражену ријеч на самом почетку). Када се једном нађе у стању READ_READY кеш меморија само мора да назад пошаље процесору тражену ријеч и, након што је процесор прихвати, тај захтјев може са сигурношћу бити уклоњен из реда за чекање.

У поређењу са полазним модулом кеш контролера, имплементација протокола SISC-WI додатно захтијева да погодак при операцији читања од стране спекулативних нити за модификоване вриједности у M стању узрокује упис у меморију уводећи додатно READ_HIT_WRITEBACK стање и придружену испрекидану линију. Ако се то не би урадило, ова ријеч би остала у спекулативном стању и можда касније била поништена, а тиме и изгубљена вриједност која припада неспекулативном стању. Испрекидане линије као и лабеле дате у љубичастој боји на слици 10-5 одговарају потребним модификацијама које је неопходно урадити у кеш контролеру приликом операције читања како би се извршила имплементација SISC-WI протокола.



Слика 10-6. Извршавање захтјева за упис ријечи у кеш меморију

Када обрађује нови захтјев за уписом у кеш меморију који је постављен у ред за чекање, немодификован модул кеш контролера има неколико опција на основу стања кеш линије као што је приказано на слици 10-6. У случају поготка у кеш меморији приликом операције уписа, захтјев прелази у стање WRITE_HIT. Ако је дошло до промашаја приликом операције читања и ако је потребно извршити избацивање кеш линије како би се могла уписати одговарајућа кеш линија, захтјев прелази у стање WRITE_MISS_EVICTION. У супротном, кеш линија је доступна и тада захтјев за читањем из кеш меморије прелази директно у стање WRITE_MISS.

Као и у случају захтјева за читањем, најгори случај одговара промашају при упису који изазива избацивање постојеће модификоване ријечи из кеш меморије (слика 10-6). Такође, у поређењу са основним модулом кеш контролера, имплементација протокола SISC-WI додатно захтјева да погодак при операцији читања од стране спекулативних нити за модификоване вриједности у M стању узрокује упис у меморију уводећи додатно стање WRITE_HIT_WRITEBACK и придружену испрекидану линију. Уколико се не би извршио упис у меморију ова ријеч би остала у спекулативном стању и можда касније била поништена, чиме би била изгубљена вриједност која припада неспекулативном стању. Поред тога, случај било неспекулативног или спекулативног поготка приликом операције уписа у кеш меморију за ријечи које нијесу у модификованом стању (нијесу у стању M за неспекулативну нит, односно у стању M, MSp или MSSp за спекулативну нит) ће бити сматран као промашај приликом уписа у кеш меморију. Тако би касније нити биле информисане да ранија нит извршава операцију уписа за дату ријеч па би биле у могућности да изврше детекцију нарушавања RAW меморијске зависности по подацима. Испрекидане линије као и лабеле дате у љубичастој боји на слици 10-6 одговарају потребним модификацијама које је неопходно урадити у кеш контролеру приликом операције уписа како би се извршила имплементација протокола SISC-WI.

10.6 Имплементација заједничке магистрале

Интеграција протокола SISC-WI у симулационо окружење UNISIM подразумијева и прилагођавање постојеће системске магистрале тако да укључи потребне додатне сигнале за подршку TLS-у. Међутим, током фазе интеграције појавио се проблем имплементације децентрализованог механизма арбитрације на магистрали који разрјешава избор могућег добављача између неспекулативне и спекулативних нити за тражену ријеч (глава 7). Постоји неколико важних ставки које ограничавају имплементацију датог механизма у симулатору UNISIM. С обзиром да UNISIM припада CLM симулаторима постојање два уписа на истој контролној линији за маске нити током истог циклуса, као што је то случај код децентрализованог механизма арбитрације на магистрали, представља имплементациони проблем са становишта саме концепције. Такође,

имплементација магистрале са двоструком фреквенцијом је веома скупо решење које је тешко тестирати. Због тога је одлучено да се имплементира централизовани механизам арбитрације на магистрали који ће одлучивати о избору могућег добављача између нити за тражену ријеч.

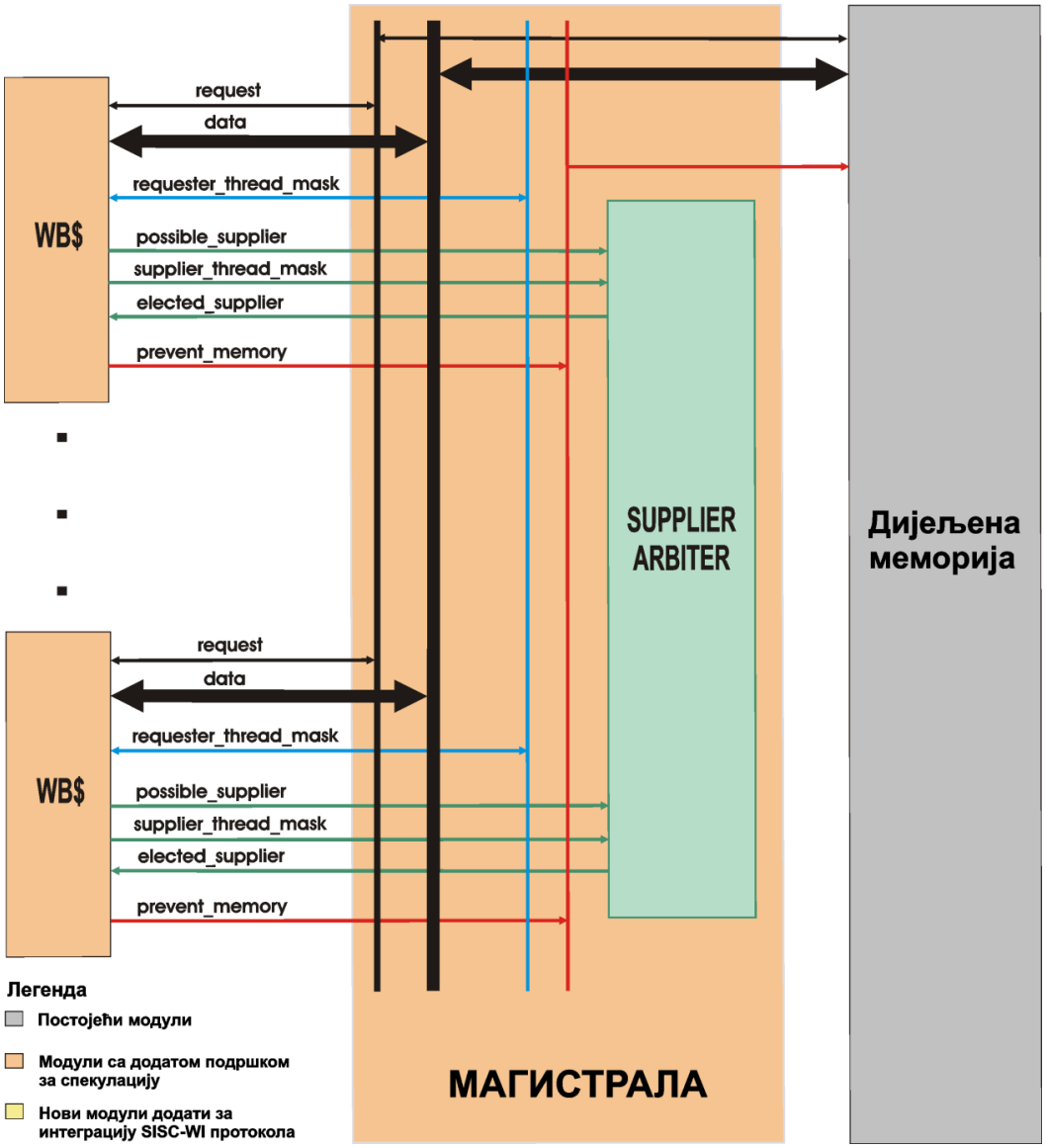
Постојећи модул магистрале који је доступан у симулатору UNISIM подржава MESI протокол кохеренције [CULL1999]. Модул магистрале обухвата сигнале захтјева (одредишна адреса и величина траженог податка), сигнале података (за пренос блока података) и сигнал *Shared*.

