УНИВЕРЗИТЕТ „Св. КИРИЛ И МЕТОДИЈ“
ВО СКОПЈЕ

ФАКУЛТЕТ ЗА ЕЛЕКТРОТЕХНИКА И
ИНФОРМАЦИСКИ ТЕХНОЛОГИИ - СКОПЈЕ

Институт за компјутерски технологии и инженерство

ПРОЕКТИРАЊЕ И ПРАКТИЧНА РЕАЛИЗАЦИЈА НА RISC-БАЗИРАНА МЕМОРИСКО-ЦЕНТРИЧНА ПРОЦЕСОРСКА АРХИТЕКТУРА И НЕЈЗИНА ПРИМЕНА

- докторски труд -

Ментор:
проф. д-р. Аристотел Тентов

Кандидат:
м-р Данијела Ефнушева

Скопје, 2017 година

Ментор: **Проф. д-р Аристотел Тентов**
Факултет за електротехника и информациски технологии, Скопје

Комисија: **Проф. д-р Тони Јаневски, претседател**
Факултет за електротехника и информациски технологии, Скопје

Проф. д-р Аристотел Тентов, ментор
Факултет за електротехника и информациски технологии, Скопје

Проф. д-р Едуард Сиеменс, член
Анхалт Универзитет за применети науки, Германија

Вонр. проф. д-р Татјана Николиќ, член
Електронски факултет при Универзитет во Ниш, Србија

Вонр. проф. д-р Катерина Ралева, член
Факултет за електротехника и информациски технологии, Скопје

Датум на одбрана:

Датум на промоција:

Научна област: технички науки, компјутерско инженерство

Посветено на двете мали срценца кои чукаат покрај моето срце

м-р Данијела Ефнушева

Докторски труд: Проектирање и практична реализација на RISC-базирана мемориско-центрична процесорска архитектура и нејзина примена

РЕЗИМЕ: Технолошкиот развој во областа на компјутерскиот хардвер и софтвер резултираше со широк спектар на брзи и евтини едно- или повеќе-јадрени процесори, компајлери, оперативни системи и програмски јазици, секој со свои предности и недостатоци, но со една заедничка цел за зголемување на целокупните перформанси на компјутерскиот систем. И покрај тоа што бројот на транзистори во чип постигна приближно дуплирање на секои две години, според Муровиот закон, сепак сè уште е многу тешко да се подобрат перформансите на секвенцијалните процесори, па дури и на паралелните повеќе-процесорски и повеќе-јадрени системи со споделена меморија. Главната причина за тоа се состои во појавата на тесно грло во комуникацијата помеѓу процесорот и меморијата во стандардните компјутерски системи, кои генерално се базираат на Фон Најмановата архитектура. Овој проблем е истражуван од многу научници, кои предлагаат различни начини за подобрување на мемориското доцнење и пропусност. Во овој докторски труд е даден преглед на тие техники и е направена детална анализа на различни мемориско-центрични системи, кои имплементираат различни пристапи за спојување или доближување на процесирачките елементи до меморијата. Во рамките на ова анализа се дискутирани предностите, недостатоците и подрачјето на примена на повеќе познати мемориско-центрични системи.

Главна цел на истражувањето кое е спроведено во овој докторски труд е да се развие и реализира нова RISC-базирана мемориско-центрична процесорска архитектура за да се обезбеди поголемо доближување на процесирачката логика до меморијата, каде се складирани податоците кои треба да се обработуваат и на тој начин да се избегне појава на тесно грло во комуникацијата помеѓу процесорот и меморијата. Имено, основната идеја во ова истражување е да се развие RISC-базиран процесор, кој интегрира меморија на заедничкиот чип и овозможува директен пристап до мемориските податоци, без при тоа да користи регистри за општа намена и кеш меморија, како редундантни ресурси во системот. За разлика од останатите познати чипови со процесирање во меморија, кои вообичаено не го напуштаат стандардниот хиерархиски модел на пристап до меморија, предложениот RISC-базиран мемориско-центричен процесор директно ги адресира податоците во меморијата во чипот без употреба на експлицитни Load и Store инструкции и имплементира специфична контролна единица која обезбедува проточно извршување на инструкциите во 4 фази (без MEM фаза), овозможувајќи секоја инструкција (аритметичка, логичка, гранење, контрола) да се извршува за еден такт циклус. Покрај тоа, предложениот RISC-базиран мемориско-центричен процесор е надграден со специфична логика која овозможува директен пристап до делови од податочните зборови кои не се со ширина на збор (word-aligned). Оваа логика може да се примени при

парсирање на заглавија на мрежни пакети и со тоа да овозможи директна работа со полиња од заглавија на пакети во случај кога предложениот процесор врши мрежно процесирање.

Предложениот RISC-базиран мемориско-центричен процесор е опишан во VHDL (Very High Speed Integrated Circuit Hardware Description Language) јазик и потоа е имплементиран во Virtex7 VC709 FPGA (Field Programmable Gate Array) со употреба на алатката Xilinx VIVADO Design Suite. Во докторскиот труд се прикажани и анализирани временски дијаграми од симулации и извештаи од синтеза (имплементација) на предложениот RISC-базиран мемориско-центричен процесор во FPGA. Покрај хардверски прототип, за предложениот RISC-базиран мемориско-центричен процесор е развиен и симулатор на ниво на инструкции, кој овозможува да се анализира состојбата на процесорот и да се прикаже бројот на процесорски циклуси, кои се потребни за извршување на различни програми. Овој симулатор е употребен при испитување на применливоста на предложената RISC-базирана мемориско-центрична архитектура за различен домен на апликации (со различен аритметички интензитет според Roofline моделот), со посебен акцент на апликациите кои побаруваат голема податочна пропусност, како што е на пр. процесирањето на пакети во мрежа. Резултатите од оваа анализа овозможуваат да се определи за каков тип на апликации предложената RISC-базирана мемориско-центрична архитектура воведува подобрување на перформансите, во споредба со други слични процесорски архитектури.

Главен придонес на ова истражување е генерирањето на нова RISC-базирана мемориско-центрична процесорска архитектура, која овозможува побрз и поедноставен пристап до податоците во меморијата, поголема внатрешна меморискa пропусност, избегнување на редундантни копирања и трансфер на податоци во регистрите и кеш меморијата и отстранување на комплексноста која ја воведува управувањето на оперативниот систем со нив. Овие промени овозможуваат предложениот RISC-базиран мемориско-центричен процесор да биде применлив при обработка на програми со голем и/или среден аритметички интензитет, пред сè поради тоа што тој обезбедува директен пристап до податоците, избегнувајќи ги временските застои кои се јавуваат при нивна размена помеѓу регистрите и кеш меморијата во стандардната мемориска хиерархија. Покрај тоа, предложениот процесор може да се примени и при обработка на мрежни пакети со висока пропусност, поради тоа што истиот вклучува дополнителна логика за парсирање на заглавија на пакети, која значително го поедноставува процесирањето и испитувањето на пакетите. Покрај овие придобивки, неопходно е да се истакне дека како резултат на истражувањето произлегува комплетен хардверски и софтверски прототип на предложениот RISC-базиран мемориско-центричен процесор.

КЛУЧНИ ЗБОРОВИ: процесорска архитектура; Фон Најманово тесно грло; процесирање во меморија; RISC архитектура; мемориско-центрична архитектура; FPGA; мрежно процесирање; парсер на заглавија на пакети; Roofline модел; аритметички интензитет;

Danijela Efnusheva, M.Sc.

PhD thesis: Design and practical implementation of RISC-based memory-centric processor architecture and its application

ABSTRACT: The technological advances in the area of computer hardware and software resulted in a wide range of fast and cheap single- or multi-core processors, compilers, operating systems and programming languages, each with its own benefits and drawbacks, but with the ultimate goal to increase the overall computer system performances. Although the number of transistors on a chip had doubled roughly every 18 months, according to the Moore's law, there is still difficult to improve the performances of the sequential processors, and even of the parallel multi-core and multi-processor shared-memory systems. The main reason for this resides in the well known Von-Neumann bottleneck which occurs during the communication between the processor and the main memory into a standard processor-centric computer system. This problem has been reviewed by many scientists, which proposed different approaches for improving the memory bandwidth and latency. A brief review of these techniques and a deep analysis of various memory-centric systems that implement different approaches of merging or placing the memory near to the processing elements are given in this PhD thesis. Within this analysis, the advantages, disadvantages and application (purpose) of several well-known memory-centric systems are discussed.

The aim of the research presented in this PhD thesis is to design and implement novel RISC (Reduced Instruction Set Computing)-based memory-centric processor architecture in order to provide stronger merge between the processor and the memory that holds the data that should be processed, and thus to avoid the occurrence of processor-memory bottleneck. The basic idea of this research is to develop a processor that integrates the memory into the same chip die and that allows direct access to the memory data, without the use of general purpose registers (GPRs) and cache memory, as redundant resources in the system. Contrary to the other memory/logic merged chips, which mostly use the standard memory hierarchy model for data access, the proposed RISC-based memory-centric processor directly addresses the data into the on-chip memory, without the use of explicit Load and Store instructions and includes specialized control unit that performs 4-stage pipelining of instructions (without MEM - memory access phase), allowing every (arithmetical, logical, branch, control) instruction to be completed in a single tact cycle. Besides that, the proposed RISC-based memory-centric processor implements a specialized logic that provides direct access to non word-aligned data words. This logic can be used for network packet headers parsing in order to allow direct access to packet header fields, when the proposed RISC-based memory-centric processor is applied in network processing.

The proposed RISC-based memory-centric processor is described in VHDL (Very High Speed Integrated Circuit Hardware Description Language) and then implemented in Virtex7 VC709 FPGA (Field Programmable Gate Array) board, by means of Xilinx VIVADO Design Suite. The simulation timing diagrams and the FPGA synthesis (implementation) reports for the proposed RISC-based memory-centric processor are discussed and analyzed in this PhD thesis. Besides the hardware prototype, a dedicated instruction-level simulator for the proposed RISC-based memory-centric processor architecture is also developed. This simulator is purposed to examine the processor's state and to present the number of executed processor cycles, during the execution of various programs. Therefore, the given simulator is used to explore the applicability of the proposed RISC-based memory-centric processor architecture in different types of applications (with diverse arithmetical intensity according to the Roofline model), with particular emphasis on applications that require large data bandwidth, such as packet processing in networks. The purpose of this analysis is to determine the type of applications for which the proposed RISC-based memory-centric processor architecture brings performance improvements, in comparison with other similar processor architectures.

The main contribution of this research is the development of novel RISC-based memory-centric processor architecture, which provides: faster and simplified access to data in memory, higher internal memory bandwidth, avoidance of redundant copy operations and data transfers into registers and cache memory, and removal of the memory management complexity that is involved by the operating system. All these modifications make the proposed RISC-based memory-centric processor applicable for execution of programs that are characterized with high and/or medium arithmetical intensity, mainly because the proposed processor provides direct access to the memory, avoiding the time consuming stalls which occur during the data exchanges between the registers and the cache memory into the standard memory hierarchy. Additionally, the proposed RISC-based memory-centric processor can be applied in high throughput network packet processing, because it includes a specialized logic for parsing of network packet headers, which can significantly simplify and speed-up the packet handling and processing. Besides these contributions, it is important to emphasize that as a result of this research a complete hardware and software prototype of the proposed RISC-based memory-centric processor is built.

KEY WORDS: processor architecture; Von Neumann bottleneck; processing in memory; RISC architecture; memory-centric architecture; FPGA; network processing; packet header parser; Roofline model; arithmetical intensity;

Содржина

Листа со слики

Листа со табели

Листа со кратенки

Кратенка	Значење	Страна
ALU	Arithmetical Logical Unit (Аритметичко-логичка единица)	35
ARM архитектура	Advanced RISC Machine архитектура	60
ASIC	Application Specific Integrated Circuit	53
ASIP	Application Specific Instruction Processor	116
BaseAdMode	Base Address Mode	76
BEQ	Branch on EQual	77
BLAS	Basic Linear Algebra Subprogram	176
BPI	Byte Peripheral Interface	143
CAD	Computer-Aided Design	52
CAS	Column Access Strobe	20
CISC	Complex Instruction Set Computing	17
CMOS	Complementary Metal Oxide Semiconductor	26
CPLD	Complex Programmable Logic Device (Комплексен програмабилен логички уред)	116
CPU	Central Processing Unit (Централна процесирачка единица)	27
DDR DRAM	Double Data Rate DRAM	4
DES алгоритам	Data Encryption Standard алгоритам	51
DIP прекинувач	Dual In-line Package прекинувач	144
DIVA	Data IntensiVe Architecture	5
DLX процесор	Deluxe процесор	30
DRAM	Dynamic Random Access Memory	3
eDRAM	Embedded (Вграден) DRAM	4
EPIC	Explicitly Parallel Instruction Computing	3
ESDRAM	Enhanced DRAM	4
EX	Execution	30
FCRAM	Fast Cycle DRAM	4
FeRAM	Ferroelectric RAM	3
FFT	Fast Fourier Transform	14
FIFO	First In First Out	60
FLOPS	Floating Point Operations Per Second	176
FMC	FPGA Mezzanine Connector	143
FP единица	Floating Point единица	34
FPGA	Field Programmable Gate Array	6
GFLOPS	Giga Floating Point Operations Per Second	42

GLUnix	Global Layer Unix	43
GOPS	Giga Operations Per Second	42
GPP	General Purpose Processor	115
GPR	General Purpose Register	5
GPU	Graphics Processing Unit	27
GTH трансивер	Gigabit трансивер	144
HBM	High-Bandwidth Memory	6
HDL	Hardware Description Language	136
HLT	Halt	103
HPC конектор	High Pin Count конектор	144
I.	Instruction	108
I/O (В/И)	Input/Output (Влез/Излез)	9
IBM компанија	International Business Machines компанија	27
IC	Integrated Circuit (Интегрирано коло)	137
ID or DE	Instruction Decode	30
IDEA	International Data Encryption Algorithm	51
IDS	Intrusion Detection System	116
IF or FE	Instruction Fetch	30
IFFT	Inverse Fast Fourier Transform	14
IHL	Internet Header Length	123
IN	Input	90
IP	Internet Protocol (Интернет протокол)	12
IPT	Inverted Page Table	67
IPv4	Internet Protocol version 4	12
IPv6	Internet Protocol version 6	12
IR	Instruction Register	56
IRAM	Intelligent (Интелигентен) RAM	5
ISA	Instruction Set Architecture	172
IW	Instruction Word	77
IXP	Internet Exchange Processor	14
JAL	Jump and Link	72
JTAG	Joint Test Action Group	142
LED	Light Emitting Diode	144
LUT	Lookup Table	124
LVDS	Low-Voltage Differential Signaling	143
MAC интерфејс	Medium Access Control интерфејс	121
MD5 алгоритам	Message Digest 5 алгоритам	51

MEM	Memory access	12
MEMRISC	Memory Access Reduced Instruction Set Computing	56
MIMOPS	Million Instructions on Memory Operands Per Second	59
MIPS	Million Instructions Per Second	11
MOP	Memory On Processor	45
MRAM	Magneto resistive Random Access Memory	3
NCD датотека	Native Circuit Description датотека	142
NGC датотека	Native Generic Circuit датотека	141
NGD датотека	Native Generic Database датотека	141
NOP	No operation	91
NOW	Network Of Workstations	43
NP	Network Processor	116
Op1	First operand (Прв операнд)	62
Op2	Second operand (Втор операнд)	62
OS	Operating System (Оперативен систем)	53
OUT	Output	90
P.A. (Ф.А.)	Physical Address (Физичка Адреса)	61
PAR	Place and Route	142
PC	Program Counter	35
PCI Express интерфејс	Peripheral Component Interconnect Express интерфејс	143
PDA	Personal Digital Assistant	50
PDE	Partial Differential Equation	176
PERL процесор	Performance Enhanced Register-Less процесор	4
PIM	Processing in memory (Процесирање во меморија)	5
PLD	Programmable Logic Device	137
PMBus конектор	Power Management Bus конектор	144
PPN	Parallel Prefix Network	38
PPRAM	Parallel Processing (Паралелен Процесирачки) RAM	43
RAM	Random Access Memory	3
RAS	Row Access Strobe	19
Res	Result operand (Резултатен операнд)	62
RISC	Reduced Instruction Set Computing	6
RLDRAM	Reduced Latency DRAM	4
RMAC	Remote Memory Access Controller	44
ROM	Read Only Memory	36
RSA алгоритам	Rivest Shamir Adleman алгоритам	51
RTL	Register Transfer Level	13

SALP	Subarray-Level Parallelism	4
SavePBR	Save Page and/or Base Registers	86
SetBR	Set Base Registers	76
SetPBR	Set Page and/or Base Registers	86
SetPR	Set Page Registers	74
SHA	Secure Hash Algorithm	51
ShiftAmt	Shift Amount	76
SIMD	Single Instruction Multiple Data	36
SIMM	Single In-line Memory Module	43
SMA такт	SubMiniature version A такт	144
SoC	System On Chip (Систем во чип)	116
SODIMM	Small Outline Dual In-line Memory Module	143
SOI	Silicon On Insulator	26
SPEC	Standard Performance Evaluation Corporation	19
SPECint	Integer performance testing component of the SPEC test suite	19
SRAM	Static Random Access Memory	4
STTRAM	Spin Transfer Torque RAM	32
TLB	Translation Look-aside Buffer	41
TL-DRAM	Tiered-Latency DRAM	4
TTL	Time To Live	121
UART интерфејс	Universal Asynchronous Receiver-Transmitter интерфејс	143
UCF	User Constraints File	141
USB	Universal Serial Bus	143
V.A. (B.A.)	Virtual Address (Виртуелна Адреса)	61
VCDRAM	Virtual Channel DRAM	4
VHDL	Very high speed integrated circuit Hardware Description Language	9
VHSIC	Very High Speed Integrated Circuit	136
VIRAM	Vector IRAM	40
VLIW	Very Long Instruction Word	3
WB	Write Back	30
XADC	Xilinx Analog to Digital Converter	144
XDC алатка	Xilinx Design Constraints алатка	137
XST компајлер	High-Level Synthesis компајлер	137

1. ВОВЕД

1.1. Преглед на научните достигнувања поврзани со предметот на истражување

Технолошкиот развој во текот на последните неколку децении предизвика драматични подобрувања на процесорските перформанси во насока на зголемување на фреквенцијата на работа и подобрување на паралелизмот преку зголемување на степенот на инструкции кои можат паралелно да се издаваат и обработуваат, [1], [2]. Во согласност со Муровиот закон, [3], [4], напредокот во технологијата на производство на интегрирани кола обезбеди дуплирање на бројот на транзистори во еден чип, на секои 18 месеци, што пак резултираше со креирање на повеќе-јадрени процесори во текот на последната деценија. Ваквиот напредок на процесорската технологија предизвика подобрување на перформансите на компјутерските системи, но не за сите типови на апликации, [5]. Причината за ваквата дивергентност се должи на појавата на тесно грло при интеракцијата помеѓу процесорот и главната меморија (која по правило се наоѓа надвор од процесорот), како последица на зголемениот пораст на брзината на работа на процесорот, во однос на брзината на работа на главната меморија, [6]. Според тоа, може да се каже дека во почетоците на развојот на компјутерските системи, главната меморијата сместена надвор од процесорскиот чип можеше да го опслужи процесорот со податоци со адекватна брзина. Сепак, годишниот пораст на процесорските перформанси од 70% и годишното подобрување на мемориското доцнење од само 7%, предизвикаа тековно да се потребни повеќе десетици такт циклуси за да се разменат податоци помеѓу процесорот и главната меморија, сместена надвор од процесорскиот чип, [7].

Компјутерските системи, кои се употребуваат во денешно време, во основа се базираат на Фон Нојмановата архитектура, [8], која се карактеризира со стриктна одвоеност на мемориските и процесорските ресурси во компјутерскиот систем. Во таков процесорско-центричен систем, меморијата служи за складирање на програмите и податоците, а процесорот ги интерпретира и

извршува програмските инструкции на секвенцијален начин, при што постојано пренесува податоци од главната меморија во процесорските регистри и обратно, [1]. Со цел да се избегне појавата на тесно грло при мемориските пристапи и да се обезбеди доближување на податоците до процесорот, денешните современи компјутерски системи вообичаено употребуваат кеш меморија организирана во повеќе нивоа, [2], која е генерално побрза, но помала и поскапа од главната меморија. Конкретно, Интеловите микропроцесори изработени во 65nm технологија употребуваат дури до 40% од површината на процесорскиот чип за имплементирање на кеш меморија, [9], [10], која се користи стриктно за редуцирање на мемориското доцнење.

Особено влијание на ефикасноста на кеш меморијата имаат параметрите: рата на промашување и време потребно за опслужување на промашување во кешот, [1]. Со цел да се обезбеди подобрување на овие параметри се употребуваат различни техники, како што се: зголемување на димензиите на кешот и/или неговите блокови, примена на поголема асоцијативност, [1], употреба на жртвен или псевдо-асоцијативен кеш, [11], [12], поделба на кешот во повеќе нивоа, работа со под-блокови, [13], и користење на неблокирачка кеш меморија, [14]. Овие методи најчесто го подобруваат времето на извршување на програми кои се карактеризираат со голема временска и/или просторна локалност во мемориските пристапи.

Покрај големата примена на кеш меморијата во современите компјутерски системи, неопходно е да се нагласи дека секое ниво во кеш меморијата претставува редундантна копија на податоците од главната меморија, која не би била потребна доколку главната меморија би можела да му парира на процесорот, во поглед на брзината на работа. И покрај тоа што кеш меморијата позитивно влијае на намалувањето на времето на пристап до меморија, сепак таа побарува постојано движење и копирање на истите податоци од повисоките кон пониските нивоа во мемориската хиерархија и обратно, што пак доведува до зголемување на потрошувачката на енергија во системот, дури до 40%, [15]. Освен тоа, имплементирањето на кеш меморија воведува зголемена комплексност во компјутерскиот систем, бидејќи употребува дополнителни хардверски ресурси, како и сложени механизми за одржување на конзистентноста во меморијата.

Дополнително, промашувањата во кеш меморијата предизвикуваат непредвидливост во времетраењето на програмите кои се извршуваат, што не е многу погодно за системите кои работат во реално време.

Другите техники за намалување или прикривање на мемориското доцнење комбинираат употреба на голем кеш со различни пристапи на процесирање, како што се: нередоследно и шпекулативно извршување на инструкции, претходно земање на инструкции и податоци, примена на алгоритми за предвидување на гранење и повеќенитност, [16], [17]. Овие техники воведуваат дополнително зголемување на површината на чипот и неговата комплексност, на хардверско и софтверско ниво, [18]. Слична е судбината и на некои моќни процесорски архитектури, како што се: векторските процесори, [19], суперскаларните процесори, [20], процесорите со долг инструкциски збор (VLIW - Very Long Instruction Word), [21] и експлицитно паралелните процесори (EPIC - Explicitly Parallel Instruction Computing), [22]. Нивните проблеми се поврзани со комплексност при имплементација, слаба искористеност на расположливите ресурси, недоволно развиена компајлерска технологија и добар модел на програмирање, на сметка на мали подобрувања на перформансите. Последното се должи на дополнителното мемориско доцнење, кое се јавува како резултат на недоволната мемориска пропусност за да се опслужи зголемениот мемориски сообраќај (изразен преку покачената рата на издадени инструкции и потреба за операнди). Од друга страна, интеграцијата на повеќе процесори или јадра во заеднички чип воведува уште поголеми барања за меморискиот систем, [23]. Всушност, за чип со било колкава површина, со интегрирање на повеќе процесори/јадра, се намалува големината на меморија во чипот, што пак предизвикува да се зголеми бројот на спорите пристапи до меморијата, надвор од чипот.

Покрај предложените промени во архитектурата на процесорските чипови, направени се и други истражувања во кои се предлага комплетна замена на DRAM (Dynamic Random Access Memory) технологијата со нова или конкретни промени во самата архитектурата на DRAM меморијата. Првата категорија предлага некои нови мемориски технологии, како што се: мемристор, Magneto-resistive Random Access Memory (MRAM), и Ferroelectric RAM (FeRAM), [4]. Во втората категорија спаѓаат повеќе различни модификации на DRAM

технологијата, меѓу кои се: асиметричен DRAM со неуниформен пристап до DRAM банките, [24]; Reduced Latency DRAM (RLDRAM), [25], кој овозможува адресирање во еден такт циклус како SRAM (Static Random Access Memory); Fast Cycle DRAM (FCRAM), [25], кој го дели секој ред од DRAM блоковите во повеќе под-редови; Enhanced DRAM (ESDRAM) и Virtual Channel DRAM (VCDRAM), [25], кои додаваат SRAM бафер до DRAM меморијата за да ги кешира најчесто пристапуваните податоци; SALP (Subarray-Level Parallel) систем, [26], кој овозможува преклопување на доцнењата на различни барања кон иста банка; Tiered-Latency DRAM (TL-DRAM), [27], кој користи пократки битски линии со помалку клетки; хибридна мемориска коцка, [28], која овозможува повеќе мемориски модули да се сместат еден врз друг во облик на 3D коцка; и вграден DRAM (embedded DRAM - eDRAM), [29], [30], кој се интегрира на истиот чип со процесорот. Покрај овие предлози, направени се и други истражувања, кои резултираат со развој на нови мемориски клетки, [31], како хибридно решение на она што го нудат DRAM и SRAM технологиите. Генерално, може да се каже дека дискутираните предлози сè уште не можат да ја надминат DDR (Double Data Rate) технологијата, бидејќи се соочуваат со повеќе проблеми, како што се: зголемена комплексност на меморискиот чип, поскапа цена на чинење, намалена густина на чипот и ограничена применливост во кеш меморија или меморија за вградливи компјутерски системи, [24] - [26].

И покрај тоа што процесорите и меморијата постигнале значителен напредок и развој како засебни компоненти во процесорско-центричните компјутерски системи, компјутерските архитекти уште во 1990^тите години предвиделе дека однесувањето на меморијата ќе има големо влијание врз перформансите на компјутерскиот систем, [32]. Како резултат на тоа, досега се истражувани повеќе алтернативни пристапи на мемориско-центрично сметање, кои предлагаат различни начини на вградување или приближување на меморијата до процесорот. Тука спаѓаат следните предлози: безрегистарски (PERL - Performance Enhanced Register-Less) процесор, [33], кој ги извршува сите операции директно со кеш меморијата, организирана во повеќе нивоа (во и надвор од чипот); употреба на бележник (Scratchpad) меморија, [34], [35], како мала софтверски управувана меморија во чипот, со време на пристап од еден такт циклус;

различни PIM (Processing in memory) чипови кои интегрираат процесирање и меморија на заеднички чип (пресметковен RAM, [36], SUN PIM чип, [37], Mitsubishi 32R/D чип, [38], Terasys PIM чип, [39], Data IntensiVe Architecture - Diva PIM чип, [40], [41], интелигентен RAM - IRAM, [42], [43], паралелен процесирачки RAM, [44] и DataScalar систем, [45]); и интелигенти мемориски системи, како што е моделот на активни страници, [46], кој доделува реконфигурабилни логички блокови на секоја виртуелна страна во меморијата. Во рамките на овие мемориско-центрични системи процесорот може да биде едноставен или софистициран суперскаларен процесор, а при тоа може да вклучува и векторска единица, како што е случајот со интелигентниот RAM, [43]. Меморијата во чипот може да биде имплементирана како SRAM или вграден DRAM, при што вообичаено до нејзе се пристапува преку кеш меморијата на процесорот, [20]. И покрај тоа што процесирањето во или блиску до меморијата воведува подобрување на мемориското доцнење и пропусноста, сепак поголемиот дел од предложените мемориско-центрични систем губат непотребни временски и хардверски ресурси за копирање и пренесување на податоците помеѓу меморијата во чипот, кешот во чипот и регистрите за општа намена (GPRs - General Purpose Registers). Покрај тоа, брзината на работа на чипот, количината на меморија во чипот и неговата цена се ограничени од применетата технологија и процесот на производство. Од друга страна, уште поголем предизвик за мемориско-центричните системи е да се развие соодветна компајлерска поддршка, која ќе го препознае паралелизмот во програмите и ќе овозможи ефикасно искористување на внатрешната мемориска пропусност во чипот (пр. широка податочна патека во Diva чип, [39]).

Разгледаните мемориско-центрични чипови се предложени во истражувања спроведени непосредно пред и после 2000-та година, во временски период кога процесорите и меморијата веќе биле развиени во посебни индустрии, со различни цели, намени и технологии на производство. Со оглед на тоа дека во тој временски период сè уште постоеле можности за технолошки развој на процесорите во насока на зголемување на брзината и паралелизмот со употреба на повеќе јадра/процесори, предложените мемориско-центрични чипови не доживеале голем успех (освен за наменски вградливи системи). Имајќи на ум дека современите процесори во последно време се соочуваат и со технолошки и

со физички ограничувања за подобрување на своите перформанси, додека пак капацитетот на главната меморија е во постојан пораст, [1], може да се каже дека сега е вистински момент за повторно да се истражи идејата за доближување/интегрирање на процесорот со меморијата со цел да се надмине разликата во нивната брзина на работа. Најновите истражувања во таа насока се од компанијата Интел, која предлага употреба на 10nm Stratix FPGA плочки, [47], кои интегрираат до два HBM2 (high-bandwidth memory) високо-пропусни мемориски чипови во едно пакување, овозможувајќи да се постигне максимална пропусност до 512 GB/s.

Инспирирани од идејата за разбивање на стандардната повеќе-нивовска мемориска хиерархија, во овој докторски труд се предлага нова RISC-базирана мемориско-центрична процесорска архитектура, слична на PERL, [33], која обезбедува директен пристап до меморијата која е интегрирана во чипот, без при тоа да употребува регистри за општа намена и кеш меморија. Предложената замена на најгорните две нивоа од мемориската хиерархија со меморија во чип овозможува да се отстранат непотребни копирања на податоци и да се избегне трошење на временски и хардверски ресурси за блоковски пренос на податоци во кешот и нивен поединечен трансфер во регистрите за општа намена, [32], [48] - [53]. Главен фокус во докторскиот труд е да се испита како овие промени ќе влијаат на применливоста на предложената процесорска архитектура за различен домен на апликации, [54] - [66].

1.2. Истражувачки проблем

Појавата на Фон Нојманово тесно грло во комуникацијата помеѓу процесорот и меморијата претставува проблем со кој долги години се соочуваат стандардните процесорско-центрични компјутерските системи. Сепак, до денес не е пронајдено дефинитивно решение на овој проблем туку се користат само различни техники за негово "ублажување", најчесто преку имплементирање на кеш меморија во повеќе нивоа помеѓу процесорот и главната меморија. Оттука може да се каже дека досега не се обрнуваше многу внимание на решавање на проблемот на тесно грло, туку истражувањата беа најмногу насочени кон

креирање на побрзи и помоќни процесори кои ќе обезбедат висок степен на паралелизам и со тоа ќе влијаат на подобрување на перформансите на компјутерскиот систем. Сепак, Муровиот закон во текот на последните две години почна да стагнира, [3], [4], поради физичките ограничувања на транзисторите, укажувајќи дека процесорите ги достигнаа своите граници во брзината на извршување на пресметки. Дури и развојот на повеќе различни мемориско-центрични системи кои значително ги подобруваат внатрешната мемориска пропусност и мемориското доцнење не постигна значителен успех, поради технолошките ограничувања на споениот мемориско-пресметковен чип и нецелосното напуштање на стандардниот модел на пристап до меморијата, преку регистрите за општа намена и кеш меморијата. Оттука се поставува прашањето дали стандардната мемориска хиерархија е сè уште оптимално решение за ефикасен пристап до податоците во општ случај? Потоа се наметнува прашањето дали со користење на различна мемориска организација и техники за управување со податоците може да се постигнат подобри перформанси? Доколку нема подобро решение во општ случај, тогаш се поставува прашањето во кои специфични случаи може да се постигнат подобри резултати со определено предлог решение? Исто така останува отворено и прашањето дали со стандардната мемориска организација може да се постигне висока податочна пропусност за да се задоволат потребите на определени апликации (пр. повеќе Gb/s во мрежно процесирање на пакети)?

Имајќи ја во предвид значително поголемата брзина на процесорот во однос на главната меморија, како главна пречка за постигнување на висока податочна пропусност може да се смета релативно малиот број на регистри кои се достапни во процесорите. Ова е особено очекувано при извршување на апликации кои работат со поголем обем на податоци кои се потребни кратко време во процесорот. Примери за вакви апликации вклучуваат: обработка на податочни текови, пресметување на обемни аритметичко-логички изрази и изминување на сложени податочни структури. Брзината на пристап до процесорските регистри во вакви случаи не дава голема предност бидејќи во нив не можат навремено да се сместат сите податоци. Доколку се пронајде друг брз начин податоците да пристигнуваат до процесорот, тогаш може да се разми-

слува и за отфрлање на регистрите. Дополнително, може да се испитаат и можностите за отфрлање на кеш меморијата, со цел да се реши проблемот со непредвидливост на времето на извршување на програмите кои интензивно користат кеш меморија. Се очекува дека овој пристап ќе овозможи прецизно одредување на временските карактеристики на извршување на програмите. Ова е од исклучително значење кај системите кои работат во "реално време", [67], каде е потребно почитување на строги рокови при извршувањето.

1.3. Предмет на истражување

Предметот на истражување во оваа докторска дисертација е испитување на можностите за развој, реализација и применливост на нова RISC-базирана мемориско-центрична процесорска архитектура, која предлага интеграција на процесорот и меморија на заеднички чип, отфрлајќи ги регистрите за општа намена и кеш меморијата (во и надвор од чипот) од стандардната мемориска хиерархија. За разлика од останатите познати чипови со процесирање во меморија, кои најчесто не го напуштаат стандардниот хиерархиски модел на пристап до меморија, во ова истражување се предлага развој на RISC-базиран мемориско-центричен процесор, кој ќе обезбеди директен пристап до податоците во меморија во чипот (без да употребува експлицитни LOAD и STORE инструкции) и ќе вклучува специфична контролна единица, која ќе овозможи проточно извршување на инструкциите без примена на регистри за општа намена, при што сите инструкции (аритметички, логички, гранење, контрола) ќе се извршуваат во само еден такт циклус. Во почетната фаза на ова истражувањето потребно е да се проектира специфична архитектура на инструкциско множество за предложениот процесор, и да се обезбедат неговите функционалности со дефинирање на повеќе компоненти, вклучувајќи: контролна единица со поддршка за повеќе адресни режими, аритметичко-логичка единица за работа со цели и реални броеви, единици за проточно работење, единица за справување со податочни меѓу-зависности, единици кои обезбедуваат хардверска поддршка за работа со виртуелна меморија (управување со мемориски блокови и табели за преведување), специфична логика која овозможува работа со делови од податочните

зборови кои не се со ширина на збор (се применува при парсирање на заглавија на мрежни пакети), механизми за справување со прекини, контрола на В/И (влез/излез) и дополнителна логика за контрола и селекција. Следен чекор во истражувањето е да се испитаат можностите за реализација на предложената RISC-базирана мемориско-центрична архитектура и креирање на хардверски прототип со употреба на FPGA компонента, [68] - [70]. Оваа фаза на истражување опфаќа експериментирање со VIVADO Design Suite развојната околина, [71], и креирање на VHDL модел за предложениот процесор, кој ќе служи за симулирање на неговата работа и анализирање на неговите карактеристики и комплексност, при FPGA синтеза (имплементација). Последен чекор во истражувањето е да се испита применливоста на предложената RISC-базирана мемориско-центрична архитектура за различен домен на апликации (според Roofline моделот, [1]), со особен акцент на апликации кои побаруваат голема податочна пропусност (пр. мрежна обработка на пакети, [72] - [74]). За да може да се спроведе ова истражување, потребно е да се разгледаат можностите за развој на симулатор на инструкциско ниво за предложената RISC-базирана мемориско-центрична архитектура. Овој симулатор ќе биде наменет за симулација на извршувањето на различни тест програми, при што добиените резултати ќе бидат анализирани и употребни за споредба со експериментални резултати за други слични процесорски архитектури, [75]. Оваа фаза во истражувањето овозможува да се определи типот на апликации за кои предложената RISC-базирана мемориско-центрична архитектура резултира со подобрување во времето на извршување.

1.4. Цели и задачи на истражувањето

Целта на истражувањето претставено во овој докторски труд е да се предложи и реализира нова RISC-базирана мемориско-центрична процесорска архитектура, за да се обезбеди поголемо доближување на процесирачката логика до меморијата, каде се складирани податоците кои треба да се обработуваат, со што ќе се надмине проблемот на појава на тесно грло во комуникацијата помеѓу процесорот и меморијата. Основната идеја во истражувањето е да се предложи и реализира решение кое интегрира процесор и меморија на

заеднички чип, при што процесорот може директно да пристапува до податоците (кои се или не се со ширина на збор) од својата меморија, без да користи регистри за општа намена и кеш меморија, како редундантни ресурси во системот. Целта на овој пристап е да се избегнат непотребни копирања и индивидуален или блоковски пренос на податоци во регистрите за општа намена и кешот; да се намали количината на редундантни мемориски ресурси; да се поедностави пристапот до меморија; и да се избегне имплементирање на комплексни механизми за управување со меморијата. Од особен интерес е да се испита како овие промени ќе влијаат на применливоста на предложената RISC-базирана мемориско-центрична архитектура за различен тип на апликации со различен аритметички интензитет и со големи барања за податочна пропусност. Целта на ова истражување е да се определи за каков тип на апликации предложената RISC-базирана мемориско-центрична архитектура воведува подобрување на перформансите, во споредба со други слични процесорски архитектури. Дополнително, со хардверската реализација на предложената RISC-базирана мемориско-центрична архитектура може да се определи колкава е комплексноста на проектираниот процесор и каква е исплатливоста за негова примена во апликациите, за кои постигнува подобри перформанси.

1.5. Структура на докторскиот труд

Целиот процес на истражување кој е претставен во докторскиот труд е структуриран во осум глави. Воведот и заклучокот се водат како прва и осма глава, соодветно. Сите глави се систематизирани во точки и потточки со наслови и поднаслови, за да се овозможи поедноставно следење на материјата што е обработена во истражувањето.

Во воведот е претставена потребата и поттикот за истражување од областа на архитектури на процесори. Најпрво се дискутирани научните постигнувања во определената област на истражување, а потоа е идентификуван проблемот кој ќе се разработува во докторскиот труд. Веднаш потоа е претставен предметот на истражување и зададени се целите кои треба да се постигнат како резултат на истражувањето.

Во втората глава е претставен проблемот на Фон Нојманово тесно грло кое се појавува при мемориските пристапи кај стандардните процесорско-центрични компјутерски системи. Овој проблем е последица на големата разлика во брзините на работа на процесорот и меморијата, и укажува дека при извршување на мемориски операции за читање или запишување доаѓа до трошење на многу процесорски циклуси за пренос на податоци помеѓу регистрите, кешот и главната меморија. Проблемот на тесно грло кое се јавува помеѓу процесорот и меморијата е проучуван од многу научници, кои предлагаат различни начини за подобрување на мемориската пропусност и доцнење. Во оваа глава е направен преглед на овие техники и воедно се дискутирани предностите кои ги воведува технологијата на вграден DRAM, при имплементирање на голема кеш меморија во чипот, со капацитет од повеќе мегабајти.

Во третата глава е дадена детална анализа на повеќе мемориско-центрични пристапи на интегрирање или доближување на меморијата до процесирачките елементи и при тоа се дискутирани неколку предлози на модифицирање на стандардната мемориска хиерархија со цел да се обезбеди побрз пристап до главната меморија. Во оваа анализа, покрај архитектурата и организацијата, претставени се предностите, недостатоците и подрачјето на примена на неколку познати мемориско-центрични системи, вклучувајќи : безрегистарски процесор кој користи само кеш меморија (внатре и надвор од чипот) за да комуницира со главната меморија, процесори кои ја надградуваат кеш меморијата во чипот со посебна и мала брза софтверски-управувана бележник меморија, чипови кои интегрираат процесирање и меморија (SUN PIM, Mitsubishi M32R/D, Terasys, Diva, интелигентен RAM, паралелен процесирачки RAM и DataScalar), и интелигентен мемориски систем со активни станици (каде активна страница е спој на реконфигурабилна логика и виртуелна мемориска страница).

Во четвртата глава е даден предлог за дизајнирање на RISC-базирана мемориско-центрична процесорска архитектура, која обединува RISC-базиран процесор и локална меморија во чипот, организирана во блокови. За да се дизајнира предложениот RISC-базиран мемориско-центричен процесор, имплементирани се неколку промени во стандарден RISC процесор (MIPS - Million Instructions Per Second), вклучувајќи: отфрлање на регистрите за општа намена и

кеш меморијата во чипот и нивна замена со локална меморија во процесорот; директен пристап до податоците во локалната меморија на процесорот при извршување на инструкциите; намалување на бројот на фазите при проточното работење со отфрлање на MEM фазата; поедноставување на инструкциското множество (отфрлање на експлицитни LOAD/STORE операции); завршување на операциите на условно и безусловно гранење во фазата на декодирање; надминување на податочните меѓу-зависности при директна работа со локалната меморијата во чипот; поддршка за инструкции форматирани со различни адресни режими со употреба на специфична контролна единица; хардверско управување со мемориските блокови и табелите за преведување во чипот; и справување со исклучоци и контрола на В/И. Во оваа глава е претставен целиот процес на креирање на предложениот RISC-базиран мемориско-центричен процесор и при тоа е даден детален опис на неговата специфична архитектура на инструкциско множество и применетиот пристап на сегментација на локалната меморија во чипот со хардверска поддршка за работа со виртуелната меморија и воедно се објаснети поважните компоненти од податочната и контролната патека на процесорот. Предложениот RISC-базиран мемориско-центричен процесор поддржува операции со цели и реални броеви, овозможувајќи сите инструкции да се извршуваат проточно во податочната патека, под надзор на компонентите од контролната патека.

Во петтата глава е даден предлог за дизајнирање на специфична хардверска логика која се додава до локалната меморија во чипот на предложениот RISC-базиран мемориско-центричен процесор со цел да обезбеди директен пристап до мемориските податоци кои не се со ширина на збор. Оваа специфична хардверска логика е имплементирана како парсер на заглавија на IP (Internet Protocol) пакети, кој овозможува директна работа со полиња од IPv4 или IPv6 заглавија на пакети, кои не се со ширина на збор. Предложениот парсер на заглавија на IP пакети се додава до локалната меморија во чипот (која содржи повеќе заглавија на IP пакети) и на тој начин овозможува предложениот RISC-базиран мемориско-центричен процесор да се употребува во мрежно процесирање на пакети. Оттука може да се каже дека предложената логика за парсирање на IP заглавија обезбедува исто време на пристап до поле од

заглавие на IP пакет, како и времето на пристап до било кој податочен збор од локалната меморија на процесорот, на тој начин што користи посебен дел од меморискиот адресен простор за директно да ги адресира полињата од IP заглавијата на пакетите. Во тој контекст, опишана е и применетата шема на адресирање, која имплементира техника на мемориски прекари. Со оглед на тоа дека пристапот до полињата од IP заглавијата на пакетите е многу честа операција при мрежното процесирање на пакети, предложениот парсер на IP заглавија на пакети овозможува да се избегнат многу непотребни аритметички и логички инструкции, при екстрахирање на полињата од IP заглавијата, кои треба да се обработуваат.

Во шестата глава е претставен процесот на проектирање и практична реализација на предложениот RISC-базиран мемориско-центричен процесор во FPGA плочка. Ова вклучува изработка на хардверски модел на предложениот процесор во VHDL јазик за опис на хардвер и експериментирање со Xilinx VIVADO Design Suite софтверската околина, која поддржува работа со повеќе серии на Virtex FPGA компоненти, меѓу кои е и Virtex7 VC709 FPGA развојната плочка. Во тој процес најпрво е направено симулирање на хардверскиот модел на предложениот RISC-базиран мемориско-центричен процесор со помош на Xilinx VIVADO Design Suite Simulator алатката. Симулацијата е извршена со тест програми кои генерираат временски дијаграми со кои се анализира однесувањето на хардверскиот модел на предложениот процесор и сите негови составни компоненти. Потоа, со алатките за синтеза и имплементација е генериран RTL (Register Transfer Level - ниво на пренос на регистри) модел на предложениот процесор и е направена имплементација на синтетизираниот процесор во Virtex7 VC709 FPGA плочка. Со помош на извештаите кои се генерираат од овие алатки, извршена е анализа на хардверските карактеристики на имплементираниот процесор. На крај е претставен начинот на В/И мапирање на интерфејсите на предложениот процесор со пиновите од Virtex7 VC709 FPGA плочката и воедно е прикажан процесот на програмирање на оваа Virtex7 VC709 FPGA плочка. Креираниот хардверски прототип овозможува да се симулира и анализира работата на предложениот RISC-базиран мемориско-центричен процесор во реален хардвер, односно во Virtex7 VC709 FPGA компонента.

Во седмата глава е направена анализа на применливоста на предложениот RISC-базиран мемориско-центричен процесор за различен домен на апликации, преку извршување на повеќе тест програми и споредба на добиените резултати со експериментални резултати за слични процесорски архитектури. За да се спроведе оваа анализа, изработен е специфичен симулатор на инструкциско ниво за предложениот RISC-базиран мемориско-центричен процесор. Покрај тоа употребен е јавно достапен симулатор на инструкциско ниво - MARS за RISC-базиран MIPS процесор, и воедно употребени се резултати од слични истражувања направени за други слични процесори. Во првото испитување е претставено симулирање на тест програми кои се карактеризираат со различен аритметички интензитет, според Roofline моделот. Ова вклучува: програма со голем аритметички интензитет, која имплементира алгоритам за множење на пополнети матрици со различни димензии; програма со среден аритметички интензитет, која извршува FFT/IFFT (Fast Fourier Transform/Inverse Fast Fourier Transform) пресметки со различен број на точки, според познатиот Cooley–Tukey алгоритам; и програма со мал аритметички интензитет, која имплементира решавање на парцијални диференцијални равенки. Со анализа на времето на извршување на овие програми за предложениот процесор и за слични процесорски архитектури (пр. MIPS, PERL, интелигентен RAM) се покажа дека предложениот RISC-базиран мемориско-центричен процесор постигнува подобрување на перформансите за програмите, кои се карактеризираат со голем и среден аритметички интензитет. Второто испитување вклучува симулирање на програми за мрежна обработка на IPv4 и IPv6 пакети, со цел да се испита применливоста на предложениот RISC-базиран мемориско-центричен процесор во апликации со барања за висока податочна пропусност, како што е мрежно процесирање на пакети. Со оглед на тоа дека предложениот RISC-базиран мемориско-центричен процесор вклучува специфичен хардвер за парсирање на IP заглавија на пакети се покажува дека предложениот процесор воведува подобрување во споредба со иницијалниот RISC-базиран MIPS процесор, постигнувајќи слични перформанси како други познати мрежни процесирачки јадра (Intel Internet Exchange Processor - IXP микро единици).

Во последната глава се дадени заклучни разгледувања за поставените и остварените цели од изработката на овој докторски труд. Во однос на тоа, потенцирани се придонесите од истражувањето и дефинирани се подрачјата на примена на предложениот RISC-базиран мемориско-центричен процесор. Дополнително, претставени се ограничувањата на предложениот RISC-базиран мемориско-центричен процесор и дадени се насоки и предлози за идна работа.

На крајот од докторскиот труд е дадена листа со литература, која е употребена во ова истражување. Оваа листа опфаќа релевантни книги од подрачјето на архитектури на процесори и компјутери, научни трудови објавени во значајни меѓународни списанија и зборници од познати меѓународни конференции, како и магистерски/докторски трудови со слична област на истражување, како овој докторски труд. Голем дел од употребената литература произлегува од истражувањата кои се претставени во овој докторски труд.

2. ПРЕГЛЕД НА РАЗЛИЧНИ ПРИСТАПИ ЗА НАДМИНУВАЊЕ НА ТЕСНОТО ГРЛО ВО КОМУНИКАЦИЈАТА ПОМЕЃУ ПРОЦЕСОР И МЕМОРИЈА

2.1. Тесно грло во комуникација помеѓу процесор и меморија

Технолошкиот развој во текот на последните дваесет години предизвика голем напредок во компјутерските системи, на хардверско и софтверско ниво. Ваквиот развој беше поддржан од Муровиот закон, кој обезбеди дуплирање на бројот на транзистори во чип, на секои 18 месеци, [3], [4]. И покрај тоа што појавата на повеќе-јадрените процесори овозможи да се постигне подобрување на перформансите кај компјутерските системи за определени апликации, сепак мулти-процесирањето предизвика зголемени барања за меморискиот систем, [3]. Ова се должи на порастот на процесирачките ресурси во чипот, без соодветно зголемување на меморијата во чипот. Во таков небалансиран систем се појавува тесно грло при комуникација помеѓу процесорот и главната меморија, која по правило е сместена надвор од процесорот. Со оглед на разликата во брзините помеѓу процесорот и главната меморија, секоја мемориска операција на читање или запишување може да предизвика трошење на многу процесорски циклуси за трансфер на податоци помеѓу регистрите за општа намена, кешот и главната меморија, или обратно.

Главната причина за големата разлика во брзините на процесорот и меморијата е поделбата на полупроводничката индустрија во две засебни гранки: една за дизајн на процесори и друга за производство на меморија. Првата е наменета за развој на брза логика (со употреба на побрзи транзистори, во согласност со Муровиот закон), додека пак втората обезбедува зголемување на капацитет за складирање на податоци (со употреба на помали мемориски клетки), [6]. Како резултат на овие различности, процесорот и меморијата се произведуваат со свој сопствен процес на производство, како посебни компоненти. Процесорите се сместуваат во скапи пакувања кои се карактеризираат со голема потрошувачка на енергија (5 - 50 W) и стотици пинови за поврзување со

надворешната меморија, додека пак DRAM мемориите употребуваат евтини пакувања кои трошат малку енергија (1 W) и користат неколку десетици пинови. Посебните пакувања за процесорот и меморијата укажуваат дека компјутерските архитекти имаат слобода да го скалираат бројот на мемориски чипови, независно од бројот на процесори.

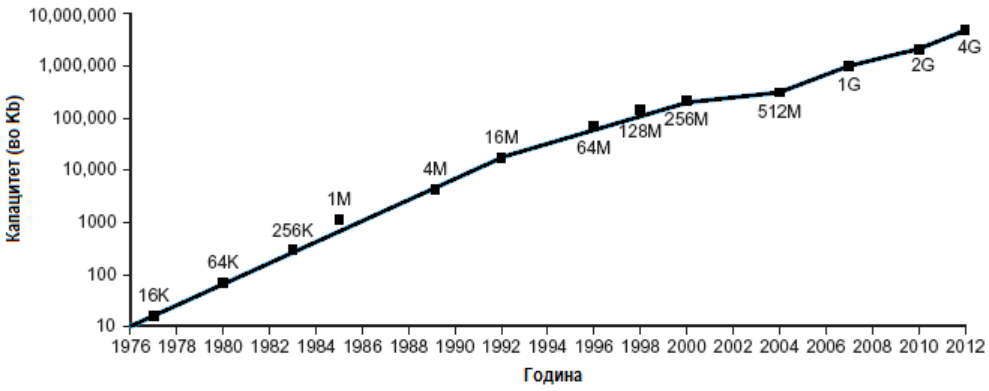
Со оглед на тоа дека процесорот ја има главната улога во стандардните компјутерски системи, кои се базираат на Фон Нојманова архитектура, [8], најголемо внимание во досегашните истражувања е посветено на развој на процесорските архитектури, и тоа во насока на подобрување на внатрешната организација на процесорот и неговото инструкциско множество, како и обезбедување на повисоко ниво на паралелизам. Како резултат на тоа, развиени се повеќе различни процесорски архитектури, вклучувајќи: CISC (Complex Instruction Set Computing), RISC (Reduced Instruction Set Computing), суперскалар, вектор, VLIW и EPIC, [16]. Секоја од нив е предложена со цел да надмине некои проблеми на претходните процесорски архитектури и на тој начин да обезбеди подобрување на перформансите. Во табела 1 се претставени најважните карактеристики на наведените процесорски архитектури, при што се забележува дека заедничко за сите нив е тоа што имплементираат методи за паралелно процесирање (пр. проточност, употреба на повеќе извршни единици, долг инструкциски збор итн.).

Табела 1 Споредба на различни процесорски архитектури.

	Карактеристики				
Архитектура	**Начин на извршување програми**	**Паралелно процесирање**	**Техника за паралелизација**	**Број на операции во еден такт циклус**	**Формат на инструкции**
CISC, [76]	Контролен тек	Да, инструкциско ниво	Секоја инструкција врши повеќе операции	Зависи од комплексноста на инструкцијата	Комплексни операции со променлива должина на
RISC, [76]	Контролен тек	Да, инструкциско ниво	Проточно извршување во неколку фази	Зависи од длабочината на проточната линија (обично 4-5 фази)	Фиксна должина, (обично: 16,32,64)
Супер Скалар, [20]	Контролен тек	Да, инструкциско ниво	Повеќе проточни линии	Зависи од бројот на фази во проточната линија (обично 8-10) и бројот на извршни единици	Фиксна должина, (обично: 16,32,64)

	Карактеристики				
Архитектура	Начин на извршување програми	Паралелно процесирање	Техника за паралелизација	Број на операции во еден такт циклус	Формат на инструкции
Вектор, [19]	Контролен тек	Да, податочно ниво	Иста операција се извршува над различни вектори со фиксна должина	Зависи од бројот на извршни единици и должина на вектор	Фиксна должина (обично: 16,32,64)
VLIW, [21]	Контролен тек	Да, инструкциско ниво	Фиксен број на мини-инструкции во инструкциски збор (имплицитен паралелизам)	Зависи од бројот на извршни единици и бројот на мини-инструкции	Фиксна должина: повеќе мини-инструкции во еден збор
EPIC, [22]	Контролен тек	Да, инструкциско ниво	Променлив, но ограничен број на мини-инструкции во инструкциски збор (експлицитен паралелизам	Зависи од бројот на извршни единици и бројот на мини-инструкции	Променлива должина: повеќе мини-инструкции во еден збор

DRAM технологијата има доминантна улога при имплементирање на главна меморија во компјутерските системи, бидејќи таа се карактеризира со ниска цена и голема густина (помала површина по бит), без разлика на тоа што има 5-10 пати подолго време на пристап од SRAM меморијата. Технолошкиот развој на DRAM индустријата обезбеди зголемување на капацитетот на DRAM меморијата за четири пати на секои три години, односно 60% секоја година, [1]. Овој прогрес е прикажан на сл. 1. Причините за ваквото зголемување доаѓаат од намалувањето на големината на мемориската клетка за фактор од 2,5 и зголемувањето на површината на чипот за фактор од 1,5, на годишно ниво.

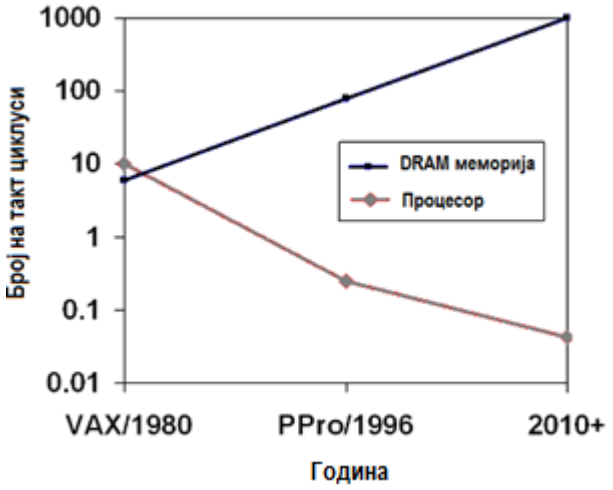


Слика 1 Зголемување на капацитет на DRAM мемориски компоненти, со текот на времето, [1].

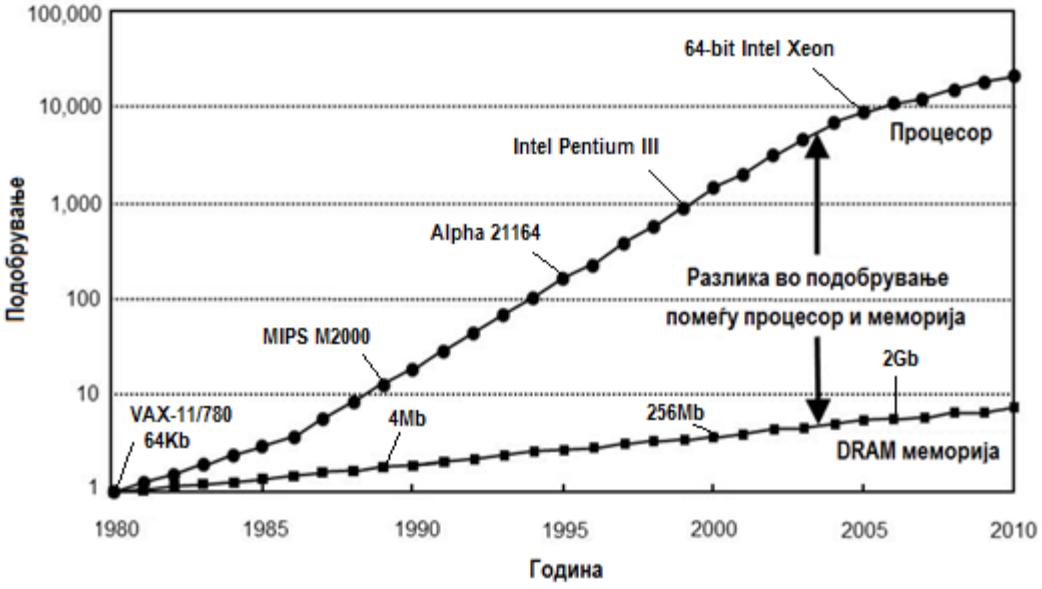
За разлика од прикажаното зголемување на капацитетот на мемориските компоненти, времето на пристап до меморијата се подобри за само 7% на годишно ниво, додека пак процесорските перформанси постигнаа пораст од 60% на годишно ниво, [6]. Овој прогрес резултираше со спори мемориски операции, кои во денешно време побаруваат повеќе десетици процесорски циклуси за да се изврши трансфер на податоци помеѓу процесорот и главната меморија. За да се избегне тесното грло во комуникацијата помеѓу процесорот и меморијата, предложени се повеќе различни пристапи, вклучувајќи: алгоритми за предвидување на гранење, техники за шпекулативно и нередоследно извршување, претходно земање на податоци и инструкции, повеќенитност, употреба на поширок и побрзи мемориски интерфејси и имплементирање на кеш меморија во повеќе нивоа, [18]. Сите овие техники можат да послужат само како привремено решение, но сè уште не можат во целост да го решат проблемот со зголеменото мемориско доцнење. Дури и развојот на некои комплексни процесорски архитектури, како што се суперскаларните, VLIW и EPIC процесори, кои можат да извршуваат неколку милиони инструкции во секунда, не го постигнаа очекуваниот успех, поради мемориските застои кои се јавуваат при извршувањето на спорите мемориски пристапи, [32].

Проблемот на тесно грло во комуникацијата помеѓу процесорот и меморијата е уште познат како мемориска блокада (анг. memory wall). Овој поим за прв пат е воведен од страна на В. Вулф во 1995 година, [5]. За да се истакне значењето на овој проблем, на сл. 2 е прикажано времето кое е потребно за изведување на: инструкција која пристапува до DRAM меморија и инструкција која се извршува во прецесор, во последните неколку децении. На сл. 2 се забележува дека со текот на времето, постојано се зголемува бројот на такт циклуси потребни за извршување на мемориските пристапи, како резултат на зголемената брзина на работа на процесорите, во споредба со брзината на пристап до DRAM меморијата. Претставената разлика исто така може да се забележи на сл. 3, каде се илустрирани: порастот на брзината на работа на процесорите, измерено преку времето потребно за извршување на стандардни SPECint тест програми со цели броеви, за различни генерации на процесори, и ратата на намалување на времето на пристап до DRAM меморијата, измерено преку RAS (Row

access strobe) и CAS (Column access strobe) временски параметри на DRAM мемориски компоненти со различен капацитет, за временски период од три децении, [1]. Резултатите кои се дадени на сл. 3 покажуваат дека времето на пристап до DRAM меморијата се одликува со годишно подобрување од 1,07 пати (два пати на секои 10 години), додека пак процесорите постигнуваат годишно подобрување на перформансите од 1,25 пати до 1986 год., 1,52 пати до 2003 год., и 1,20 пати почнувајќи од 2004 год., [2]. Како резултат на овие трендови, јазот помеѓу годишните подобрувања кај процесор и DRAM меморија расте експоненцијално.



Слика 2 Приказ на број на такт циклуси потребни за извршување на инструкција која пристапува до DRAM меморија и инструкција која се извршува во процесор, за временски период од три децении.

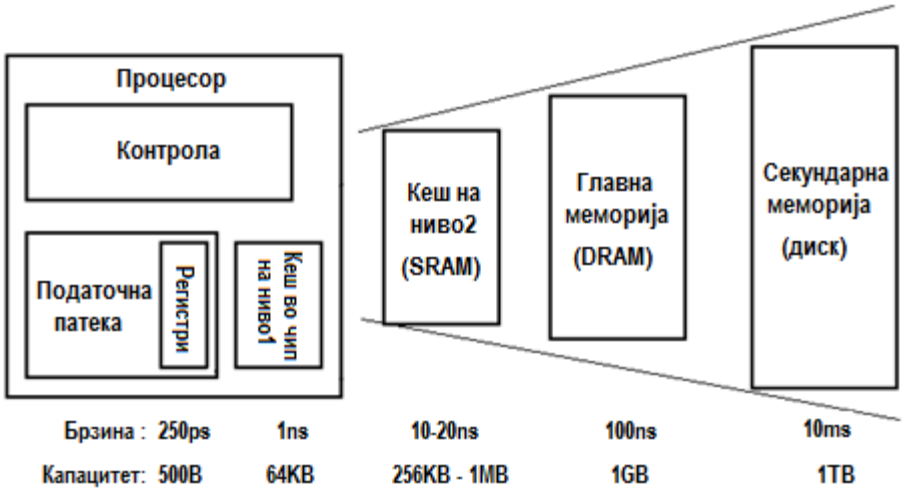


Слика 3 Разлика помеѓу годишно подобрување на брзина на работа на процесор и DRAM меморија, за временски период од три децении.

2.2. Преглед на различни пристапи за намалување на времето на пристап до меморија

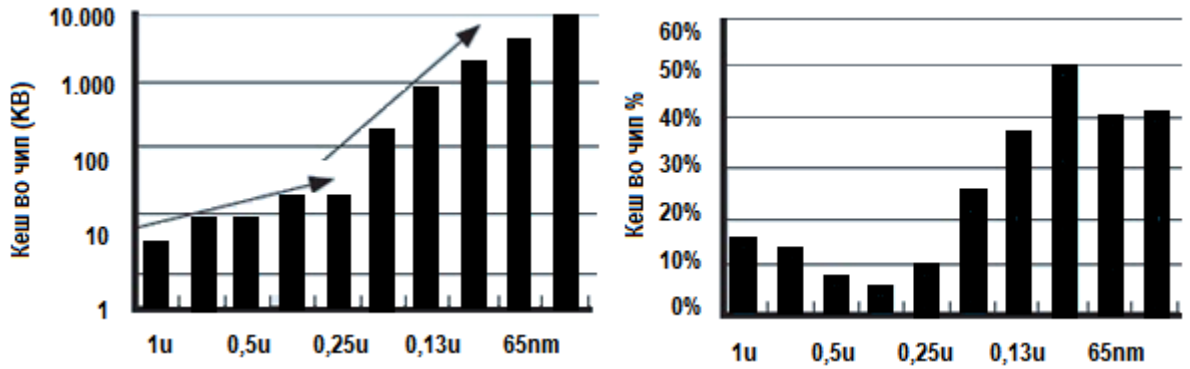
Перформансите на компјутерскиот систем генерално зависат од комуникацијата помеѓу процесорот и главната меморија. Ова однесување се дефинира со параметрите: мемориско доцнење и мемориска пропусност, [7]. Мемориското доцнење е времето помеѓу издавањето на мемориско барање и неговото завршување, додека пак мемориската пропусност е ратата со која што може да се пренесуваат податоци од и во меморискиот систем (изразен со мерка b/s). Во согласност со тоа, компјутерскиот систем може да постигне максимални перформанси, единствено во случај кога мемориското доцнење е приближно нула, а мемориската пропусност е скоро бесконечна, [6]. Ваквите карактеристики опишуваат идеален мемориски систем, кој во реалност не може да се реализира. Со оглед на тоа дека во временскиот период за кој се постигнува дуплирање на мемориската пропусност, мемориското доцнењето се намалува за не повеќе од 1,2 до 1,4 пати, [7], компјутерските архитекти ги насочиле своите истражувања кон развој на различни пристапи за подобрување на мемориското доцнење. Според тоа, воведени се две основни групи на техники за: редуцирање на мемориското доцнење и прекривање на мемориското доцнење, [18].

Групата за редуцирање на мемориското доцнење вклучува различни техники кои се наменети да го намалат времето помеѓу издавањето на мемориско барање и добивање на побараниот податок како одговор. Најпознат метод од оваа група е креирањето на повеќе-нивовска мемориска хиерархија, при што внатре во процесорот и близу до него се додаваат неколку нивоа брза кеш меморија, со цел да се избегнат спорите пристапи надвор од чипот до главната меморија. Хиерархискиот пристап на организирање на меморијата се базира на принципите на просторна и временска локалност, [1], [2], како и на правилото "помалото е побрзо", кое што укажува дека мемориите со помал капацитет се вообичаено побрзи, а воедно и поскапи од оние со поголем капацитет. На сл. 4, преземена од [2], е прикажана мемориската организација во стандарден компјутер, составена од: мали и брзи регистри, кеш меморија во повеќе нивоа, главна меморија (најчесто имплементирана како DRAM) и секундарна меморија,



Слика 4 Типична организација на мемориска хиерархија во компјутерски систем, [2].

Со оглед на тоа дека брзата меморија е скапа (користи SRAM технологија), компјутерските системи во своите почетоци користеле мало количество на кеш меморија. Сепак, транзицијата од едно-процесорски кон повеќе-процесорски и повеќе-јадрени системи воведе многу големи барања за меморискиот систем и со тоа предизвика зголемување на капацитетот и површината на кеш меморија во чипот. Овој тренд е прикажан на сл. 5, каде е илустриран порастот на капацитетот и површината на кеш меморијата во чип за Интеловите процесори имплементирани во повеќе различни технологии, [9]. Според резултатите се забележува дека Интеловите процесори изработени во 65nm технологија употребуваат дури 40% од површината на процесорскиот чип, за да имплементираат до 10MB кеш меморија во чипот.



а) Капацитет на кеш во чип (KB). б) Површина на кеш во чип (%).

Слика 5 Еволуција на кеш меморија во чип за Интелови микропроцесори, [9].

Со реализирање на кеш меморија во процесорскиот чип, се намалува бројот на пристапи надвор од чипот и се намалува просечното време кое е потребно за да се опслужи некое мемориско барање, [1]. И покрај овие предности, сепак кеш меморијата воведува непредвидливост во времето на извршување на програмите, како резултат на промашувањата кои можат да се случат при мемориските пристапи (читање или запишување). Ова се потврдува со релациите 1, 2 и 3, во кои се претставени изразите за пресметување на просечно време на пристап до меморија, просечен број на такт циклуси за мемориски застои по инструкција и време на извршување на програмите, соодветно, [2]. Според овие изрази може да се заклучи дека со зголемување на бројот на промашувања во кешот, се зголемува просечниот број на такт циклуси за мемориски застои по инструкција, што пак влијае да се зголеми времето на извршување на програмите. Всушност, бројот на процесорски циклуси кои ќе бидат потрошени за да се опслужи промашување во кешот се определува со параметарот: казна која се плаќа за промашување.

$$\textit{Просечно време на пристап до меморија} = \tag{1}$$
$$= \textit{Време на погодок во кеш меморија} +$$
$$\textit{Рата на промашување во кеш меморија} * \textit{Казна која се плаќа за промашување}$$

$$\textit{Просечен број на такт циклуси за мемориски застои по инструкција} = \tag{2}$$
$$= \textit{Просечен број на мемориски пристапи по инструкција} *$$
$$\textit{Рата на промашување во кеш меморија} * \textit{Казна која се плаќа за промашување}$$

$$\textit{Време на извршување} = \tag{3}$$
$$= \textit{Број на инструкции} * (\textit{Број на такт циклуси по инструкција} +$$
$$\textit{Просечен број на такт циклуси за мемориски застои по инструкција}) * \textit{Такт период}$$

Со цел да се обезбеди намалување на ратата на промашување во кешот се употребуваат различни техники, меѓу кои се: зголемување на димензиите на кешот и/или неговите блокови, примена на поголема асоцијативност или употреба на жртвен и/или псевдо-асоцијативен кеш, [6]. Дополнително за да се намали казната која се плаќа за промашување во кешот, се користат следните техники: приоритет за читање наместо запишување при промашување, брзо

рестартирање и предимство на критичен збор при промашување, замена на подблокови или употреба на повеќе нивоа во кешот. Во тој контекст, на сл. 6 е прикажано како влијае големината на блокот во кешот врз казната која се плаќа за промашување, ратата на промашување и просечното време на пристап до меморија, [2]. Според графиците кои се претставени на сл. 6 може да се каже дека генерално со зголемување на големината на блоковите во кешот се зголемува просторната локалност, но исто така се зголемува и казната која се плаќа при промашување. Всушност, доколку блоковите во кешот станат многу големи во споредба со големината на кешот, тогаш ратата на промашување ќе се зголеми.

Слика 6 Влијание на големина на блок во кеш врз параметрите: казна која се плаќа за промашување, рата на промашување и средно време на пристап до меморија, [2].

Според претходните согледувања може да се каже дека повеќе нивовската кеш меморија овозможува да се намали просечното време на пристап до меморија, но по цена на дополнителна хардверска комплексност, зголемена потрошувачка на енергија, редундантност во системот и непредвидливост во времетраењето на програмите. Имено, кеш меморијата ги подобрува перформансите само за оние апликации кои се карактеризираат со голема локалност. Тоа значи дека без доволна локалност, апликациите како што се: мултимедиско процесирање, пребарување на текст и други слични податочно-интензивни пресметки нема да се извршуваат ефикасно, [18].

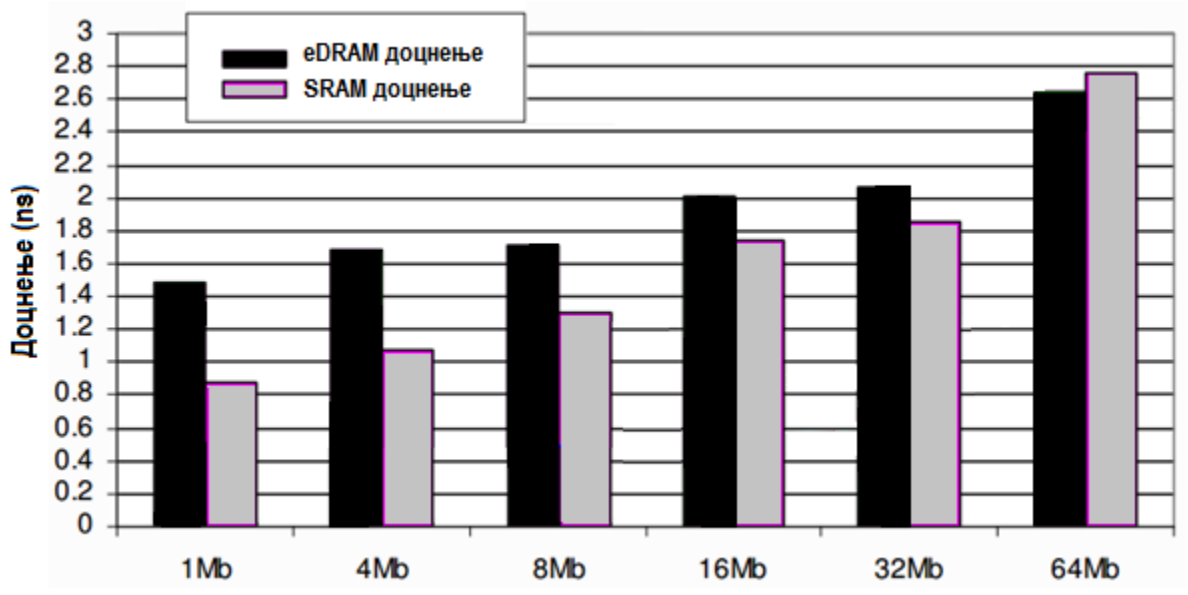
Дополнително намалување на мемориското доцнење може да се постигне со комбинирање на голема кеш меморијата и некаков пристап на шпекулативно предвидување на гранење или нередоследно извршување, [17]. Слично како кешот и овие техники ја зголемуваат површината на чипот и воведуваат дополнителна комплексност на хардверско и софтверско ниво, [18]. Дури и некои

посложени пристапи на сметање, како што се: вектор, суперскалар, VLIW и EPIC, [19] - [22], се соочуваат со слаба искористеност на расположливите ресурси, комплексност при имплементација, лоша компајлерска поддршка и неразвиен модел на програмирање. Другите начини за редуцирање на мемориското доцнење предлагаат конкретни иновации во архитектурата на DRAM меморијата (асиметричен DRAM, RLDRAM, FCRAM, SALP, Enhanced DRAM, Virtual Channel DRAM, TL-DRAM, хибридна мемориска коцка или embedded DRAM), [24] - [32], или комплетна замена на DRAM технологијата со нова, врз база на концепти како што се: мемристор, MRAM или FeRAM, [4].

Групата за толерирање на мемориското доцнење вклучува различни техники кои го прикриваат мемориското доцнење на тој начин што му овозможуваат на процесорот да извршува други операции додека се опслужува некое мемориско барање, [17]. Една од најпознатите техники од оваа група е повеќенитноста, [2], која овозможува промена на контекстот на извршување во процесорот со стартување на нова нитка, во случај кога некоја мемориска инструкција предизвикала промашување во кеш меморијата. Од друга страна, неблокирачката кеш меморија, [14], овозможува сервисирање на други барања во кешот, во периодот додека се чека опслужување на некое промашување. Овој тип на кеш меморија очигледно воведува потреба од дефинирање на дополнителен хардвер, кој ќе ја следи состојбата на меморискиte барања, кои сè уште не се опслужени. Друг метод за толерирање на доцнењето е примена на техники за земање на блокови со податоци или инструкции во кеш меморијата, пред тие да бидат побарани (анг. prefetch), [1]. Овој механизам на претходно земање може да се имплементира хардверски или со компајлерска поддршка. Поагресивна форма на техниката на претходно земање е шпекулативното вчитување, [17], кое што врши предвремено вчитување на податоци во кешот, врз основа на некои шеми за предвидување. Генерално може да се каже дека со употреба на претходно наведените техники се овозможува прикривање на мемориското доцнење, но сепак се зголемува меморискиот сообраќај, како резултат на зголемената рата на инструкции и барања за операнди. Во таков случај, ограничената пропусност на меморискиот интерфејс предизвикува дополнително доцнење и повторно тесно грло во системот.

