Aulas 20, 21 & 22

- Limitações das arquitecturas *single cycle*
- Versão de referência de uma arquitectura *multicycle*
- Exemplos de funcionamento numa arquitectura *multicycle:*
  - Instruções tipo R
  - Acesso à memória – LW
  - Salto condicional – BEQ
  - Salto incondicional – J
- Unidade de controlo para o *datapath multicycle*
  - Diagrama de estados da unidade de controlo
- Sinais de controlo e valores do *datapath multicycle*
  - Exemplo com execução sequencial de três instruções

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

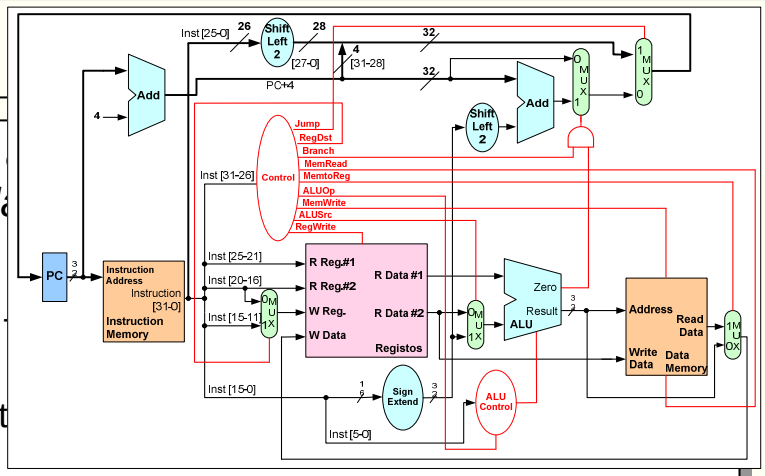
Limitações das soluções *single-cycle*

- Como discutimos já anteriormente, a **frequência máxima** do relógio de sincronização está limitada pelo **tempo de execução da instrução mais longa**
- Os **tempos de execução** das várias instruções suportadas pelo *datapath single cycle* corresponderão assim ao **somatório dos atrasos introduzidos por cada um dos elementos funcionais envolvidos na execução da instrução**

Note-se que apenas os elementos funcionais que se encontram em **série** contribuem para aumentar o tempo necessário para concluir a execução da instrução (caminho crítico).

Consideremos os seguintes tempos de atraso introduzidos por cada um dos elementos funcionais do *datapath single cycle*:

- Acesso à memória para leitura - t_{RM}
- Acesso à memória para preparar a escrita - t_{WM}
- Acesso ao *file register* para leitura - t_{RFR}
- Acesso ao *file register* para preparar a escrita - t_{WFR}
- 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}



Considerando os tempos de atraso
várias instruções suportadas pelo *da*

- **Instruções tipo R:**
  - $t_{EXEC} = t_{RM} + \max(t_{RFR}, t_{CNTL}) + t_{ALU}$
- **Instrução SW:**
  - $t_{EXEC} = t_{RM} + \max(t_{RFR}, t_{CNTL}, t_{SE}) + t$
- **Instrução LW:**
  - $t_{EXEC} = t_{RM} + \max(t_{RFR}, t_{CNTL}, t_{SE}) + t_{ALU} + t_{RM} + t_{WFR}$
- **Instrução BEQ:**
  - $t_{EXEC} = t_{RM} + \max(\underbrace{\max(t_{RFR}, t_{CNTL}) + t_{ALU}}_{\text{comparação}}, \underbrace{t_{SE} + t_{SL2} + t_{ADD}}_{\text{cálculo do BTA}}) + t_{stPC}$
- **Instrução J:**
  - $t_{EXEC} = t_{RM} + \max(t_{CNTL}, t_{SL2}) + t_{stPC}$

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 tempos de atraso introduzidos pelos *multiplexers*
4. Só se considera o t_{stPC} nas instrução de controlo de fluxo.

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

- **Instruções tipo R:**
  - $t_{EXEC} = t_{RM} + \max(t_{RFR}, t_{CNTL}) + t_{ALU} + t_{WFR}$
- **Instrução SW:**
  - $t_{EXEC} = t_{RM} + \max(t_{RFR}, t_{CNTL}, t_{SE}) + t_{ALU} + t_{WM}$
- **Instrução LW:**
  - $t_{EXEC} = t_{RM} + \max(t_{RFR}, t_{CNTL}, t_{SE}) + t_{ALU} + t_{RM} + t_{WFR}$
- **Instrução BEQ:**
  - $t_{EXEC} = t_{RM} + \max(\underbrace{\max(t_{RFR}, t_{CNTL}) + t_{ALU}}_{\text{comparação}}, \underbrace{t_{SE} + t_{SL2} + t_{ADD}}_{\text{cálculo do BTA}}) + t_{stPC}$
- **Instrução J:**
  - $t_{EXEC} = t_{RM} + \max(t_{CNTL}, t_{SL2}) + t_{stPC}$

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 tempos de atraso introduzidos pelos *multiplexers*
4. Só se considera o t_{stPC} nas instrução de controlo de fluxo.

Limitações das soluções *single-cycle*

Consideremos os seguintes valores hipotéticos para os tempos de atraso introduzidos por cada um dos elementos funcionais do *datapath single cycle*:

- Acesso à memória para leitura (t_{RM}): 5ns
- Acesso à memória para escrita (t_{WM}): 5ns
- Acesso ao *file register* para leitura (t_{RFR}): 3ns
- Acesso ao *file register* para escrita (t_{WFR}): 3ns
- Operação da ALU (t_{ALU}): 4ns
- Operação de um somador (t_{ADD}): 1ns
- *Multiplexers* e restantes elementos funcionais: 0ns
- Unidade de controlo (t_{CNTL}): 1ns
- Tempo de setup do PC (t_{stPC}): 1ns

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

- **Instruções tipo R:**
  - $t_{EXEC} = t_{RM} + \max(t_{RFR}, t_{CNTL}) + t_{ALU} + t_{WFR}$
    $= 5 + \max(3, 1) + 4 + 3 =$ **15 ns**
- **Instrução SW:**
  - $t_{EXEC} = t_{RM} + \max(t_{RFR}, t_{CNTL}, t_{SE}) + t_{ALU} + t_{WM}$
    $= 5 + \max(3, 1, 0) + 4 + 5 =$ **17 ns**
- **Instrução LW:**
  - $t_{EXEC} = t_{RM} + \max(t_{RFR}, t_{CNTL}, t_{SE}) + t_{ALU} + t_{RM} + t_{WFR}$
    $= 5 + \max(3, 1, 0) + 4 + 5 + 3 =$ **20 ns**
- **Instrução BEQ:**
  - $t_{EXEC} = t_{RM} + \max(\max(t_{RFR} , t_{CNTL}) + t_{ALU}, t_{SE} + t_{SL2} + t_{ADD}) + t_{stPC}$
    $= 5 + \max(\max(3, 1) + 4, 0 + 0 + 1) + 1 =$ **13 ns**
- **Instrução J:**
  - $t_{EXEC} = t_{RM} + \max(t_{CNTL}, t_{SL2}) + t_{stPC} = 5 + \max(1, 0) + 1 =$ **7 ns**

Limitações das soluções *single-cycle*

- Face à análise anterior, a máxima frequência de trabalho seria:

  $f_{max} = 1 / 20ns =$ **50MHz**

- Com a mesma tecnologia, contudo, uma multiplicação ou divisão poderia demorar um tempo da ordem dos 150ns
- Para poder suportar uma ALU com capacidade para efectuar operações de multiplicação/divisão, a frequência de relógio máxima do nosso *datapath* baixaria para **6.66Mhz**
- Esta frequência máxima limitaria a eficiência de todas as outras instruções, mesmo que as instruções de multiplicação ou divisão sejam raramente utilizadas
- Uma solução possível, mas tecnicamente complicada, seria usar um relógio de frequência variável, ajustável em função da instrução que vai ser executada

Limitações das soluções *single-cycle*

- **Exemplo** – Assumindo os tempos execução determinados anteriormente para os vários tipos de instruções, calcular o factor de melhoria 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

Tempos de execução:
- LW: 20 ns
- SW: 17 ns
- Tipo R: 15 ns
- BEQ: 13 ns
- J: 7 ns

$$\frac{Desempenho_{CPU\_CLOCK\_VARIAVEL}}{Desempenho_{CPU\_CLOCK\_FIXO}} = \frac{Texec_{CPU\_CLOCK\_FIXO}}{Texec_{CPU\_CLOCK\_VARIAVEL}}$$

Numa implementação *single cycle* o CPI é 1, logo

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

$$\frac{Desempenho_{CPU\_CLOCK\_VARIAVEL}}{Desempenho_{CPU\_CLOCK\_FIXO}} = \frac{\#Instruções \times 20}{\#Instruções \times (0{,}2 \times 20 + 0{,}1 \times 17 + 0{,}5 \times 15 + 0{,}15 \times 13 + 0{,}05 \times 7)} = 1{,}29$$

A implementação com clock variável, como referido, não é simples do ponto de vista prático, mas permite-nos entender o que está a ser sacrificado quando todas as instruções têm que ser executadas num único ciclo de relógio com dimensão fixa

Limitações das soluções *single-cycle*

As conclusões a tirar serão portanto:

- Num *datapath* que suporte instruções com complexidade variável, é a **instrução mais lenta que determina a máxima frequência de trabalho**, mesmo que seja um instrução pouco frequente
- Uma vez que o ciclo de relógio é igual ao maior tempo de atraso de todas as instruções, não é útil usar técnicas que reduzam o atraso do caso mais comum mas que não melhorem o maior tempo de atraso (isto é, o atraso do caminho crítico)
  - Isto contraria um dos princípios-chave de desenho: *make the common case fast* (o que é mais comum deve ser mais rápido)
- Elementos funcionais que estejam envolvidos na execução de uma mesma instrução **não podem ser usados para mais do que uma operação por ciclo de relógio** (ex: memória de instruções e de dados, ALU e somadores, ...)

Alternativa às soluções *single-cycle*

- Em vez de desenvolver uma estratégia baseada num relógio de frequência variável, é preferível abdicar do princípio de que todas as instruções devem ser executadas num único ciclo de relógio
- Em alternativa, as várias instruções que compõem o *set* de instruções podem ser executadas em vários ciclos de relógio (*multicycle)*:
  - A execução da instrução é decomposta num conjunto de operações
  - Cada uma dessas operações faz uso de um elemento funcional fundamental: memória, *file register* ou ALU
  - **Em cada ciclo de relógio poderá ser realizada uma ou mais operações, desde que sejam independentes** (por exemplo, *instruction fetch* e cálculo de PC+4 ou *operand fetch* e cálculo do BTA)
- Desta forma, o período de relógio fica apenas limitado pelo maior dos tempos de atraso de cada um dos elementos funcionais fundamentais
- Para os tempos de atraso que considerámos anteriormente, a máxima frequência de relógio seria assim: **fmax = 1 / t_{RM} = 1 / 5ns = 200MHz**

Alternativa às soluções *single-cycle*

- Uma outra vantagem duma solução de execução em vários ciclos de relógio (*multicycle)* é que um **mesmo elemento funcional** pode ser utilizado mais do que uma vez, no contexto da execução duma mesma instrução, desde que em ciclos de relógio distintos:
  - A memória externa poderá ser partilhada por instruções e dados
  - A mesma ALU poderá ser usada, para além das operações que já realizava na implementação *single cycle*, para:
    - Calcular o valor de PC+4
    - Calcular o endereço alvo das instruções de salto condicional (BTA)
- A versão ***multicycle*** passará assim a ter:
  - **Uma única memória** para programa e dados (arquitectura *Von Neumann*)
  - **Uma única ALU**, em vez de uma ALU e dois somadores

O *Datapath Multicycle*

- A arquitectura *multicycle* do MIPS que vamos analisar adopta um ciclo de instrução composto por um **máximo de cinco passos distintos**, cada um deles executado em 1 ciclo de relógio
- A distribuição das operações por estes 5 passos tenta distribuir equitativamente o trabalho a realizar em cada ciclo
- Estes passos reflectem o pressuposto de que durante um ciclo de relógio apenas seja possível efectuar uma, de cada uma, das seguintes operações:
  - Acesso à memória externa (uma escrita ou uma leitura)
  - Acesso ao *file register* (uma escrita ou uma leitura)
  - Operação na ALU
- Isto permite que, **no mesmo ciclo de relógio**, possam ser realizados, por exemplo, **um acesso à memória externa e uma operação na ALU**, ou um **acesso ao *File Register* e uma operação na ALU**

O *Datapath Multicycle*

Fases de execução das instruções no *datapath multicycle*:

Fase 1 (memória, ALU):
- *Instruction fetch* e cálculo de PC+4

Fase 2 (unidade de controlo, *file register*, ALU):
- *Instruction decode*, *operand fetch* e cálculo do *branch target address*

Fase 3 (ALU):
- Execução da operação na ALU (instruções tipo R / addi / slti), **ou**
- Cálculo do endereço de memória (instruções de acesso à memória), **ou**
- Comparação dos operandos - instrução *branch* (conclusão da instrução)

Fase 4 (memória):
- Acesso à memória para leitura (instrução LW), **ou**
- Acesso à memória para escrita (conclusão da instrução SW), **ou**
- Escrita no *File Register* (conclusão das instruções tipo R / addi / slti: ***write-back***)

Fase 5 *(file register):*
- Escrita no *File Register* (conclusão da instrução LW: ***write-back***)

O *Datapath Multicycle*

Execução instruções tipo R / addi / slti, ou

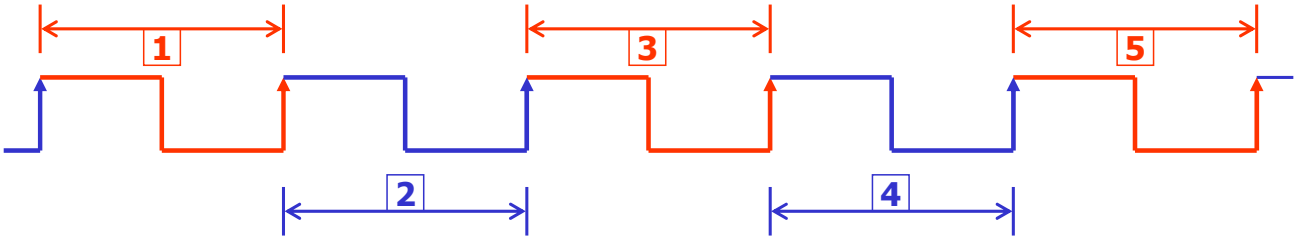
Cálculo do endereço de acesso à memória, ou

Conclusão do *branch*

***Instruction fetch*, e**

Cálculo de PC + 4

***Write-back* (LW)**



***Instruction decode*, e**

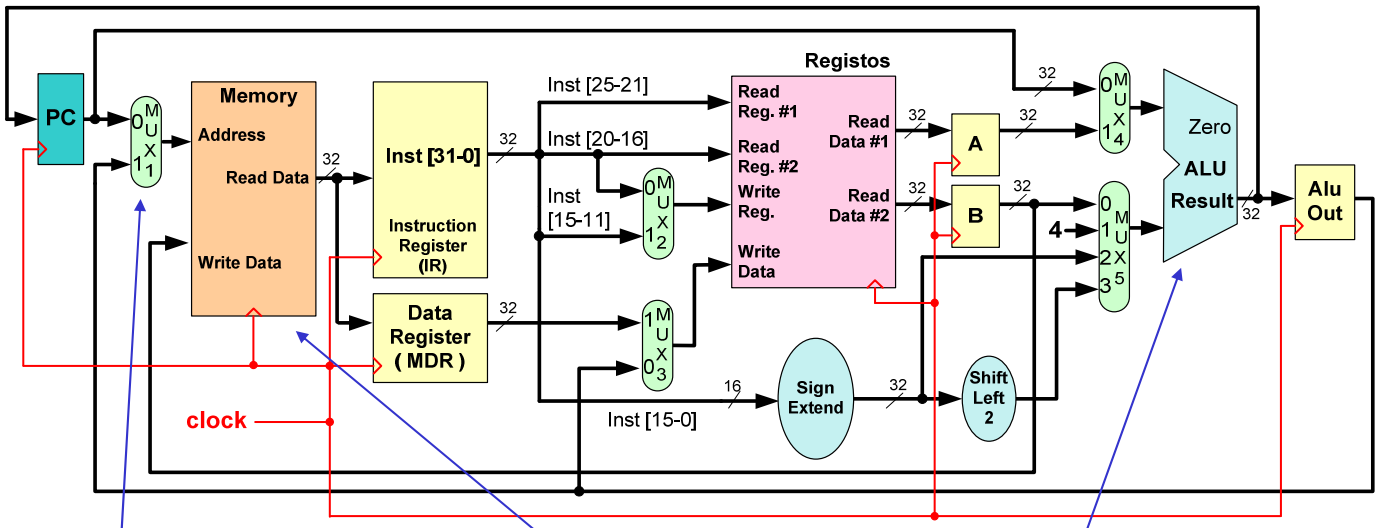
***Operand fetch*, e**

Cálculo do endereço alvo das instruções de *branch* (BTA)

