Нижегородский государственный университет им. Н.И.Лобачевского

Факультет Вычислительной математики и кибернетики

Системные основы параллельных вычислений Архитектуры: аспекты параллелизма

Линёв А.В.

2013

Нижний Новгород

Содержание

- Введение
- Архитектура и микроархитектура CPU
- Оценка производительности CPU
- Конвейер команд. Параллелизм уровня инструкций (ILP)
- Оперативная память и кеш

Что должен знать разработчик HPC-приложений…

- Программа, с которой обычно начинается изучение новой технологии параллельного программирования

```
double A[N], B[N], C;
int i;
C = 0.0;
for(i = 0; i < N; i ++){
  C += A[i] * B[i];
}
```

- Знание системных основ параллельных вычислений позволяет понимать, как должна/будет работать такая программа на аппаратном уровне

Что должен знать разработчик HPC-приложений

- Предварительные требования
  - Основы программирования
  - Алгоритмы и анализ сложности
  - Языки программирования
- Системные основы параллельных вычислений
  - Архитектура вычислительных систем (+ Ассемблер)
  - Компьютерные сети
  - Операционные системы
  - Компиляторы
- Использование параллельных вычислений
  - Параллельное программирование / Алгоритмы / Языки / Технологии / Инструменты / …

Архитектура вычислительных систем…

- Введение
- Архитектура процессора, компоненты CPU
- Конвейеризация вычислений (статическое и динамическое планирование)
- Векторные вычисления
- Иерархия памяти
- Классификация архитектур вычислительных систем
  - Симметричное мультипроцессирование
  - Массивно-параллельные системы, кластерные системы
  - Параллелизм в процессорах специального назначения
- Примеры вычислительных систем

Архитектура вычислительных систем

- John L. Hennessy and David A. Patterson. Computer Architecture: A Quantitative Approach. 4th Ed., Morgan Kaufmann, 2007
- Брайант Р., О'Халларон Д. Компьютерные системы: архитектура и программирование, СПб: БХВ-Петербург, 2005
- Столлингс Уильям. Структурная организация и архитектура компьютерных систем. М: Вильямс, 2002
- Эхтер Ш., Робертс Дж. Многоядерное программирование. СПб: Питер, 2010
- Аблязов Р.З. Программирование на ассемблере на платформе x86-64. М: ДМК Пресс, 2011
- Таненбаум Э. Архитектура компьютера. 5 изд. СПб:Питер, 2007
- Касперски К. Техника оптимизации программ. Эффективное использование памяти. СПб: БХВ-Петербург, 2003
- Muhammad Shaaban. CMPE 550
- Intel® 64 and IA-32 Architectures Optimization Reference Manual
  - http://www.intel.com/content/dam/doc/manual/64-ia-32-architectures-optimization-manual.pdf/

Уровни параллелизма

Микроуровневый параллелизм

Параллелизм **уровня команд**

Параллелизм **уровня потоков**

Параллелизм **уровня заданий**
- Мультипроцессорные системы
- Мультикомпьютерные системы

Мелкозернистый (fine grained)
обеспечивает компилятор

Среднезернистый (medium grained)
обеспечивают программист, компилятор

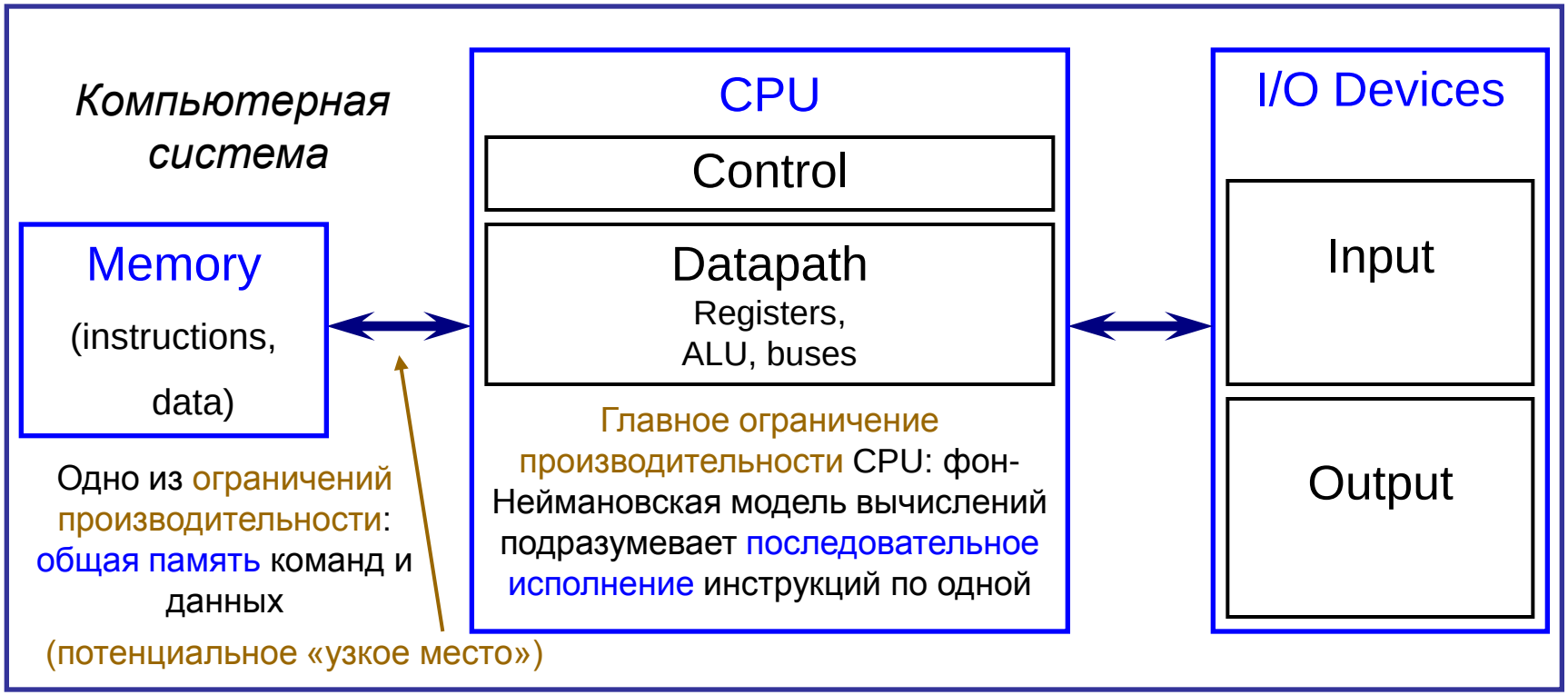
Крупнозернистый (coarse grained)
обеспечивает OS

Архитектура и микроархитектура CPU

Фон Неймановская модель компьютера

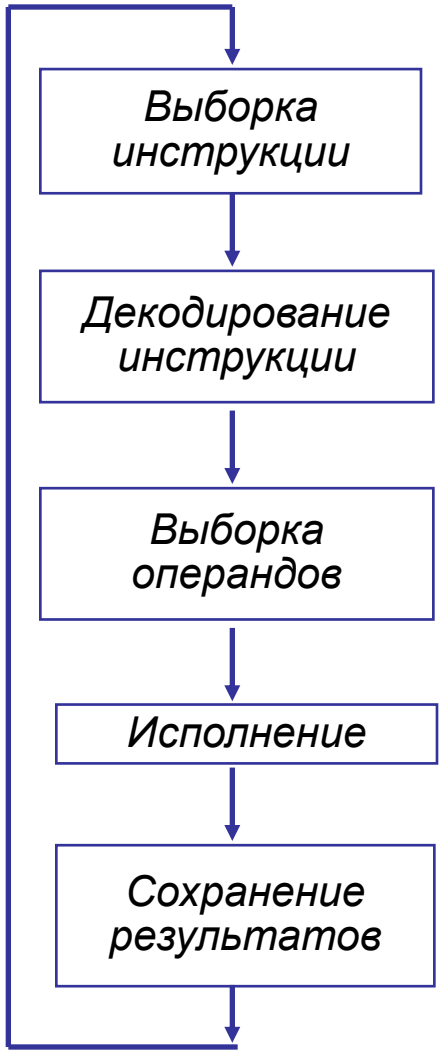
- Разделение программируемой вычислительной машины на компоненты:
  - Центральный обрабатывающий блок (Central Processing Unit, CPU): ,блок управления (Control Unit) (декодирование инструкций, порядок операций), тракт данных (Datapath) (регистры, арифметико-логическое устройство, шины).
  - Память: Хранение инструкций и их операндов.
  - Подсистема ввода/вывода (Input/Output, I/O sub-system): шина I/O, интерфейсы, устройства.
- Концепция хранения программ: Инструкции из набора команд выбираются из общей памяти и исполняются последовательно.

Фон Неймановская модель компьютера



Процессор - программируемый вычислительный элемент, выполняющий программы, написанные с использованием предопределенного набора инструкций.

Шаги обработки инструкций в CPU



Выбрать инструкцию программы из памяти

Программный счетчик (Program Counter, PC / Instruction Pointer, IP) указывает на следующую для обработки инструкцию

Определить требуемые действия и размер инструкции

Найти и получить данные операндов

Вычислить значение результата или статус

Записать результаты в запоминающее устройство для последующего использования

Главное ограничение производительности CPU: фон-Неймановская модель вычислений подразумевает последовательное исполнение инструкций по одной

Микроархитектура CPU

- Дизайн тракта данных:

Компоненты & их соединение, требуемые инструкциями ISA

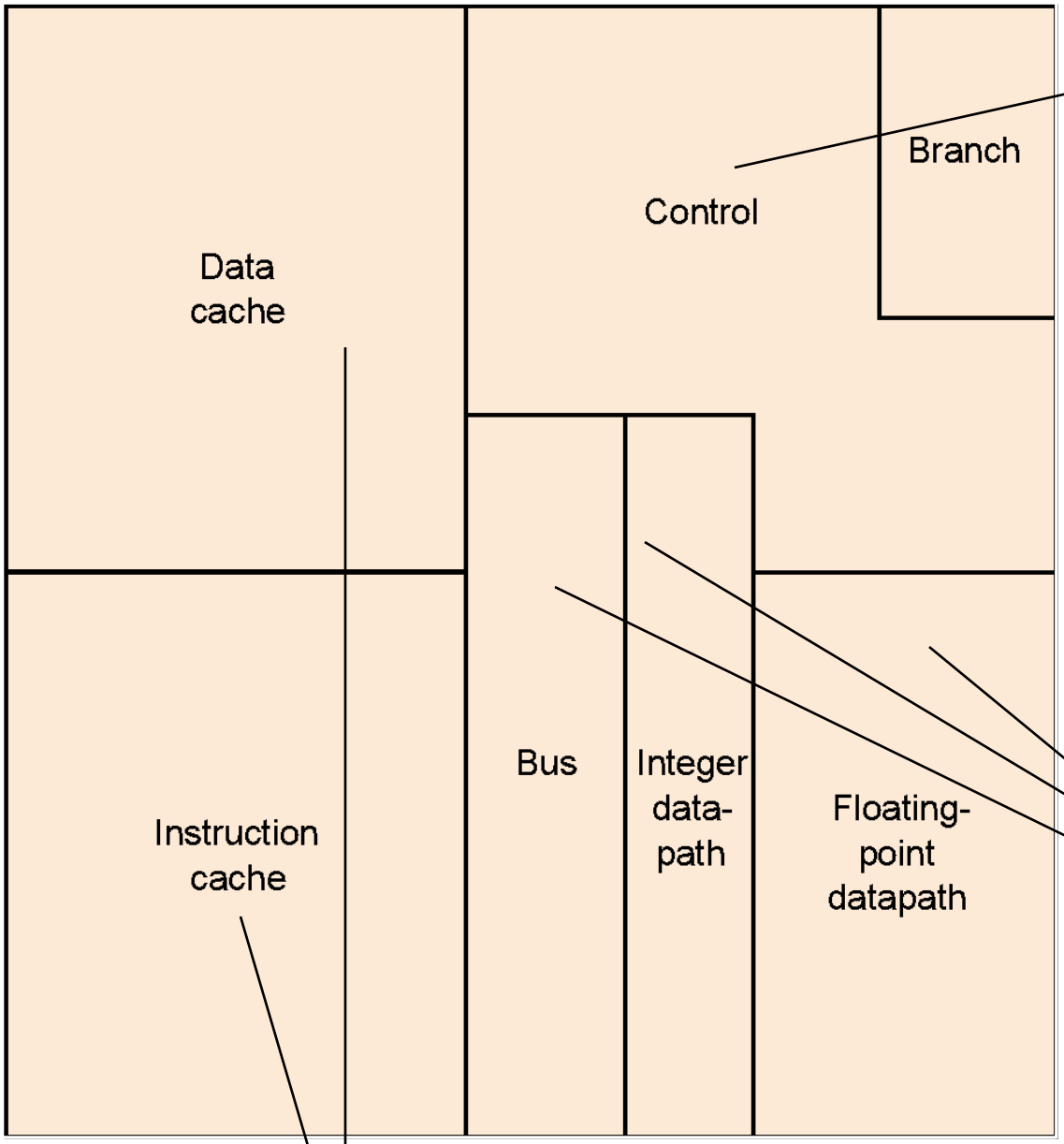
  - –Возможности и характеристики производительности главных функциональных блоков (Functional Units, FUs):
    - •(например: регистры, ALU, схемы сдвига, блоки логики, ...)
  - –Способы соединения этих компонентов (соединения через шины, мультиплексоры, и т.д.).
  - –Как идет поток информации между компонентами.