За подршку интеграције протокола SISC-WI у симулатор UNISIM потребно је било уклонити *Shared* сигнал из модула магистрале и додати у њега нове сигнале помоћу којих ће се подржати спекулативно извршавање као што су: сигнал за маску нити која захтијева ријеч (*requester_thread_mask*), сигнал за могућег добављача тражене ријечи (*possible_supplier*), сигнал за маску нити која је добављач тражене ријечи (*supplier_thread_mask*), сигнал за изабраног добављача тражене ријечи (*elected_supplier*) и сигнал за превенцију меморије (*prevent_memory*), као што је приказано линијама у боји на слици 10-7.

Сигнал *Requester_thread_mask* се поставља на магистралу заједно са захтјевима на магистрали да би се тако информисали потенцијални добављачи о томе да ли је захтјев издат од стране претходне или сљедеће нити. На основу ове информације потенцијални добављачи ће у потпуности игнорисати захтјев (нпр., наредне нити игноришу захтјеве за спекулативне ријечи) или ће себе идентификовати као могуће добављаче (нпр., претходне нити су могући добављачи за спекулативне ријечи).

С обзиром да више L1 кеш меморија за податке могу бити могући добављачи за један исти захтјев, тада они постављају додатне сигнале на магистрали како би обавјестили *Supplier Arbiter* да изврши избор стварног добављача. Према томе, сигнал *possible_supplier* се поставља на магистралу када дата кеш меморија препозна себе као потенцијалног добављача за тражену кеш ријеч, просљеђујући при томе *Supplier Arbiter*-у и одговарајућу маску нити (*supplier_thread*).

Supplier Arbiter је нова централизована компонента чија је улога да изврши избор стварног добављача између свих потенцијалних добављача чиме се избјегава да више кандидата одговори на исти захтјев. Стога, *Supplier Arbiter* врши поређење свих маски нити које су потенцијални добављачи послали како би на тај начин изабрао кандидата са највећом маском као стварног добављача. Након тога, *Supplier Arbiter* поставља сигнал *elected_supplier* који одговара изабраном кандидату омогућавајући му на тај начин да одговори на захтјев са магистрале за траженим подацима.



Слика 10-7. Модул магистрале са централизованим механизмом арбитрације

Сигнал *prevent_memory* се поставља на магистралу како би се осигурало да меморија игнорише захтјев на магистрали. Овај сигнал се поставља на магистралу у два случаја: 1) када је L1 кеш меморија могући добављач за тражене податке како би се на тај начин постигло да меморија игнорише постављени захтјев; 2) када L1 кеш меморија обезбјеђује спекулативну вриједност да би били сигурни да ниједна спекулативна вриједност неће промијенити неспекулативно стање у меморији.

У наставку ће бити илустрован поредак активације појединих сигнала у предложеној имплементацији магистрале:

- након што нит, било неспекулативна или спекулативна изврши промашај приликом операције читања, њена кеш меморија шаље захтјев за читање на магистралу заједно са маском нити (*requester_thread_mask*).
- након пријема захтјева за читање и маске нити која потражује податак (*requester_thread_mask*), остале нити одређују да ли су раније или касније нити у односу на нит која је поставила захтјев на магистралу.
- уколико касније нити имају тражени податак у спекулативном стању оне неће поставити *possible_supplier* сигнале и идентификовати себе као потенцијалне добављаче. У супротном, касније нити које су потенцијални добављачи постављају сигнал *prevent_memory*, као и сигнале *possible_supplier* заједно са својим маскама (*supplier_thread_mask*).
- уколико раније нити имају тражени податак било у неспекулативном или спекулативном стању оне ће поставити сигнал *prevent_memory*, као и сигнале *possible_supplier* заједно са својим маскама (*supplier_thread_mask*).
- након пријема сигнала *prevent_memory* меморија једноставно игнорише захтјев на магистрали. Тада *Supplier Arbiter* арбитрира између потенцијалних добављача на основу њихових *supplier_thread* маски бирајући стварног добављача за тражене податке тако што поставља сигнал *elected_supplier* само за ову нит.
- кеш меморија изабране нити ће поставити у ред за чекање захтјев за читање који је на магистралу поставила дата нит и одговориће на исти током сљедећих циклуса. Када изабрана нит касније постави тражени податак на магистралу, уколико се ради о спекулативном податку, изабрана нит ће поставити и сигнал *prevent_memory* како би спријечила меморију да упише спекулативне податке. На крају, нит која је поставила захтјев на магистралу уписује код себе тражене податке након што их прочита са магистрале.

