RELATÓRIO

Sara Gonçalves

Número mecanográfico: 98376

Laboratórios de Sistemas Digitais

Mestrado integrado em Engenharia de Computadores e Telemática

03/06/2020

Agradecimentos

Gostaria de agradecer ao Professor Manuel Violas pelas explicações nas aulas Práticas, ao Professor António Campos pelos ensinamentos nas aulas teóricas, e à Professora Iouliia Skliarova pelos vídeos disponibilizados na plataforma E-learning.

ÍNDICE

Página

Introdução

Este relatório tem como objetivo demonstrar a descrição e a interpretação dos resultados obtidos e a realização do projeto final de Laboratórios de Sistemas Digitais.

O projeto que me foi atribuído foi o Projeto nº 08. O propósito deste trabalho é modelar e simular um processador simplificado, que realiza um conjunto de operações sobre dois operandos guardados num registo de 8 bits, guardando num outro registo ou numa memória de dados. Os dados guardados são usados para serem processados ou armazenados.

Este possui uma unidade de execução (Datapath) e uma unidade de controlo (Control Unit), sendo que neste relatório irei explicar cada unidade e respetivos blocos e demonstrar a respetiva simulação.

A realização do projeto tem em conta 4 fases:

1. Modelação em VHDL, síntese e validação por simulação todos os blocos da Datapath;
2. Completar o diagrama de estados, modelar em VHDL, sintetizar e validar a Control Unit;
3. Interligar as duas unidades e testar no simulador o processador;
4. Adicionar uma instrução nova **BEQ** do tipo II, voltando a repetir os passos das fases anteriores;

Linguagem simbólica para código de máquina

```
LW   $0, $1, 0  -- carregar no registo 1 dados que se encontram no endereço [registo0+0] na memória de dados
LW   $0, $2, 1  -- carregar no registo 2 dados que se encontram no endereço [registo0+1] na memória de dados
ADD  $1, $2, $3 -- realizar a operação registo3 = registo1 + registo2
SGU  $3, $2, $4  --realizar a operação registo4 = registo3 SGU registo 2
SLL  $4, $3, $5 -- realizar a operação registo5 = registo4 SLL registo3
SW   $1, $4, 0  -- escrever no endereço [registo1+0] na memória de dados o conteúdo do registo4
SW   $1, $5, 1  -- escrever no endereço [registo1+1] na memória de dados o conteúdo do registo5
```

A memória de instruções é inicializada com estes valores, sendo que têm que ser passados da linguagem *Assembly* para o código de máquina, através das tabelas 1 e 2. Sabe-se que as instruções são de 16 bits.

Tabela 2 – Operações suportadas pela ALU.

func	Operação
0000	**ADD** – soma
0001	**SUB** – subtração
0010	**AND**
0011	**OR**
0100	**XOR**
0101	**NOR**
0110	**MUU** - multiplicação sem sinal (só é usada a metade menos significativa do resultado)
0111	**MUS** - multiplicação com sinal (só é usada a metade menos significativa do resultado)
1000	**SLL** - deslocamento lógico do registo$_{rs}$ à esquerda registo$_{rt}$ bits
1001	**SRL** - deslocamento lógico do registo$_{rs}$ à direita registo$_{rt}$ bits
1010	**SRA** - deslocamento aritmético do registo$_{rs}$ à direita registo$_{rt}$ bits
1011	**EQ** – *equal* - o resultado é 1 se registo$_{rs}$ = registo$_{rt}$
1100	**SLS** – *set less than signed* – o resultado é 1 se *signed*(registo$_{rs}$) < *signed*(registo$_{rt}$)
1101	**SLU** – *set less than unsigned* – o resultado é 1 se *unsigned*(registo$_{rs}$) < *unsigned*(registo$_{rt}$)
1110	**SGS** – *set greater than signed* – o resultado é 1 se *signed*(registo$_{rs}$) > *signed*(registo$_{rt}$)
1111	**SGU** – *set greater than unsigned* – o resultado é 1 se *unsigned*(registo$_{rs}$) > *unsigned*(registo$_{rt}$)

opcode	Instrução
000	**NOP** – não fazer nada
001	Todas as instruções aritméticas ou lógicas (a operação a executar é definida pelo campo *func*) – tipo I
100	**ADDI** - soma o conteúdo dum registo com uma constante – tipo II
110	**SW** – transferir dados dum registo para a memória de dados – tipo II
111	**LW** – transferir dados da memória de dados para um registo – tipo II

