Нижегородский государственный университет
им. Н.И.Лобачевского

Факультет Вычислительной математики и кибернетики

Параллелизм как основа архитектуры ВС

Раздел 11

Архитектура EPIC

Кудин А.В., к.т.н.

Содержание

- Multiple-issue processors
- Explicit parallelism и архитектура EPIC
- Архитектура IA-64
- Конвейер Itanium 2
- Задержки в семействе Itanium
- Спекуляции в IA-64
- Организация предвыборки данных

  (данный вопрос наиболее актуален именно для VLIW процессоров)
- Трёхуровневая иерархия кеша Itanium 2
- Дефекты IA-64

Multiple-issue processors

Цель Multiple-issue processors состоит в поддержке исполнения нескольких инструкций за такт. Multiple-issue processors бывают двух видов: **Superscalar processors** и **VLIW** (Very Long Instruction Word) **processors**.

- **Superscalar processors** исполняют переменное число инструкций за такт, используя методы как статического (развёртка кода компилятором), так и динамического (алгоритм Томасуло, out-of-order execution) планирования.
- **VLIW processors**, напротив, исполняют фиксированное число независимых инструкций, сгруппированных в одну длинную инструкцию. В таком пакете инструкций параллелизм уровня инструкций обеспечивается статически на этапе компиляции. Компания Intel именует такую методику явного распараллеливания инструкций – **EPIC** – **Explicitly Parallel Instruction Computing**.

Explicit parallelism

- Суть **явного параллелизма** заключается в том, что распределение команд между исполнительными узлами производится не процессором в ходе выполнения программы (динамически), а компилятором при формировании машинного кода (статически).
- Таким образом, схемотехника EPIC-процессора существенно упрощается (по сложности он сравним с суперскалярным процессором без поддержки неупорядоченного исполнения команд).
- Кроме того, в компиляторах алгоритмы выбора порядка исполнения команд могут быть существенно сложнее и эффективнее, чем алгоритмы аппаратного планирования инструкций (так как решение необходимо принимать в течение наносекунд).

Multiple-issue processors

Пять ключевых подходов к организации Multiple-issue processors

Common name	Issue structure	Hazard detection	Scheduling	Distinguishing characteristic	Examples
Superscalar (static)	dynamic	hardware	static	in-order execution	Sun UltraSPARC II/ III
Superscalar (dynamic)	dynamic	hardware	dynamic	some out-of-order execution	HP PA 8500, IBM RS64 III
Superscalar (speculative)	dynamic	hardware	dynamic with speculation	out-of-order execution with speculation	Pentium III/4, MIPS R10K, Alpha 21264
VLIW/LIW	static	software	static	no hazards between issue packets	Trimedia, i860
EPIC	mostly static	mostly software	mostly static	explicit dependences marked by compiler	Itanium

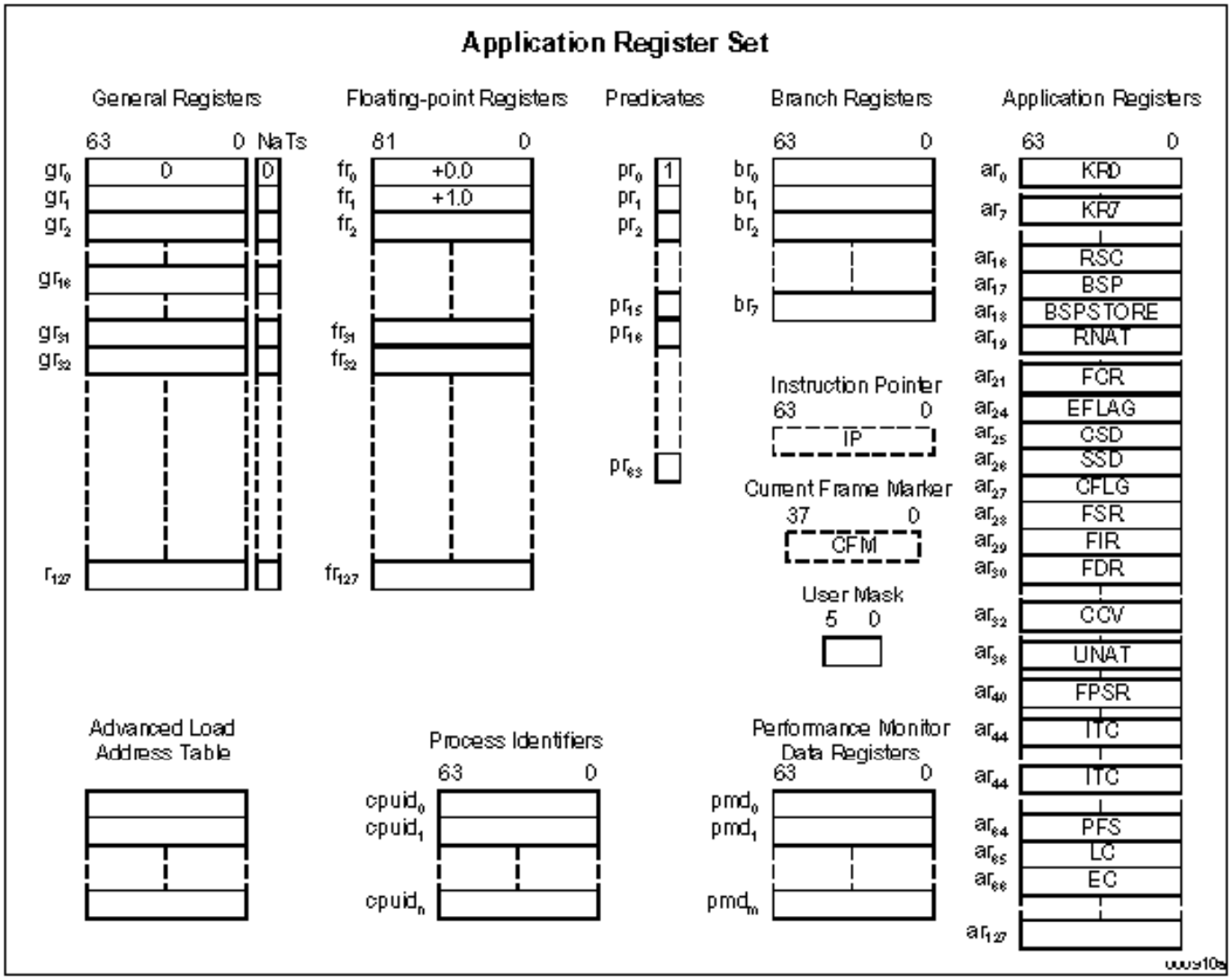
Архитектура EPIC

- Самая перспективная модификация VLIW (из существующих)
- Базовые принципы были разработаны в университете Иллинойса в начале 1990-х годов
- Наиболее важные особенности:
  - поддержка явно выделенного компилятором параллелизма
  - наличие большого регистрового файла
  - наличие предикатных регистров (и предикатного исполнения)
  - спекулятивная загрузка данных
  - аппаратная поддержка программной конвейеризации
  - используется «стек» регистров и регистровые окна
  - для предсказания трассы инструкций используется поддержка компилятора
  - поддержка инструкций циклического выполнения команд без потерь времени на инструкции циклического выполнения
- Реализация: IA-64 (используется в семействе Itanium)

