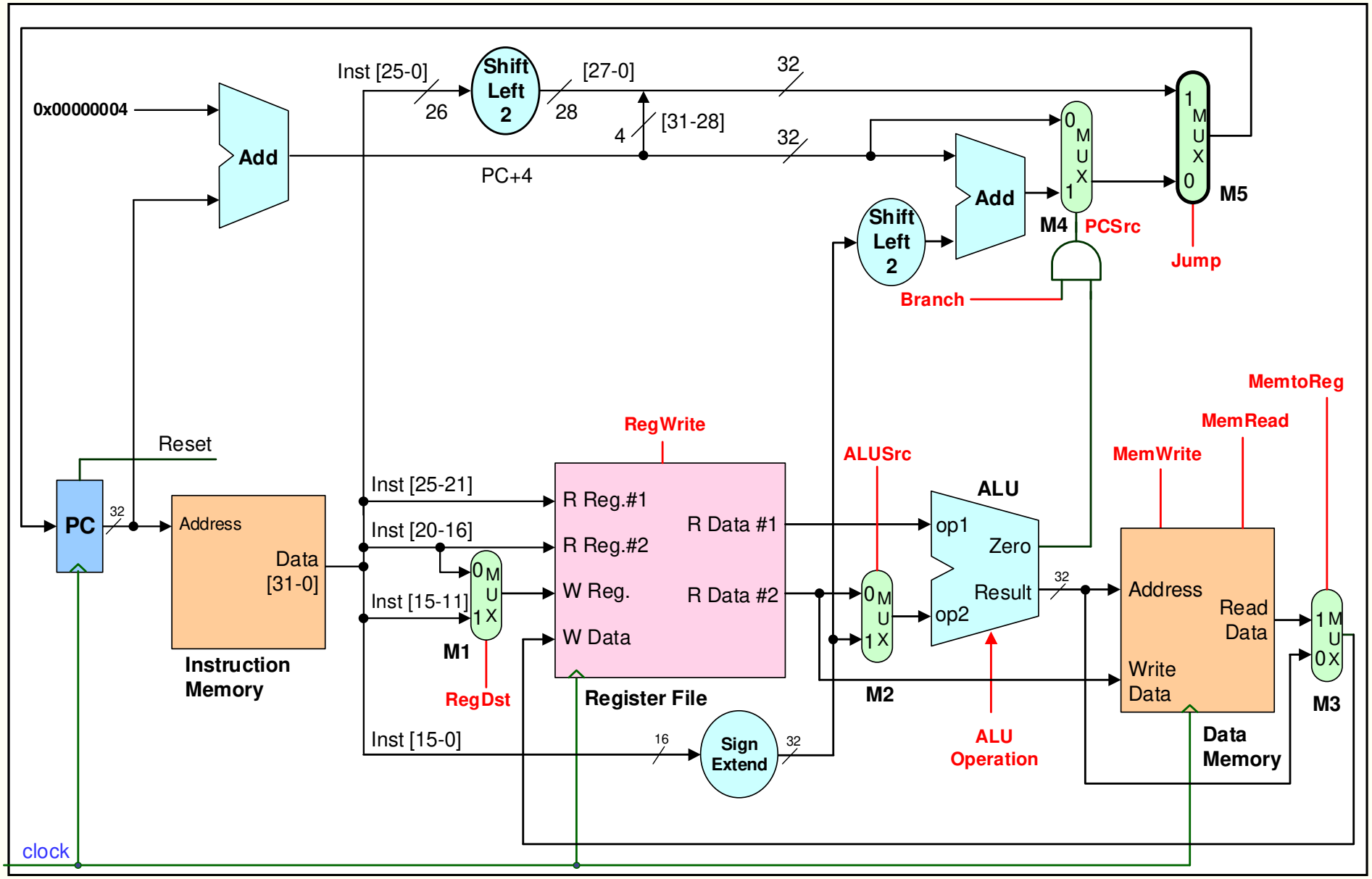
Aula 11

- A unidade de controlo do *datapath single-cycle*
- Exemplos de funcionamento do *datapath* com unidade de controlo
- Frequência máxima de funcionamento do *datapath single-cycle;* limitações da implementação *single-cycle*

Bernardo Cunha, José Luís Azevedo

Datapath single-cycle completo



Unidade de controlo

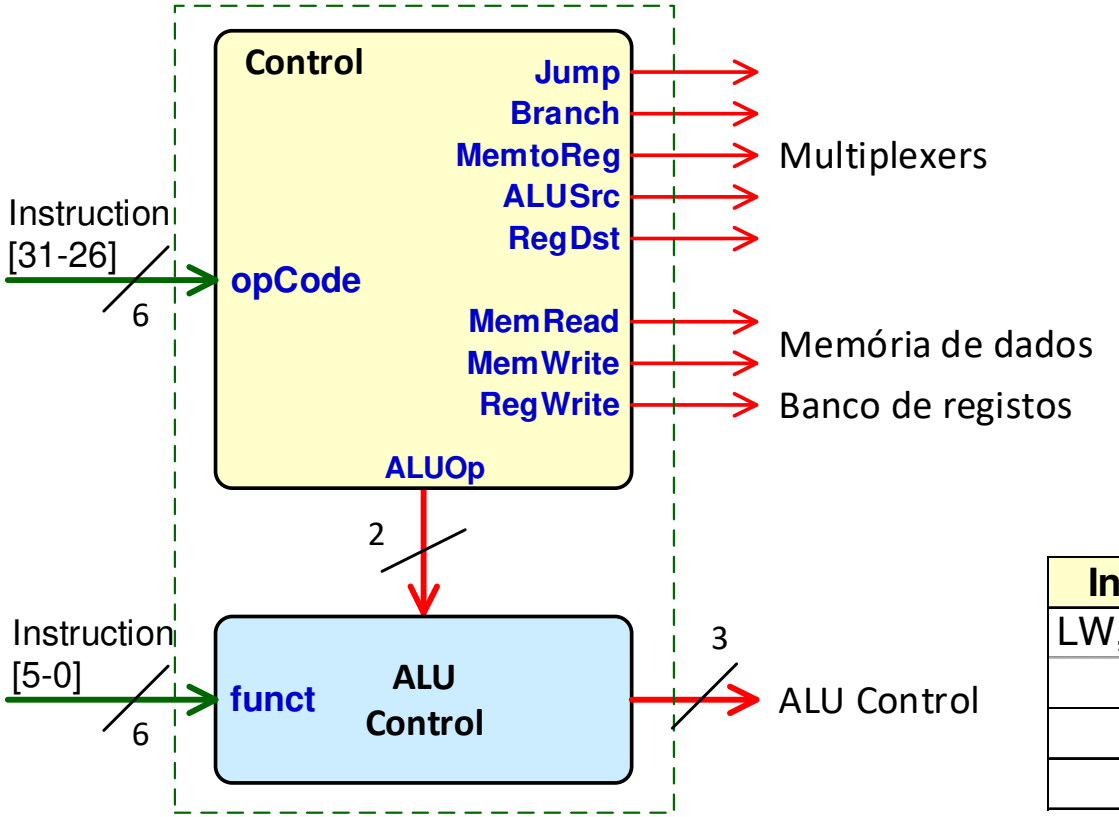
- Elementos de estado do *datapath*:
  - **PC e memória de instruções**: são acedidos em todos os ciclos de relógio, pelo que não requerem sinais explícitos
  - **Banco de registos e memória de dados**: o acesso depende da instrução e por isso requerem sinais explícitos
- A unidade de controlo gera os sinais (identificados a vermelho) para:
  - Controlar o acesso para escrita no Banco de Registos e para leitura/escrita na memória de dados
  - Definir a operação da ALU e dos *multiplexers*
- Relembrando, nos elementos de estado:
  - a **escrita** é sempre realizada de forma síncrona
  - a **leitura** é sempre realizada de forma assíncrona

Unidade de controlo

- Todas as instruções (exceto o "j") usam a ALU:
  - **LW e SW** – para calcular o endereço da memória externa (soma)
  - **Branch if equal** / **not equal** – para determinar se os operandos são iguais ou diferentes (subtração)
  - **Aritméticas e lógicas** – para efetuar a respetiva operação
- A operação a realizar na ALU depende:
  - dos campos **opcode** e **funct** nas instruções aritméticas e lógicas de tipo R (opcode=0):
    ALUControl = f(opcode, funct)
  - do campo **opcode** nas restantes instruções:
    ALUControl = f(opcode)

Unidade de controlo

- A unidade de controlo pode ser sub-dividida em duas: 1) controlo de multiplexers e elementos de estado; 2) controlo da ALU



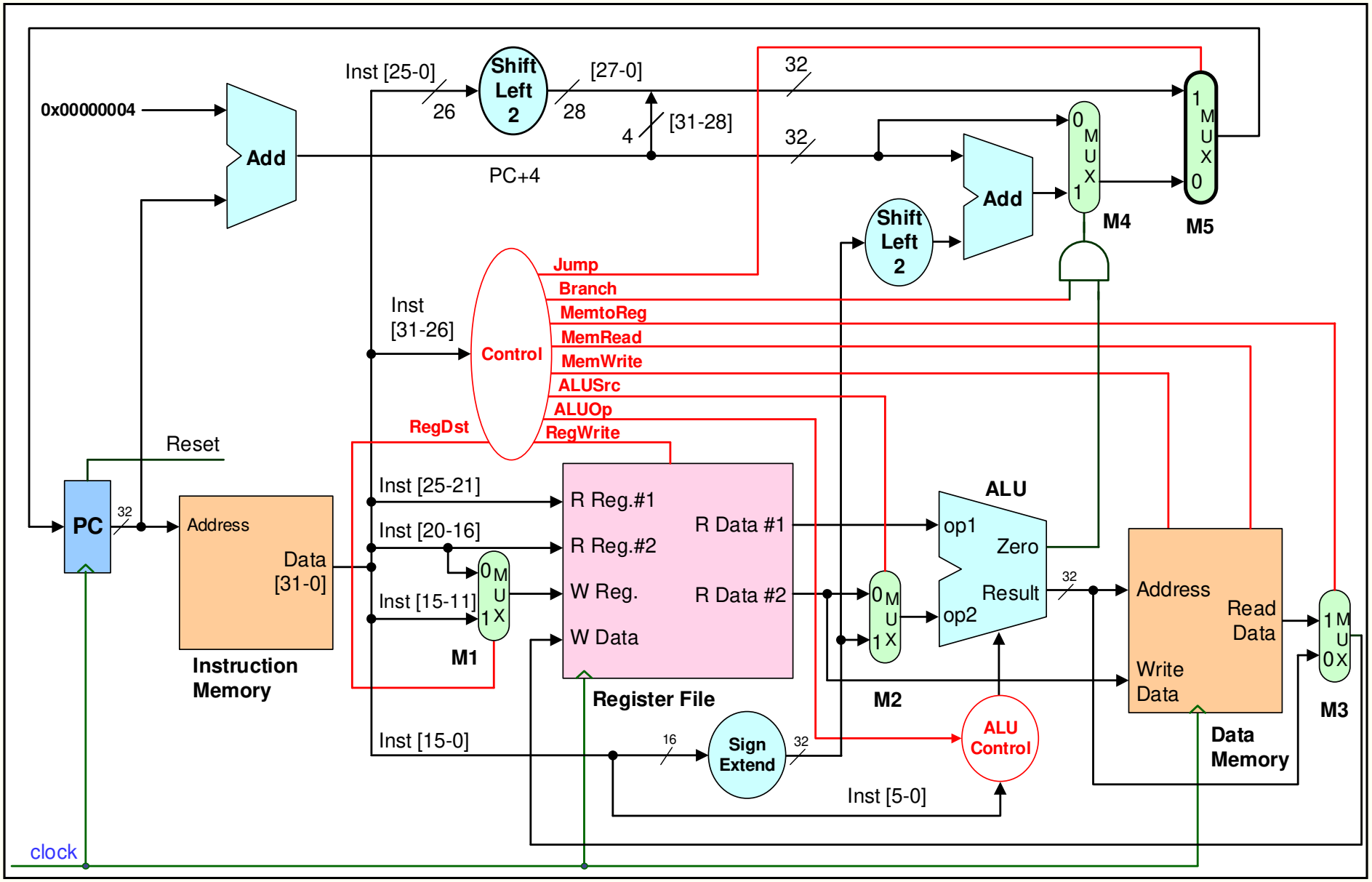
A operação na ALU é definida com 3 bits (ALU Control)

ALU Control	ALU operation
000	AND
001	OR
010	ADD
110	SUB
111	SLT

Instruction	ALUOp	ALU operation
LW, SW, ADDI	**00**	ADD
BEQ	**01**	SUB
R-Type	**10**	Depends on "**funct**"
SLTI	**11**	SLT

- A operação da ALU é definida em conjunto com a unidade de controlo principal, em função dos campos "opcode" e "funct"

