Aulas 18 & 19

- Unidade Aritmética e Lógica (ALU)
  - Desenho da unidade de controlo
- A unidade de controlo principal do *datapath*
- Exemplos de funcionamento do *datapath* com unidade de controlo
- Suporte para a instrução "Jump" ("j")

Bernardo Cunha, José Luís Azevedo, Arnaldo Oliveira, Tomás Oliveira e Silva

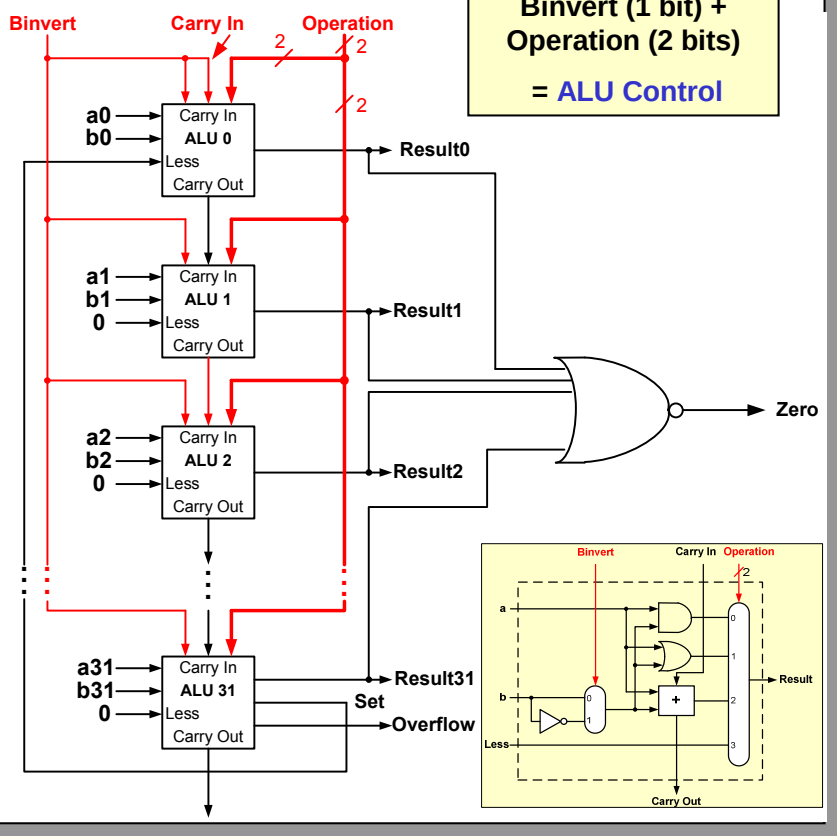
Desenho da unidade de controlo da ALU

Os sinais de controlo directo da ALU ficam reduzidos a três, uma vez que os sinais ***Binvert*** e ***Carry In*** podem ser combinados num só

(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"

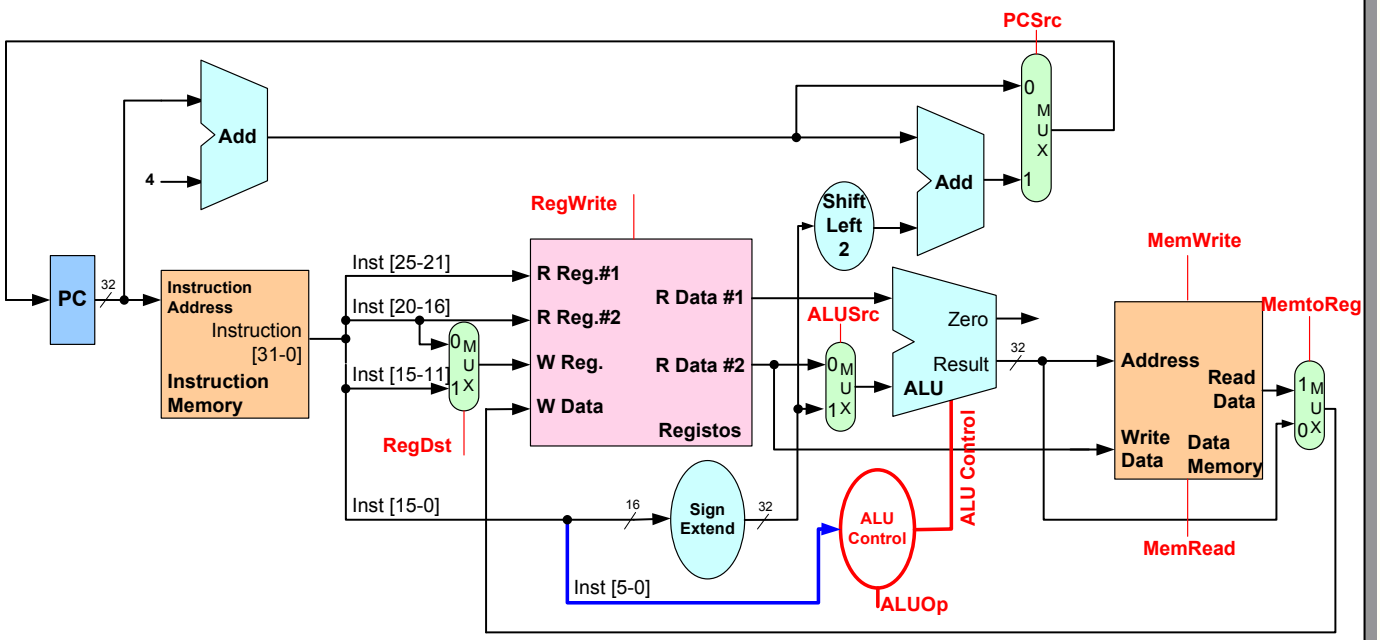


Desenho da unidade de controlo da ALU

- As instruções básicas que fazem uso da ALU são:
  - *Load e store* – para calcular o endereço da memória externa
  - *Branch if equal / not equal* – para determinar se os operandos são iguais ou diferentes
  - Aritméticas e lógicas – para efectuar a respectiva operação
- A operação a realizar na ALU depende:
  - dos campos *opcode* e *funct* nas instruções aritméticas e lógicas de tipo R: ALUControl=f(opcode, funct)
  - do campo *opcode* nas restantes instruções: ALUControl=f(opcode)
- Assim, a geração dos sinais de controlo da ALU pode ser realizada em dois níveis:
  - Nível 1: **ALUOp = g (opcode)**
  - Nível 2: **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 instruções, o campo *funct*, 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 três **sinais de controlo da ALU** em função dos 6 bits do campo ***funct*** e dos dois bits do ***ALUOp***

FUNCT									
F5	F4	F3	F2	F1	F0	ALUOp1	ALUOp0	ALU Control	
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

						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

						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

								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

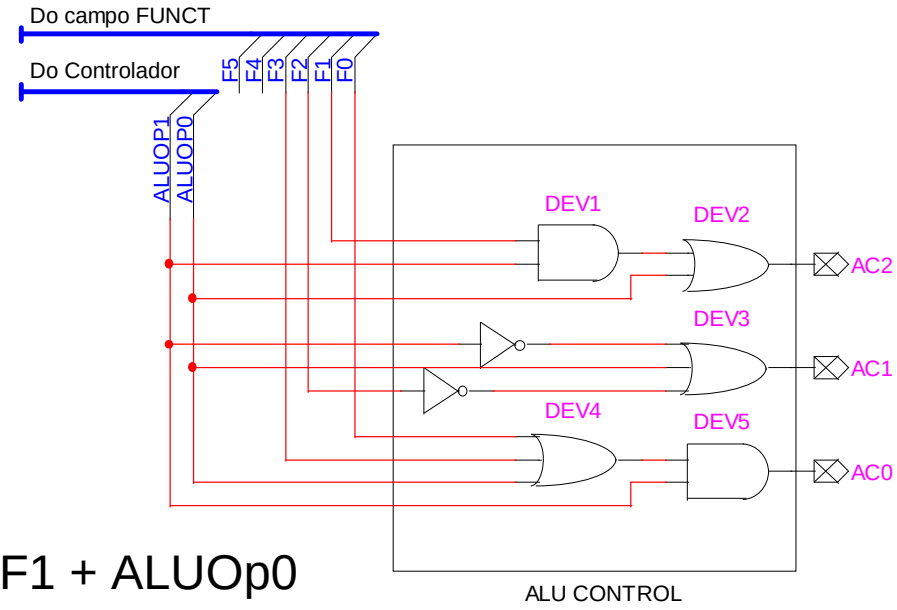
								AC0 =
X	X	X	X	X	1	1	X	ALUOp1 . F0 +
X	X	1	X	X	X	1	X	ALUOp1 . F3 +
X	X	X	X	X	X	1	1	ALUOp1 . ALUOp0

AC2 = ALUOp1 . F1 + ALUOp0

AC1 = ALUOp1\ + ALUOp0 + F2\

AC0 = ALUOp1 (ALUOp0 + F0 + F3)

Desenho da unidade de controlo da ALU



AC2 = ALUOp1 . F1 + ALUOp0

AC1 = ALUOp1\ + ALUOp0 + F2\

AC0 = ALUOp1 (ALUOp0 + F0 + F3)

Desenho da unidade de controlo principal

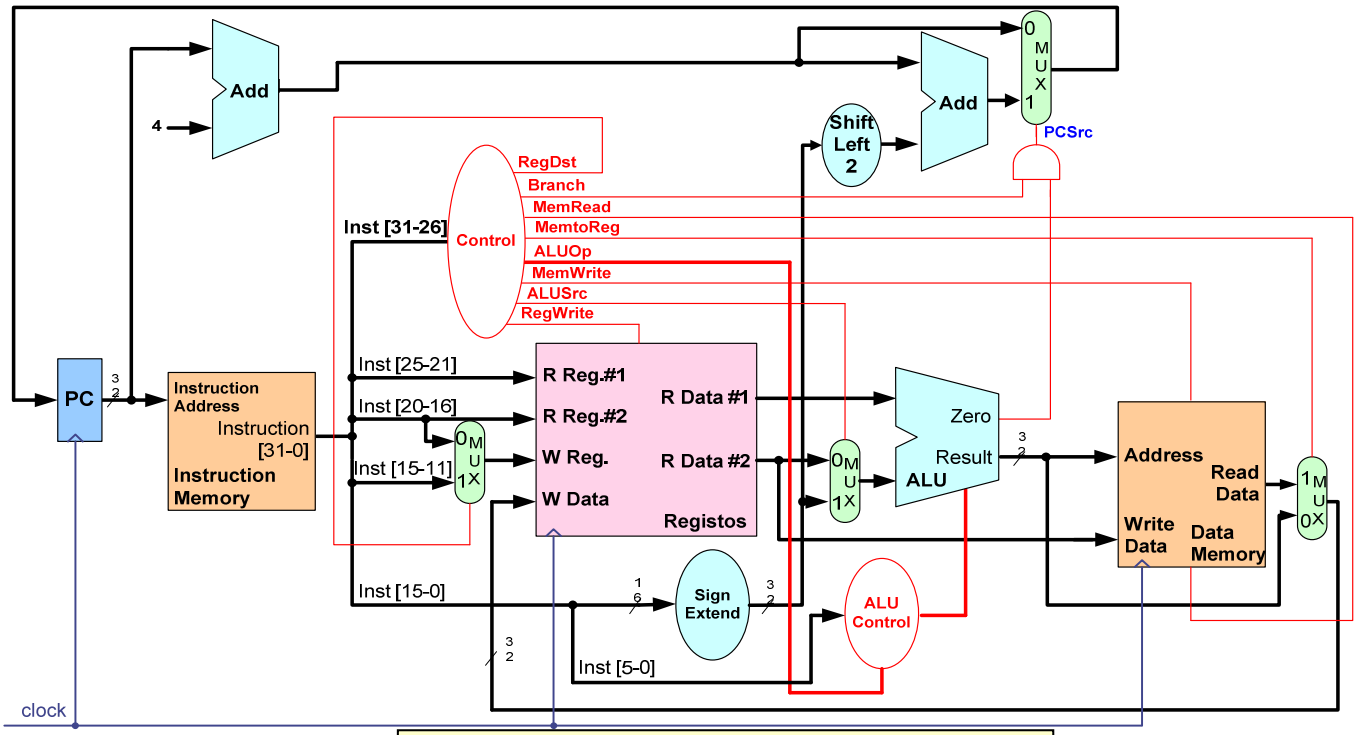
A síntese da **unidade de controlo principal** do nosso CPU simplificado apoia-se na observação de um conjunto de factos que decorrem da forma como são codificadas as instruções do MIPS:

