Arquitectura de Computadores I 2007/08

1º Semestre de 2007/2008

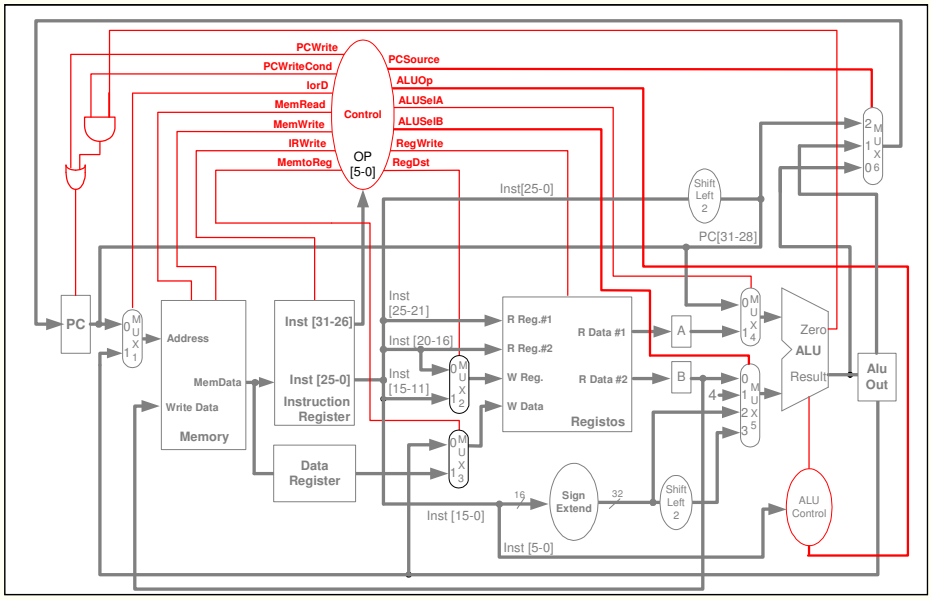
Bernardo Cunha, José Luís Azevedo, Arnaldo Oliveira

Arquitectura de Computadores I 2007/08

Aula 21

Unidade de controlo para múltiplos ciclos de relógio por instrução:
- Revisão do modelo de máquinas de estados finitos;
- Diagrama de estados da unidade de controlo

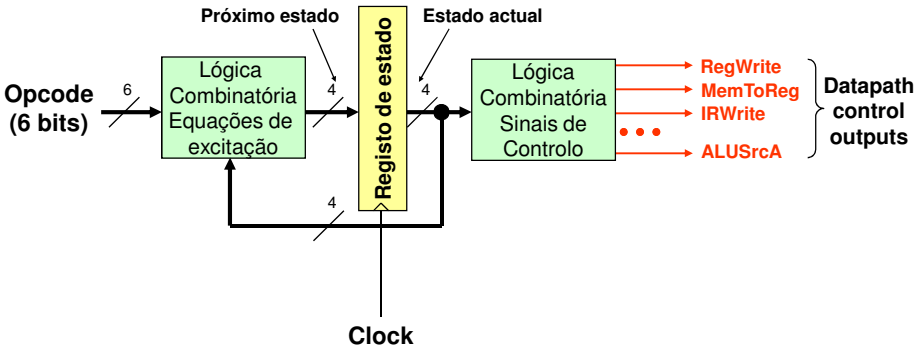
Excepções e interrupções:
- Gestão de excepções e interrupções;
- A unidade de controlo e a gestão de excepções



A unidade de controlo do *datapath* multi-cycle

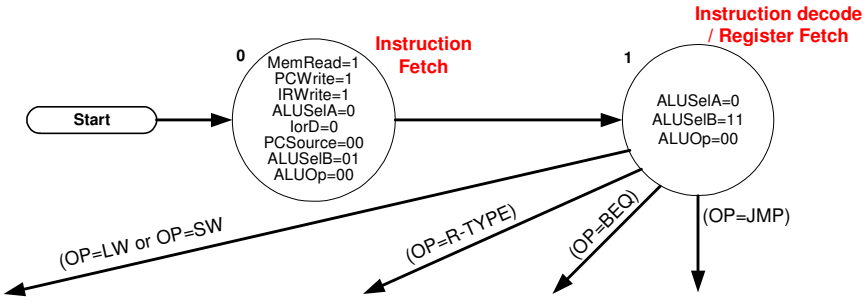
- No *datapath single cycle*, cada instrução era executada num único ciclo de relógio. Consequentemente, a unidade de controlo é responsável pela geração de um conjunto de sinais que não se alteram durante a execução de cada instrução. A relação entre os sinais de controlo e o código de operação pode assim ser gerado por um circuito meramente combinatório.
- Em contrapartida, no *datapath multi-cycle*, 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* multi-cycle



A unidade de controlo do *datapath* multi-cycle

Como víramos já, 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.