Глава 11 Закључак

У прошлој декади је дошло до радикалних промјена у парадигмама пројектовања процесора које су проузроковане све мањим потенцијалима за даље искоришћење паралелизма на нивоу инструкције, као и значајним проблемима у домену потребне снаге и дисипације. Ови трендови су резултовали појавом вишејезгарних или CMP процесора. Присуство више језгара је омогућило коришћење паралелизма на нивоу нити, али ови ресурси нијесу могли бити искоришћени при извршавању огромног броја секвенцијалних апликација. Због тога је рјешење нађено у спекулативном паралелном извршавању нити (TLS), чак и ако има потенцијалних зависности по подацима, уз постојање одређене подршке која би детектовала и разрјешавала зависности остварене у вријеме извршавања.

Тај проблем је изазвао велику пажњу истраживачке заједнице, па је предложено десетак академских и комерцијалних система са подршком за TLS. У овом раду је направљена широка и веома детаљна компаративна анализа постојећих система са циљем препознавања потенцијала за даљи допринос. Након упознавања са архитектуром система и основним пројектним рјешењима, посебно су размотрени елементи хардверске и софтверске подршке за спекулацију на регистарском и меморијском нивоу. Направљени су и предлози класификације ових система из та два аспекта да се они боље разумију.

У погледу регистарске комуникације системи су класификовани по начину организације скупа регистара и начину повезивања, а затим су подробно анализирани регистарски комуникациони механизми, технике опоравка након погрешне спекулације, перформансе и скалабилност.

Са аспекта спекулативне меморијске комуникације системи су класификовани према организацији меморијске хијерархије. Затим је размотрена сложеност хардверске и софтверске подршке за спекулацију и уочени су неки потенцијални

узроци за деградацију перформанси као што су неуједначен саобраћај при завршетку нити и неприлагођен алгоритам замјене у кеш меморији.

Анализа је показала да поједина постојећа рјешења CMP процесора са TLS подршком захтијевају знатне додатне хардверске и софтверске ресурсе за обављање специфичних операција током спекулативног извршавања секвенцијалних апликација. С обзиром да велики дио хардверске и/или софтверске подршке за TLS у CMP процесору остаје неискоришћен приликом извршавања правих паралелних апликација или истовременог извршавања више апликација, постављен је циљ да се дефинише предлог CMP система који има једноставнију подршку за TLS како би био постигнут што бољи однос цијена/перформансе и који би се отклониле уочене неефикасности.

Предложени CMP систем за спекулативно извршавање нити припада групи CMP система са основном генеричком архитектуром надграђеном подршком и за регистарску и за меморијску комуникацију. Он се састоји од четири процесорска језгра са приватним L1 кеш меморијама за податке и инструкције повезаним заједничком магистралом са заједничком L2 кеш меморијом. Преко ове магистрале се обавља спекулативна комуникација на меморијском нивоу. Регистарска комуникација између процесорских језгара током спекулативног извршавања се обавља преко посебне заједничке магистрале. Спекулативна комуникација и на регистарском и на меморијском нивоу се обавља на принципима „*snoopy*“ протокола чија сложеност практично не зависи од броја процесорских језгара, па се у зависности од пропусности и технолошких параметара обје заједничке магистрале може повећати број процесорских језгара.

Спекулативна паралелизација у предложеном CMP систему се спроводи на нивоу петљи, па спекулативне нити одговарају појединим итерацијама петље. Идентификација нити се врши уз помоћ посебне софтверске компоненте – бинарног анотатора, који ради над секвенцијалним извршним кодом чиме је избјегнута потреба за поновним превођењем изворног кода. Бинарни анотатор има и посебну одговорност да за потребе протокола који контролише регистарску комуникацију изврши класификацију инструкција уписа у регистре чије се вриједности током извршавања могу мијењати у петљи. Спекулативно произведене вриједности се држе у приватној L1 кеш меморији, док заједничка L2 кеш меморија чува секвенцијално стање апликације. Тек када нит постане неспекулативна вриједности могу бити пренесене у L2 кеш меморију. Спекулативна нит може да чита неспекулативне податке од неспекулативне нити или од претходних или наредних спекулативних нити. Предложено рјешење укључује подршку за динамичко препознавање нарушавања зависности по подацима и опоравак након погрешне спекулације.

Предложена су два протокола за регистарску комуникацију: основна варијанта - SIC протокол и проширена варијанта – ESIC протокол. У поређењу са сличним приступима из литературе, предложени регистарски протоколи имају за циљ да буду једноставнији и скалабилнији, као и да имају бољи однос

цијена/перформансе. Квалитативно поређење протокола SIC и ESIC је указало на постојање потенцијалног добитка на перформансама у ESIC у односу на SIC, а који је посљедица смањеног блокирања нити потрошача. Како је ESIC протокол заснован на спекулативном прослеђивању потенцијално сигурних вриједности регистара, овај добитак директно зависи од вјероватноће да се прослеђена, потенцијално сигурна вриједност регистра испостави као коначна. У супротном, погрешна спекулација изазива поништавање нити што може имати негативан ефекат на перформансе. Да би се то избјегло, користи се информација из профајлера о вјероватноћи успјешне спекулације.

У кратком квалитативном поређењу са подршком за регистарску комуникацију у I-ACOMA показало се да хардверска подршка за SIC и ESIC протоколе веома смањују величину локалног каталога и да не захтијева увођење додатне логике у сваком процесорском језгру за провјеру доступности регистра и статуса посљедње копије. Такође, у протоколима SIC и ESIC се користи механизам дистрибуираног одлучивања за ефикасније проналажење најближе претходне нити. Осим тога, број инвалидационих промашаја за константне регистре у SIC и ESIC протоколима је смањен коришћењем „*read-snarfing*“ технике. На крају, уочава се да су SIC и ESIC протоколи више скалабилнији са становишта комплексности зато што величина локалног каталога и хардверска подршка остају скоро константни са повећањем броја процесорских језгара.

Предложене су и двије варијанте меморијског протокола које интегришу одржавање кохеренције кеш меморија и подршку за спекулацију. Прва варијанта, SISC-WI протокол, је заснована на стратегији поништавања и користи 13 стања за праћење статуса податка. SISC-WI протокол омогућава само комуникацију међу нитима коју је покренуо потрошач.

