## Arquitectura de Computadores I 2007/08

1º Semestre de 2007/2008

Bernardo Cunha, José Luís Azevedo, Arnaldo Oliveira



Arquitectura de Computadores I 2007/08

**Aulas 19&20**

Limitações das arquitecturas *single cycle*

Versão de referência de uma arquitectura *multicycle*

Exemplos de funcionamento numa arquitectura *multicycle*

## Limitações das soluções *single-cycle*

Consideremos 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/escrita (MEM): 5ns
- Acesso ao "register file" para leitura/escrita (REG): 3ns
- Operação da ALU (ALU): 4ns
- Operação de um somador (SOM): 1ns
- Multiplexers e restantes elementos operativos: 0ns
- Unidade de controlo (CNT): 1ns



Arquitectura de Computadores I 2007/08

## Limitações das soluções *single-cycle*

Como discutíramos 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* corresponderá assim ao somatório dos atrasos introduzidos por cada um dos elementos operativos envolvidos na execução da instrução.

Note-se que apenas os elementos operativos que se encontram em série contribuem para aumentar o tempo necessário para concluir a execução da instrução.

## Limitações das soluções *single-cycle*

**Considerando os valores dos tempos de atraso que admitíramos anteriormente**, os tempos de execução das várias instruções suportadas pelo *datapath single cycle* serão:

| | | |
|---|---|---|
| Instruções tipo R: | MEM + REG + ALU + REG | = 15ns |
| Instruções de *load*: | MEM + REG + ALU + MEM + REG | = 20ns |
| Instruções de *store*: | MEM + REG + ALU + MEM | = 17ns |
| Instruções de *branch* condicional: | MEM + REG + ALU | = 12ns |
| Instruções de *jump*: | MEM + CNT | = 6ns |





## Limitações das soluções *single-cycle*

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

$$F_{max} = 1 / 20ns = \mathbf{50MHz}$$

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.

## 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 solução possível, mas tecnicamente muito complicada, seria usar um relógio de frequência variável, ajustável em função da instrução que vai ser executada).
- Elementos operativos que estejam envolvidos na execução de uma mesma instrução não podem ser agrupados (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 acomodadas num número variável de ciclos de relógio, por forma a que, em cada ciclo, apenas seja executada uma operação básica suportada por um único elemento operativo fundamental – memória, *file register* e ALU.
- Desta forma, o período de relógio fica apenas limitado pelo maior dos tempos de atraso de cada um dos elementos operativos.

## Alternativa às soluções *single-cycle*

Para os tempos de atraso que consideráramos anteriormente, a máxima frequência de relógio seria assim:

$$F_{max} = 1 / MEM = 1 / 5ns = \mathbf{200MHz}$$

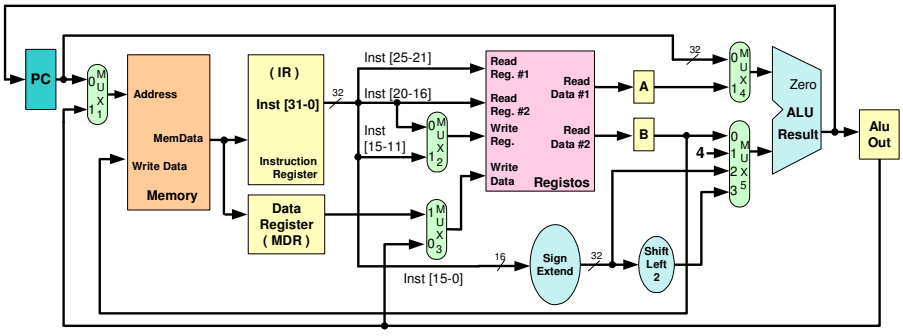
Uma outra vantagem duma solução *multicycle* é que um mesmo elemento operativo pode ser utilizado no contexto da execução duma mesma instrução desde que em ciclos de relógio distintos.

É o caso da memória externa – que poderá ser partilhada por instruções e dados, ou da ALU, que poderá ser usada para calcular o valor de PC+4, os endereços alvo dos *branches* ou as operações aritméticas e lógicas correspondentes à instrução em curso.



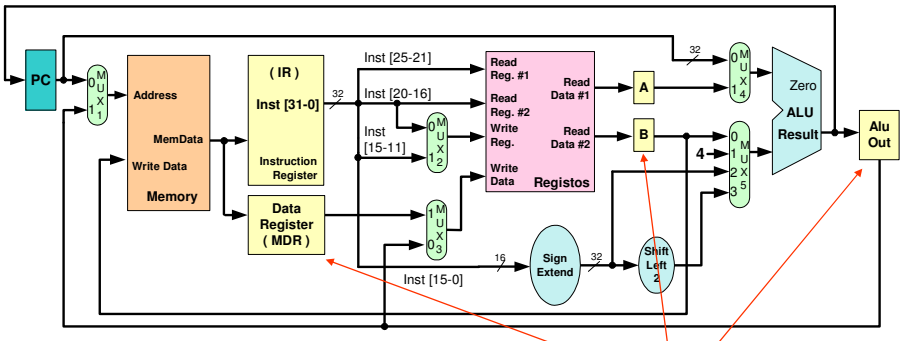


## O *Datapath Multicycle* (sem as instruções de salto)




- Uma única memória para programa e dados
- Uma única ALU (em vez de uma ALU e dois somadores)

## O *Datapath Multicycle* (sem as instruções de salto)

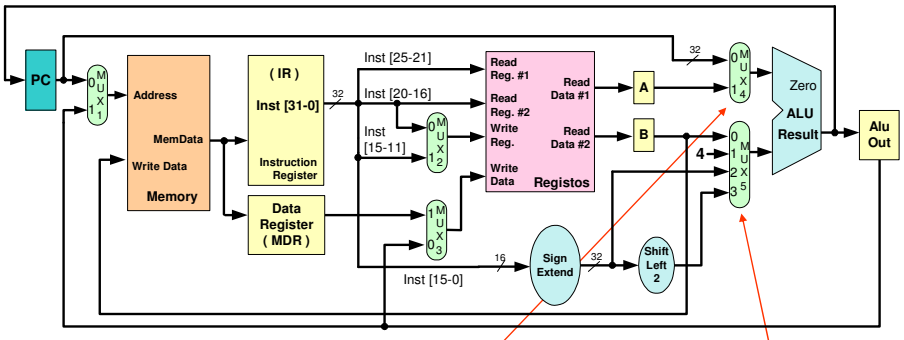



- Registos adicionados à saída das unidades funcionais mais importantes para armazenamento de informação que será utilizada no clock seguinte



