

Компютърни архитектури
Рачунарске архитектуре
Архитектуры ЭВМ
Počítačové architektury
Számítógép-architektúrák
Tietokonearkkitehtuurit
Αρχιτεκτονικές υπολογιστών
Arhitecturi de calculatoare
Architetture dei calcolatori
Architectures des ordinateurs
Arquitecturas informáticas
Arquiteturas de computadores
Computer architectures
Rechnerarchitekturen
Datorarkitekturer
Datamaskinarkitekturer
Bilgisayar mimarileri
Համակարգչային հարկածային արհեստագիտություն
计算机体系结构
計算機架構
コンピュータアーキテクチャ
컴퓨터 아키텍처
கணினி கட்டமைப்புகள்
חבליטות מחשב
عماريات الحاسوب

Използван е материал от трудове на:

Цветан Таслаков
Пламенка Боровска
Димитър Тянев
Янко Янев
Атанас Атанасов
Светлин Наков
Евгени Георгиев
Борис Цилькер
Сергей Орлов
Виктор Хорошевский
John Hennessy
David Patterson
William Stallings
David Harris
Sarah Harris
Earl Joseph
Marco Santambrogio
Lawrence Spracklen
Sarah Hitzeman
Jinghe Zhang
William Bryg
Jerome Alabado

Russ Bishop
Orion Lawlor
Fu-Hau Hsu
Dezső Sima
Henrik Wannheden
Seungryoul Maeng
Gary Kacmarcik
George Wells
Richard Paul
Dominic Sweetman
Sivarama Dandamudi
Tony Chen
Andrew Waterman
Reed Foster
Sun Microsystems Corp.
Cray Research Corp.
International Business Machines Corp.
Intel Corp.
Cypress Semiconductor Corp.
Advanced RISC Machines Ltd.
Fujitsu Ltd.
HIBEE Technologies

Конспект по ОККА – II ч.

0. Архитектурни особености на съвременните компютри. Тенденции.

1. Производителност. Ефективност – видове.

2. CISC/RISC. Суперскаларни ЦП. Множество функционални устройства. ЦП с VLIW.

3. Таксономия на Флин: SISD/SIMD/MISD/MIMD. Закон на Амдал. Суперкомпютри.

4. SMP/MPP/NUMA. Мрежи. Клъстери.

5. Многоядреност/многонишковост. SMT/CMT.

6. Векторни, матрични и графични процесори. „Копачи“. SIMD-команди на ЦП.

7. Понятие за конвейер. Особенности в работата на конвейера за команди.

8. Междукомандни зависимости – същност и начини за намаляване на влиянието им.

9. Преходи. Предиктор на преходи/стойности. Спекулативно изпълнение. Задържан преход.

10. Свърхоперативна (кеш-)памет: асоциативност, запис, обем на блока, кохерентност.

11. Хоризонтална организация: Архитектура Харвард, разслоена памет, многоканален режим.

12. Разположение на данните в паметта. Подредба на байовете.

13. Програмен модел, методи на адресация, преходи, пренос и битови полета в IA-32.

14. Програмен модел, методи на адресация, преходи, пренос и битови полета в ARM.

15. Програмен модел, методи на адресация, преходи, пренос и битови полета в POWER[PC].

16. Програмен модел, методи на адресация, преходи, пренос и битови полета в SPARC.

17. Програмен модел, методи на адресация, преходи, пренос и битови полета в MIPS.

18. Програмен модел, методи на адресация, преходи, пренос и битови полета в RISC-V.

19. Програмен модел, методи на адресация, преходи, пренос и битови полета в друга архитектура по избор.

Архитектурни особености на съвременните компютри:

1. Широко използване на RISC (поне вътрешно)
2. Паралелност $\forall$ нива: ядра/нишки/ФУ/конвейер
3. Серийност на шините и др. вътрешни връзки
4. Свърхоперативни памети (*cache*) и буфериране
5. Използване на синхронни динамични памети
6. Използване на честотни умножители (ФСС) в ИС
7. Вътрешночипови мрежи (за сървърите и СК)
8. Автоматичен контрол на ТЧ и изключване на ФУ
9. Вградена поддръжка на криптография и ИИ/МО

АРХИТЕКТУРА НА СЪВРЕМЕННИТЕ КОМПЮТЪРНИ СИСТЕМИ

ДВА ОСНОВНИ АСПЕКТА:

1. *АРХИТЕКТУРНИ ИНОВАЦИИ*
2. *ЕФЕКТИВНО ИЗПОЛЗВАНЕ НА
ТЕХНОЛОГИЧНИТЕ ПОДОБРЕНИЯ*

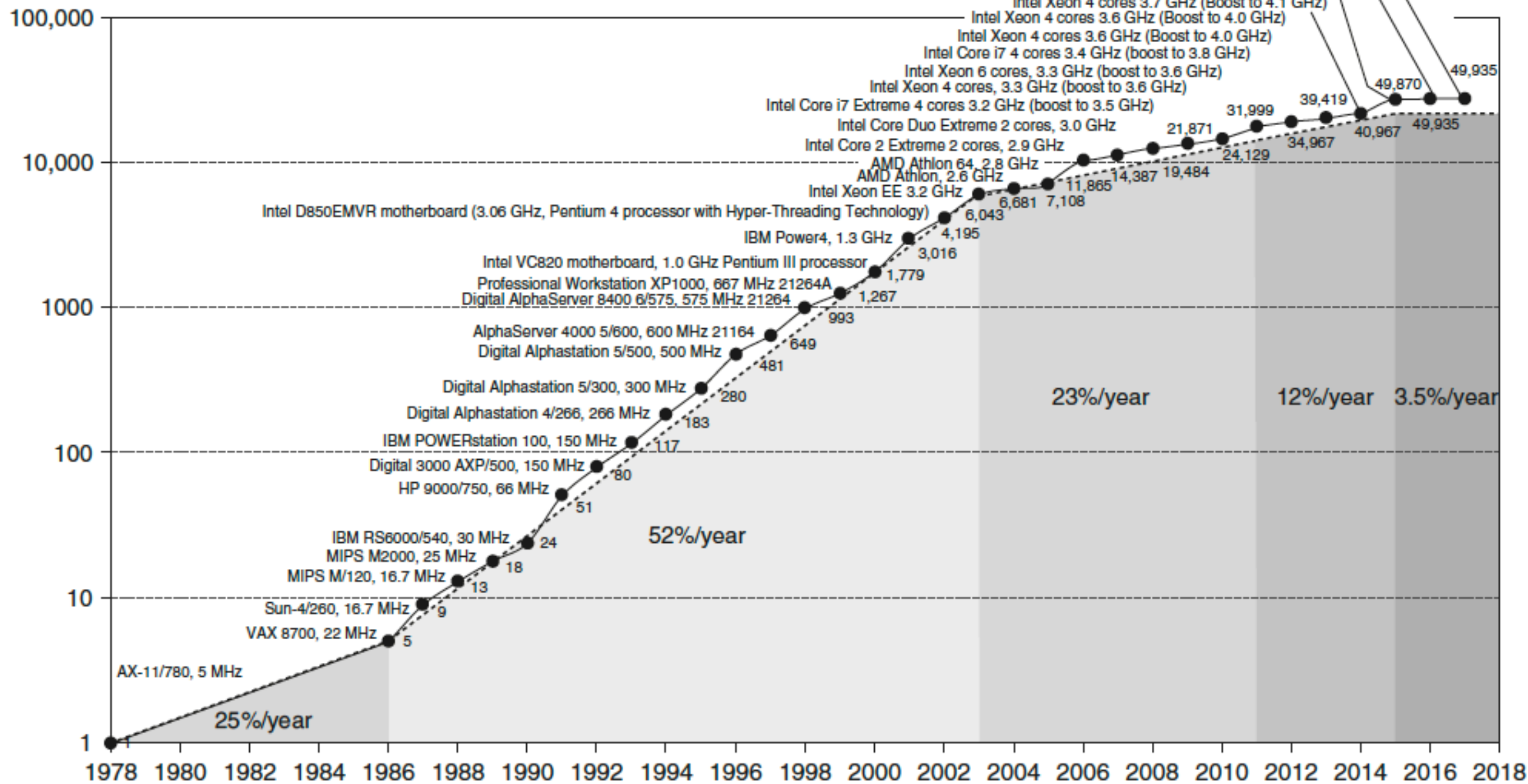
ТЕНДЕНЦИИ:

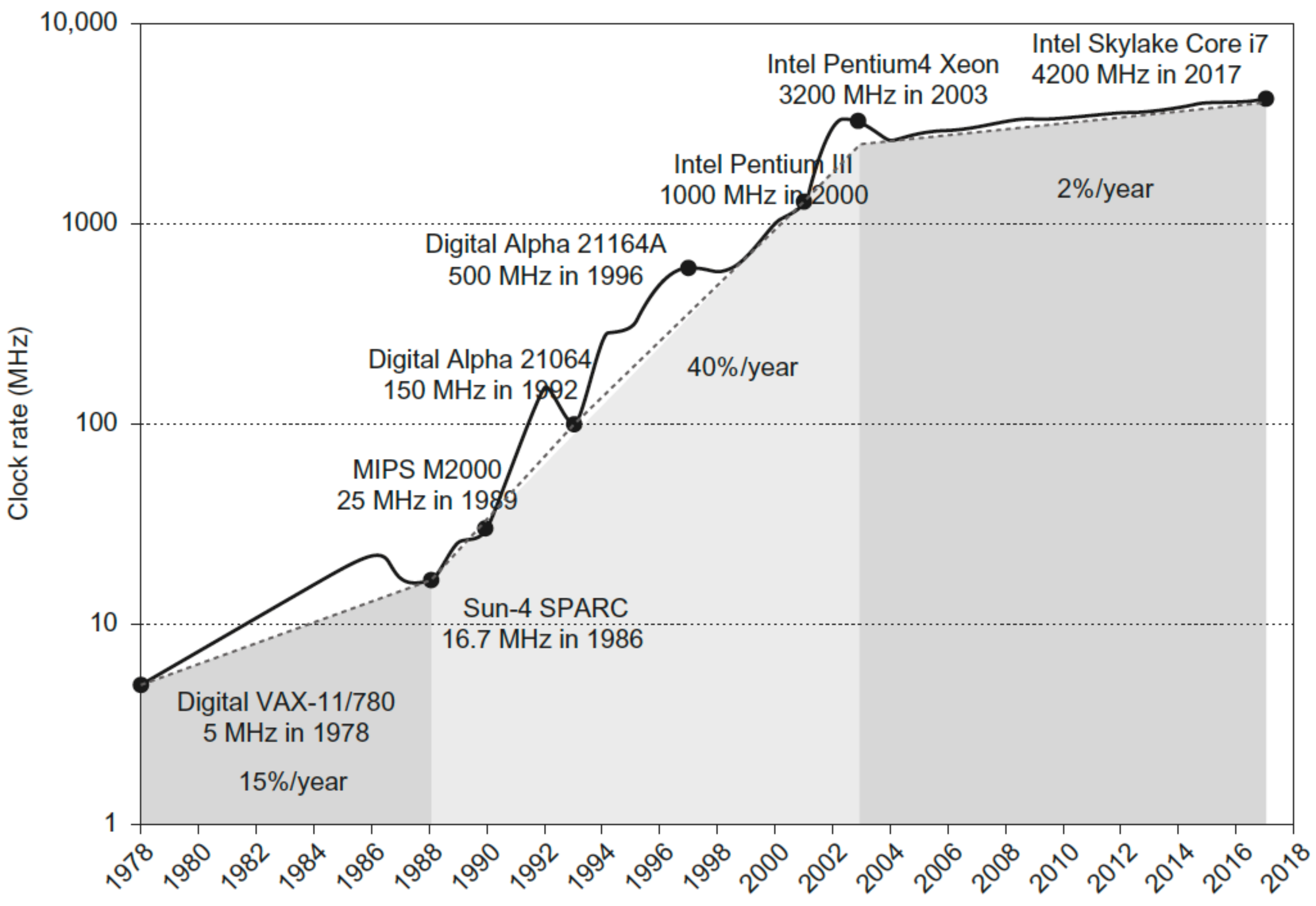
- RISC- базирани машини
- Критични техники за повишаване на производителността: ILP (паралелизми на ниво инструкции), кеш – памети, TLP

Current Trends in Architecture

- Cannot continue to leverage Instruction-Level parallelism (ILP)
 - Single processor performance improvement ended in 2003
- New models for performance:
 - Data-level parallelism (DLP)
 - Thread-level parallelism (TLP)
 - Request-level parallelism (RLP)
- These require explicit restructuring of the application

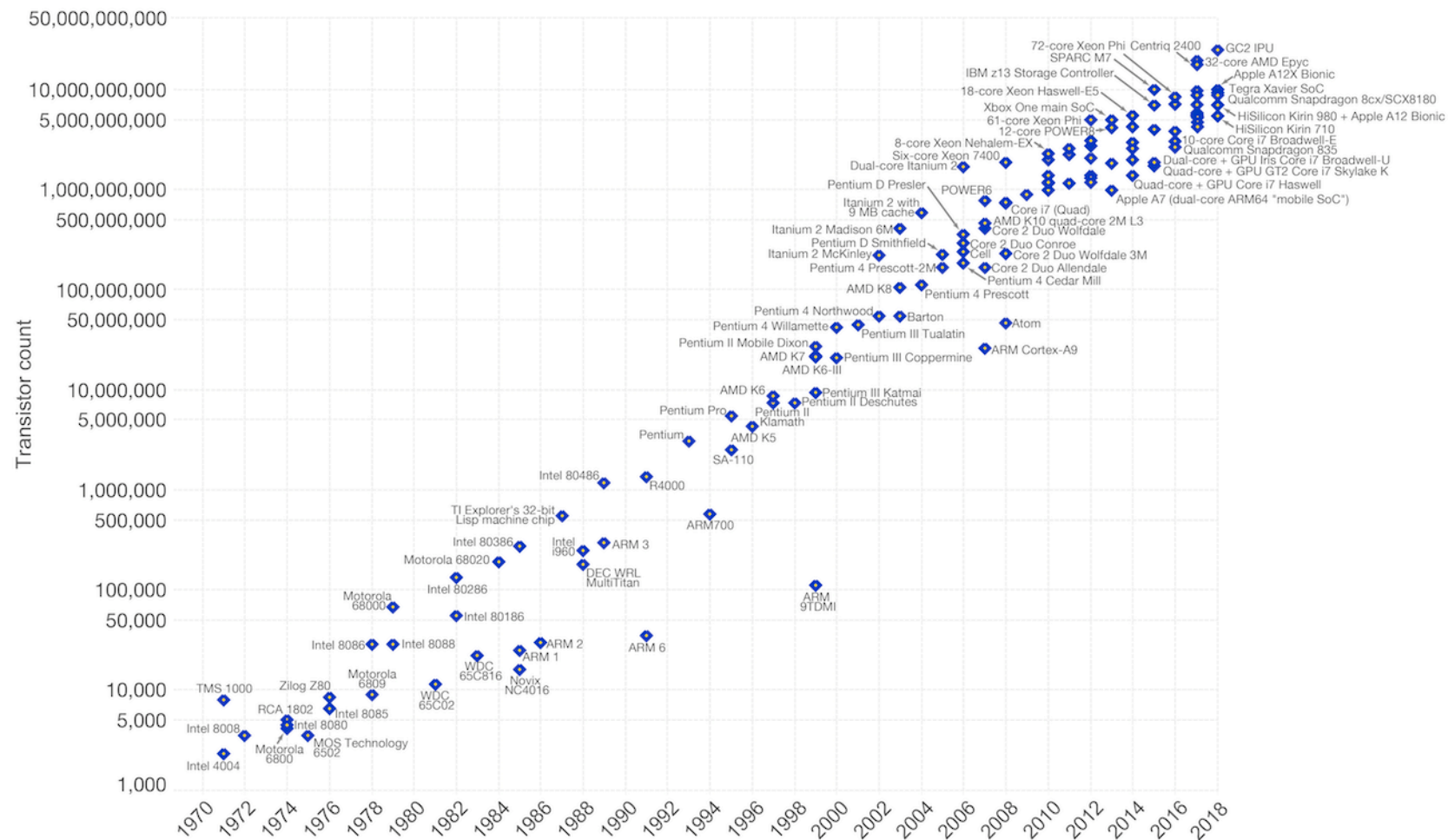
Performance (vs. VAX-11/780)





Moore's Law – The number of transistors on integrated circuit chips (1971-2018)

Moore's law describes the empirical regularity that the number of transistors on integrated circuits doubles approximately every two years. This advancement is important as other aspects of technological progress – such as processing speed or the price of electronic products – are linked to Moore's law.



Data source: Wikipedia (https://en.wikipedia.org/wiki/Transistor_count)

The data visualization is available at [OurWorldinData.org](https://www.ourworldindata.org). There you find more visualizations and research on this topic.

Licensed under CC-BY-SA by the author Max Roser.

Gates' Law:
The speed of
software halves
every 18 months.

Measuring Performance

- Typical performance metrics:
 - Response time
 - Throughput
- Speedup of X relative to Y
 - $\text{Execution time}_Y / \text{Execution time}_X$
- Execution time
 - Wall clock time: includes all system overheads
 - CPU time: only computation time
- Benchmarks
 - Kernels (e.g. matrix multiply)
 - Toy programs (e.g. sorting)
 - Synthetic benchmarks (e.g. Dhrystone)
 - Benchmark suites (e.g. SPEC06fp, TPC-C)

Мерки за производителност

- **MIPS** (**M**illions of **I**nstructions **P**er **S**econd) – милиони инструкции за секунда
- **MOPS** (**M**illions of **O**perations **P**er **S**econd) – милиони операции за секунда
- **FLOPS** (**F**loating **P**oint **O**perations **P**er **S**econd) – брой операции с плаваща запетая за секунда

ЕФЕКТИВНОСТ – ВИДОВЕ:

- ◆ Ефективност на паралелната работа: $E = S / N$
- ◆ Икономическа (ценова) ефективност: $E = S / V$
- ◆ Енергийна ефективност/икономичност: $E = S / P$
- ◆ Закон на *Grosch* (1953 г.): Производителността нараства с квадрата на цената. Или с други думи, колкото по-скъп е компютърът, толкова по-голяма икономическа (ценова) ефективност има. Подобна мисъл е изказвана и от *Seymour Cray*. По-късните изследвания сочат, че тази зависимост се размива.

Идеята при **RISC** архитектурата е да се постигне бързодействие, посредством осигуряване на минимално време за изпълнение на често използваните команди. Този подход, а също така и обстоятелството, че сложната система от команди изисква по-големи системни издръжки, води до система със съкратен набор команди.

За да бъдат два процесора функционално еквивалентни, единият от които е **CISC**, а другият **RISC**, е необходимо функциите на процесора, реализирани от рядко срещаните **CISC** команди да се заменят с последователност от **RISC** команди в **RISC** процесора.

Понастоящем няма единно определение за това, какво е **RISC** – архитектура. Все пак преобладава становището, че за "чистата" **RISC** – архитектура са характерни следните особености:

- а) командите трябва да се изпълняват за един и същ брой тактове (в идеалния случай за един такт);
- б) командите трябва да имат един и същ формат;
- в) обръщението към паметта е свързано само с команди от тип **LOAD** и **STORE**;
- г) всички аритметични и логически функции се изпълняват на ниво регистър.

Трябва да се подчертае, че не във всички **RISC** – процесори, по една или друга причина, се реализират и четирите изисквания едновременно. Най-често се нарушават изисквания а) и б).

Rank	80x86 instruction	Integer average (% total executed)
1	load	22%
2	conditional branch	20%
3	compare	16%
4	store	12%
5	add	8%
6	and	6%
7	sub	5%
8	move register-register	4%
9	call	1%
10	return	1%
Total		96%

Figure A.13 The top 10 instructions for the 80x86. Simple instructions dominate this list and are responsible for 96% of the instructions executed. These percentages are the average of the five SPECint92 programs.

Number of memory addresses	Maximum number of operands allowed	Type of architecture	Examples
0	3	Load-store	Alpha, ARM, MIPS, PowerPC, SPARC, SuperH, TM32
1	2	Register-memory	IBM 360/370, Intel 80x86, Motorola 68000, TI TMS320C54x
2	2	Memory-memory	VAX (also has three-operand formats)
3	3	Memory-memory	VAX (also has two-operand formats)

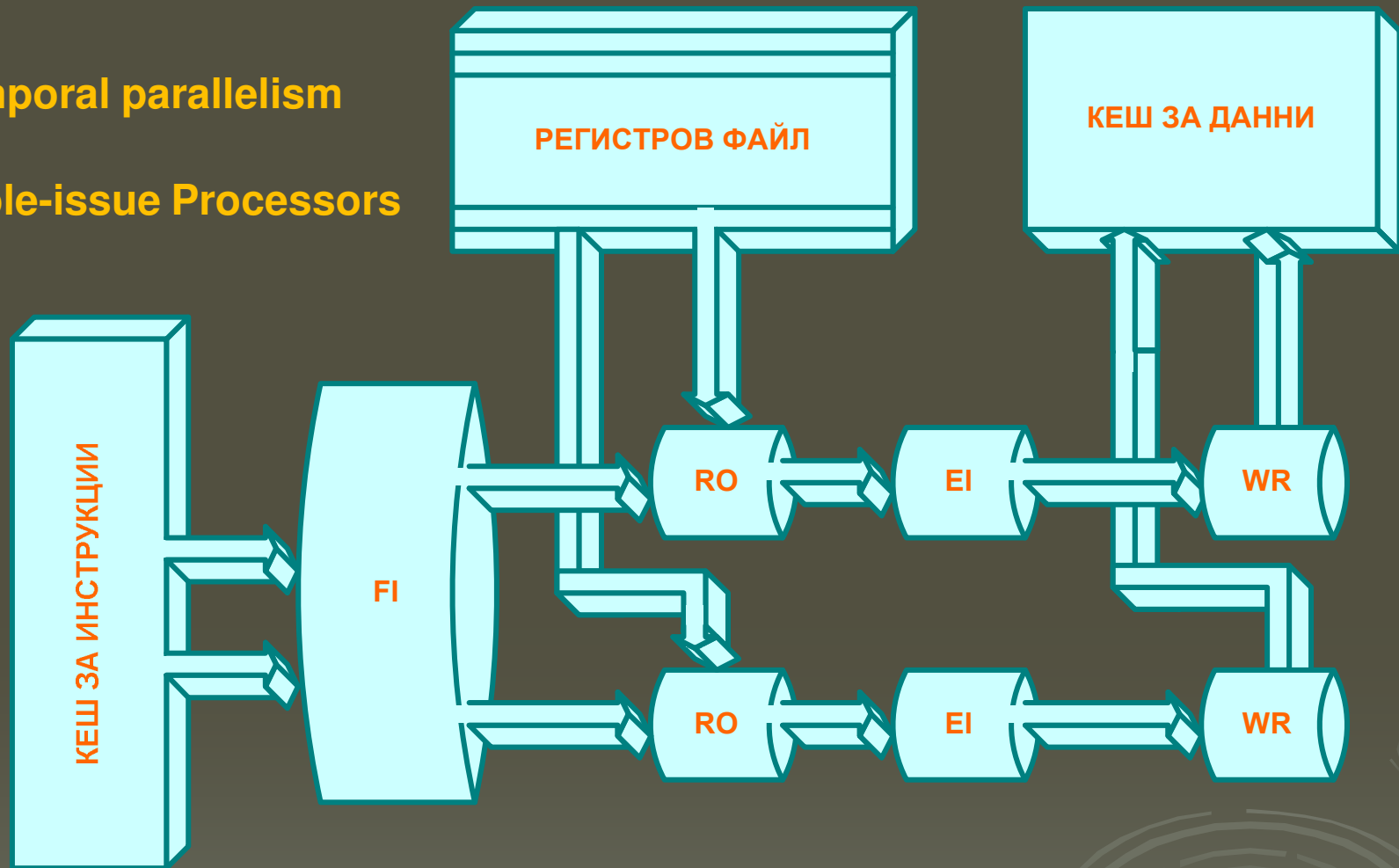
Figure A.3 Typical combinations of memory operands and total operands per typical ALU instruction with examples of computers. Computers with no memory reference per ALU instruction are called load-store or register-register computers. Instructions with multiple memory operands per typical ALU instruction are called register-memory or memory-memory, according to whether they have one or more than one memory operand.

Type	Advantages	Disadvantages
Register-register (0, 3)	Simple, fixed-length instruction encoding. Simple code generation model. Instructions take similar numbers of clocks to execute (see Appendix C).	Higher instruction count than architectures with memory references in instructions. More instructions and lower instruction density lead to larger programs.
Register-memory (1, 2)	Data can be accessed without a separate load instruction first. Instruction format tends to be easy to encode and yields good density.	Operands are not equivalent since a source operand in a binary operation is destroyed. Encoding a register number and a memory address in each instruction may restrict the number of registers. Clocks per instruction vary by operand location.
Memory-memory (2, 2) or (3, 3)	Most compact. Doesn't waste registers for temporaries.	Large variation in instruction size, especially for three-operand instructions. In addition, large variation in work per instruction. Memory accesses create memory bottleneck. (Not used today.)

Figure A.4 Advantages and disadvantages of the three most common types of general-purpose register computers. The notation (m, n) means m memory operands and n total operands. In general, computers with fewer alternatives simplify the compiler's task since there are fewer decisions for the compiler to make (see Section A.8).

СУПЕРСКАЛАРНИ ПРОЦЕСОРИ

temporal parallelism
Multiple-issue Processors



Структура на суперскаларен процесор с два
инструкционни конвейера



Изпълнение с пренареждане на инструкциите

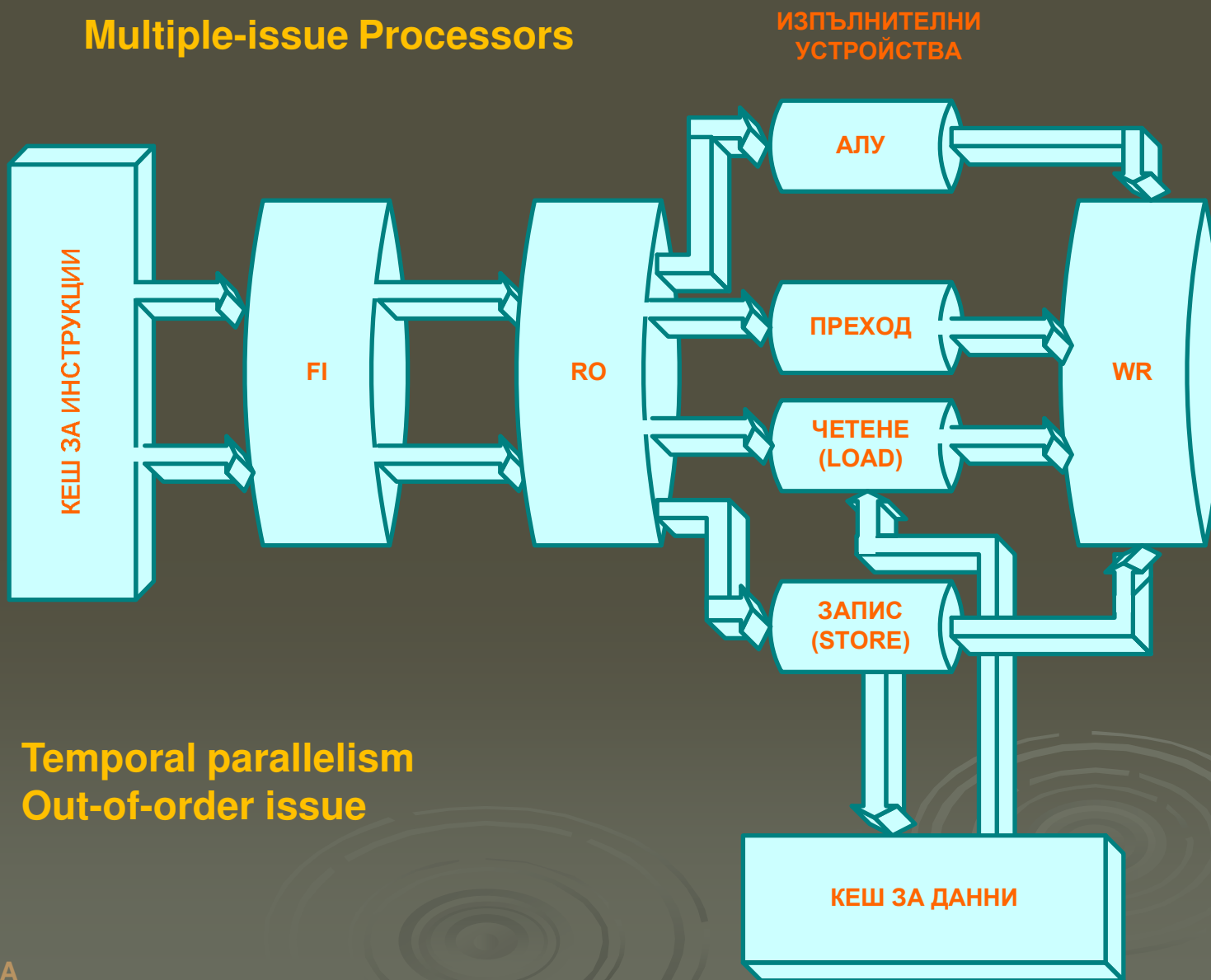
Out-of-order Instruction Issue

- Множество функционални устройства
- Могат да се имплементират множество АЛУ-та
- Инструкционен буфер, наречен *инструкционен прозорец* (*instruction window*), се имплементира между фазата за извличане на инструкциите и изпълнителната фаза
- Инструкциите се избират за изпълнение от инструкционния прозорец, когато операндите им са готови и съответното функционално устройство е свободно (*основен принцип на data flow компютрите*)
- Инструкционният прозорец може да се имплементира по 2 начина:
 1. Централизиран инструкционен прозорец
 2. Децентрализиран инструкционен прозорец



Суперскаларен процесор със специализирани изпълнителни устройства

Multiple-issue Processors



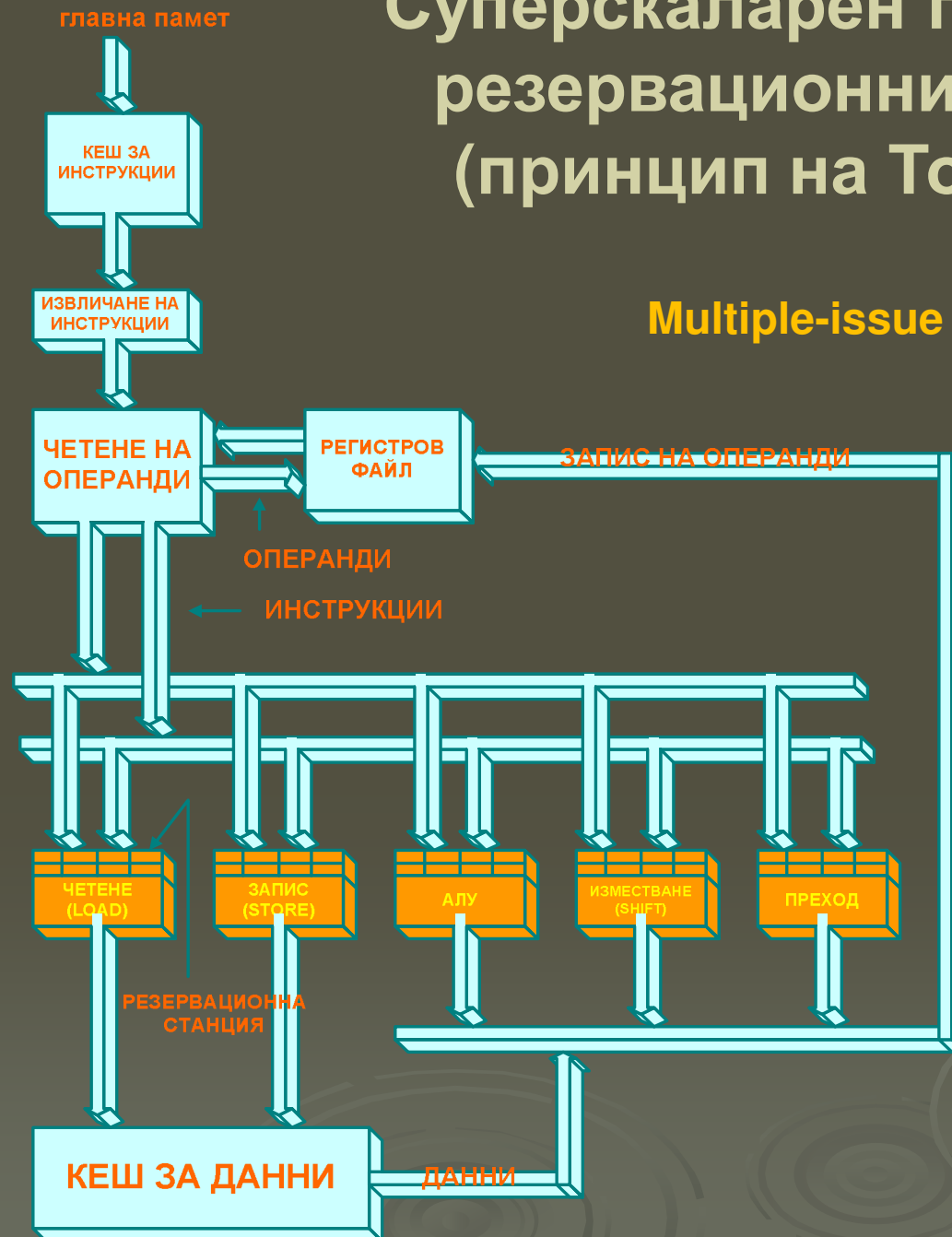
Суперскаларен процесор с централизиран инструкционен прозорец

Multiple-issue Processors



Суперскаларен прозорец с резервационни станции (принцип на Томасуло)

Multiple-issue Processors



Съдържание на инструкционния прозорец

Instruction Window Contents

Instruction	Opcode	Register Destination ID	Operand 1	Register Operand 1 ID	Operand 2	Register Operand 2 ID
1	Opcode 1	ID 1	VALUE			ID 1A
2	Opcode 2	ID 2	VALUE			ID 2A
3	Opcode 3	ID 3	VALUE		VALUE	
4	Opcode 4	ID 4			VALUE	
5	Opcode 5	ID 5	VALUE	ID		ID 5A
...	...	...	...	...	...	...
...	...	...	...	...	...	...

ГОТОВА ЗА ИЗПЪЛНЕНИЕ –
ИЗПРАЩА СЕ КЪМ ФУНКЦИОНАЛНОТО УСТРОЙСТВО

DATA FLOW PRINCIPLE



Superscalar Example

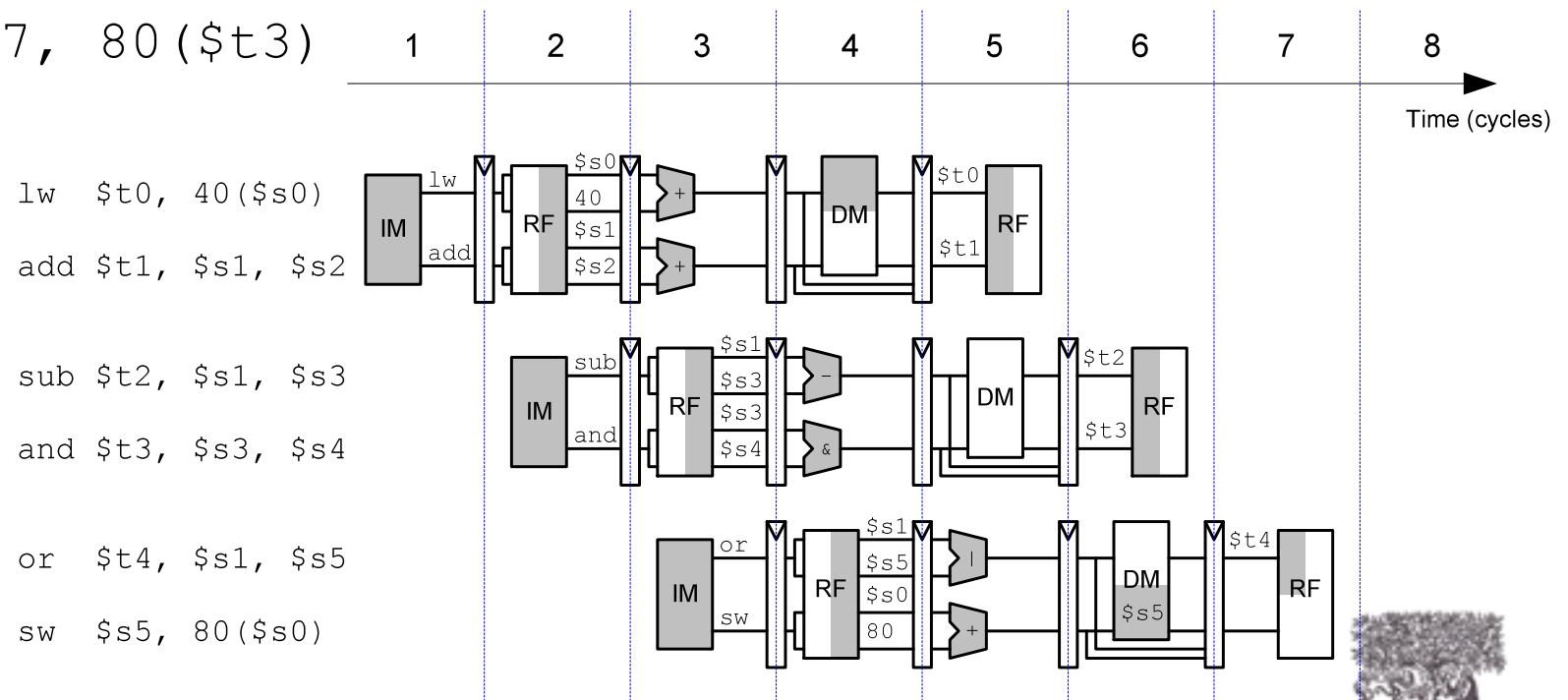
```

lw    $t0, 40($s0)
add   $t1, $t0, $s1
sub   $t0, $s2, $s3
and   $t2, $s4, $t0
or    $t3, $s5, $s6
sw    $s7, 80($t3)

```

Ideal IPC: 2

Actual IPC: 2



Superscalar with Dependencies

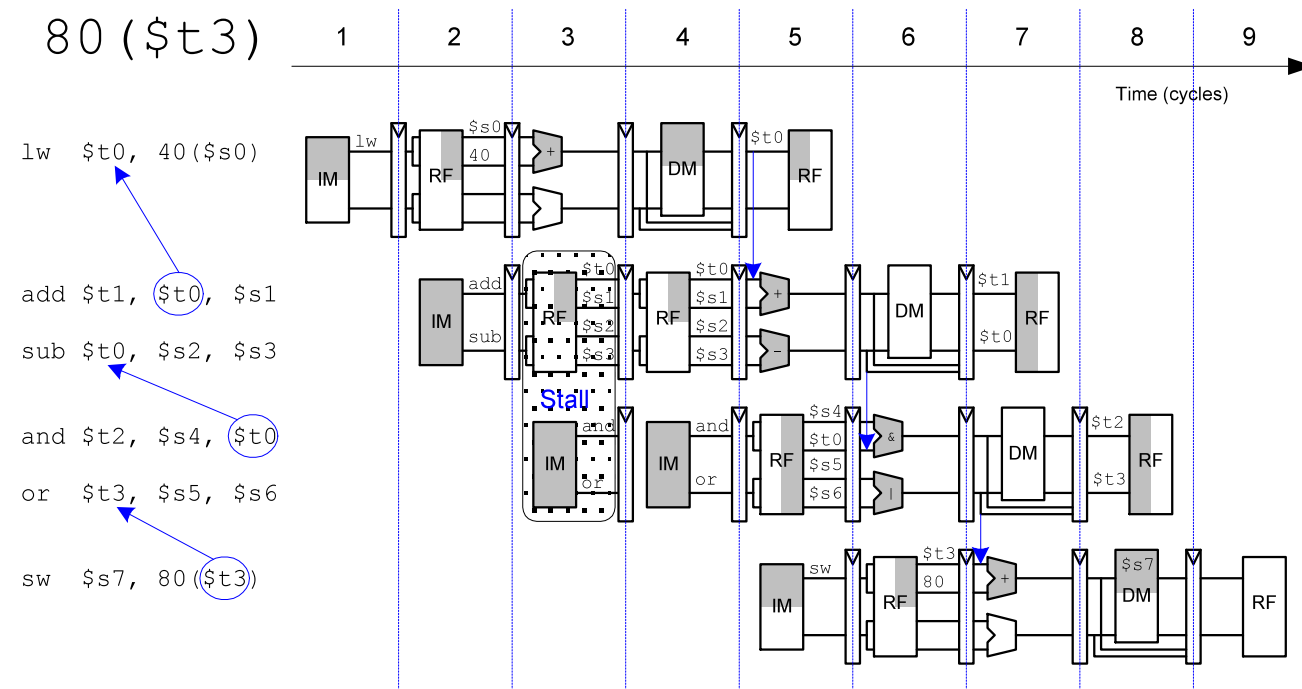
```
lw    $t0, 40($s0)
add   $t1, $t0, $s1
sub   $t0, $s2, $s3
and   $t2, $s4, $t0
or    $t3, $s5, $s6
sw    $s7, 80($t3)
```

Ideal IPC:

2

Actual IPC:

6/5 = 1.2



Out of Order Processor Example

```
lw    $t0, 40($s0)
add   $t1, $t0, $s1
sub   $t0, $s2, $s3
and   $t2, $s4, $t0
or    $t3, $s5, $s6
sw    $s7, 80($t3)
```

Ideal IPC:

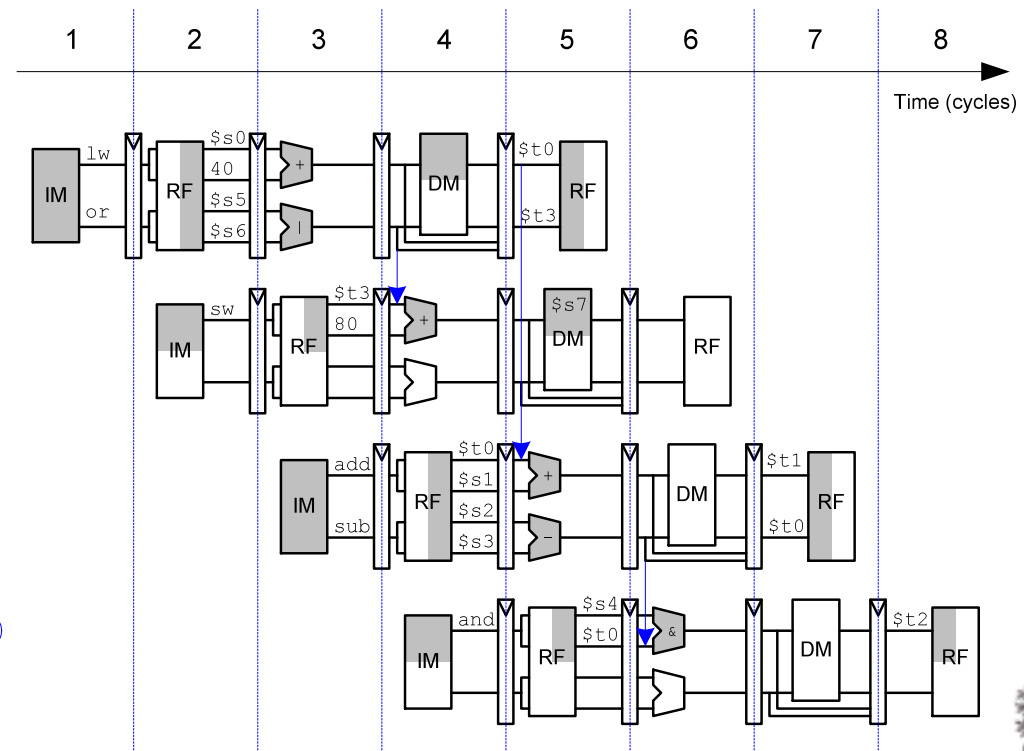
2

Actual IPC:

$6/4 = 1.5$

lw \$t0, 40(\$s0)
 or \$t3, \$s5, \$s6
 sw \$s7, 80(\$t3)
 add \$t1, \$t0, \$s1
 sub \$t0, \$s2, \$s3
 and \$t2, \$s4, \$t0

RAW
 two cycle latency
 between load and
 use of \$t0
 WAR
 RAW



Register Renaming

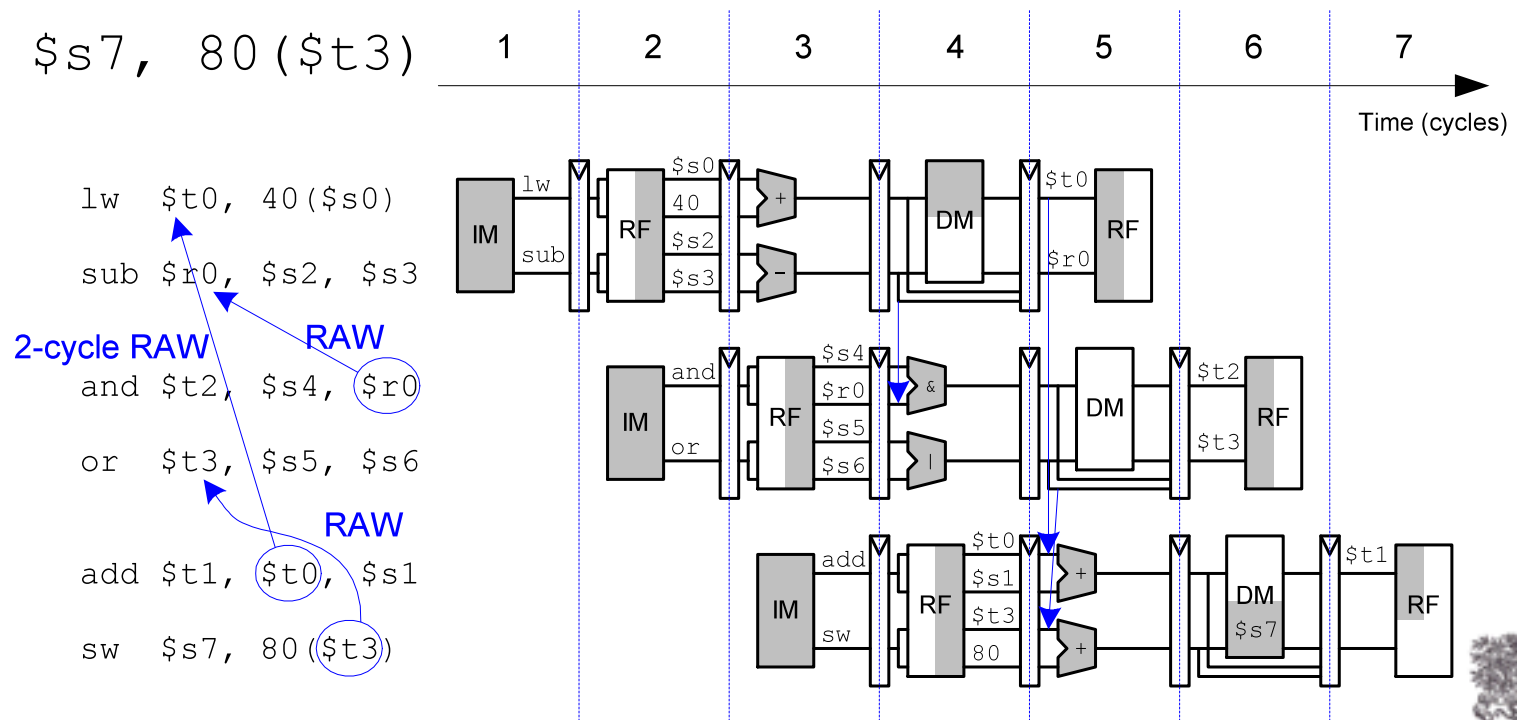
```
lw    $t0, 40($s0)
add   $t1, $t0, $s1
sub   $t0, $s2, $s3
and   $t2, $s4, $t0
or    $t3, $s5, $s6
sw    $s7, 80($t3)
```

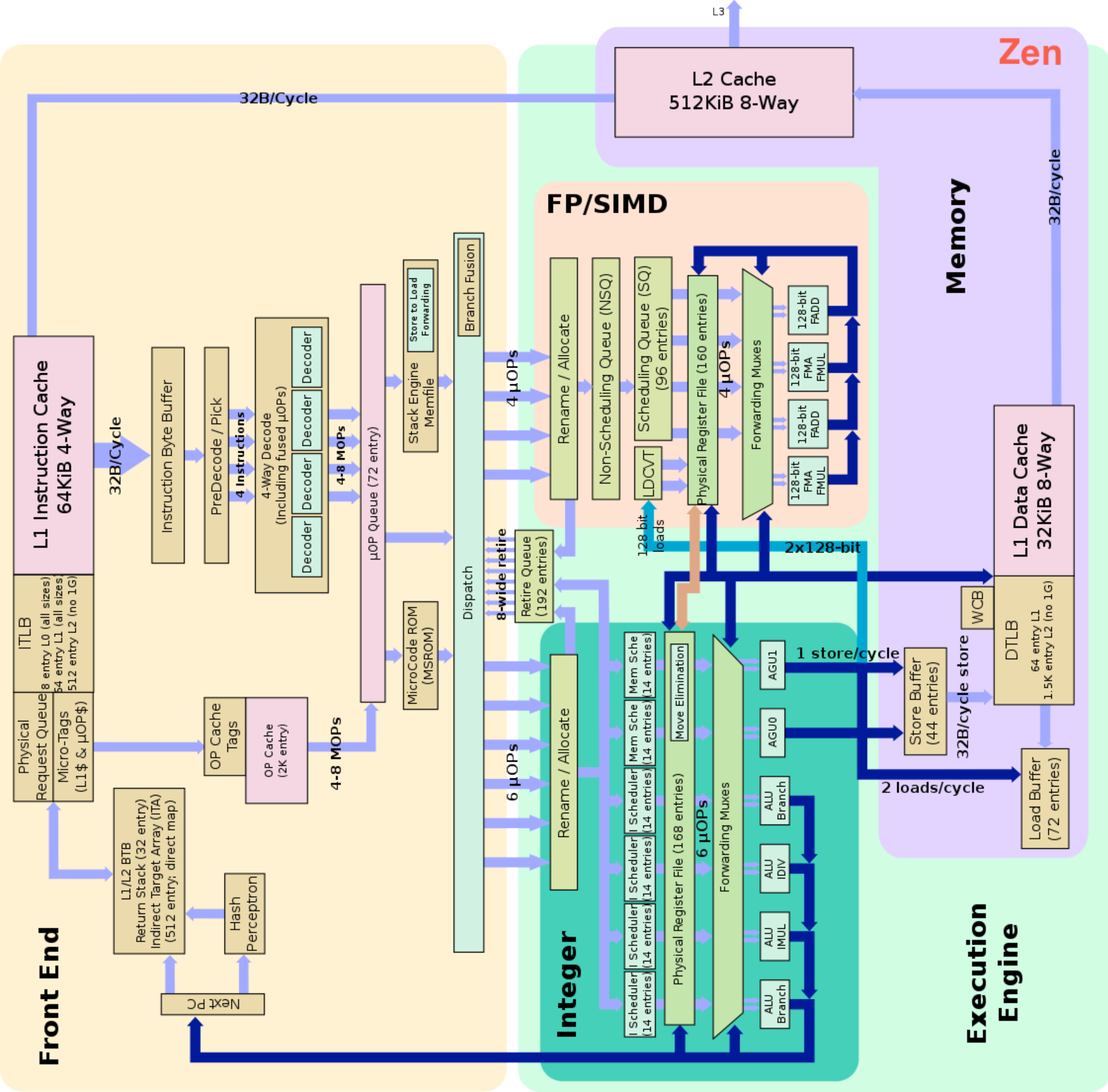
Ideal IPC:

2

Actual IPC:

$6/3 = 2$





ILP = INSTRUCTION-LEVEL PARALLELISM

~~ILP~~ АРХИТЕКТУРИ

ILP- АРХИТЕКТУРИ	ГРУПИРАНЕ	ПЛАНИРАНЕ НА ФУНКЦИОНАЛНИТЕ УСТРОЙСТВА	ИНИЦИИРАНЕ
СУПЕРСКАЛАРНИ	ОТ ХАРДУЕРА	ОТ ХАРДУЕРА	ОТ ХАРДУЕРА
ЕКСПЛИЦИТНО ПАРАЛЕЛНО ИЗПЪЛНЕНИЕ НА ИНСТРУКЦИИ EPIC	ОТ КОМПИЛАТОРА	ОТ ХАРДУЕРА	ОТ ХАРДУЕРА
ДИНАМИЧНИ АРХИТЕКТУРИ С МНОГО ДЪЛГИ ИНСТРУКЦИОННИ ДУМИ (Dynamic VLIW)	ОТ КОМПИЛАТОРА	ОТ КОМПИЛАТОРА	ОТ ХАРДУЕРА
АРХИТЕКТУРИ С МНОГО ДЪЛГИ ИНСТРУКЦИОННИ ДУМИ (VLIW)	ОТ КОМПИЛАТОРА	ОТ КОМПИЛАТОРА	ОТ КОМПИЛАТОРА

VLIW ARCHITECTURES

EPIC

- ◉ VLIW - Very Long Instruction Word
- ◉ EPIC - Explicitly Parallel Instruction Computing
- ◉ При типичната VLIW архитектура дължината на инструкциите е стотици битовете.
- ◉ В рамките на VLIW процесор функционират паралелно множество функционални устройства.
- ◉ Всички функционални устройства споделят огромен общ регистров файл.

АРХИТЕКТУРНО СРАВНЕНИЕ

АРХИТЕКТУРНИ ХАРАКТЕРИСТИКИ	CISC	RISC	VLIW
ДЪЛЖИНА НА ИНСТРУКЦИЯТА	ПРОМЕНЛИВА	ЕДНАКВА, ОБИКНОВЕНО 32 БИТА	ЕДНАКВА
ФОРМАТ НА ИНСТРУКЦИЯТА	РАЗПОЛОЖЕНИЕТО НА ПОЛЕТАТА СЕ ПРОМЕНЯ	РЕГУЛЯРНО, РАЗПОЛОЖЕНИЕТО НА ПОЛЕТАТА НЕ СЕ ПРОМЕНЯ	РЕГУЛЯРНО, РАЗПОЛОЖЕНИЕТО НА ПОЛЕТАТА НЕ СЕ ПРОМЕНЯ
СЕМАНТИКА НА ИНСТРУКЦИИТЕ	ПРОМЕНЯ СЕ ОТ ПРОСТА ДО СЛОЖНА	САМО ЕДНА ОПЕРАЦИЯ	МНОЖЕСТВО ПРОСТИ, НЕЗАВИСИМИ ОПЕРАЦИИ
РЕГИСТРИ	МАЛКО, ПОНЯКОГА СПЕЦИАЛИЗИРАНИ	МНОГО, УНИВЕРСАЛНИ	МНОГО, УНИВЕРСАЛНИ
ДОСТЪПИ ДО ПАМЕТА	ОБВЪРЗАНИ С ОПЕРАЦИИ НА РАЗЛИЧНИ ТИПОВЕ ИНСТРУКЦИИ	НЕ СА ОБВЪРЗАНИ С ОПЕРАЦИИ, LOAD/STORE АРХИТЕКТУРИ	НЕ СА ОБВЪРЗАНИ С ОПЕРАЦИИ, LOAD/STORE АРХИТЕКТУРИ
АКЦЕНТ ПРИ ПРОЕКТИРАНЕТО НА ХАРДУЕРА	ИЗПОЛЗВАНЕ НА МИКРОКОДИРАНИ ИМПЛЕМЕНТАЦИИ	ИМПЛЕМЕНТАЦИИ С ЕДИН КОНВЕЙЕР И БЕЗ МИКРОКОД	ИМПЛЕМЕНТАЦИИ С МНОЖЕСТВО КОНВЕЙЕРИ, БЕЗ МИКРОКОД И БЕЗ СЛОЖНА ЛОГИКА ЗА ДИСПЕЧЕРИЗАЦИЯ



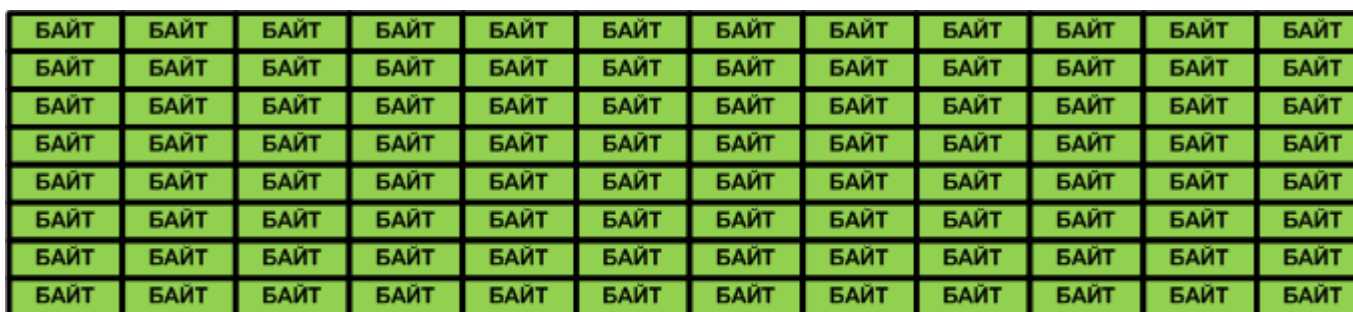
ИЗПЪЛНЕНИЕ НА ТИПИЧНИ ИНСТРУКЦИИ



CISC



RISC



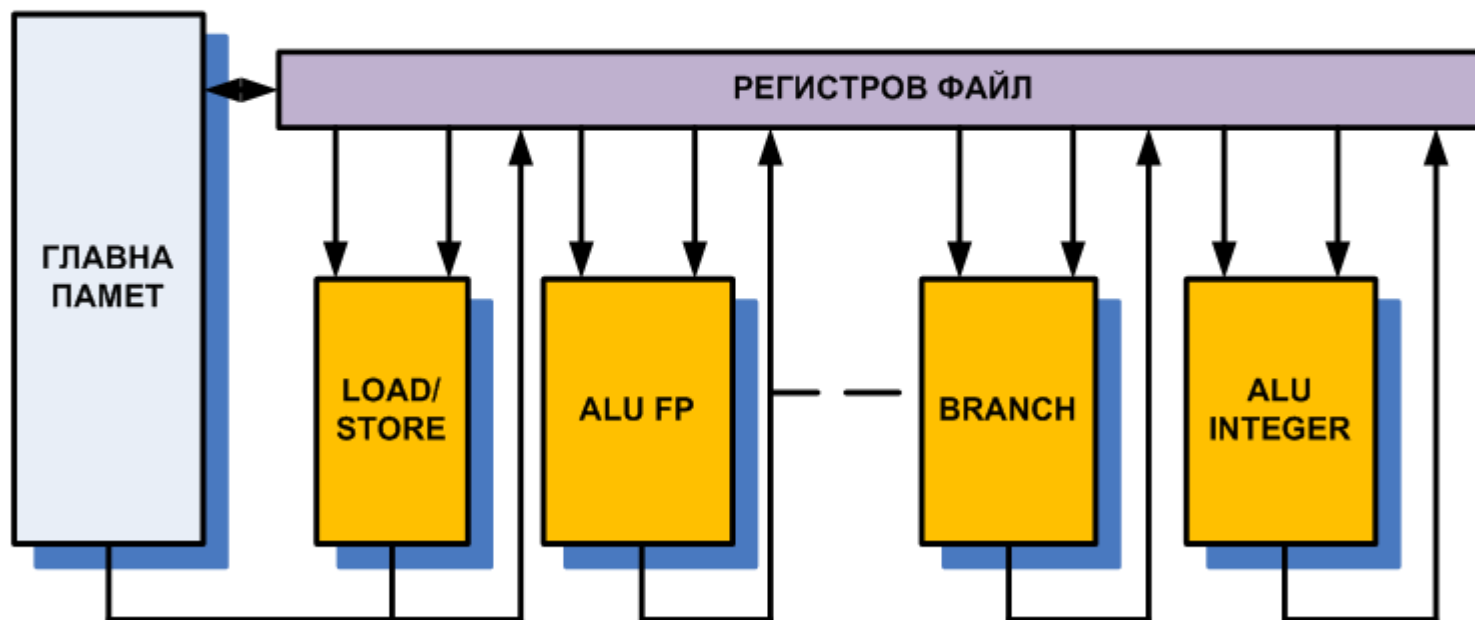
VLIW



АРХИТЕКТУРА НА VLIW ПРОЦЕСОР



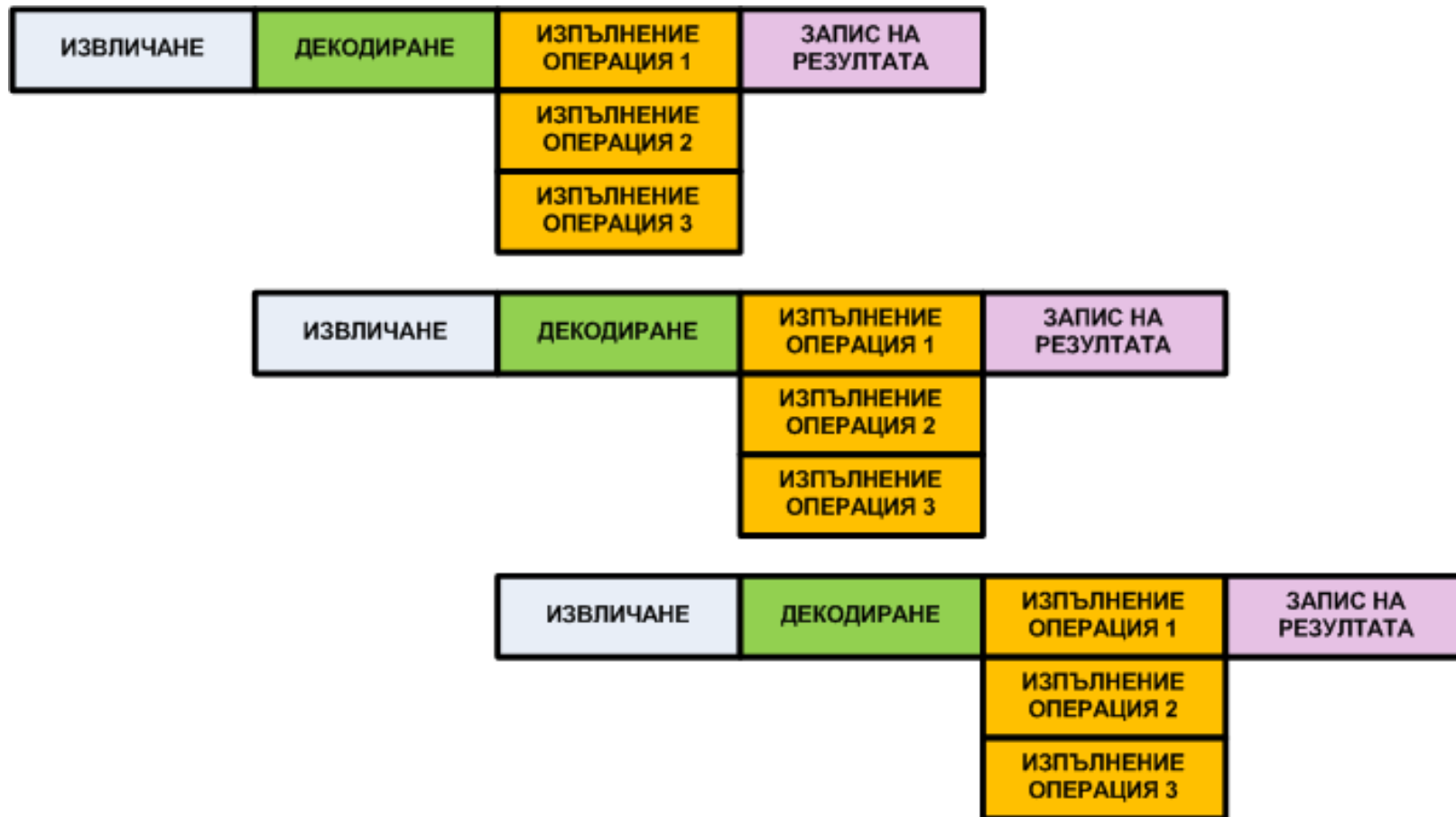
ФОРМАТ НА ИНСТРУКЦИЯТА ПРИ VLIW АРХИТЕКТУРА



АРХИТЕКТУРА НА VLIW ПРОЦЕСОР

ПРОЦЕСОРИ VLIW

spatial parallelism



КОНВЕЙЕРНО ИЗПЪЛНЕНИЕ НА ИНСТРУКЦИИТЕ
ПРИ VLIW ПРОЦЕСОР СЪС СТЕПЕН 3

ПРЕДИМСТВА НА VLIW ПРОЦЕСОРИТЕ

Компилаторите генерират фиксирани инструкционни пакети (instruction packets) с множество операции (в зависимост от степента на конвейера) и създават плана за изпълнението им

- Зависимостите между инструкциите се определят от компилатора и се използват при планирането в съответствие със латентностите на функционалните устройства
- Функционалните устройства се планират от компилатора според позицията в инструкционните пакети ("slotting")
- Компилаторът генерира напълно планиран за изпълнение код, в който липсват **ЗАВИСИМОСТИ** (fully-scheduled, hazard-free code) => не се налага хардуерът да открива зависимости или да планира

НЕДОСТАТЪЦИ НА VLIW ПРОЦЕСОРИТЕ

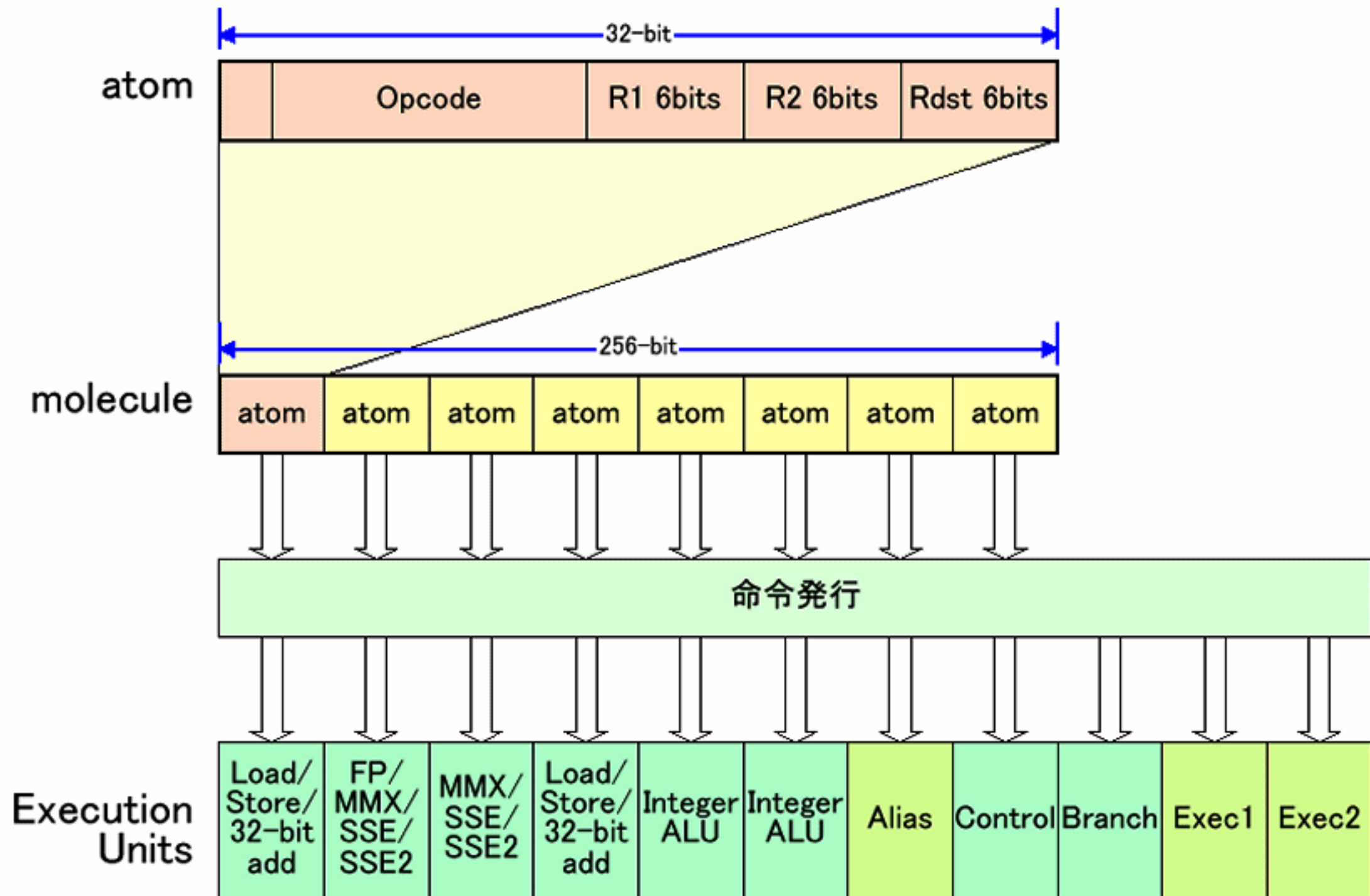
Основен проблем е съвместимостта (Compatibility) на различните имплементации

- Имплементациите са машинно-зависими - VLIW код няма да се изпълнява коректно при различен брой на функционалните устройства и различни латентности
- Непланирани събития (напр., липса в кеша) блокират целия процесор

Друг проблем е плътността на кода (Code density)

- Недостатъчна утилизация на слота (предимно pops)
- Броят на pops може да се намали чрез компресиране ("flexible VLIW", "variable-length VLIW")

EfficeonのVLIW命令



Michael Flynn



Флин през 1966 г. Класификацията на Флин се базира не на структурата на компютъра, а на това как в компютъра командите се обвързват с обработката на данните. Флин въвежда понятието поток и го дефинира така: последователност от елементи (данни и команди) изпълнявани или обработвани от процесорите. Във всеки компютър има два основни потока – единият от команди, а другият от данни. При срещата им се извършва обработка над данните.