***Write-back* - instruções tipo R / addi / slti, ou**

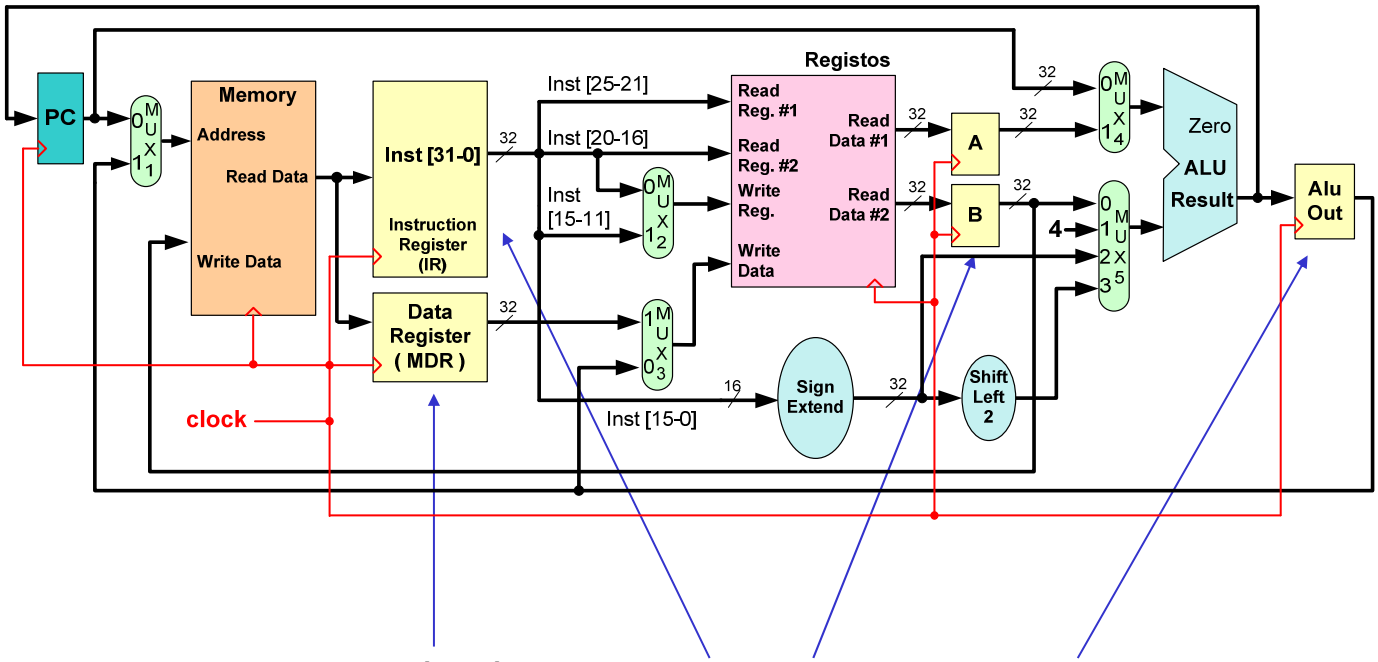
Acesso à memória para leitura ou escrita (conclusão do SW)

O *Datapath Multicycle* (sem as instruções de BEQ e J)



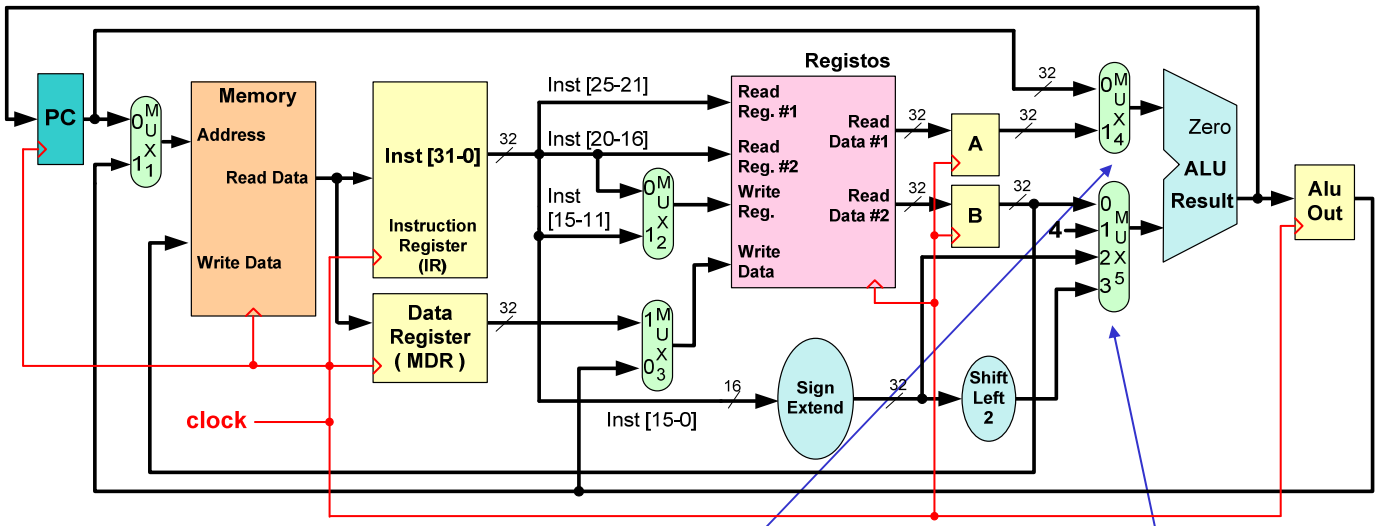
- **Uma única memória para programa e dados**
  - Um multiplexer no barramento de endereços da memória permite seleccionar qual o endereço a usar:
    - o conteúdo do PC (para leitura da instrução) ou
    - o valor calculado na ALU (para acesso leitura/escrita de dados com as instruções LW/SW)
- **Uma única ALU** (em vez de uma ALU e dois somadores)

O *Datapath Multicycle* (sem as instruções de BEQ e J)



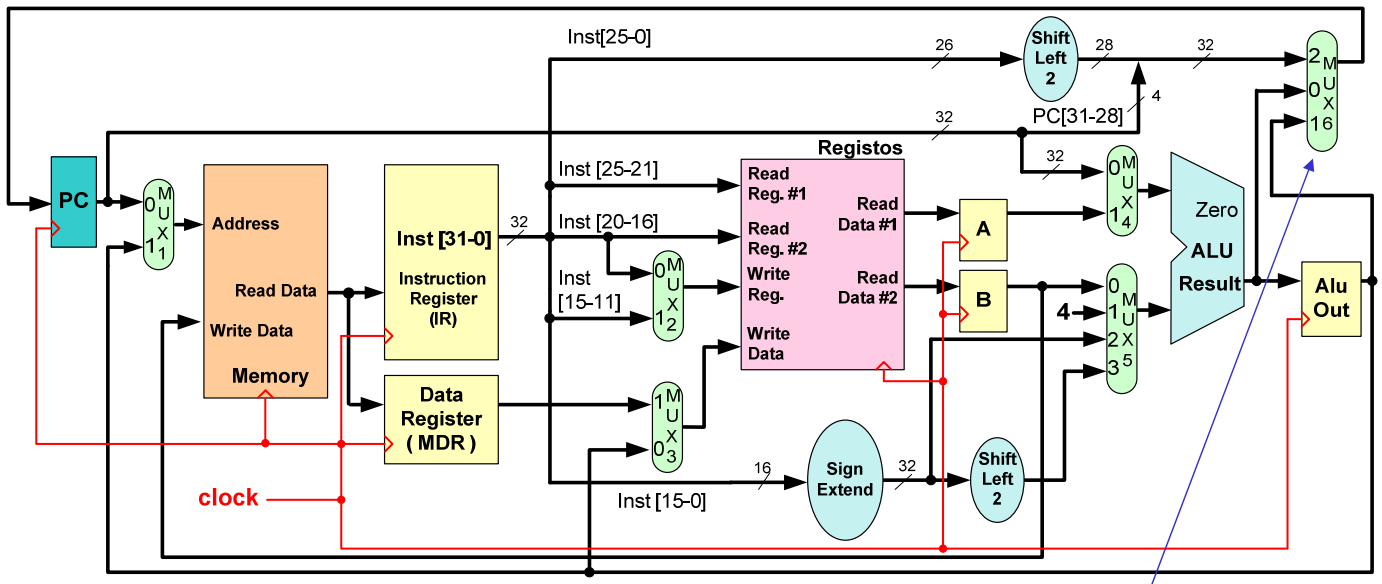
- Registos adicionados à saída dos elementos funcionais fundamentais para armazenamento de informação obtida/calculada durante o ciclo de relógio corrente e que será utilizada no ciclo de relógio seguinte

O *Datapath Multicycle* (sem as instruções de BEQ e J)



- A utilização de uma única ALU obriga às seguintes alterações nas suas entradas:
  - Um *multiplexer* adicional na primeira entrada, que escolhe entre a saída do registo A e a saída do registo PC
  - O *multiplexer* da segunda entrada é aumentado para poder suportar o incremento do PC (constante 4) e o cálculo do endereço alvo das instruções de *branch* (*branch target address*)

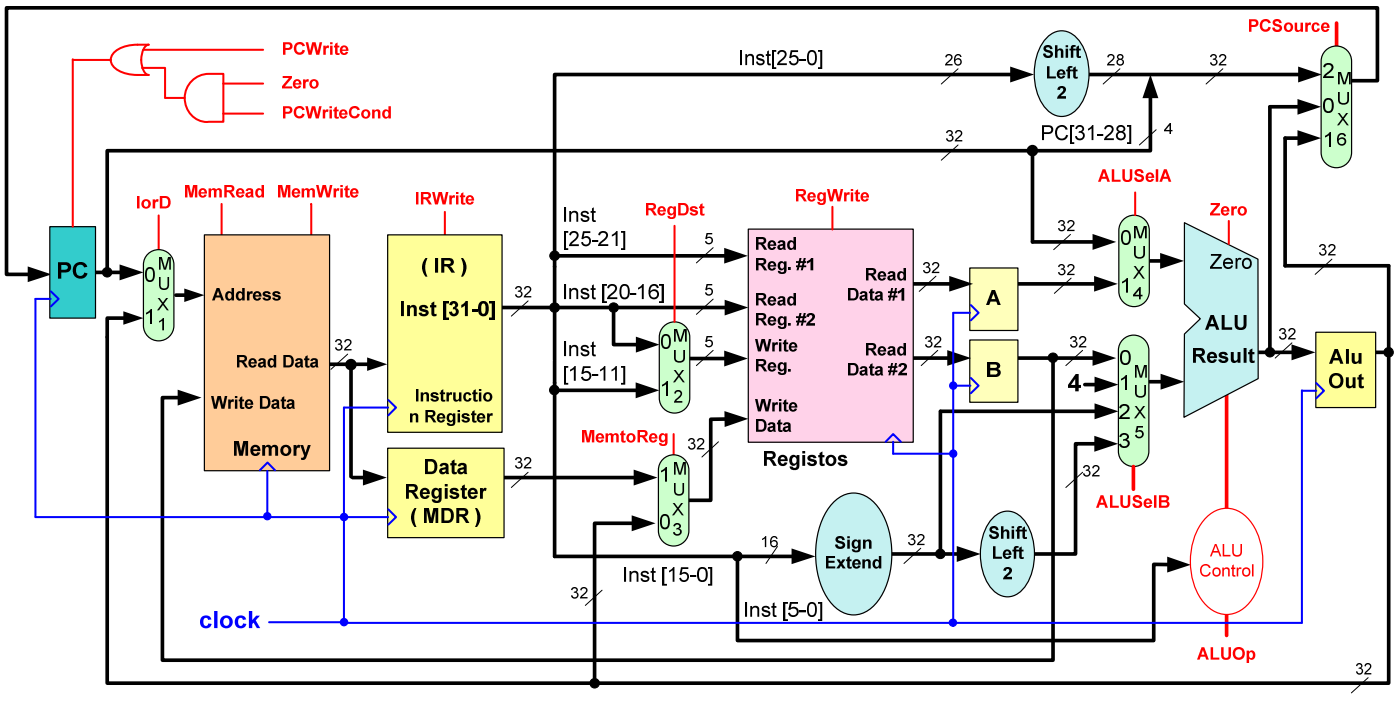
O *Datapath Multicycle* (com as instruções de salto)



Com as instruções de salto, o **registo PC** pode ser actualizado com um dos valores:

- A **saída da ALU** que contém o PC+4 calculado durante o *instruction fetch* (na 1ª fase)
- A saída do registo **ALUOut** que armazena o endereço alvo das instruções de *branch* (BTA) calculado na ALU (na 2ª fase)
- ***Jump Target Address*** - 26 LSB da instrução multiplicados por 4 (*shift left* 2) concatenados com os 4 MSB do PC actual (o PC foi já incrementado na 1ª fase)

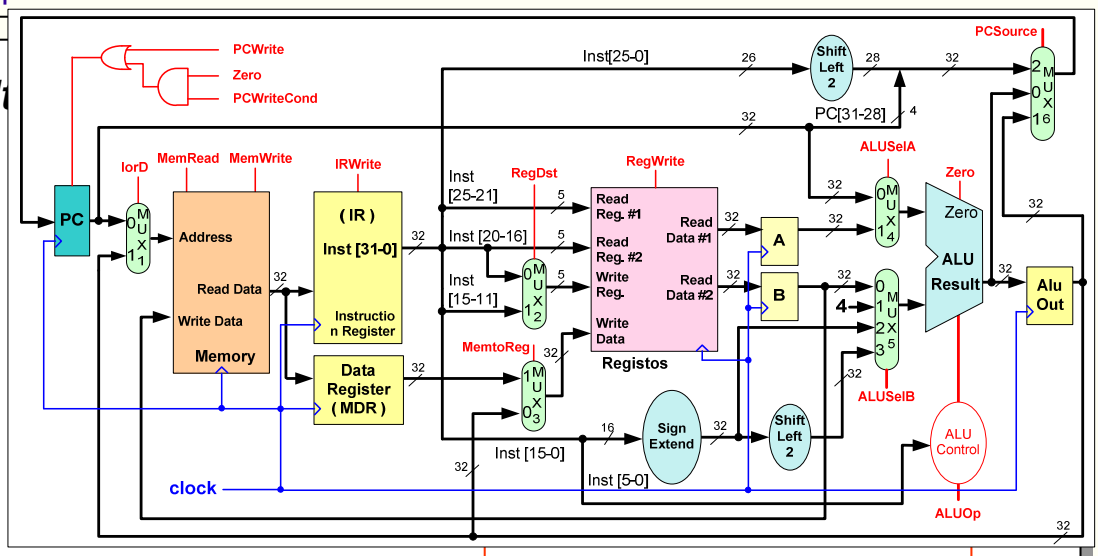
O *Datapath Multicycle*, com os sinais de controlo



Sinal	Efeito quando não activo	Efeito quando activo
MemRead	Nenhum	O conteúdo da memória no endereço indicado é apresentado à saída
MemWrite	Nenhum	O conteúdo do registo de memória cujo endereço é fornecido é substituido pelo valor apresentado à entrada
ALUSelA	O primeiro operando da ALU é o PC	O primeiro operando da ALU provém do registo indicado no campo rs
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 do Data Register
IorD	O PC é usado para fornecer o endereço à memória externa	A saída do registo AluOut é usada para providenciar um endereço para a memória externa
IRWrite	Nenhum	O valor lido da memória externa é escrito no Instruction Register
PCWrite	Nenhum	O PC é actualizado incondicionalmente na próxima transição activa do sinal de relógio
PCWriteCond	Nenhum	O PC é actualizado condicionalmente na próxima transição activa do relógio

Sinal	Valor	Efeito
ALUSelB	00	A segunda entrada da ALU provém do registo indicado pelo campo rt
	01	A segunda entrada da ALU é a constante 4
	10	A segunda entrada da ALU é a versão de sinal extendido dos 16 bits menos significativos do IR
	11	A segunda entrada da ALU é a versão de sinal extendido e deslocada de dois bits, dos 16 bits menos significativos do IR
ALUOp	00	ALU efectua uma adição
	01	ALU efectua uma subtracção
	10	O campo "function code" da instrução determina qual a operação da ALU.
	11	ALU efectua um SLT
PCSource	00	O valor do PC é actualizado com o resultado da ALU (IF)
	01	O valor do PC é actualizado com o resultado da AluOut (Branch)
	10	O valor do PC é actualizado com o valor target do Jump
	11	Não usado

O *Datapath Mult*



Início da execução da instrução

IR = Memory[PC]

PC = PC + 4

1

2

A = Reg[IR[25:21]]

B = Reg[IR[20:16]]

ALUOut = PC + (sign extend(IR[15:0]) << 2)

O *Datapath Multicycle* (acções realizadas nas transições activas do relógio (0→1)

