

III. Организация памяти ЭВМ

- Классификация памяти ЭВМ. Характеристики памяти.
- Методы организации доступа в запоминающие устройства.
- Состав, устройство и принцип действия основной памяти.
- Статические и динамические запоминающие устройства.
- Постоянные запоминающие устройства (ПЗУ).
- Организация кэш-памяти.
- Виртуальная память.

Памятью ЭВМ называется совокупность устройств, служащих для запоминания, хранения и выдачи информации.

Характеристики памяти ЭВМ:

- Назначение.
- Информационная емкость.
- Информационная емкость читаемого слова.
- Способ доступа.
- Быстродействие.
- Физический способ хранения информации.

Классификация запоминающих устройств по способу доступа.

- Адресные ЗУ

Постоянные ЗУ, ПЗУ (ROM)

ЗУ с произвольным доступом (RAM)

- Ассоциативные ЗУ

Полностью ассоциативные ЗУ

Ассоциативные ЗУ с прямым размещением

Наборно-ассоциативные ЗУ

- Последовательные ЗУ

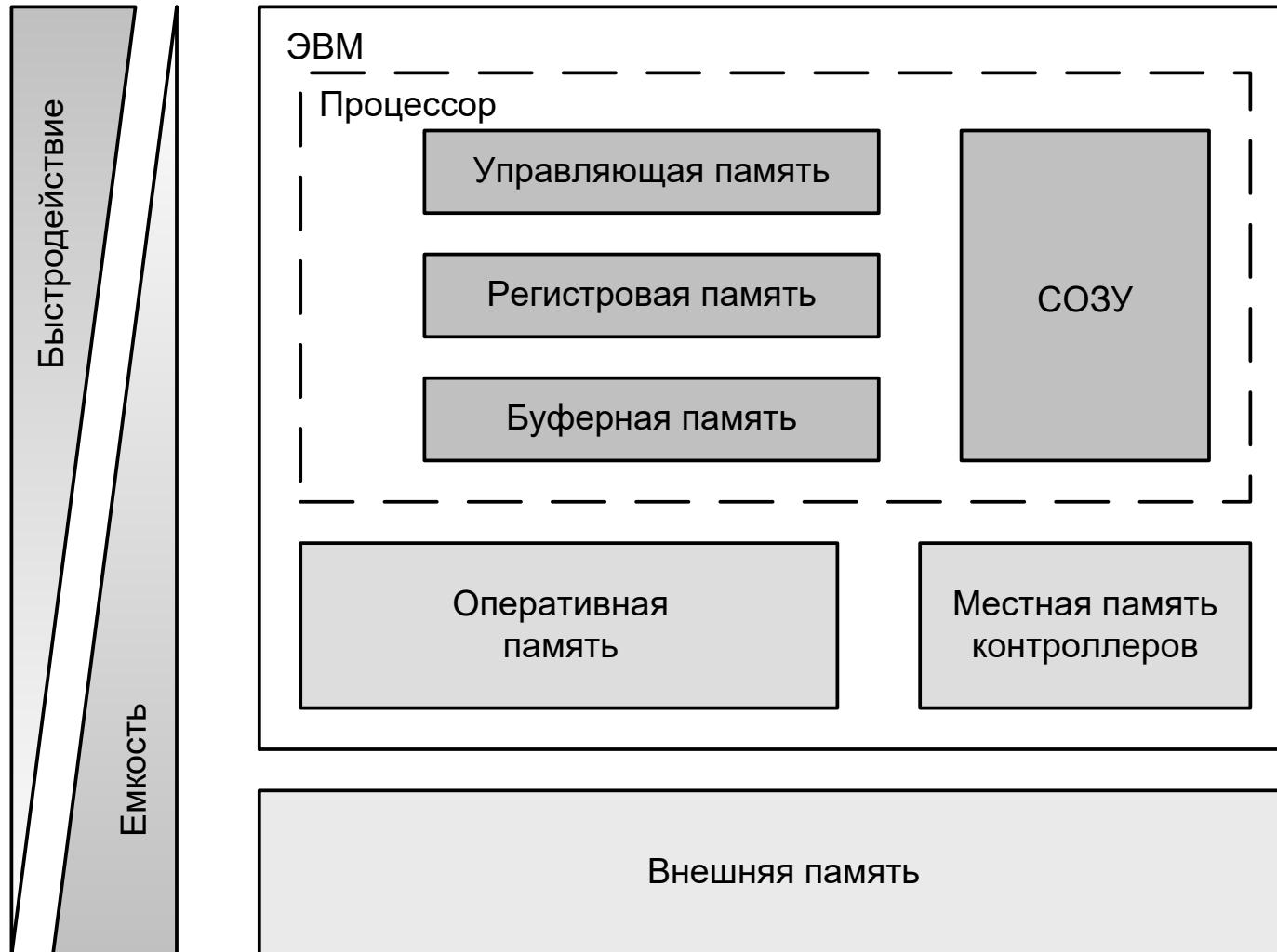
FIFO

LIFO

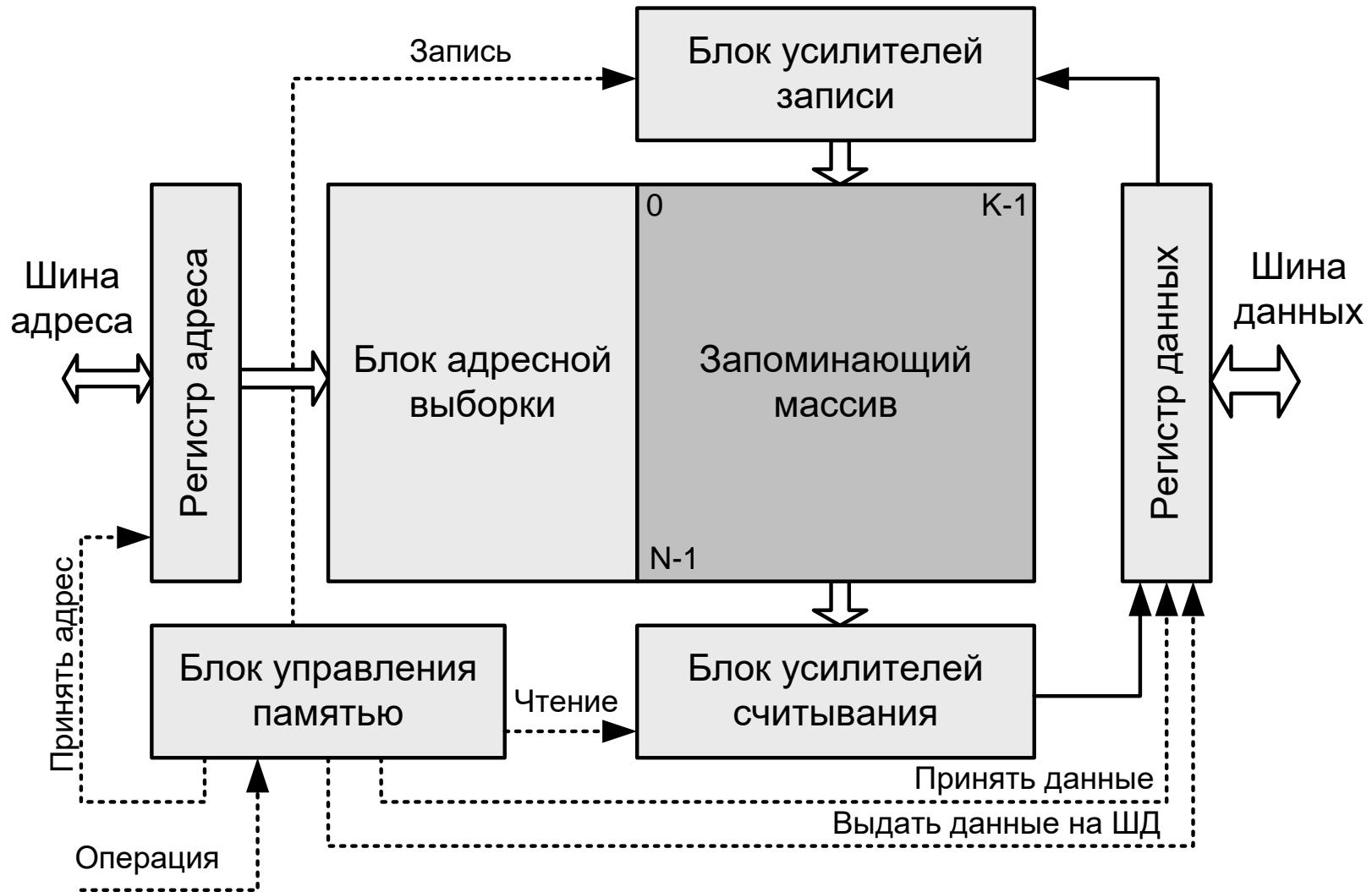
Файловые

Циклические

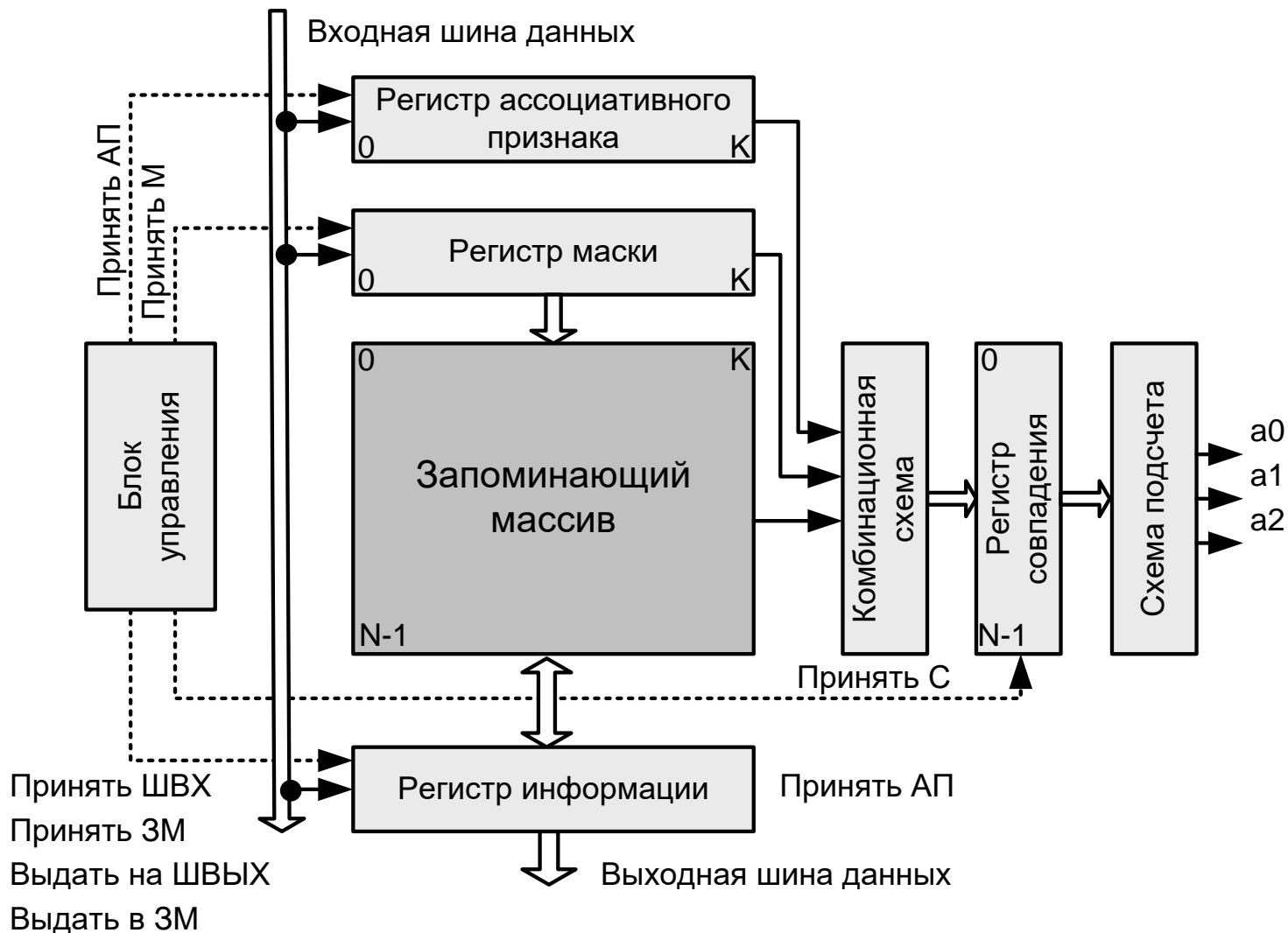
Классификация запоминающих устройств по назначению.



Обобщенная схема адресного ЗУ

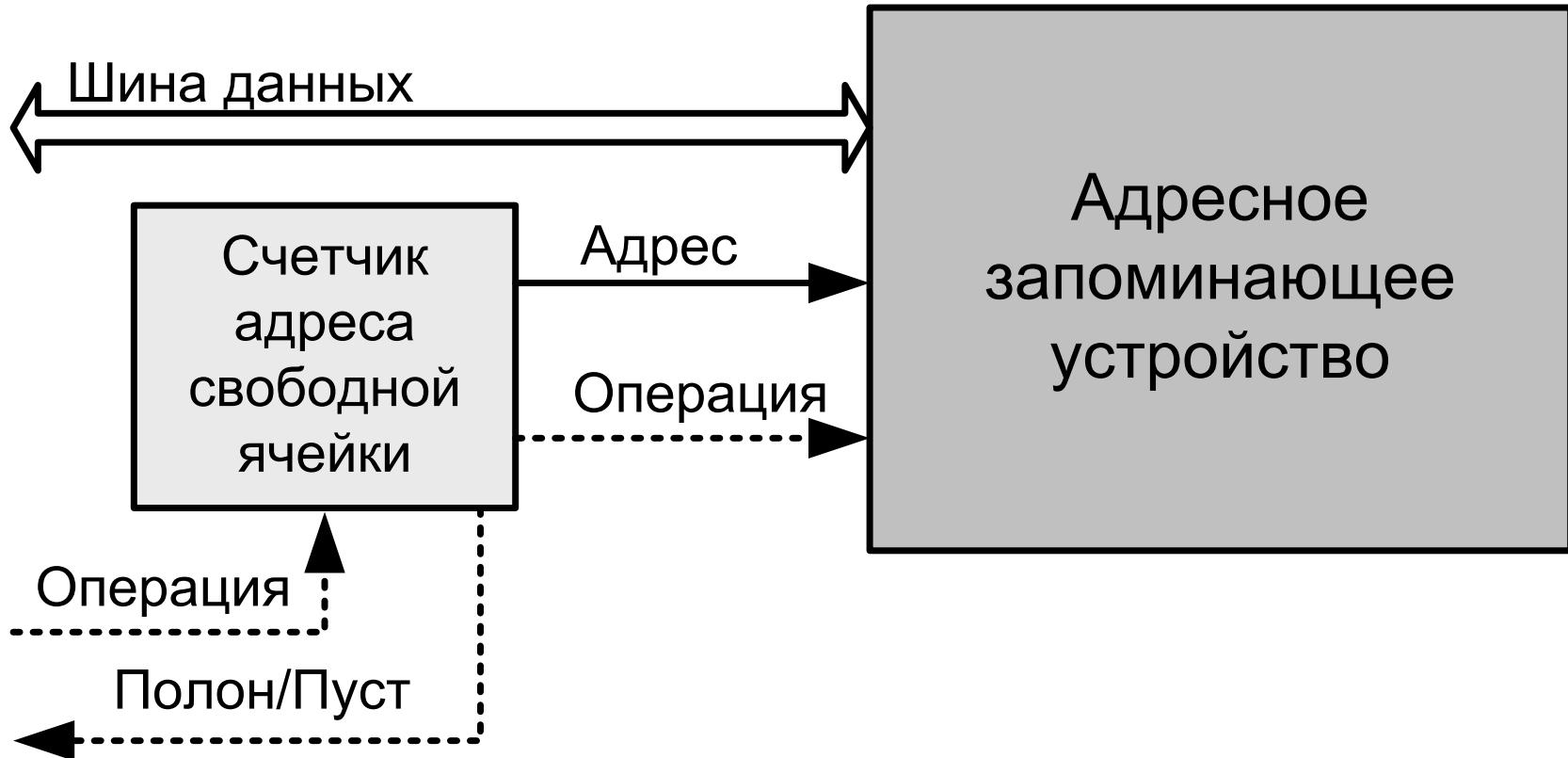


Обобщенная схема ассоциативного ЗУ

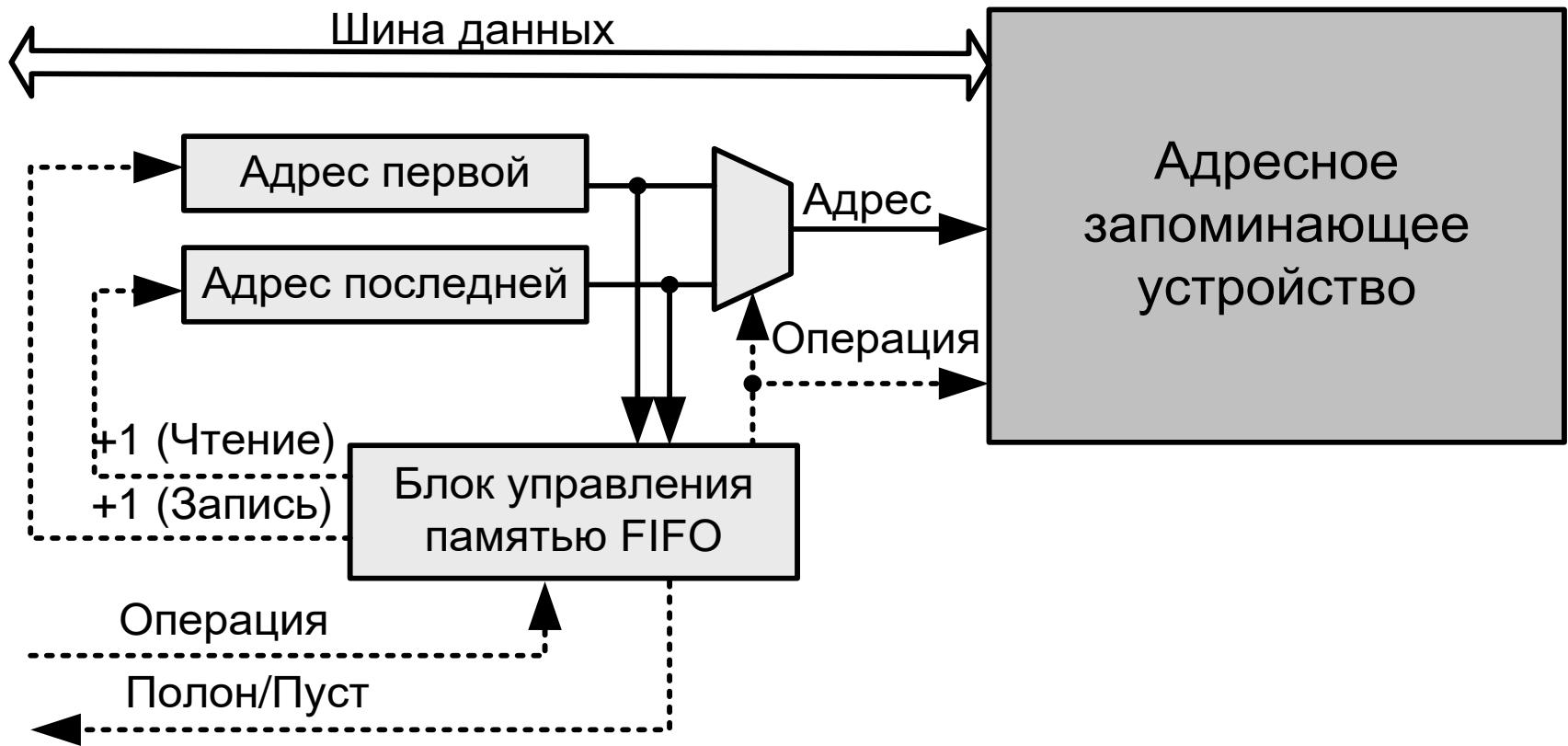


Обобщенная схема последовательного ЗУ

Стек (память типа LIFO)



Буфер (память типа FIFO)



Адресные запоминающие устройства

Постоянные ЗУ, ПЗУ (ROM)

МПЗУ (MROM)

ППЗУ (PROM)

РПЗУ-УФ (EPROM)

ОПРПЗУ-УФ (EPROM-OTP)

РПЗУ-ЭС (EEPROM)

FLASH

ЗУ с произвольным доступом (RAM)

Динамические ЗУПД (DRAM)

Использующие кучность
адресов

FPM DRAM

EDO DRAM

BEDO DRAM

SDRAM

DDR SDRAM

RDRAM

Не использующие кучность
адресов

DRAM

RLDRAM

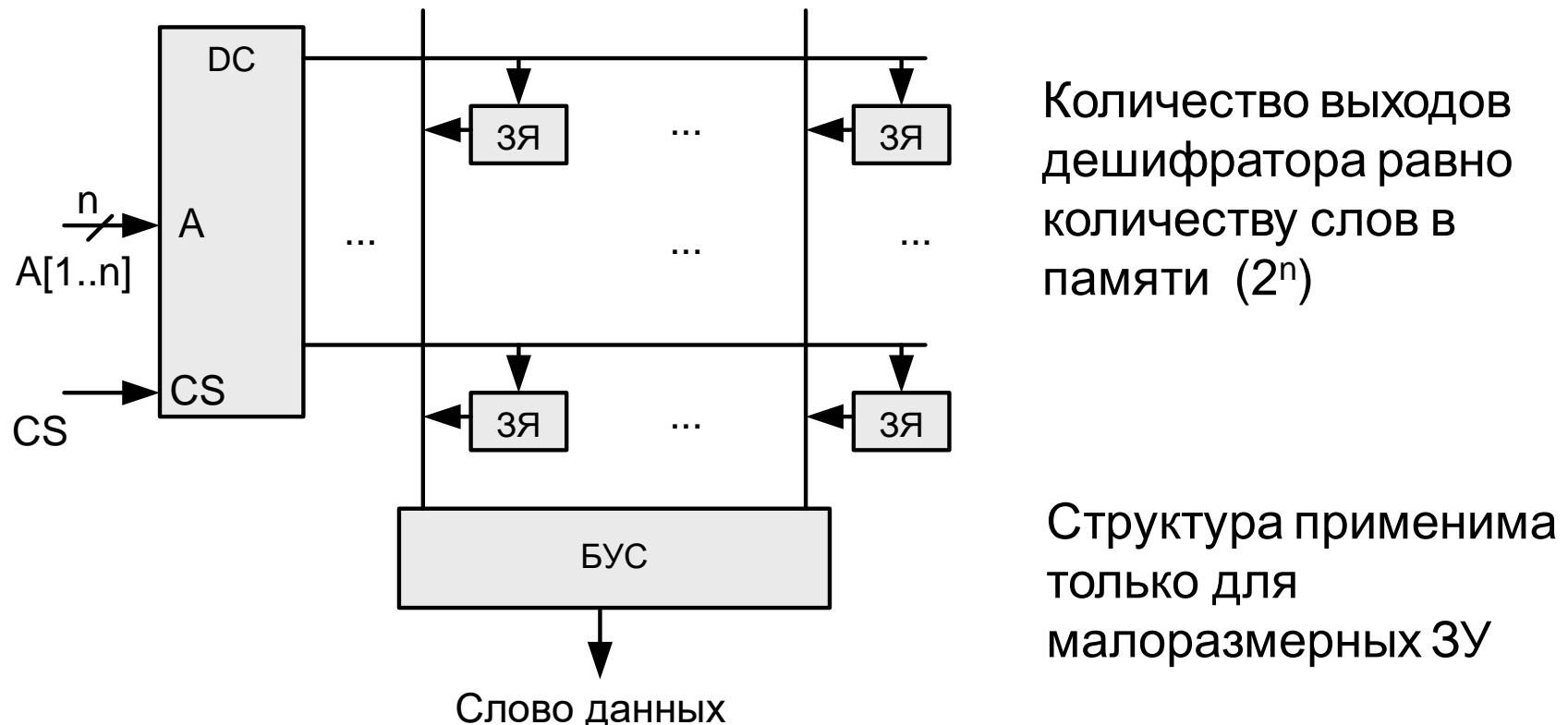
Статические ЗУПД (SRAM)

Асинхронные

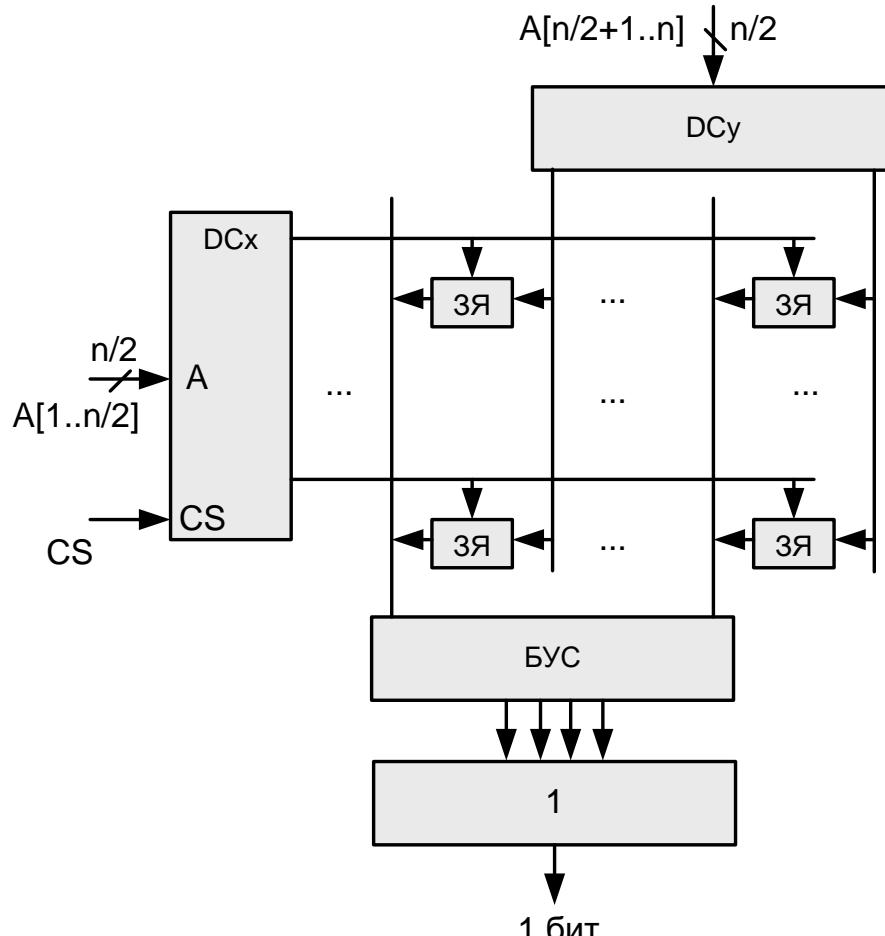
Синхронные

Организация запоминающих массивов адресных ЗУ

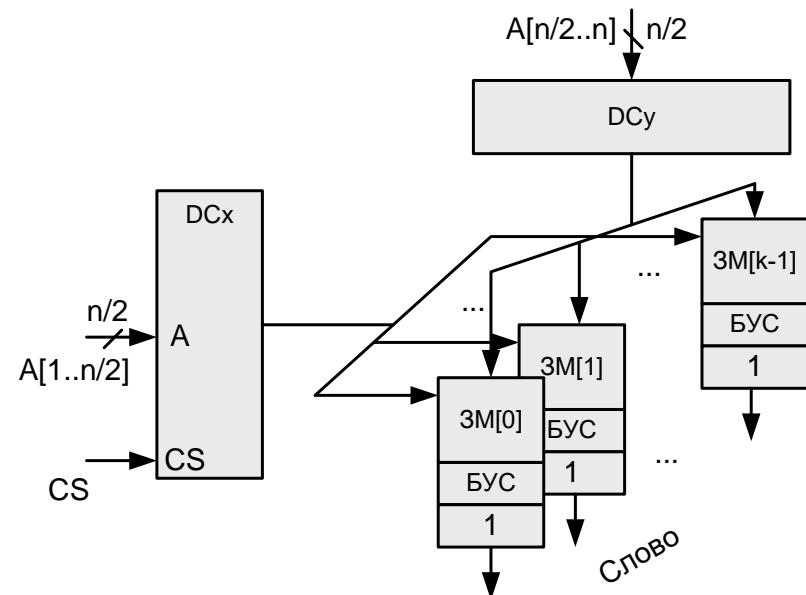
Структура ЗМ типа 2D



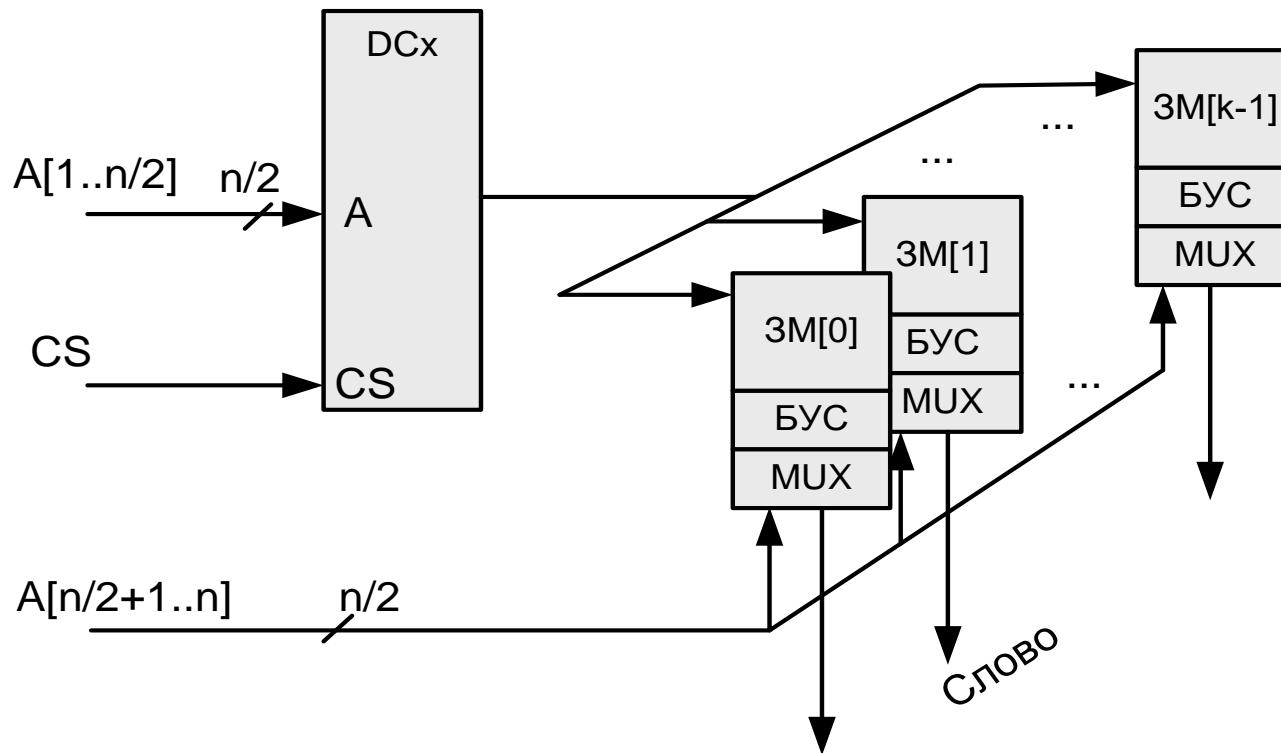
Структура ЗМ типа 3D



Адрес делится на две части (двухкоординатная выборка). Количество выходов дешифраторов: $2^{n/2} + 2^{n/2}$



Структура ЗМ типа 2DM

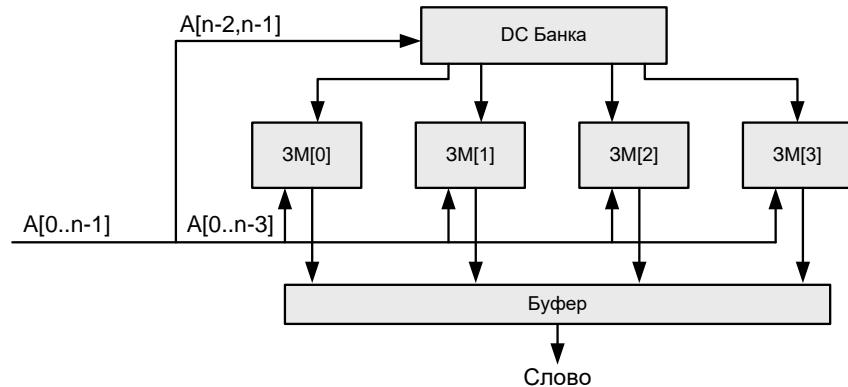


Мультиплексоры позволяют выбрать один из $2^{n/2}$ разрядов каждого из запоминающих массивов

- Размеры массивов близки к оптимальным.
- Количество линий записи/считывания минимально.

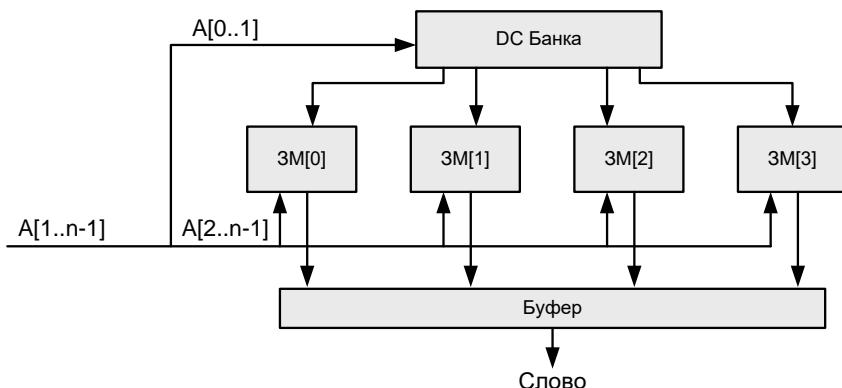
Расслоение памяти

Блочное разделение адреса



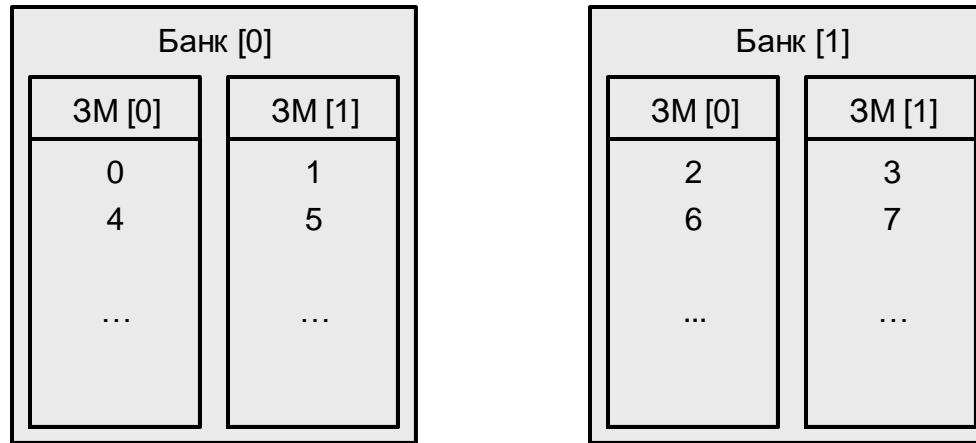
Номер банка определяется старшей частью адреса.

Циклическое разделение адреса



Номер банка определяется младшей частью адреса

Блочно-циклическое разделение адреса



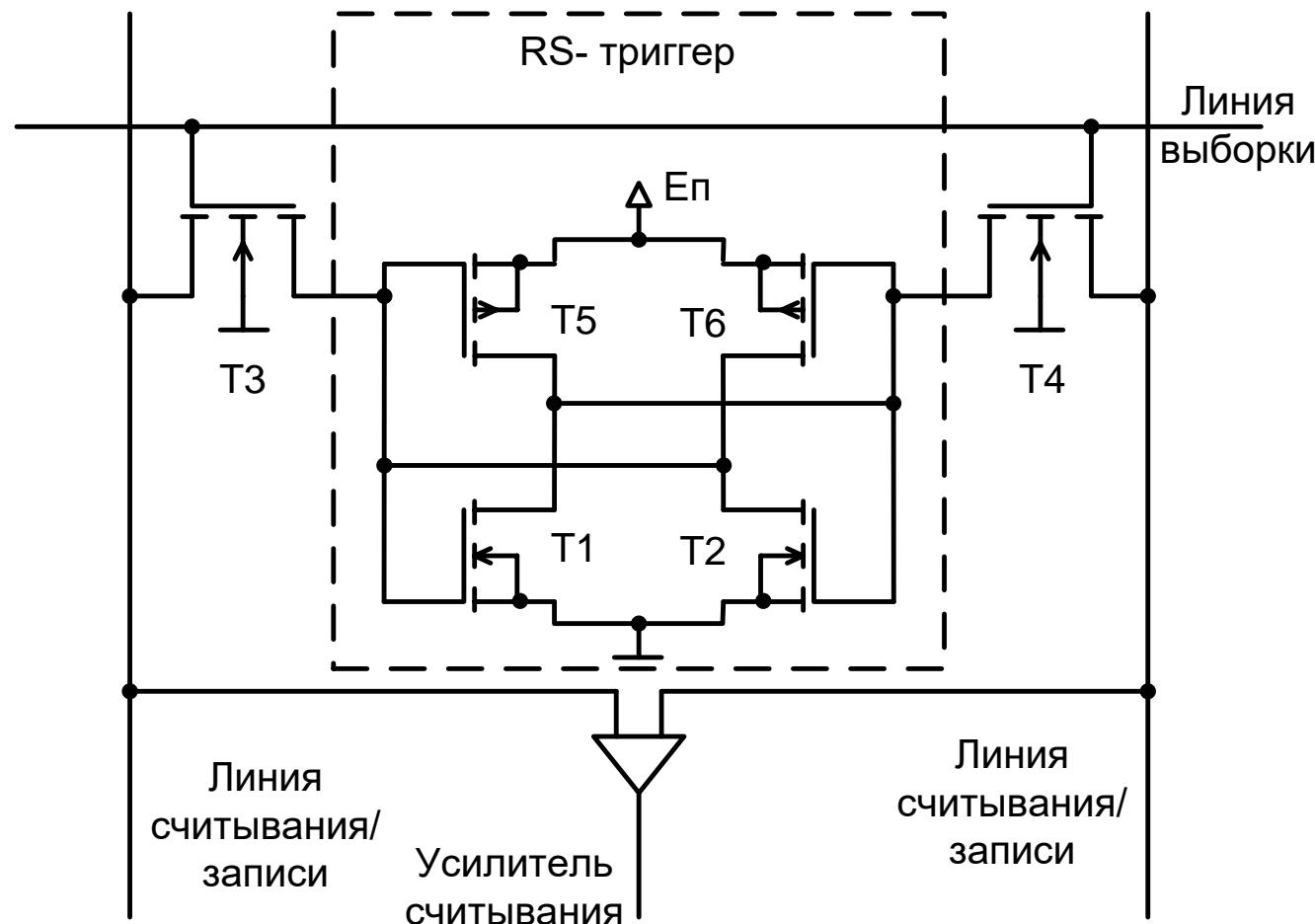
Блочно-циклический способ обеспечивает возможность пакетной передачи и ускоряет доступ при кучности адресов

Пример разделения адреса в SDRAM (PIII, P4)

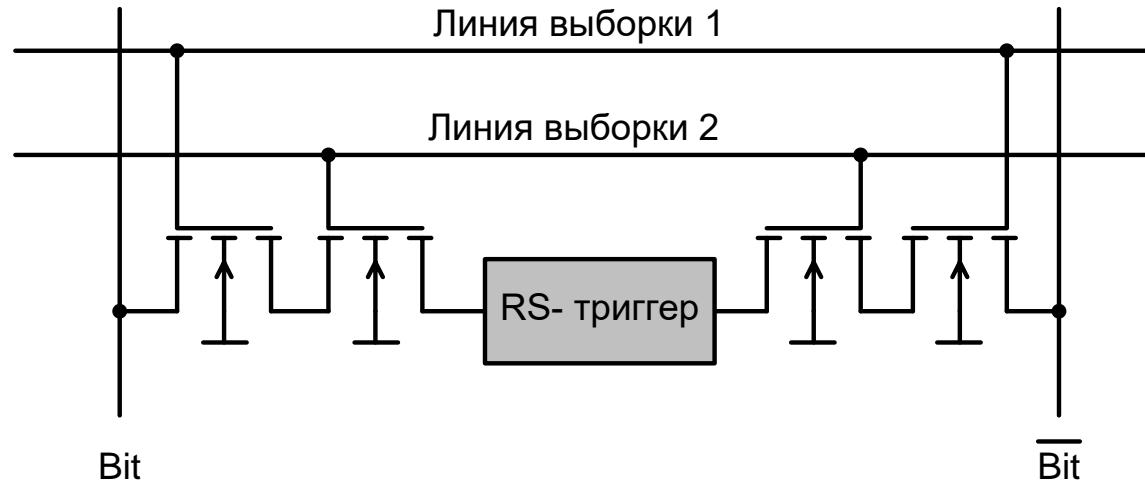


Статические ЗУ с произвольной выборкой (SRAM)

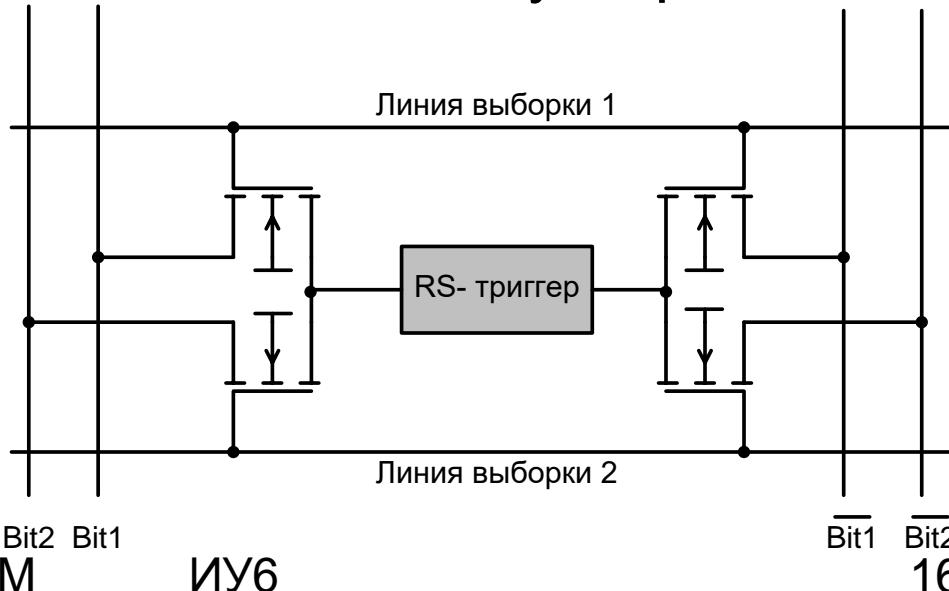
Запоминающая ячейка статической памяти



Запоминающая ячейка с двухкоординатной выборкой



Запоминающая ячейка двухпортовой выборкой



Микросхема статической памяти

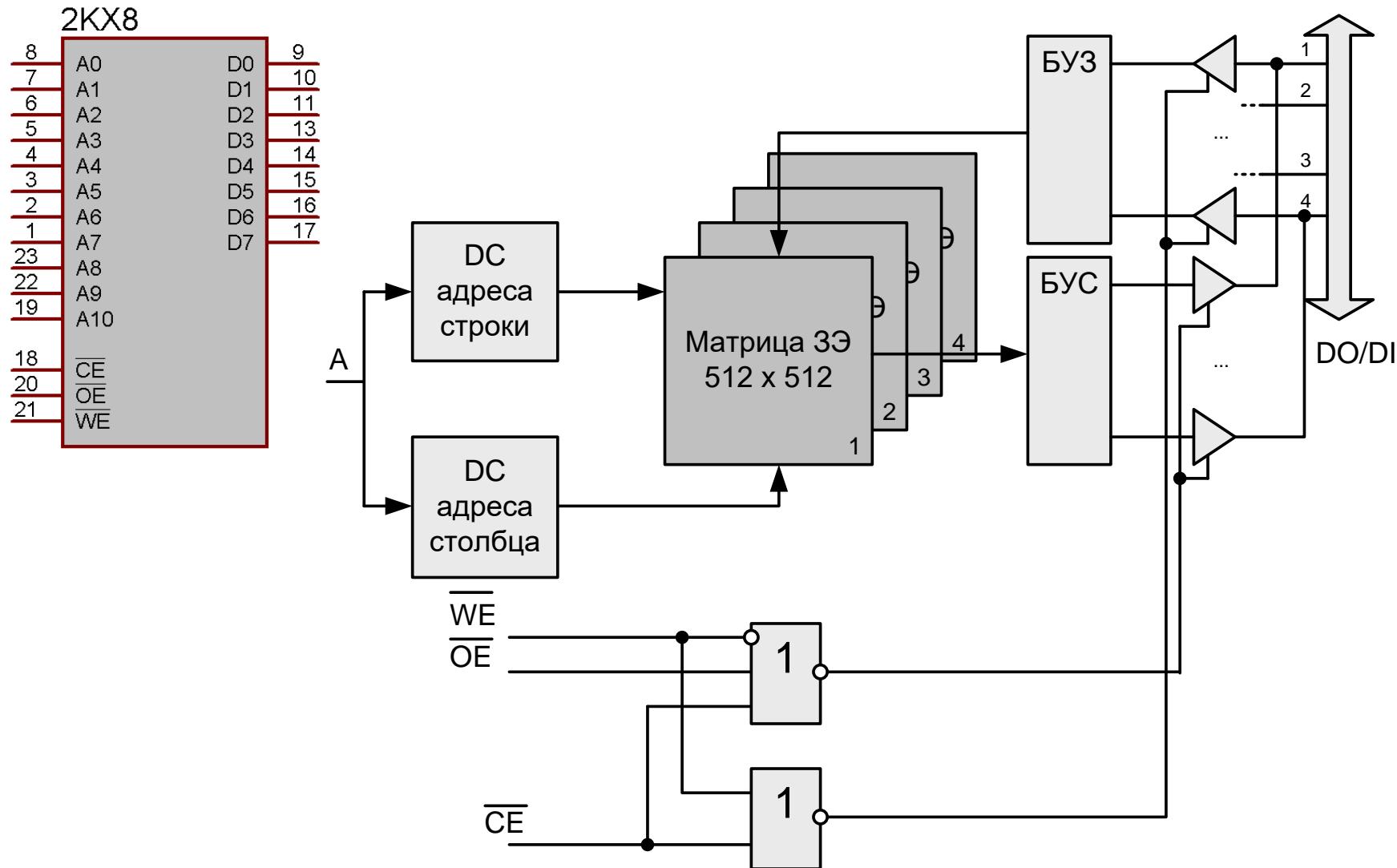
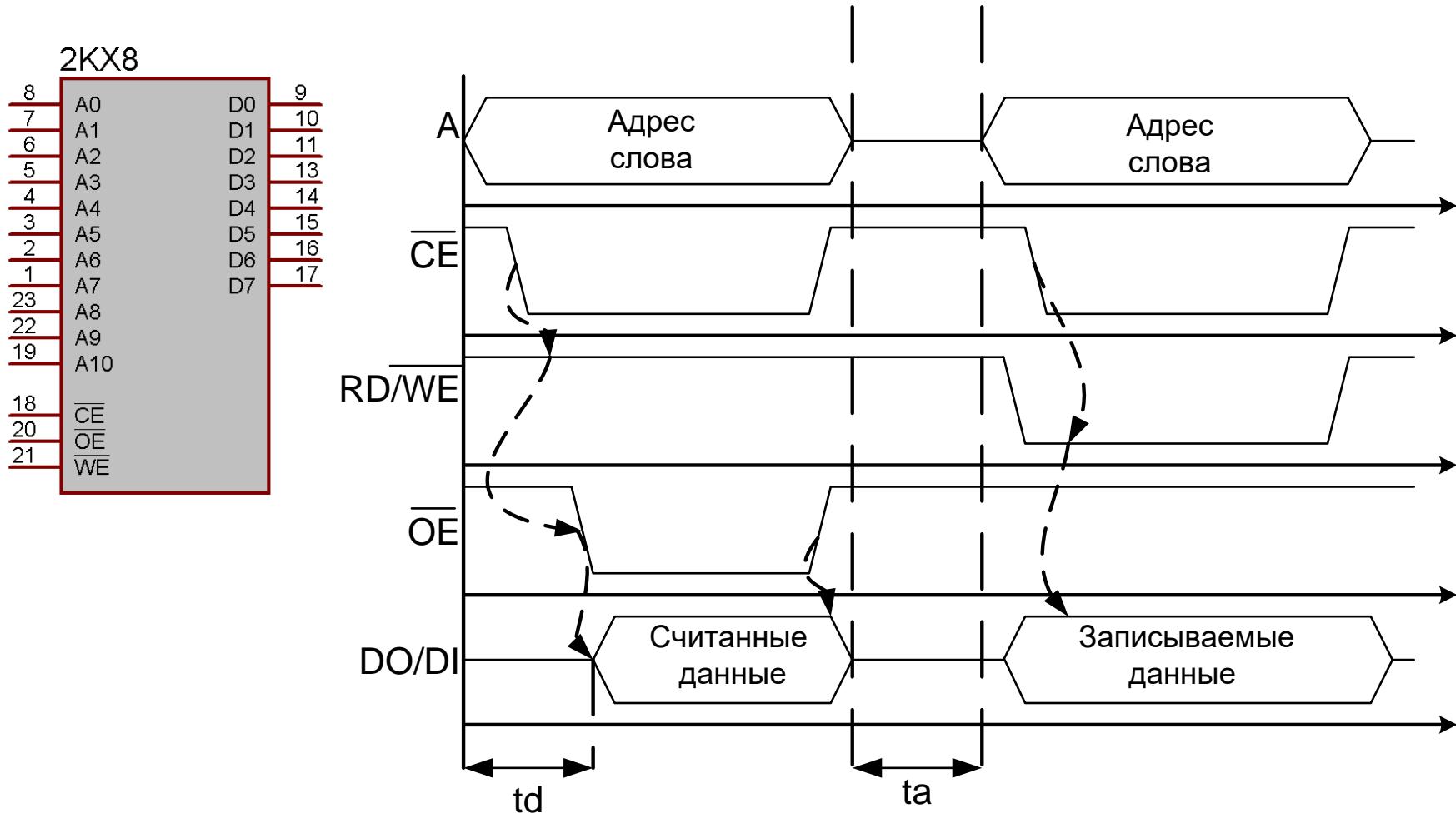
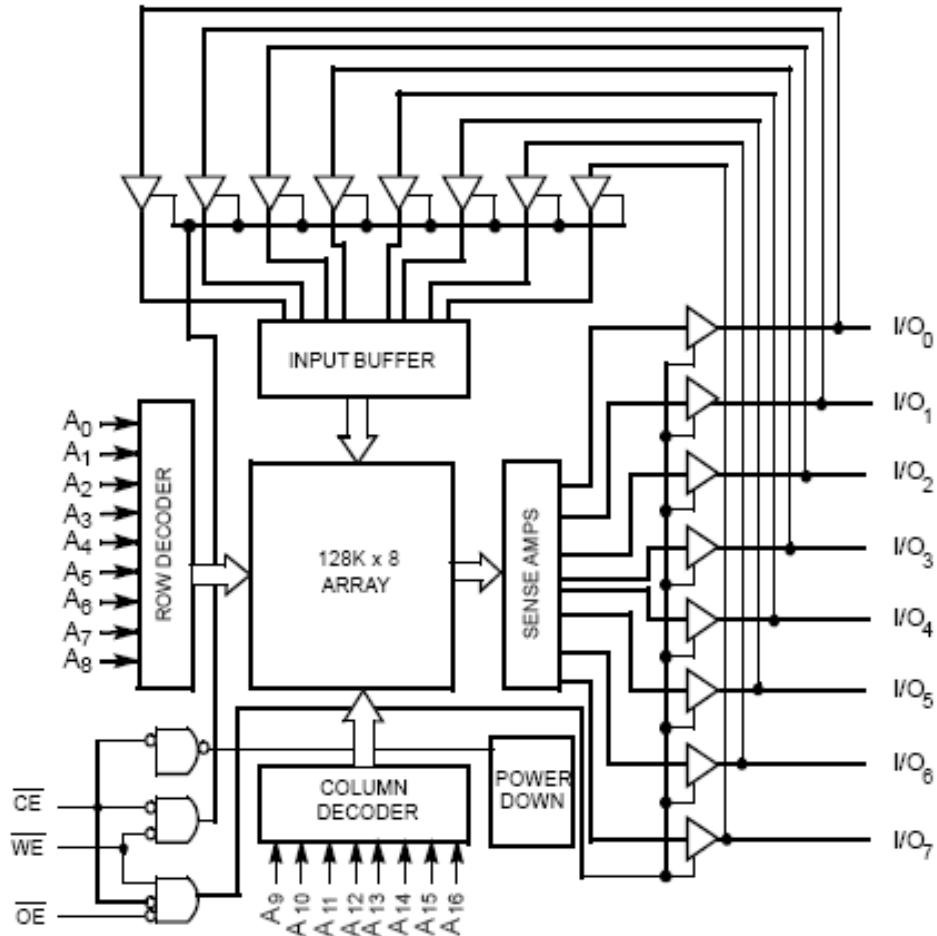


Диаграмма работы статической памяти





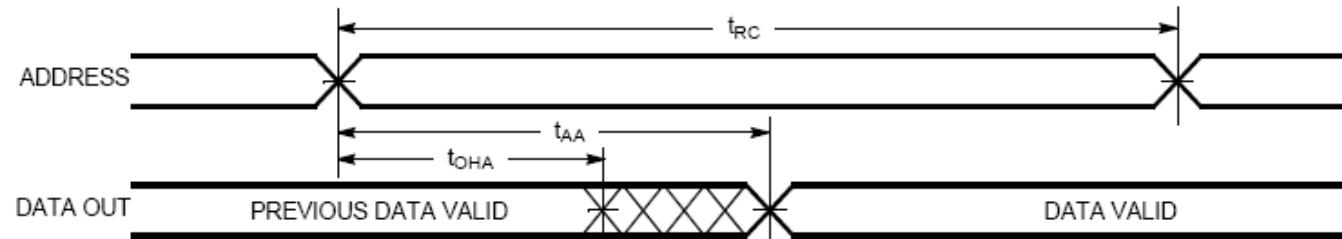
SOJ/TSOP II
Top View

A ₀	1	32	A ₁₆
A ₁	2	31	A ₁₅
A ₂	3	30	A ₁₄
A ₃	4	29	A ₁₃
CE	5	28	OE
I/O ₀	6	27	I/O ₇
I/O ₁	7	26	I/O ₆
V _{CC}	8	25	V _{SS}
V _{SS}	9	24	V _{CC}
I/O ₂	10	23	I/O ₅
I/O ₃	11	22	I/O ₄
WE	12	21	A ₁₂
A ₄	13	20	A ₁₁
A ₅	14	19	A ₁₀
A ₆	15	18	A ₉
A ₇	16	17	A ₈

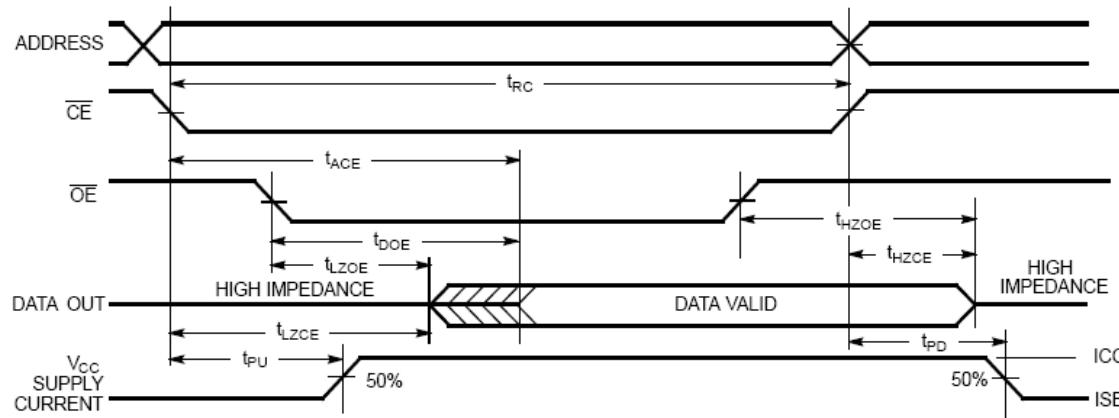
Truth Table

\overline{CE}	\overline{OE}	\overline{WE}	I/O₀-I/O₇	Mode	Power
H	X	X	High Z	Power-Down	Standby (I_{SB})
L	L	H	Data Out	Read	Active (I_{CC})
L	X	L	Data In	Write	Active (I_{CC})
L	H	H	High Z	Selected, Outputs Disabled	Active (I_{CC})

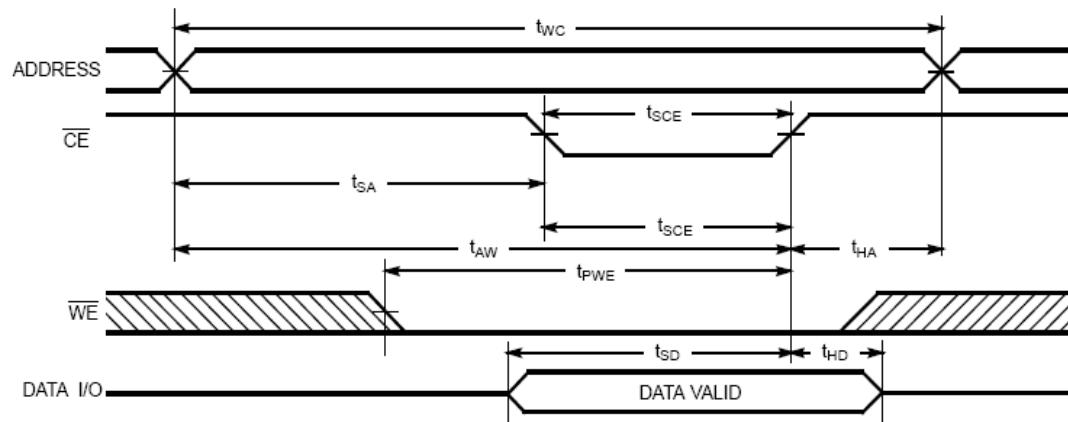
Read Cycle No. 1^[11, 12]



Read Cycle No. 2 (\overline{OE} Controlled)^[12, 13]



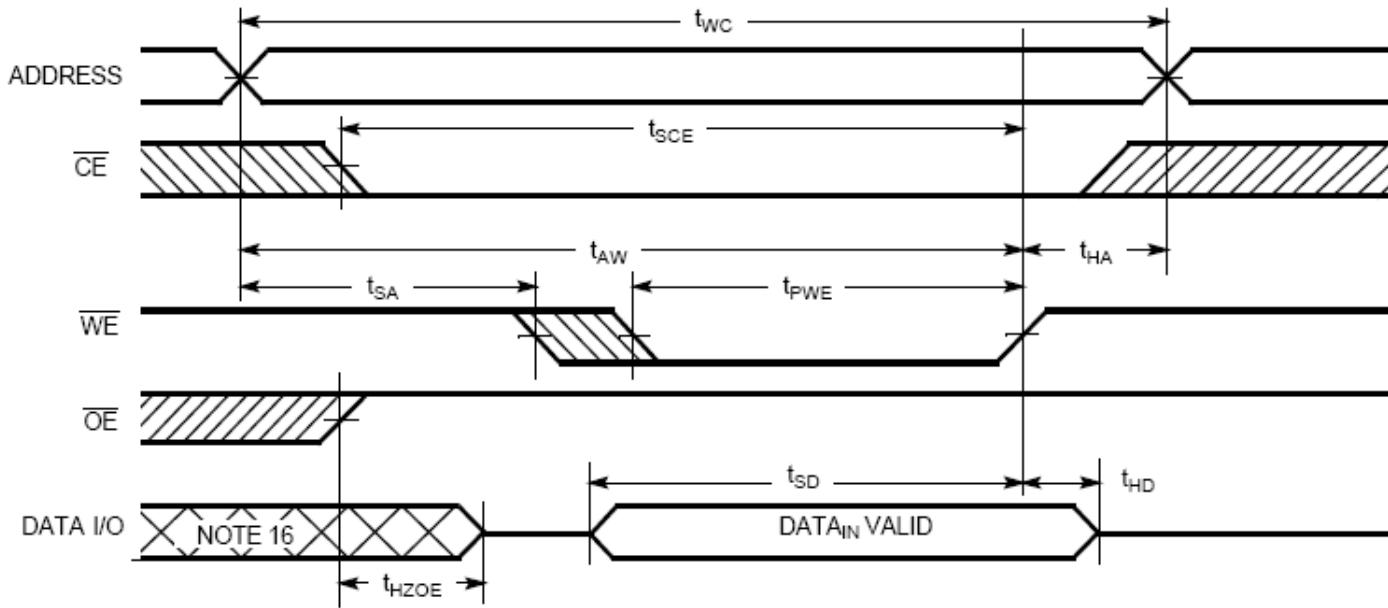
Write Cycle No. 1 (\overline{CE} Controlled)^[14, 15]



Notes:

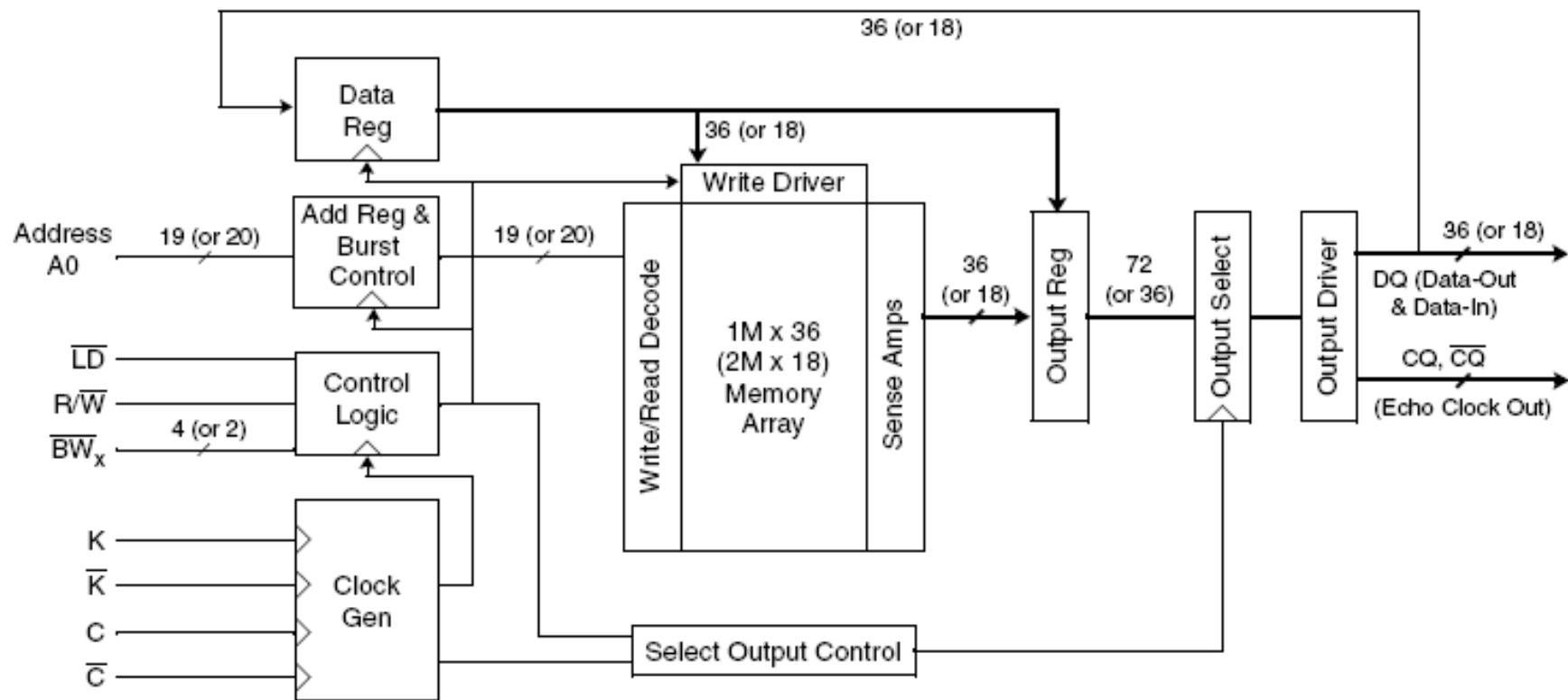
11. Device is continuously selected. $\overline{OE}, \overline{CE} = V_{IL}$.
12. WE is HIGH for read cycle.
13. Address valid prior to or coincident with \overline{CE} transition LOW.
14. Data I/O is high impedance if $\overline{OE} = V_{IH}$.
15. If CE goes HIGH simultaneously with WE going HIGH, the output remains in a high-impedance state.

Write Cycle No. 2 (\overline{WE} Controlled, \overline{OE} HIGH During Write)^[14, 15]



36 Mb (1M x 36 & 2M x 18) DDR-II (Burst of 2) CIO Synchronous SRAMs

ISSI®



36 Mb (1M x 36 & 2M x 18) DDR-II (Burst of 2) CIO Synchronous SRAMs

ISSI®

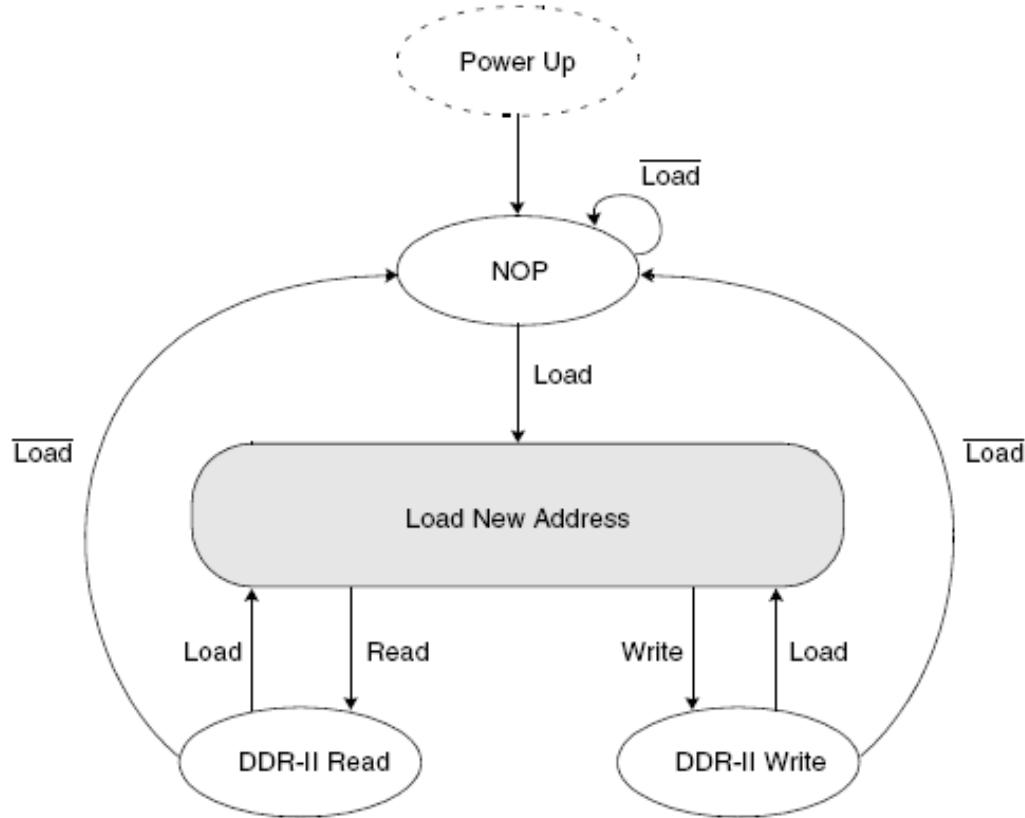
Features

- 1M x 36 or 2M x 18.
- On-chip delay-locked loop (DLL) for wide data valid window.
- Common data input/output bus.
- Synchronous pipeline read with self-timed late write operation.
- Double data rate (DDR-II) interface for read and write input ports.
- Fixed 2-bit burst for read and write operations.
- Clock stop support.
- Two input clocks (K and \bar{K}) for address and control registering at rising edges only.
- Two input clocks (C and \bar{C}) for data output control.
- Two echo clocks (CQ and \bar{CQ}) that are delivered simultaneously with data.
- +1.8V core power supply and 1.5, 1.8V V_{DDQ} , used with 0.75, 0.9V V_{REF}
- HSTL input and output levels.
- Registered addresses, write and read controls, byte writes, data in, and data outputs.
- Full data coherency.
- Boundary scan using limited set of JTAG 1149.1 functions.
- Byte write capability.
- Fine ball grid array (FBGA) package
 - 15mm x 17mm body size
 - 1mm pitch
 - 165-ball (11 x 15) array
- Programmable impedance output drivers via 5x user-supplied precision resistor.

36 Mb (1M x 36 & 2M x 18) DDR-II (Burst of 2) CIO Synchronous SRAMs

ISSI®

Symbol	Pin Number	Description
K, \bar{K}	6B, 6A	Input clock.
C, \bar{C}	6P, 6R	Input clock for output data control.
CQ, \bar{CQ}	11A, 1A	Output echo clock.
Doff	1H	DLL disable when low.
SA ₀	6C	Burst count address input.
SA	9A, 4B, 8B, 5C, 7C, 5N, 6N, 7N, 4P, 5P, 7P, 8P, 3R, 4R, 5R, 7R, 8R, 9R	1M x 36 address inputs.
SA	3A, 9A, 4B, 8B, 5C, 7C, 5N, 6N, 7N, 4P, 5P, 7P, 8P, 3R, 4R, 5R, 7R, 8R, 9R	2M x 18 address inputs.
DQ0–DQ8 DQ9–DQ17 DQ18–DQ26 DQ27–DQ35	11P, 11M, 11L, 11K, 11J, 11F, 11E, 11C, 11B 10P, 11N, 10M, 10K, 10J, 11G, 10E, 11D, 10C 3B, 3D, 3E, 3F, 3G, 3K, 3L, 3N, 3P 2B, 3C, 2D, 2F, 2G, 3J, 2L, 3M, 2N	1M x 36 DQ pins
DQ0–DQ8 DQ9–DQ17	11P, 10M, 11L, 11K, 10J, 11F, 11E, 10C, 11B 2B, 3D, 3E, 2F, 3G, 3K, 2L, 3N, 3P	2M x 18 DQ pins
R/W	4A	Read/write control. Read when active high.
LD	8A	Synchronizes load. Loads new address when low.
\overline{BW}_0 , \overline{BW}_1 , \overline{BW}_2 , \overline{BW}_3	7B, 7A, 5A, 5B	1M x 36 byte write control, active low.
\overline{BW}_0 , \overline{BW}_1	7B, 5A	2M x 18 byte write control, active low.



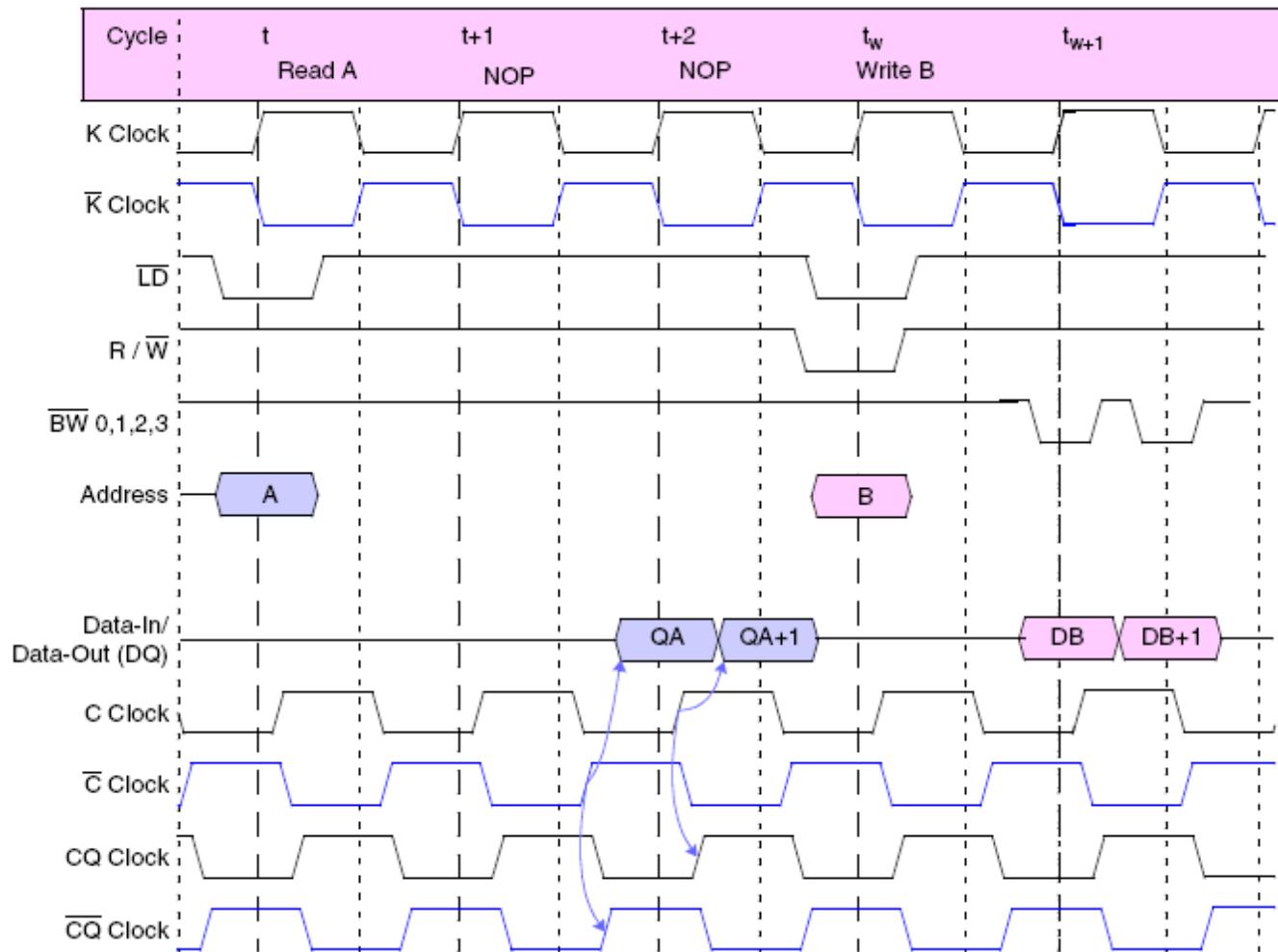
Notes:

1. Internal burst counter is fixed as two-bit linear; that is, when first address is A0+0, next internal burst address is A0+1.
2. *Read* refers to read active status with R/W = high.
3. *Write* refers to write active status with R/W = low.
4. *Load* refers to read new address active status with \overline{LD} = low.
5. *Load* is read new address inactive status with \overline{LD} = high.

36 Mb (1M x 36 & 2M x 18) DDR-II (Burst of 2) CIO Synchronous SRAMs

ISSI®

ПРИМЕР

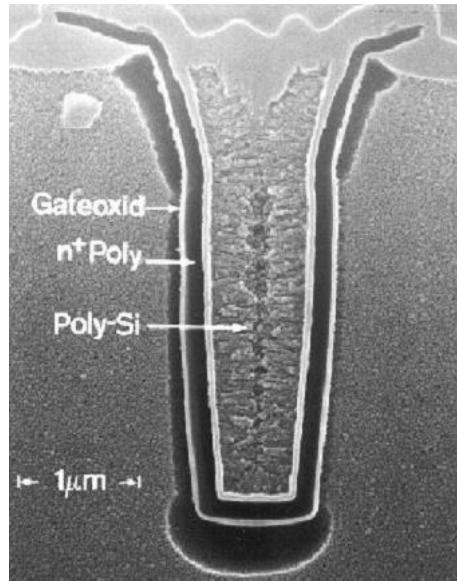


Динамические ЗУ с произвольной выборкой (DRAM)

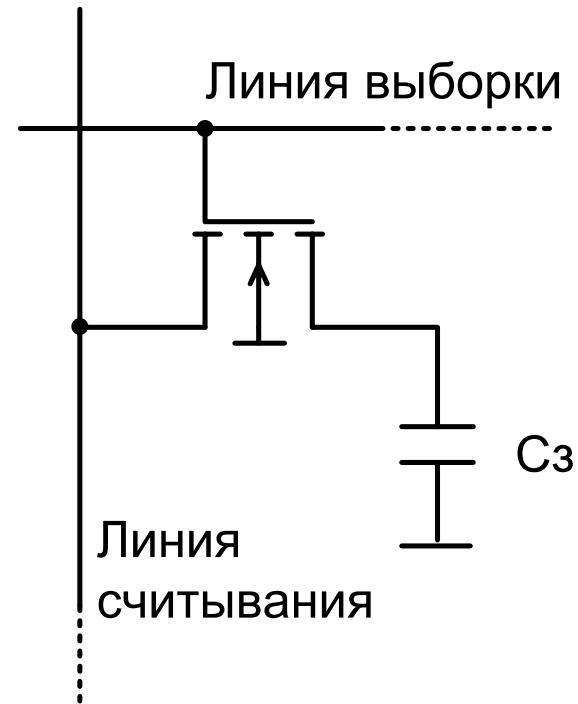
DRAM для обращения по произвольным адресам

DRAM, RLDRAM

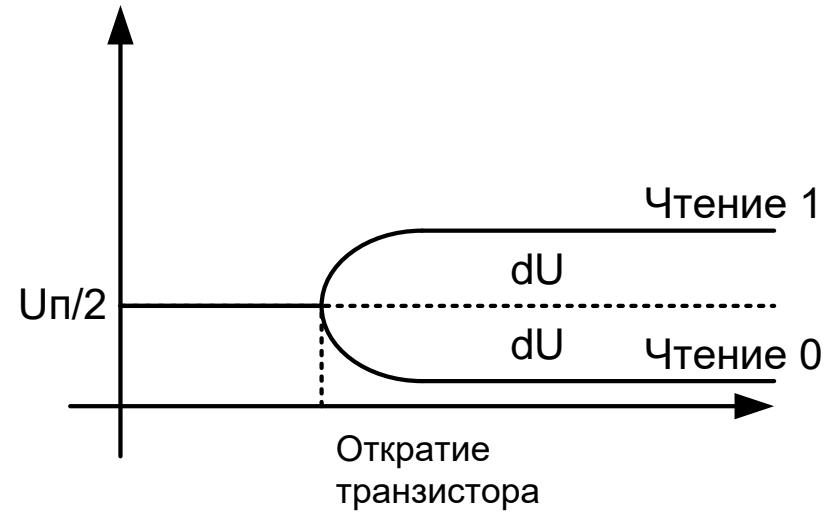
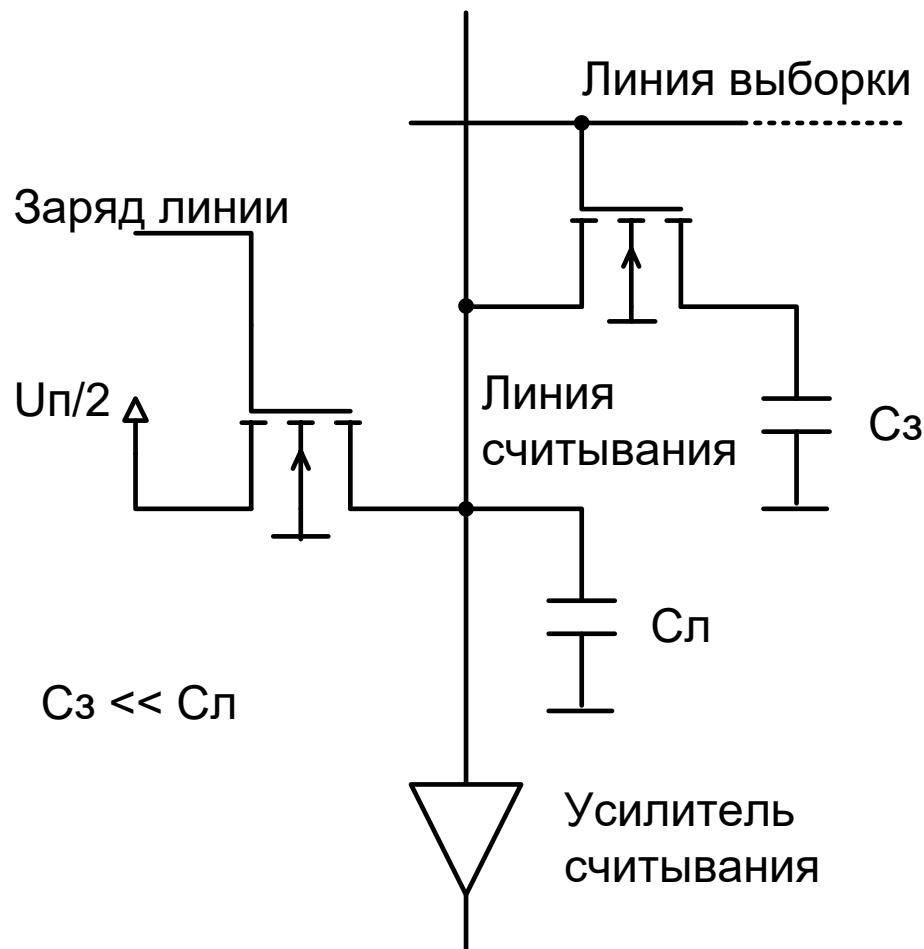
DRAM, оптимизированные для обращения по последовательным адресам:
FPM DRAM, EDO DRAM, BEDO DRAM, SDRAM, DDR SDRAM, RDRAM



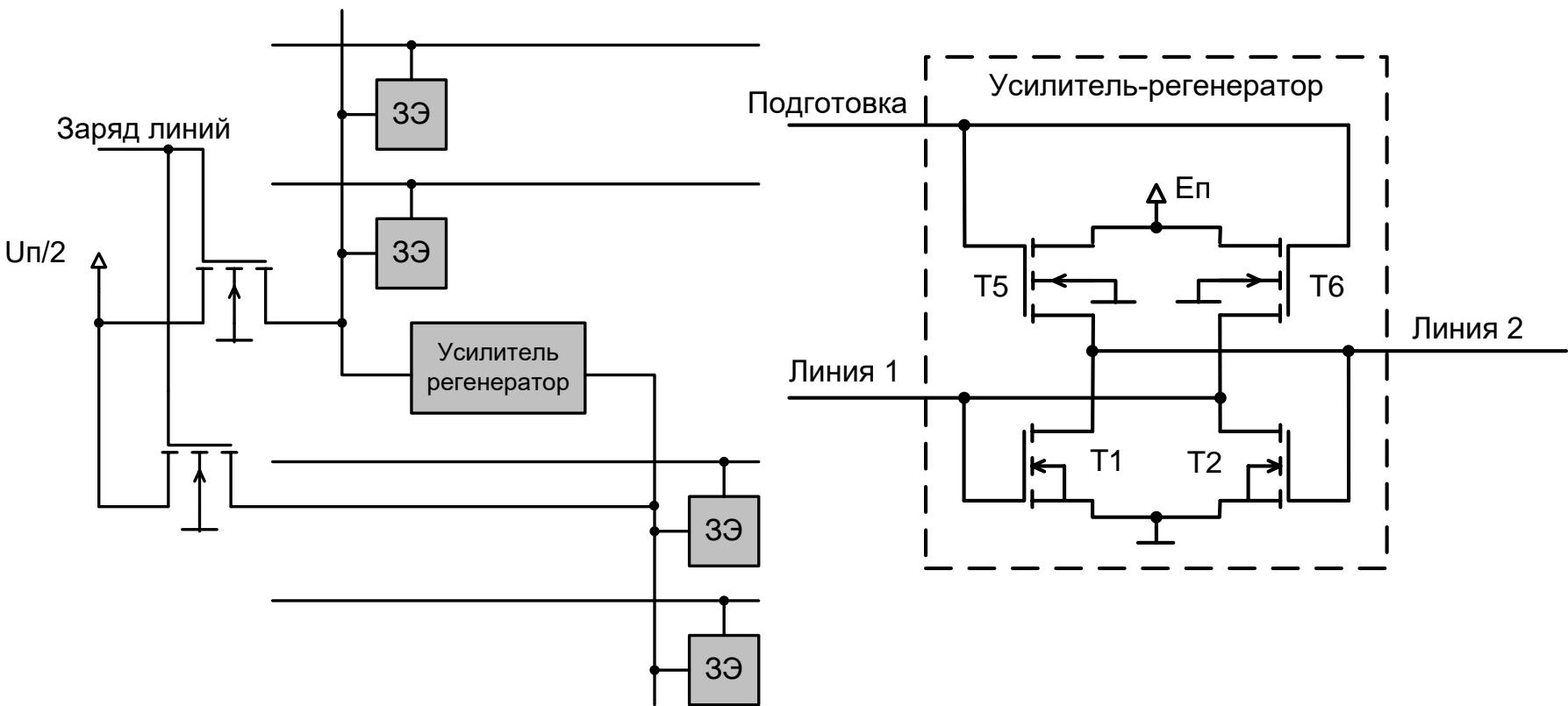
При выборке строки все Сз подключаются к линиям считывания.
После считывания необходимо произвести обратную запись информации – регенерацию.
Заряд до $10^5 - 10^6$ электронов.



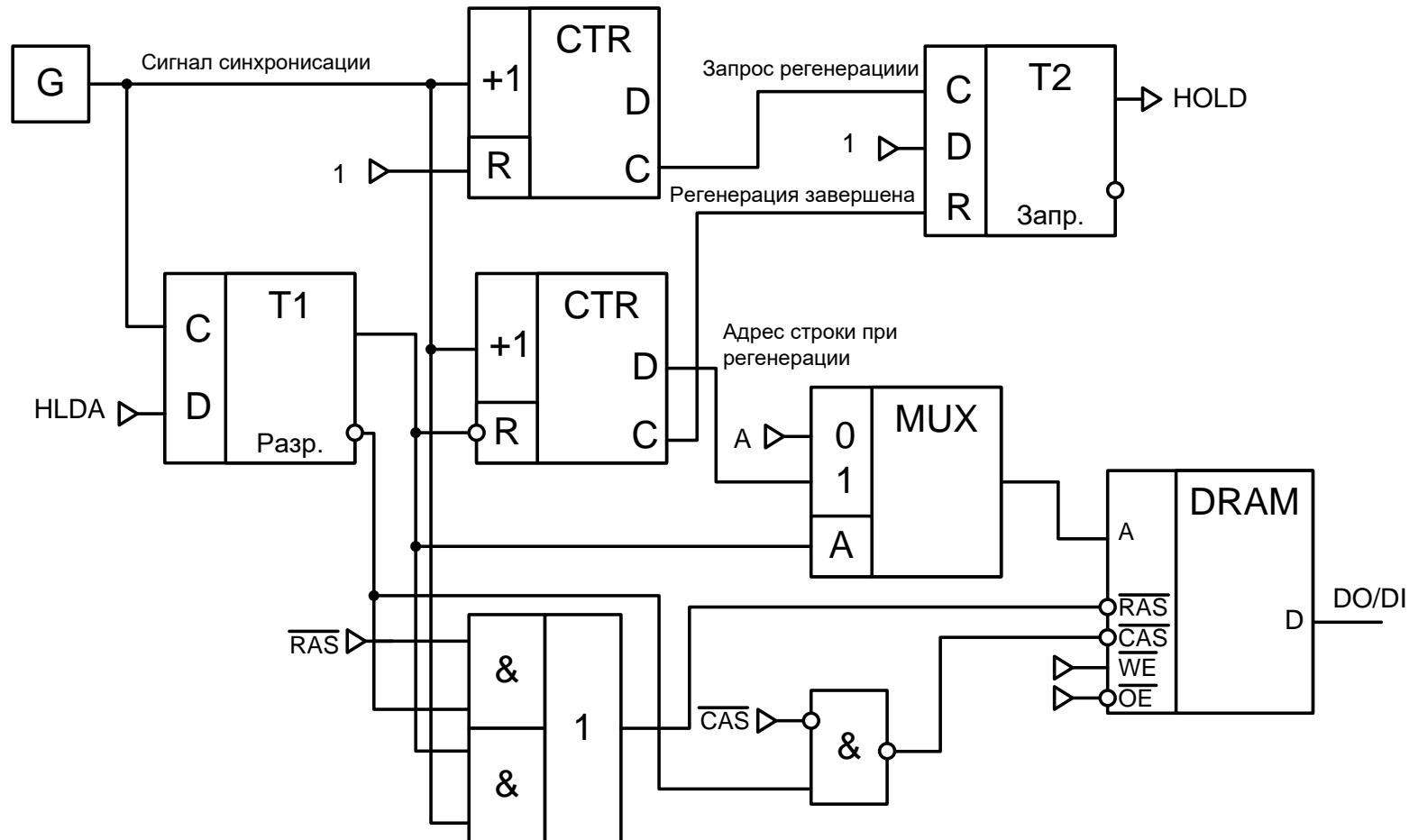
Процесс считывания в DRAM



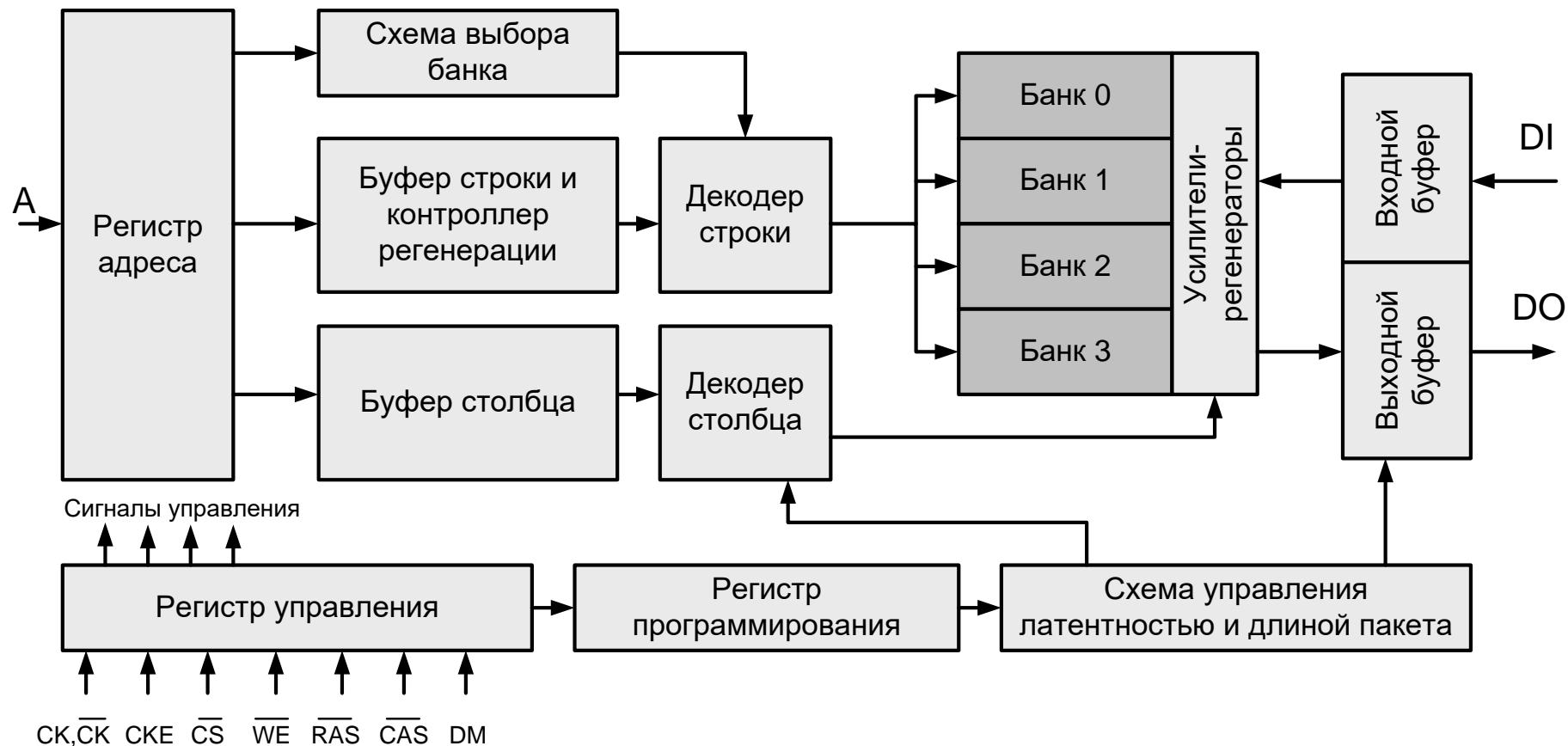
Принцип действия усилителя-регенератора

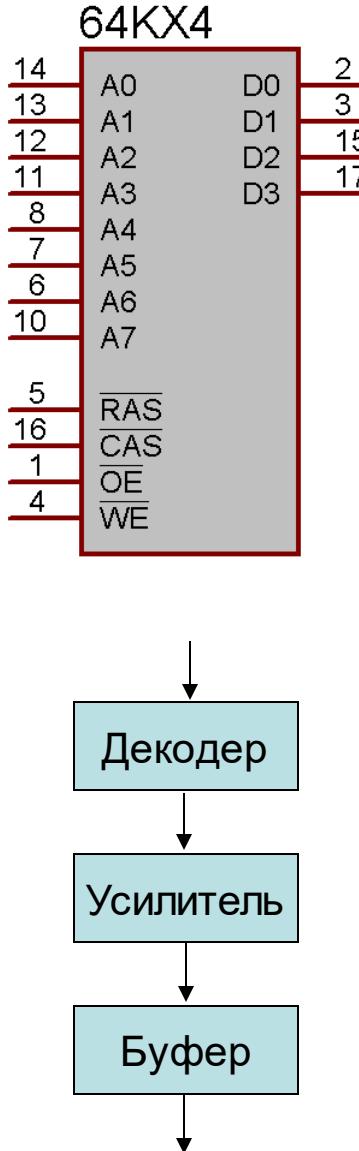


Контроллер динамической памяти



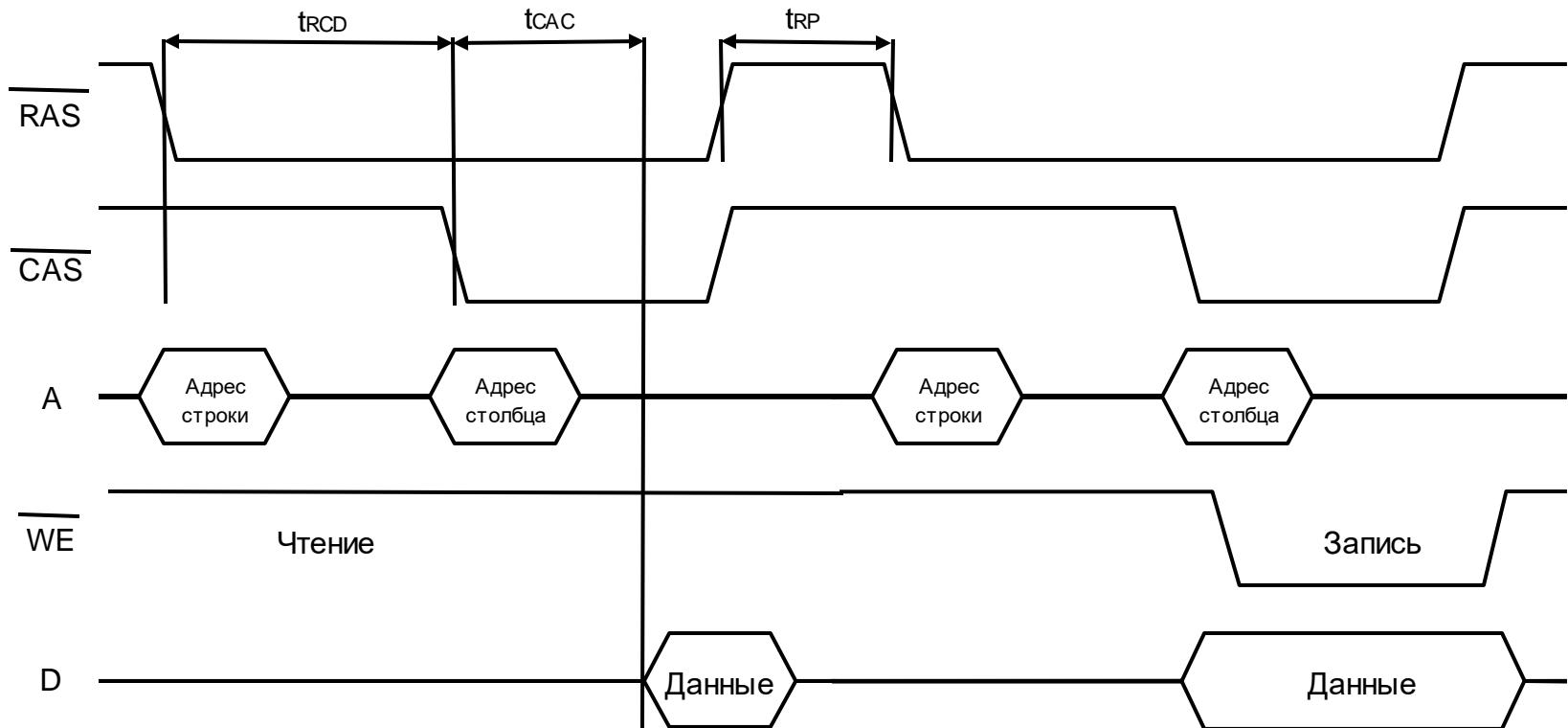
Микросхема динамической памяти





- ### Функциональные возможности SDRAM памяти:
- Многобанковая организация.
 - Командный режим работы.
 - Команды пакетного чтения/записи.
 - Использование чередования банков при последовательном увеличении адресов.
 - Команды пакетного чтения/записи с autoprecharge.
 - Возможность останова чтения/записи по режиму регенерации.
 - Возможность останова чтения/записи по новому запросу чтения/записи.
 - Управление маскированием шины данных по сигналу DQM.
 - Минимальное время (1 CLK) между последовательными командами.
 - Команда PrechargeAll.
 - CAS латентность 2 и 3 CLK.
 - Длина пакета 1,2 и 4 слова.
 - Команда саморегенерации.
 - Режим энергосбережения.

Диаграмма работы DRAM памяти



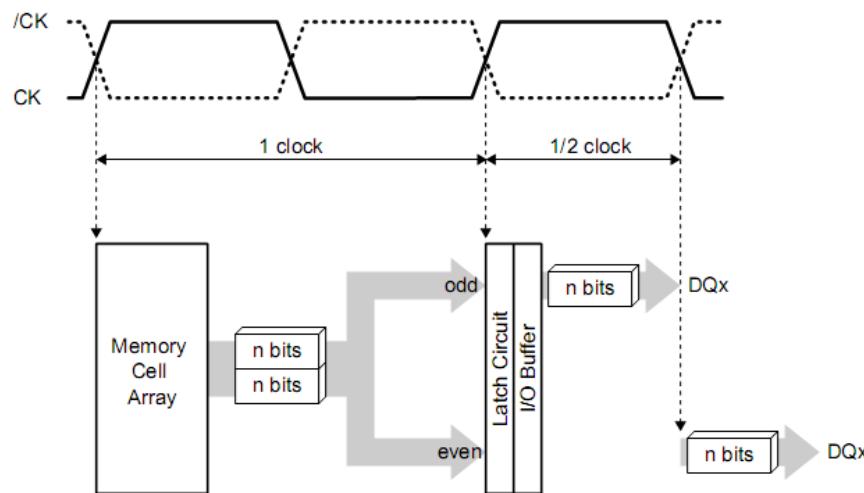
t_{RCD} – RAS to CAS Delay.

t_{RP} – RAS Precharge.

t_{CAC} – CAS Delay.

Способы повышения производительности RAM

- Синхронизация.
- Конвейеризация.
- Пакетный режим обмена.
- Ускорение реверса шины.
- Чередование банков при обращении по последовательным адресам.
- Удвоение скорости.



Регистр DDR

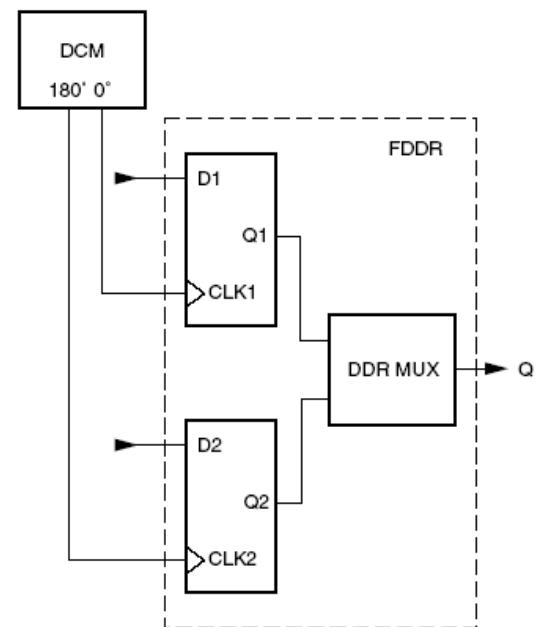


Диаграмма работы FPM DRAM памяти

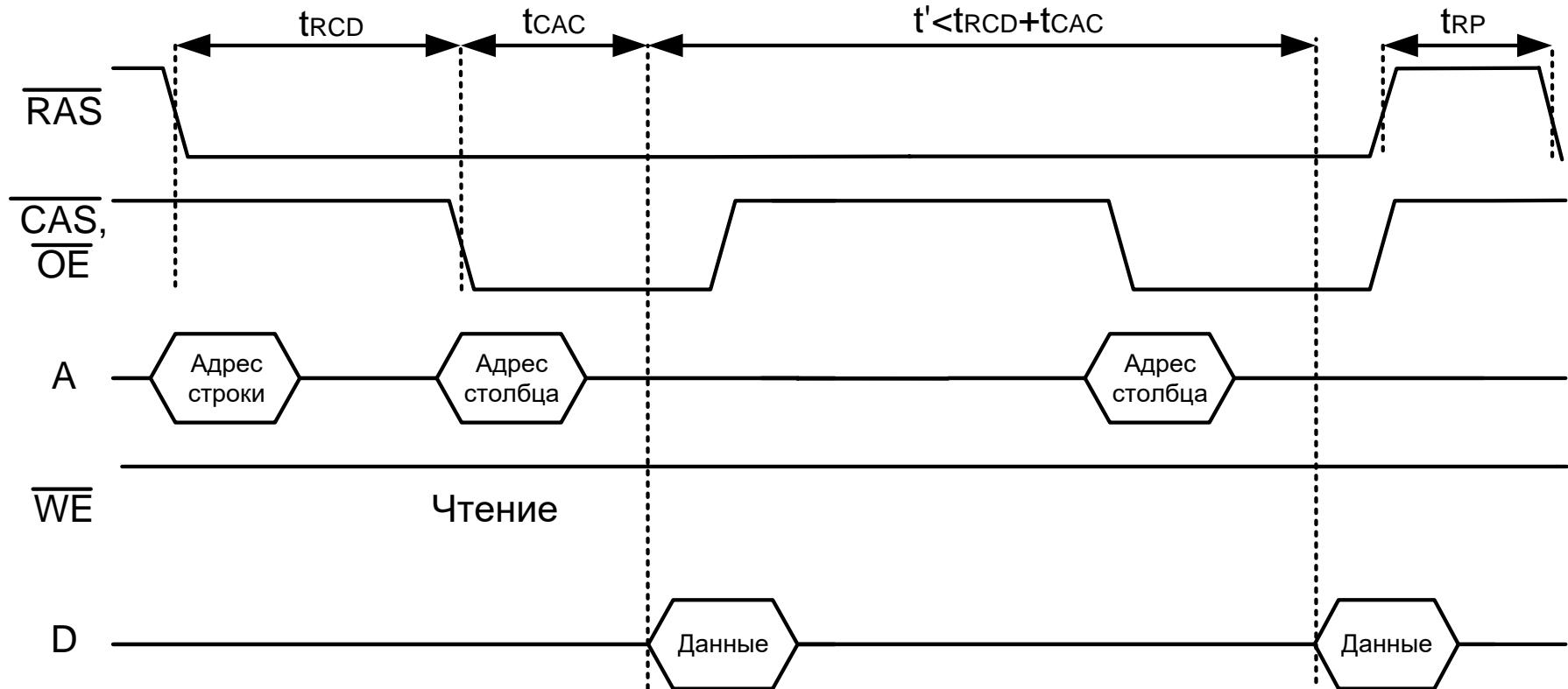


Диаграмма работы BEDO DRAM памяти

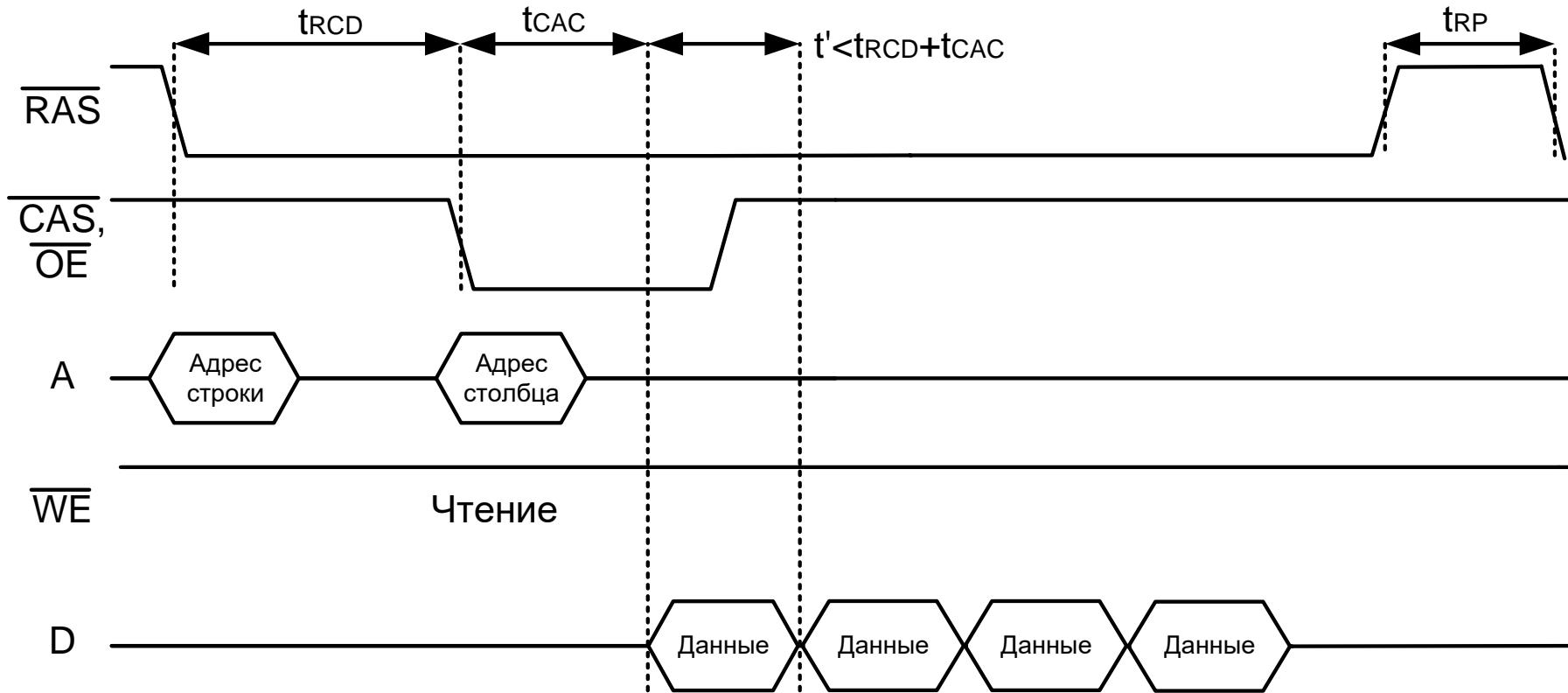
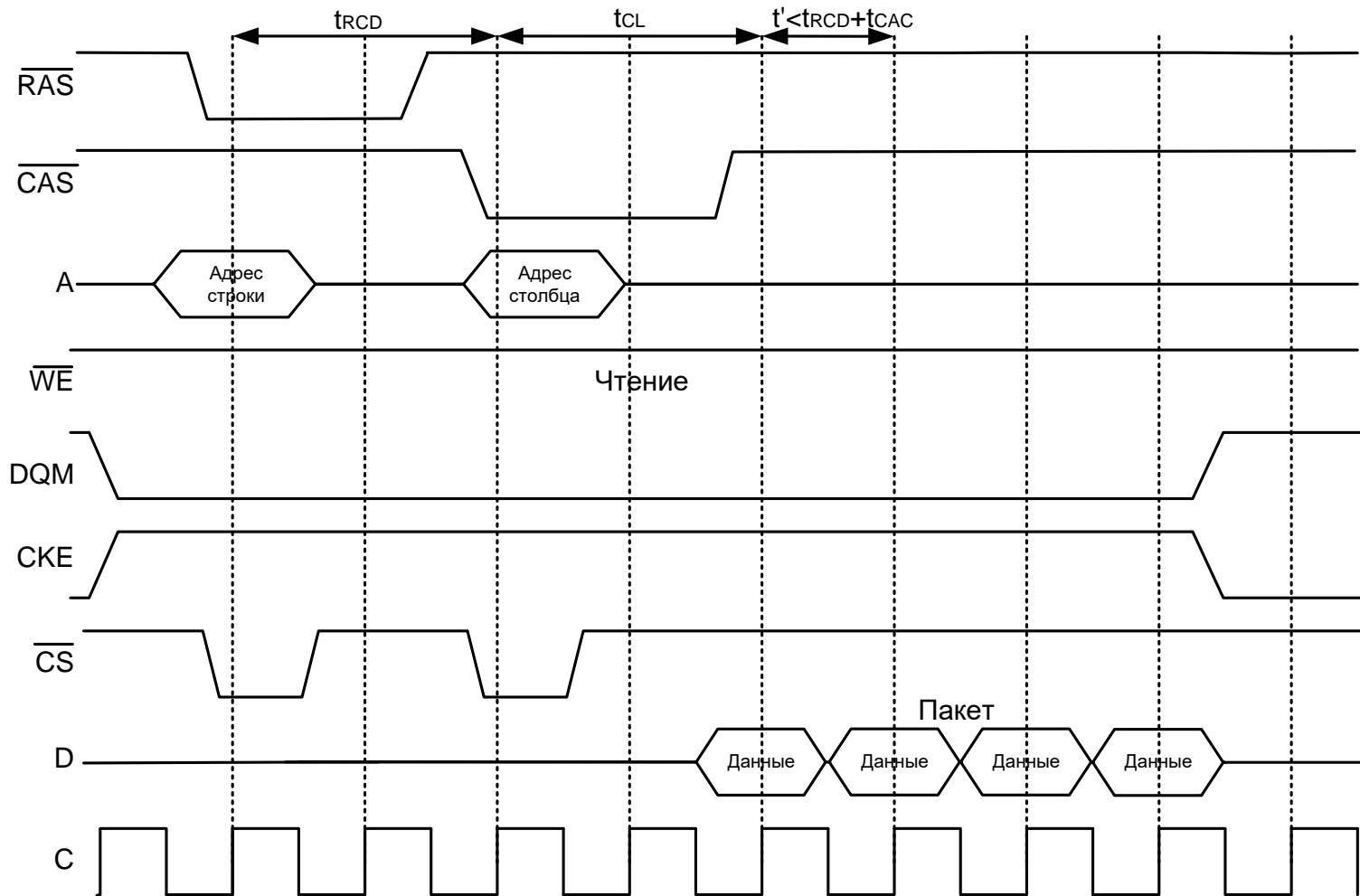
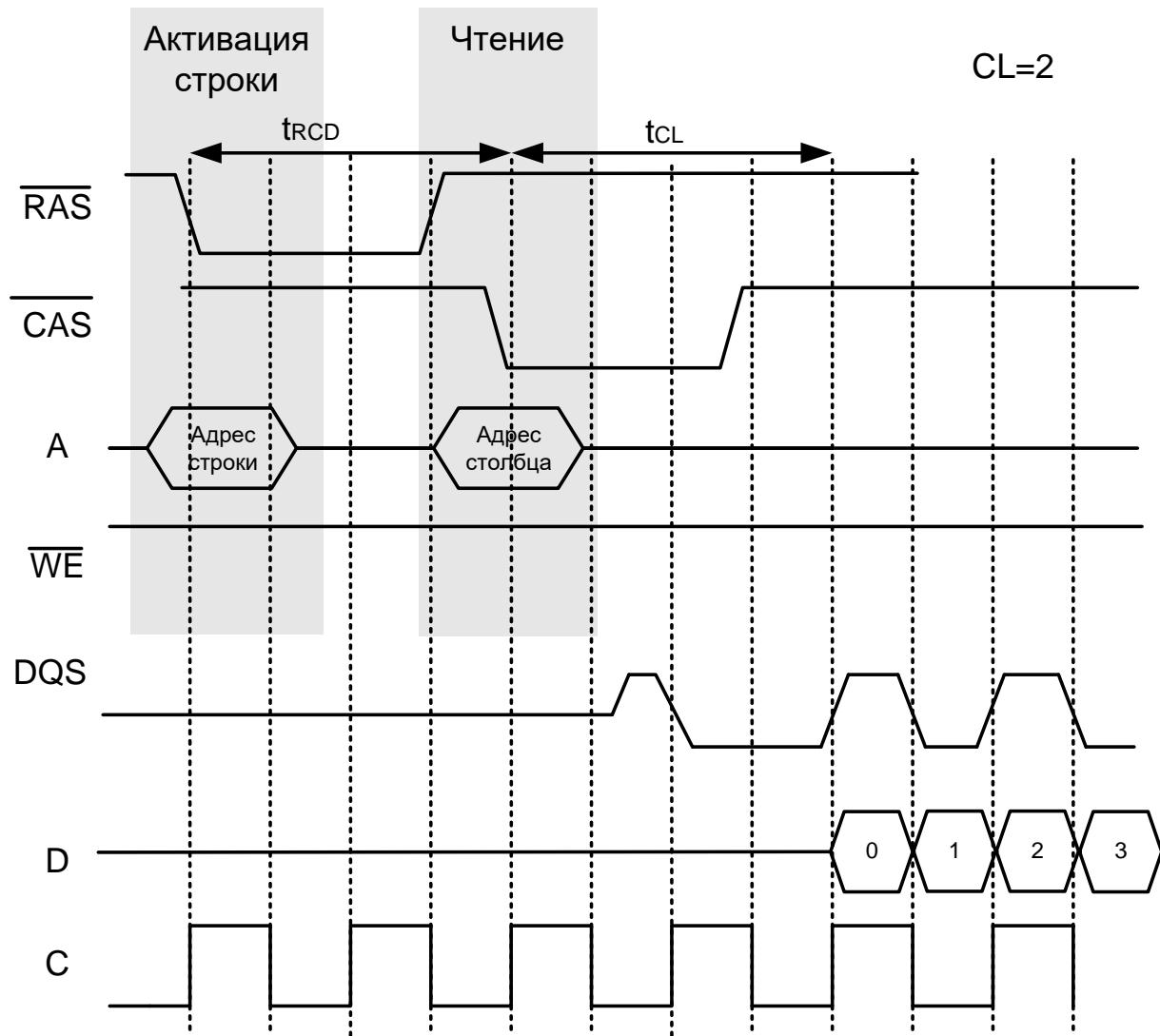


Диаграмма работы SDRAM памяти

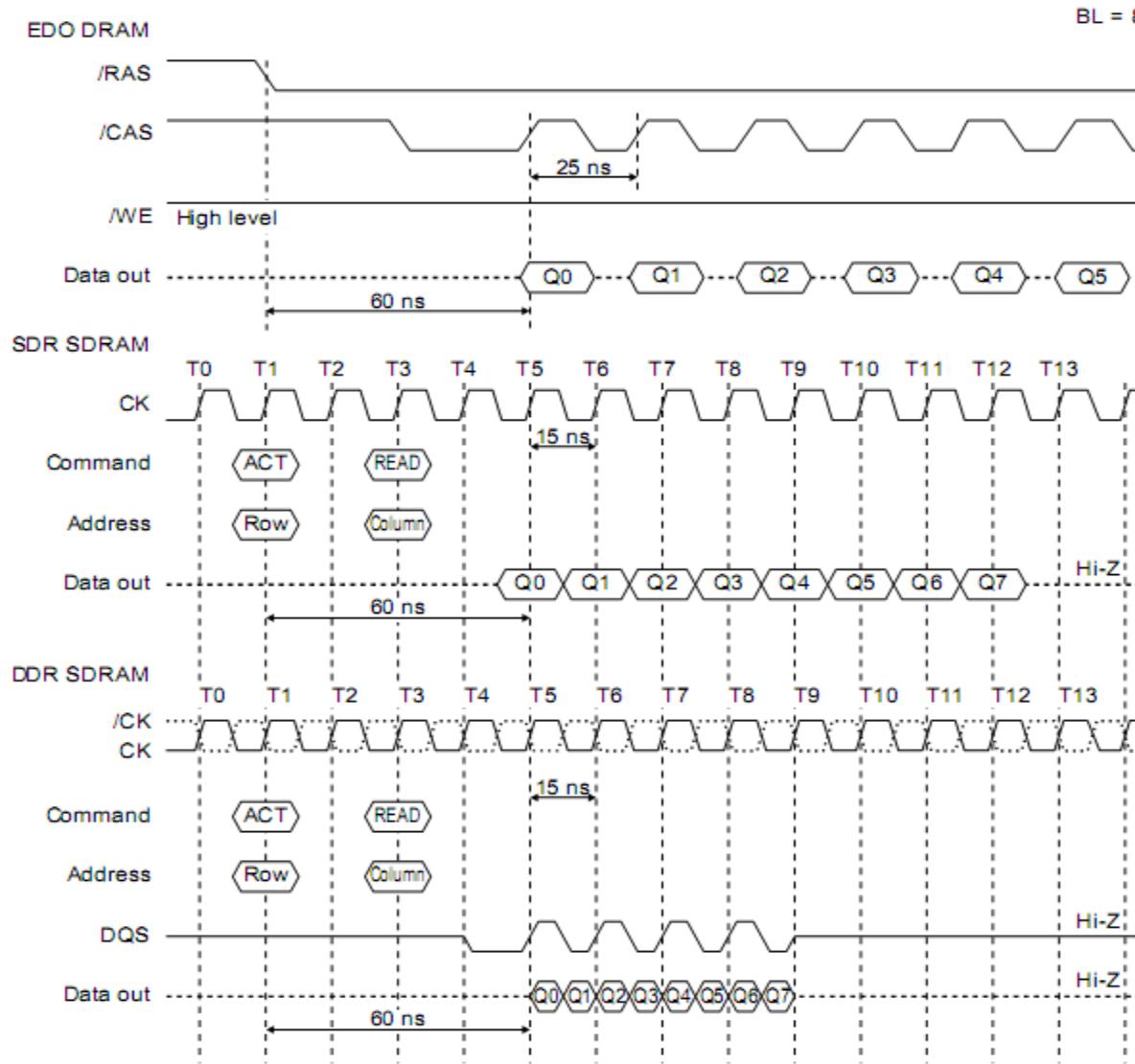


Формула памяти: 4-1-1-1

Диаграмма работы DDR SDRAM памяти



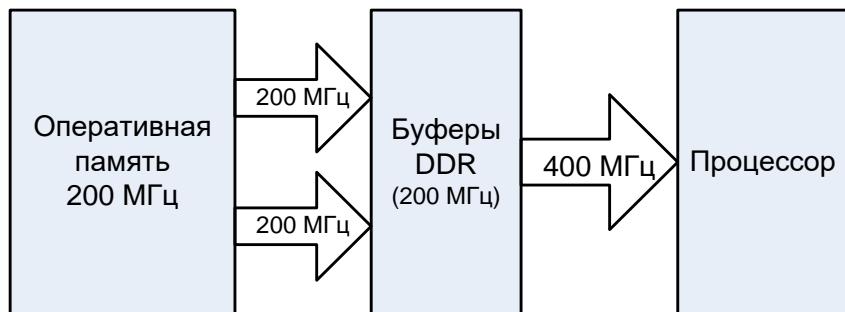
Сравнение EDO RAM, SDRAM, DDR SDRAM



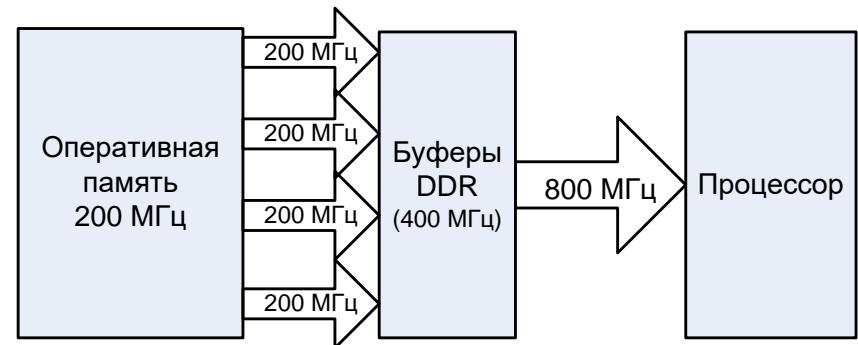
Document No. E0234E30 (Ver.3.0)
Date Published April 2002 (K) Japan
URL: <http://www.elpida.com>

Сравнение DDR и DDR2

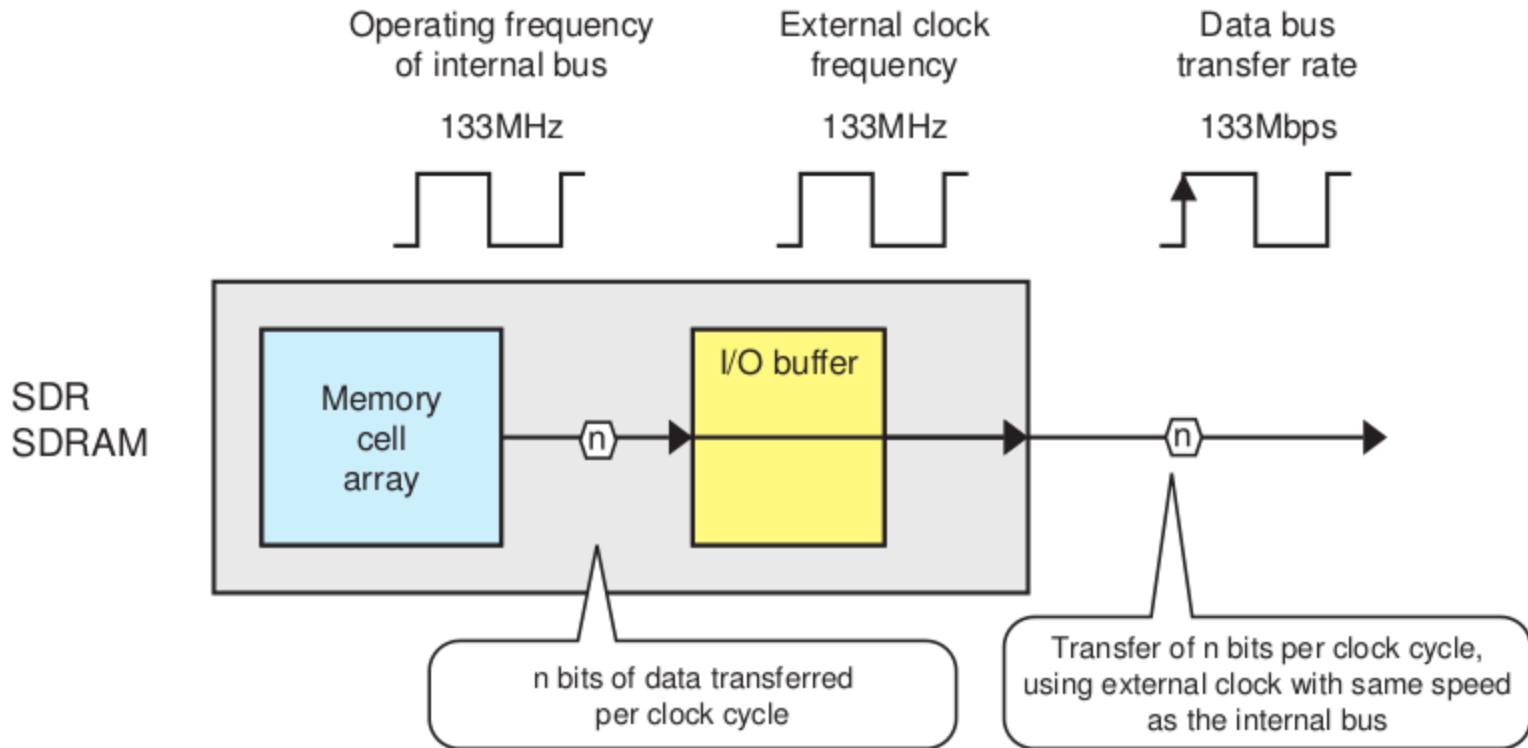
DDR память



DDR2 память

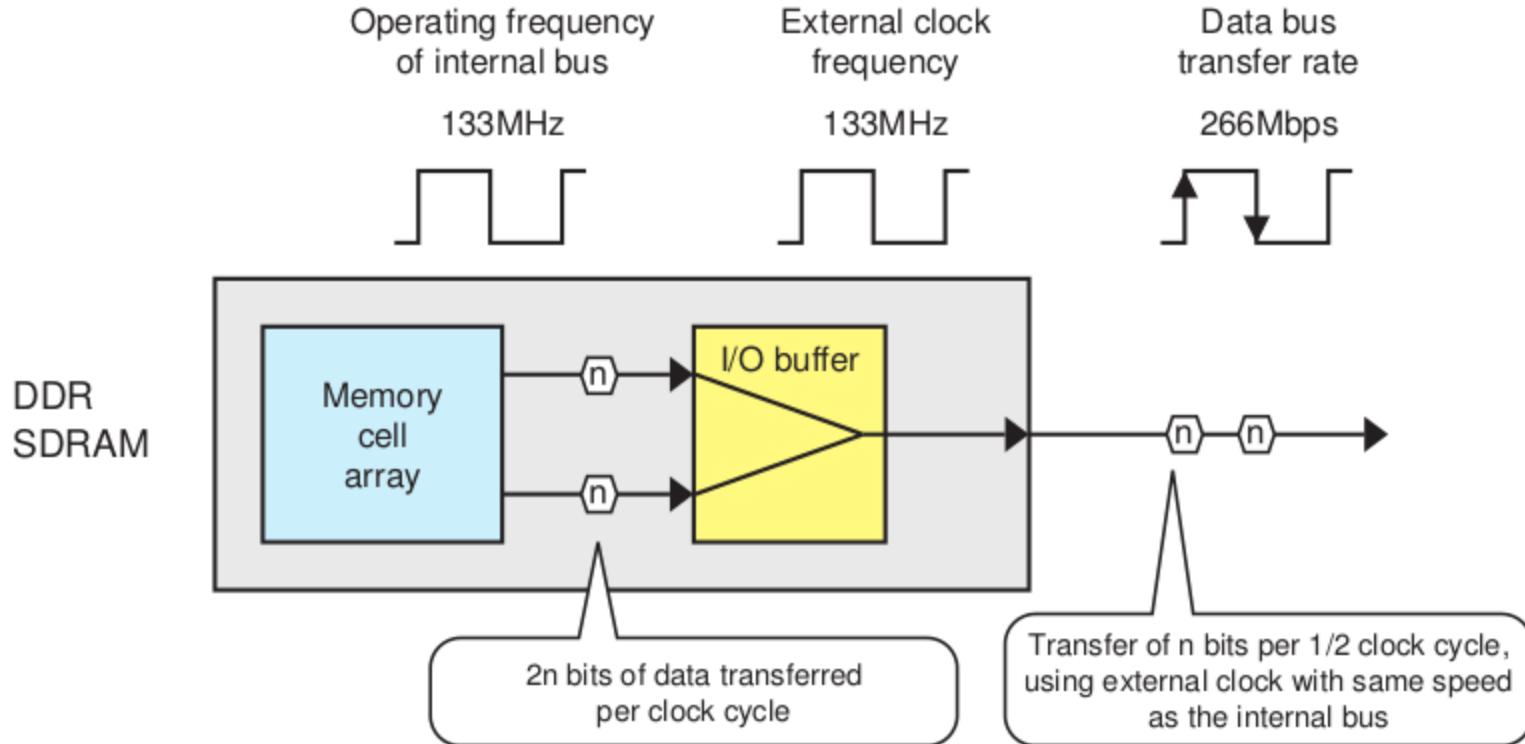


Сравнение DDR и DDR2: SDR SDRAM



Document No. E0437E40 (Ver.4.0)
Date Published September 2007 (K) Japan
URL: <http://www.elpida.com>

Сравнение DDR и DDR2: DDR SDRAM



Document No. E0437E40 (Ver.4.0)
Date Published September 2007 (K) Japan
URL: <http://www.elpida.com>

Сравнение DDR и DDR2: DDR2 SDRAM

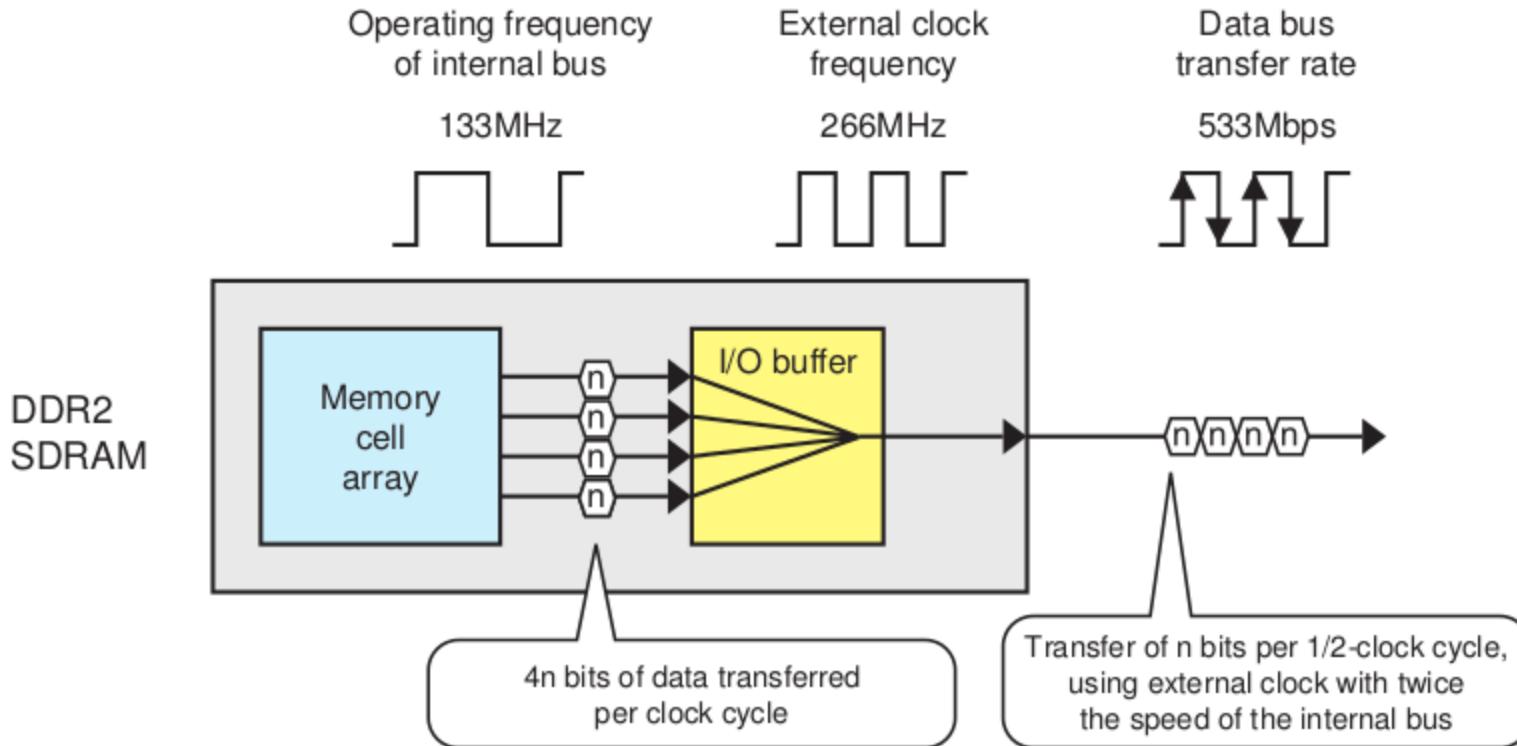
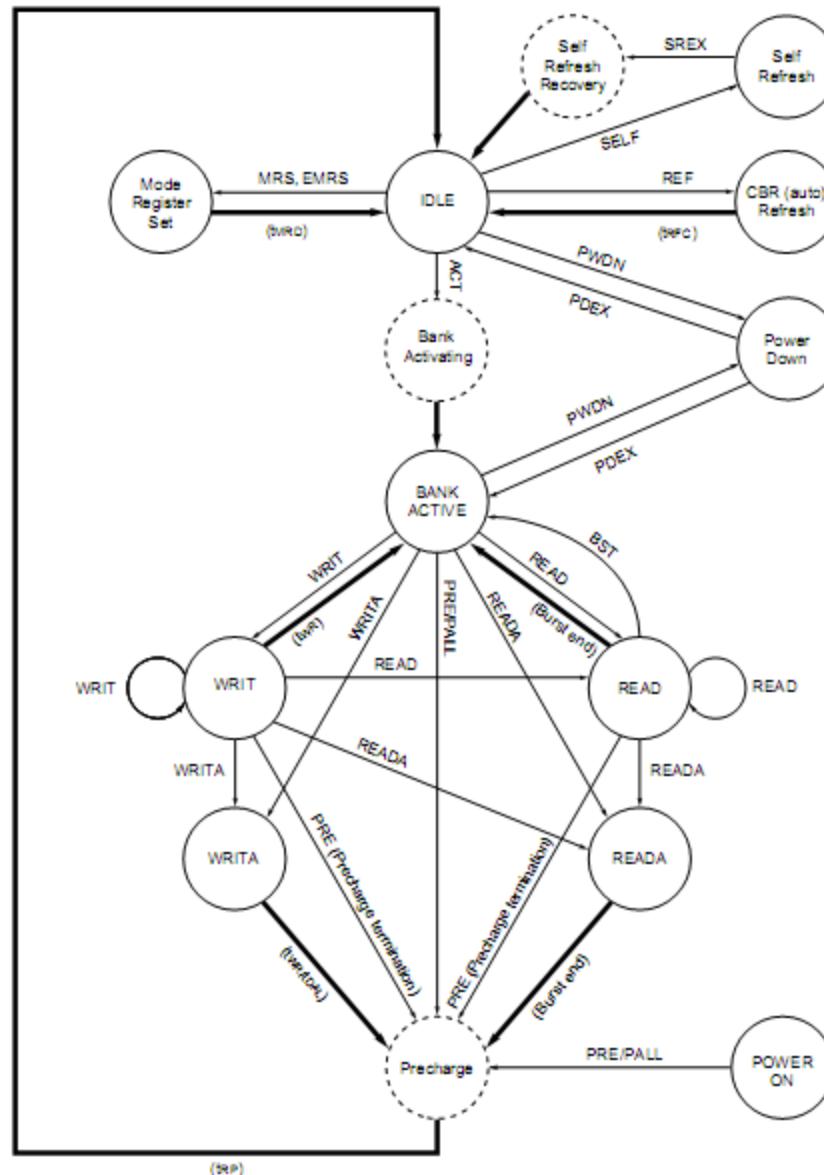


Диаграмма состояний УА DDR SDRAM

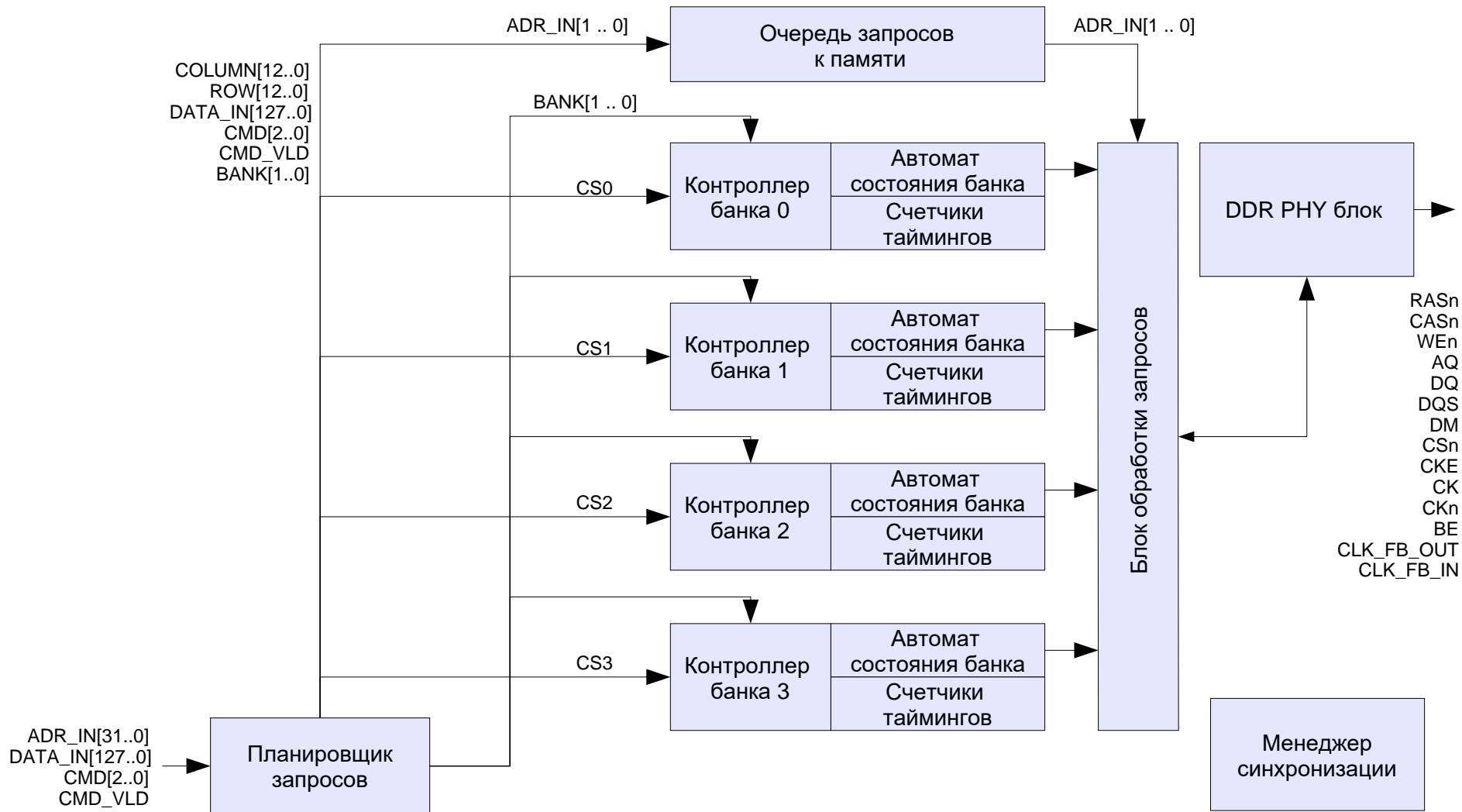


Организация Э

Automatic sequence
Manual input

Document No. E0234E30 (Ver.3.0)
Date Published April 2002 (K) Japan
URL: <http://www.elpida.com>

Контроллер DDR/DDR2



Отличие DDR и SDR DRAM

Item	DDR SDRAM	SDR SDRAM
Data transfer frequency	Twice the operation frequency	Same as the operation frequency
Data rate	2/tck	1/tck
Clock input	Differential clock	Single clock
Data strobe signal (DQS)	Essential	Not supported
Interface	SSTL_2	LVTTL
Supply voltage	2.5 V	3.3 V
/CAS read latency	2, 2.5	2, 3
/CAS write latency	1	0
Burst length	2, 4, 8	1, 2, 4, 8, full-page (256) <small>Note</small>
Burst sequence	Sequential/Interleave	Sequential/Interleave
Use of DLL	Essential	Option
Data mask	Write mask only	Write mask/Read mask

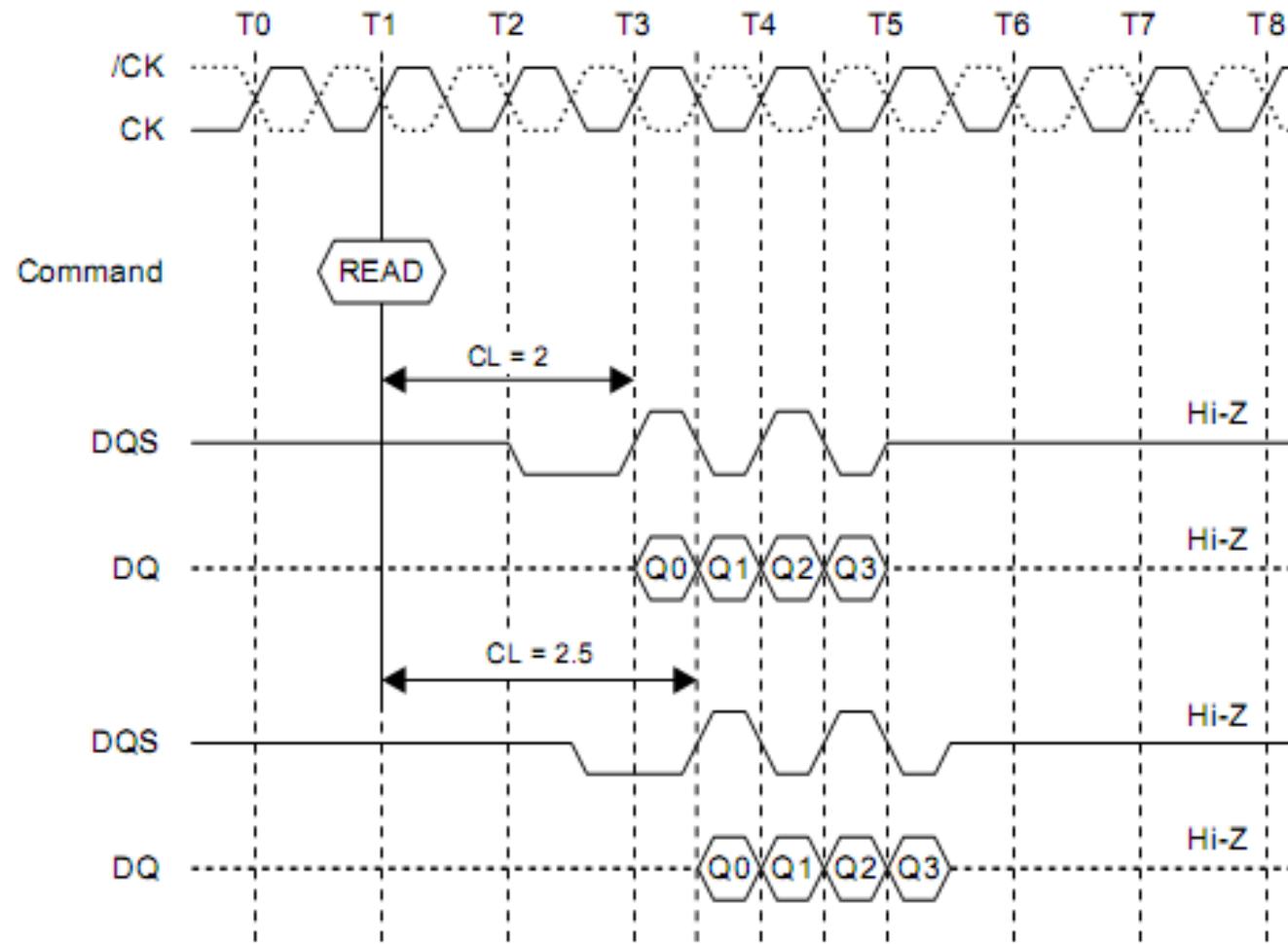
Тайминг памяти: tCL-tRCD-tRP-tRAS

- CAS Latency (tCL) - задержка в тактах между подачей сигнала CAS и непосредственно выдачей данных из соответствующей ячейки. Одна из важнейших характеристик любого модуля памяти;
- RAS to CAS Delay (tRCD) - количество тактов шины памяти, которые должны пройти после подачи сигнала RAS до того, как можно будет подать сигнал CAS;
- Row Precharge (tRP) - время закрытия страницы памяти в пределах одного банка, тратящееся на его перезарядку;
- Activate to Precharge (tRAS) - время активности строба. Минимальное количество циклов между командой активации (RAS) и командой подзарядки (Precharge), которой заканчивается работа с этой строкой, или закрытия одного и того же банка.

