

Computer Science Basics

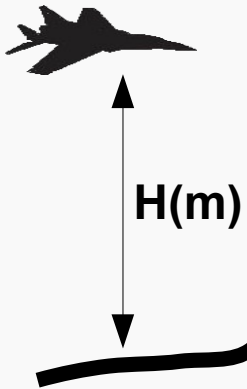
Lecturer: Serge Klimenkov
v.1.14.e from 08.06.2018

How computer works?

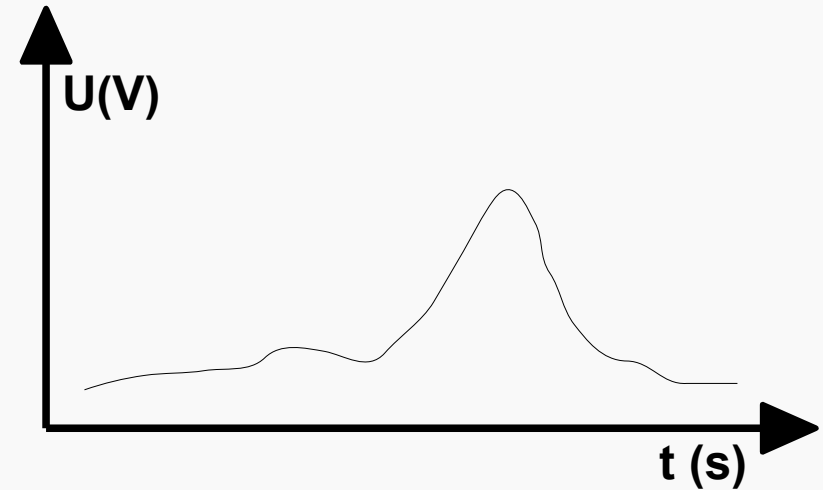
1



Analog Computers

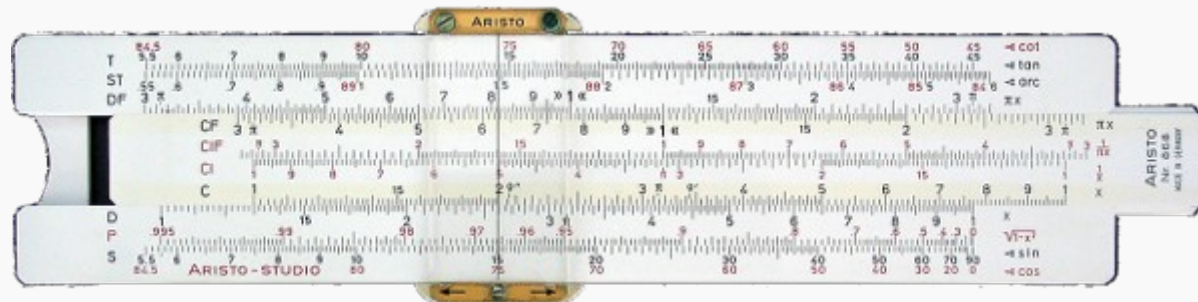


$$H(m) \sim U(V)$$

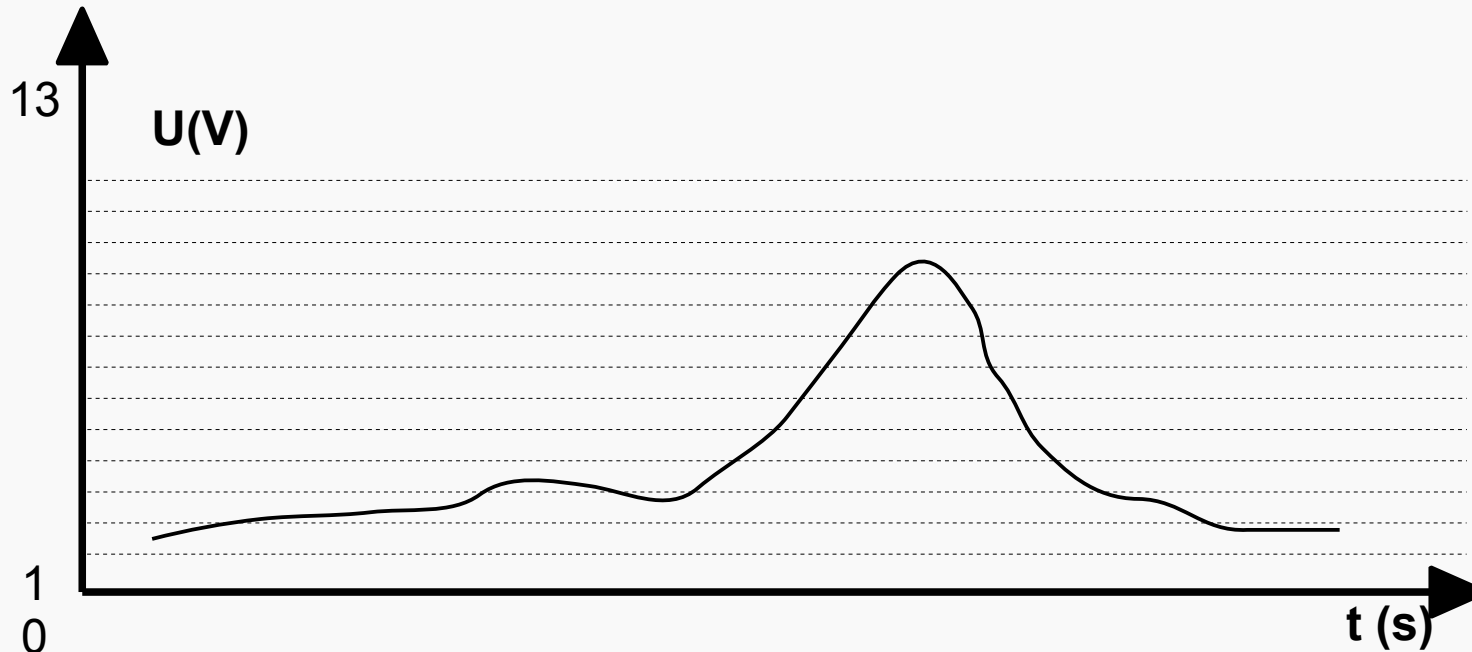


Only suitable for a particular class of tasks

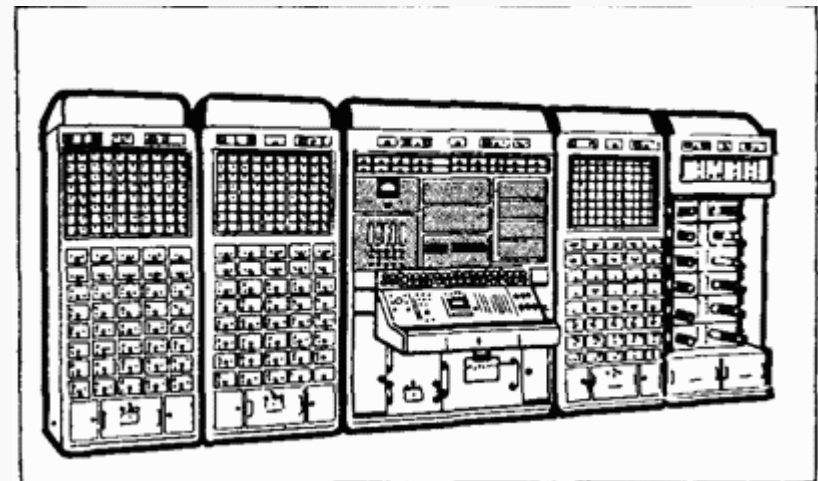
- High performance
- Lower computational errors



Analog Computers

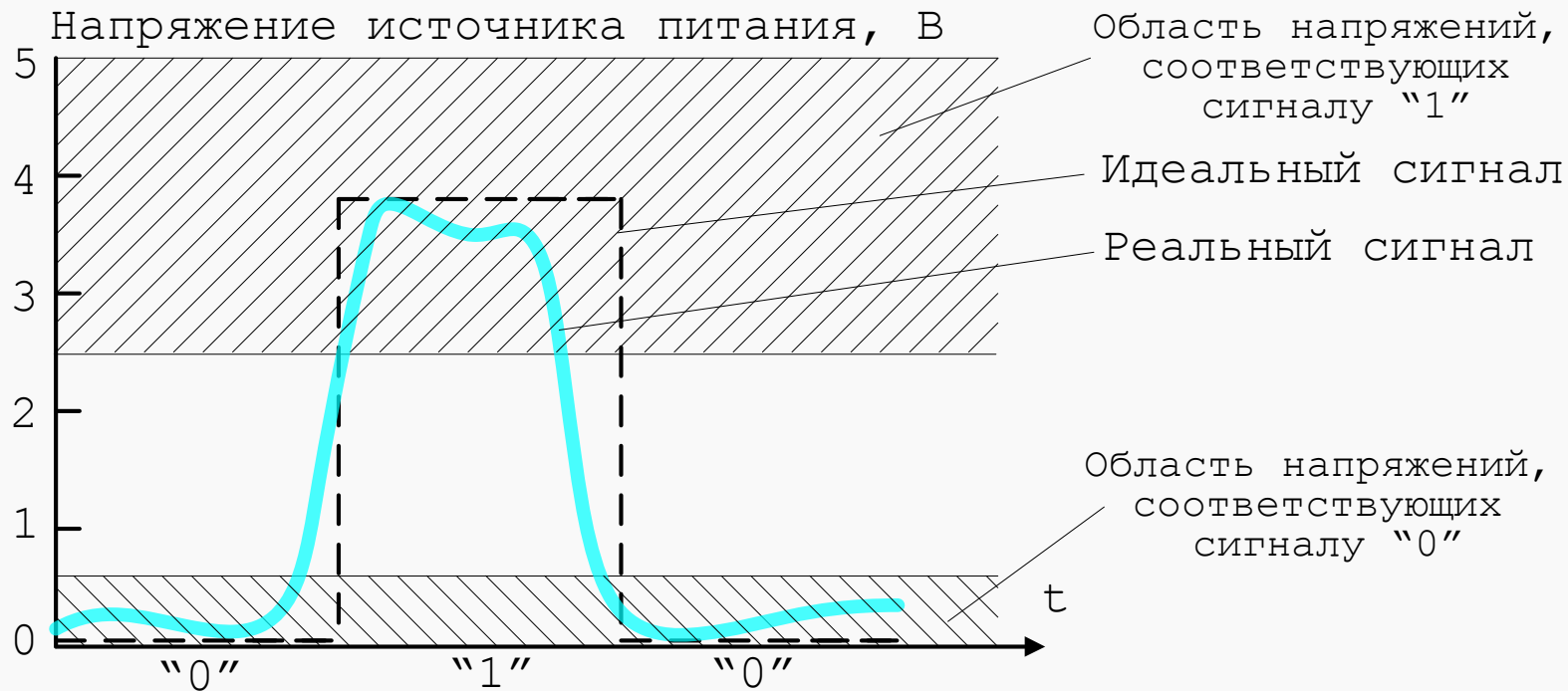


- High accuracy of data representation
- Relatively large dimensions



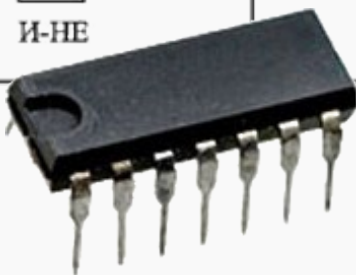
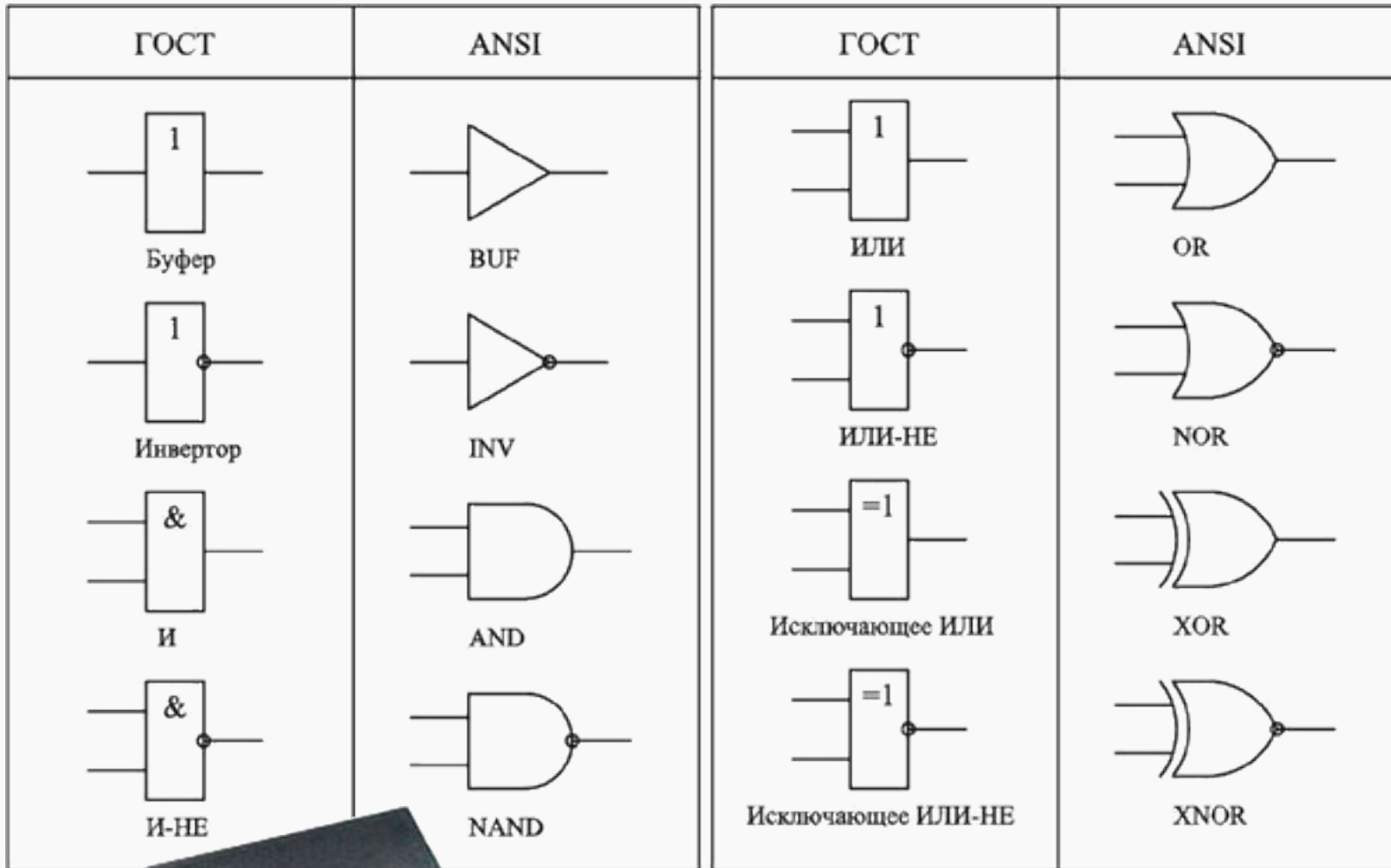
Digital Computers

- All information is represented using only two discrete values — 0 and 1



Building Blocks of the Computer

- Logic elements:



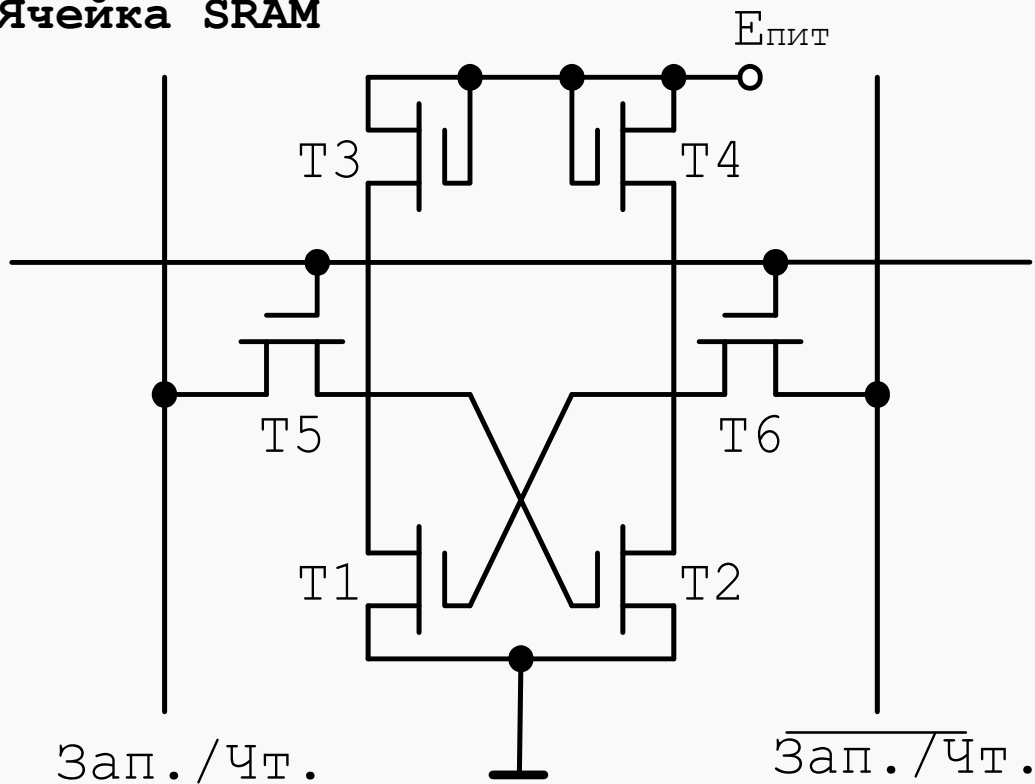
AND			
X1	X2	Y	\bar{Y}
0	0	0	1
0	1	0	1
1	0	0	1
1	1	1	0

OR			
X1	X2	Y	\bar{Y}
0	0	0	1
0	1	1	0
1	0	1	0
1	1	1	0

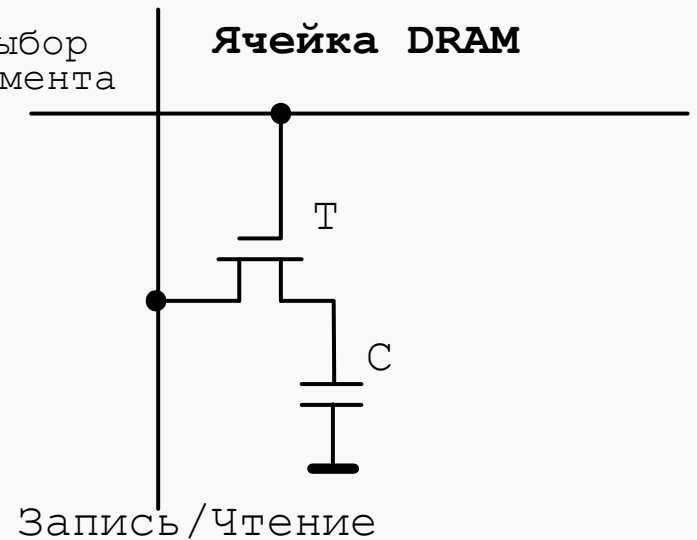
XOR			
X1	X2	Y	\bar{Y}
0	0	0	1
0	1	1	0
1	0	1	0
1	1	1	1

- Memory units (DRAM/SRAM)

Ячейка SRAM

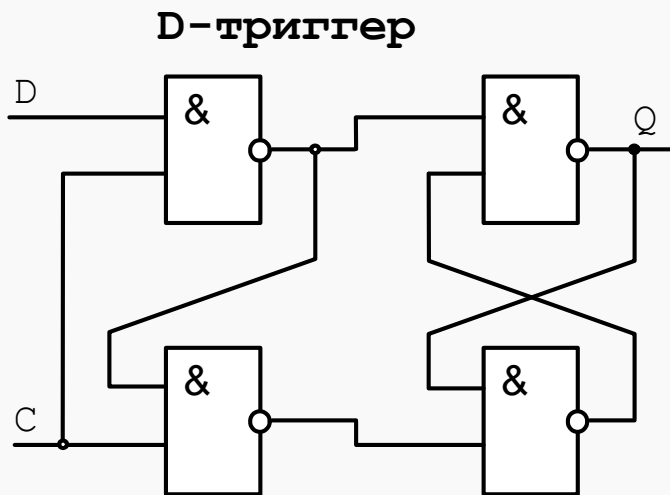


Выбор элемента
Ячейка DRAM



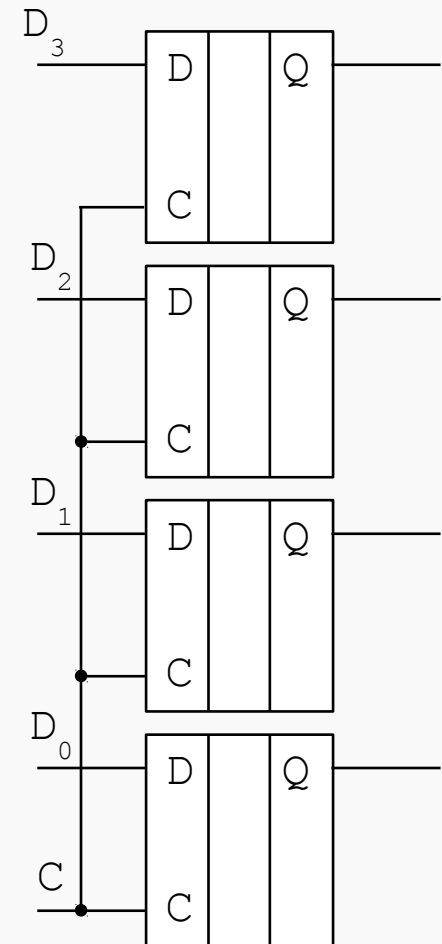
Building Blocks of the Computer

- Memory units (flip-flop circuits, registers)



15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0

0 1 1 0 0 0 1 0 0 0 0 0 1 0 1 1



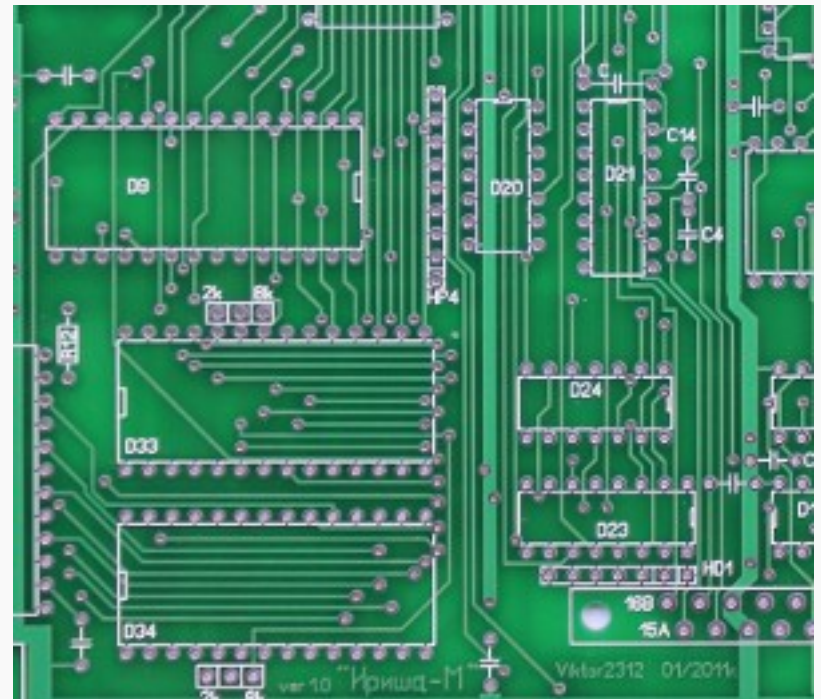
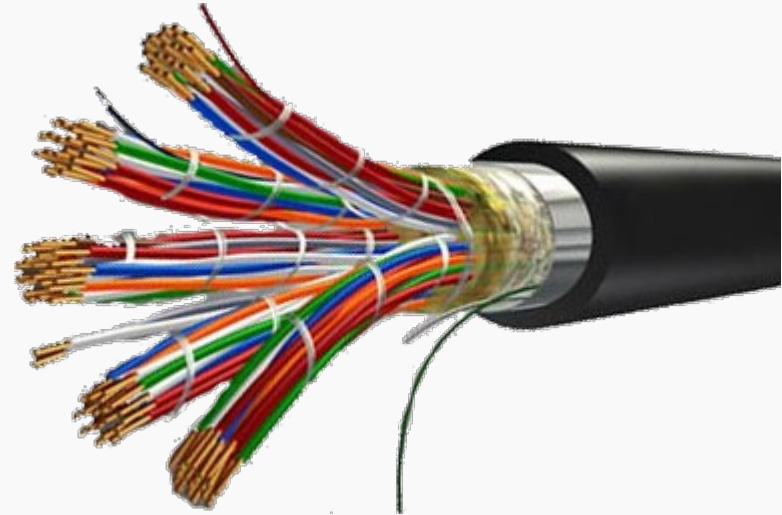
Building Blocks of the Computer

- Cables, buses

Регистр
(источник информации)

0 1 0 1 1 1 0 1

Шина



Building Blocks of the Computer

- Logic gates:

Регистр

(источник информации)

0 1 0 1 1 1 0 1

Управ-
ляющий
сигнал

Шины

0 1 0 1 1 1 0 1

Регистр

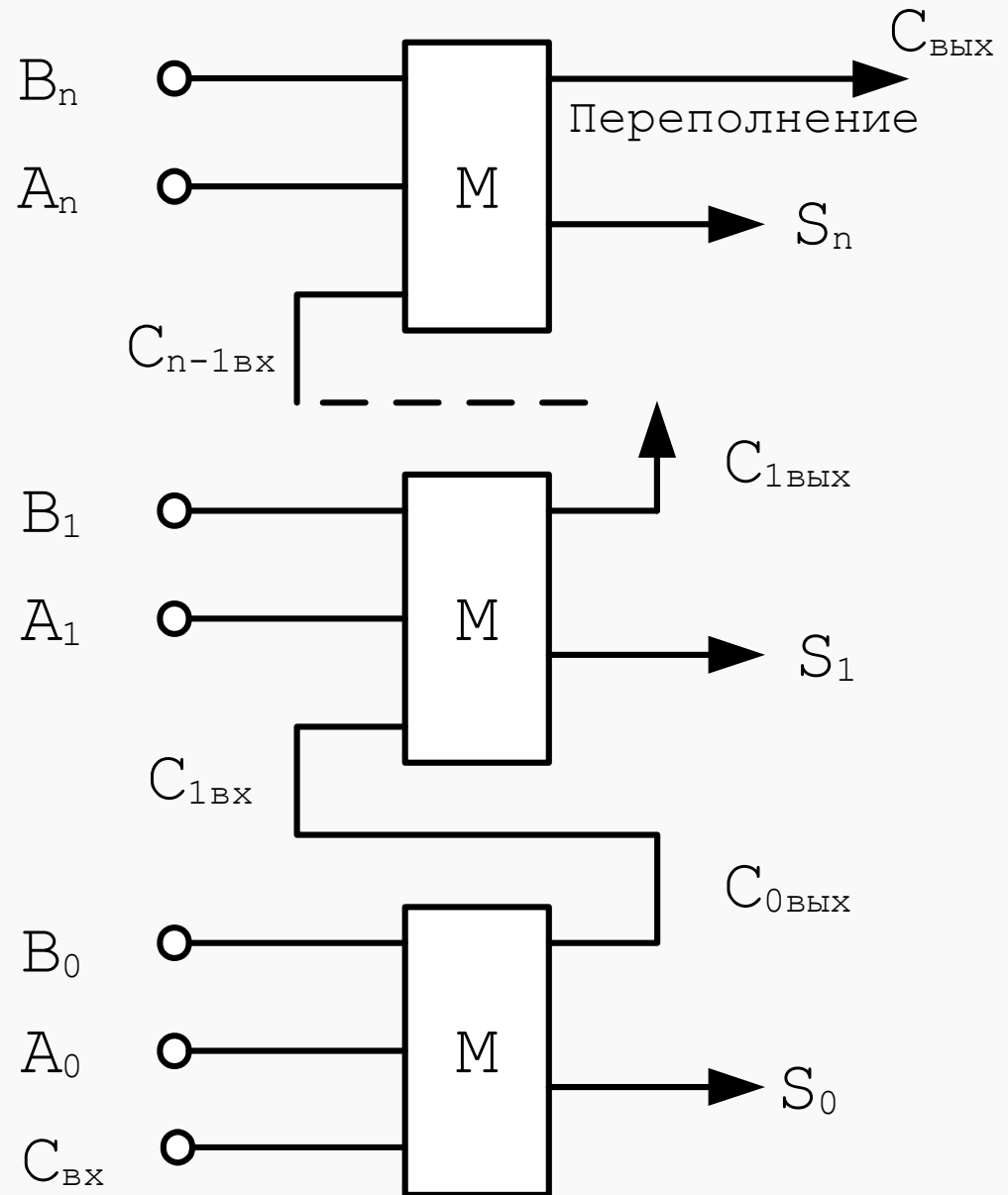
(приемник информации)

Logic gate (AND)		
Contr.	In	Out
0	0	0
0	1	0
1	0	0
1	1	1

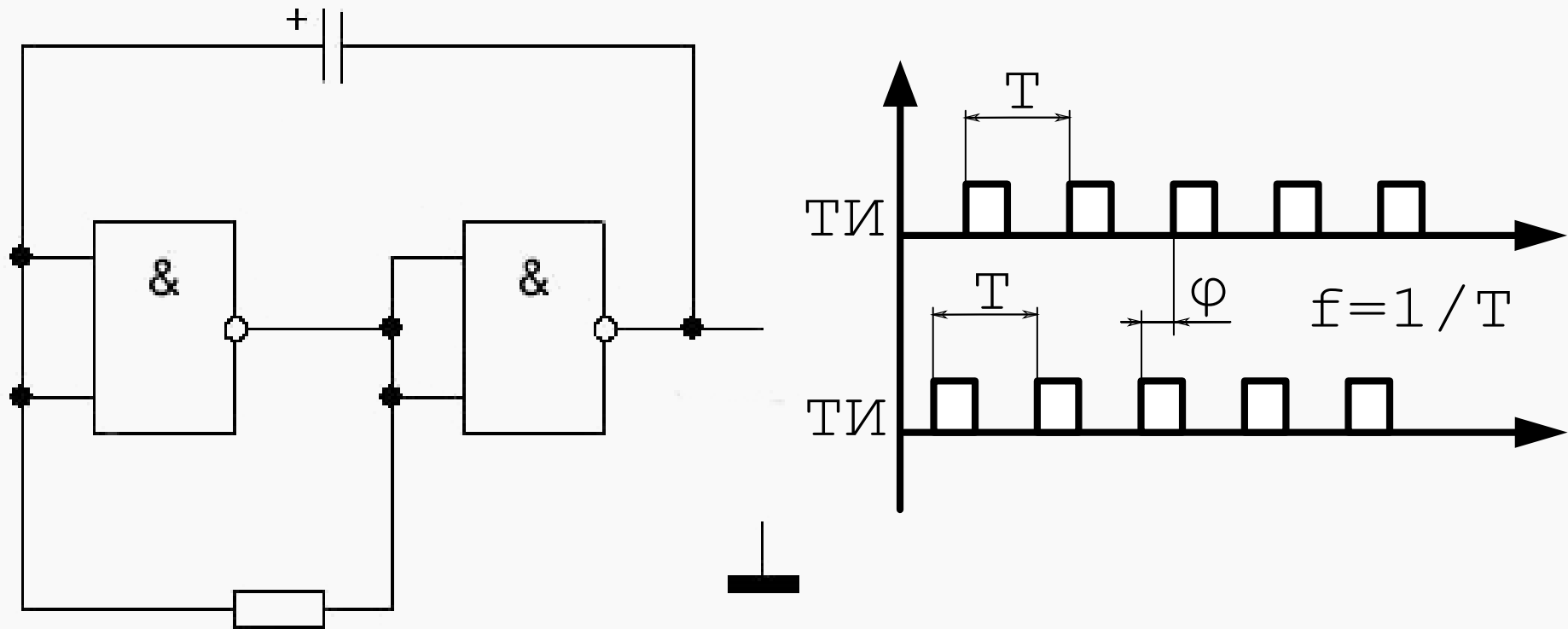


- Adders
(part of the ALU)

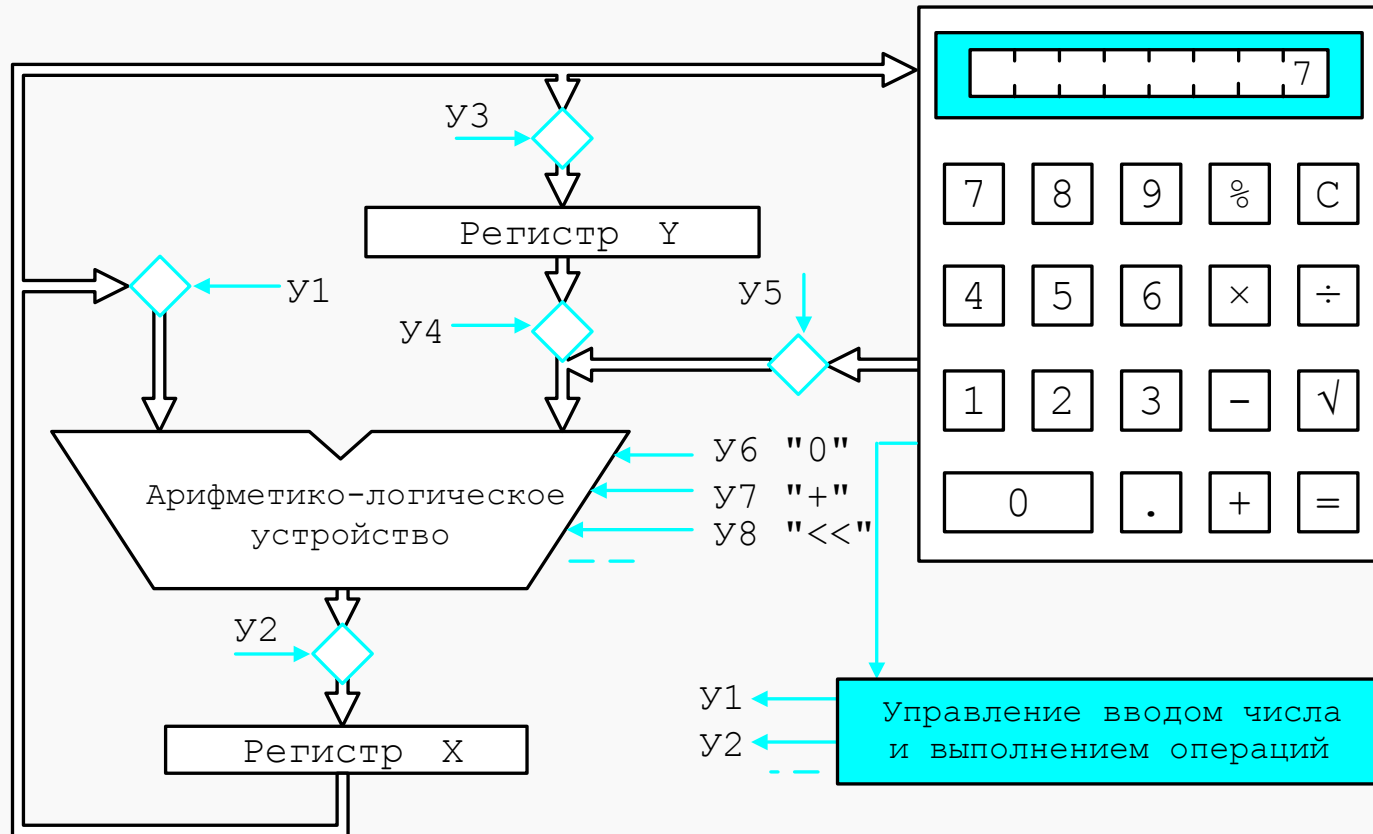
SUM				
Sin	Ai	Bi	Sout	Si
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1



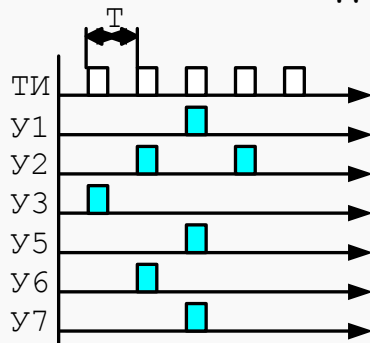
- Clock generators



The First Computer: Calculator (1)

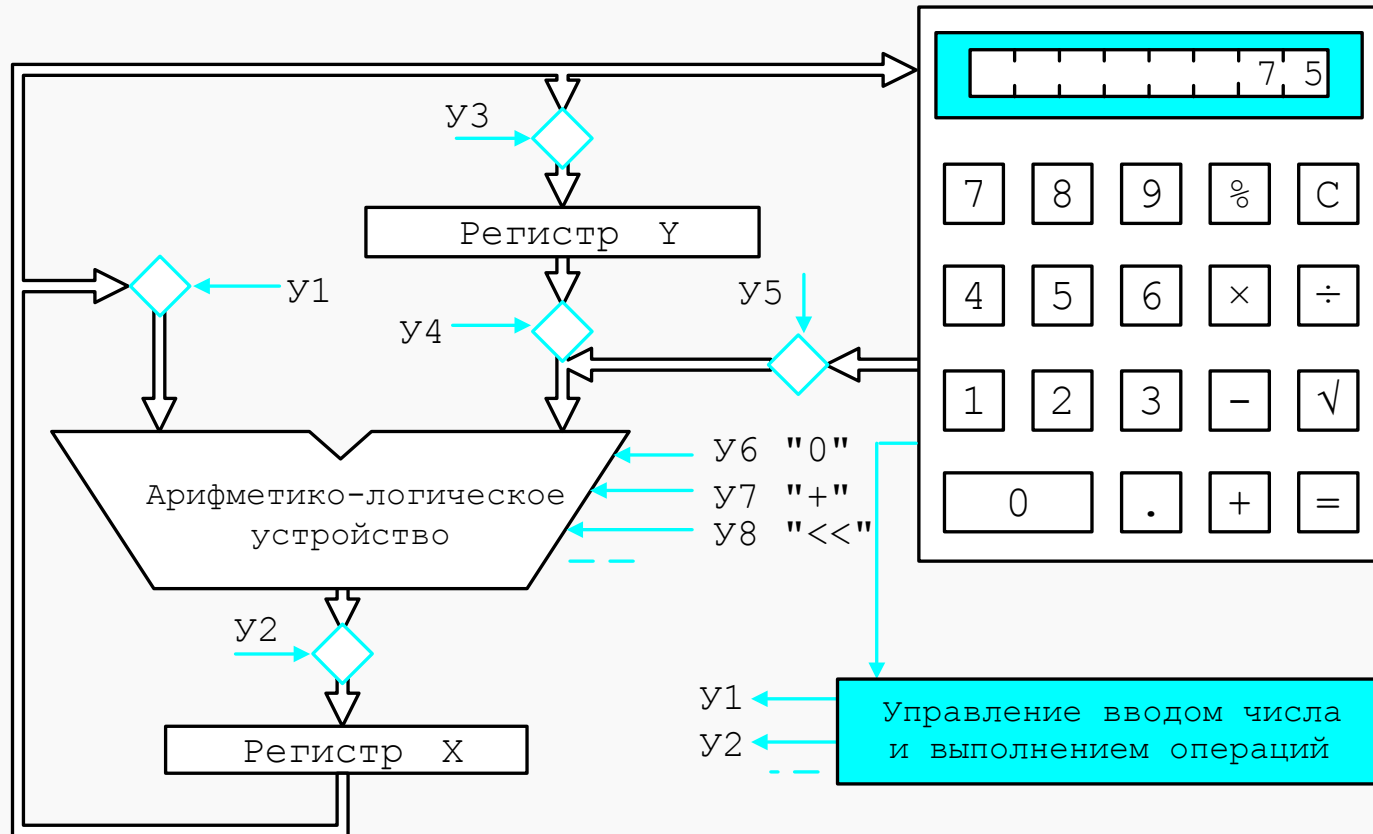


Ввод первой цифры числа

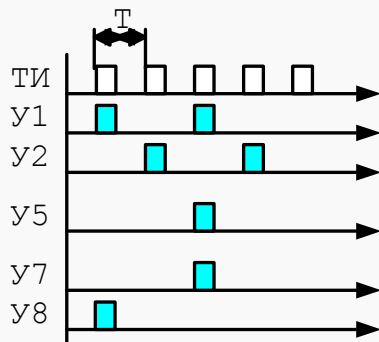


1. (U3) Переслать содержимое регистра X в регистр Y
2. (U2, U6) Записать "0" в регистр X
3. (U1, U5, U7) Сложить X (0) с цифрой с клавиатуры
4. (U2) Записать результат в X

The First Computer: Calculator (2)

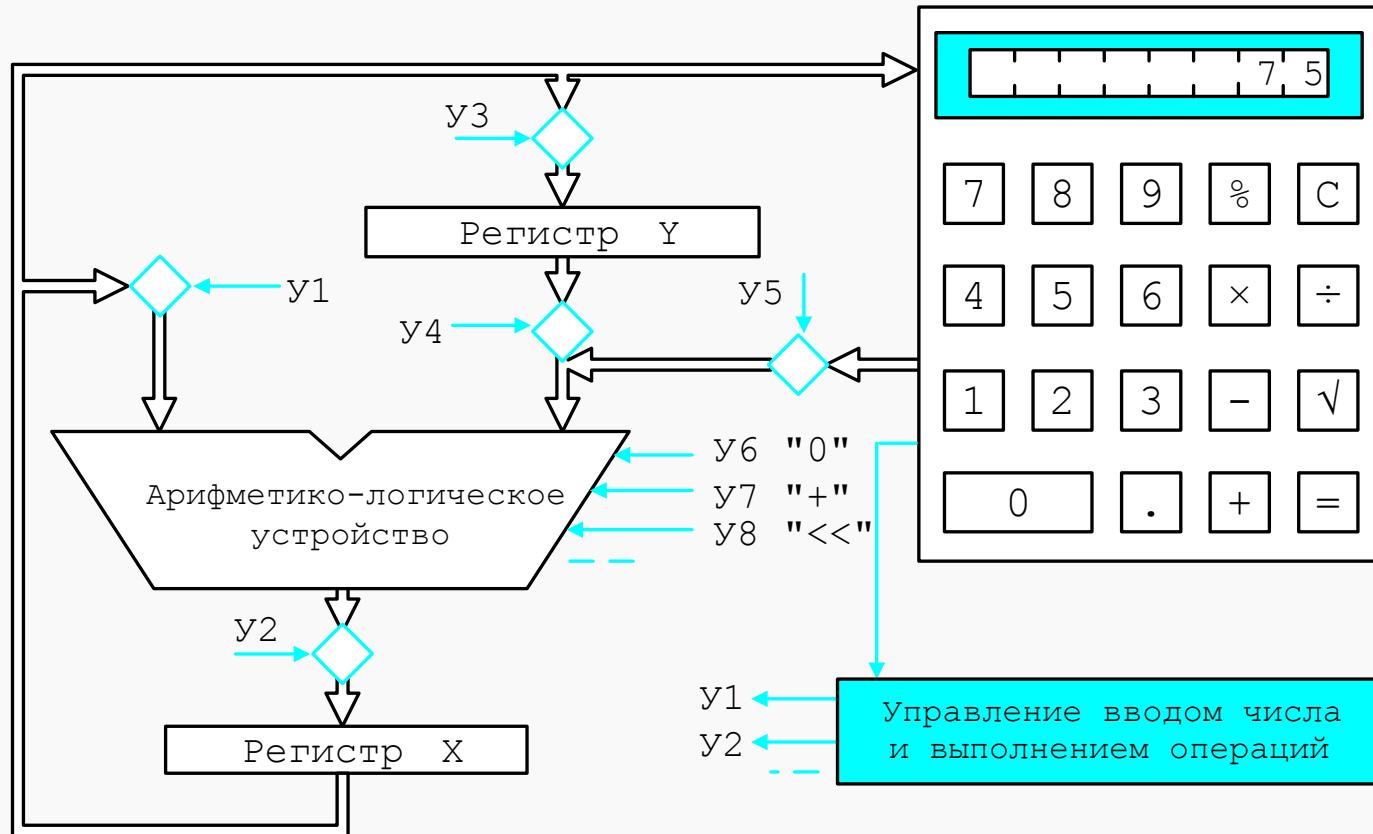


Ввод второй (и последующих) цифр числа

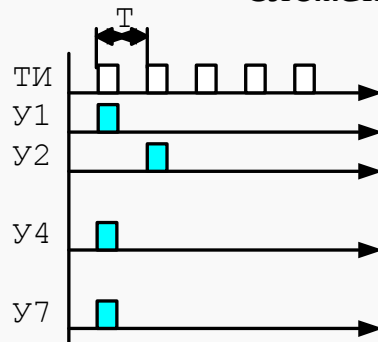


1. (y1, y8) Сдвинуть содержимое регистра X на 1 разряд (*10)
2. (y2) Записать результат в регистр X
3. (y1, y5, y7) Сложить X с цифрой с клавиатуры
4. (y2) Записать результат в X

The First Computer: Calculator (3)

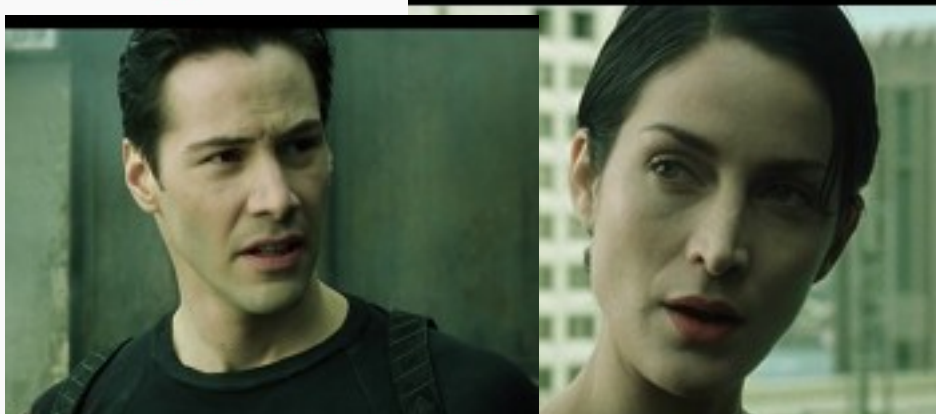


Сложение регистра X и регистра Y



1. (y1, y4, y7) Сложить содержимое регистра X и регистра Y
2. (y2) Записать результат в регистр X

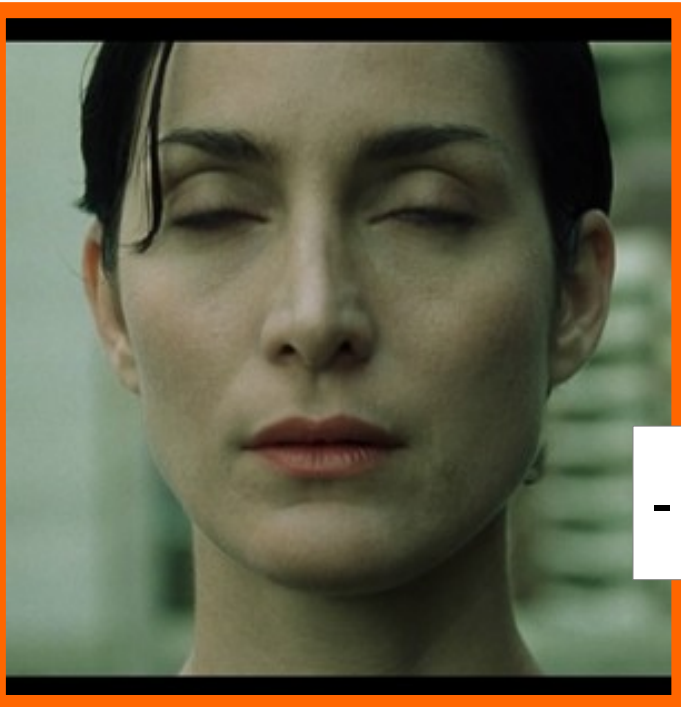
Do you know, how to drive that thing?



- Do you know, how to drive a that thing?
- Not yet.



- Tank, drive
program, please



- Let's go!



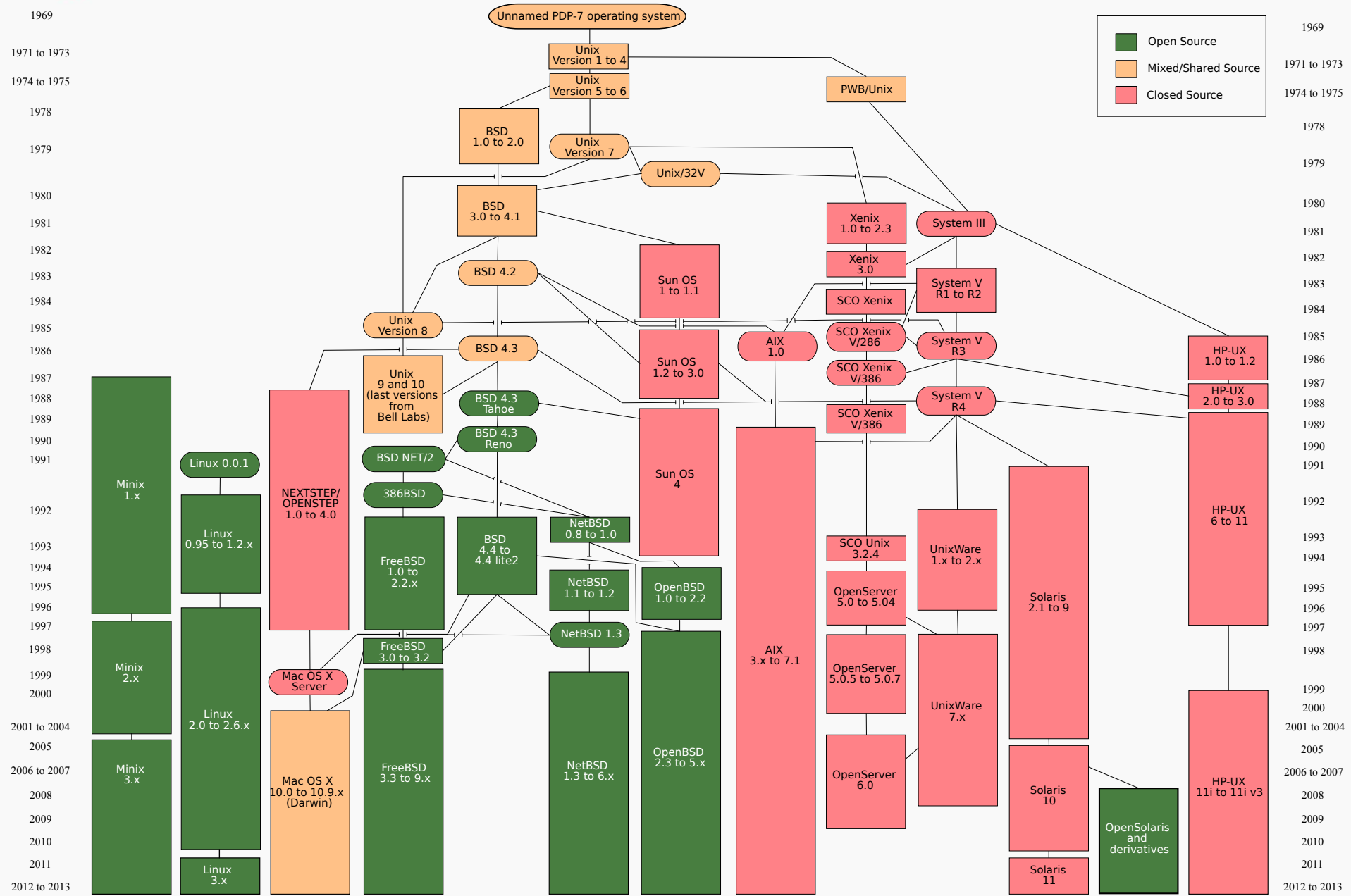
UNIX-like OS

2



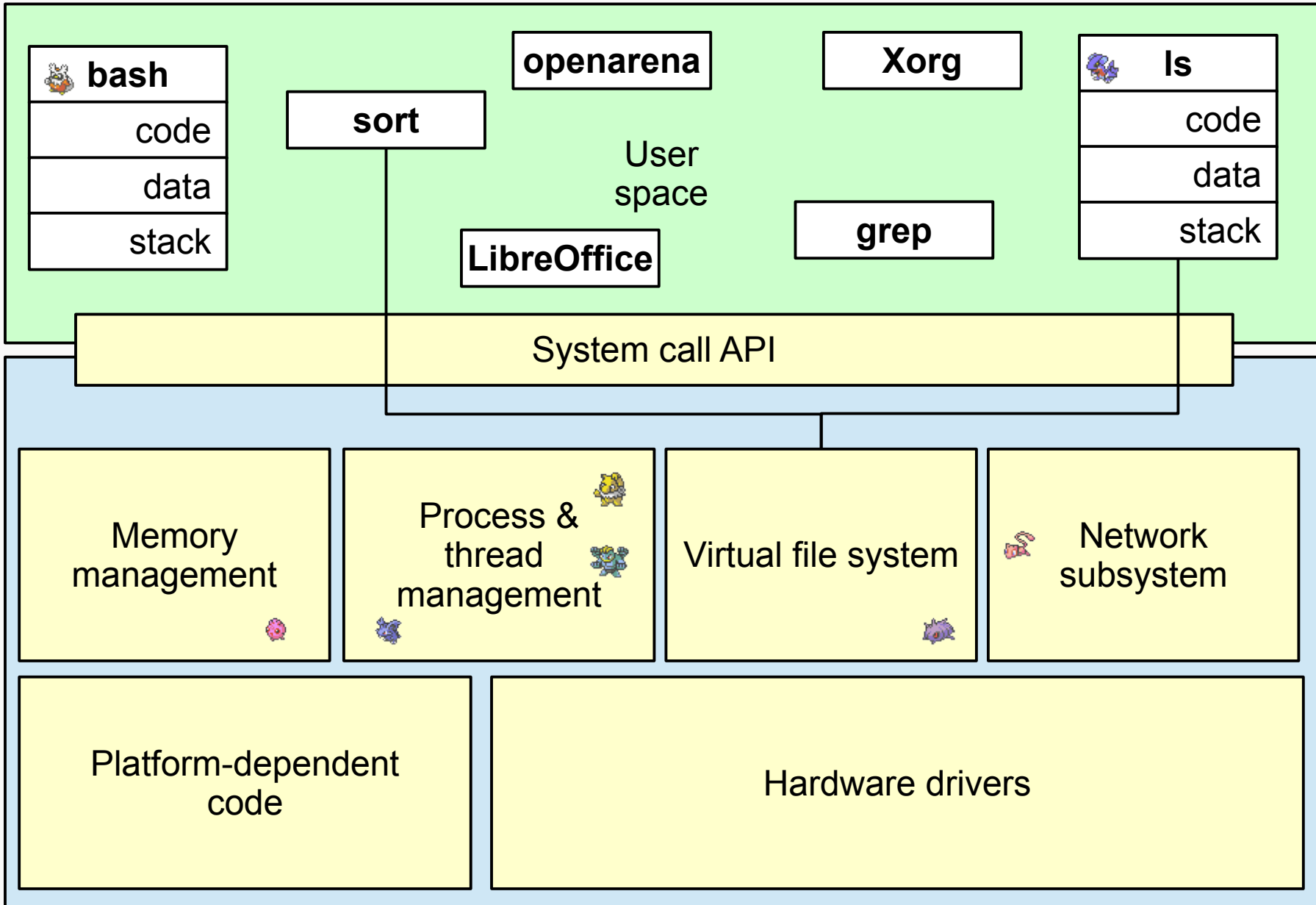


History

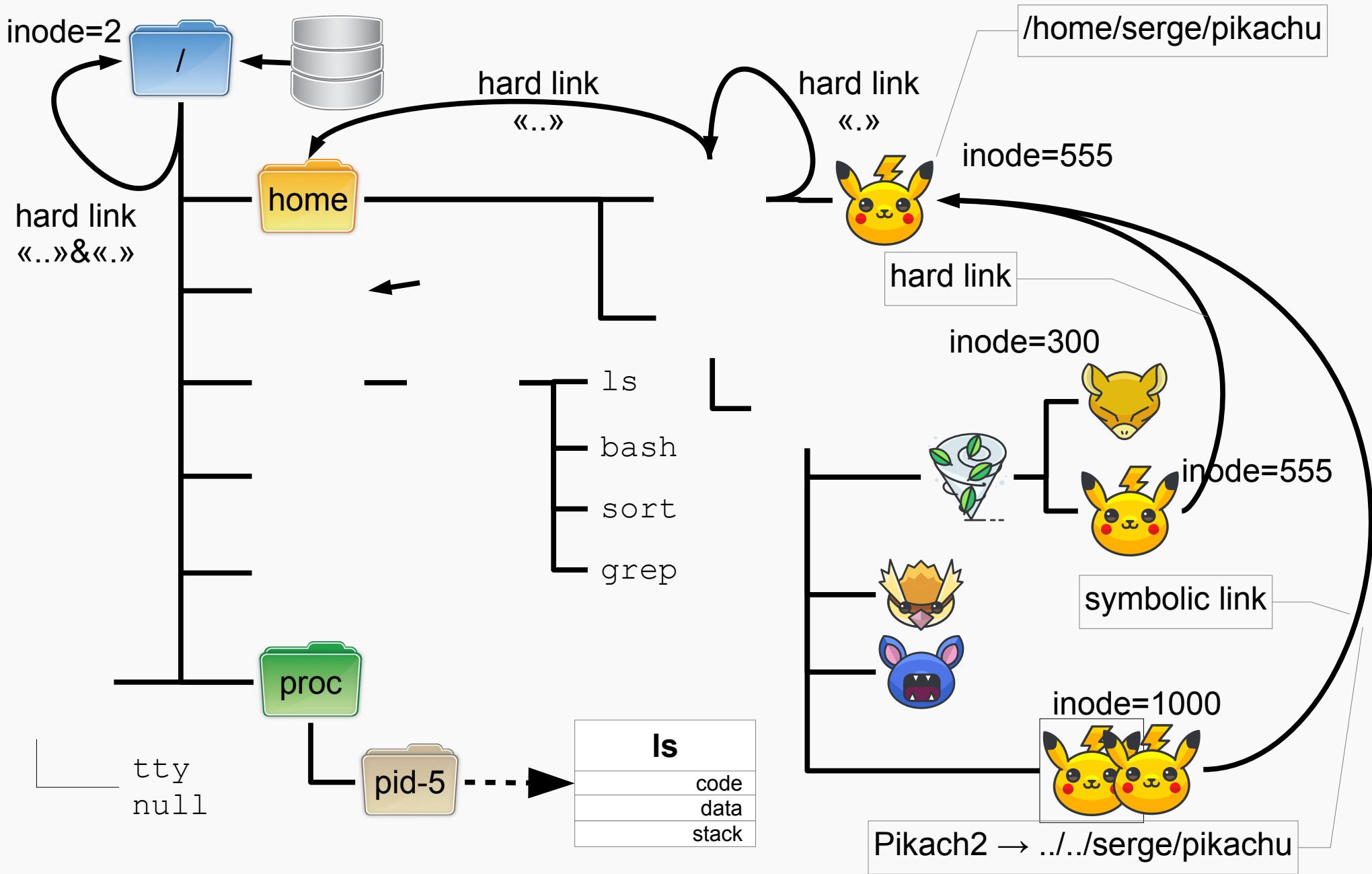


- System V
 - Solaris, AIX, HPUX
- BSD
 - Mac OS X
 - NetBSD, FreeBSD, OpenBSD
- Linux
 - RedHat, Ubuntu, SUSE
 - Fedora, Debian, OpenSUSE, ArchLinux
 - Gentoo
 - ...

