

## Aulas 18 & 19

Unidade Aritmética e Lógica (ALU)

– Desenho da unidade de controlo

A unidade de controlo principal *datapath*

Exemplos de funcionamento *datapath* com unidade de controlo

Suporte para a instrução "Jump" ("j")

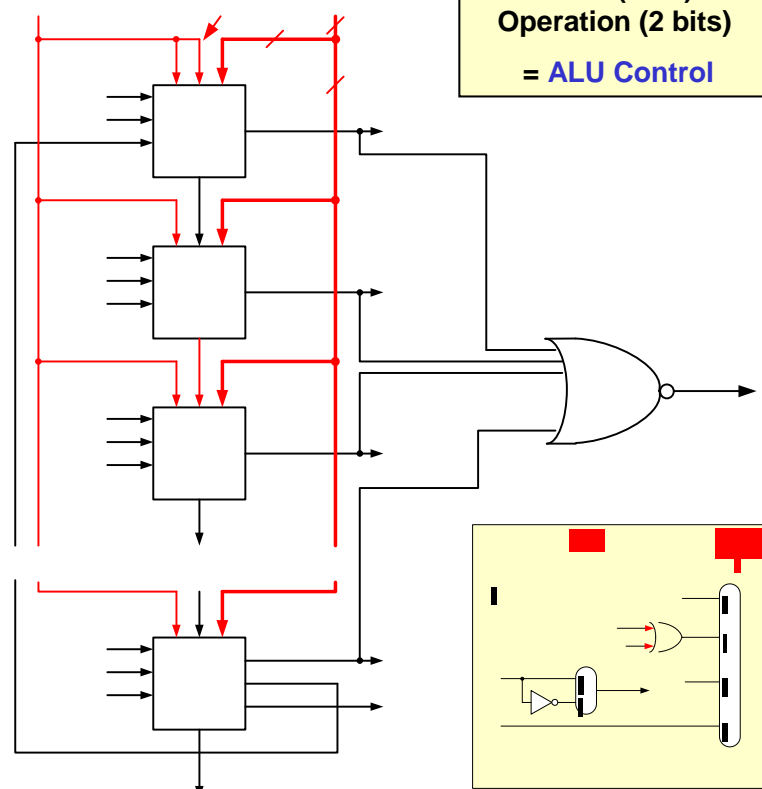
Bernardo Cunha, José Luís Azevedo, Arnaldo Oliveira e Silva

## Desenho da unidade de controlo da ALU

Os sinais de controlo directo da ALU ficam reduzidos a três, uma vez que os sinais **Binvert** e **Carry In** podem ser combinados num sinal (ver aula 10)

ALU Control	ALU Action
0 0 0	And
0 0 1	Or
0 1 0	Add
1 1 0	Subtract
1 1 1	Set if less than

Bit "Binvert"



## Desenho da unidade de controlo da ALU

As instruções básicas que fazem uso da ALU são:

**Load e store**  $i, \%o$  para calcular o endereço  $i, \%o$  da memória externa

**Branch if equal / not equal**  $jz$  para determinar se os operandos são iguais ou diferentes

İçerik: Aritmetik ve İstatistiksel para effectuer a respectiva operaiões

İçerik A operatörünün a'ya realize edilmesi ALU bağımlıdır:

dos campos *opcode* e *funct* nas instruções aritméticas e lógicas  
tipo R: `ALUControl=f(opcode, funct)`

i; % do campo **opcode** nas restantes instru; %o; % de **Control=f(opcode)**

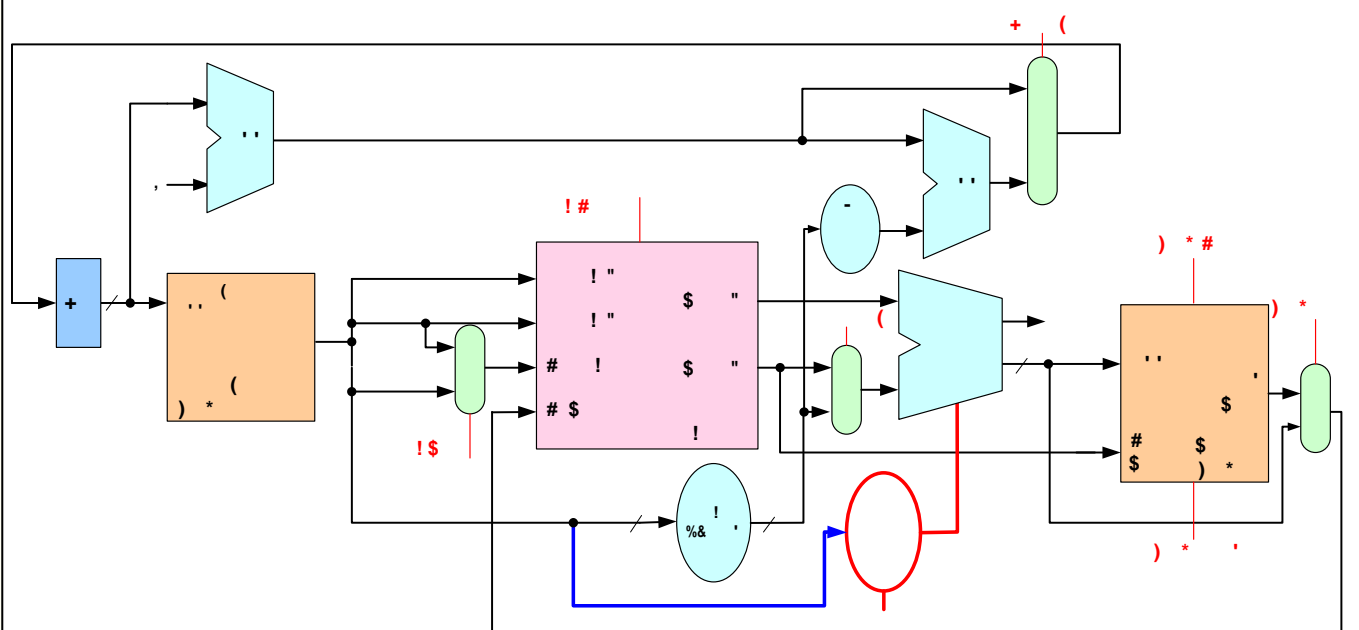
Assim, a geração dos sinais de controlo da Adu é realizada em dois níveis:

Op = g (opcode)

ALUControl = f (ALUOp, funct)

## Desenho da unidade de controlo da ALU

O *datapath* com a **unidade de controlo da ALU**:



## Desenho da unidade de controlo da ALU

A relação entre o tipo de instrução, a operação efectuada pela ALU e os sinais de controlo da mesma pode, assim, ser resumida pela seguinte tabela:

ALU Control	ALU Action
0 0 0	And
0 0 1	Or
0 1 0	Add
1 1 0	Subtract
1 1 1	Set if less than

Instruction	OpCode	Funct	ALU Action	ALUOp	ALU Control
load word	100011 ( "lw" )	xxxxxx	add	00	010
store word	101011 ( "sw" )	xxxxxx	add	00	010
addi	001000 ( "addi" )	xxxxxx	add	00	010
branch if equal	000100 ( "beq" )	xxxxxx	subtract	01	110
add	000000 (R-Type)	100000	add	10	010
subtract	000000 (R-Type)	100010	subtract	10	110
and	000000 (R-Type)	100100	and	10	000
or	000000 (R-Type)	100101	or	10	001
set if less than	000000 (R-Type)	101010	set if less than	10	111
set if less than imm	001010 ( "slti" )	xxxxxx	set if less than	11	111

## Desenho da unidade de controlo da ALU

A unidade de controlo (combinatória) pode ser sintetizada a partir de uma tabela de verdade que identifique o valor dos **6 sinais de controlo da ALU** em função dos 6 bits do **funct** e dos dois bits do **ALUOp**.