В съответствие с това дали потоците от данни или команди са единични или представляват множество, възникват и следните четири големи класа:

SISD (**S**ingle **I**nstruction Stream, **S**ingle **D**ata Stream).

В този клас влизат обикновените фон Ноймановски компютри, които имат един поток команди и един поток данни, т.е. едно устройство за обработка на командите и едно изпълнително устройство.

SIMD (**S**ingle **I**nstruction Stream, **M**ultiple **D**ata Stream).

В тези компютри се съхранява един поток от команди, но вече той е векторен, който инициира многочислени операции. Всеки елемент на вектора се разглежда като елемент на отделен поток от данни. В този клас се включват всички компютри с векторни команди, например CRAY-1, CRAY-2, CRAY X-MP, CRAY CS 6400, ILLIAC-IV, DAP на ICL, OMEN -64 и т.н.

Флин посочва следните проблеми, възникващи при разработката на **SIMD** компютрите:

а) Поддържане на комуникация между изчислителните елементи (процесори) .

б) Необходимост от съответствие между размера на вектора от данни и размера на масива от процесори, който ще обработва този вектор.

в) Наличие на не векторни операции в програмите и появяване на допълнителна работа, свързана с подготовка за изпълнение на векторни операции.

г) Бездействие на голяма част от процесорите при наличие на условни преходи в програмата.

MISD (**M**ultiple **I**nstruction Stream, **S**ingle **D**ata Stream).

Към този клас компютри се отнасят компютри с множество потоци от команди и един поток от данни. Единствената архитектура, която с някакво основание може да бъде отнесена към този клас е конвейерната и то при условие, че всеки етап от изпълнението на командата се счита за една отделна команда. Но много по-естествено е да се отнесат конвейерите към класа **SIMD**. Действително самият конвейер може да се счита като система с архитектура **MIMD**, а по отношение на векторните операции той напомня архитектурата **SIMD**.

MIMD (**M**ultiple **I**nstruction Stream, **M**ultiple **D**ata Stream).

Множеството потоци от команди означава съществуването на няколко устройства за обработка на командите в този клас компютри. Затова тука се включват всички форми на мултипроцесорни конфигурации – от обединени компютри до матрици от процесори. В конфигурациите са включени няколко независими и завършени (пълноценни) еднопроцесорни компютри, които в процеса на решаване на задачите контактуват помежду си чрез съобщения или използват обща памет. Тъй като всеки от процесорите има свое управляващо устройство и може да изпълнява собствена програма, то за даден момент от време отделните процесори имат възможност да изпълняват различни операции върху различни данни.

Следните проблеми са общи за всички **MIMD** компютри:

а) Необходимост от препращане на данни и команди между процесорите. Тези операции обикновено поглъщат много време и за тяхното осъществяване е необходима (в повечето случаи сложна и скъпо струваща) комуникационна мрежа.

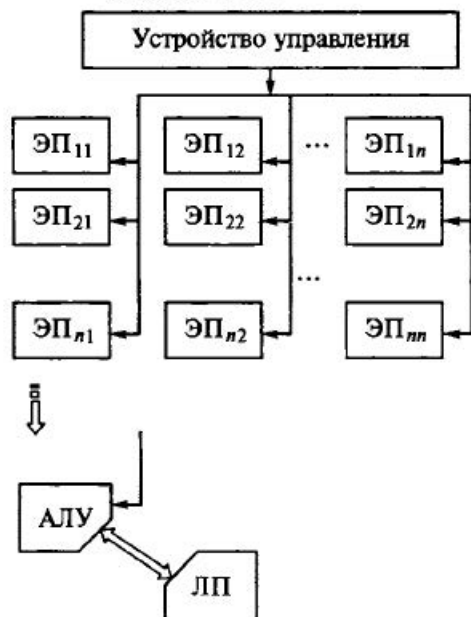
б) Цената се повишава линейно (в най-добрия случай) с увеличаване на броя на процесорите, докато скоростта на обработка се повишава много по-бавно поради увеличаване на броя на конфликтите между процесорите, използващи общи ресурси.

в) Осигуряване на методи за динамична реконфигурация на ресурсите на системата за задоволяване на постоянно променящите се изисквания и заетост на процесорите.

SISD-архитектура; ЭВМ



SIMD-архитектура;
матричные ВС



MISD-архитектура;
конвейерные ВС



MIMD-архитектура;
распределенные мультипроцессорные ВС,
ВС с программируемой структурой

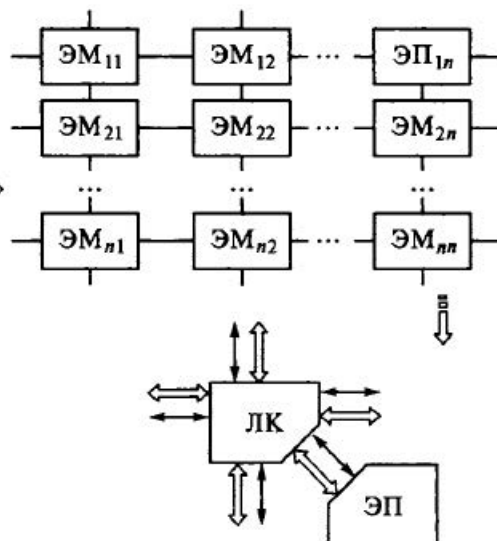


Рис. 3.7. Развитие архитектуры вычислительных средств:

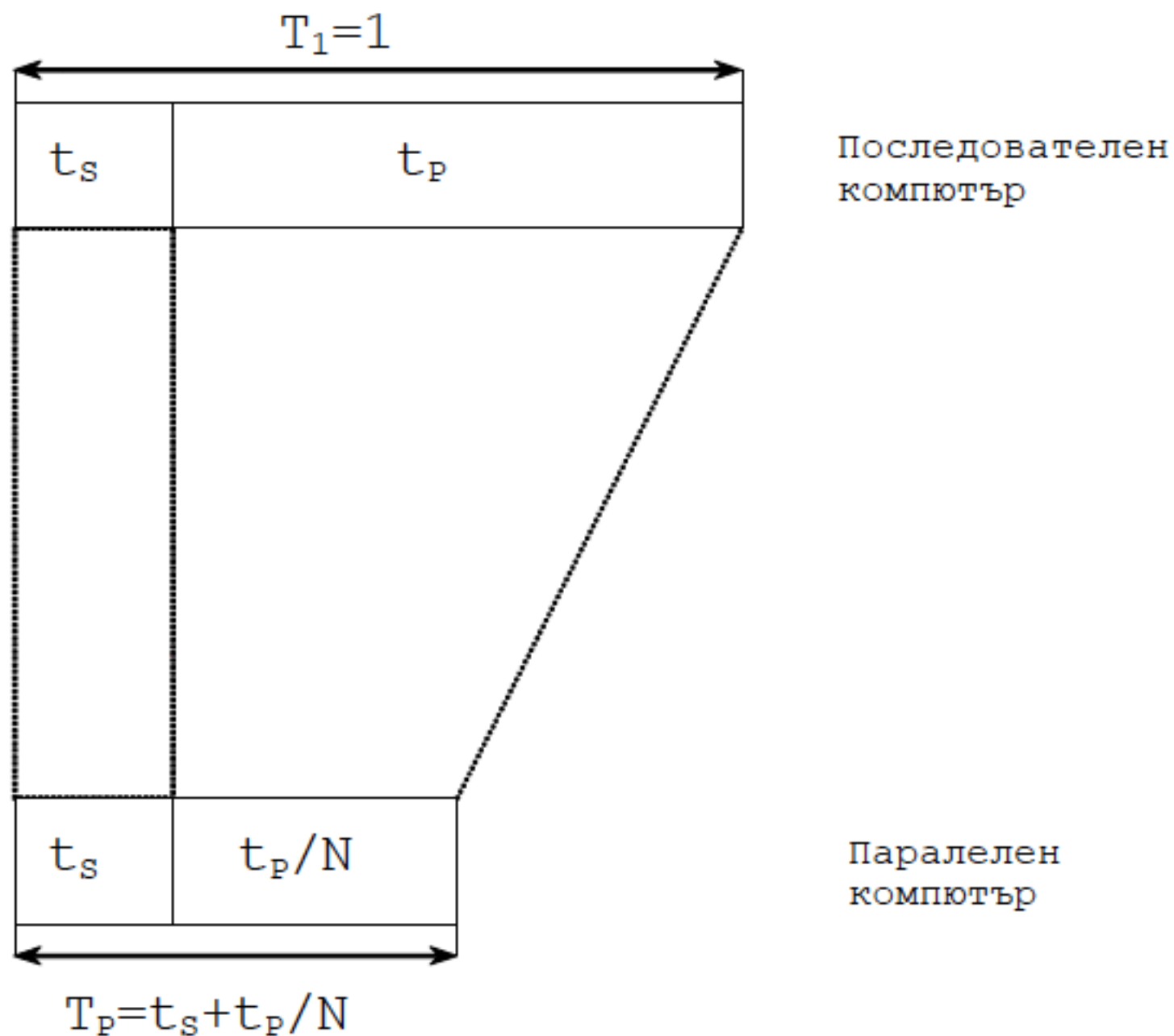
АЛУ — арифметико-логическое устройство; ЭБО — элементарный блок обработки; ЭП — элементарный процессор; ЭМ — элементарная машина; ЛП — локальная память; ЛК — локальный коммутатор; → — поток команд; ⇌ — поток данных; ⇔ — направление трансформации архитектуры



През 1967 г. Amdahl предлага своя закон гласящ, че последователната част от задачата ограничава отгоре коефициента **S**. При това той привежда следните разсъждения. В една задача, решавана на еднопроцесорна система, има част, която може да се реши само последователно (обикновено тази част е свързана с координацията на целия изчислителен процес), а друга част – паралелно. Нека **t_s** е времето изразходвано за работа по последователната част, а **t_p** е времето изразходвано за работа по паралелната част, при това с цел алгебрично опростяване полагаме **t_s+t_p=1**. Тогава, прилагайки формула (3.1) се получава

$$S = \frac{T_1}{T_N} = \frac{t_s + t_p}{t_s + \frac{t_p}{N}} = \frac{1}{t_s + \frac{t_p}{N}}.$$

Ясно е, че с увеличаване на **N** коефициентът **S** асимптотически се стреми към **1/t_s** – (виж фиг.3-2) и следователно последователната обработка, присъща на задачата, ограничава производителността на паралелния компютър.



Фиг. 3-2. Модел на производителността при фиксиран размер на задачата

According to the British "Financial Times" report in January this year, David Kahana, head of the "China Asia Technology Information Program", recently stated that China's first exascale computing system has been in operation for more than a year, and later has developed the second exascale system. In other words, there are at least two E-class supercomputers in China that are already in operation. According to reports, China has three E-class supercomputers, Sunway Ocean Light, Tianhe-3, and Shuguang, which are operating or developing on different technical paths. It is also reported that Sunway Ocean Light has a processor with 42 million cores. Its peak performance reached 1.3 exaflops on the Linpack benchmark. If we compare it with the frontier, we will quickly find that its computing power is equivalent to the sum of the first and second frontier and apex in the United States. If you compare it again, Frontier has 8.73 million cores, and Sunway Ocean Light has 42 million cores, which is 5 times that of Frontier, which shows that its system upgrade potential is amazing.

Why doesn't such a powerful Shenwei·Ocean Light participate in the ranking?

Supercomputing ranking is not just for ranking, but more often, it is a supercomputing communication mechanism. Under this mechanism, everyone can learn from and learn from each other. For example, the evaluation team of Sunway TaihuLight once praised its cooling system; another example is the GPU architecture used by Tianhe 2A; "Real-time Simulation of Stochastic Circuits", its mixed-precision performance can reach 44 billion billion times, and the ratio of its computing power to that of the Sycamore quantum computer is 304 seconds/200 seconds. For another example, in 2022, China also released the first quantum computer and supercomputer collaborative computing system solution (referred to as "supercomputer collaboration"). This solution can take advantage of the advantages of quantum computers and supercomputers in both directions. We just announced this solution And carry on.

Of course, we can also get a lot of benefits from such exchanges. However, since Obama in 2015, the United States has begun to restrict high-performance CPUs. The reason is precisely China's breakthroughs in Sunway and Tianhe supercomputers, which took turns to occupy the top spot for several years. In this case, it is worth not to participate in this ranking, so as not to come up with a few E-class supercomputers to stimulate some people.

	Countries	Count	System Share (%)	Rmax (GFlops)	Rpeak (GFlops)	Cores
1	United States	150	30	2,400,757,320	3,527,842,131	31,248,740
2	China	134	26.8	465,824,474	941,041,056	25,006,176
3	Germany	36	7.2	231,822,060	354,786,306	4,092,324
4	Japan	33	6.6	654,147,240	851,617,520	12,444,236
5	France	24	4.8	175,206,530	252,565,603	4,009,304
6	United Kingdom	14	2.8	62,248,784	88,233,824	1,891,248
7	Canada	10	2	41,208,360	71,911,529	845,984
8	Brazil	9	1.8	55,709,150	106,040,912	868,096
9	South Korea	8	1.6	105,185,000	148,274,050	1,667,876
10	Netherlands	8	1.6	34,907,120	58,411,030	551,328
11	Italy	7	1.4	317,229,000	418,977,448	3,272,304
12	Russia	7	1.4	73,715,000	101,737,460	741,328
13	Sweden	6	1.2	26,710,470	37,640,216	383,360
14	Saudi Arabia	6	1.2	55,253,040	98,982,254	1,798,260
15	Australia	5	1	42,767,840	60,177,375	532,416
16	Ireland	5	1	13,364,020	26,320,890	335,360
17	Switzerland	4	0.8	28,563,790	38,600,165	650,260
18	India	4	0.8	19,453,340	25,251,886	325,832
19	Norway	4	0.8	13,179,910	21,488,020	366,400
20	Singapore	3	0.6	9,037,830	15,785,890	234,112
21	Finland	3	0.6	320,788,310	443,391,175	2,584,576
22	Poland	3	0.6	10,923,400	17,099,060	148,800
23	Taiwan	2	0.4	11,297,560	19,562,790	220,752
24	Czechia	2	0.4	9,589,400	12,914,040	163,584
25	Luxembourg	2	0.4	12,806,560	18,291,180	172,544

TOP 500

The List.

JUNE 2023



Lawrence Berkeley
National Laboratory

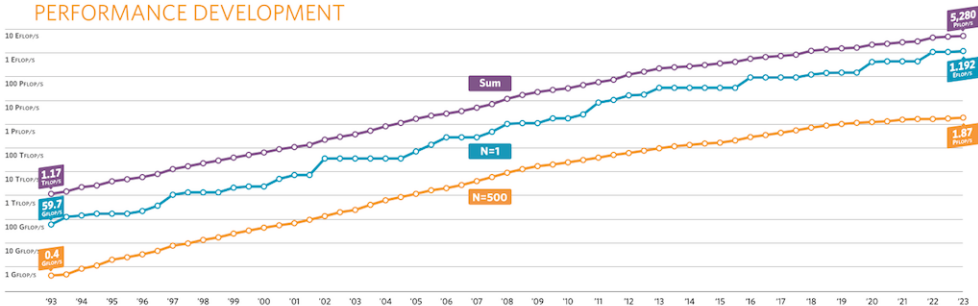


FIND OUT MORE AT
top500.org



			SITE	COUNTRY	CORES	R _{MAX} PFLOP/S	POWER MW
1	Frontier	HPE Cray EX235a, AMD Opt 3rd Gen EPYC (64C 2GHz), AMD Instinct MI250X, Slingshot-11	DOE/SC/ORN	USA	8,699,904	1,194.0	22.7
2	Fugaku	Fujitsu A64FX (48C, 2.2GHz), Tofu Interconnect D	RIKEN R-CCS	Japan	7,630,848	442.0	29.9
3	LUMI	HPE Cray EX235a, AMD Opt 3rd Gen EPYC (64C 2GHz), AMD Instinct MI250X, Slingshot-11	EuroHPC/CSC	Finland	2,220,288	309.0	6.01
4	Leonardo	Atos Bullsequana intelXeon (32C, 2.6 GHz), NVIDIA A100 quad-rail NVIDIA HDR100 Infiniband	EuroHPC/CINEC	Italy	1,824,768	238.7	7.40
5	Summit	IBM POWER9 (22C, 3.07GHz), NVIDIA Volta GV100 (80C), Dual-Rail Mellanox EDR Infiniband	DOE/SC/ORN	USA	2,414,592	148.6	10.1

PERFORMANCE DEVELOPMENT



Frontier Overview

Extraordinary Engineering



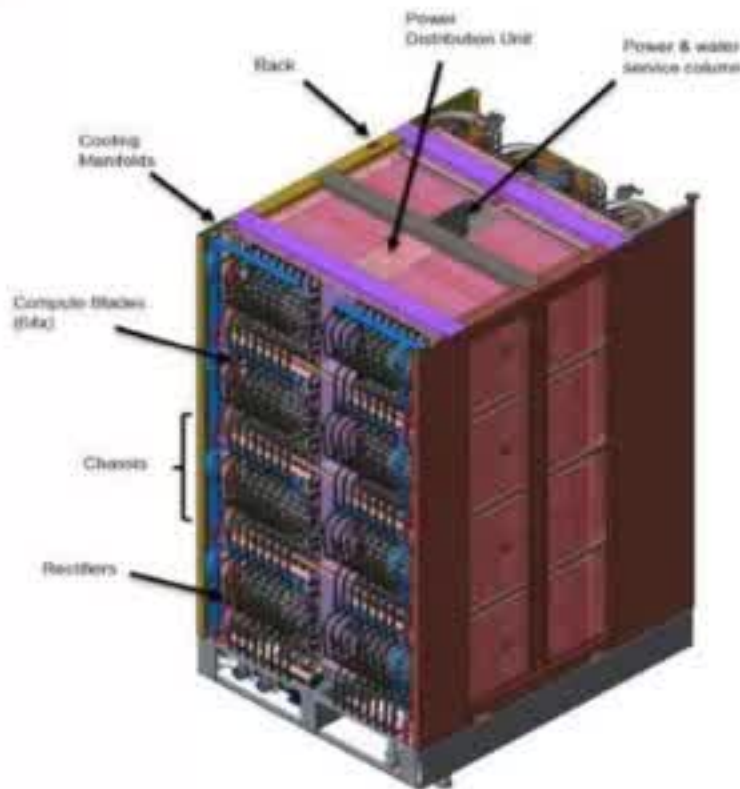
System

- 2 EF Peak DP FLOPS
- 74 compute racks
- 29 MW Power Consumption
- 9,408 nodes
- 9.2 PB memory (4.6 PB HBM, 4.6 PB DDR4)
- Cray Slingshot network with dragonfly topology
- 37 PB Node Local Storage
- 716 PB Center-wide storage
- 4000 ft² foot print

Built by HPE

Olympus rack

- 128 AMD nodes
- 8,000 lbs
- Supports 400 KW



Powered by AMD

AMD node

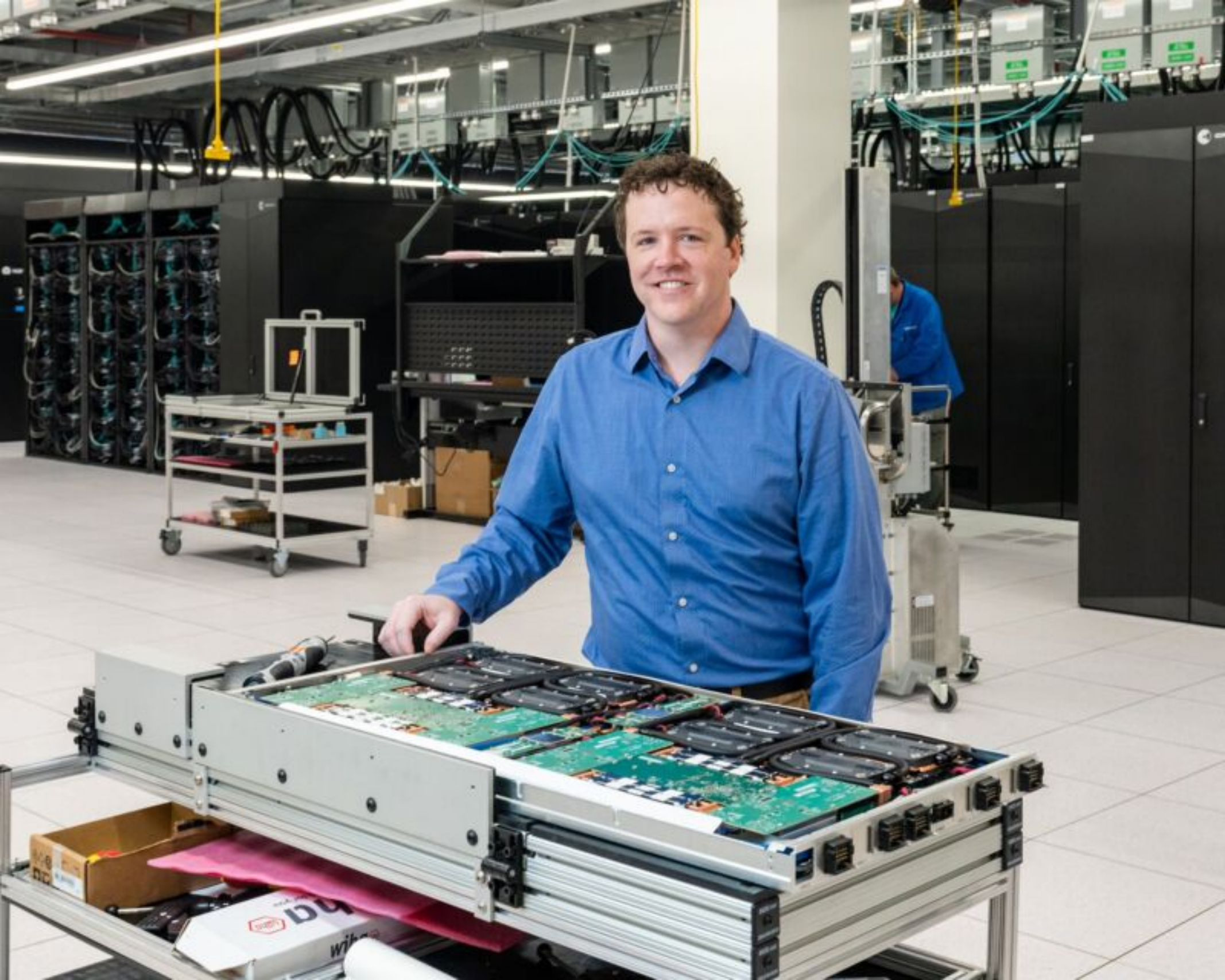
- 1 AMD "Trento" CPU
- 4 AMD MI250X GPUs
- 512 GiB DDR4 memory on CPU
- 512 GiB HBM2e total per node (128 GiB HBM per GPU)
- Coherent memory across the node
- 4 TB NVM
- GPUs & CPU fully connected with AMD Infinity Fabric
- 4 Cassini NICs, 100 GB/s network BW

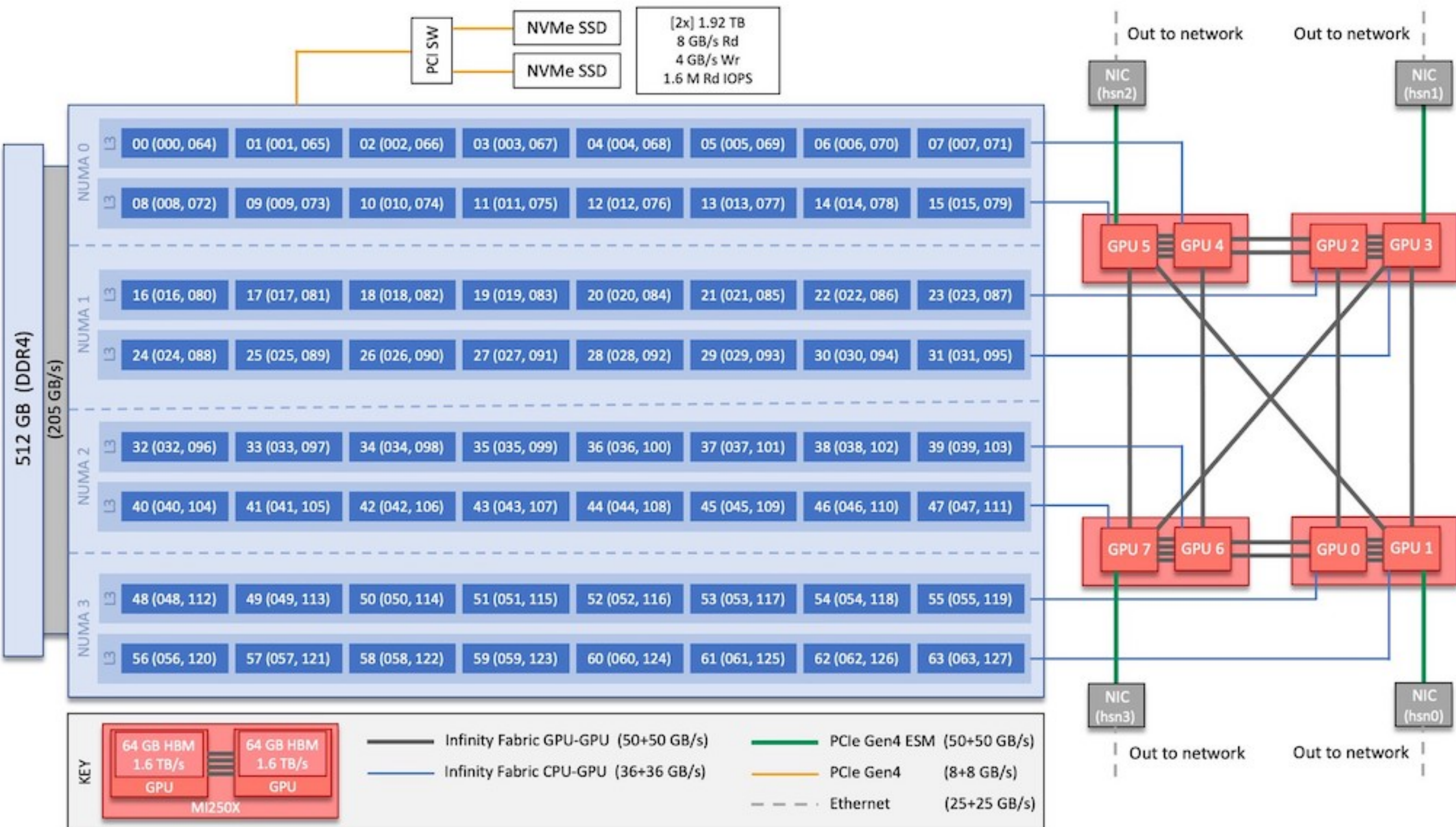
Compute blade

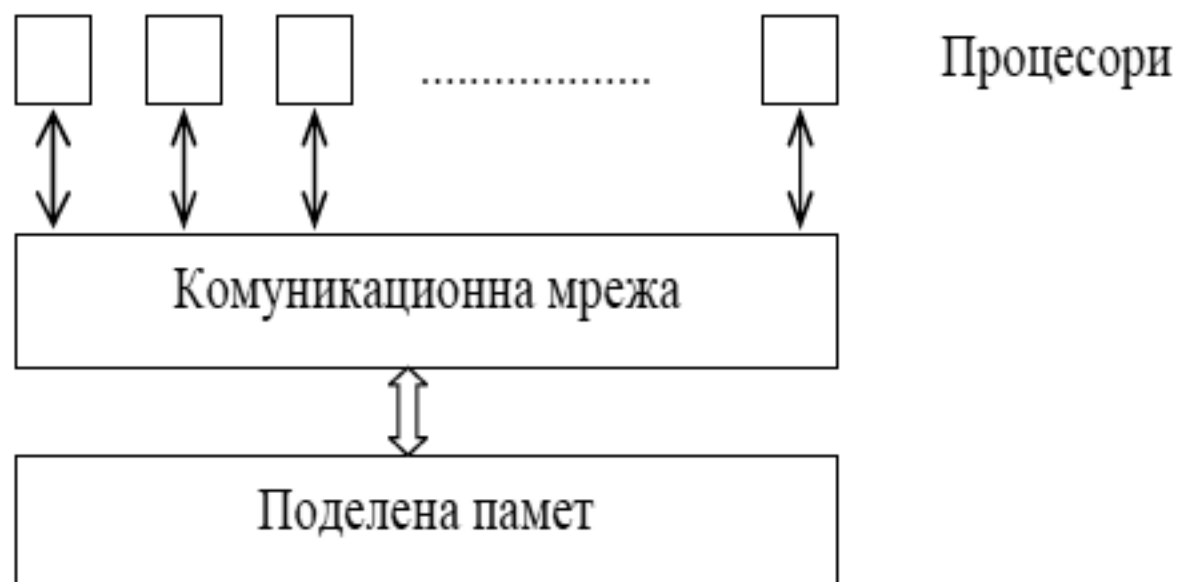
- 2 AMD nodes



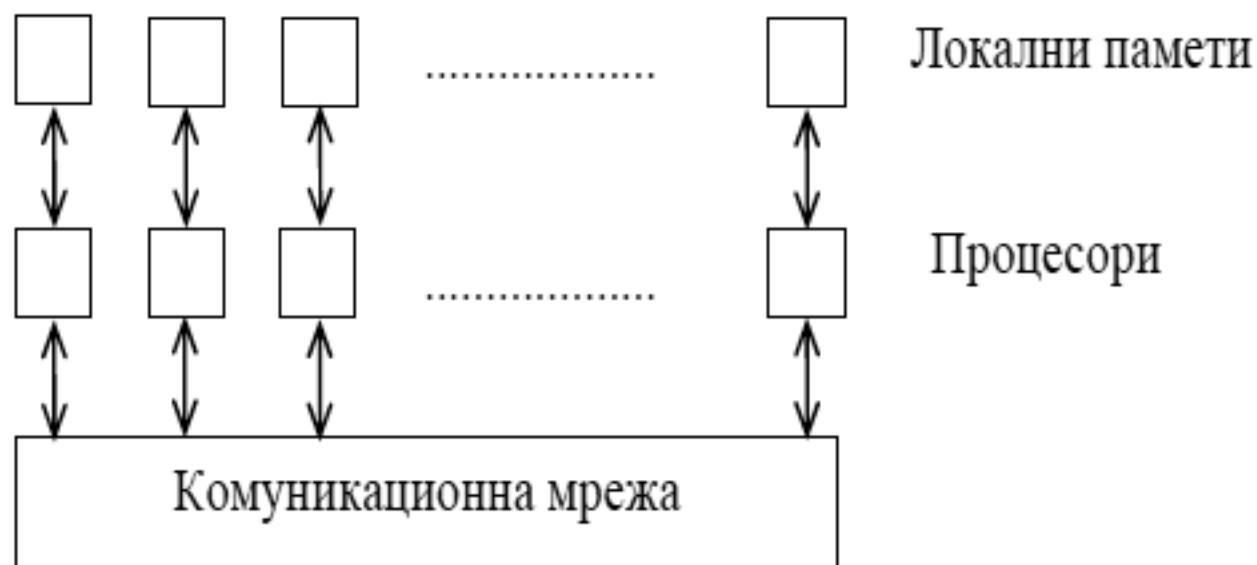
All water cooled, even DIMMS and NICs







а) Обобщена структура на SMP компютър



б) Обобщена структура на MPP компютър



а



б

Рис. 14.1. Организация симметричной мультипроцессорной системы:
 а — с локальной кэш-памятью; б — с разделяемой кэш-памятью

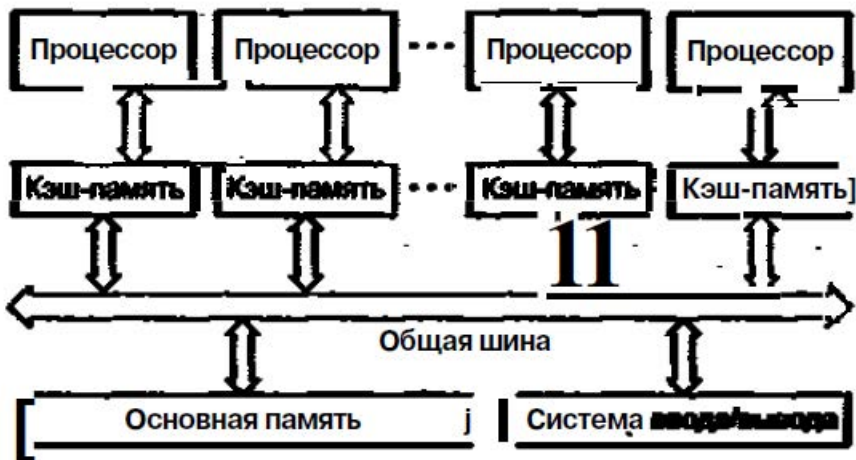


Рис. 1 4 . 5 . Структура SMP-системы с общей шиной

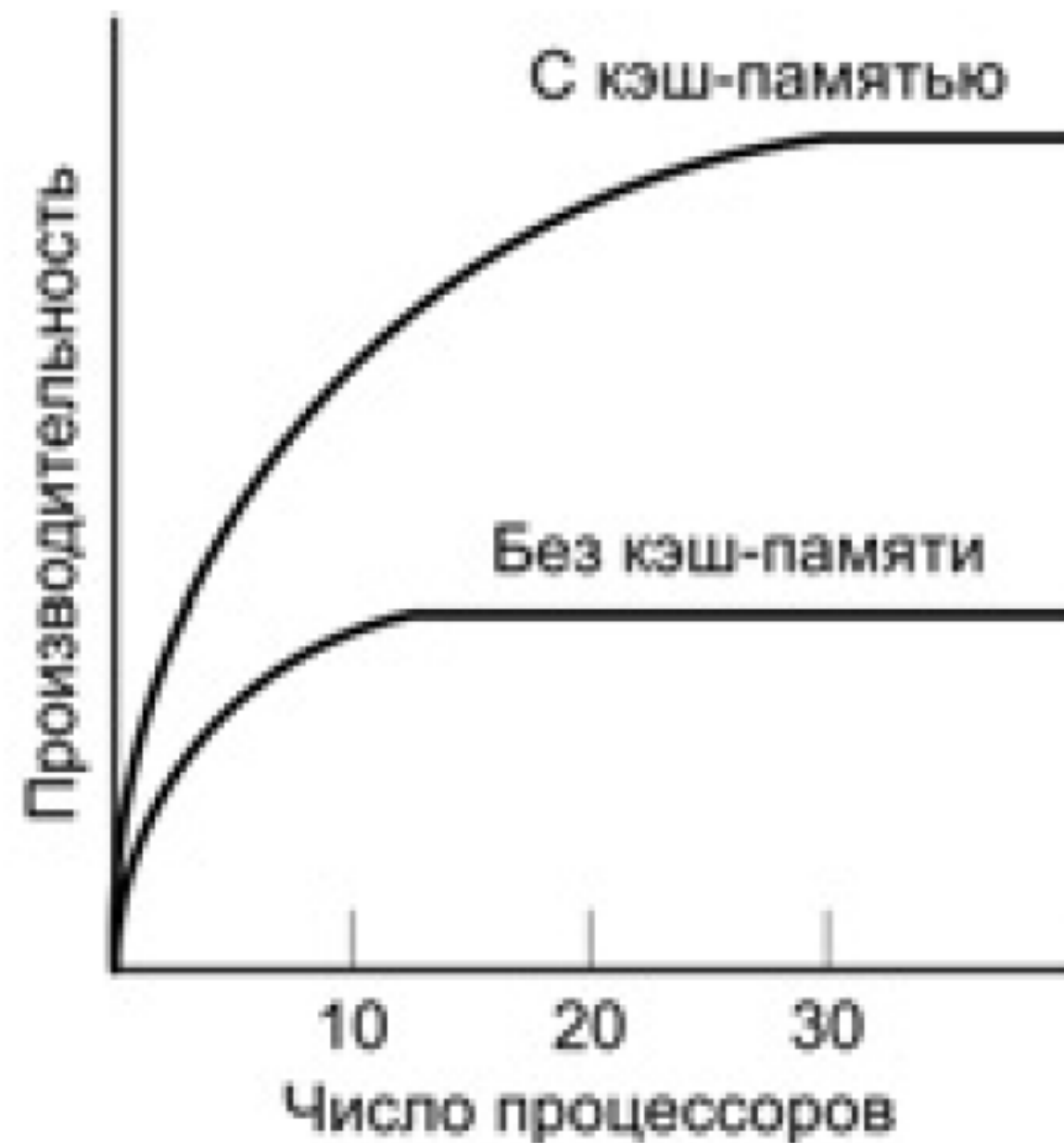


Рис. 11.3. Производительность UMA-системы на базе шины, как функция от числа процессоров на шине

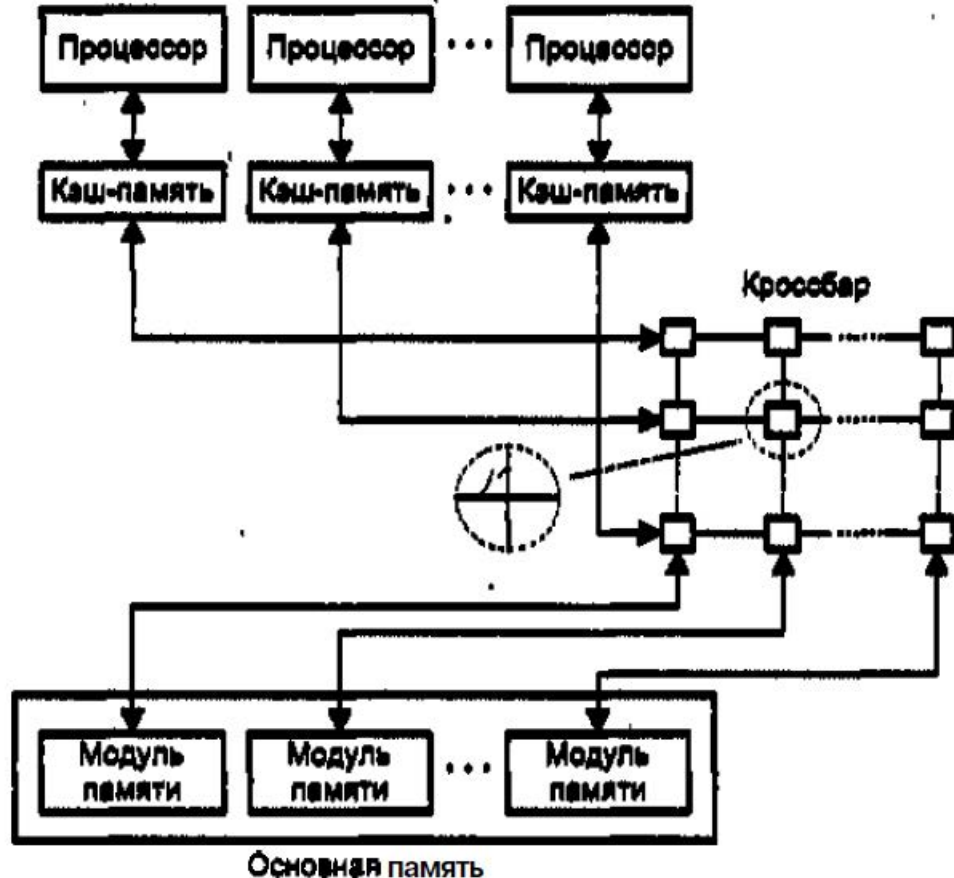


Рис. 14.7. Структура SMP-системы с коммутатором типа «кроссбар»

Коммутатор обеспечивает множественность путей между процессорами и банками памяти, причем топология связей может быть как двумерной, так и трехмерной. Результатом становится более высокая полоса пропускания, что позволяет строить SMP-системы, содержащие больше процессоров, чем в случае общей шины. Типичное число процессоров в SMP-системах на базе матричного коммутатора составляет 32 или 64. Отметим, что выигрыш в производительности достигается, лишь когда разные процессоры обращаются к разным банкам памяти.

необходимо да се осигури ефективно взаимодействие между процесорите, т.е. да се въведе обмен на данните, разположени в различните памети. Това се реализира с помощта на по-бавните входно-изходни операции на процесора. Практически единствения способ за програмиране на този клас системи е чрез използване на обмен на съобщения. Известни са две технологии – **PVM** (**P**arallel **V**irtual **M**achine) и **MPI** (**M**essage-**P**assing **I**nterface), чието приложение не винаги е просто. Благодарение на големия брой съвместно работещи процесори е възможно да се реализират системи с масов паралелизъм и да се осигури мащабируемост на архитектурата. Същевременно компютърните архитектури с разпределена памет не позволяват да се изпълняват съществуващите програми с висока скорост, а преработката на програмите отнема време и струва скъпо. Много често е необходимо да се разработят нови паралелни алгоритми.

Компютрите произвеждани от такива компании като Intel Scientific Computers – **iPSC** (**I**ntel **P**ersonal **S**uper **C**omputer), IBM (напр. Scalable Power Parallel System), NCUBE/ten NCUBE Corporation, компютрите Connection Machine на фирмата Thinking Machines и др. са представители на този клас паралелни компютри.

Програмирането става обикновено чрез написването на отделни програми за всеки процесор. Програмите се свързват чрез операциите от ниско ниво, които предоставя операционната система и са достъпни за програмиста във вид на разширение на последователните езици от типа на FORTRAN, Pascal или C. Като правило, програмите във възлите са копия на една и съща програма. Отделните копия ще се изпълняват правилно, независимо от тяхното разположение в мрежата. Такъв стил на програмиране е известен като **SCMD** (**S**ingle **C**ode **M**ultiple **D**ata) или **SPMD** (**S**ingle **P**rogram **M**ultiple **D**ata).

Още един проблем, който е свързан както с приложната задача (и по-точно с разработването на подходящ алгоритъм), така и с работата на операционната система, е осигуряването на балансирано натоварване. Под балансирано натоварване трябва да се разбира равномерното разпределение на задачата между процесорите, така че да се осигури работа на всички процесори, или на по-голямата част от тях, през цялото времетраене на решението на задачата. Неизпълнението на това изискване или лошото му изпълнение води до по-голям брой комуникации и от тук до снижаване на производителността.

Обобщенная структура МРР-системы показана на рис. 14.5.



Рис. 14.5. Структура вычислительной системы с массовой параллельной обработкой

Обобщенная структура кластерной вычислительной системы показана на рис. 14.6.

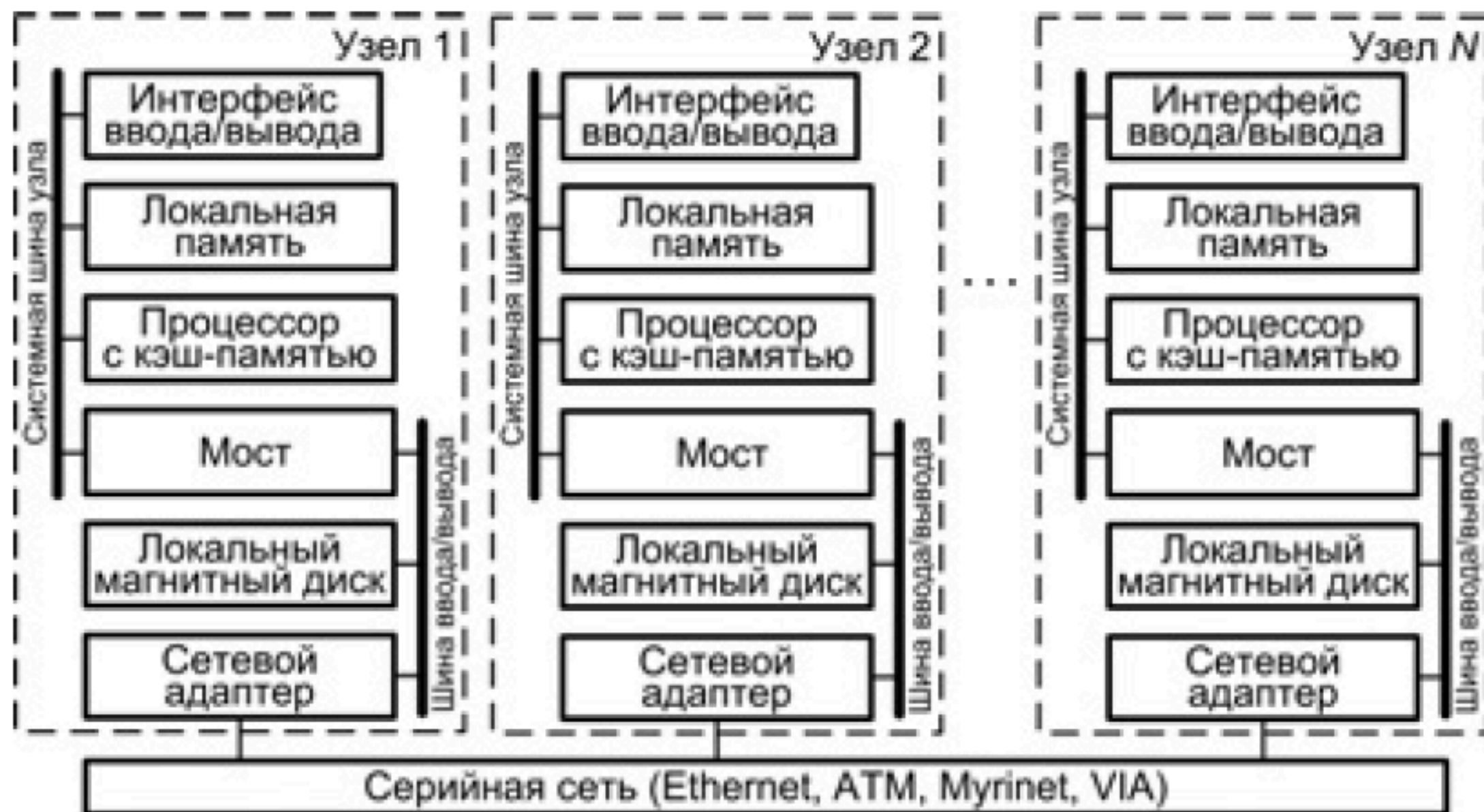
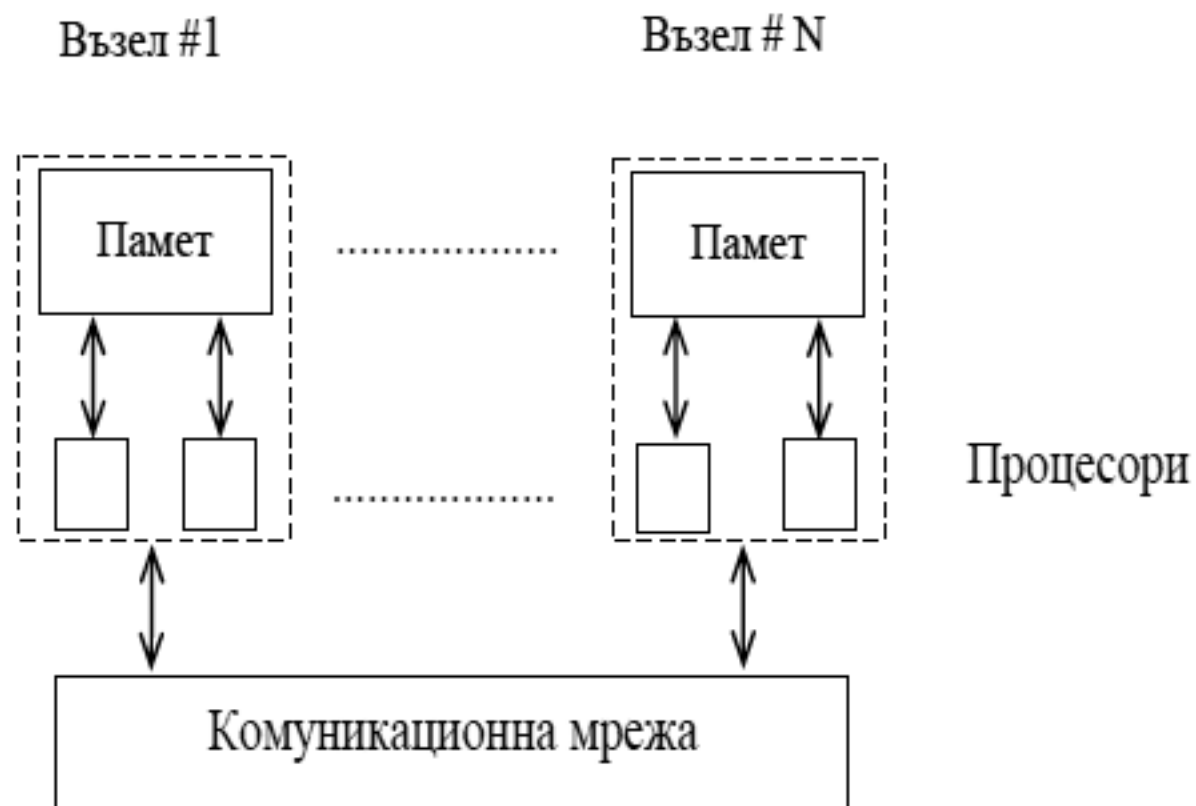


Рис. 14.6. Структура кластерной системы

Броят на процесорите в сървърите с **SMP** архитектура е от порядъка на 8-16, докато **NUMA** архитектурата обединява 256 и повече процесори.

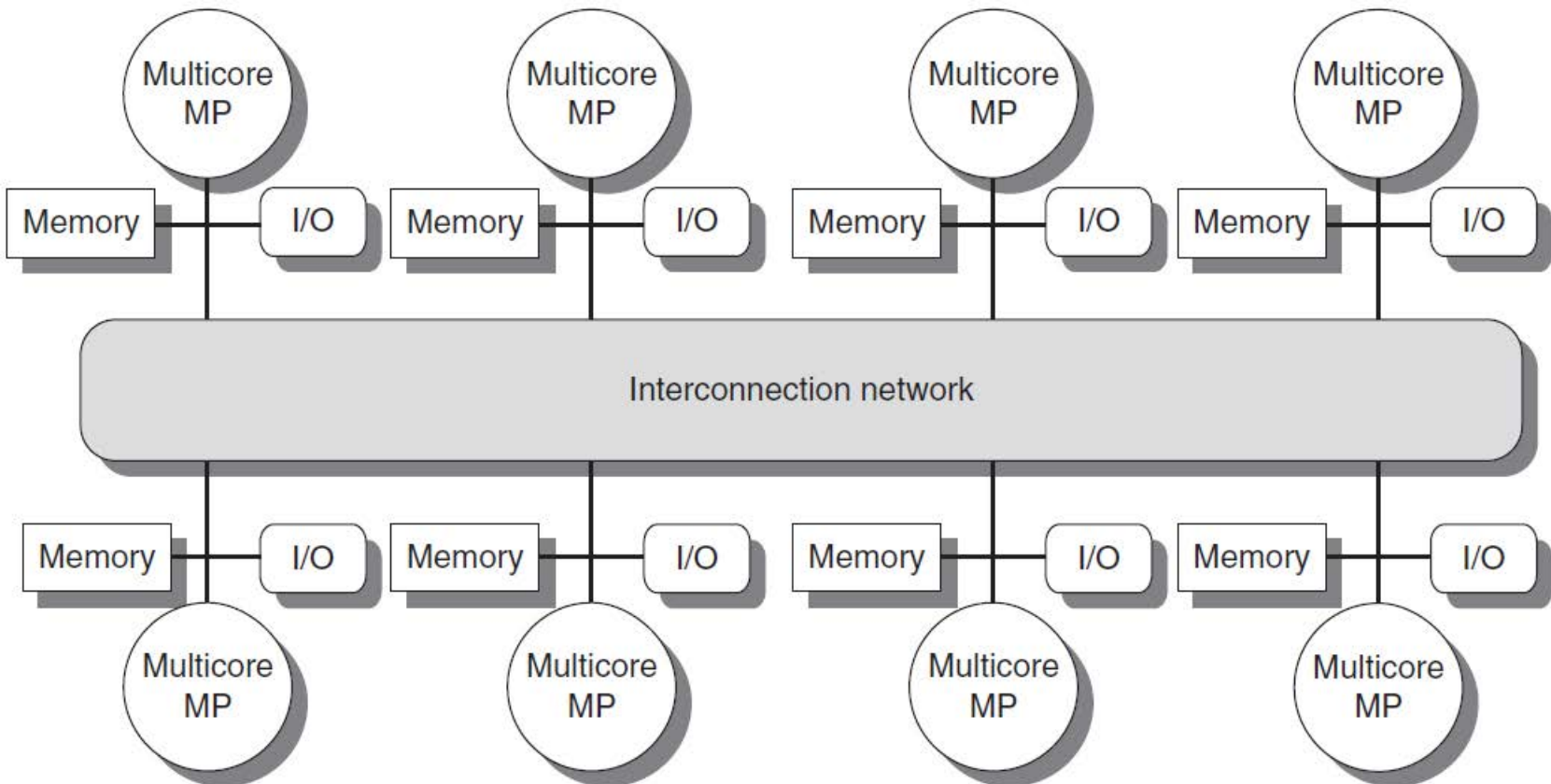
В групата компютри с разпределена памет, освен традиционните компютри, посочени по-горе, се включват и кластерните системи. В компютърната литература значението "кластер" се употребява в различен аспект. В частност, "кластерната" технология се използва за повишаване на скоростта и надеждността на сървърите за база данни и Web-сървърите. Тука ние ще обсъждаме само кластерите, ориентирани за решаване на задача от изчислителен характер. В този смисъл кластер означава съвкупност от компютри, обединени в някаква мрежа за решаване на една задача. В качеството на изчислителни възли обикновено се използват достъпни на пазара еднопроцесорни компютри, дву- или четири- процесорни **SMP** сървъри. Всеки възел работи под управлението на свое копие на операционната система. Състава и мощта на всеки възел може да се мени, което дава възможност за създаване на нееднородни системи. Към тази група паралелни компютри спадат проекта Beowulf създаден в NASA, проекта KLAT и KLAT2 [10], а също така и компютрите на SGI от серията Origin и др.



Фиг. 7-2. Обобщена структура на компютър с NUMA архитектура

От тази особеност и произтича названието на този клас компютри – компютри с нееднакъв достъп до паметта. В този смисъл, класическите **SMP** компютри са с архитектура **UMA** (**U**niform **M**emory **A**ccess), осигуряващи еднакъв достъп за произволен процесор.

Както и за **SMP** компютрите, и за този клас компютри е характерно несъгласуваност на данните на ниво кеш памет. За решаването на този проблем е разработена специална модификация на **NUMA** архитектурата – **ccNUMA** (**c**ache **c**oherent **NUMA**). За целта са разработени множество протоколи, съгласуващи съдържанието на всички кеш памети и програмиста не се грижи за този проблем.



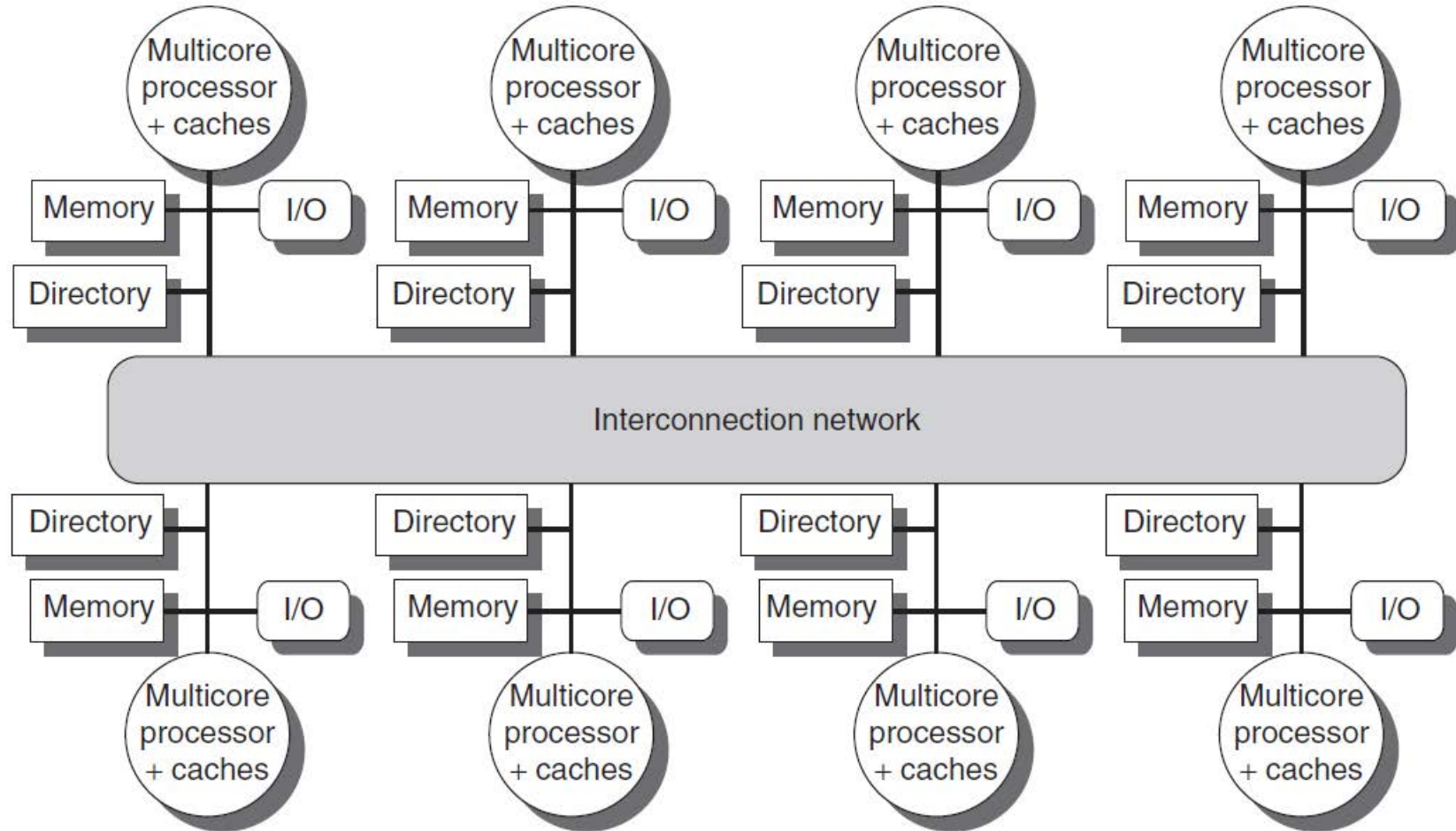


Figure 5.20 A directory is added to each node to implement cache coherence in a distributed-memory multiprocessor. In this case, a node is shown as a single multicore chip, and the directory information for the associated memory may reside either on or off the multicore. Each directory is responsible for tracking the caches that share the memory addresses of the portion of memory in the node. The coherence mechanism would handle both the maintenance of the directory information and any coherence actions needed within the multicore node.

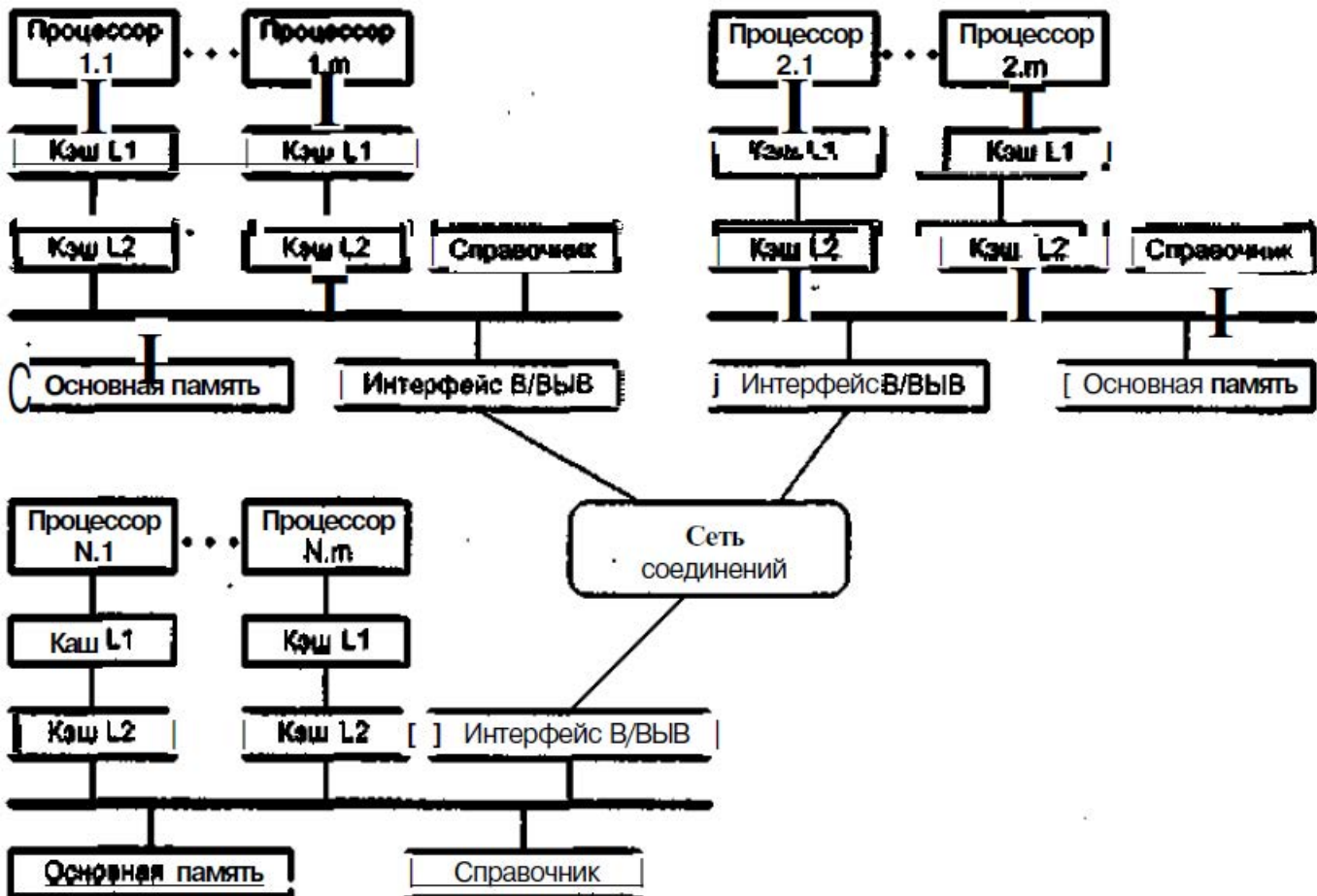


Рис. 14.15. Организация систем типа CC-NUMA

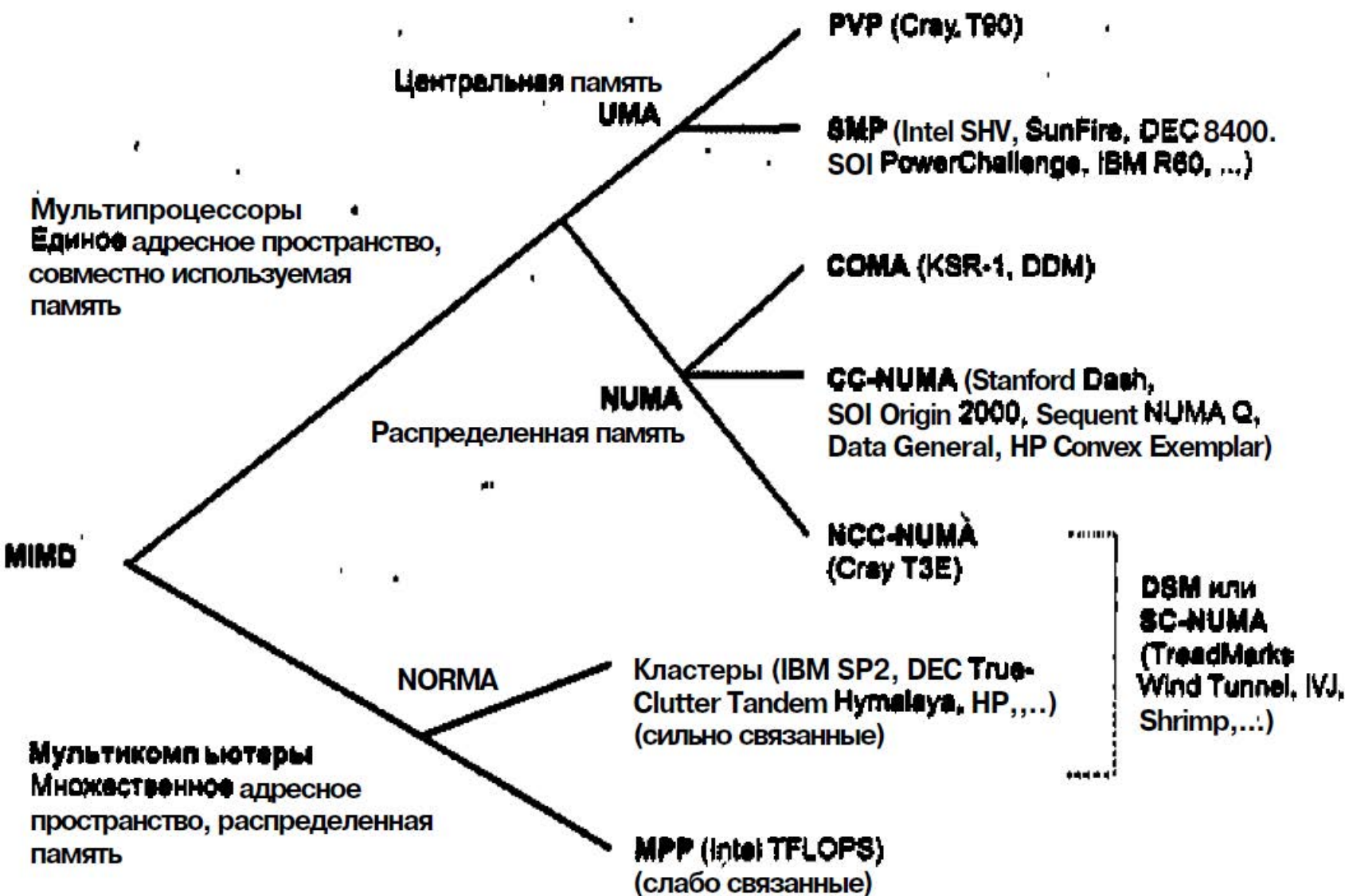


Рис. 11.2. Классификация моделей архитектур памяти вычислительных систем

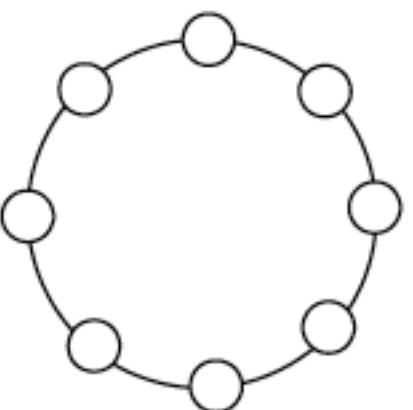
На рис. 11.2 приведена классификация таких моделей, применяемых в вычислительных системах класса MIMD (верна и для класса SIMD).