Примеры таймингов памяти DDR: 2-2-2-5; 2.5-3-3-7

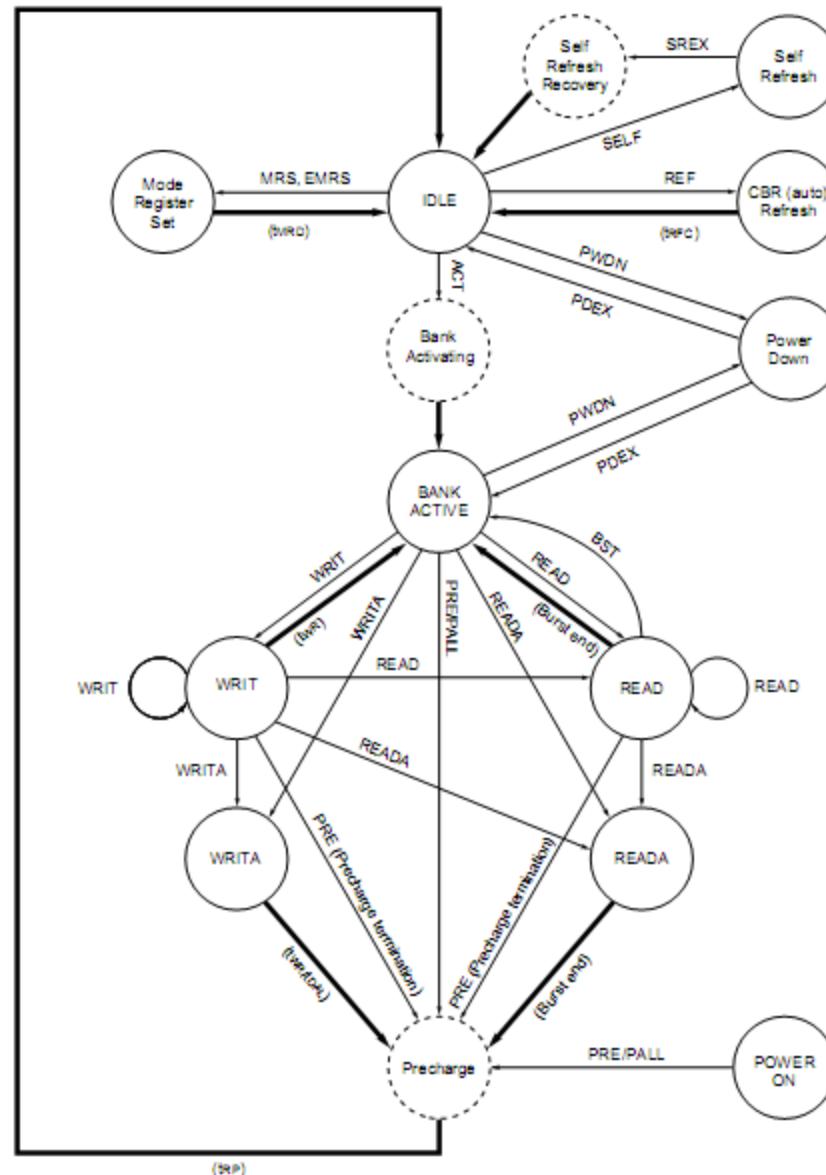
Примеры таймингов памяти DDR2: 3-3-3-9, 4-4-4-12 и 5-5-5-15

Сравнение DDR SDRAM CL=2 и CL=2.5



Document No. E0234E30 (Ver.3.0)
Date Published April 2002 (K) Japan
URL: <http://www.elpida.com>

Диаграмма состояний UA DDR SDRAM



Automatic sequence
Manual input

Организация Э

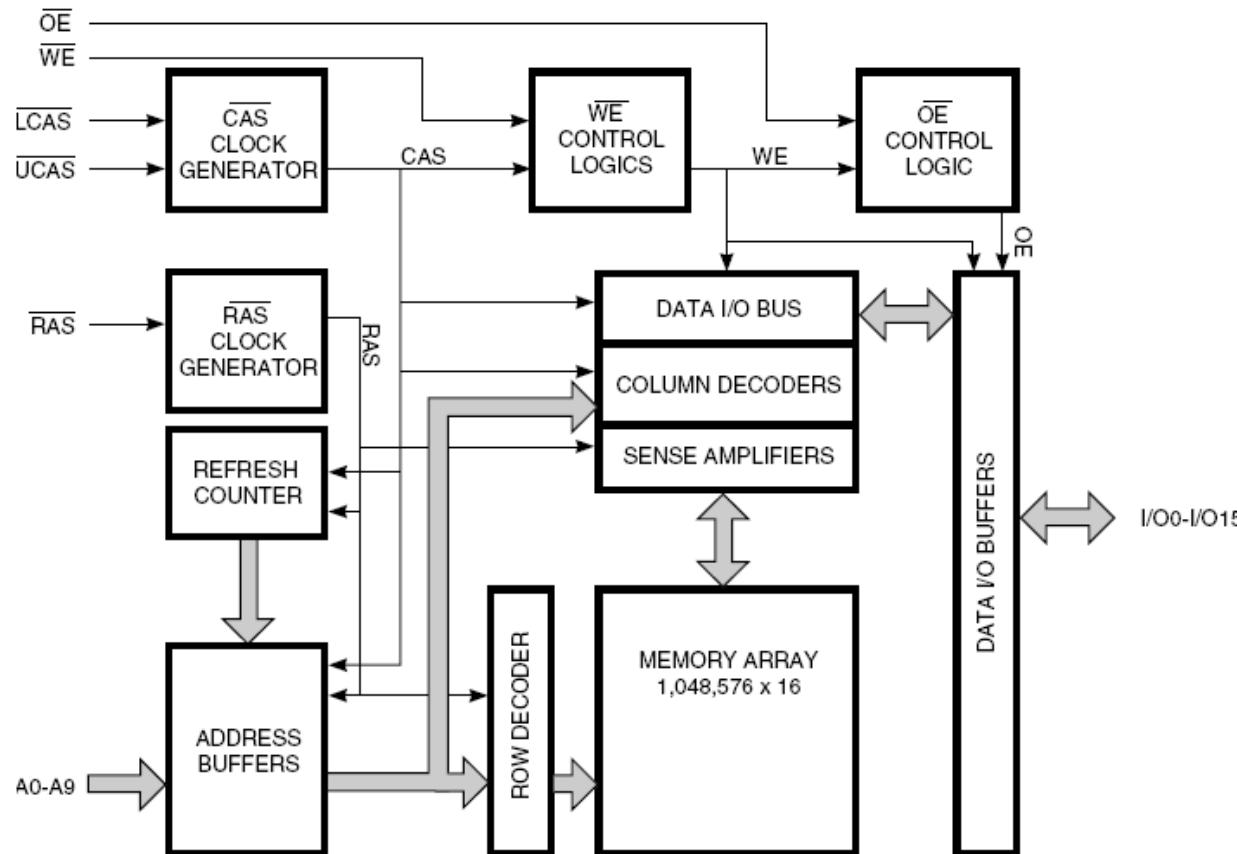
Команды DDR SDRAM

Command	Symbol	CKE		/CS	/RAS	/CAS	/WE	Address			
		n=1	n					B40	B41	A10	A0-A9, A11
Device deselect	DESL	H	X	H	X	X	X	X	X	X	X
No operation	NOP	H	X	L	H	H	H	X	X	X	X
Burst stop	BST	H	X	L	H	H	L	X	X	X	X
Read	READ	H	X	L	H	L	H	V		L	V
Read with auto precharge	READA							V		H	
Write	WRIT	H	X	L	H	L	L	V		L	V
Write with auto precharge	WRITEA							V		H	
Bank active	ACT	H	X	L	L	H	H	V	V	V	V
Precharge selected bank	PRE	H	X	L	L	H	L	V	L	X	
Precharge all banks	PALL							X	H	X	
Mode register set	MRS	H	X	L	L	L	L	L	L	L	V
Extended mode register set	EMRS							H	L	L	V
CBR (auto) refresh	REF	H	H	L	L	L	H	X	X	X	X
Self refresh entry	SELF	H	L								
Self refresh exit	SREX	L	H	H	X	X	X	X	X	X	X
				L	H	H	X	X	X	X	X
Power down entry	PWDN	H	L	H	X	X	X	X	X	X	X
				L	H	H	X	X	X	X	X
Power down exit	PDEX	L	H	H	X	X	X	X	X	X	X
				L	H	H	X	X	X	X	X

Document No. E0234E30 (Ver.3.0)
 Date Published April 2002 (K) Japan
 URL: <http://www.elpida.com>

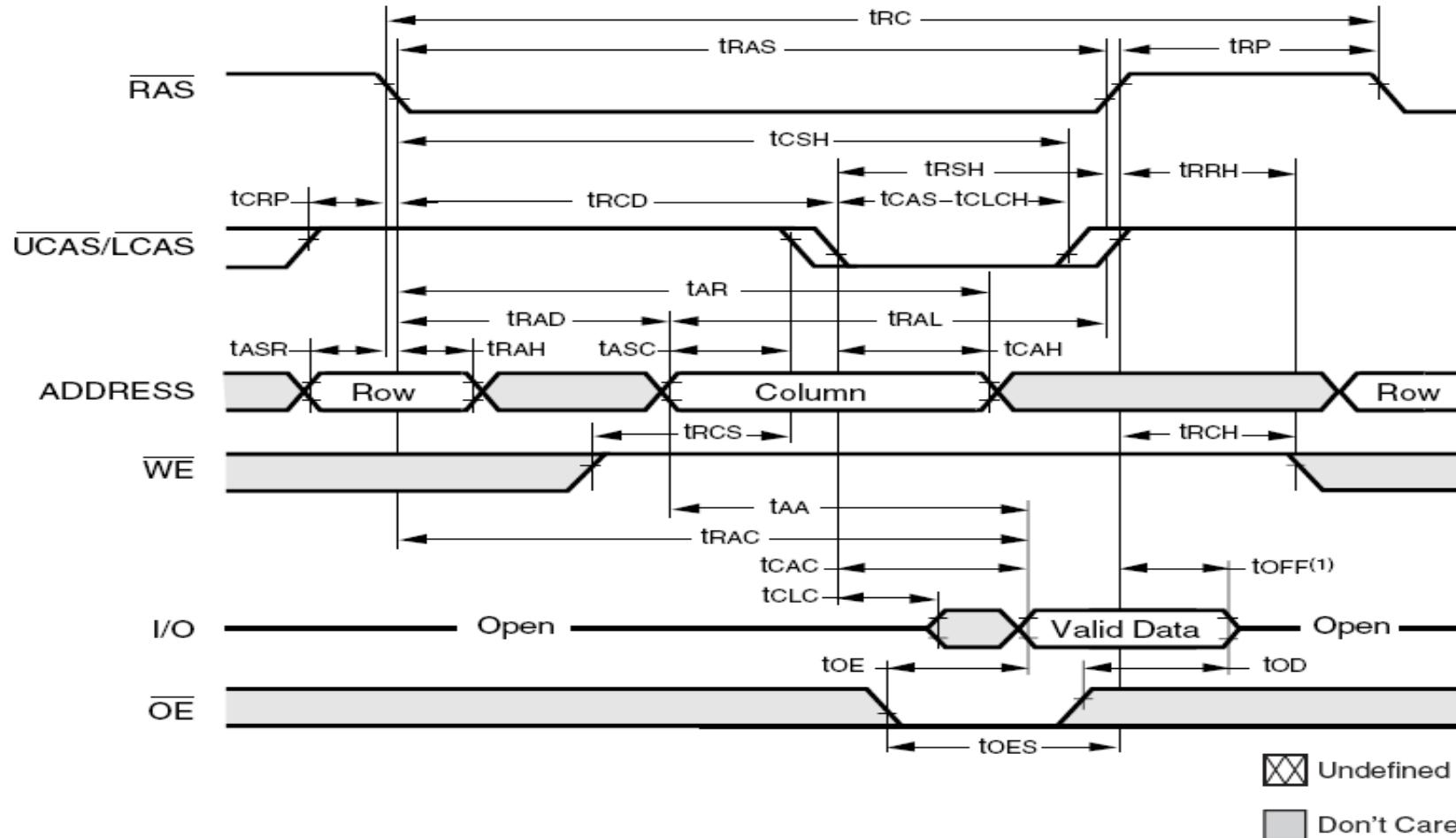
1M x 16 (16-MBIT) DYNAMIC RAM
WITH EDO PAGE MODE

December 2005



1M x 16 (16-MBIT) DYNAMIC RAM
WITH EDO PAGE MODE

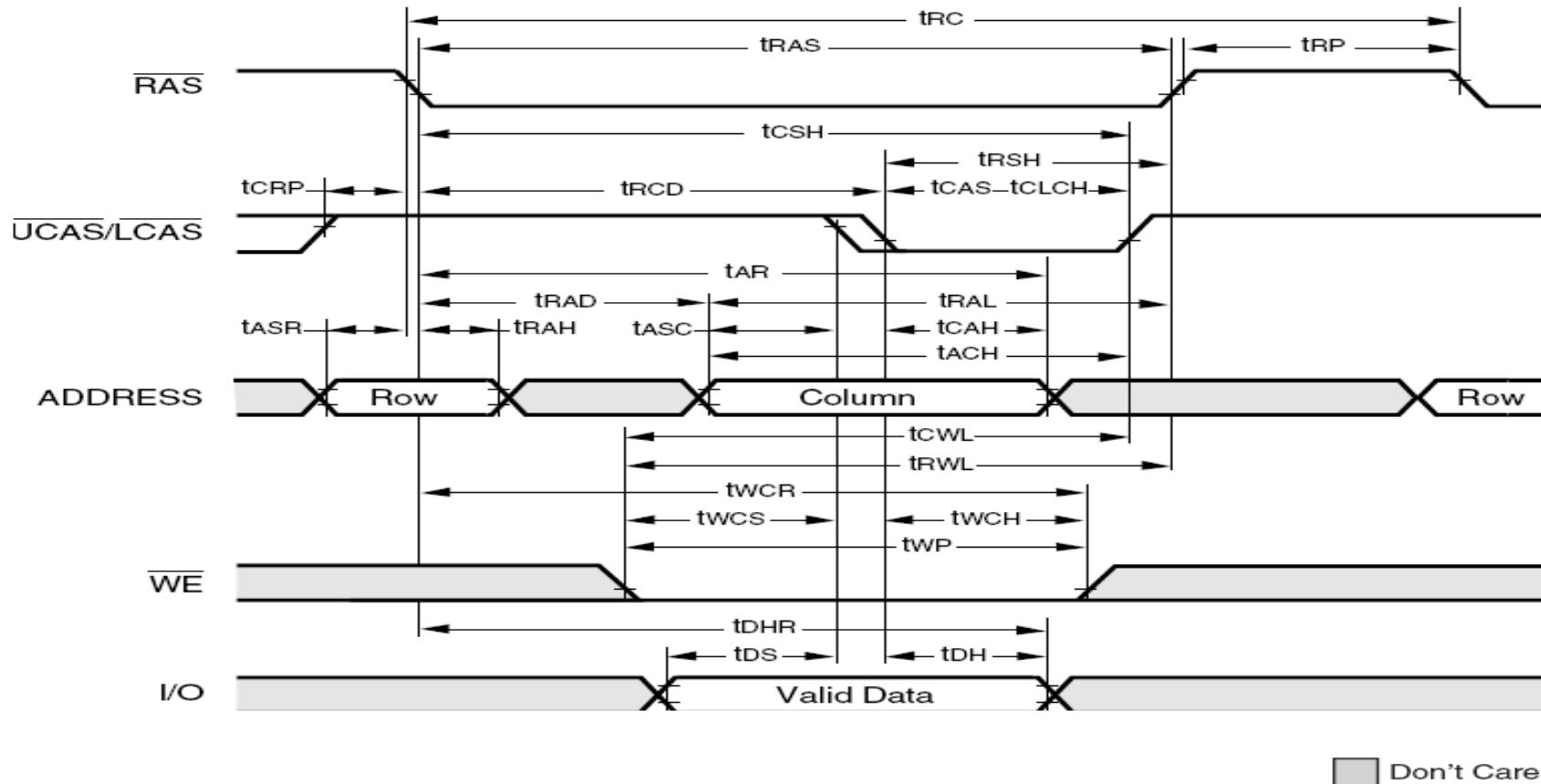
December 2005



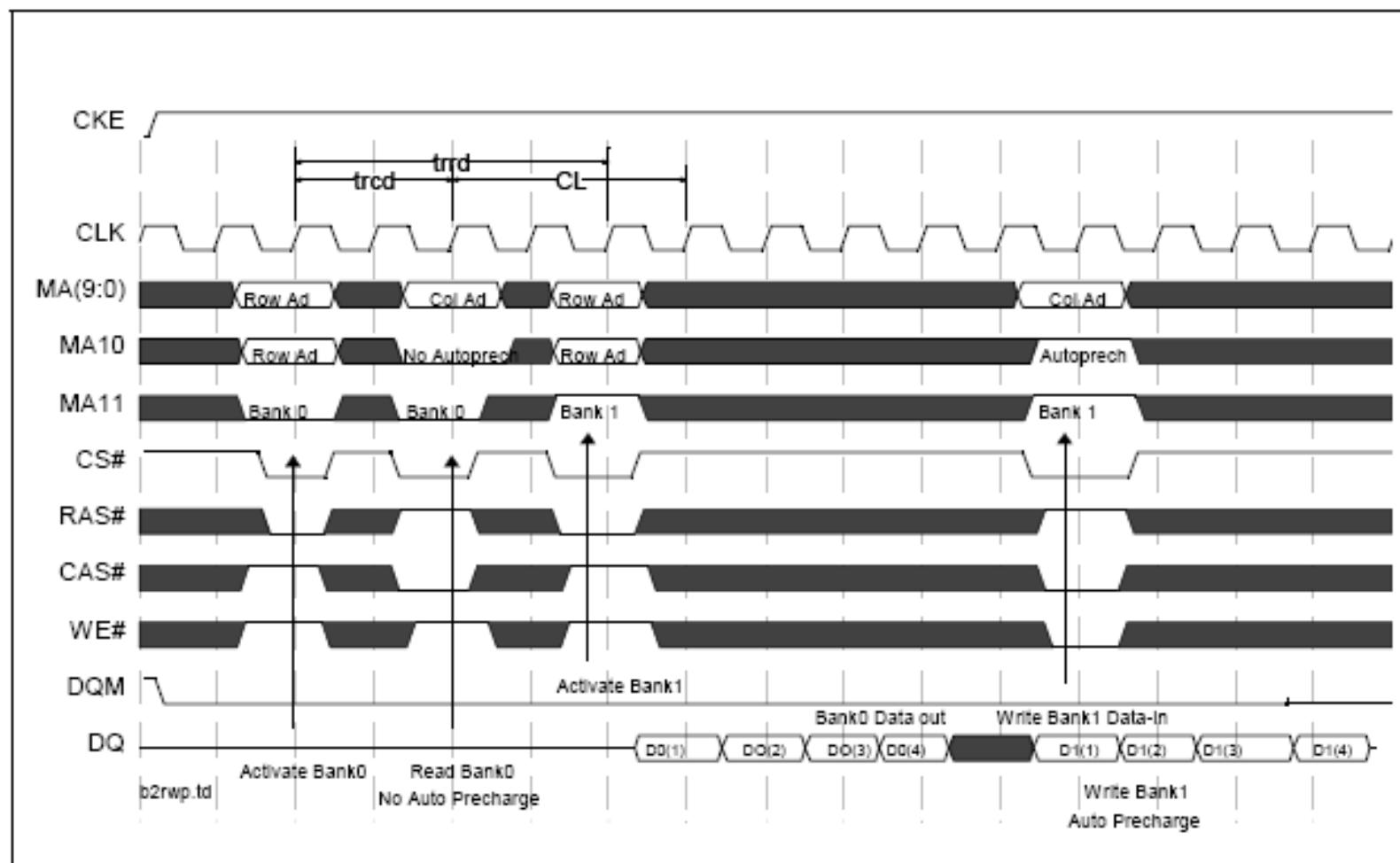
1M x 16 (16-MBIT) DYNAMIC RAM
WITH EDO PAGE MODE

December 2005

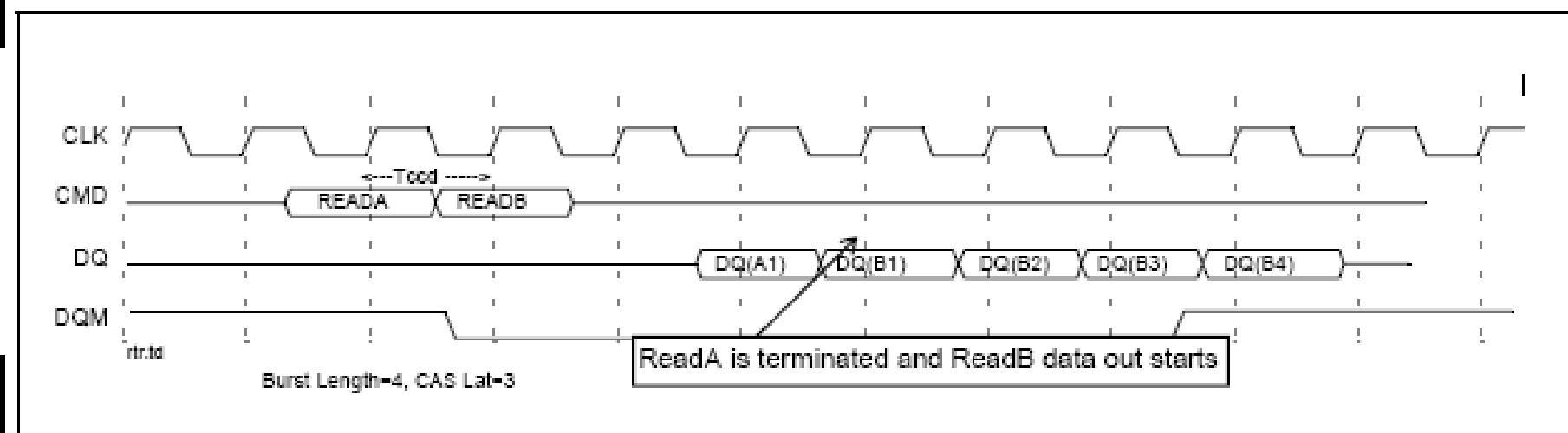
ПРИМЕР



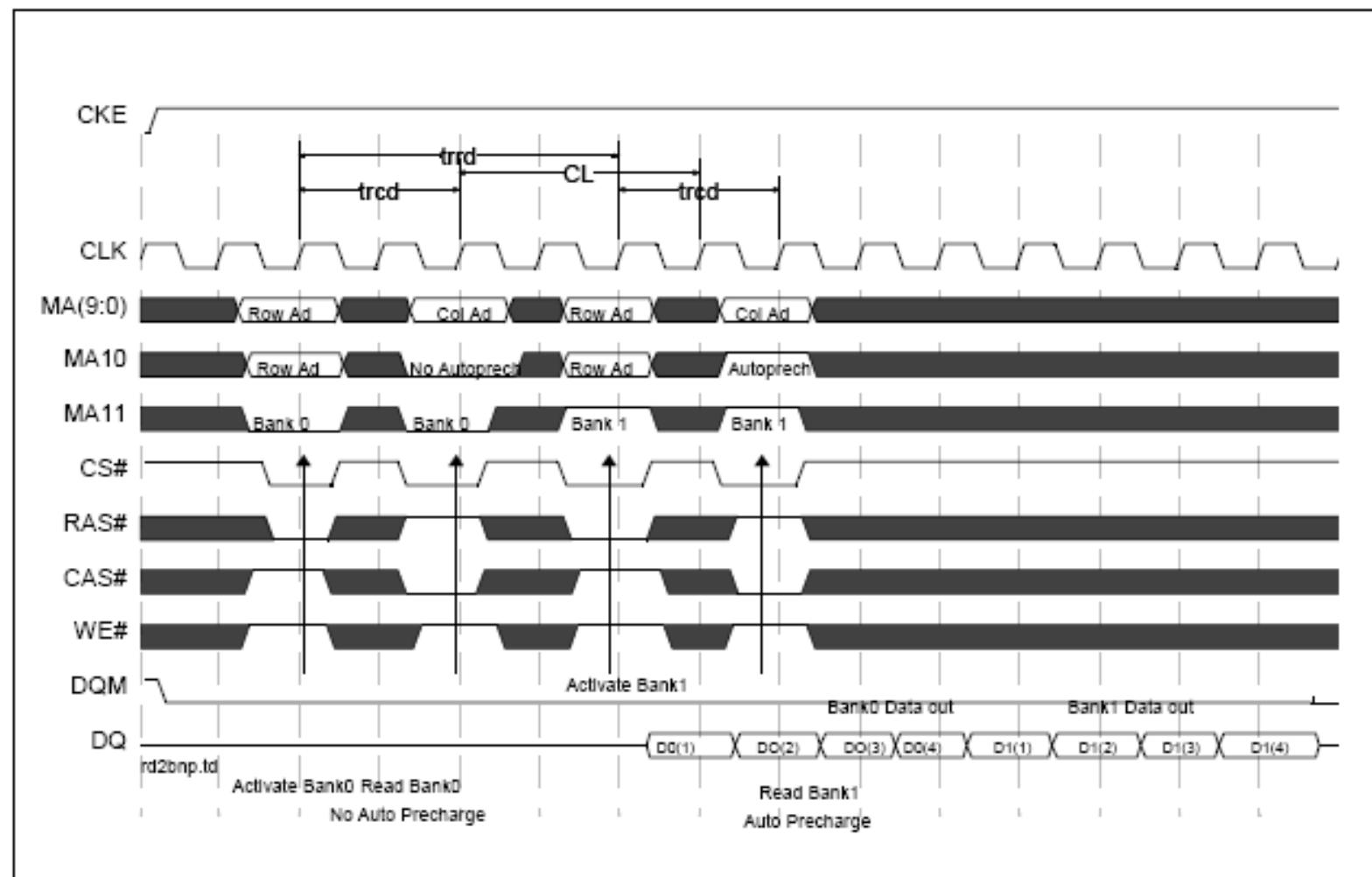
Read and Write Commands (Burst Length 4 Shown)



Read Terminated By Read



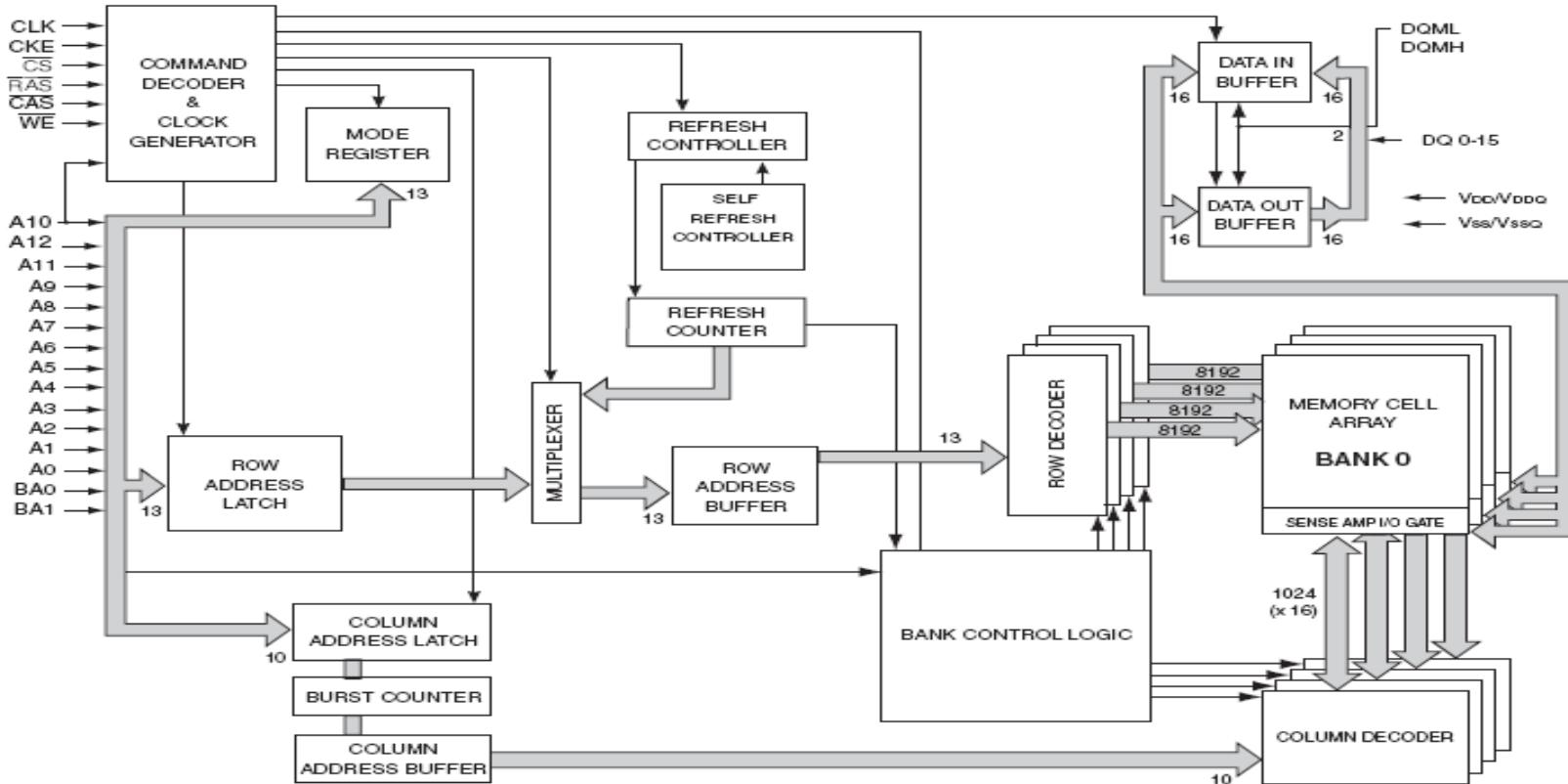
Two Bank Ping Pong Read



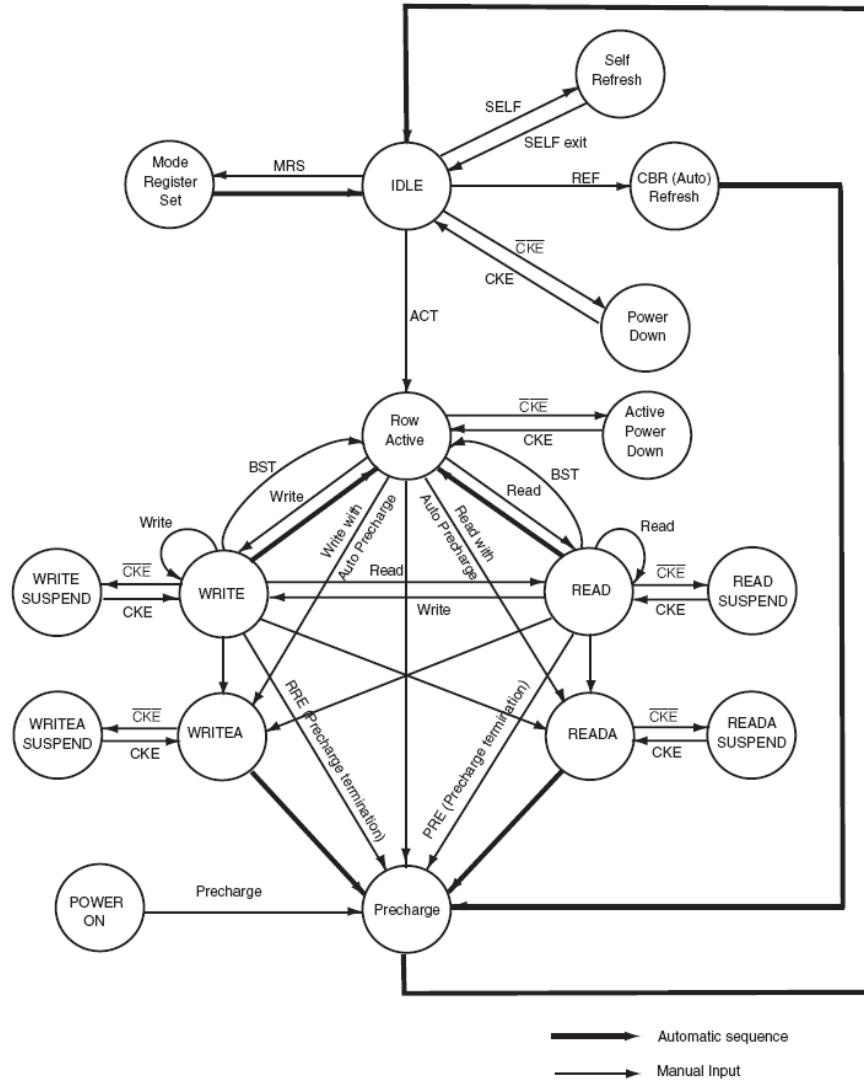
IS42S16320B

32Meg x 16
512-MBIT SYNCHRONOUS DRAM

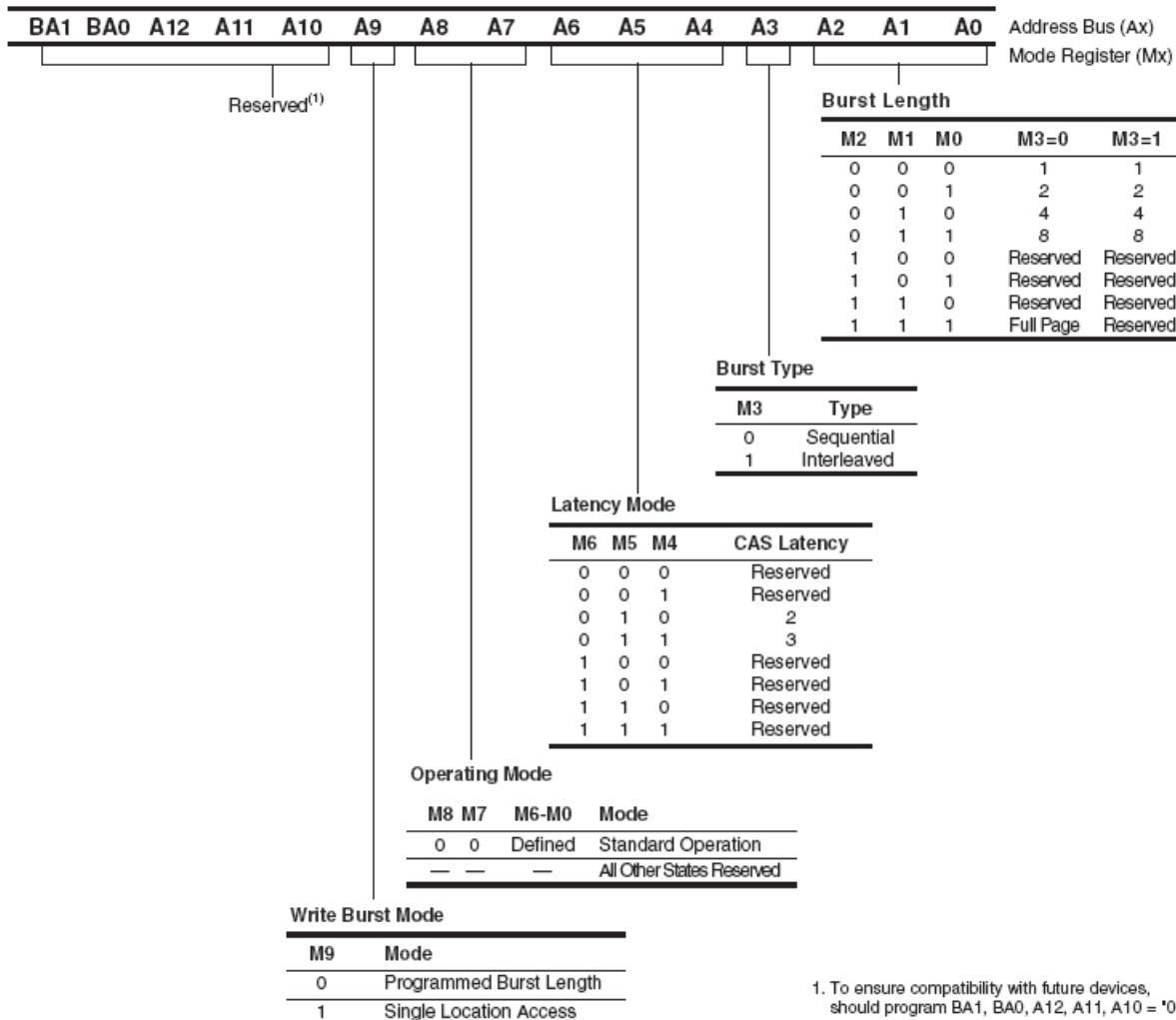
PRELIMINARY INFORMATION
JULY 2007

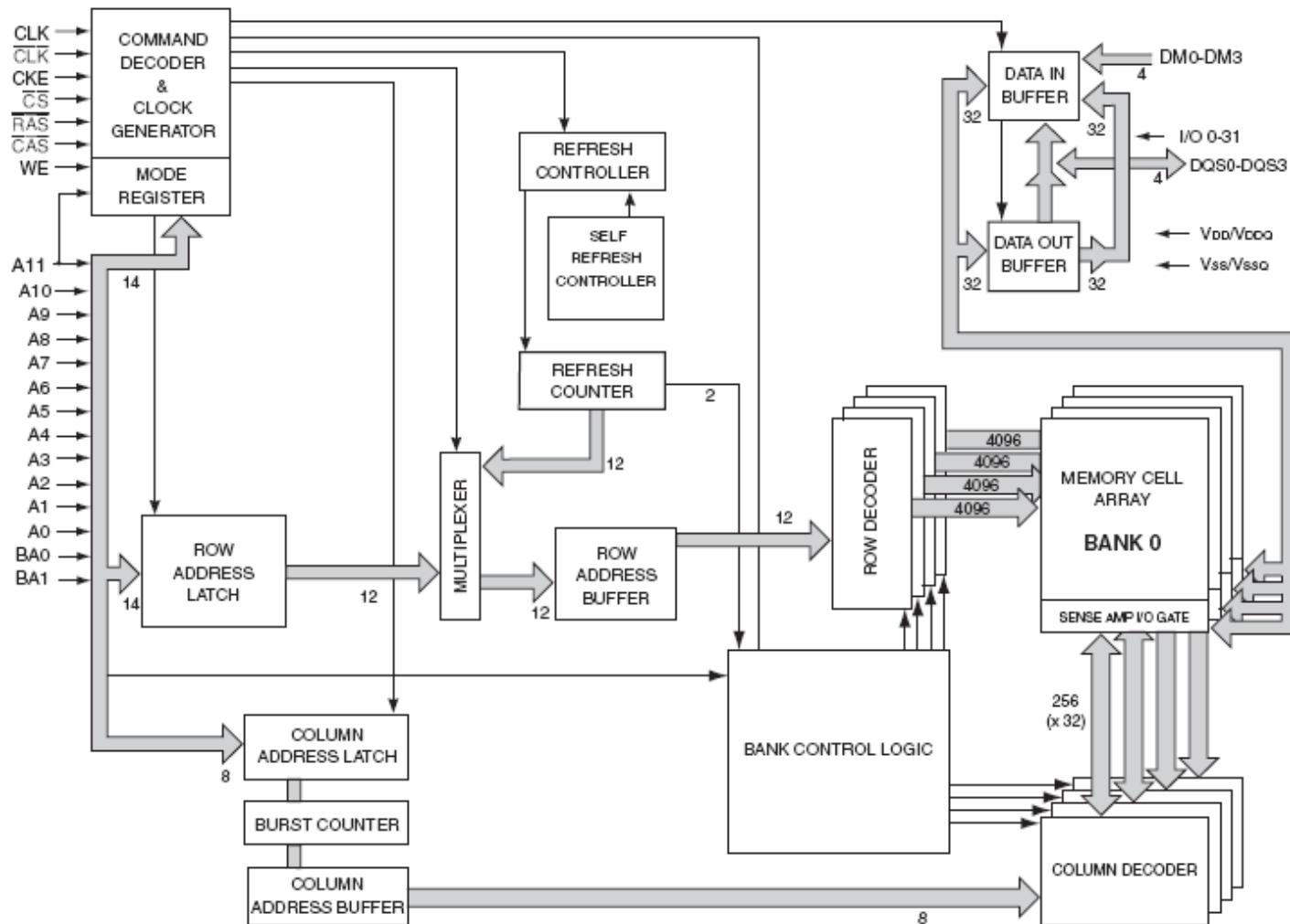


STATE DIAGRAM



IS42S16320B





Timing Waveforms

Figure 1. AC Parameters for Read Timing (Burst Length =4)

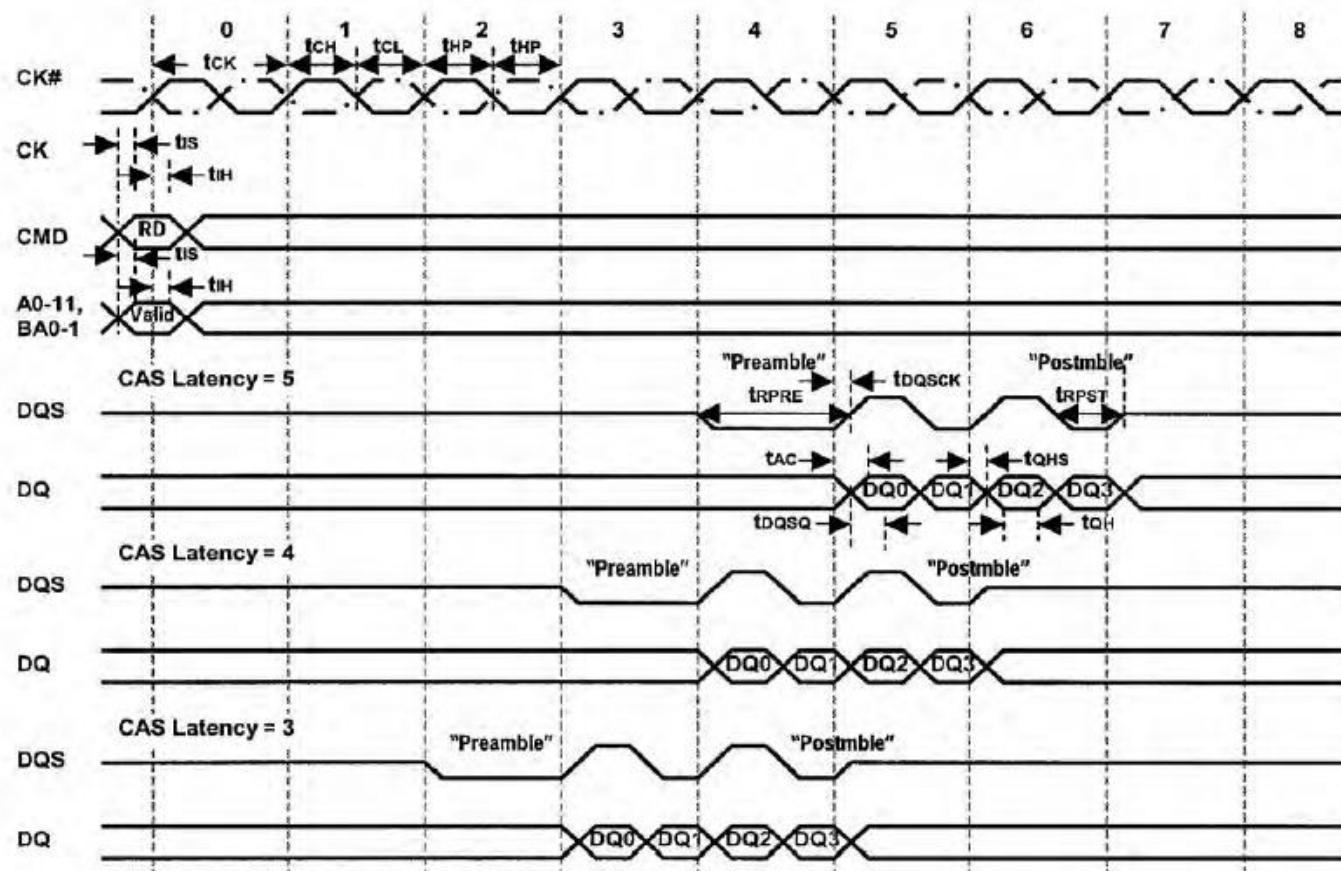
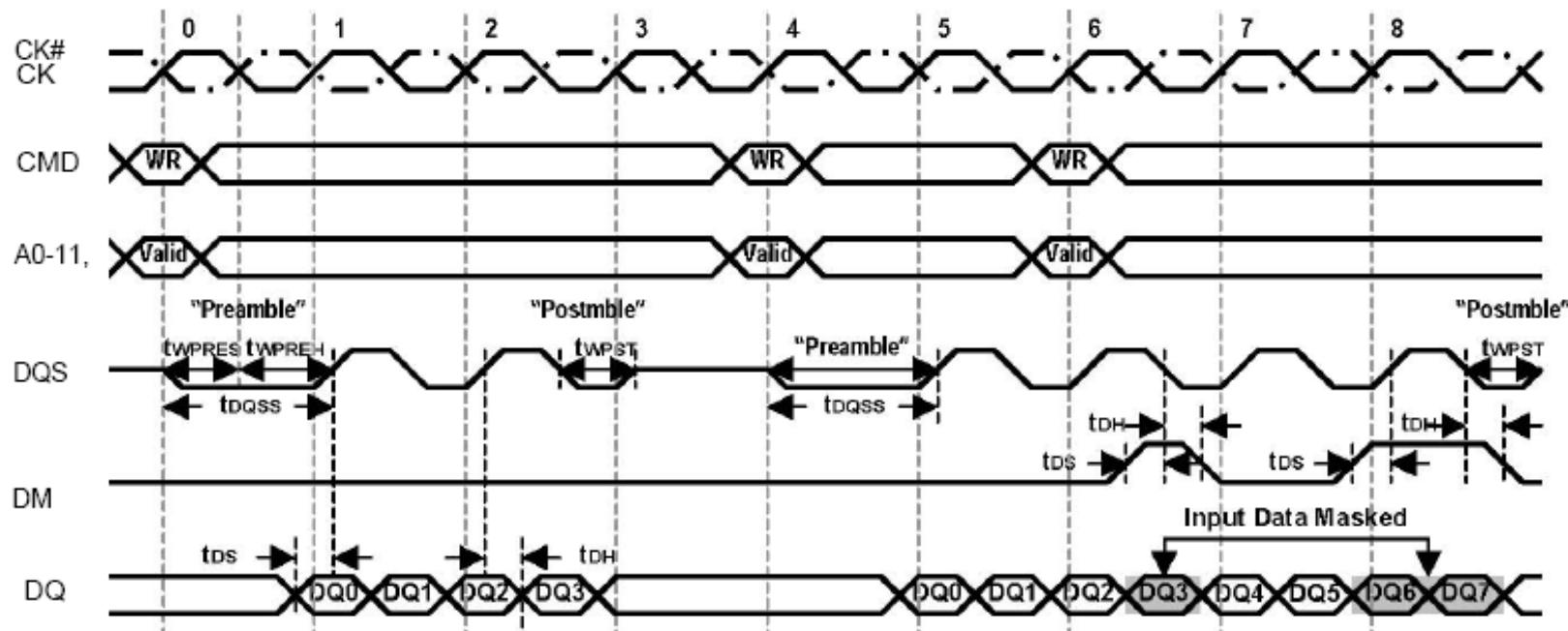


Figure 2. AC Parameters for Write Timing (Burst Length=4)



Постоянные запоминающие устройства

МПЗУ (MROM)

ППЗУ (PROM)

РПЗУ-УФ (EPROM)

ОПРПЗУ-УФ (EPROM-OTP)

РПЗУ-ЭС (EEPROM)

FLASH

NVRAM

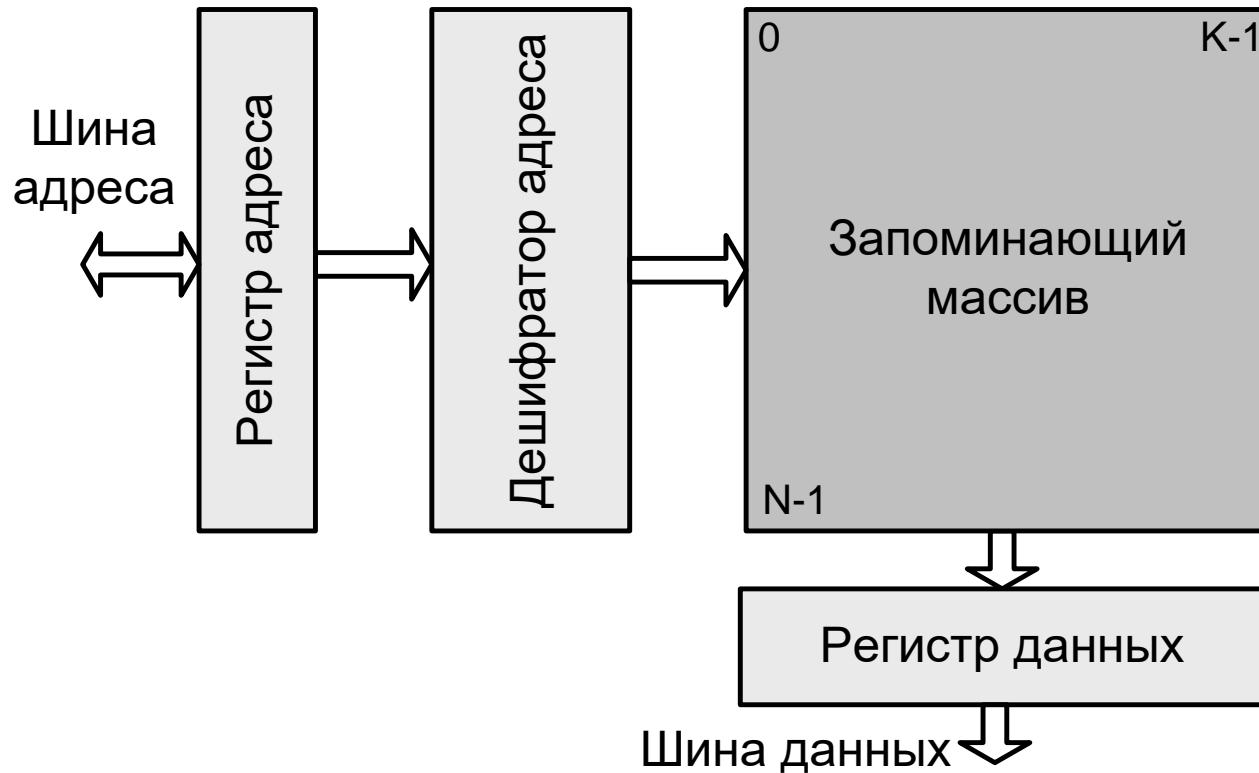
FRAM

MRAM

Преимущества ROM по сравнению RAM:

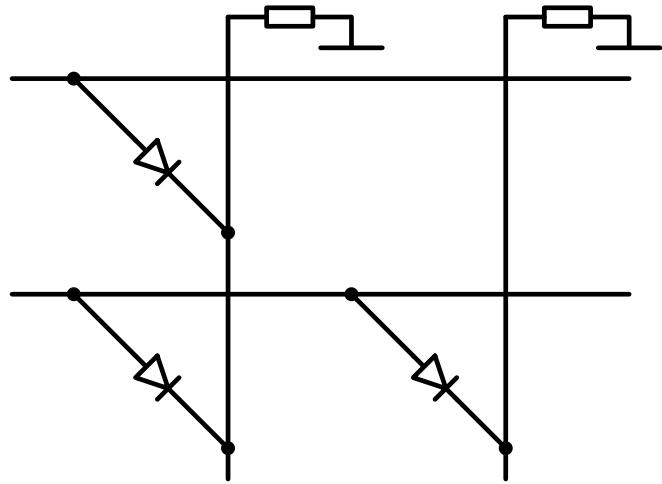
- Аппаратная простота.
- Высокая плотность размещения ЗЭ.
- Энергонезависимость.
- Большое быстродействие.

Структура ПЗУ (ROM)

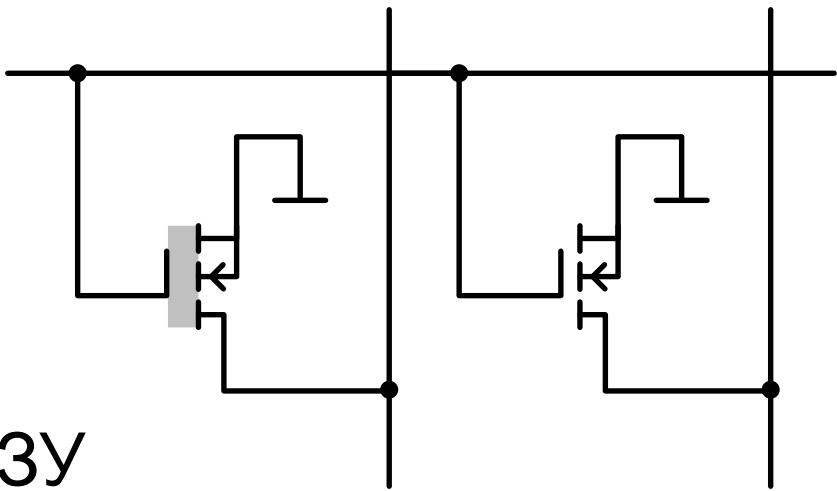


МПЗУ

ЗЭ на диодах

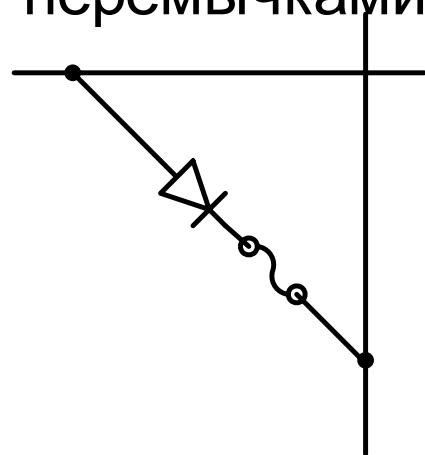


ЗЭ на МОП транзисторах

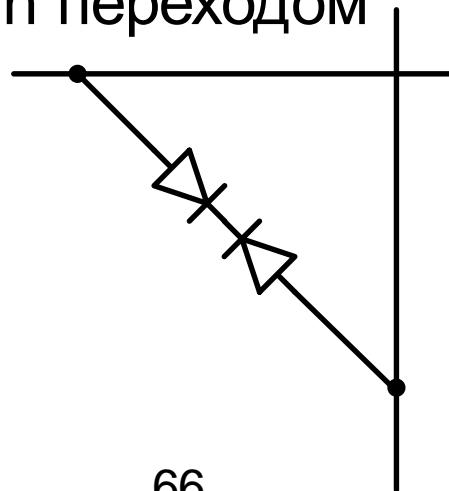


ППЗУ

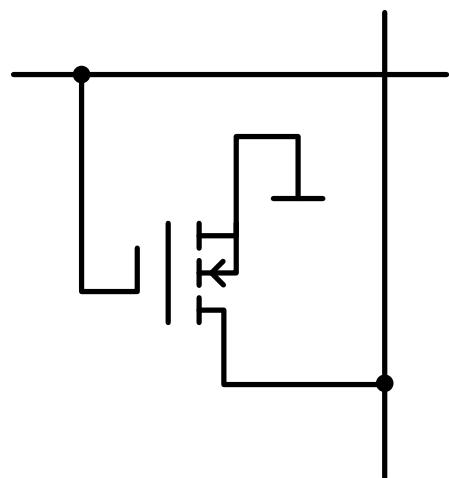
ППЗУ с плавкими
перемычками



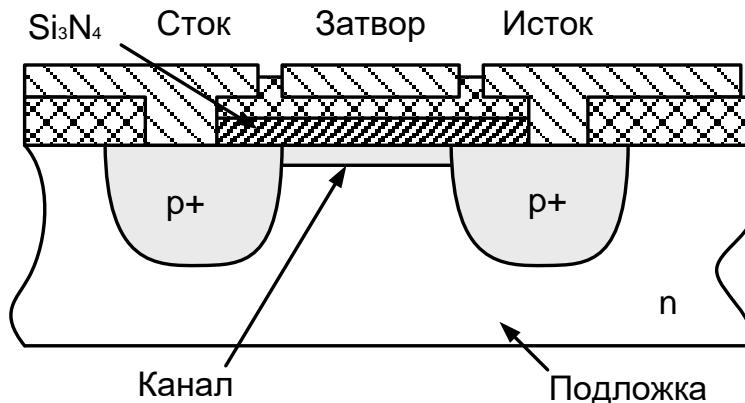
ППЗУ с переключаемым
р-n переходом



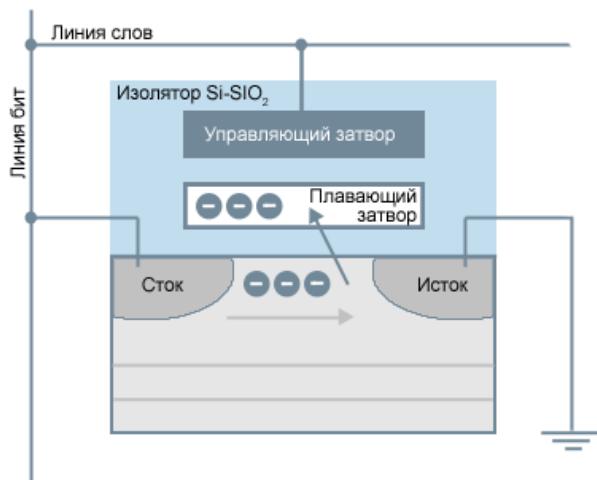
РПЗУ-УФ, ОПРРПЗУ-УФ (EPROM, EPROM-OTP)



МНОП транзистор

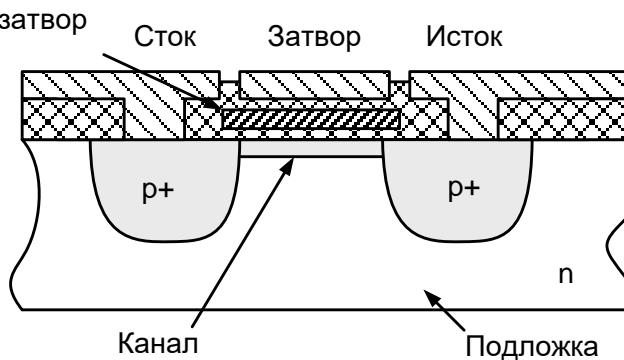


РПЗУ-ЭС (EEPROM), FLASH

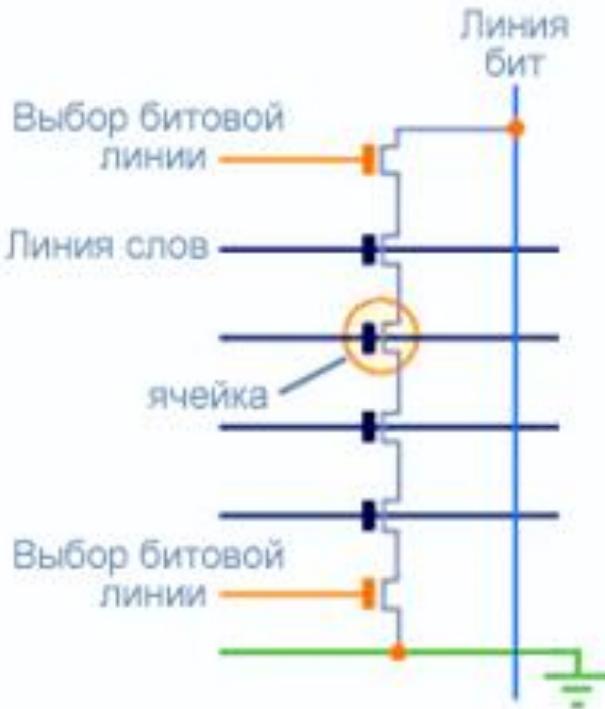


МОП транзистор с плавающим затвором

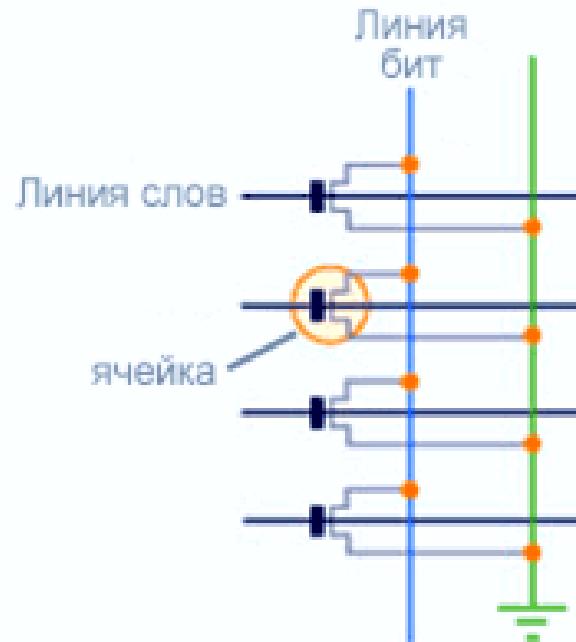
Плавающий затвор



NAND FLASH



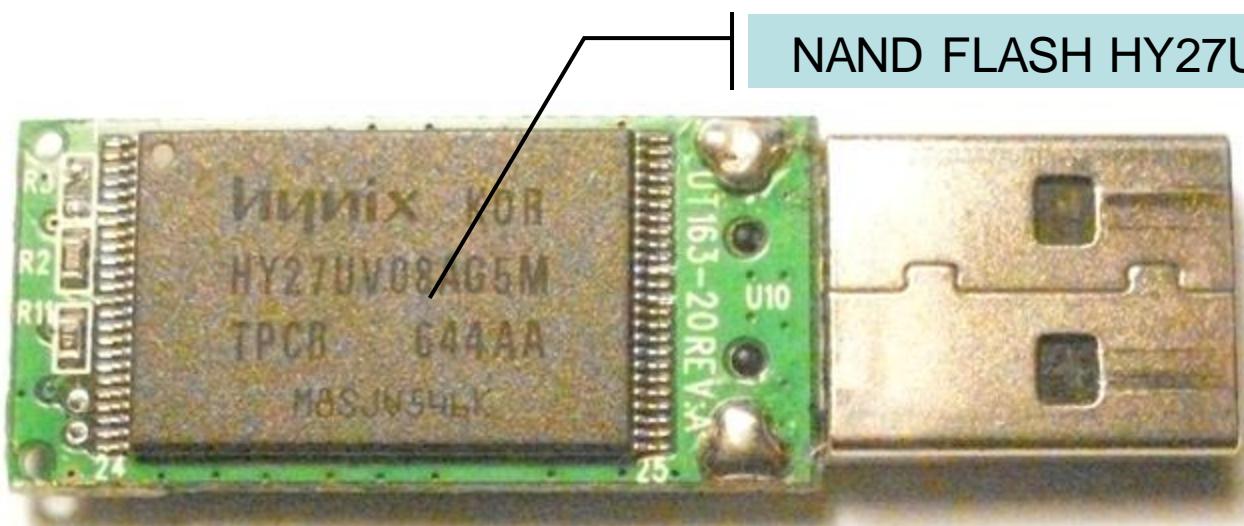
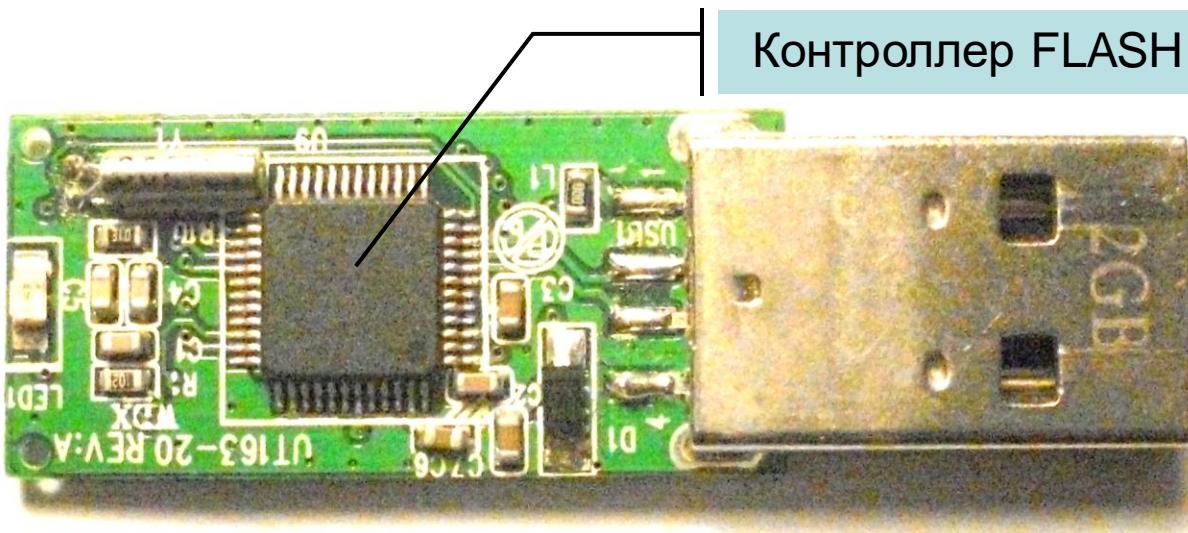
NOR FLASH



Все транзисторы, кроме адресуемого, должны быть открыты. Если на плавающем затворе есть заряд, то транзистор не откроется и на линии бит будет высокий уровень. В противном случае сигнал будет низкого уровня.
+ Большая компактность
- Меньшее быстродействие

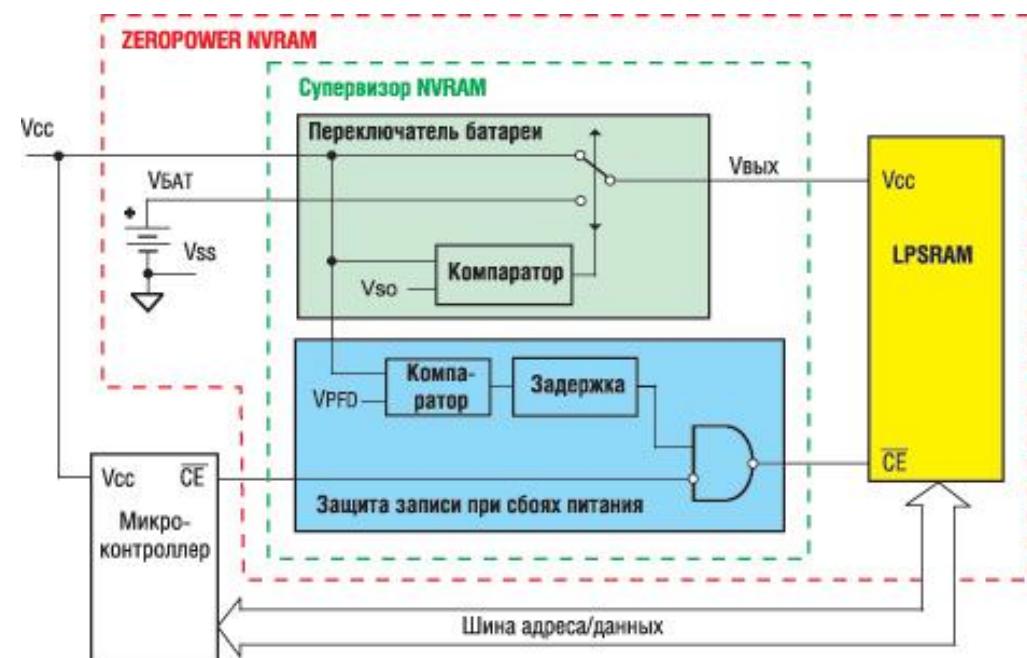
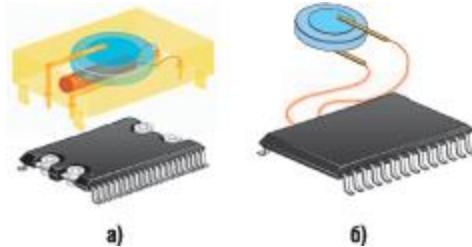
Линии слов невыбранных транзисторов находятся под низким потенциалом (транзисторы закрыты), на затворе выбранного транзистора высокий потенциал. Если на плавающем затворе выбранного транзистора есть заряд, то транзистор не откроется и на линии бит будет уровень лог. единицы.
- Меньшая компактность
+ Большее быстродействие

Накопитель на основе FLASH



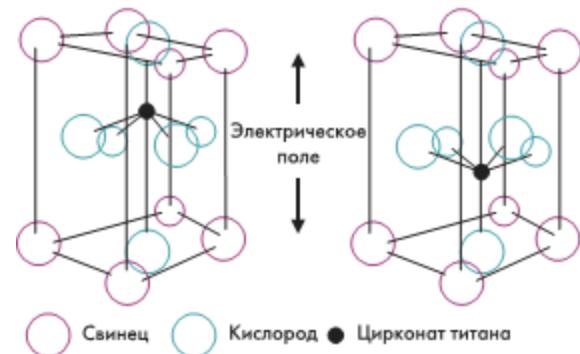
ПЗУ типа NVRAM

Энергонезависимая память NVRAM (Non-Volatile Random Access Memory) – это оперативная память LPSRAM (Low Power SRAM – статическое ОЗУ с очень низким потреблением), сохраняющая данные независимо от наличия основного питания благодаря наличию встроенной литиевой батареи для резервного питания. Интегрированная схема контроля и переключения на резервный источник питания (супервизор и коммутатор литиевой батареи) гарантирует работоспособность памяти NVRAM и сохранение данных в течение десяти лет при полном отсутствии внешнего питания.



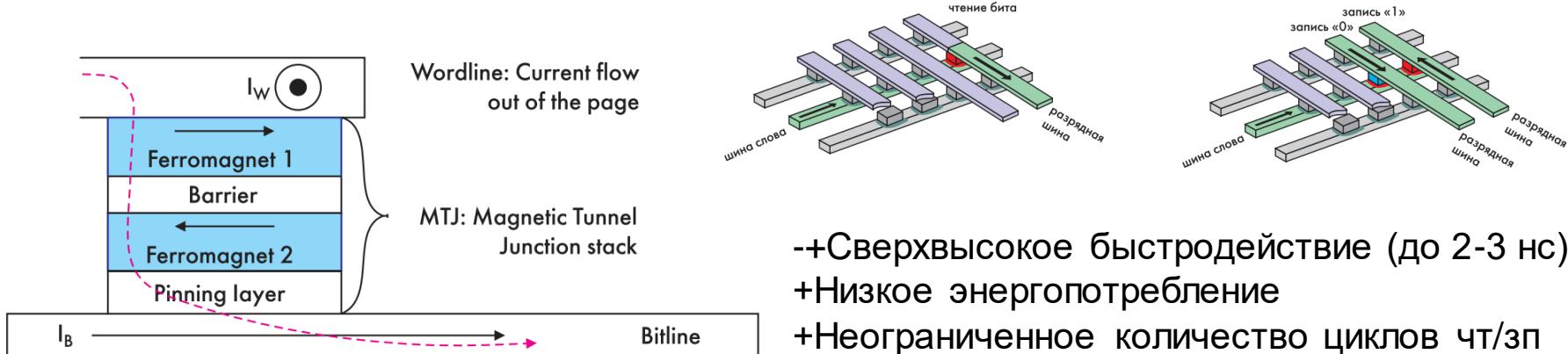
ПЗУ на основе сегнетоэлектрической пленки (FRAM)

- +Высокая скорость записи
- Необходимость восстановления информации при доступе



Основа запоминающего устройства FRAM — это конденсатор, представляющий собой две пластины с тонким слоем ферроэлектрика между ними. Приложенный к обкладкам конденсатора потенциал поляризует ферроэлектрик. Направление поляризации представляет собой двоичную информацию, хранящуюся в ячейке. При повторном приложении потенциала заряд, затрачиваемый на реполяризацию, будет зависеть от того, совпадает направление электрического поля с тем, которое поляризовало ферроэлектрик в прошлый раз, или нет. Если направление поля не совпадает, то на изменение поляризации потребуется значительный дополнительный заряд. Таким образом, если при повторном наложении потенциала наблюдается электрический ток, то направление не совпадает с предыдущим. По наличию или отсутствию тока перезаряда можно судить о содержимом ячейки.

Магниторезистивные ПЗУ (MRAM)



- +Сверхвысокое быстродействие (до 2-3 нс)
- +Низкое энергопотребление
- +Неограниченное количество циклов чт/зп

ПРИМЕР

Проводимость магниторезистивного слоя зависит от магнитного поля, в которое он помещен. Внутри запоминающего элемента MRAM сопротивление находящегося в нем магниторезистивного материала будет определяться ориентацией магнитных моментов ферромагнитных слоев. В одном из магнитных слоев домены фиксированы в одном направлении. В другом слое они в ответ на воздействие внешнего поля могут быть развернуты в противоположном направлении. В результате они могут быть либо параллельны, либо антипараллельны элементам фиксированного слоя. Эти два состояния запоминают «1» или «0».

Таблица 1. Сравнительные характеристики MRAM, выполненных с различными нормами, и других типов встраиваемой памяти

Характеристика	MRAM			Флэш	СОЗУ	ДОЗУ	FRAM
	0,6 мкм	0,18 мкм	90 нм	90 нм	90 нм	90 нм	90 нм
Объем, Мбит	256 Кбит – 1	1–32	4–256	4–64	4–64	16–256	4–64
Диаметр пластины, мм	150/200	200	200/300	200/300	200/300	200/300	200/300
Быстродействие, МГц	16	50–100	75–125	20–100 (при считывании)	50–2000	20–100	15–50
Эффективность использования матрицы, %	40–60	40–60	25–40	50–80	40	40–60	40–60
Напряжение, В	3,3	3,3/1,8	2,5/1,2	2,5/1,2; 9–12 (внутреннее)	2,5/1,2	2,5/1,2	2,5/1,2
Увеличение стоимости КМОП-технологии, %	–	15–25	15–25	25	0	15	15–25
Площадь ячейки, мкм ²	7,2	0,7–1	0,15–0,25	0,2–0,25	1–1,3	0,25	0,4
Площадь блока, мм ² /Мбит	12,0	2–3	0,3–0,5	0,6–1	1,2–1,7	0,6	0,8
Рабочий ресурс, число циклов перезаписи	$>10^{15}$	$>10^{15}$	$>10^{15}$	$>10^{15}$ (считывание), $<10^6$ (запись)	$>10^{15}$	$>10^{15}$	$>10^{13}$ (считывание/ запись)
Энергонезависимость	+	+	+	+	–	–	+

М.Валентинова

ПОЛУПРОВОДНИКОВАЯ ЭНЕРГОНЕЗАВИСИМНАЯ ПАМЯТЬ



IS93C76A IS93C86A

8K-BIT/16K-BIT SERIAL ELECTRICALLY ERASABLE PROM

MAY 2007

FEATURES

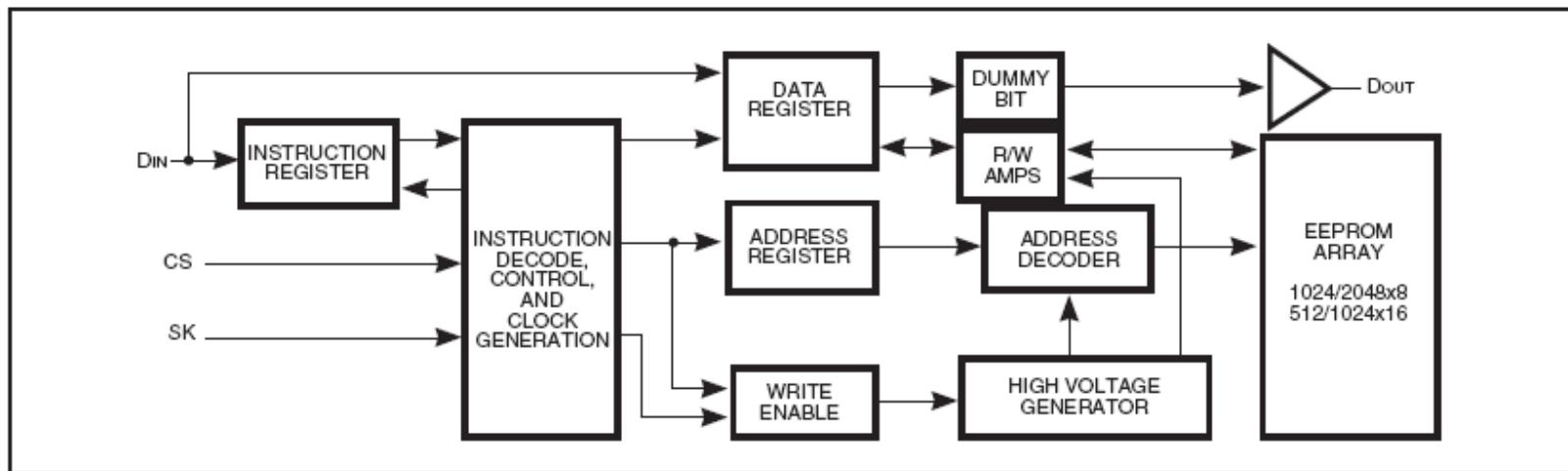
- Industry-standard Microwire Interface
 - Non-volatile data storage
 - Wide voltage operation:
 $V_{cc} = 1.8V$ to $5.5V$
 - Auto increment for efficient data dump
- User Configured Memory Organization
 - By 16-bit or by 8-bit
- Hardware and software write protection
 - Defaults to write-disabled state at power-up
 - Software instructions for write-enable/disable
- Enhanced low voltage CMOS E²PROM technology
- Versatile, easy-to-use Interface
 - Self-timed programming cycle
 - Automatic erase-before-write
 - Programming status indicator
 - Word and chip erasable
 - Chip select enables power savings
- Durable and reliable
 - 40-year data retention after 1M write cycles
 - 1 million write cycles
 - Unlimited read cycles
 - Schmitt-trigger Inputs
- Industrial and Automotive Temperature Grade
- Lead-free available

IS93C76A IS93C86A

8K-BIT/16K-BIT SERIAL ELECTRICALLY ERASABLE PROM

MAY 2007

FUNCTIONAL BLOCK DIAGRAM



IS93C76A IS93C86A

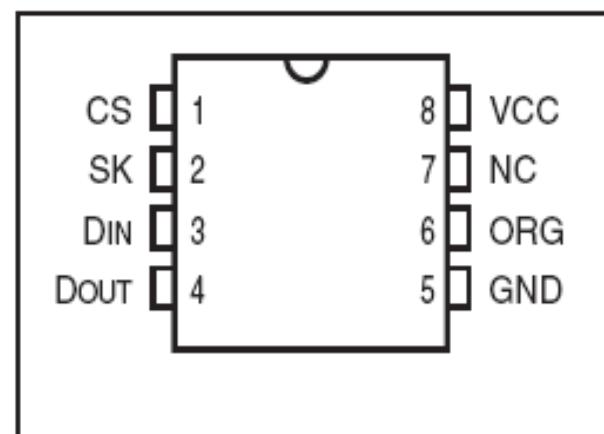
8K-BIT/16K-BIT SERIAL ELECTRICALLY ERASABLE PROM

MAY 2007

PIN DESCRIPTIONS

CS	Chip Select
SK	Serial Data Clock
D _{IN}	Serial Data Input
D _{OUT}	Serial Data Output
ORG	Organization Select
NC	Not Connected
Vcc	Power
GND	Ground

8-Pin DIP, 8-Pin TSSOP



IS93C76A IS93C86A**8K-BIT/16K-BIT SERIAL ELECTRICALLY
ERASABLE PROM**

MAY 2007

INSTRUCTION SET - IS93C86A (16kb)

Instruction ⁽²⁾	Start Bit	OP	Code	8-bit Organization (ORG = GND)		16-bit Organization (ORG = Vcc)	
				Address ⁽¹⁾	Input Data	Address ⁽¹⁾	Input Data
READ	1		10	(A10-A0)	—	(A9-A0)	—
WEN (Write Enable)	1		00	11x xxxx xxxx	—	11 xxxx xxxx	—
WRITE	1		01	(A10-A0)	(D7-Do)	(A9-A0)	(D15-Do)
WRALL (Write All Registers)	1		00	01x xxxx xxxx	(D7-Do)	01 xxxx xxxx	(D15-Do)
WDS (Write Disable)	1		00	00x xxxx xxxx	—	00 xxxx xxxx	—
ERASE	1		11	(A10-A0)	—	(A9-A0)	—
ERAL (Erase All Registers)	1		00	10x xxxx xxxx	—	10 xxxx xxxx	—

Notes:

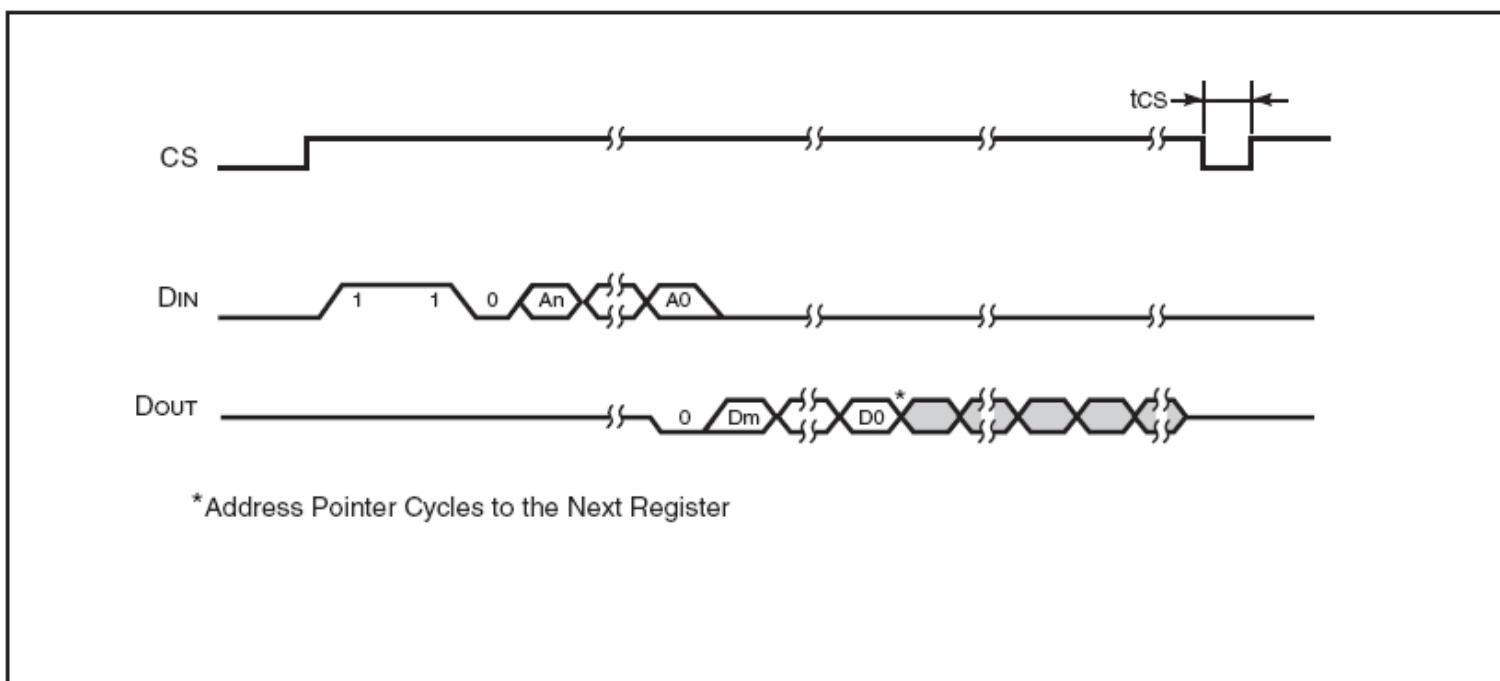
1. x = Don't care bit.
2. If the number of bits clocked-in does not match the number corresponding to a selected command, all extra trailing bits are ignored, and WRITE, WRALL, ERASE, ERAL, WEN, and WDS instructions are rejected, but READ is accepted.

IS93C76A IS93C86A

8K-BIT/16K-BIT SERIAL ELECTRICALLY ERASABLE PROM

MAY 2007

FIGURE 3. READ CYCLE TIMING

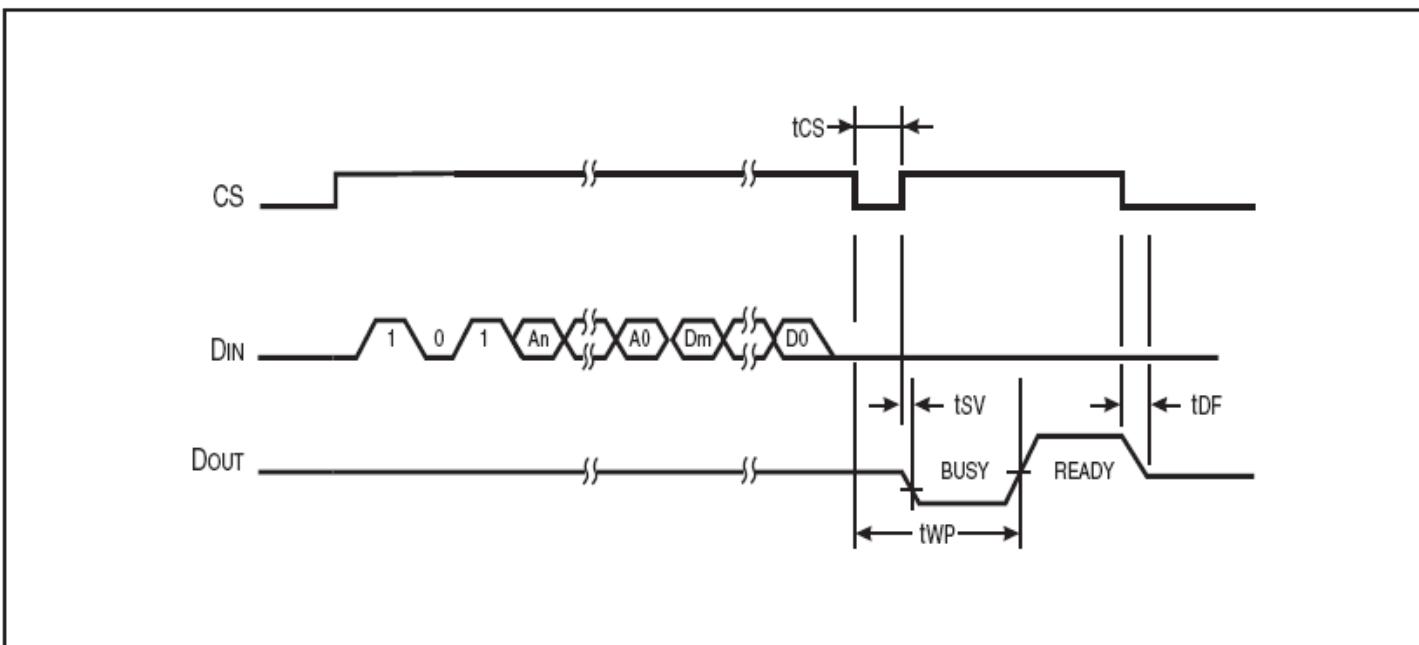


IS93C76A IS93C86A

8K-BIT/16K-BIT SERIAL ELECTRICALLY ERASABLE PROM

MAY 2007

FIGURE 5. WRITE (WRITE) CYCLE TIMING



Notes:

1. After the completion of the instruction (D_{out} is in READY status) then it may perform another instruction. If device is in BUSY status (D_{out} indicates BUSY status) then attempting to perform another instruction could cause device malfunction.
2. To determine address bits A_n - A_0 and data bits D_m - D_0 , see Instruction Set for the specific device.



Features

- Single 4.5V - 5.5V Supply
- Serial Interface Architecture
- Page Program Operation
 - Single Cycle Reprogram (Erase and Program)
 - 1024 Pages (264 Bytes/Page) Main Memory
- Two 264-Byte SRAM Data Buffers – Allows Receiving of Data while Reprogramming of Nonvolatile Memory
- Internal Program and Control Timer
- Fast Page Program Time – 7 ms Typical
- 80 µs Typical Page to Buffer Transfer Time
- Low Power Dissipation
 - 15 mA Active Read Current Typical
 - 15 µA CMOS Standby Current Typical
- 10 MHz Max Clock Frequency
- Hardware Data Protection Feature
- Serial Peripheral Interface (SPI) Compatible – Modes 0 and 3
- CMOS and TTL Compatible Inputs and Outputs
- Commercial and Industrial Temperature Ranges

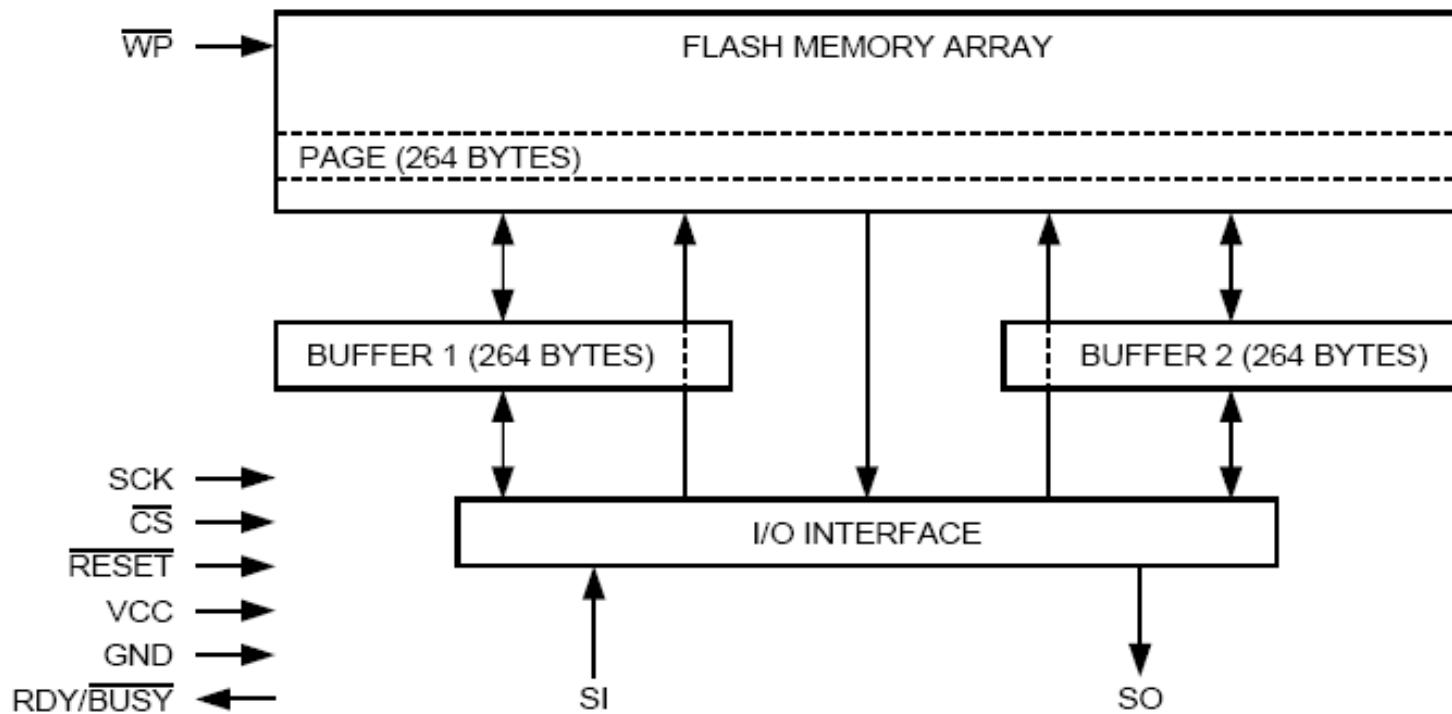
**2-Megabit
5-volt Only
Serial
DataFlash®**

AT45D021

Pin Configurations

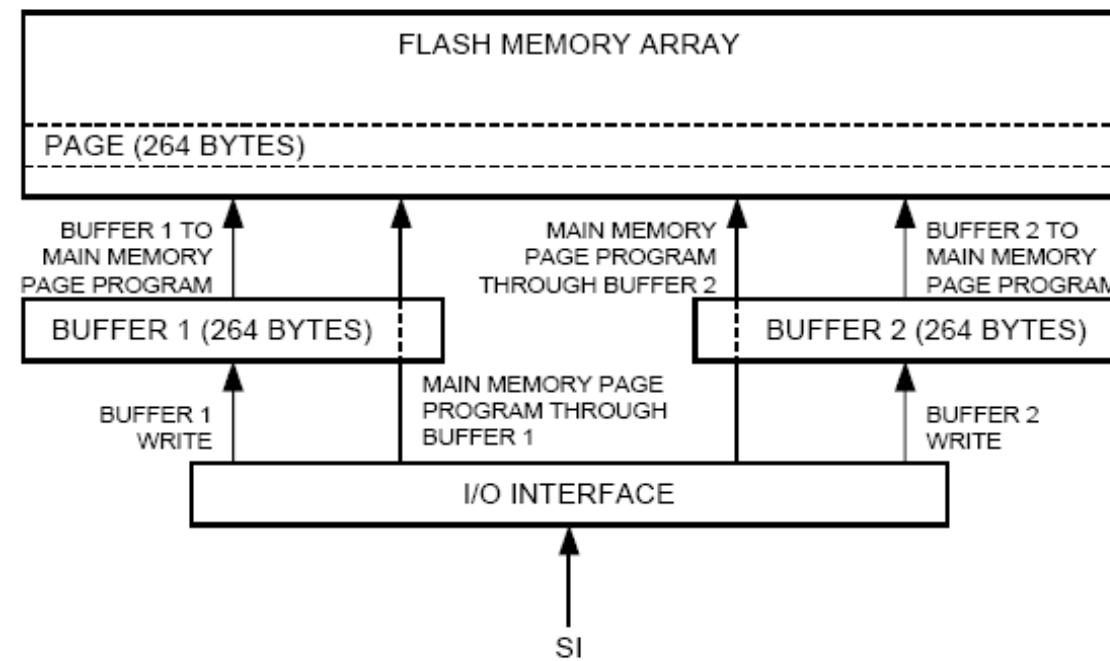
Pin Name	Function
CS	Chip Select
SCK	Serial Clock
SI	Serial Input
SO	Serial Output
WP	Hardware Page Write Protect Pin
RESET	Chip Reset
RDY/BUSY	Ready/Busy

Block Diagram

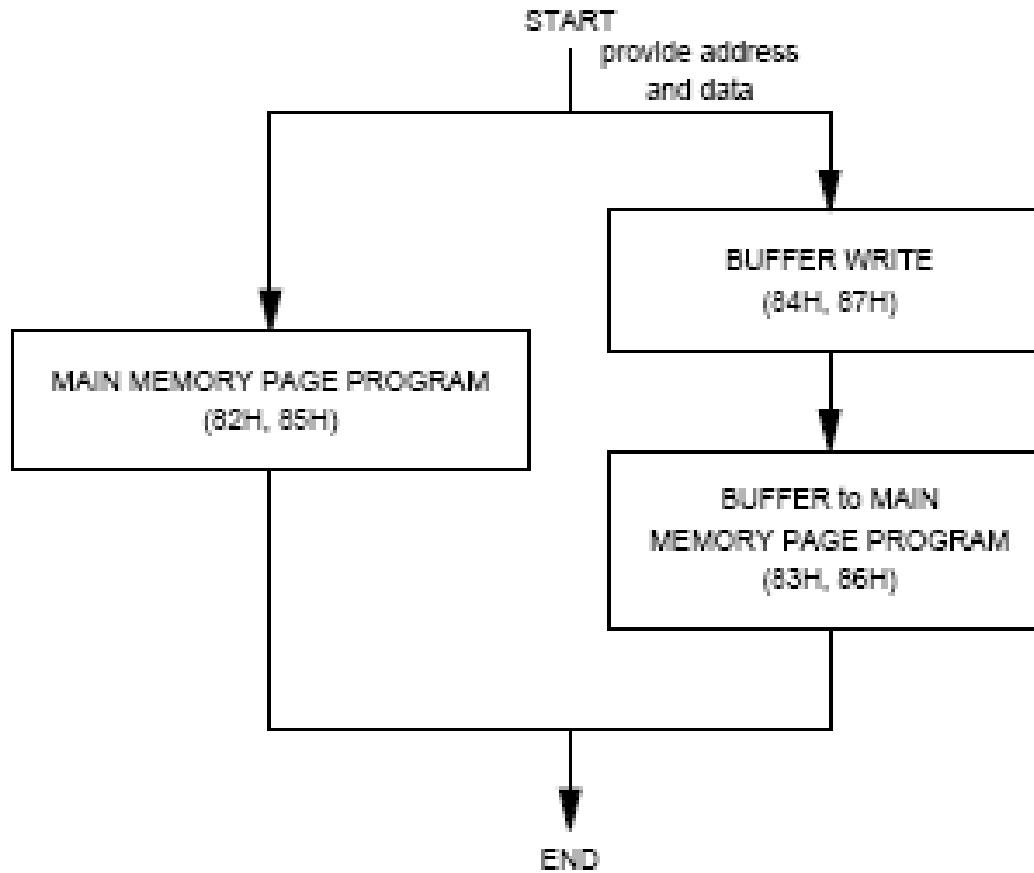


Write Operations

The following block diagram and waveforms illustrate the various write sequences available.



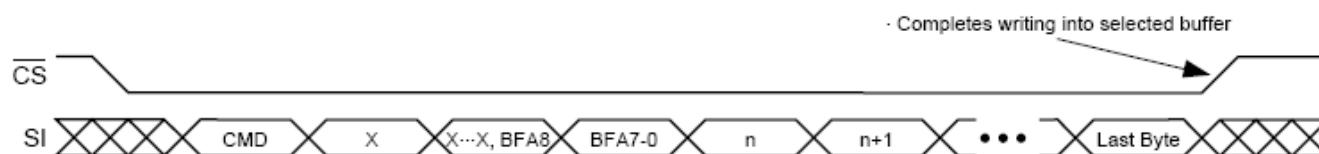
Algorithm for Programming or Reprogramming of the Entire Array Sequentially



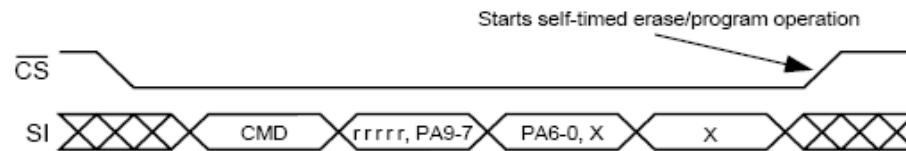
Main Memory Page Program through Buffers



Buffer Write



Buffer to Main Memory Page Program (Data from Buffer Programmed into Flash Page)

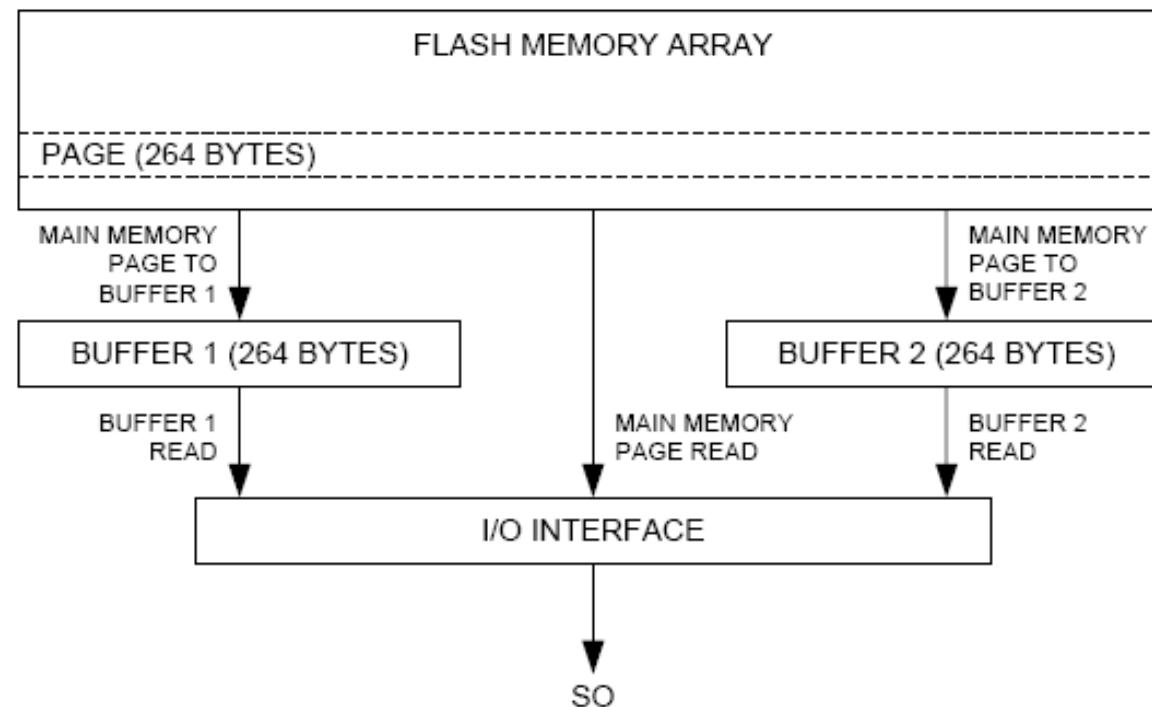


Each transition represents
8 bits and 8 clock cycles

n = 1st byte read
n+1 = 2nd byte read

Read Operations

The following block diagram and waveforms illustrate the various read sequences available.



Main Memory Page Read



Main Memory Page to Buffer Transfer (Data from Flash Page Read into Buffer)



Buffer Read

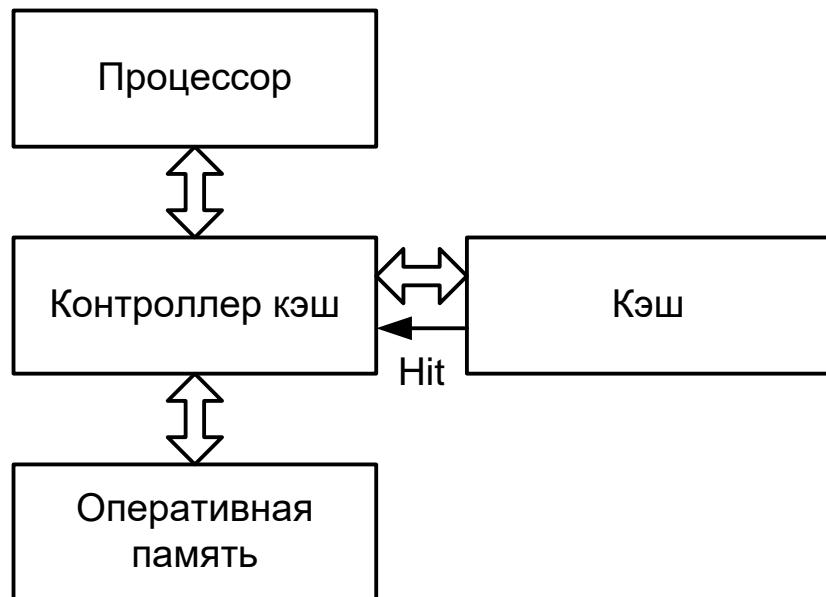


Each transition represents
8 bits and 8 clock cycles

n = 1st byte written
n+1 = 2nd byte written

Принципы построения кэш-памяти

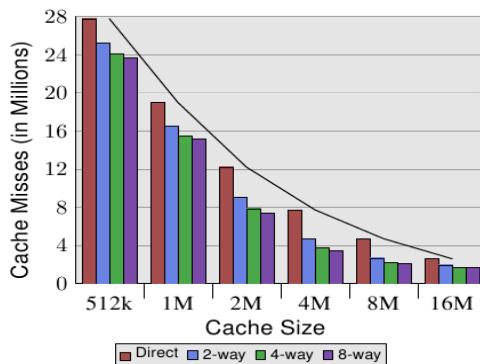
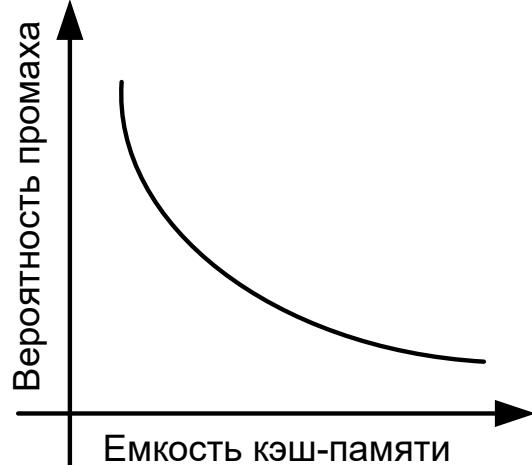
Кэш-память – ассоциативное ЗУ, позволяющее сгладить разрыв в производительности процессора и оперативной памяти. Выборка из кэш-памяти осуществляется по физическому адресу ОП.



Эффективность кэш-памяти зависит от:

- Емкости кэш-памяти.
- Размера строки.
- Способа отображения ОП в кэш.
- Алгоритма замещения информации в кэш.
- Алгоритма согласования ОП и кэш.
- Числа уровней кэш.

Емкость кэш-памяти



Размер линейки

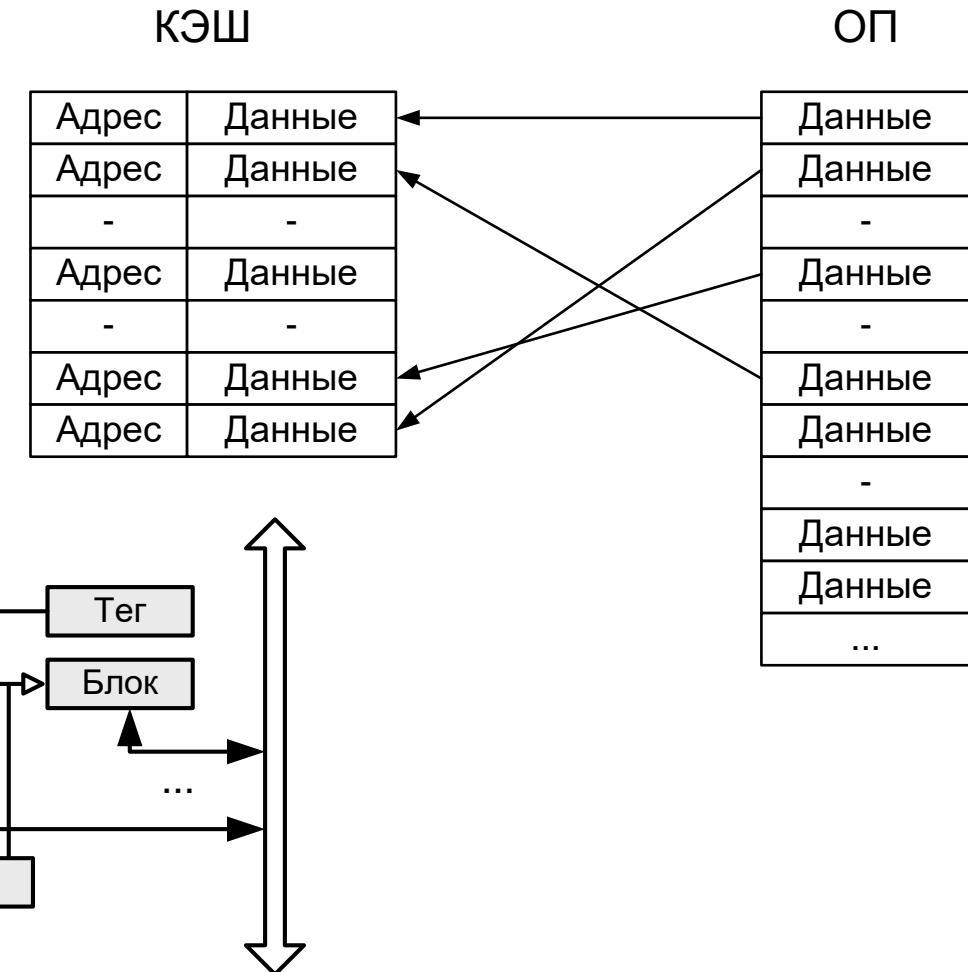
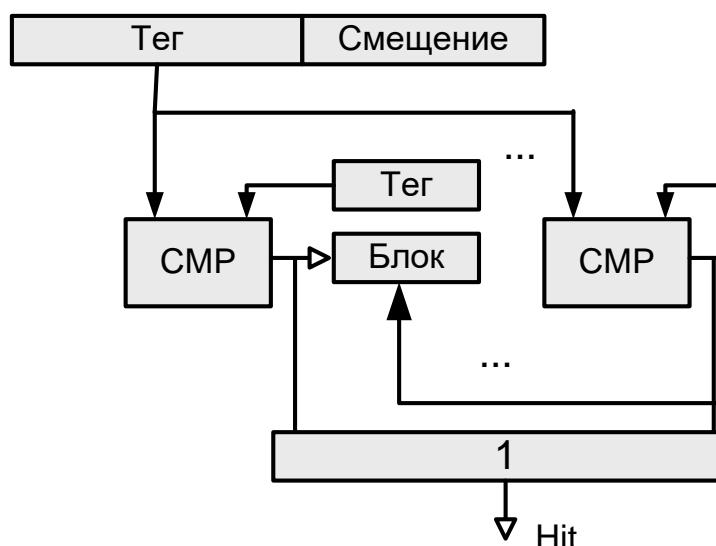


Способы отображения ОП в кэш:

- Произвольная загрузка.
- Прямое размещение.
- Наборно-ассоциативный способ отображения.