- Дизайн блока управления:

Управление/упорядочивание операций компонентов тракта данных для реализации инструкций ISA

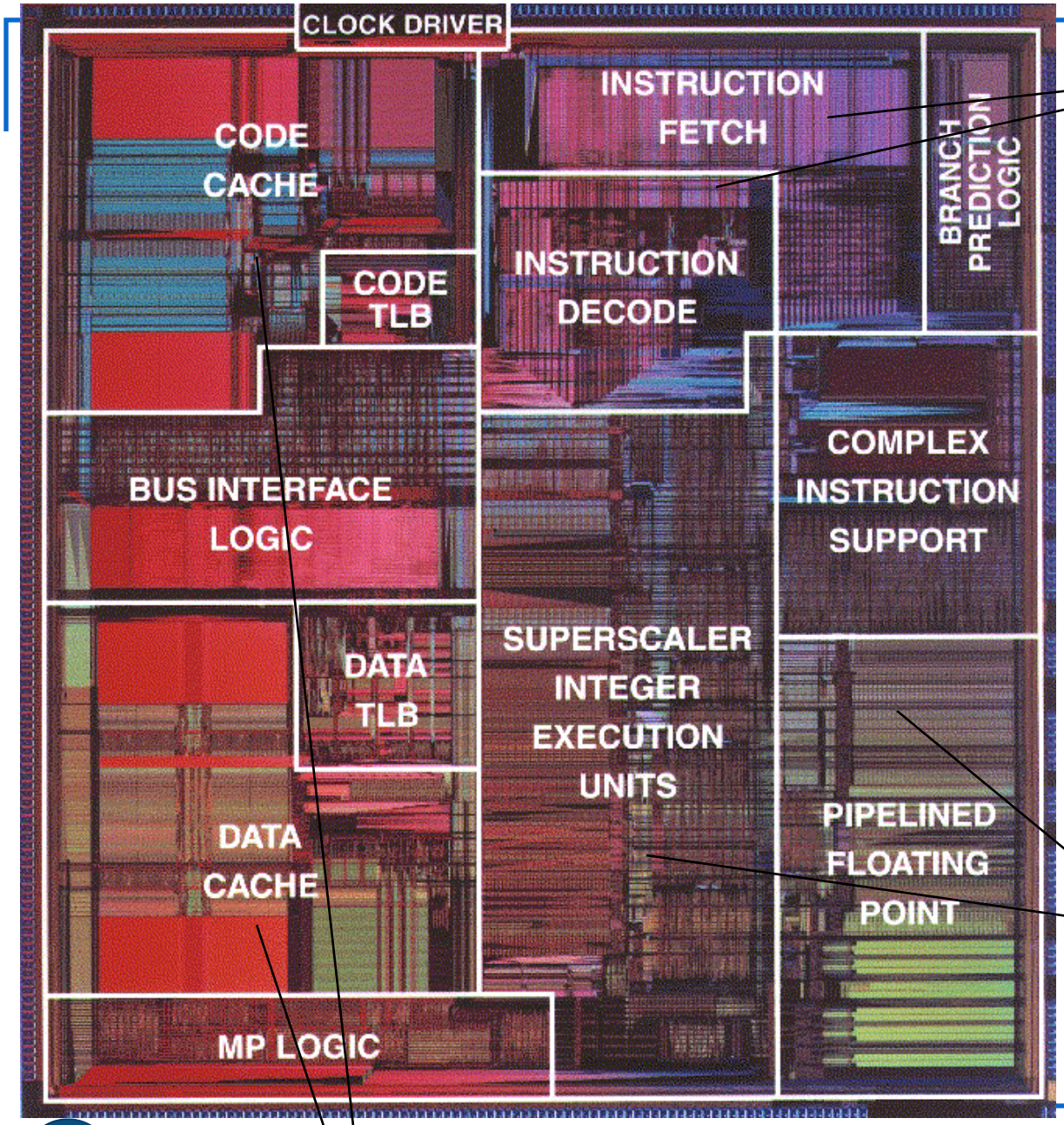
  - –Логика и средства управления потоком информации.
  - –Управление и координация операций FU для реализации требуемой архитектуры набора команд (может быть выполнено конечными автоматами или микропрограммой).

Описание аппаратных операций на подходящем языке возможно, используя язык межрегистрового обмена (Register Transfer Notation, RTN).



A Typical Microprocessor Layout: The Intel Pentium Classic

1993 - 1997
60MHz - 233 MHz



Control Unit

A Typical Microprocessor Layout: The Intel Pentium Classic

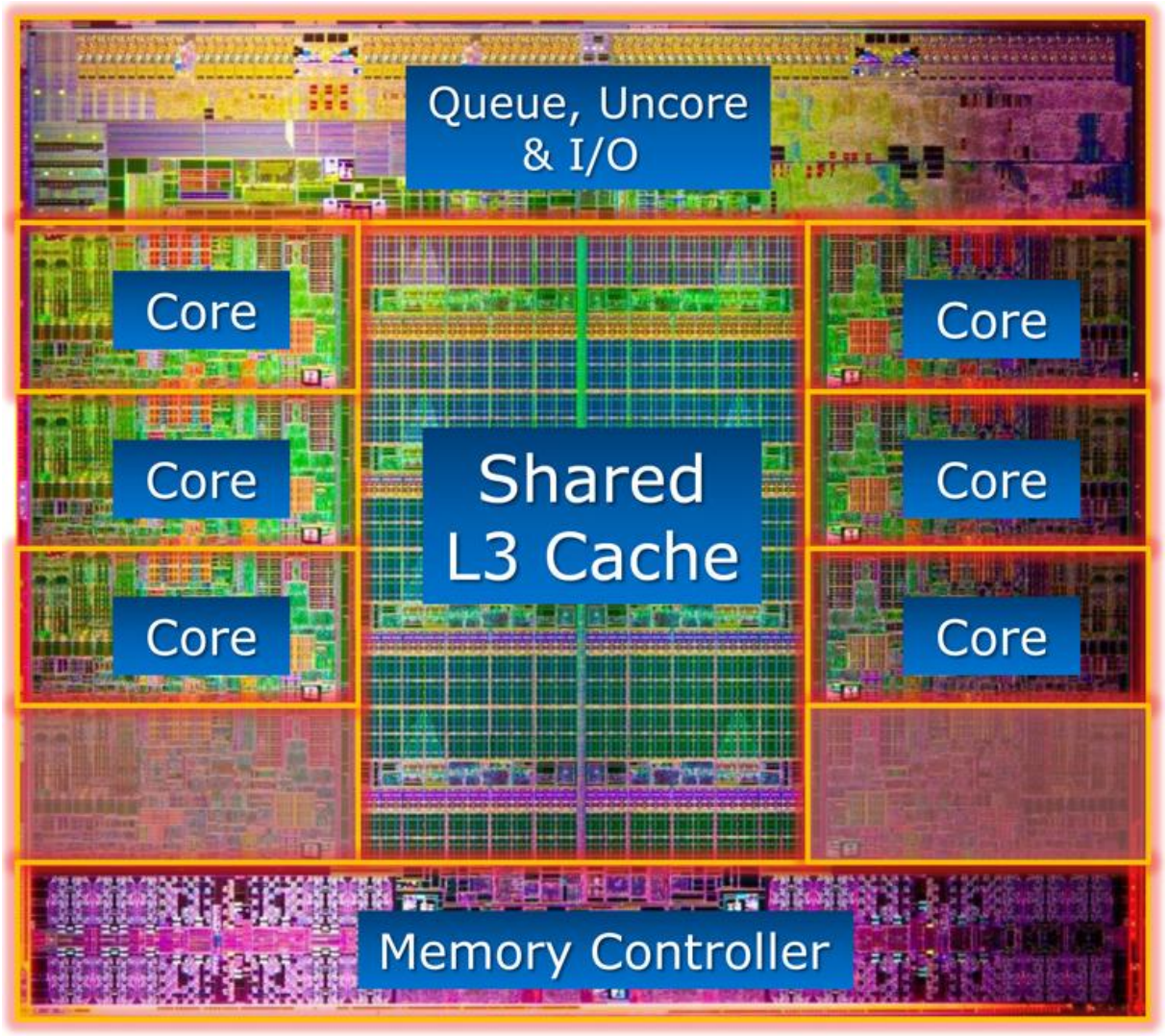
1993 - 1997

60MHz - 233 MHz

Datapath

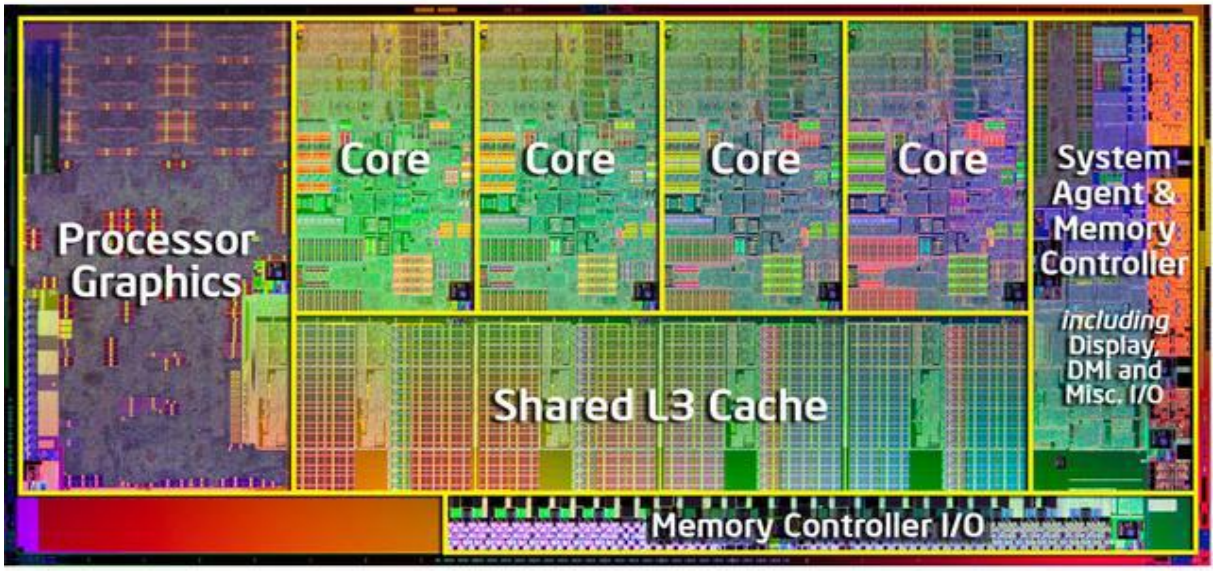
First Level of Memory (Cache)