FUNCT						ALUOp1	ALUOp0	ALU Control	
F5	F4	F3	F2	F1	F0				
X	X	X	X	X	X	0	0	0 1 0	(add)
X	X	X	X	X	X	0	1	1 1 0	(sub)
X	X	0	0	0	0	1	0	0 1 0	(add)
X	X	0	0	1	0	1	0	1 1 0	(sub)
X	X	0	1	0	0	1	0	0 0 0	(and)
X	X	0	1	0	1	1	0	0 0 1	(or)
X	X	1	0	1	0	1	0	1 1 1	(slt)
X	X	X	X	X	X	1	1	1 1 1	(slt)

## Desenho da unidade de controlo da ALU

F5	F4	F3	F2	F1	F0	ALUOp1	ALUOp0	AC2	AC1	AC0
X	X	X	X	X	X	0	0	0	1	0
X	X	X	X	X	X	0	1	1	1	0
X	X	0	0	0	0	1	0	0	1	0
X	X	0	0	1	0	1	0	1	1	0
X	X	0	1	0	0	1	0	0	0	0
X	X	0	1	0	1	1	0	0	0	1
X	X	1	0	1	0	1	0	1	1	1
X	X	X	X	X	X	1	1	1	1	1

AC2 - ALU Control 2

F3	F2	F1	F0	ALUOp1	ALUOp0	AC2 =
X	X	X	X	X	1	ALUOp0 +
X	X	1	X	1	0	ALUOp1 . ALUOp0 \ . F1

AC1 - ALU Control 1

F3	F2	F1	F0	ALUOp1	ALUOp0	AC1 =
X	X	X	X	0	X	ALUOp1 \ +
X	0	X	X	1	X	ALUOp1 . F2 \ +
X	X	X	X	1	1	ALUOp1 . ALUOp0

AC0 - ALU Control 0

F3	F2	F1	F0	ALUOp1	ALUOp0	AC0 =
X	X	X	1	1	X	ALUOp1 . F0 +
1	X	X	X	1	X	ALUOp1 . F3 +
X	X	X	X	1	1	ALUOp1 . ALUOp0

## Desenho da unidade de controlo da ALU

AC2 - ALU Control 2

F5	F4	F3	F2	F1	F0	ALUOp1	ALUOp0	AC2 =
X	X	X	X	X	X	X	1	ALUOp0 +
X	X	X	X	1	X	1	0	ALUOp1 . ALUOp0 \ . F1

AC1 - ALU Control 1

F5	F4	F3	F2	F1	F0	ALUOp1	ALUOp0	AC1 =
X	X	X	X	X	X	0	X	ALUOp1 \ +
X	X	X	0	X	X	1	X	ALUOp1 . F2 \ +
X	X	X	X	X	X	1	1	ALUOp1 . ALUOp0

AC0 - ALU Control 0

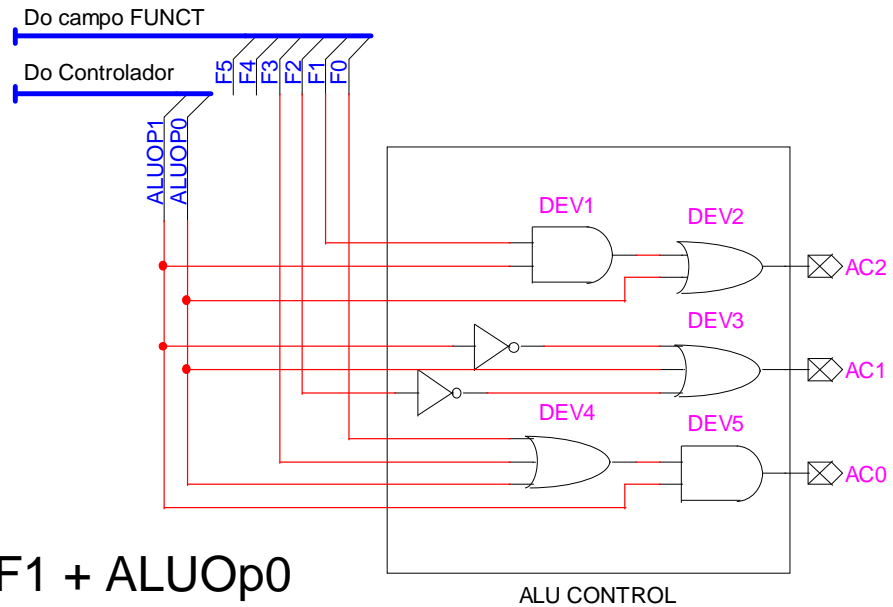
F5	F4	F3	F2	F1	F0	ALUOp1	ALUOp0	AC0 =
X	X	X	X	X	1	1	X	ALUOp1 . F0 +
X	X	1	X	X	X	1	X	ALUOp1 . F3 +
X	X	X	X	X	X	1	1	ALUOp1 . ALUOp0

$$AC2 = ALUOp1 . F1 + ALUOp0$$

$$AC1 = ALUOp1 \ + ALUOp0 + F2 \$$

$$AC0 = ALUOp1 ( ALUOp0 + F0 + F3 )$$

## Desenho da unidade de controlo da ALU



$$AC2 = ALUOp1 \cdot F1 + ALUOp0$$

$$AC1 = ALUOp1 \setminus + ALUOp0 + F2 \setminus$$

$$AC0 = ALUOp1 ( ALUOp0 + F0 + F3 )$$

## Desenho da unidade de controlo principal

A **unidade de controlo principal** do nosso CPU simplificado apoia-se na observação de um conjunto de factos que decorrem da forma como são codificadas as instruções MIPS:

O campo **op** (*Operation code*) está situado nos bits **24-26** de **todas** as instruções

Os índices dos 2 registos que devem ser lidos (na instrução em que tal se aplica), surgem sempre nos bits **25-21** (**rs**) e **20-16** (**rt**).

Nas instruções **load/store**, o **registro base de endereçamento** está sempre nos bits **25-21** (**rs**)

As **constantes ou offsets** surgem sempre nos bits **15-0** da instrução (exceto do **lw** em que surge nos bits **25-0**)

O **registro destino** (quando se aplique) pode aparecer em um de dois campos: nos bits **20-16** (**lw, addi, slti**), ou nos bits **15-11** (instruções **aritméticas e lógicas de tipo R**)

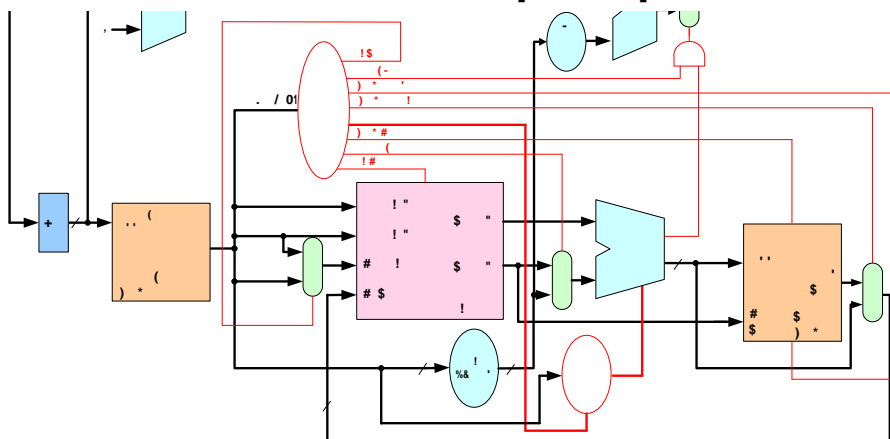


## Desenho da unidade de controlo principal

Teremos assim de especificar um total de oito sinais de controlo, para a unidade de controlo principal do ALUOp que já antes definimos. São eles:

Sinal	Efeito quando não activo	Efeito quando activo
<b>MemRead</b>	Nenhum	O conteúdo da memória de dados no endereço apresentado é apresentado à saída
<b>MemWrite</b>	Nenhum	O conteúdo do registo de memória de dados cujo endereço é fornecido é substituído pelo valor apresentado à entrada
<b>ALUSrc</b>	O segundo operando da ALU provém da segunda saída da File Register	O segundo operando da ALU provém dos 16 bits menos significativos da instr. após expansão de bits
<b>RegDst</b>	O endereço do registo destino provém do campo <b>rt</b>	O endereço do registo destino provém do campo <b>rd</b>
<b>RegWrite</b>	Nenhum	O registo indicado no endereço de escrita é alterado pelo valor presente na entrada de dados
<b>MemtoReg</b>	O valor apresentado para escrita no registo destino provém da ALU	O valor apresentado na entrada de dados dos registo internos provém da memória externa
<b>PCSrc</b>	O PC é substituído pelo seu valor actual mais 4	O PC é substituído pelo resultado do somador que calcula o endereço target de um branch condicional
<b>Branch</b>	Nenhum	Indica que a instrução é um branch condicional

## Desenho da unidade de controlo principal



A tabela de verdade respectiva, em função do tipo de instrução, será:

Instrução	Opcode	RegDst	MemtoReg	RegWrite	MemRead	MemWrite	Branch	ALUOp1	ALUOp0
<b>R - Format</b>	<b>000000</b>	<b>1</b>	<b>0</b>	<b>1</b>	<b>0</b>	<b>0</b>	<b>0</b>	<b>1</b>	<b>0</b>
<b>lw</b>	<b>100011</b>	<b>0</b>	<b>1</b>	<b>1</b>	<b>1</b>	<b>0</b>	<b>0</b>	<b>0</b>	<b>0</b>
<b>sw</b>	<b>101011</b>	<b>X</b>	<b>1</b>	<b>0</b>	<b>0</b>	<b>1</b>	<b>0</b>	<b>0</b>	<b>0</b>
<b>addi</b>	<b>001000</b>	<b>0</b>	<b>1</b>	<b>1</b>	<b>0</b>	<b>0</b>	<b>0</b>	<b>0</b>	<b>0</b>
<b>beq</b>	<b>000100</b>	<b>X</b>	<b>0</b>	<b>0</b>	<b>0</b>	<b>0</b>	<b>1</b>	<b>0</b>	<b>1</b>
<b>slti</b>	<b>001010</b>	<b>0</b>	<b>1</b>	<b>1</b>	<b>0</b>	<b>0</b>	<b>0</b>	<b>1</b>	<b>1</b>

Note-se que, tal como já aconteceu com a unidade de controlo da ALU, também a unidade de controlo principal é meramente combinatória.

## Análise do funcionamento do datapath. Exemplos.

Embora a execução de qualquer uma das instruções ocorra no intervalo de tempo correspondente a um único ciclo de relógio, poderemos, para simplificar a análise, admitir que a utilização dos elementos operativos é sequencial e decorre ao longo do seguinte código:

1. **Fetch** de uma instrução e cálculo do endereço da próxima

2. **Leitura de dois registos** *File Register*

3. **A ALU opera sobre dois valores** (a fonte dos valores a operar depende do tipo de instrução que estiver a ser executada)