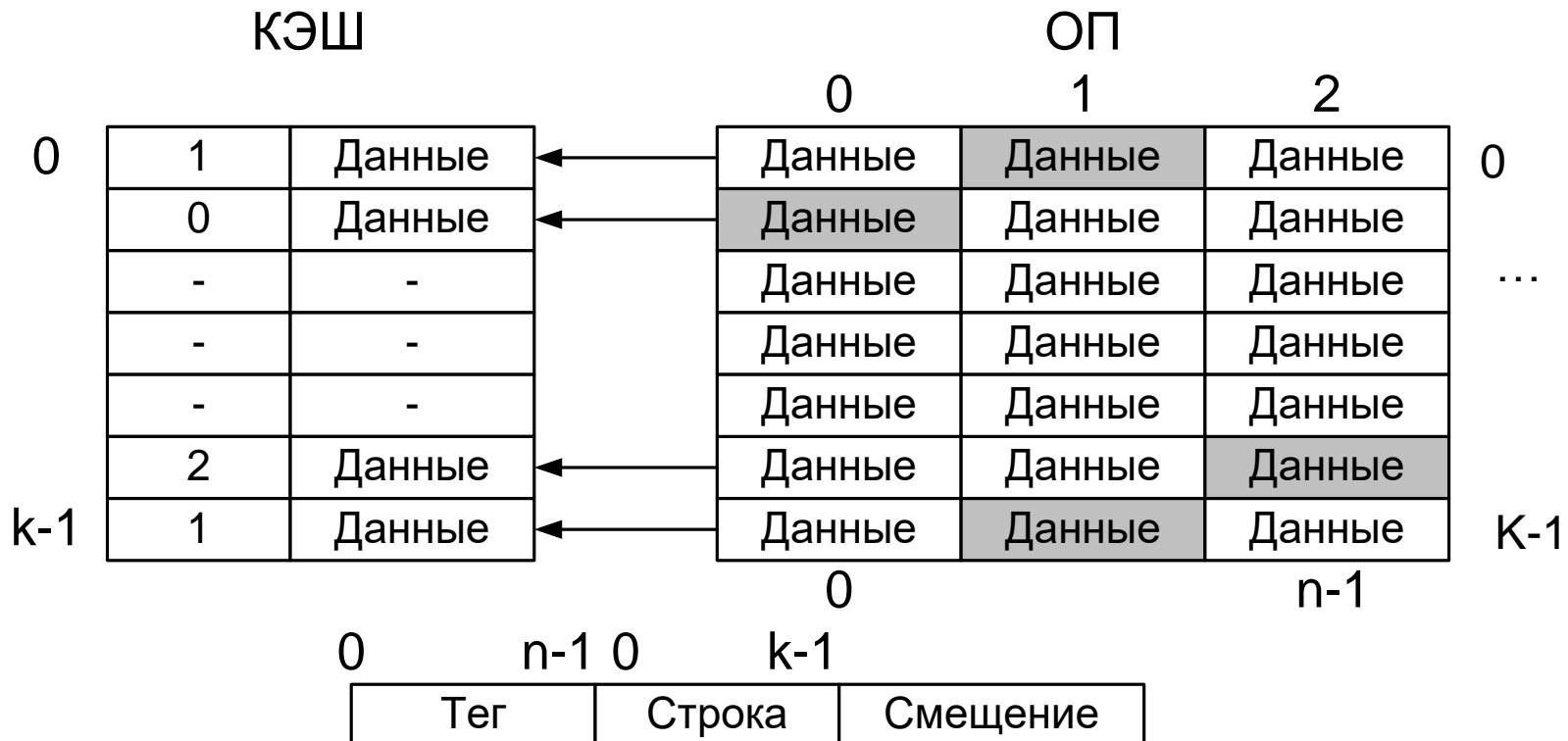
Произвольная загрузка (Fully associated cache memory, FACM).

Адрес строки FACM
определяется из условия
формирования наиболее
представительной выборки

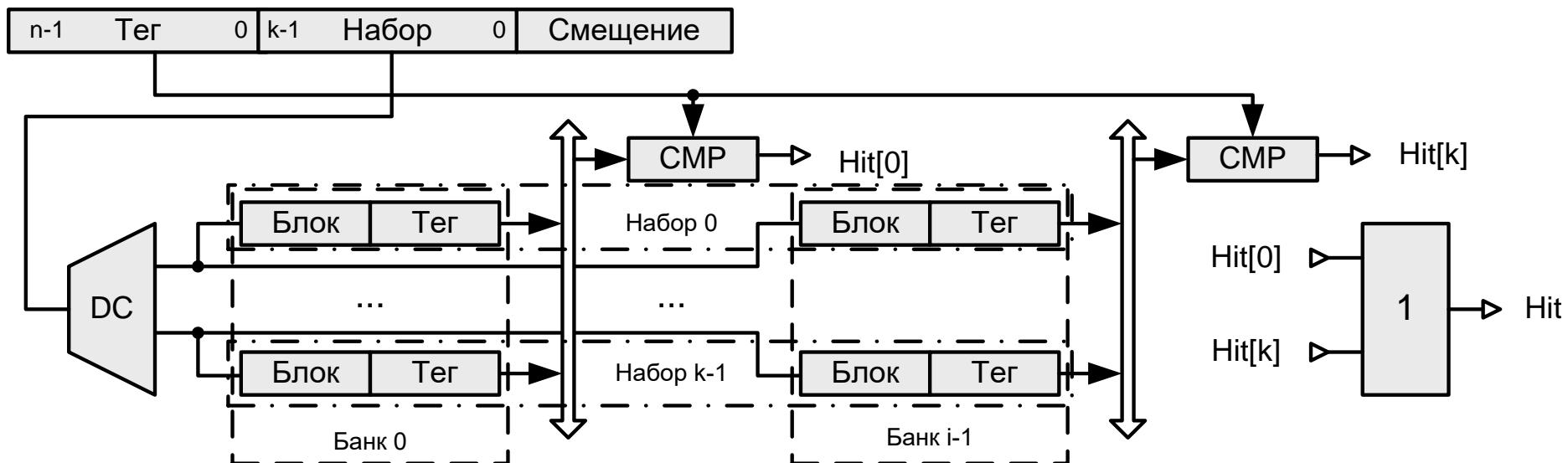
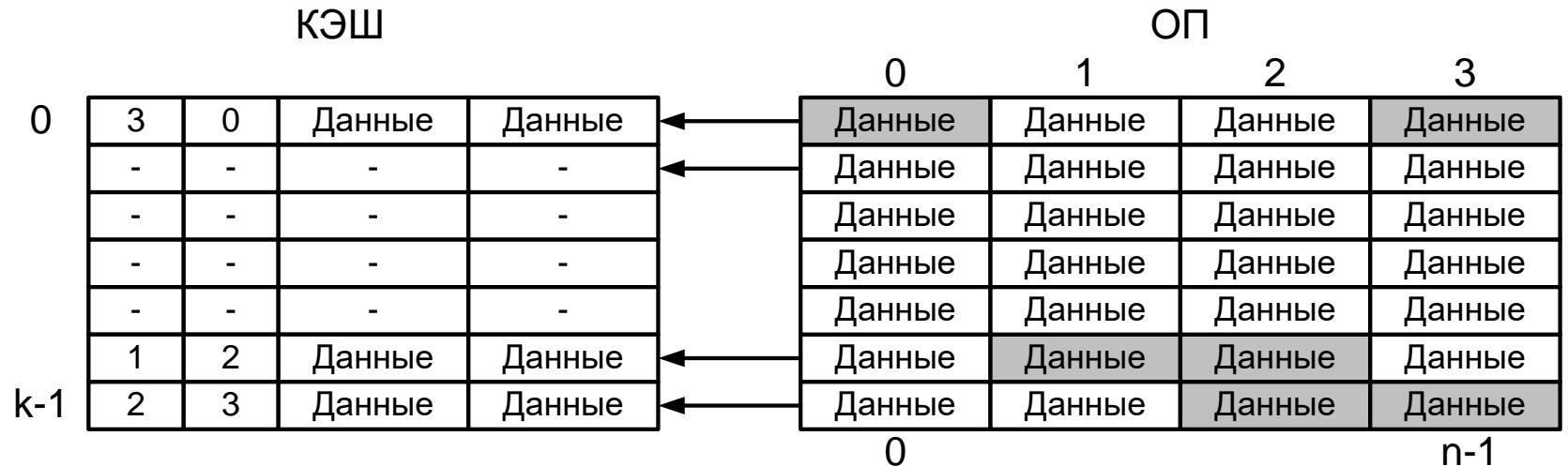


Прямое размещение.

Адрес строки однозначно определяется по тегу ($i = t \bmod k$).



Наборно-ассоциативная кэш-память (Set associated cache memory)



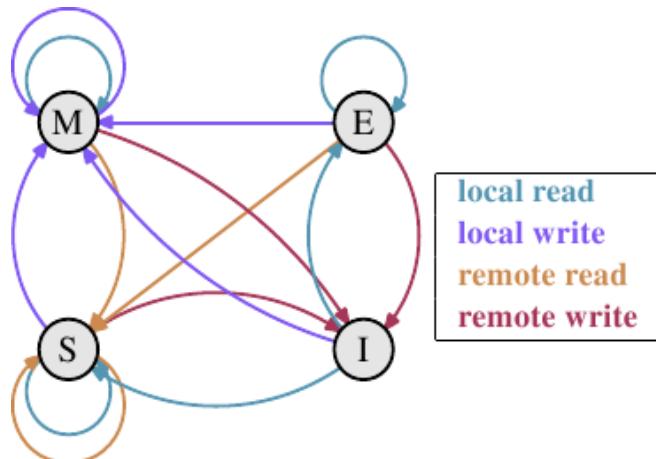
Алгоритмы замещения

- Замещение немодифицированных данных.
- Рандомизированный алгоритм.
- Замещение наименее используемого (Least Recently Used, LRU)

Согласование ОП и кэш

- Метод сквозной записи (Write True).
- Метод сквозной записи с буферизацией (Write Combining).
- Метод обратной записи (Write Back).

Протокол MESI

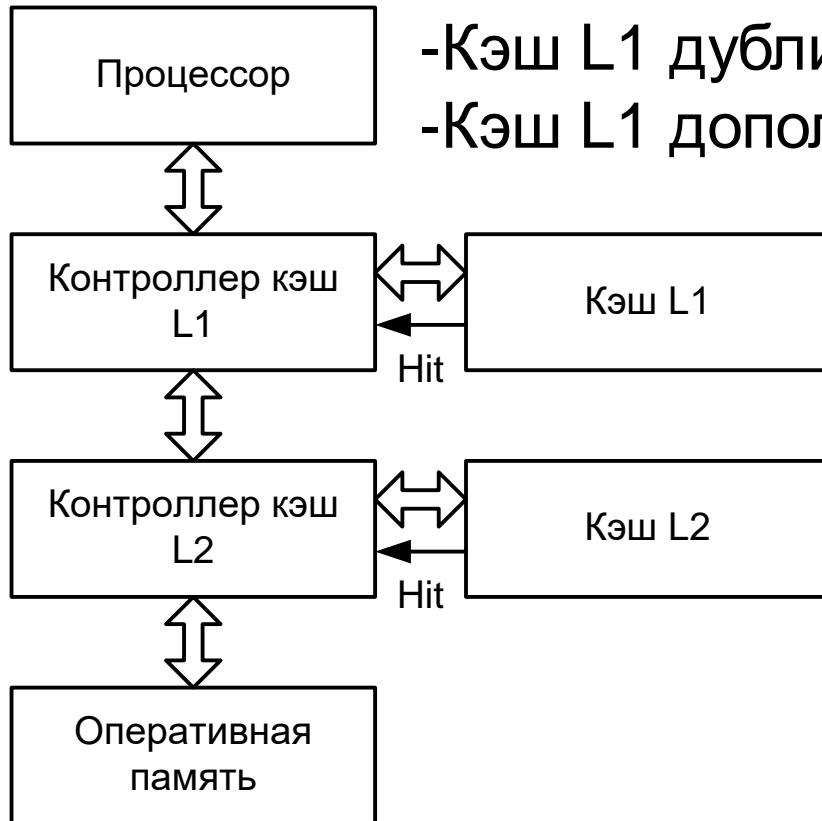


Modified
Exclusive
Shared
Invalid

- Признак несогласованных данных.
- Признак согласованных данных.
- Признак согласованных данных в ВС.
- Признак отсутствия данных.

* - <http://lwn.net/Articles/252125/>

Разделение кэш-памяти

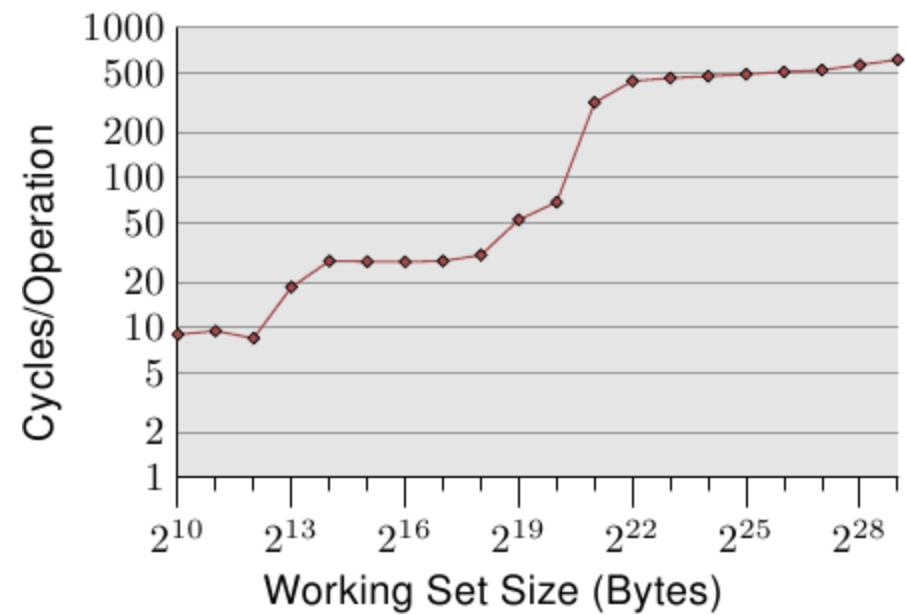


- Кэш L1 дублирует L2 (inclusive).
- Кэш L1 дополняет L2 (exclusive).

Доступ к массивам данным по
случайным адресам

L1D — 2^{13} байт

L2D — 2^{21} байт



Виртуальная память

Механизм виртуализации адресного пространства позволяет:

- Увеличить объем адресуемой памяти.
- Использовать физическую память различного объема.
- Возложить на аппаратную составляющую механизмы доступа к ВЗУ
- Сгладить разрыв в производительности ОП и ВЗУ.
- Ускоряет доступ к данным по последовательным адресам.
- Способствует реализации защиты памяти.

Виртуальные системы строятся по трем принципам:

- Системы с блоками различного размера (сегментная организация).
- Системы с блоками одинакового размера (страничная организация).
- Смешанные системы (сегментно-страничная организация).

Страницная организация

Программа отображается в память равными блоками – страницами. Преобразование логического адреса в физический осуществляется с помощью таблицы страниц.

Преобразование логического адреса в физический реализуется в устройстве управления памятью (Memory Manage Unit), который определяет, находится ли страница в физической памяти (попадение).

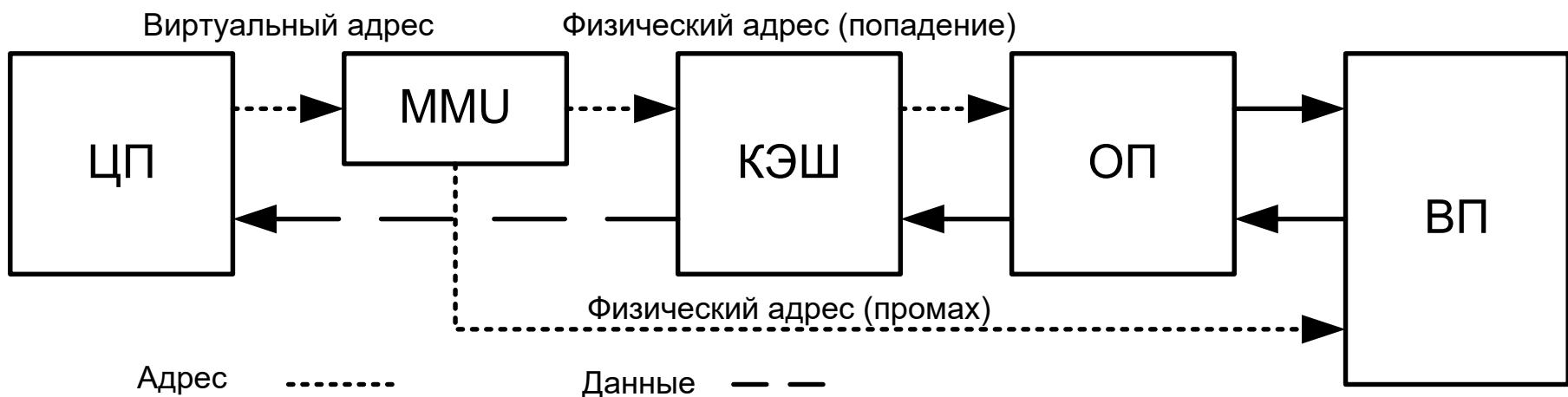
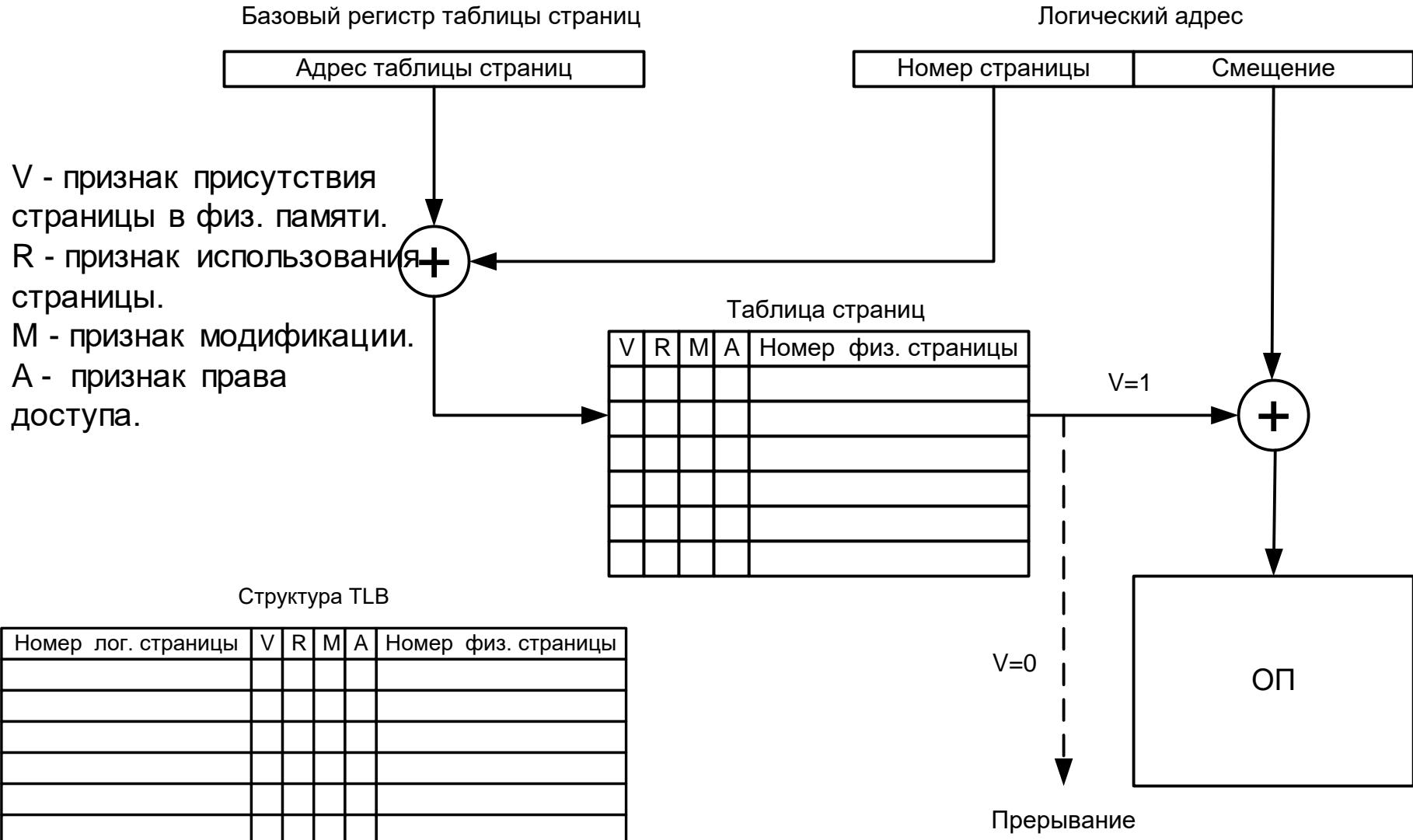
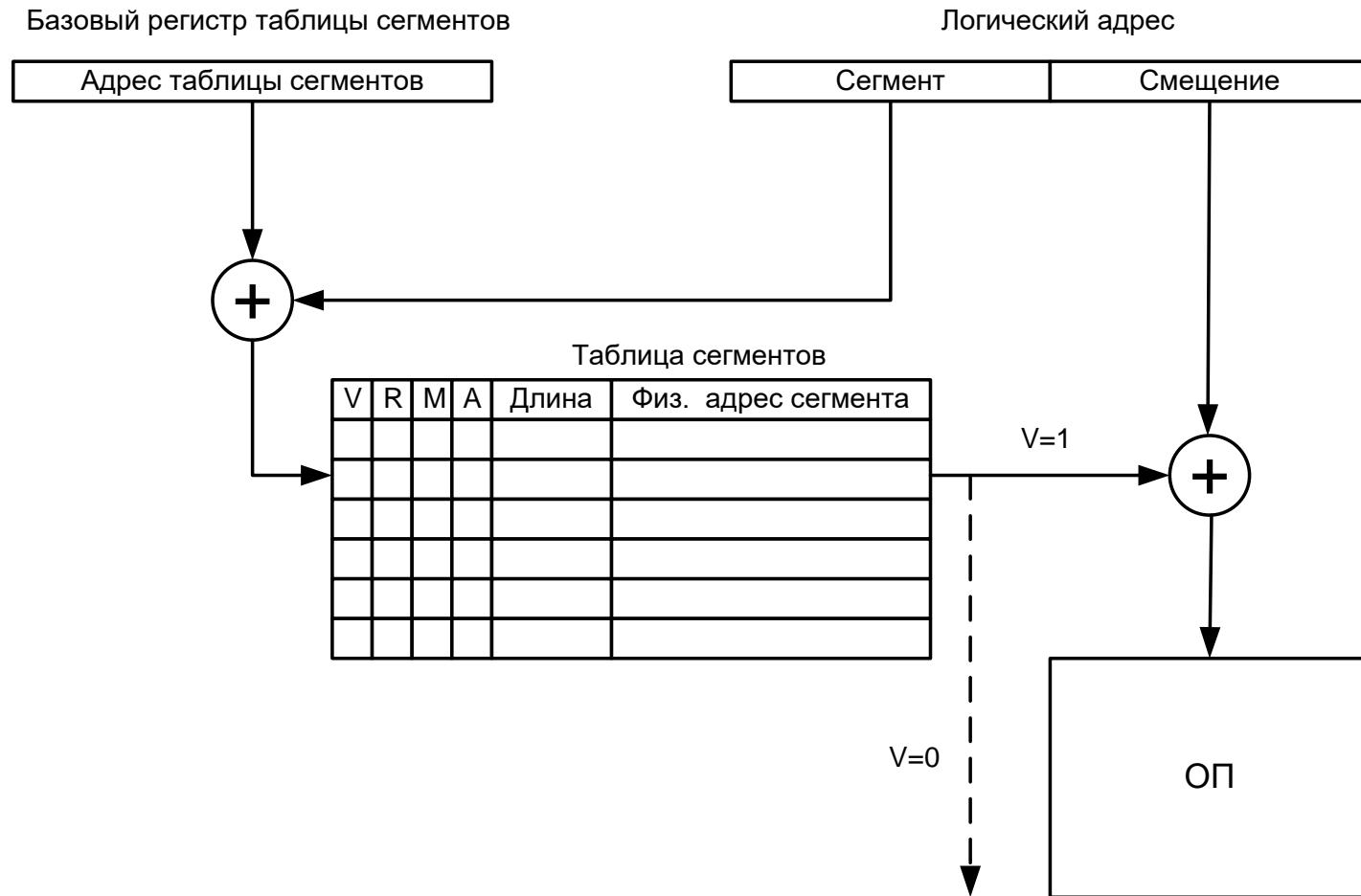


Схема страничного преобразования



Сегментная организация

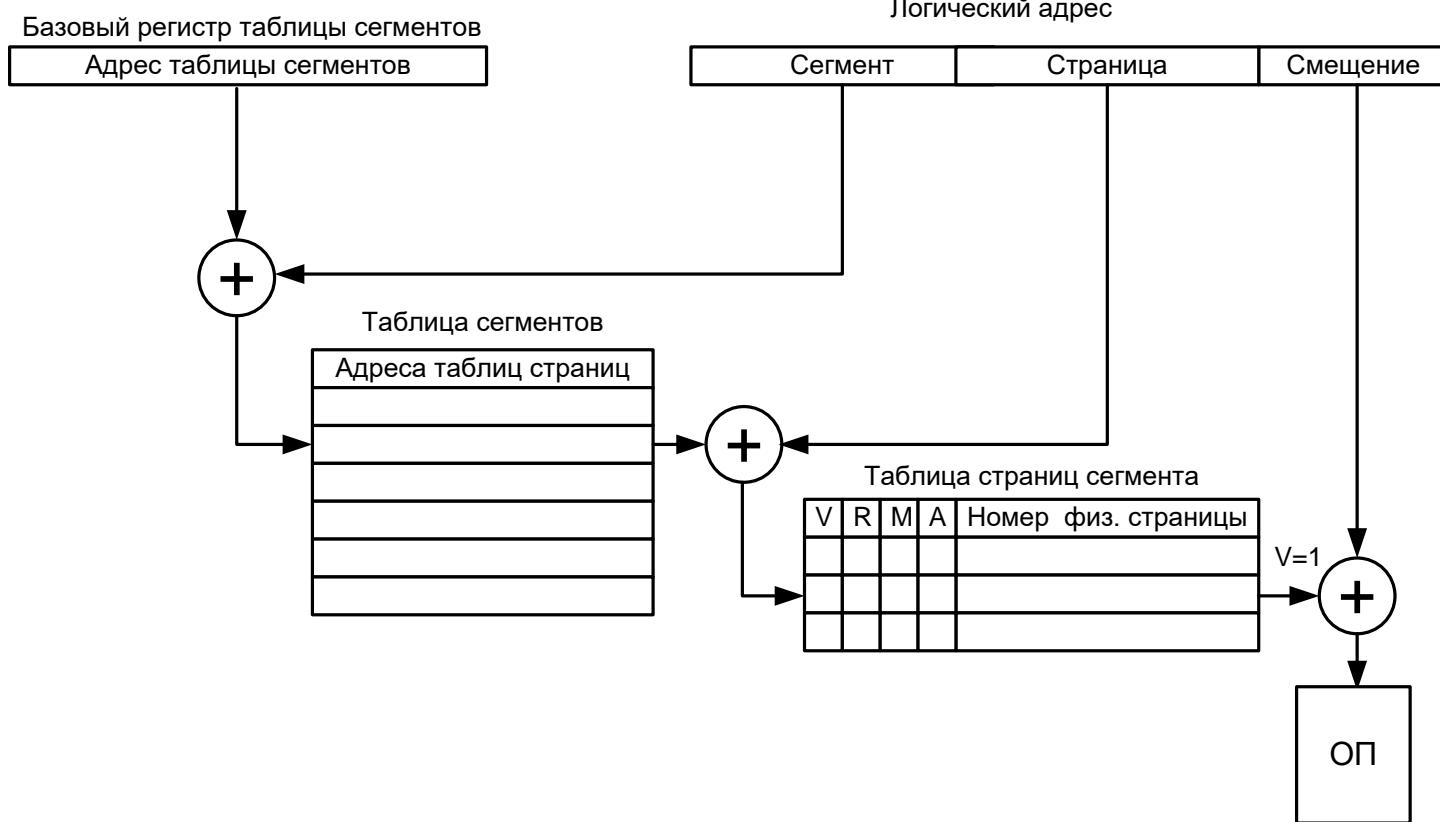
Программа отображается в память блоками различного размера – сегментами. Преобразование логического адреса в физический осуществляется с помощью таблицы сегментов.



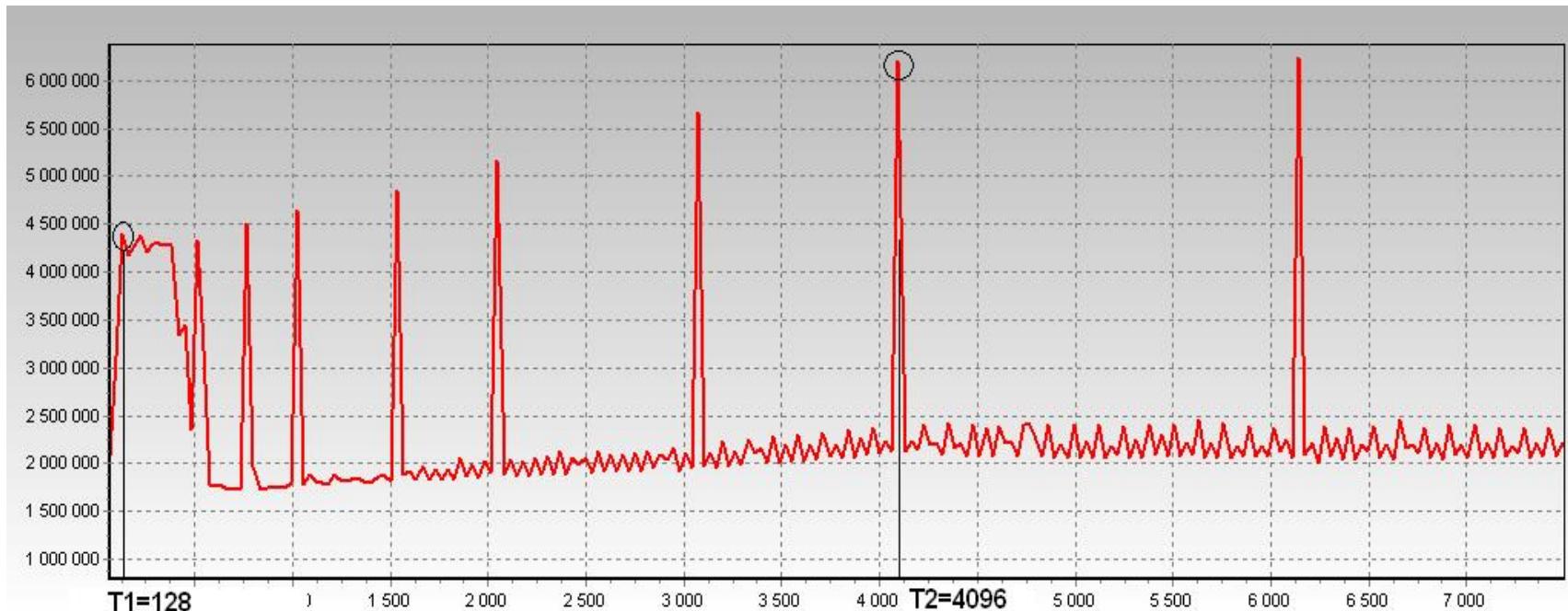
Сегментно-страничная организация памяти

Программа отображается в память блоками различного размера – сегментами, каждый из которых целое число страниц.

Преобразование логического адреса в физический осуществляется с помощью таблицы сегментов и таблицы страниц сегмента.



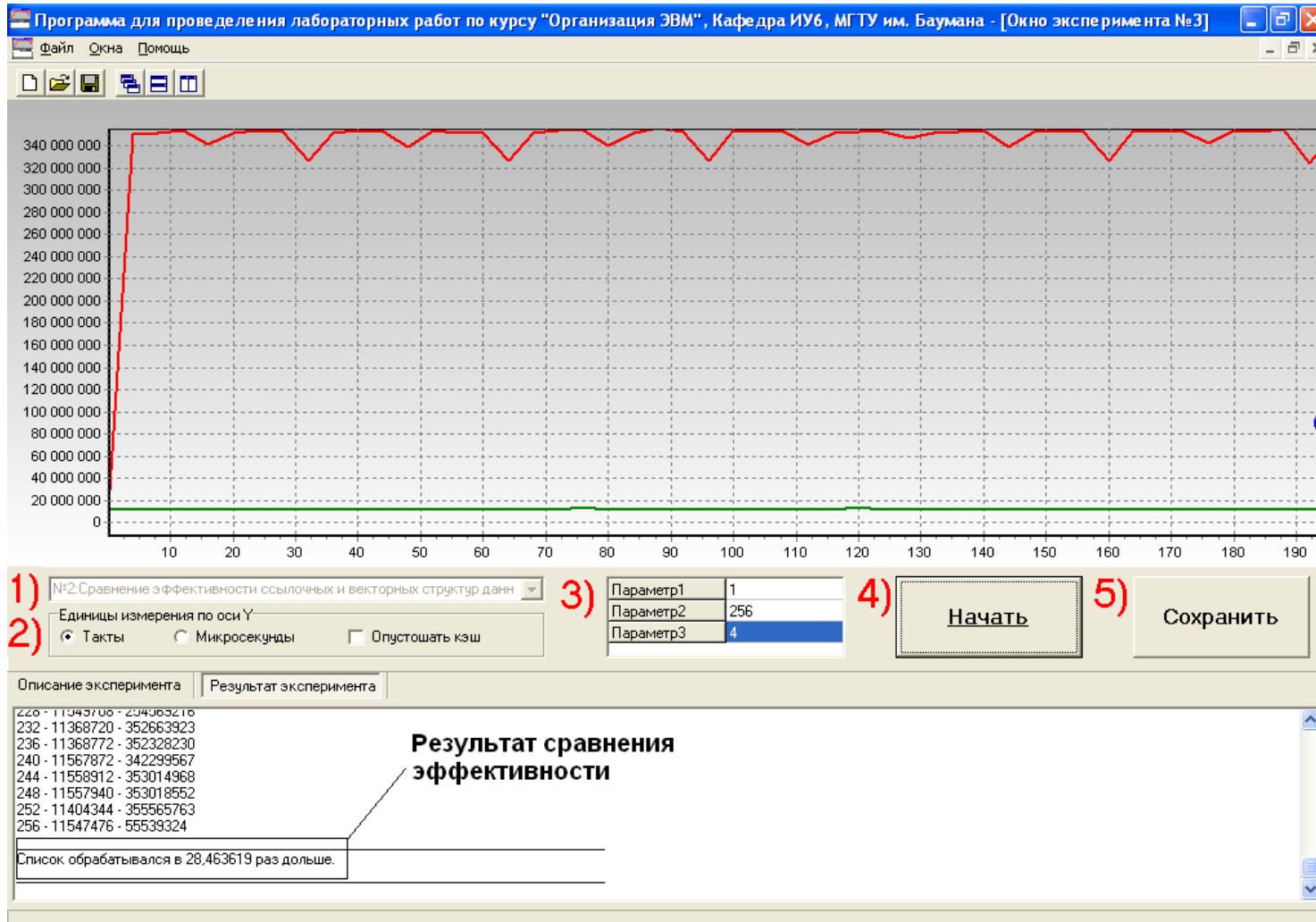
Исследование расслоения динамической памяти.



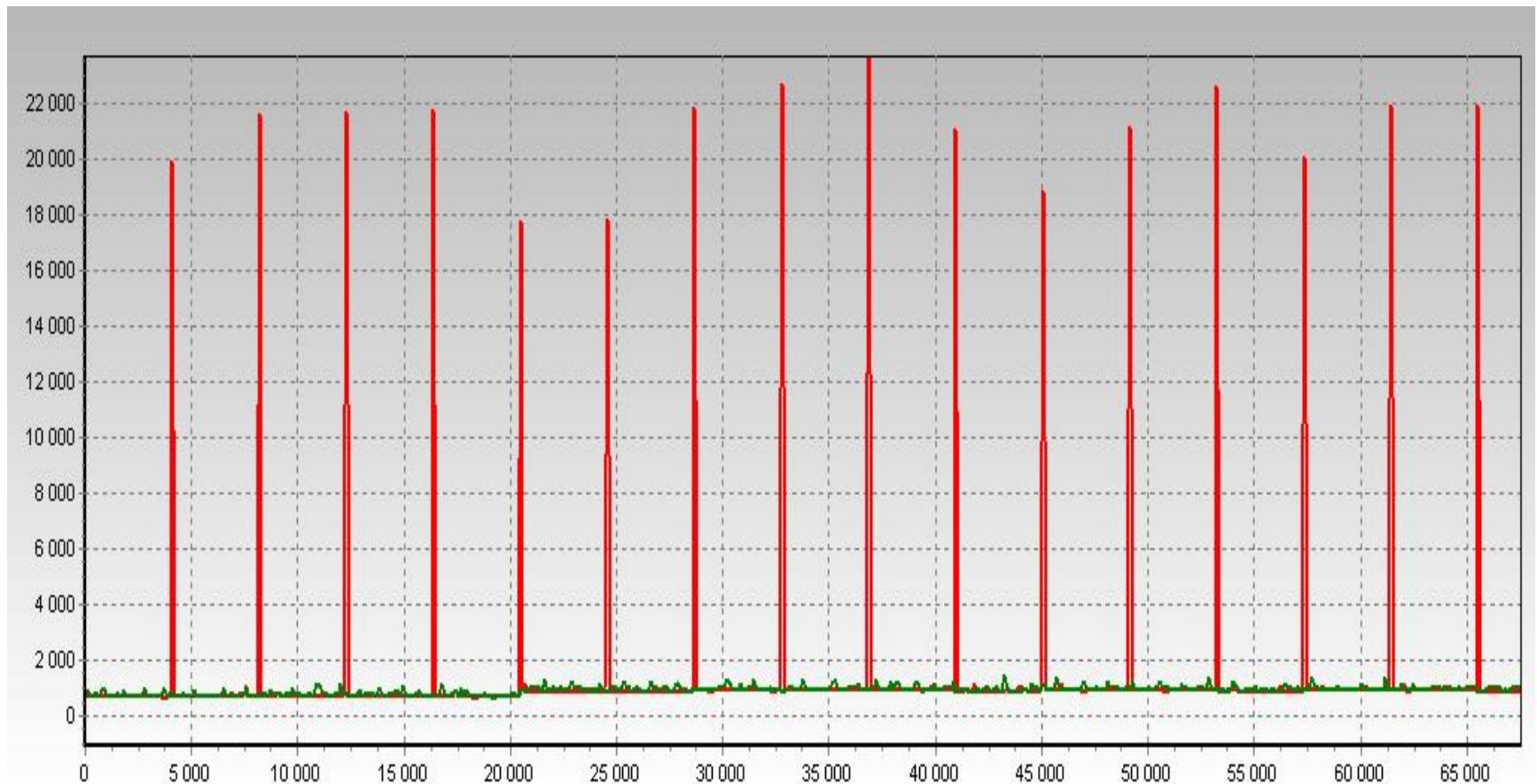
Код профилируемой программы на языке С.

```
// ВЫДЕЛЕНИЕ ПАМЯТИ
p = (int*)malloc64(Param_[3]); // АДРЕС КРАТЕН 64
for (int pg_size = Param_[2]; pg_size <= Param_[1]; pg_size += Param_[2])
{
    Start_Count(); // Начало замера времени
    volatile int x = 0;
    for (int b = 0; b < pg_size; b += Param_[2])
        for (int a = b; a < Param_[3]; a += pg_size)
            x += *(int *) (int(p) + a);
    Finish_Count(); // Конец замера времени
}
```

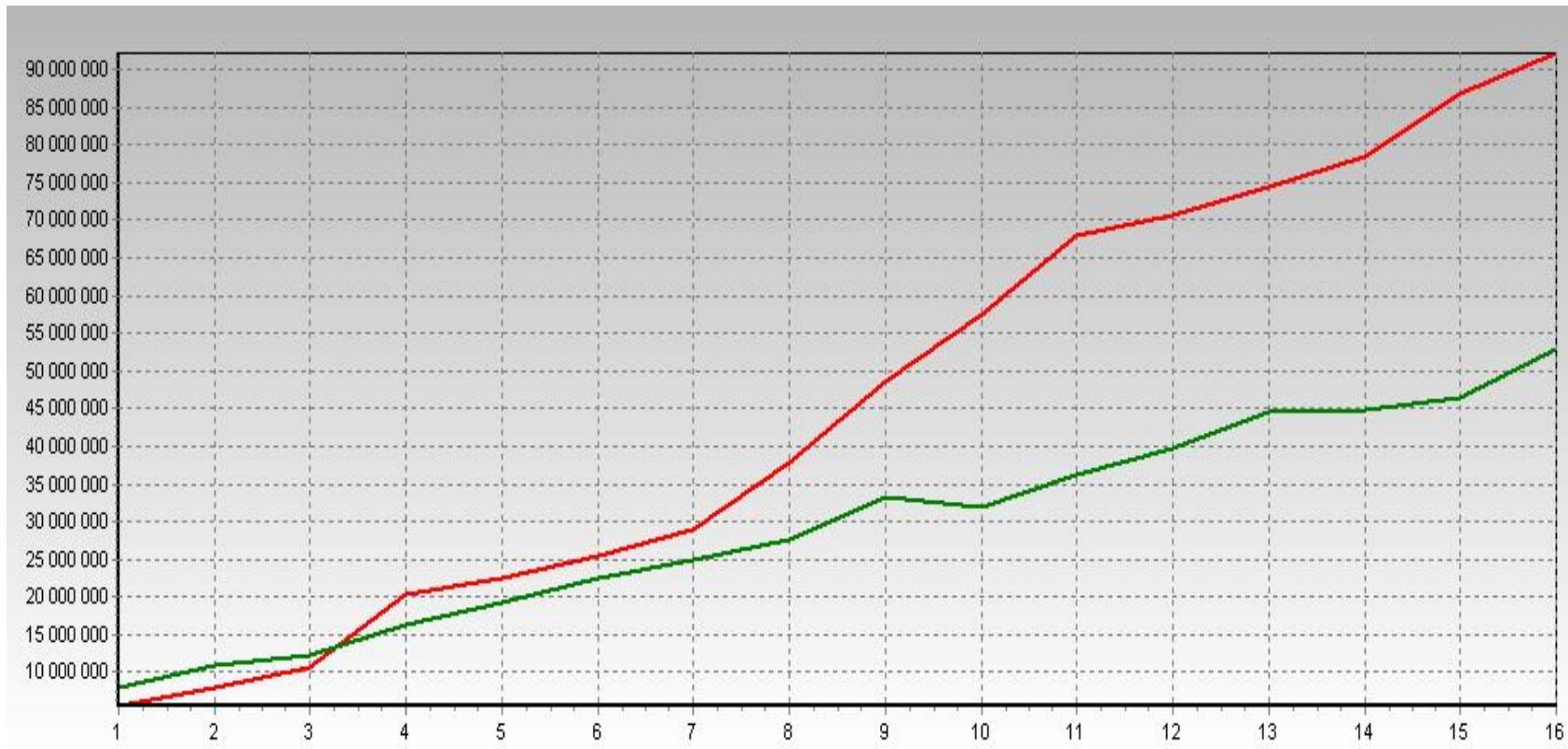
Сравнение эффективности ссылочных и векторных структур



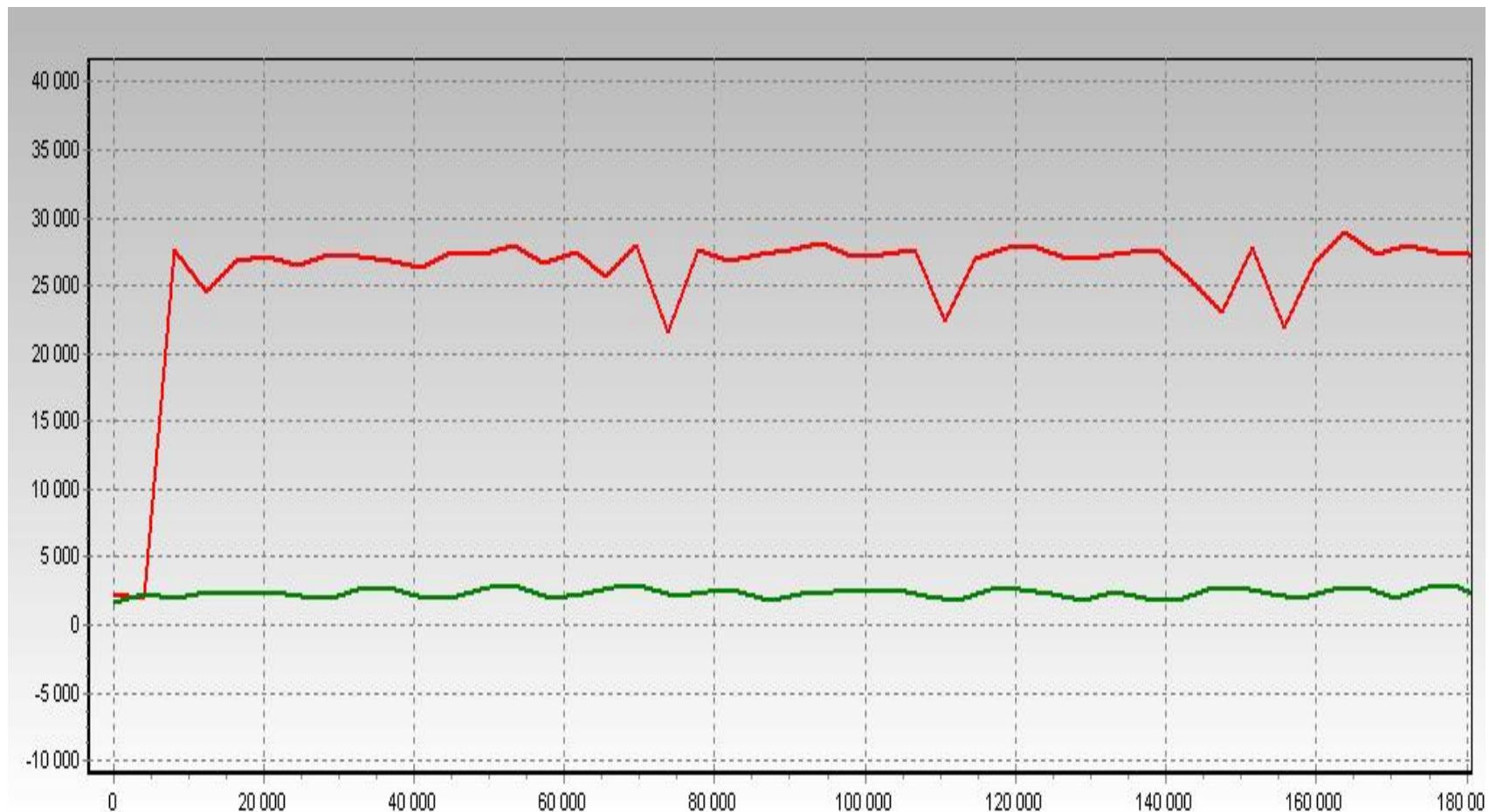
Исследование эффективности предвыборки в TLB



Использование оптимизирующих структур данных



Конфликты в кэш-памяти



Организация ЭВМ и систем

(7 семестр)

Цель дисциплины:

- получить знания и навыки, необходимые для проектирования и эффективного использования современных аппаратных вычислительных средств.

Задачами дисциплины является изучение:

- принципов организации ЭВМ;
- методики проектирования ЭВМ и устройств, их составляющих.

ОСНОВНАЯ ЛИТЕРАТУРА

1. Цилькер Б.Я., Орлов С.А. Организация ЭВМ и систем: Учебник для вузов. – СПб.: Питер, 2004. – 668 с.: ил.
2. Угрюмов Е. П. Цифровая схемотехника: Учеб. Пособие для вузов. – 2-е изд., перераб. и доп. – СПб.: БХВ-Петербург, 2004. – 800 с.: ил.
3. Каган Б.М. Электронные вычислительные машины и системы. - М.: Энергоатомиздат, 1991.

План проведения теоретических и практических занятий:

Семестр	Теоретические занятия	Лабораторные работы	Самостоятельная работа	Вид отчетности
8	Принципы построения и архитектура ЭВМ Устройства управления ЭВМ Операционные устройства ЭВМ Процессорные устройства Организация шин Организация ввода-вывода	Разработка радиоэлектронной аппаратуры на основе микроконтроллеров ARM7 TDMI в интегрированной среде Keil uVISION Изучение средств ввода и вывода алфавитно-цифровой информации и индикации с использованием микроконтроллеров ARM7 Изучение принципов работы цифровых осциллографов Синхронизация микроконтроллера и управление таймерами Система прерываний микроконтроллера и управление интерфейсом RS232 Интерфейс CAN Реализация технологии тонкого клиента на платформе RaspberryPi Свободная тема (Система мониторинга сети на RaspberryPi)	Домашнее задание. Проектирование СнК	экзамен

I. Принципы построения и архитектура ЭВМ

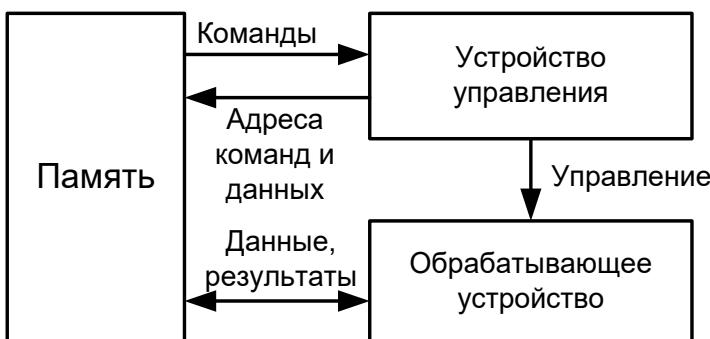
- Общие принципы построения современных ЭВМ.
- Основные тенденции развития ЭВМ.
- Классификация архитектур системы команд (СК).
- RISC, CISC, VLIW архитектура.
- Типы команд.
- Форматы команд.
- Способы адресации.

Общие принципы построения современных ЭВМ

Принципы Фон-Неймана

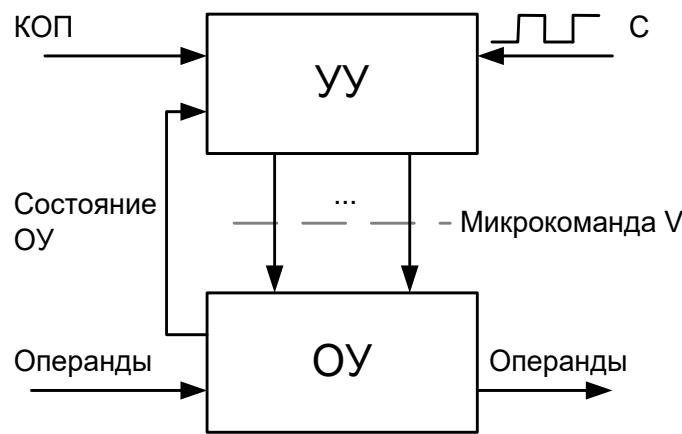
- Двоичное кодирование информации
- Программное управление
- Адресность памяти
- Однородность памяти

ОКОД, SISD



- Гарвардская архитектура
(ОП для хранения команд и ОП для хранения данных)
- Принстонская архитектура
(ОП для хранения команд и данных)

Принципы микропрограммного управления



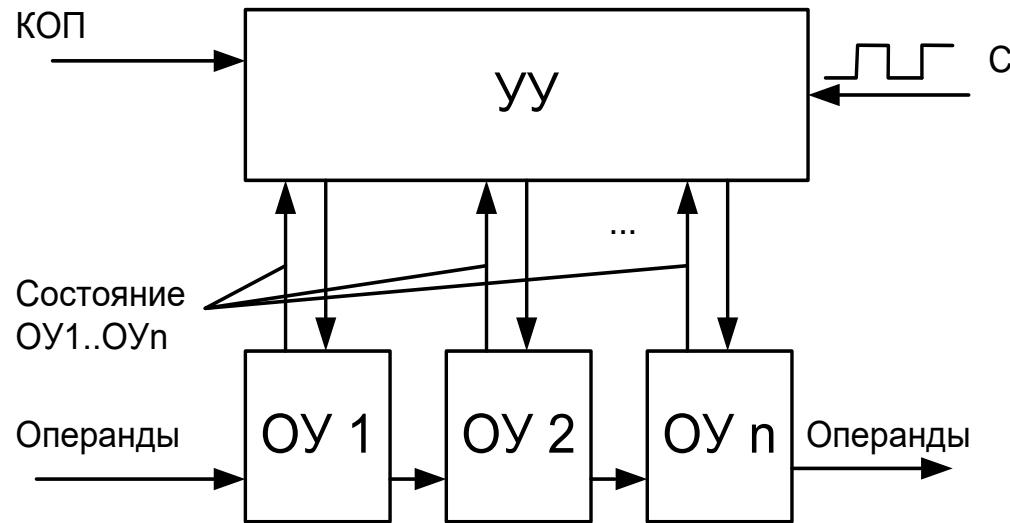
Любое цифровое устройство можно рассматривать, как совокупность операционного и управляемого блока.

Любая команда или последовательность команд реализуется в операционном блоке за несколько тактов

Последовательность сигналов управления должна выдаваться устройством управления в соответствии с поступающей на вход командой и текущим состоянием операционного блока

Состояние линий управления в каждом такте задает микрокоманду. Совокупность микрокоманд, необходимых для реализации команды называется микропрограммой.

Принцип конвейерной обработки



Конвейерная обработка представляет собой процесс, при котором сложные действия разделяются на более короткие стадии. Их параллельное выполнение для последовательности действий позволяет более полно использовать обрабатывающие ресурсы конвейера.

Структура современных ЭВМ с архитектурой Фон-Неймана

- Центральное процессорное устройство (ЦПУ).
 - Арифметико-логическое устройство (АЛУ)
 - Устройство управления (УУ)
 - Регистры общего назначения (РОН)
- Основная память
- Система ввода-вывода
- Внешние устройства
- Внешняя память
- Система передачи информации
- Система синхронизации
- Система прерываний
- Система прямого доступа к памяти
- Система подвода питания/земли и система энергосбережения
- Система повышения отказоустойчивости

Компьютеры с «не Фон-Неймовской» архитектурой

Нейрокомпьютеры — устройство переработки информации на основе принципов работы естественных нейронных систем.

Когнитивный компьютеринг — вычислительная технология, основанная на имитации процесса познания на нейросинаптических структурах

Компьютеры, управляемые потоком данных - выполнение каждой операции производится при готовности всех её операндов, при этом последовательность выполнения команд заранее не задаётся

Квантовые компьютеры - вычислительное устройство, работающее на основе квантовой механики. Квантовый компьютер принципиально отличается от классических компьютеров, работающих на основе классической механики.

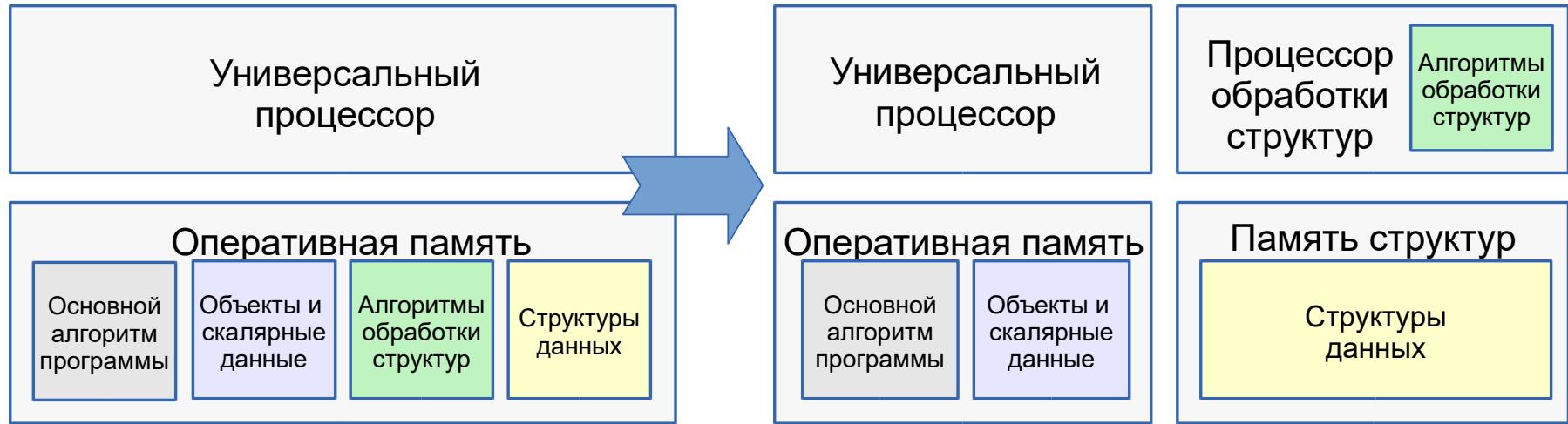
TrueNorth neurosynaptic computer chip



TrueNorth chip (08.2014):

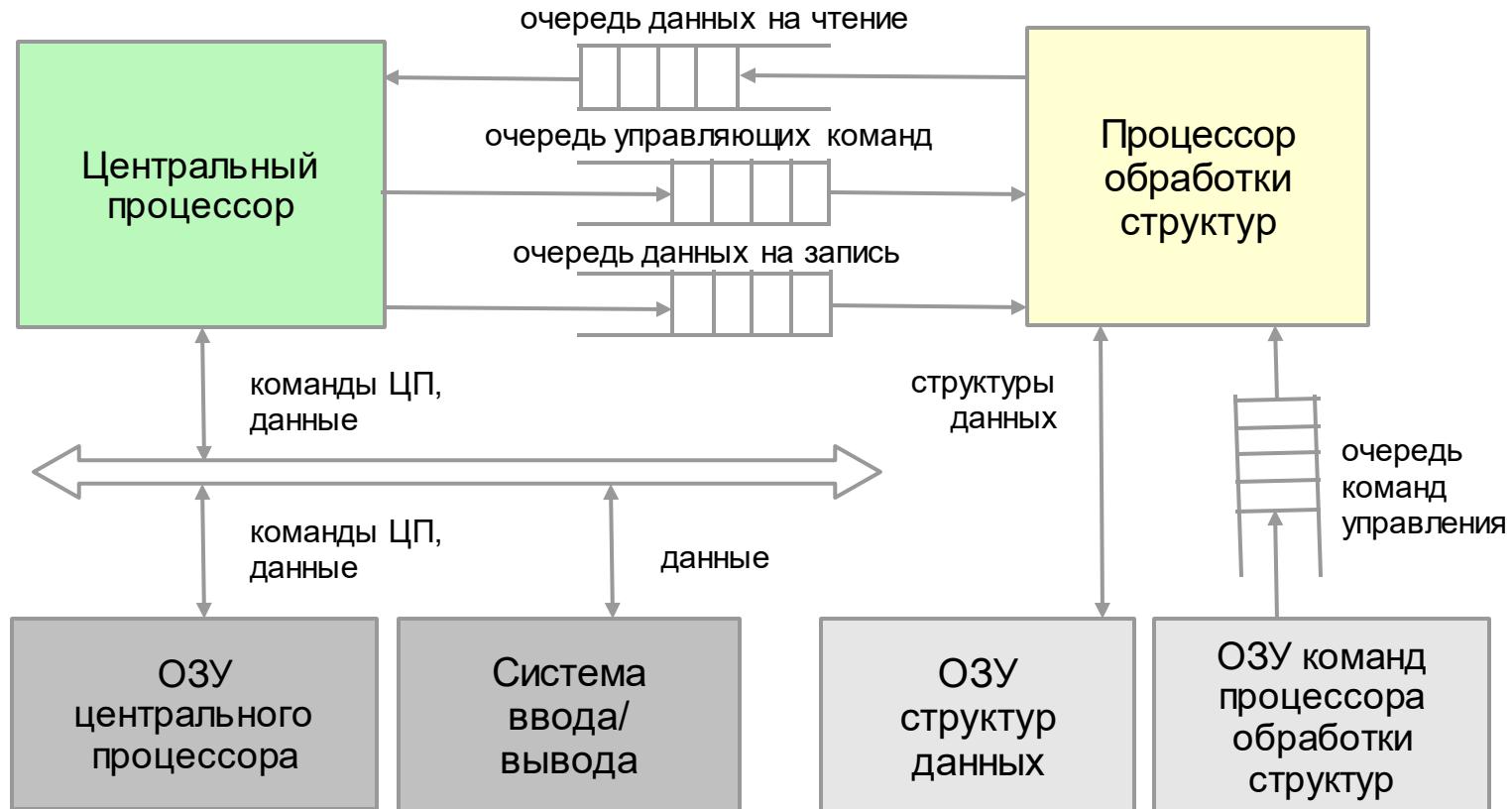
- Не Фон-Неймановская архитектура
- 5.4 миллиарда транзисторов
- 4,096 нейросинаптических ядра
- Миллион нейронов и 256 миллионов синапсов (связей между нейронами)
- Произведен по технологии 28nm
- Потребляет 70mW

Система с аппаратной поддержкой операций дискретной математики (Discrete mathematics Instructions Set Computer)

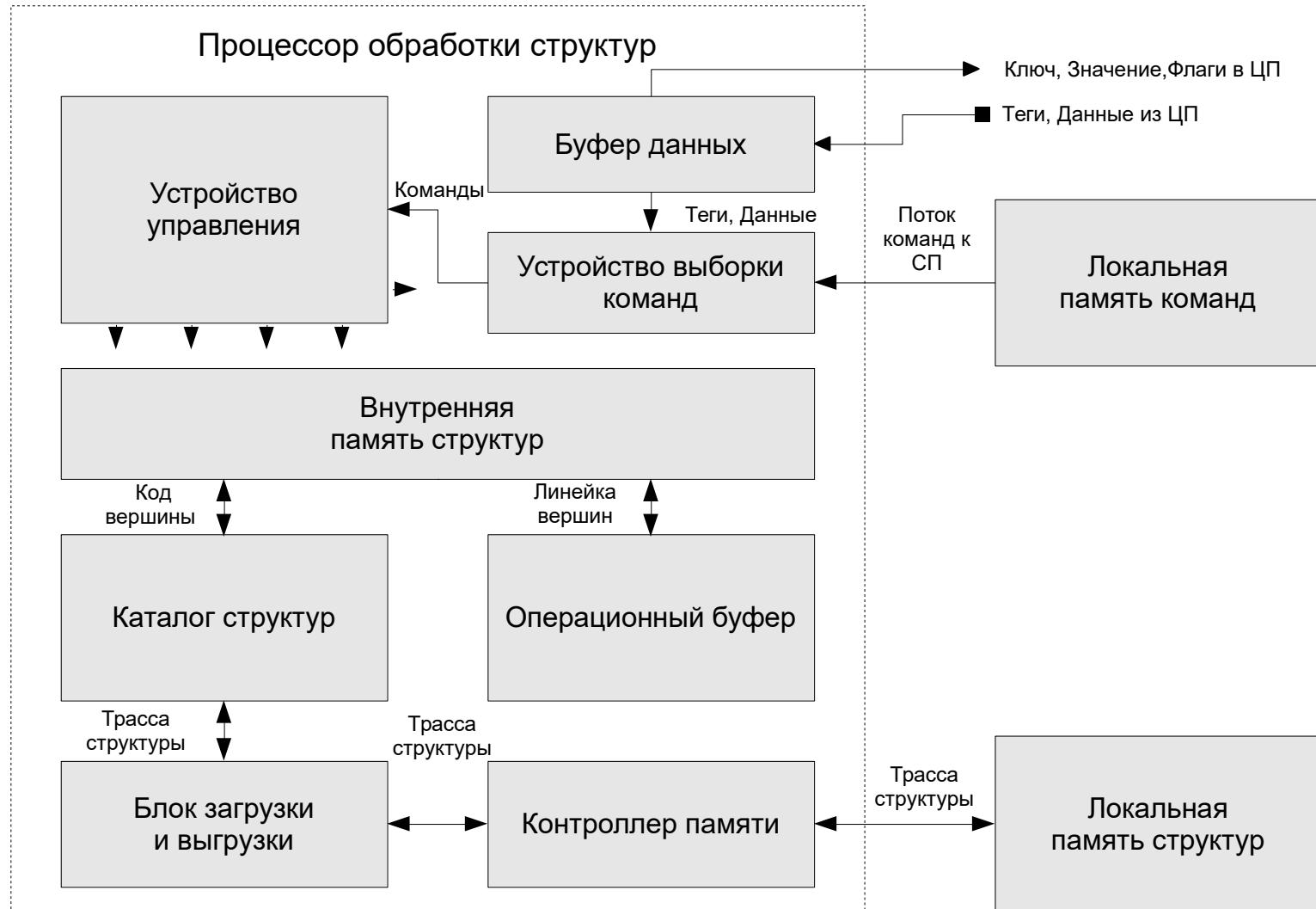


- ускорения задач оптимизации на графах в научных и инженерных расчетах;
- планирование траектории движения роботизированных систем;
- ускорение сетевой маршрутизации и планирование выделения ресурсов пр QoS;
- аппаратная поддержка реляционных СУБД;
- ускорение поиска информации в системах хранения данных;
- ускорение работы операционных систем;
- ускорители САПР и аналитических систем;
- повышение скорости разработки программного обеспечения.

Вариант реализации DISC системы с распределенной памятью



Микроархитектура процессора Leonhard



Набор команд Leonhard

- **Search** key in the specified data structure.
- **Insert** key and value into the data structure.
- **Delete** key and removes it from the data structure.
- **Smaller and Greater Neighbors** help to find neighbor keys and return its value.
- **Maximum and Minimum** instructions find the first or last key in the data structure.
- **Cardinality** instruction helps to understand keys count in the data structure.
- **AND, OR, NOT** instructions perform union, intersection, and complement operations on two data structures.
- **Slices LS, GR, LSEQ, GREQ** perform extraction the subset of one data structure into another.
- **Search next and previous** exactly find next (or previous key) in the data structure from the stored key.
- **Delete all structure** clears all resources used by the given structure.
- **Squeeze** instruction compresses the memory blocks used by the data structure.
- **Jump** instruction branches the SPU code in order to give the CPU control.

Способы хранения графов в SPU Leonhard

1) Список смежных вершин

$G.KEY$	$G.VALUE$
$u,0$	count
$u,1$	v_1, c
...	...
u, count	v_{count}, c

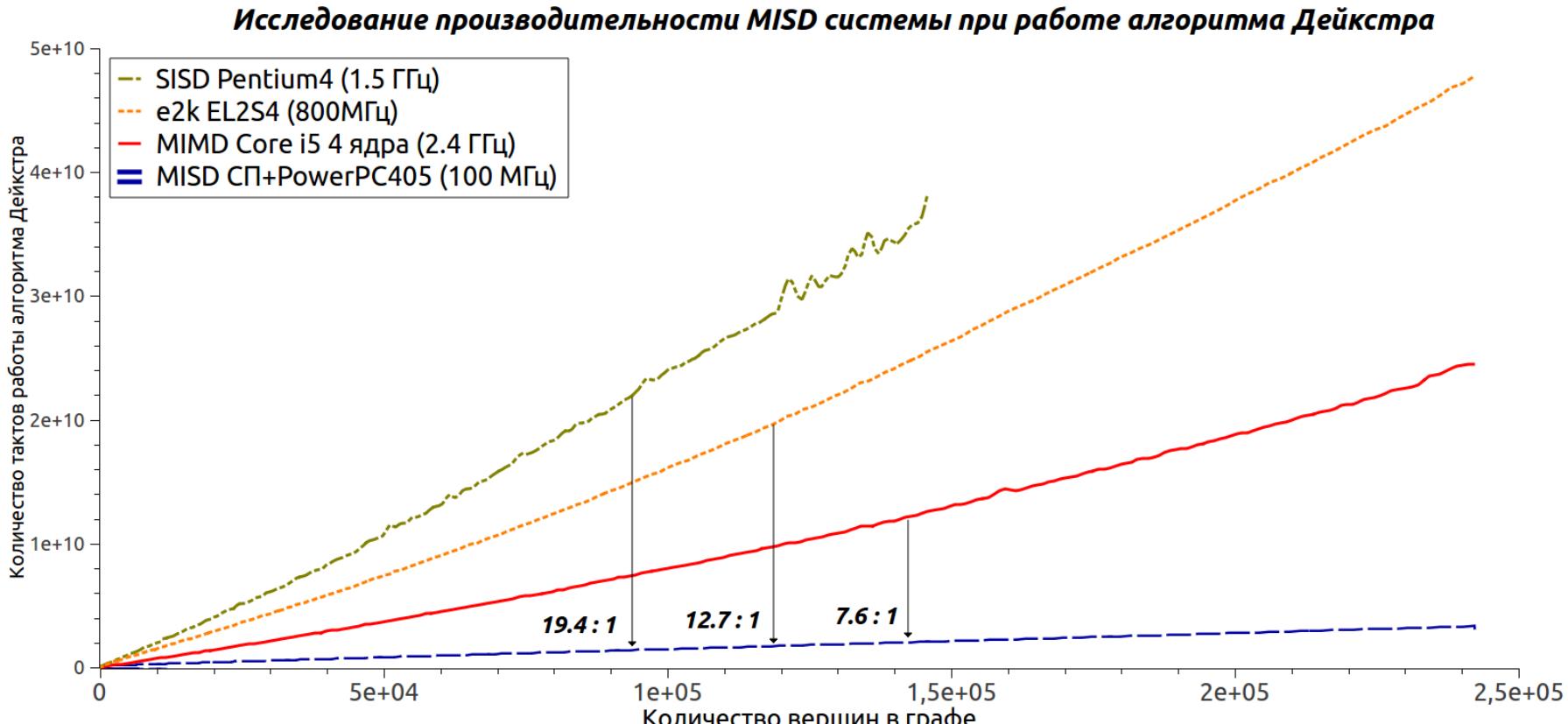
2) Список инцидентных ребер

$G.KEY$	$G.VALUE$
$u,0$	0
u,v	c

3) Упорядоченный список
инцидентных ребер

$G.KEY$	$G.VALUE$
c, u, v	

Пример повышения эффективности в гетерогенной системе

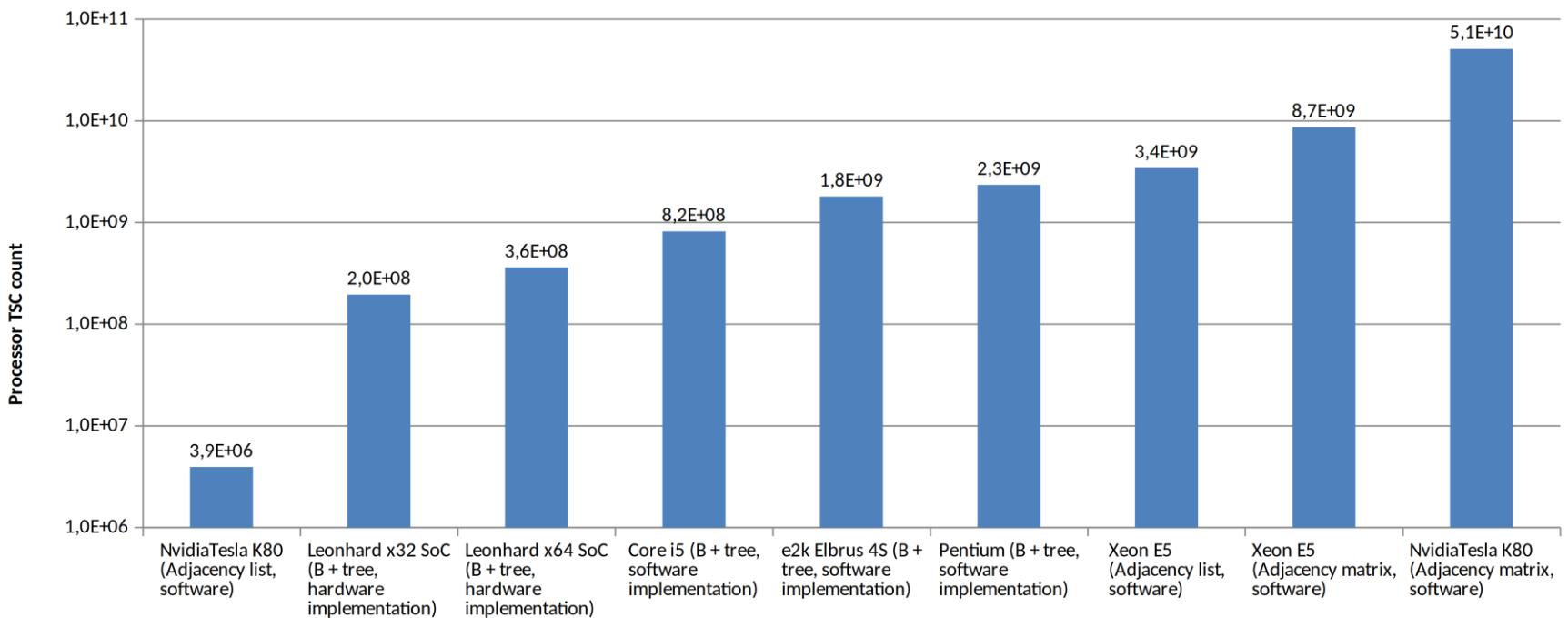


Количество транзисторов в MISD системе - 1 млн (800:1).
Рассеиваемая мощность MISD системы - 1 Вт (35:1).

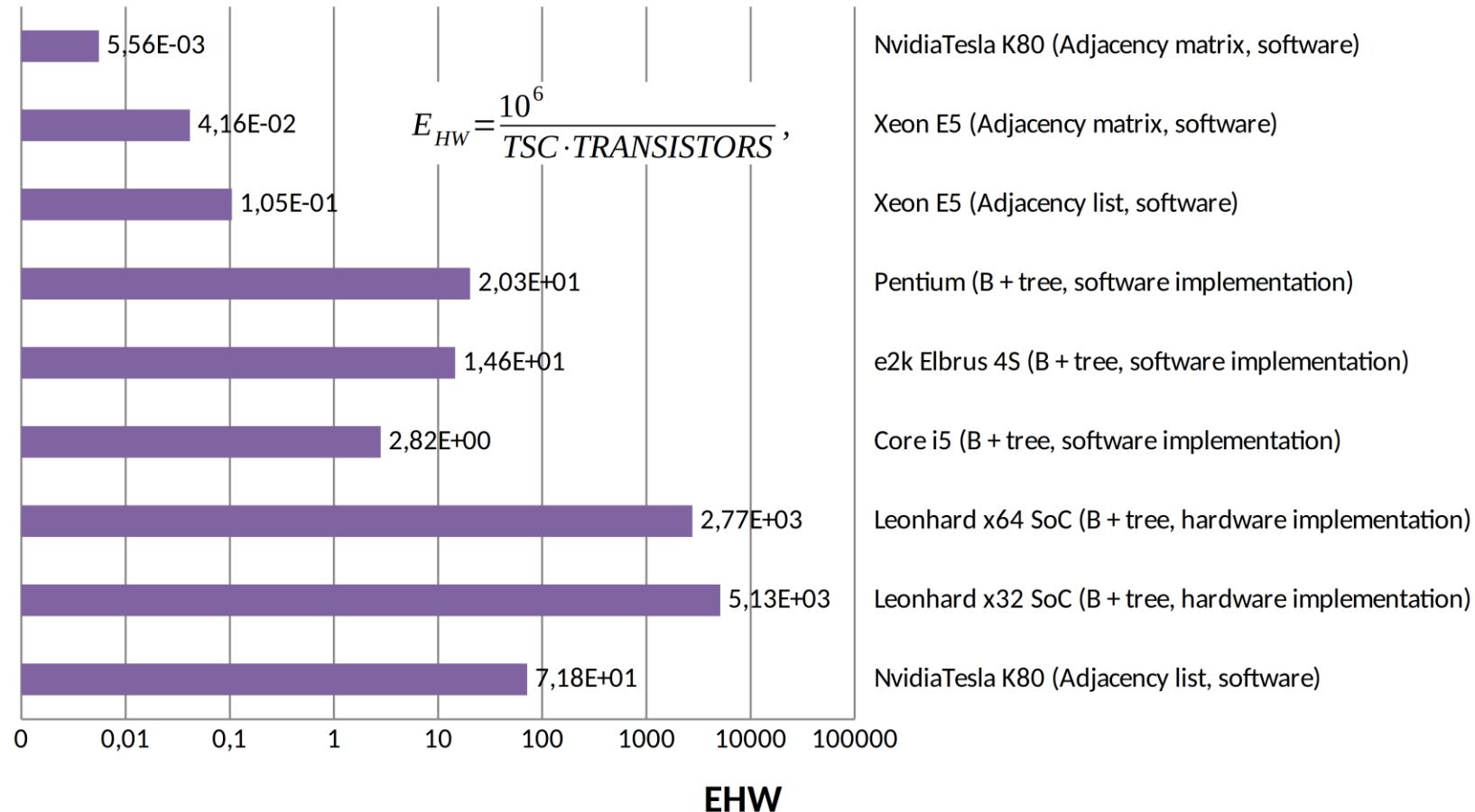
Performance experiments

Experiment	Acceleration
Delete (MISD with Embedded Microblaze Single Core)	164.4
Insert (MISD with Embedded Microblaze Single Core)	42.7
Search (MISD with Embedded Microblaze Single Core)	31.4
Delete (MISD with Intel Pentium Single Core)	22.8
Dijkstra's Algorithm (MISD with Intel Pentium Single Core)	19.4
Search (MISD with Intel Pentium Single Core)	15.3
Depth-First Search (MISD with ARM11 Single Core)	12.9
Dijkstra's Algorithm (MISD with e2k Eight Cores)	12.7
Breadth-First Search (MISD with ARM11 Single Core)	12.3
Delete (MISD with Intel Quad Core)	11.8
Prim's Algorithm (MISD with ARM11 Single Core)	10.3
Search (MISD with Intel Core Quad Core)	9.8
Dijkstra's Algorithm (MISD with Intel Core Quad Core)	7.6
Kruskal's Algorithm (MISD with ARM11 Single Core)	7.8
Insert (MISD with Pentium Single Core)	5.7
Insert (MISD with Intel Core Quad Core)	3.2
Depth-First Search (MISD with Intel Quad Core)	3.2
Breadth-First Search (MISD with Intel Quad Core)	3.0

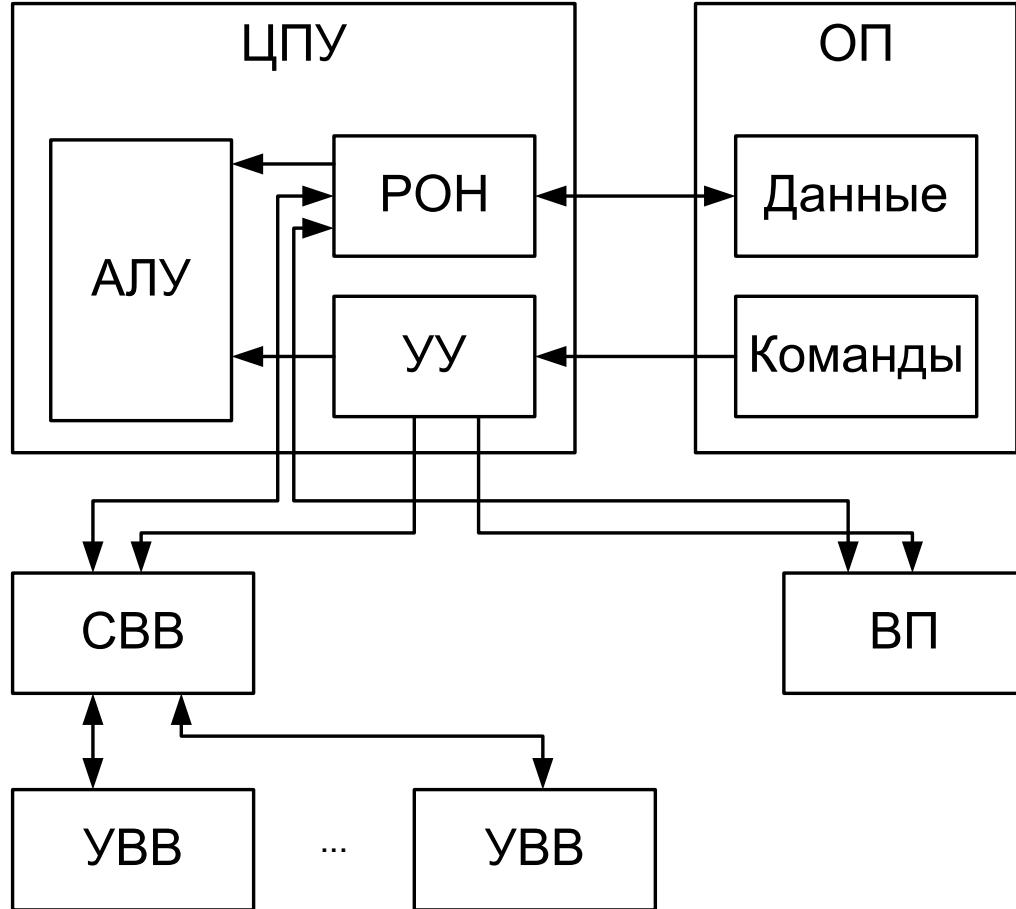
Сравнение аппаратной эффективности на примере алгоритма на графах



Сравнение аппаратной эффективности на примере алгоритма на графах



ЭВМ с непосредственными связями



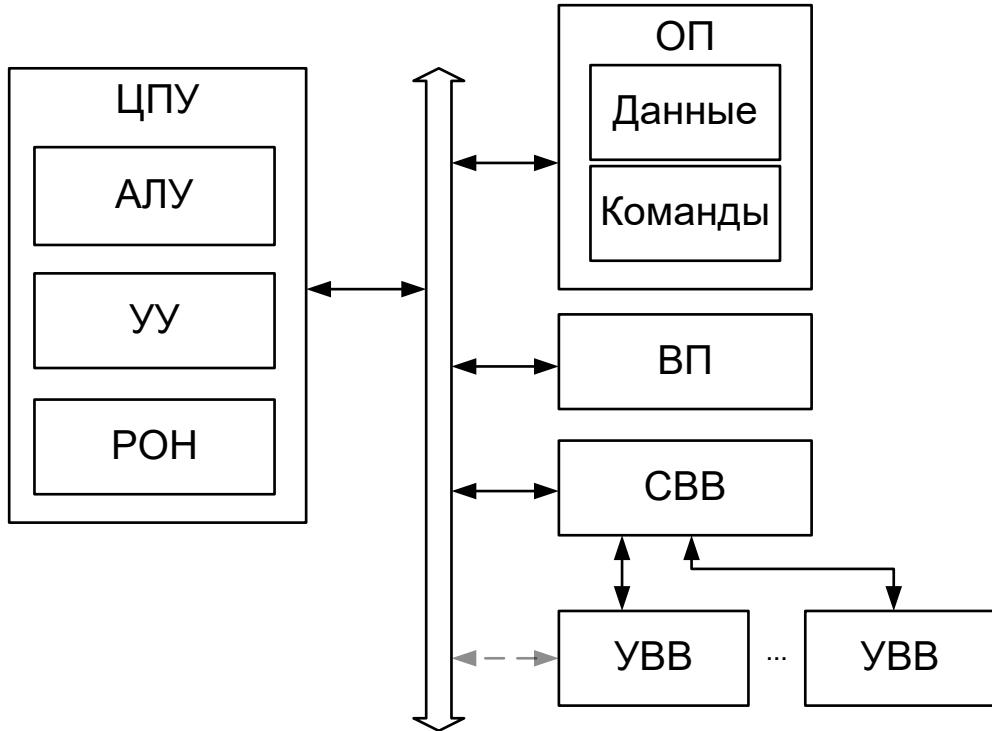
(+) При построении оптимальных линий связи вычислительная машина обладает максимальным быстродействием.

(-) Ограничение на количество выводов микросхем не позволяет организовать широкие шины.

(-) Канал между ОП и ЦПУ является узким местом.

(-) Реконфигурация системы требует изменения характеристик линий связи.

ЭВМ с магистральной структурой

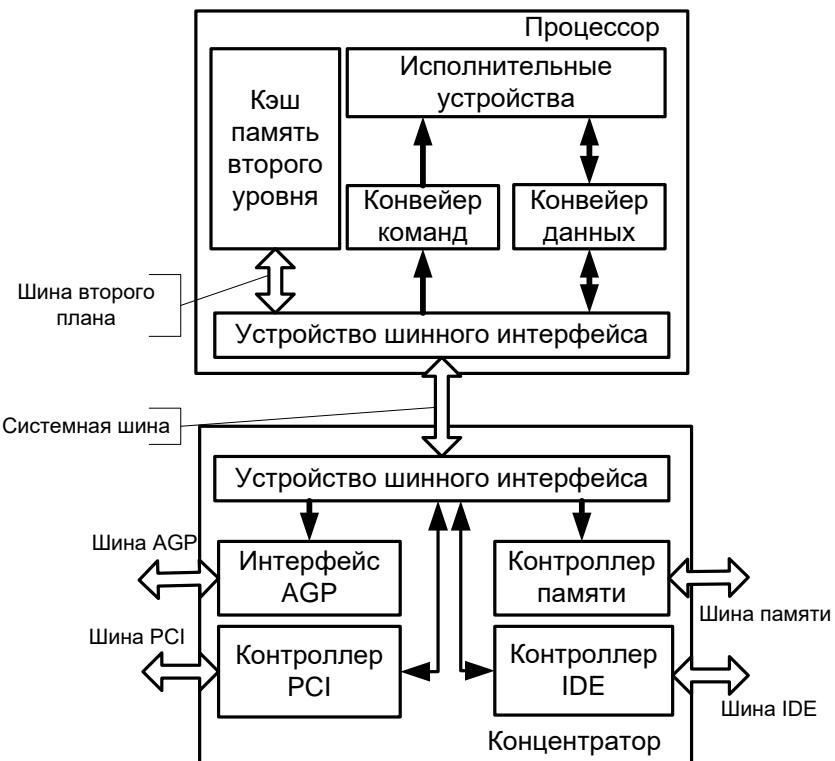
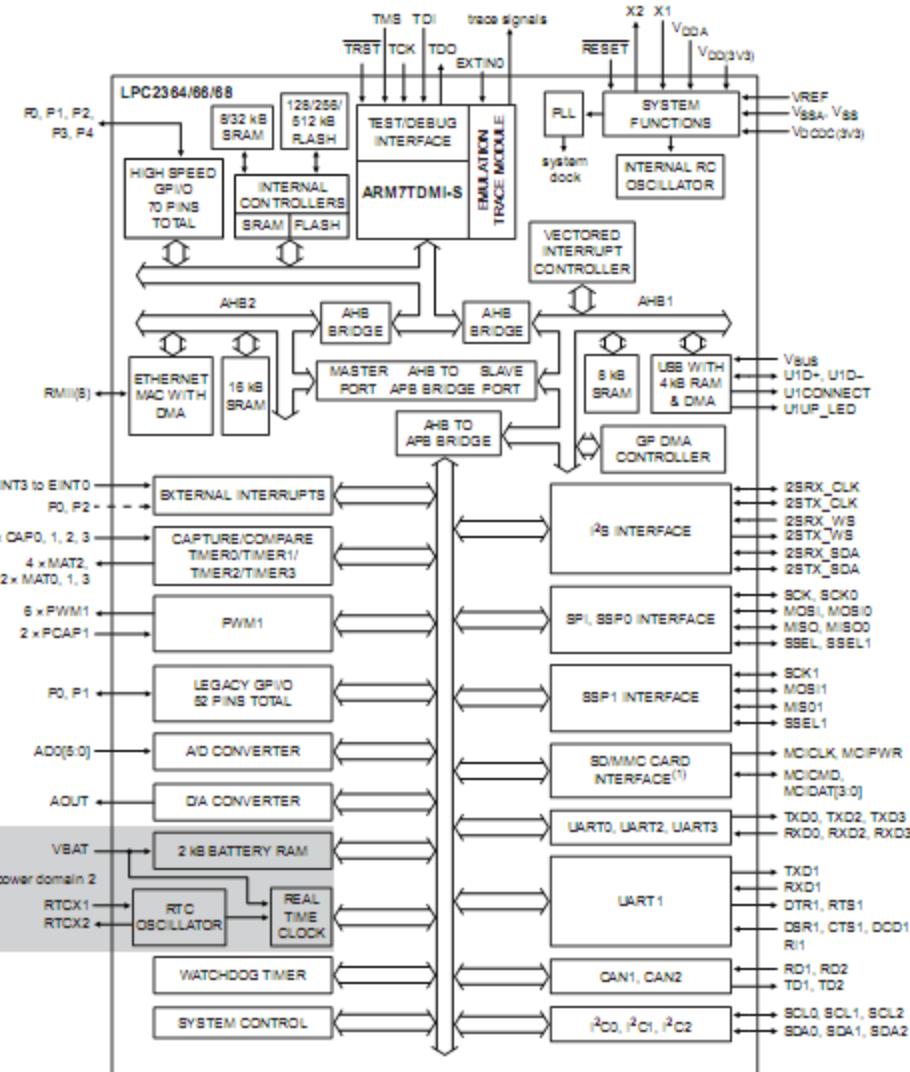


(+) Общая шина позволяет легко реконфигурировать систему.

(-) Шина является узким местом.

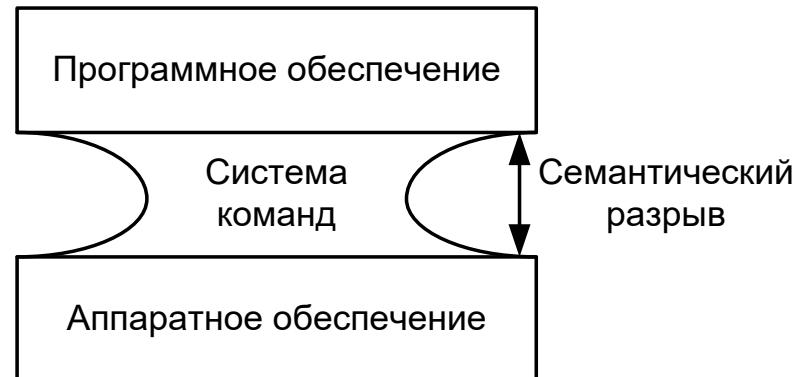
- Шина, используемая всеми устройствами системы для передачи данных называется системной.
- Для разгрузки системной шины используют иерархию шин.
- По назначению, разделяют шины адреса, шины данных и шины управления.

Примеры построения ЭВМ с иерархией шин



Основные тенденции развития ЭВМ

- Повышение степени интеграции элементной базы
 - Увеличение набора команд
 - Увеличение степени аппаратной поддержки.
- Наличие семантического разрыва



Проблема семантического разрыва

Технология программирования непрерывно развивается, что позволяет увеличивать функциональность программ и сокращать время их разработки. Создание проблемно-ориентированных языков высокого уровня усугубляет принципиальное отличие языка машинных команд, реализуемого компьютером, от языков, используемых при написании программ. Данная проблема носит название "семантического разрыва" и выражается в неоправданном падении производительности вычислительной системы.

Архитектура системы команд

В команде указывается, какую операцию выполнять (КОП), над какими операндами выполнять операцию, а также куда поместить operand.



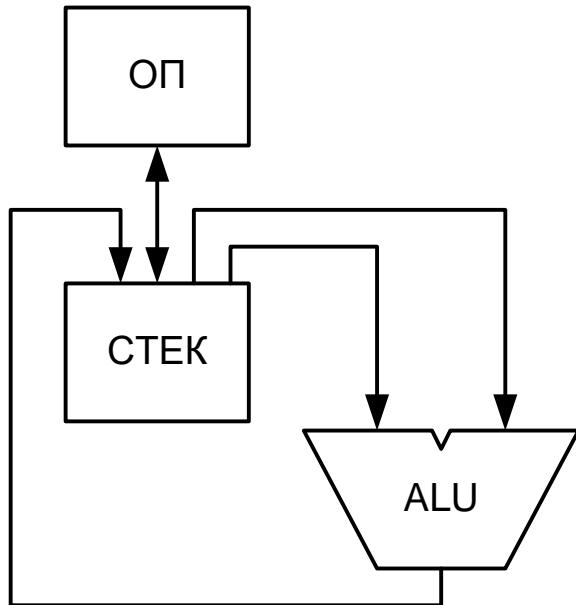
RISC – Reduced Instruction Set Computer; CISC – Complex Instruction Set Computer;
VLIW – Very Long Instruction Word; ROSC - Removed Operand Set Computer

Сравнение CISC, RISC и VLIW архитектур СК

Характеристика	CISC	RISC	VLIW
Длина команды	Различная	Однаковая	Однаковая
Расположение полей в командах	Различное	Однаковое	Однаковое
Количество архитектурных регистров	Малое. Регистры специализированные	Большое. Регистры универсальные	Большое. Регистры универсальные
Доступ к памяти	Кодируется в команде. Выполняется по микрокоманде	Выполняется по специальной команде	Выполняется по специальной команде
Длительность выполнения команд	Различная	Однаковая (для большинства команд)	Различная

Стековая архитектура СК

(+) При размещении операндов в стековой памяти (LIFO) архитектура команд упрощается (большое количество действий выполняется аппаратно)



Операции:

- занесение в стек (PUSH);
- извлечение из стека (POP);
- выполнение действий над стеком
(извлечение operandов из вершины стека, выполнение действий, помещение результата в вершину стека)

Для выполнение арифметических операций их преобразуют к постфиксной форме
(Польской записи).

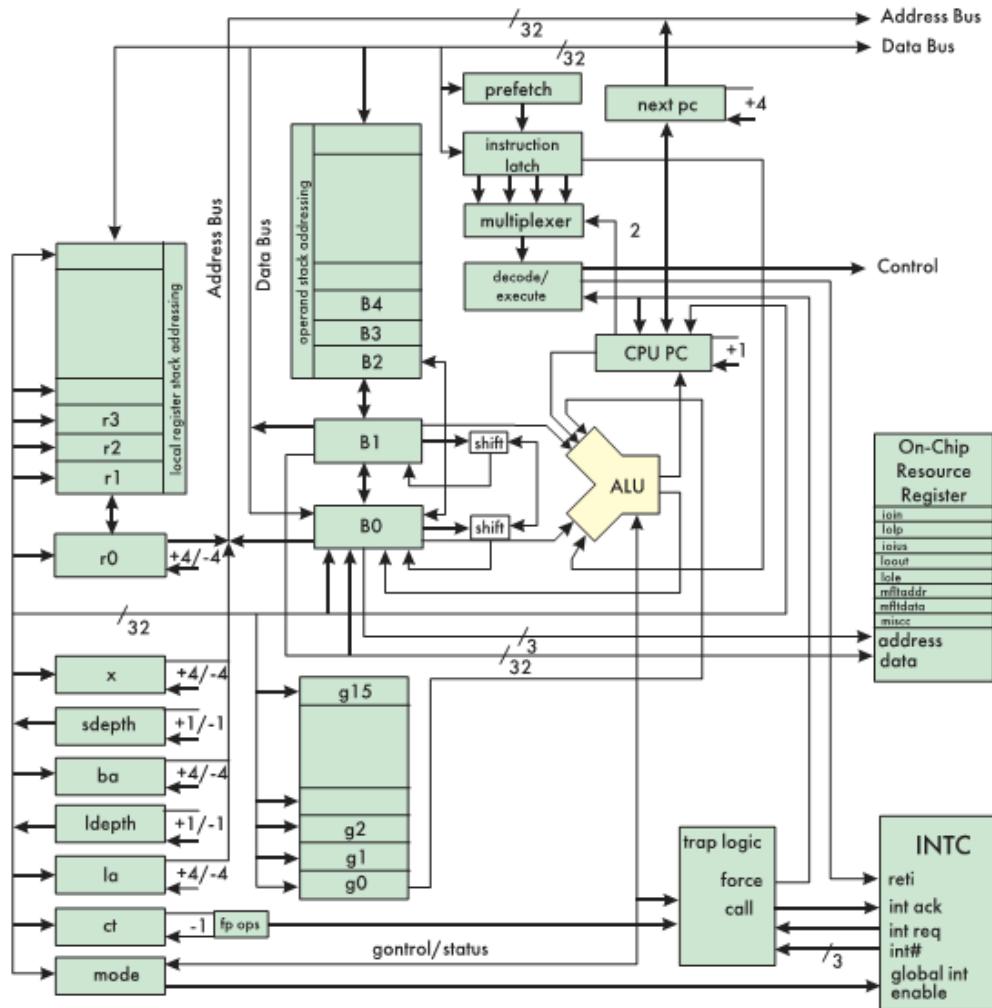
Пример: $a = a + b * (c - d)$; Постфиксная форма: abcd-*+;

Действия: PUSH a; PUSH b; PUSH c; PUSH d; SUB; MUL; ADD; POP a.

- (-) Отсутствие прямого доступа к памяти ограничивает область применения.
- (-) Сложность организации параллельной обработки.

Стековые процессоры (Форт-процессоры)

Блок-схема микропроцессора IGNITE



Сравнение выполнения программы на RISC-процессоре и на стековом микропроцессоре

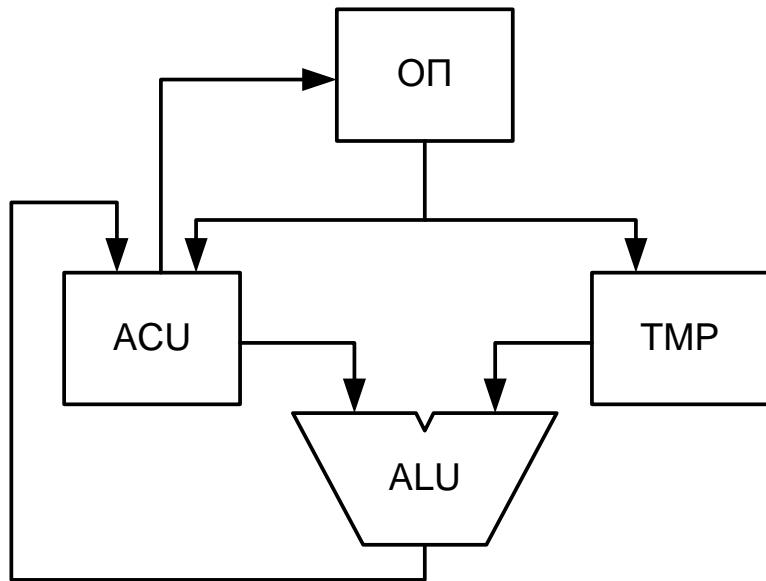
Номер команды	RISC MPU	IGNITE
1	add #1, g2, g5	push g1
		push g2
		inc #1
2	sub g1, g5, g5	sub
3	add g5, g3, g5	push g3
		add
4	shl g4, #1, temp	push g4
		shl #1
5	sub g5, temp, g5	sub
		pop g5
	Всего 20 байт	Всего 10 байт

Набор микросхем TDS9092 FORTH CHIPS



Аккумуляторная архитектура СК

Один из операндов должен обязательно находиться в специальном регистре-аккумуляторе. Результат также сохраняется в аккумулятору.



Операции:

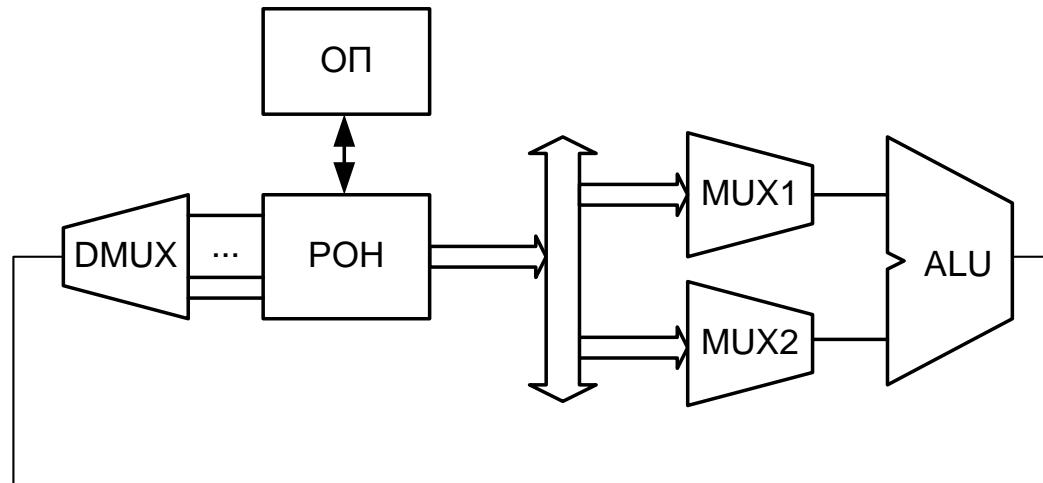
- занесение в аккумулятор (LOAD);
- извлечение из аккумулятора (STORE);
- выполнение действий над operandами (извлечение первого операнда из аккумулятора, извлечение второго операнда из ОП и помещение во временный теневой регистр TMP, выполнение действий, помещение результата в аккумулятор).

Пример: $a = a + b * (c - d)$; Определение троек: $T1=c-d$; $T2=b*T1$; $T3=a+T2$; Действия: LOAD c; SUB D; MUL b; ADD a; STORE a.

- (+) В команде необходимо указывать только адрес второго операнда.
- (+) Ускоряются длинные вычисления ($a*b/c+d-e$).
- (-) Наличие одного аккумулятора является узким местом, т.к. временно ненужный результат необходимо перезаписывать в другой регистр или ОП.

Регистровая архитектура СК

В состав процессора входит большое количество однотипных регистров. В команде необходимо указать номера регистров, хранящих операнды, а также номер регистра операнда.



Для данной архитектуры возможны варианты размещения operandов: оба операнда в памяти; один operand в памяти и один в РОН; оба операнда в РОН.

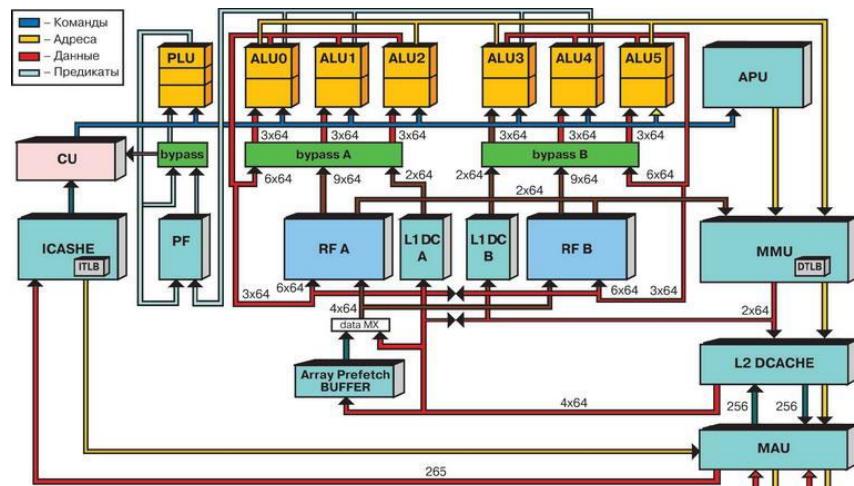
Для уменьшения размерности команд и для упрощения декодирования накладывают ограничения на размещение operandов.

Архитектура VLIW – Very Long Instruction Word

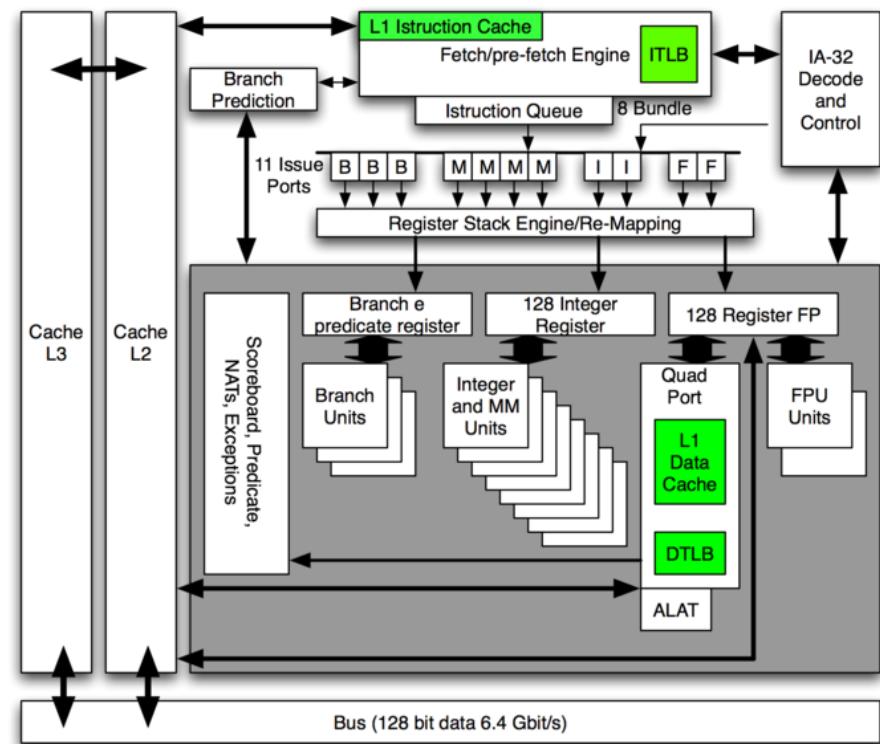
В процессорах VLIW задача распределения решается во время компиляции и в инструкциях явно указано, какое вычислительное устройство должно выполнять какую команду.

Эльбрус-3 и его микропроцессорное исполнение Эльбрус 2000 (Е2К) также являются VLIW процессорами.

ПРИМЕР



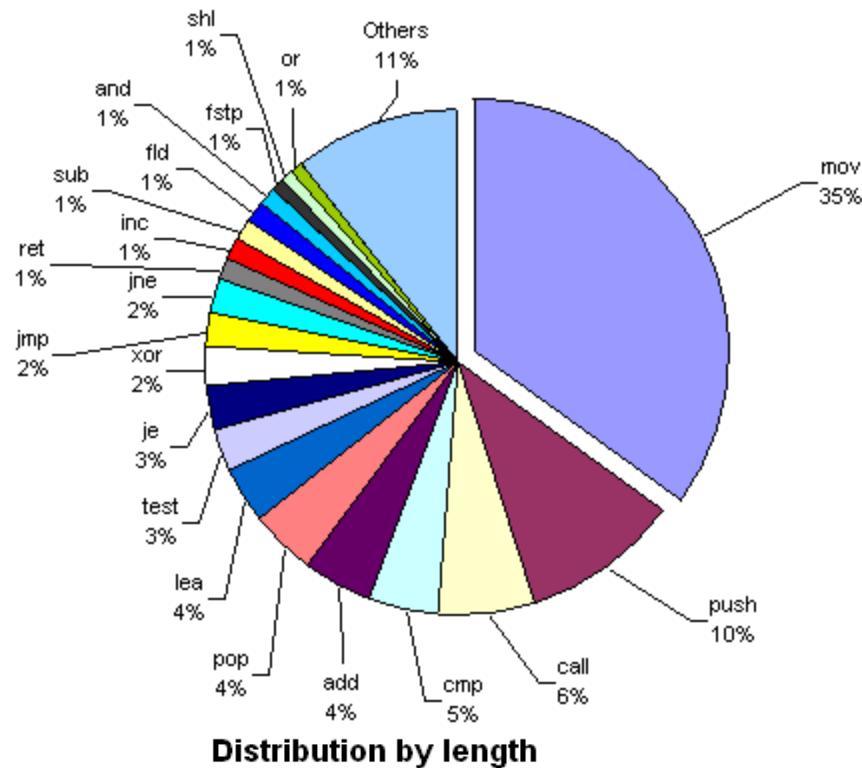
Микропроцессор Intel Itanium имеет как традиционную систему команд IA-32, так и систему команд «с явным параллелизмом» (англ. Explicitly Parallel Instruction Computing, EPIC), исполняемую VLIW-ядром



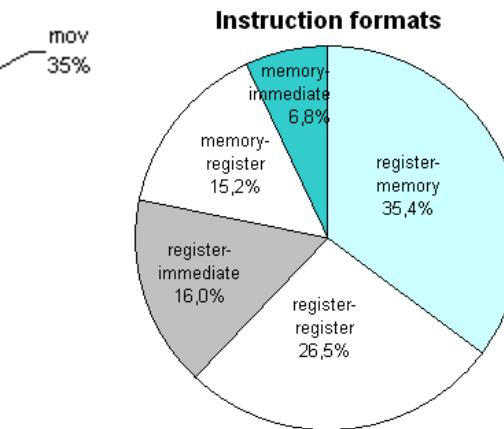
Вариант	(+)	(-)
Оба операнда находятся в регистрах	Простота аппаратной реализации. Простота параллельной обработки.	Избыточность в команде из-за сложности кодирования с кратностью 8 бит
Один операнд находится в регистре, а один в памяти	Код компактен. Данные поступают в ALU без промежуточного хранения в РОН	Наличие адреса в команде усложняет дешифрацию и сокращает возможное кол-во РОН, адресуемых в команде.
Оба операнда находятся в памяти	Код наиболее компактен. Возможность выполнения простых действий без занесения в РОН	Выполняется дольше других вариантов размещения. Команды имеют максимальную длину. Из-за наличия коротких и длинных команд трудно оптимизировать тракты передачи данных и декодеры инструкций

Статистические данные для x86 команд

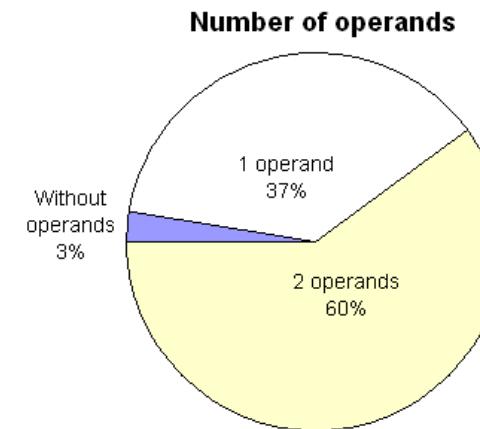
Top 20 instructions of x86 architecture



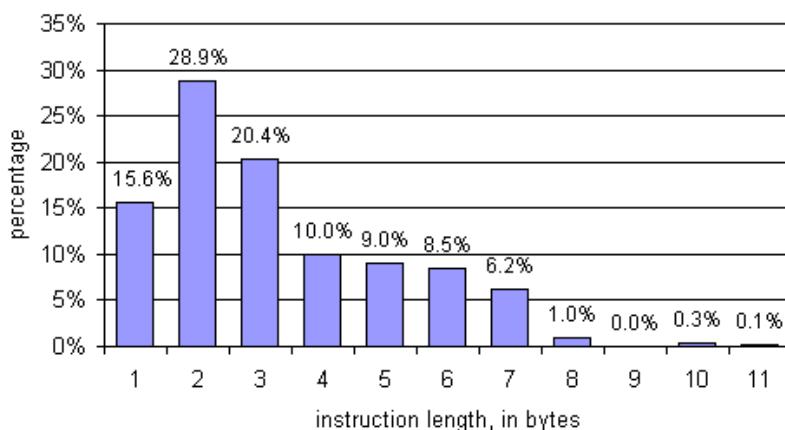
Distribution by length



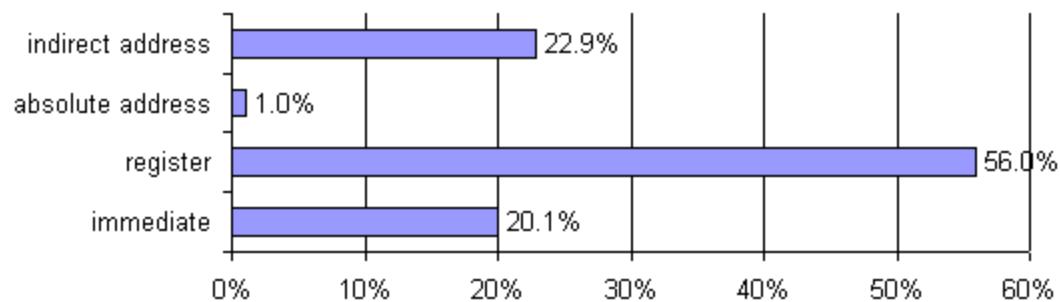
Instruction formats



Number of operands



Operand types in x86 architecture



Типы команд.

- Команды пересылки данных.
 - регистр-регистр
 - регистр-память
 - память-память
- Команды арифметической и логической обработки (сложение, вычитание, умножение, деление, инкремент, декремент, сравнение, операции над ЧПЗ, логические операции, операции сдвига).
Сдвиг: логический, арифметический, циклический, циклический через дополнительным разряд.
- Команды работы со строками (могут быть реализованы набором других команд, однако удобны при работе с символьной информацией).
- Команды векторной обработки (позволяет выполнять однотипные действия над большим количеством однородных данных). Пример арифметики с насыщением:
$$\begin{array}{r} 1011\ 0111\ 1010 \\ +\ \underline{0001\ 1001\ 1000} \\ \hline 1100\ 1111\ 1111 \end{array}$$
- Команды преобразования: служат для табличного преобразования данных из одной системы кодов в другую (2-10 <-> 2)

- Команды ввода/вывода. Служат для управления, проверки состояния и обмена данными с периферийными устройствами.

- Команды вывода в порт

- Команды ввода из порта.

- Команды управления потоком команд. Данные команды служат для указания очередности выполняемых команд.

Вычисление адреса очередной команды может выполняться несколькими способами:

- увеличением адреса на длину исполненной (естественный порядок).

- изменением адреса на длину следующей (перешагивание)

- изменением адреса на значение, указанное в текущей команде (короткий переход).

- непосредственное указание следующей команды (длинный переход).

Перечисленные команды могут выполняться лишь по некоторому условию (уловные переходы).

Команды условного перехода составляют (до 80%) команд управлений.

Команды безусловного перехода: вызовы и возвраты из процедур, и.т.д.

Форматы команд.

Операционная часть

Адресная часть

1. Четырехадресная команда.

КОП	1 операнд	2 операнд	результат	Адр след ком.
-----	-----------	-----------	-----------	---------------

2. Трехадресная команда

КОП	1 операнд	2 операнд	результат
-----	-----------	-----------	-----------

3. Двухадресная команда.

КОП	1 операнд	2 оп-д/результат	Характерна для CISC-архитектуры
-----	-----------	------------------	---------------------------------

4. Аккумуляторная архитектура

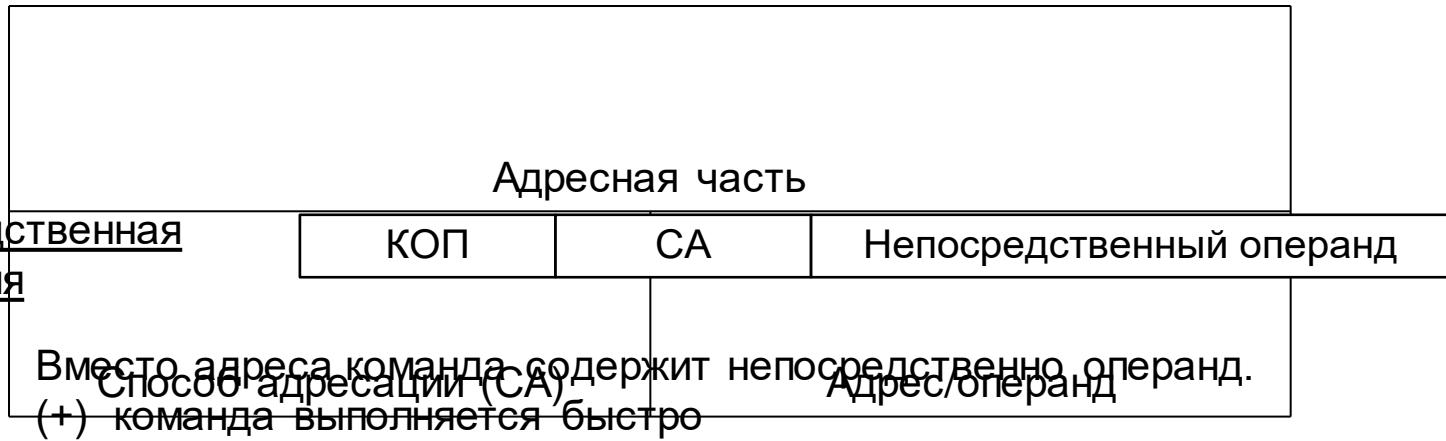
КОП	1 операнд	Второй operand хранится в аккумуляторе. Данный формат команд характерен для RISC-архитектур.
-----	-----------	---

5. Нульоперандная команда.

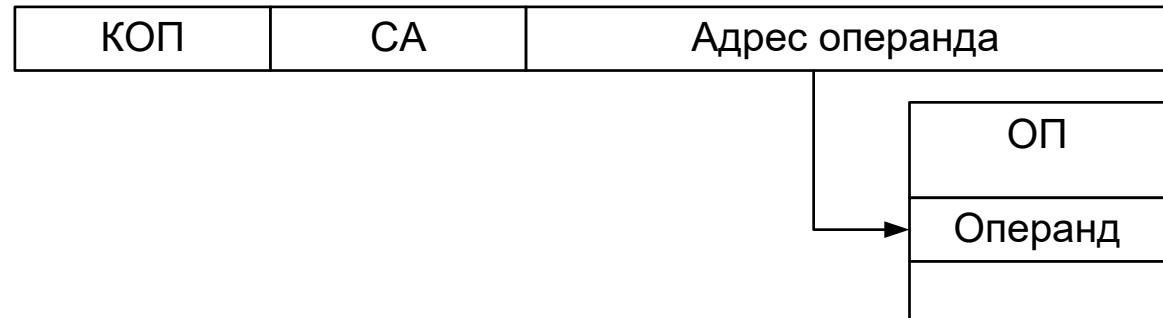
КОП

Способы адресации

Непосредственная адресация



Прямая адресация



- Адрес в команде является адресом операнда
- (+) если операнд находится в памяти, то это самый быстрый способ указать на него
- (-) заранее определенный адрес влияет на переносимость программы.
- (-) Адрес занимает много места

Неявная адресация

КОП	СА
-----	----

Операнд подразумевается (следует из КОП).

(+) Команда занимает мало места

(-) только такие командах нельзя использовать для построение всей системы команд.

Регистровая адресация

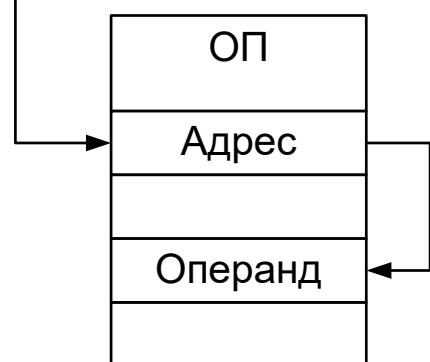
Адрес в команде указывает не на ячейку ОП, а на регистр.

(+) Быстрее прямой адресации

(-) Количество регистров ограничено

Косвенная адресация

КОП	СА	Адрес операнда
-----	----	----------------



Адрес в команде указывает на ячейку памяти, в которой находится адрес операнда.

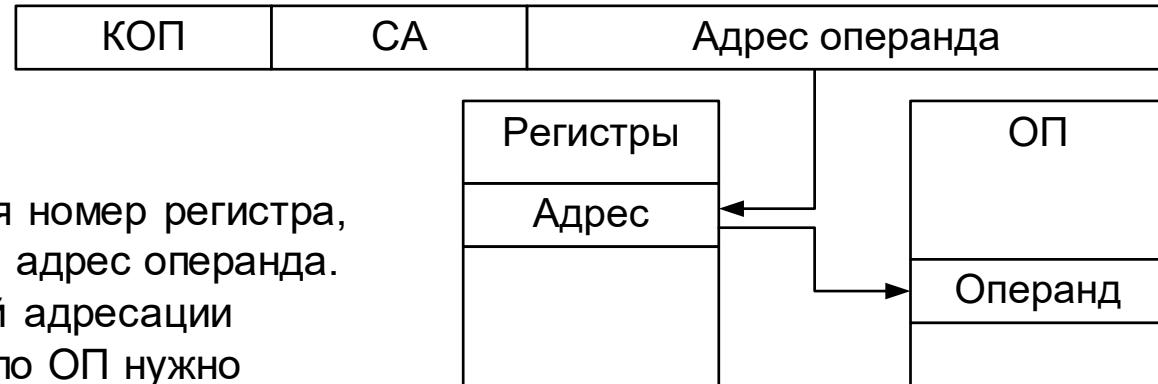
(+) удобна для обработки структурных типов данных.

(-) приходится осуществлять много обращений к ОП.

Косвенная регистровая адресация

В команде содержится номер регистра, в котором содержится адрес операнда.

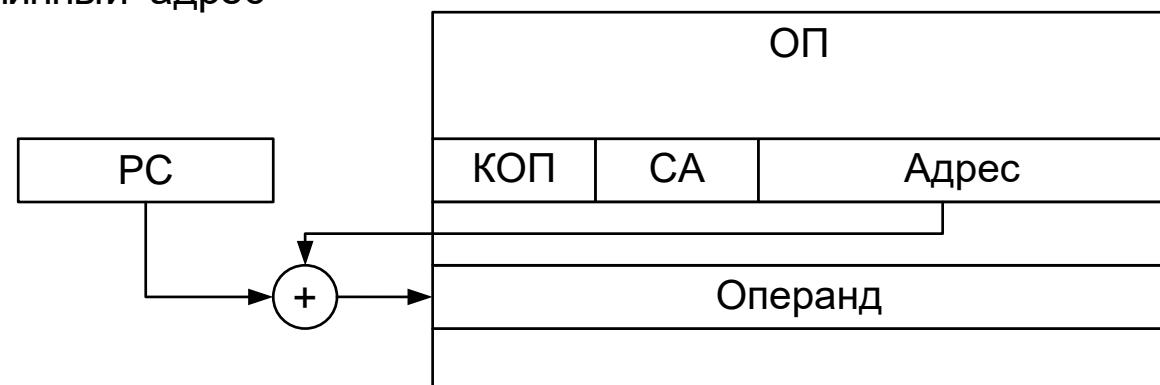
- (+) быстрее косвенной адресации
- (-) для перемещения по ОП нужно менять содержимое регистра



Относительная адресация

Адрес вычисляется относительно счётчика команд

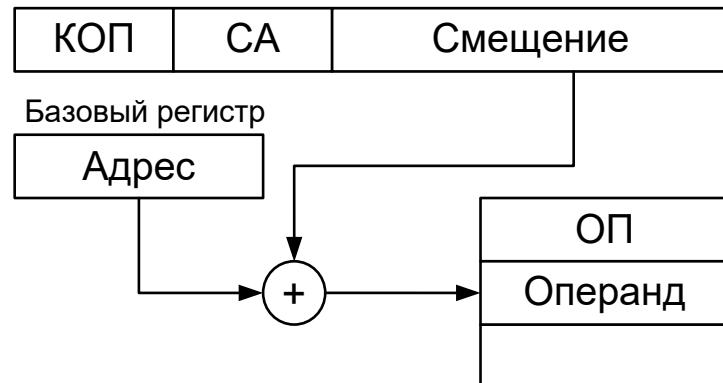
- (+) Код переносим, команды занимают мало места
- (-) Может понадобиться длинный адрес



Базовая регистровая адресация

Адрес в команде представляет собой смещение, которое складывается со значением в базовом регистре для получения адреса операнда

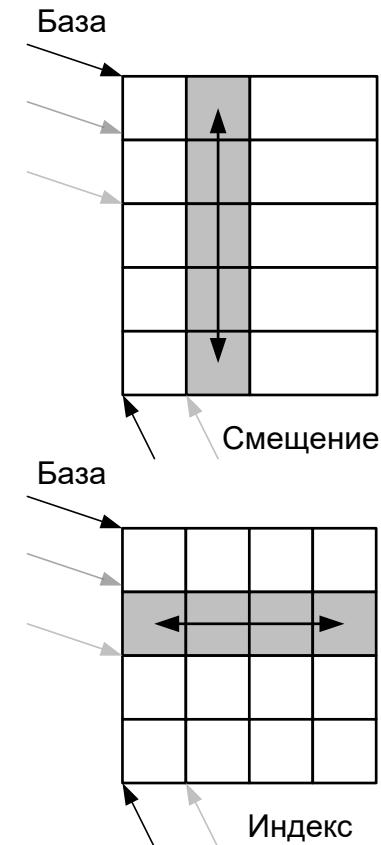
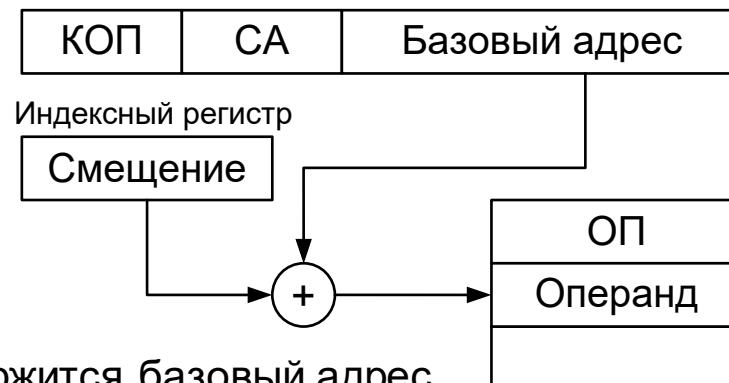
- (+) Удобна для работы со структурами данных, размещаемых динамически.
- (-) Переносимость меньше, чем у относительной адресации



Индексная регистровая адресация

В поле адреса команды содержится базовый адрес, складываемый со значением смещения в индексном регистре.

- (+) Удобна для работы со структурами данных, размещаемых динамически.
- (-) Переносимость меньше, чем у относительной адресации



Автоинкрементная/автодекрементная адресация

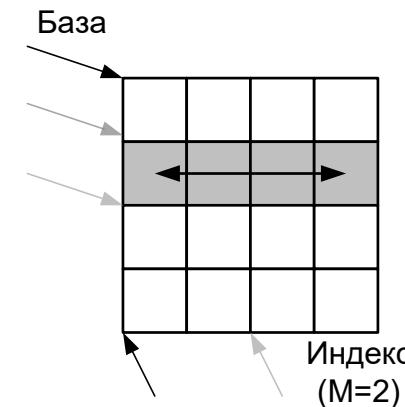
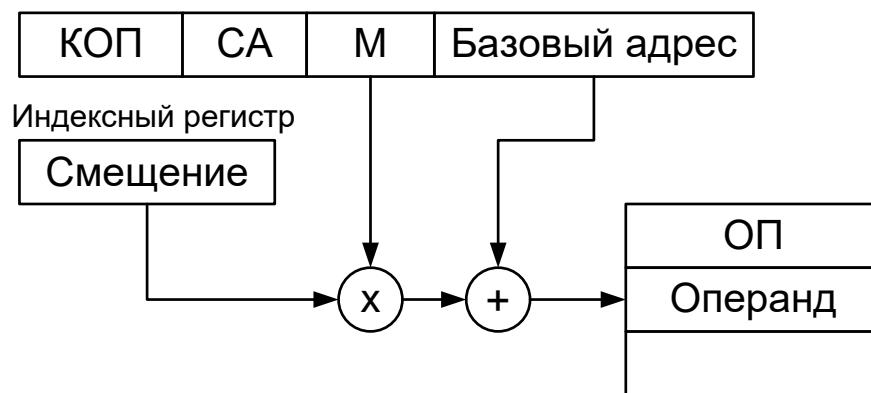
Разновидность регистровой индексной или базовой адресации. До или после выполнения команды значение базового или индексного регистра увеличивается/уменьшается на единицу.

- (+) Способ адресации удобен для команд обработки строк.
- (-) Автоматическое изменение часто требуется выполнять на величину, большую единицы.

Индексная адресация с масштабированием

Индексный регистр умножается на масштаб M и суммируется с базовым адресом из команды.

