Универзитет у Крагујевцу
ФАКУЛТЕТ ТЕХНИЧКИХ НАУКА
ЧАЧАК

МАСТЕР РАД

СПЕЦИФИКАЦИЈА АРХИТЕКТУРЕ FTN_RISC

Ментор

проф. др Синиша Ранђић

Студент

Маријана Гудовић 841/2015

У Чачку,
2016. године

САДРЖАЈ

1. УВОД

Прва документована рачунарска архитектура била је у преписци између Čarlsa Bebidža и Ade Lovlejs, у којој је описана аналитичка машина. Један од пример је први нацрт извештаја Džon fon Nojmana о EDVAC из 1945. године, који је описао организацију логичких елемената. IBM га користи за развој рачунара IBM 701, првог комерцијалног рачунара са ускладиштеним програмима, испорученог почетком 1952. године.

Термин "архитектура" у рачунарској литератури може се пратити на раду Lajle R. Johnson, Mohamed Usman Khana и Frederika P. Brooks, из 1959. године, чланова одељења Машинске организације у главном истраживачком центру IBM. Johnson је имао прилику да напише истраживачки рад о Stretch суперрачунару који IBM развија у научној лабораторији Лос Аламос. Да би описао детаље рачунара, он је истакао да је опис формата, инструкције, типова хардвера, параметара, брзине на нивоу "архитектуре система" – термин који је по његовом мишљењу више него користан од термина "машинска организација". Након тога, Brooks, почиње друго поглавље књиге (планирање рачунарског система: Пројекат Stretch, 1962) пишући, "Архитектура рачунара, као и друге архитектуре, јесте уметност утврђивања потреба корисника, структуре, а затим пројектовање да, што је могуће у оквиру економских и технолошких ограничења задовољи те потребе". Brooks је помогао развој IBM Систем/360 фамилије рачунара, у којој је "архитектура" постала именица која дефинише "шта корисник треба да зна".

Под архитектуром рачунара подразумевамо општу конфиграцију његових основних компоненти, њихових битних карактеристика и узајамних веза. Најпростије речено она се бави проблемима употребе и пројектовања рачунара. Посматрано са становишта архитектуре рачунара, употреба рачунара се своди на његово програмирање, јер је намена рачунара да извршава програме. Начин програмирања зависи од особина скупа наредби рачунара. Овим особинама се бави архитектура наредби. Циљ пројектовања рачунара је имплементација тих наредби. Реализација наредби обухвата организацију и изведбу рачунара. Организација рачунара се бави организационим компонентама које образују рачунар, као и међусобним односима ових

компоненти, док се реализација рачунара бави поменутим проблемима производње компоненти.

У процесу пројектовања користимо две основне групе алата. Прву групу чине хардверски, а другу групу чине софтверски алати. Као пример хардверских алата могу се навести радне станице за пројектовање хардвера, емулатори, логички анализатори, инструменти, итд. Алати који се користе при пројектовању, а спадају у групу софтверских алата су базирани на скупу различитих софтверских пакета. Софтверски пакети се развијају и усавршавају годинама, и по правилу садрже групе комплексних програма који се користе у свим фазама описивања функционалности, симулације, синтезе и имплементације система.

У погледу спецификације архитектуре важни критеријуми су формат и величина инструкције, тип операција, начин адресирања, извршавање инструкција, скуп регистра, о чему ћемо се доста бавити при изради ове теме мастер рада.

У поглављу Појам архитектуре рачунара дата је дефиниција архитектуре рачунара, објашњена њена важност и улога. Указано је на аспекте имплементације и перформансе архитектуре, а на крају поглавља дат је осврт о врстама архитектуре (Фон Нојманова, Харвардска, CISC, RISC). Дат је опис сваке од наведених архитектура, њихове карактеристике, предности и мане.

У поглављу Примери рачунарских архитектура приказане су основне карактеристике рачунарских архитектура, на којима се базирају савремени рачунари. Приказом су обухваћене архитектуре као што су: x86, MIPS, ARM, PowerPC и SPARC. Дат је опис сваке од наведених архитектура, њихове карактеристике као што су начин рада, регистри који се користе и формати инструкција.

У поглављу Концепт архитектуре процесора FTN_RISC_Rev.1 дате су основне препоставке на бази којих је извршено специфицирање архитектуре процесора FTN_RISC_Rev.1. Такође, дати су различити захтеви са којима се приступило специфицирању архитектуре овог процесора и њихове реперкусије на могућа архитектурна решења. И на крају су анализирана могућа решења њихове импликације на архитектуру овог процесора, као што је смањење системског времена код промене контекста.

У последњем поглављу Приказ архитектуре процесора FTN_RISC_Rev.1 дат је опис ове архитектуре процесора, карактеристике процесора као што су дужина инструкције, процесорске речи, регистри који се користе, начини адресирања. Затим је дефинисан симболички језик, а након тога формати инструкције и на крају дата је табела са симболичким машинским инструкцијама.

2. ПОЈАМ АРХИТЕКТУРЕ РАЧУНАРА

Архитектура рачунара је унутрашња структура дигиталног рачунара коју сачињава пројекат и распоред скупа инструкација и регистара за складиштење података. Одређена рачунарска архитектура се бира имајући у виду тип програма (софтвера) који ће се на њему извршавати (пословни, научни, опште намене, итд). Главне компоненте или подсистеми једне рачунарске архитектуре су улазно/излазни уређаји, складиштење (примарна и секундарна меморија), комуникација, контрола и процесирање. За сваки се може рећи да поседује властиту, посебну архитектуру.

У рачунарској науци и инжењерству, архитектура рачунара је скуп дисциплина која описује рачунарски систем наводећи његове делове и њихове међусобне односе, али са аспекта видљивог програмеру. На пример, на високом нивоу, рачунарски инжењери могу бити забринути како централна процесорска јединица (CPU) делује и како користи меморију рачунара. Неке модерне рачунарске архитектуре укључују кластере рачунарства и неуједначен приступ меморији.

Рачунарски инжењери користе рачунаре за пројектовање нових технологија у рачунарству. Иако је пројекат веома лако променити, пројектанти компајлера често сарађују са архитектама, предлажући побољшања у скупу инструкција. Модерни емулатори могу мерити време у циклусима генератора такта: проценити потрошњу енергије у Џулима и дају реалне процене величине кода у бајтовима. Они утичу на погодност корисника, трајност батерије, као и величине и цену највећег физичког дела рачунара: меморији. То јест, они помажу да се процени вредност рачунара.

2.1 Улога архитектуре рачунара

Циљ је испројектовати рачунар који има максималан учинак, а имајући потрошњу енергије у виду, ниске трошкове у односу на ниво очекиваних перформанси, а такође да је веома поуздан. Да би се ово постигло многи аспекти рада рачунара треба да се размотре, укључујући скуп инструкција, функционалну организацију, логику пројектовања и имплементације. Сама имплементација подразумева између осталог пројектовање интегрисаног кола, његово паковање, питање напајања и хлађења.

Оптимизација пројекта захтева познавање компајлера или оперативних система који најбоље одговарају логици пројектованог процесора.

Скуп инструкција архитектуре (ISA, Instruction Set Architecture) је интерфејс између хардвера и софтвера рачунара и може се посматрати из угла програмера на машинском језику. Рачунари не разумеју језике високог нивоа који имају мало, ако их има, језичких елемената који се директно преводе у изворни машински код. Процесор разуме само инструкције кодиране на неки нумеричком начин, обично као бинарне бројеве. Софтверски алати, као што су компајлери, преводе језике високог нивоа, као што је C, у машинске инструкције.

Осим инструкција, ISA дефинише ставке у рачунару који су доступни програму – на пример типови података, скуп регистара, начини адресирања. ISA процесора се обично даје у облику приручника, који описује како су инструкције кодиране. Такође, она може дефинисати кратко (нејасно) методичка имена инструкција. Имена се могу препознати помоћу развијеног софтверског алата који се назива асемблер. Асемблер је рачунарски програм који преводи људски читљив облик инструкција на ISA рачунарски читљив облик. ISA варира у квалитету и комплетности, а добри ISA праве компримис између удобности програмера (више инструкција) и трошкова рачунара да тумаче инструкције (јефтиније је боље), брзине рачунара (брже је боље), и величине кода (мањи је бољи). Меморијска организација дефинише како инструкције интерагују са меморијом, а такође и како различити делови меморије комуницирају једни са другима.

Организација рачунара помаже у оптимизацији префрормански пројекта. На пример, софтверски инжењери треба да знају могућности процесора да обради захтеве. Они ће можда морати да оптимизују софтвер како би добили највише перформансе уз најмању цену. Ово може захтевати прилично детаљну анализу организације рачунара. На пример, мултимедијални декодер, пројектанти ће можда морати да организују да се већина података обрађују у најбржем путу. Што се тиче организације рачунара она такође помаже у избору процесора за одређени пројекат. Мултимедијалним пројектима потребан је веома брз приступ подацима, док ће надзорни софтвер можда морати да има брзе прекиде. Понекад су за одређене задатке потребне додатне компоненте. На пример, рачунар способан за виртуелизацију, захтева хардверску подршку у раду виртуелне меморије, тако да се меморија различитих симулираних рачунара може држати одвојена. Организација рачунара и карактеристике такође утичу на потрошњу енергије и трошкове процесора.

2.2 Имплементација архитектуре и перформансе

Када се заврши са описом скупа инструкција и микроархитектуре рачунара, практична машина може бити пројектована. Овај део процеса пројектовања се зове имплементација. Имплементација се обично не сматра инжењерском дефиницијом, већ пројектом хардвера. Имплементација се може даље разложити на неколико корака:

- Логичка имплементација - пројектују се блокови дефинисани у микроархитектури на нивоу регистар – трансфер (RTL, Register Transfer Level) и нивоу логичког кола.
- Имплементација кола ради на транзисторском нивоу – пројекат основних елемената (логичке капије, мултиплексери, ...), као и неких већих блокова који се могу спроводити на овом нивоу, или чак (делимично) на физичком нивоу, из разлога перформанси.
- Физичка имплементација – пројектовање физичких кола. Различите компоненте кола су смештени у чипу или на табли и повезују се жицама.
- Валидација пројекта – тестира се рачунар у целини да се види да ли ради у свим ситуацијама и свим тренуцима. Када почне имплементација, као први корак у валидацији пројекта врши се симулација користећи логику емулатора. Међутим, ово је обично сувише споро да покреће реалне програме. Дакле, после корекције, прототипови су конструисани користећи програмибилна поља – низове (FPGA, Field Programmable Gate Array). Многи пројекти заустављени су у овој фази. Последњи корак је да се тестира прототип интегрисаних кола. Интегрисана кола могу захтевати неколико репројектовања у случају да постоји проблем који треба решити.

За процесоре, цео процес имплементације се често назива пројекат процесора (Processor Design).

Модерне перформансе рачунара се често описују у милиона инструкција у секунди (MIPS, Milion Instruction Per Second). Овај показатељ представља меру ефикасности архитектуре при било којој брзини такта. Једноставни модерни процесори лако достижу близу 1. Суперскаларни процесори могу достићи три до пет, извршавајући неколико инструкција у једном циклусу.

Историјски гледано, многи људи мере брзину рада рачунара по такту (обично у MHz или GHz). Ово се односи на број циклуса у секунди главног такта процесора.

Међутим, ово је погрешно метрички, машина са вишим тактом не мора нужно имати боље перформансе. Као резултат тога произвођачи су удаљили од такта као мере учинка.

Остали фактори могу утицати на брзину су брзина магистрале, брзина меморије на располагању, као и врста и редослед инструкција које се покрећу у програмима. У типичном кућном рачунару, најједноставнији, најпоузданији начин да се убрза рад је обично додавање радне меморије познатије као RAM (Random Access Memory). Више RAM повећава вероватноћу да су потребни подаци или програм у радној меморији – па је мање вероватно да ће систему требати премештање података из секундарне меморије нпр., диска. Диск је често десет хиљада пута спорији него RAM, јер има механичке делове који морају да се покрећу како би се приступило подацима.

2.1 Врсте архитектуре рачунара

Данас се у примени рачунарских система користе различите архитектуре као што су Фон Нојманова, Харвардска, RISC и CISC архитектура.

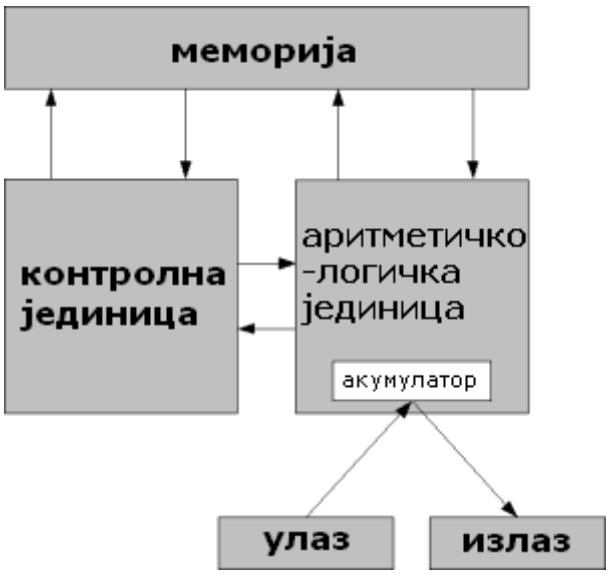
2.1.1 Фон Нојманова архитектура

Израз фон Нојманова архитектура се односи на пројекат рачунара који користи јединствену структуру за складиштење у којој чува и инструкције и податке. Израз фон Нојманова машина се може користити да опише такав рачунар. Неке од одлика ове архитектуре су и да се главној меморији приступа као једнодимензионалном низу.

Најраније рачунске машине су имале фиксне програме. Неки веома једноставни рачунари и данас користе овакав пројекат, било због једноставности, било у сврху обуке. На пример, стони калкулатор је (у принципу) рачунар са фиксним програмом. Он може да извршава просте математичке функције, али не може да се користи за обраду текста, или неку другу функцију. Како би се променио програм овакве машине, неопходно је преспајање жица, реструктурирање, или чак репројектовање машине. Најранији рачунари су у ствари више били “дизајнирани” него што су били „програмирани”. Ако је „ре-програмирање” и било могуће, оно је представљало физички посао са претходним планирањем и прављењем нацрта.

Концепт рачунара са запамћеним програмом је променио ову праксу. Стварањем скупа инструкција, и односом према израчунавањима као према серијама инструкција (рачунарски програм), рачунар постаје флексибилнији. Третирањем ових инструкција на исти начин као и осталих података, рачунар лако може да мења програм који извршава.

Изрази „фон Нојманова архитектура” и „рачунар са запамћеним програмом” се често користе синонимно. Међутим, Харвардска архитектура представља такође концепт пројекта, код кога се програм памти на начин који га чини лако измењивим, али се не користи иста меморија као и за остале податке.



Слика 2.1 Фон Нојманов модел

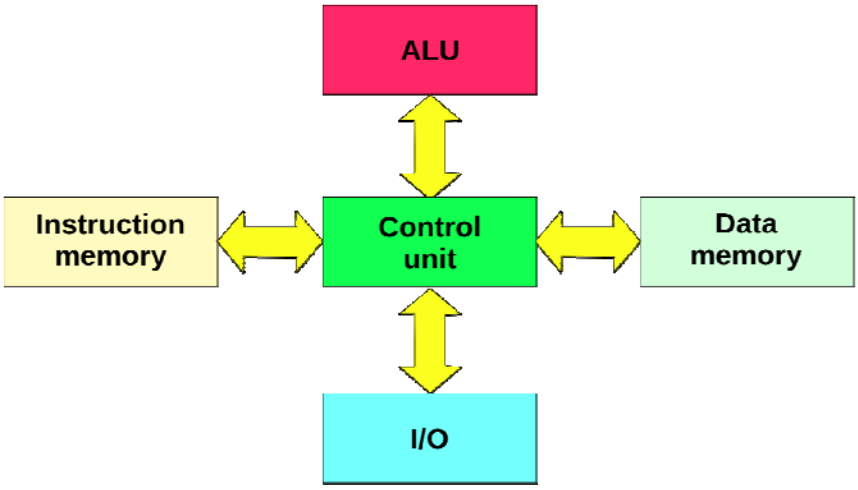
2.1.2 Харвардска архитектура

Харвардска архитектура (за разлику од Фон Нојманове) подразумева постојање два засебна меморијска простора. Меморију за смештање података, тј. меморију података и меморију за смештање програма, тј. програмску меморију. Обе меморије поседују засебне магистрале тј. разликујемо магистралу података и програмску магистралу.