Início da execução da instrução

IR = Memory[PC]

PC = PC + 4

1

ALUOut = A op B, ou

ALUOut = A + (sign extend(IR[15:0])), ou

PC = ALUOut (se A = B), ou

PC = PC[31:28]+(IR[25:0]<< 2)

3

Reg[IR[20:16]] = MDR

5

2

A = Reg[IR[25:21]]

B = Reg[IR[20:16]]

ALUOut = PC + (sign extend(IR[15:0]) << 2)

4

Reg[IR[15:11]] = ALUOut, ou

Reg[IR[20:16]] = ALUOut, ou

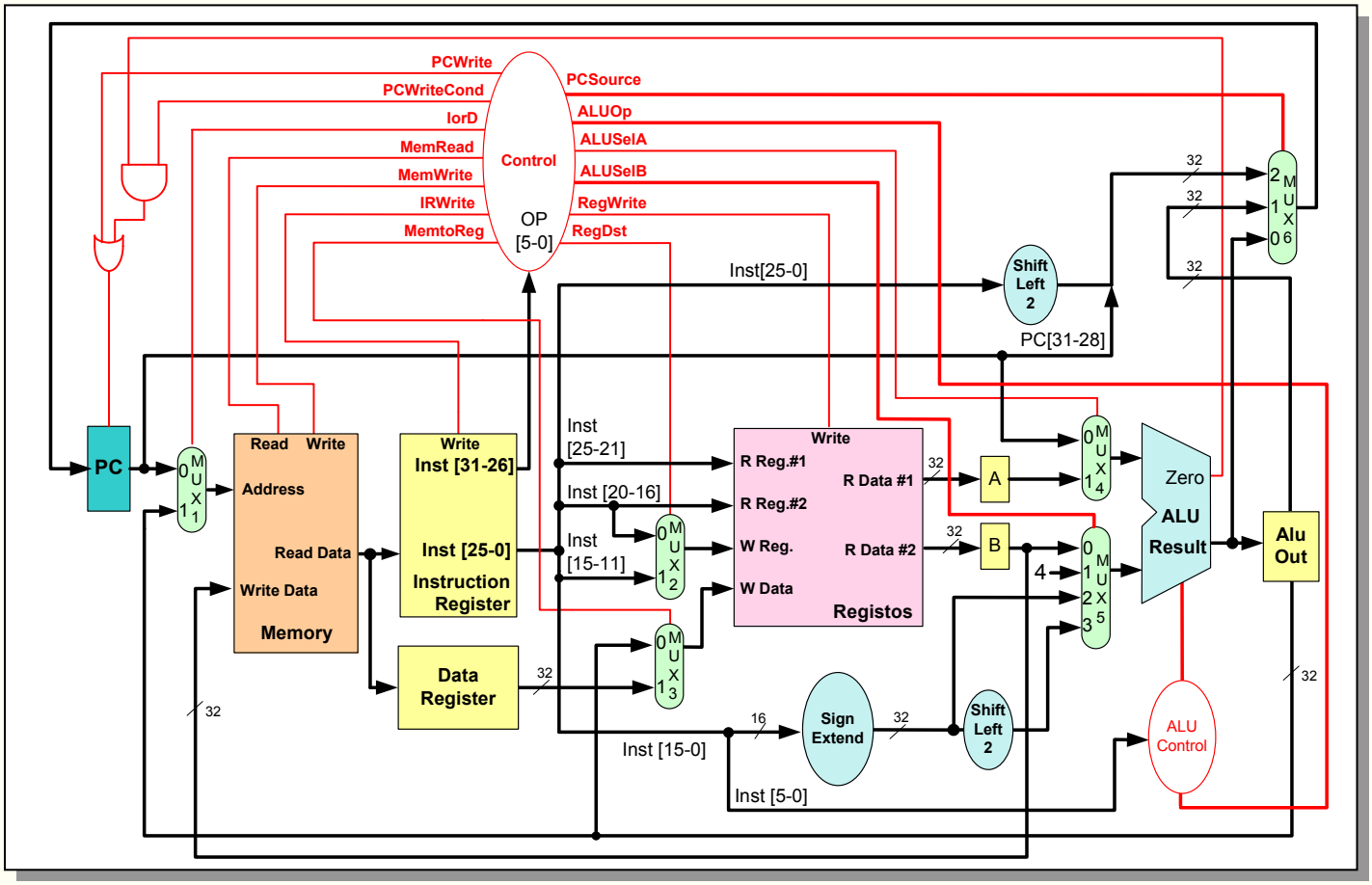
MDR = Memory[ALUOut], ou

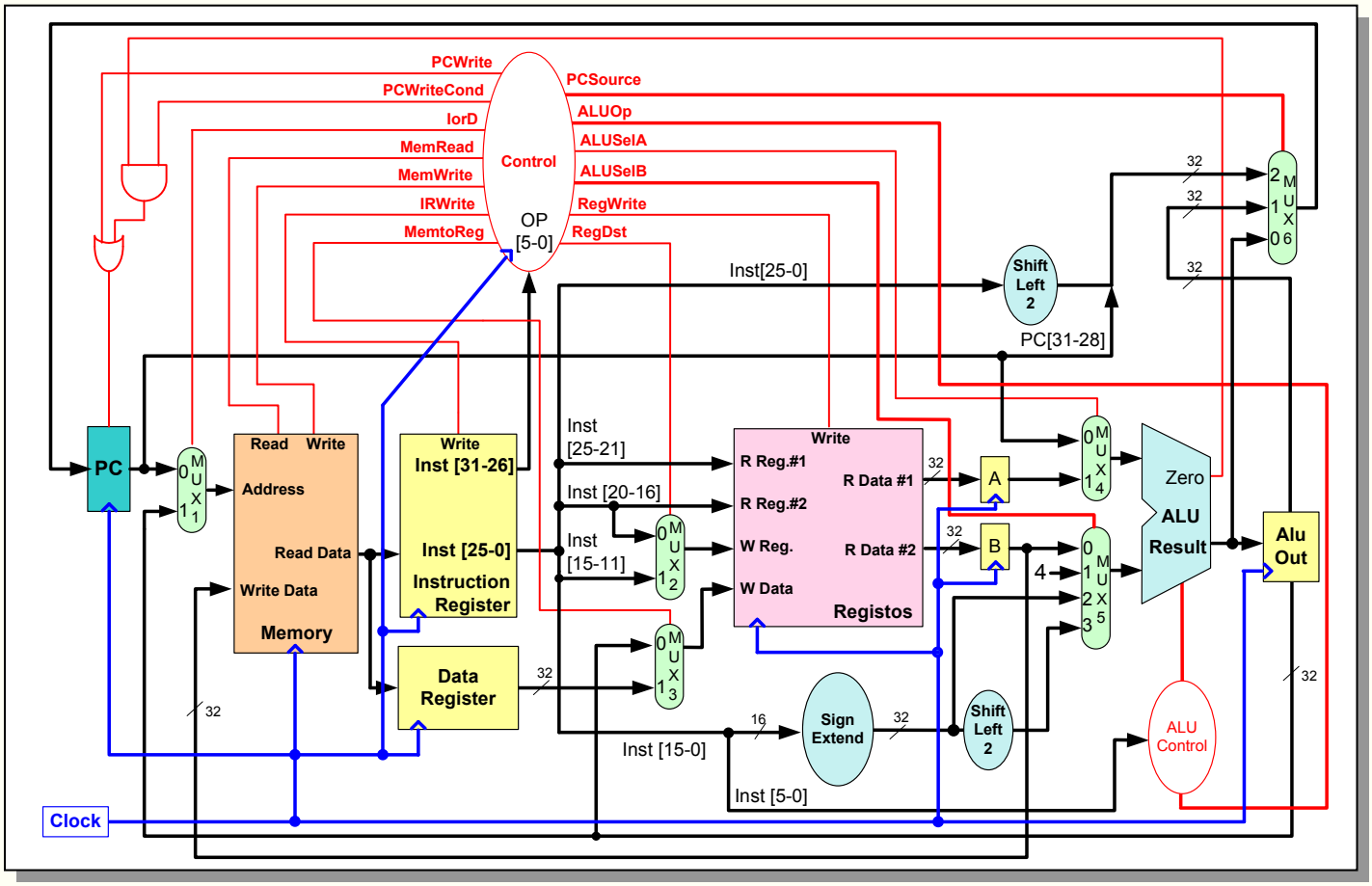
Memory[ALUOut] = B

O *Datapath Multicycle*

As operações realizadas no final (transição activa do relógio) de cada um dos cinco passos:

Passo	Acção p/ as R-Type / ADDI / SLTI	Acção p/ instruções que referenciam a memória	Acção p/ os branches
Instruction fetch	IR = Memory[PC] PC = PC + 4		
Instruction decode, register fetch, cálculo do BTA	A = Reg[IR[25:21]] B = Reg[IR[20:16]] ALUOut = PC + (sign extended(IR[15:0]) << 2)		
Execução (tipoR/addi/slti), cálculo de endereços ou conclusão dos branches	ALUOut = A op B ou ALUOut = A op extend(IR[15:0])	ALUOut = A+sign-extended(IR[15:0]))	If (A == B) then PC = ALUOut
Acesso à memória (leitura-LW; ou escrita-SW) ou escrita no File Register (write-back, instruções tipo R/addi/slti)	Tipo R: Reg[IR[15:11]]= ALUOut ADDI / SLTI: Reg[IR[20:16]]= ALUOut	MDR = Memory[ALUOut] ou Memory[ALUOut] = B	
Escrita no File Register (write-back, instrução LW)		Reg[IR[20:16]] = MDR	





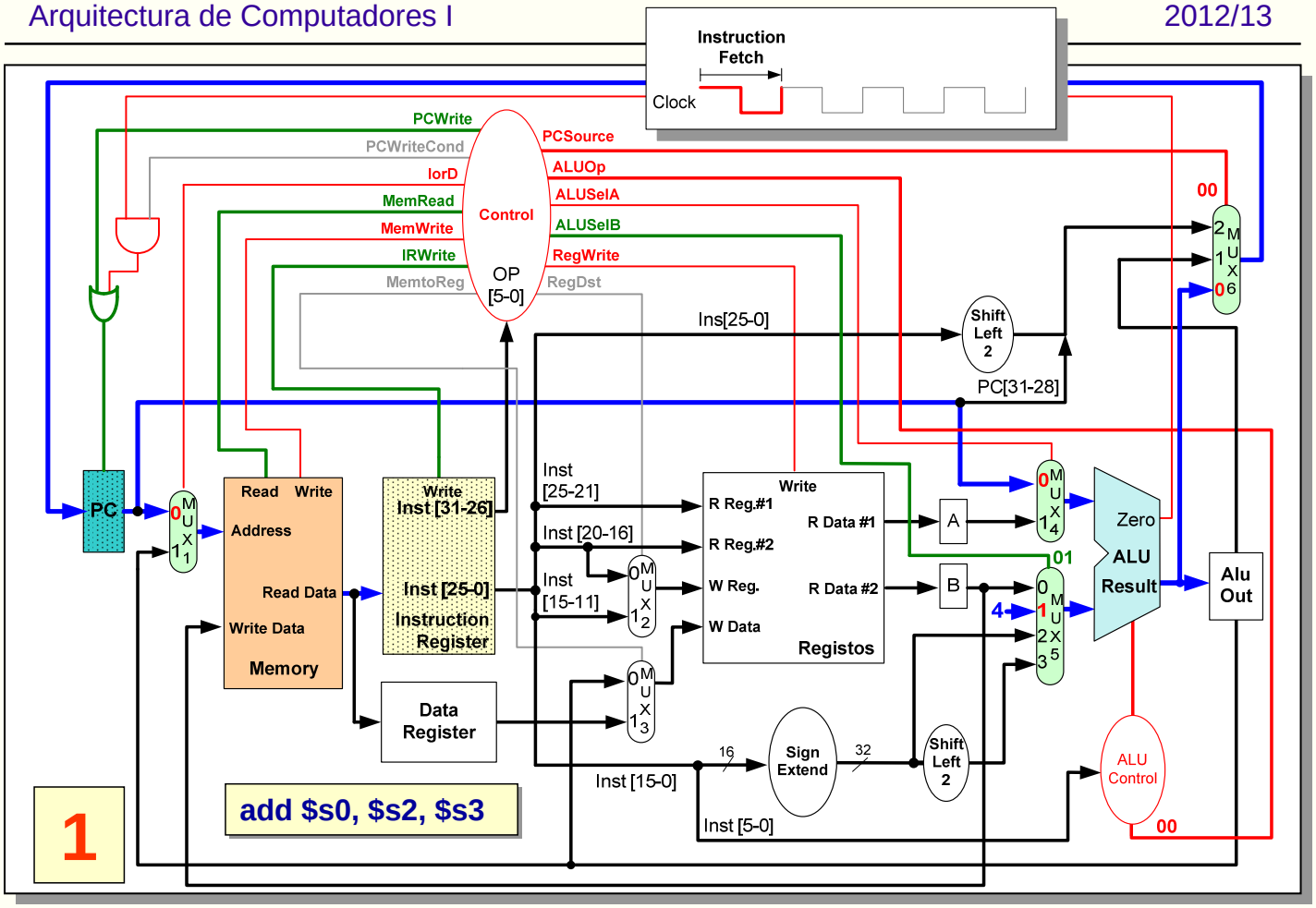
Nos exemplos que se seguem, as cores indicam o estado, o valor ou a utilização dos sinais de controlo, barramentos e elementos de estado/combinatórios.

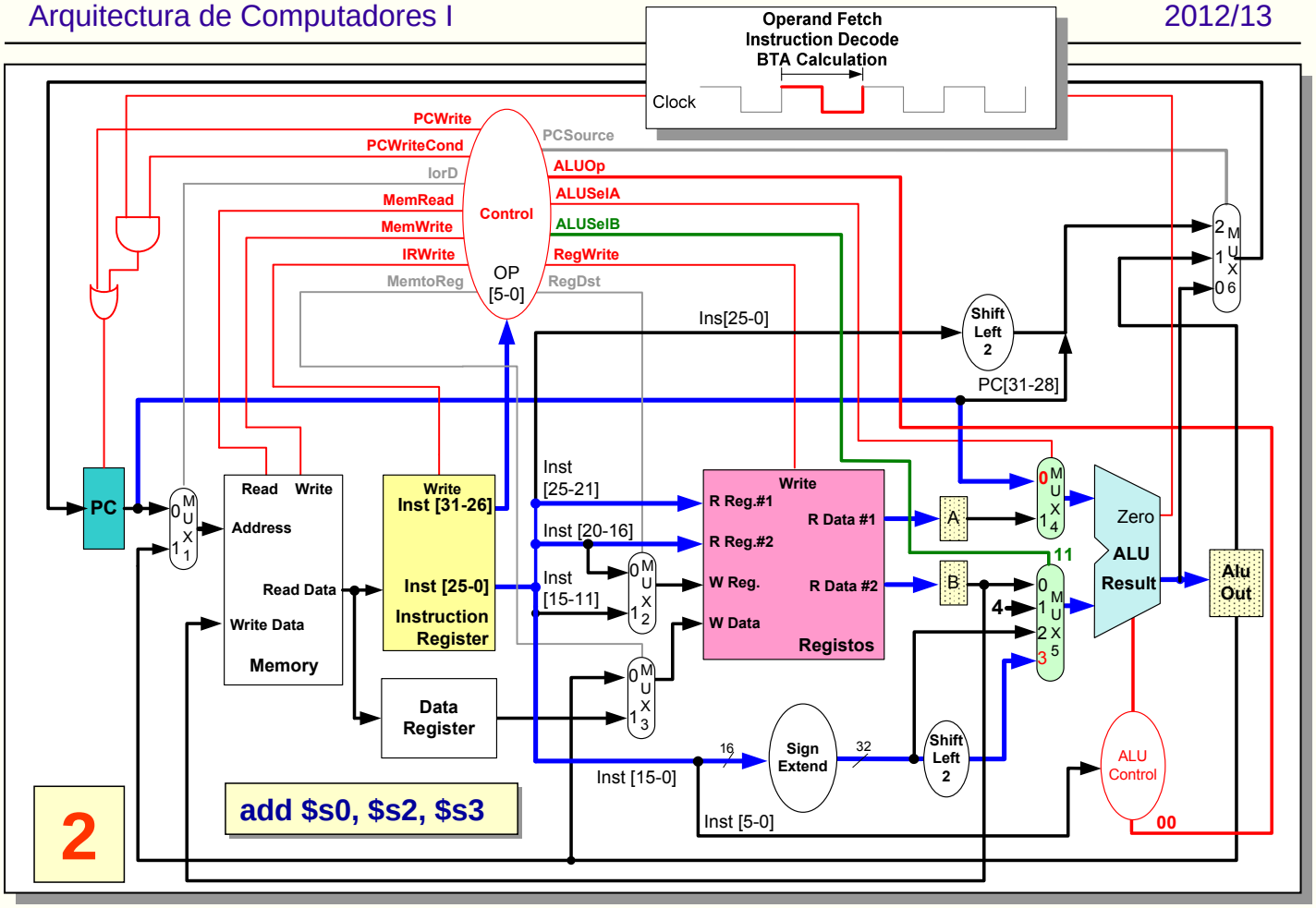
O significado atribuído a cada cor é o seguinte:

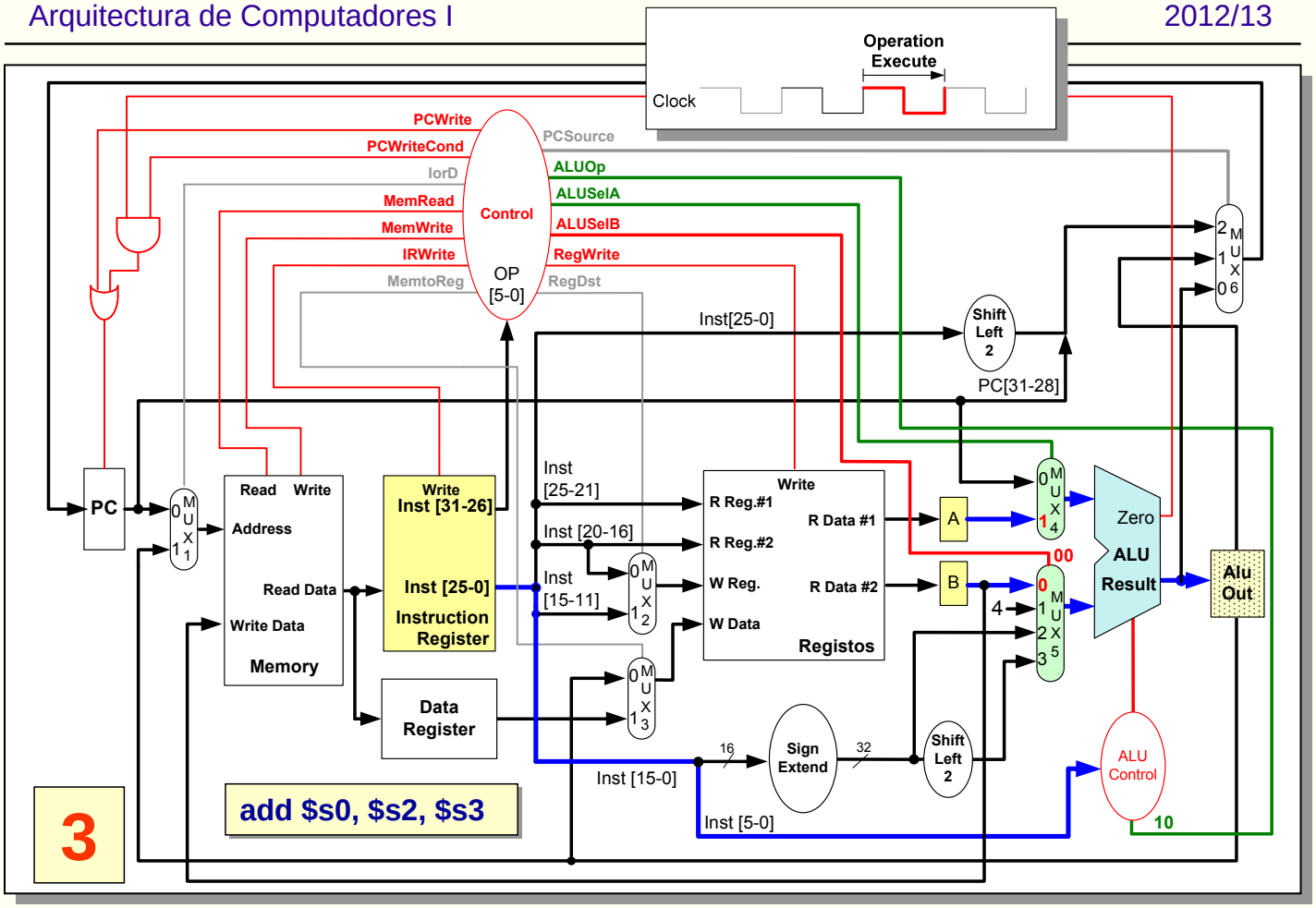
- Sinais de controlo:
  - **vermelho → 0**
  - **verde → diferente de zero**
  - **cinzento → "don't care"**
- Barramentos:
  - **azul → Relevantes no contexto do ciclo da instrução**
  - **preto → Não relevantes no contexto do ciclo da instrução**
- Elementos de estado / combinatórios:
  - fundo branco → Não usados no contexto do ciclo da instrução
  - fundo de cor → Usados no contexto do ciclo instrução
  - fundo de cor com textura → Escritos no final do ciclo de relógio corrente

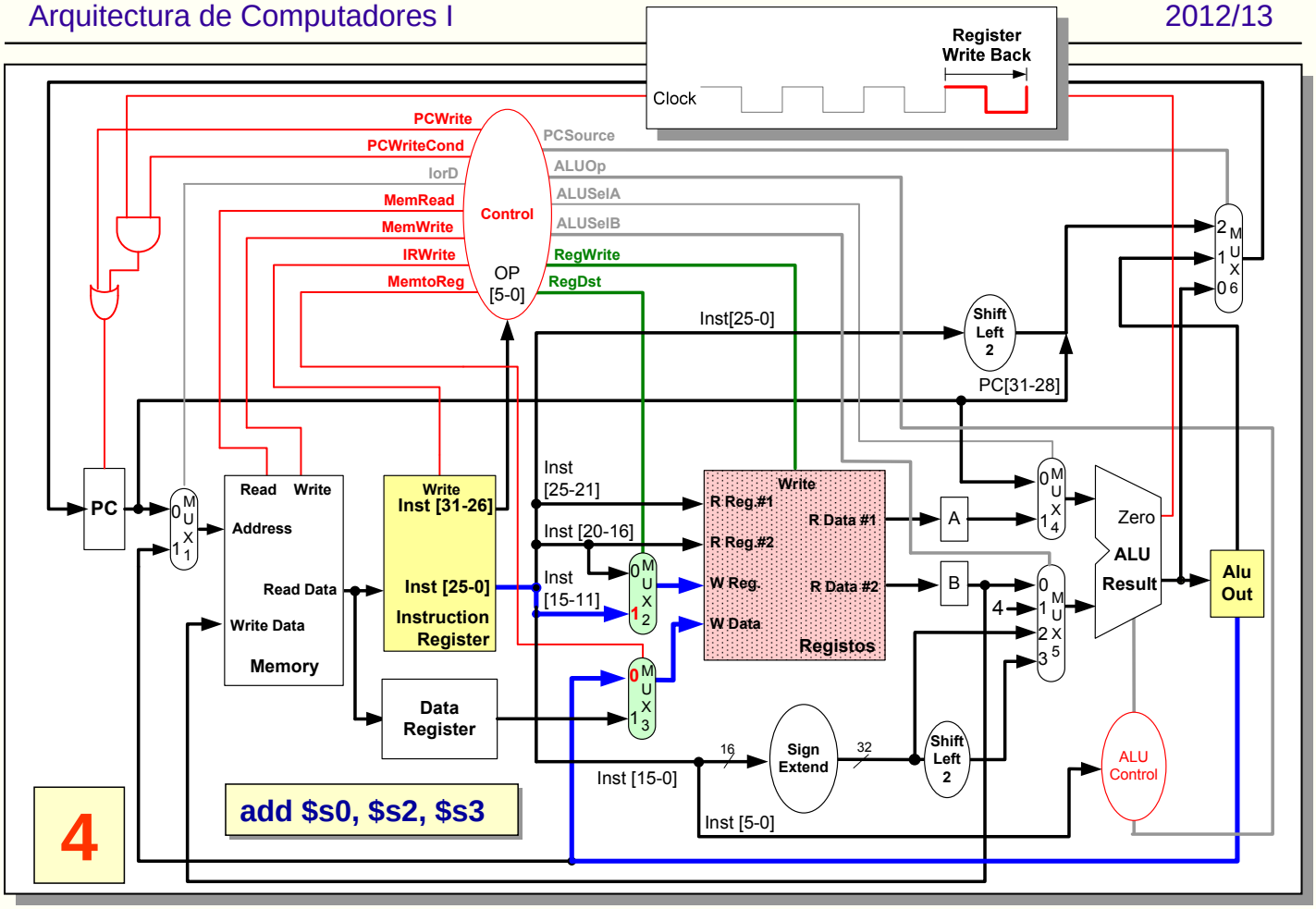
Exemplo 1

Funcionamento do *datapath* nas instruções do tipo R



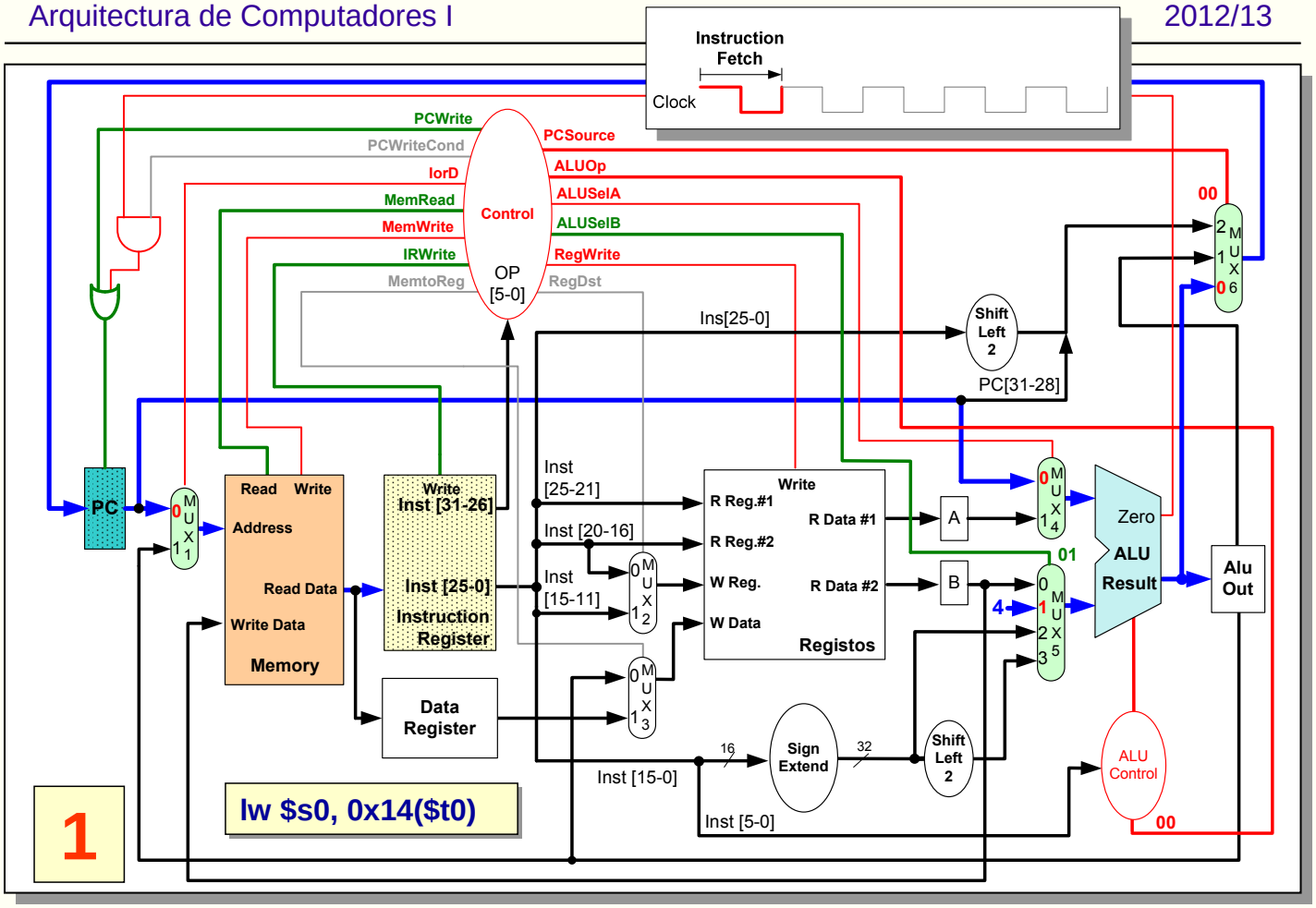

Operand Fetch
Instruction Decode
BTA Calculation
Clock
PCWrite
PCWriteCond
IorD
MemRead
MemWrite
IRWrite
MemtoReg
Control
OP
[5-0]
PCSource
ALUOp
ALUSelA
ALUSelB
RegWrite
RegDst
Ins[25-0]
Shift
Left
2
PC[31-28]
PC
Read Write
Address
Read Data
Write Data
Memory
Write
Inst [31-26]
Inst [25-0]
Instruction
Register
Data
Register
Inst
[25-21]
Inst [20-16]
Inst
[15-11]
Inst [15-0]
Inst [5-0]
Write
R Reg.#1
R Reg.#2
W Reg.
W Data
R Data #1
R Data #2
Registos
A
B
4
Zero
ALU
Result
Alu
Out
Sign
Extend
16
32
ALU
Control
11
00
2
add $s0, $s2, $s3

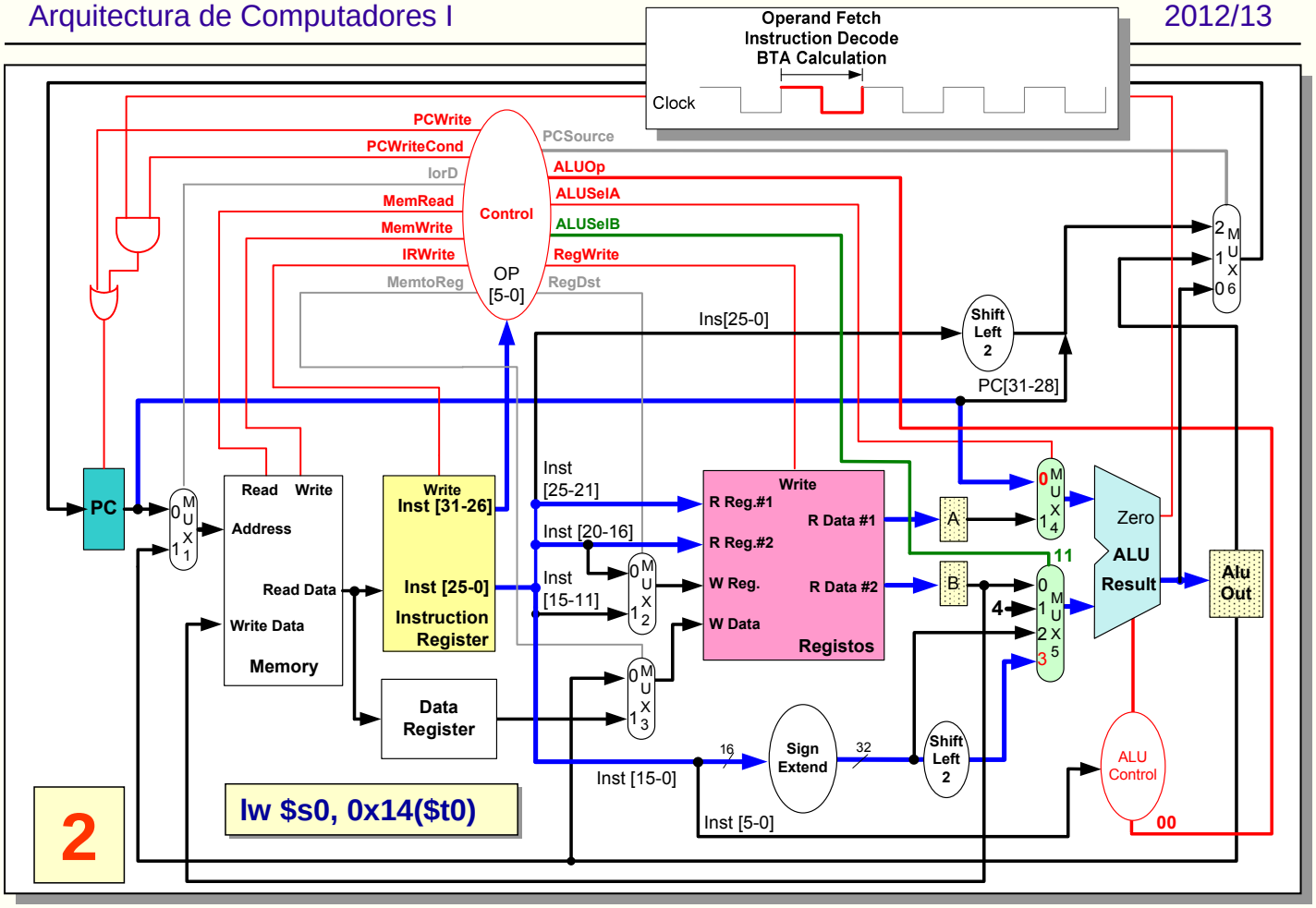

Operation
Execute
Clock
PCWrite
PCWriteCond
IorD
MemRead
MemWrite
IRWrite
MemtoReg
Control
OP
[5-0]
PCSource
ALUOp
ALUSelA
ALUSelB
RegWrite
RegDst
Inst[25-0]
Shift
Left
2
PC[31-28]
PC
Read Write
Address
Read Data
Write Data
Memory
Write
Inst [31-26]
Inst [25-0]
Instruction
Register
Data
Register
Inst
[25-21]
Inst [20-16]
Inst
[15-11]
Inst [15-0]
Inst [5-0]
Write
R Reg.#1
R Reg.#2
W Reg.
W Data
R Data #1
R Data #2
Registos
A
B
4
Zero
ALU
Result
Alu
Out
Sign
Extend
16
32
ALU
Control
00
10
3
add $s0, $s2, $s3

