Пути повышения производительности вычислительной системы

- Конвейеризация выполнения команд
- Кэш – память
- Параллелизм на уровне команд (суперскалярные и VLIW)
- Внутрипроцессорная многопоточность (*Hyper-Threading*)
- Параллелизм на уровне ядер/процессоров
- Технология Turbo Boost

Конвейер

Выполнение команд без конвейера

Fetch
Decode
Read Date
Execute
Write

Такты	Выборка команды	Декодирование команды	Чтение операндов	Выполнение операции	Сохранение результата
1	K1	-	-	-	-
2	-	K1	-	-	-
3	-	-	K1	-	-
4	-	-	-	K1	-
5	-	-	-	-	K1
6	K2	-	-	-	-
7	-	K2	-	-	-
8	-	-	K2	-	-
9	-	-	-	K2	-
10	-	-	-	-	K2
11	K3	-	-	-	-
12	-	K3	-	-	-
13	-	-	K3	-	-
14	-	-	-	K3	-
15	-	-	-	-	K3
16	K4	-	-	-	-
17	-	K4	-	-	-
18	-	-	K4	-	-
19	-	-	-	K4	-
20	-	-	-	-	K4
21	K5	-	-	-	-
22	-	K5	-	-	-
23	-	-	K5	-	-
24	-	-	-	K5	-
25	-	-	-	-	K5

- Для выполнения пяти команд процессору понадобилось 25 тактов
- Время выполнения каждой команды 5 тактов
- Время между командами на входе - 5 тактов
- Время между командами на выходе - 5 тактов

Идея конвейера

- Разбить выполнение команды на независимые стадии;
- Выделить для исполнения каждой из них специальный функциональный блок в процессоре;
- Обеспечить параллельную работу этих блоков за счет введения между стадиями промежуточных блоков памяти в которых будут храниться результаты выполнения команды на каждой стадии
- Стадии работы нескольких соседних команд могут совмещены во времени, что дает возможность одновременной обработки (но не выполнения) нескольких команд.

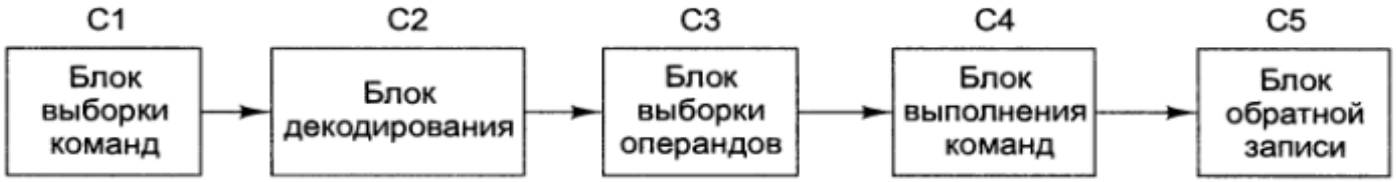
Идея конвейера

Такты	Выборка команды	РЕГИСТР	Декодирование команды	РЕГИСТР	Чтение операндов	РЕГИСТР	Выполнение операции	РЕГИСТР	Сохранение результата
1	K1		-		-		-		-
2	K2		K1		-		-		-
3	K3		K2		K1		-		-
4	K4		K3		K2		K1		-
5	K5		K4		K3		K2		K1
6	-		K5		K4		K3		K2
7	-		-		K5		K4		K3
8	-		-		-		K5		K4
9	-		-		-		-		K5

- Пять команд выполнены за 9 тактов
- Время выполнения каждой команды 5 тактов
- Время между командами на входе - 1 такт
- Время между командами на выходе - 1 такт

Конвейер

- Количество аппаратных блоков, которые участвуют в работе конвейера называют *ступенью/стадией конвейера*.



- *Пяти ступенчатый конвейер*
- **Скалярный процессор** – процессор у которого в стадии «Execute» выполняется одна команда над одной порцией данных.
- Имеет, как правило, один конвейер.

Проблемы конвейера

- Риски (hazard) = конфликты конвейера

  Конфликт по ресурсам – несколько команд обращаются к одному ресурсу (например памяти)

  - Решение проблемы : разделение памяти на кэш команд и кэш данных.
  - Многопортовое ЗУ

- **Конфликты по данным**

  - Выполнение одной команды зависит от результата второй

- **Конфликты по управлению**

  - возникают при *конвейеризации команд переходов* и других команд, изменяющих *значение счетчика команд*.

Конфликт по данным

- Команды 1 – 2 – 3 могут выполнятся одновременно

$$1)\ A = B + C$$

$$2)\ Z = X + Y$$

$$3)\ K = A + M$$

- Команды 1 - 2 не могут выполнятся одновременно, что приведет к остановке конвейера

$$1)\ A = B + C$$

$$2)\ K = A + M$$

$$3)\ Z = X + Y$$

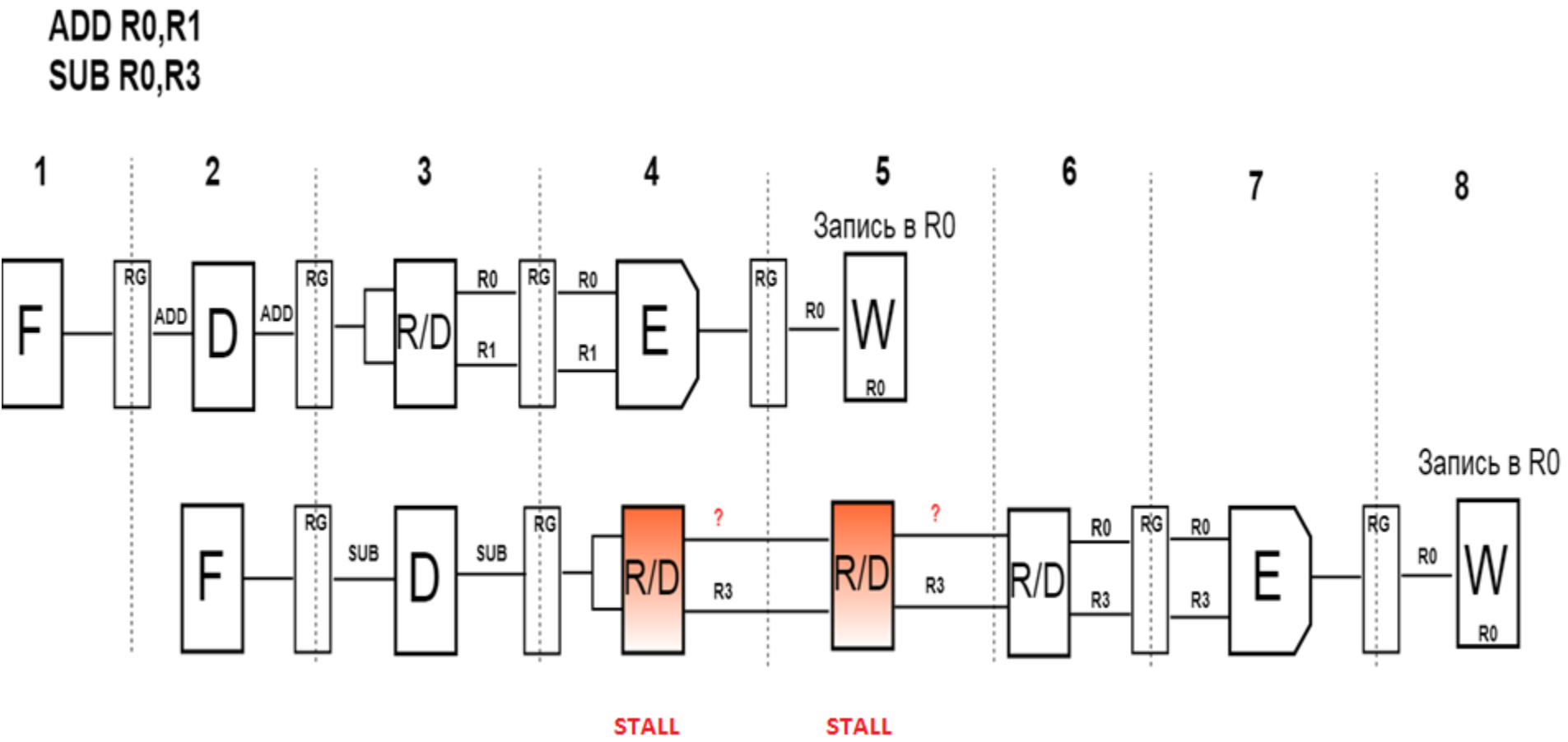
Конфликт по данным

- *I* - первая команда
- *J* - вторая команда
- ***j > i***

Конфликты по данным: *а* — «чтение после записи»; *б* — «запись после чтения»; *в* — «запись после записи»

- «Ч*тение после записи*» (ЧПЗ - **Read After Write**): вторая команда (*j)* читает *операнд* до того, как первая команда (*i)* успела записать его новое значение. **(наиболее распространены)**
- «*Запись после чтения*» (ЗПЧ - **Write After Read**): вторая команда (*j)* записывает новое значение *операнда* до того, как первая команда(*i*)успела прочитать *его*.
- «*Запись после записи*» (ЗПЗ - **Write After Write**): вторая команда (*j)* записывает новое значение *операнда* прежде, чем первая команда (*i)* успела записать в его новое значение
- **WAR и WAW появляются в результате изменения последовательности выполнения команд в конвейере**

RAW-конфликт по данным



- Вторая команда читает R0 раньше, чем первая успела записать новое значение.

