

Arquitectura de Computadores II



3º Ano

Execução de Instruções em Vários Ciclos Máquina

João Luís Ferreira Sobral
Departamento de Informática
Universidade do Minho



Março 2003

Execução multi-ciclo

● Porque não é utilizada a implementação *single-cycle* ?

- Em geral, a sua implementação é ineficiente, nunca tendo sido utilizada na implementação de processadores
- $CPI=1$, mas a duração de cada ciclo (T_{cc}) deve ser igual ao tempo necessário para executar a instrução mais demorada, uma vez que é complexa a implementação de ciclos de dimensão variável.
- Não permite partilha de unidades funcionais o que leva à sua duplicação

● Exemplo

- Tempos acesso às unidades funcionais:

Memória (Leitura ou escrita)	ALU	Banco de registos
2 nanosegundos	2 nanosegundos	1 nanosegundo

- Mistura de instruções do programa (gcc):

44% tipo R	24% <i>loads</i>	12% <i>stores</i>	18% <i>branch</i>	2% <i>jump</i>
------------	------------------	-------------------	-------------------	----------------

- Qual a diferença entre uma implementação com ciclos de duração fixa e uma implementação com ciclos de duração variável, a estritamente necessária?

- Tempo necessário para a execução de cada tipo de instrução

Tipo de Instrução	Busca da instrução	Leitura de registos	Operação na ALU	Acesso à Memória	Escrita em registo	Total
Tipo R	2	1	2		1	6 ns
<i>Load</i>	2	1	2	2	1	8 ns
<i>Store</i>	2	1	2	2		7 ns
<i>Branch</i>	2	1	2			5 ns
<i>Jump</i>	2					2 ns

- $CPI_{ciclofixo} = CPI_{ciclovariável} = 1$.
- Duração do período do relógio (T_{cc}) na versão com o ciclo de relógio fixo:

$$T_{ccfixo} = \text{duração da instrução mais longa} = \textit{load} = 8 \text{ ns}$$

- Duração média do período de relógio na versão com duração variável do ciclo:

$$T_{ccvar} = 0,44 \times 6 + 0,24 \times 8 + 0,12 \times 7 + 0,18 \times 5 + 0,02 \times 2 = 6,34 \text{ ns}$$

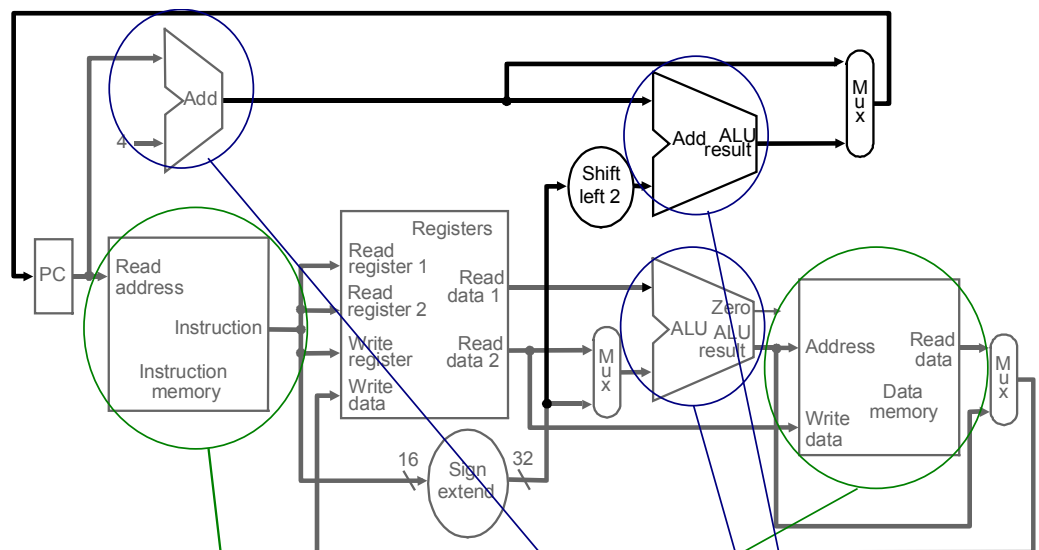
- **Ganho da versão com o ciclo de duração variável**

$$\text{ganho} = T_{execFixo} / T_{execVar} = T_{ccfixo} / T_{ccvar} = 8 / 6,34 = 1,26x$$

Execução multi-ciclo

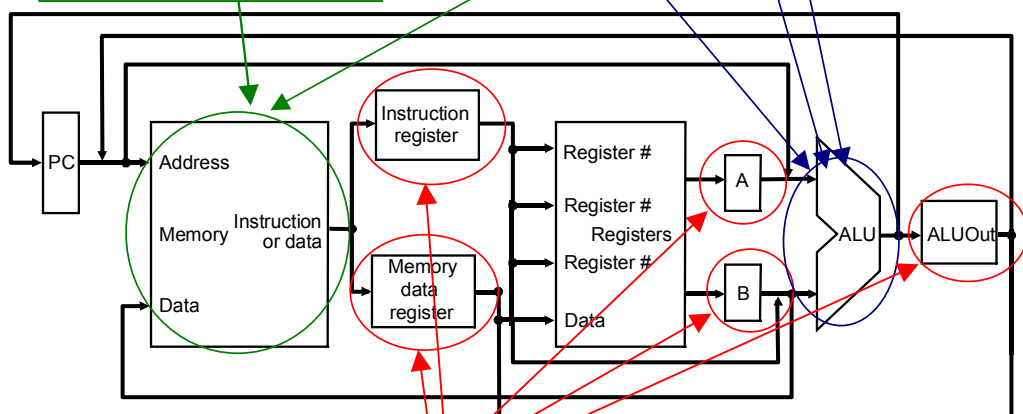
• DP para suporte à execução de instruções em vários ciclos

- A execução das instruções é dividida em várias fases, correspondentes às operações a realizar em cada unidade funcional
- Em cada ciclo pode ser realizado, no máximo, um acesso à memória e um acesso ao banco de registos.
- Cada unidade funcional pode ser utilizada mais que uma vez por instrução, mas em ciclos diferentes, o que permite a junção várias unidades funcionais.



Uma só unidade para acesso às instruções e aos dados

Uma só ALU, em vez de uma ALU e dois *adders*



Registos adicionais para permitir armazenar os resultados intermédios produzidos por uma unidade e que são necessários para fases posteriores de execução da mesma instrução.