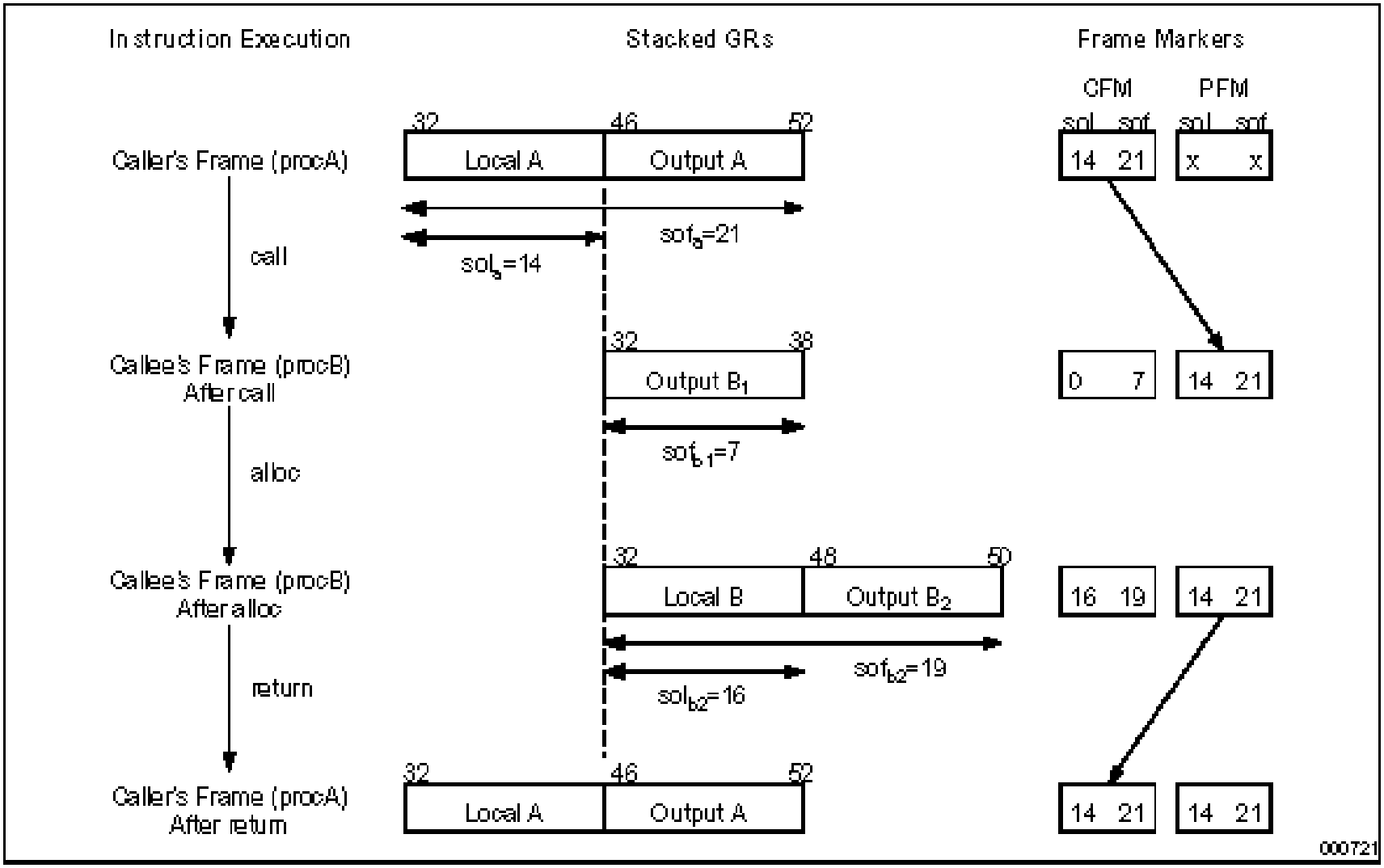
Общие сведения об IA-64

- 15 исполнительных устройств
- Исполнение до 20 операций одновременно
- 3 команды в каждой длинной инструкции, процессор способен выполнять более одной длинной инструкции за такт
- Мультимедийные команды
- Поддержка системы команд x86 со всеми расширениями
- 128 целочисленных регистров (96 регистров видны в качестве регистрового окна)
- 128 регистров с плавающей запятой (шириной 80 бит)
- 64 однобитовых предикатных регистра
- 8 регистров перехода
- Адресация до 16 Tb RAM

Регистры общего назначения IA-64



Регистровый стек IA-64



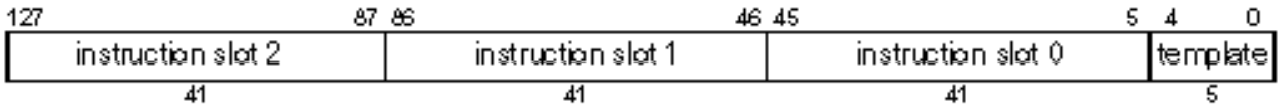
Регистровый стек IA-64

- 128 регистров (целых и с плавающей точкой) разделены на два множества:
  - 32 регистра (0÷31) – статичны и видны всем функциям,
  - 96 регистров организованы в виде «регистрового окна».
- При создании фрейма стека компилятор статически рассчитывает количество необходимой локальной регистровой памяти (не более 96 регистров).
- Локальные переменные процедуры при вызове другой процедуры могут быть сохранены посредством поворота стека таким образом, что первым локальным регистром для вызванной процедуры станет 32-й регистр.
- Регистры, которые ранее были локальными у вызывающей процедуры, либо «уходят в тень», либо сохраняются в кеше.

Инструкции IA-64

- 6 типов команд
- Каждый тип может выполняться на одном или нескольких функциональных устройствах
- Длинное командное слово (128 бит) – три RISC команды (каждая 40 бит) плюс пяти битный шаблон, определяющий для данной длинной инструкции расписание команд по функциональным устройствам
- Возможность разбиения длинных инструкций (с целью сокращения объёма кода) посредством архитектурных "остановов", регламентирующих передачу потока команд на следующий такт (из-за зависимостей по данным)

Instruction Type	Description	Execution Unit Type
A	Integer ALU	I-unit or M-unit
I	Non-ALU integer	I-unit
M	Memory	M-unit
F	Floating-point	F-unit
B	Branch	B-unit
L+X	Extended	I-unit/B-unit



Слоты выдачи в IA-64

Execution Unit Slot	Instruction type	Instruction Description	Example Instructions
I-unit	A	Integer ALU	add, subtract, and, or, compare
	I	Non-ALU Integer	integer and multimedia shifts, bit tests, moves
M-unit	A	Integer ALU	add, subtract, and, or, compare
	M	Memory access	Loads and stores for integer/FP registers
F-unit	F	Floating point	Floating point instructions.
B-unit	B	Branches	Conditional branches, calls, loop branches
L+X	L+X	Extended	Extended immediates, stops and no-ops.