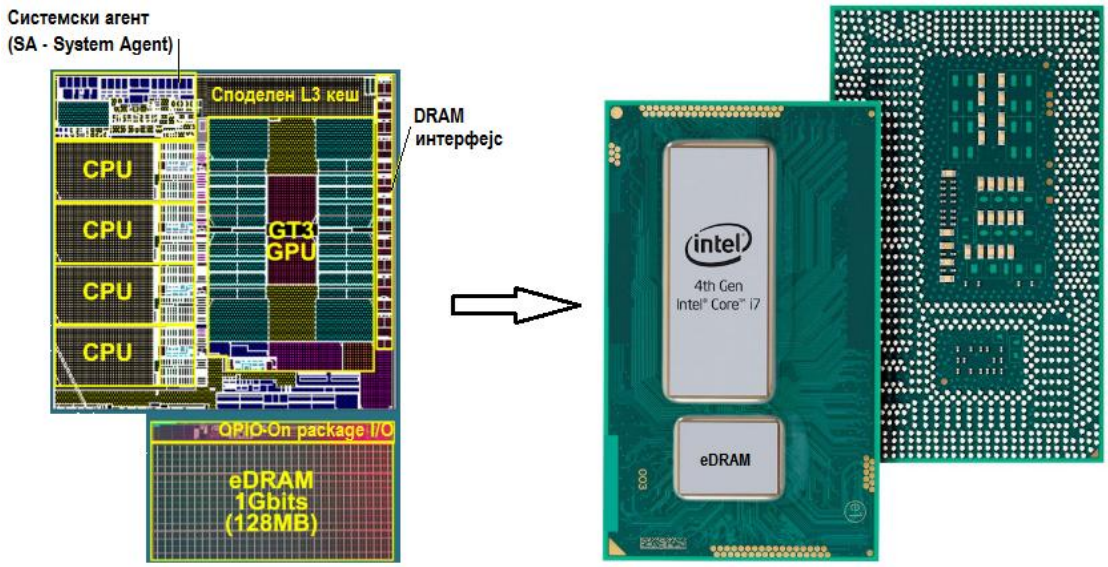
2.3. Примена на технологија на вграден (embedded) eDRAM

Технологијата на вграден eDRAM се користи за имплементирање на DRAM меморија на истиот чип со процесор или ASIC (Application Specific Integrated Circuit) коло. Вградениот eDRAM се карактеризира со поголема цена по бит во споредба со еквивалентен DRAM чип кој се користи како надворешна меморија, но од друга страна тој постигнува поголеми брзини на работа, и поголема пропусност со употреба на широки внатрешни магистрали. Во споредба со SRAM, вградениот eDRAM е со три пати поголема густина и 0,2 пати подобра енергетска ефикасност, и воедно зафаќа помала површина, што пак овозможува да се намали неговата цена на производство. Генерално, со интегрирање на eDRAM во чипот, комуникацијата помеѓу вградената меморија и процесорот се одвива преку широка внатрешна магистрала (без посебен В/И интерфејс) со значително намалено доцнење и зголемена пропусност, [30]. Всушност, на сл. 7 е направена споредба на доцнењето на кеш мемории со различен капацитет, имплементирани во процесорскиот чип како SRAM или вграден eDRAM. Големината на анализираните кеш мемории и нивното доцнење се пресметани врз основа на постоечки макро (Intellectual property) елементи на вграден eDRAM и SRAM, составени од 1Mb градбени блокови, реализирани во 45nm SOI CMOS (Silicon on insulator Complementary Metal Oxide Semiconductor) технологија, [30].



Слика 7 Споредба на доцнењето на кеш мемории со различен капацитет, имплементирани во процесорскиот чип како вграден DRAM и SRAM во 45nm технологија, [30].

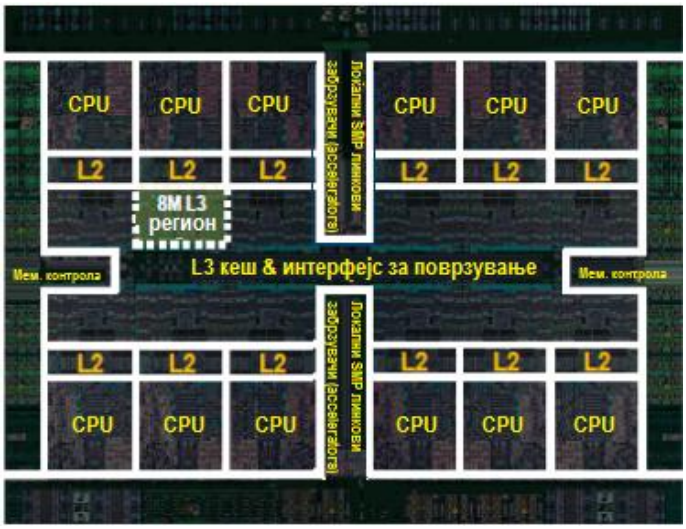
И покрај тоа што перформансите на вградениот eDRAM се имаат значително подобрено во последните неколку години, резултатите дадени на сл. 7 покажуваат дека SRAM кеш со капацитетот од 1Mb сепак постигнува два пати помало доцнење од eDRAM кеш со истиот капацитет. Всушност, со порастот на капацитетот на кеш меморијата над 16Mb се зголемуваат доцнењата на жиците, па доцнењето на вградениот DRAM кеш постојано се приближува кон доцнењето на SRAM кешот, постигнувајќи дури и подобрување за кеш со големина од 64Mb. Како резултат на тоа, вградениот eDRAM најчесто се користи за имплементирање на повисоки нивоа (3 или 4) на кеш меморија во чипот, со капацитет од неколку десетици мегабајти. Според тоа, во продолжение е претставена примената на eDRAM технологија во популарните Intel Core и IBM Power8 (од International Business Machines - IBM компанијата) процесори. Имено, на сл. 8 е прикажана организацијата на Intel Core процесор од 4^{та} генерација, кој имплементира Intel Hashwell микроархитектура, [10], а на сл. 9 е прикажана организацијата на IBM Power8 процесор, [77], претставен во 2013^{та} година како подобрена верзија на IBM Power7 процесорот, [78].



Слика 8 Организација на Intel Core процесорски чип со Intel Hashwell микроархитектура.

На сл. 8 може да се забележи дека Intel Hashwell микроархитектурата обединува 4 процесорски јадра, графичка единица, 64-битен В/И интерфејс, 8MB SRAM споделен кеш на ниво 3 и 128MB вграден eDRAM споделен кеш на ниво 4. Во таков повеќе-јадрен дизајн на чип, секое процесорско јадро располага со 2MB SRAM кеш на ниво 3, а вградениот eDRAM се споделува помеѓу графичката еди-

ница и процесорските јадра, однесувајќи се како жртвен кеш за кешот на ниво3. За да се имплементира Intel Core процесорски чип со Intel Hashwell микроархитектура (пр. Intel Core i3, i5 или i7) се користи 22nm Tri-gate CMOS технологија, [10]. При реализацијата, процесорот и вградениот eDRAM се интегрираат на заедничкиот чип, но во посебни пакувања, како што е прикажано на сл. 8.



Слика 9 Организација на IBM Power8 процесорски чип, [77].

На сл. 9 е прикажана организацијата на IBM Power8 процесорски чип, имплементиран во 22nm технологија. Овој чип вклучува 12 процесорски јадра, при што секое јадро извршува најмногу 8 нитки и располага со 96KB, 512KB и 8MB кеш меморија на ниво 1, 2 и 3, соодветно, [77]. Овие карактеристики укажуваат дека Power8 процесорот постигнал голем напредок во однос на неговата претходната верзија имплементирана во 32nm технологија - Power7 процесорот, кој вклучува вкупно 8 процесорски јадра, при што секое јадро извршува најмногу 4 нитки и користи 64KB, 256KB и 4MB кеш меморија на ниво 1, 2 и 3, соодветно, [78]. И покрај наведените разлики, генерално може да заклучи дека двете верзии на IBM Power процесори вклучуваат десетици мегабајти кеш меморија во чипот. Всушност, вкупните количини на вградена DRAM меморија, која се имплементира како споделен кеш на ниво 3 во чипот изнесува 32MB и 96MB за Power7 и Power8 процесорите, соодветно. Дополнително, Power8 процесорите ја враќаат употребата на кеш меморија на ниво 4, користејќи Centaur придружни чипови надвор од процесорот, [77]. Овие Centaur придружни чипови можат да вградат DRAM меморија со капацитет од најмногу 128MB, како L4 кеш.

3. ПРЕГЛЕД НА МЕМОРИСКО-ЦЕНТРИЧНИ СИСТЕМИ

Согледувајќи ја комплексната врска помеѓу мемориското доцнење и мемориската пропусност, компјутерските архитекти спровеле различни истражувања со кои се обидуваат да најдат начини за дизајнирање на "паметни мемории", кои ќе обезбедат истовремено намалување на мемориското доцнење и зголемување на мемориската пропусност, [32]. Овие истражувања резултираа со неколку предлози за реорганизирање на класичната повеќе-нивовска мемориска хиерархија со цел да се овозможи нестандарден побрз пристап до главната меморија. На пример, авторот на [33] предлага безрегистарски процесор (PERL), кој ги извршува сите операции директно со кеш меморијата во и надвор од чипот, без да користи експлицитни регистри за општа намена. Дополнително, авторот на [34] предлага употреба на бележник (Scratchpad) меморија, како мала софтверски управувана меморија во чипот, со време на пристап од еден такт циклус, [35].

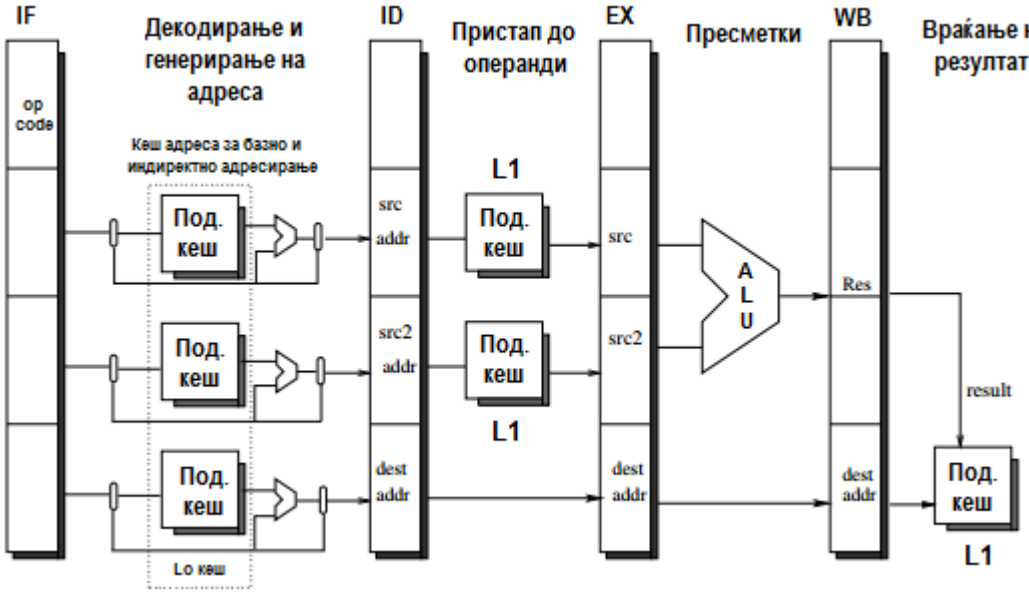
За разлика од стандардниот модел на процесорско-центрично сметање (Фон Најманов модел), [8], некои научници предложиле алтернативни пристапи на мемориско-центрично сметање со интегрирање или сместување на меморија блиску до процесорот. Овие истражувања резултираа со неколку PIM чипови кои интегрираат процесирање и меморија на заеднички чип (пресметковен RAM, [36], SUN PIM чип, [37], Mitsubishi 32R/D чип, [38], Terasys PIM чип, [39], Diva PIM чип, [40], [41], интелигентен RAM - IRAM, [42], [43], паралелен процесирачки RAM, [44] и DataScalar систем, [45]) и интелигентни мемориски системи, како што е моделот на активни страници, [46], кој доделува реконфигурабилни логички блокови на секоја виртуелна страница во меморијата. Овие споени мемориско-пресметковни чипови вклучуваат меморија во чипот, која овозможува да се постигне голема внатрешна мемориска пропусност, мало мемориско доцнење и голема енергетска ефикасност, што ги прави особено погодни за извршување на пресметки кои побаруваат голема податочна пропусност, како што се: брза Фуриева трансформација, мултимедиско процесирање и мрежно процесирање, [32].

Во следните поглавја е даден детален опис на споменатите мемориско-центрични системи, при што дискутирани се нивните: архитектура, организација, предности, недостатоци и подрачје на примена.

3.1. PERL (Performance Enhanced Register-Less) процесор без регистри

PERL претставува безрегистарски процесор за пресметување со високи перформанси кој работи директно со кеш меморијата во чипот, без да употребува регистри за општа намена, и на тој начин го модифицира стандардниот модел на пристап до главната меморија, [33]. Оваа, така наречена меморија-до-меморија архитектура му овозможува на процесорот директно да ги адресира податоците, без при тоа да користи експлицитен регистарски именски простор. Како резултат на тоа, неговите инструкциите се со големина од 128 бита и ги содржат 32-битните мемориски адреси на двата влезни операнда и резултатот. Ваквиот директен пристап до операндите укажува дека инструкциското множество на PERL не треба да вклучува експлицитни load и store инструкции.

Процесорот PERL претставува модификација на 32-битен RISC-базиран DLX (Deluxe) процесор со load-store архитектура. Основната разлика помеѓу овие два процесора е тоа што процесорот PERL работи директно со кеш меморијата во проточна податочна патека со 5 фази: земање на инструкција; декодирање на инструкција и генерирање на адреси на операнди; пристап до операнди во кешот; извршување на пресметки; и запишување на резултат во кешот. На сл. 10 е прикажана проточната податочна патека на процесорот PERL.



Слика 10 Проточна податочна патека на PERL процесор, [33].

На сл. 10 се забележува дека процесорот PERL имплементира кеш меморија на ниво 0, во фазата за декодирање и генерирање на адреси на операнди; и кеш меморија на ниво 1, во фазите за пристап до операнди и запишување на резултат во кешот. Всушност, кешот на ниво 0 е изграден од брзи регистри мапирани во мемориски локации, кои се користат за да обезбедат поддршка на базно и индиректно адресирање на операнди. Откако ќе се определи адресата на операндите (со помош на кешот на ниво 0), до податоците може да се пристапи преку кешот на ниво 1 и останатата мемориска хиерархија. Запишувањето на резултатот во кеш меморијата се прави преку посебна фаза наменета за враќање на резултат (writeback).

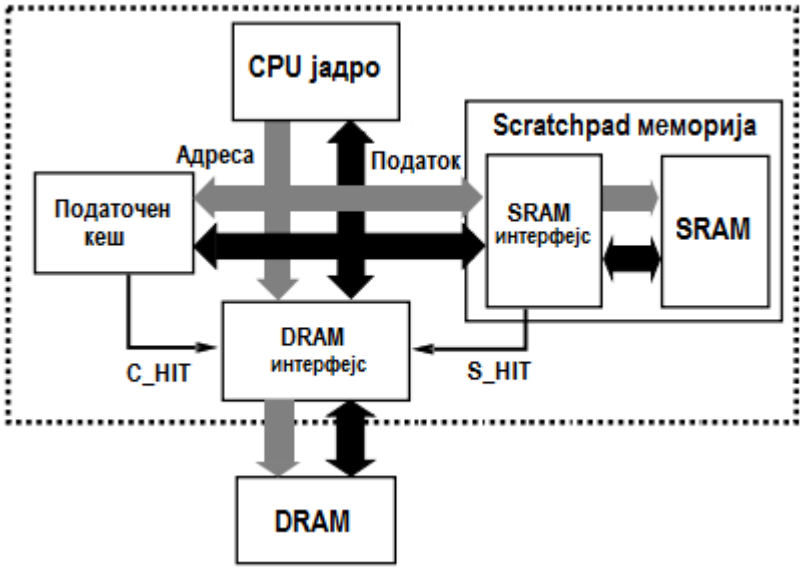
Безрегистарското работење на процесорот PERL (без употреба на експлицитни load и store инструкции) го намалува бројот на инструкции во програмите, во споредба со иницијалниот RISC-базиран DLX процесор. Како резултат на тоа, PERL постигнува слични или подобри перформанси од DLX процесорот, при извршување на различни тест програми, [79]. Дополнително, PERL овозможува развој на поефикасна компајлерска технологија, без да го вклучува комплексниот процес на алокација на регистри, кој е неопходен во стандардните load-store процесори кои работат со регистри.

3.2. Бележник (Scratchpad) меморија

Ефикасната искористеност на меморијата во чипот е многу важна за вградливите процесорски системи, па како резултат на тоа авторот на [34] предлага креирање на мала и брза компајлерски-управувана бележник SRAM меморија внатре во процесорскиот чип, [34]. Бележник меморијата се мапира во мемориски адресен простор, кој е независен од адресниот простор на надворешната главна меморија, но е поврзан на истата адресна и податочна магистрала. Оваа мала и брза меморија вообичаено се употребува за складирање на меѓу-резултати и најчесто користени податоци. Основната разлика помеѓу софтверски управуваната бележник меморија и хардверски управуваната кеш меморија е тоа што бележник меморијата гарантира време на пристап од еден такт циклус, додека пак време на пристап до кешот зависи од тоа дали настанал

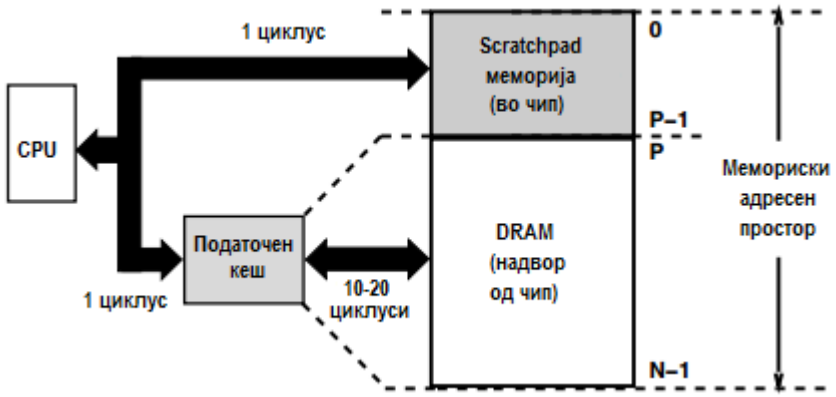
погодок (1 циклус) или промашување (10-20 такт циклуси) во кешот. Со оглед на тоа дека бележник меморија воведува повеќе компликации од софтверски аспект, таа побарува развивање на комплексни компајлерски методи за ефикасно алоцирање на податоци. Дополнително, тука може да се додадат и барањата за ниска потрошувачка на енергија при дизајнирање на вградливите системи. Како резултат на тоа, во последно време бележник меморија се реализира со употреба на STTRAM (Spin Transfer Torque RAM) технологија, [35].

На сл. 11 е прикажан блок дијаграм на типичен вградлив процесорски систем, каде со испрекината линија се ограничени компонентите кои што се имплементирани на заеднички чип, кој преку посебен интерфејс е поврзан со надворешна DRAM меморија. На блок дијаграмот се забележува дека процесорското јадро користи внатрешна адресна и податочна магистрала за да се поврзе со податочниот кеш, бележник меморијата и DRAM меморискиот интерфејс. Всушност, барањата од процесорот за пристап до меморија се извршуваат на тој начин што секој погодок во кеш или бележник меморијата се индицира до DRAM интерфејсот со поставување на C_HIT или S_HIT сигналите, соодветно. Во случај кога обраќањата и до кешот и до бележник меморијата резултираат со промашување, тогаш се врши трансфер на блок на податоци помеѓу кешот и надворешната DRAM меморија, преку DRAM интерфејсот.



Слика 11 Блок дијаграм на вградлив процесорски систем кој користи Scratchpad меморија, [34].

На сл. 12 е прикажано како се дели мемориският адресен простор (со големина од N зборови), помеѓу бележник меморијата во чипот и надворешната главната меморија. Според сл. 12 се забележува дека мемориските адреси од 0 до P-1 се мапираат во бележник меморијата, додека пак мемориските адреси од P до N-1 се мапираат во DRAM меморијата. Тоа значи дека S_HIT сигналот се поставува од процесорот во случај кога тој пристапува до било која мемориска локација во опсегот од 0 до P-1. На сличен начин, C_HIT сигналот се поставува од процесорот во случај кога тој пристапува до било која мемориска локација во опсегот од P до N-1. Секој погодок при пристап до било која мемориска локација во опсегот од 0 до N-1 трае еден такт циклус, додека пак промашувањата се реализираат преку размена на блокови помеѓу надворешната DRAM меморија и кешот, резултирајќи со доцнење од 10 до 20 такт циклуси.



Слика 12 Поделба на мемориският адресен простор, помеѓу Scratchpad SRAM меморија во чип и DRAM меморија надвор од чип, [34].

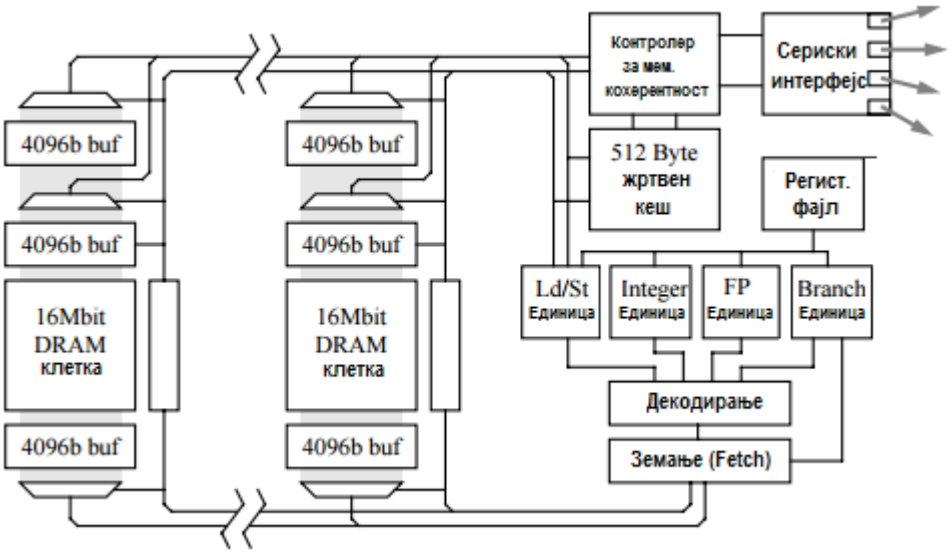
3.3. Процесирање во меморија

Технологијата на интегрирање на процесирање во меморија (PIM) подразбира комбинирање на процесирачки елементи и меморија во еден заеднички CMOS чип. Чипот со процесирање во меморија може да се класифицира во зависност од неговата улога, како: главен процесор(и) во системот, процесор со специфична намена, копроцесор или интелигентен мемориски систем, [54]. На пример, IRAM чипот, [43], е имплементиран како векторски копроцесор за MIPS процесор за општа намена. Интегрираната меморија во чипот може да биде реализирана како SRAM или вграден DRAM, при што до нејзе вообичаено се

пристапува преку кеш меморијата на процесорот. Со оглед на тоа дека процесорот и меморијата се физички блиску, чипот со процесирање во меморија постигнува поголема мемориска пропусност, како и помало мемориско доцнење и потрошувачка на енергија, во споредба со денешните стандардни мемориски чипови, како и кеш мемории во повеќе-процесорските системи, [32]. И покрај наведените предности, пристапот на процесирање во меморија се соочува со повеќе критични точки, како што се: ограниченост на меморија во чипот; потреба од редефинирање на процесот на производство (процесорите и меморијата тековно користат комплетно различни процеси); и потешкотии со претставување и прифаќање на нов дизајн на чип на пазарот. Според тоа, најповолно би било овој пристап да се употреби за вградување на 10-тици MB DRAM во процесори наменети за вградливи системи, портабилни уреди или специфични апликации.

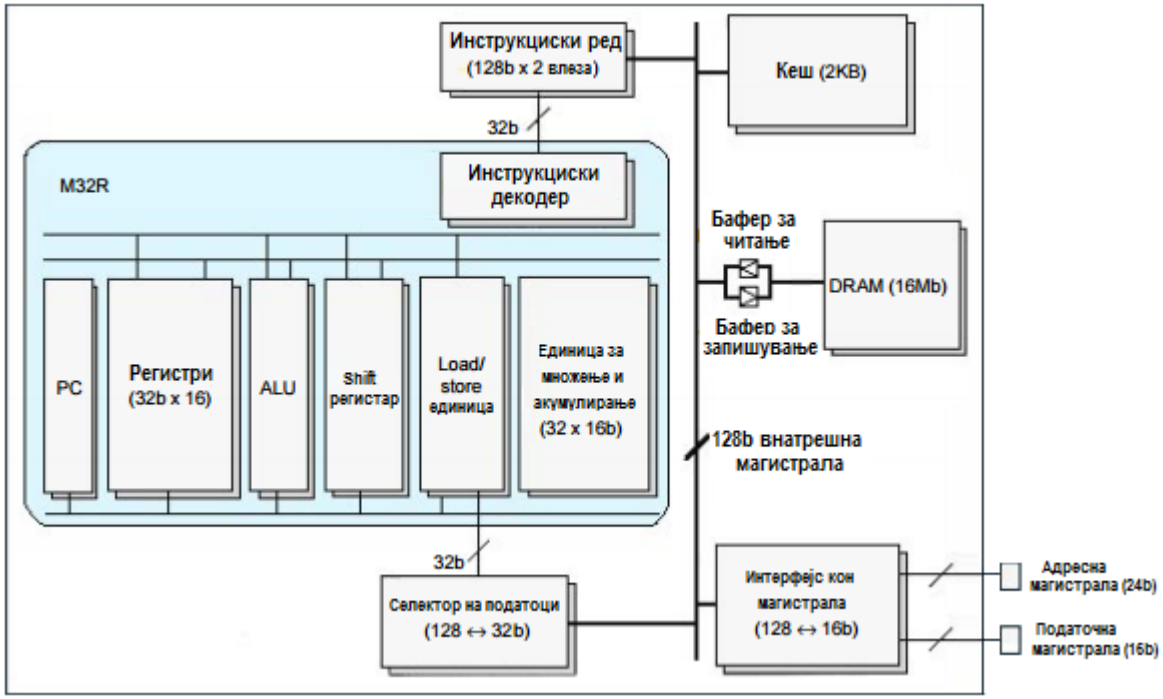
3.3.1. Sun PIM и Mitsubishi M32R/D чипови

Два предлога за интегрирање на RISC процесор и DRAM меморија во заеднички чип, предложени од Sun Microsystems и Mitsubishi Electric Corporation, а познати како Sun PIM и Mitsubishi M32R/D чипови, се претставени во [37] и [38], соодветно. Блок дијаграмите на двата чипа се прикажани на сл. 13 и сл. 14, каде што може да се забележи дека Sun PIM чипот вклучува 256Mb меморија распределена во 16 банки, а пак M32R/D чипот располага со 16Mb вграден DRAM.



Слика 13 Блок дијаграм на SUN PIM чип, [37].

Како што е прикажано на сл. 13, Sun PIM чипот вклучува проточен RISC процесор со 5 фази, (сличен на R4300i или microSparc-II), кој комуницира со меморијата преку две 64-битни магистрали, засебно наменети за пренос на податоци и инструкции. Овие магистрали работат синхроно со процесорскиот такт од 200MHz, при што секоја обезбедува мемориска пропусност од 1,6 GB/s. Sun PIM чипот вклучува 16 DRAM банки со капацитет од 16Mb и времето на пристап од 30ns, односно 6 такт циклуси. Секоја мемориска банка може да пренесува 4Kb податоци помеѓу полето од сензитивни засилувачи на DRAM клетката и трите 512-бајтни (4096b) бафери. Всушност, овие три 512-бајтни бафери го претставуваат податочниот и инструкцискиот кеш на процесорот. На тој начин се овозможува пренос на цел блок на кешот при еден DRAM пристап, што пак влијае да се подобрат перформансите на кешот. Податочниот кеш (16KB) користи асоцијативно мапирање со две групи и се формира од два бафера од секоја банка, односно 32 512-бајтни блокови, распределени во 16 мемориски банки. Останатите 16 бафера, формираат директно мапиран инструкциски кеш со 512-бајтни блокови. Дополнително, Sun PIM чипот вклучува и целосно асоцијативен кеш со 16 32-бајтни блокови кој служи како жртвен кеш за податочниот кеш. Резултатите дадени во [37] укажуваат дека Sun PIM чипот може да се примени за општа намена и при тоа може да постигне слични перформанси како суперскаларен процесор.



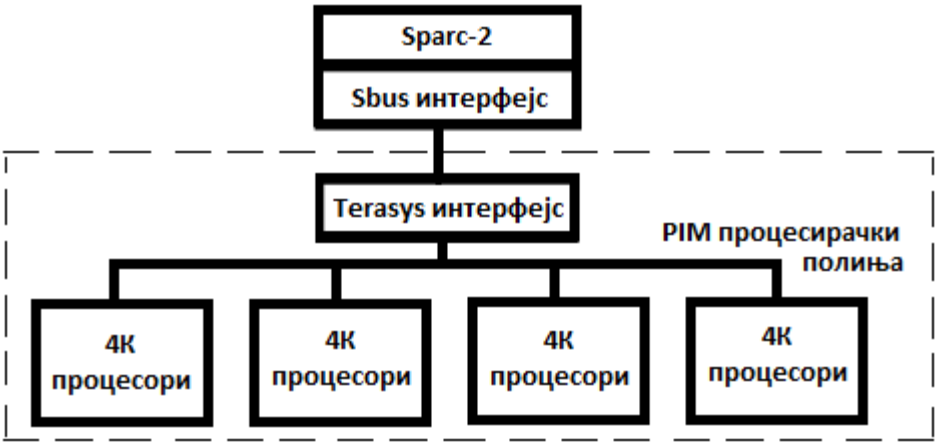
Слика 14 Блок дијаграм на Mitsubishi M32R/D чип, [38].

На сл. 14 е прикажан блок дијаграм на Mitsubishi M32R/D чип кој вклучува: едноставно RISC процесорско јадро, кое работи проточно во 5 фази; инструкциски ред со два 128-битни влеза; 16Mb DRAM и 2KB SRAM кеш, кои се организирани во 4 банки; 128-битна внатрешна магистрала за поврзување на процесорот со меморијата; 128/32b селектор на податоци кој го прилагодува трансферот помеѓу 128-битната магистрала и 32 битното процесорско јадро; и контролор за В/И. Mitsubishi M32R/D чипот работи со кеш, кој обезбедува пристап до вградениот DRAM за време од 1 такт циклус при погодок, односно 5 такт циклуси при промашување. Овој директно мапиран кеш може да функционира во два режима: M32R/D чипот ги кешира податоците и инструкциите од вградениот DRAM кој работи како главна меморија; или M32R/D чипот ги кешира податоците од вградениот DRAM, а инструкциите од надворешниот ROM (Read Only Memory). Без разлика на тоа, заклучоците дадени во [38] укажуваат дека Mitsubishi M32R/D чипот постигнува подобрени перформанси, ниска цена и намалена потрошувачка на енергија, што го прави особено погоден за примена во вградливите системи.

3.3.2. Terasys и Diva системи

Покомплицирани чипови, како што се Terasys и Diva (Data IntensiVe Architecture), се дадени во [39] и [40]. Овие чипови се изградени од стандарден процесор за општа намена и полиња од мемориски копроцесори (PIM чипови), како замена за стандардна DRAM меморија. PIM чиповите вклучуваат процесирачки елементи, кои се сместени директно на излезот од меморијата, со цел да овозможат максимално искористување на мемориската пропусност, при извршување на SIMD (Single Instruction Multiple Data) паралелни пресметки со податоци.

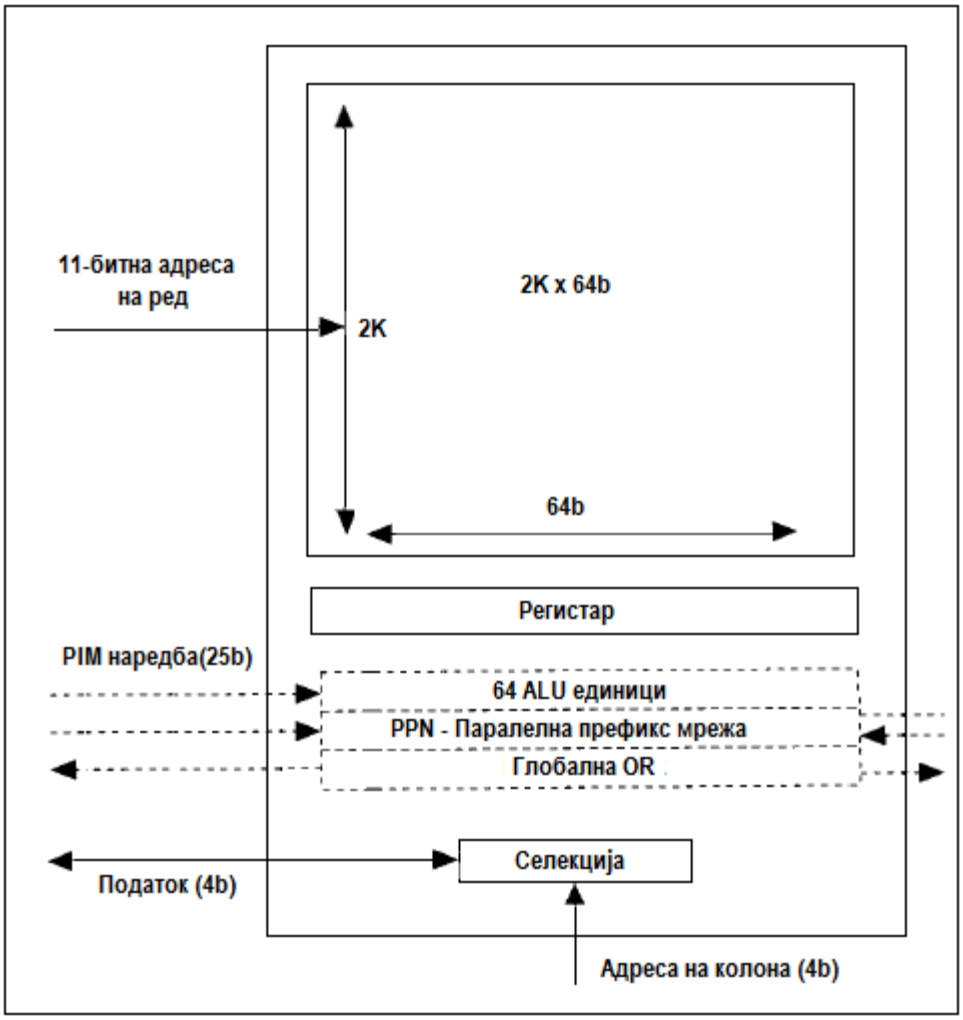
На сл. 15 е прикажан блок дијаграм на Terasys систем, кој е составен од SUN Sparc2 процесор, Sbus интерфејс, Teresys интерфејс и едно или повеќе PIM процесирачки полиња. Секое PIM процесирачко поле содржи 8 банки на PIM меморија, а секоја банка содржи 8 PIM чипови. Со оглед на тоа дека секој PIM чип содржи 64 процесирачки елементи, вкупниот број на процесирачки елементи во едно процесирачко поле изнесува 4096, односно 4K. Terasys системот може да вклучува најмногу 8 процесирачки полиња, па според тоа максималниот број на PIM процесирачки елементи во системот изнесува 32768.



Слика 15 Блок дијаграм на Terasys систем, [39].

Извршувањето на програмите во Terasys системот се одвива на тој начин што стандардните секвенцијални инструкции се извршуваат од страна на Sparc-2 процесорот, а мемориските (load и store) операции и паралелните SIMD операции над множества од податоци се проследуваат преку Terasys интерфејсот до PIM процесирачките полиња. Всушност, Terasys интерфејсот ги користи позначајните битови од адресата на податокот за да определи дали операцијата е наменета за load/store на податок во стандардната меморија или пак треба да извршува податочно-паралелни пресметки во PIM полињата. Во вториот случај се пристапува до табела со микрокодови за да се избере инструкцијата која треба да се изврши во PIM процесирачките полиња.

На сл. 16 е прикажана архитектурата на PIM чип во Terasys систем, кој интегрира 2Kx64b SRAM меморија и 64 процесирачки елементи, [39]. PIM чипот може да работи во стандарден мемориски режим или PIM режим. Всушност, на сл. 16 со полна линија е означено како изгледа чипот кога се употребува како стандардна меморија, а со испрекината линија е претставена логиката која се користи за PIM процесирање. Во стандарден мемориски режим, PIM чипот е конфигуриран како 32Kx4b SRAM, со 11-битна адреса за ред и 4-битна адреса за колона. Селектираниот ред избран со адресата за ред, се сместува во 64-битен регистар и потоа од него во зависност од адресата за колона се избира 4-битниот податок. Во PIM режим, чипот се активира со помош 25-битна командна, проследена од Terasys интерфејсот. При PIM процесирање, 64-те процесирачки елементи извршуваат паралелни операции (пресметки, рутирање и маскирање) над 1-битни елементи, прочитани од 64-битниот селектиран ред во меморијата.

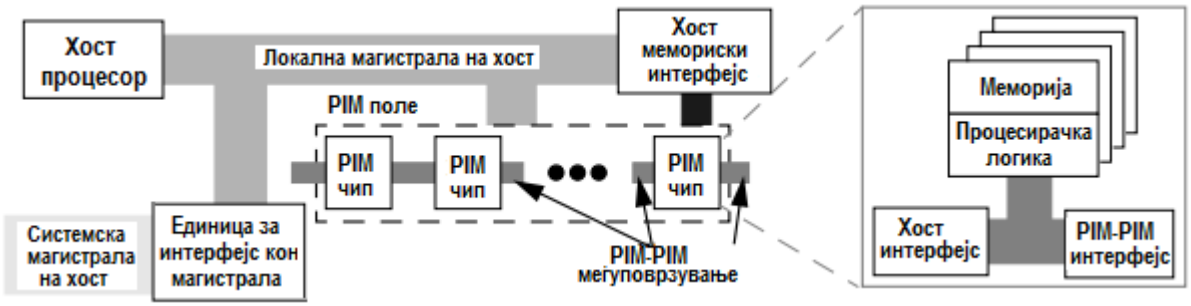


Слика 16 Архитектура на PIM чип во Terasys систем, [39].

Terasys системот може да содржи најмногу 32K процесирачки елементи, кои за инструкциска рата од 10^9 инструкции/секунда овозможуваат извршување на максимум $3{,}2 \times 10^{11}$ бит операции во секунда. Како резултат на овие можности за масивен паралелизам, овој систем може да се примени во апликации кои вклучуваат голем број независни пресметки или податочно-паралелни операции, како што се процесирање на слики на ниско ниво или матрични алгоритми.

Diva претставува уште еден систем кој интегрира поле од PIM чипови и стандарден процесор, каде што PIM чиповите може да се однесуваат како паметни мемориски копроцесори или стандардна меморија, [40]. На сл. 17 е прикажан блок дијаграм на овој систем, каде може да се забележи дека множеството од PIM чипови е поврзано со стандардниот процесор преку посебен мемориски интерфејс, додека пак PIM чиповите меѓу себе се поврзани со специфични PIM-

PIM меѓуповрзувања преку кои комуницираат, без употреба на стандардниот процесор. Стартувањето на PIM пресметките од страна на хостот и комуникацијата помеѓу PIM чиповите при собирање на резултати, синхронизирање на активности и пристап до не-локални податоци се извршува преку испраќање на парцели. Парцела претставува еден вид на активна порака која ја содржи референцата до функцијата која треба да се повика кога парцелата ќе биде примена.

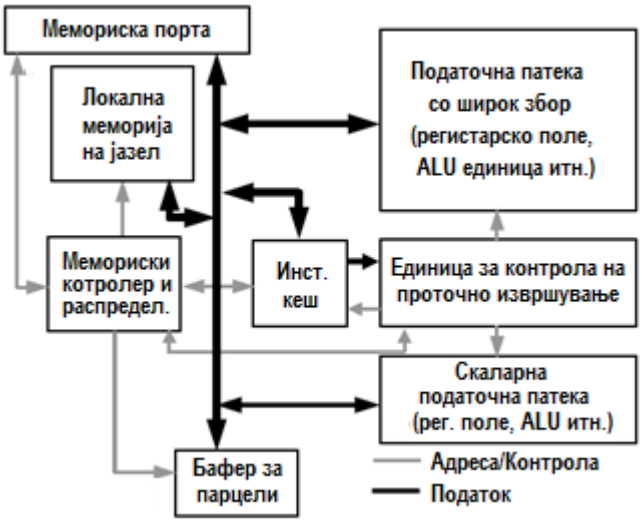


Слика 17 Блок дијаграм на Diva систем, [40].

Секој PIM чип вклучува еден или повеќе PIM јазли, кои заеднички споделуваат еден PIM-PIM интерфејс и еден интерфејс кон хостот (стандардниот процесор). Тоа е прикажано на десната страна од сл. 17, каде е дадена организација на PIM чип, составен од четири PIM јазли. Секој PIM јазел вклучува неколку MB локална меморија и процесирачки елементи наменети за работа со 32-битни кратки и 256-битни долги податочни зборови. PIM-PIM интерфејсот се користи за упатување на парцелите во и надвор од PIM чипот, а хост интерфејсот овозможува проследување на стандардни или PIM мемориски барања од хостот.

На сл. 18 е прикажана архитектурата на PIM јазел во PIM чип од Diva систем, заедно со неговите мемориски, контролни и процесирачки компоненти, [41]. Мемориският систем во PIM јазелот содржи: локална меморија; инструкциски кеш; мемориска порта за комуникација со стандардниот процесор; бафер за парцели (иницирани од хостот или од останатите PIM чипови); и мемориски контролор и распоредувач. Процесирачкиот дел од PIM јазелот вклучува две податочни патеки, кои се извршуваат од еден инструкциски тек, под контрола на единицата за проточно извршување, која обезбедува PIM јазелот да работи проточно во пет фази. Скаларната податочна патека извршува стандардни операции со 32-битни операнди, а податочна патека со широк збор извршува паралелни операции над 256-битно поле од податоци со големина од 8, 16 или

32 бита. Секоја податочна патека користи свое поле од регистри за општа намена, при што директна размена на податоци (без учество на хостот) помеѓу регистрите од двете податочните патеки се врши со специјални инструкции.



Слика 18 Архитектура на PIM јазел во Diva систем, [41].

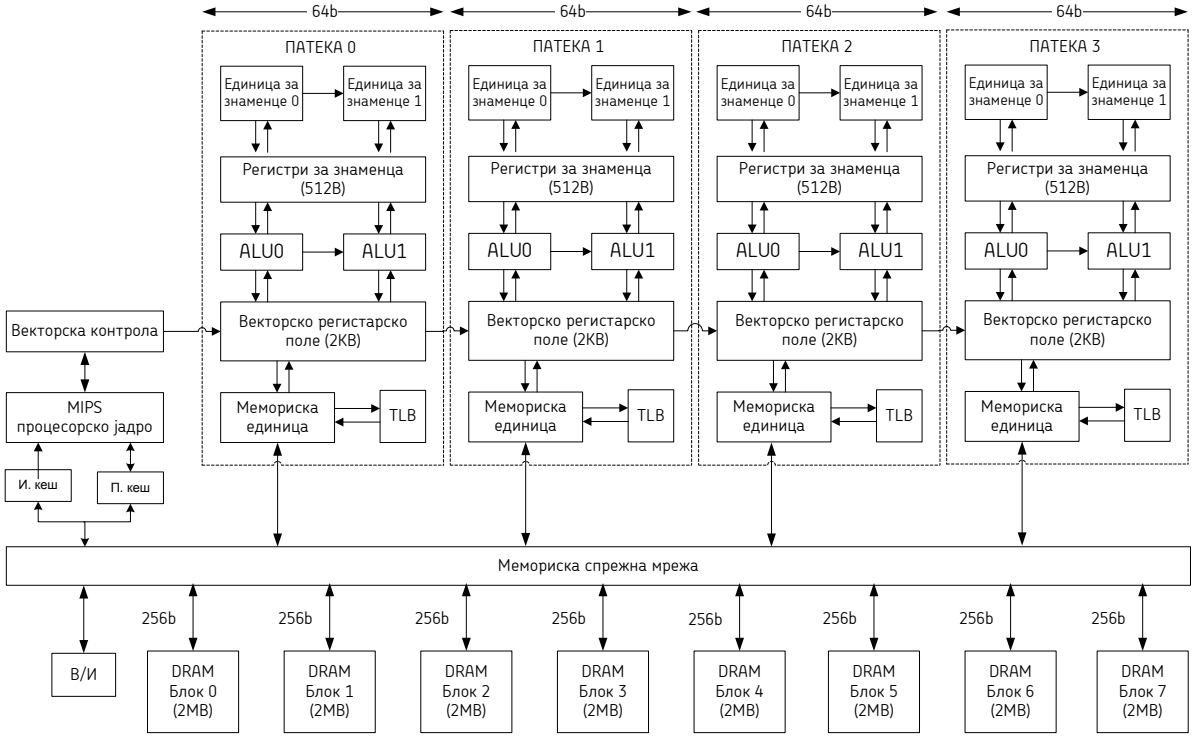
PIM јазлите овозможуваат приближување на процесирањето до податоците на тој начин што извршуваат паралелни пресметки, секој со независни податоци од својата локална меморија. Ваквиот начин на работа е многу погоден за примена во апликации кои вклучуваат регуларен (пр. процесирање на слики) или нерегуларен пристап до податоци (пр. бази на податоци). Сепак, за да се овозможи Diva пристап на сметање неопходно е да се развие компајлерска технологија, која ќе обезбеди ефикасно распоредување на податоците; да се овозможи одржување на кохерентноста во PIM чиповите и кешот на хостот; и да се развие поддршка за парцелно работење.

3.3.3. Интелигентен RAM

За разлика од претходните два чипа, професорот Д. Патерсон заедно со мала група на студенти од Беркли Универзитетот во Калифорнија, претставиле едно поинаков мемориско-пресметковен чип кој што вклучува векторско процесирање, а е познат како векторски интелигентен RAM (VIRAM), [42]. Овој чип интегрира: проточен MIPS процесор со 6 фази, кој работи со 8KB директно мапирани податочен и инструкциски кеш; векторска единица со 8KB векторско регистарско поле составено од 32 векторски регистри, секој со 32 64-битни

елементи; и 16 MB вградена DRAM меморија организирана во 8 банки, секоја со големина од 2MB. Инструкциското множество на VIRAM чипот содржи стандардни MIPS инструкции и дополнителни специфични векторски инструкции за извршување на податочно-паралелни операции со вектори.

Векторската единица во VIRAM чипот се однесува како копроцесор, кој ги извршува векторските инструкции, кои се издадени од страна на скаларното MIPS процесорско јадро. Оваа копроцесорска единица има клучна улога при обезбедување на ефикасно искористување на мемориската пропусност на вградениот DRAM и воедно постигнување на високо ниво на ситно-зрнест податочен паралелизам, [42]. Покрај тоа, применетиот пристап на векторско процесирање се карактеризира со добро развиена компајлерска поддршка, која овозможува да се задржи програмабилноста на VIRAM чипот. Сето тоа укажува дека комбинацијата од векторски копроцесор и скаларно процесорско јадро креира општо-наменска архитектура која може да постигне високи перформанси, избегнувајќи ја хардверската комплексност на суперскаларните процесори и компајлерската комплексност на VLIW процесорите. Повеќе детали за архитектурата на векторскиот IRAM чип се дадени на сл. 19.



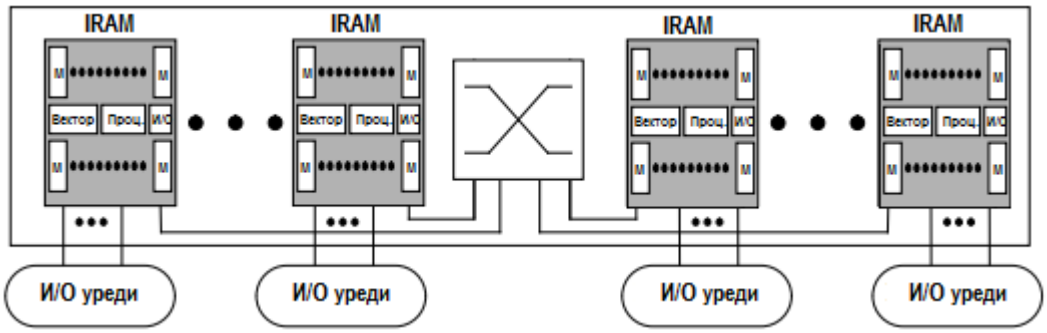
Слика 19 Архитектура на векторски IRAM чип.

Според сл. 19 може да се забележи дека векторскиот копроцесор во VIRAM чипот имплементира 256-битна податочна патека, која е поделена на четири 64-битни патеки (векторски проточни линии) кои паралелно се извршуваат. Секоја 64-битна патека користи 2 KB векторски ресурси и две целобројни функционални единици (ALU0 и ALU1), од кои едната овозможува и работа со реални броеви. Овие 64-битните патеки можат да се поделат на две и на тој начин да овозможат паралелно извршување на осум 32-битни операции. Според тоа, максималните перформанси кои ги постигнува VIRAM чипот се: 3,2 GFLOPS (Giga Floating point operations per second) за 32-битни реални броеви со единечна прецизност и 6,4 GOPS (Giga operations per second) за 32-битни цели броеви, [80].

Сл. 19 покажува дека скаларниот MIPS процесор и векторскиот копроцесор пристапуваат до 16MB вградена DRAM меморија, преку мемориска спрежна мрежа, која се карактеризира со максимална пропусност од 12 GB/s. Вградена DRAM меморија е организирана во 8 засебни блокови, поврзани со мемориската спрежна мрежа преку 256-битни синхрони магистрали. Ваквата организација овозможува директна едно-тактна размена на 256 бита помеѓу мемориските единици од векторските патеки и DRAM меморијата. После вчитувањето на податоците во векторското регистарско поле, векторскиот копроцесор може да извршува паралелни векторски операции. На сличен начин MIPS процесорот може да врши размената на блокови помеѓу вградениот DRAM и кеш меморијата, при извршување на стандардни load/store мемориски операции.

Примената на вграден наспроти стандарден DRAM во VIRAM чипот не само што ја зголемува мемориската пропусност туку и го намалува мемориското доцнење и потрошувачката на енергија (намален број на мемориски пристапи надвор од чипот), [42]. За разлика од овие предности, векторскиот IRAM се соочува и со некои проблеми како што се: недостаток од скалабилност; дивергенција на брзината и цената на чипот; и потешкотии со прифаќање на пазарот. Како резултат на овие карактеристики, VIRAM чипот наоѓа широка примена во вградливите системи и портабилните уреди. Станува збор за апликации како што се: мултимедија (обработка на слики, видео и аудио), работа со бази на податоци, податочно рударење, дигитално процесирање на сигнали и било какви алгоритми кои лесно можат да се паралелизираат на податочно ниво.

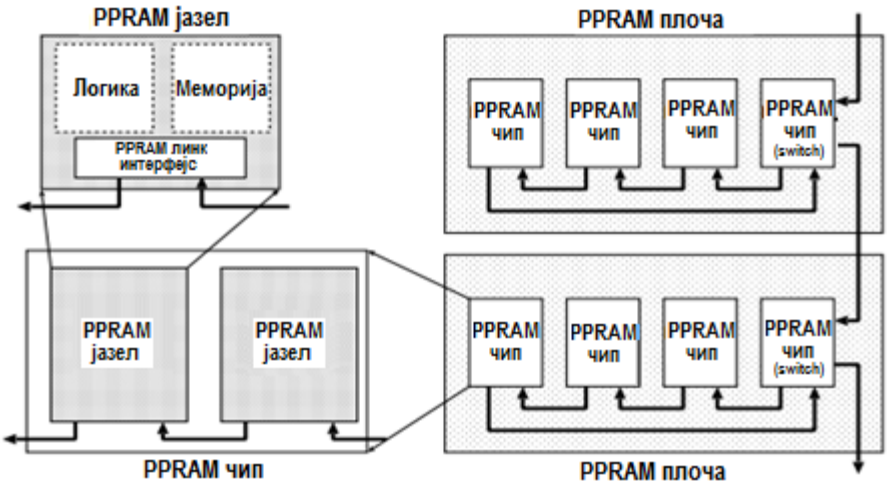
Како надградување на интелигентниот RAM, во [43] е претставен дизајн на SmartSIMM систем, изграден од повеќе IRAM чипови (8 до 16), кои се кластерирани во еден заеднички SIMM (Single In-line Memory Module) модул. Ваквата организација е прикажана на сл. 20, каде може да се забележи дека поединечните IRAM чипови се поврзани помеѓу себе со спрежна мрежа и гигабитни сериски линкови. Имено, целиот SmartSIMM систем функционира како мрежа од работни станици (NOW - network of workstations), кои извршуваат паралелни, секвенцијални или интерактивни задачи. За да се обезбеди успешно управување и искористување на работните станици, се користи посебен GLUnix оперативен систем.



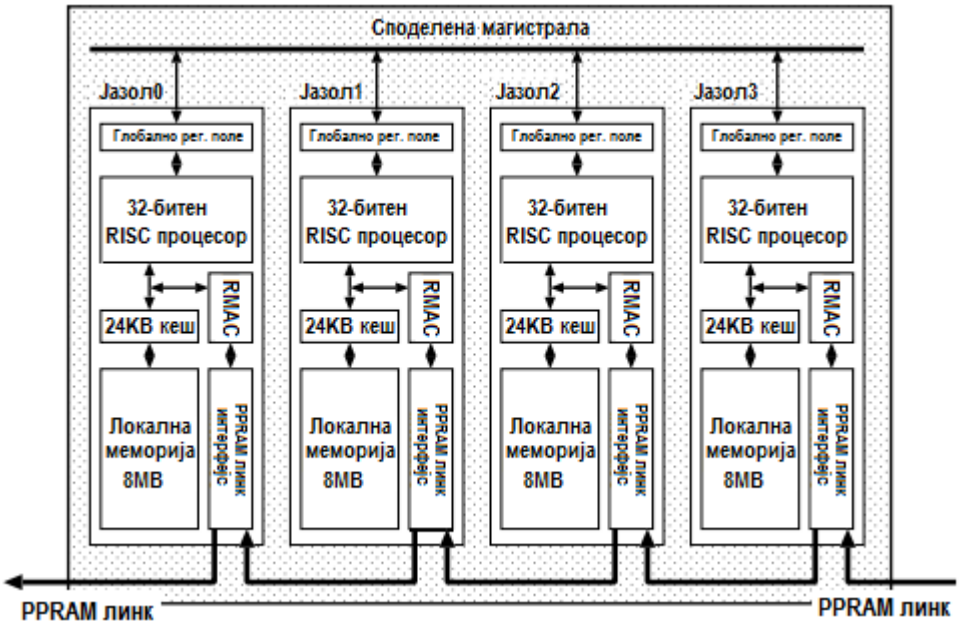
Слика 20 Блок дијаграм на SmartSIMM систем, [43].

3.3.4. Паралелен процесирачки RAM

Сличен концепт на SmartSIMM системот е даден во [44], каде е предложен систем од поврзани паралелно-процесирачки RAM (PPRAM) чипови. Секој од овие PPRAM чипови вклучува еден или повеќе PPRAM јазли, кои се поврзани со посебни PPRAM линк интерфејси. PPRAM јазлите интегрираат процесирачки и мемориски ресурси на заеднички чип, комбинирајќи нула или повеќе битови DRAM и/или SRAM и/или Flash меморија со нула или повеќе општо-наменски или специфични процесори. Целиот пристап на интегрирање и поврзување на PPRAM јазлите и чиповите е прикажан на сл. 21, каде е даден блок дијаграм на еден произволен PPRAM-базиран систем. Генерално може да се каже дека функционалноста, големината и перформансите на имплементираниот PPRAM систем ќе зависат од структурата на PPRAM јазлите и чиповите, кои се употребени во дизајнот на системот. Овој пристап воведува флексибилност и скалабилност во конструирањето на компјутерските системи, што претставува голема предност на PPRAM системите во однос на останатите PIM решенија.



Слика 21 Блок дијаграм на PPRAM-базиран систем, [44].

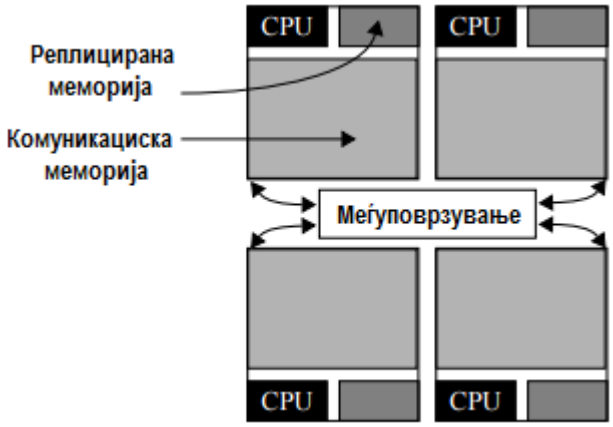


Слика 22 Архитектура на PPRAM^{R} чип, [44].

На сл. 22 е прикажана архитектура на една имплементација (PPRAM^{R}) на PPRAM чип, составен од четири PPRAM јазли и 32 MB вградена DRAM меморија. Секој јазел во PPRAM^{R} чипот вклучува 32-битен RISC процесор, 8MB локална меморија, 24KB кеш меморија, контролор за далечински пристапи (RMAC) и 16-битен PPRAM линк интерфејс. Секој PPRAM јазел разменува податоци помеѓу кешот и локалната меморија преку 1024-битна внатрешна магистрала со пропусност од 3GB/s, и воедно врши мемориските пристапи надвор од јазелот преку посебен PPRAM линк интерфејс. Дополнително, сите јазли во PPRAM^{R} чипот споделуваат глобално регистарско поле, кое им овозможува брза комуникација и синхронизација на ниво на чип. Според анализите дадени во [44] се покажува дека еден ваков PPRAM чип може соодветно да замени суперскаларен процесор.

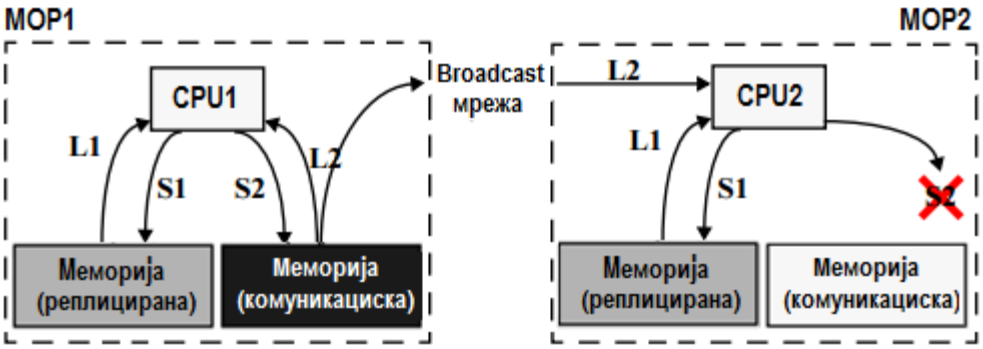
3.3.5. DataScalar систем

DataScalar претставува уште еден систем кој го користи пристапот на процесирање во меморија. Овој систем поврзува еден или повеќе процесори кои интегрираат делови (региони) од главната меморија, при што секој процесор пристапува само до својата интегрирана меморија. DataScalar системот се обидува да го подобри времето на извршување на програми кои тешко се паралелизираат со мултипроцесирање и чие податочно множество е многу големо и не може да се смести во меморијата доделена на еден процесор, [45]. За таа цел, овој систем го реплицира извршувањето на истата програма во секој процесор и го дистрибуира податочното множество на програмата во интегрираните мемории на процесорите. Всушност, на сл. 23 е прикажан блок дијаграм на DataScalar систем, изграден од четири процесорско-мемориски елементи.



Слика 23 Блок дијаграм на DataScalar систем, [45].

На сл. 23 се забележува дека секој MOP (Memory On Processor) елемент содржи своја комуникациска и реплицирана меморија, до кои може да пристапува со load и store инструкции. Комуникациската меморија се користи за складирање на дистрибуираните податоци од програмата, а пак реплицираната меморија служи за зачувување на најчесто пристапуваните податоци, со цел да го намалат бројот на пристапи надвор од чипот. При извршување на истата програма, секој MOP елемент повеќенасочно (broadcast) ги препраќа податоците од својата комуникациската меморија до кои пристапува со load инструкции, па останатите MOP елементи наместо да праќаат барања за пристап до податоци надвор од чипот, тие само треба да почекаат да им пристигнат испратените податоци преку поврзувачката мрежа.



Слика 24 Читање и запишување во реплицирана и комуникациска меморија на MOP елементи во DataScalar систем, [45].

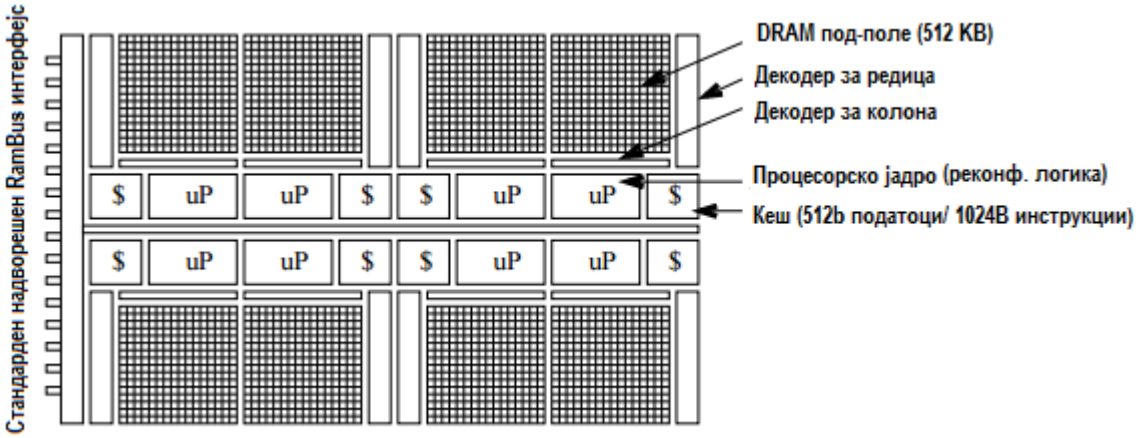
На сл. 24 е прикажано како се прави читање(load)/запишување(store) на податоци од/во реплицирана и комуникациска меморија на два MOP елементи кои извршуваат иста програма. Се смета дека оваа програма на почеток чита (L1), па потоа запишува (S1) од и во реплицираната (идентична) меморија на двата MOP елементи. Овие два мемориски пристапа на читање (L1) и запишување (S1) се извршуваат од страна на двата MOP елементи. Веднаш потоа програмата чита (L2), па запишува (S2) податок од и во комуникациската меморија на првиот елемент MOP1. Во таков случај MOP1 елементот извршува читање (L2) на податок, кој потоа повеќенасочно го препраќа преку поврзувачката мрежа, до вториот елемент MOP2. Операцијата на запишување (S2) на податок се извршува само во комуникациската меморија на првиот елемент MOP1, бидејќи MOP2 елементот може да врши запишување само во својата комуникациска меморија.

Ваквата еднонасочна комуникација помеѓу MOP елементите овозможува да се намали бројот на пристапи надвор од чипот, поради тоа што ниту еден MOP елемент не испраќа барање за читање или запишување на податоци во друг MOP елемент. На овој начин се овозможува секој MOP елемент да ги добие податоците кои му се потребни без да ги побара и при тоа да ги изврши сите потребни пресметки, правејќи промени само над податоците од својата локална меморија. Овој принцип на работа може дополнително да се подобри со примена на техники, како што се: мемориски prefetch или препраќање само на резултати, [45]. Овие подобрувања на DataScalar системот овозможуваат тој да ги надмине перформансите на еднопроцесорските системи со стандардна мемориска хиерархија во повеќе нивоа. Како и да е, сето тоа мора да биде проследено со развој на соодветна софтверска поддршка, која ќе овозможи ефикасно статичко и динамичко реплицирање на податоците во DataScalar системот.

3.4. Модел на активни страници на интелигентна меморија

За разлика од концептот на интегрирање на процесор и меморија во заеднички чип, во [46] е предложен систем кој користи одвоен процесор и паметна меморија, организирана во повеќе активни (виртуелни) страници со еднаква големина. Секоја активна страница содржи една податочна (виртуелна) страница сместена во DRAM меморија која е придружена со посебен реконфигурабилен логички блок во кој се имплементирани функциите кои можат да се извршуваат над податоците од таа страница. На пример, една активна страница може да содржи една податочна структура поле и множество на функции за вметнување, бришење и наоѓање на податоци од соодветното поле. На тој начин моделот на активни страници комплетно го заменува стандардниот мемориски систем, со таа разлика што покрај складирање на податоци, овозможува и обработка на податоците од активните страници на интелигентната меморијата.

На сл. 25 е даден интелигентен мемориски систем, изграден од 8 активни страници. Секоја активна страница вклучува реконфигурабилен логички блок составен од 256 логички елементи (4-element Look Up Table) и DRAM под-поле со капацитет од 512KB во кое може да се смести една податочна виртуелна страна. Реконфигурабилната логика имплементира VLIW, вектор или стандардно RISC процесорско јадро со кеш меморија, кое ги извршува функциите доделени на активната страница. Ова процесорско јадро користи виртуелни адреси за пристап до податоците од локалната и останатите активни страници, кои се алоцирани во иста група на активни страници на заеднички процес.



Слика 25 Архитектура на интелигентен мемориски систем со осум активни страници, [46].

Моделот на активни страници е наменет за примена во конвенционалните компјутери, кои содржат еден главен процесор и повеќе-нивовска меморија, каде виртуелната меморија користи страничење, односно ја дели главната меморија во 100 до 1000 физички рамки за виртуелни страни. Во таков случај, со имплементација на моделот на активни страница, секоја виртуелна страница ќе стане активна страница која може да извршува локални пресметки. Според тоа, колку повеќе активни страници користи главната меморија толку поголем паралелизам на податочно ниво се постигнува.

Главниот процесор во еден компјутерски систем кој користи активни страници е сличен на контролен процесор кој работи со податочно-паралелни машини. Овој процесор може да извршува повеќе операции, вклучувајќи: стандардни мемориски пристапи во активни страници; специфицирање на функции за активни страници; формирање на групи на активни страници; и доделување на функции на определена група на активни страници. Координацијата помеѓу активните страници и главниот процесор се одвива преку синхронизациски променливи, а комуникацијата помеѓу активните страници при извршување на не локални мемориски пристапи се одвива преку главниот процесор со издавање на прекини. Анализите дадени во [81] покажуваат дека компјутерски систем кој користи активни страници може да постигне до 1000 пати побрзо извршување на програми, во споредба со идентичен систем со стандардна меморија.

Најголем предизвик во компјутерски систем кој користи активни страници е да се направи ефикасна поделба на програмскиот код во глобални задачи наменети за главниот процесор и локални задачи наменети за активните страници. Во главно, глобалните задачи се соодветни за извршување на посложени пресметки (пр. аритметика со реални броеви) кои ќе ја искористат високата брзина на работа на процесорот и неговите пресметувачки можности, додека пак локалните задачи се соодветни за извршување на операции на манипулација на податоци или аритметика со цели броеви кои ќе ја искористат високата мемориска пропусност на активните страници. За да се автоматизира процесот на поделба на задачите, моделот на активни страници користи рачно изработени библиотеки, кои можат да се повикаат од конвенционален код. Ова може да се подобри со развој на специфичен компајлер за дадениот систем.

3.5. Споредбена анализа

Сите претходно дискутирани системи се обидуваат на различни начини да го надминат проблемот на тесно грло во комуникацијата помеѓу процесорот и меморијата и со тоа да овозможат намалување на времето на пристап до меморија, [32]. Покрај добро познатите техники за намалување/прекривање на мемориското доцнење (нередоследно и шпекулативно извршување, претходно земање на податоци, алгоритми за предвидување на гранење или повеќенитност), разгледаните системи имплементираат различни мемориско-центрични пристапи на интегрирање или доближување на процесирачките ресурси до меморијата (или меморискиите блокови), каде се сместени податоците кои треба да се обработуваат. Пристапот на процесирање во или блиску до меморијата обезбедува повеќе предности (зголемување на внатрешната мемориска пропусност и намалување на мемориското доцнење), но од друга страна побарува промени во архитектурата на системот и неговиот модел на програмирање, што пак има големо влијание врз подрачјето на примена на системот. Подетална анализа на најбитните карактеристики на мемориско-центричните системи кои беа претставени во претходните поглавја е дадена во табела 2.

Табела 2 Споредба на различни мемориско-центрични системи.