- O campo ***op*** (*Operation code*) está situado nos bits **31-26** de **todas** as instruções
- Os índices dos 2 registos que devem ser lidos (nas instruções em que tal se aplica), surgem sempre nos bits **25-21** (**rs**) e **20-16** (**rt**).
- Nas instruções load/store, o **registo base de endereçamento** está sempre nos bits **25-21** (**rs**)
- As **constantes ou *offsets*** surgem sempre nos **bits 15-0** da instrução (à excepção do “j” em que a constante surge nos bits 25-0)
- O **registo 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

- **Alguns dos elementos de estado** presentes no *datapath* **são acedidos em todos os ciclos de relógio** (é o caso do PC e da memória de instruções). Nestes casos não há necessidade de explicitar um sinal de controlo
- Outros podem ser lidos ou escritos casuisticamente, dependendo da instrução que estiver a ser executada (**a escrita é sempre realizada de forma síncrona**). Para estes é necessário explicitar os respectivos sinais de controlo
- No *datapath*, por outro lado, existem dispositivos combinatórios que fazem o agulhamento da informação (multiplexers). Para estes é igualmente necessário definir os respectivos sinais de controlo

O aspecto do nosso *datapath*, agora com a unidade de controlo será:



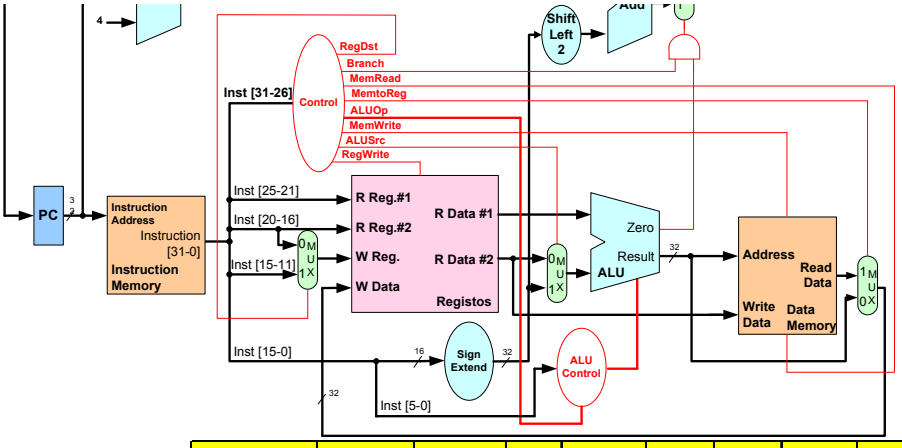
Que alterações é necessário fazer para suportar também a execução do "bne" ?

Desenho da unidade de controlo principal

Teremos assim de especificar um total de oito sinais de controlo, para além do ALUop que já antes definímos. São eles:

Sinal	Efeito quando não activo	Efeito quando activo
MemRead	Nenhum	O conteúdo da memória de dados no endereço indicado é apresentado à saída
MemWrite	Nenhum	O conteúdo do registo de memória de dados cujo endereço é fornecido é substituido pelo valor apresentado à entrada
ALUSrc	O segundo operando da ALU provém da segunda saída do *File Register*	O segundo operando da ALU provém dos 16 bits menos significativos da instr. após expansão do sinal
RegDst	O endereço do registo destino provém do campo **rt**	O endereço do registo destino provém do campo **rd**
RegWrite	Nenhum	O registo indicado no endereço de escrita é alterado pelo valor presente na entrada de dados
MemtoReg	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
PCSrc	O PC é substituido pelo seu valor actual mais 4	O PC é substituido pelo resultado do somador que calcula o endereço target do *branch* condicional
Branch	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	ALU Src	Memto Reg	Reg Write	Mem Read	Mem WRite	Branch	ALUOp1	ALUOp0
R - Format	000000	1	0	0	1	0	0	0	1	0
lw	100011	0	1	1	1	1	0	0	0	0
sw	101011	X	1	X	0	0	1	0	0	0
addi	001000	0	1	0	1	0	0	0	0	0
beq	000100	X	0	X	0	0	0	1	0	1
slti	001010	0	1	0	1	0	0	0	1	1

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

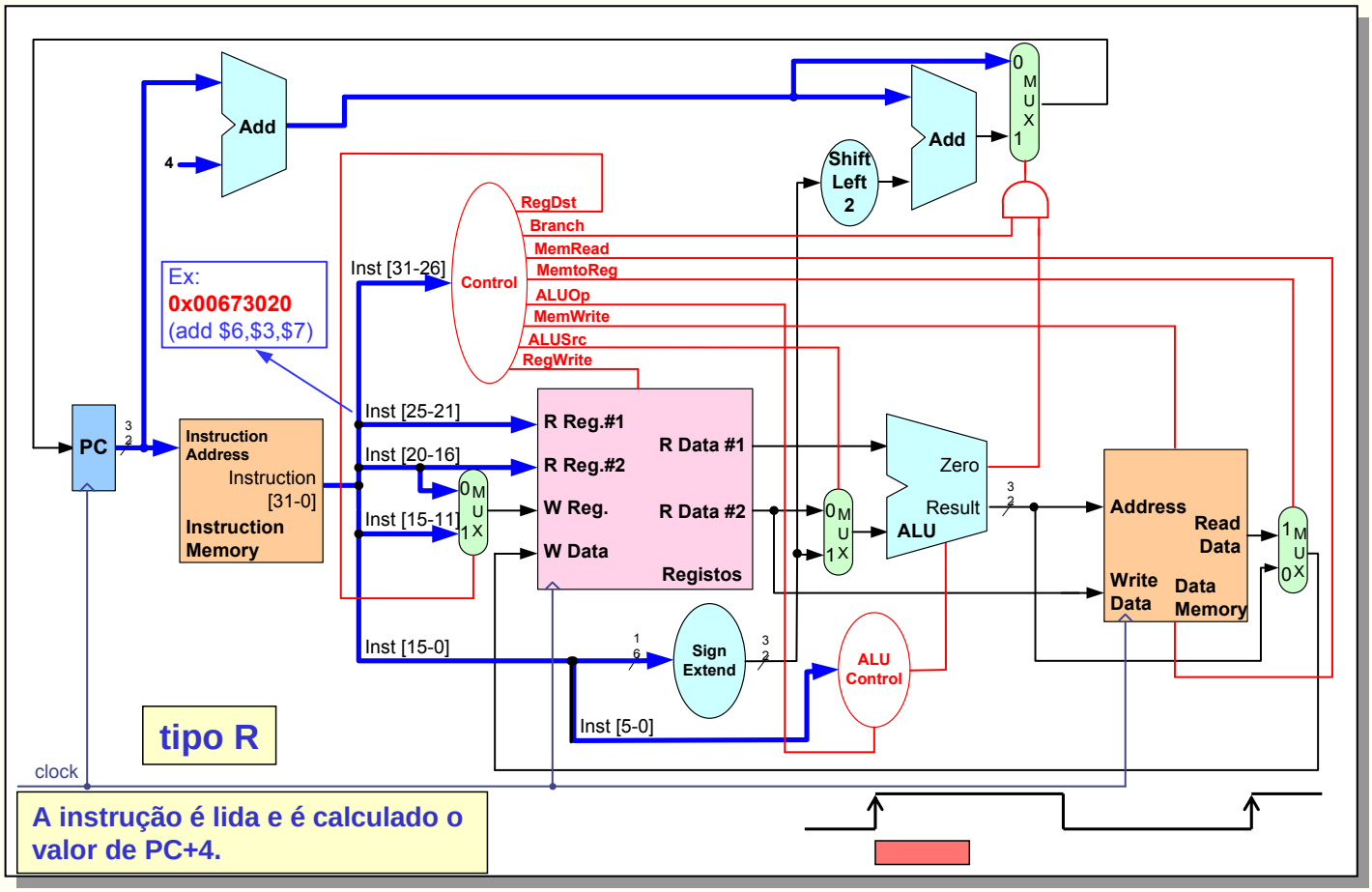
Análise do funcionamento do *datapath*. Exemplos.

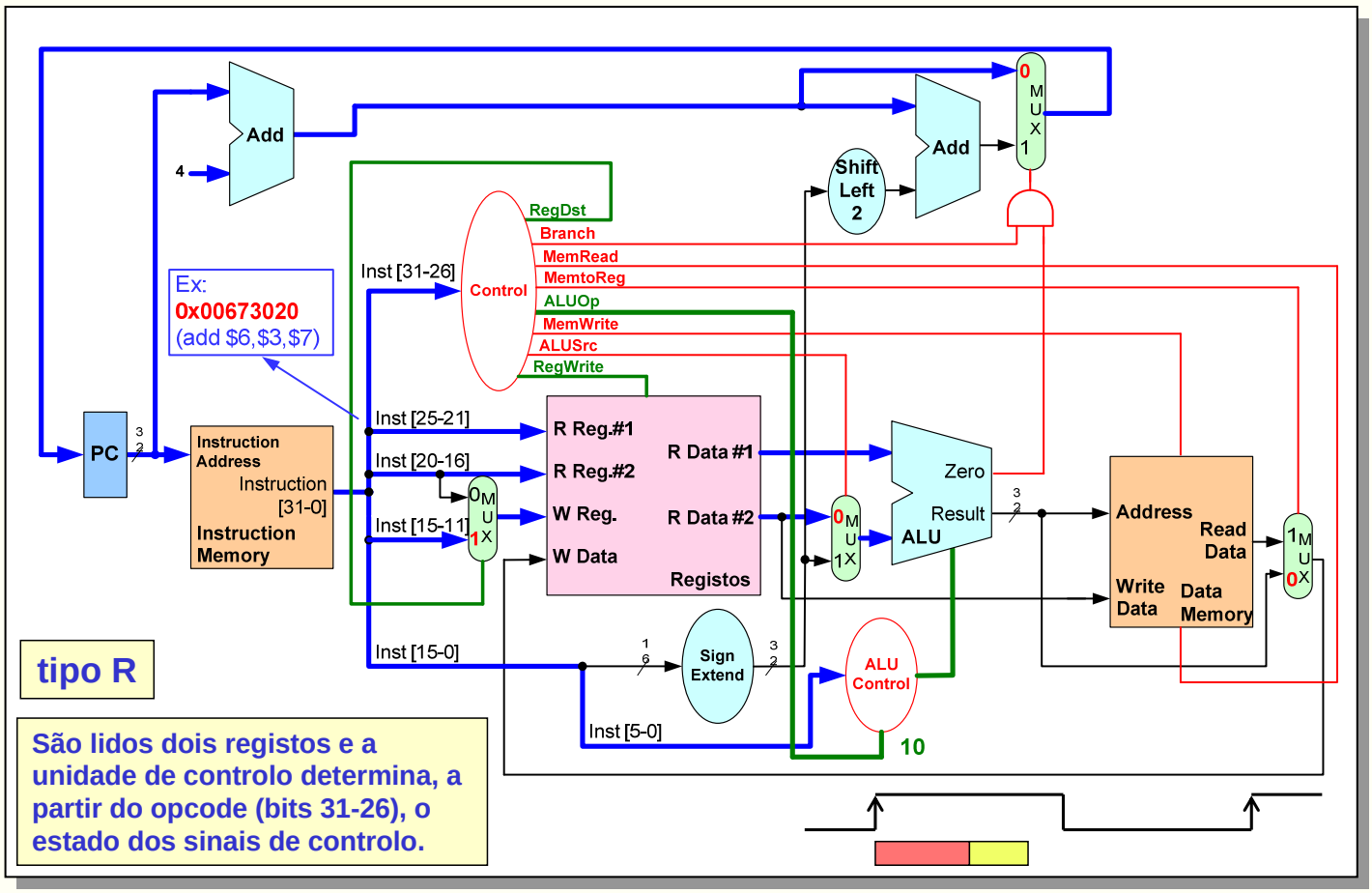
Embora a execução de qualquer uma das instruções suportadas ocorra no intervalo de tempo correspondente a um único ciclo de relógio, poderemos, para simplificar a análise, admitir que a utilização dos vários elementos operativos é "sequencial" e decorre ao longo do seguinte conjunto de operações:

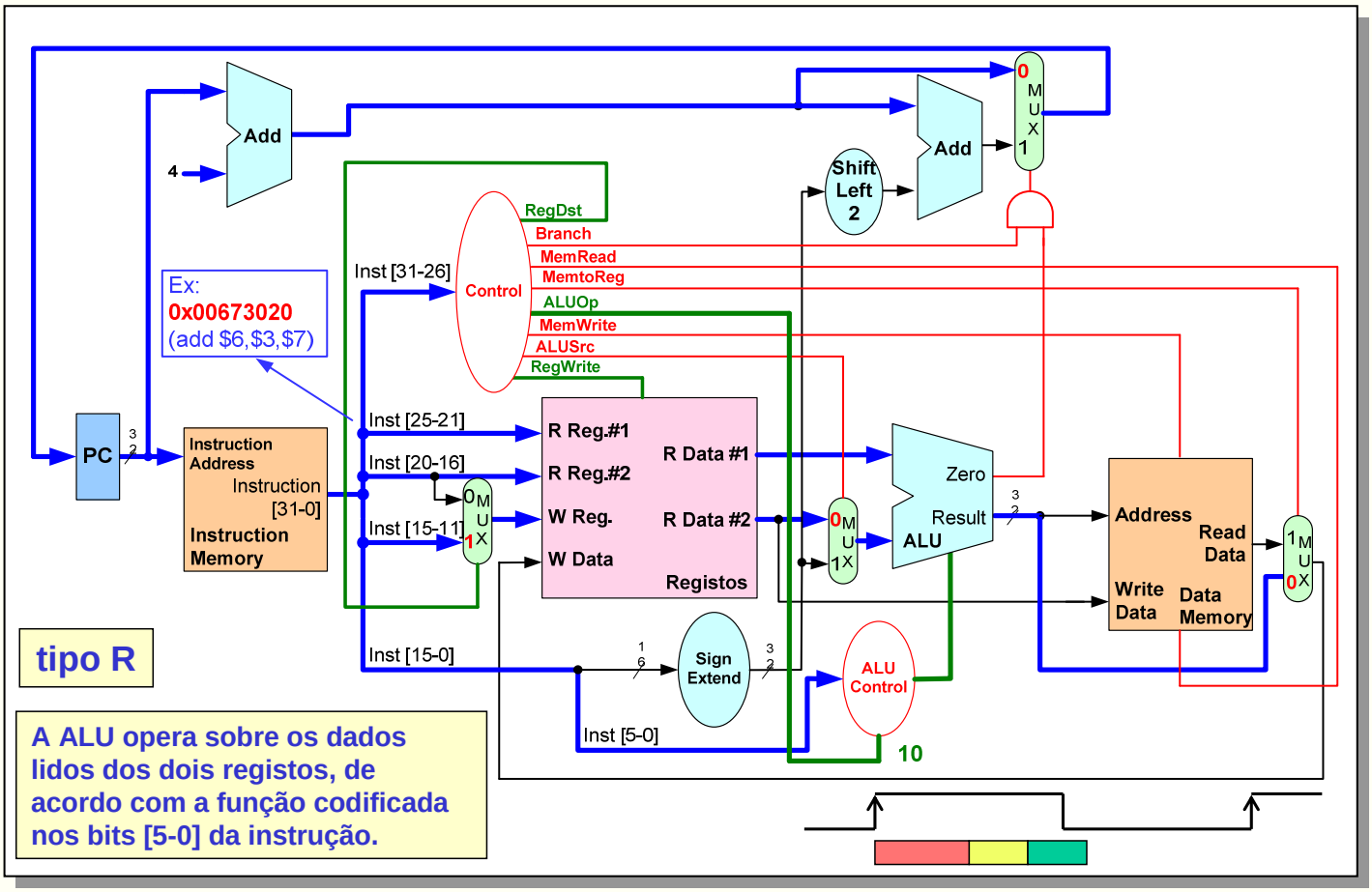
- *Fetch* de uma instrução e cálculo do endereço da próxima instrução
- Leitura de dois registos do *File Register*
- A ALU opera sobre dois valores (a fonte dos valores a operar depende do tipo de instrução que estiver a ser executada)
- O resultado da operação efectuada na ALU:
  - é escrito no *File Register* (***R-Type, "addi" e "slti"***)
  - é usado como endereço para escrever na memória de dados (***sw***)
  - é usado como endereço para efectuar uma leitura da memória de dados (***lw***) - o valor lido da memória de dados é depois escrito no *File Register*
  - é usado para decidir qual o próximo valor do PC (***beq / bne***)

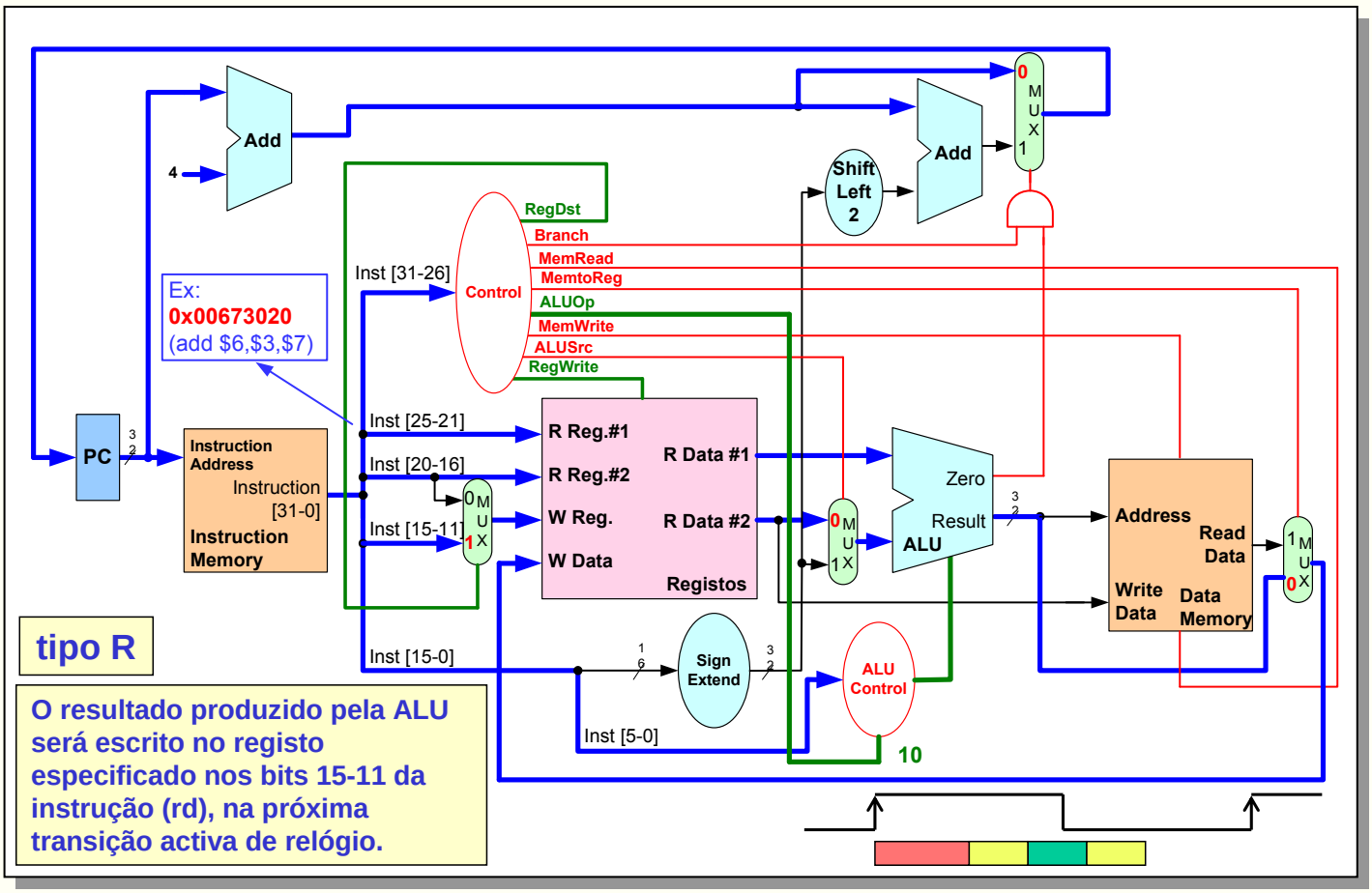
Exemplo 1

Funcionamento do datapath nas instruções do tipo R


Add
4
Add
Shift Left 2
0 M U X 1
RegDst
Branch
MemRead
MemtoReg
ALUOp
MemWrite
ALUSrc
RegWrite
Control
Inst [31-26]
Ex:
0x00673020
(add $6,$3,$7)
PC
Instruction Address
Instruction [31-0]
Instruction Memory
Inst [25-21]
Inst [20-16]
Inst [15-11]
Inst [15-0]
Inst [5-0]
R Reg.#1
R Reg.#2
W Reg.
W Data
R Data #1
R Data #2
Registos
Zero
Result
ALU
Address
Read Data
Write Data
Data Memory
Sign Extend
ALU Control
clock
tipo R
A instrução é lida e é calculado o valor de PC+4.

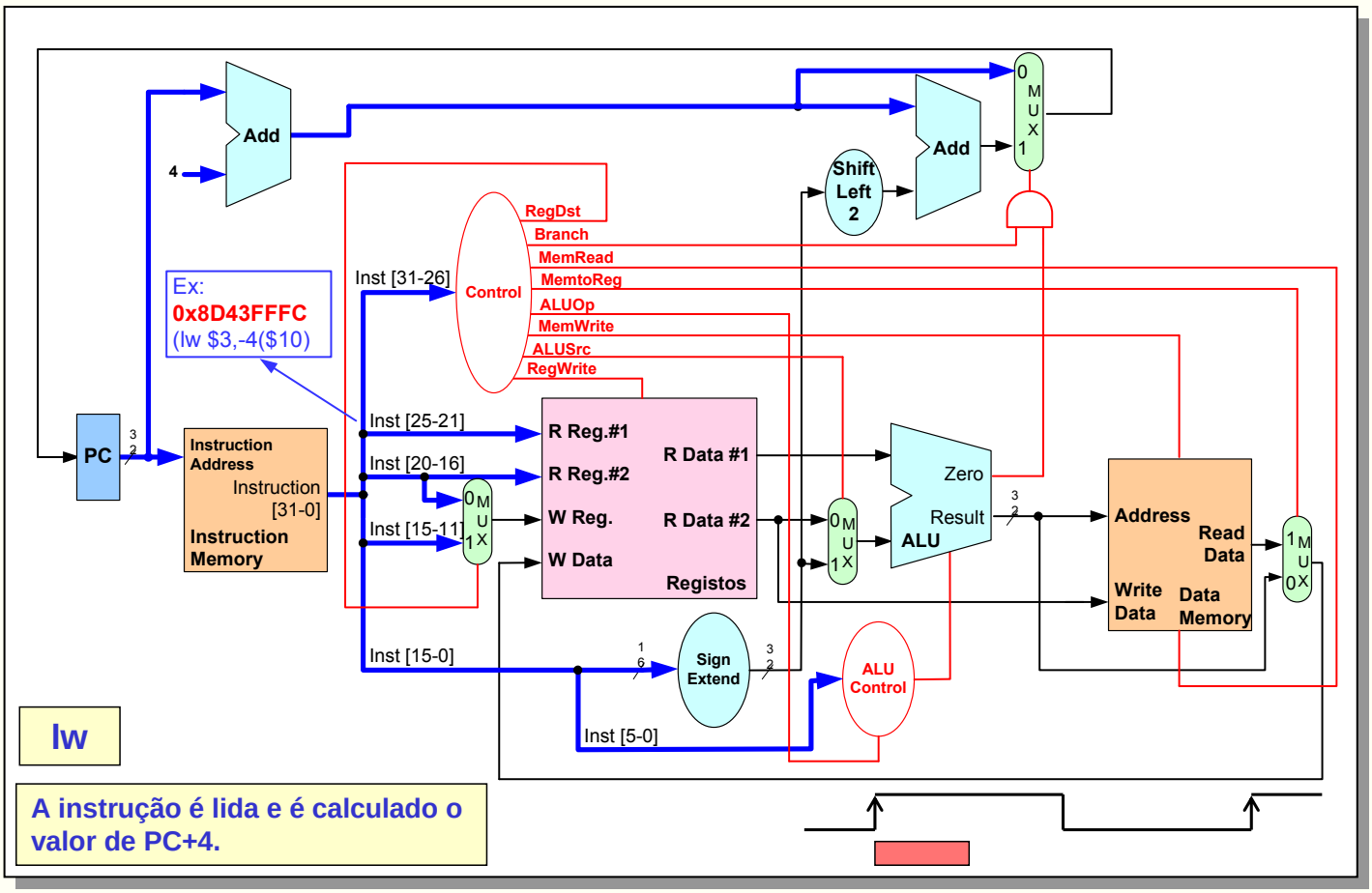

Add
4
Add
Shift Left 2
0 M U X 1
RegDst
Branch
MemRead
MemtoReg
ALUOp
MemWrite
ALUSrc
RegWrite
Control
Inst [31-26]
Ex:
0x00673020
(add $6,$3,$7)
PC
Instruction Address
Instruction [31-0]
Instruction Memory
Inst [25-21]
Inst [20-16]
Inst [15-11]
Inst [15-0]
Inst [5-0]
R Reg.#1
R Reg.#2
W Reg.
W Data
R Data #1
R Data #2
Registos
Zero
Result
ALU
Address
Read Data
Write Data
Data Memory
Sign Extend
ALU Control
10
tipo R
São lidos dois registos e a unidade de controlo determina, a partir do opcode (bits 31-26), o estado dos sinais de controlo.