4. **O resultado da operação efectuada na ALU:**

4.1. **escrito no Registo (R-Type, *addi* e *slti*)**

4.2. **usado como endereço para escrever na memória (*sw*)**

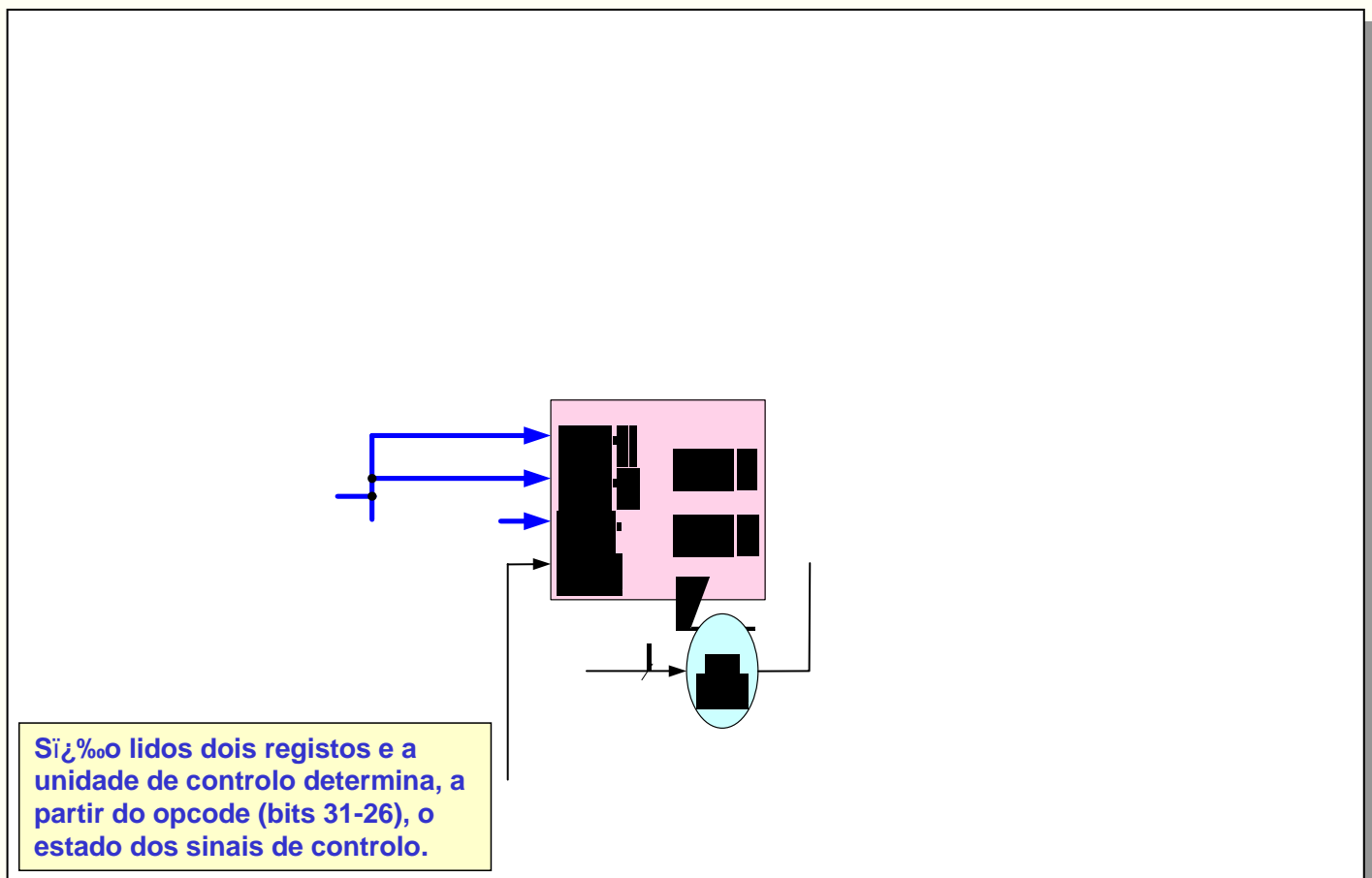
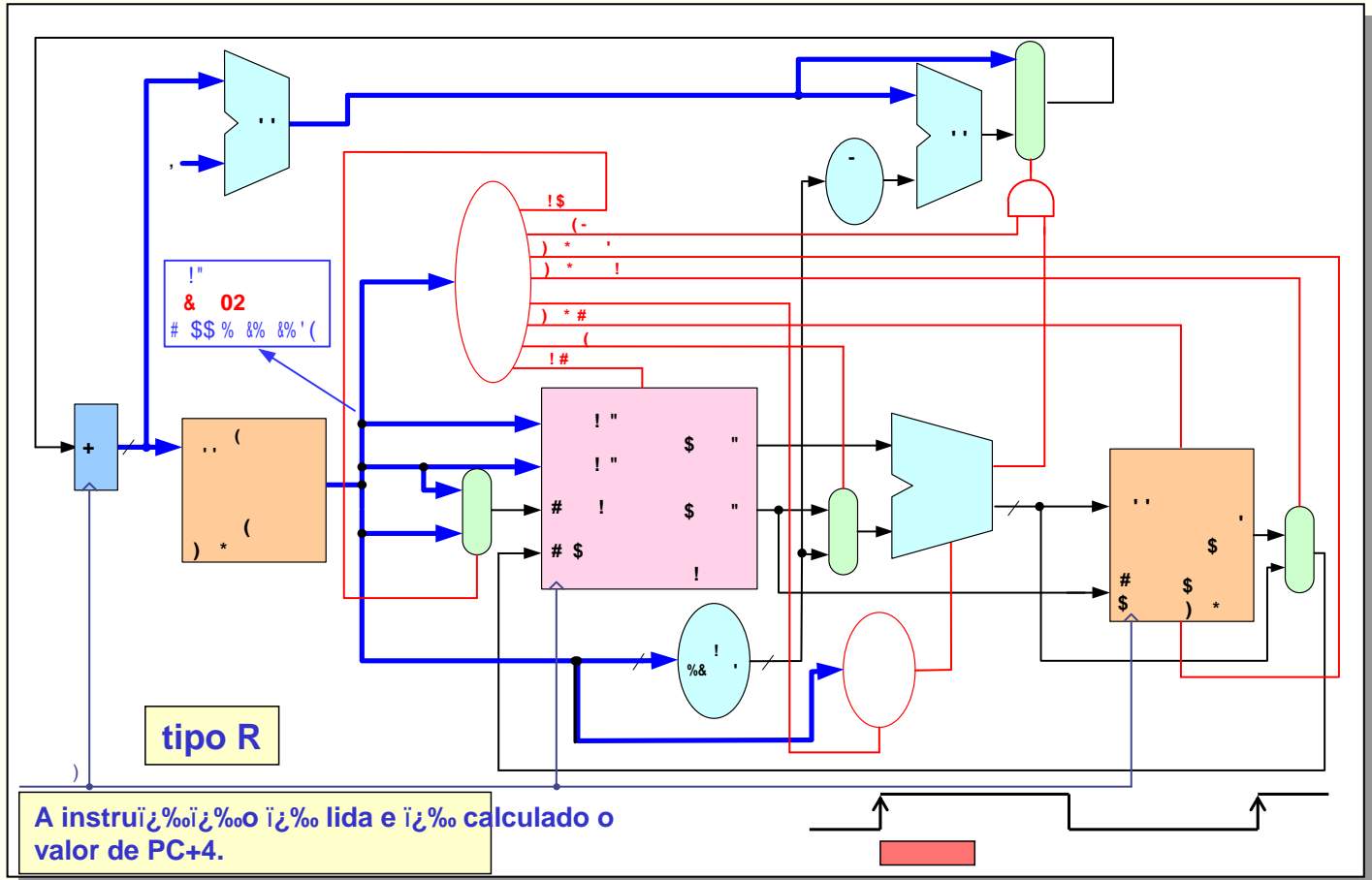
