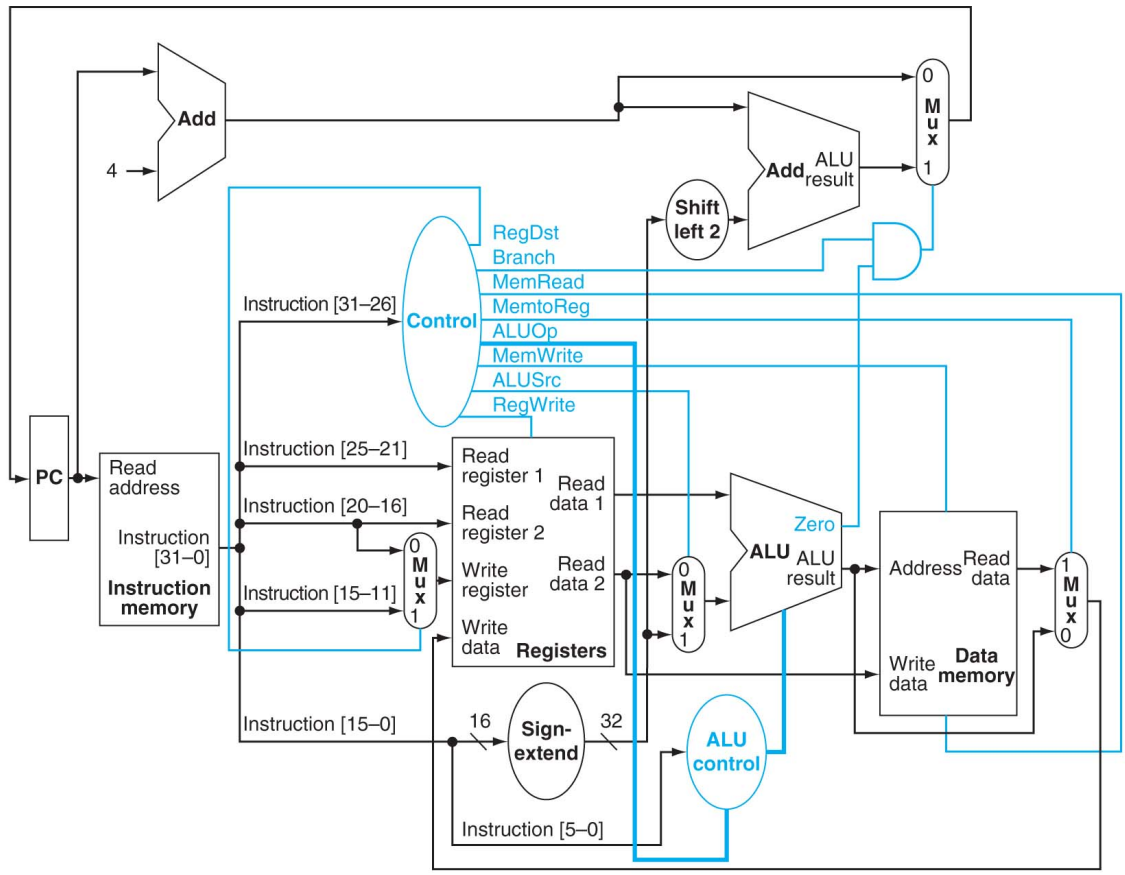
Arquitetura de Computadores I
3ª série de problemas
16.12.2014

I – Single-Cycle Datapath



Instruction	RegDst	ALUSrc	Memto-Reg	Reg-Write	Mem-Read	Mem-Write	Branch	ALUOp1	ALUOp0
R-format	1	0	0	1	0	0	0	1	0
lw	0	1	1	1	1	0	0	0	0
sw	X	1	X	0	0	1	0	0	0
beq	X	0	X	0	0	0	1	0	1

1. Descreva o efeito que teriam os seguintes bloqueios ("*stuck-at*") dos sinais de control a 0 ou a 1, indicando para cada caso quais as instruções que não executariam corretamente:
   a. RegWrite = 0

      O resultado das operações aritméticas e lógicas (de todas as instruções de tipo R) e os dados transferidos da memória (instruções de load) não seriam armazenados em registo, i.e. todas essas instruções corresponderiam a NOPs.
   b. RegWrite = 1