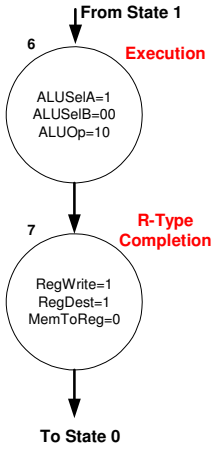


O segundo estado, por sua vez, tem quatro saídas distintas, dependendo do valor do campo OP da instrução.

A unidade de controlo do *datapath* multi-cycle

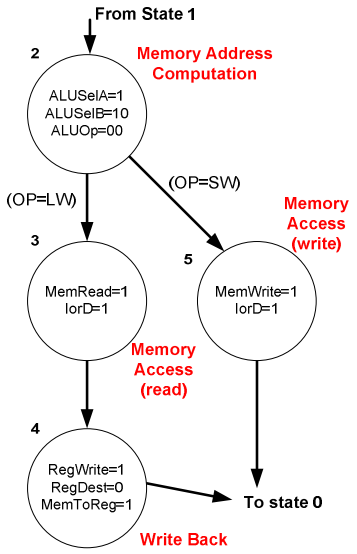
Nas instruções do tipo "*R*", são necessários mais dois estados:

- Um para controlar a execução da operação aritmética ou lógica
- Outro para escrever o resultado no registo destino.



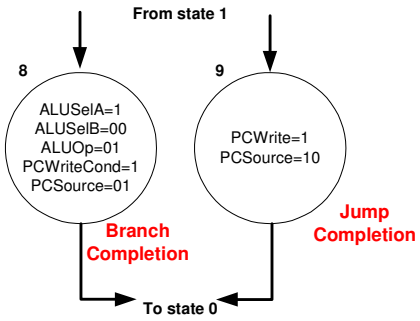
A unidade de controlo do *datapath* multi-cycle

- Nas instruções do tipo "*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.

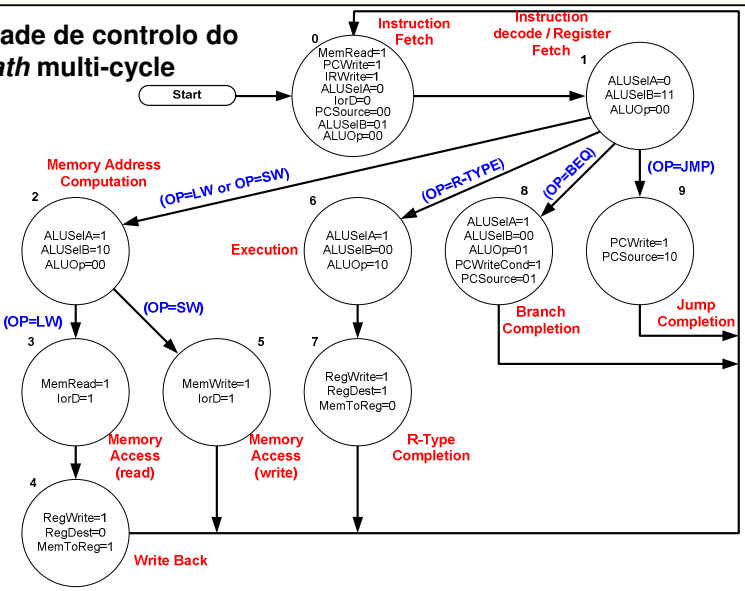


A unidade de controlo do *datapath* multi-cycle

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* multi-cycle



A unidade de controlo do *datapath* multi-cycle

- A unidade de controlo que acabamos de desenhar tem apenas 10 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.

Excepções e *interrupts*

O que são excepções?