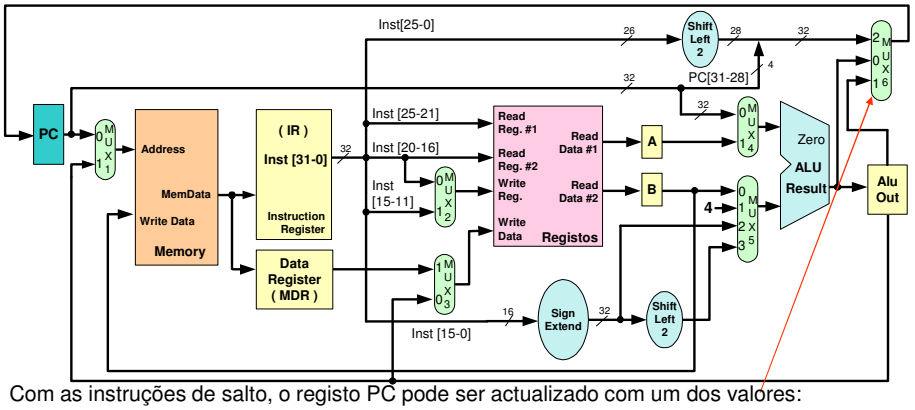
Arquitectura de Computadores I 2007/08

## O *Datapath Multicycle* (sem as instruções de salto)




- 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 destino das instruções de *branch*

## O *Datapath Multicycle*



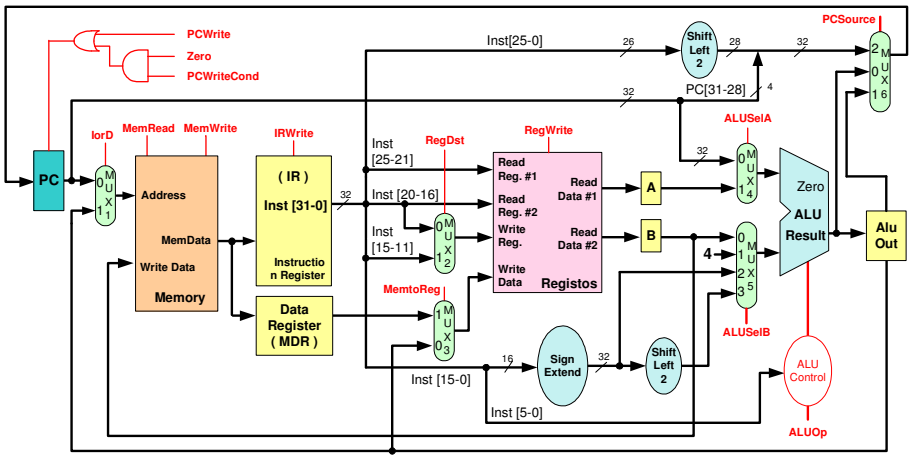

Com as instruções de salto, o registo PC pode ser actualizado com um dos valores:

- A saída da ALU que contém o PC+4 calculado durante o instruction fetch
- A saída do registo ALUOut que armazena o endereço destino das instruções de branch calculado na ALU
- O endereço da instrução "j" - 26 LSB da instrução multiplicados por 4 (shift left 2) concatenados com os 4 MSB do PC actual





## 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 da memória externa |
| IorD | O PC é usado para fornecer o endereço à memória externa | A saída da ALU 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 a versão de sinal extendido dos 16 bits menos significativos do IR |
| | 11 | A segunda entrada da ALU 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. |
| 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 Multicycle*

A arquitectura multicycle do MIPS adopta um ciclo de instrução composto por cinco passos distintos, cada uma deles executado em 1 ciclo de relógio:

1. *Instruction fetch* e cálculo de PC+4
2. *Instruction decode, register fetch* e cálculo do *branch target address*
3. Execução da instrução (instruções tipo R) ou cálculo do endereço de memória (instr. acesso à memória) ou conclusão da instrução *branch*
4. Acesso à memória (LW) ou conclusão das instruções SW ou tipo R
5. Write-back (conclusão da instrução de leitura da memória)

A distribuição das operações por estes 5 passos tenta distribuir equitativamente o trabalho a realizar em cada ciclo. Assim, estes passos reflectem o pressuposto de que durante um ciclo de relógio apenas seja possível efectuar uma das seguintes operações:

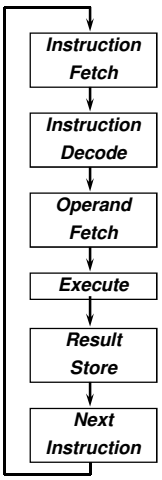
- Um acesso à memória externa (escrita ou leitura)
- Um acesso ao *file register* (escrita ou leitura)
- Uma operação da ALU



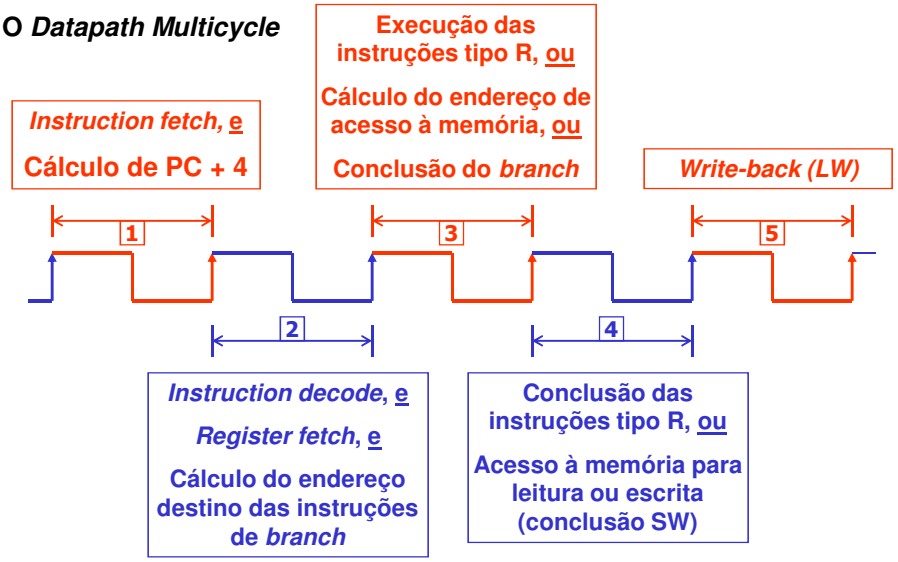


## O *Datapath Multicycle*

Este modelo multi ciclo corresponde (com algumas adaptações) a um diagrama de fluxo que já observámos na segunda aula!