      O resultado da diferença entre os dois registos que figuram como operandos nas instruções de *branch* seriam armazenados no registo **rt** se RegDst = 0 ou no registo correspondente aos bits 15-11 do código de instrução se RegDst = 1
   c. ALUop0 = 0

      Nas instruções de *branch* em lugar de **rs - rt** é efetuada a soma **rs + rt**
   d. ALUop0 = 1

Nas instruções de referência à memória (*lw* e *sw*) é fornecido o endereço errado (o endereço de memória será **(rs) – immediate** em lugar de **(rs) + immediate**). Nas instruções de tipo-R ALUop = 11, o que corresponde a um código não atribuído. Se quando ALUop1 = 1 é sempre o campo **func** (bits 26-31 da instrução) que determina os bits de controle da ALU, as instruções de tipo-R serão corretamente executadas.

e. ALUop1 = 0

   A operação da ALU será **(rs) + immediate** (se ALUop0 = 0) ou **(rs) – (rt)** (se ALUop0 = 1). As instruções de tipo-R, com a exceção de *sub*, serão erradamente executadas.

f. ALUop1 = 1

   A operação da ALU será definida pelos bits 26-31 da instrução. Apenas as instruções de tipo-R serão corretamente executadas.

g. Branch = 0

   A instrução seguinte a ser executada será sempre a instrução armazenada na palavra de memória com o endereço seguinte, i.e. as instruções de *branch* serão *nop*

h. Branch = 1

   Sempre que o resultado da operação efetuada pela ALU = 0 será efetuado um salto na execução do programa.

i. MemRead = 0

   Nunca são lidos dados da memória, i.e. as instruções de *load* não são corretamente executadas

j. MemRead = 1

   Na execução de instruções de *store* MemRead = MemWrite =1, o que poderá provocar a avaria da memoria de dados.

k. MemWrite = 0

   Nunca são escritos dados da memória, i.e. as instruções de *store* não são corretamente executadas