Да би се смањио број инвалидационих промашаја разрађена је и верзија протокола, SISC-WI-RS, која врши истовремену валидацију поништених копија. Друга варијанта, SISC-WU протокол, је заснована на стратегији ажурирања и користи 9 стања за праћење статуса податка. Свакој ријечи у L1 кеш меморији је додат један бит који означава застарјелост ријечи. Међунитна меморијска комуникација у овој варијанти протокола може бити иницирана и од стране потрошача и од стране произвођача.

У свим варијантама меморијских протокола је предложен и механизам дистрибуиране арбитрације на магистрали за избор одговарајућег достављача података на захтјев неке нити.

Такође, све три варијанте меморијских протокола користе прилагођени алгоритам замјене који приликом избора ријечи за замјену узима у обзир њено стање као примарни критеријум, умјесто уобичајеног критеријума историје приступа коришћеног код свих других система. У случају да се више ријечи налази у истом стању, за избор између њих алгоритам замјене користи, као секундарни критеријум, стандардни начин који је заснован на временској локалности или на случајном избору. Спекулативној нити је дозвољено да из L1

кеш меморије избаци модификовану потврђену ријеч која је једина копија приликом операције спекулативног читања односно писања за дату ријеч. Поред тога, све ријечи које нијесу биле спекулативно прочитане или измијењене могу се сматрати кандидатима за замјену, чиме се избјегава заустављање спекулативног извршавања.

Као и у случају регистарских протокола, прецизно поређење SISC протокола са неким другим механизмима меморијске комуникације је тешко због различитих система на којима је примијењена и начина реализације. Међутим, квалитативно поређење са рјешењима са сличном подршком за спекулативно извршавање, са становишта неуједначеног саобраћаја приликом завршетка нити, показује да SISC протоколи задржавањем неспекулативних података у L1 кеш меморијама приликом завршетка нити смањују саобраћај на интерконекционој мрежи и тиме минимизују кашњење у иницијализацији нових нити.

Да би се омогућила имплементациона и евалуациона анализа предложеног система, појавила се потреба за симулационим окружењем. С обзиром да су симулационе платформе за овакве системе неодговарајуће, недоступне или неадекватно подржане један од важних циљева истраживања је био и генерисање новог, погодног симулационог окружења за спекулативне CMP процесоре.

Као основа је изабран модуларни симулатор UNISIM који подржава симулацију на нивоу циклуса. С обзиром на сложеност укупне предложене подршке за TLS за имплементацију је одабран SISC-WI протокол, као њен најсложенији дио. Библиотека компоненти симулатора UNISIM садржи неколико модула који су модификовани како би задовољили захтјеве SISC-WI протокола. Извршена је параметризација модула кеш меморије са одложеним уписом да би се омогућила симулација L1 кеш меморије за податке са величином линије која одговара једној ријечи. За имплементацију SISC-WI протокола је искоришћен постојећи MESI протокол за кохеренцију. Извршене су модификације одговарајућег коначног аутомата стања и кеш контролера како би он могао реаговати на нове догађаје који одговарају новим прелазима између стања, као и на догађаје који су посљедица спекулативне природе SISC-WI протокола. Поред овога, за потребе имплементације потребних сигнала за TLS извршена је и модификација постојећег модула системске магистрале.

С обзиром да симулатор UNISIM није подржавао MIPS процесор, имплементиран је модул за основну MIPS архитектуру, а затим је извршено проширење са једног на четири процесорска језгра који су повезани системском магистралом. Како би се у потпуности подржао механизам спекулативног извршавања нити и интеграција SISC-WI протокола у симулатор UNISIM додата су и три нова модула: распоређивач спекулативних нити на процесорска језгра, модул који детектује нарушавање зависности по подацима и врши поништавање нити и модул који бира добављача за тражени податак између више нити. Поред тога, важно је истаћи да модул за детекцију и разјешавање погрешне спекулације

успјешно разрјешава проблеме који су везани за поништавање нити, а за које у доступној литератури нијесу нађена рјешења.

За прецизну провјеру исправности спекулације било је потребно да се имплементира експлицитна спекулација, па је развијена инфраструктура за крос-компајлер заснована на радном оквиру *crosstool-NG* који је специјализован за израду ***gcc/glibc*** крос-компајлера. Предложени полуаутоматски приступ поставља одређене позиве функцијама за генерисање нове нити односно завршетак нити на нивоу изворног кода, слично као код POSH преводиоца. Превођење изворног кода се врши помоћу *gcc* крос-компајлера генерисаног од стране *crosstool-NG* алата који преводи функцијске позиве за генерисање нове нити и завршетак нити као скокове на рутине у симулатору које их одрађују. Симулатор обрађује ове позиве функција да би на основу њих раздијелио нити на слободна процесорска језгра, односно да би управљао операцијама извршавања нити, као и прослеђивањем неспекулативног статуса наредној спекулативној нити након завршетка неспекулативне нити.

Овај рад отвара више нових праваца истраживања која би у будућности могла бити спроведена. Како је направљено погодно симулационо окружење и потребна инфраструкура за паралелизацију и с обзиром да је концепт имплементације провјерен кроз интеграцију најсложенијег дијела подршке за TLS, следећи корак би био имплементација комплетне преостале подршке и њена свестрана евалуациона анализа.

Током рада на меморијским протоколима појавила се и идеја о увођењу комуникације иницране од стране произвођача приликом операције неспекулативног или спекулативног уписа у кеш ријеч. Ова надградња протокола SISC-WI би спекулативно прослеђивала уписане ријечи без експлицитног захтјева каснијим нитима и тиме предуприједила њихове евентуалне, накнадне захтјеве. Тако би се смањио број инвалидационих промашаја и одговарајуће чекање нити. У случају спекулативних уписа ове вриједности би биле потврђене када нит која их је произвела постане неспекулативна. За потребе ове надградње треба да се дода још један сигнал магистрале, као и још једно стање, али то не би повећало број битова за кодирање стања.

Предложени систем је заснован на заједничкој магистрали чији пропусни опсег може да подржи повезивање десетак језгара. Евентуално повећање броја процесорских језгара на чипу неминовно налаже коришћење неке друге интерконекционе мреже, па би један правац истраживања био усмјерен на разматрање примјене скалабилнијих топологија мреже, било кроз хијерархије магистрала, било кроз мреже директне повезаности. Поред тога, у овом случају протоколи за регистарску и меморијску комуникацију би морали бити прилагођени типу изабране мреже.