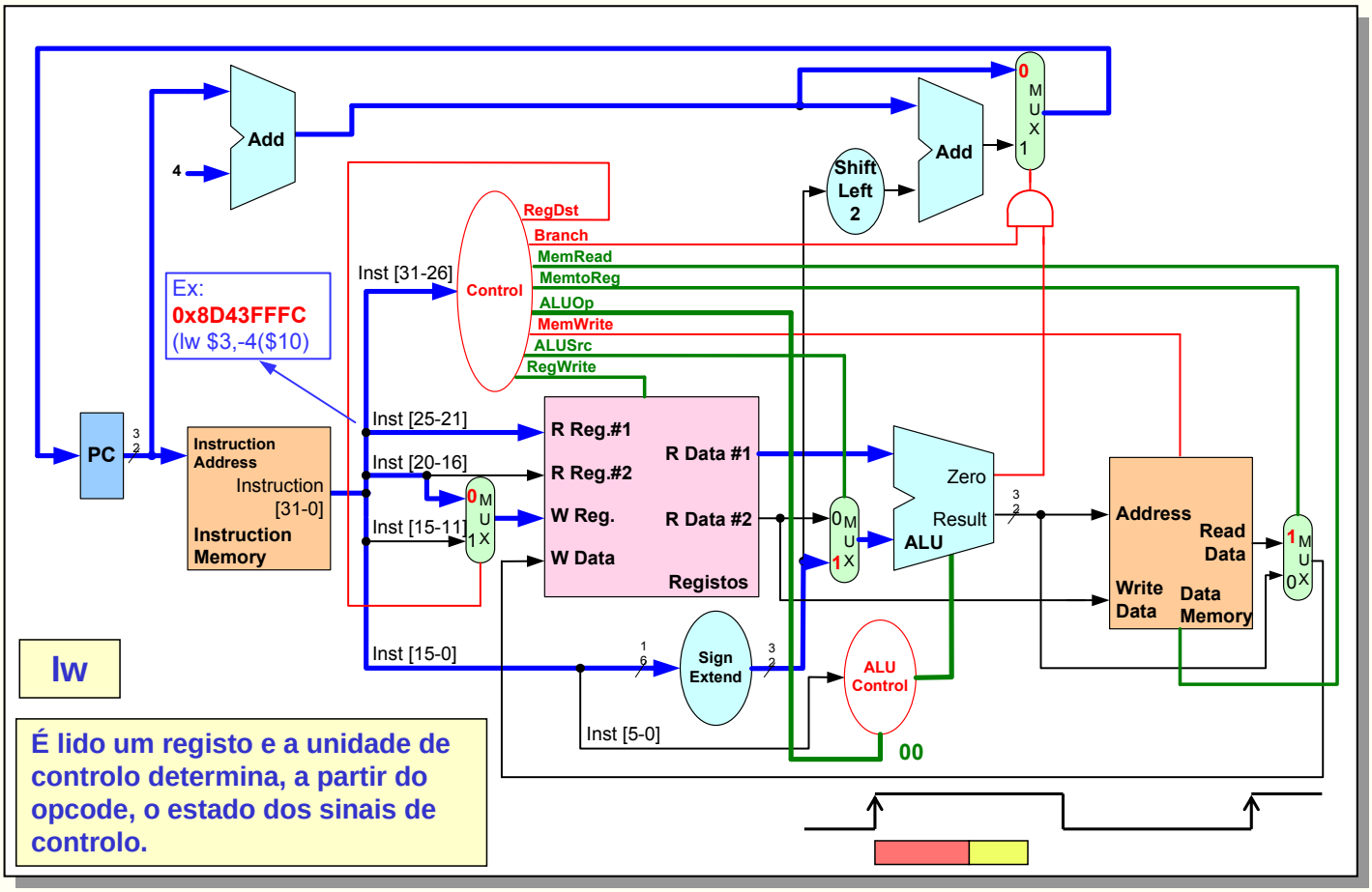

Add
4
Add
Shift Left 2
M U X
0
1
RegDst
Branch
MemRead
MemtoReg
ALUOp
MemWrite
ALUSrc
RegWrite
Control
Inst [31-26]
Ex:
0x00673020
(add $6,$3,$7)
PC
Instruction Address
Instruction [31-0]
Instruction Memory
Inst [25-21]
Inst [20-16]
Inst [15-11]
Inst [15-0]
Inst [5-0]
R Reg.#1
R Reg.#2
W Reg.
W Data
R Data #1
R Data #2
Registos
Zero
Result
ALU
Address
Read Data
Write Data
Data Memory
Sign Extend
ALU Control
10
tipo R
A ALU opera sobre os dados lidos dos dois registos, de acordo com a função codificada nos bits [5-0] da instrução.

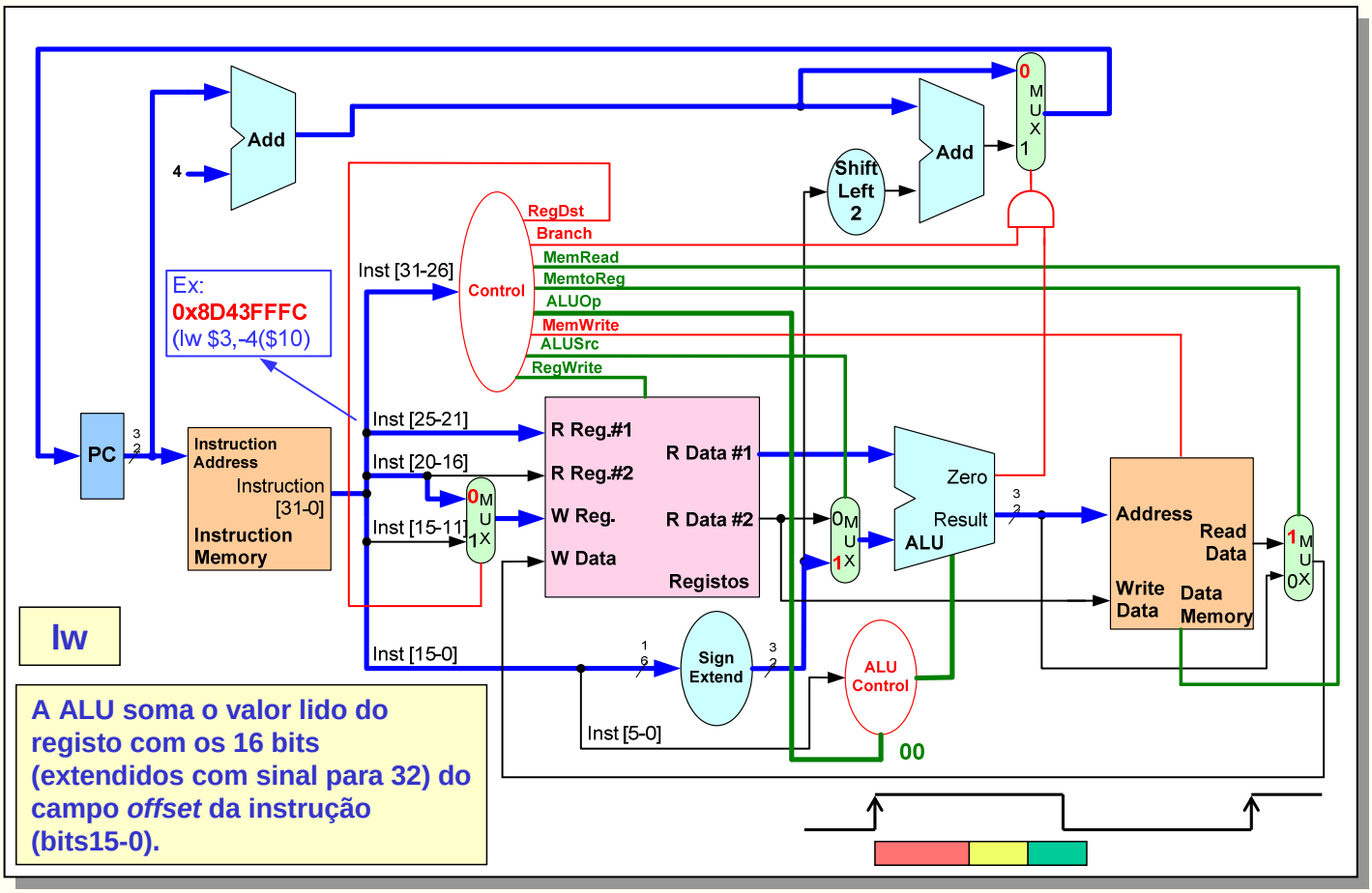

Add
4
Add
Shift Left 2
M U X
0
1
RegDst
Branch
MemRead
MemtoReg
ALUOp
MemWrite
ALUSrc
RegWrite
Control
Inst [31-26]
Ex:
0x00673020
(add $6,$3,$7)
PC
Instruction Address
Instruction [31-0]
Instruction Memory
Inst [25-21]
Inst [20-16]
Inst [15-11]
Inst [15-0]
Inst [5-0]
R Reg.#1
R Reg.#2
W Reg.
W Data
R Data #1
R Data #2
Registos
Zero
Result
ALU
Address
Read Data
Write Data
Data Memory
Sign Extend
ALU Control
10
tipo R
O resultado produzido pela ALU será escrito no registo especificado nos bits 15-11 da instrução (rd), na próxima transição activa de relógio.