Datapath single-cycle com unidade de controlo



Unidade de controlo da ALU

- A relação entre o tipo de instruções, o campo "`funct`", a operação efetuada pela ALU e os sinais de controlo da mesma, pode ser resumida pela tabela seguinte

ALU Control	ALU operation
000	AND
001	OR
010	ADD
110	SUB
111	SLT

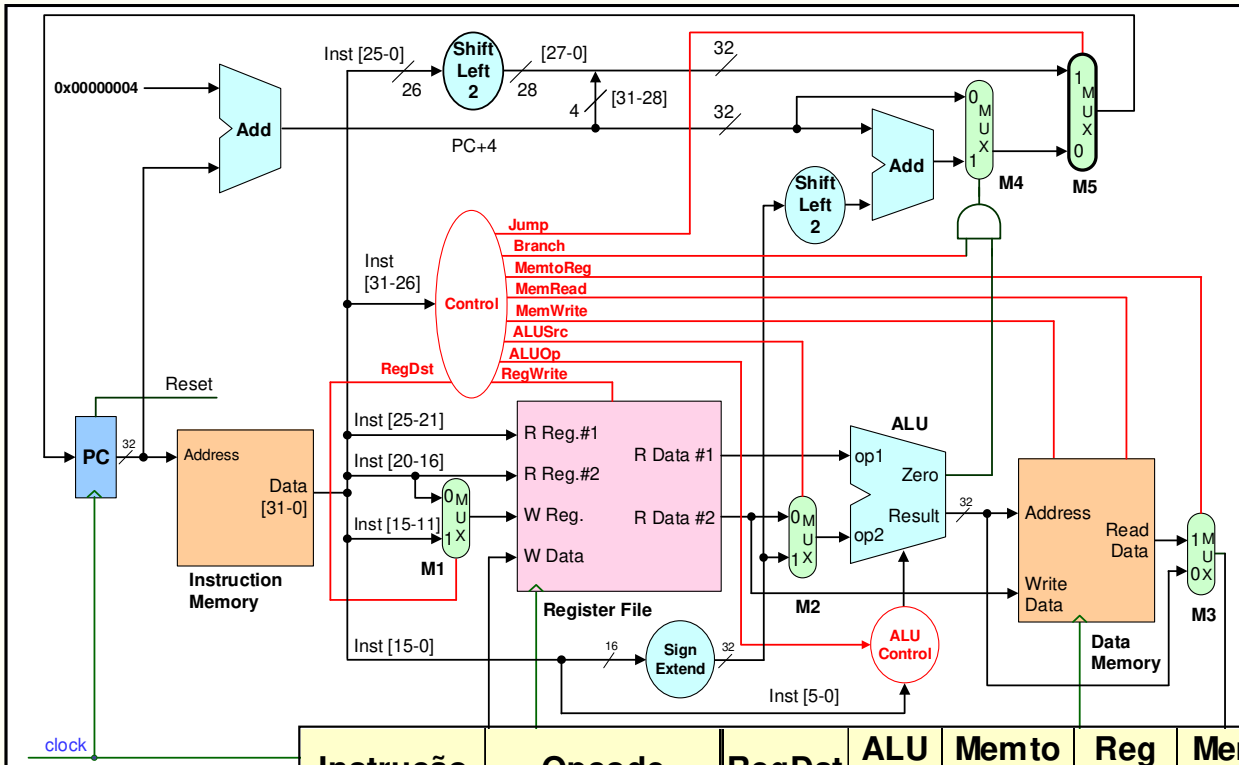
Instruction	opcode	funct	ALU Operation	ALUOp	ALU Control
load word	100011 ("**lw**")	xxxxxx	**add**	**00**	**010**
store word	101011 ("**sw**")	xxxxxx	**add**	**00**	**010**
addi	001000 ("**addi**")	xxxxxx	**add**	**00**	**010**
branch if equal	000100 ("**beq**")	xxxxxx	**subtract**	**01**	**110**
add	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**
jump	000010 ("**j**")	xxxxxx	**-**	**xx**	**xxx**

Unidade de controlo principal

- É necessário especificar um total de oito sinais de controlo (para além do ALUOp):

Sinal	Efeito quando não ativo ('0')	Efeito quando ativo ('1')
MemRead	Nenhum (barramento de dados da memória em alta impedância)	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 substituído pelo valor apresentado à entrada
RegWrite	Nenhum	O registo indicado no endereço de escrita é alterado pelo valor presente na entrada de dados
RegDst	O endereço do registo destino provém do campo "**rt**"	O endereço do registo destino provém do campo "**rd**"
ALUSrc	O segundo operando da ALU provém da segunda saída do *Register File*	O segundo operando da ALU provém dos 16 bits menos significativos da instrução após extensão do sinal
MemtoReg	O valor apresentado para escrita no registo destino provém da ALU	O valor apresentado na entrada de dados dos registos internos provém da memória externa
Branch	Nenhum	Indica que a instrução é um branch condicional
PCSrc	O PC é substituido pelo seu valor actual mais 4	O PC é substituido pelo resultado do somador que calcula o endereço alvo do *branch* condicional
Jump	Nenhum	Indica que a instrução é um *jump* incondicional

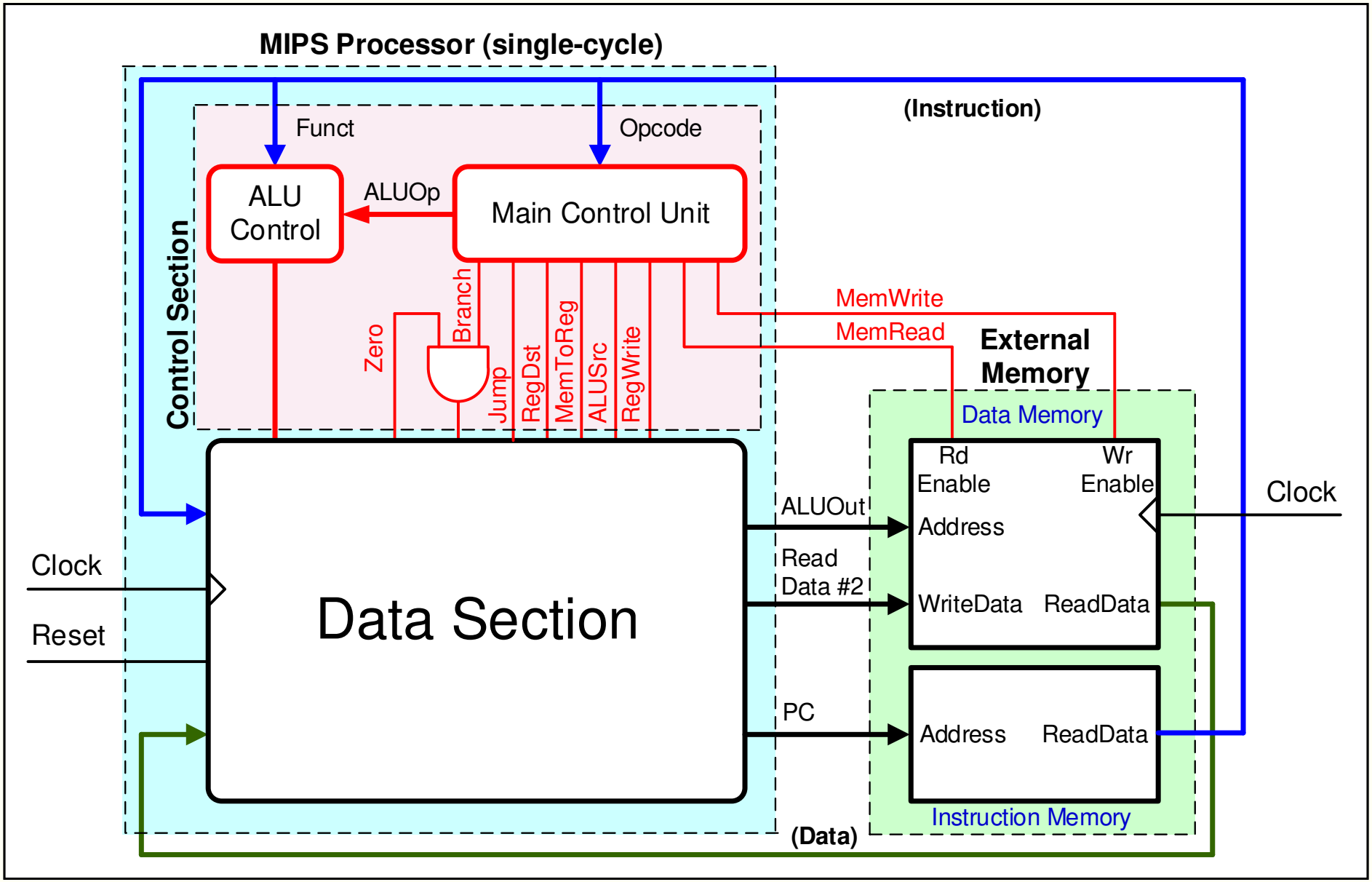
Unidade de controlo principal



Sinais gerados pela unidade controlo em função da instrução

Instrução	Opcode	RegDst	ALU Src	Memto Reg	Reg Write	Mem Read	Mem Write	Branch	Jump	ALUOp[1..0]
R - Format	000000 (0x00)	1	0	0	1	0	0	0	0	10
lw	100011 (0x23)	0	1	1	1	1	0	0	0	00
sw	101011 (0x2B)	X	1	X	0	0	1	0	0	00
addi	001000 (0x08)	0	1	0	1	0	0	0	0	00
beq	000100 (0x04)	X	0	X	0	0	0	1	0	01
slti	001010 (0x0A)	0	1	0	1	0	0	0	0	11
j	000010 (0x02)	X	X	X	0	0	0	X	1	XX

Visão global do processador



Análise do funcionamento do *datapath*

- A execução de qualquer uma das instruções suportadas ocorre no intervalo de tempo correspondente a um único ciclo de relógio: tem início numa transição ativa do relógio e termina na transição ativa seguinte
- Para simplificar a análise podemos, no entanto, considerar que os elementos operativos são usados em sequência, ao longo de várias operações
- A sequência de operações culmina com:
  - escrita no Banco de Registos: instruções tipo R, LW, ADDI, SLTI
  - escrita na Memória de Dados: SW
- O *Program Counter* é sempre atualizado com:
  - endereço-alvo da instrução BEQ, se os registos forem iguais (*branch taken*), ou PC+4 se forem diferentes (*branch not taken*)
  - endereço-alvo da instrução J
  - PC+4 nas restantes instruções

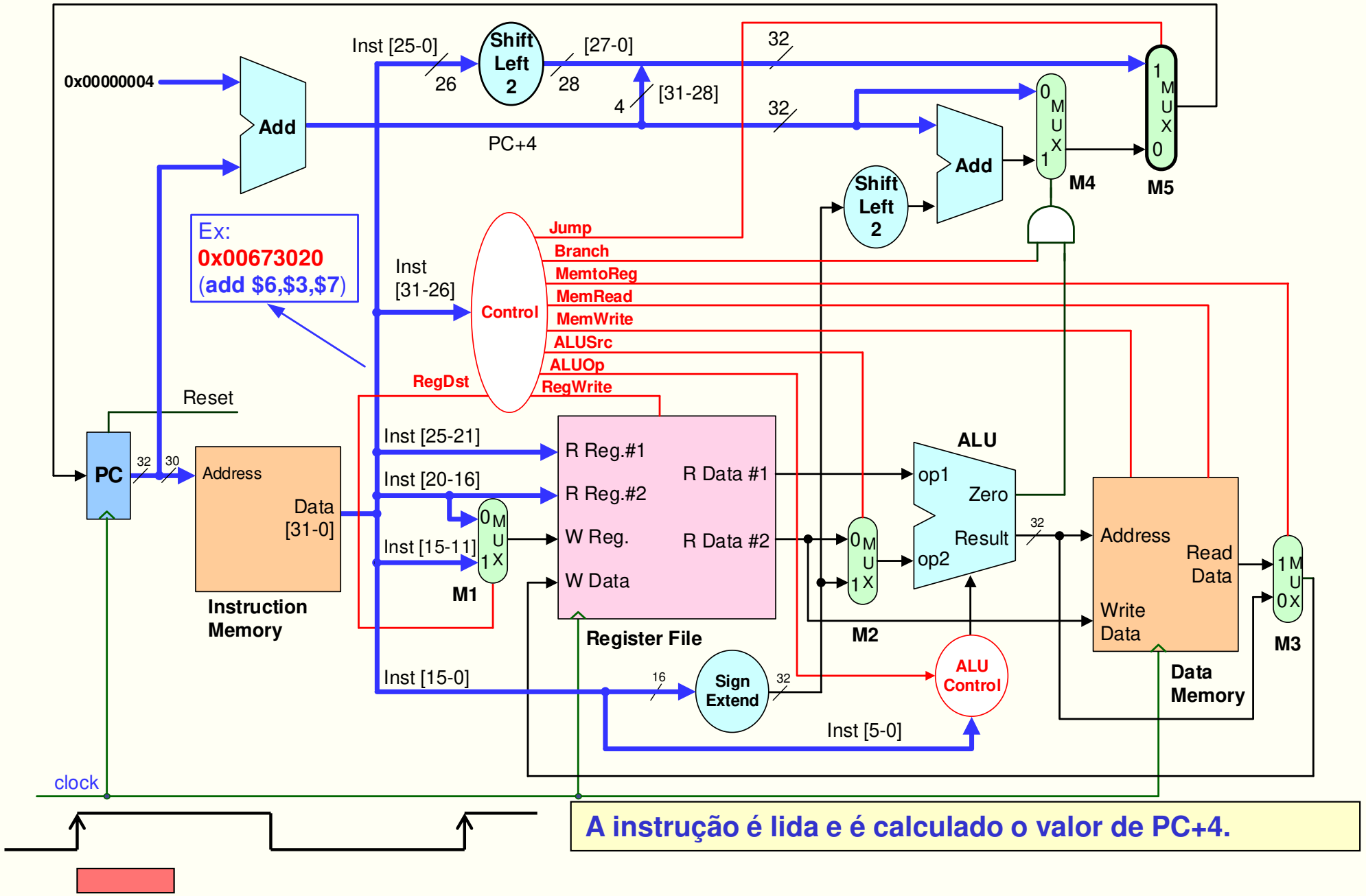
Análise do funcionamento do *datapath* – operações