На крају, природно се јавља потреба за унапређењем постојећег симулационог окружења коришћењем нових сервиса за процјену дисипације, утрошка простора на чипу и времена приступа кеш меморији. Поред тога, модуларност овог система

дозвољава да се он допуни потребним модулима који би омогућавали симулацију ширег спектра CMP система са TLS подршком. Тако би могла да се спроведе једна шира компаративна анализа различитих система која се не може наћи у отвореној литератури.

ЛИТЕРАТУРА

[AGAR2000] V. Agarwal, M. S. Hrishikesh, S. W. Keckler, D.Burger, "Clock Rate versus IPC: The End of the Road for Conventional Microarchitectures", Proc. of the 27th Annual Int. Symp on Comp. Architecture, June 2000.

[ALIP2012] M. Alipour1, H. Taghdisi, Effect of Thread-level Parallelism on the Performance of Optimum Architecture for Embedded Applications, International Journal of Embedded Systems and Applications (IJESA) Vol.2, (2012) 15-24.

[AMAR1995] S. Amarasinghe et al., "Hot compilers for future hot chips," in Hot Chips VII, Stanford, CA, Aug. 1995. http://www.hotchips.org/archives/

[AUGU2007] D. I. August, J. Chang, S. Girbal, D. Gracia-P'erez, G. Mouchard, D. Penry, O. Temam, N. Vachharajani. Unisim: An Open Simulation Environment and Library for Complex Architecture Design and Collaborative. Computer Architecture Letters, September 2007.

[BARL2003] N. D. Barli, et al., A Register Communication Mechanism for Speculative Multithreading Chip Multiprocessors, Symposium on Advanced Computer Systems and Infrastructures, (2003) 275-282.

[BARL2004] N.D. Barli, Designing NEKO: A Speculative Multithreading Chip MultiProcessor, Ph.D. thesis, The University of Tokyo, 2004.

[BELL2006] I. Bell, N. Hasasneh, C. Jessope, Supporting Microthread Scheduling and Synchronization in CMPs, International Journal of Parallel Programming, Vol. 34 Iss. 4, (2006) 343-381.

[BLUM1996] W. Blume, R. Doallo, R. Eigenmann, J. Grout, J. Hoeflinger, T. Lawrence, J. Lee, D. Padua, Y. Paek, B. Pottenger, L. Rauchwerger, P. Tu, Parallel Programming with Polaris, IEEE Computer, (1996) 78–82.

[BREA1996] S. Breach, T. Vijaykumar, S. Gopal, J. Smith, and G. Sohi. Data memory alternatives for multiscalar processors. Technical Report CS-TR-97-1344, University of Wisconsin, CS Department, Nov 1996.

[BURG1997] D. Burger, T.A. Austin, "The SimpleScalar Tool Set, Version 2.0", Technical report #1342, University of Wisconsin-Madison Computer Science, June 1997.

[CAI2003] L. Cai, D. Gajski, Transaction Level Modeling: An Overview, in proceedings of the Int. Conference on HW/SW Codesign and System Synthesis (CODES-ISSS), Oct. 2003, pp. 19–24

[CARL2003] L. P. Carloni, A. L. Sangiovanni-Vincentelli, "On-Chip Communication Design:Roadblocks and Avenues", CODECS+ISSS '03,October 2003, Newport Beach, California, USA.

[CINT2000] M. Cintra, J. F. Martýnez, and J. Torrellas, "Architectural Support for Scalable Speculative Parallelization in Shared-Memory Multiprocessors", In Proc. 27th Annual Intl. Symp. on Computer Architecture, pages 13–24, June 2000.

[CODR1998] L. Codrescu, S. Wills, The AMA Correlated Value Predictor, Pica Group Technical Report 10-98, (1998).

[CODR1999] L. Codrescu, S. Wills, On Dynamic Speculative Thread Partitioning and the MEM-Slicing Algorithm, Proc. Int'l Conf. Parallel Architectures and Compilation Techniques (PACT99), (1999) 40-46.

[CODR2001] L. Codrescu, D. Scott Wills, Architecture of the Atlas Chip-Multiprocessor: Dynamically Parallelizing Irregular Applications, IEEE Transactions on Computers, Vol. 50, No. 1, (2001) 67-82.

[CROS2014] Crosstool-NG. Available at: http://crosstool-ng.org/, Accessed: October 2014.

[CULL1999] D. E. Culler, J. P. Singh, A. Gupta, "Parallel Computer Architecture", Morgan Kaufmann Publishers, Inc., San Francisco, California, 1999.

[EDAH2000] M. Edahiro, S. Matsushita, M. Yamashina, N. Nishi, A Single-Chip Multiprocessor for Smart Terminals, IEEE Micro, Vol. 20, No. 4, (2000) 12-30.

[EGGE1989] S.J. Eggers, R.H. Katz, "Evaluating the performance of four snooping cache coherency protocols", Proceedings of the 16th International Symposium On Computer Architecture, pages 2-15, 1989.

[EMER2002] J. Emer, P. Ahuja, E. Borch, A. Klauser, C.-K. Luk, S. Manne, S. S. Mukherjee, H. Patil, S. Wallace, N. Binkert, R. Espasa, T. Juan. Asim: A performance model framework. IEEE Computer, 0018-9162:68{76, February 2002.

[FRAB2004] A. Fraboulet, T. Risset, A. Scherrer, Cycle Accurate Simulation Model Generation for SoC Prototyping, Computer Systems: Architectures, Modeling, and Simulation Lecture Notes in Computer Science, Volume 3133, 2004, pp 453-462

[FRAN1996] M. Franklin and G. S. Sohi. ARB: A hardware mechanism for dynamic reordering of memory references. IEEE Transactions on Computers, 45(5):552–571, May 1996.

[GOPA1998] S. Gopal, T. N. Vijaykumar, J. E. Smith, G. S. Sohi, "Speculative Versioning Cache", Proceeedings of the Fourth Intrenational Symposium on High-Performance Computer Architecture, Las Vegas, NV, February 1998.

[HAMM1997] L. Hammond, B. A. Nayfeh, K. Olukotun, "A Single-Chip Multiprocessor", IEEE Computer Special Issue on "Billion-Transistor Processors", September 1997, pp. 79-85.

[HAMM2000] L. Hammond et al., "The Stanford Hydra CMP", IEEE Micro, pp. 71-84, 2000.

[HENN2012] Hennessy J., Patterson D., "Computer Architecture: A Quantitative Approach", Morgan Kaufmann Publishers, 2012.