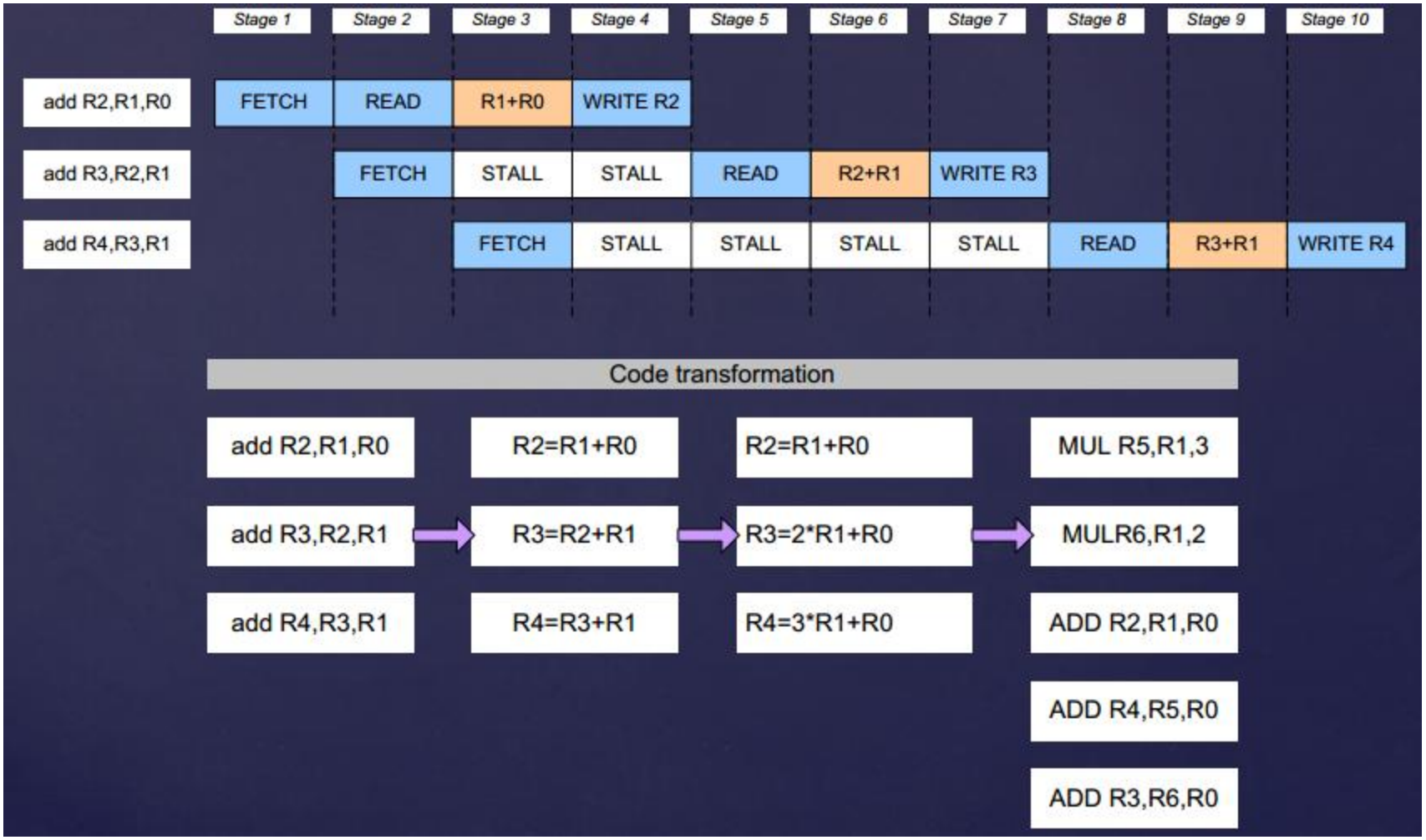
Минимизация конфликтов по данным

- **Программно** – с помощью оптимизирующего компилятора

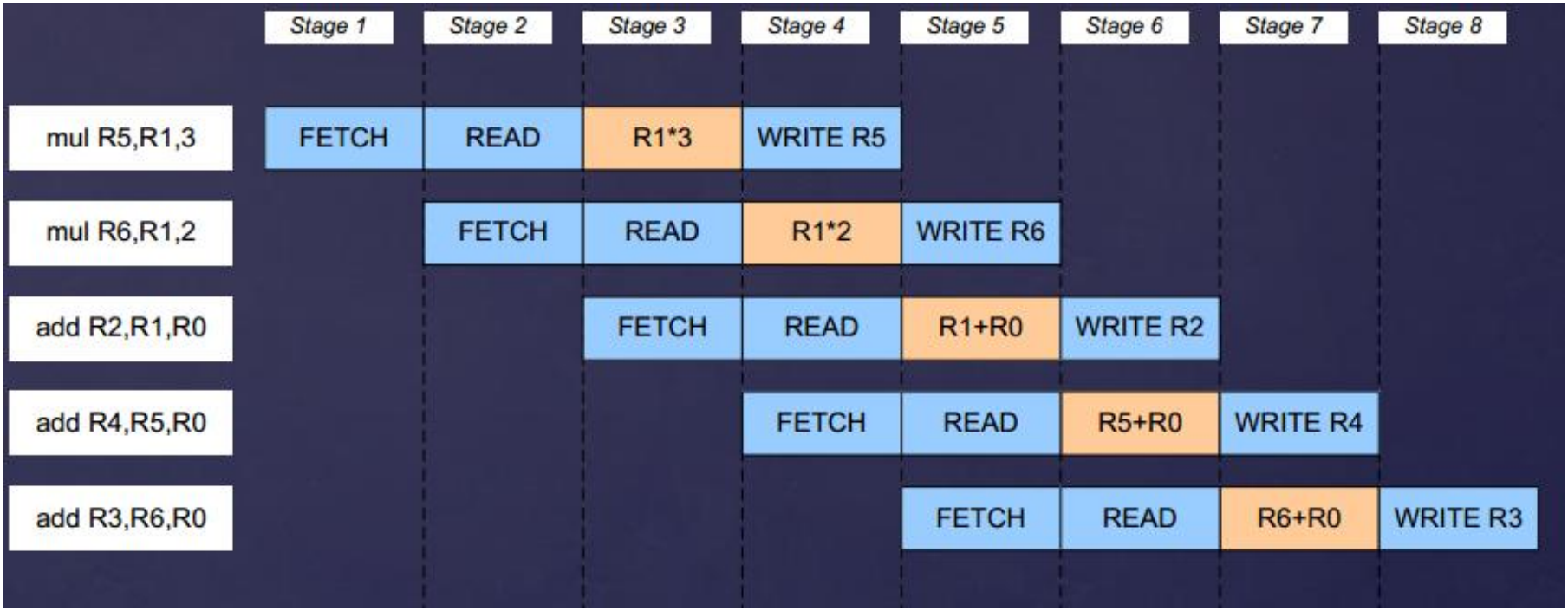
Аппаратно, дополнительными блоками процессора через:

- Подмену регистров
- Внеочередное выполнение команд (с помощью планировщика инструкций)
- Быструю пересылку (fast forwarding)
- Приостановку конвейера на несколько тактов (не желательно)

Программная минимизация конфликтов (пример)



Программная минимизация конфликтов (пример)



Аппаратная минимизация конфликтов по данным

Внеочередное выполнение команд, подмена регистров

ДО

Такт	Номер команды	Действие	Поступление	Длительность
1	1	R3:=R0×R1	1	6
2	2	R4:=R0+R2	2	6
3	3	R5:=R0+R1	3	6
4	4	R6:=R1+R3		
5				
6				
7			4	6
8	5	R7:=R0×R8	5	6
9	6	R1:=R0×R0	6	6
10	7	R3:=R1–R2		

RAW

STALL

STALL

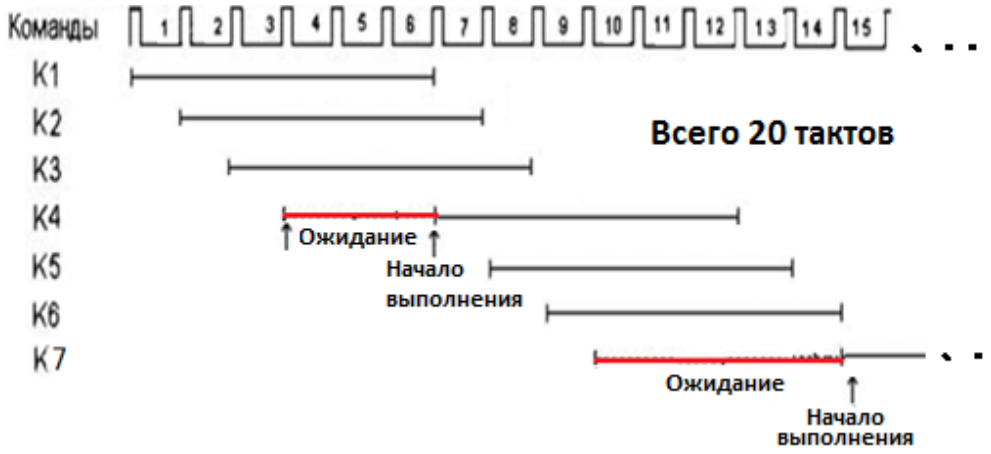
RAW

ПОСЛЕ

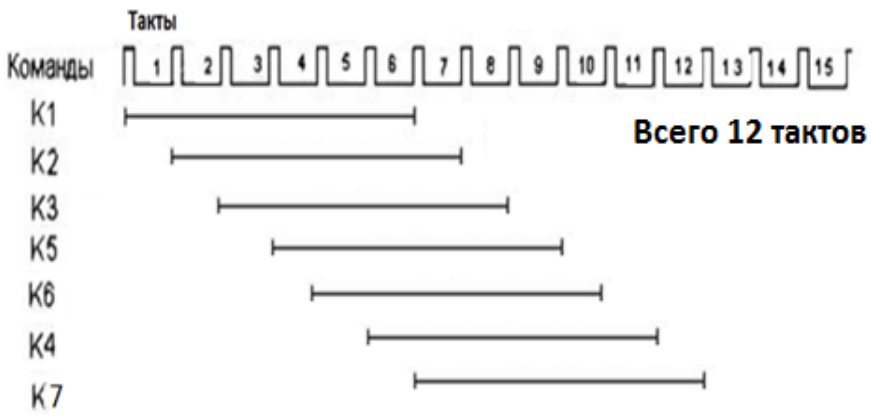
Такт	Номер команды	Действие
1	1	R3:=R0×R1
2	2	R4:=R0+R2
3	3	R5:=R0+R1
4	5	R7:=R0×R8
5	6	S1:=R0×R0
6	4	R6:=R1+R3
7	7	R3:=S1–R2
8		
9		
10		