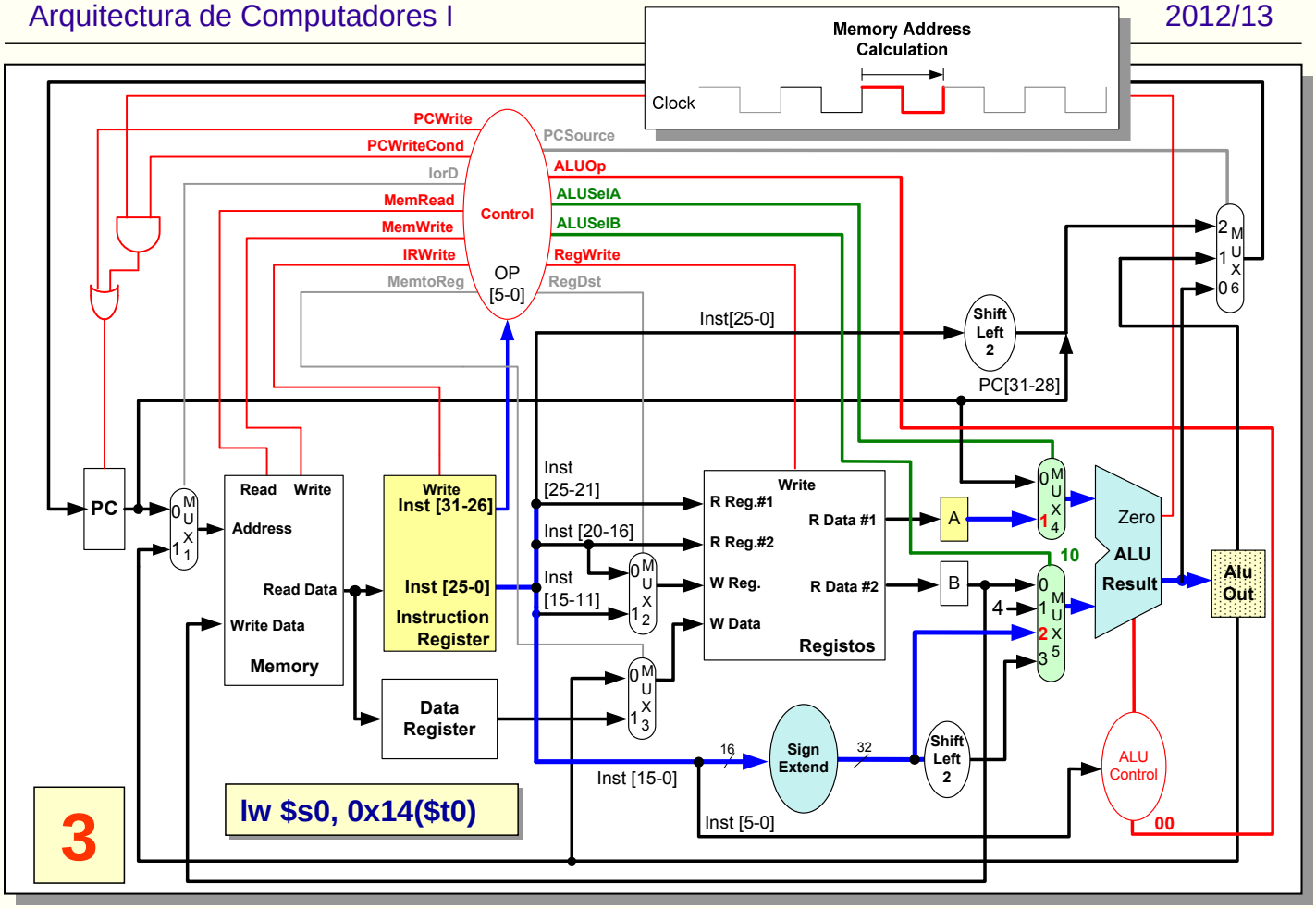


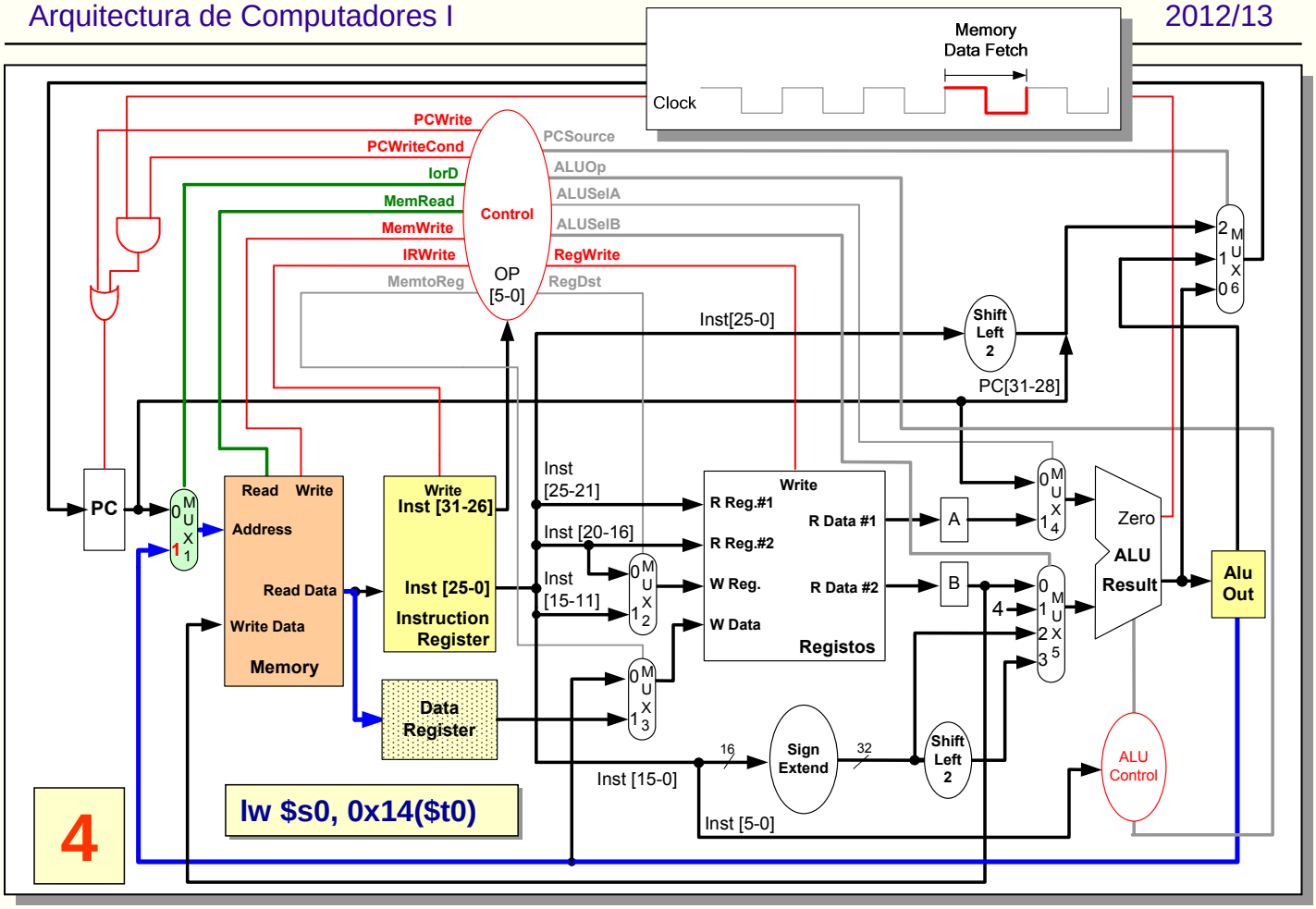
Exemplo 2

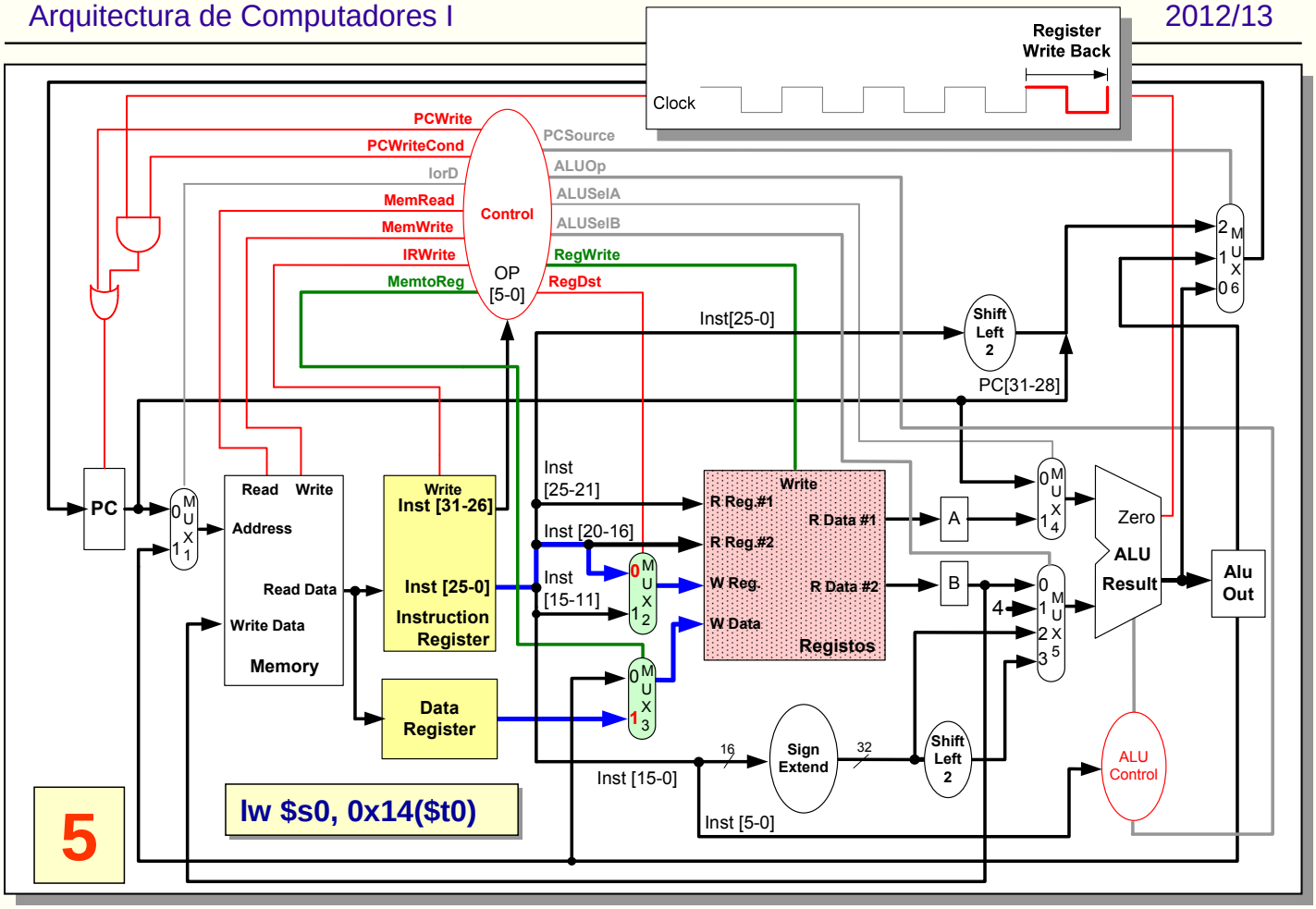
Funcionamento do *datapath* na instrução *load word* ("lw")


Instruction Fetch
Clock
PCWrite
PCWriteCond
IorD
MemRead
MemWrite
IRWrite
MemtoReg
Control
OP [5-0]
PCSource
ALUOp
ALUSelA
ALUSelB
RegWrite
RegDst
Ins[25-0]
Shift Left 2
PC[31-28]
00
PC
Read Write
Address
Read Data
Write Data
Memory
Write Inst [31-26]
Inst [25-0]
Instruction Register
Data Register
Inst [25-21]
Inst [20-16]
Inst [15-11]
Inst [15-0]
Inst [5-0]
Write
R Reg.#1
R Reg.#2
W Reg.
W Data
R Data #1
R Data #2
Registos
A
B
4
01
Zero
ALU
Result
Alu Out
16
Sign Extend
32
Shift Left 2
ALU Control
00
1
lw $s0, 0x14($t0)


Operand Fetch
Instruction Decode
BTA Calculation
Clock
PCWrite
PCWriteCond
IorD
MemRead
MemWrite
IRWrite
MemtoReg
Control
OP [5-0]
PCSource
ALUOp
ALUSelA
ALUSelB
RegWrite
RegDst
Ins[25-0]
Shift Left 2
PC[31-28]
PC
Read Write
Address
Read Data
Write Data
Memory
Write Inst [31-26]
Inst [25-0]
Instruction Register
Data Register
Inst [25-21]
Inst [20-16]
Inst [15-11]
Inst [15-0]
Inst [5-0]
Write
R Reg.#1
R Reg.#2
W Reg.
W Data
R Data #1
R Data #2
Registos
A
B
4
11
Zero
ALU
Result
Alu Out
16
Sign Extend
32
Shift Left 2
ALU Control
00
2
lw $s0, 0x14($t0)

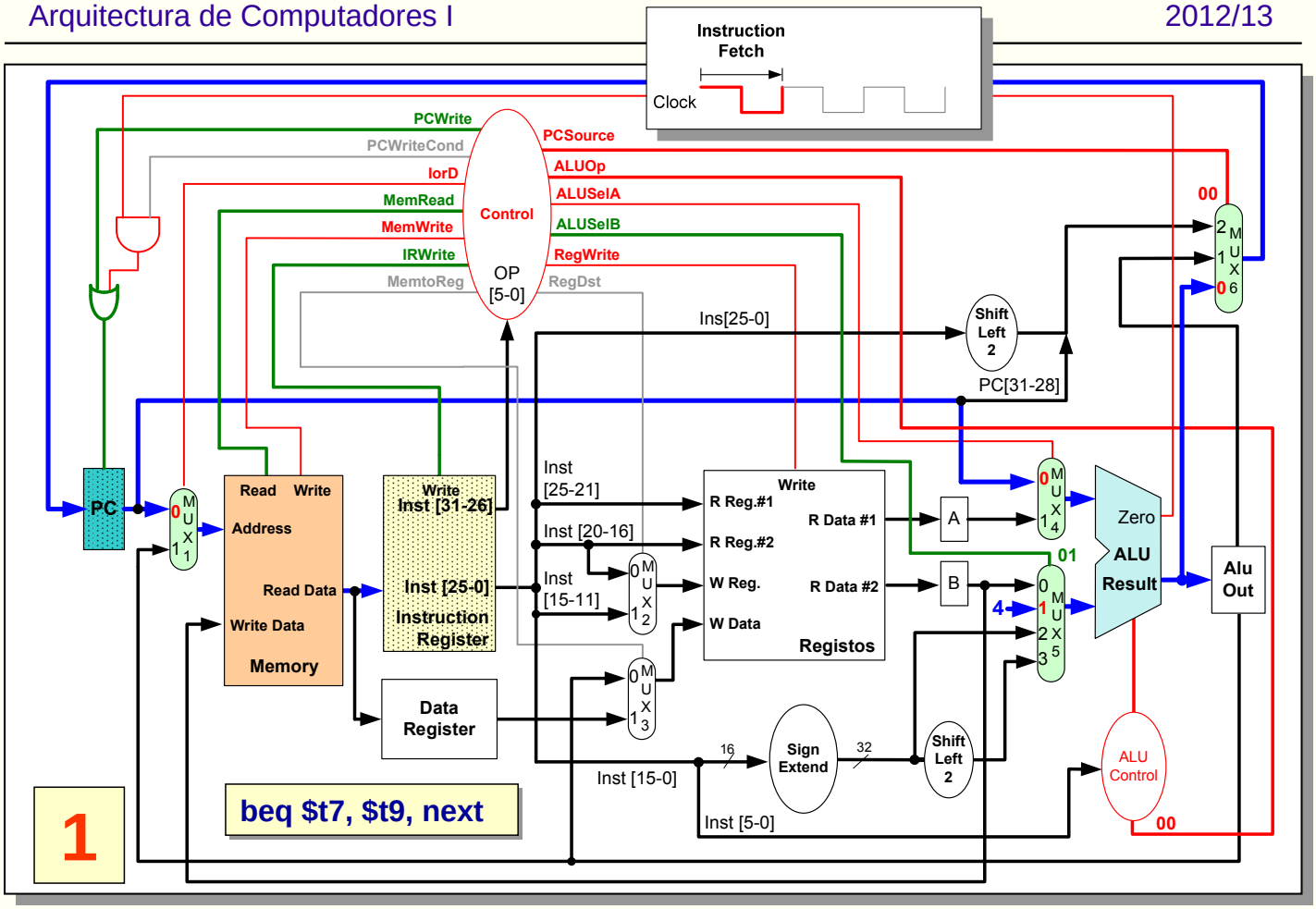

Memory Address Calculation
Clock
PCWrite
PCWriteCond
IorD
MemRead
MemWrite
IRWrite
MemtoReg
Control
OP [5-0]
PCSource
ALUOp
ALUSelA
ALUSelB
RegWrite
RegDst
Inst[25-0]
Shift Left 2
PC[31-28]
PC
Read Write
Address
Read Data
Write Data
Memory
Write Inst [31-26]
Inst [25-0]
Instruction Register
Data Register
Inst [25-21]
Inst [20-16]
Inst [15-11]
Inst [15-0]
Inst [5-0]
Write
R Reg.#1
R Reg.#2
W Reg.
W Data
R Data #1
R Data #2
Registos
A
B
4
10
Zero
ALU Result
Alu Out
Sign Extend
16
32
Shift Left 2
ALU Control
00
3
lw $s0, 0x14($t0)