Систем	Карактеристики
PERL процесор без регистри, [33], [79]	• **Предности:** не употребува експлицитни laod/store инструкции (работи директно со кеш меморија); помал број на инструкции во програма, за разлика од DLX RISC процесор; помало или слично време на извршување на програми, во споредба со DLX RISC процесор; • **Маани:** долги инструкции (128b =16B); два пати поголеми програми, во споредба со DLX RISC процесор; користи C cross compiler (базиран на GCC), наместо специфичен компајлер за системот; • **Намена:** општа намена;
Бележник (Scratch Pad) меморија, [34], [35]	• **Предности:** голема брзина (време на пристап од еден такт циклус); помала површина и потрошувачка на енергија во споредба со кеш меморија; користи посебен дел од меморискиот адресен простор (бележник меморијата не е редундантна копија на дел од главната меморија како што е кешот); • **Маани:** мал капацитет (неколку KB); комплексност во софтверот (софтверски управувана алокација на податоци во меморијата); побарува употреба на комплексни техники за распоредување на податоци во бележник SRAM меморијата и главната DRAM меморија; • **Намена:** вградливи процесорски системи;

Систем	Карактеристики
SUN PIM чип, [37]	• **Предности:** голема внатрешна мемориска пропусност; намалување на конфликти во кеш со употреба на жртвен кеш; ниска цена (мемориски процес на производство); помало или слично време на извршување на програми, во споредба со суперскаларен процесор; скалабилност (примена во дистрибуирана повеќе-процесорска околина со споделена кеш меморија) • **Маани:** ограничена количина на меморија во чип (256Mb); жртвениот кеш воведува дополнителна комплексност на хардверско ниво; меморискиот процес на производство ја ограничува брзината на работа на чипот; побарува примена на комплексни техники за одржување на кохерентност на споделениот кеш во дистрибуирана повеќе-процесорска околина; • **Намена:** општа намена
Mitsubishi M32R/D чип, [38]	• **Предности:** користи широка 128-битна податочна магистрала во чип; ниска потрошувачка на енергија; вклучува специфична единица за множење и акумулирање (примена во дигитално процесирање); користи оптимизиран C компајлер кој генерира покомпактен код (поддршка за 16b и 32b инструкции); флексибилност (два режими на работа: податоци и програми се сместуваат во DRAM во чип; или податоци се складираат во DRAM во чип, а програми се складираат во надворешен ROM); • **Маани:** ограничена количина на меморија во чип (16Mb); спори мемориски пристапи до ROM надвор од чипот, преку четири пати побавна 16-битна магистрала; оптимизацијата на компајлерот и хардверот воведува додатна комплексност во дизајнот на чипот; се применува само за апликации чие податочно множество може да се смести во DRAM меморијата во чипот; • **Намена:** вградливи системи (PDA, мултимедија, дигитално процесирање);
Terasys систем, [39]	• **Предности:** масивен паралелизам (содржи до 32K 1-битни процесирачки елементи организирани во SIMD поле); голема внатрешна мемориска пропусност; флексибилност (PIM чиповите можат да работат како стандардна меморија или паметни мемориски копроцесори кои извршуваат податочно-паралелни пресметки); имплементира специфични механизми за комуникација помеѓу процесирачките елементи; користи специфичен dbC (data-parallel bit C) програмски јазик на високо ниво; • **Маани:** потреба од развивање на специфична софтверска поддршка, која ќе го препознае паралелизмот во програмите; потреба од дефинирање на нов (паралелен) модел на програмирање; неискористеност на пресметувачките ресурсите при недоволен паралелизам во програмите; повисока цена на чинење од стандарден DRAM; • **Намена:** апликации кои вклучуваат многу податочно-паралелни пресметки; процесирање на слики на ниско ниво; матрични алгоритами;
Diva систем, [40]	• **Предности:** PIM чиповите ја подобруваат мемориската пропусност за 10 до 100 пати и го намалуваат мемориското доцнењето, во споредба со стандарден DRAM; флексибилност (PIM чиповите можат да работат како стандардна меморија или паметни мемориски копроцесори); Diva PIM чиповите се први паметни мемориски уреди кои користат виртуелно адресирање; Diva системот обезбедува директна комуникација помеѓу PIM чиповите со размена на парцели, преку посебни PIM-PIM меѓу-поврзувања (без учество на главниот процесор); висок степен на вграден паралелизам во PIM чиповите (тие содржат повеќе PIM јазли кои може да работат паралелно); секој PIM јазел вклучува локална меморија, 32-битна скаларна податочна патека и 256-битна широка податочна патека, која ја искористува големата мемориска пропусност во чипот; времето на извршување на програми во еден Diva PIM чип е 3,3 пати побрзо од сличен процесор кој работи со стандарден DRAM;

Систем	Карактеристики
Diva систем, [40], [41]	• **Маани:** ограничена количина на локална меморија во PIM јазел (неколку MB); побарува употреба на специфичен хардвер и табели за преведување во секој PIM јазел со цел да се обезбеди поддршка за виртуелно адресирање; парцелно-базираниот механизам за комуникација помеѓу PIM чиповите, воведува усложнување во дизајнот на системот; прстен топологијата која се користи кај PIM-PIM меѓу-поврзувањата не е погодна за проширување на системот во големи размери; механизмите за одржување на кохерентност помеѓу PIM чиповите и кеш меморијата на хост процесорот додаваат дополнителна комплексност во дизајнот на системот; воведува потреба од развивање на специфична компајлерска поддршка за системот и дефинирање на нов (паралелен) модел на програмирање; • **Намена:** апликации кои вклучуваат регуларен пристап до податоци (пр. процесирање на слики) или нерегуларен пристап до податоци (пр. бази на податоци);
Интелиген -тен RAM, [42], [43], [80]	• **Предности:** 5 до 10 пати помало мемориско доцнење, 50 до 100 пати поголема мемориска пропусност, и 2 до 4 подобра енергетска ефикасност, во споредба со стандарден DRAM; ефикасно искористување на расположливата мемориска пропусност преку широка податочна внатрешна магистрала, која овозможува размена на 256 бита помеѓу вградената DRAM меморија, процесорскиот кеш и векторскиот копроцесор; висок степен на паралелизам со извршување на податочно-паралелни операции од векторски копроцесор, кој вклучува 256-битна широка податочна патека, поделена на четири 64-битни проточни линии (со по две извршни единици), кои работат паралелно; векторскиот копроцесор постигнува максималните перформанси од 1,6/3,2/6,4 GOPS (64b/32b/16b), при обработка на цели броеви, и 1.6 GFLOPS (32b), при обработка на реални броеви; векторското процесирање е поедноставно за имплементација на хардверско ниво во споредба со суперскаларните дизајни, и воедно располага со подобро развиена компајлерска технологија во споредба со VLIW процесорите; векторскиот IRAM постигнува слични или подобри времиња на извршување на FFT пресметки со 256-2048 точки, во споредба со други познати процесори за дигитална обработка на сигнали; скалабилност (креирање на SmartSIMM систем со поврзување на 8 до 16 IRAM чипови во еден SIMM модул); • **Маани:** ограничена количина на меморија во чип (16 MB); неефикасна искористеност на векторскиот копроцесор за програми со ниско ниво на податочен паралелизам; за разлика од стандардната DRAM меморија, процесорските ресурси во IRAM чипот влијаат на софтверот кој што може да се извршува во чипот; дивергенција на брзината и цената во зависност од процесот на производство (IRAM чипот користи меморискиот процес на производство); времето на тестирање е поголемо, во споредба со стандарден DRAM; проблеми со прегревање на чипот; потешкотии со прифаќање на пазарот, бидејќи во денешно време процесорите и меморијата се развиваат во две посебни индустрии, секоја со различни цели, намени и технологии; • **Намена:** вградливи системи и портабилни уреди; мултимедија (обработка на слики, видео и аудио); работа со бази на податоци (подредување, пребарување, податочно рударење); криптографија (RSA, DES/IDEA, SHA/MD5); или било какви алгоритами кои можат да се паралелизираат на податочно ниво;

Систем	Карактеристики
Паралелен процеси-рачки RAM, [44]	• **Предности:** флексибилност и скалабилност (избор на оптимална конфигурација на PPRAM чипот според намена); помало мемориско доцнење и потрошувачка на енергија во споредба со стандарден DRAM; голема внатрешна меморискa пропусност со употреба на широка 1024-битна внатрешна податочна магистрала за размена на податоци помеѓу процесорскиот кеш и локалната меморија; високо ниво на расположлив паралелизам во PPRAM чиповите (тие вклучуваат повеќе PPRAM јазли, кои работат паралелно); поедноставен дизајн од суперскаларните процесори; 2,22 пати подобри перформанси (време на извршување) во споредба со сличен суперскаларен процесор; • **Маани:** ограничена количина на меморија во PPRAM јазел (неколку MB); проблеми со постигнување на висока брзина на PPRAM логиката како онаа на стандардните процесори, и голема густина на локалната меморија како онаа на стандардните DRAM мемории; зголемување на времето на дизајнирање и тестирање; недостаток на соодветна CAD (Computer-aided design) технологија; недоволна искористеност на PPRAM јазлите, во случај кога програмата која се извршува не може да се паралелизира; потреба од дизајнирање на специфична компајлерска поддршка и паралелен модел на програмирање; • **Намена:** било каква намена, во зависност од конфигурацијата на чипот;
DataScalar систем, [45]	• **Предности:** главната меморија е дистрибуирана помеѓу повеќе процесорско-мемориски MOP елементи, кои брзо пристапуваат до својата локална меморија; конзистентноста во дистрибуираната меморија лесно се одржува, бидејќи секој MOP елемент може да пристапи само до својата локална меморија; намален сообраќај, поради еднонасочната комуникација помеѓу MOP елементите; дополнително намалување на сообраќајот помеѓу MOP елементите со примена на техниката на препраќање само на резултати; намалување на комуникациско доцнење со примена на техниката на мемориски prefetch; побрзо време на извршување на еднопроцесорски програми кои не можат лесно да се паралелизираат; • **Маани:** ограничена количина на локална меморија во MOP елемент; реплицирањето на програмата во сите MOP елементи од DataScalar системот побарува повеќе мемориски ресурси и поголема хардверска комплексност, во споредба со едно-процесорски систем; користи комплексен механизам за одржување на конзистентноста во кешот, кој врши ажурирање во кешот само кога мемориската операција е завршена - commited; проблеми со одржување на конзистентноста во табелата на страници на сите MOP елементи; употребата на магистрала со прстен топологија за поврзување помеѓу MOP елементите воведува недетерминизам во времетраењето на програмите, бидејќи повеќе-насочно испратените пораките пристигаат во различен редослед кај MOP елементите; • **Намена:** програми кои тешко се паралелизират и чие податочно множество не може да се смести во локалната on-chip меморија на еден процесор;
Модел на активни страници, [46], [81]	• **Предности:** интелигентен мемориски систем, кој извршува определени операции директно врз податоците од виртуелните страници; флексибилност (активните страници можат да се однесуваат како стандардна меморија или паметни мемориски копроцесори); поддршка за виртуелно адресирање; висок степен на расположлив паралелизам на податочно ниво со извршување на паралелни пресметки во активните страници; имплементира специфичен механизам за координација помеѓу активните страници и процесорот, со употреба на синхронизациски променливи; реконфигурабилната логика во активните страници обезбедува ниска цена и стандарден мемориски интер-фејс за паметната меморија; 1000 пати побрзо извршување на програми, во споредба со идентичен систем кој работи со стандардна DRAM меморија;

Систем	Карактеристики
Модел на активни страници, [46], [81]	• **Маани:** потреба за развивање на специфичен компајлер (наместо рачно кодирани библиотеки) и оперативен систем (ActiveOS е само прототип); потреба од дополнување на моделот на програмирање со поддршка за синхронизациски променливи; комуникацијата помеѓу активните страни при не-локални мемориски пристапи се одвива преку главниот процесор со издавање на прекини; неефикасна искористеност на активните страници, во случај кога програмата е со мало податочно множество; главниот процесор станува тесно грло во системот при обработка на програми со големо податочно множество; механизмите за одржување на кохерентност помеѓу активните страници и кеш меморијата на главниот процесорот додаваат дополнителна комплексност во дизајнот на системот; реконфигурабилната логика која се користи во активните страница се карактеризира со помала брзина од интегрираните кола со специфична намена (ASIC - application specific integrated circuit); • **Намена:** општа намена (стандардниот DRAM се заменува со активни страници);

Во табела 2 се прикажани предностите, недостатоците и подрачјето на примена на повеќе различни мемориско-центрични пристапи на сметање. Генерално може да се каже дека претставените мемориско-центрични чипови ги интегрираат или доближуваат процесирачките ресурси до меморијата и на тој начин овозможуваат да се намали мемориското доцнење и потрошувачката на енергија, а при тоа да се зголеми пропусноста помеѓу процесорот и меморијата. Дополнително, повеќето од овие чипови постигнуваат високо ниво на паралелизам, имплементирајќи широки податочни патеки, векторско процесирање или други пристапи на паралелно процесирање со употреба на повеќе мемориско-пресметковни елементи (PIM јазли, PIM чипови или паметни мемориски копроцесори). Вообичаено секој од мемориско-пресметковните елементи вклучува широка податочна магистрала која обезбедува брз пренос на податоци, помеѓу процесорскиот кеш или регистрите, и меморијата во чипот. И покрај тоа што повеќето од разгледаните мемориско-центрични системи го имплементираат стандардниот модел на мемориска хиерархија (кој побарува постојано копирање и движење на редундантни податоци), сепак интеграцијата на меморијата во чипот (пр. VIRAM, Diva, PPRAM) резултира со пократки мемориски пристапи, кои влијаат да се намали времето на извршување на програмите, во споредба со стандарден процесор со слични карактеристики. Слични резултати постигнува и процесорот PERL, кој ги отфрла регистрите од највисокото ниво во мемориската хиерархија и на тој начин го поедноставува пристапот до меморија.

Генерално, најголем проблем во разгледаните мемориско-центрични системи претставуваат капацитетот на интегрираната (придружена) меморија во чипот, како и брзината и цената на споениот мемориско-пресметковен чип, кои се ограничени од применетата технологија и процесот на производство. Дополнително, повеќето од дадените мемориско-центрични системи побаруваат развој на соодветна компајлерска поддршка, која ќе го препознае паралелизмот во програмите и ќе овозможи ефикасно искористување на внатрешната мемориска пропусност, преку генерирање на специфични инструкции наменети за извршување на: векторски операции во VIRAM чипот, податочно-паралелни бит пресметки во Terasys системот, манипулации со податоци или аритметика со цели броеви во активните страници од интелигентен мемориски систем итн. Според тоа, очигледно е дека промените на хардверско и софтверско ниво кои се воведени во дадените мемориско-центрични системи се тешко прифатливи на пазарот, бидејќи процесорите и меморијата од секогаш се произведуваат како посебни компоненти, каде меморијата нема никакво влијание на софтверот кој може да се извршува во процесорот.

Со оглед на претходната дискусија, може да се каже дека подрачјето на примена на разгледаните мемориско-центрични системи е најзастапено во доменот на вградливи или наменски системи. Поширока примена постигнуваат некои мемориско-центрични системи со дистрибуирана организација (пр. Diva, PPRAM, модел на активни страници) кои можат да се скалираат со употреба на повеќе процесорско-мемориски елементи, меѓусебно поврзани со магистрали. Паралелната работа на дистрибуираните процесорско-мемориски елементи побарува развој на специфични механизми за комуникација и синхронизација помеѓу нив и хост процесорот, како и техники за одржување на кохерентноста на споделената меморија во системот (пр. кешот на хост процесорот може да содржи податоци од локалната меморија на некој процесорско-мемориски елемент). Покрај тоа, од голема важност за дистрибуираниот систем е ефикасно да ги искористи паралелните процесорско-мемориски елементи и на тој начин да постигне високо ниво на паралелизам. Ова подразбира дефинирање на нов (паралелен) модел на програмирање, кој ќе овозможи оптимално распоредување на податоците и задачите помеѓу процесорско-мемориските единици и хост процесорот.

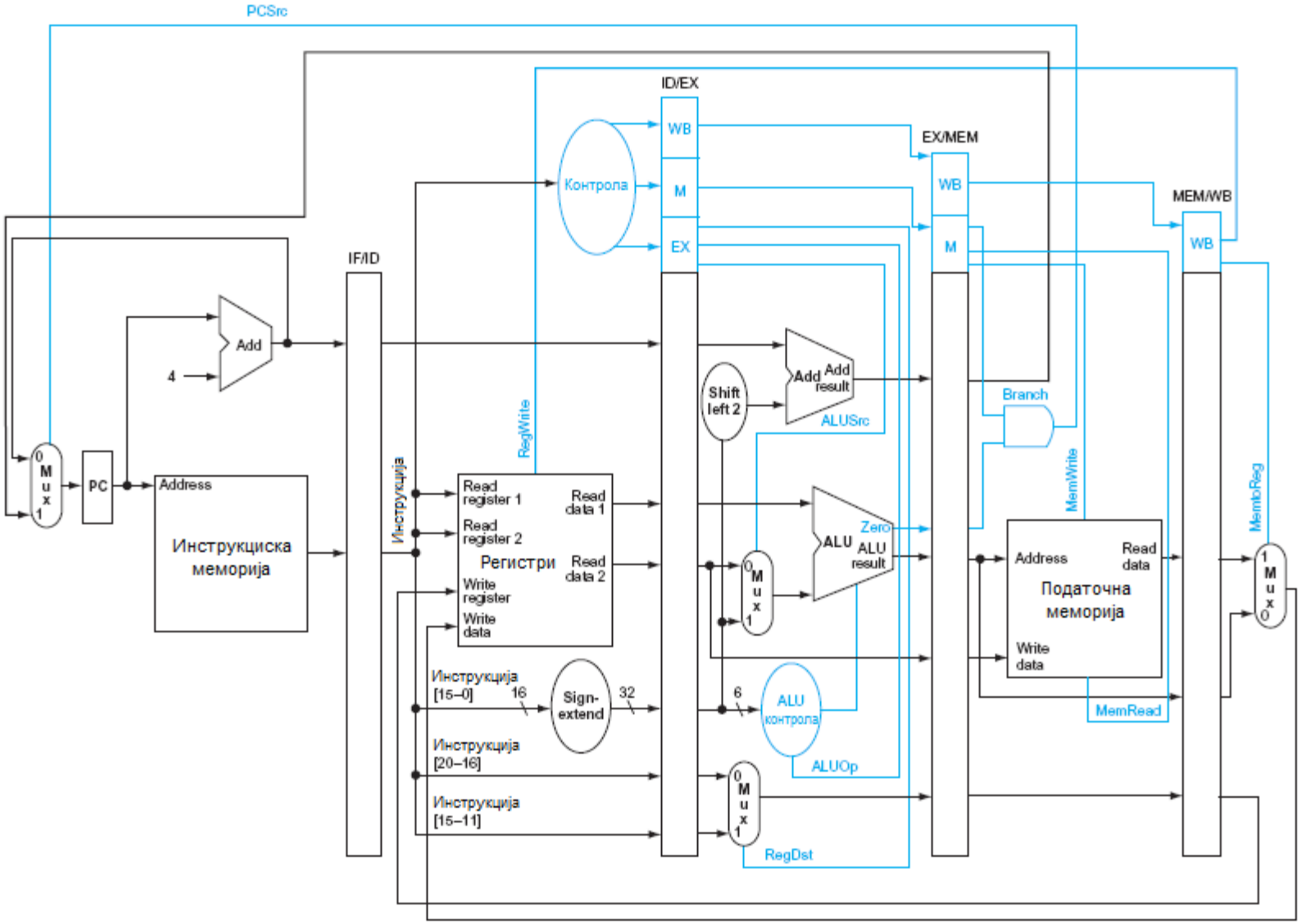
Тековните трендови во процесорскиот развој укажуваат дека голем дел од површината на чипот ќе биде наменета за меморија. Всушност, модерните микропроцесори користат дури до 50% од површината на чипот за да ги имплементираат највисоки нивоа во мемориската хиерархија, вклучувајќи: регистри за општа намена и едно или две нивоа на кеш меморија. Ако се смета дека овој тренд ќе продолжи, тогаш зголемувањето на јазот помеѓу мемориското доцнење на меморијата во чип и надвор од чип ќе предизвика да дојде до интегрирање на целата меморија во чипот, при што пристапите надвор од чипот ќе се третираат повеќе како промашувања во виртуелни страници, отколку како промашувања во кешот. Во тој случај меморијата во чипот би можела да се имплементира дури и со технологија која обезбедува голема густина и прифатливо мемориски доцнење, како што е вградениот DRAM. Ваквото интегрирање на процесор и меморија би обезбедило креирање на поедноставен и поисплатлив мемориски модел со слични или подобри перформанси од тековната мемориска хиерархија, кој сепак ќе воведе строги ограничувања во големината на меморија во чипот.

Согледувајќи ги последните технолошки трендови, може да се каже дека промените во стандардната мемориска хиерархија, кои се претставени во некои од претходно дискутираните пристапи (пр. PERL, Scrtchpad, IRAM) можат да бидат ветувачка насока за понатамошни истражувања. Како резултат на тоа, во овој докторски труд се предлага нов RISC-базиран мемориско-центричен процесор, сличен на PERL, кој обезбедува директен пристап до меморијата која е интегрирана во чипот, без да користи регистри за општа намена и кеш меморија. Предложената замена на врвот на мемориската хиерархија со меморија во чип е наменета да овозможи: отфрлање на непотребни движења и копирања на податоци во регистрите за општа намена, и блокови на податоци во кеш меморијата; намалување на капацитетот на редундантни мемориски ресурси; поедноставување на пристапот до меморија; и отфрлање на комплексни механизми за управување со меморијата. Покрај тоа, во ова истражување се испитува какви се можностите за да се дизајнира хардверски и софтверски прототип на предложениот процесор, кои подоцна ќе бидат употребени за анализа на неговите хардверски карактеристики и подрачје на примена (за различни типови на апликации).

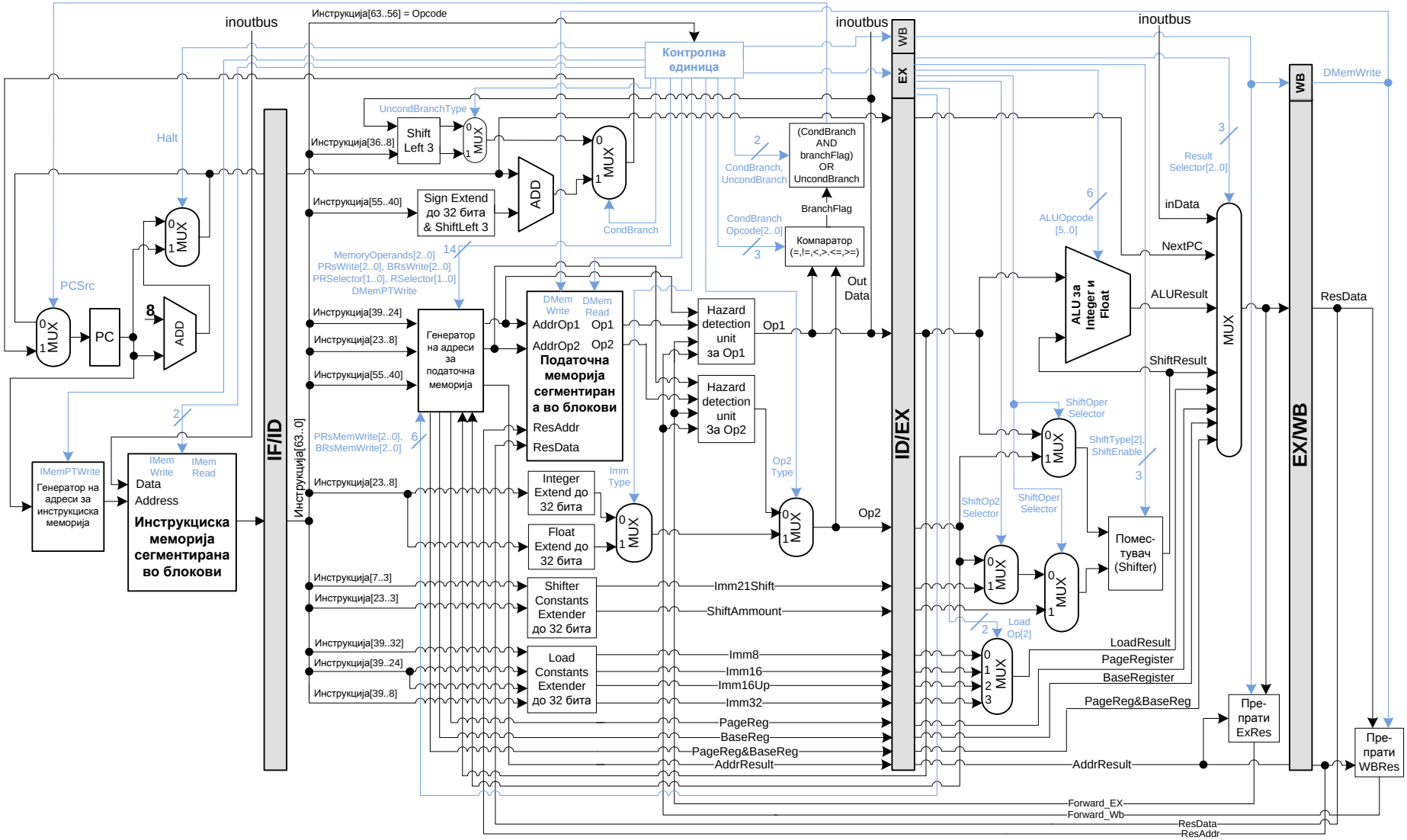
4. РАЗВОЈ НА НОВА RISC-БАЗИРАНА МЕМОРИСКО-ЦЕНТРИЧНА ПРОЦЕСОРСКА АРХИТЕКТУРА (MEMRISC)

Почетна основа за дизајнирање на новата процесорска архитектура е добро познатата RISC архитектура и една од нејзините имплементации, која широко се применува и воедно е добро документирана во водечката светска литература од областа на процесорските архитектури. Станува збор за MIPS имплементацијата на едно-тактна проточна RISC архитектура, претставена од страна на Д. А. Патерсон и Џ. Л. Хенеси во [1]. Дадениот MIPS процесор е исто употребен во дизајнот на безрегистарскиот процесор PERL, даден во претходната глава.

Главната филозофија на RISC архитектурата е да овозможи креирање на процесори кои ќе извршуваат мал број на типови на едноставни процесорски инструкции со голема брзина (повеќе милиони инструкции во секунда, или MIPS - Million Instructions Per Second). Според тоа, MIPS процесорот кој имплементира проточна RISC архитектура дефинира: инструкции со фиксна должина, едноставни адресни режими, мемориски пристапи со експлицитни load и store инструкции, хардверска контролна единица, големо множество на регистри за општа намена, и проточна работа во пет фази (земање на инструкција - IF, декодирање на инструкција/читање од регистри - ID, извршување/пресметка на ефективна адреса - EX, пристап до меморија/завршување на гранење - MEM, запишување на резултат - WB). На сл. 26 е прикажана проточната load-store архитектурата на MIPS процесорот, составена од неколку градени блокови: програмски бројач - PC, инструкциски регистар - IR, 32 регистри за општа намена, 32-битна аритметичко-логичка единица - ALU, проточни регистри, посебни инструкциска и податочна кеш меморија, контролна единица (означена со сино), и останата селективна и контролна логика (проширувачи, декодери, мултиплексери, собирачи итн.). Според сл. 26 се покажува дека MIPS процесорот работи само со операнди сместени во неговите регистри, што побарува постојан трансфер на податоци помеѓу меморијата и процесорот преку load и store операции. За да се обезбеди поедноставен пристап и манипулација со податоци, во овој докторски труд се предлага проширување и промена на оригиналната проточна RISC архитектура и креирање на нова проточна MEMRISC процесорска архитектура, дадена на сл. 27.



Слика 26 Проточна RISC архитектура на MIPS процесор, [1].



Слика 27 Проточна MEMRSIC архитектура на MIMOPS процесор.

Според сл. 27 се забележува дека предложениот процесор со MEMRISC архитектура користи посебни локални податочна и програмска меморија во чипот, наместо регистри за општа намена и кеш меморија во чипот. Тоа укажува дека овој процесор ги извршува сите операции со операнди сместени во неговата локална меморија, избегнувајќи ги непотребните и редундантни копирања и движења на податоци, кои се извршуваат во MIPS процесорот, при трансфер (load/store) на податоци. Според тоа, доколку RISC-базираниот MIPS процесор може да извршува милиони инструкции во секунда (акроним MIPS), тогаш предложениот процесор со MEMRISC архитектура би можел да извршува милиони инструкции со мемориски операнди во секунда, па тоа е причината поради која во понатамошниот текст тој е именуван како MIMOPS (**M**illion **I**nstructions on **M**emory **O**perands **P**er **S**econd) процесор.

Предложениот MIMOPS процесор ги исклучува регистрите за општа намена и кеш меморијата од мемориската хиерархија и на тој начин обезбедува директен и едновремен пристап до два изворни и еден резултатен операнд, специфицирани во инструкцијата. Овие операнди се селектираат преку специјална мемориска единица за генерирање на адреси, која овозможува преведување на влезните виртуелни адреси, во физички адреси на локалната меморија во чипот, со примена на страничење. Откако ќе бидат прочитани операндите од локалната податочна меморија во чипот, се извршува операцијата и потоа резултатот се враќа назад во локалната податочна меморија во чипот. Всушност, MIMOPS процесорот работи проточно во 4 фази (земање на инструкција, декодирање на инструкција, извршување и запишување на резултат), исклучувајќи ја MEM фазата и овозможувајќи секоја (аритметичка, логичка, гранење или контролна) MIMOPS инструкција да се извршува во еден такт циклус. Инструкциите кои ги поддржува предложениот MIMOPS процесор, се слични како оние на MIPS, но начинот на нивното интерпретирање и извршување е прилично различен.

За разлика од MIPS процесорот, претставен на сл. 26, предложениот MIMOPS процесор работи директно со меморијата во чипот и на тој начин го поедноставува пристапот до операндите, извршувањето на инструкциите (работи проточно без MEM фаза) и инструкциското множество (не користи load и store инструкции). За да се обезбеди сето тоа, MIMOPS процесорот вклучува специ-

фична контролна единица (обележана со сина боја на сл. 27), која нуди поддршка за неколку адресни режими (пр. директно, непосредно и базно адресирање). Генерално, операндите од меморијата се адресираат директно, при што за преведувањето на виртуелните адреси во физички адреси, MIMOPS процесорот имплементира специфична хардверска поддршка за работа со виртуелната меморија. Ова се однесува на: сегментарање на локалната меморија во чипот и сместување на виртуелни страници во неа, како и имплементирање на табели за преведување и механизми за размена на страници (пр. FIFO).

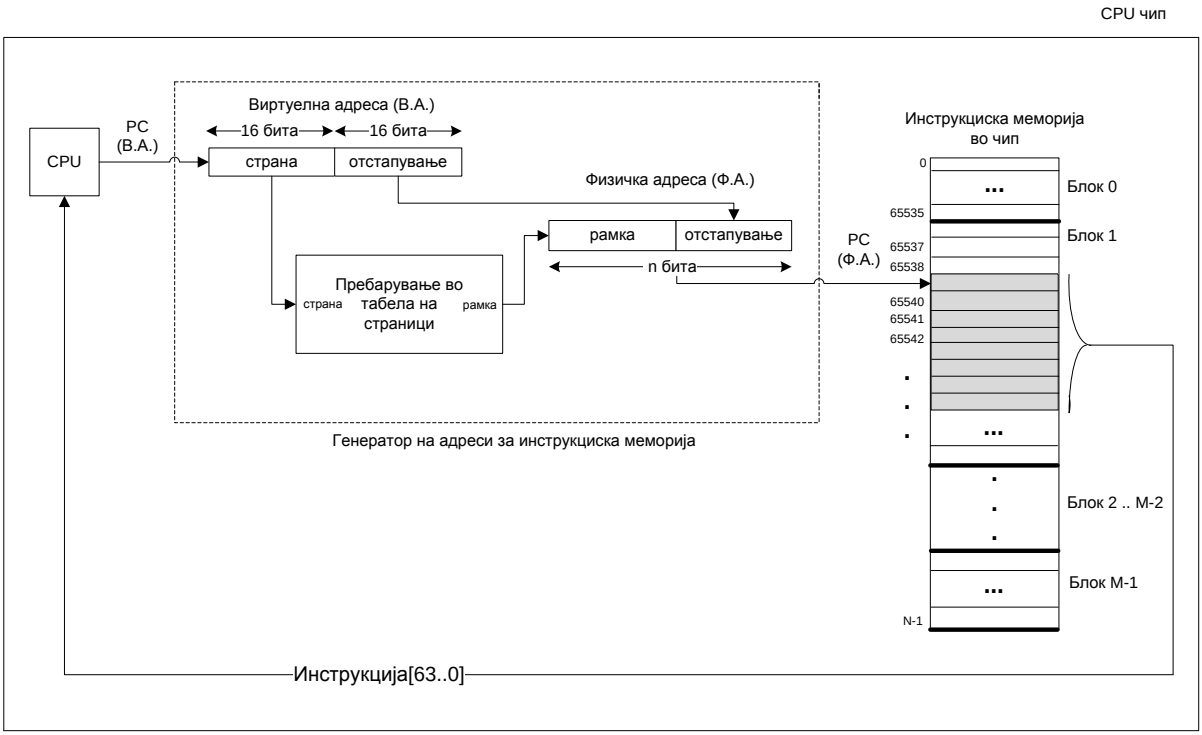
Кога станува збор за проточното работење, може да се истакне дека двата MIPS и MIMOPS процесори обезбедуваат преклопување во извршувањето на инструкциите, кое се постигнува со имплементирање на проточни регистри за секоја меѓу-фаза (пр. fetch/decode). Покрај овие сличности, MIMOPS процесорот во голема мера се разликува од MIPS процесорот, бидејќи овозможува: намалување на бројот на фази во проточното извршување за еден, подранување на извршувањето на условните и безусловни гранења во фазата на декодирање и поддршка за препраќање на операнди со цел да ги надмине податочните зависности при паралелното извршување на инструкции. Дополнително, MIMOPS процесорот имплементира и посебно коло за поместување, кое овозможува генерирање на флексибилен втор операнд за аритметичко-логичката единица (ALU). Ова се постигнува на тој начин што операндот се поместува т.е. шифтира за определена константна вредност пред да биде употребен од страна на ALU единицата (ова е слично како ARM архитектурата, [82]). Според тоа, ALU единица на MIMOPS процесорот може да извршува операции со два влезни цели или реални броеви, при што вториот операнд може да биде претходно поместен.

Целосното функционирање на MIMOPS процесорот се контролира од страна на контролната единица со нејзините контролни сигнали, кои се специфично дефинирани за MMRISC архитектурата. Како дополнение на тоа, MIMOPS процесорот имплементира и контролни механизми за справување со прекини и исклучоци (пр. погрешен резултат), и обезбедува поддршка за комуникација со В/И уреди. Повеќе детали за овие контролни механизми и за останатите најважни архитектурни концепти (инструкциско множество, податочна и контролна патека) на предложениот MIMOPS процесор се дадени во следните поглавја.

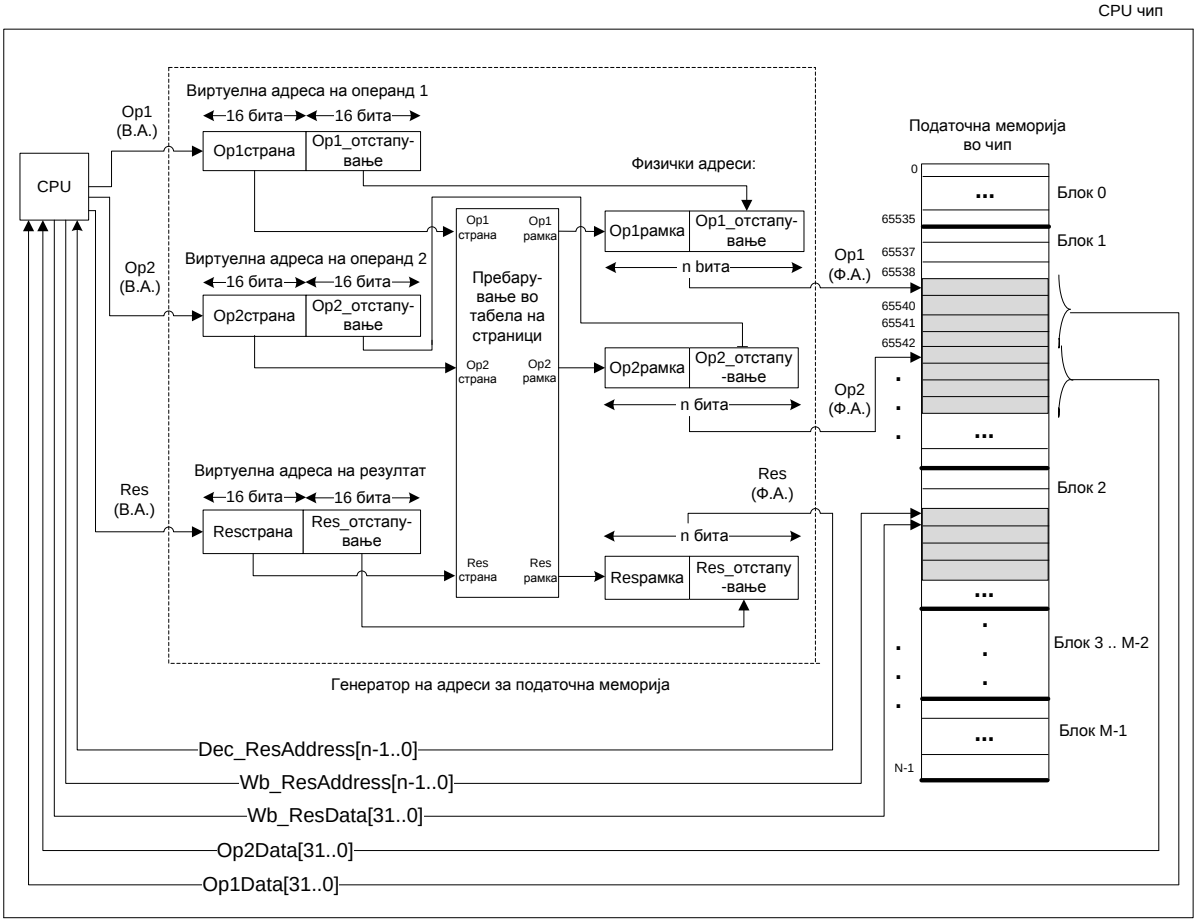
4.1. Хардверска поддршка за работа со виртуелна меморија

Предложениот MIMOPS процесор имплементира посебни инструкциски и податочни мемории во чипот, кои се сегментирани во M физички блокови (за виртуелни страници) со еднаква големина. Секоја од овие локални мемории е организирана како поле од N последователни елементи, со големина од еден бајт и со своја единствена физичка адреса. Процесорот пристапува до својата меморија во чипот со примена на виртуелно адресирање, генерирајќи 32-битна виртуелна адреса, (В.А.), која пред да се испрати до инструкцискатa или податочната меморија во чипот, се конвертира во соодветна физичка адреса, (Ф.А.). Процесот на претоварање на некоја виртуелна мемориска адреса во физичка мемориска адреса се нарекува преведување на адреса.

За да се обезбеди поддршка за преведување на адреси и при тоа да се овозможи едновремен пристап до посебните инструкциски и податочни мемории во чипот, MIMOPS процесорот имплементира две наменски хардверски единици, наречени генератори на адреси за инструкциска и податочна меморија. Овие единици вршат преведување на виртуелните адреси за секое обраќање до локалната меморија, пребарувајќи ги табелите на страници (page tables), кои се сместени во хардверската логика за земање и декодирање на инструкции во процесорот и чија содржина се управува од страна на оперативниот систем. Всушност, во согласност со проточното работење, MIMOPS процесорот обезбедува едновремен пристап до една инструкција од еден блок на инструкцискатa локална меморија преку единицата за генерирање на адреси во инструкцискатa меморија, и до три операнди од најмногу три блокови на податочната локална меморија (некои операнди може да бидат сместени во ист блок) преку единицата за генерирање на адреси во податочната меморија. Конкретно, процесот на преведување на виртуелните адреси, при пристапот до инструкцискатa и податочната меморија во чипот е претставен на сл. 28 и сл. 29, соодветно.



Слика 28 Виртуелно адресирање на инструкциска меморија во чип.



Слика 29 Виртуелно адресирање на податочна меморија во чип.

На сл. 28 е прикажано како процесорот пристапува до локалната инструкциска меморија, во текот на фазата на земање (на инструкција) од проточната линија. Всушност, кога процесорот зема инструкција, тој најпрво ја проследува нејзината виртуелна мемориска адреса, сместена во програмскиот бројач (PC), кон единицата за генерирање на адреси во инструкциската меморија. Оваа единица врши пребарување во табелата на страници за да го најде соодветен број на рамка за влезниот број на страница, и на тој начин да генерира ефективна физичка адреса за побараната инструкција. Веднаш потоа, оваа физичка адреса се проследува до инструкциската меморија преку мемориската магистрала. На тој начин од локалната инструкциската меморија се чита 8-бајтна (64 бита) инструкција, почнувајќи од генерираната физичка адреса, и потоа истата се сместува во инструкцискиот регистар на процесорот. Содржината на овој регистар се испраќа кон контролната единица, која генерира различни контролни сигнали за управување на извршувањето на дадената инструкција во следните фази од проточната линија.

На сл. 29 е прикажано како процесорот пристапува до локалната податочна меморија, во текот на фазата на декодирање (на инструкција) од проточната линија. Веднаш откако процесорот ќе ја декодира инструкцијата, тој испраќа три виртуелни мемориски адреси (за операнд1, операнд2 и резултат) до единицата за генерирање на мемориски адреси во податочната меморија. Оваа единица врши пребарување во табелата на страници за да ги најде соодветните броеви на рамки за влезните броеви на страници, и на тој начин да генерира ефективни физички адреси за двата влезни операнди и резултатниот операнд. Веднаш потоа, физичките адреси на операнд1 и операнд2 се проследуваат до податочната меморија преку мемориската магистрала, додека пак физичката адреса на резултатниот операнд, (Dec_ResAddress), се проследува до следната фаза во проточната линија на процесорот, со цел да биде подоцна употребена во фазата за запишување на резултат.

На сл. 29 се забележува дека процесорот може истовремено да извршува две читања и едно запишување во податочната меморија во чипот. Ова се постигнува на тој начин што процесорот зема два 4-бајтни (32 бита) податоци, почнувајќи од генерираните физички адреси за операнд1 и операнд2, и при тоа

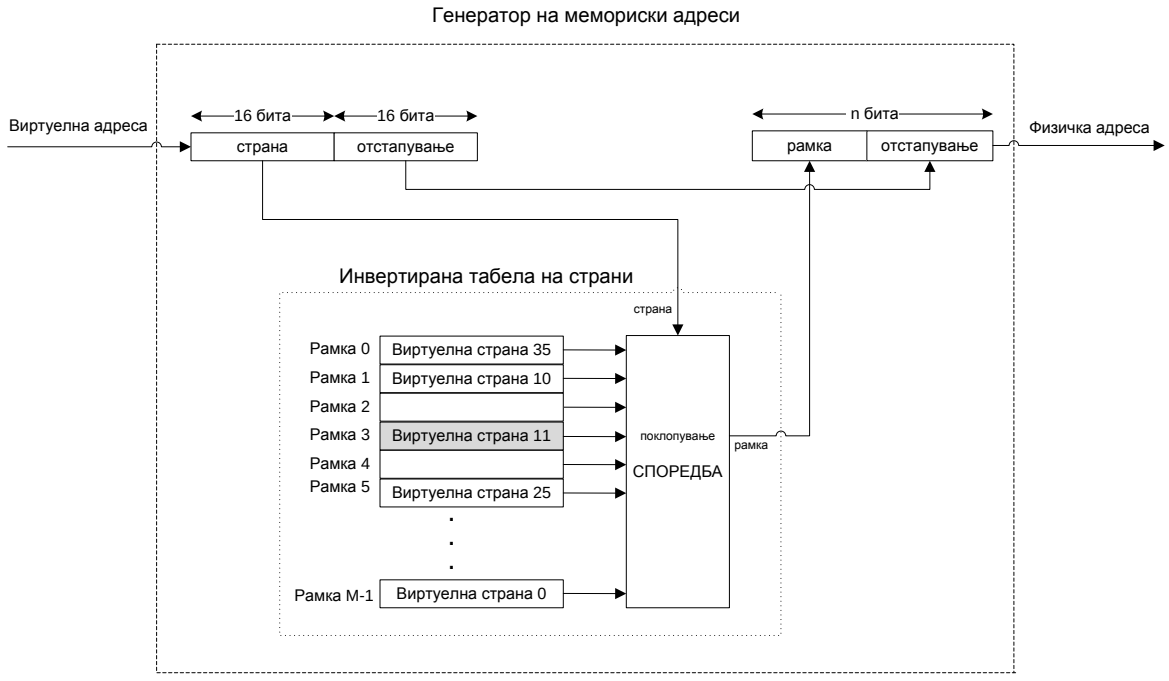
паралелно го запишува проследениот 32-битен резултатен податок, (Wb_ResData), почнувајќи од физичката адреса за резултатниот операнд, (Wb_ResAddress), која исто така е проследена од фазата за запишување на резултат од проточната линија (writeback). Слично на испраќањето на резултатните податок и адреса, (Wb_ResData, Wb_ResAddress), така и прочитаните операнди (операнд1 и операнд2) се проследуваат до следната фаза од проточната линија на процесорот, за да бидат подоцна употребени како влезни операнди при пресметување на некои аритметичко/лигички операции во фазата на извршување.

Предложениот MIMOPS процесор може да работи со 32-битен виртуелен адресен простор со капацитет од 2^{32} бајти (4GB), кој е поделен во виртуелни страници со еднаква големина. И покрај тоа што најчесто употребувана големина на виртуелна страница изнесува 4KB, сепак студиите кои се направени во [83] и [84] предлагаат зголемување на големината на виртуелните страници во опсегот од 16KB до 256KB. Согледувајќи ги претставените резултати во употребената литература, [83], [84], во ова истражување се предлага сегментација на локалната меморија на MIMOPS процесорот во физички рамки (за виртуелни страници) со големина од 64KB. Всушност, се зема во предвид дека големината на виртуелна страница од 64KB обезбедува ниска рата на промашување за повеќе од анализираните тест сценарија во [83] и воедно влијае да се намали големината на табелата на страници, па се очекува дека предложеното ограничување во големината на виртуелната страница, би овозможило да се намали бројот на пристапи надвор од чипот, што пак би влијаело да се подобрат перформансите на MIMOPS процесорот, (при извршување на различни програми). Дополнително, се подразбира и дека MIMOPS процесорот би обезбедил поддршка за вградување на табели на страници со мал капацитет, внатре во чипот (во фазите за земање и декодирање на инструкции).

Кога станува збор за големината на меморијата во чипот, може да се каже дека тоа најмногу зависи од применетата технологија (пр. SRAM, вграден DRAM) и физичките ограничувања на чипот. Како резултат на тоа, двете мемории во чипот (инструкциска и податочна), претставени на сл. 28 и сл. 29, се дефинирани со капацитет од $N=2^n$ бајти, и се сегментирани во $M=2^m$ физички рамки за

64KB=2^{16}B виртуелни страници, каде n=m+16. Овие параметри би можеле да се определат попрецизно за 45nm SOI технологија, во согласност со резултатите кои се дадени во [30], (и прикажани на сл. 7), кои укажуваат дека во еден чип би можело да се имплементира 1 - 64Mb локална меморија со доцнење од 0,8 - 2,5 ns, со примена на вграден SRAM или DRAM, соодветно. Како и да е, овие вредности би можеле да се разликуваат за други технологии на имплементација.

Со оглед на тоа дека табелите на страници се исто така имплементирани внатре во процесорскиот чип, тие додаваат дополнителна комплексност во дизајнот на процесорот. Земајќи во предвид дека една стандардна табела на страници за 4GB виртуелен мемориски систем (сегментиран во 64KB страници) содржи 2^{16} записи, што е многу захтевно, MIMOPS процесор имплементира посебен тип на инвертирани табели на страници, [17]. Овие табели имаат по еден запис за секоја физичка рамка, наместо за секоја виртуелна страница, и на тој начин тие наместо 2^{16} содржат M записи, каде што $M<<2^{16}$. На сл. 30 е прикажано како изгледа една инвертирана табела на страници, која се користи внатре во единиците за генерирање на адреси во инструкциска или податочна меморија во MIMOPS процесорот.

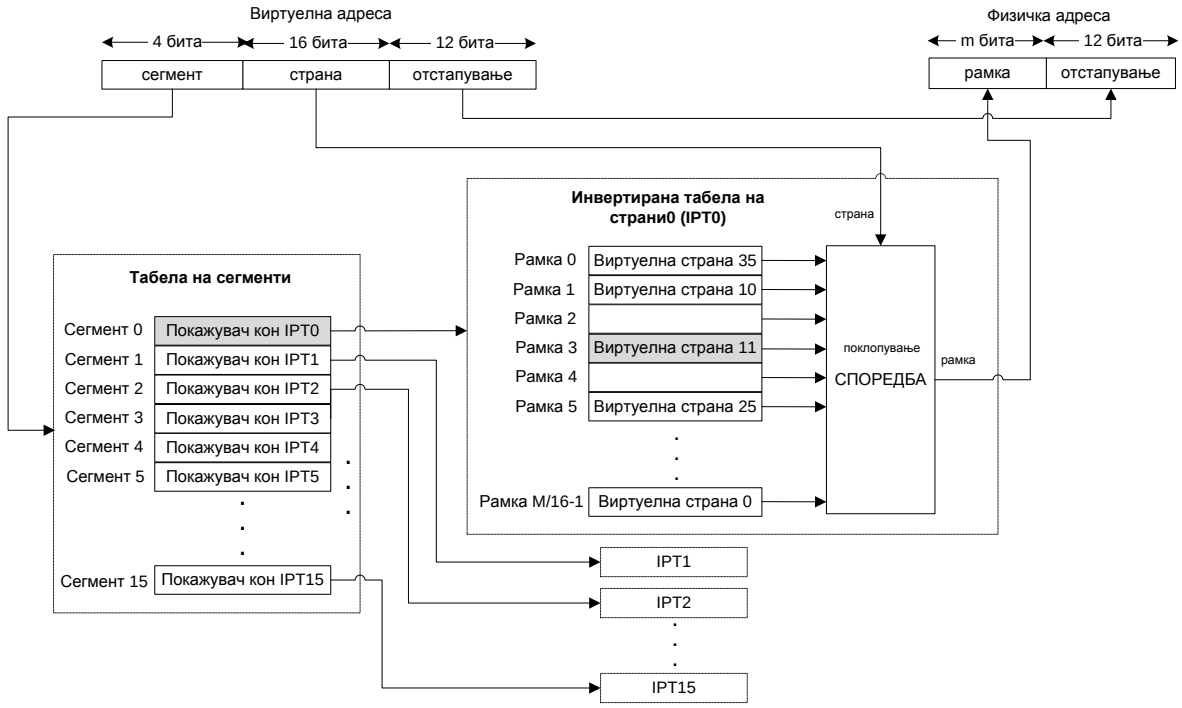


Слика 30 Инвертирана табела на страници во генератор на мемориски адреси.

Според сл. 30 се забележува дека секој запис во инвертираната табела на страници содржи број на виртуелна страница, наместо број на физичка рамка. Тоа значи дека бројот на физичка рамка не се зачувува, бидејќи индексот во табелата соодветствува на него. Според тоа, за да се преведе една виртуелна адреса со помош на инвертирана табела на страници, влезниот број на виртуелна страница се споредува со секој запис во табелата, при што полето од записи се изминува секвенцијално. Единицата за генерирање на адреси во податочната меморија го повторува овој процес три пати, за секој од трите влезни броеви на виртуелни страници (за операнд1, операнд2 и резултат) во инвертираната табела на страници. Откако ќе заврши изминувањето на полето со записи и при тоа ќе настане поклопување, бројот на виртуелна страница од виртуелната адреса се заменува со индексот за даденото поклопување и на тој начин се генерира физичката адреса. Во случај доколку нема поклопување, се генерира промашување (page fault) кое процесорот го опслужува како прекин, предавајќи ја контролата на оперативниот систем. За таа цел, предложениот MIMOPS процесор имплементира хардверски компоненти (пр. бројачи, бафери, контролни сигнали), кои обезбедуваат поддршка за FIFO размена на виртуелни страници, и го контролираат извршувањето на промени во табелите на страници.

И покрај тоа што во ова истражување се предлага употреба на виртуелни страници со големина од 64KB, сепак во продолжение е претставена дво-нивовска организација на табелите на страници, со цел да се овозможи поддршка за виртуелни страници со помал капацитет. Конкретно, на сл. 31 е прикажана дво-нивовската структура на табелите на страници, кои се користат во случај кога MIMOPS процесорот работи со стандардни виртуелни страници со капацитет од 4KB. На сл. 31 се забележува дека виртуелниот мемориски адресен простор е поделен во 16 сегменти, кои содржат 2^{16} страници, секоја со капацитет од 4KB. Од друга страна, двете локални физички мемории се поделени во $M=2^m$ физички рамки за $4KB=2^{12}B$ виртуелни страници, каде $m<=16$. Според тоа, вкупниот капацитет на инструкциската или податочната локална меморија во чипот е <= 256MB.

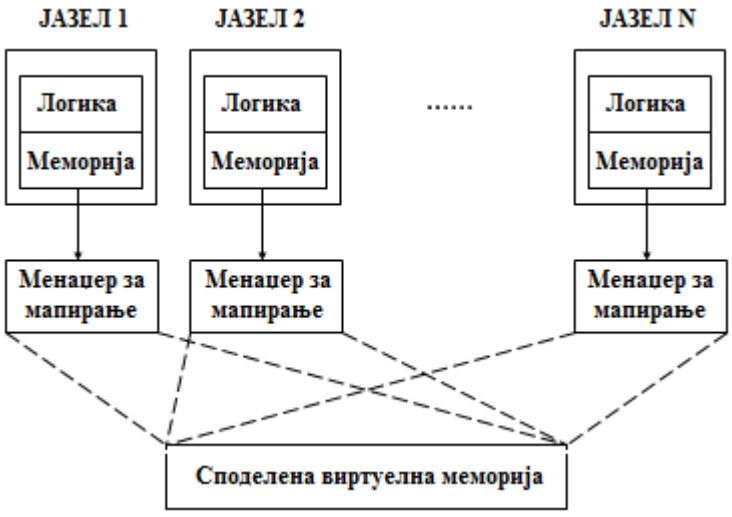
Предложената дво-нивовска структура на табелата на страници, дадена на сл. 31, содржи табела на сегменти со 16 записи т.е. покажувачи кон инвертирани табели на страници за секој сегмент, и 16 инвертирани табели на страници со M/16 записи. Според тоа, преведувањето на виртуелна во физичка адреса со предложената дво-нивовска табела на страници се извршува во два чекора. Во првиот чекор се внесува бројот на сегмент од виртуелната адреса во табелата на сегменти, и потоа се генерира покажувач, кој врши селекција на соодветна инвертирана табела на страници (inverted page table - IPT) за дадениот сегмент. Во вториот чекор се внесува бројот на страница од виртуелната адреса во селектираната IPT табела, и потоа кога ќе настане поклопување во табелата се генерира бројот на физичка рамка (индекс за поклопување во IPT табела). Веднаш потоа, на генерираниот број на физичка рамка се додава отстапувањето од виртуелната адреса и на тој начин се креира физичката адреса. И покрај тоа што претставеното преведување на виртуелна во физичка адреса се однесува на мемориски систем кој работи со 4KB страници, сепак претставениот пристап на дво-нивовско организирање на табелите на страници може да се прилагоди за други големини на страници, со менување на бројот на битови доделени за страница и отстапување во виртуелната адреса



Слика 31 Дво-нивовска организација на табела на страници.

Последните трендови во дизајнот на процесорите укажуваат дека сè поголем и поголем дел од површината на чипот ќе биде наменета за меморија. Всушност, голем дел од површината на модерните процесорски чипови веќе се употребува за имплементирање на врвот од мемориската хиерархија, кој вклучува: регистри за општа намена и едно или две нивоа на кеш меморија. Сметајќи дека овој тренд ќе продолжи и дека јазот во мемориското доцнење помеѓу меморијата во и надвор од чипот ќе се зголемува, може да се јави потреба за интегрирање на целата физичка меморија во чипот. На тој начин, обраќањата надвор од чипот ќе се третираат повеќе како промашувања во виртуелни страници, отколку како промашувања во кешот. Дури и доколку целосната физичка меморија не може да се смести во еден процесорски чип, би можело да се изгради повеќе-процесорски систем со дистрибуирана меморија. На овој начин, повеќе споени мемориско-пресметковни чипови би креирале хомоген дистрибуиран систем, кој би можел лесно да се проширува и надградува. Слични пристапи на сметање се имплементирани во неколку мемориско-центрични системи, како што се: Diva, [40], паралелен процесирачки RAM, [44], или DataScalar, [45].

Кога станува збор за дистрибуирано сметање, предложениот MIMOPS процесор би можел да се употреби во систем со дистрибуирана споделена виртуелна меморија, даден на сл. 32, а подетално дискутиран во [85]. Во таков систем виртуелната меморија претставува единствен адресен простор кој што го споделуваат повеќе процесори, така што било кој процесор може директно да пристапи до било која мемориска локација од адресниот простор.



Слика 32 Систем со дистрибуирана споделена виртуелна меморија, [85].

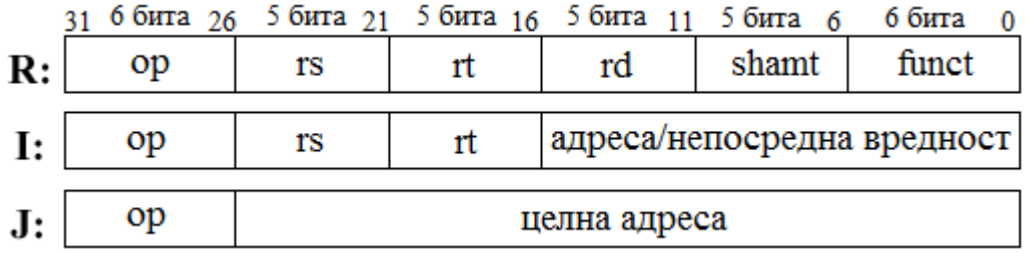
Според сл. 32, мапирањето помеѓу локалните мемории и споделената виртуелна меморија се извршува преку менаџери за мапирање, кои покрај мапирање имаат за задача да го одржуваат адресниот простор кохерентен. Секој менаџер за мапирање ја гледа својата локална меморија како голем кеш за споделениот виртуелен мемориски простор, за својот процесор. Всушност, споделената виртуелна меморија постои само виртуелно, исто како и традиционалната виртуелна меморија.

Споделениот виртуелен мемориски простор е поделен во страници. Страниците кои се обележани само за читање можат истовремено да имаат копии во физичката меморија на повеќе процесори. Од друга страна, страниците кои се обележани за запишување можат во даден момент да бидат само во физичката меморија на еден процесор. При обраќање до меморијата може да настане промашување, кога страницата која ја содржи побараната мемориска локација не се наоѓа во тековната физичка меморија на процесорот. Во таков случај, менаџерот за мапирање ја превзема страницата или од дискот, или од меморијата на друг процесор. Сепак, доколку обраќањето до меморијата резултира со запишување, менаџерот за мапирање треба да гарантира атомичност на операцијата. За таа цел, после запишувањето, останатите копии се поништуваат или ажурираат.

Во еден систем со споделена виртуелна меморија, паралелните програми извршуваат множество на процеси (или нитки), кои споделуваат еден заеднички виртуелен адресен простор. Всушност, една од главните цели на споделената виртуелна меморија е да овозможи процеси (или нитки) од програми да се извршуваат паралелно на различни процесори. Според тоа, перформансите на една паралелна програма во систем со споделена виртуелна меморија најмногу би зависеле од бројот на паралелни процесори и степенот на делење на податоци (конфликтни ситуации). Генерално, перформансите се подобруваат со зголемување на бројот на процесори и воедно извршување на програми кои се карактеризираат со локалност во обраќањата (резултира со помалку конфликти), [85]. И покрај тоа што мемориските обраќања во паралелните програми се различни од секвенцијалните, сепак еден процес претставува секвенцијална програма која би требало да има висок степен на локалност. Како и да е, постоењето на конфликти најмногу би зависело од типот на алгоритамот кој се извршува.

4.2. Архитектура на инструкциско множество

MIMOPS процесорот имплементира RISC-базирана мемориско-центрична архитектура, наречена MEMRISC архитектура. Оваа архитектура опишува процесор, кој обезбедува директно безрегистарско работење со посебни инструкциски и податочни мемории, сместени внатре во процесорскиот чип. Според тоа, овој процесор ги извршува сите операции со вредности сместени во локалната меморија во чипот, исклучувајќи ја употребата на регистри за општа намена и кеш меморија. Од друга страна, MIPS процесорот, кој беше употребен како основа за имплементирање на новиот MIMOPS процесор, користи стандардна RISC-базирана load/store архитектура, каде до главната меморија се пристапува само со load и store инструкции, а пак сите останати инструкции користат регистри за општа намена како операнди. Тоа укажува дека сите операции во MIPS процесорот се извршуваат врз податоци кои се сместени во некој од неговите 32 локални регистри за општа намена. Всушност, MIPS процесорот може да извршува 32-битни инструкции со фиксна должина, организирани во 3 различни категории, кои се прикажани на сл. 33.



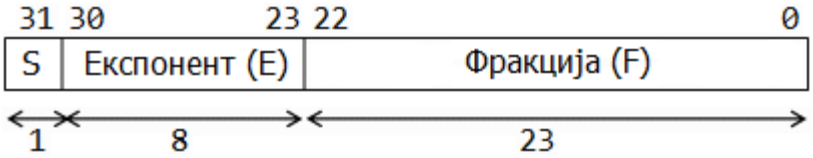
Слика 33 Формат на инструкции на MIPS процесор, [1].

Според сл. 33, полињата во MIPS инструкциите се следните: op, 6-битен код на операција; rs, rt и rd, 5-битни спецификатори за прв изворен, втор изворен и дестинациски регистар; shamt, 5-битна вредност за поместување, која се користи само во (shift) инструкции за поместување; funct, 6-битен код кој се користи за да ја специфицира функцијата која ја селектира варијантата на операцијата дадена со полето "op"; адреса/непосредна вредност, 16-битна непосредна вредност со знак која се користи за логичко/аритметички операции или како адресно отстапување во load/store инструкции; и целна адреса, 26-битна адреса на дестинација при безусловен скок. MIPS инструкциите користат

различни кодирања и на тој начин можат да извршуваат различни операции со операнди кои се адресирани со примена на некои од следните адресни режими: регистарски, непосредно, базно (за load/store инструкции), релативно во однос на PC (за условни скокови), регистарски директно (или индиректно - за скок преку регистри), или псевдо директно (за безусловни скокови), [1]. Со оглед на тоа дека сите операции се извршуваат со регистрите на процесорот, инструкциското множество на MIPS процесорот е организирано во три различни категории: R-тип, I-тип и J-тип. Оттука, R-тип се користи за аритметичко/логички инструкции, I-тип се користи за load/store операции, условни скокови или аритметичко /логички инструкции, и J-тип се користи за инструкции за безусловен скок.

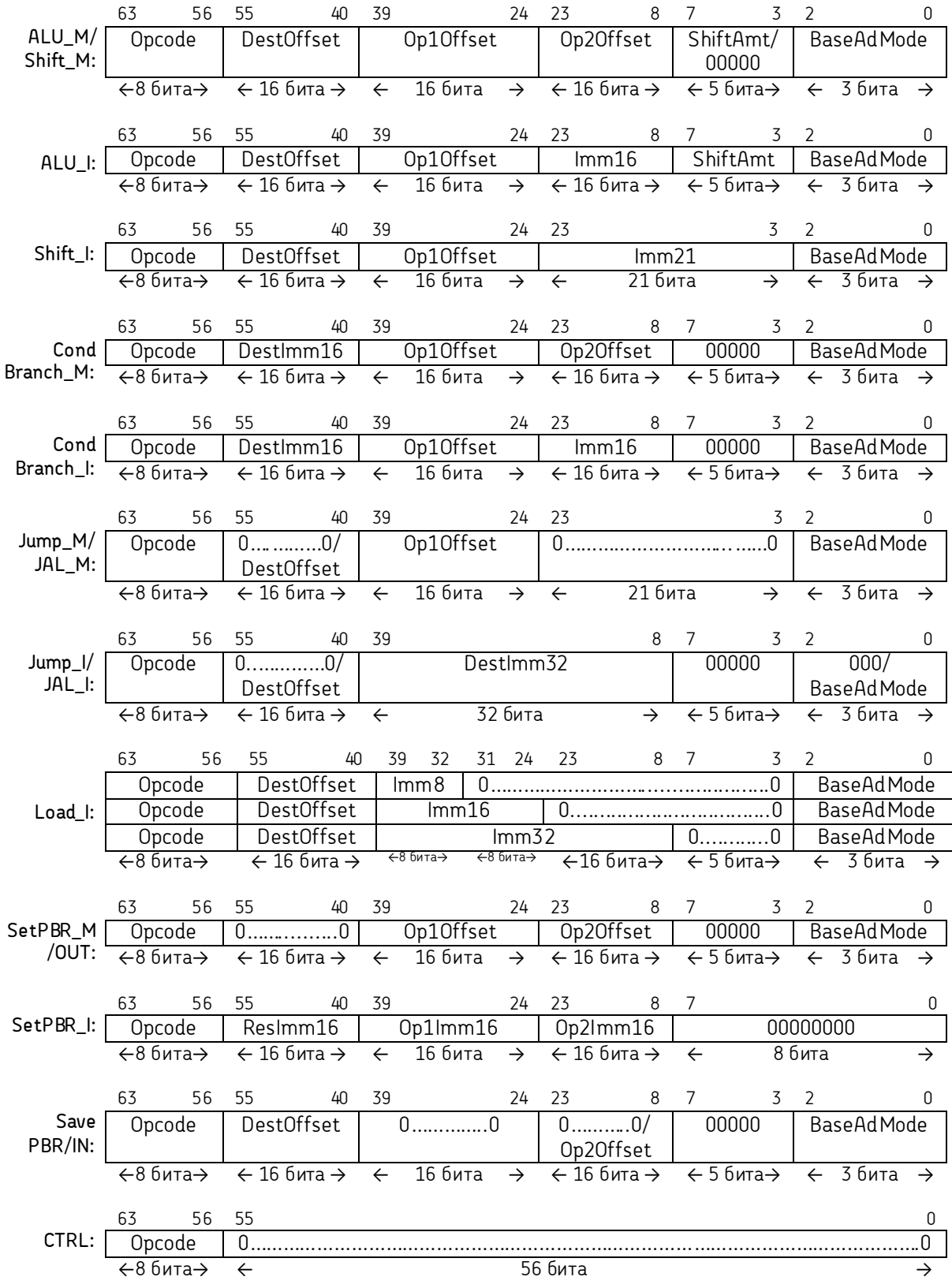
За разлика од претставените инструкциски формати за MIPS процесорот, кои во главно се наменети за регистер-до-регистер операции, предложениот MIMOPS процесор работи директно со локалната меморија во чипот, исклучувајќи ги регистрите за општа намена, па затоа тој побарува дефинирање на свое специфично инструкциско множество. Тоа би вклучувало спецификација на формат на инструкции, адресни режими за пристап до операнди и опис на дополнителни внатрешни процесорски ресурси за селекција на операнди во локалната меморија која имплементира страничење (базни регистри, регистри за виртуелни страници). Генерално, функционалноста на предложениот MIMOPS процесор би наликувала на онаа на MIPS процесорот, овозможувајќи извршување на RISC-базирани стандардни аритметичко/логички операции, споредби, поместувања, и условни и безусловни скокови, но при тоа вклучувајќи и дополнителни инструкции за работа со единиците кои се специфично дефинирани за овој процесор.

Кога станува збор за форматот на податоци кои се користат во MIMOPS процесорот може да се каже дека тој поддржува работа со 32-битни цели броеви со знак, претставени во двоен комплемент, или 32-битни реални броеви, претставени во IEEE-754 формат со единечна прецизност, [86]. Всушност, на сл. 34 е прикажан IEEE-754 форматот со единечна прецизност, според кој секој реален број може да се претстави со: 1 бит за знак, 8 бита за експонент даден во вишок-127 нотација, и 23 бита за децимален дел (фракција). Дополнително, MIMOPS процесорот поддржува и работа со 16-битни непосредни вредности, претставени во IEEE-754 формат со половична прецизност, [86], како реални броеви.



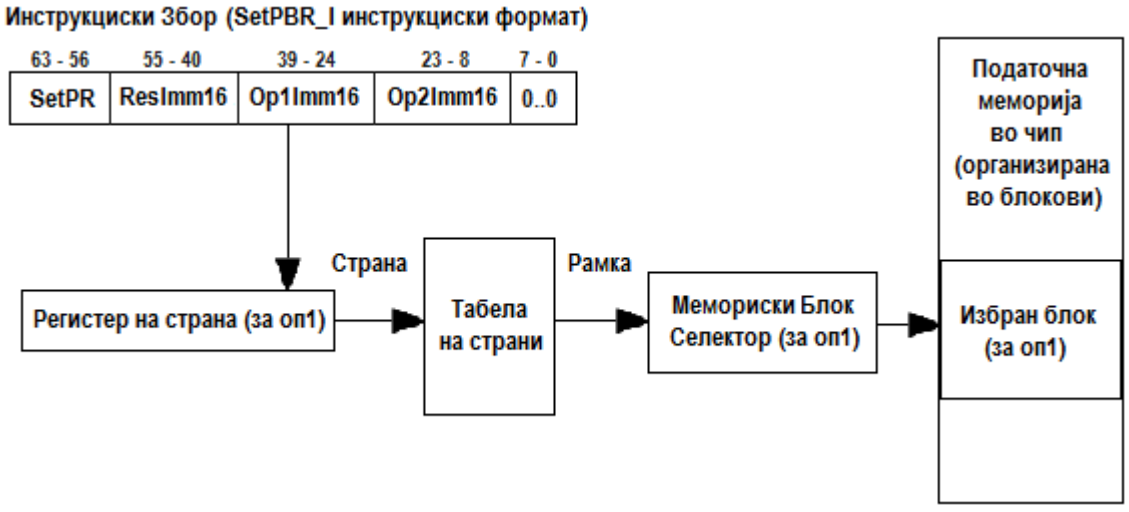
Слика 34 IEEE-754 формат со единична прецизност за броеви со подвижна запирка, [86].

MIMOPS процесорот имплементира инструкциско множество со 64-битни инструкции со фиксна должина, кои се едноставни и наликуваат на оние кои се користат во RISC-базираните процесори. Повеќето од овие инструкции извршуваат меморија-до-меморија операции, обезбедувајќи директен пристап до вредностите кои се сместени во податочната меморија во чипот. Всушност, овие инструкции се претставени со три-адресен формат, каде две полиња се користат за селектирање на изворните операнди од меморијата во чипот, а пак третото поле се користи за селекција на дестинацијата за запишување на резултатот во меморијата во чипот, каде се складира резултатот од извршената операција. Ова во главно се однесува на аритметичко-логички или споредбени операции, со цели или реални броеви. Дополнително, MIMOPS процесорот имплементира и други 64-битни инструкциски формати со цел да обезбеди поддршка за базно и непосредно адресирање, и да овозможи извршување на гранења, гранења со зачувување на состојба на адресата на следната инструкција (JAL - jump and link) и враќање назад (return), вчитување на константи, или поместувања, при тоа работејќи директно со меморијата во чипот. Од друга страна, MIMOPS процесорот го проширува своето инструкциско множество со некои специфични инструкции кои се користат за управување на процесорските ресурси кои се имплементираат само во овој тип на процесори со MEMRISC архитектура. Ова се однесува на: базните регистри, регистрите за виртуелни страници и табелите на страници, кои се дел од MIMOPS процесорот. Дополнително, MIMOPS процесорот обезбедува поддршка и за некои контролни инструкции, како што се немање на операција (no operation) или стопирање на работата на процесорот (halt), а воедно имплементира и комуникација со В/И уреди, овозможувајќи пренос на податоци од/во инструкциската или податочната меморија во чипот. Сумирајќи го сето тоа, на сл. 35 се прикажани сите категории на различни инструкциски формати, кои се поддржани од MIMOPS процесорот.



Слика 35 Формат на инструкции на MIMOPS процесор.

Со оглед на тоа дека инструкциската и податочната меморија во чипот на MIMOPS процесорот се сегментирани во физички блокови (за виртуелни страници), MIMOPS процесорот во даден момент може паралелно да работи со еден инструкциски мемориски блок и најмногу три податочни мемориски блокови (два за изворните операнди и еден за резултатот). Селекцијата на соодветните мемориски блокови се врши со посебна SetPR (set page registers) инструкција која спаѓа во категориите на инструкции: SetPBR_M или SetPBR_I, прикажани на сл. 35. SetPR инструкцијата ја ажурира содржината на регистрите на страници (за операнд1, операнд2 и резултат), кои се сместени внатре во генераторот на адреси за податочната меморија во MIMOPS процесорот. Според сл. 35, вредноста на регистрите на страници може да се прочита од локалната податочна меморија во чипот (Op1Offset, Op2Offset) кога SetPR инструкцијата имплементира директно адресирање (SetPBR_M инструкциски формат), или може да се постави со 16-битни непосредни константи (Op1Imm16, Op2Imm16 и ResImm16) кога SetPR инструкцијата користи непосредно адресирање (SetPBR_I инструкциски формат). Откако ќе се постават регистрите на страници, следниот чекор на MIMOPS процесорот е да изврши пребарување во табелата на страници и при поклопување да ги постави мемориските блок селектори и со нив да ги избере физичките блокови (за операнд1, операнд2 и резултат) од податочната меморија во чипот. На пример, на сл. 36 е прикажано како се селектира мемориски блок за првиот операнд (операнд1). Истиот процес на избирање мемориски блокови се извршува истовремено за другиот изворен операнд (операнд2) и резултатниот операнд.



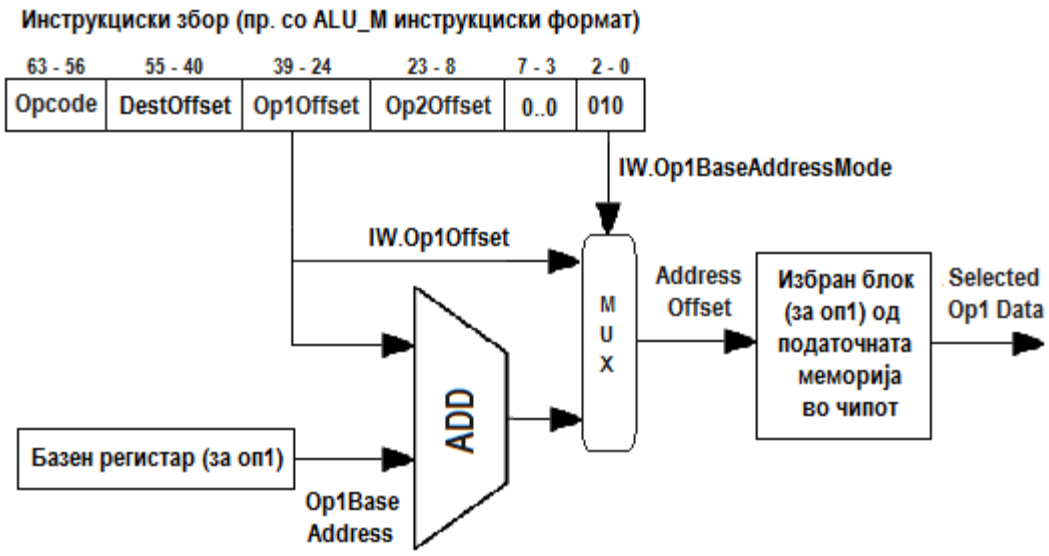
Слика 36 Селекција на физички блок од податочна меморија во чип, за прв изворен операнд.

Пристапот на сегментирање на меморијата во чипот резултира со некои поедноставувања во форматирањето на инструкциите, бидејќи адресирањето на податоците при пристап до податочната меморија во чипот се одвива во два чекора. Најпрво се врши селекција на мемориските блокови (за операнд1, операнд2 и резултат) со употреба на посебната SetPR инструкција, па потоа во следната инструкција (пр. некоја аритметичко-логичка операција од ALU_M категорија) директно се специфицираат операндите како 16-битни адресни отстапувања на селектираните блокови (Op1Offset, Op2Offset, ResOffset), наместо целосни 32-битни адреси. На тој начин се овозможува едновремено обраќање до трите операнди од селектираните блокови во податочната меморија во чипот. Овој пристап се применува во повеќето инструкциски формати прикажани на сл. 35 и неговата намена е да ја скрати должината на инструкциите (од 128 на 64 бита) и да ја намали големината на програмите, во случај кога постои голема просторна и временска локалност при пристапите до меморија. Од друга страна, овој предлог претставува подобрување на без-регистарската PERL архитектура, која користи долги 128-битни инструкции, каде операндите се специфицирани со комплетни 32-битни адреси, што пак според [33] беше претставено како недостаток во PERL архитектурата.

Според сл. 35, може да се види дека секоја MIMOPS инструкција е 64-бита долга и вообичаено содржи: 8-битен код на операција и операнди кои се специфицирани како директни адресни отстапувања (Op1Offset, Op2Offset, ResOffset) или непосредни вредности (Imm8, Imm16, Imm21, Imm32). Според тоа, MIMOPS процесорот може да имплементира 2^8 = 256 инструкции, кои поддржуваат различни операции и адресни режими. Кога станува збор за непосредно адресирање може да се каже дека MIMOPS процесорот обезбедува поддршка за работа со 8-битни (Imm8), 16-битни (Imm16), 21-битни (Imm21) и 32-битни (Imm32) непосредни вредности. Всушност, Imm16 константата се проширува до 32 бита и потоа се користи како втор операнд во ALU операции или при условни гранења, додека пак Imm21 константата се проширува до 32 бита и потоа се користи како втор операнд во операции за поместување. Дополнително, Imm8, Imm16 и Imm32 непосредните вредности се користат при вчитување на 8-битни, 16-битни и 32-битни константи во податочната меморија, соодветно. Ова вообичаено се

извршува со инструкции, кои се дадени во Load_I инструкциски формат, прикажан на сл. 35. На истата сл. 35 може да се забележи и дека повеќето инструкции го користат последниот бај за имплементирање на две дополнителни полиња: вредност на поместување (ShiftAmt) и режим на базно адресирање (BaseAdMode). Вредноста на поместување претставува 5-битно поле кое специфицира константна вредност за која се поместува вториот операнд, пред да се изврши некоја ALU операција (дадена во ALU_M или ALU_I инструкциски формат). Како резултат на тоа, вториот операнд при извршување на ALU операции, уште се нарекува флексибилен втор операнд. Од друга страна, режимот на базно адресирање претставува 3-битно поле кое се користи за да определи дали некој од операндите (операнд1, операнд2 и резултат) користи базно адресирање.

Базното адресирање претставува уште еден адресен режим кој е поддржан од MIMOPS процесорот со цел да овозможи поедноставна работа и манипулација со полиња. Овој режим на адресирање се поставува со последните три бита во инструкцијата, кои означуваат кој од операндите (операнд1, операнд2 и резултат, соодветно) користи базно адресирање. Според тоа, доколку полето за режим на базно адресирање е поставено на 111, тогаш тоа значи дека сите три операнди користат базно адресирање. За да се обезбеди поддршка за базно адресирање, MIMOPS процесорот имплементира три посебни базни регистри (за операнд1, операнд2 и резултат) внатре во единицата за генерирање на адреси во податочната меморија. Покрај тоа, MIMOPS процесорот користи и посебна SetBR инструкција (со слична синтакса на SetPR инструкцијата и ист инструкциски формат: SetPBR_M или SetPBR_I) за да ги постави иницијалните вредности на адресите во базните регистри. Според тоа, доколку некој операнд има поставен бит на единица во полето за режим на базно адресирање, тогаш неговата адреса се пресметува со собирање на вредноста од неговиот базен регистер со неговото адресно отстапување специфицирано во инструкцијата, како што е прикажано на сл. 37. Според сл. 37 може да се забележи дека само првиот изворен операнд од дадената инструкција која е претставена со ALU_M инструкциски формат користи базно адресирање. Сепак, во случај доколку сите три операнди применат базно адресирање, тогаш паралелно треба да се извршат три собирања, за секој операнд поединечно.