Перестановка

Подмена

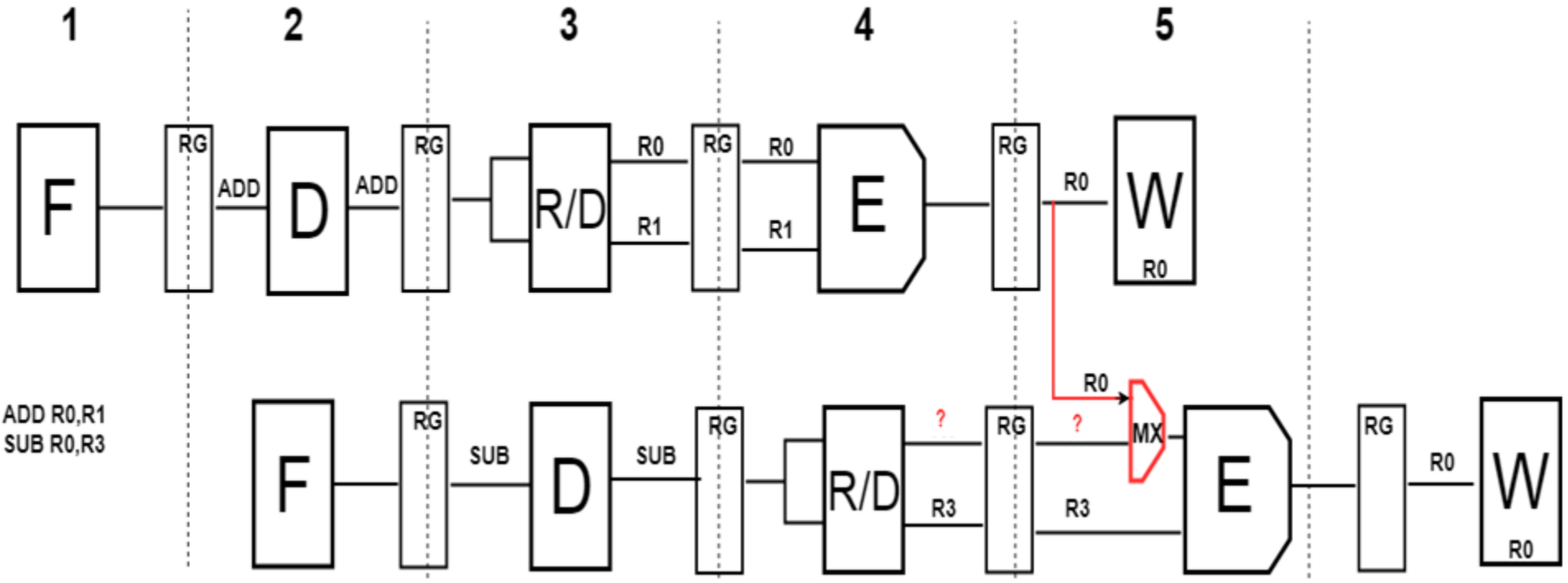


Всего 20 тактов



Всего 12 тактов

Fast forwarding (байпас)



- Быстрая пересылка данных с выхода АЛУ на его входы через мультиплексор минуя промежуточную стадию сохранения

Конфликты по управлению

- На каждые 6-8 команд приходится 1 команда перехода

- Способы конвейеризации команд ветвления:
- Аппаратный/динамический (зависит от истории выполнения программы) выполняется аппаратурой
  - Предсказание ветвления и спекулятивное выполнение команд предсказанной ветви
  - Предикатное выполнение

- Программный /статический (не зависит от истории выполнения программы) выполняется на этапе создания и компиляции программы.

Предсказание перехода (аппаратный метод)

- Для программы в специальном буфере процессора накапливается статистика по каждой используемой команде перехода .

- Статистика содержит :
  - команду перехода,
  - адрес перехода
  - поле признака перехода (который устанавливается всякий раз когда был переход по адресу перехода).

- Блок предсказания переходов использует поле признака для предсказания перехода (пример циклы).

- Вероятность удачного предсказания ветвления может превышать 90%.

Спекулятивное выполнение

- Спекулятивное выполнение – выполнение инструкции заранее до того, когда потребуется её выполнение.
- Бывает:
  - спекуляция команд управления - инструкции предсказанной ветви перехода выполняются раньше команды ветвления
  - Спекулятивное (опережающее) чтение данных - данные считываются из памяти раньше их возможного использования

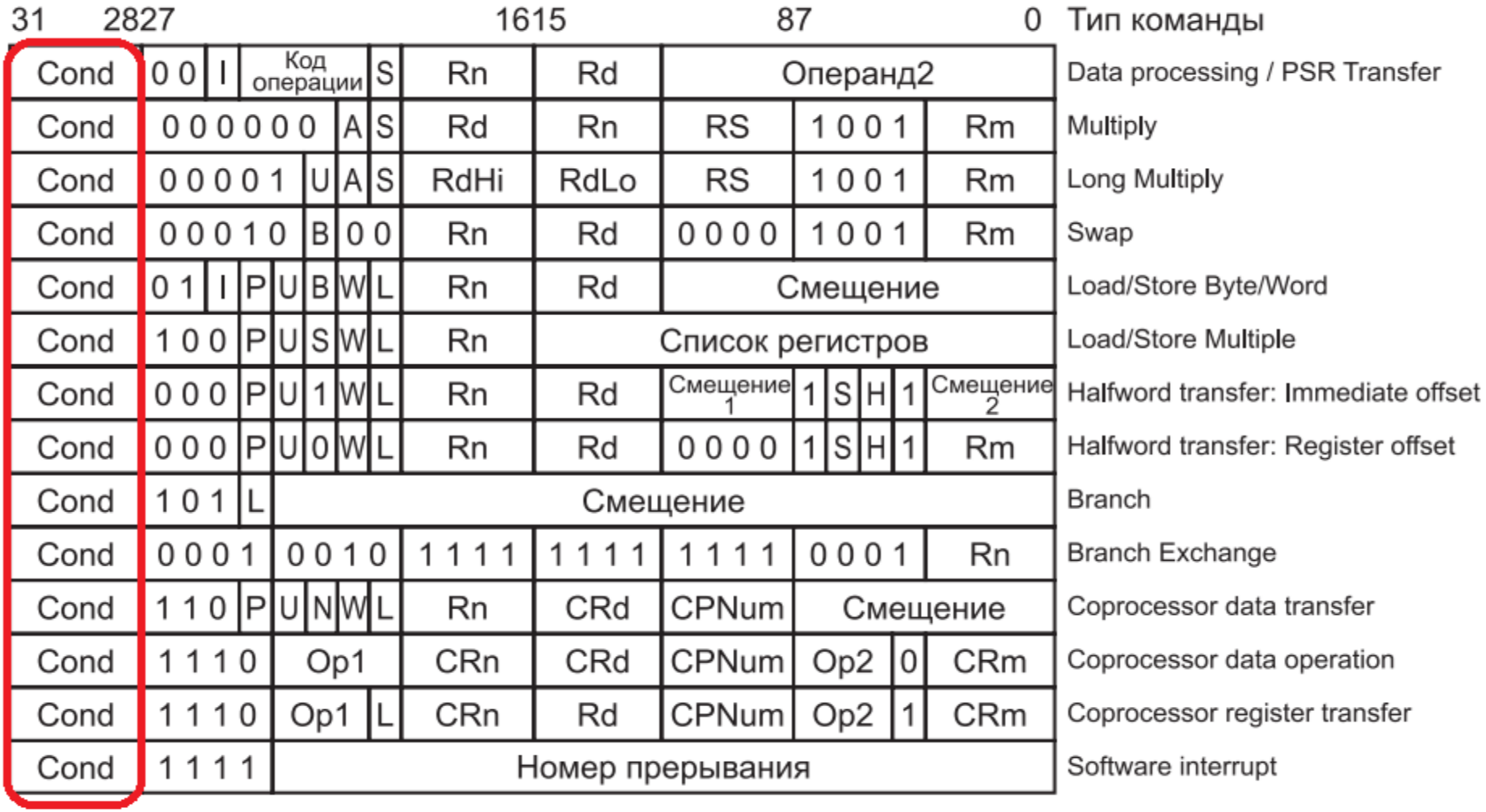
Предикатное выполнение(аппаратный метод)

- Конвейер последовательно выполняет команду проверки условия и команды из *обеих* ветвей условного перехода .