Форматы выдачи в IA-64

Template	Slot 0	Slot 1	Slot 2
0	M	I	I
1	M	I	I
2	M	I	I
3	M	I	I
4	M	L	X
5	M	L	X
8	M	M	I
9	M	M	I
10	M	M	I
11	M	M	I
12	M	F	I
13	M	F	I
14	M	M	F
15	M	M	F
16	M	I	B
17	M	I	B
18	M	B	B
19	M	B	B
22	B	B	B
23	B	B	B
24	M	M	B
25	M	M	B
28	M	F	B
29	M	F	B

Пример выдачи в IA-64

Bundle template	Slot 0	Slot 1	Slot 2	Execute cycle (1 bundle / cycle)
9: M M I	L.D F0,0(R1)	L.D F6,-8(R1)		1
14: M M F	L.D F10,-16(R1)	L.D F14,-24(R1)	ADD.D F4,F0,F2	3
15: M M F	L.D F18,-32(R1)	L.D F22,-40(R1)	ADD.D F8,F6,F2	4
15: M M F	L.D F26,-48(R1)	S.D F4,0(R1)	ADD.D F12,F10,F2	6
15: M M F	S.D F8,-8(R1)	S.D F12,-16(R1)	ADD.D F16,F14,F2	9
15: M M F	S.D F16,-24(R1)		ADD.D F20,F18,F2	12
15: M M F	S.D F20,-32(R1)		ADD.D F24,F22,F2	15
15: M M F	S.D F24,-40(R1)		ADD.D F28,F26,F2	18
12: M M F	S.D F28,-48(R1)	DADDUI R1,R1,#-56	BNE R1,R2,Loop	21

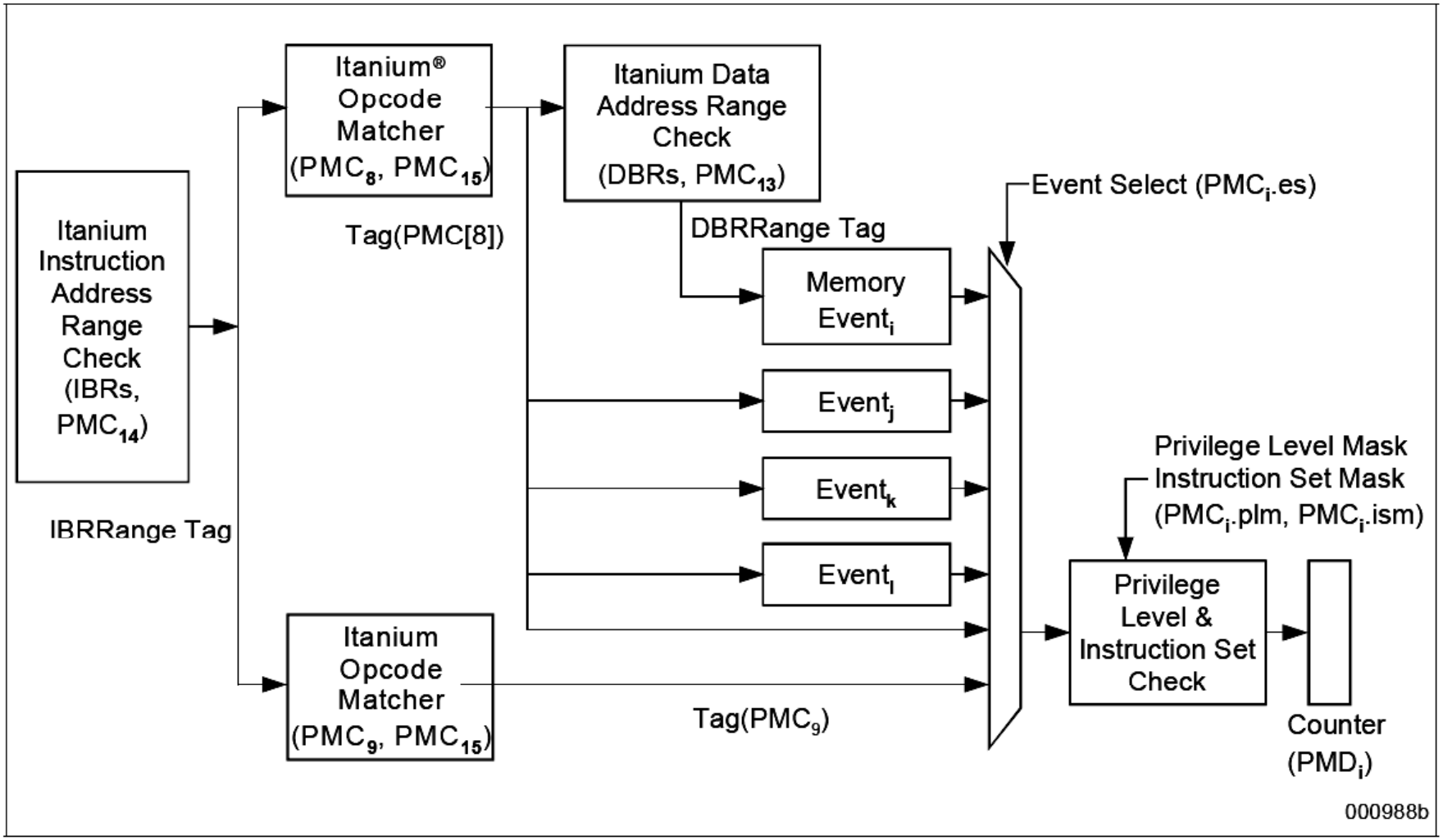
a. The code scheduled to minimize the number of bundles.

Bundle template	Slot 0	Slot 1	Slot 2	Execute cycle (1 bundle / cycle)
8: M M I	L.D F0,0(R1)	L.D F6,-8(R1)		1
9: M M I	L.D F10,-16(R1)	L.D F14,-24(R1)		2
14: M M F	L.D F18,-32(R1)	L.D F22,-40(R1)	ADD.D F4,F0,F2	3
14: M M F	L.D F26,-48(R1)		ADD.D F8,F6,F2	4
15: M M F			ADD.D F12,F10,F2	5
14: M M F		S.D F4,0(R1)	ADD.D F16,F14,F2	6
14: M M F		S.D F8,-8(R1)	ADD.D F20,F18,F2	7
15: M M F		S.D F12,-16(R1)	ADD.D F24,F22,F2	8
14: M M F		S.D F16,-24(R1)	ADD.D F28,F26,F2	9
9: M M I	S.D F20,-32(R1)	S.D F24,-40(R1)		10
8: M M I	S.D F28,-48(R1)	DADDUI R1,R1,#-56	BNE R1,R2,Loop	11

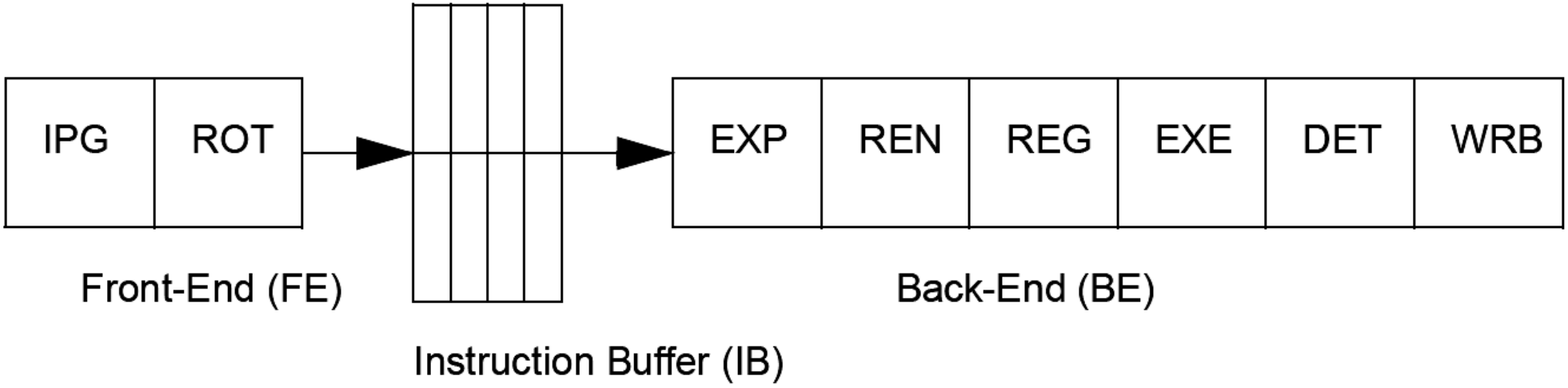
b. The code scheduled to minimize the number of cycles assuming one bundle executed per cycle.