São eventos que, não sendo *branches* ou *jumps*, alteram o fluxo normal de execução das instruções. Existem duas fontes distintas de excepções:

1. Eventos internos ao CPU, inesperados e decorrentes da execução das próprias instruções; o *overflow* ou o *fetch* de uma instrução com um OPCODE desconhecido para a unidade de controlo são dois exemplos de **excepções**.
2. Eventos gerados externamente ao CPU e que surgem assincronamente com o funcionamento deste; estes eventos são normalmente designados por ***interrupts***. Tal como as excepções provocam uma alteração do fluxo de execução das instruções. A fonte dos *interrupts* é normalmente um dispositivo de I/O que necessita de ser atendido.

Excepções e *interrupts*

Alguns eventos deste tipo podem ser observados na tabela seguinte.

Tipo de evento	Proveniência	Terminologia MIPS
Pedido de atenção por um dispositivo de I/O	Externo	*Interrupt*
Invocação do sistema operativo a partir do programa	Interno	Excepção
***Overflow* aritmético**	**Interno**	**Excepção**
Tentativa de execução de uma instrução desconhecida	**Interno**	**Excepção**
Problema de funcionamento do hardware	Qualquer	Excepção ou *interrupt*

A detecção das condições de excepção e as acções que daí decorrem podem condicionar a eficiência do CPU ao influenciar directamente o período do sinal de relógio. É assim fundamental considerar estas condições desde a fase inicial do projecto da unidade de controlo.

Excepções – operações básicas efectuadas pelo μP

Há duas operações básicas que têm de ser efectuadas pela máquina sempre que ocorre uma excepção:

1. Copiar o endereço da instrução que gerou a excepção para um registo a isso destinado. No MIPS esse registo chama-se **EPC** (*Exception Program Counter*)
2. Transferir o controlo para a rotina de tratamento da excepção (carregando no "program counter" um endereço previamente definido).

Excepções – processamento na rotina de atendimento

A rotina de tratamento da excepção poderá realizar uma de várias tarefas possíveis:

- Se a excepção corresponde a um evento que compromete a continuação da execução do programa do utilizador, então deverá terminar a execução do mesmo (e.g. acesso a um endereço não alinhado)
- Providenciar um serviço ao programa do utilizador (*syscall*)
- Tomar uma medida de reparação do evento que gerou a excepção (quando seja possível)
- Retornar ao programa do utilizador, caso a excepção não seja crítica, usando para isso a informação guardada no EPC.

Excepções – Identificação da origem / causa

Torna-se necessário que a rotina de tratamento da excepção, para além de conhecer o endereço da instrução que gerou a excepção, possa determinar igualmente a causa dessa mesma excepção, por forma a agir em conformidade.

Os dois mecanismos mais utilizados para permitir esse conhecimento são:

- A existência de um registo especial no qual é armazenado um código que identifica a causa da excepção ou do *interrupt*. Este é o mecanismo usado pelo MIPS. O registo em causa é designado ***Cause Register***.
- A vectorização das excepções. Nesta estratégia, cada tipo de excepção (ou *interrupt*) desencadeia a passagem do controlo do programa para um endereço distinto.

Excepções – integração no *datapath* multiciclo do MIPS

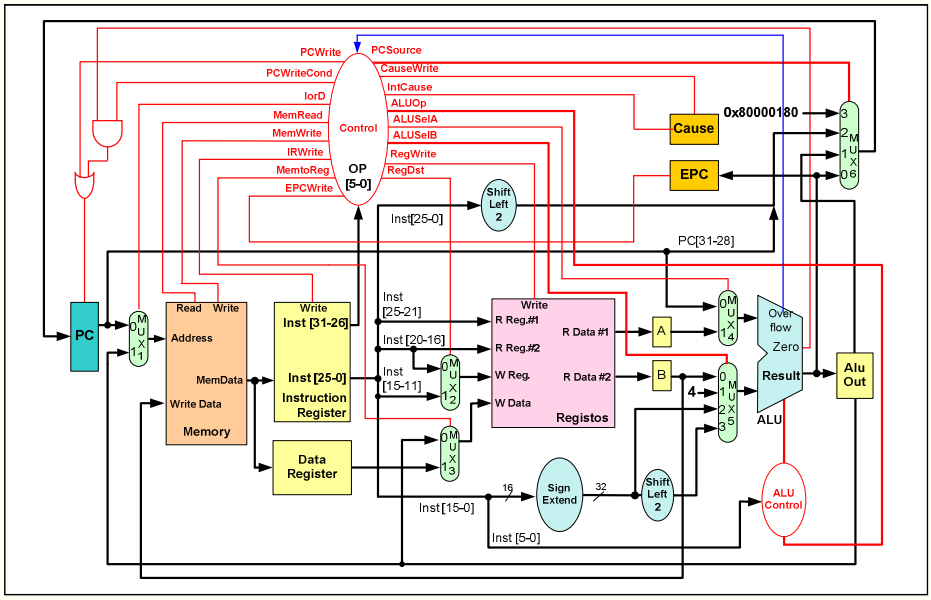
Na análise que iremos fazer a seguir, vamos apenas considerar dois tipos de excepções (os mecanismos para atendimento de *interrupts* é semelhante):