Преко програмске магистрале инструкције програма доспевају до управљачке јединице где се декодирају. На основу декодиране инструкције (адресе операнда), преко магистрале података, допрема се захтевани операнд.

Раздвајање меморије на два засебна меморијска простора је учињено ради повећања брзине обраде података, јер се могу истовремено допремати инструкције и операнди. Ова архитектура се најчешће среће код дигиталних сигнал процесора (DSP, Digital Signal Processor) од којих се захтева брза обрада података и рад у реалном времену.

Поред Харвардске архитектуре развијена је и побољшана Харвард архитектура код које је меморија података подељена на А и Б део који такође поседују засебне магистрале како би се у једном такту могла допремити два операнда процесору.



Слика 2.2 Харвардска архитектура

2.1.3 CISC архитектура

Рачунар са сложеним скупом инструкција (CISC, Complex Instruction Set Computer) је традиционална рачунарска архитектура, код које централна процесорска јединица користи микрокод да би извршавала веома широк скуп инструкција. Те инструкције могу да буду промењљиве дужине и да користе све начине адресирања, што захтева сложену електронику за њихово декодирање. Раније CISC архитектуре имале су јединствену кеш меморију у којој се чувају инструкције и подаци. Ове архитектуре имају исти пут за податке и инструкције и користе мало регистарско поље (RF, Registar File). Новији CISC процесори користе посебне кеш моморије за податке и инструкције. За имплементирање великог скупа инструкција процесори користе микропрограмски управљану меморију. Ипак новија решења су базирана на директном управљању, тзв. Управљање на бази “ожичене логике”. Основна обележја CISC концепта су:

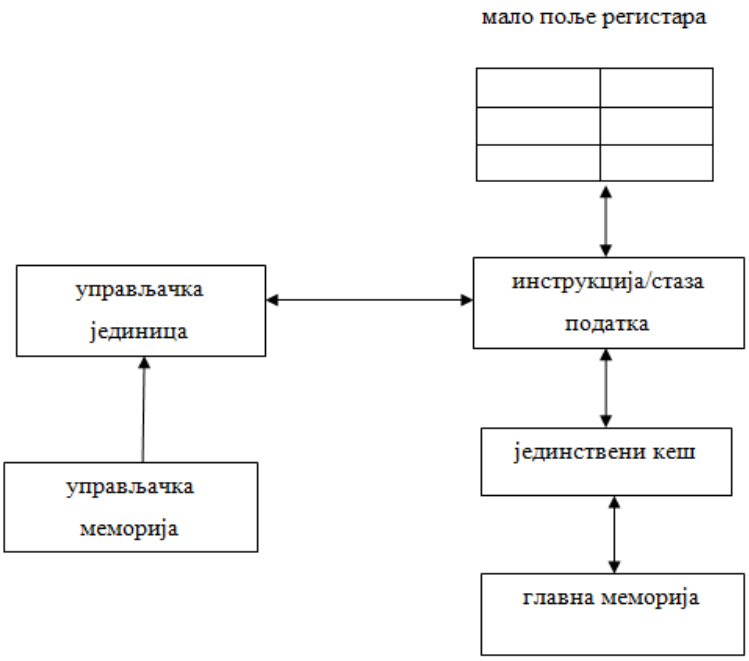
- користи се комплексни инструкцијски сет, тј. инструкцијска листа је врло бројна што има и предности и мане,
- инструкције су сложене што значи да њихово извршавање ређе траје 1 или 2 такта, а много чешће 3, 4 и више тактова, тј. извршавање инструкција није прецизно дефинисано, јер различите инструкције имају различито трајање,
- реализација CISC процесора је релативно једноставна, а развој кошта мање него конкурентске технологије.

Предности CISC технологије су:

- Постоји велики број инструкција, тако да се може пронаћи одговарајућа инструкција за сваки програмски задатак што олакшава програмирање,
- Реализација CISC-а тражи мање транзистора него еквивалентни RISC процесор што значи јефтинију производњу,
- Развој CISC процесора траје краће и кошта мање него еквивалентни RISC процесор, што се повољно одражава на цену,
- За CISC процесоре постоји велика софтверска база и широк круг корисника навикнутих на тај софтвер, што се сматра највећом предношћу CISC-а у односу на конкурентске технологије.

Недостаци CISC технологије су:

- Велики број инструкција значи дугу инструкцијску листу, што успорава претраживање, проналажење и декодирање инструкција, атиме успорава и рад процесора,
- Различите инстукције имају различита времена трајања, што значи да долази до застоја у извршавању програма. Ово је основна препрека за паралелно извршавање више инструкција,
- Архитектура CISC-а је достигла свој максимум и даљи напредак је врло слаб. Пораст перформанси се може постићи само сировим повећањем редне фреквенције процесора што најчешће није довољно.



Слика 2.3 CISC архитектура

2.1.4 RISC архитектура

Централне процесорске јединице рачунара са редукованим скупом инструкција (RISC, Reduced Instruction Set Computer) имају по правилу константну дужину инструкција, не користе индиректни начин адресирања и задржале су само оне инструкције које могу да се преклапају и извршавају у једном машинском циклусу или и мање од тога. Код RISC процесора, кеш меморије за податке и инструкције су раздвојене, а такође су различити и путеви преко којих се приступа овим меморијама. Треба нагласити да се код RISC процесора користи регистарско поље већег обима. Управљачка јединица RISC процесора је базирана на директном управљању. Оваквим приступом за случај да у програмском току не постоји велики број инструкција гранања или прекида, могуће је постићи CPI реда један циклус.

Основна обележја RISC технологије су:

- Користи се редуковани скуп инструкција које су пажљиво изабране тако да се помоћу њих може решити сваки програмски задатак,
- Све инстрикције су довољно једноставне да се могу извршити унутар једног такта, што значи да све инструкције имају једнако трајање,
- Све инструкције се реализују хардверски.

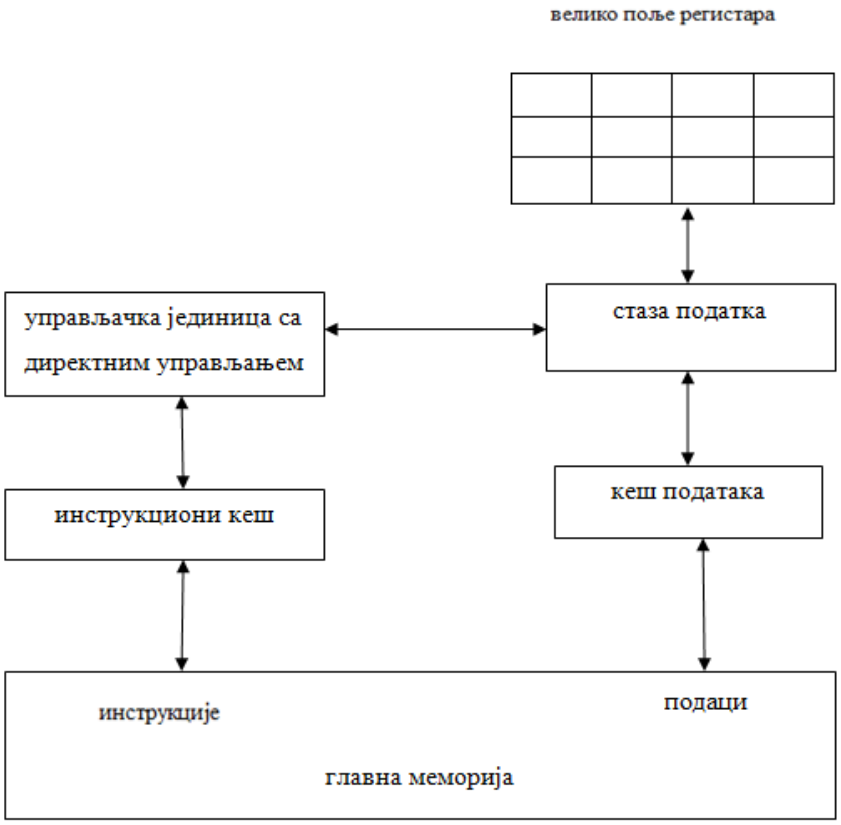
Предности RISC технологије су:

- Користи се мањи број инструкција, што значи да је претраживање, проналажење и декодирање инструкција брже, јер је инструкцијска листа краћа. То се повољно одражава на брзину процесора,
- Све инструкције имају једнако трајање (1 такт) па нема застоја у извршавању програма. Ово је веома битно код паралелног извршавања више инструкција,
- Све инструкције се извршавају хардверски што значи веома брзо, јер је хардвер увек знатно бржи од софтвера.

Недостаци RISC технологије су:

- Софтверска база је ограничена, јер је RISC од старта усмерен ка професионалним корисницима што подразумева да је софтвер прилагођен овим архитектурама веома скуп ,
- Развој процесора траје дуго и пуно кошта, што се неповољно одражава на цену процесора,
- RISC решења користе осетно више транзистора него еквивалентни CISC процесори због чега је производња скупља ,

– RISC процесори траже употребу скупих кеш меморија због потреба за процесирањем велике количине података.



Слика 2.4 RISC архитектура

3. ПРИМЕРИ АРХИТЕКТУРЕ РАЧУНАРА

Највећи број савремених рачунара је базиран на једној од следећих архитектура: MIPS, x86, ARM, PowerPC, SPARC. Поред рачунара са наведеним архитектурама постоји и читав низ рачунара који су базирани на аутохтоним архитектурним решењима.

3.1 MIPS архитектура рачунара

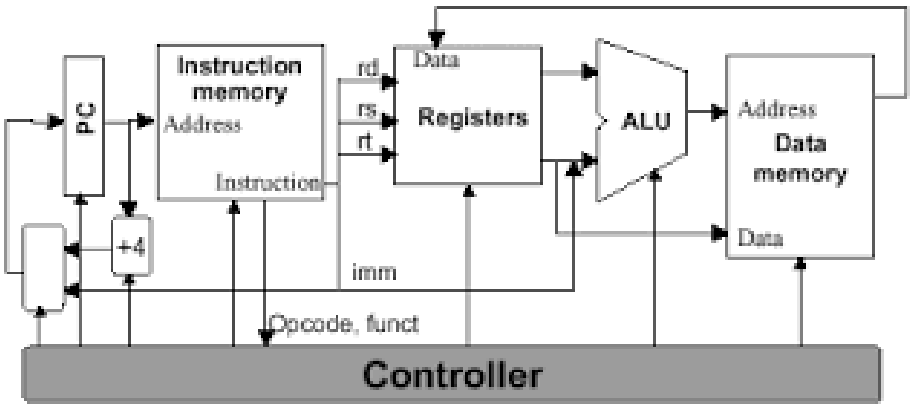
MIPS (Microprocessor without Interlocked Pipeline Stages) је RISC архитектура са фиксном дужином инструкција. Користи три различите врсте инструкција, које реализују аритметичке и логичке операције, упоређивања, условног и безусловног скока и преноса података између меморије и регистара.

MIPS процесори су представници RISC концепта са Load/Store архитектуром. То заправо значи да се све операције извршавају над операндима који се налазе у процесорским регистрима, а меморији се приступа искључиво преко Load (меморија-регистар) и Store (регистар-меморија) инструкција. На тај начин значајно се побољшавају перформансе процесора, јер је време приступа меморији знатно дуже од времена приступа регистрима.

Такође, спољну магистралу процесора чине две посебне магистрале, једна за приступ инструкцијама, а друга за приступ подацима. Тиме се приступ подацима обавља паралелно са прибављањем инструкција чиме се елиминишу конфликти између инструкција и података који реално постоје код система заснованих на јединственој магистрали.

MIPS има два начина рада: кернел режим и кориснички режим. У режиму језгра, када је статус бит постављен на 0, начин рада је пребачен на режим језгро и може да се приступи и промене сви регистри. Овај режим је привилегпван у односу на друге режиме и примењује се у случају грешке, прекида. У режиму корисника, када је статус бит 1, оперативни режим пребацује се на режим корисника. Овом режиму се приступа од стране корисника и има мање могућности него да је у режиму језгра. Такође, користи се за међусобно изоловање кориснике да не би ометали једни друге.

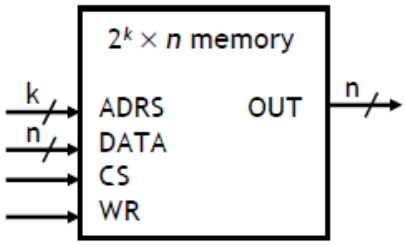
На слици 3.1 приказана је основна структура MIPS процесора.



Слика 3.1 Структура MIPS процесора

Меморија је бајт адресибилна са 32-битним адресама. Меморијска структура је слична скупу регистар, овде је 2^k x n RAM. Избором опције CS омогућава се или "онемогућава" приступ RAM меморији. Опција ADRS специфицира меморијске локације којима се приступа. Затим WR омогућава читање или писање у меморију (слика 3.2):

- кад се чита из меморије потребно је WR постави на 0. OUT ће бити n-битна вредност чувана у ADRS,
- када се пише у меморију потребно је WR поставити на 1. Податак је n-битна вредност сачувана у меморију.



CS	WR	Operation
0	x	None
1	0	Read selected address
1	1	Write selected address

Слика 3.2 Опис начина приступања MIPS меморији

MIPS архитектура дефинише следеће регистре:

- 32 регистра опште намене, сви обима 32 бита и дефинишу RF (Register File) поље,
- регистри за специјалне намене који се користе за чување међурезултата након извршавања операција множења и дељења,
- програмски бројач (PC) који чува адресу наредне инструкције, пошто су све инструкције дужине 32 бита онда се PC након извршавања сваке инструкције увећава за 4.

Постоје три основна формата инструкције:

– аритметичко-логичка инструкција (R тип) су инструкције које се састоје од 6 инструкцијских поља (слика 3.3). Поље opcode је заједничко за све врсте инструкција и дефинише операције које треба да преузме процесор при извршавању инструкција. Поље rs и rt се односе на првог и другог изворишног регистра инструкција, док поље rd се односи на одредишни регистар где се чува аритметичко-логички резултат. Поље shamt садржи број бинарних позиција за које треба померити изворишни операнд који је дефинисан у funct пољу као и у свим аритметичким и логичким операцијама референтне меморије инструкција као што су load и store.

opcode	rs	rt	rd	shamt	funct
6 bit	5 bit	5 bit	5 bit	5 bit	6 bit

Слика 3.3 Формат R типа инструкције

– референтне меморијске инструкције (I типа) се често односе на пренос података између меморије и регистара (слика 3.4). Међутим, обзиром на потребу за пољем константне вредности, аритметичке и логичке операције са тренутним вредностима као што су beq, bneq и addi, такође спадају у ову групу инструкција. За разлику од предходног типа инструкција овде rt поље има два значења. У случају у коме се инструкција односи на директне инструкције или load инструкције поље задаје одредиште регистру. А у случају store, поље одређује извориште регистра. Поље rs је комплементно rt пољу, где у случају store садржи одредишну адресу базе меморије, а у случају load или директне инструкције садржи изворишну адресу базе меморије.

opcode	rs	rt	constant or address
6 bit	5 bit	5 bit	16 bit

Слика 3.4 Формат I типа инструкције

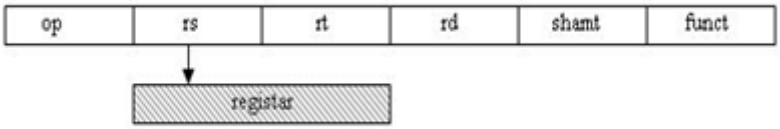
– инструкција скока (J тип) је најједноставнија, јер није потребна верификација услова (слика 3.5). Процесор при извршавању ових инструкција једноставно ажурира PC бројач са 26 адреса.

opcode	address
6 bit	26 bit

Слика 3.5 Формат J типа инструкције

MIPS архитектура има пет начина адресирања:

– Регистарско адресирање се углавном користи за израчунавање ефективне адресе инструкција регистра скока (J), операнд је у регистру (слика 3.6).