Assembly	Tipo	Opcode	rs	rt	rd	func	address	Código de máquina
LW $0, $1, 0	II	111	000	001			0000000	1110000010000000
LW $0, $2, 1	II	111	000	010			0000001	1110000100000001
ADD $1, $2, $3	I	001	001	010	011	0000		0010010100110000
SGU $3, $2, $4	I	001	011	010	100	1111		0010110101001111
SLL $4, $3, $5	I	001	100	011	101	1000		0011000111011000
SW $1, $4, 0	II	110	001	100			0000000	1100011000000000
SW $1, $5, 1	II	110	001	101			0000001	1100011010000001

A memória de Dados é inicializada com os valores

```
(X"02", X"01", others => X"00").
```

Fase 1

Blocos da Datapath

A unidade de execução é constituída por vários blocos:

- PC: Contador que guarda o endereço da próxima instrução a ser executada;

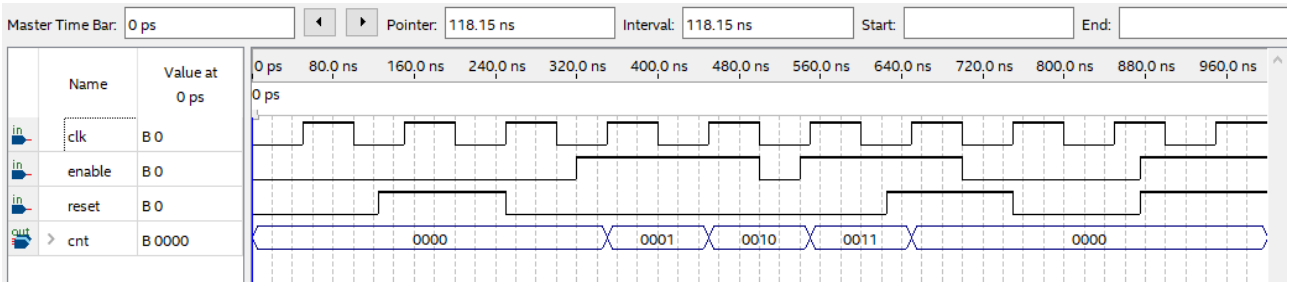


Fig 1: Simulação em vwf do contador

Contador simples realizado nas aulas práticas, onde o cnt aumenta se num pico de relógio o enable estiver ativo, e volta a '0' se num pico de relógio o reset estiver ativo.

- SignExtend : realiza a extensão de sinal de um vetor de 7 bits para um de 8 bits