- Результаты выполнения команд обеих ветвей запоминаются в теневых регистрах процессора.

- После выполнения условия, выбирается нужная ветвь и данные переписываются в реальные регистры, а альтернативная ветвь игнорируется

Предикатное выполнение (на примере ARM)



- В каждой команде есть поле условий cond (4 бита), которое делает её предикатной

Формат поля cond

Мнемонические обозначения условий

cond	Мнемоника	Название	CondEx
0000	EQ	Равно	Z
0001	NE	Не равно	$\overline{Z}$
0010	CS/HS	Флаг переноса поднят / беззнаковое больше или равно	C
0011	CC/LO	Флаг переноса сброшен / беззнаковое меньше	$\overline{C}$
0100	MI	Минус / отрицательное	N
0101	PL	Плюс / положительное или нуль	$\overline{N}$
0111	VS	Переполнение / флаг переполнения поднят	V
1000	VC	Переполнения нет / флаг переполнения сброшен	$\overline{V}$
1001	HI	Беззнаковое больше	$\overline{Z}C$
1010	LS	Беззнаковое меньше или равно	Z OR $\overline{C}$
1011	GE	Знаковое больше или равно	$\overline{N \oplus V}$
1100	LT	Знаковое меньше	$N \oplus V$
1101	GT	Знаковое больше	$\overline{Z}\,\overline{(N \oplus V)}$
1110	LE	Знаковое меньше или равно	Z OR $(N \oplus V)$
	AL (или пусто)	Всегда / безусловно	Игнорируется

- Компилятор подставляет в поле cond требуемый код мнемоники

Команды условий

- Команды условий устанавливают флаги условий в регистре состояния программы (регистр флагов)

Флаг	Название	Описание
N	Negative	Результат выполнения команды отрицателен, т. е. бит 31 равен 1
Z	Zero	Результат выполнения команды равен нулю
C	Carry	Команда привела к переносу
V	oVerflow	Команда привела к переполнению

Предикатное исполнение

```
If (R1 == R2) /* ANSI C-primer */
  R3 = R4 + R5;
else
  R6 = R4 - R5
```

Assembler with no predicate

```
    CMP R1,R2
     BNE L1
    MOV R3,R4
    ADD R3,R5
    BR L2 ; similar to previous case
L1: MOV R6,R4
    SUB R6,R5
L2: ; it's end !...
```

Assembler with predicate

```
CMP R1,R2
ADDEQ  R3, R4, R5
SUBNE  R6, R4, R5
```

- Различные ветви программы могут выполняться параллельно под предикатом (это позволяет перевести зависимость по управлению в зависимость по данным).

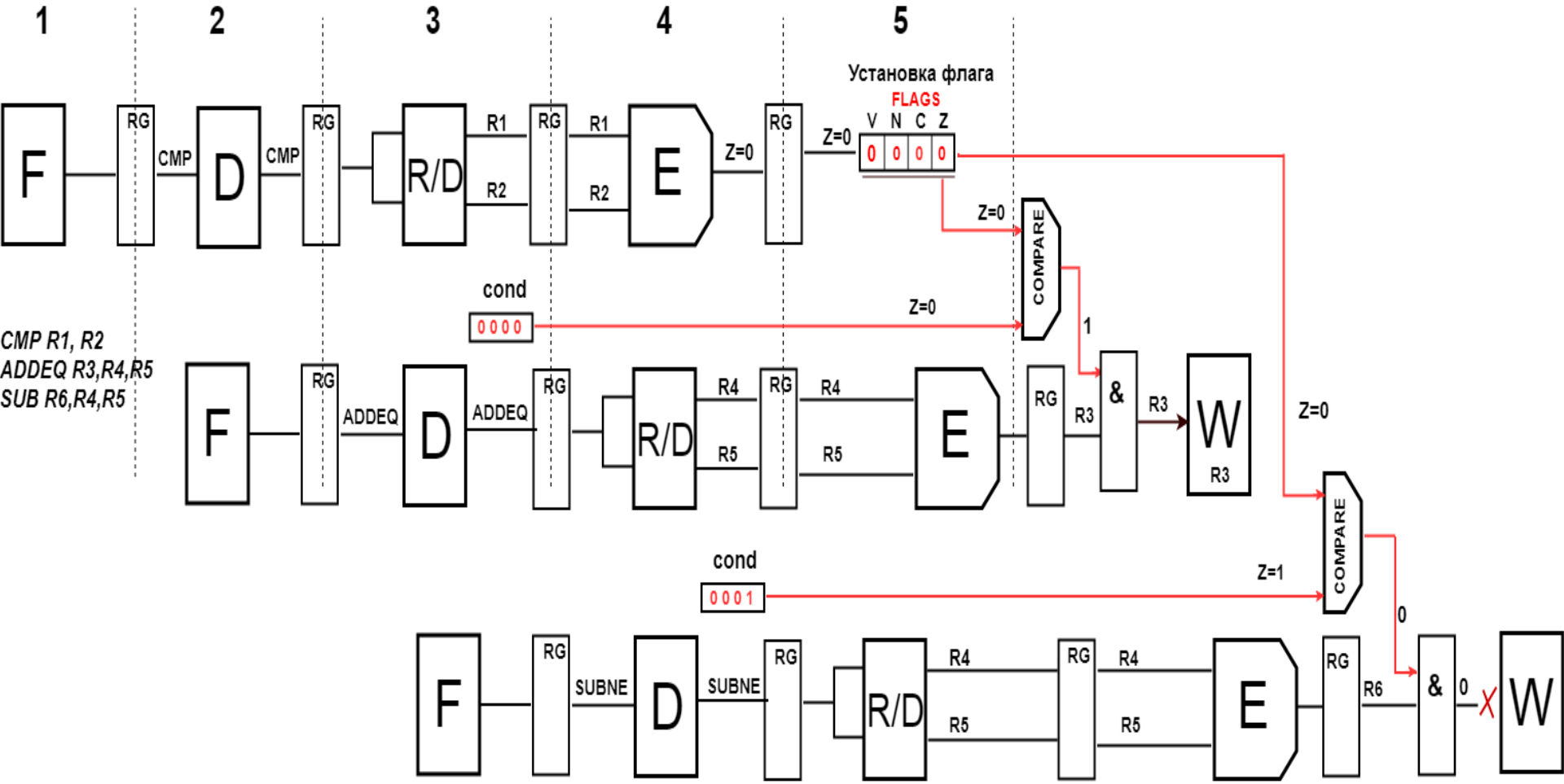
Поле cond

- Если команда использует поле cond, то после названия команды добавляется соответствующая мнемоника.
  - Например ***ADDEQ***

CMP R1,R2 ; при R1 = R2 команда CMP устанавливает флаг Z =0 в регистре флагов
ADDEQ R3,R4,R5 ; результат команды сохраняется в R3, в поле cond есть условие EQ
SUBNE R6,R4,R5 ; результат не сохраняется в R6, поле cond NE требует Z=1

- Следующие за командой условия команды сравнивают биты из своего поля cond с битами флагов и в случае сравнения результат выполнения команды сохраняется в .

Предикатное выполнение



Поле cond и выполнение условий в конвейере

- Компилятор предварительно заполняет поле Cond
- Команда условия и следующие за ней команды последовательно подаются на конвейер.
- Команда условия устанавливает соответствующий бит в слове состояния программы (в нашем случае Z=1).
- После стадии выполнения второй команды (ADD) происходит сравнение поля cond и бита признака Z=1, они соответствуют и результат выполнения записывается в R4.
- Следующая команда (SUB) также выполняется на конвейере, но так как она использует признак Z=0, то результат её выполнения не запишется в регистр R6.

Ветвления на этапе компиляции

- 1. Минимизация ветвлений (проверок условия) в программе, например:
  - вынос ветвлений за пределы цикла;
  - развертка цикла.

- 2. Предсказание ветвлений на этапе компиляции
  - В программе переход назад обычно соответствует циклу и выполняется в 90% случаев, а переход вперед обозначает ветвление и выполняется в 50% случаев.
  - На этапе компиляции это можно определить, и в машинной программе пометить предполагаемое направление перехода.

Вынос условия за пределы цикла

- До

```
for (i = 0; i < 1000; i++)
{
    x[i] += y[i];

    if (w)
    {
        y[i] = 0;
    }
}
```

После

```
if (w)
{
    for (i = 0; i < 1000; i++)
    {
        x[i] += y[i];
        y[i] = 0;
    }
}
else
{
    for (i = 0; i < 1000; i++)
    {
        x[i] += y[i];
    }
}
```

Развертка цикла

До	После	После №2

`for (int i = 0; i < iN; i++){`
`res *= a[i];`
`}`
	`for (int i = 0; i < iN; i+=3){`
	`res *= a[i];`
	`res *= a[i+1];`
	`res *= a[i+2];`
	`}`
		`for (int i = 0; i < iN; i+=3){`
		`res1 *= a[i];`
		`res2 *= a[i+1];`
		`res3 *= a[i+2];`
		`}`
		`res = res1 * res2 * res3;`

- Количество итераций (проверок условия) в три раза меньше
- Переменные res1, res2, res3 – для исключения зависимости по данным