- **Excepções geradas pela tentativa de execução de uma instrução inválida (campo OPCODE desconhecido da unidade de controlo)**
- **Excepções geradas pela ocorrência de *overflow* na execução de uma operação aritmética**

Excepções – integração no *datapath* multiciclo do MIPS

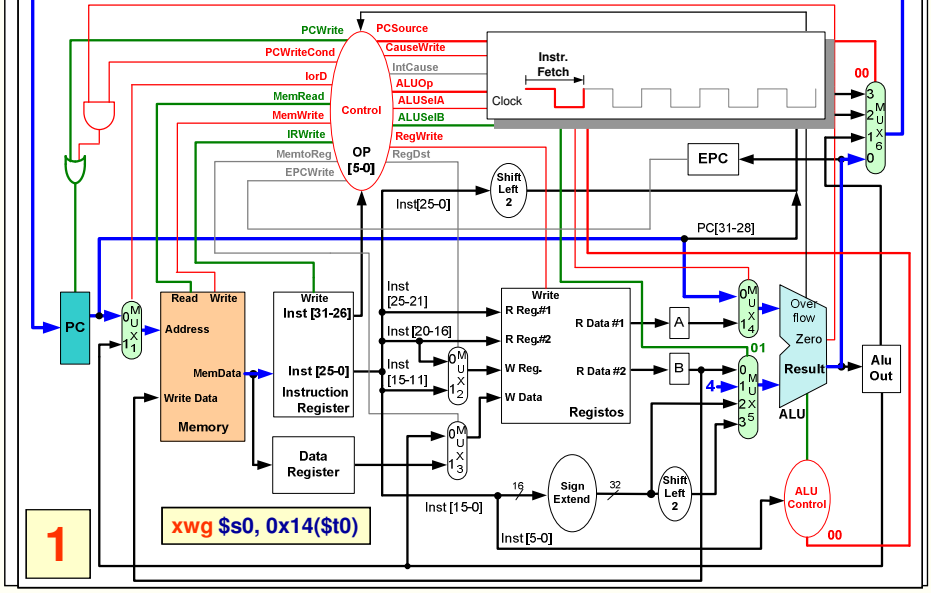
Para suportar a gestão dos dois tipos de excepção indicados, o nosso *datapath* multi-ciclo precisa de ser ligeiramente alterado. Essas alterações podem ser sintetizadas da seguinte maneira:

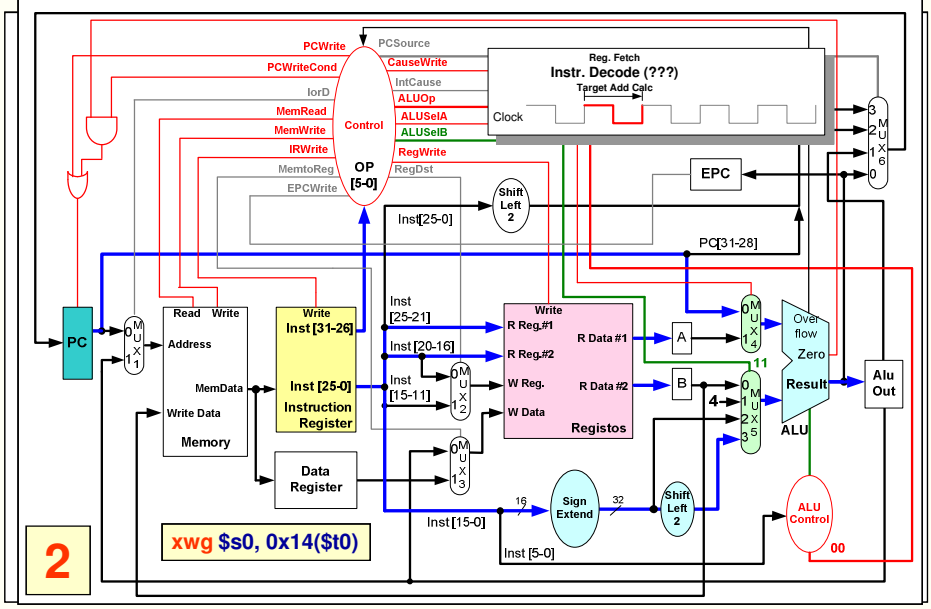
- Criação de dois novos registos – *EPC* e *Cause Register*. No caso deste último, apenas um bit será usado para identificar a causa da excepção. O valor "0" indicará uma instrução desconhecida enquanto o valor "1" indicará a ocorrência de um *overflow*.
- Criação de um conjunto suplementar de sinais de controlo que permitam operar sobre aqueles dois registos.
- Criação de um mecanismo que permita escrever no PC o endereço da primeira instrução da rotina de atendimento à excepção.

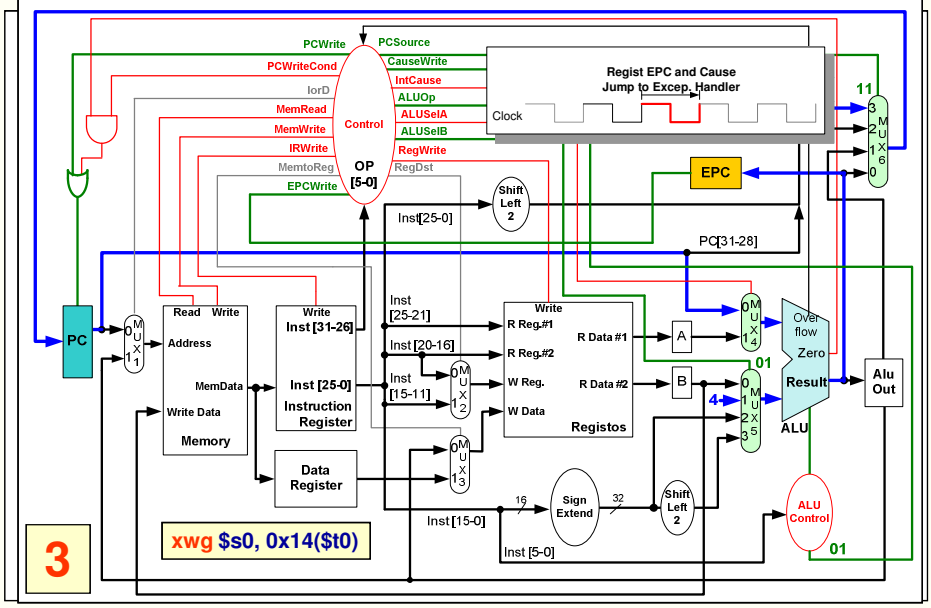


Exemplo 1

Funcionamento do *datapath* na situação em que o código da instrução lida é desconhecido