*NIX Kernel



File System



File Access Rights

Hard links
count

Owner

Owner group

Name

```
s207549@helios:/export/home/studs/s207549/lab0$ ls -la
```

total 26

drwxr-xr-x	5	s207549	studs	10 дек.	8	2015	./
drwxr-xr-x	24	s207549	studs	37 дек.	8	2015	../
----rw----	1	s207549	studs	21 дек.	6	2015	Conkeldurr2
lrwxrwxrwx	1	s207549	studs	5 дек.	6	2015	Copy_50 -> Xatu9/
dr-x--x-wx	5	s207549	studs	9 дек.	8	2015	Flareon0/
drwx-wxrwx	4	s207549	studs	8 дек.	8	2015	Gengar7/
-rw-----	2	s207549	studs	37 дек.	6	2015	Hypno5
-rw-r--r--	1	s207549	studs	183 дек.	8	2015	Hypno5_21
-r--r-----	1	s207549	studs	285 дек.	6	2015	Psyduck4
dr-xr-xr-x	5	s207549	studs	8 дек.	8	2015	Xatu9/

File
type

Owner

Owner
group

Other
users

-	r	w	-	r	-	-	r	-	-
---	---	---	---	---	---	---	---	---	---

Read, Write, eXecute

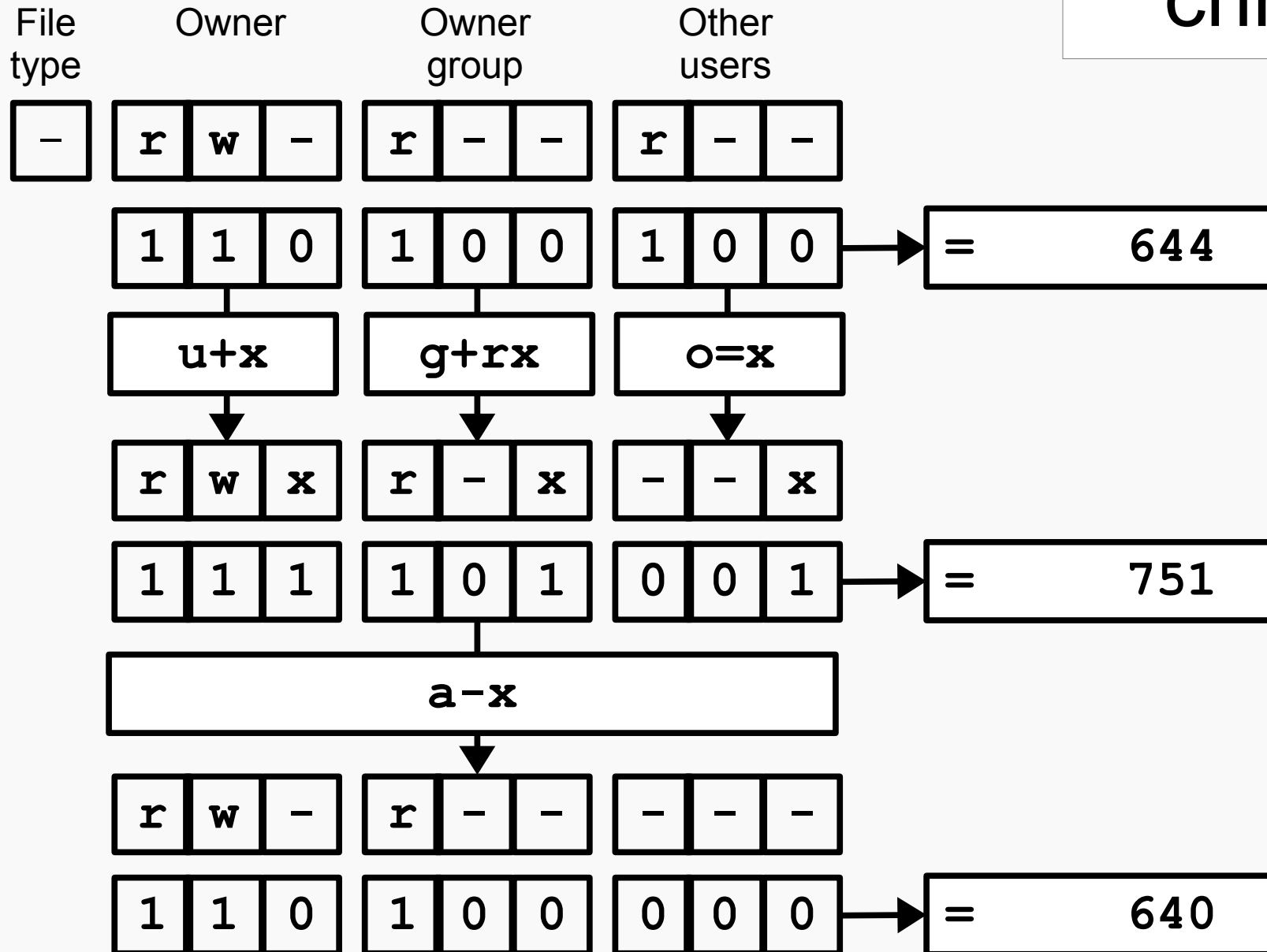
Directory, symbolic Link, file (-)

Date:

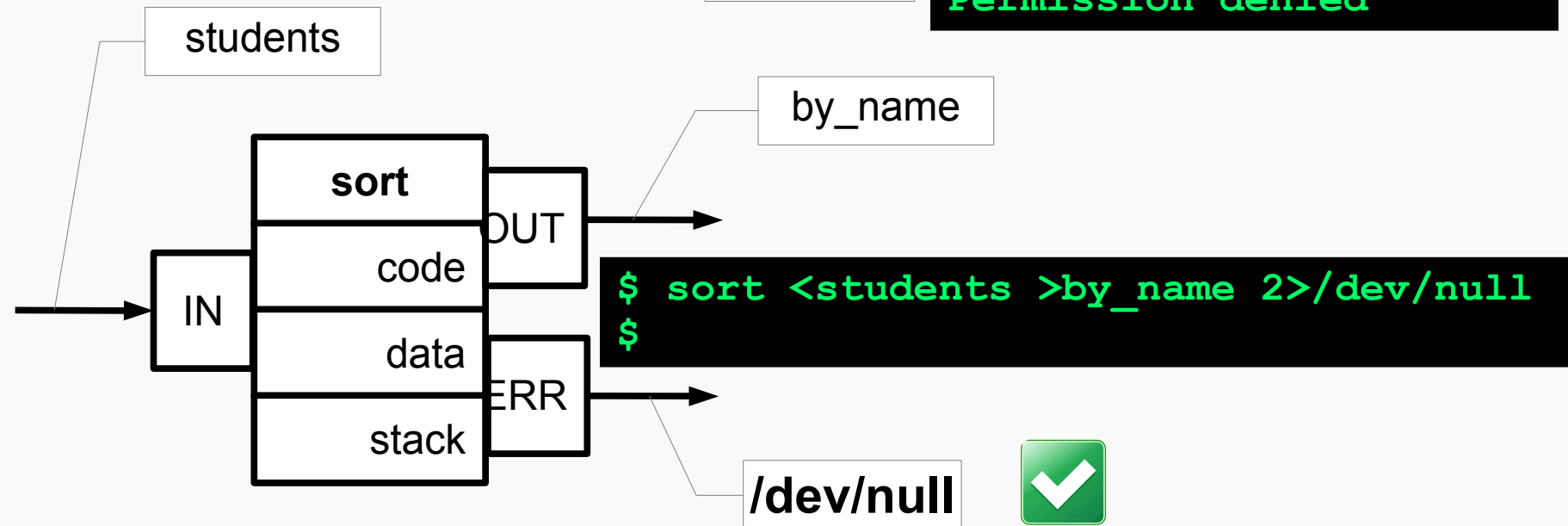
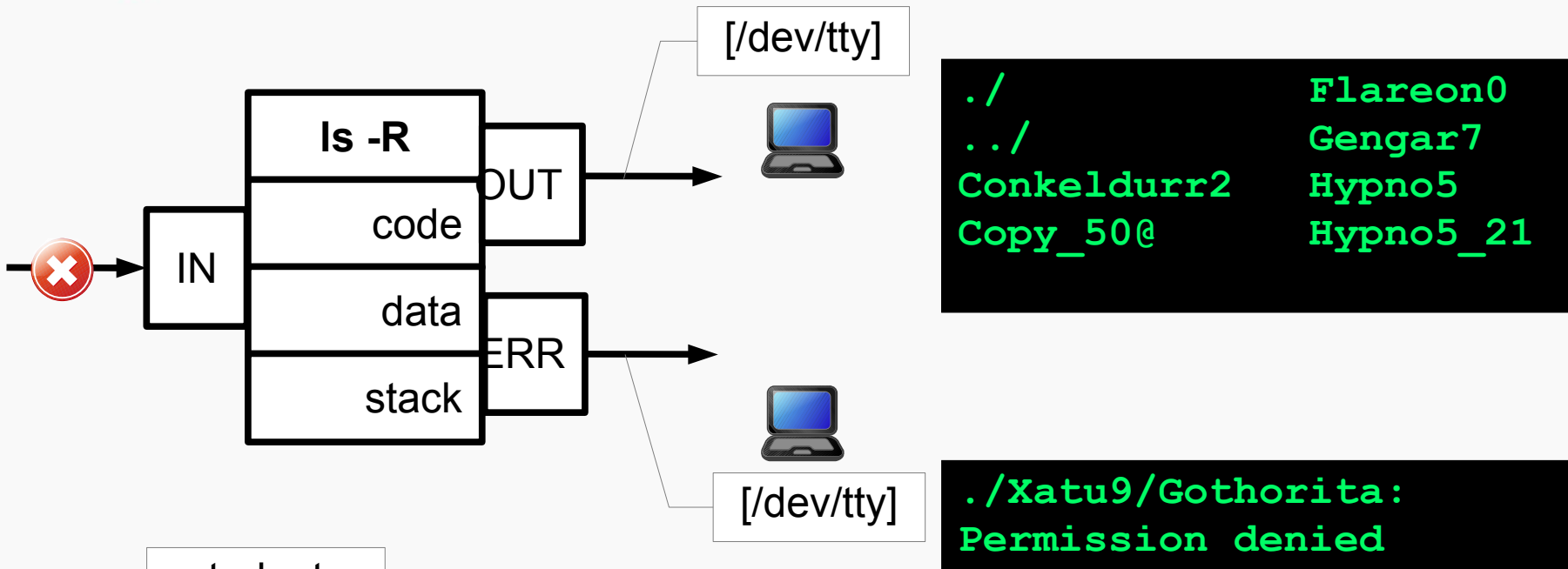
- Last modified
- Last access [-u]
- Last inode change [-c]

File Access Rights Administration

chmod

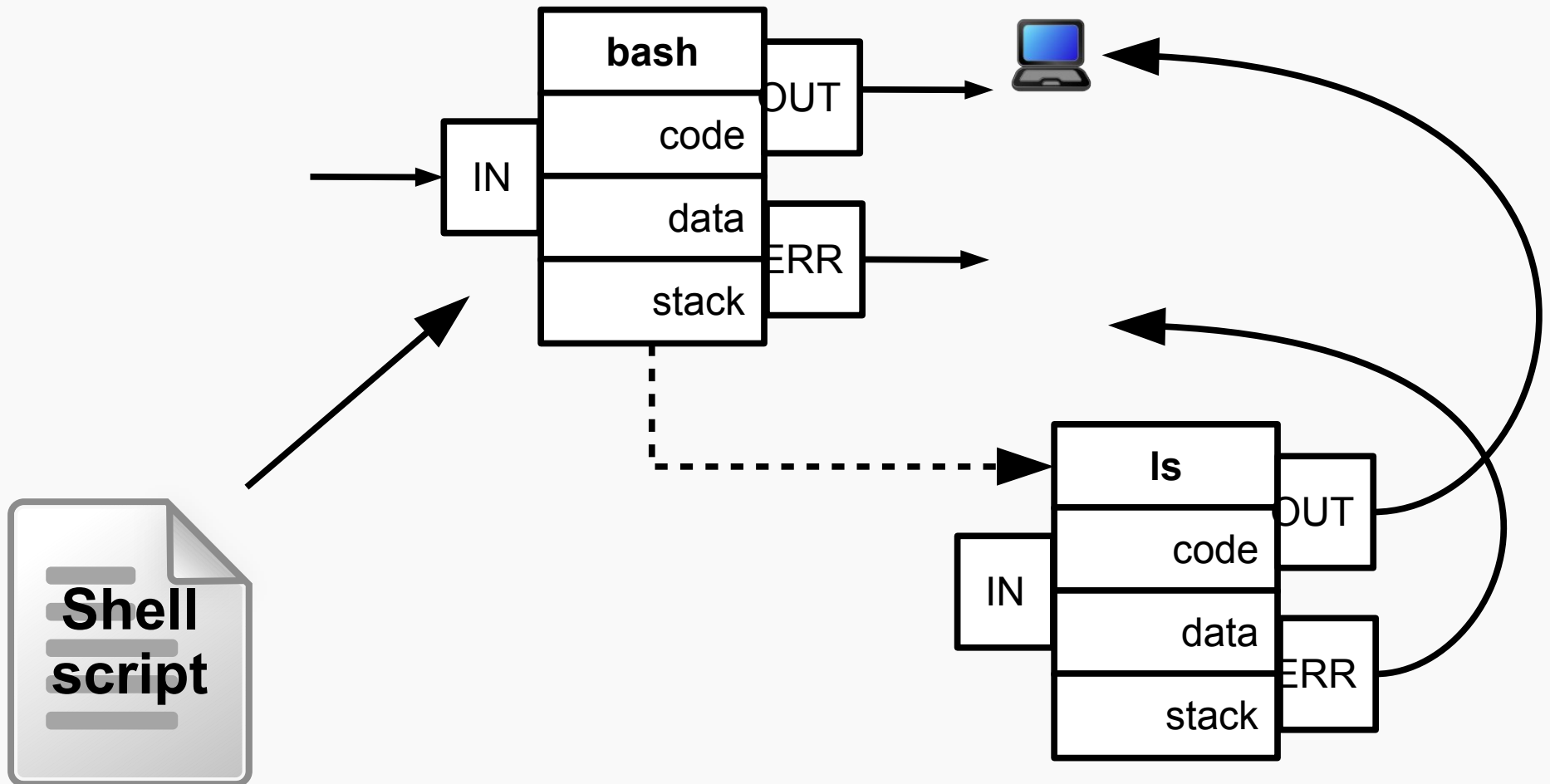


stdin(0), stdout(1) & stderr(2) streams



Shell

- sh (Bourne shell) ksh (Korn shell) csh (C shell)
bash (Bourne-again shell)



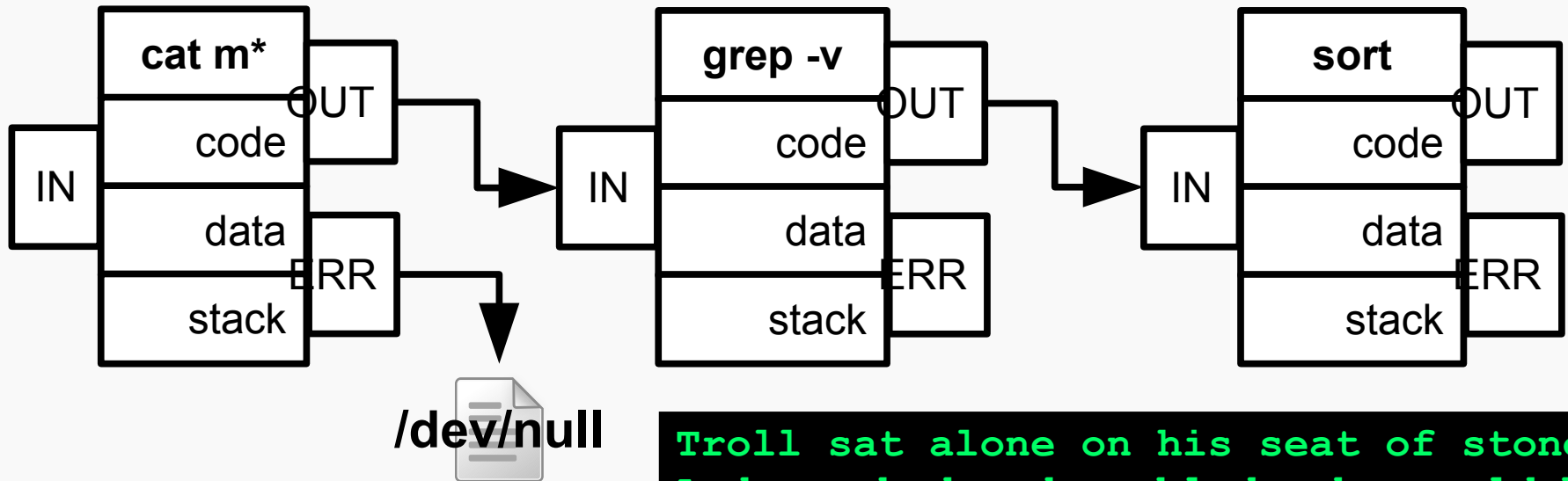


stdin(0), stdout(1) & stderr(2) streams

- `> file` — redirect stdout to `file`
- `>> file` — add stdout to `file`
- `2> file` — redirect stderr to `file`
- `2>> file` — add stderr to `file`
- `< file` — get stdin from `file`
- `<< EOF` — write all text from terminal until «`EOF`» symbol to stdin
- `ls | sort` — redirect stdout of `ls` command to stdin of `sort` command

Filters

```
$ cat m* 2>/dev/null | grep -v "^V" | sort
```



```
Troll sat alone on his seat of stone
And munched and mumbled a bare old bone
For many a year he had gnawed it near
For meat was hard to come by
```

Vasya:

For meat was hard to come by

Viktor:

And munched and mumbled a bare old bone

Vasya:

Troll sat alone on his seat of stone

Veronika:

For many a year he had gnawed it near

Regular Expressions

messages

Vasya:
Troll sat alone on his seat of stone
Viktor:
And munched and mumbled a bare old bone
Veronika:
For many a year he had gnawed it near
Vasya:
For meat was hard to come by

- Any letter corresponds to itself
- ^ - line start
- \$ - end of line
- . - any single letter

```
$ grep meat messages  
For meat was hard to come by  
$ grep "^V" messages  
Vasya:  
Viktor:  
Veronika:  
Vasya:  
$ grep "a.e" messages
```

Basic Commands

Command	Description
mkdir	mkdir [-m mode] [-p] dir...
echo	echo [string]...
cat	cat [-n] [file...] [-]
touch	touch [-am]... file...
ls	ls [options] [file/dir]...
pwd	pwd
cd	cd [argument]
more	more [file...]
cp	cp [options] SOURCE ... DEST
rm	rm [options] [file/dir]
rmdir	rmdir [dir]
mv	mv [-fi] SOURCE ... DEST
head	head [-num] [file...]
tail	tail [-/+num] [-bcl] [file...]
sort	sort [-unr] [-k num] [file...]
grep	grep [-v] regexp [file...]
wc	wc [-c -m] [-lw] [file...]

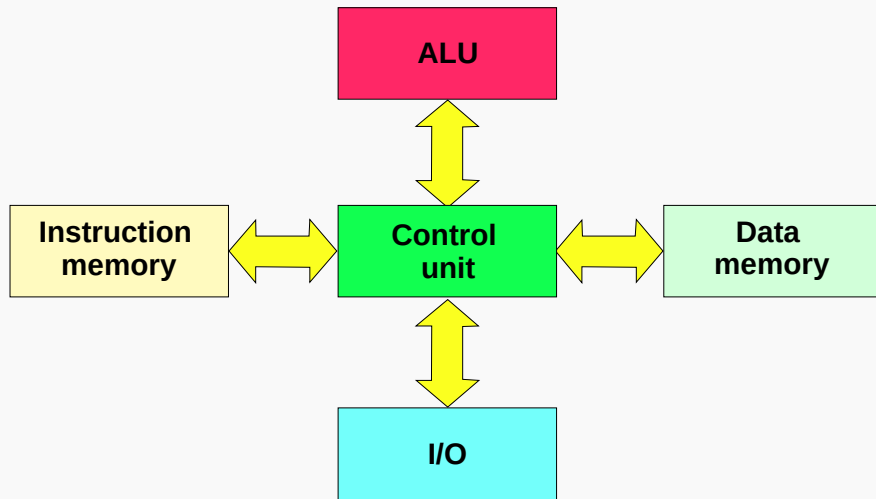
Acquaintance with the Basic Computer

3

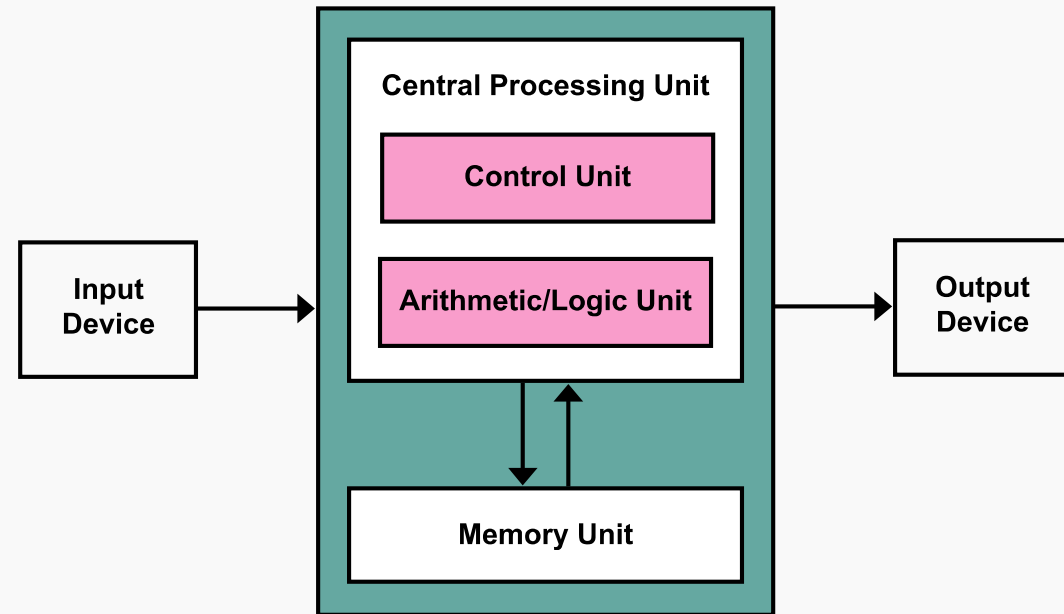


Computer Architecture

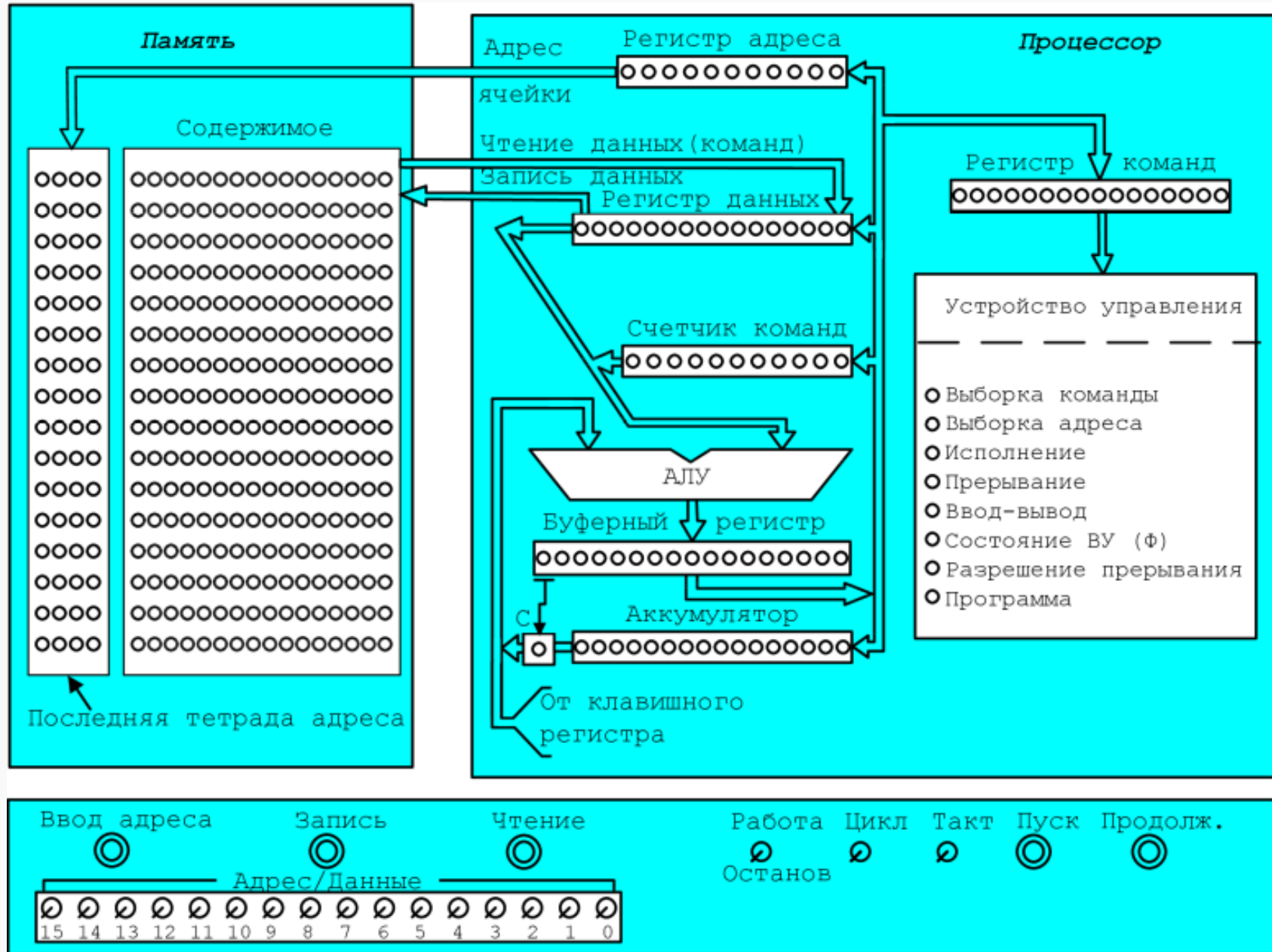
Harvard Architecture



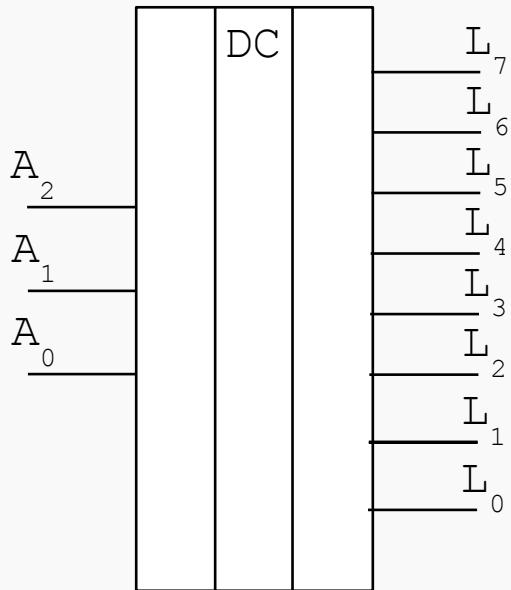
Von Neumann Architecture



Basic Computer (BComp)



Decoder

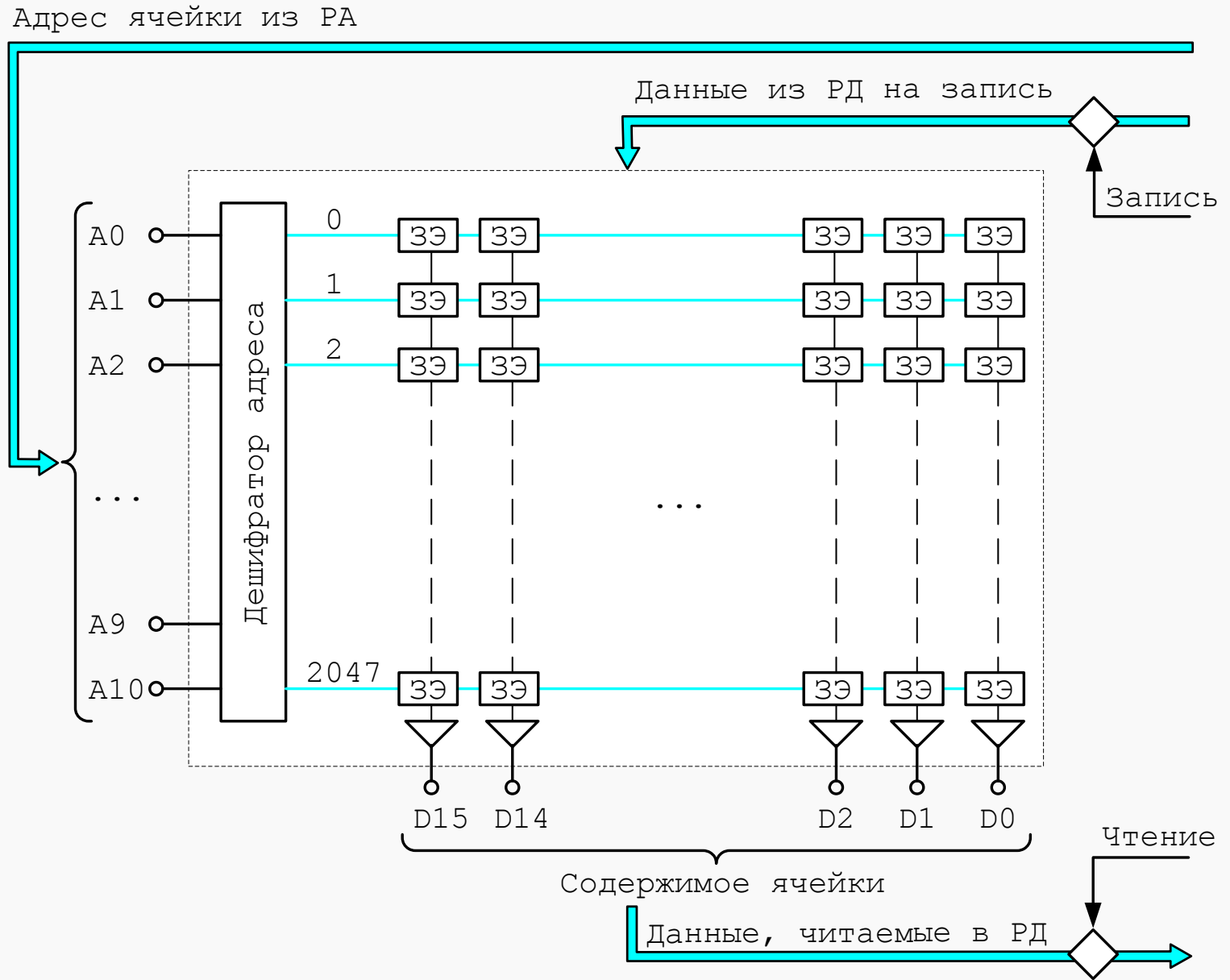


Адрес			String							
A_2	A_1	A_0	L_7	L_6	L_5	L_4	L_3	L_2	L_1	L_0
0	0	0	0	0	0	0	0	0	0	1
0	0	1	0	0	0	0	0	0	1	0
0	1	0	0	0	0	0	0	1	0	0
0	1	1	0	0	0	0	1	0	0	0
1	0	0	0	0	0	1	0	0	0	0
1	0	1	0	0	1	0	0	0	0	0
1	1	0	0	1	0	0	0	0	0	0
1	1	1	1	0	0	0	0	0	0	0

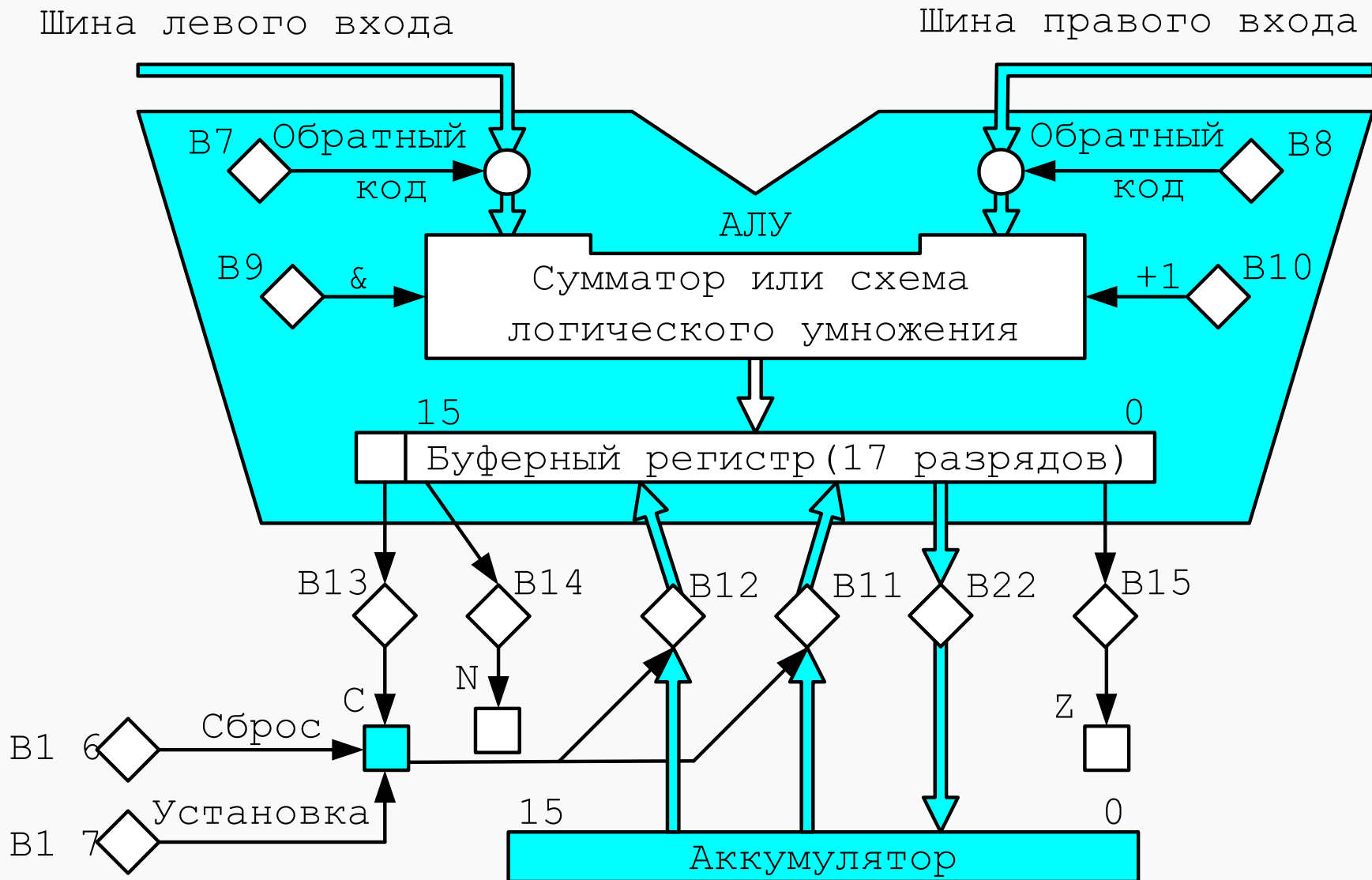
VComp Addressable Memory



- A total of 2048 16-bit cells
- Cells 008-00F are index cells
- Cells 000 & 001 are used for interrupts



ALU



Control Unit

Instruction cycle

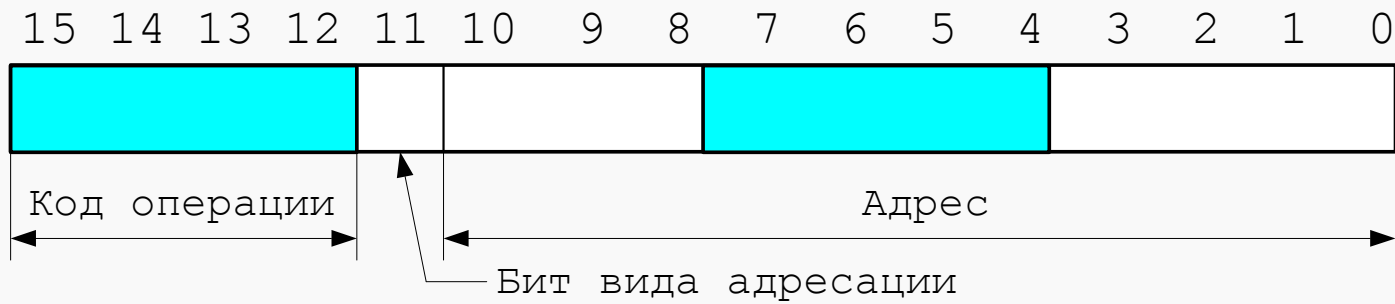
- ▶ 1. Instruction fetch cycle
- ▶ 2. Address fetch cycle
- ▶ 3. Execution cycle
- ▶ 4. Interruption cycle

Operational console cycles

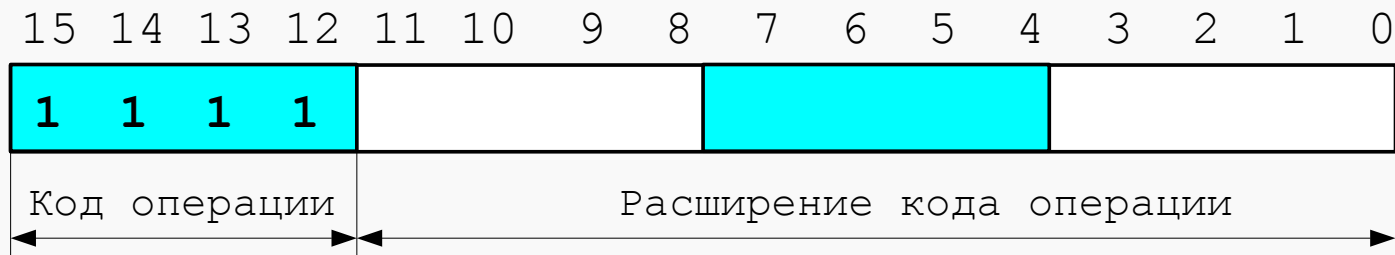
- ▶ Address input
- ▶ Read
- ▶ Write
- ▶ Start

Instruction Formats

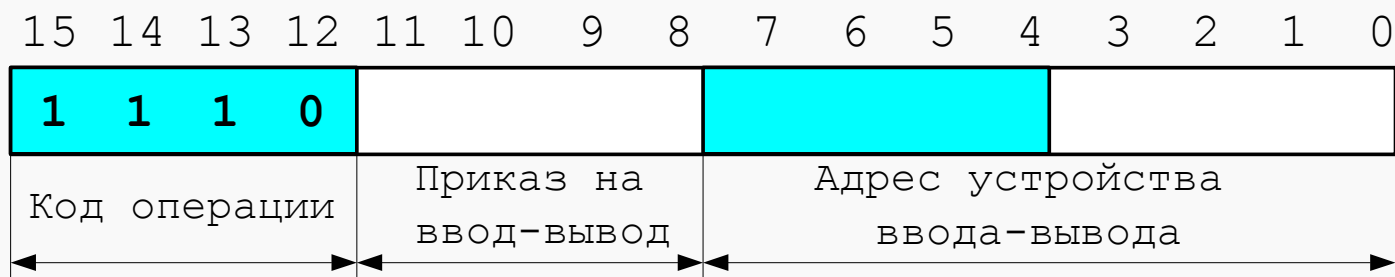
Адресные команды



Безадресные команды



Команды ввода-вывода



Address Instructions

Name	Mnemonic name	Code	Description
Conjunction	AND M	1XXX	$(M) \& (A) \rightarrow A$
Move	MOV M	3XXX	$(A) \rightarrow M$
Addition	ADD M	4XXX	$(M) + (A) \rightarrow A$
Addition with carry	ADC M	5XXX	$(M) + (A) + (C) \rightarrow A$
Substraction	SUB M	6XXX	$(A) - (M) \rightarrow A$
Branch if carry is set	BCS M	8XXX	If $(C) = 1$, then $M \rightarrow IP$
Branch if plus	BPL M	9XXX	If $(N) = 0$, then $M \rightarrow IP$
Branch if minus	BMI M	AXXX	If $(N) = 1$, then $M \rightarrow IP$
Branch if zero	BEQ M	BXXX	If $(Z) = 1$, if $M \rightarrow IP$
Unconditional branch	BR M	CXXX	$M \rightarrow IP$
Increment and skip	ISZ M	0XXX	$M + 1 \rightarrow M$, if $(M) \geq 0$, then $(IP) + 1 \rightarrow IP$
Subprogram call	JSR M	2XXX	$(IP) \rightarrow M$, $M + 1 \rightarrow IP$

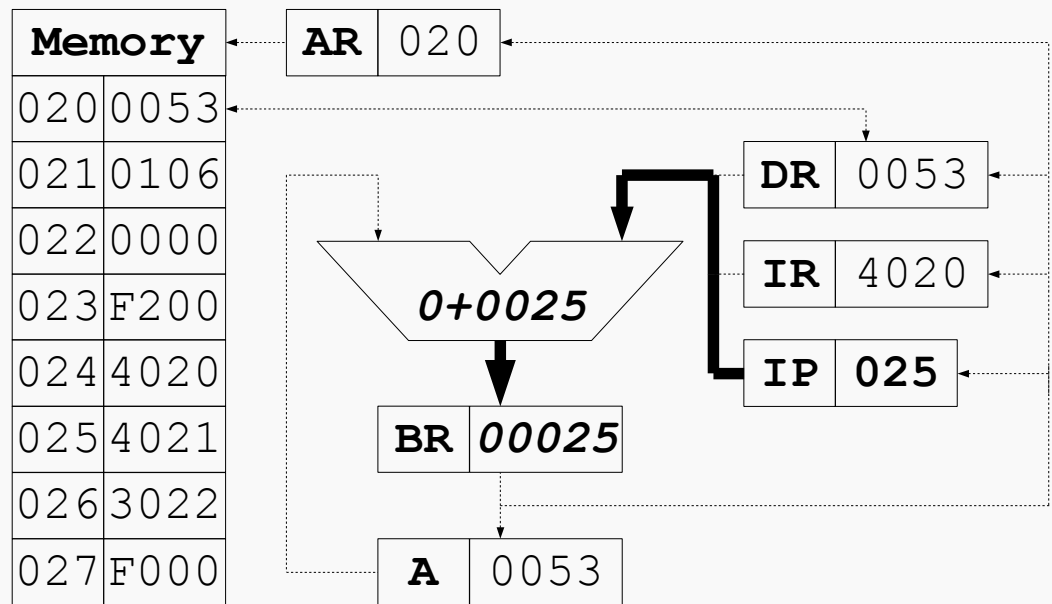
No-address Instructions

Name	Mnemonic name	Code	Description
Clear accumulator register	CLA	F200	$0 \rightarrow A$
Clear carry register	CLC	F300	$0 \rightarrow C$
Invert accumulator register	CMA	F400	$(!A) \rightarrow A$
Invert carry register	CMC	F500	$(!C) \rightarrow C$
Left cyclic shift by 1 bit	ROL	F600	A & C content moves left, $A(15) \rightarrow C$, $C \rightarrow A(0)$
Right cyclic shift by 1 bit	ROR	F700	A & C content moves right, $A(0) \rightarrow C$, $C \rightarrow A(15)$
Increment accumulator register	INC	F800	$(A) + 1 \rightarrow A$
Decrement accumulator register	DEC	F900	$(A) - 1 \rightarrow A$
Halt	HLT	F000	
No operation	NOP	F100	
Enable interruptions	EI	FA00	
Disable interruptions	DI	FB00	

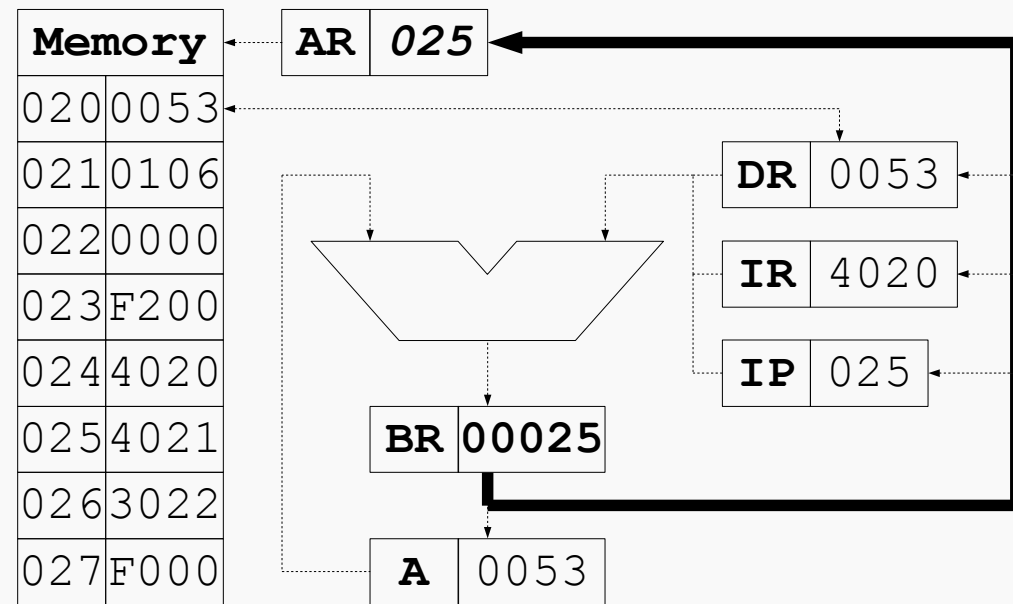
I/O Instructions

Name	Mnemonic name	Code	Description
Clear flag	CLF B	E0XX	0 → device flag
Check flag	TSF B	E1XX	If (device flag B) = 1, then (IP) + 1 → IP
Input	IN B	E2XX	(B) → A
Output	OUT B	E3XX	(A) → B

Instruction Fetch Cycle: ADD 21

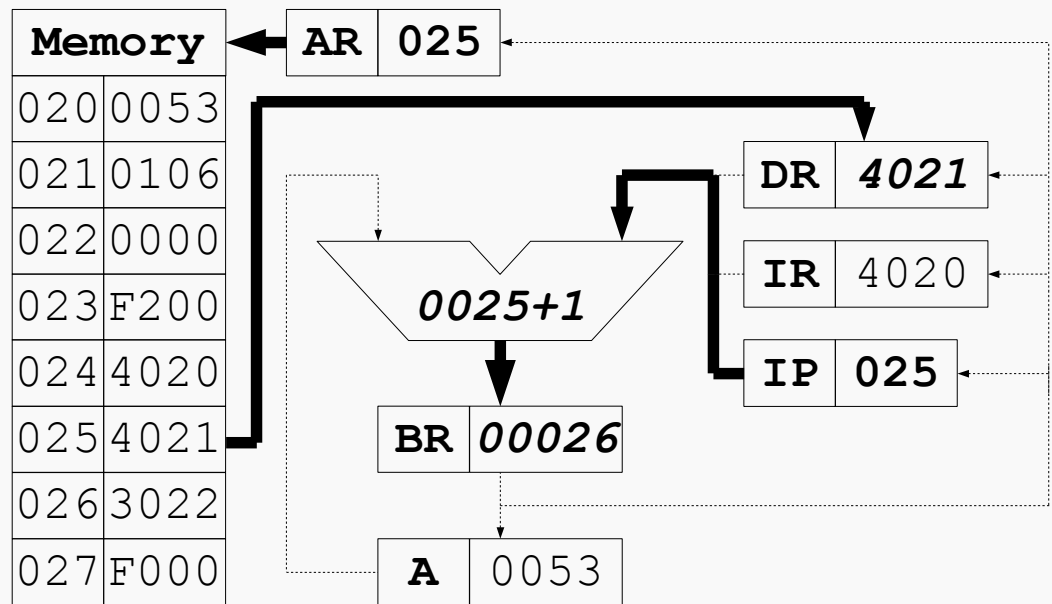


1) Transfer IP content to BR through ALU.

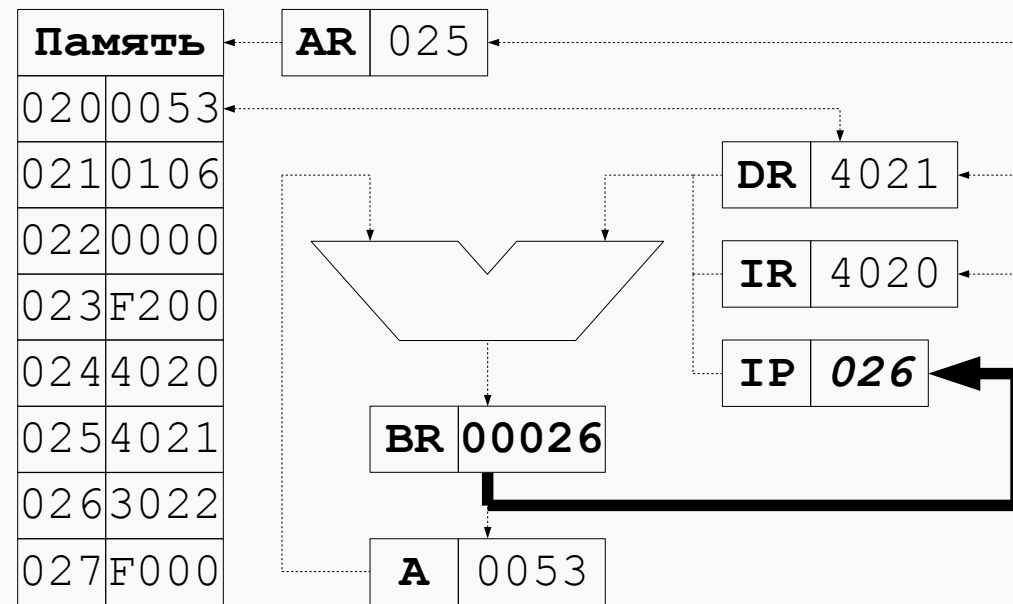


2) Transfer 11 lower bits of BR content to AR.

Instruction Fetch Cycle: ADD 21

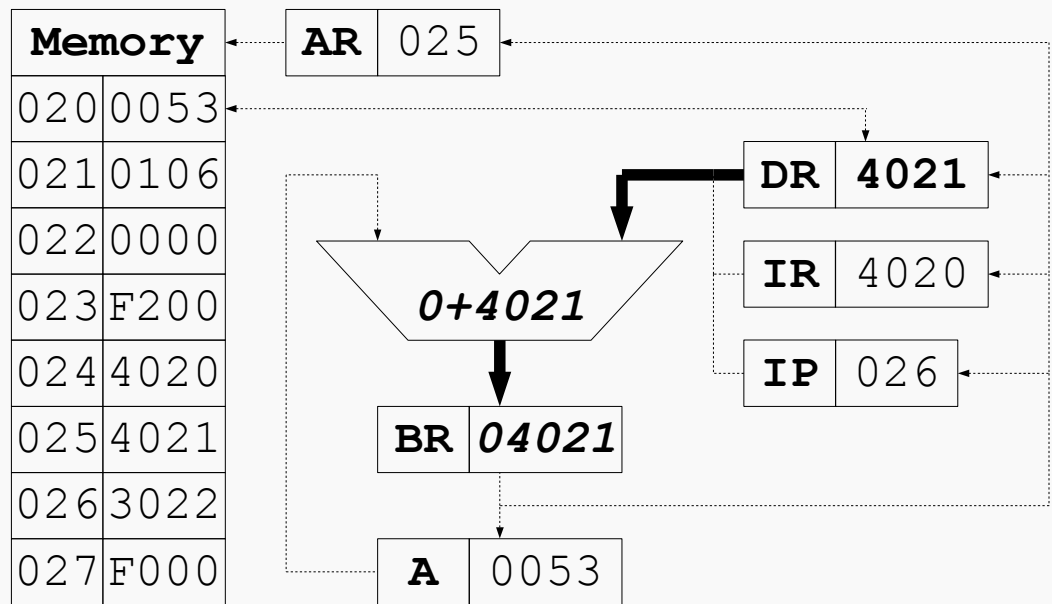


3) Fetch instruction code by address from AR (025) and save it to DR. At the same time increment IP content and save it to BR.

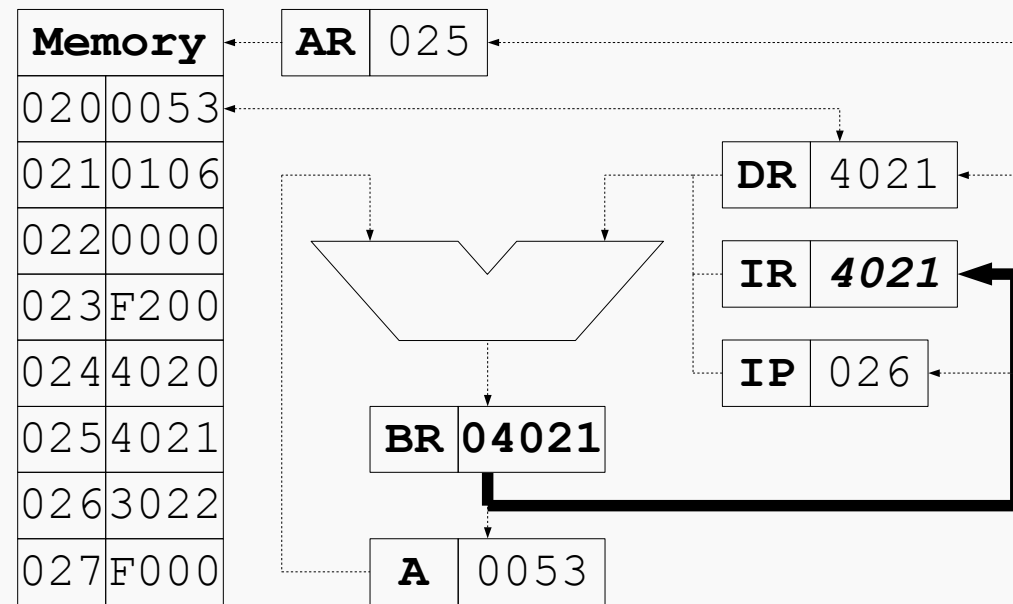


4) Transfer 11 lower bits of BR content to IP.

Instruction Fetch Cycle: ADD 21

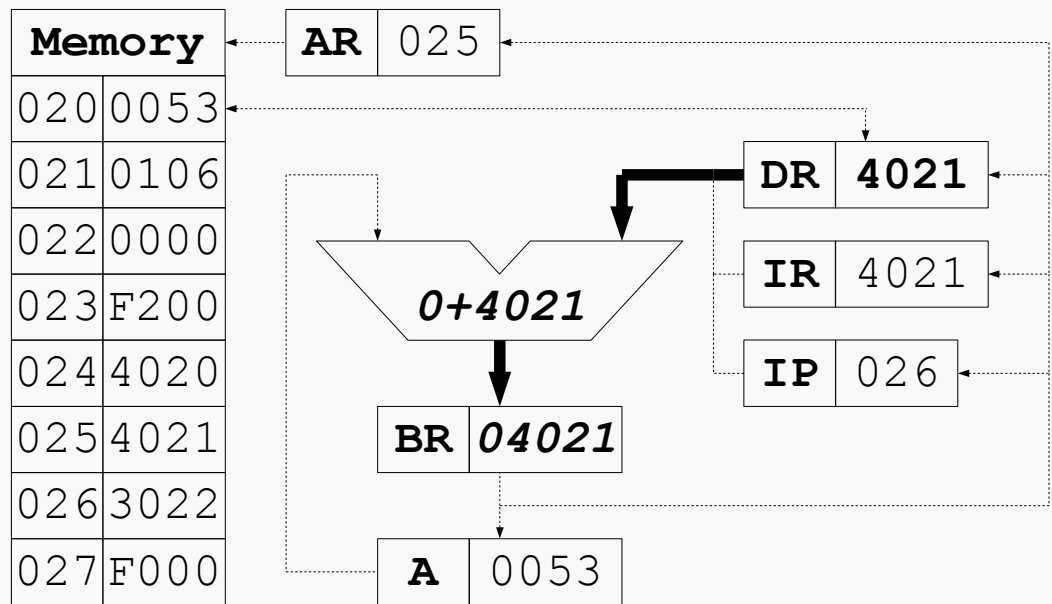


5) Save DR content to BR through the right input of the ALU.

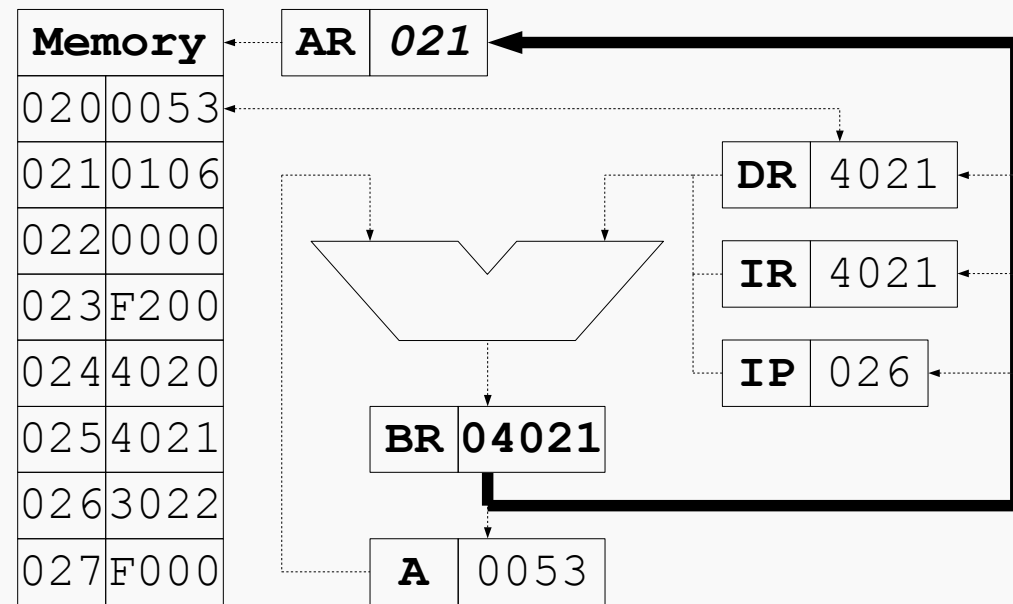


6) Transfer 16 lower bits of the BR content to IR.

Execution Cycle of the Direct Addressing Instruction: ADD 21

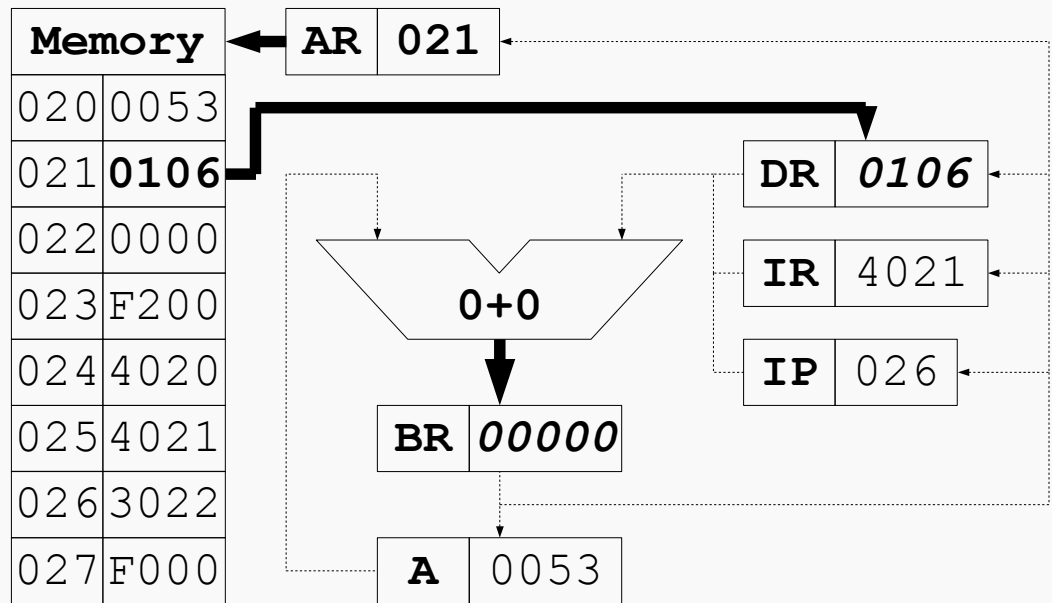


1) Save DR content to BR through the right input of the ALU.

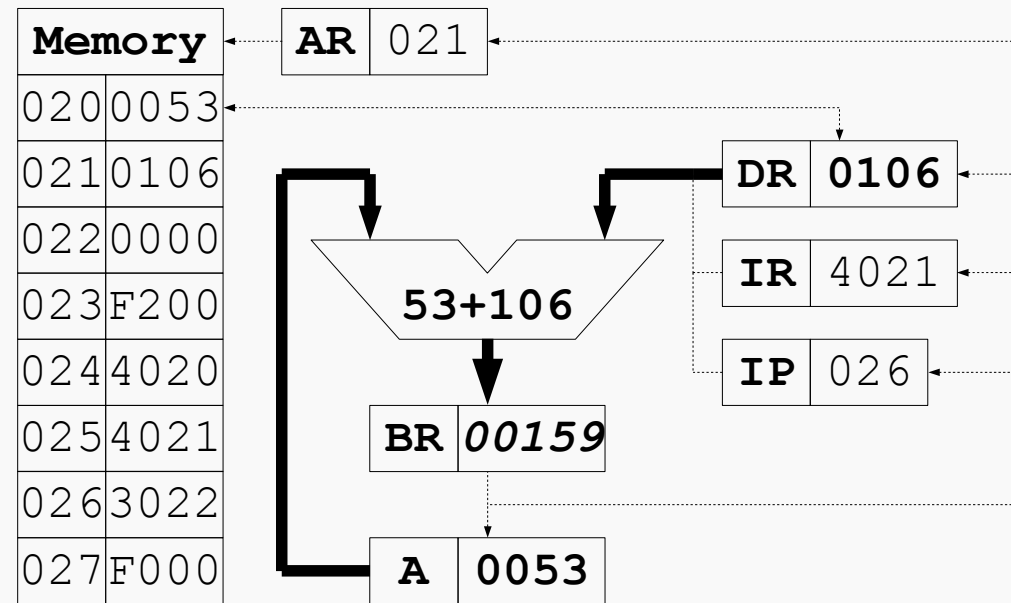


2) Transfer 11 lower bits of the BR content to AR.

Execution Cycle of the Direct Addressing Instruction: ADD 21

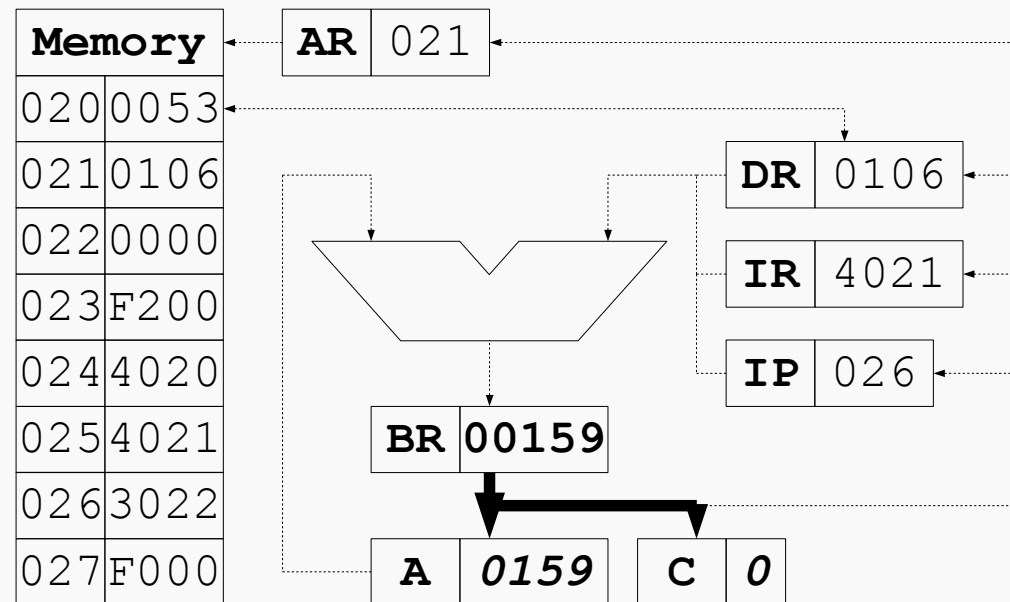


3) Save content of the memory cell 021 (addition operand) to DR. **Reset BR!**



4) Perform addition of the BR content (on the right input of the ALU) with the A content (on the left input of the ALU) and save it to BR.

Execution Cycle of the Direct Addressing Instruction: ADD 21



5) Save lower 16 bits (from 0 to 15th) of the BR content to A, save 16th bit of the BR content to C (carry) bit of the SR. Reset N & Z result attributes.

4



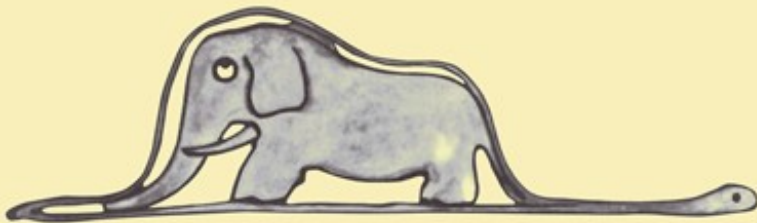


Representation Area

3021 — what does it mean?

Representation Area

Le Petit Prince™



Le BComp

1 2 3 2 1
0011 0000 0010 0001

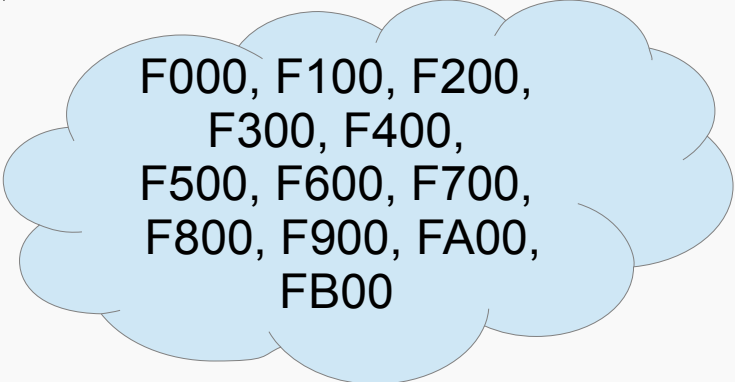
MOV 2 1
0011 0000 0010 0001
code operand address
addressing mode bit

Allowed Values

- Are determined by the representation area.

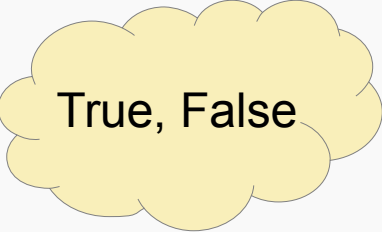
- Examples:

- No-address instructions of the BComp



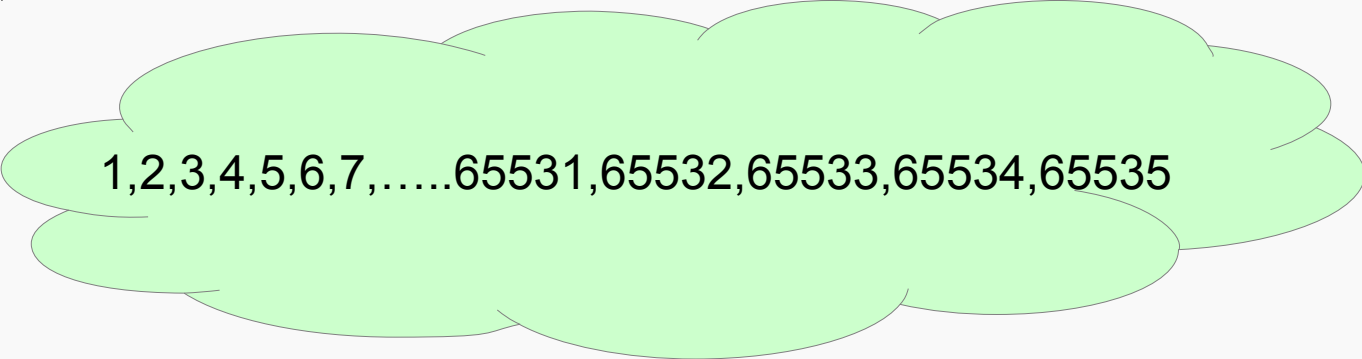
F000, F100, F200,
F300, F400,
F500, F600, F700,
F800, F900, FA00,
FB00

- Logical values

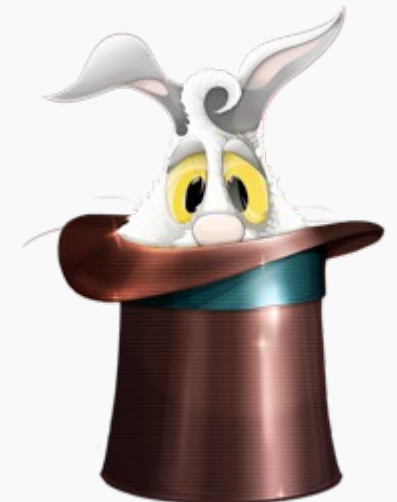


True, False

- Integers < 65535



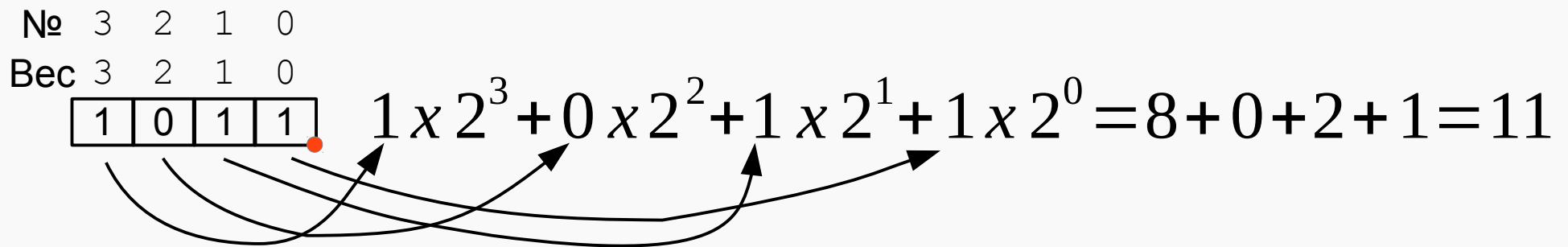
1,2,3,4,5,6,7,.....65531,65532,65533,65534,65535



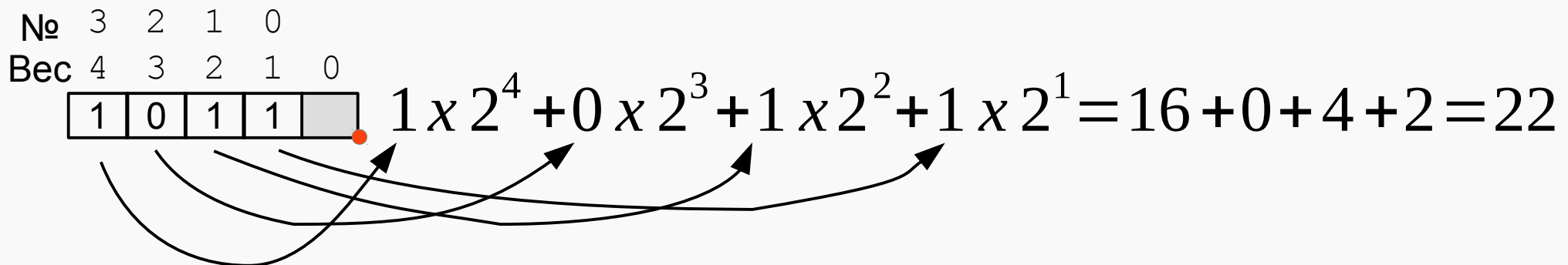
Numbers Representation: Fixed Point

- Integers: **binary** radix point is on the fixed position after bit 0, all bits have a positive place value:

1) Bit number equals its place value

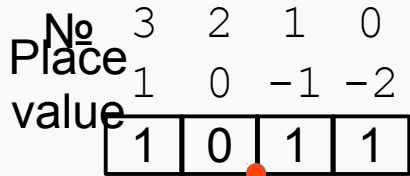


2) Bit place value = bit number + 1



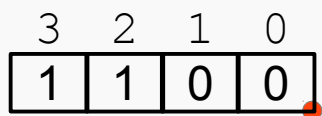
Numbers Representation: Fixed Point

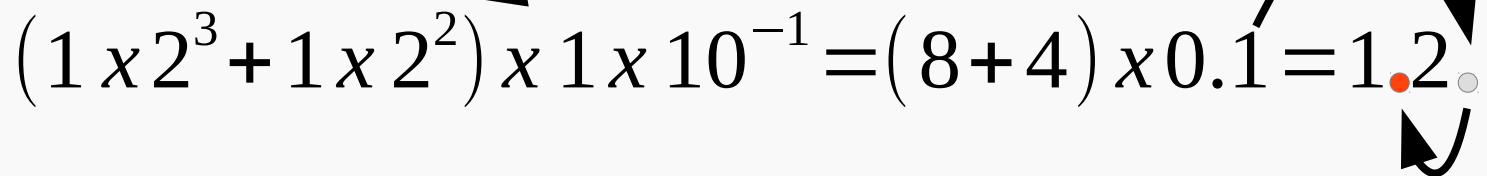
- Real numbers: **binary** radix point is on the fixed position after bit 2, lower two bits have a negative place values:



$$1 \times 2^1 + 0 \times 2^0 + 1 \times 2^{-1} + 1 \times 2^{-2} = 2 + 0 + 0.5 + 0.25 = 2.75$$

- Real numbers: moving the **decimal** radix point causes the *scale* change:



$$(1 \times 2^3 + 1 \times 2^2) \times 1 \times 10^{-1} = (8 + 4) \times 0.1 = 1.2$$


Unsigned Integers Representation

- Number of bits in the bit grid of the computer defines its allowable range:

- Minimum 4-bit unsigned number:

3	2	1	0
0	0	0	0

$$0 \times 2^3 + 0 \times 2^2 + 0 \times 2^1 + 0 \times 2^0 = 0$$

- Maximum 4-bit unsigned number:

3	2	1	0
1	1	1	1

$$1 \times 2^3 + 1 \times 2^2 + 1 \times 2^1 + 1 \times 2^0 = 15 = 2^4 - 1$$

- Range:

$$0 \leq X \leq 2^4 - 1$$

- Range for 16-bit unsigned numbers:

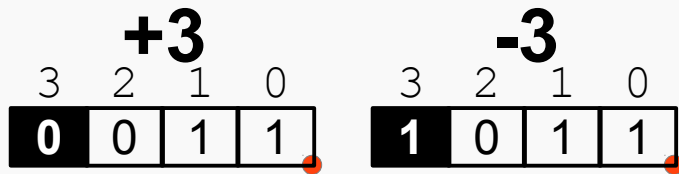
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1

$$0 \leq X \leq 2^{16} - 1 = 65535$$

Signed Integers Representation

- One bit specifies the sign of the number.
For example, «0» means «+» and «1» means «-».

