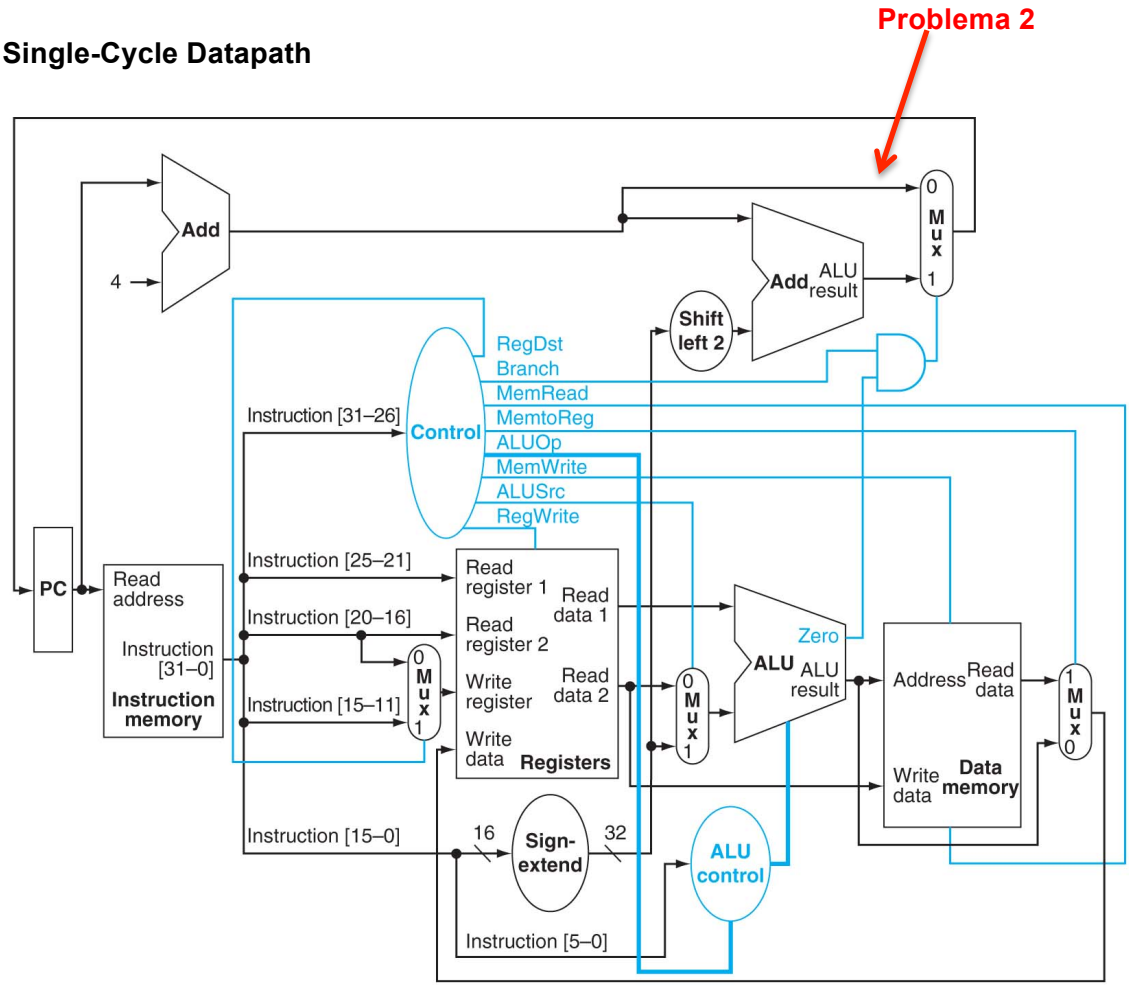
Arquitetura de Computadores I
4ª série de problemas – soluçoes
7.12.2015

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
      Nunca haveria armazenamento num registo do resultado das instruções aritméticas e lógicas nem dos dados lidos da memória, i.e. todas essas instruções corresponderiam a NOPs.
   b. RegWrite = 1
      Na instrução de branch seria escrito no registo indicado no campo rt do código da instrução o valor ($rs) – ($rt).
   c. ALUop0 = 0
      Nas instruções de branch em lugar da subtração (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. Como 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, 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 danificar a memoria de dados.