Съкращения

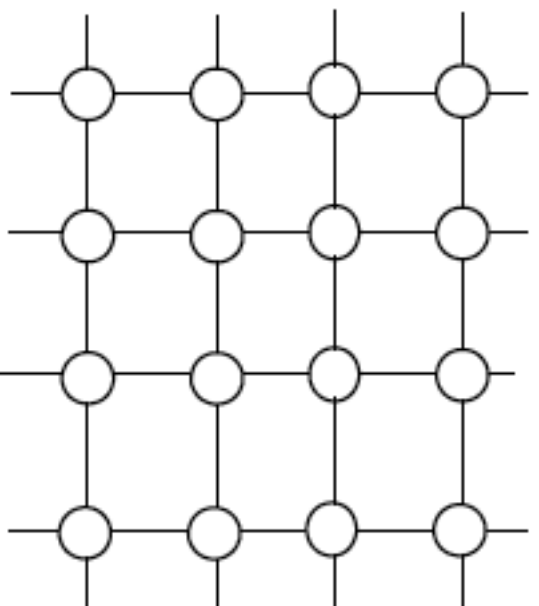
- NUMA = Non-Uniform Memory Access
- CC-NUMA = Cache-Coherent NUMA
- NCC-NUMA = No Cache-Coherent NUMA
- SC-NUMA = Software-Coherent NUMA
- DSM = Distributed Shared Memory
- COMA = Cache-Only Memory Architecture
- NORMA = No Remote Memory Access
- UMA = Uniform Memory Access
- SMP = Symmetrical Multi-Processing
- PVP = Parallel Vector Processing



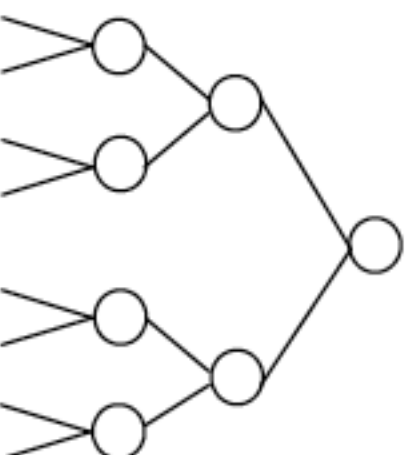
а) линия (конвейер)



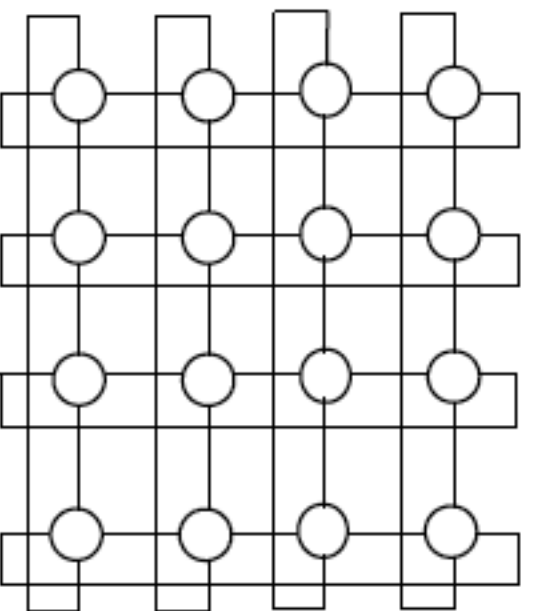
б) кръг



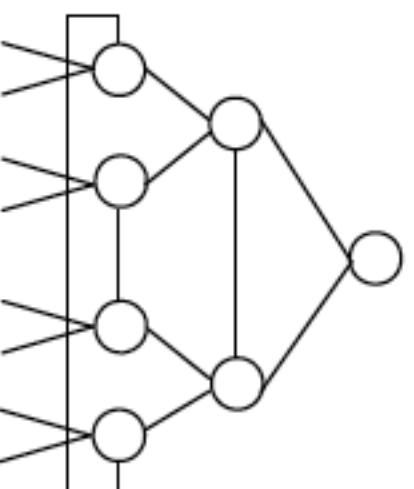
в) двумерна решетка



г) двоично дърво



д) топ



е) хипердърво от тип I

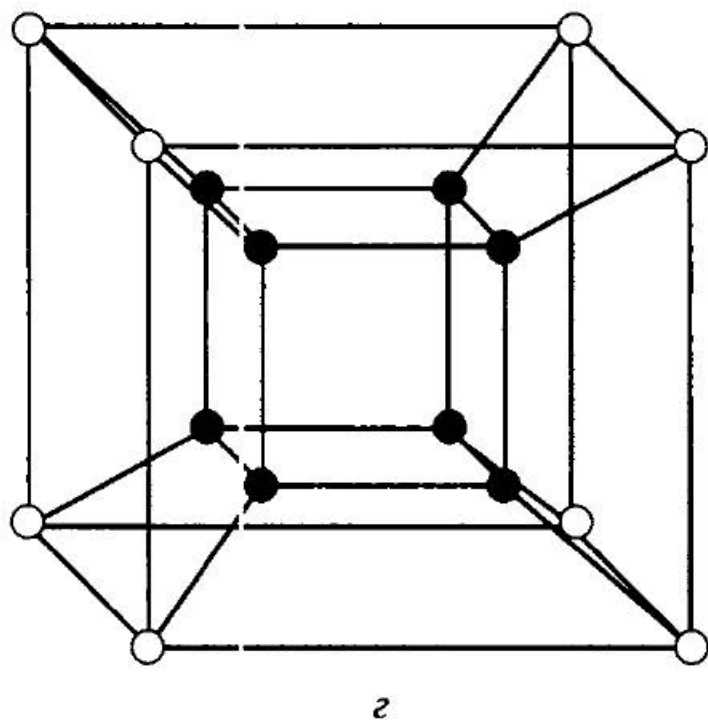
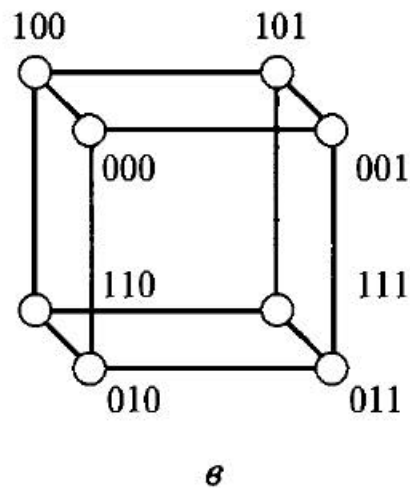
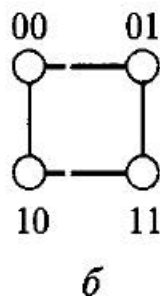
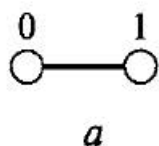


Рис. 3.2. Гиперкубические структуры ВС:

a — 1D-куб ($N = 2$, $n = 1$); *б* — 2D-куб ($N = 4$, $n = 2$); *в* — 3D-куб ($N = 8$, $n = 3$); *г* — 4D-куб ($N = 16$, $n = 4$)

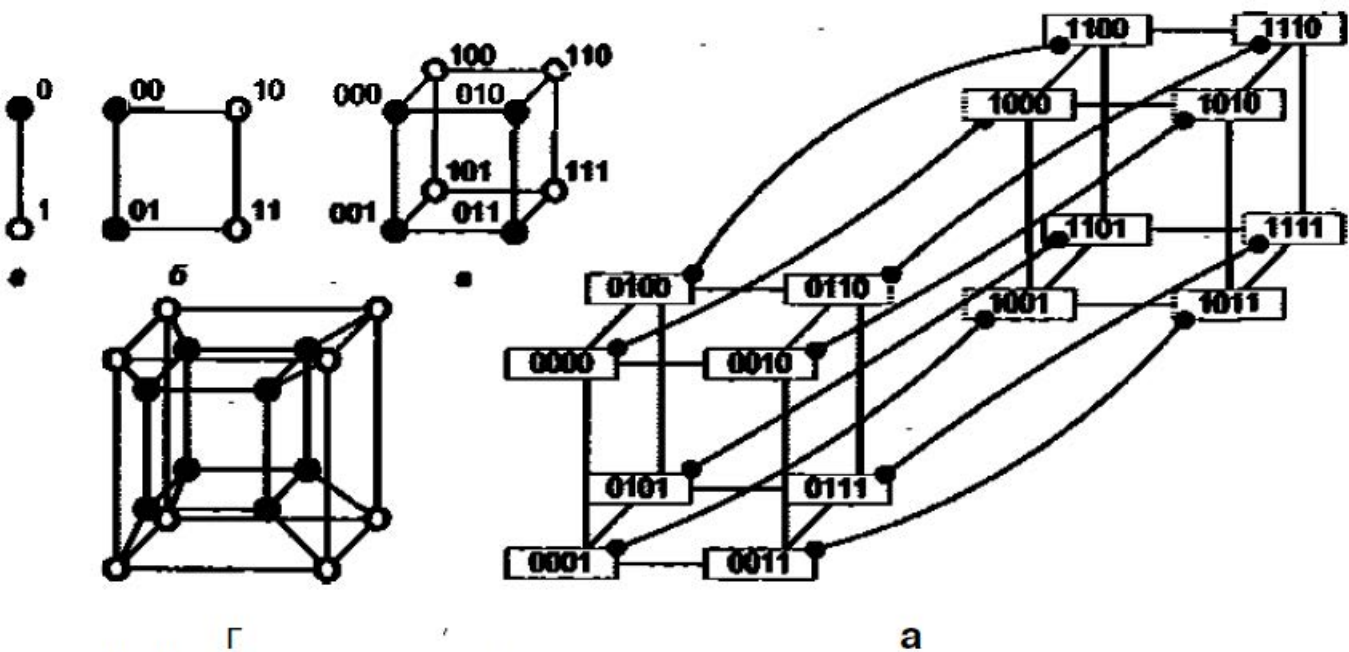


Рис. 12.15. Топология гиперкуба: а - одномерная; б - двумерная; в - трехмерная; г - четырехмерная; д - схема формирования четырехмерного гиперкуба из двух трехмерных

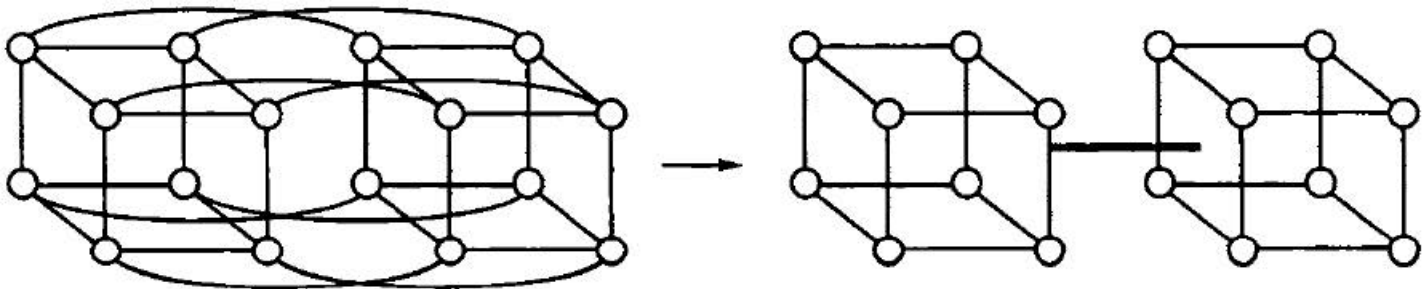


Рис. 5.7. 4D-куб:

—— гиперребро

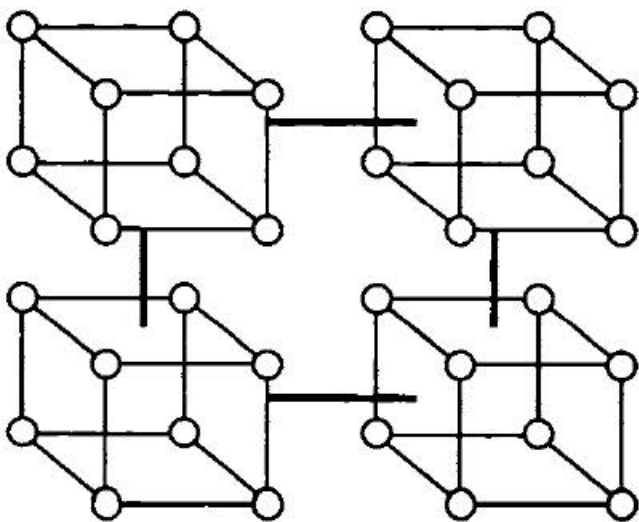


Рис. 5.8. 5D-куб

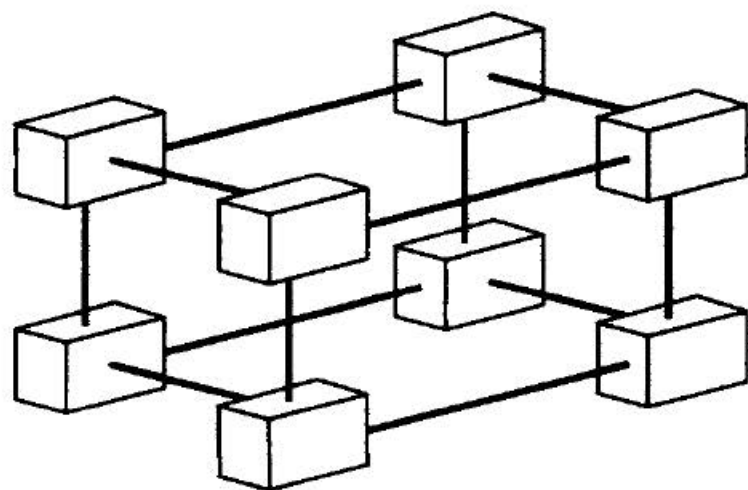


Рис. 5.9. 6D-куб

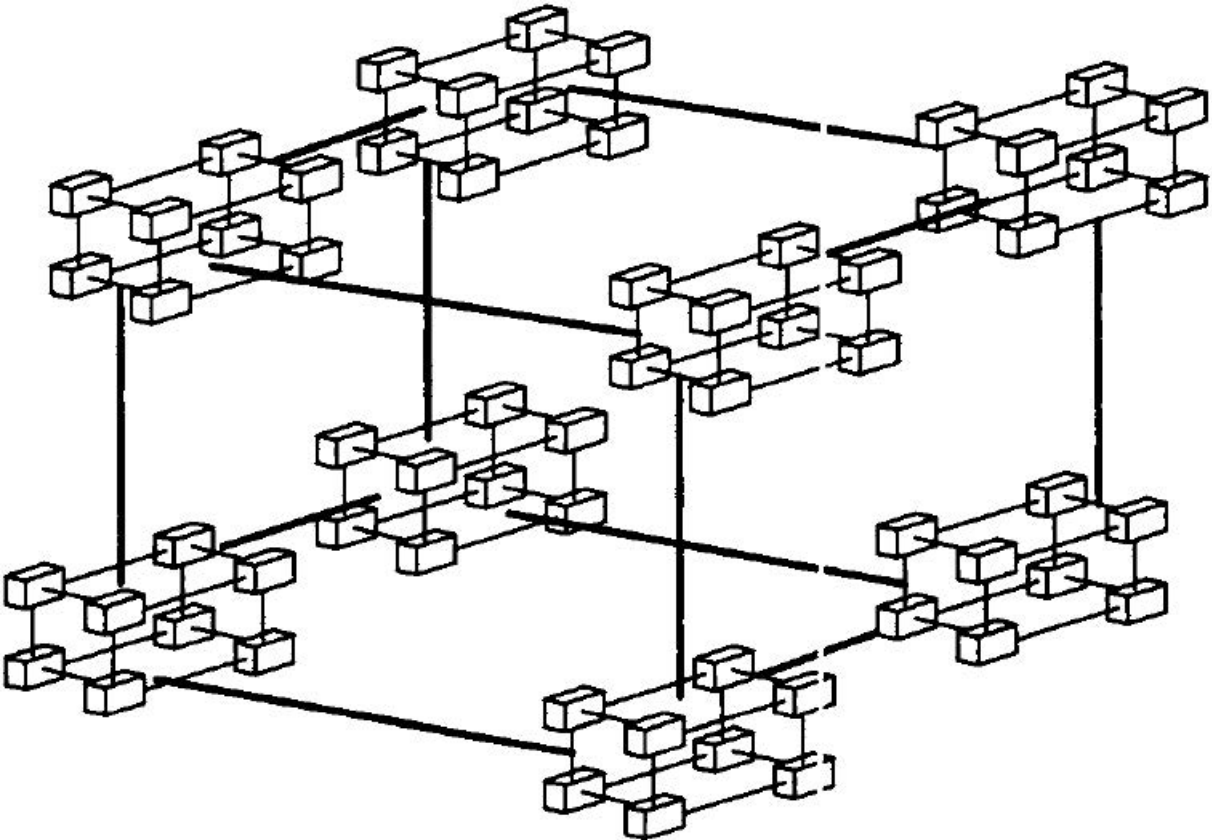
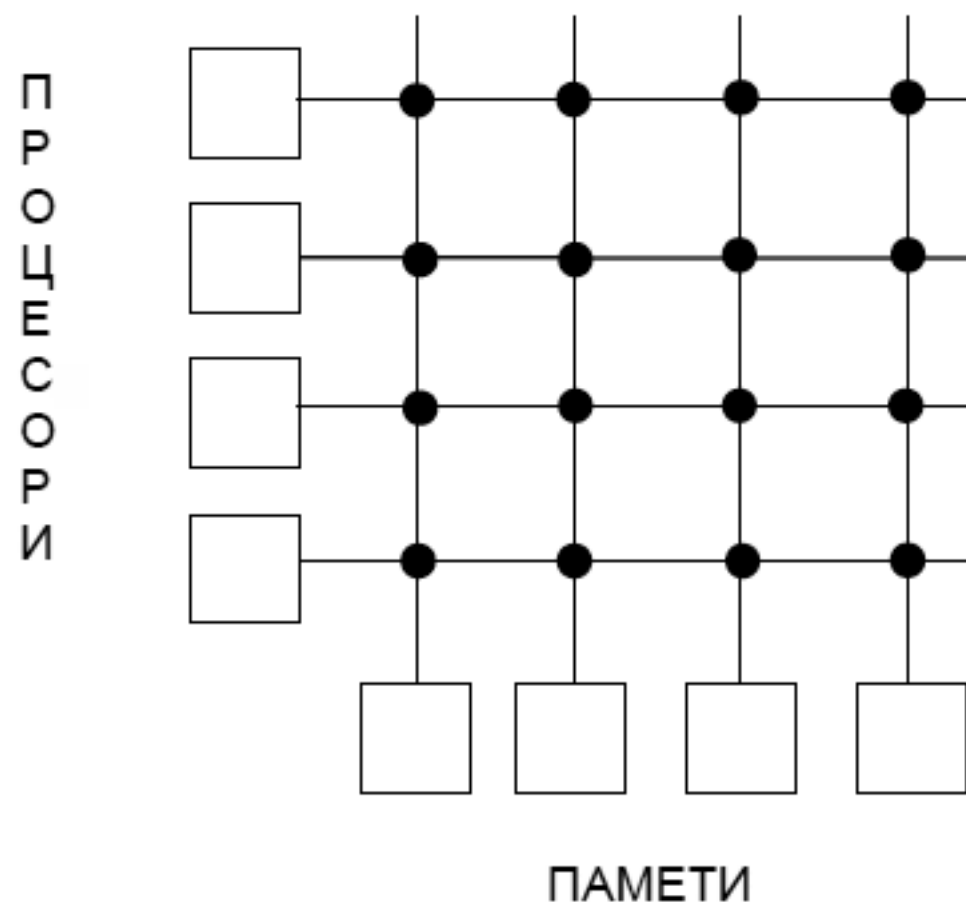
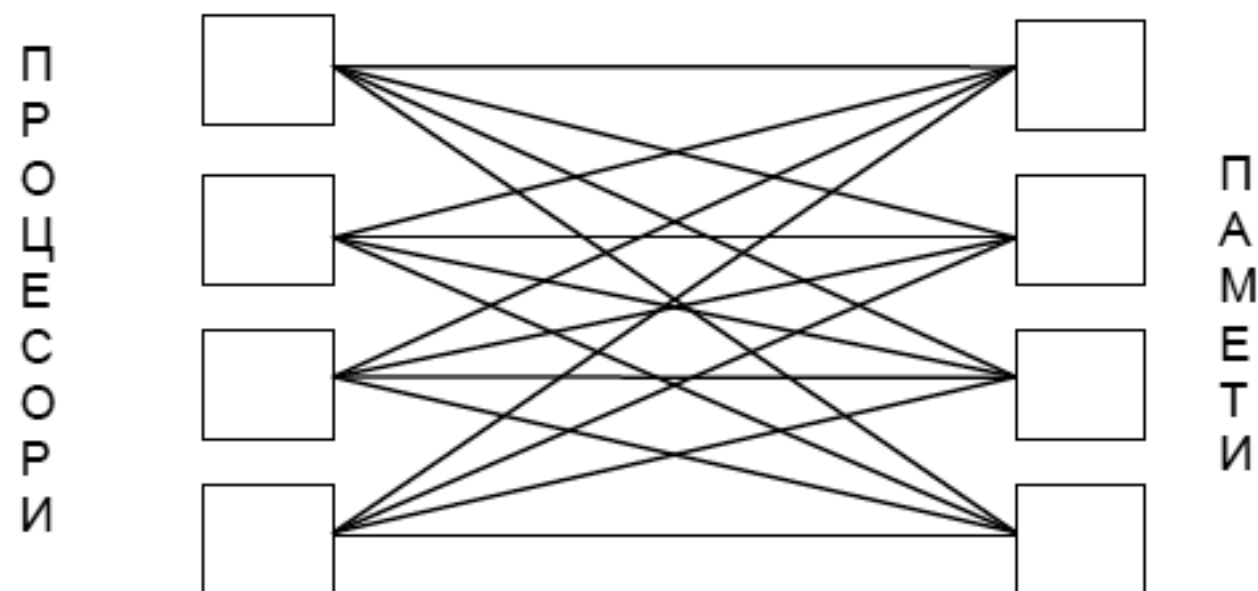


Рис. 5.10. 9D-куб



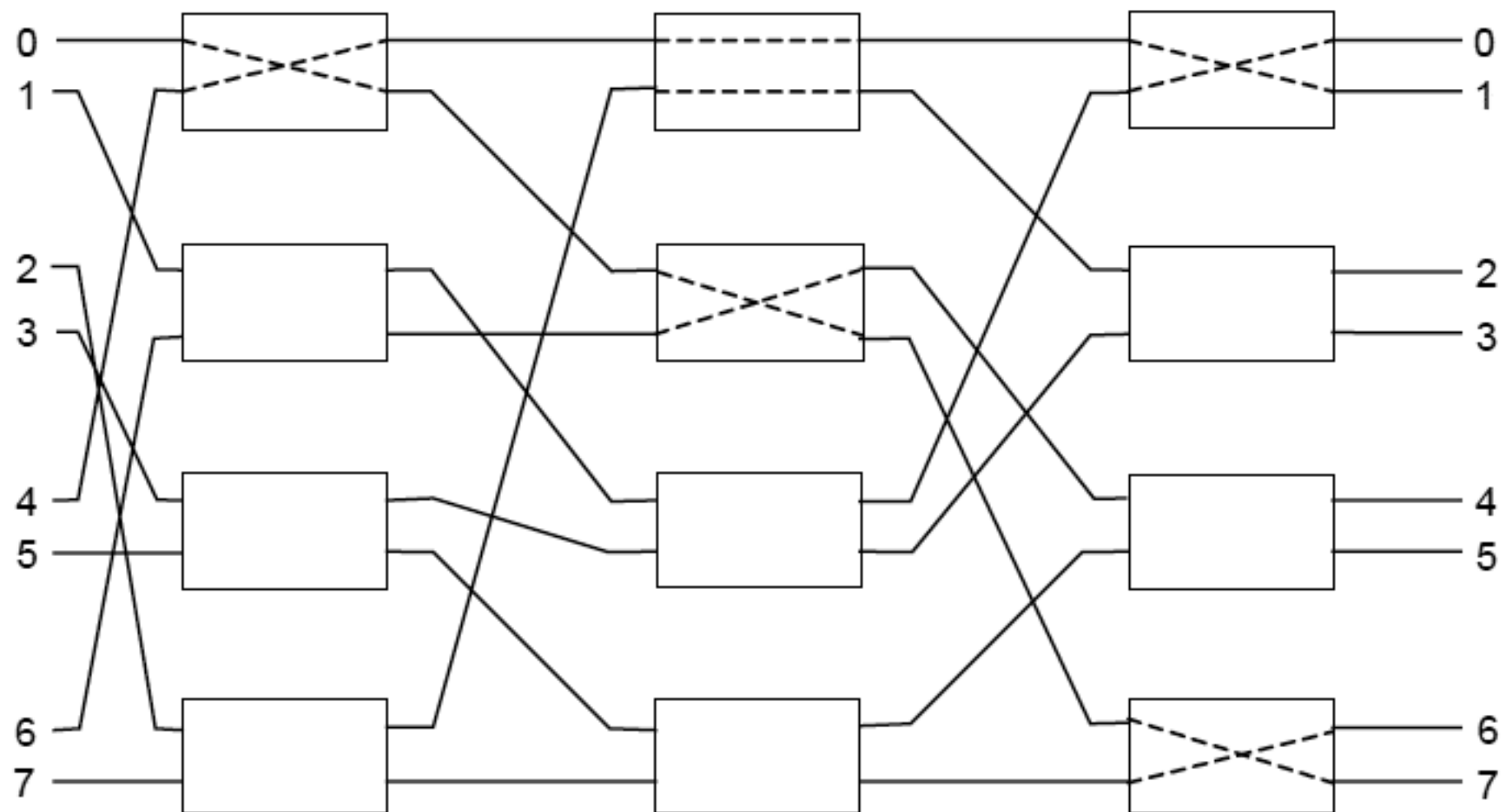
а) Матричен превключвател



б) Пълно свързана мрежа

Фиг. 8-1. Типични представители на КМ по пространство:

Състоянието на КЕ се задава от секцията за управление на динамичната реконфигурация в рамките на локалния или глобалния контролер в зависимост от приложената стратегия на управление – децентрализирана или централизирана. Последният се управлява от предварително изчислен **етикет**, най-често определен по формулата $T=X\oplus Y$, където X и Y са съответно адреси на предавателя и приемника. На фиг. **8-5** е дадена delta мрежата за $N=8$.



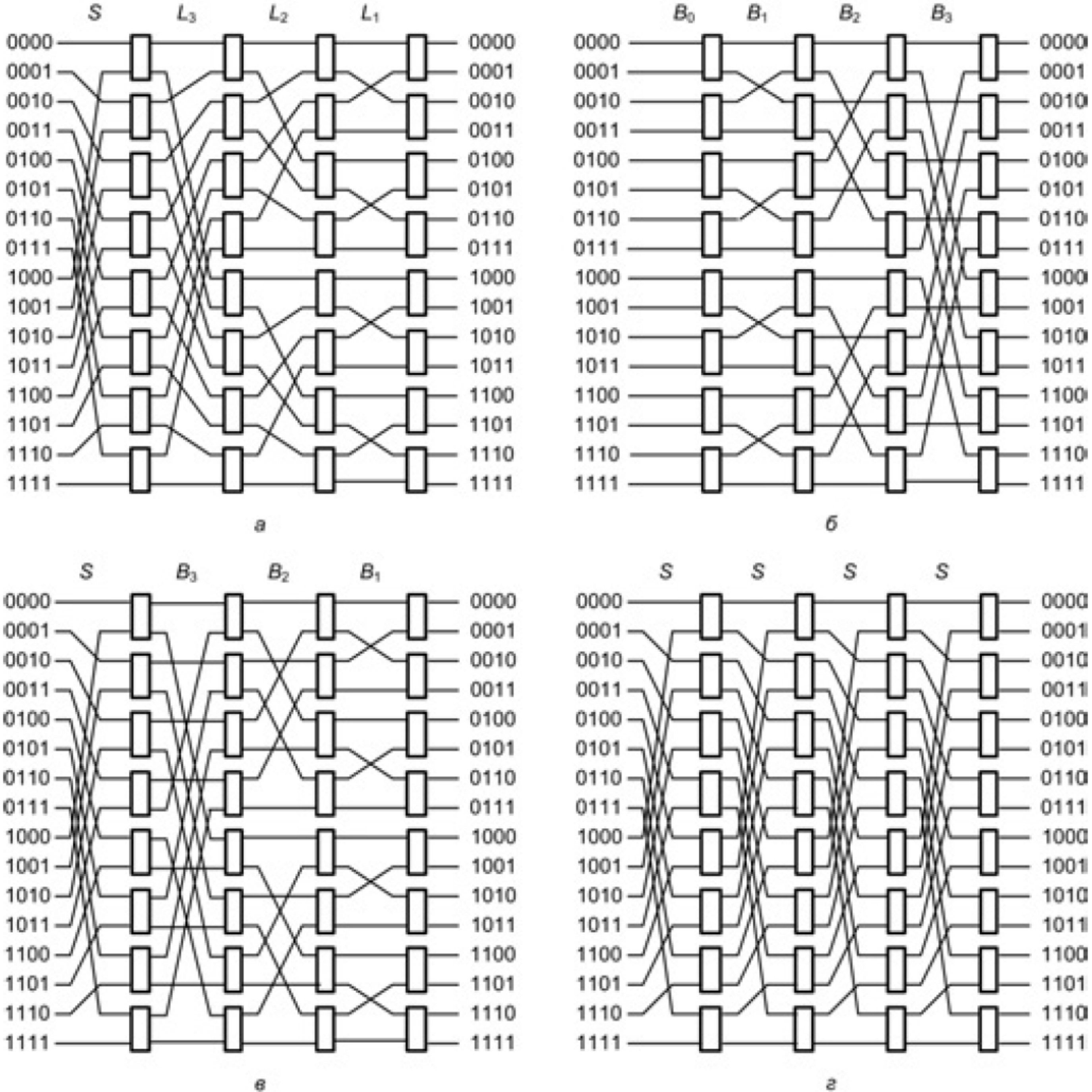
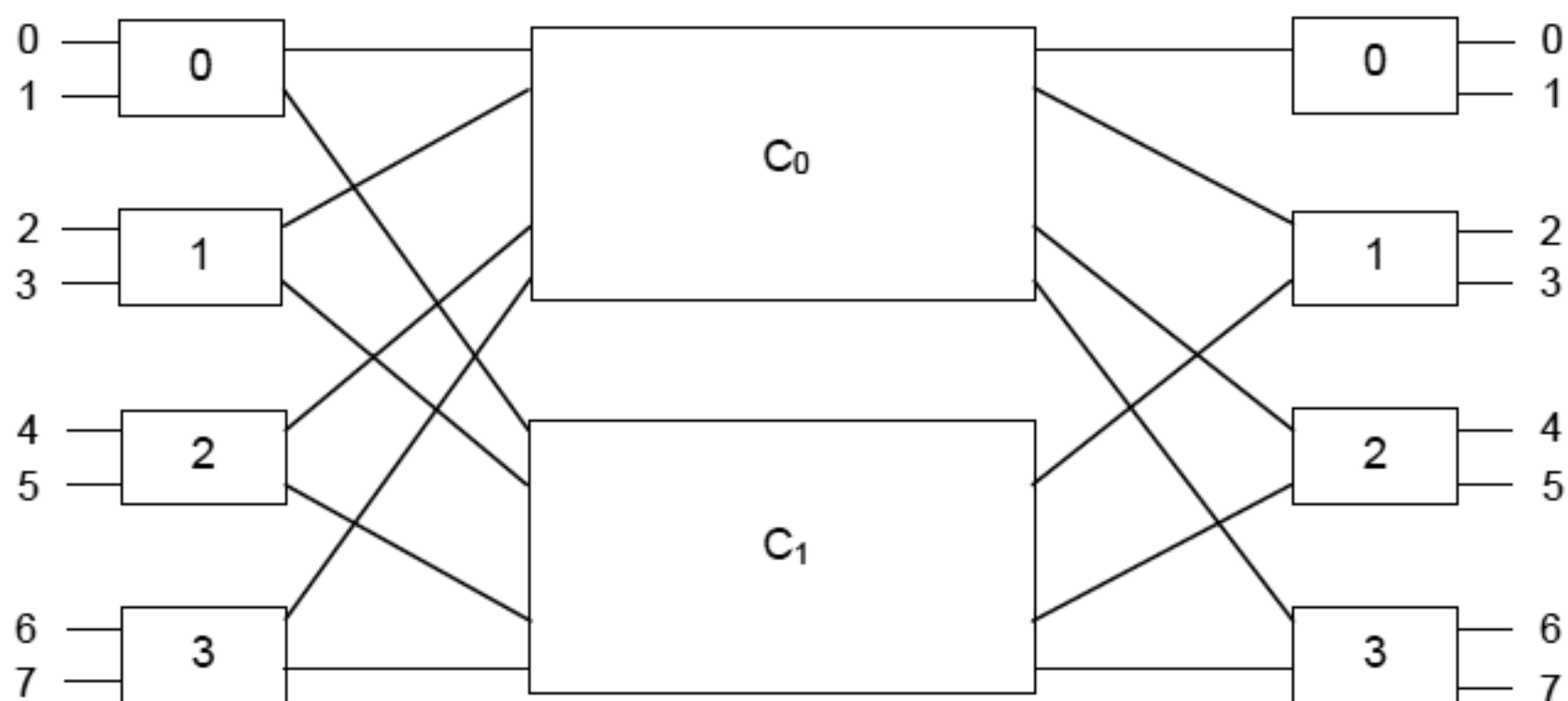


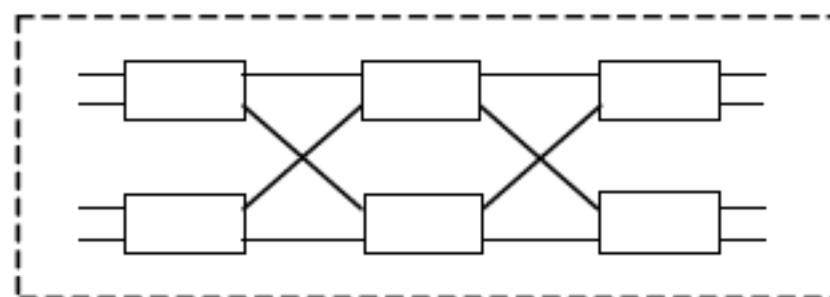
Рис. 12.25. Примеры сетей класса Дельта размерности 16×16 с топологией:
 а — базисной линии; б — баттерфляй; в — обобщенного куба; г — Омега

5.4. Мрежа на Бенес

Мрежата на Бенес с N входа и N изхода има симетрична структура – фиг.8-10, във всяка половина на която между $N/2$ входа и $N/2$ изхода на КЕ е разположена също такава мрежа на Бенес, но с $N/2$ входа и $N/2$ изхода. КЕ имат две работни състояния Π_3 и Π_6 ; едната линия на КЕ се свързва към горната част на мрежата на Бенес, а другата – към долната част. Броят на стъпалата е равен на $2\log_2 N - 1$, а общия брой на КЕ в мрежата е $N(2\log_2 N - 1)/2$.



C_0, C_1



Seventy Two Core Processor SoC with 8x 10Gb Ethernet Ports, PCIe and Networking Offloads

The TILE-Gx72™ Processor is optimized for intelligent networking, multimedia and cloud applications, and delivers remarkable computing and I/O with complete “system-on-a-chip” features. The device includes 72 identical processor cores (tiles) interconnected with the iMesh™ on-chip network. Each tile consists of a full-featured, 64-bit processor core as well as L1 and L2 cache and a non-blocking Terabit/sec switch that connects the tiles to the mesh and provides full cache coherence among all the cores.

Powerful Processor Cores

- Seventy-two cores operating at frequencies up to 1.2 GHz
- 64-bit architecture (datapath and address)
- Three execution pipelines
- Robust virtual memory system with TLBs, multiple page size support, and Hardwall™ protection
- Instruction Set Architecture (ISA) extensions for multimedia and SIMD processing

Cache

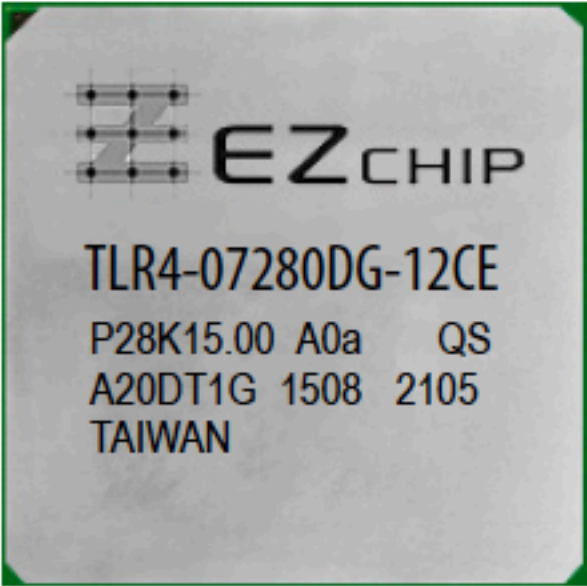
- 23.5 MBytes total on-chip cache
- Dynamic Distributed Cache (DDC™) scalable hardware coherence
- 32 KB L1 instruction cache and 32 KB L1 data cache per core
- 256 KB L2 cache per core
- 18 MBytes coherent L3 cache

iMesh Interconnect

- Five independent low-latency mesh networks
- >110 Tbps aggregate bandwidth
- Non-blocking, cut-through switching with one clock cycle per hop

Integrated Memory Controllers

- Four 72-bit DDR3 controllers with ECC support
- 1 TB total memory capacity
- Speeds up to 1,866 MT/s
- Advanced request reordering



APPLICATIONS

The TILE-Gx72 is ideal for applications such as:

- 80 - 100 Gbps of networking and security dataplane offload
- 80 Gbps packet filtering and bandwidth management
- 40 Gbps SSL/IPsec security protocol processing
- High-performance computing offload across PCIe
- H.264/H.265 high-density video transcoding

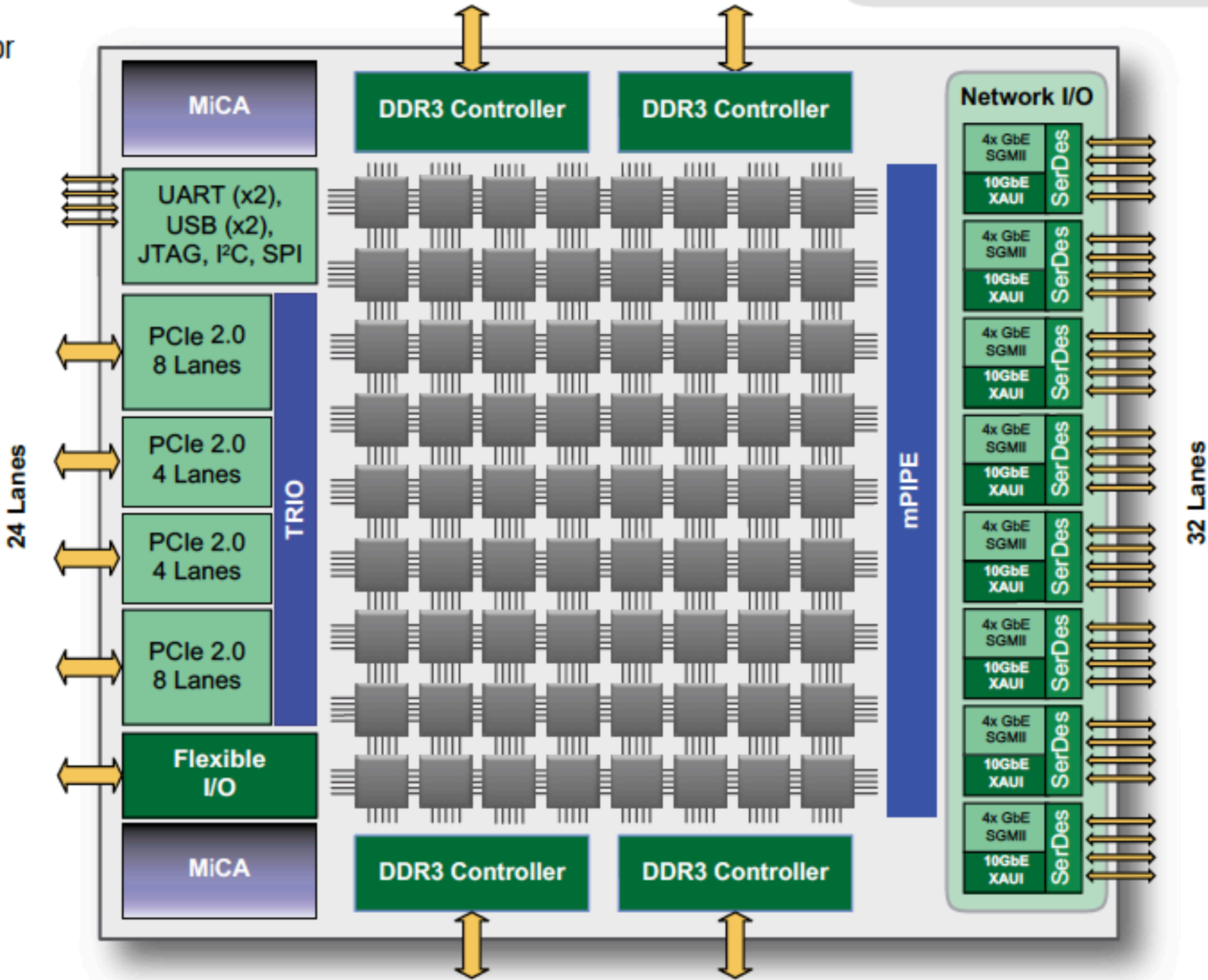
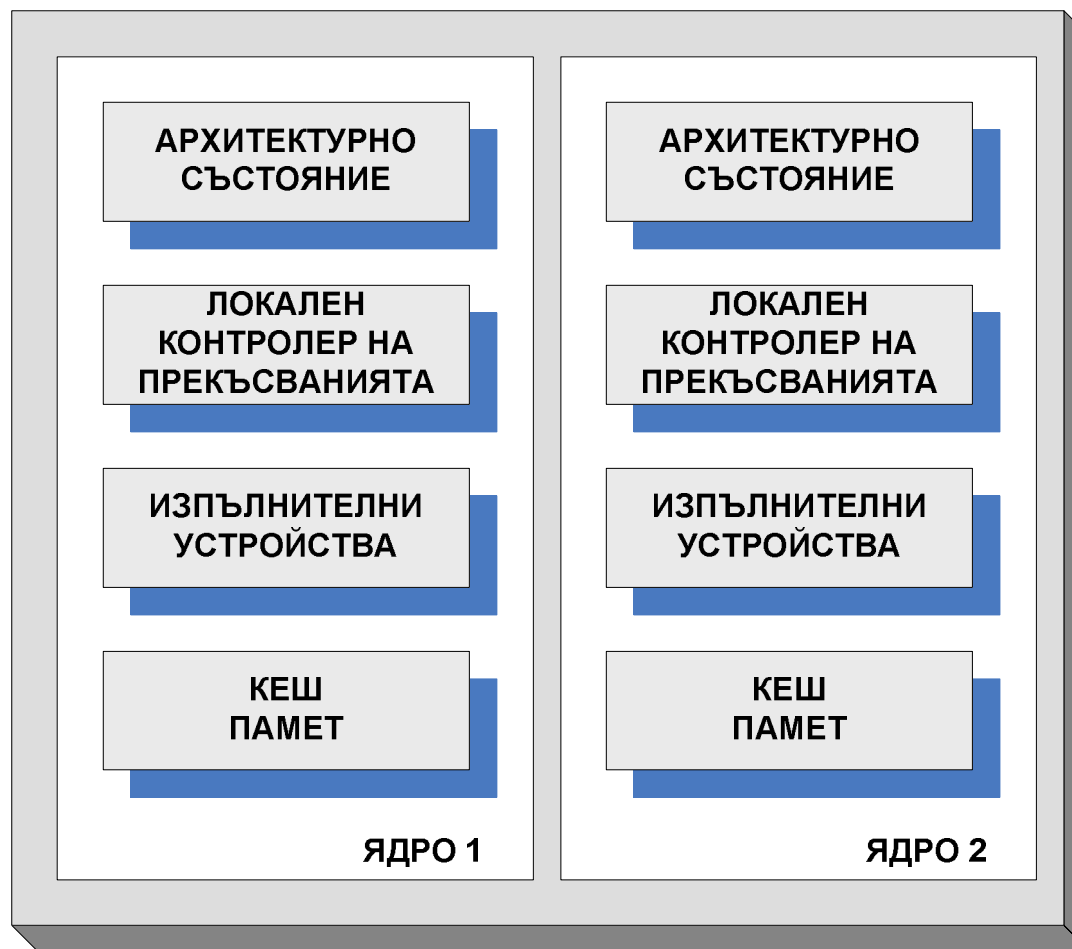


Figure 1. TILE-Gx8072 Processor Block Diagram

Архитектурата на двуядрен процесор с отделни кеш-памети

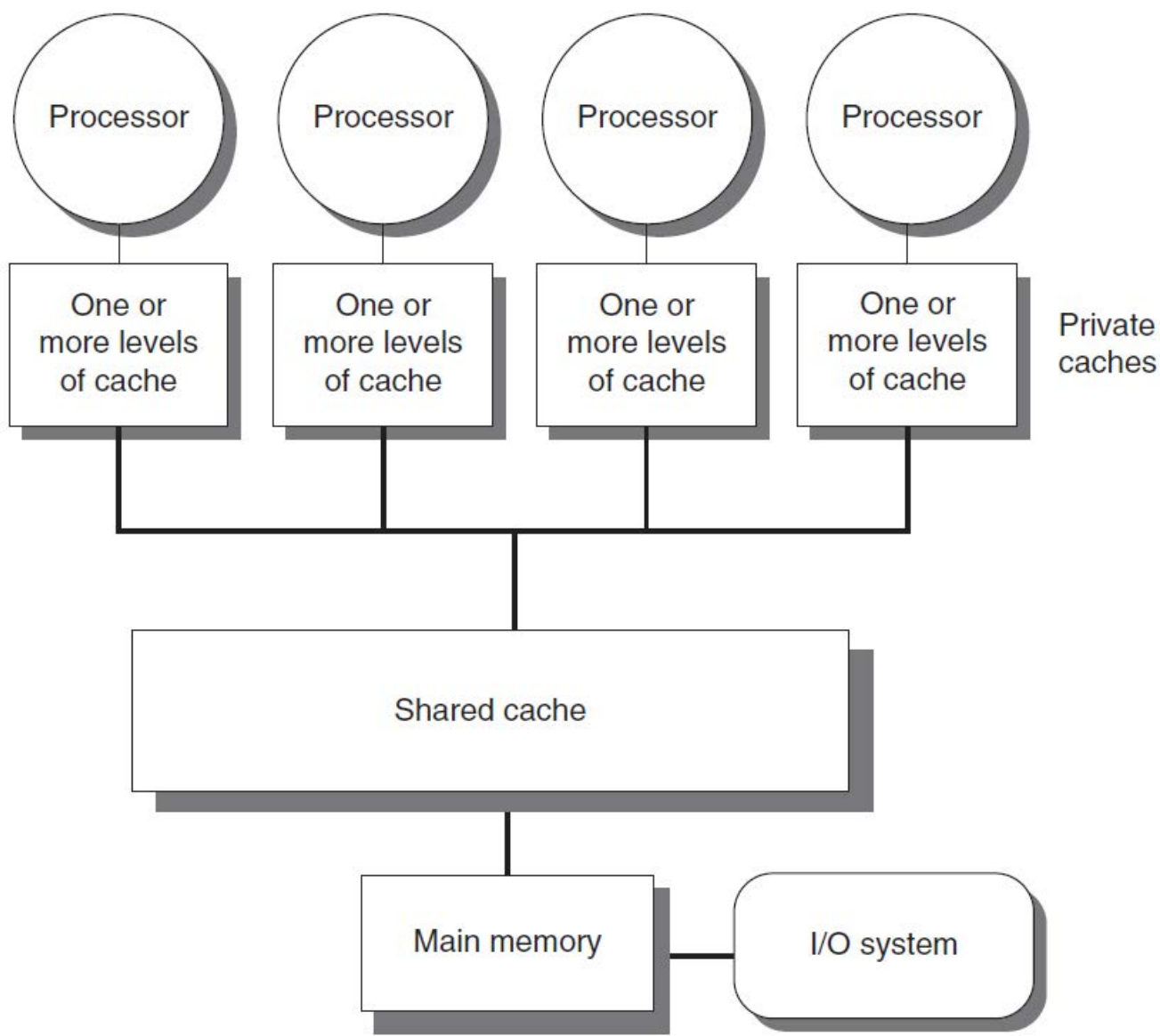


Архитектурата на двуядрен процесор с обща кеш-памет



Архитектурата на двуядрен процесор с хипернишкова технология





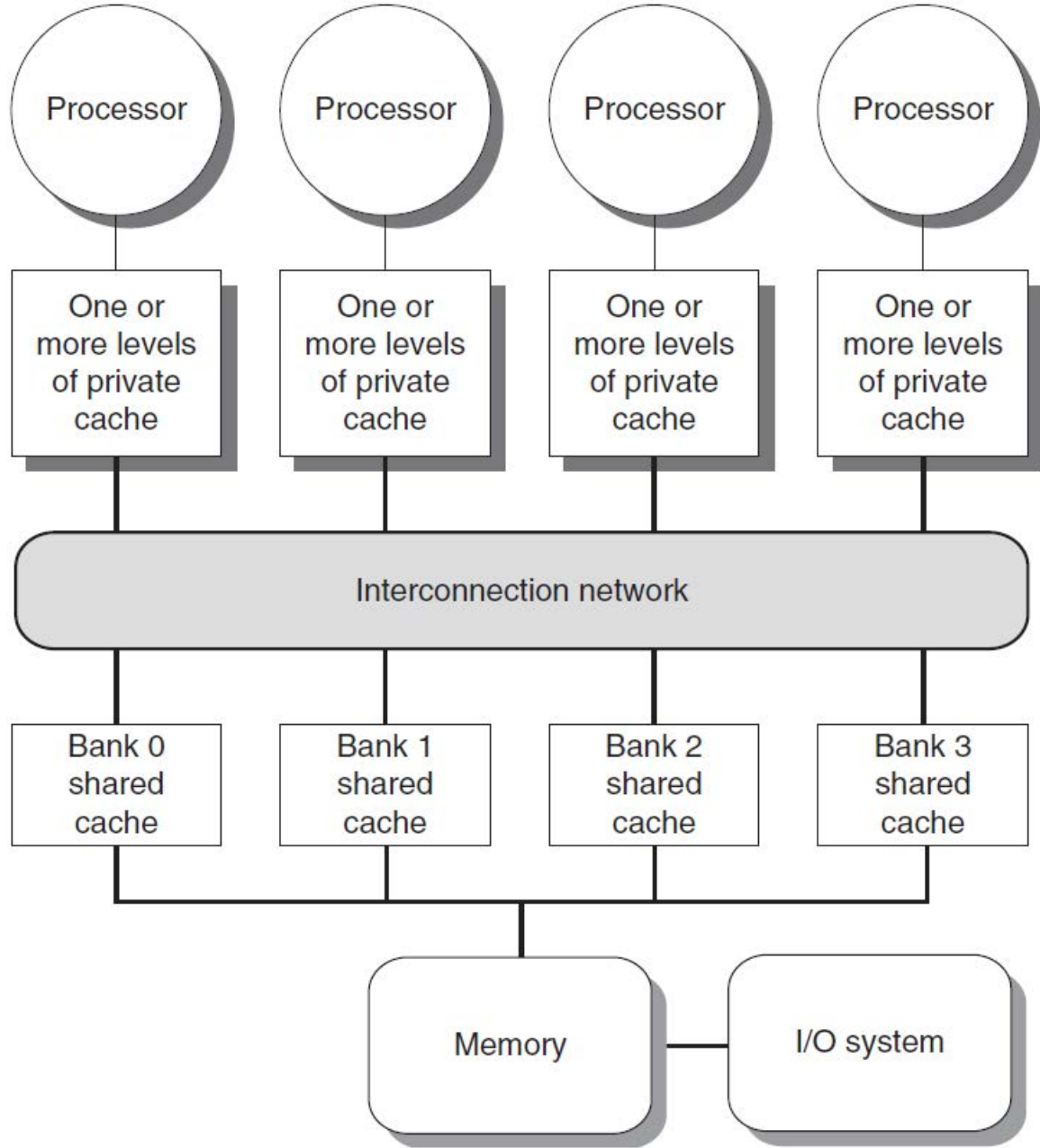


Figure 5.8 A multicore single-chip multiprocessor with uniform memory access

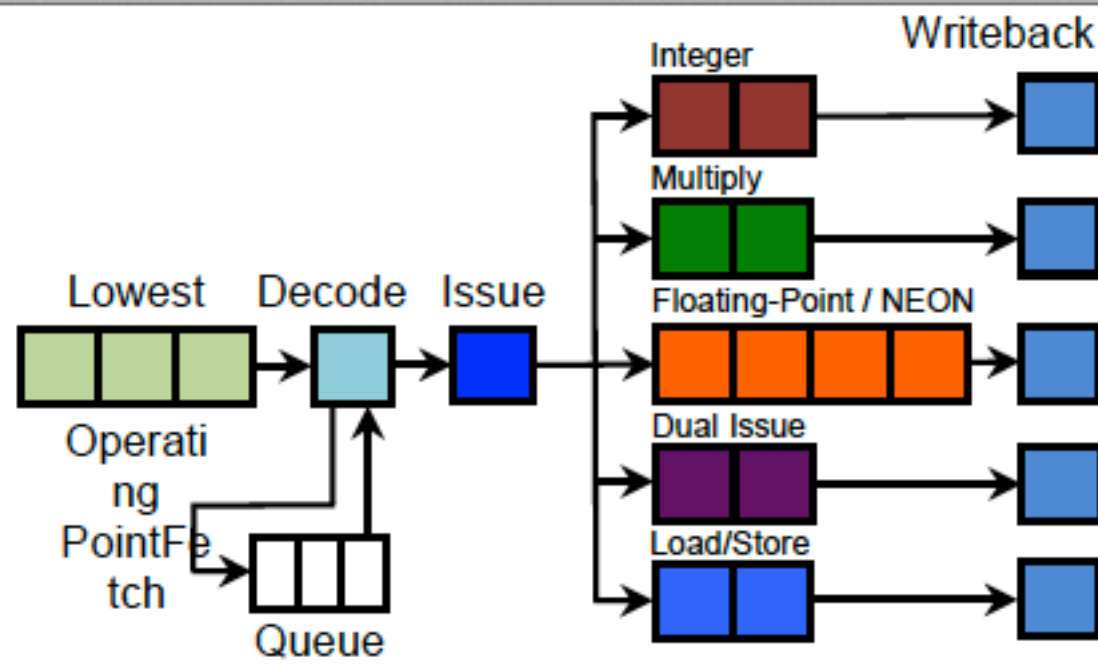


Figure 1 Cortex-A7 Pipeline

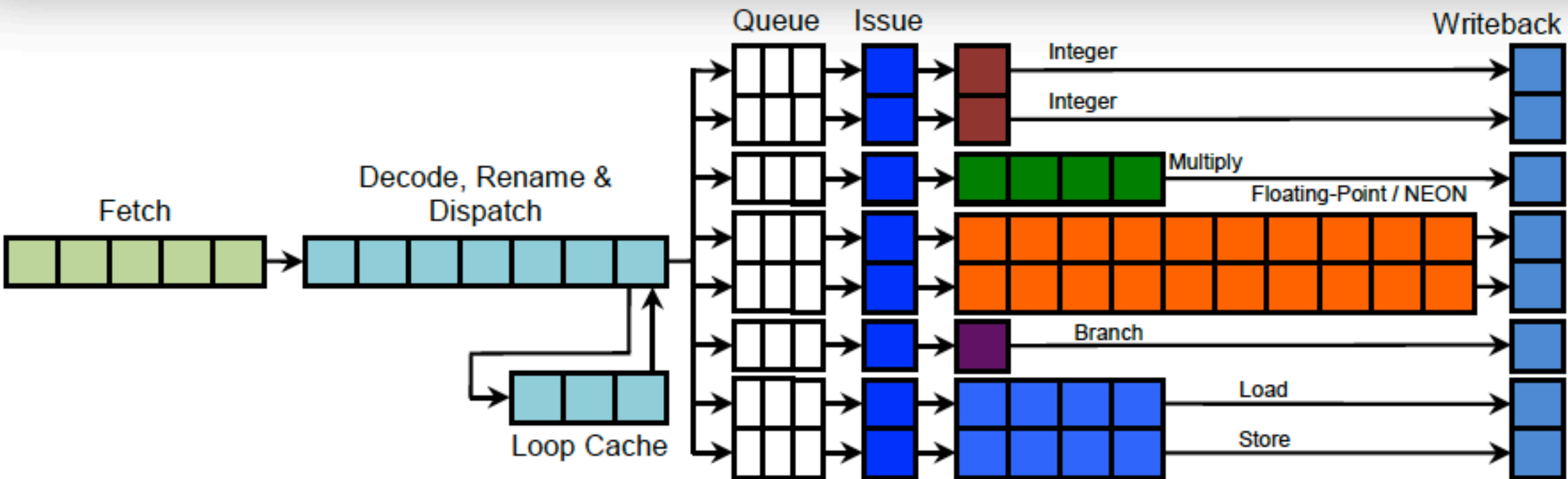


Figure 2 Cortex-A15 Pipeline

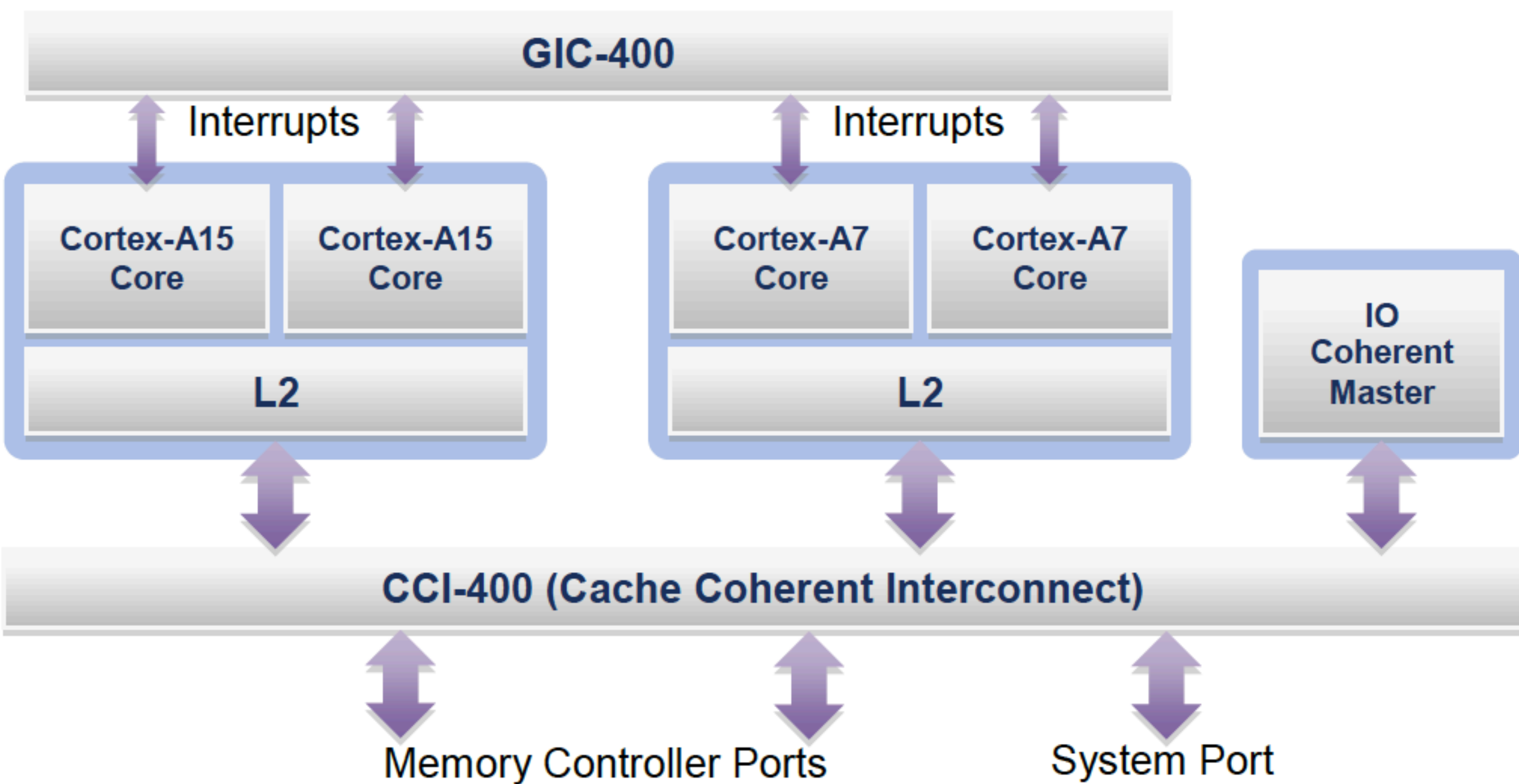
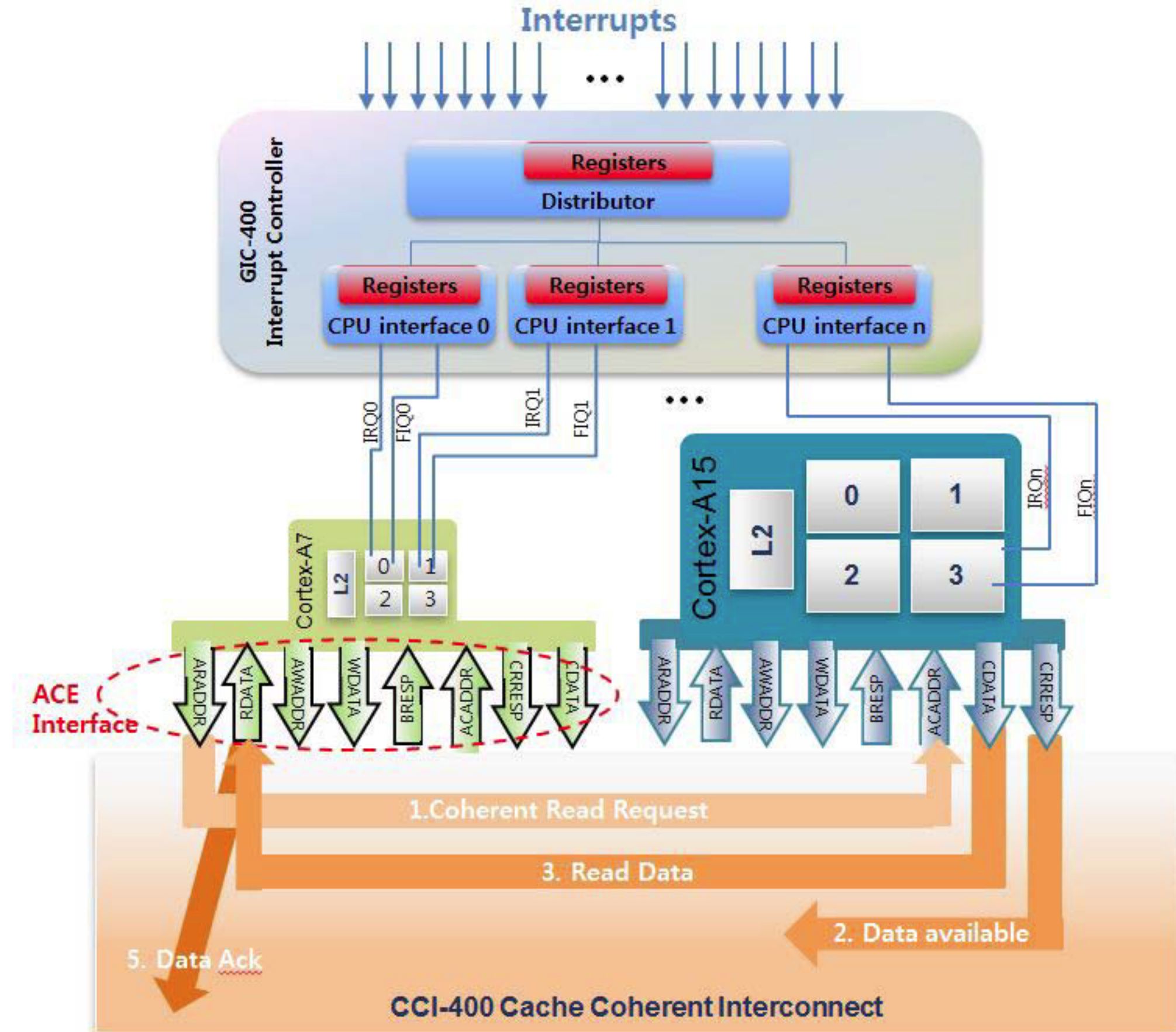
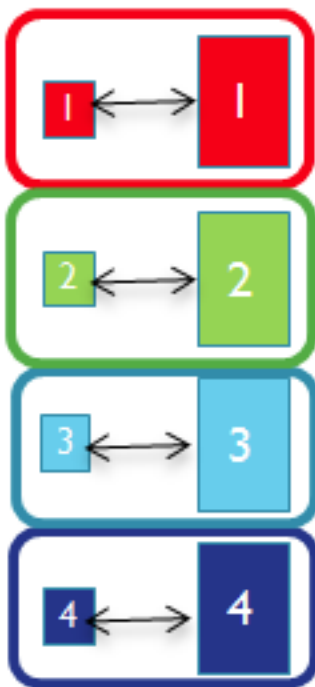


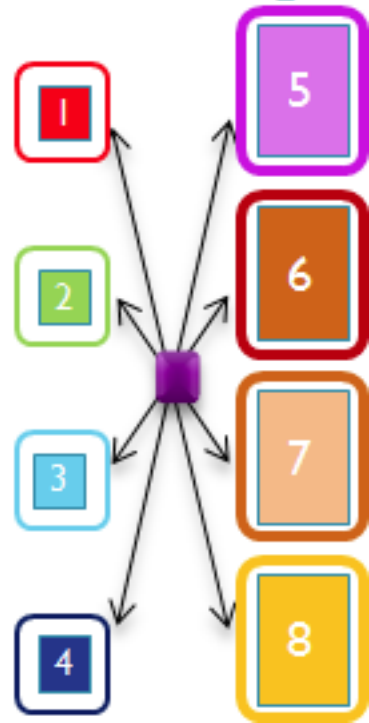
Figure 3 Cortex-A15 CCI Cortex-A7 System



CPU Migration



Global Task Scheduling



In this model, each big core is paired with a LITTLE core. Only one core in each pair is active at any one time, with the inactive core being powered down. The active core in the pair is chosen according to current load conditions. Using the example in [Figure 23-3](#), the operating system sees four logical cores. Each logical core can physically be a big or LITTLE core. This choice is driven by *Dynamic Voltage and Frequency Scaling* (DVFS). This model requires the same number of cores in both the clusters.