Execução multi-ciclo

- DP para suporte à execução de instruções em vários ciclos

- Função dos registos adicionais:

IR (registo de instrução) – Instrução em execução

MDR (registo de dados) – Dados lidos da memória

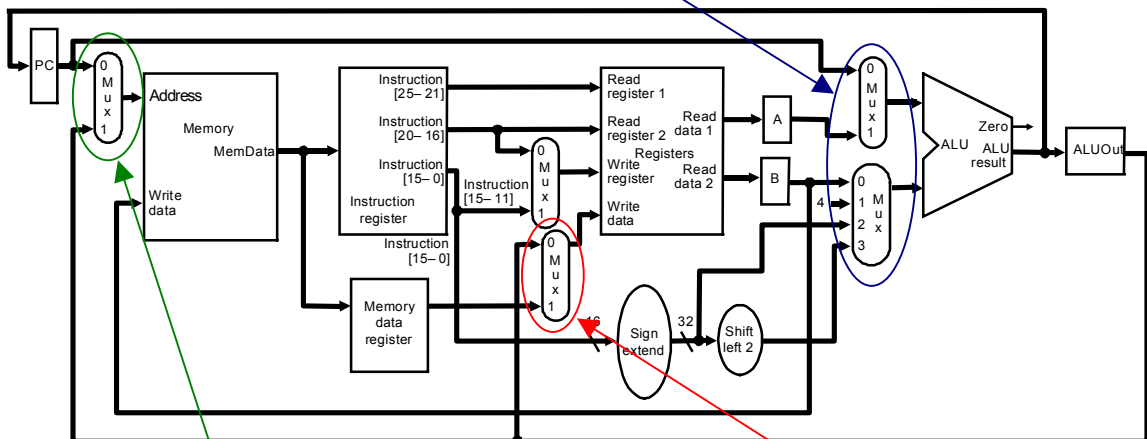
A e B – Valores lidos do banco de registos

ALUOut – Valor calculado pela ALU

- Introdução de novos multiplexers ou extensão dos existentes:

Os multiplexers das entradas da ALU devem permitir 4 funcionalidades (resultantes da junção das 3 unidades do *single-cycle*):

1. $PC + 4$
2. $PC + (Imm) \ll 2$
3. $A \text{ op } B$
4. $A \text{ op } Imm$



O endereço de memória a ler pode ser determinado pelo PC (na busca da instrução, 1) ou pela saída da ALU (em *lw* e *sw* 2):

1. $IR = Mem[PC]$
2. $MDR = Mem[ALUOut]$

O valor a escrever no banco de registos pode vir da saída da ALU (1) ou da memória (2):

1. $Reg[xxx] = ALUOut$
2. $Reg[xxx] = MDR$

Execução multi-ciclo

Sinais de controlo

- Sinais também existentes no DP *single-cycle*:

MemRead, **MemWrite**, **RegDst**, **MemtoReg**, **ALUOp**

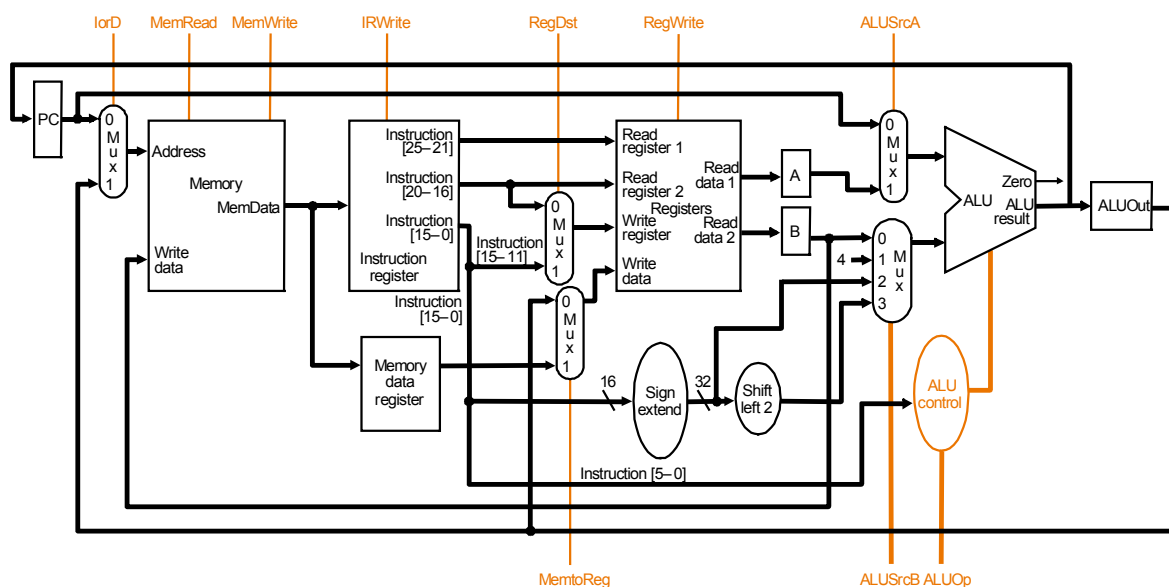
- Novos sinais ou alteração de existentes:

IorD selecciona a utilização de PC ou de ALUOut como endereço de memória

IRWrite Escrita no registo IR

ALUsrcA selecciona PC ou A como primeira entrada da ALU

ALUsrcB selecciona B, 4, Imm ou Imm<<2 como segunda entrada da ALU



- Novos sinais para suporte a saltos (*beq* e *j*, ver fig. seguinte):

PCWrite Escrita incondicional do valor do PC ($PC = PC + 4$ e *Jump*)

PCWriteCond Escrita no PC condicionada pelo resultado da operação da ALU (*beq*)