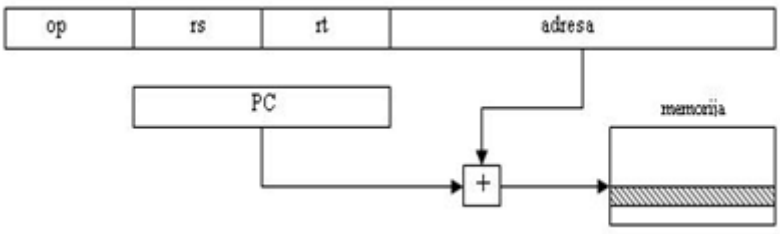


Слика 3.6 Приказ регистарско адресирања

– Непосредно адресирање подразумева да нема приступа меморији и на тај начин је релативно брже од других начина адресирања, значи операнд је константа и саставни је део инструкције (слика 3.7).

Слика 3.7 Приказ непосредног адресирања

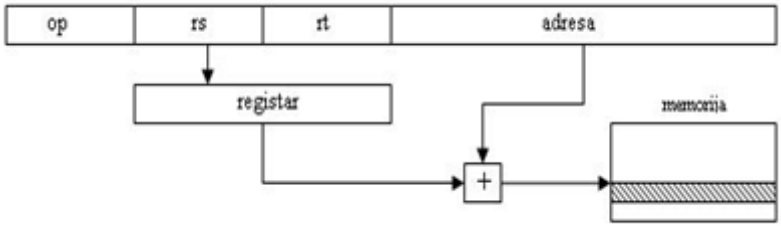
– PC релативно адресирање је адресирање код кога адреса представља збир PC и садржаји поља “адреса” инструкција (слика 3.8).



Слика 3.8 Приказ PC релативног адресирања

– Псеудо-директно адресирање се користи у инструкцији скока, где је разлика вредности 6-бита и инструкција је порасла на 26-бита. Горња четири бита PC и два најмања значајна бита који су 00 су спојени са 26-бита у 32-битну инструкцију.

– Базно адресирање се користи код load и store инструкција и познато је као индиректно адресирање које делује као показивач на неки меморијску локацију чија адреса се може наћи у регистру (слика 3.9).



Слика 3.9 Приказ базног адресирања

3.2 x86 архитектура рачунара

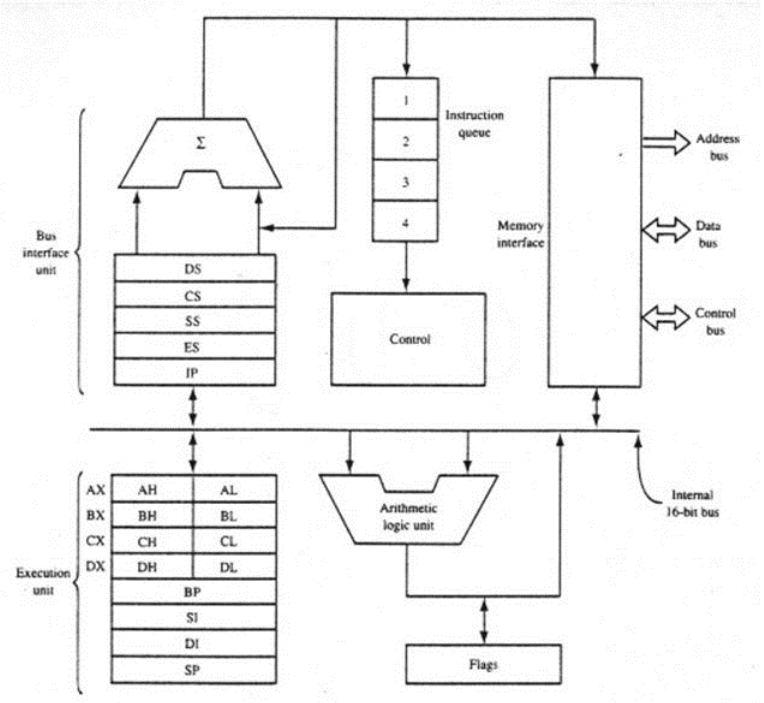
Термин x86 означава породицу компатибилног скупа инструкција архитектуре заснованих на INTEL 8086 CPU. 8086 је уведен 1978. године, као потпуно 16-битни наставак Intel 8-битне серије засноване на 8080 микропроцесору, са меморијском сегментацијом као решењем за адресирање више меморије него што би било могуће остварити обичним 16-битним адресама. Термин x86 је изведен из чињенице да су ранији наследници 8080, такође имали имена са завршетком “86”.

Многа додавања и проширења су била примењена на x86 скуп инструкција током година, већином са пуном компатибилношћу претходних процесора. Архитектура је имплементирана у процесоре као Intel-a, тако и Cyrix-a, AMD-a и многих других компанија.

x86 архитектура је променљиве дужине инструкције, пре свега “CISC” дизајна са акцентом на заосталу компатибилност. Скуп инструкција није типичан CISC, међутим, већ се ради о проширеним верзијама једноставне 8-битне 8008 и 8080 архитектуре. Адресирање бајта је омогућено и речи се чувају у меморији са little-endian (little-endian је начин записа податка у меморији тако да је на нижој адреси нижи бајт меморијске речи) редоследом битова. Приступ меморији за не-поравнате (unaligned) адресе је дозвољен за све важеће величине речи. Највећа природна величина за цели број у аритметици и меморијским адресама (или померај) је 16, 32 или 64 бита у зависности од генерације архитектуре (новији процесори укључују директну подршку и за мање целе бројеве). Вишеструке скаларне вредности могу се извршавати истовремено преко SIMD јединице где инструкције могу да раде паралелно на (једној или две) 128-битној речи, од којих свака садржи 2 и 4 броја у покретном зарезу (сваки 64 или 32 битне ширине респективно), или алтернативно, 2, 4, 8 или 16 целобројних бројева (сваки 64,

32, 16 или 8 битова дужине респективно). Широки SIMD регистри значе да постојећи x86 процесор може учитати или ускладиштити до 128 бита меморијских података у једној инструкцији и такође обављати операције над битовима на дужини пуних 128-бита, паралелно. Тако да непосредно адресирање помераја и непосредни подаци могу се изразити као 8-битна величина за честе појавне случајеве или у контекстима где је опсег -128 до 127 довољан. Типичне иструкције су стога 2 или 3 бајта дугачке (мада су неке много дуже, а неке су једнобајтне).

Већина регистара се представља у opcodes коришћењем три бита, и највише једног операнда тако да инструкција може бити меморијска локација. Међутим, овај операнд меморије такође може бити одредиште (или комбинација извора и одредишта), док други операнди, извор, могу бити садржаји регистара или immediate операнд (immediate операнд је операнд који је директно кодиран као део машинске инструкције). Међу осталим факторима, то доприноси величини кода који конкурише 8-битној машини и омогућава ефективно коришћење инструкција кеша. Релативно мали број општих регистара је направио важан метод регистар-релативног адресирања (користећи мале непосредне офсете) приступа операндима, посебно на стеку. Много напора је стога уложено у прављење таквог приступа као што је брз регистарски приступ, односно један циклус проточних инструкција, у већини случајева у којима је приступ подацима могућ на највишем нивоу у кешу. На слици 3.10 приказана је структура процесора са x86 архитектуром.



Слика 3.10 x86 архитектура

x86 архитектура има три основна начина адресирања и један специјалан случај, виртуелни 8086 режим:

- Заштићени режим је природно стање процесора у коме су све инструкције и функције доступне. Програмима су дате посебне меморијске области са именима сегмената, а процесор спречава програме из заштићене меморије да кваре један другог,
- Реални адресни режим реализује програмско окружење Intel 8086 процесор са неколико додатних карактеристика као што су могућност преласка у друге режиме. Овај режим је доступан у Windows 98, а може да се користи и за покретање MS-DOS програма који захтева директан приступ системској меморији и хардверским уређајима. Код програма који раде у овом режиму може се десити да се оперативни систем сруши (престане да се одазива на команде),
- Систем управљања режим обезбеђује оперативни систем са механизмом за спровођење функција као што су управљање напајањем и безбедност система. Ове функције се спроводе од произвођача рачунара који прилагођава процесор за подешавање система,
- Виртулени 8086 режим је у суштини радни режим који омогућава програме у реалном режиму и оперативним системима да раде док су под заштићеним режимом надзора оперативног система. Ово омогућава велику флексибилност у извршавању програма у заштићеном режиму и програма у реалном режиму, истовремено.

x86 процесори имају велики број регистара, неки од тих регистара су:

- 16-бита – Оригинални Intel 8086 и 8088 имају четрнаест 16-битних регистара. Четири од њих (AX, BX, CX, DX) су регистри опште намене, иако сваки од њих може имати додатну намену; на пример, само CX може да се користи као бројач за инструкције петље. Постоје два показивачка регистра: SP који показује на врх стека и BP (базни показивач) који се користи да покаже на неко друго место у стеку, обично горње локалне променљиве. Два регистра (SI и DI) су за индексирање низа. Четири регистара сегмента (CS, DS, SS и ES) се користе за формирање меморијске адресе. FLAGS регистар садржи заставице као carry flag (carry flag је један бит у статусном регистарском систему (застава) који се користи да означе када је аритметика ношења или позајмице генерисана од најзначајнијих ALU бит позиције), overflow flag (overflow flag је обично један бит у статусном регистарском

систему који се користи да означи када је дошло до аритметике преливања у операцији, указујући да резултат не може да стане у броју битова који се користе за операцију (ширина ALU)) и zero flag (zero flag је један бит застава која је централна карактеристика на већини конвенционалних процесора архитектуре). Коначно, показивач инструкција (IP) показује на следећу инструкцију која ће бити учитана из меморије и затим извршена. Овом регистру не може се директно приступити (читати или писати) помоћу програма,

– 32 бита – Са појавом 32-битног 80386 процесора, 16-битни регистри опште намене, базни регистар, индексни регистар, инструкцијски показивач, и FLAGS регистар, али не и сегментни регистри, су проширени на 32 бита. Ово је представљено префиксом "E" (за проширено) на име регистра у x86 асемблерском језику. Тако, AX регистар одговара најнижим битовима новог 32-битног EAX регистра, SI одговара најнижим 16 битовима ESI, и тако даље. Регистри опште намене, базни регистри и индекс регистри могу се користити као основни у режиму адресирања, и сви ти регистри осим стек показивача могу бити коришћени као индекси у режиму адресирања. Два нова сегментна регистра (FS и GS) су додата. Са већим бројем регистара, инструкција и операнада, формат машинског кода је проширен. Да се обезбеди компатибилност уназад, сегменти са извршним кодом се могу означити као они који садрже или 16-битне или 32-битне инструкције. Посебни префикси омогућавају укључивање 32-битних инструкција у 16-битном сегменту или обрнуто,
– 64 бита – Почевши са AMD Opteron процесором, x86 архитектура се проширила са 32-битних регистара на 64-битне регистре, на сличан начин како се то догодило са 16 на 32-битно проширење. Префикс R идентификује 64-битне регистре (RAX, RBX, RCX, RDX, RSI, RDI, RBP, RSP, RFLAGS, RIP) и осам додатних 64-битних регистара опште намене (R8-R15) је такође додато у стварању x86-64. Међутим, ова проширења су употребљива само у 64-битном режиму, који је један од два начина на располагању у дугом режиму. Режими адресирања нису се драматично променили од 32-битног режима, осим што је адреса проширена на 64 бита, виртуелне адресе се сада пријављују проширене на 64 бита, а други изборни детаљи су драматично смањени. Поред тога, додат је режим адресирања да омогући меморијске референце у односу на RIP (инструкциони показивач или програмски

показивач), да олакша имплементацију позиционо-независног кода, који се користи у дељеним библиотекама и неким оперативним системима,

– Мешовити/специјалне намене – x86 процесори (почевши од 80386) такође укључују разне специјалне/мешовите регистре као контролне регистре (CR0 до 4, CR8 за само 64-бита), дибаг (за дибаговање) регистре (DR0 до 3, плус 6 и 7), тест регистре (TR3 до 7; само 80486), дескриптор регистре (GDTR, LDTR, IDTR), регистре задатака (TR), и модел-специфичне регистре (MSR, који се појављује са Pentium).

На слици 3.11 је приказана структура x86 регистара.

Регистри опште намене (A, B, C и D)

64	56	48	40	32	24	16	8
R?X							
				E?X			
						?X	
						?H	?L

64-битни режим – само регистри опште намене (R8, R9, R10, R11, R12, R13, R14, R15)

64	56	48	40	32	24	16	8
?							
				?D			
						?W	
							?B

Сегментни регистри (C, D, S, E, F и G)

16	8
?S	

Показивачки регистри (S и B)

64	56	48	40	32	24	16	8
R?P							
				E?P			
						?P	
							?PL

Напомена: ?PL регистри су доступни само у 64-битном режиму

Слика 3.11 Структура x86 регистра

3.3 ARM архитектура рачунара

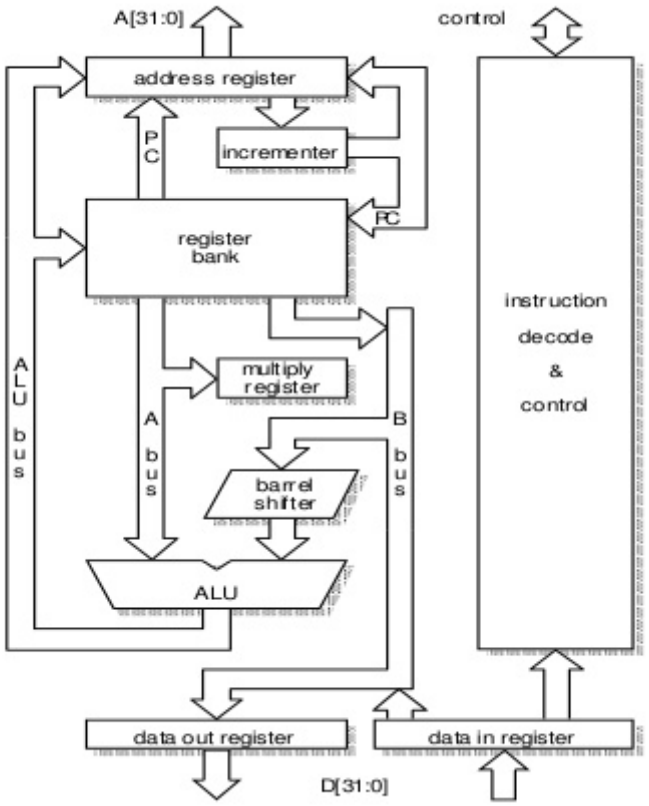
ARM је архитектура скупа инструкција за процесоре базиране на RISC архитектури развијена од стране британске компаније ARM Holdings. RISC-базирани приступ рачунарског дизајна значи да ARM процесори захтевају знатно мање транзистора од типичних процесора код просечних рачунара. Овај приступ смањује цену, загревање и потрошњу енергије. Ово су жељене особине код лаких, преносних, уређаја који раде на батерију укључујући паметне телефоне, лаптопове, таблете, и друге уграђене системе. Једноставнији дизајн омогућава ефикасније вишејезгарне процесоре и већи број језгара по мањој цени, што даје већу рачунарску моћ и вишу енергетску ефикасност за сервере и суперрачунаре.

У 2005. години, око 98% свих продатих мобилних телефона су користили бар један ARM процесор. Мала потрошња струје је учинила ARM процесоре јако популарним, тј. то је најраспротрањенија архитектура 32-битног скупа инструкција по количини производа.

ARM језгра се користе у пуно производа, посебно у PDA уређајима и телефонима. Неки примери рачунара су Microsoft Surface, Apple-ов iPad и ASUS Eee Pad Transformer. Остали укључују Apple-ов iPhone телефон и iPod преносни медија плејер, Canon PowerShot A470 дигитални фотоапарат, Nintendo DS ручна конзола за игре и Том Том систем за навигацију. Током 2005-е, ARM Holdings је имао удео у развоју рачунара за Manchester University, SpiNNaker, који користи ARM језгра да симулира људски мозак.