- Ones' complement:



$$-7 = -(2^3 - 1) \leq X \leq 2^3 - 1 = 7$$

Double zero!

- Two's complement:

$$M = b^n - K$$

M — complement of the K (10-3=7) !

b — numeral system's base

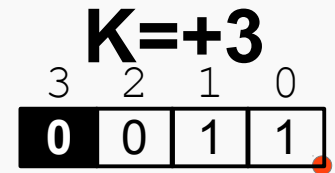
n — number of bits



Signed Integers: Two's Complement

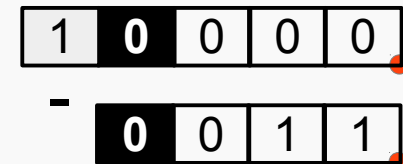
$$M = b^n - K = \underline{\underline{((b^n - 1) - K) + 1}}$$

One's compl. for 5-digit numbers	Two's complement		
	5-digit decimal numbers	4-digit hex numbers	16-bit binary numbers
-50000	50000		
-49999	50001		
-32768	67232	8000	1000 0000 0000 0000
-32767	67233	8001	1000 0000 0000 0001
-2	99998	FFFE	1111 1111 1111 1110
-1	99999	FFFF	1111 1111 1111 1111
0	00000	0000	0000 0000 0000 0000
1	00001	0001	0000 0000 0000 0001
32767	32767	7FFF	0111 1111 1111 1111
49999	49999		



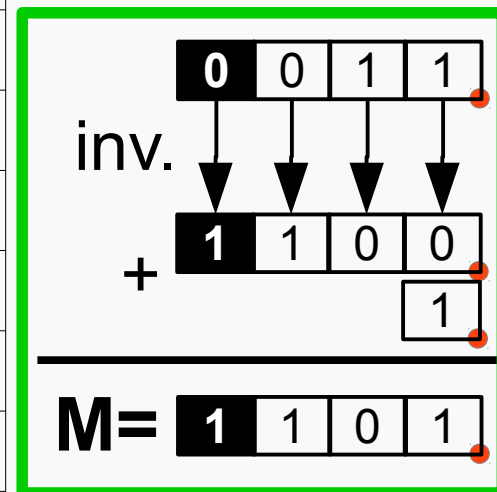
$$M = b^n - K$$

$$2^4 - 3 = 13$$

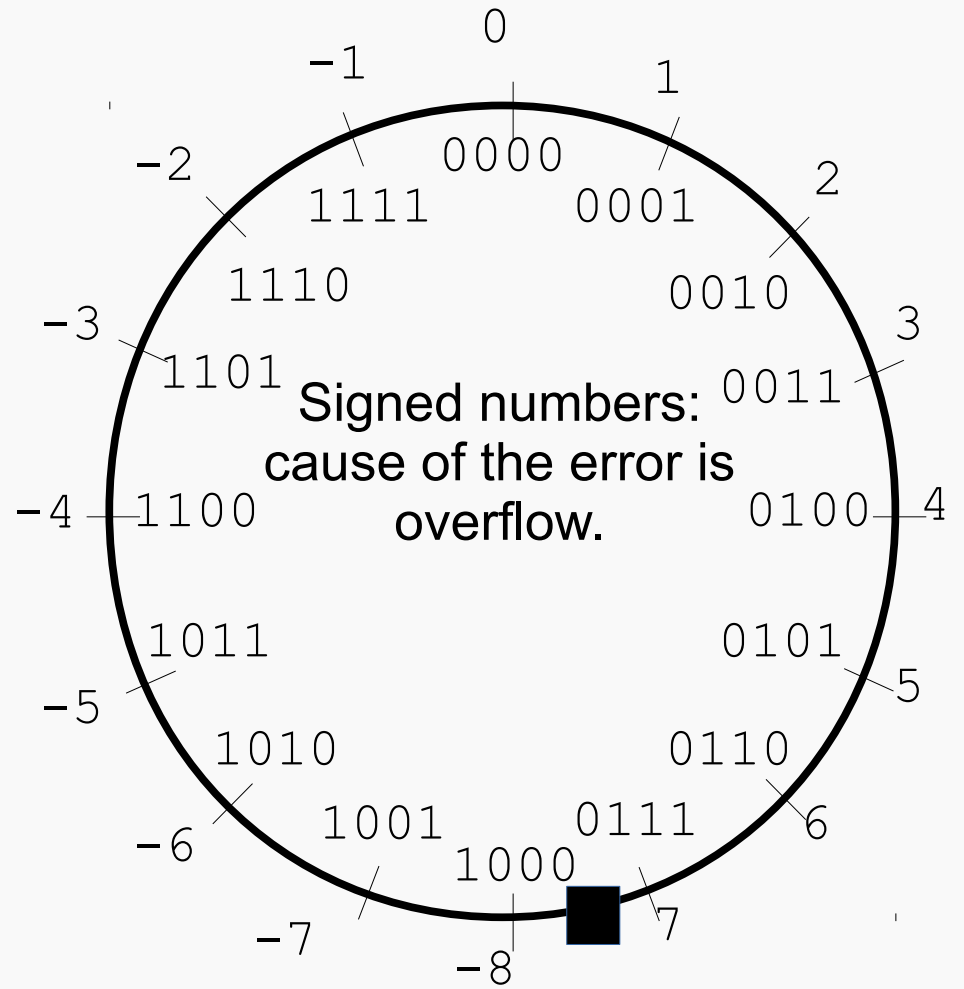
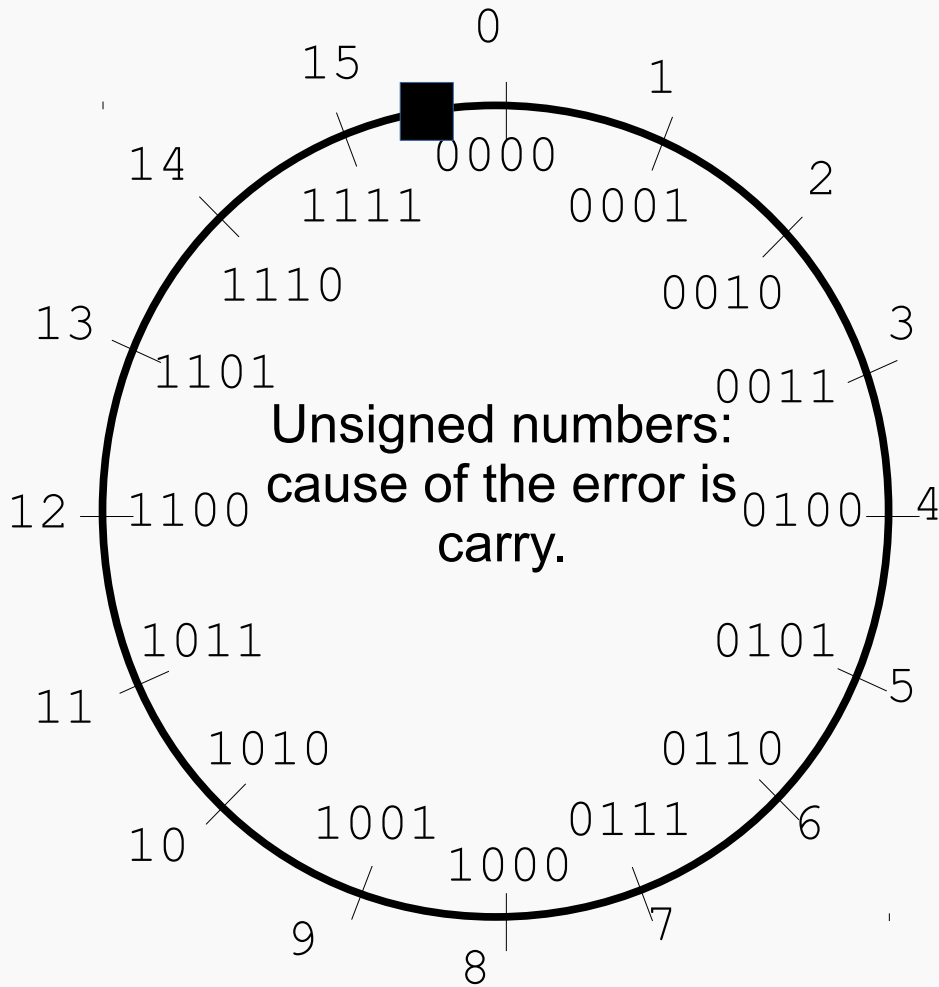


M=

1	1	0	1
---	---	---	---



Carry & Overflow





Numbers Representation in BComp

Bit grid representation	Unsigned numbers	Signed numbers
0000 0000 0000 0000	0	0
0000 0000 0000 0001	1	1
...		
0111 1111 1111 1110	32766	32766
0111 1111 1111 1111	32767	32767
1000 0000 0000 0000	32768	-32768
1000 0000 0000 0001	32769	-32767
1111 1111 1111 1110	65534	-2
1111 1111 1111 1111	65535	-1

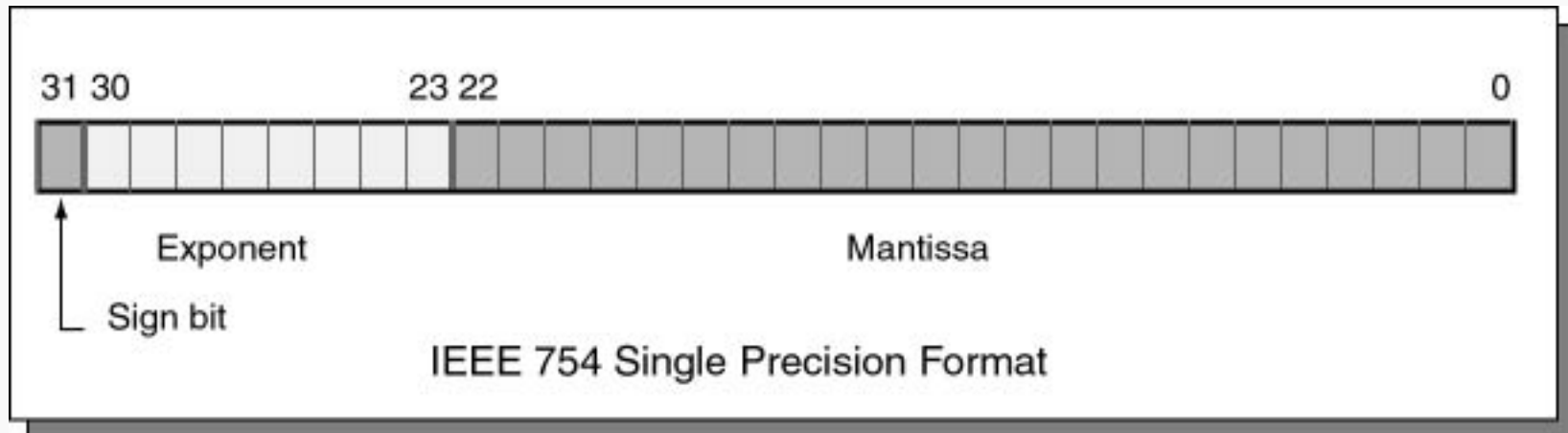
Allowable range:

↓
 $0 \leq X \leq 2^{16} - 1$

↓
 $-2^{15} \leq X \leq 2^{15} - 1$

Floating Point Numbers Representation

FP numbers doesn't exist in BComp!



$$X = (-1)^{(sign)} \times (1 + Mantissa) \times 2^{(exponent - 127)}$$

Logical Values Representation

- 1 = true, 0 = false
- 16-bit number contains 16 logical values

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1

$$X_i \in \{0,1\} \quad \text{for } 0 \leq i \leq 15$$

BComp Workshop Hints

- $R=(X\&Y)+Z$

Representation Area:

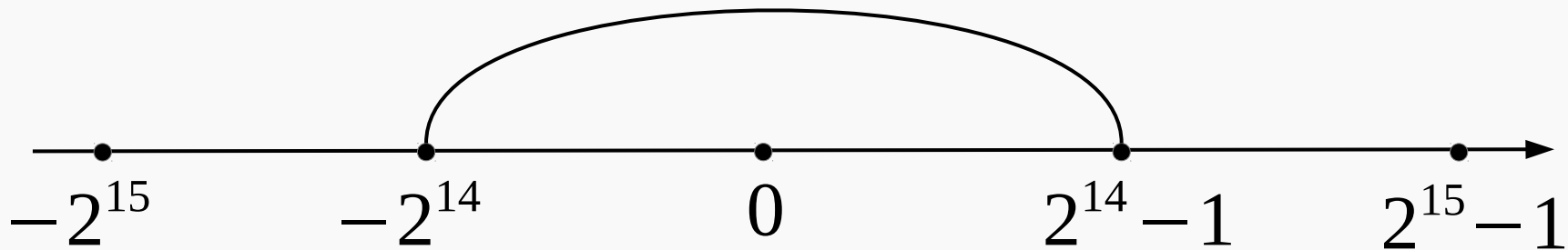
- R is signed 16-bit number
 - X, Y are the sets of the 16 one-bit logical values
 - Z is signed 16-bit number
- Result of the logical operation $X\&Y$ is interpreted as arithmetic operand:
 - $(X\&Y)$ is signed 16-bit number

BComp Workshop Hints

- Allowable values for $R=(X\&Y)+Z$:

For R:
$$-2^{15} \leq R \leq 2^{15} - 1$$

Case 1. We can prevent overflow by the limitation of the size of the operands for addition operation:



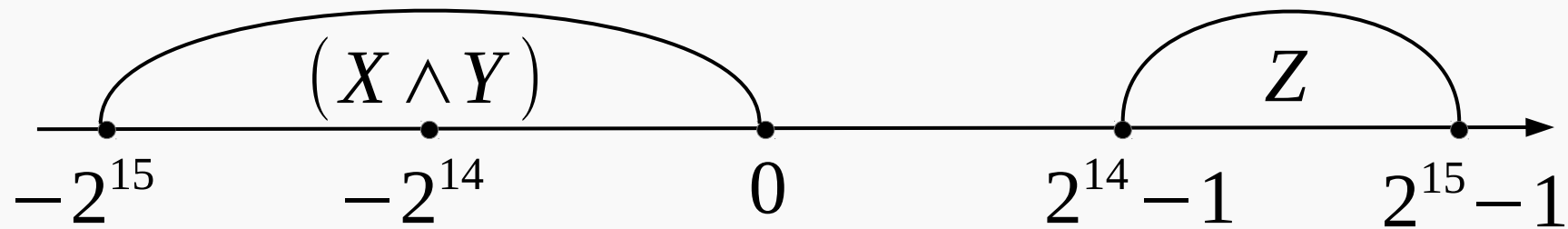
$$\begin{cases} -2^{14} \leq (X \wedge Y), Z \leq 2^{14} \\ X_i, Y_i \in \{0, 1\}, \text{ for } 0 \leq i \leq 14 \end{cases}$$

- In cause of these limits we've lost the half of the possible values for the $(X\&Y) \& Z$. It's not good!

BComp Workshop Hints

Case 2.

$$2^{14} \leq Z \leq 2^{15} - 1$$

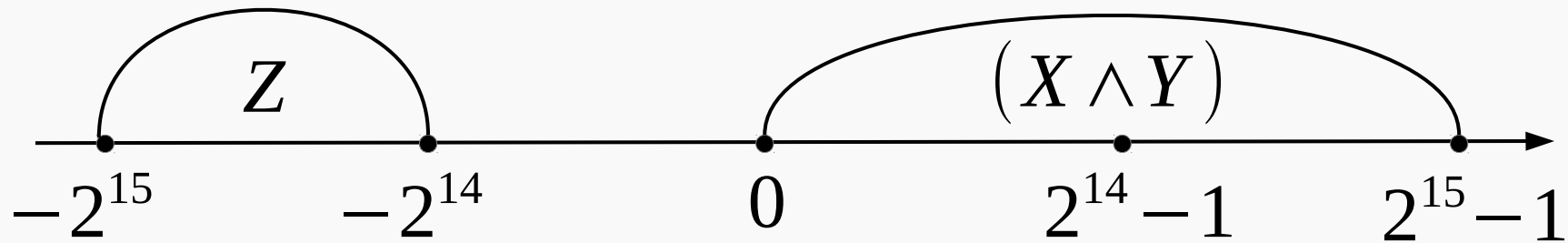


$$\left\{ \begin{array}{l} 2^{14} \leq Z \leq 2^{15} - 1 \\ X_{15} = 1, Y_{15} = 1 \\ X_i, Y_i \in \{0, 1\}, \text{ for } 0 \leq i \leq 14 \end{array} \right.$$

BComp Workshop Hints

Case 3.

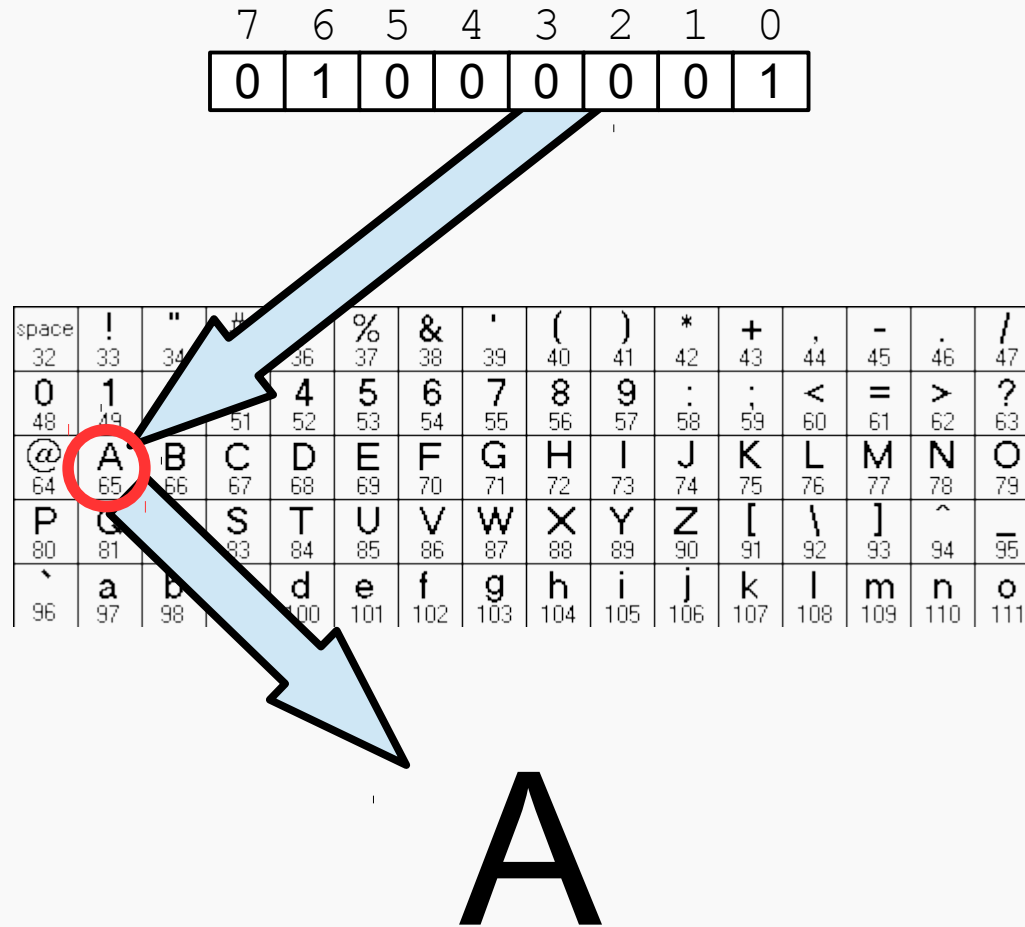
$$-2^{15} \leq Z \leq -2^{14} - 1$$



$$\left\{ \begin{array}{l} -2^{15} \leq Z \leq -2^{14} - 1 \\ X_{15} = 1, Y_{15} = 0 \\ X_i, Y_i \in \{0, 1\}, \text{ for } 0 \leq i \leq 14 \end{array} \right.$$

$$\left\{ \begin{array}{l} -2^{15} \leq Z \leq -2^{14} - 1 \\ X_{15} = 0, Y_{15} = 1 \\ X_i, Y_i \in \{0, 1\}, \text{ for } 0 \leq i \leq 14 \end{array} \right.$$

Text Data Representation



ASCII Symbols

ASCII Code Chart

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	NUL	SOH	STX	ETX	EOT	ENQ	ACK	BEL	BS	HT	LF	VT	FF	CR	SO	SI
1	DLE	DC1	DC2	DC3	DC4	NAK	SYN	ETB	CAN	EM	SUB	ESC	FS	GS	RS	US
2		!	"	#	\$	%	&	'	()	*	+	,	-	.	/
3	0	1	2	3	4	5	6	7	8	9	:	;	<	=	>	?
4	@	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
5	P	Q	R	S	T	U	V	W	X	Y	Z	[\]	^	_
6	`	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o
7	p	q	r	s	t	u	v	w	x	y	z	{		}	~	DEL

7 bits for each symbol! The 8th bit is used for parity check.



UNICODE, UTF-8

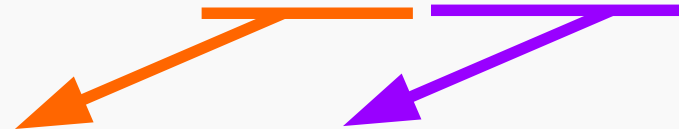
Code point

plane	row	column	
_ _ _ _ _ _ _	_ _ _ _ _ _ _	_ _ _ _ _ _ _	
0 0 0 0, 0 0 0 0	0 0 0 0, 0 0 0 0	0 x x x, x x x x	7x
0 0 0 0, 0 0 0 0	0 0 0 0, 0 x x x	x x y y, y y y y	5x6y
0 0 0 0, 0 0 0 0	x x x x, y y y y	y y z z, z z z z	4x6y6z
0 0 0 w, w w x x	x x x x, y y y y	y y z z, z z z z	3w6x6y6z

UTF-8 encoded

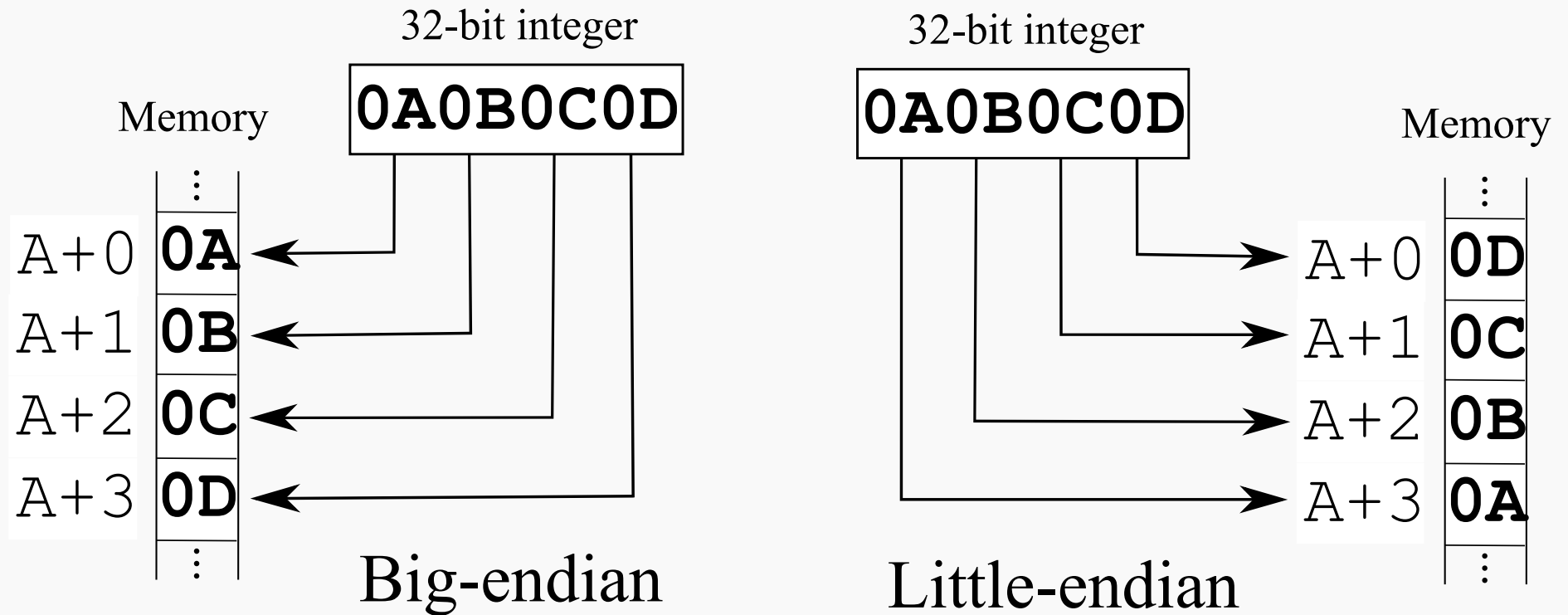
byte 0	byte 1	byte 2	byte 3
_ _ _ _ _ _ _	_ _ _ _ _ _ _	_ _ _ _ _ _ _	_ _ _ _ _ _ _
0 x x x, x x x x			
1 1 0 x, x x x x	1 0 y y, y y y y		
1 1 1 0, x x x x	1 0 y y, y y y y	1 0 z z, z z z z	
1 1 1 1, 0 w w w	1 0 x x, x x x x	1 0 y y, y y y y	1 0 z z, z z z z

Cyrillic «A» → unicode \u0410 → 0000 0100 0001 0000



→ UTF-8 110 1 0000 1001 0000 → D0 90

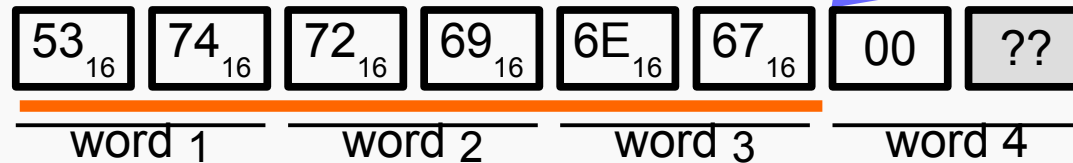
Big-endian & Little-endian



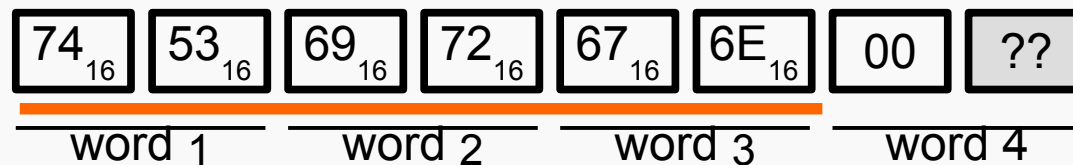
String Data Representation

1) NUL terminated String

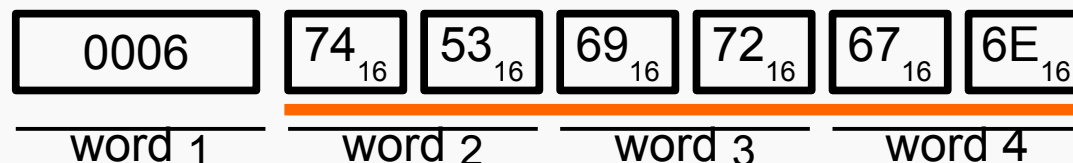
«Little-endian»



«Big-endian»



2) Packaging with length



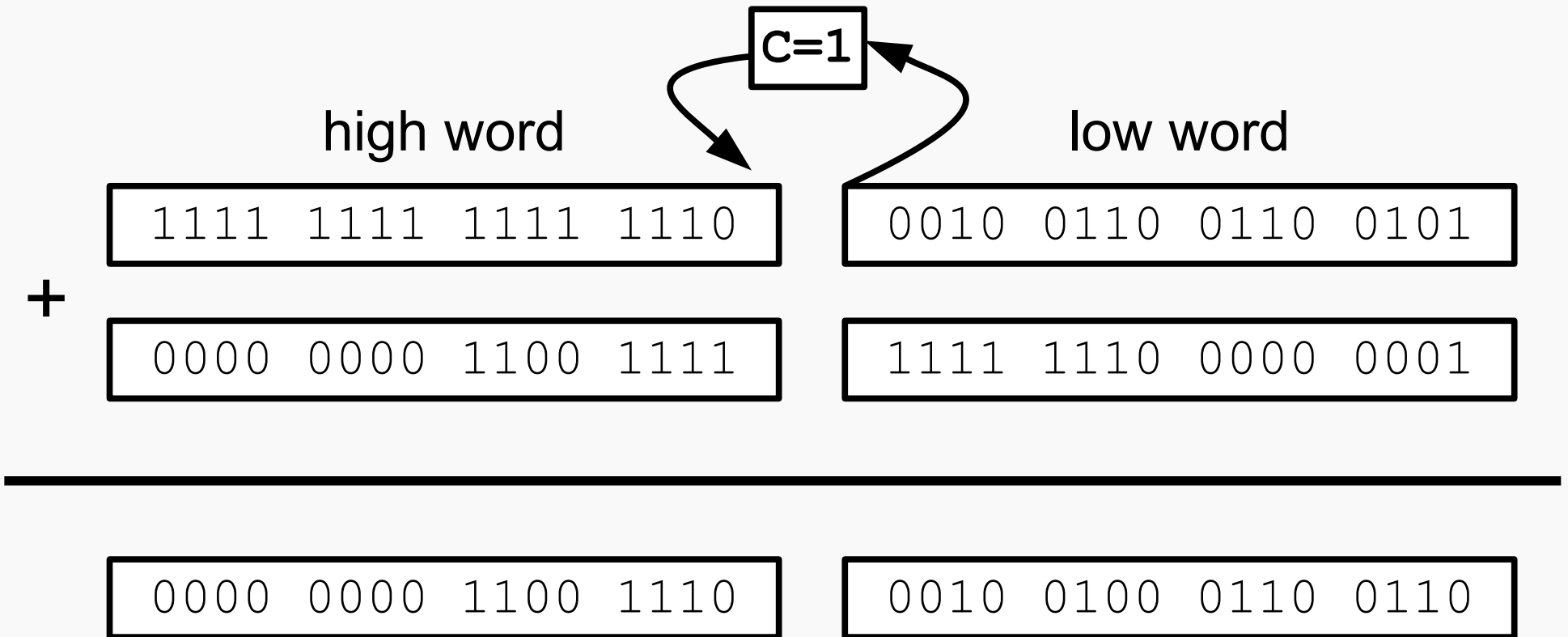
It's all about the hat!



5



32-bit Numbers in BComp



$$\mathbf{fffe2665 + cffe01 = ce2466}$$

$$\mathbf{-121243 + 13630977 = 13509734}$$

Addition of the 32-bit Numbers

Address	Содержимое		Comments
	Code	Mnemonic name	
010	F200	CLA	
011	4019	ADD 19	Low word of the first number
012	401B	ADD 1B	Low word of the second number
013	301D	MOV 1D	Saving the low word of the result
014	F200	CLA	
015	501A	ADC 1A	High word of the first number with carry between the words
016	401C	ADD 1C	High word of the second number
017	301E	MOV 1E	Saving the high word of the results
018	F000	HLT	Halt
019	2665	X	Two memory cells contains the first number
01A	FFFE	X	
01B	FE01	Y	Two memory cells contains the second number
01C	00CF	Y	
01D	2466	R	Two memory cells contains the result
01E	00CE	R	

$$\mathbf{fffe2665 + cffe01 = ce2466}$$

Change the Sign of the 16-bit Number

Address	Content		Comments
	Code	Mnemonic name	
010	F200	CLA	
011	4016	ADD 16	X in accumulator register
012	F400	CMA	Calculate the complement (invert bits of the number)
013	F800	INC	Increment
014	3017	MOV 17	Save the result
015	F000	HLT	
016	0002	X	X
017	FFFE	R	R

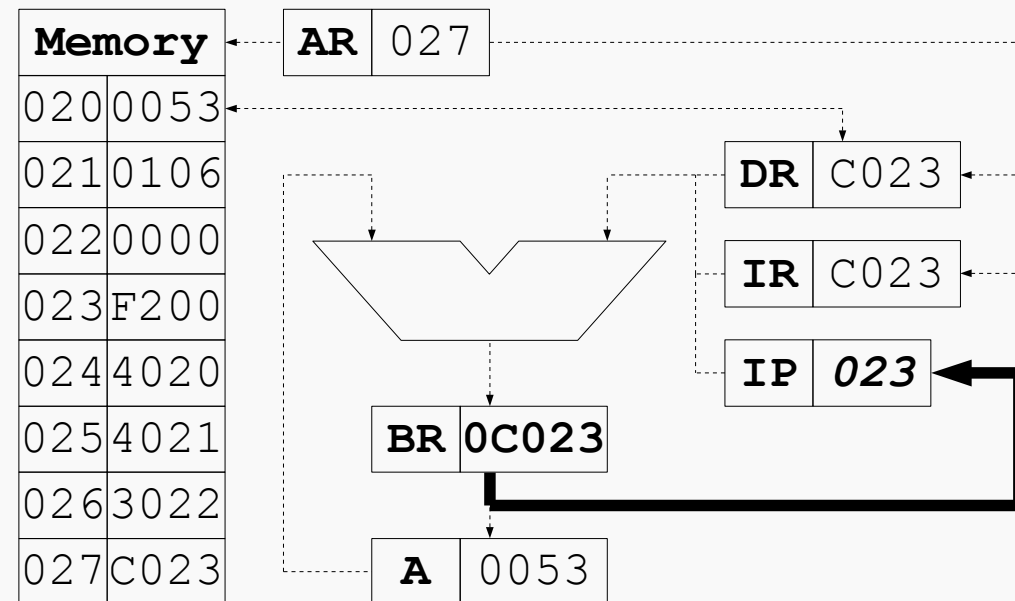
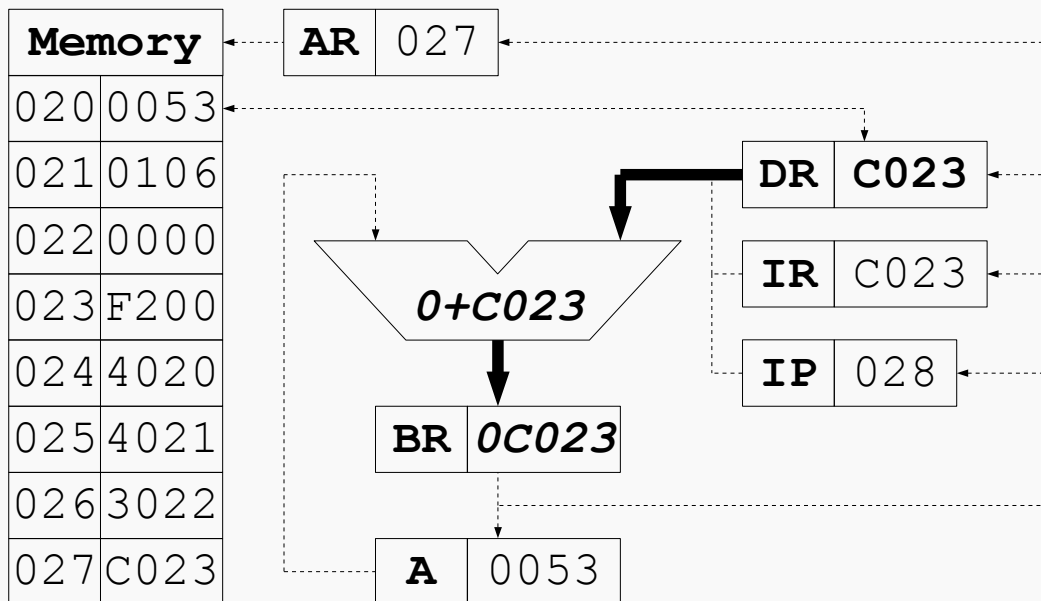
$$R = -X$$

Change the Sign of the 32-bit Number

Address	Content		Comments
	Code	Mnemonic name	
010	F200	CLA	<p>High word of the number X in the accumulator register Calculate the complement (invert bits of the number) Save the high word of the R number</p> <p>Low word of the number X in the accumulator register Calculate the complement (invert bits of the number) Increment (calculate the complement for the low word) Save the low word</p> <p>Add the possible carry bit to the high word Save the high word</p> <p>HLT</p> <p>X = -121243 $R = -X$ R = 121243</p>
011	401E	ADD 1E	
012	F400	CMA	
013	3020	MOV 20	
014	F200	CLA	
015	401D	ADD 1D	
016	F400	CMA	
017	F800	INC	
018	301F	MOV 1F	
019	F200	CLA	
01A	5020	ADC 20	
01B	3020	MOV 20	
01C	F000	HLT	
01D	2665	X	
01E	FFFE	X	
01F	D99B	R	
020	0001	R	

Computational Process Control

Branch if carry is set	BCS M	8XXX	If (C) = 1, then M → IP
Branch if plus	BPL M	9XXX	If (N)= 0, then M → IP
Branch if minus	BMI M	AXXX	If (N) = 1, then M → IP
Branch if zero	BEQ M	BXXX	If (Z) = 1, then M → IP
Unconditional branch	BR M	CXXX	M → IP



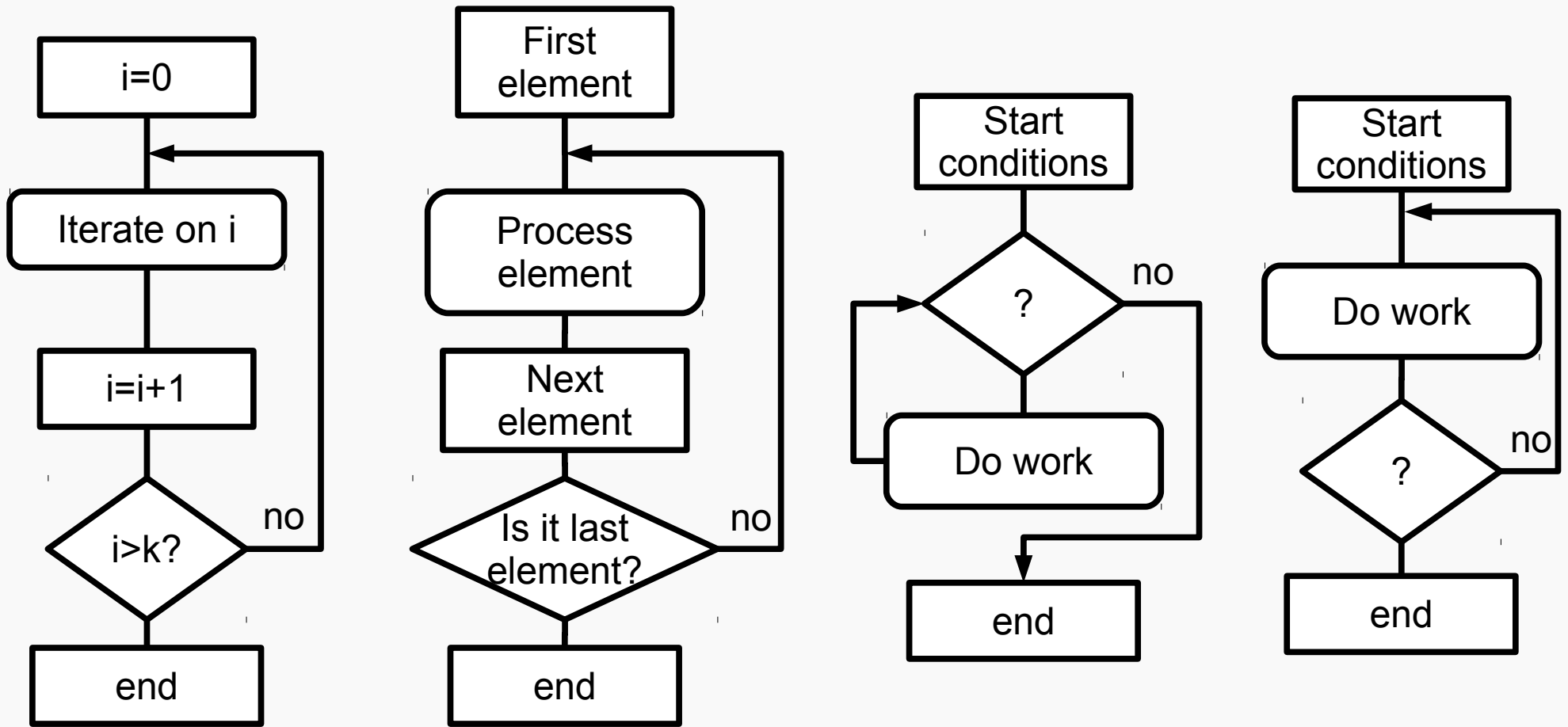
Calculating the Absolute Value of Number

Address	Content		Comments
	Code	Mnemonic name	
010	F200	CLA	
011	4017	ADD 17	X value is in accumulator register
012	9015	BPL 15	If $X \geq 0$ branch to address 15
013	F400	CMA	Calculate the complement (invert bits of the number)
014	F800	INC	Increment
015	3018	MOV 18	Save the result
016	F000	HLT	
017	FFFE	X	X
018	0002	R	$R = X $

$$R = |X|$$

Digression: Cyclic Programms

- for, foreach, while , do-while



R=50Y (case 1)

Remember about allowable ranges!

Address	Content		Comments
	Code	Mnemonic name	
005	0042	Y	First operand
006	0000	Z	Memory cell for intermediate & final results
007	0032	M	Second operand $50=(32)_{16}$
008	0000	C	Cycle counter
010	F200	CLA	Add first operand value Y to intermediate result in memory cell 6 yet another time
011	4006	ADD 6	
012	4005	ADD 5	
013	3006	MOV 6	
014	F200	CLA	Increment cycle counter C value and save it copy to accumulator register
015	4008	ADD 8	
016	F800	INC	
017	3008	MOV 8	
18	6007	SUB 7	If C < M then continue add Y to R
19	A010	BMI 10	
1A	F000	HLT	Halt when all 50 cycles are completed

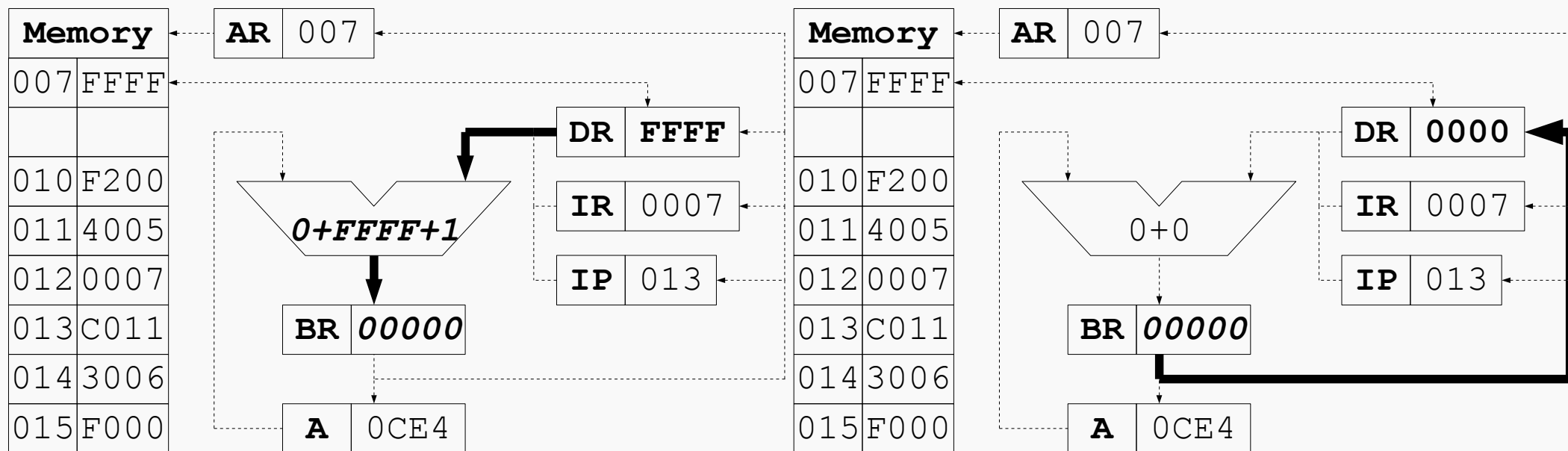
R=50Y (ISZ)

Increment and skip	ISZ M	0XXX	$M + 1 \rightarrow M$, if $(M) \geq 0$, then $(IP) + 1 \rightarrow IP$
--------------------	-------	------	--

Works without usage of the accumulator (A) & carry (C) registers!

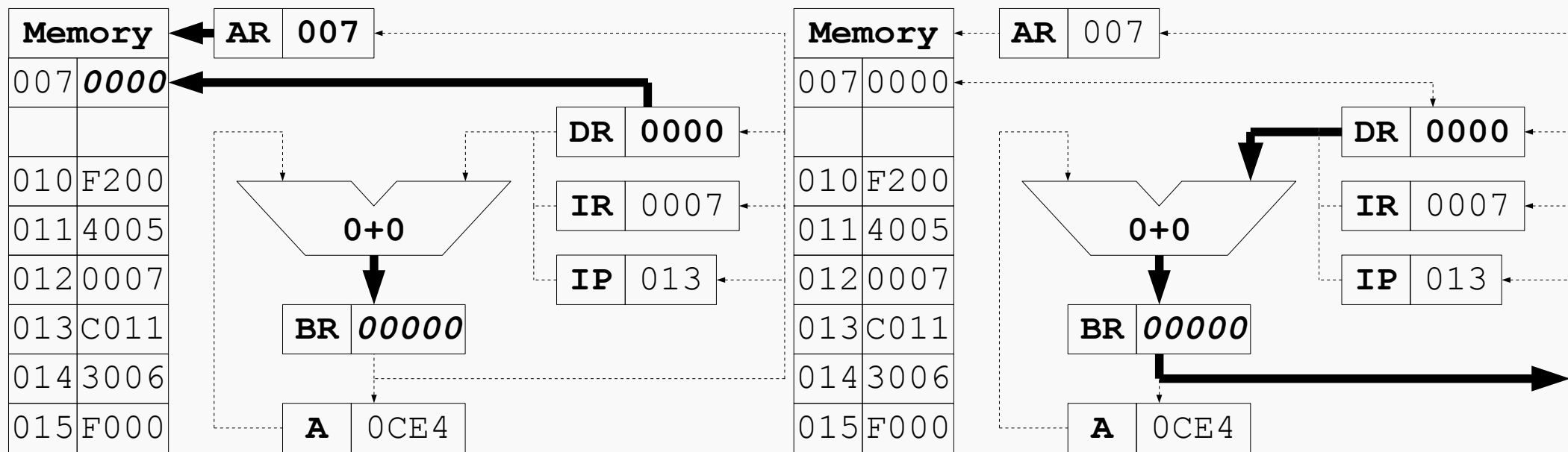
Address	Content		Comments
	Code	Mnemonic name	
005	0042	Y	First operand
006	0000	R	Memory cell for intermediate & final results
007	FFCE	M	Negative value of the second operand (-50)
010	F200	CLA	Add first operand Y value to accumulator register content Increment M. If $M < 0$ perform branch to address 11. If $M = 0$ skip BR.
011	4005	ADD 5	
012	0007	ISZ 7	
013	C011	BR 11	
014	3006	MOV 6	Increment cycle counter C value and save it copy to accumulator register
015	F000	HLT	

Execution Cycle of the ISZ Instruction



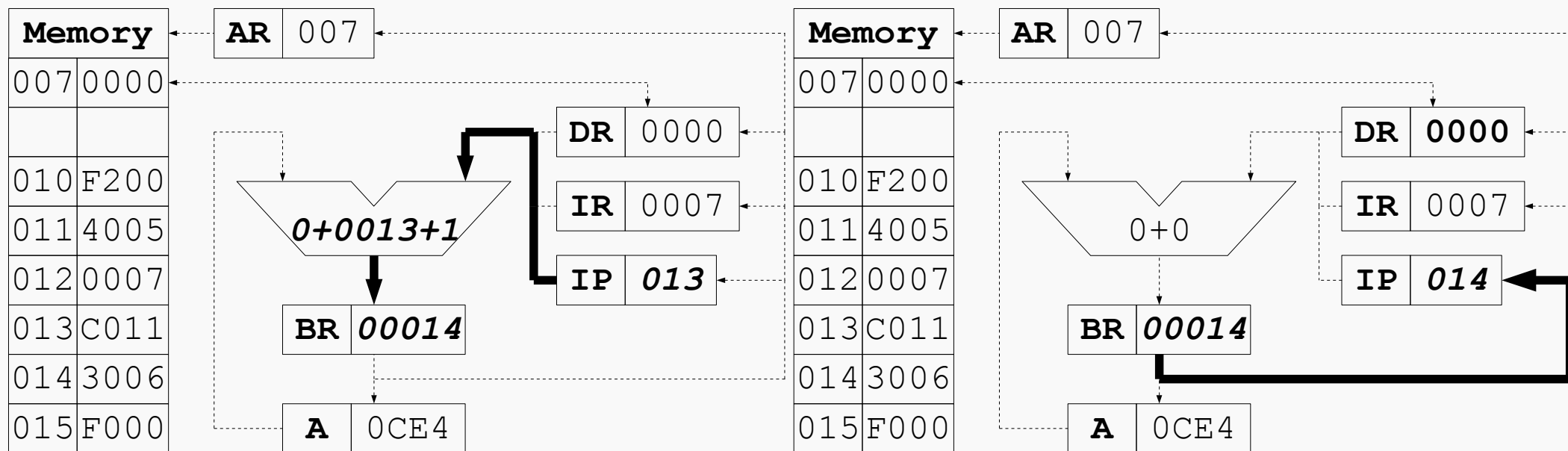
- $42_{16} = 66_{10}$; $50 * 66 = 3300_{10} = \text{CE}4_{16}$
- Before cycle DR contains the data from the memory cell (FFFF) from the address part (007) of the ISZ instruction.
- Increment DR content and save it back to DR through BR.

Execution Cycle of the ISZ Instruction



- Save data from DR to the memory, reset BR.
- Transfer the 15th bit of the DR content to compare scheme where it checks for equality with 1.

Execution Cycle of the ISZ Instruction



- Increment IP content because the 15th bit of the DR content equals 0.



R=50Y

(«chinese code», optimal)

Address	Content		Comments
	Code	Mnemonic name	
005	0078	Y	First operand
006	0000	R	Memory cell for intermediate & final results
007	0000	R'	Intermediate result Y*16
010	F200	CLA	
011	4005	ADD 5	Y
012	F600	ROL	Y*2 after ADD C=0. No need for CLC instruction call
013	3006	MOV 6	Save R to the memory cell
014	F300	CLC	
015	F600	ROL	Y*4
016	F300	CLC	
017	F600	ROL	Y*8
018	F300	CLC	
019	F600	ROL	Y*16
01A	3007	MOV 7	Save R' to the memory cell
01B	F300	CLC	
01C	F600	ROL	Y*32
01D	4007	ADD 7	Add R & R' to the accumulator register content
01E	4006	ADD 6	R=32Y+16Y+2Y=(32+16+2)Y
01F	3006	MOV 6	Save the result
020	F000	HLT	

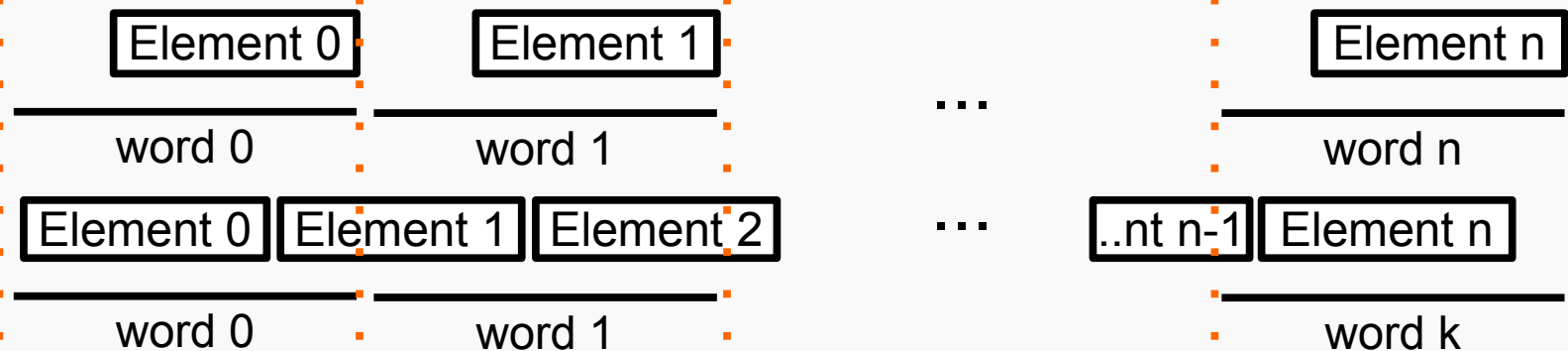
$$50_{10} = 32_{16} = 00110010_2$$

One-Dimensional Arrays Representation

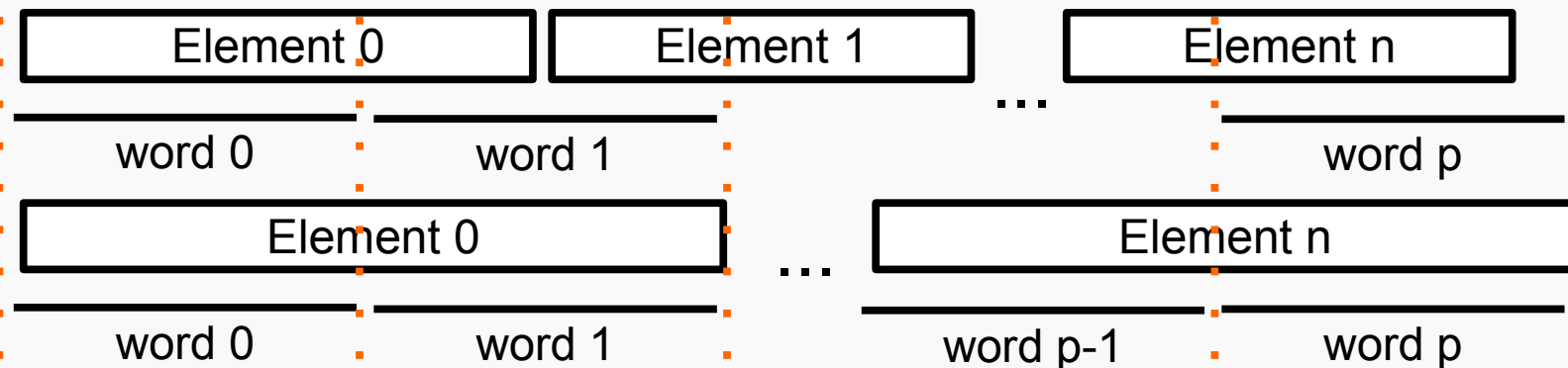
1. Each element of the array has a length of one word



2. If element length is less than word



3. If element length is greater than word



Sum of the Array Elements (Readressing)

Address	Content		Comments
	Code	Mnemonic name	
005	0000	S	Memory cell for intermediate & final results Negative value of the array size (-32)
006	FFE0	C	
010 ... 02F			Array elements
030	F200	CLA	Add result of the previous iteration (S) to the content of the memory cell set in the code of the instruction with address 32 and save it to S
031	4005	ADD 5	
032	4010	ADD ?	
033	3005	MOV 5	
034	F200	CLA	Get the code of the instruction with address 32 Increment it Save it into memory cell 32 Was it last element of the array? If no – branch to address 30 If yes – halt
035	4032	ADD 32	
036	F800	INC	
037	3032	MOV 32	
038	0006	ISZ 6	
039	C030	BR 30	
03A	F000	HLT	

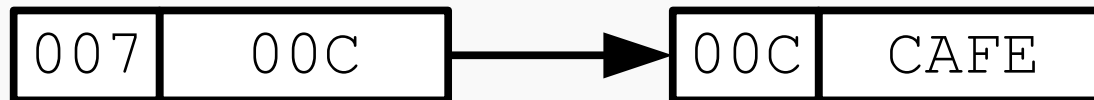
Indirect Addressing & Index Cells

1. Direct addressing

007	CAFE
-----	------

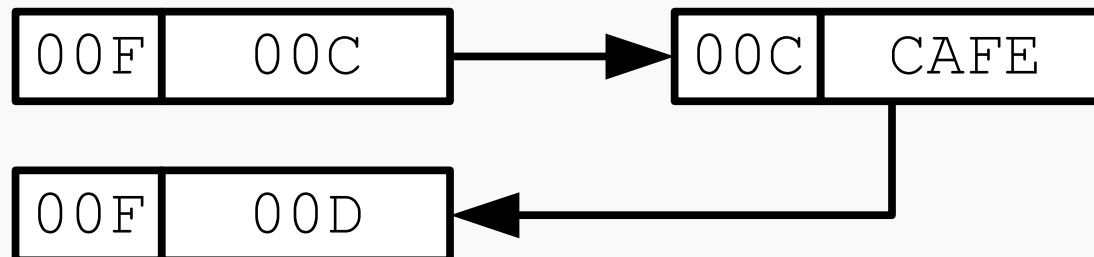
010	CLA
011	ADD 7

2. Indirect addressing



010	CLA
011	ADD (7)

3. Indirect autoincrement addressing (memory cells 8-F)



010	CLA
011	ADD (F)

Sum of the Array Elements (Indirect Addressing)

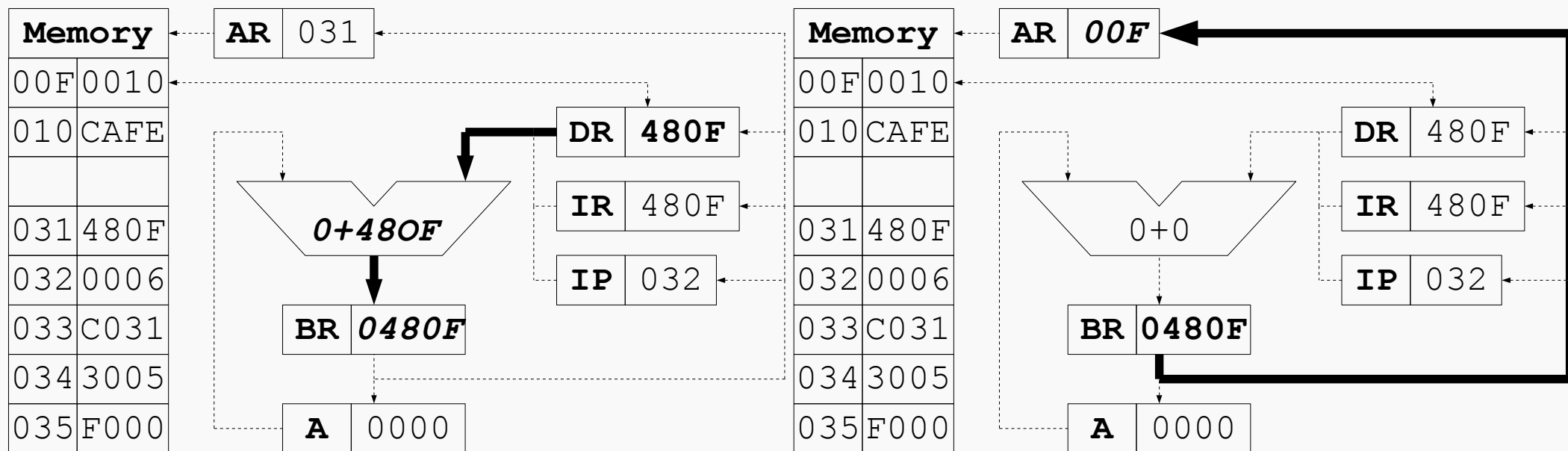
Address	Content		Comments
	Code	Mnemonic name	
005	0000	S	Memory cell for intermediate & final results Negative value of the array size (-32) Current element of the array
006	FFE0	C	
007	0010	I	
010 ... 02F			Array elements
030	F200	CLA	Accumulator register contains the next element of the array Increment element's address without A content change This instruction never will be executed Was it the last element of the array? If no – continue calculating Result is in memory cell 5
031	4807	ADD (7)	
032	0007	ISZ 7	
033	F100	NOP	
034	0006	ISZ 6	
035	C031	BR 31	
036	3005	MOV 5	
037	F000	HLT	



Sum of the Array Elements (Indirect Autoincrement Addressing)

Address	Content		Comments
	Code	Mnemonic name	
005	0000	S	Memory cell for intermediate & final results Negative value of the array size (-32)
006	FFE0	C	
...			
00F	0010	I	Current element of the array
010			Array elements
...			
02F			
030	F200	CLA	Accumulator register contains the next element of the array Increment the content of the F memory cell Was it the last element of the array? If no – continue calculating Result is in memory cell 5
031	480F	ADD (F)	
032	0006	ISZ 6	
033	C031	BR 31	
034	3005	MOV 5	
035	F000	HLT	

Address Fetch Cycle

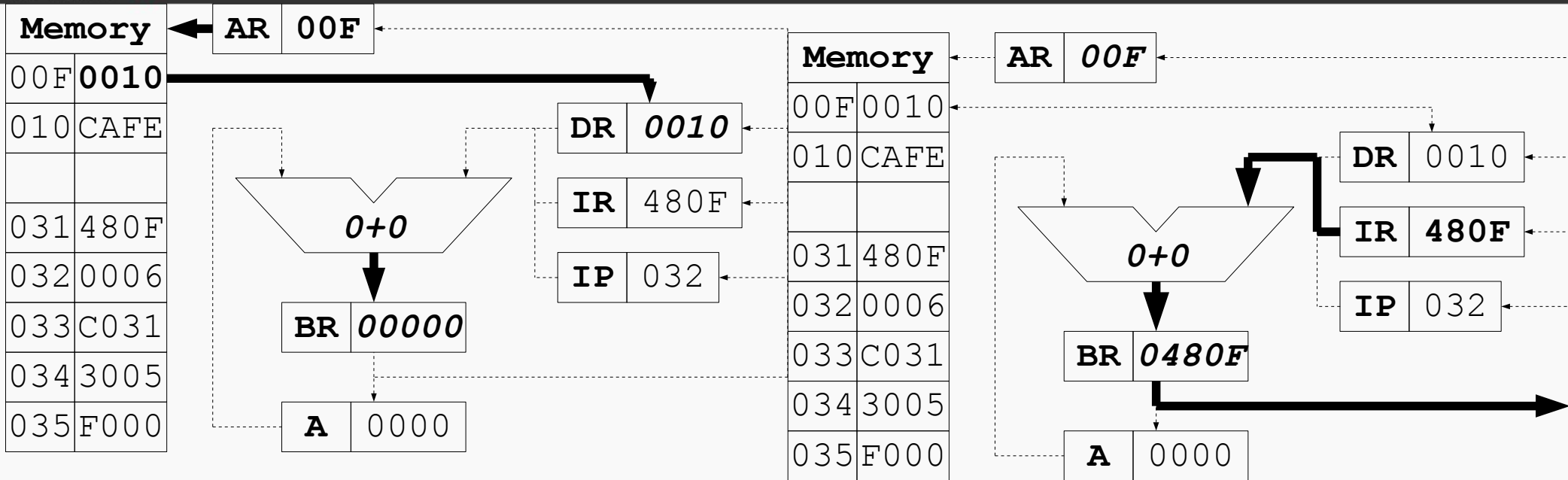


- In the end of the instruction fetch cycle IP was incremented and code $480F$ of the current instruction ADD (F) was saved into DR & IP
- On the next step lower 11 bits of the DR content transfers into AR through BR