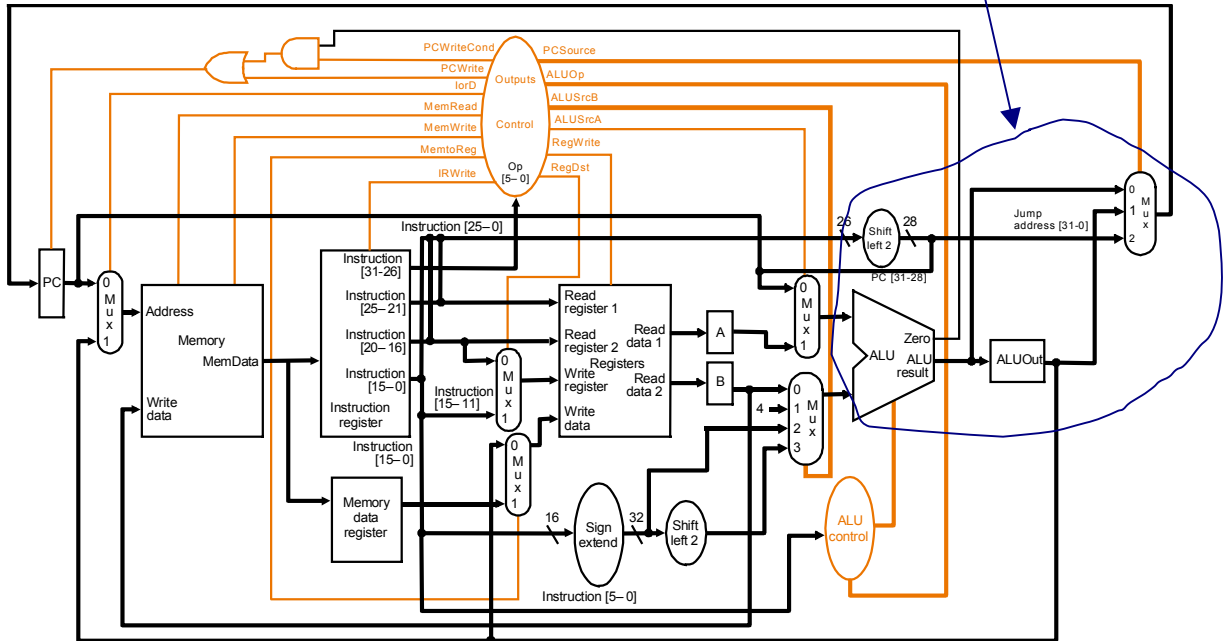
PCSource Novo valor do PC

Execução multi-ciclo

Sinais de controlo

Suporte a saltos (*beq* e *j*) origina 3 fontes possíveis para o PC:

1. Saída da ALU (cálculo de $PC + 4$)
2. ALUOut ($PC + Imm \ll 2$)
3. $PC[31-28] + (IR[25-0] \ll 2)$



Resumo dos Sinais de Controlo

Sinal	Efeito quando inactivo (0)	Efeito quando activo (1)
RegDst	O registo destino provém do campo <i>rt</i> ($I[20-16]$)	O registo destino provém do campo <i>rd</i> ($I[15-11]$)
RegWrite	nenhum	O valor em <i>Write data</i> é escrito no registo indicado em <i>Write register</i>
ALUSrcA	O primeira entrada da ALU é o valor do PC	O primeira entrada da ALU é o valor do registo A
MemRead	nenhum	O endereço de memória é lido para <i>MDR</i> (<i>Memory data register</i>)
MemWrite	nenhum	Os dados em <i>Write data</i> são escritos no endereço de memória
MemtoReg	Os dados a escrever num registo provém de ALUOut	Os dados a escrever num registo provém de <i>MDR</i>
IRWrite	nenhum	O valor de saída da memória é colocado no <i>IR</i> (<i>Instruction register</i>)
PCWrite	nenhum	O PC é escrito com o valor seleccionado por <i>PCSource</i>
PCWriteCond	nenhum	O PC é escrito se o sinal <i>Zero</i> da ALU também se encontrar activo

Execução multi-ciclo

● Resumo dos Sinais de Controlo (continuação)

Sinal	Valor	Efeito
ALUop	00	A ALU efectua uma adição
	01	A ALU efectua uma subtracção
	10	O campo <i>Funct</i> da instrução determina a operação a realizar pela ALU
ALUSrcB	00	A segunda entrada da ALU provém do registo B
	01	A segunda entrada da ALU é 4
	10	A segunda entrada da ALU são os bits IR[15-0] estendidos com sinal para 32 bits
	11	A segunda entrada da ALU são os bits IR[15-0] estendidos com sinal para 32 bits e multiplicados por 4 (shift left 2)
PCSource	00	O valor enviado para o PC é a saída da ALU (PC + 4)
	01	O valor enviado para o PC é conteúdo de ALUOut (destino do salto)
	10	O valor enviado para o PC é $IR[26-0] \ll 2 + (PC+4)[31-28]$

● Considerações sobre as fases de execução das instruções

- As várias fases de execução devem ter uma carga balanceada, uma vez que a duração de um ciclo de relógio é fixa e igual à fase mais longa
- Em cada fase de execução de uma instrução cada unidade funcional apenas pode ser utilizada uma vez
- Em cada fase podem ser utilizadas simultaneamente várias unidades funcionais
- No fim de cada fase os valores são escritos em registos (internos ou visíveis ao programador)
- O novo valor de cada registo só é visível no ciclo de relógio seguinte

Execução multi-ciclo

● Fases de Execução das Instruções:

1. Busca da instrução

$IR = Mem[PC]$ $IorD=0; MemRead=1; IRWrite=1$
 $PC = PC + 4$ $ALUSrcA=0; ALUSrcB=01; ALUOp=00; PCWrite; PCSource=00$

2. Descodificação da instrução, selecção dos registos e cálculo do end. salto

$A = Reg[IR[25-21]]$
 $B = Reg[IR[20-16]]$
 $ALUOut = PC + sgn-ext(IR[15-0]) \ll 2$ $ALUSrcA=0; ALUSrcB=11; ALUOp=00$

3. Execução da operação (depende da instrução em execução):

a) LW e SW - calcula o endereço de memória

$ALUOut = A + sgn-ext(IR[15-0])$ $ALUSrcA=1; ALUSrcB=10; ALUOp=00$

b) Tipo R - executa a operação correspondente à instrução (*op*)

$ALUOut = A \text{ op } B$ $ALUSrcA=1; ALUSrcB=00; ALUOp=10$

c) BEQ - executa o salto condicional

se $(A==B)$ $PC = ALUOut$ $ALUSrcA=1; ALUSrcB=00; ALUOp=01;$
 $PCWriteCond; PCSource=01$

d) J - executa o salto

$PC = (PC+4)[31-28] + IR[25-0] \ll 2$ $PCSource=10; PCWrite$

4. Acesso à memória ou tipo R (só em *lw*, *sw* e tipo R)

a) LW - lê o valor da memória

$MDR = Mem[ALUOut]$ $MemRead; IorD=1;$

b) SW - escreve o valor em memória

$Mem[ALUOut]=B$ $MemWrite; IorD=1;$

c) Tipo-R - completa a operação escrevendo o resultado no registo destino

$Reg[IR[15-0]]=ALUOut$ $MemtoReg=0; RegWrite; RegDst=1$

5. Completar o acesso à memória (só em *lw*)

$Reg[IR[20-16]] = MDR$ $MemtoReg=1; RegWrite; RegDst=0$

Execução multi-ciclo

● Especificação da Unidade de Controlo através de uma Máquina de Estados Finitos (FSM)