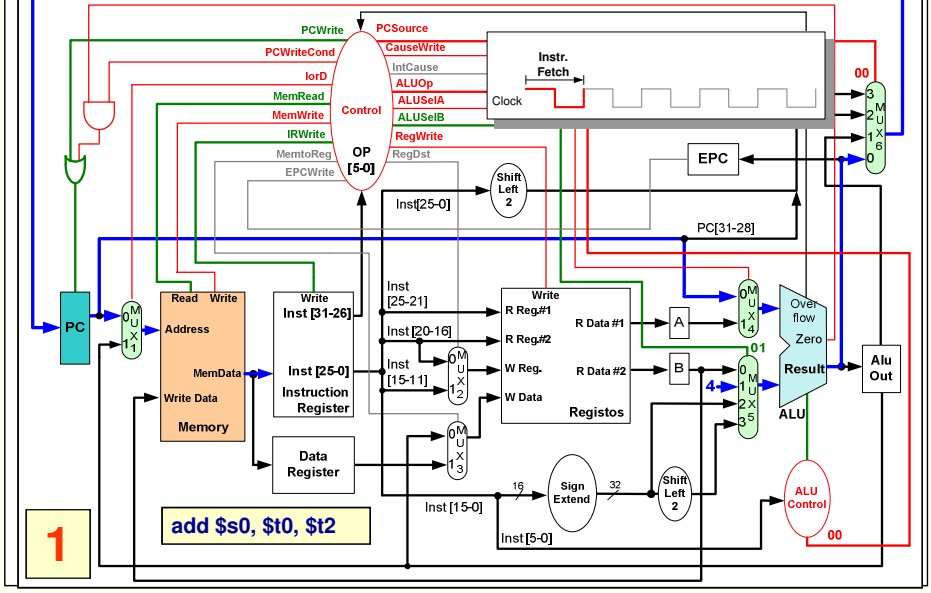

PCWrite
PCWriteCond
IorD
MemRead
MemWrite
IRWrite
MemtoReg
EPCWrite
PCSource
CauseWrite
IntCause
ALUOp
ALUSelA
ALUSelB
RegWrite
RegDst
Control
OP [5-0]
Instr. Fetch
Clock
EPC
Shift Left 2
Inst[25-0]
PC[31-28]
PC
Read Write
Address
MemData
Write Data
Memory
Write Inst [31-26]
Inst [25-0] Instruction Register
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
Over flow
Zero
Result
ALU
Alu Out
Sign Extend
16
32
Shift Left 2
ALU Control
00
01
1
xwg $s0, 0x14($t0)

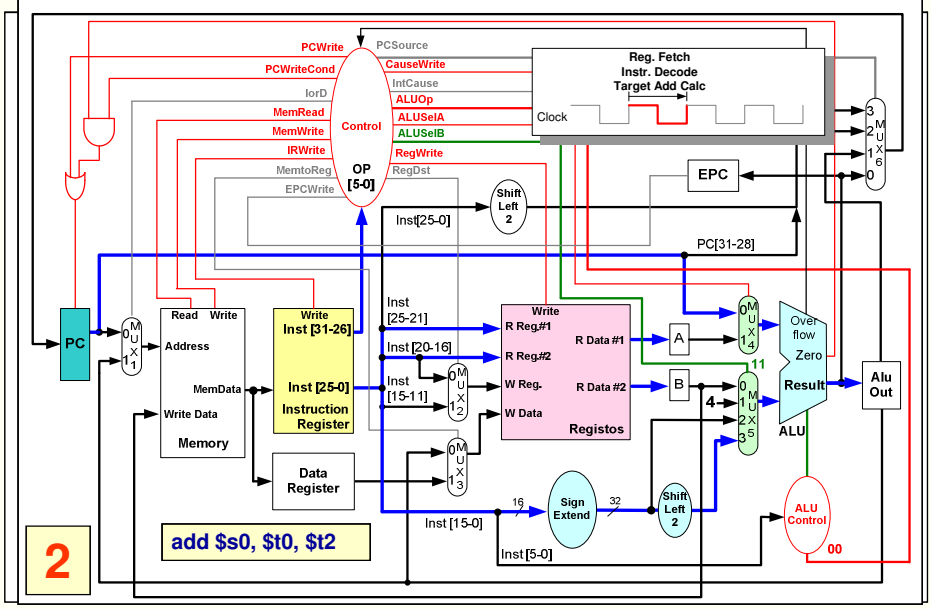

PCWrite
PCWriteCond
IorD
MemRead
MemWrite
IRWrite
MemtoReg
EPCWrite
PCSource
CauseWrite
IntCause
ALUOp
ALUSelA
ALUSelB
RegWrite
RegDst
Control
OP [5-0]
Reg. Fetch
Instr. Decode (???)
Target Add Calc
Clock
EPC
Shift Left 2
Inst[25-0]
PC[31-28]
PC
Read Write
Address
MemData
Write Data
Memory
Write Inst [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
Over flow
Zero
Result
ALU
Alu Out
Sign Extend
16
32
Shift Left 2
ALU Control
00
11
2
xwg $s0, 0x14($t0)