Multi-core Microprocessor Example

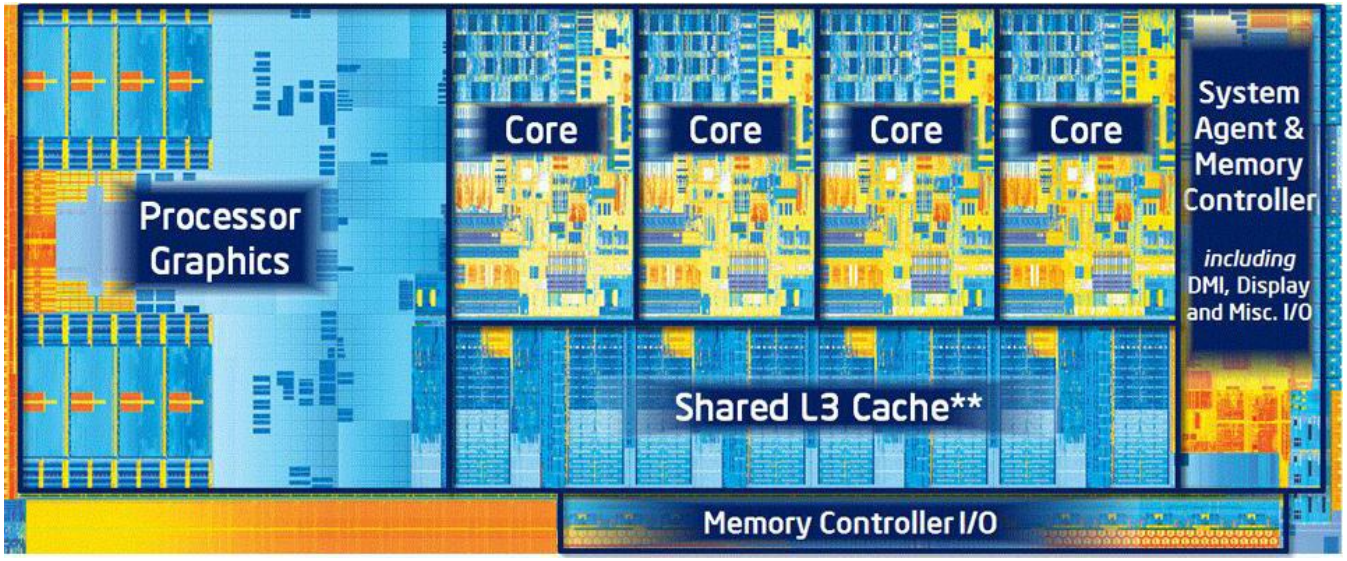


Intel Core i7-3960X

- Sandy bridge

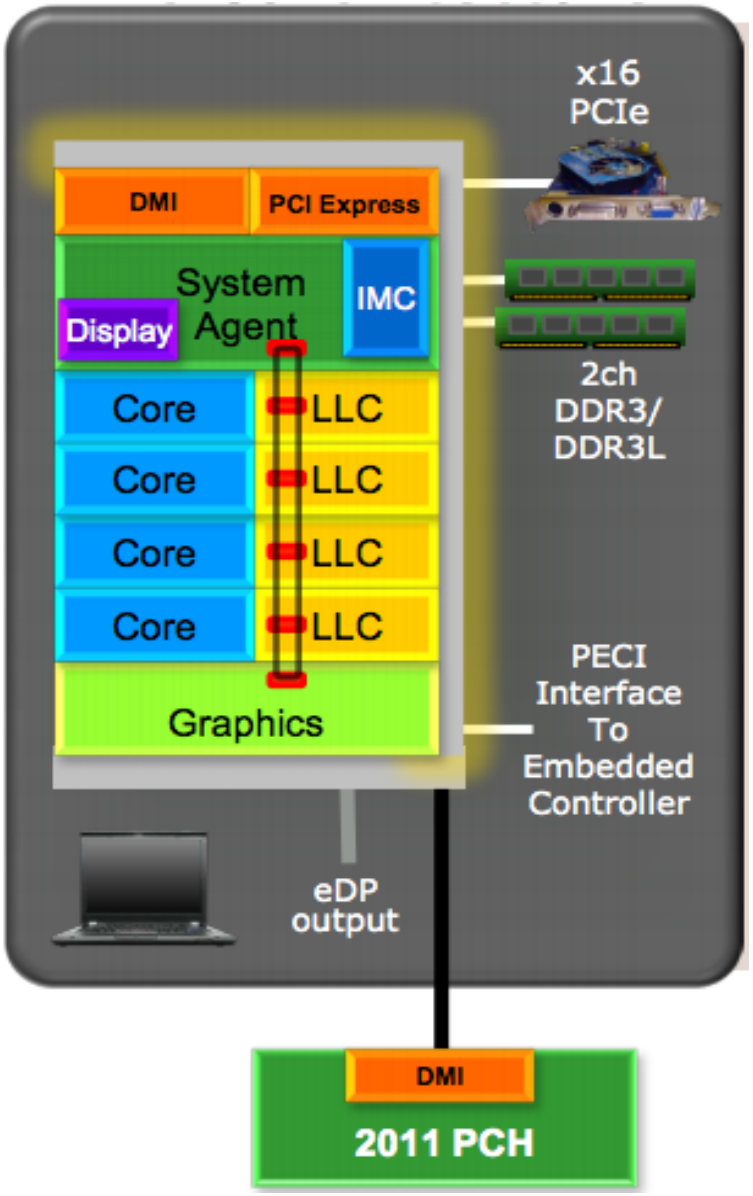


- Ivy bridge



Ivy Bridge

- Ring Interconnect
  - Connects all key components:
    - SA, LLCs, Cores, GPU
  - Actually 4 rings:
    - 32-byte data ring
      - Two packets for e.g. cache data
    - Request ring
    - Acknowledge ring
    - Snoop ring

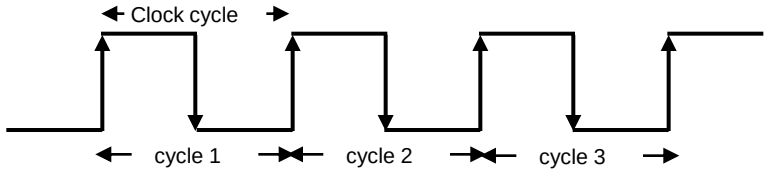


Оценка производительности CPU

Оценка производительности CPU

Большинство блоков процессора работают синхронно, используя тактовый генератор CPU, работающий с постоянной частотой (Clock rate) :

где: частота = 1 / такт (Clock cycle)



Частота CPU зависит от микроархитектуры специфического CPU и используемой технологии производства.

Машинная инструкция (ISA) состоит из набора элементарных инструкций или микроопераций, которые различаются по сложности и количеству в зависимости от инструкции и организации (дизайна) конкретного CPU.

- Микрооперация – это элементарное аппаратное действие, которое может быть произведено за 1 такт CPU.
- Она соответствует одной микроинструкции в микропрограммируемых CPU.

Одна машинная инструкция может потребовать один или больше тактов CPU, что обозначается как CPI (Cycles Per Instruction).

Среднее CPI программы: Это CPI усредненное для всех инструкций, выполненных в программе на CPU с заданным дизайном.

Число инструкций за такт (Instructions Per Cycle, IPC) = 1/CPI

Уравнение производительности CPU

Программа состоит из набора исполняемых инструкций, I

- Измеряется в: инструкции/программу

Исполнение инструкции в среднем требует некое число *тактов на инструкцию (cycles per instruction, CPI)*

- Единицы измерения: такты/инструкцию, CPI
- или Instructions Per Cycle, (IPC): IPC= 1/CPI

CPU имеет фиксированное время такта

C = 1/тактовую частоту

- Единицы измерения: секунды/такт

Время выполнения программы CPU есть произведение указанных трех параметров:

исполняемых

$$Время\,CPU = \frac{секунд}{на\,программу} = \frac{инструкций}{в\,программе} \times \frac{тактов}{инструкцию} \times \frac{секунд}{такт}$$

$$T = I \times CPI \times C$$

Факторы, влияющие на производительность CPU

	Число инструкций I	CPI	Время такта C
Программа	X	X	
Компилятор	X	X	
Instruction Set Architecture (ISA)	X	X	
Организация (дизайн CPU)		X	X
Технология (СБИС)			X

$$T = I \times CPI \times C$$

Типы инструкций & CPI

Пусть имеется программа с *n* типами инструкций, исполняемых на заданном CPU со следующими характеристиками:

C_i = число инструкций типа i (исполняемых)

CPI_i = тактов на инструкцию типа i

F_i = частота/доля исполнения инструкций типа i = C_i/I

Тогда:

CPI = CPU такты/ число инструкций I

Где:

$$CPU_{такты} = \sum_{i=1}^{n} (CPI_i \times C_i)$$

Число инструкций I = ΣC_i

Или

$$CPI = \sum_{i=1}^{n} (CPI_i \times F_i)$$

Частота типов инструкций & CPI

Профиль программы или соотношение исполняемых инструкций

$$\frac{CPI_i \times F_i}{CPI}$$

Base Machine (Reg / Reg)

Оп	Частота,F_i	CPI_i	$CPI_i \times F_i$	% Времени
ALU	50%	1	.5	23% = .5/2.2
Load	20%	5	1.0	45% = 1/2.2
Store	10%	3	.3	14% = .3/2.2
Branch	20%	2	.4	18% = .4/2.2
			Итого = 2.2	

Типичное соотношение

$$CPI = \sum_{i=1}^{n} (CPI_i \times F_i)$$

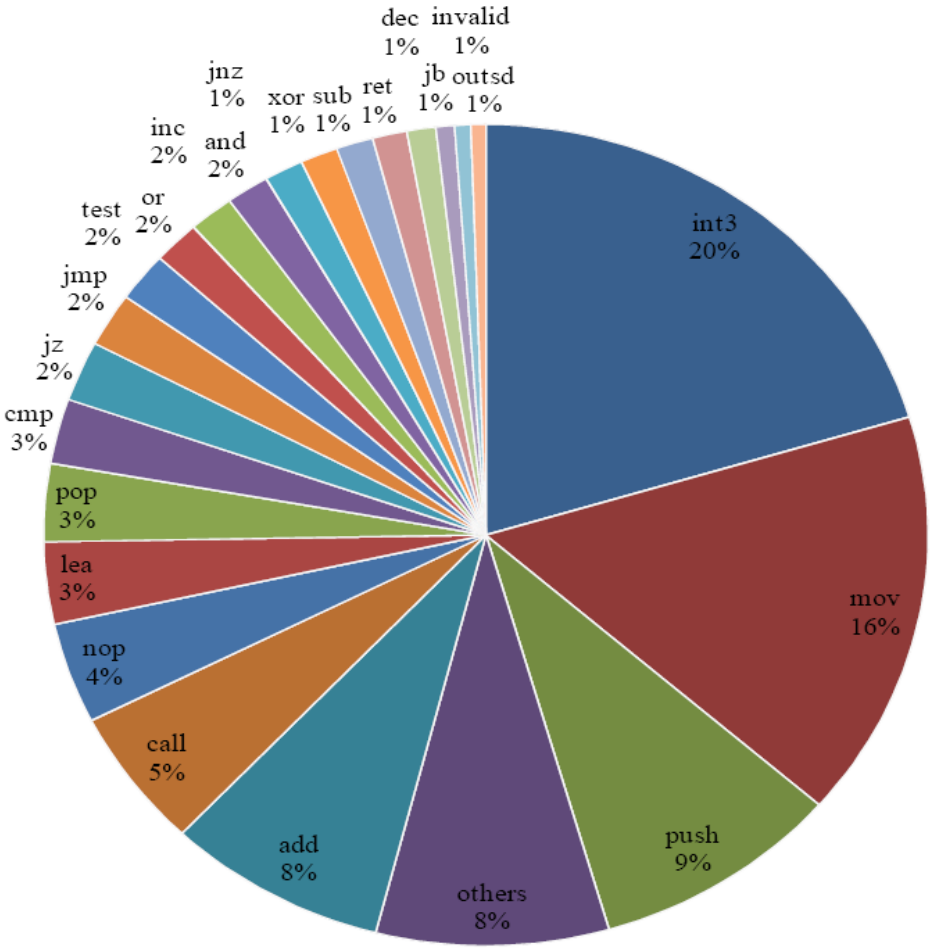
CPI = .5 x 1 + .2 x 5 + .1 x 3 + .2 x 2 = 2.2

= .5 + 1 + .3 + .4

Топ 10 инструкций Intel X86

Ранг	инструкция	процент от всех исполняющихся в среднем
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%
	Всего	96%

Топ инструкций Intel x86-64



Amr Hussam Ibrahim, Mohamed Bakr Abdelhalim, Hanadi Hussein, & Ahmed Fahmy
“An Analysis of x86-64 Instruction Set for Optimization of System Softwares”
International Journal of Advanced Computer Science, Vol. 1, No. 4, Pp. 152-162, Oct. 2011.

