Projeto Final de LSD

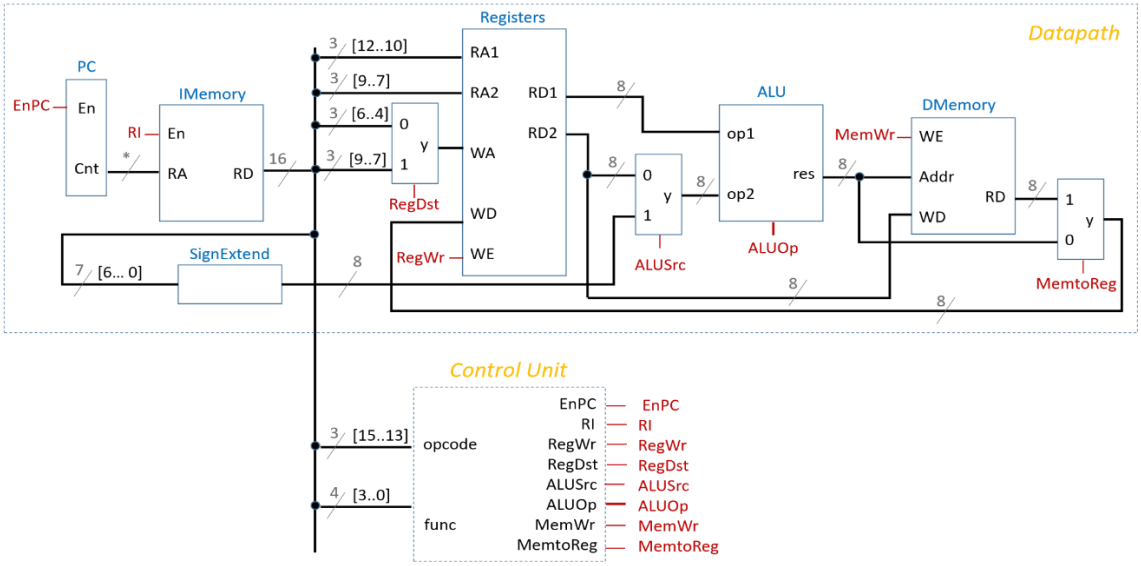
Modelação de um processador simplificado

João Gil Ferreira de Sousa Fernandes nmec: 93460

Índice

Introdução

Este projeto tem como objetivo modelar um processador com base na linguagem VHDL. Foi-me disponibilizado um diagrama com todos os blocos necessários para o funcionamento deste processador.



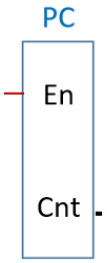
Para desenvolver e executar o projeto, concebi, no programa Quartus, cada bloco de acordo com o solicitado e, de seguida, liguei-os, de forma a obter um processador funcional. Esse processado é todo ele parametrizável.Por defeito, o processador contém um máximo de 8 instruções de 16 bits na sua ROM (IMemory) e pode armazenar até 256 palavras de 8 bits na sua RAM (DMemory).

Fase 1

Estrutura do DataPath

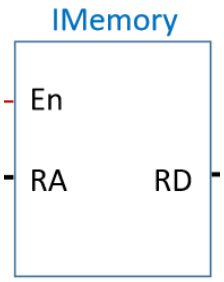
PC

- Contador que guarda um endereço, inicialmente de 3 bits, pois é parametrizável, da próxima instrução a ser executada.



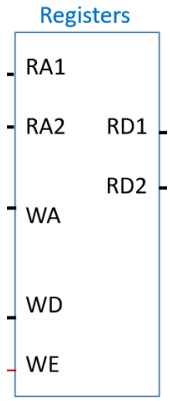
IMemory

- Memória ROM que guarda as instruções que podem ser executadas pelo programa.
- A quantidade de instruções pode ser alterada.
- Cada instrução consiste numa palavra de 16 bits.
- Recebe um endereço (RA), que é transmitido pelo PC.



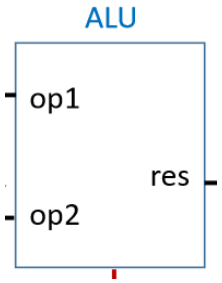
Registers

- Registo que contém até 8 palavras (parametrizável) de 8 bits cada.
- A palavra que se encontra no endereço 0 será sempre igual a 0, não podendo ser alterada posteriormente.
- Recebe instruções para ler dois endereços (RA1 e RA2), uma para escrever um endereço (WA) e uma para escrever uma palavra de 8 bits (WD) no endereço WA.
- Os outputs (RD1 e RD2) são palavras de 8 bits guardadas nos endereços RA1 e RA2 respetivamente.