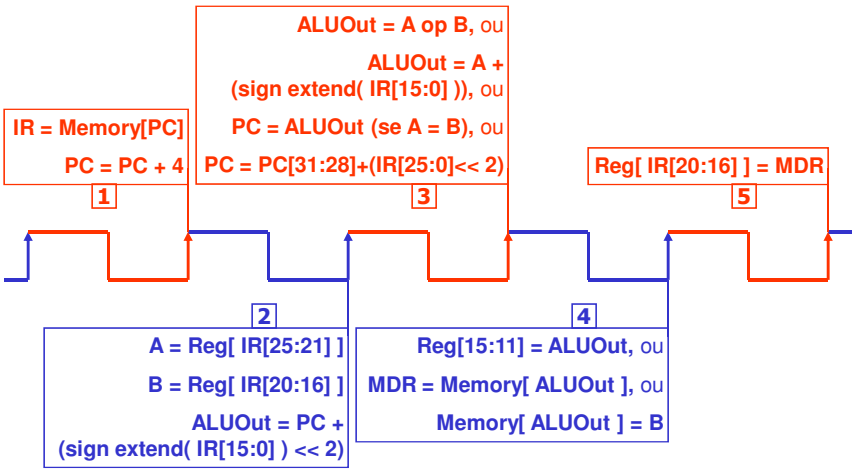
## O *Datapath Multicycle*








## O *Datapath Multicycle*



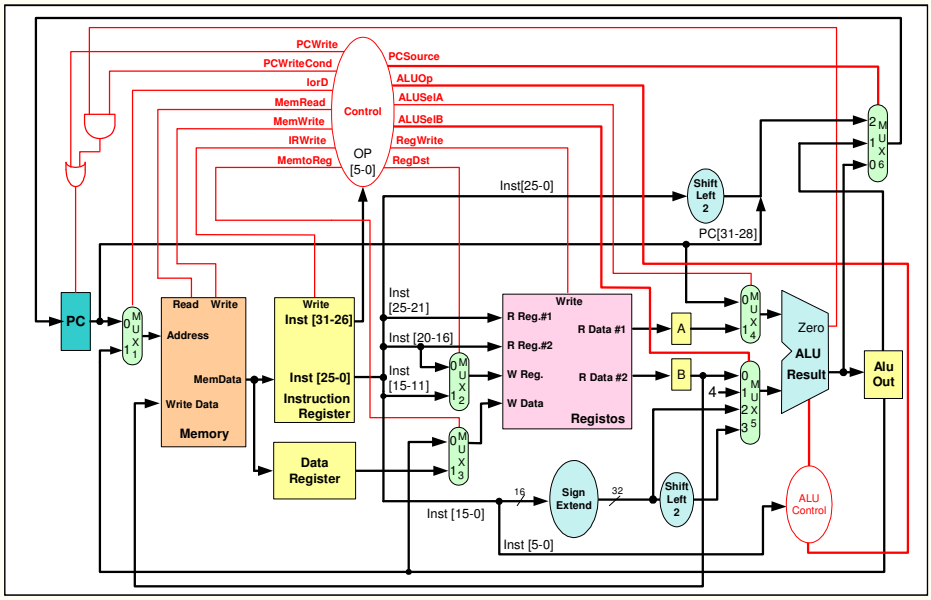
## O *Datapath Multicycle*

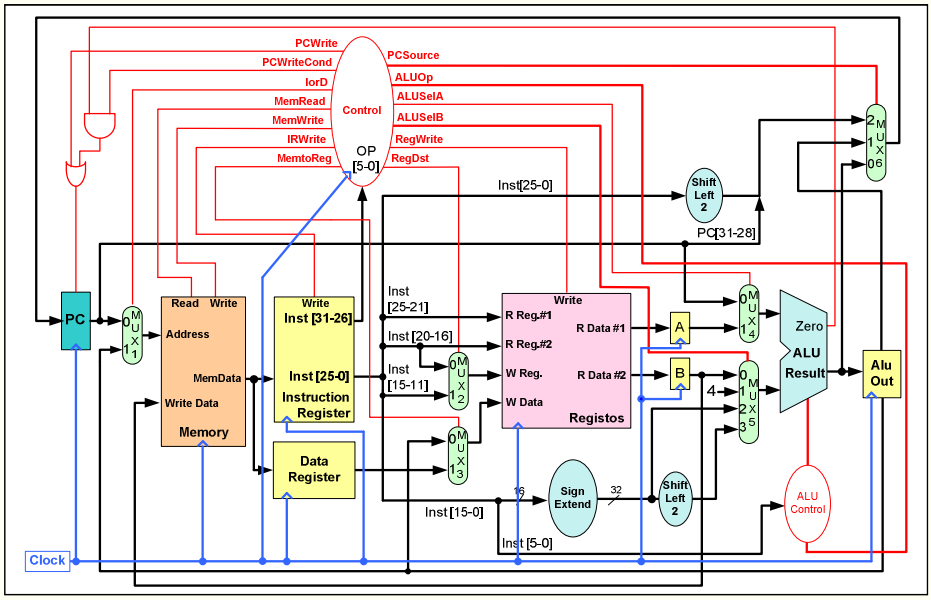
As operações realizadas no final (transição activa do relógio) de cada um dos cinco passos:

| Passo | Acção p/ as R-Type | Acção p/ instruções que referenciam a memória | Acção p/ os branches |
|---|---|---|---|
| Instruction fetch | IR = Memory[PC]<br>PC = PC + 4 | | |
| Instruction decode/register fetch | A = Reg[ IR[25:21] ]<br>B = Reg[ IR[20:16] ]<br>ALUOut = PC + (sign extended( IR[15:0] ) << 2) | | |
| Execução, cálculo de endereços e conclusão dos branches | ALUOut = A op B | ALUOut = A + sign-extended( IR[15:0]) ) | If (A == B) then<br>PC = ALUOut |
| Acesso à memória ou conclusão das instruções tipo R | Reg[ IR[15:11] ] = ALUOut | MDR = Memory[ALUOut] ou<br>Memory[ALUOut] = B | |
| Write-back | | 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.