КЭШ

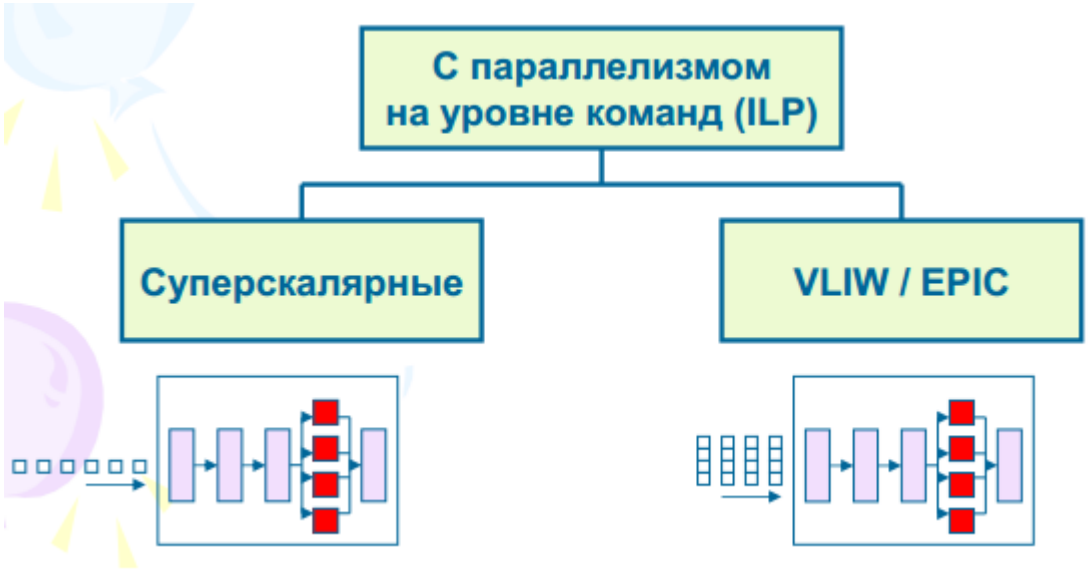
Кэш и основная память

Один раз прочитать большой блок из медленной оперативной памяти в кэш, а потом много раз обращаться к быстрому кэшу.

Параллелизм на уровне команд

Параллелизм на уровне команд

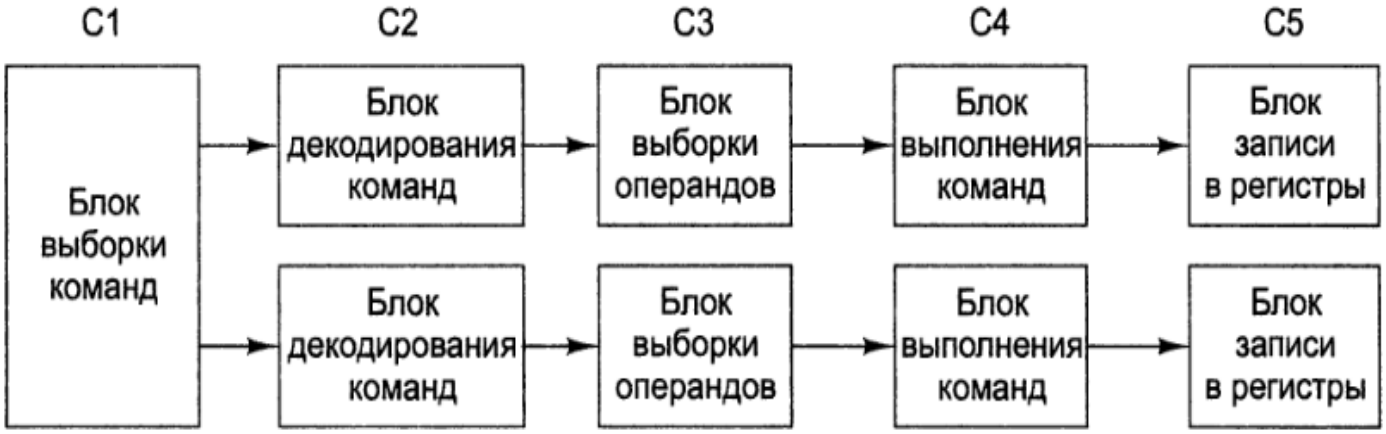
- ILP - Instruction Level Parallelism



- Несколько команд на стадии выполнения одновременно
  - **Суперскалярные процессоры**
  - **VLIW – Very Long Instruction Word** («сверхдлинная машинная команда»)
- **EPIC – Explicitly Parallel Instruction Computing** («вычисление с явным параллелизмом машинных команд»)

Суперскалярный процессор

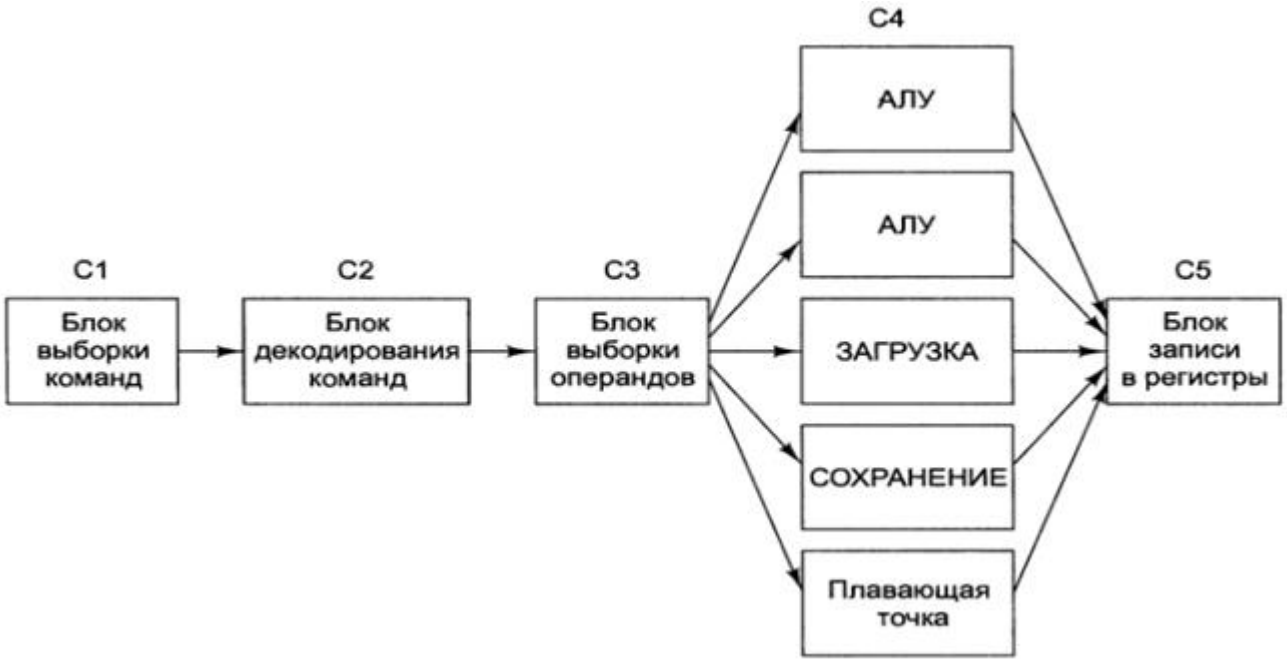
- **Суперскалярный** процессор - исполняющие модули конвейера присутствуют в нескольких экземплярах и в стадии выполнения находится более одной команды.



- Суперскалярный, сеперконвейерный процессор.