4.3. **usado como endereço para efectuar uma leitura de dados (*lw*)** - o valor lido da memória de dados é depois escrito no Registo

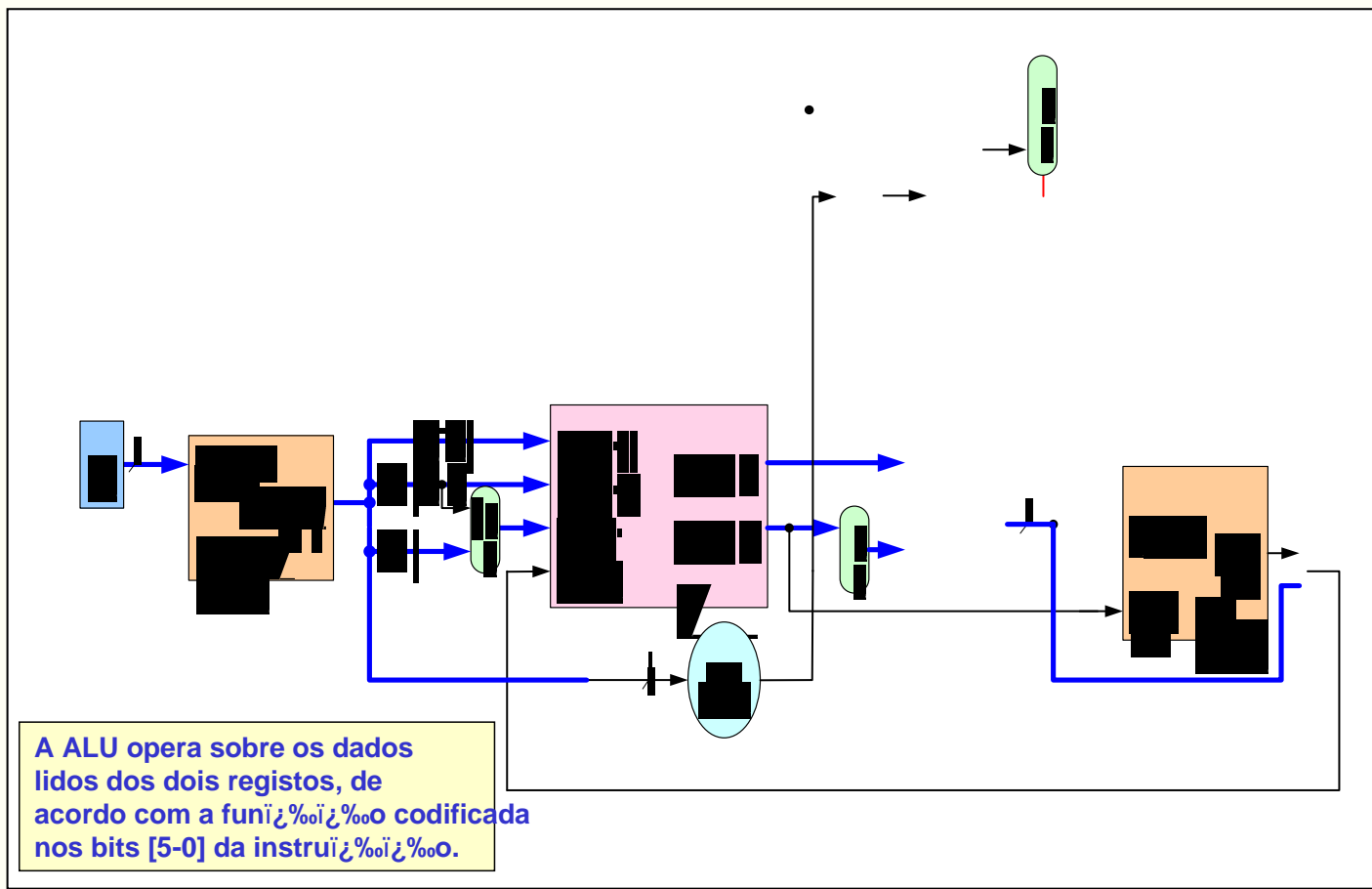
4.4. **usado para decidir qual o próximo valor do PC (*beq*, *jne*)**