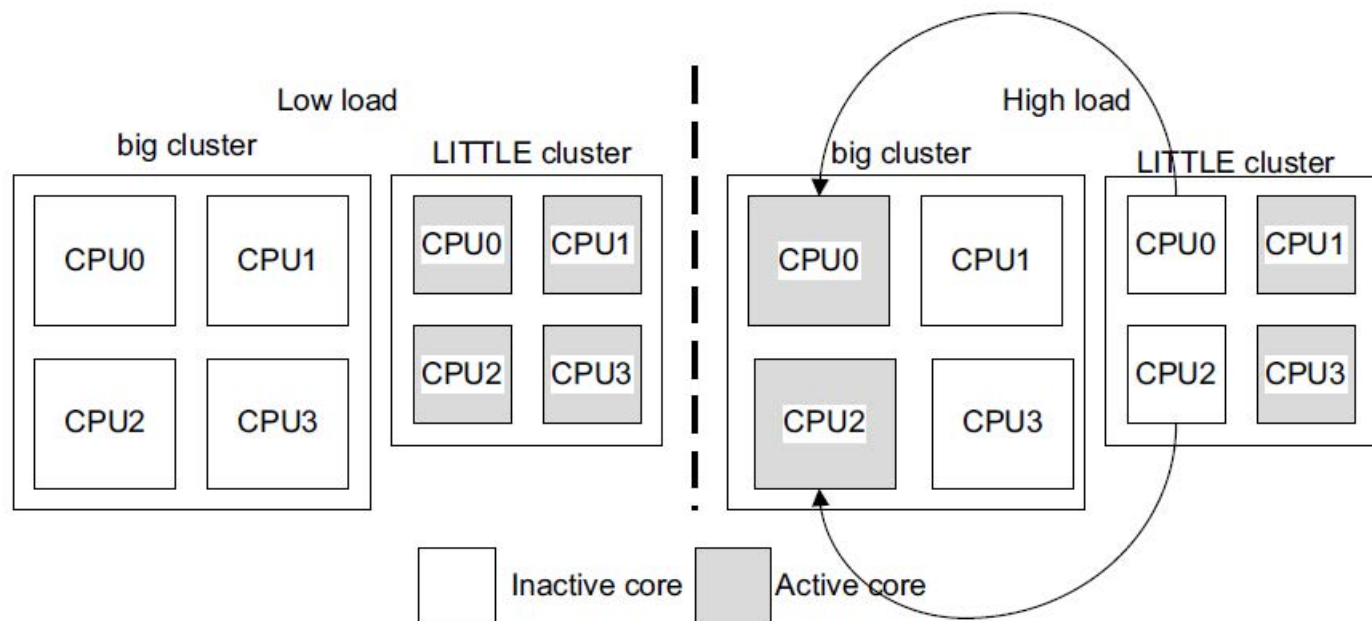


Figure 23-3 CPU migration

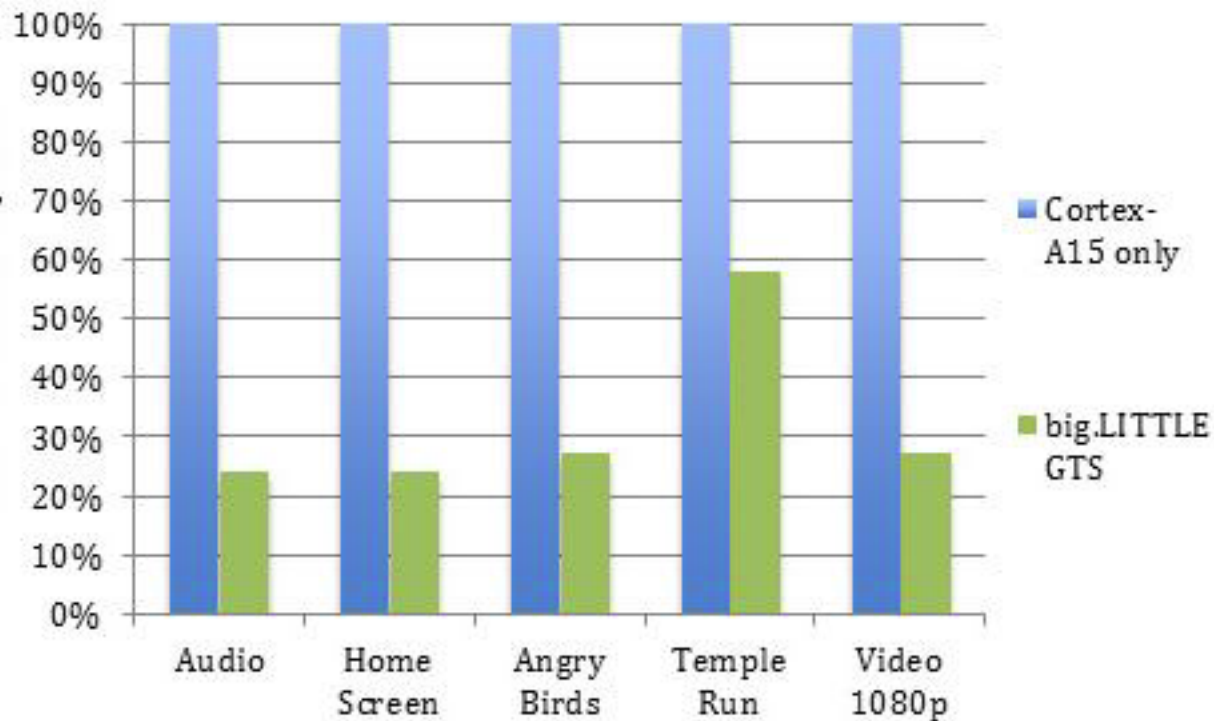
The system actively monitors the load on each core. High load causes the execution context to be moved to the big core, and conversely, when the load is low, the execution is moved to the LITTLE core. Only one core in the pairing can be active at any time. When the load is moved from an outbound core to an inbound core, the former is switched off. This model allows a mix of big and LITTLE cores to be active at any one time, and support the use of asymmetric topologies, that is, an unequal number of big and LITTLE cores.

Power



Performance

CPU Power Consumption



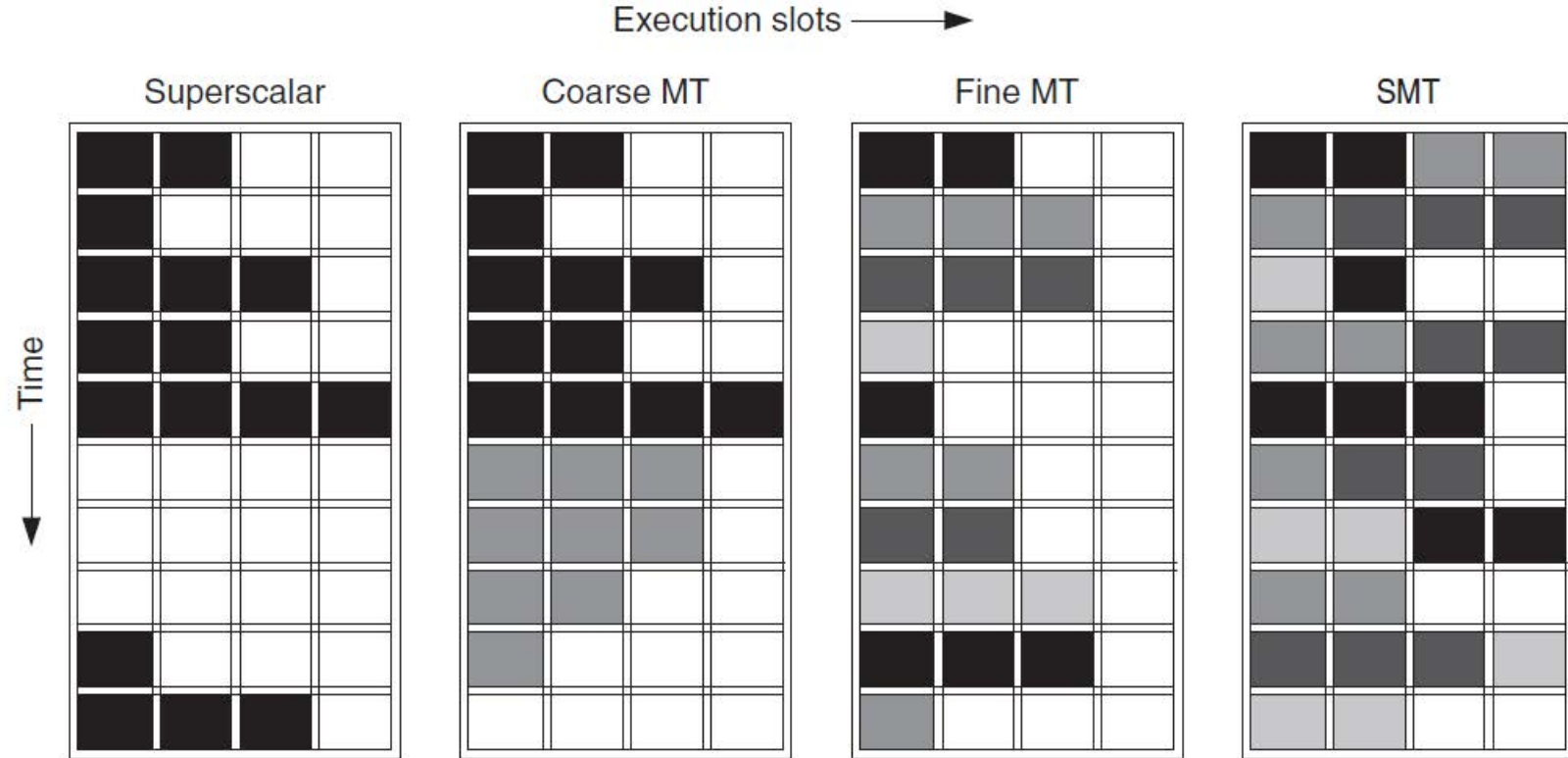
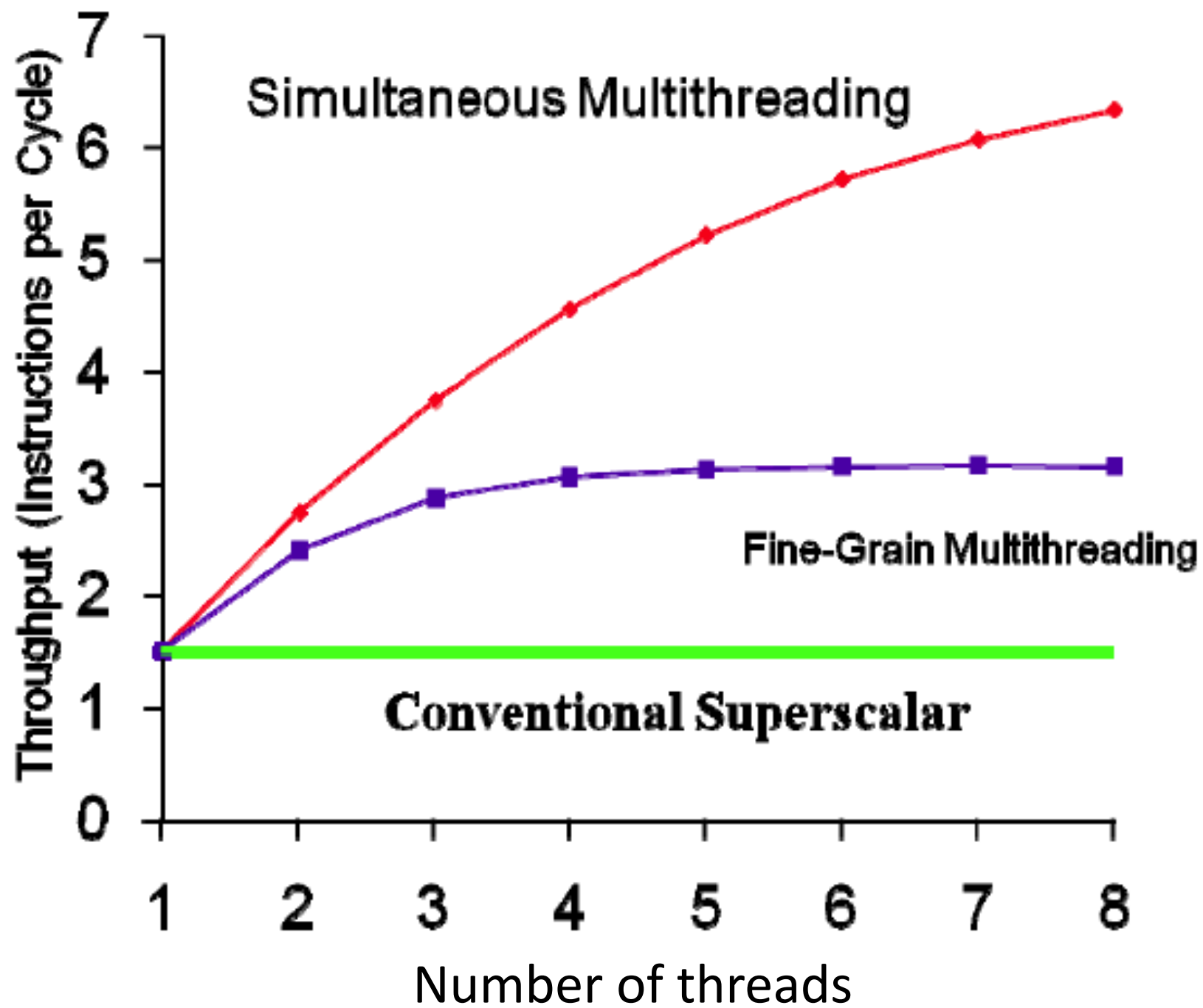


Figure 3.28 How four different approaches use the functional unit execution slots of a superscalar processor. The horizontal dimension represents the instruction execution capability in each clock cycle. The vertical dimension represents a sequence of clock cycles. An empty (white) box indicates that the corresponding execution slot is unused in that clock cycle. The shades of gray and black correspond to four different threads in the multithreading processors. Black is also used to indicate the occupied issue slots in the case of the superscalar without multithreading support. The Sun T1 and T2 (aka Niagara) processors are fine-grained multithreaded processors, while the Intel Core i7 and IBM Power7 processors use SMT. The T2 has eight threads, the Power7 has four, and the Intel i7 has two. In all existing SMTs, instructions issue from only one thread at a time. The difference in SMT is that the subsequent decision to execute an instruction is decoupled and could execute the operations coming from several different instructions in the same clock cycle.

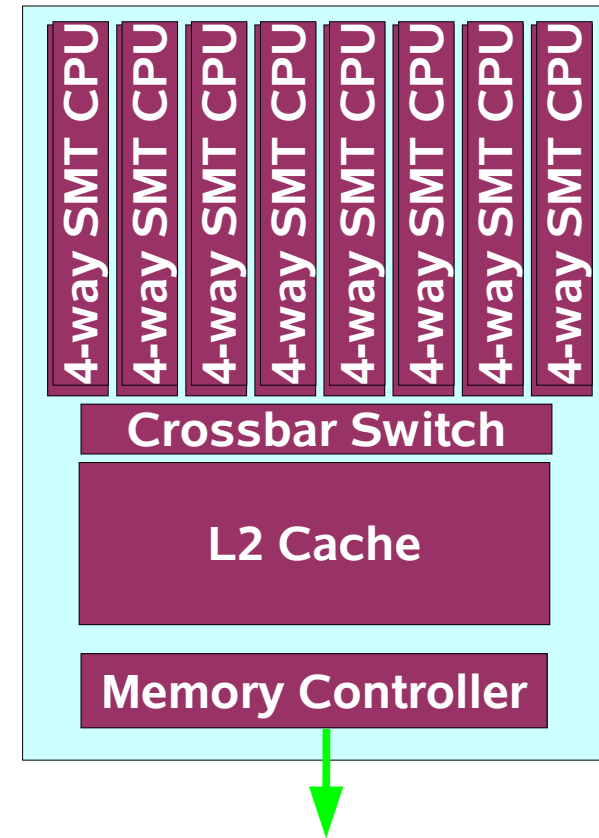


3rd Generation CMTs

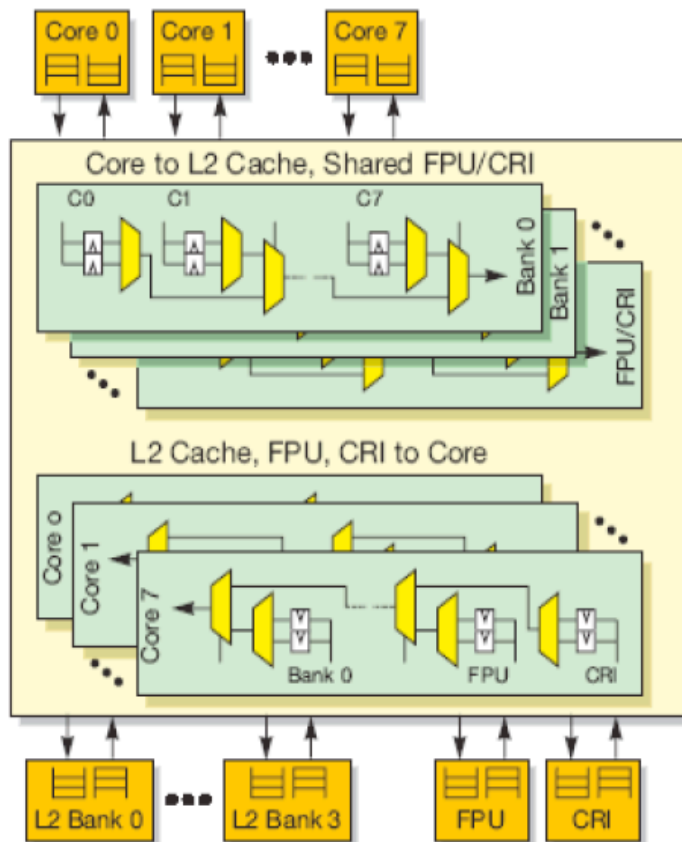
- Future CMTs need to satisfy these power and bandwidth constraints while delivering ever increasing performance
- CMT processors are likely to be designed from the ground-up
 - Entire design optimized for a CMT design point
- Multiple cores per chip

Examples:

- [Sun's Niagara Processor](#):
 - 8 four-way SMT cores



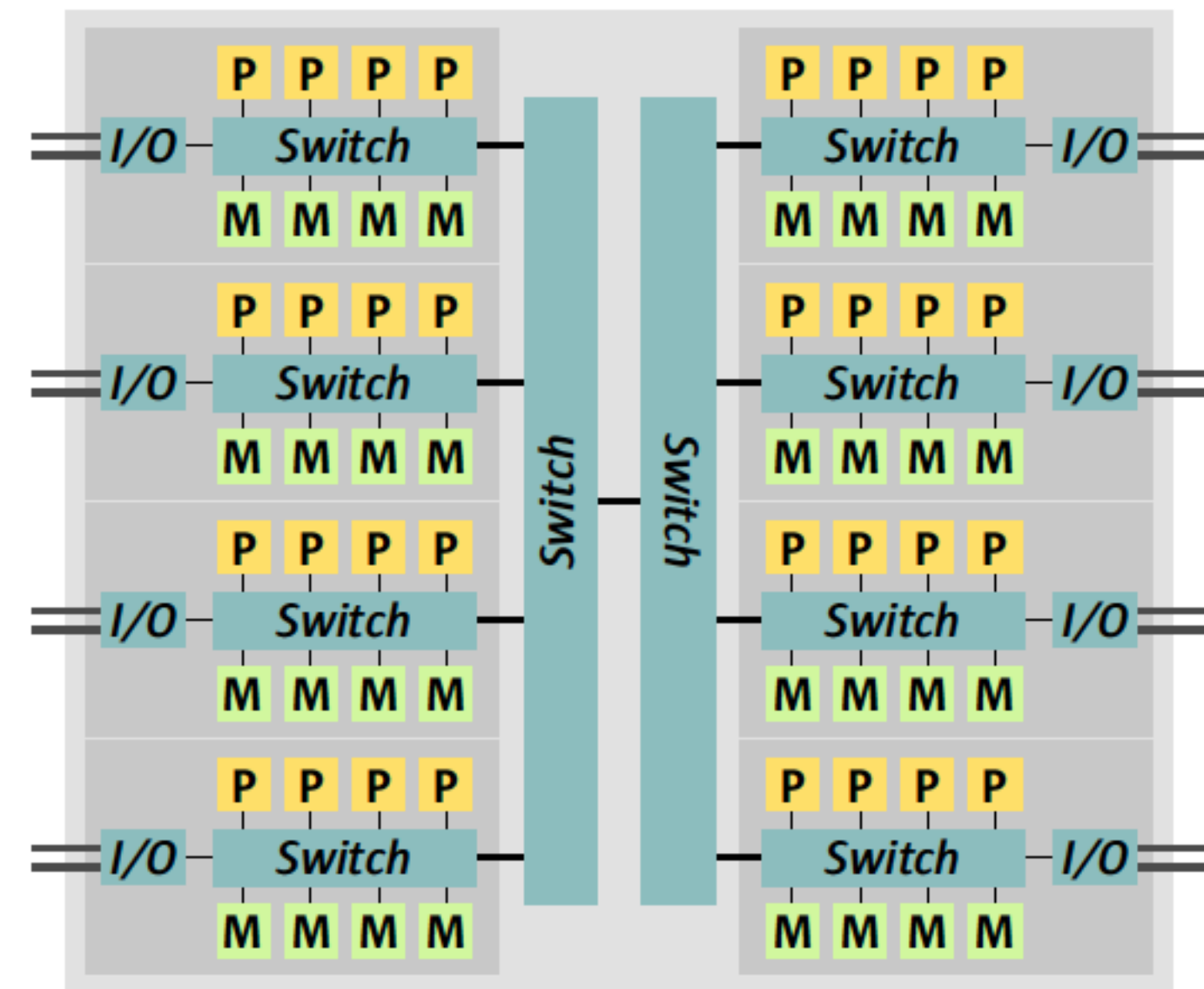
CPU-Cache Crossbar



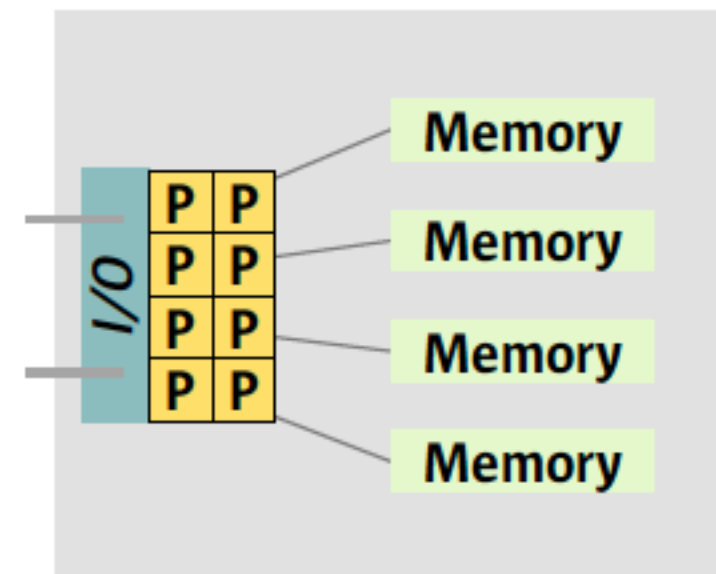
The eight SPARC cores, the four L2-cache banks, the I/O Bridge, and the FPU all interface with the crossbar.

The CPU cache crossbar (CCX) features include:

- Each requester queues up to two packets per destination.
- Three stage pipeline – request, arbitrate, and transmit.
- Centralized arbitration with oldest requester getting priority.
- Core-to-cache bus optimized for address plus double word store.
- Cache-to-core bus optimized for 16-byte line fill. 32-byte Icache line fill delivered in two back-to-back clocks.

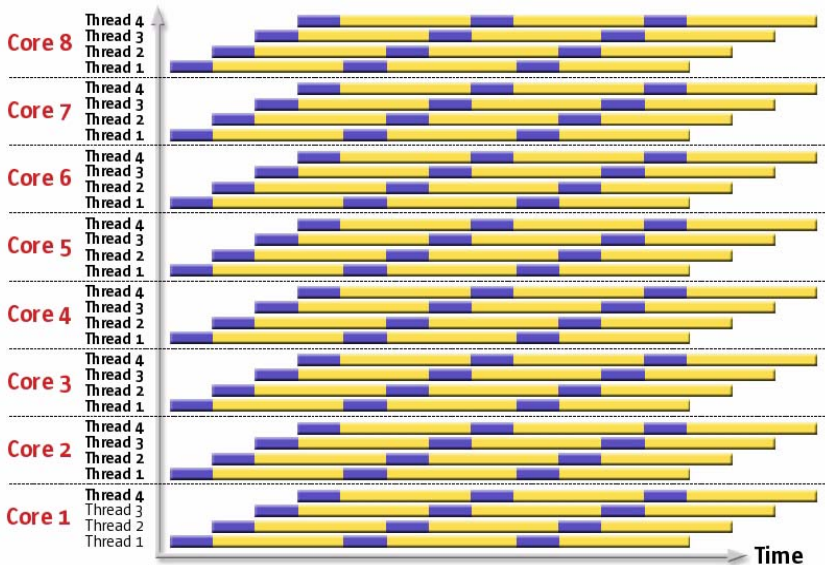


**Traditional 32-thread SMP server: One cabinet,
eight processor/memory boards,
many switch ASICs and connections**



**Sun Fire Server with CoolThreads Technology:
One motherboard, no switch ASICs,
no board connections**

Figure 7. By compressing 32 threads onto a single processor, the UltraSPARC T1 processor helps enable system designs that are less susceptible to failure, with fewer parts and connections



Memory Access

Compute

Има два подхода за изграждането на матричните процесори.

- Първият е чрез съвместяване по време (конвейеризация) на изпълнението на командите, обработващи векторите. Обикновено, процесорите въплъщаващи този подход се наричат векторни процесори, а компютрите, които са базирани на тях – векторни компютри. Примери за такива компютри (процесори) са: CRAY-1, CRAY-2, CRAY X-MP, CRAY CS 6400, Fujitsu VP-200, Hitachi S810, Hitachi S820, FPS 120L(B), FPS 164, IBM 3090 и др. [1-5]. Първите успешни векторните процесори бяха **CDC** Cyber 100 (**C**ontrol **D**ata **C**orporation) и TI ASC (**T**exas **I**nstruments **A**dvanced **S**cientific **C**omputer), които използваха адресация памет-памет за достъп до данните. За първи път в известния CRAY-1 (лансиран през 1974 г.) се въвеждат 8 векторни регистри, всеки съхраняващ 64 думи от по 64 бита всяка. Компютрите на Cray Research бяха изключително успешни и името CRAY стана синоним на суперкомпютър. Различни японски компании (Fujitsu, Hitachi, NEC) също предложиха регистрово ориентирани архитектури решения подобни на тези на Cray Research.

- Вторият подход на изграждането на архитектурата се базира на пространственото повторение на изпълнението на командата. За целта се използват множество процесори, свързани по определен начин, най-често във вид на матрица. Този тип компютри спадат към **SIMD** класа по класификацията на Флин (виж тема 3) и техни представители са Staran, DAP на ICL, Exemplar 1200/CD 2 или Exemplar1200/XA 8 на Convex, SP2 на IBM, Power Challenge на Silicon Graphics Inc.и др.

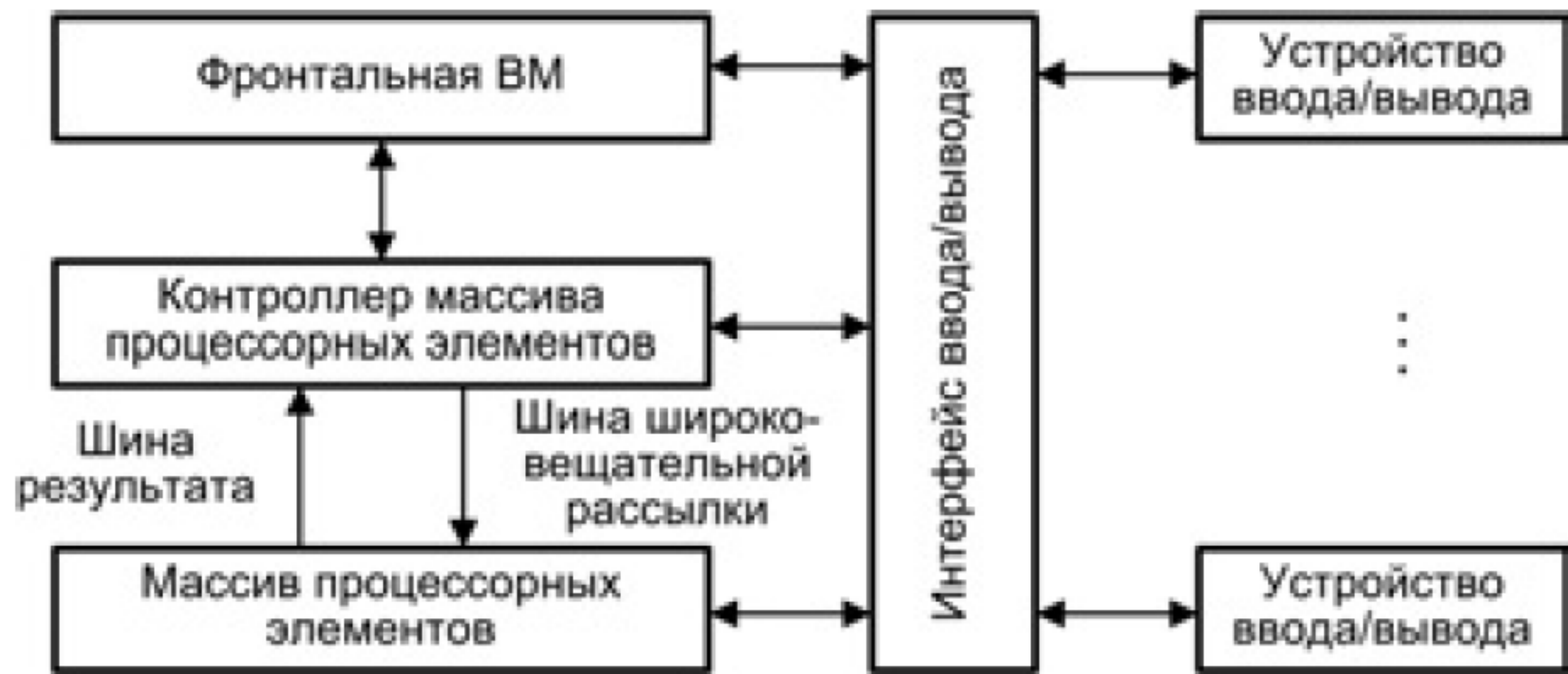


Рис. 13.12. Обобщенная модель матричной SIMD-системы

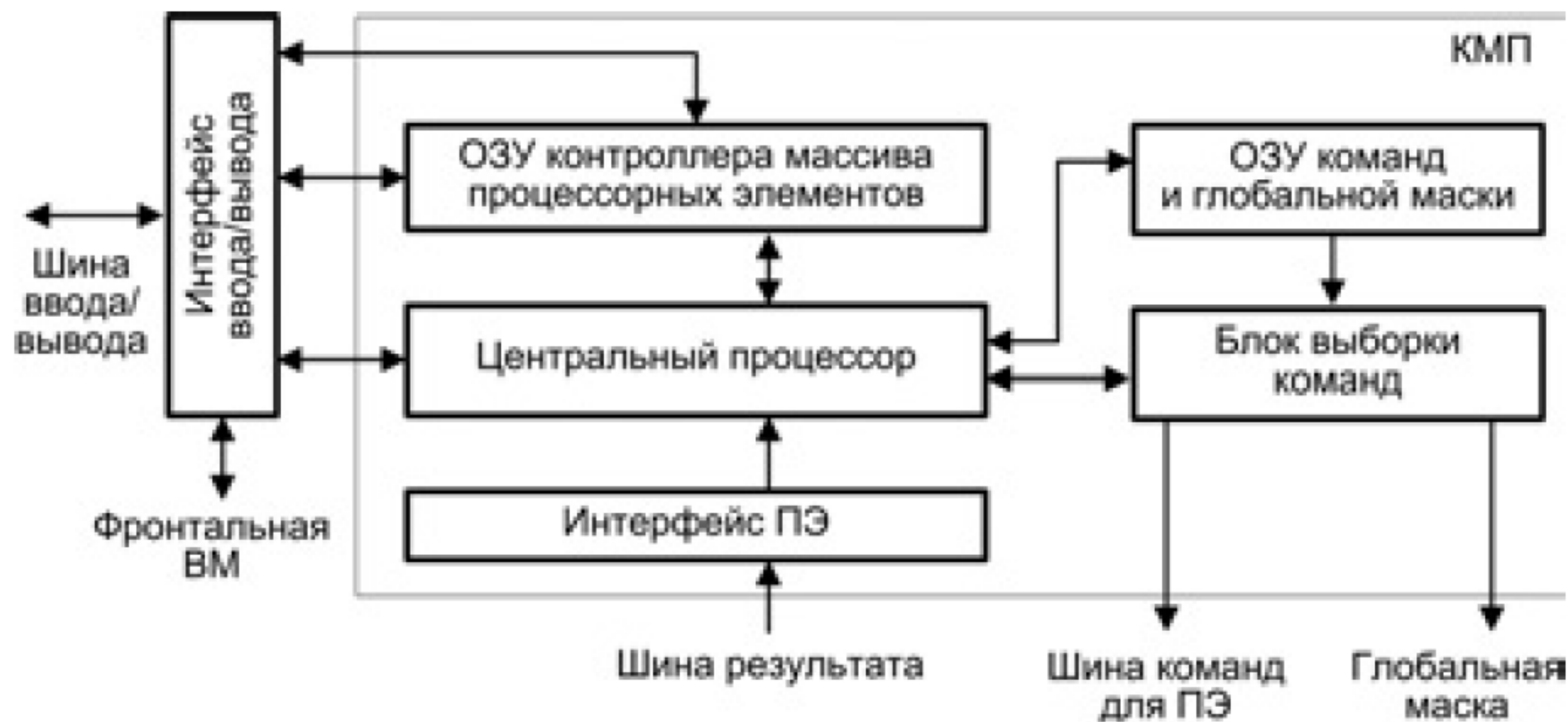
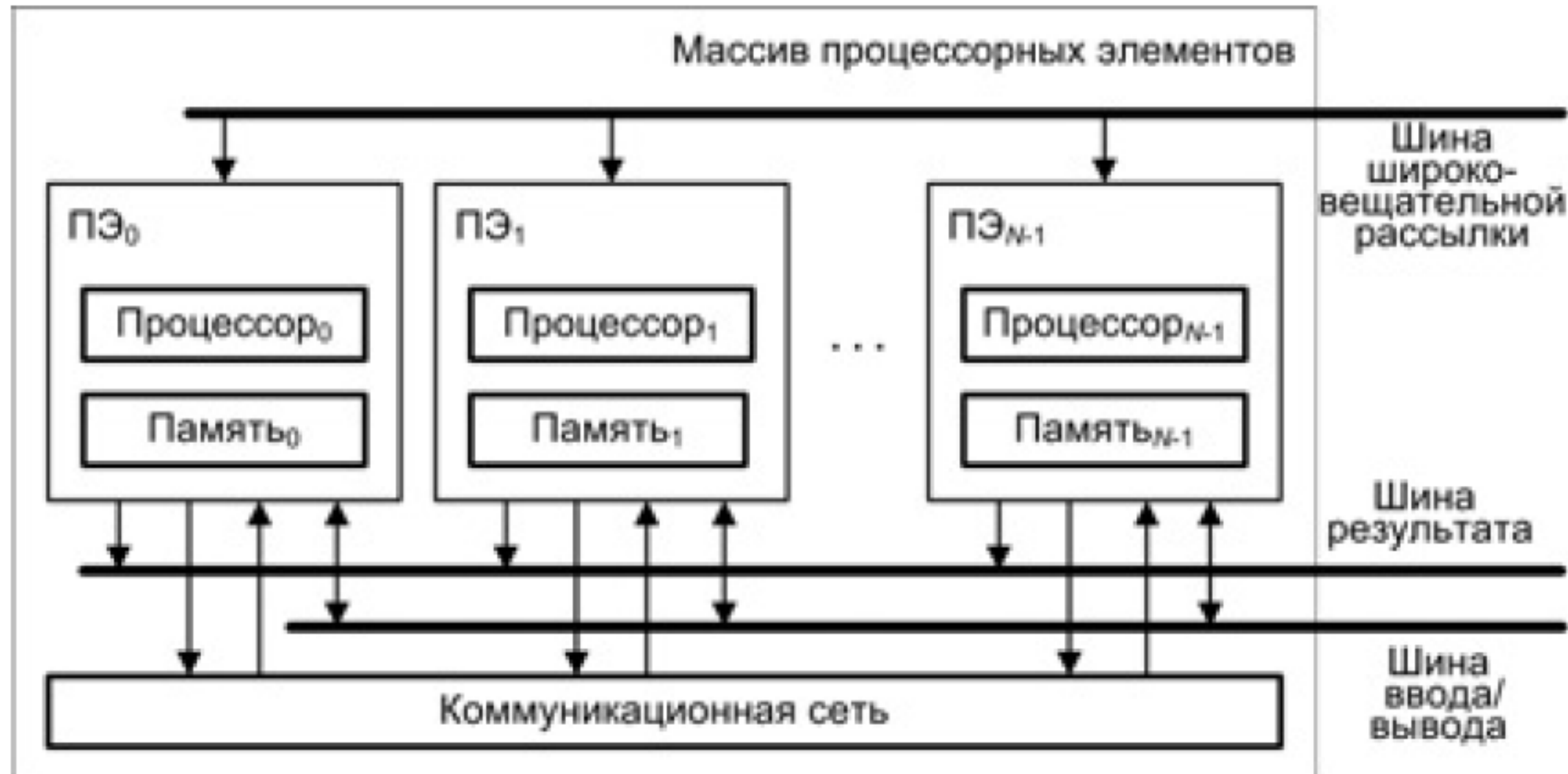
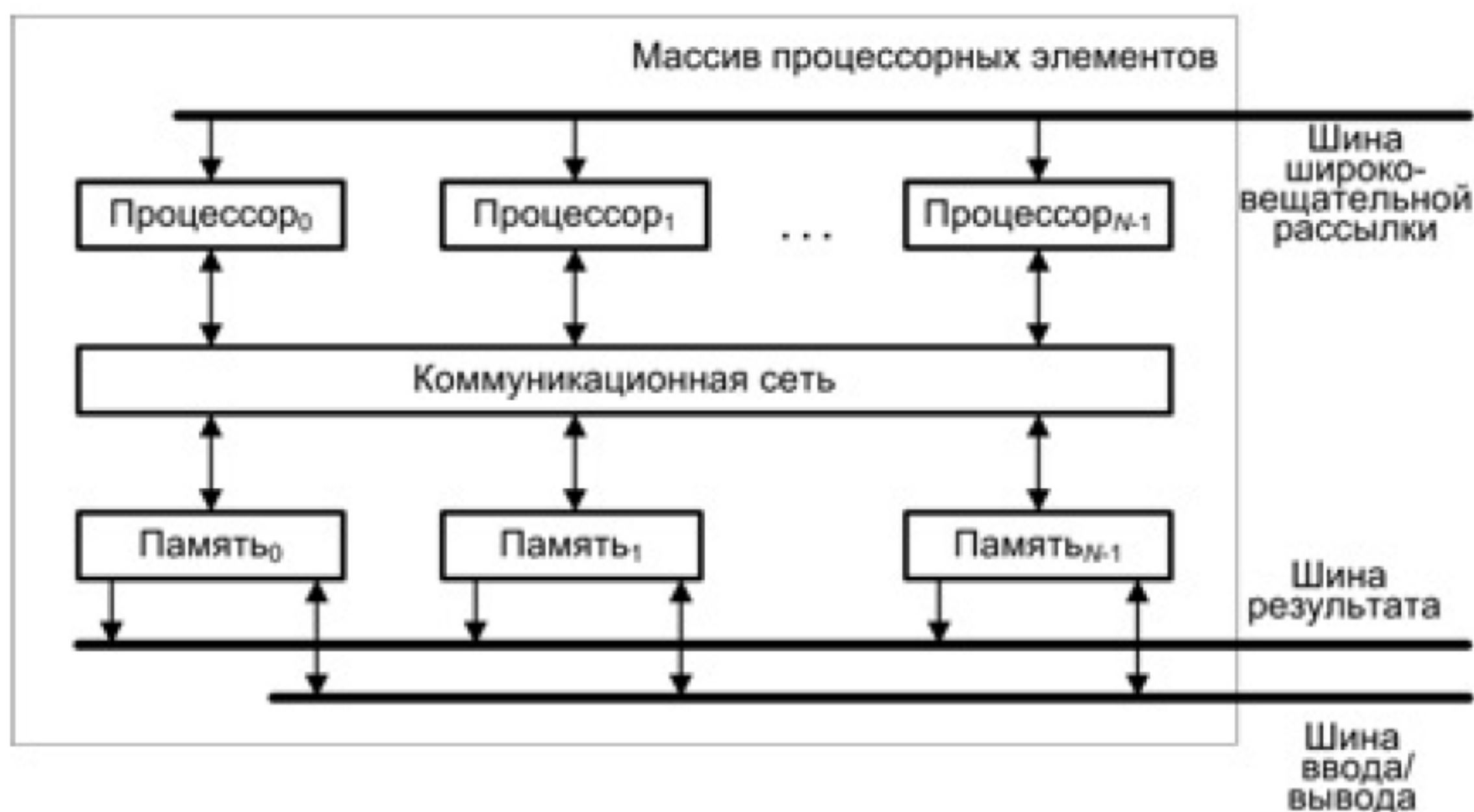


Рис. 13.13. Модель контроллера массива процессорных элементов [164]



а



б

Рис. 13.14. Модели массивов процессоров: а — «процессорный элемент-процессорный элемент»; б — «процессор-память»

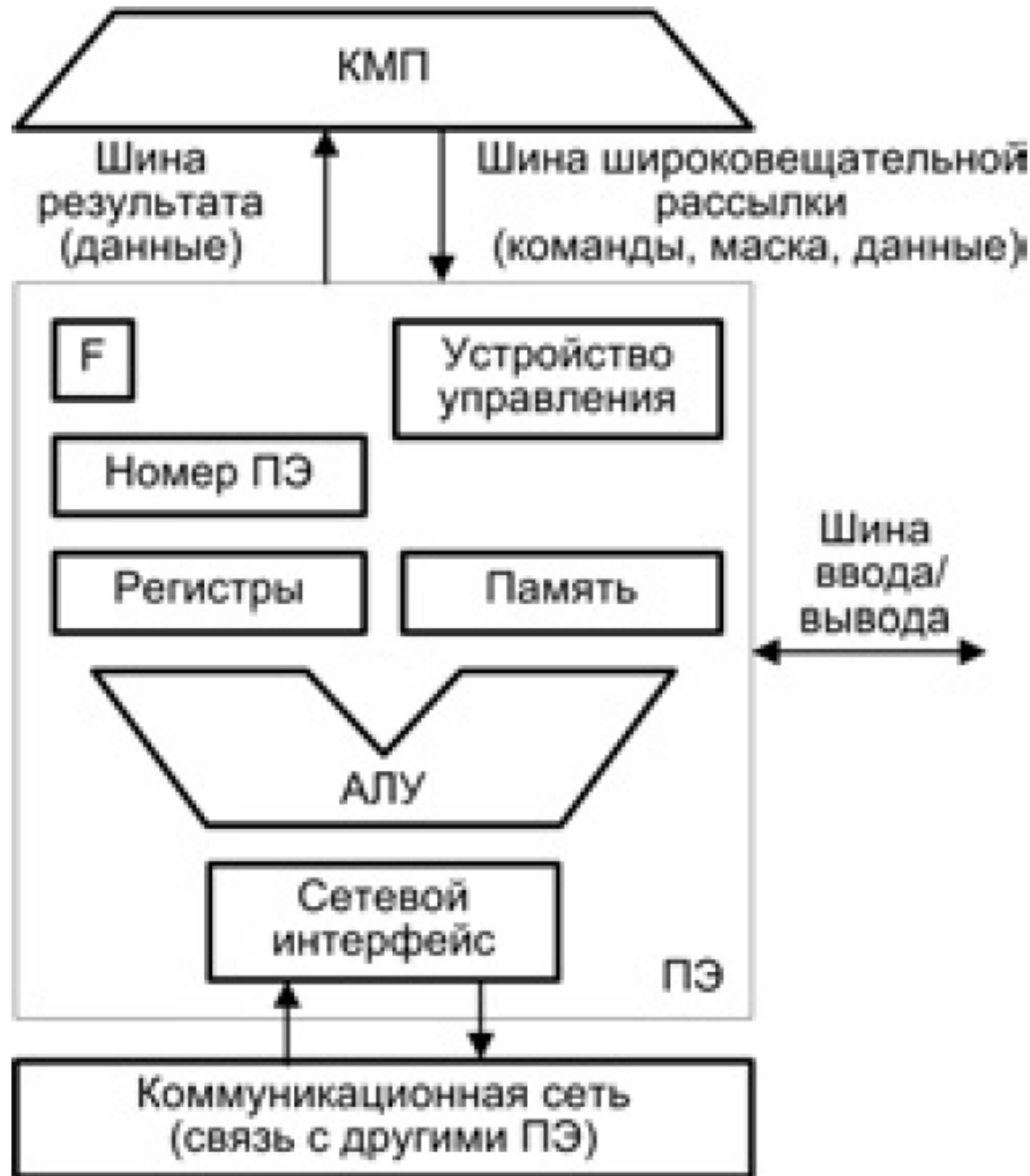
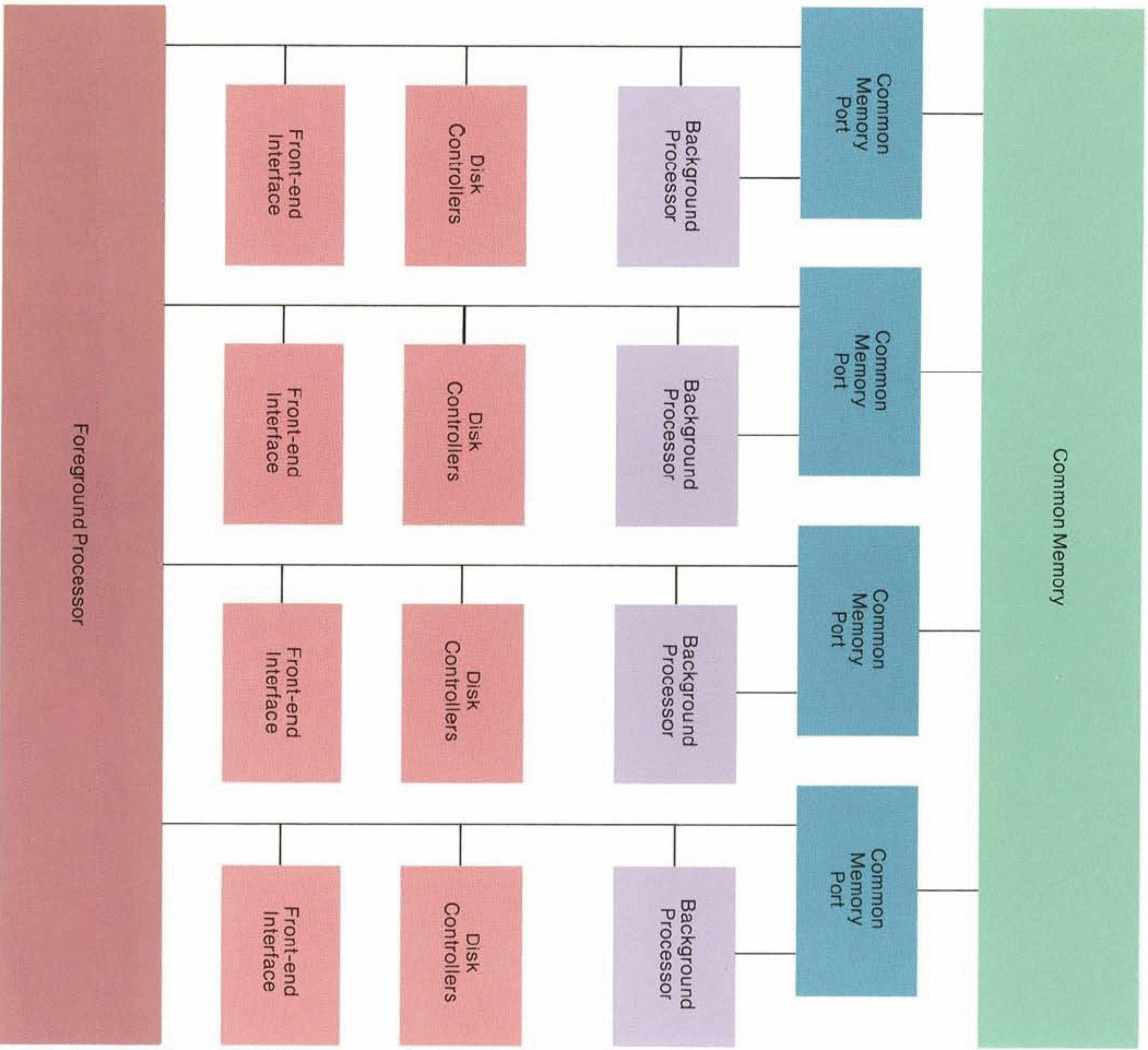


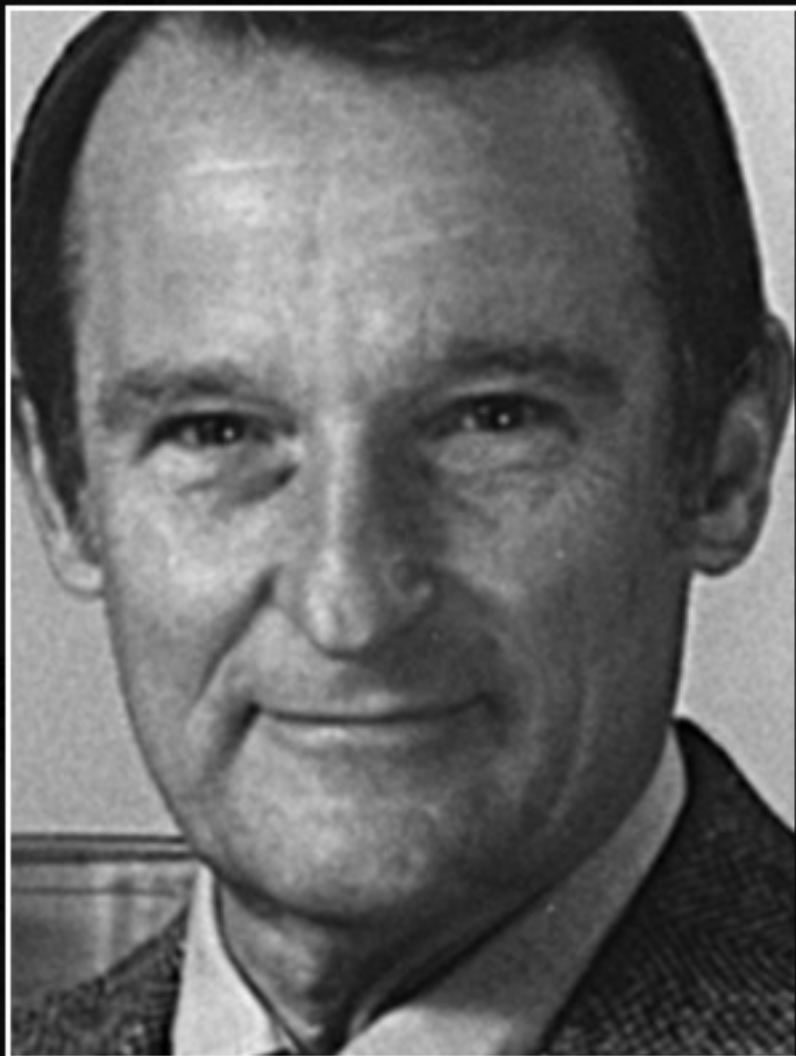
Рис. 13.15. Модель процессорного элемента [164]



Векторен суперкомпютър „Cray-2“:

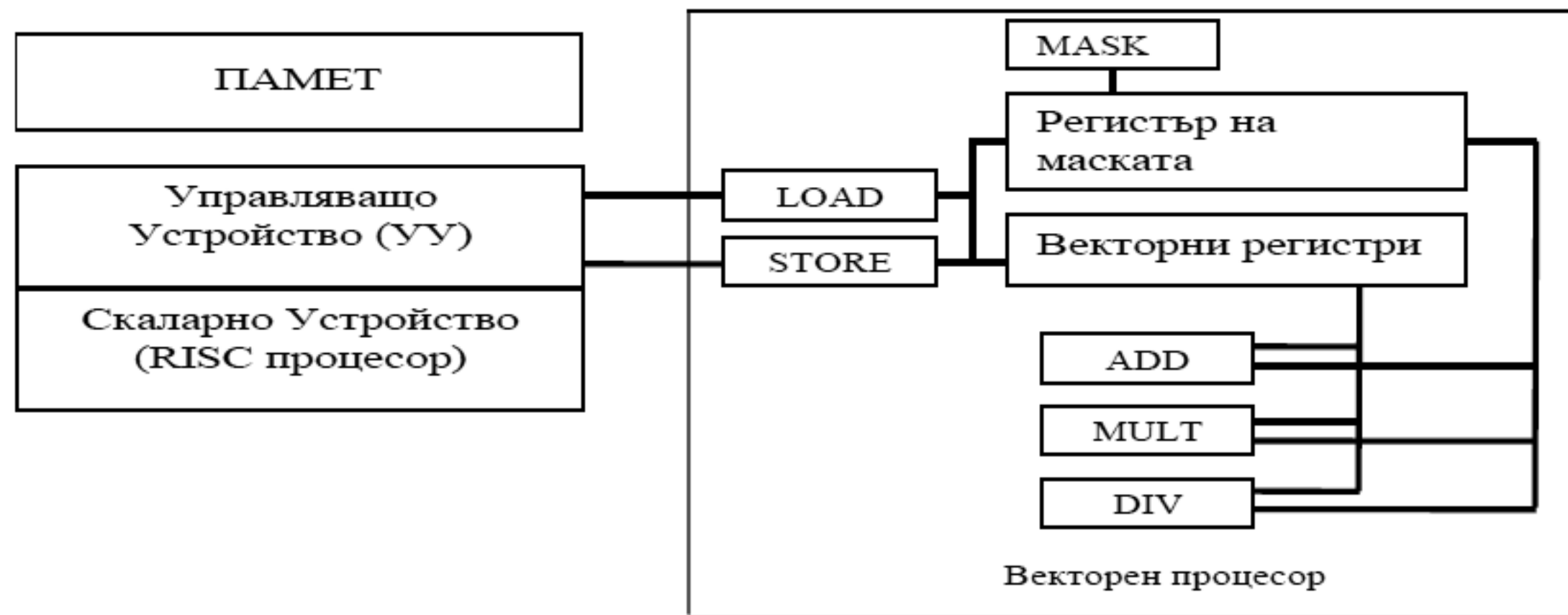
- ◆ От 1985 до 1990 г. най-бързият компютър в света
- ◆ Върхова производителност: 1,9 GFLOPS
- ◆ Състои се от четири 64-битови векторни процесора
- ◆ Елементна база: Емитерно-свързана логика (ECL) / 5 V
- ◆ Тактова честота: 244 MHz (тактов период: 4,1 ns)
- ◆ Общ обем на паметта: до 2 GB
- ◆ Потребление на електрическа енергия: до 195 kW
- ◆ Захранван от 2 трифазни 480V / 60→400Hz умформера
- ◆ През всяка захранваща шина протича до 2200 A ток
- ◆ Охлаждане чрез потапяне в „FC-74“ и маслени помпи
- ◆ Габарити: 1143 mm височина, 1346 mm „диаметър“
- ◆ 27-те произведени броя продадени за по 12–17000000\$



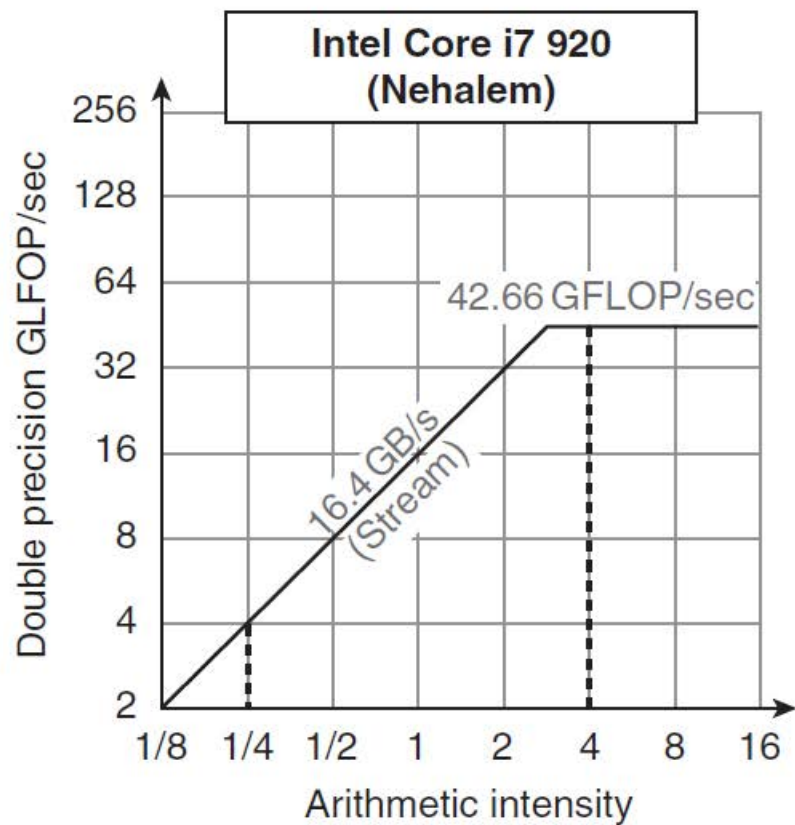
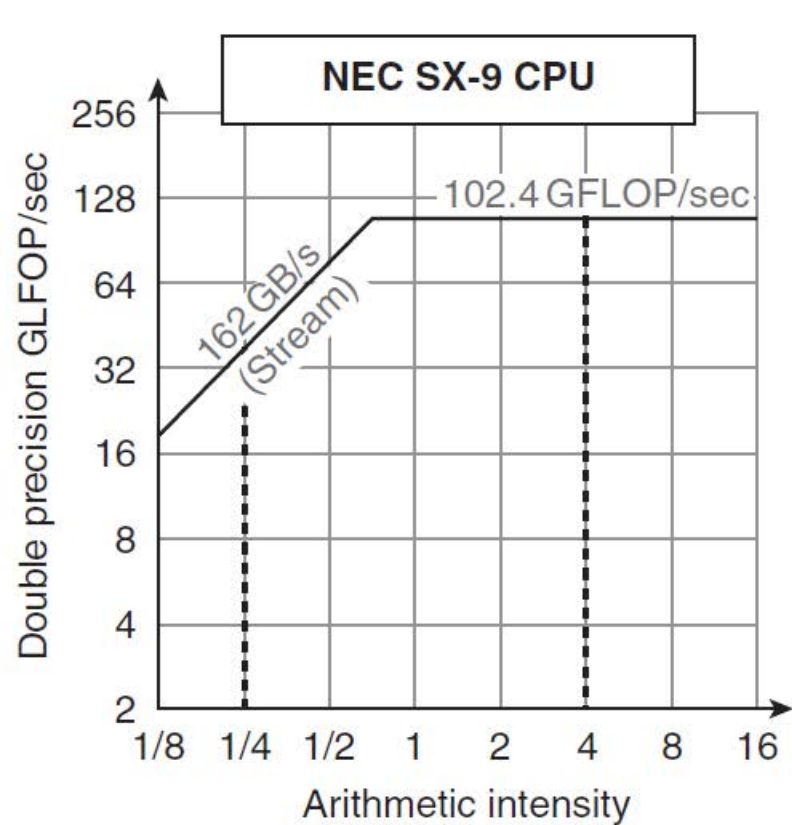


If you were plowing a field, which
would you rather use? Two strong
oxen or 1024 chickens?

— *Seymour Cray* —



Фиг.6-1. Обобщена структура на примерен векторен процесор.
(ADD, MULT, DIV, MASK, LOAD и STORE са специализирани функционални устройства, работещи на конвейерен принцип)



Roofline model for one NEC SX-9 vector processor on the left and the Intel Core i7 920

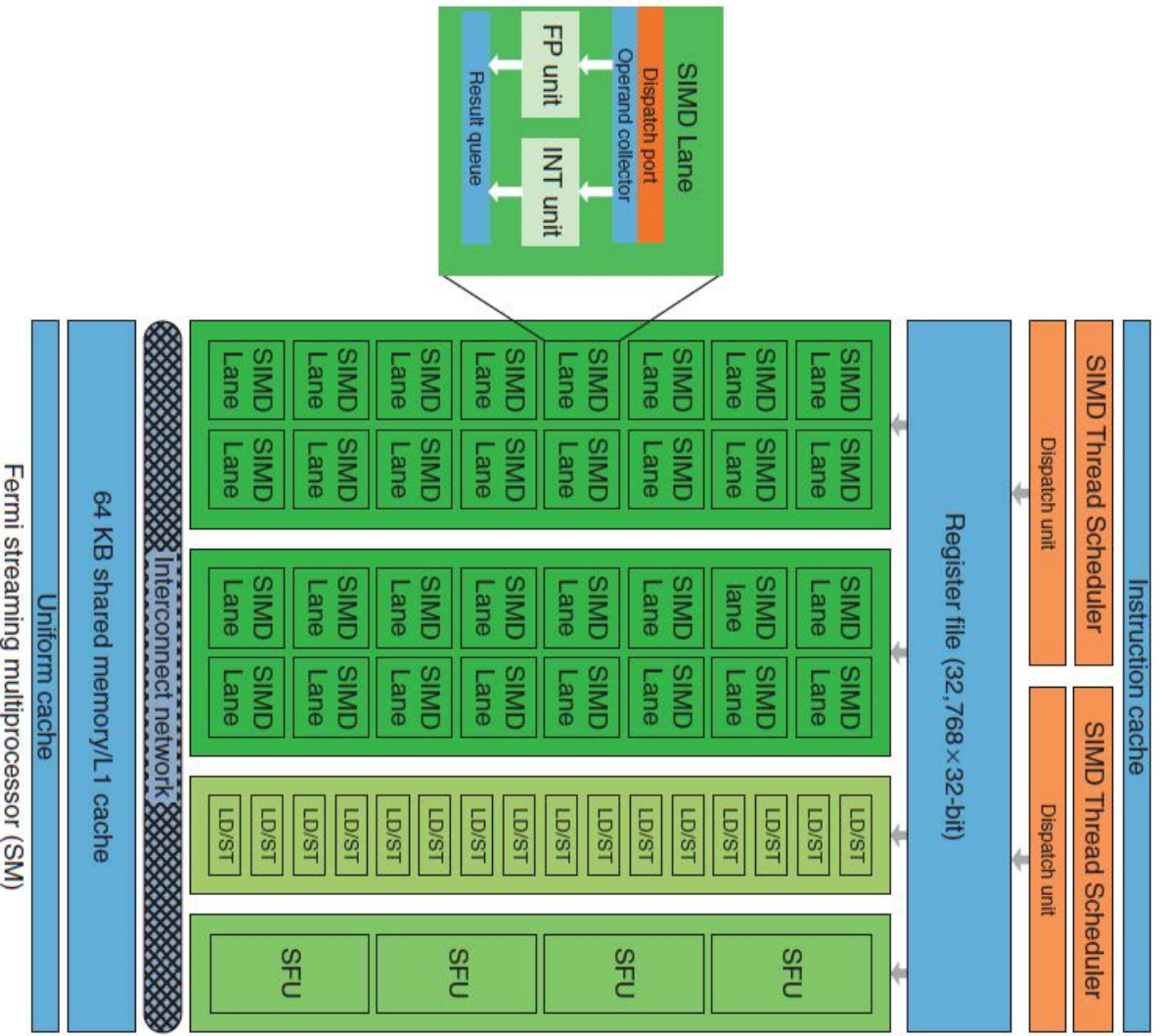


Figure 4.20 Block diagram of the multithreaded SIMD Processor of a Fermi GPU. Each SIMD Lane has a pipelined Floating-point unit, a pipelined Integer unit, some logic for dispatching instructions and operands to these units, and a queue for holding results. The four Special Function units (SFUs) calculate functions such as square roots, reciprocals, sines, and cosines.

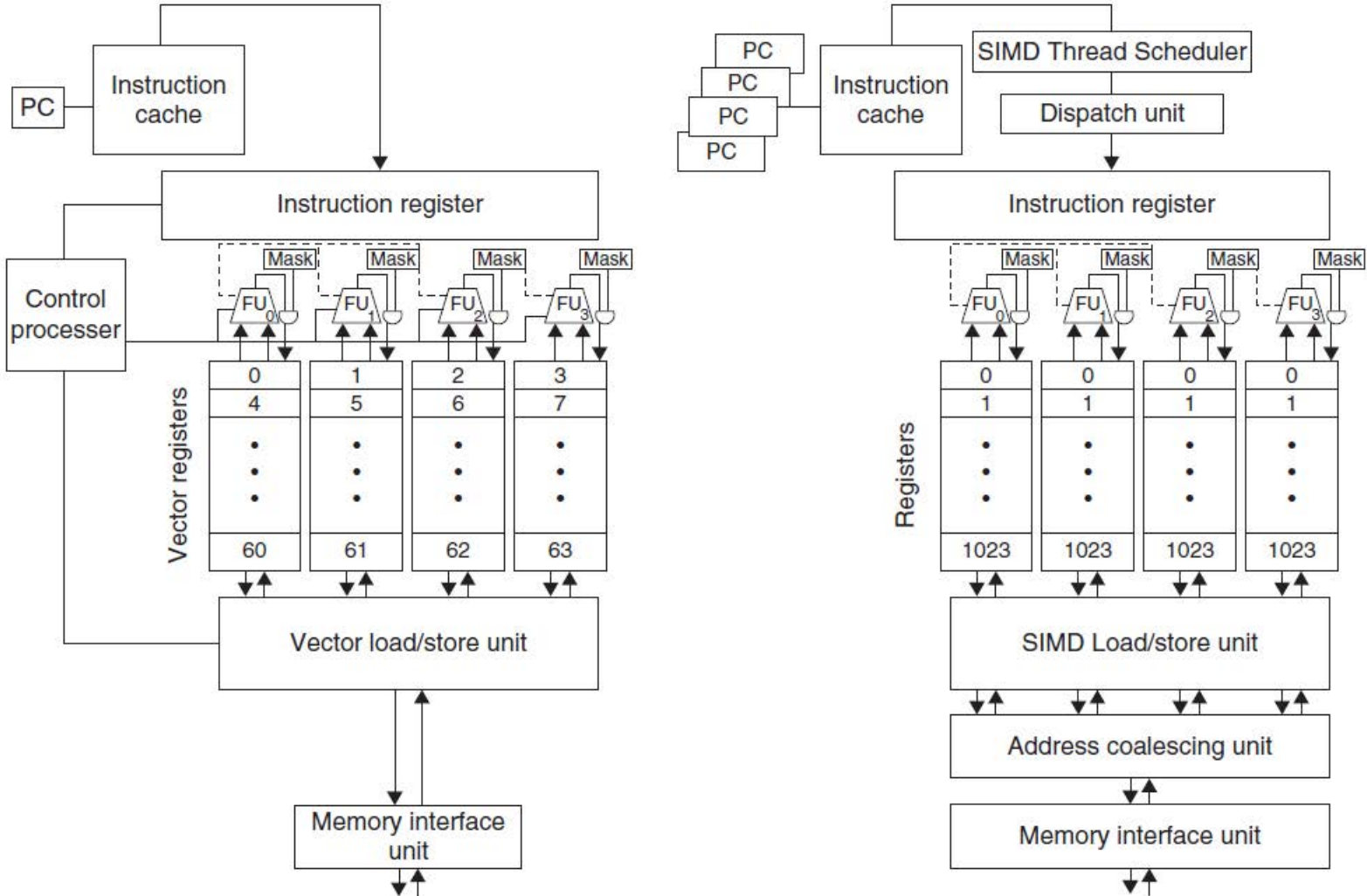


Figure 4.22 A vector processor with four lanes on the left and a multithreaded SIMD Processor of a GPU with four SIMD Lanes on the right. (GPUs typically have 8 to 16 SIMD Lanes.) The control processor supplies scalar operands for scalar-vector operations, increments addressing for unit and non-unit stride accesses to memory, and performs other accounting-type operations. Peak memory performance only occurs in a GPU when the Address Coalescing unit can discover localized addressing. Similarly, peak computational performance occurs when all internal mask bits are set identically. Note that the SIMD Processor has one PC per SIMD thread to help with multithreading.

In addition to the large numerical differences in processors, SIMD lanes, hardware thread support, and cache sizes, there are many architectural differences. The scalar processor and Multimedia SIMD instructions are tightly integrated in traditional computers; they are separated by an I/O bus in GPUs, and they even have separate main memories. The multiple SIMD processors in a GPU use a single address space, but the caches are not coherent as they are in traditional multicore computers. Unlike GPUs, multimedia SIMD instructions do not support gather-scatter memory accesses, which [Section 4.7](#) shows is a significant omission.