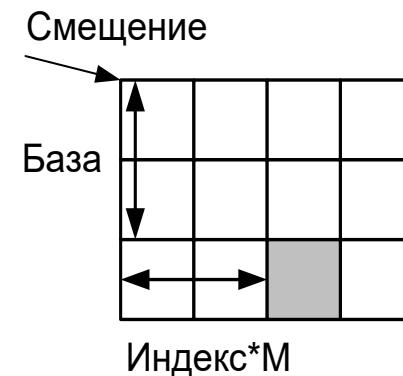
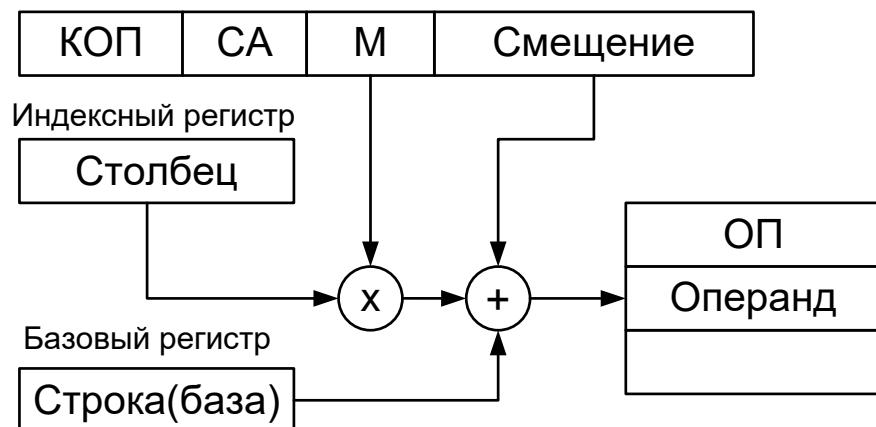
- (+) Удобен для модификации адреса на величину M .
- (-) Вычисление адреса замедляется, т.к. требуется выполнять умножение.



Базовая индексная адресация с масштабированием

Адрес определяется по формуле Адрес=Индекс*Масштаб+База+Смещение.

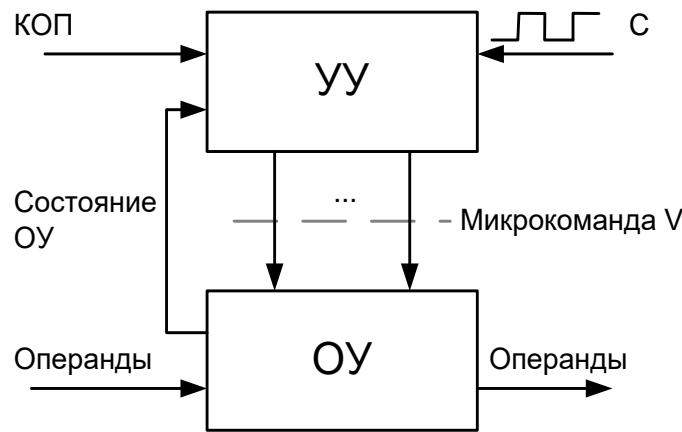
- (+) Базовая индексная адресация с масштабированием часто используется при обращении к системным таблицам, находящимся в ОП (таблица дескрипторов, таблицы страниц, таблица векторов прерываний и т.д.)
- (-) Ограничено на величину M (M=1,2,4,8).



II. Устройства управления ЭВМ

- Принципы микропрограммного управления.
- Классификация устройств управления.
- Управляющие устройства с жесткой и программируемой логикой.
- Примеры реализации устройств управления на основе автоматов Мили и Мура.

Принципы микропрограммного управления



Любое цифровое устройство можно рассматривать, как совокупность операционного и управляемого блока.

Любая команда или последовательность команд реализуется в операционном блоке за несколько тактов

Последовательность сигналов управления должна выдаваться устройством управления в соответствии с поступающей на вход командой и текущим состоянием операционного блока

Состояние линий управления в каждом такте задает микрокоманду. Совокупность микрокоманд, необходимых для реализации команды, называется микропрограммой.

Устройство управления реализуется в виде автомата.

Классификация устройств управления:

По типу автомата:

- Автомат Мили.
- Автомат Мура.

По способу реализации:

- Устройство управления с жесткой логикой.

Функции выдачи сигналов управления и разделения во времени сигналов управления реализуются с помощью комбинационных схем и триггерной памяти.

- Устройство управления с программируемой логикой.

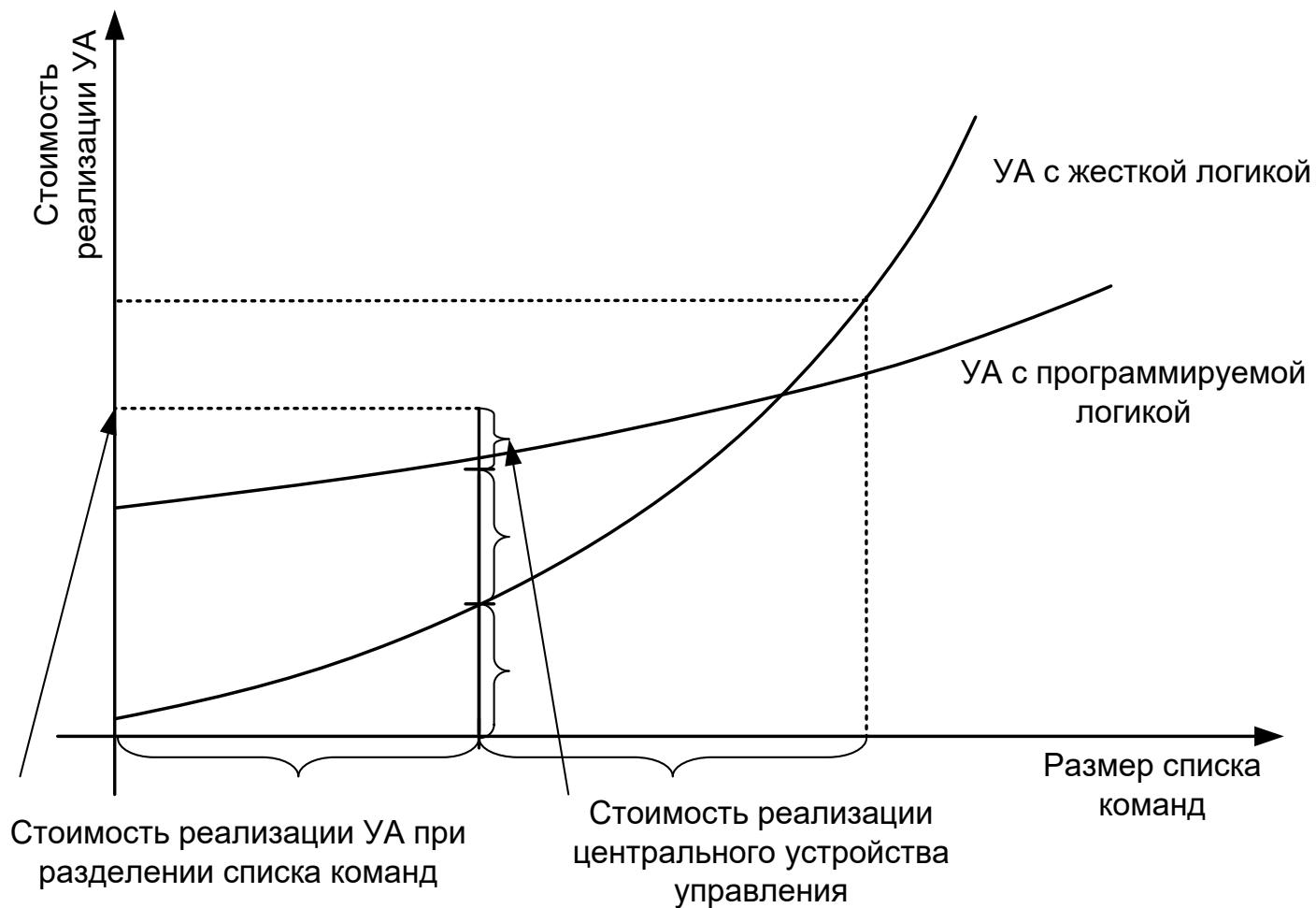
Каждой выполняемой операции ставится в соответствие совокупность хранимых в памяти слов (микрокоманд), каждая из которых содержит информацию о микрооперациях, подлежащих исполнению в текущем такте.

(+) Простота модификации и наращивания.

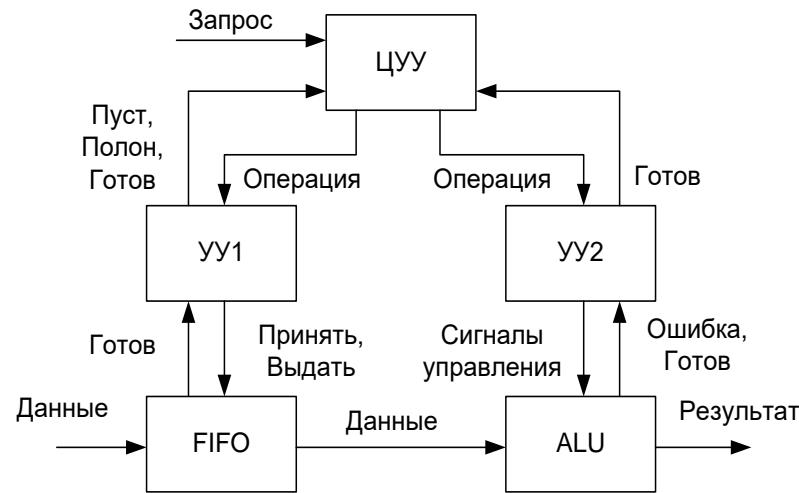
(-) Невысокое быстродействие для простых

устройств.

Сравнение способов реализации УУ



Пример декомпозиции УУ



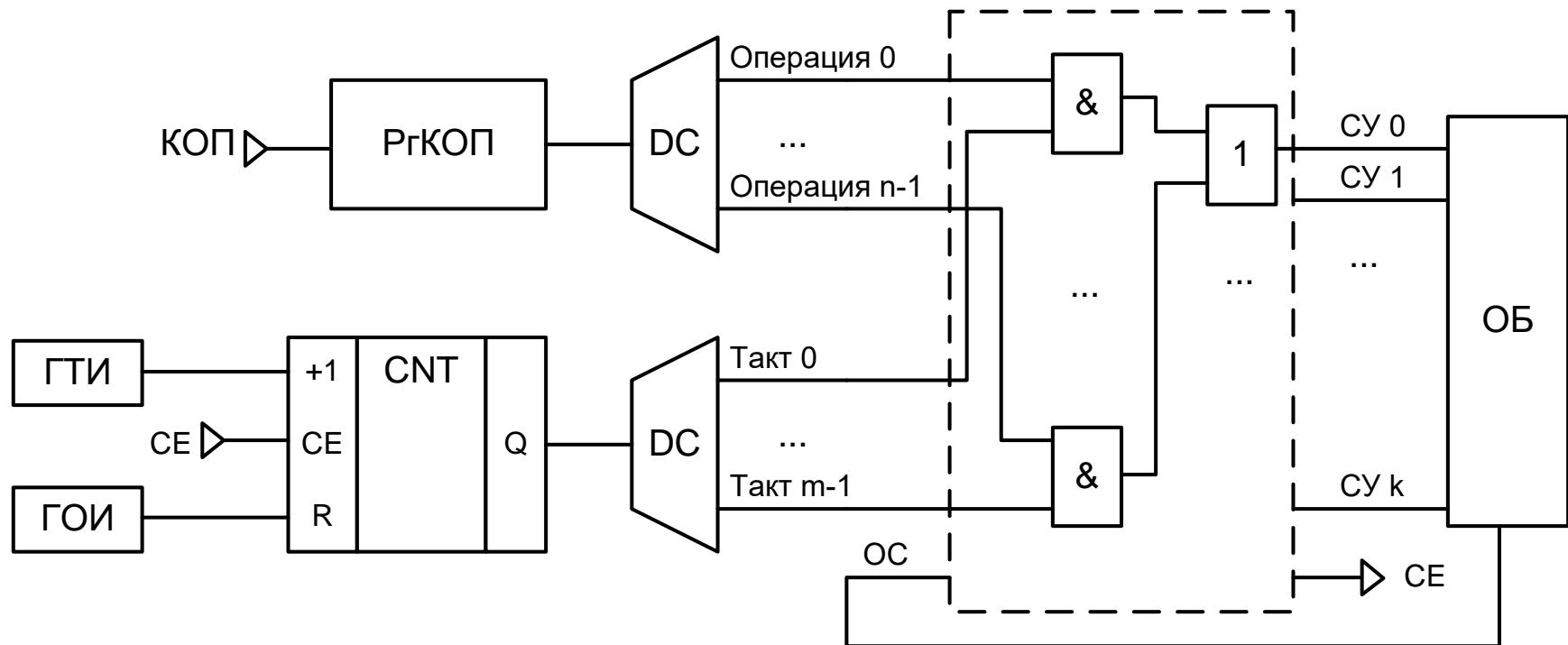
По способу кодирования микрокоманд:

- Минимальное кодирование (горизонтальное).
- Максимальное кодирование (вертикальное).
- Горизонтально-вертикальное кодирование.
- Вертикально-горизонтальное кодирование.
- Кодирование с помощью памяти нанокоманд.

По способу исполнения команд:

- Последовательные.
- Конвейерные.

Пример реализации управляющего автомата с жесткой логикой



Синтез управляющих автоматов с жесткой логикой

Исходные данные:

- Описание логики функционирования операционного блока (временные диаграммы, словесное описание, таблицы истинности, графы и т.д.).
- Сигналы управления.
- Осведомительные сигналы.
- Временные параметры.

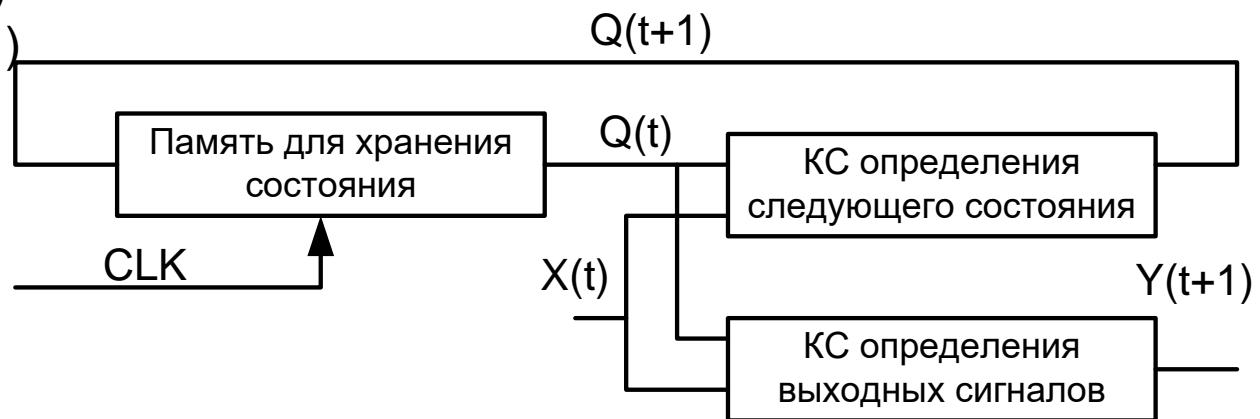
Процедура синтеза:

- Определение алгоритма функционирования управляющего автомата.
- Определение состояний с учетом выбранного типа автомата (Мили или Мура).
- Кодирование состояний автомата.
- Определение логических функций для сигналов управления и их минимизация.
- Определение логических функций перехода

Автомат Мили

$$\begin{cases} Q(t+1) = A (Q(t), x(t)) \\ Y(t+1) = B (Q(t), x(t)) \end{cases}$$

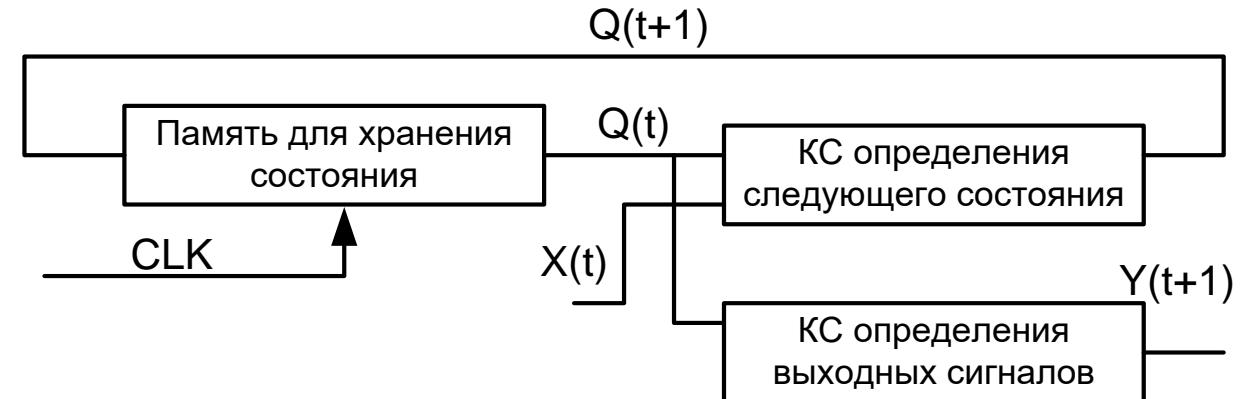
Схема автомата Мили



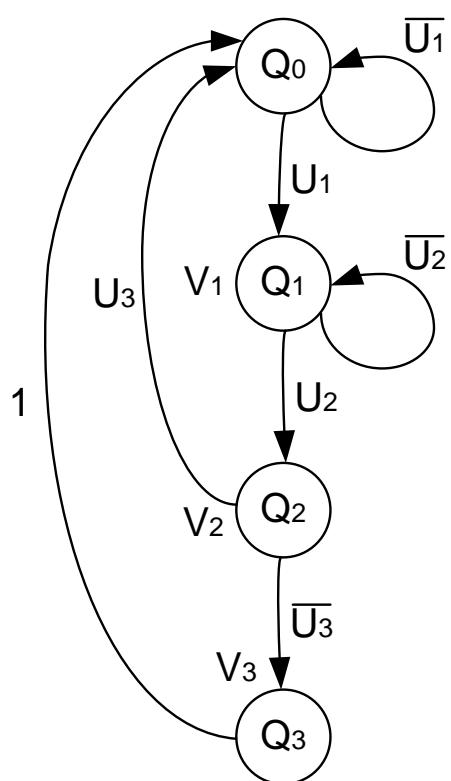
Автомат Мура

$$\begin{cases} Q(t+1) = A (Q(t), x(t)) \\ Y(t+1) = B (Q(t)) \end{cases}$$

Схема автомата Мура



Пример синтеза УУ с ЖЛ на основе автоматов Мили и Мура



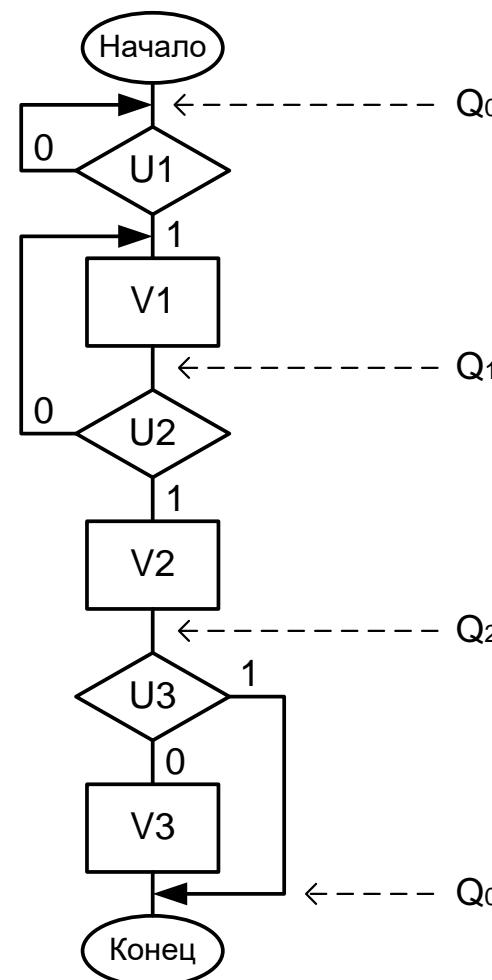
Состояния
автомата Мура

Q_0

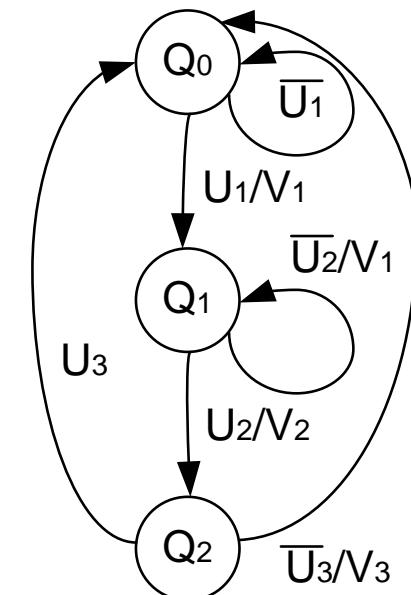
Q_1

Q_2

Q_3



Состояния
автомата Мили



Автомат Мура

$V_i = \bigcup Q_{Vi}$, где Q_{Vi} – состояния автомата, в которых сигнал V_i активен.

$$\left\{ \begin{array}{l} V_1 = Q_1; V_2 = Q_2; V_3 = Q_3; \\ Q_0 = Q_0!U_1 \cup Q_2U_3 \cup Q_3; \\ Q_1 = Q_1!U_2 \cup Q_0U_1 \\ Q_2 = Q_1U_2 \\ Q_3 = Q_2!U_3 \end{array} \right.$$

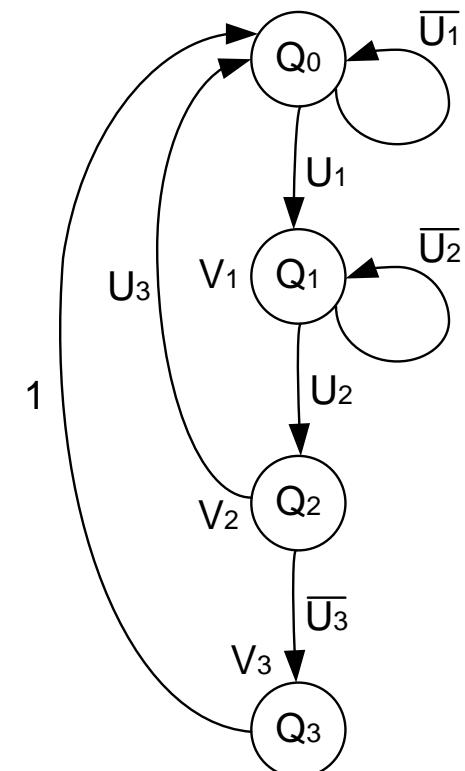
Таблица кодирования
состояний

	D1	D0
Q0	0	0
Q1	0	1
Q2	1	0
Q3	1	1

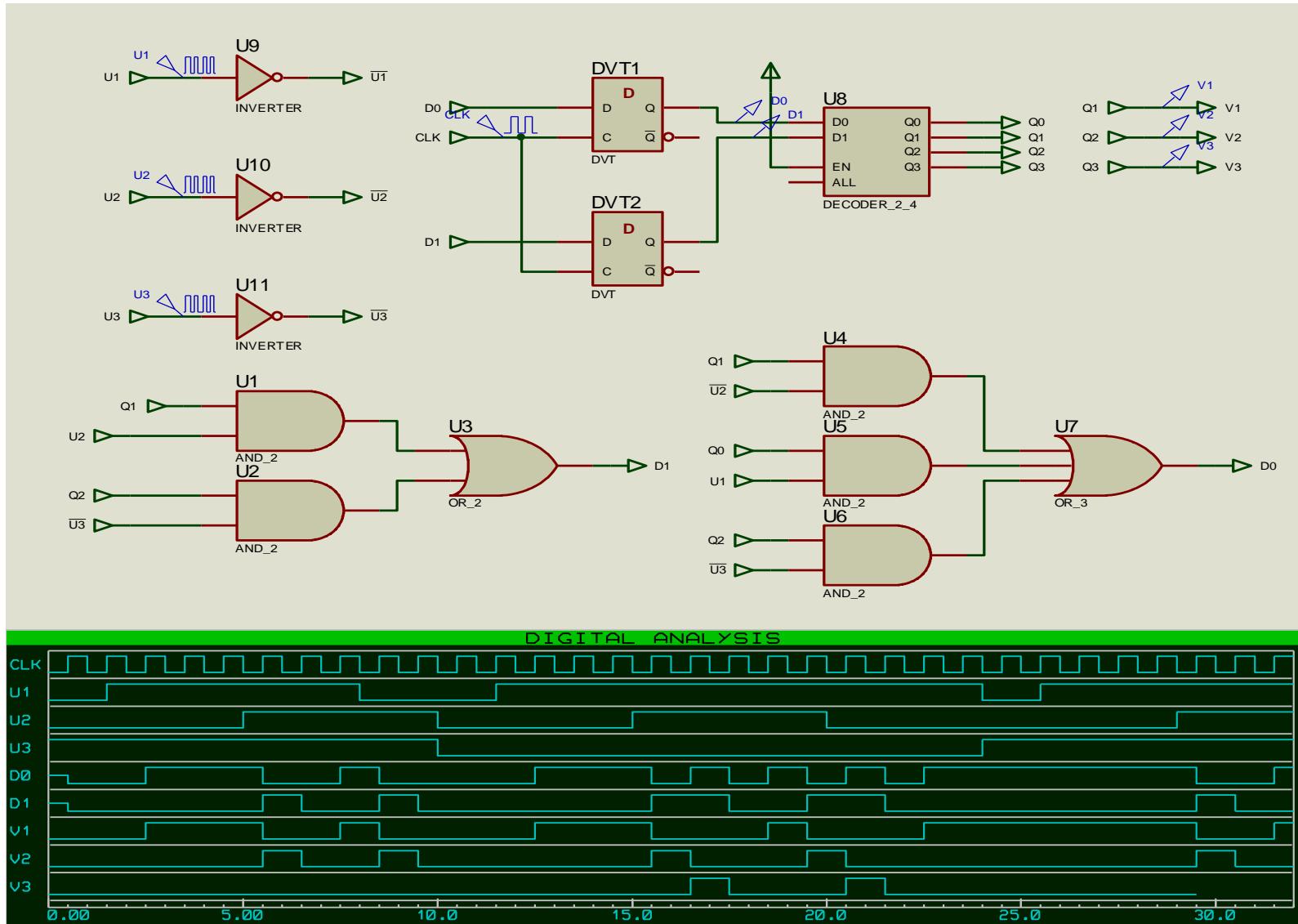
Наиболее используемые способы кодирования состояний: двоичное, One-Hot, Код Грэя

$$D_0(t+1) = Q_1(t) \cup Q_3(t) = Q_1!U_2 \cup Q_0U_1 \cup Q_2!U_3;$$

$$D_1(t+1) = Q_2(t) \cup Q_3(t) = Q_1U_2 \cup Q_2!U_3;$$



Автомат Мура



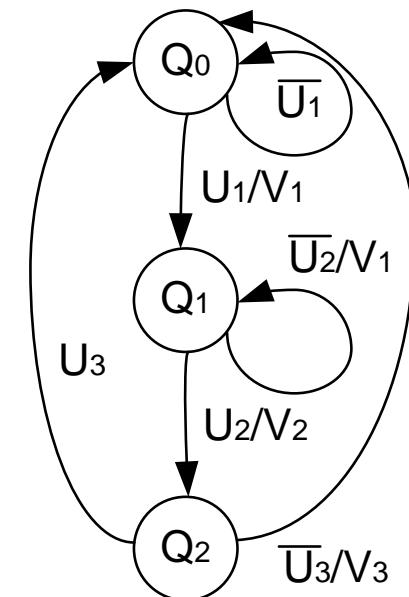
Автомат Мили

$V_i = \bigcup (Q_{Vi} U_j)$, где Q_{Vi} – состояния автомата, в которых сигнал V_i активен, U_j - условие перехода.

$$\left\{ \begin{array}{l} V_1 = Q_0 U_1 \cup Q_1! U_2; V_2 = Q_1 U_2; V_3 = Q_2! U_3; \\ Q_0 = Q_0! U_1 \cup Q_2 U_3 \cup Q_2! U_3; \\ Q_1 = Q_1! U_2 \cup Q_0 U_1 \\ Q_2 = Q_1 U_2 \end{array} \right.$$

Таблица кодирования состояний

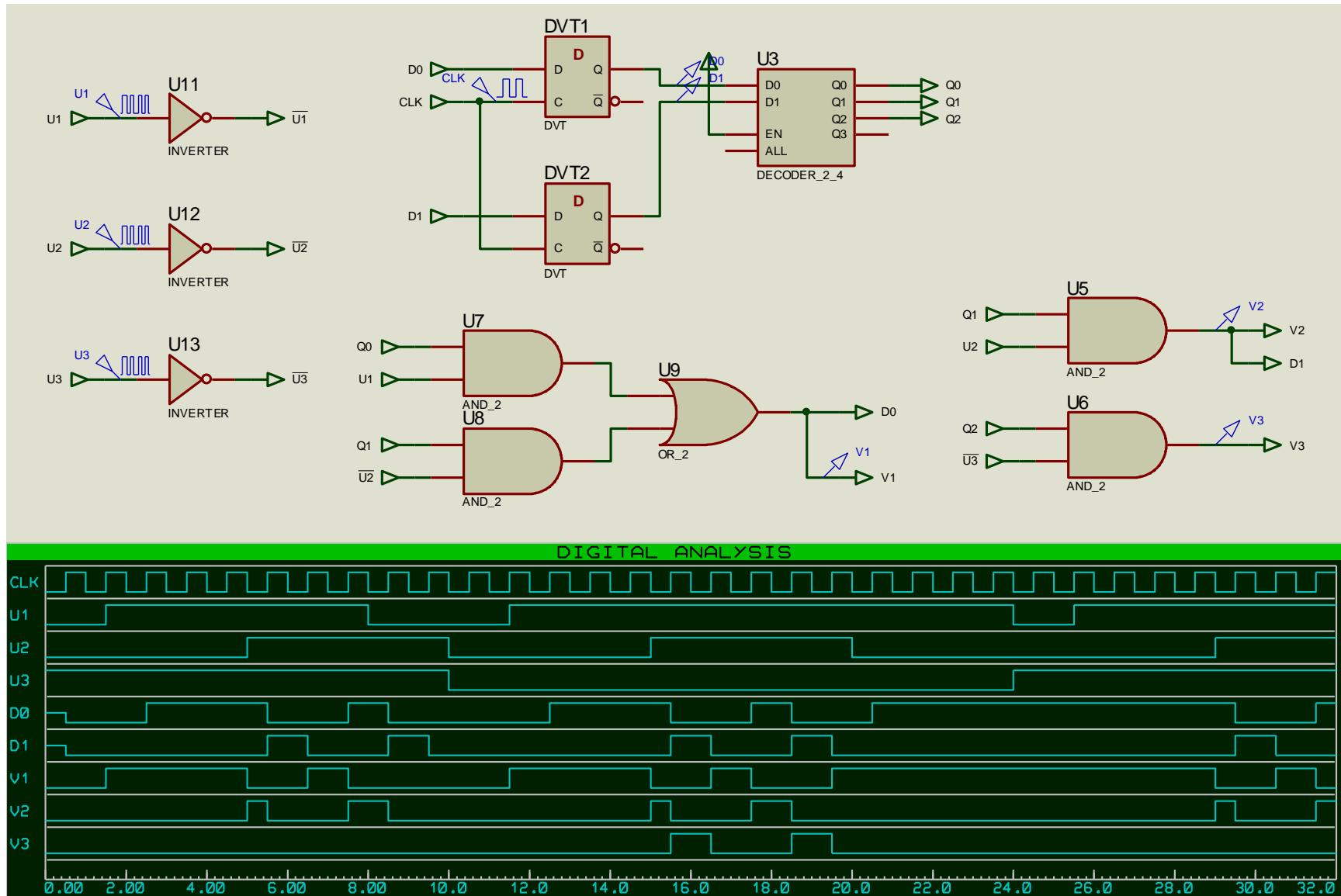
	D1	D0
Q0	0	0
Q1	0	1
Q2	1	0



$$D_0(t+1) = Q_1(t) = Q_1! U_2 \cup Q_0 U_1;$$

$$D_1(t+1) = Q_2(t) = Q_1 U_2;$$

Автомат Мили



Автоматы Мили и Мура с синхронными входами и выходами

Схема автомата Мили с синхронным выходом

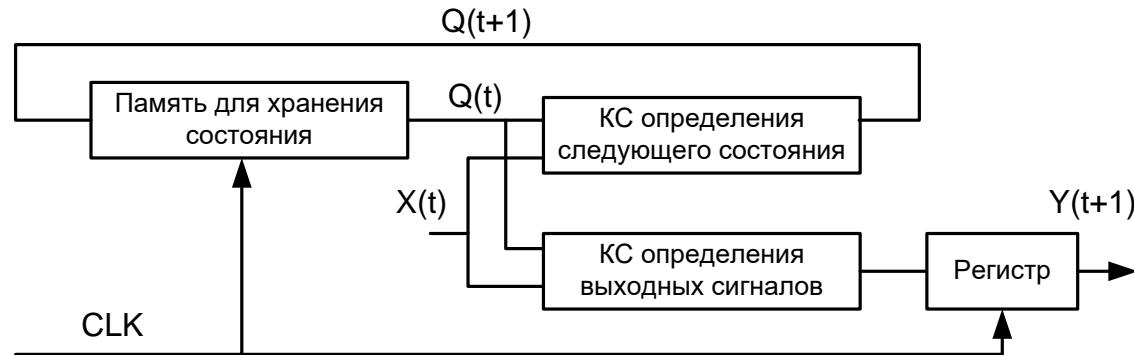
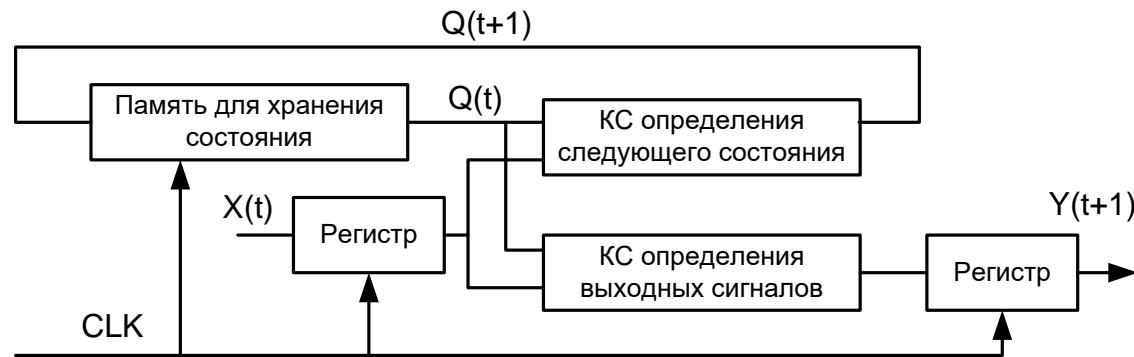


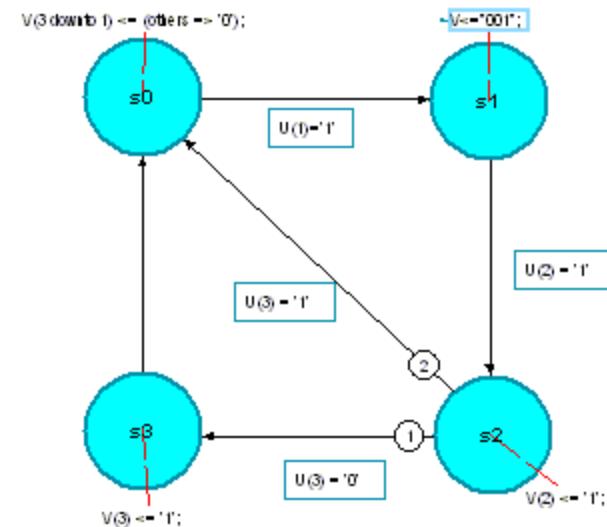
Схема автомата Мили с синхронными входами/выходами



Описание автомата Мура на языке VHDL

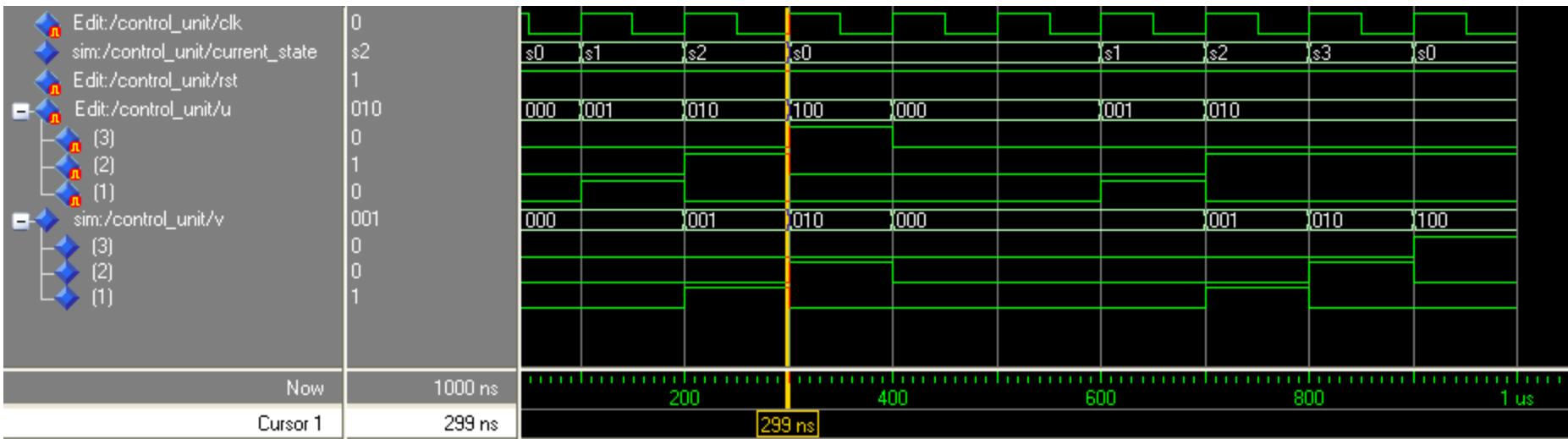
(вариант с синхронными входами и выходами)

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;
USE ieee.std_logic_arith.all;
ENTITY control_unit IS
  PORT(    U : IN  std_logic_vector ( 3 DOWNTO 1 );
            clk : IN    std_logic;
            rst : IN    std_logic;
            V  : OUT   std_logic_vector ( 3 DOWNTO 1 ) );
END control_unit;
ARCHITECTURE moore OF control_unit IS
  TYPE STATE_TYPE IS (s0, s1,s2,s3);
  SIGNAL current_state : STATE_TYPE;
BEGIN
  clocked_proc : PROCESS (clk, rst)
  BEGIN
    IF (rst = '0') THEN
      current_state <= s0;
    ELSIF (clk'EVENT AND clk = '1') THEN
      -- тут описание переходов
    END IF;
  END PROCESS;
END;
```



```
CASE current_state IS
    WHEN s0 =>
        V(3 downto 1) <= (others => '0');
        IF (U(1)='1') THEN current_state <= s1;
        ELSE current_state <= s0; END IF;
    WHEN s1 =>
        V<= "001";
        IF (U(2) = '1') THEN current_state <= s2;
        ELSE current_state <= s1; END IF;
    WHEN s2 =>
        V <= "010";
        IF (U(3) = '0') THEN current_state <= s3;
        ELSE current_state <= s0; END IF;
    WHEN s3 =>
        V <= "100";
        current_state <= s0;
    WHEN OTHERS =>
        current_state <= s0;
END CASE;
END IF;
END PROCESS clocked_proc;
END moore;
```

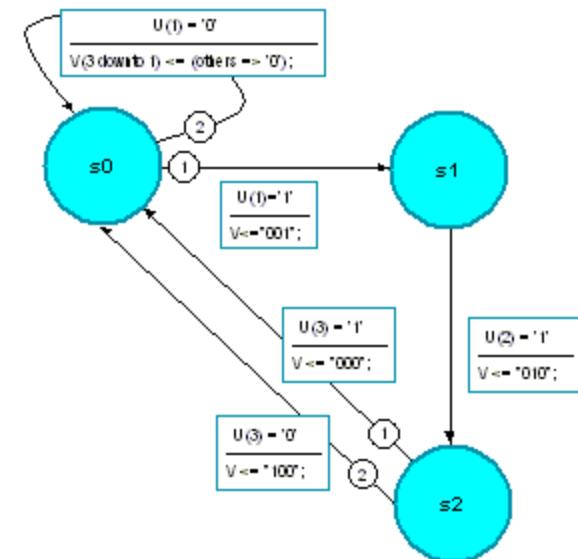
Тест автомат Мура в ModelSim 6



Описание автомата Мили на языке VHDL

(вариант с синхронными выходами)

```
LIBRARY ieee;  
USE ieee.std_logic_1164.all;  
USE ieee.std_logic_arith.all;  
ENTITY control_unit IS  
    PORT(    U : IN  std_logic_vector ( 3 DOWNTO 1 );  
            clk : IN    std_logic;  
            rst : IN    std_logic;  
            V : OUT   std_logic_vector ( 3 DOWNTO 1 ) );  
END control_unit;  
ARCHITECTURE mielie OF control_unit IS  
    TYPE STATE_TYPE IS (s0, s1,s2);  
    SIGNAL current_state : STATE_TYPE;  
BEGIN  
    clocked_proc : PROCESS (clk, rst)  
    BEGIN  
        IF (rst = '0') THEN  
            current_state <= s0;  
        ELSIF (clk'EVENT AND clk = '1') THEN
```

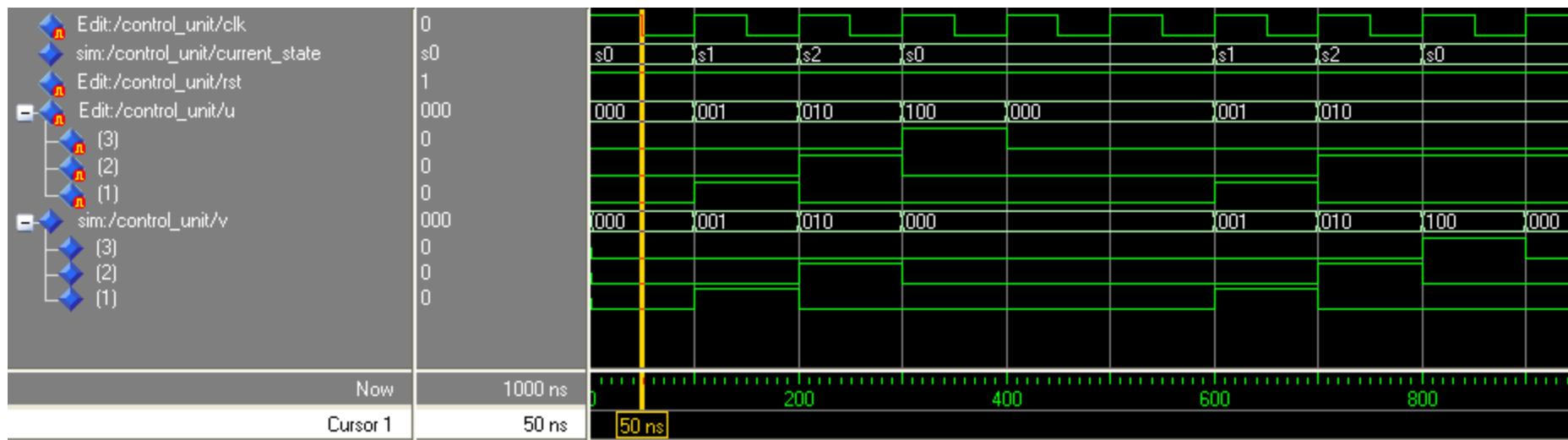


Описание автомата Мили на языке VHDL (окончание)

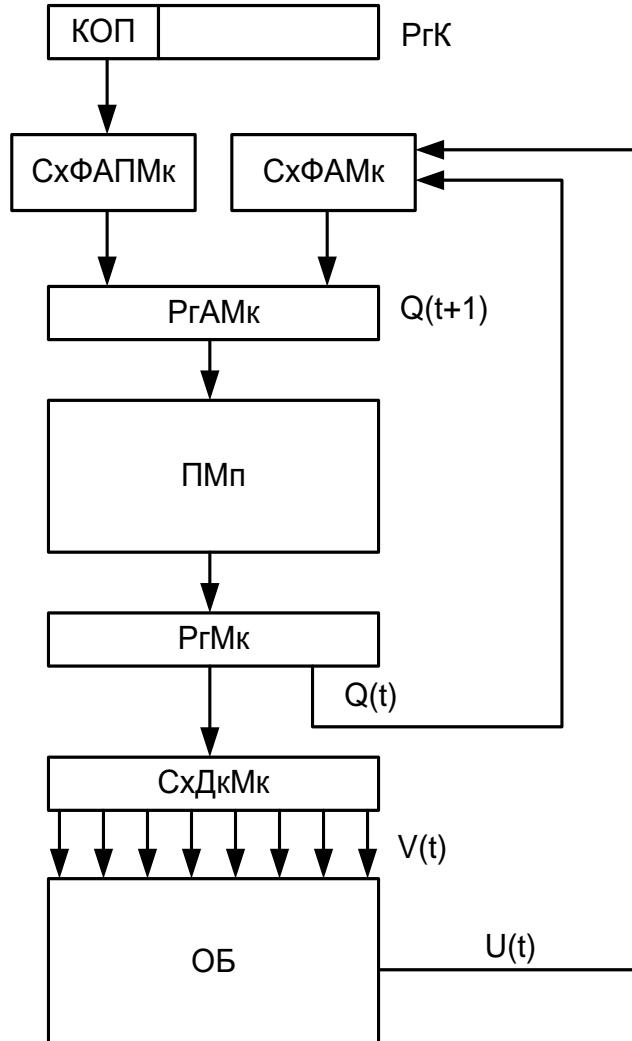
```
CASE current_state IS
    WHEN s0 =>
        IF (U(1)='1') THEN
            V<="001";
            current_state <= s1;
        ELSIF (U(1) = '0') THEN
            V(3 downto 1) <= (others => '0');
            current_state <= s0;
        ELSE
            current_state <= s0;
        END IF;
    WHEN s1 =>
        IF (U(2) = '1') THEN
            V <= "010";
            current_state <= s2;
        ELSE
            current_state <= s1;
        END IF;
    WHEN s2 =>
        IF (U(3) = '1') THEN
            V <= "000";
            current_state <= s0;
        ELSIF (U(3) = '0') THEN
            V <= "100";
            current_state <= s0;
        ELSE
            current_state <= s2;
        END IF;
    WHEN OTHERS =>
        current_state <= s0;
END CASE;
END IF;
END PROCESS clocked_proc;