## Exemplo 1

## Funcionamento do datapath nas instruções do tipo R

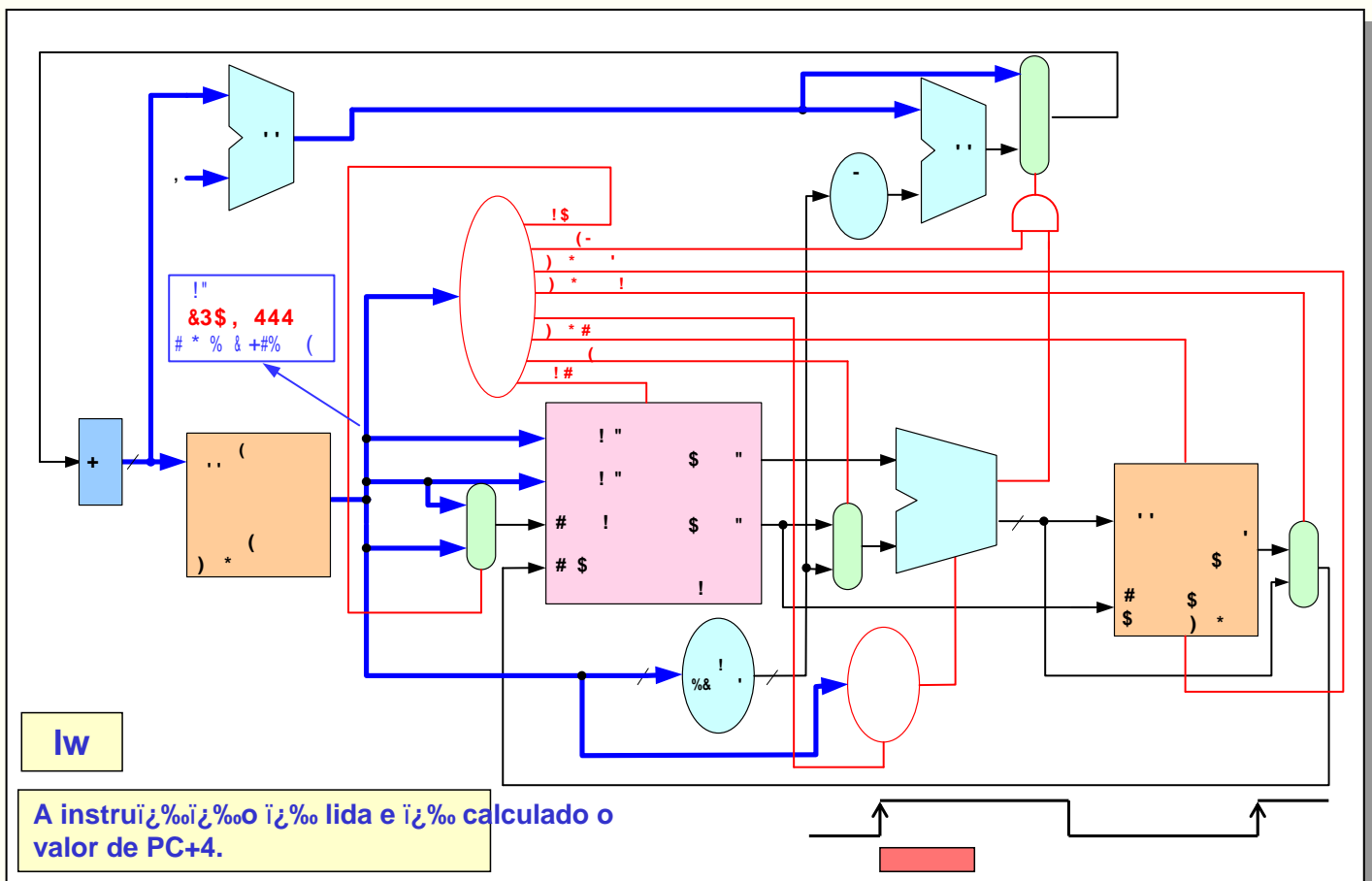


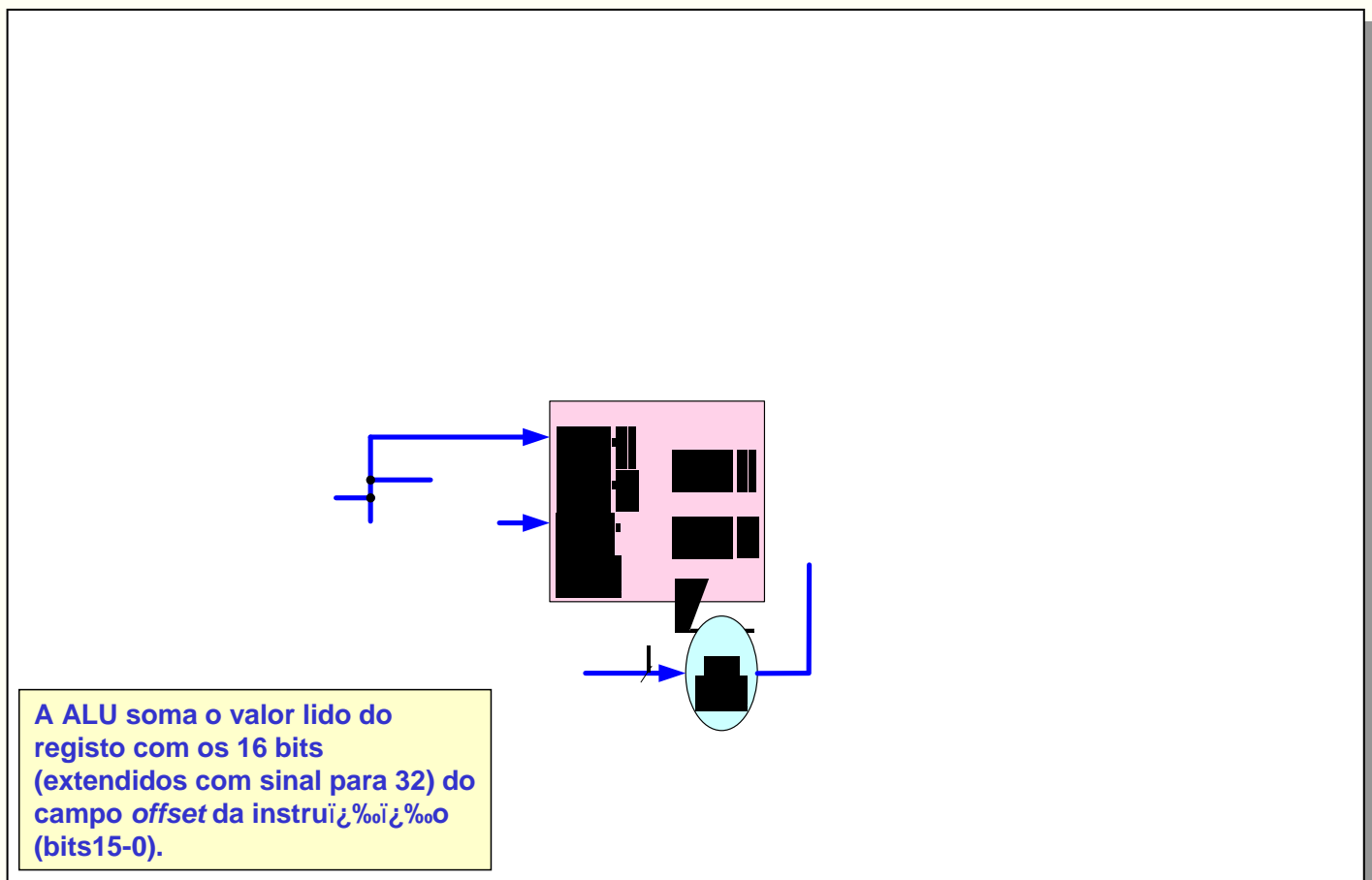
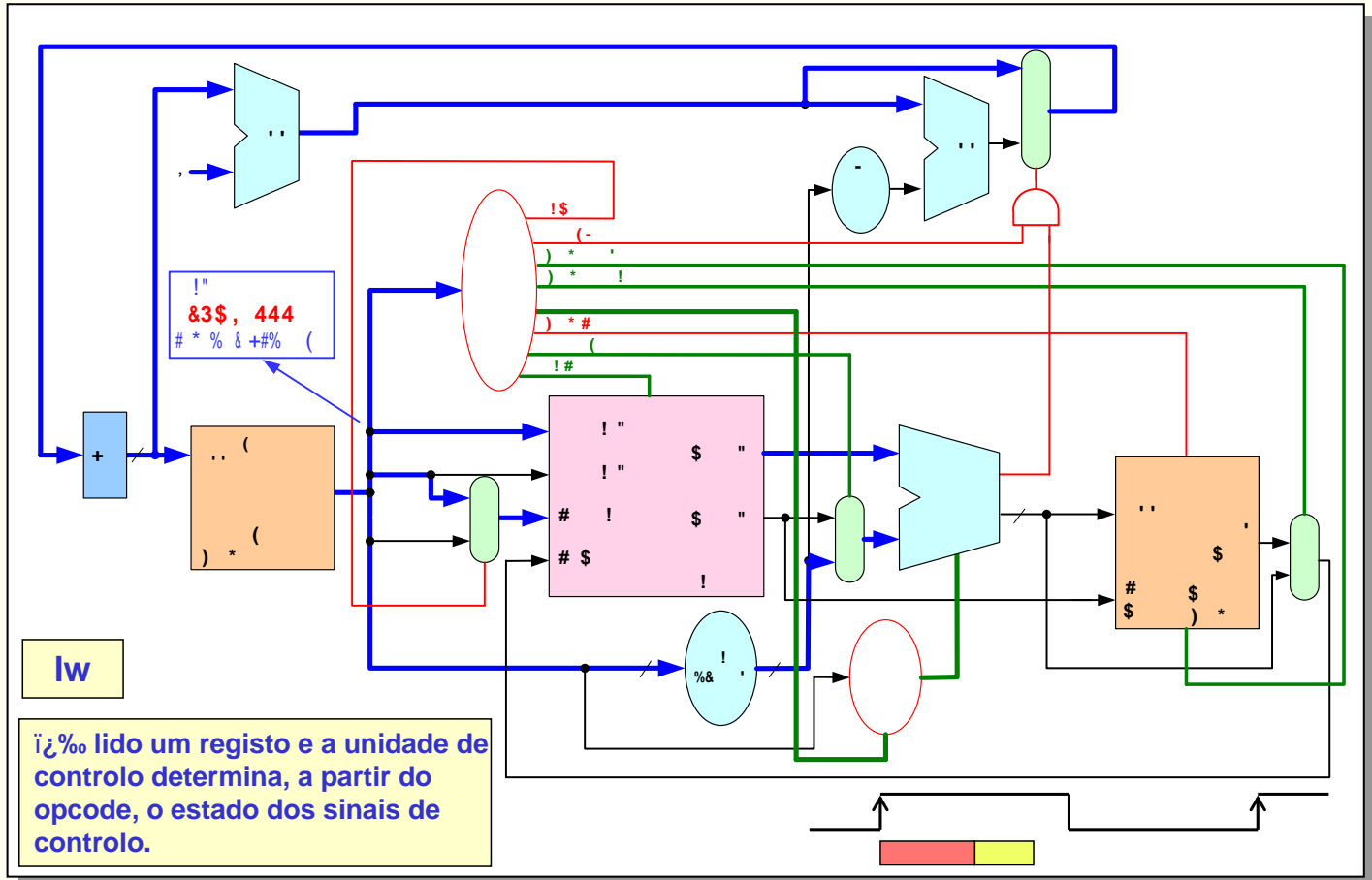