ARM чипови се такође користе у Raspberry Pi, BeagleBoard, BeagleBone, PandaBoard и другим рачунарима у оквиру једне плоче, јер су веома мали, јефтини и троше јако мало електричне енергије.

На слици 3.12 је приказана структура процесора са ARM архитектуром.



Слика 3.12 Структура процесора са ARM архитектуром

ARM архитектура дефинише неколико режима процесора, у зависности од имплементираних особина архитектуре. У једном тренутку, процесор се може налазити у само једном режиму, али може мењати режиме услед спољашњих догађаја (прекида) или програмски.

- Кориснички режим једини неповлашћен режим,
- FIQ режим – Привилегован режим у који се улази услед FIQ (Fast Interrupt Request) прекида,
- IRQ режим – Привилеговани режим у који се улази услед IRQ (Interrupt Request) прекида,
- Супервизорски (SVC) режим – Привилеговани режим у који се улази услед ресетовања процесора или извршења SVC инструкције,
- Недефинисани режим – Привилеговани режим у који се улази услед недефинисаног инструкцијског изузетка,

- Системски режим – Једини привилегован режим у који се не улази услед изузетка. Може му се приступити искључиво помоћу инструкције која директно уписује режимне битове CPSR-а,
- Зауставни режим – Привилеговани режим у који се улази услед prefetch abort или data abort изузетака.

Нешто што је веома битно и код ове архитектуре су регистри, где регистри од R0 до R7 су исти у свим режимима рада процесора; никад нису груписани. R13 и R14 су груписани у оквиру свих привилегованих режима осим системског режима. То јест, сваки режим у који се улази услед изузетка има сопствене R13 и R14. Ови регистри углавном садрже показивач стека, и повратну адресу из функцијских позива, редом.

Регистре као што су R13 се назива и SP, показивач стека, затим регистар R14 се назива и LR, Link регистар, и на крају регистар R15 се назива и PC, програмски бројач. CPSR има следећих 32 битова:

- M (битови 0–4) су битови режима процесора,
- T (бит 5) је бит стања Thumb-a,
- F (бит 6) је бит искључења FIQ,
- I (бит 7) је бит искључења IRQ,
- A (бит 8) је бит прекида услед непрецизних података,
- E (бит 9) одређује на којој се адреси налази најзначанији бајт,
- IT (битови 10–15 и 25–26) су битови ако-онда стања,
- GE (битови 16–19) одређују однос величина (веће или једнако),
- DNM (битови 20–23) су битови забране модификовања,
- J (бит 24) је бит стања Java-е,
- Q (бит 27) је бит прекорачења (који се мора ручно експлицитно ресетовати),
- V (бит 28) је бит прекорачења,
- C (бит 29) је бит преноса/позајмљивања/проширења,
- Z (бит 30) је нулти бит,
- N (бит 31) означава стање негативно/мање од.

На слици 3.13 приказана је структура скупа регистара у оквиру ARM архитектуре.

usr	sys	svc	abt	und	irq	fiq
R0						
R1						
R2						
R3						
R4						
R5						
R6						
R7						
R8						R8_fiq
R9						R9_fiq
R10						R10_fiq
R11						R11_fiq
R12						R12_fiq
R13		R13_svc	R13_abt	R13_und	R13_irq	R13_fiq
R14		R14_svc	R14_abt	R14_und	R14_irq	R14_fiq
R15						
CPSR						
		SPSR_svc	SPSR_abt	SPSR_und	SPSR_irq	SPSR_fiq

Слика 3.13 Регистри у оквиру режима рада процесора

3.4 SPARC архитектура рачунара

Scalable Processor Architecture (SPARC) је рачунар са смањеним скупом инструкција (RISC) сет инструкција архитектуре (ISA) је првобитно развијен од стране Sun Microsystems. SPARC архитектура, такође под именом трослојна архитектура, описује основне принципе система базе података. .Три слоја ове архитектуре су:

- Екстерни ниво, који пружа кориснику (или корисничком програму) индивидуални поглед на податке. Корисник види само оне делове података који су му потребни и за које има приступ,
- Концепционални, логички ниво описује који подаци се чувају у банци података и који су њихови међусобни односи,
- Интерни ниво представља физички посматран део банке података. На овом нивоу се описује како се конкретно подаци структурирају и меморишу. Сви

подаци се налазе на екстерним меморијским јединицама, нпр. магнетне плоче.

SPARC архитектура је модел који одређује недвосмислено понашање посматрано са стране софтвера за SPARC системе. Према томе, не нужно описују рад хардвера у стварној реализацији. SPARC архитектура има следеће главне функције:

- Линеарни, 32-битни адресни простор,
- Једноставни инструкцијски формат, где су све инстукције широке 32-бита и смештене у 32-битном обиму у меморији. Постоје само три основна инструкцијска формата и они имају opcode и поље за адресни регистар.Само load и store инструкције имају приступ меморији и I/O,
- Неколико начина адресирања, где је адреса меморије дата као “register +register” or “register+immediate”,
- Triadic адресни регистар – Већина инструкција ради на два регистарска операнда (или један регистар и константа) и резултат се поставља у трећи регистар,
- Велика “прозорска” регистарска датотека- у било ком тренутку програм види осам целих регистара плус 24-регистра из веће регистарске датотеке. Ту се може описати како кеш поставља аргументе, локалне вредности и повратну адресу,
- Регистарска датотека са одвојеним покретним зарезом- подесиви помоћу софтвера у 32 једноструке-тачности (32-битни), 16 дупле-тачности (64-битни), 8 четворостуке-тачности (128-битни) регистри или њихова комбинација,
- Кашњење контроле преноса-процесор преузима следећу инструкцију после кашњења контроле преноса инструкције. Она се извршава или не, што зависи од “поништеног” бита контроле преноса инструкције,
- Обележене инструкције – обележене додате/одузете инструкције претпостављају да су бар два значајна бита, таг бит,
- Микропроцесорска сихронизација инструкција – једна инструкција обавља read-then-set-memory операцију, а друга инструкција exchange-register-with-memory операцију.
- Копроцесор – архитектура дефинише јасан копроцесорски сет инструкција, поред floating-point сета инструкција.

На слици 3.14 је приказаан фотографија процесора са SPARC архитектуром.

Слика 3.14 SPARC архитектура

Инструкције су груписане у шест категорија:

- Load/store,
- Аритметика/логика/померање,
- Контрола преноса,
- Читање/писање регистар,
- Floating-point операнд,
- Копроцесор операнд.

Инструкције су кодиране у три главна 32-битна формата (слика 3.15).

Format 1 (*op* = 1): CALL

op	disp30
31	29 0

Format 2 (*op* = 0): SETHI & Branches (Bicc, FBfcc, CBccc)

op	rd		op2	imm22
op	a	cond	op2	disp22
31	29	28	24	21 0

Format 3 (*op* = 2 or 3): Remaining instructions

op	rd	op3	rs1	i=0	asi	rs2
op	rd	op3	rs1	i=1	simm13	
op	rd	op3	rs1	opf		rs2
31	29	24	18	13	12 4	0

Слика 3.15 Формат инструкција

Инструкцијска поља се тумаче на следећи начин:

- op и op2 – ова 2 и 3-битна поља кодирају 3 главне форме и 2 форме инструкција према следећим табелама:

Format	*op*	Instructions
1	1	CALL
2	0	Bicc, FBfcc, CBccc, SETHI
3	3	memory instructions
3	2	arithmetic, logical, shift, and remaining

op2	Instructions
0	UNIMP
1	unimplemented
2	Bicc
3	unimplemented
4	SETHI
5	unimplemented
6	FBfcc
7	CBccc

- rd – Ово 5-битно поље је адреса одредишта (извор) r или f или копроцесорски регистар или load/arithmetic (или store) инструкције. За двоструке инструкције read/writes најмање значајан један (или два) бита су неискоришћени и треба да се дефинишу као нула од стране софтвера,
- a – a поље у инструкцији поништава следећу инструкцију ако је грана условна или безусловна,
- cond – Ово 4-битно поље бира код услова за тестирање инструкције гране.
- imm22 – Ово 22-битно поље је константа која SETHI места смешта у дестинацију регистра.
- disp22 и disp30 – Ова 30-битна и 22-битна поља су исписане речи, знак проширења, PC релативно померање за позиве или гране, респективно,
- op3 – Ово 6-битно поље (заједно са 1-битом од op) кодира у формату 3 инструкције,
- i – бира други ALU операнд (integer) arithmetic и load/store инструкције. Ако је i = 0, онда је операнд r[rs2], а ако је i = 1, онда је операнд simm13, знак проширен са 13 на 32 бита,
- asi – Ово 8-битно поље је идентификатор простора адресе од стране load/store наизменичних инструкција.
- rs1 – Ово 5-битно поље је адреса првог r или f или копроцесорског регистра изворног операнда,
- rs2 – Ово 5-битно поље је адреса другог r или f или копроцесорског регистра изворног операнда када је i=0,
- simm13 – Ово 13-битно поље је знак проширен непосредном вредношћу који се користи као други операнд ALU за цео број, arithmetic или load/store инструкције када је i=0,

– opf – Ово 9-битно поље floating-point операнд (FPop) инструкције или копроцесорски операнд (CPop) инструкције.

Ова архитектура обухвата две врсте регистара, опште намене или регистри података и контрола/статус регистри. IU регистри опште намене називају се и r регистри, а FPU регистри називају се и f регистри.

IU контрола/статус регистри обухватају:

– PSR (Processor State Register) – 32-битни PSR садржи различите области које контролишу статус процесора и садрже информације. Може да буде модификован од стране SAVE, RESTORE, Ticc, и RETT инструкције, и свих инструкција које модификују услове кода. RDPSR и WRPSR инструкције читају и пишу PSR директно,
– WIM (Window Invalid Mask) – WIM је под контролом софтвера, а користи хардвер да би одредио да ли долази до прекорачења или не и да самим тим генерише неке од инструкција SAVE, RESTORE или RETT,
– TBR (Trap Base Register) – садржи три поља која су заједно једнака адреси која контролише пренос када дође до петље,
– Y (Multiply/Divide Register) – садржи најзначајнију реч дупле тачности производ целобројног множења, где се су коришћене SMUL, SMULcc, UMUL, UMULcc инструкције. Поред множења имамо и целобројно дељење и инструкције SDIV. SDIVcc, UDIV, UDIVcc. Овај регистар може да се чита и пише са RDY и WRY инструкцијама,
– PC,nPC (Program Counters) – PC садржи тренутну адресу инструкције добијену од стране IU, а nPC садржи адресу следеће инструкције за извршење (под претпоставком да до петље не дође),
– ASRs (implementation-dependent Ancillary State Registers) – бројеви од 1-15 су резервисани за будућу употребу архитектуре, бројеви од 16-31 су доступни за реализацију у сврхе попут тајмера, бројача, дијагностичких регистара...Овај регистар се чита и пише помоћу инструкција RDASR и WRASR,
– implementation – dependent IU Deferred-Trap Queue-може да садржи нулу или много одложених петљи низова.

FPU контрола/статус регистри обухватају:

– FSR (Floating-Point State Register) – садржи FPU режим и информације о статусу. Чита се и пише помоћу инструкција STFSR и LDFSR,

– FQ (implementation-dependent Floating-Point Deferred-Trap Queue) – ако су присутни у реализацији садрже довољно информација о стању што је довољно да омогући отпремање у наставцима и floating-point петљу.

CP (CP, Coprocessor) контрола/статус регистри обухватају:

– CSR (implementation-dependent Coprocessor State Register)
– CQ (implementation-dependent Coprocessor Deferred-Trap Queue).

3.5 PowerPC архитектура рачунара

PowerPC је RISC архитектура сета инструкција створена 1991. године од стране Apple–IBM–Motorola alliance, познате као AIM. PowerPC, еволуира као сет инструкција, од 2006. године под називом Power ISA, док стари назив природно живи, као заштитни знак наслеђа за неке имплементације Power архитектуре процесора, а у софтверском пакету на идентификаторе. Првобитно је намењен за персоналне рачунаре, али су PowerPC процесори постали популарни и као embedded процесори и као процесори високих перформанси.

Ова архитектура је пројектована уз принципе RISC архитектуре и омогућава суперслакарну имплементацију. Верзије пројекта постоје у ове 32-битне и 64-битне имплементације. Полазећи од основних Power спецификација, PowerPC је додао:

– Подршка за операције у оба big-endian и little-endian режима. Може се пребацити из једног режима у други у реалном времену, ова функција није подржана за PowerPC 970,
– Једно-прецизне форме неких инструкција са покретним зарезом, поред двоструко-прецизних форми,
– Додатне floating point инструкције указују на наредбе Apple,
– Комплетне 64-битне спецификације су компатибилне са 32-битним режимом,
– Безбедност вишеструког додавања,
– Управљањем меморијом архитектуре се интезивно користи у системима сервера и рачунара,
– Додавање нове архитектуре под називом Book-E замењујући архитектуру за управљање меморијом за уграђене апликације. Book-E је апликативни софтвер компатибилан са постојећом PowerPC имплементацијом, али су потребне мање промене у оперативном систему.

Оперативни системи који раде на PowerPC архитектури се обично деле на оне који су оријентисани ка PowerPC системима опште намене, и они оријентисани према уграђеним PowerPC системима.

На слици 3.16 приказана је фотографија процесора са PowerPC архитектуром.

Слика 3.16 PowerPC архитектура

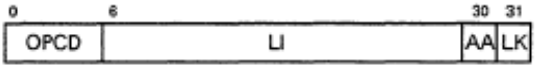
PowerPC архитектура дефинише register-to-register операције за све рачунарске инструкције. Изворним подацима за ове инструкције се приступа из регистара на-чипу или је обезбеђен као непосредна вредност уграђена у opcode. Подаци се преносе између меморије и регистара са експлицитним load и store инструкцијама. Многи од регистара спроведених у класичним и Book E верзијама архитектуре су идентични у називу и функцији. Регистри који се користе у овој архитектури су:

- Регистри фајлова – р егистри опште намене (GPRS, General-purpose registers) и floating-point (FPRs) регистри су они којима се може приступити преко изворишне или одредишне инструкције,
- Инструкцијски-доступни регистри – Регистри као што су регистар стања (CR), floating-point стање и регистар контроле (FPSCR), и неки SPRs којима се може приступати као by-products код извршења одређених инструкција,
- Регистри специјалних намена (SPRs) – Ови регистри су на чипу регистара који су део језгра процесора. Они контролишу тајмере, прекиде, јединице за управљање меморијом и друге процесорске ресурсе. Укључују имплементацију која је зависна од регистара хардвера (HIDs), а није дефинисана архитектуром, која се користи за конфигурацију и контролу.

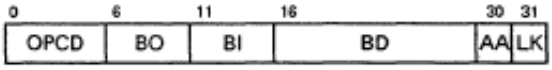
Све инструкције су дуге четири бајта и word-aligned. Тако, кад год је инструкција адресе представљена на процесору (као у Branch инструкцијама) два бита нижег реда се игноришу. Слично томе кад год процесор развија инструкцију адресе његова два ниска бита су нула. Битови од 0:5 су увек наведени као opcode. Многе инструкције имају

продужени opcode. Ова архитектура има велики број инструкцијских формата који ће бити представљени на следећим сликама:

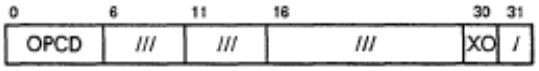
На сликама 3.17 до 3.23 приказани су инструкцијски формати PowerPC архитектуре.