k. MemWrite = 0

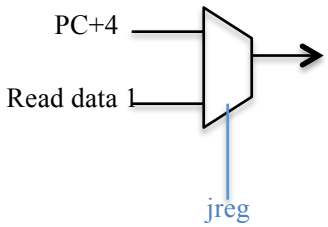
   Nunca são escritos dados da memória, i.e. as instruções de *store* não são corretamente executadas.

l. MemWrite = 1

   Erro na execução de **lw** – a memória teria MemRead = MemWrite = 1, causando falha da memória. Erro tambem em todas as outras instruções, em que o registo cujo número correspondesse aos bits 20-16 do código da instrução seria armazenado em memória.

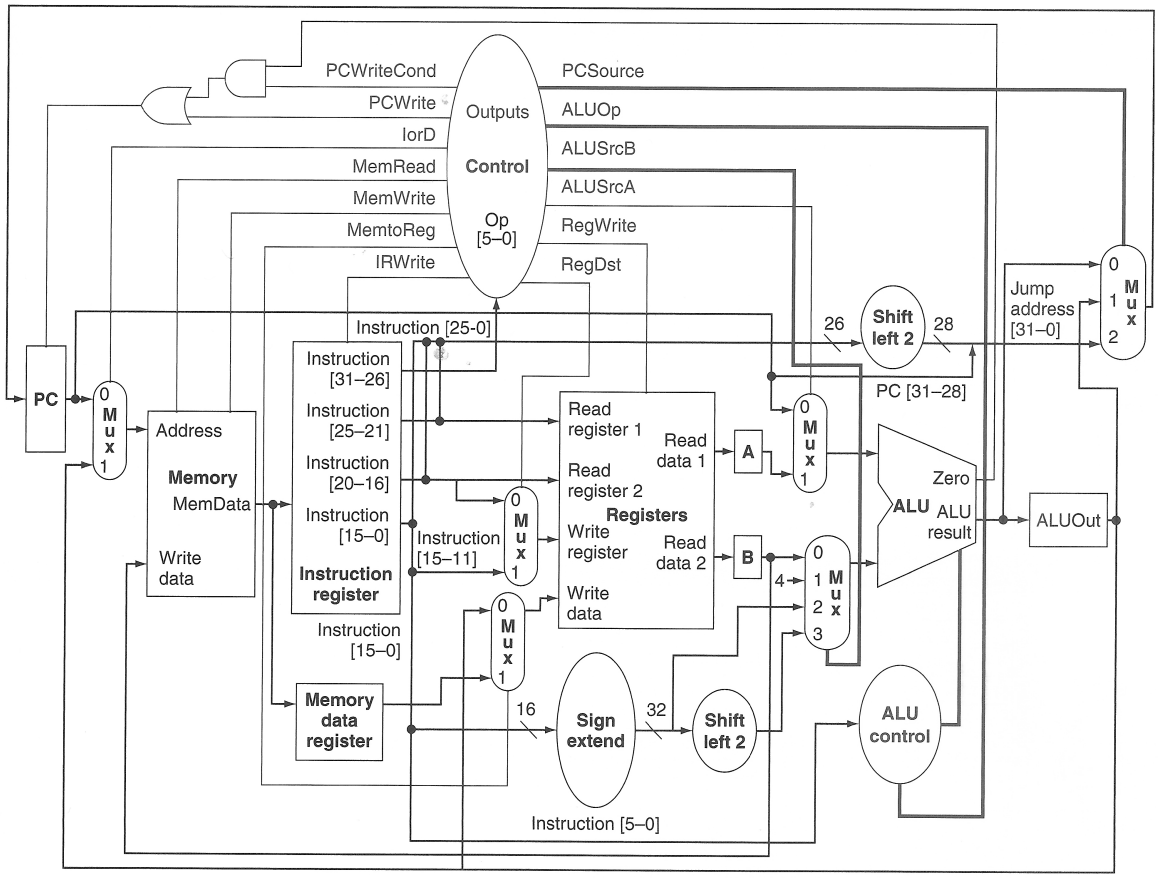
2. Pretende-se que o processador execute tambem a instrução ***jump register***. Indique o que seria necessário acrescentar ao datapath e as alterações a introduzir na tabela com os valores dos sinais de control (Nota: na instrução ***jr*** o campo rs do código de instrução indica o registo que contem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