O significado atribuído a cada cor é o seguinte:

Sinais de controlo:

- vermelho → 0
- verde → diferente de zero
- cinzento → "*don't care*"

Barramentos:

- azul → Activos no contexto da instrução
- preto → Não activo no contexto da instrução

Elementos de estado:

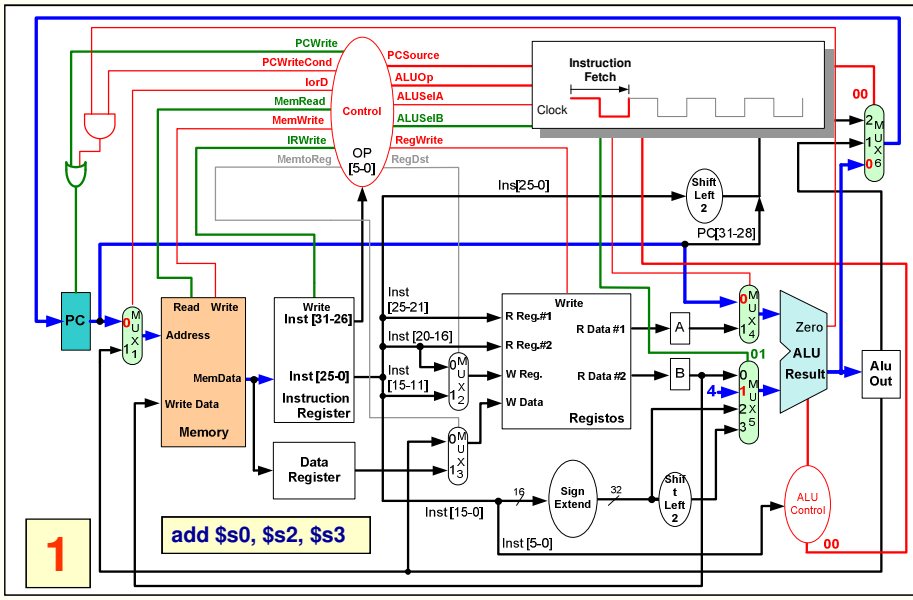
- fundo branco → Não usados no contexto da instrução
- fundo de cor → Usados no contexto da instrução

# Exemplo 1

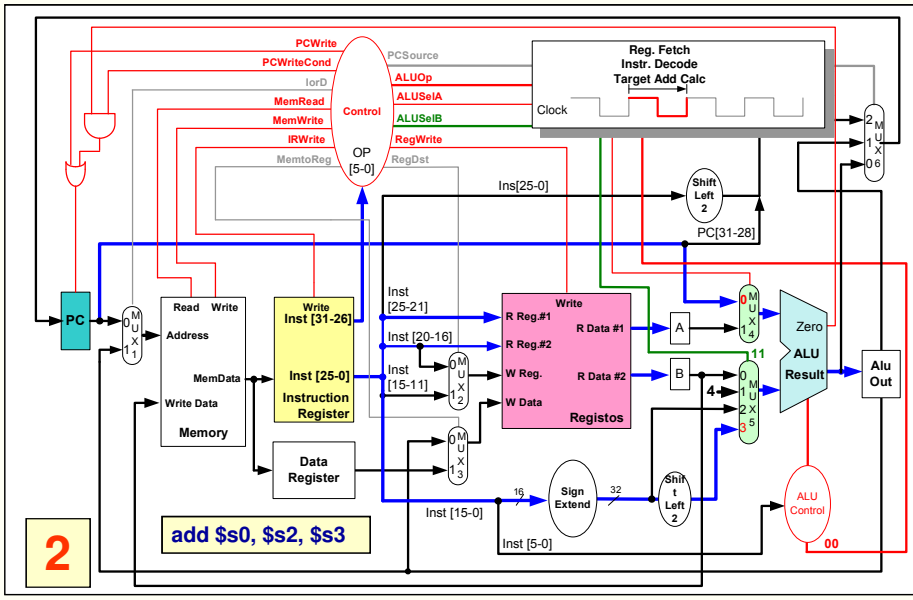
# Funcionamento do *datapath* nas instruções do tipo R