- *Fetch* de uma instrução e cálculo do endereço da próxima instrução
- Leitura de dois registos do Banco de Registos
- Operação na ALU: a ALU opera sobre dois valores (a origem do segundo operando depende do tipo de instrução que estiver a ser executada)
- O resultado da operação efetuada na ALU:
  - é escrito no Banco de Registos (**R-Type**, **addi** e **slti**)
  - é usado como endereço para escrever na memória de dados (**sw**)
  - é usado como endereço para fazer uma leitura da memória de dados (**lw**) - o valor lido da memória de dados é depois escrito no Banco de Registos
  - é usado para decidir qual o próximo valor do PC (**beq** / **bne**): BTA ou PC+4

Funcionamento do *datapath* nas instruções tipo R

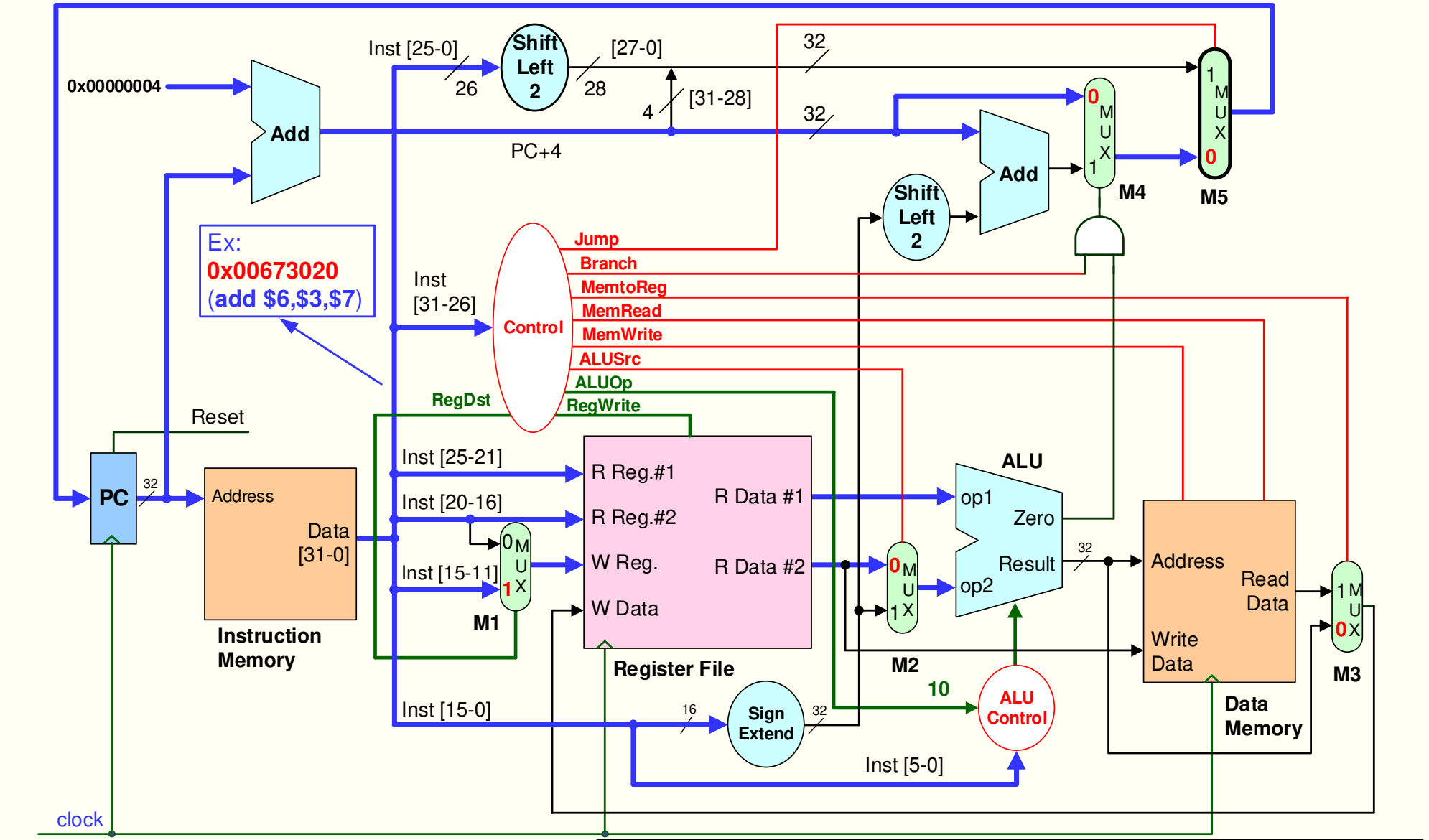
- A instrução é lida e é calculado o valor de PC+4
- São lidos dois registos e a unidade de controlo determina, a partir do *opcode* (**bits 31-26**), o estado dos sinais de controlo
- A ALU opera sobre os dados lidos dos dois registos, de acordo com a função codificada no campo *funct* (**bits 5-0**) da instrução
- O resultado produzido pela ALU será escrito no registo especificado nos **bits 15-11** da instrução ("**rd**"), na próxima transição ativa do relógio

Funcionamento do *datapath* nas instruções tipo R (1)



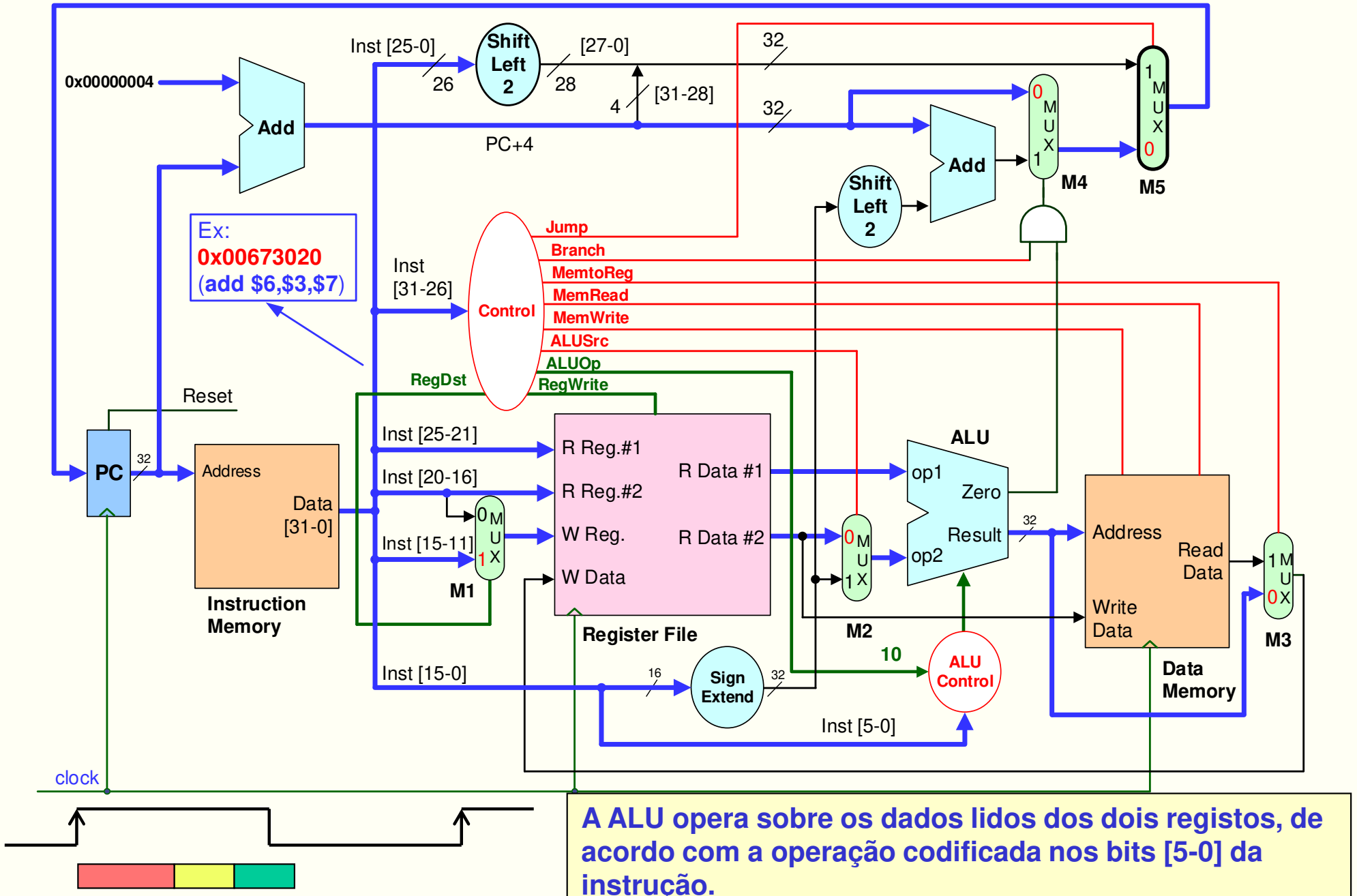
A instrução é lida e é calculado o valor de PC+4.

Funcionamento do *datapath* nas instruções tipo R (2)



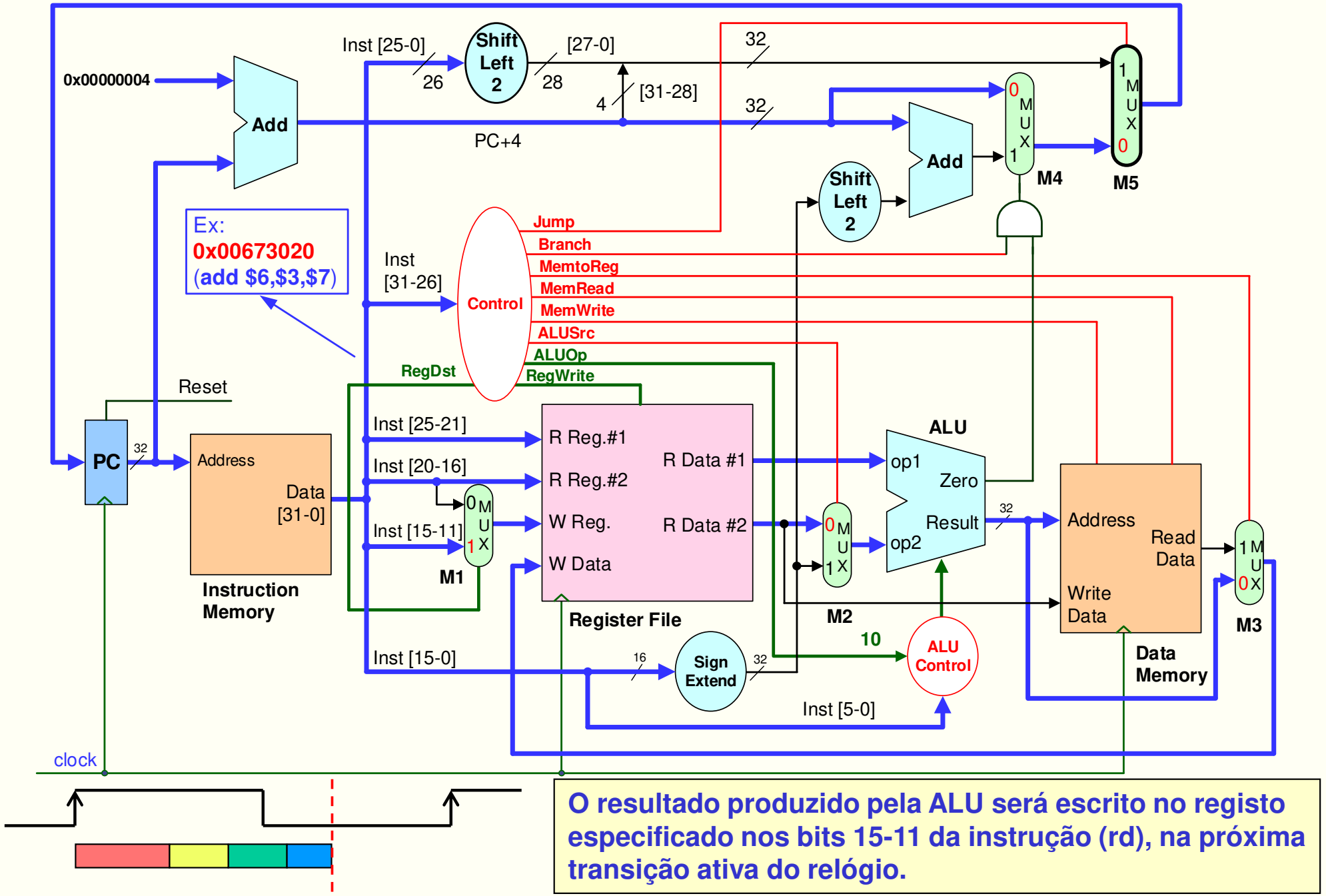
São lidos dois registos e a unidade de controlo determina, a partir do opcode (bits 31-26), o estado dos sinais de controlo.