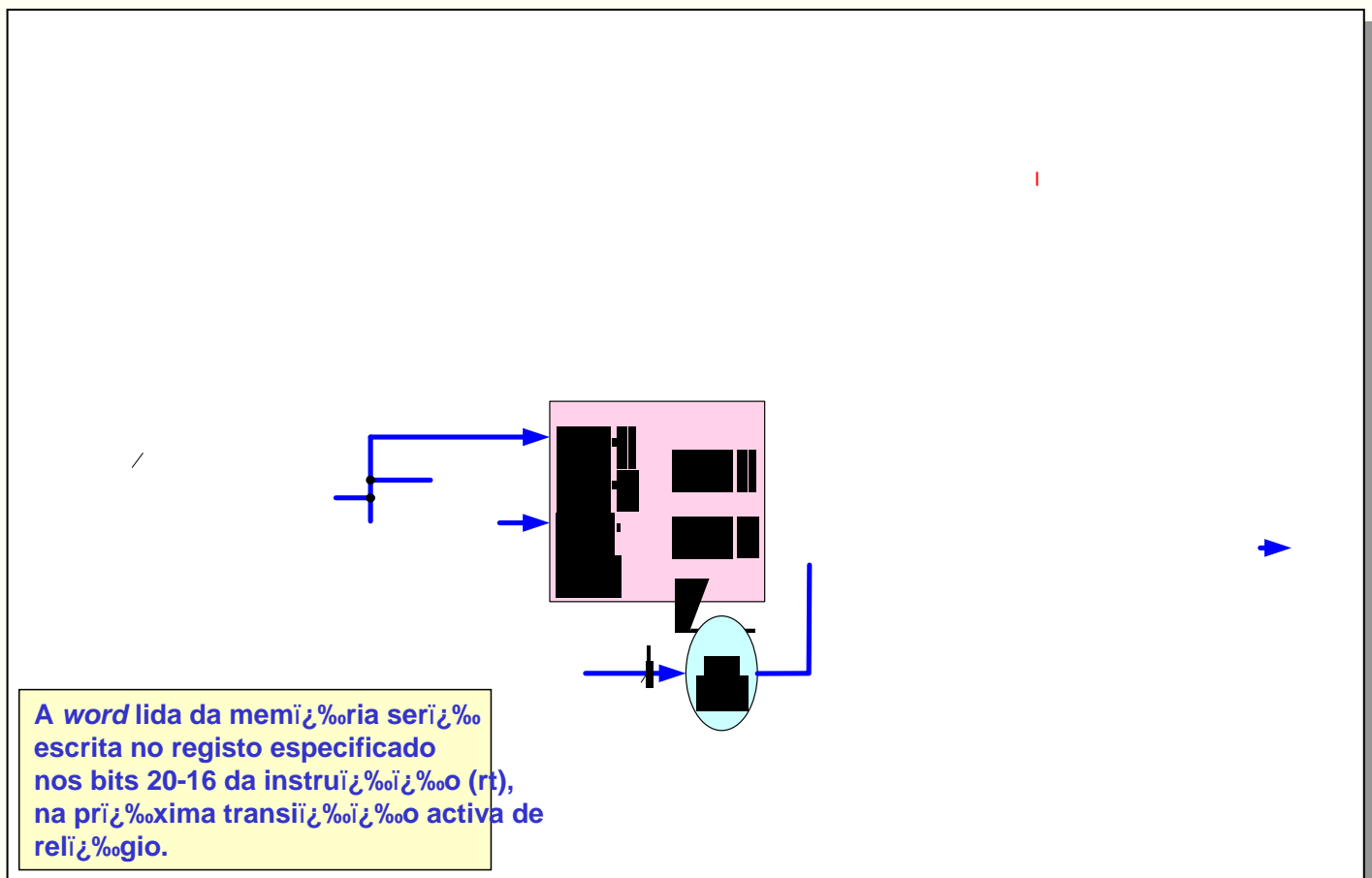
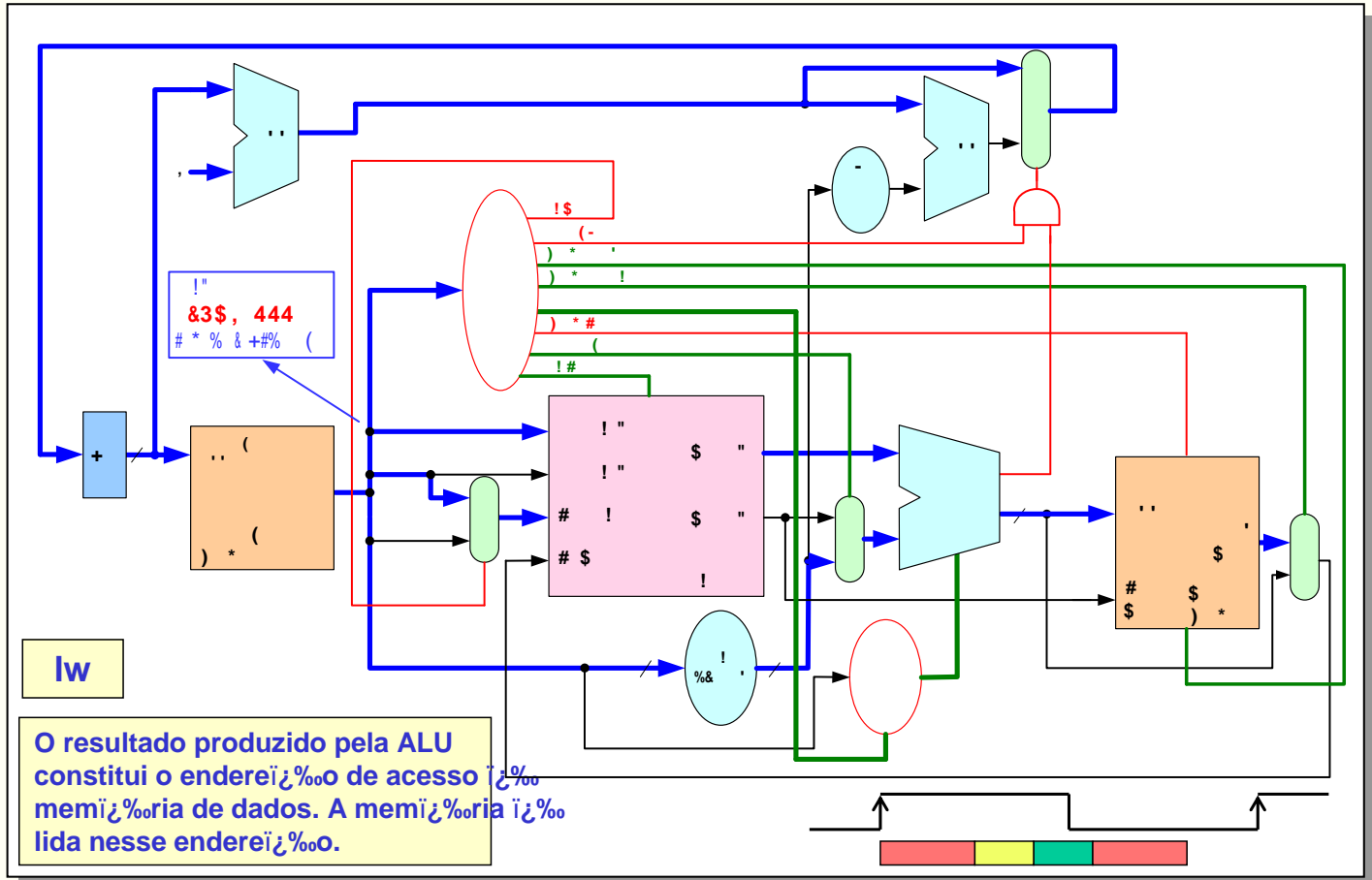


## Exemplo 2

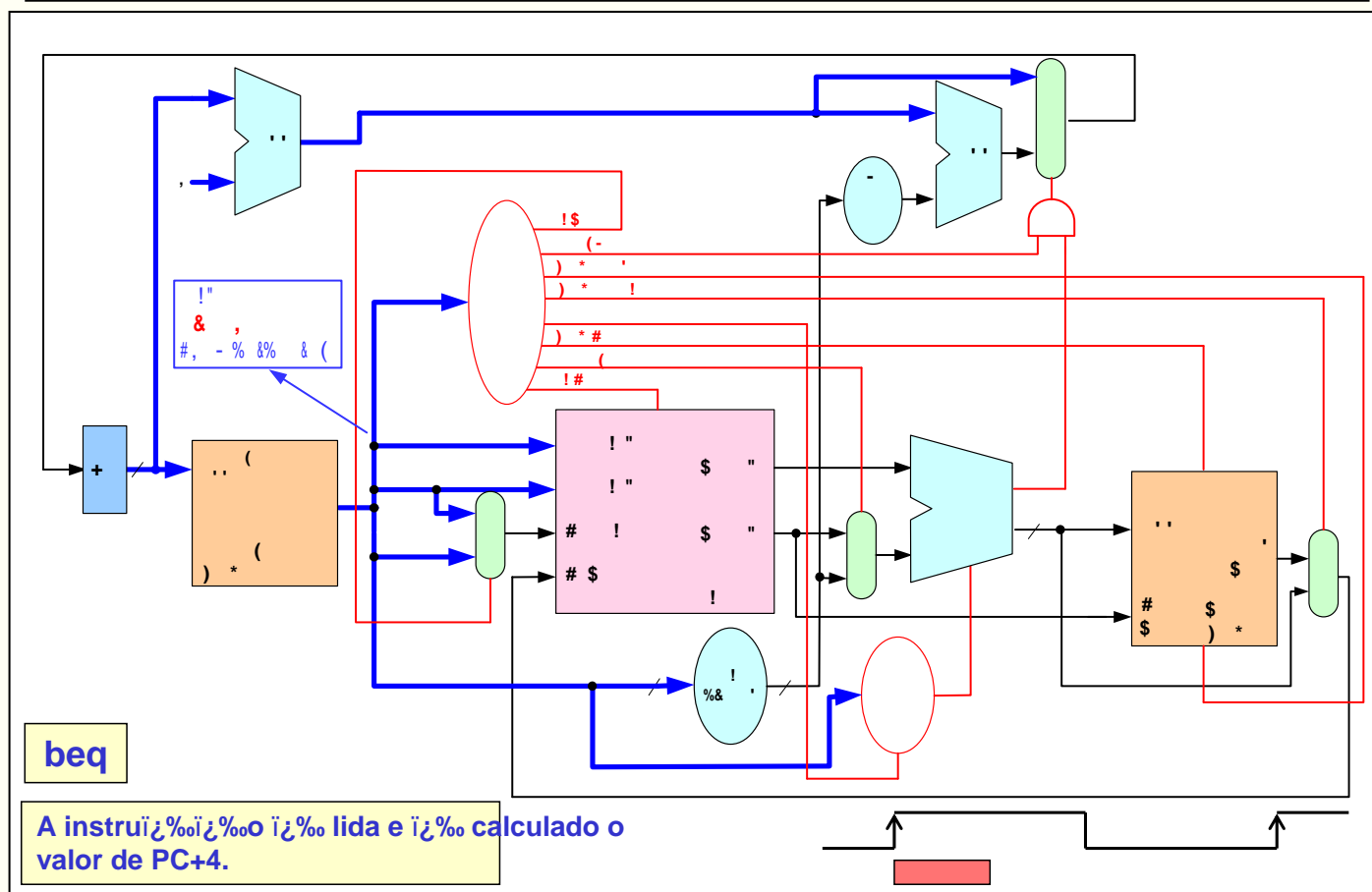
### Funcionamento do datapath na instrução *load word ( "lw" )*