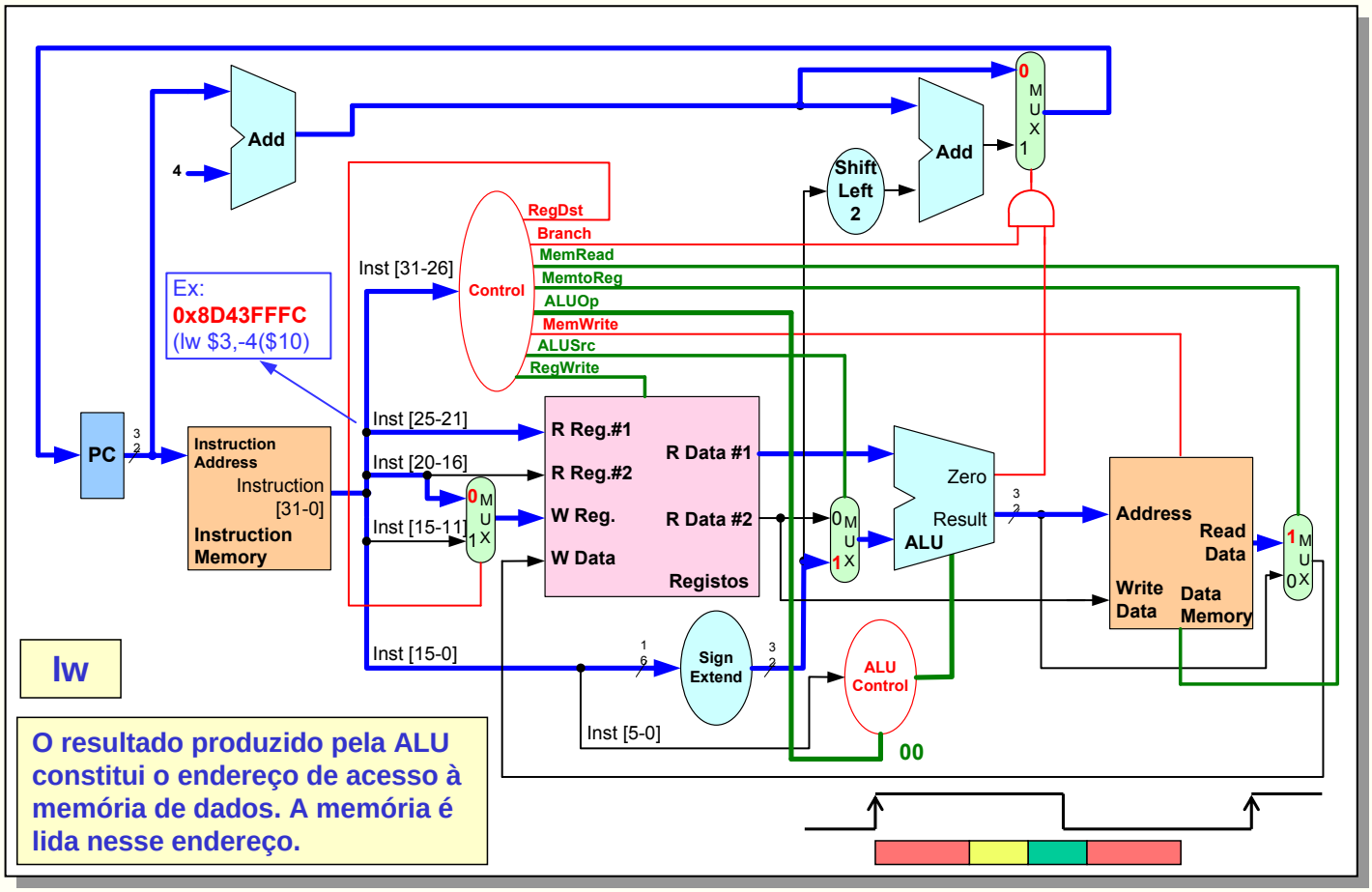
Exemplo 2

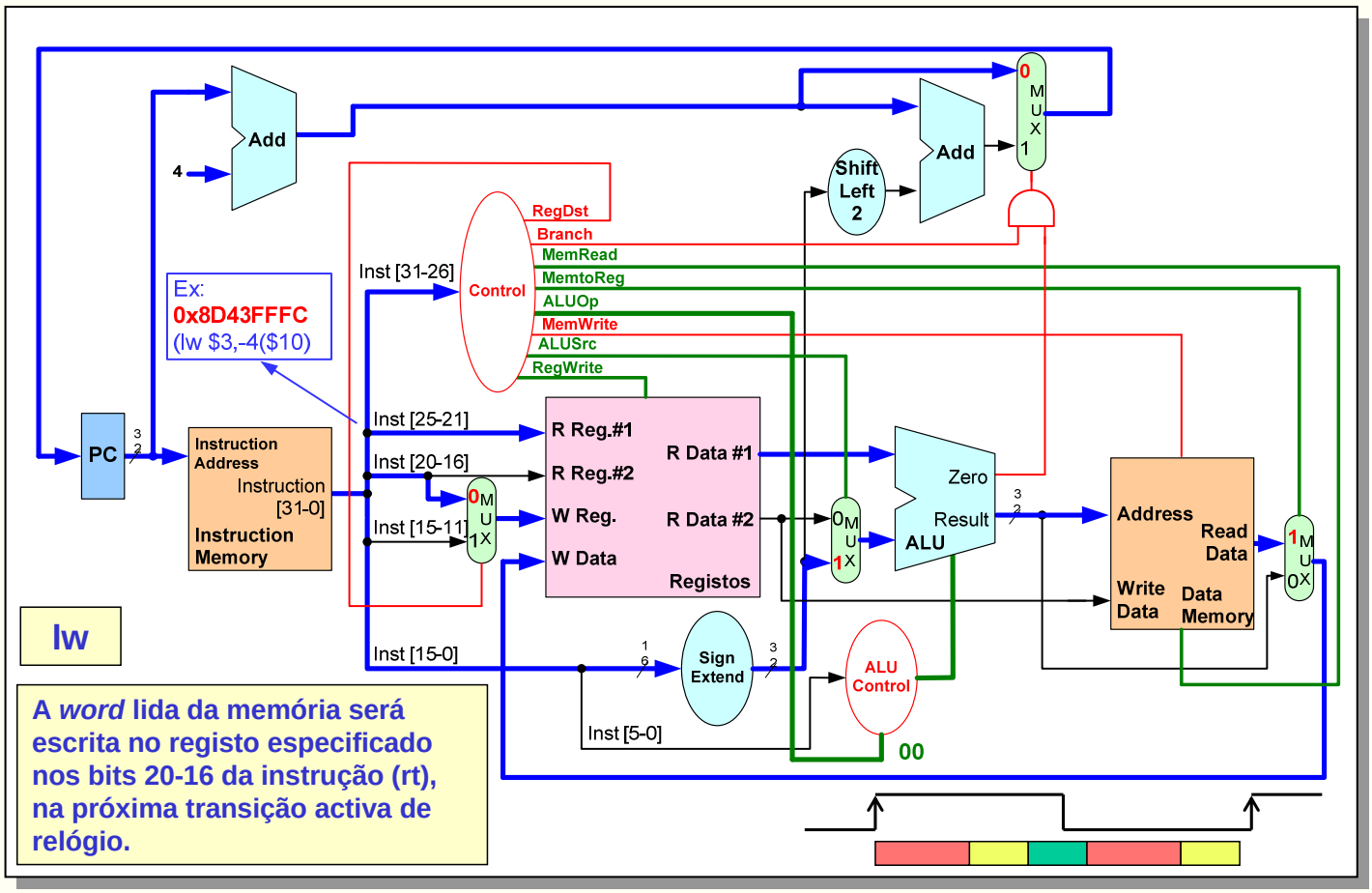
Funcionamento do datapath na instrução *load word* ("lw")




Add
4
Add
Shift Left 2
0 M U X 1
RegDst
Branch
MemRead
MemtoReg
ALUOp
MemWrite
ALUSrc
RegWrite
Control
Inst [31-26]
Ex:
0x8D43FFFC
(lw $3,-4($10)
PC
Instruction Address
Instruction [31-0]
Instruction Memory
Inst [25-21]
Inst [20-16]
Inst [15-11]
Inst [15-0]
Inst [5-0]
R Reg.#1
R Reg.#2
W Reg.
W Data
R Data #1
R Data #2
Registos
Zero
Result
ALU
Address
Read Data
Write Data
Data Memory
Sign Extend
ALU Control
00
lw
É lido um registo e a unidade de controlo determina, a partir do opcode, o estado dos sinais de controlo.

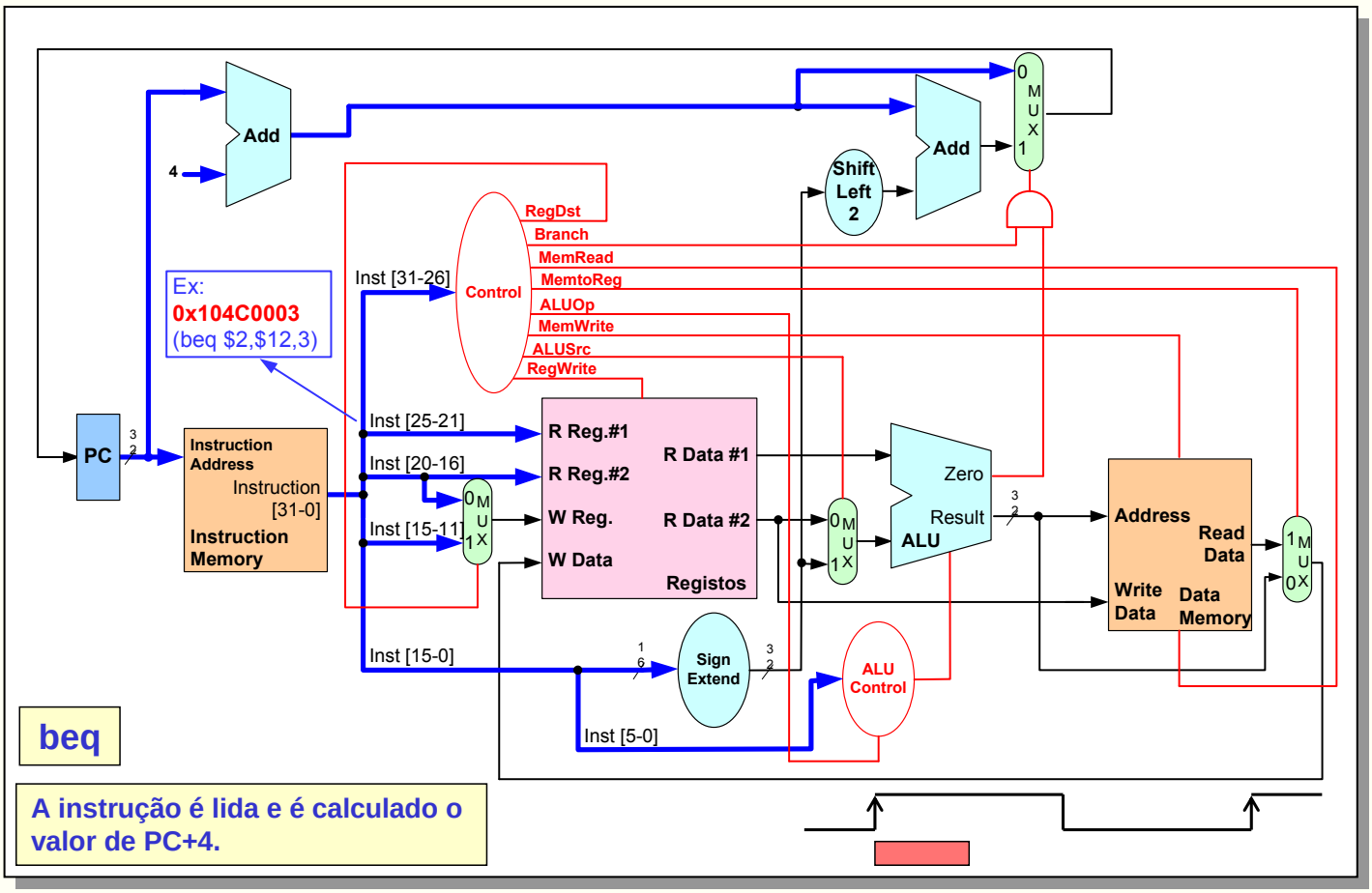

Add
4
Add
Shift Left 2
0 M U X 1
RegDst
Branch
MemRead
MemtoReg
ALUOp
MemWrite
ALUSrc
RegWrite
Control
Inst [31-26]
Ex:
0x8D43FFFC
(lw $3,-4($10)
PC
Instruction Address
Instruction [31-0]
Instruction Memory
Inst [25-21]
Inst [20-16]
Inst [15-11]
Inst [15-0]
Inst [5-0]
R Reg.#1
R Reg.#2
W Reg.
W Data
R Data #1
R Data #2
Registos
Zero
Result
ALU
Address
Read Data
Write Data
Data Memory
Sign Extend
ALU Control
00
lw
A ALU soma o valor lido do registo com os 16 bits (extendidos com sinal para 32) do campo *offset* da instrução (bits15-0).