[HUAN1998] J. Huang and D. J. Ljilja, "An Efficient Strategy for Developing a Simulator for a Novel Concurrent Multithreaded Processor". In Proceedings of the 6th International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems, pp. 185-191, July 1998.

[HUNG2002] L.D. Hung, H. Miura, C. Iwama, D. Tashiro, N.D. Barli, S. Sakai, H. Tanaka, A Hardware/Software Approach for Thread Level Control Speculation, IPSJ SIG Technical Reports 2002-ARC-149, Vol.2002, No.12, (2002) 67-72.

[JOHN1990] M. Johnson, Superscalar Microprocessor Design, Prentice Hall, 1990.

[KECK1998] S. Keckler et al., Exploiting Fine-Grain Thread-Level Parallelism on the MIT ALU Processor", Proc. 25th Ann. Int'l Symp. Computer Architecture, ACM press, Jun.-Jul. 1998, pp. 306-317.

[KHAN2012] A. Khan, M. Vijayaraghavan, S. Boyd-Wickizer, Arvind, Fast and Cycle-Accurate Modeling of a Multicore Processor, ISPASS, page 178-187. IEEE Computer Society, USA, 2012.

[KNIG1986] T. Knight, An Architecture for Mostly Functional Languages, In ACM Lisp and Functional Programming Conf., (1986) 500-519.

[KRIS1998] V. Krishnan, J. Torrellas, Executing Sequential Binaries on a Multithreaded Architecture with Speculation Support, Workshop on Multi-Threaded Execution, Architecture and Compilation (MTEAC'98), (1998).

[KRIS1999] V. Krishnan, J. Torrellas, A Chip-Multiprocessor Architecture with Speculative Multithreading, IEEE Transactions on Computers, Vol. 48, No. 9, (1999) 866-880.

[MADR2008] C. Madriles, C. Garcıa-Quinones, J. Sanchez, P. Marcuello, A. Gonzalez, D.M. Tullsen, H. Wang, J.P. Shen, Mitosis: A Speculative Multithreaded Processor Based on Precomputation Slices, IEEE

Transactions on Parallel and Distributed Systems, Vol. 19, No. 7, (2008) 914-925.

[MARC1998] P. Marcuello, A. Gonzalez, J. Tubella, Speculative Multithreaded Processors, Proc. 12th Int'l Conf. Supercomputing (ICS), (1998) 77-84.

[MARC1999] P. Marcuello, J. Tubella, A. Gonzalez, Value Prediction for Speculative Multithreaded Architectures, In Proc. of the 32th. Int. Conf. on Microarchitecture, (1999) 230-236.

[MARC2003] P. Marcuello, Speculative Multithreaded Processors, Ph.D. thesis, Universitat Politecnica de Catalunya, Barcelona, Spain, 2003.

[MART2005] Milo M. K. Martin, Daniel J. Sorin, Bradford M. Beckmann, Michael R. Marty, Min Xu, Alaa R. Alameldeen, Kevin E. Moore, Mark D. Hill, David A. Wood, Multifacet's General Execution-driven Multiprocessor Simulator (GEMS) Toolset, Newsletter, ACM SIGARCH Computer Architecture News - Special issue: dasCMP'05, Volume 33 Issue 4, November 2005 Pages 92 – 99, ACM New York, NY, USA.

[MATS2000] S. Matsushita et al., Merlot: A Single-Chip Tightly Coupled Four-Way Multi-Thread Processor, Proc. Cool Chips III Symp., (2000) 63-74.

[NAYF1996] Nayfeh, B.A., Hammond L., and Olukotun K., "Evaluation of design alternatives for a multiprocessor microprocessor, In Proc. 23rd ISCA, Philadelphia, PA, may 22-24, pp. 67-77. ACM Press, New York.

[OANC2008] C.E. Oancea, A. Mycroft, Software Thread-Level Speculation - An Optimistic Library Implementation, Proceedings of the 1st International Workshop on Multicore Software Engineering, ACM New York, NY, USA, (2008) 23-32.

[OHSA2005] T. Ohsawa, M. Takagi, S. Kawahara, S. Matsushita, Pinot: Speculative Muti-threading Processor Architecture Exploiting Parallelism over a wide Range of Granularities, Proc. 38th Int'l Symp. Microarchitecture (MICRO), (2005) 81-92.

[OLUK2007] K. Olukotun, L. Hammond, J. Laudon, Chip Multiprocessor Architecture: Techniques to Improve Throughput and Latency, Synthesis Lectures on Computer Architecture #3, Morgan & Claypool, 2007.

[OOI2001] C.L. Ooi, S.W. Kim, I. Park, R. Eigenmann, B. Falsafi, T.N. Vijaykumar, Multiplex: Unifying Conventional and Speculative Thread-Level Parallelism on a Chip Multiprocessor, ICS-15, (2001) 368-380.

[OPLI1997] Jeffery Oplinger, David Heine, Shih-Wei Liao, Basem A. Nayfeh , Monica Lam and Kunle Olukotun, "Software and Hardware for Exploiting Speculative Parallelism with a Multiprocessor", Stanford University Computer Systems Lab Technical Report CSL-TR-97-715, February 1997.

[ORTE2004] P. M. Ortego, P. Sack, "SESC: SuperESCalar Simulator", Technical Report, UIUC, December 2004.

[ORTE2004] P. M. Ortego, P. Sack, “SESC: SuperESCalar Simulator”, Technical Report, UIUC, December 2004.

[OSCI2003] SystemC, OSC Initiative. Technical report, OSCI, 2003.

[PACK2009] V. Packirisamy, A. Zhai, W.-C. Hsu, P.-C. Yew, T.-F. Ngai, Exploring Speculative Parallelism in SPEC2006, 2009, pp. 77-88.

[PATT2007] D. A. Patterson, J. L. Hennessy, Computer Organization and Design: the Hardware/Software Interface, Morgan Kaufman, 2007.

[PERE2006] D. Gracia-Perez, G. Mouchard, O. Temam. MicroLib: A case for the quantitative comparison of micro-architecture mechanisms. Pp. 43:54, December 2004.

[PRAS1996] S. Prasad, Multithreading Programming Techniques, 1996, McGraw-Hill, Inc., New York, NY, USA.