Feature	Multicore with SIMD	GPU
SIMD processors	4 to 8	8 to 16
SIMD lanes/processor	2 to 4	8 to 16
Multithreading hardware support for SIMD threads	2 to 4	16 to 32
Typical ratio of single-precision to double-precision performance	2:1	2:1
Largest cache size	8 MB	0.75 MB
Size of memory address	64-bit	64-bit
Size of main memory	8 GB to 256 GB	4 to 6 GB
Memory protection at level of page	Yes	Yes
Demand paging	Yes	No
Integrated scalar processor/SIMD processor	Yes	No
Cache coherent	Yes	No

Figure 4.23 Similarities and differences between multicore with Multimedia SIMD extensions and recent GPUs.

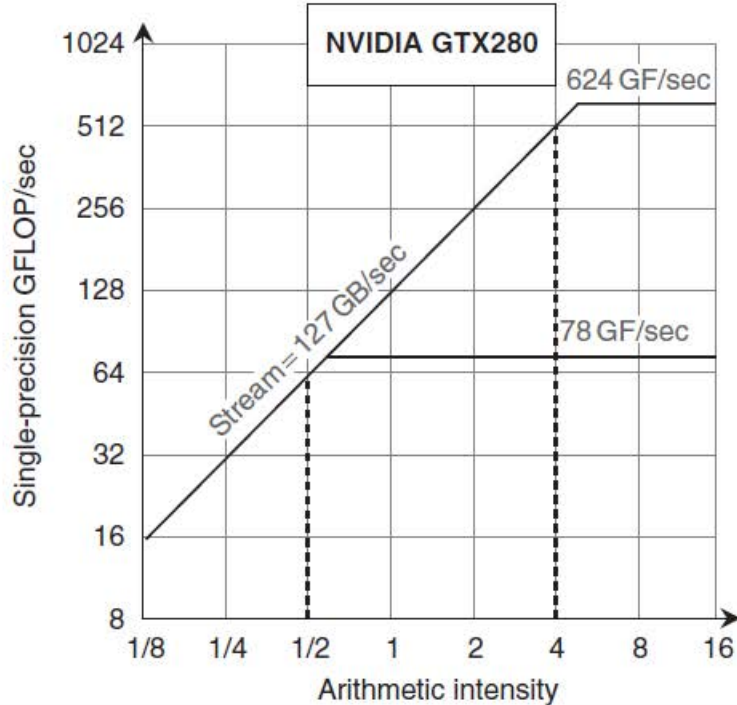
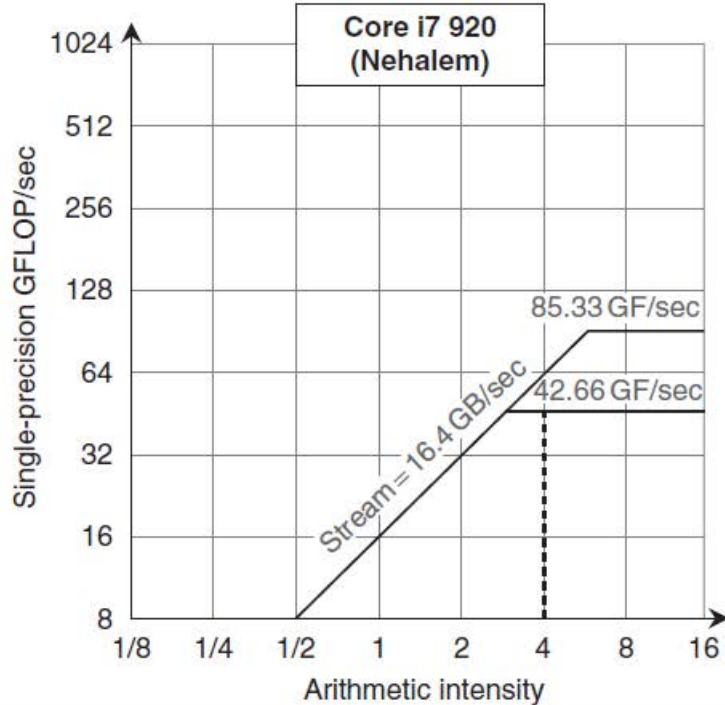
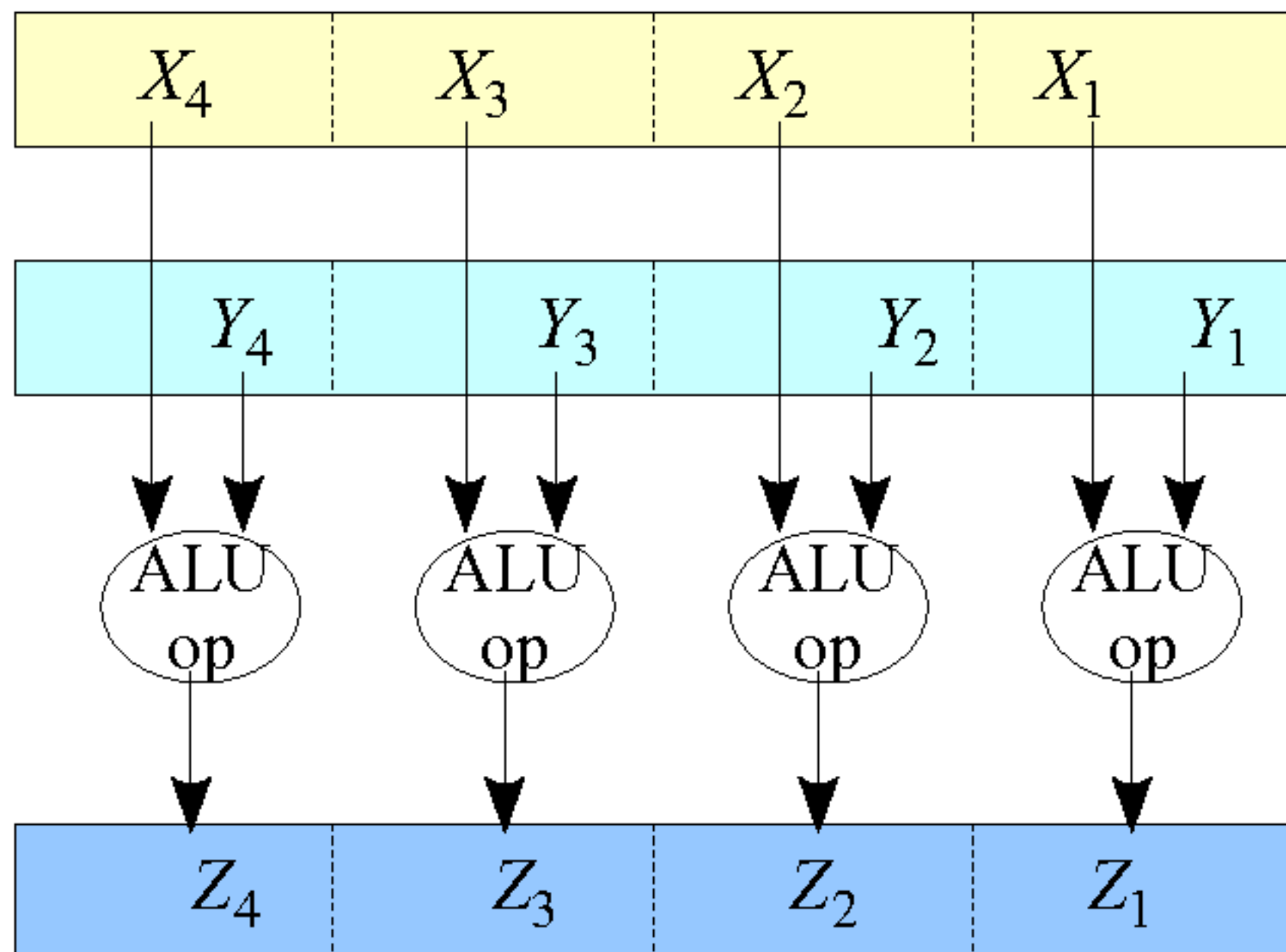


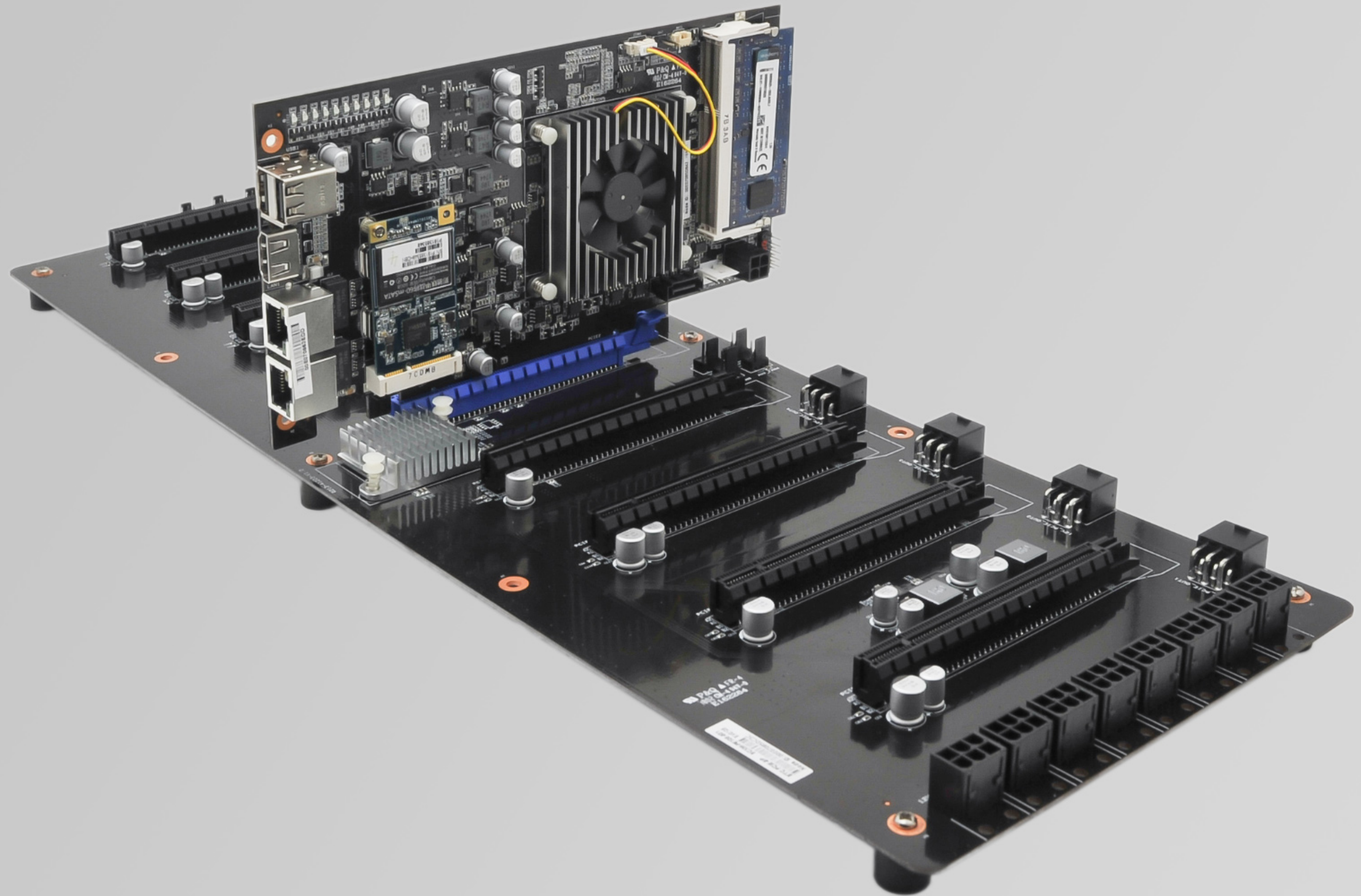
Figure 4.28 Roofline model [Williams et al. 2009]. These rooflines show double-precision floating-point performance in the top row and single-precision performance in the bottom row. (The DP FP performance ceiling is also in the bottom row to give perspective.) The Core i7 920 on the left has a peak DP FP performance of 42.66 GFLOP/sec, a SP FP peak of 85.33 GFLOP/sec, and a peak memory bandwidth of 16.4 GBytes/sec. The NVIDIA GTX 280 has a DP FP peak of 78 GFLOP/sec, SP FP peak of 624 GFLOP/sec, and 127 GBytes/sec of memory bandwidth. The dashed vertical line on the left represents an arithmetic intensity of 0.5 FLOP/byte. It is limited by memory bandwidth to no more than 8 DP GFLOP/sec or 8 SP GFLOP/sec on the Core i7. The dashed vertical line to the right has an arithmetic intensity of 4 FLOP/byte. It is limited only computationally to 42.66 DP GFLOP/sec and 64 SP GFLOP/sec on the Core i7 and 78 DP GFLOP/sec and 512 DP GFLOP/sec on the GTX 280. To hit the highest computation rate on the Core i7 you need to use all 4 cores and SSE instructions with an equal number of multiplies and adds. For the GTX 280, you need to use fused multiply-add instructions on all multithreaded SIMD processors. Guz et al. [2009] have an interesting analytic model for these two architectures.

4 pairs of input data operands
in memory or registers

SIMD ALU

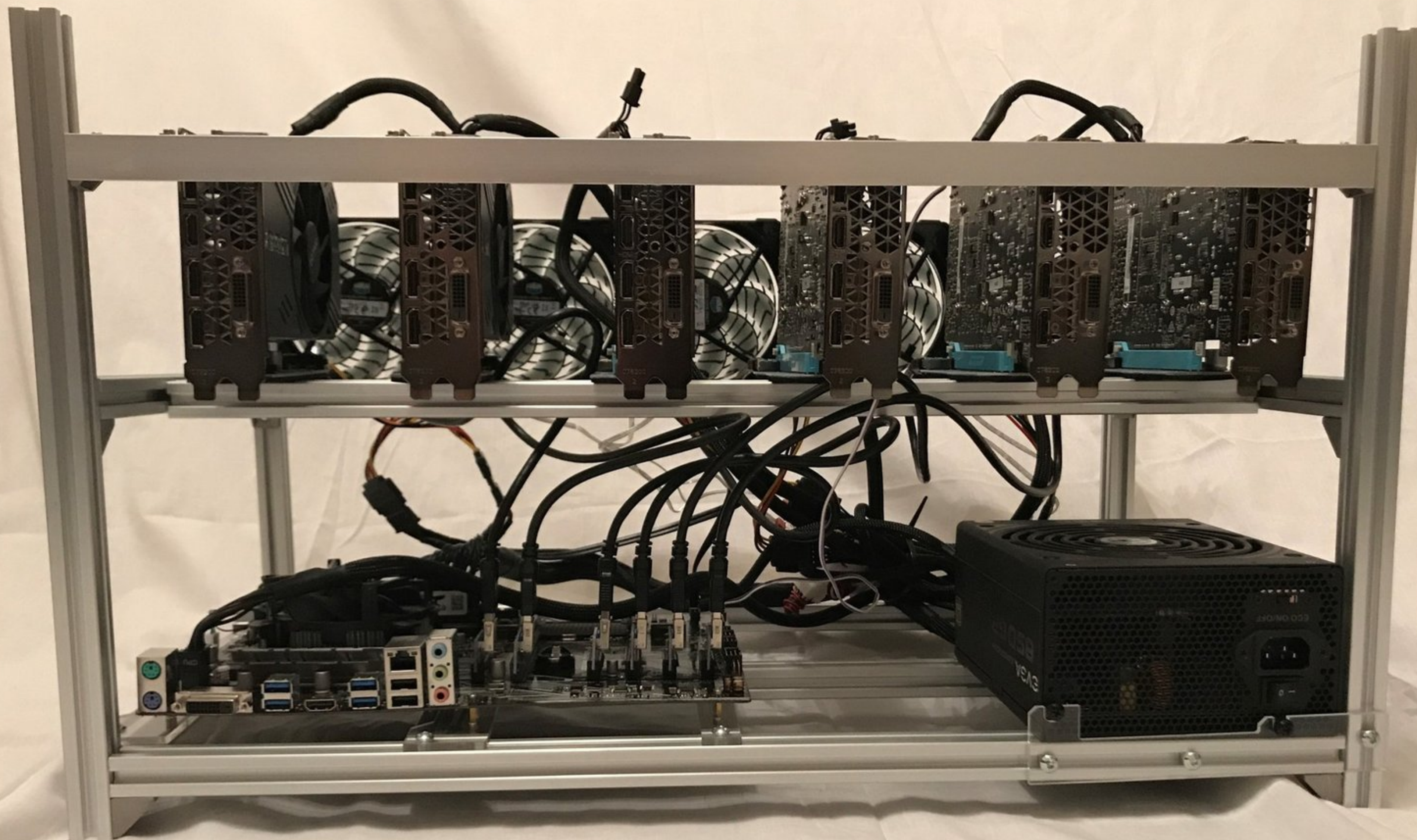


4 SIMD results
produced simultaneously

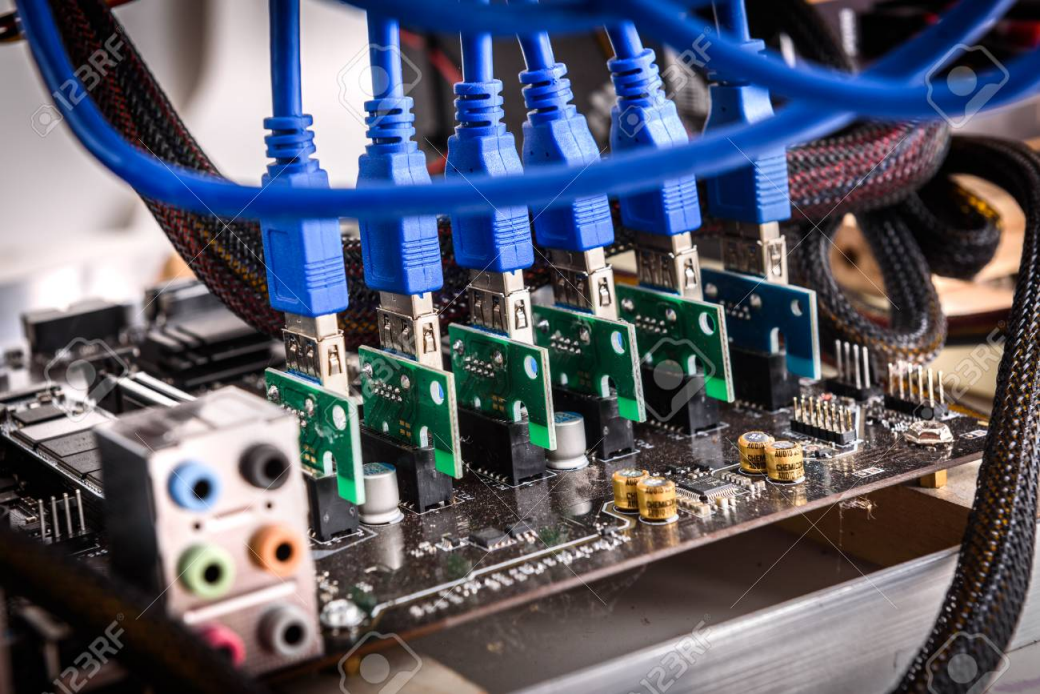














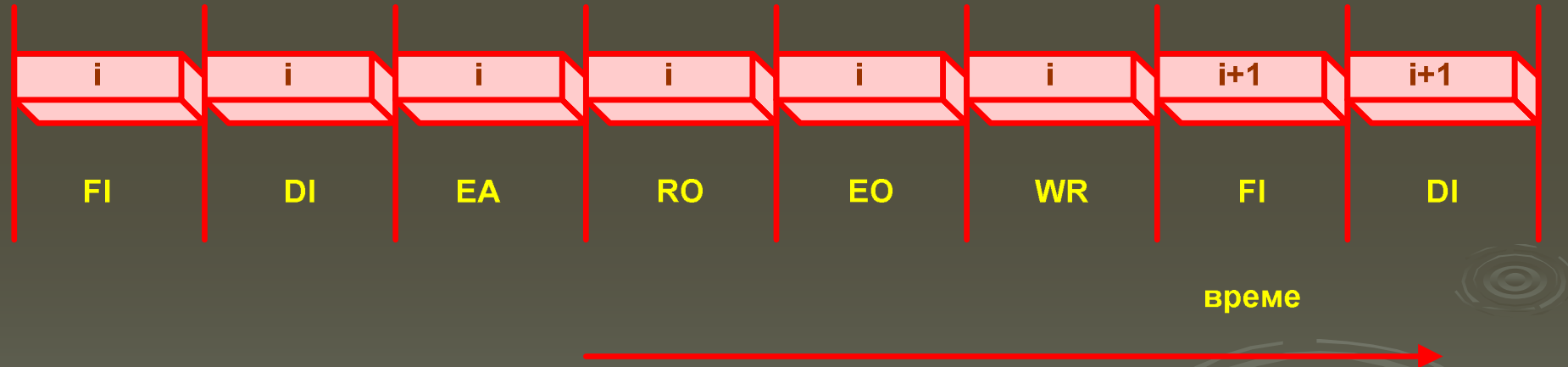
При традиционните финансови системи зад дадена валута (примерно американски долар) стои икономиката на дадена държава или общност от държави. Ако някой посегне на долара, например, при него пристига т.нар. „демокрация“, подкрепена от танкове и самолети, и скоростно доверието към долара се завръща.

[...] Зад повечето криптовалути не стои реална икономика, която да подсигурава тяхната стойност и така липсата на трайна нужда от търсене и предлагане на пазара ги прави неустойчиви и цените им могат да се променят драстично.

[...] Биткойн е балон, ще се спука и много хора ще плачат! Вече обясних защо: зад биткойн няма реална икономика. Няма причина след 5 години биткойн да продължава да се търси.

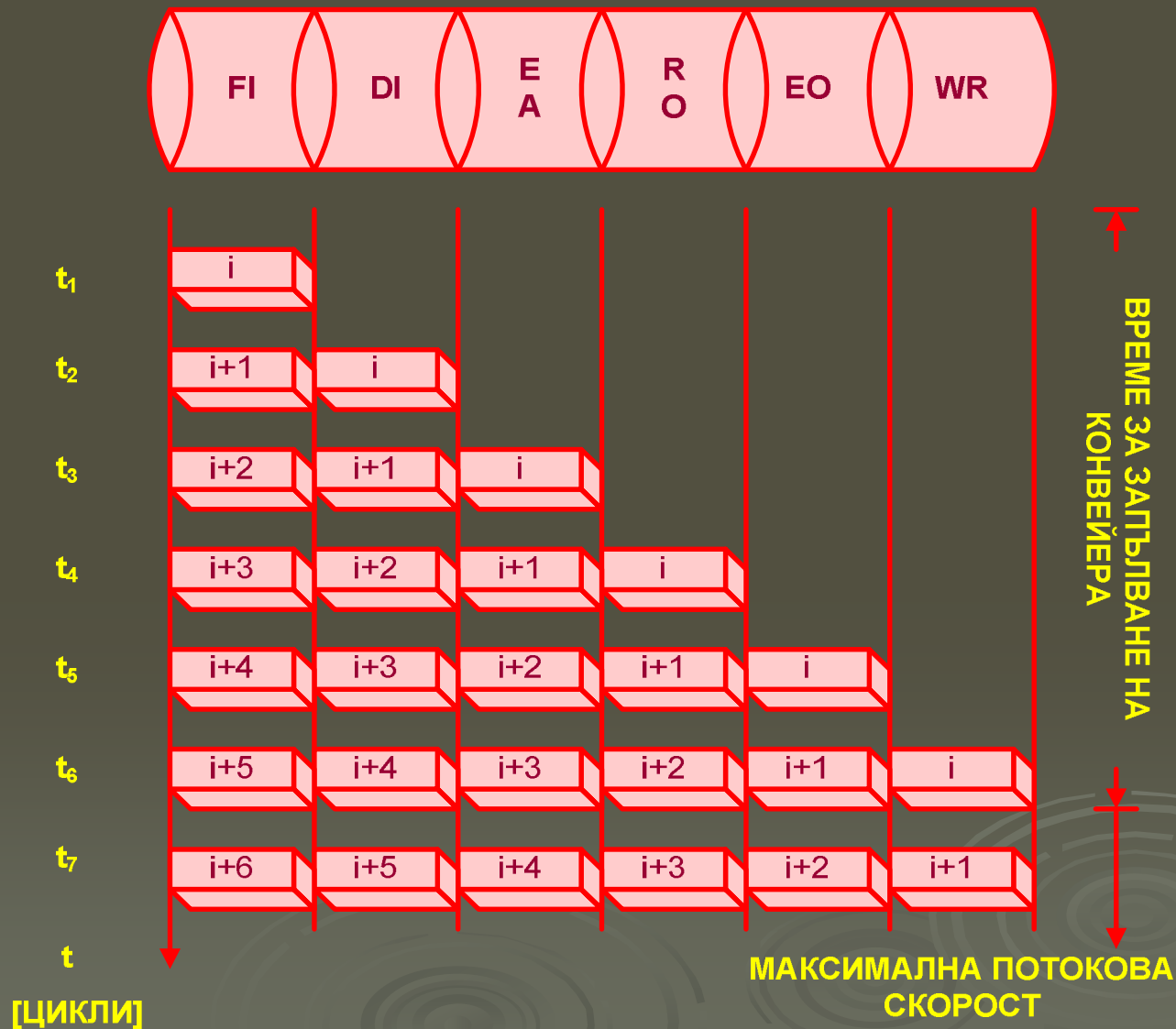
Светлин Наков, ноември 2017 г.

Последователно изпълнение на инструкциите



ИНСТРУКЦИОННИ КОНВЕЙЕРИ

ИНСТРУКЦИОНЕН КОНВЕЙЕР



Производителност на инструкционни конвейери

$$T_{init_pipeline} = (n-1)\tau$$

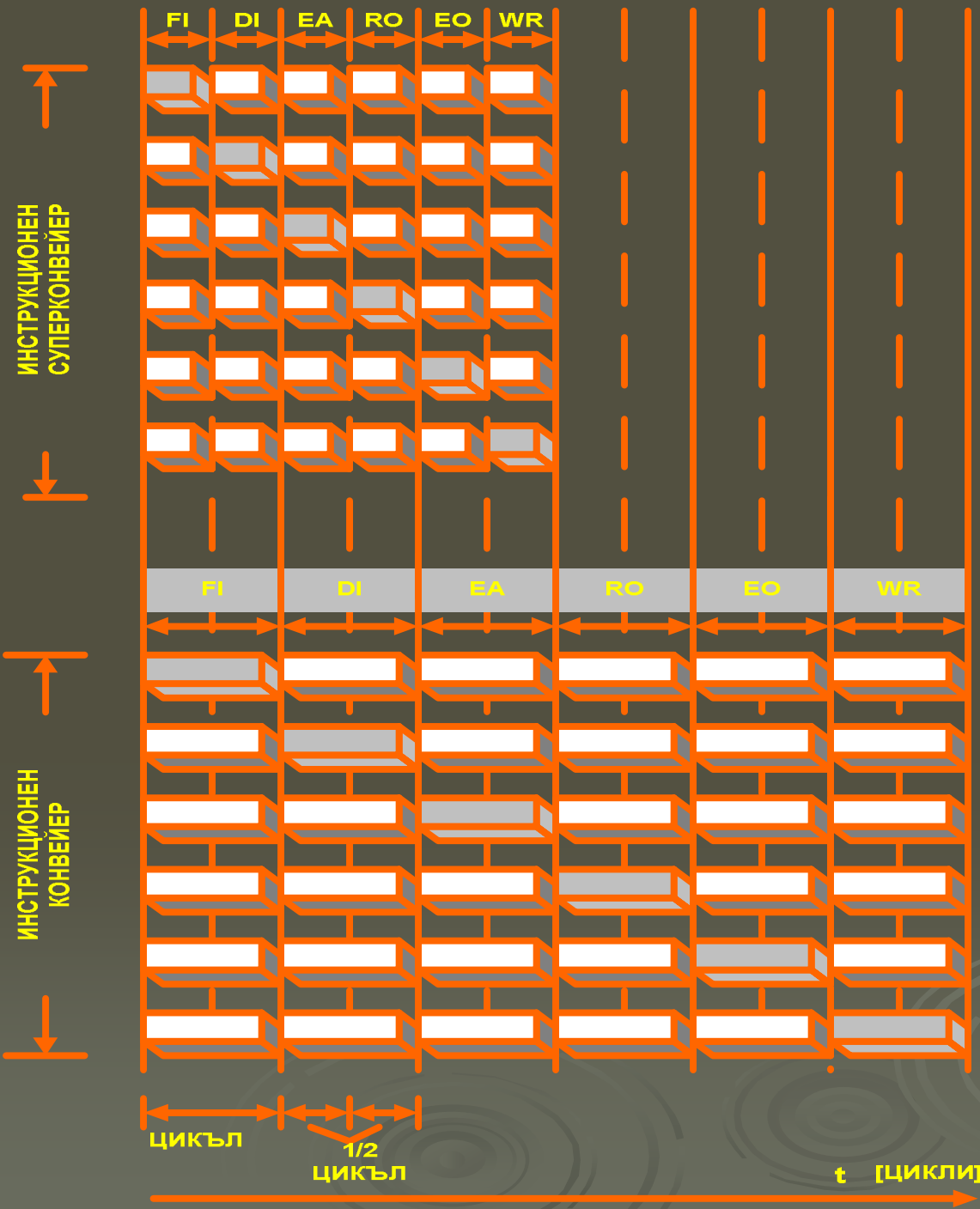
$$S_{up} = \frac{T_{seq}}{T_{pipeline}} = \frac{sn}{n+(s-1)}$$

$$Efficiency = S_{up} / n$$

Ускорението на
инструкционния
конвейер се определя
от броя на фазите му

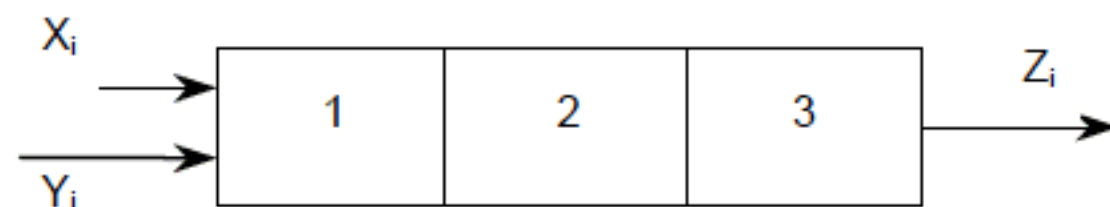
- n -фазен конвейер
- s - # инструкциите в конвейера
- τ - закъснение във фазата на конвейера (цикли)
- $T_{init_pipeline}$ – време за инициализация на конвейера
- T_{seq} – време за последователна обработка на инструкциите
- $T_{pipeline}$ – време за паралелна обработка на инструкциите
- S_{up} - ускорение





Пример. Да се разгледат процесите протичащи в един конвейер за събиране на два вектора с дължина n елемента. Форматът на данните са числа с плаваща запетая.

Базовата функция (в случая събиране на числа с плаваща запетая) може да се разбие на следните подфункции*: а) сравнение на порядъците и изместване на мантисата на числото с по-малък порядък на дясно, на толкова разряда колкото е разликата между двата порядъка; б) сумиране на мантисите; в) нормализация на резултата. Тогава конвейерната обработка може да се реализира чрез 3 степенен конвейер – Фиг.4-1.



Фиг.4-1.Обобщена структурна схема на тристепенен конвейер.

Степени

3

2

1

1

2

3

4

5

Тактове

Z_1

Z_2



Работата на конвейера протича по следния начин:

1-ви такт. Първата двойка числа постъпват за обработка в първата степен на конвейера, където се реализират операциите по подфункция а).

2-ри такт. Получените междинни резултати от първата степен се предават на втората степен, която извършва операциите по реализация на подфункция б). В същото това време първата степен е свободна и може да поеме обработката на следващите двойки числа.

3-ти такт. По времетраенето на третият такт, в третата степен окончателно се формира резултатът за първата двойка числа. Същевременно, първата степен работи по третата двойка числа, а втората е заангажирана с работата по втората двойка числа.

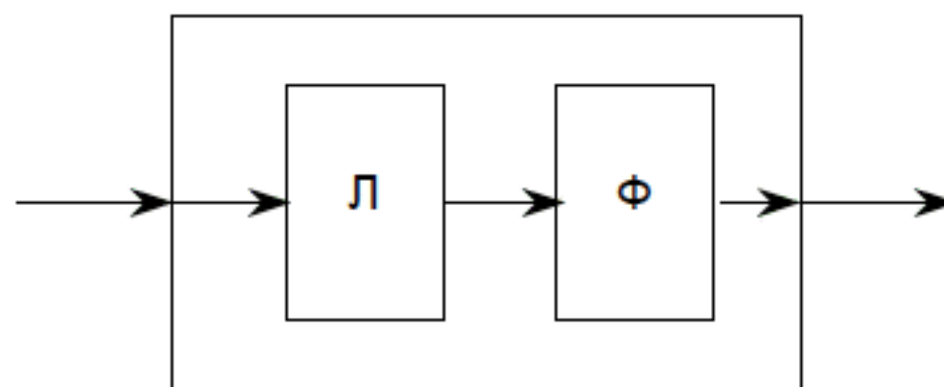
4-ти такт. През време на четвъртия такт, първият резултат напуска конвейера. Третата степен работи по подфункция в) за втората двойка числа; втората степен работи по третата двойка числа; първата степен работи по четвъртата двойка числа.

5-ти такт и т.н.

Конвейерност означава, че се осигурява съвместяване във времето на различни действия по изчисляването на базовите функции за сметка на тяхното разбиване на подфункции. За да се реализира конвейерна обработка е необходимо да са изпълнени следните пет условия:

- Изчислението на базова функция е еквивалентно на някаква последователност от изчисление на подфункции.
- Величините явяващи се входни за дадена подфункция, се явяват изходни за предходната подфункция.
- Никакви други взаимодействия между подфункциите няма освен входните и изходните величини.
- Всяка подфункция се реализира чрез апаратни блокове.
- Действията, реализирани от тези апаратни блокове изискват приблизително едно и също време.

Апаратните средства, необходими за изпълнението на всяка от тези функции, образуват степен. Всяка степен от своя страна е изградена от два блока: логика – (Л) и фиксатор – (Ф) – Фиг.4–3.



По принцип, детайлната работа на конвейера остава скрито за програмиста, пишещ приложни програми. Причини за това са:

- Обикновено се определя единен набор команди за цялото семейство компютри, всеки от които ще изпълнява правилно едни и същи програми, независимо, че способите за реализация на командите могат да бъдат различни.

- По принцип е сложно да се проследява (и управлява програмно) всяка команда в конвейера.

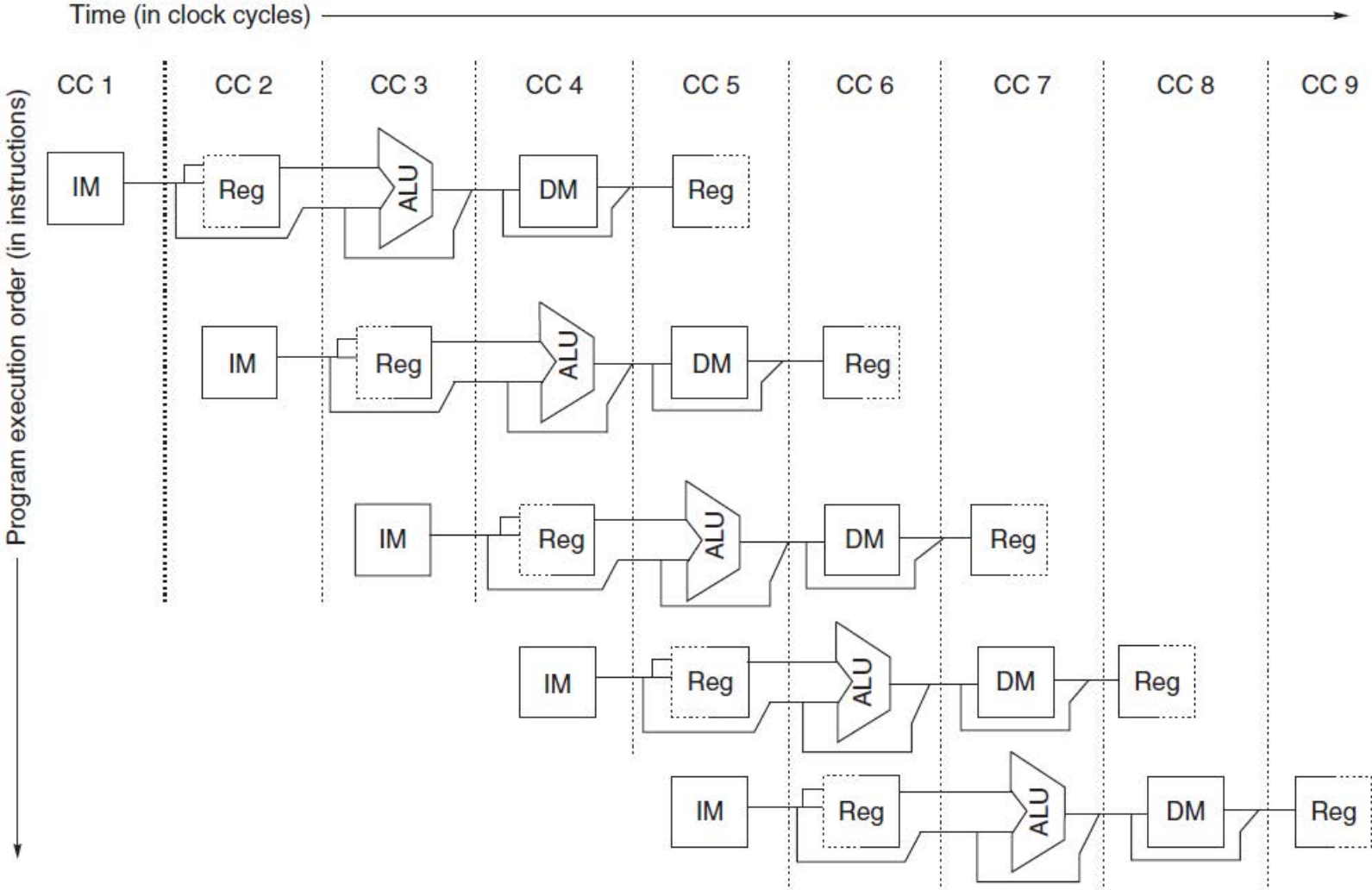


Figure C.2 The pipeline can be thought of as a series of data paths shifted in time. This shows the overlap among the parts of the data path, with clock cycle 5 (CC 5) showing the steady-state situation. Because the register file is used as a source in the ID stage and as a destination in the WB stage, it appears twice. We show that it is read in one part of the stage and written in another by using a solid line, on the right or left, respectively, and a dashed line on the other side. The abbreviation IM is used for instruction memory, DM for data memory, and CC for clock cycle.

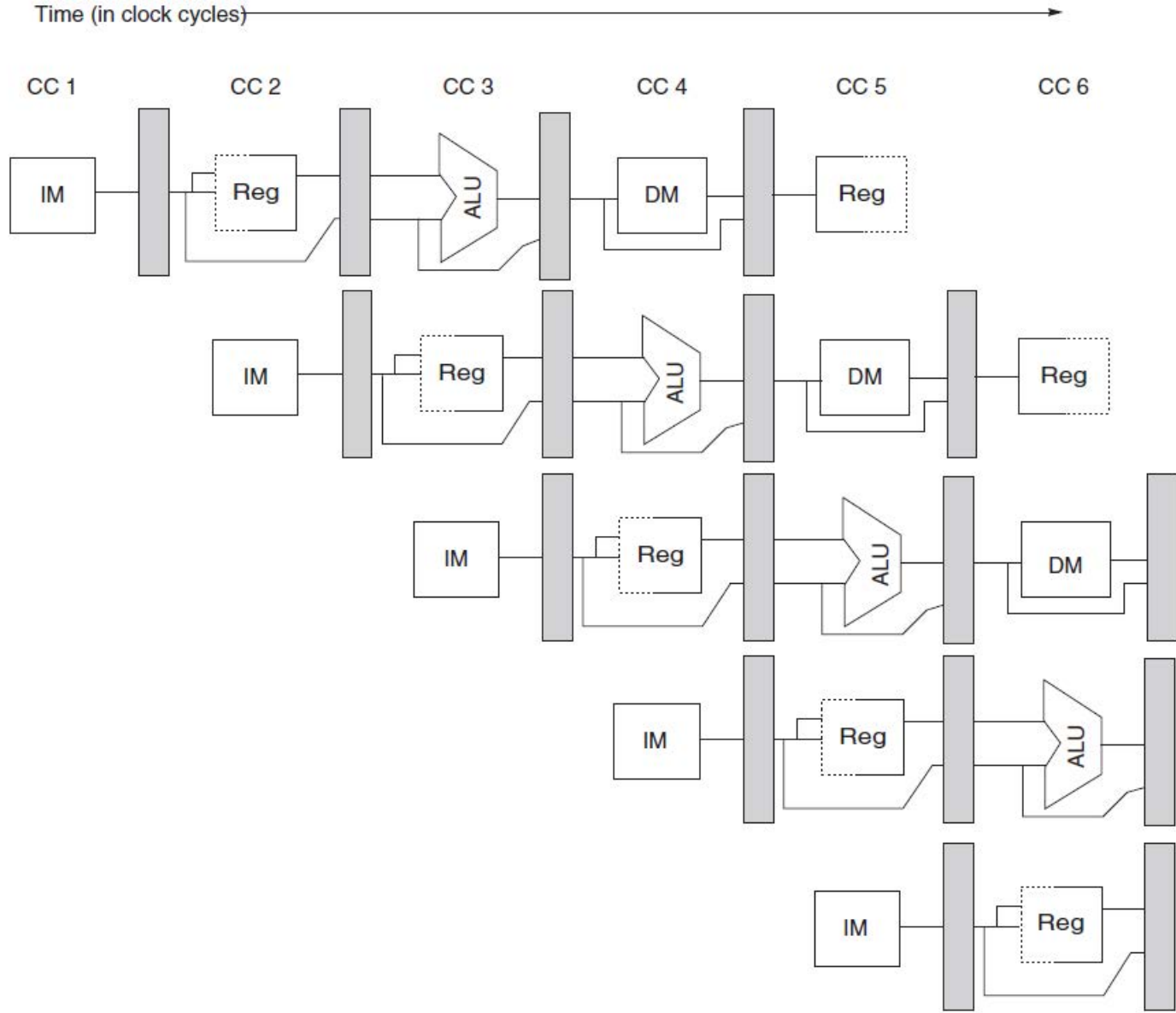


Figure C.3 A pipeline showing the pipeline registers between successive pipeline stages. Notice that the registers prevent interference between two different instructions in adjacent stages in the pipeline. The registers also play the critical role of carrying data for a given instruction from one stage to the other. The edge-triggered property of registers—that is, that the values change instantaneously on a clock edge—is critical. Otherwise, the data from one instruction could interfere with the execution of another!

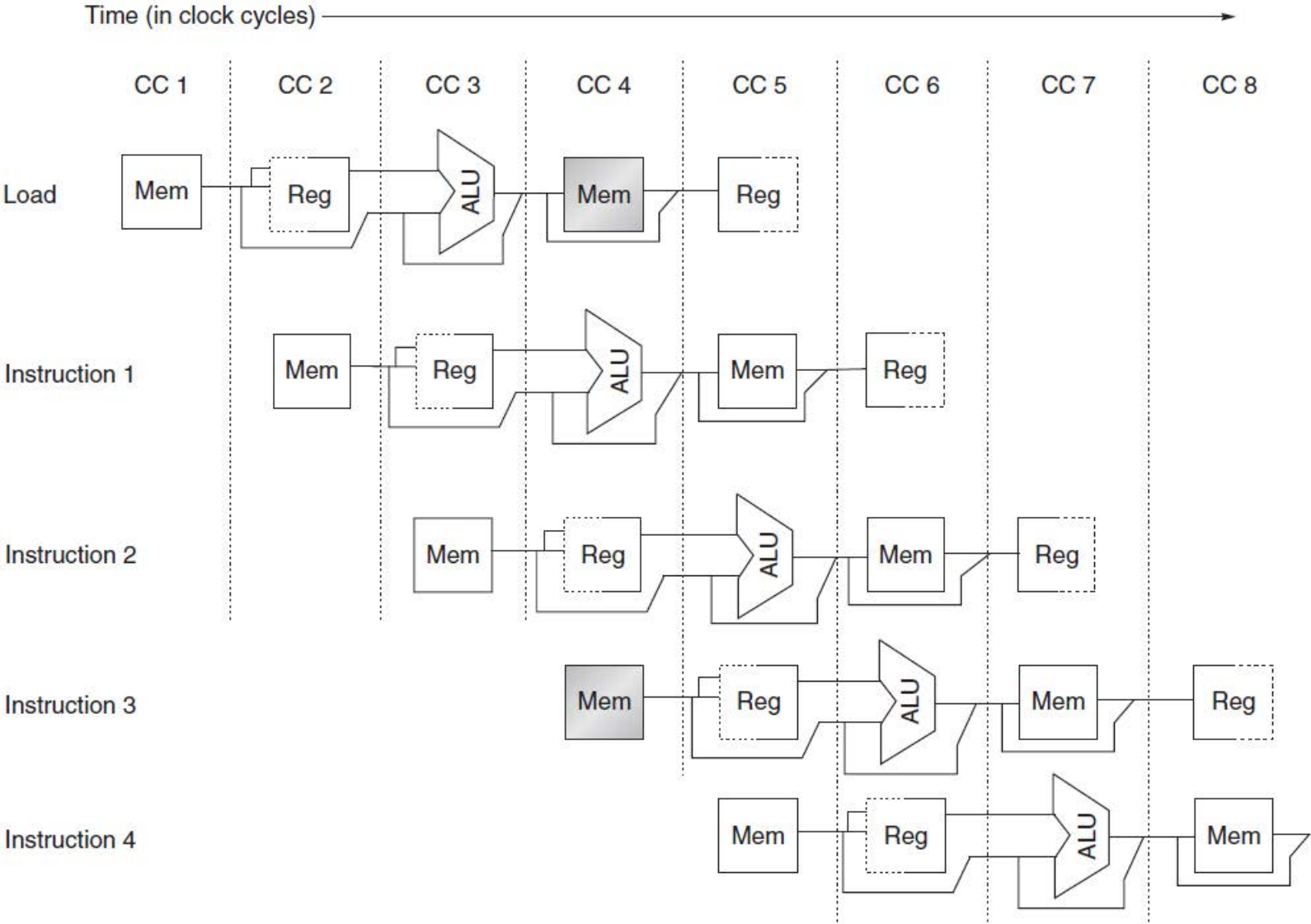


Figure C.4 A processor with only one memory port will generate a conflict whenever a memory reference occurs. In this example the load instruction uses the memory for a data access at the same time instruction 3 wants to fetch an instruction from memory.

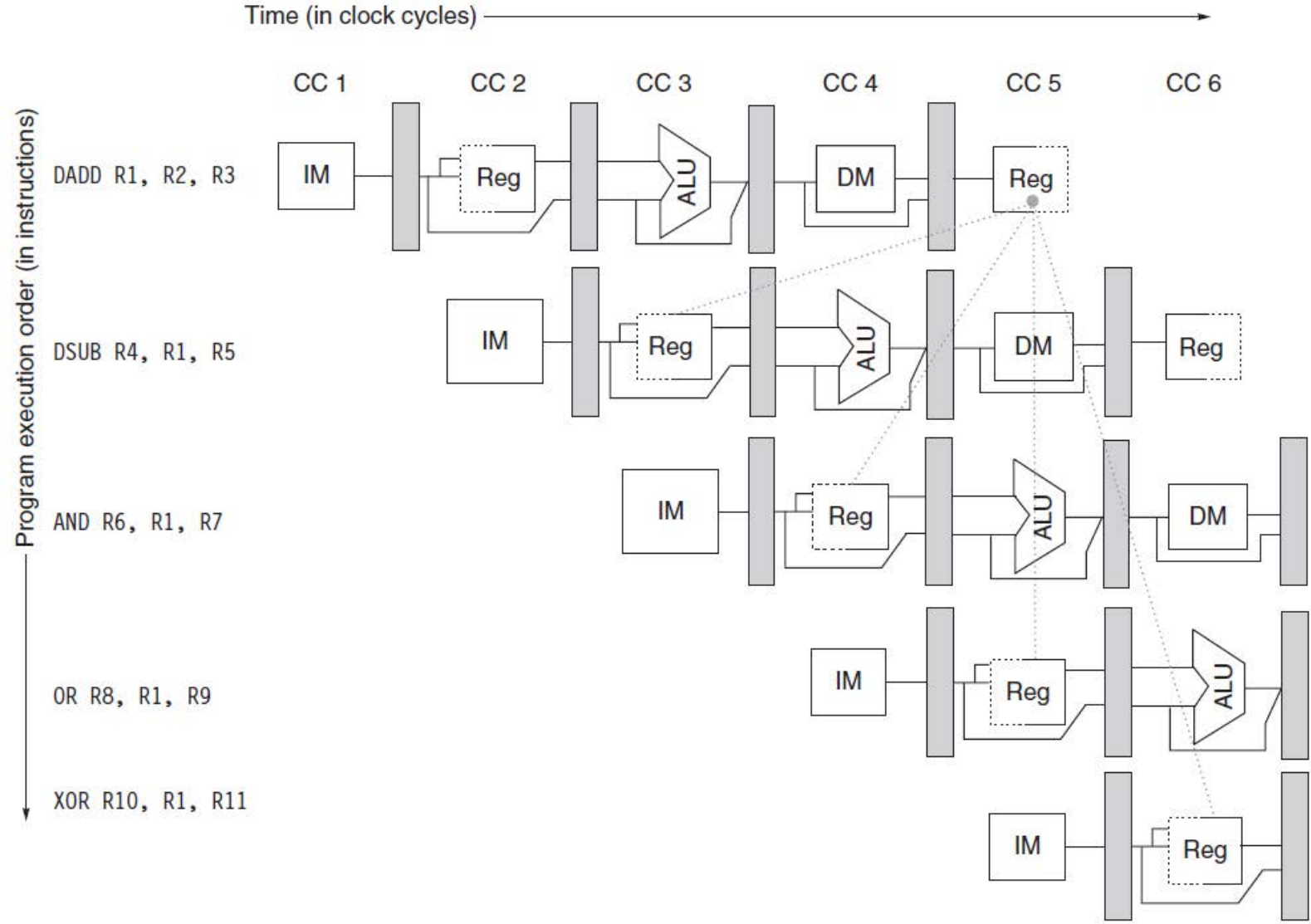


Figure C.6 The use of the result of the DADD instruction in the next three instructions causes a hazard, since the register is not written until after those instructions read it.

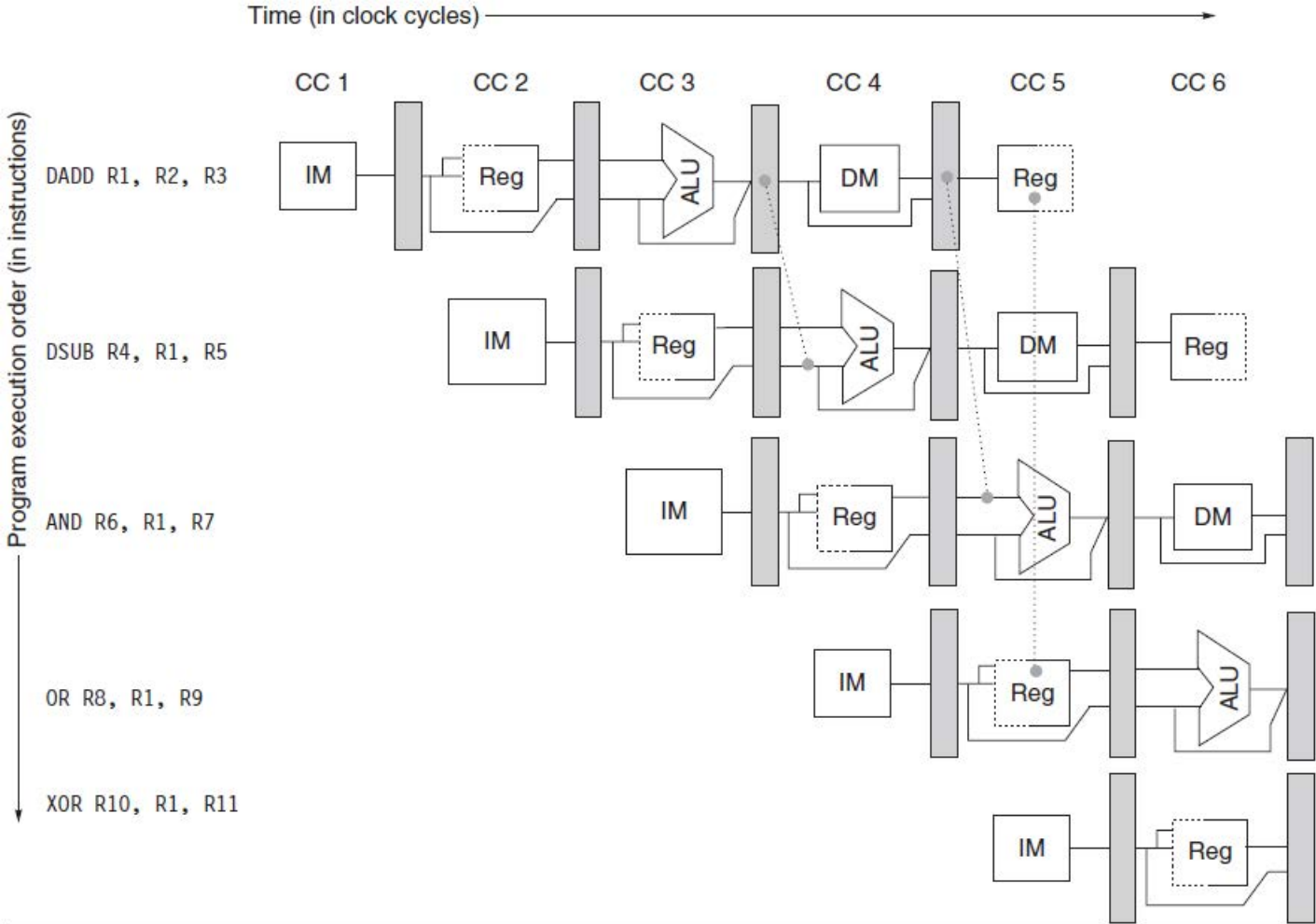


Figure C.7 A set of instructions that depends on the DADD result uses forwarding paths to avoid the data hazard. The inputs for the DSUB and AND instructions forward from the pipeline registers to the first ALU input. The OR receives its result by forwarding through the register file, which is easily accomplished by reading the registers in the second half of the cycle and writing in the first half, as the dashed lines on the registers indicate. Notice that the forwarded result can go to either ALU input; in fact, both ALU inputs could use forwarded inputs from either the same pipeline register or from different pipeline registers. This would occur, for example, if the AND instruction was AND R6, R1, R4.

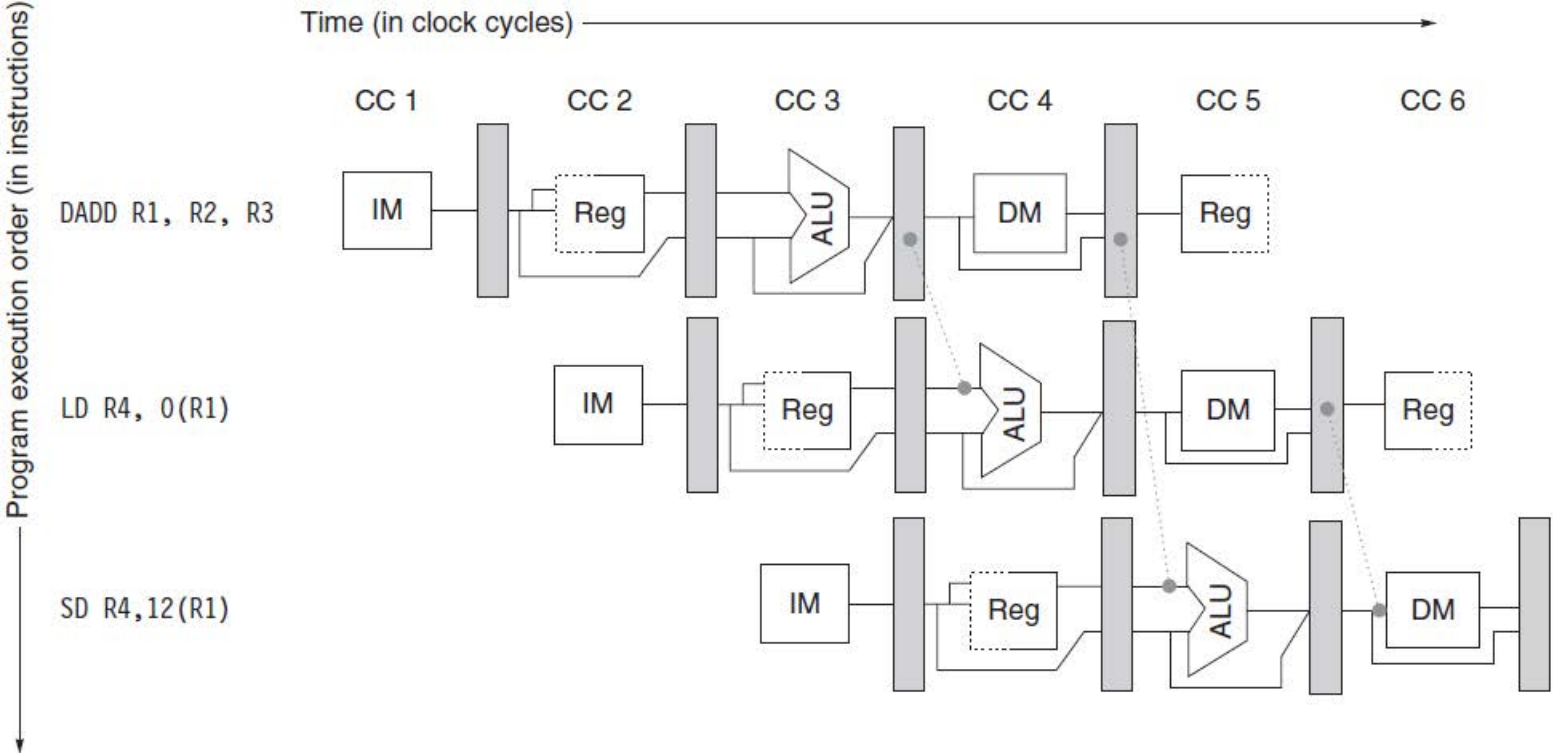


Figure C.8 Forwarding of operand required by stores during MEM. The result of the load is forwarded from the memory output to the memory input to be stored. In addition, the ALU output is forwarded to the ALU input for the address calculation of both the load and the store (this is no different than forwarding to another ALU operation). If the store depended on an immediately preceding ALU operation (not shown above), the result would need to be forwarded to prevent a stall.

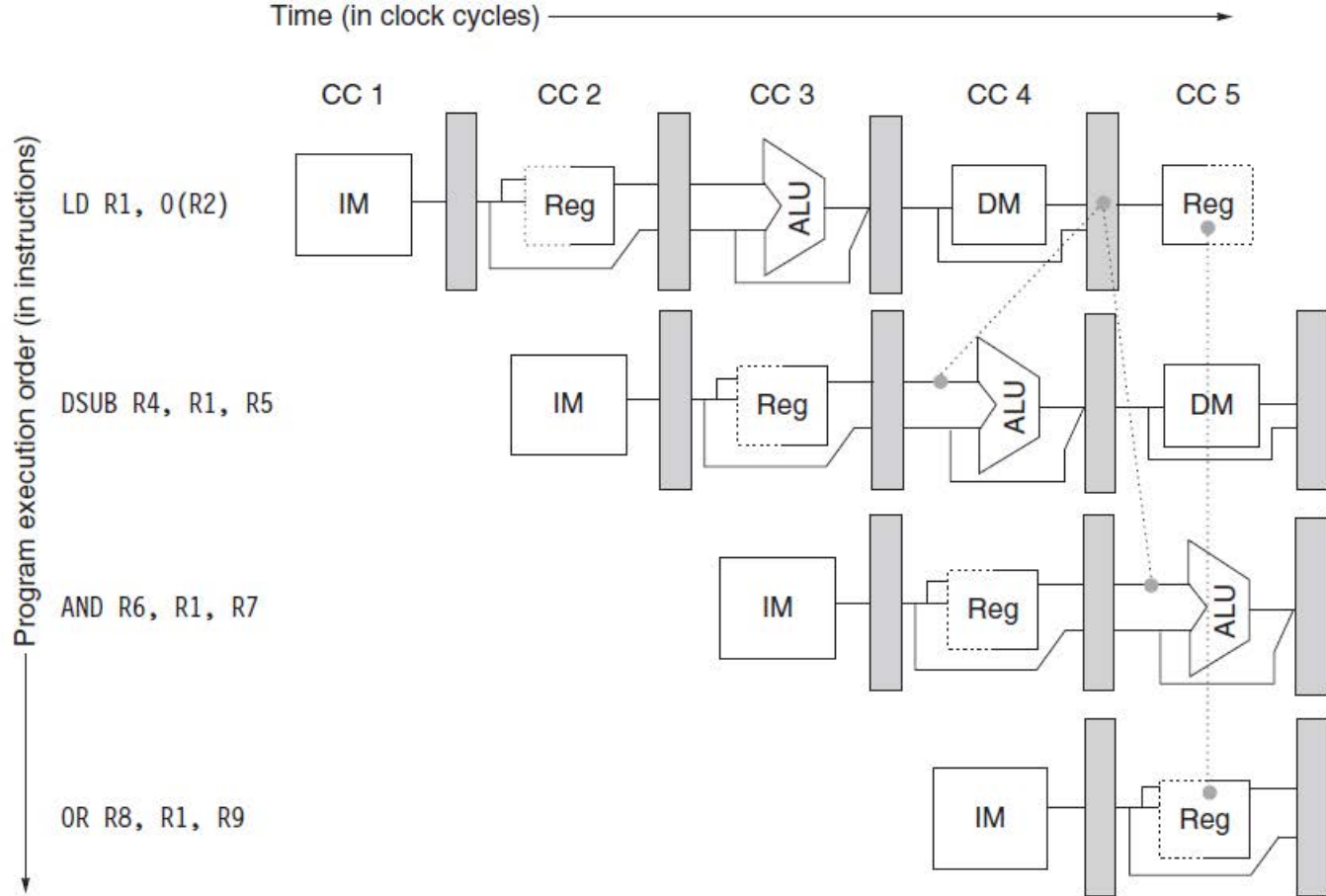


Figure C.9 The load instruction can bypass its results to the AND and OR instructions, but not to the DSUB, since that would mean forwarding the result in “negative time.”