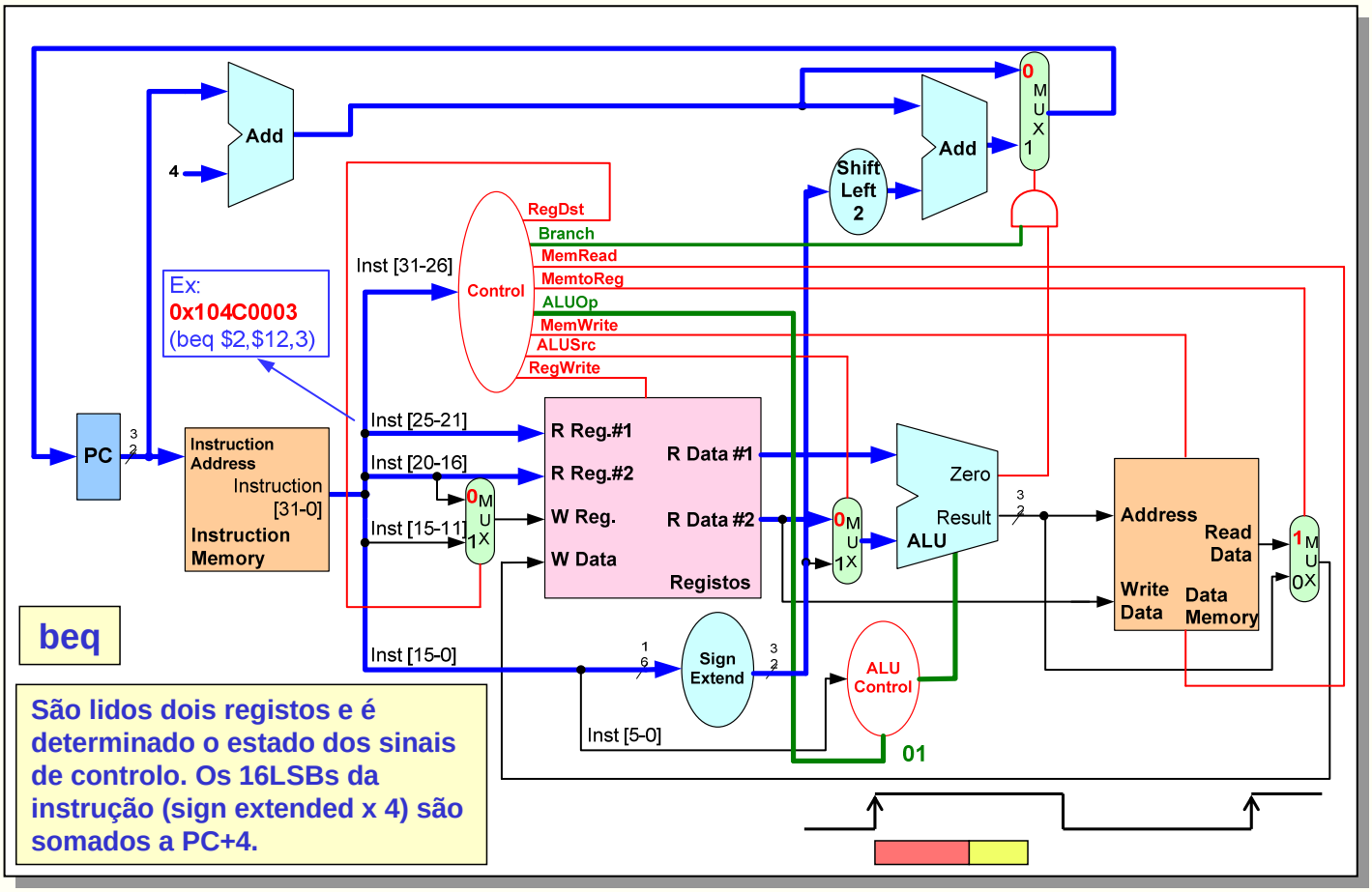

Add
4
Add
Shift Left 2
0 M U X 1
RegDst
Branch
MemRead
MemtoReg
ALUOp
MemWrite
ALUSrc
RegWrite
Control
Inst [31-26]
Ex:
0x8D43FFFC
(lw $3,-4($10)
PC
Instruction Address
Instruction [31-0]
Instruction Memory
Inst [25-21]
Inst [20-16]
Inst [15-11]
Inst [15-0]
Inst [5-0]
R Reg.#1
R Reg.#2
W Reg.
W Data
R Data #1
R Data #2
Registos
Zero
Result
ALU
Address
Read Data
Write Data
Data Memory
Sign Extend
ALU Control
00
lw
O resultado produzido pela ALU constitui o endereço de acesso à memória de dados. A memória é lida nesse endereço.

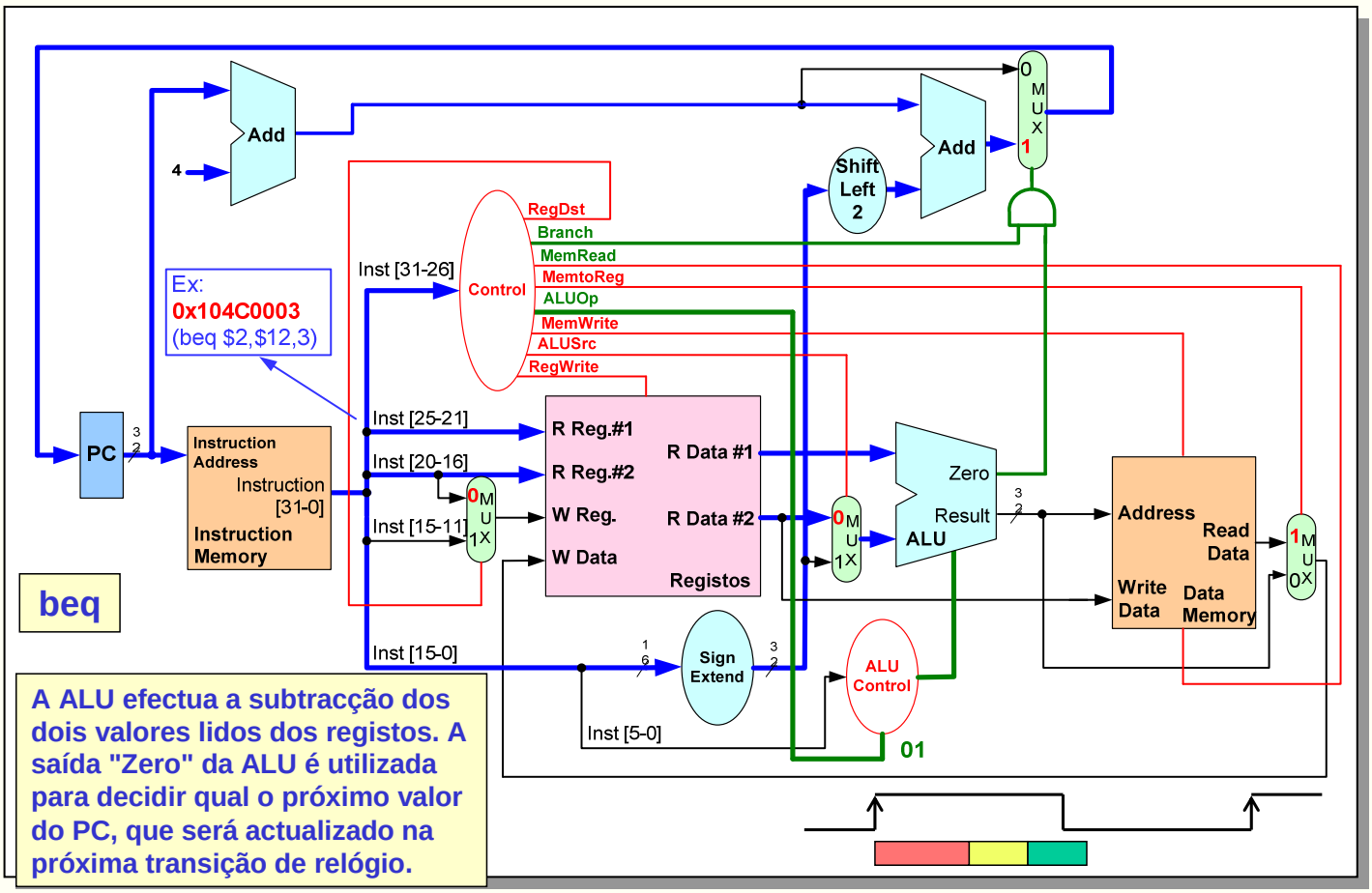

Add
4
Add
Shift Left 2
0 M U X 1
RegDst
Branch
MemRead
MemtoReg
ALUOp
MemWrite
ALUSrc
RegWrite
Control
Inst [31-26]
Ex:
0x8D43FFFC
(lw $3,-4($10)
PC
Instruction Address
Instruction [31-0]
Instruction Memory
Inst [25-21]
Inst [20-16]
Inst [15-11]
Inst [15-0]
Inst [5-0]
R Reg.#1
R Reg.#2
W Reg.
W Data
R Data #1
R Data #2
Registos
Zero
Result
ALU
Address
Read Data
Write Data
Data Memory
Sign Extend
ALU Control
00
lw
A *word* lida da memória será escrita no registo especificado nos bits 20-16 da instrução (rt), na próxima transição activa de relógio.