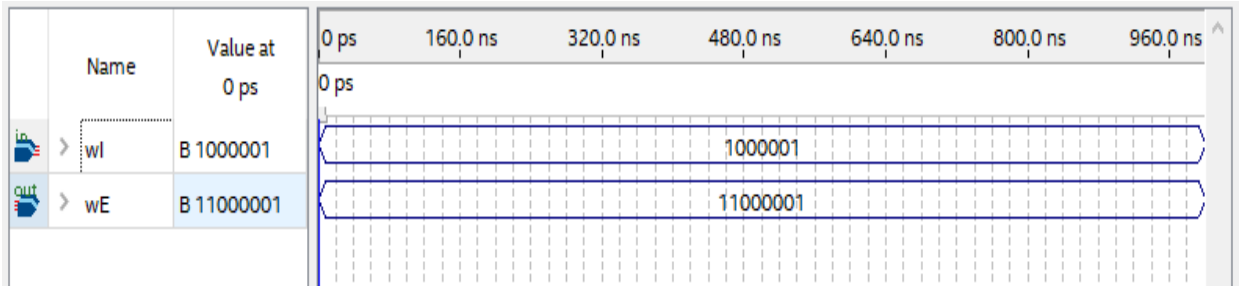


Fig 2: Simulação em vwf do SignExtend

O SignExtend realiza uma extensão de acordo com o bit mais significativo, como neste exemplo era '1', então vai aumentar para 8 bits "copiando" o '1'.

- ALU: realiza as operações lógicas e aritméticas;

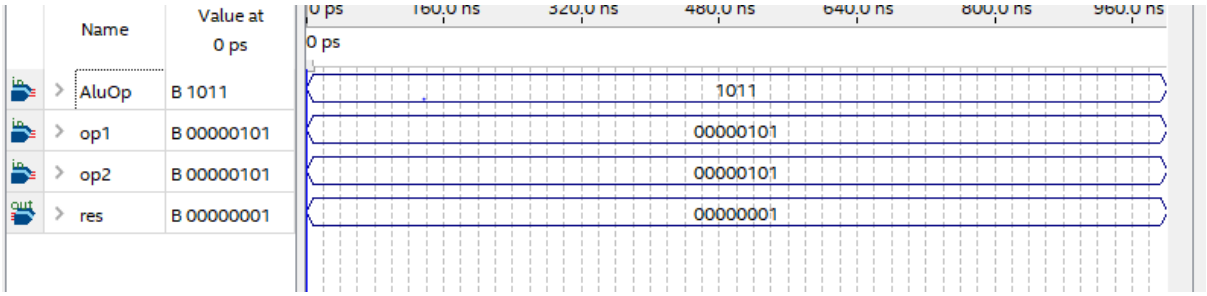


Fig3: Exemplo da simulação da ALU

Neste exemplo, percebemos que se a AluOp for ‘1011’ vai realizar a operação Equal. Significa que quando o operando op1 e o operando op2 são iguais (como neste caso), o resultado res vai ser igual a ‘1’.

- DMemory :RAM que guarda os dados de um programa

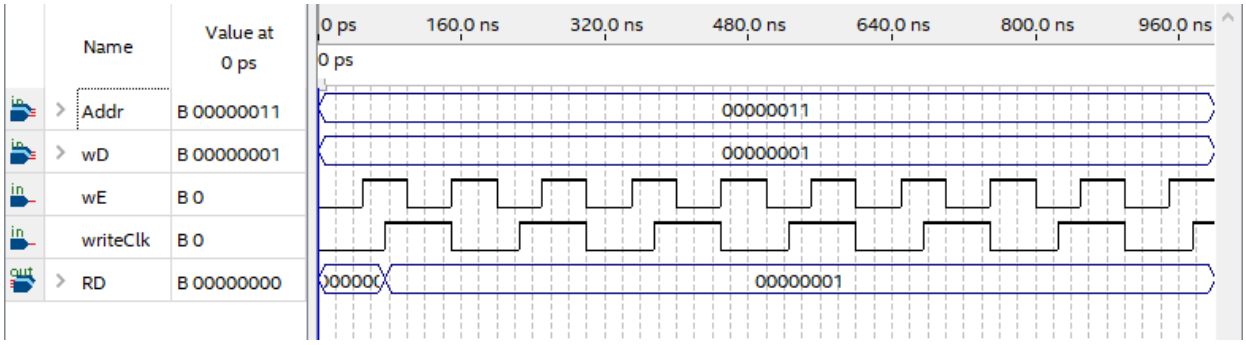


Fig4: simulação da DMemory

Recebe os valores do wD e armazena, dependendo do pico do writeClk.