Конвейер команд

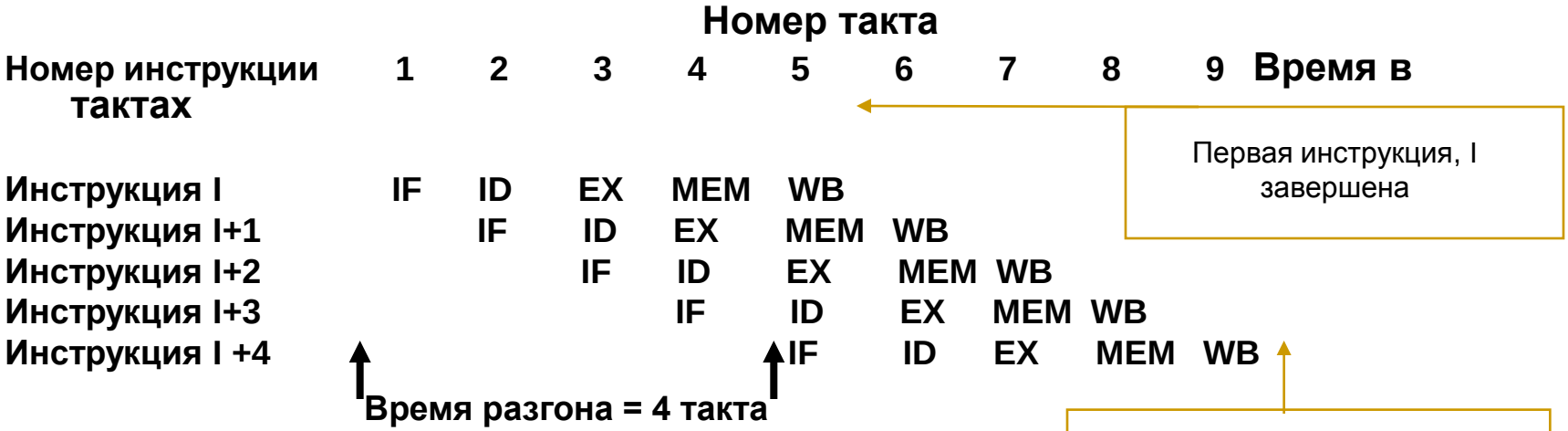
Параллелизм уровня инструкций (ILP)

Конвейер команд…

- Конвейерная обработка инструкций – это метод реализации CPU, при котором множество операции над несколькими инструкциями перекрываются.
  - Конвейерная обработка инструкций использует программный параллелизм уровня инструкций (Instruction-Level Parallelism, ILP)
- Конвейеризация увеличивает пропускную способность CPU - среднее число инструкций, завершенных за такт.
  - В идеальном случае происходит завершение одной инструкции за машинный такт
- Конвейеризация не сокращает время выполнения отдельной инструкции (также называемое временем задержки завершения инструкции).
  - Минимальное время задержки завершения инструкции - n тактов, где n – число ступеней конвейера
- Конвейер, описанный здесь, называется **упорядоченным** (**in-order**) **конвейером** так как инструкции обрабатываются или исполняются в порядке, указанном в исходной программе

Однопортовый конвейер MIPS с упорядоченной обработкой целочисленных операций

Число тактов до заполнения = время разгона = число ступеней -1



Ступени конвейера MIPS:

IF = Выборка инструкции (Instruction Fetch)

ID = Декодирование инструкции (Instruction Decode)

EX = Исполнение (Execution)

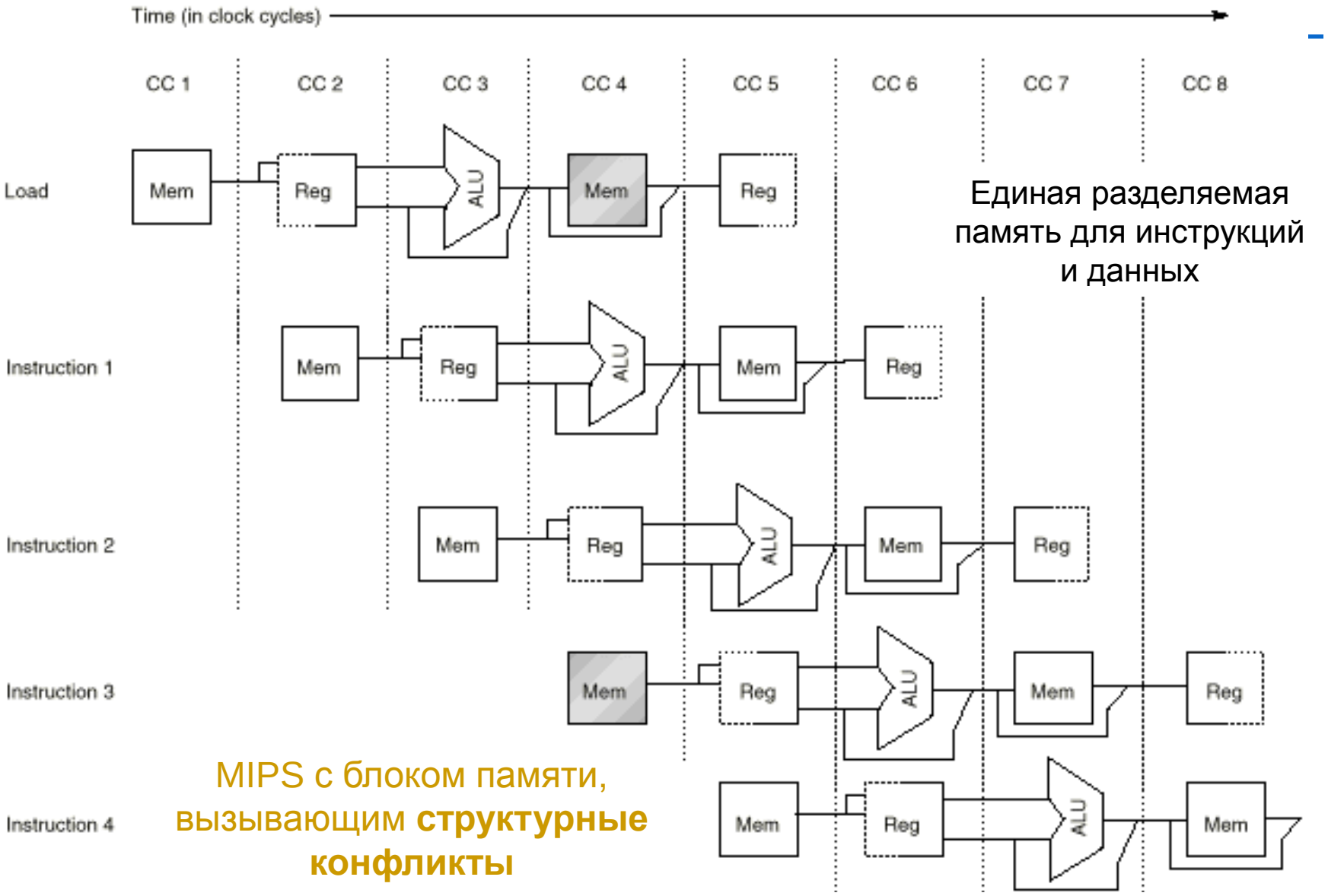
MEM = Обращение к памяти (Memory Access)

WB = Запись результата (Write Back)

Pentium 4 – 31
Core 2 – 14
Sandy Bridge – 18
Ivy Bridge – 14

Конвейер команд

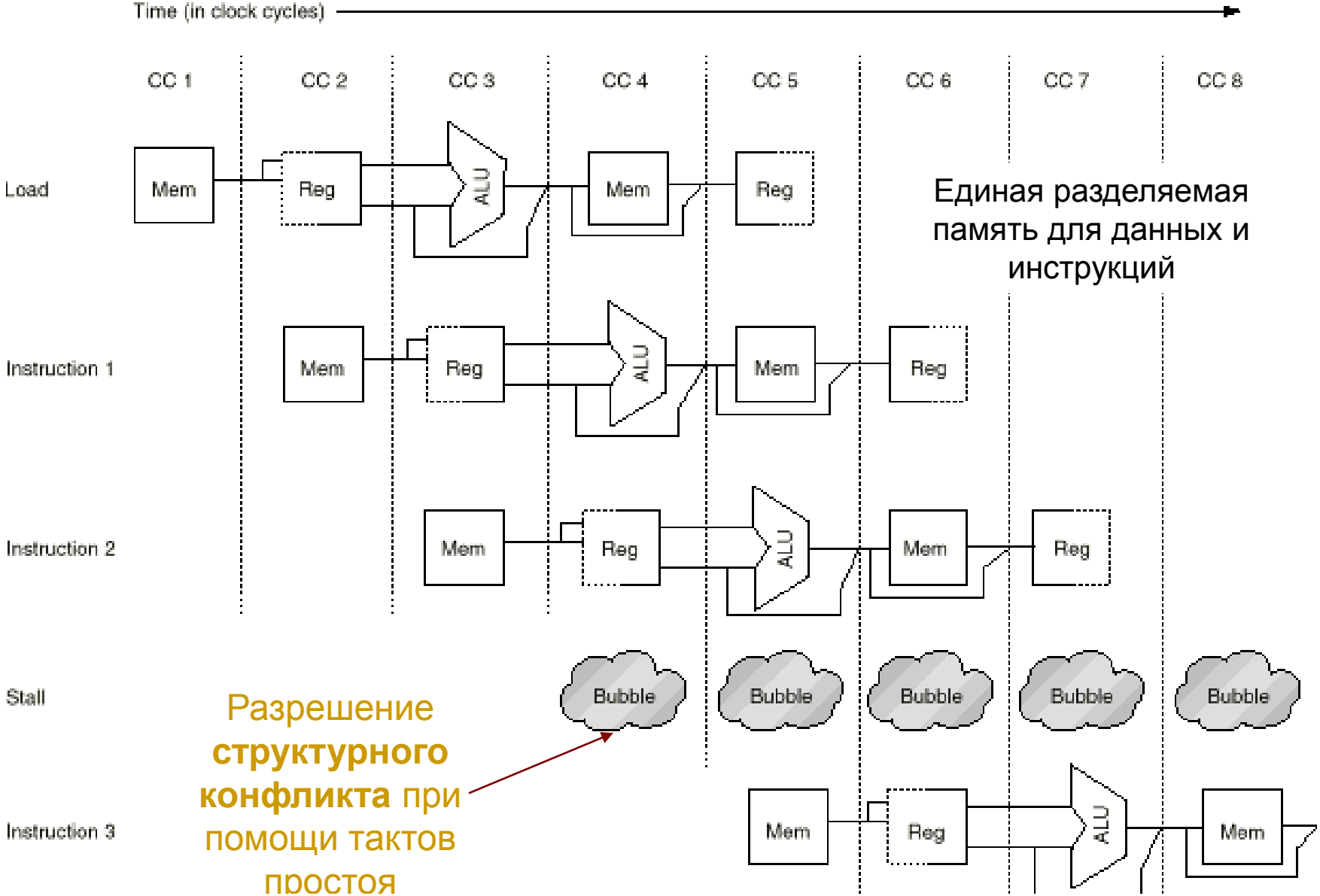
- Структурные конфликты
  - Возникают из-за недостатков аппаратных ресурсов когда доступное аппаратное обеспечение не в состоянии поддерживать все возможные комбинации инструкций
- Конфликты данных
  - Возникают когда инструкция зависит от результата выполнения предыдущей инструкции так, что это проявляется при перекрытии инструкций в конвейере
- Конфликты управления
  - Возникают при конвейеризации условных переходов и других инструкций, которые изменяют PC



Единая разделяемая память для инструкций и данных

MIPS с блоком памяти, вызывающим **структурные конфликты**

В машине с единственным портом памяти будет возникать конфликт при любом обращении к памяти.