Funcionamento do *datapath* nas instruções tipo R (3)



A ALU opera sobre os dados lidos dos dois registos, de acordo com a operação codificada nos bits [5-0] da instrução.

Funcionamento do *datapath* nas instruções tipo R (4)

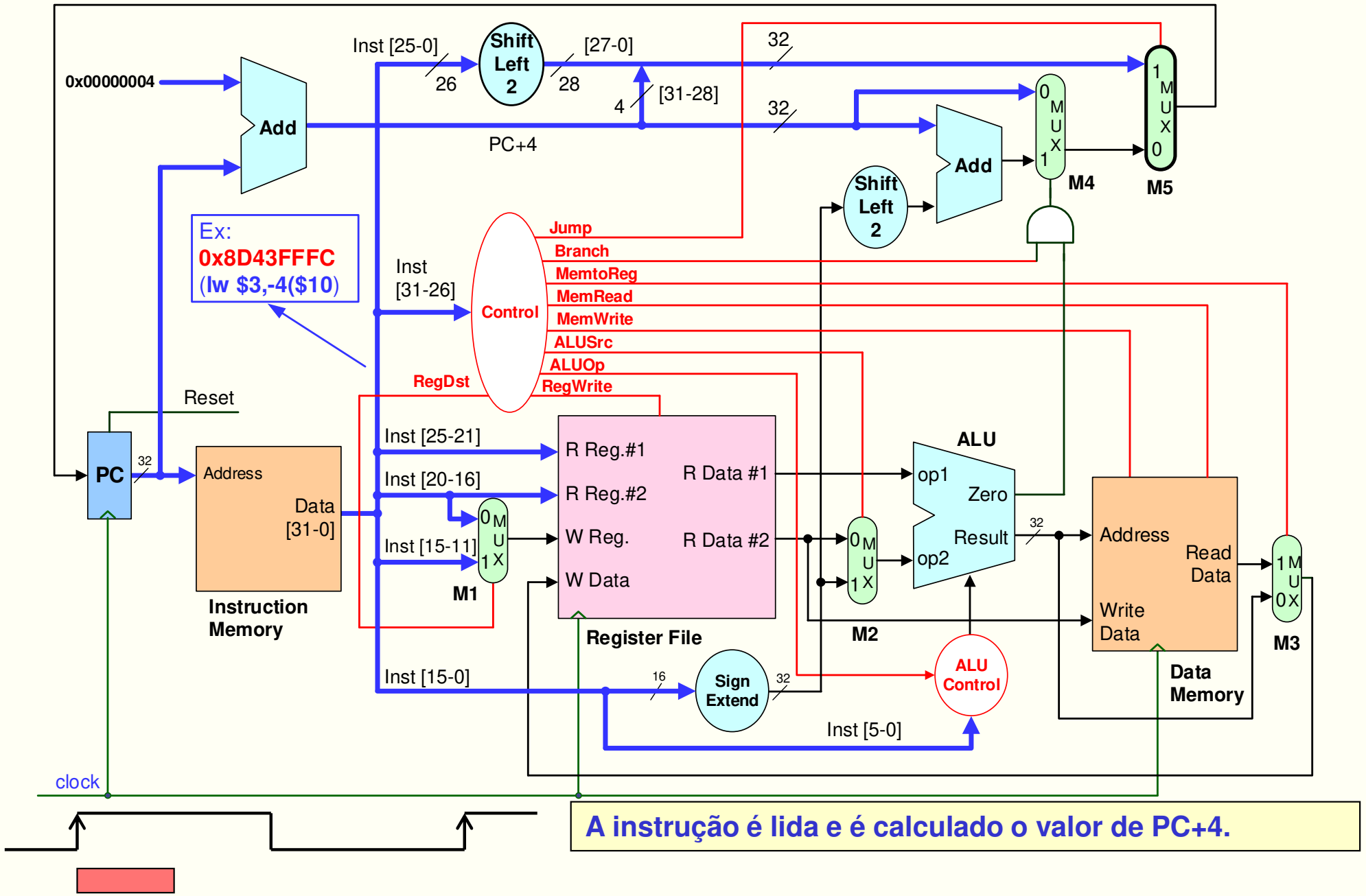


O resultado produzido pela ALU será escrito no registo especificado nos bits 15-11 da instrução (rd), na próxima transição ativa do relógio.

Funcionamento do *datapath* na instrução LW

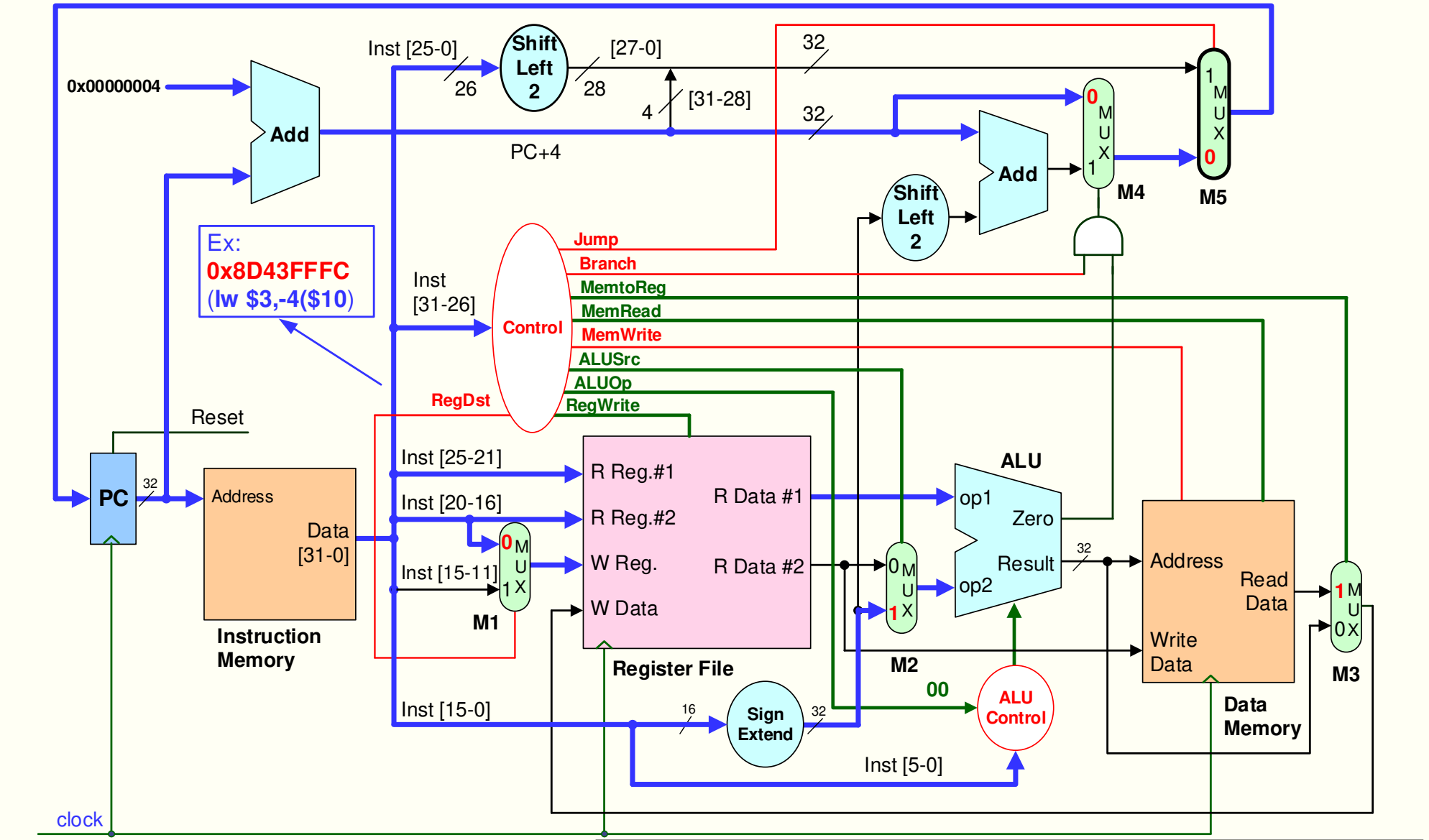
- A instrução é lida e é calculado o valor de PC+4.
- É lido um registo e a unidade de controlo determina, a partir do *opcode*, o estado dos sinais de controlo.
- A ALU soma o valor lido do registo especificado nos **bits 25-21** ("**rs**") com os 16 bits (estendidos com sinal para 32) do campo *offset* da instrução (**bits15-0**).
- O resultado produzido pela ALU constitui o endereço de acesso à memória de dados. A memória é lida nesse endereço (leitura assíncrona).
- A *word* lida da memória será escrita no registo especificado nos **bits 20-16** da instrução ("**rt**"), na próxima transição ativa do relógio.

Funcionamento do *datapath* na instrução LW (1)



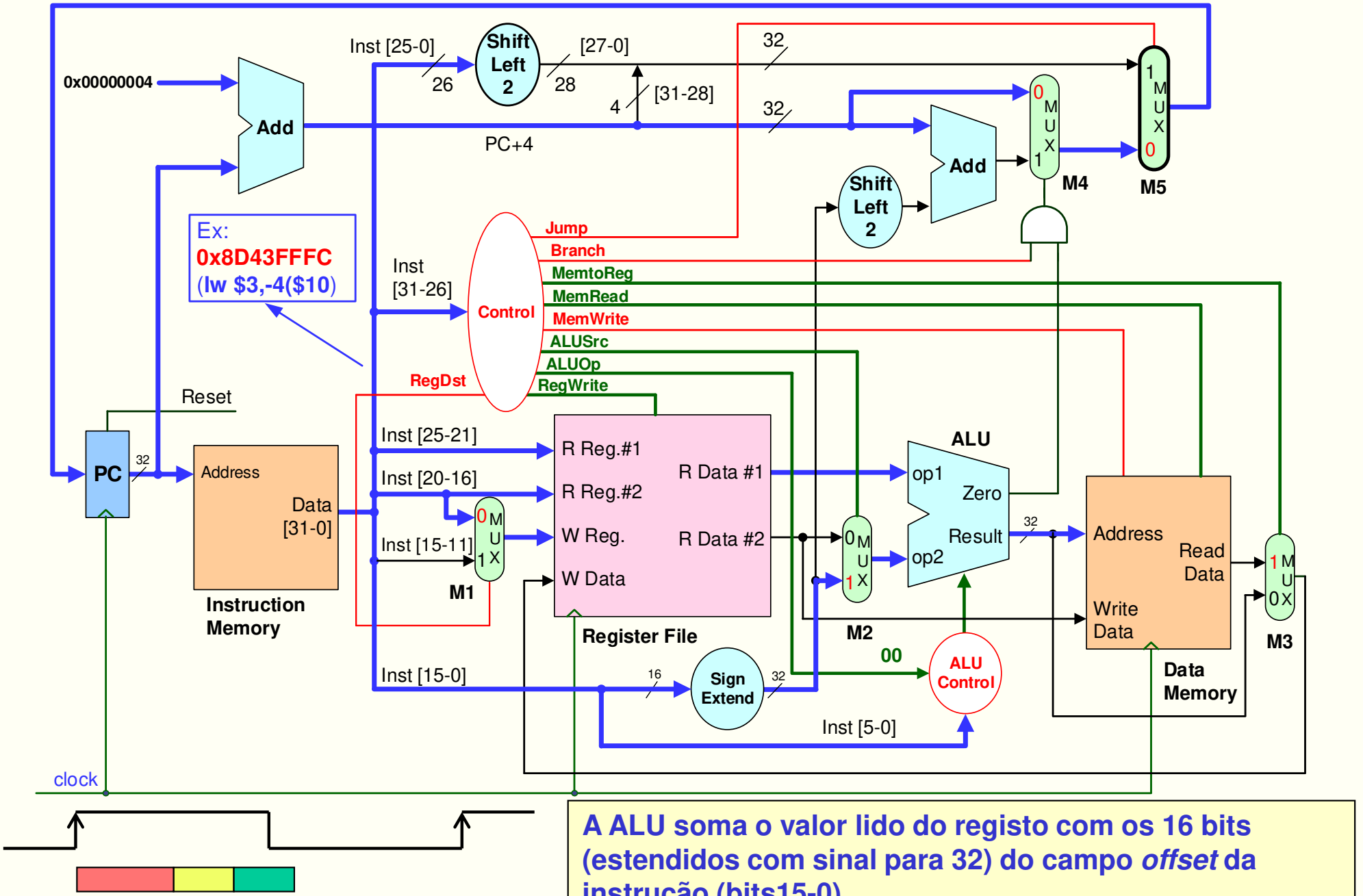
A instrução é lida e é calculado o valor de PC+4.