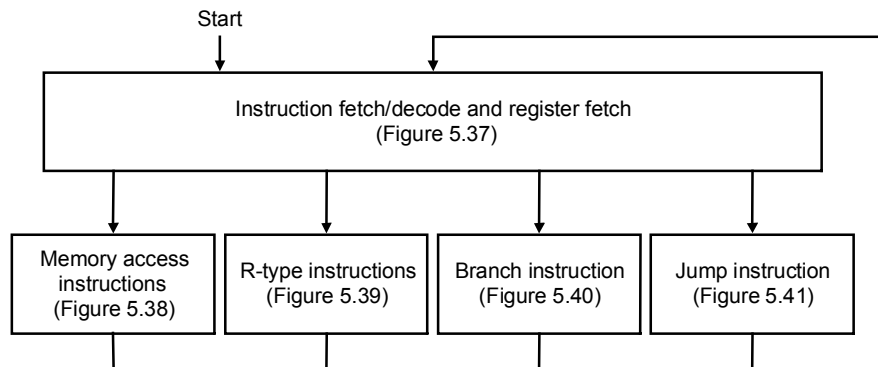
- A unidade de controlo do DP *single-cycle* pode ser especificada através de uma função do tipo:

$$\text{Saídas} = f(\text{Entradas}) \Leftrightarrow \text{Sinais de controlo} = f(\text{OpCode})$$

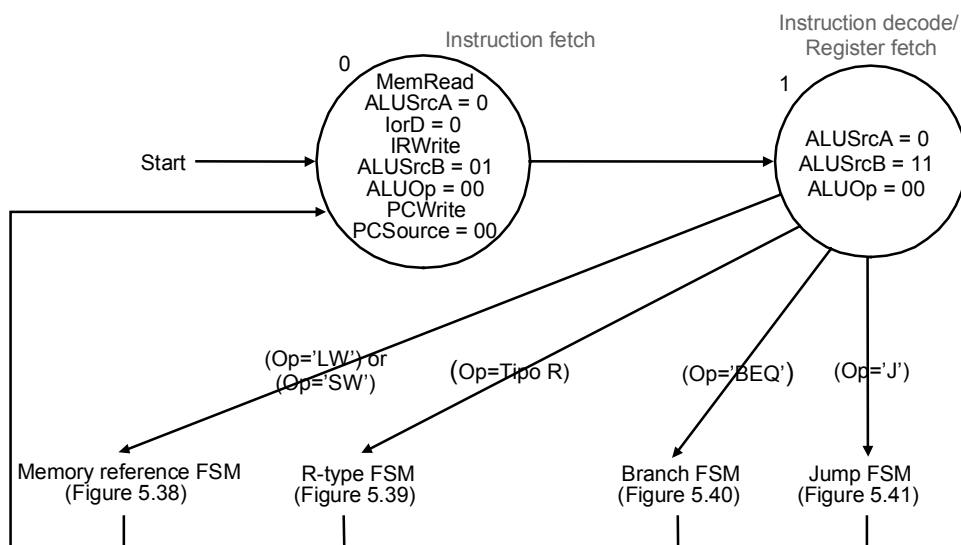
- A unidade de controlo para uma execução multi-ciclo é mais complexa porque as instruções são executadas em vários ciclos:

$$\text{Sinais de controlo} + \text{estado seguinte} = f(\text{OpCode} + \text{estado actual})$$

- Visão global da máquina de estados



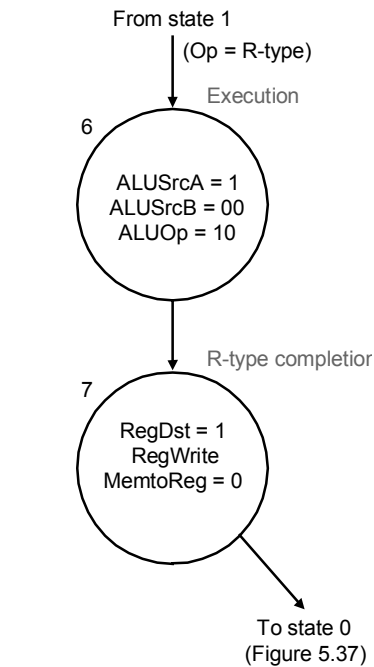
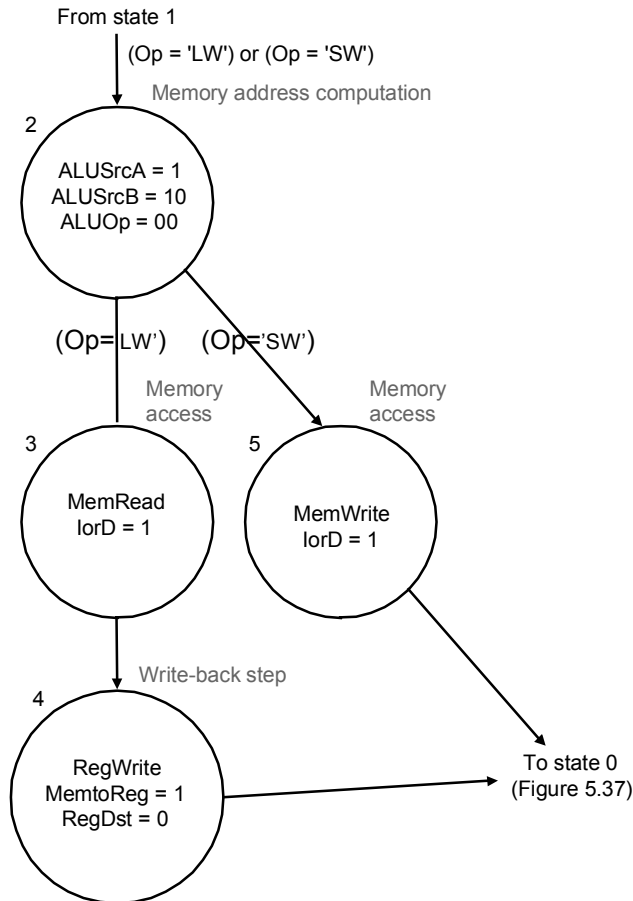
- Análise detalhada da FSM da unidade de controlo



1. Busca da instrução (estado 0)
2. Decodificação, selecção dos registos, cálculo do end. de salto (estado 1)

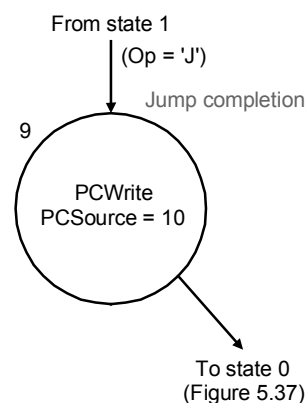
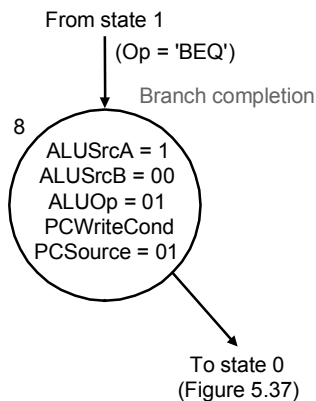
Execução multi-ciclo