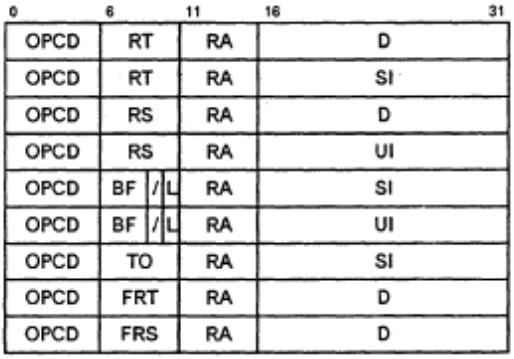
Слика 3.17 I инструкцијски формат



Слика 3.18 B инструкцијски формат

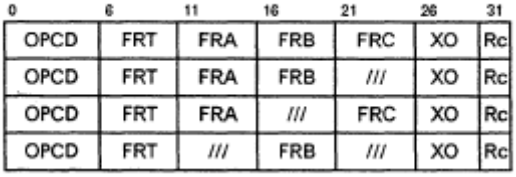


Слика 3.19 SC инструкцијски формат

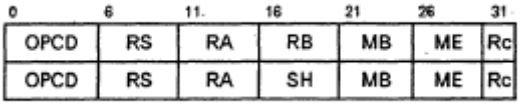


Слика 3.20 D инструкцијски формат

Слика 3.21 X инструкцијски формат



Слика 3.22 A инструкцијски формат



Слика 3.23 M инструкцијски формат

Значење неких од инструкцијских поља:

– OPCD (0:5) – Примарно opcode поље,

- XO (21 :29, 21 :30, 22:30, 26:30, 27:29, 27:30, 30, или 30:31) – Проширено opcode поље. Локације 21 :29, 27:29, 27:30 и 30:31 се односи само на 64-битну имплементацију,
- RC(31) – Када је 0 није постављен условни регистар. Када је 1 постављен је условни регистар који одржава резултат операције,
- FRT(6:10) – Поље које се користи за спецификацију FPR као мете операције,
- FRA (11:15) – Поље које се користи за спецификацију FPR као извор операције,
- RA(11:15) – Поље које се користи за спецификацију GPR као извор или мета,
- BO (6:10) – Поље које се користи за спецификацију опција за Branch Conditional инструкције.

4. КОНЦЕПТ АРХИТЕКТУРЕ ПРОЦЕСОРА FTN_RISC_REV.1

Ова архитектура је настала из потребе да у оквиру једног рада систематизује приступ пројектовању процесора и укаже на све проблеме који се при томе јављају. Такође, циљ је био да се укаже како се мењају пројектни захтеви, који проистичу из полазних претпоставки у складу са реалним могућностима њиховог задовољења. Резултат, тј. дефинисана архитектура, добијена у овиру овог пројектног поступка само је једно од могућих решења у приближању пројектних захтева реалним могућностима.

Пре самог приступања дефинисању ове архитектуре дате су неке полазне претпоставке:

- Архитектура треба да буде RISC базирана. То значи да ће основни формат инструкција бити регистарски, тј. аритметичко – логичке инструкције ће се извршавати над операндима које налазе у процесорским регистрима. Овом полазном претпоставком се јасно дефинише потреба за постојањем скупа регистара. Да би се смањила потреба за комуникацијом меморија – скуп регистара и обратно пожељно је да број регистара у скупу буде што је могуће већи. Обим скупа регистара је у конкретном случају одређен дужином инструкције;
- Дужина инструкције треба да буде фиксна и износи 32 бита, односно четири бајта. Овом претпоставком унета су ограничења у погледу:
  - Броја и структуре регистара у скупу регистара;
  - Дужине адресе која се може специфицирати у оквиру инструкције за потребе формирања меморијске адресе у свим случајевима меморијског адресирања;
  - Дужине константе која се користи у случају коришћења непосредног операнда;
- Дужина адресне речи је 32 бита, а адресибилна јединица је бајт. То значи да максимална величина оперативне меморије 32Gb;
- Дужина процесорске речи је 32 бита;

Првобитна идеја при спецификацији архитектуре овог процесора била је да се покуша дефинисати процесор, који би пружао добру подршку промени контекста (context switching) у условим мултипроцесног рада. Идеја је базирана на претпоставци да је могуће испројектовати процесор код кога не би при свакој смени процеса долазило до промене контекста. Тачније говорећи структура и број регистара у скупу регистара би омогућавао да сам процесор чува у себи одређени број статусних речи процеса (Program Status Word – PSW). То значи да би у скупу регистара постојао већи број подскупа регистара, као што су PC, RB, RX, SP, STATUS у којима би се чувао контекст додељеног му процеса. Такође, скуп општих регистара би био подељен у одговарајући број подскупова који би се придруживали одређеном процесу. На слици 4.1 приказана се структура скупа регистара који има подршку мултипшроцесном раду.

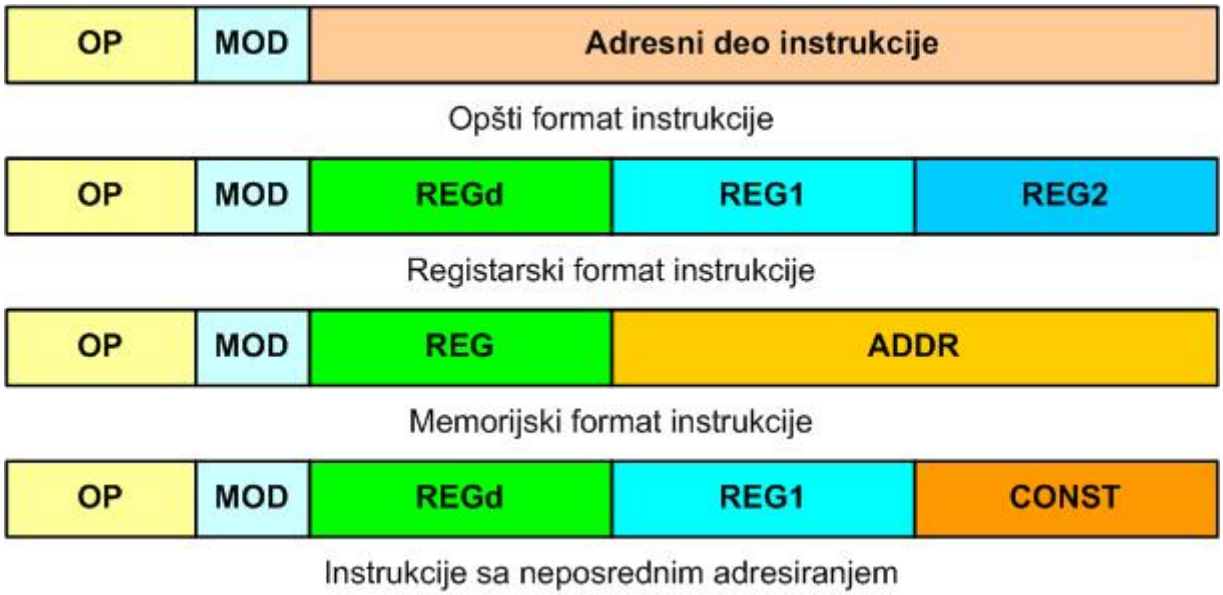
Слика 4.1 Структура скупа регистара за подршку мултипроцесном раду

Међутим, изабрана дужина инструкције од 32 бита показала се као ограничавајући фактор када је требало обезбедити додатни ниво адресирања по основу идентификације подскупа коме регистри који се користе, припадају. Такође, да би жељени концепт подршке мултипроцесном раду могао да се користи у пуном обиму неопходно је да постоји директна спрега са модулом оперативног система, који је задужен за креирање процеса. У овом случају овај модул би имао задатак да врши

транслацију програма, тј. коришћених регистара у слободни подскуп регистара. Пошто је формат инструкције подељен на четири поља:

- Поље кода операције;
- Три поља адреса (која се у случају адресирања меморије могу спајати).

Анализом кодовања жељених начина адресирања уочено је да претпостављена дужина поља за операцију није довољна да се задовоље све потребе. Због тога је одлучено да се одустане од идеје за подршком мултипроцесном раду и да се изврши спецификација RISC архитектуре са напред изнетим претпоставкама у погледу дужине и формата инструкције. На слици 4.2 су приказани начини адресирања на којима ће се заснивати архитектура FTN_RISC процесора.



Слика 4.2 Формати инструкција FTN_RISC

Као што се са слике 4.2 види извршавање операција нада садржајима регистара је основи инструкцијски формат, што је и једна од главних одлика RISC архитектуре. С обзиром да је дужина поља за специфицирање регистара дужине 8 бита следи да скуп регистара има 256 регистара. Распоред регистара – специјалних и опште намене дат је у наредном поглављу (глава 5).

Без посебних разматрања усвојено је да је адресни простор заједнички за меморијске и локације у улазно – излазном простору. Такође, је усвојено да се за потребе стека користи посебна меморија. Наведене одлуке пружају могућност да се код неке нове ревизије ове архитектуре усвоје другачија решења чиме би се створиле могућности за њихову компаративну анализу.

5. ПРИКАЗ АРХИТЕКТУРЕ ПРОЦЕСОРА FTN_RISC_REV.1

У овом документу биће дат приказ спецификације архитектуре процесора FTN_RISC_Rev.1. Процесор има следеће иницијалне карактеристике:

1. Дужина инструкције је фиксна и износи 32 бита;
2. Дужина адресне речи је 32 бита;

Адресибилна јединица у меморији процесора је бајт, што значи да је капацитет меморије 232 бајта = 4Gb. С обзиром на дужину инструкције и процесорске речи од 32 бита (4 бајта) због униформног приступа меморији меморија је организована у меморијске речи дужине 32 бита (4 бајта). Осим тога меморија је подељена у сегменте дужине 64kB.

3. Дужина процесорске речи је 32 бита;
4. Процесор има регистаре опште и специјалне намене:
    a. 256 регистра опште/специјалне намене – адресирани пољем REG у инструкцији
    b. Адресни регистар (RA) – Регистар број 0
    c. Базни регистар (RB) – Регистар број 1
    d. Индексни регистар (RX) – Регистар број 2
    e. Показивач стека (SP) – Регистар број 3
    f. Регистар базе улаза – излаза (RIO) – Регистар број 4
    g. Општи регистри (RG) – Регистри од 5 до 255

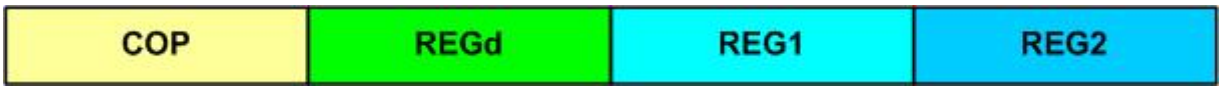
Инструкција има следећа поља:

1. Поље кода операције (COP) дужине 8 бита, од којих се 5 битова (поље OP) користи за идентификацију операције, а 3 бита (поље MOD) за специфицирање начина адресирања и друге намене (слика 5.1).

Слика 5.1 Општи формат инструкције

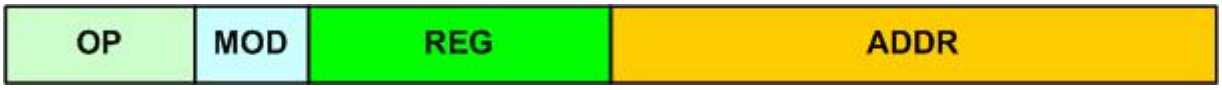
2. Три поља за специфицирање изворишта/одредишта операнада над којима се извршавају операције. Дужина ових поља је 8 бита. У случају

регистарског адресирања поље REGd је број регистра одредишта резултата, а поља REG1 и REG2 су бројеви регистара изворишта првог односно другог операнда (слика 5.2).



Слика 5.2 Регистарски формат инструкције

3. У случају приступања меморији поље REG се користи, као извориште или одредиште податка који се смешта/узима са меморијске локације за садржај поље ADDR, дужине 16 бита представља релативну адресу у односу на садржај базног регистра чија је адреса специфицирана пољем MOD (слика 5.3).



Слика 5.3 Регистарско/адресни формат инструкције

Операције које извршава процесор деле се на:

1. Аритметичко – логичке
    a. Сабирање
    b. Одузимање
    c. Аритметичко поређење
    d. Логичко I
    e. Логичко ILI
    f. Ексклузивно ILI
    g. Логичко поређење
    h. Логичка негација
2. Операције померања
    a. Логичко померање у лево
    b. Логичко померање у десно
    c. Аритметичко померање у лево
    d. Аритметичко померање у десно
    e. Ротирање у лево
    f. Ротирање у десно
3. Операције преноса података
    a. Пуњење регистара

b. Памћење регистара
c. I/O улаз
d. I/O излаз
e. Упис на стек
f. Читање са стека

4. Контролне операције
    a. Безусловни скок
    b. Условни скокови

При извршавању операција генерише се статусна информација, која ближе описује добијени резултат (слика 5.4). Статусна информација има следеће индикаторе:

а. Резултат операције једнак/различит од нуле – **ZERO** (**Z**) бит
    i. **Z=0** – Резултат различит од нуле
    ii. **Z=1 –** Резултат једнак нули
б. Резултат операције позитиван/негативан – **Negative** (**N**) бит
    i. **N=0** – Резултат позитиван
    ii. **N=1** – Резултат негативан
ц. Прекорачен опсег рачунања – **Overflow (O)** бит
    i. **O=0** – Није прекорачен опсег рачунања
    ii. **O=1** – Прекорачен опсег рачунања
д. Бит преноса – **Carry (C)** бит
    i. **C=0** – Није генерисан бит преноса
    ii. **C=1** – генерисан бит преноса
е. Бит парности – **Parity (P)** бит
    i. **P=0** – Бит парности за паран/непаран број јединица код Even Parity/Odd Parity
    ii. **P=1** - Бит парности за непаран/паран број јединица код Even Parity/Odd Parity

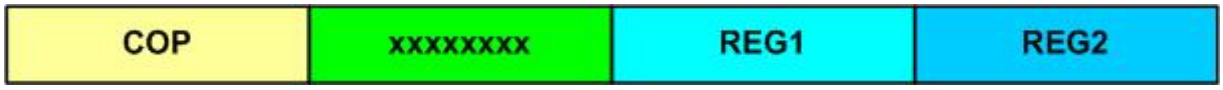
Слика 5.4 Статусна реч резултата

Процесор омогућава следеће начине адресирања:

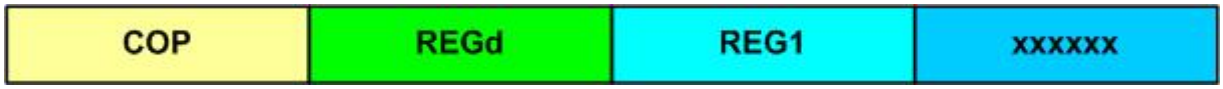
а. **Регистарско адресирање** – адреса у инструкцији одговара броју регистра са чијом се вредношћу оперише. Све аритметичко – логичке операције и операције померања извршавају се над садржајима

регистара. У случају аритметичко – логичких операција изузетке представљају операције поређења и логичке негације. Код операција поређења није потребно специфицирати број одредишног регистра, јер се не генерише резултат (резултат операције индентификује се преко статусне информације). Код логичке негације, која је унарна операција, не специфицира се број регистра другог операнда, јер не постоји.

Формат инструкције који се користи у овом случају приказан је на сликама 5.2, 5.5 и 5.6.