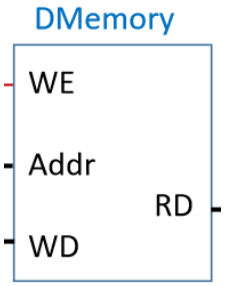
ALU

- Bloco que contém as operações logicas e aritméticas.
- Recebe duas palavras de 8 bits e uma instrução que "dirá" qual das funções executar.
- O op1 e o op2 são as palavras recebidas e o res é o resultado das duas palavras com a operação "func".



DMemory

- Memória RAM que guarda até 256 palavras (parametrizável) de 8 bits cada.
- Guarda dados fornecidos pelo bloco Registers.
- É inicializada com os valores x"F5" e x"32".
- O endereço (Addr) é o resultado do res do ALU.
- A palavra é transmitida através do RD2 do Registers (liga-se ao WD) e fica guardada no endereço (Addr) quando o WE estiver ativo.
- O RD é a leitura da palavra no endereço (Addr).

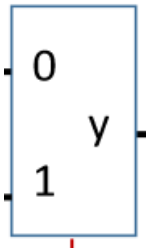


SignExtend

- Recebe uma palavra de 7 bits e “estende-a”, formando uma palavra de 8 bits.

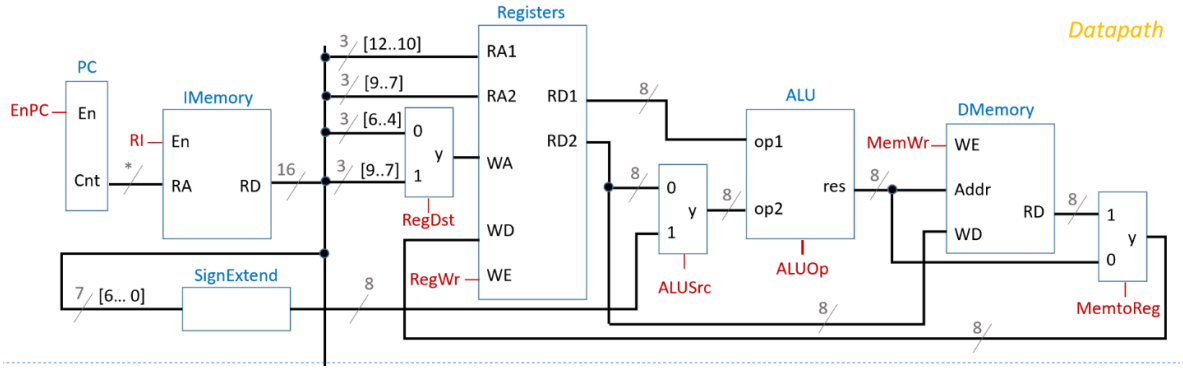
Multiplexers 2:1

- Existe um total de 3 multiplexers 2:1 neste diagrama de blocos.
- Um trabalha com duas palavras de 4 bits fornecidas pelo IMemory.
- Dois usam palavras de 8 bits: um recebe informação da saída RD2 do Registers e do SignExtend; o outro recebe o resultado do ALU e um valor da RAM DMemory.



DataPath

- O DataPath viabiliza a ligação de todos os blocos mencionados anteriormente.
- É o bloco de execução do processador, ou seja, é o que vai efetuar as operações e o armazenamento de dados.



- As operações executáveis pelo DataPath estão distribuídas por dois tipos: Tipo I e Tipo II.
  Cada um dos tipos tem instruções repartidas de diferentes formas:

Tipo I:

3 bits	*3 bits*	*3 bits*	*3 bits*	*4 bits*
opcode	rs	rt	rd	func

Tipo II:

3 bits	*3 bits*	*3 bits*	*7 bits*
opcode	rs	rt	address

- Os primeiros 3 bits são os que definem se se trata de uma operação do Tipo I, do Tipo II ou se não se intervém, como mostra a tabela seguinte:

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

- "000" -> NOP, não realiza quaisquer operações.
- "001" -> realiza funções lógicas/aritméticas que estão contidas no bloco ALU.
- "100" -> ADDI, soma o conteúdo de um registo (rs) com uma constante (address), guardando o resultado em (rt). Neste caso específico, o address servirá como valor inteiro e não como endereço da RAM.
- "110" -> SW, lê o registo rt e guarda-o na memória DMemory na posição dada pelo address.
- "111" -> LW, lê o dado da DMemory (rs) na posição address e armazena-a no Registers na posição rt.
- Quando a instrução "opcode" é "001", a instrução "func" efetuará as operações lógicas/aritméticas indicadas na tabela seguinte:

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$_{rt}$)

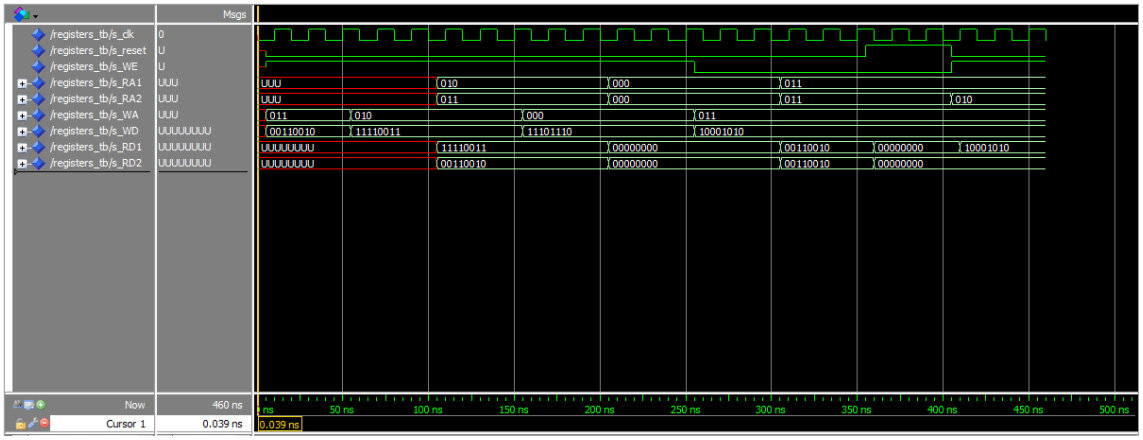
- Com estas tabelas já conseguimos traduzir o assembly para código de máquina.

Tradução de assembly para linguagem máquina das seguintes instruções:

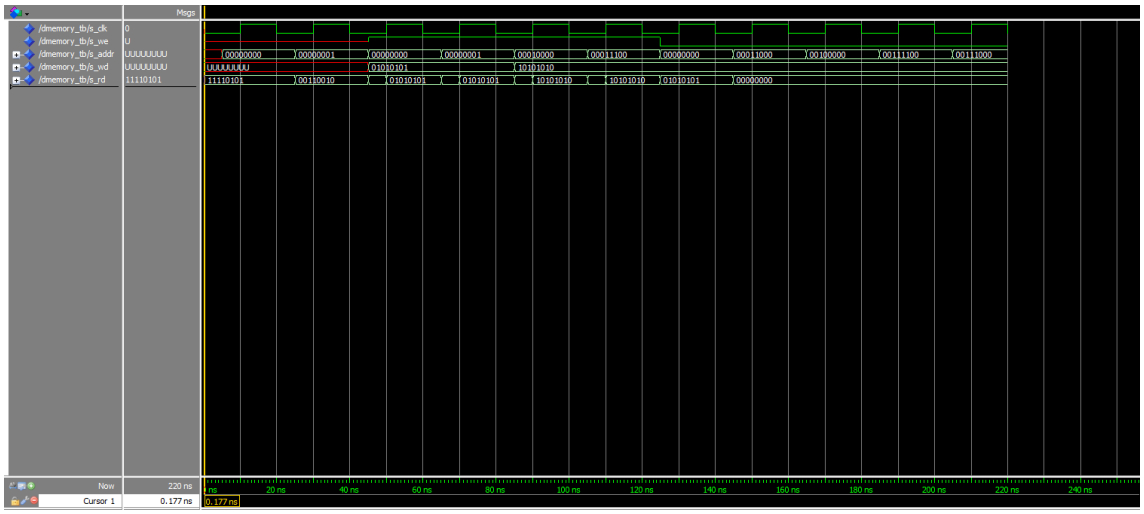
Assembly	Código de máquina
LW $0, $1, 0	-> 1110000010000000
LW $0, $2, 1	-> 1110000100000001
OR $1, $2, $3	-> 0010010100110011
XOR $1, $1, $4	-> 0010010011000100
SW $0, $3, 2	-> 1100000110000010
SW $0, $4, 3	-> 1100001000000011