Exemplo 3

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

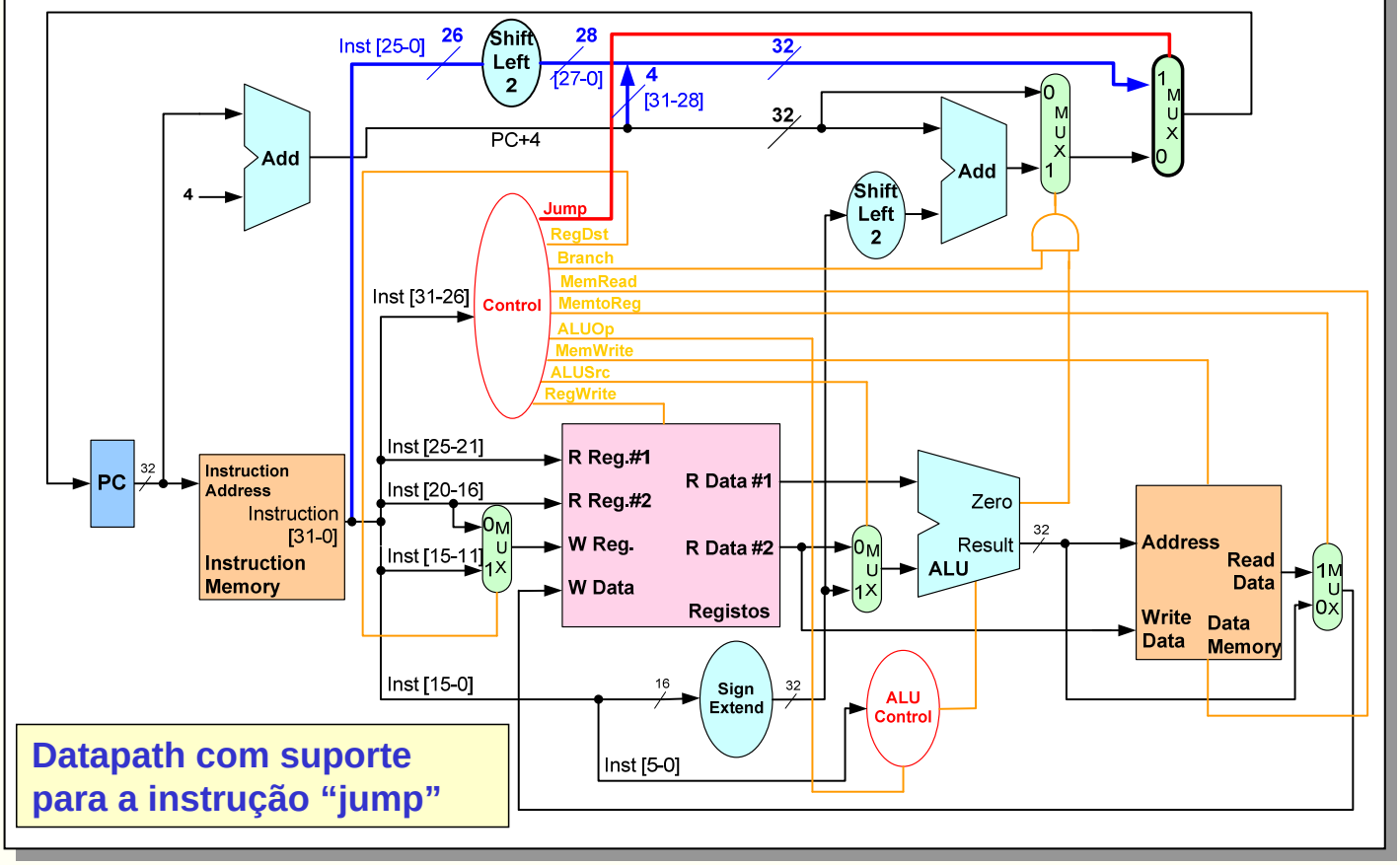


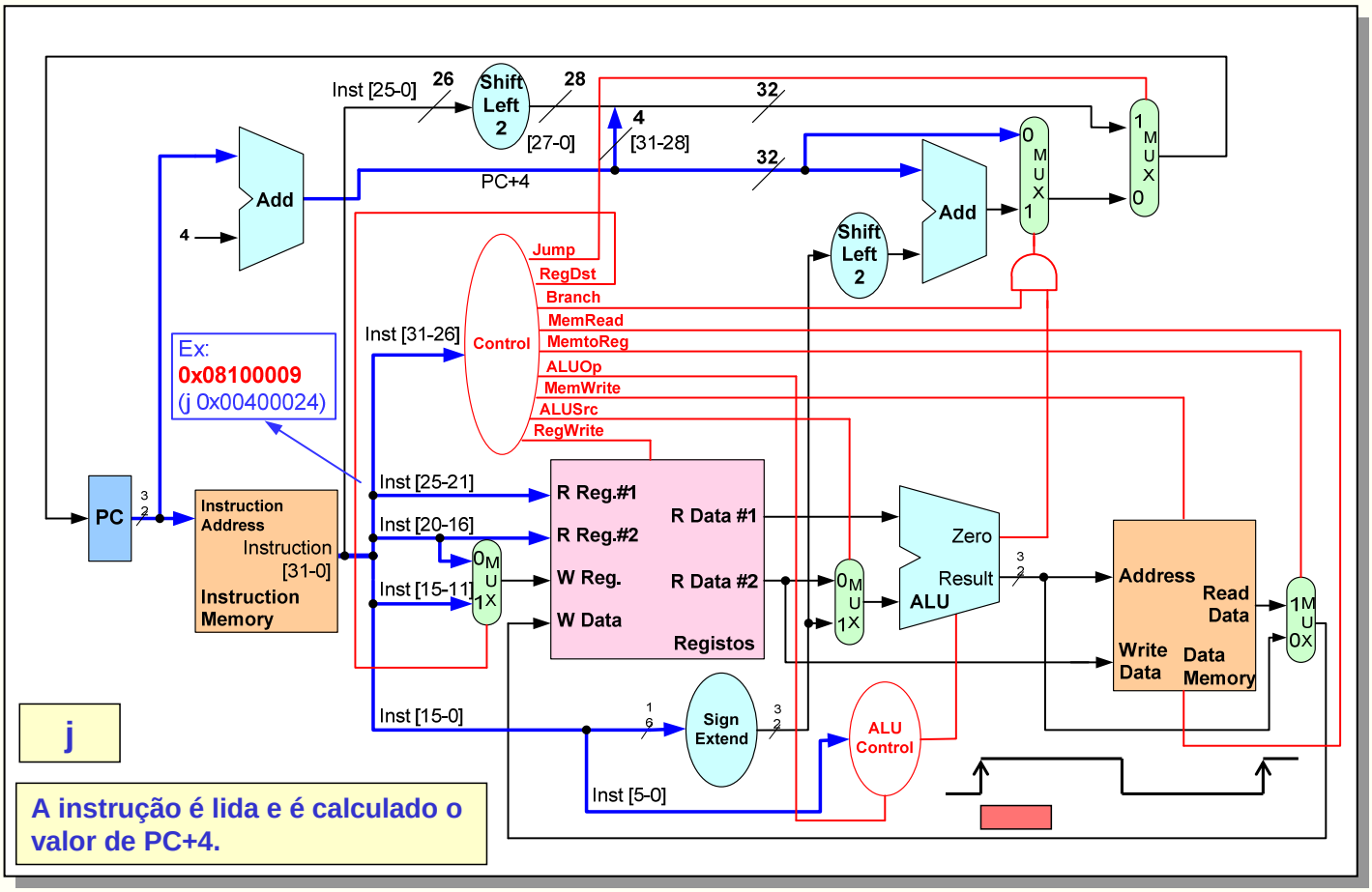

Add
4
Add
Shift Left 2
M U X
0
1
RegDst
Branch
MemRead
MemtoReg
ALUOp
MemWrite
ALUSrc
RegWrite
Control
Inst [31-26]
Ex:
0x104C0003
(beq $2,$12,3)
PC
Instruction Address
Instruction [31-0]
Instruction Memory
Inst [25-21]
Inst [20-16]
Inst [15-11]
Inst [15-0]
Inst [5-0]
R Reg.#1
R Reg.#2
W Reg.
W Data
R Data #1
R Data #2
Registos
Zero
Result
ALU
Address
Read Data
Write Data
Data Memory
Sign Extend
ALU Control
01
beq
São lidos dois registos e é determinado o estado dos sinais de controlo. Os 16LSBs da instrução (sign extended x 4) são somados a PC+4.

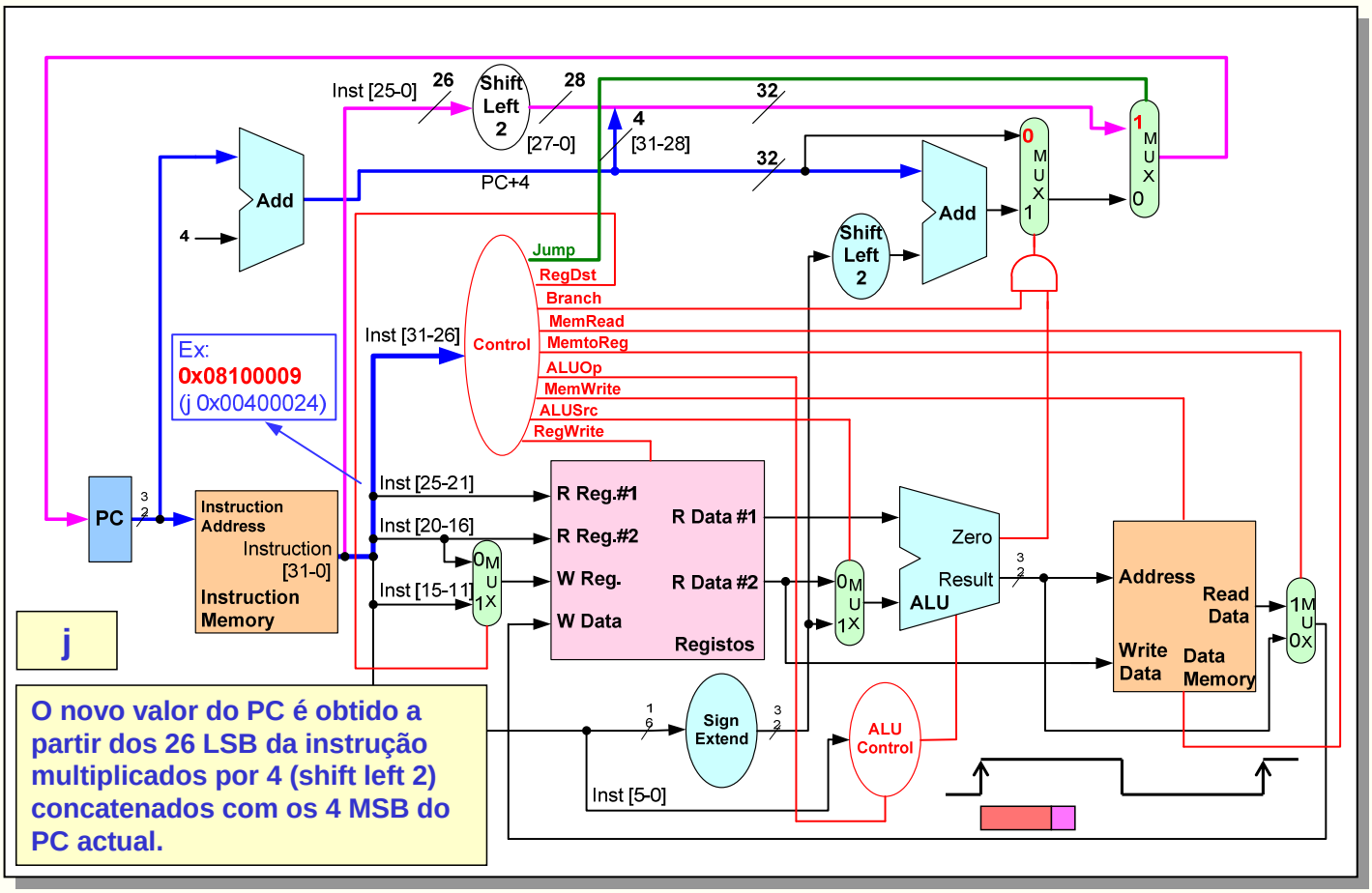

Add
4
Add
Shift Left 2
M U X
0
1
RegDst
Branch
MemRead
MemtoReg
ALUOp
MemWrite
ALUSrc
RegWrite
Control
Inst [31-26]
Ex:
0x104C0003
(beq $2,$12,3)
PC
Instruction Address
Instruction [31-0]
Instruction Memory
Inst [25-21]
Inst [20-16]
Inst [15-11]
Inst [15-0]
Inst [5-0]
R Reg.#1
R Reg.#2
W Reg.
W Data
R Data #1
R Data #2
Registos
Zero
Result
ALU
Address
Read Data
Write Data
Data Memory
Sign Extend
ALU Control
01
beq
A ALU efectua a subtracção dos dois valores lidos dos registos. A saída "Zero" da ALU é utilizada para decidir qual o próximo valor do PC, que será actualizado na próxima transição de relógio.

Datapath com suporte para a instrução "*jump*"

- A instrução "*jump*" corresponde a um caso particular de codificação (instruções do tipo J). Nestas instruções existem apenas dois campos: o campo *op* (bits 31-26) e o campo de endereço (bits 25-0)
- Nas instruções de "*jump*", o endereço alvo (*jump target address*) obtém-se pela **concatenação** dos bits **31-28** do PC+4 com os bits do campo de endereço da instrução (26 bits) multiplicados por 4.
- Será necessário acrescentar ainda um bit de saída à unidade de controlo para seleccionar a fonte de informação disponibilizada à entrada do PC
- O *datapath* simplificado, com suporte para a instrução "**j**" ("*jump*") ficará assim com a seguinte configuração:




Inst [25-0]
26
Shift Left 2
28
[27-0]
4
[31-28]
32
PC+4
Add
4
Jump
RegDst
Branch
MemRead
MemtoReg
ALUOp
MemWrite
ALUSrc
RegWrite
Control
Inst [31-26]
Ex:
0x08100009
(j 0x00400024)
PC
Instruction Address
Instruction [31-0]
Instruction Memory
Inst [25-21]
Inst [20-16]
Inst [15-11]
Inst [15-0]
Inst [5-0]
R Reg.#1
R Reg.#2
W Reg.
W Data
R Data #1
R Data #2
Registos
Zero
Result
ALU
Address
Read Data
Write Data
Data Memory
Sign Extend
ALU Control
MUX
j
A instrução é lida e é calculado o valor de PC+4.


Inst [25-0]
26
Shift Left 2
28
[27-0]
4
[31-28]
32
PC+4
Add
4
Jump
RegDst
Branch
MemRead
MemtoReg
ALUOp
MemWrite
ALUSrc
RegWrite
Control
Inst [31-26]
Ex:
0x08100009
(j 0x00400024)
PC
Instruction Address
Instruction [31-0]
Instruction Memory
Inst [25-21]
Inst [20-16]
Inst [15-11]
Inst [15-0]
Inst [5-0]
R Reg.#1
R Reg.#2
W Reg.
W Data
R Data #1
R Data #2
Registos
Zero
Result
ALU
Address
Read Data
Write Data
Data Memory
Sign Extend
ALU Control
MUX
j
O novo valor do PC é obtido a partir dos 26 LSB da instrução multiplicados por 4 (shift left 2) concatenados com os 4 MSB do PC actual.