[RADU2005] M. B. Radulović, M. V. Tomašević, “A Proposal for Register-level Communication in a Speculative Chip Multiprocessor“, XLIX ETRAN Conference, June 2005, Budva, Serbia and Montenegro.

[RADU2005a] M. B. Radulović, M. V. Tomašević, “An Aggressive Register-level Communication in a Speculative Chip Multiprocessor”, IEEE EUROCON 2005 Conference, November 21-24, pp. 689-692, Belgrade, Serbia and Montenegro, 2005.

[RADU2006] Radulović, M. B., Tomašević, M.V.: “An Analysis of Speculation and Coherence Protocols in CMPs”, - Proc’ 50th ETRAN Conference, Belgrade, 6-8 June, 2006, Vol III, pp. 38-41.

[RADU2007] Radulović, M. B., Tomašević, M.V., ”On Reducing Overheads in CMP TLS Integrated Protocols”, - IPSI Transactions On Internet Research, Volume 3, Number 1, January 2007, pp. 11-17.

[RADU2007a] Radulović, M. B., Tomašević, M.V., ”Towards an Improved Integrated Coherence and Speculation Protocol”, - IEEE EUROCON 2007 Conference, September 9-12, pp. 405-412, Warsaw, Poland, 2007.

[RADU2008] Radulović, M. B., Girbal, S., Tomašević, M. V., ”Simulation Support for Speculative Multithreading Protocols”, - ACACES 2008, July 2008, Italy, pp. 283-286.

[RADU2009] Radulović, M. B., Girbal, S., Tomašević, M. V., ”Evaluating the SISC TLS protocol through Structural Simulation”, - ACACES 2009, July 2009, Spain, pp. 319-322.

[RADU2014] Radulović, M. B., Tomašević, M. V., Milutinović, V. M.: Register-level Communication in Speculative Chip Multiprocessors, - Advances in Computers, Vol. 92, January 2014, pp. 1-66. (ISSN 0065-2458) (http://dx.doi.org/10.1016/B978-0-12-420232-0.00001-5)

[RAFE2003] Rafeeq Ur Rehman, Christopher Paul, The Linux Development Platform: Configuring, Using, and Maintaining a Complete Programming Environment Rafeeq Ur Rehman, Christopher Paul, 2003 Pearson

Education, Inc., Publishing as Prentice Hall PTR Upper Saddle River, New Jersey

[ROSE1995] M. Rosenblum, S. Herrod, E. Witchel, and A. Gupta, “The SimOS approach,” IEEE Parallel and Distributed Technology, vol. 4, no. 3, 1995.

[ROTE1997] E. Rotenberg, Q. Jacobson, Y. Sazeides, J. Smith, Trace Processors, 30th International Symposium on Microarchitecture, (1997) 138–148.

[ROTE1999] E. Rotenberg, J. Smith, Control Independence in Trace Processors, 32nd International Symposium on Microarchitecture, (1999) 4-15.

[SASS2005] Peter G. Sassone, D. Scott Wills, Scaling Up the Atlas Chip-Multiprocessor, IEEE Transactions on Computers, Vol. 54, No. 1, (2005) 82-87.

[SOHI1995] G.S. Sohi, S. Breach, T.N. Vijaykumar, Multiscalar Processors, Proceedings of the 22nd ISCA, (1995) 414-425.

[STEF2000] J.G. Steffan, C.B. Colohan, A. Zhai, T.C. Mowry, “A Scalable Approach to Thread-Level Speculation”, ISCA 2000, Vancouver, Canada, June 2000.

[STEF2005] J. Gregory Steffan, Christopher B. Colohan, Antonia Zhai, Todd C. Mowry, The STAMPede Approach to Thread-Level Speculation, ACM Transactions on Computer Systems, 23(3), (2005) 253-300.

[SUDH2000] S. Sudharsanan, "MAJC-5200: A High Performance Microprocessor for Multimedia Computing", in Proc. IPDPS Workshops, (2000) 163-170.

[SUND1997] K. Sundararaman, M. Franklin, Multiscalar Execution along a Single Flow of Control, International Conference on Parallel Processing, (1997) 106-113.

[TANA2009] Y. Tanaka, H. Ando, Reducing Register File Size through Instruction Preexecution Enhanced by Value Prediction, IEEE International Conference on Computer Design, (2009) 238–245.

[TARJ2006] D. Tarjan, S. Thoziyoor, and N. P. Jouppi. Cacti 4.0 technical report. HP Labs, 2006.

[TOMA1993] M. Tomašević and V. Milutinović, Tutorial on the Cache Coherence Problem in Shared-Memory Multiprocessors: Hardware Solutions, IEEE Computer Society Press, Los Alamitos, Calif., 1993.

[TOMA2005] Tomašević, M. V., Radulović, M. B.: “Speculative Chip Multiprocessors”, - Proceedings of the Workshop Contemporary Mathematics, Physics and Biology, 8-9 September 2005, University of Montenegro, Podgorica, pp. 168-186.

[TOMA2005a] Tomašević M., Puzović N., Leković S., “Analysis and Improvement of Replacement Algorithms in SMP cache memory systems”, Proceedings of the ACACES (Advanced Computer Architecture and Compilation for Embedded Systems), L’Aquila, July 2005.

[TORR2011] Josep Torrellas, “Thread-Level Speculation”, Encyclopedia of Parallel Computing, Springer Science+Business Media LLC, May 2011.

[TREM2000] M. Tremblay et al., The MAJC architecture: A Synthesis of Parallelism and Scalability, IEEE Micro, (2000) 12-25.

[TSAI1999] J-Y. Tsai, J. Huang, C. Amlo, D. J. Lilja, P-C. Yew, “The Superthreaded Processor Architecture”, IEEE Transactions on Computers, Special Issue on Multithreaded Architectures and Systems, pp. 881-902, September, 1999.

[VACH2007] M. Vachharajani, N. Vachharajani, D. A. Penry, J. A. Blome, D. I. August. Microarchitectural Exploration with Liberty. In Proc. of the 34th Annual Int. Symp. on Microarchitecture, Austin, Texas, USA., December 2001.

[VAJA1997] S. Vajapeyam, T. Mitra, Improving Superscalar Instruction Dispatch and Issue by Exploiting Dynamic Code Sequences, 24th International Symposium on Computer Architecture, (1997) 1-12.

[VEEN1994] J. Veenstra, R. Fowler. MINT: A Front End for Efficient Simulation of Shared-Memory Multiprocessors. In MASCOTS’94, pp. 201-207, January 1994.