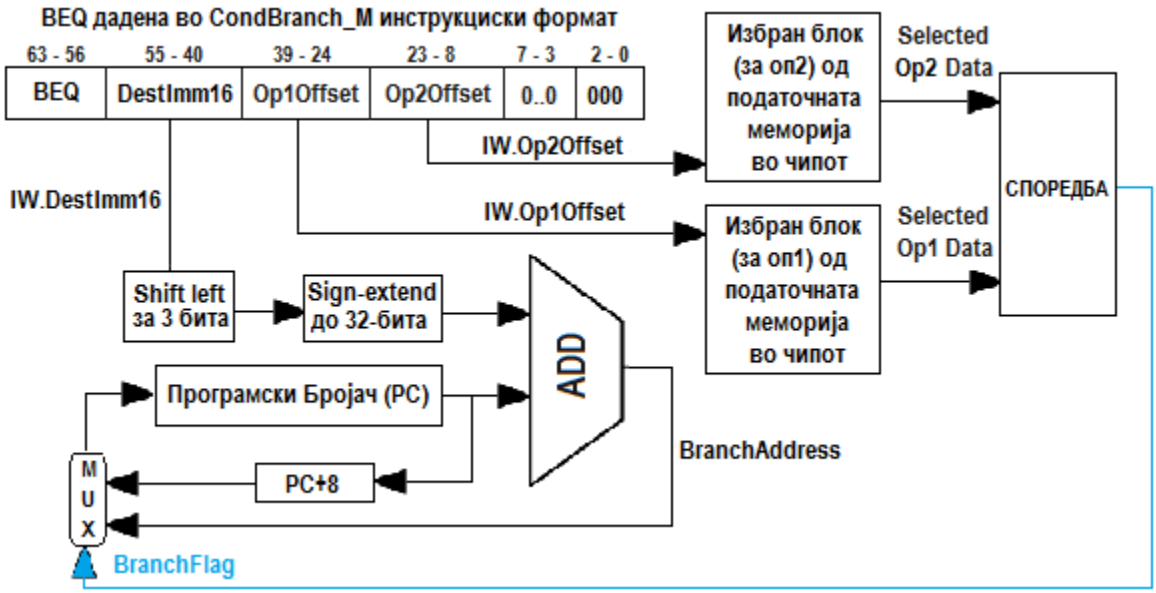


Слика 37 Пристап до прв изворен операнд од селектиран блок во меморија на чип, во случај кога операндот користи базно адресирање.

Покрај базно, директно и непосредно адресирање на операнди, MIMOPS процесорот поддржува и: релативно адресирање во однос на PC (програмски бројач) наменето за условни гранења и директно адресирање во однос на PC наменето за безусловни скокови, или гранења со зачувување на состојба на адресата на следната инструкција (jump and link). Овие инструкции за гранење вообичаено вклучуваат 16-битна DestImm16 или 32-битна DestImm32 непосредна вредност, или директно специфицираат Op1Offset адресно отстапување, како што е прикажано на сл. 35. Всушност, DestImm16 константата се користи за условни скокови (CondBranch_M или CondBranch_I инструкциски формати), додека пак DestImm32 константата и Op1Offset адресното отстапување се користат за безусловни скокови, или гранења со зачувување на состојба на адреса на следна инструкција (Jump_I и JAL_I, или Jump_M и JAL_M инструкциски формати).

Релативното адресирање во однос на PC користи 16-битна непосредна вредност, која го зголемува опсегот на гранење за фактор од осум, во однос на вредноста на програмскиот бројач. Според тоа, инструкциите на условно гранење вршат поместување за три места во лево (= множење со осум) на 16-битната DestImm16 константа и потоа го прошируваат добиениот резултат до 32-бита за истиот да го употребат за ажурирање на вредноста на PC, како што е прикажано на сл. 38 за BEQ (branch on equal) инструкцијата. Од друга страна, директното адресирање во однос на PC користи 32-битна непосредна вредност или директно

адресирана 32-битна вредност за да ја пресмета адресата на гранење. Всушност, 32-битната DestImm32 константа се поместува за три места во лево и потоа се користи за да се постави вредноста на PC, при извршување на скокови дадени во Jump_I или JAL_I инструкциски формати. На сличен начин, скоковите кои се дадени во Jump_M или JAL_M инструкциски формати ја поместуваат за три места во лево директно адресираната 32-битна вредност до која се пристапува преку Op1Offset адресното отстапување и потоа го користат добиениот резултат за да се постави вредноста на PC.



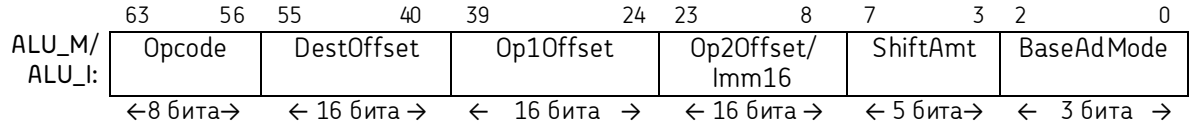
Слика 38 Примена на релативно адресирање во однос на PC во BEQ условно гранење.

На сл. 38 е претставено извршувањето на BEQ (скок при еднаквост) инструкција, дадена во CondBranch_M инструкциски формат. Оваа инструкција извршува гранење во случај кога двата влезни операнди кои се прочитани од податочната меморија во чипот, се еднакви. Доколку овој услов е исполнет, тогаш се поставува BranchFlag битот и се активира мултиплексер кој обезбедува ажурирање на програмскиот бројач со адресата на гранење (BranchAddress), која е пресметана со примена на релативно адресирање во однос на PC за непосредната вредност DestImm16. Од друга страна, доколку условот за еднаквост не е исполнет, тогаш програмскиот бројач се поставува да покажува кон следната инструкција од програмата која треба да се изврши. На сличен начин, MIMOPS процесорот може да извршува скокови и за други услови, како што се: нееднак-

вост (!=), помало од (<), помало или еднакво (<=), поголемо од (>), поголемо или еднакво (>=), итн.

Различните инструкциски формати кои се поддржани од MIMOPS процесорот беа претходно претставени на сл. 35, додека пак во продолжение е дадена подетална дискусија за инструкциското множество на MIMOPS процесорот. Генерално, инструкциското множество на MIMOPS процесорот наликува на она на RISC процесорите, и истото вклучува повеќе различни инструкциски формати, организирани во седум функционални групи на инструкции, вклучувајќи: аритметичко-логичка група, група за поместување, група за гранење, група за вчитување на константи, група за работа со регистри за страници и базни регистри, група за В/И комуникација и дополнителна група за контролни инструкции.

Инструкциите кои припаѓаат во аритметичко-логичката група можат да бидат дадени во ALU_M или ALU_I инструкциски формати, прикажани долу:



Според дадените ALU формати, може да се забележи дека ALU инструкциите се дефинирани со код на операција и три операнди кои можат да бидат адресирани со примена на директно или базно адресирање, или дури и непосредно адресирање за вториот изворен операнд во случај кога се користи ALU_I инструкциски формат. 8-битниот код на операција е поделен во 6-битно ALUOperation поле, кое ја дефинира ALU операцијата која треба да се изврши, и 2-битно Shift-Operation поле кое ја специфицира операцијата на поместување која треба да се изврши врз вториот изворен флексибилен операнд (во случај кога ShiftAmt полето има вредност различна од нула) пред да се изврши дадената ALU операција.

Извршувањето на ALU инструкцијата започнува во фазата за декодирање од проточната линија, кога изворните операнди се читаат од податочната меморија во чипот. Веднаш потоа, во фазата за извршување од проточната линија се врши поместување на вториот изворен флексибилен операнд со посебна компонента за поместување, па двата изворни операнди се испраќаат до ALU единицата каде се извршува ALU операцијата. На крај, во фазата за запишување на резултат од проточната линија, резултатот од ALU операцијата се запишува

назад во податочната меморија во чипот на резултатната адресата која е претходно декодирана од ALU инструкцијата. Всушност, ова однесување (пристап до операнди, извршување на ALU операцијата и запишување на резултат) може да се интерпретира на следниот начин:

```
Op1 = DataMem(Op1Offset + Op1BaseRegister)
// Op1 е директно адресиран; Op1Baseregister се користи кога BaseAdMode[1]==1;

Op2 = DataMem(Op2Offset + Op2BaseRegister) ShiftOperation ShiftAmt
// Op2 е директно адресиран; Op2Baseregister се користи кога BaseAdMode[0]==1;
или
Op2 = SignExtendTo32(Imm16) ShiftOperation ShiftAmt
// Op2 е непосредно адресиран;

Res = Op1 ALUOperation Op2
DataMem(DestOffset + DestBaseRegister) = Res
// Res е директно адресиран; DestBaseRegister се користи кога BaseAdMode[2]==1;
```

Во продолжение се дадени табела 3 и табела 4, каде се претставени различните операции на поместување и ALU операции кои се поддржани од MIMOPS процесорот. Овие операции се дефинирани со ShiftOperation и ALUOperation полињата од инструкцискиот код на операцијата (opcode), соодветно.

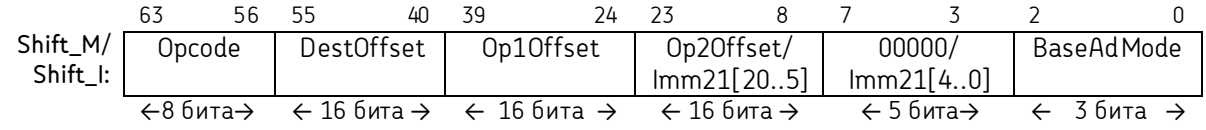
Табела 3 Операции на поместување кои се поддржани од MIMOPS процесор.

ShiftOperation	Значење	Извршување
SL(L/A)	поместување во лево (логичко или аритметичко)	Сите битови во податокот се поместуваат во лево за N места, така што крајните N места се пополнуваат со 0.
SRL	логичко поместување во десно	Сите битови во податокот се поместуваат во десно за N места, така што почетните N места се пополнуваат со 0.
SRA	аритметичко поместување во десно	Сите битови во податокот се поместуваат во десно за N места, така што почетните N места се пополнуваат со повторувања на најзначајниот бит од податокот.
ROR (ROL)	ротација во десно (ротација во лево) ROR(N) = ROL(32-N)	Сите битови во податокот се поместуваат во десно за N места, така што почетните N места се пополнуваат со истурканите битови од крајот.

Табела 4 ALU операции кои се поддржани од MIMOPS процесор.

ALUOperation	Значење	Извршување
SEQ	постави при еднаквост	Ако Op1==Op2, тогаш Res = 1
SNEQ	постави при нееднаквост	Ако Op1!=Op2, тогаш Res = 1
SGT	постави при поголемо	Ако Op1>Op2, тогаш Res = 1
SGE	постави при поголемо или еднакво	Ако Op1>=Op2, тогаш Res = 1
SLT	постави при помало	Ако Op1<Op2, тогаш Res = 1
SLE	постави при помало или еднакво	Ако Op1<=Op2, тогаш Res = 1
ADD	собирање	Res = Op1 + Op2
SUB	одземање	Res = Op1 - Op2
DIV	целобројно делење	Res = Op1 / Op2
MOD	модуларно делење	Res = Op1 % Op2
MUL	множење	Res = Op1 * Op2
AND	логичко И	Res = Op1 AND Op2
OR	логичко ИЛИ	Res = Op1 OR Op2
XOR	логичко Ексили	Res = Op1 XOR Op2
XNOR	логичко Екснили	Res = Op1 XNOR Op2
NOT	логичко Не	Res = NOT Op2

Инструкциите кои припаѓаат во групата за поместување можат да бидат дадени во Shift_M или Shift_I инструкциски формати, прикажани долу:



Според дадените формати на инструкции за поместување, може да се забележи дека инструкциите на поместување кои се специфицирани во Shift_M инструкциски формат вклучуваат код на операција и три операнди кои можат да бидат адресирани со примена на директно или базно адресирање, додека пак инструкциите на поместување кои се дефинирани во Shift_I инструкциски формат вклучуваат код на операција и два операнди (операнд1 и резултат) кои можат да бидат адресирани со примена на директно или базно адресирање, и уште еден операнд (операнд2) кој е специфициран како 21-битна непосредна вредност. Според тоа, основната разлика во овие два инструкциски формати е во режимот на адресирање, кој се применува за вториот влезен операнд.

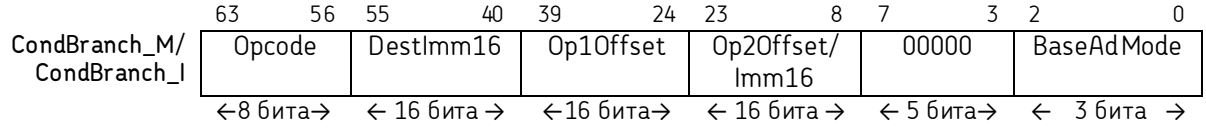
Операцијата на поместување се извршува во проточен режим, слично како ALU операциите, при што вредноста на првиот изворен операнд се поместува за вредноста во вториот изворен операнд и потоа резултатот се запишува назад во податочната меморија во чипот. Типот на операцијата на поместување која се применува на првиот изворен операнд е дефинирана со последните два бита од 8-битниот код на операцијата (2-битно ShiftOperation поле, чии што различни вредности веќе беа претставени во табела 3). Операцијата на поместување се извршува со употреба на посебна единица за поместување која е сместена во фазата за извршување од проточната линија. Всушност, извршувањето на операцијата на поместување може да се интерпретира на следниот начин:

```
Op1 = DataMem(Op1Offset + Op1BaseRegister)
// Op1 е директно адресиран; Op1Baseregister се користи кога BaseAdMode[1]==1;

Op2 = DataMem(Op2Offset + Op2BaseRegister)
// Op2 е директно адресиран; Op2Baseregister се користи кога BaseAdMode[0]==1;
или
Op2 = SignExtendTo32(Imm21)
// Op2 е непосредно адресиран;

Res = Op1 ShiftOperation Op2
DataMem(DestOffset + DestBaseRegister) = Res
// Res е директно адресиран; DestBaseRegister се користи кога BaseAdMode[2]==1;
```

Инструкциите кои припаѓаат во групата за гранење се поделени на инструкции наменети за условно гранење и инструкции наменети за безусловно гранење. Инструкциите кои вршат условно гранење можат да бидат дадени во CondBranch_M или CondBranch_I инструкциски формати, прикажани долу:



Според дадените инструкциски формати, може да се забележи дека инструкциите за условно гранење се дефинирани со код на операција, резултатен операнд кој користи релативно адресирање во однос на PC и два изворни операнди кои можат да бидат адресирани со примена на директно или базно адресирање,

или дури и непосредно адресирање за вториот изворен операнд во случај кога е употребен CondBranch_I инструкциски формат. Условот кој треба да се провери за да се изврши гранењето е дефиниран со последните три бита од 8-битниот код на операцијата, кои го претставуваат 3-битното BranchOperation поле.

Условното гранење се извршува во проточен режим, при што проверката на дадениот услов се изведува во фазата на декодирање од проточната линија, со помош на посебна компонента за споредба (компаратор). Тоа значи дека условното гранење завршува во фазата на декодирање и на тој начин овозможува извршување на скок со доцнење од само еден такт циклус, во случај кога грананата е земена. Условниот скок се реализира со ажурирање на вредноста на програмскиот бројач, како што веќе претходно беше претставено на сл. 38, каде беше објаснето релативното адресирање во однос на PC. Според тоа, извршувањето на условно гранење може да се интерпретира на следниот начин:

```
Op1 = DataMem(Op1Offset + Op1BaseRegister)
// Op1 е директно адресиран; Op1Baseregister се користи кога BaseAdMode[1]==1;

Op2 = DataMem(Op2Offset + Op2BaseRegister)
// Op2 е директно адресиран; Op2Baseregister се користи кога BaseAdMode[0]==1;
или
Op2 = SignExtendTo32(Imm16)
// Op2 е непосредно адресиран;

Ако Op1 BranchOperation Op2 е исполнето, тогаш PC = PC + SignExtendTo32(DestImm16)*8
// DestImm16 е релативно адресиран во однос на PC;
```

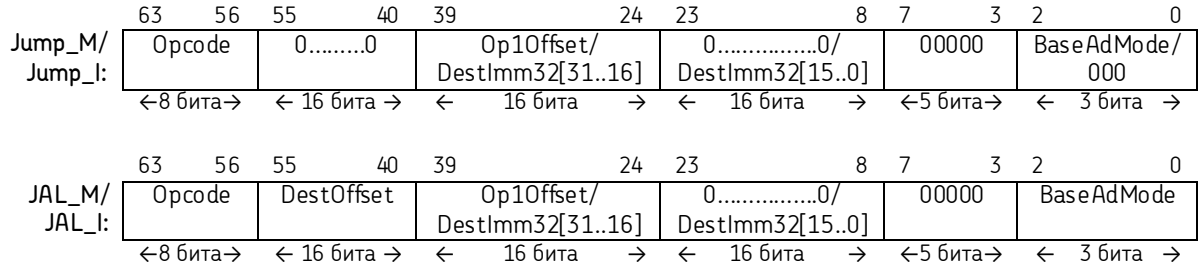
Во продолжение е дадена табела 5, каде се претставени различните условни скокови кои се поддржани од MIMOPS процесорот, а се дефинирани со BranchOperation полето од инструкцискиот код на операцијата (opcode).

Табела 5 Условни скокови кои се поддржани од MIMOPS процесор.

BranchOperation	Значење	Извршување
BEQ	скок при еднаквост	Ако Op1==Op2, тогаш PC = PC + SignExtendTo32(DestImm16)*8
BNEQ	скок при нееднаквост	Ако Op1!=Op2, тогаш PC = PC + SignExtendTo32(DestImm16)*8
BGT	скок при поголемо	Ако Op1>Op2, тогаш PC = PC + SignExtendTo32(DestImm16)*8

BGE	скок при поголемо или еднакво	Ако Op1>=Op2, тогаш PC = PC + SignExtendTo32(DestImm16)*8
BLT	скок при помало	Ако Op1<Op2, тогаш PC = PC + SignExtendTo32(DestImm16)*8
BLE	скок при помало или еднакво	Ако Op1<=Op2, тогаш PC = PC + SignExtendTo32(DestImm16)*8

Инструкциите кои вршат безусловно гранење можат да бидат интерпретирани како скокови (jump), дадени во Jump_M или Jump_I инструкциски формати, или како скокови со зачувување на состојба на адресата на следната инструкција (jump and link), дадени во JAL_M или JAL_I инструкциски формати. Овие инструкциски формати се прикажани во продолжение:



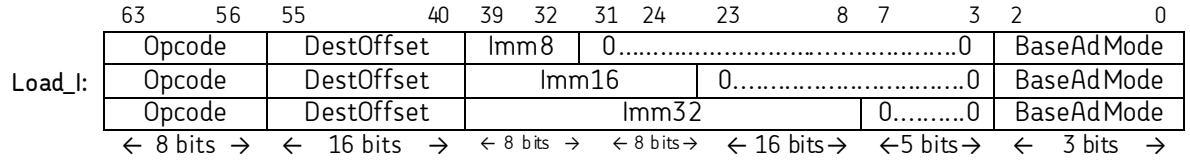
Според дадените инструкциски формати, може да се забележи дека инструкциите за безусловно гранење се дефинирани со код на операција и еден операнд кој користи директно адресирање во однос на PC. Всушност, кај скоковите кои се специфицирани во Jump_M или JAL_M инструкциски формат, операндот иницијално применува директно или базно адресирање, а потоа неговата вредност (која е прочитана од податочната меморија во чипот) се множи со осум и се добива адресата на скок. На сличен начин, кај скоковите кои се специфицирани во Jump_I или JAL_I инструкциски формат, операндот е иницијално дефиниран како 32-битна непосредна вредност, која се поместува за три бита во лево и ја дава адресата на скокот. Единствената разлика помеѓу Jump и JAL инструкциите е тоа што JAL инструкциите имаат и резултатен операнд кој може да биде адресиран со примена на директно или базно адресирање, а истиот се користи за да се зачува адресата на следната инструкција, како адреса за враќање назад (return address). Според тоа, JAL инструкциите најчесто се употребуваат при повик на функции, каде што враќањето назад во главната програма се извршува со скок кон адресата за враќање назад, која е претходно сочувана.

Безусловните гранења се извршуваат проточно, при што извршувањето на скоковите завршува во фазата на декодирање од проточната линија, додека пак скоковите со зачувување на состојба на адресата на следната инструкција завршуваат во последната фаза од проточната линија (запишување на резултат). Како и да е, двата типа на безусловно гранење веднаш преминуваат кон извршување на инструкциите кои следат после адресата на гранење, без да предизвикаат било какви застои. Генерално, однесувањето на инструкциите за скокање може да се интерпретира на следниот начин:

PC = Op1*8, каде Op1 = DataMem(Op1Offset + Op1BaseRegister)
// Op1 е директно адресиран во однос на PC; Op1Baseregister се користи кога BaseAdMode[1]==1;
или
PC = DestImm32*8
// DestImm32 е директно адресиран во однос на PC;

Res = PCfetch = PC+8 //Важи само за JAL инструкции, PCfetch е адреса на следна инструкција;
DataMem(DestOffset + DestBaseRegister) = Res
//Res е директно адресиран; DestBaseRegister се користи кога BaseAdMode[2]==1;

Инструкциите кои припаѓаат во групата за вчитување на константи во податочната меморијата во чипот, можат да бидат дадени во некоја од трите различни варијанти на Load_I инструкциски формати, прикажани долу:



Според дадените инструкциски формати, може да се забележи дека инструкциите за вчитување на константи се дефинирани со код на операција, еден изворен операнд кој користи непосредно адресирање и еден резултатен операнд кој може да биде адресиран со примена на директно или базно адресирање. Во зависност од последните два бита од операцискиот код, изворниот операнд може да биде специфициран како 8-битна, 16-битна или 32-битна константа.

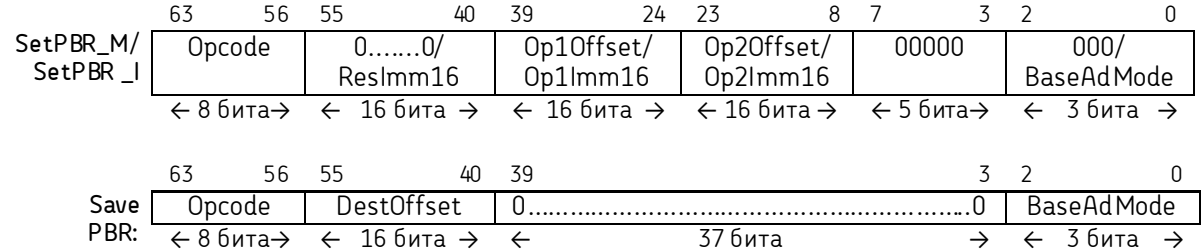
Инструкцијата за вчитување се извршува во проточен режим, при што вредноста на изворниот операнд се проширува до 32 бита и потоа се запишува во податочната меморија во чипот, на адресното отстапување кое е дефинирано

со DestOffset полето. Всушност, ваквото однесување на инструкциите за вчитување на константи може да се интерпретира на следниот начин:

```
Op1 = SignExtendTo32(Imm8)
или
Op1 = SignExtendTo32(Imm16)
или
Op1 = Imm16&zeroes[15..0];
или
Op1 = Imm32
// Op1 е непосредно адресиран;

Res = Op1;        DataMem(DestOffset + DestBaseRegister) = Res;
// Res е директно адресиран; DestBaseRegister се користи кога BaseAdMode[2]==1;
```

Инструкциите кои припаѓаат во групата за работа со регистри за страници и базни регистри се поделени на: инструкции кои ги поставуваат регистрите за страници и/или базните регистри и инструкции наменети за зачувување на регистрите за страници и/или базните регистри во податочната меморија во чипот. Инструкциите кои ги поставуваат регистрите за страници и/или базните регистри може да бидат дадени во SetPBR_M или SetPBR_I инструкциски формати, додека пак инструкциите за зачувување на регистрите за страници и/или базните регистри во податочната меморија во чипот се дефинираат со SavePBR инструкциски формат. Овие инструкциски формати се прикажани во продолжение:



Според дадените инструкциски формати, може да се забележи дека инструкциите за поставување (set) кои се специфицирани во SetPBR_M инструкциски формат вклучуваат код на операција и два изворни операнди кои може да бидат адресирани со примена на директно или базно адресирање, додека пак инструкциите за поставување (set) кои се специфицирани во SetPBR_I инструкциски

формат вклучуваат код на операција и три изворни операнди кои се дадени како 16-битни непосредни вредности. Според тоа, основната разлика помеѓу овие два формата на инструкции за поставување се состои во тоа дали регистрите за страници и/или базните регистри се поставуваат со вредности кои се прочитани од податочната меморија во чипот или со непосредни вредности кои се дадени како Op1Imm16, Op2Imm16 и ResImm16 константи во инструкцијата. Од друга страна, инструкциите за зачувување се дефинирани со код на операција и резултатен операнд кој може да биде адресиран со примена на директно или базно адресирање, а при тоа се користи за да го специфицира адресното отстапување на селектираниот мемориски блок, каде треба да се сочуваат регистрите за страници и/или базните регистри.

Операциите за поставување се извршуваат во проточен режим, при што типот на операција за поставување која треба да се изврши се дефинира со последните четири бита од 8-битниот код на операција (4-битно SetPBROperation поле, чии што различни вредности се подоцна претставени во табела 6). Извршувањето на SetPBR_I инструкциите завршува во фазата за декодирање од проточната линија, додека пак извршувањето на SetPBR_M инструкциите завршува во фазата за извршување од проточната линија, бидејќи запишувањето во регистрите за страници/базните регистри мора да се изврши еден такт циклус после пристапот до податочната меморија во чипот. Како и да е, двата типа на операции за поставување не воведуваат никакви доцнења во извршувањето на инструкциите од програмата, кои следат после операцијата на поставување. Во главно, однесувањето на инструкциите за поставување може да се интерпретира на следниот начин:

```
R1Value = DataMem(Op1Offset + Op1BaseRegister)[31..16]
// Op1 е директно адресиран; Op1Baseregister се користи кога BaseAdMode[1]==1;
или
R1Value = Op1Imm16
// Op1 е непосредно адресиран;

R2Value = DataMem(Op1Offset + Op1BaseRegister)[15..00]
// Op1 е директно адресиран; Op1Baseregister се користи кога BaseAdMode[1]==1;
или
```

R2Value = Op2Imm16
// Op2 е непосредно адресиран;

R3Value = DataMem(Op2Offset + Op2BaseRegister)[15..00]
// Op2 е директно адресиран; Op2Baseregister се користи кога BaseAdMode[0]==1;
или
R3Value = ResImm16
//Res е непосредно адресиран;

PR1/BR1 = R1Value; PR2/BR2 = R2Value; PR3/BR3 = R3Value;
// поставување на само еден или сите регистри за страници или базни регистри;
PR1/PR2/PR3 = R1Value; BR1/BR2/BR3= R2Value;
// поставување на еден регистар за страница и еден базен регистар истовремено;

SavePBR инструкциите за зачувување на регистрите за страници и/или базните регистри извршуваат различни операции на зачувување, во зависност од последните четири бита од 8-битниот код на операција, кои го дефинираат SavePBROperation полето, чии што различни вредности се подоцна претставени во табела 7. Генерално, операциите за зачувување се извршуваат во проточен режим, на тој начин што вредноста на регистрите за страници и/или базните регистри, која е прочитана во фазата за декодирање, се запишува на резултатната адреса во податочната меморија во чипот, за време на последната фаза за запишување на резултат од проточната линија. Всушност, ова однесувањето може да се интерпретира на следниот начин:

RValue = PR1/BR1; или RValue = PR2/BR2; или RValue = PR3/BR3;
// зачувување на само еден регистер за страница или базен регистер;
или
RValue = PR1 & BR1; или RValue = PR2 & BR2; или RValue = PR3 & BR3;
// зачувување на регистер за страница и базен регистер;

Res =RValue; DataMem(DestOffset + DestBaseRegister) = Res
// Res е директно адресиран; DestBaseRegister се користи кога BaseAdMode[2]==1;

Во продолжение се дадени табела 6 и табела 7, каде се претставени различните операции за поставување и зачувување на регистри за страници и базни регистри, кои се поддржани од MIMOPS процесорот. Овие операции се дефинирани со SetPBROperation и SavePBROperation полињата од инструкцискиот код на операцијата (opcode), соодветно.

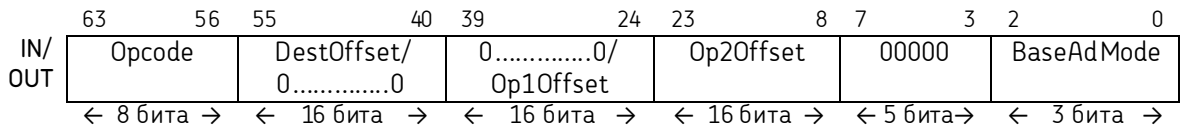
Табела 6 Операции за поставување (на регистри за страници и базни регистри) кои се поддржани од MIMOPS процесор.

SetPBROperation	Значење	Извршување
SetPRs	постави ги сите регистри за страници	PR1 = R1Value; PR2 = R2Value; PR3 = R3Value;
SetPR1	постави регистер за страница 1	PR1 = R1Value
SetPR2	постави регистер за страница 2	PR2 = R2Value
SetPR3	постави регистер за страница 3	PR3 = R3Value
SetBRs	постави ги сите базни регистри	BR1 = R1Value; BR2 = R2Value; BR3 = R3Value;
SetBR1	постави базен регистер 1	BR1 = R1Value
SetBR2	постави базен регистер 2	BR2 = R2Value
SetBR3	постави базен регистер 3	BR3 = R3Value
SetPR1BR1	постави регистер за страница и базен регистер 1	PR1 = R1Value; BR1 = R2Value;
SetPR2BR2	постави регистер за страница и базен регистер 2	PR2 = R1Value; BR2 = R2Value;
SetPR3BR3	постави регистер за страница и базен регистер 3	PR3 = R1Value; BR3 = R2Value;

Табела 7 Операции за зачувување (на регистри за страници и базни регистри) кои се поддржани од MIMOPS процесор.

SavePBROperation	Значење	Извршување
SavePR1	зачувај регистер за страница 1	Res = PR1
SavePR2	зачувај регистер за страница 2	Res = PR2
SavePR3	зачувај регистер за страница 3	Res = PR3
SaveBR1	зачувај базен регистер 1	Res = BR1
SaveBR2	зачувај базен регистер 2	Res = BR2
SaveBR3	зачувај базен регистер 3	Res = BR3
SavePR1BR1	зачувај регистер за страница и базен регистер 1	Res[31..16] = PR1 Res[15..0]=BR1
SavePR2BR2	зачувај регистер за страница и базен регистер 2	Res[31..16] = PR2 Res[15..0]=BR2
SavePR3BR3	зачувај регистер за страница и базен регистер 3	Res[31..16] = PR3 Res[15..0]=BR3

Инструкциите кои припаѓаат во групата за В/И комуникација можат да бидат дефинирани со IN или OUT инструкциски формати, прикажани долу:



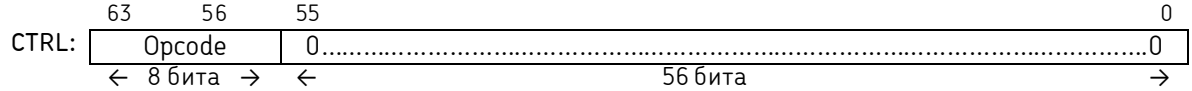
Според дадените инструкциски формати, може да се забележи дека инструкциите за комуникација со влезни уреди се специфицирани во IN инструкциски формат кој вклучува код на операција и по еден изворен и резултатен операнд, кои можат да бидат адресирани со примена на директно или базно адресирање, додека пак инструкциите за комуникација со излезни уреди се специфицирани во OUT инструкциски формат кој вклучува код на операција и два изворни операнди, кои можат да бидат адресирани со примена на директно или базно адресирање. Типот на В/И комуникација (влезна или излезна) се определува според кодот на операцијата, додека пак адресата на В/И уред се задава со вториот изворен операнд (Op2Offset) и се поставува на адресната магистрала - Address[31..0] за да го селектира соодветниот В/И уред. Дополнително, се користи и влезно/излезна податочна магистрала Data[31..0] за да се овозможи пренос на податоци помеѓу В/В уред и MIMOPS процесорот (инструкциска или податочна меморија во чипот). Податоците кои се пренесуваат се дефинирани со резултатниот операнд за IN инструкциите, односно со првиот изворен операнд (Op1Offset) за OUT инструкциите. Со оглед на тоа дека IN/OUT инструкции се извршуваат во проточен режим, нивното однесување може да се интерпретира на следниот начин:

```
Op2 = DataMem(Op2Offset + Op2BaseRegister)
// Op2 е директно адресиран; Op2Baseregister се користи кога BaseAdMode[0]==1;
Address[31..0] = Op2

Op1 = DataMem(Op1Offset + Op1BaseRegister)
// Op1 е директно адресиран; Op1Baseregister се користи кога BaseAdMode[1]==1;
Data[31..0] = Op1 кога се извршува OUT операција

Res = Data[31..0] кога се извршува IN операција
DataMem(DestOffset + DestBaseRegister) = Res
// Res е директно адресиран; DestBaseRegister се користи кога BaseAdMode[2]==1;
```

Инструкциите кои припаѓаат во групата за контрола се дадени во CTRL инструкциски формат, прикажан долу:



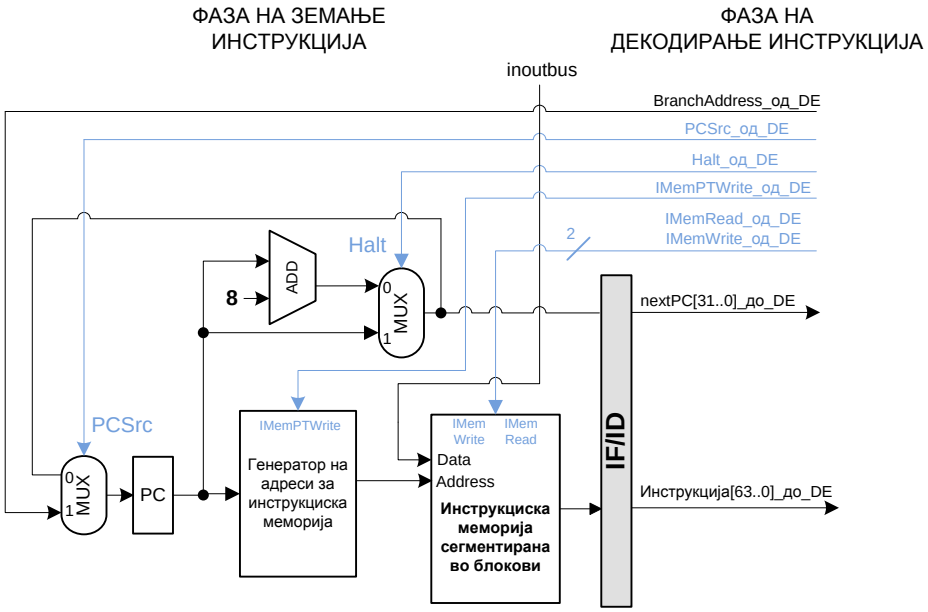
Овие инструкции вршат неколку контролни операции, во зависност од вредноста на кодот на операцијата и некои надворешни или контролни сигнали. Тука спаѓаат инструкции кои не вршат никаква операција (NOP), ја замрзнуваат состојбата на процесорот (HALT), или пак го контролираат ажурирањето во табелите на страници итн.

4.3. Проточна податочна патека

Предложениот MIMOPS процесор обезбедува проточно извршување на инструкциите во четири фази: земање на инструкција (instruction fetch - IF); декодирање на инструкција и пристап до операнди од податочната меморија во чипот (instruction decode - ID); извршување на ALU операција или операција на поместување (execute - EX); и запишување на резултат во податочната меморија во чипот (writeback - WB). Проточноста му овозможува на MIMOPS процесорот истовремено да обработува различни фази од инструкцијата, и на тој начин да овозможи повеќе инструкции да се преклопуваат во извршувањето (паралелизам на инструкциско ниво). Всушност, секоја фаза од проточната линија се користи точно еден такт циклус при паралелното извршување на различните инструкции.

Најважно за проточноста на едно-тактната имплементација на MIMOPS процесорот е воведувањето на проточни регистри, кои ја делат податочната патека на процесорот во четири фази: IF, ID, EX и WB. Проточните регистри се користат за зачувување на вредностите кои ги користи инструкцијата, откако ќе продолжи со извршувањето во наредните фази од проточната линија. Проточните регистри на MIMOPS процесорот се означени според фазите кои ги разделуваат: IF/ID, ID/EX, и EX/WB. Проточната податочна патека на MIMOPS процесорот која беше претходно претставена на сл. 27, покажува дека проточните регистри се дадени како посебни блокови со сива боја. И покрај сето тоа, сепак проточните регистри се имплементирани во соодветните модули за фазите од проточната линија, кои го генерираат влезот за соодветниот проточен регистер (пр. IF единицата генерира 64-битна инструкција и PC+8 адреса и потоа ги зачувува во IF/ID проточниот регистер). Во продолжение е даден детален опис на секој модул за четирите проточни фази (IF, ID, EX и WB) од MIMOPS процесорот.

Функцијата на единицата за земање на инструкција е да ја земе инструкцијата од инструкциската меморија со помош на тековната вредност на програмскиот бројач (PC), и потоа да ја постави вредноста на PC да покажува кон следната инструкција која треба да се изврши, како што е прикажано на сл. 39. Единицата за земање на инструкција се состои од следните логички елементи: 32-битен програмски бројач, собирач кој ја зголемува вредноста на PC за 8, генератор на адреси за инструкциска меморија, инструкциска меморија сегментирана во блокови, и мултиплексер кој се користи за да ја селектира следната вредност на PC (PC+8 или адреса на гранење).



Слика 39 Фаза на земање инструкција од проточна податочна патека на MIMOPS процесор.

Според сл. 39, земањето на инструкција започнува кога MIMOPS процесорот ја проследува тековната вредност на PC до генераторот на адреси за инструкциската меморија, кој треба да ја генерира ефективната физичка адреса на побараната инструкција, извршувајќи пребарување во својата инвертирана табела на страници. Овој процес на преведување на виртуелни адреси за инструкциската меморија веќе беше претходно објаснет и прикажан на сл. 28. Според тоа, откако ќе се генерира инструкциската адреса, инструкцијата се чита од инструкциската меморија и потоа се проследува кон IF/ID проточниот регистер, за да може подоцна да се употреби во фазата на декодирање. Покрај тоа, следната вредност на PC која е селектирана преку мултиплексер со кон-

тролниот сигнал PCSrc примен од фазата на декодирање, се препраќа кон IF/ID проточниот регистер во случај кога Halt сигналот е 0, за подоцна да се употреби за декодирање на гранења кои користат релативно адресирање во однос на PC.

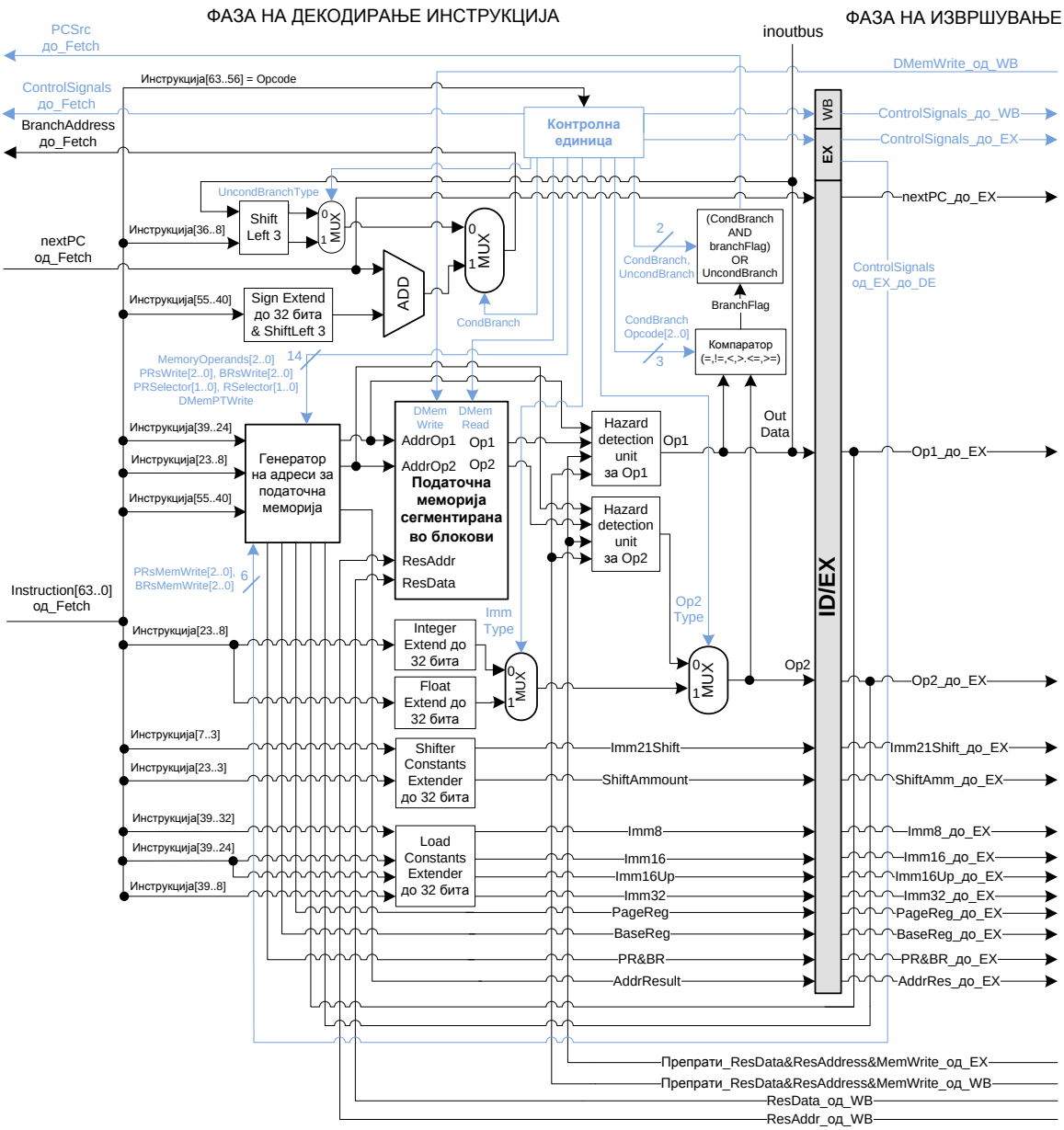
Главна функција на единицата за декодирање на инструкција е да ја декодира 64-битната инструкција која е проследена од претходната единица за земање на инструкција, и да обезбеди директен и едновремен пристап до инструкциските операнди, сместени во податочната меморија на чипот, како што е прикажано на сл. 40. Дополнително, оваа единица извршува некои операции на поместување и проширување со знак кои генерираат 32-битни вредности на операндите кои се специфицирани непосредно, директно во однос на PC или релативно во однос на PC. Покрај тоа, единицата за декодирање може да го определи начинот на гранење (условно или безусловно), врз основа на вредноста на некои контролни сигнали, и во случај на условно гранење може да извршува споредбени операции. Оваа единица овозможува и детекција на податочни зависности за двата влезни операнди, како и генерирање на комплетното множество на контролни сигнали (со контролната единица). Со цел да ги обезбеди сите овие функционалности, единицата за декодирање на инструкции ги содржи следните логички елементи: различни проширувачи со знак, поместувач во лево за 3 места, собирач, генератор на адреси за податочната меморија, податочна меморија сегментирана во блокови, мултиплексери, компаратор, едноставно логичко коло кое го поставува PCSrc контролниот сигнал, посебна единица за детекција на конфликти (hazard detection unit) за операнд1 и операнд2, и контролна единица. Во продолжение се дискутирани сите овие единици, освен контролната единица, која детално ќе биде претставена во следното поглавје.

MIMOPS процесорот најчесто користи инструкции со три-адресен формат, кој дефинира два изворни и еден резултатен операнд, до кои може да се пристапува директно во податочната меморија во чипот. Всушност, една таква декодирана MIMOPS инструкција проследува три вредности на мемориски адресни отстапувања (за операнд1, операнд2 и резултат) до генераторот на адреси за податочната меморија со цел тој да генерира физички адреси за дадените операнди (во сегментираната податочна меморија во чипот). Во таква ситуација, генераторот на адреси за податочната меморија веќе има извршено селекција на

податочни мемориски блокови (за операнд1, операнд2 и резултат), преку претходна SetPR инструкција, како што веќе беше објаснето и прикажано на сл. 36. Според тоа, генераторот на адреси за податочната меморија ги користи броевите на рамките (кои се специфицирани во мемориските блок селектори), базните регистри и влезните адресни отстапувања за да ги генерира физичките адреси на двата изворни и резултатниот операнд, како што веќе беше објаснето и прикажано на сл. 37. Тоа значи дека целиот процес на преведување на виртуелни адреси, кој се извршува од страна на генераторот на адреси за податочната меморија, вклучува употреба на регистри за страници, мемориски блок селектори, базни регистри и инвертирана табела на страници.

Веднаш после преведувањето на адресите, генераторот на адреси за податочната меморија ги испраќа физичките адреси на изворните операнди кон податочната меморија во чипот, која што потоа ги проследува прочитаните операнди кон поединечните единици кои се наменети да спречат појава на податочни конфликти (data hazards). Истовремено, генераторот на адреси за податочната меморија ја испраќа адресата на резултатот, и вредноста на некој регистер за страници, вредноста на некој базен регистар или двете вредности на некои регистер за страница и базен регистар кон ID/EX проточниот регистер, за потоа тие да можат да бидат употребени во фазата за извршување од проточната линија. Оваа единица исто така ги прима вредностите на операнд1 и операнд2 кои се прочитани од податочната меморија во чипот, а се проследени преку фазата за извршување со цел да бидат употребени за ажурирање на некој регистер за страница или базен регистер со мемориски вредности.

Според сл. 40, се забележува дека единицата за декодирање вклучува: два проширувачи до 32 бита за цели и реални броеви кои се дадени како 16-битни непосредни вредности; проширувач на константи наменети за поместување, кои се дадени како 21-битни или 5-битни непосредни вредности; и проширувач на константи наменети за вчитување, кои се дадени како 8-битни или 16-битни непосредни вредности. Овие непосредните вредности најчесто се користат како втор влезен операнд во MIMOPS инструкциите. Во тој случај, типот на операнд2 (непосредна или мемориска вредност) и типот на непосредна вредност (цел или реален број) за операнд2 се селектираат со посебни 2/1 мултиплексери.



Слика 40 Фаза на декодирање инструкција од проточна податочна патека на MIMOPS процесор.

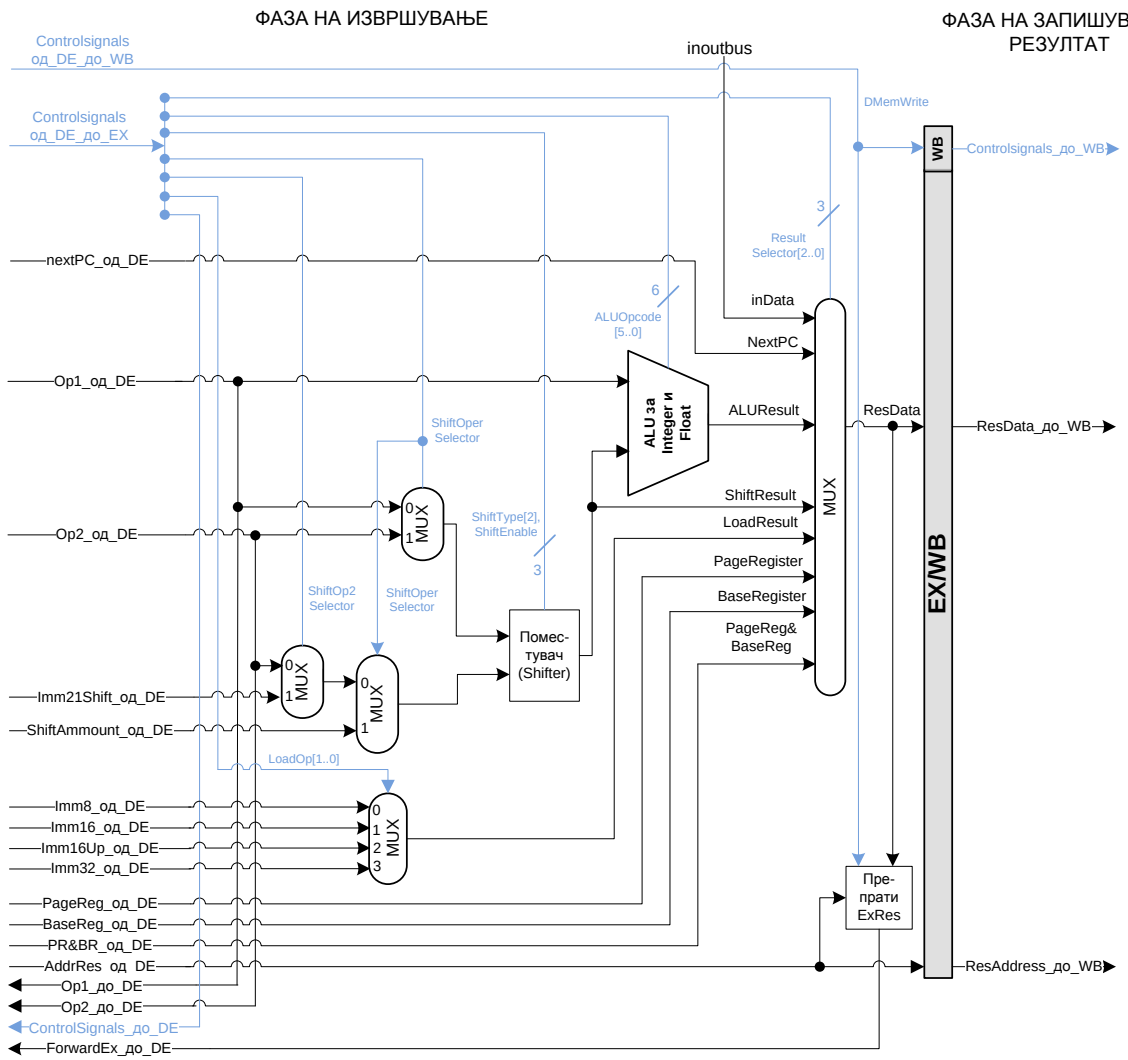
Единицата за декодирање имплементира условно и безусловно гранење и обезбедува поддршка за PC-директно и PC-релативно адресирање на операндите кои се специфицирани во инструкциите за гранење. Условното гранење се извршува со специјална единица за споредба (компаратор), која проверува неколку услови, вклучувајќи: еднаквост, нееднаквост, помало од, помало или еднакво, поголемо од, и поголемо или еднакво. Според тоа, операција на гранење се извршува само во случај доколку определен услов е исполнет. Од друга страна безусловните скокови се извршуваат секогаш. За да се овозможи сето тоа, MIMOPS процесорот дефинира неколку контролни сигнали (CondBranch,

branchFlag и UnCondBranch) кои определуваат дали ќе се изврши гранењето. Во случај на гранење, адресата која е дадена во PC-директен или PC-релативен адресен режим се пресметува со употреба на поместувач, проширувач, собирач и мултиплексери (PC = Op1*8, PC = DestImm32*8 или PC=PC + SignExtendTo32 [DestImm16]*8).

Контролниот дел од фазата за декодирање во проточната линија е обележан со сина боја на сл. 40. Според сл. 40, тој вклучува контролна единица која за 8-битниот влезен код на операција генерира контролни сигнали кои ќе го координираат проточното извршување на дадената инструкција. Некои од овие контролни сигнали се користат во фазите на земање и декодирање, додека пак останатите се проследуваат кон фазите за извршување и запишување на резултат, преку ID/EX проточниот регистер. Повеќе детали за контролната единица и намената на нејзините контролни сигнали се дадени во следното поглавје.

Функцијата на единицата за извршување на MIMOPS процесорот е да извршува поместување и аритметичко-логички операции, и да ја селектира вредноста која треба да се запише во податочната меморија на чипот како резултат. Дополнително, единицата за извршување врши и препраќање на резултатните вредност и адреса кон фазата на декодирање од проточната линија, за да спречи појава на податочни конфликти. Всушност, сите овие функционалности се обезбедени со следните логички елементи од единицата за извршување: мултиплексери, поместувач, ALU (за цели и реални броеви) и единица за препраќање на резултат. Конкретно, тоа може да се забележи на сл. 41, каде е прикажана структурата и интерфејсот на единицата за извршување од MIMOPS процесорот.

Операциите на поместување кои може да се изведуваат во фазата за извршување се: поместување во лево (логичко или аритметичко), поместување во десно (логичко или аритметичко) и ротација (во лево или десно). Всушност, можни се два различни начина на поместување, и тоа: вториот изворен флексибилен операнд се поместува за вредноста која е дадена со ShiftAmmount полето, или првиот изворен операнд се поместува за вредноста која е дадена со операнд2 или со Imm21 непосредна вредност која е проширена со знак до 32-битна константа. Изборот на соодветните операнди и типот на поместување се извршува со мултиплексери и контролни сигнали.

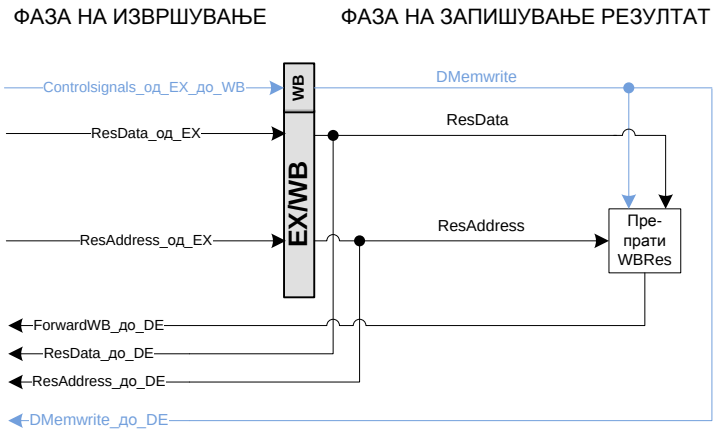


Слика 41 Фаза на извршување од проточна податочна патека на MIMOPS процесор.

Откако ќе се изврши поместување на вториот флексибилен операнд, двата операнда се пренесуваат до аритметичко-логичката единица која може да извршува операции со цели и реални броеви. Операциите со цели броеви кои се поддржани од ALU единицата се: постави при еднаквост, постави при неед-наквост, постави при поголемо, постави при поголемо или еднакво, постави при помало, постави при помало или еднакво, собирање, одземање, целобројно де-лење, модуларно делење (остаток), множење, логичко И, логичко ИЛИ, логичко Ексили, логичко Екснили и логичко Не. Дополнително, операциите со реални броеви кои се поддржани од ALU единицата се: постави при еднаквост, постави при нееднаквост,постави при поголемо, постави при поголемо или еднакво, постави при помало, постави при помало или еднакво, собирање, собирање и множење, одземање, множење, делење, реципрочна вредност, апсолутна вред-

ност, негација, логичко И и Ексили. Всушност, селекцијата на ALU операцијата за дадените влезни операнди (цели и реални броеви) се определува според вредноста на ALUOpcode контролниот сигнал.

Откако ќе заврши операцијата за поместување и/или ALU операцијата, потребно е да се испрати резултатот кон фазата за запишување на резултат. За таа цел, единицата за извршување користи 8/1 мултиплексер кој ја поставува вредноста на резултатот на еден од понудените влезови: inData (за IN инструкции), nextPC (за JAL инструкции), ALUresult (за ALU инструкции), ShiftResult (за инструкции на поместување), LoadResult (за инструкции за вчитување на константа, селектирана со 4/1 мултиплексер), PageRegister (за savePR инструкции), BaseRegister (за saveBR инструкции) или PageRegister&BaseRegister (за savePBR инструкции). Покрај тоа, селектираниот резултат (ResData), адресата на резултатот (ResAddress), и DMemWrite контролниот сигнал се испраќаат преку ForwardExRes единицата за препраќање до фазата на декодирање со цел да се спречи појава на податочни конфликти. Овие сигнали (ResData, ResAddress и DMemWrite) исто така се препраќаат и кон фазата за запишување на резултат од проточната линија, која е прикажана на сл. 42.



Слика 42 Фаза на запишување резултат од проточна податочна патека на MIMOPS процесор.

Функцијата на единицата за запишување на резултат на MIMOPS процесорот е да ја запише резултатната вредност во податочната меморија на чипот и да обезбеди препраќање на резултатот до фазата за декодирање (за да спречи појава на податочни конфликти). За таа цел се користат сигналите: ResData, ResAddress и DMemWrite, кои се испраќаат кон фазата за декодирање

од проточната линија, каде се извршува операцијата на запишување на резултат во податочната меморија во чипот. Истовремено, единицата за декодирање ги прима истите сигнали преку ForwardWbRes единицата, и потоа ги користи во своите поединечни единици за детекција на конфликти за операнд1 и операнд2, со цел да спречи појава на податочни конфликти. Според тоа, може да се заклучи дека единицата за запишување на резултат се однесува само како интерфејс кон единицата за декодирање, каде всушност се извршуваат потребните операции на запис и разрешување на податочни конфликти.

Во продолжение е дадена табела 8, во која се претставени различните акции кои се извршуваат од страна на различните типови на MIMOPS инструкции, за време на секоја фаза од проточната податочна патека на процесорот: земање на инструкција, декодирање, извршување и запишување на резултат.

Табела 8 Извршување на различни типови на инструкции во проточна податочна патека на MIMOPS процесор.

ALU_M инструкциски формат	
Земање	IR =InstMem(PC); PC=PC+8;
Декодирање	Op1 = DataMem(Op1Offset + Op1BaseRegister); Op2 = DataMem(Op2Offset + Op2BaseRegister) ShiftOperation ShiftAmt;
Извршување	Res = Op1 ALUOperation Op2;
Запишување на резултат	DataMem(DestOffset + DestBaseRegister) = Res;
ALU_I инструкциски формат	
Земање	IR =InstMem(PC); PC=PC+8;
Декодирање	Op1 = DataMem(Op1Offset + Op1BaseRegister); Op2 = SignExtendTo32(Imm16) ShiftOperation ShiftAmt;
Извршување	Res = Op1 ALUOperation Op2;
Запишување на резултат	DataMem(DestOffset + DestBaseRegister) = Res;
Shift_M инструкциски формат	
Земање	IR =InstMem(PC); PC=PC+8;
Декодирање	Op1 = DataMem(Op1Offset + Op1BaseRegister); Op2 = DataMem(Op2Offset + Op2BaseRegister);
Извршување	Res = Op1 ShiftOperation Op2;
Запишување на резултат	DataMem(DestOffset + DestBaseRegister) = Res;

Shift_I инструкциски формат	
Земање	IR =InstMem(PC); PC=PC+8;
Декодирање	Op1 = DataMem(Op1Offset + Op1BaseRegister); Op2 = Op2 = SignExtendTo32(Imm21);
Извршување	Res = Op1 ShiftOperation Op2;
Запишување на резултат	DataMem(DestOffset + DestBaseRegister) = Res;
CondBranch_M инструкциски формат	
Земање	IR =InstMem(PC); PC=PC+8;
Декодирање	Op1 = DataMem(Op1Offset + Op1BaseRegister); Op2 = DataMem(Op2Offset + Op2BaseRegister); Ако Op1 BranchOperation Op2 е исполнето, тогаш PC = PC + SignExtendTo32(DestImm16)*8;
Извршување	
Запишување на резултат	
CondBranch_I инструкциски формат	
Земање	IR =InstMem(PC); PC=PC+8;
Декодирање	Op1 = DataMem(Op1Offset + Op1BaseRegister); Op2 = SignExtendTo32(Imm16); Ако Op1 BranchOperation Op2 е исполнето, тогаш PC = PC + SignExtendTo32(DestImm16)*8;
Извршување	
Запишување на резултат	
Jump_M инструкциски формат	
Земање	IR =InstMem(PC); PC=PC+8;
Декодирање	Op1 = DataMem(Op1Offset + Op1BaseRegister); PC = Op1*8;
Извршување	
Запишување на резултат	
Jump_I инструкциски формат	
Земање	IR =InstMem(PC); PC=PC+8;
Декодирање	PC = DestImm32*8;
Извршување	
Запишување на резултат	
JAL_M инструкциски формат	
Земање	IR =InstMem(PC); PCfetch=PC+8;
Декодирање	Op1 = DataMem(Op1Offset + Op1BaseRegister); PC = Op1*8;
Извршување	Res = PCfetch;
Запишување на резултат	DataMem(DestOffset + DestBaseRegister) = Res

JAL_I инструкциски формат	
Земање	IR =InstMem(PC); PCfetch=PC+8;
Декодирање	PC = DestImm32*8;
Извршување	Res = PCfetch;
Запишување на резултат	DataMem(DestOffset + DestBaseRegister) = Res
Load_I инструкциски формат	
Земање	IR =InstMem(PC); PC=PC+8;
Декодирање	Op1 = SignExtendTo32(Imm8); или Op1 = SignExtendTo32(Imm16); или Op1 = Imm16&zeros[15..0]; или Op1 = Imm32;
Извршување	Res = Op1;
Запишување на резултат	DataMem(DestOffset + DestBaseRegister) = Res
SetPBR_M инструкциски формат	
Земање	IR =InstMem(PC); PC=PC+8;
Декодирање	R1Value = DataMem(Op1Offset + Op1BaseRegister)[31..16]; R2Value = DataMem(Op1Offset + Op1BaseRegister)[15..00]; R3Value = DataMem(Op2Offset + Op2BaseRegister)[15..00];

Извршување	PR1/BR1 = R1Value; PR2/BR2 = R2Value; PR3/BR3 = R3Value;
	//еден или сите PR-и или BR-и
	PR1/PR2/PR3 = R1Value; BR1/BR2/BR3= R2Value; // еден PR и еден BR

Запишување на резултат	
SetPBR_I инструкциски формат	
Земање	IR =InstMem(PC); PC=PC+8;

Декодирање	R1Value = Op1Imm16; R2Value = Op2Imm16; R3Value = ResImm16;
	PR1/BR1 = R1Value; PR2/BR2 = R2Value; PR3/BR3 = R3Value;
	//еден или сите PR-и или BR-и
	PR1/PR2/PR3 = R1Value; BR1/BR2/BR3= R2Value; // еден PR и еден BR

Извршување	
Запишување на резултат	
SavePBR инструкциски формат	
Земање	IR =InstMem(PC); PC=PC+8;

Декодирање	RValue = PR1/BR1; или RValue = PR2/BR2; или RValue = PR3/BR3;
	//еден PR или еден BR
	RValue = PR1 & BR1; или RValue = PR2 & BR2; или RValue = PR3 & BR3;
	// еден PR и еден BR

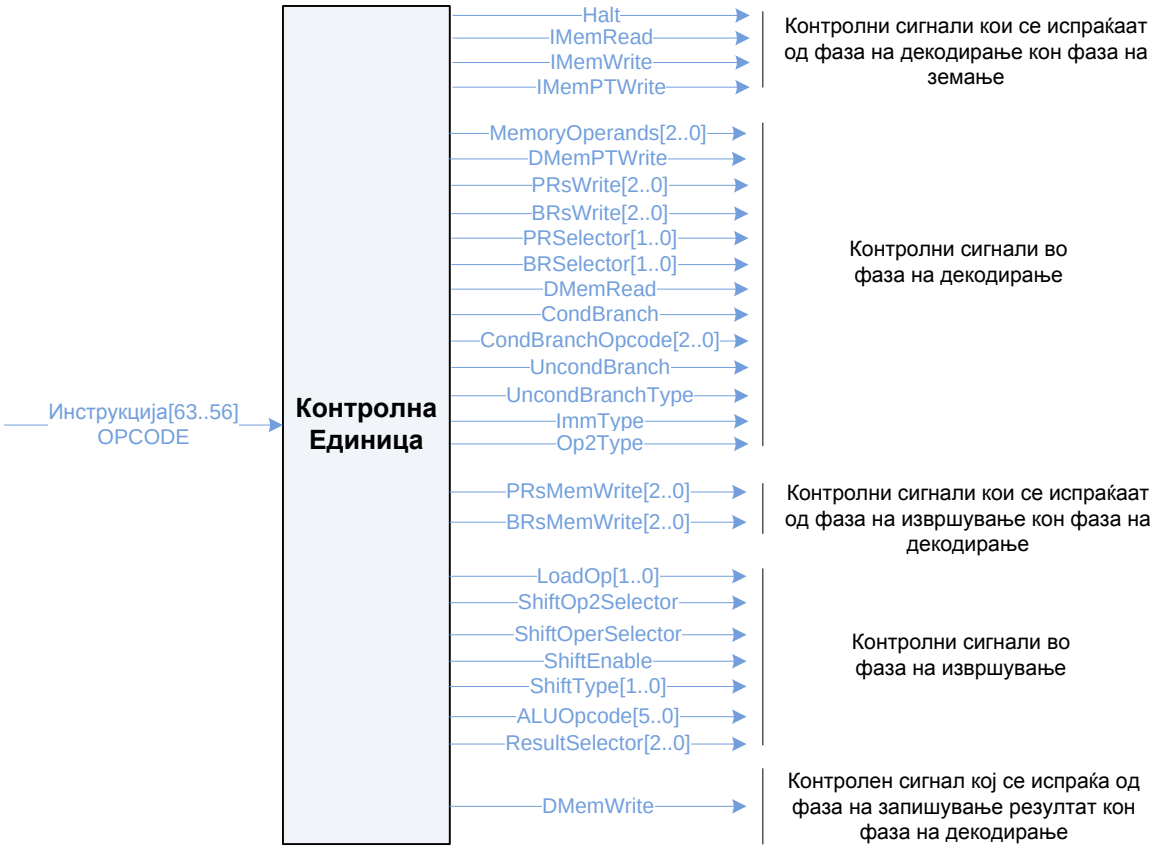
Извршување	Res = RValue;
Запишување на резултат	DataMem(DestOffset + DestBaseRegister) = Res
IN инструкциски формат	
Земање	IR =InstMem(PC); PC=PC+8;
Декодирање	Op2 = DataMem(Op2Offset + Op2BaseRegister); Address[31..0] = Op2;
Извршување	Res = Data[31..0];
Запишување на резултат	DataMem(DestOffset + DestBaseRegister) = Res

OUT инструкциски формат	
Земање	IR =InstMem(PC); PC=PC+8;
Декодирање	Op1 = DataMem(Op1Offset + Op1BaseRegister); Op2 = DataMem(Op2Offset + Op2BaseRegister); Data[31..0] = Op1; Address[31..0] = Op2;
Извршување	
Запишување на резултат	
CTRL инструкциски формат	
Земање	IR =InstMem(PC); PC=PC+8, за NOP или инструкции кои ги ажурираат табелите на страници; PC не се менува за HLT инструкции; програмата продолжува да извршува NOP операции;
Декодирање	Се овозможува запишување во табела на страници, при извршување на инструкции кои ги ажурираат табелите на страници;
Извршување	
Запишување на резултат	

4.4. Проточна контролна патека

Контролната единица на MIMOPS процесорот го испитува 8-титниот код на операција, даден со инструкциските битови [63 – 56], и потоа генерира контролни сигнали кои го контролираат проточното извршување на декодираната MIMOPS инструкција. Всушност, на сл. 43 е претставено комплетното множество на контролни сигнали, кои се генерираат од страна на контролната единица на MIMOPS процесорот при декодирање на некоја инструкција.

На сл. 43 може да се забележи дека генерираните контролни сигнали се организирани во неколку групи: контролни сигнали кои се испраќаат од фаза на декодирање кон фаза на земање, контролни сигнали во фаза на декодирање, контролни сигнали кои се испраќаат од фаза на извршување кон фаза на декодирање, контролни сигнали во фаза на извршување и контролни сигнали кои се испраќаат од фаза на запишување резултат кон фаза на декодирање. И покрај тоа што сите контролните сигнали се генерираат во фазата на декодирање, каде е имплементирана контролната единица, сепак тие се користат во различни фази од проточната податочна патека. Всушност, контролните сигнали се пренесуваат директно кон претходните фази, (пр. од DE до FE), односно преку проточни регистри кон следните фази, (пр. од DE до EX), од проточната линија.



Слика 43 Контролна единица на MIMOPS процесор.

Контролните сигнали кои се испраќаат од фазата на декодирање кон фазата на земање се: Halt, IMemRead, IMemWrite и IMemPTWrite. Halt сигналот се користи за да го замрзне програмскиот бројач, при извршување на HLT (halt) инструкција. IMemRead и IMemWrite сигналите се наменети за да овозможат читање или запишување во инструкциската меморија во чипот, соодветно. IMemPTWrite сигналот се користи за да обезбеди поддршка за ажурирање на табелата на страници, која е сместена во генераторот на адреси за инструкциската меморија.

Контролните сигнали од фазата на декодирање се: MemoryOperands[2..0], DMemPTWrite, PRsWrite[2..0], BRsWrite[2..0], PRSelector[1..0], BRSelector[1..0], DMemRead, CondBranch, CondBranchOpcode[2..0], UncondBranch, UncondBranchType, ImmType и Op2Type. Првите шест од дадените контролни сигнали од фазата на декодирање се користат од страна на генераторот на адреси за податочната меморија. Првиот сигнал - MemoryOperands[2..0] е 3-битен контролен сигнал кој определува дали изворните операнди (операнд1 и операнд1) треба да се

земат од податочната меморија во чипот, и дали резултатниот операнд треба да се запише назад во податочната меморија во чипот. Според тоа, овој контролен сигнал се користи за да го активира генерирањето на адресите на потребните операнди. Следниот сигнал - DMemPTWrite се користи за да обезбеди поддршка за ажурирање на табелата на страници, која е сместена во генераторот на адреси за податочната меморија. PRsWrite[2..0] и BRsWrite[2..0] сигналите се наменети за активирање на ажурирањето на регистрите за страници и базните регистри, соодветно. Всушност, овие 3-битни контролни сигнали содржат по еден бит за секој операнд (операнд1, операнд2, резултат), така што доколку даден бит е поставен, тогаш соодветниот регистер за страница/базен регистер за дадениот операнд треба да се ажурира. Од друга страна, сигналите PR-Selector[1..0] и BRSelector[1..0] се користат за селекција на регистер за страница и/или базен регистер кој треба да се зачува во податочната меморија во чипот.

Кога станува збор за податочната меморија во чипот, може да се каже дека нејзината работа е управувана од два контролни сигнали: DMemRead и DMemWrite (примен од фазата за запишување на резултат). DMemRead и DMem-Write сигналите се наменети за да овозможат читање или запишување во податочната меморија во чипот, соодветно. Кога станува збор за гранење, може да се каже дека оваа операција е управувана со четири контролни сигнали: CondBranch, CondBranchOpcode[2..0], UncondBranch и UncondBranchType. Cond-Branch сигналот се поставува при извршување на условно гранење и се користи за да се селектира адресата на гранење и да се проследи кон PC. Всушност, операцијата на гранење се извршува само во случај доколку условот, кој е специфициран со 3-битен CondBranchOpcode[2..0] сигнал, е исполнет. UncondBranch сигналот се поставува при извршување на скокови и се користи за да се селектира адресата на гранење и да се проследи кон PC. Начинот на пресметување на адресата на скокот се определува со UncondBranchType контролниот сигнал (PC = Op1*8 или PC = DestImm32*8). Последните два контролни сигнали од фазата на декодирање, ImmType и Op2Type, се користат за да се определи типот на вториот изворен операнд. Всушност, ImmType сигналот определува дали вториот изворен операнд е цел или реален број, а Op2Type определува дали операнд2 е прочитан од податочната меморија во чипот или е непосредна вредност.

Контролните сигнали кои се испраќаат од фазата на извршување кон фазата на декодирање се: PRsMemWrite[2..0] and BRsMemWrite[2..0]. Овие контролни сигнали се наменети за активирање на ажурирањето на регистрите за страници и базните регистри, соодветно. Нивната функција е слична со онаа на контролните сигнали PRsWrite[2..0] и BRsWrite[2..0], со таа разлика што тие обезбедуваат ажурирање на регистрите за страници/базните регистри со вредности кои се прочитани од податочната меморија во чипот, наместо со непосредни вредности.

Контролните сигнали од фазата на извршување се: LoadOp[1..0], ShiftOp2Selector, ShiftOperSelector, ShiftEnable, ShiftType[1..0], ALUOpcode[5..0] и ResultSelector[2..0]. LoadOp[1..0] е 2-битен сигнал кој се користи за селекција на една од четирите непосредни вредности (Imm8, Imm16, Imm16Up и Imm32) која треба да се вчита во податочната меморија во чипот. Кога станува збор за поместување, може да се каже дека оваа операција е управувана со четири контролни сигнали: ShiftOp2Selector, ShiftOperSelector, ShiftEnable и ShiftType[1..0]. ShiftOp2Selector сигналот определува дали операнд1 се поместува за вредноста на операнд2 или за вредноста на Imm21 константа која е проширена со знак до 32 бита. ShiftOperSelector сигналот го определува начинот на поместување: првиот изворен операнд се поместува за вредноста која е селектирана со ShiftOp2Selector сигналот или вториот изворен флексибилен операнд се поместува за вредноста која е дадена со ShiftAmmount полето. Извршувањето на операцијата на поместување се овозможува со ShiftEnable контролниот сигнал. Во тој случај, поместувачот извршува операција (SL[L/A], SRL, SRA, ROR[ROL]), која е дефинирана со 2-битниот ShiftType[1..0] сигнал. Откако ќе заврши поместувањето, се извршува ALU операцијата. Селекцијата на ALU операцијата која треба да се изврши со влезните цели или реални броеви, се прави со 6-битниот ALUOpcode[5..0] сигнал. Веднаш откако ќе заврши ALU операцијата, се користи 3-битен ResultSelector[2..0] сигнал за да се избере резултатниот податок (ResultData), кој треба да се запише во податочната меморија во чипот. Резултатниот податок се поставува како една од следните осум можности: inData, nextPC, ALUresult, ShiftResult, LoadResult, PageRegister, BaseRegister или PageRegister&BaseRegister.

Контролниот сигнал кој се испраќа од фазата на запишување резултат кон фазата на декодирање е: DMemWrite. DMemWrite контролниот сигнал се поставува, кога некоја вредност треба да се складира во податочната меморија во чипот. Земајќи во предвид дека дадената вредност се зачувува истовремено додека два изворни операнди од друга инструкција се читаат од податочната меморија во чипот, може да се заклучи дека настанува проточно преклопување во извршувањето на инструкциите.

Во табела 9 која е дадена во продолжение е претставено доделувањето на контроли сигнали, за различните типови на MIMOPS инструкции, кои се декодирани од контролната единица. Во оваа табела се прикажани само оние контролни сигнали кои се поставуваат и кои влијаат на извршувањето на дадената инструкција, која е специфицирана со својот 8-битен код на операција (InstOpcode[7..0]).

Табела 9 Доделување на контролни сигнали за различни типови на MIMOPS инструкции.