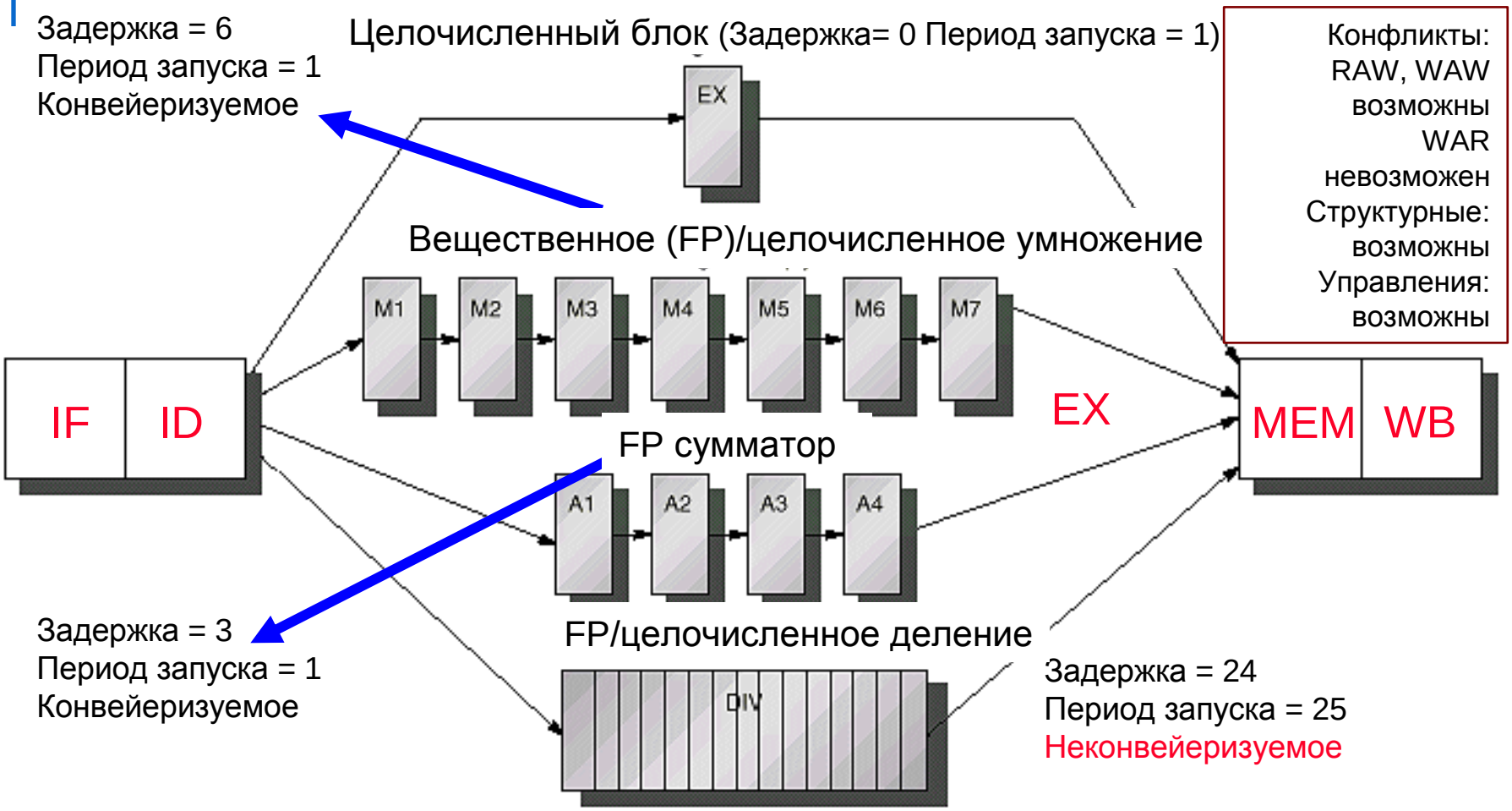


Единая разделяемая память для данных и инструкций

Разрешение **структурного конфликта** при помощи тактов простоя

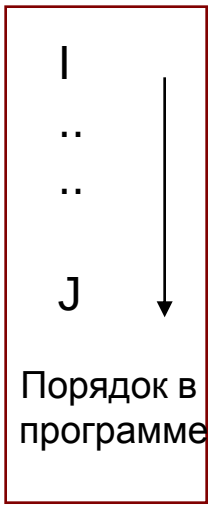
Структурный конфликт приводит к необходимости вставки «пузырей» в конвейер.

Многотактовый конвейер вещественных операций



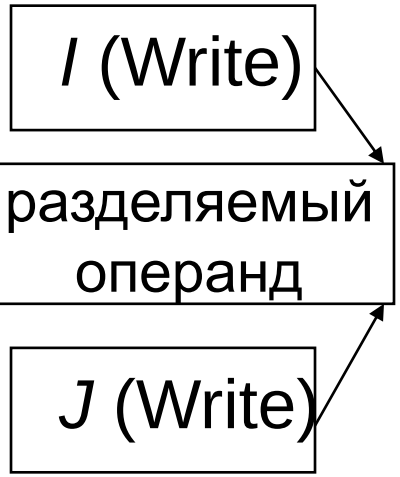
Однопортовый конвейер MIPS с упорядоченной обработкой и поддержкой FP
Супер-конвейерный CPU: Конвейерный CPU с конвейеризуемыми FP блоками

DisplayFamily_DisplayModel	Latency 06_3AH	Latency 06_2AH, 06_2DH	Throughput 06_3AH	Throughput 06_2AH, 06_2DH
VMOVDDUP ymm1, ymm2		1		1
VMULPD/PS ymm1, ymm2, ymm3		5		1
VSUBPD/PS ymm1, ymm2, imm		3		1
VDIVPD ymm1, ymm2, ymm3	35	45	28	44
VDIVPS ymm1, ymm2, ymm3	21	29	14	28
VSQRTPD ymm1, ymm2	35	45	28	44
VMULPD/PS ymm1, ymm2, ymm3		5		1
VRSQRTPS ymm1, ymm2		7		1
FSQRT EP		43		X87 FPU
F2XM1		90-150		X87 FPU
FCOS		190-240		X87 FPU
FPATAN		150-300		X87 FPU

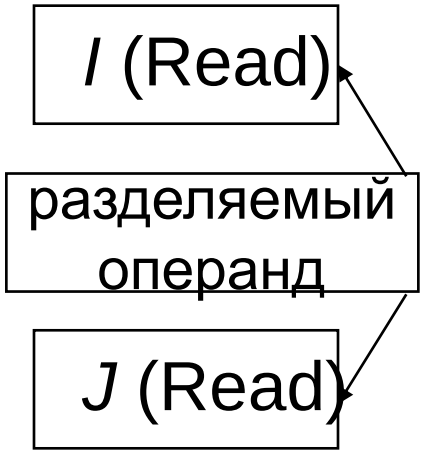


Read after Write (RAW)

Write after Read (WAR)



Write after Write (WAW)

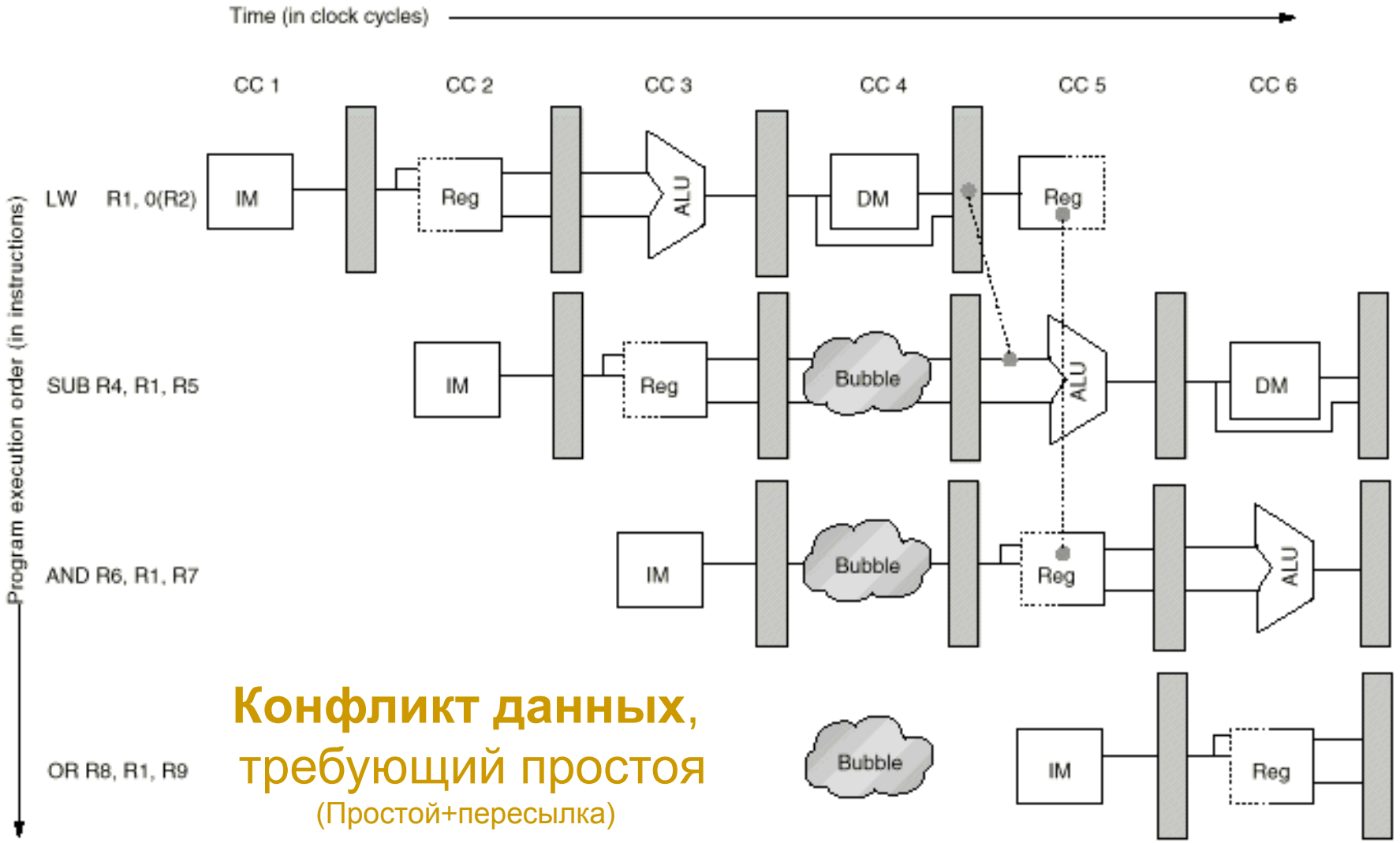


Read after Read (RAR)

нет конфликта

Пример **конфликта данных**

Использование результата инструкции DADD в трех следующих инструкциях вызывает конфликт, так как пока записи в регистр не произойдет, до тех пор эти инструкции не могут его прочитать.