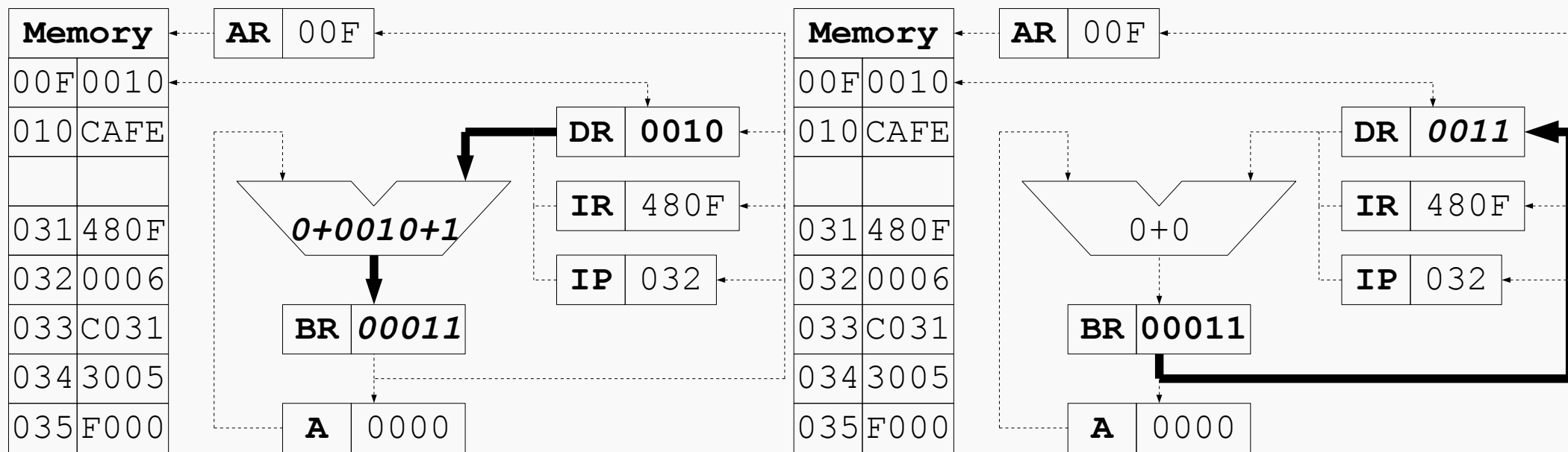
NTMO BT

Address Fetch Cycle



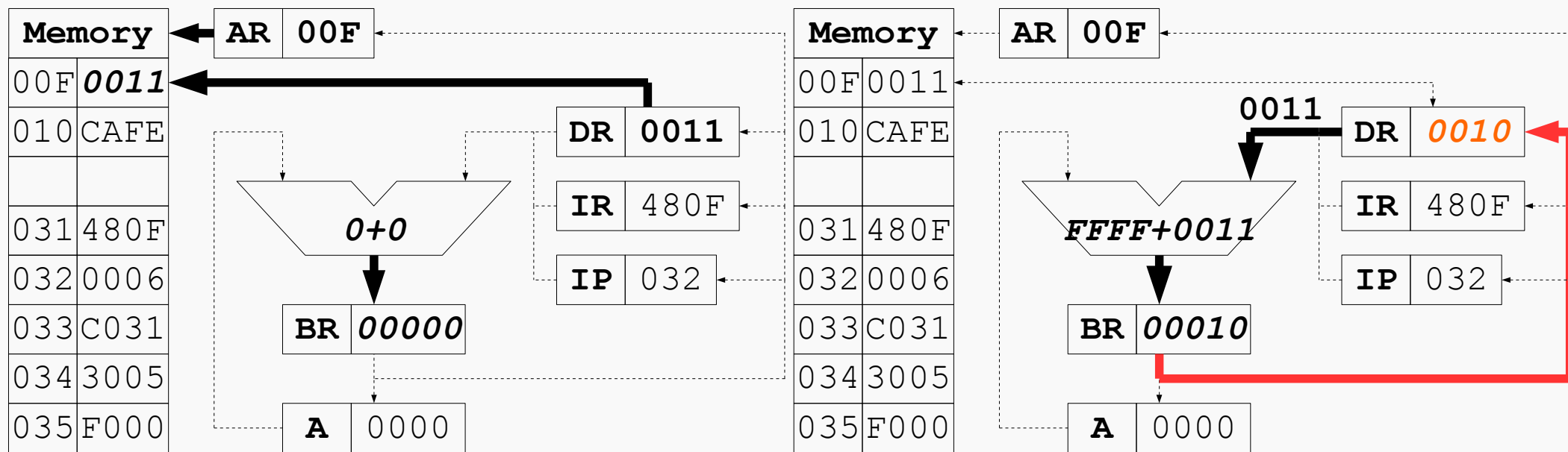
- Save the content of the AR to DR
- Sequentially check lower bits of the IR content for compliance with mask 000 0000 1XXX, i.e. check is it index cell (008-00F) or not

Address Fetch Cycle



- If it is index cell increment it content in DR and save it to BR
- Transfer incremented content of BR to DR

Address Fetch Cycle



- Save the new, incremented content of the index cell into the memory
- On the next step we will need to use the old value (on the operand fetch cycle) so the content of the DR decrements, saves to BR and on the **next** clock saves back to DR ($DR+COM(0)=DR-1 \rightarrow \mathbf{BR} \rightarrow \mathbf{DR}$)

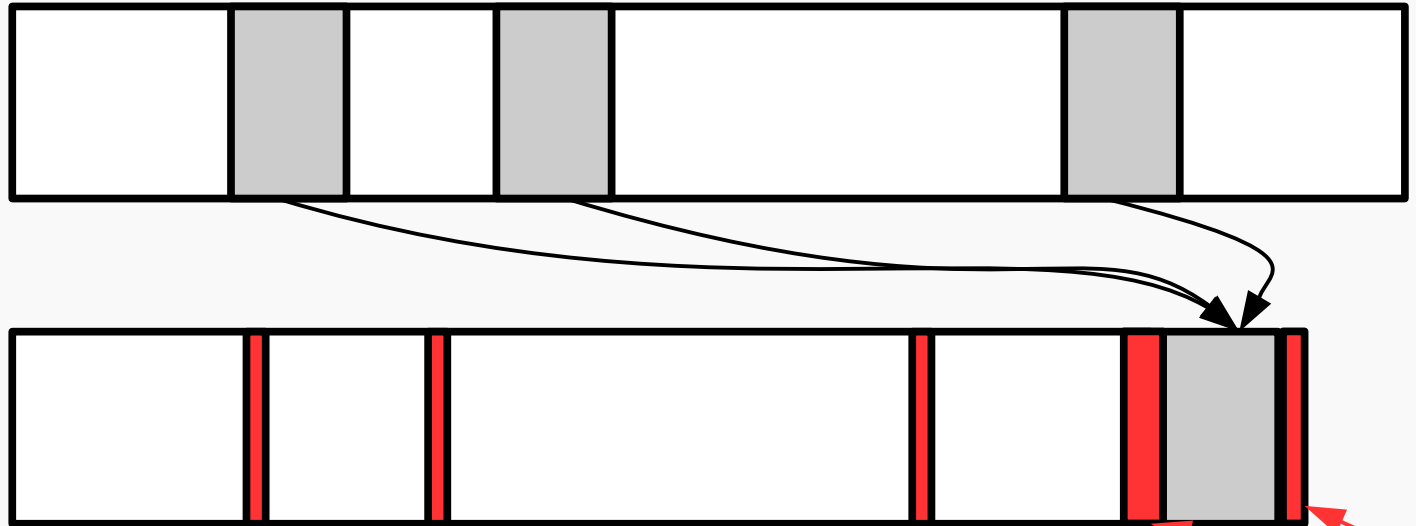


6



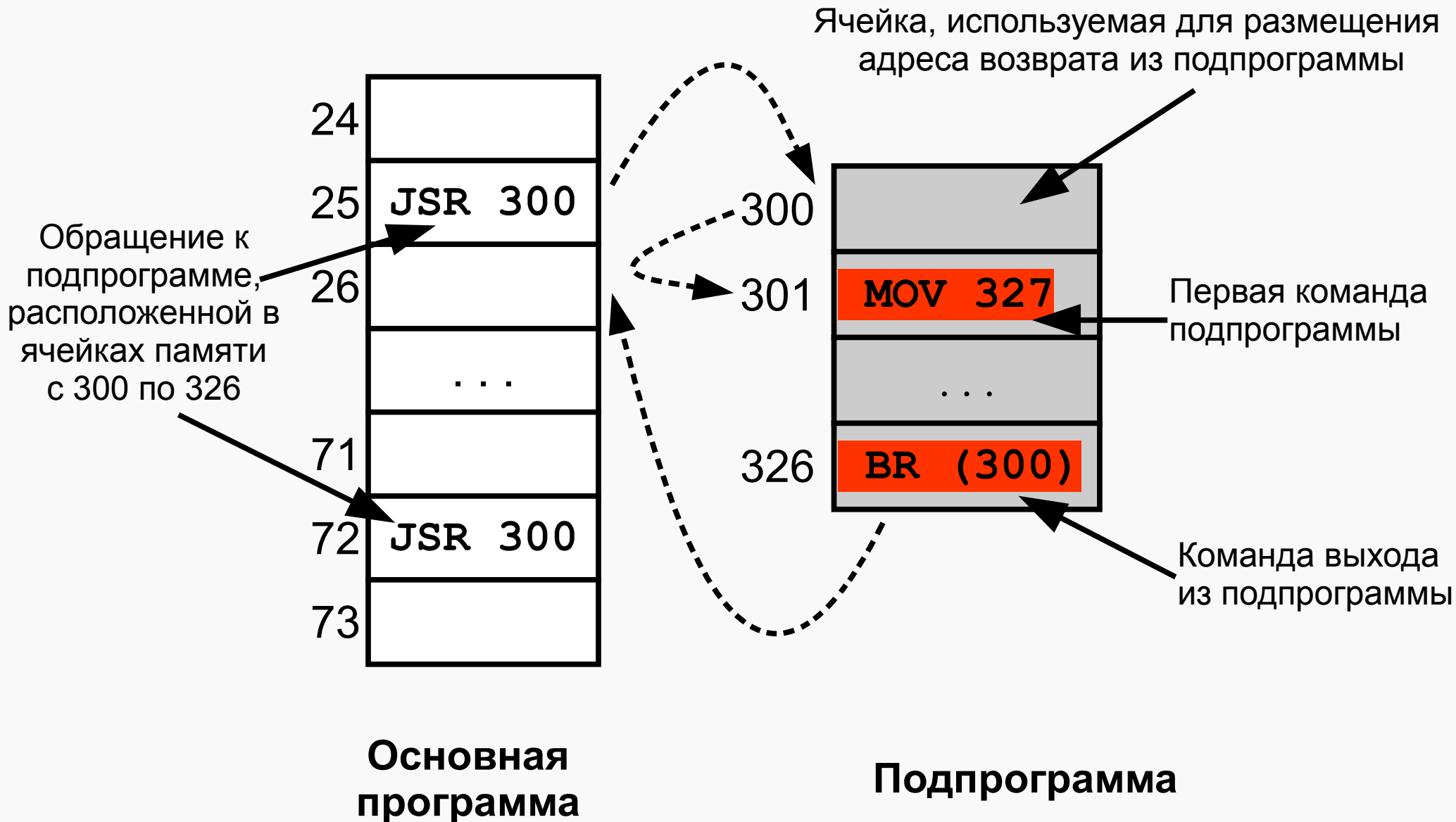
Мотивация

- Зачем?



- Когда выгодно создавать подпрограмму?
 - Размер кода
 - Передача и обработка параметров; возвращение результатов
- Инлайнинг — обратный процесс

БЭВМ: Вызов программы и возврат из нее



Передача параметров и получение результатов

- Аккумулятор (Регистры Общего Назначения)
 - Сколько параметров можно передать в БЭВМ?
- Адресуемые ячейки памяти
 - Необходимо организовать
- Стек
- Регистровые окна

Комплекс программ: передача параметров через аккумулятор

Подпрограмма вычисления удвоенного модуля для чисел в ячейках 58,63,71 с размещением результатов в ячейках 74,77,82

Адрес	Содержимое		Комментарии
	Код	Мнемоника	
010	F200	CLA	Первое число Аргумент в 58 Результат в 74
011	4058	ADD 58	
012	2030	JSR 30	
013	3074	MOV 74	
014	F200	CLA	Первое число Аргумент в 63 Результат в 77
015	4063	ADD 63	
016	2030	JSR 30	
017	3077	MOV 77	
18	F200	CLA	Первое число Аргумент в 71 Результат в 82
19	4071	ADD 71	
1A	2030	JSR 30	
1B	3082	MOV 82	
1C	F000	HLT	

Подпрограмма
вычисления
 $R=2|X|$

Адрес	Содержимое	
	Код	Мнемоника
030	0000	Адрес возв.
031	9034	BR 34
032	F400	CMA
033	F800	INC
034	F300	CLC
035	F600	ROL
036	C830	BR (30)

Комплекс программ: передача через адреса ячеек памяти

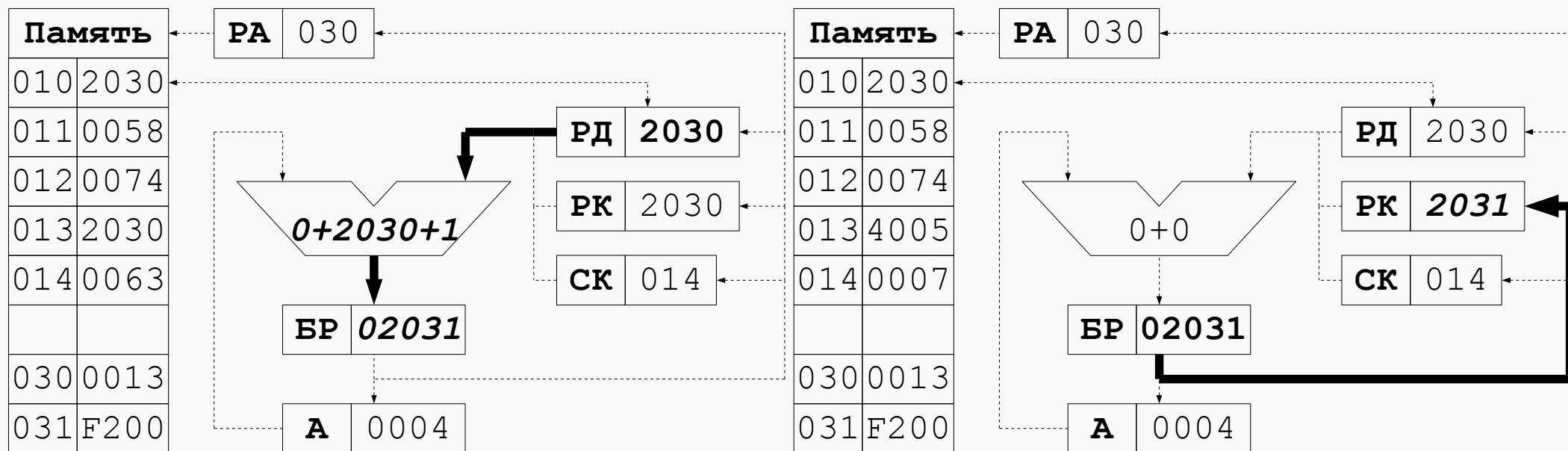
Подпрограмма
вычисления
 $R=2|X|$

Основная программа

Адрес	Содержимое	
	Код	Мнемоника
010	2030	JSR 30
011	0058	Адрес X1
012	0074	Адрес R1
013	2030	JSR 30
014	0063	Адрес X2
015	0077	Адрес R2
016	2030	JSR 30
017	0071	Адрес X3
018	0082	Адрес R3
019	F000	HLT

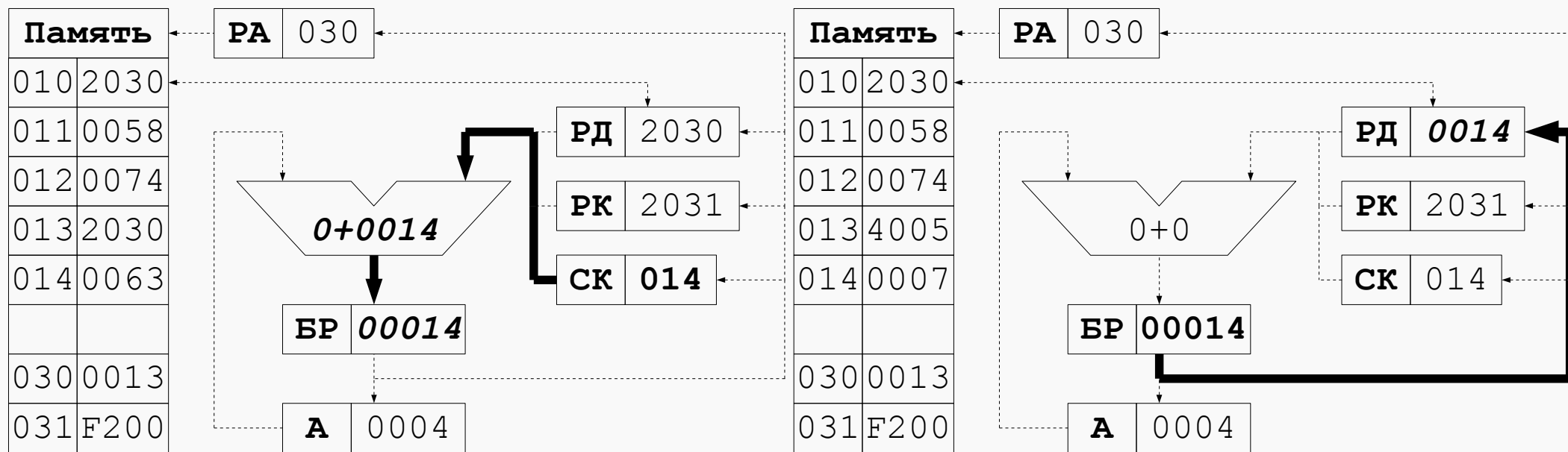
Адрес	Содержимое		Комментарии
	Код	Мнемоника	
030	0000	Адрес возв.	По адресу возврата выбрать адрес Xi и записать в яч. 035 Адрес возврата++ СК>0 всегда!
031	F200	CLA	
032	4830	ADD (30)	
033	3035	MOV 35	
034	0030	ISZ 30	
035	0000	Адрес Xi	
036	F200	CLA	По адресу возврата выбрать адрес Ri и записать в яч. 03A Адрес возврата++ СК>0 всегда!
037	4830	ADD (30)	
038	303A	MOV 3A	
039	0030	ISZ 30	
03A	0000	Адрес Ri	
03B	F200	CLA	Выбрать Xi
03C	4835	ADD (35)	
03D	9040	BPL 40	Xi<0, меняем знак
03E	F400	CMA	
03F	F800	INC	Умножаем на 2
040	F300	CLC	
041	F600	ROL	Записываем Ri
042	383A	MOV (3A)	
043	C830	BR (30)	Возврат

Цикл исполнения JSR



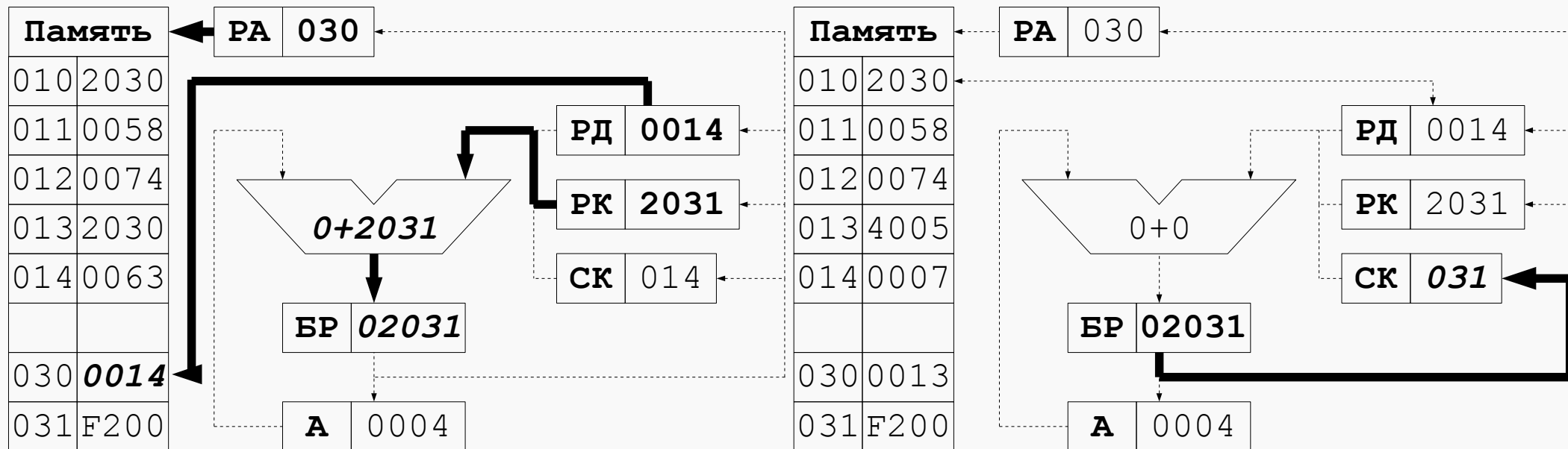
- Окончание декодирования адресной команды записывает в РА адресную часть команды
- Содержимое РД увеличивается на 1 и через БР помещается на временное хранение в РК, который не будет больше использоваться в декодировании

Цикл исполнения JSR



- Адрес возврата из подпрограммы (014 — содержимое СК) записывается через БР в РД

Цикл исполнения JSR



- Содержимое адреса возврата записывается в первую ячейку подпрограммы, одновременно ПК переписывается в БР
- Младшие 11 бит содержимого БР записывается в СК, происходит переход к первой команде подпрограммы

Цикл исполнения BR (адр)

Чем отличается от
цикла исполнения BR адр
из лекции 5?

Рекурсивность

- Рекурсивность — способность подпрограммы вызывать саму себя.

- Подсчет факториала

```
f (n) {
    if (n==0) return 1;
    return n*f(n-1);
}
```

- В БЭВМ?

Адрес	Содержимое	
	Код	Мнемоника
030	0000	Адрес возв.
031	9034	BEQ 37
032	F500	DEC
033	2030	JSR 30
034		...
040	C830	BR (30)

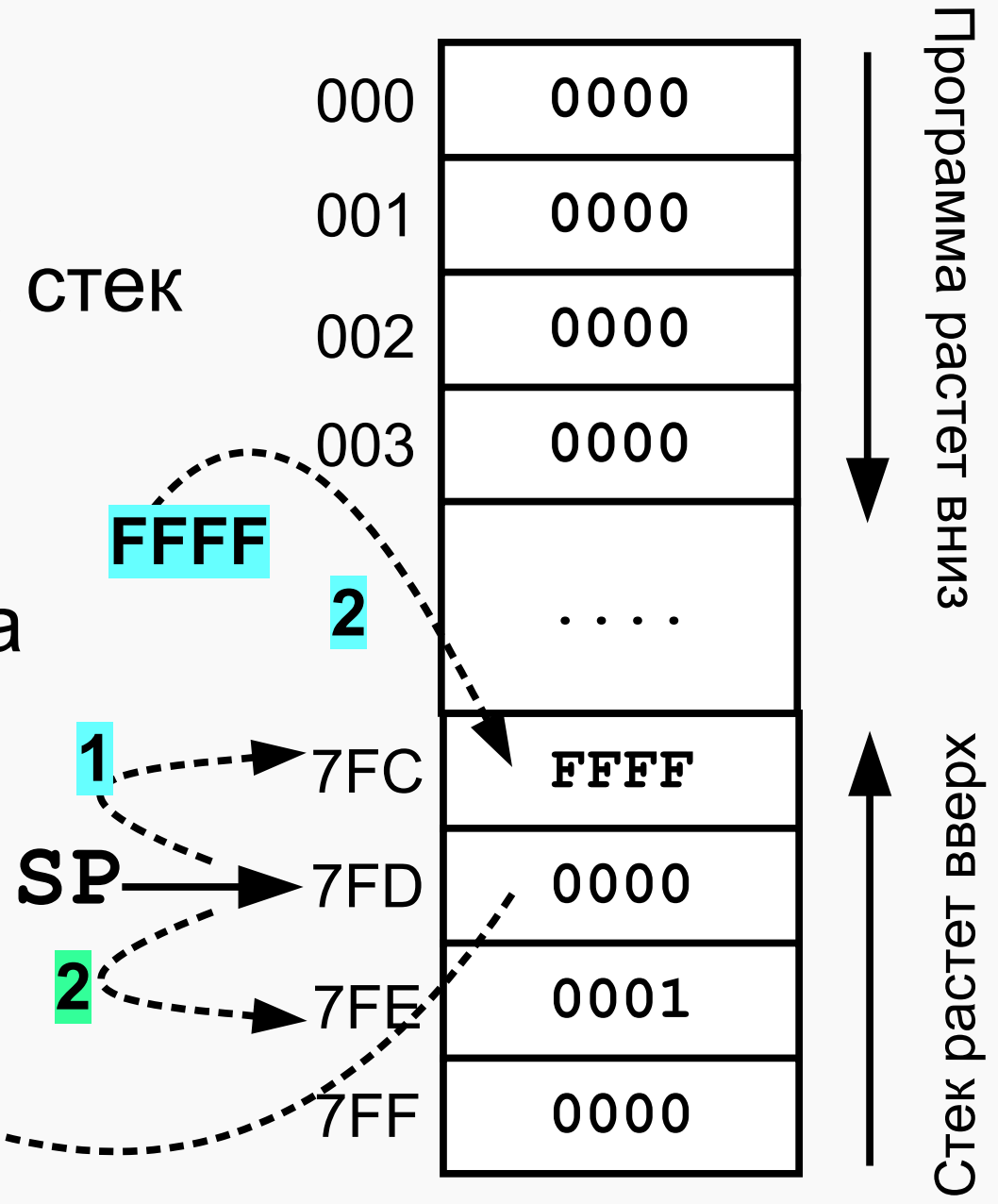
Реентерабельность R=50Y

- Реентерабельность — способность программы быть запущенной несколько раз

Адрес	Содержимое		Комментарии
	Код	Мнемоника	
005	0042	Y	Множимое Ячейка для накопления и хранения результата Текущее (цикловое) значение множителя Начальное отрицательное значение множителя (-50)
006	0000	R	
007	0000	M	
008	FFCE	Mнач	
00C	F200	CLA	Очистка ячейки для накопления результата Инициализация текущего множителя начальным значением
00D	3006	MOV 6	
00E	4008	ADD 8	
00F	3007	MOV 7	
010	F200	CLA	К содержимому аккумулятора добавляется Y M наращивается на 1, в случае если M<0 выполняется переход на 11 адрес. Если M=0, то BR пропускается.
011	4005	ADD 5	
012	0007	ISZ 7	
013	C011	BR 11	
014	3006	MOV 6	Содержимое счетчика циклов C увеличивается на 1, а его копия пока сохраняется в аккумулятор
015	F000	HLT	

Стек

- SP — stack pointer (указатель стека)
- **PUSH** — положить на стек
 1. --SP
 2. Записать по (SP)
- **POP** — снять со стека
 1. Прочитать по (SP)
 2. SP++
- Взять i-й элемент
 1. (SP+/-i)

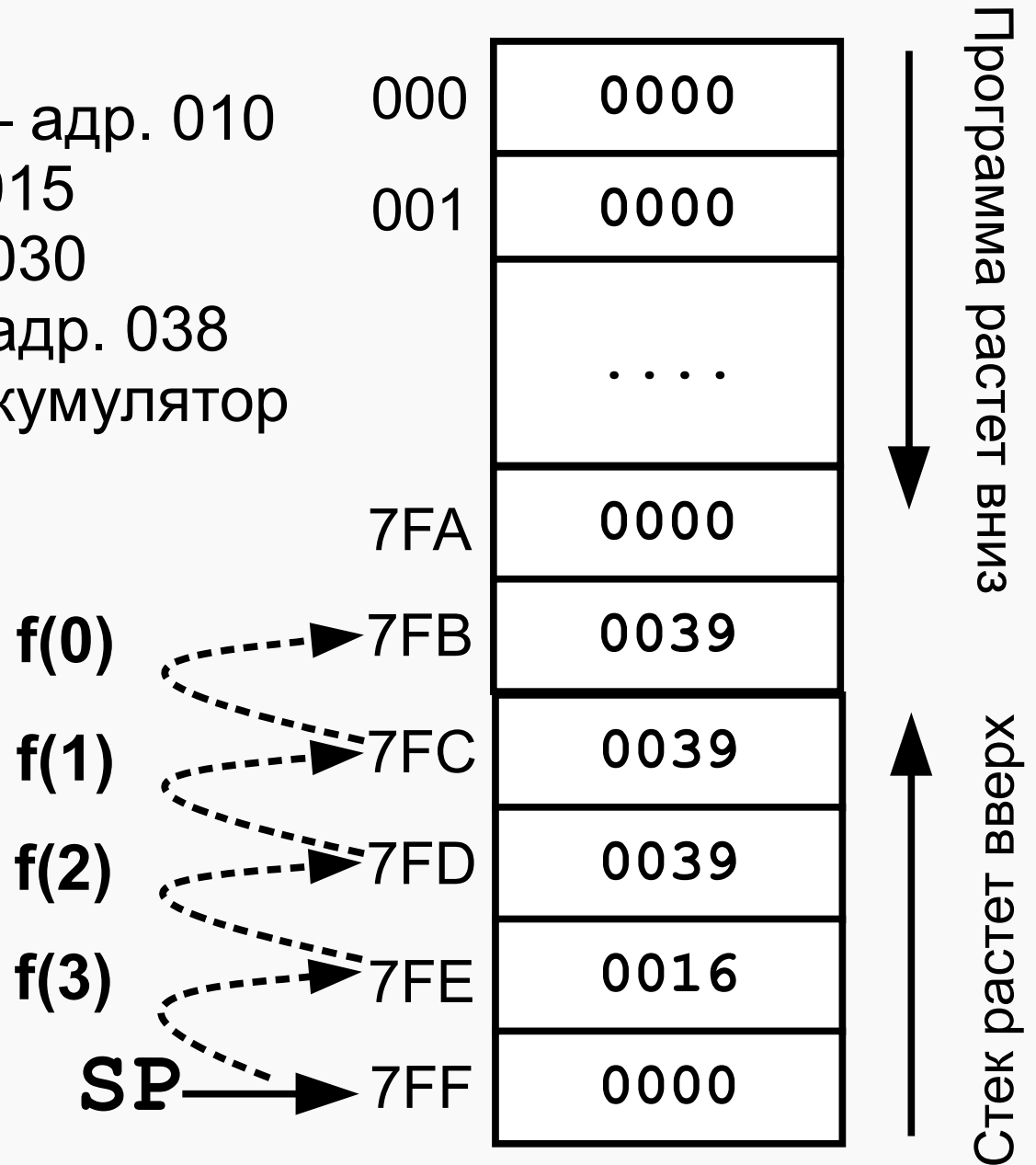


Вызов подпрограмм с использованием стека

Основная программа — адр. 010
 Первый вызов — адр. 015
 Подпрограмма — адр. 030
 Рекурсивный вызов — адр. 038
 Параметры — через аккумулятор

```
f (n) {
    if (n==0) return 1;
    return n*f(n-1);
}
```

• f(3)

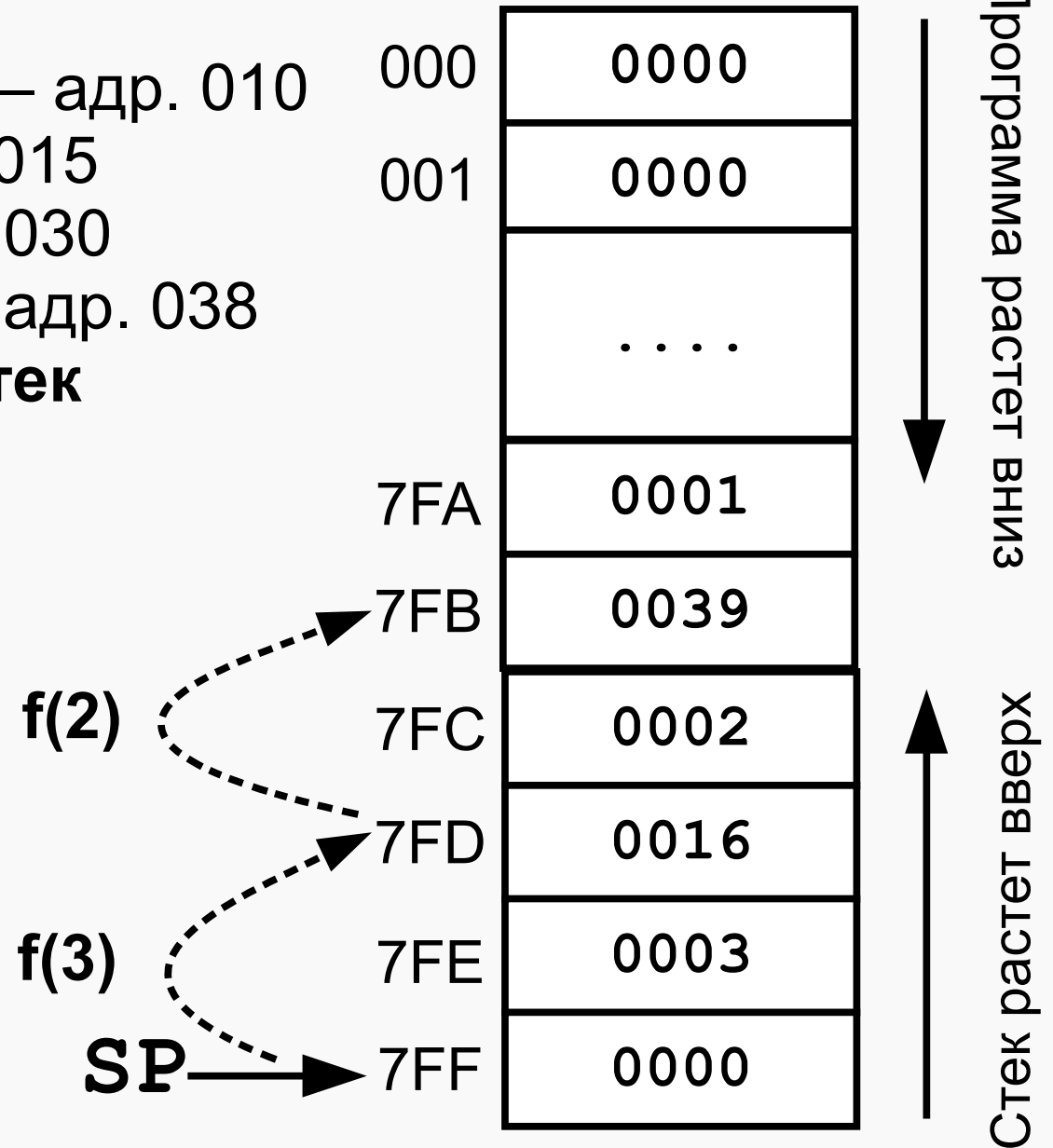


Передача параметров с использованием стека

Основная программа — адр. 010
 Первый вызов — адр. 015
 Подпрограмма — адр. 030
 Рекурсивный вызов — адр. 038
 Параметры — через **стек**

```
f (n) {
    if (n==0) return 1;
    return n*f(n-1);
}
```

- f(3)
 - push
 - call f
 - pop



Ассемблер БЭВМ

Назначение	Синтаксис	Пример использования
Размещение в памяти	ORG адрес	ORG 10
Прямая адресация	[метка:] МНЕМОНИКА АРГУМЕНТ	MOV R
Косвенная адресация	[метка:] МНЕМОНИКА (АРГУМЕНТ)	ADD (K)
Безадресная команда	[метка:] МНЕМОНИКА	BEGIN: CLA
Команда ввода-вывода	[метка:] МНЕМОНИКА АДРЕСВУ	OUT 3
Константы	[метка:] WORD знач. [, знач...] [метка:] WORD кол. DUP (знач.)	X: WORD ? Y: WORD X VALUES: WORD 1,2,3 ARRAY: WORD 10 DUP (?)

Ассемблер БЭВМ

Подсчет отрицательных элементов массива

```
ORG      00D
K:       WORD      ? ; Адрес первого элемента массива
N:       WORD      ? ; Количество элементов массива
R:       WORD      ? ; Результат
BEGIN:   CLA           ; Первая команда – адрес 010
         MOV      R
         ADD      (K)
         BPL      SKIP
         CLA
         ADD      R
         INC
         MOV      R
SKIP:    ISZ      N
         BR       BEGIN
         HLT

ORG      020
X:       WORD      6 DUP (?) ; Элементы массива
```

PIC - Position Independent Code (перемещаемый код)

- Код, который работает относительно того адреса на который загружен

Адрес	Содержимое	
	Код	Мнемоника
010	F200	CLA
011	4017	ADD 17
012	9015	BPL 15
013	F400	CMA
014	F800	INC
015	3018	MOV 18
016	F000	HLT
017	FFFE	X
018	0002	R= X

- Относительная (СК) адресация
 - ADD +5 ;(12+5=17)
 - BPL +2
- Адресация относительно базового адреса
 - ADD 17+0
 - MOV 17+1
- Сегментная адресация
 - Code-, Data-, StackSegment
 - 17 → DS; MOV DS:1
- Выполняется транслятором ассемблера из обычных меток и загрузчиком программы



Загрузчик и динамический линковщик программ

- Любая ОС имеет соответствующую программу или часть ядра
 - Загрузка по выбранному ОС адресу (даже в виртуальной памяти)
 - Изменение константных частей адресов в программе
 - Загрузка базовых значений регистров
 - Динамическая загрузка разделяемых библиотек
 - Связывание адресов основной программы с вызываемыми библиотеками

Библиотеки

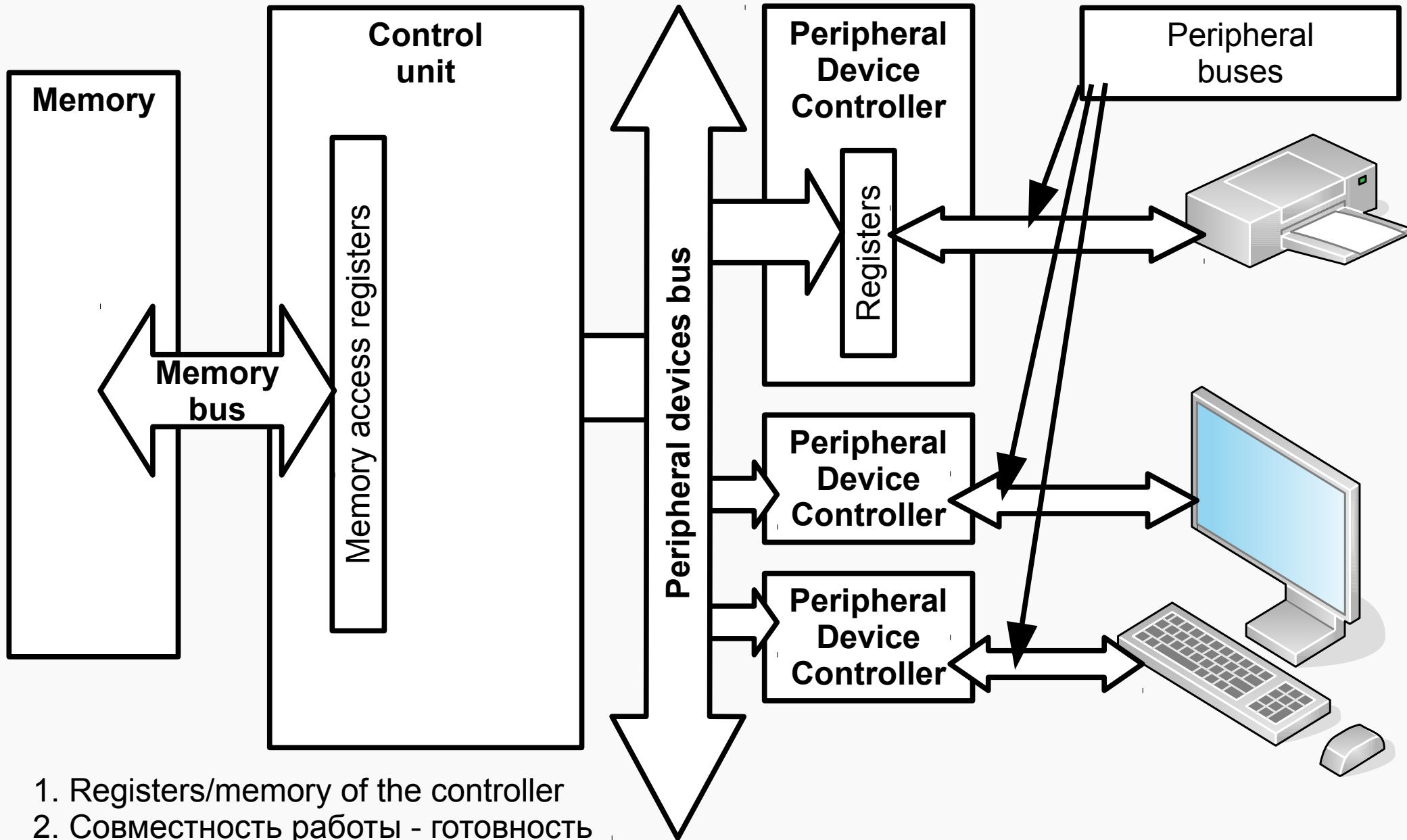
- Набор стандартных библиотечных функций
- Разделяемые (динамически линкуемые) и архивные (статически линкуемые)
 - `# find /lib /usr/lib -name "*.so" |wc -l`
3510
 - Статические связывают вызовы функций с телом функции в процессе компиляции
 - Динамические — в момент загрузки
- Если вам нужна функция — см. в библиотеки

Input / Output

7



Connecting Devices to Computer



1. Registers/memory of the controller
2. Совместность работы - готовность

Drivers

- Organizes collaboration with devices.
- «Knows» about device work principles, register addresses and supported work modes.
- Are managed by the uniform program interface.

Input / Output

Programmed (PIO)

Direct Memory Access (DMA)

- Data exchange initiation:
 - Synchronous
 - Asynchronous
 - Interrupt-driven
- Data transfer:
 - Synchronous/Asynchronous
- Data exchange finish and results receiving by driver (or other program):
 - Synchronous/Asynchronous



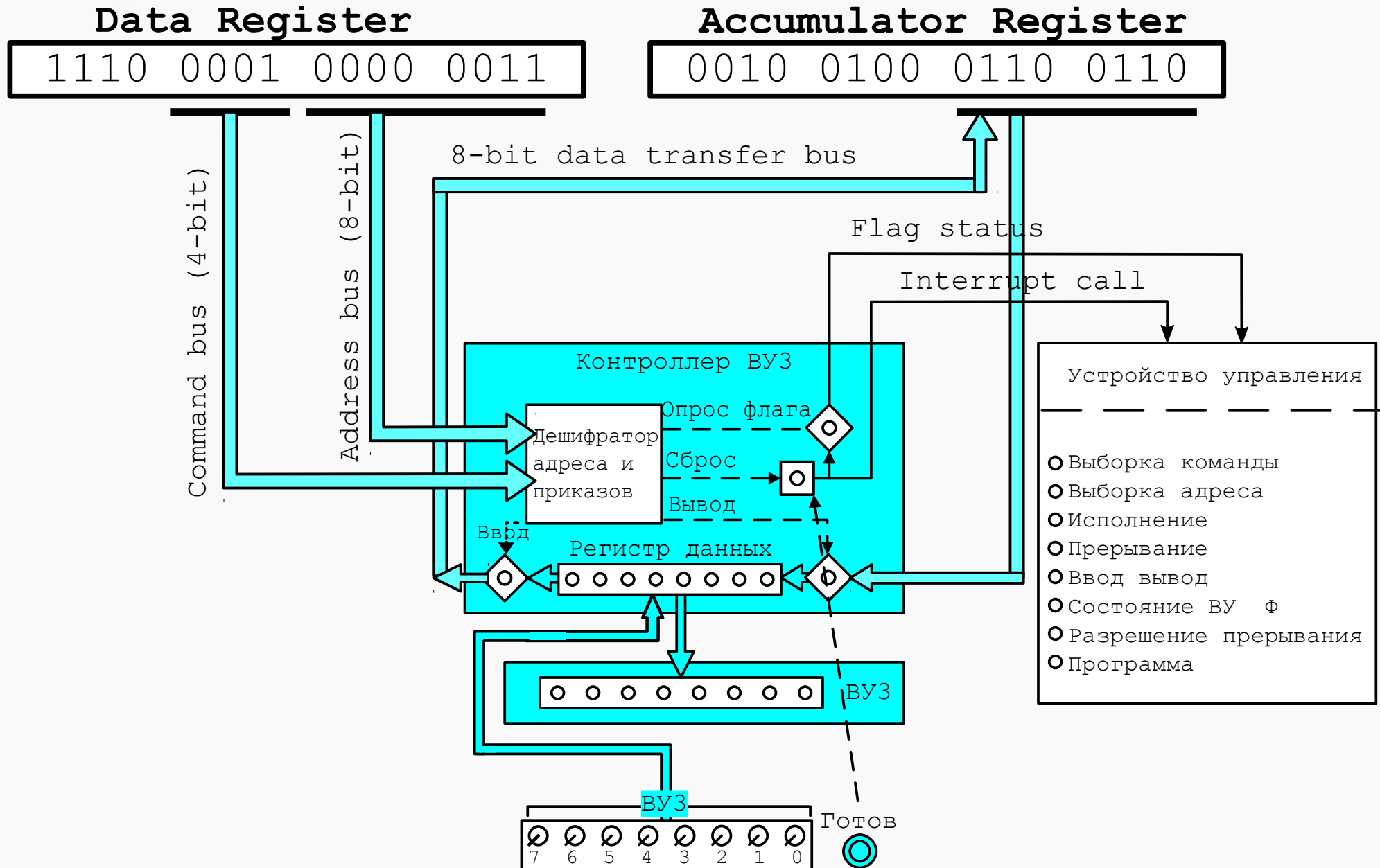
BComp I/O Instructions

Name	Mnemonic name	Code	Description
Clear flag	CLF B	E0XX	0 → device flag
Check flag	TSF B	E1XX	If (device flag B) = 1, then (IP) + 1 → IP
Input	IN B	E2XX	(B) → A
Output	OUT B	E3XX	(A) → B

Команды ввода-вывода

1 1 1 0			
Код операции _____	Приказ на ввод-вывод _____	Адрес устройства ввода-вывода _____	

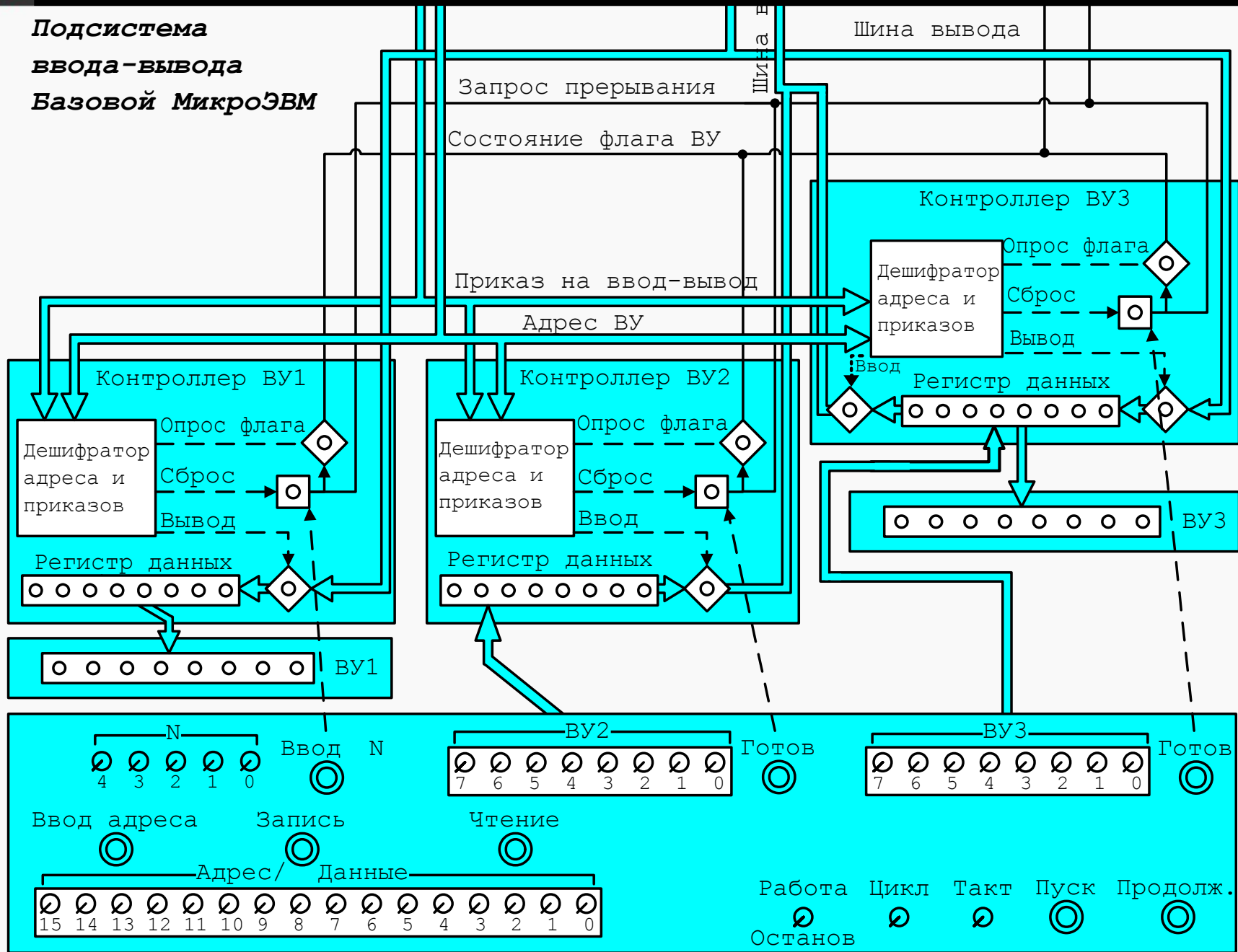
Device Controller





Basic I/O Devices of the BComp: ED1, ED2, ED3

Подсистема
ввода-вывода
Базовой МикроЭВМ



Waiting for Input Cycle

Enter two symbols from the input device

```

ORG      005
L8:      WORD      FFF8      ; -8 number of shifts
RES:     WORD      ?        ; Memory cell for the word "YES"
ORG      020      ; Address of the first instruction 020

BEGIN:   CLA
SPIN1:   TSF      2        ; Waiting for the first symbol
         BR      SPIN1    ;
         IN      2        ; Enter the first symbol
         CLF     2        ; Clear the flag. Able to enter the next symbol

RL:      ROL      ; Shift the first symbol
         ISZ     L8      ; High byte of the accumulator register
         BR      RL

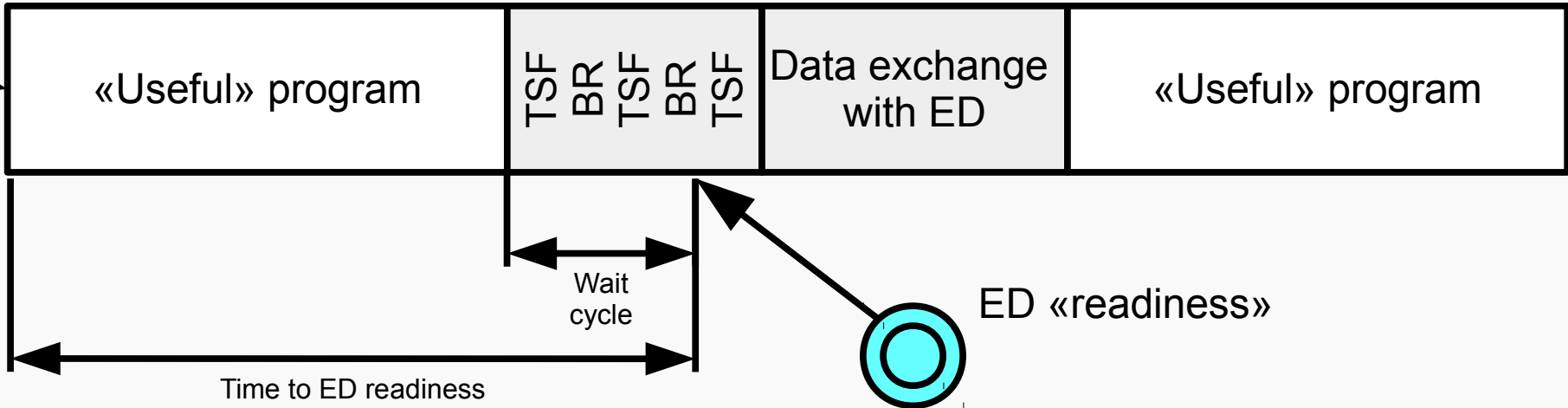
SPIN2:   TSF      2        ; Waiting for the second symbol
         BR      SPIN2
         IN      2        ; Enter the 2nd symbol to low 8 bits of the A
         CLF     2
         MOV     RES
         HLT

```

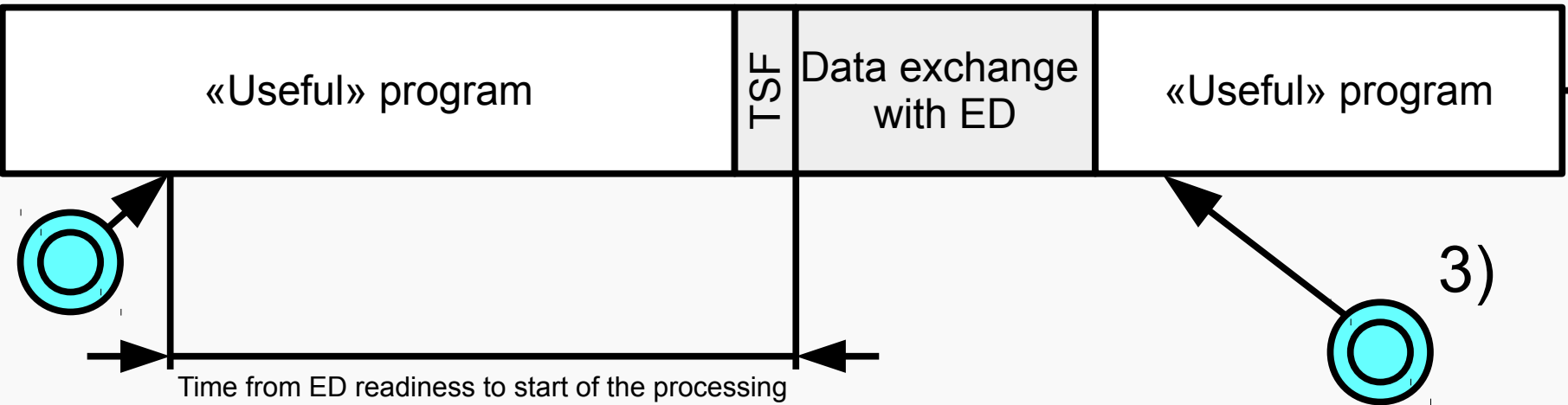
How many instruction cycles will BComp wait for the second symbol?

Time Costs

1)



2)

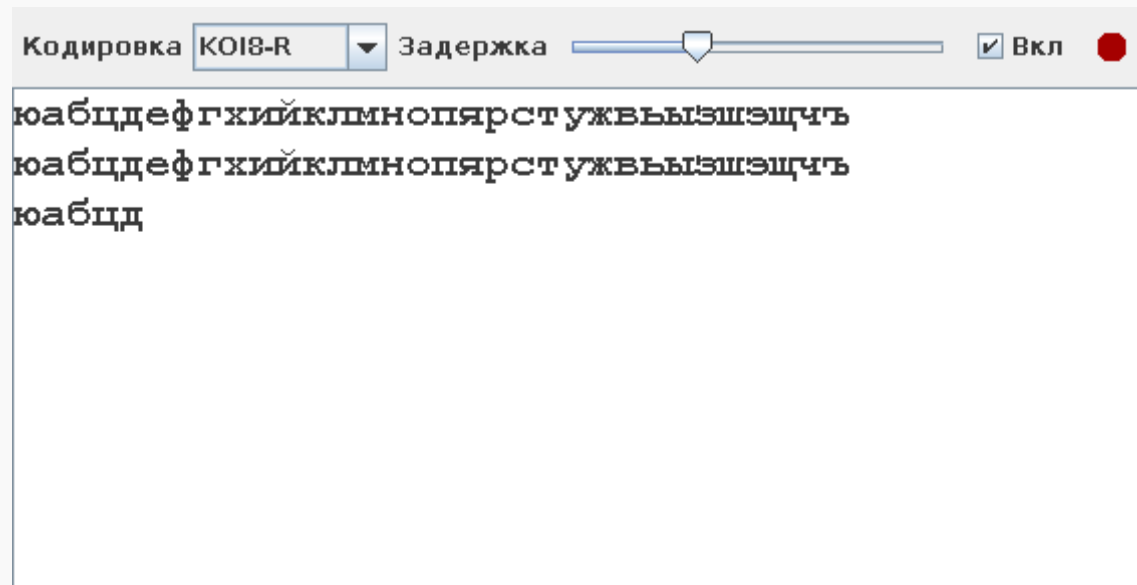


BComp ED0 Timer

- Sets ED readiness every $(100 * DR \text{ of ED content})$ ms:
 - Shifted fixed decimal radix point
- If DR of ED value == 0 readiness won't set
- Could be used for asynchronous data exchange (How?)