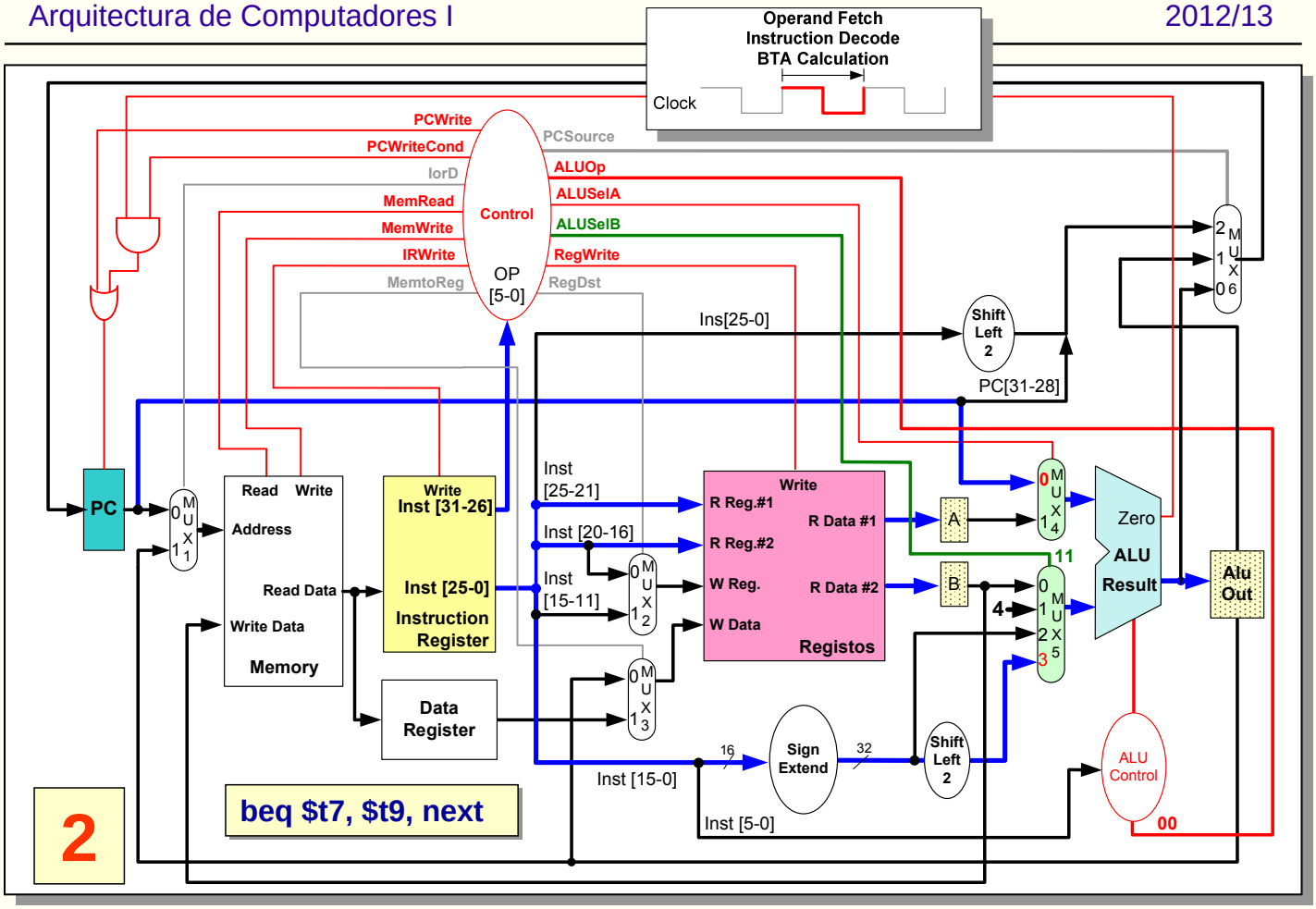

Memory Data Fetch
Clock
PCWrite
PCWriteCond
IorD
MemRead
MemWrite
IRWrite
MemtoReg
Control
OP [5-0]
PCSource
ALUOp
ALUSelA
ALUSelB
RegWrite
RegDst
Inst[25-0]
Shift Left 2
PC[31-28]
PC
Read Write
Address
Read Data
Write Data
Memory
Write Inst [31-26]
Inst [25-0]
Instruction Register
Data Register
Inst [25-21]
Inst [20-16]
Inst [15-11]
Inst [15-0]
Inst [5-0]
Write
R Reg.#1
R Reg.#2
W Reg.
W Data
R Data #1
R Data #2
Registos
A
B
4
Zero
ALU Result
Alu Out
Sign Extend
16
32
Shift Left 2
ALU Control
4
lw $s0, 0x14($t0)

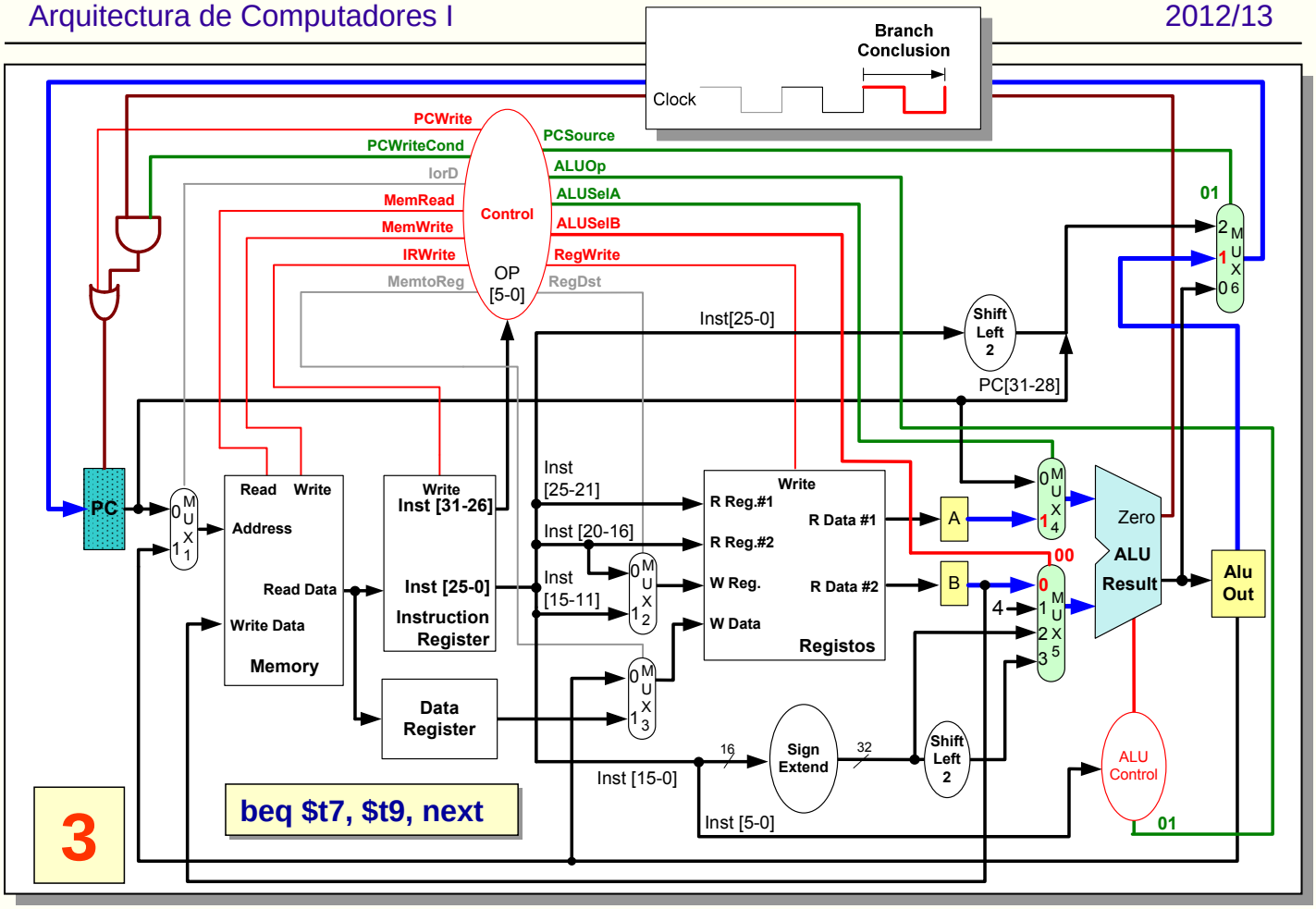


Exemplo 3

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

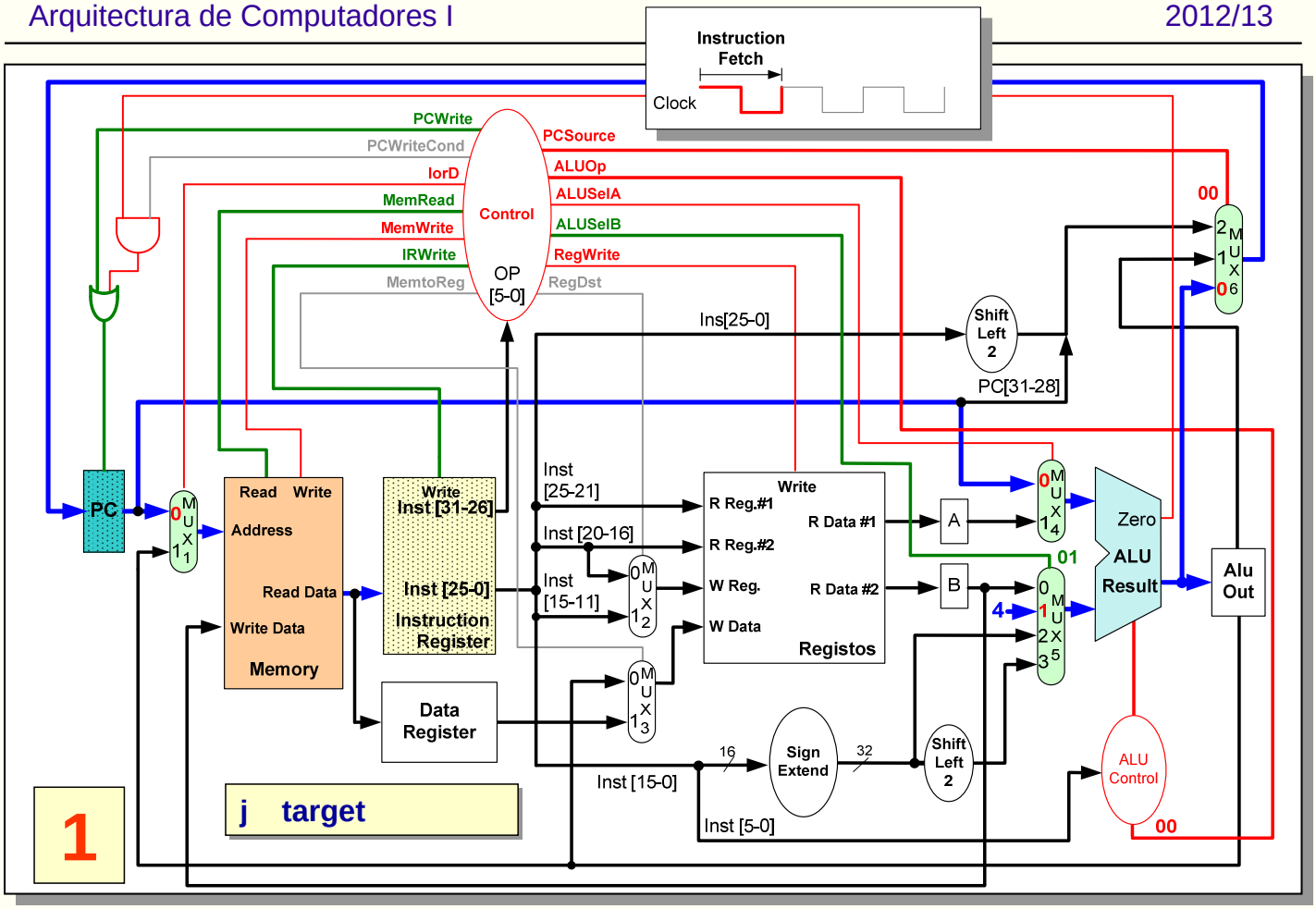

Instruction Fetch
Clock
PCWrite
PCWriteCond
IorD
MemRead
MemWrite
IRWrite
MemtoReg
Control
OP [5-0]
PCSource
ALUOp
ALUSelA
ALUSelB
RegWrite
RegDst
Ins[25-0]
Shift Left 2
PC[31-28]
00
PC
Read Write
Address
Read Data
Write Data
Memory
Write Inst [31-26]
Inst [25-0]
Instruction Register
Data Register
Inst [25-21]
Inst [20-16]
Inst [15-11]
Inst [15-0]
Inst [5-0]
Write
R Reg.#1
R Reg.#2
W Reg.
W Data
R Data #1
R Data #2
Registos
A
B
4
01
Zero
ALU
Result
Alu Out
16
Sign Extend
32
Shift Left 2
ALU Control
00
1
beq $t7, $t9, next

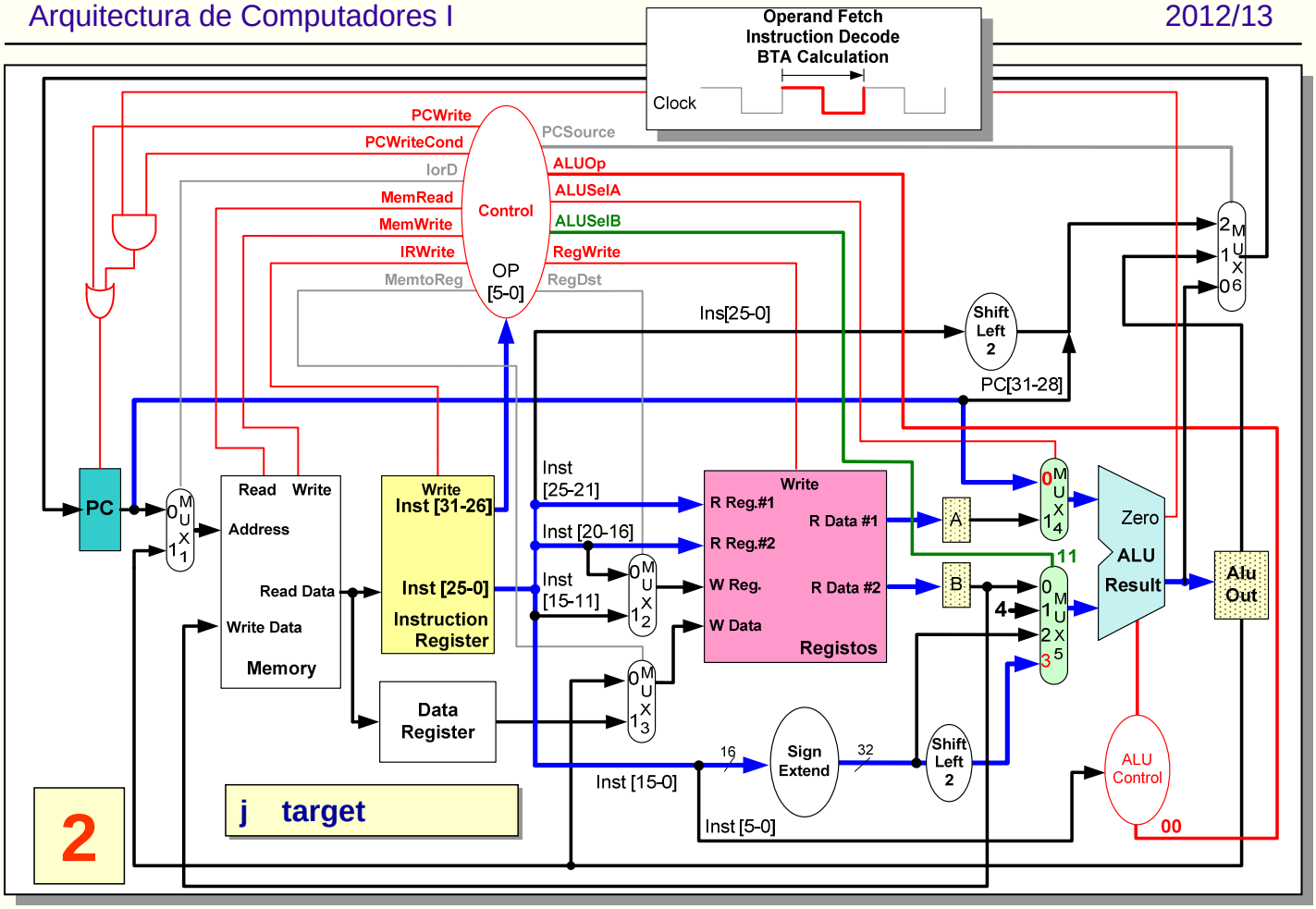

Operand Fetch
Instruction Decode
BTA Calculation
Clock
PCWrite
PCWriteCond
IorD
MemRead
MemWrite
IRWrite
MemtoReg
Control
OP [5-0]
PCSource
ALUOp
ALUSelA
ALUSelB
RegWrite
RegDst
Ins[25-0]
Shift Left 2
PC[31-28]
PC
Read Write
Address
Read Data
Write Data
Memory
Write Inst [31-26]
Inst [25-0]
Instruction Register
Data Register
Inst [25-21]
Inst [20-16]
Inst [15-11]
Inst [15-0]
Inst [5-0]
Write
R Reg.#1
R Reg.#2
W Reg.
W Data
R Data #1
R Data #2
Registos
A
B
4
11
Zero
ALU
Result
Alu Out
16
Sign Extend
32
Shift Left 2
ALU Control
00
2
beq $t7, $t9, next