Funcionamento do *datapath* na instrução LW (2)



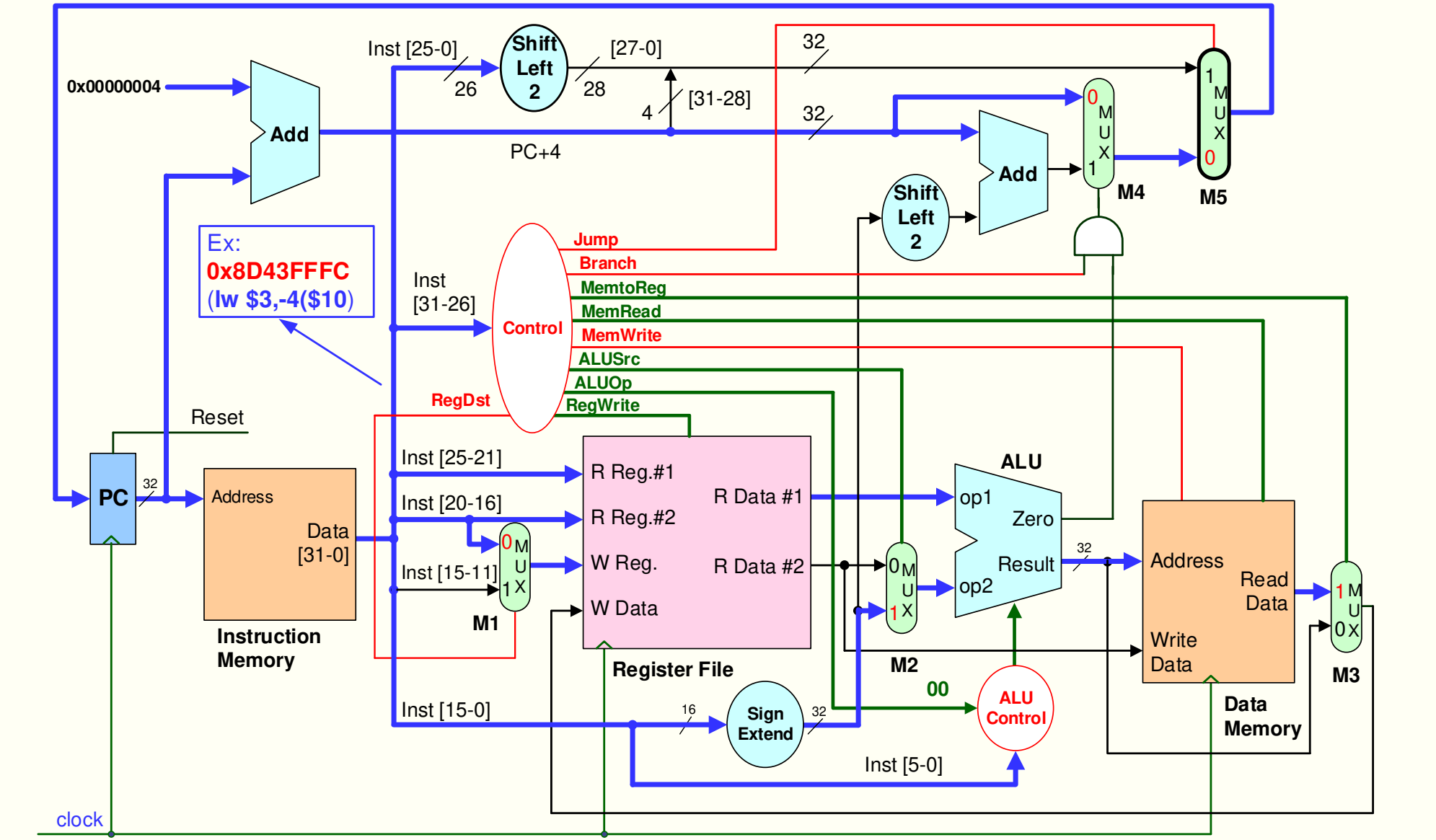
São lidos dois registos e a unidade de controlo determina, a partir do opcode (bits 31-26), o estado dos sinais de controlo.

Funcionamento do *datapath* na instrução LW (3)



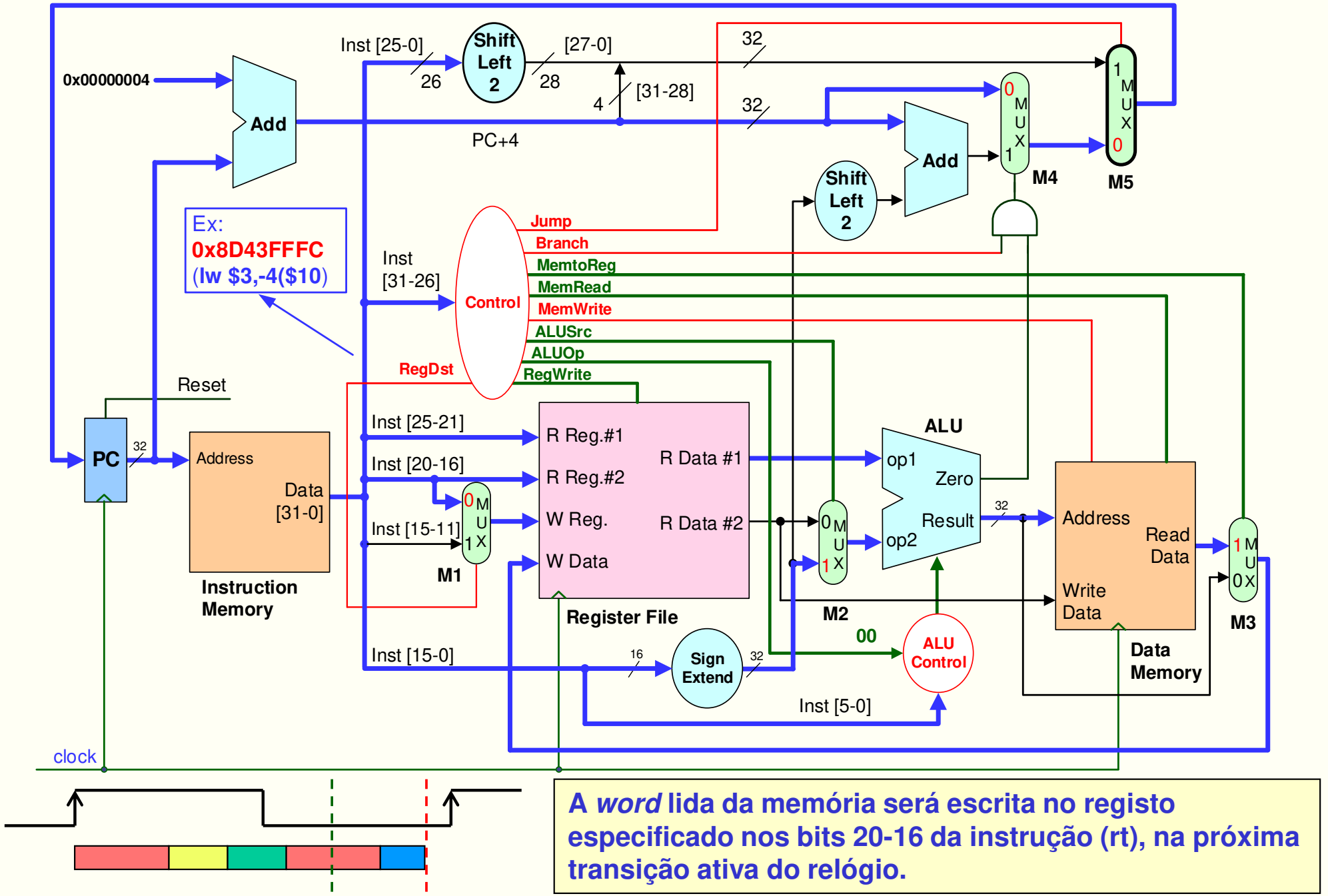
A ALU soma o valor lido do registo com os 16 bits (estendidos com sinal para 32) do campo *offset* da instrução (bits15-0).

Funcionamento do *datapath* na instrução LW (4)



O resultado produzido pela ALU constitui o endereço de acesso à memória de dados. A memória é lida nesse endereço.

Funcionamento do *datapath* na instrução LW (5)

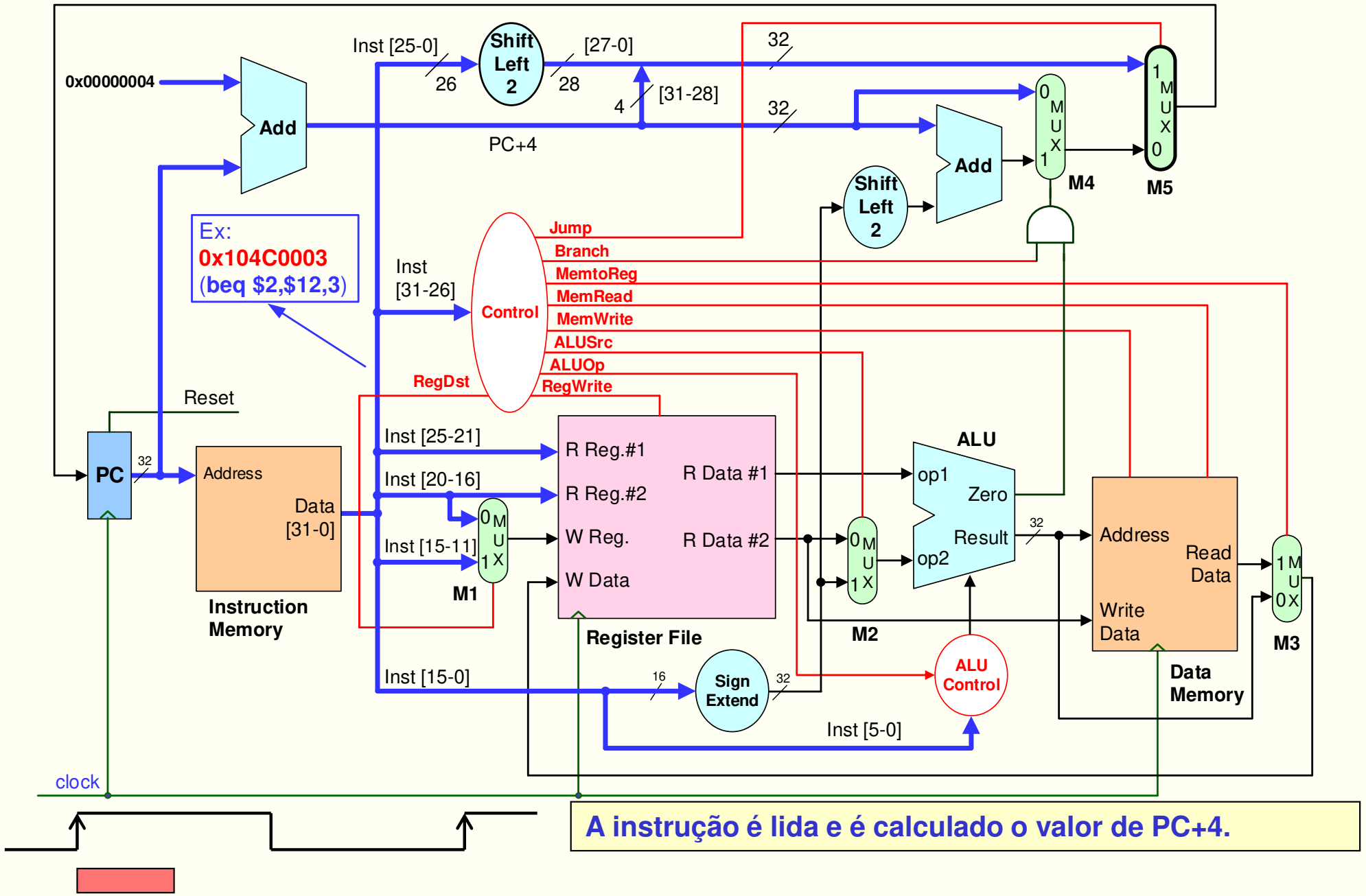


A *word* lida da memória será escrita no registo especificado nos bits 20-16 da instrução (rt), na próxima transição ativa do relógio.