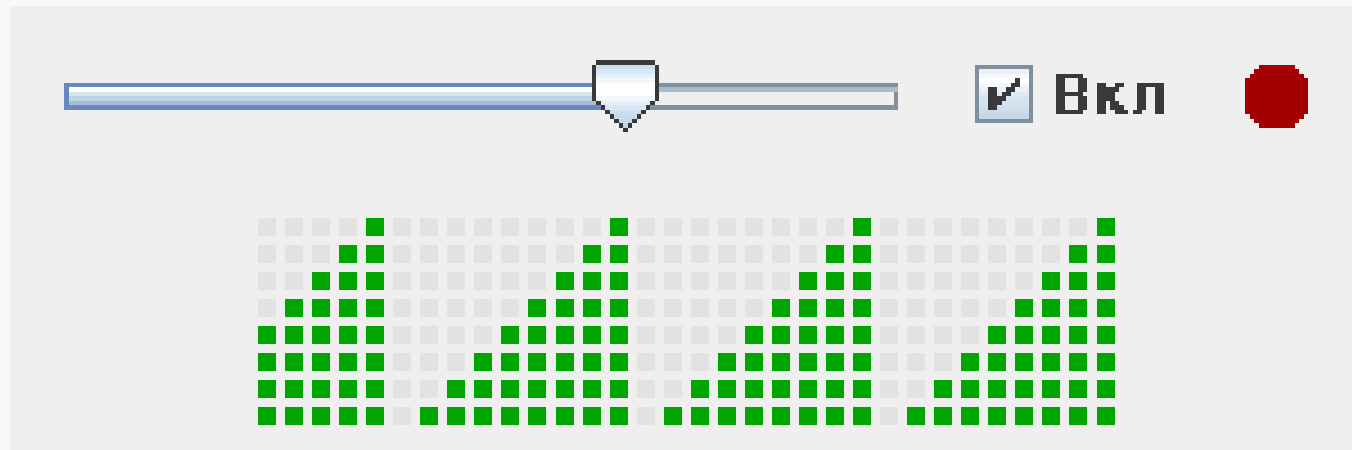
BComp ED4 Text Printer

- Prints symbols from DR of ED in specified encoding
- Adjustable delay (print time) from 100 ms to 10 s
- Line break by CR symbol (0A₁₆)
- NUL (0) — clear sheet
- Another symbols — unspecified behaviour



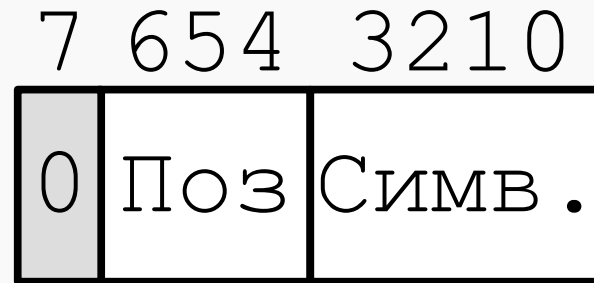
BComp ED5 Ticker

- Screen matrix size: 32x8
- Shifts left on every new value in DR of ED
- New value appears on the right side of the screen
- Low bit is on the bottom

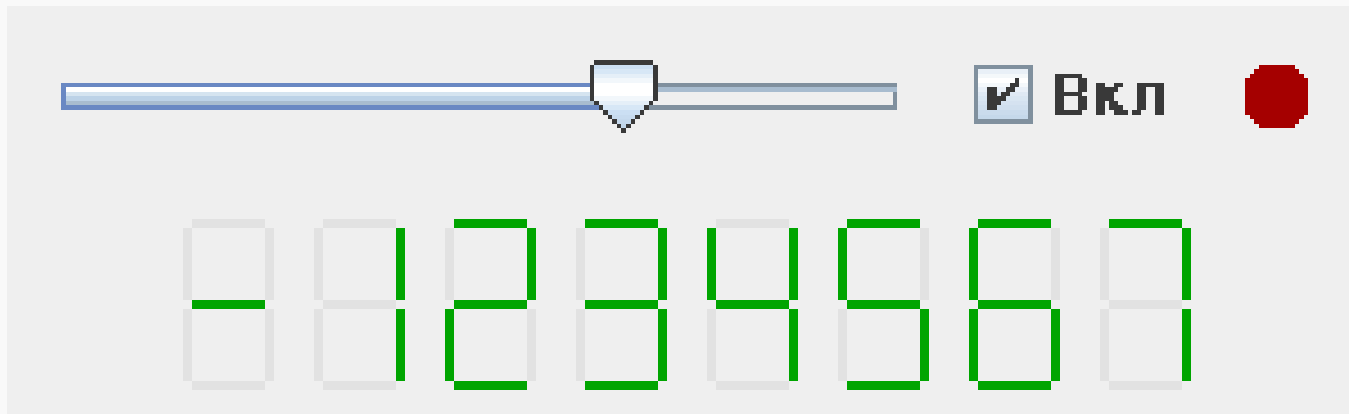


БЭВМ: ВУ-6 8-ми разрядный семисегментный индикатор

- Формат РДВУ:

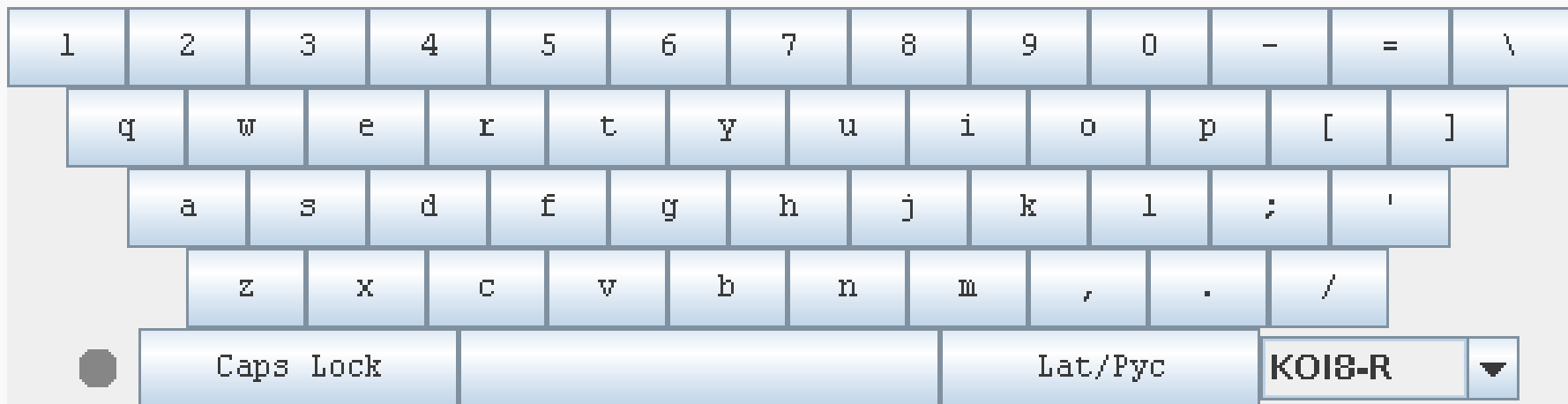


- Симв==(A_{16}) — установка в разряде знака «-»
- Симв==($B_{16}-F_{16}$) — сброс разряда



БЭВМ: ВУ-7 клавиатура

- Код нажатой клавиши в выбранной кодировке устанавливается в РДВУ
- Автоматически устанавливается готовность



БЭВМ: ВУ-8 Цифровая клавиатура

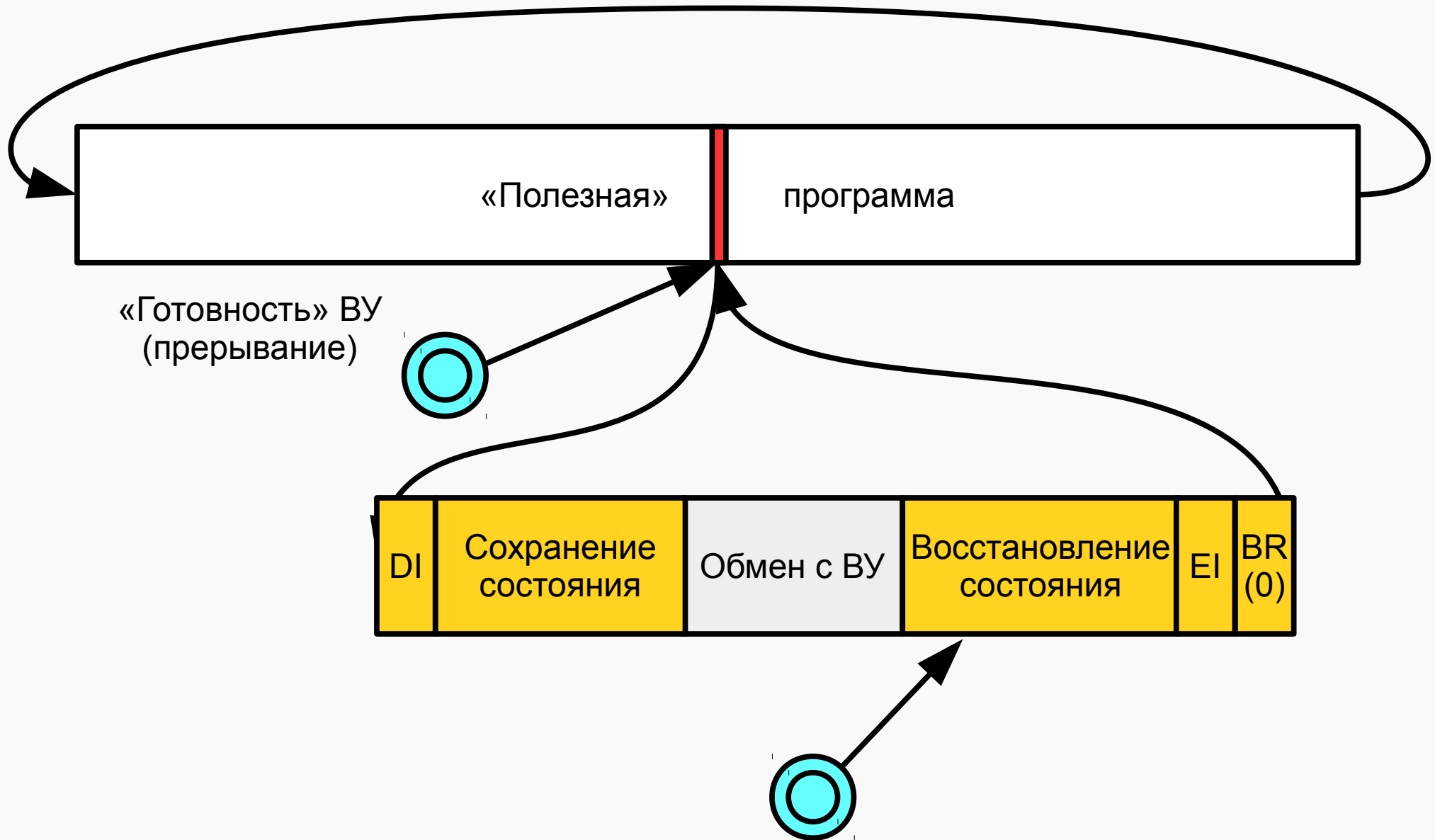
- При нажатии клавиши ее код помещается в РДВУ
- Клавиша 0-9 код 0-9
- Клавиша «-» код А
- «+» код В
- «/» код С
- «*» код D
- «.» код E
- «=» код F

7	8	9	/
4	5	6	*
1	2	3	-
0	.	=	+

БЭВМ: ВУ-9 БЭВМ

- Два стандартных контроллера ввода-вывода (как ВУ-3) связаны между собой
- IN — читает информацию из РДВУ контроллера
- OUT — записывает данные в РДВУ **обоих** контроллеров
- CLF одного контроллера приводит к установке флага связанного контроллера

Инициация обмена по прерыванию

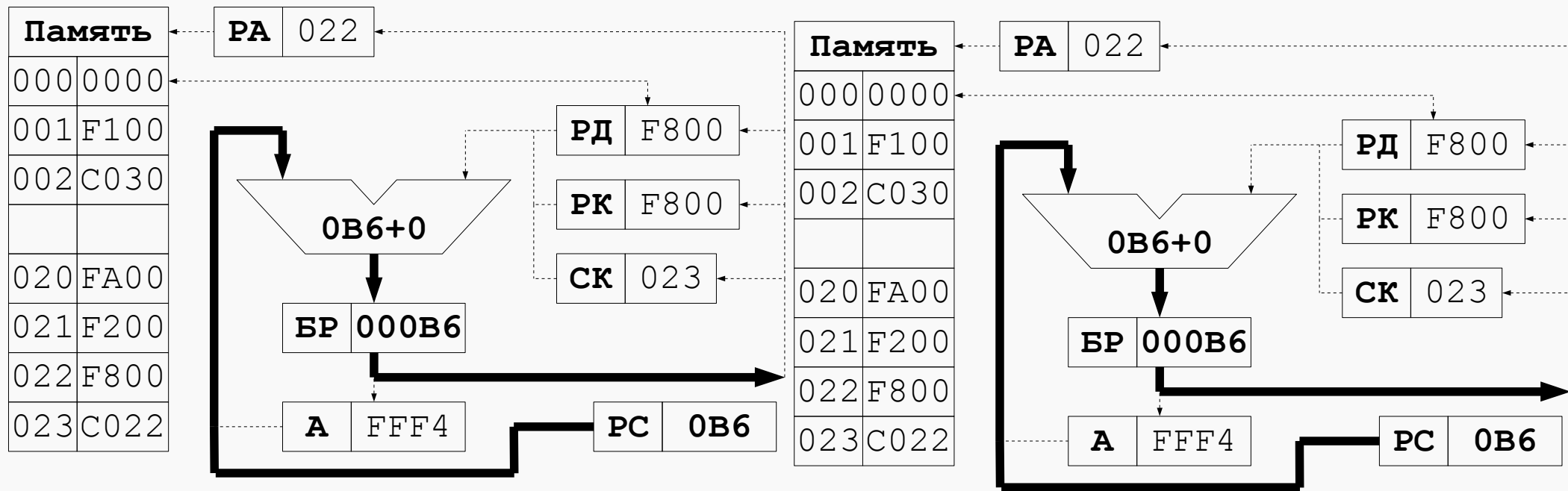


Регистр состояния и команды

Бит	Мнем.	Содержимое
0	C	Перенос
1	Z	Нуль
2	N	Знак
3	0	0 - используется для организации безусловных переходов в МПУ
4	EI	Разрешение прерываний
5	I	Прерывание (логическое "И" шины запроса на прерывание и бита 4 РС - "разрешение прерываний")
6	F	Состояние ВУ (Ф)
7	W	Состояние тумблеров РАБОТА/ОСТАНОВ (1 - РАБОТА)
8	P	Программа

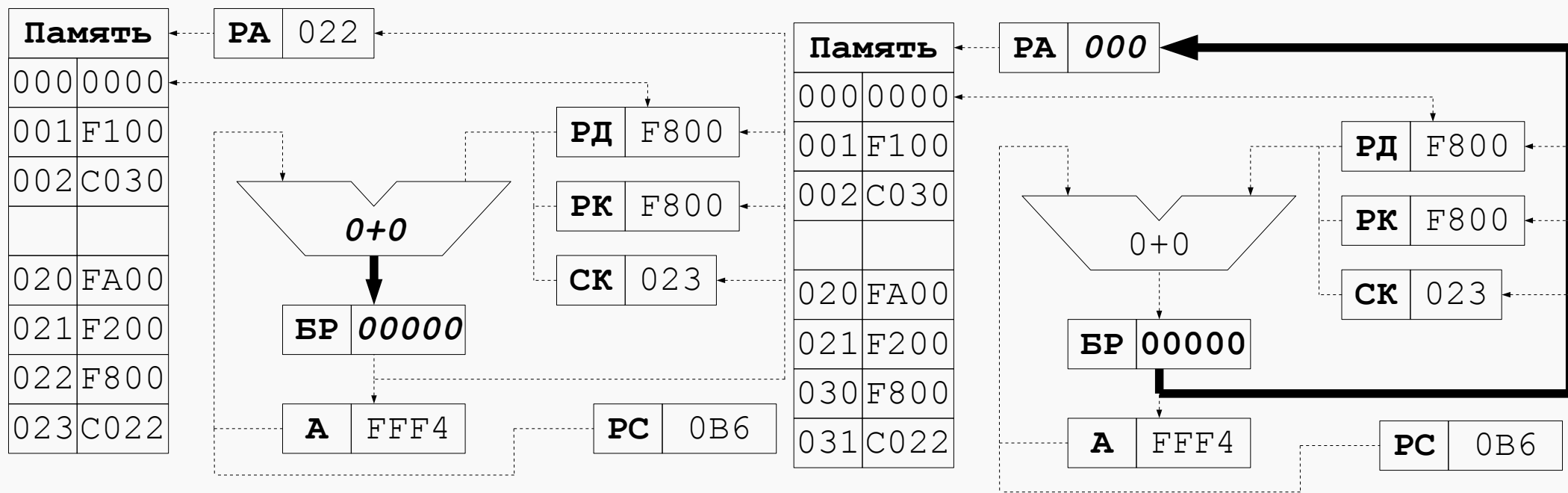
Наименование	Мнемон.	Код	Описание
Разрешение прерываний	EI	FA00	Установка бита 4 (разрешение прерываний)
Запрещение прерываний	DI	FB00	Сброс в 0 бита 4 (разрешение прерываний)
Возврат из прерывания	BR (0)	C800	Переход по адресу, сохраненному в яч. 0

Цикл прерывания



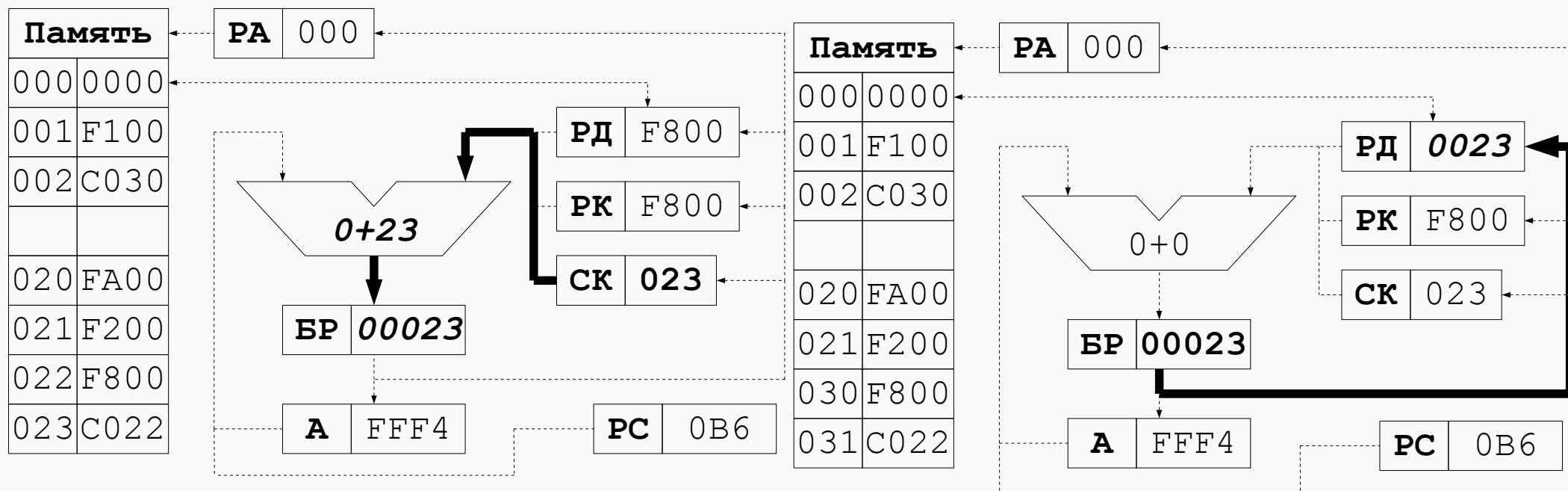
- После выполнения команды управление передается циклу прерывания
- Если бит 7 РС (Работа/Останов) == 0 то останов БЭВМ
- Если бит 5 РС (Прерывание) == 0 то переход к следующей команде (циклу выборки команды)

Цикл прерывания



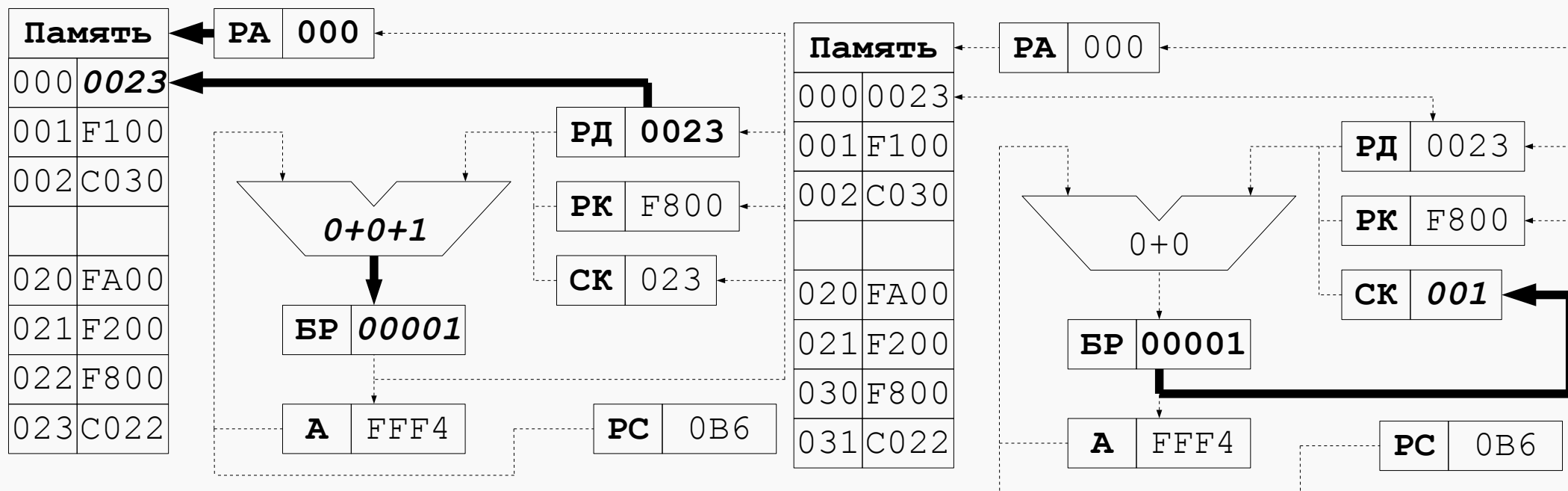
- 0 — (ячейка сохранения адреса возврата) записывается в RA

Цикл прерывания



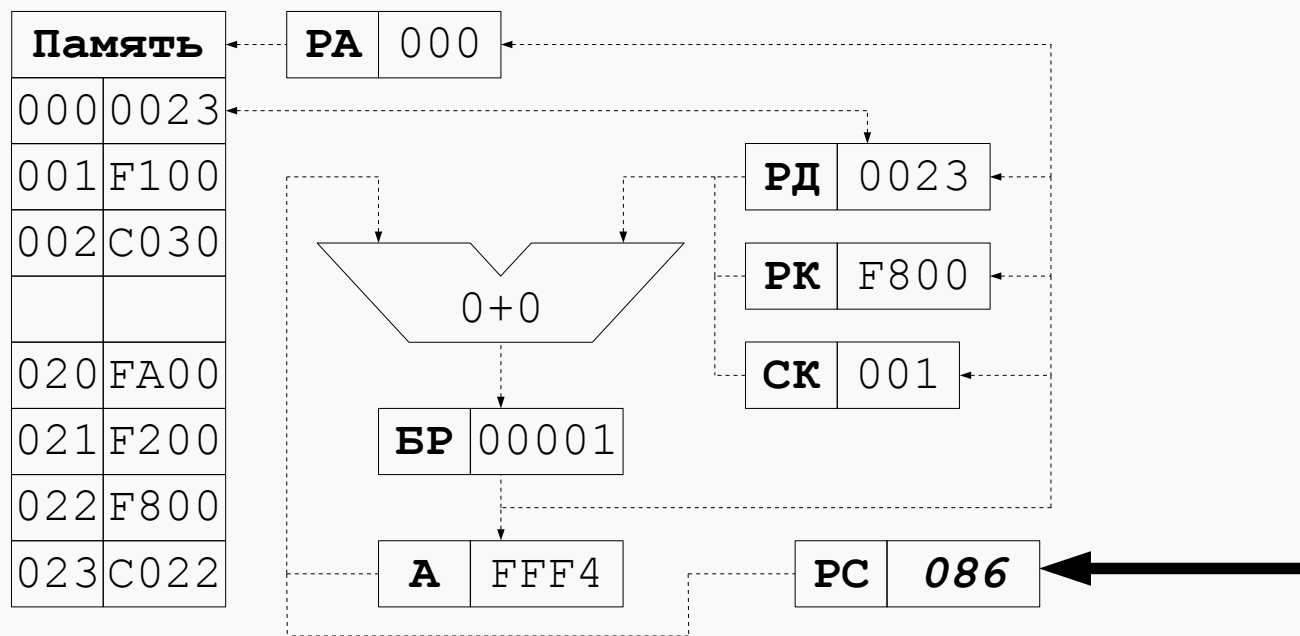
- Содержимое СК через БР переписывается в РД

Цикл прерывания



- Содержимое РД записывается по адресу 0, тем самым обеспечивая возможность возврата из прерывания
- 1 записывается в БР и далее в СК, определяя первую исполняемую команду

Цикл прерывания



- Производится запрещение прерываний в РС, сбрасывается бит 4 «разрешение прерываний», что приводит к сбросу бита 5 «прерывание» (В->8)
- После этого переход к циклу выборки команды из ячейки 1, т.е. обработке прерывания

После каких команд нет цикла прерывания?

- DI
 - Нет необходимости. Прерывания запрещены
- HLT
 - Нет необходимости. Программа остановлена
- EI
 - Приведет к зацикливанию обработчика прерывания
Завершение обработчика:
EI
BR (0)

Обработка прерываний: Основная программа

Обработка прерываний

Готовность ВУ1: 2*А→РДВУ1, Готовность ВУ3: РДВУ3→ яч.29

	ORG 000	
RET:	WORD ?	; Ячейка адреса возврата
	NOP	; Ячейка отладочной точки
		; остановка (NOP/HLT)
	BR INT	; Переход к обработке прерываний
	ORG 020	; Основная программа
BEGIN:	EI	; Установка состояния разрешения прерывания
	CLA	; Первоначальная очистка аккумулятора
LOOP:	INC	; Цикл для наращивания
	BR LOOP	; содержимого аккумулятора
	ORG 029	; Ячейка для хранения кодов,
IO3:	WORD ?	; поступающих с ВУ-3

Обработка прерываний: Сохранение и восстановление

Готовность ВУ1: 2*А→РДВУ1, Готовность ВУ3: РДВУ3→ яч.29

```
INT:   ORG    030
        MOV    SAVED_A    ; Сохранение содержимого А и С
        ROL
        MOV    SAVED_C
```

Основная программа обработки прерывания
(см. следующий слайд)

```
RESTORE: NOP          ; отладочная точка останова (NOP/HLT)
        CLA           ; Восстановление содержимого С и А
        ADD SAVED_C
        ROR
        CLA
        CMA
        AND SAVED_A
        EI           ; Возобновление состояния
                   ; разрешения прерывания
        BR (RET)     ; Возврат из обработки прерывания
SAVED_A: WORD ?
SAVED_C: WORD ?
```

Обработка прерываний: Прерывание

Готовность ВУ1: 2*А→РДВУ1, Готовность ВУ3: РДВУ3→ яч.29

```

                TSF 3           ; Опрос флага ВУ-3
                BR CHECK1      ; Если сброшен → к опросу флага ВУ-1
                BR READY3      ; Переход на ввод данных из ВУ-3
CHECK1:         TSF 1           ; Опрос флага ВУ-1
                BR READY2      ; Если сброшен → к опросу флага ВУ-2
                BR READY1      ; Переход на вывод данных в ВУ-1
READY3:        CLA            ; Ввод данных из ВУ-3
                IN 3
                CLF 3          ; Сброс флага ВУ-3
                MOV IO3        ; Пересылка значения в ячейку 29
                BR RESTORE
READY1:        CLA
                ADD SAVED_A
                ROL
                OUT 1          ; Вывод на ВУ-1 8-ми
                CLF 1          ; Сброс флага ВУ-1
                BR RESTORE
READY2:        CLF 2
```

ВВОД-ВЫВОД



8



Многоуровневая ЭВМ

Уровень программных систем (специальный язык)

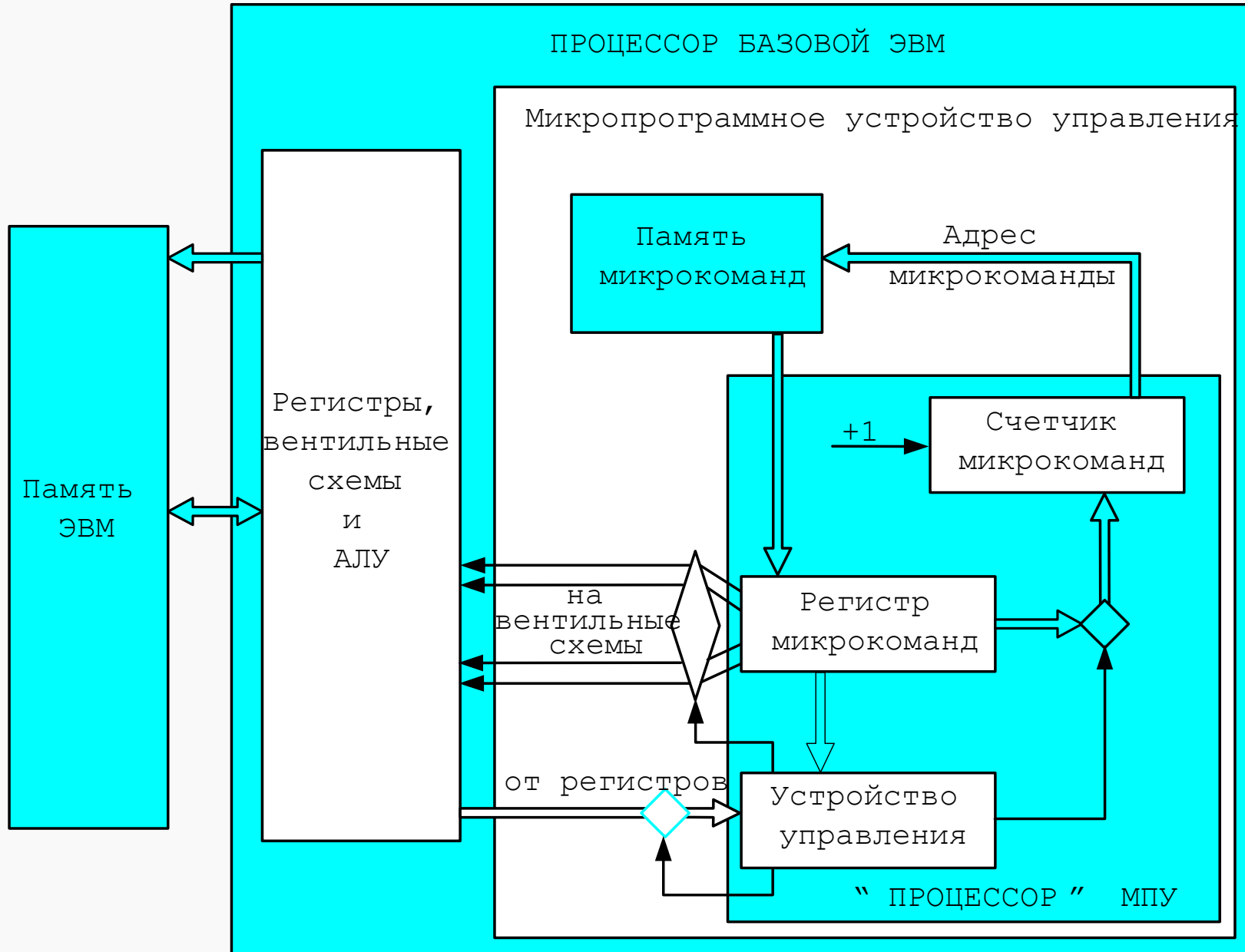
Уровень проблемно-ориентированных задач
(один из алгоритмических языков)

Язык Ассемблера

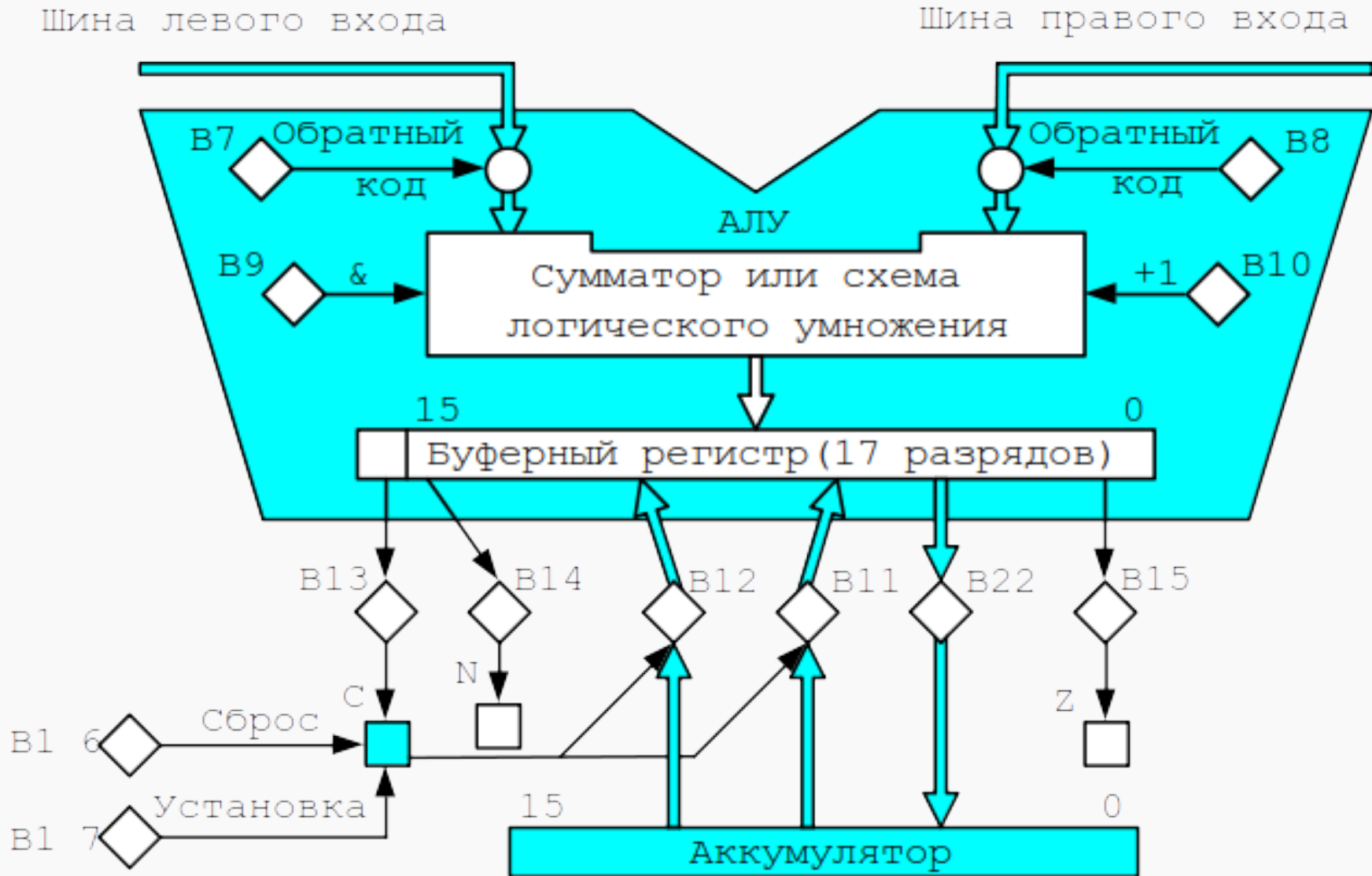
Машинные команды

Микропрограммный
Уровень

МПУ

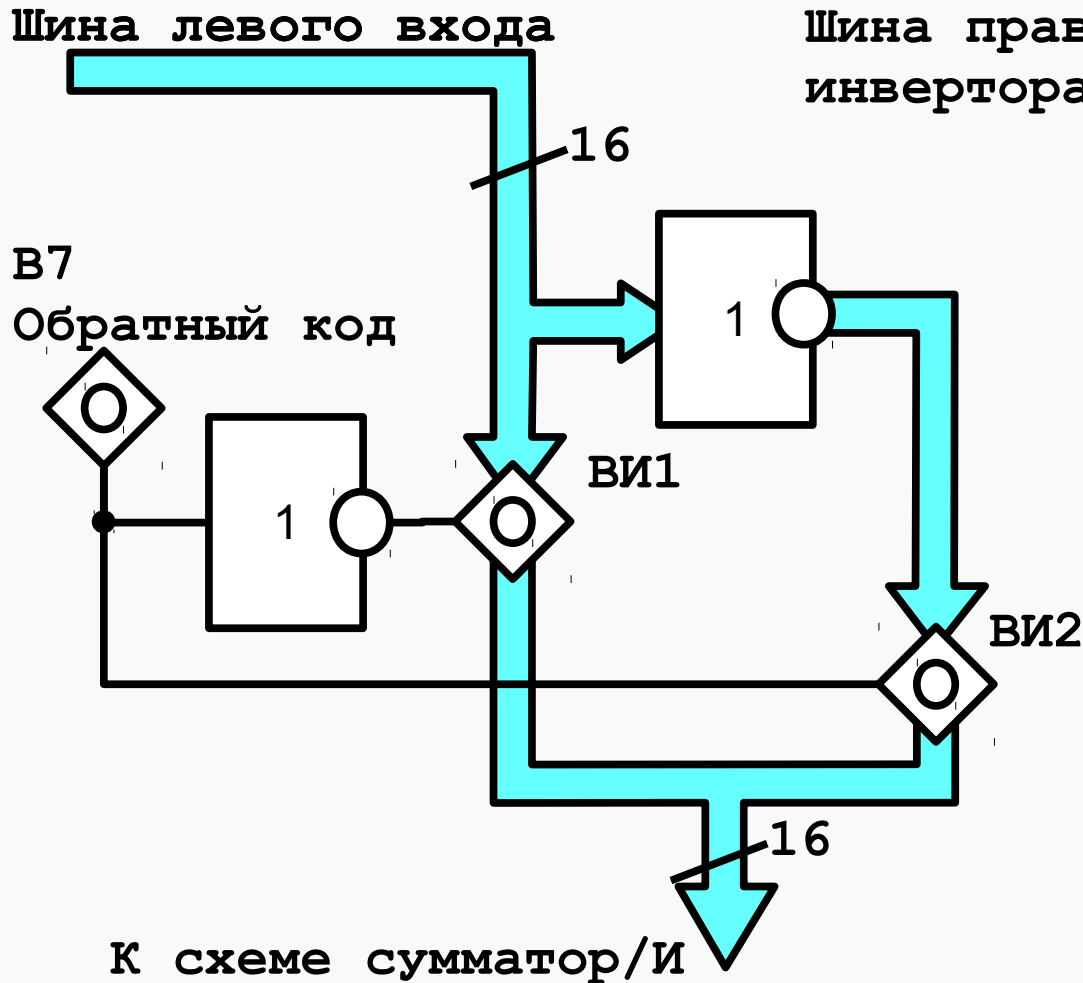


АЛУ

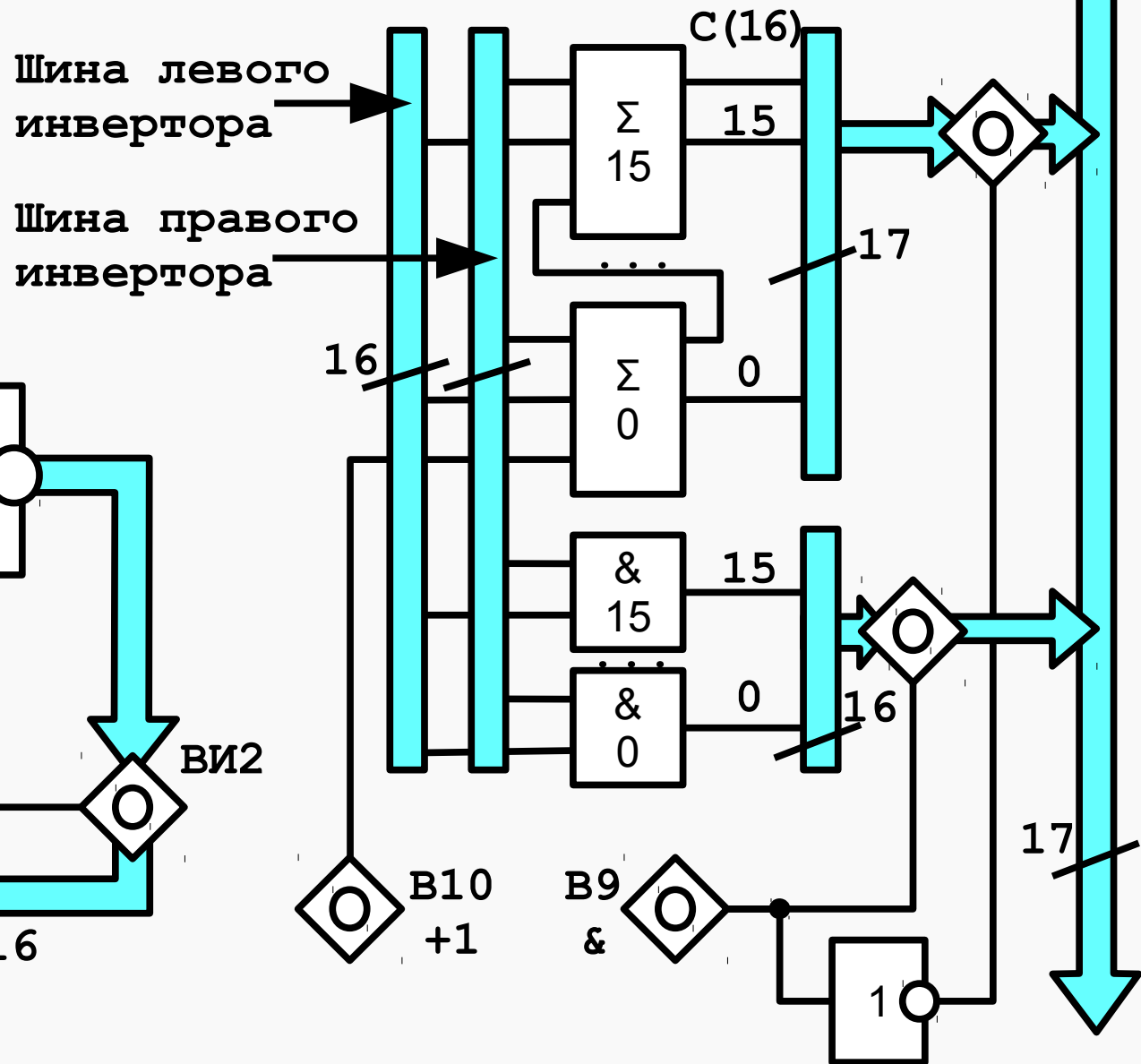


АЛУ: Обратный код, сумматор и логическое «И»

Схема обратного кода

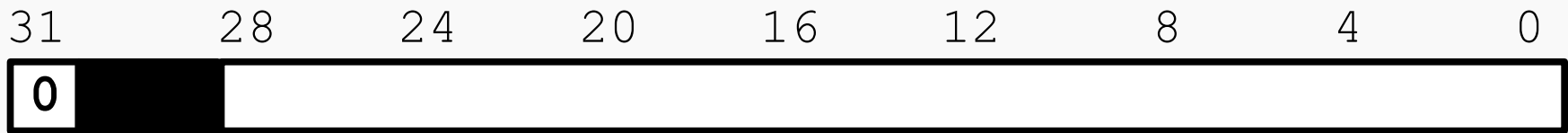


Сумматор и лог. «И»



Горизонтальные микрокоманды

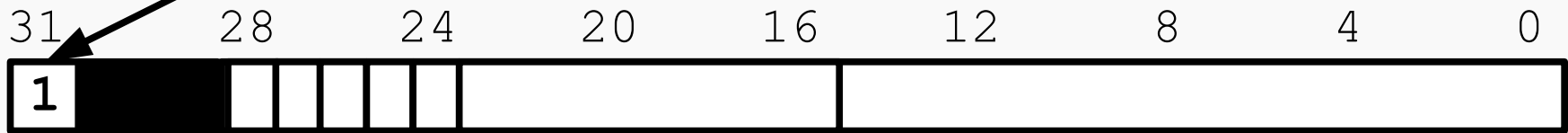
Операционная микрокоманда



Биты управления отдельными вентиляными схемами

Код операции

Управляющая микрокоманда



Адрес перехода Поле выбора проверяемого бита

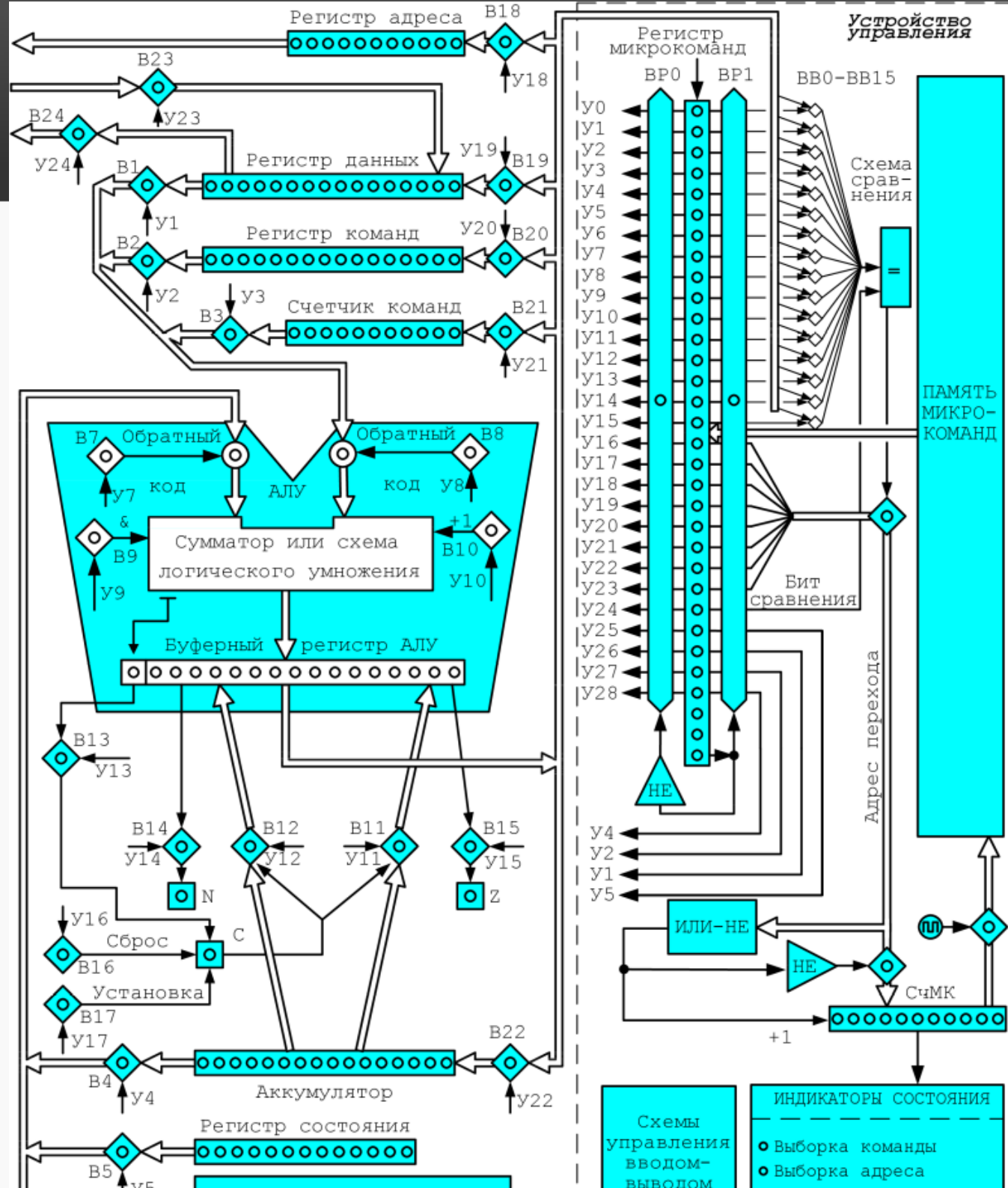
Однобитовое поле сравнения



Поле выбора проверяемого бита

МПУ

ИТМО ВТ



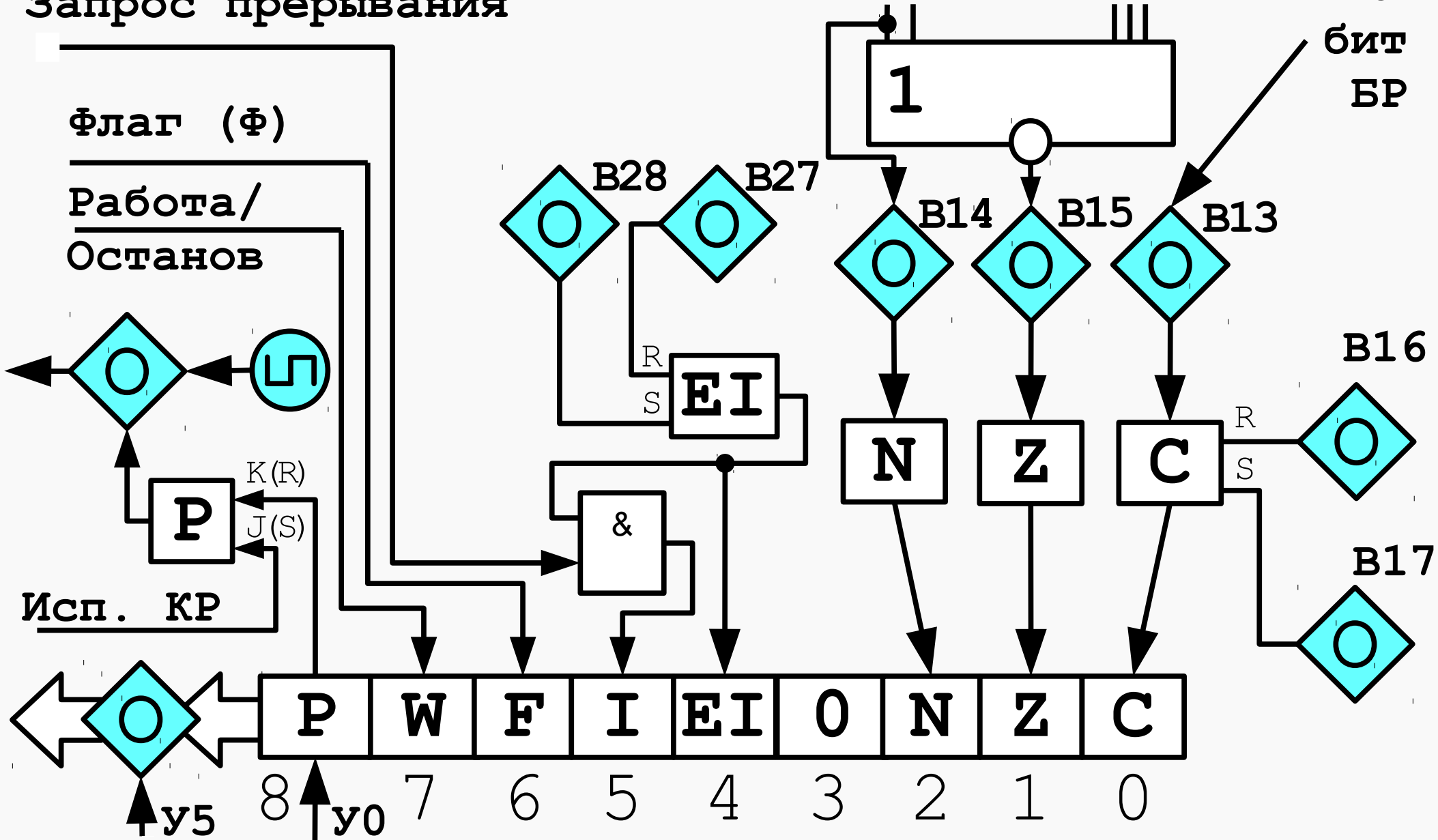
Схемы управления вводом-выводом	ИНДИКАТОРЫ СОСТОЯНИЯ
	● Выборка команды
	● Выборка адреса

Регистр состояния

Запрос прерывания

биты 15...0 из БР

16 бит БР



Вентильные схемы

- Входные сигналы на входы АЛУ
 - В1-В3 — левый вход АЛУ
 - В4-В6 — правый вход АЛУ
- Арифметические операции и сдвиги
 - В7-В12
- Установка признаков
 - В13-В22
- Работа с памятью
 - В23-В24
- Организация ввода-вывода информации
 - В25-В28
- Останов ЭВМ — В0

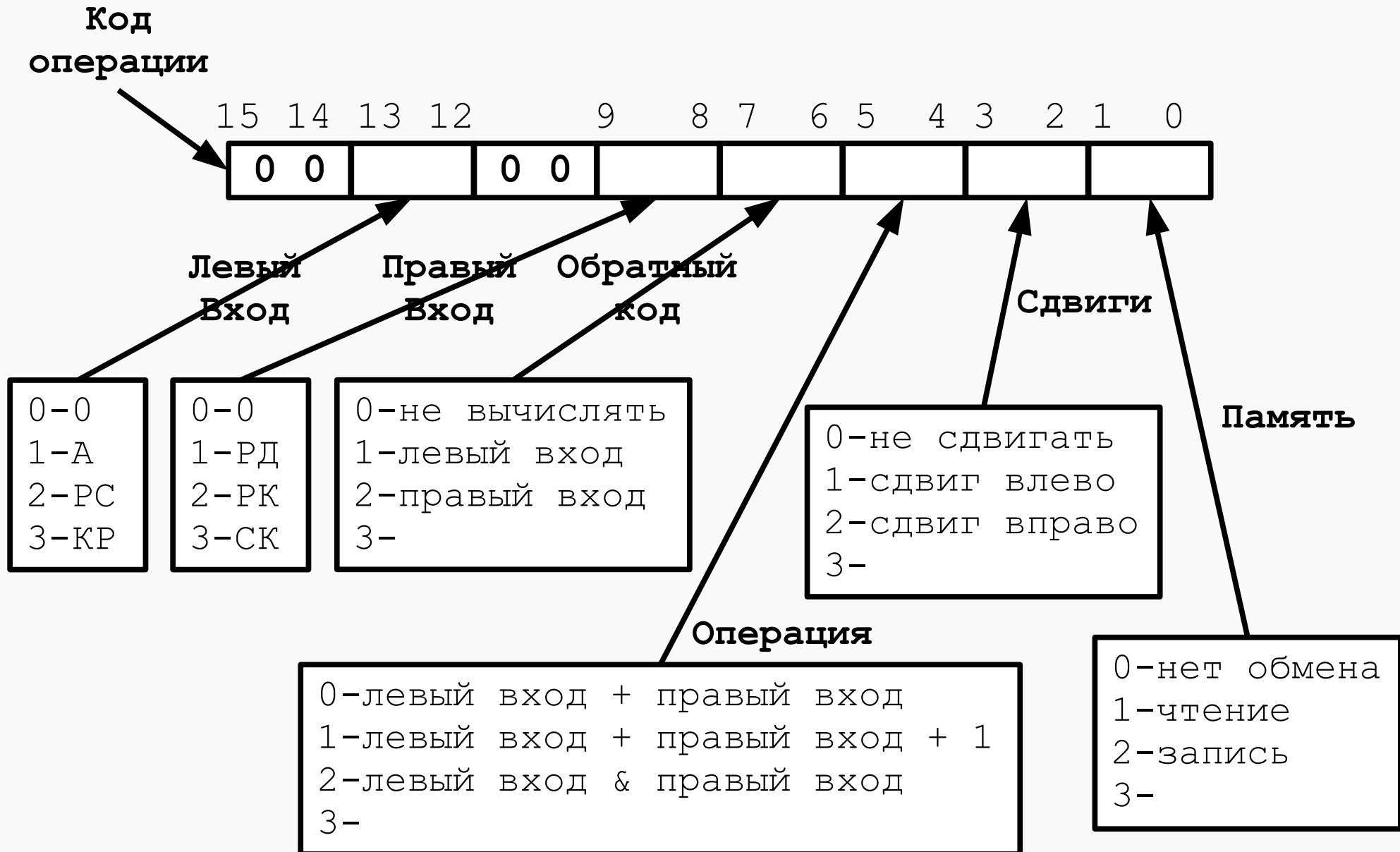
Интерпретатор БЭВМ

- 256 ячеек для хранения микрокоманд, включая резерв
- Содержит вертикальные микрокоманды (см. далее)
- Оформлено в виде таблицы (см. Методу!)

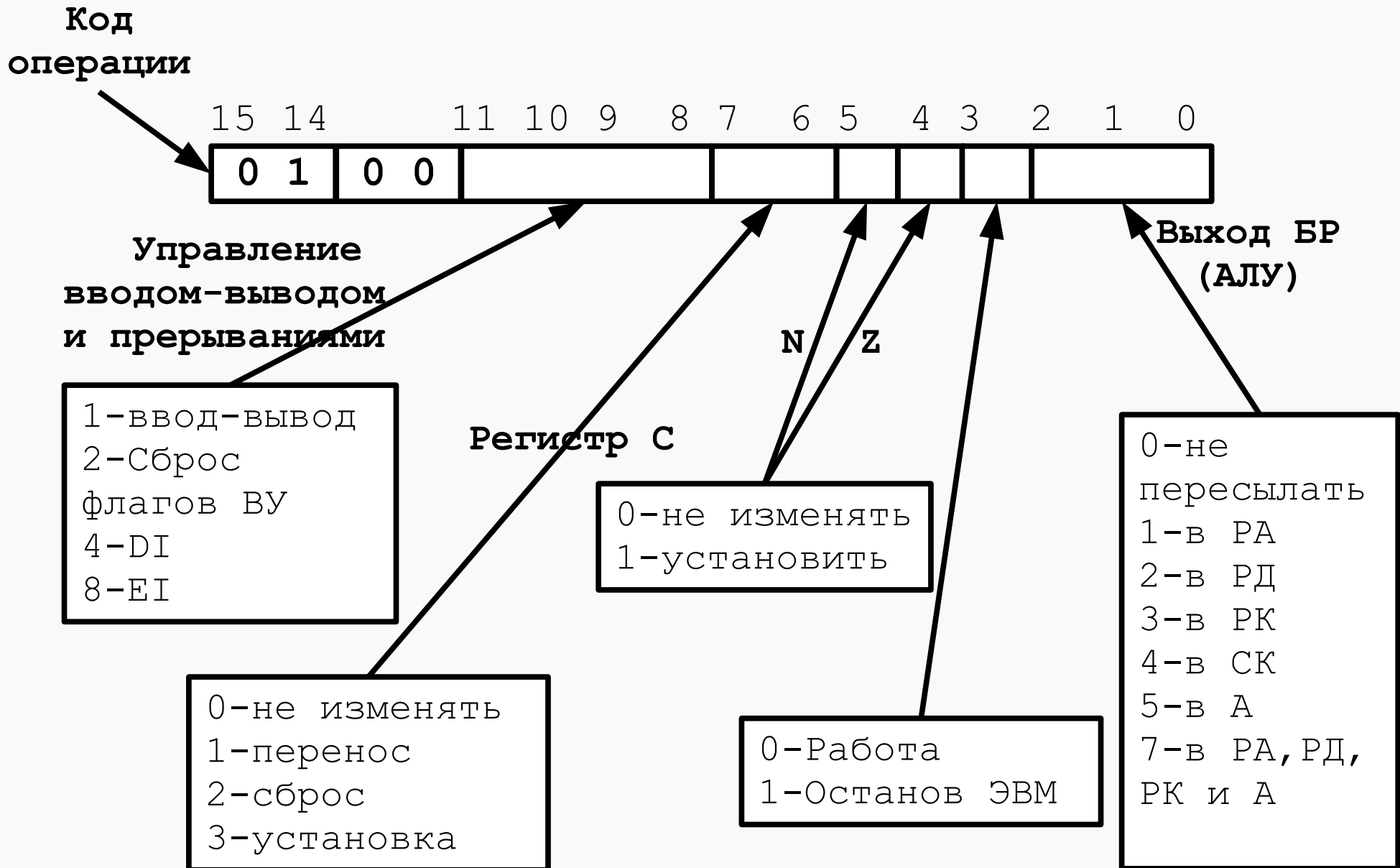
Адрес	Микрокоманды		Комментарии	
	Горизонтальная	Вертикальная	Метка	Действие
01	0000 0008	0300	НАЧ	СК→БР
02	0004 0000	4001		БР→РА
03	0080 0408	0311		ОП (РА) →РД, СК+1→БР

- Цикл выборки команд
- Цикл выборки адреса операнда
- Цикл исполнения адресных команд
 - Декодирование адресных команд
 - Исполнение адресных команд
- Декодирование и исполнение безадресных команд
- Декодирование и исполнение команд ввода-вывода
- Цикл прерывания
- Пультовые операции
- Свободные ячейки для:
 - Арифметической команды
 - Команды перехода
 - Безадресной команды

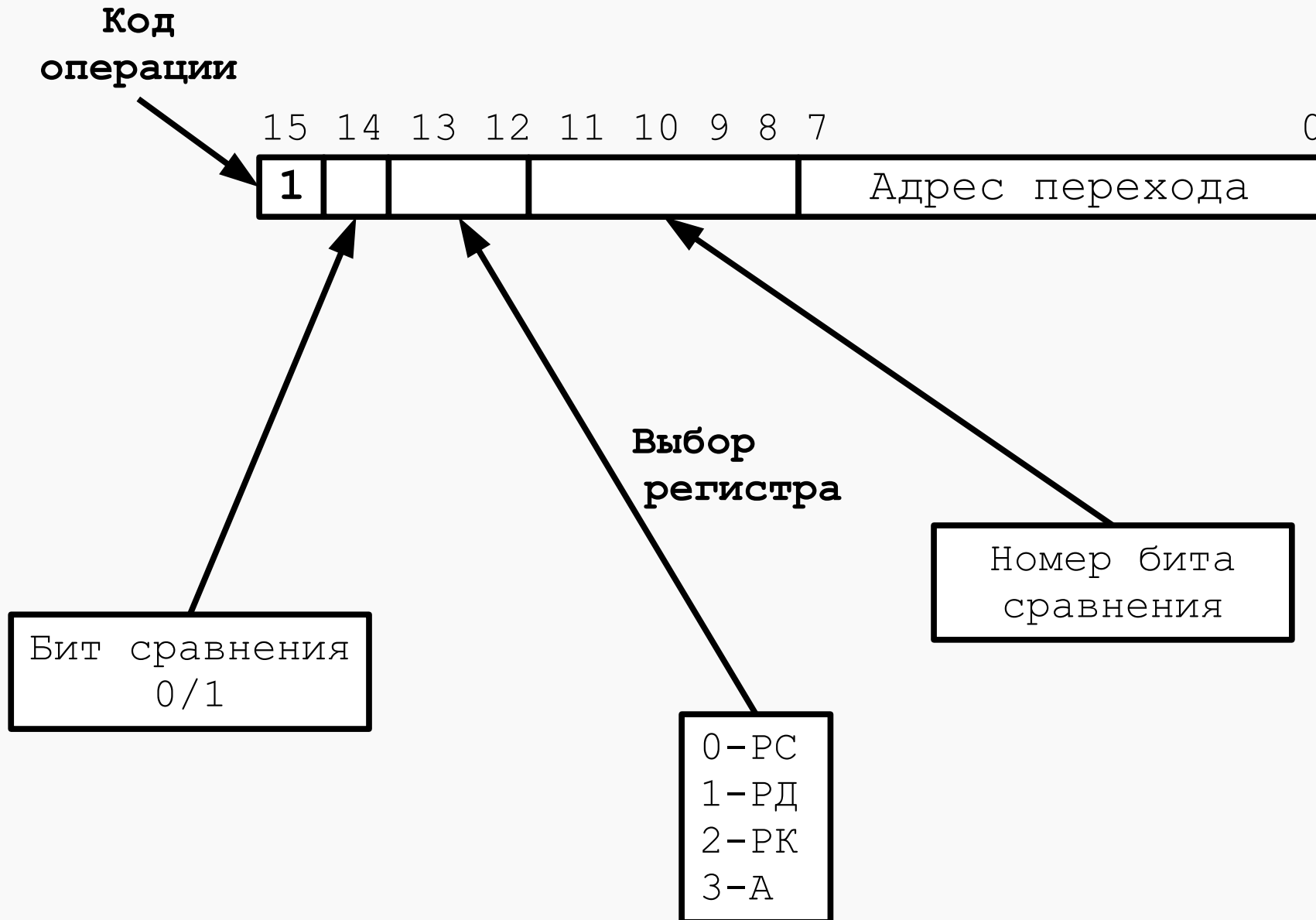
Вертикальные микрокоманды: ОМКО



Вертикальные микрокоманды: ОМК1

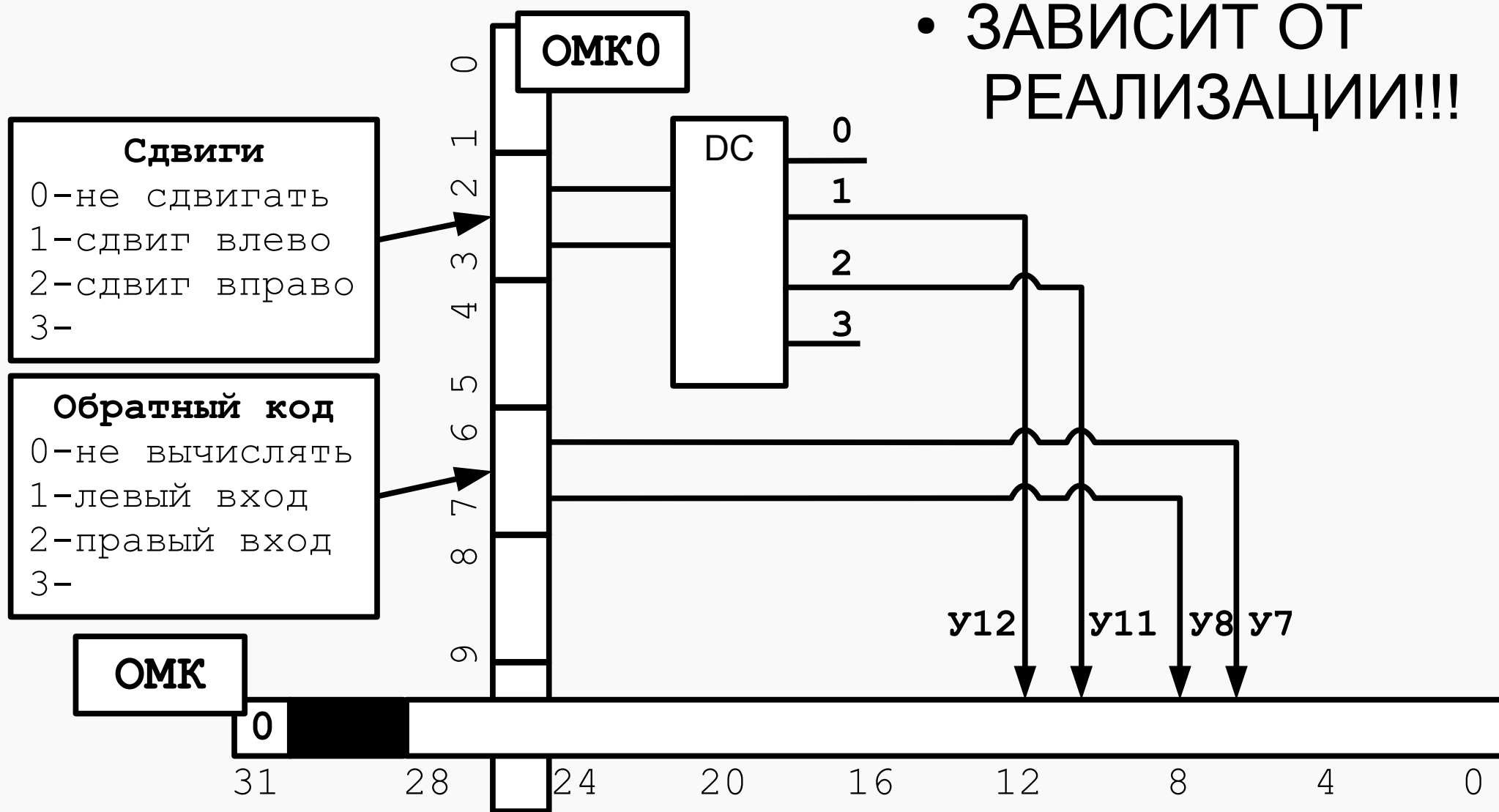


Вертикальные микрокоманды: УМК



Преобразование вертикальных микрокоманд в горизонтальные

- Задача — преобразовать коды в управляющие сигналы на вентилях



- ЗАВИСИТ ОТ РЕАЛИЗАЦИИ!!!



ИТМО ВТ

С БЭВМ — все!!!!





История и Архитектуры ЭВМ

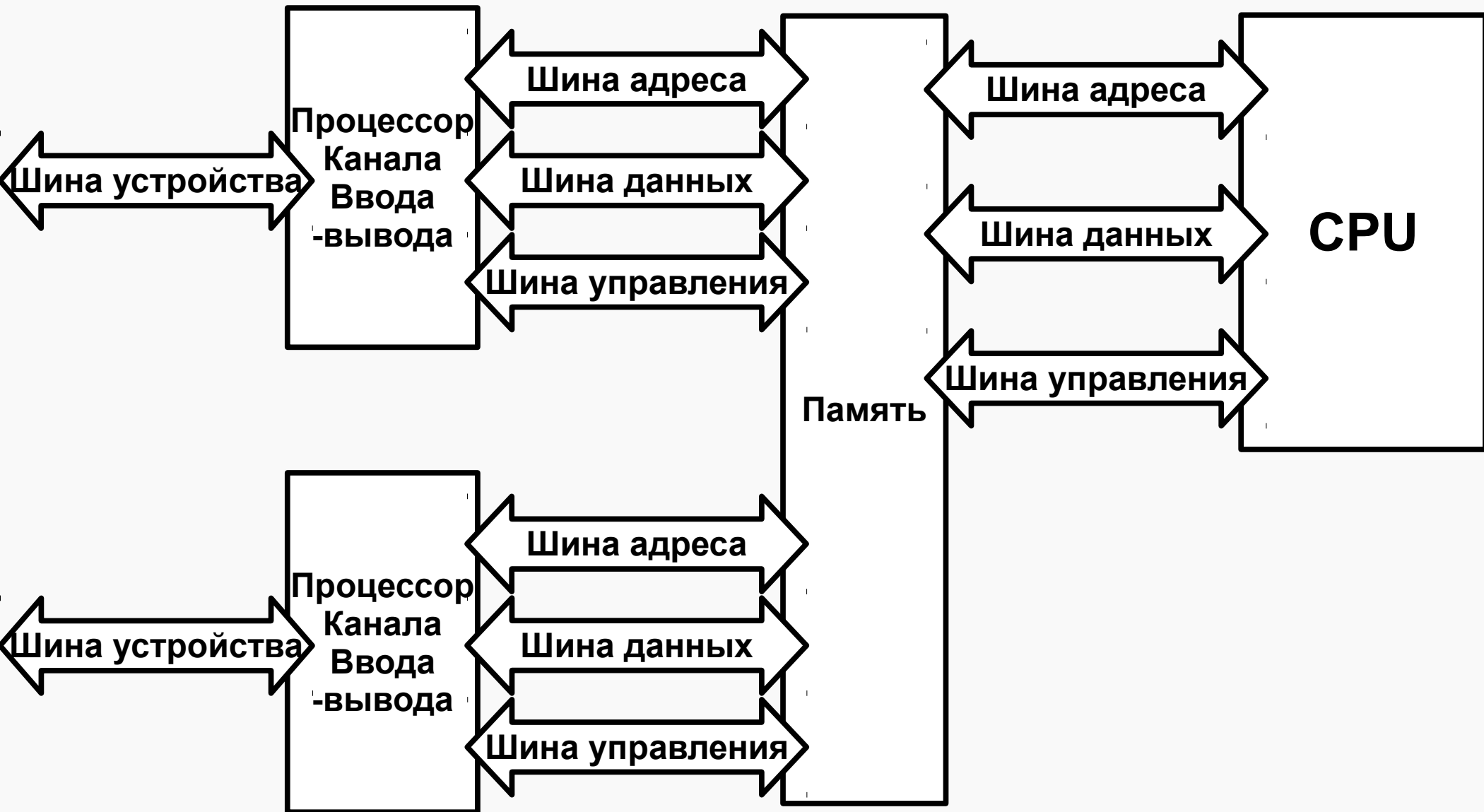
9

- Нулевое поколение — механические компьютеры (1642–1945)
 - Налоговый сумматор (Паскаль), калькулятор на 4 действия (Лейбниц)
- Первое поколение — электронные лампы (1945–1955)
 - COLOSSUS (1943, Тьюринг), ENIAC (1946, Моушли), IAS (1951, фон Нейман)
- Второе поколение — транзисторы (1955–1965)
 - TX-0 (1955, МТИ), PDP-1 (1961, DEC), PDP-8, 7090 (IBM), 6600 (1964, CDC)
- Третье поколение — интегральные схемы (1965–1980)
 - Семейство System/360 (1965, IBM), PDP-11 (1970, DEC)
- Четвертое поколение — сверхбольшие интегральные схемы (1980–?)
 - IBM PC (1981), Apple, Intel, IBM, Dec, Compaq, HP, Sun...
- Пятое поколение — небольшие и «невидимые» компьютеры (1989-?)