семикратная развёртка цикла x[i] = x[i] + s

Механизм маркировки инструкций Itanium 2



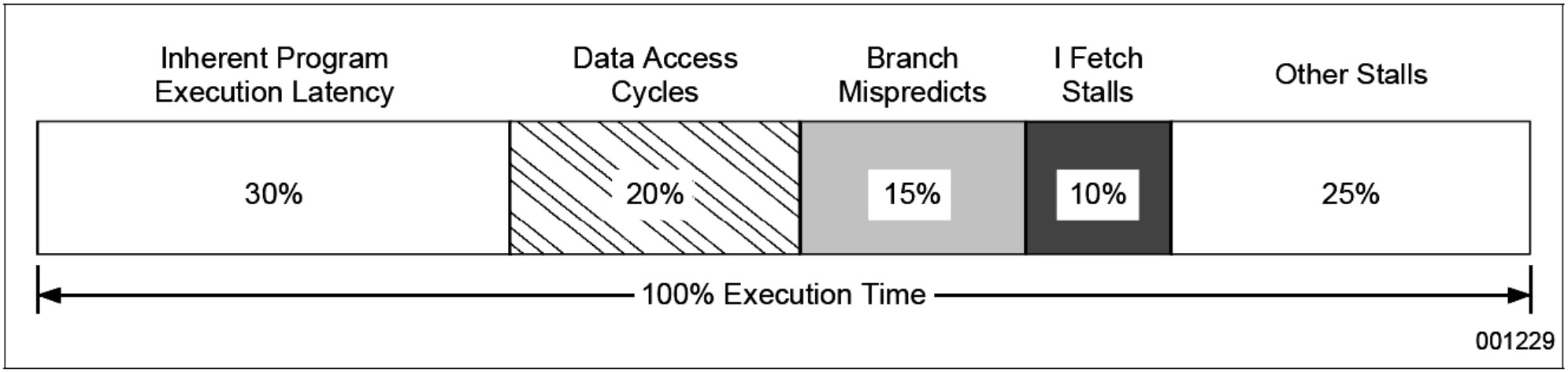
Конвейер Itanium 2



FPU Micro-Pipeline

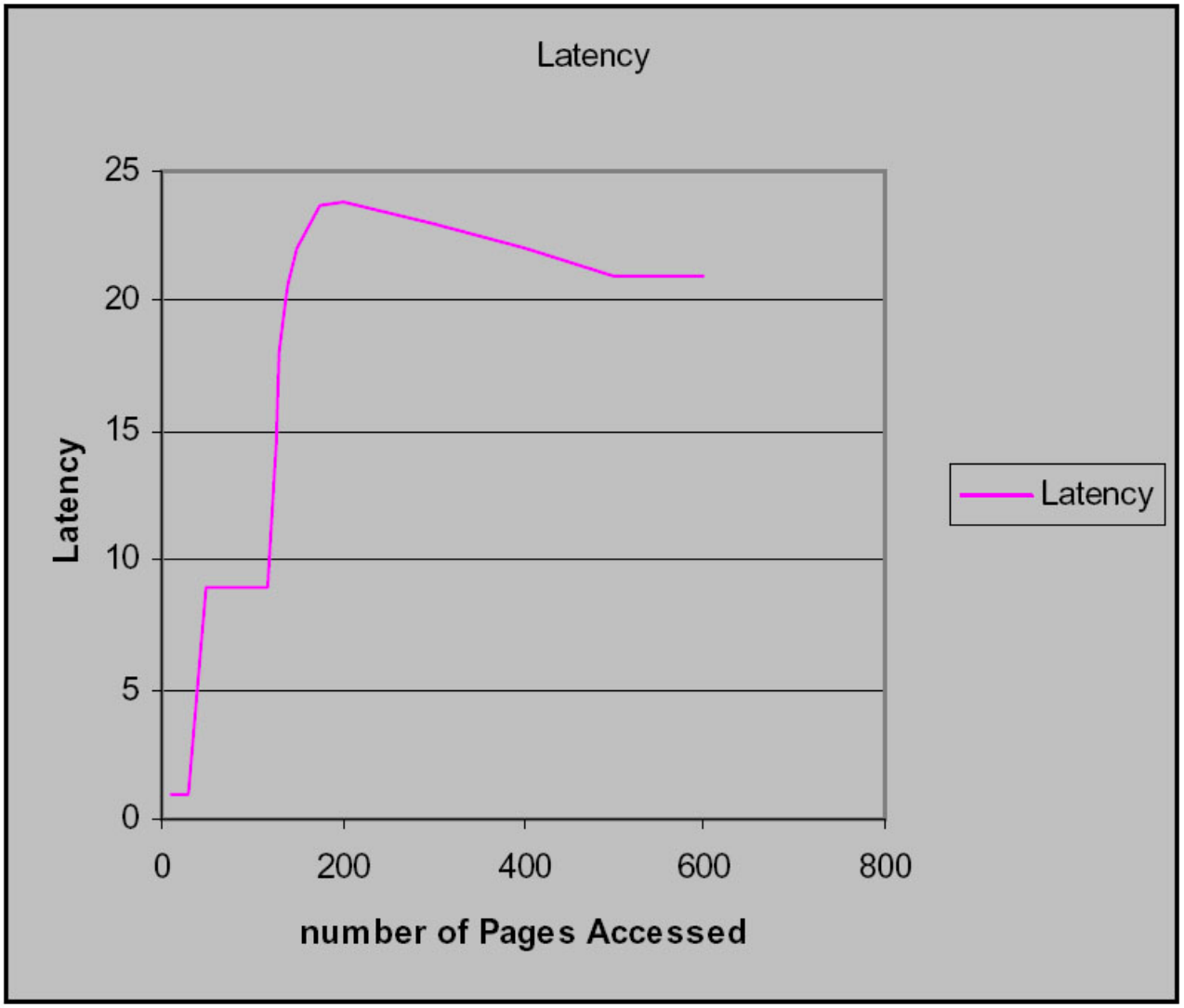
Core Pipeline	REG	EXE	DET	WRB		
FPU Pipeline		FP1	FP2	FP3	FP4	FWB

Задержки в семействе Itanium



Instruction	Latency
Integer load	1
Floating point load	9
Correctly predicted taken branch	0-3
Mispredicted branch	9
Integer ALU operations	0
FP arithmetic	4

Задержка доступа к ОП



Спекуляции по командам в IA-64

Крайне актуальна проблема разрыва производительности (между CPU и RAM), так как в случае задержки загрузки конвейер быстро блокируется, ибо команд, которые можно выполнить без нарушения зависимости по данным, обычно оказывается очень мало.