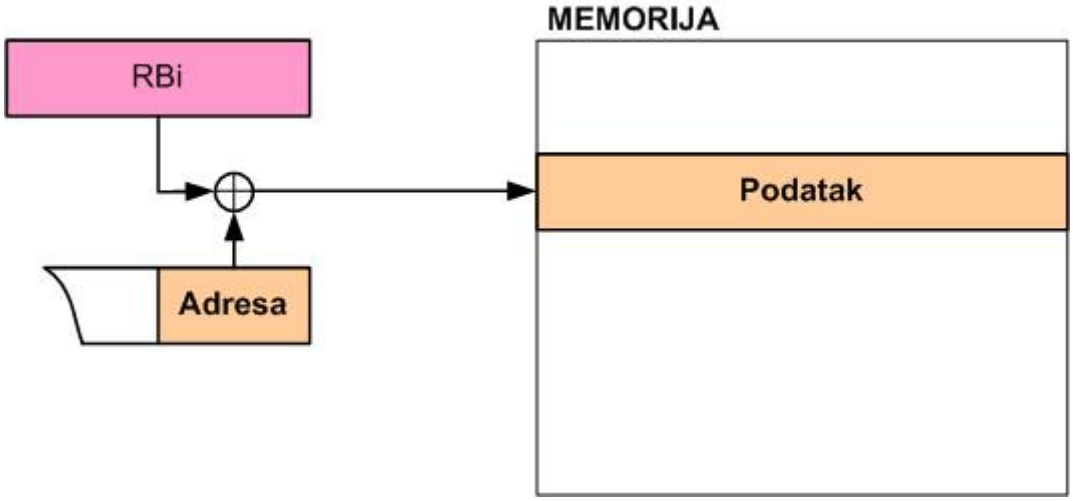


Слика 5.5 Формат инструкције код операције поређења



Слика 5.6 Формат инструкције у случају логичке негације

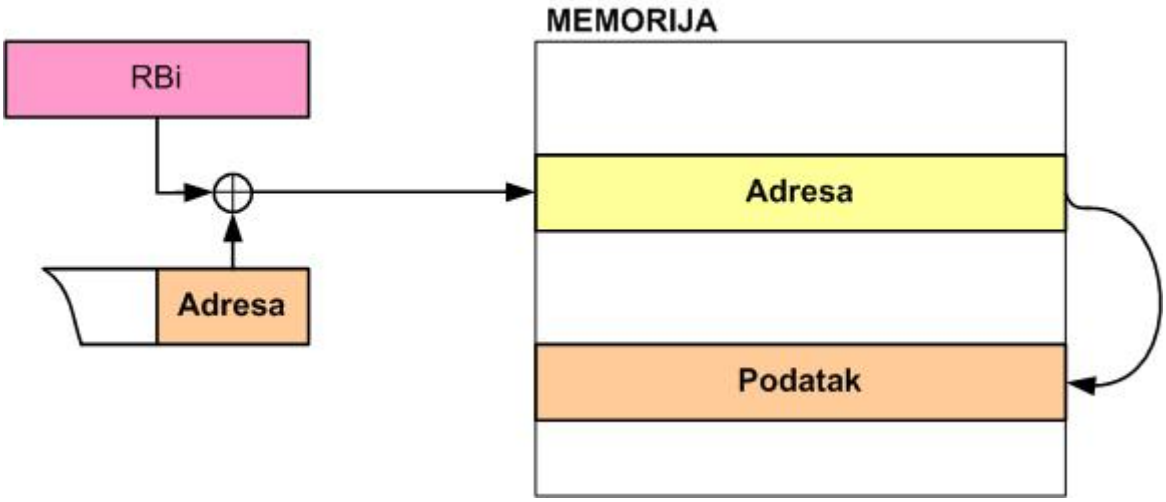
б. **Директно/релативно меморијско адресирање** – адреса која се специфицира у одговарајућим пољу инструкције увек је релативна у односу на вредност одређеног регистра базе. Ефективна/извршна адреса указује директно на локацију где се налази жељени операнд. Начин коришћења директно/релативног адресирања приказан је на слици 5.7.



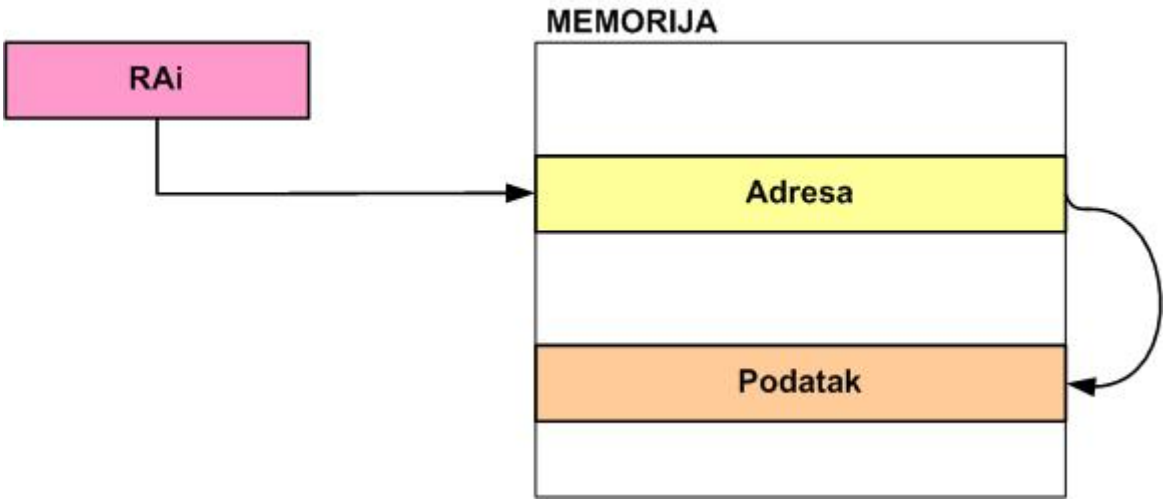
Слика 5.7 Начин коришћења директно/релативног адресирања

ц. **Индиректно/релативно меморијско адресирање** – адреса која се специфицира у одговарајућим пољу инструкције увек је релативна у

односу на вредност одређеног регистра базе. Ефективна/извршна адреса указује на локацију где се налази стварна адреса, адреса жељеног операнда, као што је приказано на слици 5.8. Ово је случај меморијског индиректног адресирања. На слици 5.9 је приказан начин адресирања у случају регистарског индиректног адресирања.



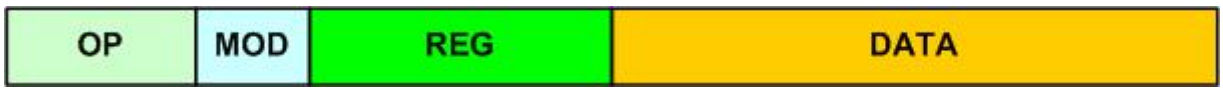
Слика 5.8 Начин коришћења меморијског индиректно/релативног адресирања



Слика 5.9 Регистарско индиректно адресирање

У оба случаја формат инструкције одговара приказу на слици 5.3.

д. **Непосредно адресирање** – садржај адресног дела инструкције представља непосредни операнд, чије коришћење зависи од операције која је специфицирана инструкцијом. Формати инструкције у којој се користи непосредно адресирање приказан је на слици 5.10.



Слика 5.10 Формат инструкције у случају непосредног адресирања

5.1 Симболички језик

1. **Сабирање**

   ADD R3,R1,R2 – регистарско сабирање
   ADI R3,R1,C – сабирање са константом C (8 бита)

2. **Одузимање**

   SUB R3,R1,R2 – регистарско одузимање
   SUI R3,R1,C – одузимање константе C

3. **Аритметичко поређење**

   CPA R1,R2 – аритметичко поређење садржаја два регистра
   CPI R,C – поређење садржаја регистра са константом C

4. **Логичко I**

   AND R3,R1,R2 – логичка I операција над садржајима регистара
   ANI R3,R1,C – логичка I операција над садржајем регистра и константе C

5. **Логичко ILI**

   OR R3,R1,R2 – логичка ILI операција над садржајима регистара
   ORI R3,R1,C – логичка ILI операција на д садржајем регистра и константе C

6. **Ексклузивно ILI**

   XOR R3,R1,R2 – логичка XOR операција над садржајима регистара
   XOI R3,R1,C – логичка XOR операција на д садржајем регистра и константе C

7. **Логичко поређење**

   CPL R1,R2 – логичко поређење садржаја регистара
   CLI R1,C – логичко поређење садржаја регистра и константе C

8. **Логичка негација**

   NOT R3,R1 – логичка негација садржаја регистра
   NOI R,C – логичка негација константе C

9. **Операција логичког померања у лево**

   SLL R,C – логичко померање садржаја регистра R у лево за број битова дефинисан константом C

10. **Операција логичког померања у десно**

    SLR R,C – логичко померање садржаја регистра R у десно за број битова дефинисан константом C

11. **Операција аритметичког померања у лево**

    SAL R,C – аритметичко померање садржаја регистра R у лево за број битова дефинисан константом C

12. **Операција аритметичког померања у десно**

    SAR R,C – аритметичко померање садржаја регистра R у десно за број битова дефинисан константом C

13. Операција ротирања у лево

ROL R,C – логичко ротирање садржаја регистра R у лево за број битова дефинисан константом C

14. Операција ротирања у десно

ROR R,C – логичко ротирање садржаја регистра R у десно за број битова дефинисан константом C

15. Пуњење регистара

LDL R,A – пуњење доњег бајта регистра R непосредним садржајем A

LDH R,A –пуњење горњег бајта регистра R непосредним садржајем A

LDB R,M – пуњење регистра R садржајем меморијске локације чија је адреса одређена сумом садржаја RB и M

LDR R3,[R1] – пуњење регистра R3 садржајем меморијске локације чија је адреса одређена индиректно адресом која је садржана у R1

LDM R, [M] –пуњење регистра R садражајем меморијске локације чија је адреса одређена индиректно сумом садржаја RB и M

LDBX R,M –пуњење регистра R садржајем меморијске локације чија је адреса одређена сумом садржаја RB, RX и M

LDRX R3,[R1] – пуњење регистра R3 садржајем меморијске локације чија је адреса одређена индиректно сумом регистара R1, RX

LDMX R,[M] – пуњење регистра Rсадржајем меморијске локације чија је адреса одређена индиректно сумом садржаја регистара RB, RX и M

16. Памћење регистара

STR R,M –памћење садржаја регистра R у меморијску локацију чија је адреса одређена сумом регистра A и M

STR R3, [R1] –памћење садржаја регистра R3 у меморијску локацију чија је адреса одређена индиректно садржајем регистра R1

STM R, [M] –памћење садржаја регистра R у меморијску локацију чија је адреса индиректно одређена сумом RB и M

STRX R3, [R1] – памћење садржаја регистра R3 у меморијску локацију чија је адреса индиректно одређена сумом садржаја регистара R1 и RX

SDMX R, [M] – памћење садржаја регистра R у меморијску локацију чија је адреса индиректно одређена сумом регистара RB, RX и M

17. I/O улаз

IND R,[M] – читање података са улазно – излазног уређаја чија је адреса одређена сумом RIO и M

18. I/O излаз

OUTD R,[M] – упис података са улазно – излазног уређаја чија је адреса одређена сумом RIO и M

19. Упис на стек

PUSH R – упис садржаја регистра R на стек

20. Читање са стека

POP R – читање стека у регистар R

21. Безусловни скок

JUMPD M –безусловни скок на адресу која је одређена сумом RB и M

JUMPP M –безусловни скок на адресу која је одређена сумом PC и M

22. Скок ако је EQU

JED M – условни скок по услову једнако на адресу која је одређена сумом RB и M

JEP M – условни скок по услову једнако на адресу која је одређена сумом PC и M

23. Скок ако је GE

JGED M – условни скок по услову веће – једнако на адресу која је одређена сумом RB и M

JGEP M – условни скок по услову веће – једнако на адресу која је одређена сумом PC и M

24. Скок ако је LE

JLED M – условни скок по услову мање – једнако на адресу која је одређена сумом RB и M

JLEP M – условни скок по услову мање – једнако на адресу која је одређена сумом PC и M

25. Скок ако је GT

JGTD M – условни скок по услову веће на адресу која је одређена сумом RB и M

JGTP M – условни скок по услову веће на адресу која је одређена сумом PC и M

26. Скок ако је LT

JLTD M – условни скок по услову мање на адресу која је одређена сумом RB и M

JLTP M – условни скок по услову мање на адресу која је одређена сумом PC и M

27. Скок ако је C

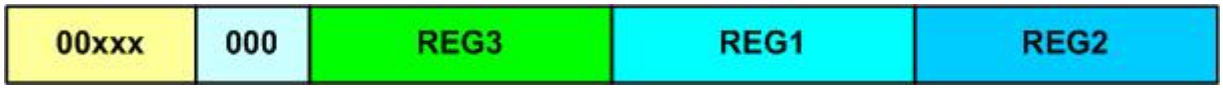
JCD M – условни скок по услову постојања на адресу која је одређена сумом RB и M

JCP M – условни скок по услову постојања на адресу која је одређена сумом PC и M

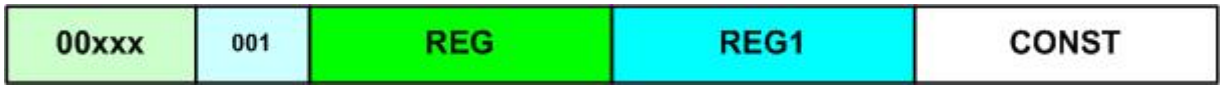
5.2 Формати инструкција

Аритметичко-логичке инструкције

Регистарски формат – Операнди над којима се извршава операција су садржаји регистара R1 и R2, а резултат се памти у регистру R3. Бројеви регистара који учуствују у извршењу операције дати су у одговарајућим 8 – битним пољима инструкције. У пољу COP виших пет битова намењено је дефинисању саме операције, а доња три бита служе за специфицирање начина адресирања. У конкретном случају постоји 8 аритметичко – логичких операција за које је резевисано првих 8 кодова поља операције у COP (00000 – 00111). Битом COP_0 се дефинише начин адресирања, у конкретном случају $COP_0 = 0$ указује да се ради о регистарском адресирању. У случају логичке негације поље REG2 се не користи.

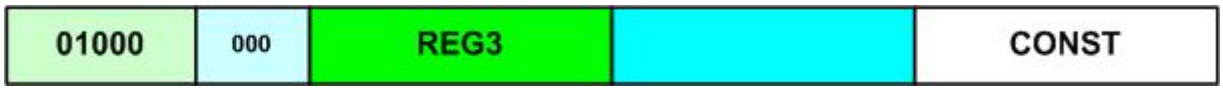


Регистарско – непосредни формат – У случају овог начина извршавања аритметичко – логичких операција уместо садржаја регистра као други операнд користи се 8 – битна константа, која је смештена у поље регистра REG3. Кодови операције су исти као у случају регистарског адресирања, а рад са константом указан је вредношћу $COP_0 = 1$. У случају логичке негације поље REG1 се не користи.



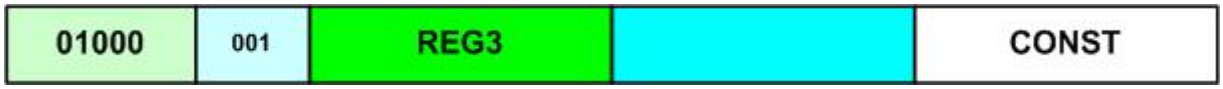
Операције померања

Логичко померање у лево – Операција померања има заједнички код 01000 дефинисан преко виших пет битова поља COP. За дефинисање смера померања користи се бит COP_0, који у случају да има вредност $COP_0 = 0$ дефинише смер у лево. Бит COP_1 служи за дефинсање врсте померања, $COP_1 = 0$ означава логичко померање. Поље REG2 садржи константу која дефинише број битова померања. Поље REG1 се не користи.

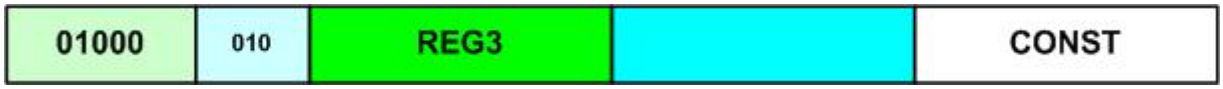


Логичко померање у десно – Операција померања има заједнички код 01000 дефинисан преко виших пет битова поља COP. За дефинисање смера померања

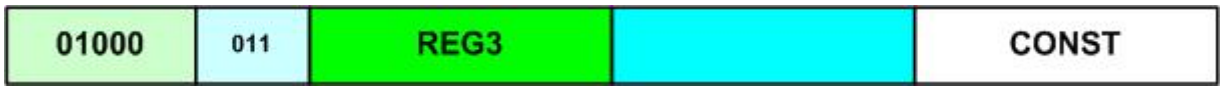
користи се бит COP_0, који у случају да има вредност $COP_0 = 1$ дефинише смер у десно. Бит COP_1 служи за дефинисање врсте померања, $COP_1 = 0$ означава логичко померање. Поље REG2 садржи константу која дефинише број битова померања. Поље REG1 се не користи.



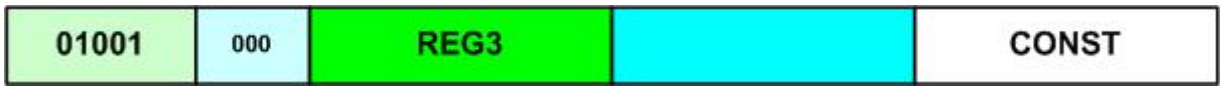
Аритметичко померање у лево – Операција померања има заједнички код 01000 дефинисан преко виших пет битова поља COP. За дефинисање смера померања користи се бит COP_0, који у случају да има вредност $COP_0 = 0$ дефинише смер у лево. Бит COP_1 служи за дефинсање врсте померања, $COP_1 = 1$ означава аритметичко померање. Поље REG2 садржи константу која дефинише број битова померања. Поље REG1 се не користи.