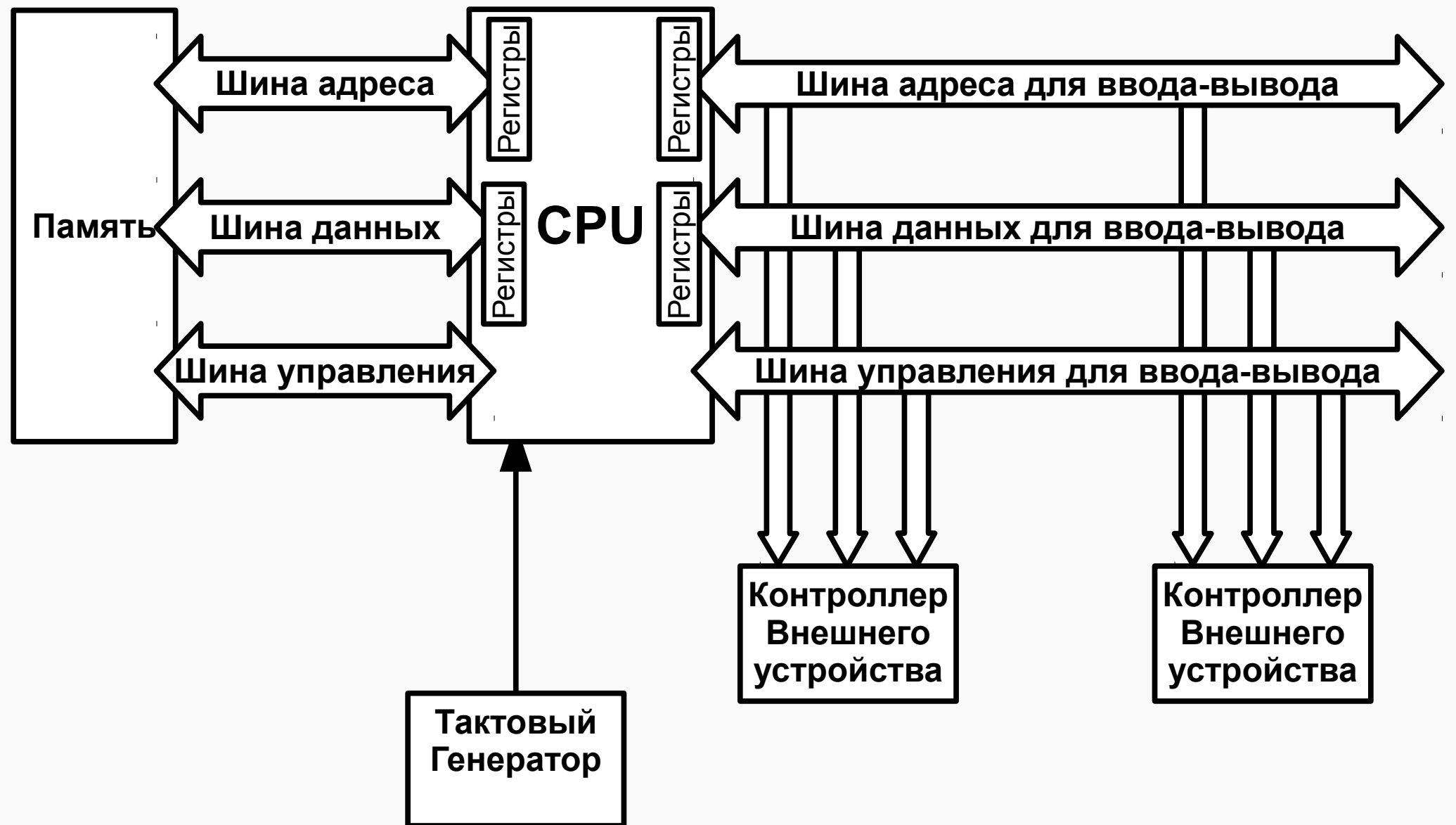
История в СССР/России

- Первое поколение — электронные лампы
 - Лебедев, 1950, МЭСМ
 - БЭСМ, 1953, БЭСМ — 10000 оп/с, 53КВТ.
- Второе поколение — транзисторы
 - 5Э926, 1964, самодиагностика, горячая замена, 500000 оп/с
 - БЭСМ-6, 1965 год, +ковейерная обработка, удаленное управление по телеф. Линиями
- Третье поколение — интегральные схемы
 - Директива «Ряд», 1968 год, клонирование S/360, 1971 год — ЕС ЭВМ
 - Клоны PDP-11
- Четвертое поколение — сверхбольшие интегральные схемы
 - Эльбрус — разработка по настоящее время

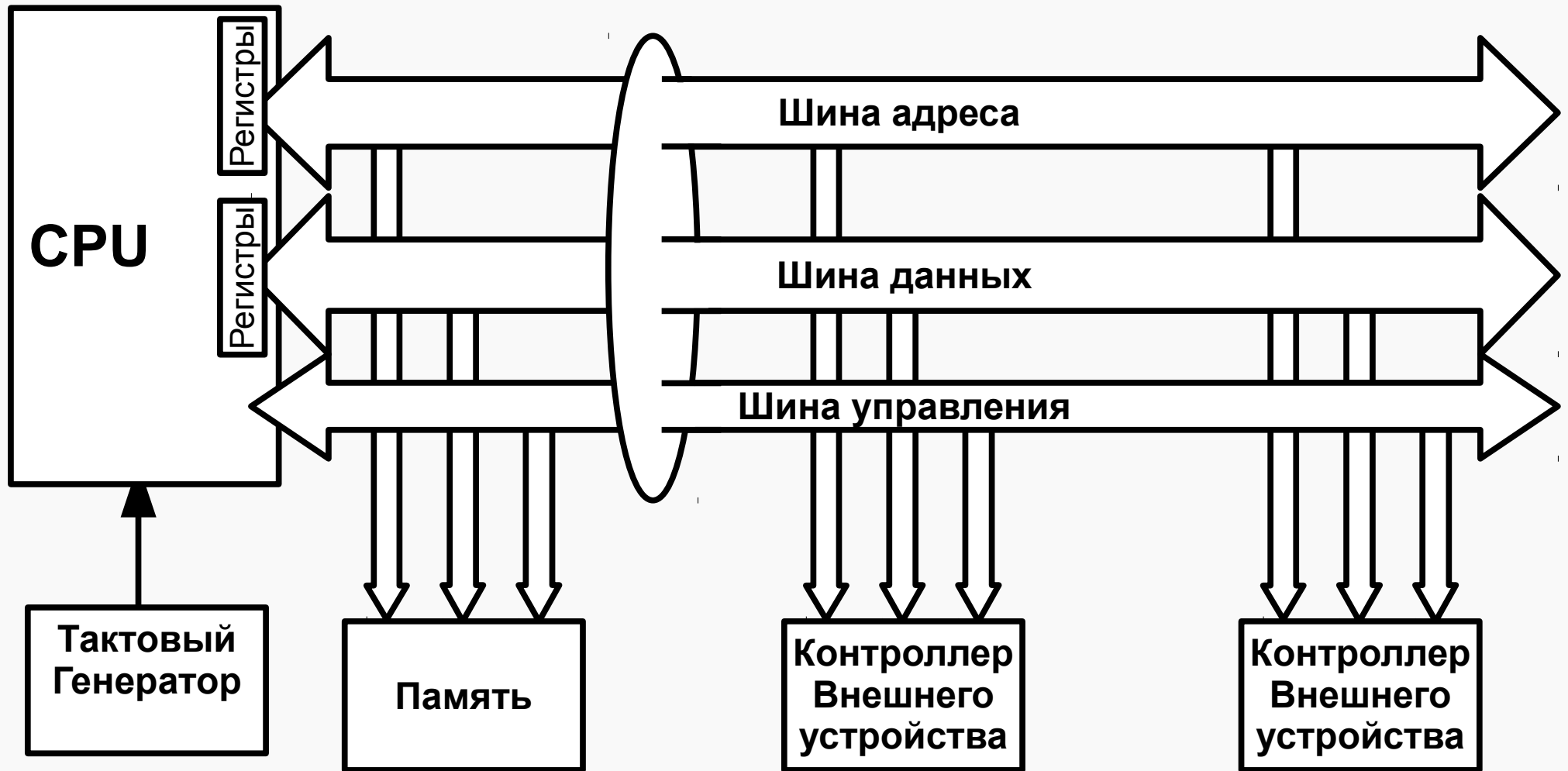
Канальная организация



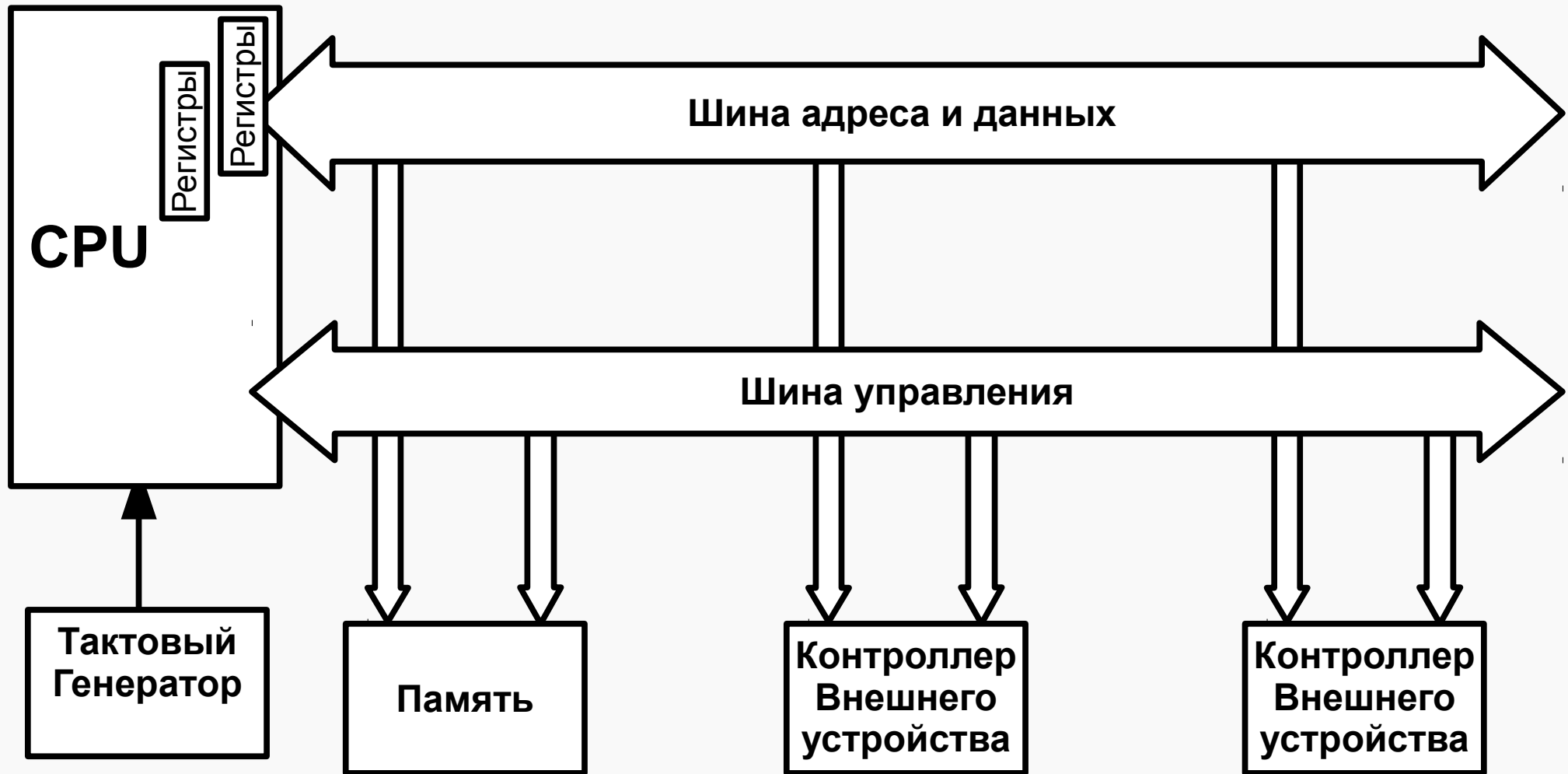
Раздельные шины



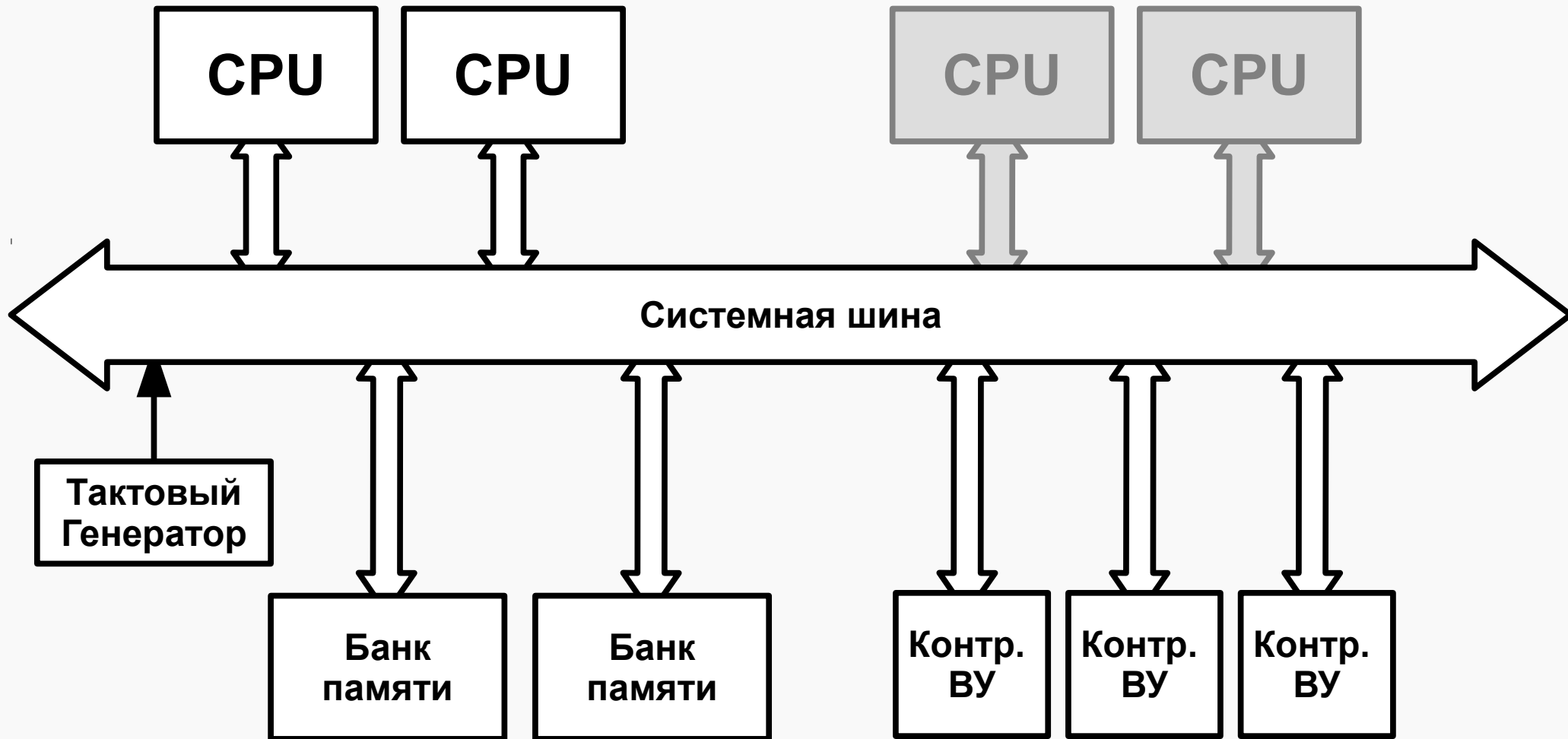
Общие шины



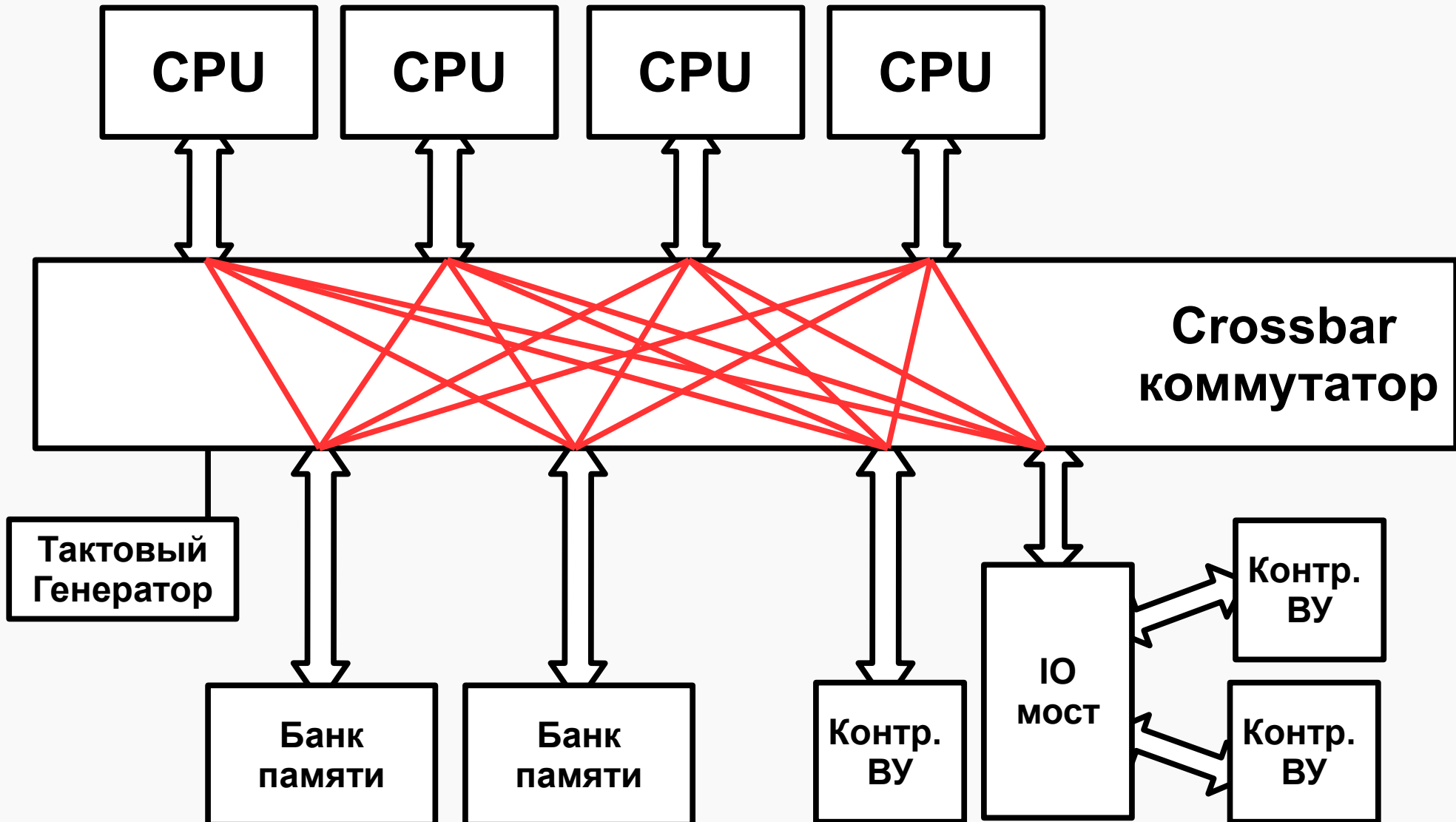
Мультиплексирование шин



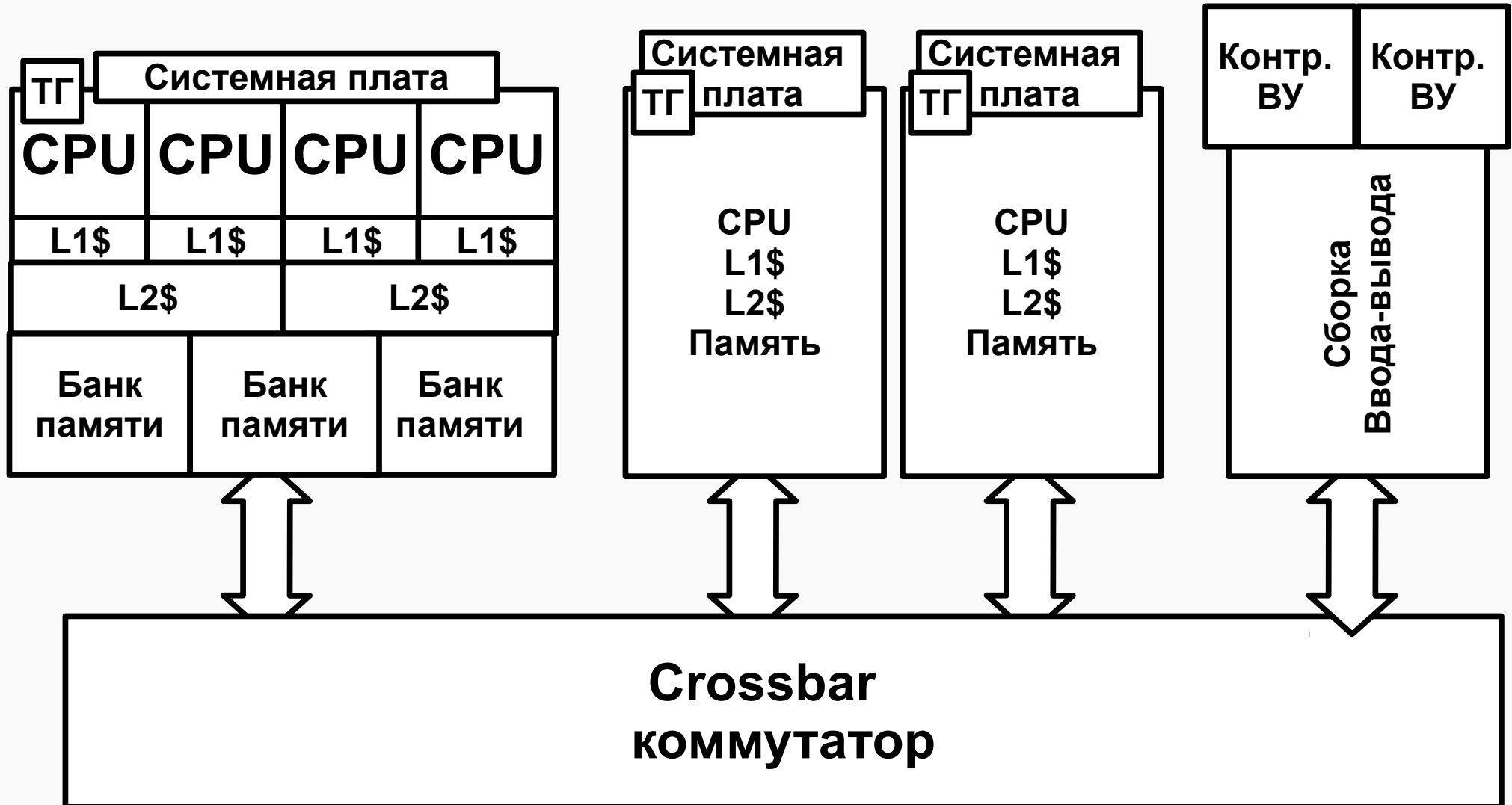
Мультипроцессорность: UMA - Uniform Memory Access



Мультипроцессорность: Коммутатор



Мультипроцессорность: NUMA Non-Uniform Memory Access



Современные коммерческие процессоры



- Разрядность адреса и данных 16/32/64 бита
- Тактовые частоты 500МГц-5Ггц.
- Многопроцессорные 1-100+ CPU
- Многоядерные 1-16 ядер
- От 1 ГБ до терабайтов ОЗУ
- Используют кэш-память разных уровней
- Суперскалярные
- CISC, RISC, VLIW

CISC, RISC, VLIW

- Complex Instruction Set Computer
 - Традиционные процессоры (например Intel), отягощенные совместимостью
- Reduced Instruction Set Computer
 - Простой набор инструкций, выполнение инструкции за такт
- Very Long Instructions Word
 - Несколько инструкций, упакованных в одну команду
 - Упаковка операций в инструкцию ложится на компилятор



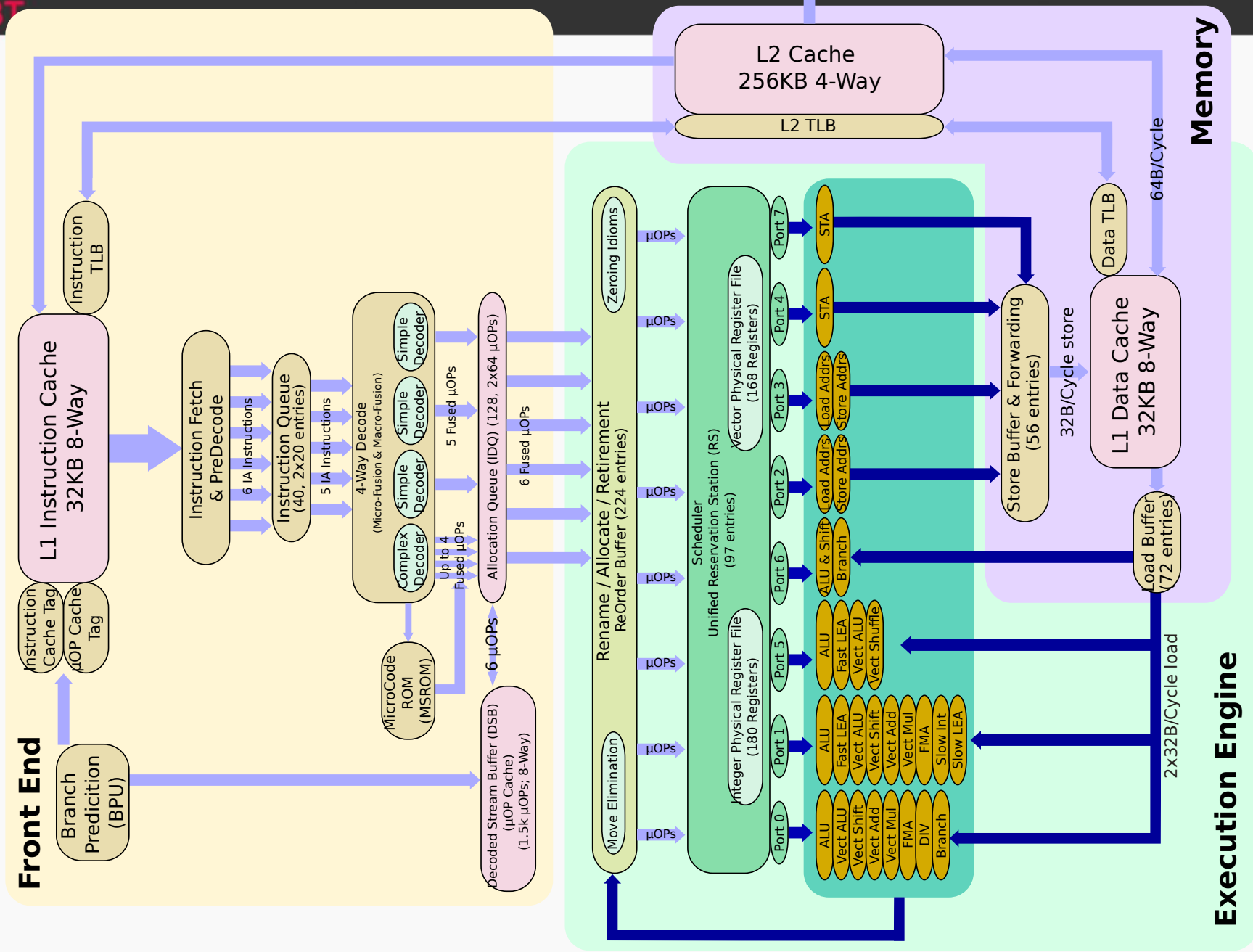
«Современные» процессоры Intel

Year	Microarchitecture	Pipeline stages	max. Clock
1989	486 (80486)	3	100 MHz
1993	P5 (Pentium)	5	300 MHz
1995	P6 (Pentium II)	14 (17 with load & store/retire)	450 MHz
1999	P6 (Pentium III)	12 (15 with load & store/retire)	450~1400 MHz
2000	NetBurst (Pentium 4)	20	800~3466 MHz
2003	Pentium M	10 (12 with fetch/retire)	400~1000 MHz
2004	Prescott	31	3800 MHz
2006	Intel Core	12 (14 with fetch/retire)	3000 MHz
2008	Nehalem	20	3000 MHz
2008	Bonnell	16 (20 with prediction miss)	2100 MHz
2011	Sandy Bridge	14 (16 with fetch/retire)	4000 MHz
2013	Silvermon	14-17 (16-19 with fetch/retire)	2670 MHz
2013	Haswell	14 (16 with fetch/retire)	≈4000 MHz
2015	Skylake	14 (16 with fetch/retire)	≈4000 MHz
2016	Kaby Lake	14 (16 with fetch/retire)	4500 MHz
2017	Coffe Lake	14 (16 with fetch/retire)	4700 MHz
2018	Cannon Lake		

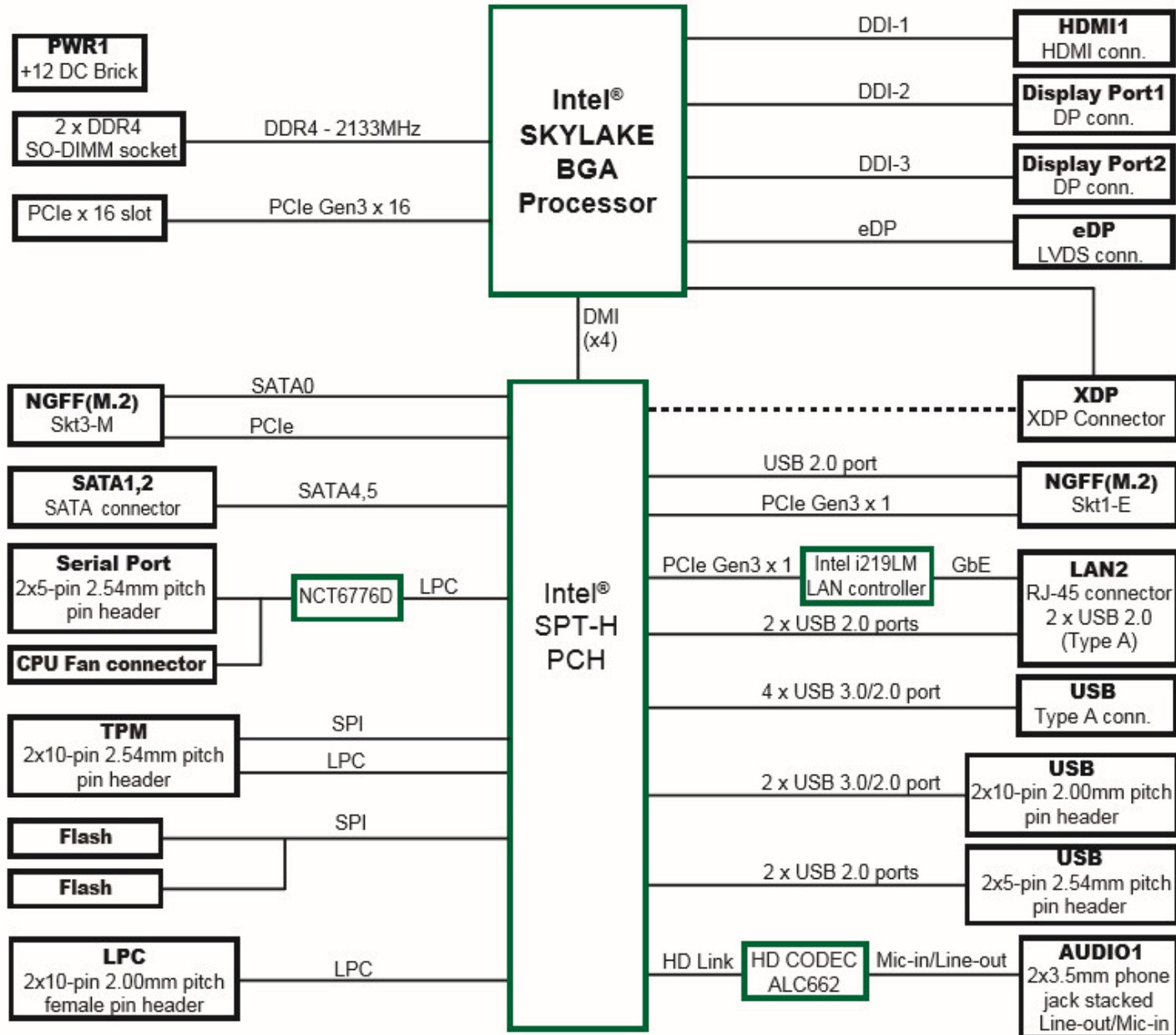


NTMO BT

Intel Skylake block diagram

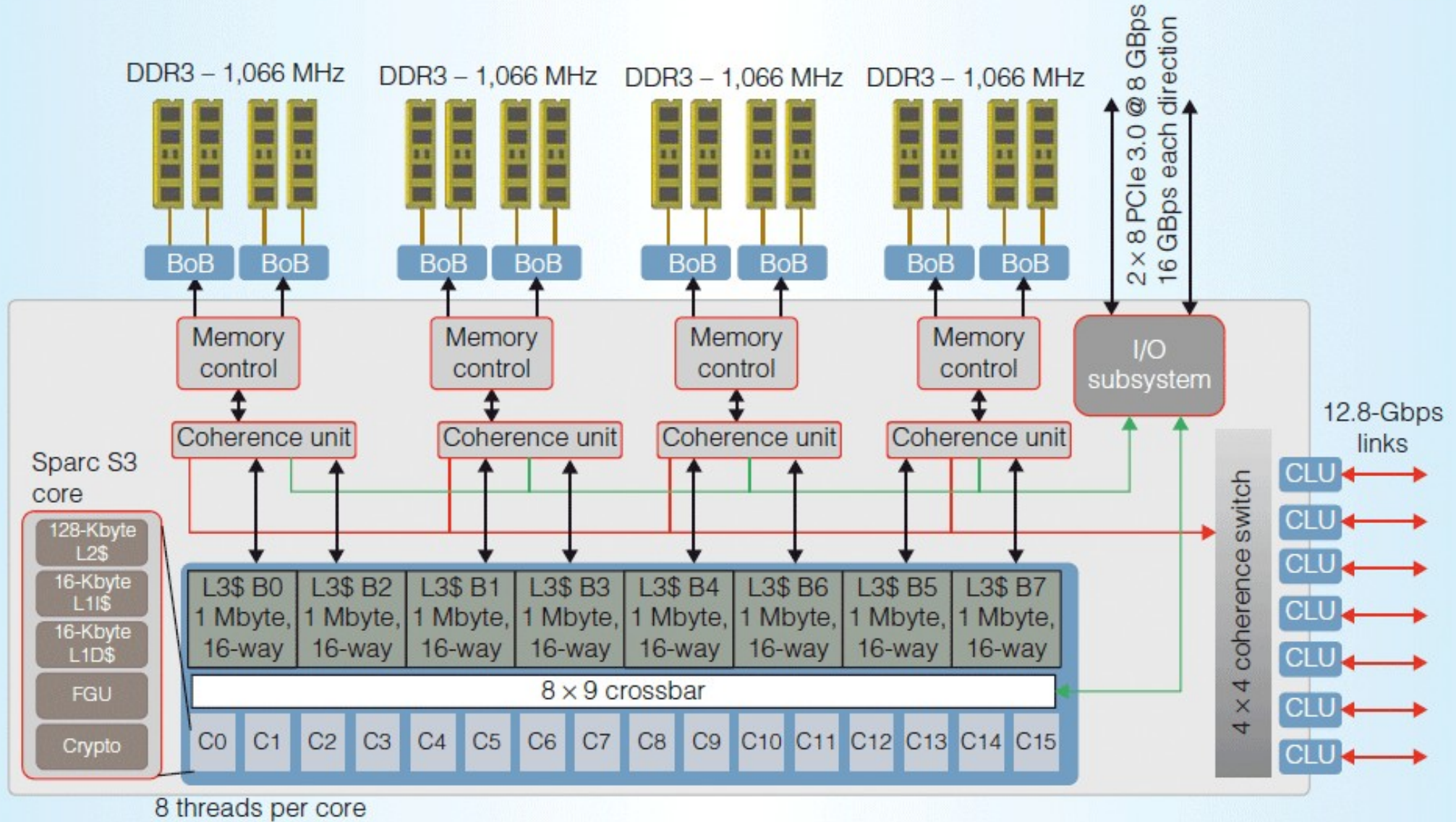


Архитектура: Arbor ITX-i89H



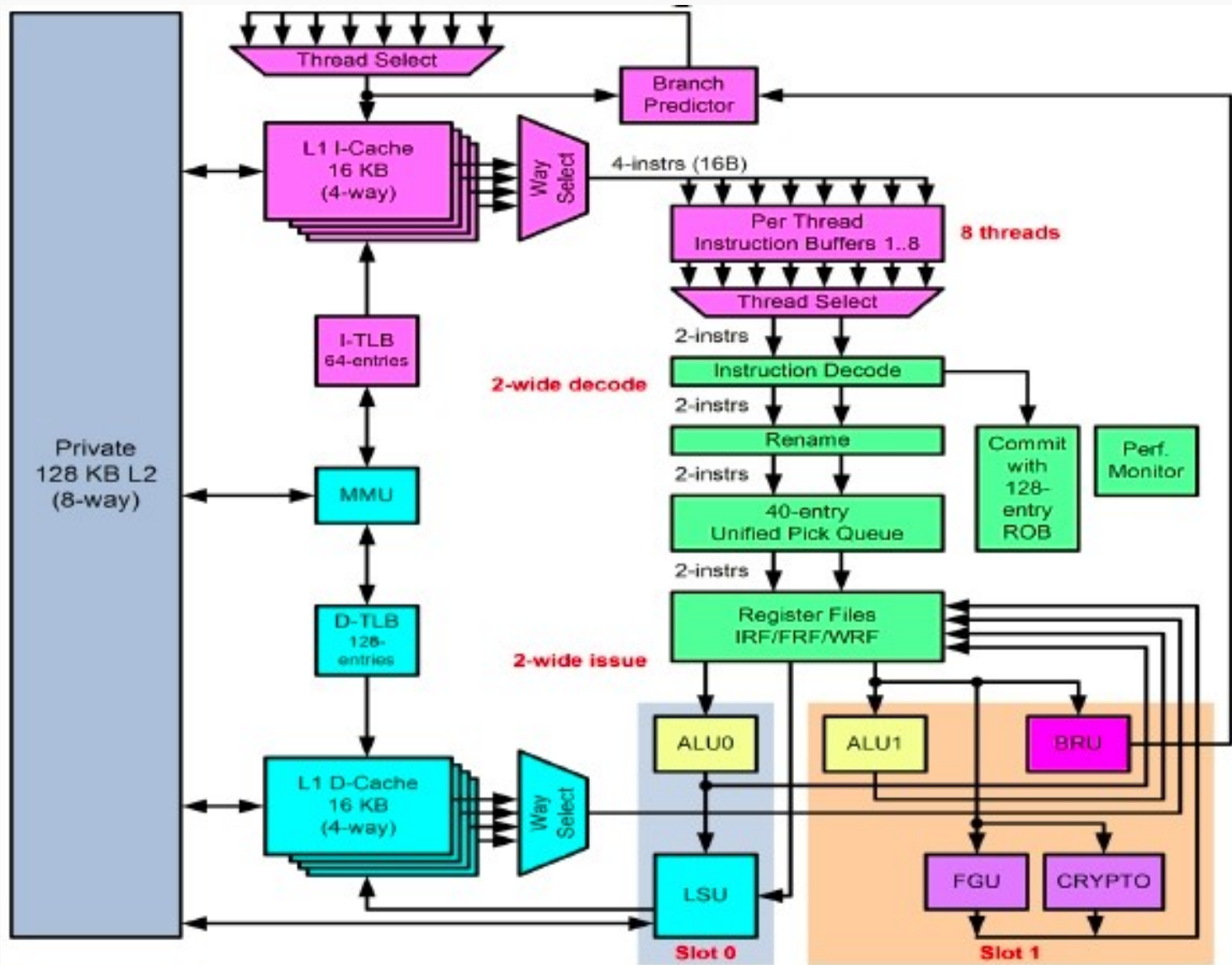


SPARC T5 16xCores

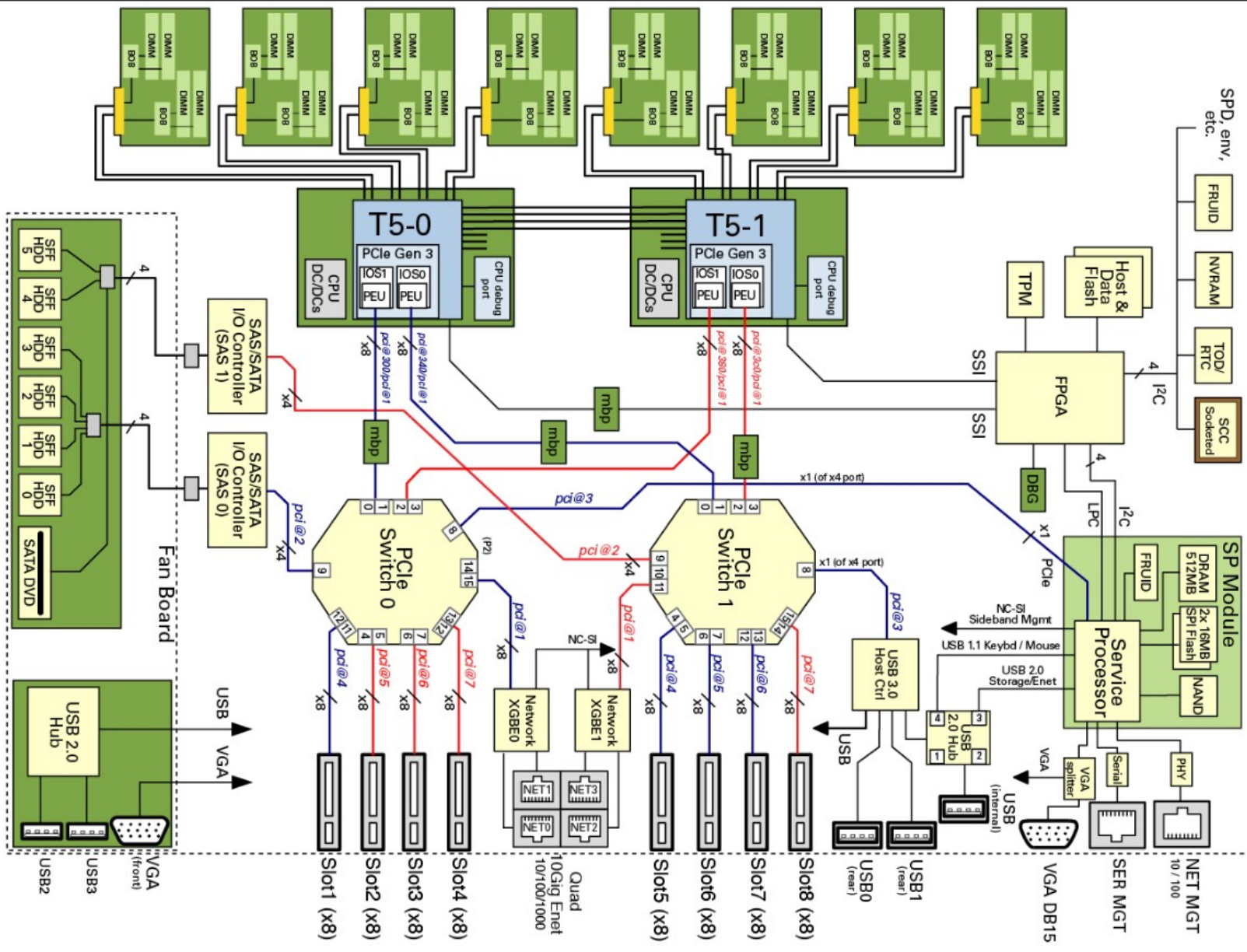


(C) Oracle marketing material

Sparc S3 Core



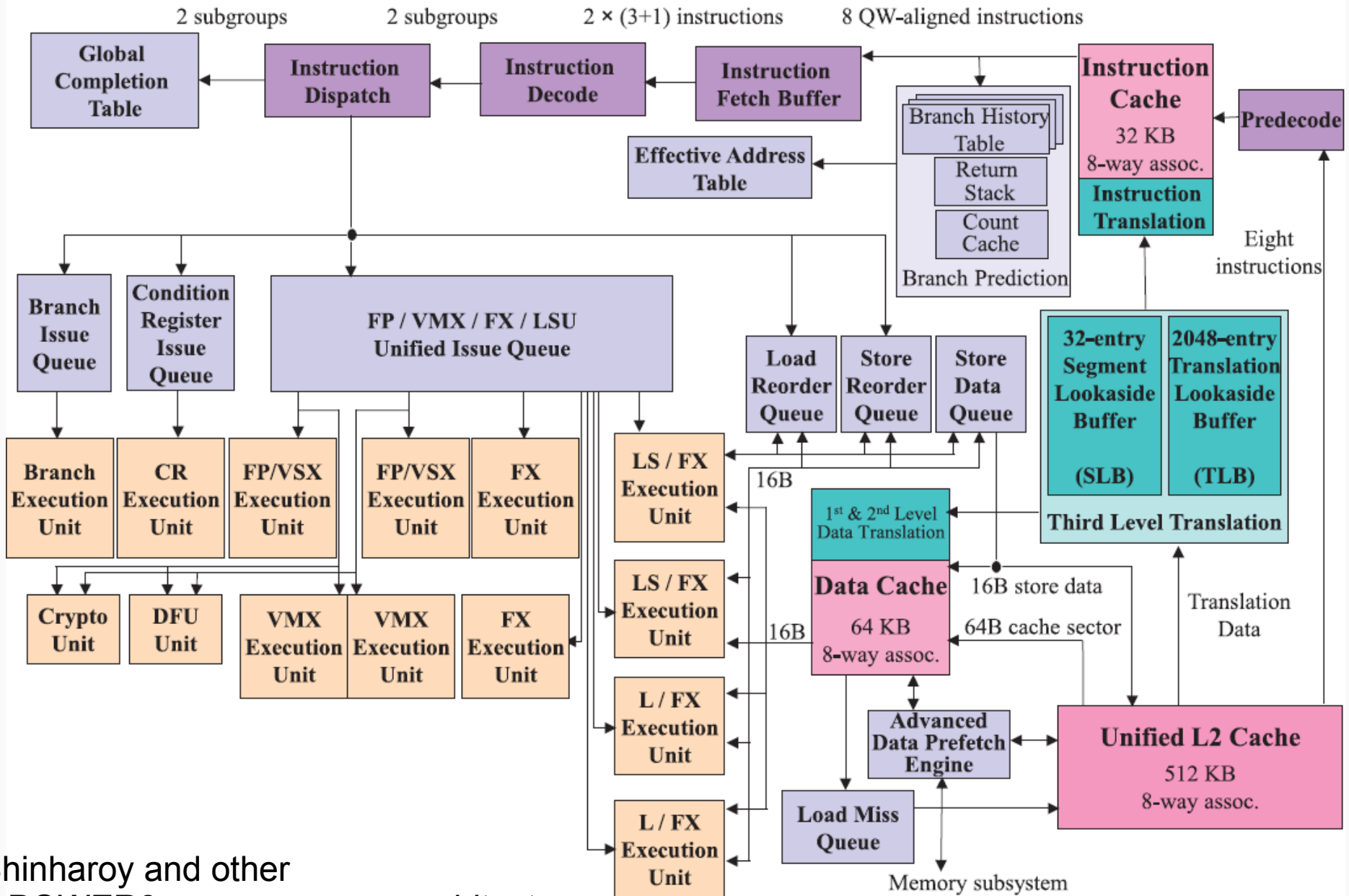
Двухпроцессорная система на базе SPARC-T5





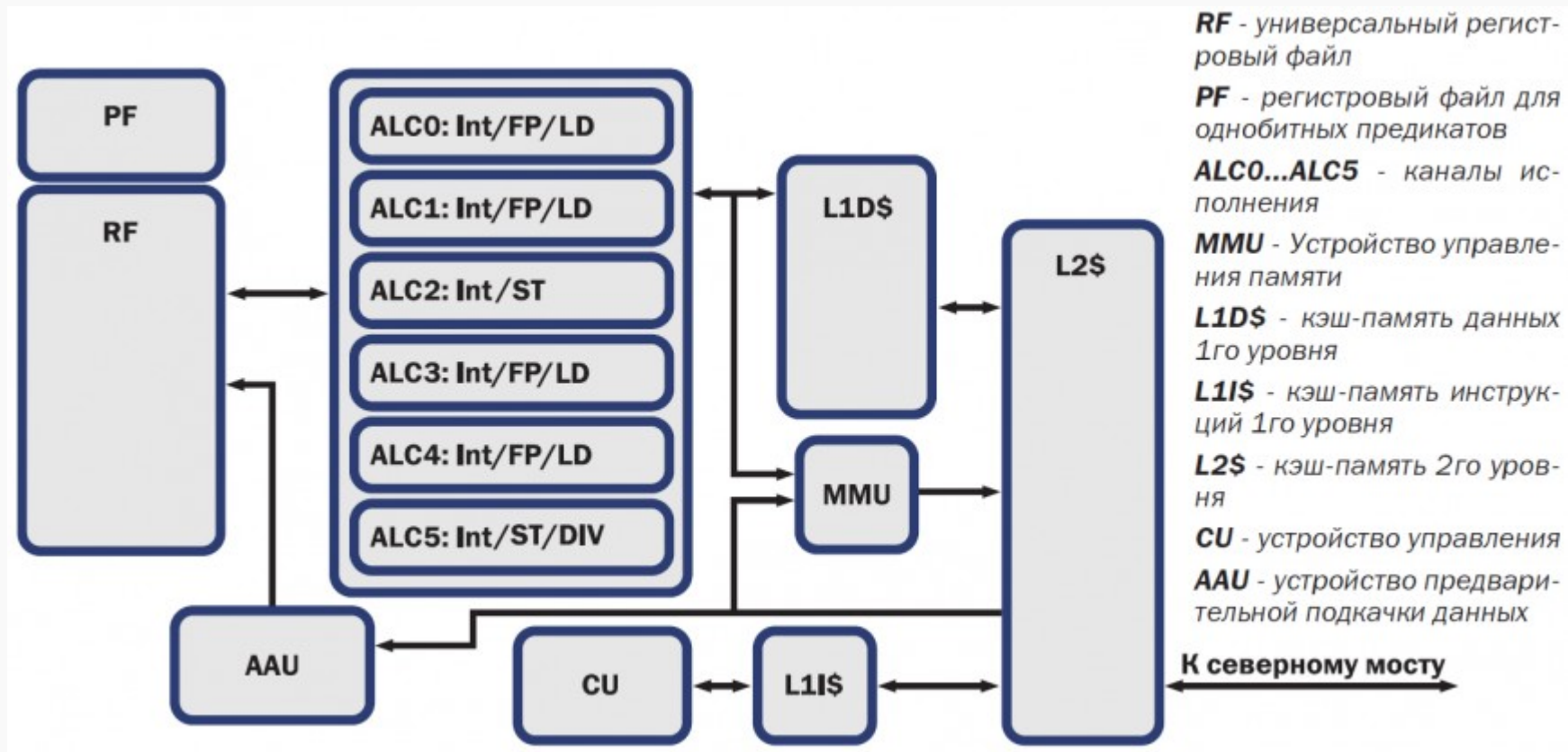
NTMO BT

IBM POWER8



B. Shinharoy and other
IBM POWER8 processor core architecture

Эльбрус 8с



- Технология 28нм
- 8 ядер x 1300МГц
- L1 I128K + D64K
- L2 512 К
- L3 16Мб

- 3-х канальная DDR3-1600MHZ
- До 80 Вт
- Linux 2.2
- Эмуляция x86 (-30%)



Устройства хранения данных

10

Характеристики памяти

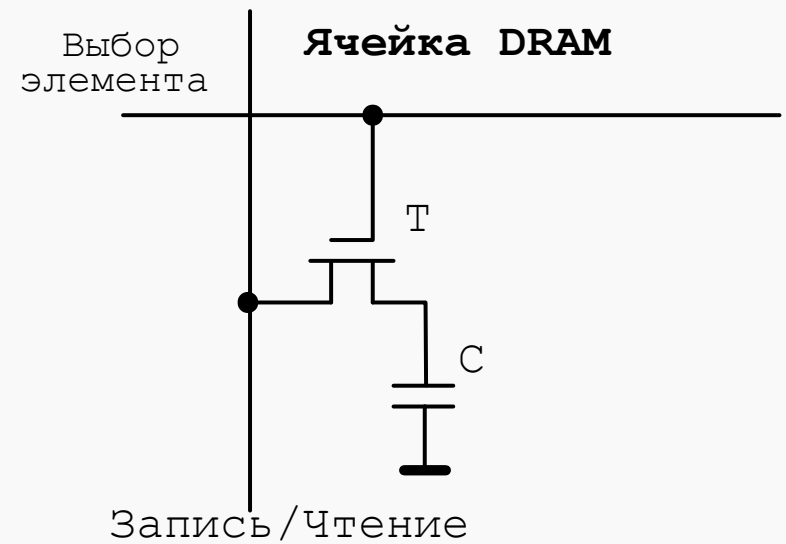
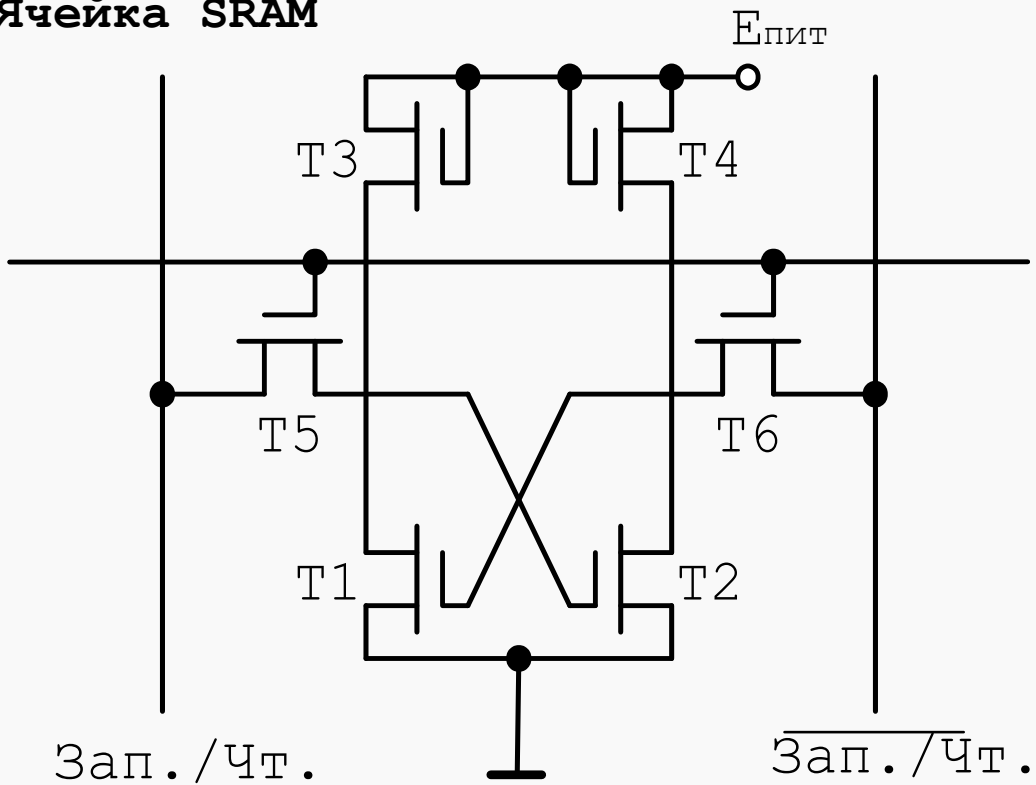
- Месторасположение
 - процессорные, внутренние, внешние
- Емкость
 - В метрических (Кило-) и двоичных (Кибби-) множителях
- Единица пересылки
 - Слово, строка кэша, блок на диске
- Метод доступа
 - Произвольный (адресный), ориентированных на записи (прямой), последовательный, ассоциативный

Характеристики памяти

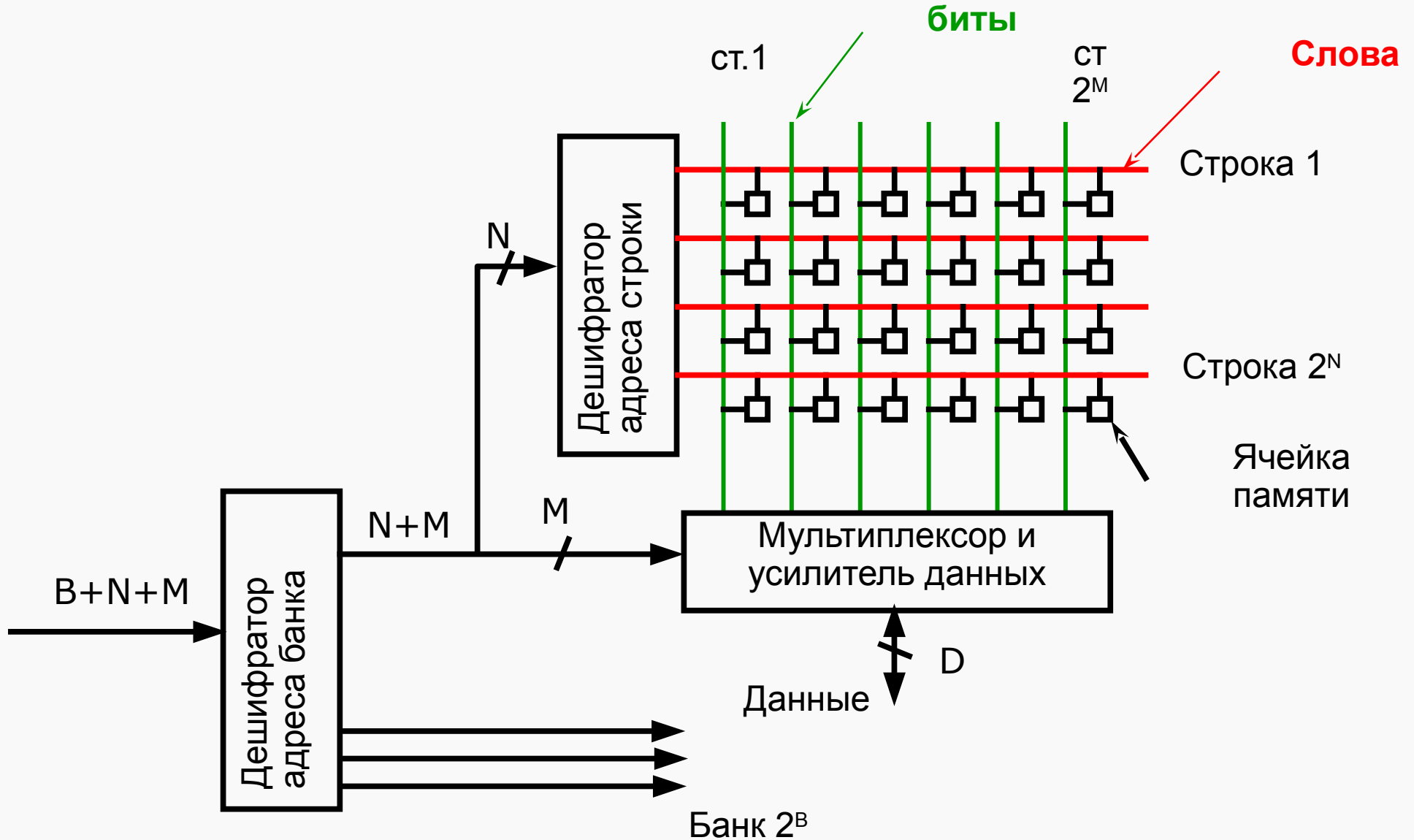
- Быстродействие и временные соотношения
 - Время доступа T_d
 - Длительность цикла памяти (время обращения) T_c
 - Время чтения и время записи
 - Время восстановления T_v
 - Скорость передачи информации
- Физический тип и особенности
- Стоимость