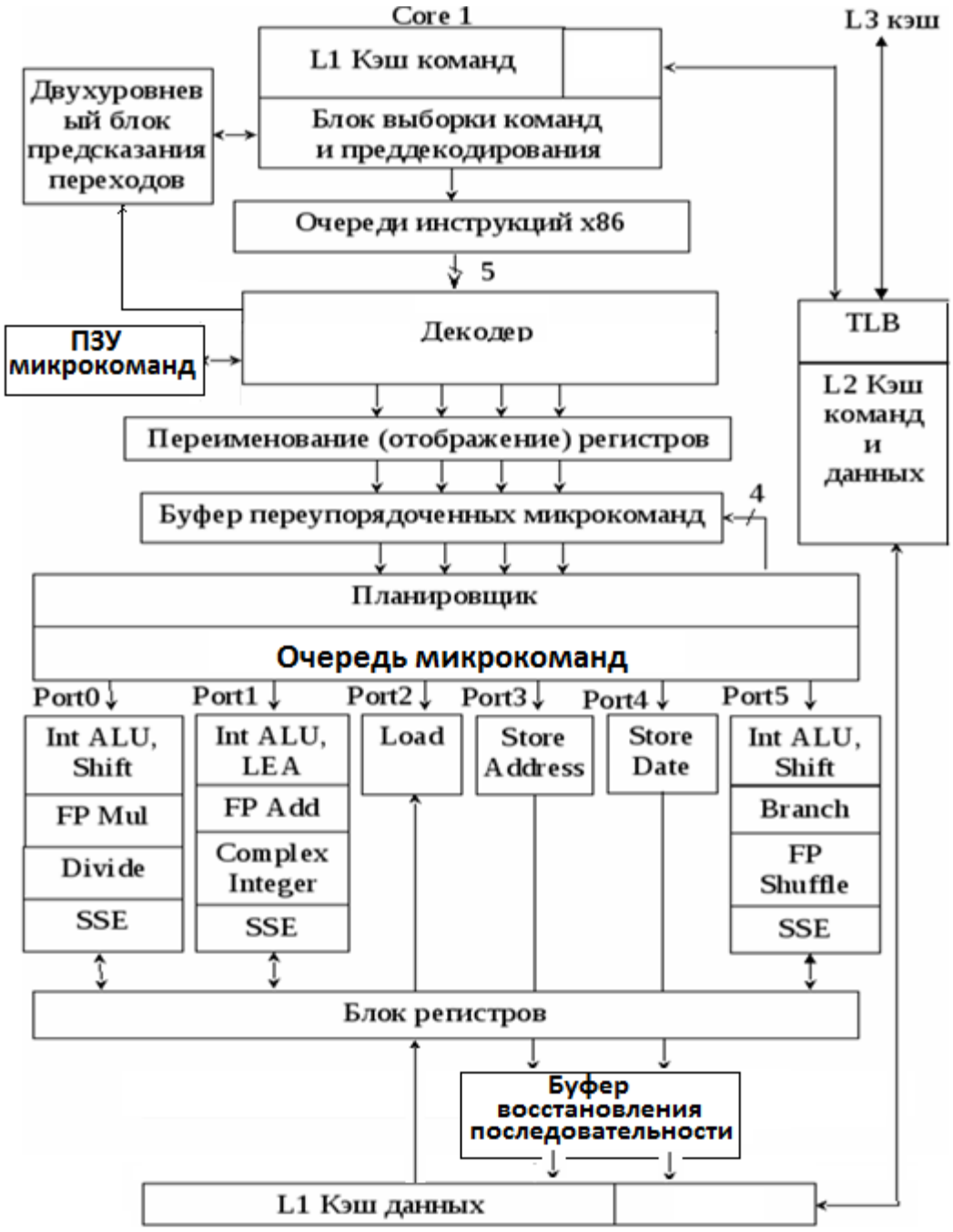
Вариант суперскалярной архитектуры

- Пятиступенчатый конвейер с пятью функциональными блоками

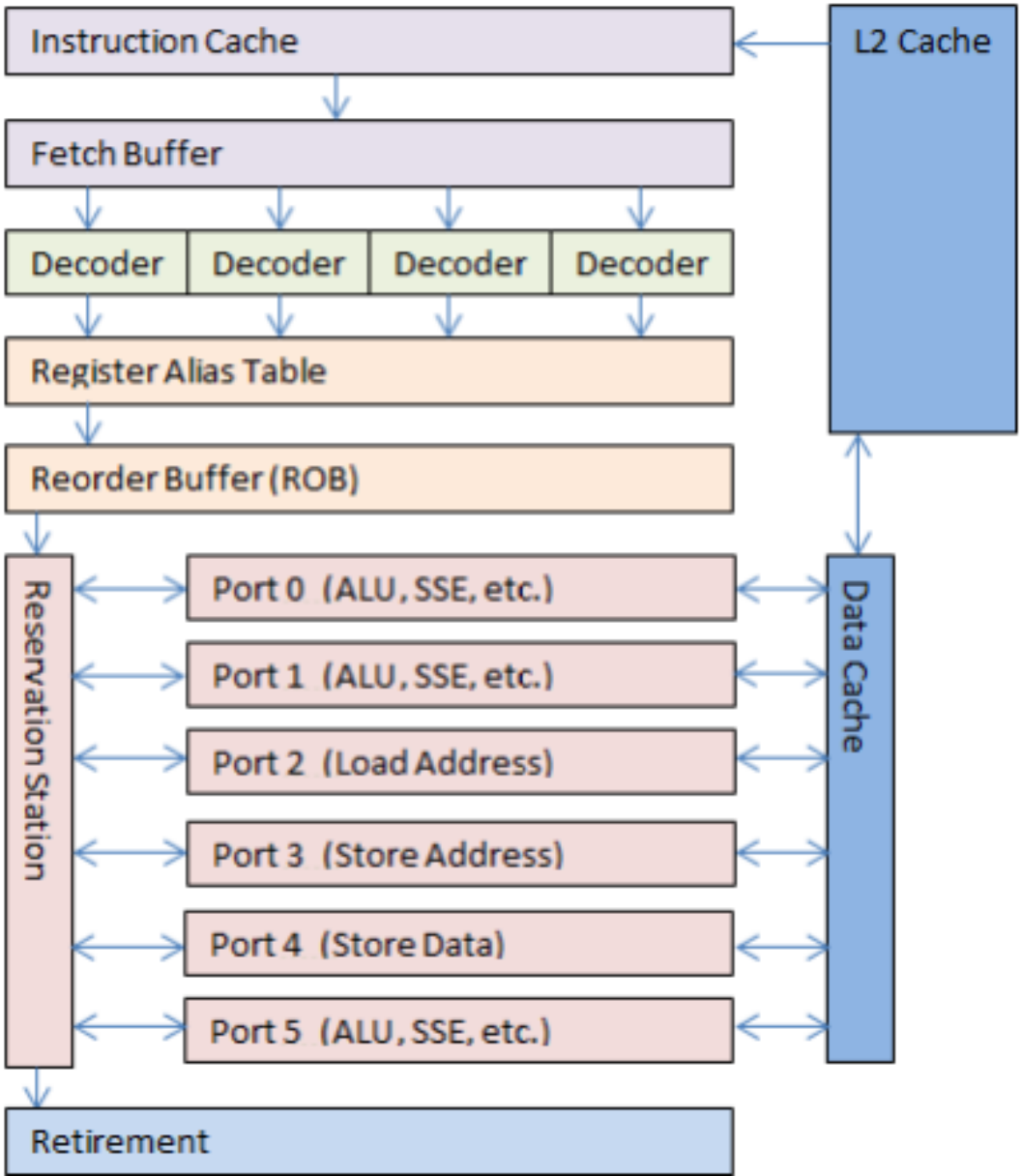


- Более дешевый вариант
- Число параллельно выполняющихся инструкций от 2 до 6

Конвейер суперскалярного процессора Intel Core i7 (Nehalem)

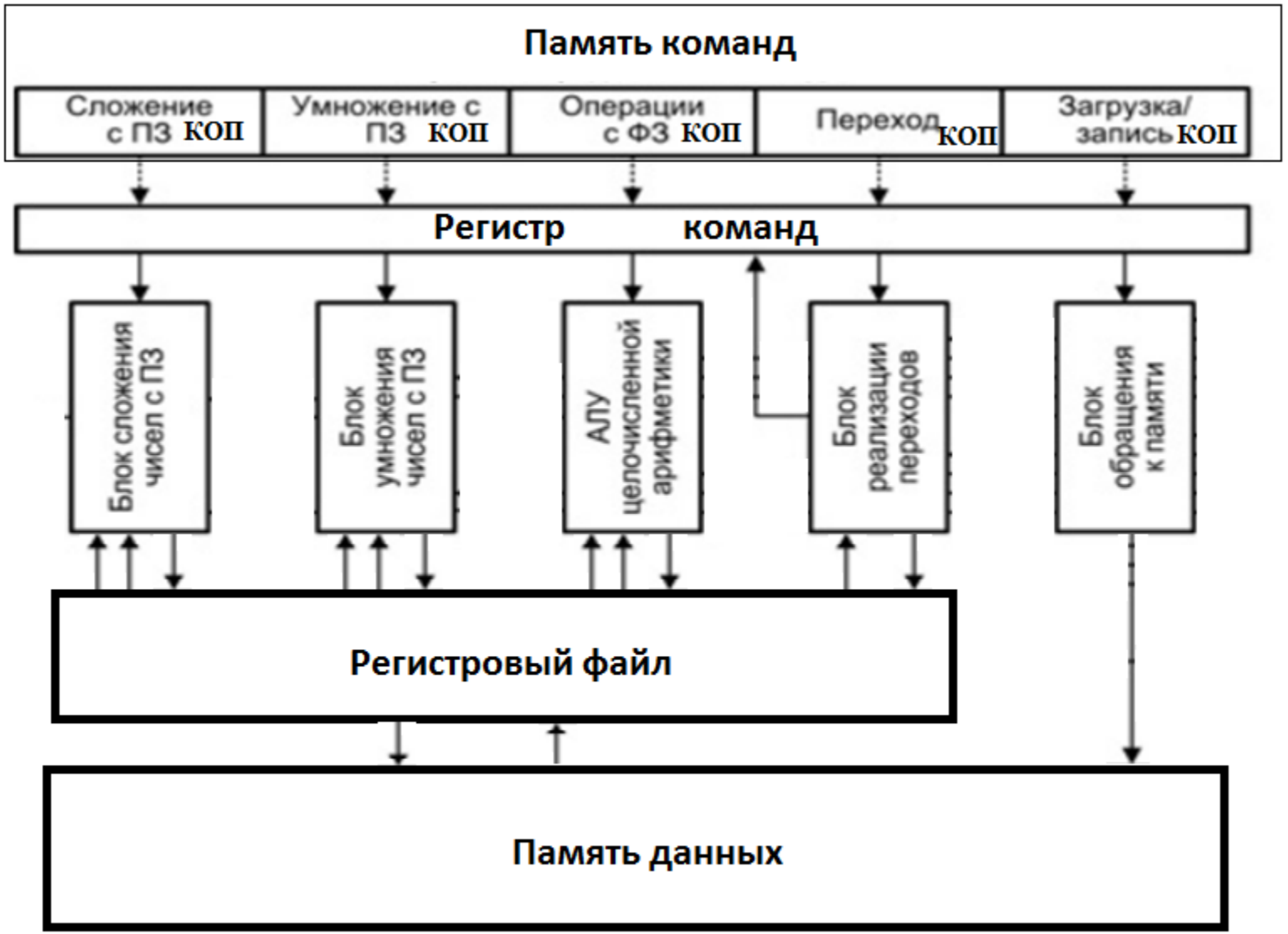


Суперскалярный процессор



Very Long Instruction Word

Концепция VLIW (Very Long Instruction Word)

