Arquitetura de Computadores
Terceira Lista de Exercícios
Arquitetura de Ciclo Único e Pipeline
(Gabarito)

Prof. Norton Trevisan Roman

21 de outubro de 2019

1. (a)

Sinal	*Valor*	Sinal	Valor
RegWrite	1	MemRead	0
ALUSrc	0	MemWrite	0
ALUOp	and	MemToReg	0
Branch	0		

   (b) Bloco de registradores e ALU, além da memória de instruções e PC

   (c) Memória, somador do branch

   (d) Toda unidade produz saída

2. (a)

Sinal	*Valor*	Sinal	Valor
RegWrite	0	MemRead	0
ALUSrc	0	MemWrite	0
ALUOp	sub	MemToReg	0
Branch	1		

   (b) Bloco de registradores, ALU e somador do branch, além da memória de instruções e PC

   (c) Memória

   (d) Toda unidade produz saída

3. (a) Conjunto de registradores, ALU (para a soma) e memória de dados, além da memória de instruções e PC

   (b) Não precisamos de mais nada

   (c) Sinais não precisaremos, mas precisaremos sim mudar a lógica do circuito, adicionando um modo de `rd` ser lido junto com `rs` e `rt`

4. Para esse exercício, lembre que o `sw rt, desl(rs)` é codificado como

`101011`	`00011`	`00010`	`0000000000010100`
opcode	`rs`	`rt`	deslocamento

   (a) A extensão de sinal usa os bits `[15-0]` (`0000.0000.0001.0100`), estendendo o bit de sinal até termos 32b: `0000.0000.0000.0000.0000.0000.0001.0100`

   A saída da unidade de deslocamento de *jumps* (*shift left 2*) é tão somente o deslocamento à esquerda dos bits `[25-0]`: `0001.1000.1000.0000.0000.0101.0000`

   (b) A unidade de controle da ALU recebe os bits `[5-0]`, ou seja, `01.0100`, além dos 2 bits de ALUOp (`00`, conforme tabela na aula 06)

   (c) Será `PC+4`. O caminho é `PC` → `PC+4` → Mux do *branch* → Mux do *jump* → `PC`

   (d)

Mux dos Registradores	`2` se `RegDst=0` e `0` se `RegDst=1`
Mux da ALU	20 (bits `[0-15]` com extensão de sinal)
Mux da Memória	X (Não há como dizer, e não importa)
Mux do *Branch*	`PC+4`
Mux do `Jump`	`PC+4`

(e) Somador do `PC`: `PC` e 4
Somador do *Branch*: `PC+4` e 80 (bits `[15-0]` deslocados em 2 à esquerda)
ALU: -3 (`rs=3` ⇒ `$r3=-3`) e 20 (saída do Mux da ALU)

(f) RegWrite: 0
Read Register 1: `[25-21]` = 3
Read Register 2: `[20-16]` = 2
Write Register: 2 (se RegDst=0) ou 0 (`[15-11]`) se RegDST=1
Write Data: Não há como dizer a partir dos dados

5. Na implementação de ciclo único, o *datapath* precisa acomodar a instrução mais lenta, e uma só entra no *datapath* após a outra sair. Então o tempo entre uma instrução e outra é de 800ps.

   No caso da *pipeline*, e supondo a ausência de problemas, assim que uma instrução deixa a *pipeline* a seguinte já está entrando no último estágio. Então a diferença de tempo entre 2 instruções é o tempo gasto em cada estágio. Como todo estágio, em nosso modelo, leva o mesmo tempo, eles precisam ser projetados para acomodar o maior estágio. Assim, o tempo entre uma instrução e outra é de 200ps.

6. (a) A memória de dados somente é usada por `lw` e `sw`, ocupando então $25 + 10 = 35\%$ dos ciclos.

   (b) A extensão de sinal é usada para cálculo de endereço e deslocamento em `lw`, `sw`, `beq` e `addi`, ocupando então $25 + 10 + 25 + 20 = 80\%$ dos ciclos. Nos ciclos em que sua entrada não é necessária nada muda, pois ele continua computando esse valor.

7. (a) Sem *pipeline*, o período deve acomodar a instrução mais longa. Essa, no caso, é `lw`, que utiliza todos os estágios, totalizando 1.250ps. Com pipeline, o período deve acomodar o estágio mais longo, ou seja, 350ps.

   (b) Na implementação de ciclo único a instrução terá o tamanho do ciclo, que é de 1.250ps.
   Na implementação com *pipeline*, cada estágio possui um tempo único de 350ps e, como `lw` utiliza todos os 5 estágios, sua latência total será de $5 \times 350 = 1.750$ps

   (c) A memória de dados é usada por `lw` e `sw`, ocupando $20 + 15 = 35\%$ do total de ciclos

   (d) A porta de escrita do conjunto de registradores é usada toda vez que precisamos armazenar algo em um registrador, e isso serve para `lw` e todas as demais instruções (que não `beq` e `sw`), ocupando então $45 + 20 = 65\%$ dos ciclos