Funcionamento do *datapath* na instrução BEQ

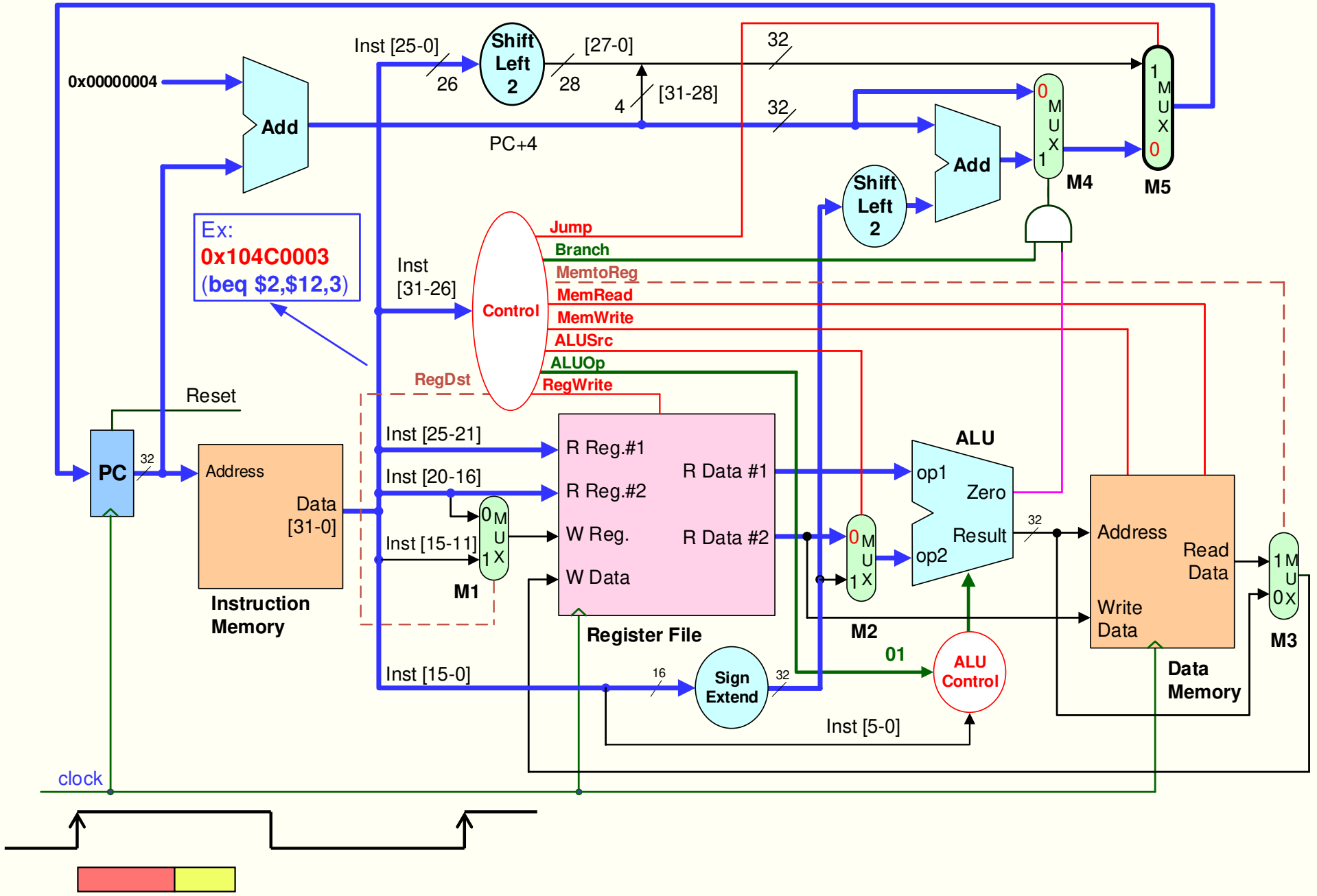
- A instrução é lida e é calculado o valor de PC+4
- São lidos dois registos e é determinado o estado dos sinais de controlo. Os 16 LSbits da instrução (sign extended x 4) são somados a PC+4 (BTA)
- A ALU faz a subtração dos dois valores lidos dos registos
- A saída "Zero" da ALU é utilizada para decidir qual o próximo valor do PC, que será atualizado na próxima transição ativa do relógio

Funcionamento do *datapath* na instrução BEQ (1)

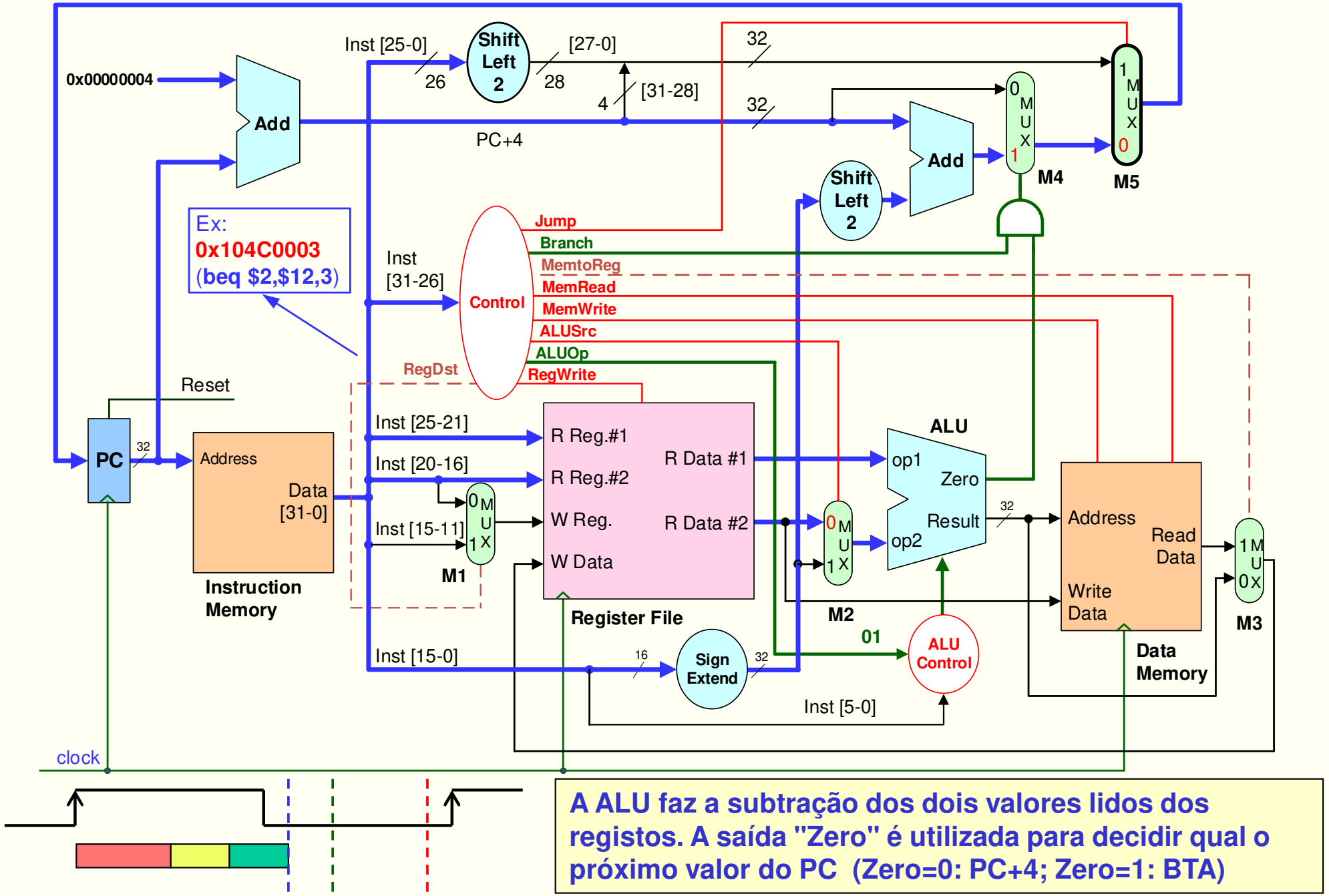


A instrução é lida e é calculado o valor de PC+4.

Funcionamento do *datapath* na instrução BEQ (2)



Funcionamento do *datapath* na instrução BEQ (3)

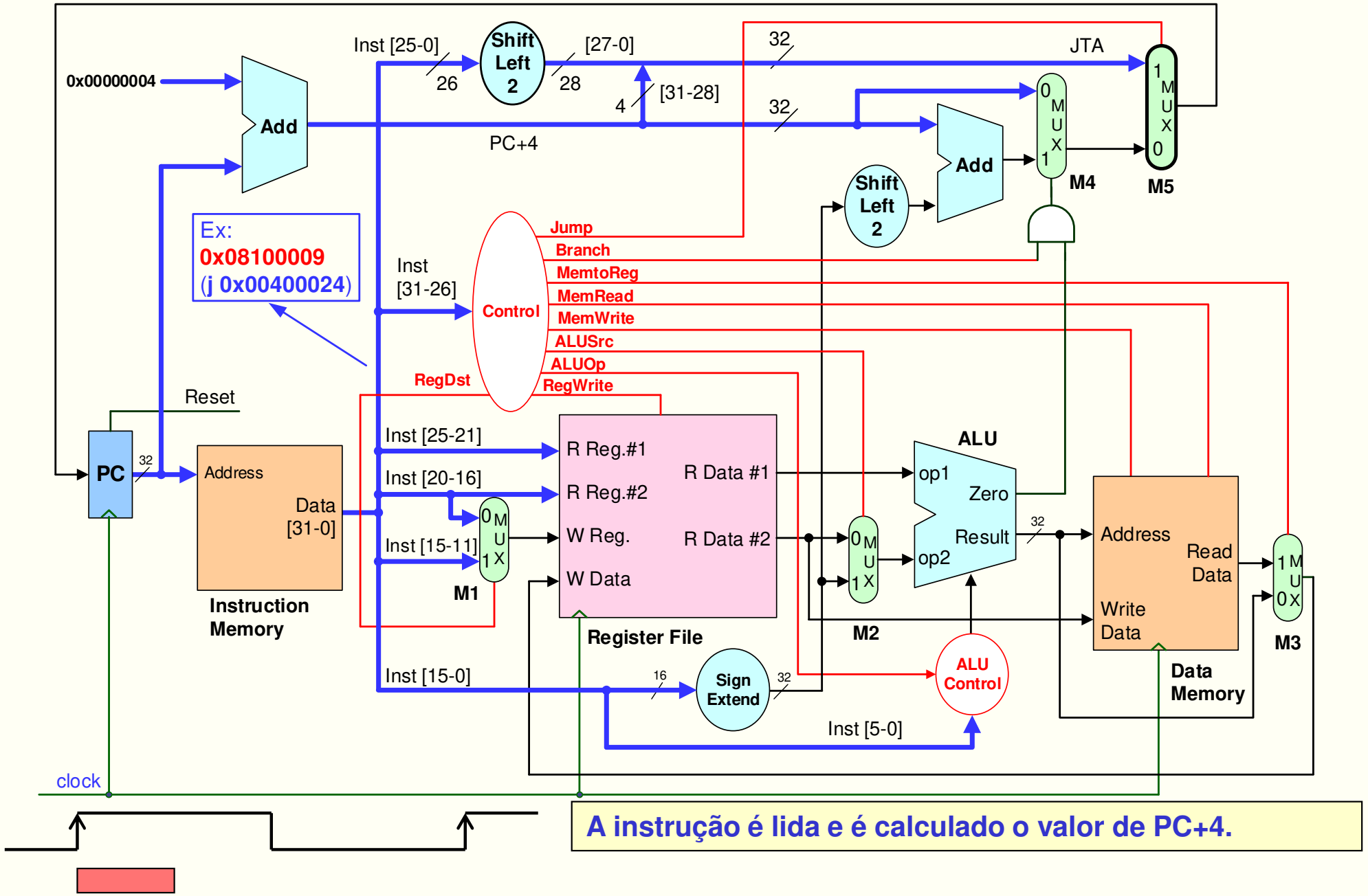


A ALU faz a subtração dos dois valores lidos dos registos. A saída "Zero" é utilizada para decidir qual o próximo valor do PC (Zero=0: PC+4; Zero=1: BTA)