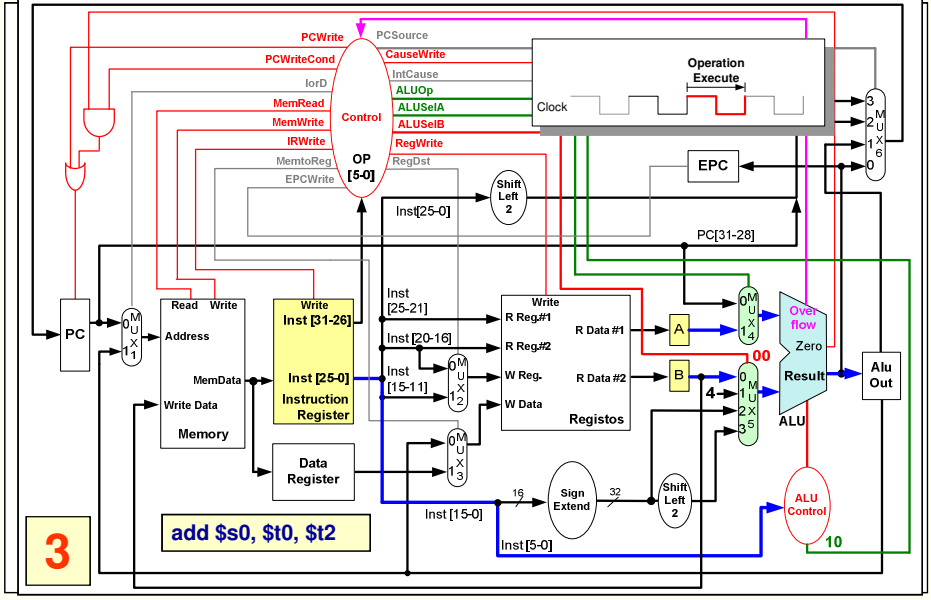


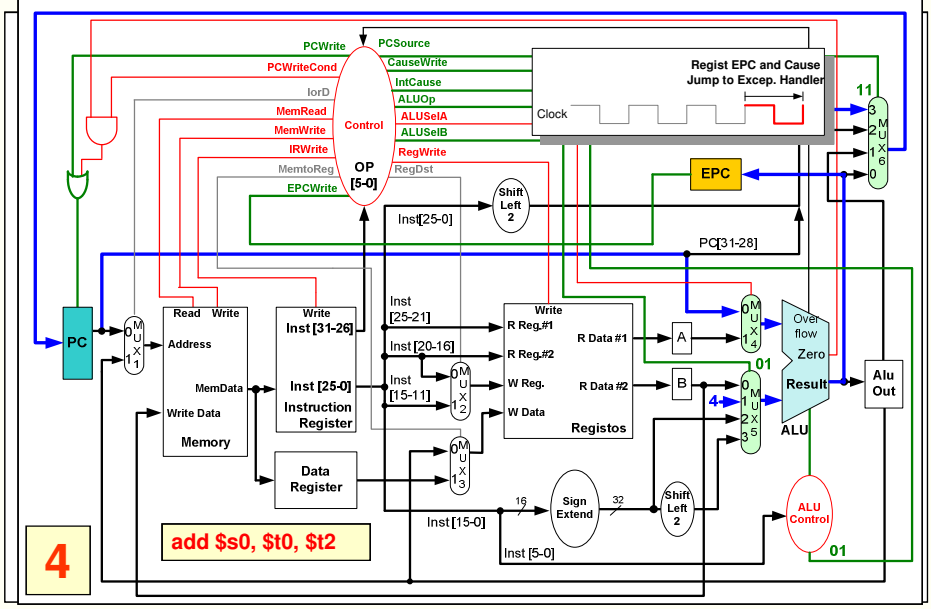
Exemplo 2

Funcionamento do *datapath* na situação em que uma operação aritmética gera *overflow*


PCWrite
PCWriteCond
IorD
MemRead
MemWrite
IRWrite
MemtoReg
EPCWrite
PCSource
CauseWrite
IntCause
ALUOp
ALUSelA
ALUSelB
RegWrite
RegDst
Control
OP [5-0]
Instr. Fetch
Clock
EPC
Shift Left 2
Inst[25-0]
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
Over flow
Zero
Result
ALU
Alu Out
ALU Control
Inst [15-0]
16
Sign Extend
32
Shift Left 2
Inst [5-0]
add $s0, $t0, $t2
1