Alteração a introduzir no diagram na ligação assinalada a vermelho:



jreg – sinal a gerar pela unidade de controle; jreg = 1 quando a instrução descodificada é ***jr***

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
      Nunca haveria armazenamento em registo do resultado das instruções aritméticas e lógicas nem dos dados lidos da memória (instruções de *load*), i.e. todas essas instruções corresponderiam a NOPs.
   b. MemRead = 0
      Nunca seria lido da memória o código das instruções – o processador não funcionaria.
   c. MemRead = 1
      Erro na execução de **sw** – a memória teria MemRead = MemWrite = 1, causando falha da memória.
   d. MemWrite = 0
      Erro na execução de **sw** – o conteúdo do registo $rt não seria armazenado na memória.
   e. MemWrite = 1
      No ciclo de *Fecth a* memória teria MemRead = MemWrite = 1, causando falha da memória. Nunca seria lido da memória o código das instruções – o processador não funcionaria.
   f. IRWrite = 0
      Nenhuma instrução seria executada pois não há armazenamento do código de instrução no IR.
   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 e um dado é interpretado, e executado, como uma instrução.
   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 memória.

i. PCWrite = 1

A instrução de jump executa corretamente. Nas instruções de Branch o salto efetua-se sempre, independentemente da respetiva condição ser verdadeira ou falsa. Em todas as outras instruções é armazenado no PC o resultado da ALU no ultimo ciclo de execução da instrução corrente, resultado esse que é indeterminado já que nesse ciclo a ALU não realiza trabalho util.

j. PCWriteCond = 0

Não se efetuariam saltos; seria sempre executada a instrução seguinte.

k. PCWriteCond = 1

As instruções de branch e jump executariam corretamente. Em todas as outras instruções haverá erro. Por exemplo, se PCSource se mantiver = 0, sempre que o resultado da operação efetuada pela ALU seja 0 é alterado o fluxo de execução, i.e. efetua-se um salto, sendo o endereço alvo o resultado da operação efetuada pela ALU (que poderá até corresponder a um endereço do segmento de dados da memória).

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.

Há que acrescentar 2 novos estados ao diagrama de estados, os estados 10 e 11. O estado 10 seria um novo successor do estado 1 e teria como unico successor o estado 11 cujo successor será o estado 0: 1 -> 10 -> 11 -> 0

Estado 10: ALUSrcA = 1, ALUSrcB = 10, ALUOp = 00

Estado 11: RegDst = 0, RegWrite = 1, MemtoReg = 0

Datapath mantem-se inalterado.