Máquina de Estados da Unidade de Controlo (continuação)



- **Tipo R**
 3. Execução da operação (estado 6)
 4. Escrita em registo (estado 7)

- **LW e SW**
 3. Cálculo do endereço de memória (estado 2)
 4. Acesso à memória (LW – leitura – estado 3) (SW – escrita - estado 5)
 5. Escrita do valor em registo (LW – estado 4)

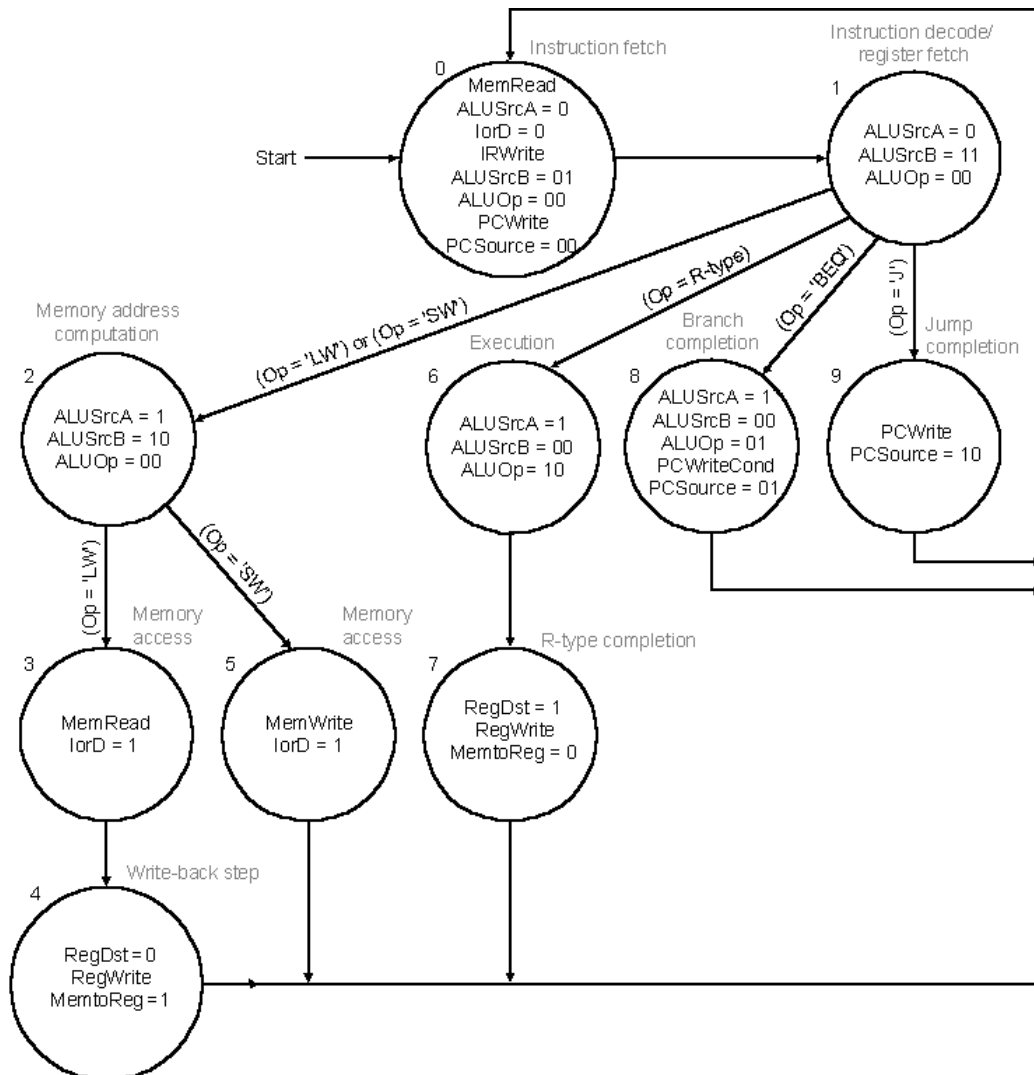


- **BEQ e J**
 3. Execução do salto (beq, condicional - estado 8)(j, incondicional, estado 9)

Execução multi-ciclo

• Máquina de Estados da Unidade de Controlo (continuação)

- Diagrama completo



• Exemplo do cálculo do CPI na versão multi-ciclo

- Número de ciclos necessários para cada instrução:

Lw	Sw	Tipo R	Beq	J
5	4	4	3	3

- Mistura de instruções do programa (F_i)

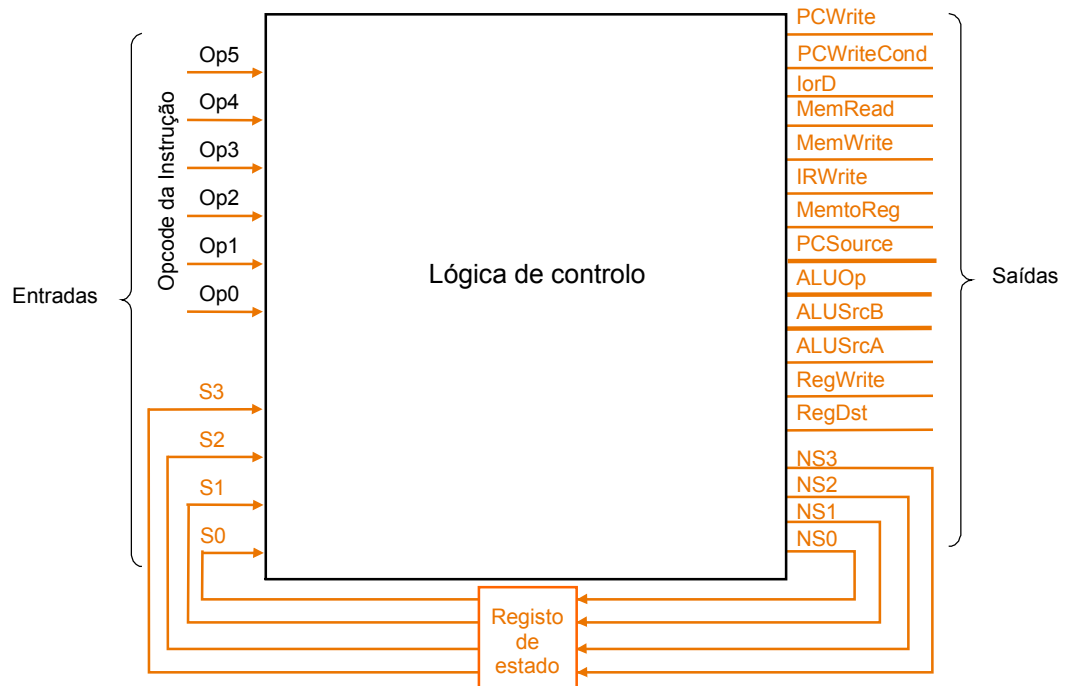
Lw	Sw	Tipo R	Beq	J
22%	11%	49%	16%	2%

$$CPI_{global} = \sum_{i=1}^n CPI_i * F_i = 0,22 \times 5 + 0,11 \times 4 + 0,49 \times 4 + 0,16 \times 3 + 0,02 \times 3 = 4,04$$