Validação dos blocos

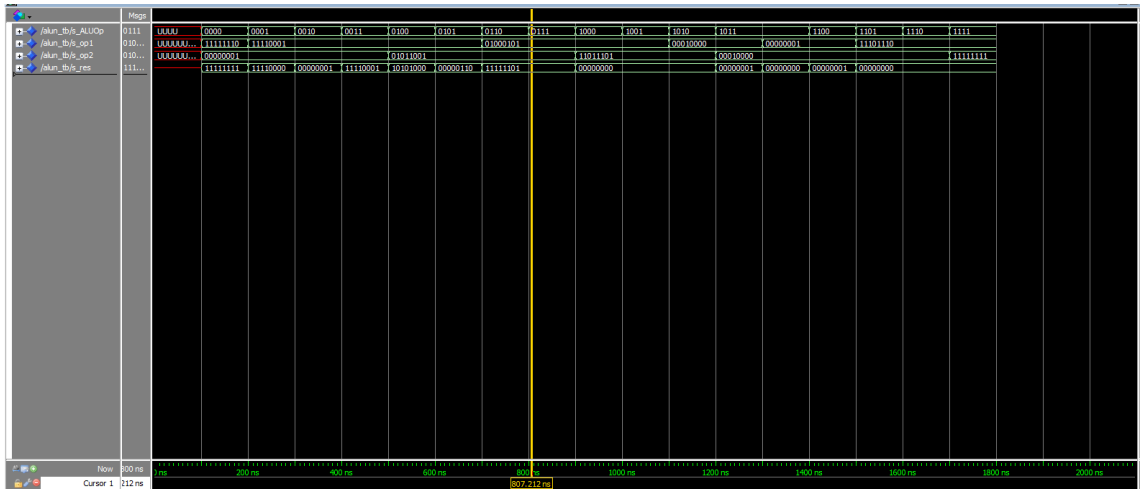
Execução do Registers



Execução do DMemory

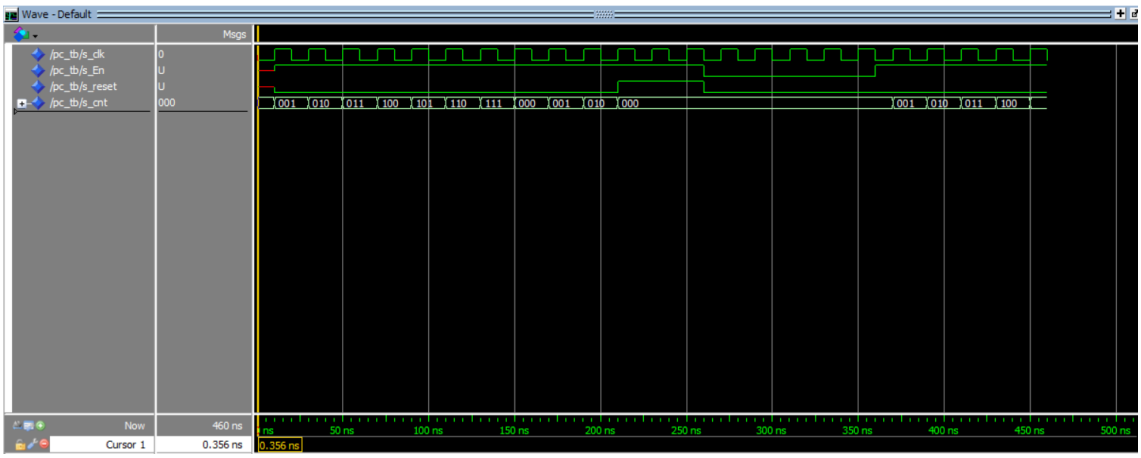


Execução do ALUN



Execução do IMemory

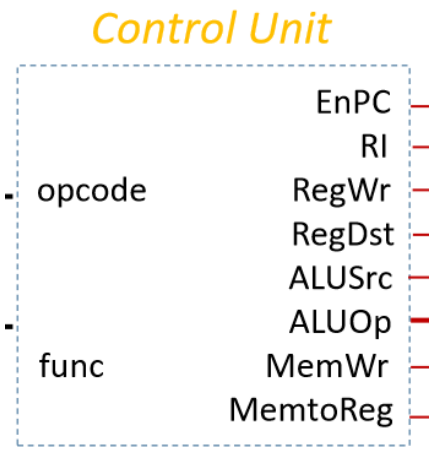
Execução do PC



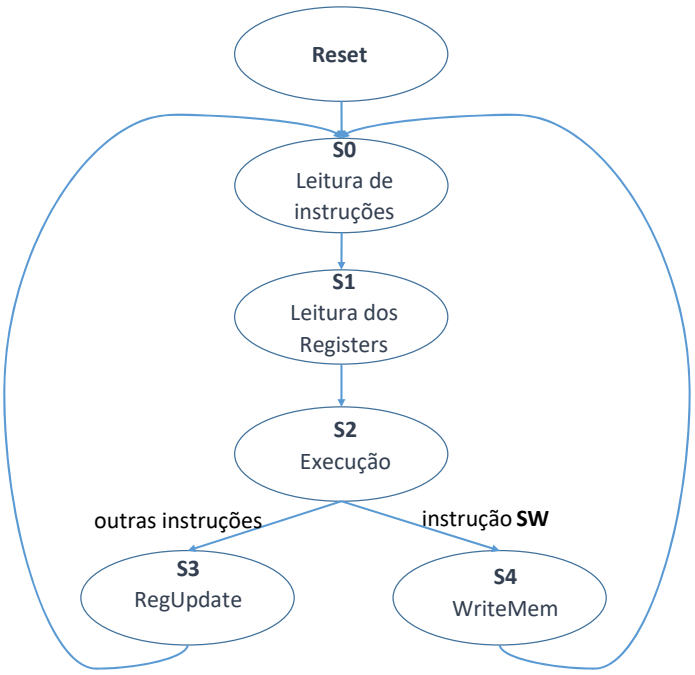
Fase 2

ControlUnit

- Trata-se de uma máquina de estados finitos composta por 5 estados.
- Recebe o opcode e func da IMemory e controla todo o processo com base nessas duas operações.



Máquina de estados do ControlUnit



Legenda:

- Leitura de Instruções – Lê os dados da IMemory relativos à instrução a executar. Atualiza o PC. Apenas o PC e o IMemory estão ativos nesta fase.

```
when S0 =>
    EnPc <= '1';
    RI <= '1';
    RegWr <= '0';
    RegDst <= '0';
    ALUSrc <= '0';
    ALUOp <= "0000";
    MemWr <= '0';
    MemtoReg <= '0';
    reg_fstate <= S1;
```

- Leitura dos Registers – Lê os conteúdos dos Registers. Este estado apenas lê conteúdos, ou seja, não tem nenhum enable ativo.

```
when S1 =>
    EnPc <= '0';
    RI <= '0';
    RegWr <= '0';
    RegDst <= '0';
    ALUSrc <= '0';
    ALUOp <= "0000";
    MemWr <= '0';
    MemtoReg <= '0';
    reg_fstate <= S2;
```