Статическая vs Динамическая

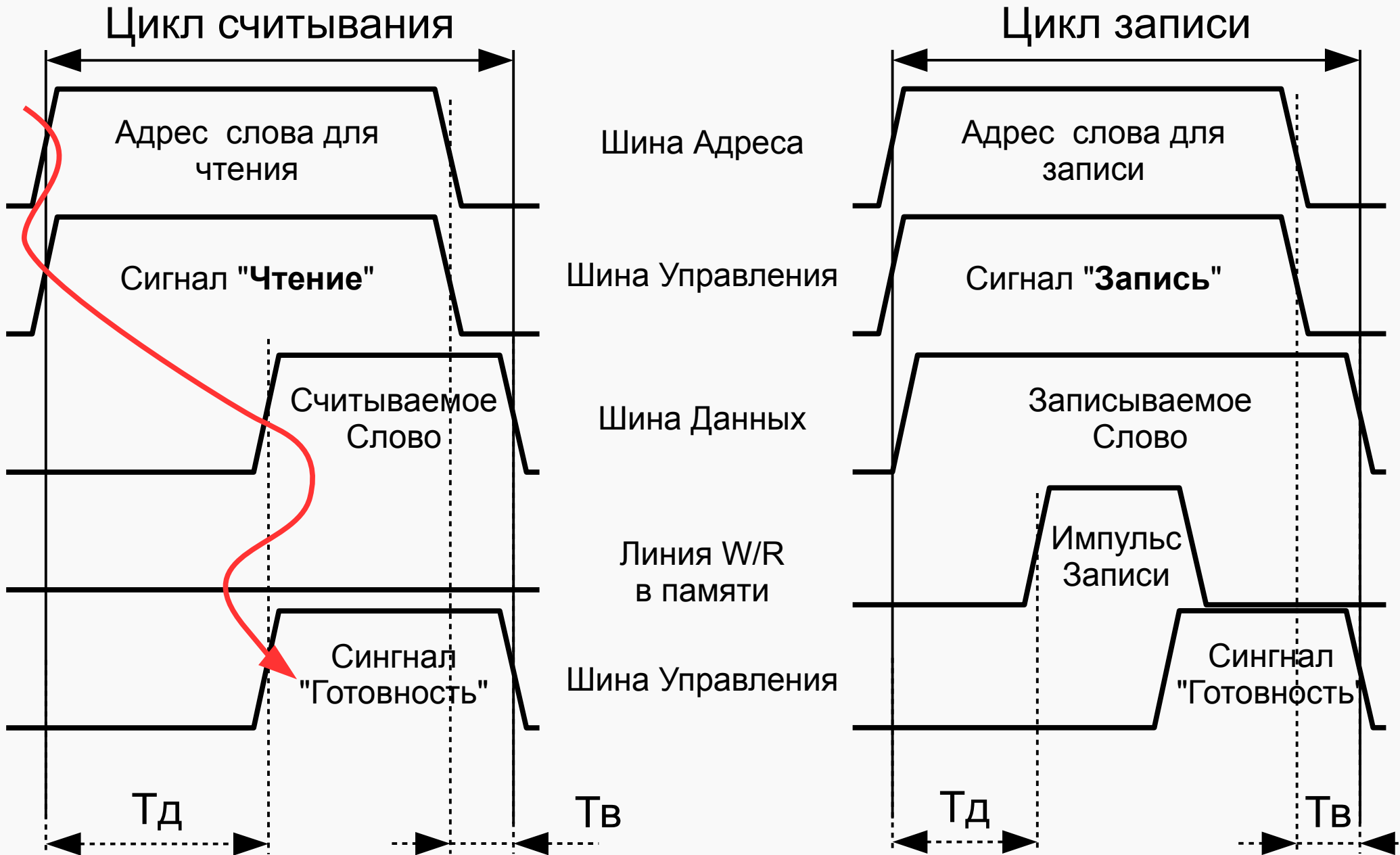
Ячейка SRAM



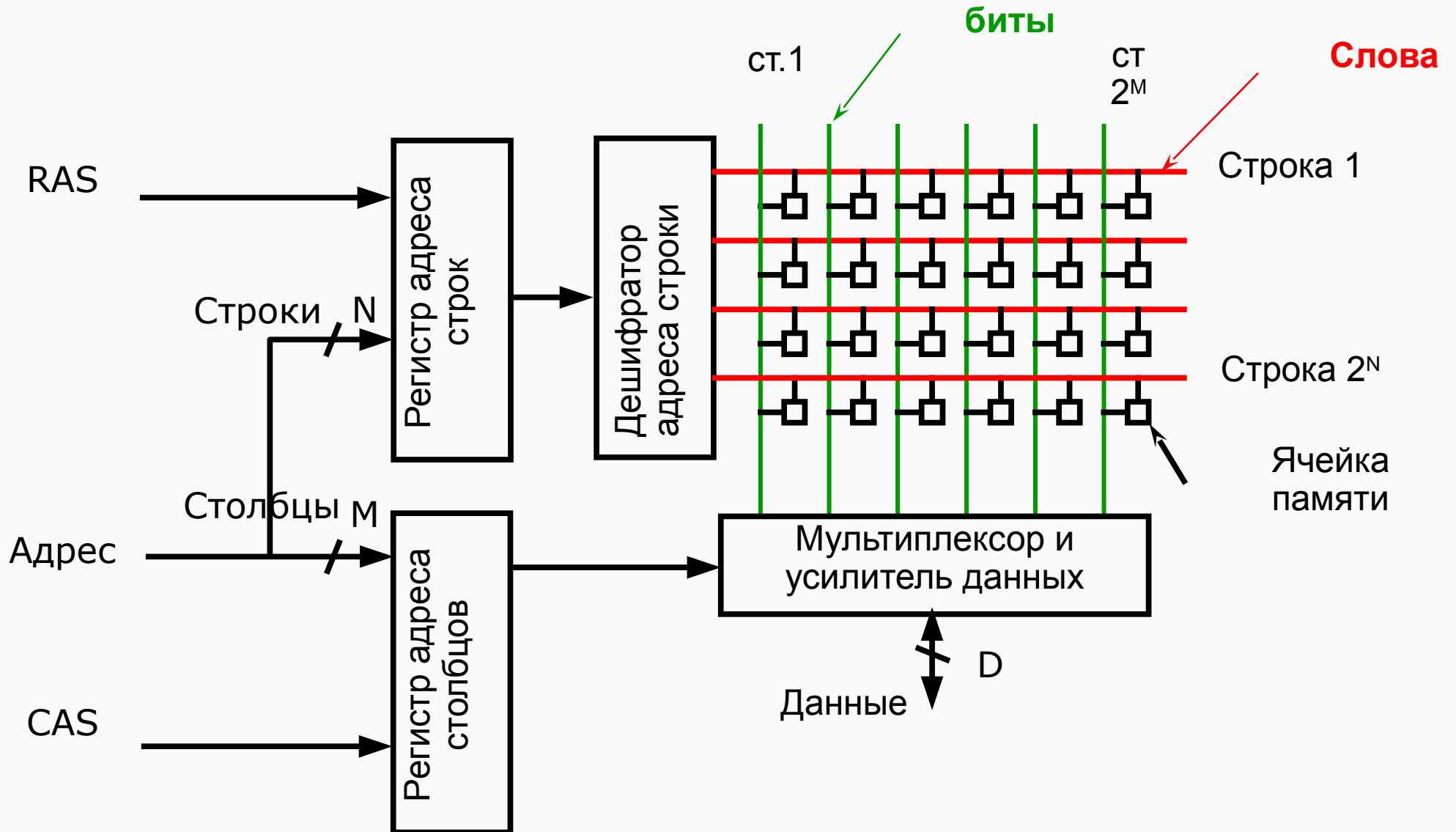
Адресуемая память



Диаграммы работы с адресуемой асинхронной памятью

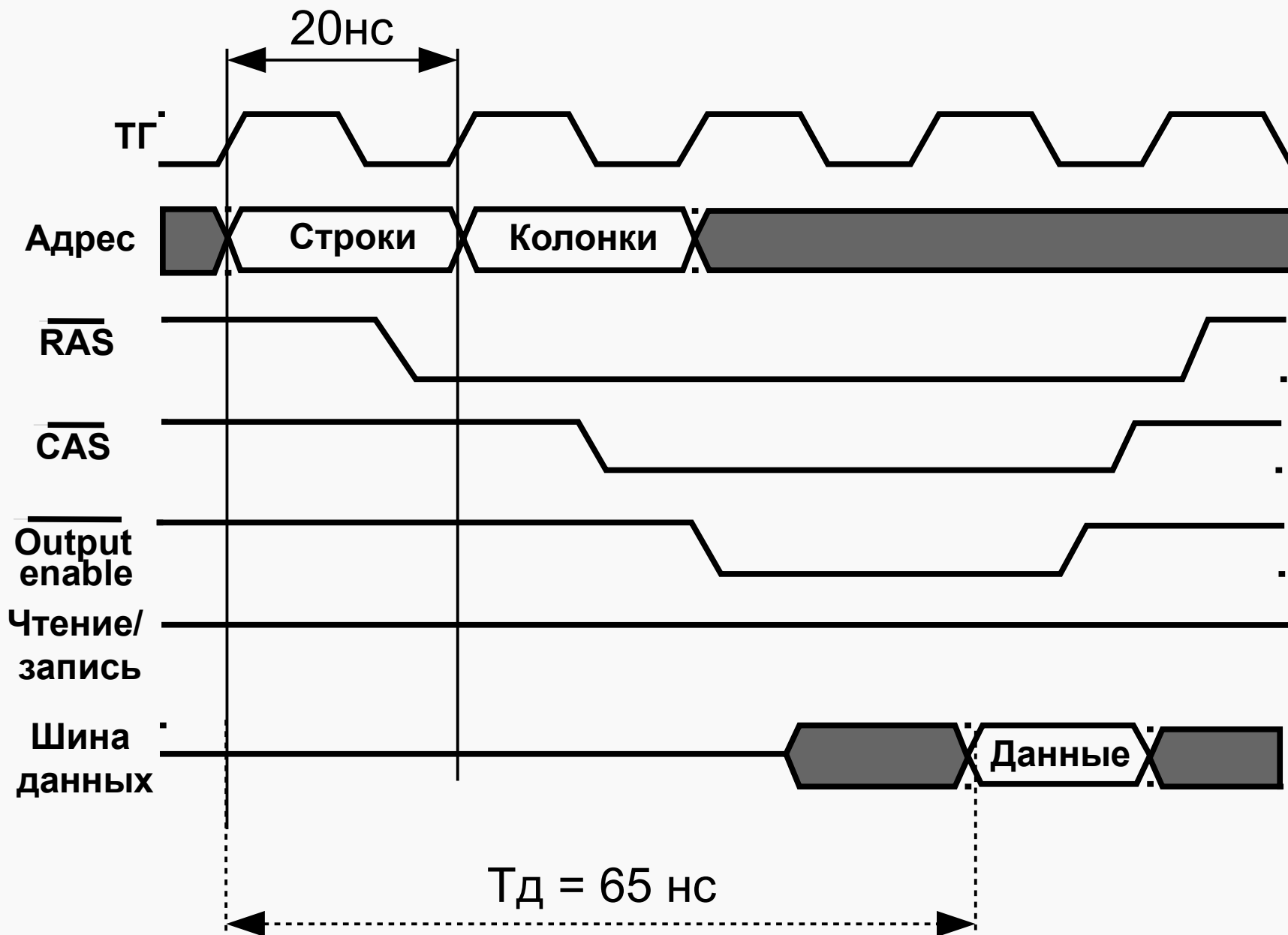


Адресуемая память с фиксацией строк и столбцов





Синхронная память SDRAM





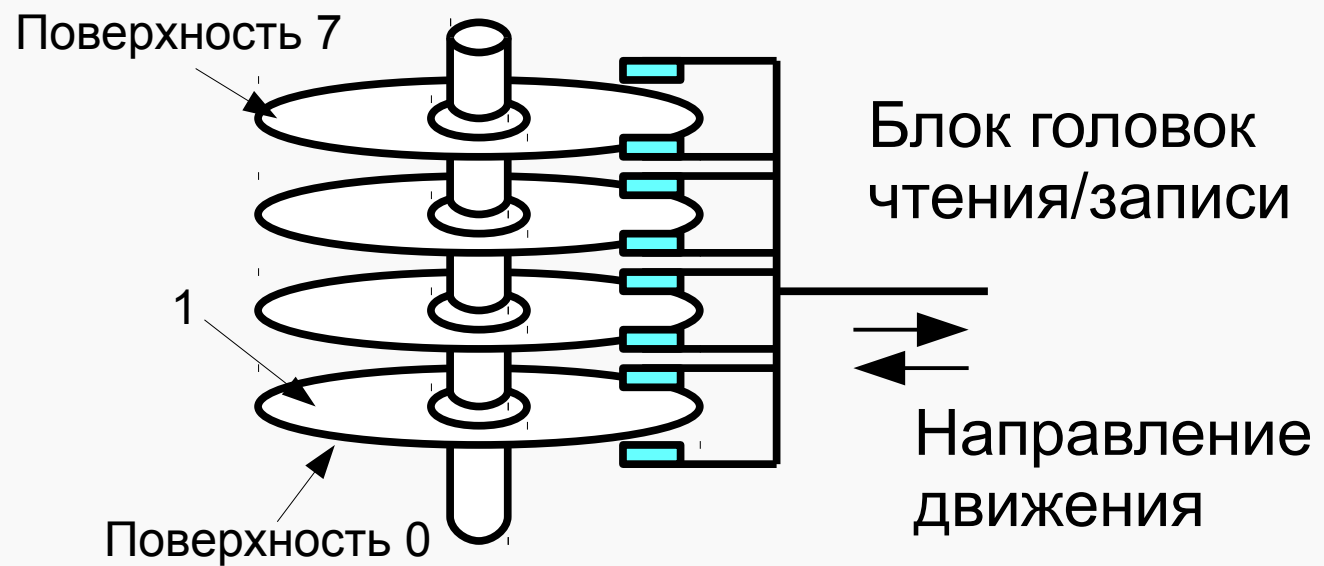
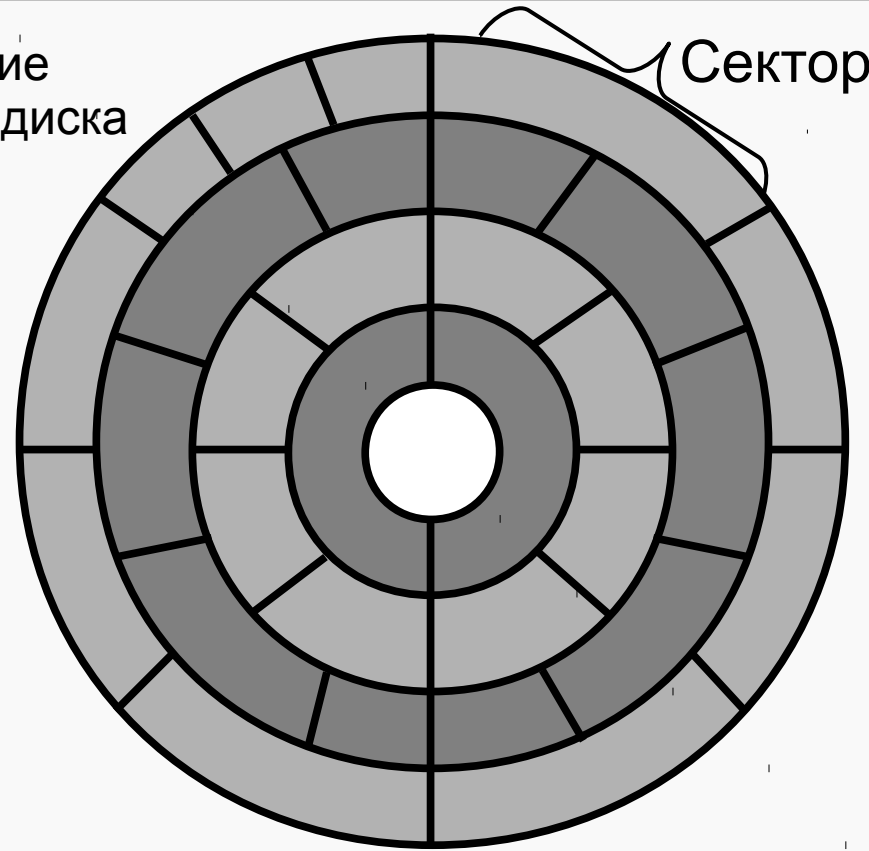
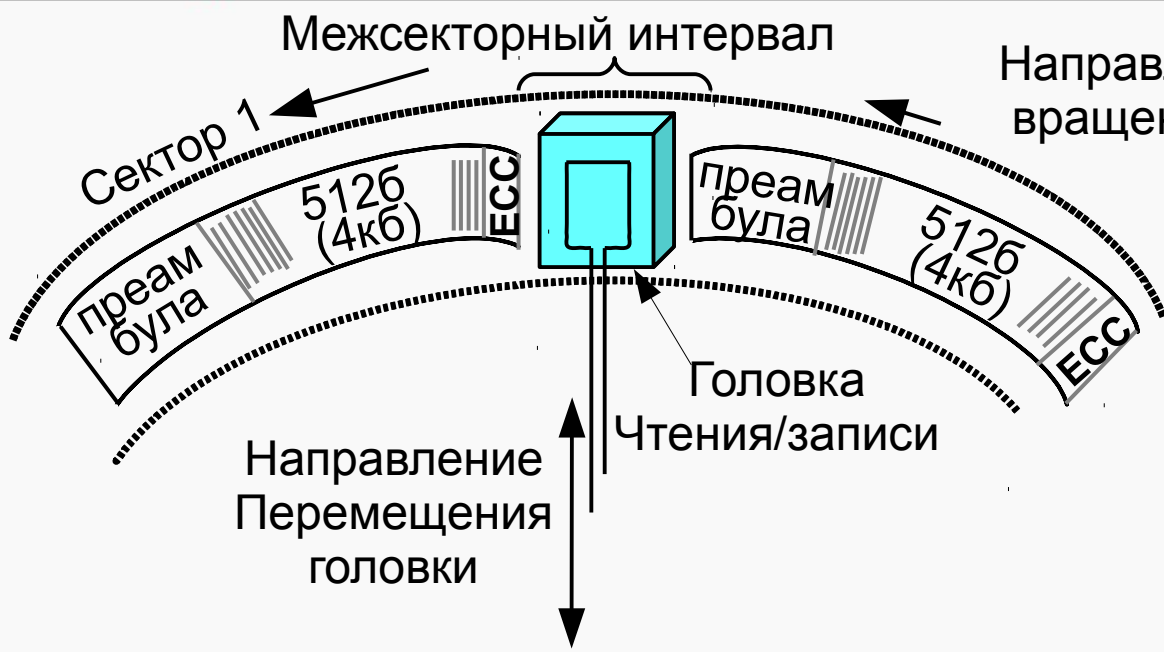
Конструктивные особенности современной памяти

- Burst mode — пакетный режим
- Double Data Rate — передача данных и по фронту и по спаду
- SPD — чип, содержащий идентификационную информацию
- Interleaving — расслоение памяти, повышает производительность
- DDR4-2133 8192MB **PC4-17000** индекс производительности
-



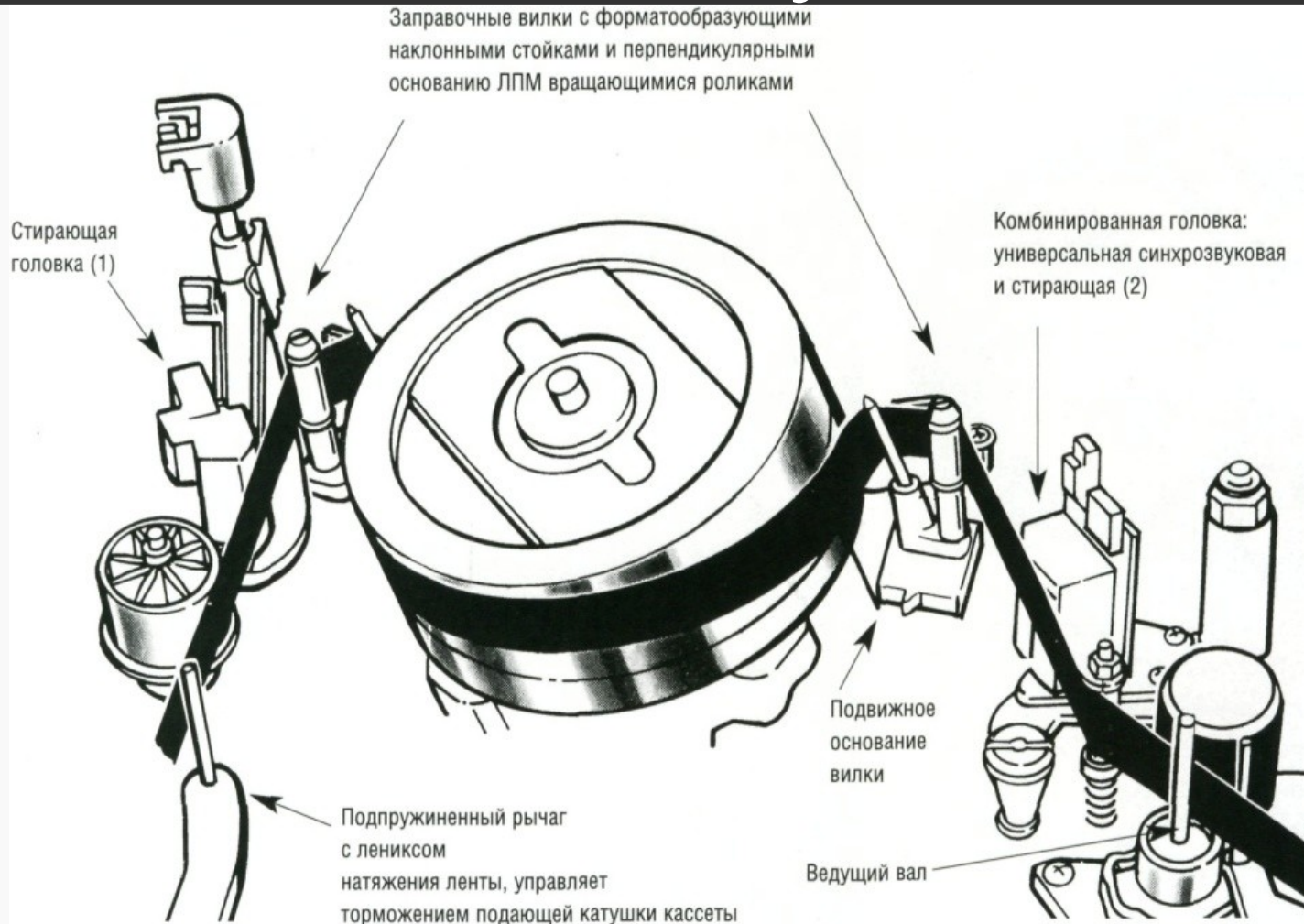
Память, ориентированная на записи

ИТМО ВТ




$$T_{\partial} = \frac{(T_{noz} + T_{вр})}{2}$$

Память, с последовательным доступом*



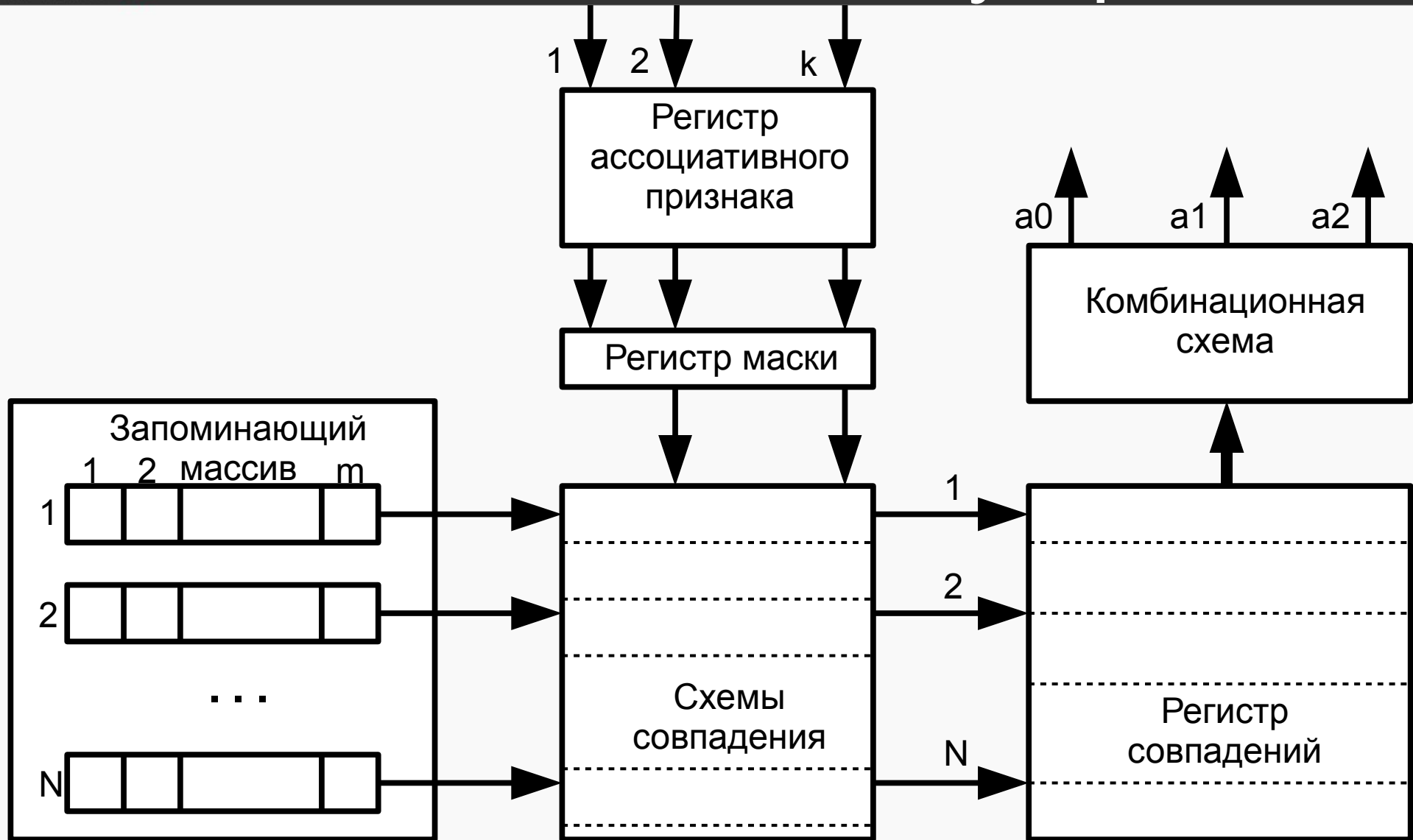
* на самом деле эта картинка от видеомэгнитофона, но принцип тот же



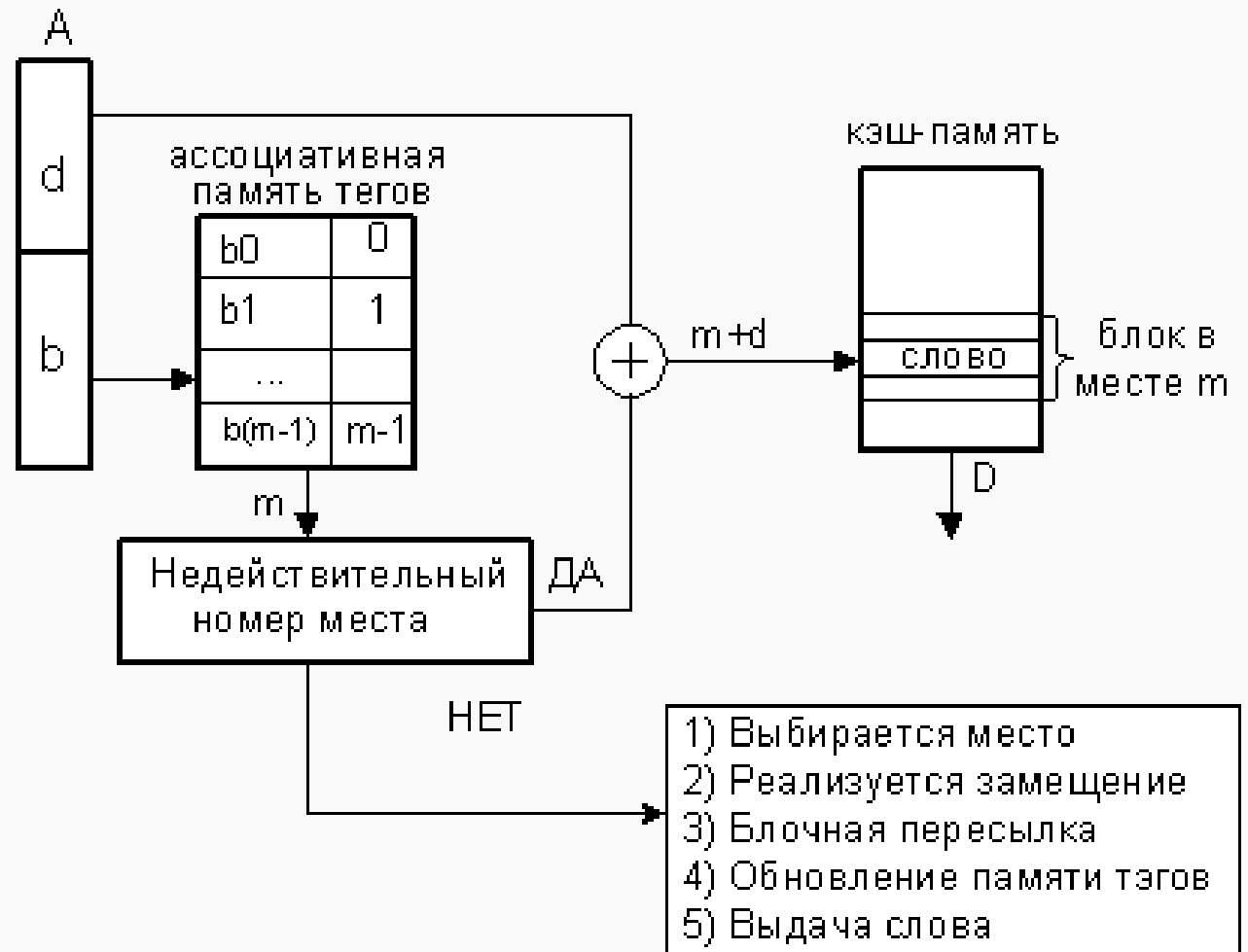
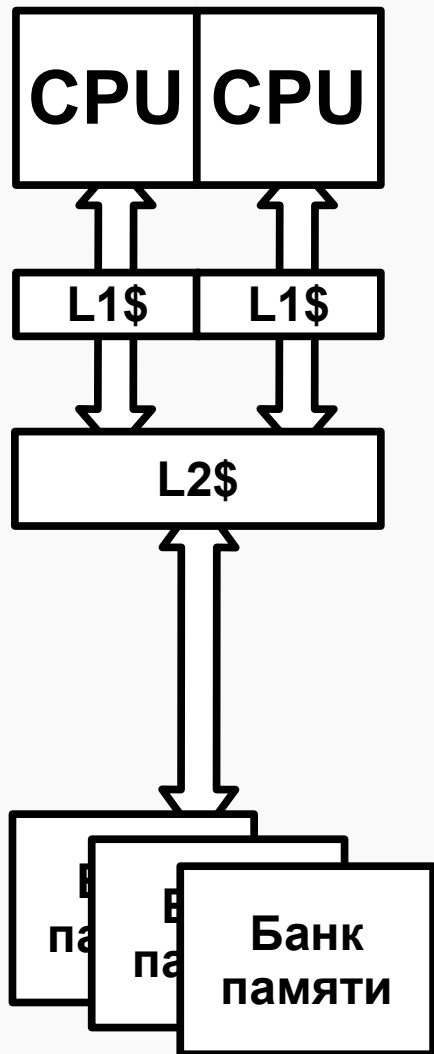
Расположение дорожек

Лента

Структура ассоциативного запоминающего устройства



Кэш память

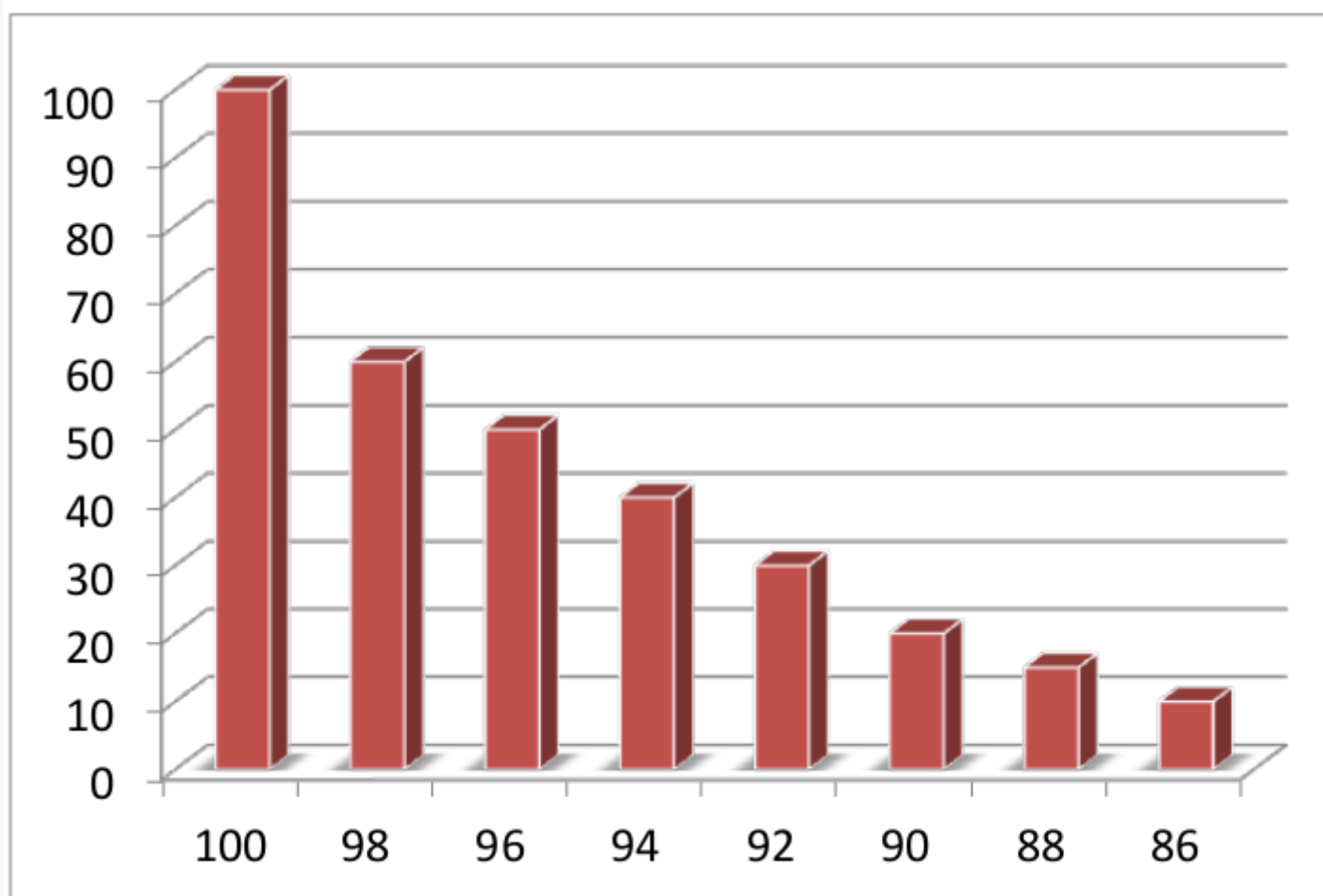


Пирамида памяти

	Объем	Тд	*	Тип	Управл.
CPU	100-1000 б.	<1нс	1с	Регистр	компилятор
L1 Cache	32-128Кб	1-4нс	2с	Ассоц.	аппаратура
L2-L3 Cache	0.5-32Мб	8-20нс	19с	Ассоц.	аппаратура
Основная память	0.5Гб-4ТБ	60-200нс	50-300с	Адресная	программно
SSD	128Гб-1Тб/drive	25-250мкс	5д	Блочн.	программно
Жесткие диски	0.5Тб-4Тб/drive	5-20мс	4м	Блочн.	программно
Магнитные ленты	1-6Тб/к	1-240с	200л	Последов.	программно

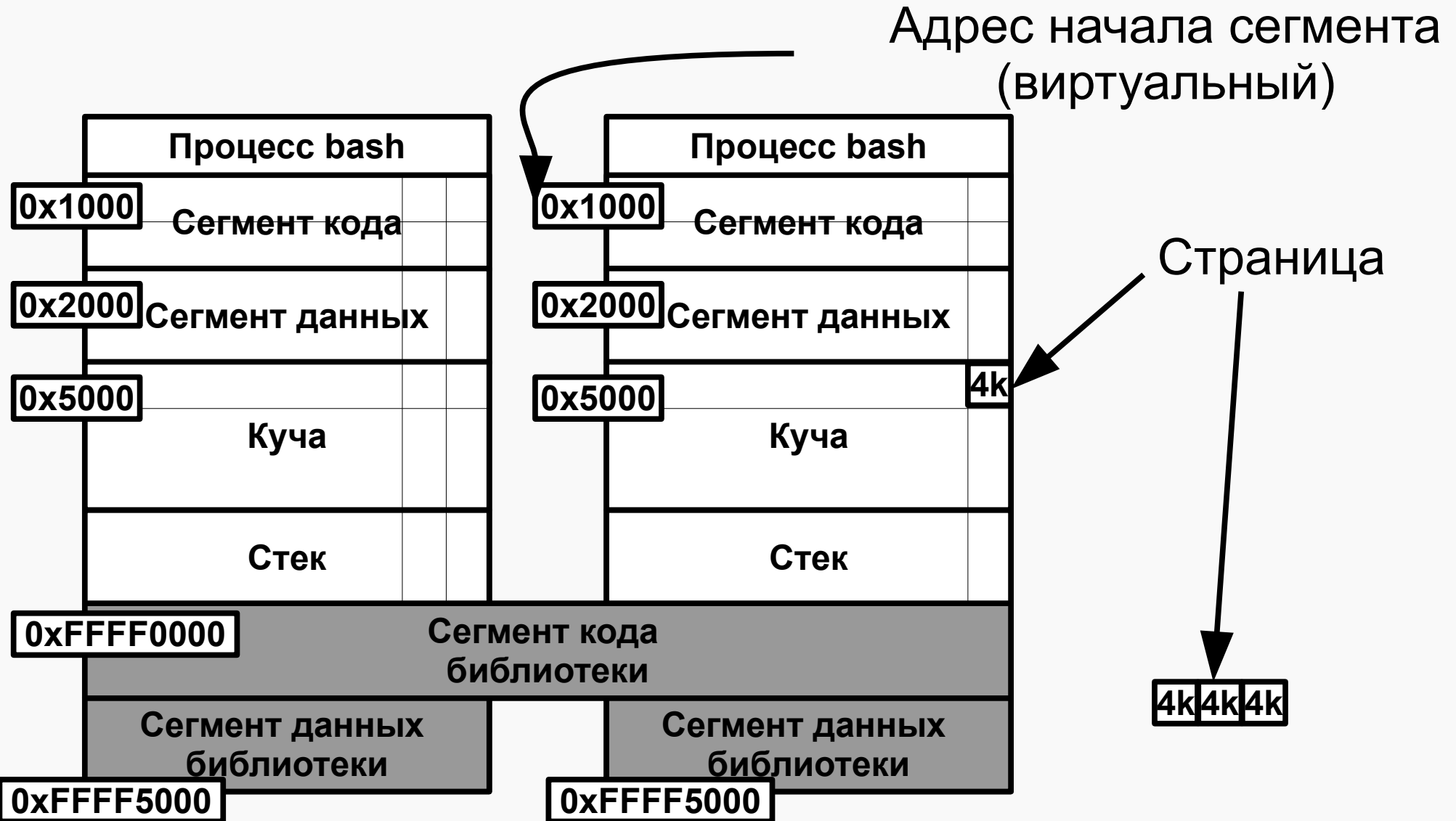
Влияние промахов кэш-памяти

Производительность в % от макс.



Попадания в кэш в %

Сегментно-страничная виртуальная память



Реальный пример

```
root@ra:~# pmap -x $$
```

```
5081: /bin/bash
```

Address	Kbytes	RSS	Dirty	Mode	Mapping
08048000	1064	852	0	r-x--	bash
08152000	4	4	4	r----	bash
08153000	20	20	20	rw---	bash
08158000	20	20	20	rw---	[anon]
09c79000	1204	1204	1204	rw---	[anon]
b716a000	44	44	0	r-x--	libnss_files-2.23.so
b7176000	4	4	4	rw---	libnss_files-2.23.so
b74e5000	1724	1248	0	r-x--	libc-2.23.so
b7697000	4	4	4	rw---	libc-2.23.so

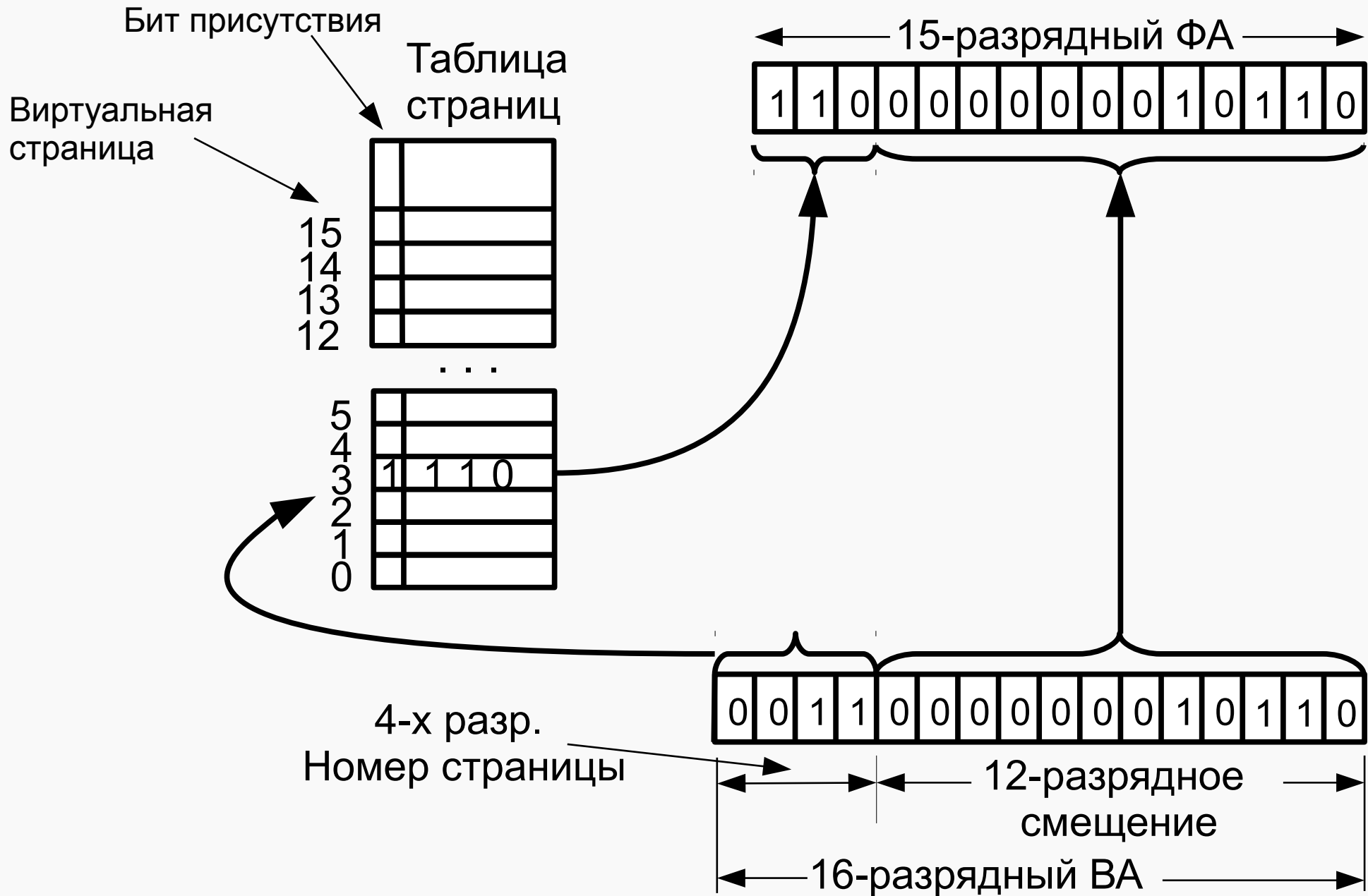
```
root@ra:~# pmap -x 6964
```

```
6964: cat
```

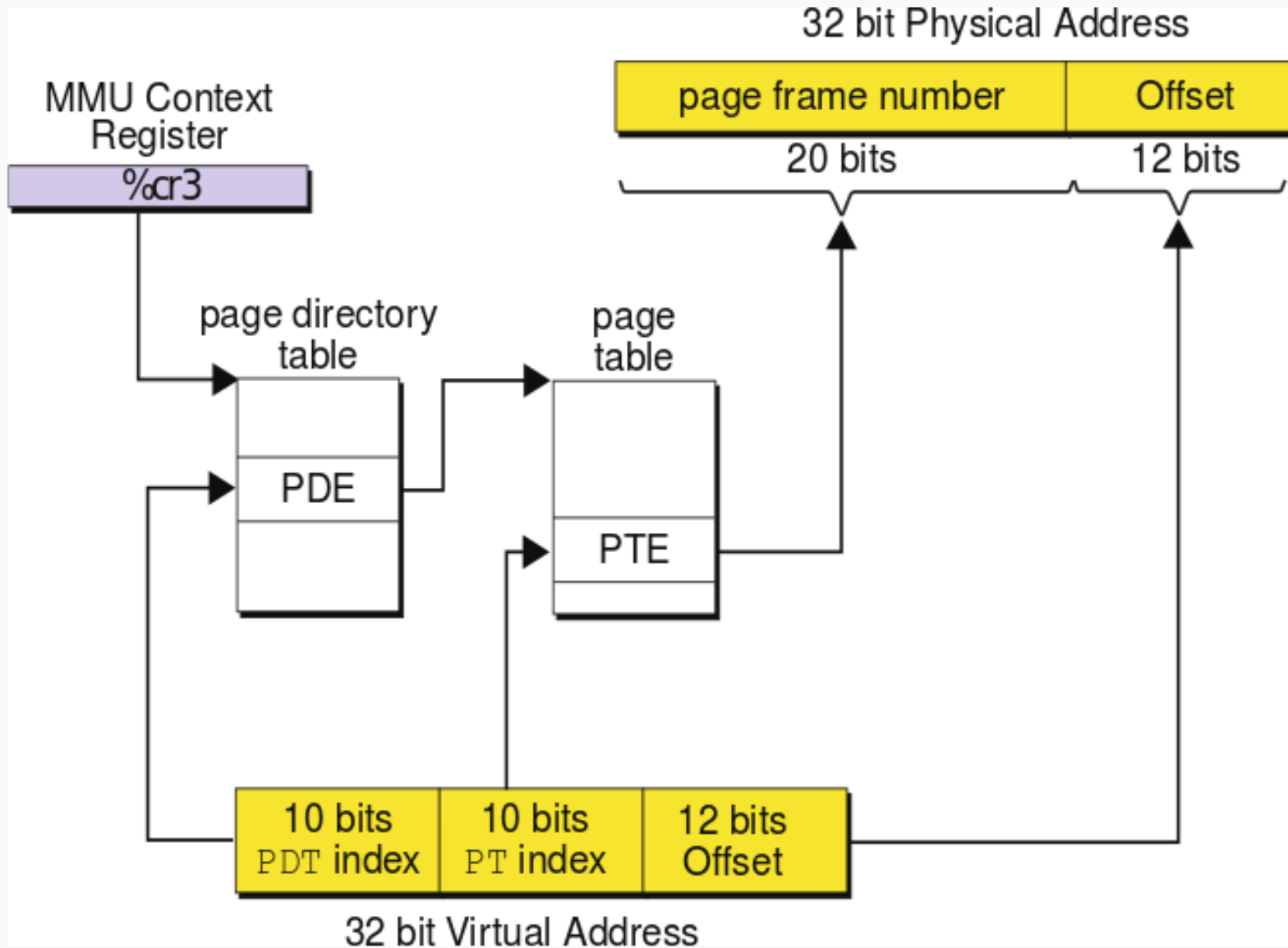
Address	Kbytes	RSS	Dirty	Mode	Mapping
08048000	48	48	0	r-x--	cat
08054000	4	4	4	r----	cat
08055000	4	4	4	rw---	cat
09609000	132	8	8	rw---	[anon]
b724a000	136	4	4	rw---	[anon]
b739e000	2048	372	0	r----	locale-archive
b759e000	1724	964	0	r-x--	libc-2.23.so
b7750000	4	4	4	rw---	libc-2.23.so



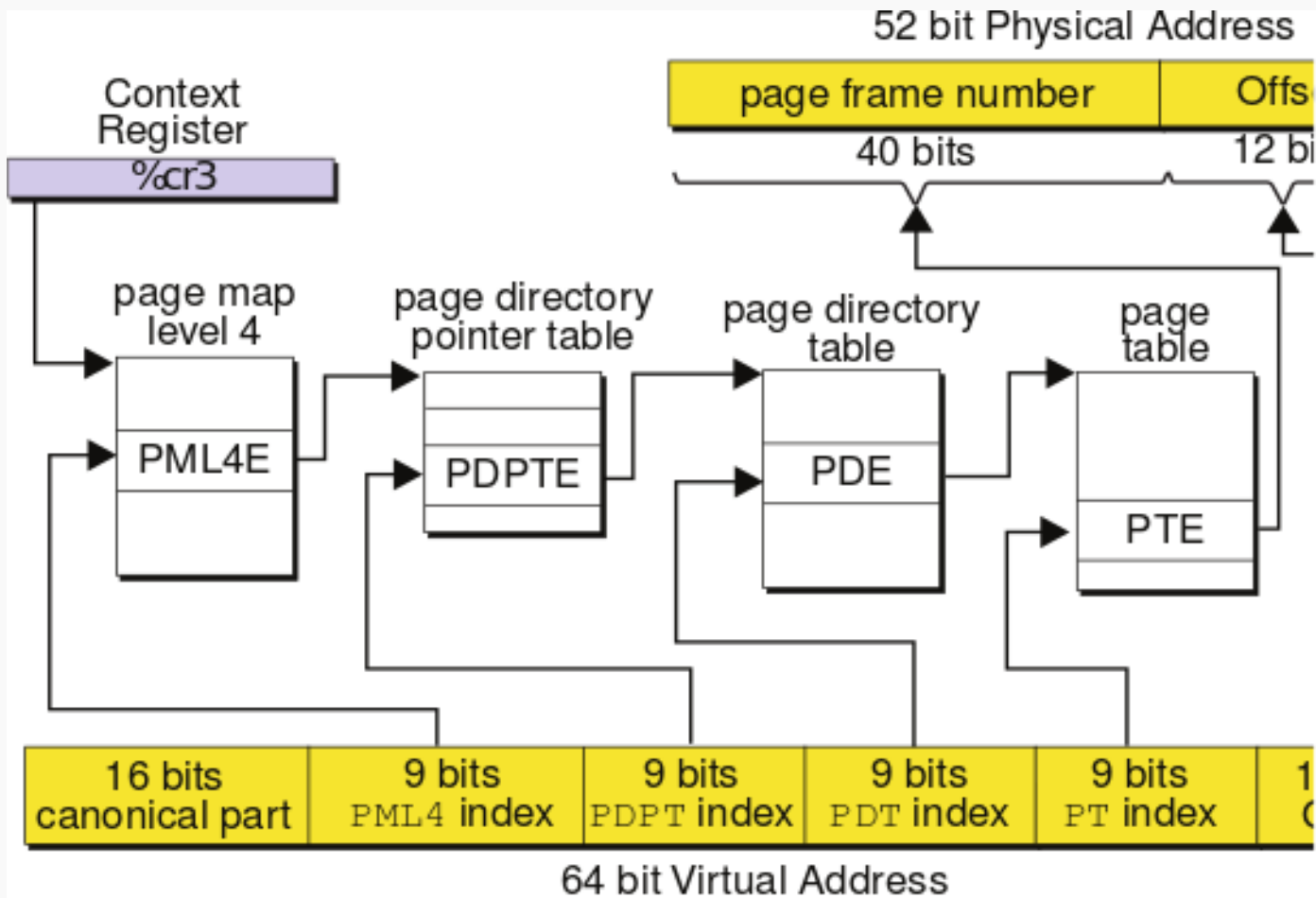
MMU: Трансляция адресов



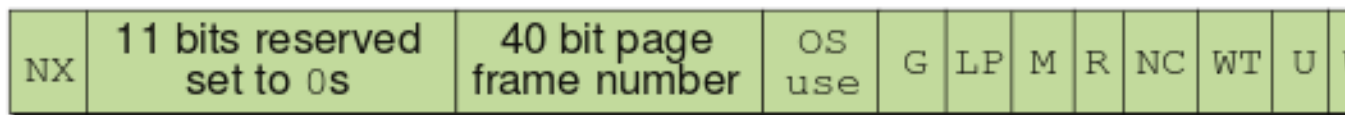
Архитектура x86: MMU



Архитектура x64



64 bit PTE - 64 bit mode



3 bits

- Кэширует часто используемые преобразования
- Обычно раздельный для адреса и данных
- Организован в виде ассоциативной памяти

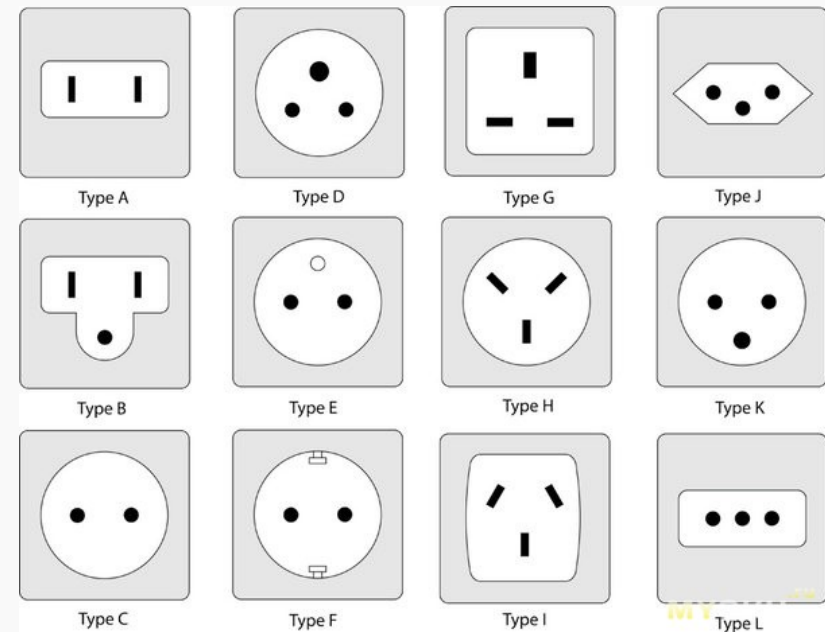
11

Интерфейсы

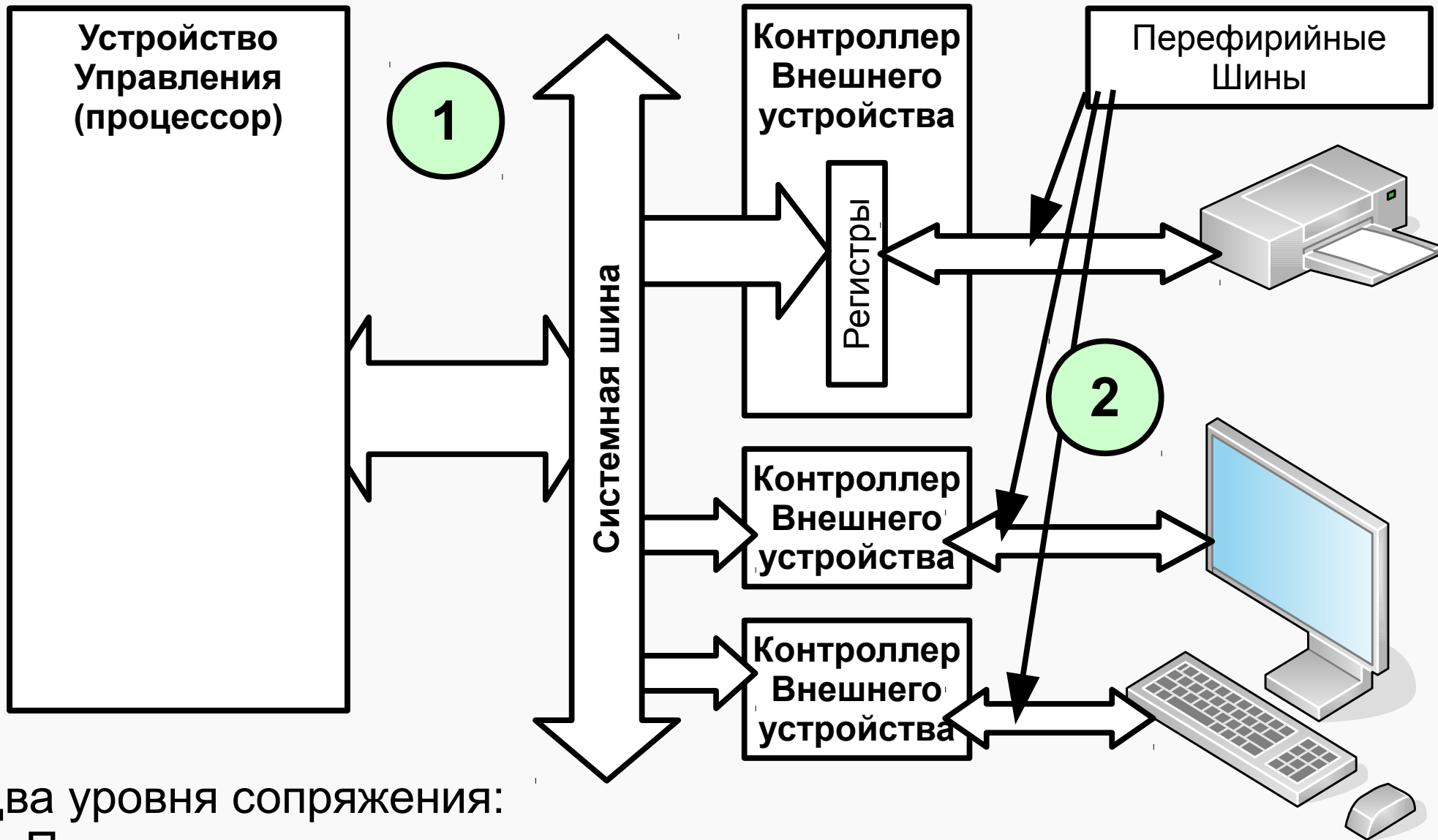
- Определяет конкретные детали обмена
 - Частота, набор каналов передачи, способ кодирования, команды, представления данных, набор данных и последовательность,
- Аппаратная и/или программная реализация
- Нуждаются в точной спецификации и/или стандартизации
 - Стороны обмена должны однозначно интерпретировать детали обмена

Уровни стандартизации интерфейсов

- Логическое подключение
- Физические параметры сигналов
- Конструктивные особенности

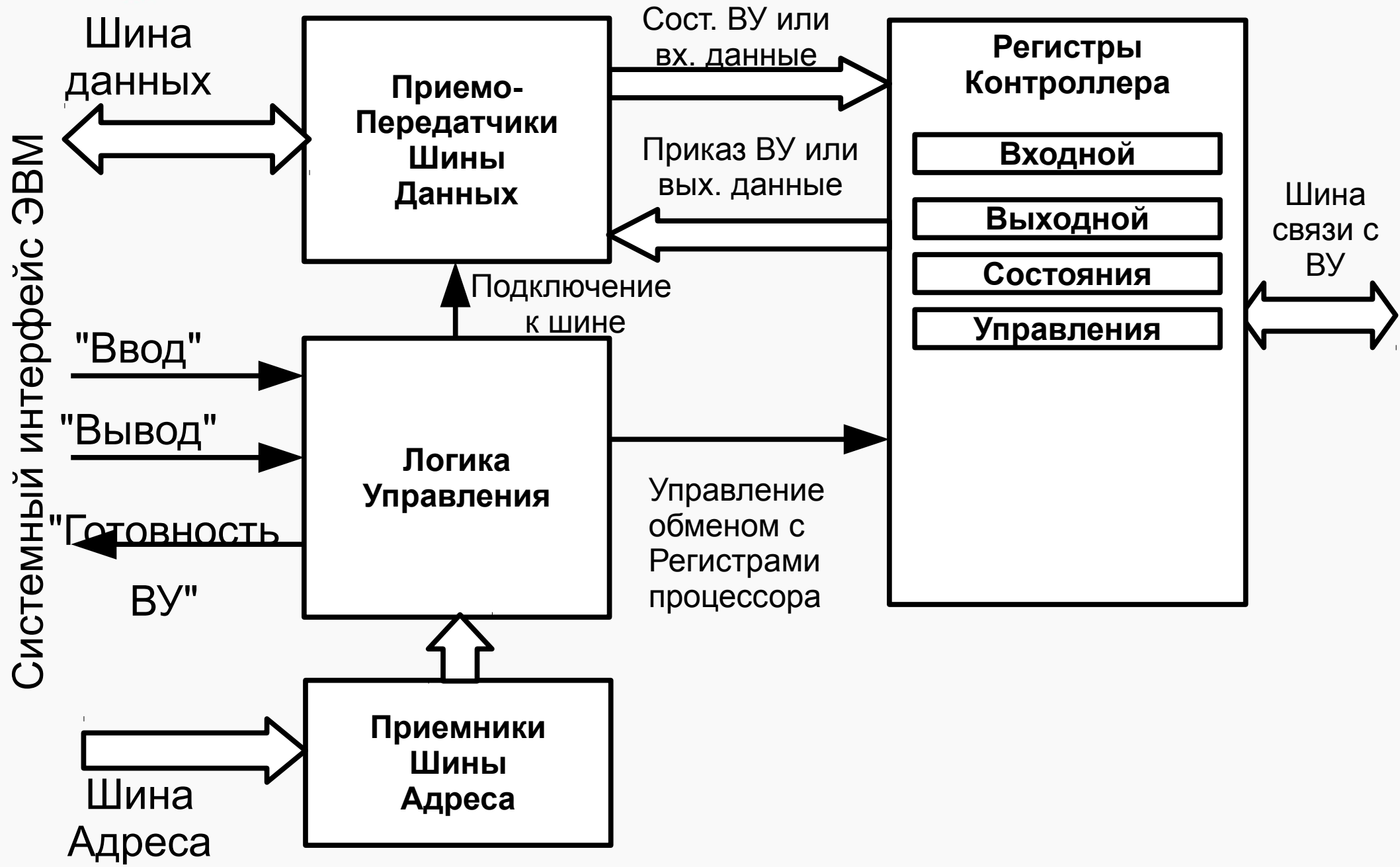


Сопряжение устройств с ЭВМ



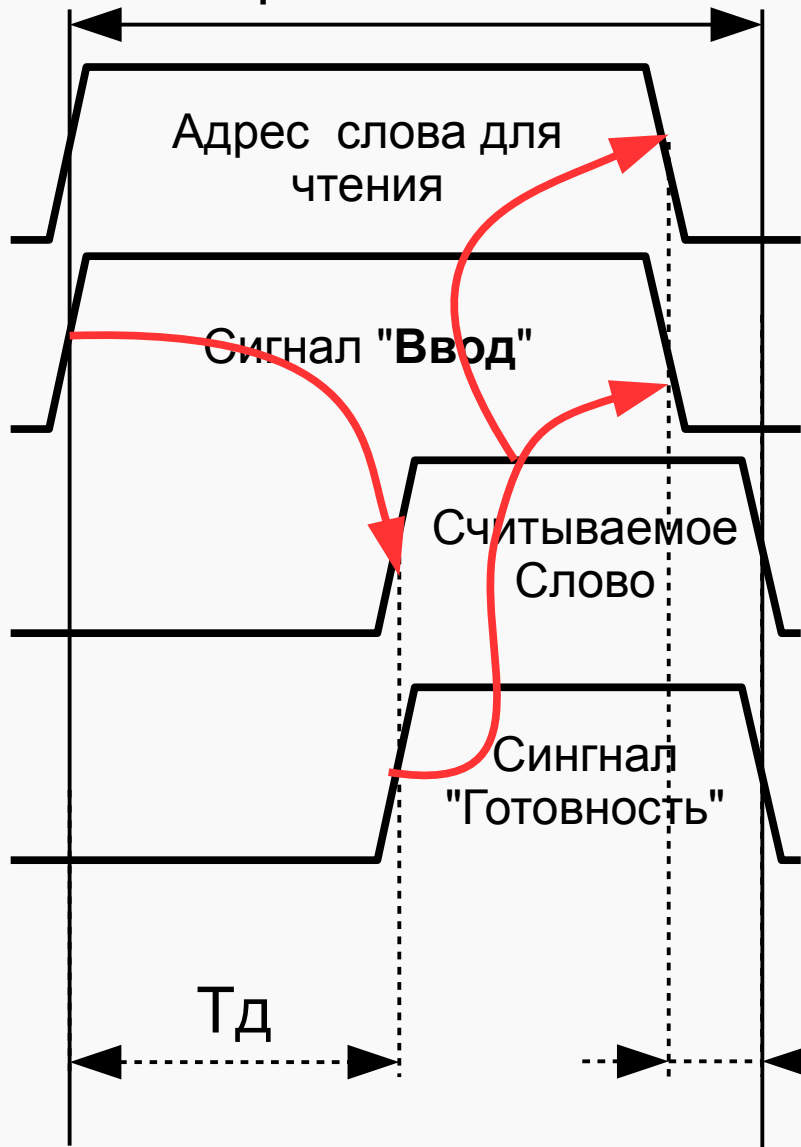
- Два уровня сопряжения:
1. Процессор ↔ контроллеры
 2. Контроллеры ↔ ВУ