Execução multi-ciclo

● Implementação da Unidade de Controlo

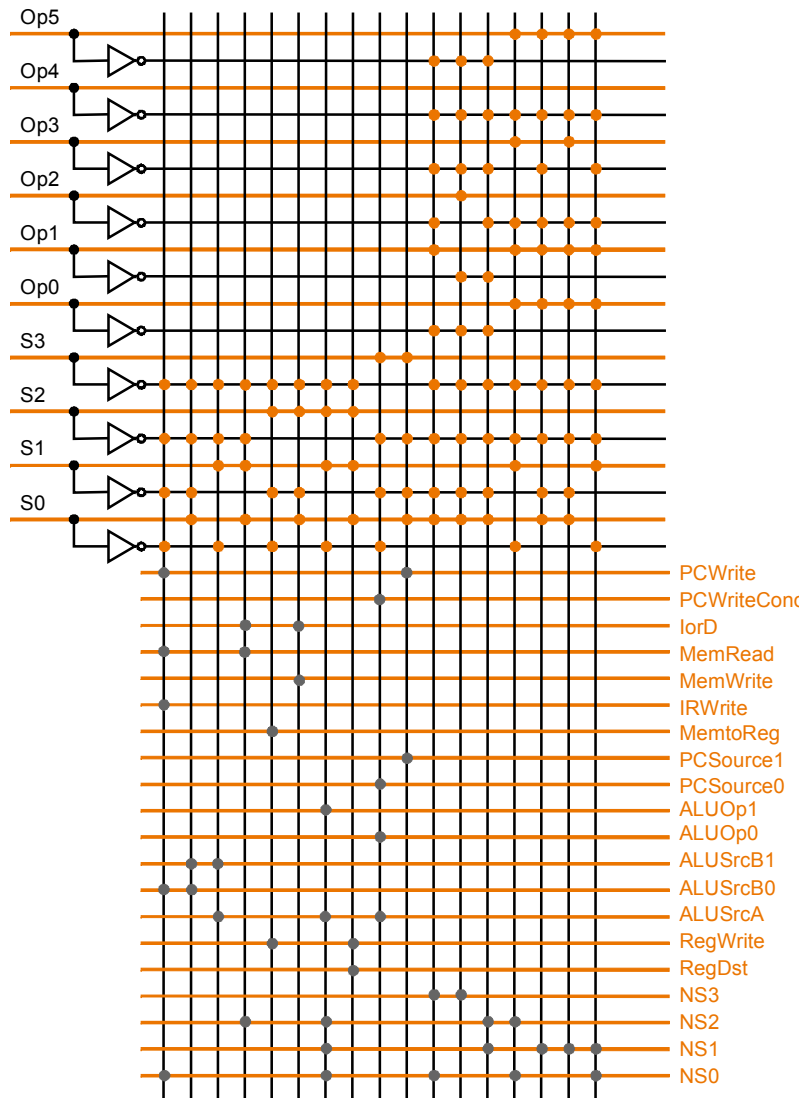
- A unidade de controlo necessita do estado actual para produzir os sinais de controlo adequados e calcular o próximo estado
- São necessários 4 bits para representar o estado, uma vez que existem 10 estados diferentes



Saída	Estado actual	Opcode
PCWrite	state0 + state9	
PCWriteCond	state8	
lOrD	state3 + state5	
MemRead	state0 + state3	
MemWrite	state5	
IRWrite	state0	
MemtoReg	state4	
PCSource1	state9	
PCSource0	state8	
ALUOp1	state6	
ALUOp0	state8	
ALUSrcB1	state1 + state2	
ALUSrcB0	state0 + state1	
ALUSrcA	state2 + state6 + state8	
RegWrite	state4 + state7	
RegDst	state7	
NextState0	state4 + state5 + state8 + state9	
NextState1	state0	
NextState2	state1	(Op='lw')+(Op='sw')
NextState3	state2	(Op='lw')
NextState4	state3	
NextState5	state2	(Op='sw')
...

Execução multi-ciclo

● Implementação da Unidade de Controlo com PLA



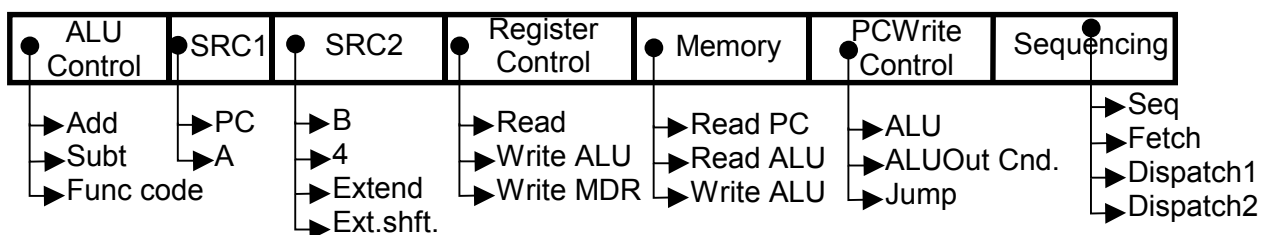
● Limitações da Especificação da Unidade de Controlo com uma FSM

- Os diagramas de estados de arquiteturas reais são bastante mais complexos (MIPS contém cerca de 100 instruções com 1 a 20 ciclos)
- Os diagramas de estados apenas são úteis para um número reduzido de estados
- Solução: utilizar **microinstruções** que especificam os sinais de controlo a activar em cada estado

Execução multi-ciclo

● Implementação da Unidade de Controlo com Microprogramação

- Um micro-programa é constituído com um conjunto de micro-instruções
- Cada micro-instrução especifica os sinais de controlo activos e o próximo estado, de forma simbólica
- As micro-instruções, tal como as instruções máquina contêm vários campos, podendo cada campo agrupar vários de sinais de controlo
- Exemplo de um formato de micro-instrução