Инструкциски тип	Контролни сигнали кои се поставени (останатите контролни сигнали имаат неопределена вредност x)
ALU_M	Halt = 0, MemoryOperands[2..0]=111, CondBranch=0 и UncondBranch=0 (PCSrc=0), Op2Type=0, ShiftOperSelector=1, ShiftEnable=1, ShiftType[1..0]=InstOpcode[1 downto 0], ALUOpcode[5..0]=InstOpcode[7 downto 2], ResultSelector[2..0]=010 и DMemWrite=1.
ALU_I	Halt = 0, MemoryOperands[2..0]=101, CondBranch=0 и UncondBranch=0 (PCSrc=0), ImmType=1 кога InstOpcode[7 downto 4]==1011, Op2Type=1, ShiftOperSelector=1, ShiftEnable=1, ShiftType[1..0]=InstOpcode[1 downto 0], ALUOpcode[5..0]=InstOpcode[7 downto 2], ResultSelector[2..0]=010 и DMemWrite=1.
Shift_M	Halt = 0, MemoryOperands[2..0]=111, CondBranch=0 и UncondBranch=0 (PCSrc=0), Op2Type=0, ShiftOperSelector=0 ShiftOp2Selector=0, ShiftEnable=1, ShiftType[1..0]= InstOpcode[1 downto 0], ResultSelector[2..0]=011 и DMemWrite=1.
Shift_I	Halt = 0, MemoryOperands[2..0]=101, CondBranch=0 и UncondBranch=0 (PCSrc=0), ShiftOperSelector=0 ShiftOp2Selector=1, ShiftEnable=1, ShiftType[1..0]= InstOpcode[1 downto 0], ResultSelector[2..0]=011 и DMemWrite=1.
Cond Branch_M	Halt = 0, MemoryOperands[2..0]=110, CondBranchOpcode[2..0]=InstOpcode[2 downto 0], CondBranch=1 и UncondBranch=0 (PCSrc=1 ако BrachFlag==1), Op2Type=0.
Cond Branch_I	Halt = 0, MemoryOperands[2..0]=100, CondBranchOpcode[2..0]=InstOpcode[2 downto 0], CondBranch=1 и UncondBranch=0 (PCSrc=1 ако BrachFlag==1), Op2Type=1, ImmType=0.

Инструк-циски тип	Контролни сигнали кои се поставени (останатите контролни сигнали имаат неопределена вредност x)
Jump_M	Halt = 0, MemoryOperands[2..0]=100, CondBranch=0 и UncondBranch=1 (PCSrc=1), UncondBranchType=0.
Jump_I	Halt = 0, MemoryOperands[2..0]=000, CondBranch=0 и UncondBranch=1 (PCSrc=1), UncondBranchType=1.
JAL_M	Halt = 0, MemoryOperands[2..0]=101, CondBranch=0 и UncondBranch=1 (PCSrc=1), UncondBranchType=0, ResultSelector[2..0]=001 и DMemWrite=1.
JAL_I	Halt = 0, MemoryOperands[2..0]=001, CondBranch=0 и UncondBranch=1 (PCSrc=1), UncondBranchType=1, ResultSelector[2..0]=001 и DMemWrite=1.
Load_I	Halt = 0, MemoryOperands[2..0]=001, CondBranch=0 и UncondBranch=0 (PCSrc=0), LoadOp[1..0]=InstOpcode[1 downto 0], ResultSelector[2..0]=100 и DMemWrite=1.
SetPBR_M	Halt = 0, MemoryOperands[2..0]=110, CondBranch=0 и UncondBranch=0 (PCSrc=0), Op2Type=0, PRsMemWrite[0]=1 кога InstOpcode[3 downto 0] == SetPRs or SetPR1 or SetPR1BR1, PRsMemWrite[1]=1 кога InstOpcode[3 downto 0] == SetPRs or SetPR2 or SetPR2BR2, PRsMemWrite[2]=1 кога InstOpcode[3 downto 0] == SetPRs or SetPR3 or SetPR2BR3, BRsMemWrite[0]=1 кога InstOpcode[3 downto 0] == SetBRs or SetBR1 or SetPR1BR1, BRsMemWrite[1]=1 кога InstOpcode[3 downto 0] == SetBRs or SetBR2 or SetPR2BR2, BRsMemWrite[2]=1 кога InstOpcode[3 downto 0] == SetBRs or SetBR3 or SetPR2BR3. // види табела 6, каде е објаснето SetPBROperation полето;
SetPBR_I	Halt = 0, MemoryOperands[2..0]=000, CondBranch=0 и UncondBranch=0 (PCSrc=0), PRsWrite[0]=1 кога InstOpcode[3 downto 0] == SetPRs or SetPR1 or SetPR1BR1, PRsWrite[1]=1 кога InstOpcode[3 downto 0] == SetPRs or SetPR2 or SetPR2BR2, PRsWrite[2]=1 кога InstOpcode[3 downto 0] == SetPRs or SetPR3 or SetPR2BR3, BRsWrite[0]=1 кога InstOpcode[3 downto 0] == SetBRs or SetBR1 or SetPR1BR1, BRsWrite[1]=1 кога InstOpcode[3 downto 0] == SetBRs or SetBR2 or SetPR2BR2, BRsWrite[2]=1 кога InstOpcode[3 downto 0] == SetBRs or SetBR3 or SetPR2BR3. // види табела 6, каде е објаснето SetPBROperation полето;
SavePBR	Halt = 0, MemoryOperands[2..0]=001, CondBranch=0 и UncondBranch=0 (PCSrc=0), DMemWrite=1, PRSelector[1..0]=00 кога InstOpcode[3 downto 0] == SavePR1 or SavePR1BR1, PRSelector[1..0]=01 кога InstOpcode[3 downto 0] == SavePR2 or SavePR2BR2, PRSelector[1..0]=10 кога InstOpcode[3 downto 0] == SavePR3 or SavePR3BR3, BRSelector[1..0]=00 кога InstOpcode[3 downto 0] == SaveBR1 or SavePR1BR1, BRSelector[1..0]=01 кога InstOpcode[3 downto 0] == SaveBR2 or SavePR2BR2, BRSelector[1..0]=10 кога InstOpcode[3 downto 0] == SaveBR3 or SavePR3BR3, ResultSelector [2..0]=101 кога InstOpcode[3 downto 0] == SavePR1 or SavePR2 or SavePR3, ResultSelector [2..0]=110 кога InstOpcode[3 downto 0] == SaveBR1 or SaveBR2 or SaveBR3, ResultSelector [2..0]=110 кога InstOpcode[3 downto 0] == SavePR1BR1 or SavePR2BR3 or SavePR3BR3. // види табела 7, каде е објаснето SavePBROperation полето;
IN	**Влез во податочна меморија:** Halt = 0, MemoryOperands[2..0]=011, CondBranch=0 и UncondBranch=0 (PCSrc=0), Op2Type=0, ResultSelector[2..0]=000 и DMemWrite=1. **Влез во инструкциска меморија:** Halt = 0, MemoryOperands[2..0]=010, CondBranch=0 и UncondBranch=0 (PCSrc=0), Op2Type=0 и IMemWrite=1.

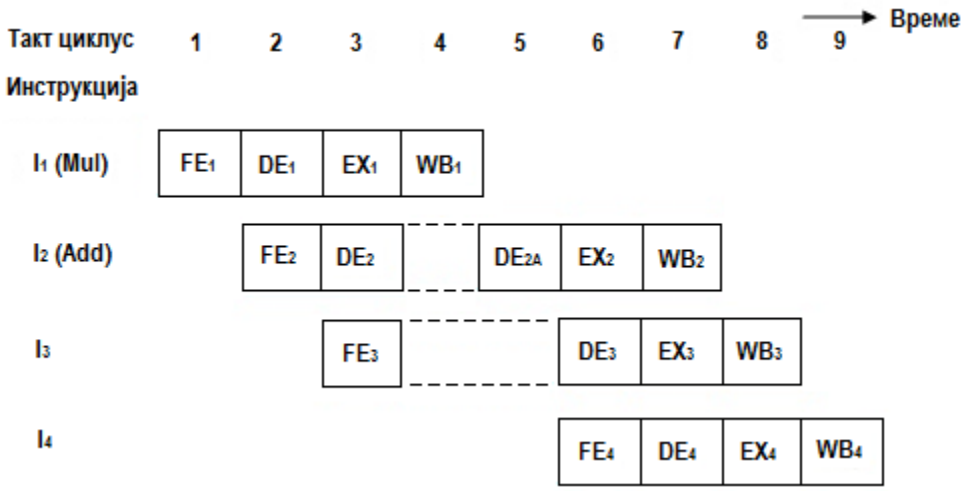
Инструк-циски тип	Контролни сигнали кои се поставени (останатите контролни сигнали имаат неопределена вредност x)
OUT	**Излез од податочна меморија:** Halt = 0, IMemRead=0, MemoryOperands[2..0]=110, DMemRead=1, CondBranch=0 и UncondBranch=0 (PCSrc=0), Op2Type=0. **Излез од инструкциска меморија:** Halt = 0, IMemRead=1, MemoryOperands[2..0]=010, DMemRead=0, CondBranch=0 и UncondBranch=0 (PCSrc=0), Op2Type=0.
CTRL	**HLT:** Halt = 1, CondBranch=0 и UncondBranch=0 (PCSrc=0). **NOP:** Halt = 0, CondBranch=0 and UncondBranch=0 (PCSrc=0) **WritetoInstructionPageTable:** Halt = 0, CondBranch=0 и UncondBranch=0 (PCSrc=0) и IMemPTWrite=1. **WritetoDataPageTable:** Halt = 0, CondBranch=0 и UncondBranch=0 (PCSrc=0) и DMemPTWrite=1

4.5. Разрешување на конфликти и справување со исклучоци

Применетиот пристап на проточно работење кај MIMOPS процесорот воведува проблеми, наречени конфликти, кои што го оневозможуваат нормалното функционирање на следната инструкција од инструкцискиот ток во наредниот такт циклус (предизвикуваат застои). Типовите на конфликти кои можат да се појават при проточно работење се: структурни, податочни и контролни зависности. Структурните конфликти се јавуваат кога хардверот не може да поддржи паралелно извршување на инструкции во ист такт циклус. Со оглед на тоа дека MIMOPS процесорот е дизајниран да работи проточно, кај него не се јавуваат никакви структурни конфликти. Сепак, доколку MIMOPS процесорот е дизајниран да работи со една заедничка меморија за инструкции и податоци, во тој случај би се јавила структурна зависност (процесорот нема да може истовремено да зема инструкција и да пристапува до податоци).

Податочни зависности се јавуваат кога една инструкција зависи од резултатот од претходната инструкција, а при тоа двете инструкции се извршуваат паралелно. Овој проблем резултира со застој во проточната линија, кој трае се додека резултатот од првата инструкција не стане расположлив. Ваквата ситуација е прикажана на сл. 44, каде е претставен ризикот кој се јавува при читање после запишување (read after write) помеѓу инструкција1 - I_1 и инструкција2 - I_2. Според даденото сценарио, проточната линија се успорува за два такт циклуса, како резултат на податочната зависност помеѓу I_2 која врши читање на некој

операнд во DE_2 фазата и I_1 која врши запишување на истиот операнд во WB_1 фазата. Сето тоа предизвикува да дојде до одложување во извршувањето на секоја следна инструкција (I_2, I_3 и I_4), за два такт циклуса.



Слика 44 Податочни зависности во проточна линија на MIMOPS процесор.

Едно можно решение за ваков вид на податочни конфликти е препраќањето (forwarding), односно непосредното проследување на резултатниот операнд кон инструкцијата која зависи од него, веднаш штом ќе биде пресметан. За сценариото кое е дадено на сл. 44 ова би значело дека WB_1 фазата веднаш би го испратила резултатот кон DE_2 фазата, па на тој начин DE_2 фазата ќе може веднаш да го користи овој операнд, без при тоа да се предизвика застој во проточната линија.

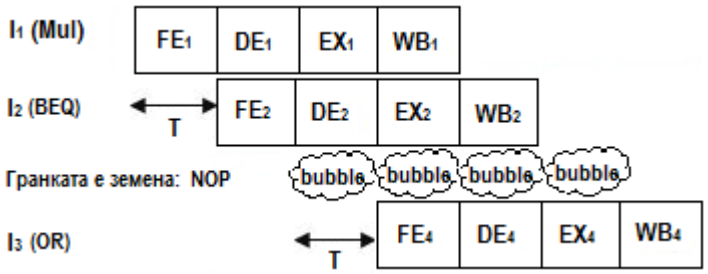
За да се справи со податочни конфликти, MIMOPS процесорот имплементира препраќање на резултатот во фазите за извршување и запишување резултат (види сл. 27). Ова подразбира проследување на вредноста на резултатот, адресата на резултатот и DMemWrite контролниот сигнал од дадените (EX или WB) фази кон фазата за декодирање, каде се врши разрешување на податочни конфликти со две посебни единици за детекција на податочни конфликти (за операнд1 и операнд2). Овие единици проверуваат дали пристапот до операнд1 или операнд2 предизвикува ризик од читање после запишување. Доколку постои ризик, препратениот резултат треба да се употреби за поставување на вредноста на операнд1 или операнд2, соодветно. Ова однесување (пр. детекција на податочни конфликти за операнд1), може да се опише со следниот програмски код:

```
If ((DMemWrite_ex = '1') AND (Write_Address_ex = Addressop1)) then
      Forward_A <= "01";  -- EX HAZARD
elsif ((DMemWrite_wb = '1') AND (Write_Address_wb = Addressop1)) then
      Forward_A <= "10";  -- WB HAZARD
else
      Forward_A <= "00";  -- NO HAZARD
```

Кога станува збор за податочни конфликти, може да се каже дека MIMOPS процесорот не страда од појава на конфликти при вчитување на податоци од меморија во регистер (load-use hazards). Овој тип на податочен конфликт најчесто се јавува кај RISC-базираните процесори (пр. MIPS), кога некоја инструкција се обидува да прочита некој регистер, кој претходно треба да се постави на вредност прочитана од меморија (со load инструкција). Со оглед на тоа дека MIMOPS процесорот работи директно со податочната меморија во чипот (без експлицитни load и store инструкции), овој конфликт не претставува никаков проблем за процесорите кои имплементираат MEMRISC архитектура.

Последниот тип на конфликти кои се јавуваат кај MIMOPS процесорот се контролните конфликти, кои уште се познати и како конфликти при гранење. Овие конфликти настануваат кога треба да се донесе одлука врз основа на резултатот од една инструкција, додека останатите инструкции продолжуваат со извршување. На пример, инструкцијата на гранење при еднаквост (BEQ - branch on equal) ќе изврши гранење кон некој дел од инструкциската меморија, доколку двата операнда кои се прочитани од податочната меморија во чипот се еднакви. Всушност, додека трае споредбата на податоците, процесорот продолжува со земање на други инструкции (fetch). Според тоа доколку гранката е земена, погрешни инструкции се вчитани во проточната линија и истите мора некако да бидат отстранети. Најчесто решение на овој проблем е да се продолжи со извршување на инструкциите како гранката да не била земена. Тоа значи дека доколку подоцна се открие дека гранката треба да се земе, тогаш инструкциите кои биле земени треба да се отстранат, што пак се постигнува со чистење (flushing) на IF/ID проточниот регистер. Овој процес (чистење) значи дека сите вредности кои се сместени во проточниот регистер се поништуваат т.е. ресетираат, па инструкцијата која е погрешно земена се извршува како NOP операција, како што е прикажано на сл. 45. Со оглед на тоа дека MIMOPS процесорот го

извршува условното гранење во фазата на декодирање, застојот при гранење се намалува на 1 такт циклус (MIPS процесорот врши условно гранење во фазата на извршување, предизвикувајќи застој од 2 такт циклуса). Во случај кога гранката не е земена, не настанува никаков застој кај MIMOPS процесорот.



Слика 45 Контролна зависност при условно гранење во MIMOPS процесор.

MIMOPS процесорот обезбедува поддршка за справување со исклучоци кои настануваат при извршување на ALU операции (пр. претекување на резултат или делење со нула). Наједноставен начин за справување со овие исклучоци е да се направи ресетирање на DMemWrite контролниот сигнал за ALU инструкцијата која го предизвикала исклучокот и на тој начин да се спречи истата да направи промени во податочната меморија во чипот. Покрај овие исклучоци со ALU операции кои се јавуваат во фазата за извршување, MIMOPS процесорот може да се справи и со промашувања (page fault) во инструкциската или податочната меморија во чипот, кои настануваат во фазата на земање или декодирање, соодветно. Овие настани се сигнализираат со посебни сигнали кои определуваат дали промашувањето настанало при пристап до инструкција, операнд1, операнд2 или резултат. Основната задача која треба да ја направи процесорот кога ќе настане исклучок е да ја зачува адресата на инструкцијата која го предизвикала исклучокот на предефинирана адреса во податочната меморија во чипот и потоа да ја предаде контролата на оперативниот систем на некоја специфична адреса (до рутина за опслужување). Веднаш потоа оперативниот систем може да извршува некои акции, како што се: обезбедување на некаков сервис за корисничката програма, извршување на предефинирана акција како одговор на исклучок на претекување, или стопирање на извршувањето на програмата и репортирање на грешка. Откако ќе се изведе соодветната акција за исклучокот, оперативниот систем може да ја заврши програмата или да продолжи со нејзино извршување, користејќи ја адресата за враќање назад која определува каде треба да се ре-

стартира извршувањето на програмата. На пример, кога ќе настане промашување во меморијата, процесорот продолжува со извршување откако побараната страница ќе се смести во меморијата на чипот на MIMOPS процесорот.

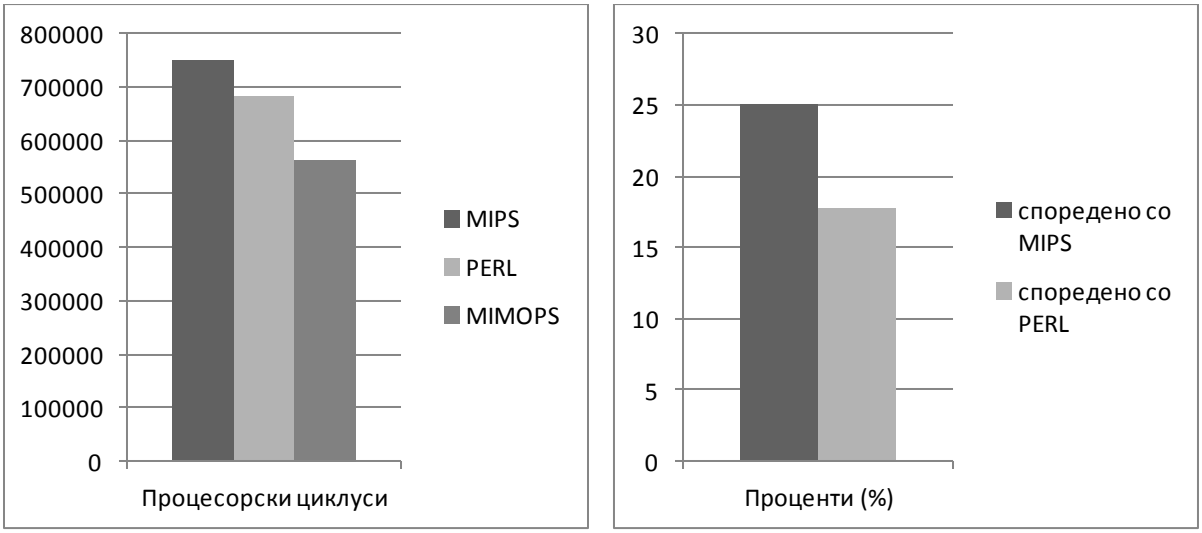
Покрај исклучоци, MIMOPS процесорот обезбедува и поддршка за опслужување на прекини кои се предизвикани од надворешни уреди. Ваков тип на настан се случува кога InterruptRequest сигналот е поставен и при тоа процесорот е подготвен да одговори на барањето. Во тој случај, зависно од бројот на прекинот, процесорот ја предава контролата на оперативниот систем на некоја специфична адреса, каде се наоѓа рутината за опслужување на прекинот.

4.6. Дискусија за очекувани подобрувања и можни проблеми

Новата RISC-базирана мемориско-центрична архитектура - MEMRISC, во главно се базира на идејата дека интеграцијата на меморија во процесорски чип ќе обезбеди директен и побрз пристап до мемориските податоци. Според тоа, се очекува дека со замена на врвот од мемориската хиерархија (регистри за општа намена и кеш) со меморија до која може директно да се пристапува во чипот, ќе се избегнат непотребните операции на движење и копирање на податоци во кешот и регистрите за општа намена и на тој начин ќе се заштедат многу временски и хардверски ресурси. Главните три придобивки од овој пристап би биле: отстранување на комплексни механизми за одржување на кеш меморијата, отфрлање од употреба на експлицитни инструкции за читање и запишување, и отстранување на многу редундантни копии на податоците. Сето тоа е особено погодно за апликации кои извршуваат многу аритметичко-логички операции врз податочно множество до кое се пристапува со висок степен на локалност. Пример за ваков тип на апликации се оние кои извршуваат матрични пресметки, како што е множење на матрици.

За да се претстават перформансните придобивки (во однос на брзина) на новата MEMRISC архитектура, направена е споредба помеѓу три слични процесори. Станува збор за MIMOPS процесор кој имплементира MEMRISC архитектура (работи без кеш и регистри за општа намена), безрегистарски процесор PERL и стандарден RISC-базиран MIPS процесор. Со оваа компаративна анализа се мери

времето на извршување на програма за множење на 32x32 матрици за секој од трите дадени процесори. Симулирањето на програмата се врши со посебен симулатор на инструкциско ниво кој е развиен за MIMOPS процесорот (и подоцна е објаснет во глава 7), MARS симулатор за MIPS процесор, [75], и посебен симулатор на инструкциско ниво кој е развиен за PERL процесорот (претставен во [33]). Резултатите од компаративната анализа се прикажани на сл. 46, каде на сл. 46 а) е претставено времето на извршување на тест програмата, додека пак на сл. 46 б) е илустрирано подобрувањето кое го постигнува предложениот MIMOPS процесор. Според овие резултати, може да се забележи дека PERL процесорот постигнува подобрување од 8,82% во споредба со MIPS, но од друга страна MIMOPS процесорот ги надминува перформансите и на двата, постигнувајќи 1,33 пати (25%) подобри резултати од MIPS, и 1,21 пати (17,7%) подобри резултати од PERL. Оваа анализа е направена само за да го прикаже потенцијалот за остварување на високи перформанси на MIMOPS процесорот. Понатамошна и подетална анализа, која вклучува и други тест сценарија, е дадена во глава 7.



а) Време на извршување. б) Подобрување на MIMOPS.

Слика 46 Анализа на извршување на програма за множење на 32x32 матрици во три различни процесори: MIPS, PERL и MIMOPS.

И покрај тоа што MIMOPS процесорот постигнува побрзо сметање, сепак истиот се соочува со некои проблеми кои би требало да се разгледаат. Пред сè, пристапот на директно обраќање до меморијата на чипот во MIMOPS процесорот побарува креирање на архитектура на инструкциско множество (ISA) која е едноставна и слична на RISC, но се интерпретира на поинаков начин. Понатаму ова

повлекува потреба за креирање на: специфична контролна единица која ќе може соодветно да ги декодира инструкциите, и специјализиран компајлер кој ќе може да преведува програми од јазик на високо ниво во MIMOPS асемблер. Тоа значи дека и покрај тоа што MIMOPS процесорот е иницијално дизајниран како модификација на MIPS процесор, сепак неговата MEMRISC архитектура во многу сегменти се разликува од RISC архитектурата. Тука спаѓаат: пристапот до меморија, адресирањето на податоци, организацијата на меморија во чип, интерпретирањето на инструкции, компајлерската поддршка итн.

Покрај потешкотиите со дизајнирање на нов специфичен софтвер (компајлер) од нула, друг проблем кај MIMOPS процесорот е тоа што тој воведува зголемена комплексност во хардверот на определени критични точки, во споредба со MIPS процесор. На пр. фазите на земање и декодирање во MIMOPS процесор вклучуваат дополнителни компоненти (пр. генератори на адреси за меморијата кои содржат табели на страници), меморија во чип и покомплексна контролна единица. Сепак, доколку се земе во предвид дека регистрите за општа намена и кеш меморијата се отстранети од мемориската хиерархија заедно со комплексноста за нивно одржување, тогаш може да се каже дека хардверската комплексност на MIMOPS процесорот нема да се зголеми многу, во споредба со процесор кој имплементира неколку МБајти кеш меморија во чипот (пр. IBM POWER7).

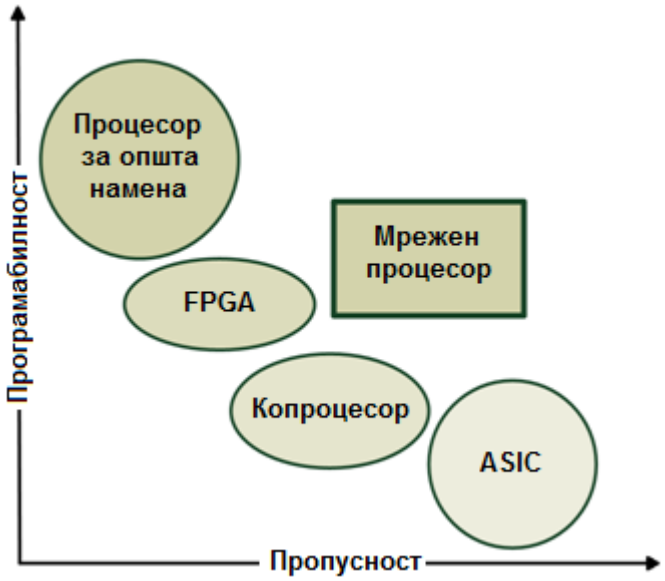
Кога станува збор за количината на меморија во чипот може да се забележи дека овој параметар најмногу е ограничен од технологијата на имплементација. Согледувајќи го тековниот развој на мемориската архитектура и технологија (пр. вграден DRAM или мемристор технологија) станува многу веројатно дека во блиска иднина би можело да се дизајнира скалабилен повеќе-процесорски MIMOPS-базиран систем. Овој тип на систем би имплементирал дистрибуирана споделена виртуелна меморија, каде големината на меморија во чипот не би претставувала проблем, па системот би бил пошироко применлив. Сепак, MIMOPS процесорот така како што е дизајнирани (со ограничена количина на меморија во чип) би можел да се употреби за ограничено множество на апликации, каде на пример брзината на процесирање е од голема важност. Како резултат на тоа, во следното поглавје е дискутирана примената на предложениот MIMOPS процесор во мрежно процесирање со висока пропусност (неколку Gb/s).

5. ПРОШИРУВАЊЕ НА ПРЕДЛОЖЕНИОТ RISC-БАЗИРАН МЕМОРИСКО-ЦЕНТРИЧЕН ПРОЦЕСОР (MIMOPS) ЗА ПРИМЕНА ВО МРЕЖНО ПРОЦЕСИРАЊЕ

Брзиот развој на Интернет мрежата предизвика огромно зголемување во бројот на корисници, сервери, врски, како и барања за нови апликации, сервиси и протоколи во модерните повеќе-гигабитни компјутерски мрежи, [87]. Ваквата еволуција доведе до неколку-кратно зголемување на количината на мрежниот сообраќај само во последните неколку години, во однос на вкупните досегашни разменети податоци од појавата на Интернетот до пред неколку години. Со развојот на технологијата, капацитетот на мрежните линкови продолжи да се зголемува (особено со напредокот на оптичките комуникации), [88], но од друга страна мрежниот хардвер и софтвер се соочија со многу потешкотии за временски да ги задоволат новите барања за висока пропусност и брзина, и воедно да овозможат мали доцнења, [89]. Според тоа, може да се каже дека зголемениот сообраќај предизвика да мрежниот хардвер стане тесно грло при конструирање на брзи гигабитни мрежи.

Скоро сите мрежни уреди, вклучувајќи ги Ethernet комутаторите (анг. switch), IP рутери, веб сервери или хардверски заштитни ѕидови, извршуваат некакво мрежно процесирање на пакети. Во минатото, мрежното процесирање било имплементирано со софтвер кој се извршувал на процесор за општа намена (GPP - General purpose processor). Ова се должи на тоа што во тој временски период барањата за перформанси не биле многу големи и воедно мрежните протоколи кои се употребувале биле прилично едноставни. Со развојот на технологијата, процесорите за општа намена не можеле да ги задоволат новите барања за високи податочни рати на процесирање, па мрежните инженери решиле да развијат хардверски-базирани решенија со примена на интегрирани кола за специфична намена (ASICs), [90]. И покрај тоа што ASIC колата успеале да постигнат високи процесирачки брзини, сепак тие биле направени за специфична намена, без можност да им се направи било каква промена после дизајнирањето. Тоа значи дека ASIC колата не би можеле да обезбедат репрограма-

билност. Сето тоа предизвика понатамошен развој на нови технологии (систем во чип - SoC, програмабилна FPGA логика или комплексни програмабилни логички уреди - CPLD), кои воведоа поголема флексибилност и модуларност при дизајнирањето на процесори. Во тој контекст, на сл. 47 е дадена споредба на различни технологии кои би можеле да се применат при имплементирање на хардвер за мрежно процесирање.



Слика 47 Споредба на различни технологии за имплементирање на мрежно процесирање.

Најчесто користен хардвер, кој е оптимизиран за обработка на пакети со големи брзини е мрежниот процесор (NP - Network processor). Генерално, мрежните процесори се дефинираат како уреди на чип кои може да се програмираат, и кои се специјално дизајнирани да овозможат мрежно процесирање на пакети со повеќе-гигабитни брзини, [89] - [91]. Тие вообичаено се имплементираат како ASIP (Application specific instruction processors) процесори со специфични и специјално дизајнирани инструкциски множества, кои се базираат на RISC, CISC, VLIW или некоја друга архитектура на инструкциско множество, [89]. Досега овој вид на процесори се имаат покажано како најдобро решение за примена во мрежно процесирање, бидејќи според сл. 47 тие обезбедуваат добар баланс помеѓу флексибилноста на процесорите за општа намена и големата брзина на обработка која ја постигнуваат ASIC уредите. Како резултат на тоа, мрежните процесори се широко користени во најразновидни мрежни уреди, како што се: рутери, комутатори, заштитни ѕидови или системи за детекција на упади (IDS).

Во однос на полето на истражување на мрежните процесори како уреди, може да се каже дека досега е предложен широк спектар на архитектури на мрежни процесори. И покрај тоа што не постои стандардна архитектура на мрежен процесор, повеќето дизајни на мрежни процесори споделуваат некои заеднички карактеристики кои на некој начин ги прават слични. Според тоа, може да се генерализира дека секој мрежен процесор вклучува: повеќе процесирачки јадра, специфични хардверски забрзувачи на пресметки (копроцесори или функционални единици), разновидна организација на меморискиte ресурси, механизми за поврзување, техники за хардверска паралелизација (пр. проточност) и софтверска поддршка, [90], [91]. Развојот на архитектурата на мрежните процесори е сè уште актуелно подрачје на истражување, кое во блиска иднина ќе предизвика голем напредок во индустријата за производство на мрежни процесори. Всушност, само во текот на последната деценија голем број на компании развија свои сопствени мрежни процесори, (Интел, Агере, IBM итн). Со оглед на тоа дека ова истражувачко поле брзо се развива, постојано се појавуваат и нови идеи, како што се: NetFPGA архитектурата [92], или софтверските рутери [93].

Најпопуларните мрежни процесори кои се користат денес вклучуваат едно или повеќе идентични или различни процесирачки јадра кои работат паралелно. На пример, процесорот IXP2800 на Интел, [94], содржи 16 идентични повеќе-нитни RISC процесори за општа намена, организирани како базен од паралелни хомогени процесирачки јадра, кои лесно можат да се програмираат и да се прилагодуваат кон промените на сервисите и протоколите. Дополнително, компанијата EZChip го има воведено првиот мрежен процесор со 100 ARM кеш-кохерентни програмабилни процесирачки јадра, [95], кој меѓу другото претставува најголемиот 64-битен ARM процесор кој е дизајниран до сега. Покрај ARM процесорските јадра за општа намена, овој чип вклучува и мрежа за поврзување помеѓу секое јадро (анг. mesh core interconnect), која обезбедува голема пропусност, мало доцнење и висок степен на скалабилност.

Претставените мрежни процесори укажуваат дека поголемиот дел од мрежното процесирање најчесто се извршува со RISC-базирани процесирачки јадра за општа намена кои претставуваат поефтино, но поспоро решение, во комбинација со специфични хардверски блокови (пр. за управување на сообраќај

или брзо пребарување во табели) кои се поскапи но истовремено и поенергетско ефикасни и побрзи. Според тоа, доколку мрежното процесирање се анализира на општо-наменски процесорски јадра, тогаш едноставно е да се заклучи дека поголемиот дел од процесорските циклуси се трошат за пристап до полиња од заглавијата на пакетите (т.е. при парсирање на заглавијата на пакетите), особено кога полињата од заглавијата не се со должина на збор. Во тој случај потребно е да се извршат логички или аритметички операции за да се издвои вредноста на полето од заглавието на пакетот, кое треба да се обработува.

Во оваа глава се предлага модификација на MIMOPS процесорот, со која се овозможува овој тој да стане применлив во мрежно процесирање. Според тоа, за да може да процесира IP заглавија, MIMOPS процесорот ги складира IP заглавијата во својата податочната меморија во чипот и потоа директно пристапува до зборовите од IP заглавијата, без да користи регистри за општа намена и кеш меморија. Покрај тоа, податочната меморија во чипот е надградена со специфичен хардвер кој врши екстрахирање на полињата директно на нејзиниот излез, пред да го препрати полето кон ALU единицата. На овој начин се избегнува извршување на логички или аритметички операции, потребни за да се издвои вредноста на полето од IP заглавието, кое не е со должина на збор. Повеќе детали за овој предлог се дадени во следните подглавја.

5.1. Пристапи за подобрување на мрежното процесирање

Развојот на мрежните процесори започнува кон крајот на 1990<u>тите</u> години, во период кога мрежните уреди повеќе не можеле да ги опслужат комплексните барања за мрежно процесирање. Генерално, мрежните процесори се применуваат за извршување на брзи обработки и пребарувања во податочната рамнина (data plane т.е. делот од рутерите кој препраќа пакети), додека пак бавната обработка на пакети (контрола, обликување на сообраќај, класи на сообраќај, рутирачки протокол) вообичаено се извршува од процесор за општа намена во контролната рамнина. Соодветно на тоа може да се каже дека мрежното процесирање кое се извршува во мрежниот уред е поделено во две категории: брза и спора обработка на пакети во податочната и контролната рамнина, соодветно.

Мрежниот процесор започнува со својата работа при прием на влезен тек на податочни пакети од физичкиот интерфејс. Неговата задача е да ги провери, обработи, анализира и евентуално промени IP заглавијата на примените пакети. При обработката на пакетите може да се употребат некои специјализирани хардверски единици за извршување на конкретни задачи, како на пример: класификација на пакети, пребарување во табели, препраќање, управување со редици и контрола на сообраќај, [89]. Откако ќе заврши мрежното процесирање, пакетите се испраќаат преку прекинувачката мрежа (switching fabric) кон соодветните излезни порти на мрежниот уред.

Според истражувањата кои се направени во [73], обработката на пакетите може да се забрза доколку се поедностават некои спори мрежни процесирачки операции и при тоа се направи соодветен избор на функционалностите на рутирачките протоколи. За таа цел, досега се предложени повеќе различни пристапи како што се: примена на концептот на лабели за забрзување на операциите на пребарување или употреба на специјализирани хардверски единици за извршување на некои комплексни и спори операции како што е на пр. пресметување на сума на проверка за заглавие. Дополнително, предложени се и неколку алгоритми за побрзо пребарување во рутирачките табели, кои се претставени во [96] и [97]. Од друга страна, авторите на истражувањето дадено во [73] предлагаат употреба на опцијата за изворно рутирање на IP протоколот со цел да се избегнат спорите операции на пребарување во рутирачки табели. Друг сличен пристап е исто така претставен во истражувањето, дадено во [72].

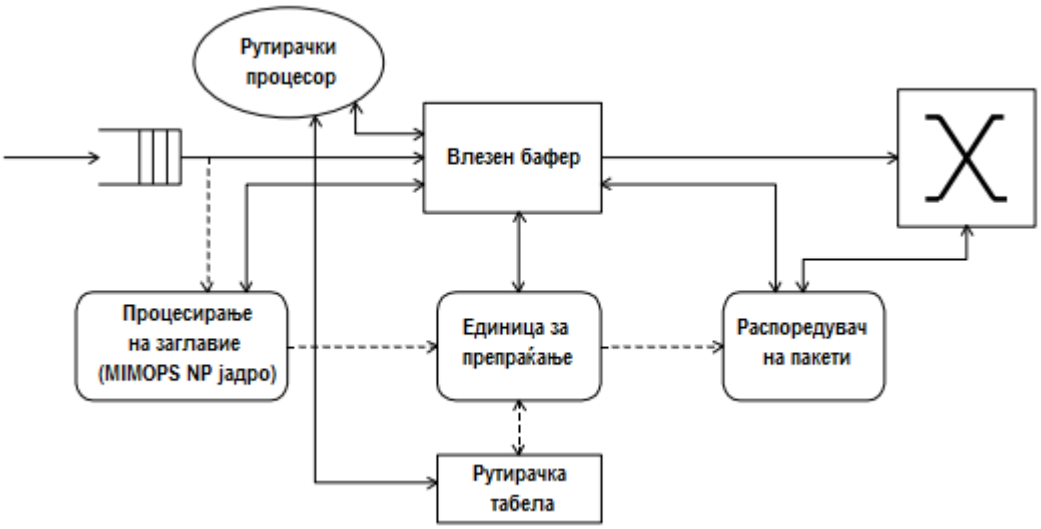
Генерално може да се каже дека мрежниот софтвер се доближува до мрежниот хардвер, како што е претставено во [98], каде се предлага дел од процесирањето на пакетите да се имплементира хардверски со копроцесори кои се користат и контролираат софтверски. На овој начин, хардверот извршува поголем дел од обработката на пакетот, додека пак определени покомплексни и специфични анализи на мрежниот сообраќај се извршуваат од страна на процесор за општа намена. Тоа овозможува да се изгради високо-пропусен флексибилен систем за мрежно процесирање. Покрај ова, определени истражувачи се обидуваат да го унифицираат погледот кон мрежниот хардвер и копроцесорите, со развој на едно заедничко апстрактно ниво за развој на мрежен софтвер, [99].

Кога станува збор за парсирање на мрежни пакети, може да се каже дека поголемиот дел од предложените пристапи ја користат FPGA технологијата, бидејќи таа е особено погодна за имплементација на проточни архитектури и воедно е идеална за мрежно процесирање со високи брзини, [100]. Тоа овозможува примена на реконфигурабилните FPGA плочки за дизајнирање на флексибилни повеќе-процесорски системи кои ќе можат сами да се прилагодуваат кон тековните протоколи и карактеристики на мрежниот сообраќај. Овој пристап е прикажан во [101], каде авторите предлагаат употреба на едноставен PP јазик на високо ниво за опис на алгоритмите за парсирање на пакети, на начин независен од имплементацијата. Слично на тоа, во истражувањето претставено во [102] се користи PX јазик за опис на типот на мрежното процесирање кое треба да се изврши во системот и потоа со специјална алатка се генерира целосен мулти-процесорски систем во RTL опис. Ваквиот системот може да се мапира во FPGA платформа и потоа динамички да се реконфигурира.

5.2. Прилагодување на MIMOPS процесорот за мрежно процесирање

MIPS процесорот кој е иницијално употребен за дизајнирање на MIMOPS процесорот имплементира RISC архитектура за општа намена. Како што беше претходно спомнато, мрежните процесори можат да се базираат на општонаменско процесирање, но сепак мора да обезбедат и некои функционалности за специфична намена, како што се: обработка на IP заглавија, класификација и рутирање, управување со редици на чекање и препраќање на пакети. Како резултат на тоа, во продолжение е претставен пристап со кој предложениот MIMOPS процесор се прилагодува да работи како мрежно процесирачко јадро, кое директно ги процесира заглавијата на мрежните пакети кои се сместени во податочната меморија во чипот, без употреба на регистри за општа намена и кеш меморија. Дополнително, прилагоденото MIMOPS мрежно процесорско јадро (network processor core) имплементира и посебна хардверска логика за парсирање на IP заглавија, која обезбедува директен пристап до полињата од IP заглавијата кои не се со должина на збор во податочната меморија на чипот.

Прилагоденото MIMOPS мрежно процесирачко јадро има дизајнирано карактеристики за брза обработка и препраќање на IP пакети во рамките на податочната рамнина на еден рутер. Генерално, мрежното процесирање во рутерот започнува кога мрежните пакети ќе пристигнат преку MAC (Medium Access Control) мрежниот интерфејс и потоа ќе се зачувуваат во меморијата, како што е прикажано на сл. 48. Еден пристап за нивна обработка, [103], е пакетите да се снимат директно во влезниот бафер, а потоа заглавјата да се читаат и обработуваат оттаму. Втора алтернатива, [103] е пакетите да поминат низ хардверот за обработка на заглавието, пред да се копираат во баферот. Прилагоденото MIMOPS мрежно процесирачко јадро користи комбинација од двата пристапи: при доаѓањето на пакетите, нивното заглавје се запишува во брзата податочна меморија во MIMOPS мрежно процесорско јадро, додека пак содржината на пакетот се сместува во поголем, но побавен бафер. На сл. 48 е даден блок дијаграм на систем за мрежно процесирање кој вклучува прилагодено MIMOPS мрежно процесирачко јадро во својата архитектура.



Слика 48 Блок дијаграм на систем за мрежно процесирање (пр. рутер).

Според сл. 48 може да се забележи дека прилагоденото MIMOPS мрежно процесирачко јадро е задолжено само за обработка на IP заглавијата на примените пакети. Процесирањето на IP заглавијата вклучува повеќе операции: валидација на специфични полиња од IP заглавието, пресметка на сума за проверка, намалување на времето на живот на пакетот (TTL - Time to Live) и проверка на IP опции. По обработката на IP заглавјето, следи препраќањето на пакетот врз

основа на неговата дестинациска IP адреса, со помош на единицата за препраќање (forwarding engine). Задачата на оваа единица е да ја пребара табелата за препраќање и да определи кон кој следен јазел (хоп) треба да се проследи пакетот. Табелата за препраќање ги содржи оние рути кои најчесто се користат при препраќањето, а додека пак комплетното множество на сите рути кои ги учи рутерот се сместени во неговата рутирачка табела. За ажурирањето на рутирачката табела се грижи посебен рутирачки процесор. Откако ќе заврши пребарувањето за дадената IP адреса во табелата за препраќање, се избира излезната порта и пакетот е подготвен за препраќање. Оттука работата ја превзема распоредувачот на пакети, кој управува со текот на пакетите и одлучува кои пакети треба да се испратат кон прекинувачката мрежа (switching fabric),

Прилагоденото MIMOPS мрежно процесирачко јадро ги сместува заглавијата на примените IP пакети во својата податочна меморија преку IN инструкции, и ги запишува обработените IP заглавија во пакетскиот бафер преку OUT инструкции. Пред да се извршат овие трансфери на податоци, MIMOPS процесорот треба да ја постави почетната адреса на IP заглавието, преку SetPR и SetBR инструкции со кои се селектира потребниот мемориски блок во податочната меморија на чипот и се ажурира вредноста на базниот регистар. Потоа, со помош на полето за должина на IP заглавие (IP header length) се определува бројот на 32-битни зборови кои треба да се прочитаат или запишат на последователни мемориски локации. Овој трансфер на податоци се изведува преку 32-битна влезно-излезна (INOUT) податочна магистрала, која овозможува комплетен пренос на IP заглавието после завршувањето на определен број IN/OUT инструкции.

Мемориско-центричниот пристап на имплементирање внатрешна меморија во прилагоденото MIMOPS мрежно процесирачко јадро овозможува комплетно складирање на секое IPv4 заглавие (заедно со опции) или IPv6 заглавие (заедно со заглавија за проширување) од пакетите кои се примени во мрежниот процесирачки систем. На тој начин се обезбедува директен пристап до податоците од IP заглавието, кои се обработуваат како 32-битни зборови во прилагоденото MIMOPS мрежно процесирачко јадро. Заради ова ограничување, MIMOPS мрежното процесирачко јадро троши повеќе процесорски циклуси во обработката на полиња од IP заглавието кои не се точно порамнети со должината на 32-битен

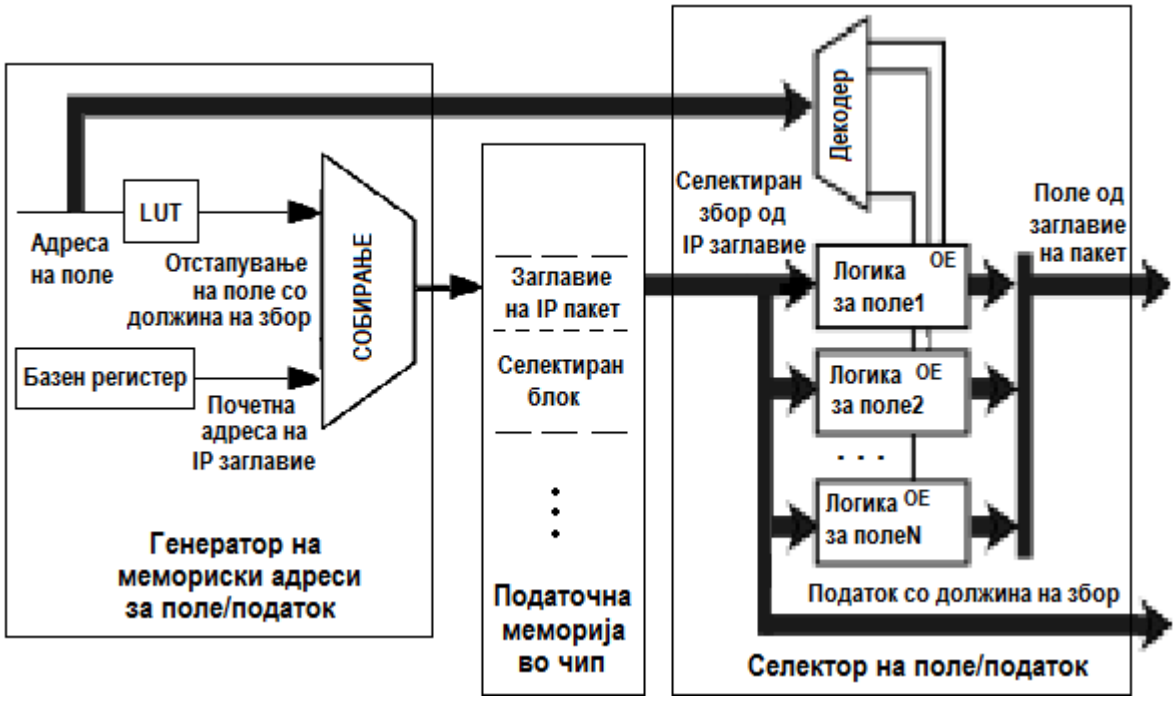
збор. Во таков случај, потребни се повеќе операции на поместување и логички битови операции за да се добие потребното IP поле од вчитаниот 32-битен збор. Како резултат на тоа, MIMOPS мрежното процесирачко јадро е надградено со посебна хардверска единица (селектор на податоци/полиња) која врши екстрахирање на полињата од IP заглавието, кои не се со должина на збор (пр. IP верзија, IHL - Internet Header Length, TTL итн.). На овој начин MIMOPS процесорот може засебно да пристапува до секое поле од IP заглавието, и при тоа може да го менува и запишува назад во податочната меморија на чипот со нови вредности, и тоа за време од само еден процесорски циклус. Дизајнот на хардверската логика за парсирање на IP заглавија (екстрахирање на полиња) е претставен во следното поглавје.

5.3. Проектирање на хардверска логика за парсирање на заглавија на IP пакети

Основната идеја за проектирање на посебна хардверска логика за парсирање на заглавија на IP пакети е да се обезбеди директен и едно-тактен пристап до секое поле од IP заглавијата. За да се постигне тоа, предложениот парсер на IP заглавија овозможува исто време на пристап до поле од заглавие на IP пакет како времето на пристап до било кој мемориски збор, дури и кога полето не е порамнето со должина на збор. Овој пристап има големо влијание врз мрежниот процесирачки хардвер, бидејќи се очекува дека тој ќе го забрза процесирањето на IP пакетите и со тоа ќе овозможи да се подобри податочната пропусност на мрежниот уред.

За да се постигне едно-тактен пристап, предложената логика за парсирање на IP заглавија користи дел од меморискиот адресен простор за директно да ги адресира различните полиња од IP заглавието на пакетот. Оваа техника на мемориски прекари (анг. memory aliasing) овозможува пристап до секое поле од IP заглавието преку посебни вредности на мемориски адресни отстапувања (адреси на поле). Кога таква адреса на поле ќе се проследи до единицата за парсирање на IP заглавија, истата го селектира соодветниот збор од меморијата, кој го содржи даденото поле. Веднаш потоа избраниот збор се обработува и во за-

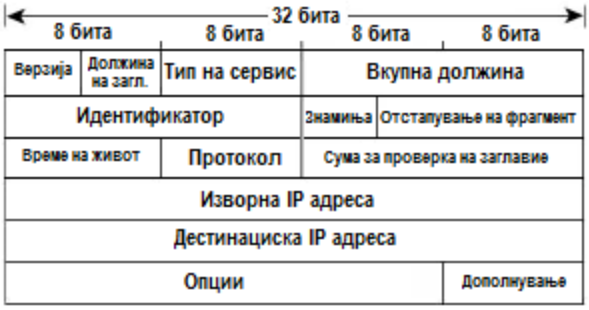
висност од адресата на полето, се екстрахира вредноста на полето. Овој процес може да вклучува некои операции на поместување на зборот и/или модификација на неговите битови. На сл. 49 е претставена шема на парсерот на заглавија на IP пакети, кој се користи при читање на некое поле од IP заглавие.



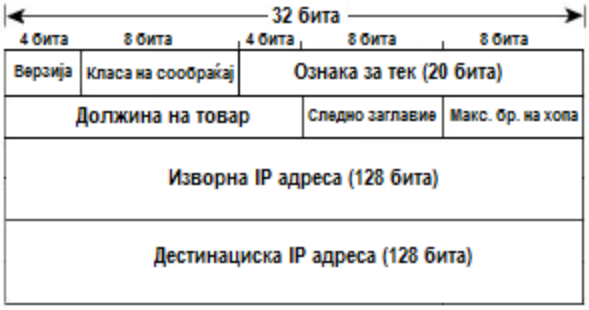
Слика 49 Читање на поле од IPv4/IPv6 заглавие со хардверска единица за парсирање на IP заглавија.

Хардверската логика за парсирање на IP пакети е дизајнирана така што се подразбира дека IPv4 или IPv6 заглавијата на пакетите кои треба да се обработуваат, се складирани во фиксен дел од податочната меморија на чипот. Описите на форматите на IPv4 и IPv6 заглавијата на пакети, кои можат да се обработуваат од парсерот на заглавија на IP пакети се прикажани на сл. 50. Во дадените описи на IP заглавија, првата линија го дефинира типот на IP заглавието и неговата локација во меморијата, додека пак секоја следна линија содржи дефиниција на по едно поле. Следствено на тоа, за секое поле се специфицирани неговото име и големина, изразена во битови. Според сл. 50 се забележува дека полињата се дефинирани според редоследот по кој се појавуваат во IP заглавието.

IPv4 заглавие



IPv6 заглавие



IPv4 Заглавие @ ПочетнаАдресанаIPv4Заглавие

Верзија 4

ДолжинанаЗаглавие 4

ТипнаСервис 8

ПрвЗборПрваПоловина 16 //се користи за IP checksum

ВкупнаДолжина 16

Идентификатор16

Знаменца 3

ОтстапувањенаФрагмент 13

ВторЗборВтораПоловина 16 // се користи за IP checksum

ВременаЖивот 8

Протокол 8

ТретЗборПрваПоловина16 // се користи за IP checksum

СумазаПроверканаЗаглавие 16

ИзворнаIPАдреса 32

ЧетвртЗборПрваПоловина 16 // се користи за IP checksum

ЧетвртЗборВтораПоловина 16 // се користи за IP checksum

ДестинацискаIPАдреса 32

ПеттиЗборПрваПоловина 16 // се користи за IP checksum

ПеттиЗборВтораПоловина 16 // се користи за IP checksum

IPv6 Заглавие @ ПочетнаАдресанаIPv6Заглавие

Верзија 4

КласанаСообраќај 8

ОзнаказаТек 20

ДолжинанаТовар 16

СледноЗаглавие 8

МаксималенБројнаХопови 8

ИзворнаIPАдреса 128

ДестинацискаIPАдреса 128

Слика 50 Опис на IPv4 и IPv6 заглавија.

Почетната адреса на IP заглавието која е специфицирана во описот на IP заглавието се користи за да се постави вредноста на базниот регистар во единицата за генерирање на мемориски адреси за поле/податок, која е дел од логиката за парсирање на IP заглавија. Оваа единица исто така на влез прима адреса на поле која се преведува во отстапување на поле со помош на табелата за пребарување (Lookup Table - LUT), која е прикажана во продолжение (табела 10). Отстапувањето на поле претставува отстапување со должина на збор од почетната

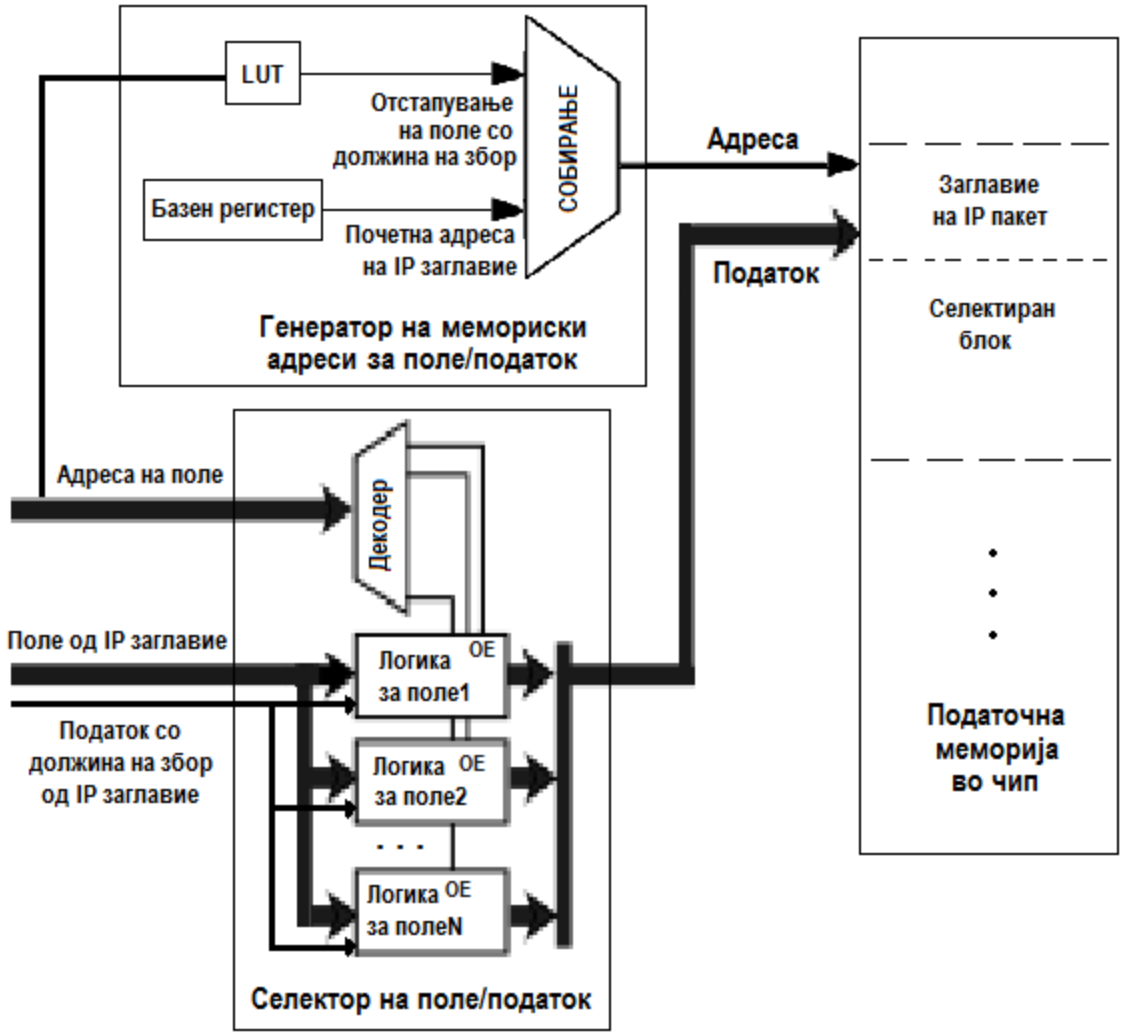
адреса на IP заглавието, и тоа покажува кон локацијата каде е сместено полето од IP заглавието во податочната меморија на чипот. Тоа значи дека доколку должината на некое поле е помала од должината на мемориски збор, тогаш во LUT табелата се сместува најблиското отстапување со должина на збор за даденото поле од IP заглавие.

Табела 10 Табела за пребарување (LUT) во парсер за заглавија на IP пакети.

Адреса на поле од IP заглавие (Мемориско адресно отстапување)	Отстапување на поле од IP заглавие со должина на збор
0000h (IPv4 Верзија)	0000h (прв збор во IPv4 заглавие)
0001h (IPv4 ДолжинаЗаглавие)	0000h (прв збор во IPv4 заглавие)
0002h (IPv4 ТипнаСервис)	0000h (прв збор во IPv4 заглавие)
0003h (IPv4 ПрвЗборПрваПоловина)	0000h (прв збор во IPv4 заглавие)
0004h (IPv4 ВкупнаДолжина)	0000h (прв збор во IPv4 заглавие)
0005h (IPv4 Идентификатор)	0001h (втор збор во IPv4 заглавие)
0006h (IPv4 Знаменца)	0001h (втор збор во IPv4 заглавие)
0007h (IPv4 ОтстапувањенаФрагмент)	0001h (втор збор во IPv4 заглавие)
0008h (IPv4 ВторЗборВтораПоловина)	0001h (втор збор во IPv4 заглавие)
0009h (IPv4 ВременаЖивот)	0002h (трети збор во IPv4 заглавие)
000Ah (IPv4 Протокол)	0002h (трети збор во IPv4 заглавие)
000Bh (IPv4 ТретЗборПрваПоловина)	0002h (трети збор во IPv4 заглавие)
000Ch (IPv4 СумазаПроверканаЗаглавие)	0002h (трети збор во IPv4 заглавие)
000Dh (IPv4 ИзворнаIPАдреса)	0003h (четврти збор во IPv4 заглавие)
000Eh (IPv4 ЧетвртЗборПрваПоловина)	0003h (четврти збор во IPv4 заглавие)
000Fh (IPv4 ЧетвртЗборВтораПоловина)	0003h (четврти збор во IPv4 заглавие)
0010h (IPv4 ДестинацискаIPАдреса)	0004h (петти збор во IPv4 заглавие)
0011h (IPv4 ПеттиЗборПрваПоловина)	0004h (петти збор во IPv4 заглавие)
0012h (IPv4 ПеттиЗборВтораПоловина)	0004h (петти збор во IPv4 заглавие)
0013h (IPv6 Верзија)	0000h (прв збор во IPv6 заглавие)
0014h (IPv6 КласанаСообраќај)	0000h (прв збор во IPv6 заглавие)
0015h (IPv6 ОзнаказаТек)	0000h (прв збор во IPv6 заглавие)
0016h (IPv6 ДолжинаноТовар)	0001h (втор збор во IPv6 заглавие)
0017h (IPv6 СледноЗаглавие)	0001h (втор збор во IPv6 заглавие)
0018h (IPv6 МаксималенБројнаХопови)	0001h (втор збор во IPv6 заглавие)

Според сл. 49, избраното отстапување на поле од LUT табелата се додава на почетната адреса на IP заглавието и на тој начин се генерира адресата на мемориският збор кој го содржи потребното поле од IP заглавието. Оваа адреса се користи за читање на дадениот збор од податочната меморија во чипот, кој потоа се проследува кон единицата за селектирање на поле/податок. Оваа единица вклучува посебни логички блокови за екстрахирање на различните полиња од IP заглавието (Логика за поле1 ... N). Тоа значи дека секое поле се екстрахира со посебен логички блок, кој се активира со сигнал за овозможување (output enable - OE), добиен како излез од декодер. Дадениот декодер на влез ја прима адресата на полето и на излез овозможува селекција на само еден логички блок во даден момент. Веднаш потоа, селектираниот логички блок извршува некои битови операции или поместувања со цел да го екстрахира и потоа дополни со нули соодветното поле од IP заглавието. Во случај кога полето од IP заглавието е со должина на збор, тогаш неговиот логички блок е празен и зборот се проследува директно од меморијата кон излезот на селекторот за поле/податок.

Логиката за парсирање на IP заглавија која е дадена на сл. 49 го прикажува хардверот кој се користи за да се прочита едно поле од IP заглавие од меморијата. Истиот концепт се користи за да се овозможи директно запишување во поле од IP заглавие во меморијата, што е конкретно прикажано на сл. 51. Според сл. 49 и сл. 51 може да се забележи дека двата модула користат идентична единица за генерирање на мемориски адреси за поле/податок, која генерира адреса на мемориският збор кој го содржи потребното поле од IP заглавието до кое треба да се пристапи. Всушност, единствената разлика помеѓу овие два модула е во единицата за селектирање на поле/податок, бидејќи логичките блокови од оваа единица примаат два влеза при запишување, и тоа: податок со должина на збор од IP заглавие кој бил прочитан од меморијата и поле од IP заглавие кое треба да се запише во меморијата. За да се овозможи запишување на потребното поле, декодерот активира само еден логички блок со помош на влезната адреса на полето. Овој логички блок го поставува влезното поле од IP заглавието на соодветна позиција во влезниот податок со должина на збор од IP заглавието. Веднаш потоа целиот збор, заедно со запишаното поле од IP заглавието се сместува на генерираната адреса во податочната меморија на чипот.



Слика 51 Запишување на поле од IPv4/IPv6 заглавие со хардверска единица за парсирање на IP заглавија.

Логиката за парсирање на IP заглавија на пакети е едноставна за дизајнирање, ако се земе во предвид дека форматите на заглавија на пакети кои треба да бидат поддржани се добро дефинирани. Со оглед на тоа дека предложената единица е наменета да работи само со IP заглавија на пакети, овозможувањето на поддршка за други формати на заглавија на пакети би побарувало проширување на LUT табелата и дефинирање на нови логички блокови во селекторот за поле/податок. Покрај флексибилноста во можностите за надградување, неопходно е да се истакне дека претставениот хардвер за директен пристап до полиња од IP заглавија ќе воведе и многу побрзо процесирање на пакетите во споредба со општо-наменското процесирање кое се користи во скоро сите мрежни процесори. Подетална анализа за тоа е дадена во следното поглавје.

5.4. Проценка на подобрување во брзина на парсирање на IP заглавија

Со цел да се процени подобрувањето кое се постигнува со употреба на предложениот парсер на IP заглавија при обработка на заглавија на IPv4 и IPv6 пакети, направена е споредба помеѓу MIPS и MIMOPS процесори без и со логика за парсирање на IP заглавија. Деталниот опис на употребените MIPS и MIMOPS процесори е даден во глава 4. Проширените верзии на MIPS и MIMOPS процесорите вклучуваат логика за парсирање на IP заглавија која е додадена до кеш и податочната меморија на чипот, соодветно. Во анализата се врши споредба на брзината на парсирање на IP заглавија (за IPv4 и IPv6), од страна на двата процесори кои работат без и со логика за парсирање на IP заглавија, [65].

На сл. 52 се прикажани асемблерски програми за обработка на IPv4 заглавие со MIPS и MIMOPS процесори, без и со логика за парсирање на IP заглавија. Дадените програми вршат парсирање на IPv4 заглавие, со екстрахирање на неговите полиња: верзија, должина на заглавие, тип на сервис, вкупна должина, идентификатор, знаменца, отстапување на фрагмент, време на живот, протокол, сума за проверка на заглавие, изворна IP адреса и дестинациска IP адреса. Покрај тоа, се врши екстрахирање и на некои полиња кои не се стандардни IPv4 полиња (пр. прв збор прва половина, втор збор втора половина, итн.), но се наменети да го поедностават пресметувањето на сумата за проверка на заглавието. Екстрахирањето на полињата генерално вклучува логички битови операции и поместувања. MIPS процесорот поддржува извршување на аритметичко и логичко поместување во лево или десно, кое се дефинира со посебна инструкција во која вредноста за поместување се задава како 5-битна константа. Од друга страна MIMOPS процесорот овозможува поместување на вториот флексибилен влезен операнд и потоа извршување на некоја ALU операција, со употреба на само една инструкција. Покрај различностите во можностите за поместување, најзначајната разлика помеѓу овие два процесора е тоа што MIPS процесорот мора да го вчита IP заглавието од кешот во регистрите за општа намена пред да го обработува, додека пак MIMOPS процесорот може директно да работи со IP заглавието, сместено во неговата внатрешна податочна меморија.

MIPS	MIPS со парсер на IP заглавија	MIMOPS	MIMOPS со парсер на IP заглавија

mov r0, #0
mov r1, #0
lw r2, Header[r1]
	--Постави базен рег.
	la t0, Header
		--Постави базни регистри
		SetBRs Fields, Header, Header
			--Постави базен рег.
			SetBR1, Header

--Верзија and r3, r2, 15	*--Верзија* lw r3, h0	*--Верзија* and Fields[0], Header[0], 15, 110b	*--Верзија* //h0
--Долж. на заглавие srl r4, r2, 4 and r4, r4, 15	*--Долж. на заглавие* lw r4, h1	*--Долж. на заглавие* lw t1, 15, 000b and Fields[4], t1, Header[0] srl 4, 101b	*--Долж. на заглавие* //h1
--Тип на сервис srl r5, r2, 8 and r5, r5, 255	*--Тип на сервис* lw r5, h2	*--Тип на сервис* lw t2, 255, 000b and Fields[8], t2, Header[0] srl 8, 101b	*--Тип на сервис* //h2
--Прв збор прва поло. and r6, r2, 65535	*--Прв збор прва поло.* lw r6, h3	*--Прв збор прва поло.* and Fields[12], Header[0], 65535, 110b	*--Прв збор прва поло.* //h3
--Вкупна должина srl r7, r2, 16	*--Вкупна должина* lw r7, h4	*--Вкупна должина* srl Fields[16], Header[0], 16, 110b	*--Вкупна должина* //h4
mov r1, #4 lw r2, Header[r1]			
--Идентификатор and r8, r2, 65535	*--Идентификатор* lw r8, h5	*--Идентификатор* and Fields[20], Header[4], 65535, 110b	*--Идентификатор* //h5
--Знаменца srl r9, r2, 16 and r9, r9, 7	*--Знаменца* lw r9, h6	*--Знаменца* lw t3, 7, 000b and Fields[24], t3, Header[4] srl 16, 101b	*--Знаменца* //h6
--Отстап. на фрагмент srl r10, r2, 19	*--Отстап. на фрагмент* lw r10, h7	*--Отстап. на фрагмент* srl Fields[28], Header[4], 19,110b	*--Отстап. на фрагмент* //h7
--Втор збор втора поло. srl r11, r2, 16	*--Втор збор втора поло.* lw r11, h8	*--Втор збор втора поло.* srl Fields[32], Header[4], 16,110b	*--Втор збор втора поло.* //h8
mov r1, #8 lw r2, Header[r1]			
--Време на живот and r12, r2, 255	*--Време на живот* lw r12, h9	*--Време на живот* and Fields[36], Header[8], 255, 110b	*--Време на живот* //h9
--Протокол srl r13, r2, 8 and r13, r13, 255	*--Протокол* lw r13, h10	*--Протокол* and Fields[40], t2, Header[8] srl 8, 101b	*--Протокол* //h10
--Трет збор прва поло. and r14, r2, 65535	*--Трет збор прва поло.* lw r14, h11	*--Трет збор прва поло.* and Fields[44], Header[8], 65535, 110b	*--Трет збор прва поло.* //h11
--Сума за пров. на загл. srl r15, r2, 16	*--Сума за пров. на загл.* lw r15, h12	*--Сума за пров. на загл.* srl Fields[48], Header[8], 16, 110b	*--Сума за пров. на загл.* //h12

mov r1, #12
--изворна IP адреса
lw r16, Header[r1]
	--изворна IP адреса
	lw r16, h13
		--изворна IP адреса
		add Fields[52], Header[12], 0, 110b
			--изворна IP адреса
			//h13

--Четврт збор прва поло. and r17, r16, 65535	*--Четврт збор прва поло.* lw r17, h14	*--Четврт збор прва поло.* and Fields[56], Header[12], 65535, 110b	*--Четврт збор прва поло.* //h14
--Четврт збор втора поло. srl r18, r16, 16	*--Четврт збор втора поло.* lw r18, h15	*--Четврт збор втора поло.* srl Fields[60], Header[12], 16, 110b	*--Четврт збор втора поло.* //h15

mov r1, #16
--дест. IP адреса
lw r19, Header[r1]
	--дест. IP адреса
	lw r19, h16
		--дест. IP адреса
		add Fields[64], Header[16], 0, 110b
			--дест. IP адреса
			//h16

--Петти збор прва поло. and r20, r19, 65535	*--Петти збор прва поло.* lw r20, h17	*--Петти збор прва поло.* and Fields[68], Header[16], 65535, 110b	*--Петти збор прва поло.* //h17
--Петти збор втора поло. srl r21, r19, 16	*--Петти збор втора поло.* lw r21, h18	*--Петти збор втора поло.* srl Fields[72], Header[16], 16, 110b	*--Петти збор втора поло.* //h18

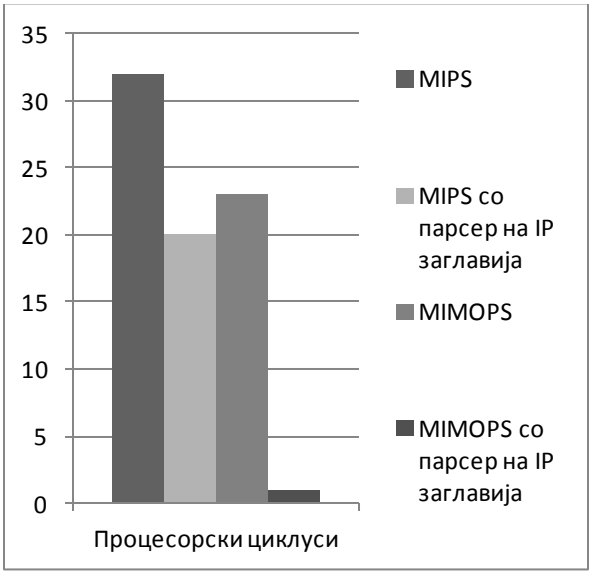
Слика 52 Асемблерски програми за парсирање на IPv4 заглавие во MIPS и MIMOPS процесори, без и со хардверска логика за парсирање на IP заглавија.

Првата програма која е дадена на сл. 52 имплементира пристап до сите полиња од IPv4 заглавие со MIPS процесор. Оваа програма го користи r0 регистерот како регистер со вредност 0, и r1 регистарот како покажувач кон зборовите од заглавието. Дополнително, регистарот r2 служи за зачувување на зборовите од заглавието кои се прочитани од меморија и кои подоцна се користат за екстрахирање на секое поле од IPv4 заглавието во регистрите r3-r21, последователно. За да се добие полето за верзија се врши логичка И операција со која сите битови во зборот се поставуваат на нула, освен последните 4, кои ја даваат вредноста на полето. Од друга страна, полето за должина на заглавие се екстрахира на тој начин што најпрво се врши поместување на зборот. Веднаш потоа се изведува логичка И операција со која сите битови во поместениот збор се поставуваат на нула, освен последните 4, кои ја даваат вредноста на полето. Сите останати полиња исто така се екстрахираат со употреба на операции за поместување и логички операции. Втората програма која е дадена на сл. 52 претставува еквивалент на претходната, со таа разлика што се однесува на MIPS процесор, кој вклучува посебен хардвер за парсирање на IP заглавија. Оваа програма овозможува директно адресирање на полињата, со употреба на мнемоници кои започнуваат со буквата h а продолжуваат со бројот на полето, според редоследот кој е определен со описот на заглавието. Тоа значи дека за да се прочита некое поле и смести во регистер потребна е само една инструкција.

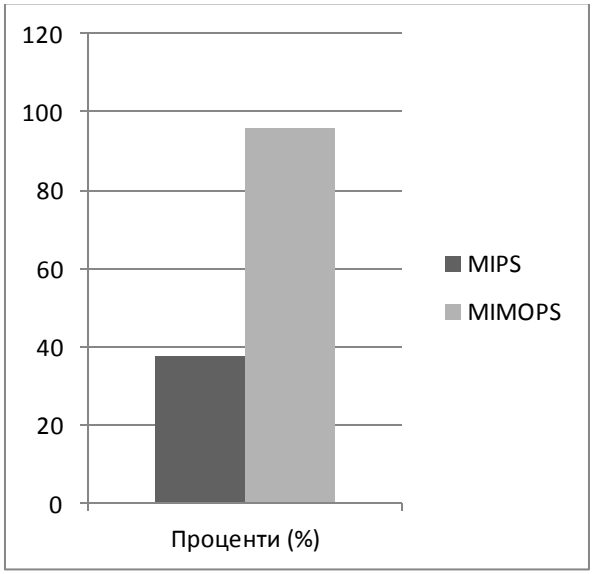
Третата и четвртата програма кои се дадени на сл. 52 имплементираат пристап до сите полиња од IPv4 заглавие со MIMOPS процесор, и MIMOPS процесор кој вклучува хардверска логика за парсирање на IP заглавија, соодветно. Оттука може да се каже дека третата програма има многу сличности со првата, додека пак четвртата програма наликува на втората. И покрај тоа што третата програма (која е наменета за MIMOPS процесор) работи директно со зборовите од IP заглавието, сместени во внатрешната меморија на чипот, сепак неопходно е оваа програма да ги екстрахира полињата кои не се со должина на збор. Инструкциите кои вршат екстрахирање на полињата можат да адресираат најмногу три операнди, при што со 3-битна непосредна вредност се определува кој операнд користи базно адресирање. Екстрахираните полиња се сместуваат на последователни мемориски локации во низата Fields, после што процесорот

може директно да пристапува до нив и да ги обработува со ALU единицата. Од друга страна, четвртата програма (која е наменета за MIMOPS процесор кој вклучува логика за парсирање на IP заглавија) треба само да го постави базниот регистар да покажува кон почетната адреса на IP заглавието, и на тој начин да овозможи директен пристап до полињата од IP заглавието (специфицирани со мнемоници, како во втората програма). Според тоа, една инструкција која ја намалува вредноста на полето за време на живот (h9) би можела да се специфицира како *SUB h9, h9, 1*, овозможувајќи и на ALU единицата веднаш да го процесира екстрахираното поле за време на живот. Повеќе детали околу целосната обработка на IP заглавијата со модифицираното MIMOPS процесорско јадро за мрежна примена, се дадени во глава 7.