Arquitectura de Computadores I 2007/08

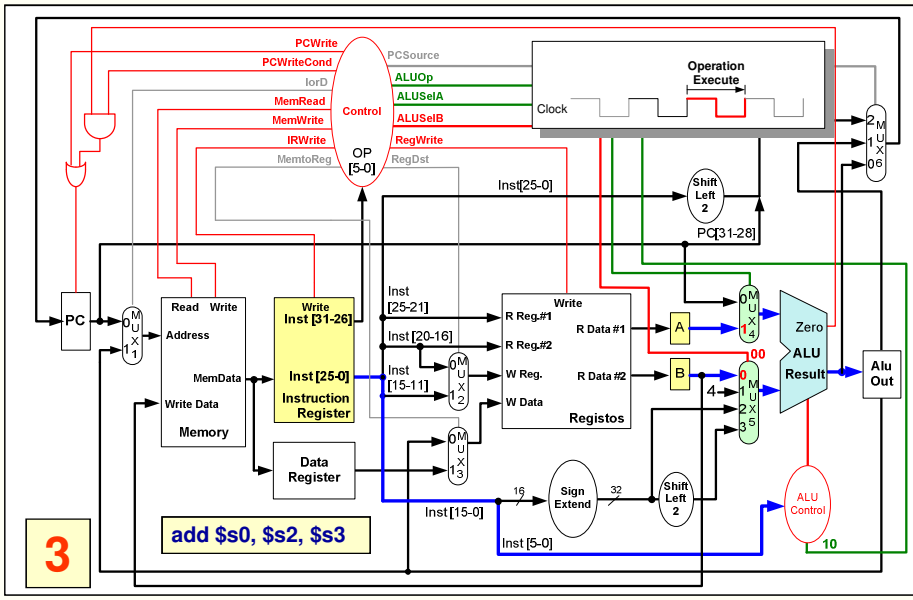


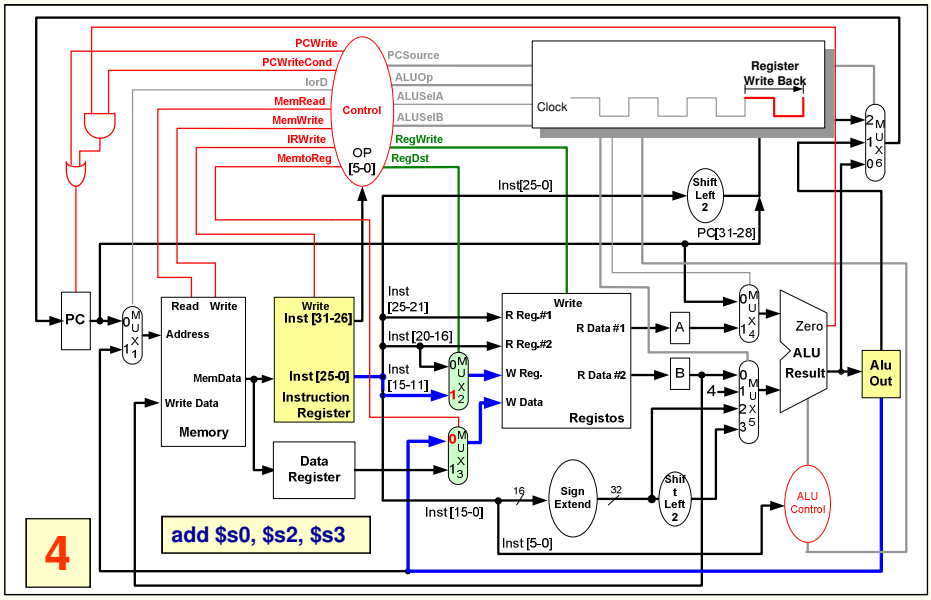

Universidade de Aveiro Slide 19&20 - 26


PCWrite
PCWriteCond
IorD
MemRead
MemWrite
IRWrite
MemtoReg
Control
OP [5-0]
PCSource
ALUOp
ALUSelA
ALUSelB
RegWrite
RegDst
Reg. Fetch
Instr. Decode
Target Add Calc
Clock
Ins[25-0]
Shift Left 2
PC[31-28]
PC
Read Write
Address
MemData
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]
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
Sign Extend
16
32
Shift Left 2
ALU Control
00
Inst [5-0]
2
add $s0, $s2, $s3






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

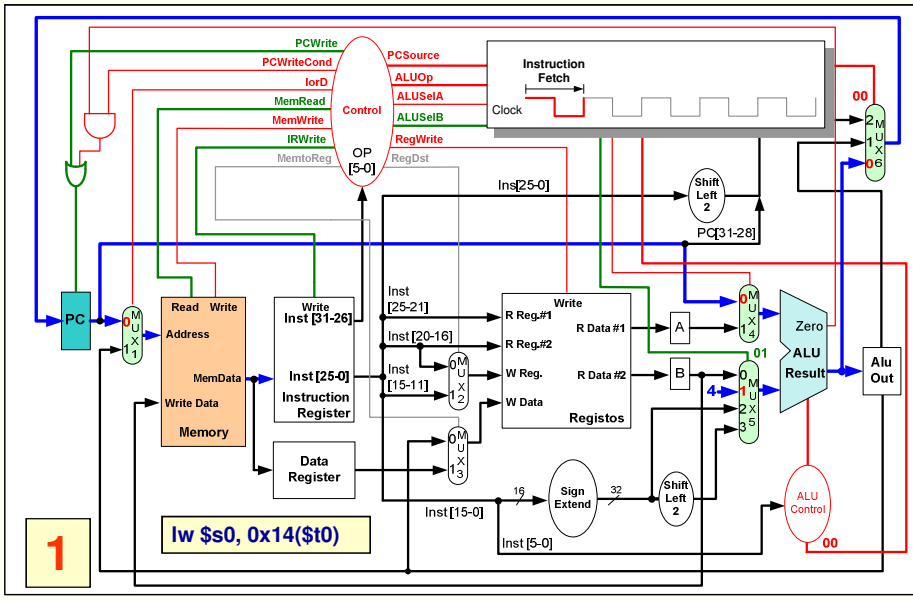






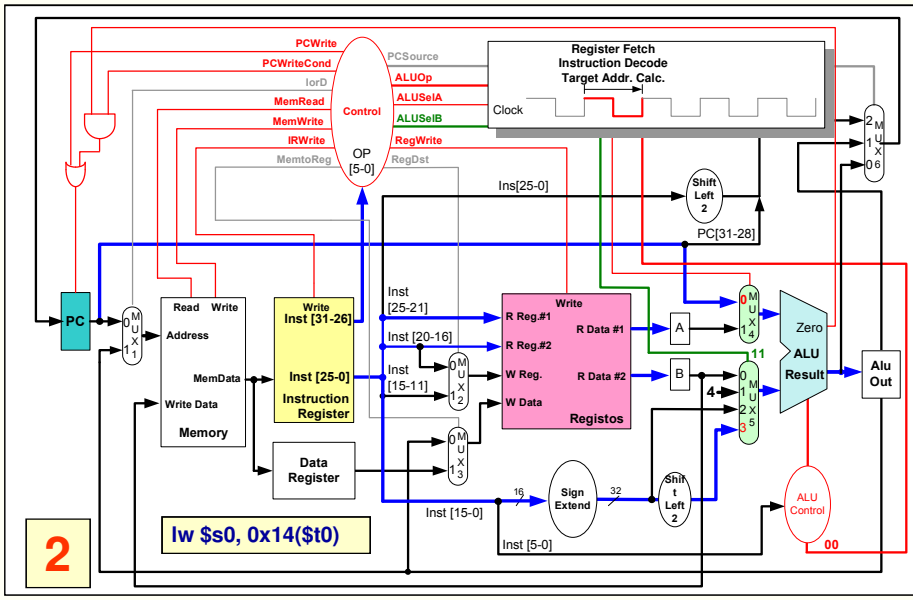

# Exemplo 2

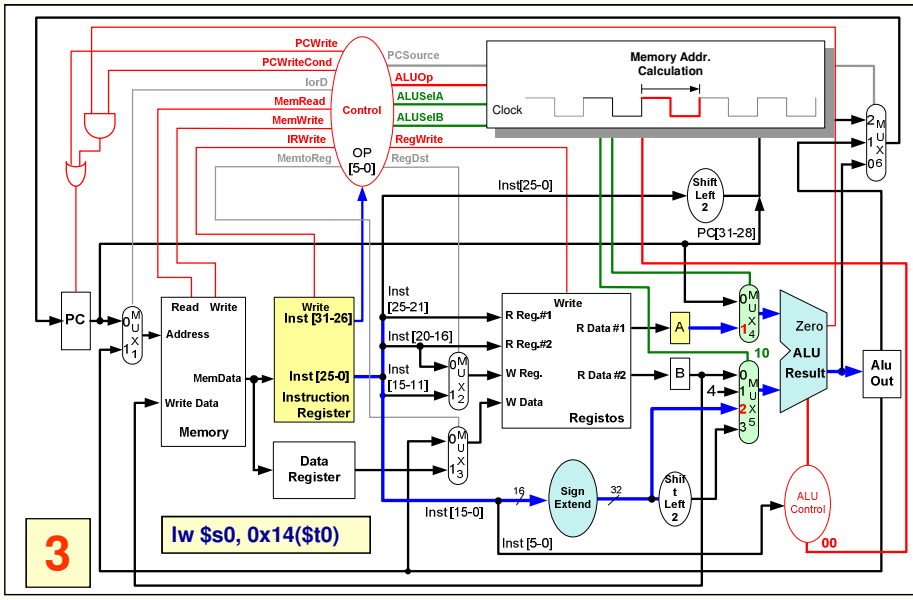
## Funcionamento do *datapath* na instrução *load word* ( "lw" )