Nome do Campo	Valor do Campo	Saídas Activas	Descrição
ALU Control	Add	ALUOp=00	A ALU efectua uma adição
	Subt	ALUOp=01	A ALU efectua um subtracção
	Func code	ALUOp=10	O campo <i>funct</i> indica a operação
SRC1	PC	ALUSrcA=0	A primeira entrada da ALU é PC
	A	ALUSrcA=1	A primeira entrada da ALU é o registo A
SRC2	B	ALUSrcB=00	A segunda entrada da ALU é B
	4	ALUSrcB=01	A segunda entrada da ALU é 4
	Extend	ALUSrcB=10	A segunda entrada da ALU é ext.(Imm16)
	Ext.shft	ALUSrcB=11	A segunda entrada da ALU é ext.(Imm16)<<2
Register Control	Read		Os registos indicados nos campos rs e rt são lidos para A e B
	Write ALU	RegWrite RegDst=1 MemtoReg=0	Escreve o valor de ALUOut no registo indicado no campo da instrução rd
	Write MDR	RegWrite RegDst=0 MemtoReg=1	Escreve o valor de MDR no registo indicado no campo da instrução rt
Memory	Read PC	MemRead lrd=0, IRWrite	Ler o conteúdo da memória indicado por PC para o IR
	Read ALU	MemRead, lrd=1	Ler o conteúdo da memória indicado por ALUOut para o MDR
	Write ALU	MemWrite, lrd=1	Escreve no conteúdo da memória indicado por ALUOut o valor do registo B
PCWrite Control	ALU	PCSource=00, PCWrite	Escreve o conteúdo da saída da ALU no PC
	ALUOut cond	PCSource=01, PCWrite Cond	Se zero estiver activo, escreve no PC conteúdo de ALUout
	Jump address	PCSource=10, PCWrite	Escreve no PC o endereço de salto indicado na instrução
Sequencing	Seq	AddrCtl=11	Escolhe a próxima microinstrução seq.
	Fetch	AddrCtl=00	Vai para a primeira micro-instrução
	Dispatch	AddrCtl=01	<i>Dispatch</i> utilizando a ROM

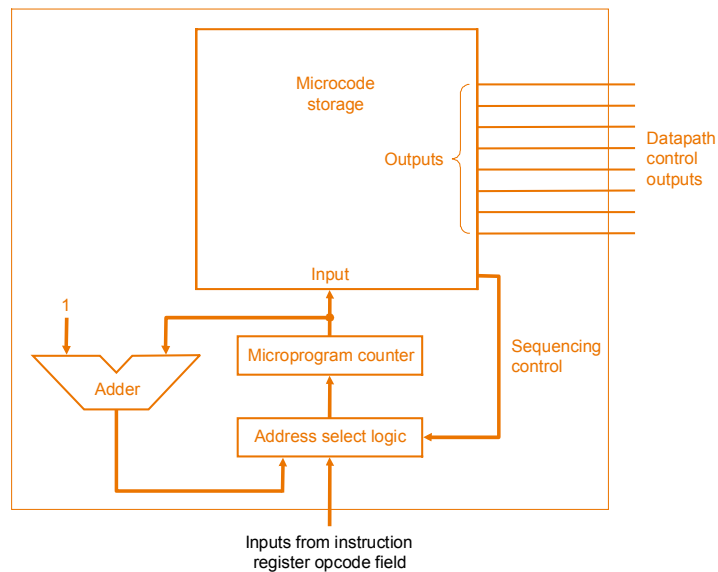
Execução multi-ciclo

● Implementação da Unidade de Controlo com Microprogramação

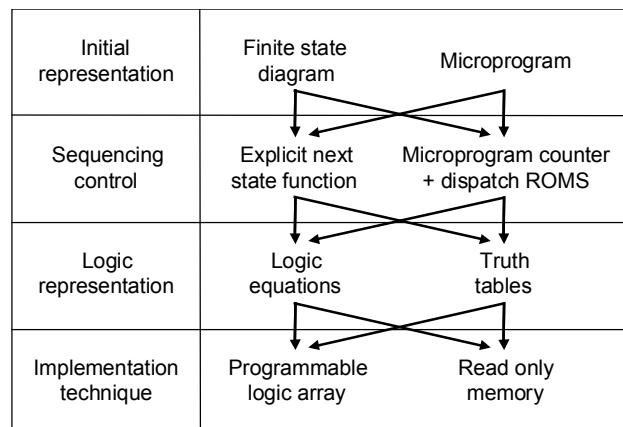
- Micro-programa completo

Etiqueta	ALU control	SRC1	SRC2	Register control	Memory	PCWrite control	Sequencing
Fetch	Add	PC	4		Read PC	ALU	Seq
	Add	PC	Extshft	Read			Dispatch 1
Mem1	Add	A	Extend				Dispatch 2
Lw2					Read ALU		Seq
				Write MDR			Fetch
Sw2					Write ALU		Fetch
Tipo-R1	Func code	A	B				Seq
				Write ALU			Fetch
Beq1	Subt	A	B			ALUOut-con	Fetch
Jump1						Jump	Fetch

- Implementação do micro-programa com uma ROM



● Alternativas para a implementação multi-ciclo



Execução multi-ciclo

● Tratamento de Excepções

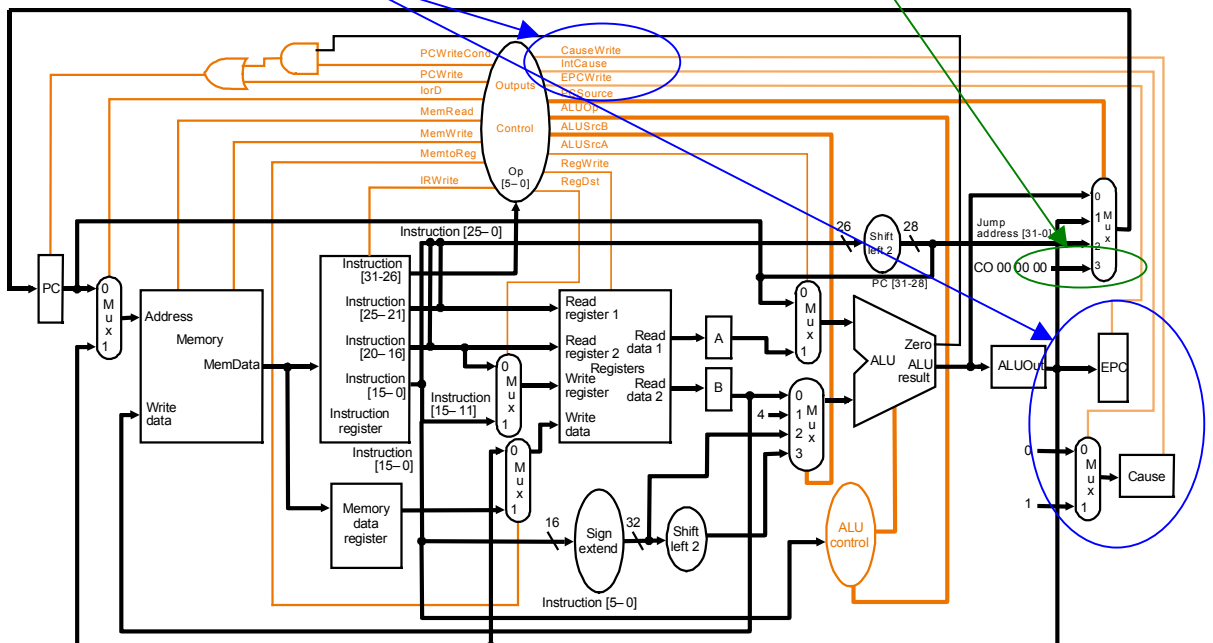
- Excepções – eventos inesperados que ocorrem no processador (ex. *overflow*)
- As interrupções são excepções provocadas externamente (i.e. de periféricos)
- Tratamento de excepções em MIPS:
 1. O endereço da instrução em execução é colocado em EPC
 2. O processador salta para uma posição de memória pré-definida
 3. A causa da excepção é indicada num registo de causa.
- DP para suporte a excepções de *overflow* e instrução indefinida