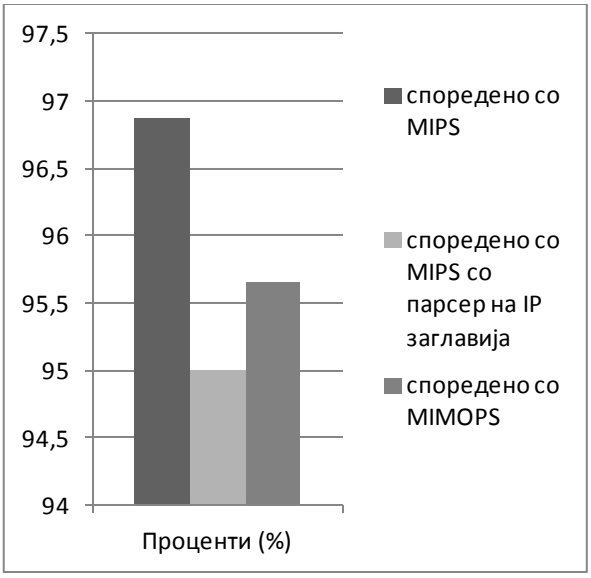
Резултатите од компаративната анализа се прикажани на сл. 53, каде на сл. 53 а) е претставено времето на извршување на програмата за парсирање на IPv4 заглавие, додека пак на сл. 53 б) е илустрирано подобрувањето кое се постигнува со употреба на предложената логика за парсирање на IP заглавија, во MIPS и MIMOPS процесорот, соодветно. Според овие резултати, може да се забележи дека MIPS процесорот кој користи парсер за заглавија на IP пакети постигнува подобрување од 37,5% во споредба со основен MIPS процесор, додека пак MIMOPS процесорот кој користи парсер за заглавија на IP пакети постигнува подобрување од 95,6% во споредба со основен MIMOPS процесор.



а) Време на извршување.



б) Подобрување со примена на логика за парсирање на IP заглавија.



в) Вкупно подобрување на MIMOPS процесор кој имплементира парсер на IP заглавија.

Слика 53 Анализа на парсирање на IPv4 заглавие во MIPS и MIMOPS процесори, без и со логика за парсирање на IP заглавија.

Вкупното подобрување на MIMOPS процесорот кој користи парсер за заглавија на IP пакети е дадено на сл. 53 в), каде се забележува дека овој процесор постигнува 96.8%, 95%, 95.6% подобри резултати при парсирање на IPv4 заглавие, во споредба со процесорите: MIPS, MIPS со парсер на IP заглавие и MIMOPS, соодветно. Оваа анализа потврдува дека предложената логика за парсирање на IP заглавија може да ја подобри брзината на парсирање на IPv4 заглавија, на различни процесорски архитектури.

На сл. 54 се прикажани асемблерски програми за обработка на IPv6 заглавие со MIPS и MIMOPS процесори, без и со логика за парсирање на IP заглавија. Дадените програми вршат парсирање на IPv6 заглавие, со екстрахирање на неговите полиња: верзија, класа на сообраќај, ознака за тек, должина на товар, следно заглавие, максимален број на хопови, изворна IP адреса и дестинациска IP адреса. Овие програми се многу слични со оние кои се однесуваат на парсирање на IPv4 заглавие, и исто така вклучуваат мноштво логички битови операции и операции за поместување.

MIPS	MIPS со парсер на IP заглавија	MIMOPS	MIMOPS со парсер на IP заглавија

mov r0, #0
mov r1, #0
lw r2, Header[r1]
	--Постави базен рег.
	la t0, Header
		--Постави базни регистри
		SetBRs Fields, Header, Header
			--Постави базен рег.
			SetBR1, Header

--Верзија and r3, r2, 15	*--Верзија* lw r3, h0	*--Верзија* and Fields[0], Header[0], 15, 110b	*--Верзија* //h0
--Класа на сообраќај srl r4, r2, 4 and r4, r4, 255	*--Класа на сообраќај* lw r4, h1	*--Класа на сообраќај* lw t1, 255, 000b and Fields[4], t1, Header[0] srl 4, 101b	*--Класа на сообраќај* //h1
--Ознака за тек srl r5, r2, 12	*--Ознака за тек* lw r5, h2	*--Ознака за тек* srl Fields[8], Header[0], 12, 110b	*--Ознака за тек* //h2
mov r1, #4 lw r2, Header[r1]			
--Долж. на товар and r6, r2, 65535	*--Долж. на товар* lw r6, h3	*--Долж. на товар* and Fields[12], Header[4], 65535, 110b	*--Долж. на товар* //h3
--Следно заглавие srl r7, r2, 16 and r7, r7, 255	*--Следно заглавие* lw r7, h4	*--Следно заглавие* and Fields[16], t1, Header[4] srl 16, 101b	*--Следно заглавие* //h4
--Макс. бр. на хопа srl r8, r2, 24	*--Макс. бр. на хопа* lw r8, h5	*--Макс. бр. на хопа* srl Fields[20], Header[4], 24, 110b	*--Макс. бр. на хопа* //h5

--изворна IP адреса
mov r1, #8
lw r9, Header[r1]
	--изворна IP адреса
	lw r9, h6
		--изворна IP адреса
		add Fields[24], Header[8], 0, 110b
			--изворна IP адреса
			//h6

mov r1, #12 lw r10, Header[r1]	lw r10, h7	add Fields[28], Header[12], 0, 110b	//h7
mov r1, #16 lw r11, Header[r1]	lw r11, h8	add Fields[32], Header[16], 0, 110b	//h8
mov r1, #20 lw r12, Header[r1]	lw r12, h9	add Fields[36], Header[20], 0, 110b	//h9

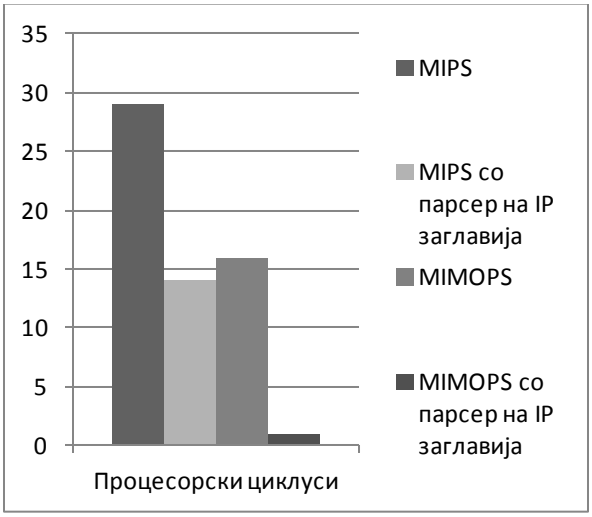
--дест. IP адреса
mov r1, #24
lw r13, Header[r1]
	--дест. IP адреса
	lw r13, h10
		--дест. IP адреса
		add Fields[40], Header[24], 0, 110b
			--дест. IP адреса
			//h10

mov r1, #28 lw r14, Header[r1]	lw r14, h11	add Fields[44], Header[28], 0, 110b	//h11
mov r1, #32 lw r15, Header[r1]	lw r15, h12	add Fields[48], Header[32], 0, 110b	//h12
mov r1, #36 lw r16, Header[r1]	lw r16, h13	add Fields[52], Header[36], 0, 110b	//h13

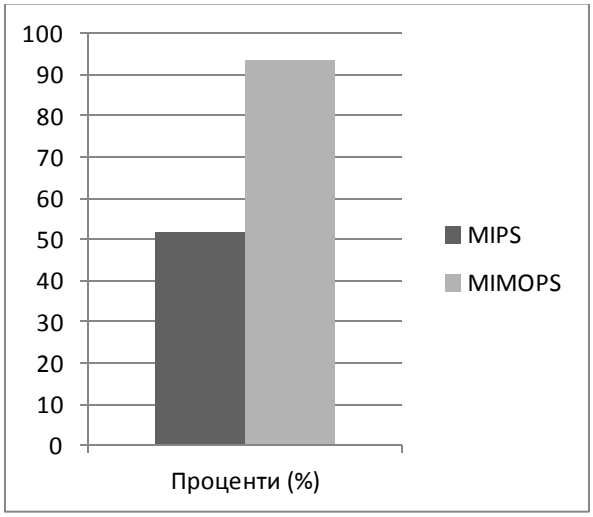
Слика 54 Асемблерски програми за парсирање на IPv6 заглавие во MIPS и MIMOPS процесори, без и со хардверска логика за парсирање на IP заглавија.

Резултатите од компаративната анализа се прикажани на сл. 55, каде на сл. 55 а) е претставено времето на извршување на програмата за парсирање на IPv6 заглавие, додека пак на сл. 55 б) е илустрирано подобрувањето кое се постигнува со употреба на предложената логика за парсирање на IP заглавија, во MIPS и MIMOPS процесорот, соодветно. Според овие резултати, може да се забележи дека MIPS процесорот кој користи парсер за заглавија на IP пакети постигнува подобрување од 51,7% во споредба со основен MIPS процесор, додека пак

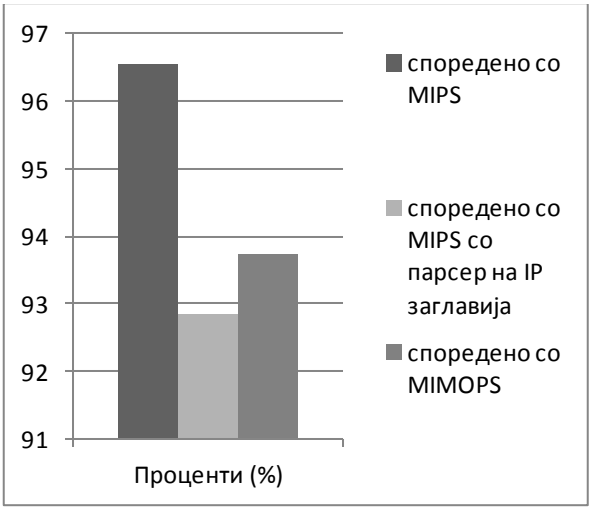
MIMOPS процесорот кој користи парсер за заглавија на IP пакети постигнува подобрување од 93,7% во споредба со основен MIMOPS процесор. Вкупното подобрување на MIMOPS процесорот кој користи парсер за заглавија на IP пакети е дадено на сл. 55 в), каде се забележува дека овој процесор постигнува 96,5%, 92,8%, 93,7% подобри резултати при парсирање на IPv6 заглавие, во споредба со процесорите: MIPS, MIPS со парсер на IP заглавие и MIMOPS, соодветно. Оваа анализа потврдува дека предложената логика за парсирање на IP заглавија може да ја подобри брзината на парсирање на IPv6 заглавија, на различни процесорски архитектури.



а) Време на извршување.



б) Подобрување со примена на логика за парсирање на IP заглавија.



в) Вкупно подобрување на MIMOPS процесор кој имплементира парсер на IP заглавија.

Слика 55 Анализа на парсирање на IPv6 заглавие во MIPS и MIMOPS процесори, без и со логика за парсирање на IP заглавија.

6. ПРАКТИЧНА РЕАЛИЗАЦИЈА НА ПРЕДЛОЖЕНИОТ RISC-БАЗИРАН МЕМОРИСКО-ЦЕНТРИЧЕН ПРОЦЕСОР (MIMOPS) ВО FPGA

Популарноста во употребата на јазиците за опис на хардвер (hardware description languages - HDL) започнува во средните - 1990ти, кога на пазарот се појавуваат комерцијални алатки за синтеза на дигитални кола. Постојат два популарни јазика за опис на хардвер кои се во широка употреба во денешно време, а тоа се: VHDL и Verilog, [68]. VHDL е кратенка од VHSIC HDL, каде VHSIC се однесува на Very High Speed Integrated Circuit (интегрирано коло со многу голема брзина). VHDL јазикот е развиен во средните-1980ти, и подоцна е стандардизиран од IEEE во 1987 год. (VHDL-87), па надграден во 1993 год. (VHDL-93). Од друга страна, Verilog е првично воведен во 1984, па потоа во 1988 е превземен како комерцијален јазик за опис на хардвер од компаниите Synopsys и Cadence Design Systems. Двата HDL јазика (Verilog и VHDL) се базираат на слични принципи, но имаат различна синтакса. Во овој докторски труд се користи VHDL јазик за развој на HDL модел на предложениот MIMOPS процесор.

VHDL јазикот во многу аспекти наликува на стандардните програмски компјутерски јазици, како што е C++. На пример, истиот се карактеризира со изрази за доделување на променливи, условни искази, јамки и функции. На сличен начин како што кај компјутерските програмски јазици се користи компајлер за да се овозможи преведување на изворниот код (опишан во високо ниво) во машински код, така и кај VHDL се употребува алатка за синтеза за да се овозможи преведување на изворниот код во опис на конкретно хардверско коло кое го имплементира тој код. Овој опис, кој се нарекува мрежна листа овозможува автоматски да се генерира конкретното физичко дигитално коло кое го реализира дадениот изворен код. Покрај тоа, постои можност да се направат и прецизни функционални и временски симулации на кодот со цел да се потврди исправноста на колото. Дадените алатки кои се потребни за симулирање, синетизирање и имплементирање на VHDL код во конкретен хардвер, вообичаено се организирани во CAD (Computer Aided Design) софтверско множество на алатки. Во

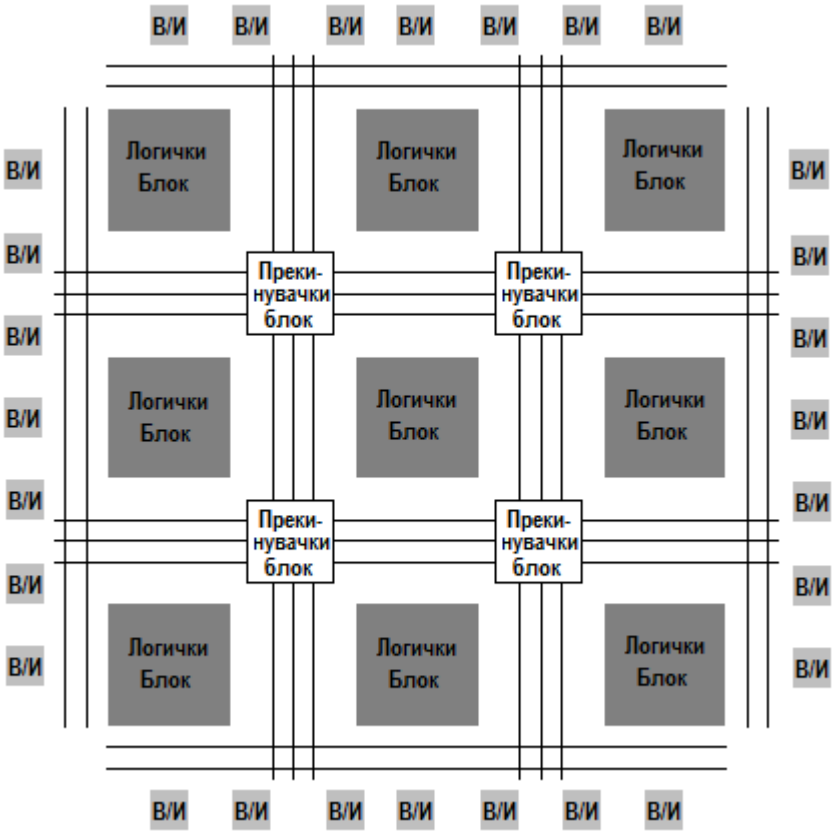
овој докторски труд се користи Xilinx Vivado Design Suite 2016.4 софтверската околина при имплементирање на VHDL моделот на MIMOPS процесорот во Xilinx Virtex7 VC709 FPGA развојна плочка, [69].

Vivado Design Suite претставува софтверско множество на алатки произведено од Xilinx кое е наменето за синтеза и анализа на HDL дизајни, [71]. Оваа целосно нова софтверска околина ја заменува Xilinx ISE Design Suite софтверската околината и ја надградува со дополнителни можности за развој на системи во чип и синтеза на високо ниво. Генерално, Vivado Design Suite им овозможува на хардверските архитекти да ги синтетизираат (компајлираат) своите HDL кодови, да извршуваат временски анализи, да испитуваат RTL дијаграми, да го симулираат однесувањето на дизајнот за различни влезови и да конфигурираат односно програмираат крајни FPGA уреди. Овие функционалности се постигнуваат со неколку алатки кои се дел од Vivado софтверската околина, вклучувајќи: Vivado компајлер за обична синтеза и синтеза на високо ниво (XST- High-Level Synthesis), Vivado алатка за имплементирање т.е. преведување, мапирање, сместување и рутирање (translate, map, place и route), Vivado интегратор на Intellectual Property јадра, Vivado дебагер за сериски В/И и анализатор на логика, XDC (Xilinx Design Constraints) алатка за ограничувања на временски параметри и влез, Vivado алатка за програмирање (Xilinx iImpact) итн. Во главно, Vivado е софтверската околина, која е наменета за FPGA плочи од Xilinx, и е тесно врзана со архитектурата на овие чипови, и при тоа не може да се употреби за FPGA плочи од други производители. Всушност, оваа софтверска околина е наменета да ја подобри продуктивноста при дизајнирање, интегрирање и имплементирање на хардвер во Xilinx серија7 (Virtex-7, Kintex-7, Artix-7), Zynq-7000 и UltraScale уреди.

FPGA претставува програмабилен логички уред (PLD) кој може да се репрограмира произволен број на пати, откако е произведен, [68]. За разлика од обичните интегрирани кола (IC) кои се скапи и за кои е потребно долго време да се дизајнираат, FPGA уредите можат лесно и брзо да се програмираат со употреба на CAD алатка, бидејќи тие не побаруваат правење на физички layout и маски, ниту пак производство на проектираниот чип. Постојат повеќе различни FPGA архитектури, кои се понудени од различни производители, вклучувајќи

Altera, Xilinx, Actel, Lucent, QuickLogic и Cypress. И покрај тоа што конкретната структура на овие FPGA уреди варира од производител до производител, сепак сите FPGA уреди содржат поле од претходно произведени програмабилни логички блокови, кои внатрешно се организирани во матрична конфигурација. Овие логички блокови автоматски се поврзуваат едни со други, со помош на програмабилна поврзувачка мрежа. На сл. 56 е прикажана внатрешната структура на еден FPGA уред.

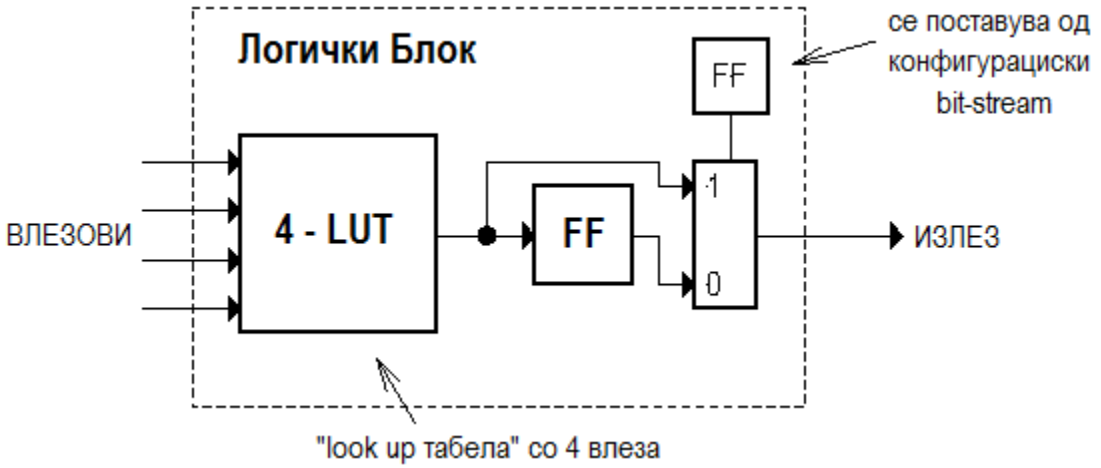
Кога еден FPGA уред се програмира, тогаш тој ги користи своите логички блокови за да реализира мали делови од колото кое треба да се имплементира, додека пак програмабилната поврзувачка мрежа ги конфигурира сите потребни врски помеѓу различните логички блокови и кон В/И (влезно/излезните) блокови. Тоа значи дека при имплементација на некој покомплексен дизајн, како што е процесор, истиот треба да се подели на повеќе функционални делови и секој од нив да се реализира како еден логички блок. Конечниот резултат (CPU во FPGA) се постигнува кога сите функционални делови кои се имплементирани во логичките блокови ќе се поврзат, со програмирање на FPGA меѓу-поврзувањата.



Слика 56 Архитектура на FPGA чип, [68].

Според сл. 56, еден FPGA уред вообичаено вклучува: логички блокови, меѓу-поврзувања и прекинувачки блокови (влезно/излезни блокови), [68]. Прекинувачките блокови се сместени периферно од логичките блокови и меѓу-поврзувањата. Жичаните сегменти (две крајни точки од меѓу-поврзување без програмабилен прекинувач помеѓу нив) се поврзуваат со логичките блокови преку прекинувачките блокови. Во зависност од хардверскиот дизајн кој се реализира, еден логички блок се поврзува со друг итн.

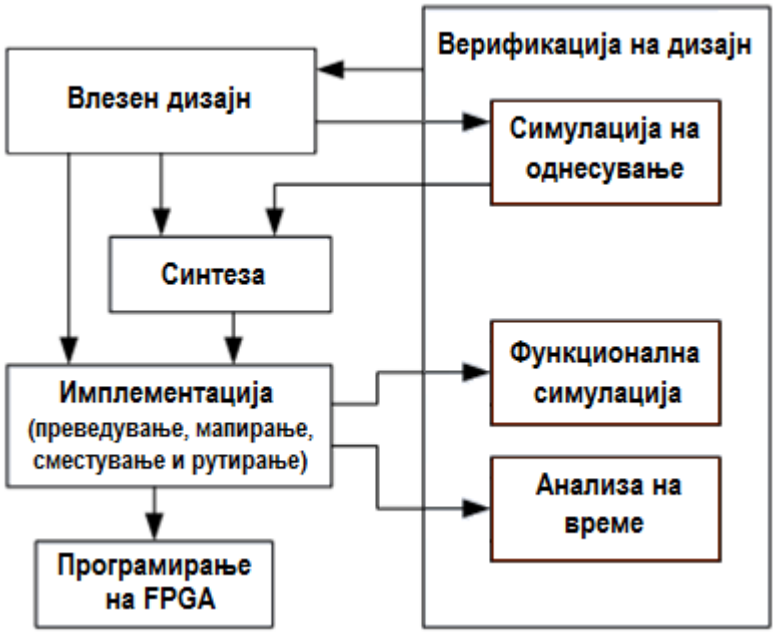
Еден логички блок во Xilinx FPGA може да биде составен од една look up табела (LUT) и еден флип флоп. LUT табелата се користи за имплементирање на различни функционалности. Влезните линии од логичкиот блок влегуваат во LUT табелата и ја овозможуваат. Излезот од LUT табелата го дава резултатот од логичката функција која е имплементирана во оваа табела, и потоа кон излез се пренасочува излезот од LUT табелата кој е прочитан или не е прочитан од регистер. Една логичка функција со k влезови може да се имплементира со LUT табела која користи SRAM со капацитет од 2^k*1b, додека пак бројот на различни можни функции кои може да се имплементираат со LUT табела со k влезови изнесува 2^{2^k}. Предноста на оваа архитектура е тоа што поддржува имплементирање на многу логички функции, но недостаток е употребата на голем број на мемориски клетки во случај кога треба да се имплементира логички блок со голем број на влезови. Во продолжение, на сл. 57 е прикажана структурата на логички блок во Xilinx FPGA, кој употребува LUT табела со 4 влеза.



Слика 57 Структура на логички блок со LUT табела со 4 влеза во Xilinx FPGA чип.

6.1. Тек на дизајнирање на хардвер во FPGA со употреба на Xilinx Vivado софтверска околина

Проектирањето на хардвер во FPGA е слично како и за било која друга технологија. Производителот Xilinx го употребува Vivado design suite множеството на алатки за таа цел, [104]. Генерално, овој процес вклучува конвертирање на дадено дигитално коло (претставено со шема или HDL опис со VHDL или Verilog) во низа од битови (bitstream), која потоа се користи за програмирање на соодветен Xilinx FPGA уред. На сл. 58 е даден еден едноставен дијаграм со кој е опишан текот на дизајнирање на хардвер во FPGA со Xilinx Vivado софтверската околина.



Слика 58 Тек на дизајнирање на хардвер во FPGA со Xilinx Vivado софтверска околина, [104].

Според сл. 58 веднаш може да се забележи дека во првата фаза се врши опишување на дизајнот со шема, HDL код, или комбинација од двете. Изборот на начинот на дизајнирање на колото зависи од самиот дизајн и од дизајнерот. Според тоа, доколку дизајнерот сака повеќе да се занимава со хардвер тогаш подобро е да избере шематски приказ на дизајнот, но доколку дизајнот е многу комплексен тогаш HDL јазикот е подобар избор. Генерално, побрзо и поедноставно е да се опише некој дизајн во HDL, но од друга страна ова воведува недостатоци во поглед на перформанси и густина на дизајнот. Со оглед на комплексноста на предложениот MIMOPS процесор, тој е дизајниран во VHDL

Веднаш откако ќе се опише дизајнот, следен чекор пред синтезата е да се направи негова верификација со симулација на однесување (RTL симулација). Ова вклучува: анализа на RTL кодот и потврдување на функционалноста на дизајнот со набљудување на неговите сигнали и променливи, следење на извршувањето на процедурите и функциите, и по потреба поставување на точки на прекини. Овој тип на симулација се извршува брзо и му овозможува на дизајнерот за кратко време да го промени HDL кодот доколку потребната функционалност не е постигната. Со оглед на тоа дека дизајнот не е синтетизиран на ниво на порти, параметрите за времето и искористеност на ресурси се уште не се познати и не можат да се анализираат. Во оваа фаза се вршат неколку симулации на MIMOPS процесорот и неговите основни компоненти (пр. контролна единица, ALU итн.).

Во фазата на синтеза се прави анализа на HDL дизајнот (проверка на синтакса на кодот и анализа на хиерархија на дизајнот), после што тој се конвертира во "мрежна листа" (netlist), составена од генерички логички компоненти (порти, флип флопови, итн.), кои се поврзани меѓу себе со врски наречени "мрежи" (nets). Доколку дизајнот вклучува повеќе под-дизајни тогаш при синтеза се генерира мрежна листа за секој посебен елемент. Резултатната мрежна листа (листи) се зачувува во NGC (Native Generic Circuit) датотека (за Xilinx технологија за синтеза - XST). Ваквиот дизајн не е наменет за некој конкретен FPGA уред, па истата мрежна листа(листи) може да се употреби за да се проектира интегрирано ASIC коло за специфична намена или обичен чип. Синтезата на MIMOPS процесорот овозможува да се анализира неговата шема и да се добијат првични резултати за просторните ресурси кои се потребни за негова имплементација.

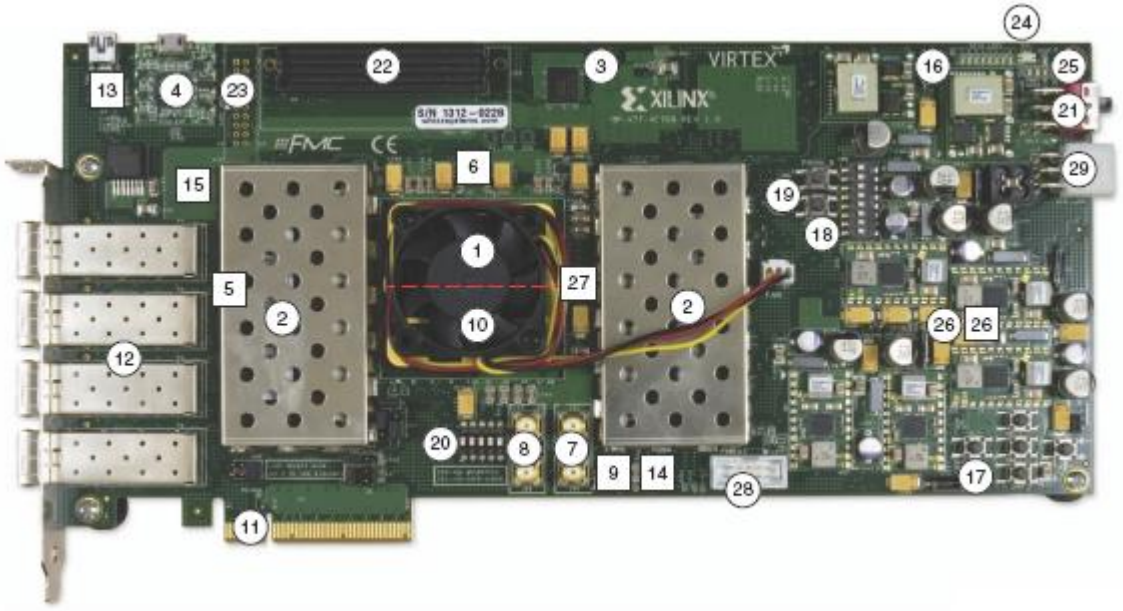
Фазата на имплементација е составена од четири под-фази: преведување, мапирање, сместување и рутирање. Во фазата на преведување се врши конвертирање на општата мрежна листа на дизајнот во специфична мрежна листа за Xilinx, која се зачувува како NGD (Native Generic Database) датотека. Во генерираната датотека, сите компоненти на дизајнот се специфични за Xilinx, како на пример: флип флопови, look-up табели, мултиплексери итн. Во оваа фаза исто така се дефинира и UCF (User Constraints File) датотека на ограничувања, во која се наведува поврзувањето на В/И порти на дизајнот со физичките елементи (пр. пинови, прекинувачи, копчиња) на крајниот уред и воедно се специфицираат

временските ограничувања за дизајнот. Како и да е, во оваа фаза дизајнот е се уште општо дефиниран за Xilinx уреди, а не за конкретната FPGA компонента која е избрана да се употреби. Дополнително, во оваа фаза може да се изврши и функционална симулација (Post Translate Simulation) за да се анализира логичкото функционирање на дизајнот. Следната фаза на мапирање ја конвертира специфичната мрежна листа за Xilinx во прилагодена мрежна листа со конкретни компоненти од FPGA уредот кој е селектиран за имплементација. На пример, предложениот MIMOPS процесор се мапира во компонентите од Xilinx Virtex7 XC7VX690T FPGA плоча. Всушност, при мапирање се врши прилагодување на логиката која е дефинирана со NGD датотеката кон елементите на крајниот FPGA уред (логички блокови и влезно/излезни блокови), при што се генерира NCD (Native Circuit Description) датотека која физички го претставува дизајнот мапиран во компонентите на селектираниот FPGA уред. Следната фаза на сместување ја зема мрежната листа која е прилагодена за селектираниот уред и специфицира точно каде во дадениот уред треба да биде сместена секоја од компонентите. Откако компонентите ќе се сместат на уредот потребно е да се направи заедничко поврзување (рутирање) помеѓу нив. FPGA уредите располагаат со голема количина на рутирачки ресурси на чипот (локални жици, глобални жици, прекинувачки блокови итн). Сите компоненти треба точно да се поврзат меѓу себе, но идеално би било да се овозможи нивно поврзување со најкратко доцнење за да се постигне поголема фреквенција на дизајнот. Всушност, за таа цел може да се направи анализа на времето после фазите на мапирање, или сместување и рутирање (PAR), за да се набљудува доцнењето на сигналите во дизајнот. Како и да е, после имплементацијата се генерираат подетални извештаи за карактеристиките на проектираниот хардвер (т.е MIMOPS процесорот).

На крај дизајнот е подготвен да се употреби за програмирање на уредот со помош на Xilinx iImpact алатката. Во тој процес, генерираната NCD датотека се проследува кон BITGEN програма која генерира низа од битови (.BIT датотека) со која се програмира крајниот FPGA уред (поставување на равенства во LUT табели, конфигурирање на рути и прекинувачи итн.). Програмирањето најчесто се врши преку JTAG кабел. Креираниот хардверски прототип на MIMOPS процесорот овозможува да се симулира и анализира неговата работа во реален хардвер.

6.2. Опис на Virtex7 VC709 FPGA развојна плоча

Со оглед на тоа дека Xilinx VC709 развојната плочка се употребува за имплементирање на MIMOPS процесорот во Virtex7 XC7VX690T FPGA, овде се претставени повеќе детели за неа. Генерално, оваа развојна плочка претставува хардверска средина за развивање и еволуирање на дизајни кои се однесуваат на Virtex7 XC7VX690T-2FFG1761C FPGA, [69]. Оваа плочка нуди карактеристики кои се заеднички за повеќе вградливи процесорски системи, како што се: DDR3 мемориски модули, 8-lane PCI Express интерфејс, В/И за општа намена, како и UART интерфејс. Други карактеристики може да се додадат со користење на картички прикачени на VITA-57 FPGA конекторот (FMC) кој се наоѓа на плочката. На сл. 59 е претставена структурата на оваа развојна плоча, при што со бројчиња се означени сите нејзини составни компоненти, кои потоа се опишани во табела 11.



Слика 59 Xilinx Virtex7 VC709 развојна плочка, [69].

Табела 11 Опис на составни компоненти на VC709 плочка, [69].

Број	Референтна ознака	Опис на компонента
1	U1	Virtex7 FPGAXC7VX690T-2FFG1761C со вентилатор за ладење
2	J1, J3	Две DDR3 SODIMM мемории (4 GB секоја)
3	U3	BPI паралелна NOR flash меморија (1 Gb)
4	U26	USB JTAG интерфејс (микро-B USB конектор)
5	U51	Системски такт, 200 MHz, LVDS (задна страна на плочката)
6	U34	I^2C програмабилен кориснички такт LVDS, 156.250 MHz предефинирана фреквенција (задна страна на плочката)

Број	Референтна ознака	Опис на компонента
7	J31, J32	Кориснички SMA такт
8	J25, J26	GTH трансивер SMA референтен такт
9	U24	Такт за намалување на неправилности (задна страна на плочката)
10	U1	GTH трансивер Quad 111–Quad 119
11	P1	PCI Express конектор
12	P2-5	Конектори за 4 X SFP/SFP+ модули (I²C 0x50)
13	U44, J17	USB-до-UART мост со мини-B USB конектор
14	U52	I^2C прекинувач за магистрала (I²C 0x74) (задна страна на плочката)
15	U14	I^2C прекинувач за магистрала (I²C 0x75) (задна страна на плочката)
16	DS2–DS9	Кориснички зелени LED диоди
17	SW3–SW7	Кориснички копчиња, активни на високо
18	SW2	Кориснички DIP прекинувач, активен на високо
19	SW8, SW9	CPU RESET, FPGA PROG копчиња
20	SW11	Конфигурациски режим/горна линеарна флеш адреса за DIP прекинувач
21	SW12	Прекинувач за вклучување/исклучување
22	J35	FMC HPC конектор
23	J19	Xilinx XADC блок
24	DS1	INIT LED диода со две бои, црвена/зелена
25	DS10, DS14, DS16–DS18	LED диоди за напојување ON и напојување GOOD
26	Различно	Систем за управување со напојувањето (предна и задна страна на плочката)
27	U13	Мемориски такт, 233.33 MHz, LVDS (задна страна на плочката)
28	J35	2 x 5 обвиткан PMBus конектор
29	J18	2 x 3 конектор за 12V влезно напојување

Во табела 12 се прикажани расположливите ресурси на комбинациони логички блокови во дадената Virtex7 7VX690T FPGA, [105]. Секој slice ресурс од оваа FPGA плоча содржи четири look-up табели и осум флип флопови. Овие slice ресурси се поделени во логички (SLICEL) и мемориски (SLICEM) slice-ови, при што само SLICEM ресурсите може да ги употребат своите look-up табели за имплементирање на дистрибуиран RAM или регистри за поместување.

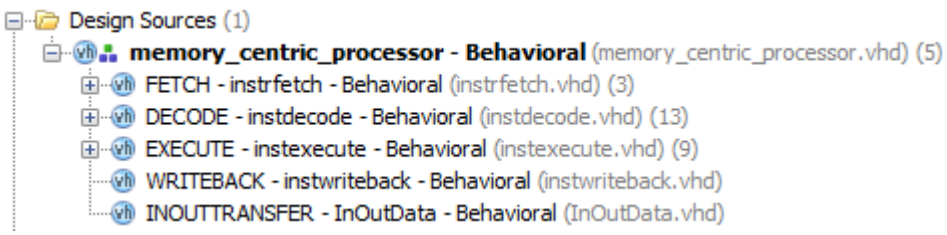
Табела 12 Расположливи логички блокови во 7VX690T Virtex7 FPGA, [105].

Уред	Slices	SLICEL	SLICEM	LUT со 6 влеза	Дистрибуиран RAM (Kb)	Регистри за поместување (Kb)	Флип флопови
7VX690T	108.300	64.750	43.550	433.200	10.888	5.444	866.400

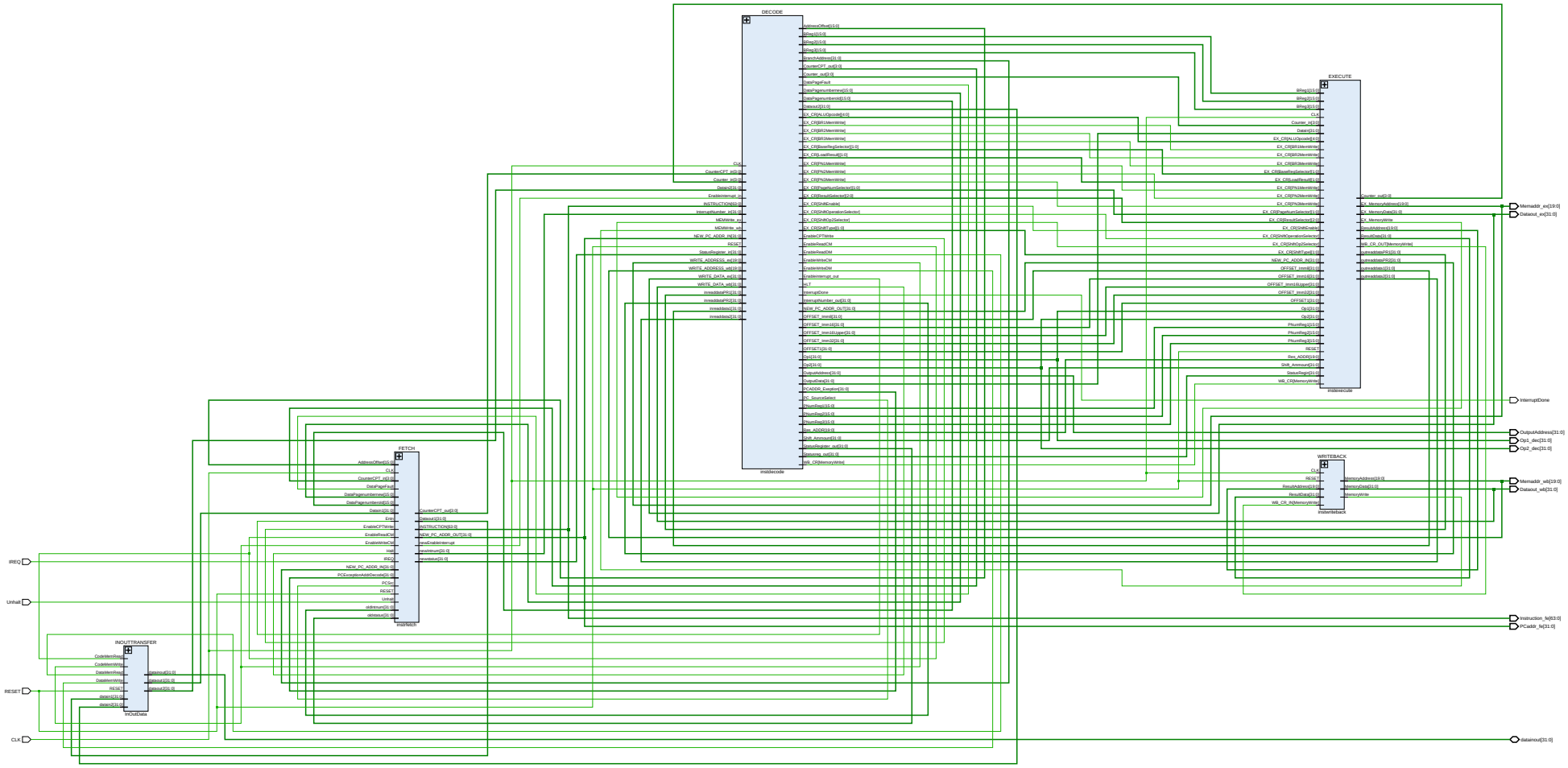
6.3. Проектирање на предложениот MIMOPS процесор во VHDL и негово симулирање со Xilinx Vivado софтверска околина

Иницијалниот чекот во FPGA имплементацијата на MIMOPS процесорот е да се развие VHDL модел на предложениот процесор во Vivado Design Suite софтверската околина, според описот кој е даден во главите 4 и 5. Генерално, имплементацијата на MIMOPS процесорот содржи два основни вида на логички елементи: комбинациони и секвенцијални. Комбинационите елементи се оние кои извршуваат операции каде излезите зависат само од тековните влезови. Такви елементи во имплементацијата на MIMOPS процесорот се: аритметичко-логичка единица, собирачи, проширувачи, поместувачи, генератори на мемориски адреси итн. Од друга страна, секвенцијалните елементи се оние кои зачувуваат некоја состојба. Секој секвенцијален елемент содржи најмалку два влеза и еден излез, при што двата неопходни влеза се: податокот кој треба да се запише во колото и такт сигналот, додека пак излезот го дава податокот кој бил запишан во претходниот такт циклус. Вакви секвенцијални елементи во имплементацијата на MIMOPS процесорот се: програмски бројач, проточни регистри, инструкциска меморија, и податочна меморија итн.

Проточноста (анг. pipelining) претставува стандардна карактеристика во RISC-базираните процесори, која се употребува за да ги подобри неговите вкупни перформанси. Оваа техника му овозможува на процесорот истовремено да извршува различни фази од инструкции, и на тој начин да обезбеди извршување на повеќе инструкции за пократок временски период. На пример во VHDL моделот на MIMOPS процесорот, проточната податочна патека е поделена во четири модули (земање, декодирање, извршување и запишување резултат), каде секој модул мора да го почека претходниот пред да започне со извршувањето. На тој начин се овозможува завршување на инструкциите за еден такт циклус. На сл. 60 е прикажана организацијата на VHDL моделот на MIMOPS процесорот.



Слика 60 Хиерархиска организација на модули во VHDL модел на MIMOPS процесор.



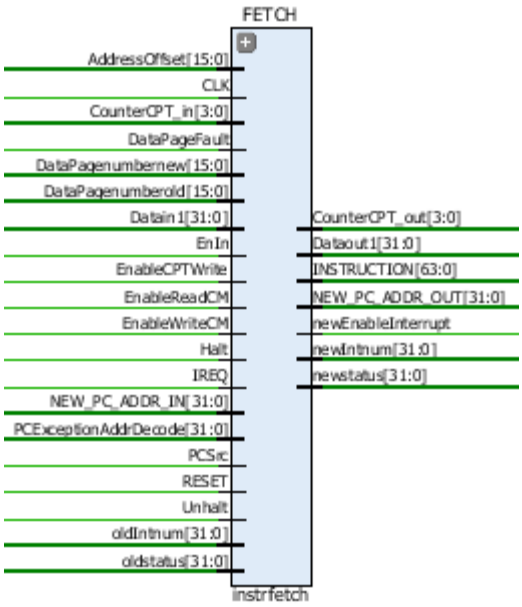
Слика 61 Блок дијаграм на VHDL модел на MIMOPS процесор.

На сл. 60 се забележува дека VHDL моделот на MIMOPS процесорот покрај модулите за проточна работа, вклучува и уште еден модул кој овозможува комуникација со В/И уреди. Истото е прикажано и на сл. 61, каде е претставен блок дијаграм на VHDL моделот на MIMOPS процесорот. Согледувајќи ја комплексноста на шемата која е дадена на сл. 61, во продолжение е дадена анализа и симулација на поединечните VHDL модули од MIMOPS процесорот, и на некои од нивните најзначајни компоненти. Тоа значи дека за секој од дадените модули е генериран блок дијаграм, со помош на Vivado design suite софтверската околина. Дополнително, употребена е алатката за симулирање во Vivado, со која се верификува работата на овие VHDL модули со посебни тест-програми кои се напишани за таа намена. Конечно, на крај е направена симулација на целиот MIMOPS процесор, со што е потврдена неговата севкупна функционалност.

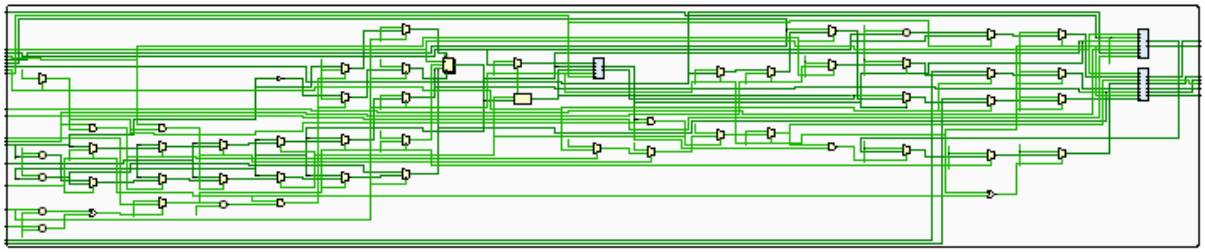
Модулот за земање на инструкција (fetch) е наменет за да прочита инструкција од меморијата во чипот и да ја генерира следната вредност на програмскиот бројач која ќе биде употребена за земање на инструкција во следниот такт циклус. Во согласност со претходниот опис и објаснување на принципот на работа на единицата за земање на инструкција, даден во поглавје 4.3., напишан е VHDL модел за дадената единица. VHDL моделот на единицата за земање инструкција ги имплементира поголемиот дел од функционалностите внатре во описот на архитектурата на ентитетот на единицата за земање, и при тоа вклучува три поединечни компоненти за: инструкциска меморија, табела на страници за инструкциска меморија и IF/ID проточен регистер. На сл. 62 се дадени повеќе детали за проектираниот VHDL модел на единицата за земање на инструкција во Vivado design suite софтверската околина.

```
process(CLK, RESET, PCSrc, Halt_in, NEW_PC_ADDR_IN, PC_ADDR_AUX2)
	begin
		if(RESET = '1') then
			PC_ADDR_AUX1 <= (others => '0');
			--PC = 0, кога reset е поставен
		elsif (PCSrc= '1' and Halt_in=zero) then  --гранење
			PC_ADDR_AUX1 <= NEW_PC_ADDR_IN;
			--PC се поставува на адреса на гранење
		elsif (rising_edge(CLK) and Halt_in=zero) then
			PC_ADDR_AUX1 <= PC_ADDR_AUX2;
			--PC се поставува да покажува на следна инструкција(PC+8)
		end if;
			--ако Halt_in е нула, тогаш PC не се менува
	end process;
```

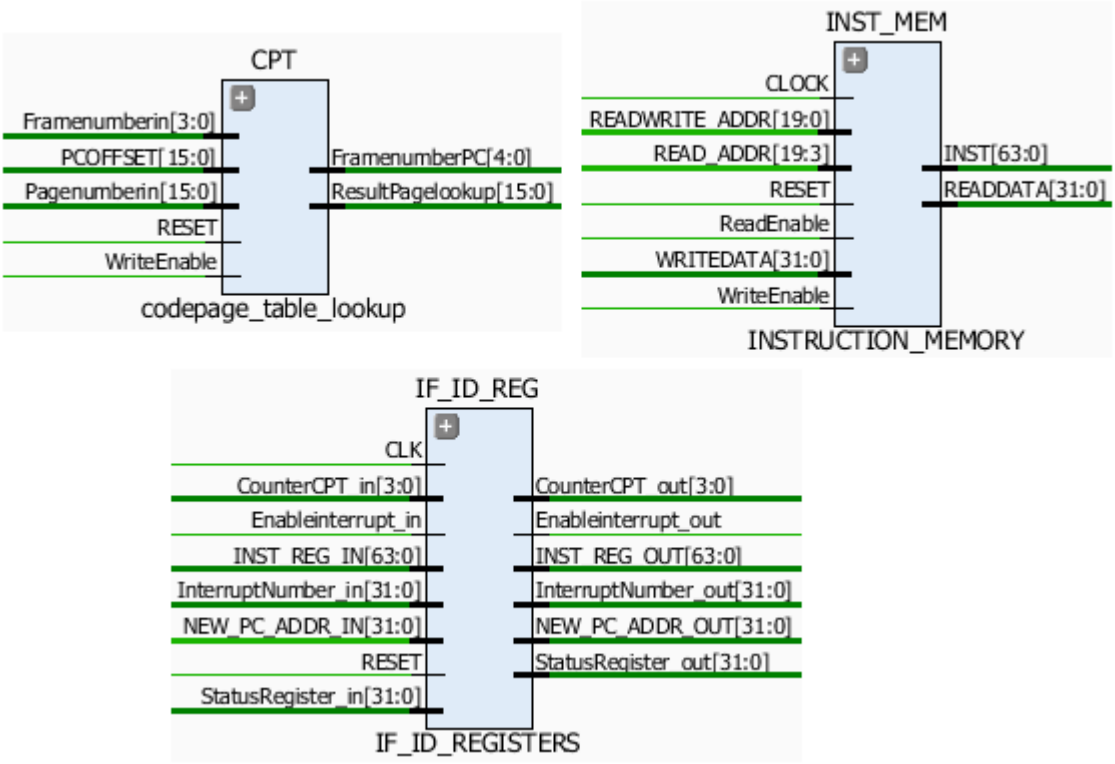
а) VHDL код за поставување на следната вредност на PC.



б) Интерфејси на VHDL модул за земање на инструкција.



в) Блок дијаграм на VHDL модул за земање инструкција.



г) Компоненти во VHDL модул за земање инструкција.

Слика 62 Опис на VHDL модел на единица за земање инструкција.

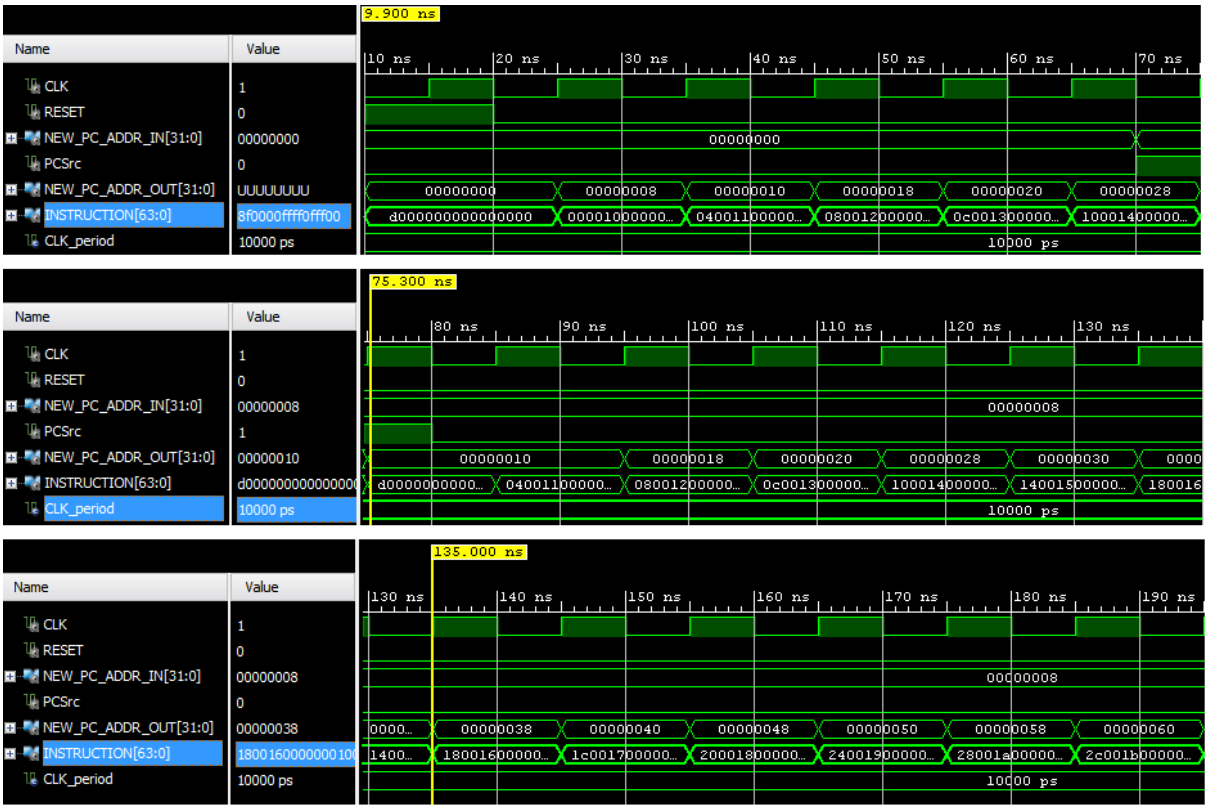
Откако е креиран VHDL модул за земање на инструкција, следен чекор е да се направи симулација на однесување на овој модул, со помош на Vivado симулаторот. За таа цел, напишана е тест програма со која се анализираат излезите за прочитана инструкција и следна вредност на PC, при последователно читање на инструкциите од меморија и при гранење (кога PCSrc=1). Се смета дека инструкцискатa меморија е веќе пополнета со некоја едноставна програма, прикажана на сл. 63 а). Тест програмата на почеток го поставува RESET сигналот на 1, па потоа на 0, што овозможува инструкциите да се земаат последователно. Во моментот кога PCSrc сигналот ќе стане 1, тогаш PC се поставува на адресата на гранење, од каде продолжува земањето на инструкции. На сл. 63 б) е прикажана тест програмата која се симулира, а во продолжение на сл. 63 в) е даден излезот од симулацијата.

```
signal InstructionMemory : rom_type := (
      0 => X"0000100000000100", --mem(16)=mem(0)==mem(1) споредба
      1 => X"0400110000000100", --mem(17)=mem(0)!=mem(1) споредба
      2 => X"0800120000000100", --mem(18)=mem(0)> mem(1) споредба
      3 => X"0C00130000000100", --mem(19)=mem(0)>=mem(1) споредба
      4 => X"1000140000000100", --mem(20)=mem(0)<mem(1)  споредба
      5 => X"1400150000000100", --mem(21)=mem(0)<=mem(1) споредба
      6 => X"1800160000000100", --mem(22)=mem(0)+ mem(1) собирање
      7 => X"1C00170000000100", --mem(23)=mem(0)- mem(1) одземање
      8 => X"2000180000000100", --mem(24)=mem(0)/mem(1)  делење
      9 => X"2400190000000100", --mem(25)=mem(0) mod mem(1)  mod дели
      10 => X"28001A0000000100", --mem(26)=mem(0)* mem(1)    множење
      11 => X"2C001B0000000100", --mem(27)=mem(0)and mem(1)   AND
      12 => X"30001C0000000100", --mem(28)=mem(0) or mem(1)   OR
      13 => X"34001D0000000100", --mem(29)=mem(0) xor mem(1)  XOR
      14 => X"38001E0000000100", --mem(30)=mem(0) xnor mem(1) XNOR
      15 => X"3C001F0000000100", --mem(31)=not mem(1)         NOT
      others => X"D000000000000000" --нема операција          NOP
);
```

а) Состојба на инструкциска меморија.

```
stim_proc: process
 begin
     wait for CLK_period;
     RESET<='1';
     wait for CLK_period;
            RESET<='0';
     wait for CLK_period*5;
            PCSrc<='1';                --скок
     NEW_PC_ADDR_IN<=x"00000008";      --адреса на скок
     wait for CLK_period;
            PCSrc<='0';
    wait;
 end process;
```

б) Тест програма за симулирање на VHDL модел на единица за земање инструкција.

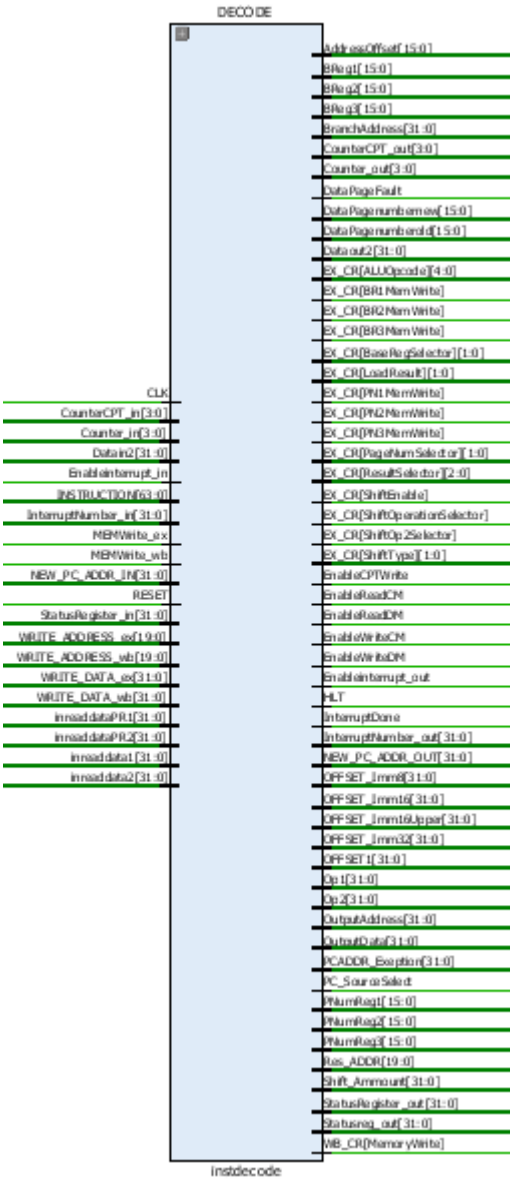


в) Резултати од симулација на VHDL модел на единица за земање инструкција.

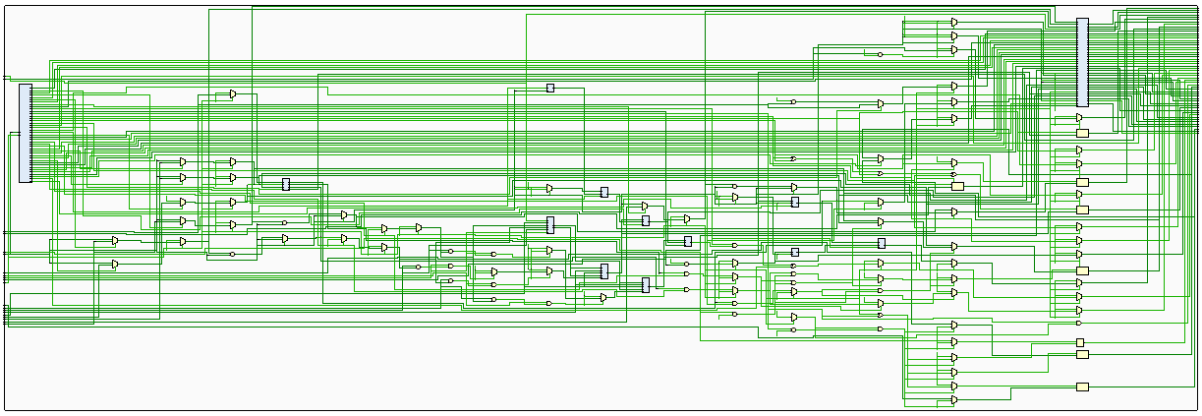
Слика 63 Симулирање на VHDL модел на единица за земање инструкција.

Модулот за декодирање (decode) е наменет да ја декодира инструкција која се проследува од модулот за земање инструкција и при тоа да ги прочита инструкциските операнди кои се сместени во податочната меморија на чипот. Покрај тоа, овој модул исто така извршува операции на поместување и проширување со знак за непосредно адресирани операнди, споредби за условно гранење, детекција на податочни конфликти и генерирање на контролни сигнали со контролната единица. Во согласност со претходниот опис и објаснување на принципот на работа на единицата за декодирање, даден во поглавје 4.3., направен е VHDL модел за истата. Всушност, VHDL моделот на единицата за декодирање ги имплементира поголемиот дел од функционалностите внатре во описот на архитектурата на ентитетот на единицата за декодирање, и при тоа референцира повеќе поединечни компоненти за: податочна меморија, табела на страници за податочна меморија, ID/EX проточен регистер, компаратор, контролна единица и неколку мултиплексери и проширувачи. На сл. 64 се дадени повеќе детали за проектираниот VHDL модел на единицата за декодирање, вклучувајќи интерфеј-

си, блок шема и некои поважни компоненти. Генерираните шеми се добиени во Vivado design suite софтверската околина.



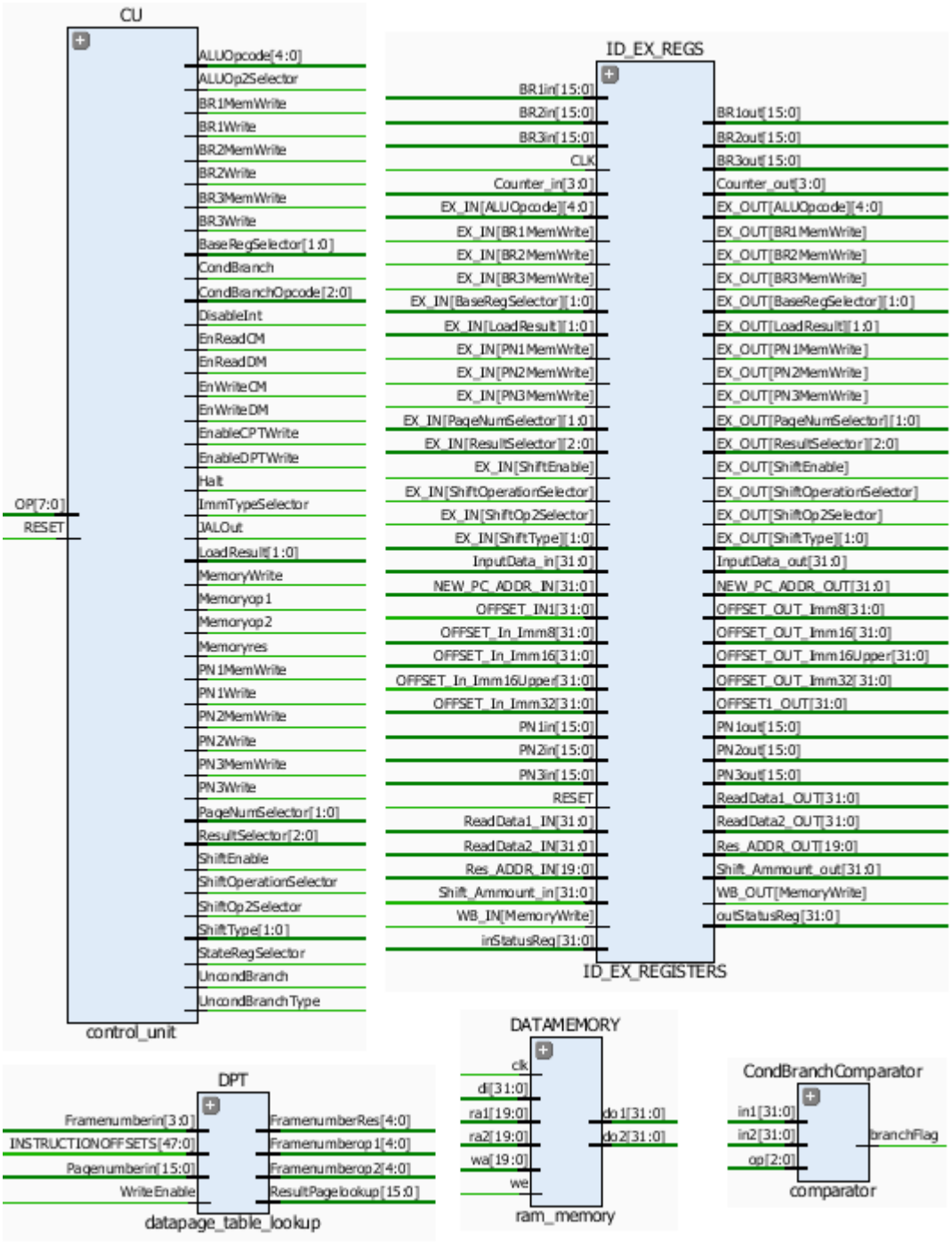
а) Интерфејси на VHDL модул за декодирање инструкција.



б) Блок дијаграм на VHDL модул за декодирање инструкција.

```
Addressop1 <=x"00000" when reset='1'  --Генерирање на адреса на операнд1
       else   FrNumop1(FRAMEADDR_SIZE-1 downto 0)&Frameoffset1 when Memoryop1='1'
       else   x"00000";
Addressop2 <=x"00000" when reset='1'  -- Генерирање на адреса на операнд2
       else   FrNumop2(FRAMEADDR_SIZE-1 downto 0)&Frameoffset2 when Memoryop2='1'
       else   x"00000";
AddressRes <=x"00000" when reset='1'  -- Генерирање на адреса на резултатен операнд
       else   FrNumRes(FRAMEADDR_SIZE-1 downto 0)&Frameoffset3 when Memoryres='1'
       else   x"00000";
```

в) VHDL код за генерирање на адреси на инструкциски операнди.



г) Компоненти во VHDL модул за декодирање инструкција.

Слика 64 Опис на VHDL модел на единица за декодирање инструкција

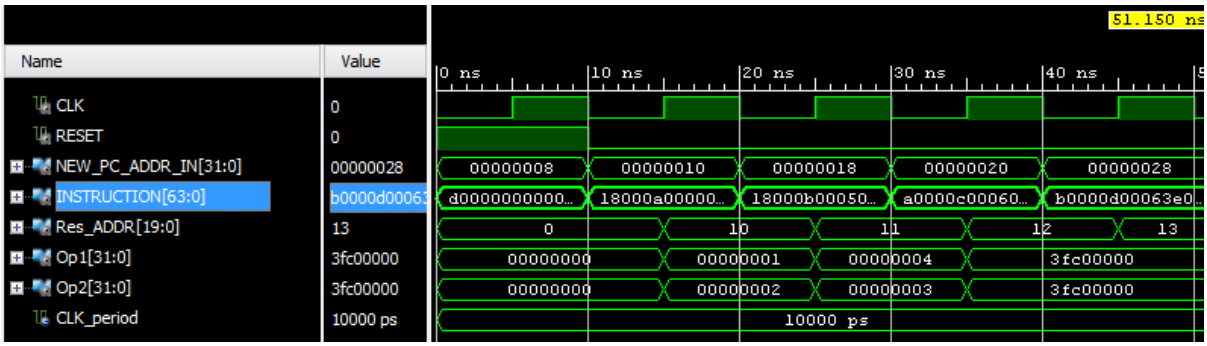
Откако е креиран VHDL модул за декодирање на инструкција, следен чекор е да се направи симулација на однесување на овој модул, со помош на Vivado симулаторот. За таа цел, напишана е тест програма со која се анализираат некои излези од единицата за декодирање (прочитани изворни операнди и генерирана адреса на резултат) за неколку влезни инструкции. Се смета дека податочната меморија е веќе пополнета со неколку податоци, како што е прикажано на сл. 65 а). Дополнително, на сл. 65 б) е претставена тест програмата која се симулира, а во продолжение на сл. 65 в) е даден излезот од симулацијата.

```
signal DataMemory : ram_type
:= (
      0 => X"00000001",
      1 => X"00000002",
      2 => X"00000000",
      3 => X"00000010",
      4 => X"00000003",
      5 => X"00000004",
      6 => X"3fc00000", --1.5 кога се користи како реален број
      7 => X"3fc00000", --1.5 кога се користи како реален број
      8 => X"00010000",
      9 => X"00000014",
      others => X"00000000"
);
```

а) Состојба на податочна меморија.

```
stim_proc: process
begin
            RESET<='1';
            INSTRUCTION<=X"D000000000000000";--нема операција
            NEW_PC_ADDR_IN<=x"00000008";
      wait for CLK_period;
            RESET<='0';
            INSTRUCTION<=X"18000A0000000100";--mem(10)=mem(0)+ mem(1)
            NEW_PC_ADDR_IN<=x"00000010";
      wait for CLK_period;
            INSTRUCTION<=X"18000B0005000400";--mem(11)=mem(5)+ mem(4)
            NEW_PC_ADDR_IN<=x"00000018";
      wait for CLK_period;
            INSTRUCTION<=X"A0000C0006000700";--mem(12)=mem(6)==mem(7) float
            NEW_PC_ADDR_IN<=x"00000020";
      wait for CLK_period;
            INSTRUCTION<=X"B0000D00063E0000";--mem(13)=mem(6)==1.5 float
            NEW_PC_ADDR_IN<=x"00000028";
wait;
end process;
```

б) Тест програма за симулирање на VHDL модел на единица за декодирање инструкција.



в) Резултати од симулација на VHDL модел на единица за декодирање инструкција.

Слика 65 Симулирање на VHDL модел на единица за декодирање инструкција.

Со оглед на тоа дека контролната единица претставува еден многу важен дел од единицата за декодирање, напишана е тест програма со која се анализира однесувањето на оваа компонента за неколку различни влезни инструкциски кодови на операција. Повеќе детали за програмата која се симулира се прикажани на сл. 66 а), додека пак на сл. 66 б) е даден излезот од симулацијата.

```
stim_proc: process
begin
            OP<=x"01";   --ALU_M операција со цели броеви
      wait for 10 ns;
            OP<=x"41";   --ALU_I операција со цели броеви
      wait for 10 ns;
            OP<=x"A0";   --ALU_M операција со реални броеви
      wait for 10 ns;
            OP<=x"B0";   --ALU_I операција со реални броеви
      wait for 10 ns;
            OP<=x"91";   --Shift_M операција на поместување
      wait for 10 ns;
            OP<=x"94";   --Shift_I операција на поместување
      wait for 10 ns;
            OP<=x"80";   --CondBranch_M операција на условен скок
      wait for 10 ns;
            OP<=x"88";   --CondBranch_I операција на условен скок
      wait for 10 ns;
            OP<=x"86";   --Jump_M безусловен скок JMem
      wait for 10 ns;
            OP<=x"87";   --JAL_M безусловен скок со зачувување на PC, JALMem
      wait for 10 ns;
            OP<=x"C0";   --SetPBR_I поставување на базни/странични регистри
      wait for 10 ns
            OP<=x"D7";   --SetPBR_M поставување на базни/странични регистри
      wait for 10 ns;
            OP<=x"E0";   --Load_I вчитување на константа во подат. меморија
      wait;
end process;
```

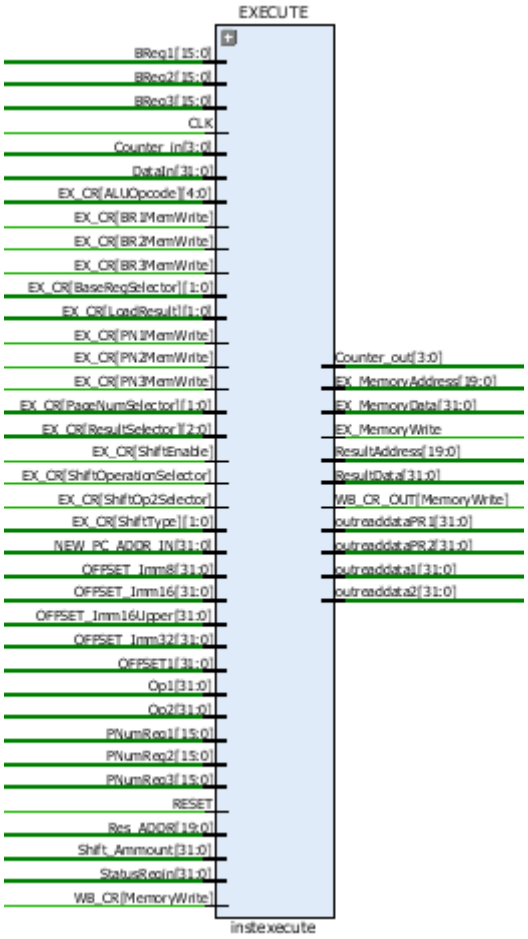
а) Тест програма за симулирање на VHDL модел на контролна единица.

б) Резултати од симулација на VHDL модел на контролна единица.

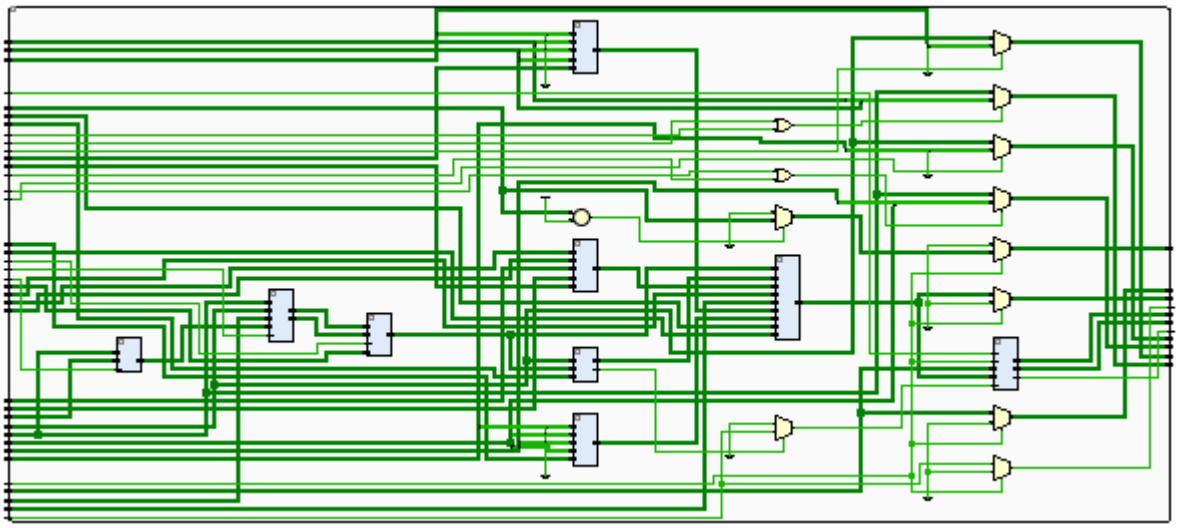
Слика 66 Симулирање на VHDL модел на контролна единица.

Модулот за извршување (execute) е наменет да извршува аритметичко-логички операции и операции на поместување, и воедно да ја селектира вредноста на резултатот (резултат од ALU единица, резултат од поместувач, регистер на страна итн.) која треба да се запише во податочната меморија. Дополнително овој модул исто така извршува препраќање на резултатните вредност и адреса кон модулот за декодирање со цел да спречи појава на податочни конфликти. Во согласност со претходниот опис и објаснување на принципот на работа на единицата за извршување, даден во поглавје 4.3., направен е VHDL модел за оваа единица. Овој VHDL модел ги имплементира поголемиот дел од функционалностите на единицата за извршување со помош на неколку компоненти, вклучувајќи: АЛУ единица за цели и реални броеви, поместувач, EX/WB проточен регистер, мултиплексер за селекција на резултат и неколку дополнителни мултиплексери. На сл. 67 се дадени повеќе детали за проектираниот VHDL модел на единицата за извршување, вклучувајќи интерфејси, блок шема и некои

поважни компоненти. Генерираните шеми се добиени во Vivado design suite софтверската околина.