PCWrite
PCWriteCond
IorD
MemRead
MemWrite
IRWrite
MemtoReg
PCSource
ALUOp
ALUSelA
ALUSelB
RegWrite
RegDst
Control
OP [5-0]
Instruction Fetch
Clock
Ins[25-0]
Shift Left 2
PC[31-28]
PC
Read Write
Address
MemData
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
ALU Control
Sign Extend
16
32
00
01
lw $s0, 0x14($t0)
1



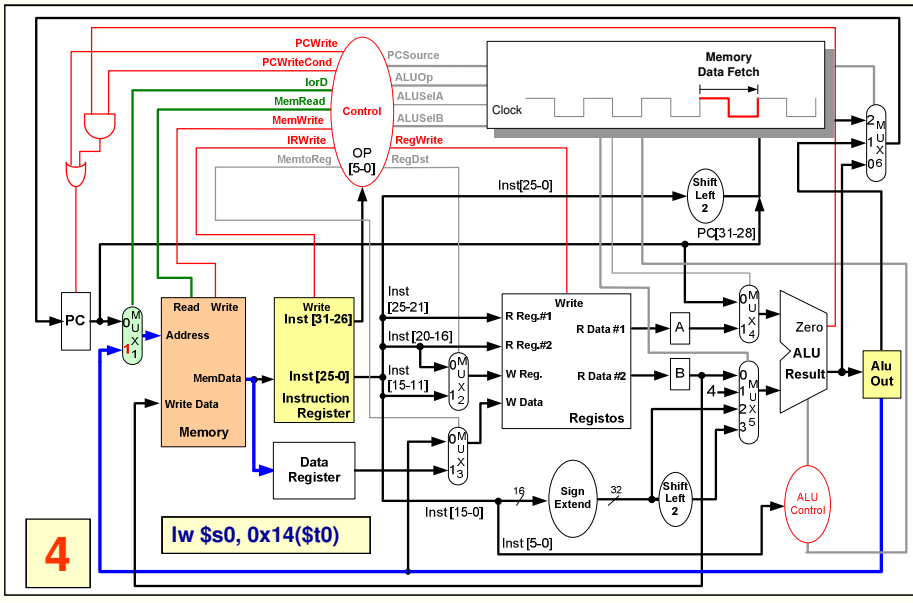


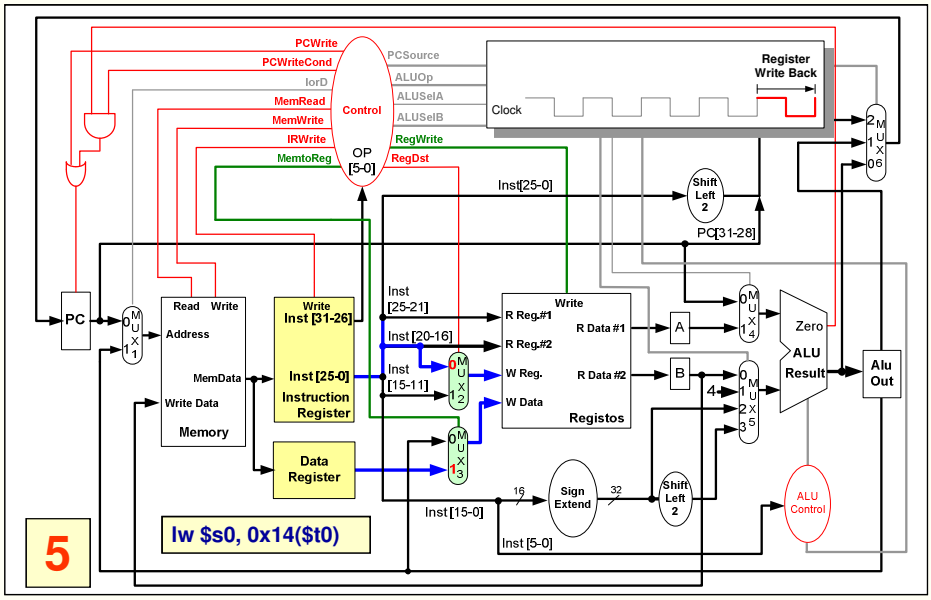

PCWrite
PCWriteCond
IorD
MemRead
MemWrite
IRWrite
MemtoReg
PCSource
ALUOp
ALUSelA
ALUSelB
RegWrite
RegDst
Control
OP [5-0]
Register Fetch
Instruction Decode
Target Addr. Calc.
Clock
Ins[25-0]
Shift Left 2
PC[31-28]
PC
Read Write
Address
MemData
Write Data
Memory
Write Inst [31-26]
Inst [25-0]
Instruction 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
ALU Control
Sign Extend
16
32
00
11
lw $s0, 0x14($t0)
2