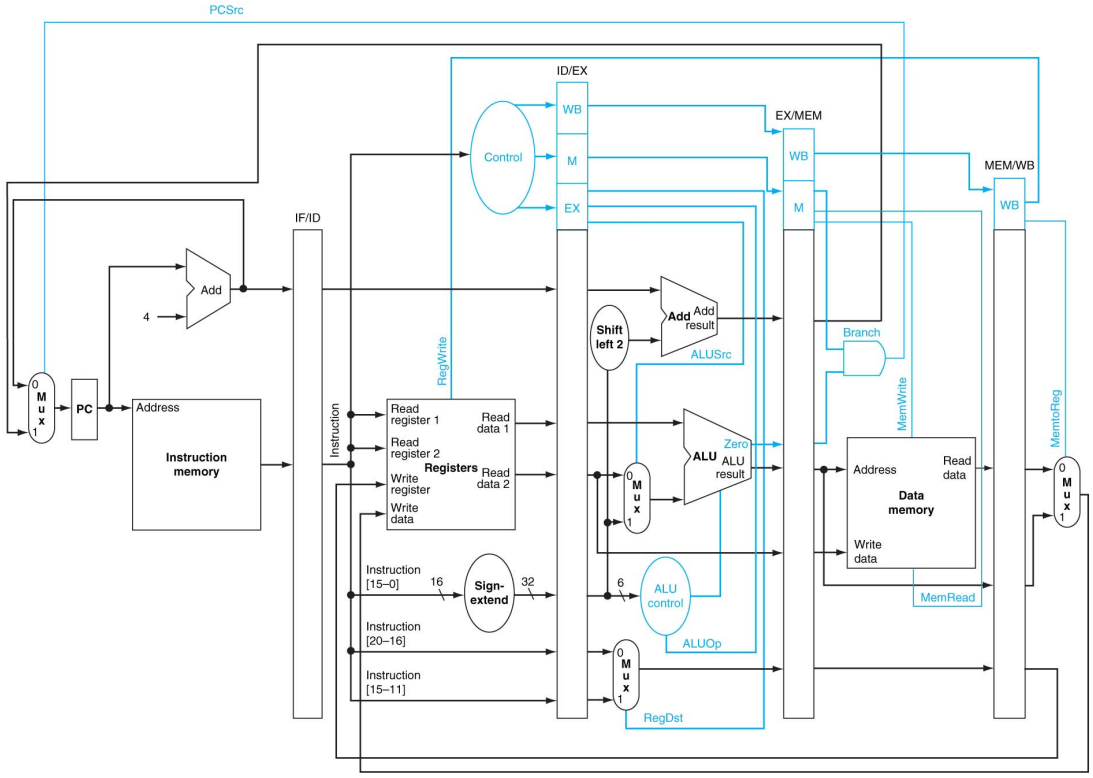
5. Assuma que a unidade de controle é microprogramada. Escreva um microprograma que implemente **addi**.

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

É tambem necessário acrescentar uma palavra à Dispatch ROM 1 no endereço correspondente ao opcode de addi (8_{hex}):

Address	**Value**
000000	0110
000010	1001
000100	1000
100011	0010
101011	0010
001000	**1010**

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      <- forwardA EX/MEM ALUresult
lw   $7,  100($5)      <- forwardA EX/MEM ALUresult
add  $8,  $7,  $2      <- forwardA MEM/WB Read data (após um ciclo de stall)
```

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

```
add  $3,  $4,  $2
sub  $5,  $3,  $1
lw   $6,  200($3)
add  $7,  $3,  $6
```

```
add  $3,  $4,  $2
sub  $5,  $3,  $1      é resolvida com forwarding do conteúdo do campo ALUresult de
EX/MEM para a entrada A da ALU
```

```
add  $3,  $4,  $2
sub  $5,  $3,  $1
lw   $6,  200($3)      também é resolvida com forwarding do conteúdo do campo
ALUresult de MEM/WB para a entrada A da ALU
```

```
lw   $6,  200($3)
add  $7,  $3,  $6      tem de se parar durante um ciclo a entrada de novas instruções no
pipeline, não deixando progredir a execução das instruções seguintes a lw (pipeline stall
– é introduzida uma "bubble"). No ciclo de relógio seguinte o campo de MEM/WB com o
valor lido da memoria é forward para a entrada B da ALU (forwardB).
```

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.

Ciclo 5:
Lw na fase Write back, Add na fase Memory, Or na fase Execute, And na fase Decode, Sub na fase Fetch

MEM/WB: WB = 1, MemtoReg = 0; Read data = 1052_{10}
EX/MEM: WB = 1, MemtoReg = 1, Branch = 0, MemWrite = MemRead = 0; ALU result = 25_{10}
ID/EX: WB = 1, MemtoReg = 1, Branch = 0, MemWrite = MemRead = 0, RegDst = 1,
ALUSrc = 0, ALUop = 10; PC = 96_{10}, Read data 1 = 12_{10}, Read data 2 = 11_{10},
SignExt = 00003825_{16}, Instruction[20-16] = 2, Instruction[15-11] = 7
IF/ID: PC = 100_{10}, IR = 00623020_{16}