Novos registos e sinais de controlo

- CauseWrite – Escrita no registo de causa
- EPCWrite – Escrita no registo EPC
- IntCause – selecciona a causa da excepção

Novo endereço de salto

- PCSource=11 – Força a execução da instrução no endereço 0xC0000000



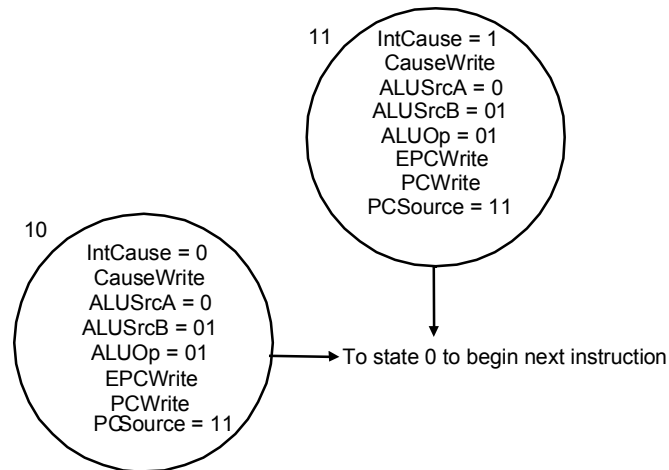
- Como identificar as excepções:

1. Instrução não definida – o opcode (IR[31-25]) não é conhecido
2. *Overflow* – Requer uma saída da ALU para a unidade de controlo

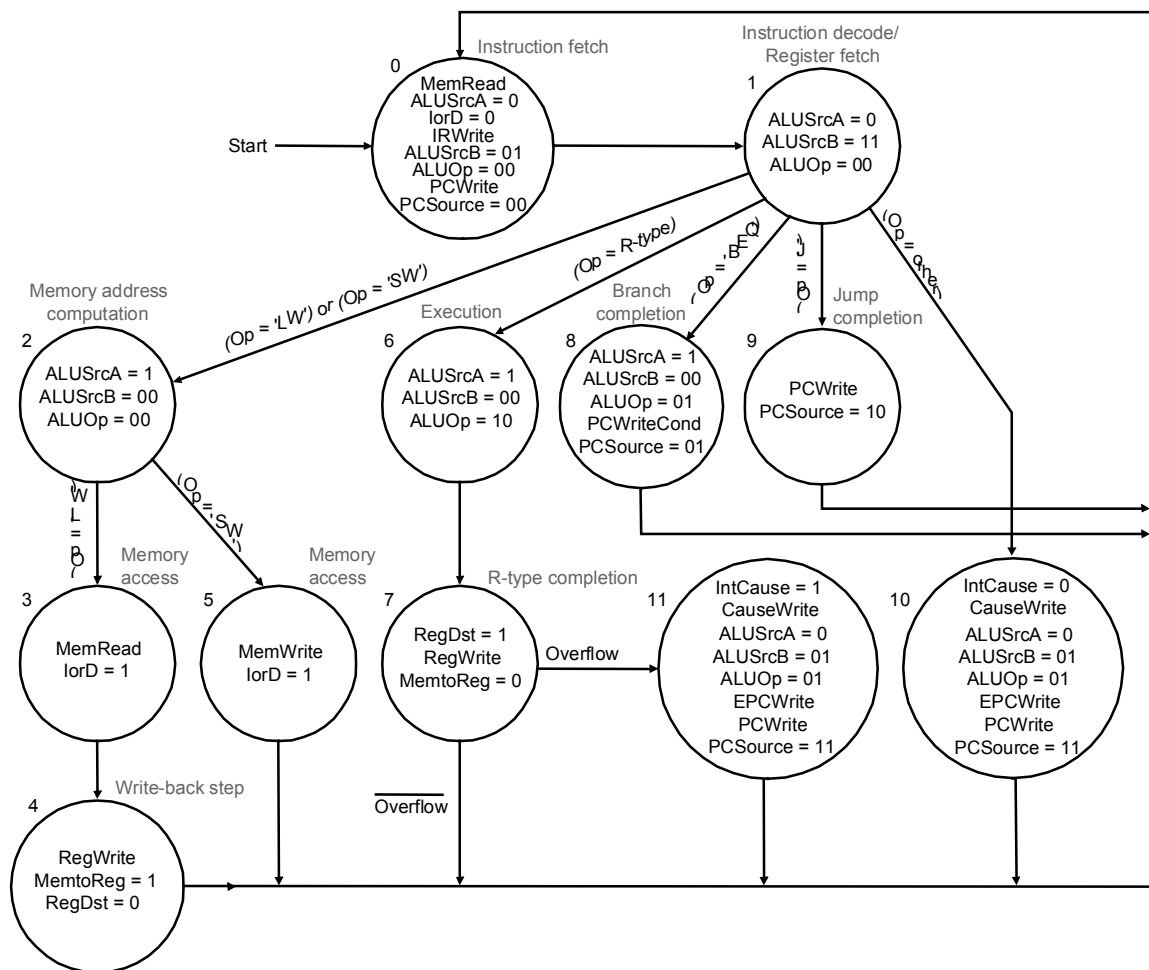
Execução multi-ciclo

Tratamento de Exceções

- Novos estados



- Diagrama de estados completo, incluindo o suporte a exceções



Bibliografia

Matéria	Bibliografia
<i>Data path</i> Multi-ciclo e Unidade de Controlo	COD2e 5.4
Implementação da Unidade de Controlo	COD2e C.3, 5.7, 5.9
Micro-código	COD2e 5.5, C.5
Tratamento de exceções	COD2e 5.6

COD2e *Computer Organization and Design: the hardware/software interface*, D. Patterson and J. Hennessy, Morgan Kaufmann, 2ª edição, 1998.