Arquitectura de Computadores I 2007/08



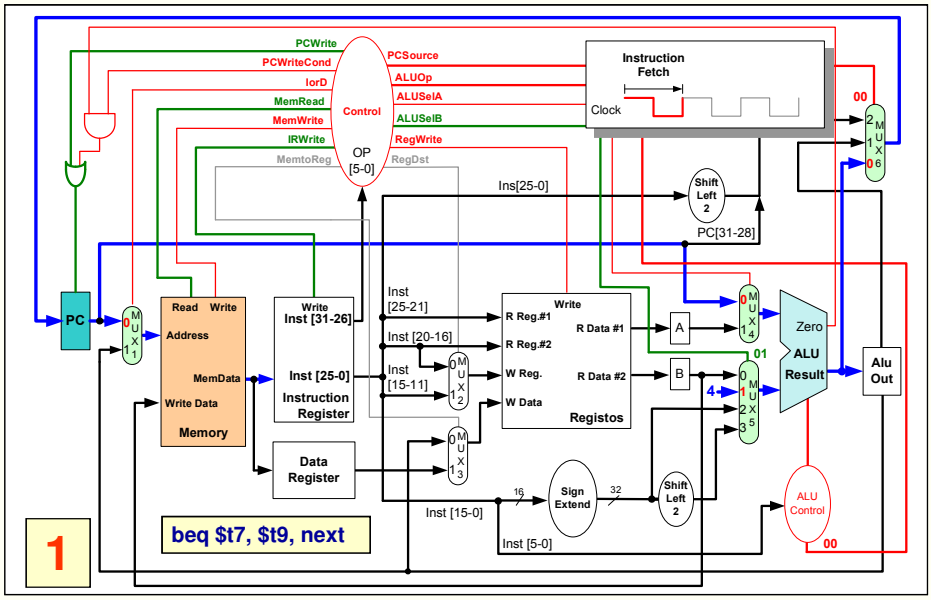






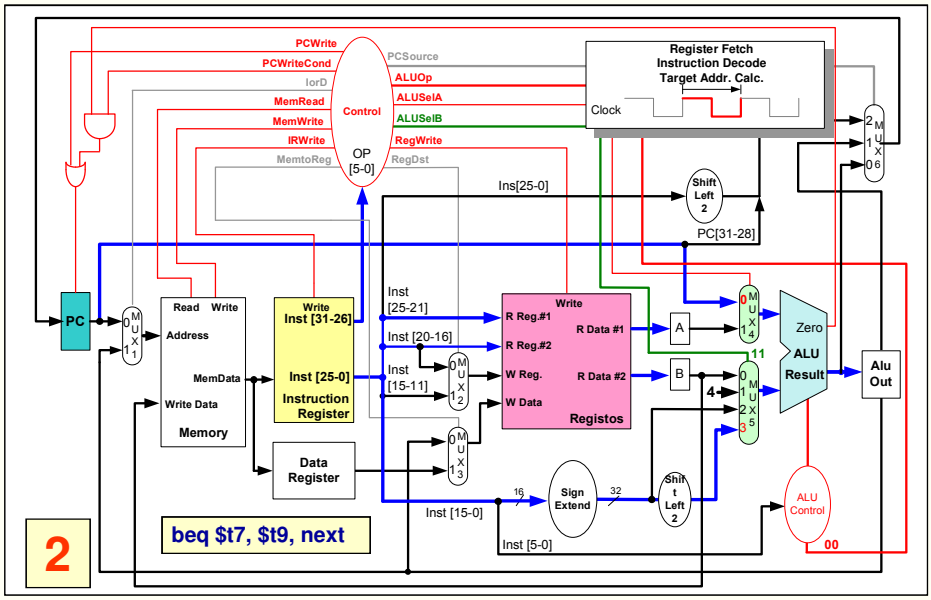

# Exemplo 3

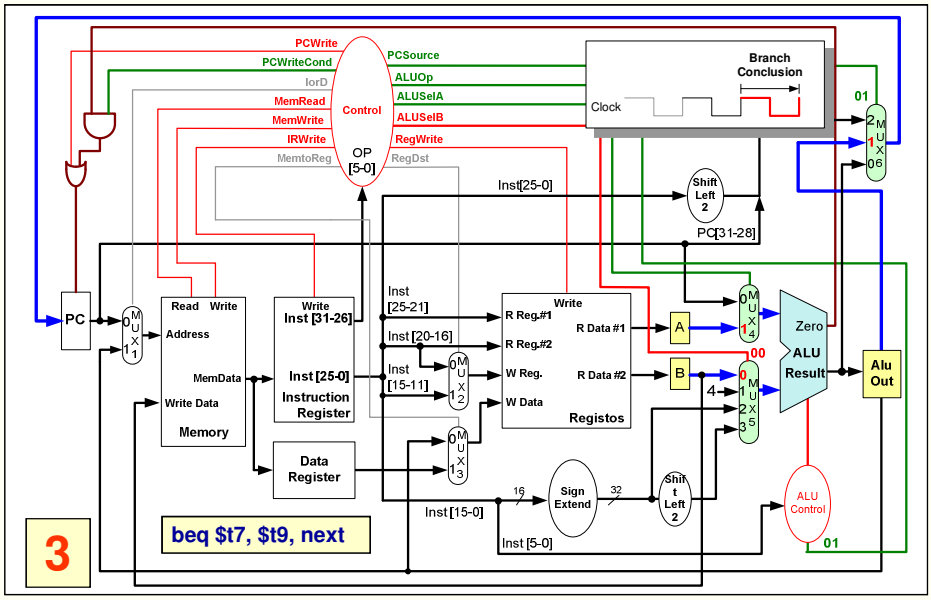
## Funcionamento do *datapath* na instrução *branch if equal* ( "beq" )


PCWrite
PCWriteCond
IorD
MemRead
MemWrite
IRWrite
MemtoReg
Control
OP [5-0]
PCSource
ALUOp
ALUSelA
ALUSelB
RegWrite
RegDst
Instruction Fetch
Clock
Ins[25-0]
Shift Left 2
PC[31-28]
PC
Read Write
Address
MemData
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
ALU Control
Sign Extend
16
32
00
01
1
beq $t7, $t9, next






PCWrite
PCWriteCond
IorD
MemRead
MemWrite
IRWrite
MemtoReg
Control
OP [5-0]
PCSource
ALUOp
ALUSelA
ALUSelB
RegWrite
RegDst
Register Fetch
Instruction Decode
Target Addr. Calc.
Clock
Ins[25-0]
Shift Left 2
PC[31-28]
PC
Read Write
Address
MemData
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
ALU Control
Sign Extend
16
32
00
11
2
beq $t7, $t9, next