[WALL1993] D. W. Wall, “Limits of Instruction-Level Parallelism”, Digital Western Research laboratory, WRL Research Report 93/6, November 1993.

[WARG2006] F. Warg, Ph.D. thesis, Techniques to Reduce Thread-Level Speculation Overhead, Department of Computer Science and Engineering, Chalmers University of Technology, 2006.

[WEI2006] Wei Liu, James Tuck, Luis Ceze, Wonsun Ahn, Karin Strauss, Jose Renau and Josep Torrellas, “POSH: A TLS Compiler that Exploits Program Structure”, ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming (PPoPP), March 2006.

[YANA2003] Y. Yanagawa, et al., “Complexity Analysis of a Cache Controller for Speculative Multithreading Chip Multiprocessors”, HiPC - International Conference on High Performance Computing, December 2003.

[ZAHR2003] M.M. Zahran, “On Cache Memory Hierarchy for Chip Multiprocessor”, ACM SIGARCH: Volume 31, Issue 1, pp. 39-48, March 2003.

[ZIAR2009] L. Ziarek, S. Jagannathan, M. Fluet, U,A. Acar, Speculative N-Way Barriers, ACM New York, NY, USA, (2009) 1-12.

БИОГРАФИЈА АУТОРА

Милан Бранков Радуловић је рођен 24.02.1968. г. у Никшићу гдје је завршио основну, нижу музичку и средњу школу. Добитник је дипломе "Луча" за постигнути успјех током основног и средњег образовања, као и дипломе ђака генерације. Поред тога, добитник је награда за прва мјеста на општинским и републичким такмичењима из физике. На Електротехничком факултету Универзитета Црне Горе у Подгорици дипломирао је 1992. г. са просјечном оцјеном 9,50, а 1996.г. је завршио постдипломске студије са просјечном оцјеном 10 и одбранио магистарску тезу. Од стране Скупштине Општине Никшић је 1996.г. награђен дипломом "Луча" за постигнути успјех током основних и постдипломских студија.

Од 1993. до 1996. г. био је запослен као стручни сарадник на Електротехничком факултету у Подгорици. Током 1996. г. је био и стипендиста Министарства просвјете и науке Републике Црне Горе, Сектор за науку. Са стипендијом British Foundation for the Citizens of former Yugoslavia је боравио 1997. г. у звању Visiting Research Fellow на Engineering Department, Универзитета у Warwick-у, UK. У периоду од 1998. до 2005 г. био је запослен у Центру информационог система Универзитета Црне Горе у Подгорици у звању главни систем инжењер за комуникације. У периоду 1998.-2002. г. је радио у звању хонорарног асистента на Вишој рачунарској школи Електротехничког факултета у Подгорици. У току 2001. г., са стипендијом фондације Универзитета Европе Coimbra Group је боравио у звању Visiting Research Fellow на Faculty of Science, Универзитет у Гранади, Шпанија. У два наврата, 2002. г. са стипендијом Владе Републике Италије и 2003-2004 г. са стипендијом IMG, TEMPUS фондације из Брисела, боравио је у звању Visiting Research Fellow на Information Engineering Department, Универзитет у Сијени, Италија, гдје је под менторством др Roberto Giorgi-ja и др Мила Томашевића интензивно радио на проблемима везаним за мултипроцесорске системе. У периоду 2006.-2009. г. радио је у звању хонорарног асистента на Факултету за информационе технологије и Факултету за пословне студије Универзитета Медитеран у Подгорици. Од 2006. г. до данас ради у звању виши специјалиста за комутациону мрежу у Црногорском Телекому А.Д., у Подгорици.

Милан Радуловић је члан међународних удружења HiPEAC и IEEE. Добитник је награде за најбољи научно-истраживачки рад младог истраживача у области Рачунарска техника и информатика на XLIX конференцији ETRAN-а одржаној у Будви јуна 2005. г. Добитник је награде за најбољег радника у Сектору Технике Црногорског Телекома, А. Д., за Q1 2014. г.

Коаутор је једног рада у међународним часописима са impact фактором са SCI листе, четири рада у домаћим часописима, пет радова на међународним скуповима и осам радова на домаћим скуповима.

Прилог 1.

Изјава о ауторству

Потписани-а **Милан Б. Радуловић**

Број уписа _____

Изјављујем

да је докторска дисертација под насловом

„Предлог подршке за спекулативно извршавање нити у CMP процесорима“

_____

- резултат сопственог истраживачког рада,
- да предложена дисертација у целини ни у деловима није била предложена за добијање било које дипломе према студијским програмима других високошколских установа,
- да су резултати коректно наведени и
- да нисам кршио/ла ауторска права и користио/ла интелектуалну својину других лица

Потпис докторанда

У Београду, 09.12.2014.г.

Милан Б. Радуловић

Прилог 2.

Изјава о истоветности штампане и електронске верзије докторског рада

Име и презиме аутора **Милан Б. Радуловић**

Број уписа

Студијски програм

Наслов рада **„Предлог подршке за спекулативно извршавање нити у CMP процесорима“**

Ментор **др. Мило В. Томашевић, ванредни професор**

Потписани **Милан Б. Радуловић**

изјављујем да је штампана верзија мог докторског рада истоветна електронској верзији коју сам предао/ла за објављивање на порталу **Дигиталног репозиторијума Универзитета у Београду**.

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

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

Потпис докторанда

У Београду, 09.12.2014.г.

Милан Б. Радуловић

Прилог 3.

Изјава о коришћењу

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

„Предлог подршке за спекулативно извршавање нити у CMP процесорима“

које је моје ауторско дело.

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

Моју докторску дисертацију похрањену у Дигитални репозиторијум Универзитета у Београду могу да користе сви који поштују одредбе садржане у одабраном типу лиценце Креативне заједнице (Creative Commons) за коју сам се одлучио/ла.

1. Ауторство
2. Ауторство – некомерцијално
3. Ауторство – некомерцијално – без прераде
4. Ауторство – некомерцијално – делити под истим условима
5. Ауторство – без прераде
6. Ауторство – делити под истим условима

(Молимо да заокружите само једну од шест понуђених лиценци, кратак опис лиценци је дат на полеђини листа).

Потпис докторанда

У Београду, 09.12.2014.г.

Милан Б. Радуловић

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 прераде. 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 стане 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 прераде. 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.