- IMemory: ROM que guarda as instruções do programa

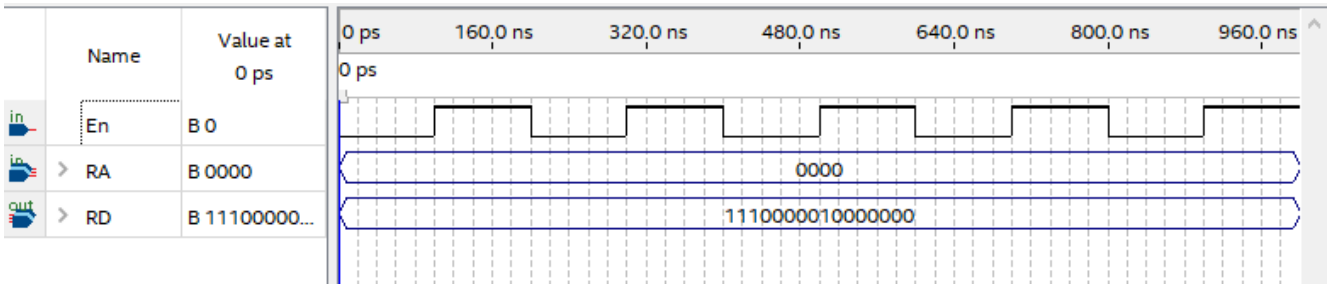


Fig5: Exemplo da simulação da IMemory

Neste exemplo estamos perante a um RA ‘0000’ onde RD é o primeiro valor da constante da ROM.

- Bloco de Registos

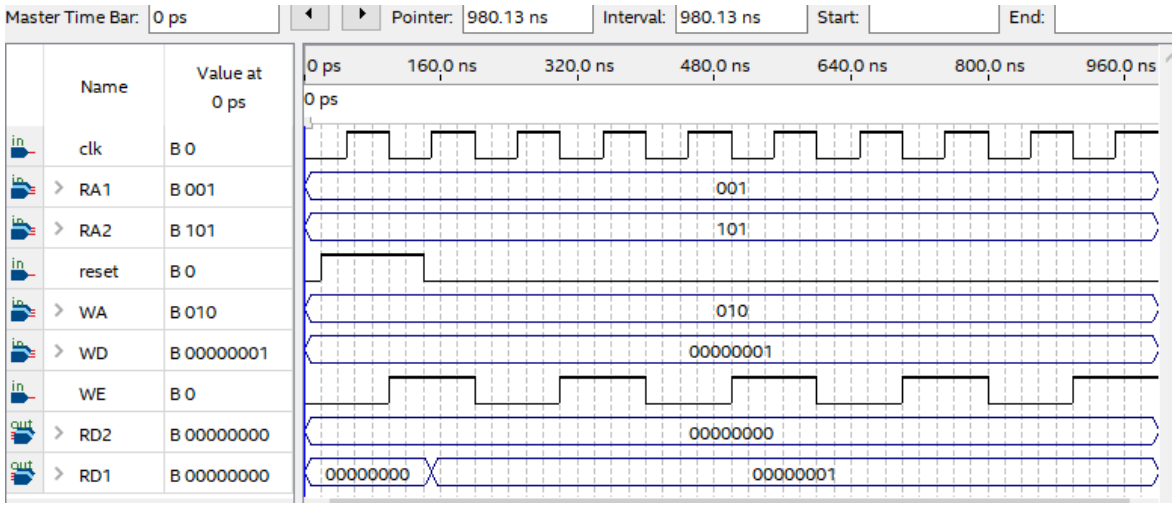


Fig6: Exemplo da simulação do Register

O bloco de registos é um descodificador 3_8 cuja entrada corresponde ao endereço do registo onde vai escrever e 8 aos outputs ligam à enable dos registos que selecionam de acordo com WA, juntamente com um registo de 8 bits com clk, reset e enable.