Execução de uma instrução no datapath *single cycle* - exemplo

Endereços	Dados
(…)	(…)
0x10010030	0x63F78395
0x10010034	0xA0FCF3F0
0x10010038	0x147FAF83
(…)	(…)

Endereços	Código
(…)	(…)
0x00400020	0x00E82820
0x00400024	0x8CA30024
0x00400028	0x00681824
(…)	(…)

$PC	0x00400024
$3	0x7F421231
$4	0x15A73C49
$5	0x10010010

$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 da memória de dados é o indicado. Qual o conteúdo dos registos após a execução da instrução?

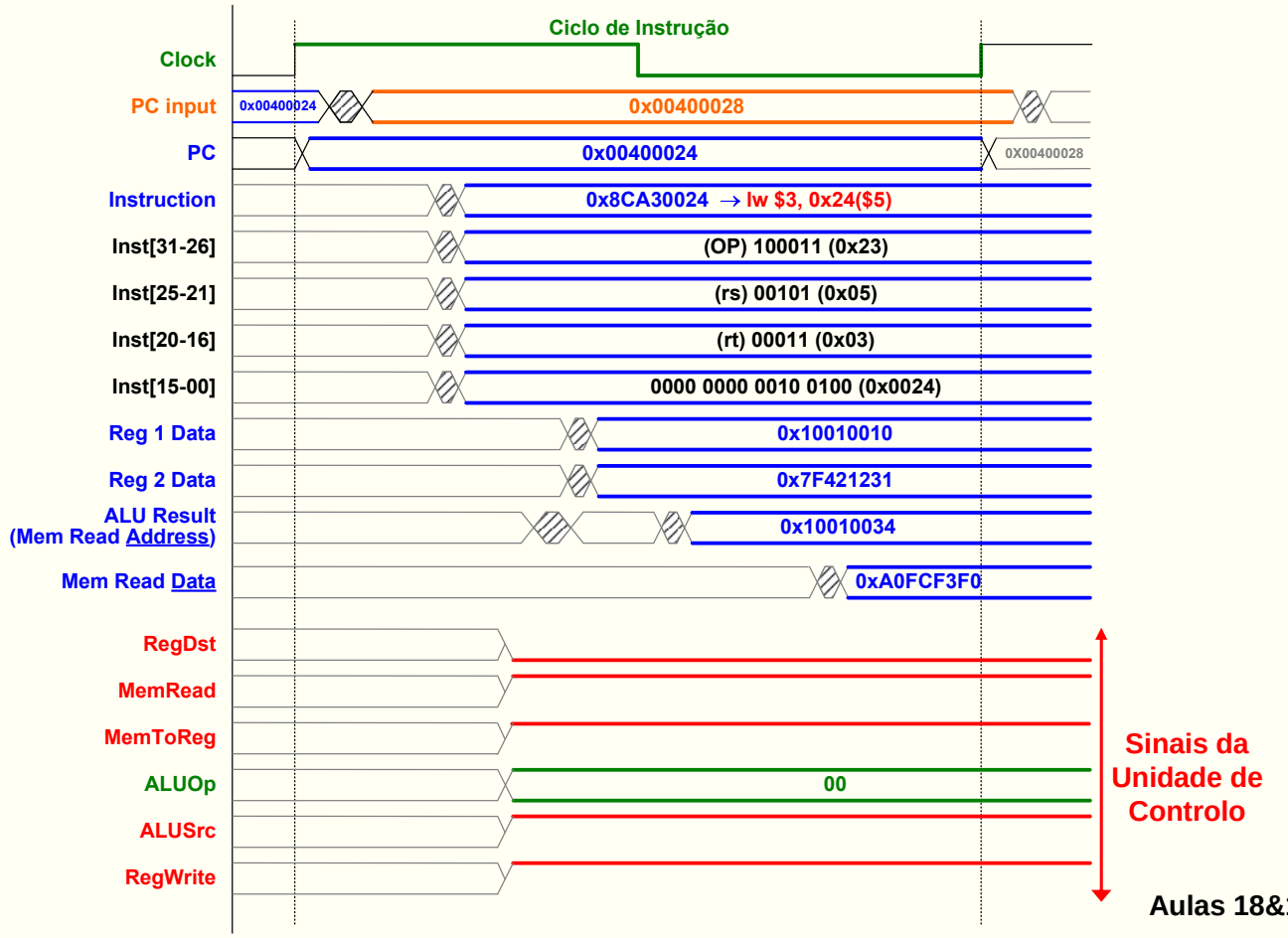
0x8CA30024 → lw $3, 0x24($5)

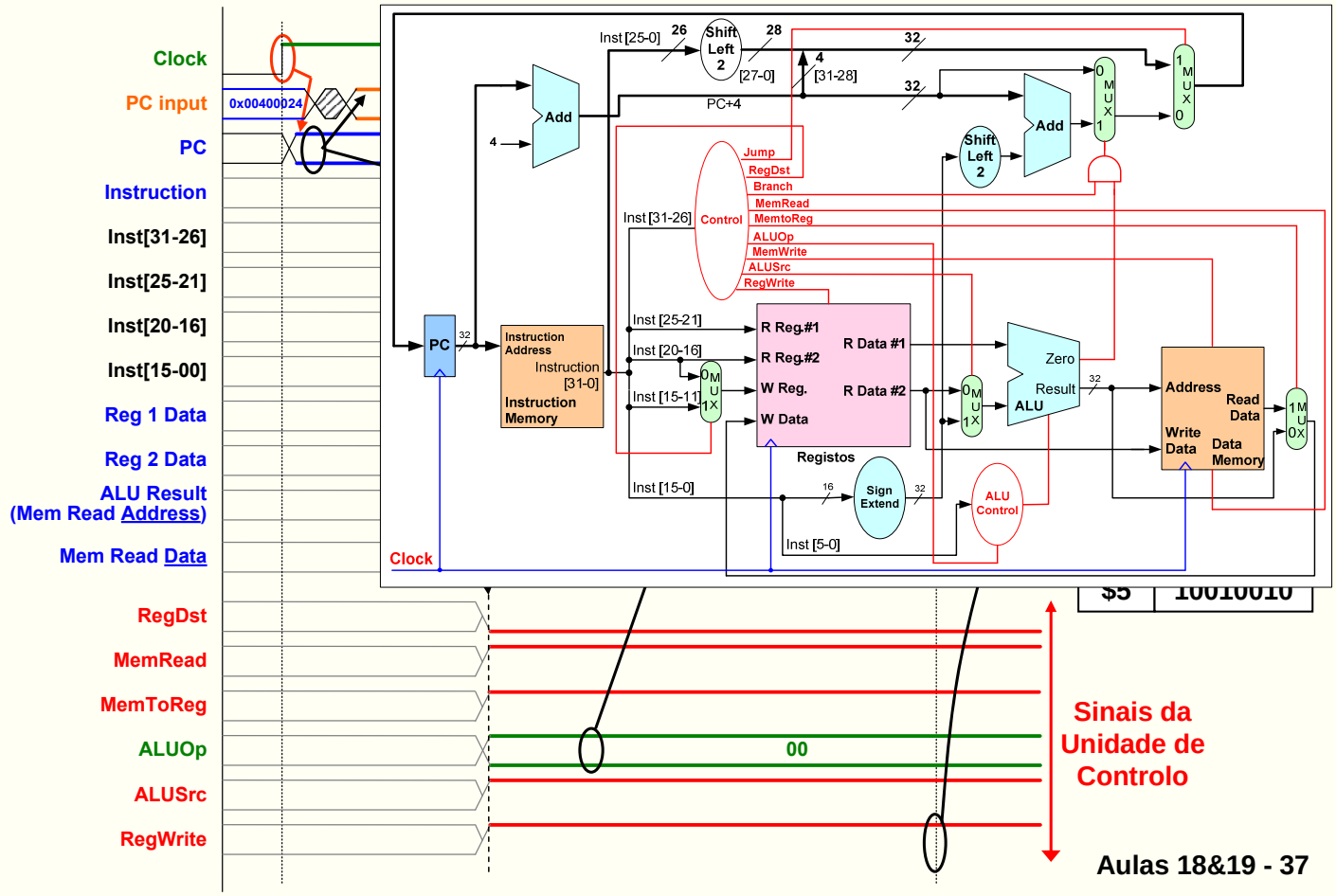
Endereço_mem: 0x10010010 + 0x24 = 0x10010034

10001100101000110000000000100100

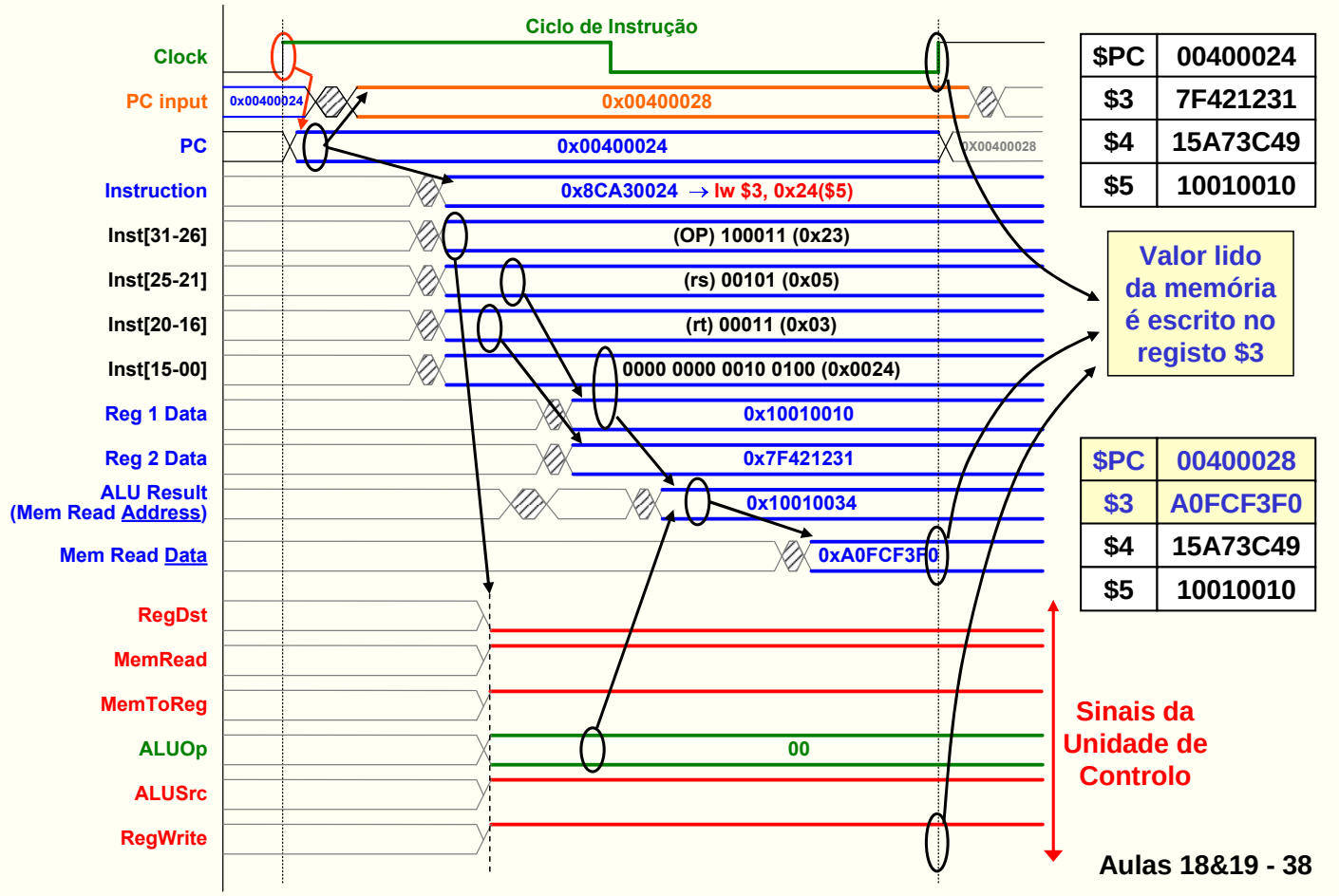
$3 = [0x10010034] = 0xA0FCF3F0

Execução de uma instrução no datapath *single cycle* – diagrama temporal





$5	10010010



$PC	00400024
$3	7F421231
$4	15A73C49
$5	10010010

Valor lido da memória é escrito no registo $3

$PC	00400028
$3	A0FCF3F0
$4	15A73C49
$5	10010010