Зависимости по данни

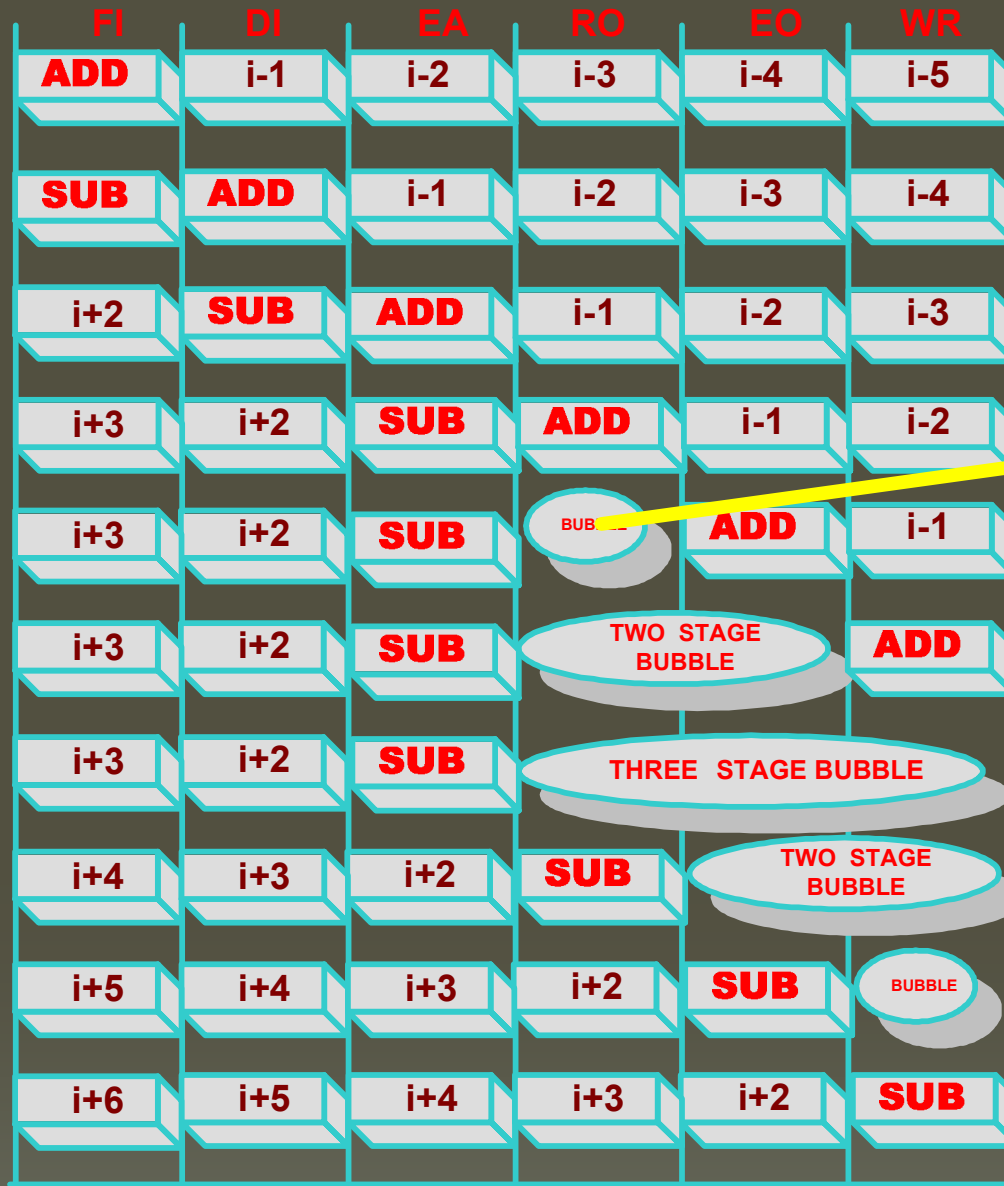
Data Dependencies

(read-after-write hazard)

- Резултатът на i -тата инструкция се използва за операнд на $(i+1)$ -та инструкция
- **Типове зависимости по данни:**
 1. **Потокови (flow) зависимости по данни**
 2. **Антизависимости (Antidependency)**
 3. **Зависимости по изход (Output dependency)**



FLOW DATA DEPENDENCIES



PIPELINE STALLS

ADD R3, R2, R1 ; **R3** = R2 + R1
SUB R4, **R3**, 1 ; **R4** = **R3** - 1



Antidependencies (write-after-read hazards)

- Инструкцията осъществява запис в регистър (или адрес в паметта), съдържанието на който се чете от предходна инструкция

```
1.  ADD R3, R2, R1 ; R3 = R2 + R1  
2.  SUB R2, R3, 1   ; R2 = R3 - 1
```

Output Dependencies (write-after-write hazard)

- 
1. **ADD** *R3*, *R2*, *R1* ; *R3* = *R2* + *R1*
 2. **SUB** *R2*, *R3*, 1 ; *R2* = *R3* - 1
 3. **ADD** *R3*, *R2*, *R5*; *R3* = *R2* + *R5*

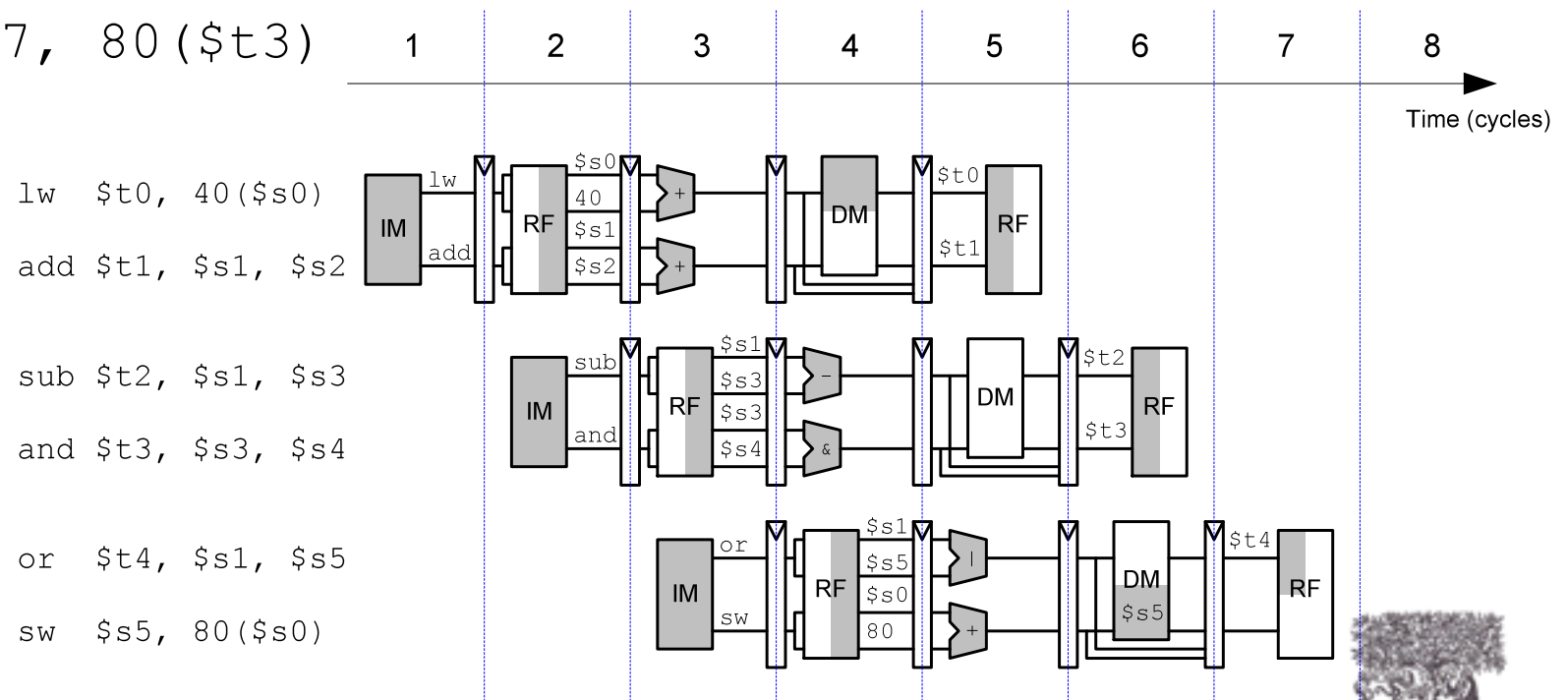
- По своята същност е вид ресурсен конфликт
- Регистър *R3* се използва от няколко последователни инструкции
- Елиминиране на този **КОНФЛИКТ**: да се използва друг регистър

Superscalar Example

```
lw    $t0, 40($s0)
add   $t1, $t0, $s1
sub   $t0, $s2, $s3
and   $t2, $s4, $t0
or    $t3, $s5, $s6
sw    $s7, 80($t3)
```

Ideal IPC: 2

Actual IPC: 2



Superscalar with Dependencies

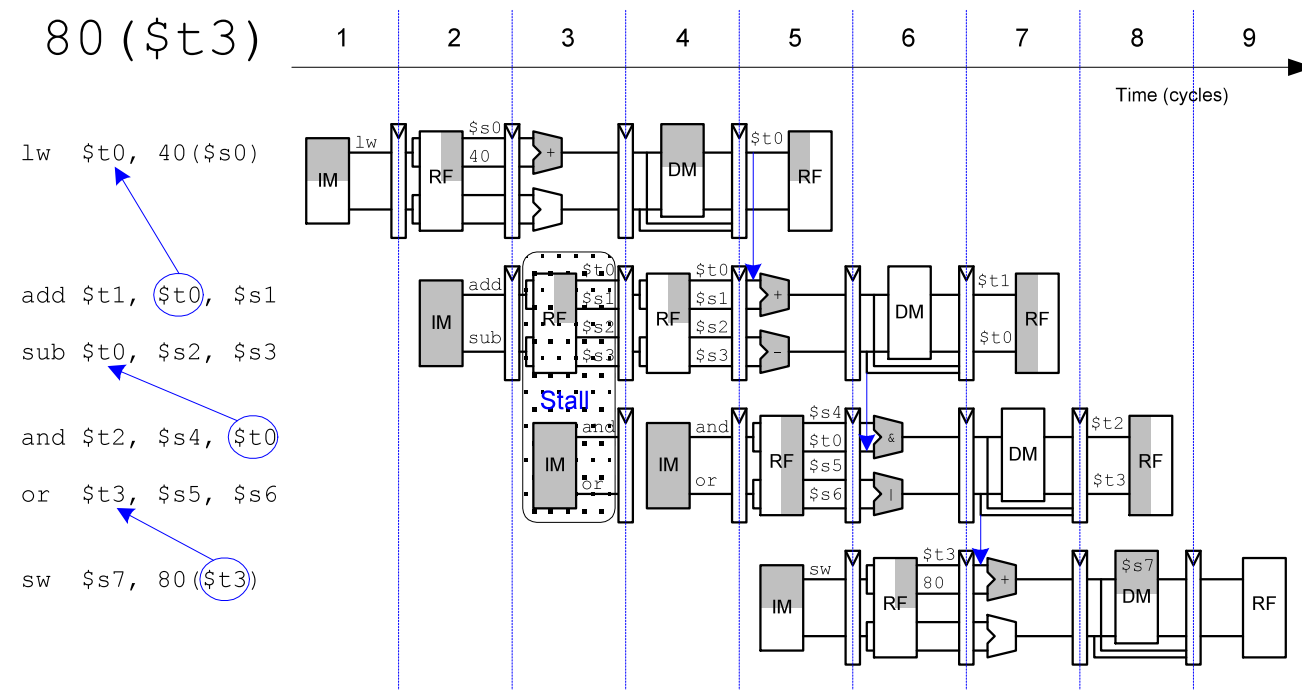
```
lw    $t0, 40($s0)
add   $t1, $t0, $s1
sub   $t0, $s2, $s3
and   $t2, $s4, $t0
or    $t3, $s5, $s6
sw    $s7, 80($t3)
```

Ideal IPC:

2

Actual IPC:

6/5 = 1.2



Out of Order Processor Example

```

lw    $t0, 40($s0)
add   $t1, $t0, $s1
sub   $t0, $s2, $s3
and   $t2, $s4, $t0
or    $t3, $s5, $s6
sw    $s7, 80($t3)

```

Ideal IPC:

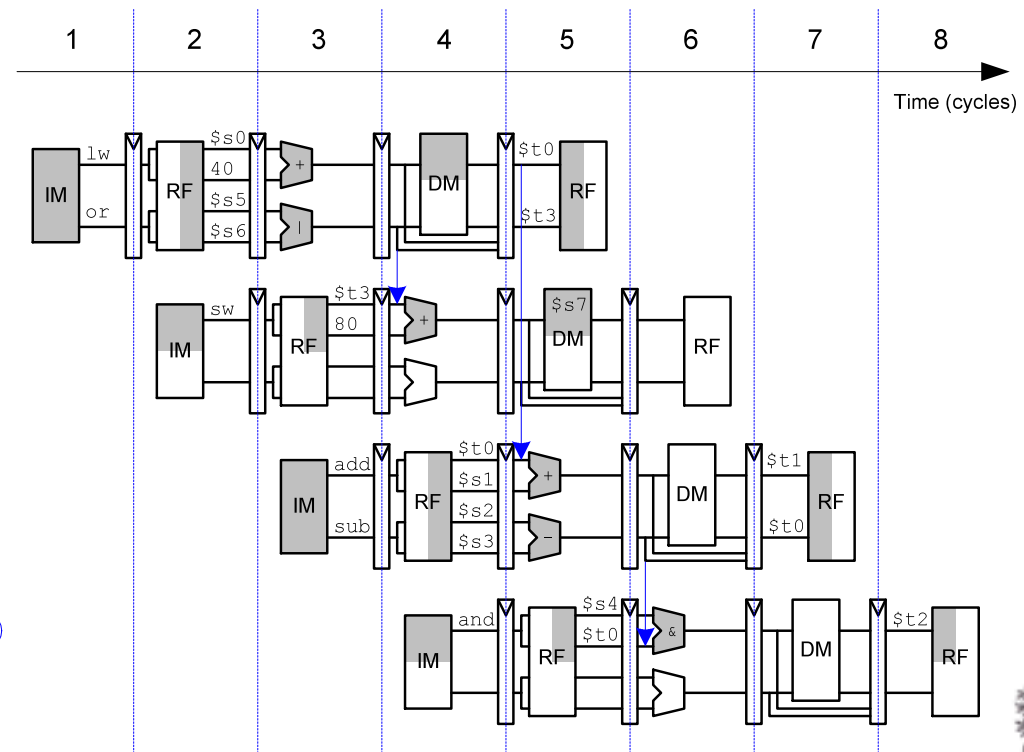
2

Actual IPC:

$6/4 = 1.5$

lw \$t0, 40(\$s0)
 or \$t3, \$s5, \$s6
 sw \$s7, 80(\$t3)
 add \$t1, \$t0, \$s1
 sub \$t0, \$s2, \$s3
 and \$t2, \$s4, \$t0

RAW
 two cycle latency
 between load and
 use of \$t0
 WAR
 RAW



Register Renaming

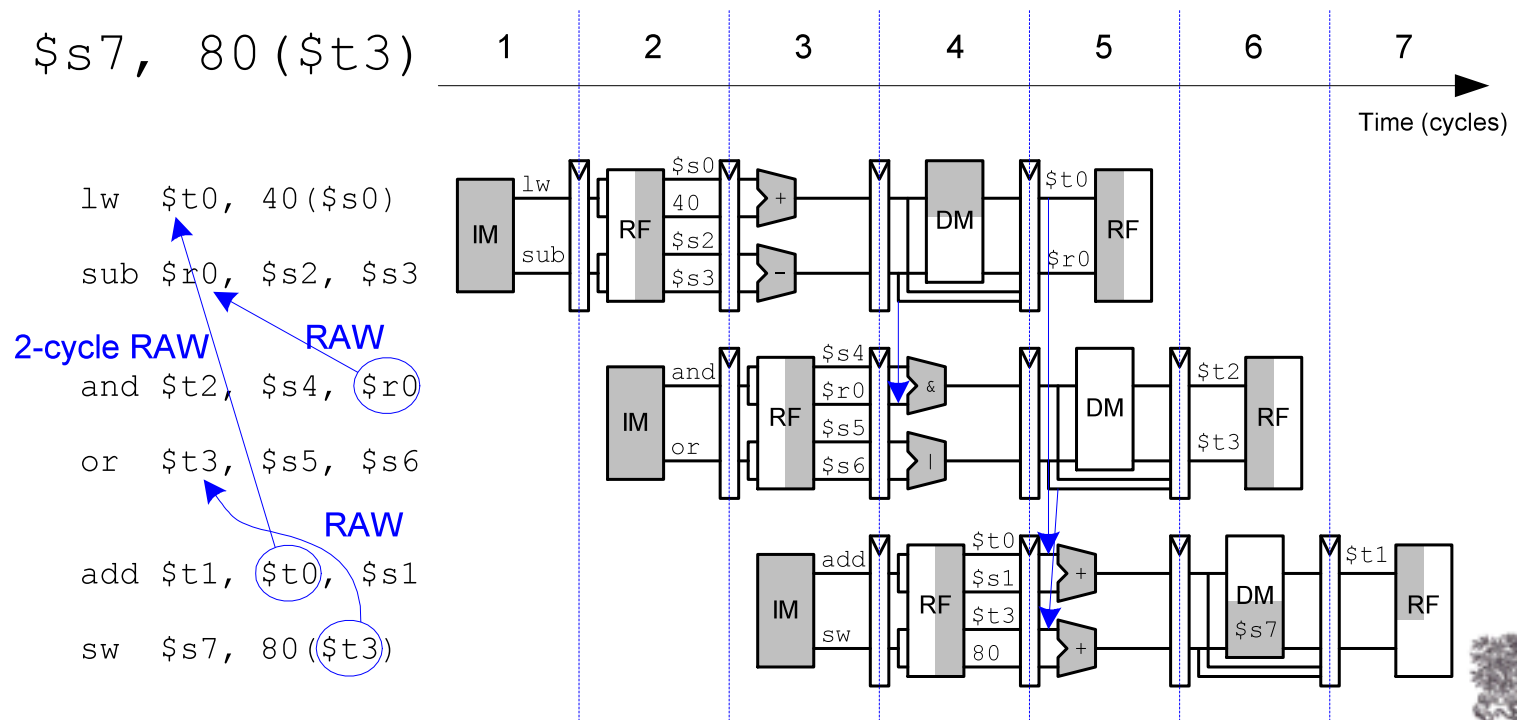
```
lw    $t0, 40($s0)
add   $t1, $t0, $s1
sub   $t0, $s2, $s3
and   $t2, $s4, $t0
or    $t3, $s5, $s6
sw    $s7, 80($t3)
```

Ideal IPC:

2

Actual IPC:

$6/3 = 2$

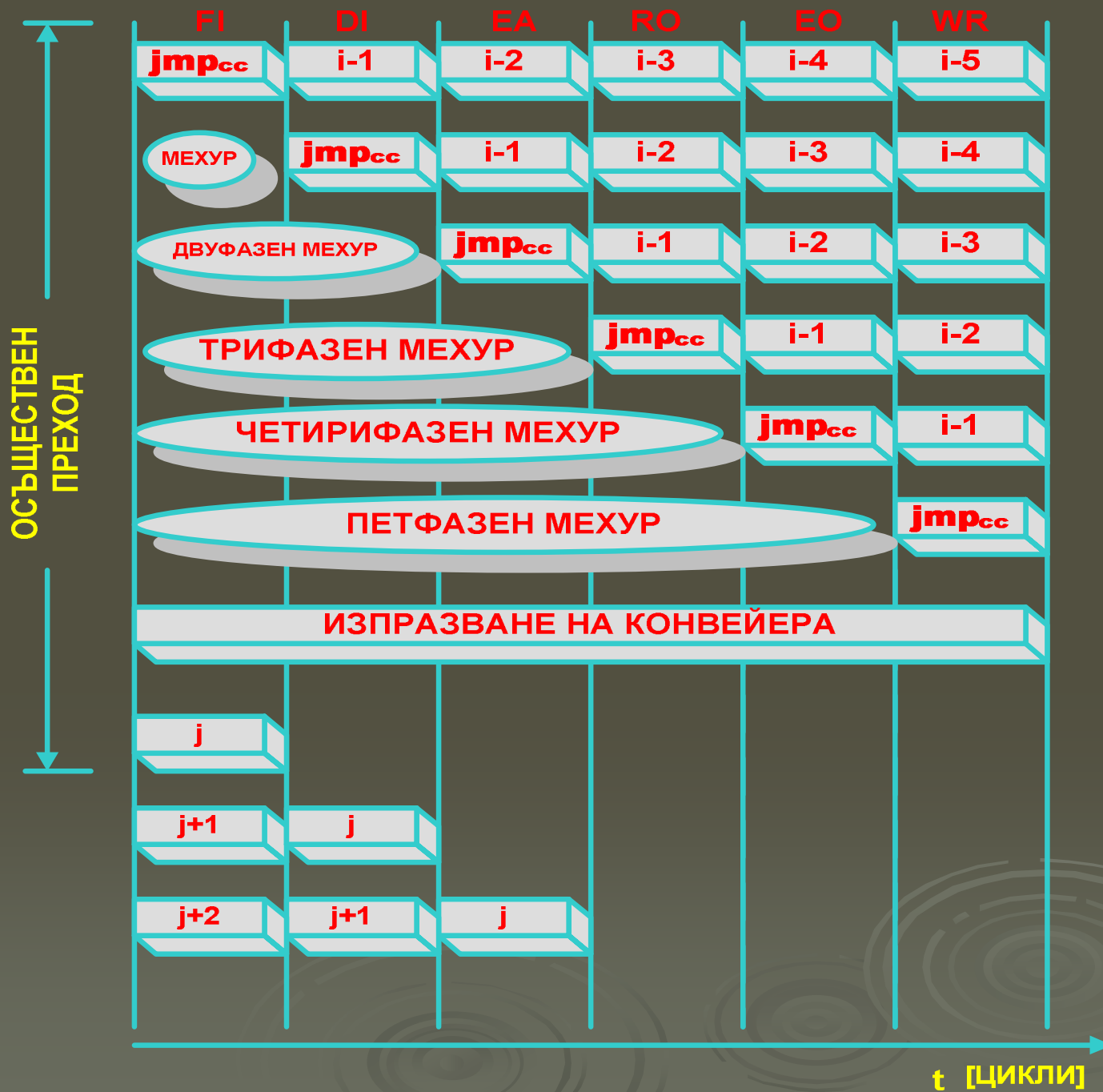



```
int i, fact = 1;
for (i = 1; i <= 12; i++) {
    fact *= i;
}
return fact;
```

```
int i, factO = 1, factE = 1;
for (i = 1; i <= 12; i += 2) {
    factO *= i;
    factE *= i + 1;
}
return factO * factE;
```

Горната промяна (вдясно) води до намаляване на времето за изпълнение **до два пъти** при новите суперскаларни процесори.

—*Orion Sky Lawlor*



Стратегии

- Дублиране на инструкционните буфери за извличане на двете възможни инструкции – **множествено предварително извличане на инструкции** или *branch bypassing*
- Логика за динамично предсказване на прехода – извличане на най-вероятната инструкция след инструкция *branch*
- Статично предсказване на прехода
- Инструкции за отложен преход (*Delayed branch instructions*)



Hardware-Based Speculation

- Execute instructions along predicted execution paths but only commit the results if prediction was correct
- Instruction commit: allowing an instruction to update the register file when instruction is no longer speculative
- Need an additional piece of hardware to prevent any irrevocable action until an instruction commits
 - I.e. updating state or taking an execution

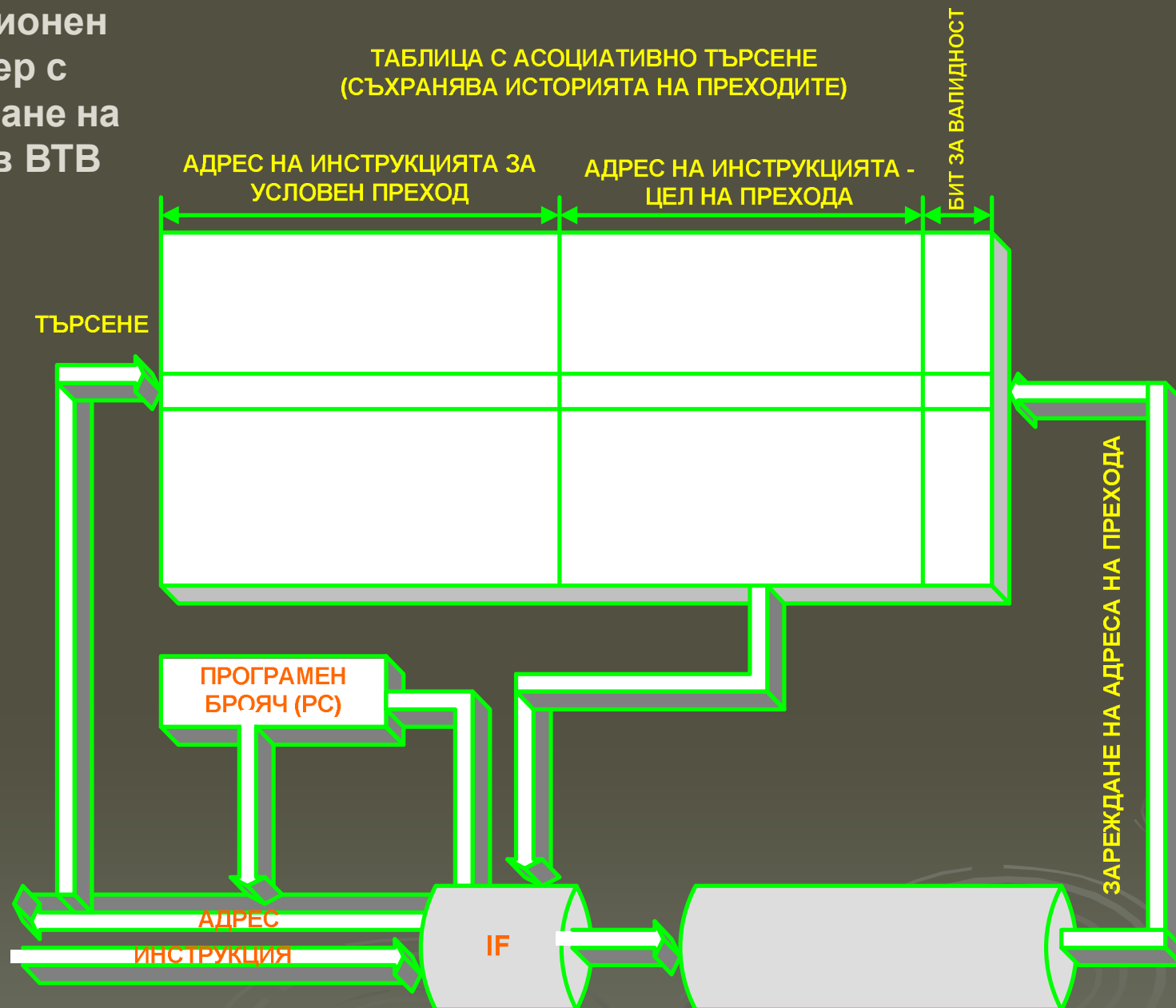
Логика за предсказване на прехода и история на преходите (Prediction Logic and Branch History)

- Динамично предсказване базирано на историята в хода на изпълнение на програмата (*run-time execution history*)
- *Prediction Look-up Table* (таблица за предсказване на преходите) – адрес на инструкцията на прехода, адрес на прехода, бит за предходно предсказване
- *Branch Target Buffer* (Буфер за историята на преходите) – имплементира се с кеш (напълно асоциативен или асоциативен с множества)



Инструкционен конвейер с предсказване на прехода в ВТВ

ТАБЛИЦА С АСОЦИАТИВНО ТЪРСЕНЕ (СЪХРАНЯВА ИСТОРИЯТА НА ПРЕХОДИТЕ)



Ресурсни конфликти

- *Едновременни заявки* за едни и същи ресурси (памет или функционални устройства)
- Разрешават се чрез *дублиране на ресурсите*
- Елиминирание на *конфликтите за памет* :
 1. Специализирано функционално устройство за четене на операндите или запис на резултатите в паметта (load/store)
 2. Осигуряване на отделни кешове за инструкции и данни на ниво L1



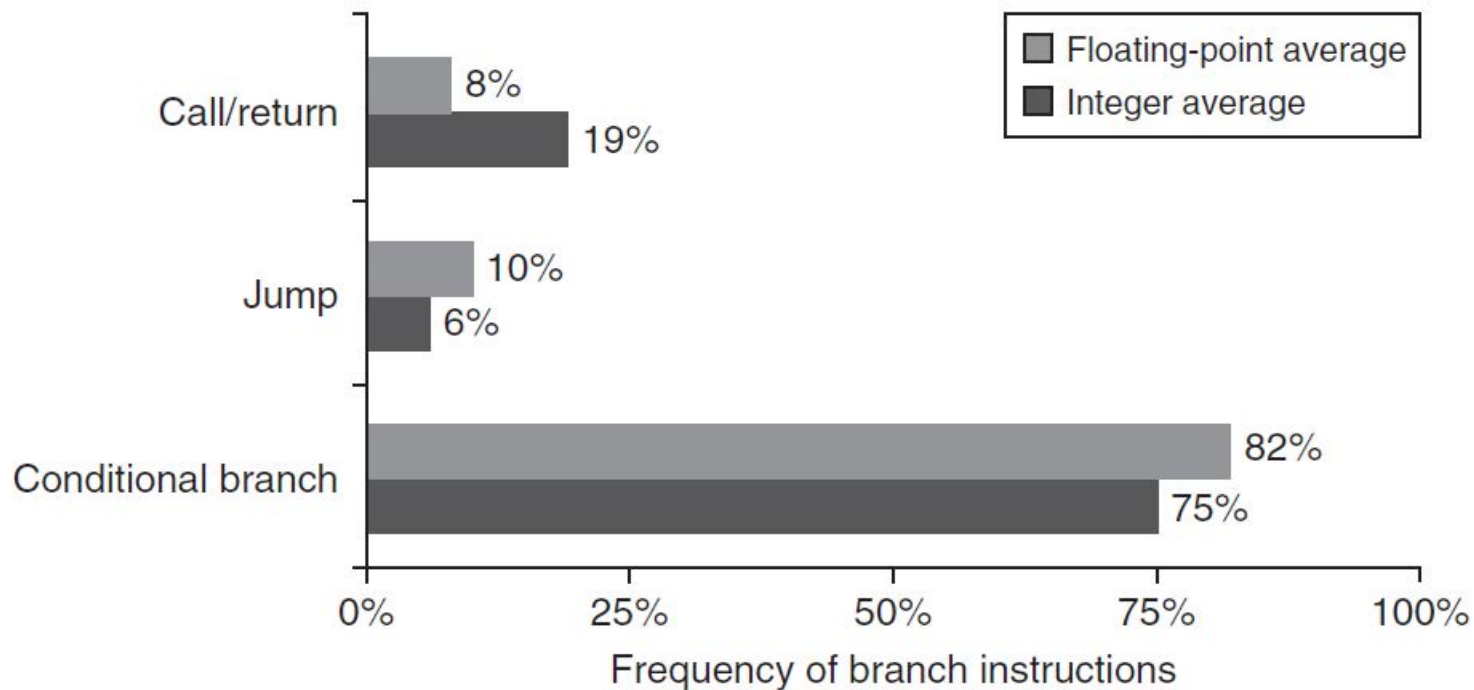
Процедурни зависимости

- *Редът на изпълнение* не е известен преди изпълнението на инструкциите
- Основна причина – *инструкции за безусловен и условен преход - branch*
- *Инструкции за безусловен преход* → всички инструкции в конвейера се игнорират и *конвейерът се изпразва (the pipeline stalls)*
- *Инструкции за условен преход* → същият ефект ако преходът се осъществи

Статично предсказване

- Прогнозата се прави преди изпълнението
- Форматът на инструкцията за преход съдържа *допълнителен бит* за предсказване на прехода
- Добавя се множество от нови инструкции за преход: *избери програмния брояч PC или избери адреса на прехода*
- *Отговорност на компилатора*
- *Атрактивен подход за RISC архитектури*





Return Address Predictor

- Most unconditional branches come from function returns
- The same procedure can be called from multiple sites
 - Causes the buffer to potentially forget about the return address from previous calls
- Create return address buffer organized as a stack

Energy Efficiency

- Value prediction
 - Uses:
 - Loads that load from a constant pool
 - Instruction that produces a value from a small set of values
 - Not incorporated into modern processors
 - Similar idea--*address aliasing prediction*--is used on some processors to determine if two stores or a load and a store reference the same address to allow for reordering

Normal vs. Delayed Branch

Address	Normal	Delayed
100	LOAD X,A	LOAD X,A
101	ADD 1,A	ADD 1,A
102	JUMP 105	JUMP 106
103	ADD A,B	NOOP
104	SUB C,B	ADD A,B
105	STORE A,Z	SUB C,B
106		STORE A,Z

One delay slot - Next instruction is always in the pipeline.

“Normal” path contains an implicit “NOP” instruction as the pipeline gets flushed. Delayed branch requires explicit NOP instruction placed in the code!

Optimized Delayed Branch

But we can optimize this code by rearrangement! Notice we always Add 1 to A so we can use this instruction to fill the delay slot

Address	Normal	Delayed	Optimized
100	LOAD X,A	LOAD X,A	LOAD X,A
101	ADD 1,A	ADD 1,A	JUMP 105
102	JUMP 105	JUMP 106	ADD 1,A
103	ADD A,B	NOOP	ADD A,B
104	SUB C,B	ADD A,B	SUB C,B
105	STORE A,Z	SUB C,B	STORE A,Z
106		STORE A,Z	

Инструкции за отложен преход

Delayed Branch Instructions

- Запълване на мехурите в конвейера с инструкции, които не зависят от това, дали ще се осъществи прехода (или поне мехурите се запълват с NOP)

З
а
п
ъ
л
в
а
н
е
н
а
м
е
х
у
р
и
т
е

ADD R3, R4, R5
SUB R2, R2, 1
BEQ R2, R0, L1

L1:

SUB R2, R2, 1
BEQD R2, R0, L1
ADD R3, R4, R5

L1:



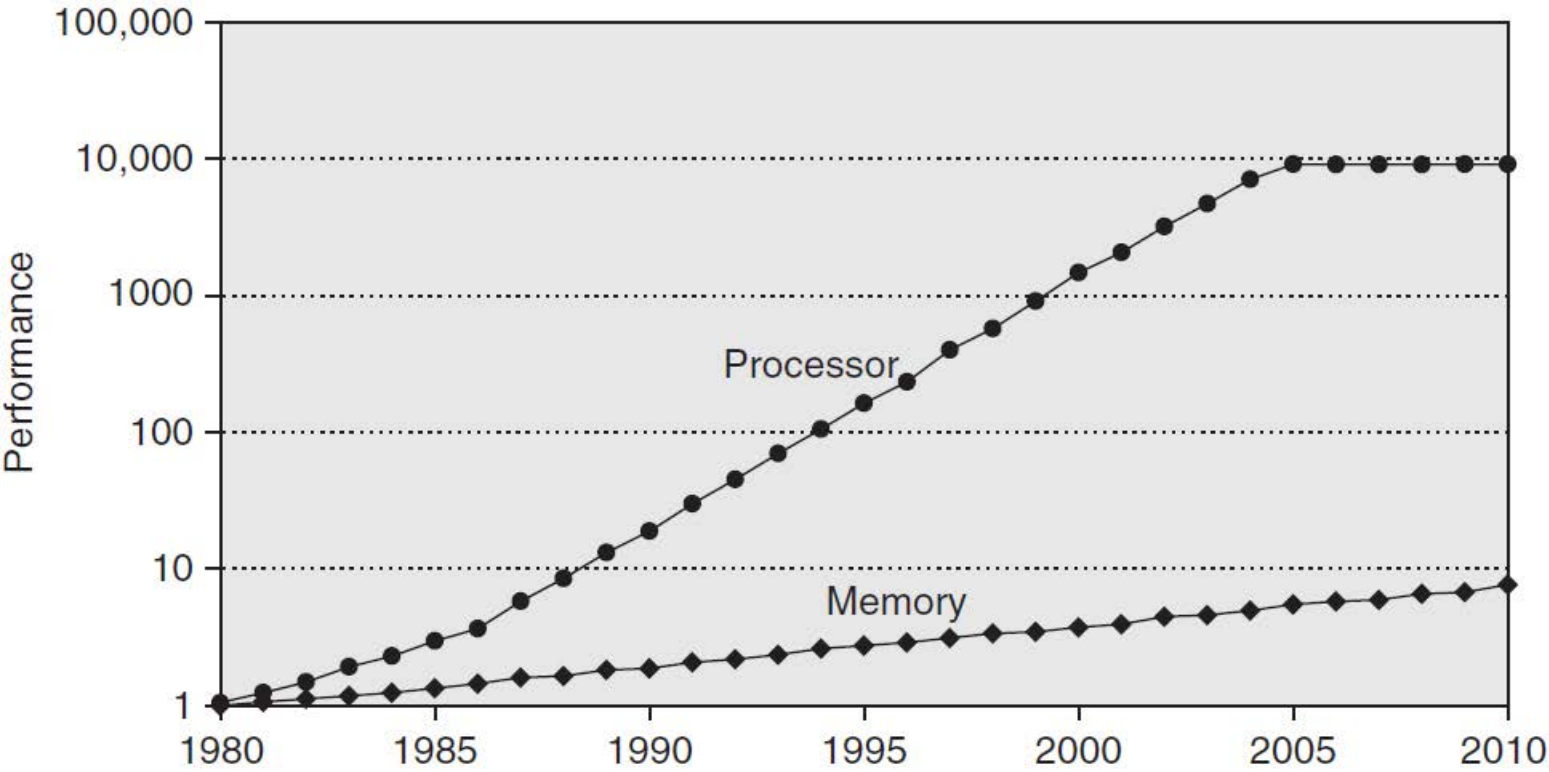
ВНИМАНИЕ: Условният преход се изпълнява след следващата прехода команда, но **условието** за прехода **се изчислява преди нея!** По тази причина команда, от която зависи това условие, не може да се премести след командата за преход.

Две последователни задържани команди за преход не може да има. В такива случаи между тях се поставя NOP.

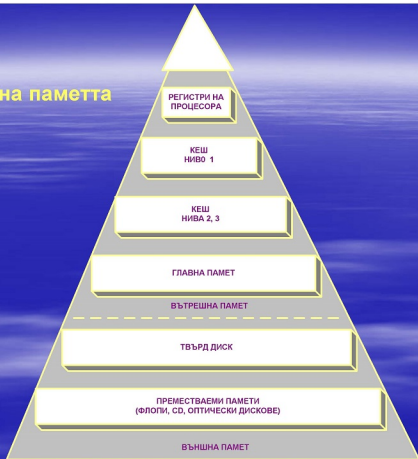
При някои процесори (напр. Alpha AXP и MIPS) асемблерът може автоматично да вмъква NOP след комадите за преход и дори да **пренарежда** командите, за да избегне нуждата от NOP. Този режим („*reorder*“) може да се изключи с директивата „*set noreorder*“.

Някои процесори (SPARC, MIPS) имат възможност за **анулиране** на следващата команда, ако преходът не се осъществи. Тя е полезна при проверка на условието в края на цикъла (*do-while, repeat-until*), ако при **последната итерация** командата след командата за преход **не бива** да се изпълнява. При MIPS тези команди (с „L“ от „*likely*“ много вероятен (т. н. „**намек**“).

Някои процесори (напр. POWER[PC]) имат възможност за такива „намеци“ с "+" и "-", при все че **не използват** задържани преходи.



Йерархия на паметта



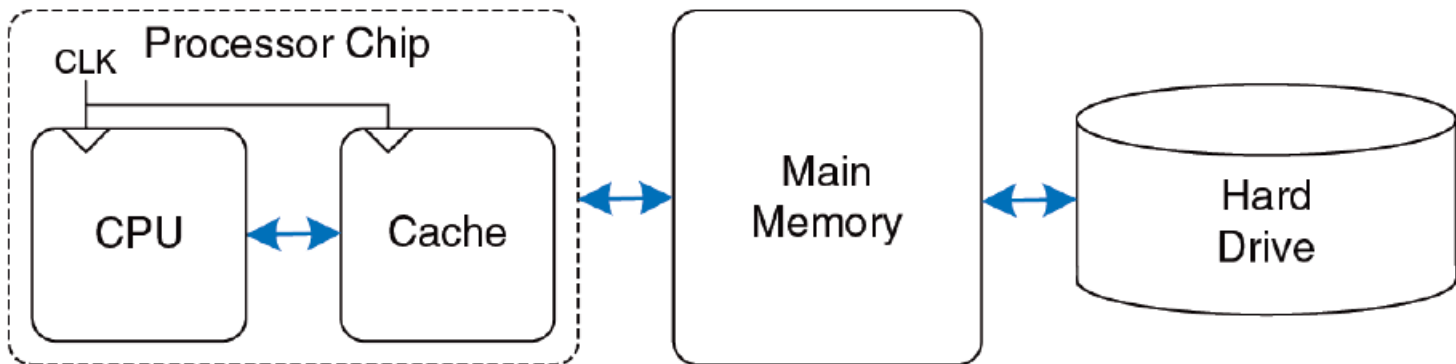


Рис. 8.3 Типичная иерархия памяти

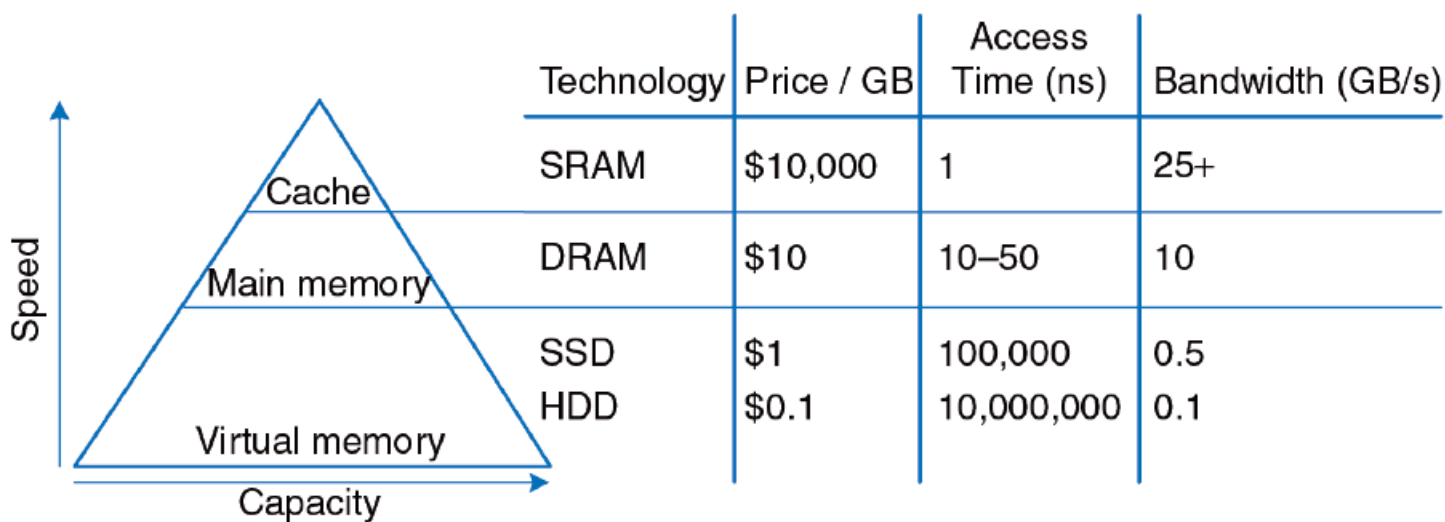


Рис. 8.4 Компоненты иерархии памяти и их характеристики на 2012 год

Level	1	2	3	4
Name	Registers	Cache	Main memory	Disk storage
Typical size	<1 KB	32 KB–8 MB	<512 GB	>1 TB
Implementation technology	Custom memory with multiple ports, CMOS	On-chip CMOS SRAM	CMOS DRAM	Magnetic disk
Access time (ns)	0.15–0.30	0.5–15	30–200	5,000,000
Bandwidth (MB/sec)	100,000–1,000,000	10,000–40,000	5000–20,000	50–500
Managed by	Compiler	Hardware	Operating system	Operating system/ operator
Backed by	Cache	Main memory	Disk	Other disks and DVD

Figure B.1 The typical levels in the hierarchy slow down and get larger as we move away from the processor for a large workstation or small server. Embedded computers might have no disk storage and much smaller memories and caches. The access times increase as we move to lower levels of the hierarchy, which makes it feasible to manage the transfer less responsively. The implementation technology shows the typical technology used for these functions. The access time is given in nanoseconds for typical values in 2006; these times will decrease over time. Bandwidth is given in megabytes per second between levels in the memory hierarchy. Bandwidth for disk storage includes both the media and the buffered interfaces.

1.2. Архитектура на кеш паметта

1.2.1 Работа на кеш паметта

Кеш паметта е ключа за осигуряване на производителност при съвременните процесори. Типично в една програма около 25% от инструкциите се отнасят до паметта, така че времето за достъп до паметта се явява критичен фактор в изпълнението на програмата. Чрез ефективно редуциране времето за достъп до паметта, кеш паметта позволява повече от една инструкция за цикъл да се изпълнява в модерните процесори (виж тема 5). Допълнителна индикация за важността на кеш паметта е напр., че от $6.6 \cdot 10^6$ транзистора в процесора MIPS R10000, $4.4 \cdot 10^6$ се използват в първичния кеш (в това число и TLB).

Работата на кеш паметта се базира на локалността на обръщанията.

Всички програми показват локалност на обръщанията. Това се оказва универсално свойство на програмите – независимо дали са комерсиални, научни, игри т.н. Кеш паметта експлоатира това свойство, подобрявайки времето за достъп и намалява цената на достъп до главната памет. Има два типа локалност

А) Локалност по време.

Б) Локалност по пространство (специална локалност).

Локалност по време – Веднъж направено обръщение към дадена позиция (клетка) от паметта има голяма вероятност, че тя ще бъде потърсена отново в недалечно бъдеще.

Примери: Най-простият пример за локалност по време е изпълнението на цикъл: веднъж въведен (включен) цикълът, всички команди в цикъла ще се посочват отново, вероятно много пъти, преди цикълът да завърши. Обобщено извикването на подпрограми, функции и прекъсванията по време също проявяват такова свойство.

Много типове данни показват локалност по време: в някаква точка на програмата съществува тенденция за обръщение към "горещи" данни, които програмата използва или променя многократно, преди да премине към друг блок от данни. Някои примери са:

- броячи;
- преглед на таблици;
- натрупване на променливи;
- стекови променливи.

Локалност по пространство – Когато се адресира клетка от паметта, много вероятно е, че съседните клетки ще бъдат скоро достъпни.

Примери: Ясно е, че потокът от инструкции ще проявява значителна специална локалност. В отсъствие на преходи следващата команда да бъде изпълнена е едно непосредствено, пряко следствие на текущата.

Данните също показват значителна специална локалност – напр. когато матрица или низ са достъпни, обработката започва от началото на матрицата до края, последователно.

Локалността по време включва локалността по пространство, но не и обратно.

Като правило, приложенията проявяват изключителна локалност по време за код и бедна за данни. Този факт е довел до разделянето на кеш паметта на две части – едната за команди (i-cache), а другата за данни (d-cache). Това разделяне води до значително увеличаване на производителността, зависещо от размера на кеш паметта, вида на приложението, др. фактори.

Feature	AMD Opteron 8439	IBM Power 7	Intel Xenon 7560	Sun T2
Transistors	904 M	1200 M	2300 M	500 M
Power (nominal)	137 W	140 W	130 W	95 W
Max. cores/chip	6	8	8	8
Multithreading	No	SMT	SMT	Fine-grained
Threads/core	1	4	2	8
Instruction issue/clock	3 from one thread	6 from one thread	4 from one thread	2 from 2 threads
Clock rate	2.8 GHz	4.1 GHz	2.7 GHz	1.6 GHz
Outermost cache	L3; 6 MB; shared	L3; 32 MB (using embedded DRAM); shared or private/core	L3; 24 MB; shared	L2; 4 MB; shared
Inclusion	No, although L2 is superset of L1	Yes, L3 superset	Yes, L3 superset	Yes
Multicore coherence protocol	MOESI	Extended MESI with behavioral and locality hints (13-state protocol)	MESIF	MOESI
Multicore coherence implementation	Snooping	Directory at L3	Directory at L3	Directory at L2
Extended coherence support	Up to 8 processor chips can be connected with HyperTransport in a ring, using directory or snooping. System is NUMA.	Up to 32 processor chips can be connected with the SMP links. Dynamic distributed directory structure. Memory access is symmetric outside of an 8-core chip.	Up to 8 processor cores can be implemented via Quickpath Interconnect. Support for directories with external logic.	Implemented via four coherence links per processor that can be used to snoop. Up to two chips directly connect, and up to four connect using external ASICs.

Figure 5.27 Summary of the characteristics of four recent high-end multicore processors (2010 releases) designed for servers. The table includes the highest core count versions of these processors; there are versions with lower core counts and higher clock rates for several of these processors. The L3 in the IBM Power7 can be all shared or partitioned into faster private regions dedicated to individual cores. We include only single-chip implementations of multicores.

1.2.3. Операции с кеш паметта

Един от основните въпроси при работа с кеш паметта е: "Как процесорът разбира дали информацията, която му е необходима, е налична в кеш паметта?"

Оперативната памет и кеш паметта са разделени на отделни области (frames), съдържащи определено количество байтове. Обменът на информацията между оперативната памет и кеш паметта се реализира по блокове. Размерът на блока съвпада с размера на фрейма и освен това в една област (един фрейм) може да се съхранява един блок. Когато процесорът отправя заявка към конкретен байт от определен блок в паметта, трябва много бързо да се определят следните три неща:

- дали търсения блок се намира в кеш паметта, т.е. дали има попадение в кеш паметта (cache hit) или не (cache miss);
- позицията на блока в кеш паметта в случай на наличие на блока в кеш паметта;
- положението на търсения байт в блока, отново само когато става дума за наличие на блока в кеш паметта.

Кеш памет с пълна асоциативност

При пълната асоциативност всеки блок от оперативната памет може да бъде свързан към всеки фрейм от кеш паметта. При това обаче се налага да се претърсва цялата tag RAM, за да бъде открит нужния блок. По тази причина колкото по-голяма е кеш паметта, толкова по-голямо ще е забавянето, преди нужния байт да стане достъпен за процесора. Ето защо пълната асоциативност не е често използвана организация в съвременните процесори.

Кеш памет с директно съответствие

При еш памет с директно съответствие на всеки блоков фрейм се съпоставя определено подмножество от основната памет. Например, ако разполагаме с 8 фрейма, във всеки от тях ще се разполага всеки осми блок от паметта, т.е. във фрейм 0 ще се разполагат блокове 0, 8, 16 и т.н., във фрейм 1 – блокове 1, 9, 17 и т.н. Голямото предимство в случая е, че потенциалните положения на всеки блок са значително редуцирани и съответно броят на търсенията в tag RAM е много по-малък. Например, ако на процесорът в нужен блок 0, 8 или 16 той трябва да провери само фрейм 0. Недостатъкът, който се появява обаче е следния – напр. на процесорът са нужни блокове с номера от 0 до 3 и от 8 до 11. При това положение двете групи блокове не могат да се поместят едновременно във фреймите, понеже 0 и 8 се разполагат във фрейм 0, а 1 и 9 – във фрейм 1 и т.н. При това положение процесорът ще трябва постоянно да променя съдържанието на кеш паметта, да отправя заявки към оперативната памет и като краен резултат рязко се снижава ефективността от кеширането и се губи преимуществото на намалената латентност. Такава ситуация се нарича колизия.

n-кратна (частична) асоциативност

Третият възможен вариант на организация представлява комбинация от горните два. При него определени подмножества от паметта могат да се поместят в определени групи фреймове. Например, четните блокове се поместват в първите 4 фрейма, а нечетните – във вторите 4. По този начин закъсненията при търсене в спомагателната памет се намалява два пъти спрямо пълната асоциативност, но все пак остава по-голямо отколкото при директното свързване и освен това вероятността да се достигне до колизия се намалява значително в сравнение с директното свързване. Когато процесорът определи в коя група би следвало да се намира информацията, той ще трябва да претърси само четири фрейма. Конкретно тази реализация се нарича четирикратно асоциативна кеш памет (four way set associative), поради факта, че кеш паметта е разделена на групи от по четири фрейма. Могат да се използват различни модификации на тази техника от типа двукратна (групи от по два фрейма) или осемкратна (групи от по осем фрейма) асоциативност. Не трябва да се забравя обаче, че всяко увеличаване на нивото на асоциативност (от две, към четири или от четири към осем) също увеличава броя на тяговете (етикетите), които трябва да се проверяват за да се определи конкретен блок. Следователно увеличаването на асоциативността също означава увеличаване на латентността.

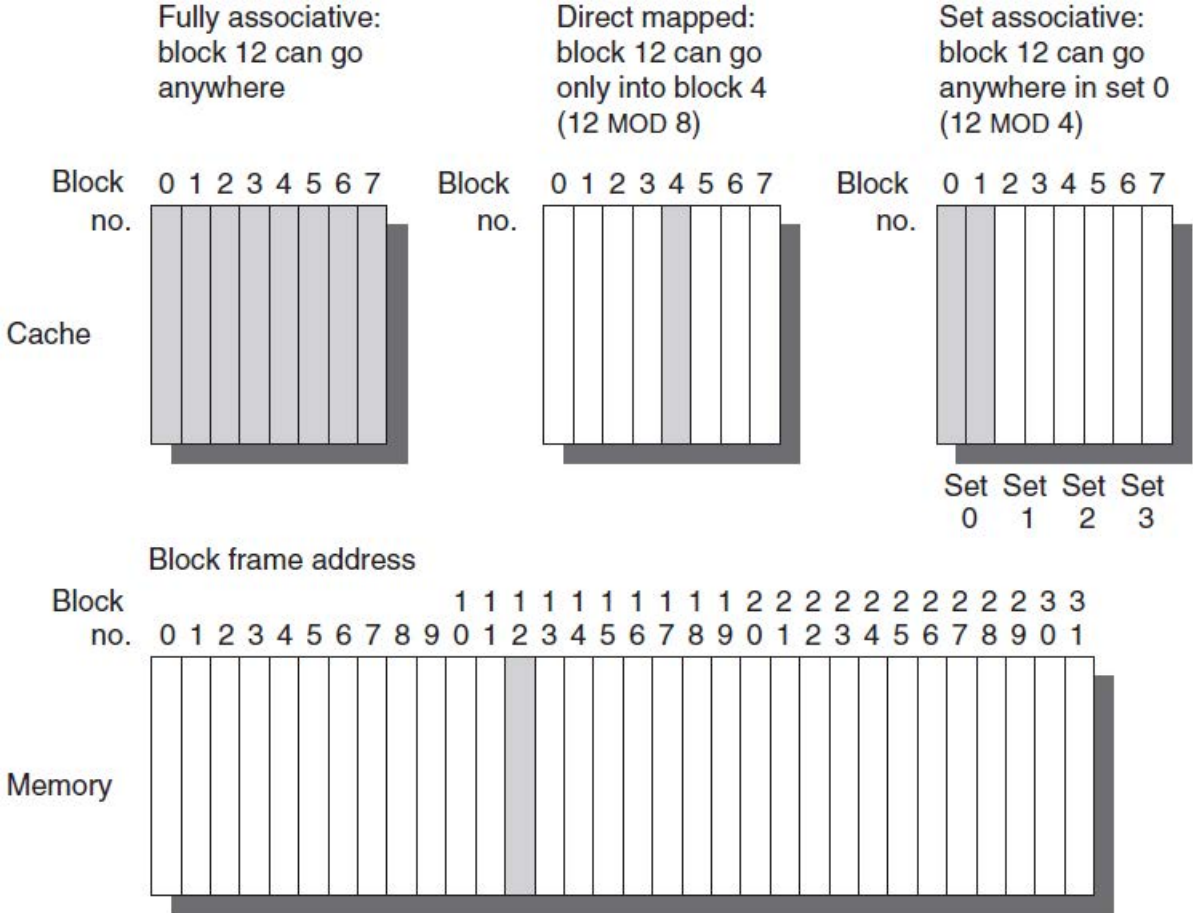


Figure B.2 This example cache has eight block frames and memory has 32 blocks. The three options for caches are shown left to right. In fully associative, block 12 from the lower level can go into any of the eight block frames of the cache. With direct mapped, block 12 can only be placed into block frame 4 ($12 \bmod 8$). Set associative, which has some of both features, allows the block to be placed anywhere in set 0 ($12 \bmod 4$). With two blocks per set, this means block 12 can be placed either in block 0 or in block 1 of the cache. Real caches contain thousands of block frames, and real memories contain millions of blocks. The set associative organization has four sets with two blocks per set, called *two-way set associative*. Assume that there is nothing in the cache and that the block address in question identifies lower-level block 12.

1.2.4. Заместване на информацията в кеш паметта.

Важен въпрос при кеширането е кой точно блок от кеш паметта да бъде заместен, когато постъпва нов блок. Най-простите стратегии са: да се замести случаен блок или да се използва някоя от методиките LIFO (Last In, Last Out) или FIFO (First In, First Out). За съжаление никоя от горепосочените методиките не отчита локалността на обръщенията. По тази причина, по-добра стратегия на заместване е **LRU** (**L**east **R**ecently **U**sed – най-отдавна използван), т.е. на заместване подлежи блокът използван преди най-много време. Именно тази стратегия и намира най-голямо приложение в съвременните процесори.

1.2.5. Запис в кеш паметта

Досега разгледахме поведението на кеш паметта при операция четене. При операция запис, която се среща основно при работа с кеш паметта за данни, има една особеност на които ще се спрем по-долу. При получаване на резултат, процесорът го записва в кеш паметта от най-високо ниво. Въпросът е как и кога този резултат се прехвърля в основната памет. Има две стратегии:

- Да се запише променената информация само в съответния ѝ блок в кеш паметта от първо ниво и да се актуализира по другите нива едва когато дадения блок бъде заместен. Това е така наречената стратегия `write back`.
- Да се запише променената информация едновременно по всички нива на йерархията на паметта. Това е така наречената стратегия `write through`.

По отношение на производителността, първата стратегия е по-добра, защото относително по-рядко се налага да се прехвърля модифицираното съдържание на блока към останалите нива на паметта. От друга страна втората стратегия позволява по-лесно да се поддържа валидността на информацията в многопроцесорните системи и при системи с интензивни входно-изходни процеси.

1.2.6. Съгласуваност на данните в кеш паметите [5]

В паралелните компютри от тип **SMP** (виж тема 10), където множеството процесори имат кеш памет на самия чип и използват обща памет, възниква един допълнителен проблем, известен под името "съгласуване на данните" или "кохеретност на данните" в кеш паметите. Същността на проблема се проявява в следното. В този тип компютри има едно копие от операнда в основната памет и по едно копие от него във всяка кеш памет. Когато копието от операнда е променено, да кажем от процесор #1, то следва и другите копия да бъдат променени също. Съгласуването на данните в кеш паметите е дисциплина с която, промените в стойностите на операндите, направени от отделен процесор, да се разпространят навсякъде в системата по подходящ начин и да бъдат достъпни за използване от другите процесори. Това става като на хардуерно ниво се въвежда "подслушване" на системния интерфейс. Един от най-често използваните протоколи за осигуряване на кохеретност на данните е **MESI** протокола, където буквите от акронима определят четирите състояния, в които кешовата линия може да бъде:

- **Modified** – линията е била модифицирана; копието на паметта е невалидно;
- **Exclusive** – кеш паметта има само едно копие на данните; паметта е валидна;
- **Shared** – повече от една кеш памет съдържа копие от тази линия; копието на паметта е валидно;
- **Invalid** – кешовата линия е невалидна

The Intel i7 uses a variant of a MESI protocol, called **MESIF**, which adds a state (Forward) to designate which sharing processor should respond to a request. It is designed to enhance performance in distributed memory organizations.

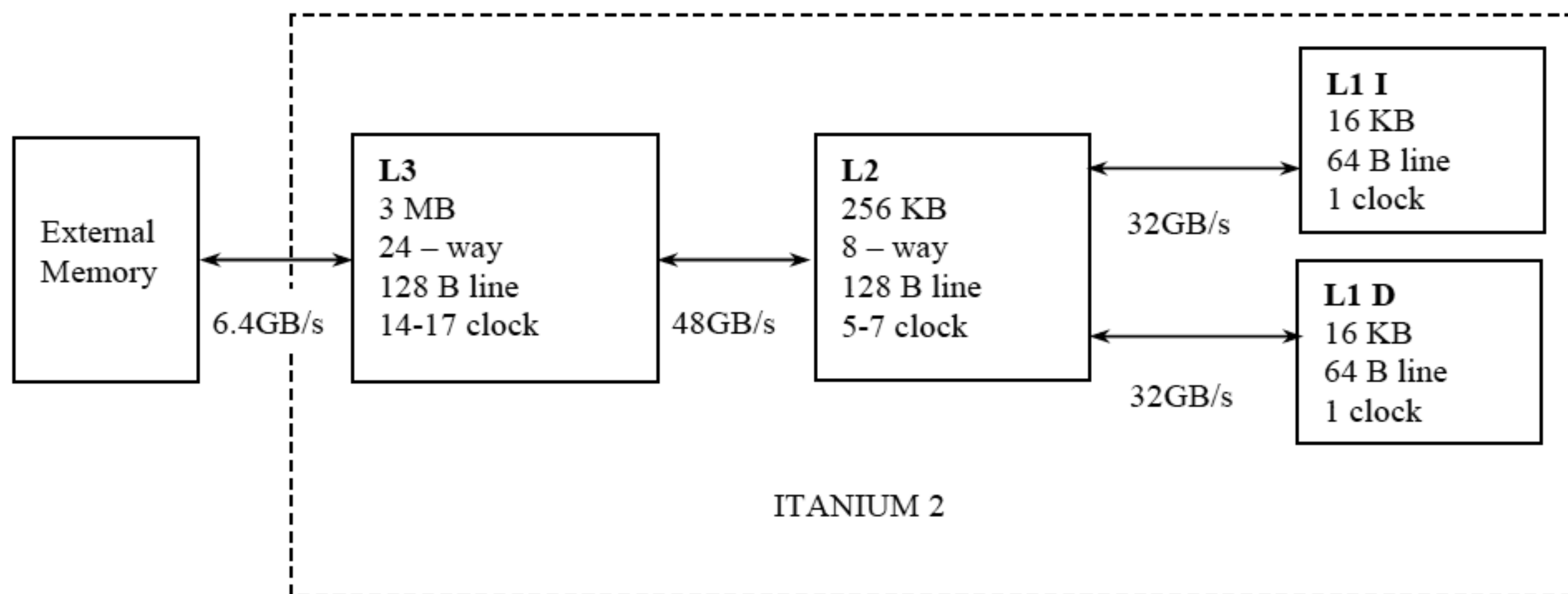
MOESI adds the state Owned to the MESI protocol to indicate that the associated block is owned by that cache and out-of-date in memory. In MSI and MESI protocols, when there is an attempt to share a block in the Modified state, the state is changed to Shared (in both the original and newly sharing cache), and the block must be written back to memory. In a MOESI protocol, the block can be changed from the Modified to Owned state in the original cache without writing it to memory. Other caches, which are newly sharing the block, keep the block in the Shared state; the O state, which only the original cache holds, indicates that the main memory copy is out of date and that the designated cache is the owner. The owner of the block must supply it on a miss, since memory is not up to date and must write the block back to memory if it is replaced. The AMD Opteron uses the MOESI protocol.

1.2.7. Размер на кешовия блок

С увеличаване на размерът на блока е възможно да се реализират по-добре предимствата на локалността по пространство. Това е вярно до някакъв определен размер. Ако се увеличи размерът на блока, докато размера на кеш паметта остава постоянен, тогава се намалява броят на блоковете, които кеш паметта може да съхрани. По-малко блокове в кеш паметта означава по-малко набори, което влече и по-голяма вероятност от отсъствия на търсената информация. Освен това, ако размерът на блока е много голям, налице е много загубено пространство, защото много блокове ще съдържат само няколко байта от работното множество, докато останалото пространство е неизползваемо. Казано по друг начин, големият размер на блока е по-склонен да "замърсява" кеш паметта с неизползвани данни отколкото малкия размер на блока.

Друг проблем на блока с голям размер е свързан с ширината на лентата на пропускане на шината. При по-голям блок повече данни се извличат с една команда от типа LOAD, големият блоков размер може да "изяжда" лентата на пропускане на шината, особено ако степента на не попадение е висок. Така, компютър с голям блоков размер на кеш паметта трябва да има по-висока ширина на лентата на пропускане, в противен случай латентността ще се увеличава.

Пропускателната способност се дефинира като скоростта, с която информацията стига до процесора и се измерва с байтове за секунда. По принцип, когато се говори за пропускателна способност, се цитира максималната пропускателна способност, която никога не може да се постигне. Това е така защото по шината непрекъснато трябва да се предава информация. Това не може да се получи по няколко причини. Първо, процесорът за да изпрати заявка до паметта, трябва да заеме шината. След като бъде изпратена заявката, паметта трябва да намери съответните данни и на свой ред да заеме шината и на края – да предаде данните. По тази причина натоварването на шината с предаването на данните никога не може да бъде 100%.



Фиг.9-2. Организация на кеш паметта в Itanium 2

Трите нива – L1, L2 и L3 на кеш паметта са разположени в чипа, заедно с ядрото на процесора. Прави впечатление, че паметта с по-ниско ниво е с по-малък обем от предходното, но времето за достъп е също по-малко. Освен това промяната засяга и големината на кешовата линия (блок), типа асоциативност, също така и скоростта на трансфер. Стратегията за запис е through write.

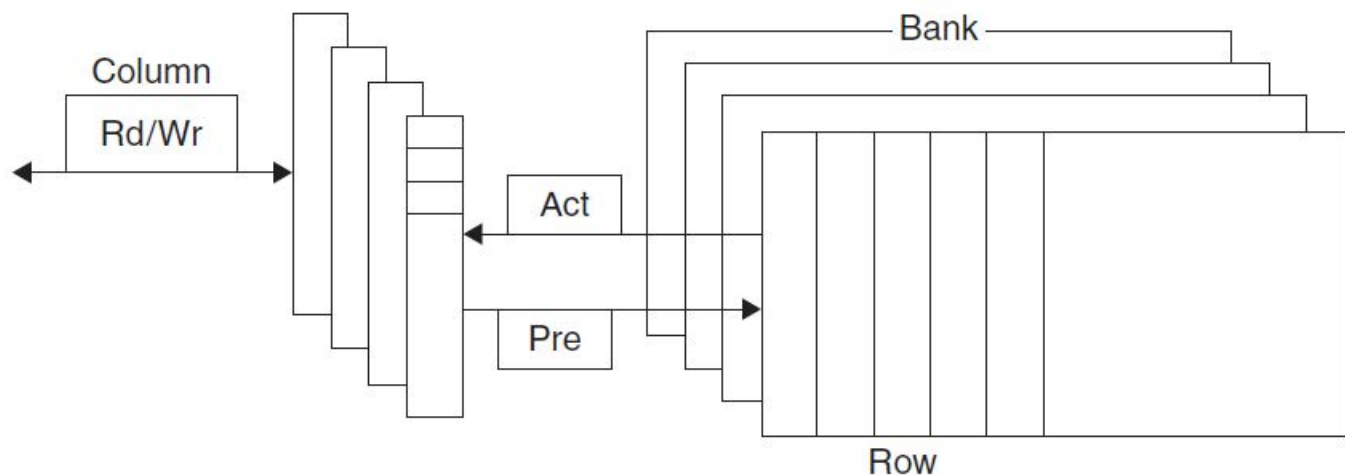
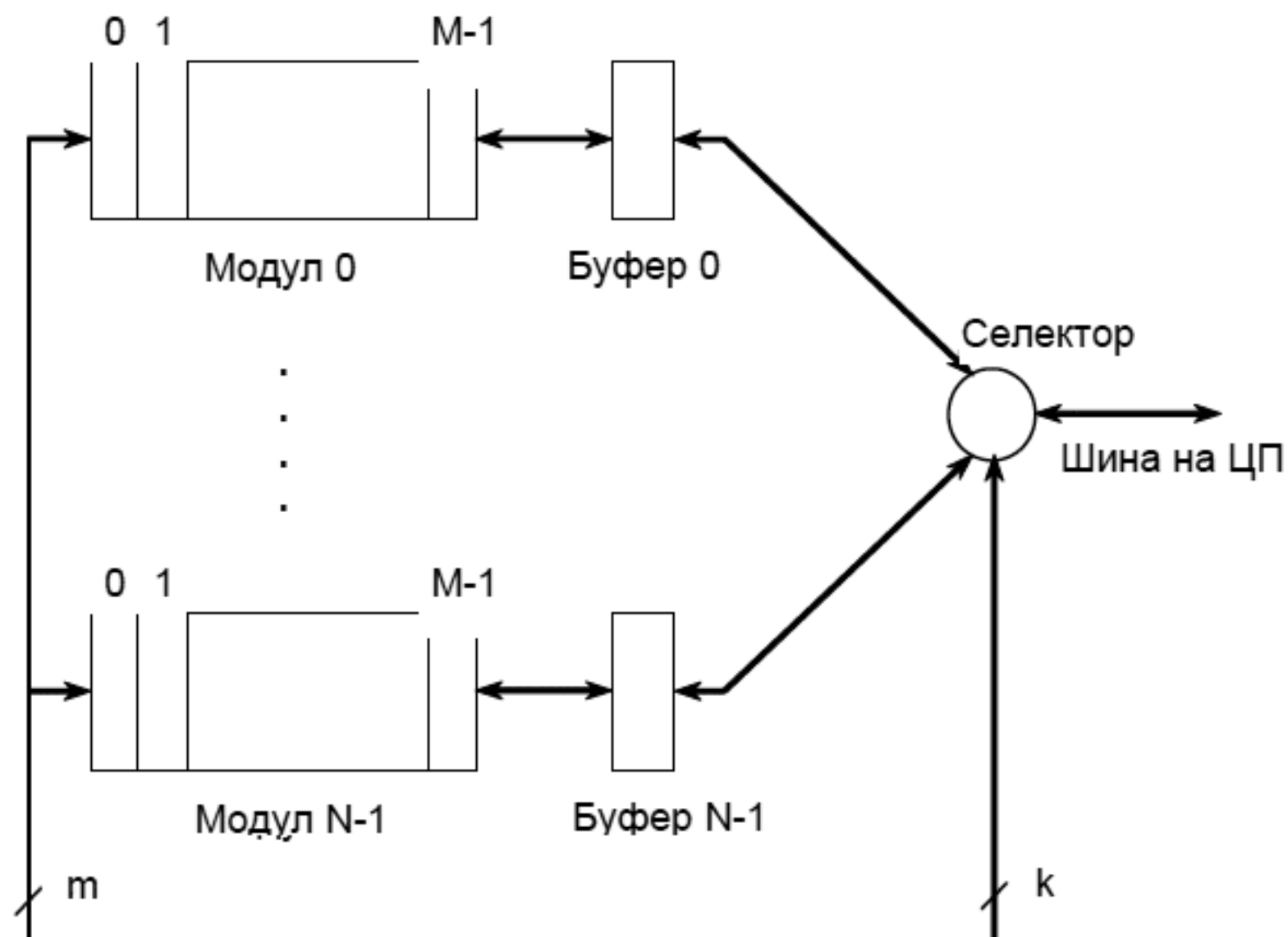


Figure 2.12 Internal organization of a DRAM. Modern DRAMs are organized in banks, typically four for DDR3. Each bank consists of a series of rows. Sending a PRE (pre-charge) command opens or closes a bank. A row address is sent with an Act (activate), which causes the row to transfer to a buffer. When the row is in the buffer, it can be transferred by successive column addresses at whatever the width of the DRAM is (typically 4, 8, or 16 bits in DDR3) or by specifying a block transfer and the starting address. Each command, as well as block transfers, are synchronized with a clock.

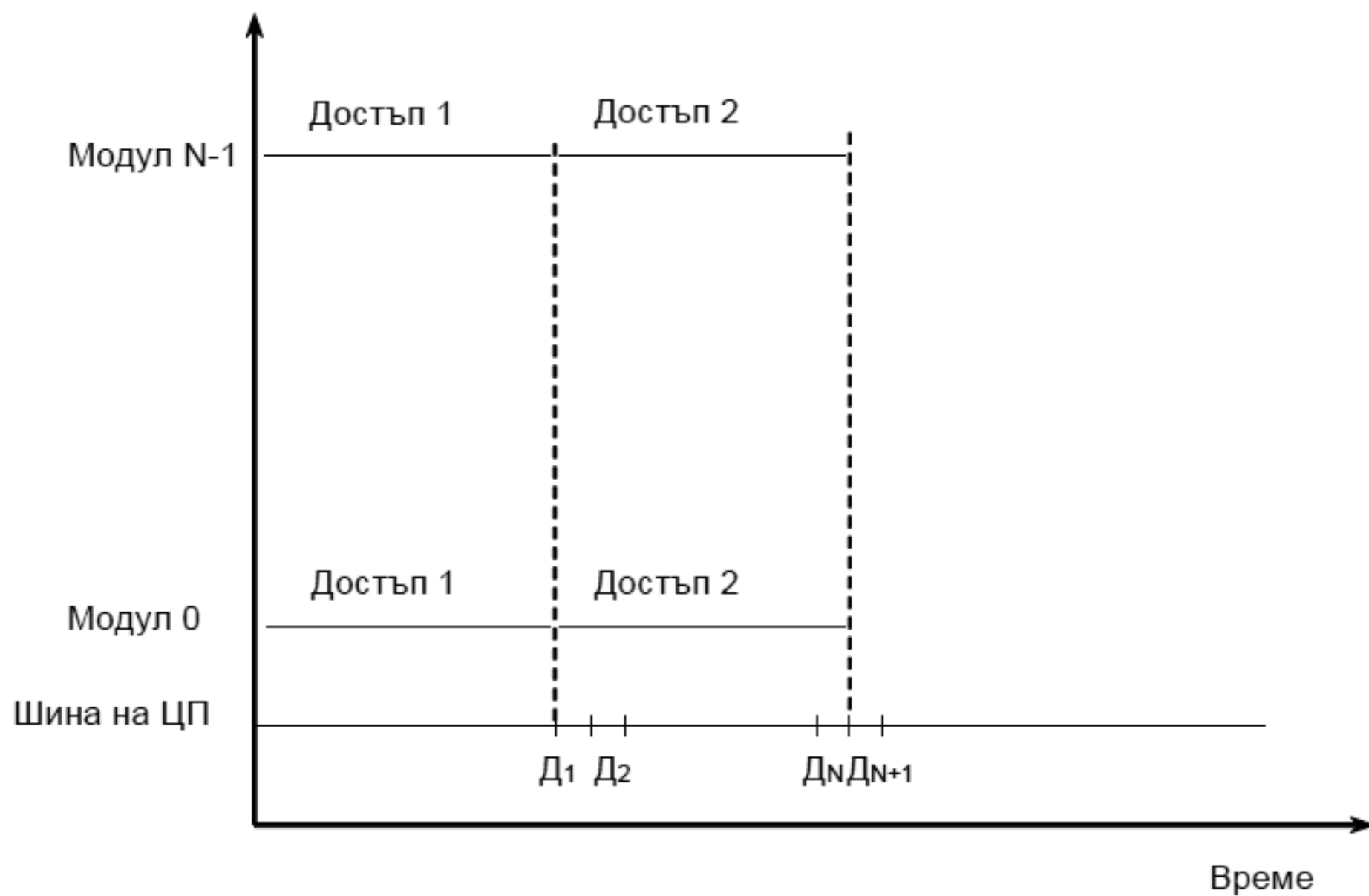
Хоризонтална организация на паметта може да бъде изградена по два начина:

- пакетна обработка на множество достъпи към паметта;
- конвейерна обработка на множество достъпи към паметта.