Конфликт данных,
требующий простоя
(Простой+пересылка)

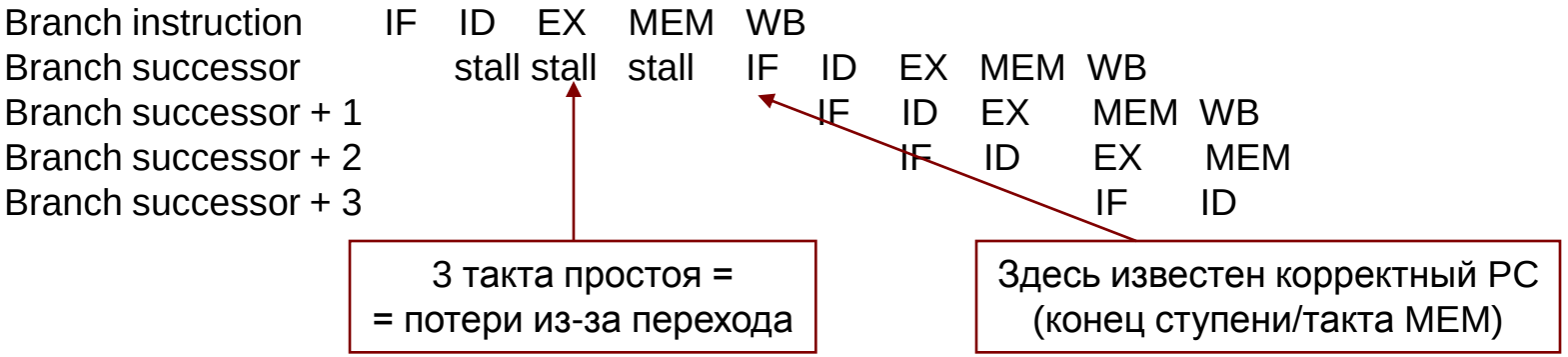
Блокировка при загрузке приводит к необходимости вставки «пузыря» на такте 4, задерживая инструкцию SUB и следующие за ней на 1 такт

Конфликты управления

При выполнении условного перехода, может измениться PC что, если не принять специальных мер, ведет к остановке конвейера на много тактов, пока не будет вычислено условие перехода (переход определится).

- Иначе PC может быть неверным, когда потребуется на ступени IF.

В рассматриваемом конвейере MIPS, условный переход определится на ступени 4 (MEM), приводя к трем тактам простоя, как показано ниже:



Предположим, что мы останавливаем или сбрасываем конвейер при инструкции перехода, тогда в рассматриваемом конвейере MIPS, тратится впустую три такта для каждого перехода.

Потери из-за перехода = номер ступени, когда переход определится - 1

здесь потери = 4 - 1 = 3 такта

Конвейерная обработка данных и использование параллелизма уровня инструкций(ILP)

- Параллелизм уровня инструкций (ILP) существует, когда инструкции в последовательности независимы и поэтому могут исполняться в конвейере параллельно (с перекрытием).
- Конвейерная обработка повышает производительность именно за счет перекрытия при исполнении независимых инструкций и таким образом использует ILP кода.
- Программы, обладающие большим ILP (меньше зависимостей) обычно показывают лучшую производительность на CPU с конвейером

Стандартные подходы и механизмы

- Статическое планирование (компилятор)
  - (Очень) длинное командное слово (V)LIW
- Динамическое планирование (CPU)
  - Суперскалярность
- Статическое предсказание переходов компилятором
- Динамическое предсказание переходов в CPU и спекулятивное выполнение
- SMT (Simultaneous Multi-Threading), Hyper-Threading Technology

Пример статического планирования инструкций компилятором

Для последовательности кода:

a = b + c

d = e - f

a, b, c, d ,e, и f находятся в памяти

Считая, что загрузки занимают один такт, следующий код (планирование конвейера компилятором) исключает простои:

Исходный код с простоями:

```
LD    Rb,b
LD    Rc,c
DADD  Ra,Rb,Rc
SD    Ra,a
LD    Re,e
LD    Rf,f
DSUB  Rd,Re,Rf
SD    Rd,d
```

простой → (после DADD)

простой → (после LD Rf,f)

2 простоя в исходном коде

Перепланированный код без простоев:

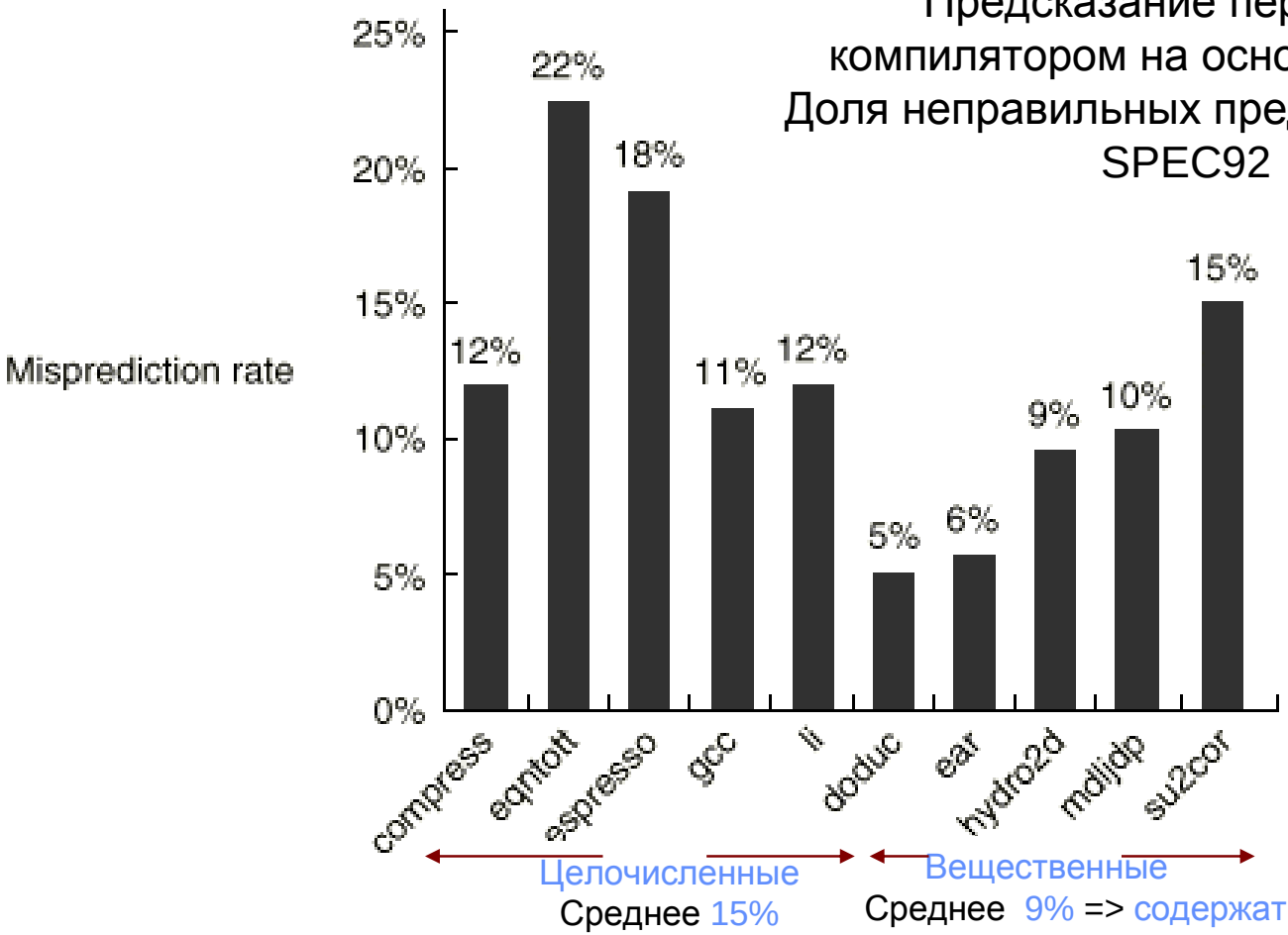
```
LD    Rb,b
LD    Rc,c
LD    Re,e
DADD  Ra,Rb,Rc
LD    Rf,f
SD    Ra,a
DSUB  Rd,Re,Rf
SD    Rd,d
```

Предполагается, что конвейер поддерживает пересылку

Статическое предсказание переходов компилятором

- Статическое предсказание переходов кодируется в инструкциях перехода используя один бит предсказания:
  0 = не происходит, 1 = происходит
  - Требует поддержки ISA, примеры реализации: HP PA-RISC, PowerPC, UltraSPARC.
- Существует два основных метода для статического предсказания переходов во время компиляции:
  - Сбор информации о поведении программы при ее запусках и использование при перекомпиляции (профилирование).
    - Например, профиль программы может показать, что большинство переходов вперед и назад (это часто вызвано циклами) происходят. Простейшая схема в данном случае - всегда предсказывать, что переход происходит.
- Эвристическое предсказание переходов на основе направления перехода, помечая переходы назад как происходящие и переходы вперед как не происходящие

Предсказание переходов компилятором на основе профиля Доля неправильных предсказаний для SPEC92



Доля неправильных для предсказаний, основанных на профиле, широко варьируется, но обычно лучше для программ с FP, которые имеют среднюю долю неправильных предсказаний в 9 % со стандартным отклонением в 4%, чем для программ с целочисленными операциями, которые имеют среднюю долю неправильных предсказаний в 15 % со стандартным отклонением в 5%.

Динамическое предсказание переходов в CPU

- Предположение о направлении перехода основывается на истории переходов
- Пример: двухуровневый предсказатель с глобальной историей
  - Хранит результаты для M последних использованных инструкций перехода
  - Для каждой хранятся последние N переходов
  - Sandy Bridge использует 32-битный регистр для хранения истории переходов
    - Точность предсказания >90%
- Процессор загружает на конвейер инструкции из предсказанной ветки перехода, в случае неверного предсказания результат их исполнения не используется

Минимизация ошибок предсказания переходов

- Размещать наиболее вероятные ветви в начале ветвлений
- Выносить выше (по уровню вложенности в циклах) инвариантные ветвления
- Использовать разворачивание циклов