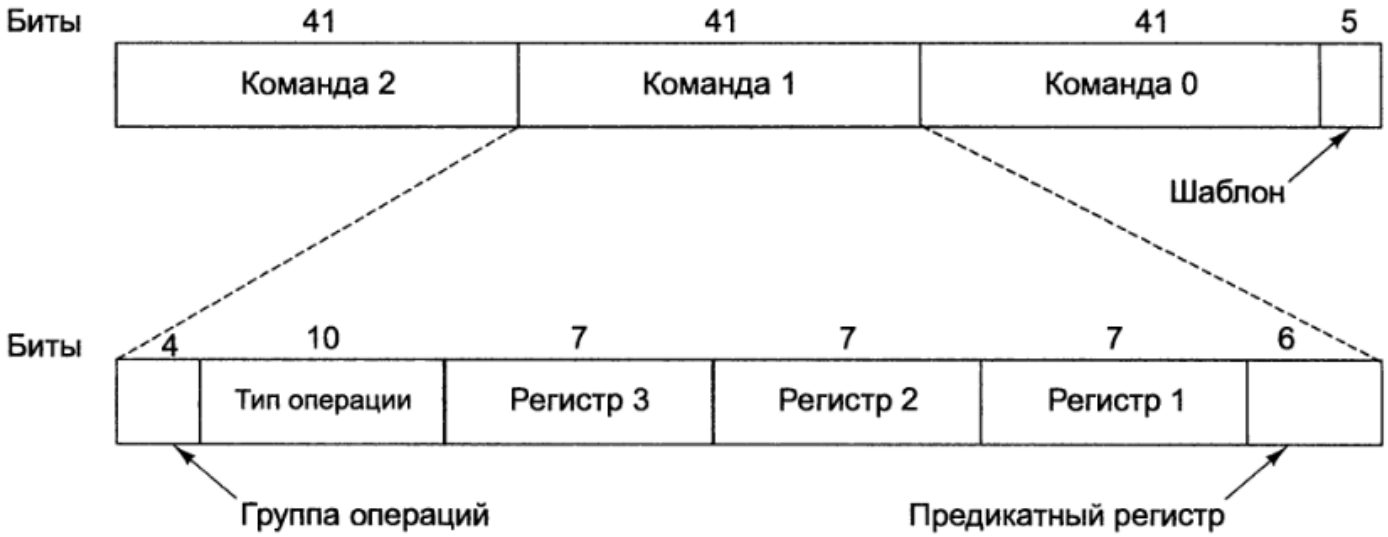


КОП - Код Операции

Концепция VLIW/EPIC

- Компилятор преобразует программу в набор простых команд
- Компилятор анализирует программу и объединяет простые независимые команды в сверхдлинные команды
- Количество простых команд в длинной команде равно количеству исполнительных блоков процессора работающих параллельно
- Компилятор предсказывает переходы
- Так как команды независимы, то меньше конфликтов, меньше аппаратных блоков для их разрешения, больше места для регистров и функциональных блоков, меньше потребление.
- Из-за использования предикатных регистров отсутствует аппаратный блок предсказания ветвлений, они предсказываются на этапе компиляции.
- АЛУ выполняет операции над операндами, хранящимися только в регистрах.
- Недостатки:
  - Сложный компилятор и большее время компиляции
  - Сложно учесть динамику исполнения программы

Пример команды архитектуры IA-64



- Команды, которые можно выполнить параллельно, объединяются компилятором в 128-разрядные пучки (bundles).
- В каждом пучке содержится три 41-разрядных команды и 5-разрядный шаблон.
- Шаблон пучка указывает, какие функциональные блоки процессора и в какой последовательности нужны для его обработки.
- Предикатный регистр служит для организации ветвлений

Особенности IA – 64 (Itanium)

- Трехадресные простые команды только с регистровой адресаций , нет обращения к внешней медленной памяти
- Для обращения к памяти две специальные команды «Загрузка» и «Сохранение»
- Регистровый файл (128*64-разр. РОН, 128 * 82-разр. для пл.точки, 128 – специальных регистра, 64 *1 – ых предикатных регистра)
- Параллельно работающие функциональные блоки процессора
- Трехуровневый кэш
- Конвейер
- Предикатное выполнение условий

- Недостаток:
  - Очень медленная эмуляция инструкций IA32

Предикатное выполнение условий

Оператор if

```
if(R1 == R2)
        R3 = R4 + R5;
Else
        R6 = R4 – R5
```

Код на ассемблере **IA 32**

```
        CMP R1,R2
        BNE L1
        MOV R3,R4
        ADD R3,R5
        BR L2
L1:     MOV R6,R4
        SUB R6,R5
L2:
```

Предикатное выполнение **на асемблере IA64**

```
CMPEQ R1,R2,P4
<P4> ADD R3,R4,R5
<P5> SUB R6,R4,R5
```

P4, P5 - предикатные одноб итные регистры

Предикатное выполнение условий

- С каждой командой проверки условия связываются два однобитных предикатных регистра P4 и P5.
- Если условие выполняется, то P4 устанавливается в 1, а P5 в 0 и наоборот, если условие не выполняется.
- Номера предикатных регистров указывается в командах, следующих за командой проверки условия (P4 в ADD R3,R4,R5 и P5 в SUB R6, R4, R5)
- Сначала на конвейере выполняется команда проверки условия и устанавливаются значения в предикатных регистров
- Далее выполняются команды с обеих ветвей
- На заключительном этапе выбирается та команда, в которой соответствующий предикатный регистр установлен в единицу.
- При этом используются теневые регистры

Сравнение VLIM и суперскаляра

- VLIM
  - Зависимость по данным устраняет компилятор за сет оптимизации кода.
  - Предсказание ветвлений компилятором
  - Предикатное выполнение условий
  - Сложнее компилятор
  - Проще процессор

- Суперскаляр
  - Перестановка команд и подмена регитров выполняется аппаратно во время выполнения процессором (можно лучше использовать динамику выполнения программы)
  - Аппаратное предсказание ветвлений и спекулятивное выполнение

  - Проще компилятор
  - Сложнее процессор

Бабаян Б.А.

https://www.youtube.com/watch?v=mKyA9Ock55E

VLIW в России

Характеристика	Описание
Год выпуска	Производство с 2020 года
Техпроцесс, нм	28
Количество ядер	8
Тактовая частота, ГГЦ	1,5
Производительность, Гфлопс	576 — 32 бита; 288 — 64 бита
Мощность, ВТ	90
Кеш	1 уровень — 128 Кбайт + 64 Кбайт (команд+данных); 2 уровень - 4 Мбайт;3 уровень — 16 Мбайт
Число транзисторов, миллионов	3500

Сравнительная характеристика

ЦП	Кол-во ядер	GFlops	Частота, ГГц	Кэш L3, Мб	Техпроцесс, нм	ОЗУ тип	Max ОЗУ	Кол-во слотов ОЗУ
Core i7 975	4	50	3.3	8	45	DDR3/1066	24	3
Эльбрус-4С	4	50	0.8	0	65	DDR3/1600	48	3
2X Xeon x5677	4	104	3.5	12	32	DDR3/1333	288	9
Core i7-5960X	8	350	3.5	20	22 (2014 год)	DDR4/2400	128	4
Эльбрус-8СВ	8	288	1.5	16	28 (2020 год)	DDR4/2400	64	8
Xeon E7-8890 v4	24	844	2.2	60	14	DDR4/1600	3078	12

FL*oating-point* ***OP****erations per* ***S****econd – количество* операций с плавающей запятой в секунду.

Сравнительная характеристика

	Эльбрус-16СВ	**Эльбрус-8СВ**	**Core i7-2600**
Семейство ISA	VLIW	VLIW	CISC
Архитектура	e2k	e2k	x86-64
Микроархитектура	elbrus-v6	elbrus-v5	Sandy Bridge
Частота (МГц)	2000	1500	3400*
Ядра; Потоки	16	8	4; 8
Техпроцесс (нм)	16	28	32
TDP (Вт)	130	80-90	95
Тип ОЗУ	DDR4-3200 (2400)	DDR4-2400	DDR3-1333
Год	2021	2018	2011