END mielie;
```

Тест автомат Мили в ModelSim 6



Управляющие устройства с программируемой логикой



PrK – регистр команды;
КОП – код операции;
СхФАПМк – схема формирования адреса первой микрокоманды;
СхФАМк – схема формирования адреса микрокоманды;
РгАМк – регистр адреса микрокоманды;
ПМп – память микропрограмм;
РгМк – регистр микрокоманды;
СхДкМк – схема декодирования микрокоманд;
ОБ – операционный блок.

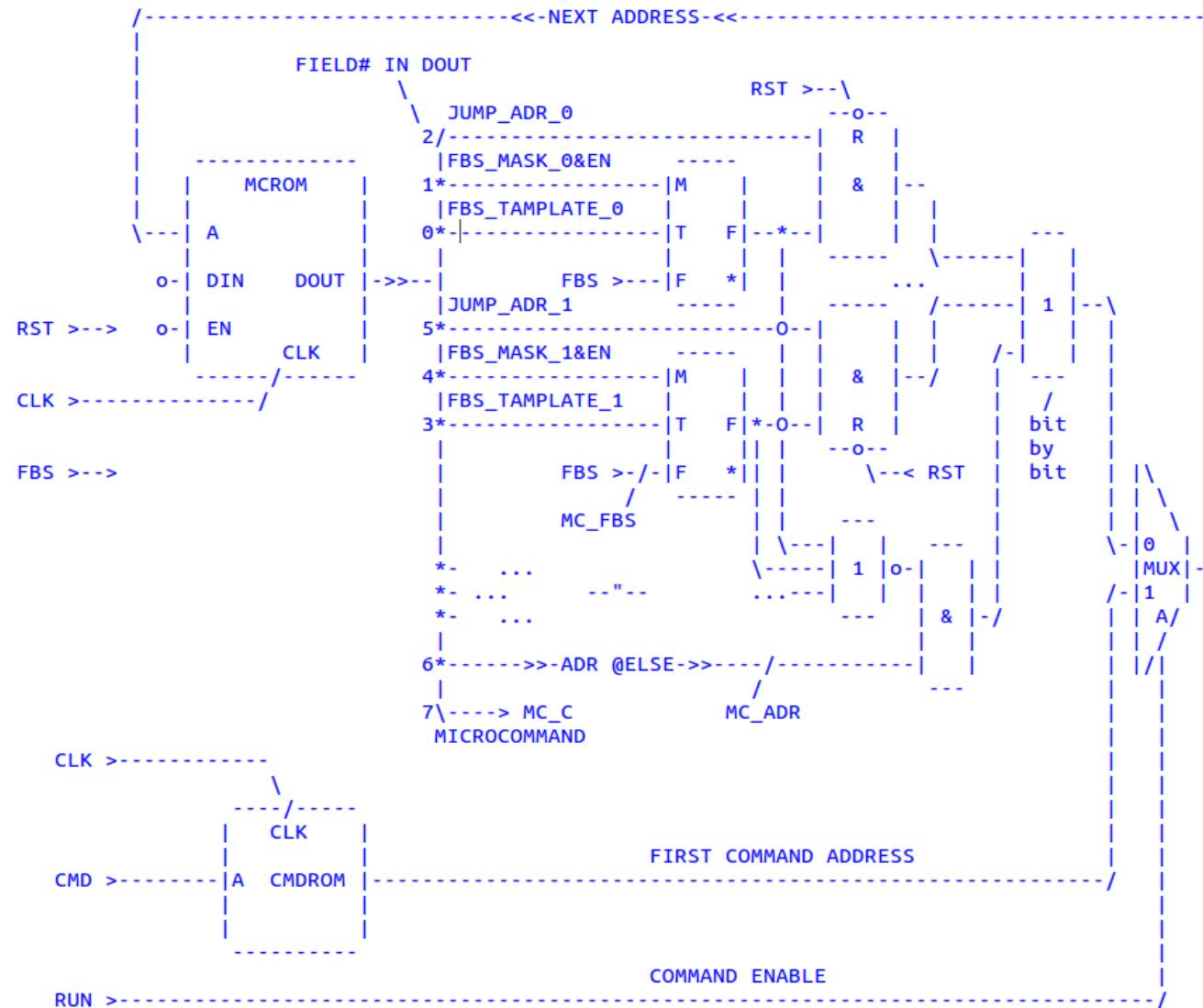
В зависимости от типа ПМп:

- Статическое микропрограммирование (ROM);
- Динамическое программирование (RAM).

По способу адресации микрокоманд:

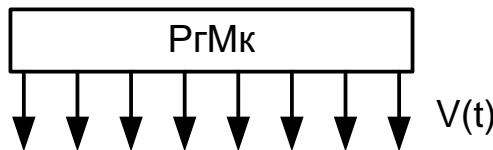
- Принудительная адресация;
- Естественная адресация.

Управляющие устройства с программируемой логикой

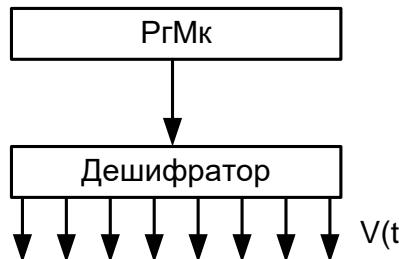


Способы кодирования микрокоманд

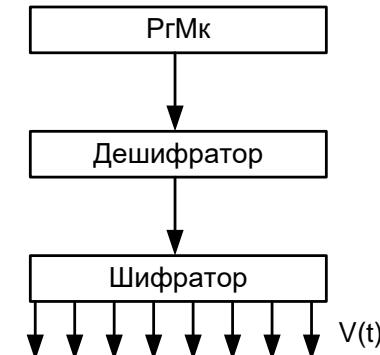
Минимальное
кодирование
(горизонтальное).



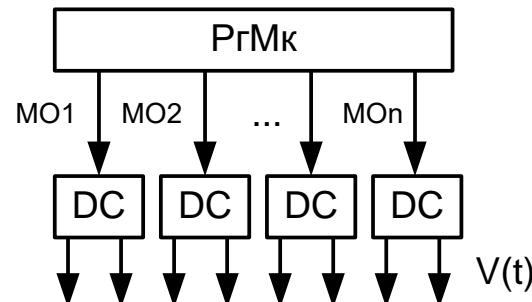
Максимальное
кодирование
(вертикальное).



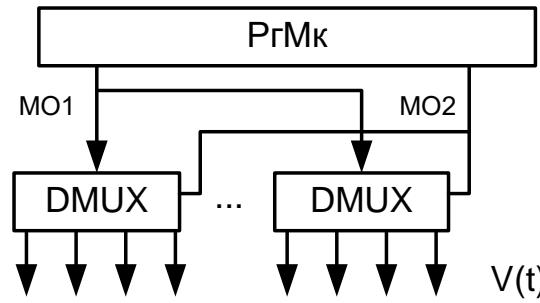
Максимальное
кодирование с
шифратором.



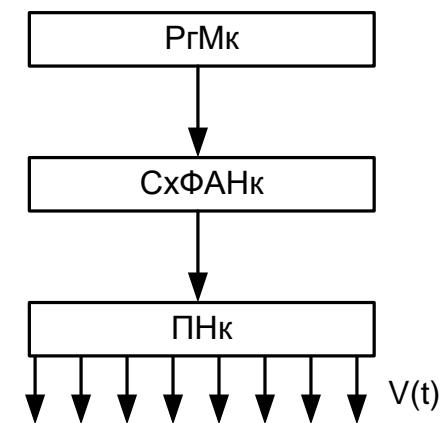
Вертикально-
горизонтальное
кодирование



Горизонтально-
вертикальное
кодирование.



Кодирование с помощью
памяти нанокоманд



Цикл микрокоманды

Цикл исполнения микрокоманды;

- Формирование адреса микрокоманды.
- Выборка микрокоманды.
- Исполнение микрокоманды.

При последовательном исполнении цикла:

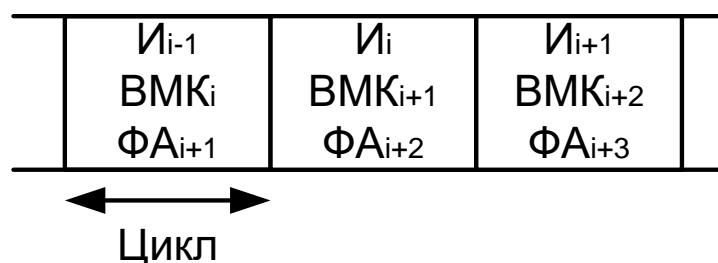
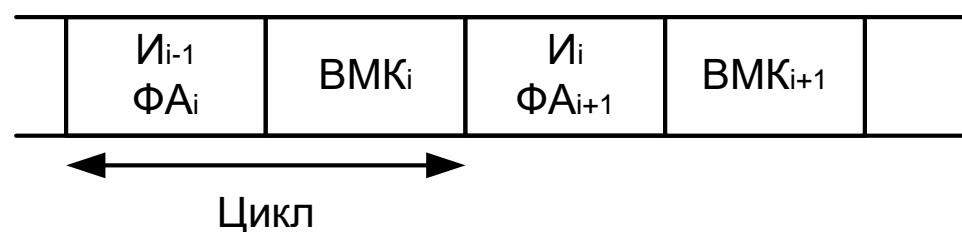
$$T_{MK} = T_{FA} + T_{VMK} + T_i.$$

При совмещении исполнения и формирования адреса следующей микрокоманды:

$$T_{MK} = \text{MAX}(T_{FA} + T_{VMK}, T_i).$$

При совмещении всех действий:

$$T_{MK} = \text{MAX}(T_{FA}, T_{VMK}, T_i).$$



III. Процессорные устройства

- Классификация процессорных устройств.
- Обобщенная структура универсального процессорного устройства.
- Архитектура конвейерного суперскалярного процессора P6.
- Современные суперскалярные микропроцессоры

Литература

- Цилькер Б.Я., Орлов С.А. Организация ЭВМ и систем: Учебник для вузов.– СПб.: Питер, 2004.– 668 с.
- IA-32 Intel Architecture Software Developer's Manual, September 2005, (www.intel.com)
- Intel 64 and IA-32 Architecture Optimization Reference Manual, Order Number 248966-016, November 2007 (www.intel.com)
- Darrell Boggs, Aravindh Baktha, Jason Hawkins, et al, The Microarchitecture of the Intel Pentium 4 Processor on 90nm Technology, Intel Technology Journal, Volume 8, Issue 1, 2004 (<http://developer.intel.com/technology/itj/index.htm>)
- Glenn Hinton, Dave Sager, Mike Upton, Darrell Boggs, et al, The Microarchitecture of the Pentium 4 Processor, Intel Technology Journal Q1, 2001
- P6 Family of Processors Hardware Developer's Manual, Order No: 244001-001, September 1998
- Шагурин И.И., Бердышев Е.М. Процессоры семейства Intel P6. Архитектура, программирование, интерфейс. – М.: Горячая линия – Телеком, 2000.– 248 с.
- AMD64 Architecture Programmer's Manual, Volume 1, Volume 2, Volume 3, Advanced Micro Devices, September 2007 (www.amd.com)
- AMD Athlon Processor x86 Code Optimization Guide, Publication No. 22007, February 2002 (www.amd.com)
- Software Optimization Guide for AMD Family 10h Processors, Publication 40546, December 2007 (www.amd.com)
- Бессонов О. Обзор микроархитектур современных десктопных процессоров, Июль 2006 (www.ixbt.com)
- J. M. Tendler, J. S. Dodson, J. S. Fields, Jr., H. Le and B. Sinharoy, POWER4 system microarchitecture, IBM J. RES. & DEV. VOL. 46 NO. 1 JANUARY 2002, pp. 5-25
- An Overview of UltraSPARC III Cu. UltraSPARC III Moves to Copper Technology, A White Paper, Sun Microsystems, Version 1.1, September 2003 (www.sun.com)
- K.R. Kishore, Vidya Rajagopalan, Georgi Beloev, and Radhika Thekkath, Architectural Strengths of the MIPS32 74K Core Family, MIPS Technologies, Inc., December 2007
- MIPS32 74Kc Processor Core Datasheet, MIPS Technologies, Inc., December 14, 2007

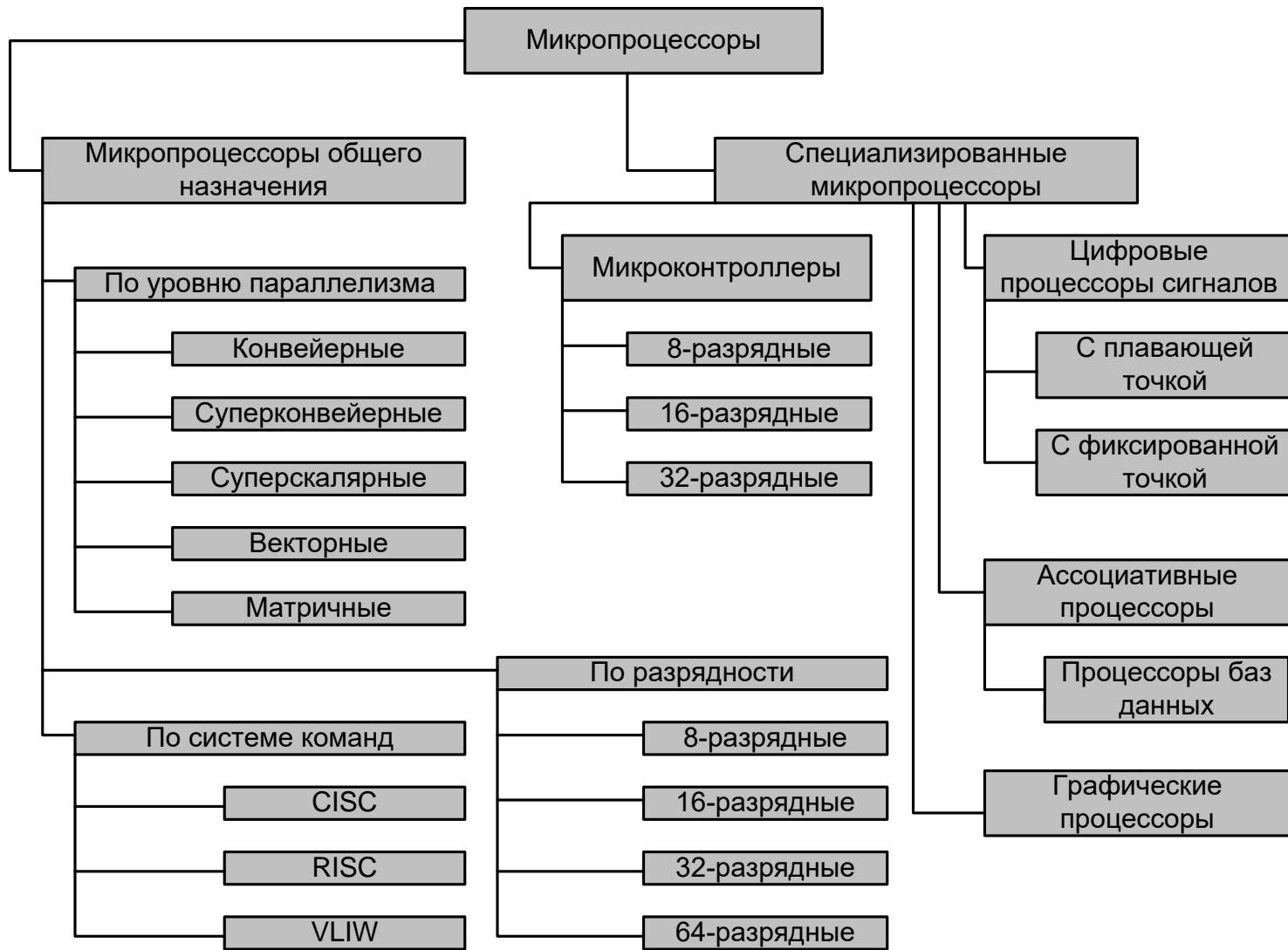
Процессором или процессорным ядром называется устройство ЭВМ, непосредственно осуществляющее процесс переработки информации и управление им в соответствии с заданным алгоритмом, который, как правило, представлен программой.

ЭВМ может содержать несколько процессоров. Процессор, управляющий вычислительным процессом, называется центральным.

Микропроцессором называется функционально законченное устройство, представляющее собой вариант процессора или нескольких процессорных ядер современной ЭВМ и реализованное в виде одной или нескольких СБИС.

Микропроцессорный комплект представляет собой совокупность микропроцессора и специализированных ИС, совместимых по временными, электрическим и конструктивным параметрам, совместное использование которых позволяет реализовать основные функции ЭВМ.

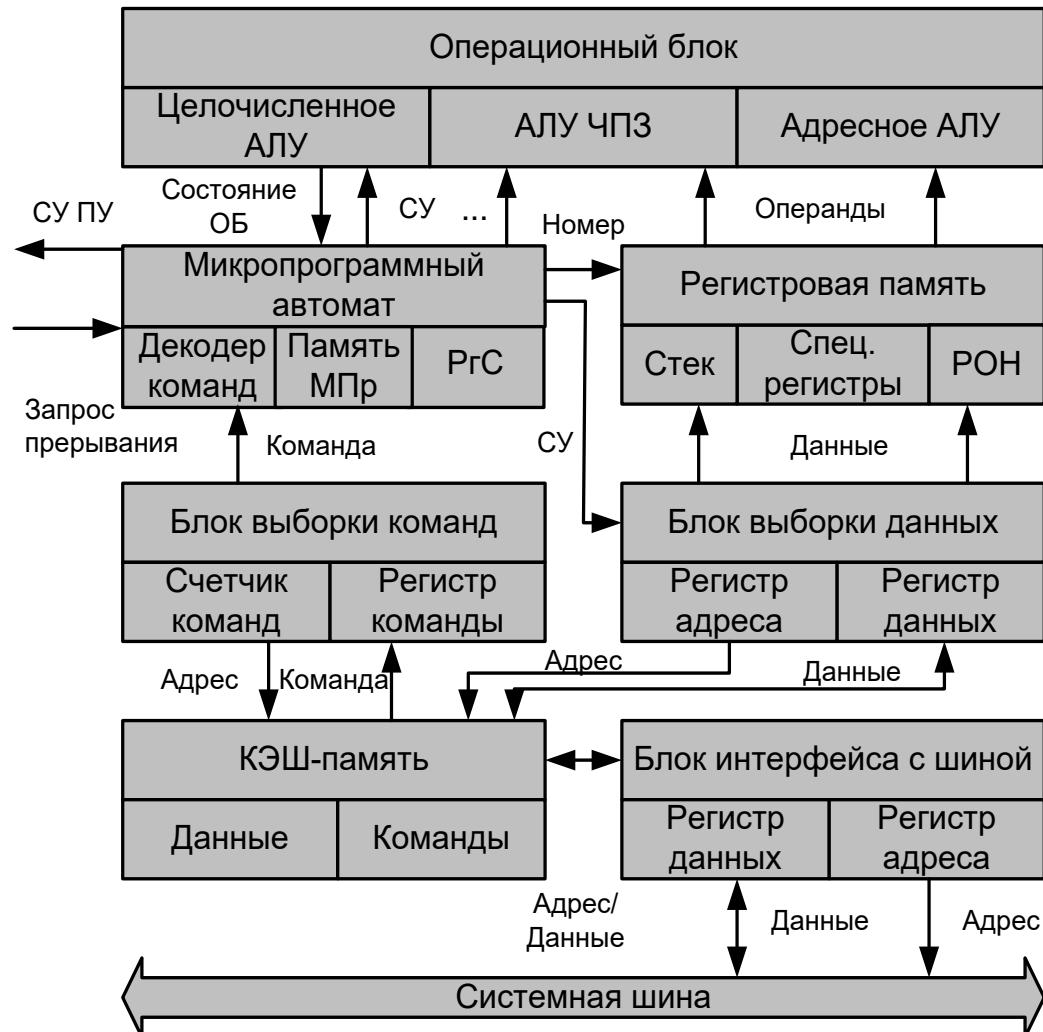
Классификация процессорных устройств



Обобщенная структура универсального процессорного устройства

Архитектурные особенности:

- Конвейерное исполнение команд.
- Внутренняя КЭШ-память.
- Целочисленное АЛУ.
- Устройство выполнения операций над числами с плавающей запятой.
- Обработка прерываний от ПУ.
- Поддержка мультипроцессорной обработки.



Сравнение способов организации параллельных вычислений



Конфликты в конвейере (риски)

1. Структурный риск

Команды одновременно обращаются к одному и тому же ресурсу (например, к ОП).

2. Риск по данным

Команды имеют зависимость по данным.

$O(i)$ – множество ячеек, изменяемых командой i ;

$I(j)$ – множество ячеек, читаемых командой j .

А) Чтение после записи (ЧПЗ).

Б) Запись после чтения (ЗПЧ).



$$O(i) \cap I(j) \neq \emptyset$$

$$I(i) \cap O(j) \neq \emptyset$$

В) Запись после записи (ЗПЗ).



$$O(i) \cap O(j) \neq \emptyset$$

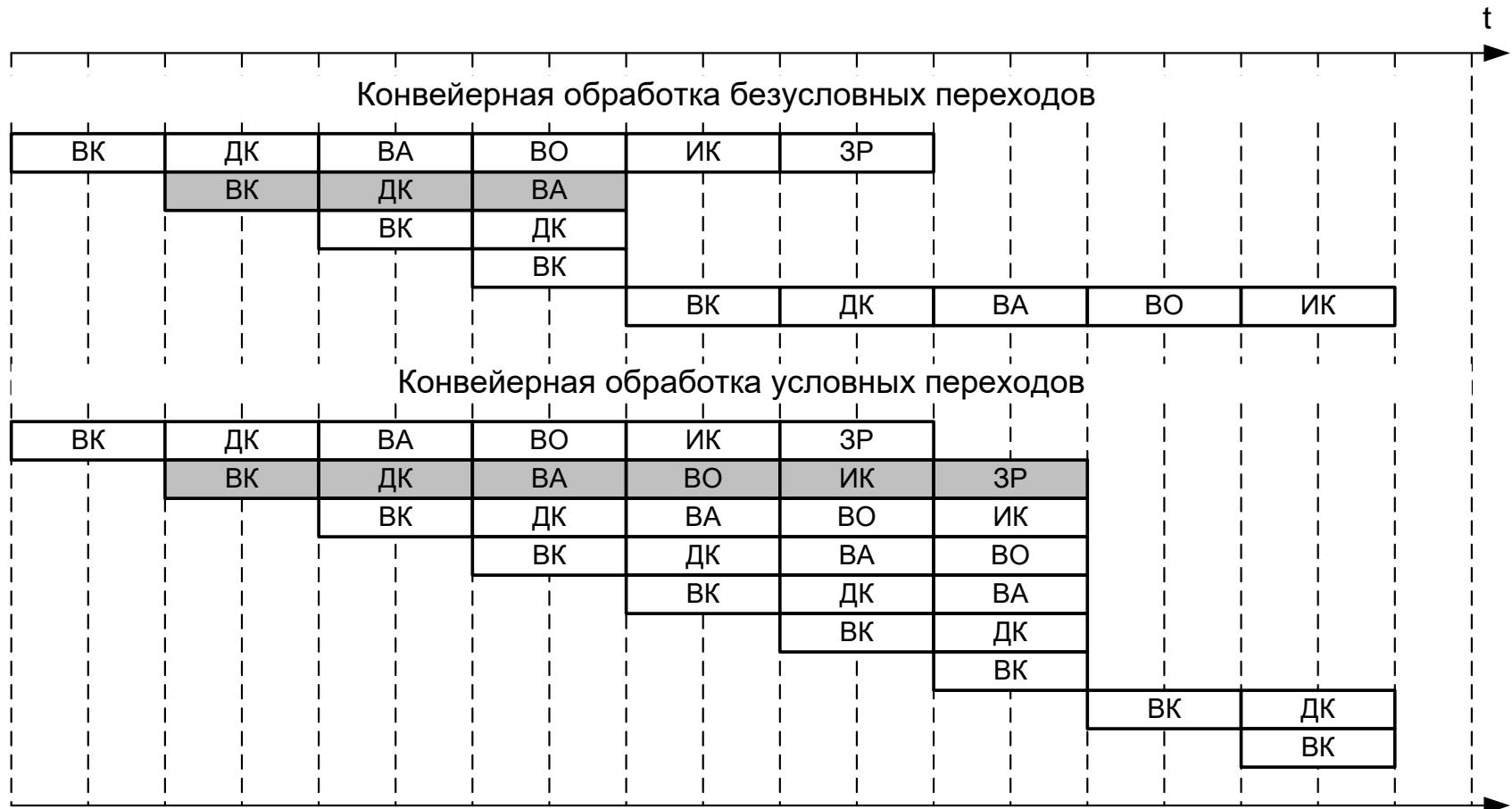
3. Риск по управлению.

Из-за наличия команд перехода (10-20% потока команд) возможна неоднозначность при выборе очередной инструкции.

Потери в лучшем случае: сброс всех поступивших команд за время декодирования команды ветвления.

Потери в худшем случае: сброс всех поступивших команд за время декодирования, выборки operandов и исполнения команды ветвления.

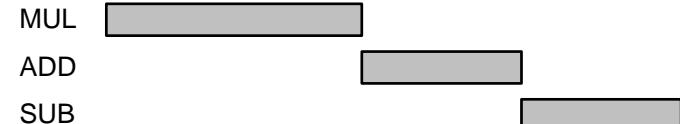
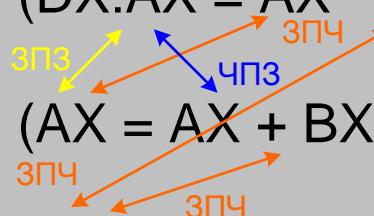
Временные потери при обработке команд переходов



Способы устранения конфликтов по данным, находящихся в регистрах

Пример 1:

MUL BX	; (DX:AX = AX * BX)
ADD AX,BX	; (AX = AX + BX)
SUB BX,2	; (BX = BX - 2)

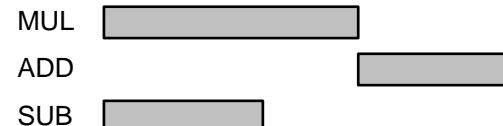


Правило:

Каждый новый результат записывается в новый регистр замещения.

MUL BX	; (DX':AX' = AX * BX)
ADD AX,BX	; (AX'' = AX' + BX)
SUB BX,2	; (BX' = BX - 2)

ЧПЗ



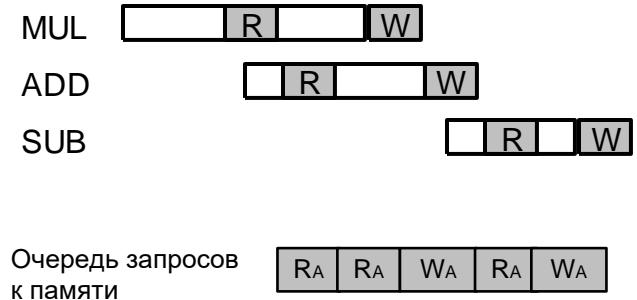
Конфликт типа ЧПЗ по данным, находящимся в регистрах, может быть устранен с помощью бита достоверности

Способы устранения конфликтов по данным, находящихся в памяти

Пример 2:

MUL A	; (DX:AX = AX * A)
ADD A,BX	; (A = A + BX)
SUB A,2	; (A = A - 2)

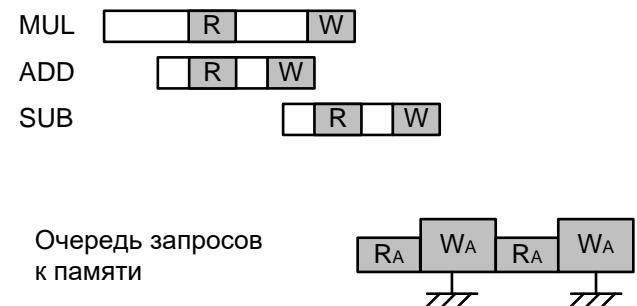
зПЧ
зПЗ
ЧПЗ



Правило:

При обнаружении конфликтов по данным, находящимся в ОП, запросы на запись результатов в память выполняются упорядочено.

MUL A	; (DX:AX = AX * A)
ADD A,BX	; (A = A + BX)
SUB A,2	; (A = A - 2)



Способы устранения конфликтов по управлению

- Дублирование ступеней конвейера для обработки обеих ветвей
- Оптимизация кода на этапе компиляции с целью увеличения полезной нагрузки на дублированные ступени конвейера.
- Предсказание переходов.

Способы предсказания переходов

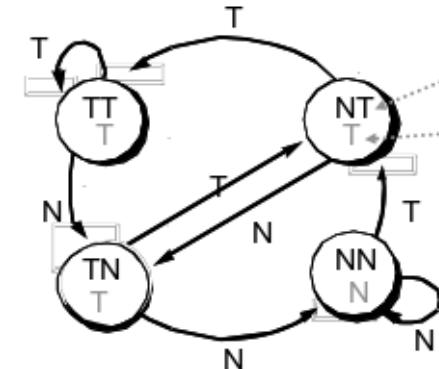
Точность предсказания: отношение числа правильно предсказанных переходов к их общему количеству.

Эффективность алгоритмов предсказания зависит от использования статистических данных, накопленных:

- заранее при компиляции и тестовых прогонах (статическое предсказание переходов);
- полученных в процессе исполнения программы (динамическое предсказание переходов).
- На основе статического и динамического подходов.

Стратегии статического предсказания переходов

- Переход происходит всегда (60-70%).
- Переход не происходит никогда (50%).
- Переход выполняется по результатам профилирования (75%).
- Переход определяется по коду операции (75%).
- Переход выполняется исходя из направления (85%).
- При первом выполнении переход имеет место всегда (90%).



Стратегии динамического предсказания переходов

- Одноуровневое предсказание: использует Шаблонную Таблицы Истории (Pattern History Table) или Branch Target Buffer (Буфер меток перехода).

Выборка информации может происходить: по адресу команды перехода; по истории всех команд перехода; по истории исполнения только предсказываемой команды перехода.

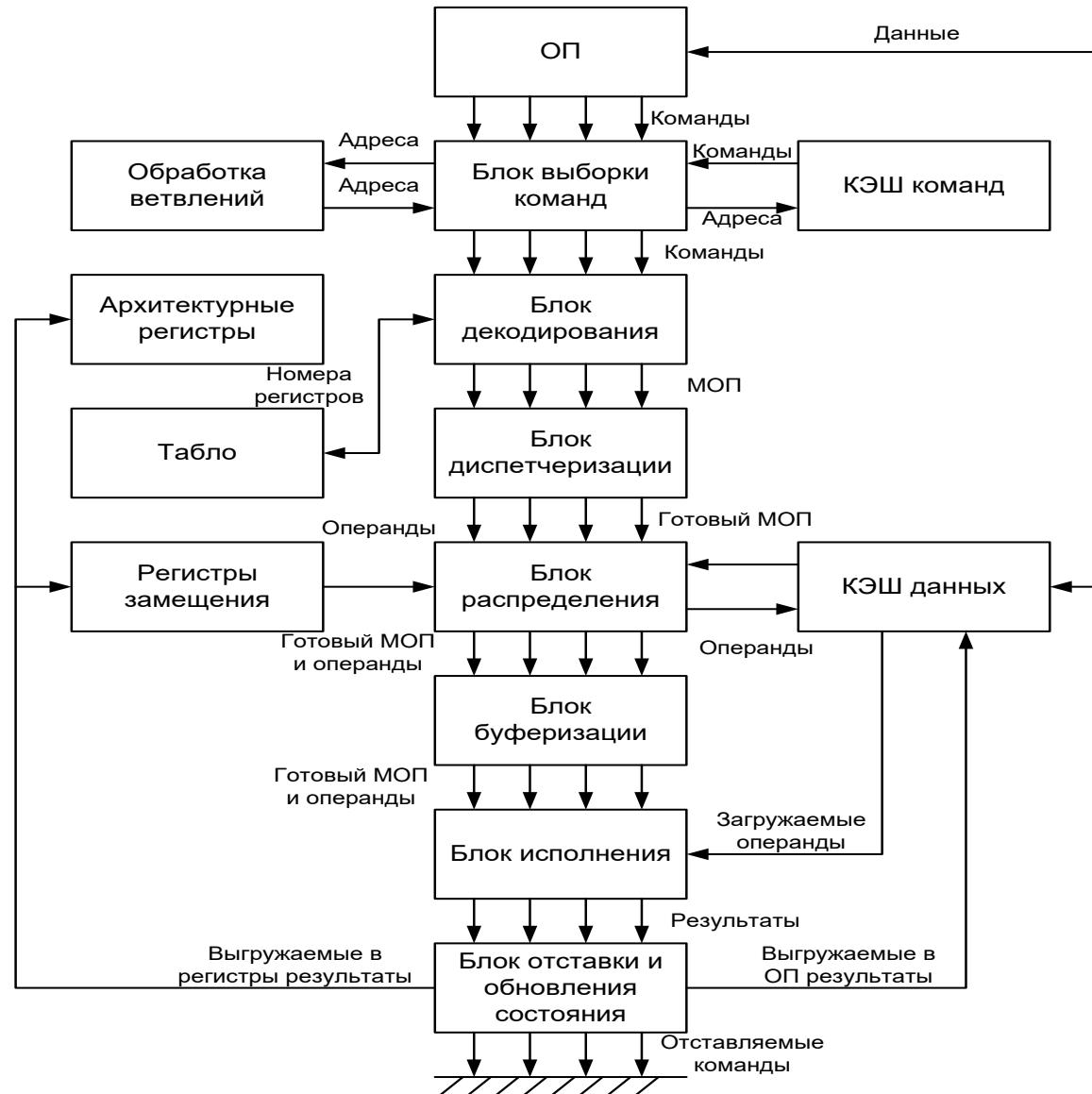
Алгоритм предсказания зависит от размера строк РНТ. При хранении одного бита переход предсказывается в соответствии с предыдущим итогом выполнения команды (точность ~78%).

При хранении двух бит учитывается переход для двух последних исполнений команды (точность ~90%).

- Двухуровневое предсказание (динамическое и статическое).

- Гибридное предсказание (РНТ и ВТВ)

Обобщенная схема суперскалярного суперконвейерного процессора



Структура процессора Р6

История процессоров с архитектурой IA32

Модель	Годы выпуска	Функциональность
8086	1978	16 разрядный микропроцессор. Сегментация. 20-разрядная шина адреса (до 1 Мб).
80286	1982	Защищенный режим с использованием дескрипторного регистра (четыре уровня привилегий, поддержка сегментов только для чтения и только для выполнения, ограничение прав доступа). Поддержка виртуальной памяти. 24-разрядная шина адреса (16 Мб)
80386	1985	32 разрядный микропроцессор. Поддержка 16 разрядного кода. Режим Virtual-8086. Сегментная и непрерывная модель памяти. Страницчная организация виртуальной памяти (4 Кб страницы). 32-разрядная шина адреса (4 Гб). Совмещение исполнения команд с обращением к памяти.
80486	1989	Конвейер команд (5 стадий). 8 Кб кэш первого уровня. Интегрированный FPU. Режим энергосбережения.

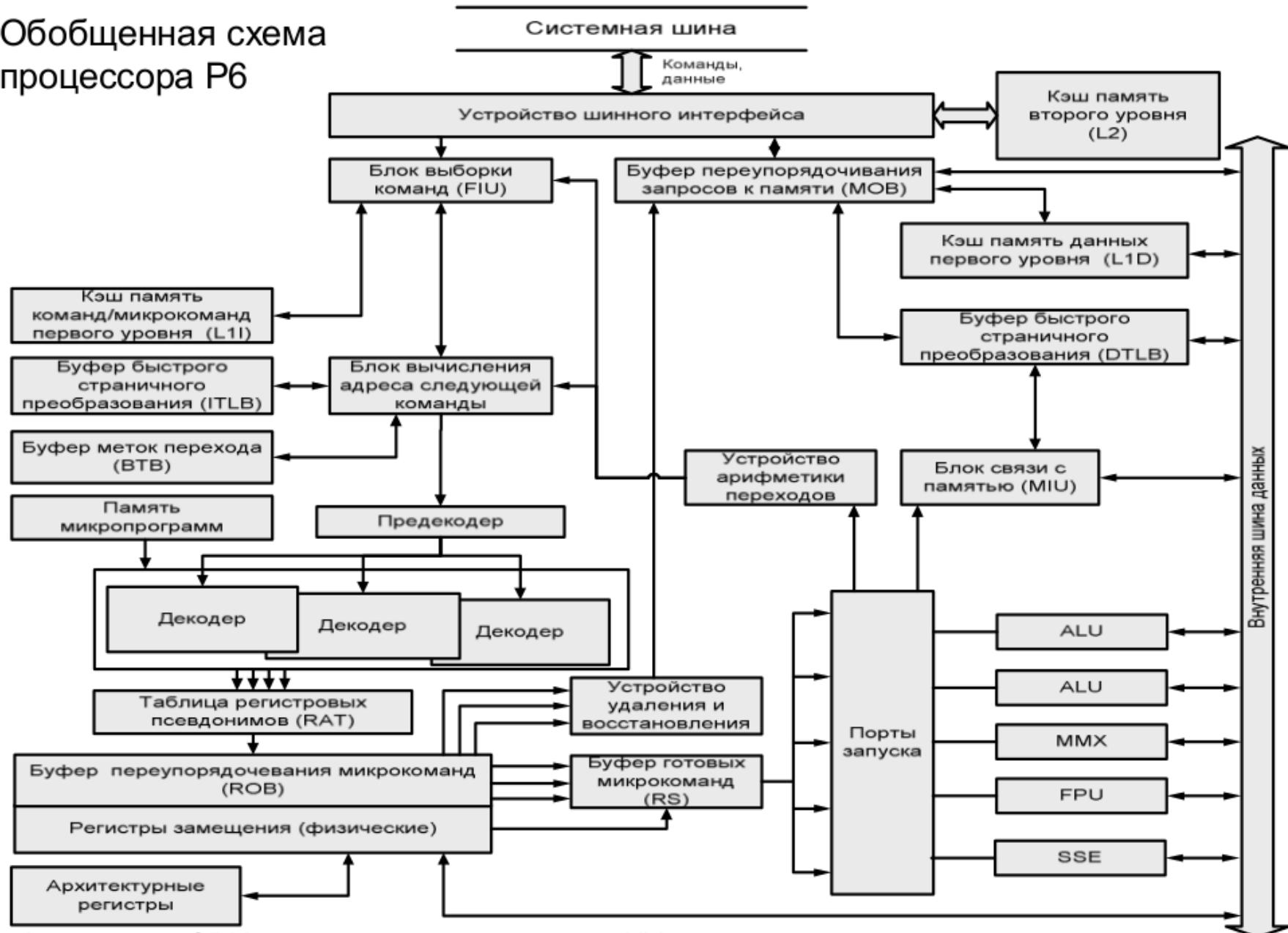
История процессоров с архитектурой IA32 (продолжение)

Модель	Годы выпуска	Функциональность
Pentium	1993	Двухконвейерная архитектура (возможность суперскалярной обработки). Раздельные кэш первого уровня команд и данных (по 8 Кб, протокол MESI Write Back). Предсказание ветвления. 4Кб и 4Мб страницы. 64 разрядная внешняя шина данных. Поддержка пакетного режима. Поддержка построения многопроцессорных ВС. MMX SIMD обработка (64 разряда).
Семейство P6 (Pentium Pro, Celeron, Pentium II, Pentium III)	1995	Суперскалярная обработка на основе техники переупорядочивания (до трех операций за такт). Динамическое выполнение команд (анализ зависимостей по данным, неупорядоченное и спекулятивное выполнение, предсказание ветвления). Интегрированный смешанный кэш второго уровня (до 2 Мб). SSE (128 разрядов)
Семейство NetBurst (Pentium 4)	2000	Быстрое выполнение на удвоенной скорости. Гиперконвейерная суперскалярная организация. Глубокая предвыборка. Кэш трасс. SSE2 и SSE3

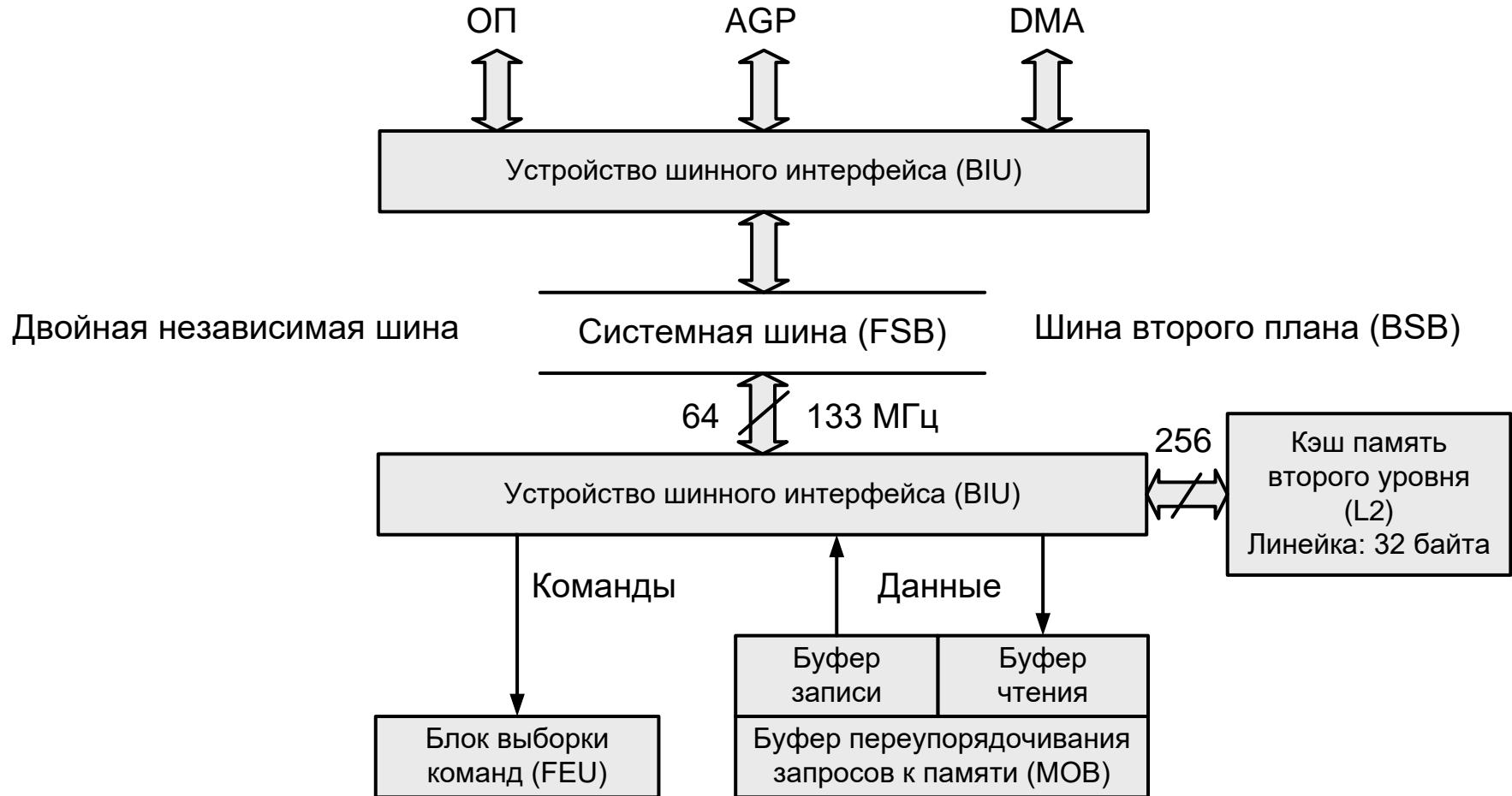
История процессоров с архитектурой IA32 (окончание)

Модель	Годы выпуска	Функциональность
Семейство Pentium M	2003	Низкое энергопотребление, Сбалансированная производительность
Семейство Intel Core (Intel Xeon 5100, 5300, Intel Pentium Dual-Core, Intel Core 2, Intel Core 2 Quad)	2006	Многоядерная архитектура, Intel 64 Architecture, Intelligent Power Capability
Семейство Intel Atom (Intel Atom)	2008	Сверхнизкое энергопотребление, Dual Pipeline In-order Execution, Dynamic Cache Sizing
Семейство Intel Nehalem (Intel Core i7, Intel Xeon 5500, 7500)	2008	Выделенный блок управления энергопотреблением, До 8 ядер. SSE4
Семейство Intel Westmere (Intel Core i3,i5,i7, Intel Xeon 5600)	2010	Усовершенствованный блок управления энергопотреблением, Интегрированное графическое ядро, Интегрированный контроллер памяти DDR3

Обобщенная схема процессора Р6



Устройство шинного интерфейса



Обращение к ОП выполняется через L2. Разделение системной магистрали на две независимые шины снижает нагрузку на системную магистраль до 10% от максимальной.

Кэш память второго уровня (L2)

Тип: Наборно-ассоциативная неблокируемая

Размер: до 2 МБ

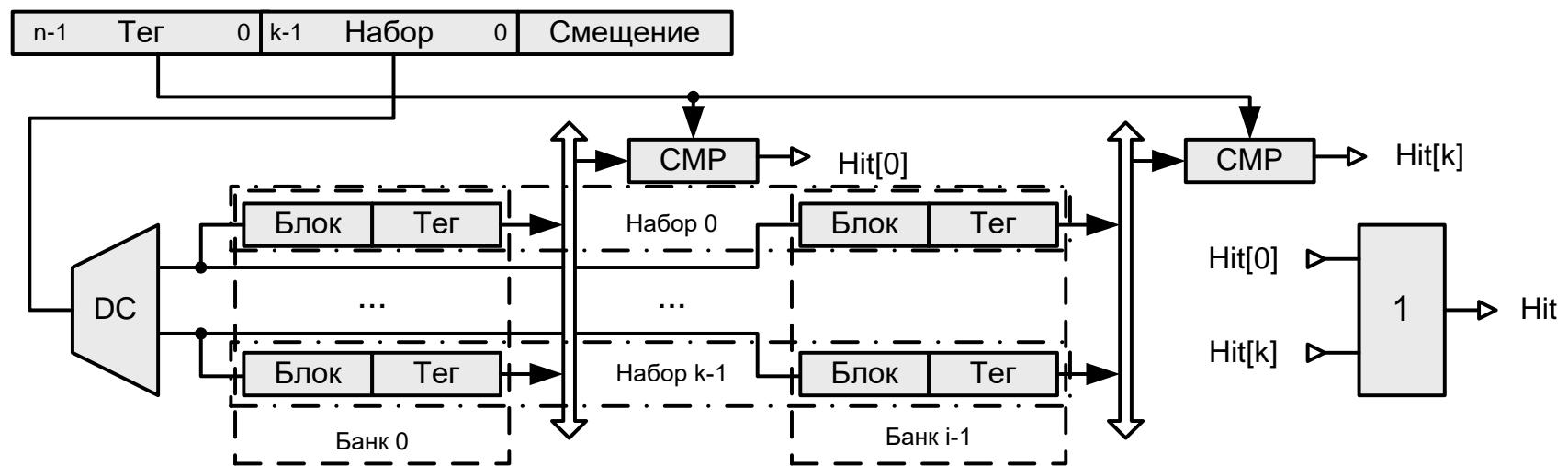
Размер линейки: 32 байта.

Ассоциативность: 4

Политика записи: Write Back

Алгоритм замещения: LRU

Протокол: MESI



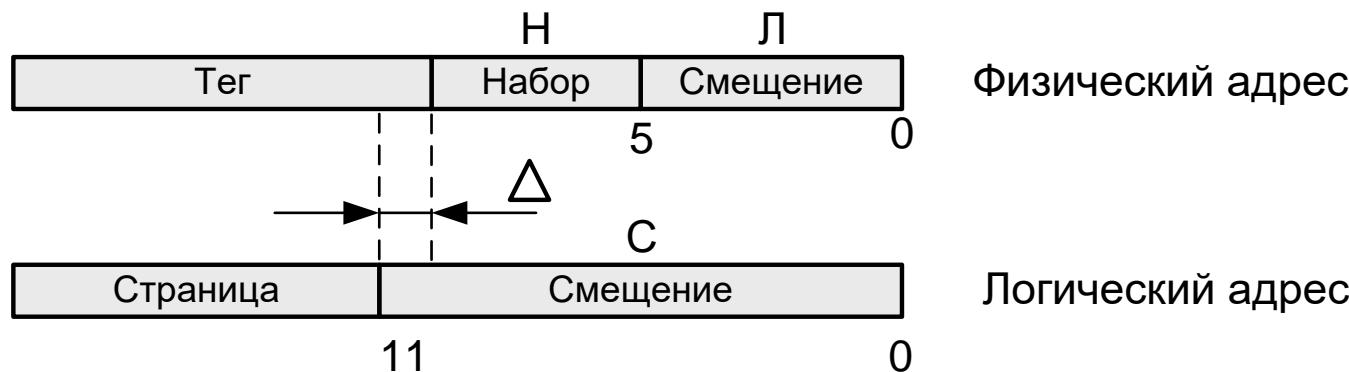
Проблема алиасинга

$$H^*L = (\text{Размер кэш}) / (\text{Ассоциативность})$$

При размещении в кэш данных по адресам с шагом H^*L они размещаются в одном и том же наборе.

Проблема выборки по физическому адресу

Выборка из кэш-памяти осуществляется по физическому адресу.
Для ускорения доступа используют часть логического адреса.

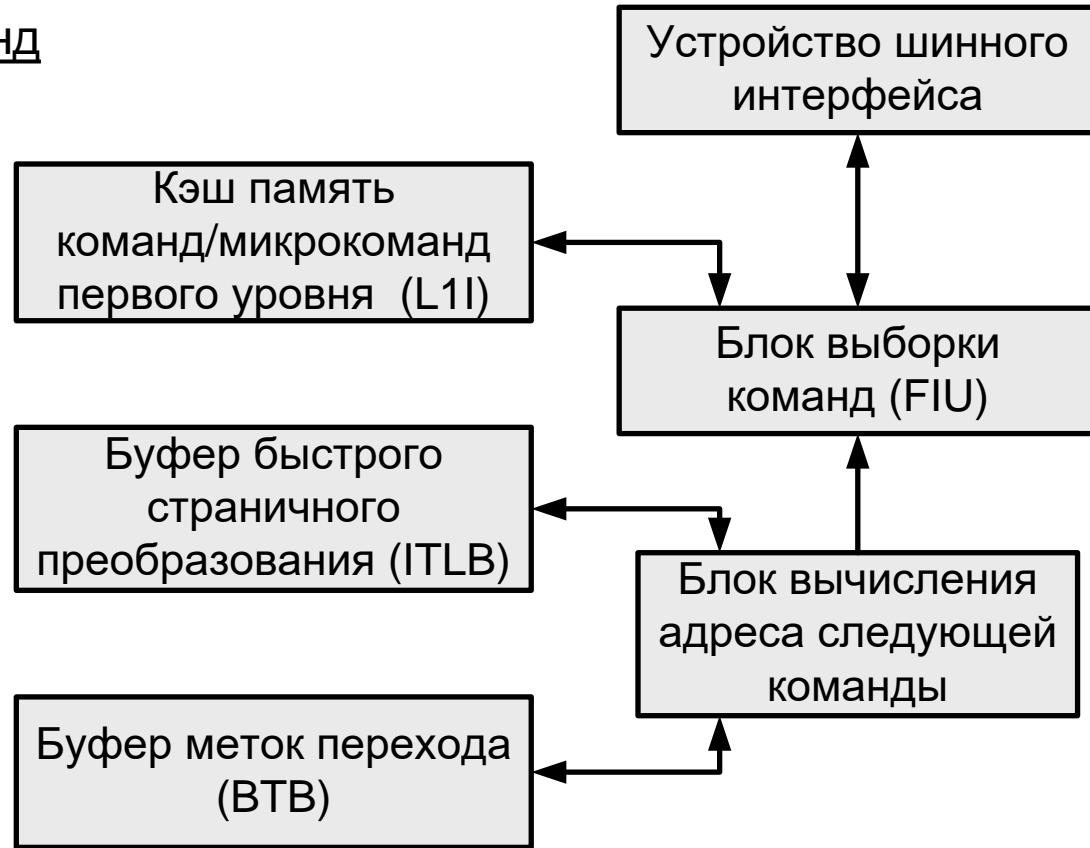


При обнаружении логического адреса в команде с учетом того, что $(C >= H^*L)$ уже известен набор, а возможно и часть тега.

Тогда можно заранее определить наличие кандидатов на выборку из кэш или заранее обнаружить кэш промах.

Блок выборки команд

Блок выборки команд получает физический адрес очередной команды из Блока вычисления адреса следующей команды. По этому адресу сначала происходит обращение в L1I. Если указанного блока команд (линейки) там нет, то запрос передается в BIU.



Блок вычисления адреса следующей команды реализует механизм статического и динамического предсказания с использованием наборно-ассоц. BTB (Branch Target Buffer). BTB в P6 состоит из 512 элементов (4-х ассоциативный).

Для преобразования логического адреса в физический используется ITLB (Instruction Translation Lookaside Buffer) и DTLB (Data Translation Lookaside Buffer).

Структура TLB

Номер лог. страницы	V	R	M	A	Номер физ. страницы

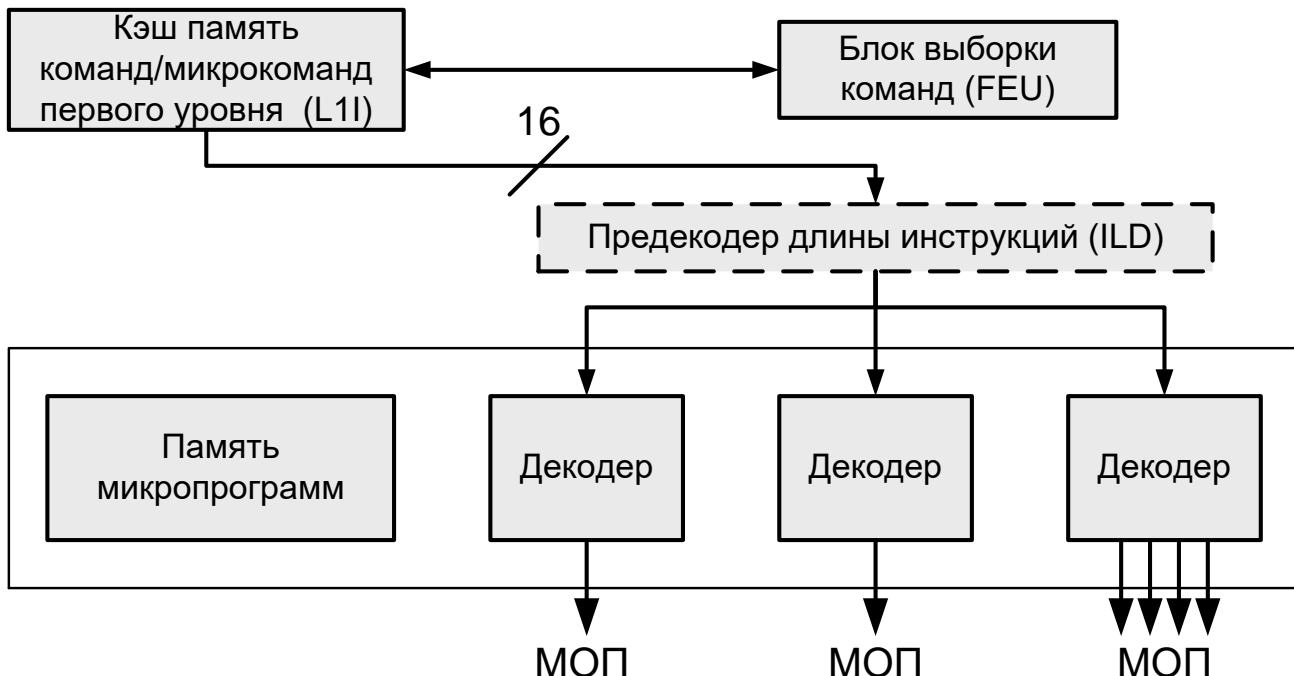
Информация, записываемая в TLB, не подлежит кэшированию.

При доступе к данным или командам по адресу, для преобразования которого информация в TLB отсутствует, необходимо обратиться в оперативную память дважды: сначала за информацией из таблицы страниц, и после преобразования за самими данными или командами.

Этот можно устранить с помощью предвыборки. Команды предвыборки:

Команда	Pentium III (32 байта)	Pentium 4 (128 байт)	Примечание для P6
prefetchNTA	Загрузка только в L1D. В L2 не загружаются	Загрузка в L2. В L1D не загружаются	Загрузка в ближайший кэш для немедленного использования
Prefetch0	Загрузка в L1D и L2	Загрузка только в L2. В L1D не загружается	Загрузка в кэш всех уровней
Prefetch1	Загрузка только в L2. В L1D не загружается	Загрузка в L2. В L1D не загружаются	Загрузка в кэш кроме нулевого (только L2)
Prefetch2	Загрузка только в L2. В L1D не загружается	Загрузка в L2. В L1D не загружаются	Загрузка в кэш кроме нулевого и первого (только L2)

Декодеры



Команды поступают из L1I блоками по 16 байт. В предекодере определяются границы команд и наличие префиксов.

Декодер состоит из трех параллельных каналов: два канала для декодирования простых команд, порождающих одну микрооперацию; один декодер обрабатывает любые инструкции и генерирует по 4 МОП за такт.

Для загрузки operandов и исполнения операций порождаются различные микрооперации. Для вычисления адреса также порождается МОП.

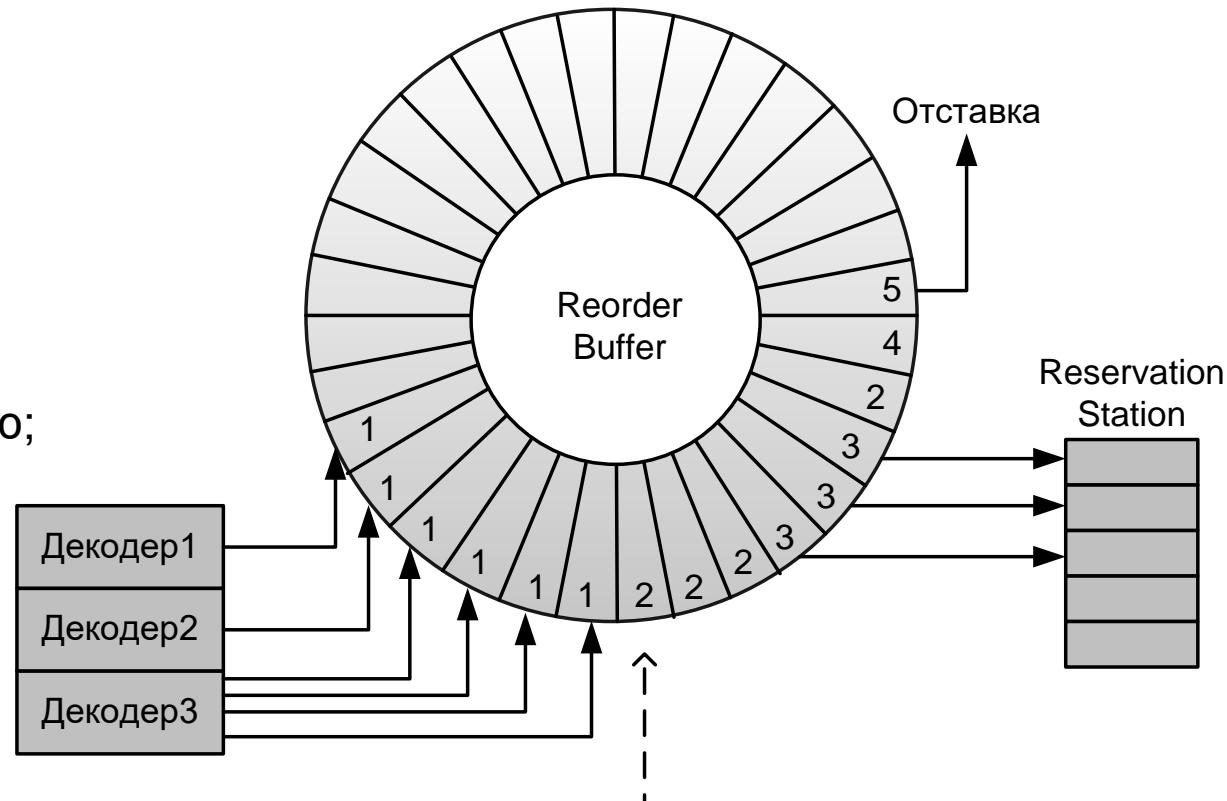
Буфер переупорядочивания микрокоманд

- Микрооперации помещаются в ROB в исходном порядке.
- Исполнение МОП происходит неупорядочено по мере готовности operandов.
- Удаление (отставка) МОП происходит упорядочено из-за: прерываний, исключений, точек останова, неправильно предсказанных переходов.

Микрокоманды в ROB

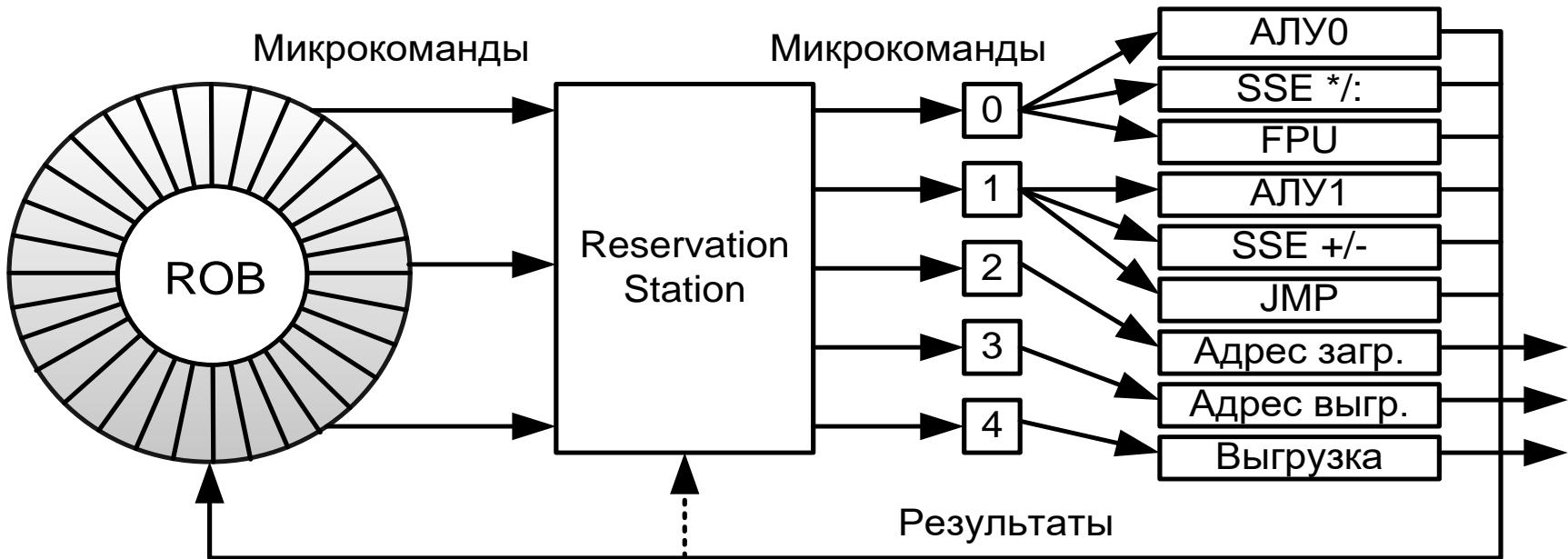
могут находиться в
одном из следующих
состояний:

1. Не готова к исполнению;
2. Готова к исполнению;
3. Исполняется;
4. Исполнена и ожидает отставки;
5. Находится в процессе отставки.



Команда	Поле операции	Поле результата	Поле операнда1	Д1	Поле операнда2	Д2
---------	---------------	-----------------	----------------	----	----------------	----

Порты запуска и исполнительные устройства



В процессорах Р6 пять портов запуска.

В RS в каждом такте могут быть помещены три микрооперации из ROB и пять микроопераций могут быть направлены в порты запуска. Если претендентов на исполнительное устройство несколько, то выбор производится по алгоритму «псевдо-FIFO».

Загрузка и выгрузка

Выгрузка в память (store) происходит в соответствии с порядком отставки (в исходном порядке). Только после отставки возможно незначительное переупорядочивание для оптимизации работы ВИУ.

Загрузка (load) может происходить неупорядоченно в случае отсутствия зависимостей по данным.

Для ускорения выполнения микроопераций загрузки выполняется поиск требуемых данных в микрооперациях выгрузки (forwarding of data from stores to dependent loads).

Однако возможны приложения, в которых процессор не может обнаружить зависимость по данным (I/O operations).

Команды управления загрузкой и выгрузкой

Команда	Назначение	Примечание
lfence	Упорядочивание загрузки	Команда позволяет управлять загрузкой, запрещая переупорядочивать микрооперации загрузки до данной команды с микрооперациями после данной команды.
sfence	Упорядочивание выгрузки	Команда позволяет управлять выгрузкой, запрещая переупорядочивать микрооперации выгрузки до данной команды с микрооперациями после данной команды.
mfence	Упорядочивание загрузки и выгрузки	Команда позволяет управлять загрузкой и выгрузкой, запрещая переупорядочивать микрооперации загрузки и выгрузки до данной команды с микрооперациями после данной команды.

Достоинства Р6

- Суперскалярная обработка.
- Интегрированная кэш-память.
- Сбалансированность фаз конвейера.

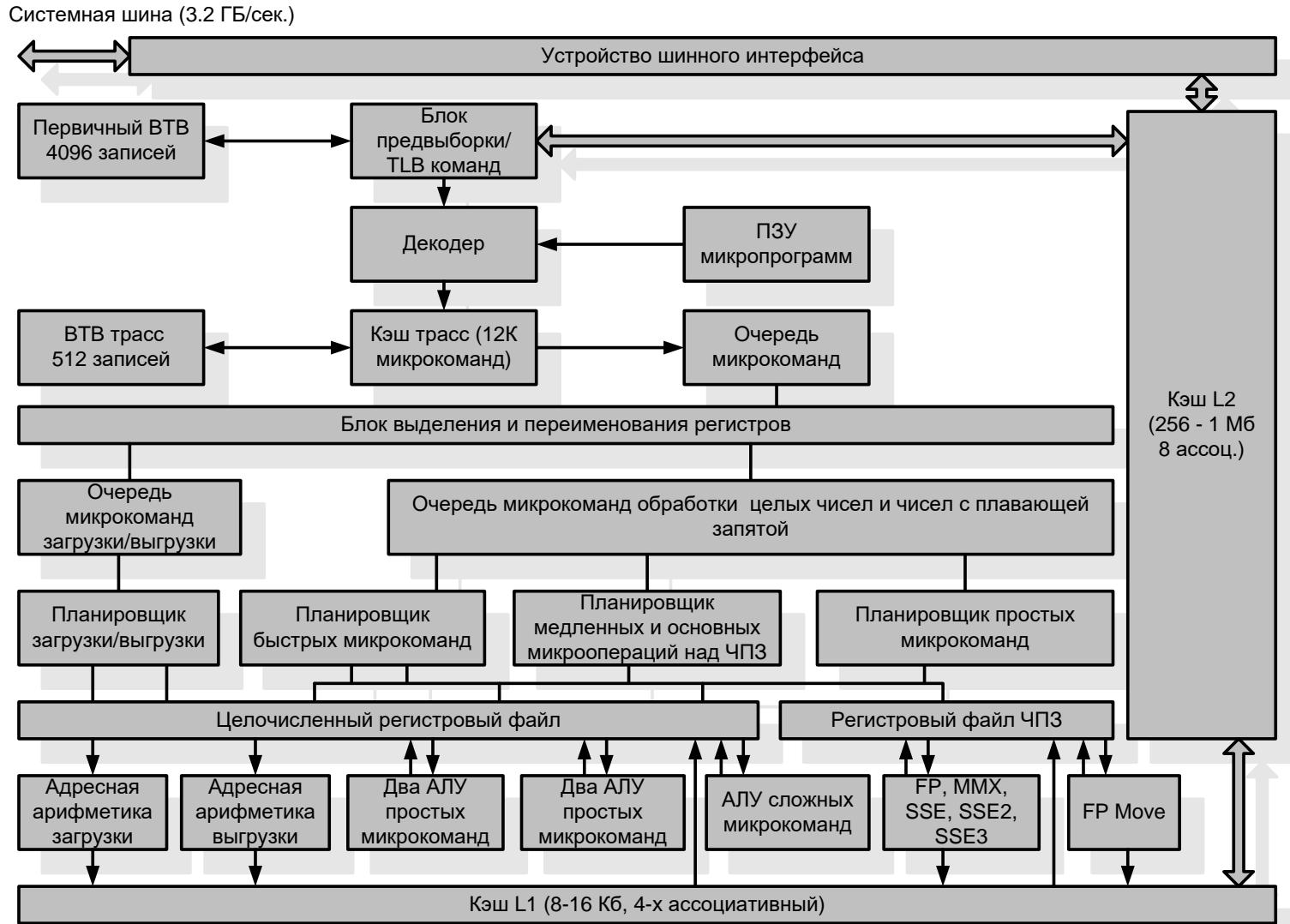
Недостатки Р6

- Длительное декодирование сложных команд.
- Отсутствие слияния микроопераций загрузки/выгрузки и обработки.
- Малое количество входов ROB.
- Наличие медленных команд.

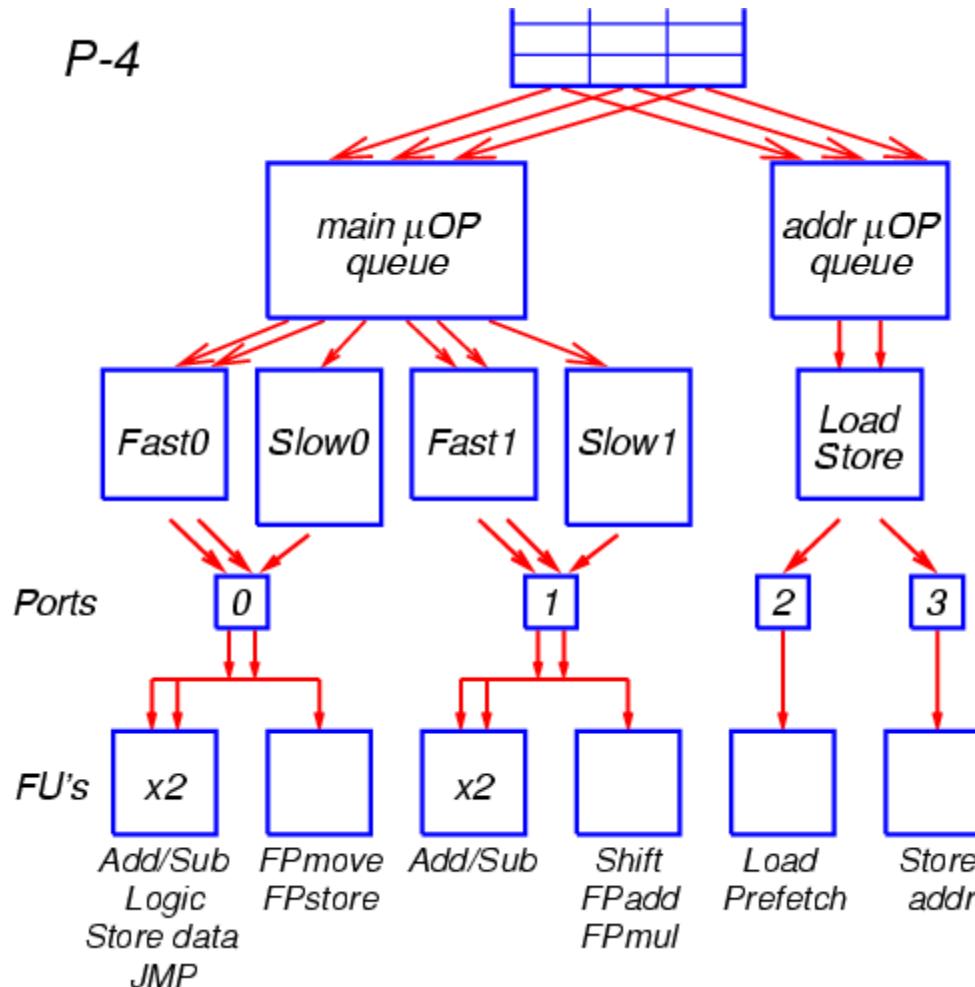
Отличие архитектуры NetBurst от P6

- Использование кэш-памяти первого уровня для хранения декодированных команд (кэш трасс, 12 КМОП). Это позволяет разворачивать циклы, ускоряет декодирование за счет выборки уже декодированных команд.
- Использование TBWB и TTLB для определения адресов в кэш трасс.
- Механизм ранней спекулятивной диспетчеризации, заключающийся в продвижении на исполнение МОПов, ожидающих операнды уже обрабатываемых МОПов.
- Разделение МОПов на медленные и быстрые.
- Работа АЛУ на удвоенной частоте.
- Увеличение длины ROB до 126 входов.
- Увеличение размеров регистров замещения.
- Увеличение размеров других буферов (BTB до 4096 и т.д.)
- Слияние микроопераций загрузки/выгрузки и обработки.

Особенности микроархитектуры NetBurst



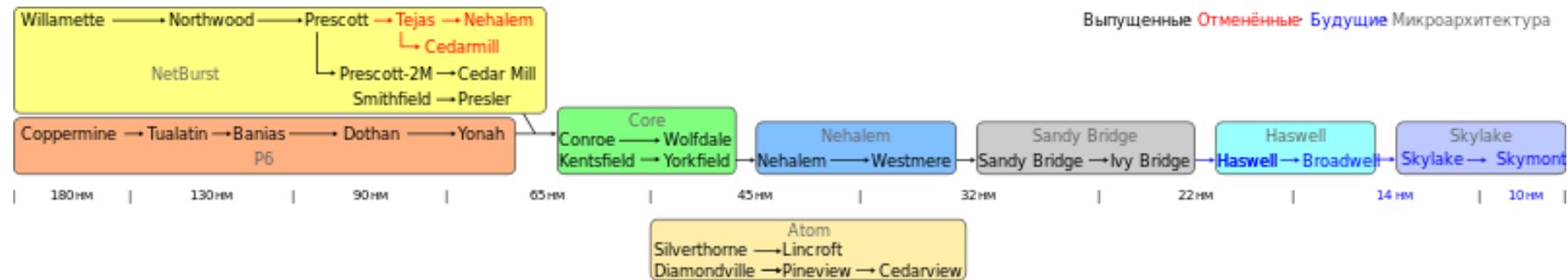
Планировщик в NetBurst



Конвейеры микропроцессоров Intel

Микроархитектура	Количество стадий конвейера
<u>486</u> (80486)	3
<u>P5</u> (Pentium)	5
<u>P6</u> (Pentium Pro/II)	14 (17 с загрузкой-выгрузкой и отставкой)
P6 (Pentium 3)	8 (11 с загрузкой-выгрузкой и отставкой)
P6 (Pentium M, Yonah)	10 (12 с выборкой и отставкой)
<u>NetBurst</u> (Willamette)	20
NetBurst (Northwood)	20
NetBurst (Prescott)	31
NetBurst (Cedar Mill)	31
<u>Core</u> (Merom/Conroe/Woodcrest)	12 (14 с выборкой и отставкой)
<u>Nehalem</u>	20
<u>Sandy Bridge</u>	14 (16 с выборкой и отставкой)
<u>Haswell</u>	14 (16 с выборкой и отставкой)
<u>Bonnell</u>	16 (19 с загрузкой-выгрузкой и отставкой)

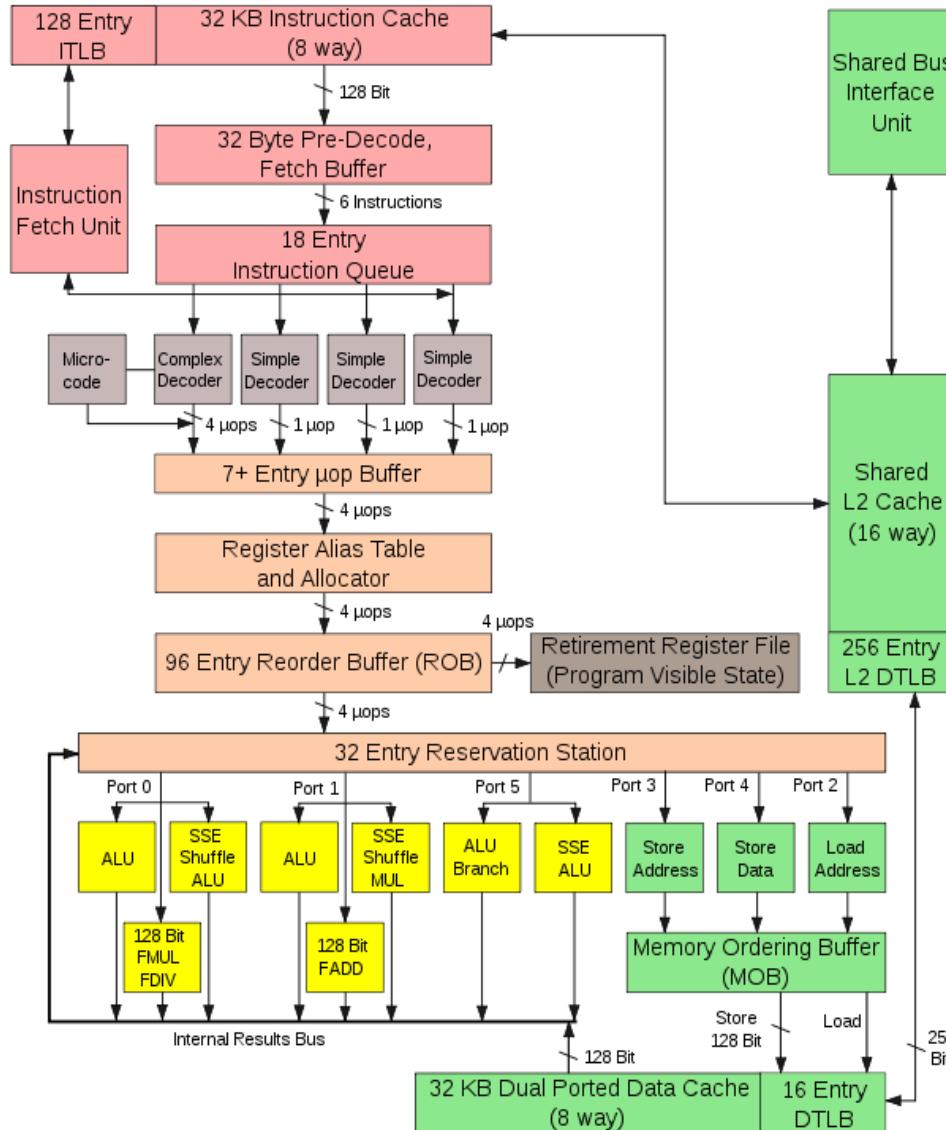
Микроархитектуры процессоров Intel



<http://www.ixbt.com/cpu/sandy-bridge.shtml>

https://en.wikipedia.org/wiki/List_of_Intel_CPU_microarchitectures https://ru.wikipedia.org/wiki/Sandy_Bridge

Микроархитектура Core (P6+)

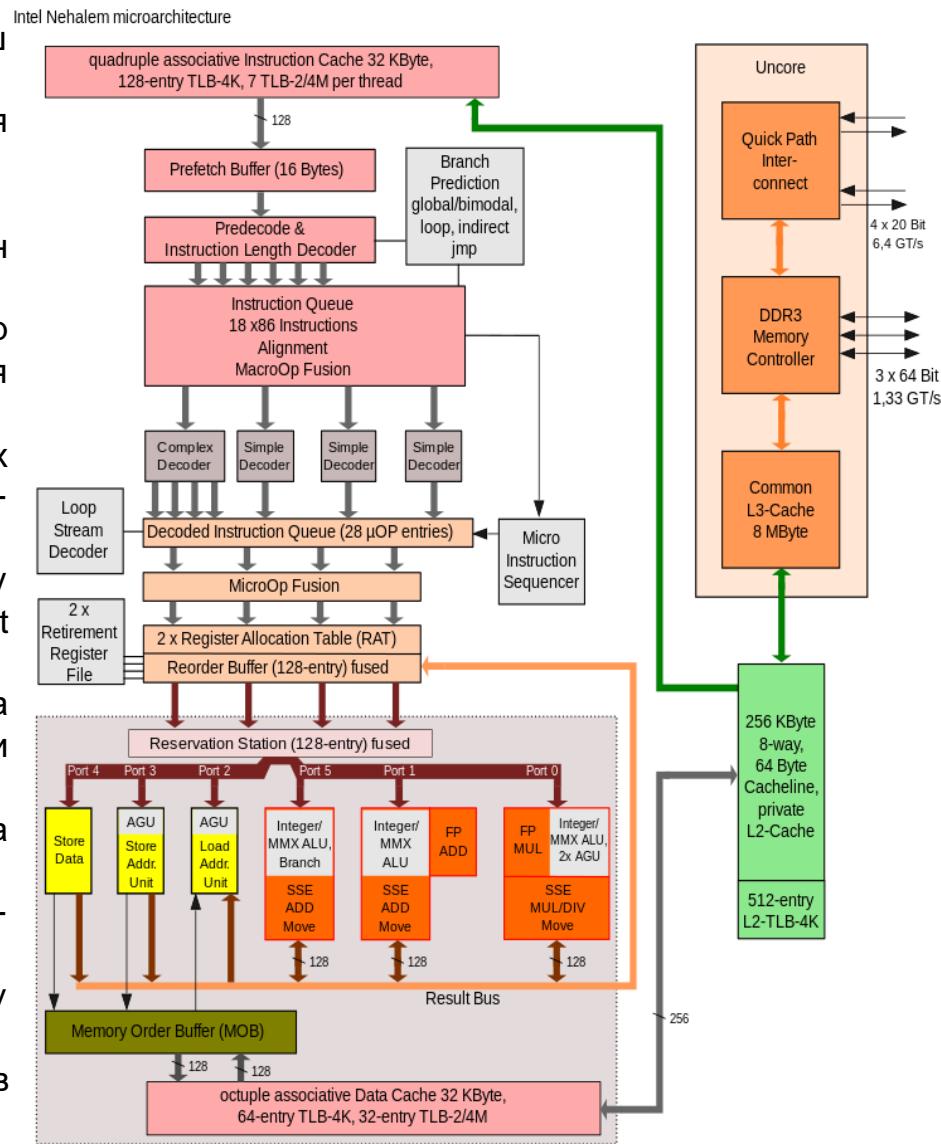


Intel Core 2 Architecture

Микроархитектура Nehalem

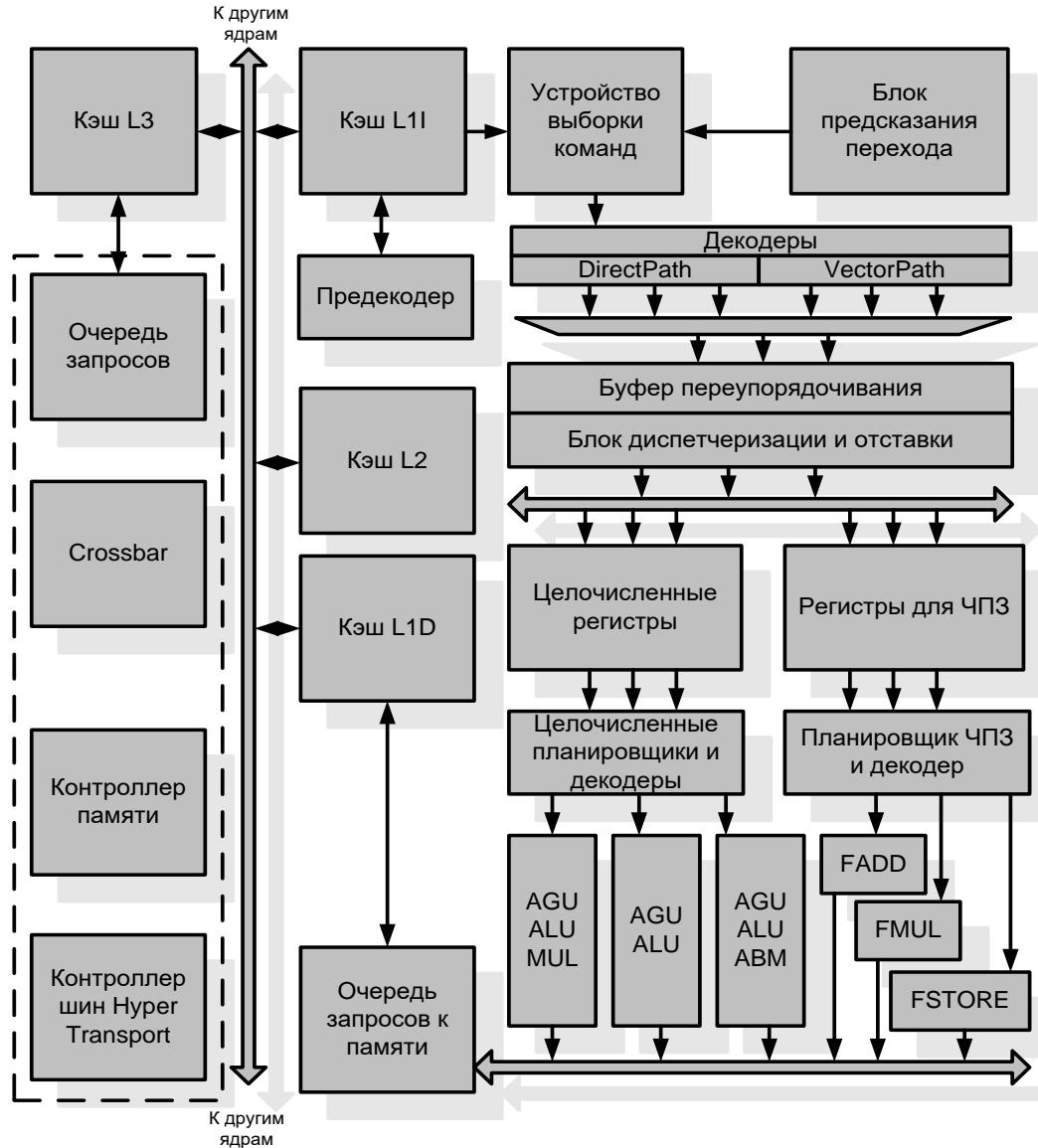
Особенности микроархитектуры Nehalem:

- 32 KB data + 32 KB L1I (4 clocks) и 256 KB L2 кэш (11 clocks) на одно ядро.
- Разделяемая L3 кэш-память, доступная для графического ядра.
- 64-байта размер кэш-линейки.
- Выполнение до двух команд загрузки/выгрузки за один такт для каждого канала памяти
- Кэш для хранения декодированных микрокоманд (iop cache) и более совершенны блок предсказания направления ветвления.
- Повышенная производительность для математических функций, AES кодирования (AES instruction set), и SHA-1 хэширования.
- 256-битная шина с топологией кольца для связи между ядрами графическим ядром, кэш и System Agent Domain (Advanced Vector Extensions).
- Advanced Vector Extensions (AVX) длина вектора расширена до 256 бит, добавлены новые команды и расширен синтаксис.
- Intel Quick Sync Video - аппаратная поддержка кодирования/декодирования видео.
- До 8 физических ядер (16 логических ядер при Hyper-threading) на кристалл.
- Интеграция GMCH (integrated graphics and memory controller) и процессоров на одном кристалле.
- От 14 до 19 ступеней конвейеров команд (в зависимости от промахов в кэш).



GT/s: gigatransfers per second

Микроархитектура AMD64

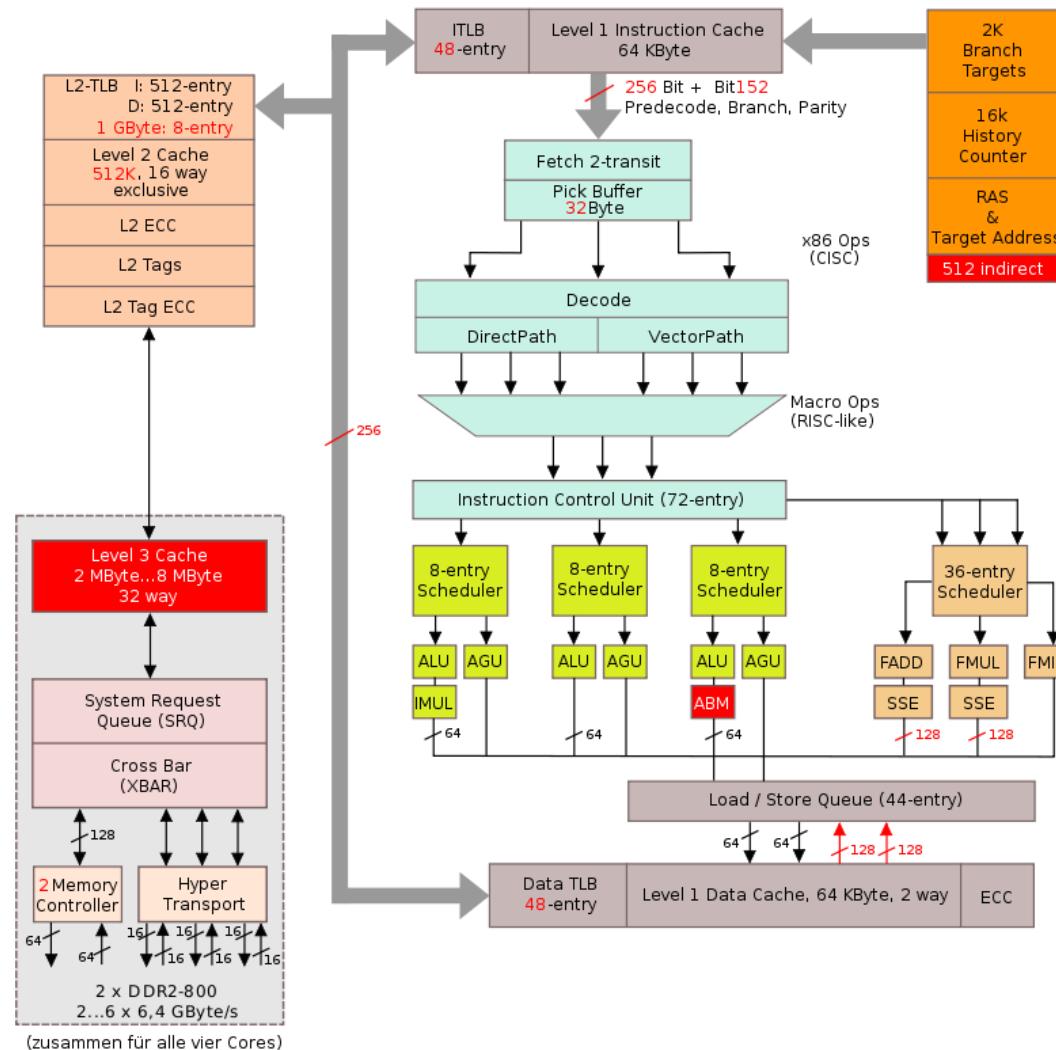


Микроархитектура AMD K10

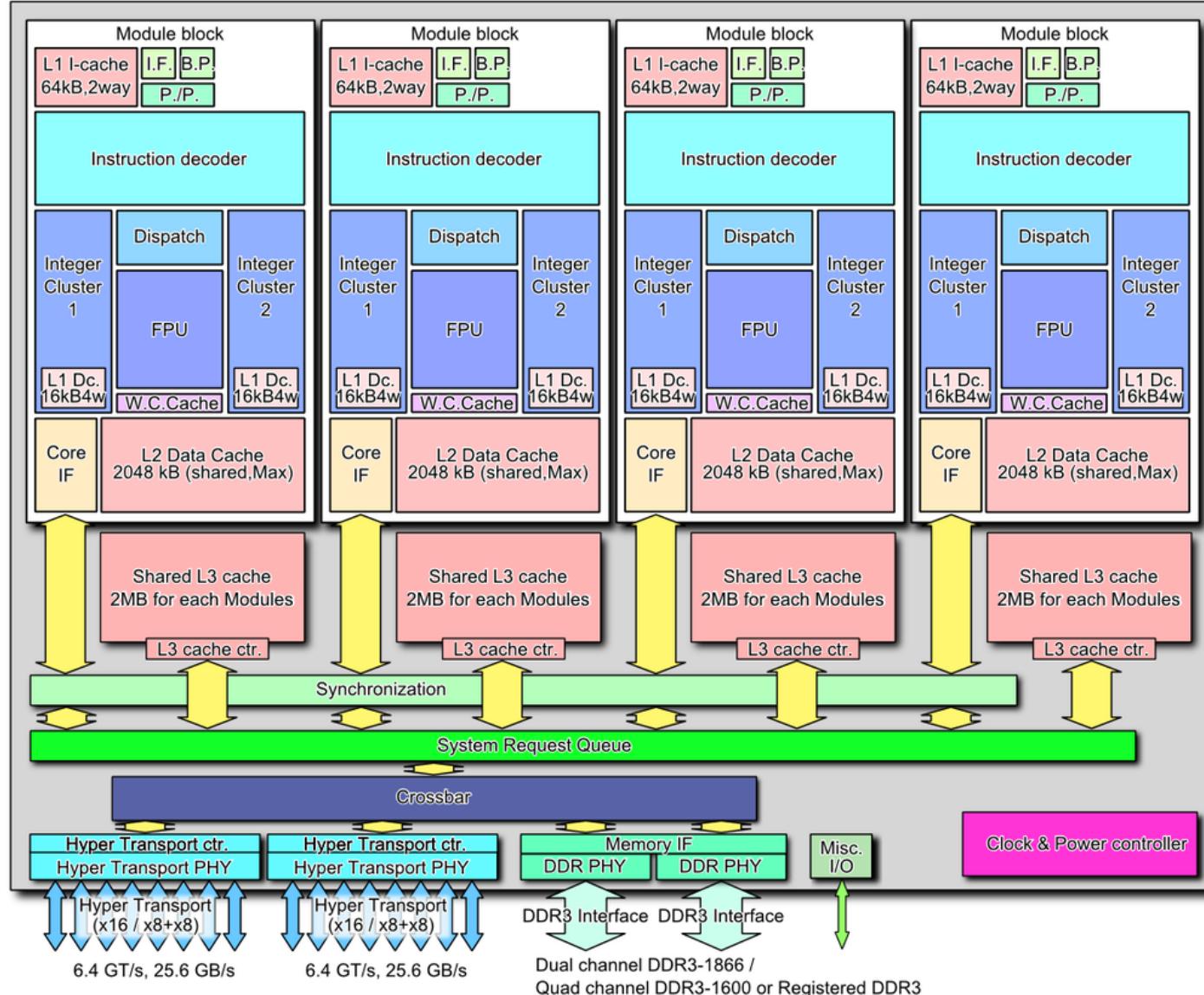
AMD K10 Architecture

Red: Difference between K8 and K10 Architecture

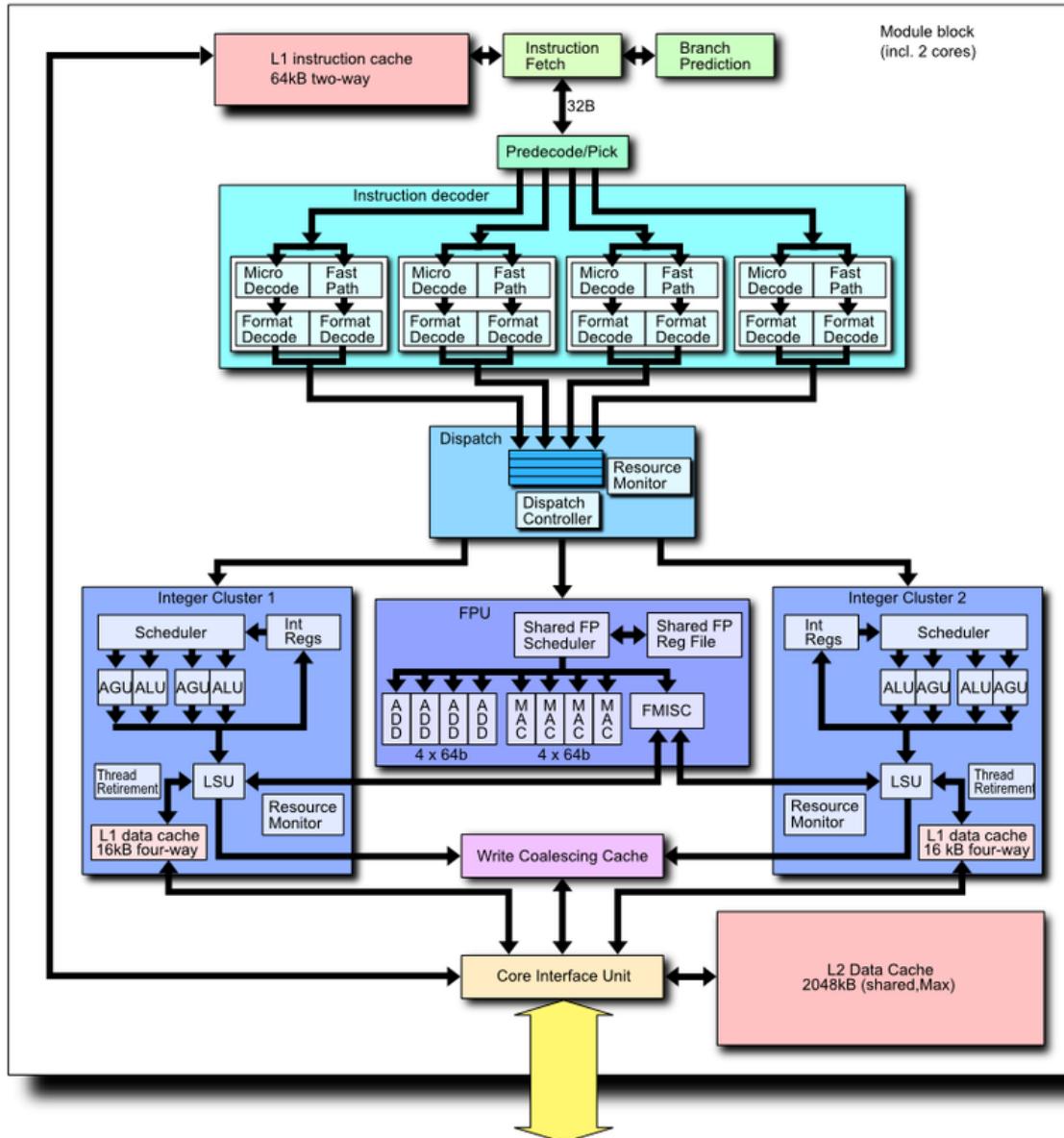
(Die Änderungen zwischen der K8- und K10-Architektur sind rot markiert)



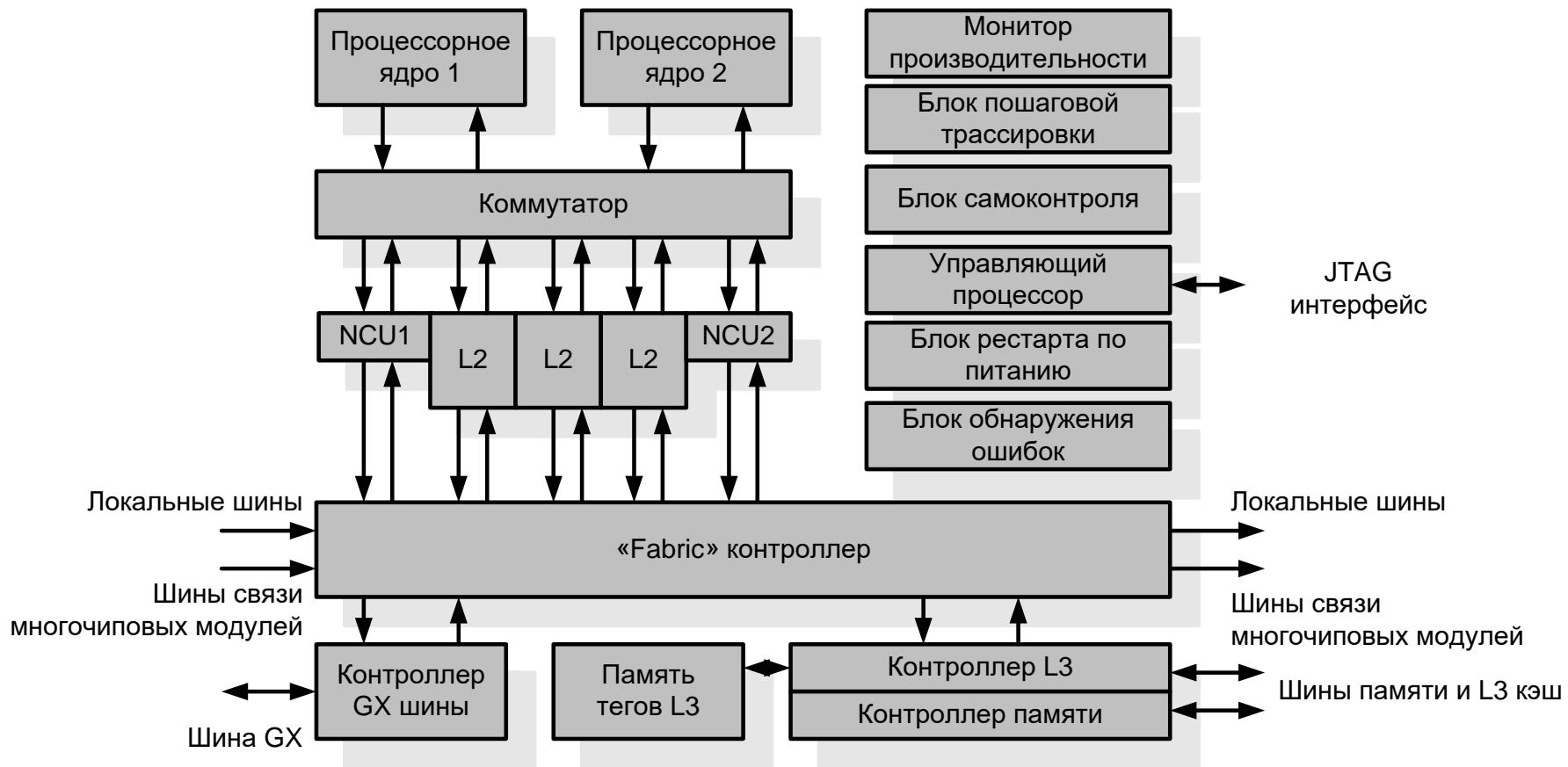
Многоядерный чип AMD Bulldozer



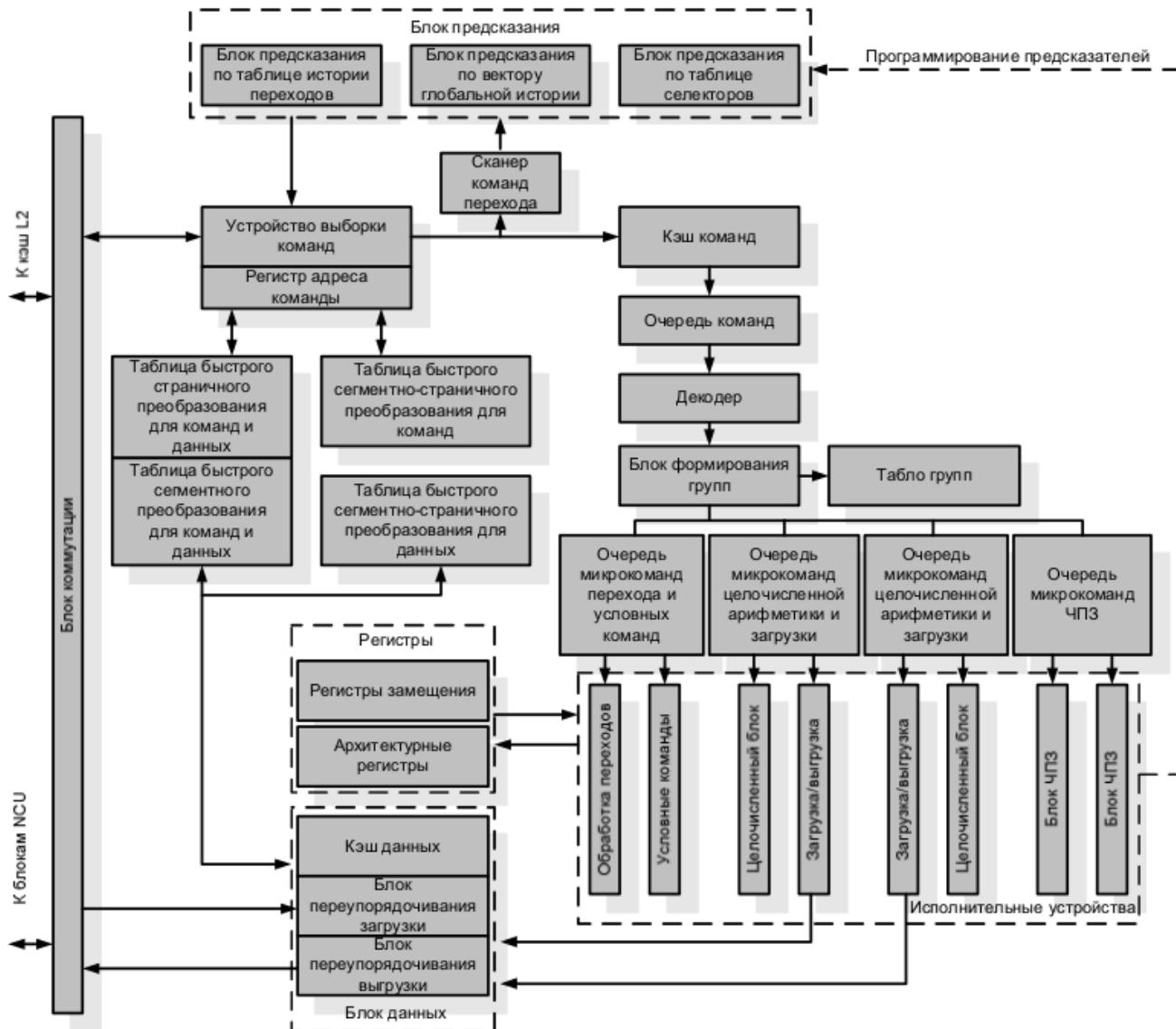
Микроархитектура ядра AMD Bulldozer



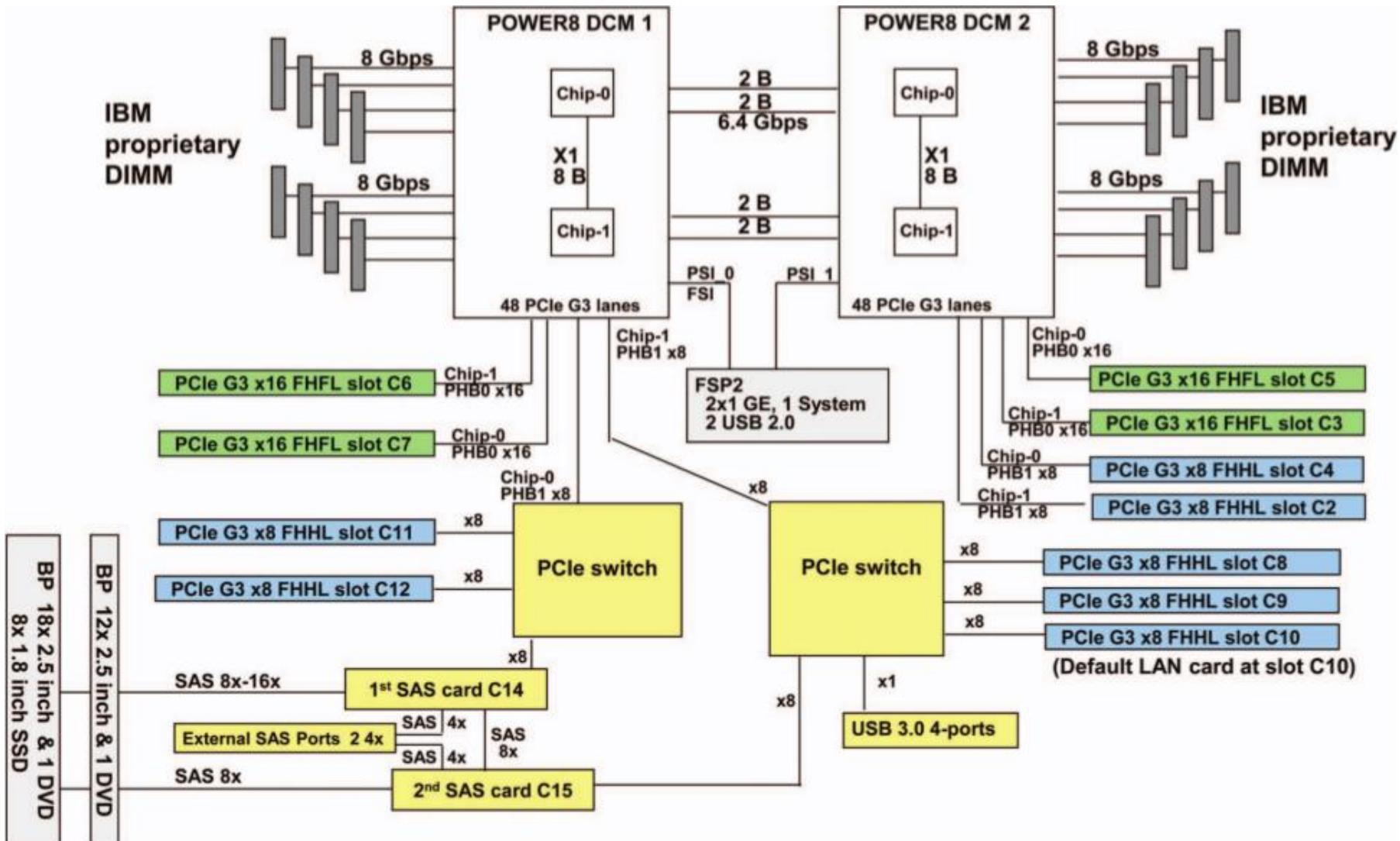
Микроархитектура суперскалярных процессоров IBM POWER4



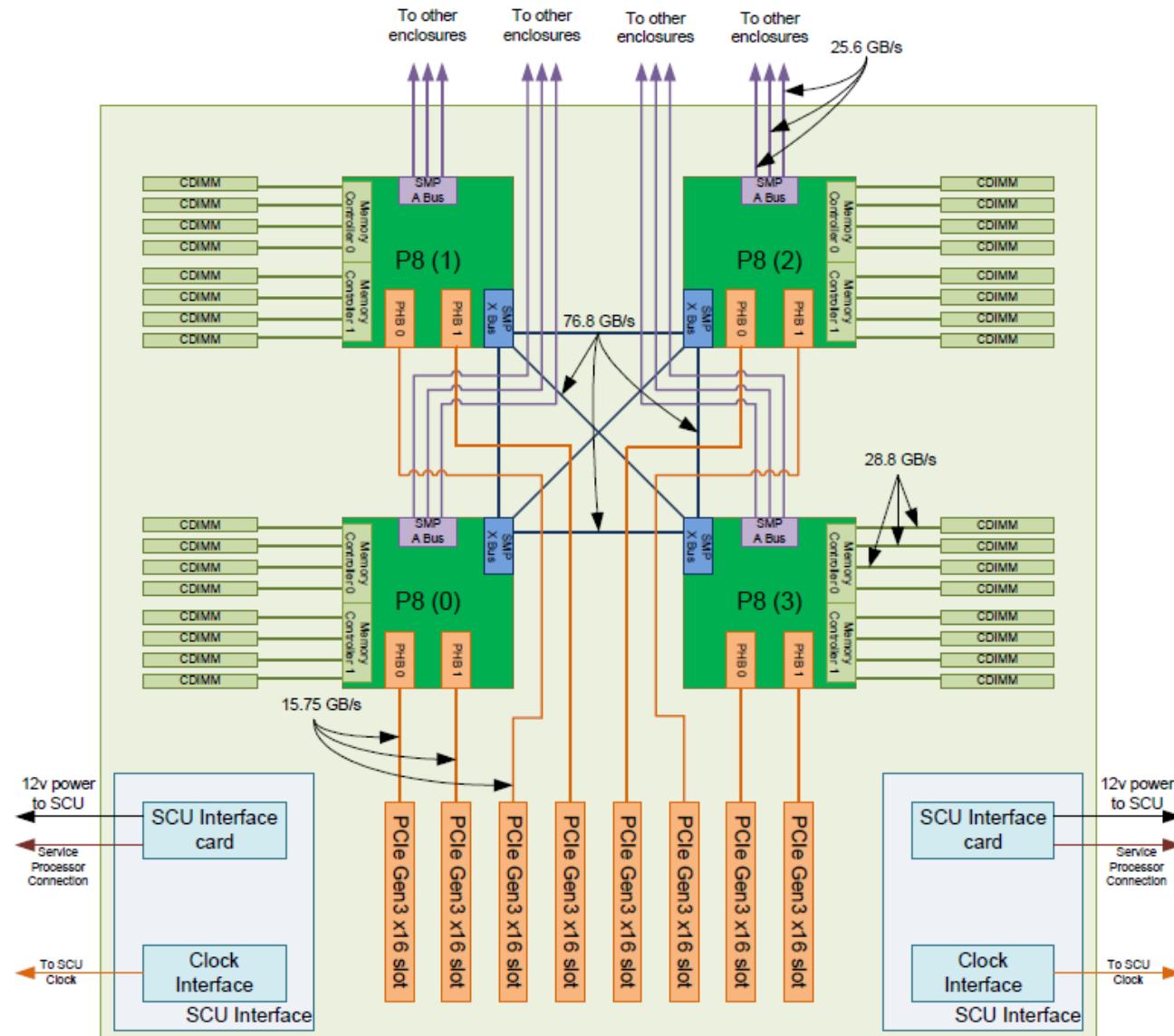
Структура процессорного ядра IBM POWER4



Серверная платформа IBM POWER8

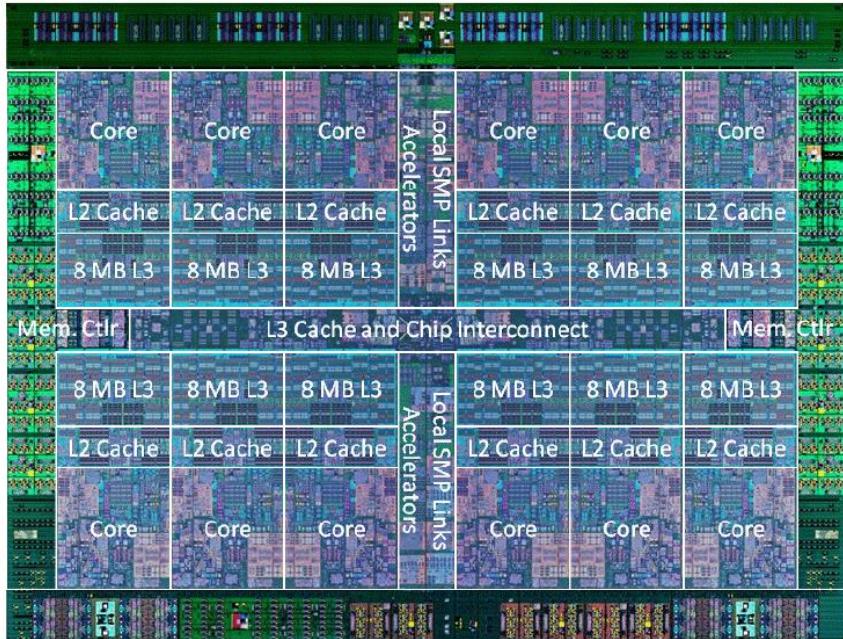


Вычислительный узел серверной платформы IBM POWER8

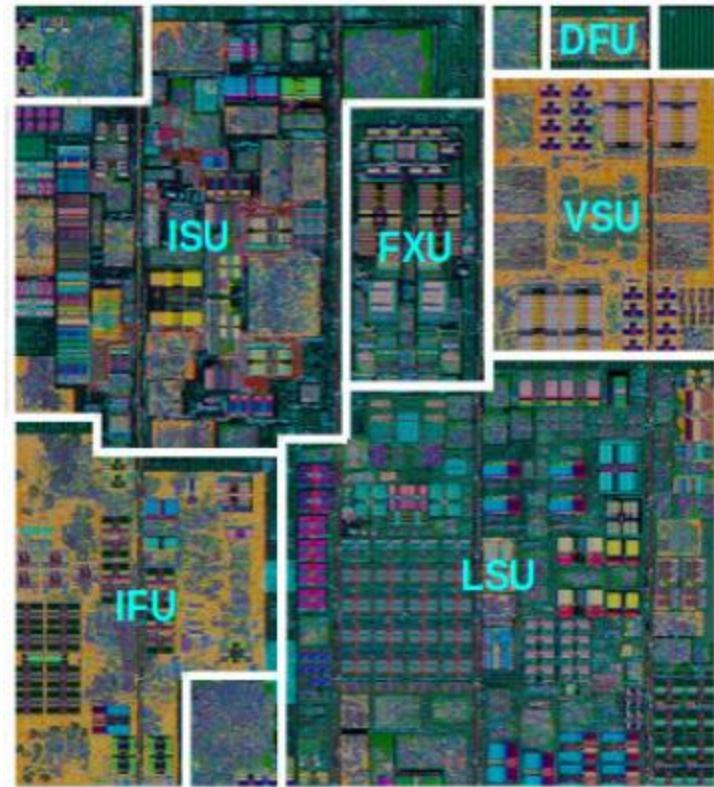


Структура процессорного ядра IBM POWER8

Микросхема POWER8

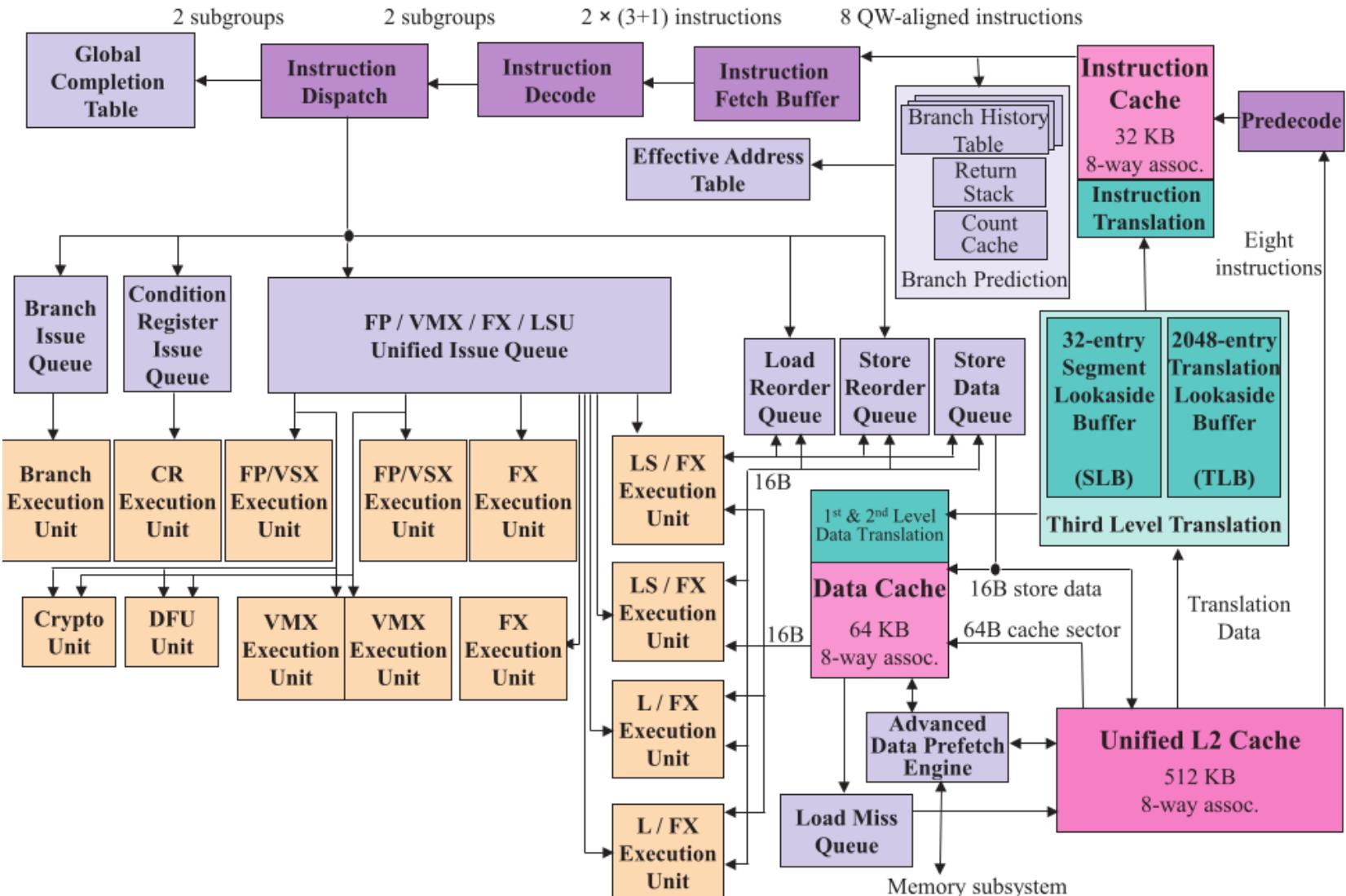


Микропроцессорное ядро

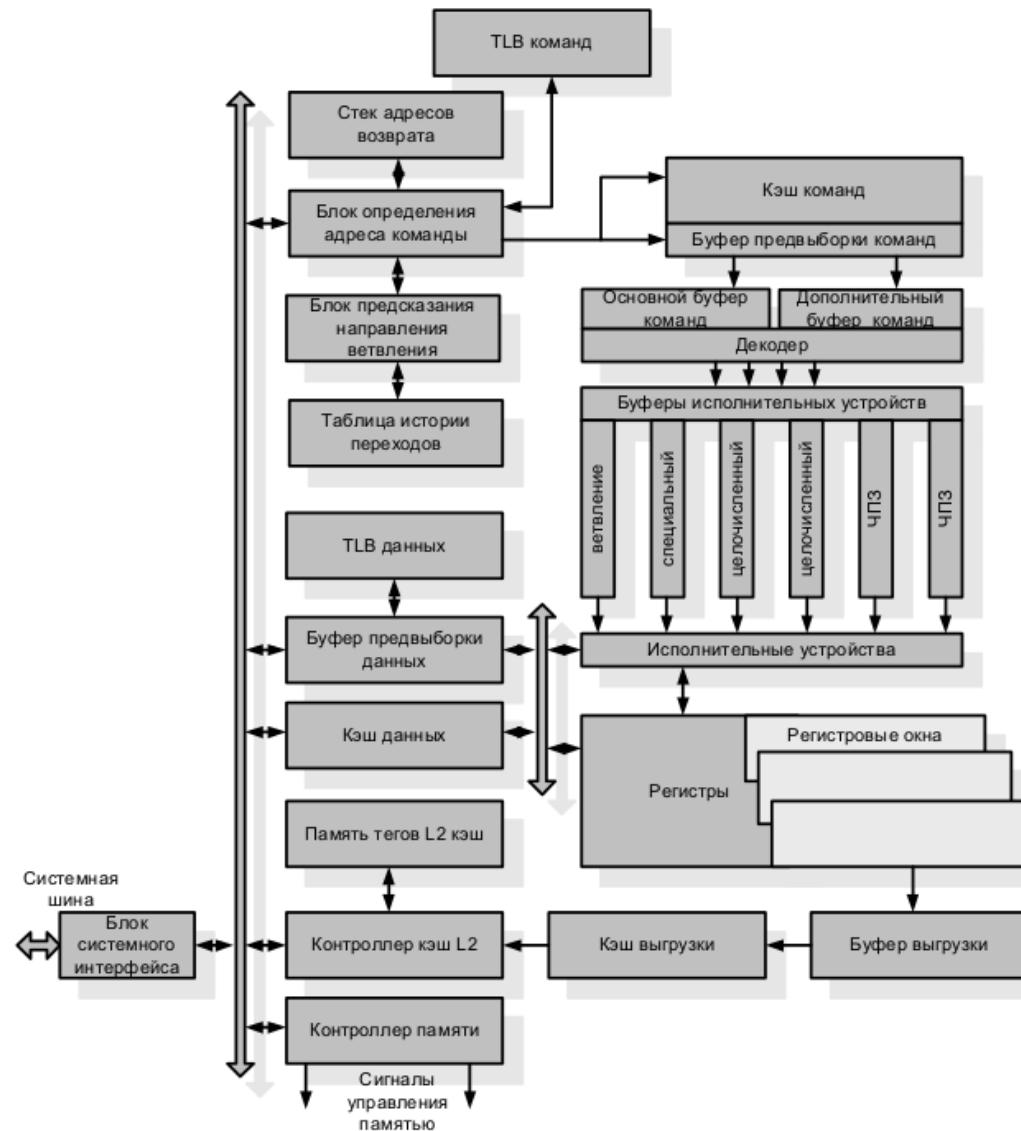


Структура процессорного ядра IBM POWER8

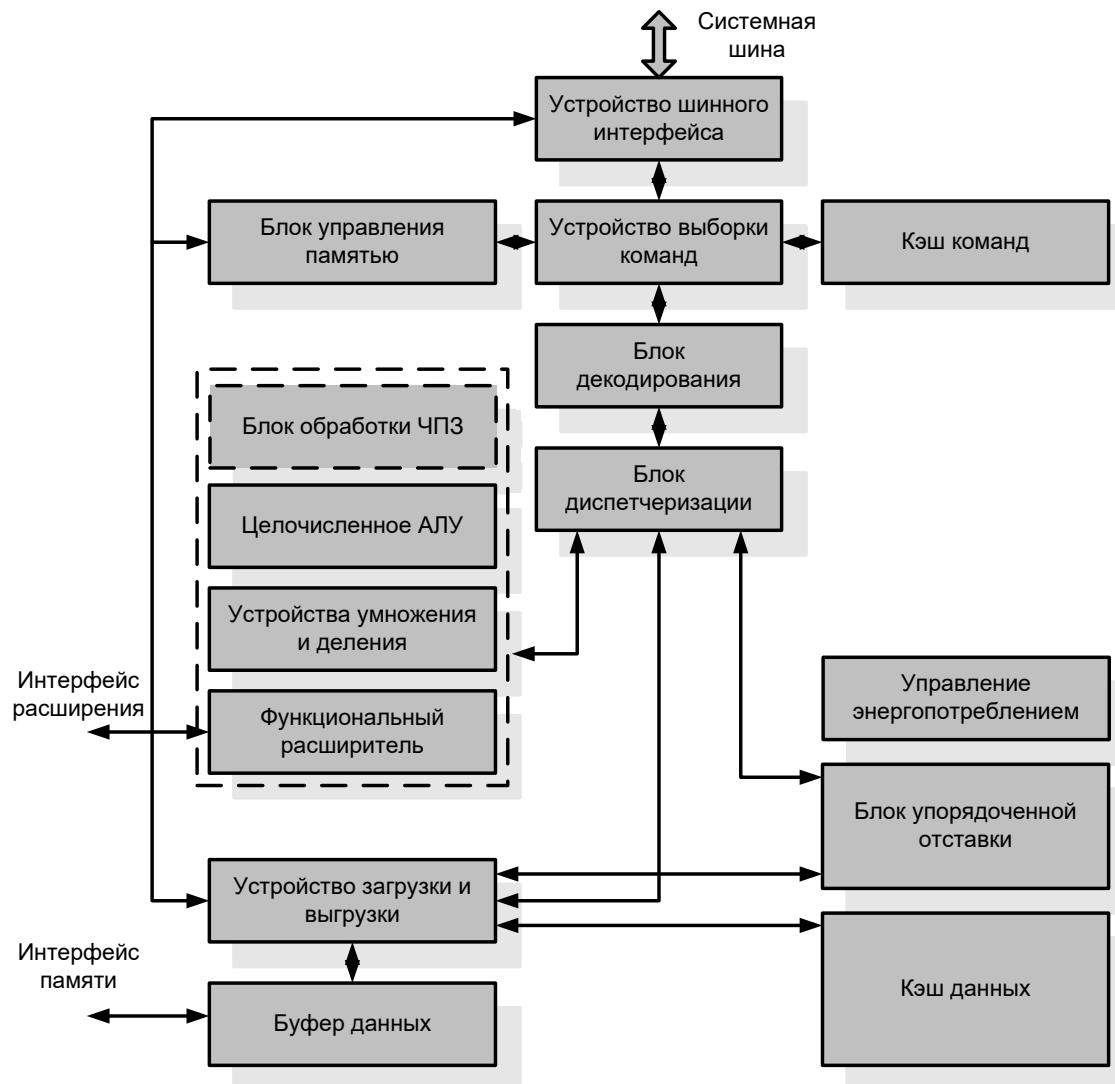
Конвейер



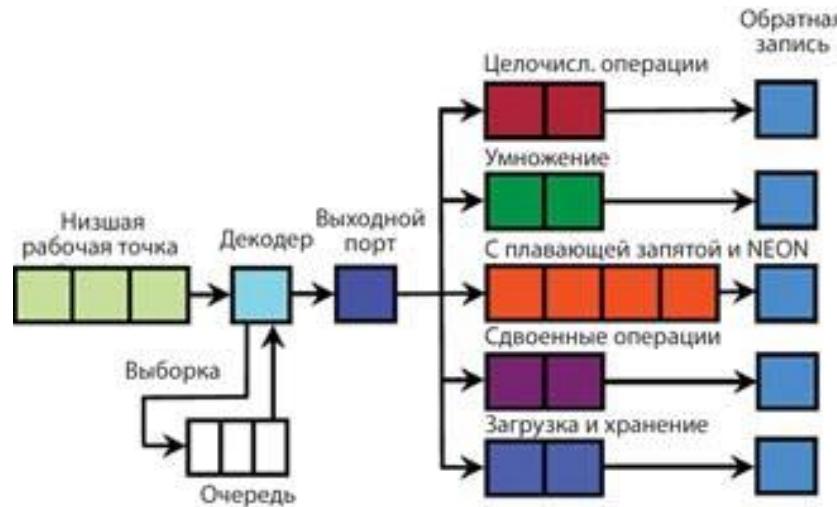
Микроархитектура суперскалярных процессоров Sun UltraSPARC III



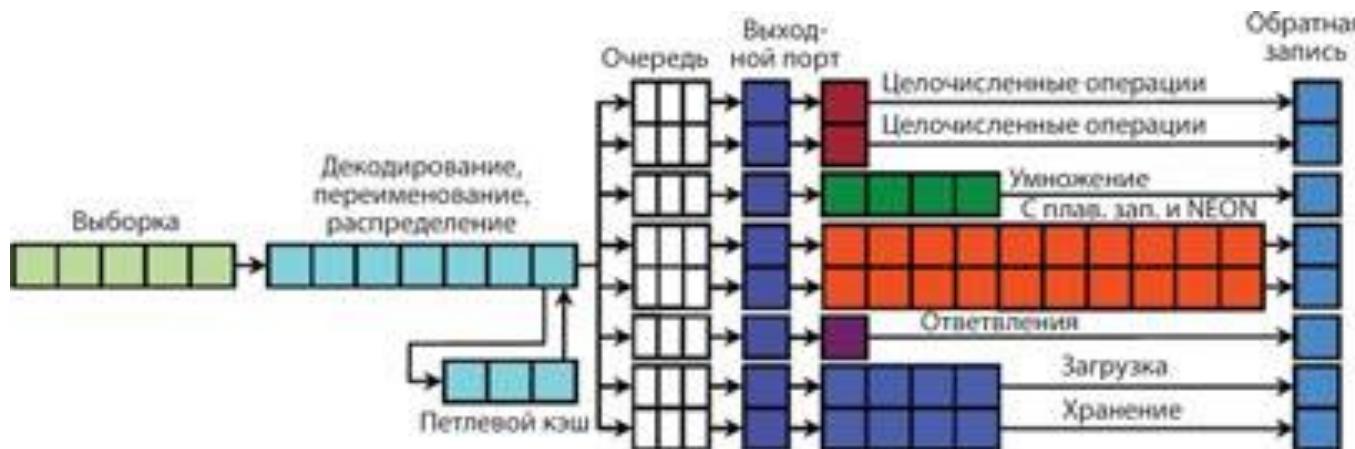
Архитектура синтезируемых суперскалярных процессорных ядер MIPS32 74K



Микроархитектура Cortex-A7

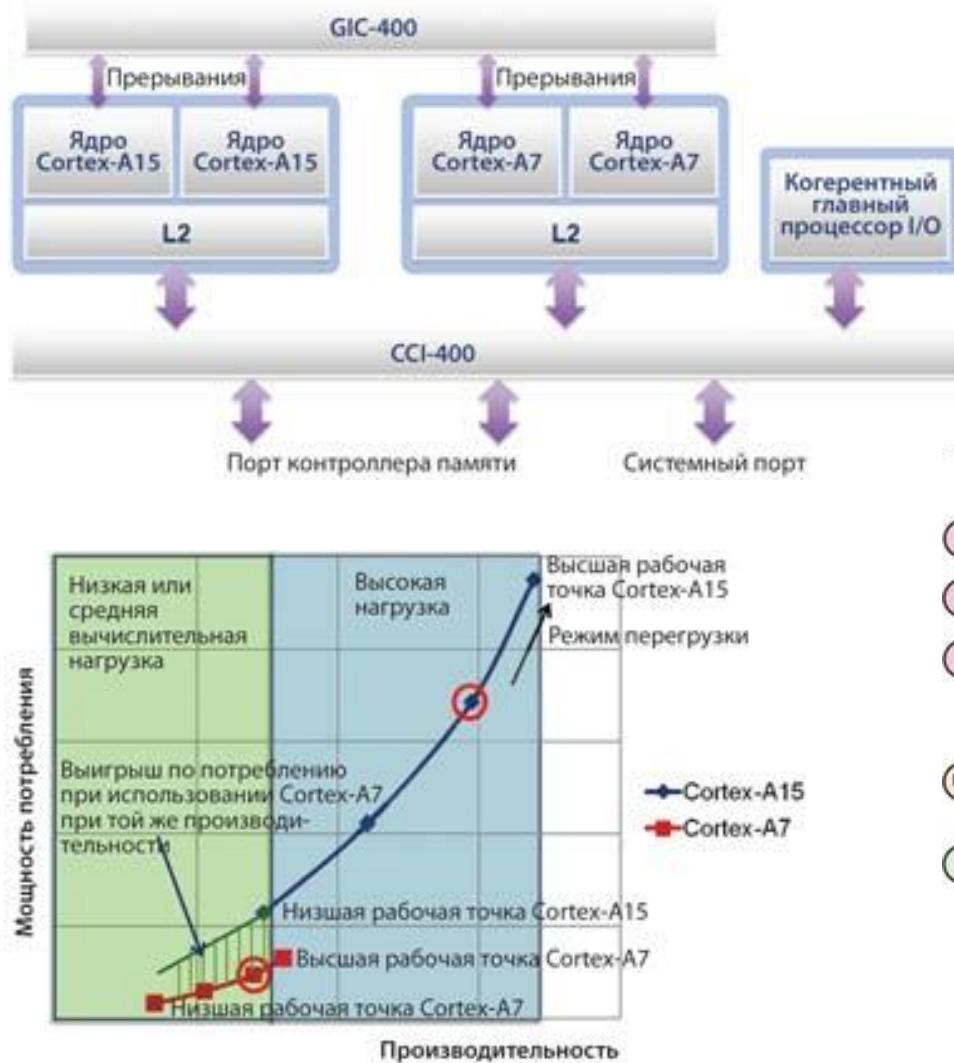


Микроархитектура Cortex-A15

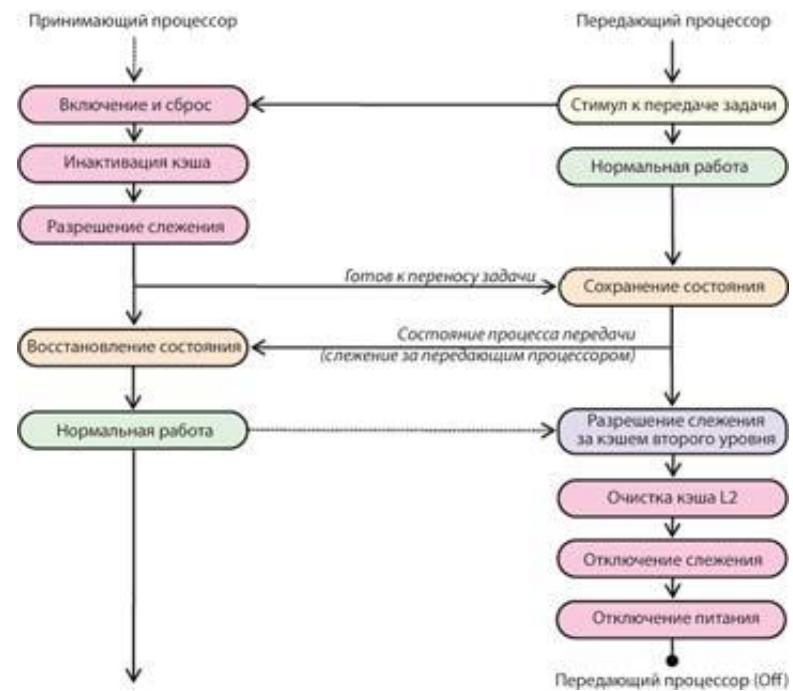


Greenhalgh P. Big.LITTLE processing with ARM Cortex-A15 & Cortex-A7//www.eetimes.com.

big.LITTLE



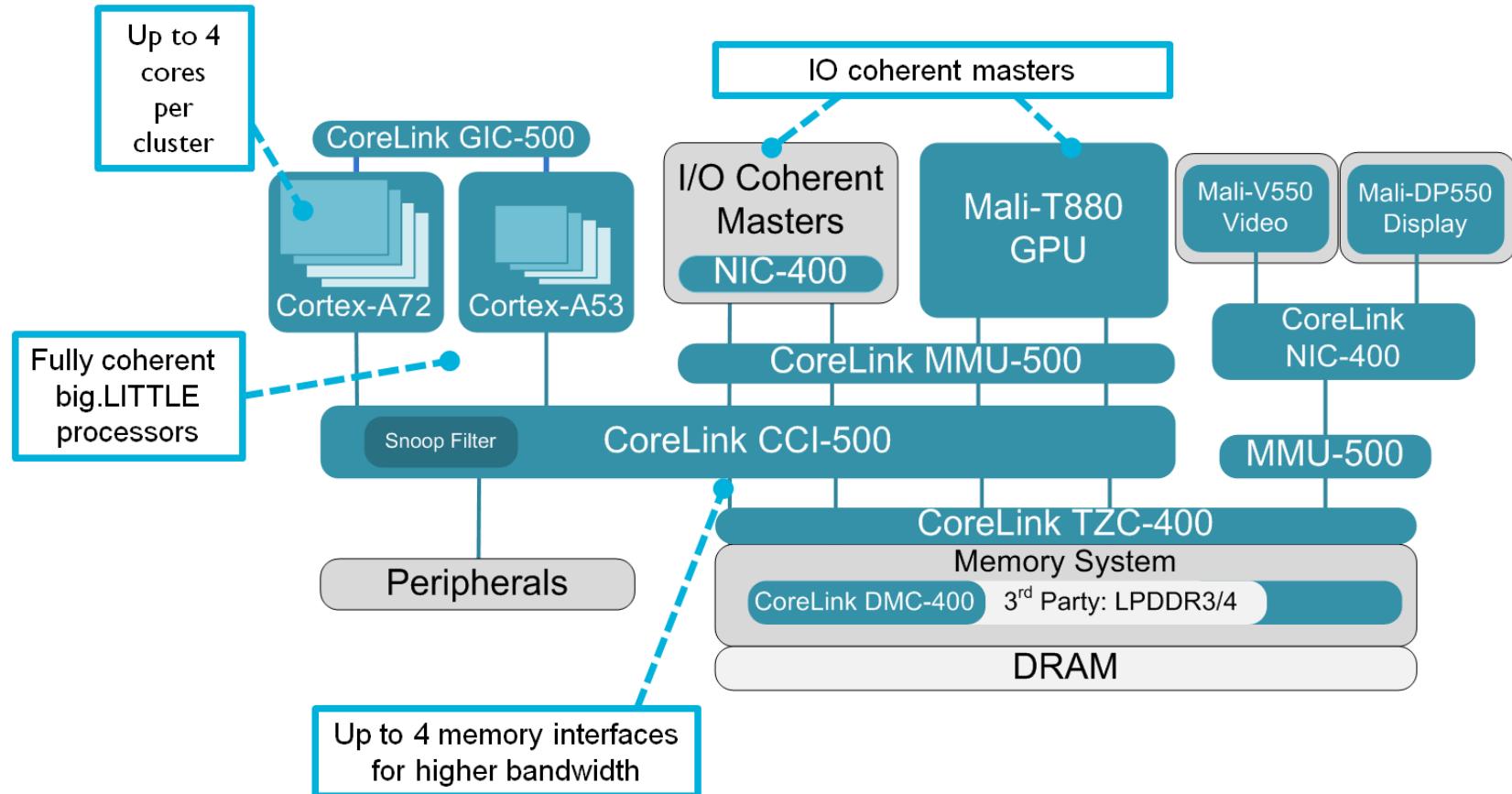
Одним из ключевых элементов является система межсоединений CCI-400, которая обеспечивает полную согласованность между Cortex-A15 и Cortex-A7, а также устройствами ввода-вывода. Контроллер прерываний GIC-400 способен распределить до 480 прерываний, причем прерывания могут направляться в любое ядро в кластере Cortex-A15 или Cortex-A7. big.LITTLE распределяет задачи операционной системы (ОС) и приложений так, чтобы они выполнялись только на одном процессоре в каждый момент времени



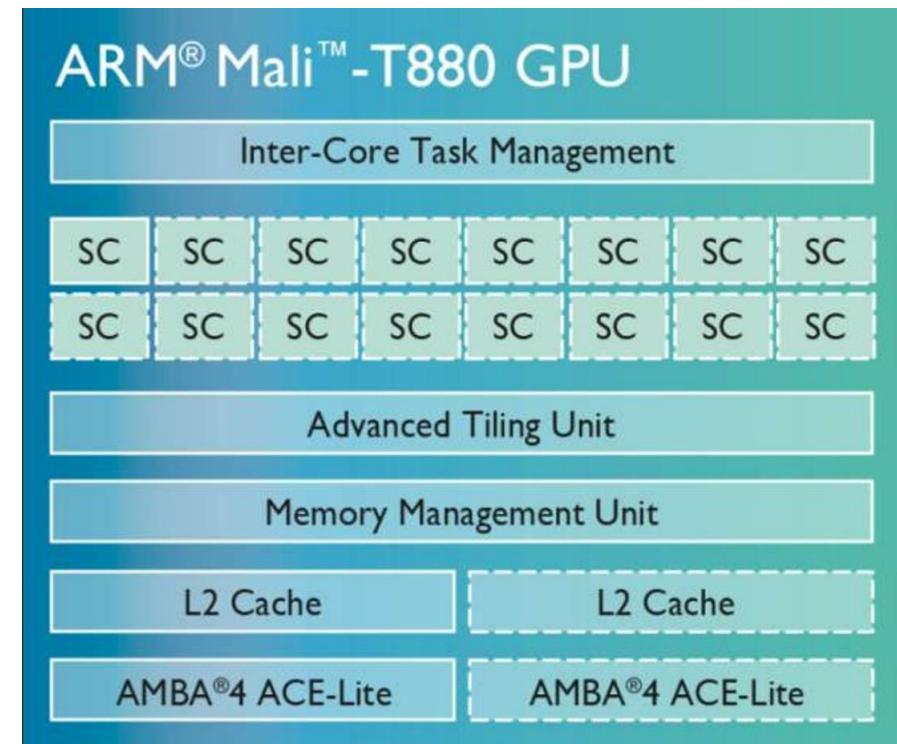
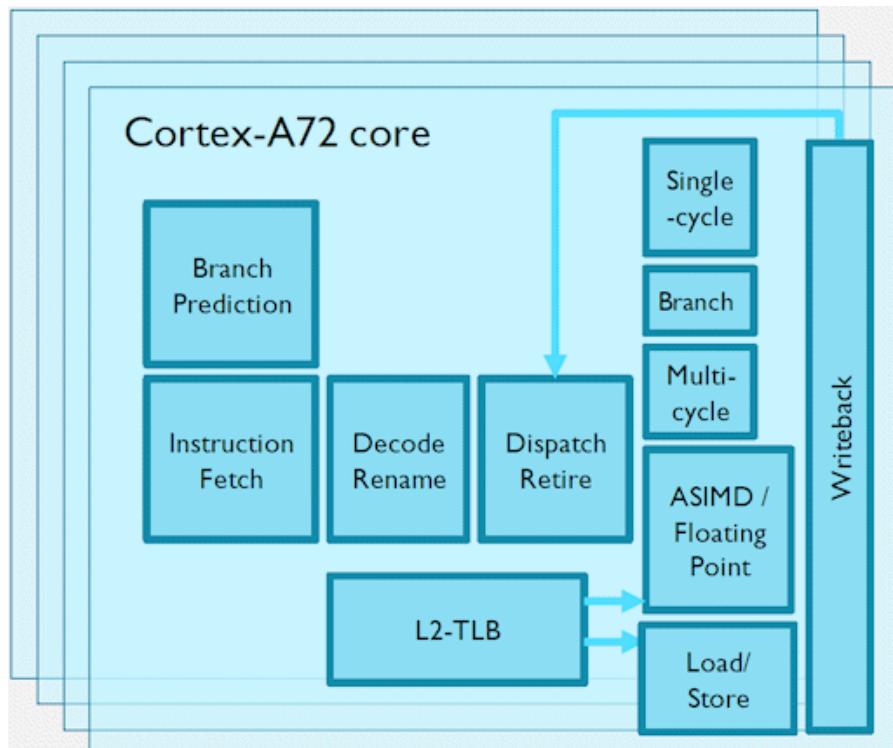
<http://www.russianelectronics.ru/leader-r/review/2192/doc/58808/>

Архитектура системы на кристалле Cortex-A72

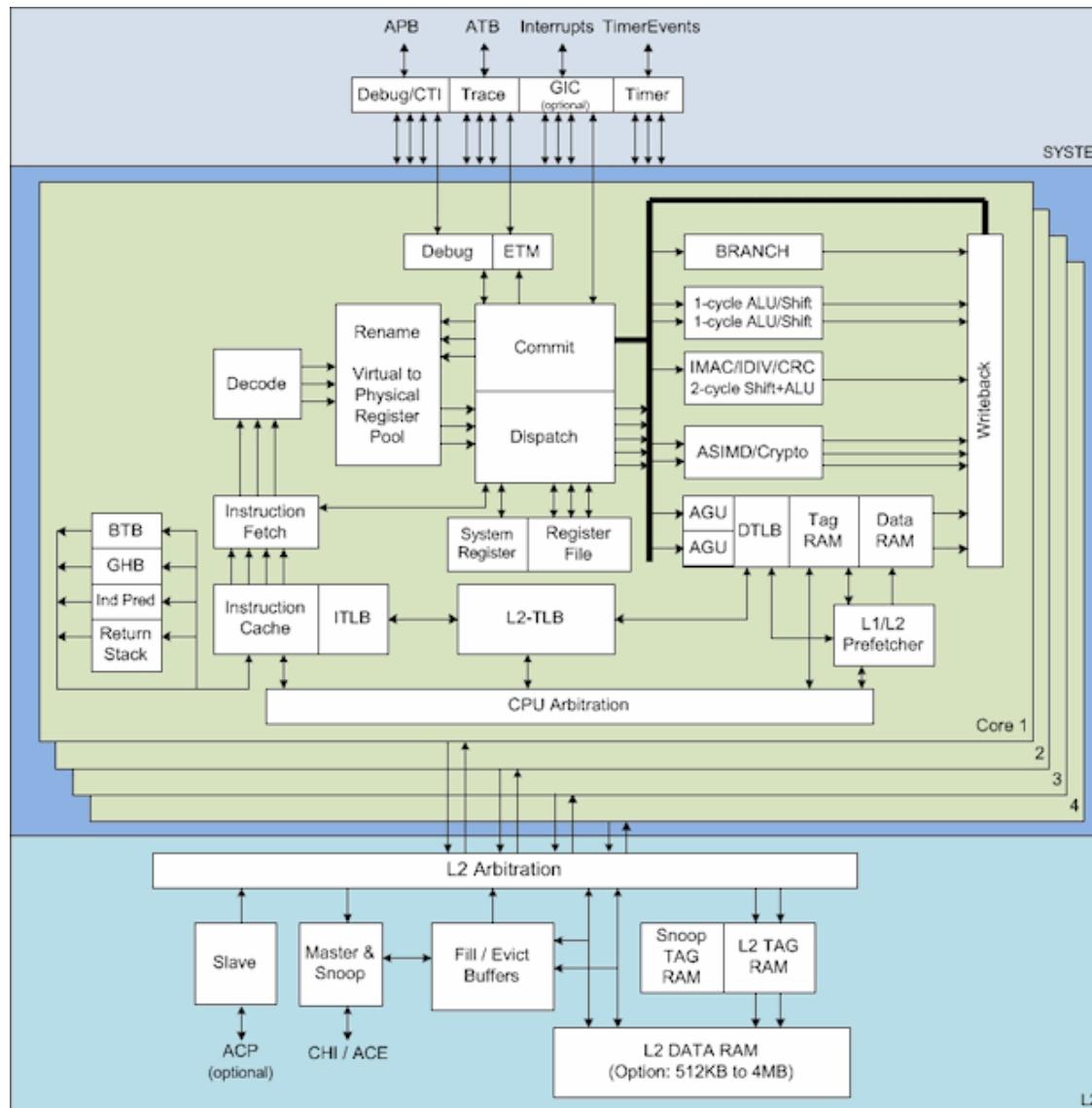
CoreLink™ CCI-500



Микропроцессор Cortex-A72



Микроархитектура Cortex-A72



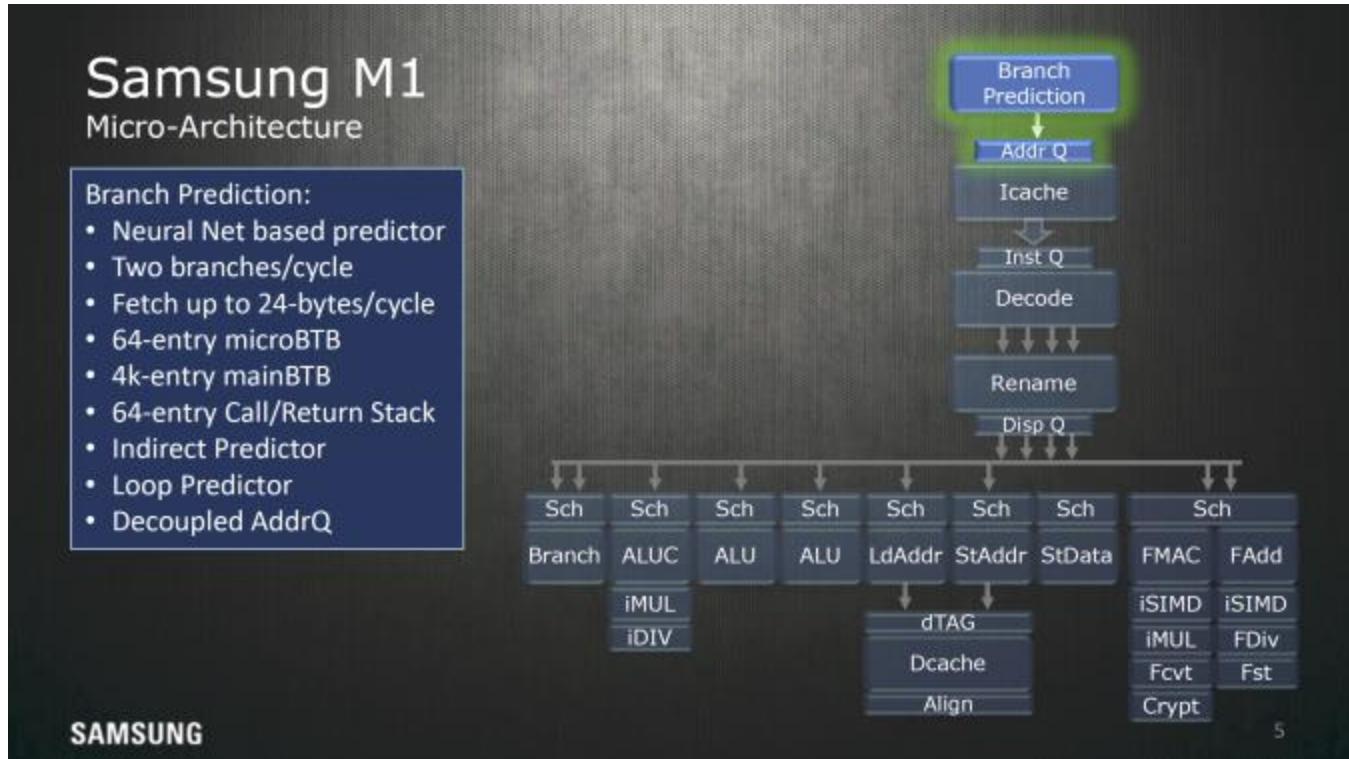
Microarchitecture Overview

- Advanced Branch Prediction
- Quad instruction decode
 - Most instructions map directly to a single uop
- Quad uOP dispatch and retire
- Full Out-of-Order instruction execution
 - Full OoO loads and stores
- Multistride / multistream prefetcher
- Low latency / low power caches

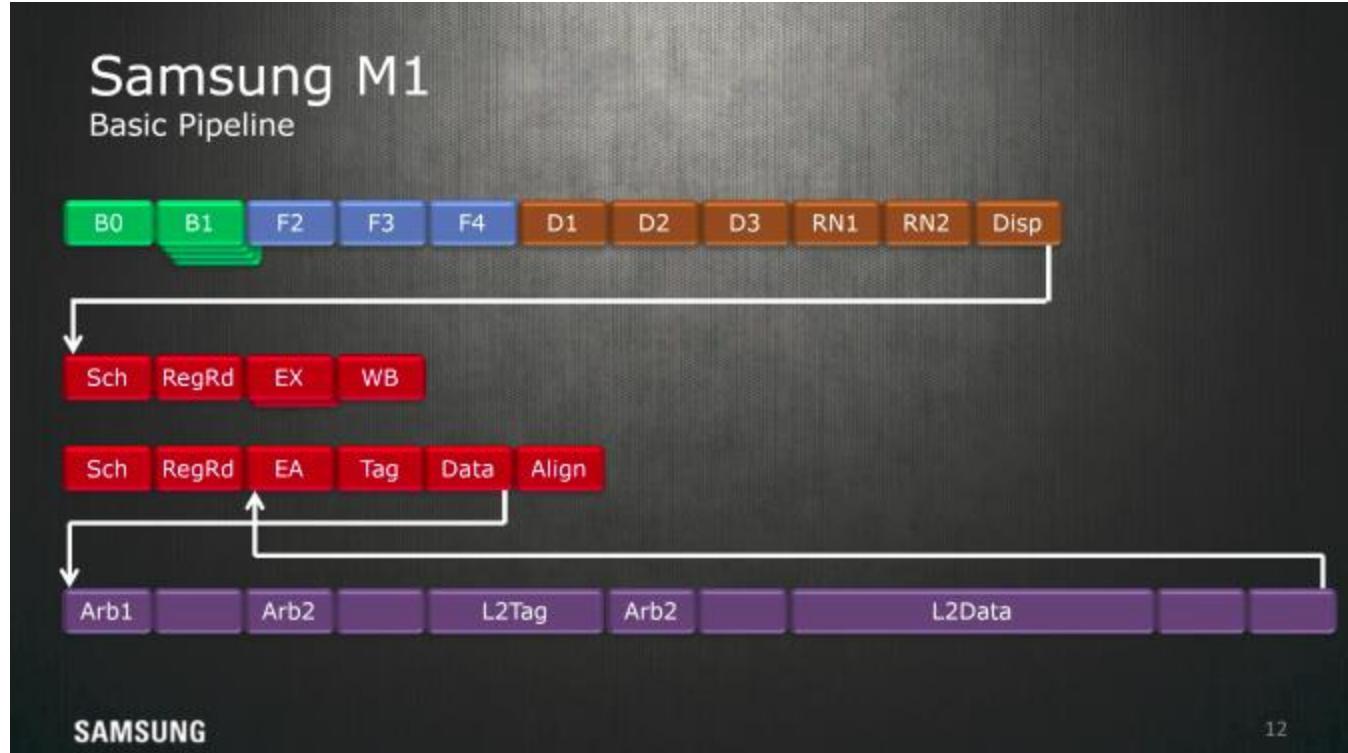
SAMSUNG

3

Exynos M1 CPU (Samsung, 1.6GHz ARM Cortex-A53)

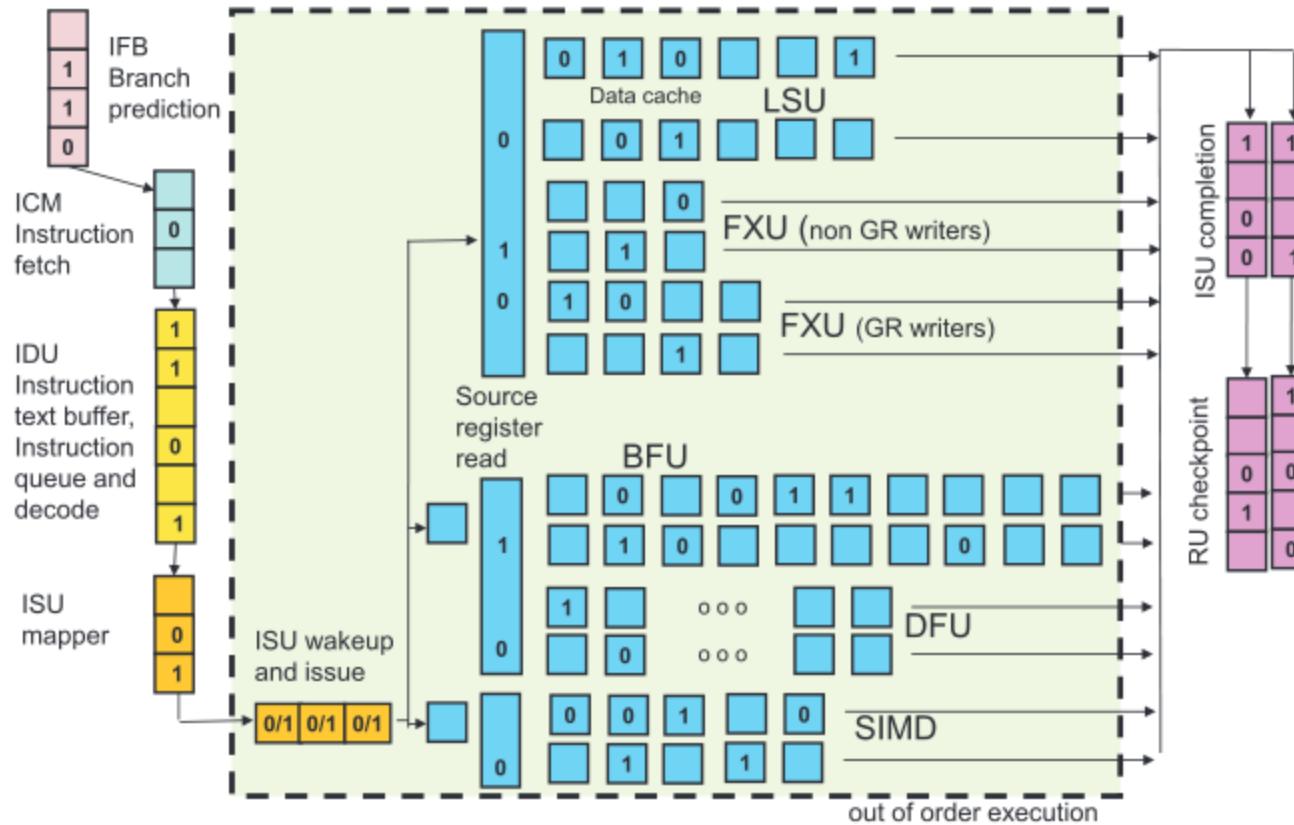


Exynos M1 CPU (Samsung, 1.6GHz ARM Cortex-A53)



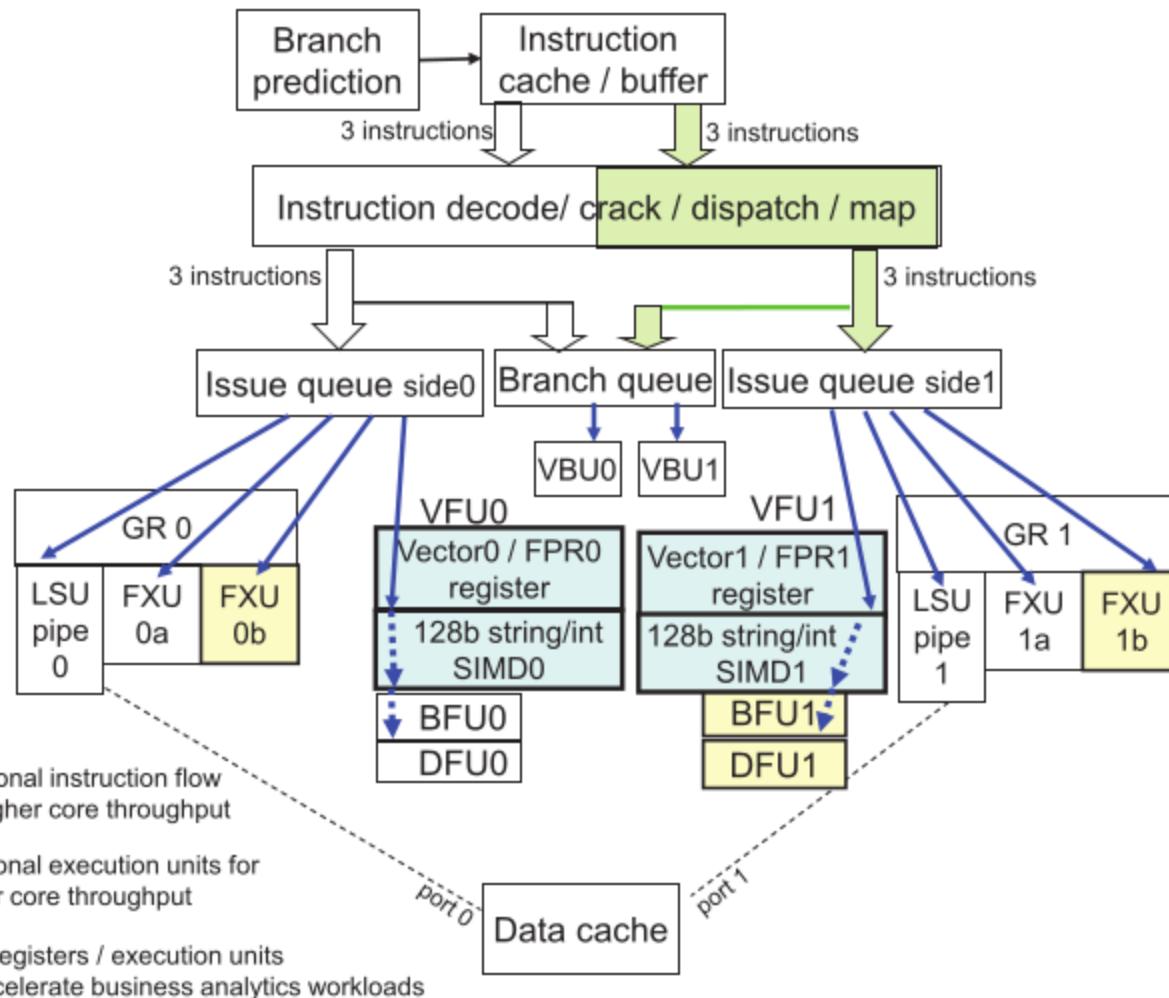
Микроархитектура IBM System z13

Конвейер



Микроархитектура IBM System z13

Стадии обработки команд



IV. Операционные устройства ЭВМ

- Целочисленные арифметико-логические устройства: устройства выполнения логических операций, устройства целочисленного сложения/вычитания, устройства целочисленного умножения, устройства целочисленного деления.

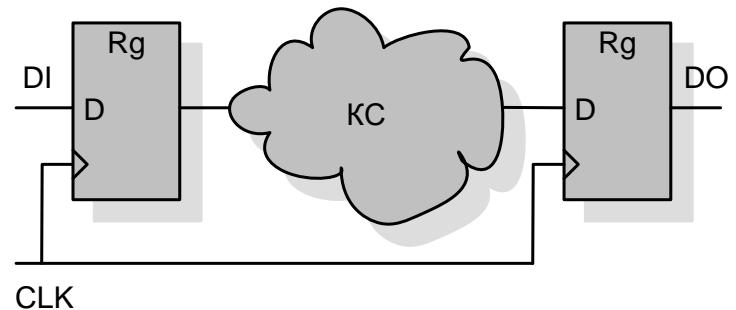
Устройства обработки чисел с плавающей запятой:
устройства сложения/вычитания, устройства умножения, устройства деления, устройства вычисления функций.

- Устройства SSE арифметики.

- Операционные устройства состоят из:

Регистров для хранения
данных;

Шин передачи данных;
Комбинационных схем
реализации функций;



Устройства целочисленного сложения/вычитания

-Накапливающие сумматоры (последовательные).

-Параллельные сумматоры: с последовательным переносом, с параллельным переносом, с условным переносом, с групповой структурой.

Схема последовательного сумматора

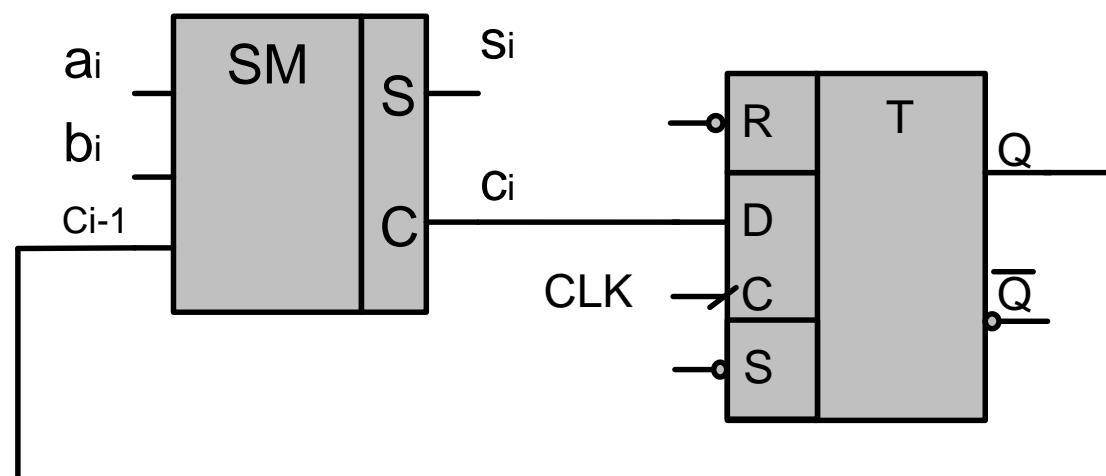


Схема параллельного сумматора с последовательным переносом

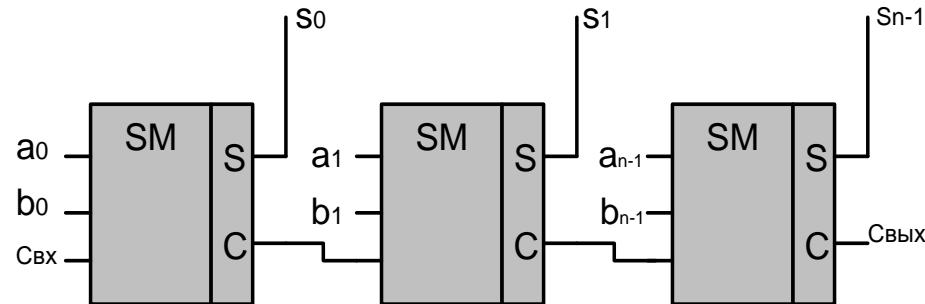


Схема сумматора с условным переносом

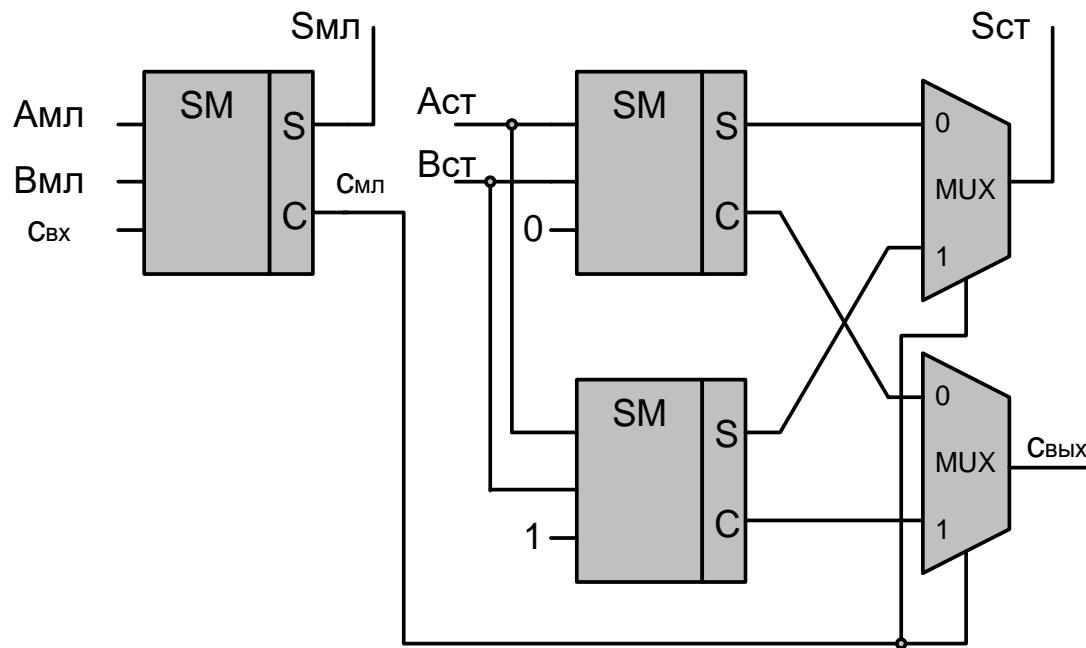
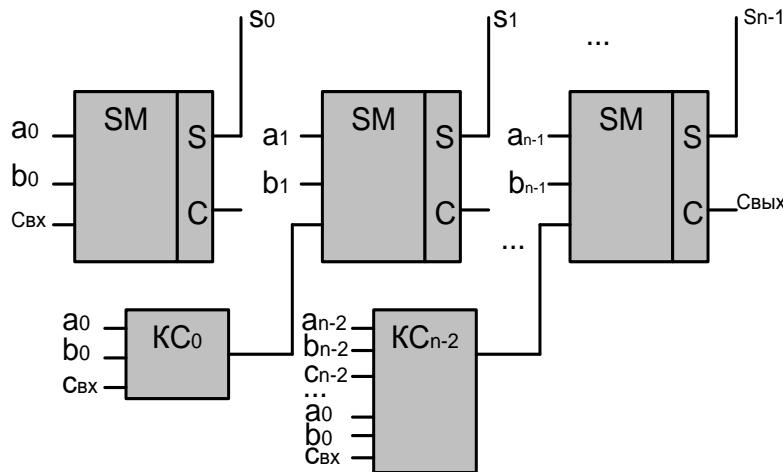
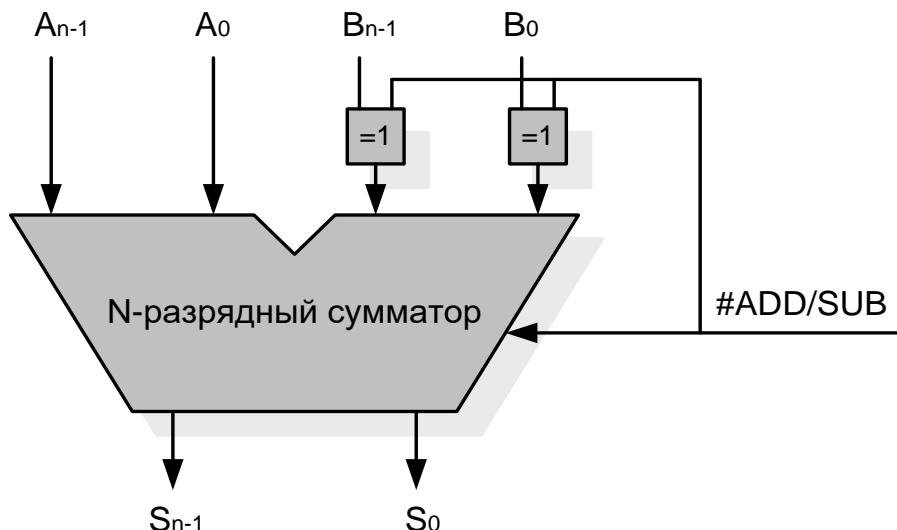


Схема параллельного сумматора с параллельным переносом



Устройство целочисленного сложения/вычитания



Устройства целочисленного умножения

Умножение сводится к последовательному формированию частных произведений и их сложению.

По способу формирования частных произведений:
умножение со старших разрядов множителя со сдвигом влево,
умножение с младших разрядов множителя со сдвигом вправо.

По способу накопления частных произведений: матричные
умножители, древовидные умножители.

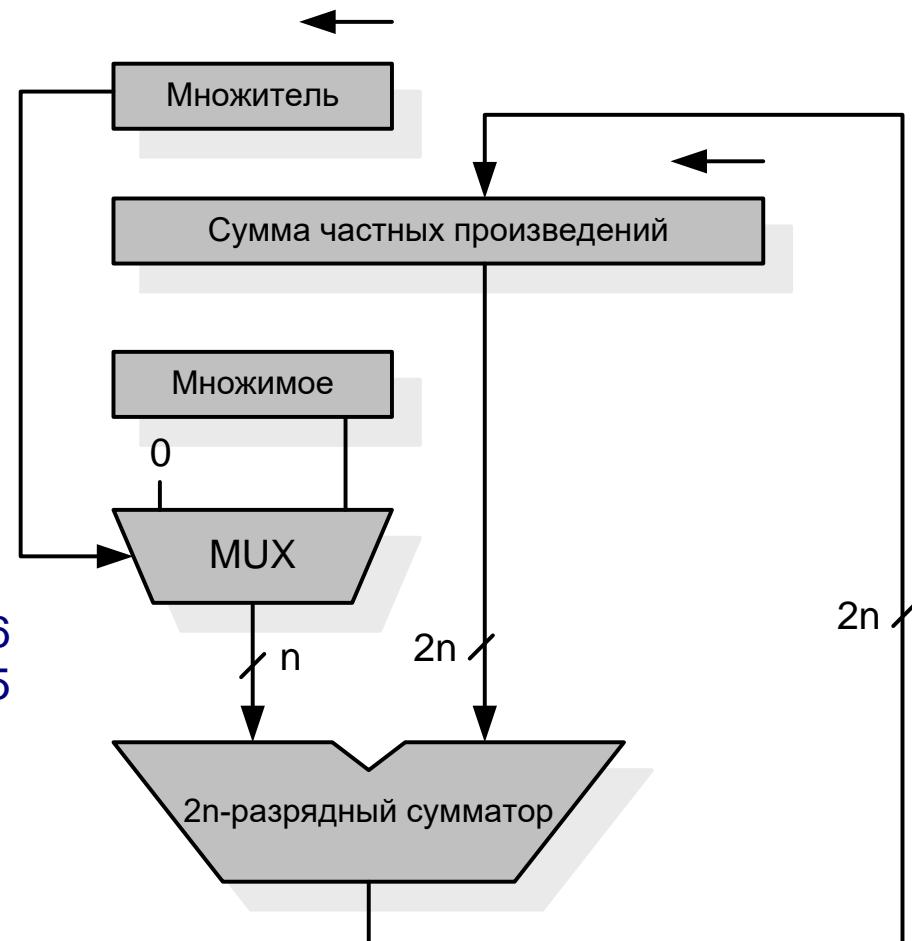
Способы ускорения работы устройств умножения:

- сокращение количества частных произведений;
- обработка нескольких разрядов множителя за такт;
- параллельное вычисление нескольких СЧП;
- конвейеризация умножителей.

Умножение со старших разрядов множителя со сдвигом влево

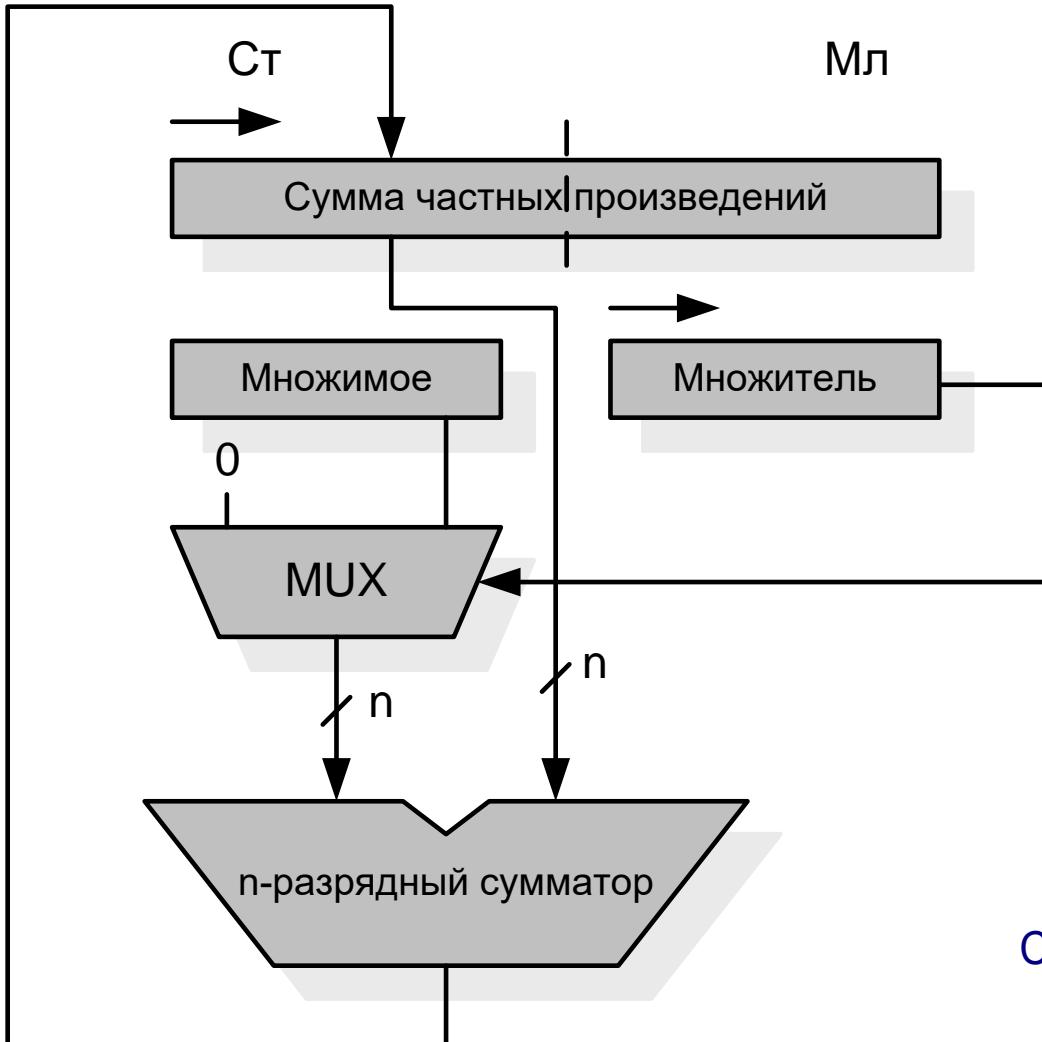
Старший разряд множителя определяет очередное частное произведение (ЧП), которое складывается с накопленной суммой частных произведений (СЧП). После этого СЧП и множитель сдвигаются на один разряд влево.

x	A	1	1	0
	B	1	0	1
ЧП0		1	1	0
СЧП0		1	1	0
<-СЧП0		1	1	0
ЧП1		0	0	0
СЧП1=СЧП0+ЧП1		1	1	0
<-СЧП1		1	1	0
ЧП2		1	1	0
СЧП2=СЧП1+ЧП2		1	1	1



(-) 2-п разрядный сумматор и шины данных.

Умножение с младших разрядов множителя со сдвигом вправо

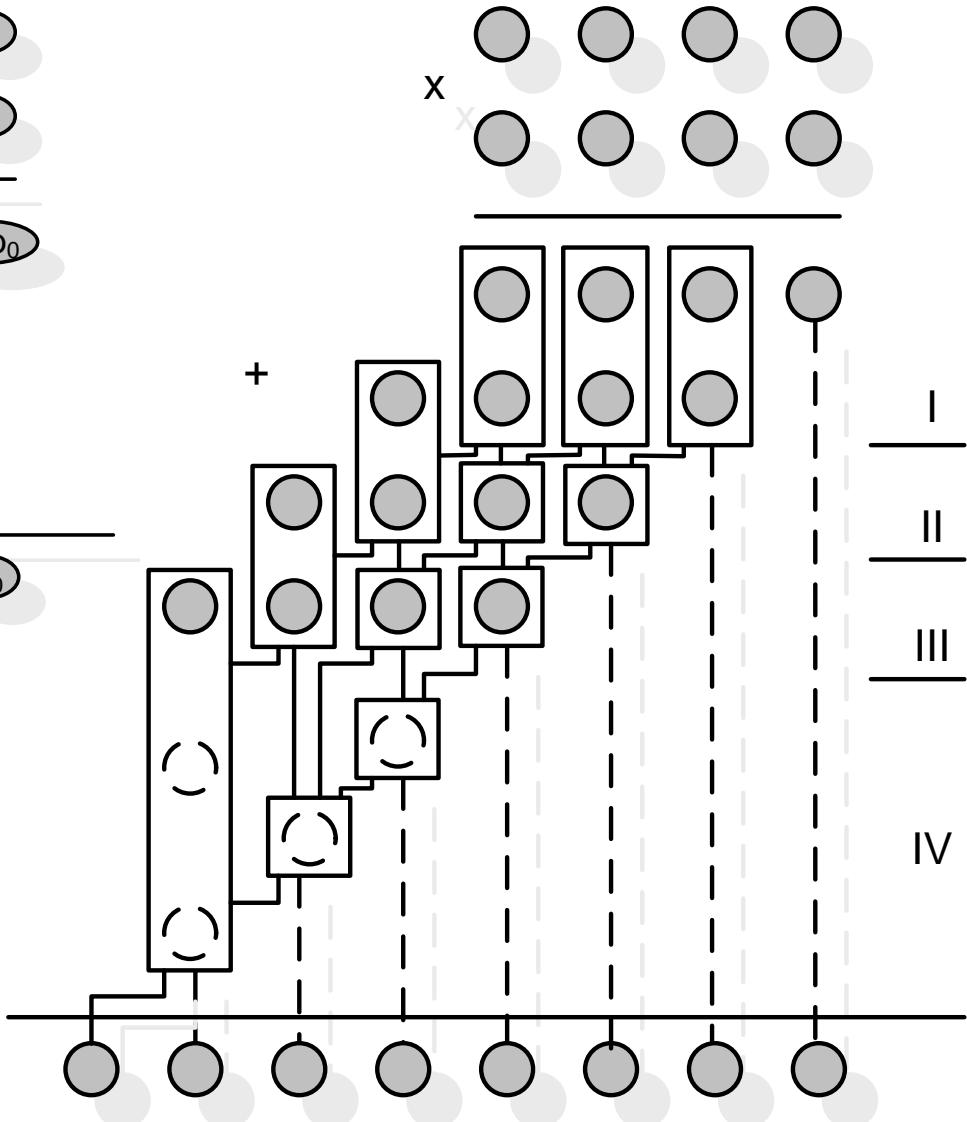
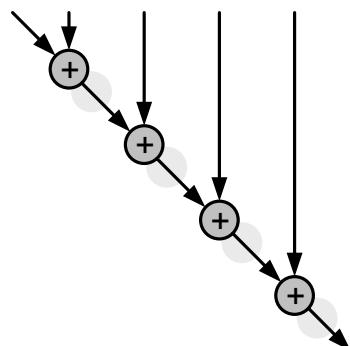
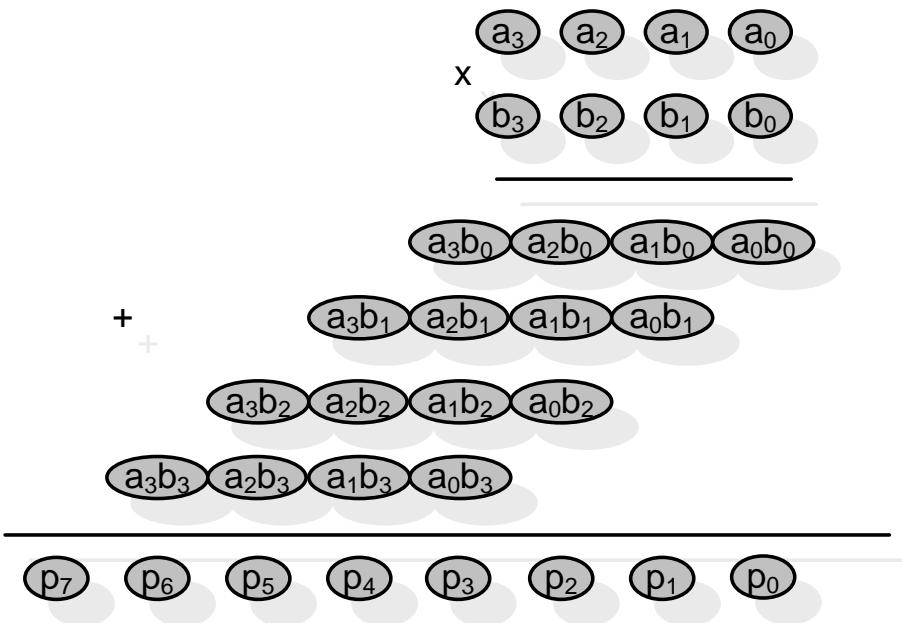


Младший разряд множителя определяет очередное частное произведение (ЧП), которое складывается с накопленной суммой частных произведений (СЧП). После этого СЧП и множитель сдвигаются на один разряд вправо.

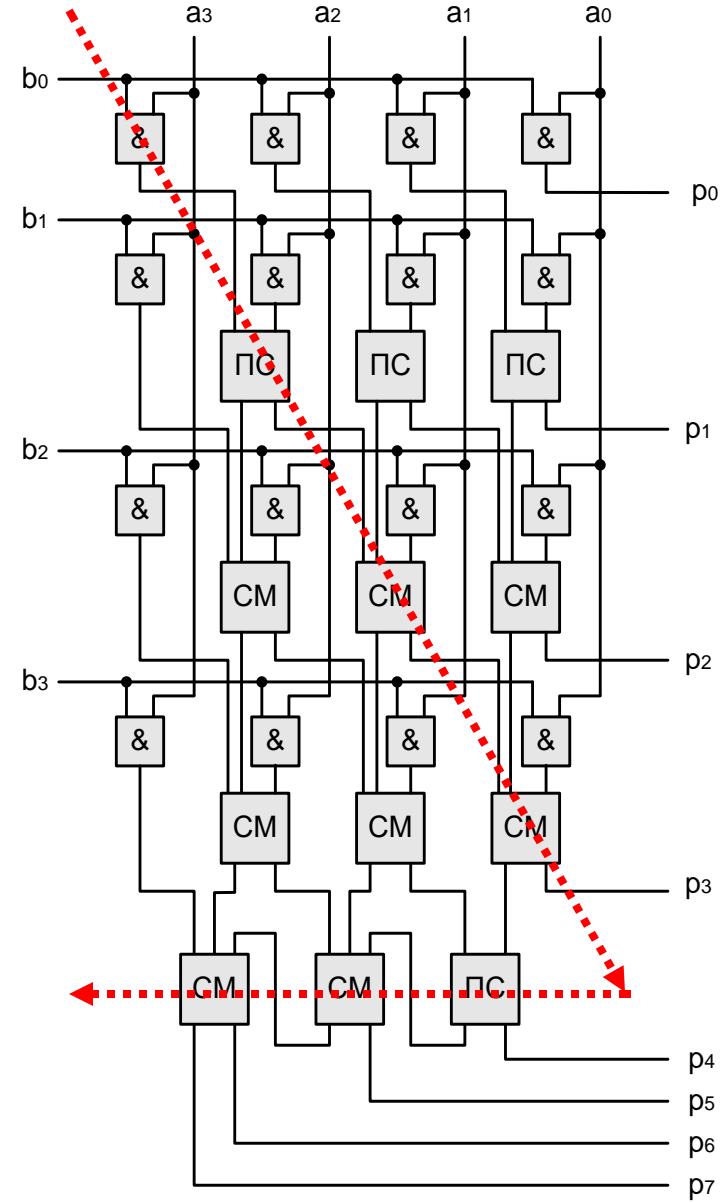
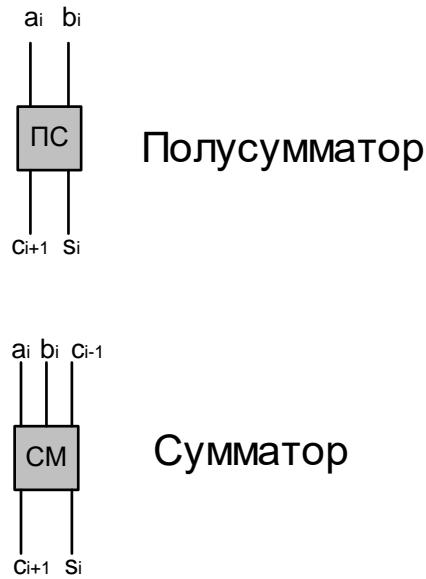
	A	1	1	0	6
x	B	1	0	1	5
ЧП0		1	1	0	
СЧП0		1	1	0	
СЧП0->		0	1	1	0
ЧП1		0	0	0	
СЧП1=СЧП0+ЧП1		0	1	1	0
СЧП1->		0	0	1	1
ЧП2		1	1	0	
СЧП2=СЧП1+ЧП2		1	1	1	0
					30

(+) n-разрядный сумматор и шины данных.
Организация ИУБ
ЭВМ

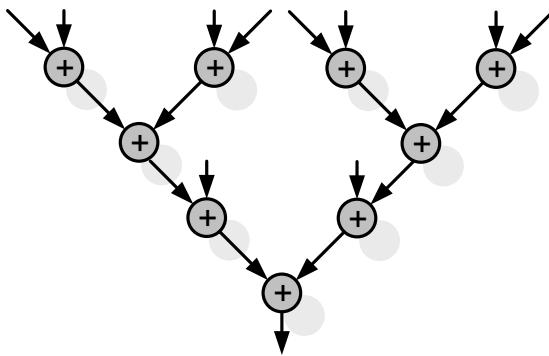
Матричные умножители



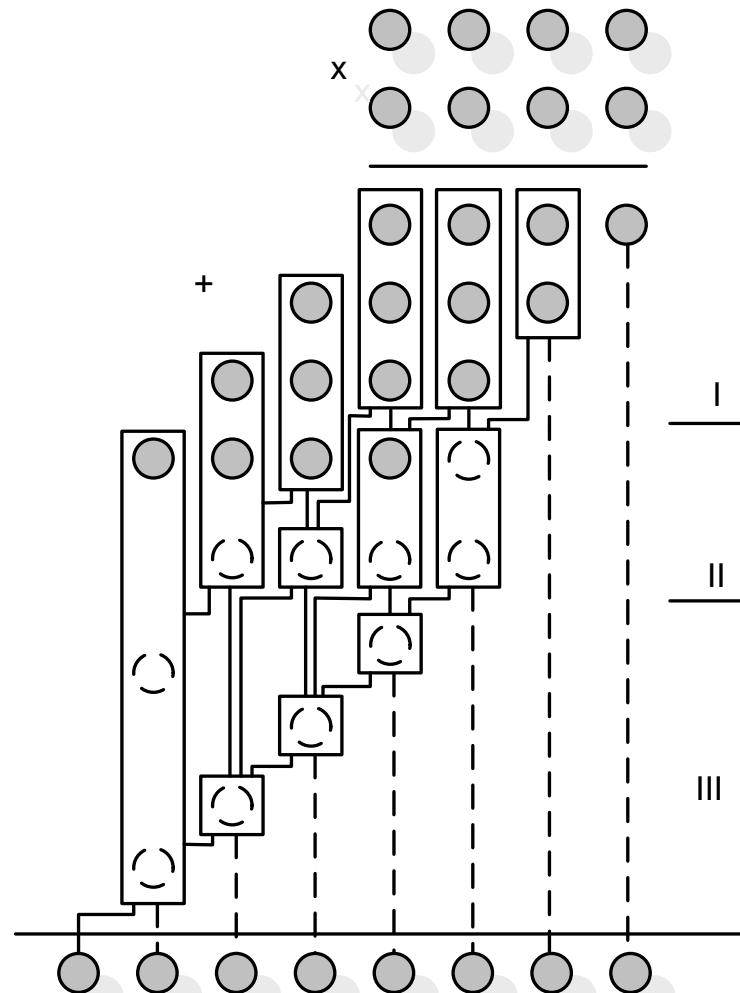
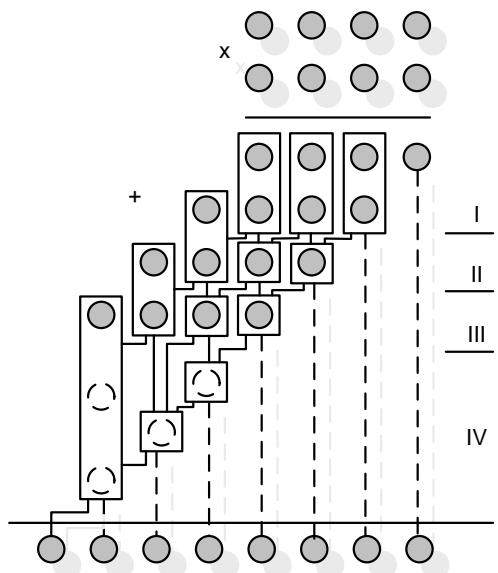
Матричные умножители



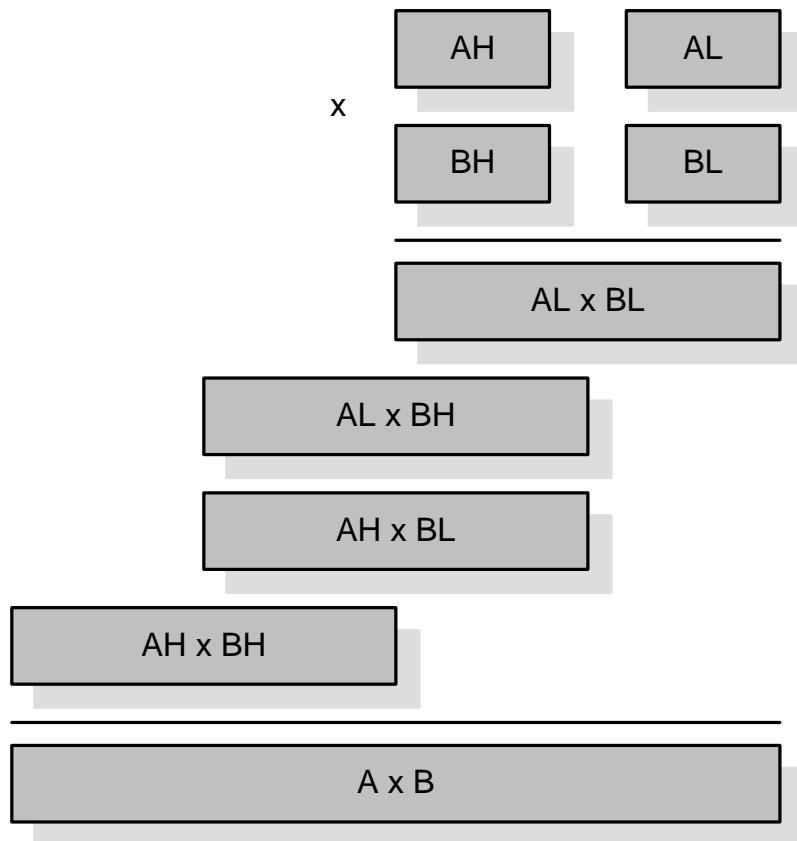
Древовидные умножители (схема Уоллеса)



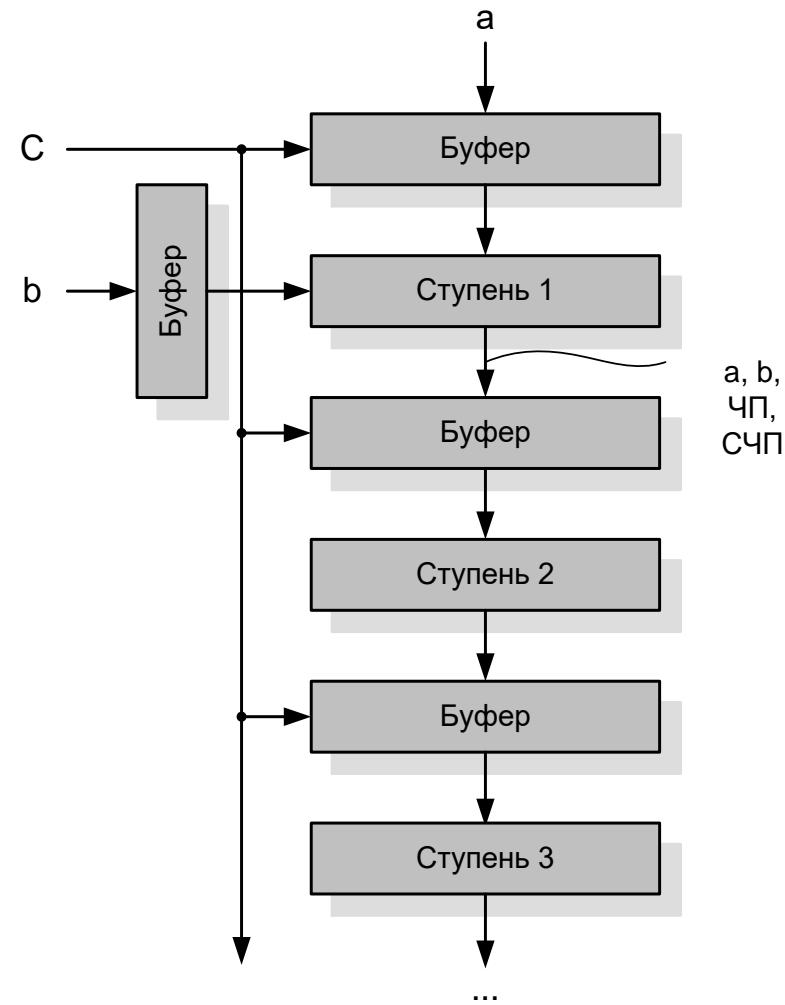
Матричные умножители



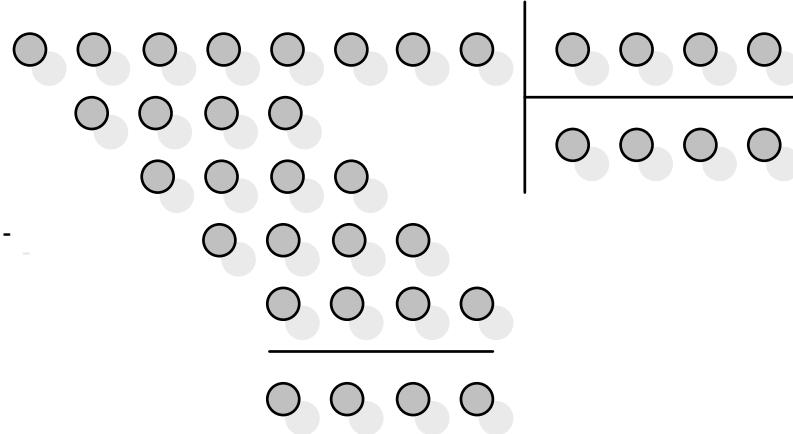
Наращивание размерности умножителей



Конвейеризация умножителей



Устройства целочисленного деления



Деление с восстановлением остатка

	17						
Делимое							
ЧО							
<-ЧО							
-Делитель							
ЧО>0							
<-ЧО							
-Делитель							
ЧО<0							
+Делитель							
Восст. ЧО							
<-ЧО							
-Делитель							
ЧО>0	0 0 0	0 1 0	0 0 0	1 0 0	0 1 1	3	
	0 1 0	0 0 0	0 0 1	0 1 0	1 0 1	5	
	0 1 1	0 0 1	0 1 0	0 1 0			
	0 0 0	0 1 0	0 1 0	1 0 0			
	0 1 1	0 0 1	0 1 0	0 1 0			
	1 1	1 1 1	1 1 1	1 0 0			
	0 1 1	0 1 1	0 1 1	0 1 0			
	0 0 0	0 1 0	0 1 0	1 0 0			
	0 0 0	1 0 1	1 0 1	0 0 0			
	0 1 1	0 1 1	0 1 1	0 0 0			

Алгоритм:

- 1) ЧО = Делимое;
- 2) ЧО = ЧО*2;
- 3) ЧО = ЧО – Делитель * 2^n ;
- 4) Если ЧО<0 то
Ч<-0,
ЧО = ЧО + Делитель * 2^n
иначе Ч<-1;
- 5) Если все цифры то конец
иначе пункт 2.

Деление без восстановления остатка

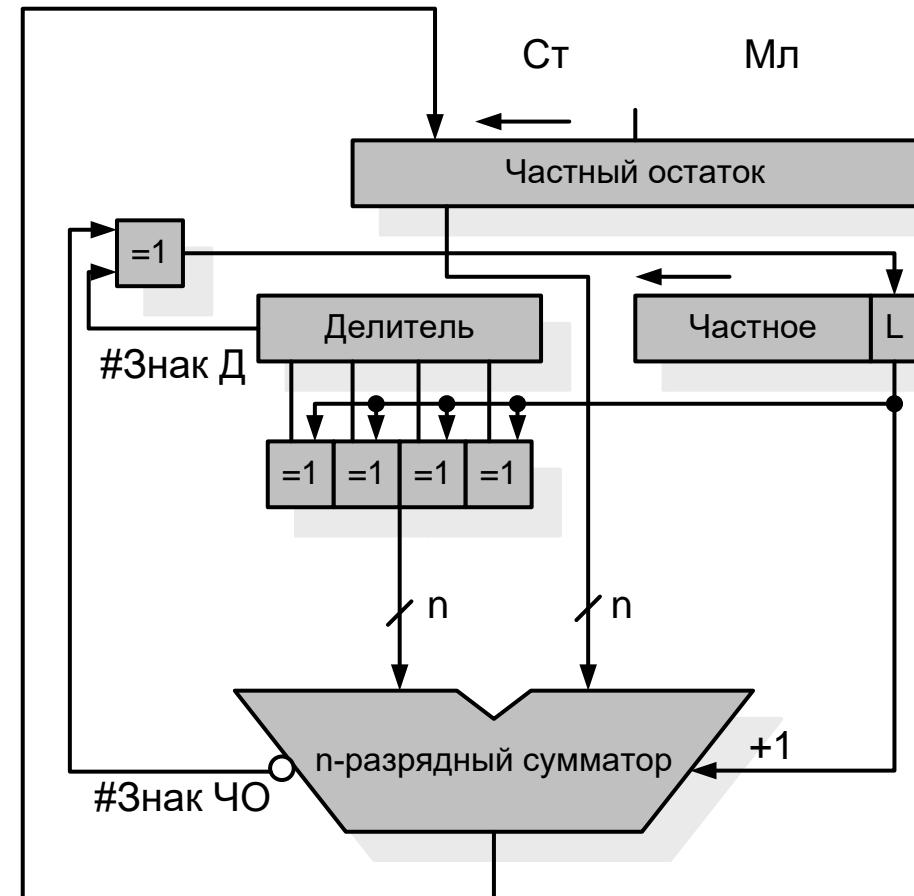
		17	
Делимое		0 1 0 0 0 1 0 1 1	3
ЧО		0 1 0 0 0 1 1 0 1	5
<-ЧО	-	0 1 0 0 0 1 0	
-Делитель		0 1 1	
ЧО>0		0 0 0 1 0 1 0	
<-ЧО	-	0 0 0 1 0 1 0 0	
-Делитель		0 1 1	
ЧО<0		1 1 1 1 1 1 1 0 0	
<-ЧО	+	1 1 1 1 1 1 0 0 0	
+Делитель		0 1 1	
ЧО>0		0 0 0 0 1 0 0 0 0	

Алгоритм:

- 1) ЧО = Делимое;
 - 2) ЧО = ЧО*2;
 - 3) ЧО = ЧО – Делитель * 2^n ;
 - 4) Если ЧО<0 то
 Ч<-0,
 ЧО = ЧО*2;
 ЧО = ЧО + Делитель * 2^n
- иначе
- Ч<-1;
ЧО = ЧО*2;
ЧО = ЧО – Делитель * 2^n

- 5) Если все цифры то конец, иначе пункт 4.

Схема АЛУ для целочисленного деления



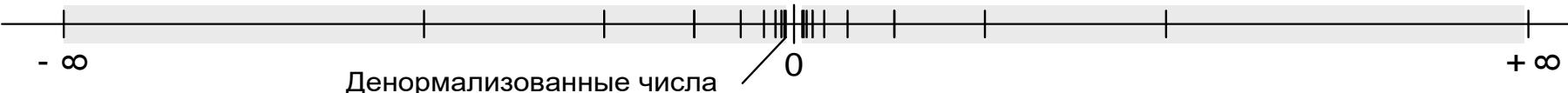
Форматы представления чисел с плавающей запятой (по стандарту IEEE 754 и 784).

Мантисса M числа представляется в нормализованном виде (старший разряд не сохраняется).

$$+ 1.0001 * 2^{+011} = 8.5 = \underset{\text{Знак}}{0} \underset{\text{Мантисса}}{10000010} \underset{\text{Порядок}}{00010\dots0} \underset{\text{Нормализованная мантисса}}{1}$$

Формат	Длина числа	Длина мантиссы	Длина порядка	Смещение порядка	Диапазон чисел
Короткий формат	32	24	8	+127	$10^{-38}..10^{+38}$
Длинный формат	64	53	11	+1023	$10^{-308}..10^{+308}$
Расширенный формат	80	64	15	+16383	$10^{-4932}..10^{+4932}$

Специальные числовые значения.



Переполнение

Потеря значимости

Переполнение

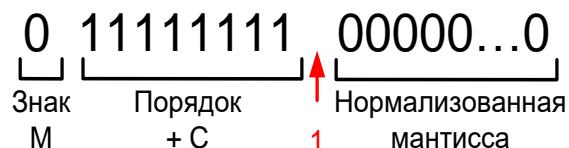
Числовые значения



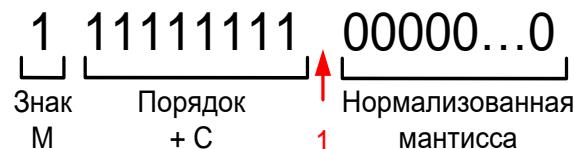
+Ноль



-Ноль
+Бесконечность

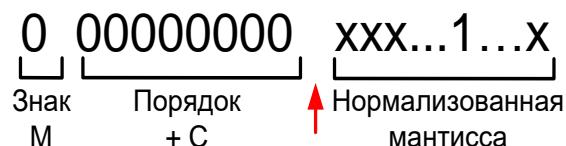


-Бесконечность

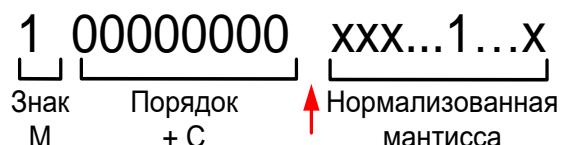


“Нечисла”

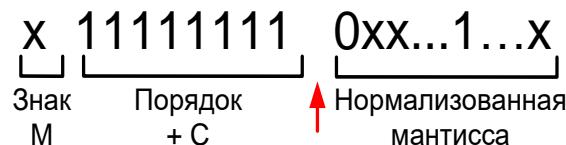
Денормализованное число >0



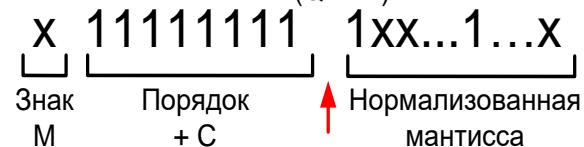
Денормализованное число <0



Нечисло (SNAN)

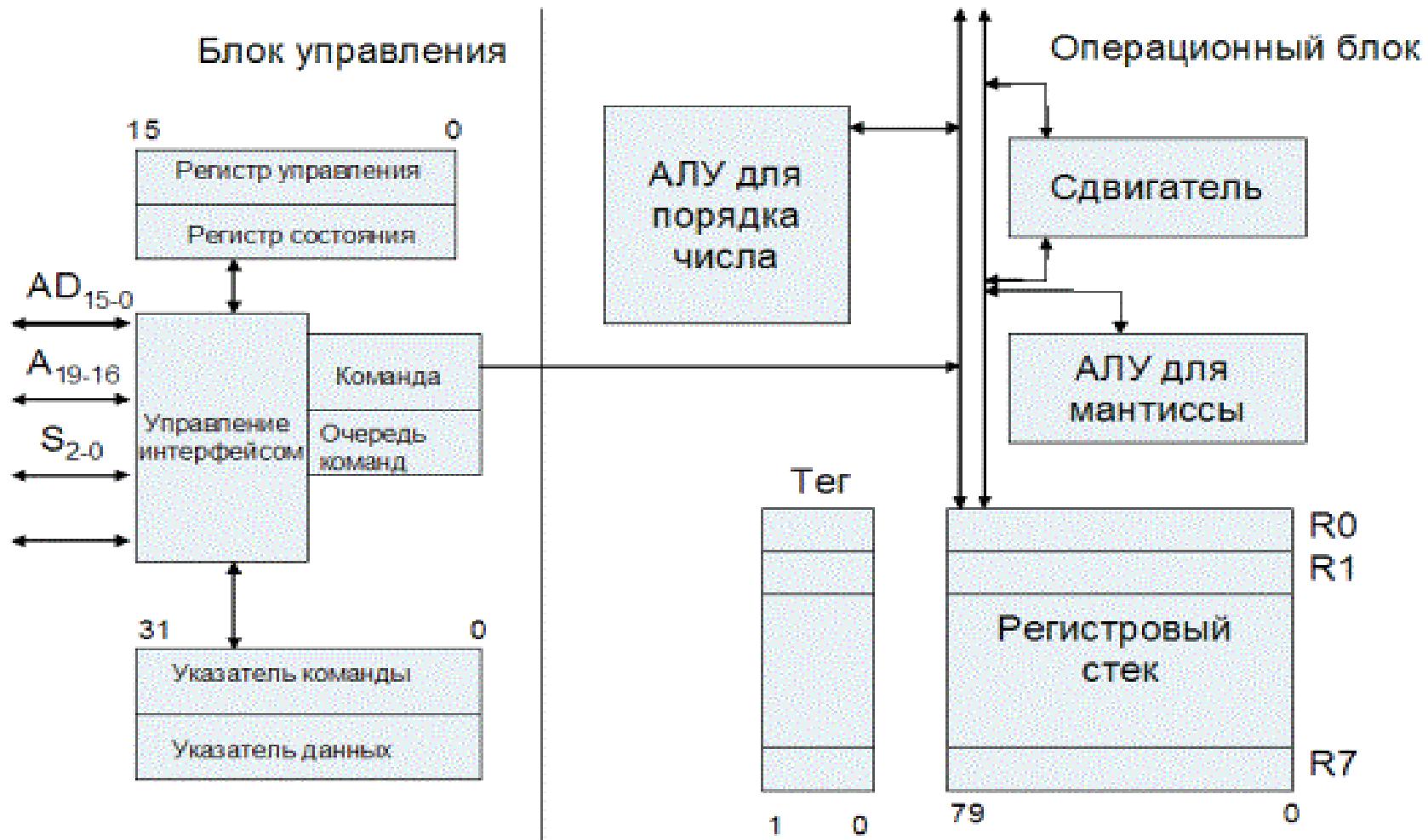


Нечисло (QNAN)



	Название	Ситуация	Реакция
	Денормализованный operand	Один из operandов или результат – ненормализованы	Продолжение операции
	Деление на ноль	Делитель равен нулю, а делимое не ноль и не бесконечность	Результат – бесконечность со знаком
	Переполнение	Результат операции не представим в указанном формате (слишком большой по модулю)	Результат – наибольшее число или бесконечность
	Антiperеполнение	Результат операции не представим в указанном формате (слишком маленький по модулю)	Результат – денормализованное число или ноль
	Точность	Результат операции не представим в указанном формате (мало цифр)	Округление

Структура FPU.



Операции над числами с плавающей запятой.

1. Подготовительный этап.

- Разделение упакованного ЧПЗ на группы М,П,З.
- Проверка на специальное числовое значение.

2. Выполнение операции.

- Приведение порядков.
- Определение знака результата.
- Определение мантиссы результата.
- Определение порядка результата.
- Проверка на переполнение, потери значимости мантиссы, потери значимости порядка, неточности, деления на 0.

3. Заключительный этап.

- Проверка на специальное числовое значение.
- Нормализация результата.
- Проверка на переполнение, потери значимости порядка.
- Упаковка полей З,П,М в ЧПЗ.

Организация операций сложения и вычитания над числами с плавающей запятой.

1. Подготовительный этап
2. Определение меньшего из двух порядков и проведение операции выравнивания порядков (сдвиг вправо на разность порядков).
3. Проверка на потерю значимости одного операнда (неточность).
4. Определение результирующего порядка как максимума.
5. Сложение мантисс и определение знака результата.
6. Проверка на переполнение мантиссы. Если да, то сдвигаем мантиссу вправо и увеличиваем порядок на 1.
7. Проверка на переполнение порядка.
8. Заключительный этап.

Организация операций умножения чисел с плавающей запятой.

1. Подготовительный этап
2. Проверка ($M_1=0$ или $M_2=0$). Если да, то $R=0$.
3. Определение порядка результата: $Pr = P_1 + P_2 - C$.
4. Проверка на переполнение порядка.
5. Определение мантиссы результата: $Mr = M_1 * M_2$.
6. Определение знака результата.
7. Заключительный этап.

Организация операций деления чисел с плавающей запятой.

1. Подготовительный этап
2. Проверка ($M1=0$ или $M2=0$). Если деление на ноль, то +/-бесконечность или ошибка.
3. Определение порядка результата: $Pr = P1 - P2 + C$.
4. Проверка на переполнение порядка.
5. Определение мантиссы результата: $Mp = M1 * (1/M2)$.
6. Определение знака результата.
7. Заключительный этап.

TABLE 6-5 Floating-Point Multiplication

MULTIPLICATION Instruction FMUL $rs_1, rs_2 / rs_2, rs_3 \rightarrow rd$	Result from the operation includes one or more of the following:			
	Masked Exception, TEM = 0		Enabled Exception, TEM = 1	
	Destination Register Written (rd)	Flag(s)	Destination Register Written (rd)	Flag(s), Trap
+0, [+0]+Normal]	+0	None set.	+0	None set.
+0, [-0]-Normal]	-0	None set.	-0	None set.
-0, [+0]+Normal]	-0	None set.	-0	None set.
-0, [-0]-Normal]	+0	None set.	+0	None set.
+0, +Infinity	QNaN	Asserts nvc, nva.	No	Asserts nvc. IEEE trap ¹ enabled.
+0, -Infinity	QNaN	Asserts nvc, nva.	No	Asserts nvc. IEEE trap enabled.
-0, +Infinity	QNaN	Asserts nvc, nva.	No	Asserts nvc. IEEE trap enabled.
-0, -Infinity	QNaN	Asserts nvc, nva.	No	Asserts nvc. IEEE trap enabled.
±Normal, ±Normal	Can underflow/ overflow. See 6.5.		Can underflow/ overflow. See 6.5.	
[+Normal]+Infinity], +Infinity	+Infinity	None set.	+Infinity	None set.
[+Normal]+Infinity], -Infinity	-Infinity	None set.	-Infinity	None set.
[-Normal]-Infinity], +Infinity	-Infinity	None set.	-Infinity	None set.
[-Normal]-Infinity], -Infinity	+Infinity	None set.	+Infinity	None set.

1.IEEE trap means fp_exception_IEEE_754.

TABLE 6-3 Floating-Point Addition

ADDITION Instruction	Result from the operation includes one or more of the following:			
	Masked Exception, TEM = 0		Enabled Exception, TEM = 1	
	Destination Register Written (rd)	Flag(s)	Destination Register Written (rd)	Flag(s), Trap
+0, +0	+0	None set.	+0	None set.
+0, -0	+0 (FSR.RD = 0,1,2) -0 (FSR.RD = 3)	None set.	+0 (FSR.RD = 0,1,2) -0 (FSR.RD = 3)	None set.
-0, -0	-0	None set.	-0	None set.
±0, +Normal	+Normal	None set.	+Normal	None set.
±0, -Normal	-Normal	None set.	-Normal	None set.
±0, +Infinity	+Infinity	None set.	+Infinity	None set.
±0, -Infinity	-Infinity	None set.	-Infinity	None set.
±Normal, +Infinity	+Infinity	Asserts ofc, ofa, nvc, nva.	No	Asserts ofc, nvc. IEEE trap ¹ enabled.
±Normal, -Infinity	-Infinity	Asserts ofc, ofa, nvc, nva.	No	Asserts ofc, nvc. IEEE trap enabled.
+Normal, +Normal	Can overflow. See 6.5.3.		Can overflow. See 6.5.3.	
+Normal, -Normal	±Normal		Normal	
-Normal, +Normal	±Normal		Normal	
-Normal, -Normal	Can underflow. See 6.5.4.		Can underflow. See 6.5.4.	
+Infinity, +Infinity	+Infinity	None set.	+Infinity	None set.
+Infinity, -Infinity	QNaN	Asserts nvc, nva.	No	Asserts nvc. IEEE trap enabled.
-Infinity, +Infinity	QNaN	Asserts nvc, nva.	No	Asserts nvc. IEEE trap enabled.
-Infinity, -Infinity	-Infinity	None set.	-Infinity	None set.

1.IEEE trap means *fp_exception_IEEE_754*.

TABLE 6-6 Floating-Point Division

DIVISION Instruction $rs_1\ rs_2$	Result from the operation includes one or more of the following:			
	Masked Exception, TEM = 0		Enabled Exception, TEM = 1	
	Destination Register Written (rd)	Flag(s)	Destination Register Written (rd)	Flag(s), Trap
$\pm 0, \pm 0$	sign=0, expo=111...111, frac=111...111 (QNaN)	Asserts nvc, nva.	No	Asserts nvc. IEEE trap ¹ enabled.
$\pm 0, \pm$ Normal	± 0	None set.	± 0	None set.
$\pm 0, \pm$ Infinity	± 0	None set.	± 0	None set.
+Normal, +0	+Infinity	Asserts nvc, nva.	No	Asserts dzc, nvc. IEEE trap enabled.
+Normal, -0	-Infinity	Asserts nvc, nva.	No	Asserts dzc, nvc. IEEE trap enabled.
-Normal, +0	-Infinity	Asserts nvc, nva.	No	Asserts dzc, nvc. IEEE trap enabled.
-Normal, -0	+Infinity	Asserts nvc, nva.	No	Asserts dzc, nvc. IEEE trap enabled.
\pm Normal, \pm Normal	Can underflow/overflow. See 6.5.		Can underflow/overflow. See 6.5.	
\pm Infinity, \pm Infinity	QNaN	Asserts nvc, nva.	No	Asserts nvc. IEEE trap enabled.
+Infinity, +Normal	+Infinity	None set.	+Infinity	None set.
+Infinity, -Normal	-Infinity	None set.	-Infinity	None set.
-Infinity, +Normal	-Infinity	None set.	-Infinity	None set.
-Infinity, -Normal	+Infinity	None set.	+Infinity	None set.

1.IEEE trap means *fp_exception_IEEE_754*.

Устройства выполнения векторных операций (Эльбрус1,Intel,AMD,Sun,IBM,MIPS).

Устройство выполнения целочисленных MMX операций (MultiMedia eXtensions, Intel) и SSE операций (Streaming SIMD Extension) предназначены для ускорения приложений, ориентированных на выполнение однотипных действий с большими массивами целочисленных и вещественных данных. С данными такого типа обычно работают мультимедийные, графические и коммуникационные программы.