Аритметичко померање у десно – Операција померања има заједнички код 01000 дефинисан преко виших пет битова поља COP. За дефинисање смера померања користи се бит COP_0, који у случају да има вредност $COP_0 = 1$ дефинише смер у десно. Бит COP_1 служи за дефинисање врсте померања, $COP_1 = 1$ означава аритметичко померање. Поље REG2 садржи константу која дефинише број битова померања. Поље REG1 се не користи.

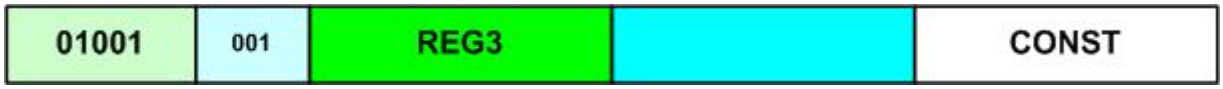


Ротирање у лево – Операција ротирања има заједнички код 01001 дефинисан преко виших пет битова поља COP. За дефинисање смера ротирања користи се бит COP_0, који у случају да има вредност $COP_0 = 0$ дефинише смер у лево. Поље REG2 садржи константу која дефинише број битова померања. Поље REG1 се не користи. Операција ротирања се извршава као логичка операција.



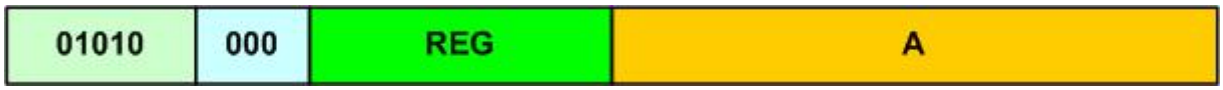
Ротирање у десно – Операција ротирања има заједнички код 01001 дефинисан преко виших пет битова поља COP. За дефинисање смера померања користи се бит COP_0, који у случају да има вредност $COP_0 = 1$ дефинише смер у десно.

Поље REG2 садржи константу која дефинише број битова померања. Поље REG1 се не користи. Операција ротирања се извршава као логичка операција.

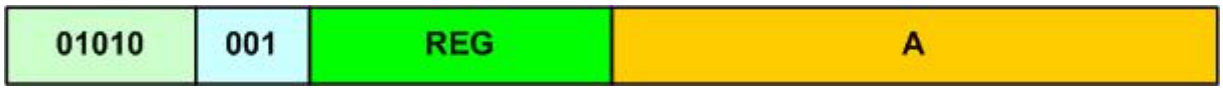


Операције преноса података

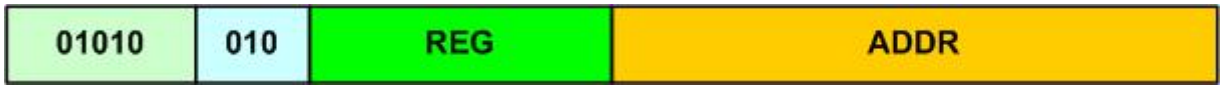
Пуњење доњег бајта регистара непосредним садржајем поља A – Операција пуњења регистра има заједнички код 01010 дефинисан преко виших пет битова поља COP. Садржај поља MOD = 000 означава да се непосредни податак из поља A инструкције уписује у нижи бајт регистра дефинисаног садржајем поља REG.



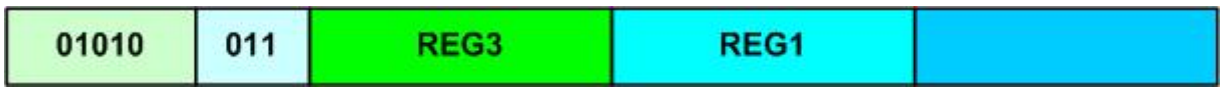
Пуњење горњег бајта регистара непосредним садржајем поља A – Операција пуњења регистра има заједнички код 01010 дефинисан преко виших пет битова поља COP. Садржај поља MOD = 001 означава да се непосредни податак из поља A инструкције уписује у виши бајт регистра дефинисаног садржајем поља REG.



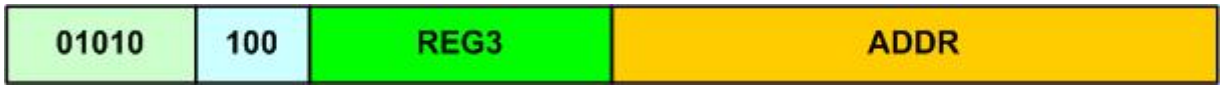
Пуњење регистара садржајем меморијске локације (директно) – Операција пуњења регистра има заједнички код 01010 дефинисан преко виших пет битова поља COP. Садржај поља MOD = 010 означава да се податак из меморијске локације чија је адреса одређена пољем ADDR инструкције уписује у регистар дефинисан садржајем поља REG.



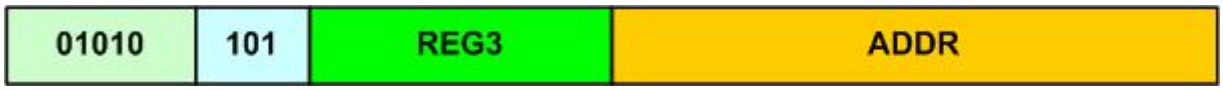
Пуњење регистара садржајем меморијске локације (индиректно) – Операција пуњења регистра има заједнички код 01010 дефинисан преко виших пет битова поља COP. Садржај поља MOD = 011 означава да се податак из меморијске локације чија је адреса индиректно одређена пољем REG1 инструкције уписује у регистар дефинисан садржајем поља REG3. Поље REG2 се не користи.



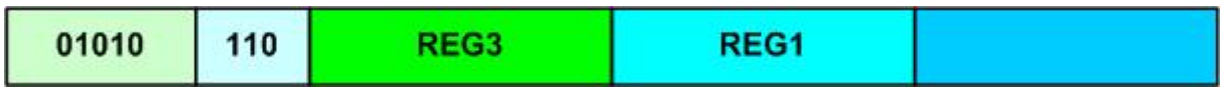
Пуњење регистара садржајем меморијске локације (индиректно) – Операција пуњења регистра има заједнички код 01010 дефинисан преко виших пет битова поља COP. Садржај поља MOD= 100 означава да се податак из меморијске локације чија је адреса индиректно одређена садржајем регистра RB и пољем ADDR инструкције уписује у регистар дефинисан садржајем поља REG3.



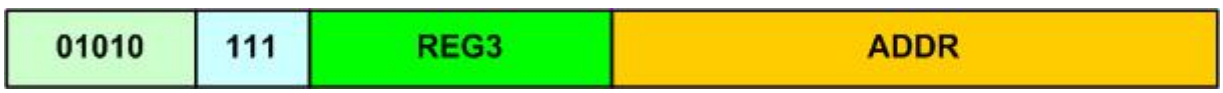
Пуњење регистара садржајем меморијске локације (директно) – Операција пуњења регистра има заједнички код 01010 дефинисан преко виших пет битова поља COP. Садржај поља MOD = 101 означава да се податак из меморијске локације чија је адреса одређена сумом садржаја регистара RB и RX и поља ADDR инструкције уписује у регистар дефинисан садржајем поља REG3.



Пуњење регистара садржајем меморијске локације (индиректно) – Операција пуњења регистра има заједнички код 01010 дефинисан преко виших пет битова поља COP. Садржај поља MOD = 110 означава да се податак из меморијске локације чија је адреса индиректно одређена сумом садржаја регистара RX и поља REG1 инструкције уписује у регистар дефинисан садржајем поља REG3. Поље REG2 се не користи.

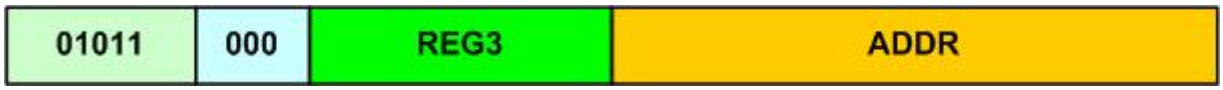


Пуњење регистара садржајем меморијске локације (индиректно) – Операција пуњења регистра има заједнички код 01010 дефинисан преко виших пет битова поља COP. Садржај поља MOD= 111 означава да се податак из меморијске локације чија је адреса индиректно одређена сумом садржаја регистара RB и RX и поља ADDR инструкције уписује у регистар дефинисан садржајем поља REG3.

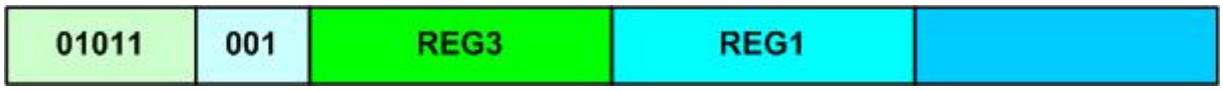


Памћење регистара

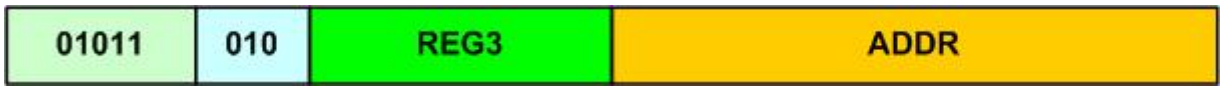
Памћење регистра (директно) – Операција памћења регистра има заједнички код 01011 дефинисан преко виших пет битова поља COP. Садржај поља MOD = 000 означава да се садржај регистра дефинисаног садржајем поља REG3 памти директно на меморијској локацији чија је адреса одређена сумом садржаја регистра RA и садржаја поља ADDR из инструкције.



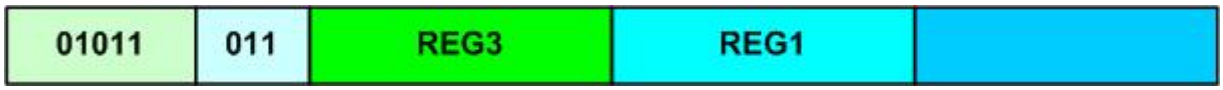
Памћење регистра (индиректно) – Операција памћења регистра има заједнички код 01011 дефинисан преко виших пет битова поља COP. Садржај поља MOD = 001 означава да се садржај регистра дефинисаног садржајем поља REG3 памти индиректно на меморијској локацији чија је адреса одређена садржајем регистра REG1. Поље REG2 се не користи.



Памћење регистра (индиректно) – Операција памћења регистра има заједнички код 01011 дефинисан преко виших пет битова поља COP. Садржај поља MOD = 010 означава да се садржај регистра дефинисаног садржајем поља REG3 памти индиректно на меморијској локацији чија је адреса одређена сумом садржаја регистра RB и садржаја поља ADDR из инструкције.



Памћење регистра (индиректно) – Операција памћења регистра има заједнички код 01011 дефинисан преко виших пет битова поља COP. Садржај поља MOD = 011 означава да се садржај регистра дефинисаног садржајем поља REG3 памти индиректно на меморијској локацији чија је адреса одређена сумом садржаја регистра RX и садржаја регистра REG1. Поље REG2 се не користи.

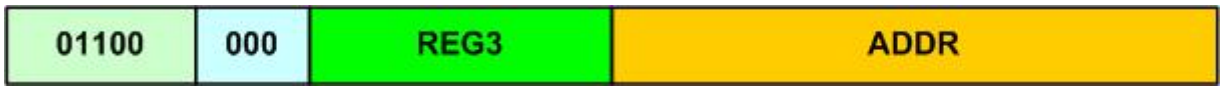


Памћење регистра (индиректно) – Операција памћења регистра има заједнички код 01011 дефинисан преко виших пет битова поља COP. Садржај поља MOD = 100 означава да се садржај регистра дефинисаног садржајем поља

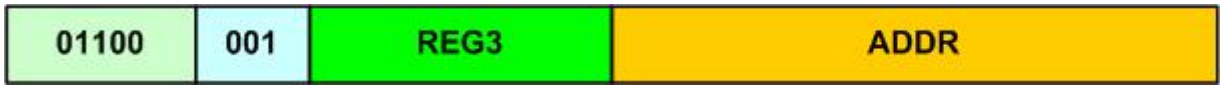
REG3 памти индиректно на меморијској локацији чија је адреса одређена сумом садржаја регистра RX и регистра RB и поља ADDR инструкције.

Улазно – излазне операције

Улаз података – Операција улаза/излаза података има заједнички код 01100 дефинисан преко виших пет битова поља COP. Садржај поља MOD = 000 означава да се у регистар дефинисан пољем REG3 пуни садржај излазног порта чија је адреса одређена сумом садржаја регистра RIO и садржаја поља ADDR инструкције.

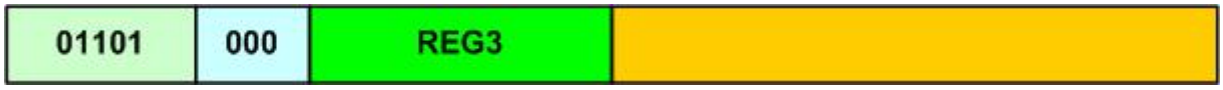


Излаз података – Операција улаза/излаза података има заједнички код 01100 дефинисан преко виших пет битова поља COP. Садржај поља MOD = 001 означава да се садржај регистра дефинисаног пољем REG3 памти на излазном порту чија је адреса одређена сумом садржаја регистра RIO и садржаја поља ADDR инструкције.

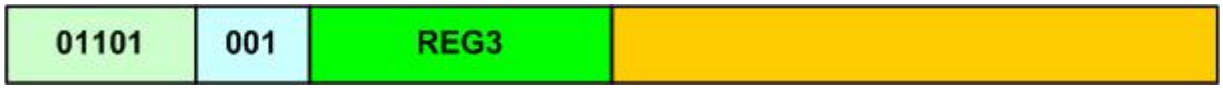


Операције над стеком

Стављање на стек – Операције над стеком имају заједнички код 01101 дефинисан преко виших пет битова поља COP. Садржај поља MOD = 000 означава да се регистар дефинисан пољем REG3 памти на стеку.



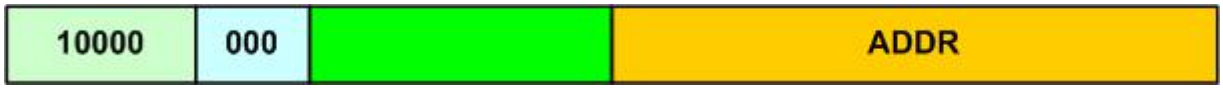
Узимање са стека – Операције над стеком имају заједнички код 01101 дефинисан преко виших пет битова поља COP. Садржај поља MOD = 001 означава да се у регистар дефинисан пољем REG3 чита садржај врха стека.



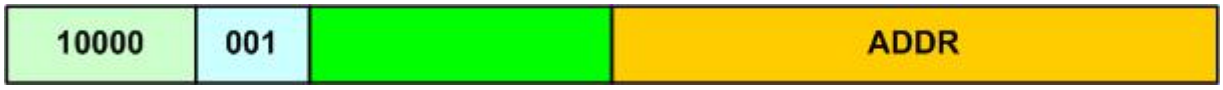
Операције скока

Безусловни скок по RB – Операција безусловног скока има заједнички код 10000 дефинисан преко виших пет битова поља COP. Садржај поља MOD = 000

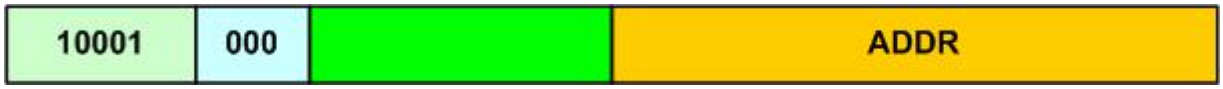
означава да је адреса скока дефинисана сумом садржаја регистра RB и поља ADDR инструкције.