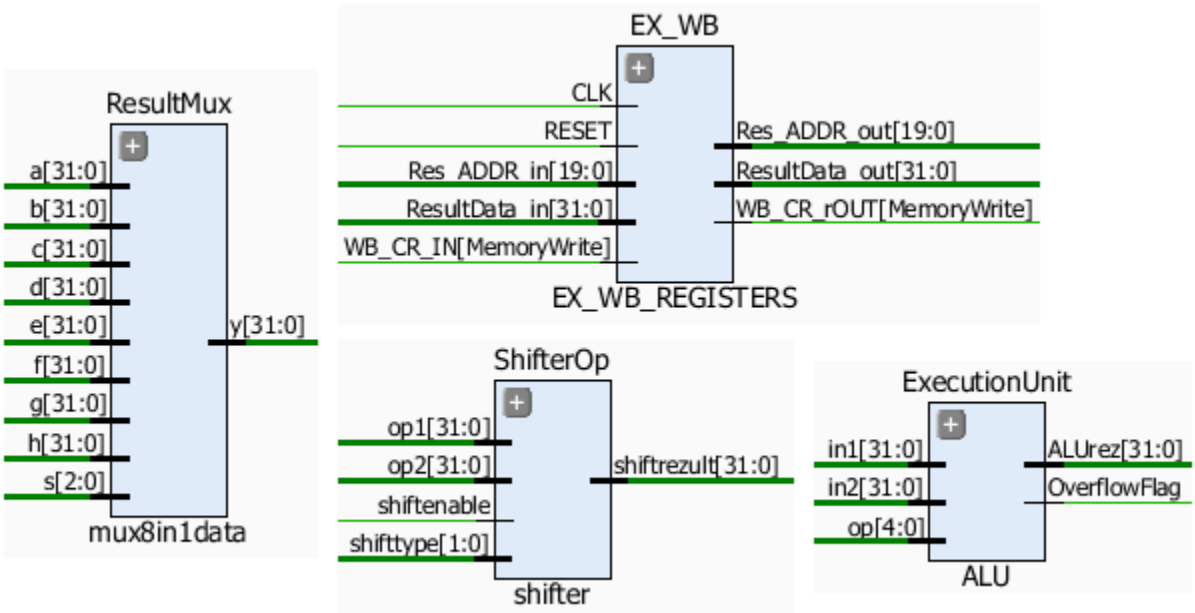
а) Интерфејси на VHDL модул за извршување на инструкција.



б) Блок дијаграм на VHDL модул за извршување на инструкција.

```
library ieee_proposed; --библиотека со аритметички функции за реални броеви
use ieee_proposed.fixed_float_types.all;
use ieee_proposed.float_pkg.all;
use ieee.numeric_std.all; --библиотека со аритмет. функции за цели  броеви
...
main: process(in1,in2,op)
begin
      case op is
      ...
      when "00100"=>    --постави ако е помало, за цели броеви
            if(in1 < in2) then
                  ALUrez<=x"00000001";
            else
                  ALUrez<=x"00000000";
            end if;
      when "00110" =>   --собирање на цели броеви
            ALUrez<=std_logic_vector(signed(in1)+signed(in2));
      when "00111"=>    --одземање на цели броеви
            ALUrez<=std_logic_vector(signed(in1)-signed(in2));
      when "01011"=>    --логичко AND на цели броеви
            ALUrez<=in1 and in2;
      when "10100"=>    -- постави ако е помало, за реални броеви
            if(to_float(in1, tmp) < to_float(in2, tmp))
                  ALUrez<=x"00000001";
            else
                  ALUrez<=x"00000000";
            end if;
      when "11001"=>    --множење на реални броеви
      ALUrez<=to_slv(to_float(in1, tmp)*to_float(in2, tmp));
      ...
      end case;
end process;
```

в) VHDL код кој имплементира дел од ALU единицата.



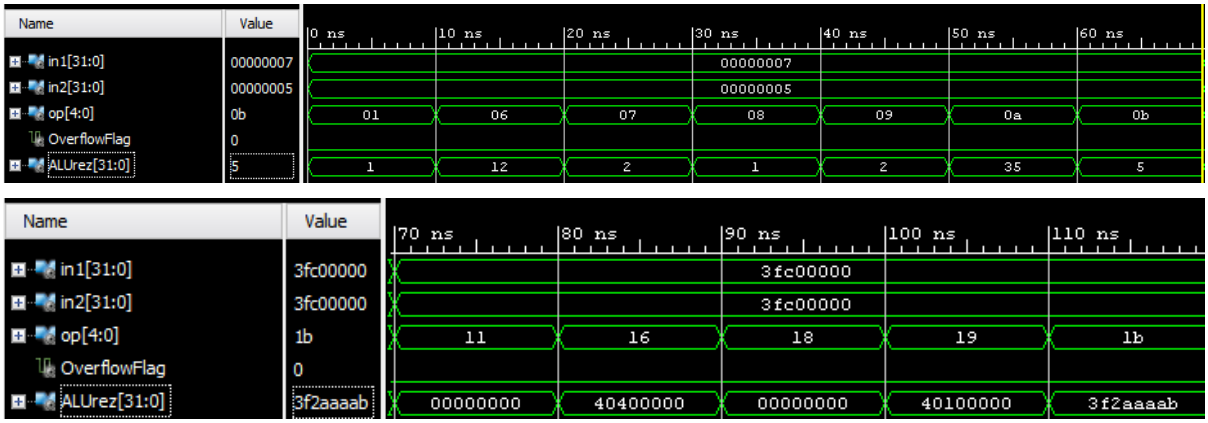
г) Компоненти во VHDL модул за извршување на инструкција.

Слика 67 Опис на VHDL модел на единица за извршување на инструкција

Со оглед на тоа дека аритметичко-логичката единица претставува еден многу важен дел од единицата за извршување, напишана е тест програма со која се анализира однесувањето на оваа компонента за неколку различни влезни кодови на ALU операција. Повеќе детали за програмата која се симулира се прикажани на сл. 68 а), додека пак на сл. 68 б) е даден излезот од симулацијата.

```
stim_proc: process
begin
            in1<=x"00000007";  --op1=7 како цел број
            in2<=x"00000005";  --op2=5 како цел број
            op<="00001";  --постави ако не е еднакво, за цели броеви
      wait for 10 ns;
            op<="00110";  --собирање на цели броеви
      wait for 10 ns;
            op<="00111";  --одземање на цели броеви
      wait for 10 ns;
            op<="01000";  --делење на цели броеви
      wait for 10 ns;
            op<="01001";  --modulo делење на цели броеви
      wait for 10 ns;
            op<="01010";  --множење на цели броеви
      wait for 10 ns;
            op<="01011";  --логичко AND на цели броеви
      wait for 10 ns;
            in1<=x"3fc00000";  --op1=1.5 како реален број
            in2<=x"3fc00000";  --op2=1.5 како реален број
            op<="10001";   --постави ако не е еднакво, за реални броеви
      wait for 10 ns
            op<="10110";   --собирање на реални броеви
      wait for 10 ns;
            op<="11000";   --одземање на реални броеви
      wait for 10 ns;
            op<="11001";   --множење на реални броеви
      wait;
end process;
```

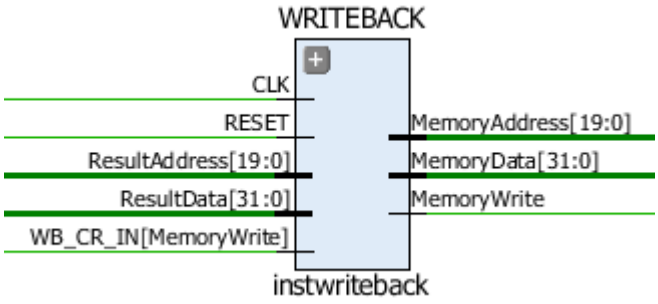
а) Тест програма за симулирање на VHDL модел на аритметичко-логичка единица.



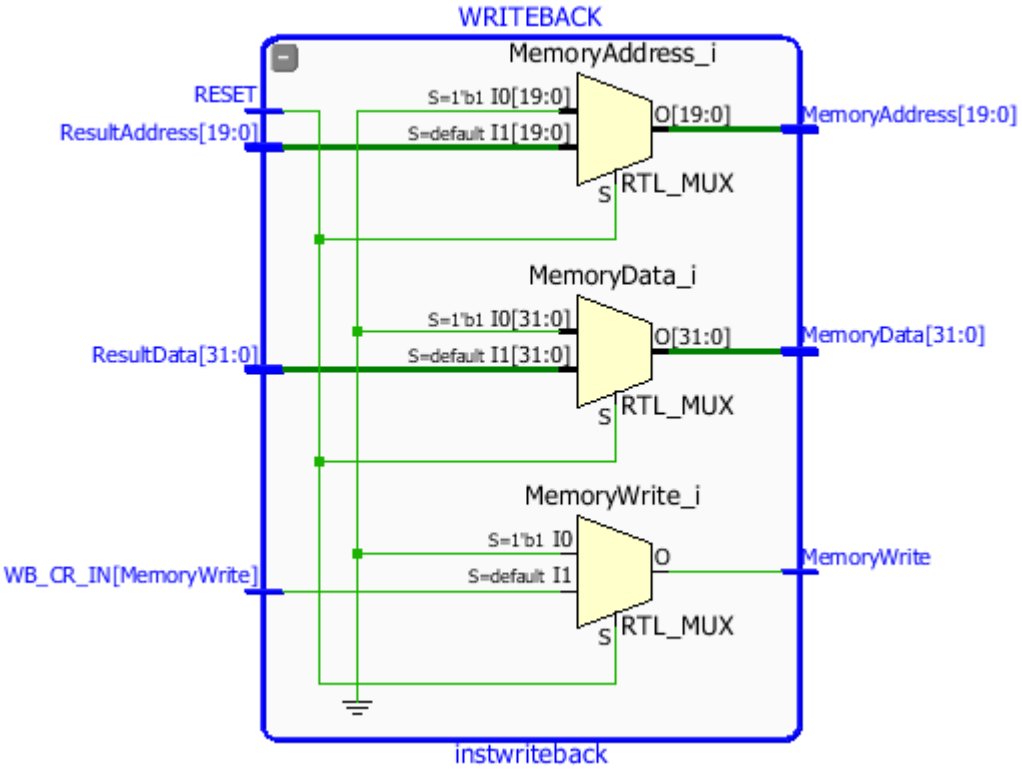
б) Резултати од симулација на VHDL модел на аритметичко-логичка единица.

Слика 68 Симулирање на VHDL модел на аритметичко-логичка единица.

Модулот за запишување на резултат (write-back) е наменет да ја запише резултатната вредност во податочната меморија на чипот и да обезбеди препраќање на резултатот до модулот за декодирање со цел да спречи појава на податочни конфликти. Во согласност со претходниот опис и објаснување на принципот на работа на единицата за запишување резултат, даден во поглавје 4.3., направен е VHDL модел за оваа единица. VHDL моделот на единицата за запишшување резултат претставува само интерфејс кон модулот за декодирање, каде всушност се извршуваат операциите на запишување и разрешување на конфликти. На сл. 69 се дадени повеќе детали за интерфејсот и шемата на проектираниот VHDL модел на единицата за запишување резултат во Vivado design suite софтверската околина.



а) Интерфејси на VHDL модул за запишување на резултат.



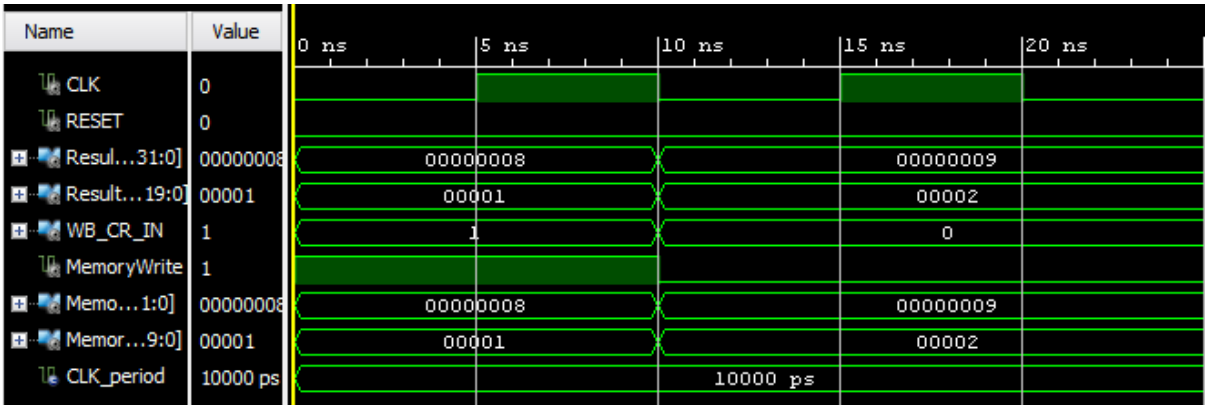
б) Блок дијаграм на VHDL модул за запишување на резултат.

Слика 69 Опис на VHDL модел на единица за запишување на резултат.

Откако е креиран VHDL модул за запишување на резултат, следен чекор е да се направи симулација на однесување на овој модул, со помош на Vivado симулаторот. За таа цел, напишана е тест програма со која се анализира однесувањето на овој модул за неколку различни вредности на влезот. Повеќе детали за програмата која се симулира се прикажани на сл. 70 а), додека пак на сл. 70 б) е даден излезот од симулацијата.

```
stim_proc: process
begin
            ResultData <= x"00000008";
            ResultAddress <= x"00001";
            WB_CR_IN.MemoryWrite <= '1';
      wait for CLK_period;
            ResultData <= x"00000009";
            ResultAddress <= x"00002";
            WB_CR_IN.MemoryWrite <= '0';
      wait;
end process;
```

а) Тест програма за симулирање на VHDL модел на единица за запишување на резултат.



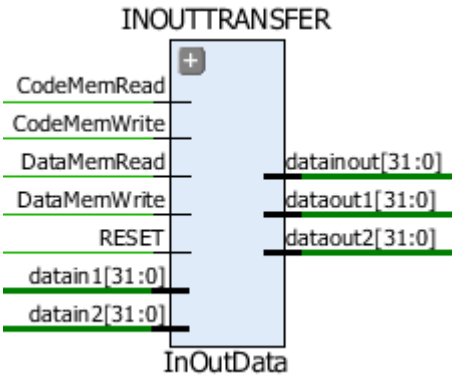
б) Резултати од симулација на VHDL модел на единица за запишување на резултат.

Слика 70 Симулирање на VHDL модел на единица за запишување на резултат.

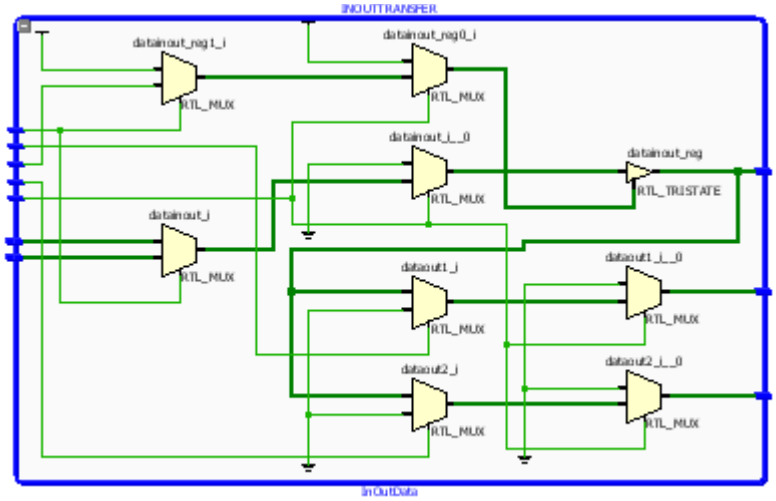
Модулот за комуникацијата со В/И уреди е наменет да обезбеди трансфер на податоци помеѓу некој В/И уред и MIMOPS процесорот (инструкциска или податочна меморија во чип) со помош на IN или OUT инструкции, кои се претходно објаснети во поглавје 4.2. За таа цел, овој модул користи влезно/излезна податочна магистрала преку која врши прием на податоци од В/И уред кон својата внатрешна меморија (при извршување на IN инструкција) или испраќа податоци кон В/И уред од својата внатрешна меморија (при извршување на OUT инструкција). VHDL моделот на единицата за комуникацијата со В/И уреди заедно со својата шема, генерирана во Vivado design suite, се прикажани на сл. 71.

```
datainout    <=    zero32b when RESET='1' else    --OUT инструкција
                   datain1 when CodeMemRead='1' else
                   datain2 when DataMemRead='1' else
                   (others => 'Z');
dataout1     <=    zero32b when RESET='1' else    --IN инструкција
                   datainout when CodeMemWrite='1' else zero32b;
dataout2     <=    zero32b when RESET='1' else    --IN инструкција
                   datainout when DataMemWrite='1' else zero32b;
```

а) VHDL код на модул за комуникација со В/И уреди.



б) Интерфејси VHDL модул за комуникација со В/И уреди.



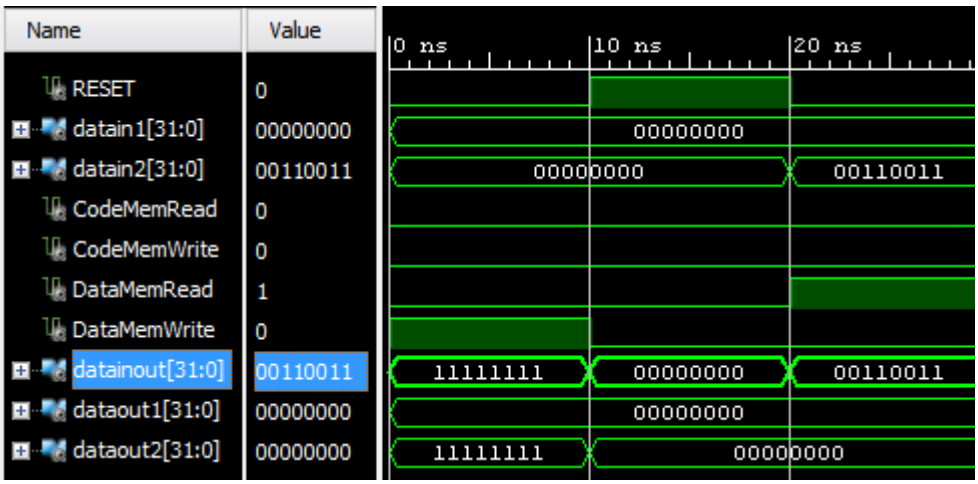
в) Блок дијаграм на VHDL модул за комуникација со В/И уреди.

Слика 71 Опис на VHDL модел на единица за комуникација со В/И уреди.

Откако е креиран VHDL модул за комуникација со В/И уреди, направена е негова симулација во Vivado симулаторот. За таа цел, напишана е тест програма која го анализира однесувањето на овој модул за неколку различни вредности на влезот. Повеќе детали за програмата која се симулира се прикажани на сл. 72 а), додека пак на сл. 72 б) е даден излезот од симулацијата.

```
stim_proc: process
begin
            datainout <= x"11111111"; --Запишување во подат. меморија (IN)
            DataMemWrite<='1';
      wait for 10 ns;
            RESET<='1';               --RESET на состојбата
            DataMemWrite<='0';
            datainout <= (others => 'Z');
      wait for 10 ns;
            RESET<='0';               --Читање од податочна меморија (OUT)
            datain2<=x"00110011";
            DataMemRead<='1';
            datainout <= (others => 'Z');
      wait;
end process;
```

а) Тест програма за симулирање на VHDL модел на единица за комуникација со В/И уреди.



б) Резултати од симулација на VHDL модел на единица за комуникација со В/И уреди.

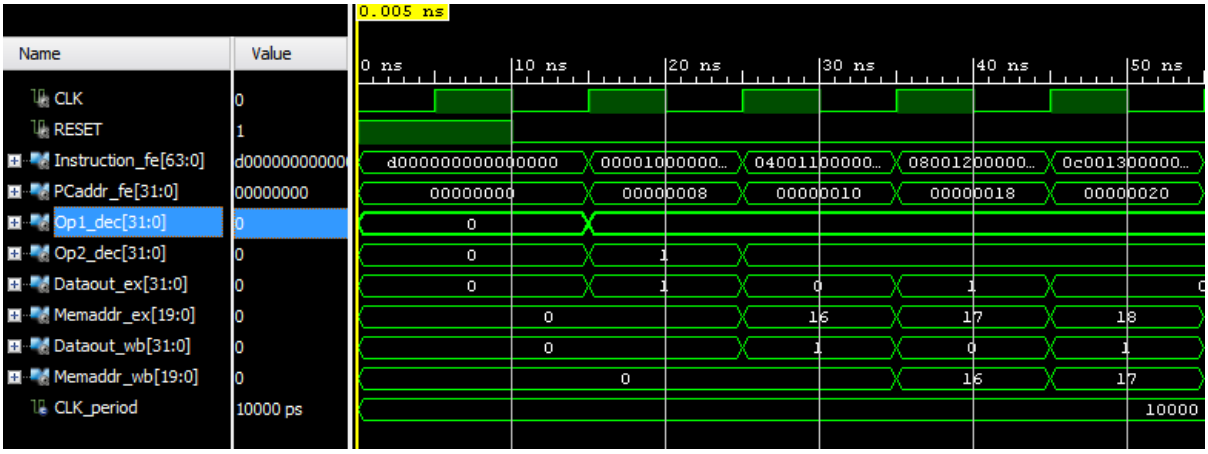
Слика 72 Симулирање на VHDL модел на единица за комуникација со В/И уреди.

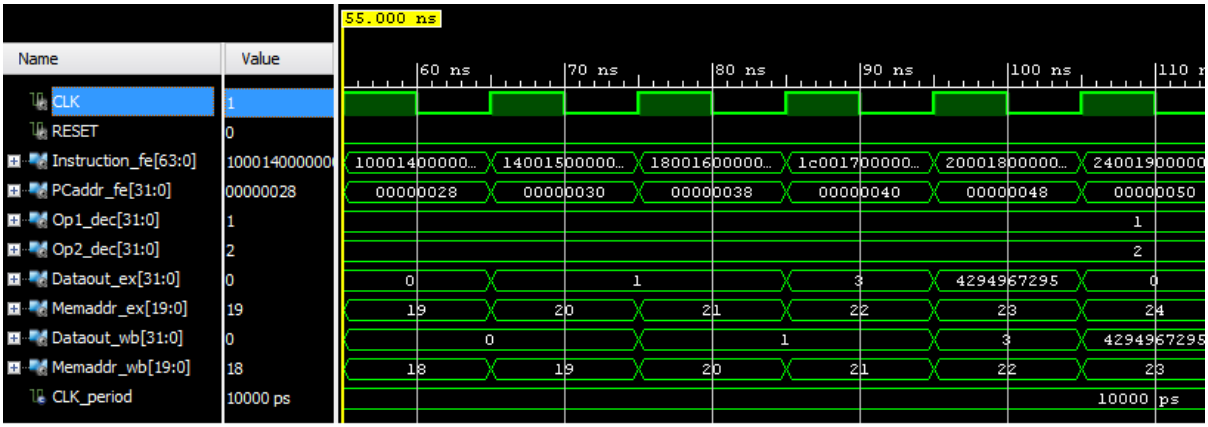
Откако ќе се дизајнираат поединечните модули, се прави нивно меѓусебно поврзување и на тој начин се креира VHDL модел на MIMOPS процесорот. Шемата на VHDL моделот на MIMOPS процесорот која се генерира во Vivado design suite софтверската околина, веќе беше прикажана на сл. 61. Со оглед на тоа дека секој од поединечните модули на MIMOPS процесорот е веќе верификуван со посебна тест програма, следниот чекор е да се направи симулација на однесувањето на комплетниот MIMOPS процесор, со Vivado симулаторот. За таа цел, напишана е тест програма која го анализира однесувањето на процесорот при извршување на една едноставна програма, сместена во инструкциската меморија, а прикажана на сл. 63 а). Дополнително се смета дека податочната меморија е пополнета со неколку податоци, како што е прикажано на сл. 65 а). Повеќе

детали за програмата која се симулира се прикажани на сл. 73 а), додека пак на сл. 73 б) е даден излезот од симулацијата.

```
...
--Излези од MIMOPS процесор кои се анализираат
Instruction_fe : out STD_LOGIC_VECTOR(INST_SIZE-1 downto 0);
--Инструкција која е прочитана во фаза на земање
PCaddr_fe : out STD_LOGIC_VECTOR(PC_SIZE-1 downto 0);
--Нова вредност на PC која е генерирана во фаза на земање
Op1_dec : out STD_LOGIC_VECTOR(DATA_SIZE-1 downto 0);
--Операнд1 кој е прочитан во фаза на декодирање
Op2_dec : out STD_LOGIC_VECTOR(DATA_SIZE-1 downto 0);
--Операнд2 кој е прочитан во фаза на декодирање
Dataout_ex : out STD_LOGIC_VECTOR (DATA_SIZE-1 downto 0);
--Резултатен операнд кој е генериран во фаза на извршување
Memaddr_ex : out STD_LOGIC_VECTOR (LOCMEMADDR_SIZE-1 downto 0);
--Адреса на резултат која се пренесува од фаза на извршување
Dataout_wb : out STD_LOGIC_VECTOR (DATA_SIZE-1 downto 0);
--Резултатен операнд кој се пренесува од фаза на запишување резултат
Memaddr_wb : out STD_LOGIC_VECTOR (LOCMEMADDR_SIZE-1 downto 0);
--Адреса на резултат која се пренесува од фаза на запишување резултат
...
stim_proc: process
begin
            RESET<='1';        --RESET на состојбата на процесорот
      wait for CLK_period
            RESET<='0';        --Процесорот започнува да извршува програма
      wait;
end process;
```

а) Тест програма за симулирање на VHDL модел на MIMOPS процесор.

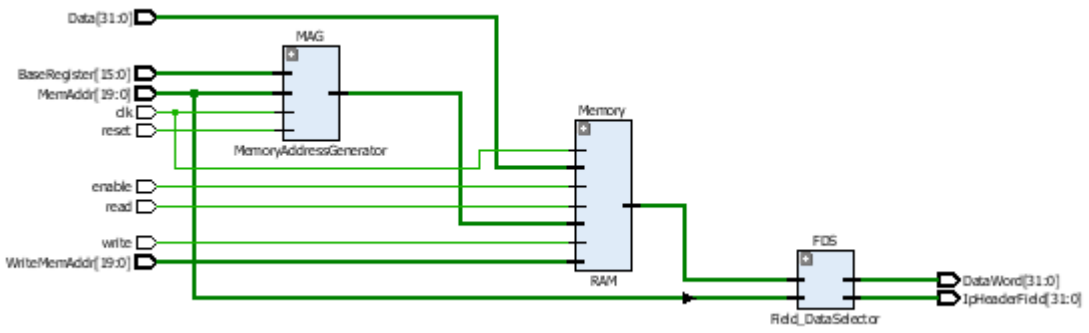




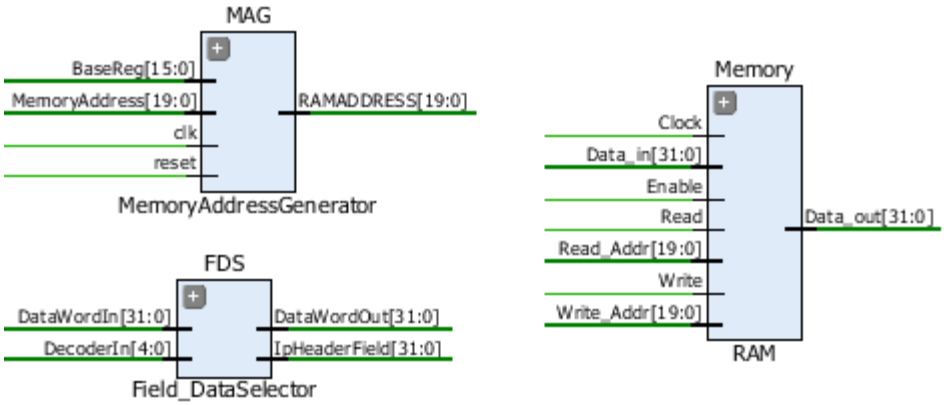
б) Резултати од симулација на VHDL модел на MIMOPS процесор.

Слика 73 Симулирање на VHDL модел на MIMOPS процесор.

Дизајнираниот MIMOPS процесор е проширен со логика за парсирање на заглавија на IP пакети, која обезбедува директна работа со полиња од IPv4 или IPv6 заглавија на пакети, кои не се со ширина на збор. Дадениот парсер на заглавија на IP пакети се додава до внатрешната меморија на процесорот (која содржи заглавија на IP пакети) и на тој начин овозможува MIMOPS процесорот да се употребува во мрежно процесирање на пакети. Во согласност со претходниот опис на единицата за парсирање на заглавија на IP пакети, даден во поглавје 5.3., направен е VHDL модел за оваа единица. Овој VHDL модел ги имплементира функционалностите на единицата за парсирање на IP пакети со помош на три компоненти: генератор на мемориски адреси за поле/податок, податочна меморија во чип и селектор на поле/податок. Всушност, хардверската логика која е потребна за директно читање и запишување на поле од IP заглавие се имплементирани според шемите кои се дадени на сл. 49 и сл. 51, соодветно. Повеќе детали за VHDL моделот на единицата за парсирање на IP заглавија, проектирана во Vivado design suite софтверската околина, се дадени на сл. 74.



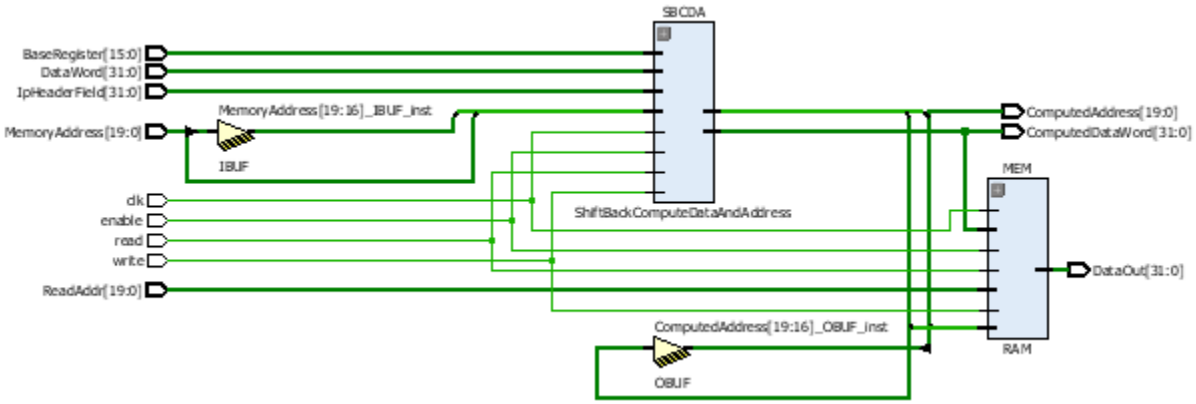
а) Блок дијаграм на VHDL модул на хардвер за парсирање на IP заглавија, потребен за читање на поле од IP заглавие.



б) Компоненти во VHDL модул на единица за парсирање на IP заглавие (хардвер за читање).

```
...
LUTsignalout <=
     X"0000" when LUTin = X"0000" else     --IPv4 верзија
     X"0000" when LUTin = X"0001" else     --IPv4 должина на заглавие
     X"0000" when LUTin = X"0002" else     --IPv4 тип на сервис
     X"0000" when LUTin = X"0003" else     --IPv4 прв збор прва половина
     X"0000" when LUTin = X"0004" else     --IPv4 вкупна должина
     X"0001" when LUTin = X"0005" else     --IPv4 идентификатор
     X"0001" when LUTin = X"0006" else     --IPv4 знаменца
     X"0001" when LUTin = X"0007" else     --IPv4 отстапување на Фрагмент
     X"0001" when LUTin = X"0008" else     --IPv4 втор збор втора половина
     X"0002" when LUTin = X"0009" else     --IPv4 време на живот
...
```

в) VHDL код кој имплементира look-up табела во генераторот на мемориски адреси.



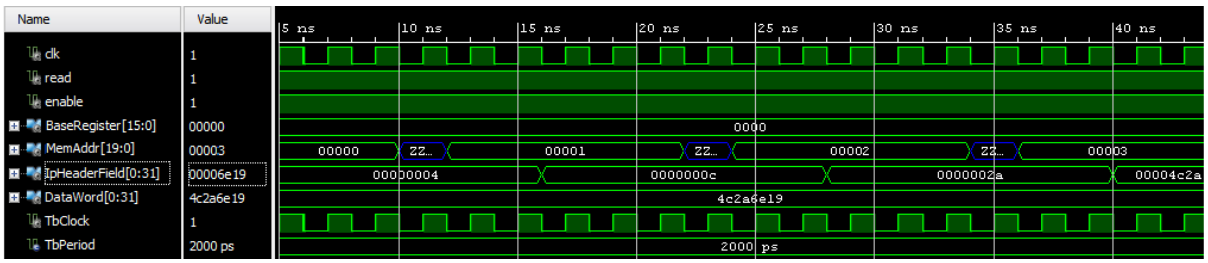
г) Блок дијаграм на VHDL модул на хардвер за парсирање на IP заглавија, потребен за запишување на поле од IP заглавие. Генераторот на мемориски адреси и селекторот на поле/податок се интегрирани во една компонента - SBCDA.

Слика 74 Опис на VHDL модел на единица за парсирање на IP заглавија на пакети.

Откако е креиран VHDL модул на единицата за парсирање на IP заглавија на пакети, направена е негова симулација во Vivado симулаторот. За таа цел, напишана е тест програма која го анализира однесувањето на овој модул, при екстрахирање на полиња од IPv4 заглавие, кое е сместено во меморија, [66]. Се смета дека вредноста на првиот збор од IPv4 заглавието кое се испитува е: $4c2a6e19_{16}$ Повеќе детали за тест програмата која се симулира се прикажани на сл. 75 а), додека пак на сл. 75 б) е даден излезот од симулацијата.

```
stim_proc: process
begin
            enable <= '1';
            read <= '1';                --Читање на полиња од IPv4 заглавие
            BaseRegister <= x"0000";
            MemAddr <= x"00000";        --IPv4 верзија
      wait for 10 ns;
            MemAddr <= x"ZZZZZ";
      wait for 2 ns;
            BaseRegister <= x"0000";
            MemAddr <= x"00001";         --IPv4 должина на заглавие
      wait for 10 ns;
            MemAddr <= x"ZZZZZ";
      wait for 2 ns;
            BaseRegister <= x"0000";
            MemAddr <= x"00002";         --IPv4 тип на сервис
      wait for 10 ns;
            MemAddr <= x"ZZZZZ";
      wait for 2 ns;
            BaseRegister <= x"0000";
            MemAddr <= x"00003";         --IPv4 прв збор прва половина
      wait;
end process;
```

а) Тест програма за симулирање на VHDL модел на единица за парсирање на IP заглавија на пакети.



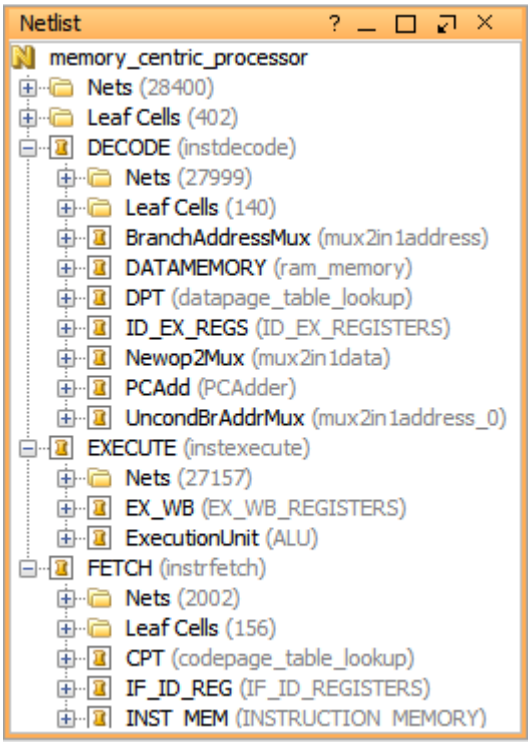
б) Резултати од симулација на VHDL модел на единица за парсирање на IP заглавија на пакети.

Слика 75 Симулирање на VHDL модел на единица за парсирање на IP заглавија на пакети.

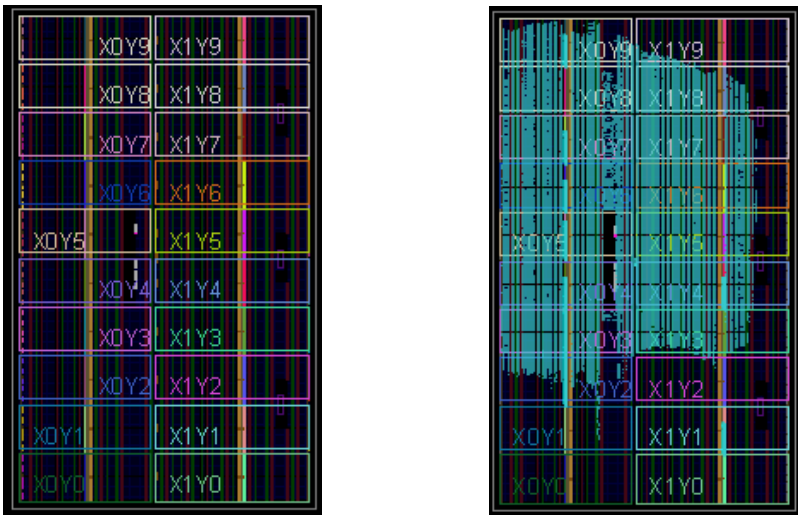
6.4. Синтеза и имплементација на предложениот MIMOPS процесот во Virtex7 VC709 развојна околина

Откако е направена симулација и верификација на VHDL моделот на MIMOPS процесорот, следен чекор е да се изврши синтеза и имплементација на соодветниот процесор со помош на Vivado design suite софтверската околина. Овие активности се извршуваат автоматски со алатките за синтеза и имплементација, кои се претходно поставени да овозможат FPGA реализација на MIMOPS процесорот во Virtex7 VC709 развојна плоча (претставена во поглавје 6.2).

Во фазата на синтеза се врши конвертирање на VHDL моделот на MIMOPS процесорот во мрежна листа, која е составена од генерички компоненти меѓусебно поврзани со врски. На сл. 76 се дадени повеќе детали за генерираната мрежна листа на MIMOPS процесорот. Веднаш после синтезата, Vivado алатката за имплементирање ги извршува фазите на: преведување, мапирање, сместување и рутирање. На тој начин, MIMOPS процесорот се мапира и преведува во компонентите на Xilinx Virtex7 XC7VX690T FPGA плочата, после што овие компоненти физички се сместуваат и меѓусебно поврзуваат (рутираат) на соодветната FPGA плоча. На сл. 77 е прикажано како изгледа Virtex7 VC709 FPGA уредот, после синтезата и имплементацијата на MIMOPS процесорот.



Слика 76 Мрежна листа на MIMOPS процесор.



а) FPGA после синтеза. б) FPGA после имплементација.

Слика 77 FPGA имплементација на MIMOPS процесор во Virtex7 VC709 развојна плоча.

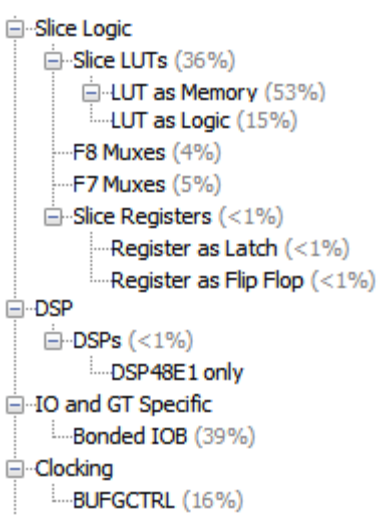
Откако е завршена имплементацијата на процесорот, се генерираат подетални извештаи за хардверските карактеристики на дизајнираниот MIMOPS процесор. Како резултат на тоа, во продолжение е дискутиран извештајот за искористеност на ресурси, кој содржи детали за искористеноста на ресурси на Virtex7 VC709 FPGA уредот, поле имплементацијата на MIMOPS процесорот. Дополнително, на истата FPGA плоча е имплементиран и VHDL модел на MIPS процесор (превземен од [106]), при што генерираниот извештајот за искористеност на ресурси е употребен за да се спореди комплексноста на предложениот MIMOPS процесор со MIPS процесорот (кој беше употребен како основа за дизајн на MIMOPS процесорот). На сл. 78 се дадени повеќе детали за оваа анализа.

Name	Slice LUTs (433200)	Slice Registers (866400)	F7 Muxes (216600)	F8 Muxes (108300)	DSPs (3600)	Bonded IOB (850)	BUFGCTRL (32)
memory_centric_processor	155810	1346	9894	4388	3	333	5
DECODE (instdecode)	111383	790	8773	4372	0	0	0
EXECUTE (instexecute)	40606	53	1	0	3	0	0
FETCH (instrfetch)	3820	503	1120	16	0	0	0

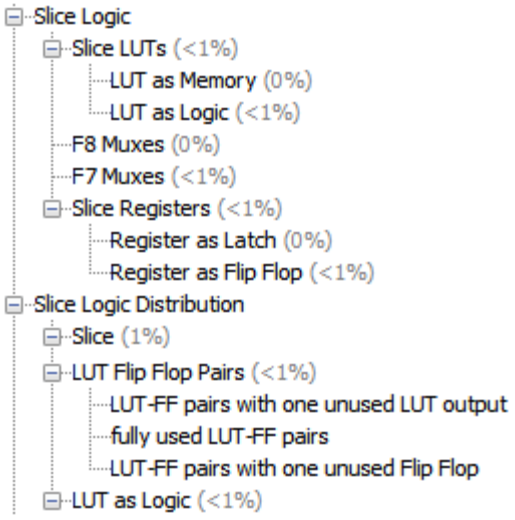
а) Извештај за искористеност на FPGA ресурси за MIMOPS процесор.

Name	Slice LUTs (433200)	Slice Registers (866400)	F7 Muxes (216600)	Slice (108300)	LUT as Logic (433200)	LUT Flip Flop Pairs (433200)	Bonded IOB (850)	BUFGCTRL (32)
SEGMENTED_MIPS	863	1455	256	519	863	92	194	1
MEM_ACC (M...	49	38	0	30	49	1	0	0
INST_FETCH ...	84	148	0	68	84	34	0	0
INST_DEC...	702	1167	256	463	702	3	0	0
EXE (EXECUT...	28	102	0	59	28	1	0	0

б) Извештај за искористеност на FPGA ресурси за MIPS процесор.



в) FPGA искористеност(%) за MIMOPS процесор.



г) FPGA искористеност(%) за MIPS процесор.

Слика 78 Анализа на искористеност на Virtex7 VC709 FPGA ресурси за MIMOPS и MIPS процесор.

Според сл. 78 може да се заклучи дека MIMOPS процесорот е многу покомплексен од MIPS процесорот. Овој резултат е очекуван, бидејќи MIMOPS процесорот ја интегрира меморијата во чипот и имплементира комплексни механизми за да обезбеди хардверска поддршка за работа со виртуелната меморија (вклучува генератори на мемориски адреси со табели на страници во чип и врши управување со мемориски блокови итн). Дополнително, MIMOPS процесорот поддржува повеќе инструкциски формати, што пак додава комплексност во контролната единица. Покрај контролната единица, дополнителна комплексност додаваат и: единицата за справување со податочни конфликти, компаративната логика за условно гранење во фазата на декодирање, ALU единицата која е проширена да работи со реални броеви, и единицата за поместување која обезбедува поддршка за употреба на втор изворен флексибилен операнд. Сите овие хардверски единици се имплементирани со цел да ги подобрат пресметувачките перформанси на MIMOPS процесорот, што всушност е постигнато, но комплексноста на чипот е зголемена. Кога станува збор за логиката за парсирање на IP заглавија може да се каже дека оваа единица искористува само 0.01% од slice регистрите и 0.35% од slice LUT логичките ресурси, што е помалку од 1% од расположливите Virtex7 VC709 FPGA slice ресурси. Тоа значи дека дадената логика за парсирање на IP заглавија би можела да се додаде во било која процесорска архитектура и на тој начин да обезбеди подобри перформансите при мрежно процесирање, на сметка на мало зголемување на хардверската комплексност на чипот.

6.5. Програмирање на Virtex7 7VX690T FPGA чип

За да може да се програмира Virtex7 VC709 FPGA плочата неопходно е да се подготви датотека со ограничувања (constraint file), во која се специфицира поврзувањето на сигналите од VHDL кодот на MIMOPS процесорот со пиновите од Virtex7 VC709 развојната плоча. На пример, сигналот за reset на процесорот се поврзува со копчето за reset, кое му овозможува на корисникот мануелно да го ресетира процесорот. На сличен начин, CLK сигналот се поврзува со 200 MHz системски такт сигнал од FPGA плочката, кој е активен на растечка ивица. Дополнително, последните 8 бита од ResultData сигналот кој се препраќа од фазата на извршување кон фазата за запишување на резултат се поврзуваат со 8 LED диоди од FPGA плочата. Повеќе детали за поврзувањето на В/И пинови од Virtex7 VC709 плоча со сигнали од MIMOPS процесор се дадени во Табела 13.

Табела 13 Поврзување на В/И пинови од Virtex7 VC709 плоча со сигнали од MIMOPS процесор.

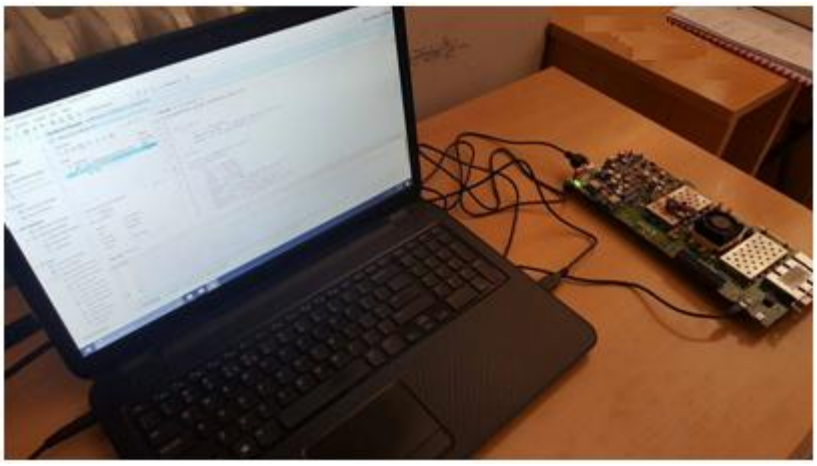
Сигнал	Пин	Тип на пин	Функција на пин
CLK	H19	IN	200 MHz системски такт активен на растечка ивица
Reset	AV40	IN	Копче за Reset
ResultData[7]	AU39	OUT	Корисничка LED 7 диода
ResultData[6]	AP42	OUT	Корисничка LED 6 диода
ResultData[5]	AP41	OUT	Корисничка LED 5 диода
ResultData[4]	AR35	OUT	Корисничка LED 4 диода
ResultData[3]	AT37	OUT	Корисничка LED 3 диода
ResultData[2]	AR37	OUT	Корисничка LED 2 диода
ResultData[1]	AN39	OUT	Корисничка LED 1 диода
ResultData[0]	AM39	OUT	Корисничка LED 0 диода

После програмирањето на FPGA уредот, корисникот може да го анализира извршувањето на некоја програма која е веќе вчитана во инструкцискатa меморија на чипот на процесорот, преку набљудување на промените на состојбата на LED диодите. Се смета дека дадената програма работи со броеви во опсегот од [0-255]. Со оглед на тоа дека MIMOPS процесорот кој користи такт сигнал од 200MHz ја извршува програмата многу брзо, дефинирана е дополнителна компонента која го скалира влезниот 200MHz такт сигнал во 1 Hz такт сигнал (со период од 1s). На тој начин, промената на состојбата на LED диодите се врши поспоро, па корисникот може лесно да го следи извршувањето на програмата. VHDL кодот на модулот за скалирање на брзината на такт сигналот е даден на сл. 79.

```
signal prescaler: STD_LOGIC_VECTOR(26 downto 0) := "10111110101111000010000000";
--prescalar = 100.000.000 [prescaler =(clock_speed/desired_clock_speed)/2]
countClock: process(clock_in, newClock)
begin                                                    --генерира 1Hz такт сигнал
       if rising_edge(clock_in) then
            counter <= counter + 1;
            if(counter > prescaler) then
                    newClock <= not newClock;            --промена на ивица и
                    counter <= (others => '0');          --ресетирање на такт сигналот
             end if;
       end if;
end process countClock;
```

Слика 79 VHDL код на модул за скалирање на брзина на такт сигнал.

Конечно процесорот е подготвен за да се изврши програмирање на FPGA уредот, со Xilinx iImpact алатката. Во тој процес се генерира bitstream датотека и се програмира целниот FPGA уред преку JTAG кабел. Креираниот прототип на предложениот MIMOPS процесор во реален хардвер т.е Virtex7 VC709 XC7VX690T FPGA плоча е прикажан на сл. 80.



а) Програмирање на Virtex7 VC709 FPGA плоча со Xilinx iImpact алатка.

б) Симулација на MIMOPS процесор во реален хардвер.

Слика 80 Хардверски прототип на MIMOPS процесор во Virtex7 VC709 FPGA плоча.

7. АНАЛИЗА НА ПРИМЕНЛИВОСТ НА ПРЕДЛОЖЕНИОТ RISC-БАЗИРАН МЕМОРИСКО-ЦЕНТРИЧЕН ПРОЦЕСОР (MIMOPS)

За да може да се анализира применливоста на предложениот MIMOPS процесор изработен е симулатор на инструкциско ниво, кој овозможува да се набљудува состојбата на процесорот и да се прикаже бројот на потребни циклуси за извршување на различни програми. Во првата студија се испитуваат перформансите на MIMOPS процесорот, при извршување на: програми кои се карактеризираат со различен аритметички интензитет, според Roofline моделот, [1], (множење на матрици, пресметка на FFT/IFFT и решавање на парцијална диференцијална равенка). Во втората студија се врши анализа на применливоста на прилагоденото MIMOPS мрежно процесорско јадро при обработка на мрежни IP пакети, со голема пропусност. Овие анализи исто така вклучуваат експериментирање со симулатори за други процесори и употреба на резултати од споредливи експерименти наменети за слични процесорски архитектури.

7.1. Развој на симулатор на инструкциско ниво за предложениот MIMOPS процесор

Симулаторот на инструкциско ниво за MIMOPS процесорот е дизајниран според архитектурата на инструкциско множество (ISA) на MIMOPS процесорот, која беше претставена во поглавје 4.2. Според тоа, MIMOPS процесорот имплементира едноставни три-адресни RISC-базирани меморија-до-меморија инструкции, кои обезбедуваат директен пристап до операндите кои се сместени во податочната меморија на чипот. Всушност, податочните операнди се дефинираат со податочни мемориски директиви: .space, .byte, .half и .word, кои алоцираат и иницијализираат меморија за дадениот податок. Слично на останатите асемблерски јазици, секоја MIMOPS програма на почетокот вклучува .data секција која се употребува за дефинирање на лабели (променливи). На пример, следниот код дефинира три 4-бајтни променливи (a, b и c) со почетни вредности:

```
.data
      a: .word 0d
      b: .word 5d
      c: .word -156d
```

Потоа со овие лабели се задаваат операндите во рамки на инструкциите. Со примерот во продолжение се прикажува множење на вредностите кои се наоѓаат на претходно дефинираните мемориски локации b и c, а резултатот се сместува на мемориска локација a.

```
      mul a, b, c
```

Асемблерот прво доделува последователни мемориски адреси на трите променливи и потоа ги заменува во инструкцијата за множење. Дополнително во кодот може да се работи и директно со мемориски адреси:

```
      mul 0x00000000, 0x00000004, 0x00000008
```

Употребата на директно мемориско адресирање во секоја инструкција не е практично, бидејќи резултира со пораст на програмскиот код заради зголемената должина на инструкциите (употреба на три 32-битни адреси). Како резултат на тоа, MIMOPS процесорот воведува сегментирање на меморијата на чипот во блокови. На тој начин, за секој операнд се дефинира по еден мемориски блок селектор (регистер на страница), кој може да се менува во кое било време од текот на извршувањето на програмата. На сличен начин, со дополнителни три базни регистри се имплементира и базно адресирање, одделно за секој од трите мемориски операнди.

Граматиката на инструкциското множество е дефинирана со користење на библиотеката pyparsing, [107]. Со оглед на тоа дека сите инструкции од групата за аритметичко-логички операции и гранење ја следат истата синтакса, дефинирањето на граматиката е едноставно. На пример, со наредните две линии се дефинираат аритметичко-логичките инструкции:

```
alu3 = oneOf ("add sub and or xor nor mul div mod shl shr") +
                  Optional('[') + ident + Optional(']') + comma +
                  Optional('[') + ident + Optional(']') + comma +
                  Optional('[') + ident + Optional(']')
alui = oneOf ("addi subi andi ori xori nori muli divi modii shli shri") +
                  Optional('[') + ident + Optional(']') + comma +
                  Optional('[') + ident + Optional(']') + comma + const
```

Во првата линија од кодот се специфицираат операциите со три мемориски аргументи. Дополнително, за секој операнд одделно може да се специфицира и типот на адресирање (директно или базно). На сличен начин, во втората линија се дефинираат истите инструкции, меѓутоа последниот операнд се претставува како непосредна вредност (константа).

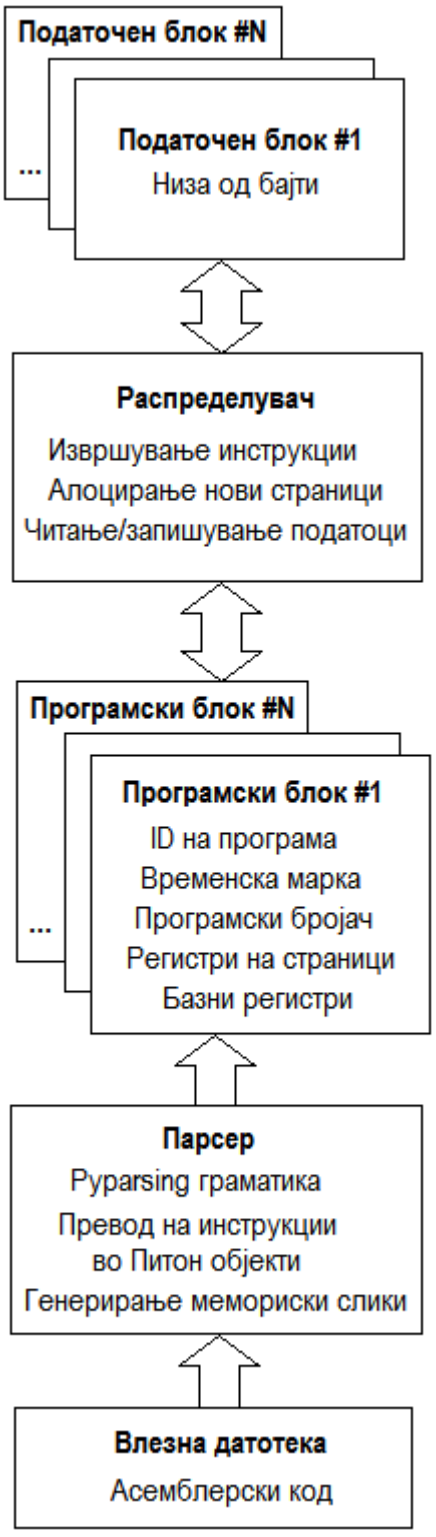
Според зададената граматика, библиотеката *pyparsing* овозможува парсирање на асемблерскиот код, одвојување на составните елементи на инструкциите и нивно пакување во питон листи:

```
"add a, [b], c" => [ "add", "a", "[", "b", "]", "c" ]
```

Асемблерот е имплементиран така што за секоја линија од влезната асемблерска датотека се повикува парсерот.

Структурата и текот на податоци во симулаторот, кој е изработен во Питон, се прикажани на сл. 81. Оттука може да се забележи дека симулаторот е составен од три основни компоненти:

- Парсер: овој дел директно ги превзема објектите кои се излез од pyparsing парсерот и со нив ги иницијализира програмските блокови;
- Програмски блокови: ова се блокови од меморијата кои содржат програмски код. Симулаторот овозможува паралелно извршување на одделни програми во сите програмски блокови. Секој програмски блок има сопствена логика за извршување (програмски бројач, временска марка, како и странични и базни регистри). Доколку е потребно да се започне нов процес, а сите блокови се веќе зафатени, се користи временската марка за да се избере блок чие извршување се прекинува и се заменува со новиот процес. Заради оваа можност, симулаторот е погоден за симулирање и испитување на распределба на процеси во рамките на мемориско-центричната MEMRISC архитектура;
- Распределувач: овој дел извршува по една инструкција на секој активен програмски блок во секој чекор. Доколку инструкцијата работи со мемориски операнди кои не се достапни во податочните блокови, распределувачот ја вчитува новата страница во слободен податочен блок. Доколку сите блокови се зафатени, распределувачот избира жртвен блок според временската марка, и неговата содржина ја заменува со новата страница.



Слика 81 Структура на MIMOPS симулатор на ниво на инструкции

Симулаторот при стартувањето вчитува една или повеќе асемблерски програми и ги изминува нивните кодови во две фази. Во првата фаза се градат следните структури:

- слика на меморијата: листа од мемориски блокови, каде што секој блок е претставен како низа од бајти. Содржината на блоковите се пополнува со иницијалните вредности на променливите, доколку се зададени во кодот. Големината на блоковите може да се поставува при секое повикување на симулаторот за да се испита извршувањето на програмите со различни параметри.
- речник на променливи: овде се чуваат меморискиите адреси на променливите. Речникот содржи парови: име_на_променлива/мемориска_адреса
- речник на лабели: овде се чуваат меморискиите адреси на лабелите во кодот кои се користат во инструкциите за гранење. Речникот содржи парови име_на_лабела/мемориска адреса

Во втората фаза следува извршувањето на инструкциите. При тоа, во секој чекор се прикажува состојбата на сите променливи, како и на сите процесирачки блокови (програмските бројачи, страничите и базните регистри). Дополнително се прикажува бројот на инструкции кои се потребни за да се изврши дадената програма.

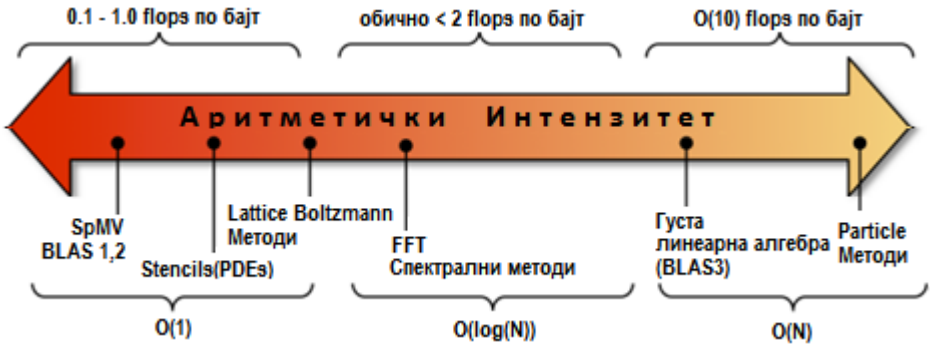
7.2. Анализа на применливост на MIMOPS процесор во апликации со различен аритметички интензитет според Roofline модел

Roofline моделот се користи за да се измерат перформансите на различни нумерички методи и операции кои се извршуваат на повеќе-процесорски, повеќе-јадрени или други процесорски архитектури, [1]. Со оглед на тоа дека во блиска иднина, пропусноста на меморијата надвор од чипот ќе стане ограничувачки ресурс во перформансите на системот, Roofline моделот е така дизајниран да овозможи поврзување на процесорските перформанси со мемориското соодбраќај надвор од чипот. Всушност, овој модел ги обединува перформансите на процесорот, аритметичкиот интензитет и перформансите на меморија.

Клучен параметар во Roofline моделот е аритметичкиот интензитет. Овој параметар е дефиниран во рав. (4) како однос помеѓу вкупната количина на операции (пр. FLOPS) и вкупната количина на податоци (бајти) кои се пренесуваат.

$$\textbf{\textit{Аритметички интензитет}} = \frac{\text{Број на операции}/sec}{\text{Број на операции/бајт}}\ [\text{бајти}/sec] \tag{4}$$

На сл. 82 е прикажан аритметичкиот интензитет за различни типови на алгоритми. Оттука се забележува дека BLAS-1 алгоритамот кој врши инкремент на вектор со друг вектор (x[i]+=y[i]) има низок аритметички интензитет од 0.0417 (N FLOPS/24N бајти), независно од должината на векторот. Спротивно на тоа, FFT метод со N точки извршува 5*N*logN FLOPS операции и пренесува најмалку 48N бајти, постигнувајќи аритметички интензитет од 0.104*logN, кој бавно се зголемува со големината на податоци. Всушност, кај FFT методот аритметичкиот интензитет е ограничен до 2 FLOPS по бајт, од страна на капацитетот на кешот. Конечно, методите на густа линеарна алгебра (dense linear algebra) се карактеризираат со аритметички интензитет кој расте многу брзо.



Слика 82 Аритметички интензитет на различни алгоритми, [1].

Проценката на перформансите која е направена во продолжение го анализира однесувањето на иницијалниот MIPS процесор и на предложениот MIMOPS процесор, при извршување на тест програми кои се карактеризираат со различен аритметички интензитет. Се смета дека предложениот MIMOPS процесор вклучува меморија на чип која е сегментирана во физички блокови со големина од 64KB и која е со вкупен капацитет кој соодветствува на количината на кеш меморија на чип со која работи MIPS процесорот (128KB L1 и 2M L2 кеш). Двата процесора работат проточно, овозможувајќи сите инструкции да се извршуваат за време од еден такт циклус, освен мемориските инструкции кај MIPS процесорот, кои генерираат промашување во L1 или L2 кешот. Се смета дека MIPS процесорот користи кеш со асоцијативно мапирање и работи со блокови со големина од 128 зборови, при што времето на пристап до L1 кешот изнесува 1 такт циклус, времето на пристап до L2 кешот изнесува 21 такт циклуси, а казната која се плаќа за промашување во L2 кешот при пристап до главната меморија изнесува 272 такт циклуси, [52].

Анализата на перформансите на предложениот MIMOPS процесор се врши со помош симулатор на ниво на инструкции, кој е специјално изработен за таа намена (објаснет е во поглавје 7.1). Дополнително, за симулирање на работата на MIPS процесорот и анализирање на промените во кеш меморијата се користи симулаторот MARS, [75]. Оваа студија врши анализа на три различни групи на алгоритми, кои според Roofline моделот се карактеризираат со различен аритметички интензитет (голем, среден и мал). Со споредба на времето на извршување (мерено во такт циклуси) на дадените тест програми за двата различни процесора може да се определи подрачјето на примена, каде предложениот MIMOPS процесор постигнува забрзување во извршувањето, [52].

Првата програма е со најголем аритметички интензитет и врши множење на пополнети матрици со различни димензии (8x8, 16x16, 32x32, 64x64, 128x128, соодветно). Оваа програма извршува голем број на аритметички операции кои се повторуваат, над обемни множества на податоци (768B, 3072B, 12288B, 49152B, 196608B, соодветно). Во продолжение е даден C код кој имплементира множење на пополнети матрици:

```
void multiply(int A[][N], int B[][N], int C[][N])
{
    for (int i = 0; i < N; i++)
    {
        for (int j = 0; j < N; j++)
        {
            C[i][j] = 0;
            for (int k = 0; k < N; k++)
            {
                C[i][j] += A[i][k]*B[k][j];
            }
        }
    }
}
```

Резултатите кои се добиени од симулацијата на програмата за множење на пополнети матрици за MIMOPS и MIPS процесорот се сумаризирани во табела 14 и табела 15, соодветно. Кај MIMOPS процесорот времето на извршување на оваа програма соодветствува на бројот на извршени инструкции, а кај MIPS процесорот времето на извршување се добива како збир од доцнење1 (број на извршени инструкции) и доцнење2 (време кое се троши заради промашувања во кеш).

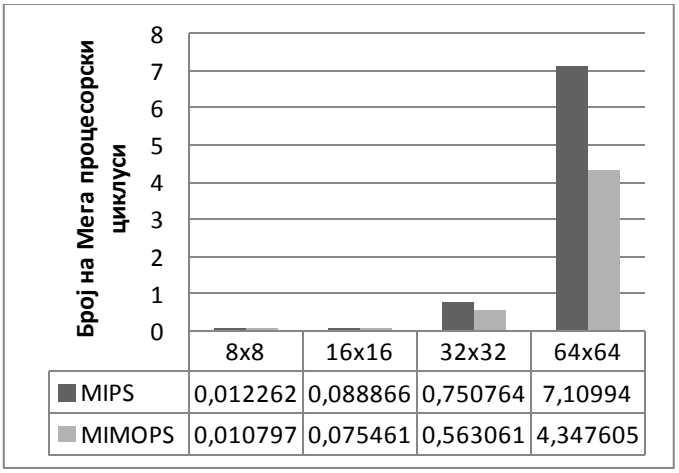
Табела 14 Резултати од симулација на програма за множење на пополнети матрици во MIMOPS процесор.

Димензија на матрици	Големина на под. множество во бајти	Број на инструкции во програма	Големина на програма во бајти	Број на извршени инструкции	Време на извршување
8x8	768	71	568	10797	10797
16x16	3072	71	568	75461	75461
32X32	12288	71	568	563061	563061
64X64	49152	71	568	4347605	4347605
128X128	196608	71	568	32607037	32607037

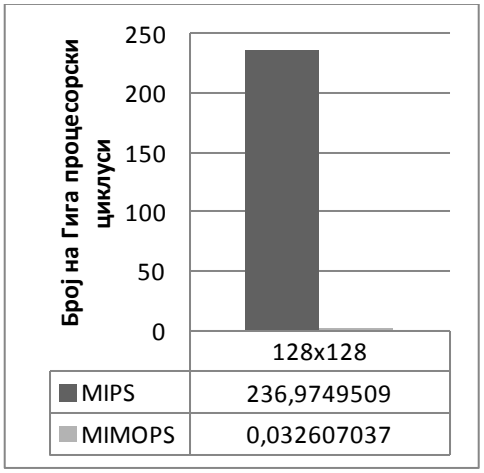
Табела 15 Резултати од симулација на програма за множење на пополнети матрици во MIPS процесор.

Димензија на матрици	Големина на под. множ. во бајти	Број на инстр. во програма	Големина на прогр. во бајти	Број на извршени инструкции (Доцнење1)	Број на промашувања во L1 кеш	Број на промашувања во L2 кеш	Доцнење заради промашувања во кеш (Доцнење2)	Време на извршување
8x8	768	69	276	11131	2	2	1131	12262
16x16	3072	69	276	78947	6	6	9919	88866
32X32	12288	69	276	593587	24	24	157177	750764
64X64	49152	69	276	4601171	96	96	2508769	7109940
128X128	196608	69	276	36227730	2113920	412	2,36939E+11	2,3697E+11

На сл. 83 се прикажани резултатите од компаративната анализа на времето на извршување на множење на пополнети матрици со различни димензии (8x8, 16x16, 32x32, 64x64 и 128x128), со иницијалниот MIPS процесор и предложениот MIMOPS процесор. Оттука може да се забележи дека MIMOPS процесорот го намалува времето на извршување за секоја од дадените димензии на матриците, во споредба со MIPS. Дополнително, резултатите кои се дадени на сл. 84 го прикажуваат процентуално подобрување во времето на извршување на програмите за множење на пополнети матрици (за димензии: 8x8, 16x16, 32x32, 64x64 и 128x128), кое се постигнува со MIMOPS процесорот. Според резултатите се забележува дека со зголемување на големината на проблемот, MIMOPS го забрзува извршувањето на програмите за 12%, 15%, 25%, 39%, 99%, соодветно, споредено со MIPS. Ова се должи на постојано зголемениот број на промашувања во кешот на MIPS процесорот, кои воведуваат дополнително доцнење.

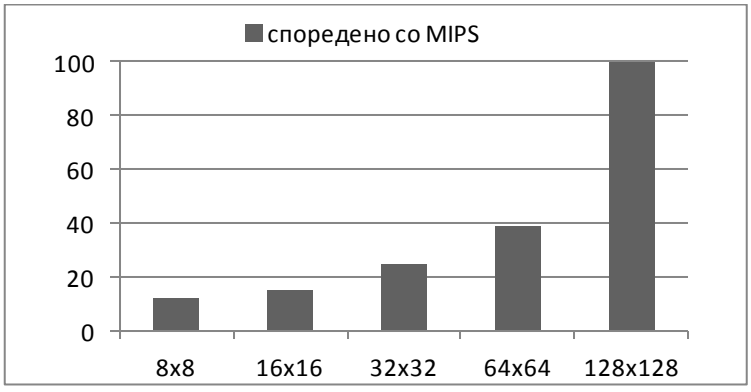


а) Матрици со димензии: 8x8, 16x16, 32x32, 64x64.



б) Матрици со димензии: 128x128.

Слика 83 Анализа на извршување на програма за множење на пополнети матрици со различни димензии во MIPS и MIMOPS процесори.



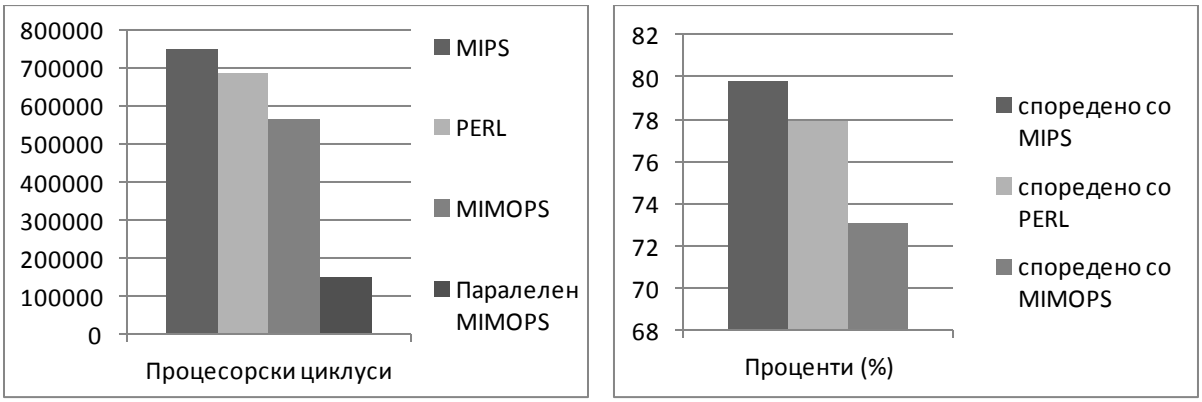
Слика 84 Процентуално подобрување на брзина на извршување на програма за множење на пополнети матрици со различни димензии, за MIMOPS процесор.

Со оглед на тоа дека временската комплексност на претходната програма е $O(N^3)$, во продолжение е објаснет методот "раздели и владеј", кој се користи за паралелизирање на множење на две квадратни матрици со димензии NxN (A и B). Овој метод вклучува два чекора:

1) Подели ги матриците A и B во 4 под-матрици со димензии N/2 x N/2;
2) Пресметај ги следните вредности: ae + bg, af + bh, ce + dg и cf + dh;

$$\underbrace{\begin{bmatrix} a & b \\ c & d \end{bmatrix}}_{A} \times \underbrace{\begin{bmatrix} e & f \\ g & h \end{bmatrix}}_{B} = \underbrace{\begin{bmatrix} ae + bg & af + bh \\ ce + dg & cf + dh \end{bmatrix}}_{C}$$

Овој метод врши 8 множења и 4 собирања на матрици со димензии N/2 x N/2. Собирањето на две матрици трае $O(N^2)$ време, па временската комплексност на методот изнесува $T(N) = 8T(N/2) + O(N^2)$. Овој метод е применет во 4-јадрена паралелна имплементација на MIMOPS процесор, кој извршува множење на две 32x32 пополнети матрици. Во тој случај, секое MIMOPS јадро работи со 16x16 под-матрици, извршувајќи две множења и едно собирање. Се смета дека потребните под-матрици се копирани во соодветните MIMOPS јадра (пр. a е сместена во јадро1 и јадро3). Резултатите од компаративната анализа на програмата за множење на 32x32 пополнети матрици се прикажани на сл. 85, каде на сл. 85 а) е дадено времето на извршување на тест програмата на четири различни процесори, додека пак на сл. 85 б) е дадено подобрувањето кое го постигнува паралелниот MIMOPS процесор. Според резултатите може да се заклучи дека паралелниот MIMOPS процесор овозможува подобрување од 79,8% (4,96 пати), 77,8% (4,52 пати) и 73,1% (3,72 пати) во споредба со MIPS, PERL и MIMOPS, соодветно.



а) Време на извршување. б) Подобрување за паралелен MIMOPS.

Слика 85 Анализа на извршување на програма за множење на 32x32 матрици на четири различни процесори: MIPS, PERL, MIMOPS и паралелен MIMOPS.