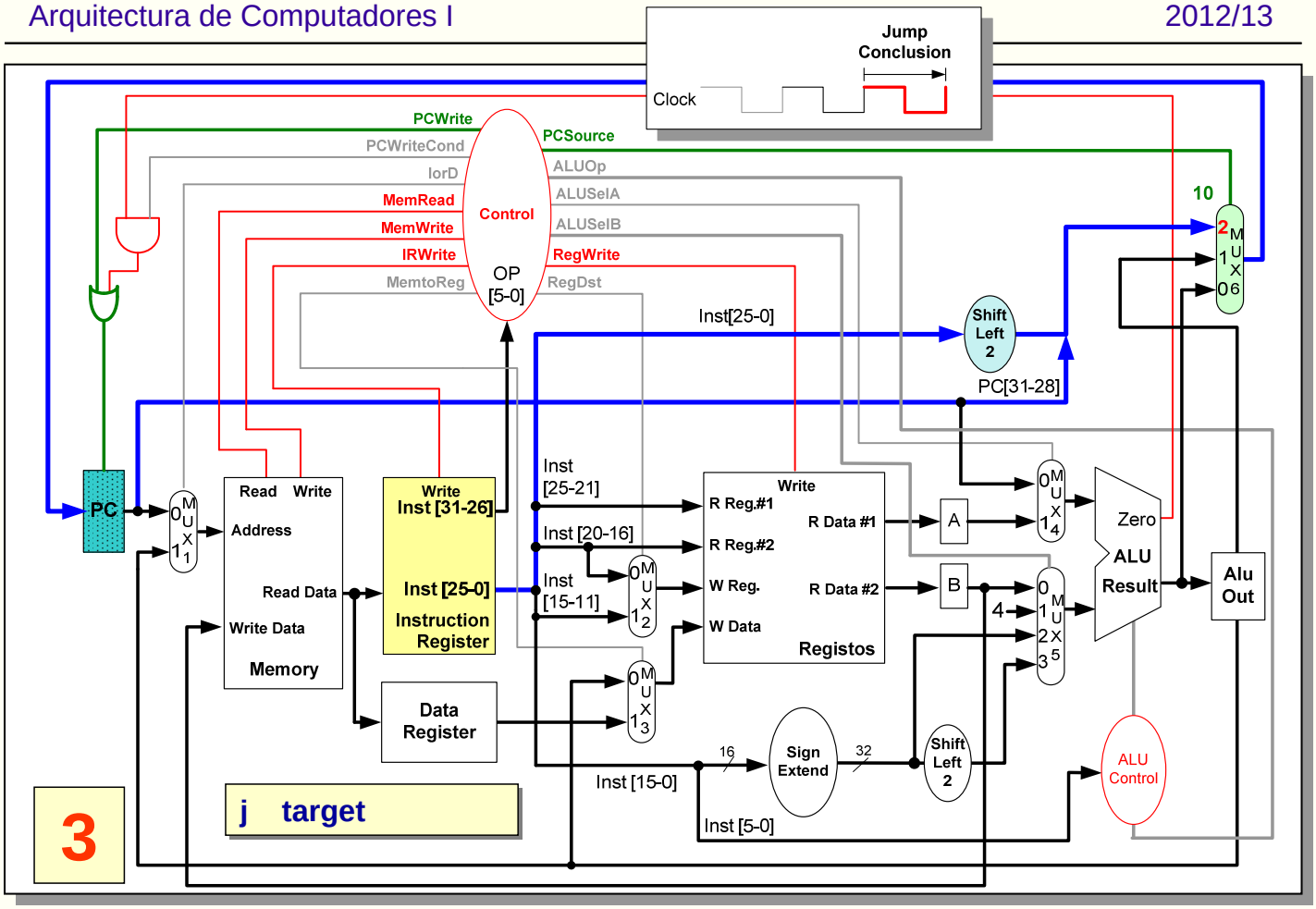


Exemplo 4

Funcionamento do *datapath* na instrução de salto incondicional ("j")


Instruction Fetch
Clock
PCWrite
PCWriteCond
IorD
MemRead
MemWrite
IRWrite
MemtoReg
Control
OP [5-0]
PCSource
ALUOp
ALUSelA
ALUSelB
RegWrite
RegDst
Ins[25-0]
Shift Left 2
PC[31-28]
PC
Read Write
Address
Read Data
Write Data
Memory
Write Inst [31-26]
Inst [25-0]
Instruction Register
Data Register
Inst [25-21]
Inst [20-16]
Inst [15-11]
Write
R Reg.#1
R Reg.#2
W Reg.
W Data
R Data #1
R Data #2
Registos
A
B
4
Zero
ALU Result
Alu Out
Inst [15-0]
16
Sign Extend
32
Shift Left 2
Inst [5-0]
ALU Control
00
01
1
j target

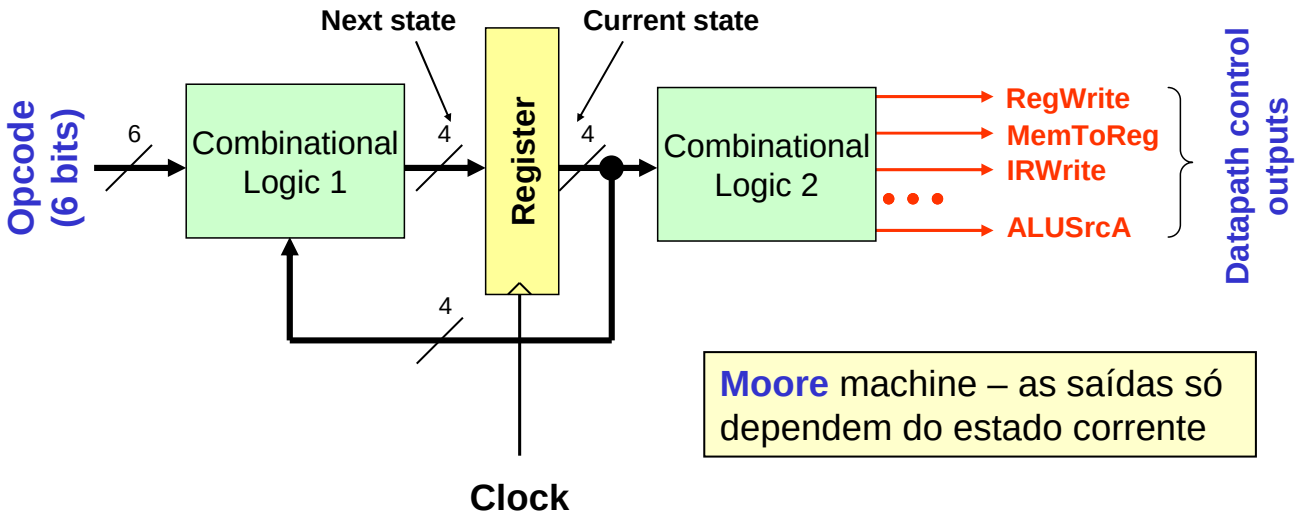

Operand Fetch
Instruction Decode
BTA Calculation
Clock
PCWrite
PCWriteCond
IorD
MemRead
MemWrite
IRWrite
MemtoReg
Control
OP [5-0]
PCSource
ALUOp
ALUSelA
ALUSelB
RegWrite
RegDst
Ins[25-0]
Shift Left 2
PC[31-28]
PC
Read Write
Address
Read Data
Write Data
Memory
Write Inst [31-26]
Inst [25-0]
Instruction Register
Data Register
Inst [25-21]
Inst [20-16]
Inst [15-11]
Write
R Reg.#1
R Reg.#2
W Reg.
W Data
R Data #1
R Data #2
Registos
A
B
4
Zero
ALU Result
Alu Out
Inst [15-0]
16
Sign Extend
32
Shift Left 2
Inst [5-0]
ALU Control
00
11
2
j target



A unidade de controlo do *datapath multicycle*

- No *datapath* ***single cycle***, cada instrução era executada num único ciclo de relógio:
  - A unidade de controlo é responsável pela geração de um conjunto de sinais que não se alteram durante a execução de cada instrução.
  - A relação entre os sinais de controlo e o código de operação pode assim ser gerado por um circuito meramente combinatório.
- No *datapath* ***multicycle***, cada instrução é decomposta num conjunto de ciclos de execução, correspondendo cada um destes a um período de relógio distinto:
  - A geração dos sinais de controlo ao longo do conjunto de ciclos em que é decomposta cada instrução depende da instrução particular que está a ser executada.
  - A solução combinatória deixa portanto de poder ser utilizada neste caso, sendo necessário recorrer a uma máquina de estados.

A unidade de controlo do *datapath multicycle*

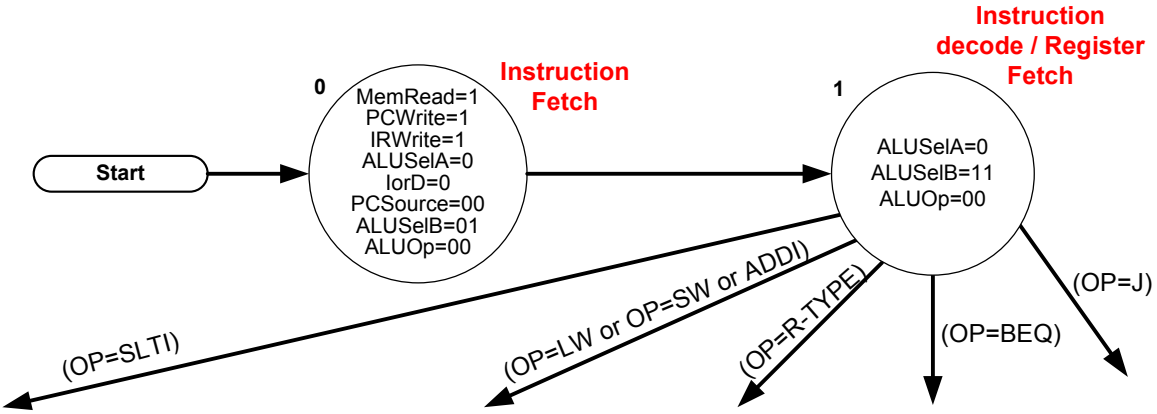


Moore machine – as saídas só dependem do estado corrente

O estado seguinte é função do estado actual e das entradas (opcode)

A unidade de controlo do *datapath multicycle*

Como já vimos, os dois primeiros ciclos de instrução são comuns a todas as instruções. Correspondem assim a dois estados únicos, sendo a transição entre ambos incondicional e independente de qualquer sinal de entrada.



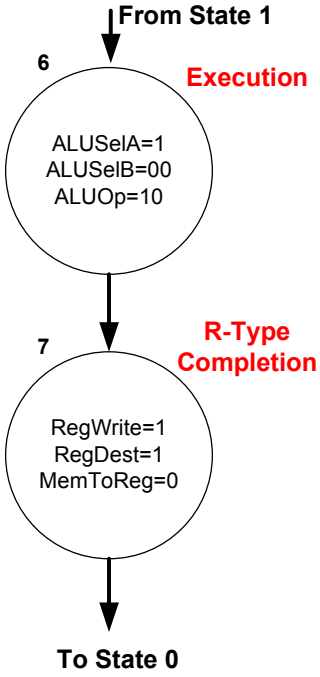
O segundo estado, por sua vez, tem cinco destinos distintos, dependendo do valor do campo OP da instrução.

Presume-se que os sinais de saída não explicitados em cada estado são irrelevantes (multiplexers) ou se encontram no estado não activo (controlo de elementos de estado).

A unidade de controlo do *datapath multicycle*

Nas instruções do tipo "*R*", são necessários mais dois estados:

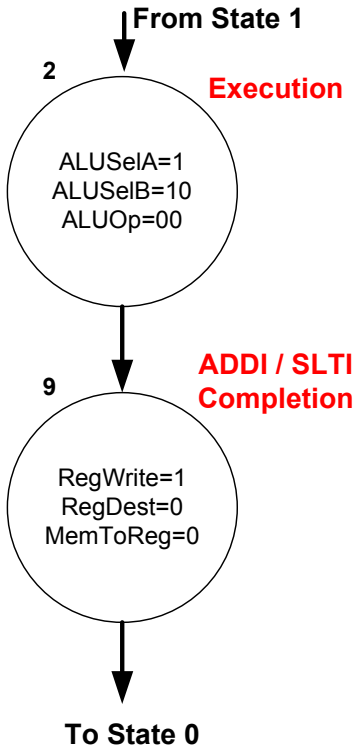
- Um para controlar a execução da operação aritmética ou lógica e para assegurar o encaminhamento do 2º operando para a ALU
- Outro para escrever o resultado no registo destino.



A unidade de controlo do *datapath multicycle*

Na instrução "*ADDI*", são necessários mais dois estados:

- Um para definir a operação a realizar na ALU e para assegurar o encaminhamento do 2º operando para a ALU
- Outro para escrever o resultado no registo destino.