Многопоточность
Hyper-Threading

Внутрипроцессорная многопоточность

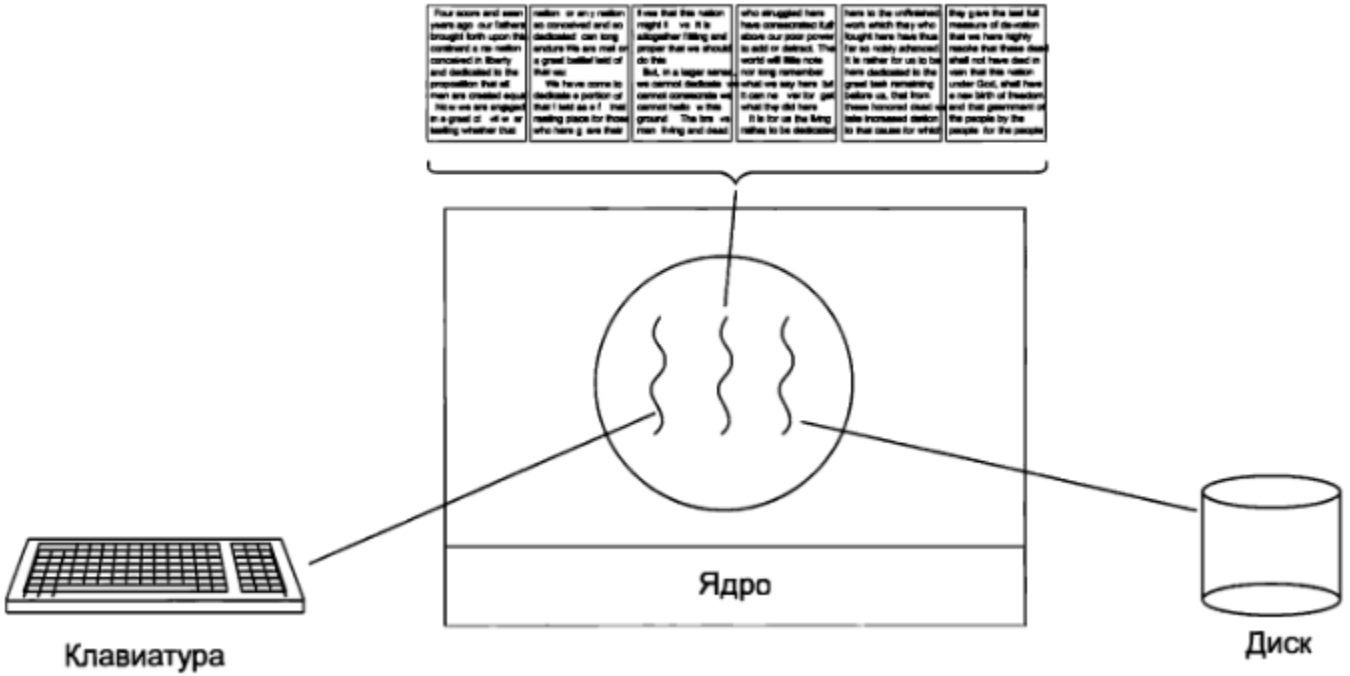
Причины простоя конвейера

- Обычный процессор в большинстве задач использует не более **70%** всей вычислительной мощности по причинам:
  - произошёл промах при обращении к кэшу процессора и требуется перезагрузка кэша;
  - выполнено неверное предсказание ветвления;
  - зависимость по данным;

Многопоточность

- Для реализации многозадачности ОС для каждой запускаемой программы выделяет ресурсы и создает специальное окружение – процесс, а в нем как минимум один поток.
- Программист разрабатывая программу может разбивать её на отдельные дополнительные программные потоки с помощью системных вызовов, исходя из логики работы программы.
- Для поочередного выполнения потоков, диспетчер потоков ОС прерывает текущий поток, сохраняет в памяти содержимое его регистров (регистровый контекст), загружает из памяти в регистры процессора регистровый контекст другого потока и запускает его. Т.е. в процессоре всегда находится один поток. **(Потоки переключает ОС)**
- Контекст потока:
  - Счетчик команд
  - Указатель стека
  - Регистры процессора, используемые потоком

Пример многопоточного приложения



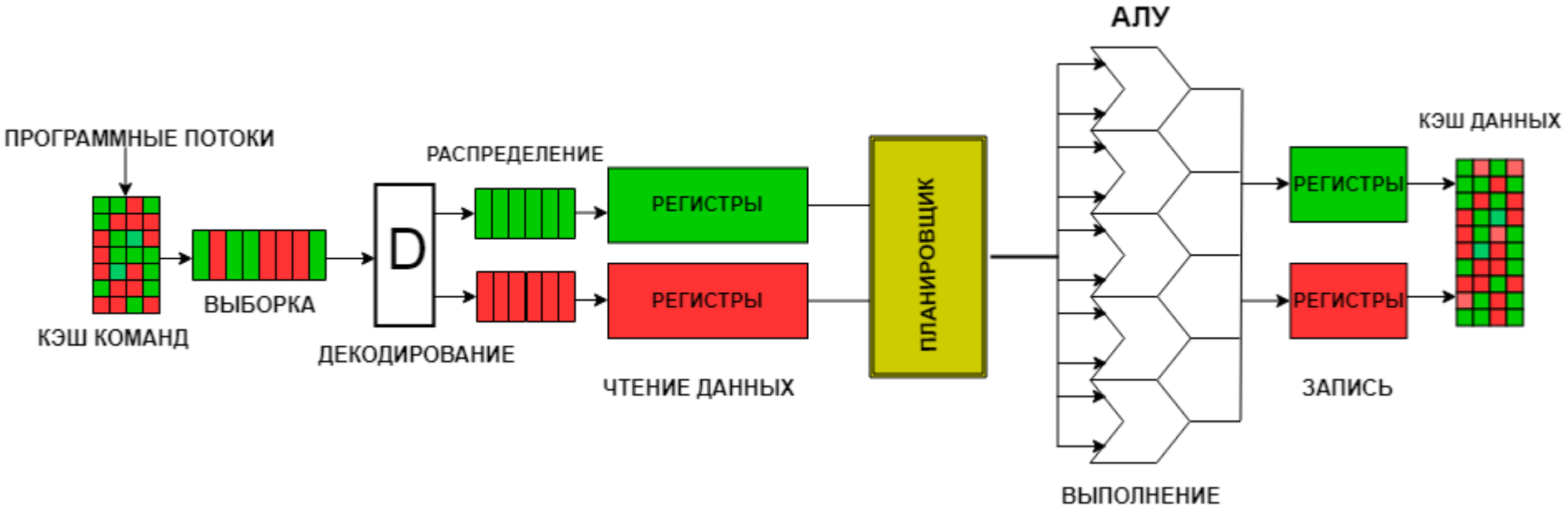
Текстовый редактор использующий три потока
– поток чтения данных с клавиатуры;
– поток вывода на экран ;
– поток сохранения на диск

Многопоточность в случае *Hyper-Threading*

- При технологии HT в процессоре **дублируются аппаратные ресурсы для хранения контекстов двух потоков.** (счетчик команд, указатель стека , РОН).
- ОС видит такой процессор как два **и загружает в него сразу** контексты двух потоков.
- В случае остановки одного потока, его регистровый контекст не выгружается, а остается в процессоре, то есть в процессоре находятся контексты сразу двух потоков.
- Если при выполнении первого потока произойдет, например, останов конвейера из за отсутствия данных в кэше, то процессор автоматически переключается на выполнение второго потока, пока первый подкачивает данные в кэш. **(Потоки переключает процессор аппаратно)**
- Аппаратное переключение происходит быстро и с гораздо меньшим потреблением ресурсов.

- При этом некоторые ресурсы потоки используют сообща (кэш память, регистры)

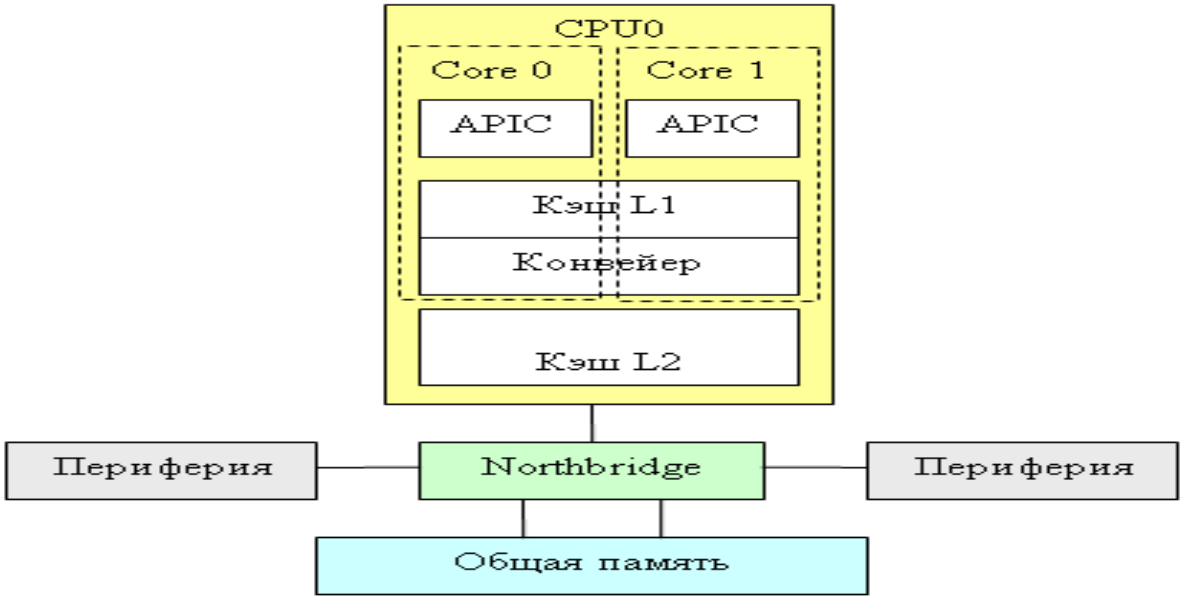
Разделение ресурсов при многопоточности



- HT увеличивает площадь кристалла процессора на 5%.
- При этом дает прирост производительности для многих приложений до 25%
- Общие ресурсы увеличивают количество промахов
- Программы должны поддерживать многопоточность

Технология *Hyper-Threading*

- Один физический процессор с HT можно представить как два логических процессора.



- Технология многопоточности использована впервые Intel в 2002 году в процессоре Intel Xeon.
- В процессорах Intel Core2 эта технология не использовалась.
- Появилась вновь в процессорах *Core i3*, *Core i7*, и некоторых *Core i5*

Технология Turbo Boost

- Технология позволяет процессорам самим динамически, на короткий промежуток времени, изменять тактовую частоту, тем самым, изменяя свою производительность.
- При этом процессор контролирует все параметры своей работы: напряжение, силу тока, температуру и т.д., не допуская сбоев и выхода из строя.

Модель	Тактовая частота, ГГц	Turbo Boost, ГГц
Core i7-7Y75	1.3	3.6
Core m7-6Y75	1.2	3.1
Core i5-7Y54	1.2	3.2