Втората програма се карактеризира со среден аритметички интензитет и го претставува подрачјето на спектрални методи. Оваа програма имплементира извршување на IFFT/FFT пресметки, според добро познатиот Cooley–Tukey radix-2 алгоритам. Генерално, инверзната брза фуриева трансформација (IFFT) претставува ефикасен и брз алгоритам за пренесување на сигналите од фреквентен во временски домен, [59]. На сличен начин, брзата фуриева трансформација (FFT) е ефикасен и брз алгоритам за пренесување на сигналите од временски во фреквентен домен. Пресметките кои треба да се извршат за да се добие резултатот од IFFT и FFT, се дадени во равенките (5) и (6), соодветно:

$$x(n) = \frac{1}{N}\sum_{k=0}^{N-1} X(k) W_N^{-nk},\ n = 0,1,\ldots, N-1 \tag{5}$$

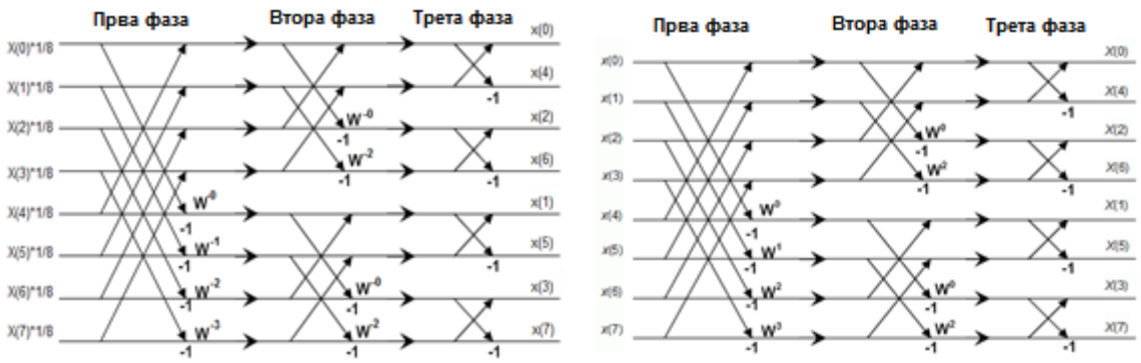
$$X(k) = \sum_{n=0}^{N-1} x(n) W_N^{nk},\ \mathrm{k} = 0,1,\ldots,\mathrm{N}-1 \tag{6}$$

Во равенките (5)/(6), x(n)/X(k) претставува временски/фреквенциски излез, за n/k-тата спектрална точка, каде N претставува број на примероци (точки), а X(k)/x(n) е k/n-тиот примерок земен при семплирањето, соодветно. Влезовите и излезите на IFFT/FFT трансформацијата се комплексни или реални броеви. Вредноста W_N^{-nk}, употребена во равенка (5) се нарекува Twiddle фактор, додека пак вредноста W_N^{nk}, употребена во равенка (6) се нарекува конјугиран Twiddle фактор. Овие вредности се дефинирани во равенките (7) и (8), соодветно.

$$W_N^{-nk} = e^{\frac{j2\pi nk}{N}} \tag{7}$$

$$W_N^{nk} = e^{\frac{-j2\pi nk}{N}} \tag{8}$$

Cooley–Tukey radix-2 алгоритамот користи "раздели па владеј" парадигма на извршување, така што поединечно пресметува IFFT/FFT за влезовите со парен индекс и за влезовите со непарен индекс, па потоа ги комбинира овие резултати за да го добие излезот за целата низа. На овој начин се овозможува рекурзивно извршување на алгоритамот во log2N фази, така што во секоја фаза се одвиваат паралелни пресметки, организирани во "пеперутка" процеси. Секој процес на "пепрутка" вклучува една операција на множење со twiddle фактор, и две операции на собирање на комплексни броеви. На сл. 86 е прикажан граф на извршување на Cooley–Tukey radix-2 алгоритам, за пресметка на IFFT/FFT со 8 точки.



а) Инверзната брза фуриева трансформација. б) Брза фуриева трансформација.

Слика 86 Radix 2 Cooley–Tukey алгоритам за пресметка на IFFT/FFT со 8 точки.

Резултатите кои се добиени од симулацијата на програмата за пресметка на IFFT со различен број на точки (8, 16, 32, 64, 128, 256, 512, 1024, 2048, 4096, 8192 и 16384) за MIMOPS и MIPS процесори се сумаризирани во табела 16 и табела 17, соодветно. Времето на извршување на оваа тест програма се пресметува исто како за програмата за можење на матрици, за двата процесора.

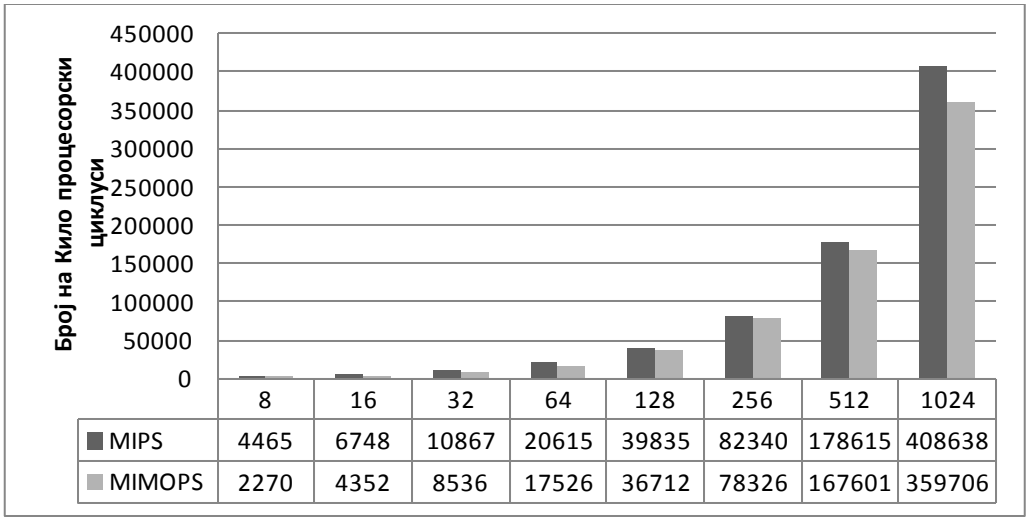
Табела 16 Резултати од симулација на програма за пресметка на IFFT во MIMOPS процесор.

Број на точки	Големина на под. множество во бајти	Број на инструкции во програма	Големина на програма во бајти	Број на извршени инструкции	Време на извршување
8	64	295	2360	2270	2270
16	128	295	2360	4352	4352
32	256	295	2360	8536	8536
64	512	295	2360	17526	17526
128	1024	295	2360	36712	36712
256	2048	295	2360	78326	78326
512	4096	295	2360	167601	167601
1024	8192	295	2360	359706	359706
2048	16384	295	2360	769656	769656
4096	32768	295	2360	1643462	1643462
8192	65536	295	2360	3496244	3496244
16384	131072	295	2360	7416450	7416450

Табела 17 Резултати од симулација на програма за пресметка на IFFT во MIPS процесор.

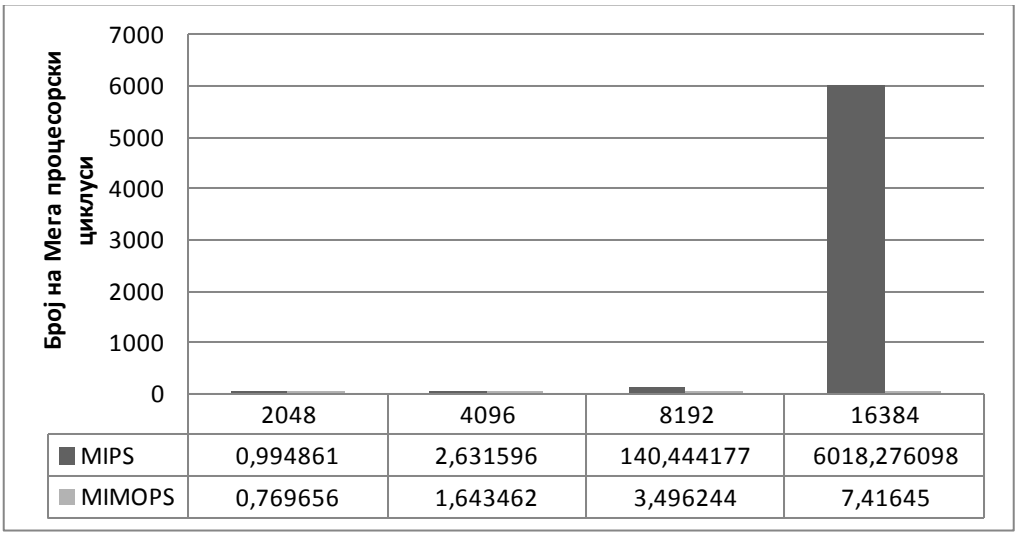
Број на точки	Големина на под. множ. во бајти	Број на инстр. во програма	Големина на прогр. во бајти	Број на извршени инструкции (Доцнење1)	Број на промашувања во L1 кеш	Број на промашувања во L2 кеш	Доцнење заради промашувања во кеш (Доцнење2)	Време на извршување
8	64	270	1080	3334	2	2	1131	4465
16	128	270	1080	5617	2	2	1131	6748
32	256	270	1080	9736	2	2	1131	10867
64	512	270	1080	18103	3	3	2512	20615
128	1024	270	1080	35398	4	4	4437	39835
256	2048	270	1080	72421	6	6	9919	82340
512	4096	270	1080	151204	10	10	27411	178615
1024	8192	270	1080	320131	18	18	88507	408638
2048	16384	270	1080	679714	34	34	315147	994861
4096	32768	270	1080	1445377	66	66	1186219	2631596
8192	65536	270	1080	3066976	3824	132	137377201	140444177
16384	131072	270	1080	6496063	79482	278	6011780035	6018276098

На сл. 87 се прикажани резултатите од компаративната анализа на времето на извршување на IFFT пресметки со различен број на точки (8, 16, 32, 64, 128, 256, 512, 1024, 2048, 4096, 8192 и 16384), за иницијалниот MIPS процесор и предложениот MIMOPS процесор. Оттука може да се забележи дека MIMOPS процесорот го намалува времето на извршување за секоја од извршените IFFT пресметки со различен број на точки, во споредба со MIPS. Дополнително, резултатите кои се дадени на сл. 88 го прикажуваат процентуално подобрување во времето на извршување на програмите за пресметка на IFFT (за 8, 16, 32, 64, 128, 256, 512, 1024, 2048, 4096, 8192 и 16384 точки), добиено со примена на MIMOPS процесорот. Според резултатите се забележува дека постои опаѓање, па пораст на подобрувањето кое се постигнува со MIMOPS процесорот. Ова се должи на тоа што MIMOPS процесорот го зголемува бројот на инструкции кои се извршуваат со зголемување на големината на проблемот за IFFT пресметки, во однос на MIPS. Сепак, како резултат на зголемениот број на промашувања во кешот, кои се јавуваат кај MIPS процесорот при пресметување на IFFT со над 500 точки, MIMOPS процесорот наместо опаѓање, воведува зголемување на забрзувањето.



	8	16	32	64	128	256	512	1024
MIPS	4465	6748	10867	20615	39835	82340	178615	408638
MIMOPS	2270	4352	8536	17526	36712	78326	167601	359706

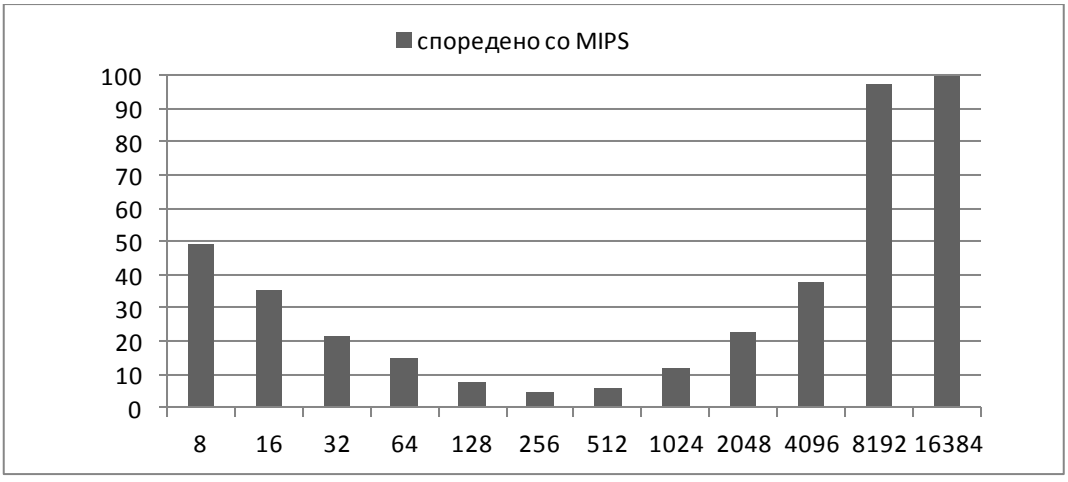
а) IFFT пресметки со 8, 16, 32, 64, 128, 256, 512 и 1024 точки.



	2048	4096	8192	16384
MIPS	0,994861	2,631596	140,444177	6018,276098
MIMOPS	0,769656	1,643462	3,496244	7,41645

б) IFFT пресметки со 2048, 4096, 8192 и 16384 точки.

Слика 87 Анализа на извршување на програма за пресметка на IFFT со различен број на точки во MIPS и MIMOPS процесори.



Слика 88 Процентуално подобрување на брзина на извршување на програма за пресметка на IFFT со различен број на точки, за MIMOPS процесор.

Резултатите кои се добиени од симулацијата на програмата за пресметка на FFT со различен број на точки (8, 16, 32, 64, 128, 256, 512, 1024, 2048, 4096, 8192 и 16384) за MIMOPS и MIPS процесори се сумаризирани во табела 18 и табела 19, соодветно. Времето на извршување на оваа тест програма се пресметува исто како за програмата за пресметка на IFFT, за двата процесора.

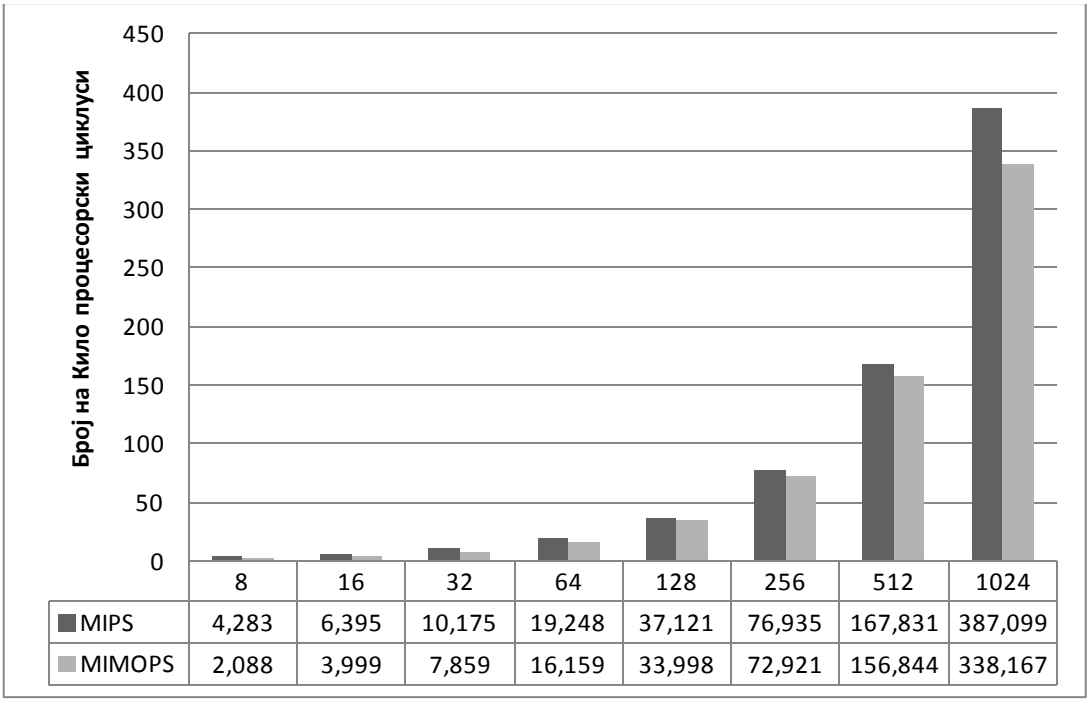
Табела 18 Резултати од симулација на програма за пресметка на FFT во MIMOPS процесор.

Број на точки	Големина на под. множество во бајти	Број на инструкции во програма	Големина на програма во бајти	Број на извршени инструкции	Време на извршување
8	64	295	2360	2088	2088
16	128	295	2360	3999	3999
32	256	295	2360	7859	7859
64	512	295	2360	16159	16159
128	1024	295	2360	33998	33998
256	2048	295	2360	72921	72921
512	4096	295	2360	156844	156844
1024	8192	295	2360	338167	338167
2048	16384	295	2360	726610	726610
4096	32768	295	2360	1557405	1557405
8192	65536	295	2360	3324168	3324168
16384	131072	295	2360	7072339	7072339

Табела 19 Резултати од симулација на програма за пресметка на FFT во MIPS процесор.

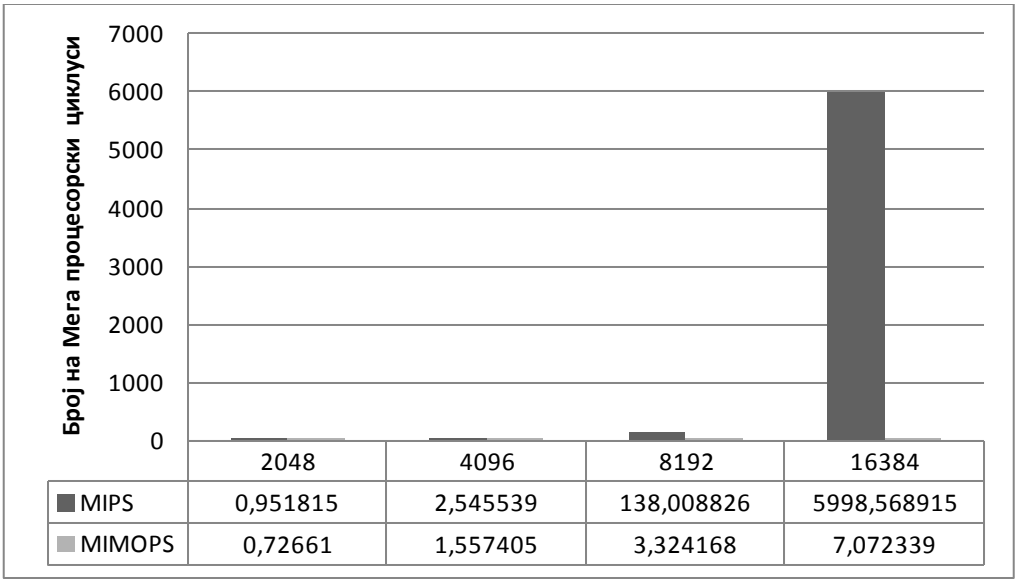
Број на точки	Големина на под. множ. во бајти	Број на инстр. во програма	Големина на прогр. во бајти	Број на извршени инструкции (Доцнење1)	Број на промашувања во L1 кеш	Број на промашувања во L2 кеш	Доцнење заради промашувања во кеш (Доцнење2)	Време на извршување
8	64	270	1080	3152	2	2	1131	4283
16	128	270	1080	5264	2	2	1131	6395
32	256	270	1080	9044	2	2	1131	10175
64	512	270	1080	16736	3	3	2512	19248
128	1024	270	1080	32684	4	4	4437	37121
256	2048	270	1080	67016	6	6	9919	76935
512	4096	270	1080	140420	10	10	27411	167831
1024	8192	270	1080	298592	18	18	88507	387099
2048	16384	270	1080	636668	34	34	315147	951815
4096	32768	270	1080	1359320	66	66	1186219	2545539
8192	65536	270	1080	2894900	3761	132	135113926	138008826
16384	131072	270	1080	6151952	79226	278	5992416963	5998568915

На сл. 89 се прикажани резултатите од компаративната анализа на времето на извршување на FFT пресметки со различен број на точки (8, 16, 32, 64, 128, 256, 512, 1024, 2048, 4096, 8192 и 16384), за иницијалниот MIPS процесор и предложениот MIMOPS процесор. Оттука може да се забележи дека MIMOPS процесорот го намалува времето на извршување за секоја од извршените FFT пресметки со различен број на точки, во споредба со MIPS. Дополнително, резултатите кои се дадени на сл. 90 го прикажуваат процентуално подобрување во времето на извршување на програмите за пресметка на FFT (за 8, 16, 32, 64, 128, 256, 512, 1024, 2048, 4096, 8192 и 16384 точки), добиено со примена на MIMOPS процесорот. Според резултатите се забележува дека постои опаѓање, па пораст на подобрувањето кое се постигнува со MIMOPS процесорот. Ова се должи на тоа што MIMOPS процесорот го зголемува бројот на инструкции кои се извршуваат со зголемување на големината на проблемот за FFT пресметки, во однос на MIPS. Сепак, како резултат на зголемениот број на промашувања во кешот, кои се јавуваат кај MIPS процесорот при пресметување на FFT со над 500 точки, MIMOPS процесорот наместо опаѓање, воведува зголемување на забрзувањето.



	8	16	32	64	128	256	512	1024
MIPS	4,283	6,395	10,175	19,248	37,121	76,935	167,831	387,099
MIMOPS	2,088	3,999	7,859	16,159	33,998	72,921	156,844	338,167

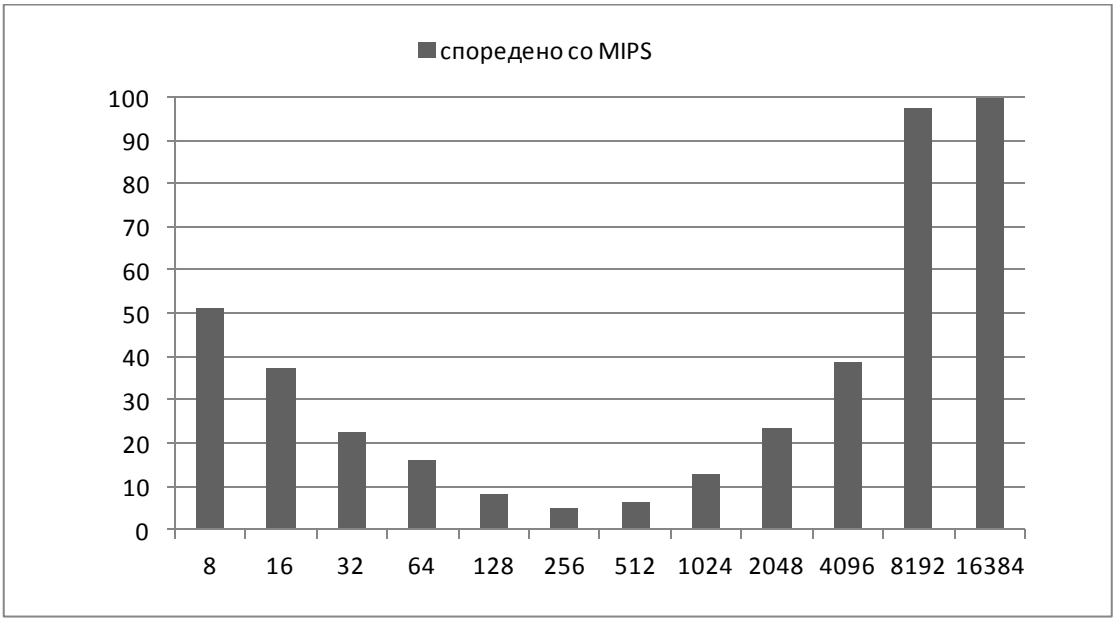
а) FFT пресметки со 8, 16, 32, 64, 128, 256, 512 и 1024 точки.



	2048	4096	8192	16384
MIPS	0,951815	2,545539	138,008826	5998,568915
MIMOPS	0,72661	1,557405	3,324168	7,072339

б) FFT пресметки со 2048, 4096, 8192 и 16384 точки.

Слика 89 Анализа на извршување на програма за пресметка на FFT со различен број на точки во MIPS и MIMOPS процесори.



Слика 90 Процентуално подобрување на брзина на извршување на програма за пресметка на FFT со различен број на точки, за MIMOPS процесор.

За да се паралелизира извршувањето на FFT пресметки со 1024 точки, се користи еден пристап кој овозможува пресметување на FFT со 1024 точки како 2D структура од два FFT модули со 32 точки. Овој пристап го користи правилото, [56], [58], дека секој FFT модул со N точки може да се претстави како декомпозиција од други два FFT модули со p и q точки, така што $N=p*q$, каде $p, q > 1$ (во случајов $1024 = 32*32$).

За да се овозможи пресметка на FFT со 1024 точки, како 2D структура од два FFT модули со 32 точки, се извршува следниот алгоритам во пет чекори:

1) Распределување на векторот со 1024 точки, во 32x32 матрица, во редослед по редици/колони;
2) Пресметка на FFT со 32 точки за секоја колона/редица од матрицата, и сместување на резултатите од пресметката во соодветните колони/ редици;
3) Множење на секој (i,j) елемент од матрицата со twiddle фактор, каде i = [0-31], j = [0-31];
4) Пресметка на FFT со 32 точки за секоја редица/колона од матрицата, и сместување на резултатите од пресметката во соодветните редици/ колони;
5) Резултатот се добива со исчитување на добиената 32x32 матрица, во редослед по колони/редици;

Овој метод е применет во 8-јадрена паралелна имплементација на MIMOPS процесор, кој извршува пресметка на FFT со 1024 точки. Во тој случај, секое MIMOPS јадро извршува FFT пресметки со 32 точки за 4 редици/колони од 32x32 матрица. Се смета дека потребните податоци се дистрибуираат до останатите MIMOPS јадра, после третиот чекор од алгоритамот. Резултатите од компаративната анализа на извршувањето на пресметка на FFT со 1024 точки за различни дигитални процесори на сигнали се дадени во табела 20, каде поголемиот дел од резултатите се превземени од [108]. Дополнително, во анализата се смета дека паралелниот MIMOP процесор работи на фреквенција од 500MHz.

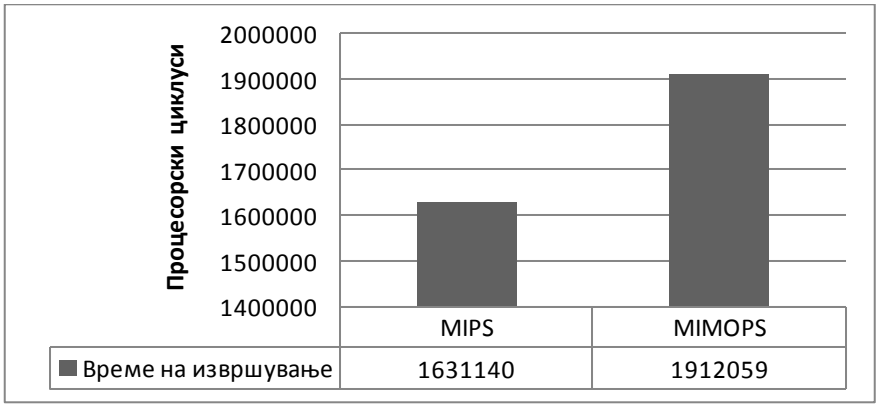
Табела 20 Резултати од симулација на програма за пресметка на FFT со 1024-точки во различни дигитални процесори на сигнали.

Процесор	Брзина на работа во MHz	Големина на FFT	Време на извршување (μs)
Pathfinder-1	100	1024	22.3
Wildstar	n/a	1024	25
Pulsar	80	1024	27.9
VIRAM	200	1024	37
TigerSHARC	250	1024	41
ADSP21160	100	1024	92
TMS3206701	167	1024	124.3
Паралелен MIMOPS	500	1024	129.7

Третата програма е со најмал аритметички интензитет и имплементира решавање на парцијална диференцијална равенка, дадена во продолжение, [109], како претставник на проблемите од подрачјето на структурирани гридови:

$$\begin{cases} \frac{\partial^2}{\partial x^2}u(x,t) = \frac{\partial}{\partial t}u(x,t) \\ u(0,t) = u(1,t) = 0 \\ u(x,0) = \sin \pi x \end{cases}$$

Оваа програма вклучува работа со мало количество на податоци (128B), и при тоа извршува повторливи операции над овие податоци, со повик на две функции. Како резултат на тоа, бројот на промашувања во кеш меморијата на MIPS процесорот е само 2, што значи дека два блока се пренесуваат од главната меморија до дво-нивовската кеш меморија. Резултатите кои се прикажани на сл. 91 го даваат времето на извршување на оваа програма за MIPS и MIMOPS процесори. Според резултатите може да се заклучи дека MIMOPS процесорот го забавува извршувањето на програмата за 14,7%, во споредба со MIPS. Тоа се должи на зголемениот број на инструкции (пр. SETBR), кои ги воведува MIMOPS процесорот, што во иднина би можело да се подобри со компајлерска оптимизација.



Слика 91 Анализа на извршување на програма за пресметка на парцијална диференцијална равенка во MIPS и MIMOPS процесори.

Претходно дадената проценка на перформанси за анализираните програми покажува дека предложениот MIMOPS процесор постигнува значително подобрување при извршување на програми со голем аритметички интензитет, и делумни подобрувања при извршување на програми со среден аритметички интензитет, во споредба со MIPS процесор. Од друга страна гледано, MIMOPS не доведува до подобрувања при извршување на програми со мал аритметички интензитет, па затоа во такви случаи подобро е да се употреби MIPS процесор.

7.3. Анализа на применливост на MIMOPS процесор во мрежно процесирање

Евалуацијата на перформансите на прилагоденото MIMOPS мрежно процесорско јадро е направена преку анализа на брзината на обработка на мрежни пакети за двете верзии на IP протокол: IPv4 и IPv6. Обработката на IPv4 пакети вклучува неколку специфични операции: верификација на определени полиња од IP заглавие (верзија, должина на пакет, изворна и дестинациска IP адреса), валидација на сума за проверка на IP заглавие, пребарување во рутирачка табела за да се избере излезна порта, модификација на некои полиња од IP заглавие (намалување на TTL и повторна пресметка на сума за проверка на IP заглавие) и препраќање на IP пакет, [103]. Обработката на IPv6 пакети, [110], исклучува некои од овие операции како што се на пр. валидација и пресметка на сума за проверка на IP заглавие и воедно ја заменува употребата на IPv4 TTL полето и IPv4 опциите со IPv6 поле за број на хопови и IPv6 заглавија за проширување, соодветно.

На почетокот се врши проценка на перформансните можностите на прилагоденото MIMOPS мрежно процесорско јадро. За таа цел се користат равенките (9) и (10), кои служат за пресметување на теоретскиот максимален број на процесорски циклуси, кој може да се "дозволи“ при обработка на еден пакет за да се обезбедат посакуваните брзини од 10 Gb/s, односно 100 Gb/s, [53].

$$\text{Просечна рата на пакети} = \frac{\text{Податочна рата}}{\text{Просечна големина на пакети}}\ [\text{број на пакети/s}] \tag{9}$$

$$\textbf{\textit{Просечно време за обработка на еден пакет}} = \frac{1}{\text{Просечна рата на пакети}}\ [\text{ns}] \tag{10}$$

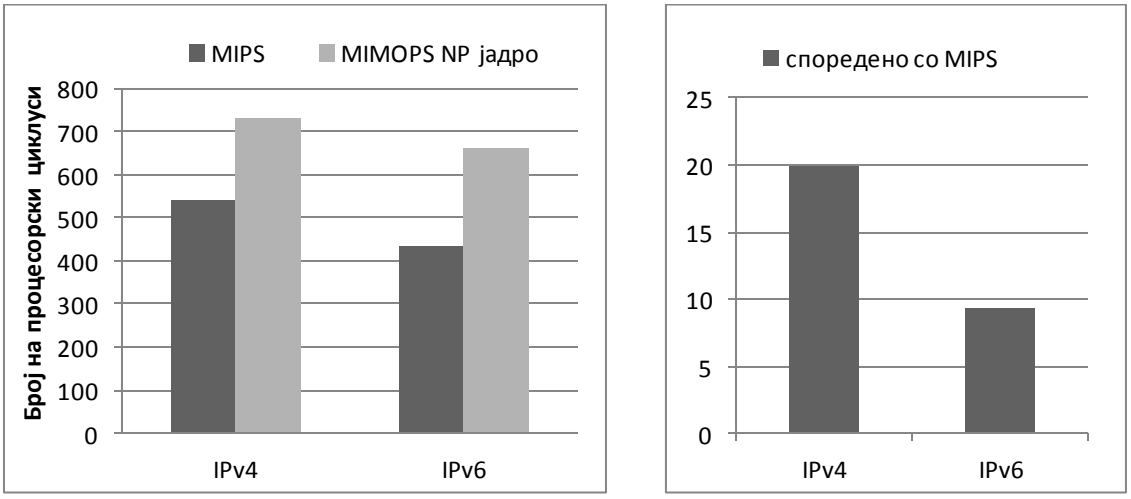
$$= \frac{\text{Просечно време за обработка на еден пакет во ns}}{\text{Траење на еден процесорски циклус во ns}}\ [\text{број на проц. циклуси}]$$

Со пресметување на теоретските граници може да се проценат процесирачките можности на прилагоденото MIMOPS мрежно процесорско јадро. Според тоа, доколку се земе во предвид дека прилагоденото MIMOPS мрежно процесорско јадро работи на фреквенција од 2 GHz, и при тоа обработува пакети со просечна големина од 512B, тогаш теоретскиот број на процесорски циклуси кој

може да се "потроши" за еден пакет, за да се обезбеди 10, односно 100 гигабитна брзина (Gb/s) на обработка изнесува 820, односно 82 процесорски циклуси. За да се задоволат овие барања за високи перформанси, прилагоденото MIMOPS мрежно процесорско јадро имплементира проточност во 4 фази и обезбедува директен пристап до IP заглавијата (кои се сместени во меморијата на чипот) со специјален хардверски модул (парсер на IP заглавија на пакети).

Проценатa на перформансите на прилагоденото MIMOPS мрежно процесорско јадро, е направена со симулации на соодветни асемблерски програми за обработка на IPv4 и IPv6 пакети и споредба на резултатите со оние кои се добиени при симулација на истите програми на стандардно RISC-базирано MIPS процесорско јадро (кое иницијално е употребено во дизајнот на MIMOPS процесорот). За да може да се следи извршувањето на асемблерските програми употребени се два различни симулатори: симулатор на инструкциско ниво за MIMOPS процесор, претставен во поглавје 7.1, и MARS симулатор за MIPS процесор, [75].

На сл. 92 се прикажани резултатите од компаративната анализа на брзината на обработка на IPv4 и IPv6 пакети (во процесорски циклуси) за иницијалниот MIPS процесор и прилагоденото MIMOPS мрежно процесорско јадро. Според сл. 92 а) може да се забележи дека основното MIPS процесорско јадро ја завршува обработката на IPv4/IPv6 пакети за 540/730 процесорски циклуси, додека пак прилагоденото MIMOPS мрежно процесорско јадро ја завршува обработката на IPv4/IPv6 пакети за 432/662 процесорски циклуси, [53].

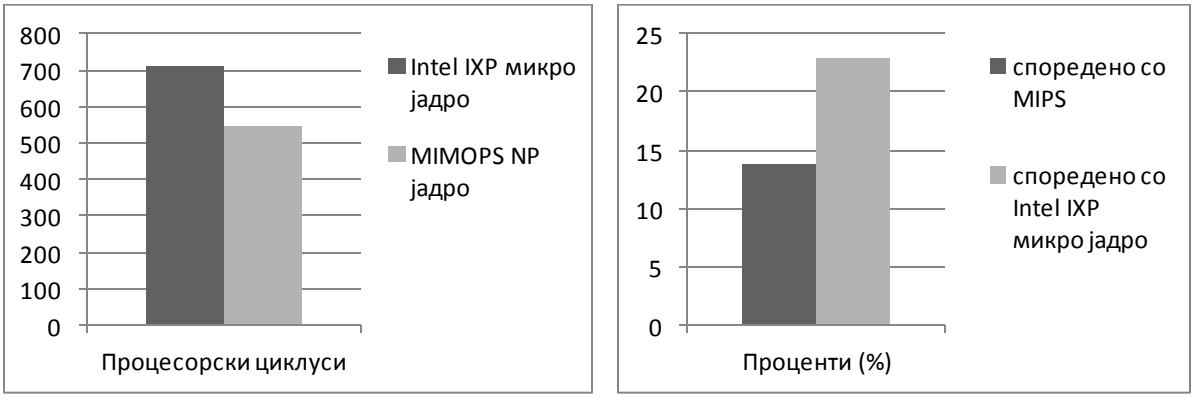


а) Време на извршување. б) Подобрување (%) за MIMOPS NP јадро.

Слика 92 Анализа на брзина на обработка на IPv4 и IPv6 пакети со стандарден RISC-базиран MIPS процесор и прилагодено MIMOPS мрежно процесорско јадро.

Според сл. 92 се забележува дека прилагоденото MIMOPS мрежно процесорско јадро обезбедува намалување на времето на извршување за двете верзии на IP протоколот, во споредба со MIPS процесорско јадро. Всушност, процентуалните подобрувања се прикажани на сл. 92 б), од каде се гледа дека забрзувањето во обработката кое го постигнува прилагоденото MIMOPS мрежно процесорско јадро изнесува 20% за IPv4, односно 9,3% за IPv6 пакети.

Прикажаните резултати, покажуваат дека прилагоденото MIMOPS мрежно процесорско јадро може да постигне повеќе-гигабитна обработка на мрежни пакети во границите од 12,3 - 18,9 Gb/s, или 14.99 Gb/s во просек. Ова укажува дека прилагоденото MIMOPS мрежно процесорско јадро е добра појдовна основа за дизајнирање на хомогена повеќе-јадрена архитектура на мрежен процесор каде секое јадро самостојно и паралелно со другите јадра ќе обработува IP пакети. Овој концепт е сличен со архитектурата на добро познатиот Intel IXP1200 повеќе-гигабитен мрежен процесор, изграден од RISC-базирани микро јадра. Според [60], едно Intel IXP1200 микро јадро може да ја заврши обработката на пакет за 710 процесорски циклуси од кои 280 се наменети за регистарски инструкции, а останатите 430 за мемориски операции. Со оглед на тоа дека просечното време на обработка на еден IP пакет (за двете верзии) од страна на прилагоденото MIMOPS мрежно процесорско јадро изнесува 567 процесорски циклуси, може да се заклучи дека предложеното MIMOPS мрежно јадро постигнува сосема солидни перформанси, кои се дури 1,11 пати (22.9%) подобри од едно Intel IXP 1200 микро јадро, како што е прикажано на сл. 93. Овие резултати се добра основа за дизајнирање на хомогена повеќе-јадрена архитектура за мрежно процесирање.



а) Средно време на извршување. б) Средно подобрување (%) за MIMOPS NP јадро.

Слика 93 Анализа на средна брзина на обработка на мрежни пакети со прилагодено MIMOPS мрежно процесорско јадро и Intel IXP микро јадро.

8. ЗАКЛУЧОК

Ова истражување претставува новина во однос на процесорските архитектури, која досега не е претставена во светската литература од ова подрачје и како таква нуди понатамошен напредок во поглед на развојот на процесорските архитектури. Според тоа, главен резултат од истражување кое е претставено во овој докторски труд е генерирањето на нова RISC-базирана мемориско-центрична (MEMRISC) процесорска архитектура, која интегрира процесор и меморија на заеднички чип, отфрлајќи ги регистрите за општа намена и кеш меморијата (во и надвор од чипот) од стандардната мемориска хиерархија. За разлика од останатите познати чипови со процесирање во меморија, кои најчесто не го напуштаат стандардниот хиерархиски модел на пристап до меморија, предложениот RISC-базиран мемориско-центричен процесор (MIMOPS) обезбедува директен пристап до податоците во меморија во чипот (без да употребува експлицитни LOAD и STORE инструкции), при тоа работејќи во проточен режим со 4 фази (без MEM фаза) и овозможувајќи сите инструкции (аритметички, логички, гранење, контрола) да завршуваат за само еден такт циклус. Основните карактеристики на предложениот MIMOPS процесор се: специфична архитектура на инструкциско множество, која дефинира формати и типови на инструкции, и адресни режими за директна работа со меморијата во чипот; хардверска поддршка за работа со виртуелната меморија која обезбедува управување на мемориските блокови и табелите за преведување во чипот; проточна податочна патека со четири фази (земање, декодирање, извршување и запишување на резултат), која е изградена од повеќе компоненти (поделена инструкциска и податочна меморија во чип, посебни генератори на мемориски адреси за инструкцискатa и податочната меморија во чип, аритметичко-логичка единица која поддржува операции со цели и реални броеви, единица за поместување, проточни регистри за секоја меѓу-фаза, проширувачи, собирачи и дополнителна логика за селекција и контрола); контролна единица која обезбедува поддршка за проточно извршување на инструкции; и механизми за разрешување на структурни, податочни и контролни конфликти и справување со исклучоци. Како резултат на овие карактеристики, проектираниот MIMOPS процесор овозможува: брз и едноставен при-

стап до меморијата во чипот (без експлицитни load/store операции и MEM фаза во проточна линија), голема внатрешна мемориска пропусност, намалување на капацитетот на редундантните мемориски ресурси (регистри за општа намена и кеш меморијата) во чипот, избегнување на копирање и трансфер на редундантни податоци и блокови во регистрите за општа намена и кеш меморијата, и отстранување на комплексните механизми за управување со кеш меморијата.

Во докторскиот труд е предложена надградба на MIMOPS процесорот со специфична хардверска логика за парсирање на IP заглавија, која овозможува MIMOPS процесорот да се прилагоди за мрежно процесирање на IP пакети. Предложениот парсер на заглавија на IP пакети се додава до локалната меморија во чипот (која содржи повеќе заглавија на IP пакети) на MIMOPS процесорот и на тој начин овозможува екстрахирање на полињата од IP заглавието директно на мемориските излез. Всушност, парсерот на заглавија на IP пакети користи шема на адресирање која имплементира техника на мемориски прекари за да обезбеди директен пристап до било кое поле од IP заглавието (со различни должини). Овој пристап му овозможува на MIMOPS процесорот да избегне извршување на логички или аритметички операции, при издвојување на вредноста на полињата од IP заглавието, кои не се со должина на збор. Како резултат на тоа, MIMOPS процесор кој користи парсер за заглавија на IP пакети постигнува подобрување од 95,6%/93,7% во споредба со основен MIMOPS процесор, при парсирање на IPv4/IPv6 заглавија. Генерално, може да се заклучи дека предложениот парсер на заглавија на IP пакети е едноставен за дизајнирање, може лесно да се надгради за други формати на заглавија на пакети и истиот воведува забрзување во парсирањето на IP заглавијата на пакети кога е употребен во различни процесорски архитектури (пр. MIPS, MIMOPS итн.).

Еден од главните придонеси на докторскиот труд е креирањето на хардверски прототип на предложениот MIMOPS процесор во Virtex7 VC709 FPGA компонента, со Xilinx VIVADO Design Suite софтверската околина. Ова вклучува изработка на хардверски модел на предложениот MIMOPS процесор во VHDL јазик за опис на хардвер и симулирање на неговата работа со Xilinx VIVADO Design Suite Simulator алатката. За таа цел изработени се повеќе различни тест програми со кои е анализирана работата на поединечните компоненти и целиот проце-

сор, и на тој начин е потврдено дека MIMOPS процесорот ја постигнува потребната функционалност. Потоа, со алатките за синтеза и имплементација е генериран RTL модел на предложениот процесор и е направена имплементација на синтетизираниот процесор во Virtex7 VC709 FPGA плочка. Извештаите кои се генерираат од овие алатки, покажуваат дека фазите за земање и декодирање од проточната линија на MIMOPS процесорот се покомплексни во споредба со оние на стандарден RISC-базиран процесор, бидејќи меморијата и единицата за управување со меморијата (генератори на мемориски адреси со табели за преведување и регистри за страници) на MIMOPS процесорот се сместени внатре во чипот. Во последната фаза од FPGA имплементацијата на MIMOPS процесорот, направено е мапирање на В/И интерфејси на MIMOPS процесорот со пиновите од Virtex7 VC709 FPGA плочката и потоа FPGA плочката е програмирана да ја симулира работата на предложениот MIMOPS процесор. На овој начин е направена практична имплементација на MIMOPS процесорот во реален хардвер (FPGA).

Покрај хардверски прототип, во докторскиот труд е изработен софтверски прототип за предложениот MIMOPS процесор, односно специфичен симулатор на инструкциско ниво. Овој симулатор е употребен за испитување на перформансите на MIMOPS процесорот, при извршување на: програми кои се карактеризираат со различен аритметички интензитет според Roofline моделот (множење на матрици, пресметка на FFT/IFFT и решавање на парцијални диференцијални равенки) и програми за мрежно процесирање (на IPv4 и IPv6 пакети). Со анализа на времето на извршување на овие програми и нивна споредба со резултатите кои ги постигнуваат слични процесорски архитектури (пр. MIPS, PERL, интелигентен RAM) се определува типот на апликации за кои предложениот MIMOPS процесор воведува подобрување на перформансите. Според првата анализа која е направена, се покажува дека MIMOPS процесорот е погоден за извршување на програми кои се карактеризираат со висок и/или среден аритметички интензитет, пред сè поради тоа што тој обезбедува директен пристап до податоците, избегнувајќи ги временските застои кои се јавуваат при нивна размена помеѓу регистрите и кеш меморијата во стандардната мемориска хиерархија. Конкретно, MIMOPS процесорот постигнува 1,33 пати (25%) подобри резултати од MIPS, и 1,21 пати (17,7%) подобри резултати од PERL, при извршување на програма за множење на

32x32 матрици, која се карактеризира со висок аритметички интензитет. Дополнително, MIMOPS процесорот постигнува 12,6/11,9% подобри резултати од MIPS, при извршување на програма за пресметување на FFT/IFFT со 1024 точки, која се карактеризира со среден аритметички интензитет. Уште повеќе, паралелната имплементација на MIMOPS процесорот со 8 јадра постигнува слични резултати како интелигентниот RAM и други дигитални процесори на сигнали, при извршување на FFT пресметки со 1024 точки. Кога станува збор за мрежно процесирање, втората анализа покажува дека прилагоденото MIMOPS мрежно процесорско јадро, кое обезбедува директна манипулација со полиња од IP заглавие, овозможува повеќе-гигабитна обработка на мрежни пакети во граници од 12 - 19 Gb/s. Всушност, ова мрежно процесорско јадро го забрзува процесирањето на Ipv4/IPv6 пакети за 20/9,3%, во споредба со MIPS процесор за општа намена, и истовремено обезбедува слични перформанси како други познати мрежни процесирачки јадра (пр. микро единици во Intel IXP мрежен процесор).

8.1. Научни придонеси на докторскиот труд

Главните научни придонеси, кои произлегуваат од истражувањето, кое е претставено во овој докторски труд се:

- Теоретска анализа на техниките и методите за надминување и избегнување на појава на тесно грло при комуникација помеѓу процесор и меморија.
- Компаративна анализа на различни мемориско-центрични системи (чипови), нивни предности, недостатоци и подрачје на примена.
- Развој на RISC-базирана мемориско-центрична (MEMRISC) процесорска архитектура, која обезбедува директна комуникација помеѓу процесорот и меморијата, без употреба на регистри за општа намена и кеш меморија. Овој предлог вклучува:
  - хардверска поддршка за работа со виртуелната меморија (управување со мемориски блокови и табели за преведување во чипот, развој на специфични единици за генерирање на мемориски адреси кои вклу-

чуваат инвертирани табели на страници и регистри за виртуелни страници, и предлог за дизајнирање на систем со дистрибуирана споделена виртуелна меморија);

- развој на специфично инструкциско множество со адресни режими, кои обезбедуваат директен пристап до меморијата во чипот;
- исфрлање на експлицитни LOAD/STORE инструкции:
- развој на проточна податочна патека, која овозможува извршување на сите инструкции (аритметички, логички, гранења и контрола) во еден такт циклус;
- намалување на бројот на фази при проточното работење на 4 (земање, декодирање, извршување и запишување на резултат), со отстранување на експлицитната MEM фаза од стандардната RISC проточна линија со 5 фази;
- развој на аритметичко-логичка единица која може да работи со цели и реални броеви;
- развој на специфична контролна единица која обезбедува поддршка за проточно извршување на инструкциите;
- развој на единици за препраќање и единица за детекција на конфликти, наменети за справување со податочните меѓу-зависности, кои се јавуваат при проточно извршување на инструкциите;
- употреба на механизми за разрешување на конфликти и справување со исклучоци и прекини;

- Развој на мемориско-центричен пристап на мрежно процесирање со додавање на специфична хардверска логика (парсер на IP заглавија) до меморијата во чипот на процесор со MEMRISC архитектура со цел да обезбеди директен пристап до податоци кои не се со ширина на збор (полиња од IP заглавија на мрежни пакети).
- Анализа на забрзување на парсирање на IP заглавија на пакети, кога предложениот парсер на IP заглавија се применува во процесори со RISC (MIPS) и MEMRISC архитектура, и споредба на добиените резултати.

- Развој на VHDL модел на MIMOPS процесор со MEMMRISC архитектура и имплементација на предложениот процесор во реален хардвер, со употреба на Xilinx Virtex7 VC 709 FPGA развојна плочка.
- Анализа на карактеристиките на хардверската реализација на предложениот MIMOPS процесор, синтетизиран во Virtex7 VC 709 FPGA плочка.
- Развој на симулатор на инструкциско ниво за предложениот MIMOPS процесот, кој овозможува преглед на состојбата на процесорот при извршување на програми. Дел од параметрите кои можат да се анализираат се: големина на податочно множество, големина на програма, број на алоцирани виртуелни страни, извршени процесорски циклуси итн.
- Анализа на применливоста на предложениот MIMOPS процесор во апликации кои се карактеризираат со различен аритметички интензитет (според Roofline моделот), и споредба на добиените резултати со оние за слични процесорски архитектури.
- Анализа на перформансите на прилагоденото MIMOPS мрежно процесирачко јадро (кое вклучува парсер на заглавија на IP пакети) при обработка на IPv4/IPv6 мрежни пакети и споредба на добиените резултатите со оние за други слични процесирачки јадра кои се користат при мрежно процесирање на пакети.

8.2. Идна работа

Последните трендови во дизајнот на процесорите покажуваат дека голем дел од површината на модерните микропроцесорски чипови е наменета за имплементирање на врвот од мемориската хиерархија, кој вклучува: регистри за општа намена и едно или две нивоа на кеш меморија. Сметајќи дека овој тренд продолжува да расте и дека јазот во мемориското доцнење помеѓу меморијата во и надвор од чипот се зголемува, се јавува потреба за интегрирање на целата физичка меморија во чипот, при што обраќањата надвор од чипот би се третирале повеќе како промашувања во виртуелни страници, отколку како промашувања во кеш меморијата. Како резултат на тоа, во овој докторски труд е пред-

ложена нова процесорска архитектура - MEMRISC, која е подготвена да одговори на овие барања и со тоа да креира нова генерација на процесорски чипови. Се очекува дека ваквите чипови ќе се појават во многу блиска иднина, и воедно ќе станат стандард во индустријата за дизајнирање на процесори.

Предложениот MIMOPS процесор обезбедува многу предности, особено во поглед на брзината на обработка, но од друга страна тој наметнува дополнителни барања за системскиот хардвер и софтвер, кои предизвикуваат ограничување во неговото подрачје на примена. Како резултат на тоа предложениот MIMOPS процесор имплементира специфична архитектура на инструкциско множество и неколку хардверски компоненти со специфична намена кои обезбедуваат директна работа со меморијата во чипот. Оттука, очигледно е дека треба да се развие соодветна софтверска поддршка за предложениот MIMOPS процесор. Првично, би требало да се развие специфичен компајлер, кој ќе може да преведува програми напишани во некој јазик на високо ниво во MIMOPS асемблер, (кој значително се разликува од асемблерот на останатите RISC-базирани процесори), при тоа задржувајќи го стандардниот модел на програмирање. Понатаму идните истражувања би требало да вклучуваат и развој на специфичен оперативен систем со распределувач на процеси, кој ќе управува со меморијата на чипот на MIMOPS процесорот и воедно ќе го координира целиот виртуелен адресен простор, при извршување на повеќе процеси.

Едно од главните ограничувања на предложениот MIMOPS процесор е количината на меморија во чипот, која најмногу зависи од употребената технологија за имплементација. Овој параметар има големо влијание врз комплексноста на чипот, неговата цена, примена и прифатливост на пазарот. Ако се земат во предвид последните иновации во архитектурата и технологијата на процесирање во меморија (пр. вграден DRAM, паметна мемориска коцка), тогаш во блиска иднина може да се размислува за дизајнирање на скалабилен повеќепроцесорски MIMOPS-базиран систем. Овој тип на систем би имплементирал дистрибуирана споделена виртуелна меморија, избегнувајќи го проблемот со ограничената големина на меморија во процесорски чип. И покрај тоа што ваквиот систем ќе биде скалабилен, флексибилен и со пошироко подрачје на примена, сепак тој ќе наметне некои дополнителни барања, како што се: нов (пара-

лелен) модел на програмирање, техники за синхронизација и комуникација помеѓу процесорите, механизми за меѓу-поврзување, методи за одржување на конзистентноста на споделената меморија, ефикасна искористеност на мемориските ресурси итн. Иницијални идеи за развој на ваков систем се дадени во [51].

Овој докторски труд покажува дека MIMOPS процесорот, таков каков што е дизајниран (со ограничена количина на меморија во чип), би можел да се примени во определено множество на апликации, каде брзината на обработка и податочната пропусност се од голема важност. Како резултат на тоа, во иднина би можело да се размислува за правење на дополнителни промени на предложената MEMRISC архитектура со кои таа ќе стане повеќе погодна за примена во некои специфични апликации, како што тоа беше направено за мрежното процесирање. На пример, MIMOPS процесорот би можел да се прилагоди за обработка на други формати на пакети (не само IP), или пак за извршување на некои покомплексни операции со мрежни пакети, како што е филтрирањето на пакети (почетни истражувања се прикажани во [111]). Покрај тоа, MIMOPS процесорот би можел дополнително да се прилагоди за извршување на некои матрични или математички операции, наменети за обработка на дигитални сигнали. Соодветно на тоа, може да се каже дека сè уште има голем простор за понатамошни истражувања и експериментирања со она што е започнато во овој докторски труд.

9. КОРИСТЕНА ЛИТЕРАТУРА

[1] David A. Patterson, John L. Hennessy, *Computer Organization and Design: The hardware/software interface*, 5th ed., Elsevier, 2014.

[2] John L. Hennessy, David A. Patterson, *Computer Architecture: A Quantitative Approach*, 5th ed., Morgan Kaufmann Publishers, 2011.

[3] "Moore's law is dead - long live Moore's law", *in IEEE Spectrum Magazine*, April 2015.

[4] Joel Hruska, "Forget Moore's law: hot and slow DRAM is a major roadblock to exascale and beyond", *in Extreme Tech Magazine*, 2014.

[5] Win. A. Wulf, Sally A. McKee, "Hitting the memory wall: implications of the obvious*", in ACM SIGARCH Computer Architecture News*, Vol. 23, Issue 1, March 1995.

[6] Carlos Carvalho, "The gap between processor and memory speeds", *in Proc. of 3rd Internal Conference on Computer Architecture*, Braga, Portugal, 2002.

[7] David Patterson, "Latency lags bandwidth", *in Communications of the ACM*, Vol. 47, No. 10, pp 71-75, 2004.

[8] Rudolf Eigenmann, David J. Lilja, "Von Neumann computers", *in Wiley Encyclopedia of Electrical and Electronics Engineering*, Vol. 23, pp. 387-400, 1998.

[9] Shekhar Borkar, Andrew A. Chien, "The future of microprocessors", *in Communications of the ACM,* Vol. 54, No. 5, pp 67-77, May 2011.

[10] Intel Corporation, "New microarchitecture for 4th gen. Intel core processor platforms", *Product Brief*, 2013.

[11] Samira Khan, Daniel A. Jimenez, Doug Burger, Babak Falsafi, "Using dead blocks as a virtual victim cache", *in Proc. of the 19th international conference on Parallel architectures and compilation techniques*, Austria, 2010.

[12] Lakshmi Deepika Bobbala, Monobrata Debnath and Byeong Kil Lee, "Composite pseudo-associative cache for mobile processors", *Journal of Computer Science*, Vol. 7, No. 10, 2011.

[13] M. A. Z. Alves, K. Khubaib, E. Ebrahimi, V. T. Narasiman, C. Villavieja, P. O. A. Navaux, Y. N. Patt, "Energy savings via dead sub-block prediction", *in Proc. of 24th International Symposium on Computer Architecture and High Performance Computing*, USA, 2012.

[14] Sheng Li, Ke Chen, Jay B. Brockman, Norman P. Jouppi, "Performance impacts of non-blocking caches in out-of-order processors", *Technical Paper*, 2011.

[15] Wenlei Bao, Sanket Tavarageri, Fusun Ozguner, P. Sadayappan, "PWCET: power-aware worst case execution time analysis", *in Proc. of 43rd International Conference on Parallel Processing Workshops*, 2014.

[16] D. Jakimovska, A. Tentov, G.Jakimovski, S. Gjorgjievska, M. Malenko, "Modern processor architectures overview", *in Proc. of XVIII International Scientific Conference on Information, Communication and Energy Systems and Technologies*, Bulgaria, pp. 239-242, 2012.

[17] Philip Machanick, "Approaches to addressing the memory wall", *Technical Report*, School of IT and Electrical Engineering, University of Queensland Brisbane, Australia, 2002.

[18] A. Bakshi, J. Gaudiot, W. Lin, M. Makhija, V. K. Prasanna, W. Ro, C. Shin, "Memory latency: to tolerate or to reduce?", *in Proc. of 12th Symposium on Computer Architecture and High Performance Computing*, 2000.

[19] Christoforos Kozyrakis, David Patterson, "Vector vs. superscalar and VLIW architectures for embedded multimedia benchmarks", *In proc. of the 35th International Symposium on Microarchitecture*, Instabul, Turkey, November 2002.

[20] J. Silc, B. Robic, T. Ungerer, *Processor architecture: From Dataflow to Superscalar and Beyond*, Springer, 1999.

[21] Nicholas FitzRoy-Dale, "The VLIW and EPIC processor architectures", *Master Thesis*, New South Wales University, July 2005.

[22] M. Smotherman, "Understanding EPIC architectures and implementations", *in Proc. of ACM Southeast Conference*, 2002.

[23] Danijela Jakimovska, Goran Jakimovski, Aristotel Tentov, Dimitar Bojchev, "Performance estimation of parallel processing techniques on various platforms", *in Proc. of 20th IEEE Telecommunications forum - TELFOR*, Serbia, pp. 1409 - 1412, November 2012.

[24] Young Hoon Son, O. Seongil, Yuhwan Ro, Jae W. Lee, Jung Ho Ahn, "Reducing memory access latency with asymmetric DRAM bank organizations", *in ACM SIGARCH Computer Architecture News*, Vol. 41, Issue 3, 2013.

[25] Bruce L. Jacob, "Synchronous DRAM architectures, organizations, and alternative technologies", *Technical Paper*, 2002.

[26] Yoongu Kim, Vivek Seshadri, Donghyuk Lee, Jamie Liu, Onur Mutlu, "A case for exploiting subarray-level parallelism (SALP) in DRAM", *in Proc. of 39th Annual International Symposium on Computer Architecture,* 2012.

[27] Donghyuk Lee, Yoongu Kim, Vivek Seshadri, Jamie Liu, Lavanya Subramanian, Onur Mutlu, "Tiered-latency DRAM: a low latency and low cost DRAM architecture", *in Proc. of 19th IEEE International Symposium on High Performance Computer Architecture*, 2013.

[28] Erfan Azarkhish, Davide Rossi, Igor Loi, Luca Benini, "Design and evaluation of a processing-in-memory architecture for the smart memory cube", *in Proc. of the 29th International Conference Architecture of Computing Systems*, Germany, 2016.

[29] Fatih Hamzaoglu, Umut Arslan, Nabhendra Bisnik, Swaroop Ghosh, Manoj B. Lal, Nick Lindert, Mesut Meterelliyoz, Randy B. Osborne, Joodong Park, Shigeki Tomishima, Yih Wang, Kevin Zhang, "A 1GB 2GHz embedded DRAM in 22nm tri-gate CMOS technology", *in Proc. of IEEE International Solid-State Circuits Conference*, 2014.

[30] John Barth, Don Plass, Erik Nelson, Charlie Hwang, Gregory Fredeman, Michael Sperling, Abraham Mathews, Toshiaki Kirihata, William R. Reohr, Kavita Nair, Nianzheng Cao, "A 45 nm SOI embedded DRAM macro for the POWER™ processor 32 MByte on-Chip L3 cache", *IEEE Journal of Solid-state Circuits*, Vol. 46, No. 1, Jan. 2011.

[31] Wing-kei S. Yu, Ruirui Huang, Sarah Q. Xu, Sung-En Wang, Edwin Kan, G. Edward Suh, "SRAM-DRAM hybrid memory with applications to efficient register files in fine-grained multi-threading", *in Proc. of 38th IEEE International Symposium on Computer Architecture*, 2011.

[32] Danijela Efnusheva, Ana Cholakoska, Aristotel Tentov, "A survey of different approaches for overcoming the processor-memory bottleneck", *International Journal of Computer Science & Information Technology*, Vol. 9, No. 2, April 2017.

[33] P. Suresh, "PERL - A register-less processor", *PhD Thesis*, Department of Computer Science & Engineering, Indian Institute of Technology, Kanpur, 2004.

[34] P. R. Panda, "On-chip vs. off-chip memory: the data partitioning problem in embedded processor-based systems", *ACM Trans. on Design Automation of Electronic Systems*, 2000.

[35] Peng Wang, "Designing scratchpad memory architecture with emerging STT-RAM memory technologies", *in Proc. of IEEE International Symposium on Circuits and Systems*, 2013.

[36] C. Cojocaru, "Computational RAM: implementation and bit-parallel architecture", *Master Thesis*, Carleton University, Ottawa, 1995.

[37] Ashley Saulsbury, Fong Pong, Andreas Nowatzyk, "Missing the memory wall: the case for processor/memory integration", *in Proc. of the 23rd Annual International Symposium on Computer Architecture,* USA, 1996.

[38] Hideo Tsubota and Toshifumi Kobayashi, "The M32R/D, a 32b RISC microprocessor with 16Mb embedded DRAM", *Technical Report*, 1996.

[39] Maya Gokhale, Bill Holmes, Ken Jobst, "Processing in memory: the Terasys massively parallel PIM array", *IEEE Computer Journal*, 1995.

[40] Jeffrey Draper, J. Tim Barrett, Jeff Sondeen, Sumit Mediratta, Chang Woo Kang, Ihn Kim, Gokhan Daglikoca, "A prototype processing-in-memory (PIM) chip for the data-Intensive architecture (DIVA) system", *Journal of VLSI Signal Processing Systems*, Vol. 40, Issue 1, 2005.

[41] Jeff Draper, Jacqueline Chame, Mary Hall, et al, "The architecture of the DIVA processing in memory chip", *in Proc. of the 16th International Conference on Supercomputing*, 2002.

[42] Joseph Gebis, Sam Williams, David Patterson, Christos Kozyrakis, "VIRAM1: a media-oriented vector processor with embedded DRAM", *41st Design Automation Student Design Contest*, San Diego, CA, USA, June 2004.

[43] K. Keeton, R. Arpaci-Dusseau, and D.A. Patterson, "IRAM and SmartSIMM: overcoming the I/O bus bottleneck", *in Proc. of the 24th Annual International Symposium on Computer Architecture*, June 1997.

[44] K. Murakami, S. Shirakawa, H. Miyajima, "Parallel processing RAM chip with 256 Mb DRAM and quad processors", *in Proc. of Solid-State Circuits Conference*, 1997.

[45] Doug Burger, Stefanos Kaxiras, and James R. Goodman, "DataScalar architectures and the SPSD execution model", *Technical Report*, University of Wisconsin, 1996.

[46] D. Keen, M. Oskin, J. Hensley, F.T. Chong, "Cache coherence in intelligent memory systems", *IEEE Transactions on Computers*, Vol. 52, Issue 7, 2003.

[47] Intel, "Stratix 10MX FPGA: 512 GBps memory bandwidth in a single FPGA", *White Paper*, 2017.

[48] Danijela Efnusheva, Goce Dokoski, Aristotel Tentov, Marija Kalendar, "A novel memory-centric architecture and organization of processors and computers", *in Proc. of Third International Conference on Applied Innovations in IT*, Koethen, Germany, March 19, 2015.

[49] Goce Dokoski, Danijela Efnusheva, Aristotel Tentov, Marija Kalendar, "Software for explicitly parallel memory-centric processor architecture", *in Proc. of Third International Conference on Applied Innovations in IT*, Koethen, Germany, March 19, 2015.

[50] Danijela Efnusheva, Goce Dokoski, Aristotel Tentov, Marija Kalendar, "Проектирање на нова мемориско-центрична архитектура и организација на процесори", *in. Proc. of XII International Conference ETAI,* Ohrid, Macedonia, September 24-26, pp. EI-4, 2015.

[51] Goce Dokoski, Danijela Efnusheva, Aristotel Tentov, Marija Kalendar, "Софтверска околина за мемориско-центрична процесорска архитектура", *in Proc. of XII International Conference ETAI*, Ohrid, Macedonia, September 24-26, pp. EI-4, 2015.

[52] Danijela Efnusheva, Aristotel Tentov, "Design of processor in memory with RISC-modified memory-centric architecture", *Advances in Intelligent Systems and Computing*, Springer (ISSN 2194-5357), April, 2017.

[53] Danijela Efnusheva, Goce Dokoski, Aristotel Tentov, Marija Kalendar, "Memory-centric approach of network processing in a modified RISC-based processing core", *in Proc. of IEEE Future Technologies Conference*, San Francisco, CA, USA, December, 2016.

[54] Danijela Efnusheva, Aristotel Tentov, "Integrating processing in RAM memory and its application to high speed FFT computation", *in Proc. of 4th International Conference on Information Society and Technology - ICIST*, Kopaonik, Serbia, pp. 382-387, March 2014.

[55] Danijela Efnusheva, Natasha Tagasovska, Aristotel Tentov, Marija Kalendar, "Efficiency comparison of DFT/IDFT algorithms by evaluating diverse hardware implementations, parallelization prospects and possible improvements", *in Proc. of Second International Conference on Applied Innovations in IT*, Koethen, Germany, March 26-27, 2014, pp. 11-20.

[56] Danijela Efnusheva, Aristotel Tentov, Natasha Tagasovska, "An efficient 64-point IFFT hardware module design", *New Trends in Networking, Computing, E-learning, Systems Sciences, and Engineering*, Springer (ISBN 978-3-319-06764-3), pp. 463-470, November 2014.

[57] D. Efnusheva, G. Dokoski, A. Tentov, M. Kalendar, "Different Approaches for OFDM Transmitter and Receiver Design in Hardware, FPGA Design and Implementation with Performance Comparison", *International Journal of Advanced Research in Computer Science and Software Engineering* (ISSN 2277-128X), Volume 3, Issue 10, pp. 1163-1172, 2013.

[58] Danijela Efnusheva, Aristotel Tentov, "Хардверска реализација на FFT модул со 64 точки во FPGA", *in Proc. of XI International Conference ETAI*, Macedonia, pp. EI-4, 2013.

[59] Richard E. Blahut, *Fast Algorithms for Signal Processing*, United Kingdom, Cambridge University Press, 2010.

[60] Niti Madan, "Asynchronous micro engines for network processing", *Master Thesis*, School of Computing, University of Utah, 2006.

[61] D. Jakimovska, G. Dokoski, A. Tentov, M. Kalendar, "Network processors: evolution and trends", *in Proc. of IX National Conference with International Participation ETAI*, Ohrid, R. Macedonia, pp. IE2-4, September 2009.

[62] Danijela Jakimovska, Goce Dokoski, Marija Kalendar, Aristotel Tentov, "Design and HDL implementation of original 64–bit network processor core", *in Proc. of XVIII Telecommunications Forum – TELFOR*, Belgrde, R.Serbia, pp. 155 - 158, November 2010.

[63] D. Jakimovska, A. Tentov, S. Gjorgjievska, G. Dokoski, M. Kalendar, "Performance Estimation of Novel 32-bit and 64-bit RISC based Network Processor Cores", *Cyber Journals: Multidisciplinary Journals in Science and Technology, Journal of Selected Areas in Telecommunications (JSAT) - ISSN: 1925-2676*, May Edition, 2011.

[64] Danijela Efnusheva, Marija Kalendar, Aristotel Tentov, Goce Dokoski, "Модифициран мемориско-центричен пристап за обработка на мрежни пакети", *in Proc. of XIII International Conference ETAI*, Ohrid, Macedonia, September 22-24, 2016.

[65] Danijela Efnusheva, Ana Cholakoska, Aristotel Tentov, "Design of reconfigurable memory for fast network packet header parsing", *in Proc. of 7th International Conference on Information Society and Technology - ICIST*, Kopaonik, Serbia, March, 2017.

[66] Danijela Efnusheva, Aristotel Tentov, Ana Cholakoska, Marija Kalendar, "FPGA implementation of IP packet header parsing hardware", *in Proc. of Fifth International Conference on Applied Innovations in IT*, Koethen, Germany, pp. 33-40, March 2017.

[67] B. Cogswell, Z. Segall, "MACS: a predictable architecture for real time systems", *in Proc. of Real-Time Systems Symposium*, 1991.

[68] Joao M. P. Cardoso, Michael Hubner, *Reconfigurable Computing: From FPGAs to Hardware/Software Codesign*, Springer-Verlag, New York, 2011.

[69] Xilinx, "VC709 evaluation board for the Virtex-7 FPGA", *User guide*, 2016.

[70] Xilinx, "Xilinx Virtex-7 FPGA VC709 connectivity kit", *User guide*, available online: https://www.xilinx.com/products/boards-and-kits/dk-v7-vc709-g.html

[71] Xilinx, "Getting started Vivado Design Suite user guide", *User guide*, 2014.

[72] Danijela Jakimovska, Goce Dokoski, Marija Kalendar, Aristotel Tentov "Network processor architecture design for multi-gigabit networks", *in Proc. of VIII International Symposium on Industrial Electronics - INDEL*, Banja Luka, BiH, pp. 357 – 362, November 2010.

[73] Simon Hauger, Thomas Wild, Arthur Mutter, Andreas Kirstädter, Kimon Karras, Rainer Ohlendorf, Frank Feller, Joachim Scharf, "Packet processing at 100 Gbps and beyond—challenges and perspectives", *in Proc. of the 10. ITG Symposium on Photonic Networks*, 2009.

[74] Thilan Ganegedara, Viktor K. Prasanna, "StrideBV: single chip 400G+ packet classification", *in Proc. of IEEE Conference on High Performance Switching and Routing*, Serbia, 2012.

[75] Kenneth Vollmar and Pete Sanderson, "MARS: an education-oriented MIPS assembly language simulator", *in Proc. of the 37th SIGCSE Tech. Symp. on Computer Science Education*, 2007.

[76] S. F. Chevtchenko, R. F. Vale, "A comparison of RISC and CISC architectures", *Technical paper*, 2013.

[77] J. Stuecheli, "Power8", *HotChips25*, August 2013.

[78] Ronald Kalla, Balaram Sinharoy, William J. Starke, Michael Floyd, "Power7: IBM's next-generation server processor", *in Proc. of IEEE MICRO Conference*, May 2010.

[79] P. Suresh, Rajat Moona, "PERL - a registerless architecture", *in Proc. of 5th IEEE International Conference On High Performance Computing*, 1998.

[80] Randi Thomas, "An architectural performance study of the fast Fourier transform on vector IRAM", *Technical Paper*, Berkeley University of California, 2000.

[81] M. Oskin, F. T. Chong, T. Sherwood, "Active pages a computation model for intelligent memory", *in Proc. of the 25th annual int. symposium on Computer architecture*, 1998.

[82] ARM, "ARM architecture reference manual", *Technical Paper*, 2005.

[83] P. Weisberg and Y. Wiseman, "Using 4KB page size for virtual memory is obsolete", *In Proc. of IEEE Conference on Information Reuse and Integration*, pp. 262-265, USA, 2009.

[84] P. Weisberg and Y. Wiseman, "Virtual memory systems should use larger pages rather than the traditional 4KB pages", *International Journal of Hybrid Information Technology*, Vol. 8, No. 8, 2015.

[85] Kai Li, "IVY: a shared virtual memory system for parallel computing", *in Proc. of the 1988 International Conference on Parallel Processing*, pp. 94-101, 1988.

[86] IEEE, "IEEE-754 standard for floating-point arithmetic", *IEEE Standard*, 2008.

[87] R. Giladi, *Network processors - architecture, programming and implementation*, Ben-Gurion University of the Negev and EZchip Technologies Ltd., 2008.

[88] H. J. Chao, Bin Liu, *High performance switches and routers*, Wiley-IEEE Press, May 2007.

[89] M. Ahmadi, S. Wong, "Network processors: challenges and trends", *in Proc. of the 17th Annual Workshop on Circuits, Systems and Signal Processing*, pp. 222-232, 2006.

[90] Panos C. Lekkas, *Network processors: architectures, protocols and platforms*, McGraw-Hill Professional, 2003.

[91] M. Shorfuzzaman, R. Eskicioglu, P. Graham, "Architectures for network processors: key features, evaluation, and trends", *in Proc. of Communications in Computing*, 2004.

[92] J. Naous, G. Gibb, S. Bolouki, N. McKeown, "NetFPGA: reusable router architecture for experimental research", *in Proc. of the SIGCOMM PRESTO Workshop*, USA, 2008.

[93] M. Petraccaa, R. Birkea, A. Bianco, "HERO: High speed enhanced routing operation in software routers NICs", *in Proc. of the IEEE Telecommunication Networking Workshop on QoS in Multiservice IP Networks*, Politec. di Torino, 2008.

[94] Intel Corporation, "Intel® IXP2800 and IXP2850 network processors", *Product Brief*, 2005.

[95] Bob Doud, "Accelerating the data plane with the Tile-mx manycore processor", *in Proc. of Linley Data Center Conference*, 2015.

[96] P. Gupta, S. Lin, and N. McKeown, "Routing lookups in hardware at memory access speeds", *in Proc of IEEE Conference on Computer Communications*, pp. 1240–1247, 1998.

[97] W. Eatherton, G. Varghese, Z. Dittia, "Tree bitmap: hardware/software IP lookups with incremental updates", *in Sigcomm Computer Communication Review*, vol. 34, no. 2, 2004.

[98] L. Kekely, V. Puš, and J. Kořenek, "Software defined monitoring of application protocols", *in Proc of IEEE Conference on Computer Communications*, pp. 1725–1733, 2014.

[99] R. Bolla, R. Bruschi, C. Lombardo, F. Podda, "OpenFlow in the small: a flexible and efficient network acceleration framework for multi-core system", *in IEEE Transactions on Network and Service Management*, pp. 390-404, 2014.

[100] V. Puš, L. Kekely, and J. Kořenek, "Design methodology of configurable high performance packet parser for FPGA", *in Proc of 17th International Symposium on Design and Diagnostics of Electronic Circuits Systems*, pp. 189–194, 2014.

[101] M. Attig and G. Brebner, "400 Gb/s programmable packet parsing on a single FPGA", *in Proc. of Seventh ACM/IEEE Symposium on Architectures for Networking and Communications Systems (ANCS)*, pp. 12–23, 2011.

[102] G. Brebner and W. Jiang, "High-speed packet processing using reconfigurable computing", *IEEE Micro*, vol. 34, no. 1, pp. 8– 18, 2014.

[103] A. Moestedt, P. Sjödin, T. Köhler, "Header processing requirements and implementation complexity for IPv4 routers", *White paper*, HP Laboratories, Bristol, September, 1998.

[104] Xilinx, "Design flows overview, Vivado Design Suite user guide", *User guide*, 2014.

[105] Xilinx, "7 Series FPGAs configurable logic block", *User guide*, 2016.

[106] The reference community for free and open source gateware IP cores, www.opencores.org.

[107] Pyparsing, http://pyparsing.wikispaces.com/.

[108] R. Thomas, K. Yelick, "Efficient FFTs on IRAM", *in Proc. of the 16th International Parallel and Distributed Processing Symposium*, 2002.

[109] W. Cheney, D. Kincaid, *Numerical Mathematics and Computing*, 6th ed., Thomson Brooks, 2008.

[110] IETF, "Internet protocol, version 6 (IPv6) specification ", *IETF Standard RFC2460*, 1998.

[111] Ана Чолакоска, "Проектирање и реализација на филтер на мрежни пакети", *Магистерски Труд*, ФЕИТ-Скопје, Македонија, 2017.