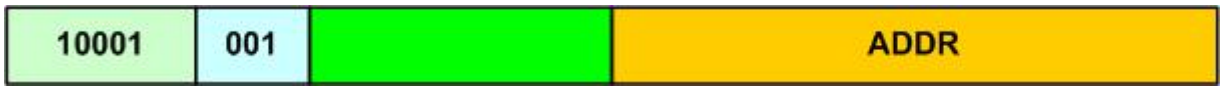
Безусловни скок по PC – Операција безусловног скока има заједнички код 10000 дефинисан преко виших пет битова поља COP. Садржај поља MOD = 001 означава да је адреса скока дефинисана сумом садржаја регистра PC и поља ADDR инструкције.



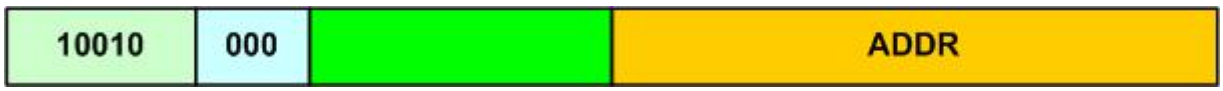
Условни скок по EQU и по RB – Операција условног скока по EQU има заједнички код 10001 дефинисан преко виших пет битова поља COP. Садржај поља MOD = 000 означава да је адреса скока дефинисана сумом садржаја регистра RB и поља ADDR инструкције.



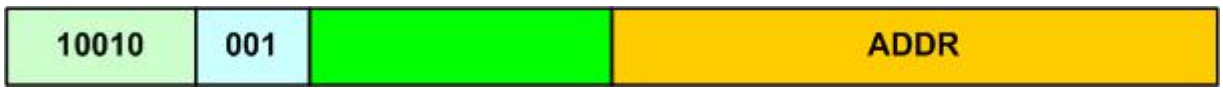
Условни скок по EQU и по PC – Операција условног скока по EQU има заједнички код 10001 дефинисан преко виших пет битова поља COP. Садржај поља MOD = 001 означава да је адреса скока дефинисана сумом садржаја регистра PC и поља ADDR инструкције.



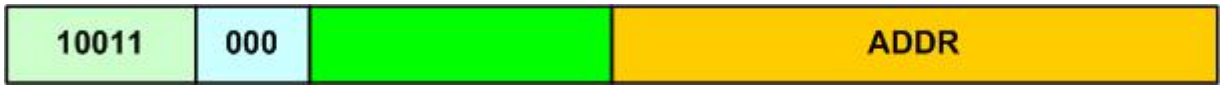
Условни скок по GE и по PB – Операција условног скока по GE има заједнички код 10010 дефинисан преко виших пет битова поља COP. Садржај поља MOD = 000 означава да је адреса скока дефинисана сумом садржаја регистра PB и поља ADDR инструкције.



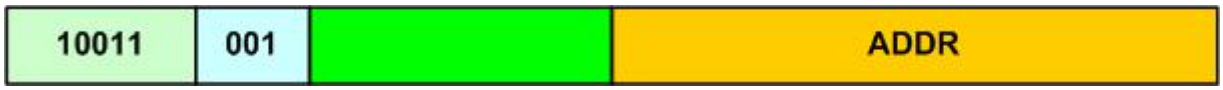
Условни скок по GE и по PC – Операција условног скока по GE има заједнички код 10010 дефинисан преко виших пет битова поља COP. Садржај поља MOD = 001 означава да је адреса скока дефинисана сумом садржаја регистра PC и поља ADDR инструкције.



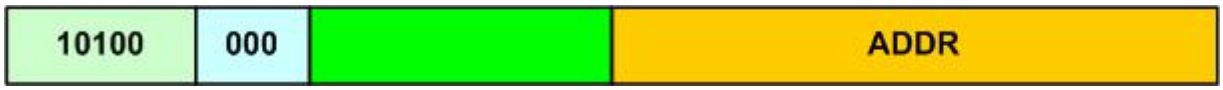
Условни скок по LE и по PB – Операција условног скока по LE има заједнички код 10011 дефинисан преко виших пет битова поља COP. Садржај поља MOD = 000 означава да је адреса скока дефинисана сумом садржаја регистра PB и поља ADDR инструкције.



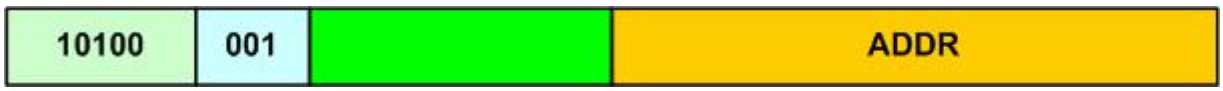
Условни скок по LE и по PC – Операција условног скока по LE има заједнички код 10011 дефинисан преко виших пет битова поља COP. Садржај поља MOD = 001 означава да је адреса скока дефинисана сумом садржаја регистра PC и поља ADDR инструкције.



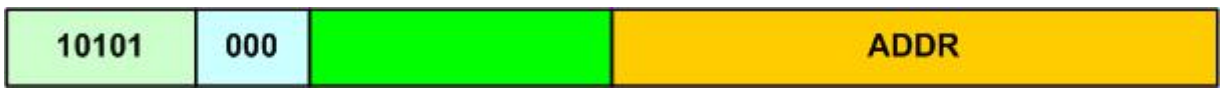
Условни скок по GT и по PB – Операција условног скока по GT има заједнички код 10100 дефинисан преко виших пет битова поља COP. Садржај поља MOD = 000 означава да је адреса скока дефинисана сумом садржаја регистра RB и поља ADDR инструкције.



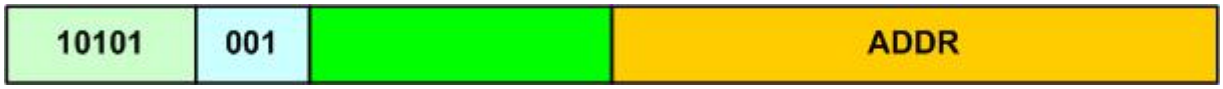
Условни скок по GT и по PC – Операција условног скока по GT има заједнички код 10100 дефинисан преко виших пет битова поља COP. Садржај поља MOD = 001 означава да је адреса скока дефинисана сумом садржаја регистра PC и поља ADDR инструкције.



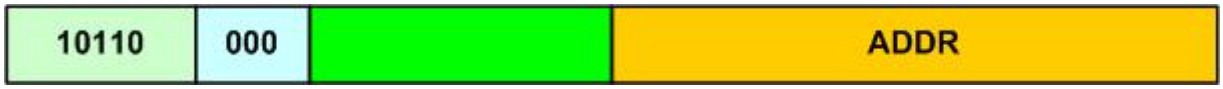
Условни скок по LT и по PB – Операција условног скока по LT има заједнички код 10101 дефинисан преко виших пет битова поља COP. Садржај поља MOD = 000 означава да је адреса скока дефинисана сумом садржаја регистра RB и поља ADDR инструкције.



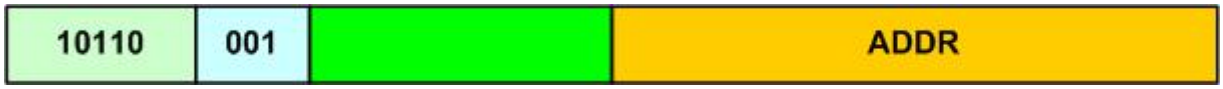
Условни скок по LT и по PC – Операција условног скока по LT има заједнички код 10101 дефинисан преко виших пет битова поља COP. Садржај поља MOD = 001 означава да је адреса скока дефинисана сумом садржаја регистра PC и поља ADDR инструкције.



Условни скок по C и по PB – Операција условног скока по C има заједнички код 10110 дефинисан преко виших пет битова поља COP. Садржај поља MOD = 000 означава да је адреса скока дефинисана сумом садржаја регистра PB и поља ADDR инструкције.



Условни скок по C и по PC – Операција условног скока по C има заједнички код 10110 дефинисан преко виших пет битова поља COP. Садржај поља MOD = 001 означава да је адреса скока дефинисана сумом садржаја регистра PC и поља ADDR инструкције.



5.3 Табела симболичких машинских инструкција

Код операције	Бинарни еквивалент	Адресни део
ADD	**00000000**	R3,R1,R2
ADI	**00000001**	R3,R1,Const
SUB	**00001000**	R3,R1,R2
SUI	**00001001**	R3,R1,Const
CPA	**00010000**	R1,R2
CPI	**00010001**	R1,Const
AND	**00011000**	R3,R1,R2
ANI	**00011001**	R3,R1,Const
OR	**00100000**	R3,R1,R2
ORI	**00100001**	R3,R1,Const
XOR	**00101000**	R3,R1,R2
XOI	**00101001**	R3,R1,Const
CPL	**00110000**	R1,R2
CLI	**00110001**	R1,Const
NOT	**00111000**	R3,R1
NOI	**00111001**	R3,Const
SLL	**01000000**	R3,Const

SLR	**01000001**	R3,Const
SAL	**01000010**	R3,Const
SAR	**01000011**	R3,Const
ROL	**01001000**	R3,Const
ROR	**01001001**	R3,Const
LDL	**01010000**	R3,Const
LDH	**01001001**	R3,Const
LDB	**01010010**	R3,Addr
LDR	**01010011**	R3,R1
LDM	**01010100**	R3,Addr
LDBX	**01010101**	R3,Addr
LDRX	**01010110**	R3,Addr
LDMX	**01010111**	R3,Addr
STR	**01011000**	R3,Addr
STR	**01011001**	R3,R1
STM	**01011010**	R3,Addr
STRX	**01011011**	R3,R1
IND	**01100000**	R3,Addr
OUTD	**01100001**	R3,Addr
PUSH	**01101001**	R3
POP	**01101001**	R3
JUMPD	**10000000**	Addr
JUMPP	**10000001**	Addr
JED	**10001000**	Addr
JEP	**10001001**	Addr
JGED	**10010000**	Addr
JGEP	**10010001**	Addr
JLED	**10011000**	Addr
JLEP	**10011001**	Addr
JGTD	**10100000**	Addr
JGTP	**00100001**	Addr
JLTD	**00101000**	Addr
JLTP	**00101001**	Addr
JCD	**10110000**	Addr
JCP	**10110001**	Addr

6. ЗАКЉУЧАК

Развој полупроводничке технологије и алата за описивање хардвера актуелизовао је потребу и могућност дефинисања специјализованих рачунарских компонената. То подразумева да се инжењери из области рачунарске технике срећу са могућношћу да специфицирају архитектуру таквих компонената, али и изазовом да самостално пројектују и реализују процесоре. Ово нарочито долази до изражаја у условима преласка на пројектовање система по концепту Hardware/Software Co – design чији резултат су савремена електронска решења позната као System on Chip (SoC) кола. Класичан пример ових система су мобилни телефони и слични уређаји.

У овом мастер раду приказан је приступ спецификацији архитектуре једног RISC процесора. Да би се сагледали сви аспекти поступка пројектовања једног процесора почев од саме његове архитектуре било је потребно поћи од дефиниције појма архитектуре рачунара и њене улоге у функционисању система. При томе су изучене постојеће врсте архитектуре и сагледане њихове карактеристике, предности и мане. Такође, сагледане су карактеристике рачунарских архитектура на којима су најчешће базирани савремени рачунари.

Након подсећања, али проширених знања о архитектури рачунара, њиховим врстама и карактеристикама, стечених током досадашњих студирања приступило се реализације односно специфицирању концепта архитектуре процесора FTN_RISC_Rev.1. На почетку су дате основне претпоставке на бази којих извршена спецификација архитектуре. Затим су анализирани различити захтеви и на крају могућа решења реализације архитектуре овог процесора.

На крају смо приказали архитектуру процесора FTN_RISC_Rev.1, где смо кроз дефинисање карактеристика овог процесора што обухвата дужину процесорске речи, регистре које користи, начине адресирања, формат инструкције видели како треба да изгледа спрецификација архитектуре једног процесора.

Искуства, стечена кроз реализацију овог мастер рада показују да савремени приступи у реализацији рачунарских система, пружају прилику да куповином или самосталним развојем модела рачунарског језгра и његовом надоградњом кроз моделирање додатног хардвера могу да се добију решења каква не постоје на тржишту

рачунарских компоненти. Важан аспект савременог пројектовања рачунара, укључујући и спецификације нових архитектура подразумева примену тзв. интегрисаног приступа пројектовању хардвера и софтвера (Hardware/Software Co – Design). Овај приступ, омогућава развој специјалзованих рачунарских компонената код којих је добар део алгоритамске подршке реализован директно у хардверу, што омогућава далеко ефикасније извршавање жељене обраде података.

У наставку рада на овој архитектури интересантно би било реализовати одговарајући процесор и на тестном софтверу проверити ефекете приступа у дефинисању његове архитектуре. Такође, занимљиво би било по сличном моделу реалозовати процесор код кога би се користила архитектура, која би имала уграђене карактеристике које су у процесу дефинисања актуелне архитектуре FTN_RISC_Rev.1 стављене у други план. На тај начин би се могло размишљати о моделу за поређење утицаја архитектурних решења на понашање процесора односно његове појединe перформансе.

ЛИТЕРАТУРА

[1] Blaaw, G. A., Brooks, F. P., "*Computer Architecture: Concepts and Evaluation*", 1st edition, Addison – Wrseley Professional, 1997

[2] Laplante, P. A., „*Dictionary of Computer Science – Engineering, and Technology*", CRC Press, 1998

[3] Mueller, S. M., Paul, W. J., „*Computer Architecture: Complexity and Correctness*", Springer, 2000

[4] Dandamudi, S. P., „*Guide to RISC Processors: for Programmers and Engineers*", Springer, 2005

[5] Heudin, J. C., Panetto, C., „*RISC Architecture*", Springer, 2008

[6] Shanley, T., "*x86 Set Architecture*", 1st edition, MindShare Press, 2010

[7] Hennessy, D. L., Patterson, D.A., „*Computer Architecture: A Quantitative Approach*", 5th edition, Morgan Kaufmann, 2011

[8] Stallings, W., „*Computer Organization and Architecture*", 9th edition, Pearson, 2012

[9] Catanzaro, B. J., „*The SPARC Technical Papers*", Springer, 2013

[10] Patterson, D. A., Hennessy, J. L., „*Computer Organization and Design:The Hardware/Software Interface* ", 5th edition, Morgan Kaufmann, 2013

[11] Stokes, J., „*Inside the Machine: An Illustrated Introduction to Microprocessors and Computer Architecture*", 1st edition, No Starch Press, 2015

[12] Bryant, R. E., O'Hallaron, D. R., „*Computer Systems: A programmers's Perspective*", 3rd edition, Pearson, 2015

[13] Neuenfeldt, C., „*ARM Architecture: RISC – Based Computer Design*", CreateSpace Independent Publishing Platform, 2016

Коришћени WEB ресурси

[14] http://www.mindshare.com/files/ebooks/x86%20Instruction%20Set%20Architecture.pdf

[15] http://www.gaisler.com/doc/sparcv8.pdf

[16] http://www.nxp.com/assets/documents/data/en/white-papers/POWRPCARCPRMRM.pdf

[17] https://imagination-technologies-cloudfront-assets.s3.amazonaws.com/documentation/MIPS_Architecture_microMIPS32_InstructionSet_AFP_P_MD00582_06.04.pdf

[18] https://www.scss.tcd.ie/~waldroj/3d1/arm_arm.pdf

[19] http://www.ifp.illinois.edu/~jones/RISCvCISCvDSP.pdf

[20] http://www.gaisler.com/doc/sparcv8.pdf

[21] http://pages.cs.wisc.edu/~fischer/cs701.f08/sparc.v9.pdf

[22] https://www.scss.tcd.ie/~waldroj/3d1/arm_arm.pdf

[23] http://bitsavers.informatik.uni-stuttgart.de/pdf/ibm/powerpc/_dataBooks/SR28-5124-00_PowerPC_Architecture_First_Edition_May93.pdf

[24] http://www.cs.cornell.edu/courses/cs3410/2015sp/MIPS_Vol1.pdf

[25] http://www.cs.cornell.edu/courses/cs3410/2015sp/MIPS_Vol2.pdf