Операнды MMX и SSE операций упакованы в группы по 32,64,80,128 разрядов. Выполнение арифметических операций над операндами группы выполняются параллельно.

Технология SIMD (ИТМ и ВТ, Эльбрус 1) 1978 год

Технология MMX (Intel Pentium MMX, Intel P6, ...) 1992 год

Технология SSE (Intel P6, Intel NetBurst, ...)

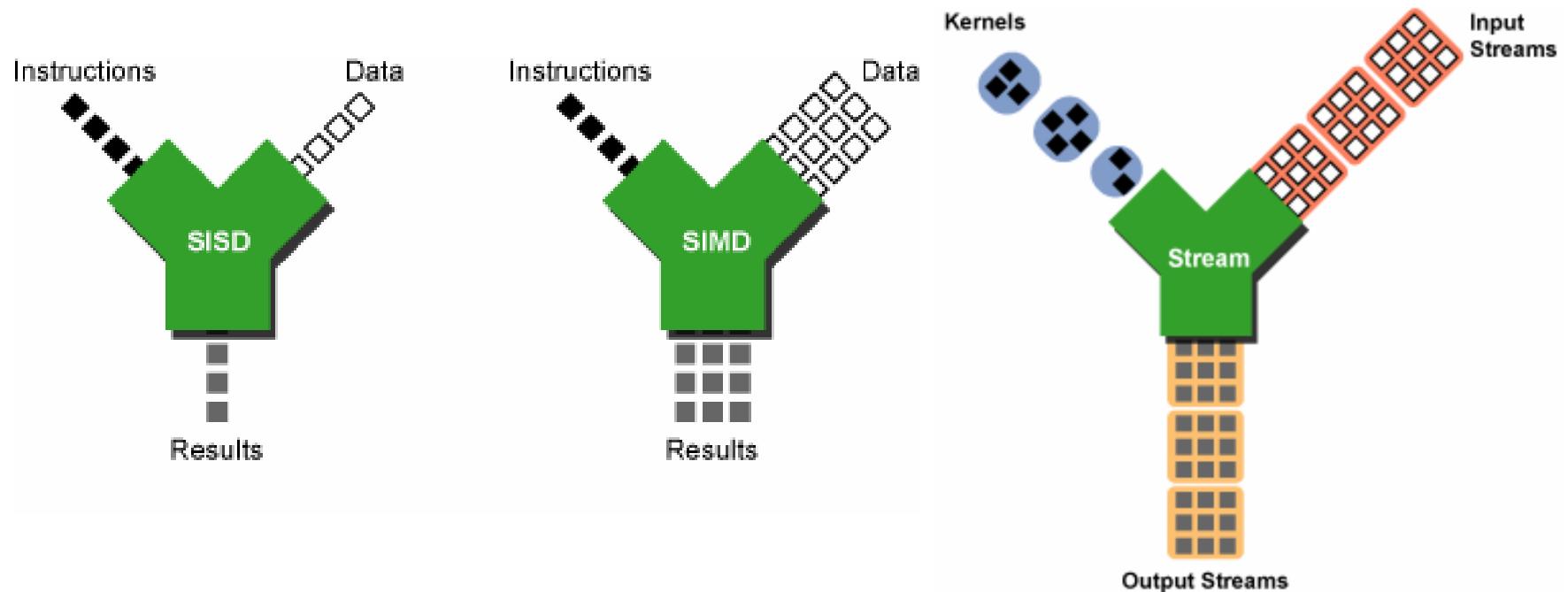
Технология 3DNow (AMD K6, ...)

Технология AltiVec (IBM PowerPC)

Технология VIS (Sun UltraSPARC II)

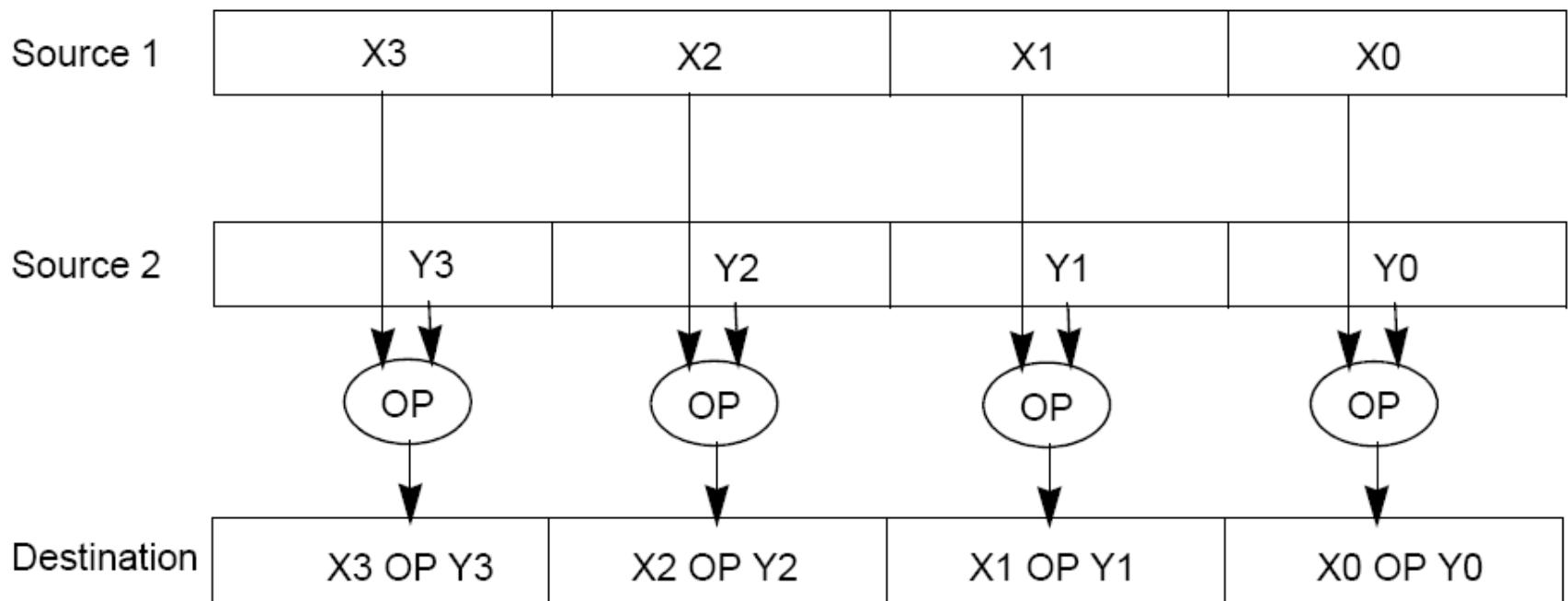
Технология ASE (MIPS 24KE, 74K)

SISD/SIMD/Streaming



SIMD

- SIMD (single instruction multiple data)
обеспечивает выполнение одной команды сразу над несколькими operandами
- **PADDW MM0 , MM1**



IA-32 SIMD расширение

MMX (Multimedia Ex~~tension~~) была впервые реализована в 1996 (Pentium MMX и Pentium II).

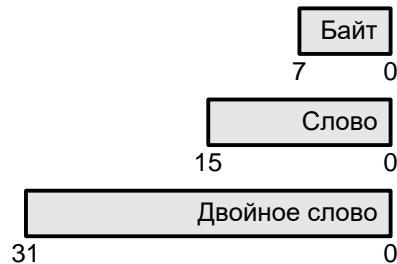
SSE (Streaming SExtension) был включен в состав Pentium III.

SSE2 был включен в Pentium 4.

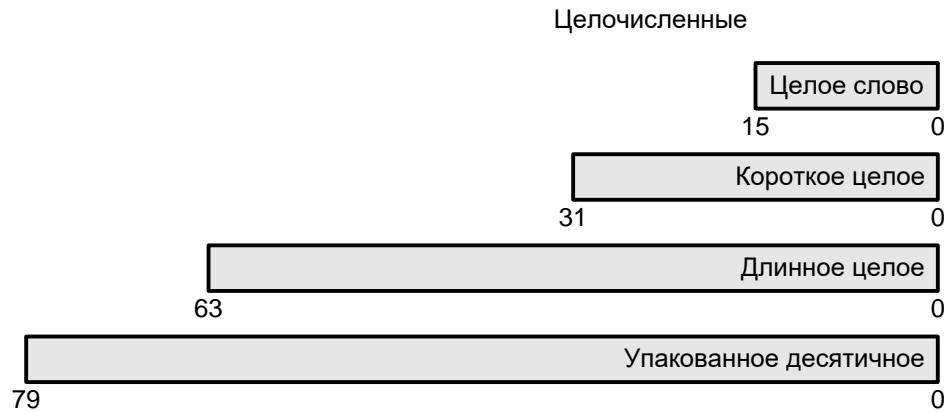
SSE3 был включен в Pentium 4 с поддержкой Hyperthreading.

Форматы чисел в микропроцессорах Intel, AMD

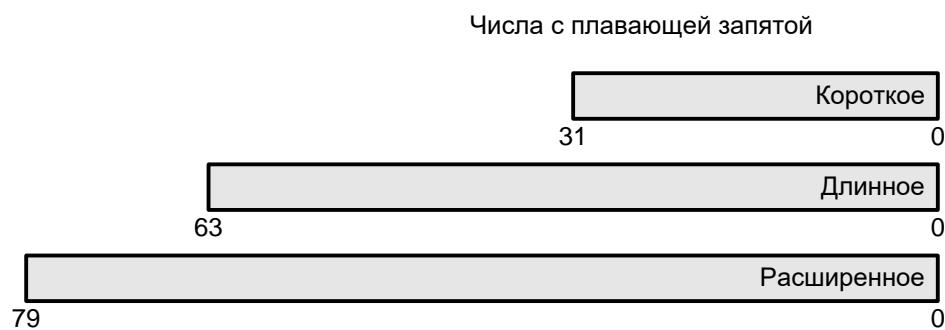
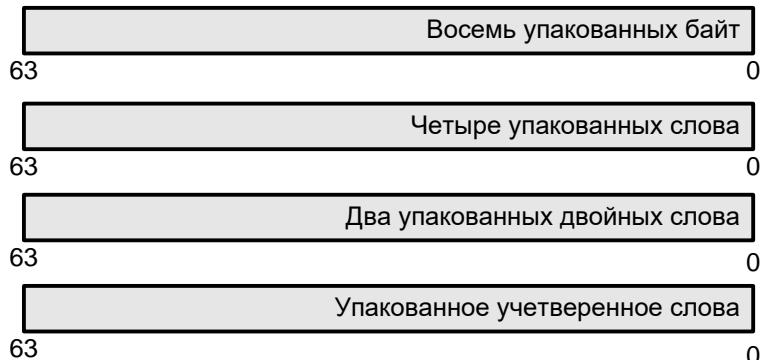
Форматы целочисленного АЛУ



Форматы блока обработки ЧПЗ



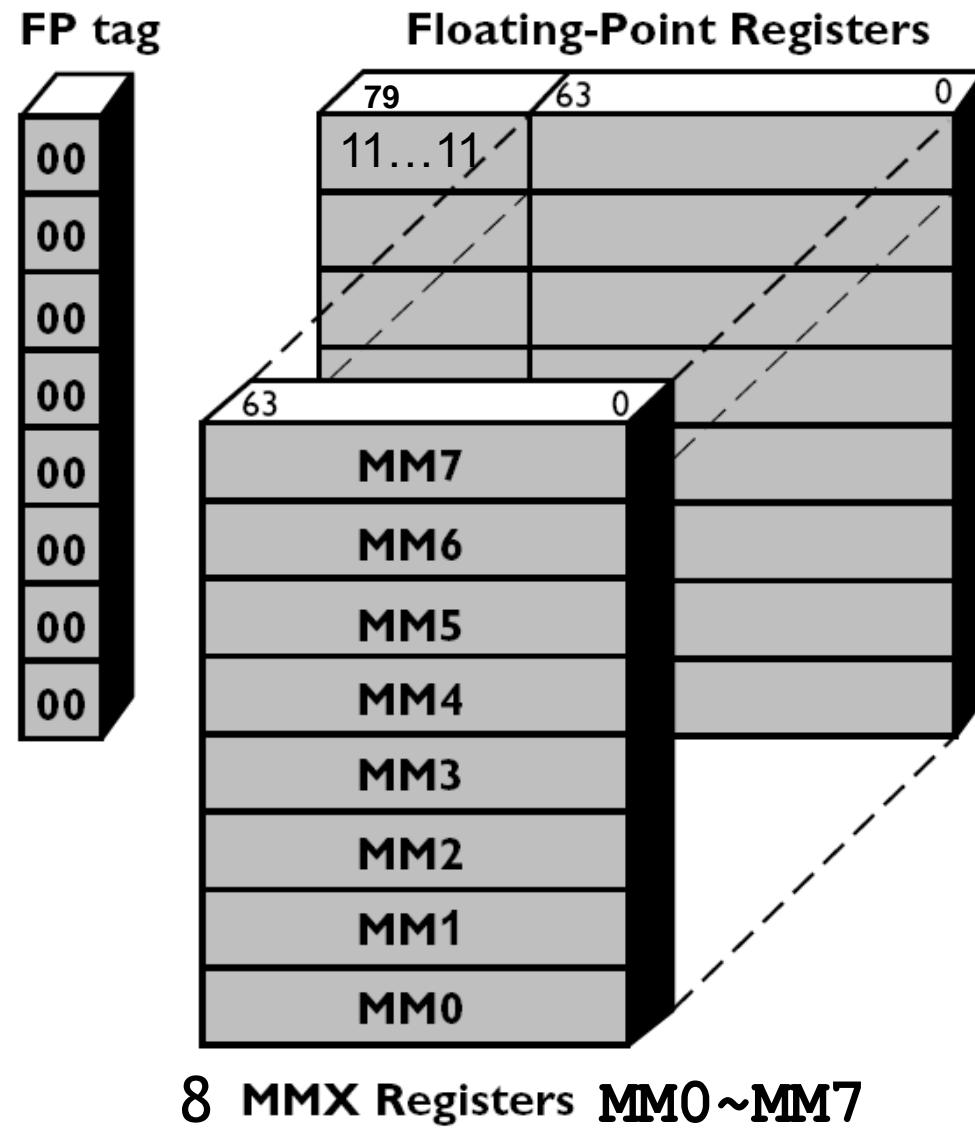
Форматы MMX



Форматы SSE



MMX реализация в IA



ММХ команды

Category		Wraparound	Signed Saturation	Unsigned Saturation
Arithmetic	Addition	PADDB, PADDW, PADDD	PADDSB, PADDSW	PADDUSB, PADDUSW
	Subtraction	PSUBB, PSUBW, PSUBD	PSUBSB, PSUBSW	PSUBUSB, PSUBUSW
	Multiplication Multiply and Add	PMULL, PMULH PMADD		
Comparison	Compare for Equal	PCMPEQB, PCMPEQW, PCMPEQD		
	Compare for Greater Than	PCMPGTPB, PCMPGTPW, PCMPGTPD		
Conversion	Pack		PACKSSWB, PACKSSDW	PACKUSWB
Unpack	Unpack High	PUNPCKHBW, PUNPCKHWD, PUNPCKHDQ		
	Unpack Low	PUNPCKLBW, PUNPCKLWD, PUNPCKLDQ		

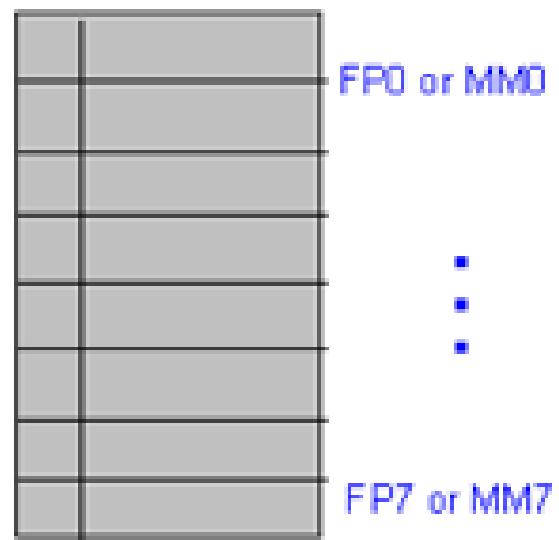
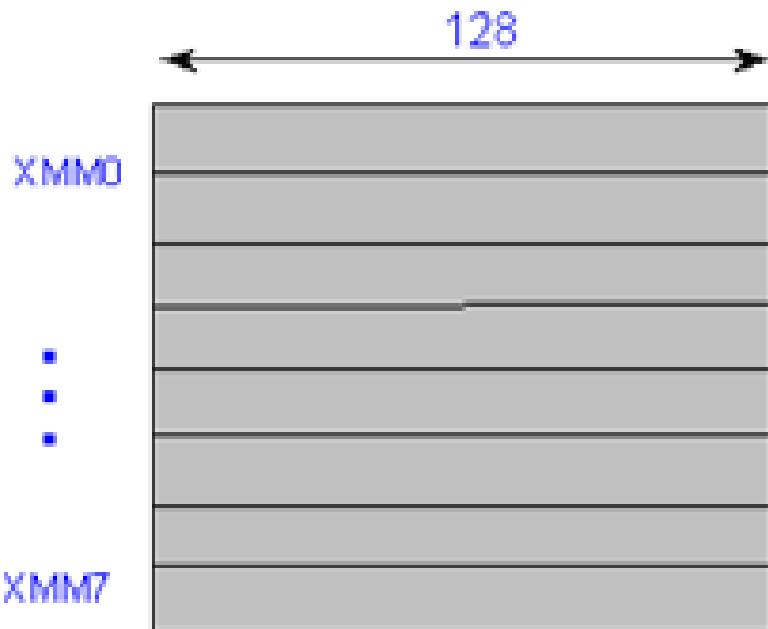
MMX команды

		Packed	Full Quadword
Logical	And And Not Or Exclusive OR		PAND PANDN POR PXOR
Shift	Shift Left Logical Shift Right Logical Shift Right Arithmetic	PSLLW, PSLLD PSRLW, PSRLD PSRAW, PSRAD	PSLLQ PSRLQ
		Doubleword Transfers	Quadword Transfers
Data Transfer	Register to Register Load from Memory Store to Memory	MOVD MOVD MOVD	MOVQ MOVQ MOVQ
Empty MMX State		EMMS	

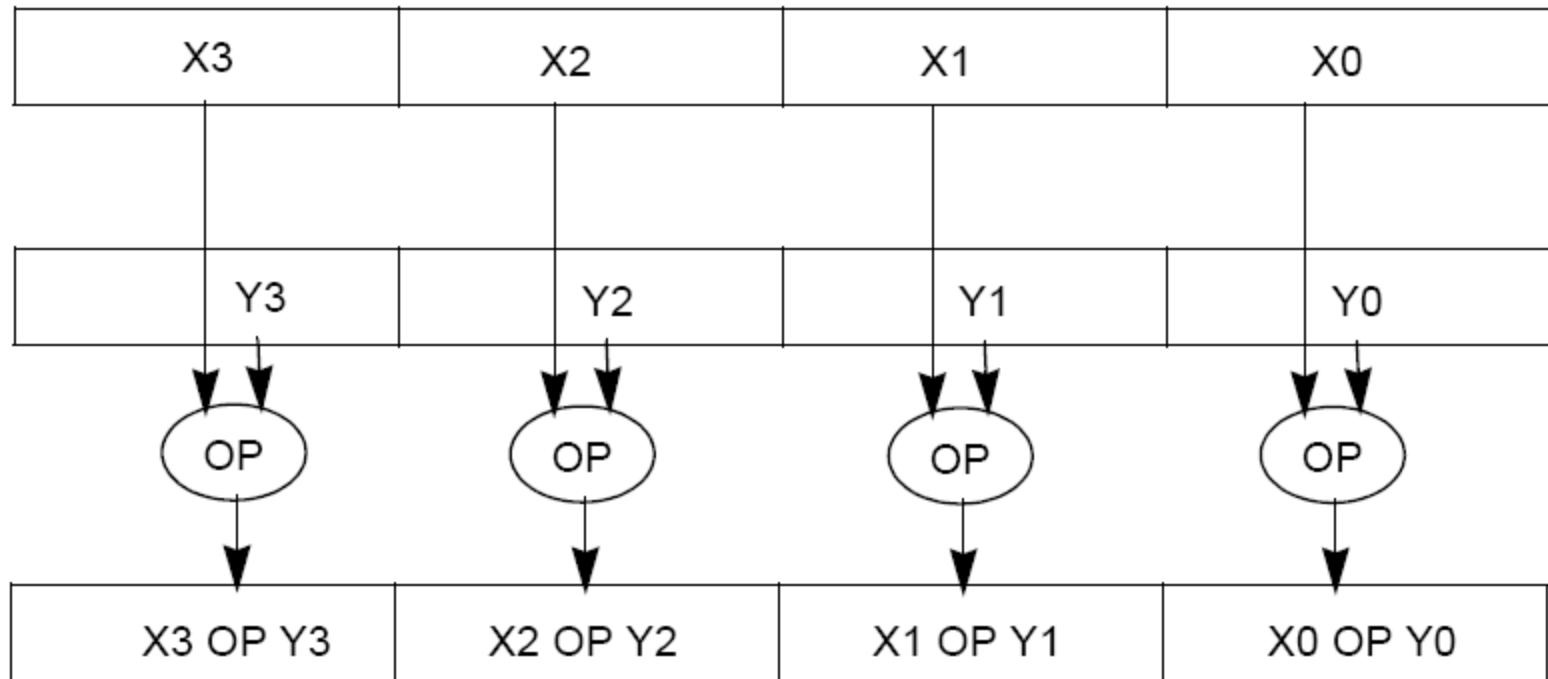
SSE регистры

Internet SSE
(Scalar/packed SIMD-SP)

MMX/x87
(64-bit Integer, x87)

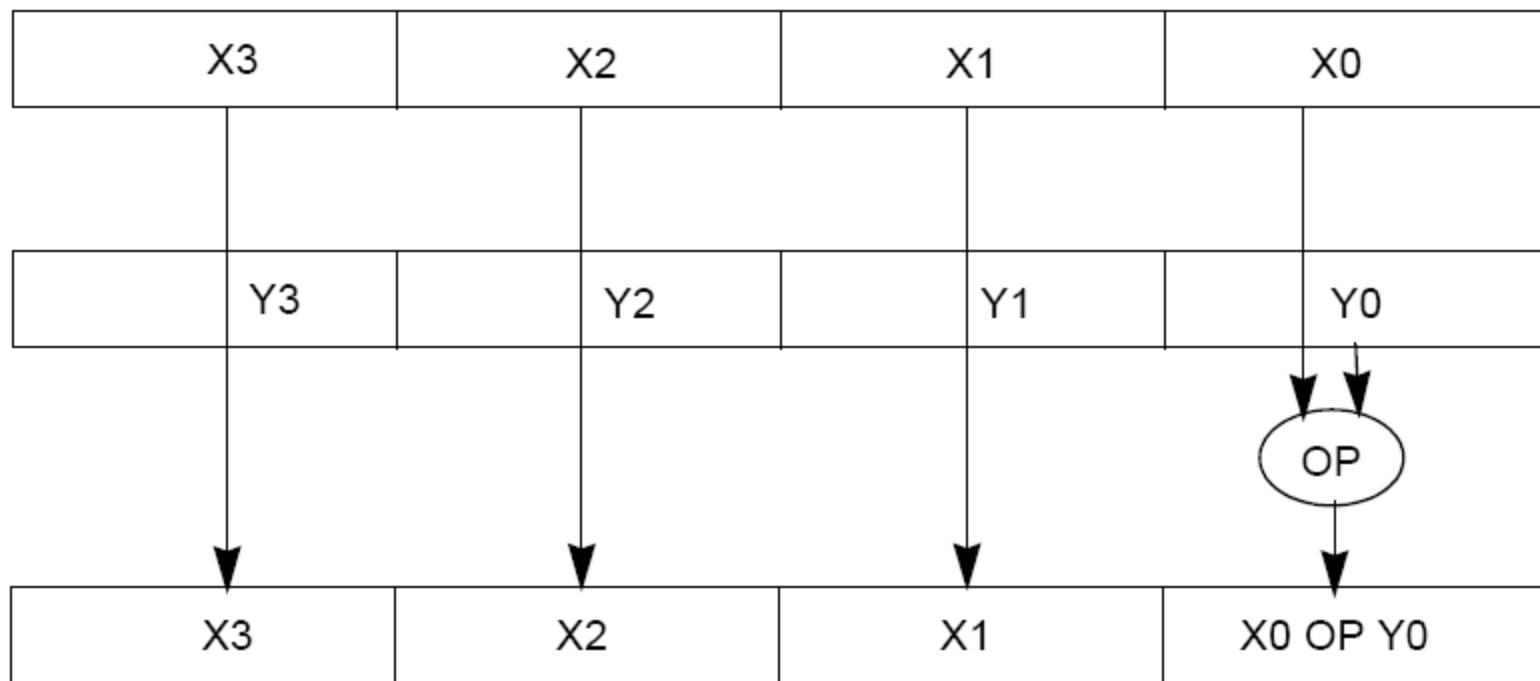


Пакетные операции SSE (packed)



- **ADDPS/SUBPS**: пакетная операция одинарной точности

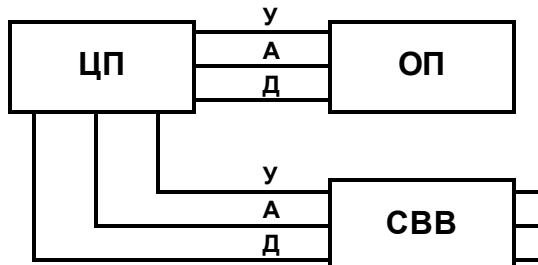
SSE скалярная операция



- **ADDSS/SUBSS:** скалярная операция одинарной точности

V. Организация ввода-вывода

Совокупность технических и программных средств, обеспечивающих обмен данными между центральным процессором и внешними устройствами называется системой ввода вывода.

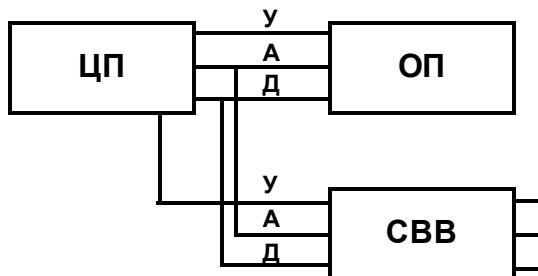


Шины полностью независимы.

(+) Обмен можно осуществлять параллельно.

(+) Каждую шину можно оптимизировать независимо.

(-) Увеличивается количество выводов ЦП.

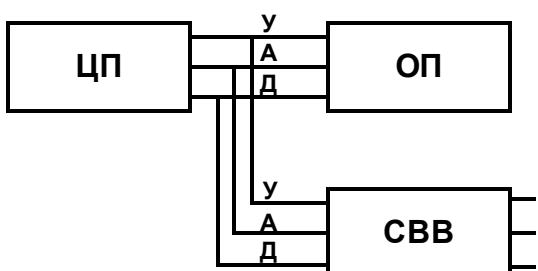


Совместно используются ШД и ША. ШУ независимы.

(+) Управление обменом выполняется параллельно.

(+) Экономично используются выводы.

(-) Одновременный обмен данными невозможен.



Совместно используются ШД и ША и ШУ.

(+) Минимальные аппаратные затраты.

(-) Одновременная работа ЦП с СВВ и с ОП невозможна.

Способы адресации внешних устройств

Для выбора конкретного ВУ используются адреса.

Адресное пространство ОП и СВВ может быть совмещенным и раздельным.

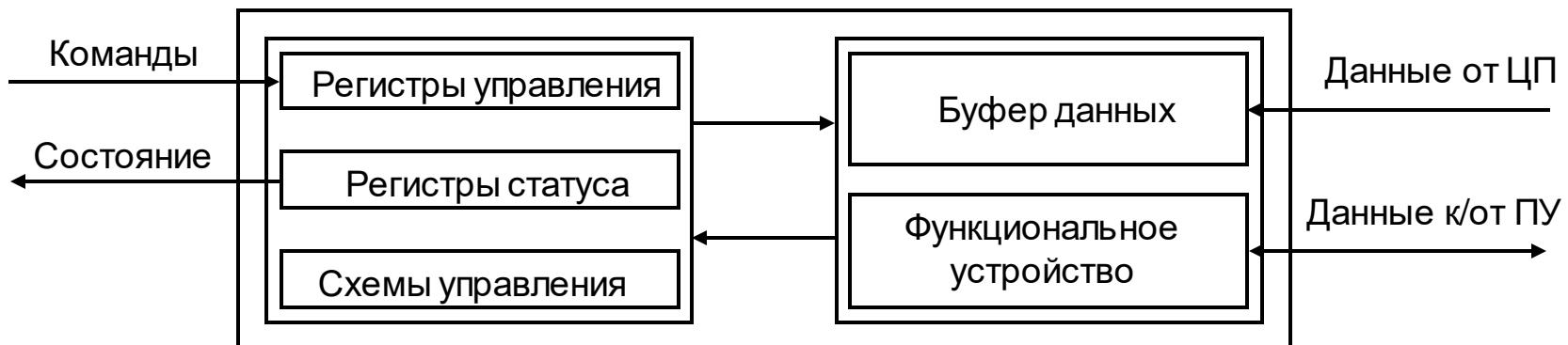
При совмешенном адресном пространстве обращение за чтением и записью в некоторый диапазон адресов приводит к обращению в СВВ.

- (+) Возможное количество ВУ может быть велико. Нет жестких ограничений на объем адресуемых портов ввода вывода (гибкость СВВ).
- (+) Команды для доступа к внешним устройствам не отличаются от команд доступа к ОП.
- (-) Потребность в дополнительных средствах декодирования адресов СВВ.
- (-) Сложность организации виртуальной памяти и поддержки когерентности кэш-памяти.
- (-) Непредсказуемость времени выполнения простых команд.
- (-) Сокращение адресного пространства ОП.

При раздельном адресном пространстве используются отдельные команды при обращении к ОП и СВВ.

- (+) Команды ввода/вывода короткие.
 - (+) Легкость дешифрации запросов к СВВ для ЦП.
 - (+/-) Изменение СВВ не затрагивает память
 - (-) Малое количество способов адресации и обработки информации (Аккумулятор - СВВ).
 - (-) Сложность работы с быстродействующими устройствами и устройствами с большим количеством адресуемой памяти.
- Устройство вычислительной машины, осуществляющее связь ЦП и ПУ называется модулем ввода-вывода.

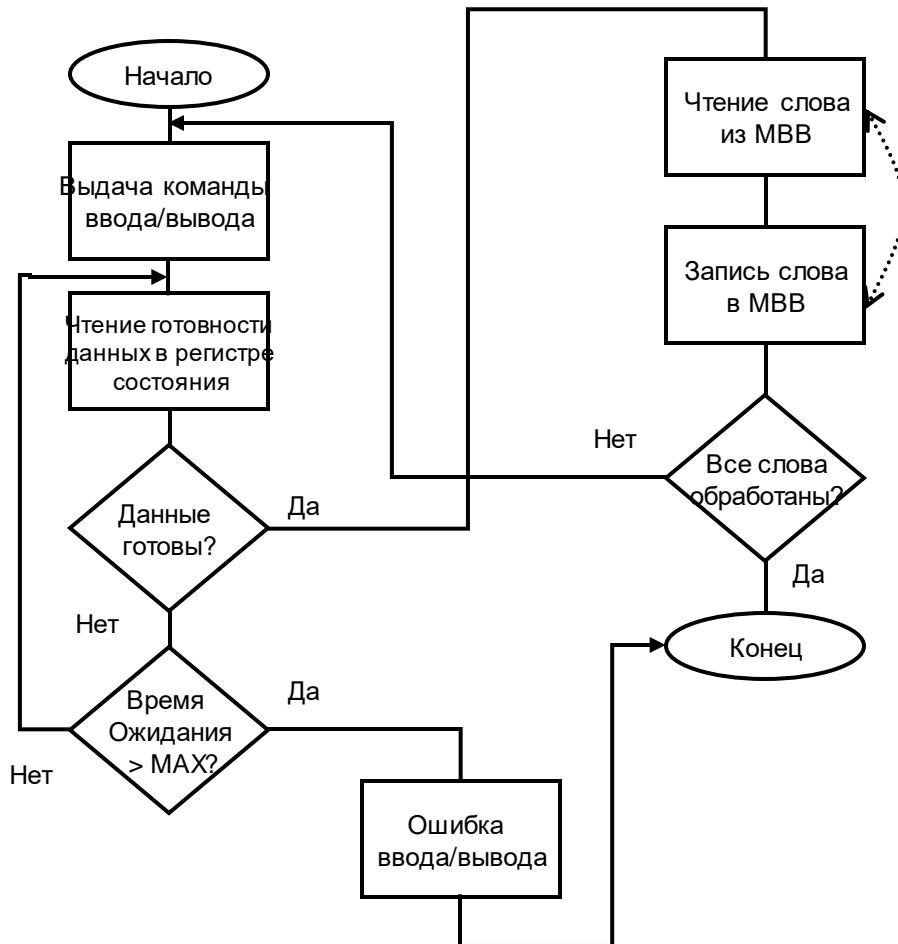
Обобщенная схема МВВ



Методы управления вводом/выводом

- Программно-управляемый ввод/вывод.
- Ввод/вывод по прерыванию.
- Прямой доступ к памяти.

Программно-управляемый ввод/вывод.

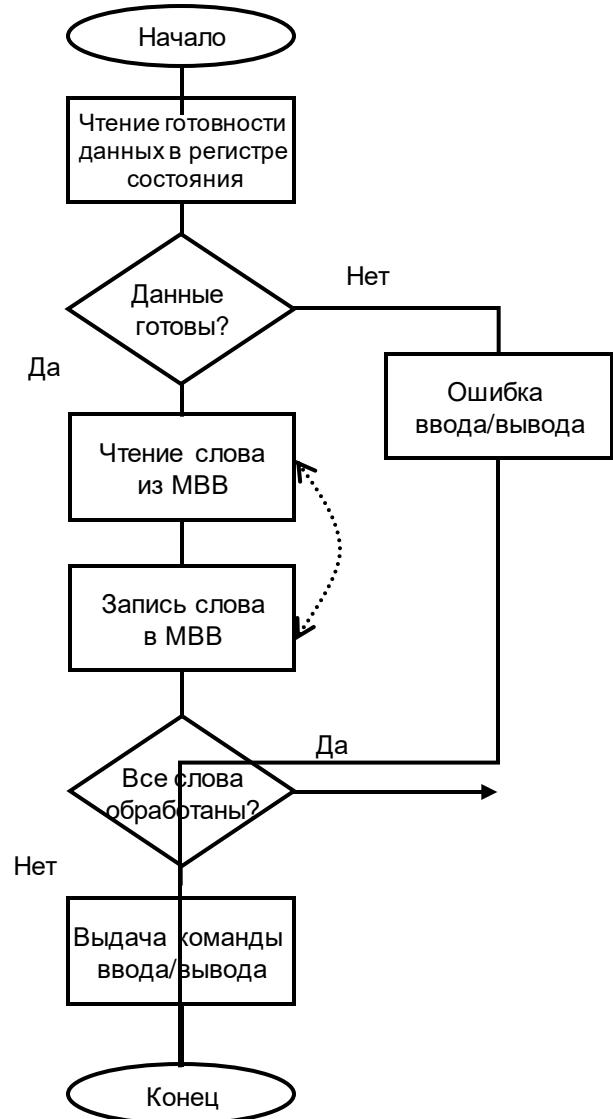
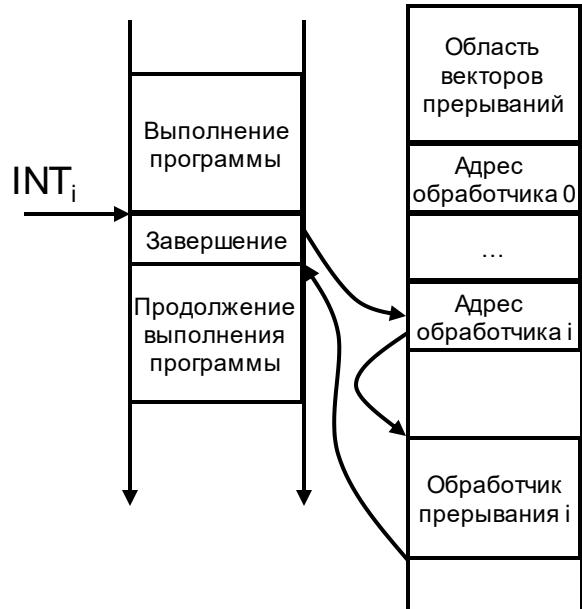


(+) Минимальные аппаратные затраты

(+) Простота изменения процедур ввода/вывода

(-) Простой ЦП из-за ожидания готовности МВВ к передаче.

Ввод/вывод по прерыванию.



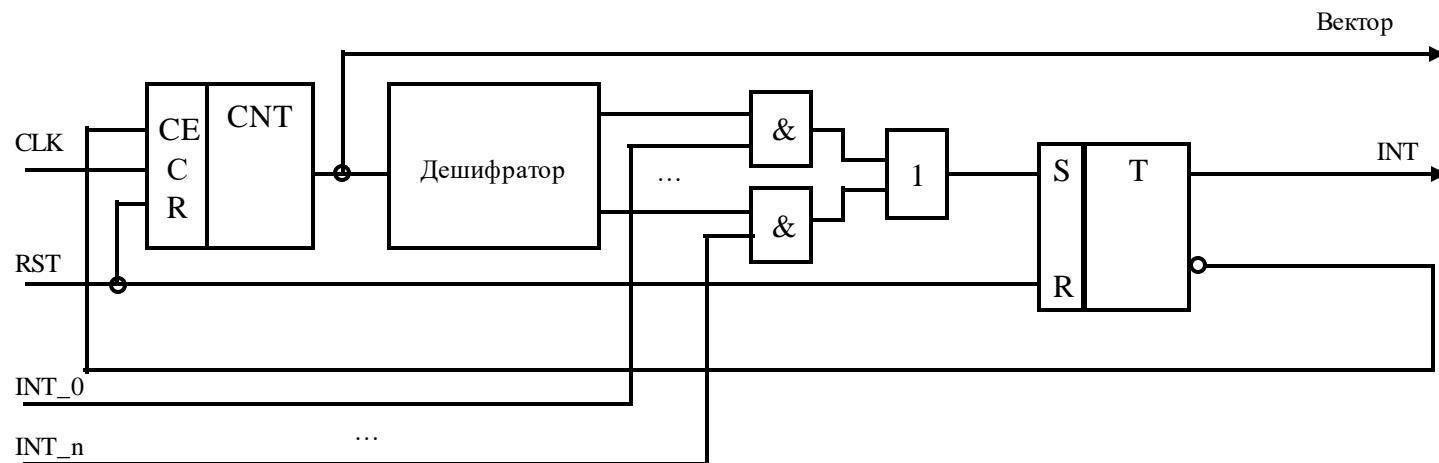
(+) Эффективнее программного ввода/вывода

(-) Запись данных в ОП через ЦП

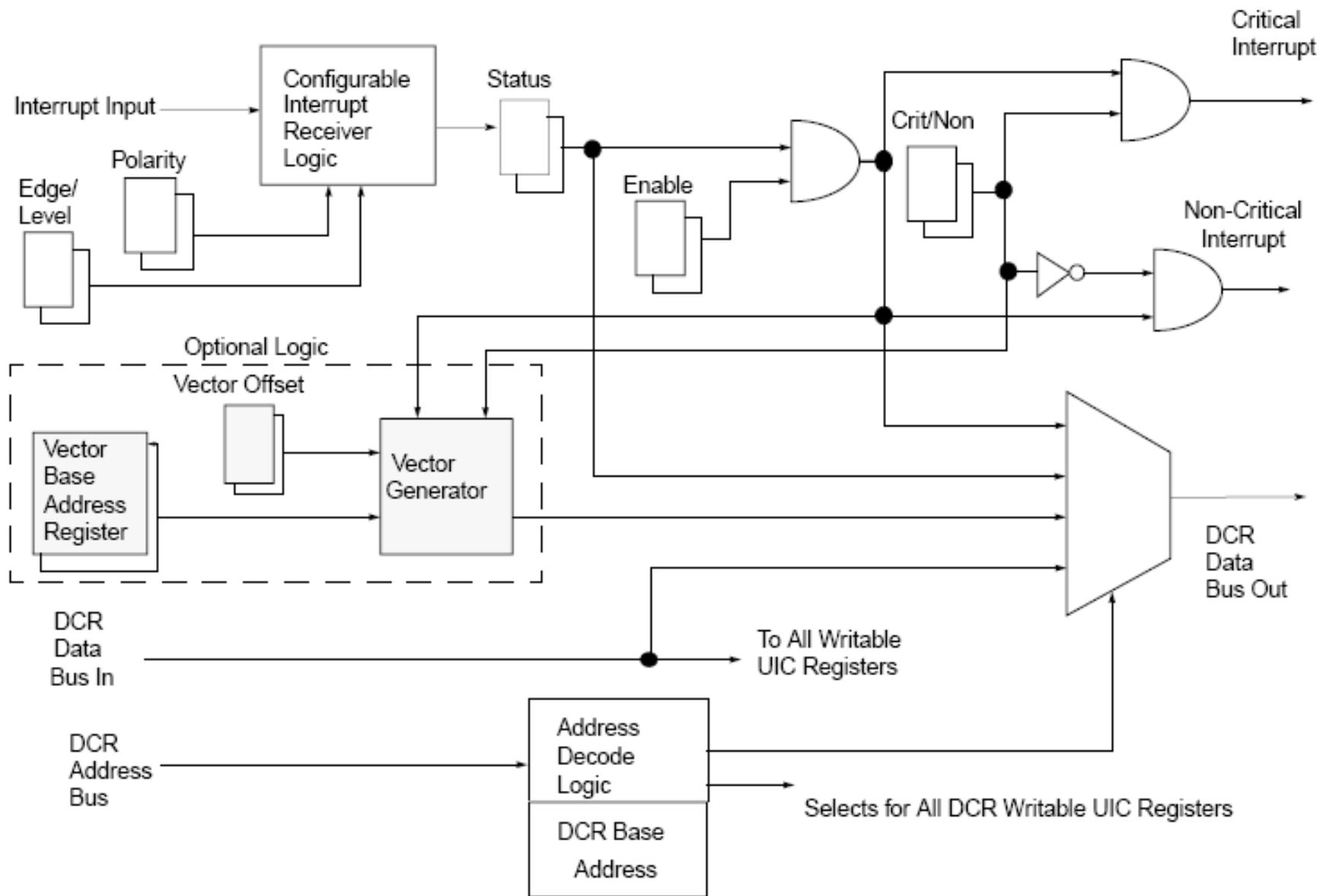
Способы определения адреса источника прерывания.

- По номеру линии прерывания (необходимо много линий).
- Программным опросом всех МВВ (необходимо много времени на обработку).
- Векторизация прерываний (необходимо использовать устройство для передачи запросов прерываний).

Схема циклического опроса



Универсальный контроллер прерываний (IBM PowerPC405 UIC).



Прямой доступ к памяти

Для пересылки информации между ОП и быстродействующими устройствами необходимо обеспечивать независимую от ЦП передачу данных без использования ввода/вывода по прерыванию. Для этих целей используется специальное устройство – контроллер прямого доступа к памяти. ЦП занят только инициализацией ПДП.

Прямой доступ к памяти (англ. direct memory access, DMA) — режим обмена данными между устройствами компьютера или же между устройством и основной памятью, в котором центральный процессор (ЦП) не участвует. Так как данные не пересылаются в ЦП и обратно, скорость передачи увеличивается.

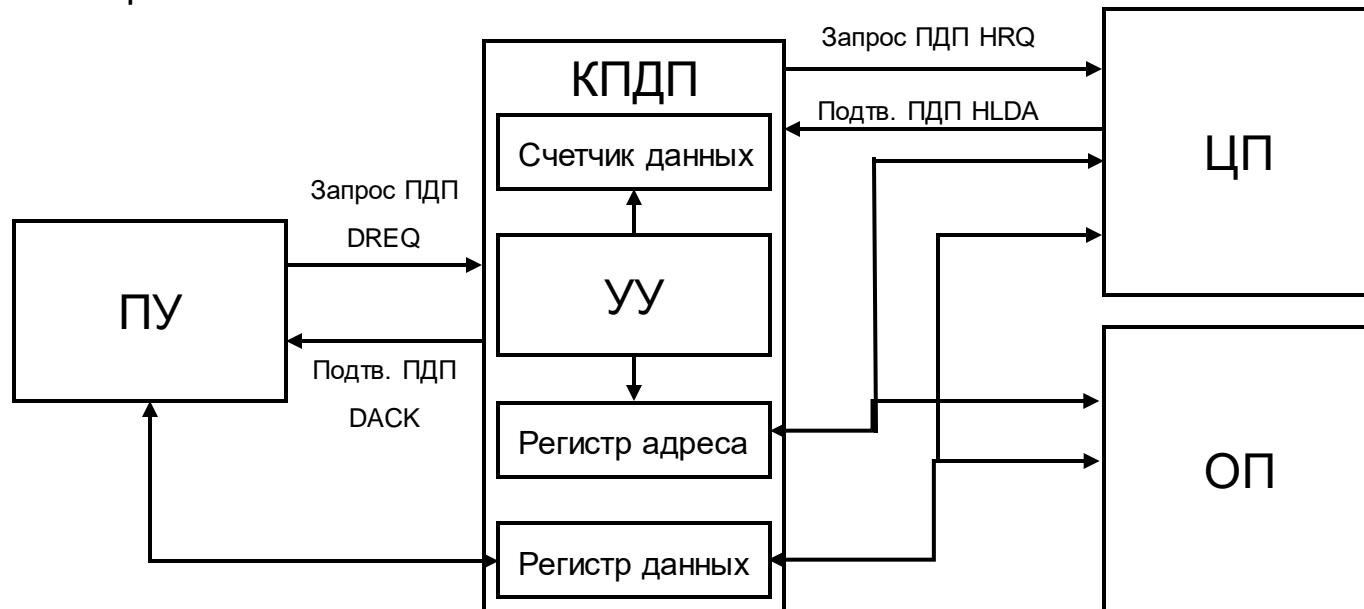
По инициативе ЦП или ПУ в КПДП передается:

- Вид запроса (чтение или запись);
- Адрес буфера в ОП;
- Количество слов для пересылки;
- Адрес ПУ.

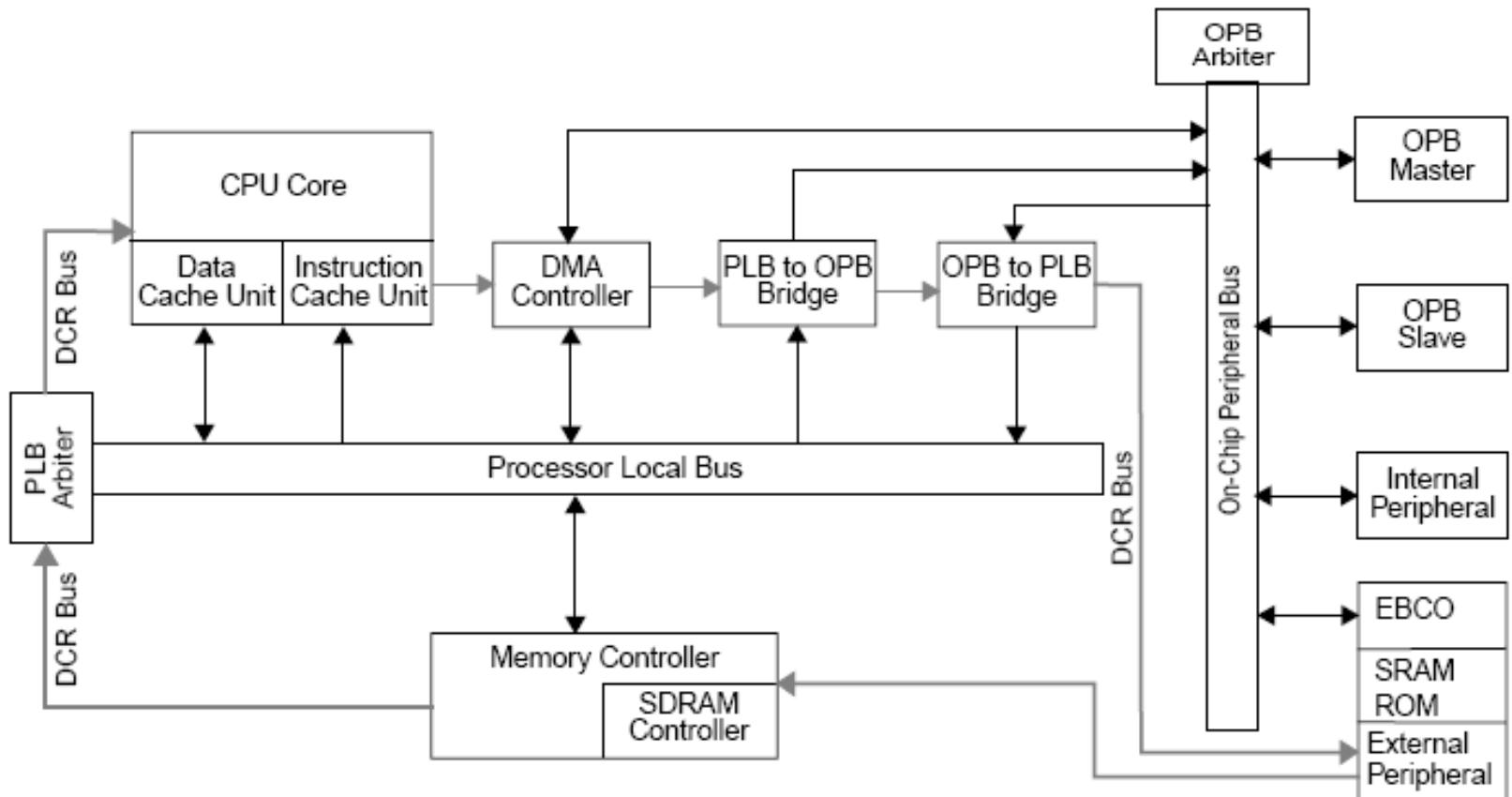
Общий алгоритм ПДП.

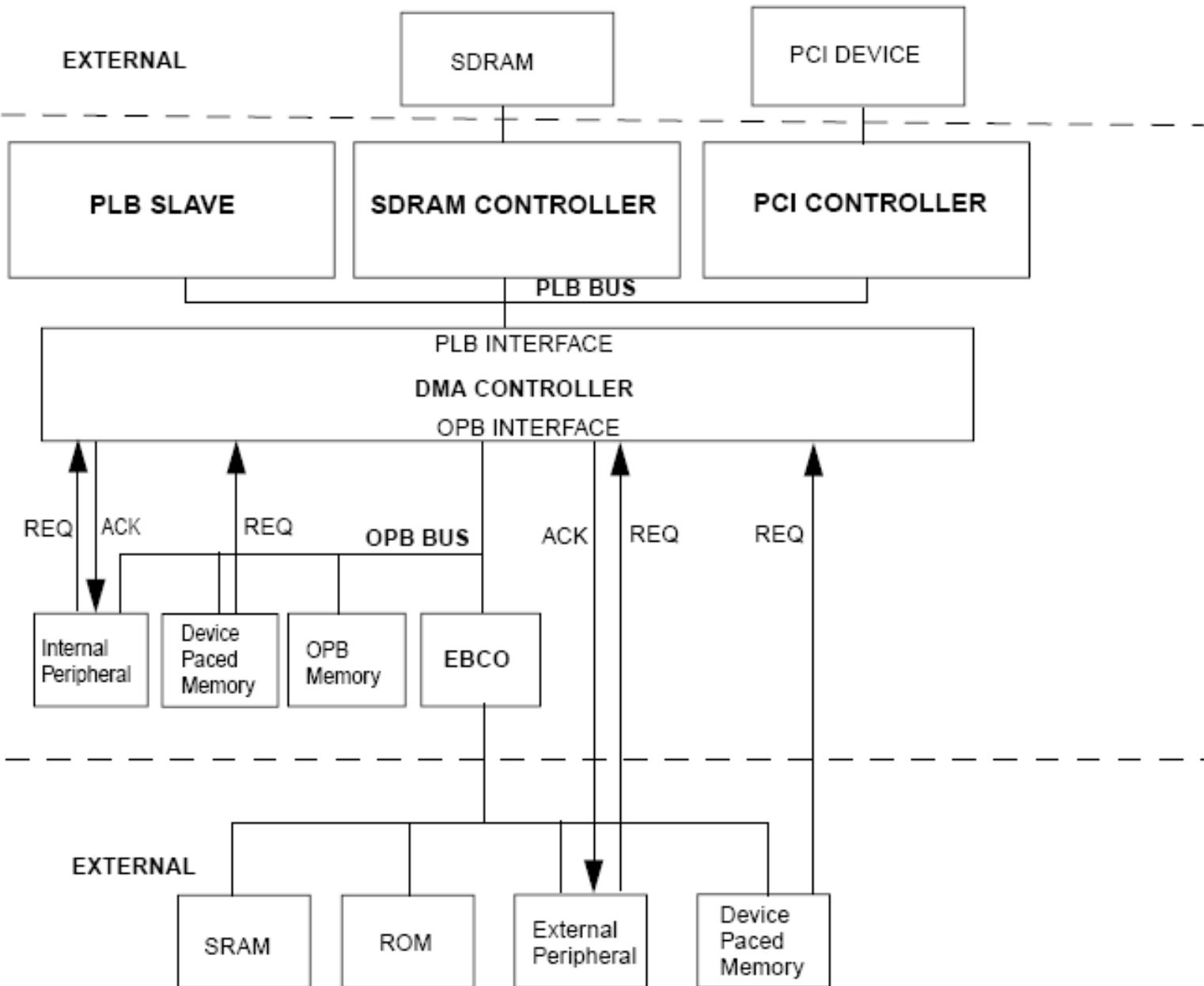
Для осуществления прямого доступа к памяти контроллер должен выполнить ряд последовательных операций:

- принять запрос (DREQ) от устройства ввода-вывода;
- сформировать запрос (HRQ) в процессор на захват шины;
- принять сигнал (HLDA), подтверждающий захват шины;
- сформировать сигнал (DACK), сообщающий устройству о начале обмена данными;
- выдать адрес ячейки памяти, предназначеннной для обмена;
- выработать сигналы (MEMR, IOW или MEMW, IOR), обеспечивающие управление обменом;
- по окончании цикла DMA либо повторить цикл DMA, изменив адрес, либо прекратить цикл.

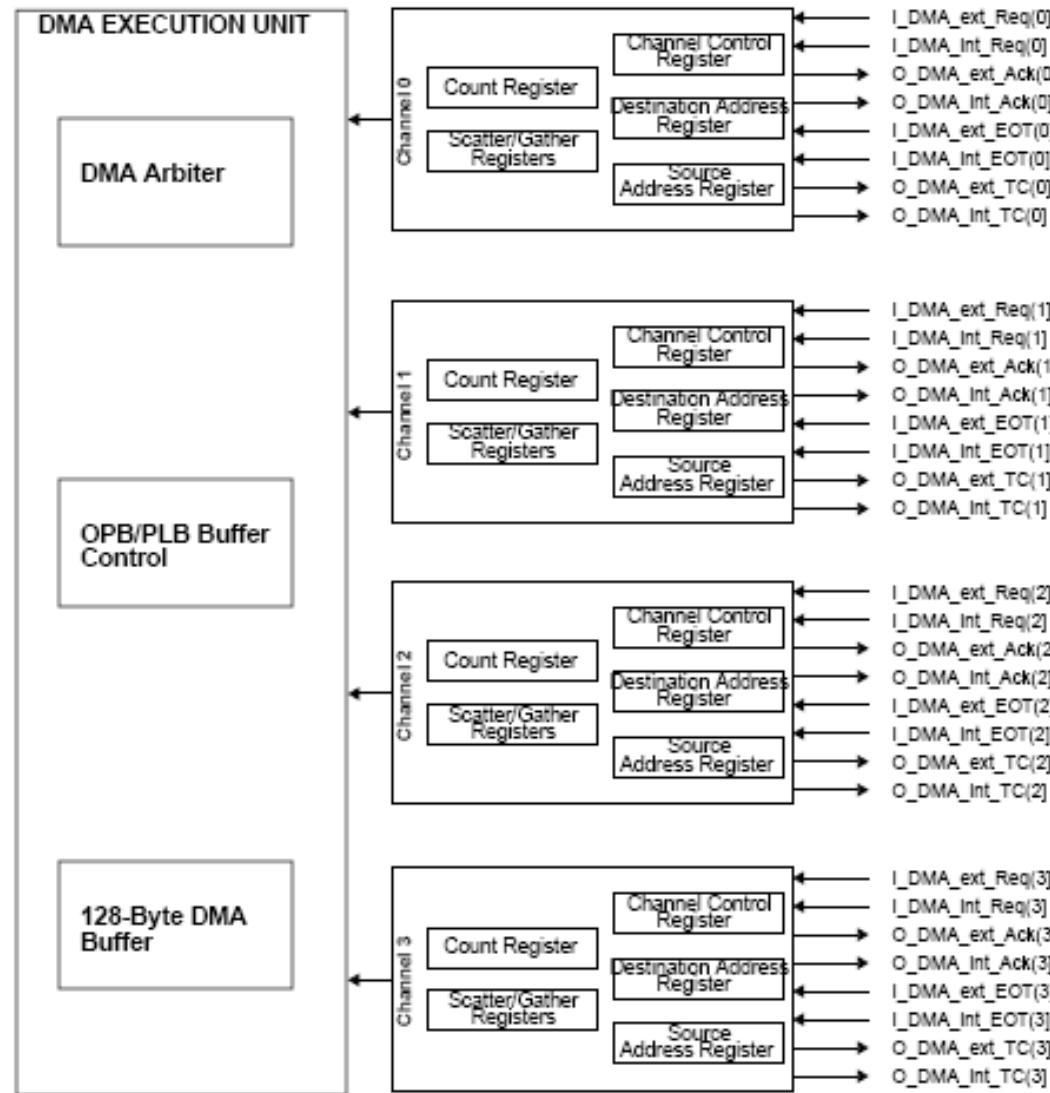


Пример построение системы с контроллером прямого доступа в память (IBM PowerPC405).

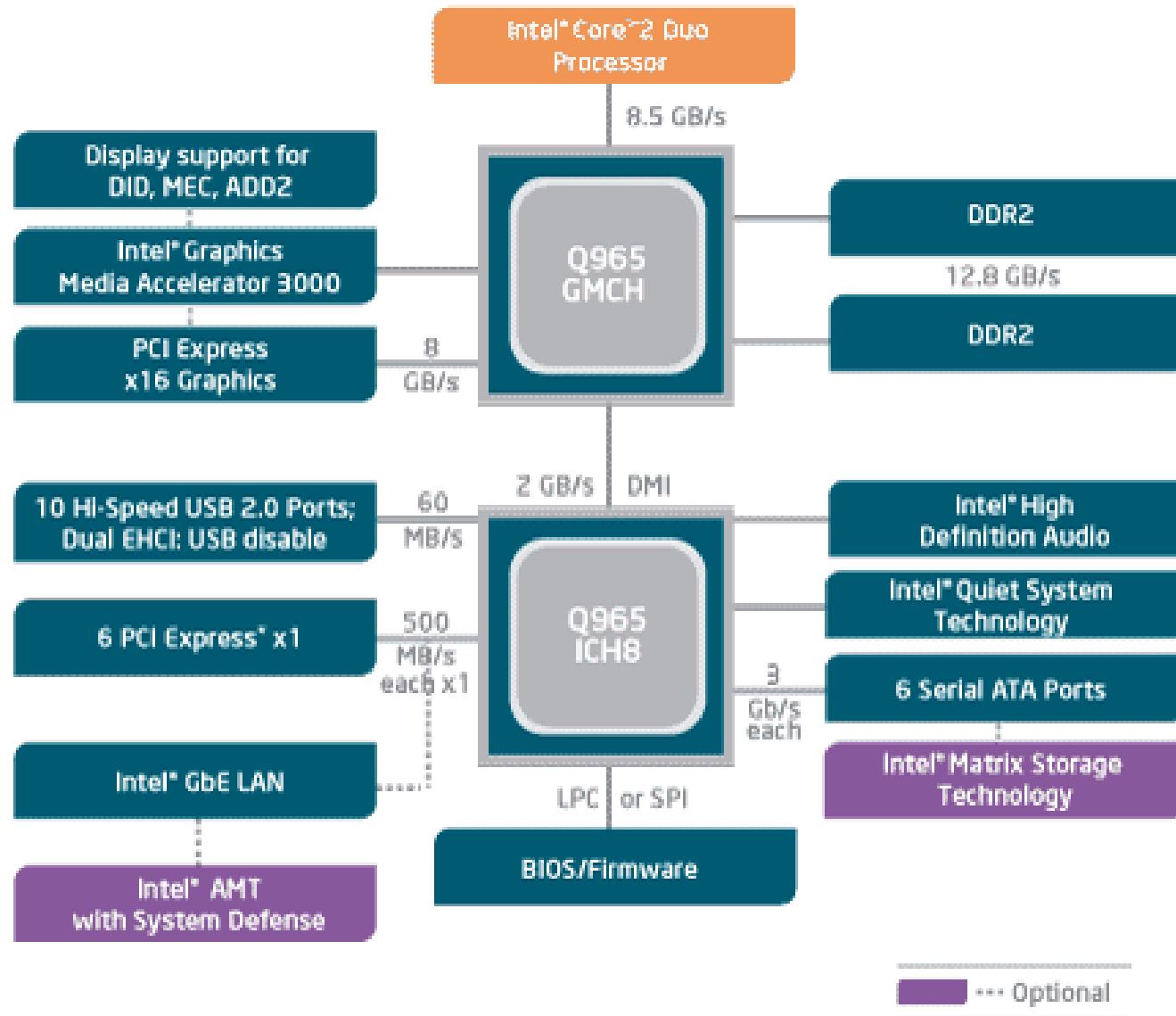




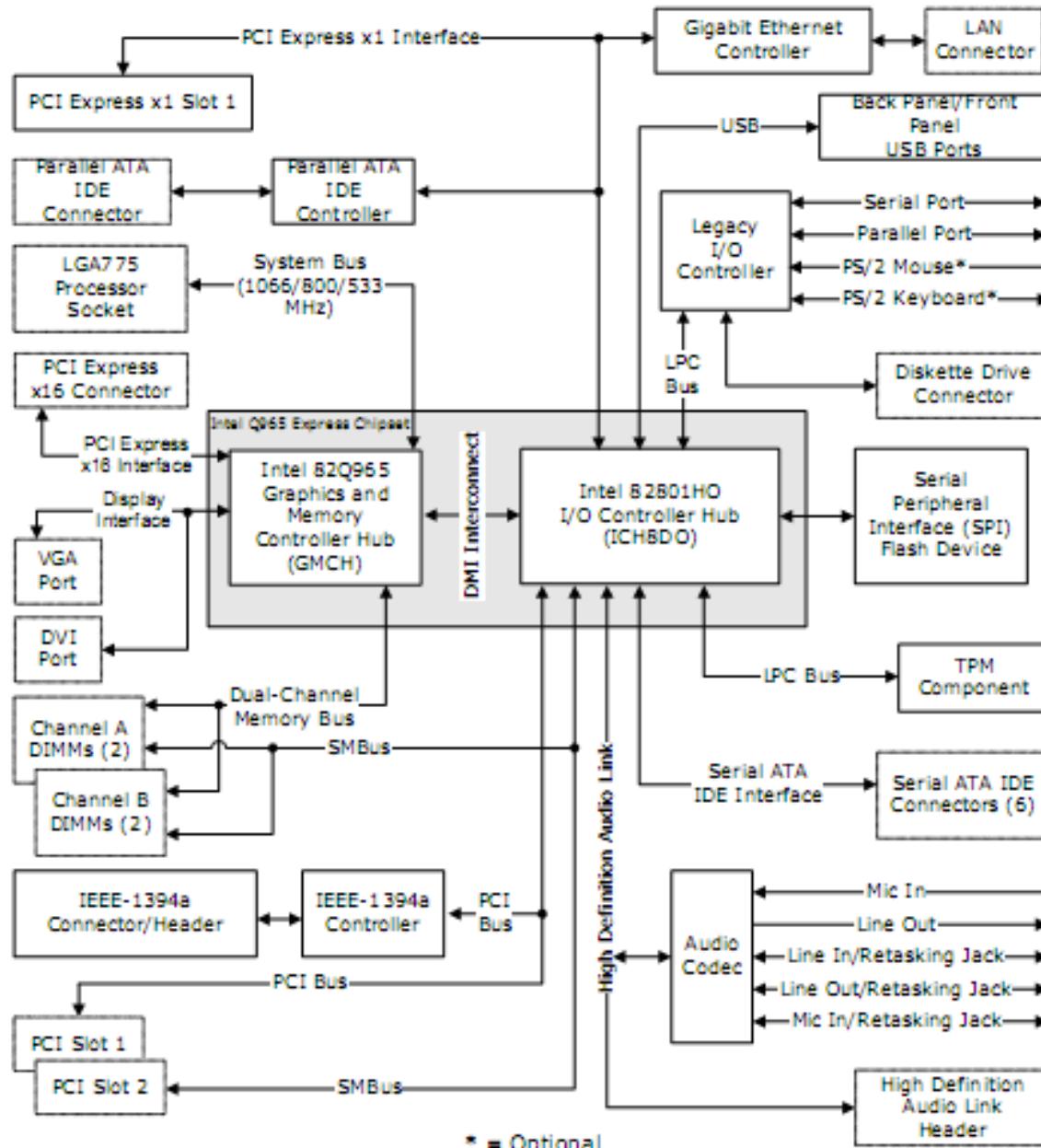
Контроллер прямого доступа в память (IBM PowerPC405 DMA).



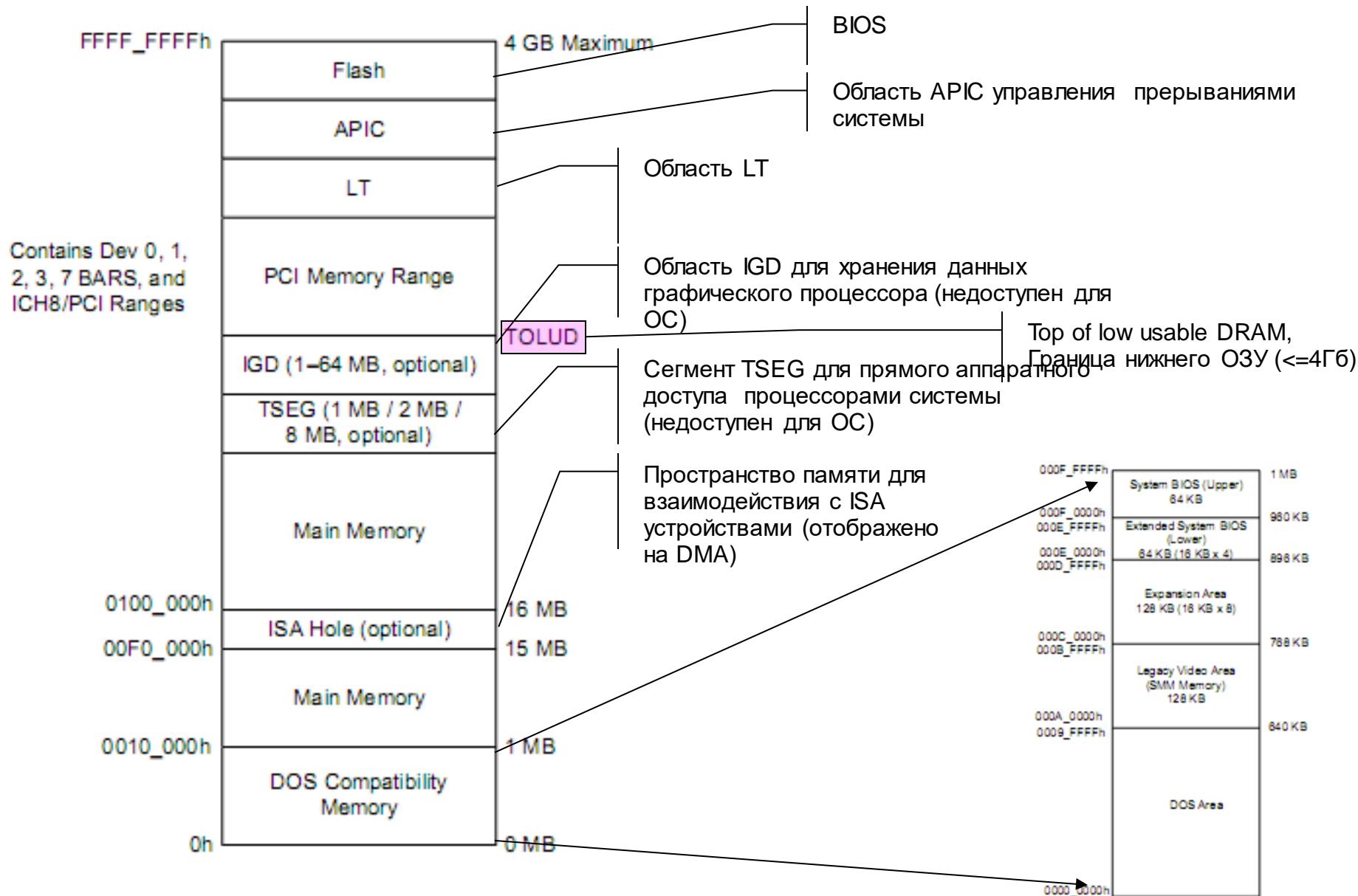
Архитектура системы на базе набора Q965



Архитектура системы на базе набора Q965



Карта памяти набора Q965



VI. Организация шин

Совокупность устройств и сигнальных линий, обеспечивающих взаимосвязь всех частей ЭВМ образуют систему шин.

Состав системы:

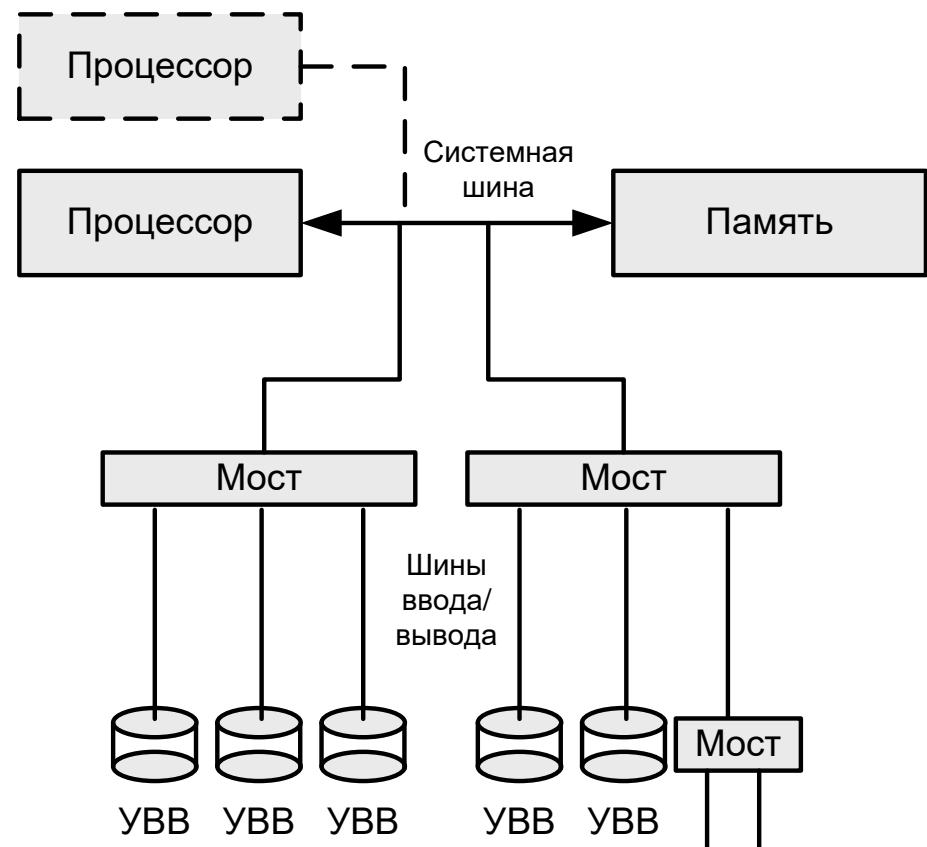
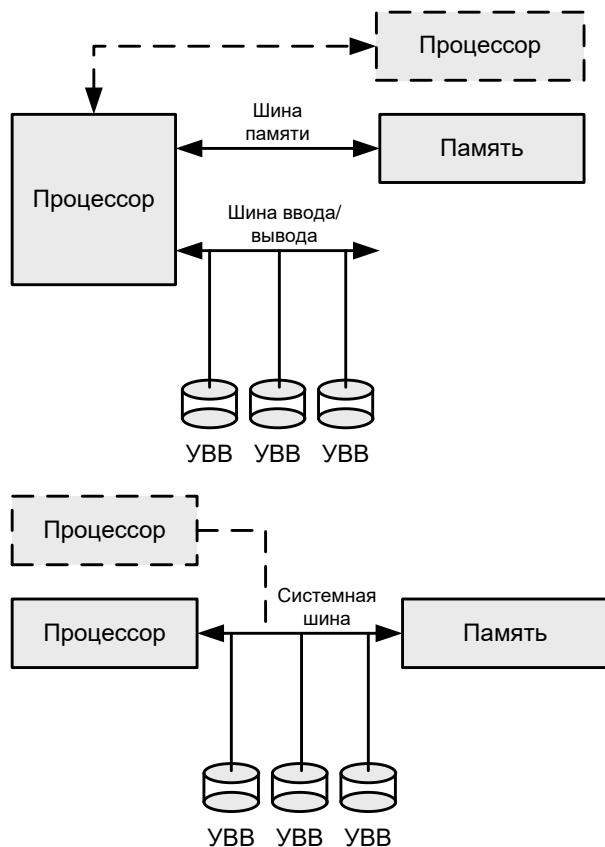
- Сигнальные линии
- Устройства арбитража
- Разъемы

Для описания (спецификации) конкретной шины необходимо описать:

- Совокупность линий (сигналов) и их назначение.
- Протокол: правила взаимодействия устройств с помощью шины (диаграммы, алгоритмы, автоматы).
- Физические, механические и электрические параметры шины и подключаемых устройств (частота, уровни сигналов, способ согласования волновых сопротивлений, длины линий, характеристики разъемов и т.д.).

Типы шин:

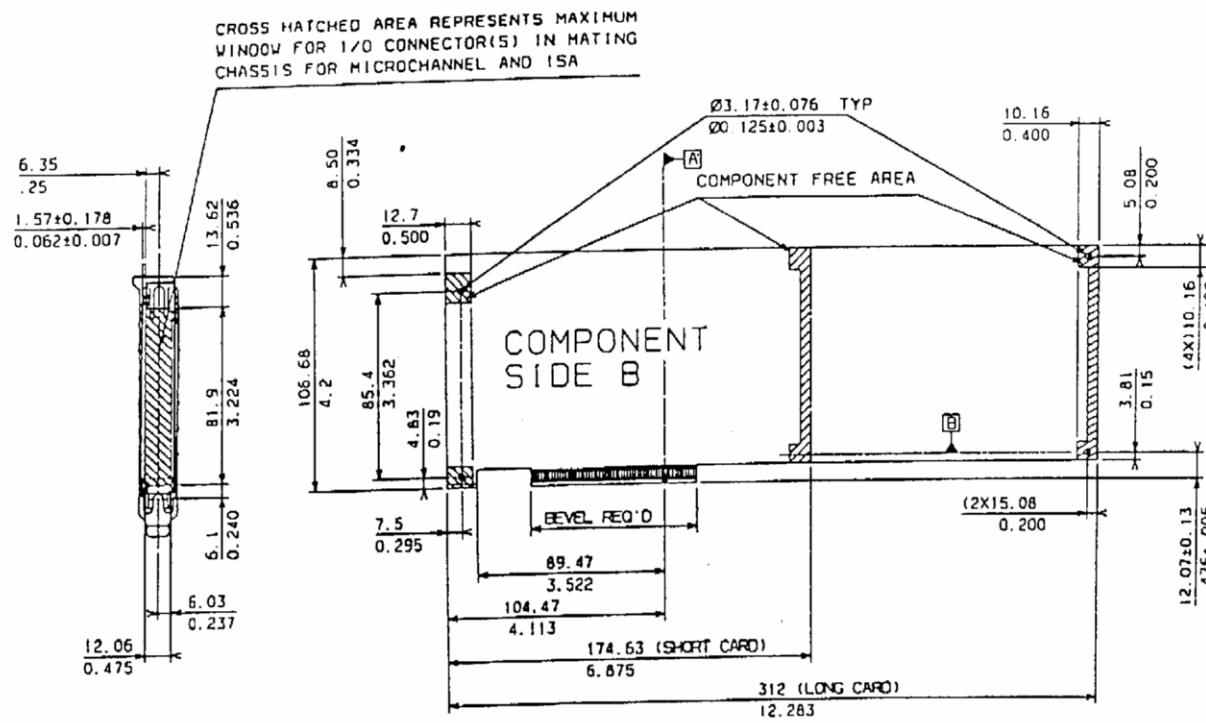
- Шины «процессор-память», «процессор-кэш», «процессор-процессор» (HyperTransport, P6 Back-Side Bus, ...)
- Шины ввода-вывода (PCI, SCSI, PCI-X, IDE, GX, ...)
- Системные шины (EISA, Pentium 4 FSB, Multibus II, NuBus, ...)



Механические аспекты спецификации шин

Механические аспекты спецификации шин включают описание вариантов исполнения, чертежи разъемов, размеры и допуски печатных плат, описание направляющих и ключей, описание способов испытаний и т.д.

ПРИМЕР



Электрические аспекты спецификации шин

- Спецификация динамических и статических характеристик
- Спецификация синхронизации
- Спецификация инициализации
- Спецификация нагрузки
- Спецификация питания
- Назначения контактов разъема

Для каждого сигнала должны быть определены параметры:

- Уровни сигналов логического «0» и логической «1» (ТТЛ, ЭСЛ, КМОП и др.).
- Время переключения
- Моменты фиксации состояния относительно сигнала синхронизации
- Условия перевода в третье состояние
- Способ согласования.

При распространении сигналов по параллельным линиям необходимо учитывать:

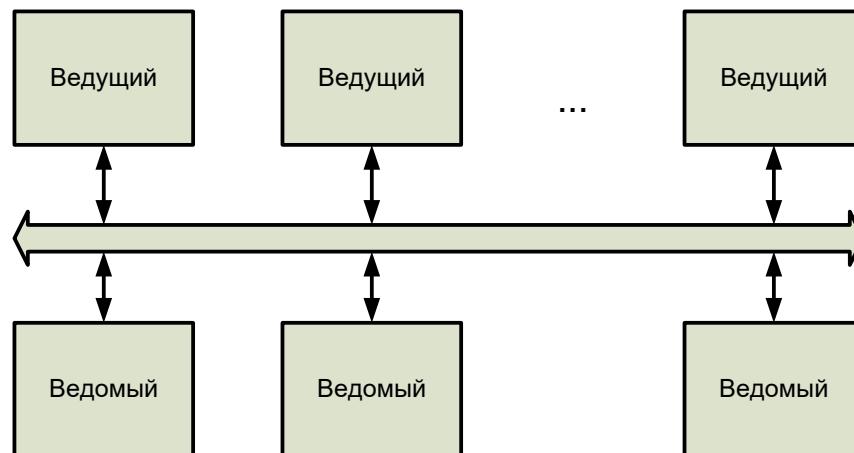
- Скорость распространения (обычно составляет ~70% скорости света $0.7 \cdot 300$ мм/нс).
- Отражение сигнала
- Перекос сигналов
- Перекрестные помехи

Типы сигнальных линий:

- Линии адреса
- Линии данных (линии адреса и данных могут объединяться).
- Линии управления для передачи типа транзакции, статуса устройства, сигналов арбитража, запросов прерываний, резервных линий, линий константных значений.

Арбитраж шин

Ведущие устройства используют шину в разное время и должны отдавать и захватывать ее. Для определения очередности подключения ведущих устройств и учета их приоритетности используются: статические приоритеты, динамические приоритеты.

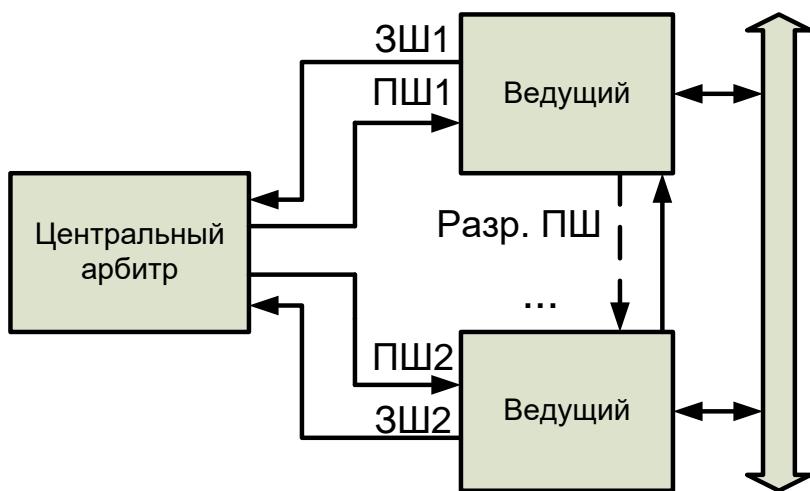


По способу арбитража: централизованный арбитраж, децентрализованный арбитраж.

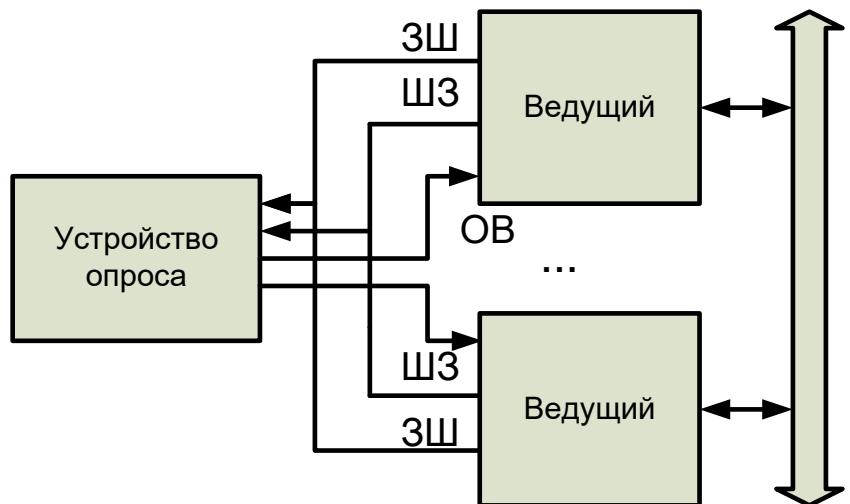
Схемы арбитража: циклическая схема, циклическая с учетом последнего, рандомизированный, схема с равными приоритетами, схема LRU

Централизованный арбитраж

Параллельный арбитраж

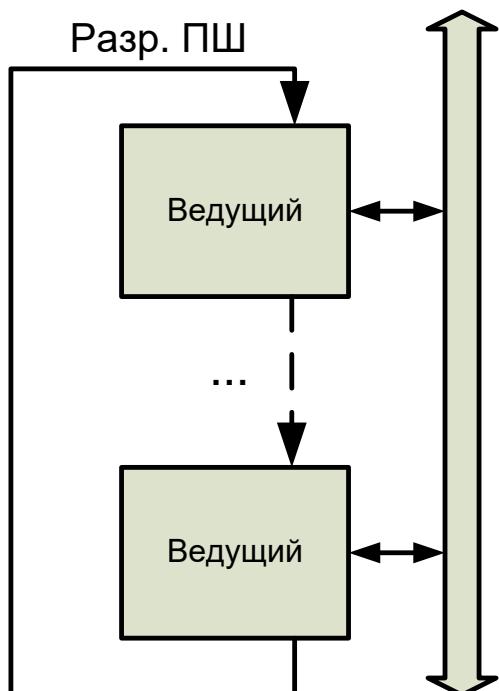


Централизованный опрос

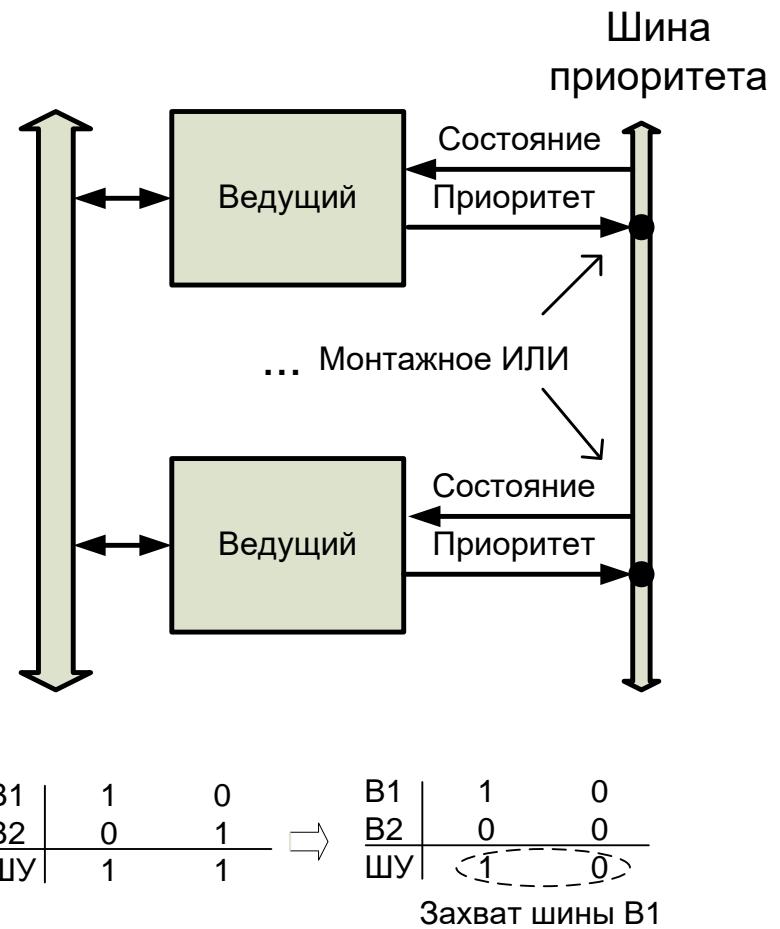


Децентрализованный арбитраж

Цепочечная схема без централизованного арбитража



Распределенный арбитраж



Системная шина процессоров Р6

Частота: 66, 100, 133.

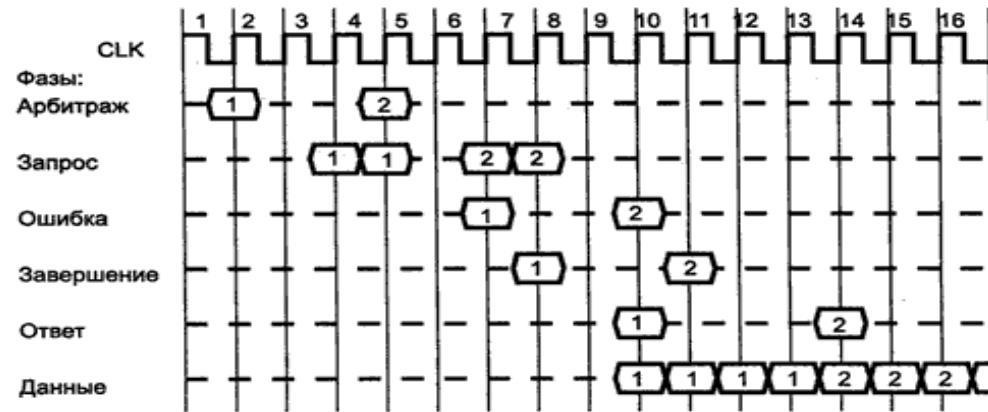
Количество абонентов: 16

Разрядность адреса: 32

Разрядность данных: 64

Контроль информации: РЕ и ECC

Поддержка операций с кэш



Пакетная и транзакционная передача

Каждое устройство-агент, подключенное к этой шине (например, любой из процессоров), до инициализации запроса должно получить через механизм арбитража право на использование шины запроса. Запрос выходит за два смежных такта: в первом такте передается адрес, тип обращения (чтение-запись памяти или ввода/вывода) и тому подобная информация. Во втором такте передается уникальный идентификатор транзакции, длина запроса, разрешенные байты шины и т. п. Через три такта после запроса проверяется состояние ошибки (error status) для защиты от ошибок передачи или нарушений протокола.

Любая обнаруженная ошибка вызывает повтор запроса, а вторая ошибка для того же запроса вызывает исключение контроля (machine check exception).

Описание сигналов системной шины

Сигнал	I/O	Назначение
A[35:3]#	I/O	Address - сигналы шины адреса. Когда сигнал AdS# активен, на шине присутствует адрес, когда пассивен - информация о типе транзакции. По окончании действия сигнала Reset# процессор с шины адреса получает конфигурационную информацию.
A20M#	I	A20 Mask - маскирование бита A20 физического адреса для эмуляции адресного пространства 8086 в реальном режиме (его использование в защищенном режиме приведет к непредсказуемым результатам). Во время действия сигнала Reset# используется для конфигурирования умножителя частоты.
AdS#	I/O	Address Strobe - строб адреса, вводимый инициатором обмена как индикатор действительности адреса. По этому сигналу все агенты шины начинают проверять проверку паритета и протокола, декодирование адреса, внутреннее слежение и другие операции, связанные с новой транзакцией.
AErr#	I/O	Address Parity Error - ошибка паритета на шине адреса. В зависимости от конфигурирования по включению питания сигнал может приводить к аварийному прекращению транзакции.
AP[1:0]#	I/O	Address Parity - биты паритета шины адреса. AP1# относится к AP[35:24]#, AP0# - к A[23:3]#. Сигнал корректного паритета должен иметь низкий уровень, если низкий уровень имеет нечетное количество контролируемых им линий.
BClk	I	Bus Clock - синхронизация шины. Значения всех синхронных сигналов действительны по положительному перепаду этого сигнала.

BErr#	I/O	Bus Error - неисправимая ошибка шины.												
BInit#	I/O	Bus Initialization - инициализация шины. Если при конфигурировании использование сигнала разрешено, то он вызывает прерывание текущей транзакции с потерей данных и сброс в исходное состояние управляемых автоматов всех агентов шины и их циклических идентификаторов арбитража.												
BNR#	I/O	Block Next Request - запрос блокировки следующей транзакции. Вводится любым агентом шины как запрос на приостановку, когда он не может воспринять следующую транзакцию.												
BP[3:2]#	I/O	Breakpoint - сигналы от процессоров, указывающие на попадание в точку останова.												
BPM[1:0]#	I/O	Breakpoint Monitor - сигналы от процессоров, указывающие на попадание в точку останова или срабатывание счетчиков, используемых для мониторинга производительности процессора.												
BPri#	I	Bus priority Request - сигнал, используемый для арбитража запросов на владение шиной.												
BR0#	I/O	Bus Request - запрос шины.												
BR[3:1]# ¹	I	Эти сигналы процессоры соединяются с линиями шины BReq[3:0]#												
BSel#	O	Bus Select - Задает частоту системной шины процессора. BSel1# BSel0# Частота системной шины, МГц <table border="1"> <tbody> <tr> <td>0</td><td>0</td><td>66</td></tr> <tr> <td>0</td><td>1</td><td>100</td></tr> <tr> <td>1</td><td>0</td><td>-</td></tr> <tr> <td>1</td><td>1</td><td>133</td></tr> </tbody> </table>	0	0	66	0	1	100	1	0	-	1	1	133
0	0	66												
0	1	100												
1	0	-												
1	1	133												

CPUPres#	O	CPU Present - признак наличия процессора в сокете.
D[63:0]#	I/O	Data - сигналы шины данных. Источник данных при передаче указывает на их действительность сигналом DRdy#.
DBsy#	I/O	Data Bus Busy - шина данных занята. Используется агентом, передающим данные, для указания на занятость шины данных
DEFER#	I	Сигнал, указывающий на то, что исходный порядок выполнения транзакций не гарантируется.
DEP[7:0]#	I/O	Data Bus ECC Protection - дополнительные сигналы защиты шины данных ECC-кодом.
DRdy#	I/O	Data Ready - готовность данных. Устанавливается источником данных для указания на присутствие достоверных данных нашине.
FErr#	O	Floating-point Error - ошибка FPU.
Flush#	I	Асинхронный сигнал на очистку внутреннего кэша. По этому сигналу выполняются все обратные записи и аннулируются строки кэша обоих уровней. Новые строки не выделяются до окончания действия сигнала. Значение сигнала во время окончания действия сигнала Reset# используется для конфигурирования процессора.
FRCER#	I/O	Functional Redundancy Checking Error - сигнал ошибки, обнаруженной проверочным процессором в функционально-избыточной системе.
Hit#, HitM#	I/O	Сигналы результатов операции слежения за транзакцией. Hit# (Snopp hit) указывает на кэш-попадание. HitM#(Hit Modified) указывает на попадание в модифицированную строку, запрещая другим контроллерам шины обращаться к этим данным до выполнения обратной записи.

IErr#	O	Internal Error - сигнал обнаружения внутренней ошибки. Обычно появляется вместе с транзакцией Shutdown. Сигнал ошибки сохраняется до его программного сброса обработчиком NMI или аппаратного сброса сигналами Reset#, BInit# или Init#
IgnNE#	I	Ignore Numeric Error - игнорирование ошибки сопроцессора - запрет вырабатывать исключения. Используется для совместимости с AT, где вместо исключения вырабатывается аппаратное прерывание. Во время действия сигнала Reset# используется для конфигурирования умножителя частоты.
Init#	I	Initialization - "Мягкая" инициализация процессора. Сигнал приводит к сбросу общих регистров и переходу по вектору, заданному при конфигурировании по включению. Содержимое кэш-памяти, буферов записи и регистров FPU не затрагивается. Если сигнал активен во время окончания действия сигнала Reset#, процессор выполняет BIST
LInt[1:0] (NMI,IntR)	I	Local APIC Interrupt - входы прерываний локальных контроллеров APIC. Если работа APIC запрещена, LInt0 становится сигналом IntR, LInt1 - сигналом NMI. По сигналу Reset# работа APIC разрешается и входы работают в режиме APIC, который может быть отменен программно. Во время действия сигнала Reset# используются для конфигурирования умножителя частоты.
Lock#	I/O	Блокировка шины на время транзакции.
PICCIk	I	APIC Clock - синхронизация шины APIC. В режиме FRC частота должна быть равной 1/4 BCIk.
PICD[1:0]	I/O	APIC Data - двунаправленная последовательная шина обмена сообщениями APIC.

PRdy#	O	Probe Ready - сигнал готовности зонда, используемый аппаратными средствами отладки. Указывает на остановку нормального исполнения в ответ на сигнал R/S#(вход в зондовый режим).
Preq#	I	Probe Request - запрос зонда на отладочную операцию.
PwrGood	I	Power Good - сигнал исправности питания, указывающий на стабильность питающих напряжений и сигнала синхронизации.
Req[4:0]#	I/O	Request Command - запрос команды. Вводится текущим владельцем шины для определения типа активной транзакции.
Reset#	I	Сброс процессора - конфигурирование процессора, инициализация регистров, очистка кэша обоих уровней (без выполнения обратной записи) и переход к вектору сброса (по умолчанию 0xFFFFFFF0h).
RP#	I/O	Request Parity - бит паритета запроса, защищающий линии AdS# и Req[4:0]#. Сигнал корректного паритета должен иметь низкий уровень, если низкий уровень имеет нечетное количество контролируемых им линий.
RS[2:0]#	I	Response Status - состояние ответчика. Сигналы управляются агентом, отвечающим за завершение текущей транзакции.
RsP#	I	Response Parity - бит паритета для сигналов RS[2:0]#.
SlotOcc# ²	O	Slot Occupied - слот занят. Низкий уровень сигнала свидетельствует о наличии процессора или терминатора в слоте Pentium-2. Используется с комбинацией сигналов VID[4:0] для определения наличия процессорного ядра в слоте: при VID[4:0]=11111 в слоте терминатор.

ПРИМЕР

Slp# ²	I	Sleep - сигнал, переводящий процессор из состояния Stop Grant в состояние Sleep (спящий режим).
SMI#	I	System Management Interrupt - сигнал прерывания для входа в режим SMM.
StpClk#	I	Stop Clock - асинхронный сигнал, переводящий процессор в состояние Stop Grant с малым потреблением .
TCK	I	Test Clock - вход синхронизации шины тестирования Test Bus, называемой также TAP (Test Access Port).
TDI	I	Test Data In - последовательный вход данных интерфейса JTAG.
TDO	O	Test Data Out - последовательный выход данных интерфейса JTAG.
TestHi	I	Сигнал, подключаемый к источнику 2,5 В через резистор 1-10 кОм.
TestLo	I	Сигнал, подключаемый к шине GND.
ThermTrip#	O	Thermal Trip - сигнал останова процессора по перегреву. Если внутренняя температура поднимается примерно до 130°C, процессор останавливается и формирует данный сигнал, который может быть сброшен только сигналом Reset# после снижения температуры ниже этого порога.
TMS	I	Test Mode State - выбор режима тестирования по JTAG.
TRdy#	I	Target Ready - сигнал, которым целевое устройство указывает на готовность приема данных к записи или неявной обратной записи.
TRst#	I	Test Reset - сигнал сброса логики ТАР (самосброс происходит автоматически по включению).
<u>VID[4:0]</u>	O	Voltage ID - выводы идентификации для автоматической установки уровня питающего напряжения. Эти выводы могут либо быть свободными, либо подключаться к шине GND. Блок питания должен установить соответствующее напряжение, либо отключиться.
UP#	O	Upgrade Present - признак установки в сокет 8 процессора OverDrive.

Интерфейс CAN

CAN (Controller Area Network — сеть контроллеров) — стандарт промышленной сети, ориентированный прежде всего на объединение в единую сеть различных исполнительных устройств и датчиков. Режим передачи — последовательный, широковещательный, пакетный. Арбитраж — децентрализованный.

CAN является синхронной шиной с типом доступа Collision Resolving (CR, разрешение коллизии), который в отличие от Collision Detect (CD, обнаружение коллизии) сетей ([Ethernet](#)) детерминировано (приоритетно) обеспечивает доступ на передачу сообщения, что особо ценно для промышленных сетей управления (fieldbus). Передача ведётся [кадрами](#). Полезная [информация](#) в кадре состоит из [идентификатора](#) длиной 11 бит (стандартный формат) или 29 бит (расширенный формат, надмножество предыдущего) и поля данных длиной от 0 до 8 байт. Идентификатор говорит о содержимом пакета и служит для определения приоритета при попытке одновременной передачи несколькими сетевыми узлами.

Модель взаимодействия открытых систем ISO OSI

Application Layer (уровень приложений)

Presentation Layer (уровень представлений)

Session Layer (сеансовый уровень)

Transport Layer (транспортный уровень)

Network Layer (сетевой уровень)

Data Link Layer (канальный уровень)

Physical Layer (физический уровень)

Уровни взаимодействия по протоколу CAN

Для CAN протокола можно выделить два основных уровня взаимодействия устройств:

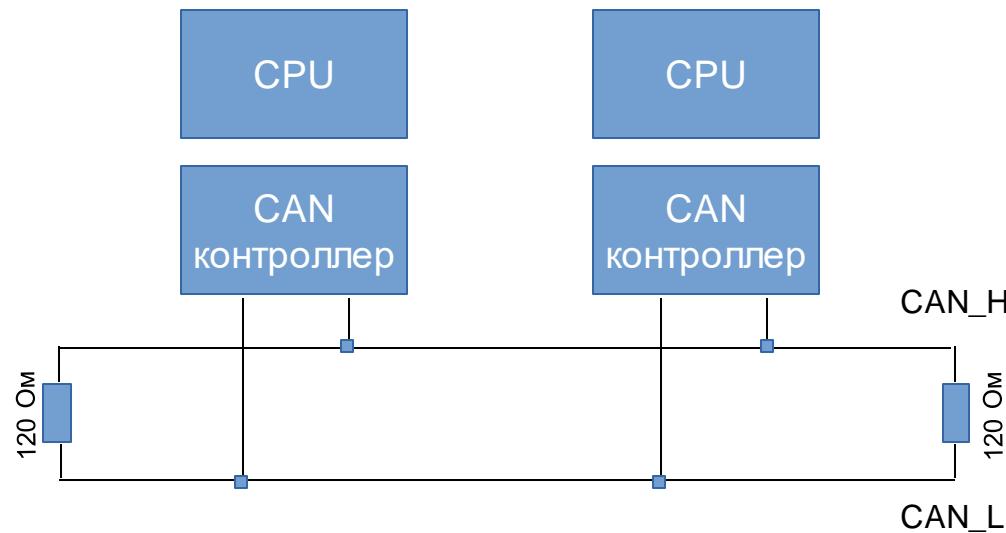
- Физический.
- Канальный.

Сетевой уровень CAN не входит в стандарт и существует в нескольких несовместимых вариантах (CANopen, DeviceNet, SDS, CAN Kingdom и др.), в связи с чем в данной работе рассматриваться не будет. На физическом уровне протокола описываются аспекты передачи бит информации: варианты организации несущей среды, средства сопряжения устройств с шиной, способ кодирования информации, устранение коллизий, низкоуровневый арбитраж. Для канального уровня описываются такие аспекты взаимодействия, как: форматы кадров, процедура передачи сообщений, адресацию, организацию приоритетов, конфигурирование, контроль ошибок и другие.

Основы CAN протокола

Доступ к шине

В CAN используется множественный доступ с опросом несущей и разрешением конфликтов. Каждый узел сети должен контролировать бездействие шины в течение некоторого времени, прежде чем послать сообщение. Как только обнаружено бездействие шины, все узлы сети имеют равную возможность передать данные.



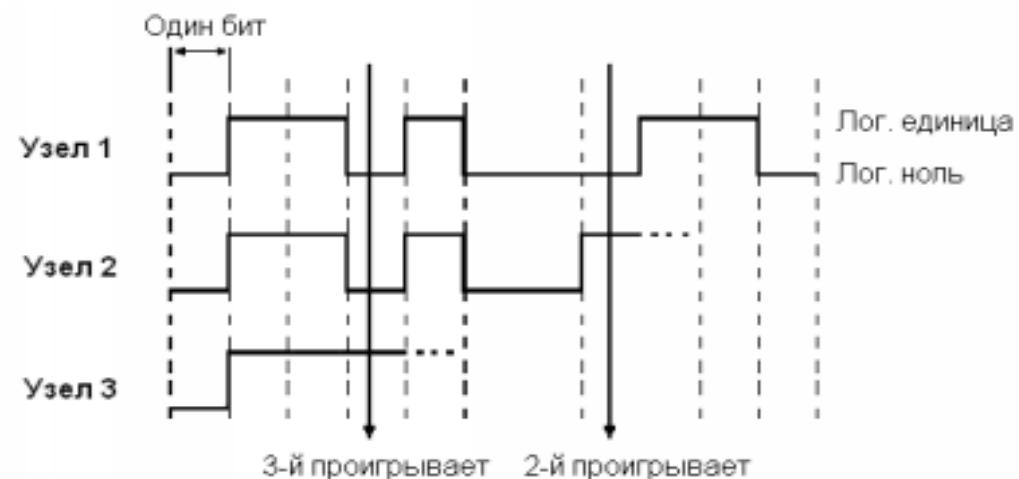
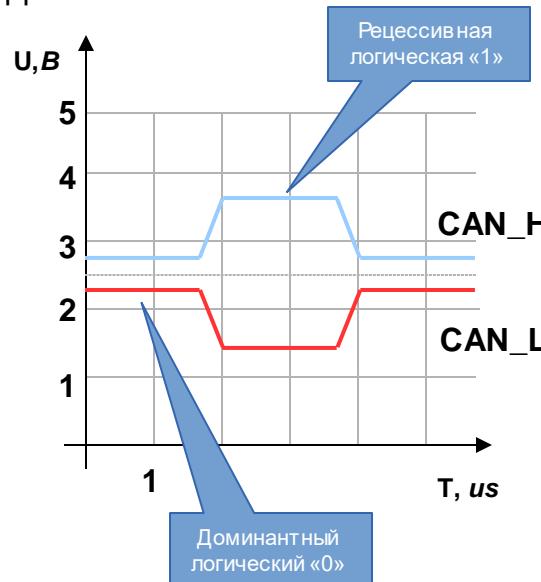
Подключенные к шине модули взаимодействуют через сеть, состоящую из двух линий, по которым передается синфазный код: CAN_H (позитивный сигнал) и CAN_L (негативный сигнал). На концах линий должны находиться резисторы 120 Ом для согласования волнового сопротивления кабеля. Скорость передачи может достигать 1 Мбит в секунду, а длина линии: 1 км. При этом все устройства подключены к шине и могут одновременно начать передачу. В результате этого возможно возникновение коллизий, которые разрешается благодаря конструкции приёмопередатчиков на физическом уровне и специальными форматами кадров на канальном уровне.

Основы CAN протокола

Арбитраж

В CAN используется неразрушающий поразрядный арбитраж на шине, сохраняющий сообщение не поврежденным при потере инициативы. Арбитраж позволяет передавать сообщения с высоким приоритетом, без каких-либо задержек и искажений. Логические уровни на шине определяются как ‘dominant’ и ‘recessive’. Узел передачи должен контролировать логическое состояние шины для начала передачи данных. На шине CAN логический бит 0 определяется как ‘dominant’, а бит – 1 как ‘recessive’.

Бит ‘dominant’ всегда будет выигрывать арбитраж у ‘recessive’ бита, поскольку имеет низкий уровень сигнала и более высокий приоритет. Каждый узел, передающий данные на шину, должен контролировать наличие передаваемого бита на шине. Сообщение с более низким приоритетом будет формировать на шине бит ‘recessive’, а при проверке обнаружит бит ‘dominant’, в этом случае узел, формирующий сообщение более низкого приоритета, теряет арбитраж. Узел, потерявший арбитраж, должен дождаться отсутствия активности на шине и повторить попытку передать данные.



Основы CAN протокола

Сообщения

В CAN протоколе сообщения не являются адресными, т.е. сообщения не адресуются от одного узла к другому. Сообщение содержит идентификатор источника и собственно данные. Все узлы CAN сети могут принять каждое сообщение на шине и самостоятельно определить: данное сообщение должно быть отвергнуто или обработано.

Другой полезной особенностью CAN протокола является возможность удаленного запроса данных (RTR). В отличие от предыдущего случая, требуемые данные не ожидаются, когда появляются на шине, а запрашиваются у конкретного узла. Проектировщик может использовать эту особенность для снижения трафика шины при сохранении целостности сети. Добавление нового узла в систему не требует перенастройки остальных устройств сети. Новый узел начинает принимать сообщения из сети на основе их идентификаторов.

По протоколу предусмотрено четыре вида кадров:

- **Кадр данных (data frame)** — используется при передаче данных. Размер передаваемой полезной информации может варьироваться в каждом кадре, так как в структуре заголовка кадра предусмотрено поле длины. Максимальный размер данных: 8 байт.
- **Кадр удаленного запроса (remote frame)** — служит для запроса на передачу кадра данных с тем же идентификатором. Кадр запроса позволяет на прикладном уровне строить более сложную логику взаимодействия «запрос-ответ».
- **Кадр перегрузки (overload frame)** — обеспечивает промежуток между кадрами данных или запроса. Данный кадр необходим для перевода приемо-передатчиков в исходное состояние.
- **Кадр ошибки (error frame)** — передаётся узлом, обнаружившим в сети ошибку. Данный кадр использован в механизме контроля ошибок протокола CAN.

Форматы кадра данных

Базовый кадр		Описание	Расширенный кадр	
Название поля	Длина (в битах)		Название поля	Длина (в битах)
Начало кадра	1	Сигнализирует начало передачи кадра	Начало кадра	1
Идентификатор	11	Уникальный идентификатор	Идентификатор А	11
Запрос на передачу (RTR)	1	Должен быть доминантным	—	—
—	—	Должен быть рецессивным. Используется для тестирования несущей	Подмена запроса на передачу (SRR)	1
Бит расширения идентификатора (IDE)	1	Должен быть доминантным (определяет длину идентификатора)	—	—
—	—	Должен быть рецессивным (определяет длину идентификатора)	Бит расширения идентификатора (IDE)	1
—	—	Вторая часть идентификатора	Идентификатор В	18
—	—	Должен быть доминантным	Запрос на передачу (RTR)	1
Зарезервированный бит (r0)	1	Резерв	Зарезервированные биты (г1 и г0)	2
Длина данных (DLC)	4	Длина поля данных в байтах (0-8)	Длина данных (DLC)	4
Поле данных	0-8 байт	Передаваемые данные (длина в поле DLC)	Поле данных	0-8 байт
Контрольная сумма (CRC)	15	Контрольная сумма всего кадра	Контрольная сумма (CRC)	15
Разграничитель контрольной суммы	1	Должен быть рецессивным	Разграничитель контрольной суммы	1
Промежуток подтверждения (ACK)	1	Передатчик шлёт рецессивный, приёмник вставляет доминанту	Промежуток подтверждения (ACK)	1
Разграничитель подтверждения	1	Должен быть рецессивным	Разграничитель подтверждения	1
Конец кадра (EOF)	7	Должен быть рецессивным	Конец кадра (EOF)	7

Контроль шины

Все передающие устройства должны использовать дополняющие биты (bit-stuffing), что гарантирует появление бита противоположного значению в шести битах шины.

Кадры не должны передаваться на шину одновременно, т.е. никакой контроллер не должен начинать передачу, если на шине передается другой кадр.

Если два или более устройств начинают передавать кадры на шину одновременно, при передаче идентификатора обнаруживается, что переданный устройством с меньшим приоритетом идентификатор не соответствует состоянию шины и вынуждены прекратить передачу кадра.

Пример:

10001111100110100110000001100111110
100011111010011010011000001011001111100

Контроль ошибок

- **Контроль передающим устройством:** приемник активного приемо-передатчика сравнивает битовые уровни в сети с передаваемыми битами.
- **Дополняющие биты (bit stuffing):** после передачи пяти одинаковых битов подряд автоматически передаётся бит противоположного значения. Таким образом кодируются все поля кадров данных или запроса, кроме разграничителя контрольной суммы, промежутка подтверждения и EOF.
- **Контрольная сумма:** передатчик вычисляет её и добавляет в передаваемый кадр, приёмник считает контрольную сумму принимаемого кадра в реальном времени (одновременно с передатчиком), сравнивает с суммой в самом кадре и в случае совпадения передаёт доминантный бит в промежутке подтверждения.
- **Контроль значений полей при приёме.**
- **Контроль правильности приема сообщения** передатчиком по биту ACK (подтверждение приема). В случае неподтвержденного приема сообщения, CAN передатчик должен повторить посылку сообщения вплоть до его подтверждения. В связи с этим CAN контроллеры имеют встроенные счетчики ошибок, позволяющие остановить передачу при превышении количества ошибок заранее установленного предела.
- Разработчики оценивают вероятность невыявления ошибки передачи как $4,7 \times 10^{-11}$.

Шина PCI

Частота: 33,66.

Количество абонентов: 21

Разрядность адреса/данных: 32,64 (132...528 МБ/сек)

Контроль информации: по четности

Поддержка операций с кэш

Пакетная передача

Поддержка иерархии шин (до 256)

Plug and Play технология.

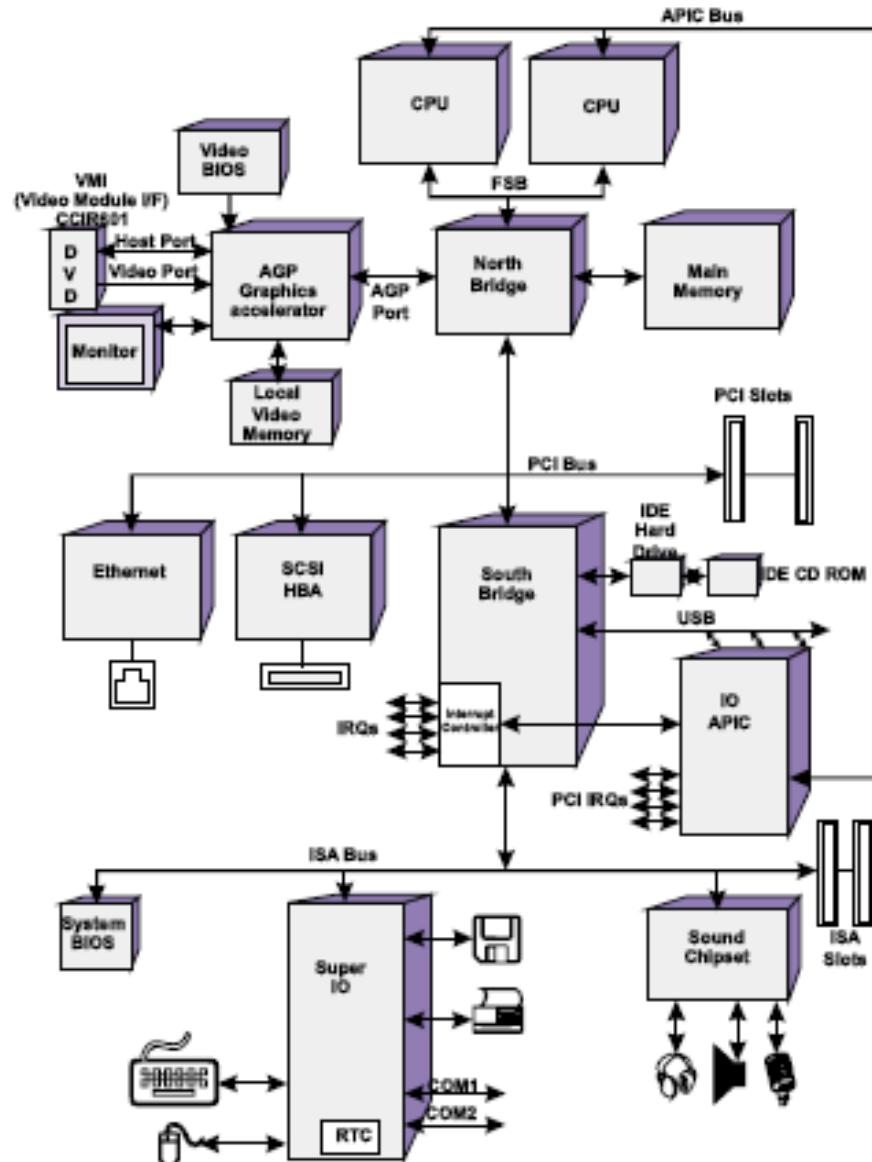
Поддержка многих управителей шины (Masters) и ведомых (Targets)/

Спецификации: 1.0 (1992), 2.0(1993), 2.1(1995), 2.2(1999)

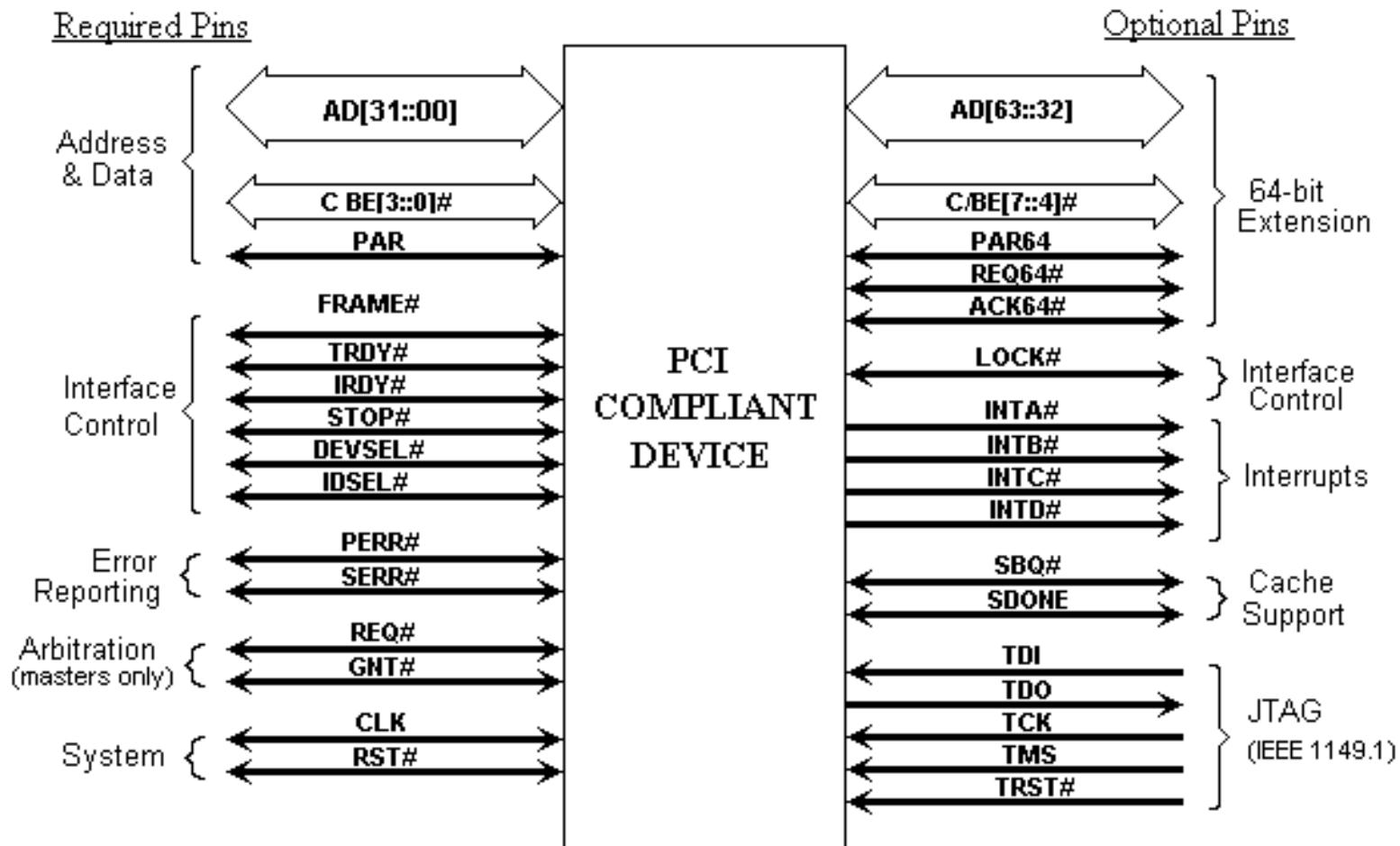
Локальная шина PCI - это высокопроизводительная 32-битная или 64-битная шина с мультиплексированными линиями адреса и данных. Она предназначена для использования в качестве связующего механизма между высокоинтегрированными периферийными контроллерами ввода-вывода, периферийными встраиваемыми платами и системами процессор/память.

Спецификация локальной шины PCI, реализация 2.0, включает протокол, электрическую, механическую и конфигурационную спецификации для локальной шины PCI и плат расширения. Описания электрических сигналов приводятся для напряжений питания 3.3В и 5.0В.

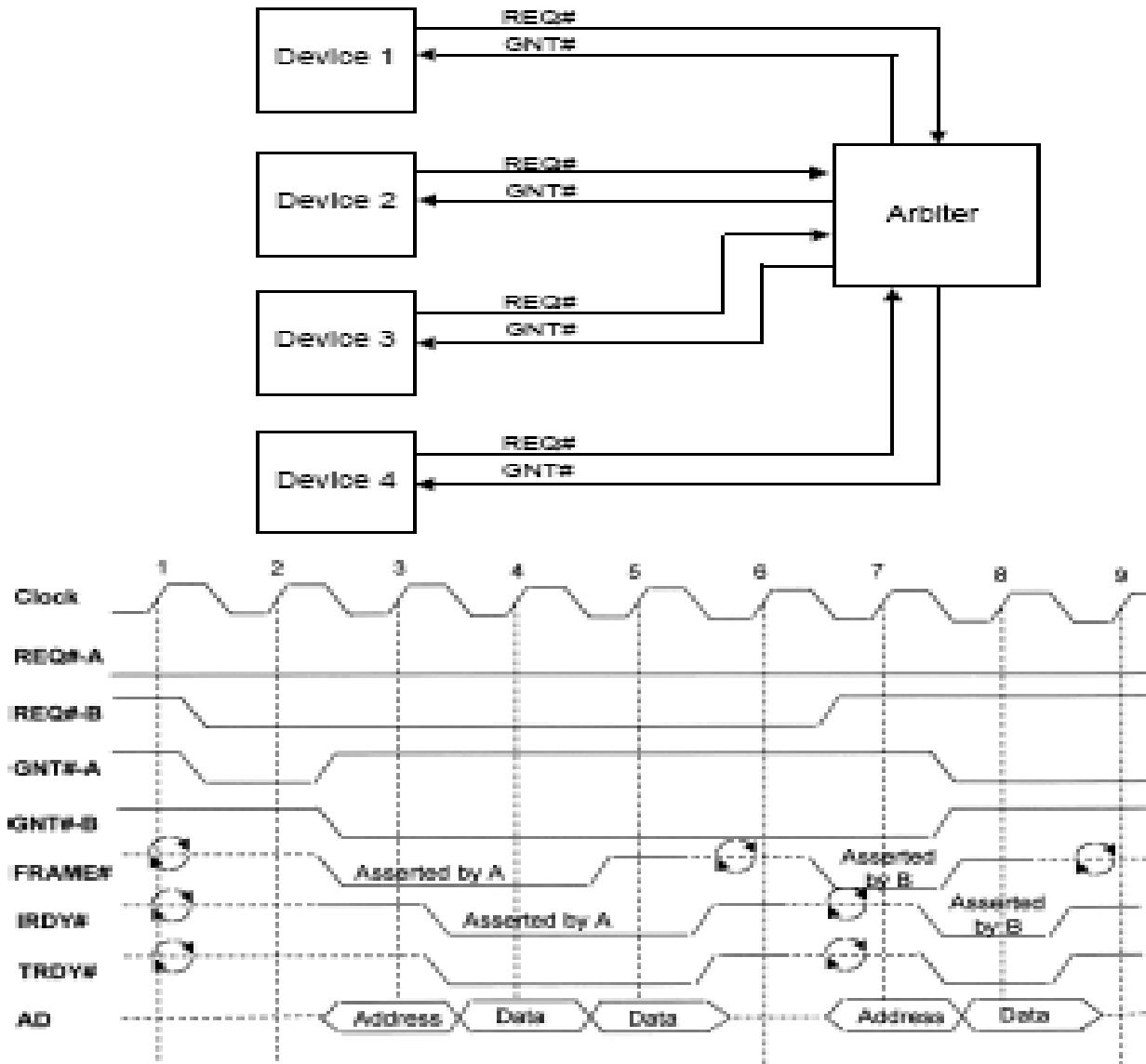
Пример построения систем на основе шины PCI*



Сигналы шины PCI



Арбитраж PCI шины



Операции на шине PCI

C/BE[3::0]#	Тип операции
0000	Interrupt Acknowledge (подтверждение прерывания)
0001	Special Cycle (специальный цикл)
0010	I/O Read (чтение при вводе - выводе)
0011	I/O Write (запись при вводе - выводе)
0100	Зарезервировано
0101	Зарезервировано
0110	Memory Read (чтение памяти)
0111	Memory Write (запись в память)
1000	Зарезервировано
1001	Зарезервировано
1010	Configuration Read (чтение конфигурации)
1011	Configuration Write (запись конфигурации)
1100	Memory Read Multiple (множественное чтение памяти)
1101	Dual Address Cycle (двойной цикл адреса)
1110	Memory read Line (линия чтения памяти)
1111	Memory Write and Invalidate (запись в память и недействительные данные)

FRAME# - Управляется мастером для того, чтобы он мог указать начало и конец транзакции.

IRDY# - Управляется мастером, чтобы он мог инициировать циклы ожидания.

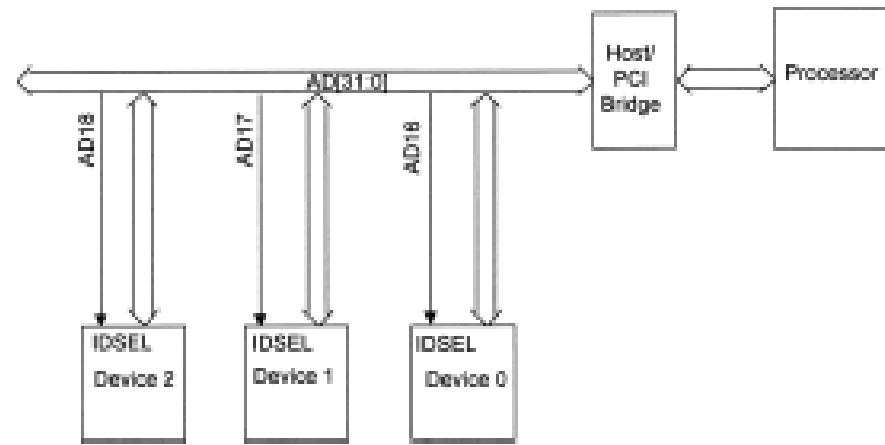
TRDY# - Управляется целевым устройством, чтобы оно могло инициировать циклы ожидания.

Адресация на PCI шине

Все определено три физических адресных пространства. Пространства адресов памяти и ввода-вывода объединены. Адресное пространство конфигураций было введено для обеспечения аппаратной конфигурации PCI.

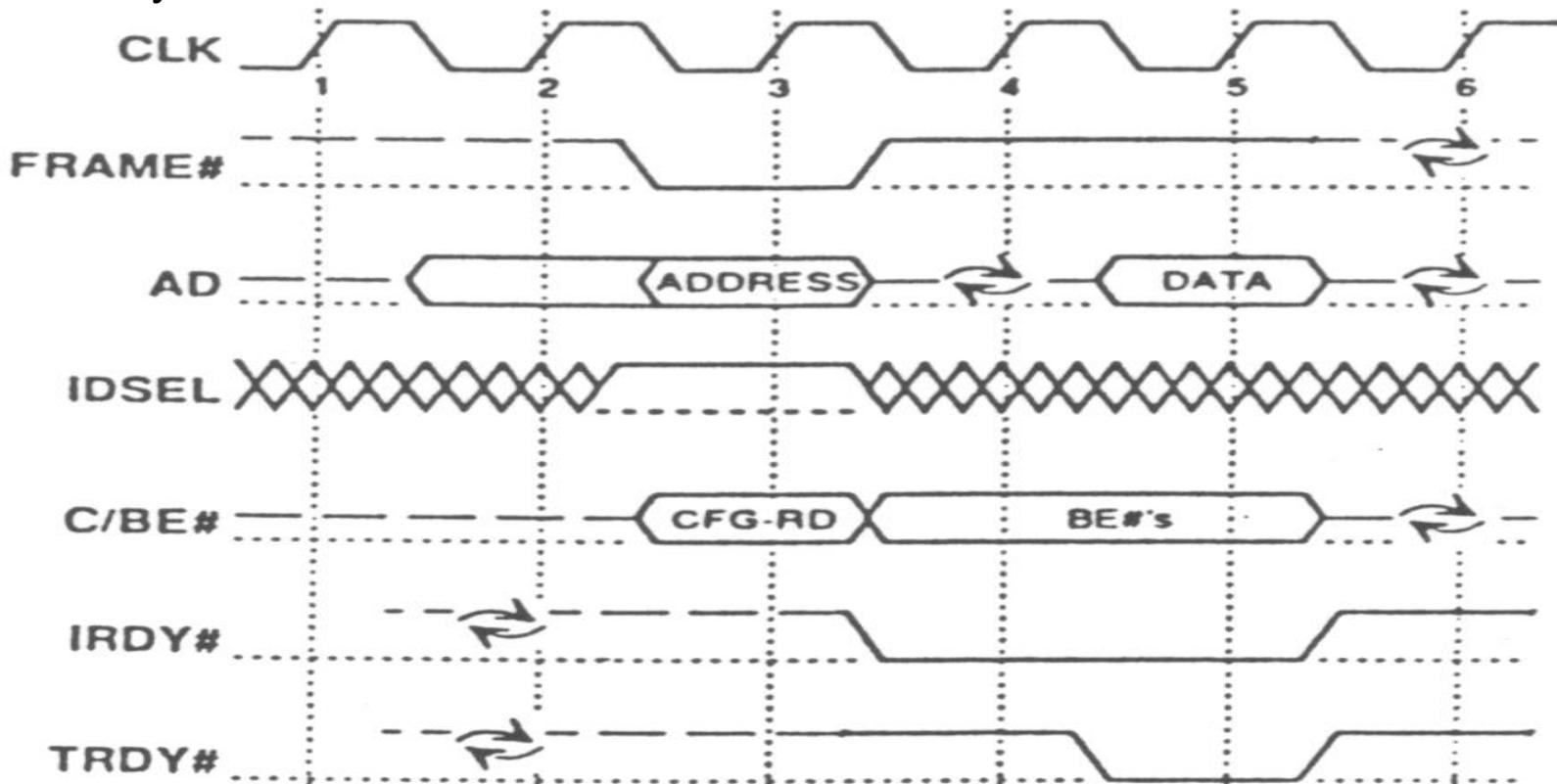
Дешифрирование адреса на шине PCI распределено; это означает, что оно выполняется на каждом устройстве. Это устраняет проблемы для центральной дешифрирующей логики, а также для сигналов выбора устройства, независимо от их использования для конфигурации. Каждый агент отвечает только на свой дешифрированный адрес.

PCI обеспечивает полную программно управляемую инициализацию и конфигурацию через отдельное адресное пространство конфигурации. PCI устройства должны обеспечить 256 байтов регистров конфигурации для этой цели.



Чтение конфигурации

Для поддержки иерархии PCI шины используются два типа доступа конфигурации. Тип 1 и тип 0 доступа конфигурации различаются значениями на AD[1::0]. Тип 0 цикла конфигурации (когда AD[1::0] = "00") используется, чтобы выбрать устройство на PCIшине, где цикл выполняется. Тип 1 цикла конфигурации (когда AD[1::0] = "01") используется, чтобы передать запрос конфигурации на другую PCIшину.

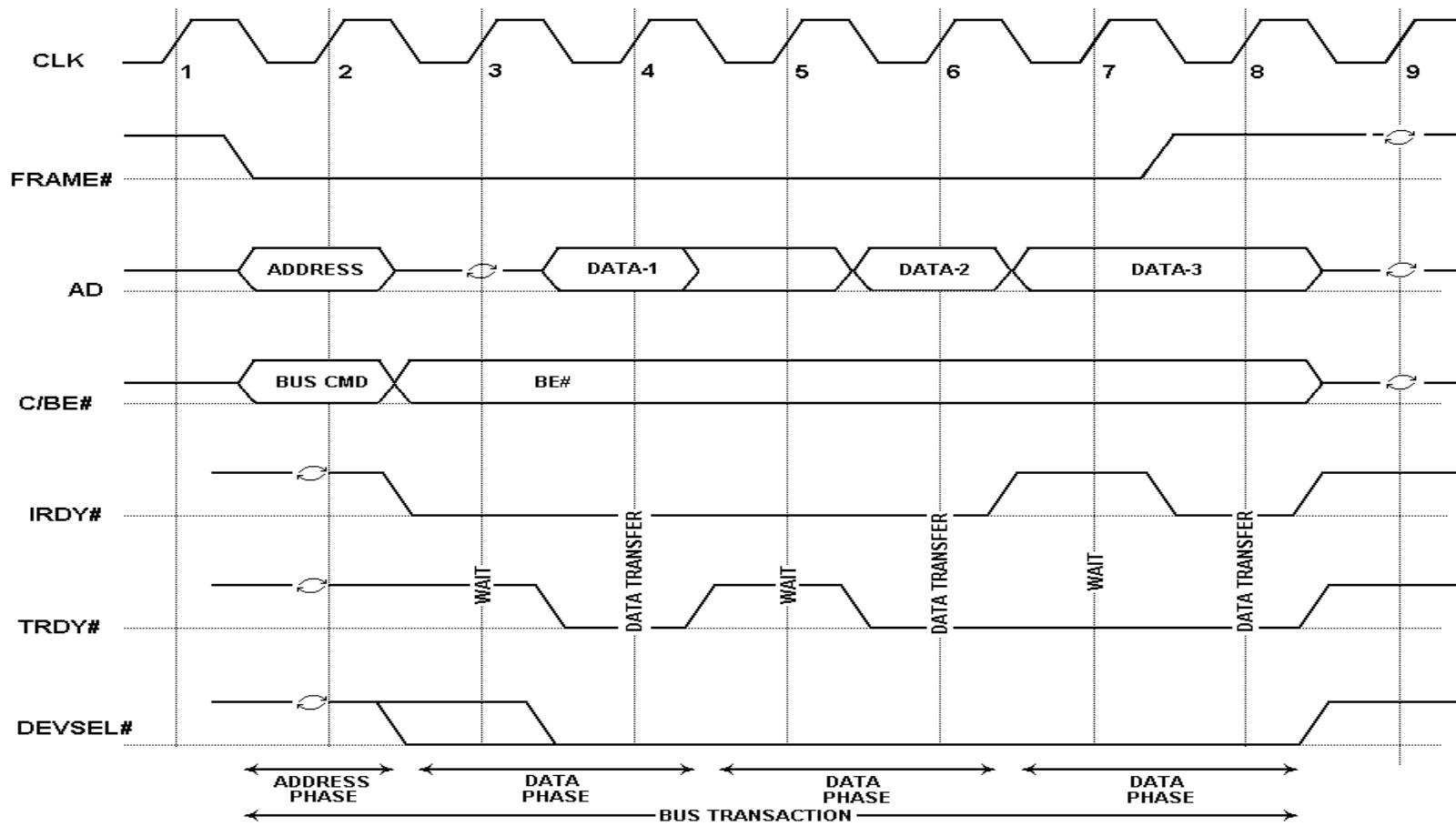


Пространство конфигурации Тип 0

31	16 15	0
Device ID	Vendor ID	00h
Status	Command	04h
Class Code	Revision ID	08h
BIST	Header Type	Cache Line Size
		0ch
		10h
		14h
		18h
	Base Address Registers	1Ch
		20h
		24h
	Cardbus CIS Pointer	28h
Subsystem ID	Subsystem Vendor ID	2ch
Expansion Bus ROM Base		30h
Reserved	Cap Pntr	34h
Reserved		38h
Max_Lat	Min_Gnt	Interrupt Pin
		Interrupt Line
		3Ch

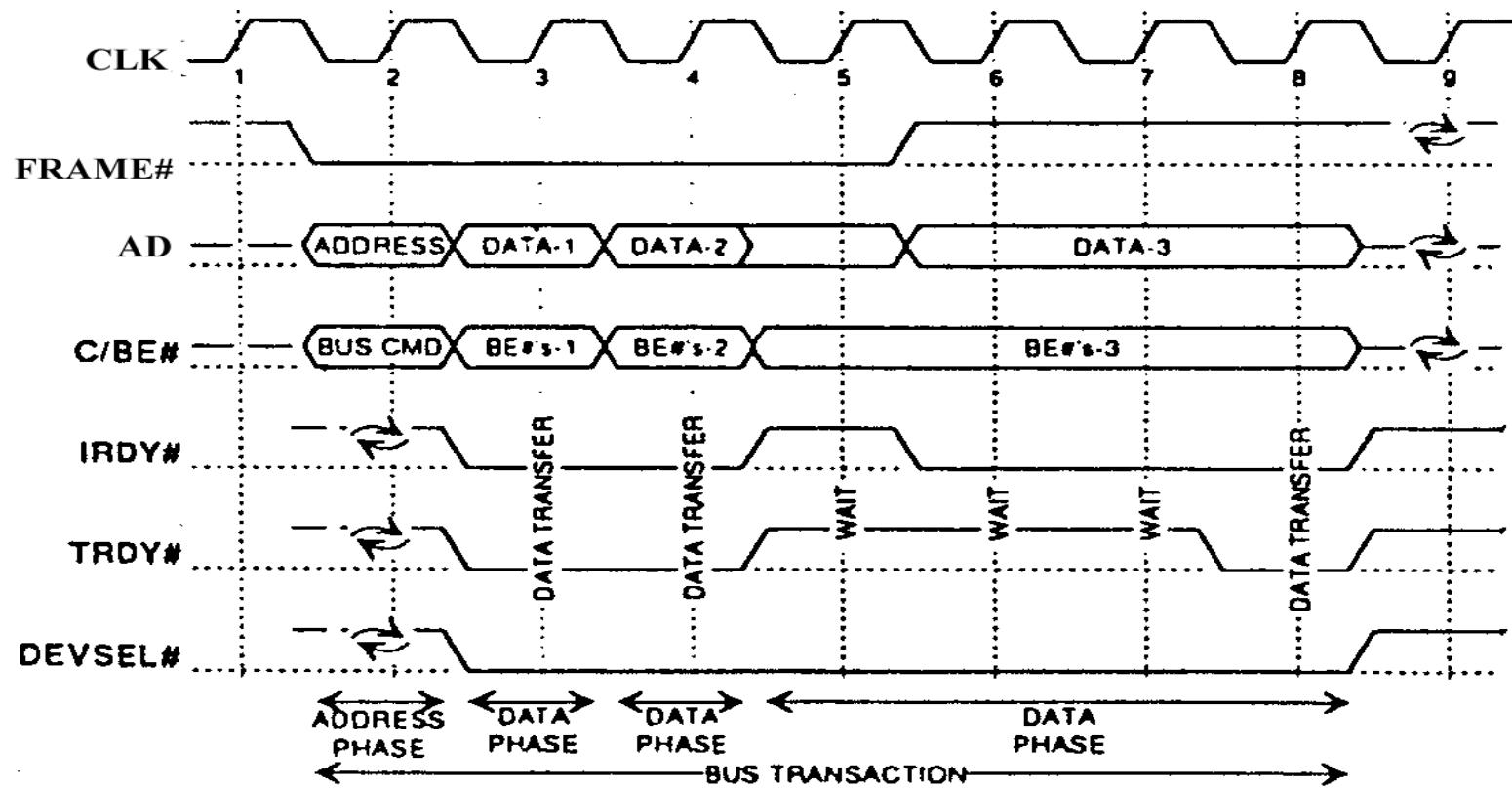
Базовая операция чтения

Транзакция чтения начинается фазой адреса, которая происходит во 2-ом такте, когда впервые устанавливается сигнал FRAME#. В течение фазы адреса AD[31::00] содержит допустимый адрес, а C/BE[3::0]# - допустимую команду шины.



Базовая операция записи

Транзакция записи начинается в такте 2, когда впервые устанавливается в активное состояние сигнал FRAME#. Транзакция записи подобна транзакции чтения, за исключением того, что после фазы адреса не требуется обратный цикл, так мастер обеспечивает и адрес, и данные. Фазы данных для транзакции записи такие, как и для транзакции чтения.

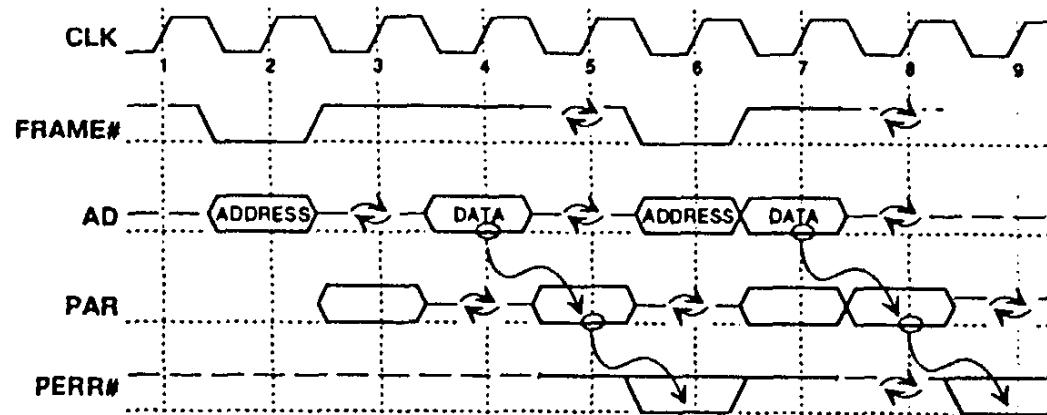


Контроль ошибок

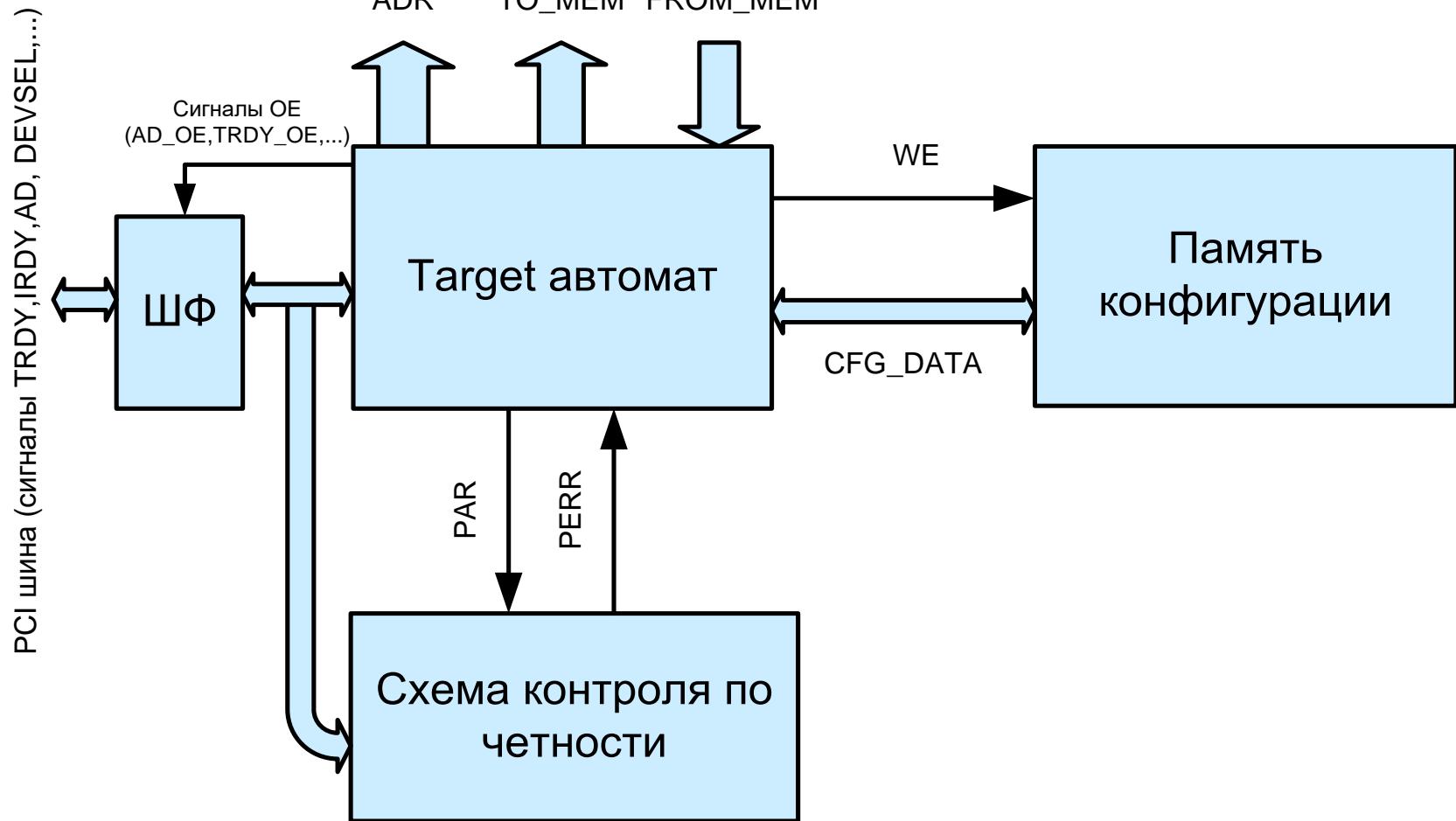
Выводы для сообщения об ошибках требуются всем устройствам:

PERR# s/t/s

Вывод Parity Error (ошибка контроля по четности) предназначен только для сообщения об ошибках контроля по четности во время всех транзакций PCI, за исключением специального цикла (Special Cycle). Вывод PERR# - три-стабильный и должен активно управляться агентом, получающим данные в течение двух тактов, после того, как обнаружена ошибка контроля данных по четности. Минимальная продолжительность PERR# - один такт для любой фазы данных, у которой обнаружена ошибка контроля данных по четности (если идут последовательно несколько фаз данных, каждая из которых имеет ошибку контроля данных по четности, то сигнал PERR# будет установлен за более, чем один такт). PERR# должен быть установлен в высокое состояние за один такт прежде, перед тем, как он перейдет в третье состояние со всеми соответствующими тристабильными сигналами. Не существует никаких специальных условий для случая, когда теряется ошибка контроля данных по четности или сообщается об отсроченной ошибке. Агент не может установить PERR#, пока он не разрешил доступ, установив DEVSEL# и завершив фазу данных.



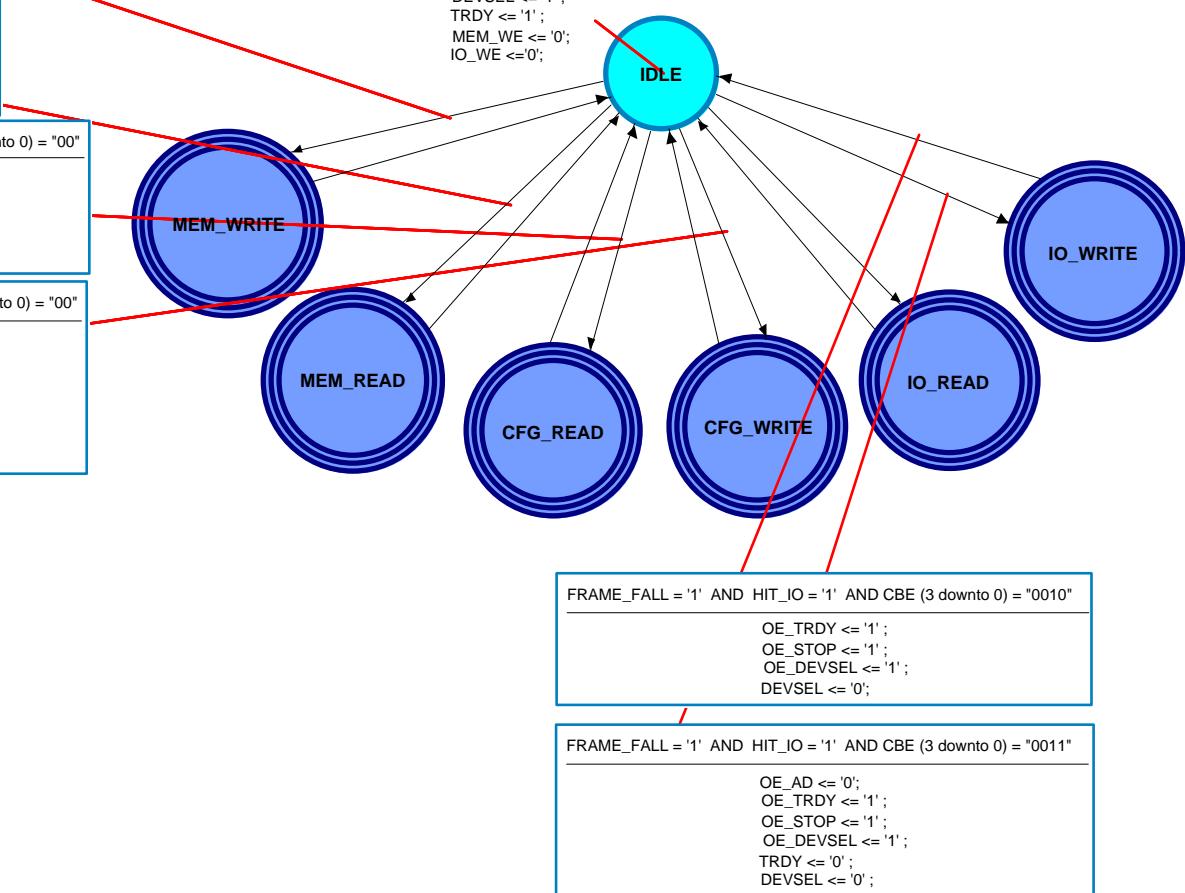
Пример PCI Target устройства



PCI Target автомат

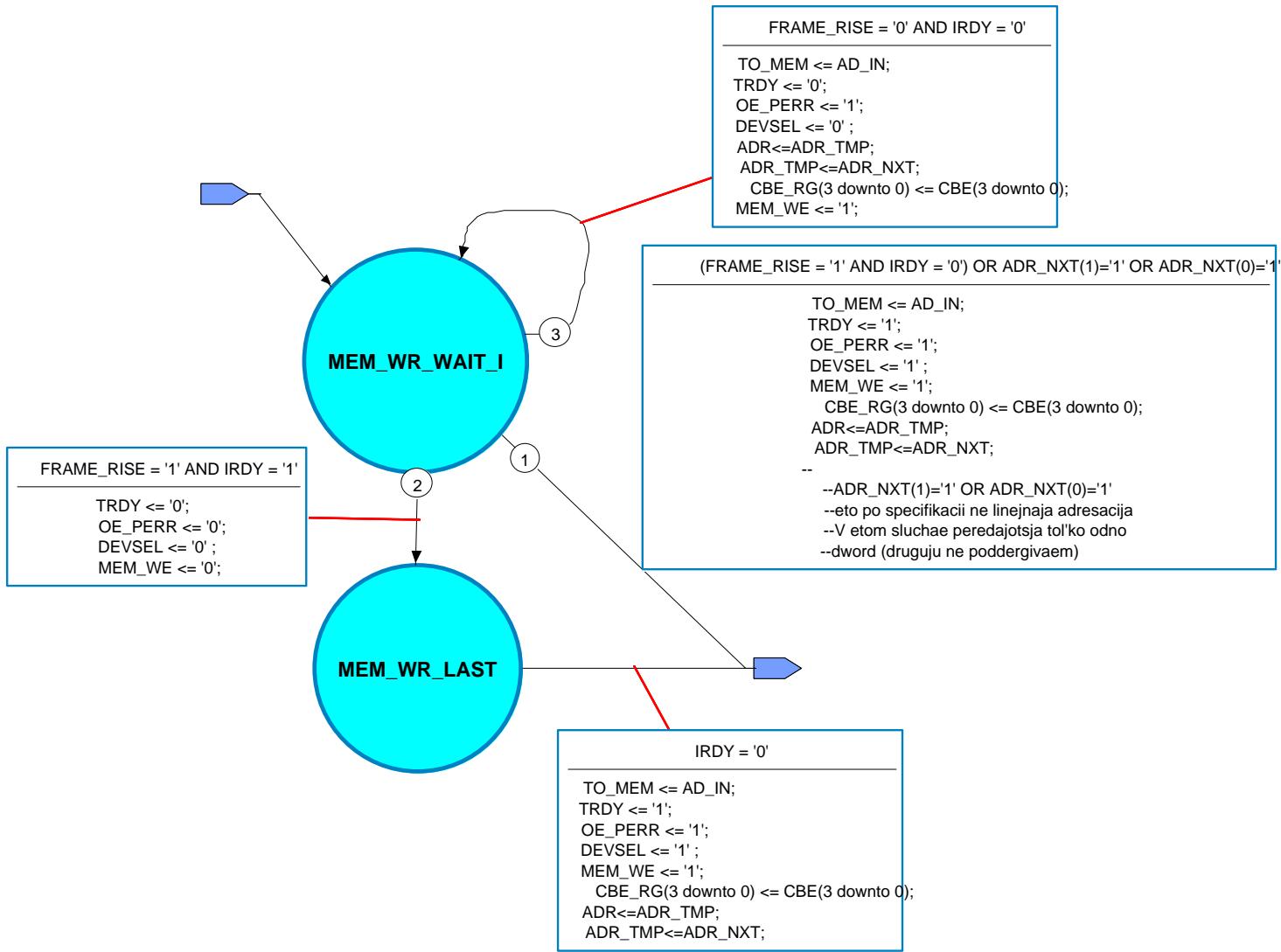
ПРИМЕР

FRAME_FALL = '1' AND HIT_MEM = '1' AND CBE (3 downto 0) = "0111" AND CRO='0'
OE_AD <= '0'; OE_TRDY <= '1'; OE_STOP <= '1'; OE_DEVSEL <= '1'; TRDY <= '0'; DEVSEL <= '0'; ADR_TMP <= AD_IN;
FRAME_FALL = '1' AND HIT_MEM = '1' AND CBE (3 downto 0) = "0110" AND CRO='0'
OE_TRDY <= '1'; OE_STOP <= '1'; OE_DEVSEL <= '1'; DEVSEL <= '0'; ADR <= AD_IN;
FRAME_FALL = '1' AND CBE (3 downto 0) = "1010" AND IDSEL = '1' AND AD_IN (1 downto 0) = "00"
ADR <= AD_IN; OE_TRDY <= '1'; OE_STOP <= '1'; OE_DEVSEL <= '1'; DEVSEL <= '0';
FRAME_FALL = '1' AND CBE (3 downto 0) = "1011" AND IDSEL = '1' AND AD_IN (1 downto 0) = "00"
ADR <= AD_IN; OE_AD <= '0'; OE_TRDY <= '1'; OE_STOP <= '1'; OE_DEVSEL <= '1'; TRDY <= '0'; DEVSEL <= '0';



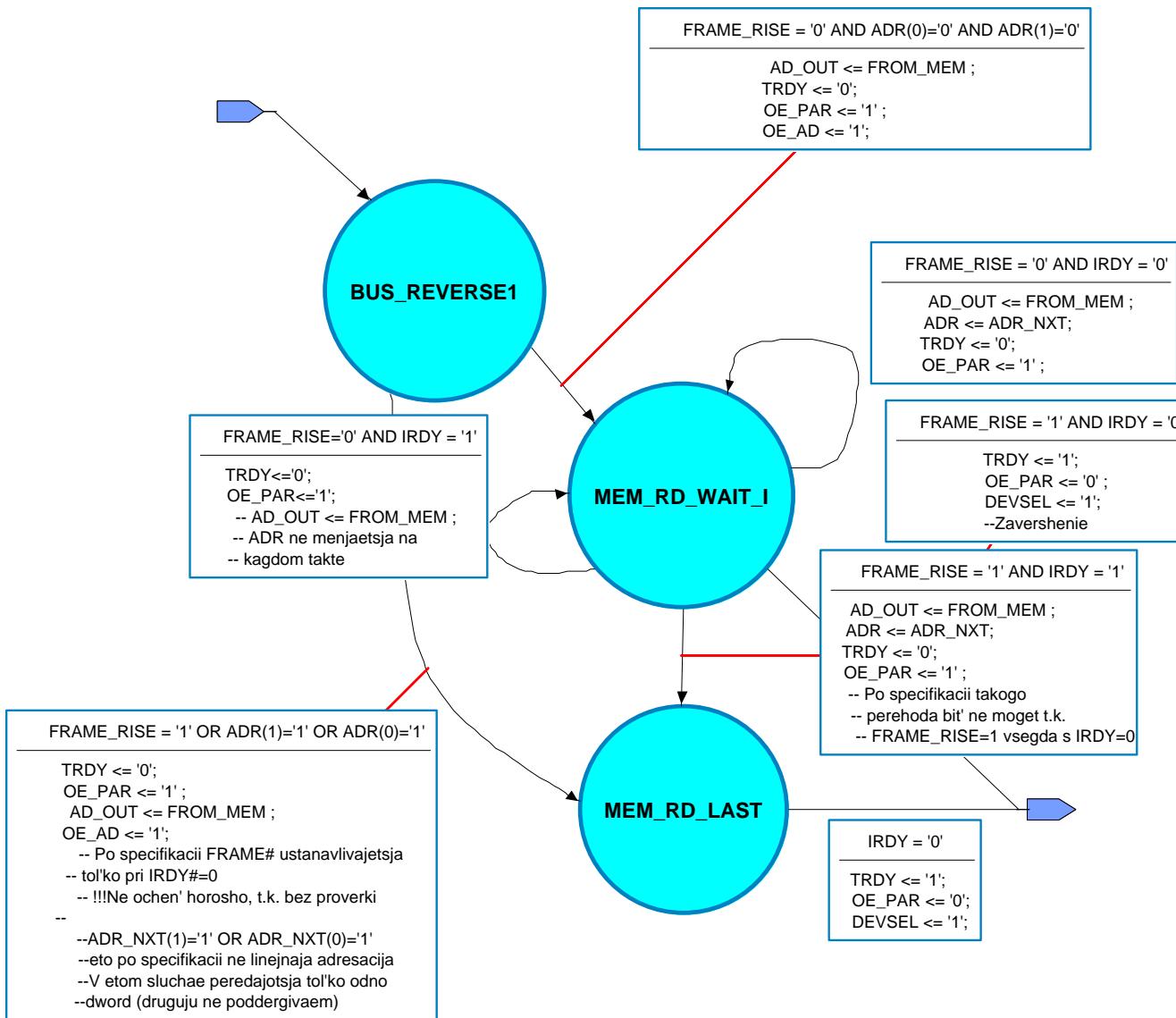
Memory Write

ПРИМЕР



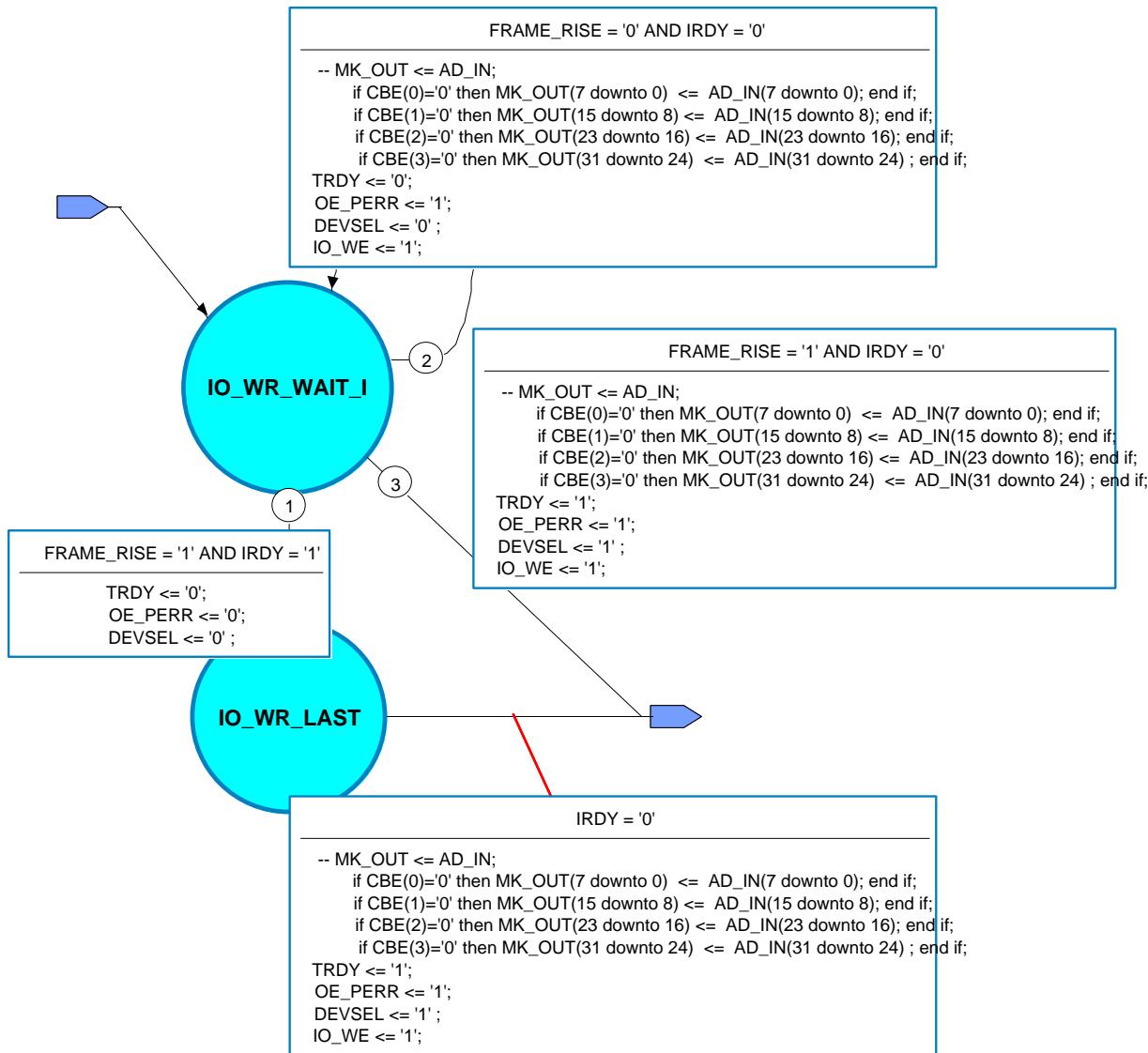
Memory Read

ПРИМЕР



IO_WRITE

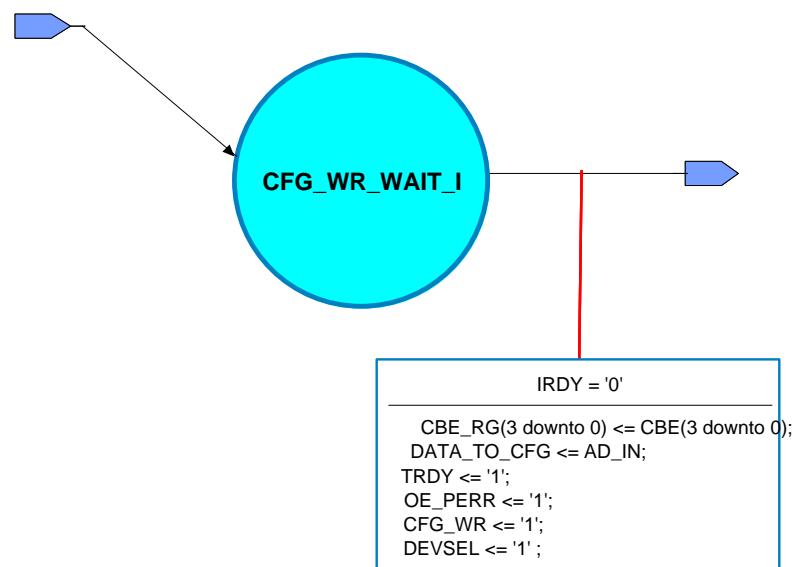
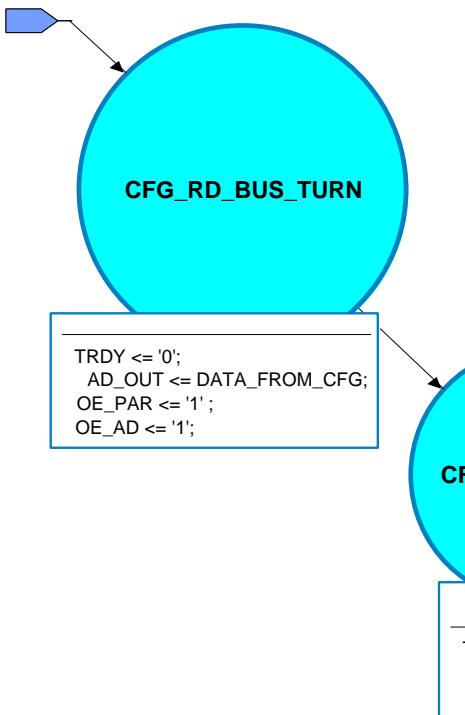
ПРИМЕР



ПРИМЕР

CFG_READ

CFG_WRITE



Пространство конфигурации