Simultaneous Multi-Threading
Одновременная многопоточность

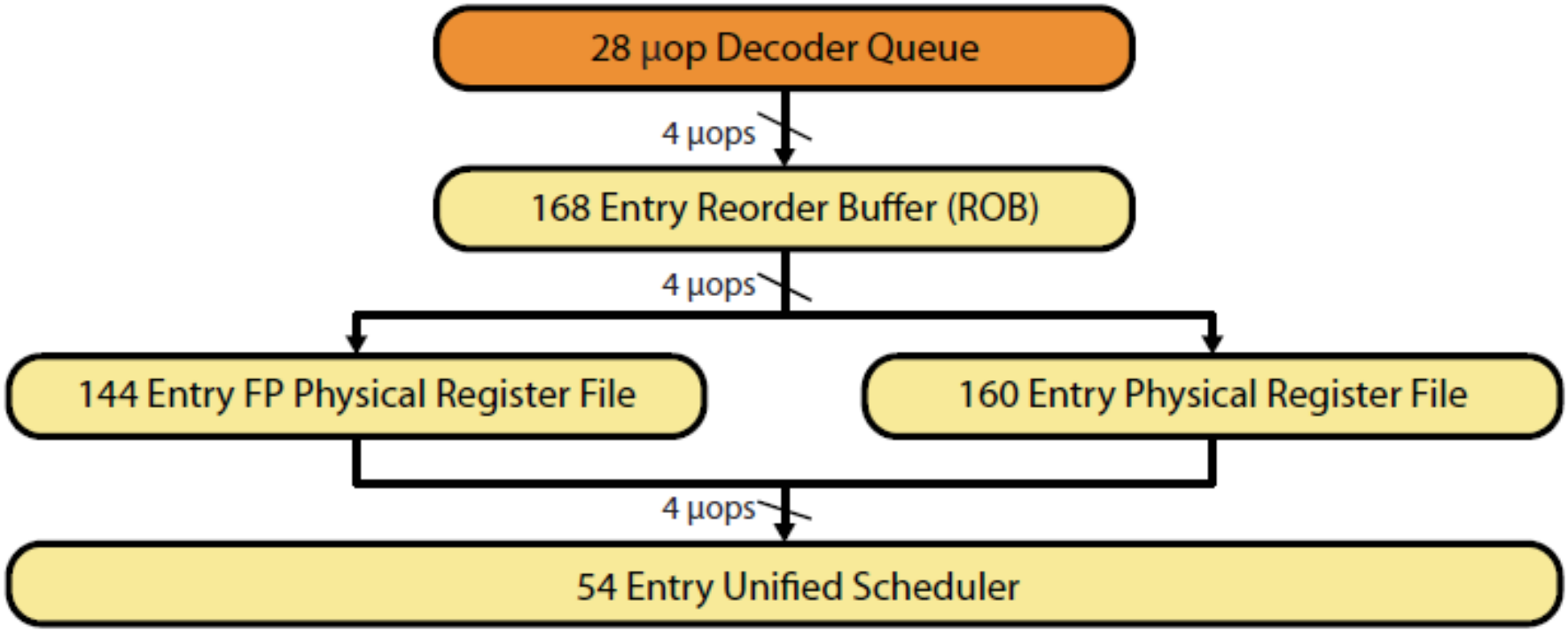
- При выполнении большинства операций оказываются полностью задействованными не все составные компоненты процессоров
- При использовании одновременной многопоточности на одном конвейере выполняются несколько программных потоков
  - Процессор дополняется средствами запоминания состояния потоков, схемами контроля одновременного выполнения нескольких потоков и т. д.
  - Одновременно выполняемые потоки конкурируют за исполнительные блоки единственного процессора, что приводит к возникновению структурных конфликтов

Суперскалярность

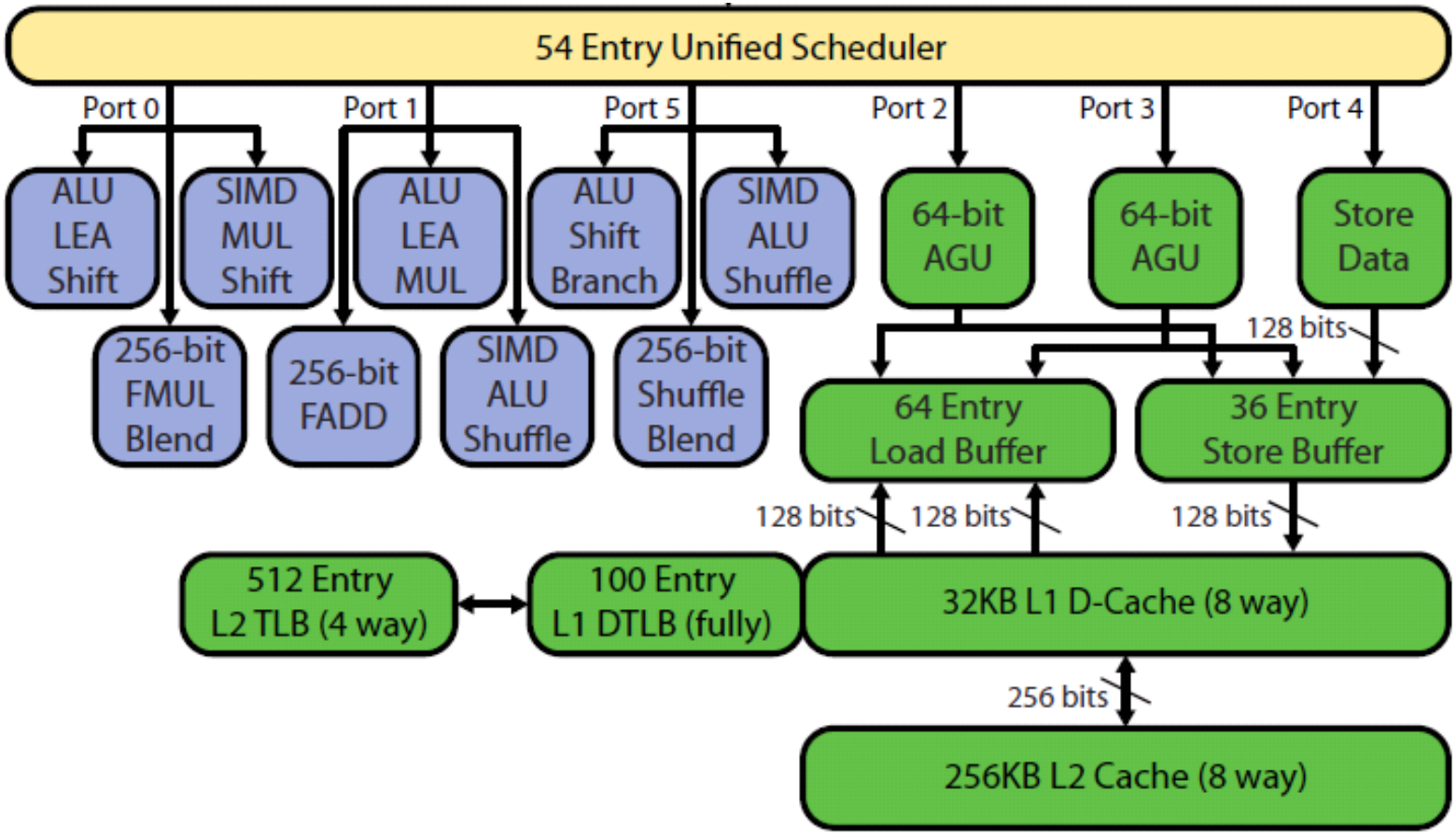
Суперскалярность

- Если инструкции, обрабатываемые конвейером, не противоречат друг другу, и ни одна не зависит от результата другой, то они могут быть выполнены параллельно
- Суперскалярность — запуск на выполнение нескольких инструкций за один такт для использование нескольких исполнительных блоков на различных стадиях конвейера
  - Используются несколько декодеров инструкций
  - Планирование исполнения потока инструкций является динамическим

Суперскалярность
Out-of-Order Execution
(исполнение с изменением последовательности команд)



Суперскалярность Sandy Bridge



Векторные вычисления

Векторные вычисления…

- Вектор – набор однотипных данных (обычно INT/FLOAT/DOUBLE-массив)
- Длина вектора – определяется архитектурой
- Реализации
  - MMX, SSE, SSE2, SSE3, SSE4, AVX, …

Векторные вычисления

- Специальные директивы или intrinsic
- Автоматически компилятором

```
float a[N], b[N], c[N];
for(i = 0; i <= MAX; i++)
    c[i] = a[i] + b[i];
```

Оперативная память и кеш

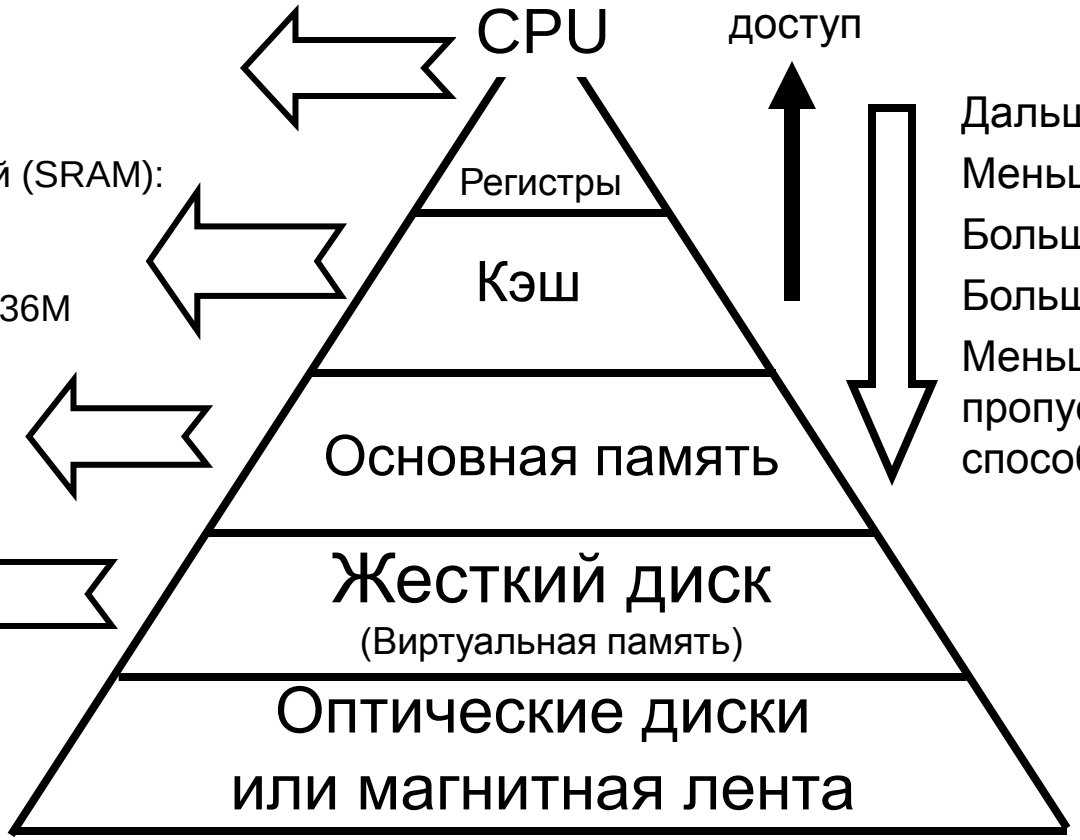
Иерархия памяти

Внутр. регистры
общего назначения
ISA 16-128

Один или более уровней (SRAM):
1: внутр. 16-128K
2: внутр. 256K-16M
3: внутр. или внеш. 1M-36M