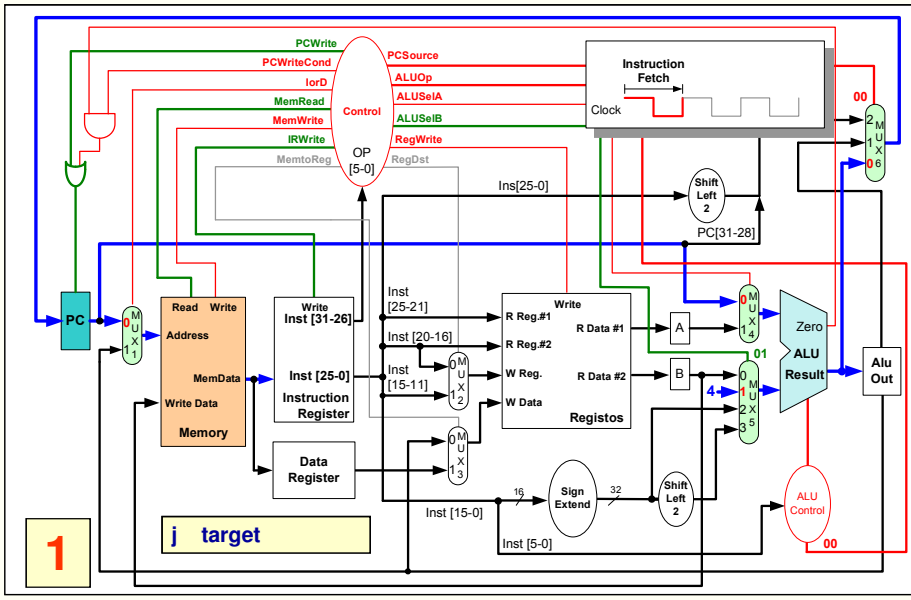






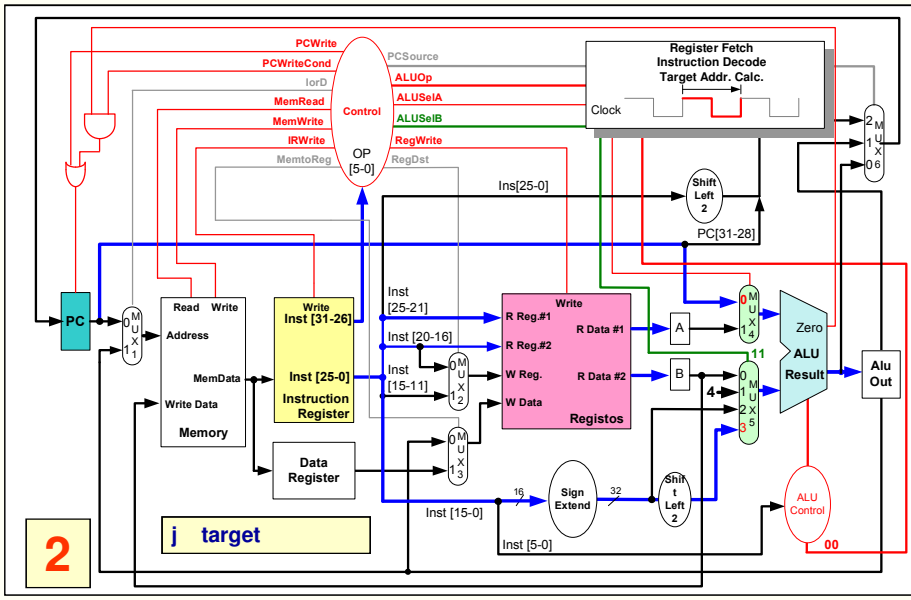

# Exemplo 4

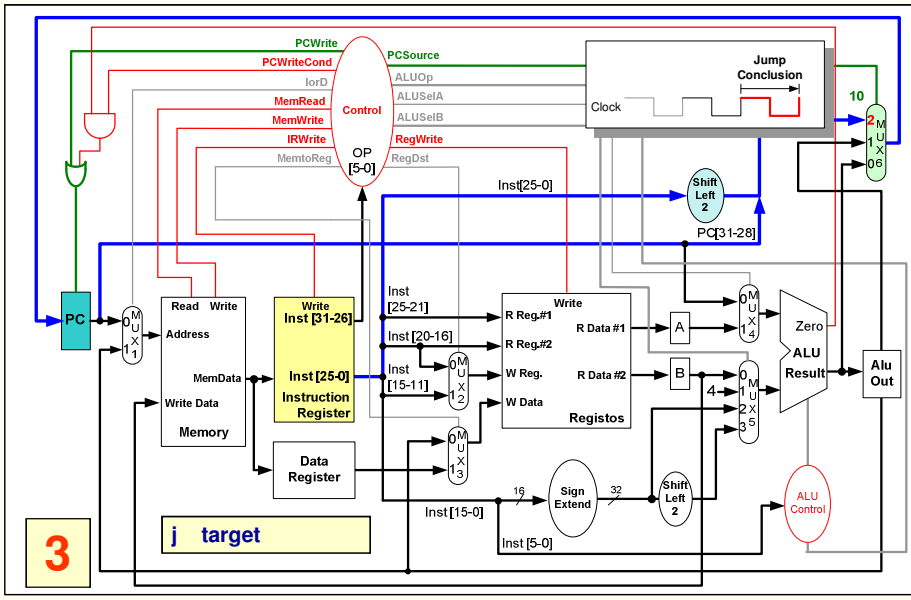
## Funcionamento do *datapath* na instrução de salto incondicional ( "j" )


PCWrite
PCWriteCond
IorD
MemRead
MemWrite
IRWrite
MemtoReg
Control
OP [5-0]
PCSource
ALUOp
ALUSelA
ALUSelB
RegWrite
RegDst
Instruction Fetch
Clock
00
Ins[25-0]
Shift Left 2
PC[31-28]
PC
Read Write
Address
MemData
Write Data
Memory
Write Inst [31-26]
Inst [25-0]
Instruction Register
Inst [25-21]
Inst [20-16]
Inst [15-11]
Inst [15-0]
Data Register
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
Inst [5-0]
1
j target






PCWrite
PCWriteCond
IorD
MemRead
MemWrite
IRWrite
MemtoReg
Control
OP [5-0]
PCSource
ALUOp
ALUSelA
ALUSelB
RegWrite
RegDst
Register Fetch
Instruction Decode
Target Addr. Calc.
Clock
Ins[25-0]
Shift Left 2
PC[31-28]
PC
Read Write
Address
MemData
Write Data
Memory
Write Inst [31-26]
Inst [25-0]
Instruction Register
Inst [25-21]
Inst [20-16]
Inst [15-11]
Inst [15-0]
Data Register
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
Inst [5-0]
2
j target


PCWrite
PCWriteCond
IorD
MemRead
MemWrite
IRWrite
MemtoReg
Control
OP
[5-0]
PCSource
ALUOp
ALUSelA
ALUSelB
RegWrite
RegDst
Clock
Jump
Conclusion
10
Inst[25-0]
Shift
Left
2
PC[31-28]
PC
Read Write
Address
MemData
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
16
Sign
Extend
32
Shift
Left
2
ALU
Control
Inst [15-0]
Inst [5-0]
3
j target