Funcionamento do *datapath* na instrução J

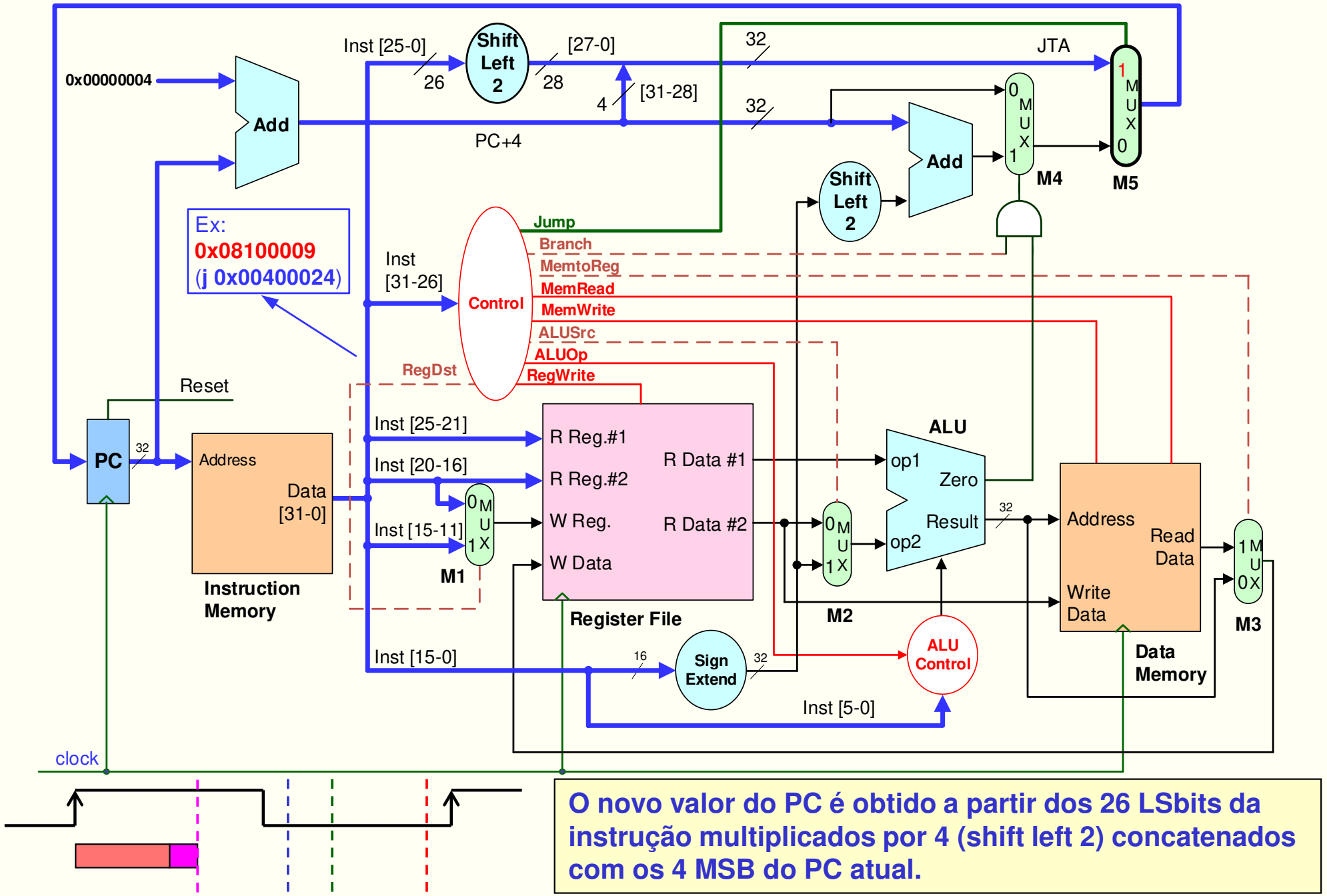
- A instrução é lida e é calculado o valor de PC+4
- São determinados os sinais de controlo. O endereço alvo é obtido a partir dos 26 LSbits da instrução multiplicados por 4 (*shift left 2*) concatenados com os 4 bits mais significativos do PC+4

Funcionamento do *datapath* na instrução J (1)



A instrução é lida e é calculado o valor de PC+4.

Funcionamento do *datapath* na instrução J (2)



O novo valor do PC é obtido a partir dos 26 LSbits da instrução multiplicados por 4 (shift left 2) concatenados com os 4 MSB do PC atual.

Execução de uma instrução no DP *single-cycle* – exemplo

- Vai iniciar-se o *instruction fetch* da instrução apontada pelo Program Counter (PC: **0x00400024**). Nesse instante o conteúdo dos registos do CPU e da memória de dados e instruções é o indicado na figura.
  Qual o conteúdo dos registos após a execução da instrução?

Memória de dados

Endereço	Valor
(…)	(…)
0x10010030	0x63F78395
0x10010034	0xA0FCF3F0
0x10010038	0x147FAF83
(…)	(…)

Memória de instruções

Endereço	Código máquina
(…)	(…)
0x00400020	0x00E82820
0x00400024	0x8CA30024
0x00400028	0x00681824
(…)	(…)

CPU antes

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

→

CPU depois

PC	0x00400028
$3	0xA0FCF3F0
$4	0x15A73C49
$5	0x10010010

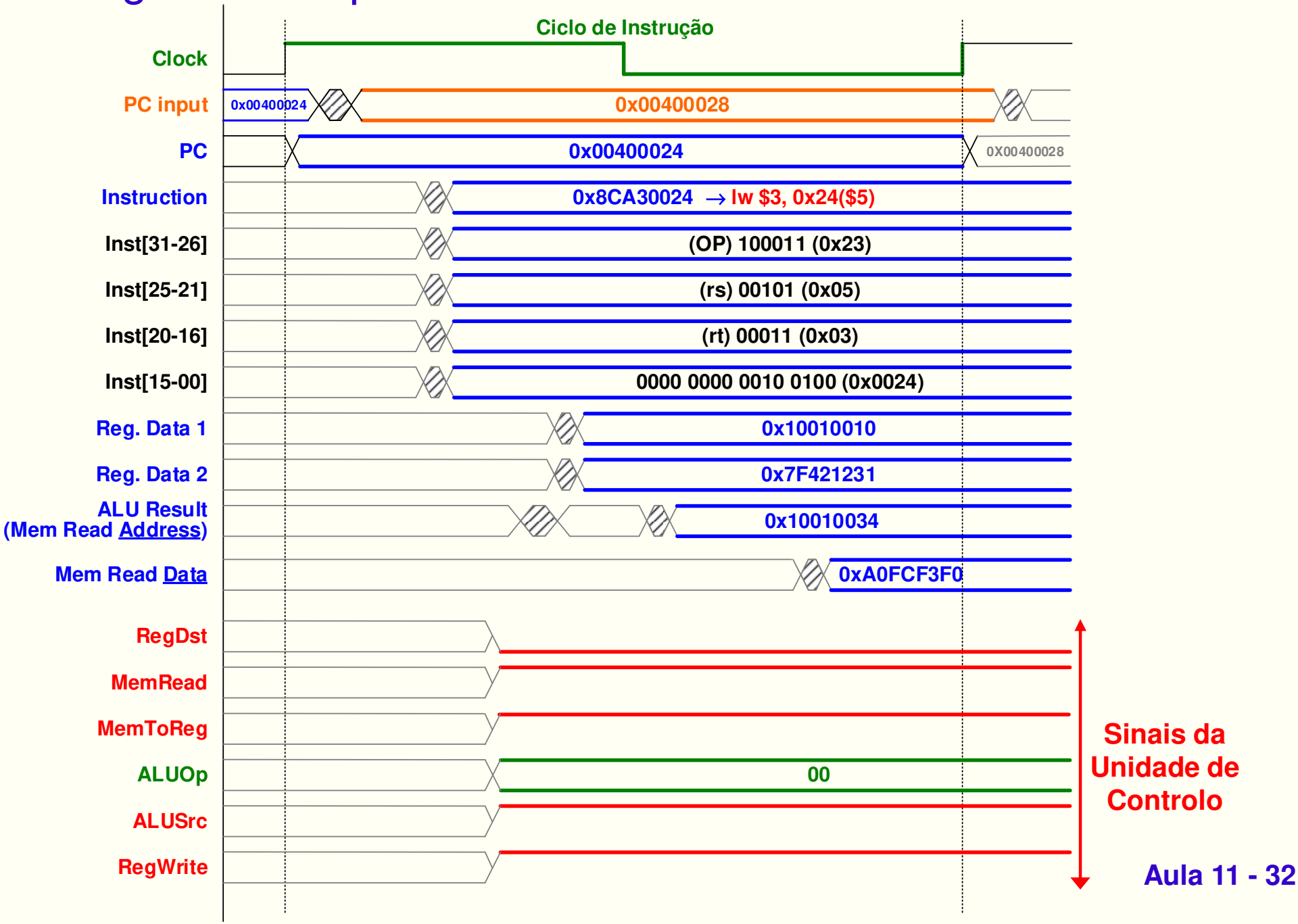
0x8CA30024 → lw $3, 0x24($5)