PCWrite
PCWriteCond
IorD
MemRead
MemWrite
IRWrite
MemtoReg
EPCWrite
PCSource
CauseWrite
IntCause
ALUOp
ALUSelA
ALUSelB
RegWrite
RegDst
Control
OP [5-0]
Reg. Fetch
Instr. Decode
Target Add Calc
Clock
EPC
Shift Left 2
Inst[25-0]
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
Over flow
Zero
Result
ALU
Alu Out
ALU Control
Inst [15-0]
16
Sign Extend
32
Shift Left 2
Inst [5-0]
add $s0, $t0, $t2
2

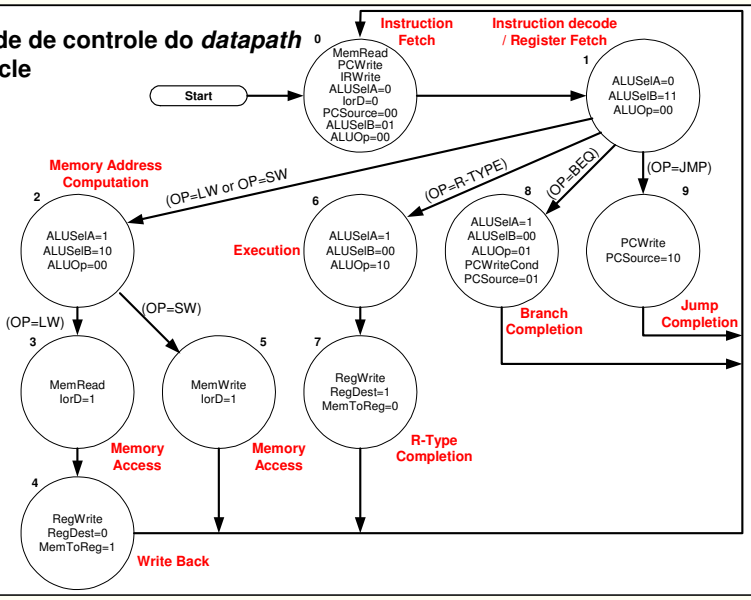

PCWrite
PCWriteCond
IorD
MemRead
MemWrite
IRWrite
MemtoReg
EPCWrite
PCSource
CauseWrite
IntCause
ALUOp
ALUSelA
ALUSelB
RegWrite
RegDst
Control
OP [5-0]
Operation Execute
Clock
EPC
Shift Left 2
Inst[25-0]
PC[31-28]
PC
Read Write
Address
MemData
Write Data
Memory
Write Inst [31-26]
Inst [25-0]
Instruction 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
Over flow
Zero
Result
ALU
Alu Out
Sign Extend
16
32
Shift Left 2
ALU Control
00
10
3
add $s0, $t0, $t2
Universidade de Aveiro
Slide 21 - 27


PCWrite
PCWriteCond
IorD
MemRead
MemWrite
IRWrite
MemtoReg
EPCWrite
PCSource
CauseWrite
IntCause
ALUOp
ALUSelA
ALUSelB
RegWrite
RegDst
Control
OP [5-0]
Regist EPC and Cause Jump to Excep. Handler
Clock
EPC
11
Shift Left 2
Inst[25-0]
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
01
Over flow
Zero
Result
ALU
Alu Out
Sign Extend
16
32
Shift Left 2
ALU Control
01
4
add $s0, $t0, $t2
Universidade de Aveiro
Slide 21 - 28

A unidade de controlo com suporte para as excepções

- A unidade de controlo tem também de ser redesenhada para acomodar a gestão das excepções e a geração dos respectivos sinais de controlo.
- Como pudemos observar no datapath, a gestão dos dois tipos de excepção considerados nestes exemplos, carecem apenas de um ciclo de relógio suplementar cada um.
- Ao nível do diagrama de estados precisaremos portanto de dois estados suplementares.

A unidade de controle do *datapath* multi-cycle



A unidade de controle do *datapath* multi-cycle com suporte p/ excepções