```
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.STD_LOGIC_UNSIGNED.all;
library WORK;

entity CFG_MEM is
  port(
    CLK      : in      std_logic;
    RST      : in      std_logic;
    ADR      : in      std_logic_vector(3 downto 0);
    D        : in      std_logic_vector(31 downto 0);
    WE       : in      std_logic;
    CBE      : in      std_logic_vector(3 downto 0);
    AD_IN    : in      std_logic_vector(31 downto 0);
    PERR     : in      std_logic;
    PERR_OE  : in      std_logic;
    Q        : out     std_logic_vector(31 downto 0);
    PERR_Enable : out  std_logic;
    HIT_IO   : out     std_logic;
    HIT_MEM  : out     std_logic
  );
end CFG_MEM;
```

```
architecture RTL of CFG_MEM is
subtype R_DWORD is std_logic_vector(31 downto 0);
subtype R_HWORD is std_logic_vector(23 downto 0);
subtype R_WORD is std_logic_vector(15 downto 0);
subtype R_BYT E is std_logic_vector(7 downto 0);
constant Z_DWORD :R_DWORD := "00000000000000000000000000000000";
constant DeviceID :R_WORD := "0000000000000001";
constant VendorID :R_WORD := "0010001100100011";--RO :2323h
signal Command :R_WORD;
signal Status :R_WORD;
constant RevisionID :R_BYT E := "00000001";
constant ClassCode :R_HWORD := "000100010000000000000000";
constant CashLineSize :R_BYT E := "00000000";
constant LatencyTimer :R_BYT E := "00000000";
constant HeaderType :R_BYT E := "00000000";
constant BIST :R_BYT E := "00000000";
signal BAR0 :R_DWORD;--IO BAR0
signal BAR1 :R_DWORD;--MEM BAR1
constant BAR2 :R_DWORD := "00000000000000000000000000000000"; --RW
constant BAR3 :R_DWORD := "00000000000000000000000000000000"; --RW
constant BAR4 :R_DWORD := "00000000000000000000000000000000"; --RW
constant BAR5 :R_DWORD := "00000000000000000000000000000000";
constant CardbusCISPointer :R_DWORD := "00000000000000000000000000000000";
constant SubsystemVendorID :R_WORD := "0000000000000000";
constant SubsystemID :R_WORD := "0000000000000000";
constant ExpansionBusROMBase :R_DWORD := "00000000000000000000000000000000";
constant CapPntr :R_BYT E := "00000000";
constant InterruptLine :R_BYT E := "00000000";
constant InterruptPin :R_BYT E := "00000000";
constant MinGnt :R_BYT E := "00000000";
constant MaxLat :R_BYT E := "00000000";
```

```
begin  
  
CFG_RD:process(ADR,Command,Status,BAR0)  
begin  
    case ADR is  
        when "0000" => Q <= DeviceID & VendorID;  
        when "0001" => Q <= Status & Command;  
        when "0010" => Q <= ClassCode & RevisionID;  
        when "0011" => Q <= BIST & HeaderType & LatencyTimer & CashLineSize;  
        when "0100" => Q <= BAR0;  
        when "0101" => Q <= BAR1;  
        when "0110" => Q <= BAR2;  
        when "0111" => Q <= BAR3;  
        when "1000" => Q <= BAR4;  
        when "1001" => Q <= BAR5;  
        when "1010" => Q <= CardbusCISPointer;  
        when "1011" => Q <= SubsystemID & SubsystemVendorID;  
        when "1100" => Q <= ExpansionBusROMBase;  
        when "1101" => Q <= Z_DWORD;  
        when "1110" => Q <= Z_DWORD;  
        when "1111" => Q <= MaxLat & MinGnt & InterruptPin & InterruptLine;  
        when others => NULL;  
    end case;  
end process;
```

```
CFG_WR:      process(ADR,CLK,CBE,RST,WE,D,PERR,PERR_OE)
begin
if (RST = '0') then
    Command  <= "0000000001000000";  --RW
    Status    <= "00000000000000000000";  --RW
    BAR0     <= "11111111111111111111111111111101";
    BAR1     <= "111111111111111111111111111111000000";
elsif (CLK'event and CLK='1') then
    if WE = '1' then
        if ADR(3 downto 0) = "0001" then
            if CBE(0)='0' then
                Command(0)  <= D(0); --IO Space
                Command(1)  <= D(1); --Mem Space
                Command(6)  <= D(6); --Parity Error
            end if;
            if CBE(1)='0' then
                Command(8)  <= D(8); --SERR Enable
            end if;
        end if;
    end if;
end process;
```

```
    if CBE(2)='0' then Status(7 downto 0)  <= (Status(7 downto 0) and not D(23 downto 16)); end if;
    if CBE(3)='0' then Status(15 downto 8) <= (Status(15 downto 8) and not D(31 downto 24)); end if;
elsif ADR(3 downto 0) = "0100" then
    if CBE(0)='0' then BAR0(7 downto 2)  <= D(7 downto 2); end if;
    if CBE(1)='0' then BAR0(15 downto 8) <= D(15 downto 8); end if;
    if CBE(2)='0' then BAR0(23 downto 16) <= D(23 downto 16); end if;
    if CBE(3)='0' then BAR0(31 downto 24) <= D(31 downto 24); end if;
elsif ADR(3 downto 0) = "0101" then
    if CBE(0)='0' then BAR1(7 downto 6)  <= D(7 downto 6); end if;
    if CBE(1)='0' then BAR1(15 downto 8) <= D(15 downto 8); end if;
    if CBE(2)='0' then BAR1(23 downto 16) <= D(23 downto 16); end if;
    if CBE(3)='0' then BAR1(31 downto 24) <= D(31 downto 24); end if;
end if;
    end if;
    if PERR = '0' and PERR_OE = '1' then Status(15) <= '1'; end if;
end if;
end process;
IO_HIT_DETECT: process(AD_IN)
begin
if (BAR0(31 downto 2) = AD_IN(31 downto 2)) and (Command(0) = '1') then Hit_IO <= '1';
else Hit_IO <= '0'; end if;
end process;
MEM_HIT_DETECT: process(AD_IN)
begin
if (BAR1(31 downto 6) = AD_IN(31 downto 6)) and (Command(1) = '1') then Hit_MEM <= '1';
else Hit_MEM <= '0'; end if;
end process;
PERR_Enable <= Command(6);
end RTL;
```

Контроль по четности

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;
USE ieee.std_logic_arith.all;

ENTITY Parity_PRT IS
PORT(
    CLK          : IN      std_logic;
    RST          : IN      std_logic;
    AD_IN        : IN      std_logic_vector (31 DOWNTO 0);
    AD_OUT       : IN      std_logic_vector (31 DOWNTO 0);
    CBE          : IN      std_logic_vector (3 DOWNTO 0);
    PAR_IN       : IN      std_logic;
    PERR_IN      : IN      std_logic;
    OE_AD         : IN      std_logic;
    OE_PAR        : IN      std_logic;
    OE_PAR1       : OUT     std_logic;
    OE_PERR        : IN      std_logic;
    OE_PERR2       : OUT     std_logic;
    PAR_OUT       : OUT     std_logic;
    PERR_OUT      : OUT     std_logic;
    STOP          : OUT     std_logic;
    PERR_EN        : IN      std_logic
);
END Parity_PRT;
```

ARCHITECTURE PRT OF Parity_PRT IS

```
SIGNAL PAR_NEW : std_logic;
SIGNAL P_IN: std_logic_vector (8 downto 0);
SIGNAL CBE_PAR: std_logic;
SIGNAL P_OUT: std_logic_vector (8 downto 0);
-- SIGNAL OE_PERR1 : std_logic;
```

BEGIN

```
P0: CBE_PAR <= CBE(0) xor CBE(1) xor CBE(2) xor CBE(3);
PC0: P_IN(0) <= AD_IN(0) xor AD_IN(1) xor AD_IN(2) xor AD_IN(3);
PC1: P_IN(1) <= AD_IN(4) xor AD_IN(5) xor AD_IN(6) xor AD_IN(7);
PC2: P_IN(2) <= AD_IN(8) xor AD_IN(9) xor AD_IN(10) xor AD_IN(11);
PC3: P_IN(3) <= AD_IN(12) xor AD_IN(13) xor AD_IN(14) xor AD_IN(15);
PC4: P_IN(4) <= AD_IN(16) xor AD_IN(17) xor AD_IN(18) xor AD_IN(19);
PC5: P_IN(5) <= AD_IN(20) xor AD_IN(21) xor AD_IN(22) xor AD_IN(23);
PC6: P_IN(6) <= AD_IN(24) xor AD_IN(25) xor AD_IN(26) xor AD_IN(27);
PC7: P_IN(7) <= AD_IN(28) xor AD_IN(29) xor AD_IN(30) xor AD_IN(31);
PC8: P_IN(8) <= P_IN(0) xor P_IN(1) xor P_IN(2) xor P_IN(3) xor P_IN(4) xor
P_IN(5) xor P_IN(6) xor P_IN(7) xor CBE_PAR;
```

```
PG0: P_OUT(0) <= AD_OUT(0) xor AD_OUT(1) xor AD_OUT(2) xor AD_OUT(3);  
PG1: P_OUT(1) <= AD_OUT(4) xor AD_OUT(5) xor AD_OUT(6) xor AD_OUT(7);  
PG2: P_OUT(2) <= AD_OUT(8) xor AD_OUT(9) xor AD_OUT(10) xor AD_OUT(11);  
PG3: P_OUT(3) <= AD_OUT(12) xor AD_OUT(13) xor AD_OUT(14) xor AD_OUT(15);  
PG4: P_OUT(4) <= AD_OUT(16) xor AD_OUT(17) xor AD_OUT(18) xor AD_OUT(19);  
PG5: P_OUT(5) <= AD_OUT(20) xor AD_OUT(21) xor AD_OUT(22) xor AD_OUT(23);  
PG6: P_OUT(6) <= AD_OUT(24) xor AD_OUT(25) xor AD_OUT(26) xor AD_OUT(27);  
PG7: P_OUT(7) <= AD_OUT(28) xor AD_OUT(29) xor AD_OUT(30) xor AD_OUT(31);  
PG8: P_OUT(8) <= P_OUT(0) xor P_OUT(1) xor P_OUT(2) xor P_OUT(3) xor P_OUT(4) xor  
P_OUT(5) xor P_OUT(6) xor P_OUT(7) xor CBE_PAR;
```

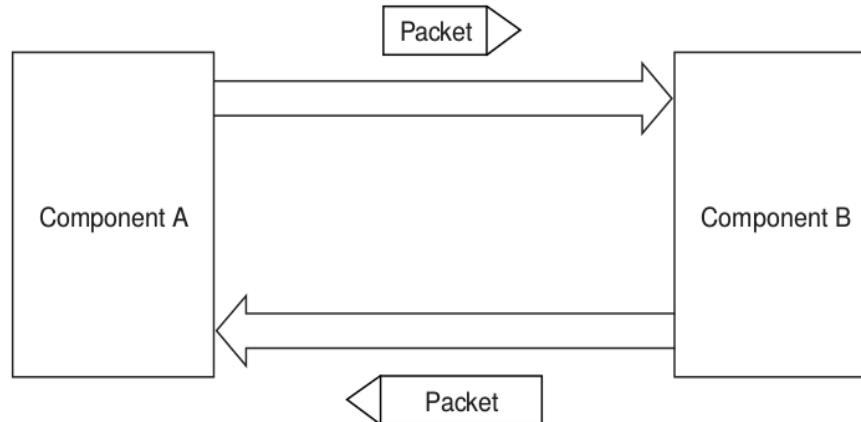
```
NXT_PAR_OUT: process (CLK,RST,OE_PAR,P_OUT(8))
begin
if RST='0' then
    PAR_OUT <='0';
elsif (CLK'event and CLK = '1') then
    PAR_OUT <= P_OUT(8);
    OE_PAR1 <= OE_PAR;
    end if;
end process;
CHK_PAR_IN: process (CLK,RST,OE_PERR,PAR_NEW)
begin
if RST='0' then
    PERR_OUT <='1';
elsif (CLK'event and CLK = '1') then
    PERR_OUT <= (PAR_NEW xor PAR_IN) nand PERR_EN;
--    (PAR_NEW xor PAR_IN) (PERR_EN) (PERR_OUT)
--    1 1 0
--    0 1 1
--    1 0 1
--    0 0 1
    end if;
end process;
```

```
process (CLK,RST,OE_PERR,P_IN(8),PAR_IN)
begin
if RST='0' then
PAR_NEW <= '1';
    OE_PERR2 <='0';
elsif (CLK'event and CLK = '1') then
PAR_NEW <= P_IN(8);
    OE_PERR2 <= OE_PERR;
end if;
end process;
END PRT;
```

Шина PCI express

Преимущества последовательных шин и интерфейсов:

- Большой акцент на сложную логику при простой топологии физического уровня;
- Перспектива перехода на оптический физический уровень;
- Экономия пространства печатных плат и снижение сложности монтажа;
- Простота реализации PnP и динамическую конфигурацию в любом смысле;
- Возможность выделять гарантированные и изохронные каналы;
- Переход от разделяемых шин с арбитражем к более предсказуемым соединениям точка-точка;
- Лучшая с точки зрения затрат и более гибкая с точки зрения топологии масштабируемость;



Сравнение пропускной способности шин семейства PCI

PCI и PCI-X

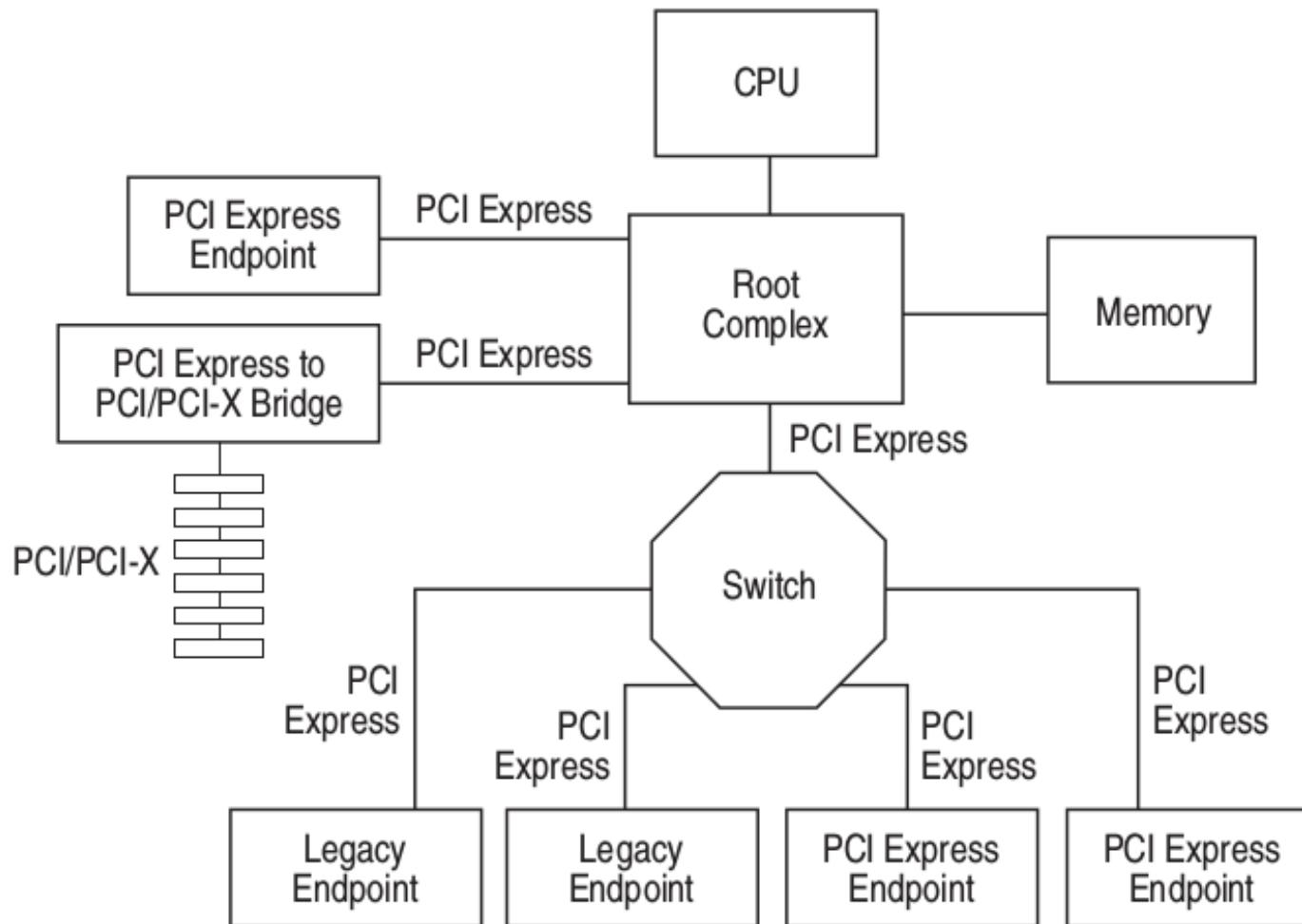
Bus Type	Clock Frequency	Peak Bandwidth *	Number of Card Slots per Bus
PCI 32-bit	33 MHz	133 MBytes/sec	4-5
PCI 32-bit	66 MHz	266 MBytes/sec	1-2
PCI-X 32-bit	66 MHz	266 MBytes/sec	4
PCI-X 32-bit	133 MHz	533 MBytes/sec	1-2
PCI-X 32-bit	266 MHz effective	1066 MBytes/sec	1
PCI-X 32-bit	533 MHz effective	2131 MByte/sec	1

* Double all these bandwidth numbers for 64-bit bus implementations

PCI Express

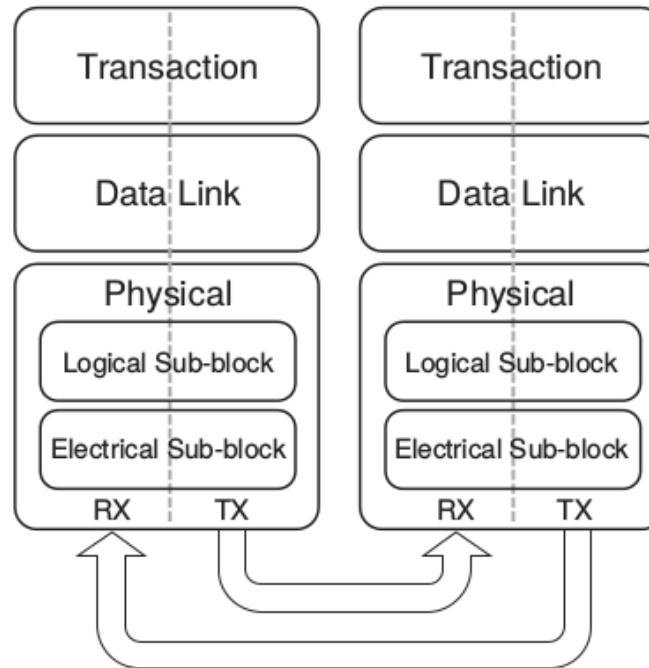
PCI Express version	Line code	Transfer rate ^[i]	Throughput ^[i]				
			x1	x2	x4	x8	x16
1.0	8b/10b	2.5 GT/s	250 MB/s	500 MB/s	1 GB/s	2 GB/s	4 GB/s
2.0	8b/10b	5.0 GT/s	500 MB/s	1 GB/s	2 GB/s	4 GB/s	8 GB/s
3.0	128b/130b	8.0 GT/s	984.6 MB/s	1.97 GB/s	3.94 GB/s	7.9 GB/s	15.8 GB/s
4.0	128b/130b	16.0 GT/s	1969 MB/s	3.94 GB/s	7.9 GB/s	15.8 GB/s	31.5 GB/s
5.0 ^{[30][31]} (expected in Q2 2019) ^[33]	128b/130b	32.0 GT/s ^[ii]	3938 MB/s	7.9 GB/s	15.8 GB/s	31.5 GB/s	63.0 GB/s

Принципы взаимодействия устройств по PCI Express



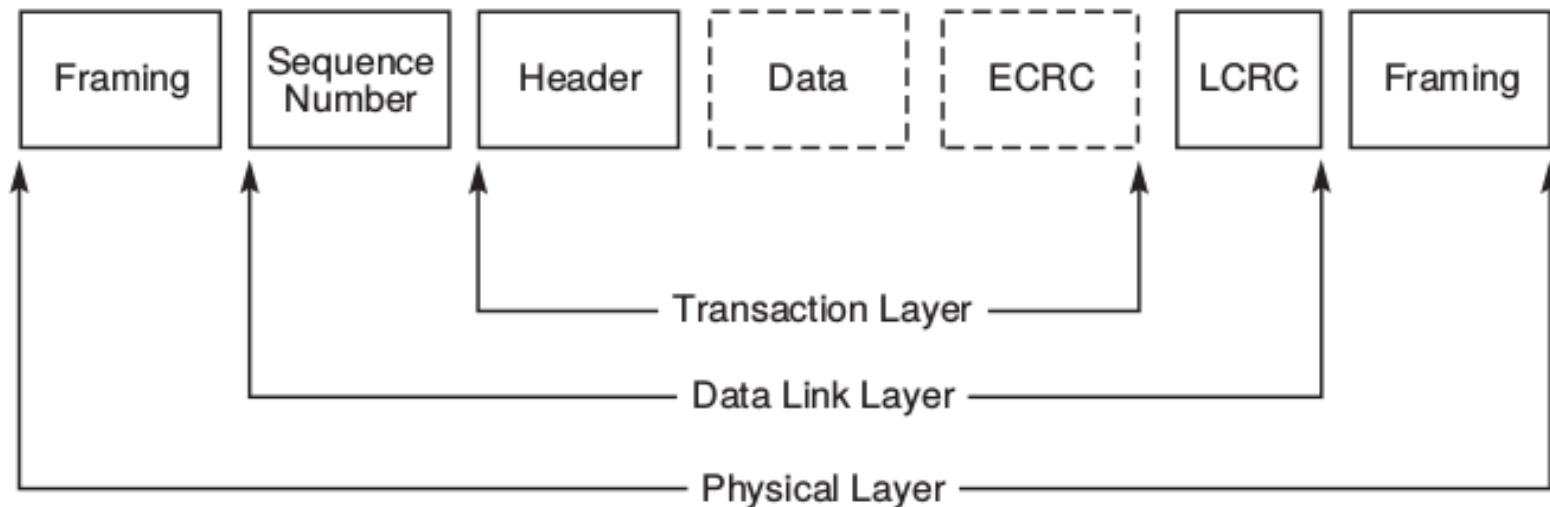
Соединение между двумя устройствами PCI Express называется link, и состоит из одного (называемого 1x) или нескольких (2x, 4x, 8x, 12x, 16x и 32x) двунаправленных последовательных соединений lane. Каждое устройство должно поддерживать соединение 1x.

Принципы взаимодействия устройств по PCI Express



- Уровень транзакций отвечает за взаимодействие с пригладным уровнем, сборку и разборку TLP пакетов (используются для передачи транзакций на нижние уровни, чтение и запись, передача событий). Уровень транзакций также отвечает за управление кредитами.
- Уровень линка данных принимает TLP пакеты от транзакционного уровня и осуществляет их передачу через физический уровень. Основные обязанности уровня линка данных включают управление линками и обеспечение целостности данных при передаче, включая обнаружение ошибок и исправление ошибок.
- Физический уровень включает в себя все необходимые схемы для работы интерфейса, параллельно-последовательные преобразователи, кодеры и декодеры 8/10, ФАПЧ(ы), схемы согласования сопротивлений.

Принципы взаимодействия устройств по PCI Express



- TLP, которые не проходят проверку целостности данных (LCRC и порядковый номер) или которые теряются при передаче из одного компонента в другой, повторно отправляются передатчиком.
- Передатчик хранит копию всех отправленных TLP, повторно отправляет эти копии, когда это необходимо, и очищает копии только тогда, когда он получает положительное подтверждение безошибочного получения от другого компонента.
- Если положительное подтверждение не было получено в течение указанного периода времени, Передатчик автоматически начнет повторную передачу.
- Приемник может запросить немедленную повторную передачу, используя отрицательное подтверждение NAK.

Принципы взаимодействия устройств по PCI Express

Transaction Type	Non-Posted or Posted
Memory Read	Non-Posted
Memory Write	Posted
Memory Read Lock	Non-Posted
IO Read	Non-Posted
IO Write	Non-Posted
Configuration Read (Type 0 and Type 1)	Non-Posted
Configuration Write (Type 0 and Type 1)	Non-Posted
Message	Posted

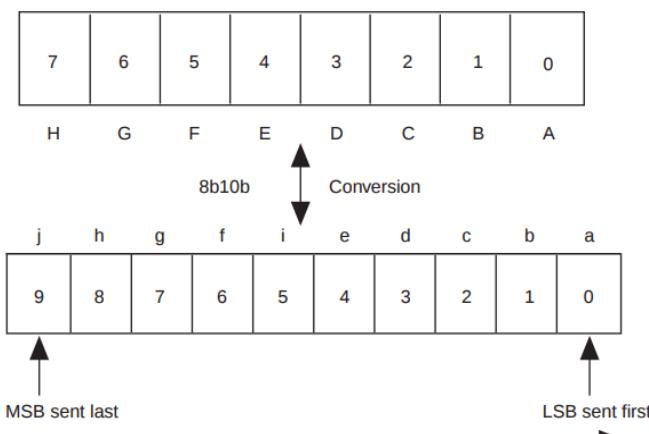
Участниками транзакции является Requester (инициатор) и Completer (исполнитель).

Транзакции Non-Posted предполагают получение ответа инициатором от исполнителя об успешном завершении.

Транзакция Posted не требует ответа от исполнителя.

Способы кодирования данных при приемопередаче

Кодирование 8/10 (PCIe v1.0, v2.0)



Всевозможных 10-битовых комбинаций больше, чем реально используется для представления 256 обычных символов. В наборе символов представлены числовые данные D и специальные символы K (control symbols).

Каждый символ имеет два образа: с положительным/отрицательным балансом нулевых и единичных бит (или с одинаковым количеством нулевых и единичных символов).

При получении 10-битовой последовательности битов, не

Encoding	Symbol	Name	Description
K28.5	COM	Comma	Used for Lane and Link initialization and management
K27.7	STP	Start TLP	Marks the start of a Transaction Layer Packet
K28.2	SDP	Start DLLP	Marks the start of a Data Link Layer Packet
K29.7	END	End	Marks the end of a Transaction Layer Packet or a Data Link Layer Packet
K30.7	EDB	EnD Bad	Marks the end of a nullified TLP
K23.7	PAD	Pad	Used in Framing and Link Width and Lane ordering negotiations
K28.0	SKP	Skip	Used for compensating for different bit rates for two communicating Ports
K28.1	FTS	Fast Training Sequence	Used within an Ordered Set to exit from L0s to L0
K28.3	IDL	Idle	Used in the Electrical Idle Ordered Set (EIOS)
K28.4			Reserved
K28.6			Reserved
K28.7	EIE	Electrical Idle Exit	Reserved in 2.5 GT/s Used in the Electrical Idle Exit Ordered Set (EIEOS) and sent prior to sending FTS at data rates other than 2.5 GT/s

СОСИ	Data Byte Name	Data Byte Value	Bits HGF EDCBA	Current RD - abcdei fghj	Current RD + abcdei fghj
K28.0	1C	000 11100	001111 0100	110000 1011	
K28.1	3C	001 11100	001111 1001	110000 0110	
K28.2	5C	010 11100	001111 0101	110000 1010	
K28.3	7C	011 11100	001111 0011	110000 1100	
K28.4	9C	100 11100	001111 0010	110000 1101	
K28.5	BC	101 11100	001111 1010	110000 0101	
K28.6	DC	110 11100	001111 0110	110000 1001	
K28.7	FC	111 11100	001111 1000	110000 0111	
K23.7	F7	111 10111	111010 1000	000101 0111	
K27.7	FB	111 11011	110110 1000	001001 0111	
K29.7	FD	111 11101	101110 1000	010001 0111	
K30.7	FE	111 11110	011110 1000	100001 0111	

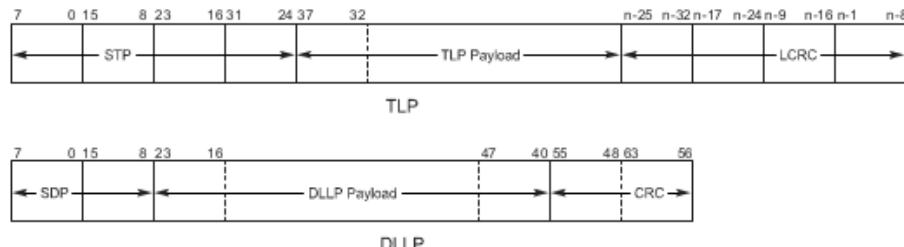
Physical Layer Protocol

Формат пакетов TS1 на Physical Layer

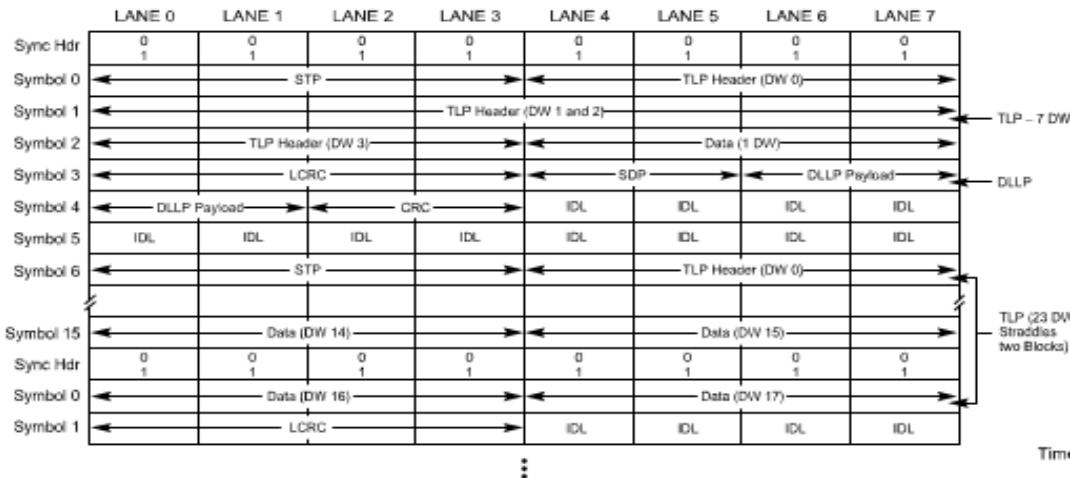
Table 4-5: TS1 Ordered Set

Symbol Number	Description
0	When operating at 2.5 or 5.0 GT/s: COM (K28.5) for Symbol alignment. When operating at 8.0 GT/s or above: Encoded as 1Eh (TS1 Ordered Set).
1	Link Number. Ports that do not support 8.0 GT/s or above: 0-255, PAD. Downstream Ports that support 8.0 GT/s or above: 0-31, PAD. Upstream Ports that support 8.0 GT/s or above: 0-255, PAD. When operating at 2.5 or 5.0 GT/s: PAD is encoded as K23.7. When operating at 8.0 GT/s or above: PAD is encoded as F7h.
2	Lane Number within Link. When operating at 2.5 or 5.0 GT/s: 0-31, PAD. PAD is encoded as K23.7. When operating at 8.0 GT/s or above: 0-31, PAD. PAD is encoded as F7h.
3	N_FTS. The number of Fast Training Sequences required by the Receiver: 0-255.
4	Data Rate Identifier Bit 0 – Reserved Bit 1 – 2.5 GT/s Data Rate Supported. Must be set to 1b. Bit 2 – 5.0 GT/s Data Rate Supported. Must be set to 1b if Bit 3 is 1b. Bit 3 – 8.0 GT/s Data Rate Supported. Bit 4:5 – Reserved. Bit 6 – Autonomous Change>Selectable De-emphasis. Downstream Ports: This bit is defined for use in the following LTSSM states: Polling.Active, Configuration.LinkWidth.Start, and Loopback.Entry. In all other LTSSM states, it is Reserved. Upstream Ports: This bit is defined for use in the following LTSSM states: Polling.Active, Configuration, Recovery, and Loopback.Entry. In all other LTSSM states, it is Reserved. Bit 7 – speed_change. This bit can be set to 1b only in the Recovery.RcvrLock LTSSM state. In all other LTSSM states, it is Reserved.
5	Training Control Bit0 – <u>Hol Reset</u> Bit 0 = 0b, De-assert Bit 0 = 1b, Assert Bit1 – <u>Disable Link</u> Bit 1 = 0b, De-assert Bit 1 = 1b, Assert Bit2 – <u>Loopback</u> Bit 2 = 0b, De-assert Bit 2 = 1b, Assert Bit3 – <u>Disable Scrambling in 2.5 GT/s and 5.0 GT/s data rates; Reserved in other data rates</u> Bit 3 = 0b, De-assert Bit 3 = 1b, Assert Bit4 – <u>Compliance Receive</u> Bit 4 = 0b, De-assert Bit 4 = 1b, Assert Ports that support 5.0 GT/s and above data rate(s) must implement the Compliance Receive bit. Ports that support only 2.5 GT/s data rate may optionally implement the Compliance Receive bit. If not implemented, the bit is Reserved. Bit 5:7 – Reserved

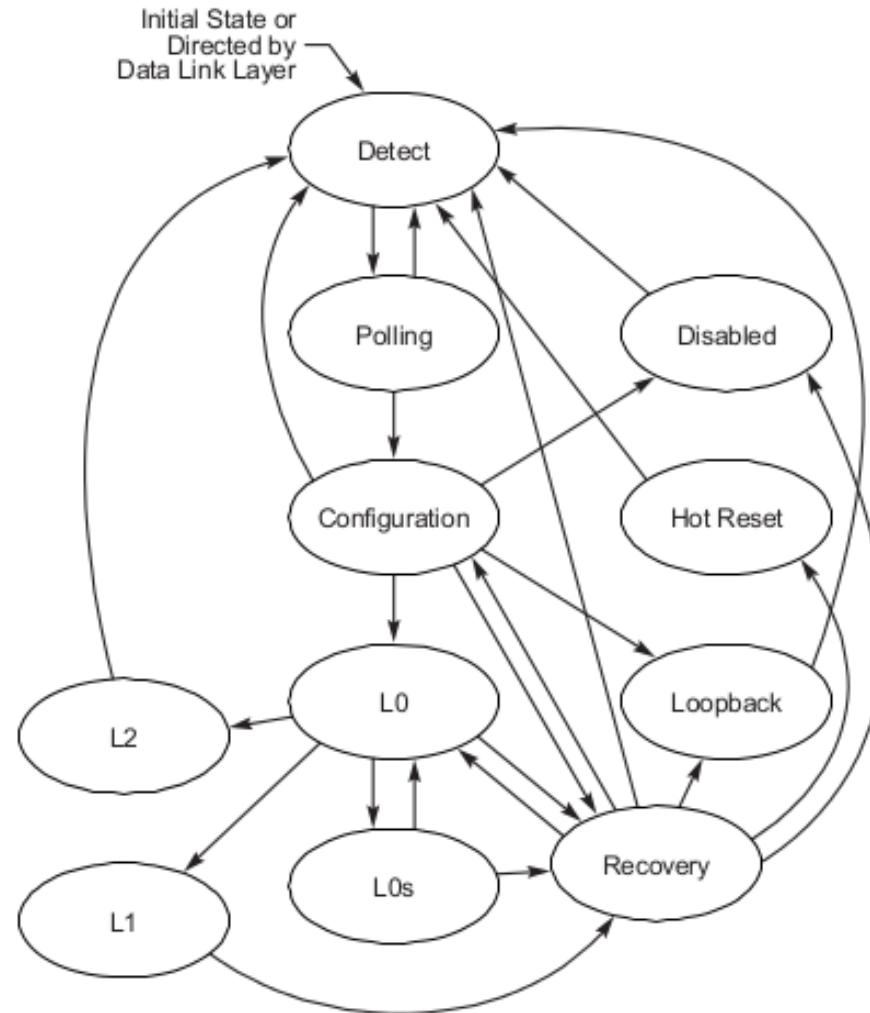
Формирование пакетов DLLP и TLP уровней



Пример передачи пакетов на PHY уровне



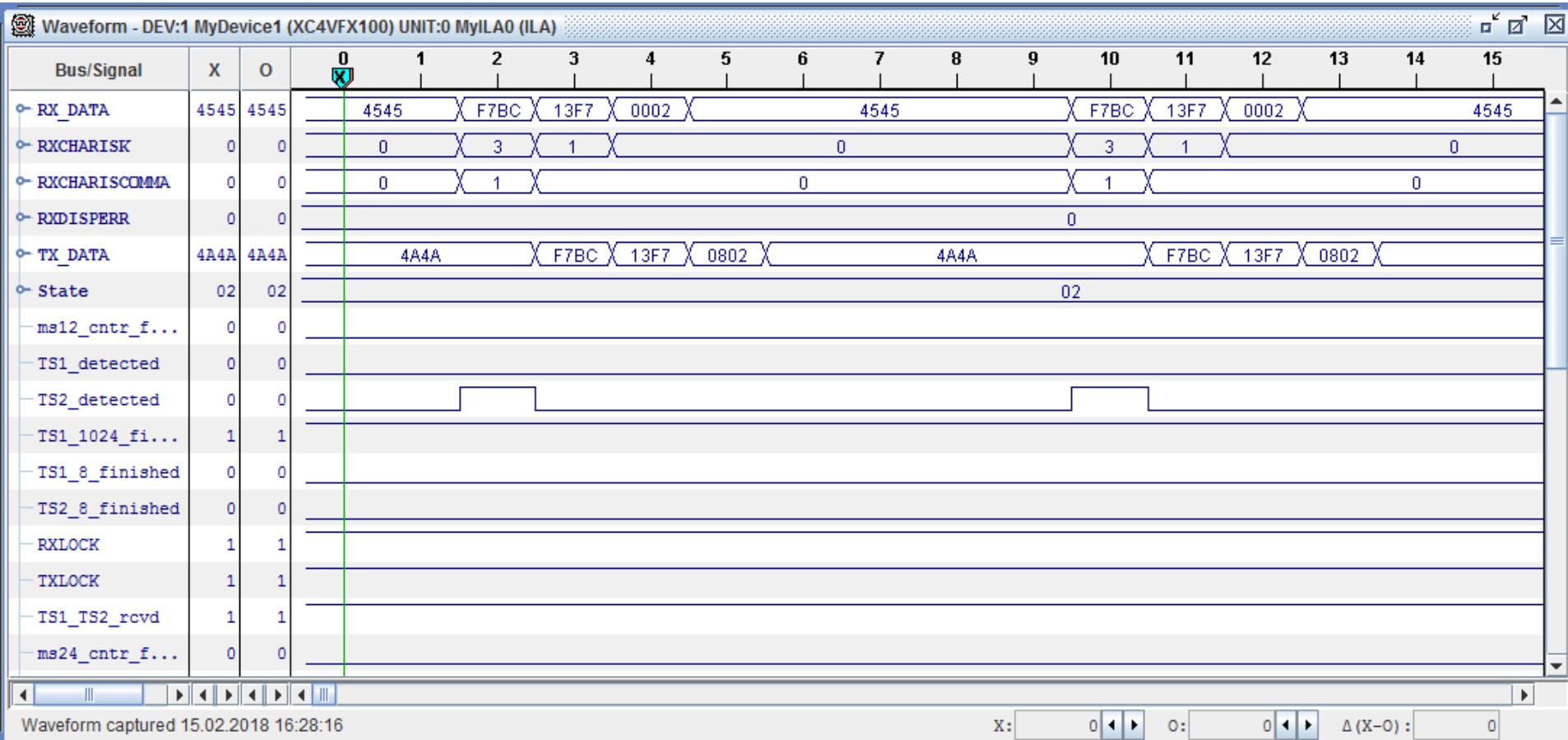
Инициализация физического соединения



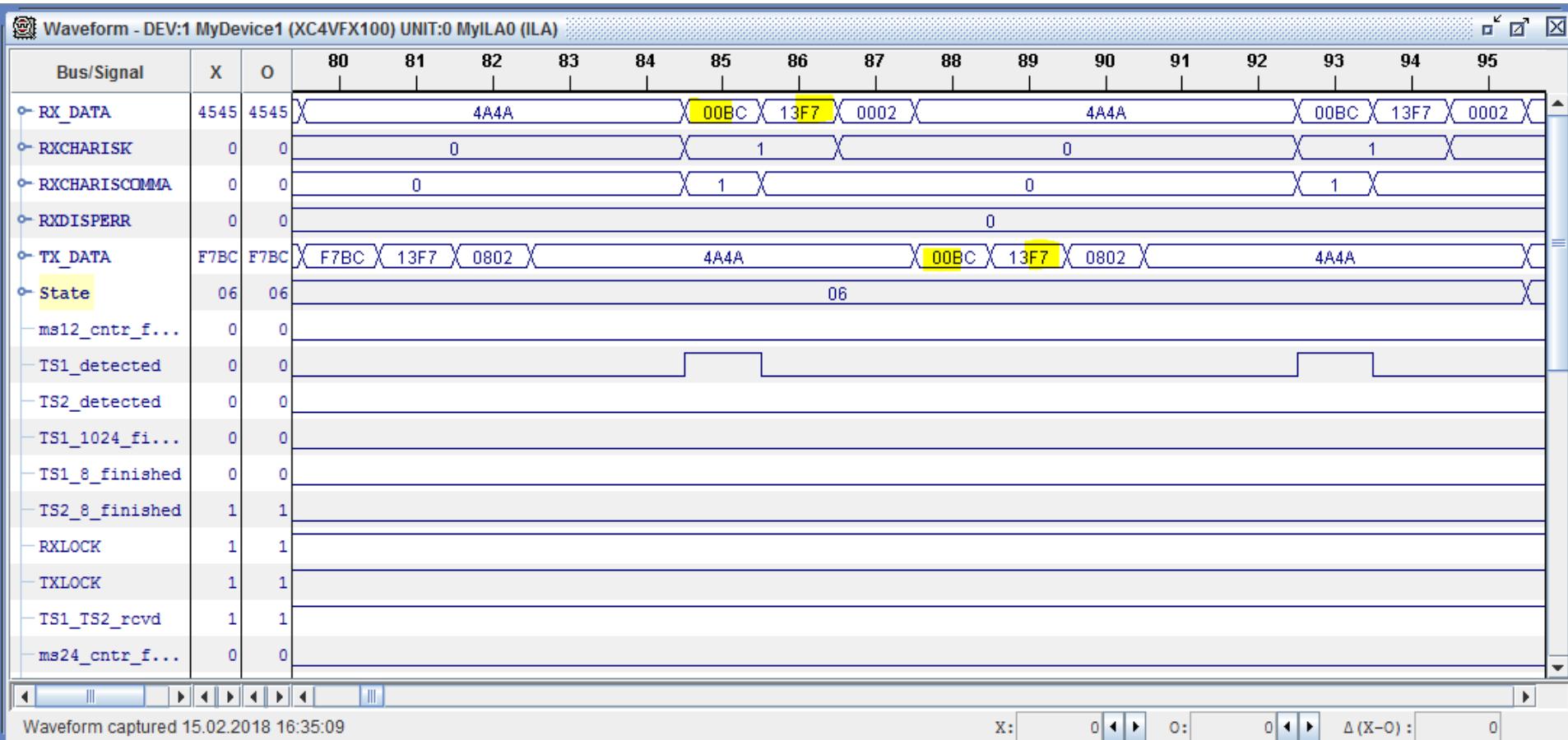
OM13800B

Figure 4-22: Main State Diagram for Link Training and Status State Machine

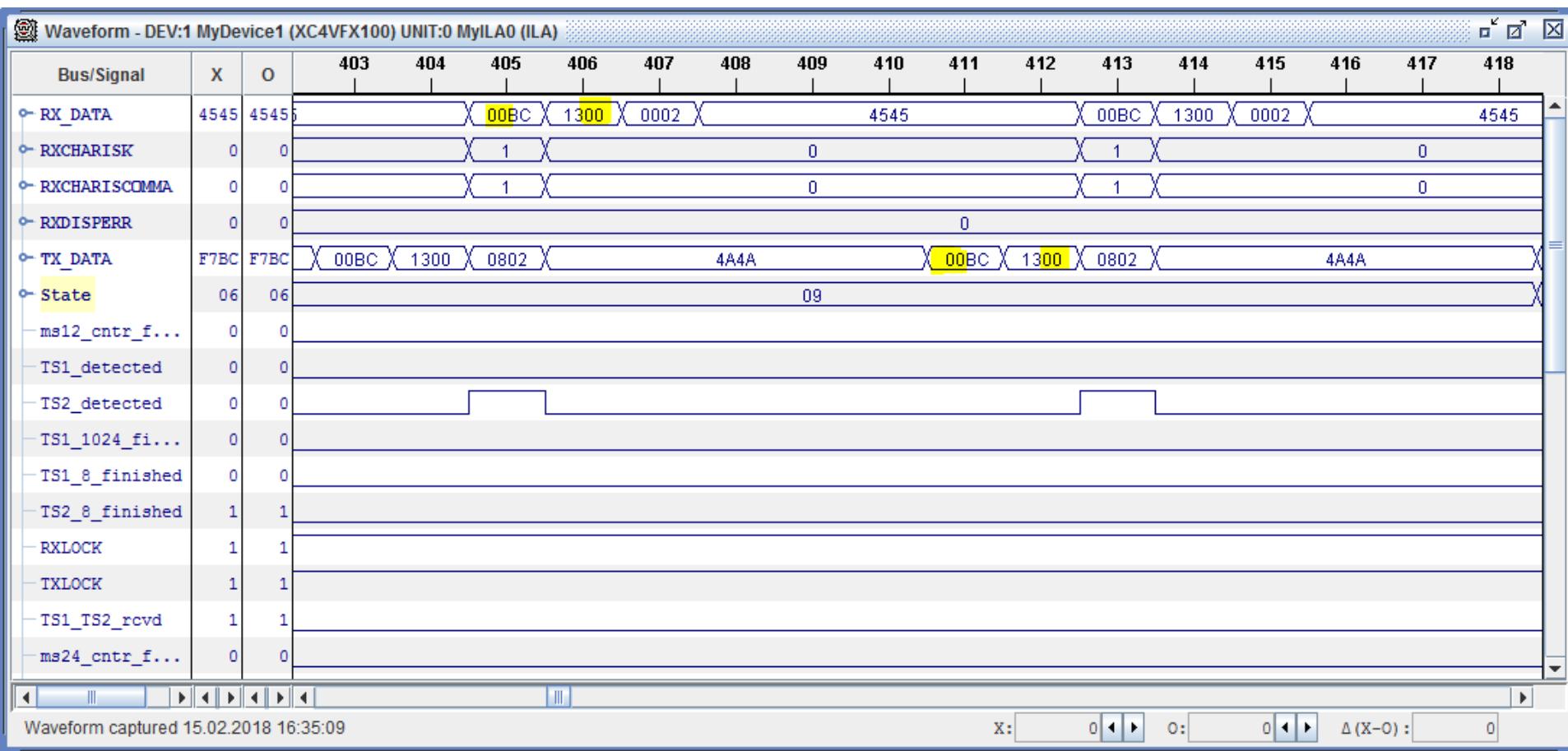
Опрос линий (POLLING_ACTIVE)



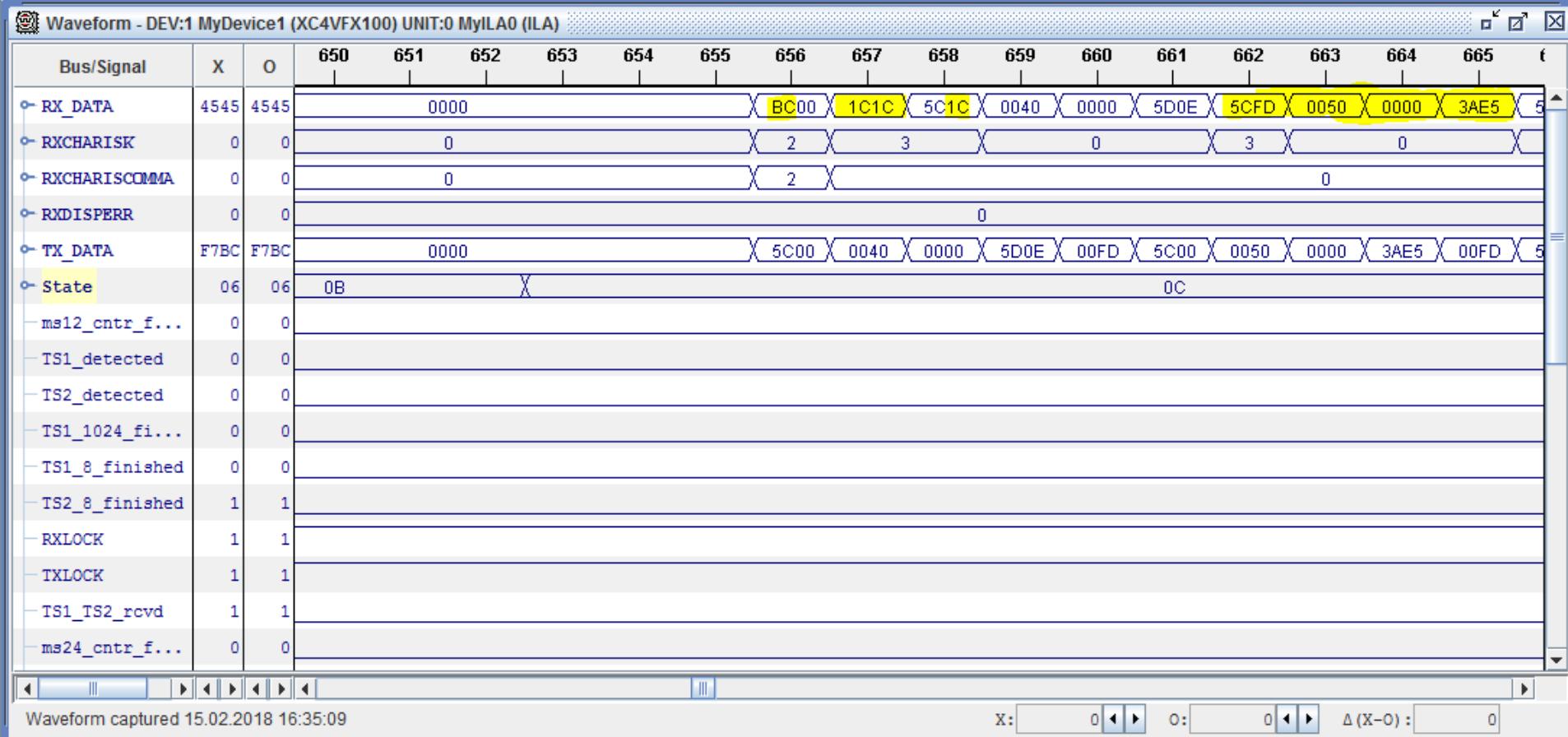
Определение разрядности линка (CONFIGURATION_LINKWIDTH_START)



Определение номеров линий (CONFIGURATION_LANENUM_ACCEP T)

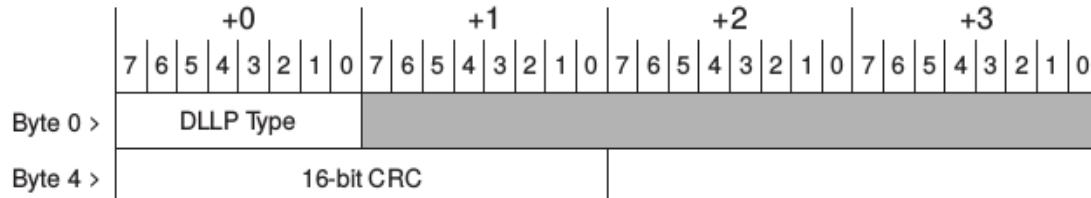


Готовность к приему и передаче (L0)

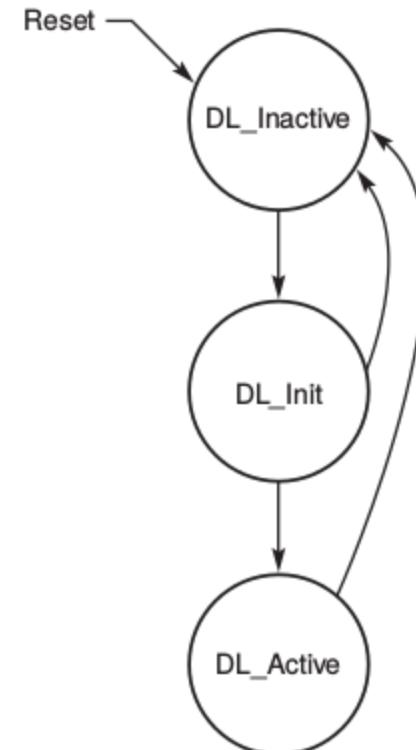


Data Link Layer Protocol

Формат Data Link Layer Packet



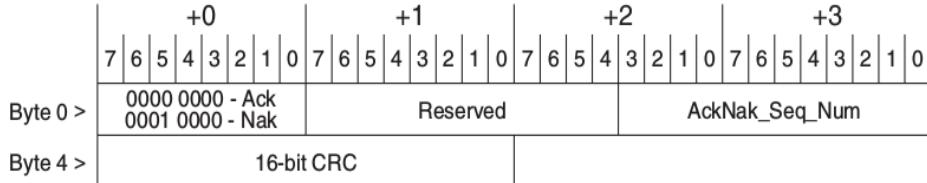
Состояния DLL



Типы пакетов DLLP

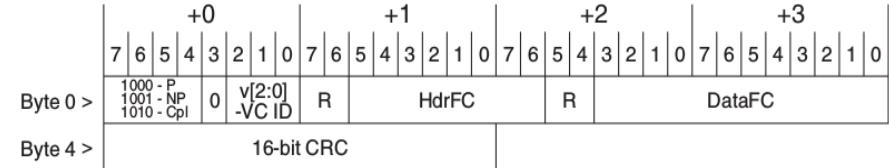
Encodings	DLLP Type
0000 0000	Ack
0001 0000	Nak
0010 0000	PM_Enter_L1
0010 0001	PM_Enter_L23
0010 0011	PM_Active_State_Request_L1
0010 0100	PM_Request_Ack
0011 0000	Vendor Specific – Not used in normal operation
0100 0v ₂ v ₁ v ₀	InitFC1-P (v[2:0] specifies Virtual Channel)
0101 0v ₂ v ₁ v ₀	InitFC1-NP
0110 0v ₂ v ₁ v ₀	InitFC1-Cpl
1100 0v ₂ v ₁ v ₀	InitFC2-P
1101 0v ₂ v ₁ v ₀	InitFC2-NP
1110 0v ₂ v ₁ v ₀	InitFC2-Cpl
1000 0v ₂ v ₁ v ₀	UpdateFC-P
1001 0v ₂ v ₁ v ₀	UpdateFC-NP
1010 0v ₂ v ₁ v ₀	UpdateFC-Cpl
All other encodings	Reserved

Форматы DLLP пакетов



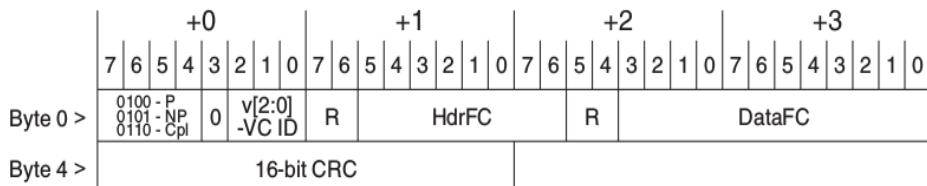
OM13781A

Figure 3-6: Data Link Layer Packet Format for Ack and Nak



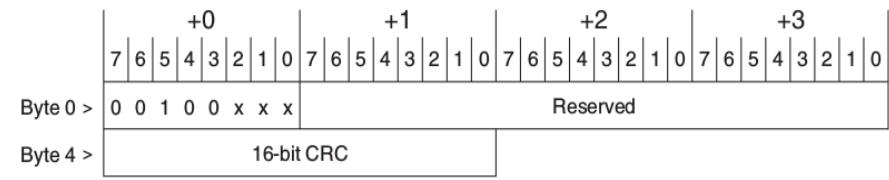
OM13784A

Figure 3-9: Data Link Layer Packet Format for UpdateFC



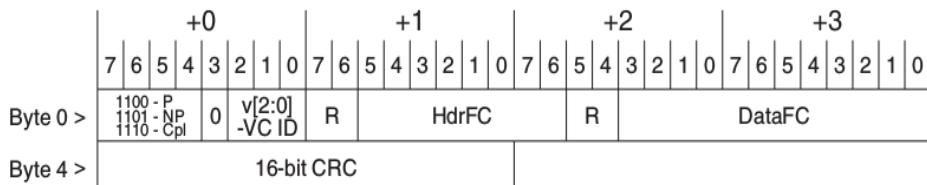
OM13782A

Figure 3-7: Data Link Layer Packet Format for InitFC1

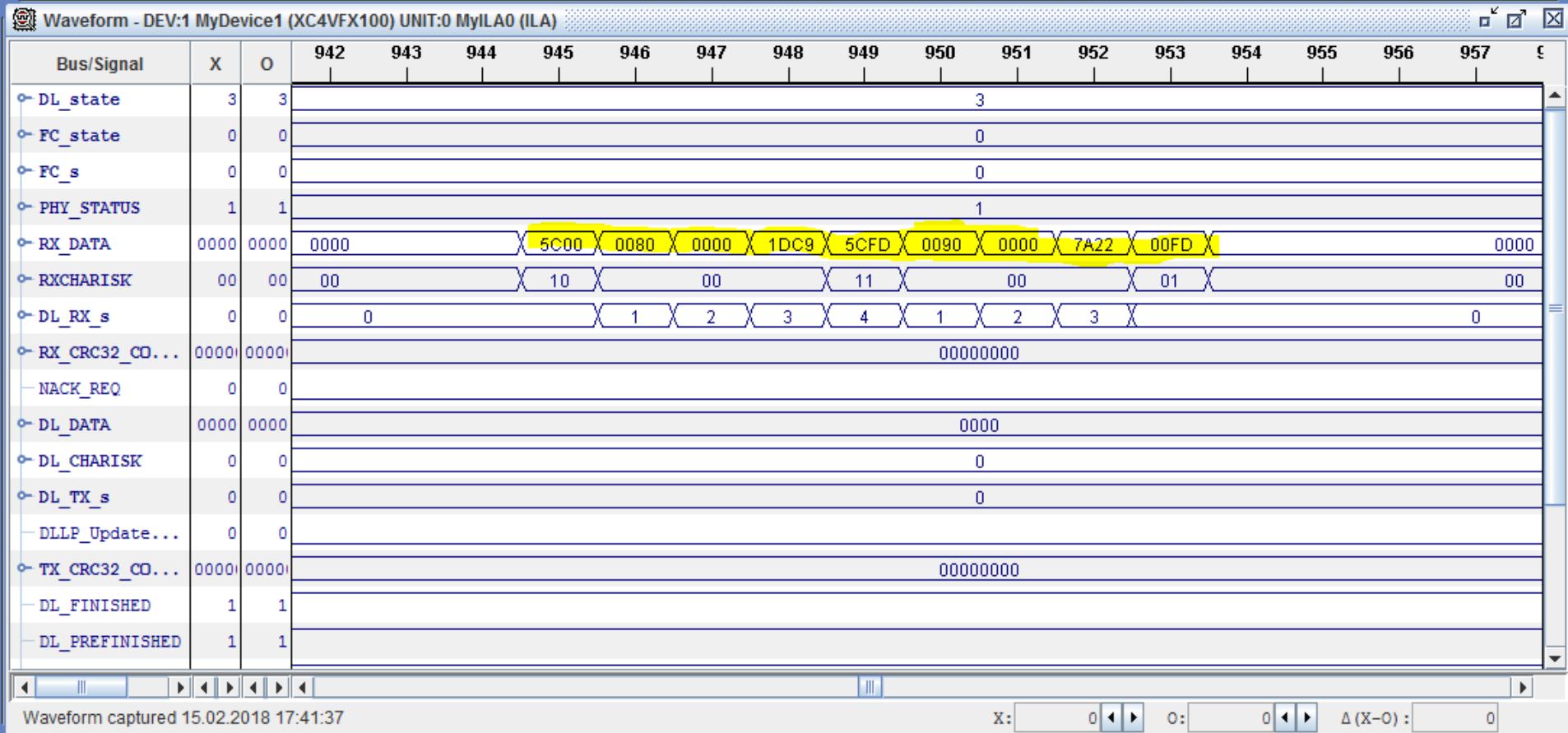


OM14304A

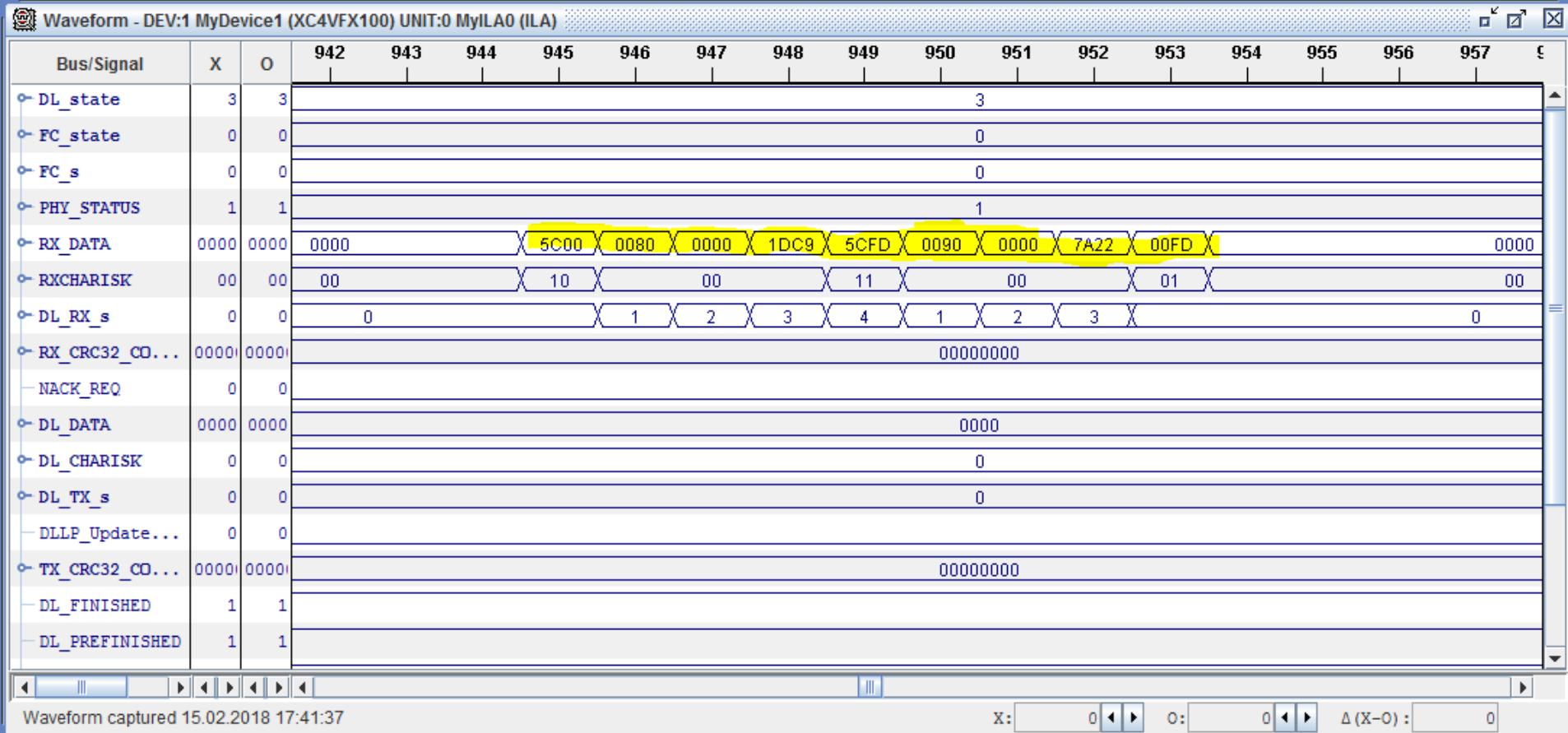
Figure 3-10: PM Data Link Layer Packet Format



Получение кредитов FC1, FC2, UpdateFC по DLLP



Передача кредитов FC1, FC2, UpdateFC по DLLP



Пример пакета TLP записи*

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
DW 0	R	Fmt		Type	R	TC		R	TDEP	Attr	R																					
	0	0x2		0x00	0	0		0	0	0	0																					
DW 1																																
DW 2																																
DW 3																																

Значение 0x12345678 читается по адресу 0xfdaff040 (2 LSB бита не передаются)

Формат идентификатора устройства

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0														
Requester/Completer ID																													
0x0100																													
Bus Number		Dev. Number		Function																									
0x01		0x00		0x0																									

Поля заголовка пакета

- Формат пакета
- Тип пакета
- Длина для всех связанных данных
- Описатель транзакций, включая:
 - идентификатор транзакции
 - Атрибуты
 - класс трафика
- Информация об адресе/маршрутизации
- Разрешение байт
- Кодировка сообщения
- Состояние завершения

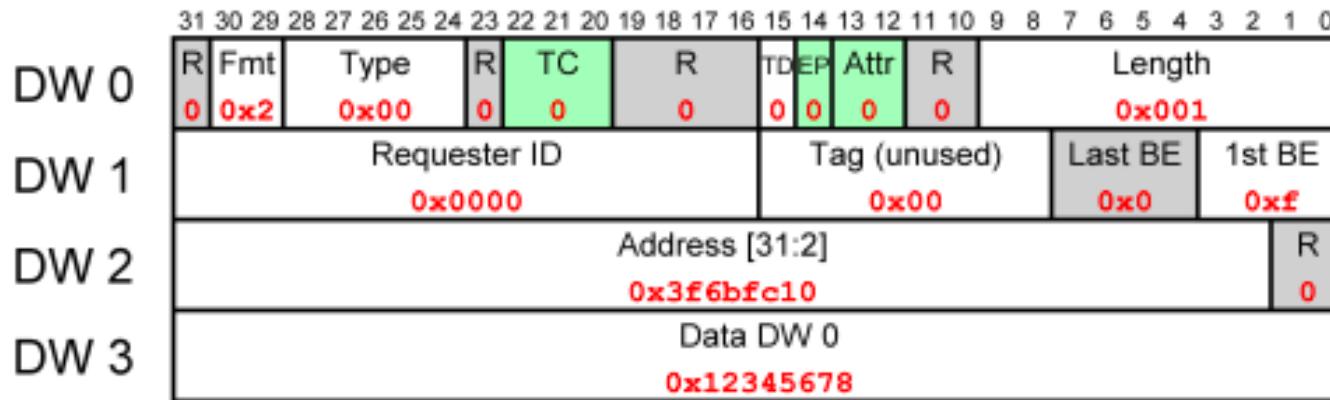
(*)

- Серые поля зарезервированы (игнорируются получателем).
- Зеленые поля могут иметь ненулевые значения, но используются редко.
- Значения определенного пакета помечаются красным цветом.

Типы транзакций

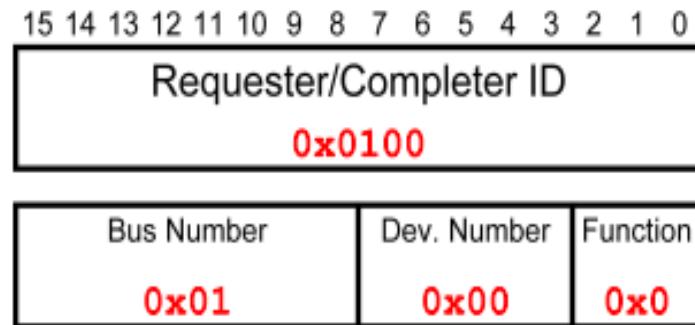
TLP Type	Fmt [2:0] ² (b)	Type [4:0] (b)	Description	TLP Type	Fmt [2:0] ² (b)	Type [4:0] (b)	Description
MRd	000 001	0 0000	Memory Read Request	FetchAdd	010 011	0 1100	Fetch and Add AtomicOp Request
MRdLk	000 001	0 0001	Memory Read Request-Locked	Swap	010 011	0 1101	Unconditional Swap AtomicOp Request
MWr	010 011	0 0000	Memory Write Request	CAS	010 011	0 1110	Compare and Swap AtomicOp Request
IOrd	000	0 0010	I/O Read Request	LPrfx	100	0L ₃ L ₂ L ₁ L ₀	Local TLP Prefix – The sub-field L[3:0] specifies the Local TLP Prefix type (see Table 2-30).
IOWr	010	0 0010	I/O Write Request	EPrfx	100	1E ₃ E ₂ E ₁ E ₀	End-End TLP Prefix – The sub-field E[3:0] specifies the End-End TLP Prefix type (see Table 2-31).
CfgRd0	000	0 0100	Configuration Read Type 0				All encodings not shown above are Reserved (see Section 2.3).
CfgWr0	010	0 0100	Configuration Write Type 0				
CfgRd1	000	0 0101	Configuration Read Type 1				
CfgWr1	010	0 0101	Configuration Write Type 1				
TCfgRd	000	1 1011	Deprecated TLP Type ³				
TCfgWr	010	1 1011	Deprecated TLP Type ³				
Msg	001	1 0r ₂ r ₁ r ₀	Message Request – The sub-field r[2:0] specifies the Message routing mechanism (see Table 2-18).				
MsgD	011	1 0r ₂ r ₁ r ₀	Message Request with data payload – The sub-field r[2:0] specifies the Message routing mechanism (see Table 2-18).				
Cpl	000	0 1010	Completion without Data – Used for I/O and Configuration Write Completions with any Completion Status. Also used for AtomicOp Completions and Read Completions (I/O, Configuration, or Memory) with Completion Status other than Successful Completion.				
CplD	010	0 1010	Completion with Data – Used for Memory, I/O, and Configuration Read Completions. Also used for AtomicOp Completions.				
CplLk	000	0 1011	Completion for Locked Memory Read without Data – Used only in error case.				
CplDLk	010	0 1011	Completion for Locked Memory Read – otherwise like CplD.				