- 3 Multiplexers 2:1 já realizados numa aula prática;

A unidade de execução é assim, simulada:

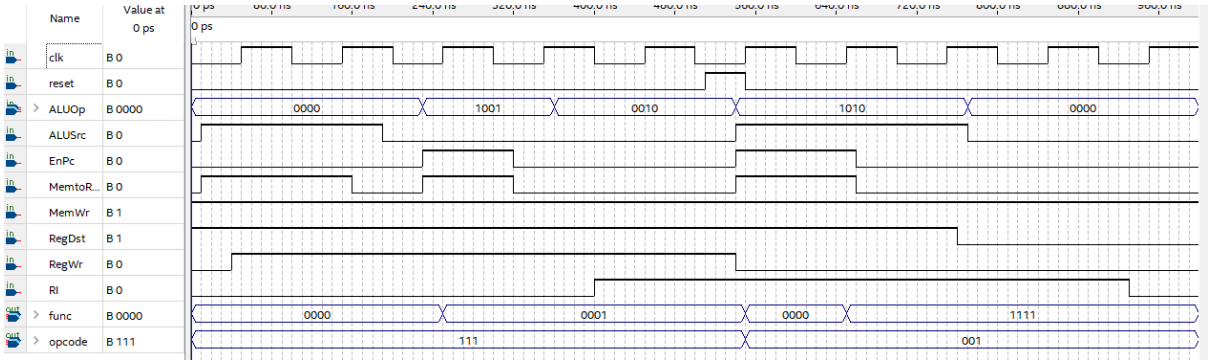


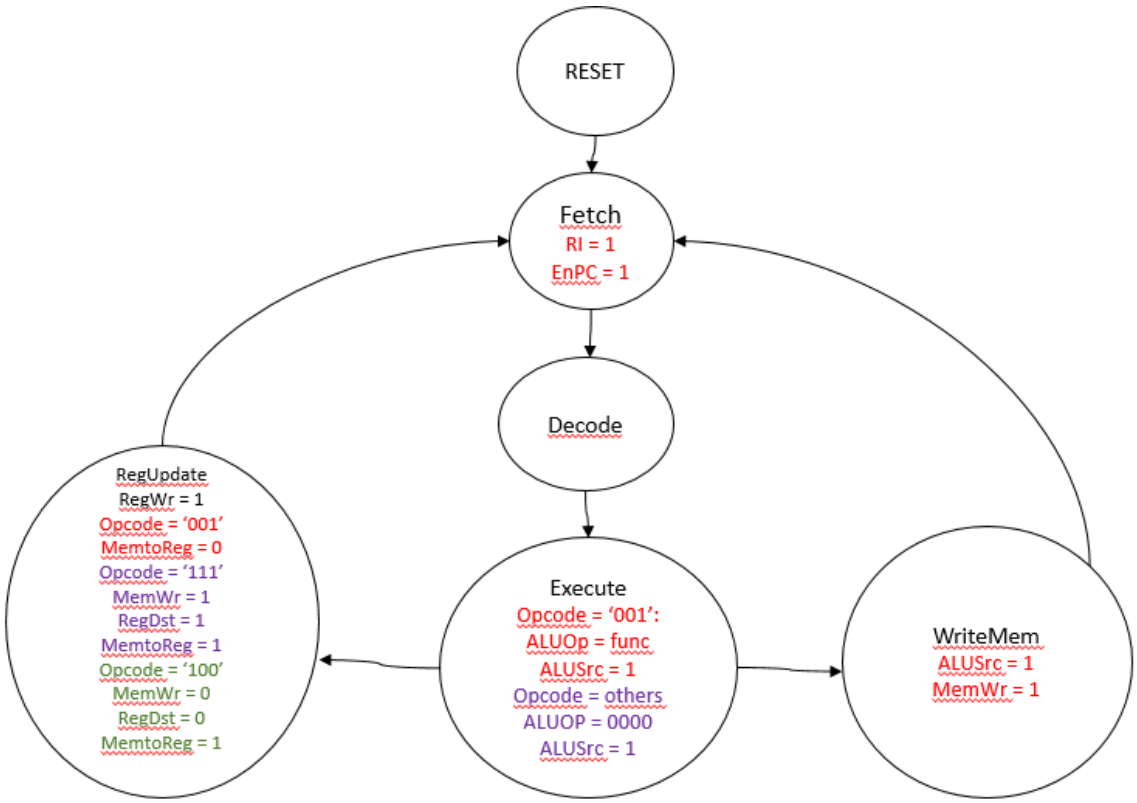
Fig7: Exemplo da simulação do DataPath

Considero, no entanto, que apesar de todos os blocos estarem bem simulados, a interligação da Datapath não deve estar totalmente correta, sendo que esta simulação não está completa.

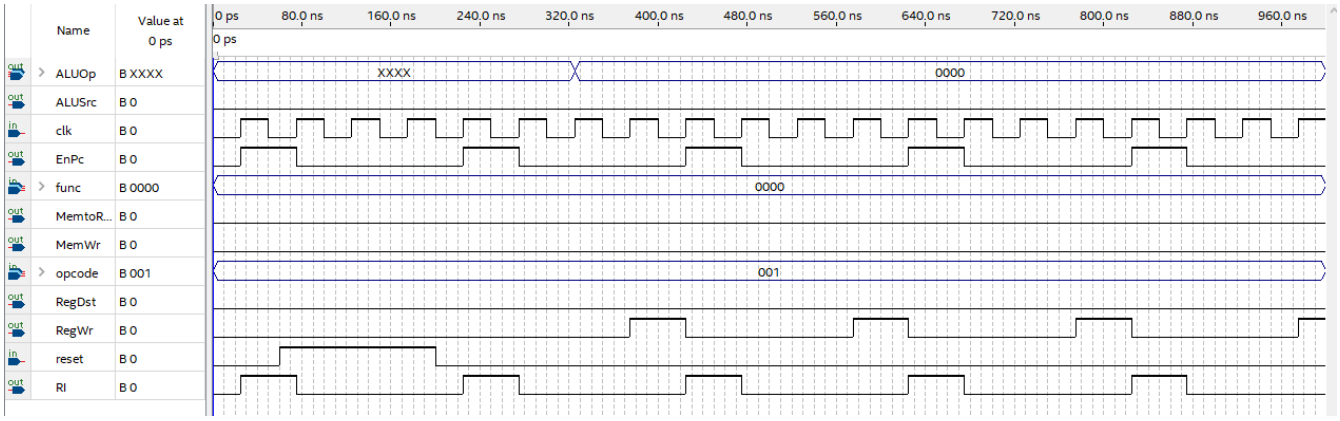
Fase 2

ControlUnit

O diagrama de estados da unidade de controlo específica todos os sinais de controlo ativos para cada um dos estados.



Neste caso, o estado "Decode" é um campo de espera e, por isso, não é especificada nenhuma saída durante o mesmo.



Simulação da ControlUnit através do Quartus

Fase 3

O processador

Nesta terceira fase, interliga-se as duas unidades realizadas nas fases anteriores.

Neste ficheiro as saídas do bloco ControlUnit vão servir de sinais de controlo para a Datapath.

Apesar de ter criado em vhdl uma TestBench para o processador, na prática não consegui simular com o modelSim.

Fase 4

Exemplo da instrução BEQ:

BEQ $rs , $rt , 3

BEQ $1 , $2 , 3 = 010 001 010 0000100

Conclusão

Em conclusão, considero que apesar de a simulação do processador não estar totalmente certa, considero os códigos do Datapath e a simulação destes blocos correta de acordo com os objetivos que é pedido. No entanto após uma leitura intensiva do que era pedido na Fase 4, não consegui criar a nova instrução de modo a que conseguisse compilar os blocos todos. Na minha opinião, uma nota positiva é justificável, apesar de não se saber a cotação de cada fase.