Типичная схема программно-управляемого контроллера



Диаграммы ввода-вывода

Цикл обмена



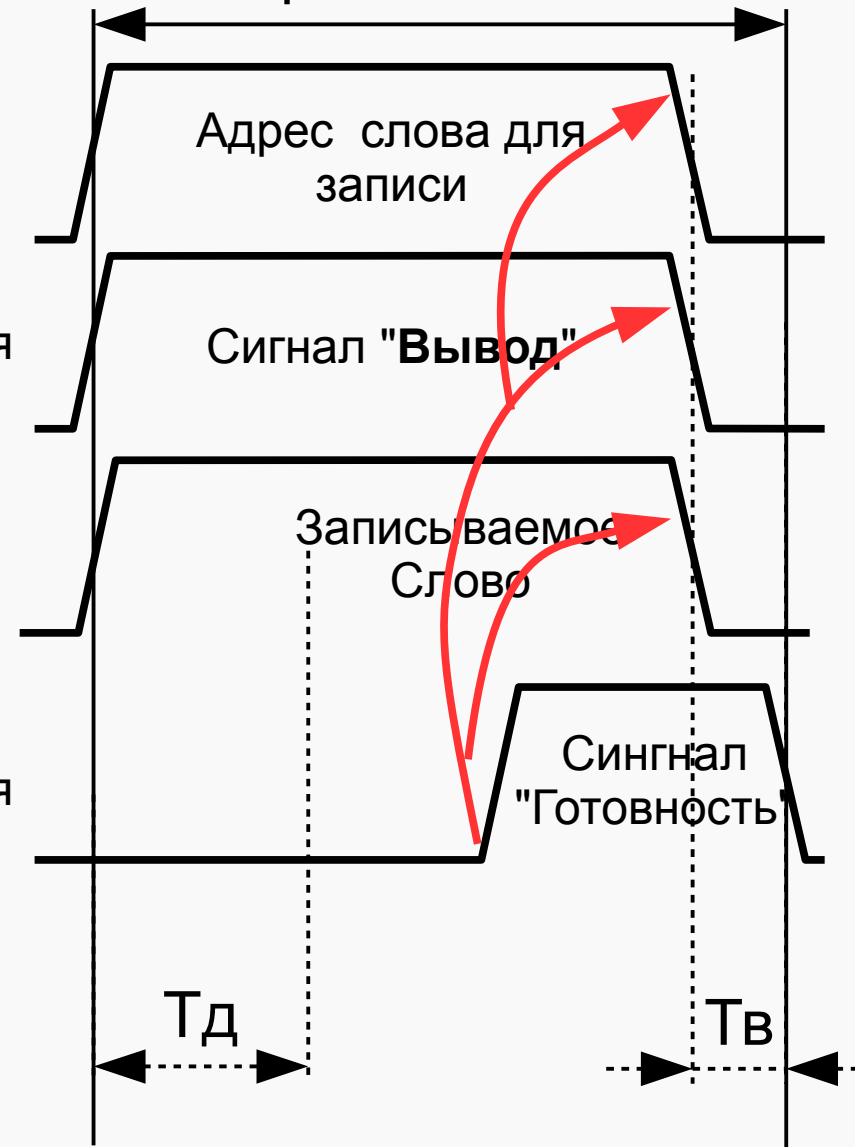
Шина Адреса

Шина Управления

Шина Данных

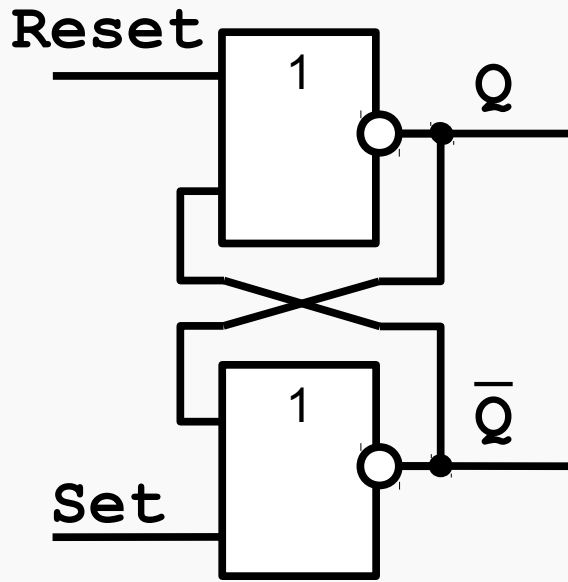
Шина Управления

Цикл обмена



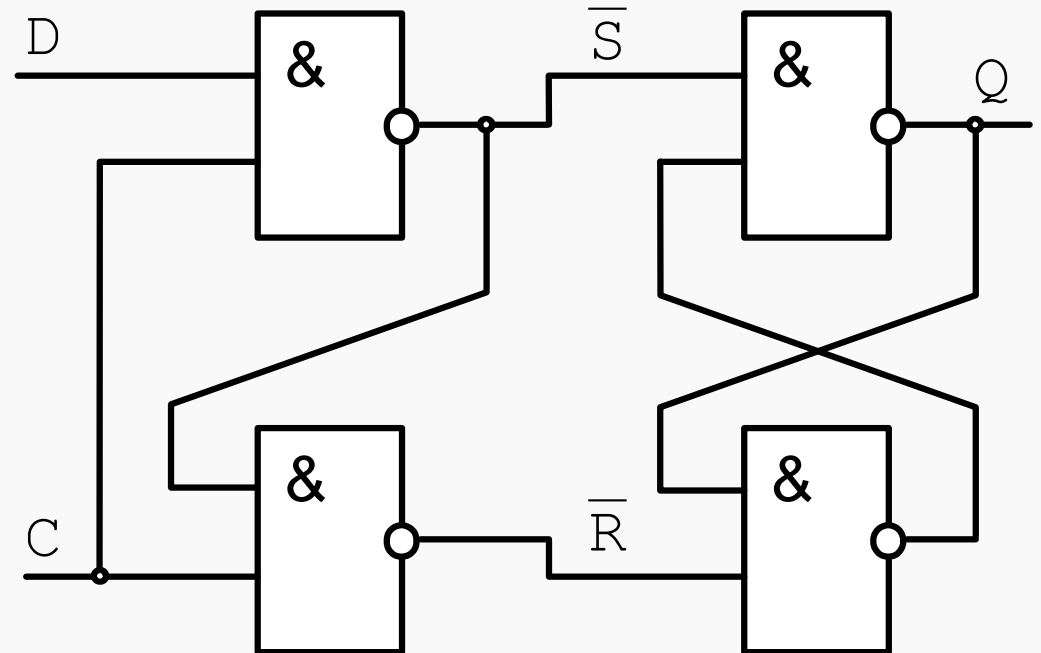
Отступление-напоминание: триггеры

RS-триггер

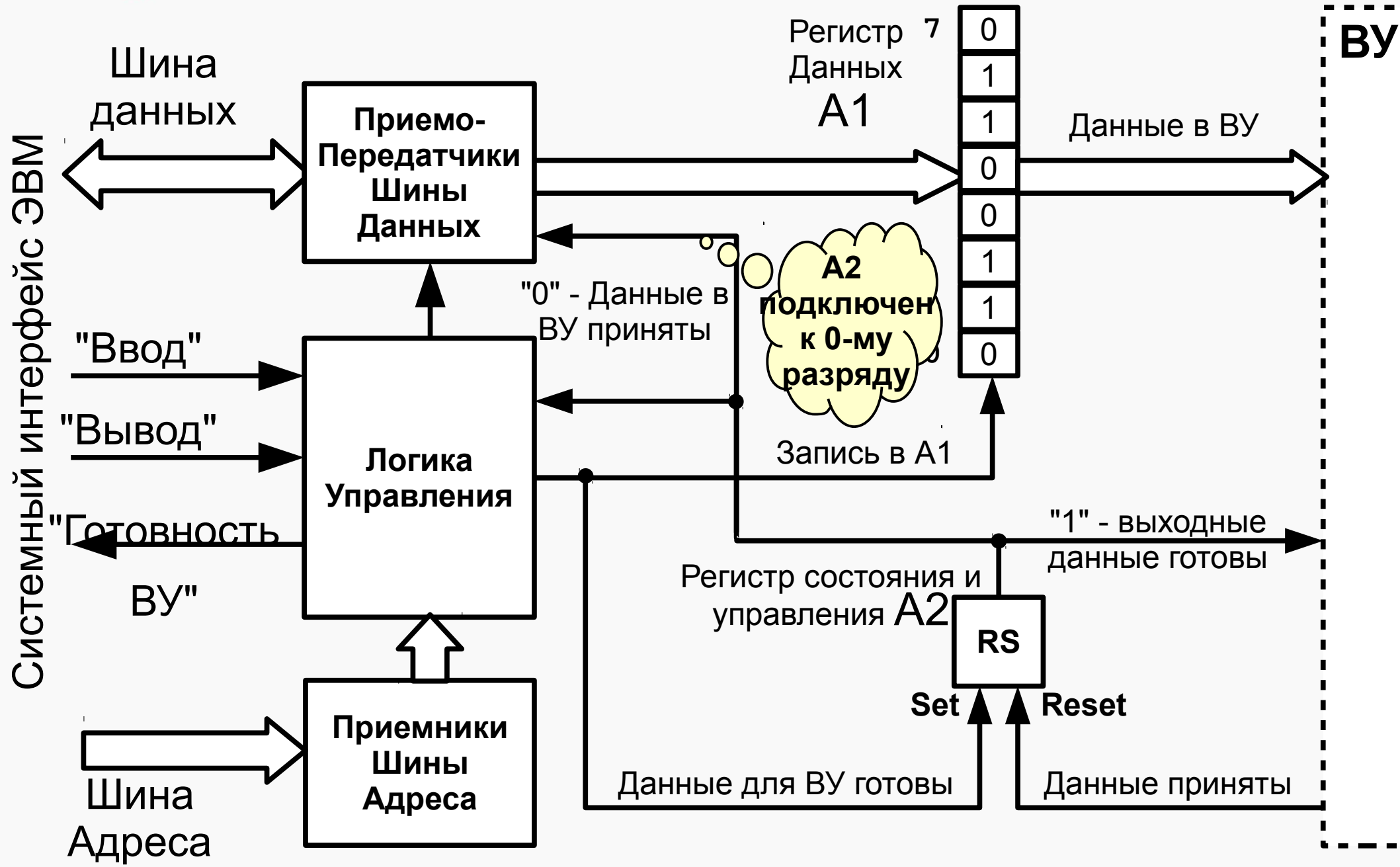


Запрещенная комбинация
 $R=1 S=1$

D-триггер



Контр. передачи параллельного асинхронного интерфейса



Цикл вывода контроллера

Вывод строки символов на устройство

```

ORG      005
ADDR:    WORD      BUF      ; Адрес начала буфера
BUF:     WORD      10 DUP (?) ; десять символов
        WORD      0        ; стоп-символ

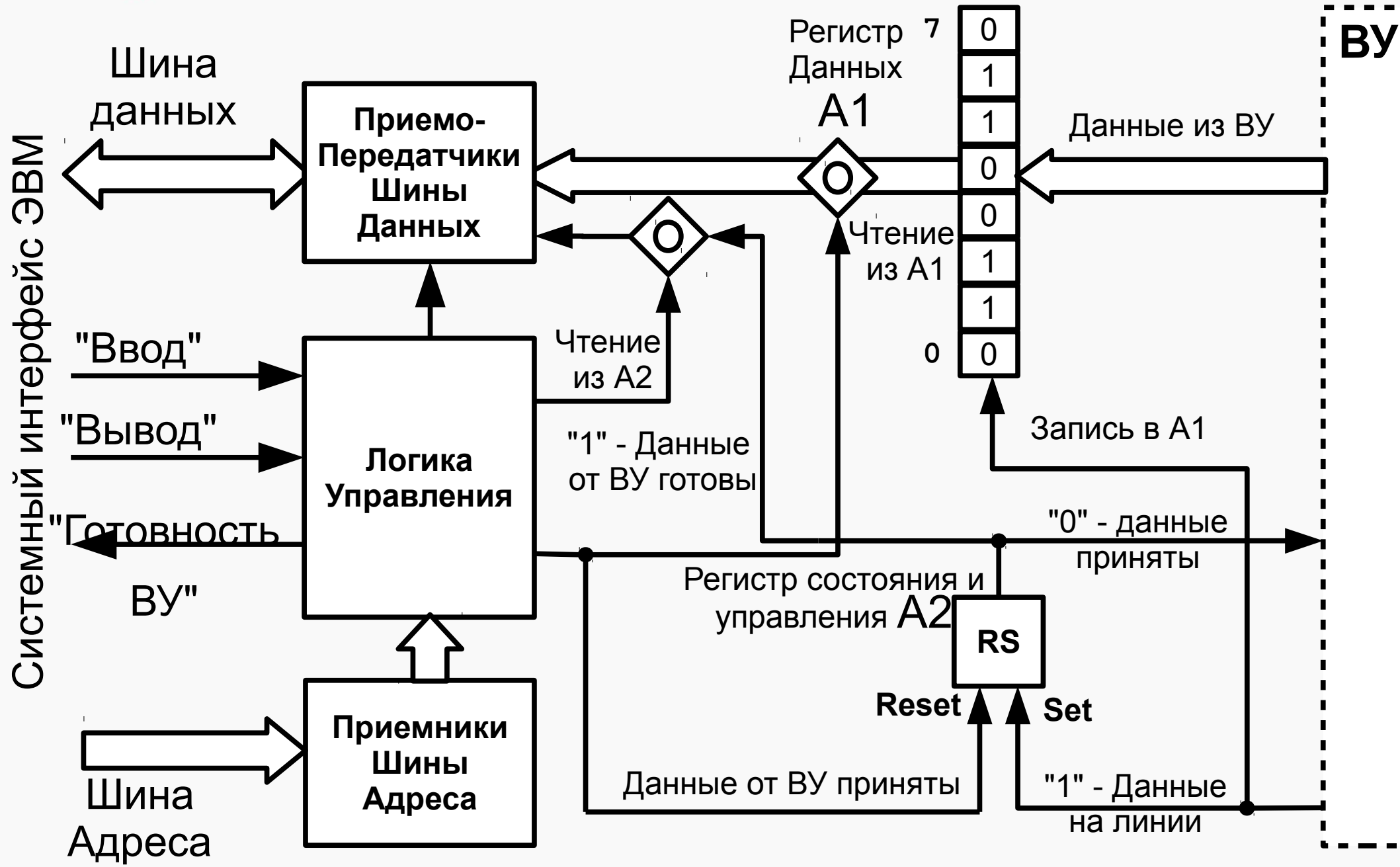
ORG      030
BEGIN:
LOOP:  CLA          ; ввод передаваемого байта в
        ADD      (ADDR) ; аккумулятор
        BEQ      ENDO ; Если стоп-символ, то окончание вывода
        OUT      A1   ; Запись данных в регистр A1
        ; с одновременной установкой A2

SPIN:  IN        A2   ; чтение регистра состояния
        ROR          ; если "1" то ожидаем
        BCS      SPIN ; передачи данных в ВУ
        BR        LOOP ; "0"-данные переданы, следующий символ

ENDO:  ...          ; окончание вывода
        HLT

```

Контр. приема параллельного асинхронного интерфейса



Цикл ввода контроллера

Вывод строки символов на устройство

```

ORG      005
ADDR:    WORD      BUF      ; Адрес начала буфера
BUF:     WORD      10 DUP (?) ; десять слов для хранения символов
MASK:    WORD      00FF     ; маска для очистки старшего байта

ORG      030
BEGIN:
SPIN:  IN      A2      ; Цикл проверки приема данных
           ROR          ; в регистре A1 (Ждем "1" в A2)
           BCS      IN      ; Если "1" – можно вводить символ
           BR       SPIN   ; Если "0" – продолжаем ждать

IN:    IN      A1      ; ввод передаваемого байта
           AND      MASK   ; очистка битов 8–15
           BEQ      ENDO   ; Если стоп-символ, то окончание ввода
           MOV      (ADDR) ; аккумулятор
           BR       SPIN   ; К следующему символу

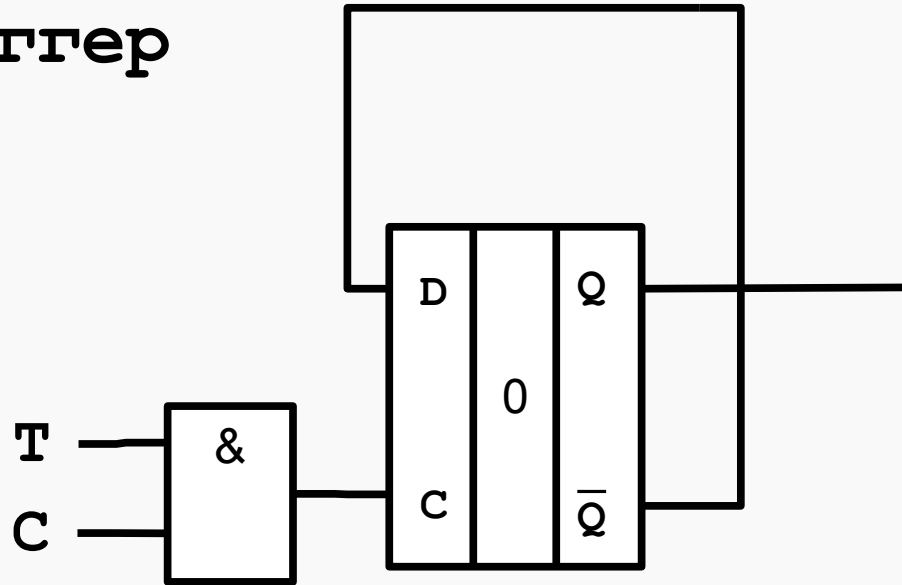
ENDO:  ...          ; окончание ввода
           HLT

```

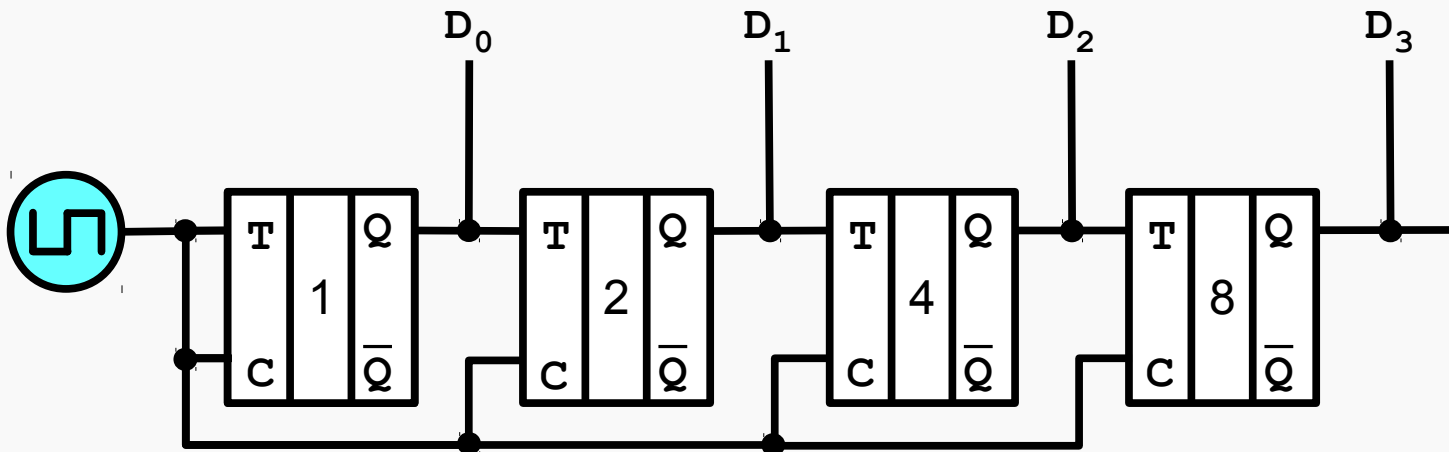
Для юных хакеров: здесь можно выполнить **buffer-overflow** атаку

Отступление: счетчик

T-триггер

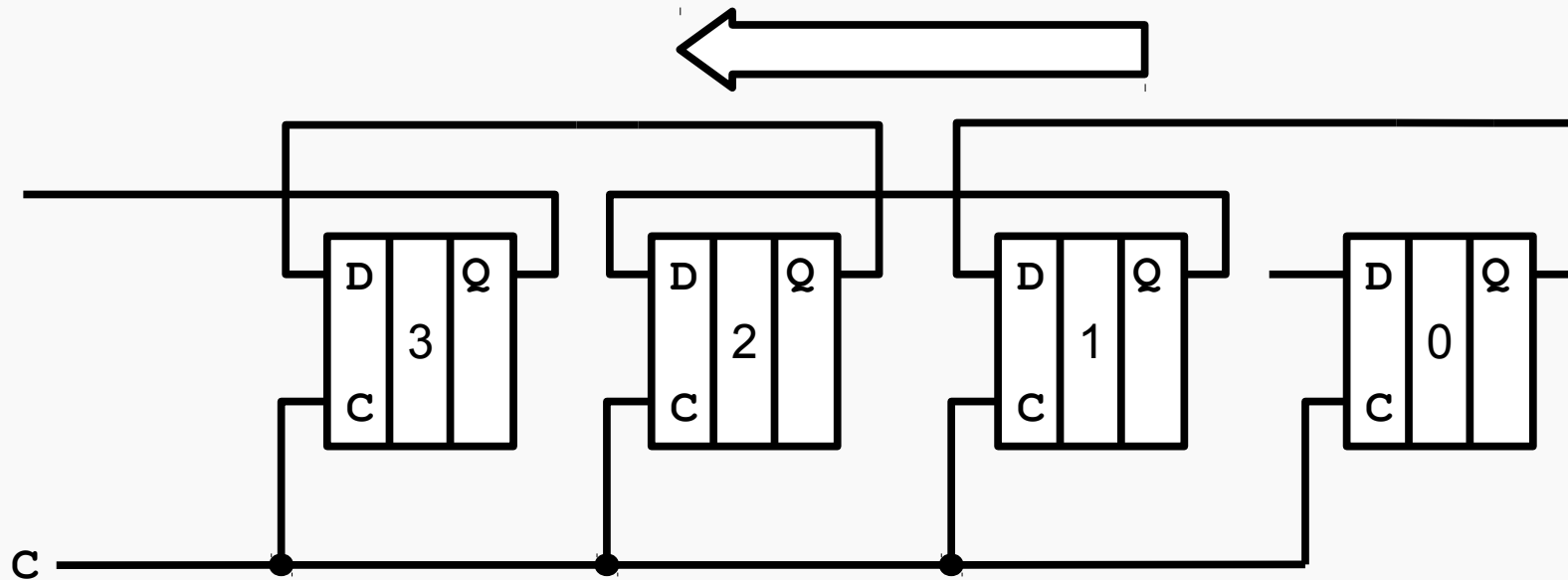
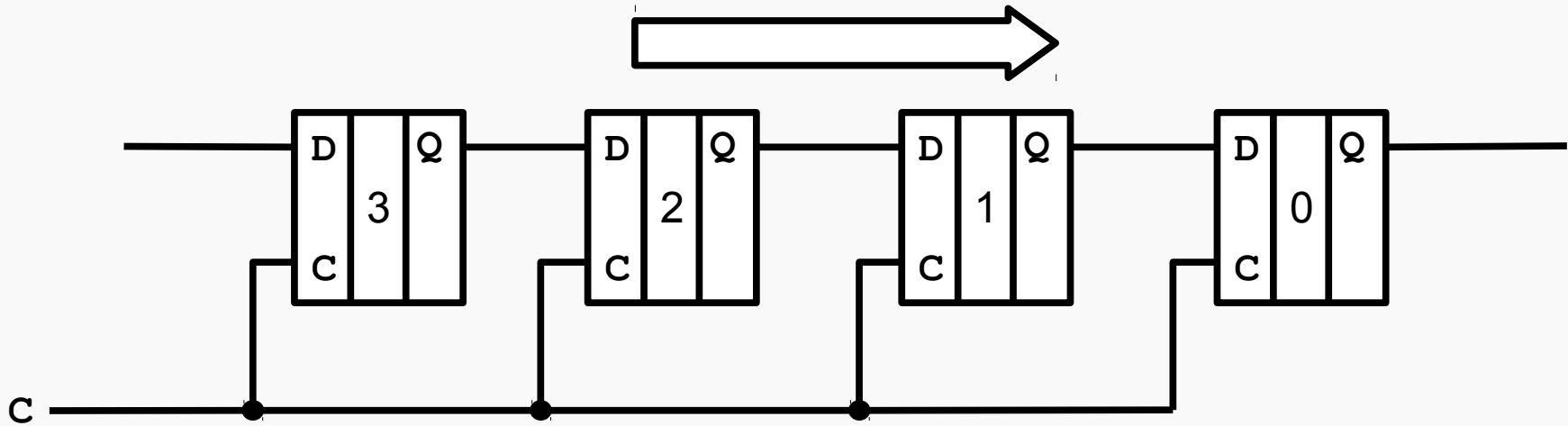


!!!!



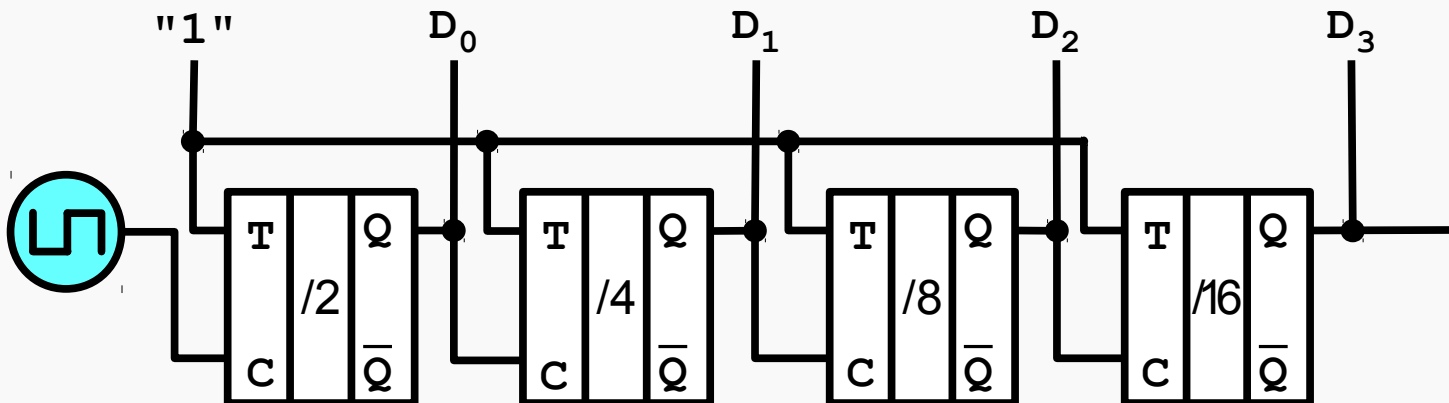
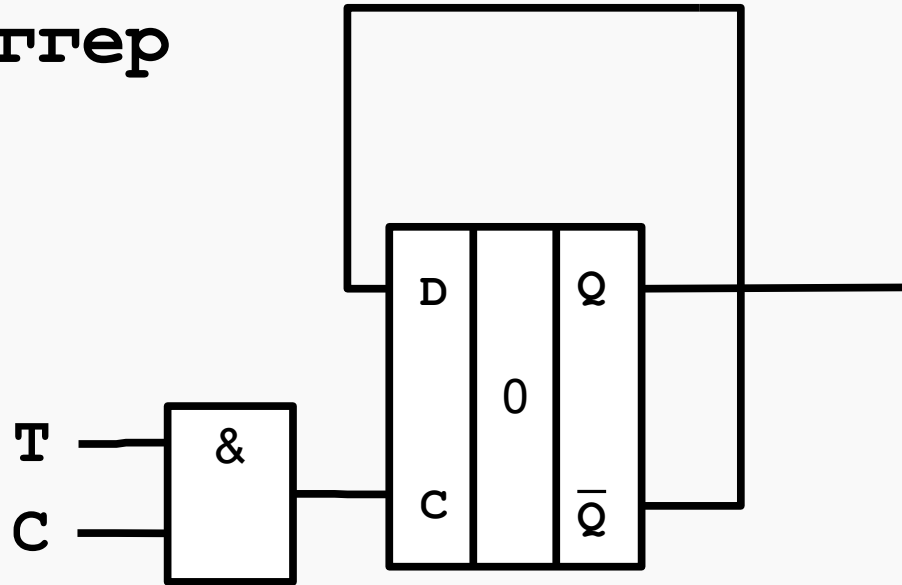
Циклический счетчик			
8	4	2	1
0	0	0	0
0	0	0	1
0	0	1	0
0	0	1	1
0	1	0	0
0	1	0	1
0	1	1	0
0	1	1	1
1	0	0	0
1	0	0	1
1	0	1	0
1	0	1	1
1	1	0	0
1	1	0	1
1	1	1	0
1	1	1	1

Отступление: сдвиговой регистр



Отступление: счетчик

T-триггер

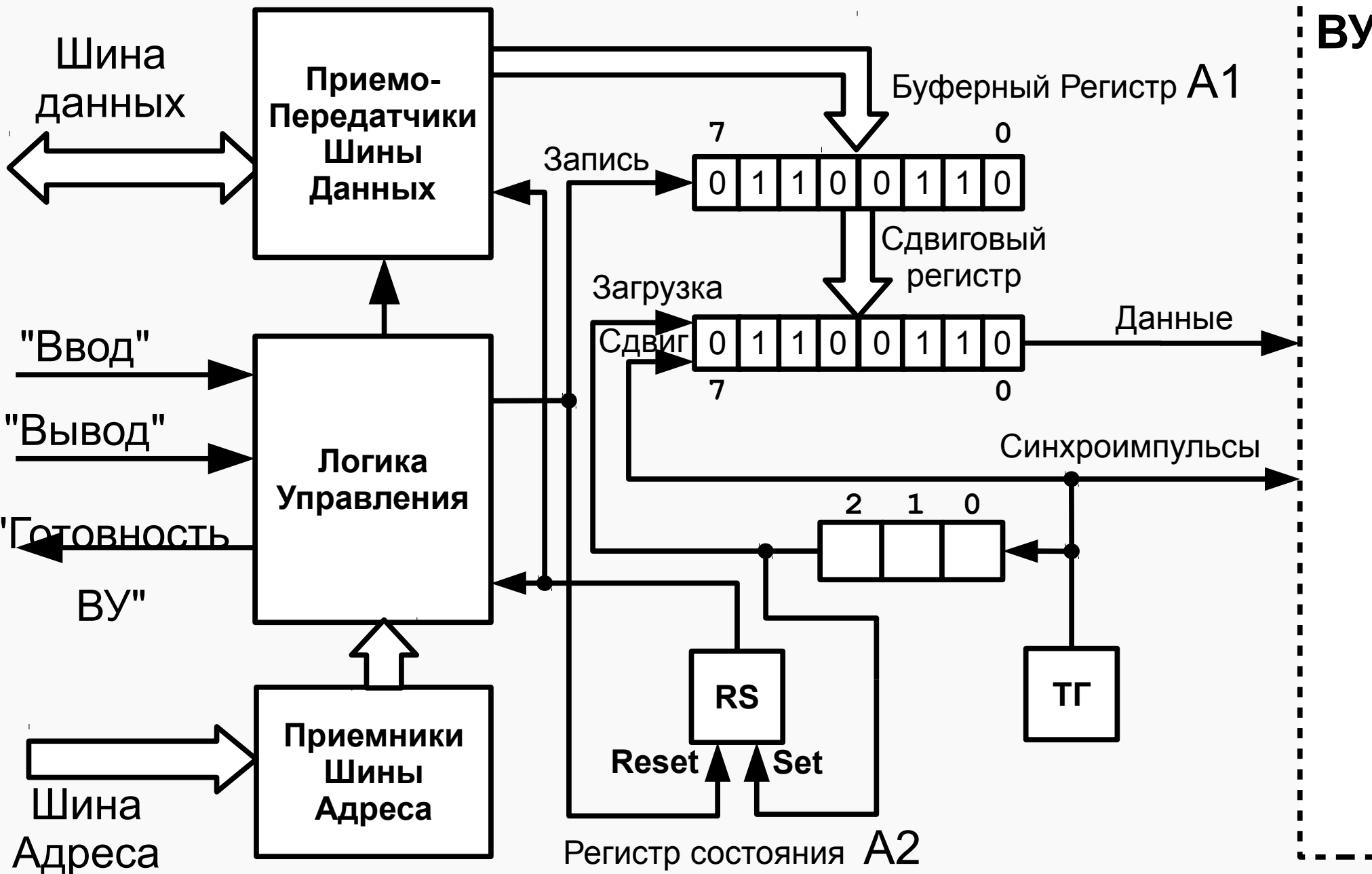


Циклический счетчик			
8	4	2	1
0	0	0	0
0	0	0	1
0	0	1	0
0	0	1	1
0	1	0	0
0	1	0	1
0	1	1	0
0	1	1	1
1	0	0	0
1	0	0	1
1	0	1	0
1	0	1	1
1	1	0	0
1	1	0	1
1	1	1	0
1	1	1	1



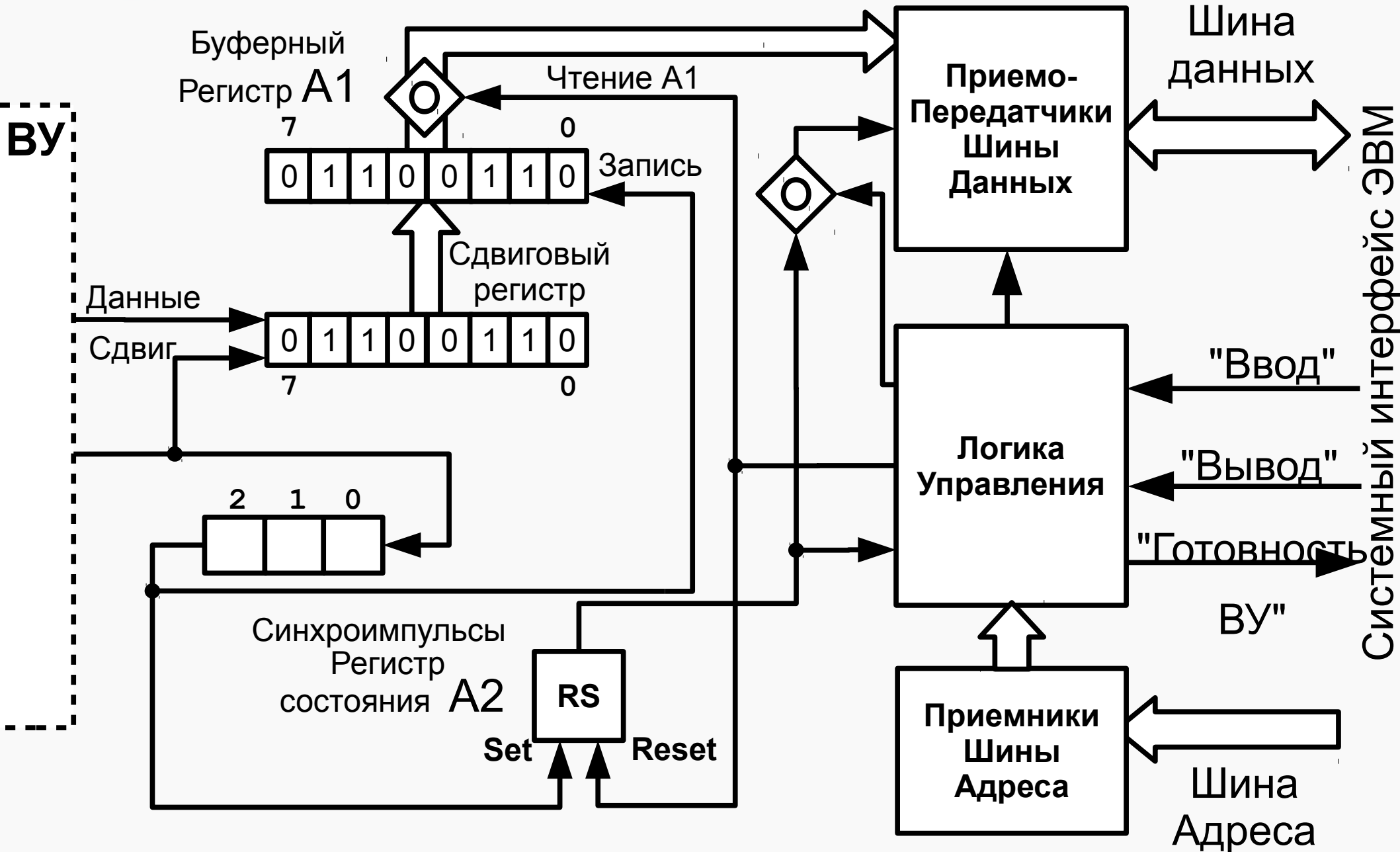
Контролер передачи синхронного последовательного интерфейса

Системный интерфейс ЭВМ



ВУ

Контр. приема синхронного последовательного интерфейса





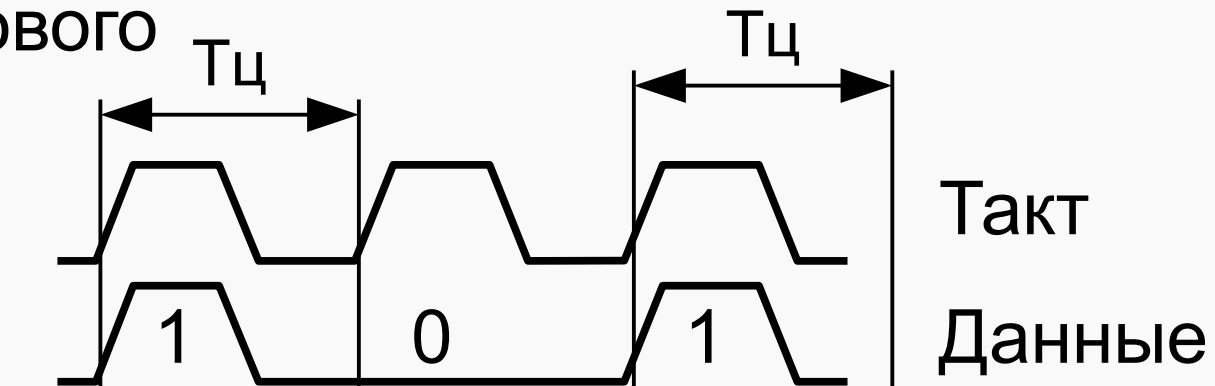
Отступление: последовательный vs параллельный

- Параллельная обмен
 - Быстрее (при одинаковых условиях)
 - Дороже, больше аппаратных ресурсов
 - Менее помехозащитен
 - Дальность передачи меньше
- Последовательный обмен
 - (все наоборот =))

Отступление: синхронный vs асинхронный

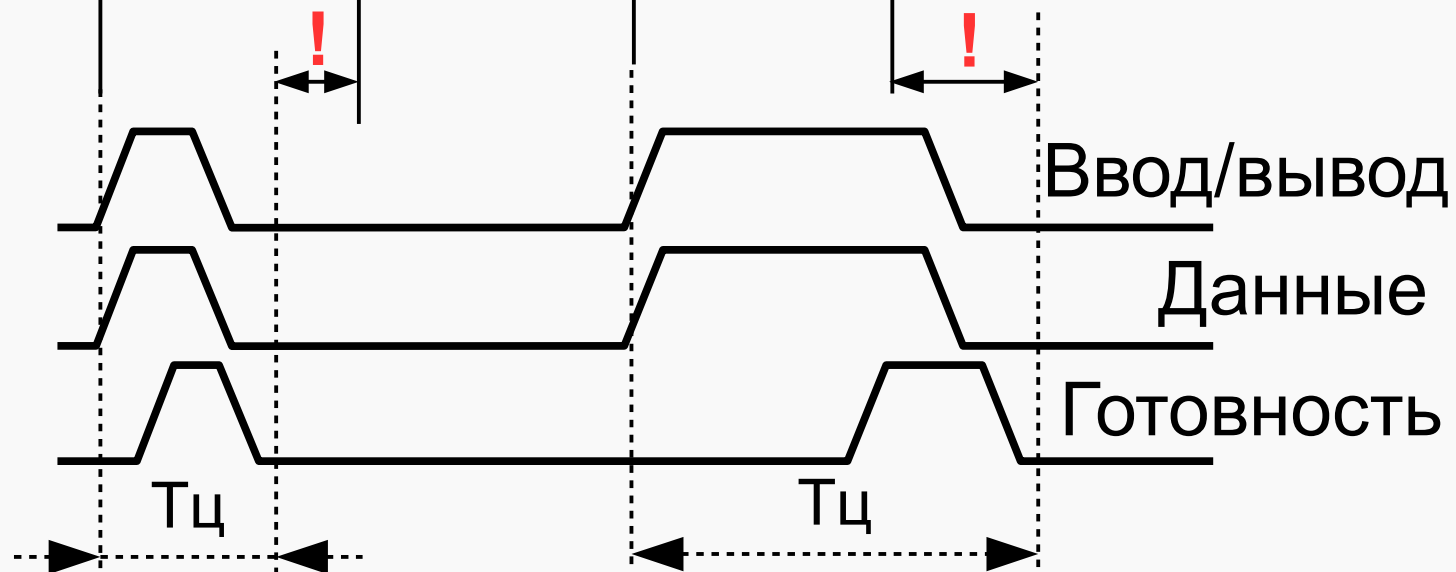
- Синхронная передача данных

- Частота тактового генератора определяет скорость



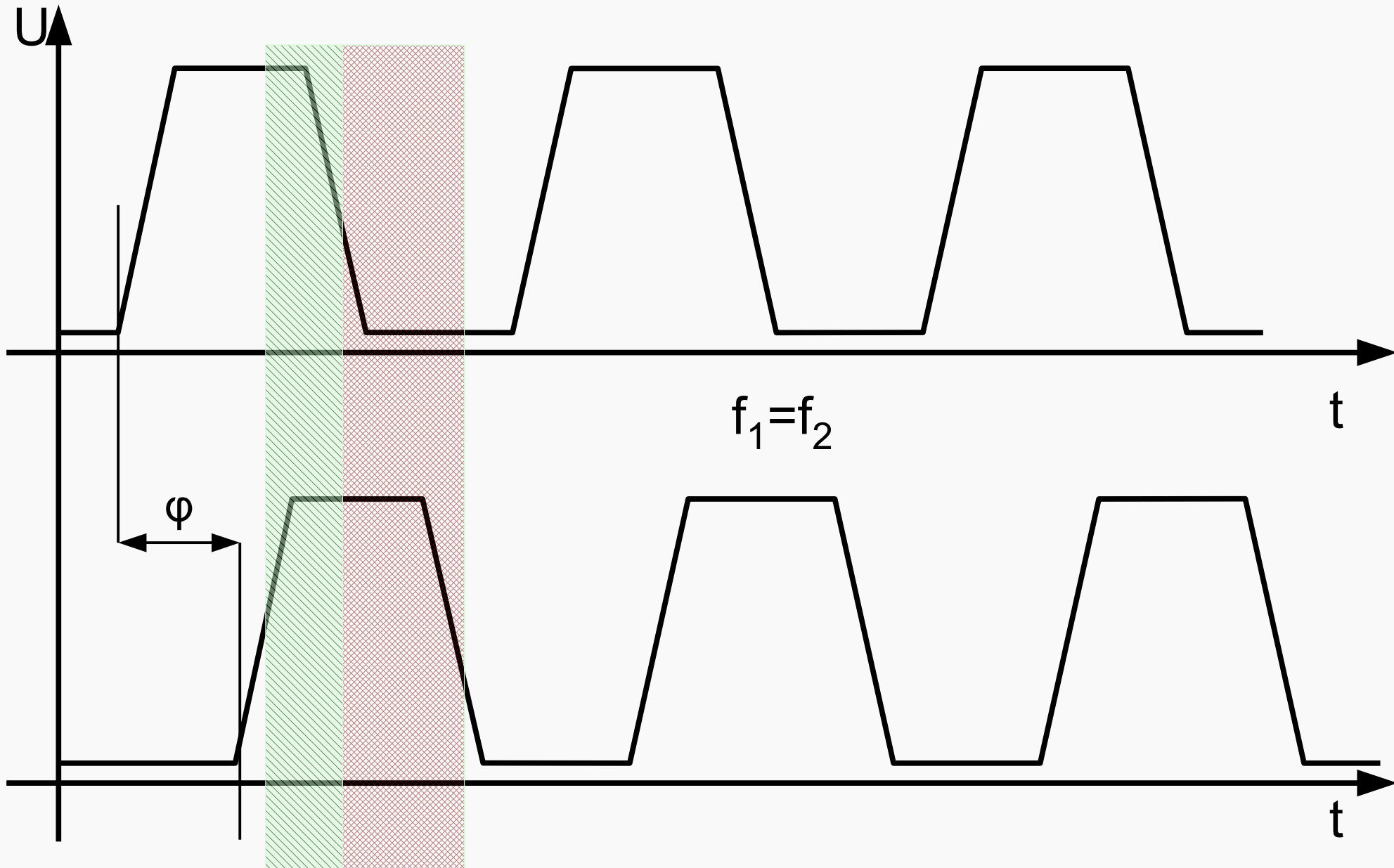
- Асинхронная передача данных

- Скорость определяет сигнал "готовность"

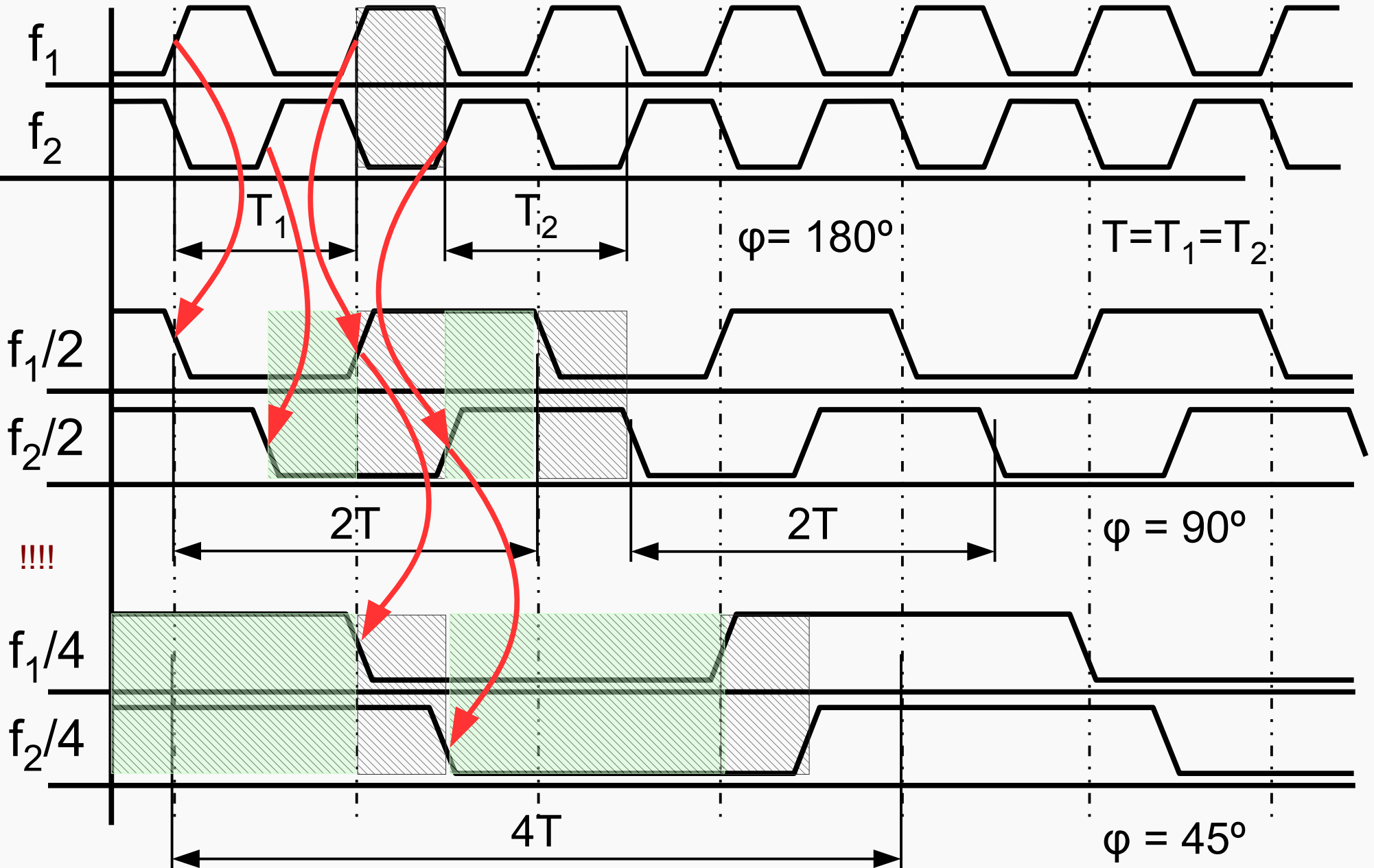


- Хочется сделать каналов передачи еще меньше.
 - В идеале 1-2 "провода"
 - Как бы избавиться от линии синхроимпульсов?
- Что будет, если поместить два разных тактовых генератора одинаковой частоты в приемник и передатчик информации?

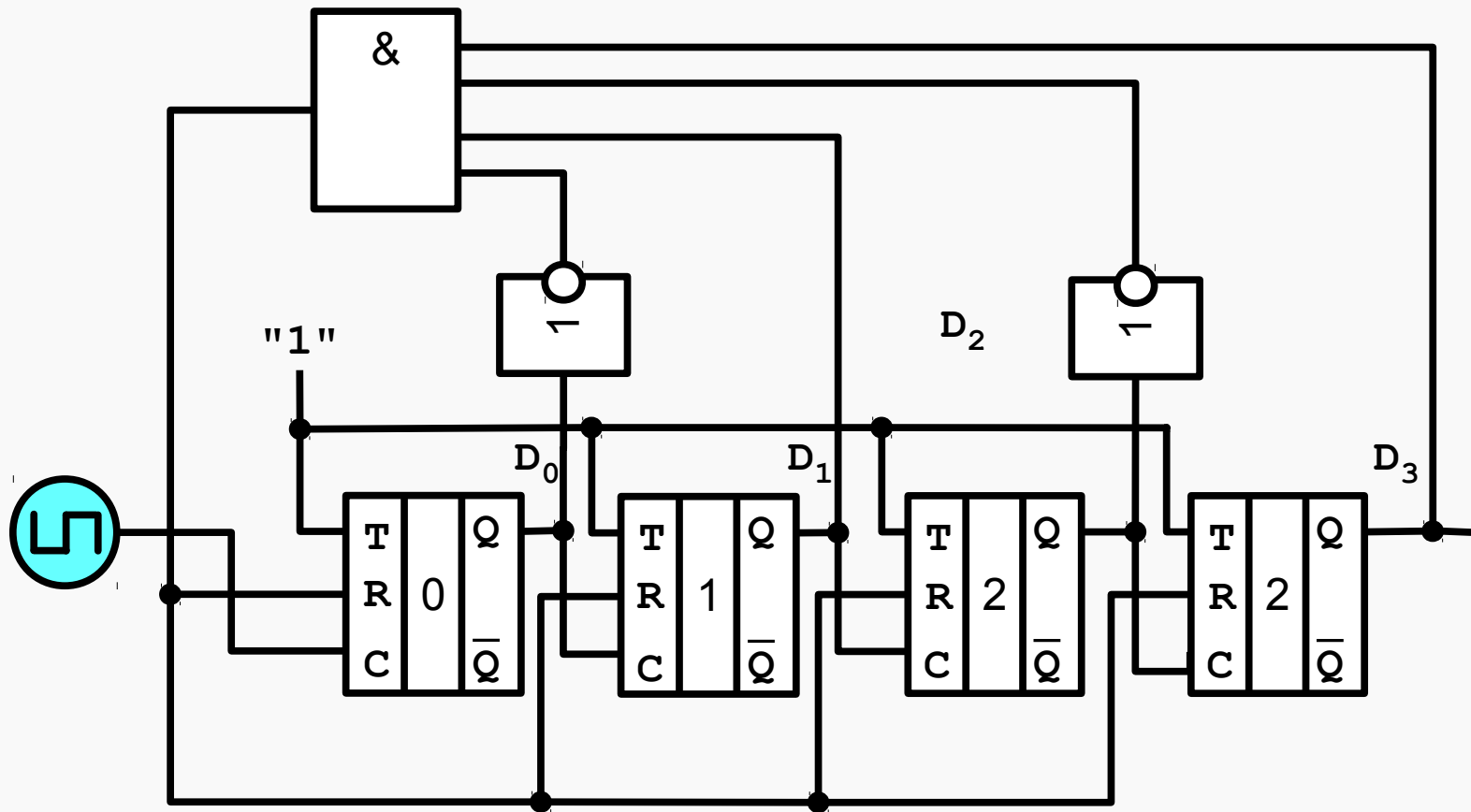
Фазовый сдвиг



Деление частоты



Отступление: счетчик по модулю 10



8	4	2	1
0	0	0	0
0	0	0	1
0	0	1	0
0	0	1	1
0	1	0	0
0	1	0	1
0	1	1	0
0	1	1	1
1	0	0	0
1	0	0	1
1	0	1	0

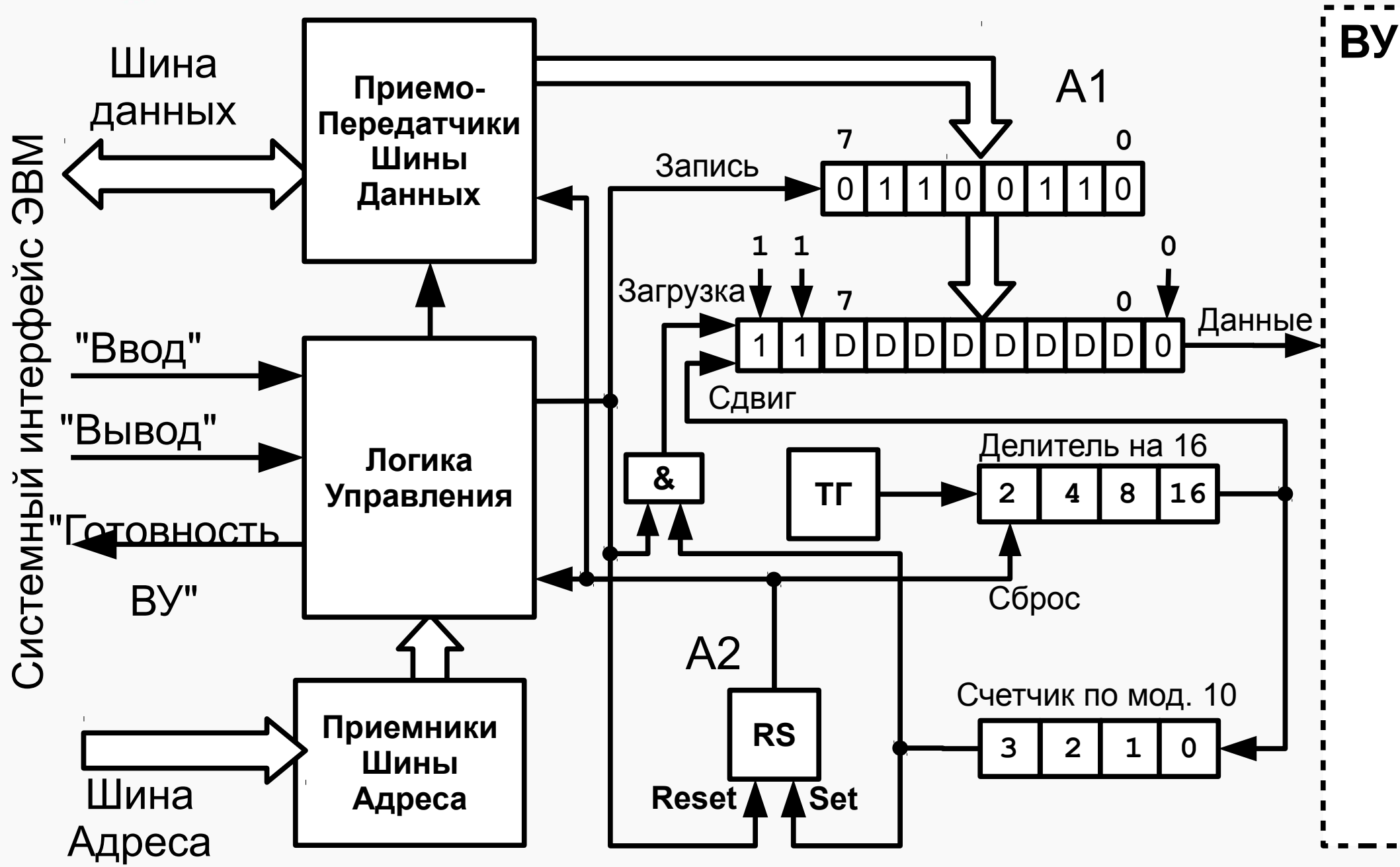
Имеет 10 устойчивых состояний!

Формат кадра асинхронного обмена

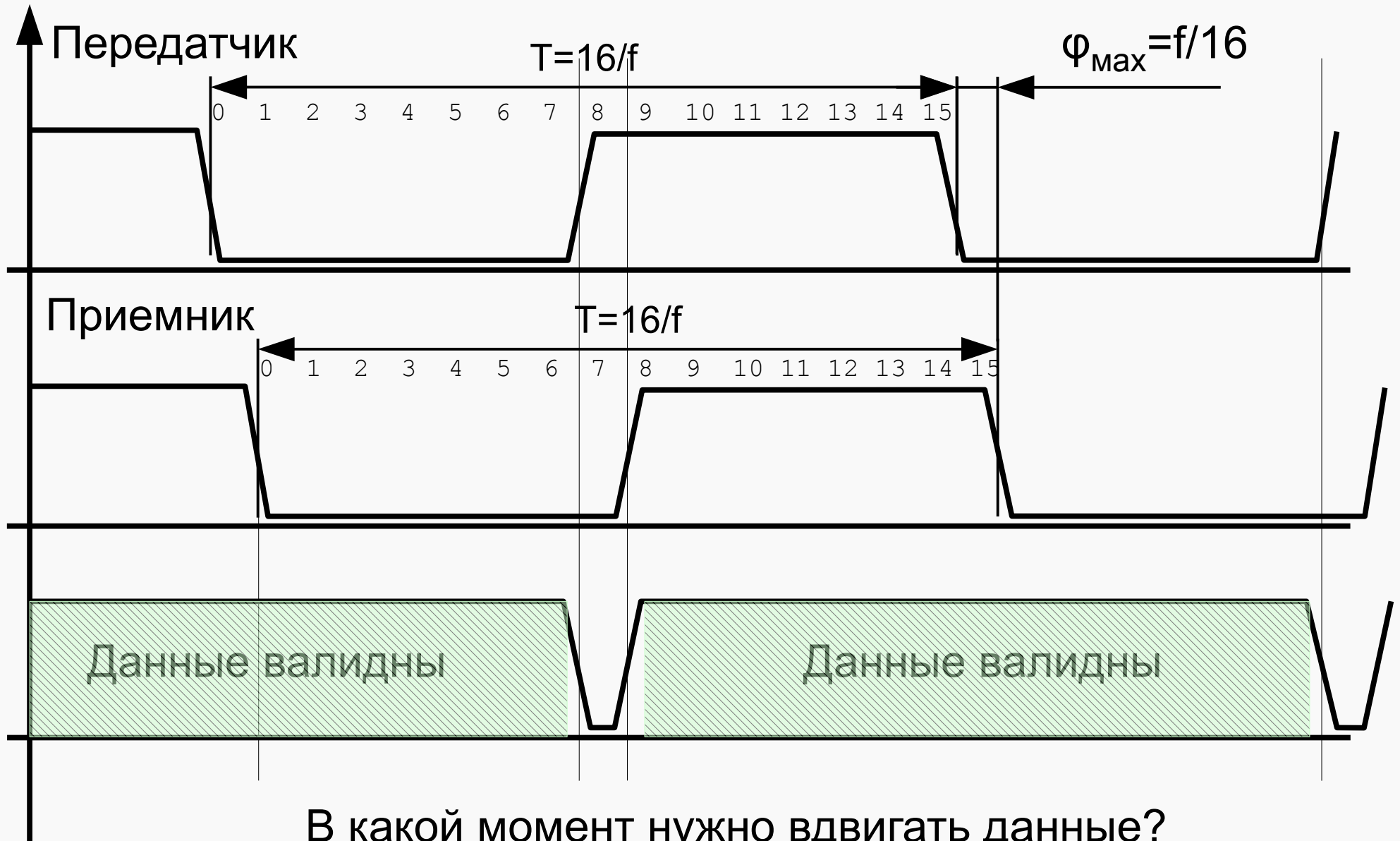
- Пример параметров настройки последовательного асинхронного порта 9600,8,n,1



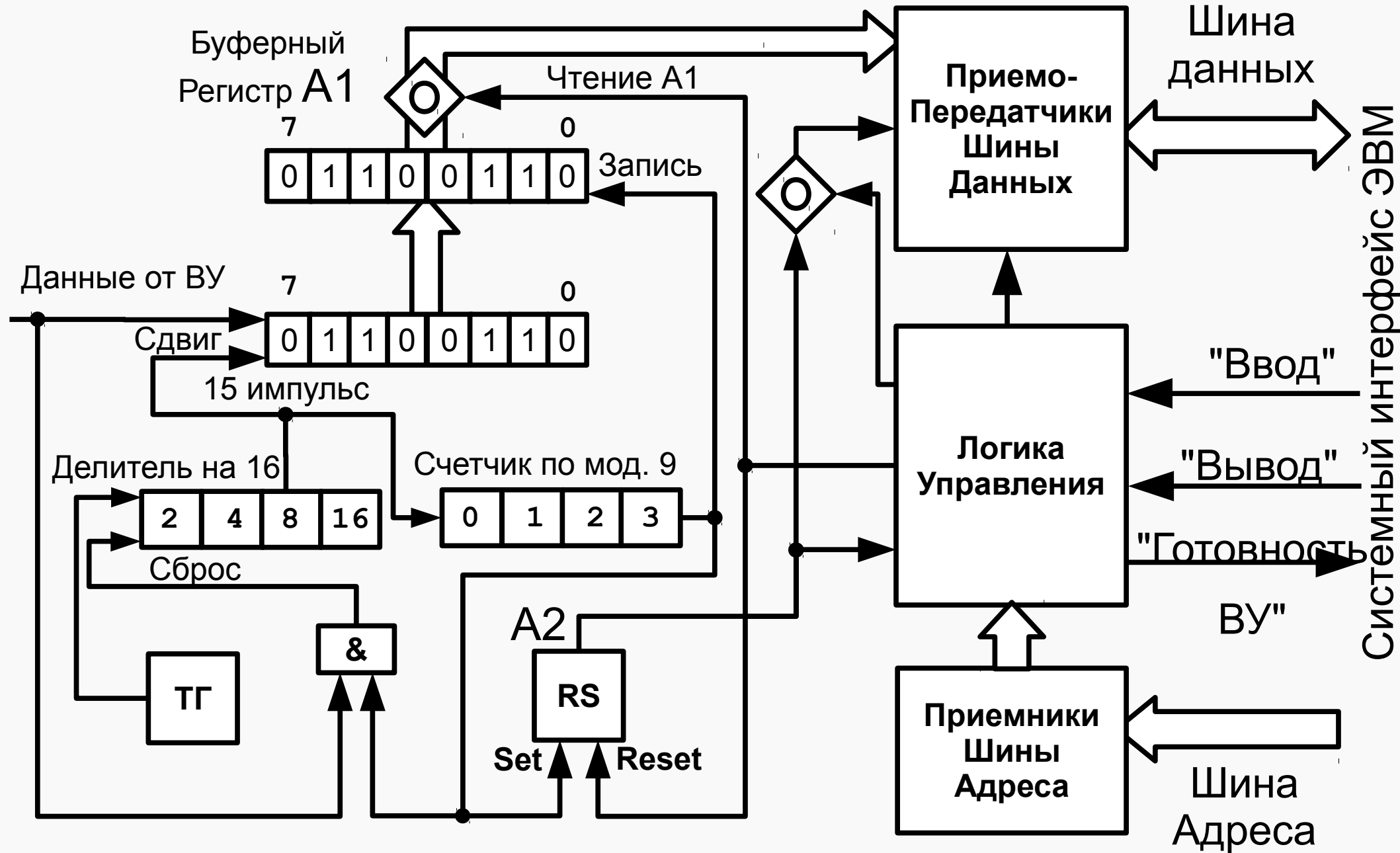
Контр. передачи асинхронного последовательного интерфейса



Надежный прием: выбор правильного момента



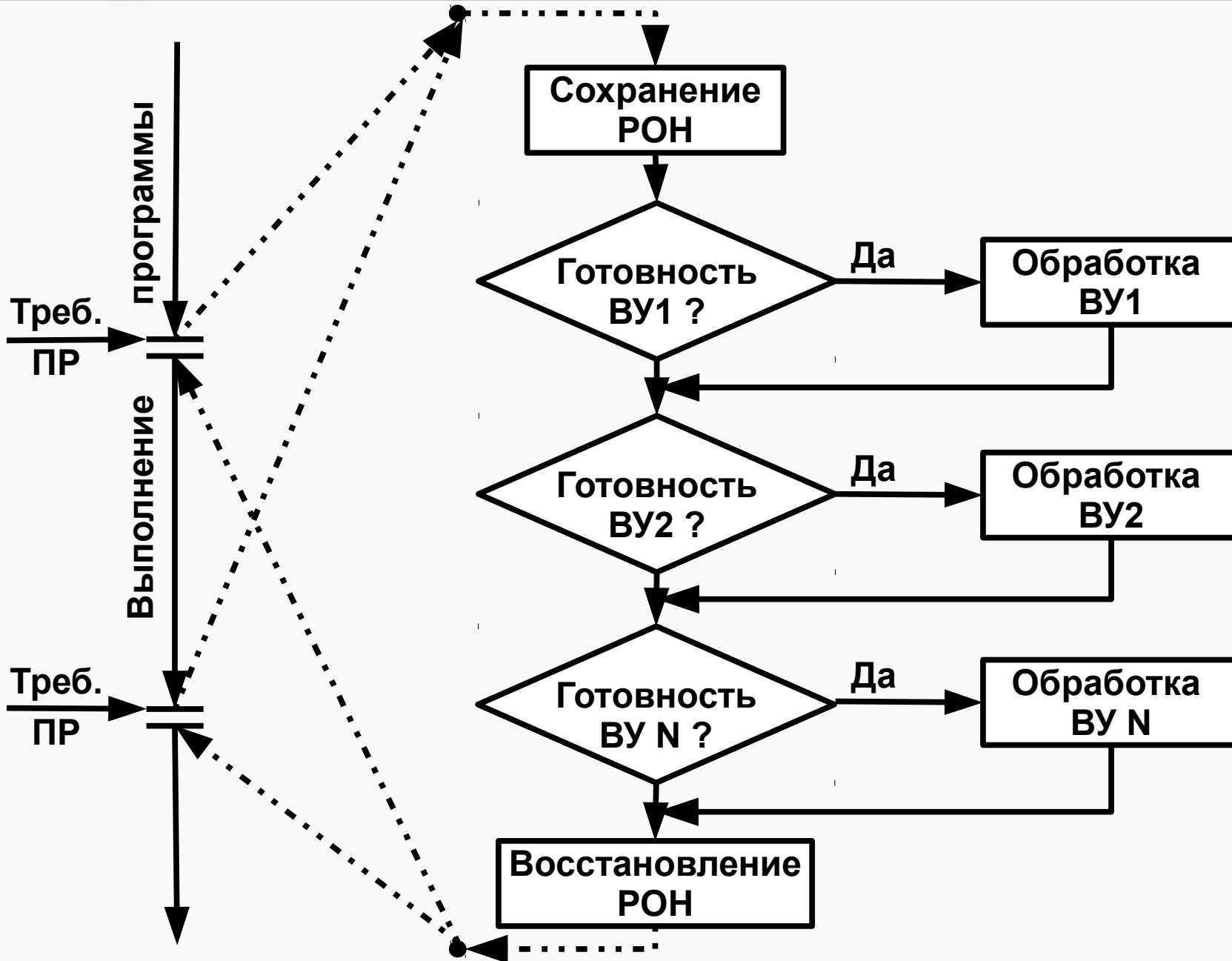
Контр. приема асинхронного последовательного интерфейса



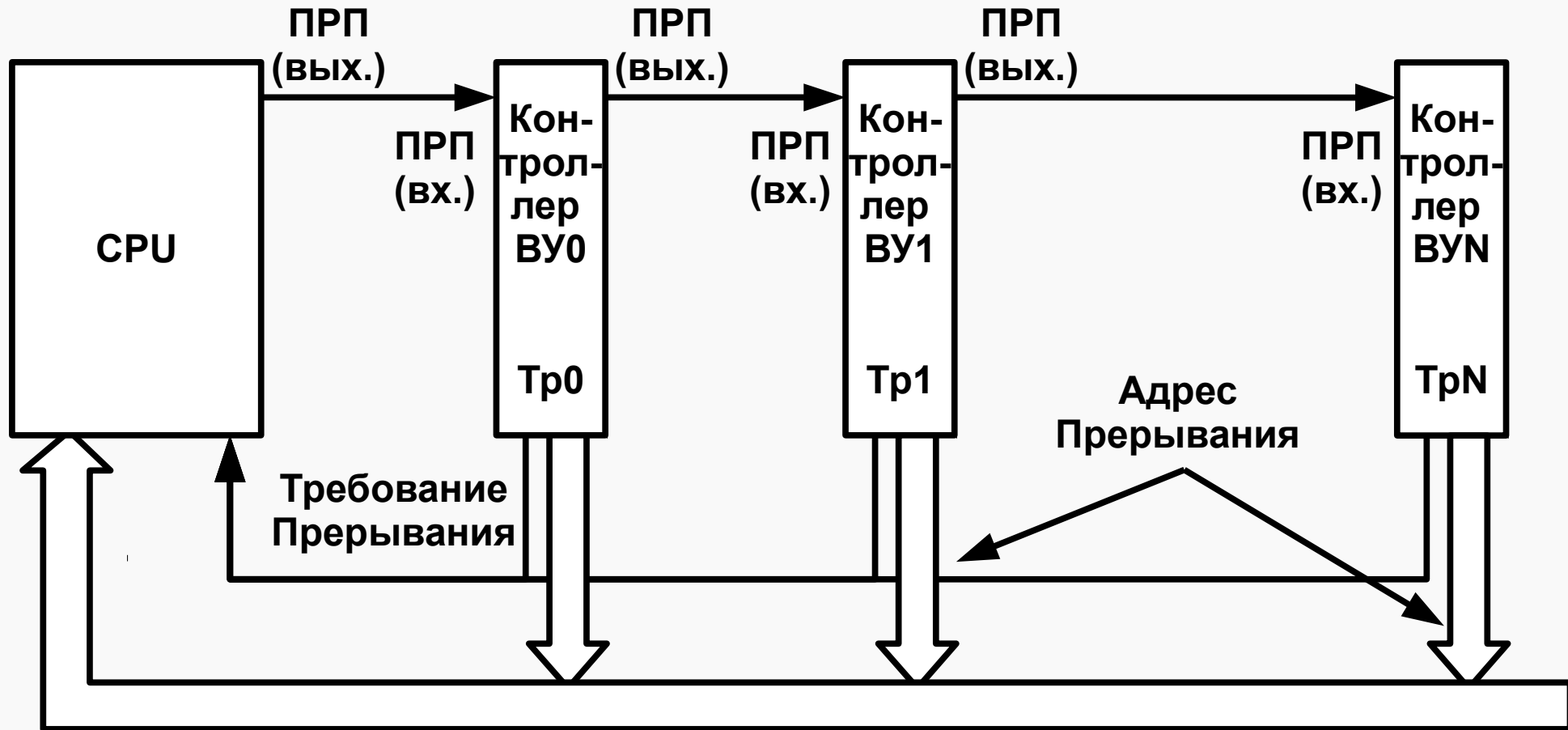
Организация прерываний

- Вектор прерываний — позволяет перейти к программе обработки прерывания.
 - Транслируется в адрес программы обработки
 - Содержит новый регистр состояния в состоянии прерывания
 - Состояние программы сохраняются в стеке
 - Специальная команда возврата (RTI)
- Как учесть приоритет прерываний?
 - В БЭВМ — порядок обработки
 - В современных ЭВМ - диспетчеры прерываний

Логика обработки и приоритет



Аппаратная организация прерываний



ПРП — Предоставление Прерывания. Может быть входной и выходной.

Схема инициации прерывания в контроллере

Системный интерфейс ЭВМ