Решение – спекуляция по командам:

- Выдача команд загрузки данных задолго до инструкций, использующих эти данные (с вынесением сквозь точки ветвления).
- При достижении потоком команд точки, в которой необходимо использовать загружаемые данные, выполняется специальная инструкция, проверяющая наличие исключений в процессе загрузки (например, какая-либо инструкция произвела запись по данному адресу). Если такое исключение произошло, то выполняется специальный восстановительный код, который попросту перезагружает значение в РОН.

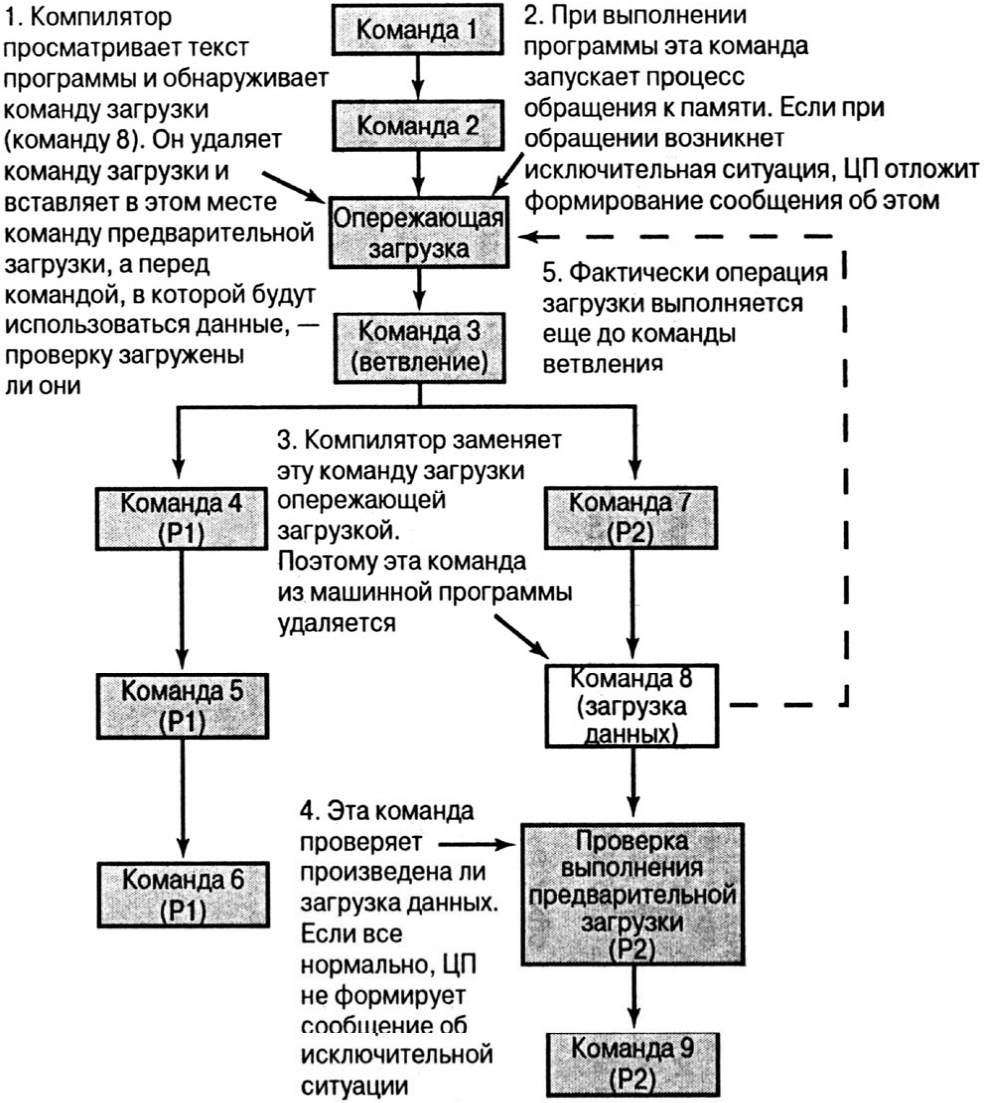
Спекуляции по данным в IA-64

Если компилятор не имеет достаточной информации для анализа потока данных, он не имеет возможности определить, перекрываются ли области памяти, адресуемые парой указателей. Спекуляция по данным состоит в переупорядочивании инструкций косвенной адресации. Если после перенесённой вверх инструкции загрузки произведена запись, затрагивающая загружаемое содержимое, то производится вызов кода восстановления, перезагружающего значение этого содержимого.

Решение – спекуляция по данным:

- Все РОН имеют дополнительное одно битовое поле, называемое NaT.
- Спекуляция состоит из двух частей: из команды, начинающей спекулятивную загрузку, и из команды, производящей проверку на наличие исключений.
- Команда, начинающая загрузку, очищает бит NaT. В случае, если происходит интерференция обращений к памяти, генерируются события, проявляющиеся в установке бита NaT в единицу. Команда, производящая проверку на наличие исключений, тестирует бит NaT и, если он установлен, производит вызов восстанавливающего кода.

Предвыборка данных в IA-64



Предикатное исполнение в IA-64

Компилятор может переставить команды, сдвоив для параллельного выполнения 4-ю и 7-ю команды, 5-ю и 8-ю, 6-ю и 9-ю

Команда 1	Команда 2	Команда 3
Команда 4	Команда 7	Команда 5
Команда 8	Команда 6	Команда 9

Организация предвыборки данных

Простейшее определение дистанции предвыборки:

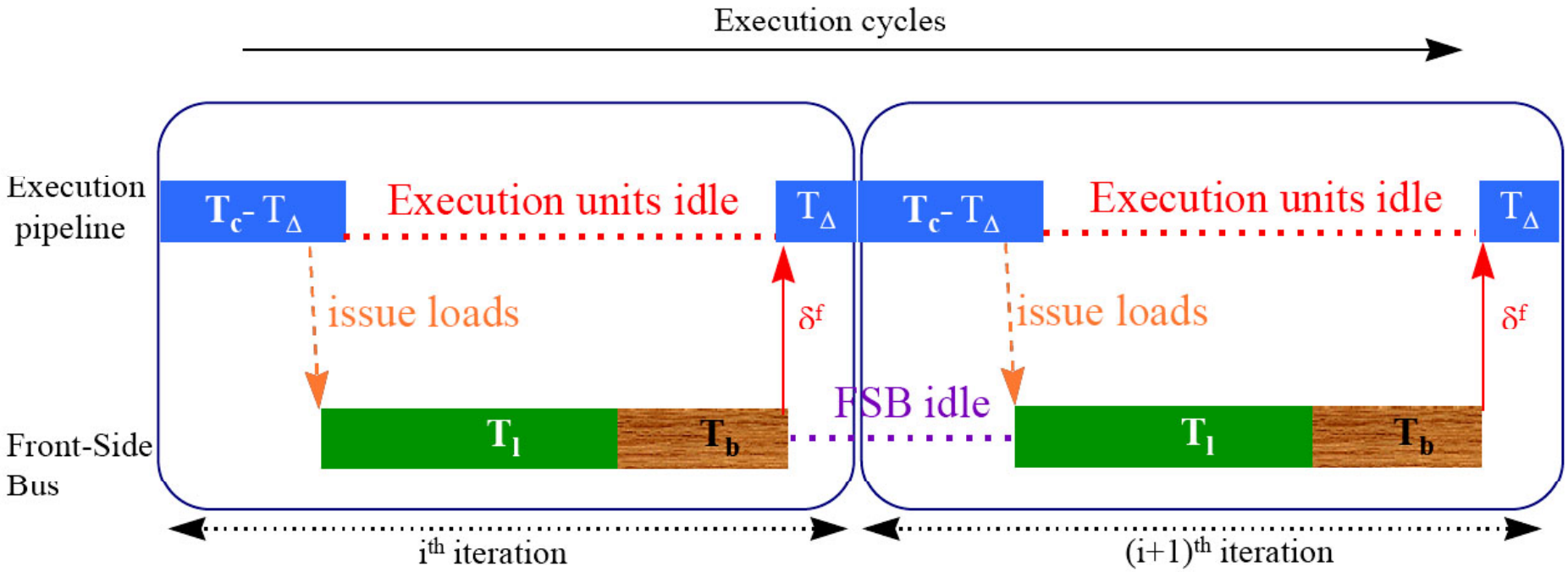
$$psd = \left| \frac{Nlookup + Nxfer \cdot (N_{pref} + N_{st})}{CPI \cdot N_{inst}} \right|$$

where

`psd`	is prefetch scheduling distance.
`Nlookup`	is the number of clocks for lookup latency. This parameter is system-dependent. The type of memory used and the chipset implementation affect its value.
`Nxfer`	is the number of clocks to transfer a cache-line. This parameter is implementation-dependent.
N_{pref} and N_{st}	are the numbers of cache lines to be prefetched and stored.
`CPI`	is the number of clocks per instruction. This parameter is implementation-dependent.
N_{inst}	is the number of instructions in the scope of one loop iteration.

Организация предвыборки данных

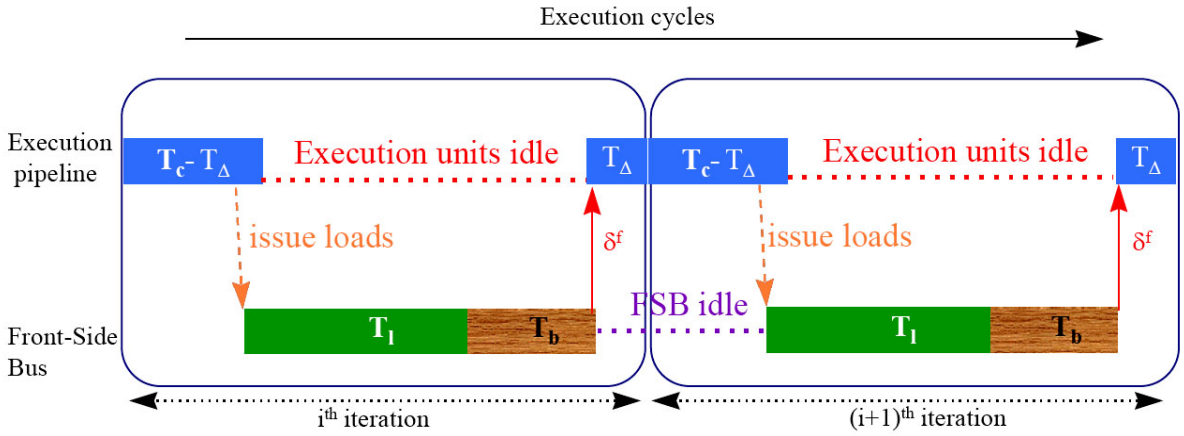
В случае без предвыборки:



T_c – computation latency per iteration with prefetch caches
T_l – memory leadoff latency including cache miss latency, chip set latency, bus arbitration, etc.
T_b – data transfer latency (number of lines per iteration × line burst latency)

Организация предвыборки данных

В случае без предвыборки:



здесь **iteration latency (il) = $T_c + T_l + T_b$**

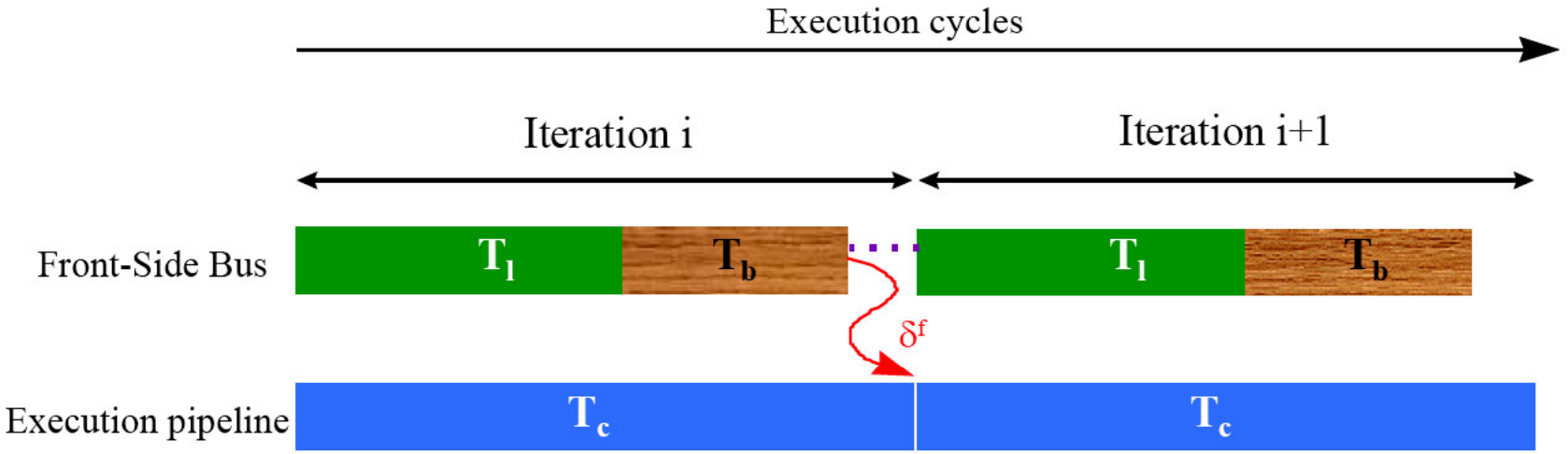
плохой микроуровневый параллелизм

в идеальном варианте **il = max { T_c , T_b }**

T_c – computation latency per iteration with prefetch caches
T_l – memory leadoff latency including cache miss latency, chip set latency, bus arbitration, etc.
T_b – data transfer latency (number of lines per iteration × line burst latency)

Организация предвыборки данных

В случае $T_c > T_l + T_b$:



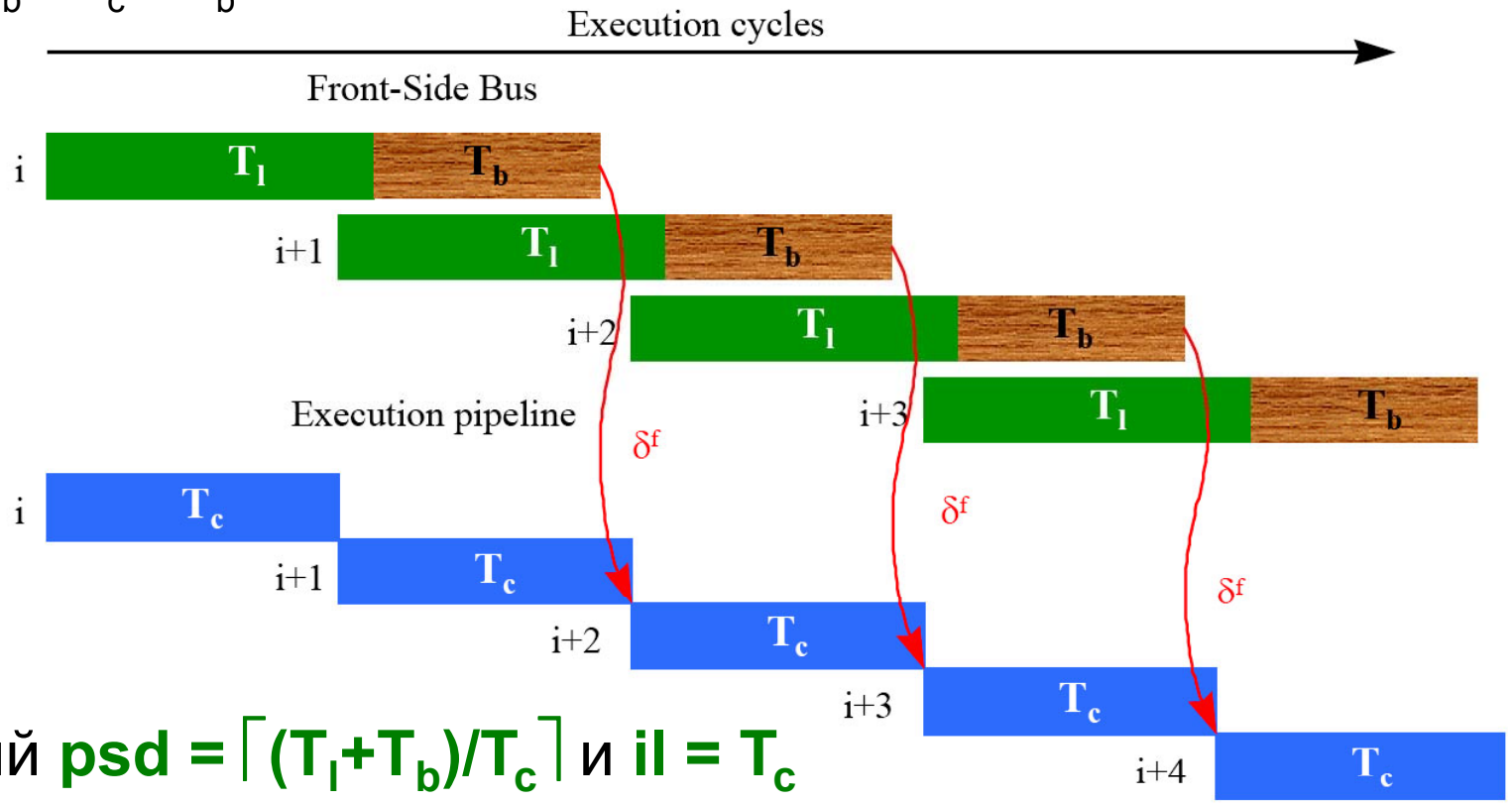
оптимальный **psd ≡ 1** и **il = T_c**

доступ к памяти осуществляется в фоновом режиме

T_c – computation latency per iteration with prefetch caches
T_l – memory leadoff latency including cache miss latency, chip set latency, bus arbitration, etc.
T_b – data transfer latency (number of lines per iteration × line burst latency)

Организация предвыборки данных

В случае $T_l + T_b > T_c > T_b$:



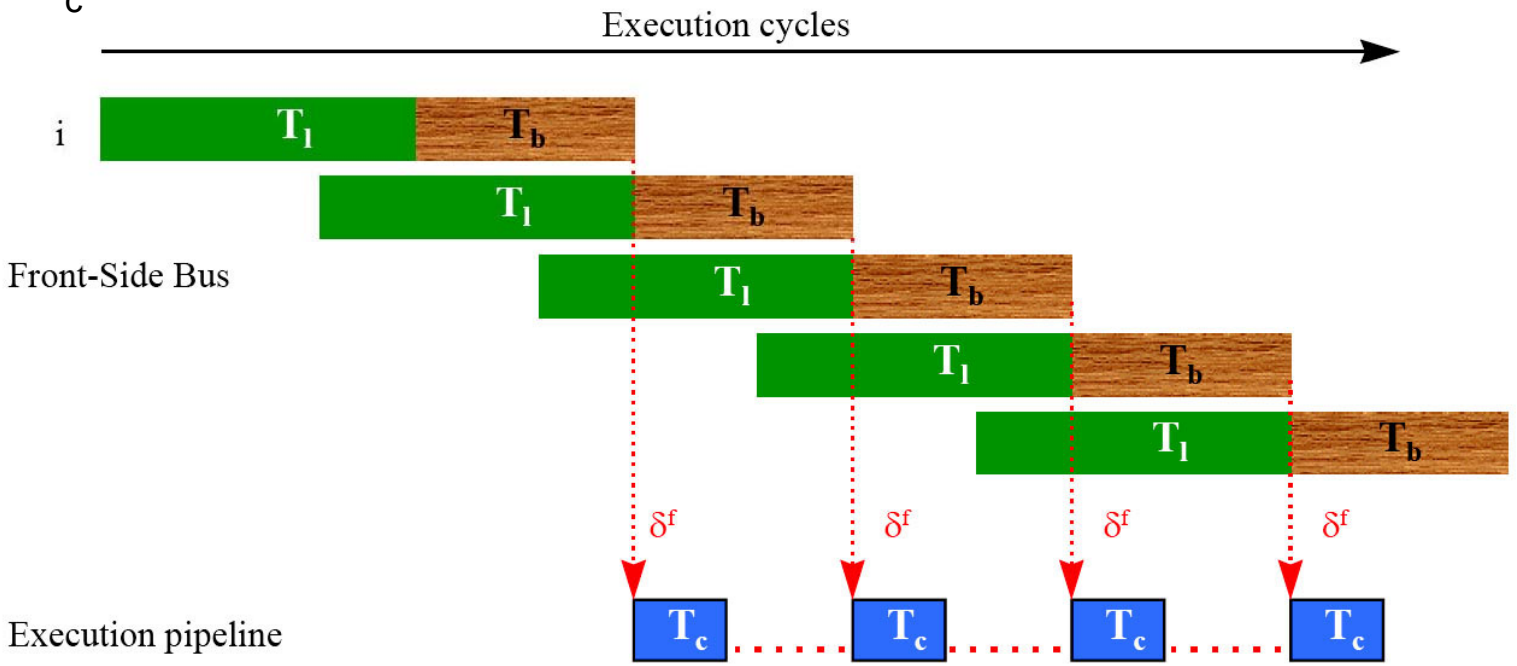
оптимальный **psd = $\lceil (T_l+T_b)/T_c \rceil$** и **il = T_c**