Пример пакета TLP записи



Значение 0x12345678 читается по адресу 0xfdaff040 (2 LSB бита не передаются)

Формат идентификатора устройства



Пример пакета TLP чтения

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0						
DW 0	R	Fmt	Type	R	TC	R	TDEP	Attr	R																					Length	0x001							
DW 1																															Requester ID	0x0000	Tag	0x0c	Last BE	0x0	1st BE	0xf
DW 2																															Address [31:2]	0x3f6bfcc10	R	0				

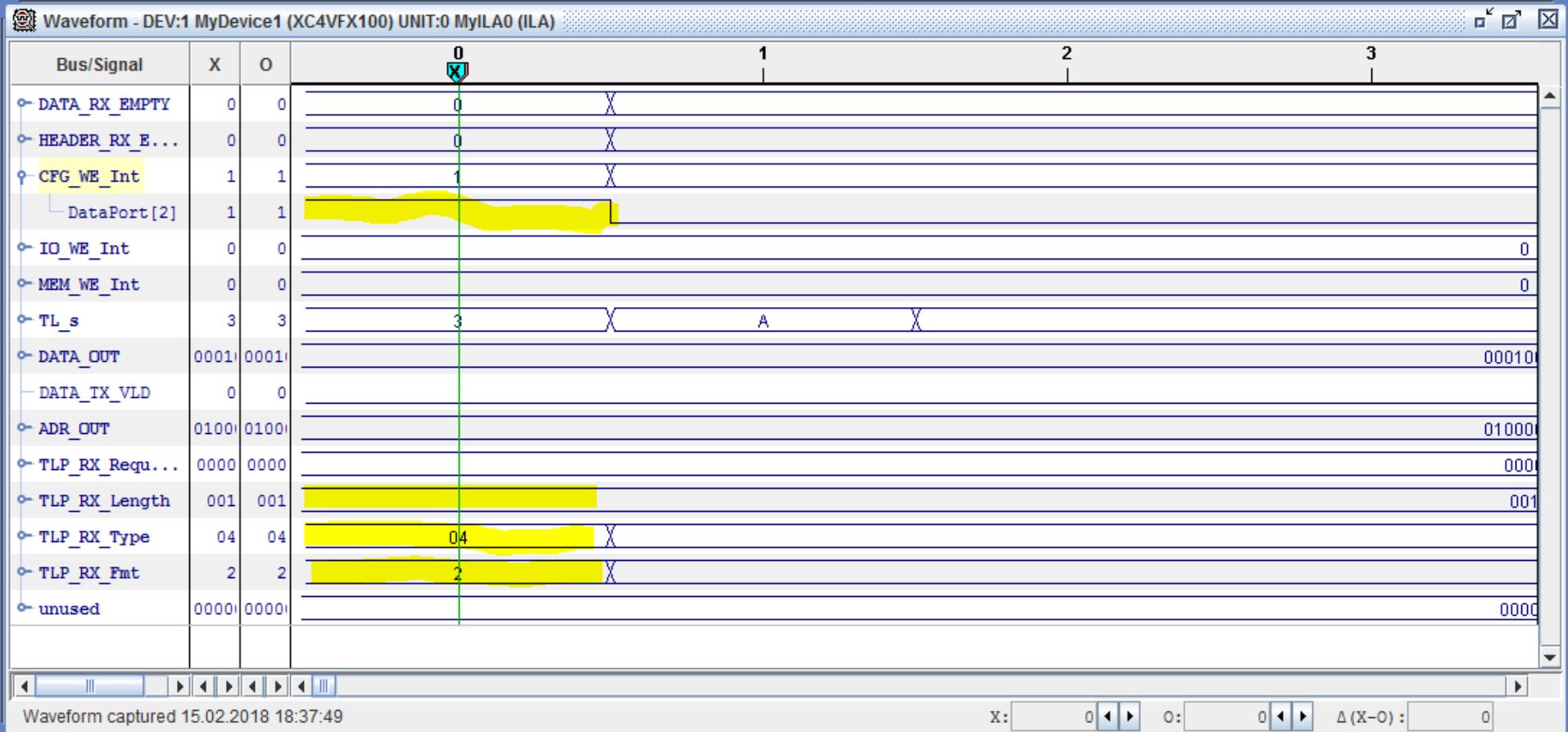
Значение читается по адресу 0xfdaff040 (2 LSB бита не передаются)

Пример пакета завершения (completion)

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0				
DW 0	R	Fmt	Type	R	TC	R	TDEP	Attr	R																			Length	0x001							
DW 1																													Completer ID	0x0100	Status	0x00	B/C/M	0	Byte Count	0x004
DW 2																													Requester ID	0x0000	Tag	0x0c	R	0	Lower Address	0x40
DW 3																													Data DW 0	0x12345678						

Значение 0x12345678 передаются инициатору

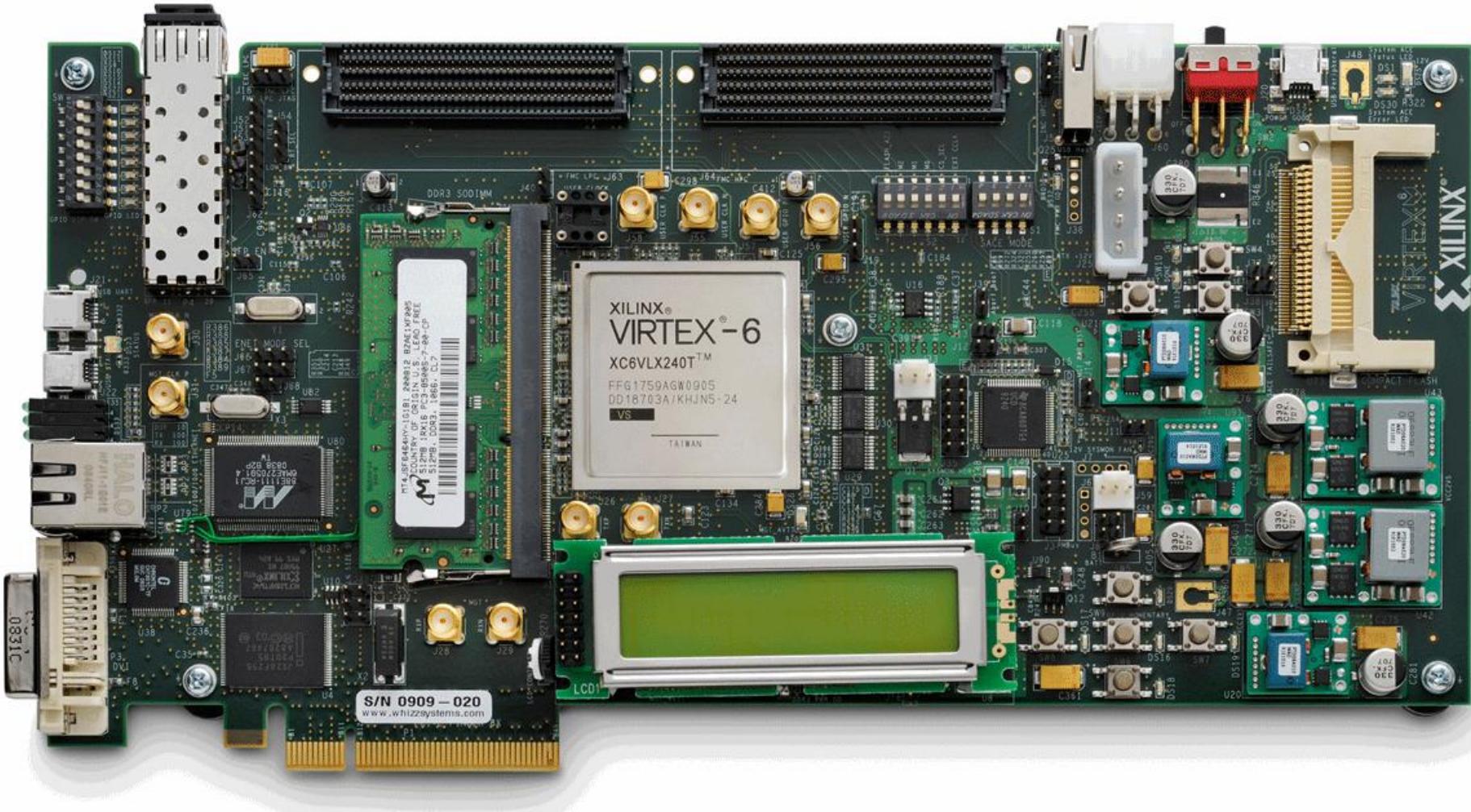
Прием и декодирование пакетов TLP



Сигналы разъема PCI Express

Pin	Side B	Side A	Description
1	+12 V	PRSNT1#	Must connect to farthest PRSNT2# pin
2	+12 V	+12 V	Main power pins
3	+12 V	+12 V	
4	Ground	Ground	
5	SMCLK	TCK	
6	SMDAT	TDI	
7	Ground	TDO	SMBus and JTAG port pins
8	+3.3 V	TMS	
9	TRST#	+3.3 V	
10	+3.3 V aux	+3.3 V	Standby power
11	WAKE#	PERST#	Link reactivation; fundamental reset
Key notch			
12	CLKREQ#	Ground	Request running clock
13	Ground	REFCLK+	Reference clock differential pair
14	HSoP(0)	REFCLK-	
15	HSoN(0)	Ground	Lane 0 transmit data, + and –
16	Ground	HSlP(0)	
17	PRSNT2#	HSlN(0)	Lane 0 receive data, + and –
18	Ground	Ground	
PCI Express ×1 cards end at pin 18			

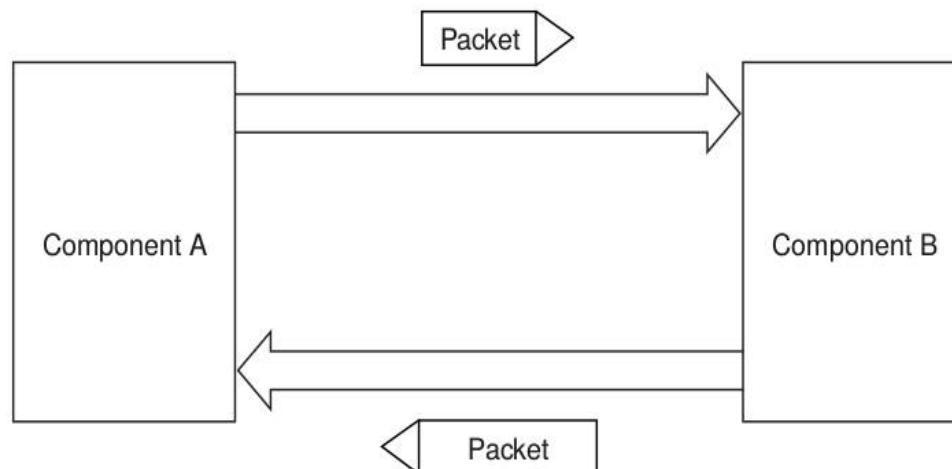
Плата ML605



Шина PCI express

Преимущества последовательных шин и интерфейсов:

- Большой акцент на сложную логику при простой топологии физического уровня;
- Перспектива перехода на оптический физический уровень;
- Экономия пространства печатных плат и снижение сложности монтажа;
- Простота реализации PnP и динамическую конфигурацию в любом смысле;
- Возможность выделять гарантированные и изохронные каналы;
- Переход от разделяемых шин с арбитражем к более предсказуемым соединениям точка-точка;
- Лучшая с точки зрения затрат и более гибкая с точки зрения топологии масштабируемость;



Сравнение пропускной способности шин семейства PCI

PCI и PCI-X

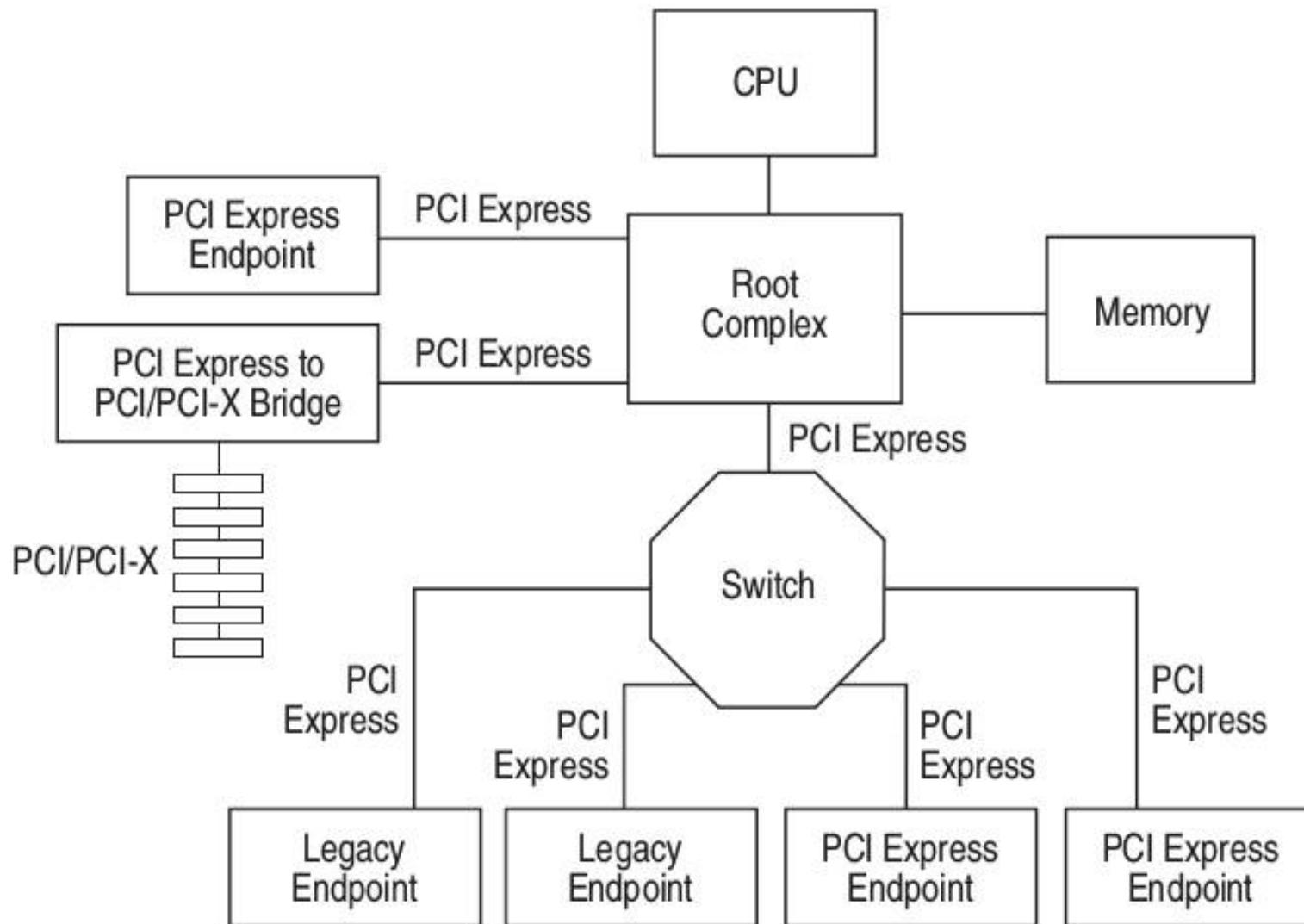
Bus Type	Clock Frequency	Peak Bandwidth *	Number of Card Slots per Bus
PCI 32-bit	33 MHz	133 MBytes/sec	4-5
PCI 32-bit	66 MHz	266 MBytes/sec	1-2
PCI-X 32-bit	66 MHz	266 MBytes/sec	4
PCI-X 32-bit	133 MHz	533 MBytes/sec	1-2
PCI-X 32-bit	266 MHz effective	1066 MBytes/sec	1
PCI-X 32-bit	533 MHz effective	2131 MByte/sec	1

* Double all these bandwidth numbers for 64-bit bus implementations

PCI Express

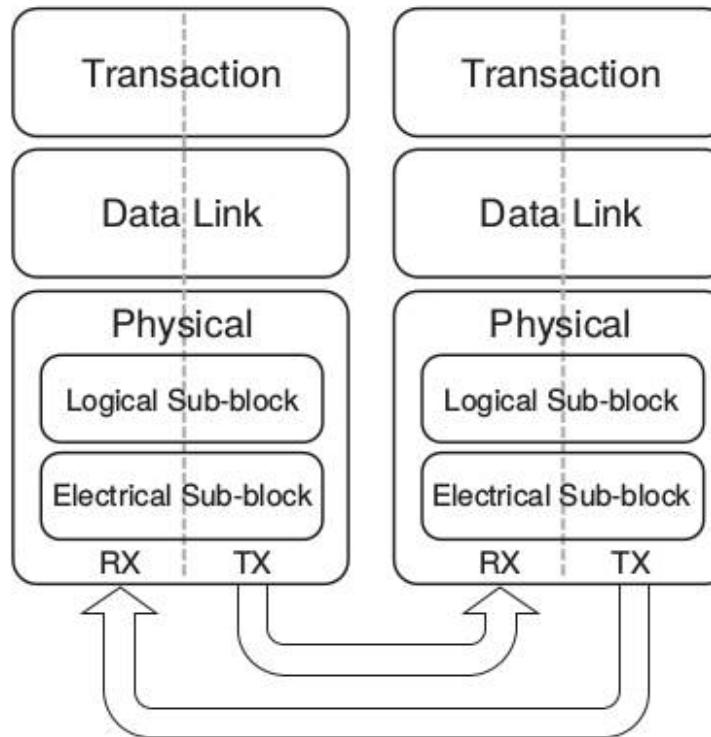
PCI Express version	Line code	Transfer rate ^[i]	Throughput ^[i]				
			×1	×2	×4	×8	×16
1.0	8b/10b	2.5 GT/s	250 MB/s	500 MB/s	1 GB/s	2 GB/s	4 GB/s
2.0	8b/10b	5.0 GT/s	500 MB/s	1 GB/s	2 GB/s	4 GB/s	8 GB/s
3.0	128b/130b	8.0 GT/s	984.6 MB/s	1.97 GB/s	3.94 GB/s	7.9 GB/s	15.8 GB/s
4.0	128b/130b	16.0 GT/s	1969 MB/s	3.94 GB/s	7.9 GB/s	15.8 GB/s	31.5 GB/s
5.0 ^{[30][31]} (expected in Q2 2019) ^[33]	128b/130b	32.0 GT/s ^[ii]	3938 MB/s	7.9 GB/s	15.8 GB/s	31.5 GB/s	63.0 GB/s

Принципы взаимодействия устройств по PCI Express



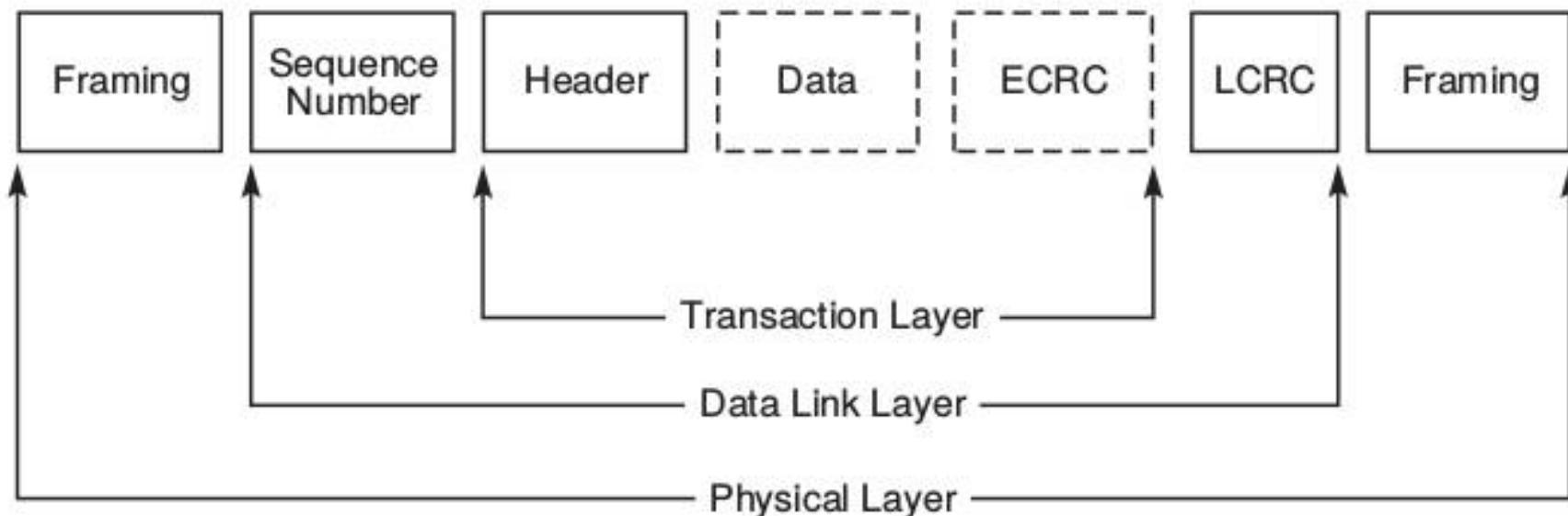
Соединение между двумя устройствами PCI Express называется link, и состоит из одного (называемого 1x) или нескольких (2x, 4x, 8x, 12x, 16x и 32x) двунаправленных последовательных соединений lane. Каждое устройство должно поддерживать соединение 1x.

Принципы взаимодействия устройств по PCI Express



- Уровень транзакций отвечает за взаимодействие с пригладным уровнем, сборку и разборку TLP пакетов (используются для передачи транзакций на нижние уровни, чтение и запись, передача событий). Уровень транзакций также отвечает за управление кредитами.
- Уровень линка данных принимает TLP пакеты от транзакционного уровня и осуществляет их передачу через физический уровень. Основные обязанности уровня линка данных включают управление линками и обеспечение целостности данных при передаче, включая обнаружение ошибок и исправление ошибок.
- Физический уровень включает в себя все необходимые схемы для работы интерфейса, параллельно-последовательные преобразователи, кодеры и декодеры 8/10, ФАПЧ(ы), схемы согласования сопротивлений.

Принципы взаимодействия устройств по PCI Express



- TLP, которые не проходят проверку целостности данных (LCRC и порядковый номер) или которые теряются при передаче из одного компонента в другой, повторно отправляются передатчиком.
- Передатчик хранит копию всех отправленных TLP, повторно отправляет эти копии, когда это необходимо, и очищает копии только тогда, когда он получает положительное подтверждение безошибочного получения от другого компонента.
- Если положительное подтверждение не было получено в течение указанного периода времени, Передатчик автоматически начнет повторную передачу.
- Приемник может запросить немедленную повторную передачу, используя отрицательное подтверждение NAK.

Принципы взаимодействия устройств по PCI Express

Transaction Type	Non-Posted or Posted
Memory Read	Non-Posted
Memory Write	Posted
Memory Read Lock	Non-Posted
IO Read	Non-Posted
IO Write	Non-Posted
Configuration Read (Type 0 and Type 1)	Non-Posted
Configuration Write (Type 0 and Type 1)	Non-Posted
Message	Posted

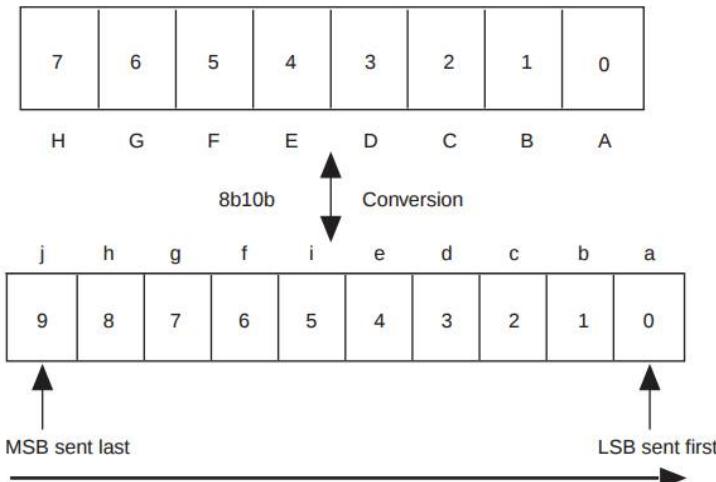
Участниками транзакции является Requester (инициатор) и Completer (исполнитель).

Транзакции Non-Posted предполагают получение ответа инициатором от исполнителя об успешном завершении.

Транзакция Posted не требует ответа от исполнителя.

Способы кодирования данных при приемопередаче

Кодирование 8/10 (PCIe v1.0, v2.0)



Всевозможных 10-битовых комбинаций больше, чем реально используется для представления 256 обычных символов. В наборе символов представлены числовые данные D и специальные символы K (control symbols).

Каждый символ имеет два образа: с положительным/отрицательным балансом нулевых и единичных бит (или с одинаковым количеством нулевых и единичных символов).

При получении 10-битовой последовательности битов, не соответствующей ни D-типу, ни K-типу, получатель сигнализирует об ошибке кодирования.

Encoding	Symbol	Name	Description
K28.5	COM	Comma	Used for Lane and Link initialization and management
K27.7	STP	Start TLP	Marks the start of a Transaction Layer Packet
K28.2	SDP	Start DLLP	Marks the start of a Data Link Layer Packet
K29.7	END	End	Marks the end of a Transaction Layer Packet or a Data Link Layer Packet
K30.7	EDB	EnD Bad	Marks the end of a nullified TLP
K23.7	PAD	Pad	Used in Framing and Link Width and Lane ordering negotiations
K28.0	SKP	Skip	Used for compensating for different bit rates for two communicating Ports
K28.1	FTS	Fast Training Sequence	Used within an Ordered Set to exit from L0s to L0
K28.3	IDL	Idle	Used in the Electrical Idle Ordered Set (EIOS)
K28.4			Reserved
K28.6			Reserved
K28.7	EIE	Electrical Idle Exit	Reserved in 2.5 GT/s Used in the Electrical Idle Exit Ordered Set (EIEOS) and sent prior to sending FTS at data rates other than 2.5 GT/s

Data Byte Name	Data Byte Value	Bits HGF EDCBA	Current RD - abcdei fghj	Current RD + abcdei fghj
K28.0	1C	000 11100	001111 0100	110000 1011
K28.1	3C	001 11100	001111 1001	110000 0110
K28.2	5C	010 11100	001111 0101	110000 1010
K28.3	7C	011 11100	001111 0011	110000 1100
K28.4	9C	100 11100	001111 0010	110000 1101
K28.5	BC	101 11100	001111 1010	110000 0101
K28.6	DC	110 11100	001111 0110	110000 1001
K28.7	FC	111 11100	001111 1000	110000 0111
K23.7	F7	111 10111	111010 1000	000101 0111
K27.7	FB	111 11011	110110 1000	001001 0111
K29.7	FD	111 11101	101110 1000	010001 0111
K30.7	FE	111 11110	011110 1000	100001 0111

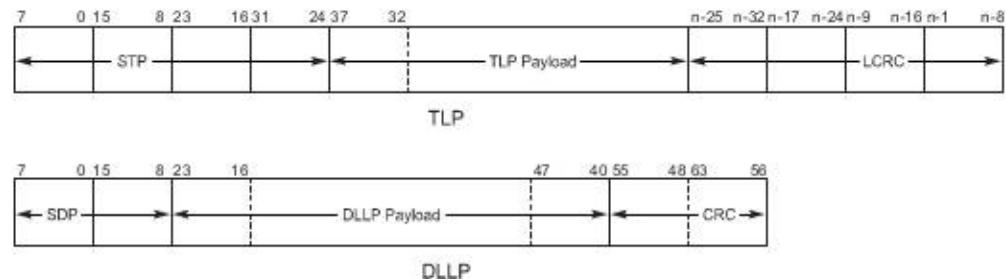
Physical Layer Protocol

Формат пакетов TS1 на Physical Layer

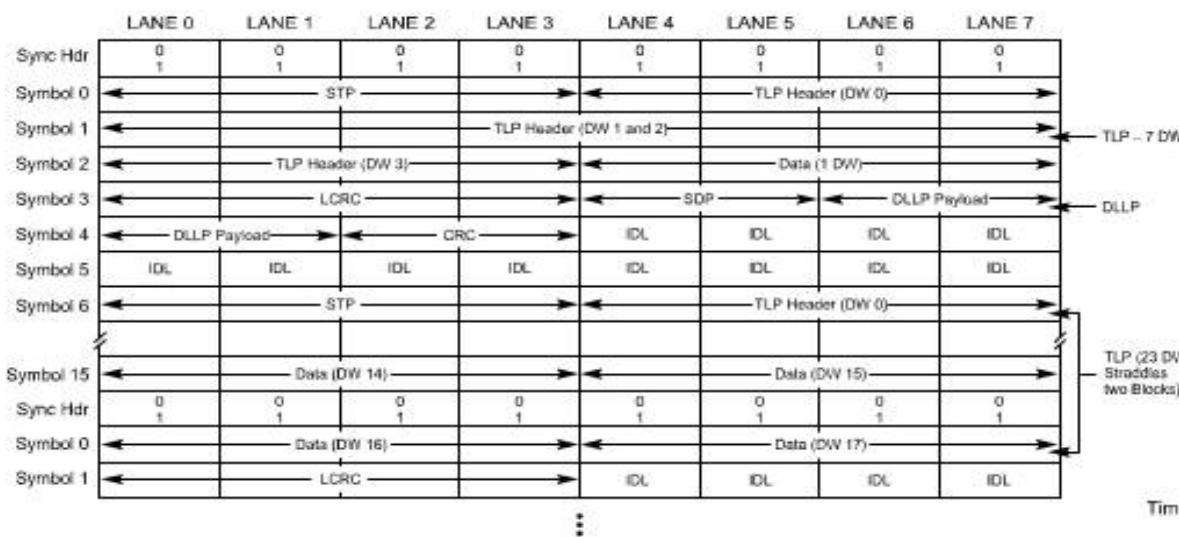
Table 4-5: TS1 Ordered Set

Symbol Number	Description
0	When operating at 2.5 or 5.0 GT/s: COM (K28.5) for Symbol alignment. When operating at 8.0 GT/s or above: Encoded as 1Eh (TS1 Ordered Set).
1	Link Number. Ports that do not support 8.0 GT/s or above: 0-255, PAD. Downstream Ports that support 8.0 GT/s or above: 0-31, PAD. Upstream Ports that support 8.0 GT/s or above: 0-255, PAD. When operating at 2.5 or 5.0 GT/s: PAD is encoded as K23.7. When operating at 8.0 GT/s or above: PAD is encoded as F7h.
2	Lane Number within Link. When operating at 2.5 or 5.0 GT/s: 0-31, PAD. PAD is encoded as K23.7. When operating at 8.0 GT/s or above: 0-31, PAD. PAD is encoded as F7h.
3	N_FTS. The number of Fast Training Sequences required by the Receiver: 0-255.
4	Data Rate Identifier Bit 0 – Reserved Bit 1 – 2.5 GT/s Data Rate Supported. Must be set to 1b. Bit 2 – 5.0 GT/s Data Rate Supported. Must be set to 1b if Bit 3 is 1b. Bit 3 – 8.0 GT/s Data Rate Supported. Bit 4:5 – Reserved. Bit 6 – Autonomous Change>Selectable De-emphasis. Downstream Ports: This bit is defined for use in the following LTSSM states: Polling.Active, Configuration.LinkWidth.Start, and Loopback.Entry. In all other LTSSM states, it is Reserved. Upstream Ports: This bit is defined for use in the following LTSSM states: Polling.Active, Configuration, Recovery, and Loopback.Entry. In all other LTSSM states, it is Reserved. Bit 7 – speed_change. This bit can be set to 1b only in the Recovery.RcvrLock LTSSM state. In all other LTSSM states, it is Reserved.
5	Training Control Bit 0 – Hot Reset Bit 0 = 0b, De-assert Bit 0 = 1b, Assert Bit 1 – Disable Link Bit 1 = 0b, De-assert Bit 1 = 1b, Assert Bit 2 – Loopback Bit 2 = 0b, De-assert Bit 2 = 1b, Assert Bit 3 – Disable Scrambling in 2.5 GT/s and 5.0 GT/s data rates: Reserved in other data rates Bit 3 = 0b, De-assert Bit 3 = 1b, Assert Bit 4 – Compliance Receive Bit 4 = 0b, De-assert Bit 4 = 1b, Assert Ports that support 5.0 GT/s and above data rate(s) must implement the Compliance Receive bit. Ports that support only 2.5 GT/s data rate may optionally implement the Compliance Receive bit. If not implemented, the bit is Reserved. Bit 5:7 – Reserved

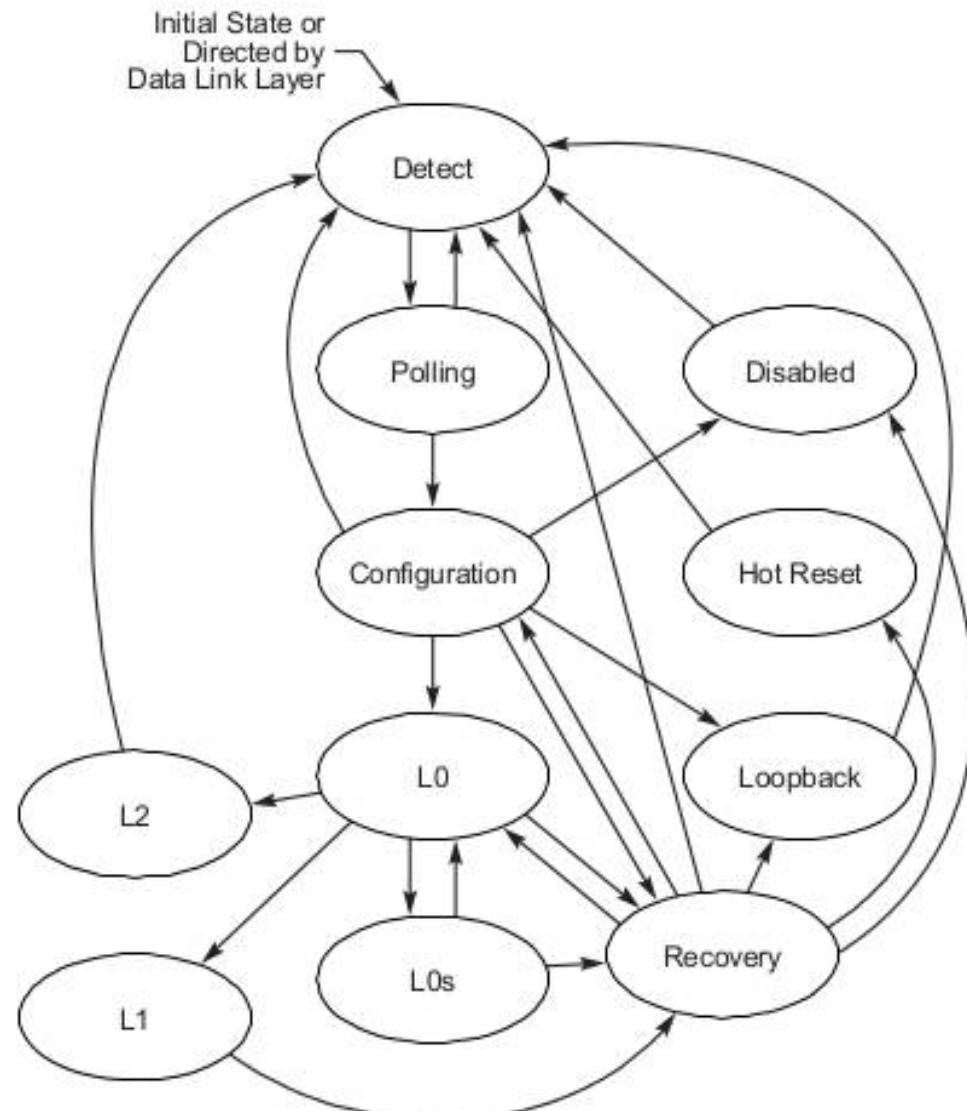
Формирование пакетов DLLP и TLP уровней



Пример передачи пакетов на PHY уровне



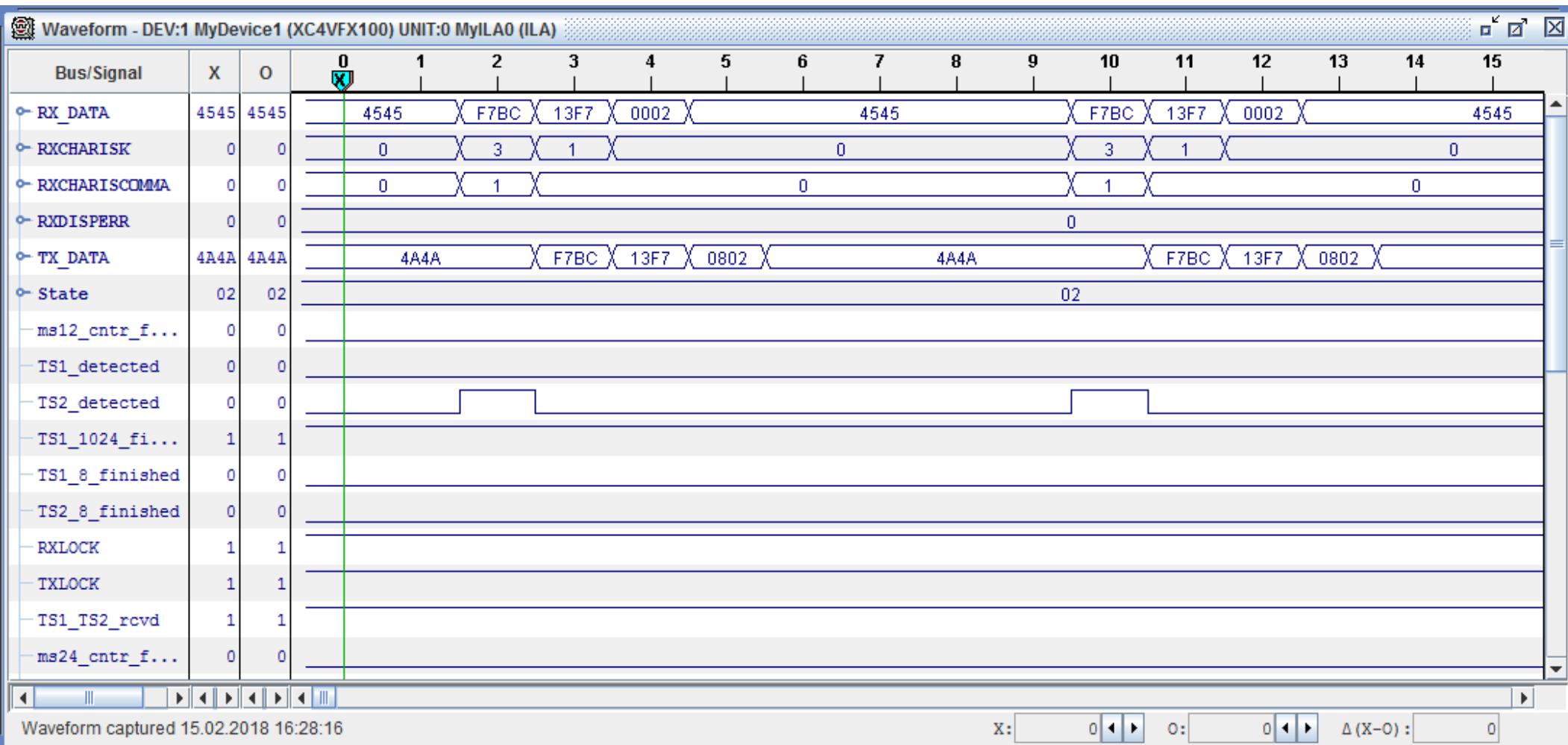
Инициализация физического соединения



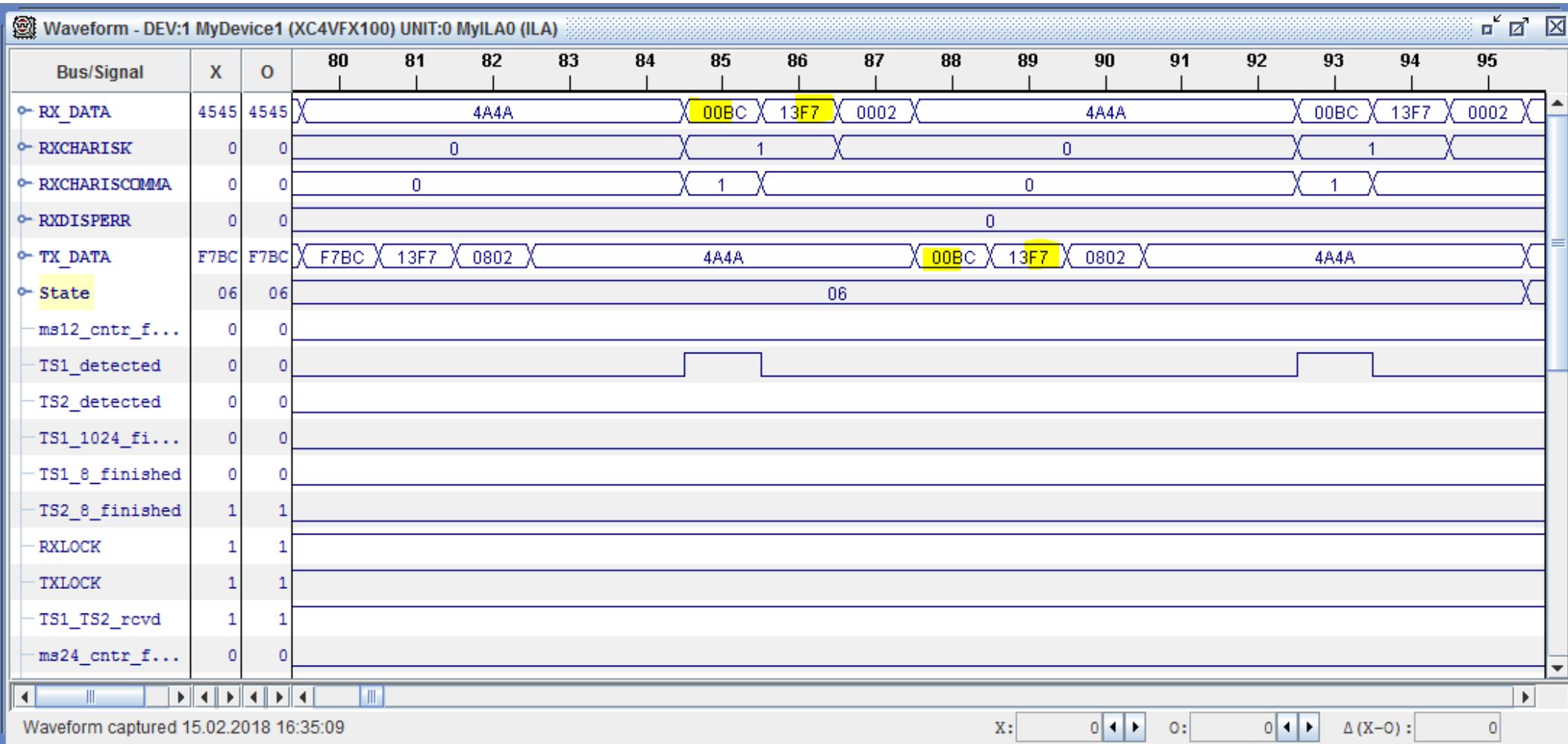
OM13800B

Figure 4-22: Main State Diagram for Link Training and Status State Machine

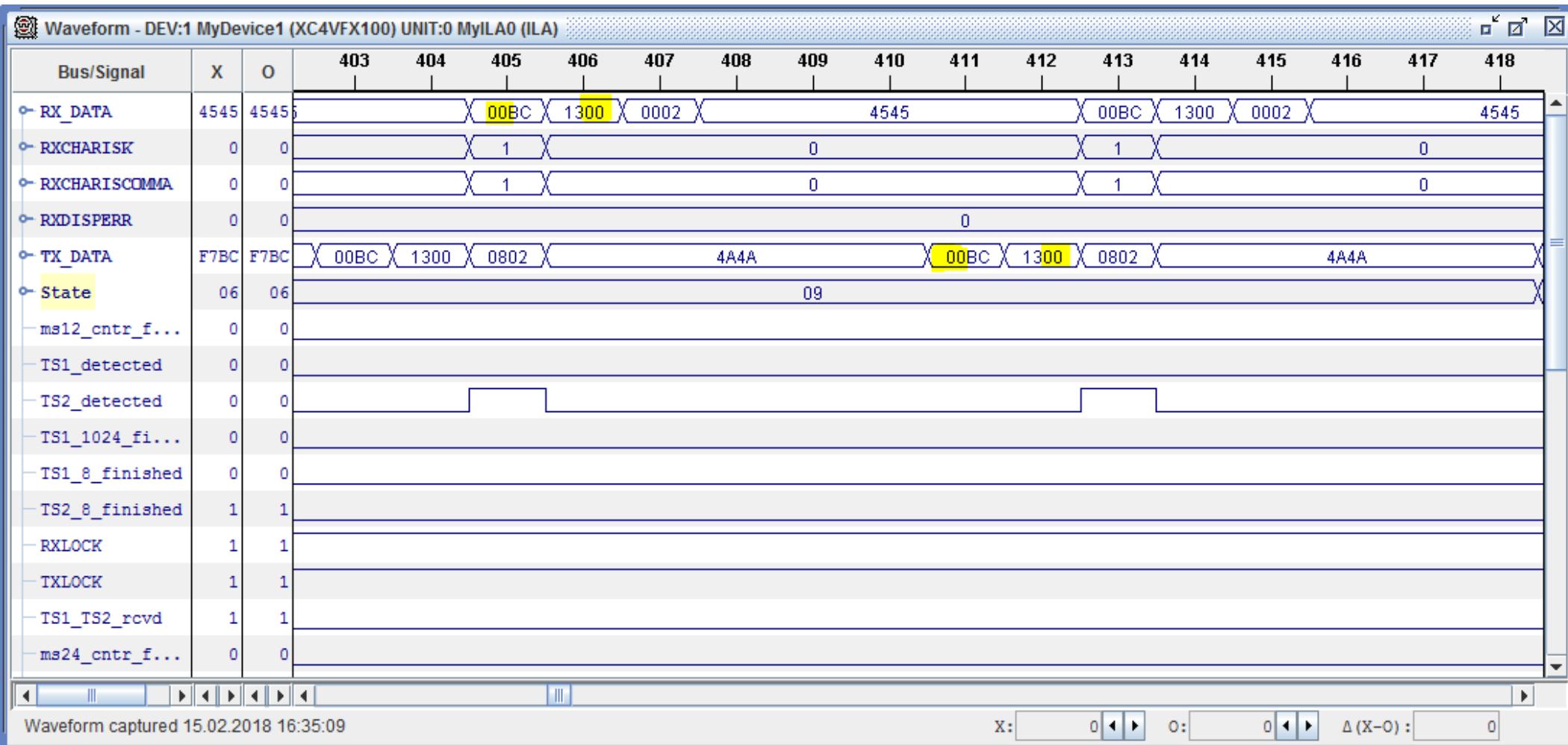
Опрос линий (POLLING_ACTIVE)



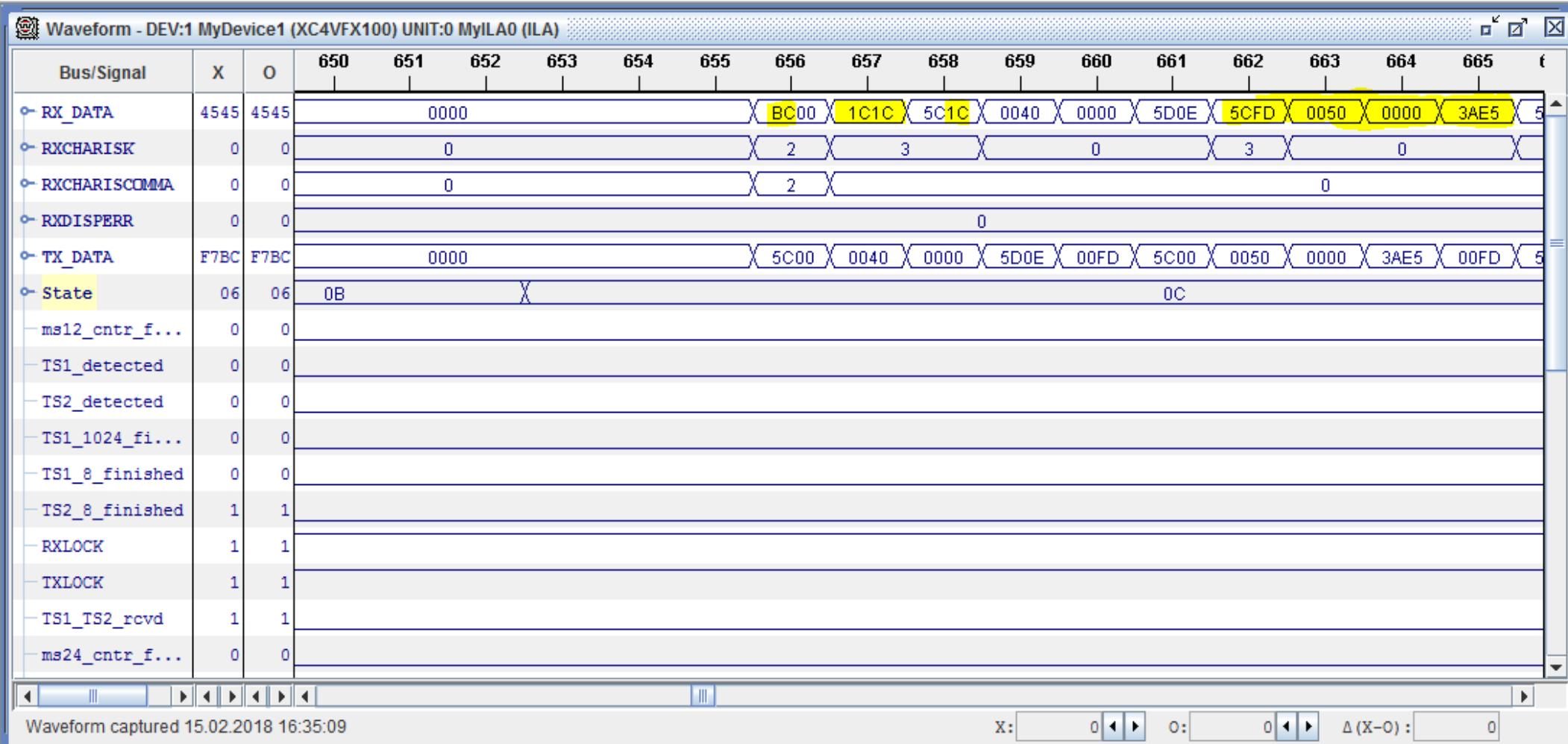
Определение разрядности линка (CONFIGURATION_LINKWIDTH_START)



Определение номеров линий (CONFIGURATION_LANENUM_ACCEPT)

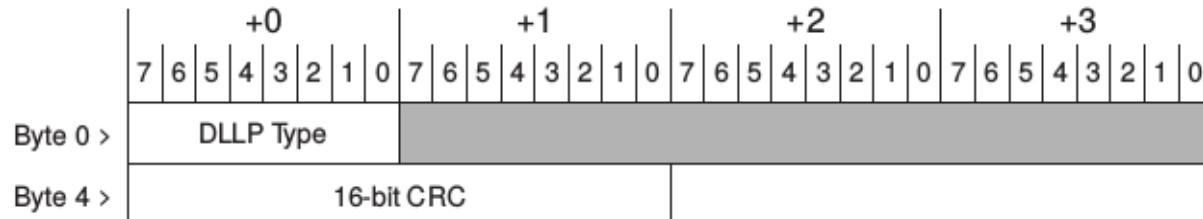


Готовность к приему и передаче (L0)

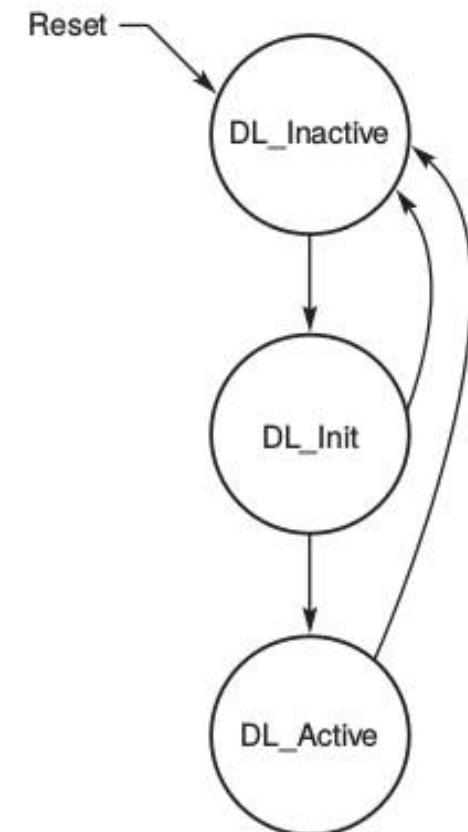


Data Link Layer Protocol

Формат Data Link Layer Packet



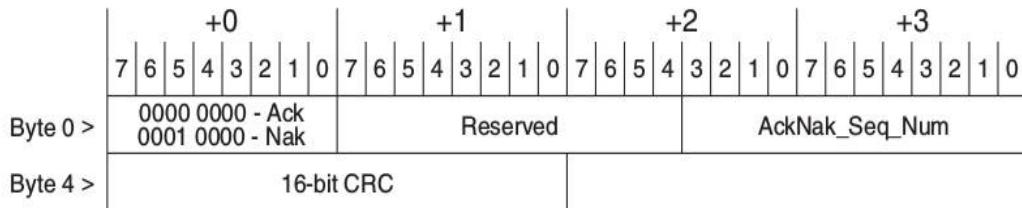
Состояния DLL



Типы пакетов DLLP

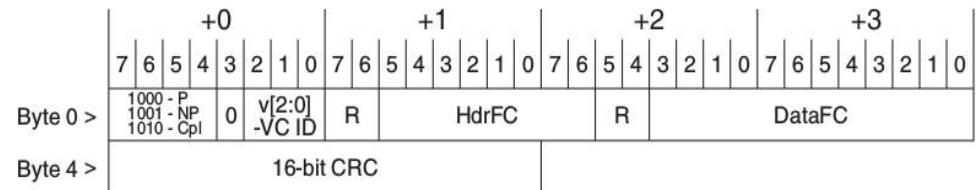
Encodings	DLLP Type
0000 0000	Ack
0001 0000	Nak
0010 0000	PM_Enter_L1
0010 0001	PM_Enter_L23
0010 0011	PM_Active_State_Request_L1
0010 0100	PM_Request_Ack
0011 0000	Vendor Specific – Not used in normal operation
0100 0v ₂ v ₁ v ₀	InitFC1-P (v[2:0] specifies Virtual Channel)
0101 0v ₂ v ₁ v ₀	InitFC1-NP
0110 0v ₂ v ₁ v ₀	InitFC1-Cpl
1100 0v ₂ v ₁ v ₀	InitFC2-P
1101 0v ₂ v ₁ v ₀	InitFC2-NP
1110 0v ₂ v ₁ v ₀	InitFC2-Cpl
1000 0v ₂ v ₁ v ₀	UpdateFC-P
1001 0v ₂ v ₁ v ₀	UpdateFC-NP
1010 0v ₂ v ₁ v ₀	UpdateFC-Cpl
All other encodings	Reserved

Форматы DLLP пакетов



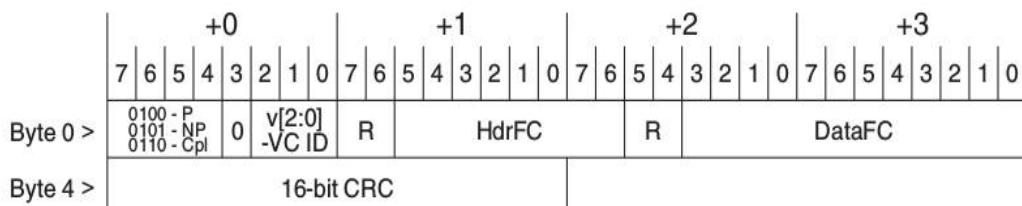
OM13781A

Figure 3-6: Data Link Layer Packet Format for Ack and Nak



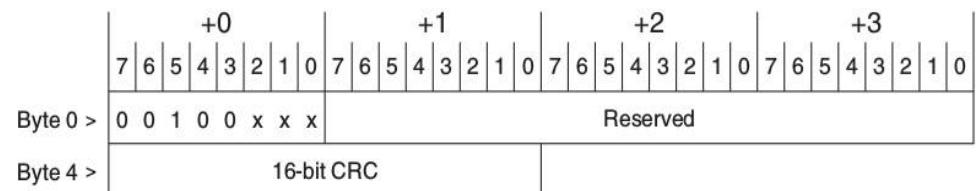
OM13784A

Figure 3-9: Data Link Layer Packet Format for UpdateFC



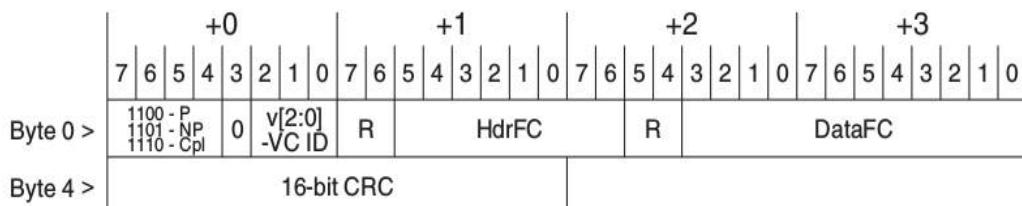
OM13782A

Figure 3-7: Data Link Layer Packet Format for InitFC1

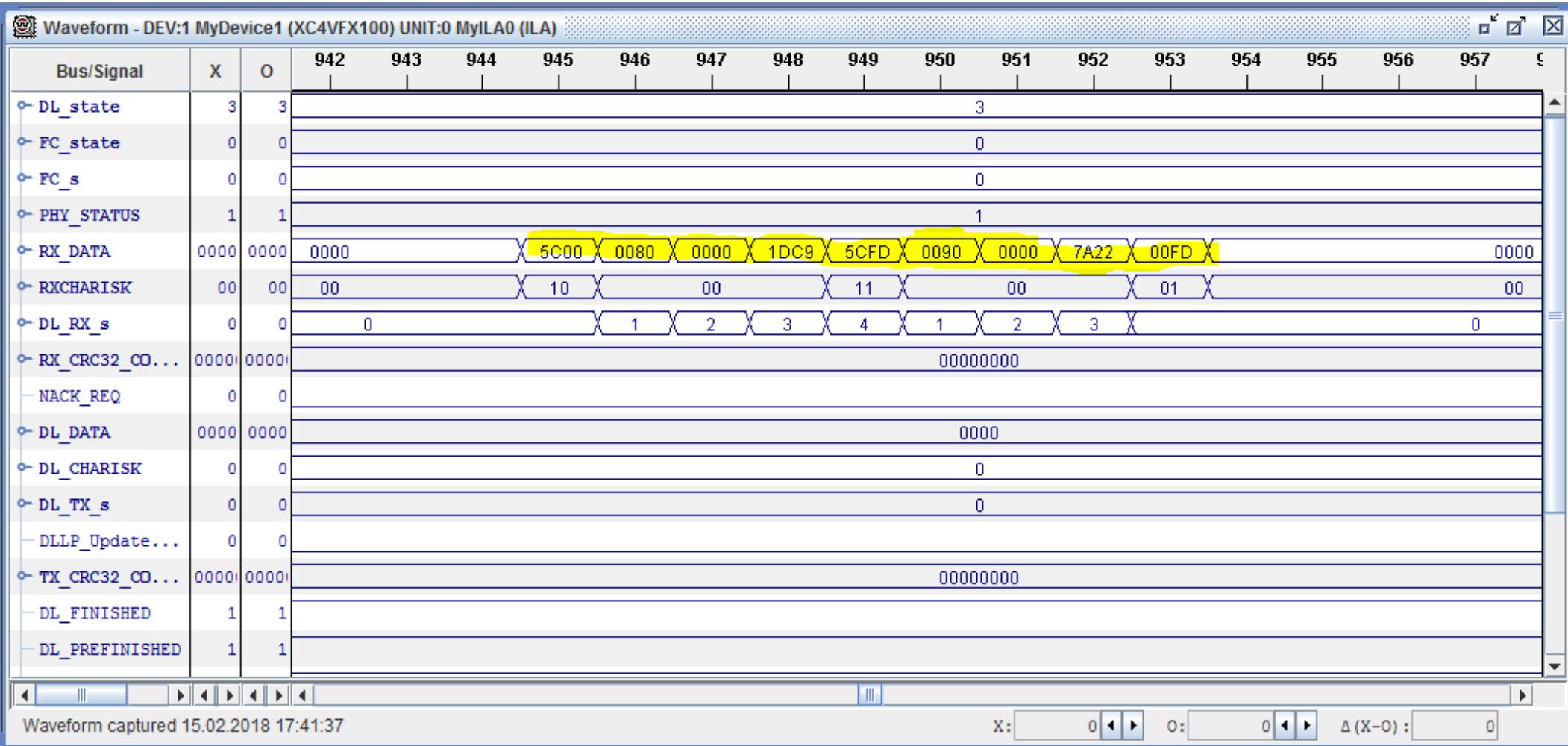


OM14304A

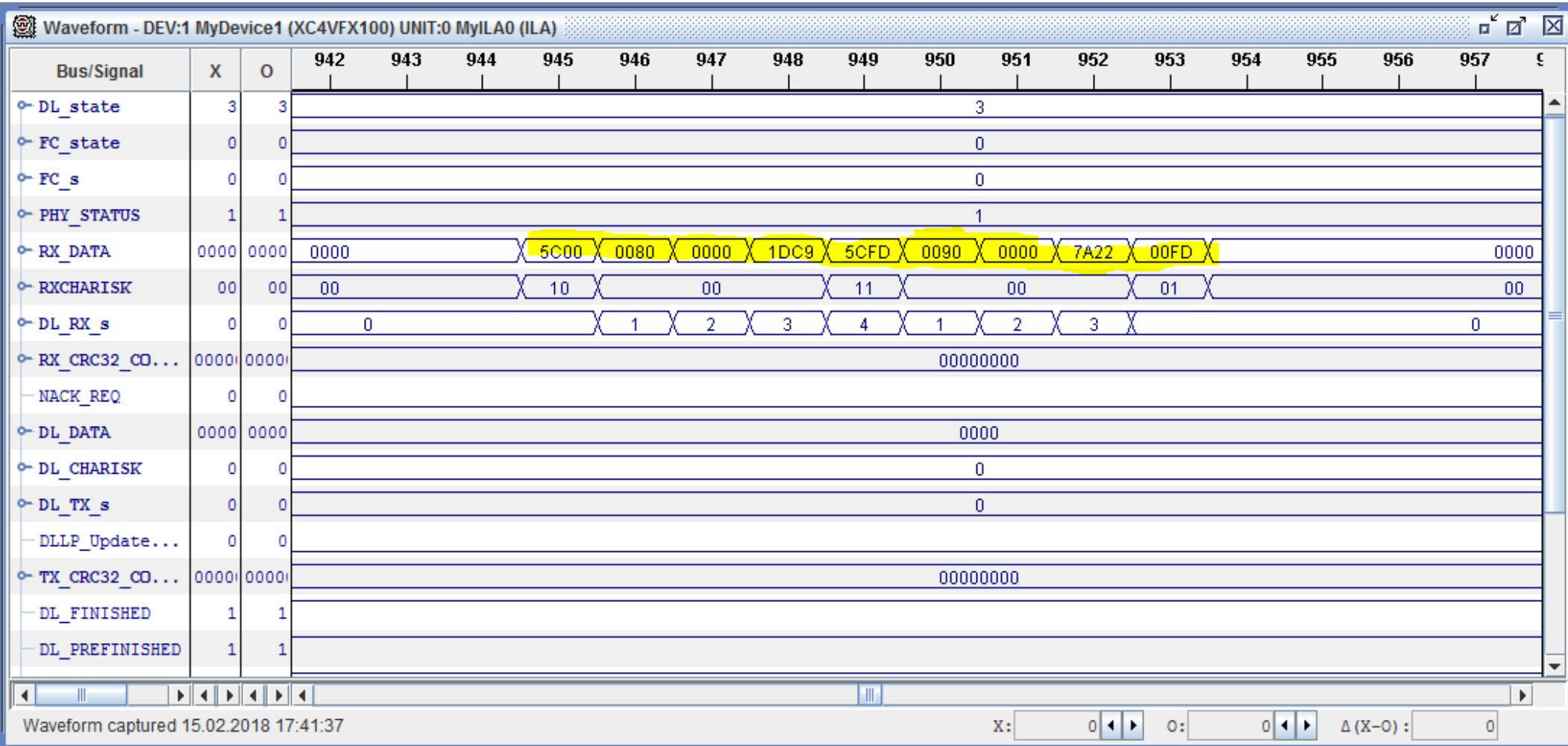
Figure 3-10: PM Data Link Layer Packet Format



Получение кредитов FC1, FC2, UpdateFC по DLLP



Передача кредитов FC1, FC2, UpdateFC по DLLP



Пример пакета TLP записи*

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
DW 0	R	Fmt	Type	R	TC	R	TDEP	Attr	R																					Length		
	0	0x2	0x00	0	0	0	0	0	0																				0x001			
DW 1																														Requester ID		
																														Tag (unused)		
																														Last BE		
																														1st BE		
DW 2																														Address [31:2]		
																														0x3f6bfcc10		
DW 3																														Data DW 0		
																														0x12345678		

Значение 0x12345678 читается по адресу 0xfdaff040 (2 LSB бита не передаются)

Формат идентификатора устройства

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Requester/Completer ID															
0x0100															
Bus Number	Dev. Number	Function	0x01	0x00	0x0										

Поля заголовка пакета

- › Формат пакета
- › Тип пакета
- › Длина для всех связанных данных
- › Описатель транзакций, включая:
 - идентификатор транзакции
 - Атрибуты
 - класс трафика
- › Информация об адресе/маршрутизации
- › Разрешение байт
- › Кодировка сообщения
- › Состояние завершения

(*)

- Серые поля зарезервированы (игнорируются получателем).
- Зеленые поля могут иметь ненулевые значения, но используются редко.
- Значения определенного пакета помечаются красным цветом.

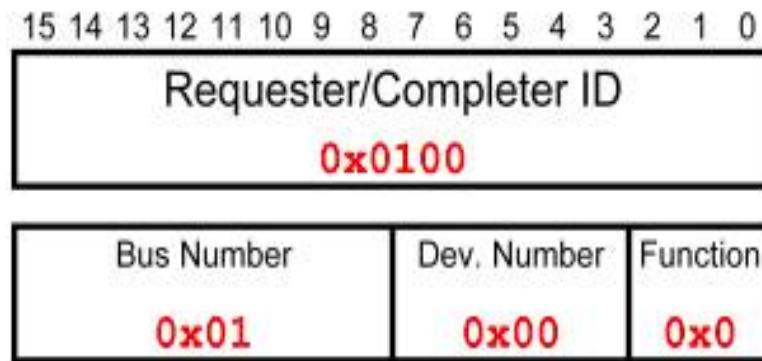
Типы транзакций

TLP Type	Fmt [2:0] ² (b)	Type [4:0] (b)	Description	TLP Type	Fmt [2:0] ² (b)	Type [4:0] (b)	Description
MRd	000 001	0 0000	Memory Read Request	FetchAdd	010 011	0 1100	Fetch and Add AtomicOp Request
MRdLk	000 001	0 0001	Memory Read Request-Locked	Swap	010 011	0 1101	Unconditional Swap AtomicOp Request
MWr	010 011	0 0000	Memory Write Request	CAS	010 011	0 1110	Compare and Swap AtomicOp Request
IORd	000	0 0010	I/O Read Request	LPrfx	100	0L ₃ L ₂ L ₁ L ₀	Local TLP Prefix – The sub-field L[3:0] specifies the Local TLP Prefix type (see Table 2-30).
IOWr	010	0 0010	I/O Write Request	EPrfx	100	1E ₃ E ₂ E ₁ E ₀	End-End TLP Prefix – The sub-field E[3:0] specifies the End-End TLP Prefix type (see Table 2-31).
CfgRd0	000	0 0100	Configuration Read Type 0				All encodings not shown above are Reserved (see Section 2.3).
CfgWr0	010	0 0100	Configuration Write Type 0				
CfgRd1	000	0 0101	Configuration Read Type 1				
CfgWr1	010	0 0101	Configuration Write Type 1				
TCfgRd	000	1 1011	Deprecated TLP Type ³				
TCfgWr	010	1 1011	Deprecated TLP Type ³				
Msg	001	1 0r ₂ r ₁ r ₀	Message Request – The sub-field r[2:0] specifies the Message routing mechanism (see Table 2-18).				
MsgD	011	1 0r ₂ r ₁ r ₀	Message Request with data payload – The sub-field r[2:0] specifies the Message routing mechanism (see Table 2-18).				
Cpl	000	0 1010	Completion without Data – Used for I/O and Configuration Write Completions with any Completion Status. Also used for AtomicOp Completions and Read Completions (I/O, Configuration, or Memory) with Completion Status other than Successful Completion.				
CpID	010	0 1010	Completion with Data – Used for Memory, I/O, and Configuration Read Completions. Also used for AtomicOp Completions.				
CpILk	000	0 1011	Completion for Locked Memory Read without Data – Used only in error case.				
CpIDLk	010	0 1011	Completion for Locked Memory Read – otherwise like CpID.				

Пример пакета TLP записи

Значение 0x12345678 читается по адресу 0xfdaff040 (2 LSB бита не передаются)

Формат идентификатора устройства



Пример пакета TLP чтения

	31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0											
DW 0	R	Fmt	Type	R	TC	R	TDEP	Attr	R	Length		
	0	0x0	0x00	0	0	0	0	0	0	0x001		
DW 1	Requester ID 0x0000					Tag 0x0c			Last BE	1st BE 0x0		
DW 2	Address [31:2] 0x3f6bfcc10										R	0

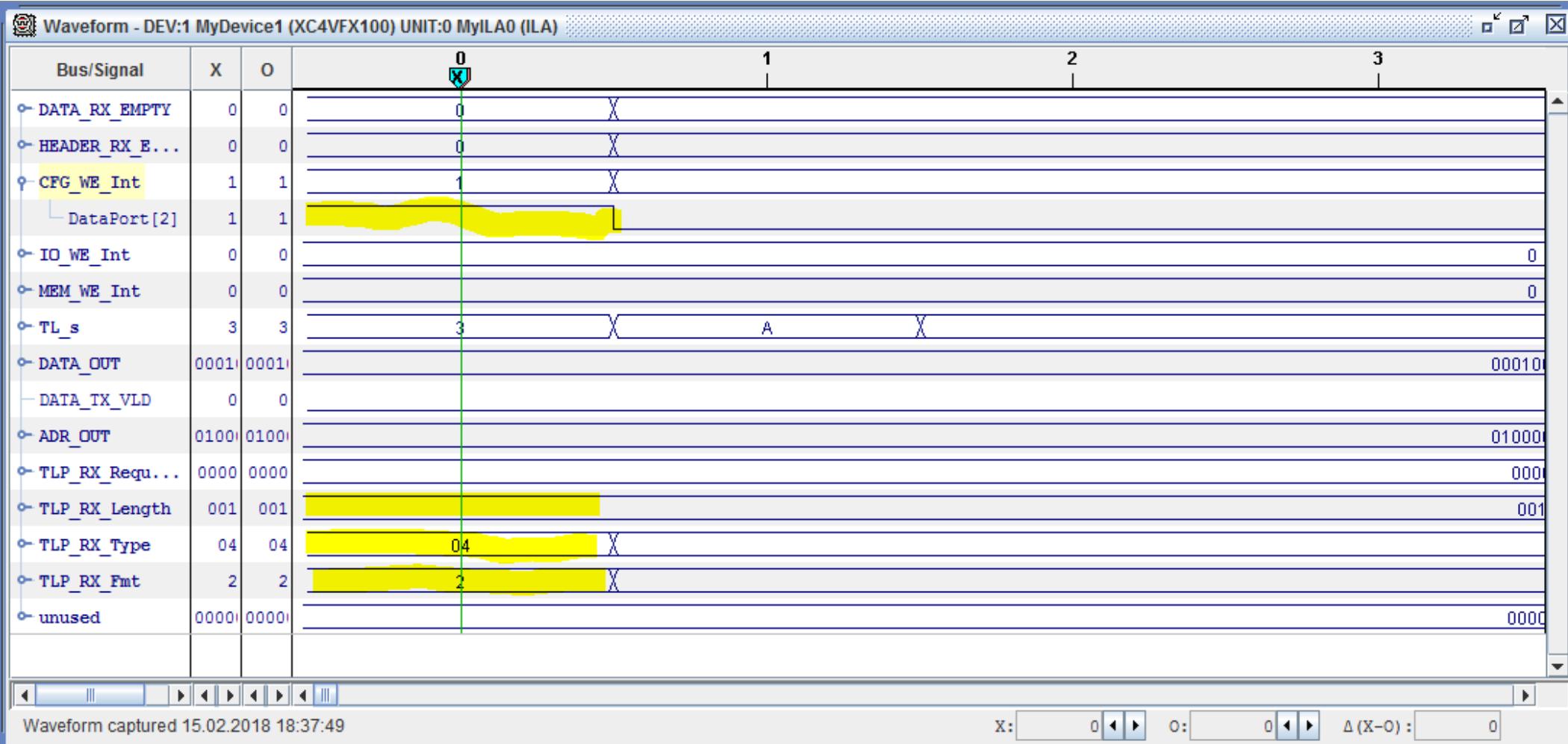
Значение читается по адресу 0xfdaff040 (2 LSB бита не передаются)

Пример пакета завершения (completion)

	31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0											
DW 0	R	Fmt	Type	R	TC	R	TDEP	Attr	R	Length		
	0	0x2	0x0a	0	0	0	0	0	0	0x001		
DW 1	Completer ID 0x0100					Status	B C M	Byte Count 0x00				
DW 2	Requester ID 0x0000					Tag 0x0c			R	Lower Address 0		
DW 3	Data DW 0 0x12345678											

Значение 0x12345678 передаются инициатору

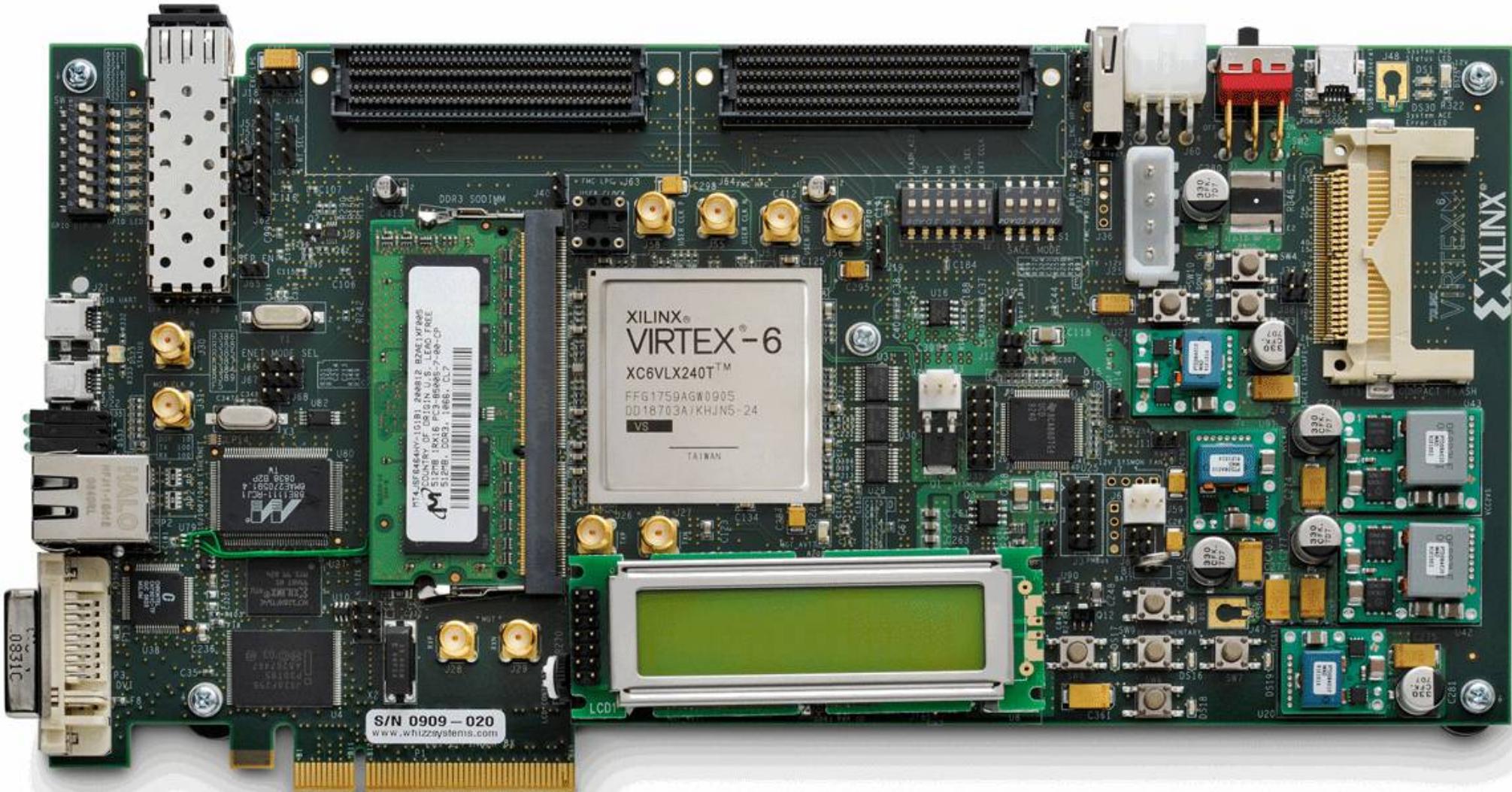
Прием и декодирование пакетов TLP



Сигналы разъема PCI Express

Pin	Side B	Side A	Description
1	+12 V	PRSNT1#	Must connect to farthest PRSNT2# pin
2	+12 V	+12 V	Main power pins
3	+12 V	+12 V	
4	Ground	Ground	
5	SMCLK	TCK	
6	SMDAT	TDI	
7	Ground	TDO	SMBus and JTAG port pins
8	+3.3 V	TMS	
9	TRST#	+3.3 V	
10	+3.3 V aux	+3.3 V	Standby power
11	WAKE#	PERST#	Link reactivation; fundamental reset
Key notch			
12	CLKREQ#	Ground	Request running clock
13	Ground	REFCLK+	Reference clock differential pair
14	HOp(0)	REFCLK-	
15	HOn(0)	Ground	Lane 0 transmit data, + and -
16	Ground	HSIp(0)	
17	PRSNT2#	HSIn(0)	Lane 0 receive data, + and -
18	Ground	Ground	
PCI Express ×1 cards end at pin 18			

Плата ML605



Архитектура графических процессоров и GPGPU

Алгоритмы машинной графики (потоковые, вычислительные, параллельные):

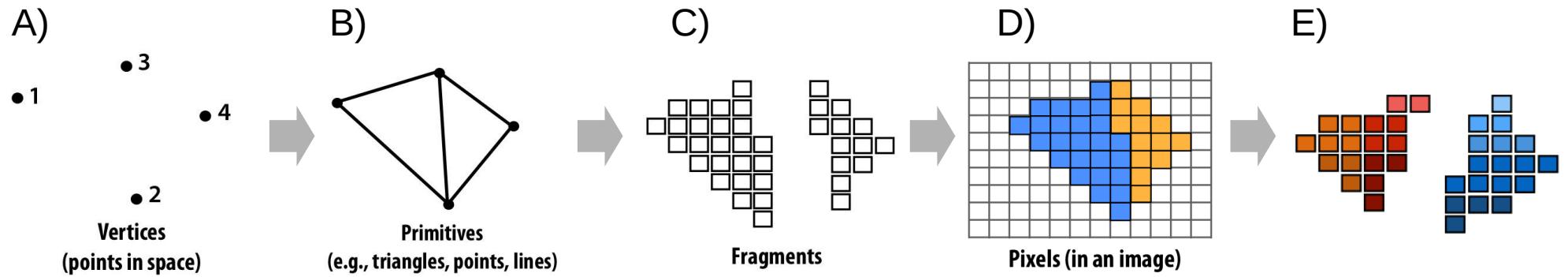
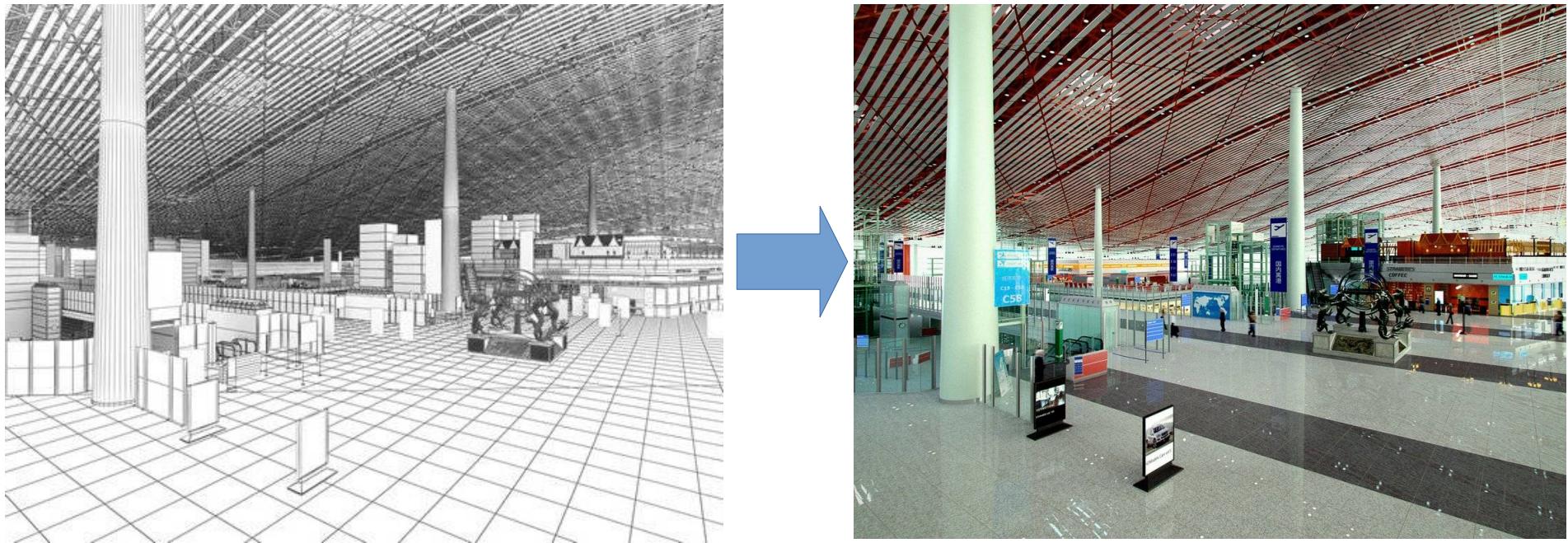
- преобразование систем координат;
- удаление невидимых поверхностей;
- отсечение невидимых областей;
- отрисовка базовых графических примитивов (точек, прямых, ломаных и т.п.);
- заливка / штриховка (растровая развертка сплошных областей);

Графический конвейер



- *Transform* – задание положения каждой вершины в сцене;
- *Clip* – отсечение скрытых областей, в т.ч. за пределами области видимости;
- *Rasterize* – переход от векторного представления к пикельному;
- *Shade* – вычисление цвета каждого пикселя;
- *Visibility/Blend* - расчет наложений и цвета для прозрачных объектов.
- *Display* – окончательное формирование изображения в памяти.

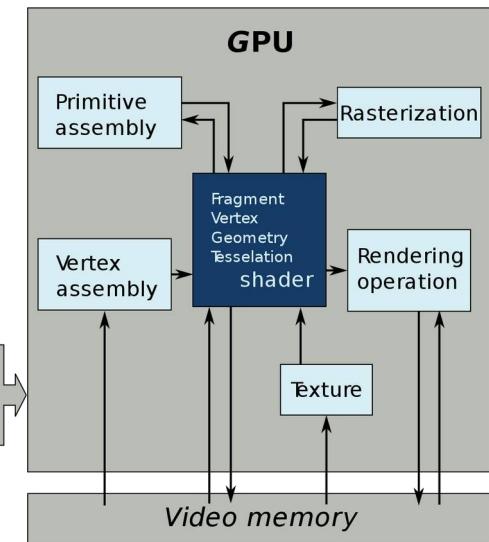
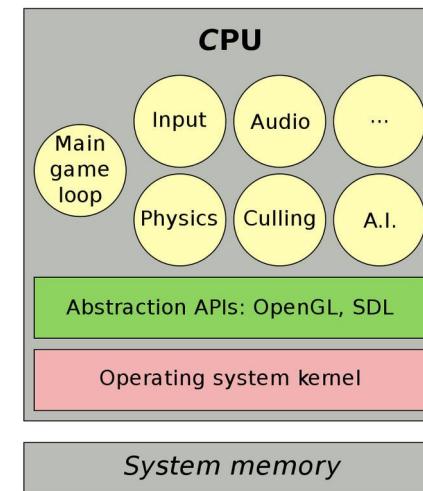
Графический конвейер (пример)



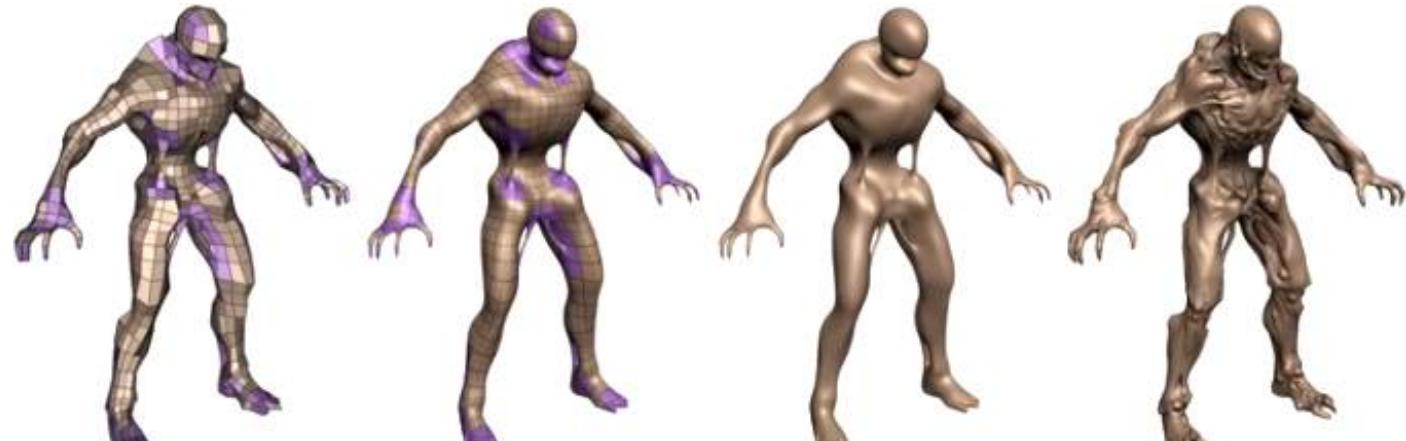
Шейдеры

Шейдер – это программа, которая загружается в ускоритель, и конфигурирует его узлы для обработки соответствующих элементов. Шейдер позволяет снять ограничение на способ обработки эффектов.

- вершинные;
- геометрические;
- пиксельные или фрагментные.



Shader Forge это
визуальный редактор
шейдеров для Unity.



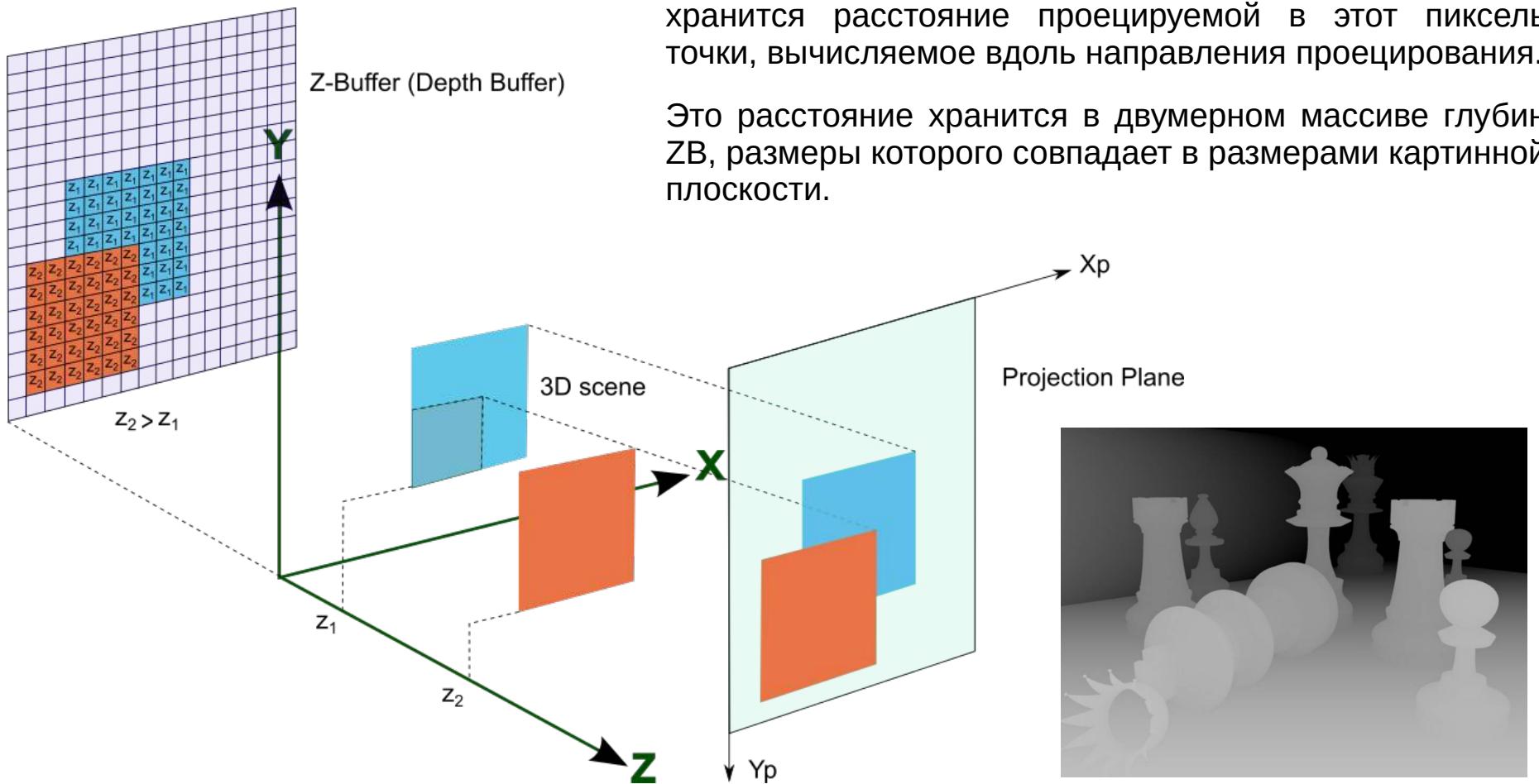
Тесселяция используется для увеличения геометрической сложности моделей, когда из низкополигональной получается более сложная.

Z-буферизация

Z-буферизация — в компьютерной трёхмерной графике способ учёта удалённости элемента изображения.

При решении задачи загораживания методом Z-буфера при выводе текущего полигона формируется его растровое представление на картинной плоскости и для каждого пикселя картинной плоскости, кроме цвета, хранится расстояние проецируемой в этот пиксель точки, вычисляемое вдоль направления проецирования.

Это расстояние хранится в двумерном массиве глубин ZB, размеры которого совпадают в размерами картинной плоскости.



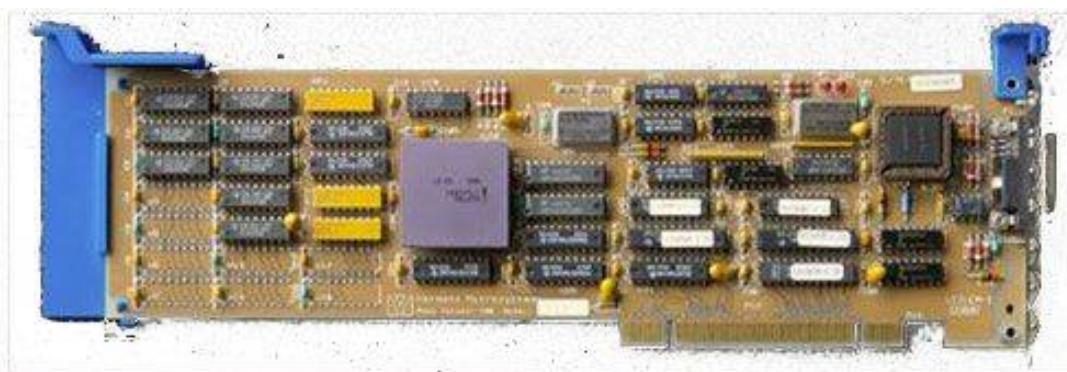
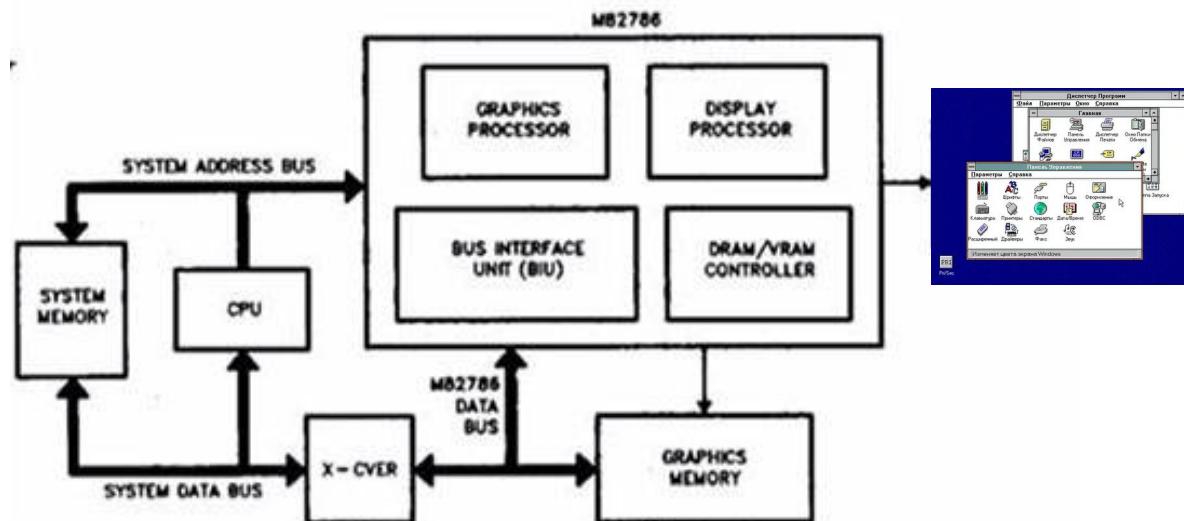
Различия в архитектурах GPU и CPU

CPU	GPU
Ядра CPU проектируются для выполнения одного потока последовательных инструкций с максимальной производительностью	GPU предназначен для выполнения большого количества параллельных потоков команд.
В CPU доступ к памяти зависит от поступивших команд и часто происходит по случайным адресам. Требуется использовать кэш-память для ускорения доступа к памяти.	В GPU доступ к памяти преимущественно последовательный (пиксели и текстуры читаются и пишутся последовательно). Большой кэш не требуется.
Доступ к памяти плохо распараллеливается, данные сосредоточены в сегментах и выборка осуществляется в небольшое количество модулей памяти (по крайней мере, для одной задачи.)	Доступ к памяти легко распараллеливается и пропускная способность каналов памяти велика.
В универсальных процессорах большую часть площади кристалла занимают различные блоки конвейера: декодеры, буферы, ROB, кэш-память и пр.	Аппаратная часть GPU оптимизирована под выполнение небольших и программ (шейдеров).
CPU хорошо справляется с зависимыми данными.	GPU предназначен для вычислений независимых данных (пикселей, текстур). При наличии зависимостей скорость вычислений существенно падает.

Первый дискретный графический сопроцессор Intel 82786

Графический процессор (GP) и дисплейный процессор (DP) были независимыми процессорами в 82786. Шинный интерфейсный блок (BIU) с контроллером DRAM / VRAM обрабатывал запросы шины между графическим процессором, дисплеем и внешним ЦП или шиной.

Графический процессор выполнял команды, размещенные в системной памяти и формирует изображения в битовых картах видеопамяти для дисплейного процессора во взаимодействии с контроллером видеопамяти и интерфейсным устройством шины.



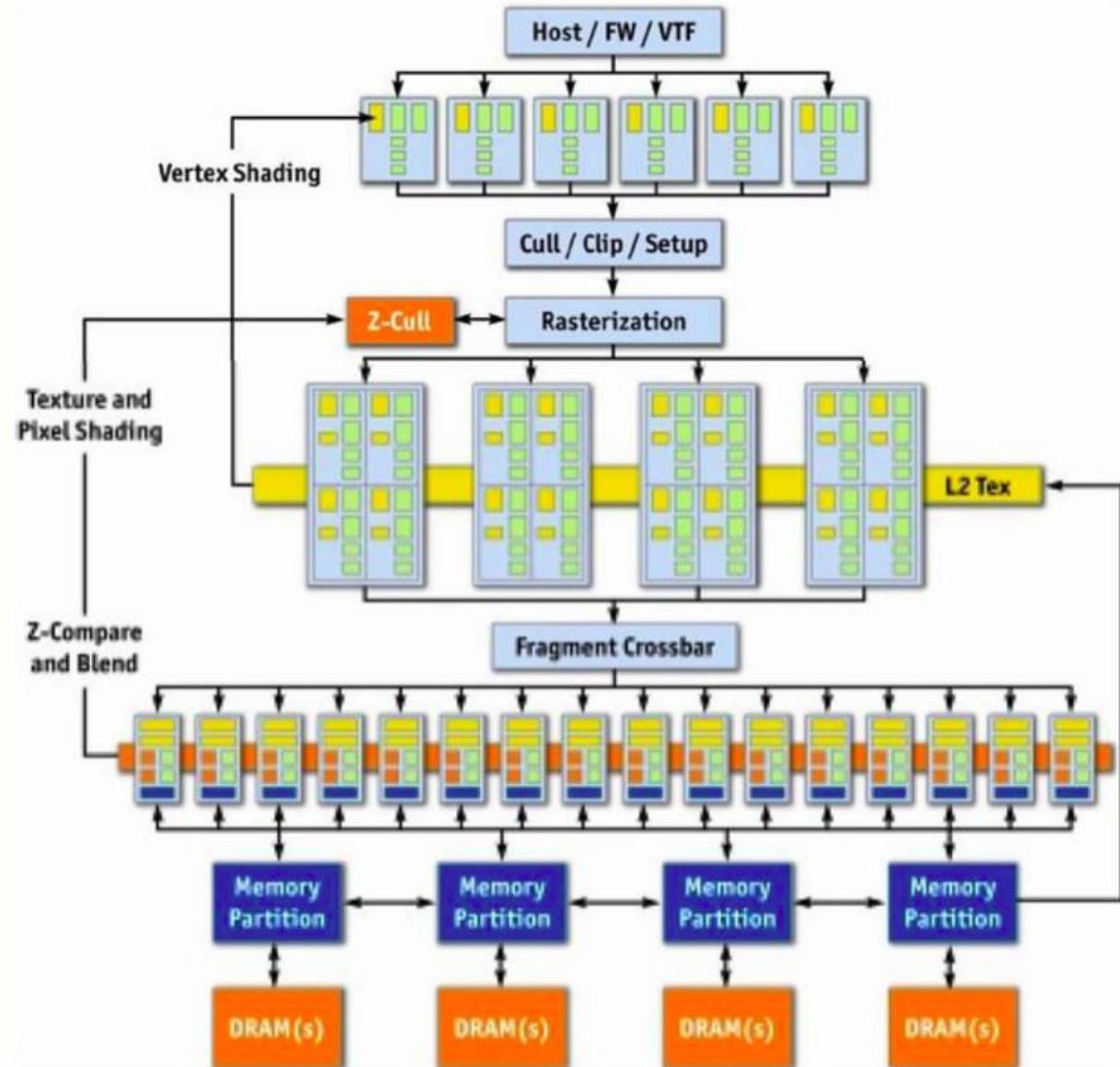
Дисплейный процессор преобразует битовые карты, создаваемые графическим процессором в растровые последовательности для видеоконтрольного устройства, которое отображает их в виде отдельных окон на экране графического монитора.

Современные графические процессоры GPU

Nvidia GeForce6 (NV40), 2004

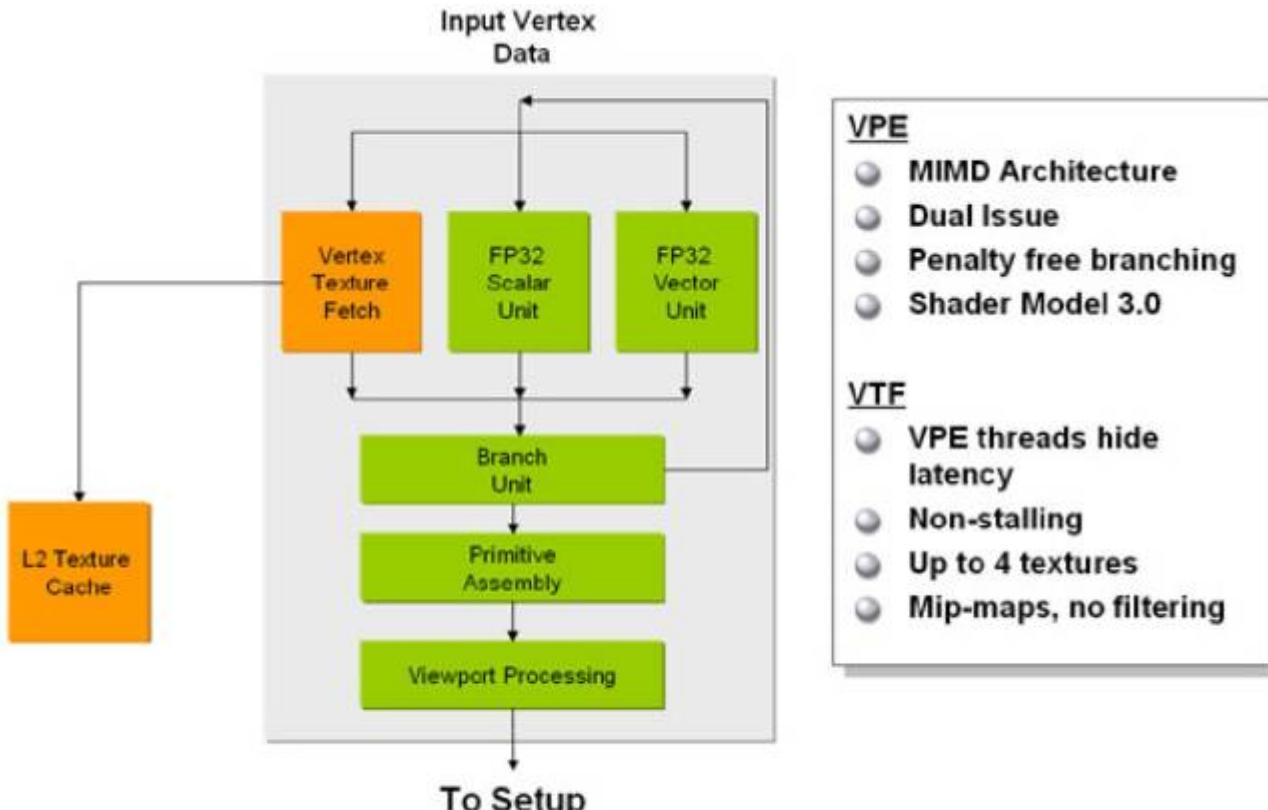
Этапы обработки:

- Загрузка в чип ускорителя вершин из памяти акселератора
- Обработка в вершинных процессорах.
- Установка треугольников, отсечение невидимых поверхностей.
- Треугольники растеризуются и производится отсечение невидимых частей с использованием Z-буфера (Hidden Surface Removal, HSR).
- Формируются квады 2x2 пикселя, подлежащие закраске. На квады накладываются текстурные фрагменты и производится интерполяция.
- Закраска фрагментов.
- Производится блендинг (смещение)
- Значения квадов записываются в буфер кадра
- Вывод изображения на экран.



Современные графические процессоры GPU

Nvidia GeForce6 Вершинные процессоры

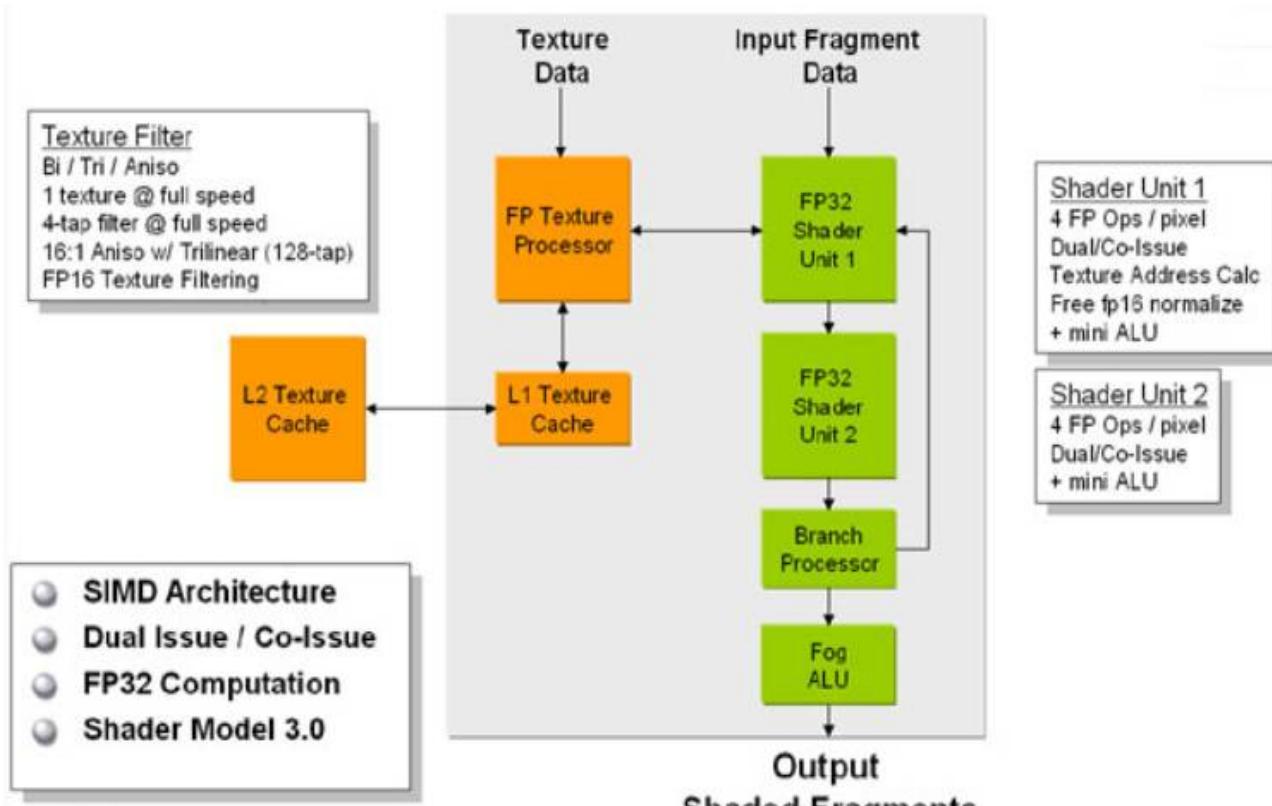


Благодаря расширению динамического выполнения (больше вариантов по циклам/ветвлениям и новые функции подпрограмм), можно создавать более эффективный код и реализовывать новые возможности для эффектов.

- полная поддержка вершинных программ 3.0;
- 216 (65,535) длинных вершинных программ;
- вершинная обработка с учётом текстуры - карты смещения (displacement mapping);
- динамический контроль выполнения (flow control) - циклы и ветвления, вызов подпрограмм и возврат;
- Geometry Instancing (vertex stream divider).

Современные графические процессоры GPU

Nvidia GeForce Пиксельные конвейеры

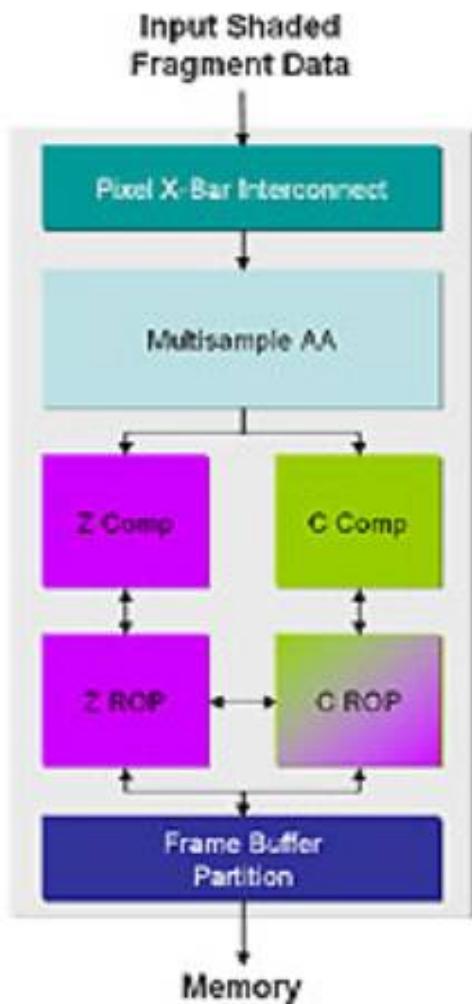


- Каждый из 16 пиксельных конвейеров имеет два блока пиксельных программ (суперскалярный дизайн) и один текстурный процессор с плавающей запятой.
- NV40 также оснащён четырьмя кэшами текстур L1, каждый из которых обслуживает четыре конвейера.
- Разгрузить интерфейс памяти помогает и массивный кэш L2.
- Архитектура блоков пиксельных программ-шейдеров имеет настоящий дизайн SIMD (одна инструкция - много данных).
- Если первый блок пиксельных программ на каждом конвейере может выполнять как арифметические операции, так и чтение текстур и нормализацию, то второй блок ограничен только арифметикой.
- Если блок не занимается текстурированием, то он может выполнять (в данный проход) пиксельное затенение. Блок 2 всегда доступен для пиксельного затенения.

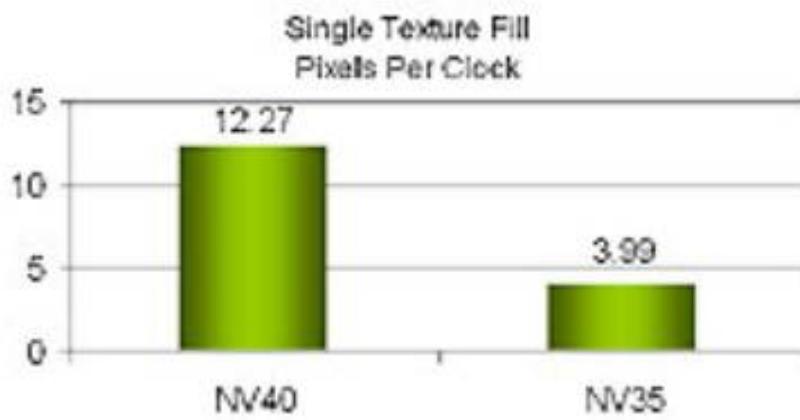
Современные графические процессоры GPU

Nvidia GeForce6

Пиксельные конвейеры ROP (растровые операции)



- OpenEXR Floating point blending
- Rotated Grid AA
- Double-speed Z
- Lossless Color & Z Compression
- Multiple Render Targets



Подсистема ROP карты GeForce 6800 имеет следующие характеристики:

- 16 пикселей за такт, цвет и Z;
- 32 пикселей за такт, только Z;
- 64-bit FP Frame Buffer Blending;
- цветовое и Z-сжатие без потерь;
- качественное сглаживание - поворот сетки (Rotated Grid);
- полная поддержка MRT;
- ускоренный рендеринг теней.

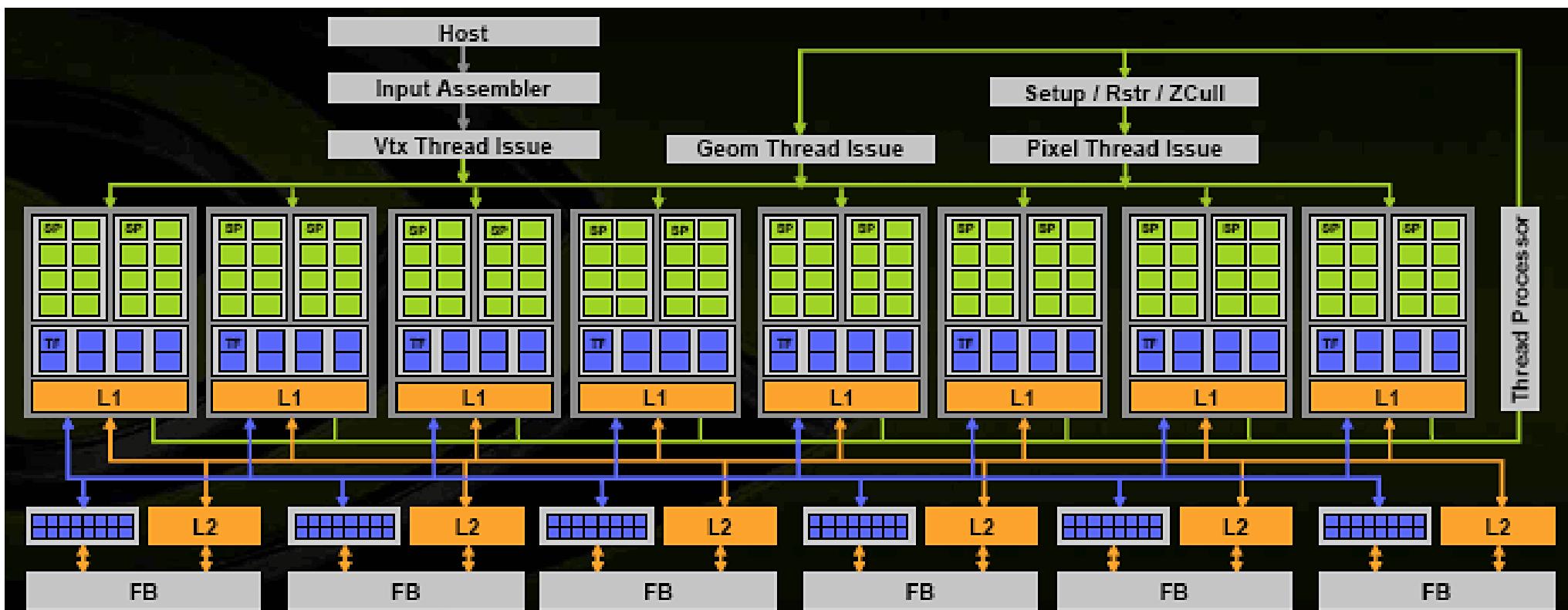
Современные графические процессоры GPGPU

NVidia G80

Чип состоит из 8 универсальных вычислительных блоков (шейдерных процессоров) по 4 TMU и 16 ALU в каждом. Всего, таким образом, имеется 128 ALU (называются потоковыми процессорами (SM, Stream Processors) и 32 TMU. Все ветвления, переходы, условия и т.д. применяются целиком к одному блоку и таким образом логичнее всего, его и называть шейдерным процессором, пускай и очень широким.

Каждый такой процессор снабжен собственным кэшем первого уровня, в котором теперь хранятся не только текстуры, но и другие данные, которые могут быть запрошены шейдерным процессором.

Кроме управляющего блока и 8 вычислительных шейдерных процессоров в наличии 6 блоков ROP, исполняющих определение видимости, запись в буфер кадра и MSAA (синие, рядом с блоками кэша L2) сгруппированные с контроллерами памяти, очередями записи и кэшем второго уровня.



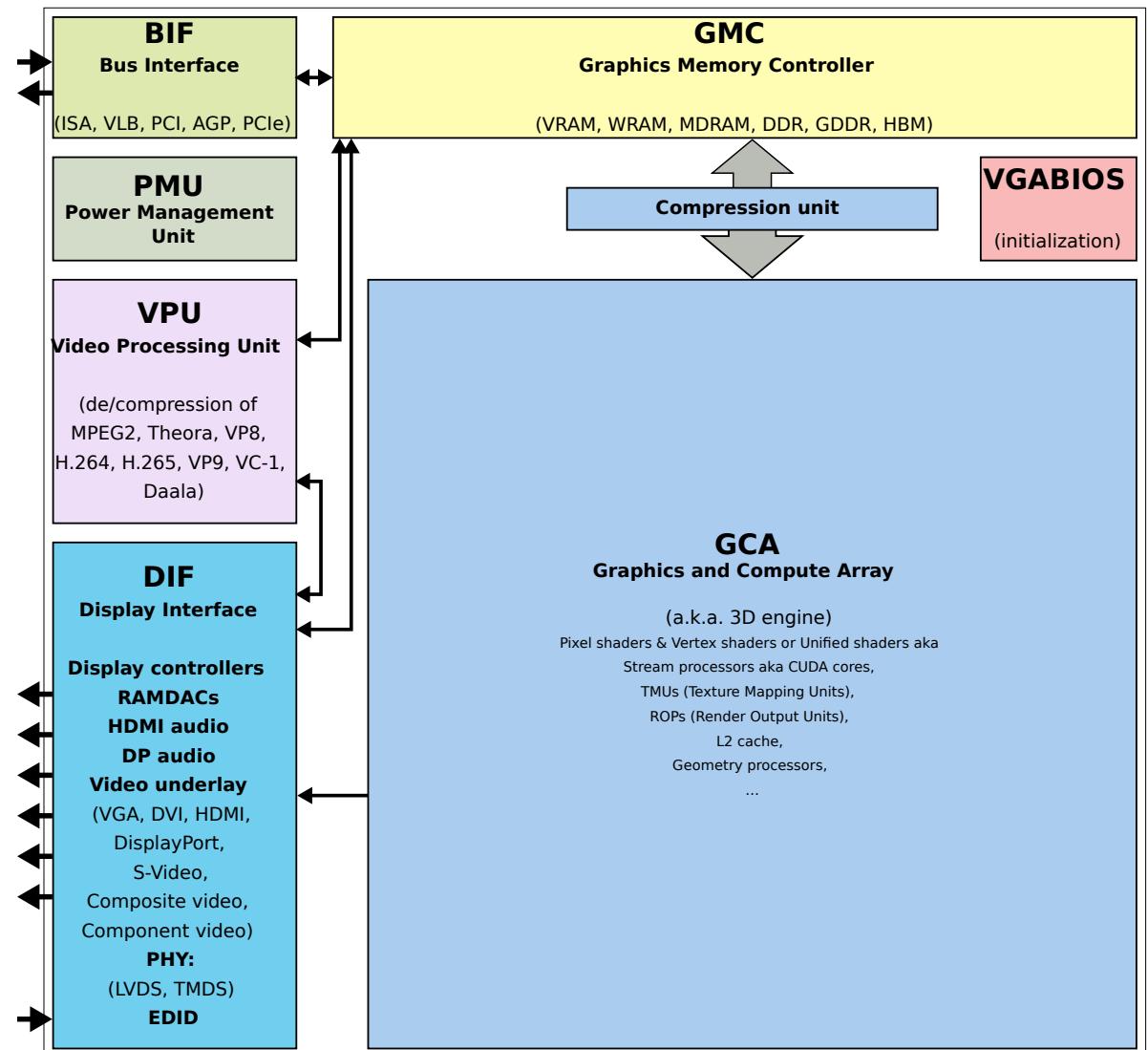
Обобщенная схема GPGPU

Отличительными особенностями по сравнению с ЦП являются:

- архитектура, максимально нацеленная на увеличение скорости расчёта текстур и сложных графических объектов;
- ограниченный набор команд.
-

Высокая вычислительная мощность GPU объясняется особенностями архитектуры. Современные CPU содержат несколько ядер, тогда как графический процессор изначально создавался как многопоточная структура с множеством ядер. Разница в архитектуре обуславливает и разницу в принципах работы. Если архитектура CPU предполагает последовательную обработку информации, то GPU исторически предназначался для обработки компьютерной графики, поэтому рассчитан на массивно параллельные вычисления.

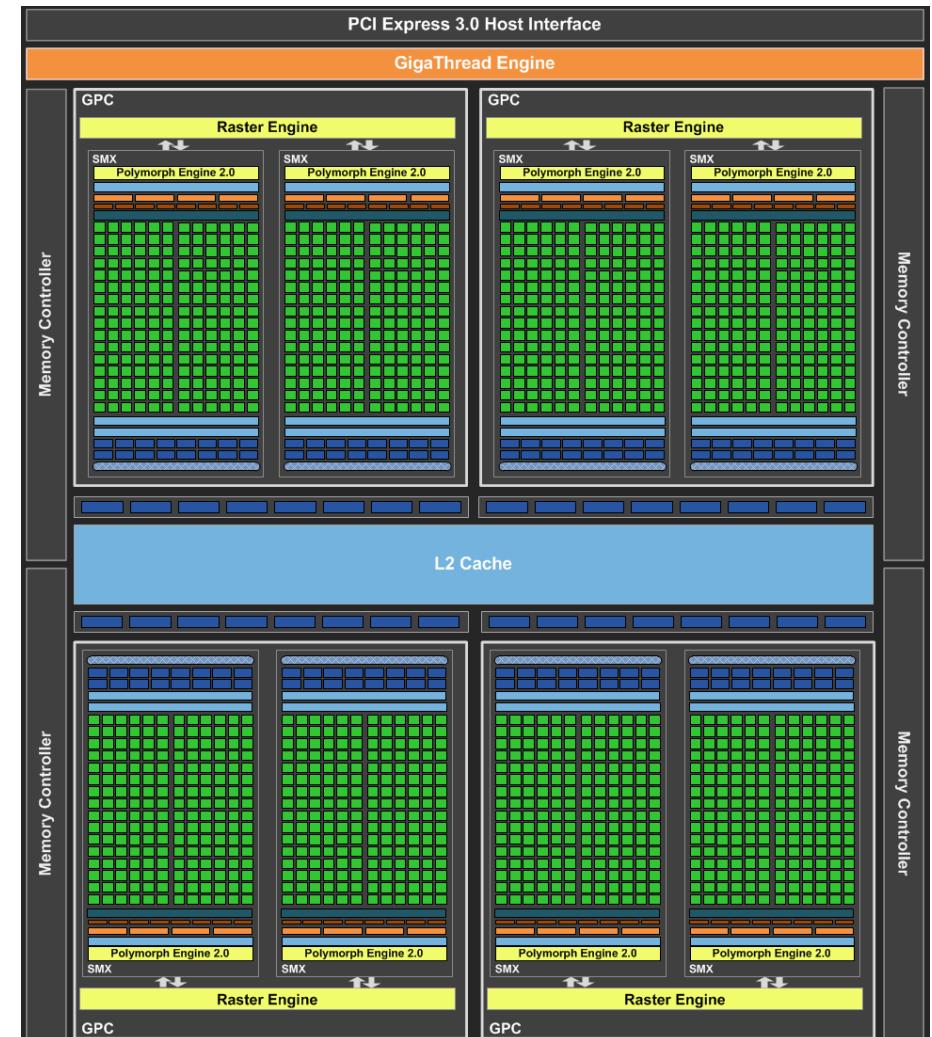
Каждая из этих двух архитектур имеет свои достоинства. CPU лучше работает с последовательными задачами. При большом объёме обрабатываемой информации очевидное преимущество имеет GPU. Условие только одно — в задаче должен наблюдаться параллелизм.



Современные графические процессоры GPU

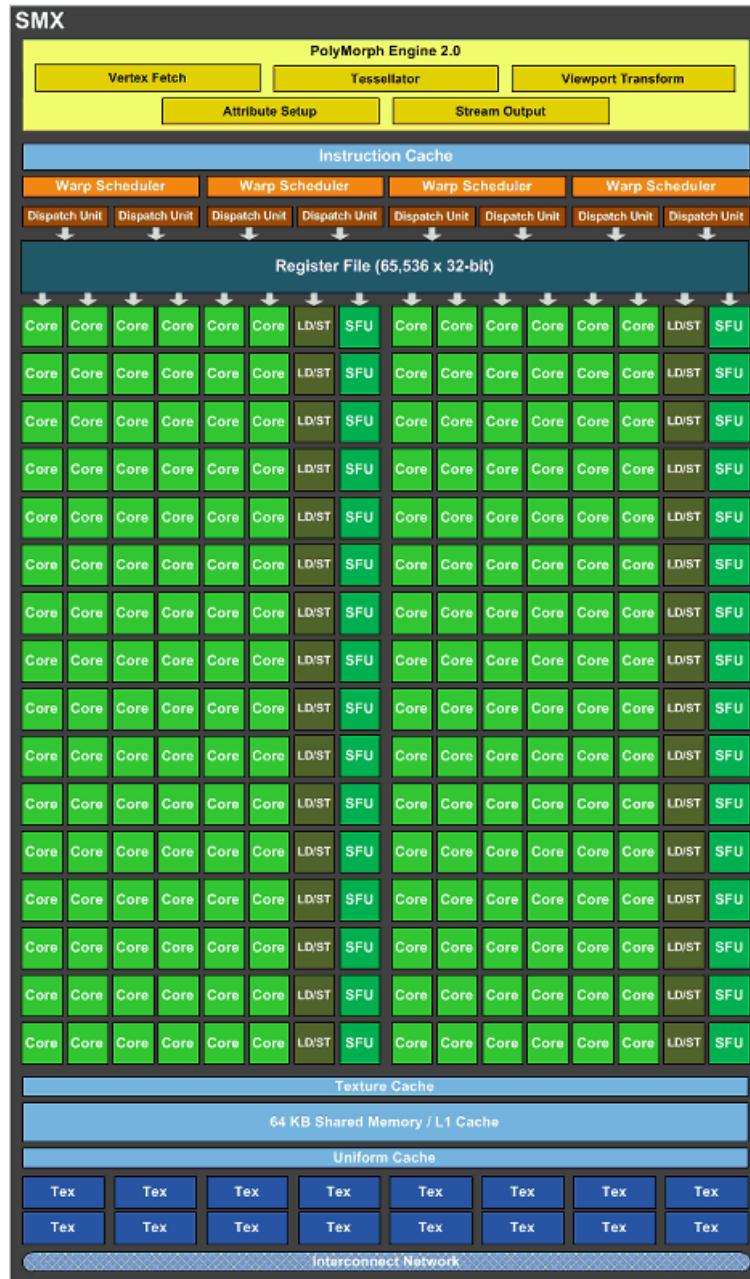
NVidia GeForce GTX 600

- Технология производства 28 нм;
- 3.54 миллиардов транзисторов;
- Площадь ядра 294 мм²;
- Унифицированная архитектура с массивом процессоров для потоковой обработки различных видов данных: вершин, пикселей и др.;
- Аппаратная поддержка DirectX 11 API, в том числе шейдерной модели Shader Model 5.0, геометрических и вычислительных шейдеров, а также тесселяции;
- 256-битная шина памяти, четыре независимых контроллера шириной по 64 бита каждый, с поддержкой GDDR5 памяти;
- Базовая частота ядра 1006 МГц;
- Средняя турбо-частота ядра 1058 МГц;
- 8 потоковых мультипроцессоров, включающих 1536 скалярных ALU для расчётов с плавающей запятой (поддержка вычислений в целочисленном формате, с плавающей запятой, с FP32 и FP64 точностью в рамках стандарта IEEE 754-2008);
- 128 блоков текстурной адресации и фильтрации с поддержкой FP16 и FP32 компонент в текстурах и поддержкой трилинейной и анизотропной фильтрации для всех текстурных форматов;
- 4 широких блока ROP (32 пикселя) с поддержкой режимов антиалиасинга до 32 выборок на пиксель, в том числе при FP16 или FP32 формате буфера кадра. Каждый блок состоит из массива конфигурируемых ALU и отвечает за генерацию и сравнение Z, MSAA, блендинг;
- Интегрированная поддержка RAMDAC, двух портов Dual Link DVI, а также HDMI и DisplayPort.
- Интегрированная поддержка четырёх мониторов, включая два порта Dual Link DVI, а также HDMI 1.4a и DisplayPort 1.2
- Поддержка шины PCI Express 3.0



Современные графические процессоры GPU

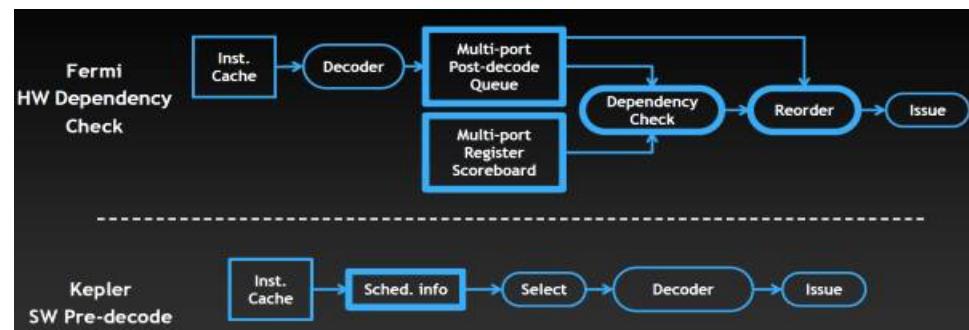
NVidia GeForce GTX 600



Мультипроцессоры — это основная составная часть GPU компании NVIDIA. По сравнению с предыдущими SM, новые SMX обеспечивают более высокую производительность, что видно по количеству функциональных устройств в составе SMX, но при этом потребляют значительно меньше энергии. А уменьшенное количество мультипроцессоров на GPU (8 в отличие от 16 в GF100/GF110) было продиктовано установленными рамками по площади ядра.

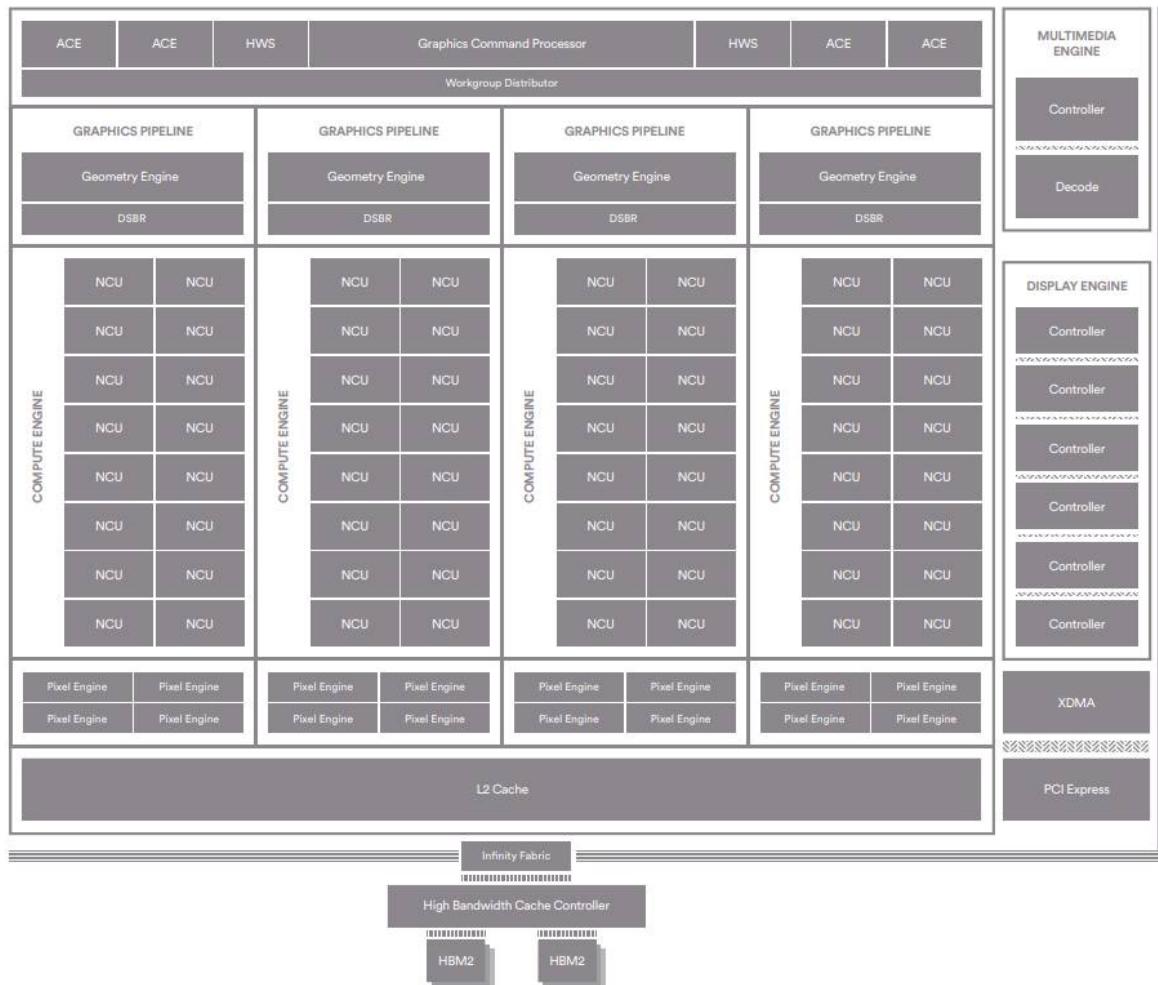
Большая часть ключевых блоков GPU включена в состав SMX: потоковые процессоры (CUDA Cores) выполняют все математические операции над пикселями, вершинами и занимаются неграфическими вычислениями, текстурные модули (TMU) фильтруют текстурные данные, загружают и записывают их из/в видеопамять, блоки специальных функций (Special Function Units, SFU) выполняют сложные операции (вычисление синуса, косинуса, квадратного корня и т.п.) и интерполяции графических атрибутов. Ну а движок PolyMorph обеспечивает выборку вершин, занимается тесселяцией, преобразованием в экранные координаты, установкой атрибутов и потоковым выводом (stream output).

Процессор содержит сложную аппаратную стадию, служащую для предотвращения конфликтов доступа к данным. Специальная таблица регистров (multi-port register scoreboard) отслеживает регистры, данные в которых ещё не готовы, а блок проверки зависимостей (dependency check) анализирует их использование, проверяя зависимости команд.

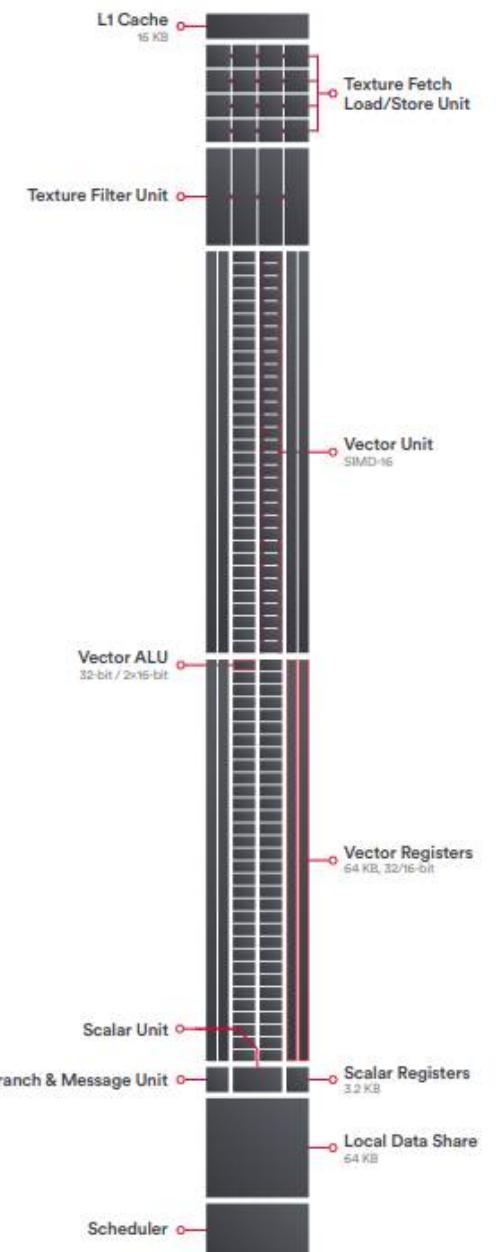


Современные графические процессоры GPU

AMD Radeon VII



Количество универсальных процессоров – 3840; Количество текстурных блоков – 240; Количество блоков блендинга – 64; Эффективная частота памяти - 2000 МГц; Тип памяти – HBM2; Шина памяти – 4096-бит; Объем памяти - 16 ГБ; Пропускная способность памяти - 1 ТБ/с; Количество транзисторов – 13.2 млрд.



Полезные ссылки

- 1) DX Current: Настоящее аппаратного ускорения графики: <https://www.ixbt.com/video2/dx-current.shtml>
- 2) Современная терминология 3D графики: <https://www.ixbt.com/video2/terms2k5.shtml#dm>