Mem Addr: 0x10010010 + 0x24 = 0x10010034

10001100101000110000000000100100

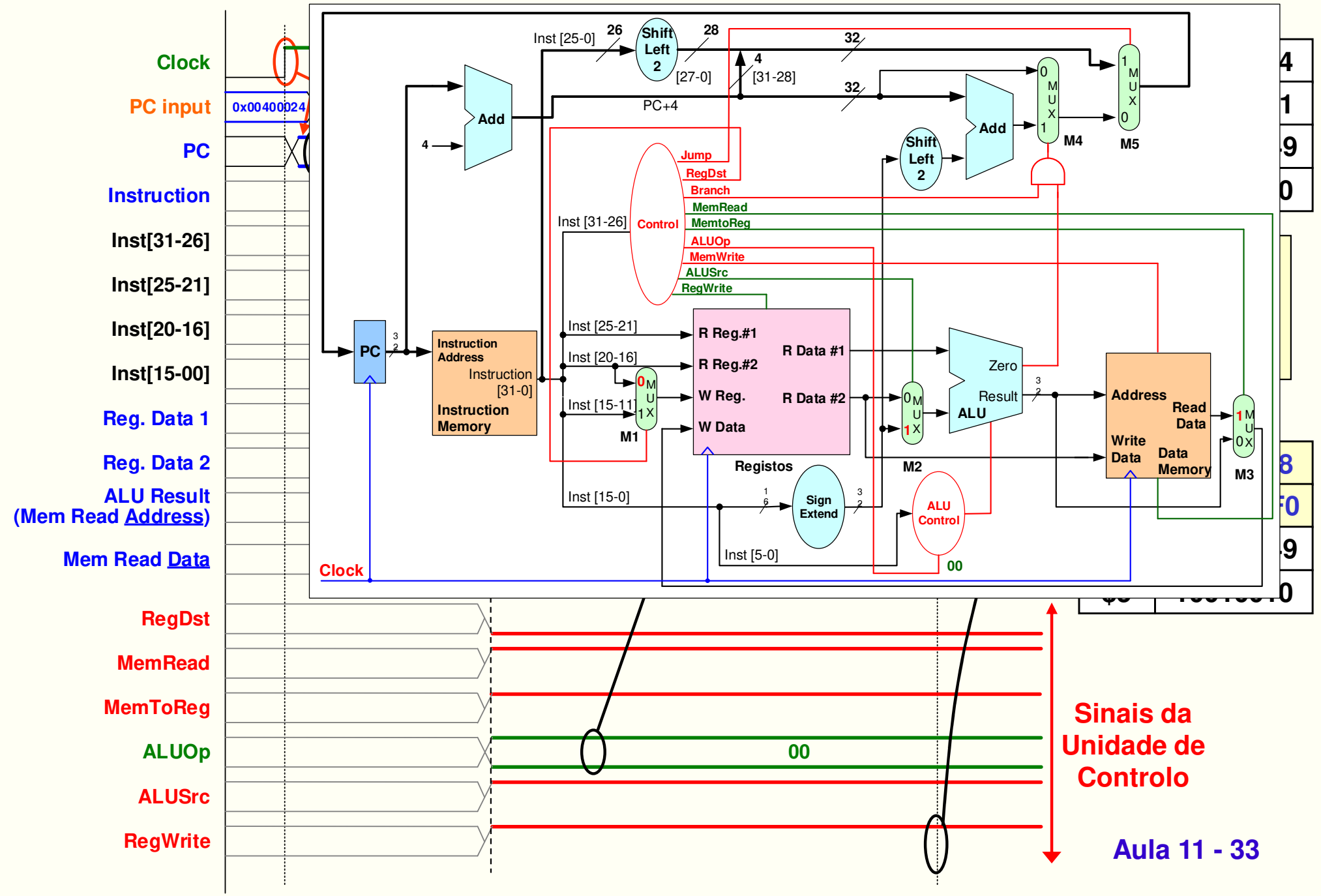
$3 = [0x10010034] = 0xA0FCF3F0

Execução de uma instrução no DP *single-cycle* – diagrama temporal

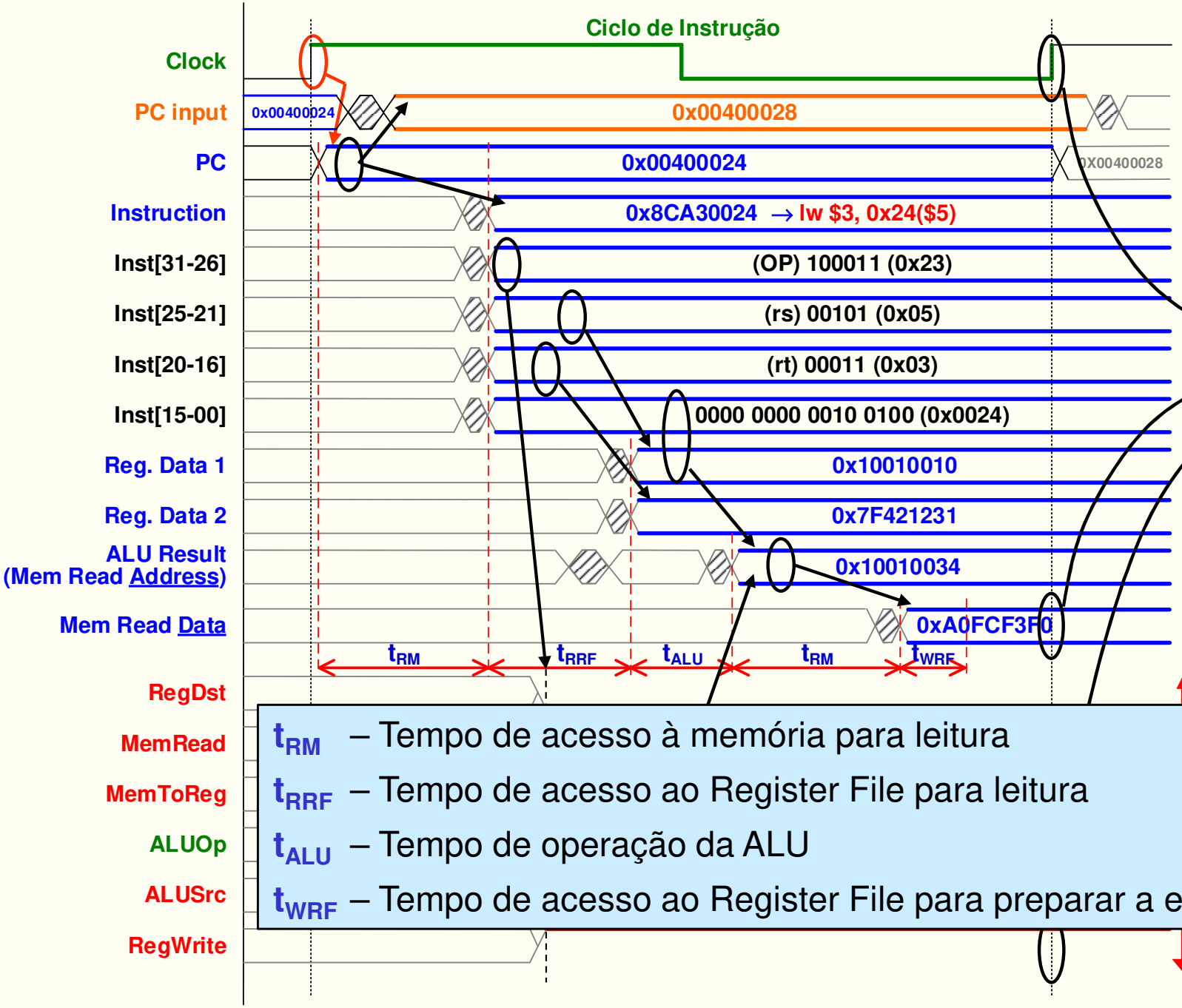


0x00400024	0x8CA30024
0x10010034	0xA0FCF3F0

op	rs	rt	offset
100011	00101	00011	0000000000100100



op	rs	rt	offset
100011	00101	00011	0000000000100100



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

t_{RM} – Tempo de acesso à memória para leitura

t_{RRF} – Tempo de acesso ao Register File para leitura

t_{ALU} – Tempo de operação da ALU

t_{WRF} – Tempo de acesso ao Register File para preparar a escrita

da
e de
olo

Tempo de execução das instruções

- Tempos de atraso introduzidos por cada um dos elementos operativos do *datapath single-cycle*:
  - Acesso à memória para leitura - t_{RM}
  - Acesso à memória para preparar a escrita - t_{WM}
  - Acesso ao *register file* para leitura - t_{RRF}
  - Acesso ao *register file* para preparar a escrita - t_{WRF}
  - Operação da ALU - t_{ALU}
  - Operação de um somador - t_{ADD}
  - Unidade de controlo - t_{CNTL}
  - Extensor de sinal - t_{SE}
  - Shift Left 2 - t_{SL2}
  - Tempo de *setup* do PC - t_{stPC}

Tempo de execução das instruções

- A **frequência máxima** do relógio de sincronização do *datapath single-cycle* está limitada pelo **tempo de execução da instrução "mais longa"**
- O **tempo de execução** de uma instrução é obtido através do **somatório dos atrasos introduzidos por cada um dos elementos operativos envolvidos na sua execução**
- Apenas os elementos operativos que se encontram em **série** contribuem para aumentar o tempo necessário para concluir a execução da instrução (caminho crítico)

Tempo de execução das instruções

- Considerando os tempos de atraso anteriores, os tempos de execução das várias instruções suportadas pelo *datapath single cycle* serão:

Instruções tipo R:

- $t_{EXEC} = t_{RM} + \max(t_{RRF}, t_{CNTL}) + t_{ALU} + t_{WRF}$

Instrução SW:

- $t_{EXEC} = t_{RM} + \max(t_{RRF}, t_{CNTL}) + t_{ALU} + t_{WM}$

Instrução LW:

- $t_{EXEC} = t_{RM} + \max(t_{RRF}, t_{CNTL}) + t_{ALU} + t_{RM} + t_{WRF}$

Instrução BEQ:

- $t_{EXEC} = t_{RM} + \max(\underbrace{\mathbf{\max(t_{RRF} , t_{CNTL}) + t_{ALU}}}_{\text{comparação}}, \underbrace{t_{ADD}}_{\text{cálculo do BTA}}) + t_{stPC}$

Instrução J:

- $t_{EXEC} = t_{RM} + t_{CNTL} + t_{stPC}$

Notas:

1. Considera-se que o tempo de cálculo de PC+4 é muito inferior ao somatório dos restantes tempos envolvidos na execução da instrução
2. O tempo t_{CNTL} inclui o tempo de atraso da unidade de controlo da ALU
3. Desprezam-se os atrasos introduzidos pelos *multiplexers*, *sign extender* e *left shifter*
4. Só se considera o t_{stPC} nas instruções de controlo de fluxo.

Tempo de execução das instruções - exemplo

- Considerem-se os seguintes valores hipotéticos para os tempos de atraso introduzidos por cada um dos elementos operativos do *datapath single-cycle*:
  - Acesso à memória para leitura (t_{RM}): 5ns
  - Acesso à memória para preparar escrita (t_{WM}): 5ns
  - Acesso ao *register file* para leitura (t_{RRF}): 3ns
  - Acesso ao *register file* para preparar escrita (t_{WRF}): 3ns
  - Operação da ALU (t_{ALU}): 4ns
  - Operação de um somador (t_{ADD}): 1ns
  - *Multiplexers* e restantes elementos operativos: 0ns
  - Unidade de controlo (t_{CNTL}): 1ns
  - Tempo de *setup* do PC (t_{stPC}): 1ns

Tempo de execução das instruções - exemplo

- **Instruções tipo R:**
  - $t_{EXEC} = t_{RM} + \max(t_{RFR}, t_{CNTL}) + t_{ALU} + t_{WFR}$
    $= 5 + \max(3, 1) + 4 + 3 =$ **15 ns**
- **Instrução SW:**
  - $t_{EXEC} = t_{RM} + \max(t_{RFR}, t_{CNTL}) + t_{ALU} + t_{WM}$
    $= 5 + \max(3, 1) + 4 + 5 =$ **17 ns**
- **Instrução LW:**
  - $t_{EXEC} = t_{RM} + \max(t_{RFR}, t_{CNTL}) + t_{ALU} + t_{RM} + t_{WFR}$
    $= 5 + \max(3, 1) + 4 + 5 + 3 =$ **20 ns**
- **Instrução BEQ:**
  - $t_{EXEC} = t_{RM} + \max(\max(t_{RFR} , t_{CNTL}) + t_{ALU}, t_{ADD}) + t_{stPC}$
    $= 5 + \max(\max(3, 1) + 4, 1) + 1 =$ **13 ns**
- **Instrução J:**
  - $t_{EXEC} = t_{RM} + t_{CNTL} + t_{stPC} = 5 + 1 =$ **7 ns**

Tmin = 20 ns
fmax = 1 / 20ns = 50MHz

Tempo de execução de um programa

- O tempo de execução de um programa pode ser calculado como:

$$Texec_{CPU} = \#\,Instruções \times CPI \times Clock\_Cycle_{CPU}$$

sendo **CPI o número médio de ciclos de relógio por instrução** na execução do programa em causa; no caso da implementação *single-cycle* o CPI é 1, logo:

$$Texec_{CPU} = \#\,Instruções \times Clock\_Cycle_{CPU}$$

- Define-se ainda:

$$Desempenho_{CPU} = 1/TexecCPU$$

- O desempenho de um CPU ($CPU_{ANALISE}$) relativamente a outro ($CPU_{REFERENCIA}$) pode ser expresso por:

$$\frac{Desempenho_{CPU\_ANALISE}}{Desempenho_{CPU\_REFERENCIA}} = \frac{Texec_{CPU\_REFERENCIA}}{Texec_{CPU\_ANALISE}}$$

Limitações das soluções *single-cycle* - Exemplo

- **Exercício**: calcular o ganho de desempenho que se obteria com uma implementação de *clock* variável relativamente a uma com o *clock* fixo, na execução de um programa com o seguinte *mix* de instruções:
  - 20% de lw, 10% de sw, 50% de tipo R, 15% de branches e 5% de jumps
  - assumindo os tempos execução determinados anteriormente para os vários tipos de instruções (LW: 20ns, SW: 17ns, R-Type: 15ns, BEQ: 13ns, J: 7ns)
- Para este exemplo, o tempo médio de execução de cada instrução num CPU com *clock* variável seria calculado como:

$$T_{MED\_INSTR} = 0,2*20 + 0,1*17 + 0,5*15 + 0,15*13 + 0,05*7 = 15,5ns$$

- O ganho de desempenho do CPU com *clock* variável relativamente a um com *clock* fixo seria então:

$$\frac{Des_{CPU\_CLOCK\_VARIAVEL}}{Des_{CPU\_CLOCK\_FIXO}} = \frac{\#Instruções \times 20}{\#Instruções \times T_{MED\_INSTR}} = 1,29$$

NOTE: A implementação com *clock* variável não é viável mas permite entender o que está a ser sacrificado quando todas as instruções têm que ser executadas num único ciclo de relógio com tempo fixo

Exercícios

- De que tipo é a unidade de controlo principal do *datapath single-cycle*?
- Como calcularia o tempo mínimo necessário para executar cada uma das instruções anteriormente analisadas?
- O que limita a frequência máxima do relógio do *datapath single-cycle*?
- Que alterações é necessário fazer ao *datapath single-cycle* para permitir a execução das instruções:
  - "**`bne`**" – branch not equal
  - "**`jal`**" – jump and link
  - "**`jr`**" – jump register
  - "**`nor`**", "**`xor`**" e "**`sltu`**" (todas tipo R)
- Analise o *datapath* e identifique que instruções deixariam de funcionar corretamente se a unidade de controlo bloqueasse o sinal **`RegWrite`** a '**`1`**'.
- Repita o exercício anterior para cada uma das seguintes situações: **`RegWrite='0'`**, **`MemRead='0'`**, **`MemWrite='0'`**, **`ALUop="00"`**, **`RegDst='1'`**, **`ALUSrc='0'`**, **`MemtoReg='0'`**, **`MemtoReg='1'`**
- Que consequência teria para o funcionamento do *datapth* o bloqueio do sinal **`Branch`** a '**`1`**'?