Динамическая память
(DRAM) 256M-64G

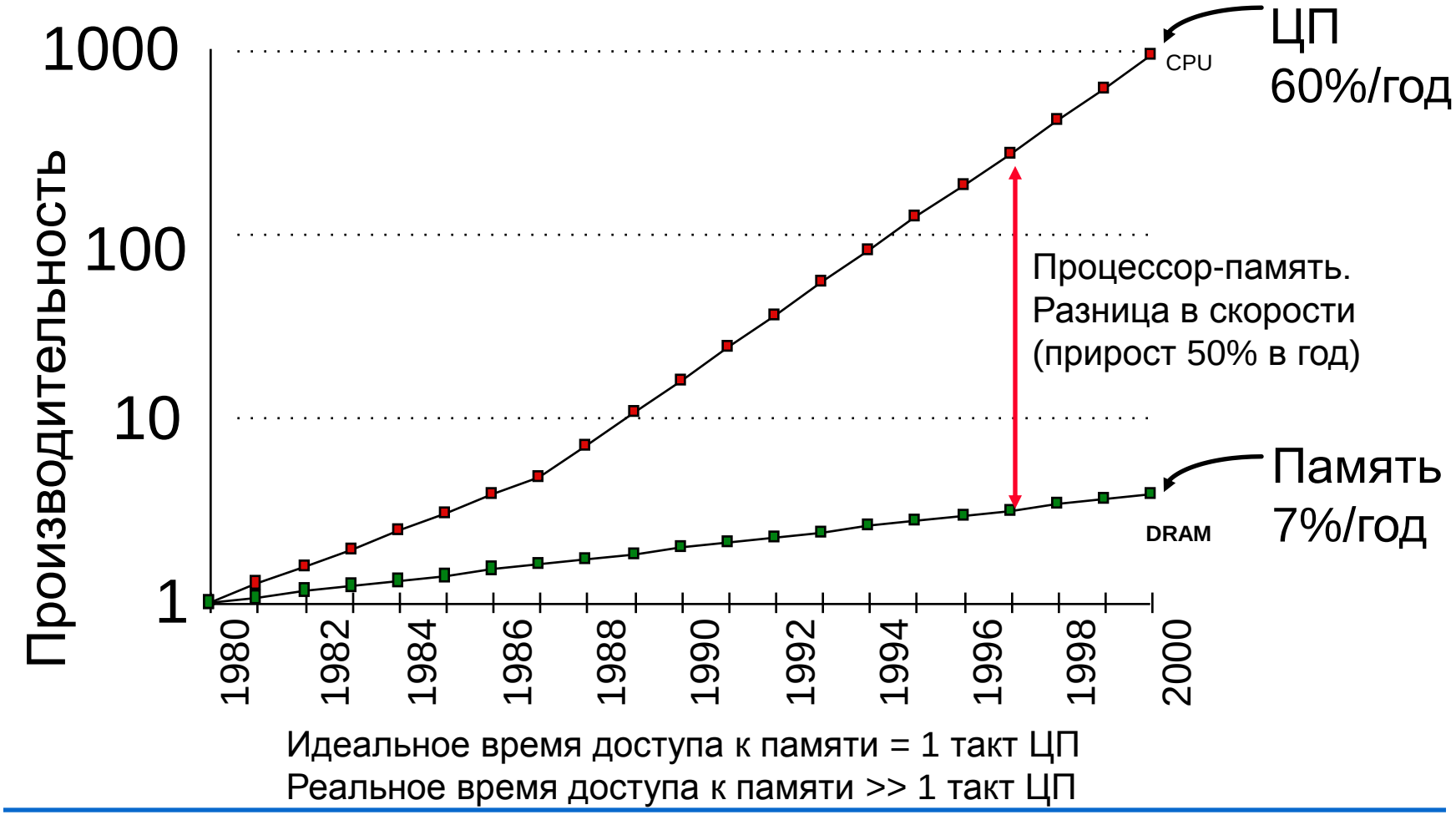
Интерфейсы:
SCSI, RAID,
IDE, 1394
$10^{ки}$Gb - $1^{цы}$Pb



Быстрее
доступ

Дальше от ЦП:
Меньше цена
Больше емкость
Больше задержки
Меньше
пропускная
способность

Разрыв между временем доступа к памяти и временем такта ЦП



Идеальное время доступа к памяти = 1 такт ЦП
Реальное время доступа к памяти >> 1 такт ЦП

Ivy bridge

	Latency	Bandwidth	Size
SRAM (L1, L2, L3)	1-2ns	200GBps	1-20MB
DRAM (memory)	70ns	20GBps	1-20GB
Flash (disk)	70-90µs	200MBps	100-1000GB
HDD (disk)	10ms	1-150MBps	500-3000GB

Базовая модель кэша

- Q1: Как можно разместить блок в кеше?
  (Политика размещения блоков и организация кэша)
  - Полностью ассоциативный, наборно-ассоциативный, кэш с прямым отображением.
- Q2: Как проверяется наличие блока в кэше (попадание/промах)?
  (Идентификация блока)
  - Сравнение тегов.
- Q3: Какой блок должен быть заменен в случае промаха?
  (Алгоритм замещения блока)
  - Случайный, LRU, FIFO.
- Q4: Что происходит при записи?
  (Политика записи кэша)
  - Write through, write back.

Кэш прямого отображения

N блоков памяти могут проецироваться только на одну позицию в кэше. Если код требует доступа к 2 или более блокам проецирующимся на одну позицию кэша, то возникает конфликт и блоки поочередно вытесняют друг друга.

(Адрес блока) MOD (Число блоков в кэше)

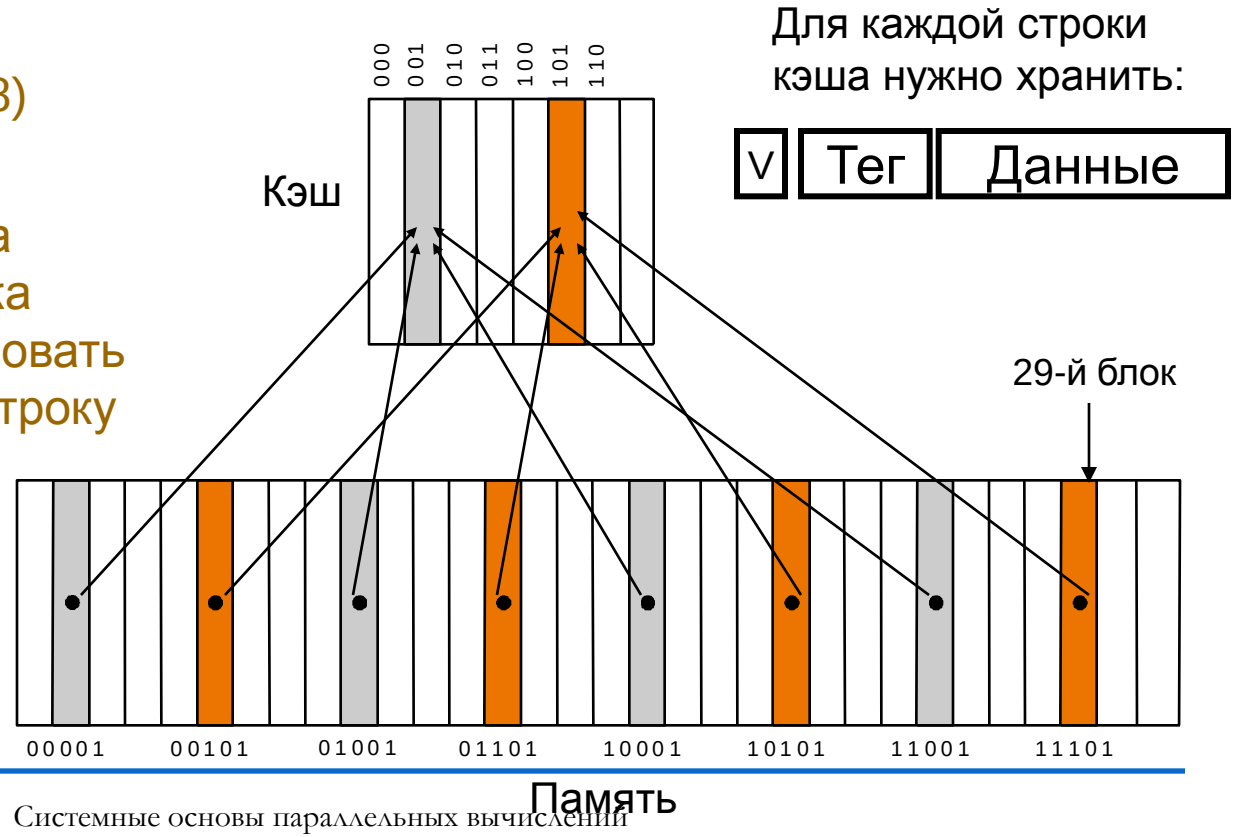
Пример:

(Адрес блока) MOD (8)

Имеем кэш из 8 строк. Объем памяти в 4 раза больше, то есть 4 блока памяти могут конкурировать за «честь» попасть в строку кэша.

29 MOD 8 = 5
(11101) MOD (1000) = 101

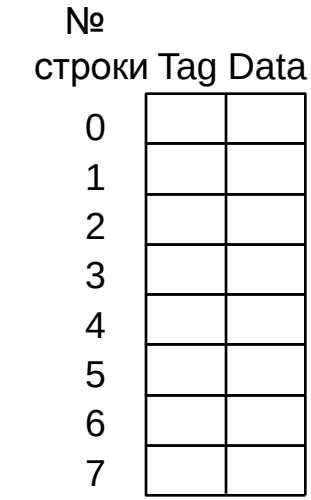
То есть 29-й блок памяти проецируется на 5-ю строку кеша.



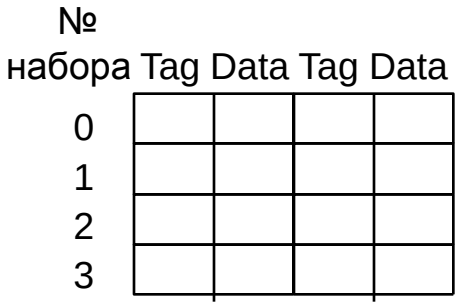
Способы организации кэш-памяти

Рассмотрим варианты организации кэша из 8-ми строк.

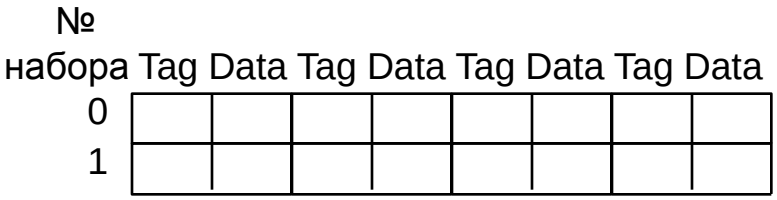
Для каждой строки кэша нужно хранить:



1-way set associative:
(direct mapped)
1 строка в наборе

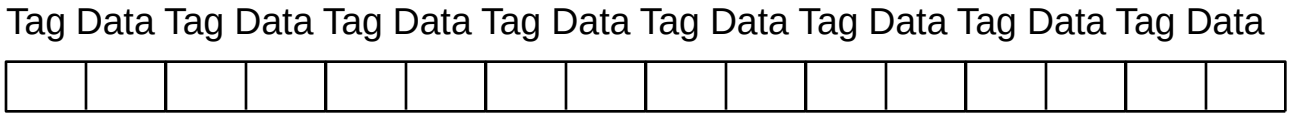


2-way set associative:
2 строки в наборе



4-way set associative:
4 строки в наборе

В наборно-ассоциативном кэше меньше промахов в результате уменьшения числа конфликтов между блоками, которые были бы спроецированы в одну и ту же строку в кэше прямого отображения.



8-way set associative: 8 строк в наборе

Кэш превратился в полностью ассоциативный, поскольку общее количество строк = 8

Ivy bridge

- L1 cache per core
  - L1I 32 KB, 8-WAY, 64 B/line
  - L1D 32 KB, 8-WAY, 64 B/line
- L2 cache 256 KB per core
  - 64 B/line, 8-WAY
- L3 cache 2 MB to 37,5 MB shared
  - 64 B/line, 12-WAY

Основные рекомендации

- Используйте выравнивание
- Учитывайте размер и степень ассоциативности кеша
- Улучшайте локальность своих программ
- Используйте предвыборку данных
  - Hardware Instruction Fetching and Software Prefetching

Современные направления развития архитектуры CPU

- Техники сокрытия длительных задержек при работе с памятью, включающие:
  - рост оптимизации и эффективности систем кэширования
- Улучшенная обработка конфликтов в конвейере
- Улучшенные техники аппаратного предсказания ветвлений
- Оптимизация исполнения инструкций в конвейере
  - Динамическое аппаратное планирование в конвейере
  - Динамическое спекулятивное исполнение
- Использование параллелизма на уровне инструкций (Instruction-Level Parallelism, ILP) при параллельной выдаче множества инструкций на исполнение в множество функциональных устройств
- Включение специальных инструкций для обработки мультимедиа приложений (ограниченная векторная обработка)
- Высокоскоростные шины для повышения скорости передачи данных

Спасибо за внимание