На фиг.9-3 е дадена примерна структура на памет от първия вид, осигуряваща достъп до **N** думи паралелно при всяко обръщение. Ако $N=2^k$ и $M=2^m$, то **m** разряда от **m+k** разрядния адрес се изпращат към всички модули памети, а младшите **k** разряда се използват за избор, коя от **N-те** думи да се чете първа. Така при всяко обръщение към паметта се прочитат **N** последователни думи (с адреси $iN+j$, където $0 \leq j \leq N-1$). Тези думи се помещават в буфери (фиксатори) и при следващото обръщение се четат от там със скорост **N** пъти по-голяма, отколкото времето за достъп до модула.

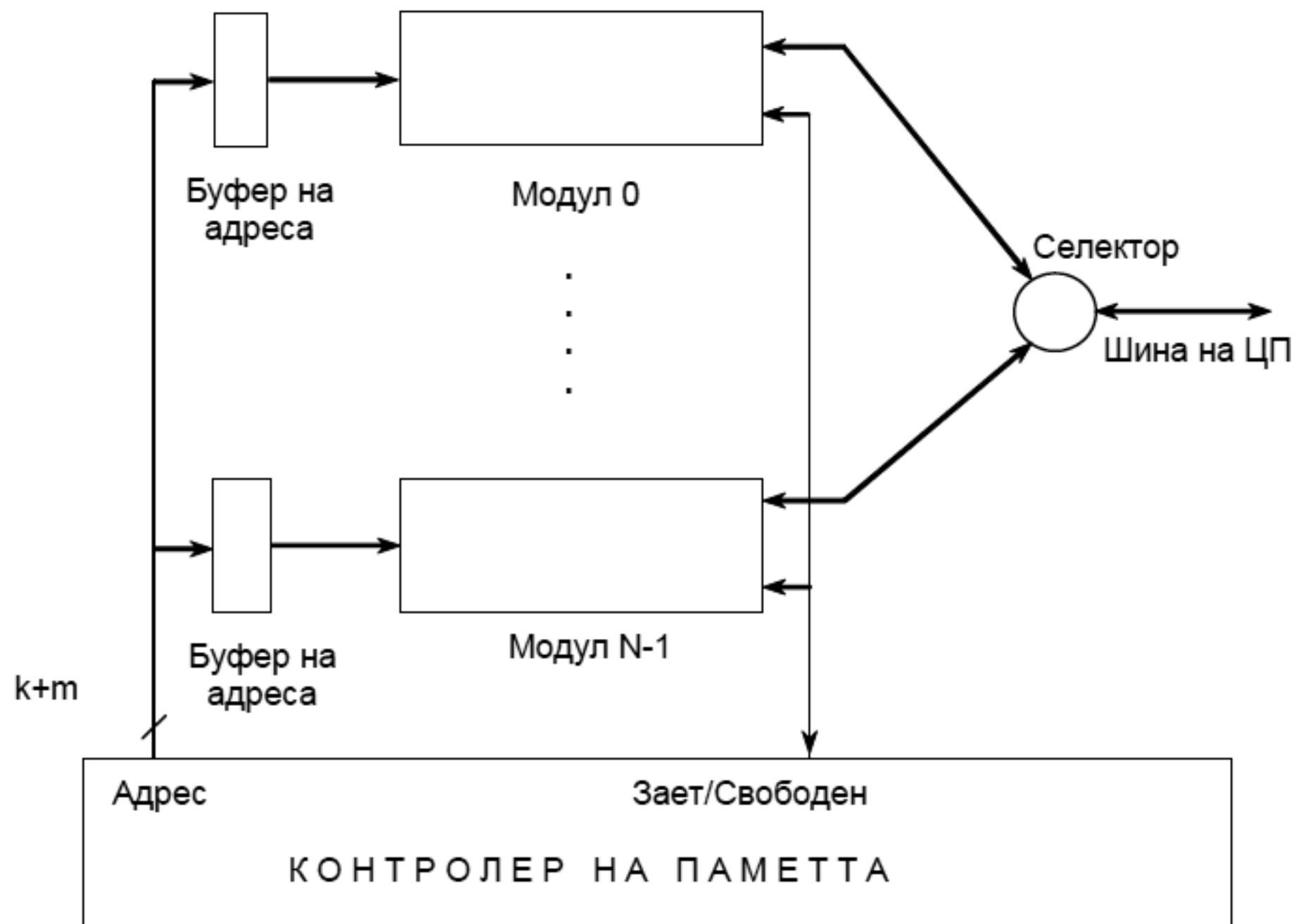


Фиг.9-3. Пакетен достъп до паметта с използване на редуване на адресите

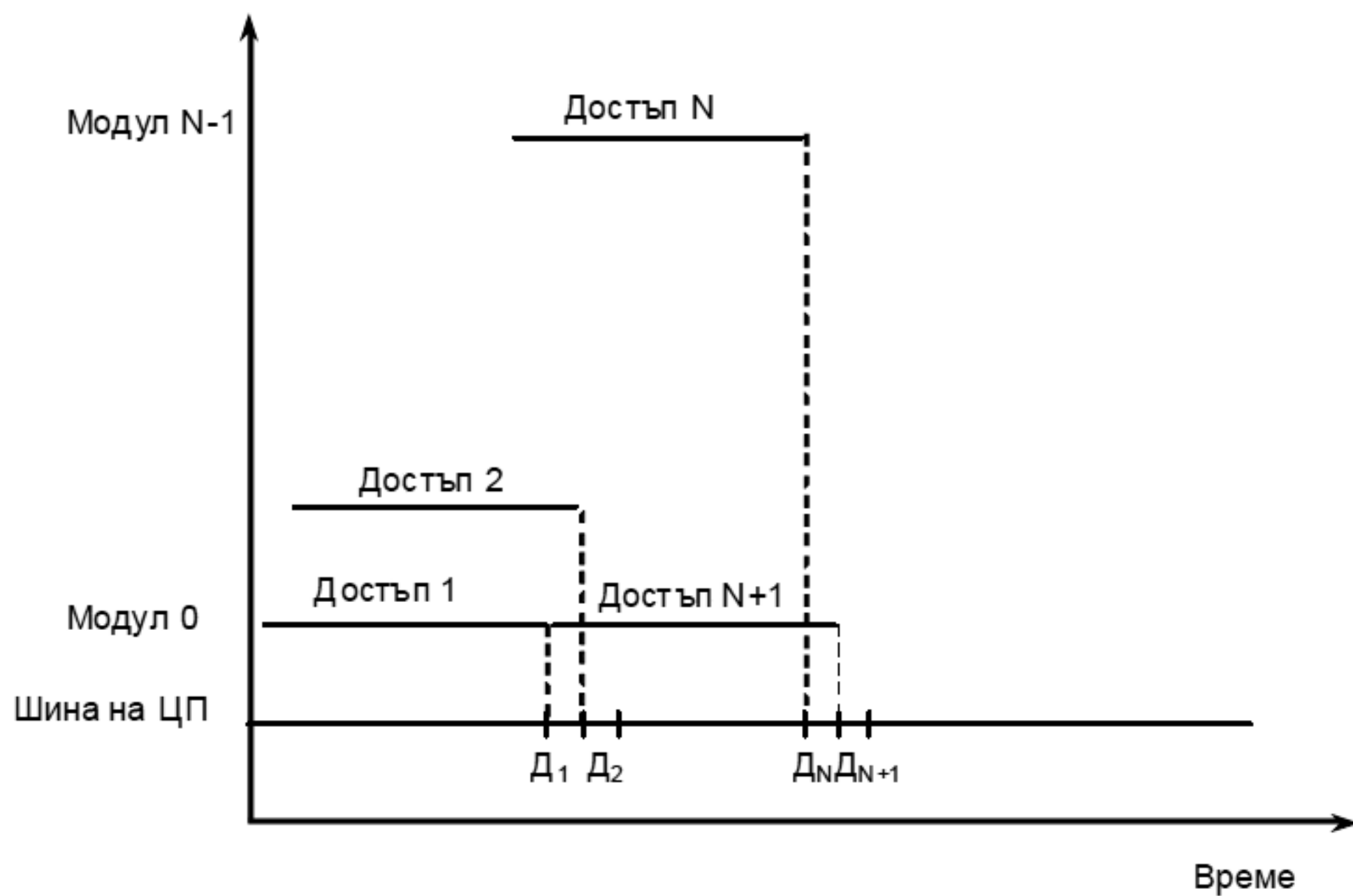


Фиг. 9-4. Диаграма за четене на N думи в пакет

Тази проста структура се явява идеална в тези случаи, когато обръщението към паметта се осъществява последователно, както при избор на вектори във векторните процесори така и при изпълнение на последователни команди. Но времето за достъп нараства значително когато е нужен достъп до непоследователно разположени думи (например, в програми с висок процент на команди за преход). Може да се докаже, че за последователност с интервал q , средното време за достъп към елементите на паметта ще бъде qt/N при $q \leq N$ и t при $q > N$ (t е времето за достъп до произволен модул).



Фиг. 9-5. Разслоена памет с конвейерна обработка на заявките.



Фиг. 9-6. Времедиаграма на работата на конвейерната памет.

Двете разгледани системи ще обработват заявките на последователните думи **N** пъти по-бързо, отколкото отделния модул. Освен това, те със същата скорост ще обработват произволна серия от заявки, в която нито един модул в **N** последователни заявки не се използва повече от един път. Например, 8-кратно разслоената памет от фиг.9-4 ще работи 8 пъти по бързо отколкото единия модул, ако последователността на адресите са разпределени равномерно не само с интервал 1, но и с интервал 3, 5, 7, 9, 11.... За интервал 3 последователността на адреси е 0, 3, 6, 9, 12, 15, 18...., а последователността на използваните модули е 0, 3, 6, 1, 4, 7, 2, 5, При някои други значения на интервала, например 2, работата няма да бъде с такава максимална скорост, но все пак по-бърза в сравнение с един модул. Може да се докаже, че в общия случай работата за произволна равномерна последователност от адреси с интервал, отличен от **N**, ще се осъществява на **N**кратно разслоена памет най-малко два пъти по-бързо, отколкото за памет имаща само един модул.

To support the default 128-bit operation the memory controller data path and the internal buses that connect the memory controller to PI, HT and PCIe are all 128-bits wide. When configured for 64-bit operation, the memory controller assembles every two 64-bit external transfers into 128-bit internal transfers, and data is transferred on the internal buses every other clock.

Figure 7-2. External Memory Configurations

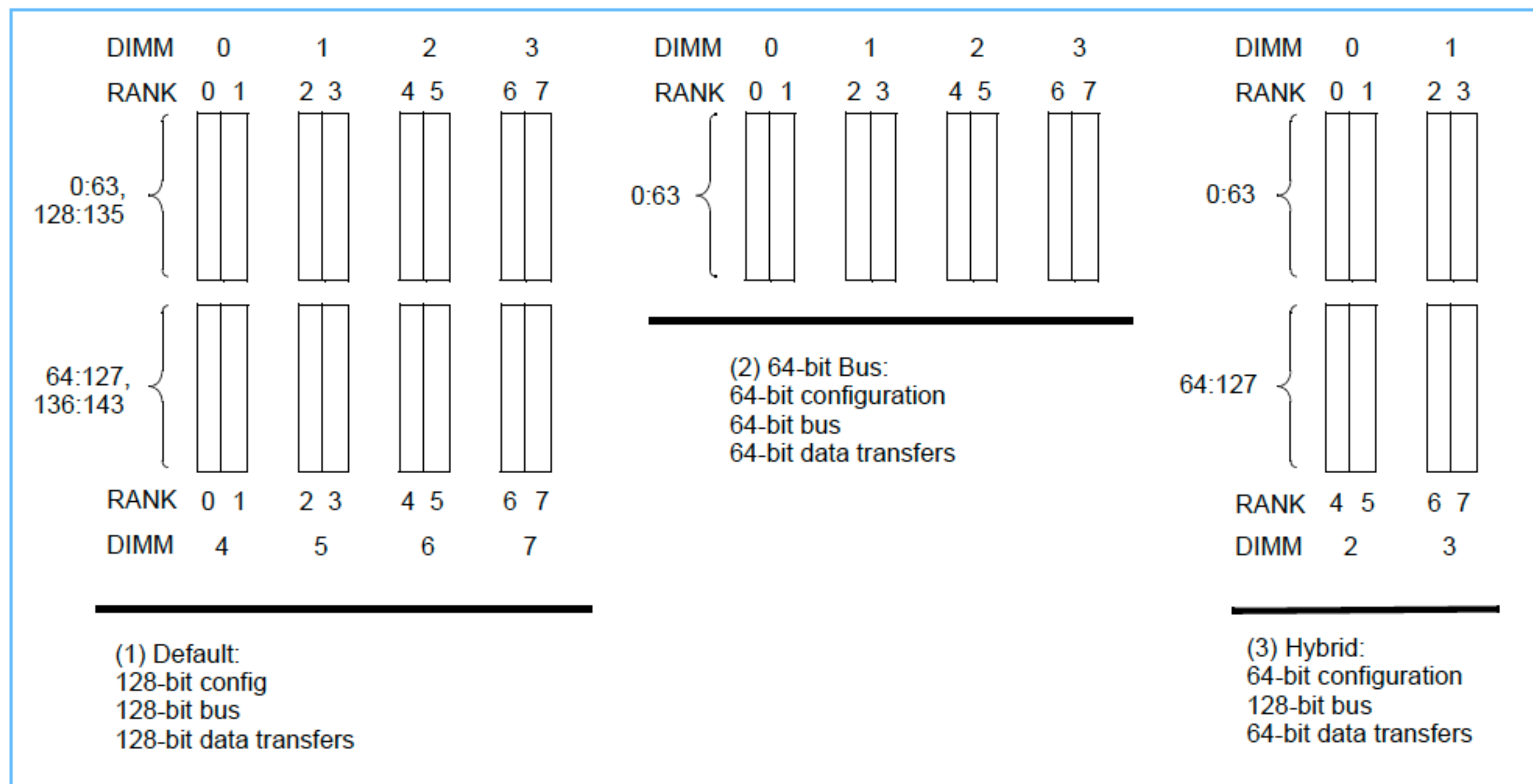
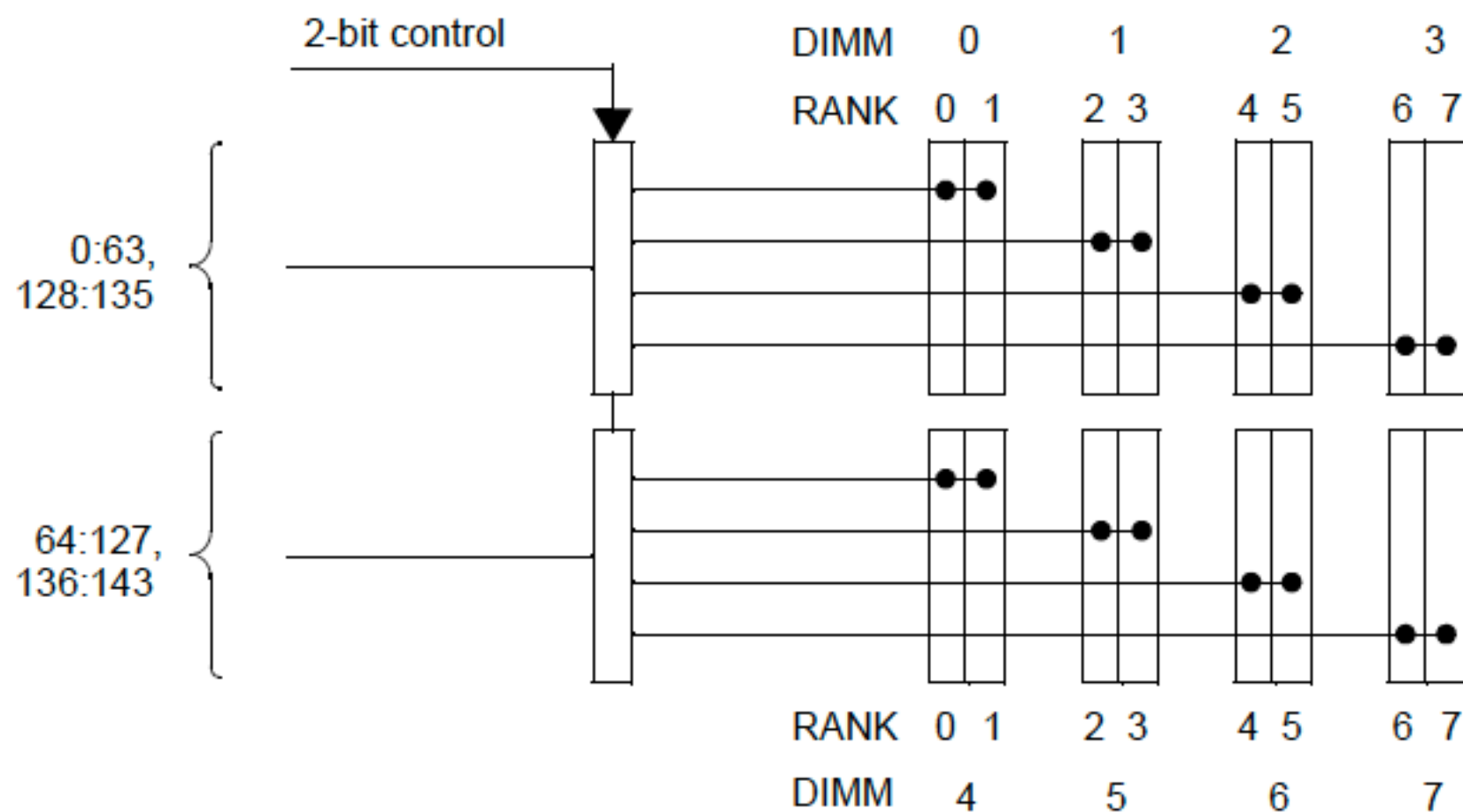
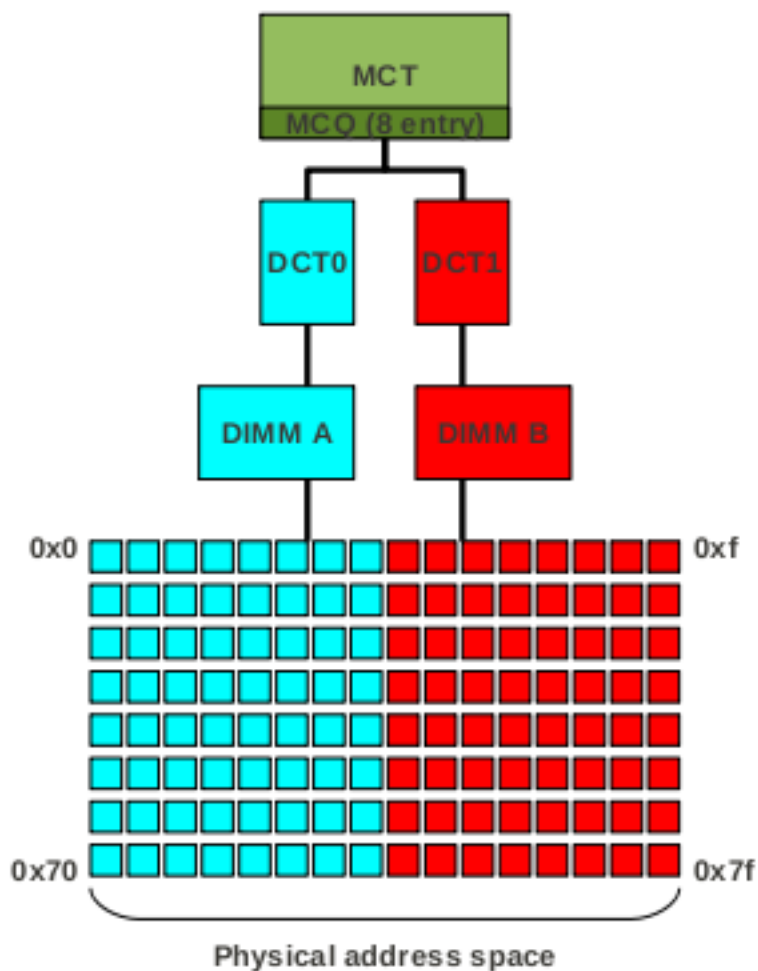


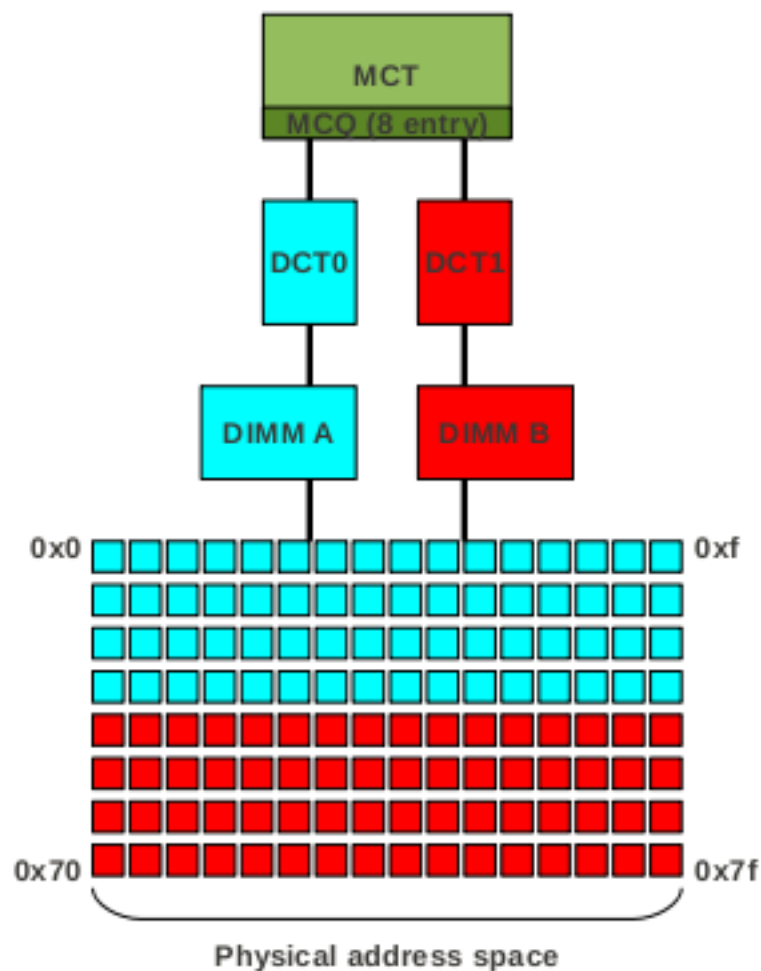
Figure 7-6. External Multiplexers



Ganged Mode

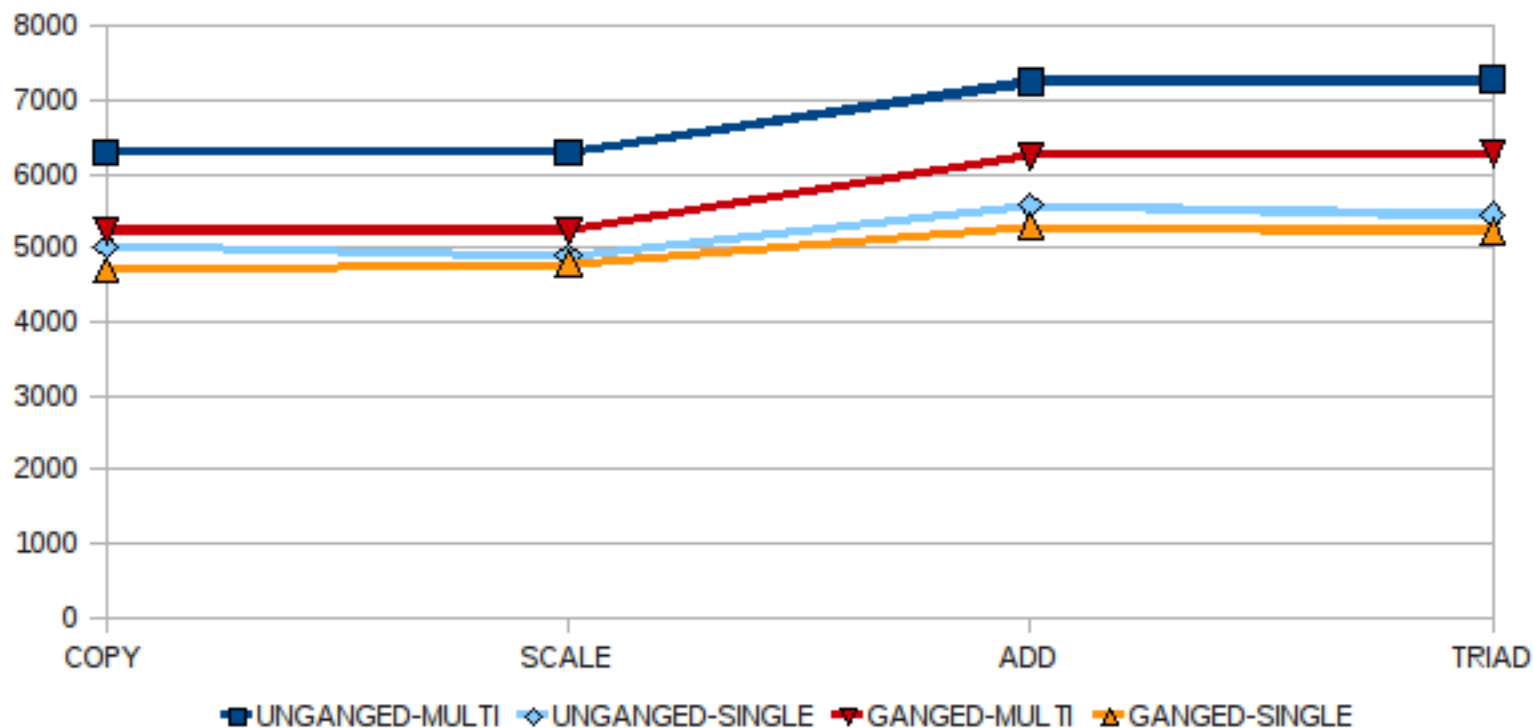


Unganged Mode



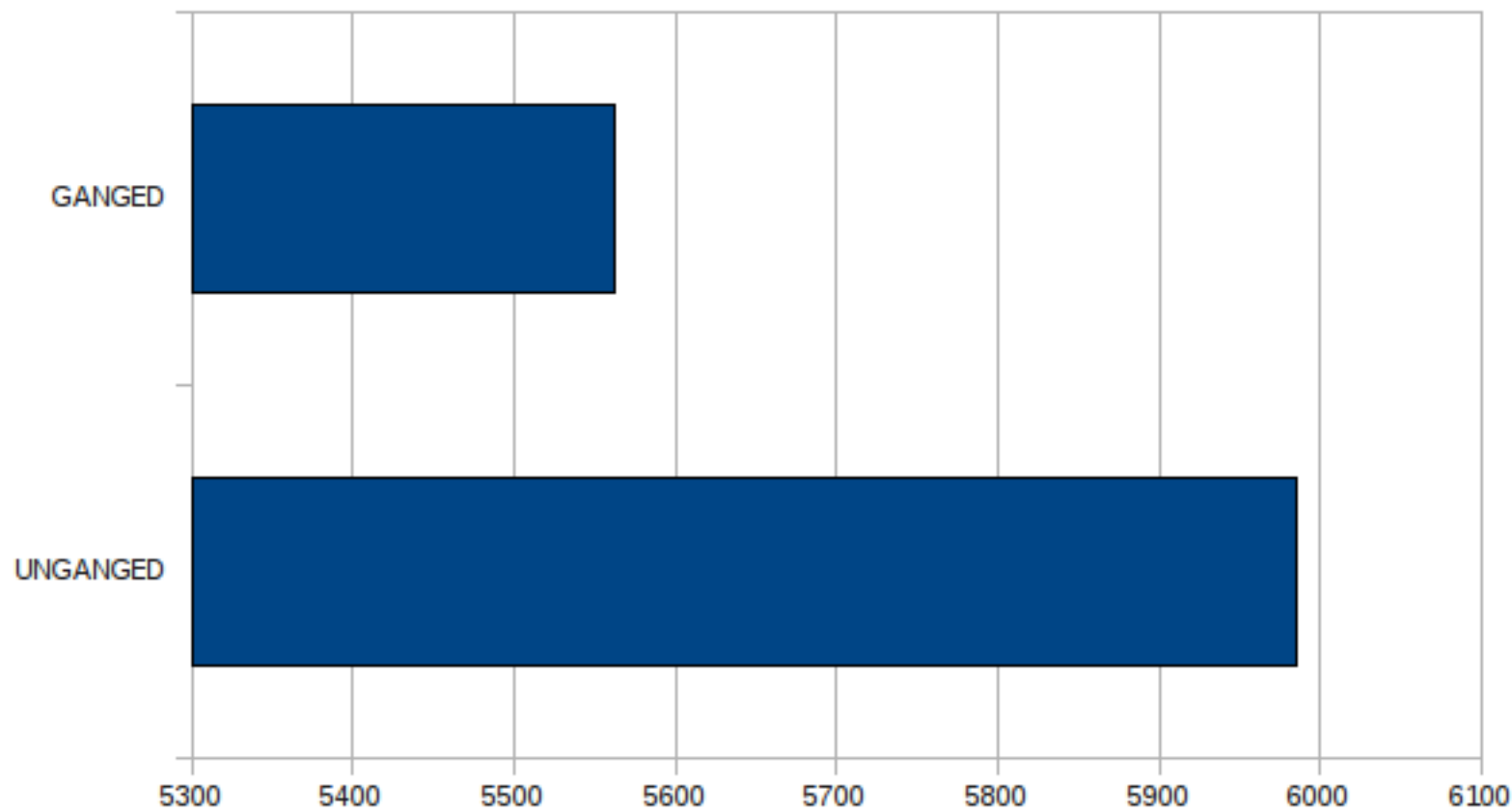
STREAM memory benchmark (MB/s)

Single and multi thread performance



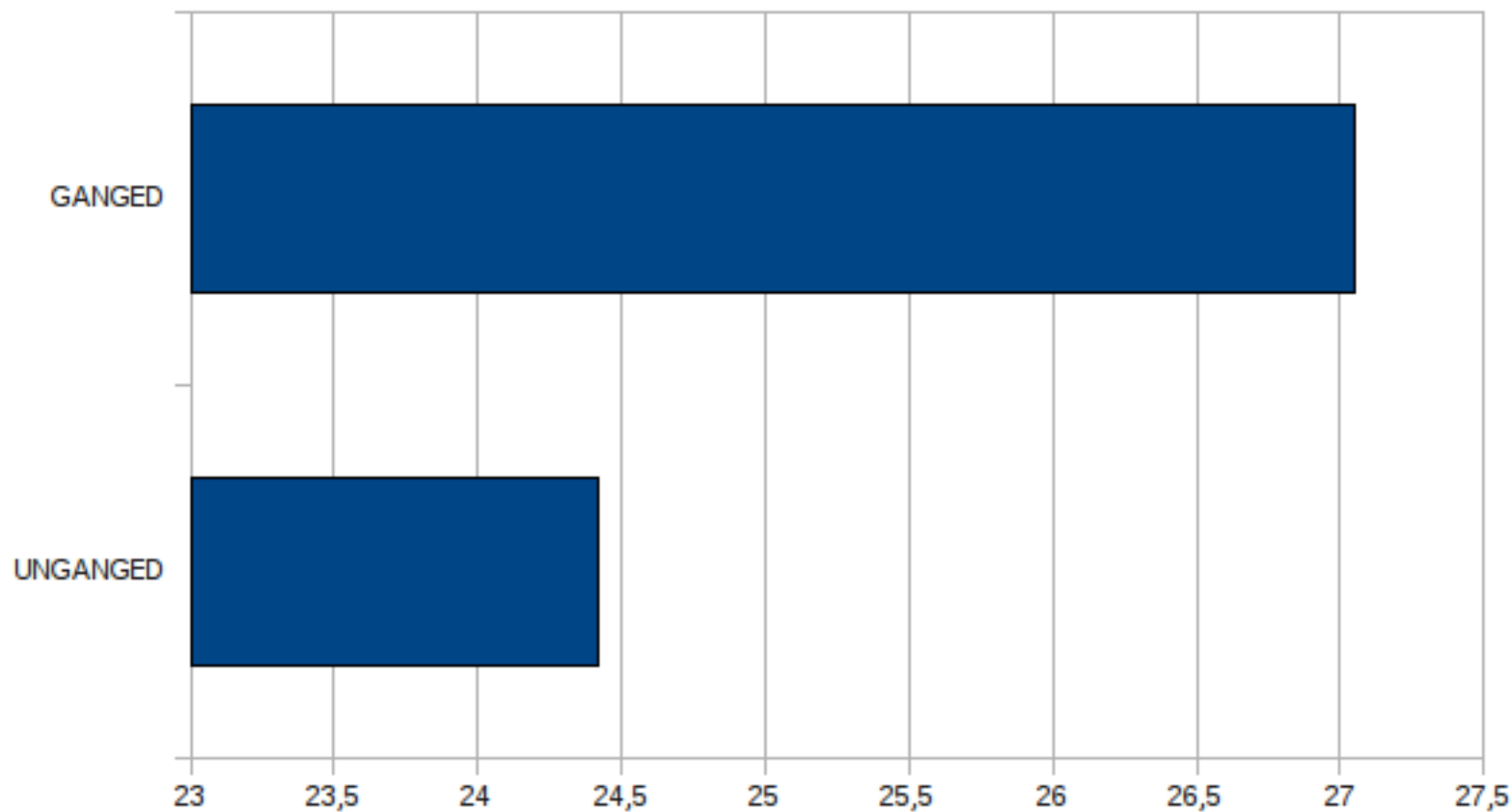
Sequential concurrent memory speed

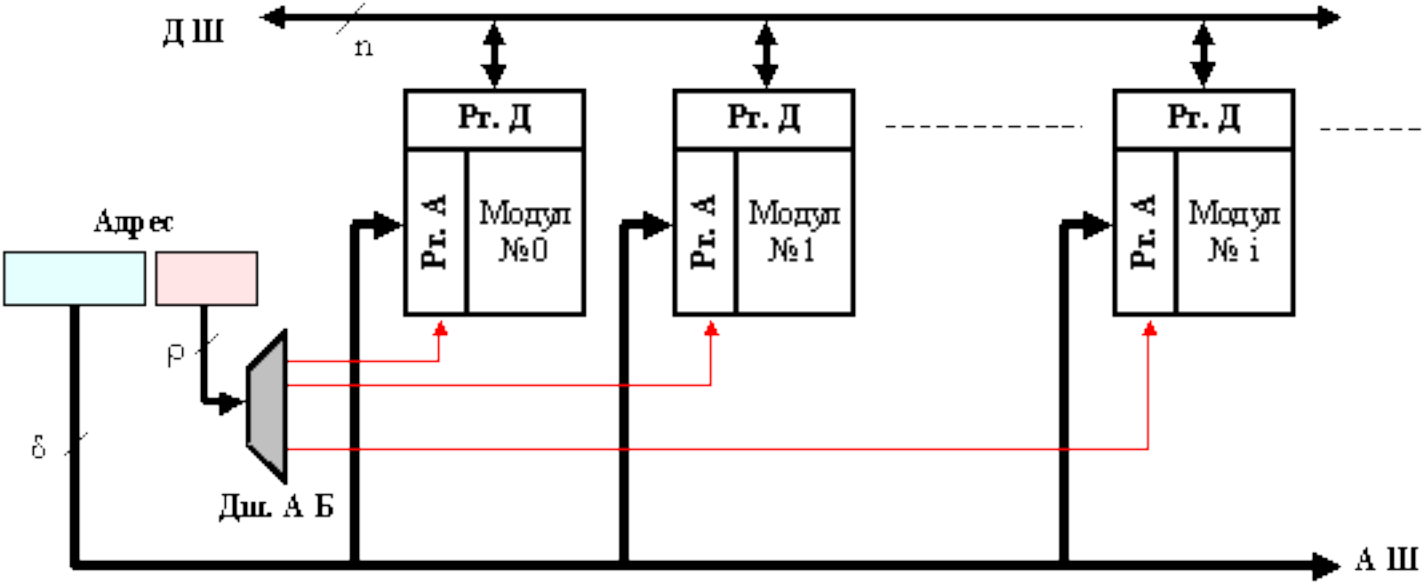
One write process and one read process, in MB/s



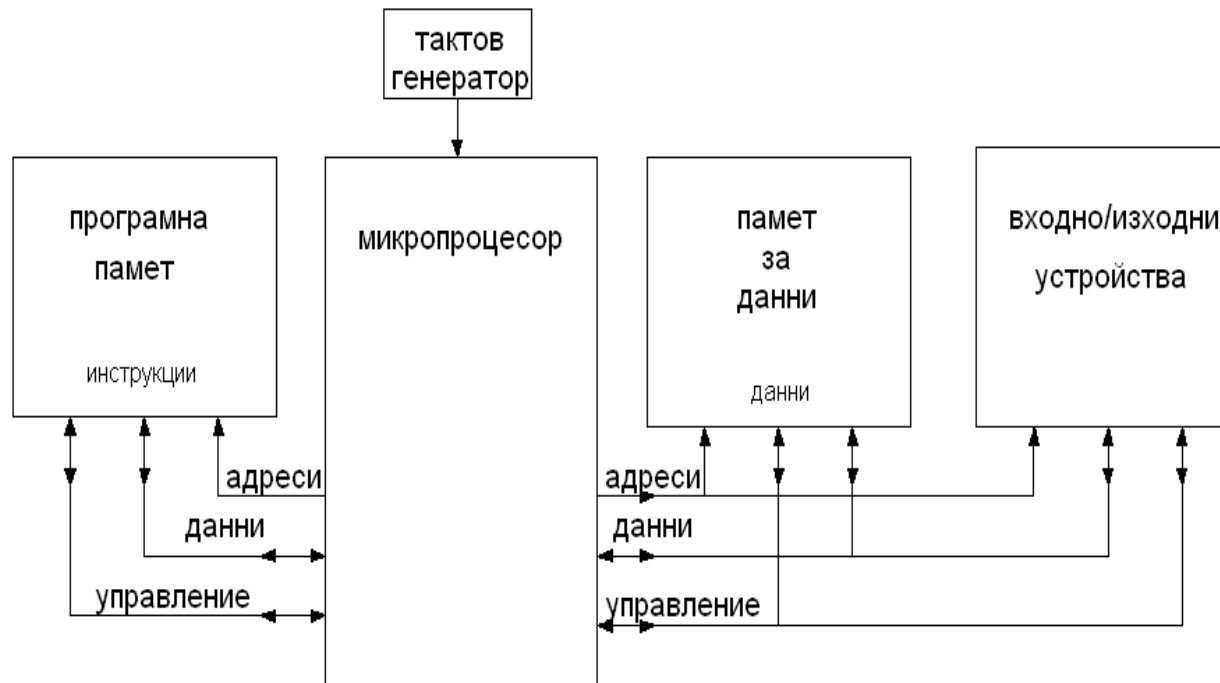
Joomla benchmark, 256 threads

Expressed in requests/sec





АРХИТЕКТУРА ТИП “ХАРВАРД”



- **Основни елементи:** централен процесор; програмна памет; памет за данни; периферни устройства; линии за връзка между тях.
- **Особености :** физическо разделяне на паметта на две части – за инструкции и данни, със собствени шини за адреси и данни.

Както стана ясно, разслоената памет е ефективна при подходящо обръщение към модулите ѝ. За да се осигури такова обръщение се използва съответно разположение на данните в паметта. За да стане ясно какъв е проблемът, да разгледаме съхраняването на матрица $X(4,4)$ в четирикратно разслоена памет. На фиг.9-7 е показан един от възможните начини.

Модул	0	1	2	3
	X_{11}	X_{12}	X_{13}	X_{14}
	X_{21}	X_{22}	X_{23}	X_{24}
	X_{31}	X_{32}	X_{33}	X_{34}
	X_{41}	X_{42}	X_{43}	X_{44}

Фиг.9-7. Разположение на елементите на матрицата X по редове.

В този случай е възможно безконфликтно обръщение към елементите от един ред или от диагоналите, т.е. извличат се 4 елемента за един цикъл на паметта. При опит да се обработват паралелно елементите на матрицата принадлежащи на един стълб, настъпва конфликт, защото всичките те се съхраняват в един и същ модул памет. За решаването на този проблем се използва така нареченото скосено (skewed) разположение – фиг.9-8.

Модул	0	1	2	3
	X_{11}	X_{12}	X_{13}	X_{14}
	X_{24}	X_{21}	X_{22}	X_{23}
	X_{33}	X_{34}	X_{31}	X_{32}
	X_{42}	X_{43}	X_{44}	X_{41}

Фиг.9-8. Скосено разположение на елементите на матрицата X

В езика TRANQUIL за ILLIAC-IV са били предвидени средства, използвани от програмиста за указване способа на разместване в паметта. Операторът STRAIGHT подрежда данните по начин показан на фиг.9-7, а операторът SKEWED подрежда данните по начин показан на фиг.9-8.

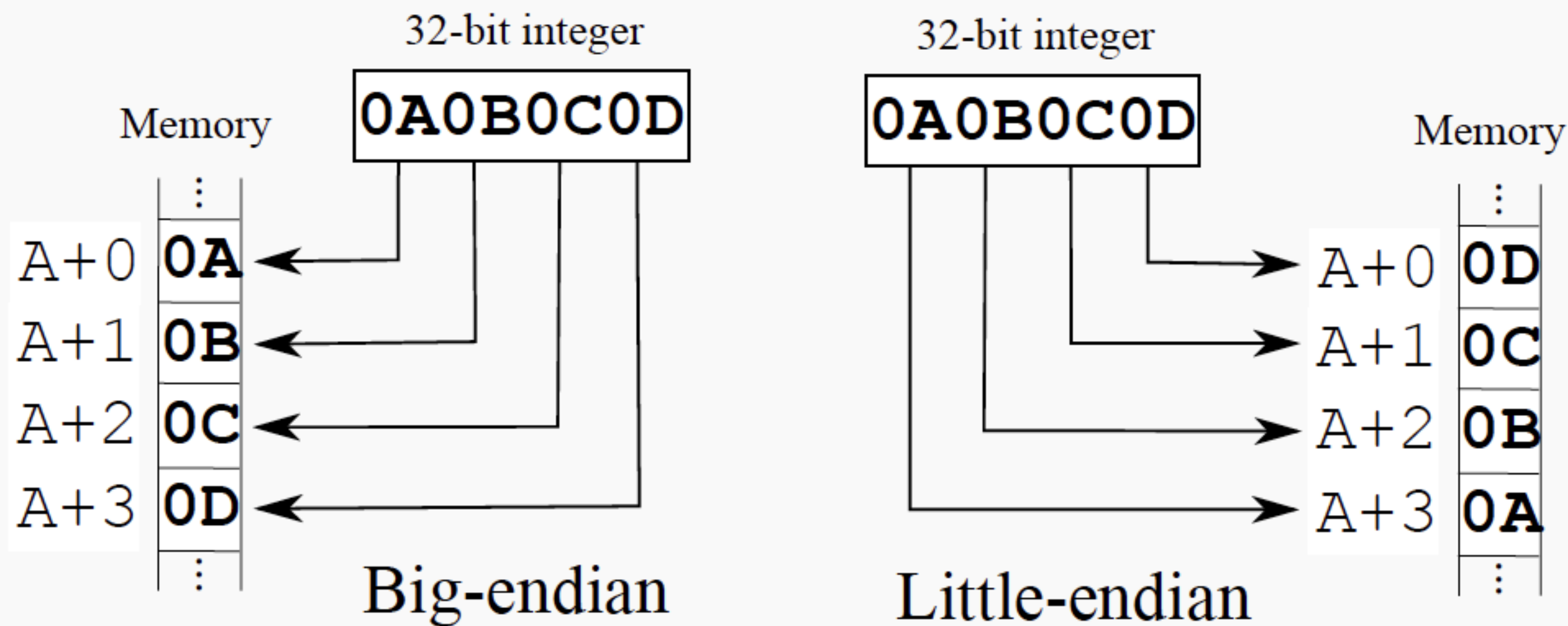
Модул 0 1 2 3 4

X_{11}	X_{12}	X_{13}	X_{14}	
X_{24}		X_{21}	X_{22}	X_{23}
X_{32}	X_{33}	X_{34}		X_{31}
	X_{41}	X_{42}	X_{43}	X_{44}

Фиг. 9–9. Напълно безконфликтен достъп до елементите на X .

Тук се използват повече на брой модули памет от броя процесори (в случая 5). Празните позиции в съответните клетки означава, че в тях не се записват елементи на матрицата X .

Може да се докаже, че за да се достигне степен на паралелизъм N , модулите памет M трябва да бъдат първото просто число по-голямо от броят на процесорите N . Така е решен проблемът в компютъра BSP на Burroughs. Броят на модулите памет е 521 (първото просто число по-голямо от 512 – броят на процесорите).



The architecture of a system defines its attributes as seen by the programmer, that is, the conceptual structure and functional behavior of the machine, as distinct from the organization of the data flow, the logical design, the physical design, and the performance of any particular implementation. Several dissimilar machine implementations may conform to a single architecture. When the execution of a set of programs on different machine implementations produces the results that are defined by a single architecture, the implementations are considered to be compatible for those programs.

ISA = Instruction Set Architectures:

1. IA-32 = Intel Architecture – 32-bit (INTEL = INTEgrated ELEctronics)
2. ARM = Acorn RISC Machines → Advanced RISC Machines
3. POWER = Performance Optimization With Enhanced RISC
4. SPARC = Scalable Processor ARChitecture
5. MIPS = Microprocessor without Interlocked Pipeline Stages
6. RISC-V (RISC = Reduced Instruction Set Computer)
7. PA-RISC = Precision Architecture RISC
8. Alpha AXP (AXP = Almost eXactly PRISM, PRISM = Parallel Reduced Instruction Set Machine)
9. ARC = Argonaut RISC Core
10. SuperH = Super Hitachi (Hitachi = 日立 = изгрев) SH-4
11. SuperH SH-5
12. Xtensa
13. OpenRISC
14. AMD64 (AMD = Advanced Micro Devices)
15. ARM64
16. ARC64
17. AVR32 (AVR = Alf and Vegard's RISC processor)
18. CRIS (Code Reduced Instruction Set)
19. LatticeMico32
20. Adapteva Eiphany
21. XILINX MicroBlaze
22. NIOS II
23. Blackfin
24. M32R
25. V850
26. LoongArch
27. MMIX
28. ESP32 (Xtensa)
29. Intel i860
30. Intel i960
31. AMD Am29000
32. Motorola 88000 (MOTOROLA = MOTOR radiOLA)
33. Transputer (TRANSPUTER = TRANSistor + comPUTER)
34. MicroVAX 78032 (VAX = Virtual Address Extension)
35. Motorola 680x0
36. NS32x32
37. H8SX
38. RX (Renesas eXtreme, RENESAS = RENaissance Semiconductor for Advanced Solutions)
39. WE32x00 (WE = Western Electric)
40. Interdata 7/32, 8/32
41. Z80000 (Z = Zilog, ZILOG = Z Integrated LOGic)
42. z/Architecture
43. Intel Itanium (Їtanium)
44. Transmeta Crusoe
45. Transmeta Efficeon
46. Qualcomm Hexagon
47. Tile GX
48. Эльбрус 2000...

1978	96	8086 (3,2 μm nMOSFET)	
1980	83	8087 (3 μm)	
1982	105	80186	
1982	112	80286	
1982	84	80287	
1985	166	80386 (1,5 μm CHMOS III)	
1987	96	80387	
1989	267	80486DX/P4 (1 μm CHMOS IV)	FPU
1993	273	80586/P5/Pentium(0,8 μ BiCMOS)	FPU
1995	304	80686/P6/Pentium Pro (350 nm)	FPU
1997	321	Pentium MMX (280 nm)	FPU, MMX
1997	333	6x86MX (Cyrix)	FPU, MMX, EMMI
1998	353	K6-2 (AMD, 250 nm)	FPU, MMX, 3DNow!
1999	358	K6-2+ (AMD, 180 nm)	FPU, MMX, Enhanced 3DNow!
1999	420	Pentium III (250 nm CMOS)	FPU, MMX, SSE
2000	489	Pentium 4 (180 nm)	FPU, MMX, SSE, SSE2
2003	528	K8 / Athlon 64 (AMD, 130 nm)	FPU,MMX,Enhanced 3DNow!,SSE,SSE2,AMD64
2004	499	Pentium 4 Prescott (90 nm)	FPU, MMX, SSE, SSE2, SSE3

Октомври 2015 г.: Xeon E3 v5 Skylake-DT (14 nm FinFET):

208 + 5 (CLMUL = Carry-less Multiplication) + 24 (BMI = Bit Manipulation Instructions)

+ 96 (FPU) + 20 (FMA = Fused Multiply-Add) =

+ 48 (MMX = Multimedia Extensions | Multiple Math Extensions | Matrix Math Extensions)

+ 68 (SSE = Streaming SIMD (Single Instruction, Multiple Data) Extensions)

+ 69 (SSE2)

+ 10 (SSE3)

+ 16 (SSSE3 = Supplemental SSE) = 515

+ 49 (SSE4.1)

+ 6 (SSE4.2)

+ 14 (x86-64)

+ 2 (ADX = Multi-Precision Add-Carry Instruction Extensions)

+ 12 (AVX = Advanced Vector Extensions)

+ 30 (AVX2)

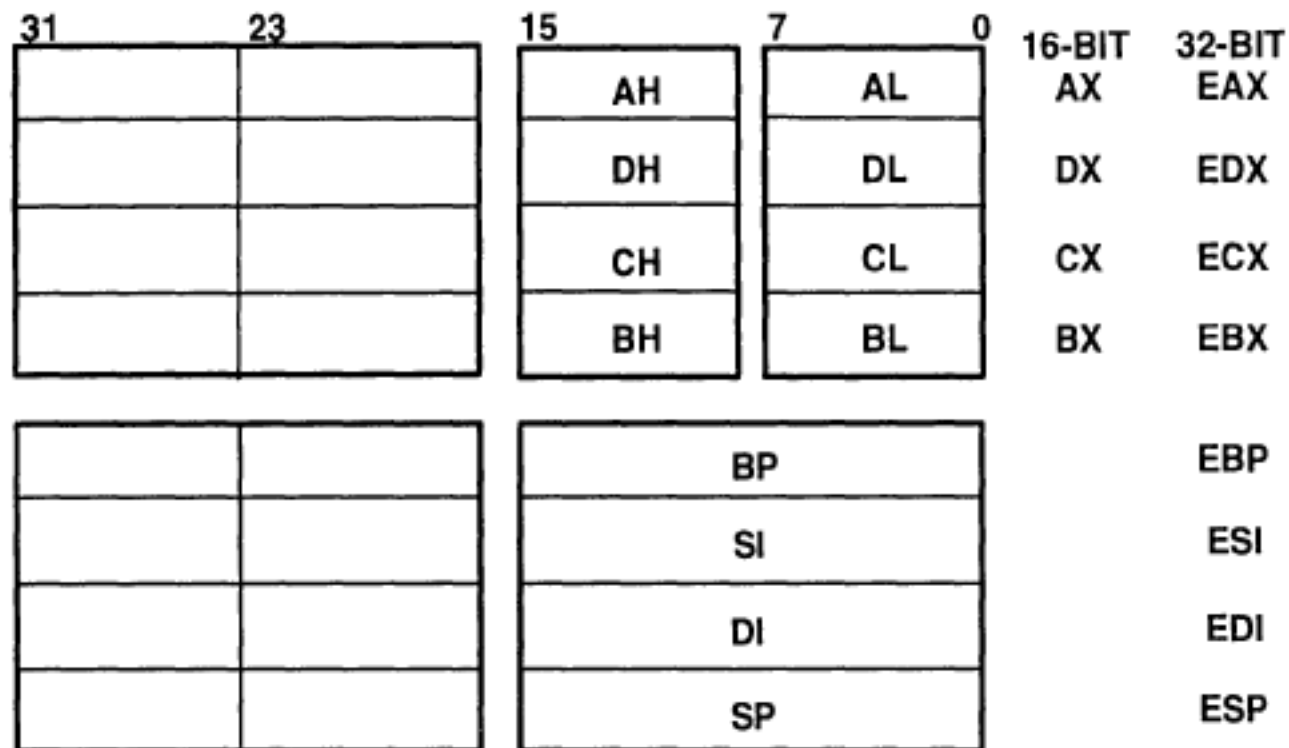
+ 13 + 6 + 8 + 8 + 8 + 18 + 2 + 44 + 12 + 16 + 63 + 6 + 10 + 16 + 12 + 9 + 2 (253 AVX3)

+ 8 (MPX = Memory Protection Extensions)

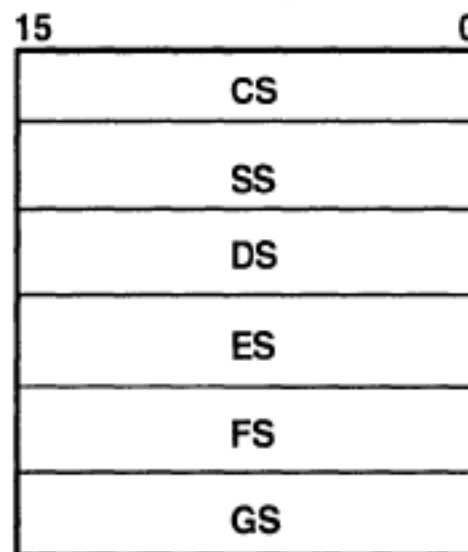
+ 4 (TSX = Transactional Synchronization Extensions) + 2 (SGX = Software Guard Extensions)

+ 10 (VT-x Virtualization) + 7 (AES-NI = Advanced Encryption Standard New Instructions) = 961

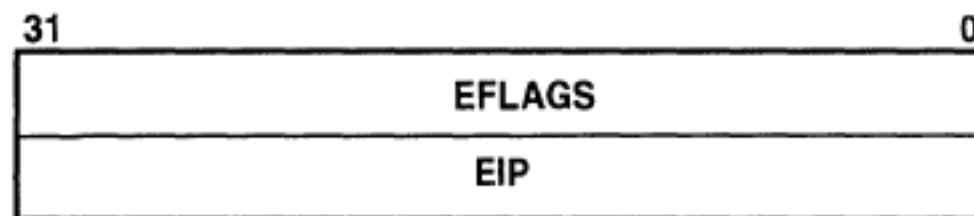
GENERAL REGISTERS



SEGMENT REGISTERS

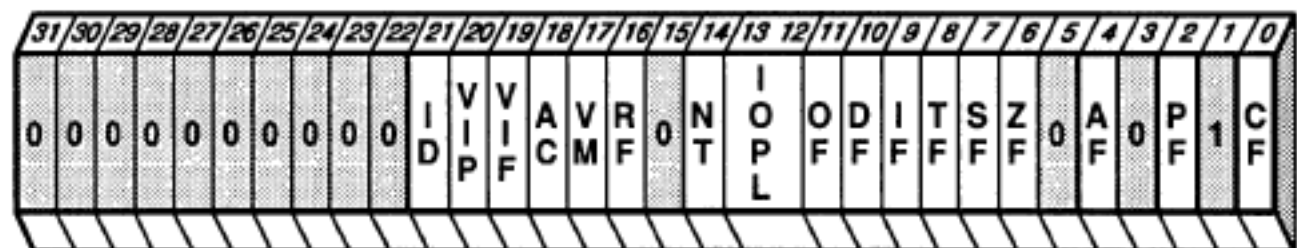


STATUS AND CONTROL REGISTERS



APM47

Figure 3-5. Application Register Set



- X ID FLAG (ID)
- X VIRTUAL INTERRUPT PENDING (VIP)
- X VIRTUAL INTERRUPT FLAG (VIF)
- X ALIGNMENT CHECK (AC)
- X VIRTUAL 8086 MODE (VM)
- X RESUME FLAG (RF)
- X NESTED TASK (NT)
- X I/O PRIVILEGE LEVEL (IOPL)
- S OVERFLOW FLAG (OF)
- C DIRECTION FLAG (DF)
- X INTERRUPT ENABLE FLAG (IF)
- X TRAP FLAG (TF)
- S SIGN FLAG (SF)
- S ZERO FLAG (ZF)
- S AUXILIARY CARRY FLAG (AF)
- S PARITY FLAG (PF)
- S CARRY FLAG (CF)

S INDICATES A STATUS FLAG
 C INDICATES A CONTROL FLAG
 X INDICATES A SYSTEM FLAG



BIT POSITIONS SHOWN AS 0 OR 1 ARE INTEL RESERVED.
 DO NOT USE. ALWAYS SET THEM TO THE VALUE PREVIOUSLY READ.

APM45

Figure 3-9. EFLAGS Register

Table 3-2. Status Flags

Name	Purpose	Condition Reported
OF	Overflow	Result exceeds positive or negative limit of number range
SF	Sign	Result is negative (less than zero)
ZF	Zero	Result is zero
AF	Auxiliary carry	Carry out of bit position 3 (used for BCD)
PF	Parity	Low byte of result has even parity (even number of set bits)
CF	Carry flag	Carry out of most significant bit of result

regparm

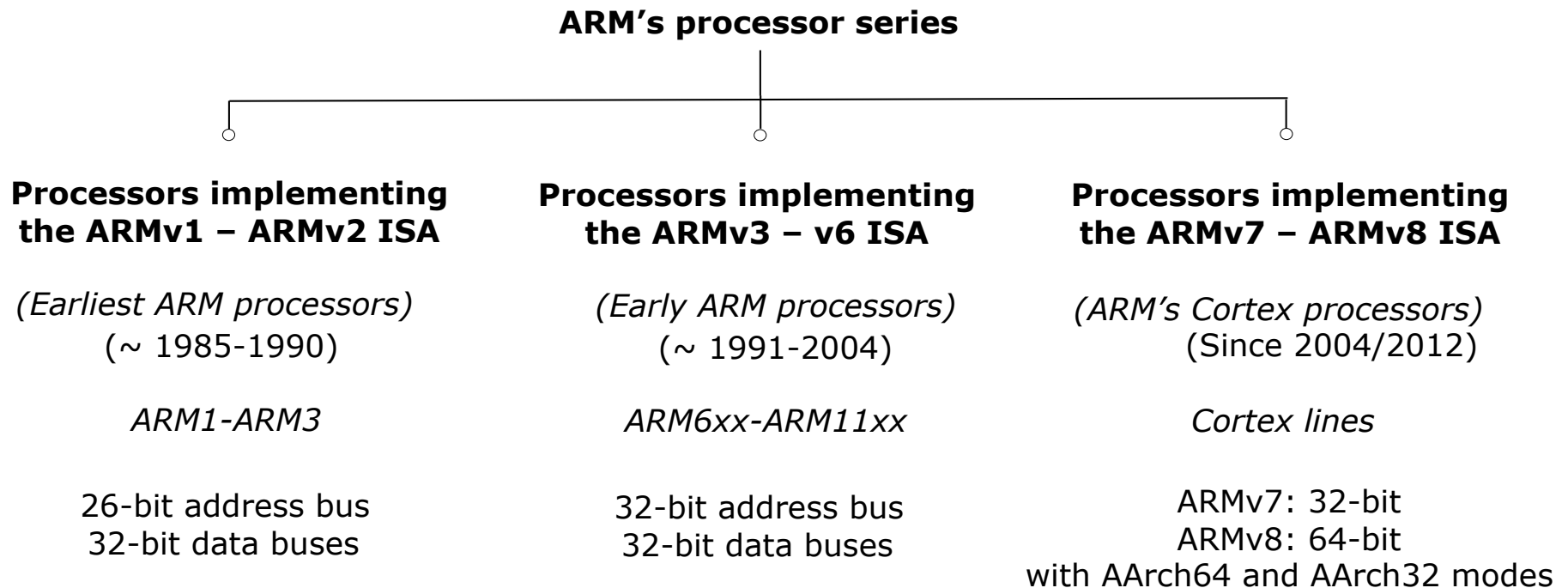
✧ **regparm** (**number**)

- ◆ On the Intel 386, the **regparm** attribute causes the compiler to pass up to **number** integer arguments in registers **EAX**, **EDX**, and **ECX** instead of on the stack.
- ◆ Functions that take a variable number of arguments will continue to be passed all of their arguments on the stack.

3.1 Overview of ARM's processor lines (1)

3.1 Overview of ARM's processor series

Subsequently, we give an overview of ARM's processor series **subdivided into three sections, according their underlying ISAs**, as follows.



Процесорната фамилия ARM (Advanced RISC Machines) се състои от RISC микропроцесори, които имат 16 регистъра (фиг. 12.1) с общо предназначение с имена от R0 до R15. Регистрите са 32-битови. Те могат да съдържат както адреси, така и данни. Последният регистър R15 се използва за програмен брояч (PC), а регистър R13 служи за организиране на програмен стек (SP). Регистър R14 (LR) се използва като регистър, съдържащ адреса за връщане след подпрограма. Регистрите могат да се използват за съхранение на 8, 16 и 32-битови числа.