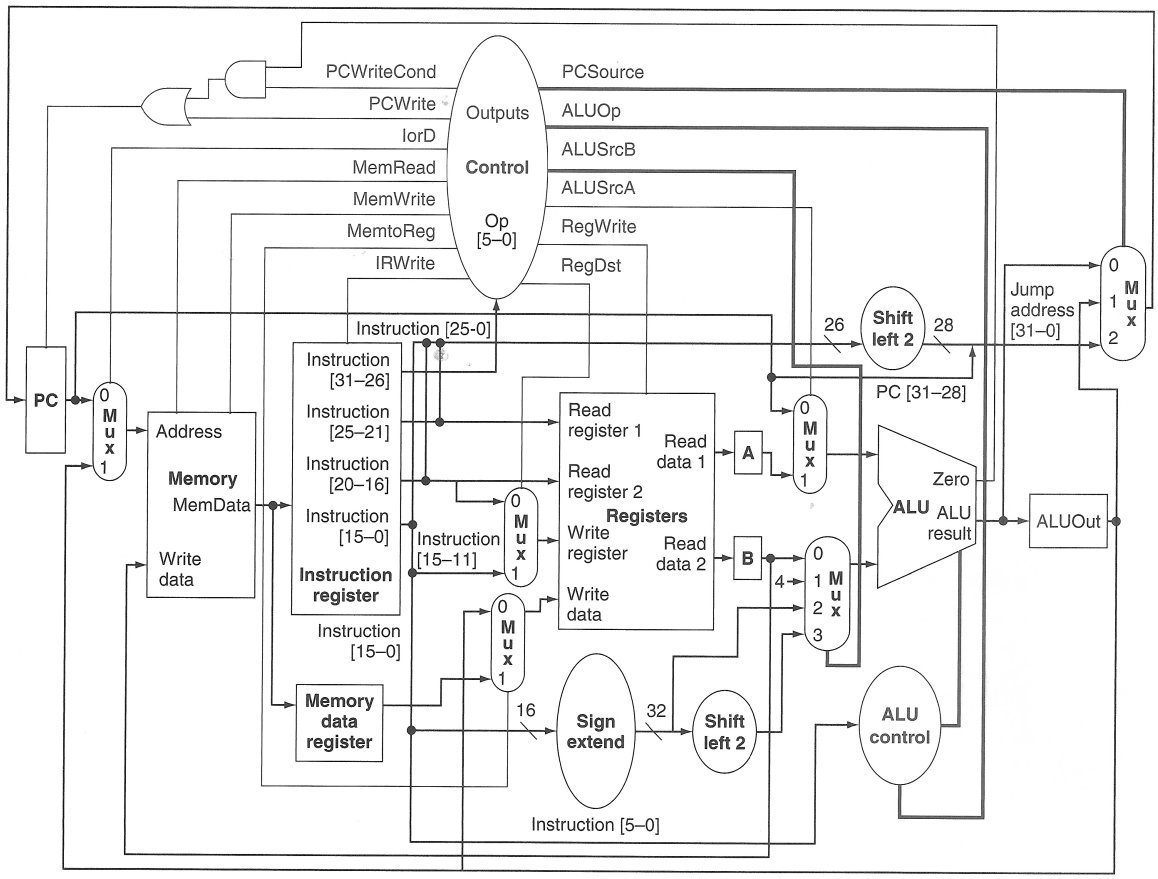
l. MemWrite = 1

   Na execução de instruções de *load* MemRead = MemWrite =1. o que poderá provocar a avaria da memoria de dados.

2. Pretende-se que o processador execute tambem a instrução ***jump register***. Indique o que seria necessário acrescentar ao datapath e as alterações a introduzir na tabela com os valores dos sinais de control (Nota: na instrução ***jr*** o campo rs do código de instrução indica o registo que contem o endereço da instrução seguinte a executar).

   A solução mais simples é introduzir um novo multiplexer (32 MUX de duas entradas) no circuito de atualização do PC, a seguir ao atual. O novo MUX teria na entrada 0 a saída do MUX já existente e na entrada 1 Read data 1, i.e. o conteúdo do registo Rs, sendo controlado por um novo sinal da unidade de control, *JumpReg* que tem o valor 1 quando é ***jr*** a instrução em execução.

II - **Multi-Cycle Datapath**



3. Descreva o efeito que teriam os seguintes bloqueios ("*stuck-at*") dos sinais de control a 0 ou a 1, indicando para cada caso quais as instruções que não executariam corretamente:
   a. RegWrite = 0

      O resultado das operações aritméticas e lógicas (de todas as instruções de tipo R) e os dados transferidos da memória (instruções de load) não seriam armazenados em registo, i.e. todas essas instruções corresponderiam a NOPs.

   b. MemRead = 0

      Nunca é lida informação da memória (nem instruções nem dados), pelo que o processador não funciona.

   c. MemRead = 1

      Na execução de instruções de *store* MemRead = MemWrite =1, o que poderá provocar a avaria da memoria

   d. MemWrite = 0

      Nunca são escritos dados da memória, i.e. as instruções de *store* não são corretamente executadas

   e. MemWrite = 1

      Nenhuma operação de leitura da memória, seja para instruções ou para dados, é executada corretamente (MemRead = MemWrite =1). O processador não funciona e a memoria poderá ficar danificada.

   f. IRWrite = 0

      O processador não funciona pois o conteúdo do Instuction Register nunca é alterado. Assumindo que na inicialização é feito o *reset* do Instruction Register, o seu conteúdo será permanentemente 0, o que corresponde a *nop*.

   g. IRWrite = 1

      Nas instruções de *load* o dado lido da memória é carregado no Instruction Register. As

saídas da unidade de control deixarão de corresponder à instrução em curso de execução

h. PCWrite = 0

O conteúdo do PC mantém permanentemente o valor inicial, quando o sistema é ligado, repetindo-se indefinidamente a execução da instrução armazenada nessa posição de memoria (salvo se for uma instrução de *branch* (PCWriteCond = 1) cuja condição se verifique (Zero = 1), caso em que o conteúdo do PC passa a ser o endereço da instrução alvo do *branch* e o bloqueio acontece nessa instrução e não na 1ª)

i. PCWrite = 1

Em cada ciclo de memória um novo valor, determinado por PCSource, é escrito no PC. Por exemplo, nas instruções aritméticas e lógicas será armazenado no PC o resultado da operação, sendo o conteúdo da célula de memória com esse endereço de memória (que poderá até corresponder à zona da memória de dados) o valor armazenado em IR.