доступ к памяти осуществляется в фоновом режиме

T_c – computation latency per iteration with prefetch caches
T_l – memory leadoff latency including cache miss latency, chip set latency, bus arbitration, etc.
T_b – data transfer latency (number of lines per iteration × line burst latency)

Организация предвыборки данных

В случае $T_b > T_c$:

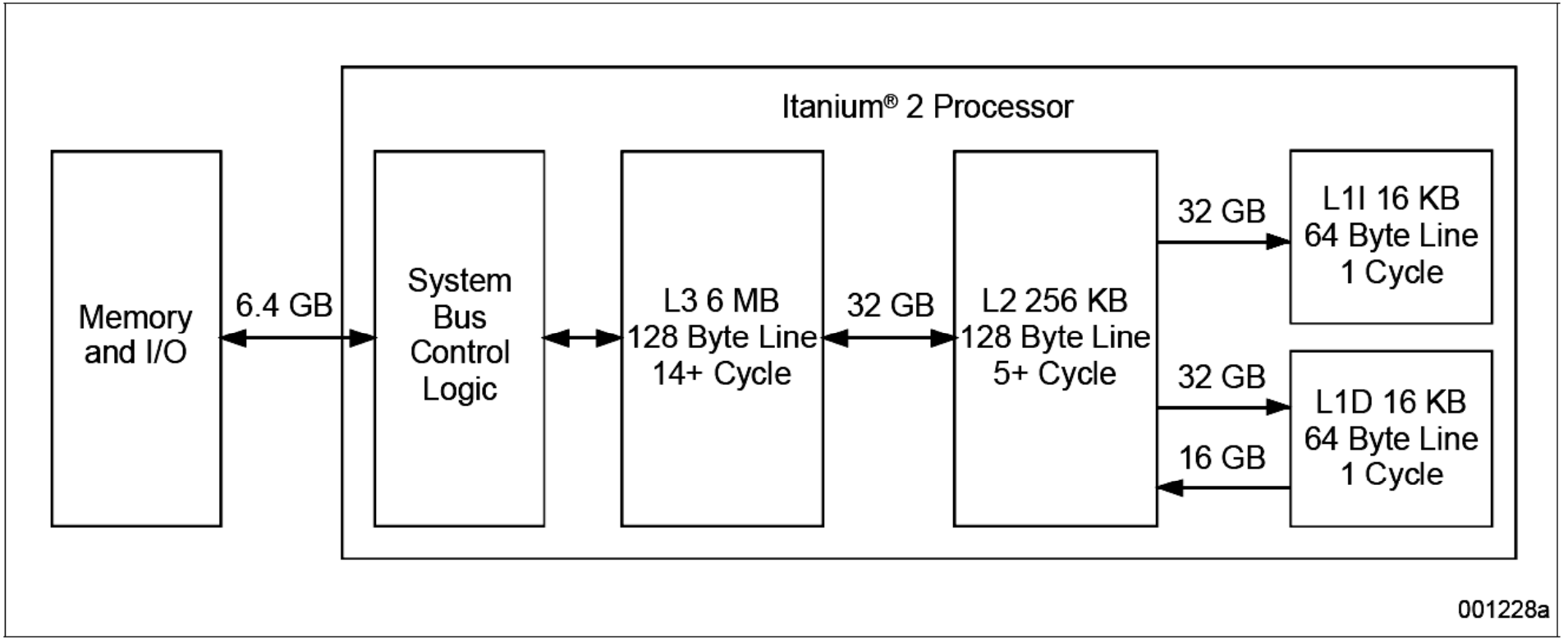


оптимальный **psd = 1+ $\lceil T_l/T_b \rceil$** и **il = T_b**

вычисления производятся в фоновом режиме

T_c – computation latency per iteration with prefetch caches
T_l – memory leadoff latency including cache miss latency, chip set latency, bus arbitration, etc.
T_b – data transfer latency (number of lines per iteration × line burst latency)

Трёхуровневая иерархия кеша Itanium 2



Трёхуровневая иерархия кеша Itanium 2

	L1I	L1D	L2	L3
Size	16 KB	16 KB	256 KB	3 MB or 1.5 MB
Associativity	4-way	4-way	8-way	12-way
Line size	64 Bytes	64 Bytes	128 Bytes	128 Bytes
Latency	1 cycle	1 cycle	Minimum 5 cycles integer load use. Minimum 6 cycles floating-point load use. 7 cycles with 6 cycle stall penalty in *ROT* stage for instruction load use.	Minimum 12 cycles load use.
Tag Read Bandwidth	2 / cycle	4 / cycle	4 / cycle	1 / cycle
Data Read Bandwidth	1 X 32B / cycle	2 X 8B / cycle	2 x 16B / cycle + 2x 8B[1]	1 x 32B / cycle
Data banks	n/a	8 bytes/bank (store only)	16 bytes/bank	n/a
Write Bandwidth	n/a	2 x 8B / cycle	4 x 16B / cycle	1 x 32B / cycle
Fill Bandwidth	64 bytes assembly 2 cycles write - 1 cycle	64 bytes assembly 2 cycles write - 1 cycle	128 bytes assembly 4 cycles write - 1 cycle	128 bytes in 4 cycles
Outstanding Misses	7 prefetches	8 unique lines	16 unique lines	22 (16 read shared with L2, 6 write)
Line Size	64 Bytes	64 Bytes	128 Bytes	128 Bytes

Дефекты IA-64

- В IA-64 последовательность команд дистрибутива разделяется на группы, состоящие из независимых команд, которые могут исполняться одновременно только при наличии реально имеющихся ресурсов определенной модели.
- Дистрибутив представляет собой код, оптимизированный только для одной конкретной модели. IA-64 исключил из суперскалярного подхода механизмы адаптации к конкретной модели без какой-либо замены.
- Авторы IA-64 либо предположили, что все будущие модели будут основываться на одной и той же схеме ресурсов (что нереально), либо решили, что всех устроит большая потеря эффективности при работе программы, скомпилированной для той или иной модели. Фактически это означает, что компьютеры IA-64 будут (обычно) использовать код, оптимизированный для какой-либо предыдущей модели с соответствующей потерей эффективности.
- В целях обеспечения совместимости IA-64 предлагает реализацию обеих (x86 и IA-64) систем команд в одном микропроцессоре ("два в одном"). Помня о плохой адаптируемости IA-64, естественное обобщение решения проблемы совместимости в IA-64 для будущих моделей будет состоять во включении всех предыдущих моделей в настоящий микропроцессор.

Вопросы для обсуждения

- Сопоставьте IA-32 и IA-64.
- Что произошло с механизмом переименования регистров в IA-64?
- Чем отличается архитектура EPIC от классической архитектуры VLIW?
- Разные процессоры IA-64 могут содержать разное количество функциональных устройств, оставаясь при этом совместимыми по коду. Укажите недостатки такой масштабируемости IA-64.
- Сопоставьте встроенный в IA-64 механизм предвыборки данных с механизмом предвыборки, используемым контроллером RAM (северным мостом).
- При каких значениях T_c, T_l, T_b в целочисленной программе и FSB и конвейер Itanium 2 будут загружены на 100%?
- Укажите факторы, сдерживающие распространение архитектуры EPIC.

Следующая тема

- **Code Morphing**