R0
R1
R2
R3
R4
R5
R6
R7
R8
R9
R10
R11
R12
R13 (Stack pointer – SP)
R14 (link register – LR)
R15 (PC)

Фигура 12.1. Регистри на процесора ARM

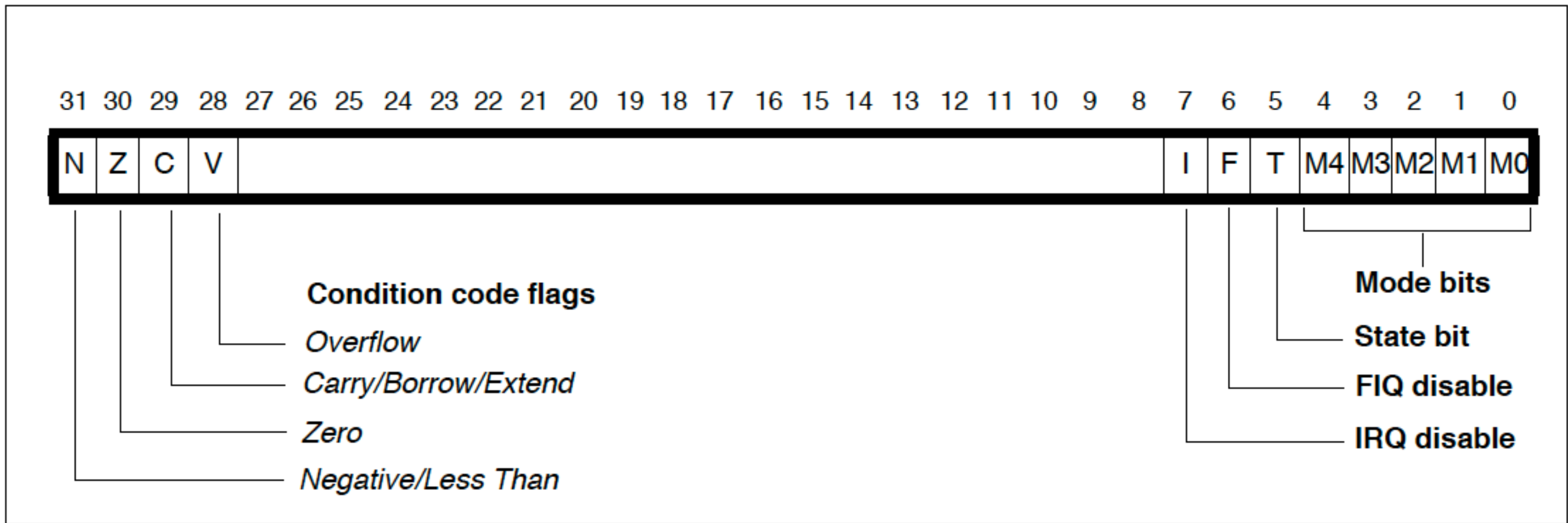
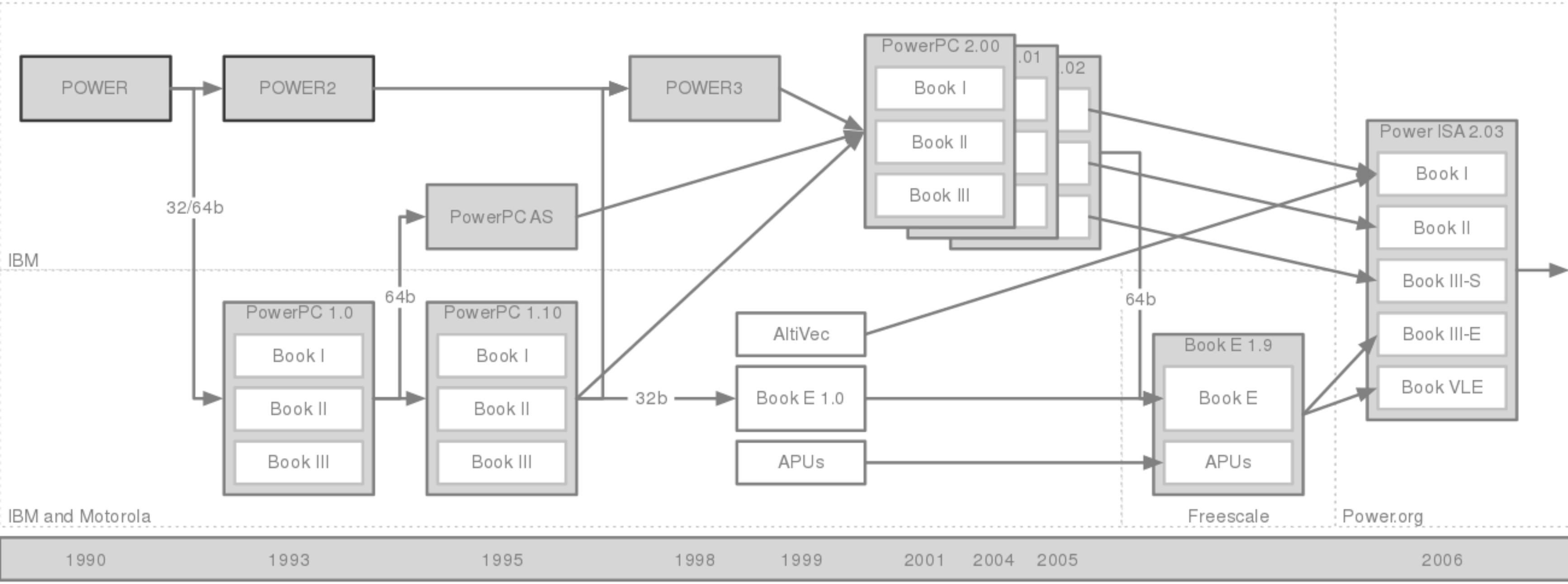


Figure 3-4: Program Status Register format

The condition code flags

The N, Z, C and V (Negative, Zero, Carry and oVerflow) bits are collectively known as the *condition code flags*. The condition code flags in the CPSR can be changed as a result of arithmetic and logical operations in the processor, and can be tested by all ARM instructions to determine if the instruction is to be executed. All ARM instructions may be executed conditionally

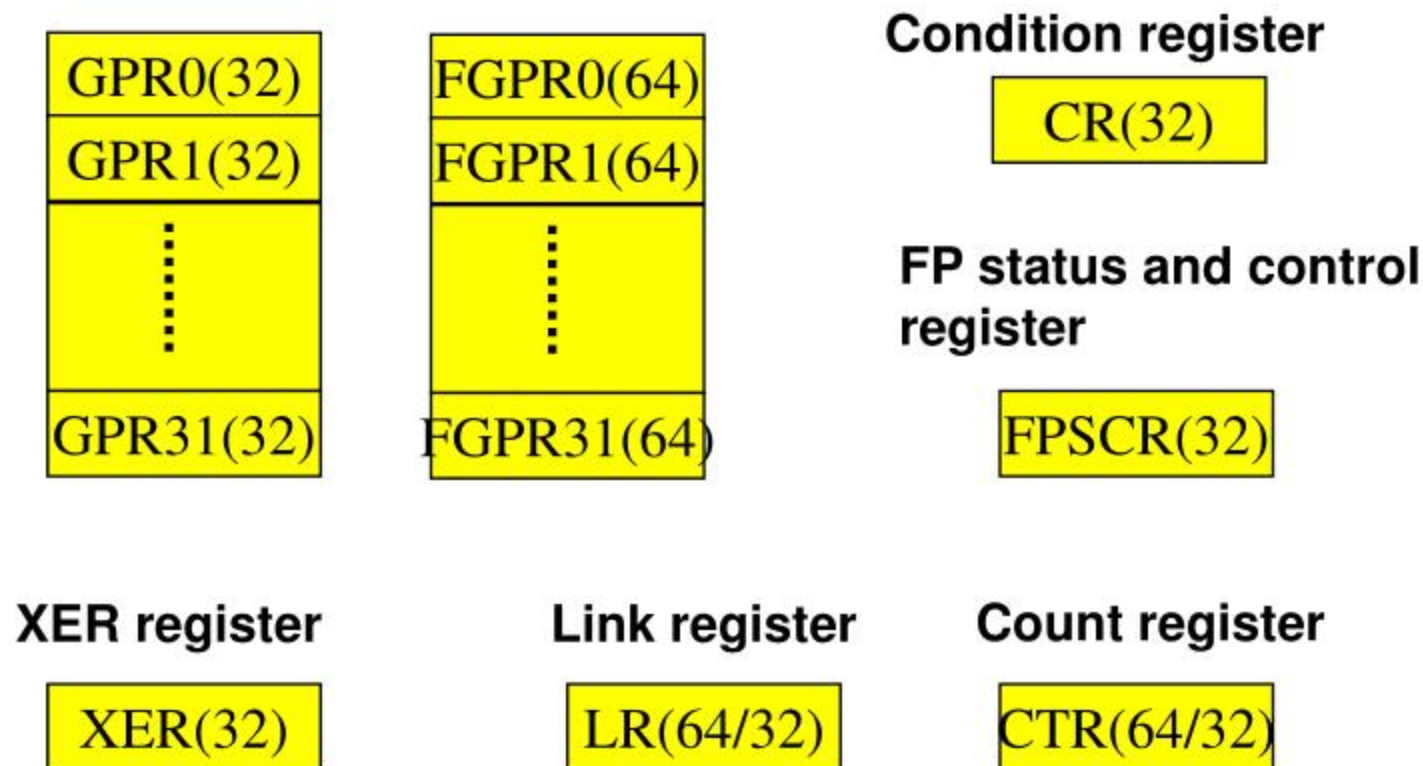
Register	Synonym	Special	Role in the procedure call standard	Preserve across function calls?
R15		PC	The Program Counter.	Special role register Special role register Special role register
R14		LR	The Link Register.	
R13		SP	The Stack Pointer.	
R12		IP	The Intra-Procedure-call scratch register.	No
R11	v8	FP	ARM-state variable-register 8. ARM-state frame pointer.	Yes, if used
R10	v7	SL	ARM-state variable-register 7. Stack Limit pointer in stack-checked variants.	Yes, if used
R9	v6	SB	ARM-state v-register 6. Static Base in PID,/re-entrant/shared-library variants.	Yes, if used
R8	v5		ARM-state variable-register 5.	Yes, if used
R7	v4	WR	Variable register (v-register) 4. Thumb-state Work Register.	Yes, if used
R6	v3		Variable register (v-register) 3.	Yes, if used
R5	v2		Variable register (v-register) 2.	Yes, if used
R4	v1		Variable register (v-register) 1.	Yes, if used
R3	a4		Argument/result/scratch register 4.	No
R2	a3		Argument/result/scratch register 3.	No
R1	a2		Argument/result/scratch register 2.	No
R0	a1		Argument/result/scratch register 1.	No



PowerPC programming model

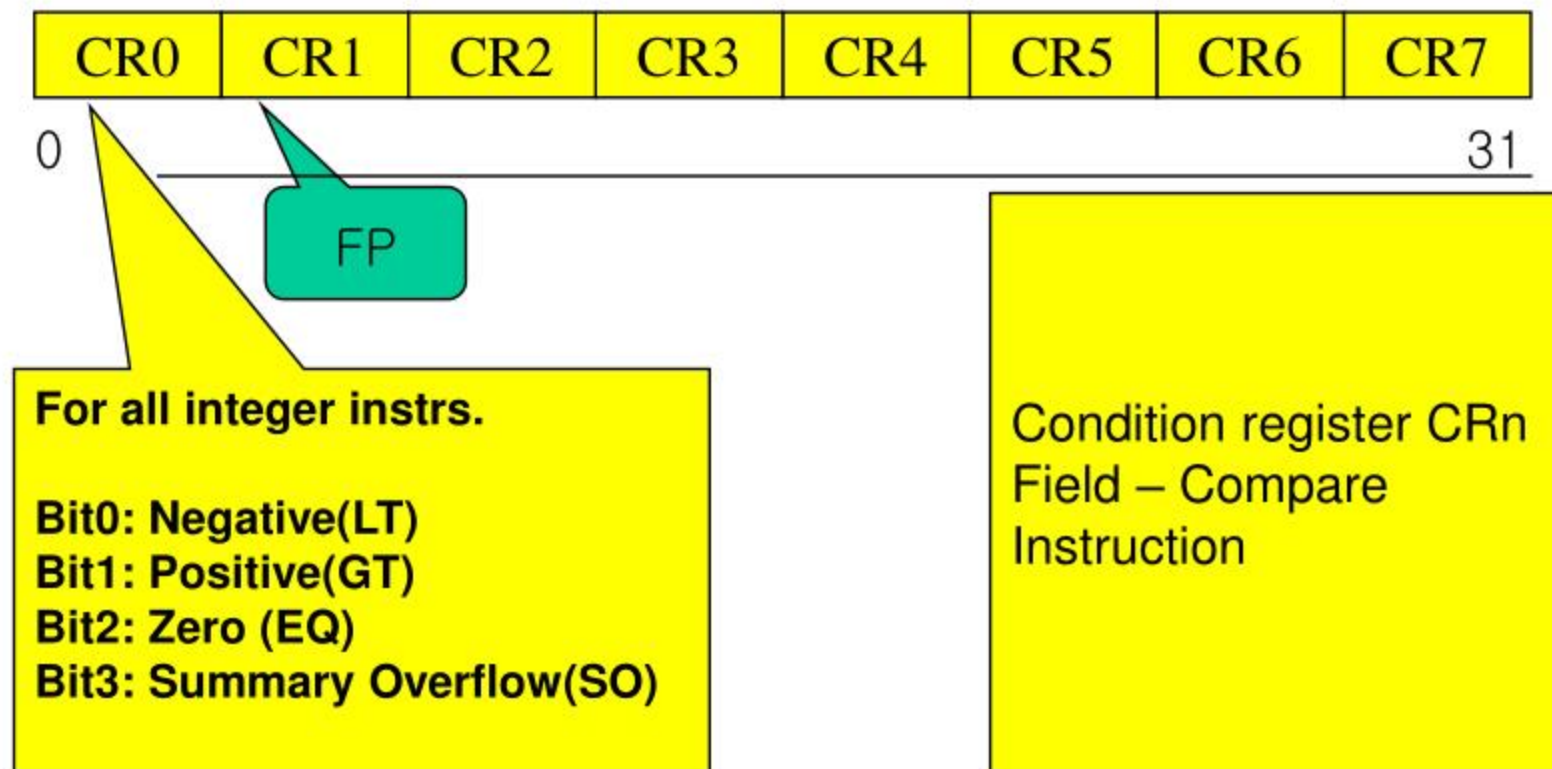
- Register Set

- User Model – UISA (32-bit architecture)



Condition Registers (CR)

- For testing and branching



[back](#)

XER Register (XER)

☐ Reserved

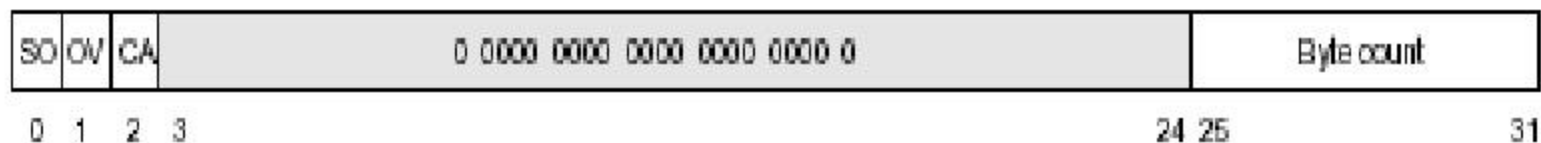


Figure 2-6. XER Register

back

Table 2-6. XER Bit Definitions

Bits	Name	Description
0	SO	Summary overflow. The summary overflow bit (SO) is set whenever an instruction (except mtspr) sets the overflow bit (OV). Once set, the SO bit remains set until it is cleared by an mtspr instruction (specifying the XER) or an mcrxr instruction. It is not altered by compare instructions, nor by other instructions (except mtspr to the XER, and mcrxr) that cannot overflow. Executing an mtspr instruction to the XER, supplying the values zero for SO and one for OV, causes SO to be cleared and OV to be set.
1	OV	Overflow. The overflow bit (OV) is set to indicate that an overflow has occurred during execution of an instruction. Add, subtract from, and negate instructions having OE = 1 set the OV bit if the carry out of the msb is not equal to the carry out of the msb + 1, and clear it otherwise. Multiply low and divide instructions having OE = 1 set the OV bit if the result cannot be represented in 64 bits (mulld, divd, divdu) or in 32 bits (mullw, divw, divwu), and clear it otherwise. The OV bit is not altered by compare instructions that cannot overflow (except mtspr to the XER, and mcrxr).

XER Register (XER), cont'd



Table 2-6. XER Bit Definitions (continued)

Bits	Name	Description
2	CA	Carry. Set during execution of the following instructions: • Add carrying, subtract from carrying, add extended, and subtract from extended instructions set CA if there is a carry out of the msb, and clear it otherwise. • Shift right algebraic instructions set CA if any 1 bits have been shifted out of a negative operand, and clear it otherwise. The CA bit is not altered by compare instructions, nor by other instructions that cannot carry (except shift right algebraic, mtspr to the XER, and mcrxr).
3–24	—	Reserved
25–31	Byte count	This field specifies the number of bytes to be transferred by a Load String Word Indexed (lswx) or Store String Word Indexed (stswx) instruction.

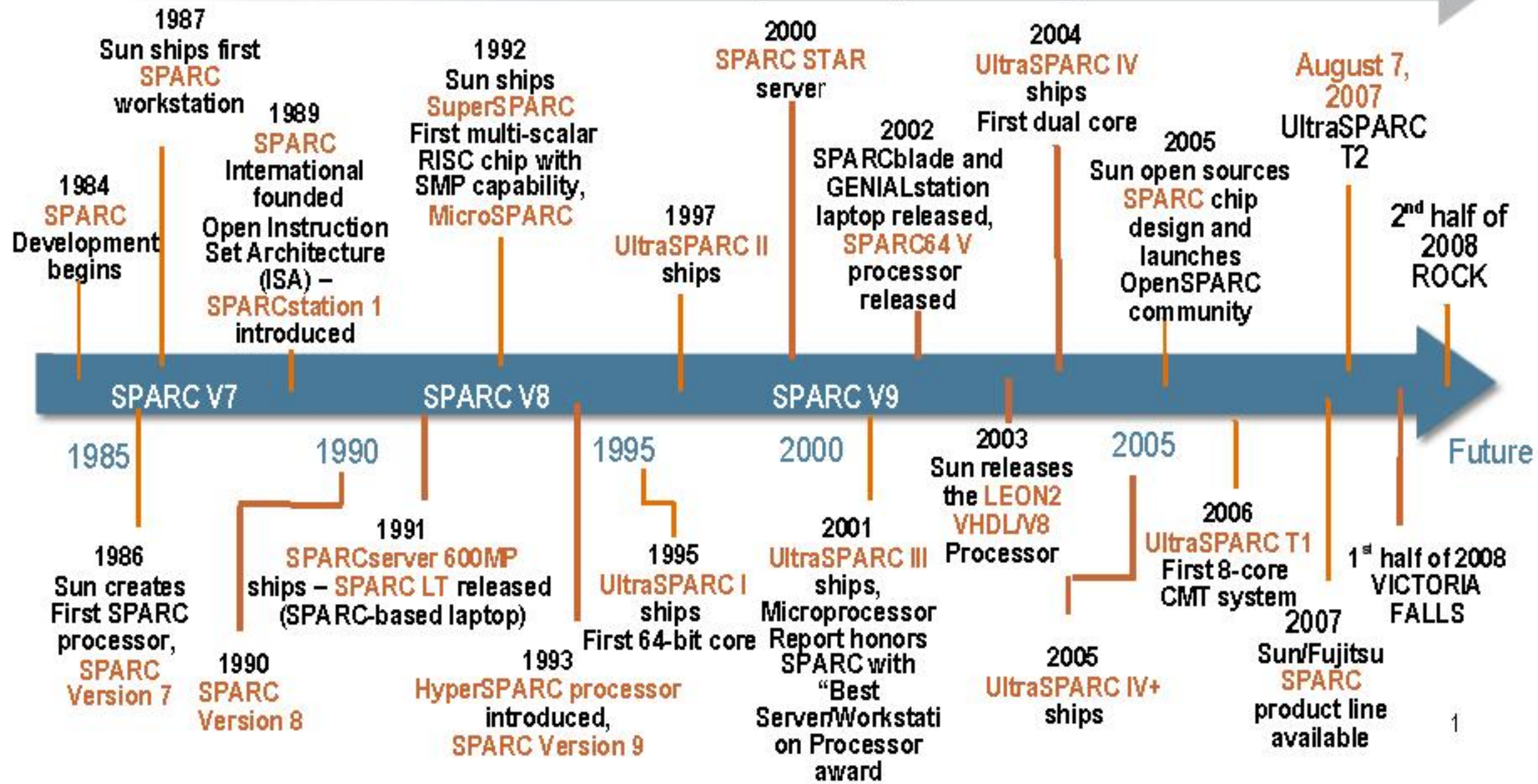
Table 13-1 GPR Register Usage Conventions

GPR	Type	Must be Preserved?	Usage
r0	volatile	no	used in prolog/epilog code
r1	dedicated	yes	stack pointer
r2			table of contents (TOC) pointer
r3	volatile	no	1st fixed-point parameter 1st word of return value
r4			2nd fixed-point parameter 2nd word of return value
r5	volatile	no	3rd fixed-point parameter
...			...
r10			8th fixed-point parameter
r11	volatile	no	environment pointer (if needed)
r12			used by global linkage routines
r13	non-volatile	yes	general registers that must be preserved across function calls
...			
r31			

The History of SPARC®

Over Two Decades of Continuous Technical Innovation

100% Binary Compatibility



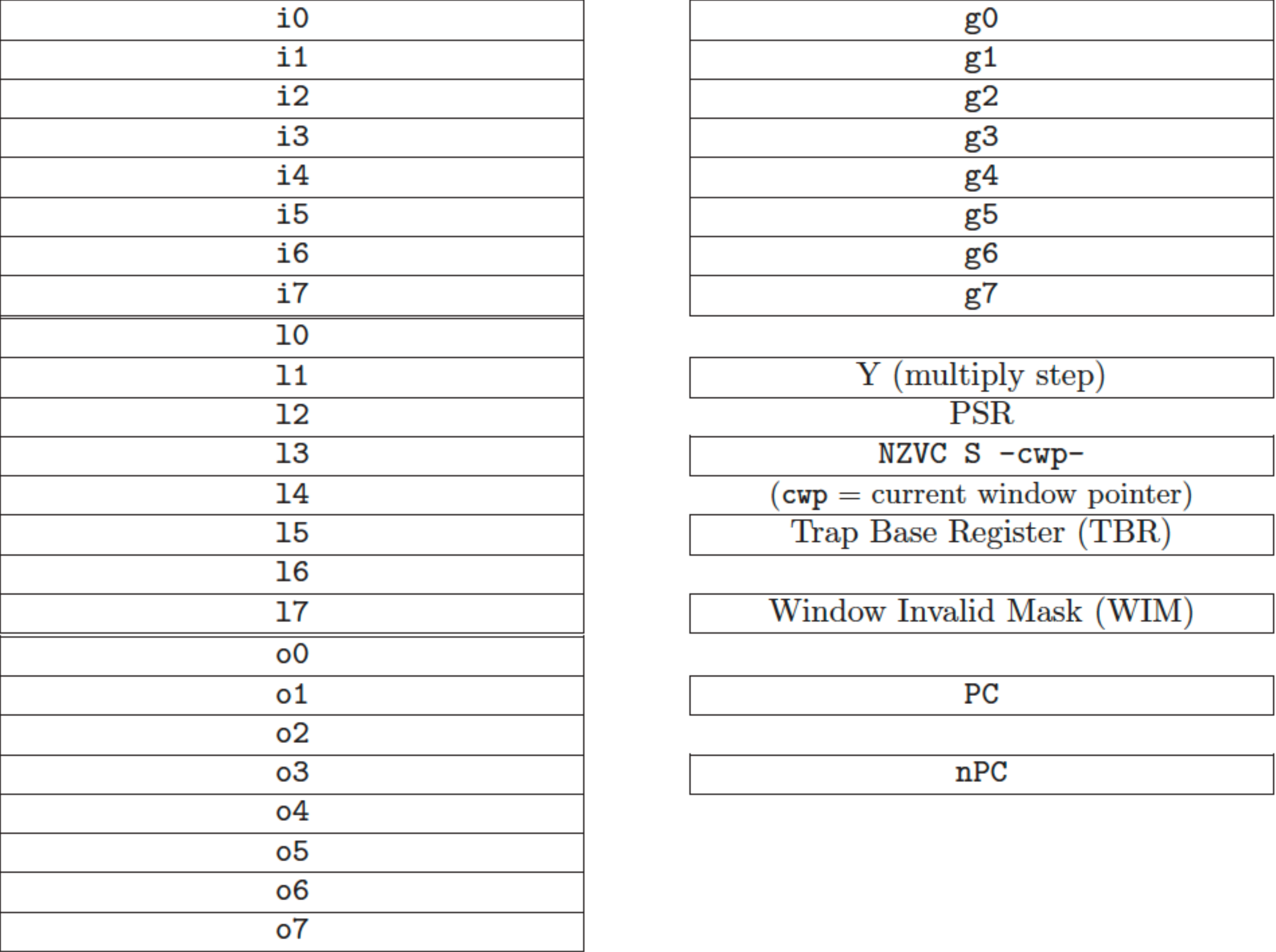


Figure 2.1: SPARC Programming Model

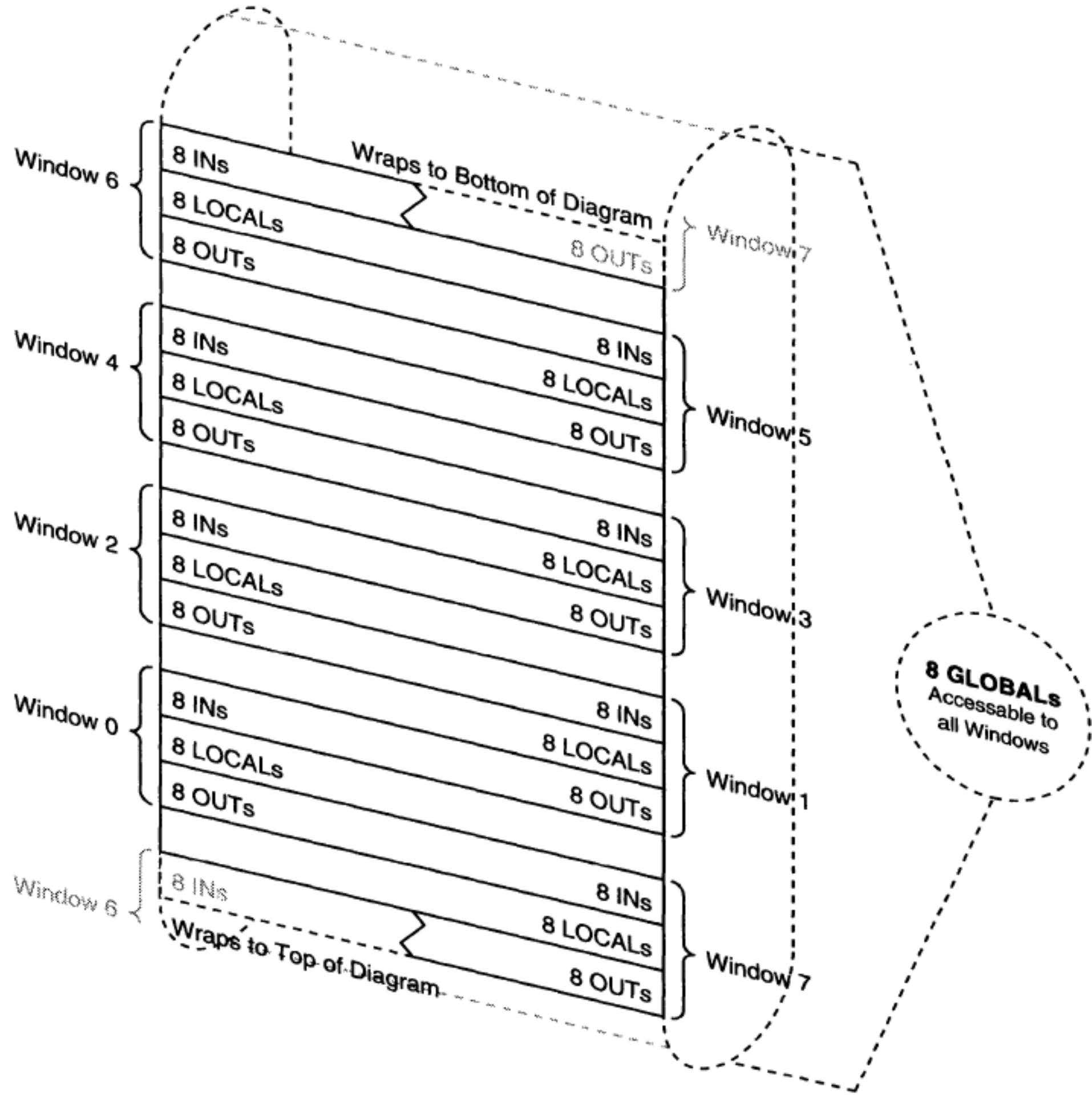


Figure 2-2. Register Windows

Register	Synonyms	Usage
%g0	%r0	Always discards writes and returns zero First of seven registers for data with global context
%g1	%r1	
%g2	%r2	
%g3	%r3	
%g4	%r4	
%g5	%r5	
%g6	%r6	
%g7	%r7	
%o0	%r8	First of six registers for local data and arguments to called subroutines Stack pointer Called subroutine return address
%o1	%r9	
%o2	%r10	
%o3	%r11	
%o4	%r12	
%o5	%r13	
%sp	%r14 %o6	
%o7	%r15	
%l0	%r16	First of eight registers for local variables
%l1	%r17	
%l2	%r18	
%l3	%r19	
%l4	%r20	
%l5	%r21	
%l6	%r22	
%l7	%r23	
%i0	%r24	First of six registers for incoming subroutine arguments Frame pointer Subroutine return address
%i1	%r25	
%i2	%r26	
%i3	%r27	
%i4	%r28	
%i5	%r29	
%fp	%r30 %i6	
%i7	%r31	

In the HP Calculator, the last number computed could be tested. For example, there was an instruction `ifeq`, which would skip the next instruction in line if the result last computed was zero. A similar technique is used in many computers, in which the state of the execution of each instruction may be tested. In order to do this, only information about the result need be kept, not the result itself. The state of execution is saved in terms of four variables:

Z whether the result was zero

N whether the result was negative

V whether execution resulted in a number too large to store in the register

C whether execution resulted in a number that generated a carry out of the register

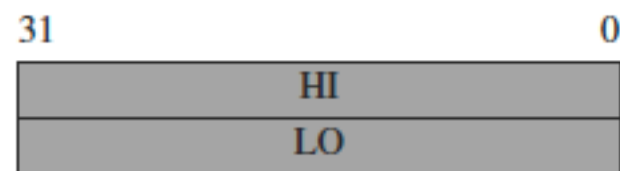
This information is kept in four variables, the integer condition codes: Z, N, V, and, C.

TABLE 1.1 Milestones in MIPS CPUs

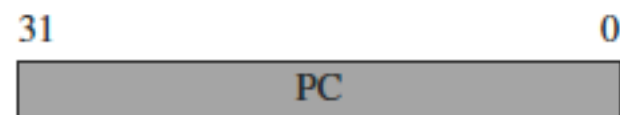
<i>Year</i>	<i>Designer/model/ clock rate (MHz)</i>	<i>Instruction set</i>	<i>Cache (I+D)</i>	<i>Notes</i>
1987	MIPS R2000-16	MIPS I	External: 4 K+4 K to 32 K+32 K	External (R2010) FPU.
1990	IDT R3051-20		4 K+1 K	The first embedded MIPS CPU with on-chip cache and progenitor of a family of pin-compatible parts.
1991	MIPS R4000-100	MIPS III	8 K+8 K	Integrates FPU and L2 cache controller with pinout option. Full 64-bit CPU—but five years later, few MIPS CPUs were exploiting their 64-bit instruction set. Long pipeline and half-speed interface help achieve high clock rates.
1993	IDT/QED R4600-100		16 K+16 K	QED's brilliantly tuned redesign is much faster than R4000 or R4400 at the same clock rate—partly because it returned to the classic MIPS five-stage pipeline. Important to SGI's fast and affordable low-end Indy workstation and Cisco's routers.
1995	NEC/MIPS Vr4300-133		16 K+8 K	Low cost, low power but full-featured R4000 derivative. Initially aimed at Nintendo 64 games console, but embedded uses include HP's LJ4000 laser printers.
1996	MIPS R10000-200	MIPS IV	32 K+32 K	Bristling with microprocessor innovations, the R10000 is not at all simple. The main MIPS tradition it upholds is that of taking a principle to extremes. The result was hot, unfriendly, but with unmatched performance/MHz.
1998	QED RM7000		16 K+16 K+256 K L2	The first MIPS CPU with on-chip L2 cache, this powered generations of high-end laser printers and Internet routers.
2000	MIPS 4 K core family	MIPS32	16 K+16 K (typ)	The most successful MIPS core to date—synthesizable and frugal.
2001	Alchemy AU-1000		16 K+16 K	If you wanted 400 MHz for 500 mW, this was the only show in town. But it lost markets.
2001	Broadcom BCM1250	MIPS64	32 K+32 K+256 K L2	Dual-CPU design at 600 MHz+ (the L2 is shared).
2002	PMC-Sierra RM9000x2	MIPS64	16 K+16 K+256 K L2	Dual-CPU design at 1 GHz (the L2 is NOT shared; each CPU has its own 256 K). First MIPS CPU to reach 1 GHz.
2003	Intrinsity FastMath	MIPS32	16 K+16 K+1 M L2	Awesome 2-GHz CPU with vector DSP did not find a market.
2003	MIPS 24 K core	MIPS32 R2	At 500 MHz in synthesizable logic, a solidly successful core design.	
2005	MIPS 34 K core	MIPS32+MT ASE	32 K+32 K (typ)	MIPS multithreading pioneer.

	31	0
zero	0	
at	1	
v0	2	
v1	3	
a0	4	
a1	5	
a2	6	
a3	7	
t0	8	
t1	9	
t2	10	
t3	11	
t4	12	
t5	13	
t6	14	
t7	15	
s0	16	
s1	17	
s2	18	
s3	19	
s4	20	
s5	21	
s6	22	
s7	23	
t8	24	
t9	25	
k0	26	
k1	27	
gp	28	
sp	29	
fp	30	
ra	31	

General-purpose registers



Multiply and divide registers



Program counter

Figure 4.1 MIPS registers. All registers are 32-bits wide.

Table 4.1 MIPS registers and their conventional usage

Register name	Number	Intended usage
zero	0	Constant 0
\$at	1	Reserved for assembler
\$v0, \$v1	2, 3	Results of a procedure
\$a0, \$a1, \$a2, \$a3	4–7	Arguments 1–4 (not preserved across call)
\$t0–\$t7	8–15	Temporary (not preserved across call)
\$s0–\$s7	16–23	Saved temporary (preserved across call)
\$t8, \$t9	24, 25	Temporary (not preserved across call)
\$k0, \$k1	26, 27	Reserved for OS kernel
\$gp	28	Pointer to global area
\$sp	29	Stack pointer
\$fp	30	Frame pointer (if needed); otherwise, a saved register \$s8
\$ra	31	Return address (used to return from a procedure)

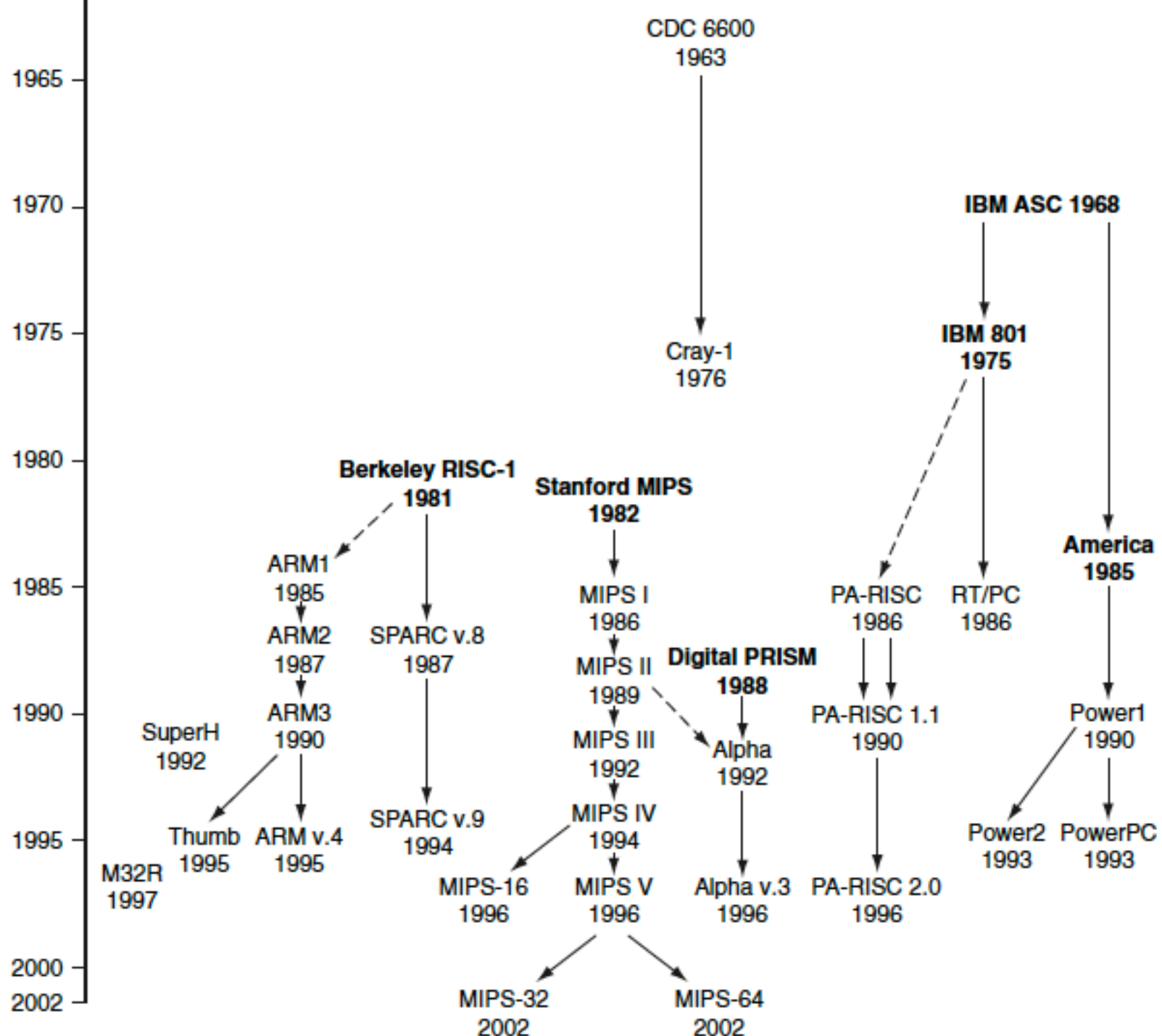


FIGURE E.17.2 The lineage of RISC instruction sets. Commercial machines are shown in plain text and research machines in bold. The CDC 6600 and Cray-1 were load-store machines with register 0 fixed at 0, and separate integer and floating-point registers. Instructions could not cross word boundaries. An early IBM research machine led to the 801 and America research projects, with the 801 leading to the unsuccessful RT/PC and America leading to the successful Power architecture. Some people who worked on the 801 later joined Hewlett-Packard to work on the PA-RISC. The two university projects were the basis of MIPS and SPARC machines. According to Furber [1996], the Berkeley RISC project was the inspiration of the ARM architecture. While ARM1, ARM2, and ARM3 were names of both architectures and chips, ARM version 4 is the name of the architecture used in ARM7, ARM8, and StrongARM chips. (There are no ARM v.4 and ARM5 chips, but ARM6 and early ARM7 chips use the ARM3 architecture.) DEC built a RISC microprocessor in 1988 but did not introduce it. Instead, DEC shipped workstations using MIPS microprocessors for three years before they brought out their own RISC instruction set, Alpha 21064, which is very similar to MIPS III and PRISM. The Alpha architecture has had small extensions, but they have not been formalized with version numbers; we used version 3 because that is the version of the reference manual. The Alpha 21164A chip added byte and halfword loads and stores, and the Alpha 21264 includes the MAX multimedia and bit count instructions. Internally, Digital names chips after the fabrication technology: EV4 (21064), EV45 (21064A), EV5 (21164), EV56 (21164A), and EV6 (21264). “EV” stands for “extended VAX.”

Year Published	Instruction Set Architecture	Year Published	Instruction Set Architecture
1964	CDC 6600 [1]	1992	DEC Alpha [12]
1981	RISC I [2] / RISC II [3]	1992	MIPS III [13]
1984	SOAR (RISC III) [4]	1992	IBM PowerPC [14]
1984	Intel i960 [5]	1992	Torrent T0 [15][16][17]
1985	IBM RP3 [6][7]	1994	MIPS IV [18]
1987	ARMv2 [8]	1995	PA-RISC 2.0 [19]
1988	SPUR (RISC IV) [9]	1997	Hitachi SH-4 [20]
1990	DLX [10]	2002	ARMv6 [21]
1990	SPARCV8 [11]	2003	Cray X1 [22]

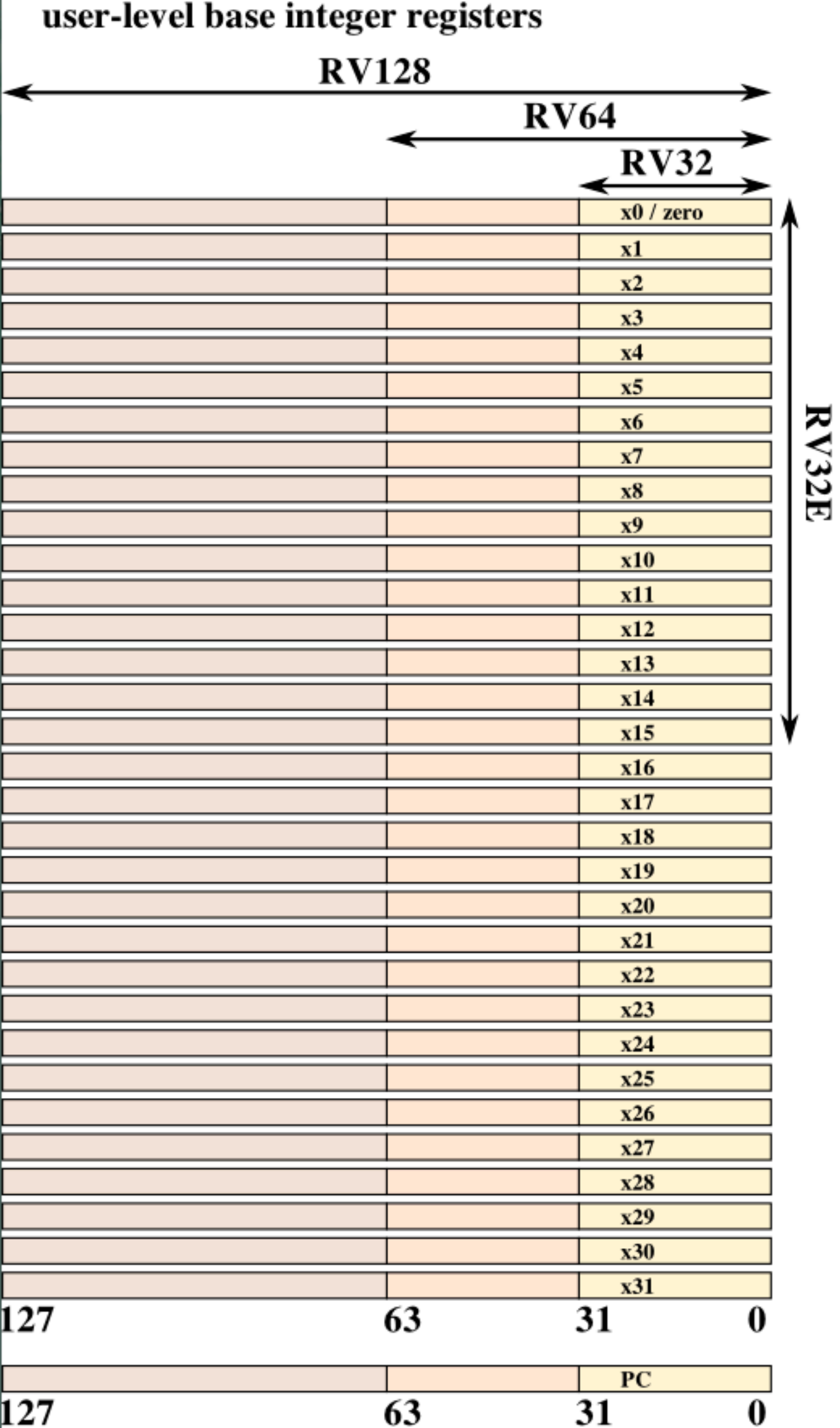


RISC-V

Designer	University of California, Berkeley
Bits	32, 64, 128
Introduced	2010
Version	Unprivileged ISA 2.2, ^[1] privileged ISA 1.11 ^[2]
Design	RISC
Type	Load-store
Encoding	Variable
Branching	Compare-and-branch
Endianness	L ^{[1][3]}
Extensions	M, A, F, D, Q, C
Open	Yes

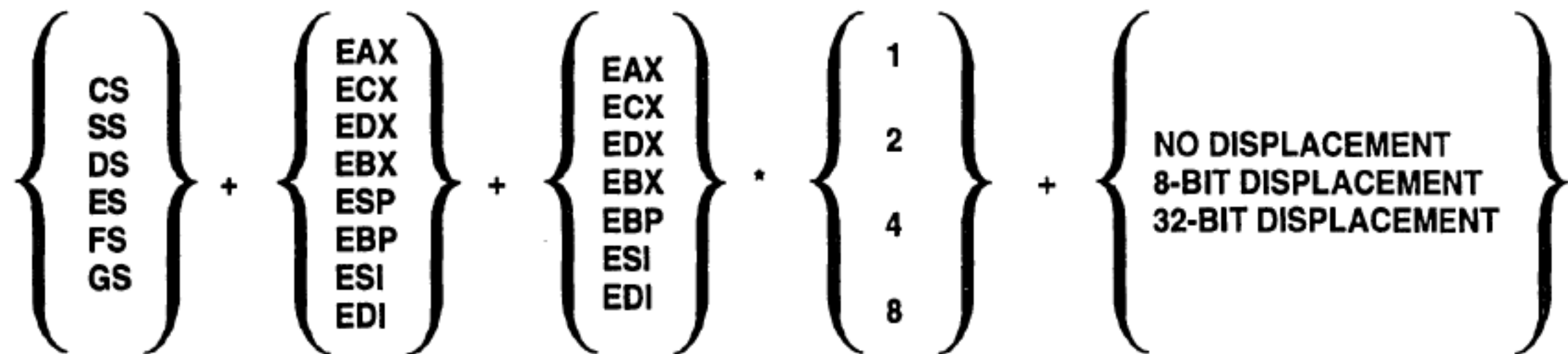
Registers

General purpose	16, 32 (including one always-zero register)
Floating point	32 (optional)



Register	ABI Name	Description	Saver
x0	zero	hardwired zero	-
x1	ra	return address	Caller
x2	sp	stack pointer	Callee
x3	gp	global pointer	-
x4	tp	thread pointer	-
x5-7	t0-2	temporary registers	Caller
x8	s0 / fp	saved register / frame pointer	Callee
x9	s1	saved register	Callee
x10-11	a0-1	function arguments / return values	Caller
x12-17	a2-7	function arguments	Caller
x18-27	s2-11	saved registers	Callee
x28-31	t3-6	temporary registers	Caller

$$\text{SEGMENT} + \text{BASE} + (\text{INDEX} * \text{SCALE}) + \text{DISPLACEMENT}$$



APM42

Figure 3-10. Effective Address Computation

The scaling factor permits efficient indexing into an array when the array elements are 2, 4, or 8 bytes. The scaling of the index register is done in hardware at the time the address is evaluated. This eliminates an extra shift or multiply instruction.

The base, index, and displacement components can be used in any combination; any of these components can be null. A scale factor can be used only when an index also is used. Each possible combination is useful for data structures commonly used by programmers in high-level languages and assembly language. Suggested uses for some combinations of address components are described below.

ARM Addressing Modes

Quick Reference Card

Addressing Mode 2

Immediate offset	<code>[Rn, #+/-12bit_Offset]</code>
Register offset	<code>[Rn, +/-Rm]</code>
Scaled register offset	<code>[Rn, +/-Rm, LSL #5bit_shift_imm]</code> <code>[Rn, +/-Rm, LSR #5bit_shift_imm]</code> <code>[Rn, +/-Rm, ASR #5bit_shift_imm]</code> <code>[Rn, +/-Rm, ROR #5bit_shift_imm]</code> <code>[Rn, +/-Rm, RRX]</code>
Pre-indexed offset	
Immediate	<code>[Rn, #+/-12bit_Offset]!</code>
Register	<code>[Rn, +/-Rm]!</code>
Scaled register	<code>[Rn, +/-Rm, LSL #5bit_shift_imm]!</code> <code>[Rn, +/-Rm, LSR #5bit_shift_imm]!</code> <code>[Rn, +/-Rm, ASR #5bit_shift_imm]!</code> <code>[Rn, +/-Rm, ROR #5bit_shift_imm]!</code> <code>[Rn, +/-Rm, RRX]!</code>
Post-indexed offset	
Immediate	<code>[Rn], #+/-12bit_Offset</code>
Register	<code>[Rn], +/-Rm</code>
Scaled register	<code>[Rn], +/-Rm, LSL #5bit_shift_imm</code> <code>[Rn], +/-Rm, LSR #5bit_shift_imm</code> <code>[Rn], +/-Rm, ASR #5bit_shift_imm</code> <code>[Rn], +/-Rm, ROR #5bit_shift_imm</code> <code>[Rn], +/-Rm, RRX]</code>

Addressing Mode 4 (Load)			
Addressing Mode		Stack Type	
IA	Increment After	FD	Full Descending
IB	Increment Before	ED	Empty Descending
DA	Decrement After	FA	Full Ascending
DB	Decrement Before	EA	Empty Ascending

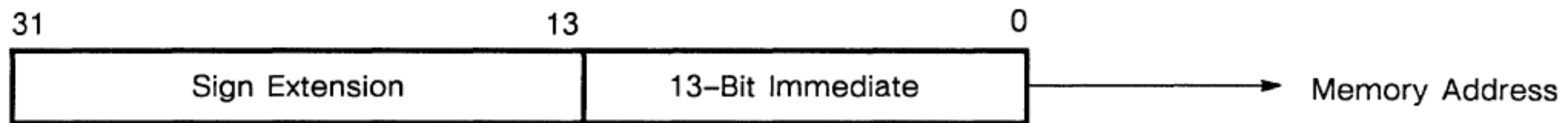
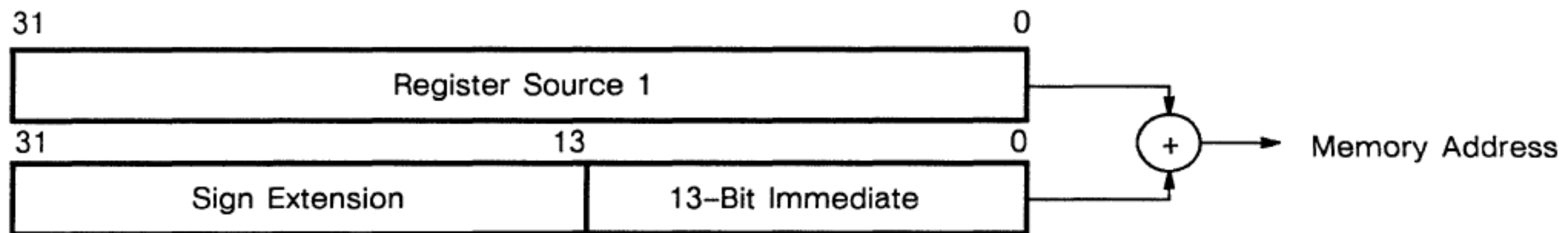
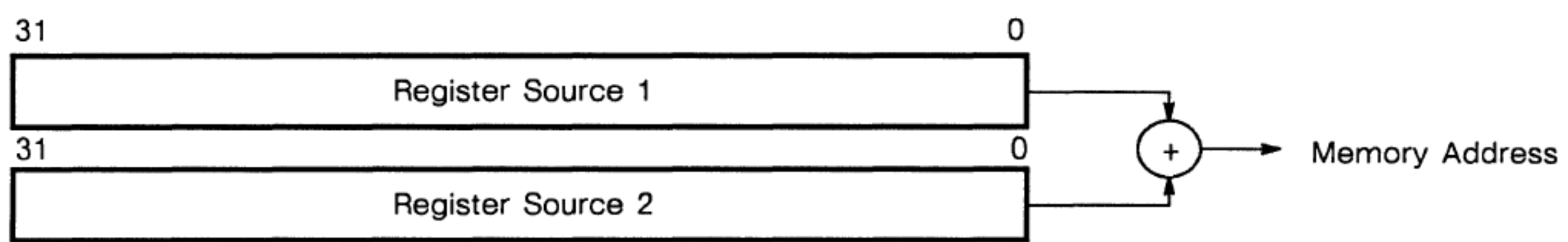
Addressing Mode 4 (Store)			
Addressing Mode		Stack Type	
IA	Increment After	EA	Empty Ascending
IB	Increment Before	FA	Full Ascending
DA	Decrement After	ED	Empty Descending
DB	Decrement Before	FD	Full Descending

Load and store instructions

Addressing modes

The PowerPC has only two addressing modes, but combining them with the load/store instructions options yields many possibilities. The addressing modes (using **lwz** as an example) are:

- **lwz** rD, offset(rA|0) → Register-indirect with immediate offset
→ $EA = (rA + \text{offset})$ or $(0 + \text{offset})$
→ Offset is a 16 bit signed immediate value
- **lwzx** rD, (rA|0), rB → Register-indirect with indexing
→ $EA = (rA + rB)$ or $(0 + rB)$



LOAD/STORE

2.4 Loading and Storing: Addressing Modes

As mentioned previously, there is only one addressing mode.¹ Any load or store machine instruction can be written:

```
lw          $1, offset ($2)
```

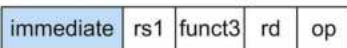
You can use any registers for the destination and source. The offset is a signed, 16-bit number (and so can be anywhere between -32768 and 32767); the program address used for the load is the sum of **\$2** and the offset. This address mode is normally enough to pick out a particular member of a C structure (offset being the distance between the start of the structure and the member required). It implements an array indexed by a constant; it is enough to reference function variables from the stack or frame pointer and to provide a reasonable-sized global area around the **gp** value for static and extern variables.

The assembler provides the semblance of a simple direct addressing mode to load the values of memory variables whose address can be computed at link time.

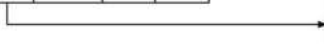
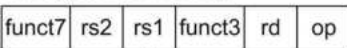
More complex modes such as double-register or scaled index must be implemented with sequences of instructions.

The only addressing mode for loads and stores is adding a sign-extended 12-bit immediate to a register, called displacement addressing mode in x86-32 [Irvine2014].

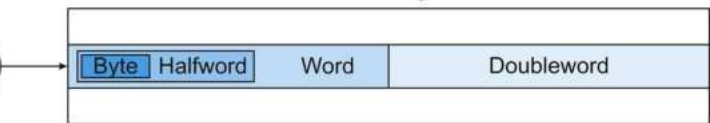
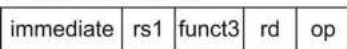
1. Immediate addressing



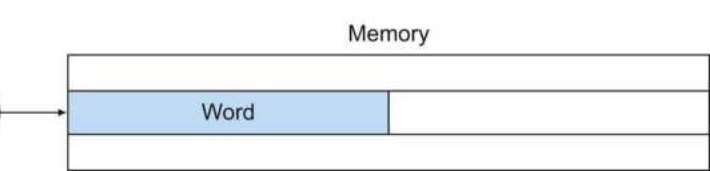
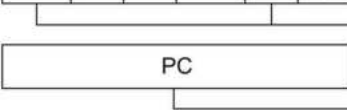
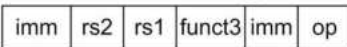
2. Register addressing



3. Base addressing



4. PC-relative addressing



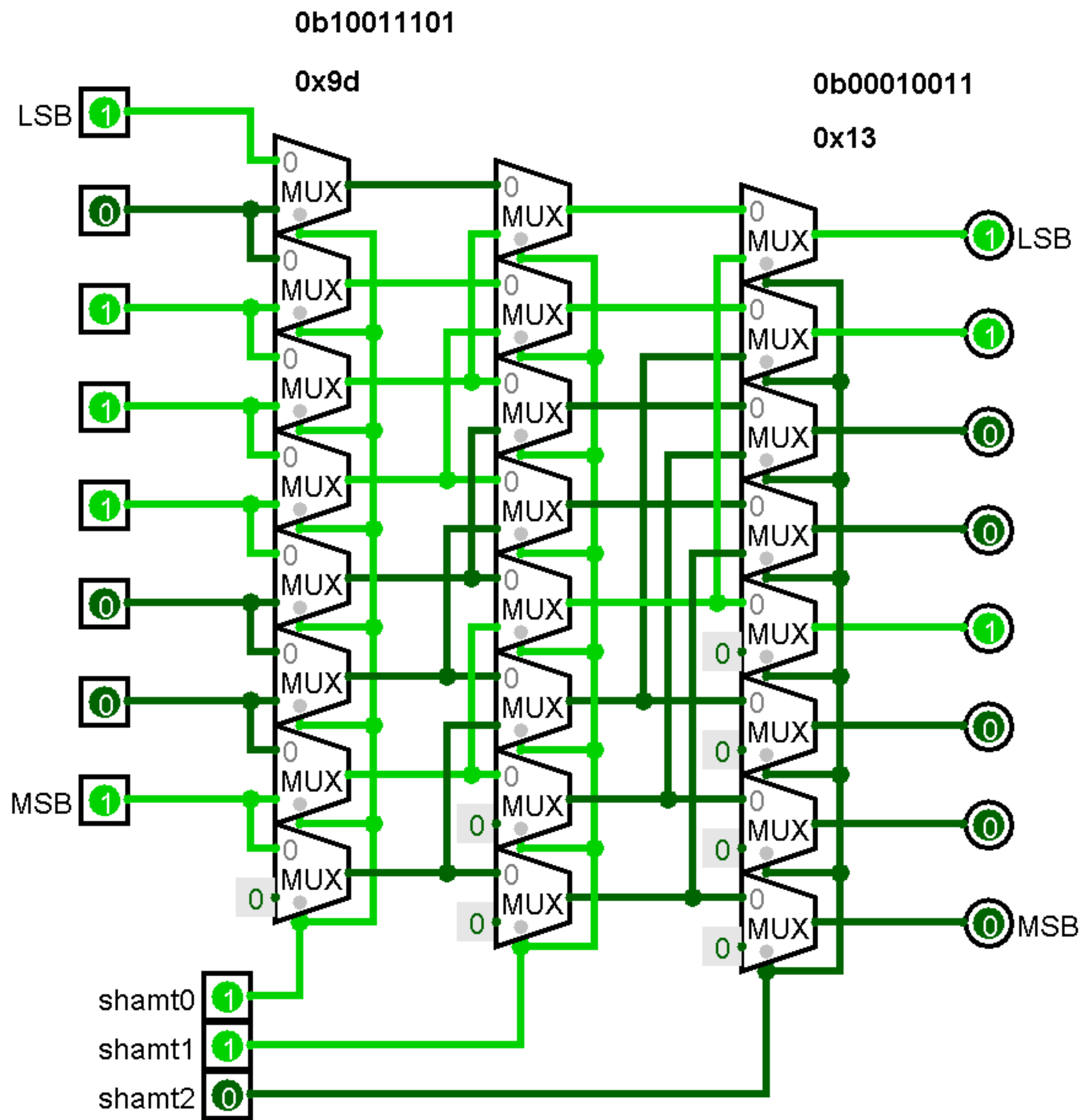
Условни и безусловни преходи

Архитект.	Забележка	Какво се прави при проверка на условието за преход и [евентуално] след каква команда за сравнение?	Дължина на отместването на относителната адресация при условните преходи	Адресации при безусловните преходи (с или без връщане)	Къде се запазва адресът на връщане?
IA-32		Проверяват се флагове <i>C, O, P, S, Z</i> след евентуално сравнение с обща команда за числа с и без знак (<i>cmp</i>)	8, 16 или 32 бита	Относителна, абсолютна, регистрова, косвени	В стековата памет
ARM		Като при IA-32, но с флагове <i>C, N, V, Z</i> . Всяка команда може да е условна	24 бита, но отместването се учетворява	Относителна, регистрова	В регистър
POWER	[Н]	Проверяват се флагове <i>LT, GT, EQ, SO (CR0–7)</i> след различни команди за сравнение за числа с и без знак	Като при ARM, но отместването е 14 бита	Относителна, абсолютна	В регистър
SPARC	3 [А]	Като при ARM, но условни могат да са само командите за преход	Като при ARM, но отместването е 22 бита	Относителна, базово-индексна, базова с отместване	В регистър
MIPS	3 [П] [Н, А]	Сравняват се регистри за = или ≠, евентуално след установяване в 1 или нулиране на регистър при сравнение за < с различни команди за числа с и без знак (<i>slt/sltu</i>)	Като при ARM, но отместването е 16 бита	Относителна, регистрова, абсолютна в рамките на текущия 256-MB регион спрямо ПБ	В регистър
RISC-V		Сравняват се регистри за =, ≠, < и ≥ с различни команди за числа с и без знак. Няма нужда от втора команда	12 бита, но отместването се удвоява	Относителна, базова с отместване	В регистър

Заб.: З)адържан преход, П)ренареждане от асемблера, Н)амек, А)нулиране на следващата команда при невзет преход.

Целочислена аритметика, пренос и битови полета

Архитектура	Особености на целочислените аритметични операции (само на 32-битовите)	Специфика в преноса и в „ <i>barrel shifter</i> “	Възможности за обработка на битови полета
IA-32	Почти всички команди имат само 2 операнда, единият от които може да е в паметта. Беззнаковото умножение и деленето имат само по 1 явен операнд.	Многоразредна ротация през C. Кой команди го обновяват е твърдо зададено.	Извличане и сгъстяване на поле по дадена маска и обратно (<i>pext/pdep</i>), от <i>Haswell</i> (2013)/ <i>Excavator</i> (2015)
ARM	Десният операнд минава през „ <i>barrel shifter</i> “. Има умножения с натрупване с 4 операнда. Деленето (от „Cortex A7“ (2013)) има само 32-битово делимо; няма команди за [извличане на] остатъка.	Ротация през пренос само надясно с <i>rrx</i> ; наляво – с <i>adc</i> . Преносът след изваждане е обратен	Извличане на поле и обратно (<i>ubfx sbfx/bfi</i>), от ARM1156 (2003) нататък
POWER	Извличането на старшата дума на произведението става с отделна команда за умножение (<i>mulhw[.]/mulhww[.]</i>). Деленето има само 32-битово делимо; няма команди за [извличане на] остатъка.	Изместването не работи с CA. CA след изваждане е обратен. Няма команда за проверка на CA.	Извличане след ротация и обратно (<i>rlw[i]nm[.]/rlwimi[.]</i>). Същите команди се използват за ротация и също не работят с флаг CA.
SPARC	Старшите думи на произведението и на делимото са в регистър Y. Умножение и делене има от SPARC V8 (1990) нататък.	Изместването не работи с флаг C. Няма команди за ротация.	Няма такива възможности
MIPS	Произведението и делимото са в <i>HI:LO</i> . Умножение+делене – от MIPS 4K (2000).	Няма програмно достъпни флагове.	Извличане и обратно (<i>ext/ins</i>) от MIPS 24K core (2003)
RISC-V	Умножение и делене – като в POWER, но има команди за [извличане на] остатъка	Преносът се проверява с команда <i>slt[u]</i>	Чрез незадължително разширение „ <i>Bitmanip</i> “, още β



PEXT — Parallel Bits Extract

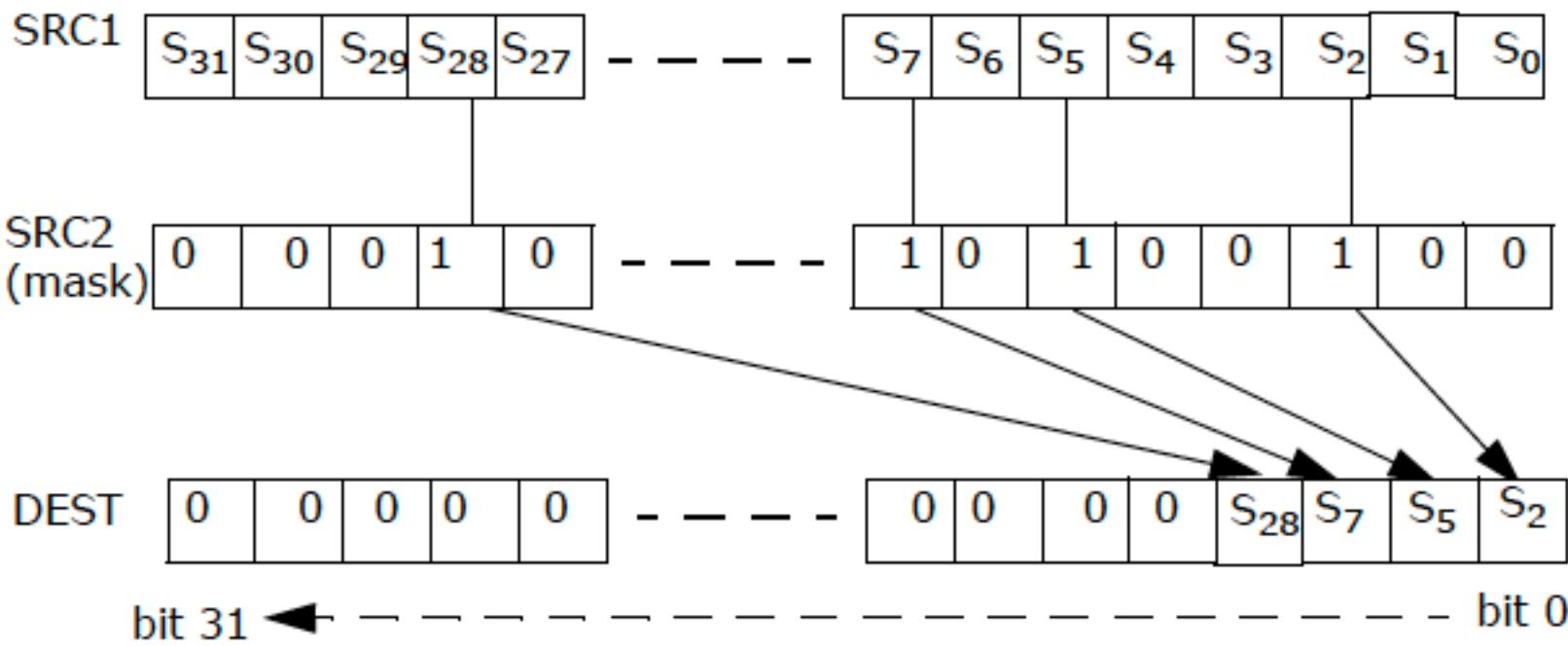
Opcode/ Instruction	Op/ En	64/32 -bit Mode	CPUID Feature Flag	Description
VEX.NDS.LZ.F3.0F38.W0 F5 /r PEXT <i>r32a, r32b, r/m32</i>	RVM	V/V	BMI2	Parallel extract of bits from <i>r32b</i> using mask in <i>r/m32</i> , result is written to <i>r32a</i> .
VEX.NDS.LZ.F3.0F38.W1 F5 /r PEXT <i>r64a, r64b, r/m64</i>	RVM	V/N.E.	BMI2	Parallel extract of bits from <i>r64b</i> using mask in <i>r/m64</i> , result is written to <i>r64a</i> .

Instruction Operand Encoding

Op/En	Operand 1	Operand 2	Operand 3	Operand 4
RVM	ModRM:reg (w)	VEX.vvvv (r)	ModRM:r/m (r)	NA

Description

PEXT uses a mask in the second source operand (the third operand) to transfer either contiguous or non-contiguous bits in the first source operand (the second operand) to contiguous low order bit positions in the destination (the first operand). For each bit set in the MASK, PEXT extracts the corresponding bits from the first source operand and writes them into contiguous lower bits of destination operand. The remaining upper bits of destination are zeroed.



PDEP – Parallel Bits Deposit

Opcode/ Instruction	Op/ En	64/32 -bit Mode	CPUID Feature Flag	Description
VEX.NDS.LZ.F2.0F38.W0 F5 /r PDEP <i>r32a, r32b, r/m32</i>	RVM	V/V	BMI2	Parallel deposit of bits from <i>r32b</i> using mask in <i>r/m32</i> , result is written to <i>r32a</i> .
VEX.NDS.LZ.F2.0F38.W1 F5 /r PDEP <i>r64a, r64b, r/m64</i>	RVM	V/N.E.	BMI2	Parallel deposit of bits from <i>r64b</i> using mask in <i>r/m64</i> , result is written to <i>r64a</i> .

Instruction Operand Encoding

Op/En	Operand 1	Operand 2	Operand 3	Operand 4
RVM	ModRM:reg (w)	VEX.vvvv (r)	ModRM:r/m (r)	NA

Description

PDEP uses a mask in the second source operand (the third operand) to transfer/scatter contiguous low order bits in the first source operand (the second operand) into the destination (the first operand). PDEP takes the low bits from the first source operand and deposit them in the destination operand at the corresponding bit locations that are set in the second source operand (mask). All other bits (bits not set in mask) in destination are set to zero.

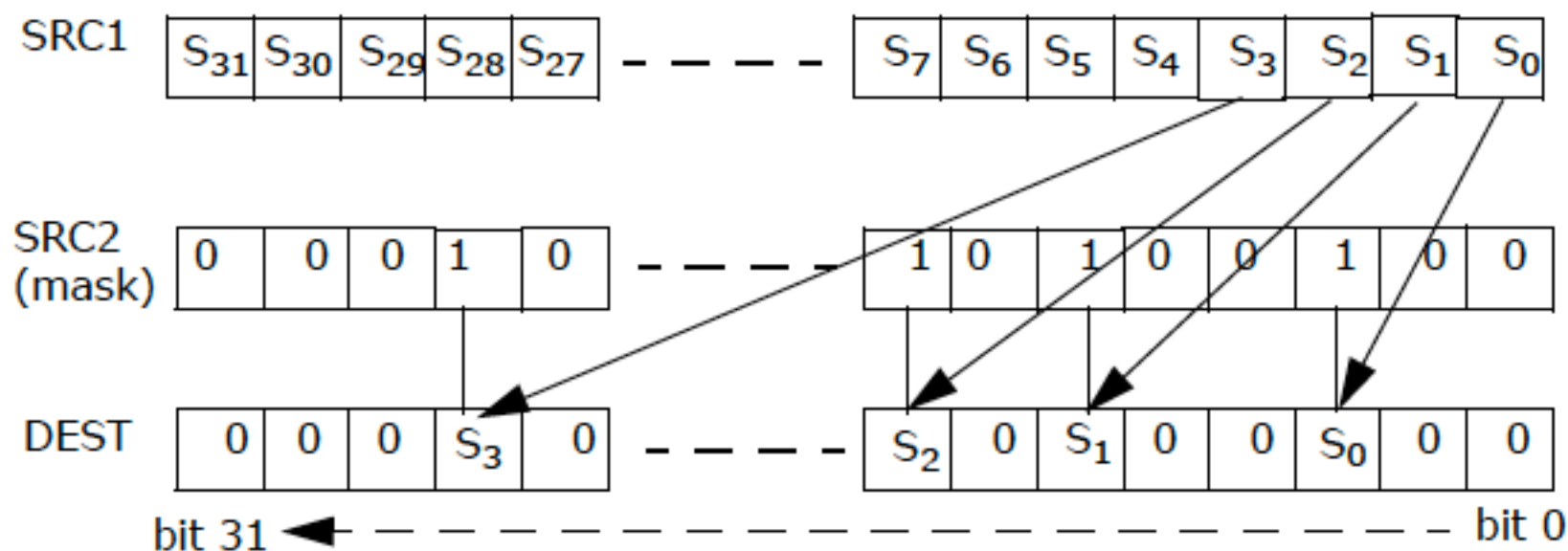


Figure 4-4. PDEP Example

Уникални особености

- **IA-32:** Командите за обработка на низове с префикс за повторение с неявна адресация, позволяващи напр. откриване на думата в EAX в масив с начален адрес в EDI и брой елементи в ECX с *„repne scasd“*.
- **ARM:** Всяка команда може да бъде условна.
- **POWER:** Осемте записа CR0–7, които могат да съхраняват отделни условия във всеки от четирите флага LT, GT, EQ, SO с команди за копиране и 8 булеви операции за сложни условия за проверка; преносът и временното препълване са отделни.
- **SPARC:** Осем регистрови прозореца, сменяни със *„save/restore“*.
- **MIPS:** Има само 2 команди за проверка на неравенство – *„slt“* и *„sltu“*, за проверка за по-малко съответно при числа с и без знак и установяване на регистър с резултата, като с размяна на регистровите операнди и инверсия на условието за проверка на този регистър се постигат всичките 8 възможни проверки за неравенство; същите се използват и за проверка на преноса.
- **RISC-V:** Макар че POWER, SPARC и MIPS ги откриха, RISC-V е първата изначално открита архитектура на система от команди.

Край

Fim

エンド

Конец

Fin

完

Amaiera

Final

نهاية

Avsluta

Fund

अंत

Dulo

Pää

끝

Einde

Slut

ଏଠି

Ende

Son

শেষ

Konec

The End

Kraj

Vége