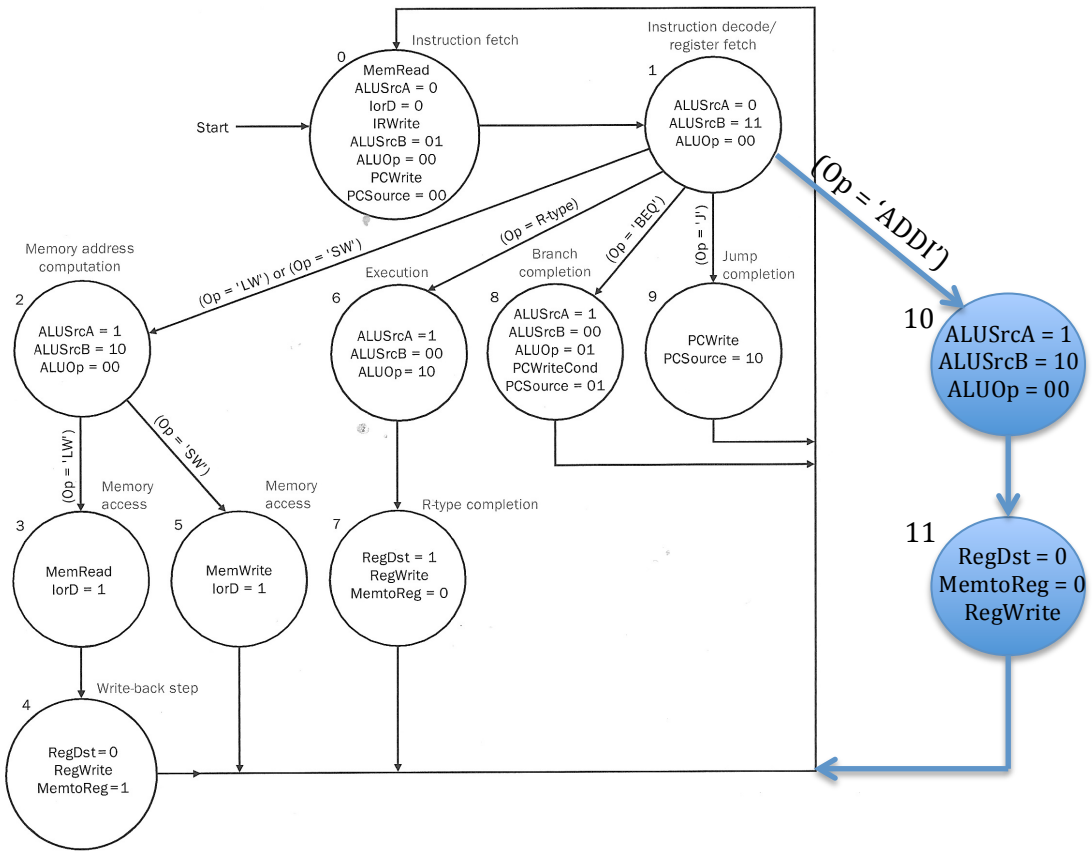
j. PCWriteCond = 0

É sempre executada a instrução seguinte. As instruções de *branch* tornam-se *nops*

k. PCWriteCond = 1

Sempre que o resultado da operação efetuada pela ALU for 0 (Zero = 1), e não apenas nas instruções de *branch*, é carregado no PC um novo valor, determinado por PCSource, com consequências idênticas às descritas na alínea i

4. Pretende-se que o processador execute tambem a instrução ***add immediate*** **(addi)**. Indique as modificações a introduzir na máquina de estados que representa a unidade de controle e nas respetivas saídas e eventuais alterações necessárias no datapath. Indique o valor dos sinais de control em cada ciclo da execução de addi.