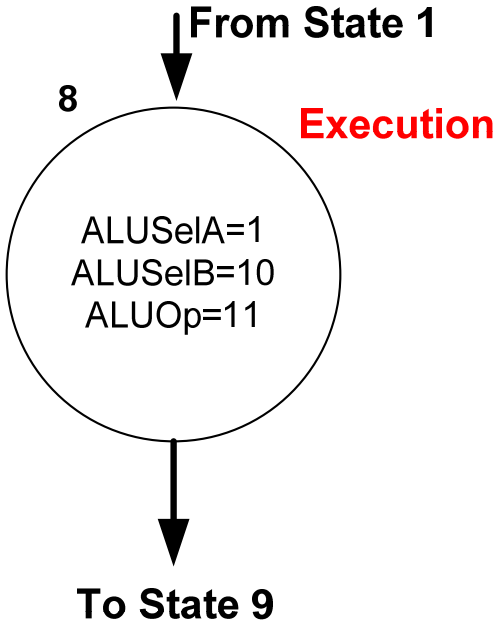


A unidade de controlo do *datapath multicycle*

Para a instrução "*SLTI*", é necessário mais um estado:

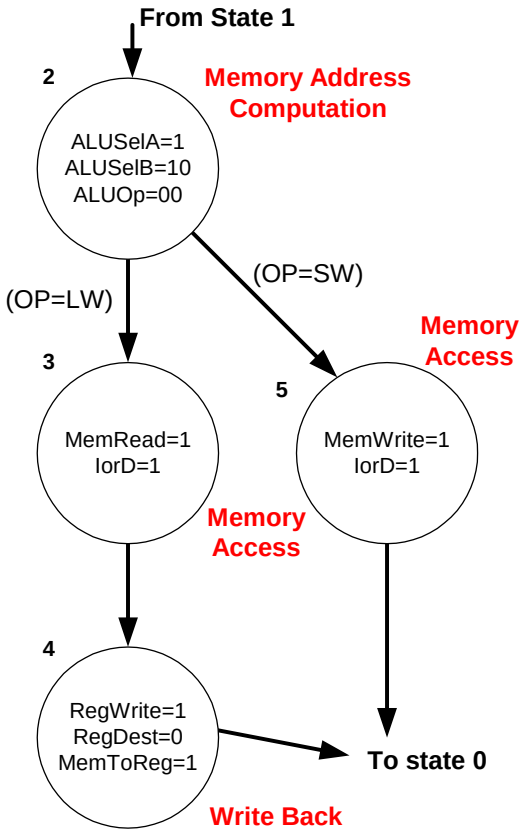
- Para definir a operação a realizar na ALU e para assegurar o encaminhamento do 2º operando para a ALU

A conclusão desta instrução é igual à instrução "*ADDI*" (daí a partilha do estado 9)



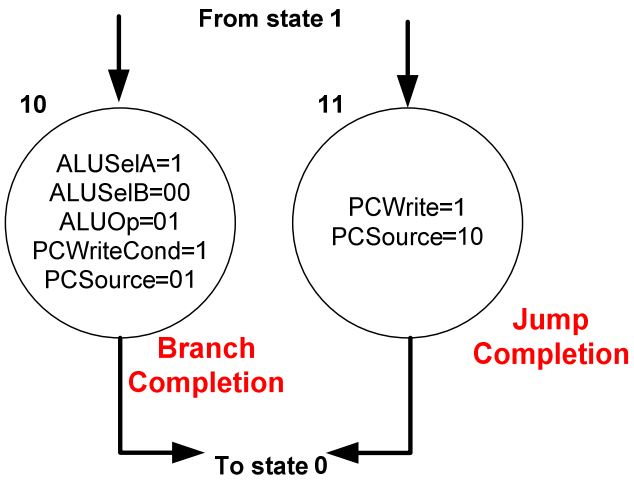
A unidade de controlo do *datapath multicycle*

- Nas instruções de "load/store", o estado dois é dedicado a determinar o endereço da memória externa sobre a qual será efectuada a operação de escrita ou leitura.
- A instrução de "load" obriga a um estado suplementar, face à instrução de "store", para permitir a escrita do valor lido no registo destino.

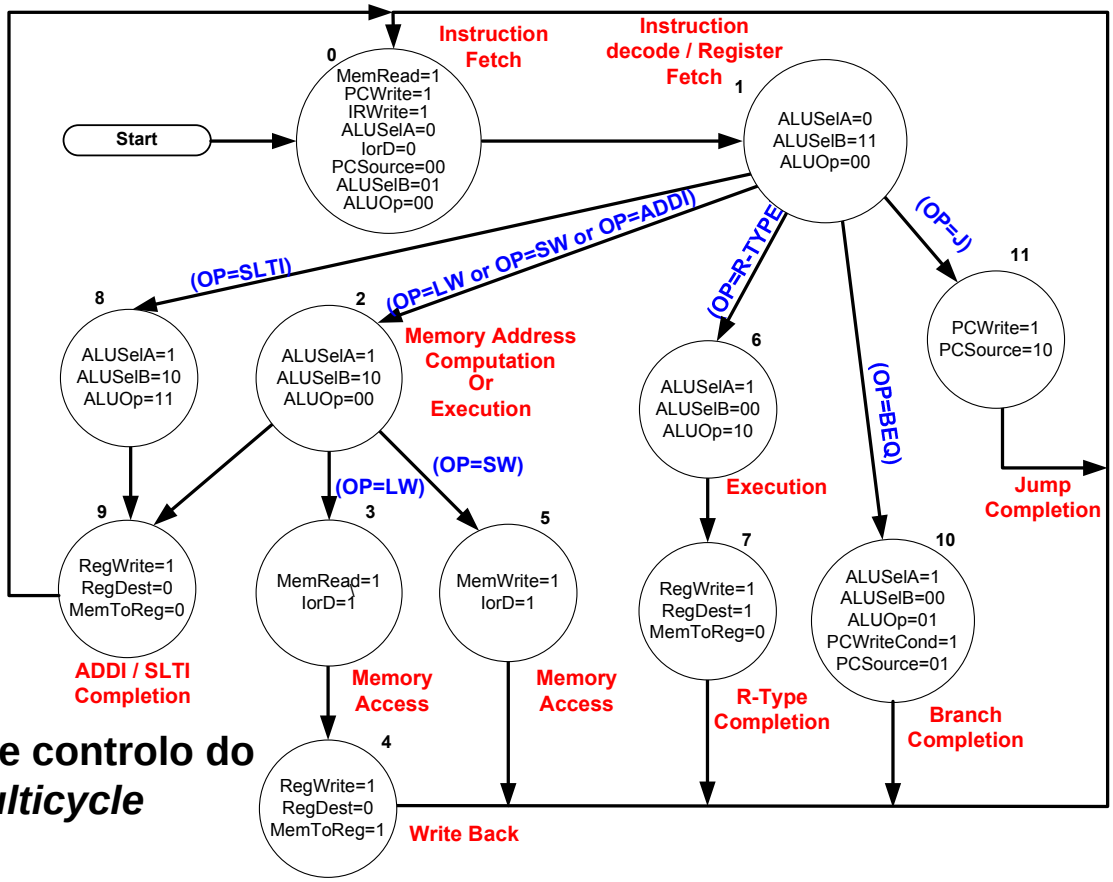


A unidade de controlo do *datapath multicycle*

As instruções de "branch" condicional e as instruções de "jump", finalmente, carecem apenas de mais um estado para poderem ser completadas.



A unidade de controlo do *datapath multicycle*



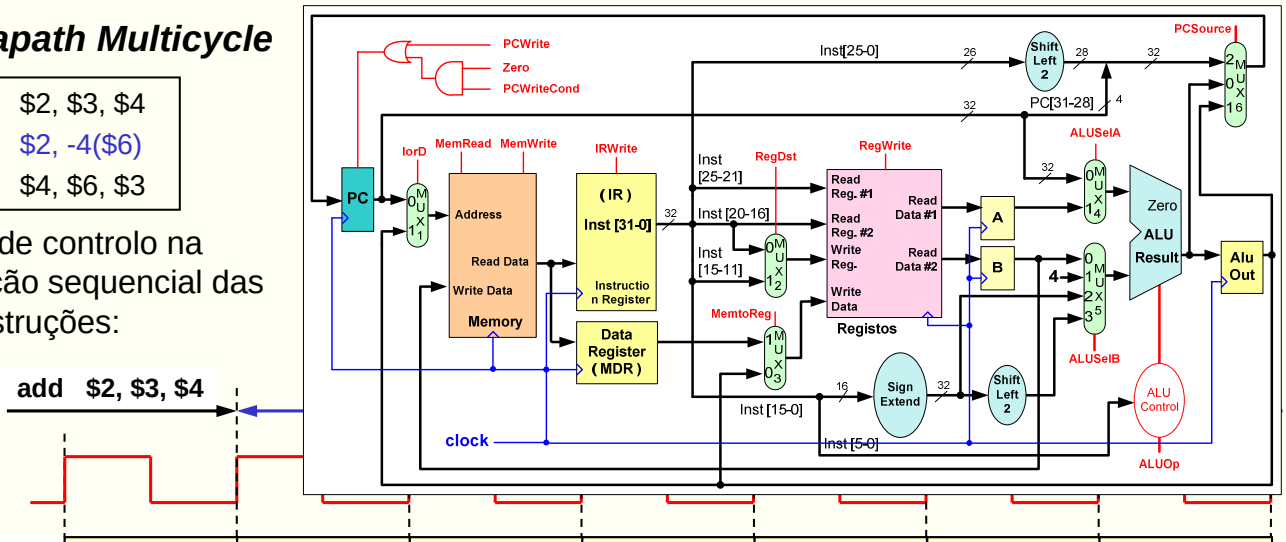
A unidade de controlo do *datapath multicycle*

- A unidade de controlo que acabamos de desenhar tem apenas 12 estados (4 variáveis de estado). A representação do seu aspecto funcional na forma de um diagrama de estados é portanto perfeitamente razoável.
- Uma versão completa do *datapath* do MIPS (mais de cem instruções distintas) pode implicar ciclos de execução que variem entre dois e vinte períodos de relógio, complicando significativamente o diagrama de estados.
- Em arquitecturas do Set de Instruções mais complexas, com um número muito superior de instruções agrupadas num número muito variado de classes, a unidade de controlo pode requerer milhares de estados agrupados em centenas de sequências distintas.
- Nestes casos, o recurso a uma representação gráfica da máquina de estados é não só inapropriada como virtualmente impossível de realizar. A **micro-programação** é uma forma alternativa de representar a unidade de controlo do ponto de vista funcional.

O *Datapath Multicycle*

add	$2, $3, $4
sw	$2, -4($6)
or	$4, $6, $3

Sinais de controlo na execução sequencial das três instruções:



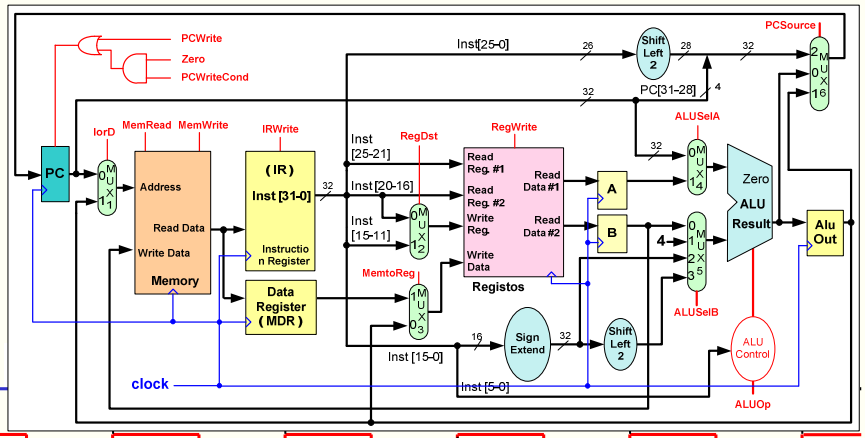
PCWriteCond	0	X	0	0	0	X	0
PCWrite	0	1	0	0	0	1	0
MemWrite	0	0	0	0	1	0	0
MemRead	0	1	0	0	0	1	0
MemToReg	0	X	X	X	X	X	X
IRWrite	0	1	0	0	0	1	0
ALUSelA	X	0	0	1	X	0	0
ALUSelB	XX	01	11	10	XX	01	11
ALUOp	XX	00	00	00	XX	00	00
IorD	X	0	X	X	1	0	X
PCSource	XX	00	XX	XX	XX	00	XX
RegWrite	1	0	0	0	0	0	0
RegDst	1	X	X	X	X	X	X

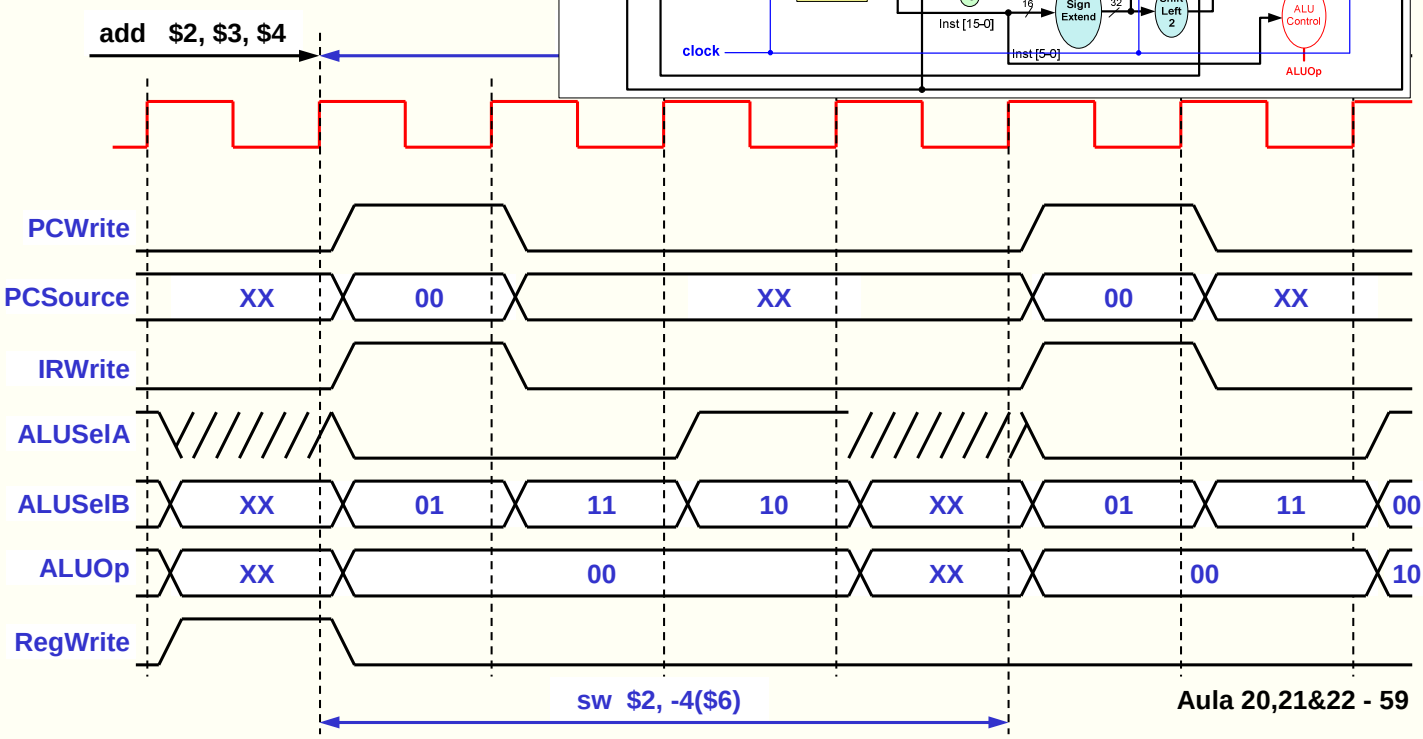
sw $2, -4($6)

O *Datapath Multicycle*

add	$2, $3, $4
sw	$2, -4($6)
or	$4, $6, $3

Sinais de controlo: diagrama temporal



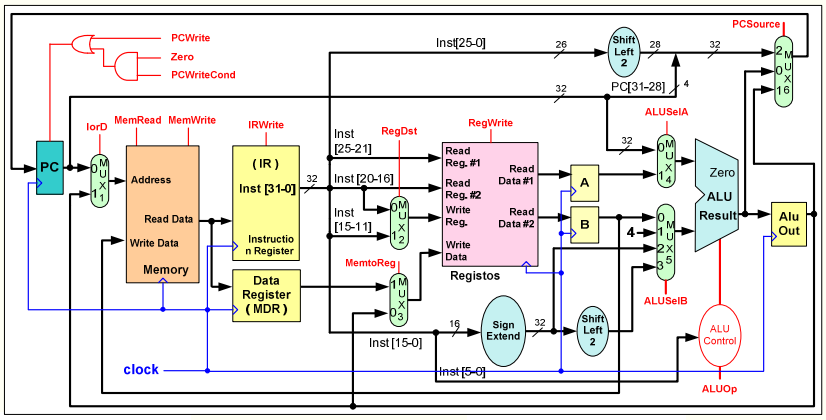


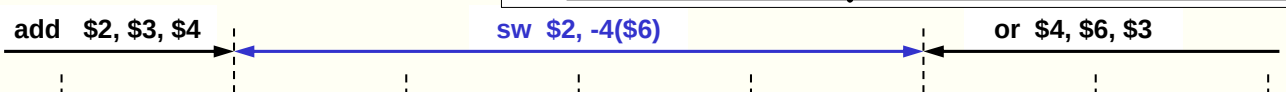
O *Datapath Multicycle*

```
00400048  add $2, $3, $4  # 00641020
0040004C  sw  $2, -4($6)  # ACC2FFFC
00400050  or  $4, $6, $3  # 00C32025
```

Valores calculados / obtidos em cada ciclo de relógio:

$3	20001FA6
$4	81002378
$6	10012480





PC	0040004C	0040004C	00400050	00400050	00400050	00400050	00400054
IR	00641020	00641020	ACC2FFFC	ACC2FFFC	ACC2FFFC	ACC2FFFC	00C32025
MDR	?	?	ACC2FFFC	?	?	?	00C32025
ALU Res	?	00400050	00400040	1001247C	?	00400054	004080E8
ALU Out	A100431E	?	00400050	00400040	1001247C	?	00400054
ALU Zero	?	0	0	0	?	0	0
A	20001FA6	20001FA6	20001FA6	10012480	10012480	10012480	10012480
B	81002378	81002378	81002378	A100431E	A100431E	A100431E	A100431E
ALUOp	XX	00	00	00	XX	00	00

DETI-UA, AC1 | Todos os valores em hexadecimal | Cód.: sw-0x2B, add-0x20, or-0x25