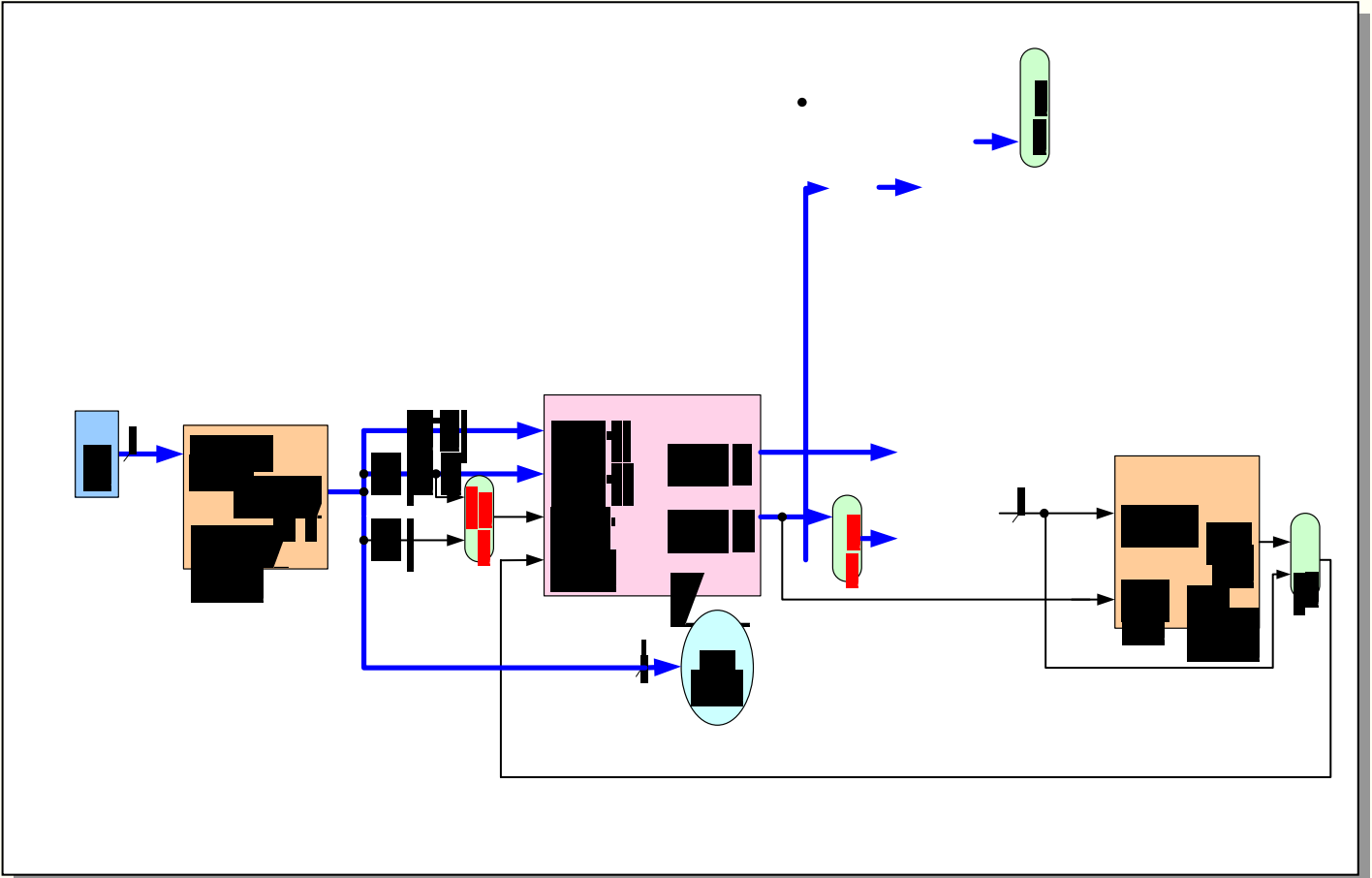






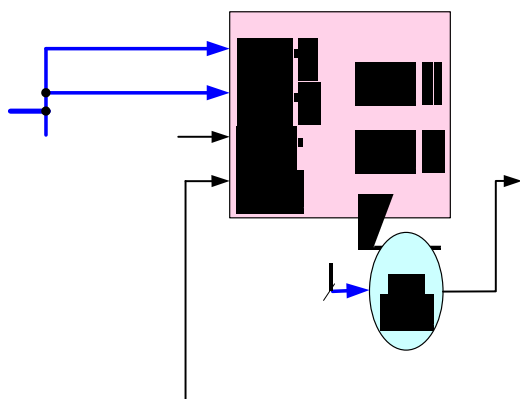
## Funcionamento do datapath na instrução *branch if equal* ( "beq" )











## Execução de uma instrução MIPS - exemplo

Endereço	Dados
(i)	(i)
0x10010030	0x63F78395
0x10010034	0xA0FCF3F0
0x10010038	0x147FAF83
(i)	(i)
\$PC	0x00400024
\$3	0x7F421231
\$4	0x15A73C49
\$5	0x10010010



Endereço	Código
(i)	(i)
0x00400020	0x00E82820
0x00400024	0x8CA30024
0x00400028	0x00681824
(i)	(i)
\$PC	0x00400028
\$3	0xA0FCF3F0
\$4	0x15A73C49
\$5	0x10010010

Vai iniciar-se o *instruction fetch* da instrução apontada pelo registo **\$PC** (0x00400024). Nesse instante o conteúdo dos registos do CPU e o endereço de dados a ser lido. Qual o conteúdo dos registos após a execução da instrução?

0x8CA30024 | lw \$3, 0x24(\$5)

Endereço\_mem: 0x10010010 + 0x24 = 0x10010034

1000110010100011000000000100100

\$3 = [0x10010034] = 0xA0FCF3F0

## Execução de uma instrução MIPS - diagrama temporal