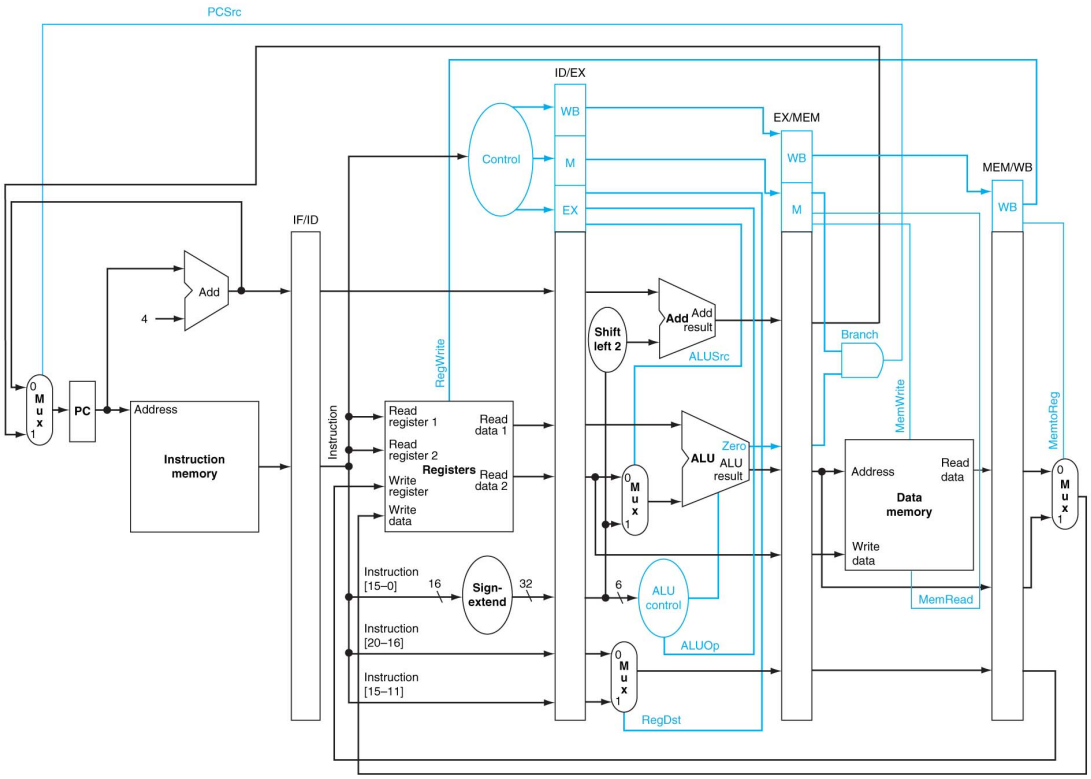


Não são necessárias alterações no datapath.

5. Assuma que a unidade de controle é microprogramada. Escreva um microprograma que implemente **addi**.

Label	ALU control	SRC1	SRC2	Register control	Memory	PCWrite control	Sequencing
Fetch	Add	PC	4		Read PC	ALU	Seq
	Add	PC	Extshft	Read			Dispatch 1
ADDI1	Add	A	Extend				Seq
				Write ALU			Fetch

III – Pipelined Datapath



6. Utilizando a representação multicycle do pipeline mostre os reencaminhamentos de dados (*forwarding paths*) necessários para executar a seguinte sequência de instruções:

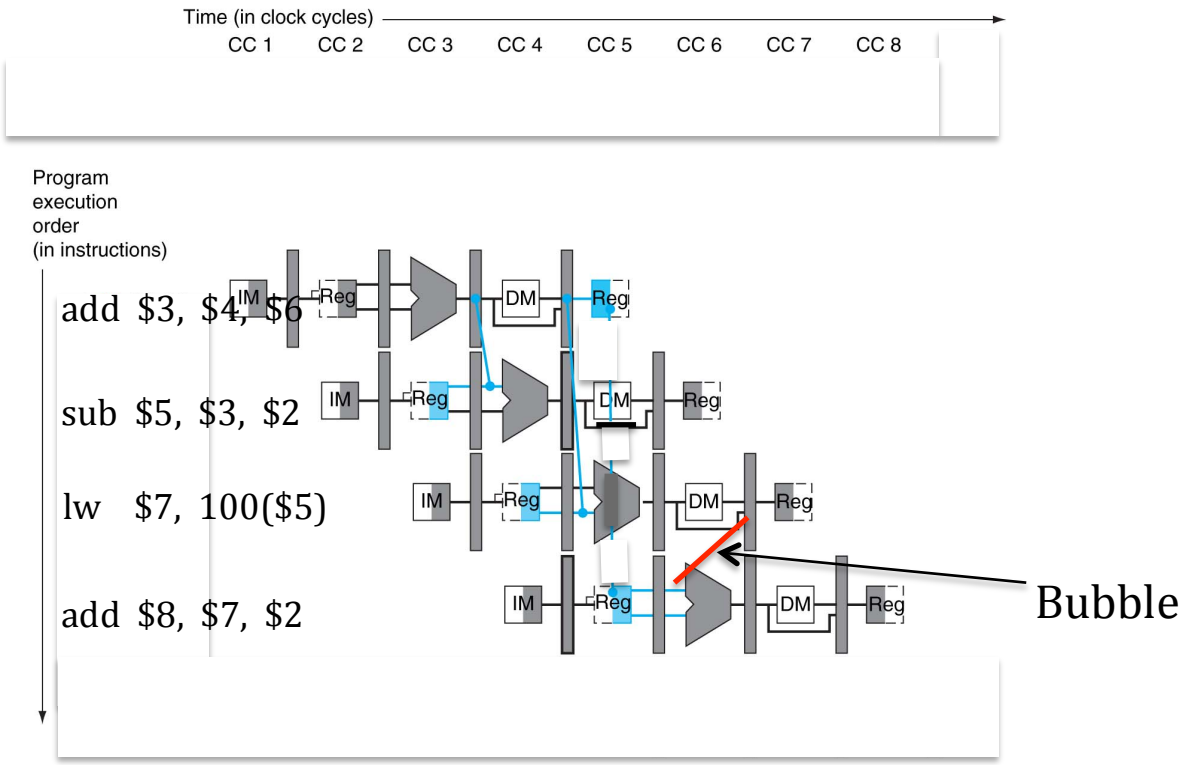
```
add  $3,  $4,  $6
sub  $5,  $3,  $2
lw   $7,  100($5)
add  $8,  $7,  $2
```

Forward Paths (indicados a azul na figura):

EX/MEM – ALU B

EX/MEM – ALU A

Pipeline Stall (Bubble) – a vermelho na figura



7. Identifique todas as dependências de dados na sequência de instruções seguinte. Quais dessas dependências podem ser resolvidas com *forwarding* e quais e quais introduzem uma bolha (*bubble*) no pipeline.

```
add  $3,  $4,  $2
sub  $5,  $3,  $1      # $3 dependente do resultado de add – resolúvel com forwarding
lw   $6,  200($3)      # $3 dependente do resultado de add – resolúvel com forwarding
add  $7,  $3,  $6      # $6 dependente do resultado de lw - Bubble
```

8. A seguinte sequência de instruções é executada no pipeline:

```
lw   $5,  40($2)
add  $6,  $3,  $2
or   $7,  $2,  $1
and  $8,  $2,  $3
sub  $9,  $2,  $1
```

Cada registo tem o valor inicial 10_{10} + número do registo (p.ex. o registo $8 tem o valor 18_{10}). Cada posição da memória de dados tem o valor inicial 1000_{10} + o seu endereço (p.ex. Memory[8] tem o valor 1008_{10}). No ciclo 5 o PC tem o valor 100_{10}, o endereço da instrução **sub.**

Indique o valor em cada campo dos 4 registos do pipeline, IF/ID, ID/EX, EX/MEM, MEM/WB, no ciclo 5.

IF/ID (and $8, $2, $3):
IF/ID _PC = 96_{10}
IF/ID _IR = and $8, $2, $3 instruction code: 000000 00010 00011 01000 00000 011000

ID/EX (or $7, $2, $1):
ID/EX _PC = 92_{10}
ID/EX _Read Data1 = 12_{10}
ID/EX _Read Data2 = 11_{10}
ID/EX _Instruction_Immediate = 0000000000000000 0011100000011001

ID/EX _Instruction[20-16] = 00001
ID/EX _Instruction[15-11] = 00111
ID/EX _WB = 1
ID/EX _M = 0, 0, 0
ID/EX _EX = 0, 10, 0

EX/MEM (add $6, $3, $2):
EX/MEM_Add Result = 11111111111111111100000011011000
EX/MEM_ALU Result = 25_{10}
EX/MEM_Read Data2 = 12_{10}
EX/MEM_Instruction_RegDst = 6_{10}
EX/MEM_WB = 1
EX/MEM_M = 00

MEM/WB (lw $5, 40($2)):
MEM/WB_Read Data = 1052_{10}
MEM/WB_ALU Result = 52_{10}
MEM/WB_WB = 10