- Execução – Realiza a operação indicada pelo opcode (SW, LW, ADDI ou operações do Tipo I). Nesta fase, utilizei um case para o opcode para cada instrução acima especificada. Para este estado, criei um “case” para o opcode.

```
when "000" =>--NOP
    EnPc <= '0';
    RI <= '0';
    RegWr <= '0';
    RegDst <= '0';
    ALUSrc <= '0';
    ALUOp <= "0000";
    MemWr <= '0';
    MemtoReg <= '0';
    reg_fstate <= S0;
```

```
when "001"=>--funcoes ALU
    EnPc <= '0';
    RI <= '0';
    RegWr <= '0';
    RegDst <= '0';
    ALUSrc <= '1';
    ALUOp <= func;
    MemWr <= '0';
    MemtoReg <= '0';
    reg_fstate <= S3;
```

```
when "100" =>--ADDI
    EnPc <= '0';
    RI <= '0';
    RegWr <= '0';
    RegDst <= '0';
    ALUSrc <= '1';
    ALUOp <= "0000";
    MemWr <= '0';
    MemtoReg <= '0';
    reg_fstate <= S3;
```

```
when "110" =>--SW
    EnPc <= '0';
    RI <= '0';
    RegWr <= '0';
    RegDst <= '0';
    ALUSrc <= '1';
    ALUOp <= "0000";
    MemWr <= '0';
    MemtoReg <= '0';
    reg_fstate <= S4;
```

```
when "111"=>--LW
    EnPc <= '0';
    RI <= '0';
    RegWr <= '0';
    RegDst <= '0';
    ALUSrc <= '1';
    ALUOp <= "0000";
    MemWr <= '0';
    MemtoReg <= '0';
    reg_fstate <= S3;
```

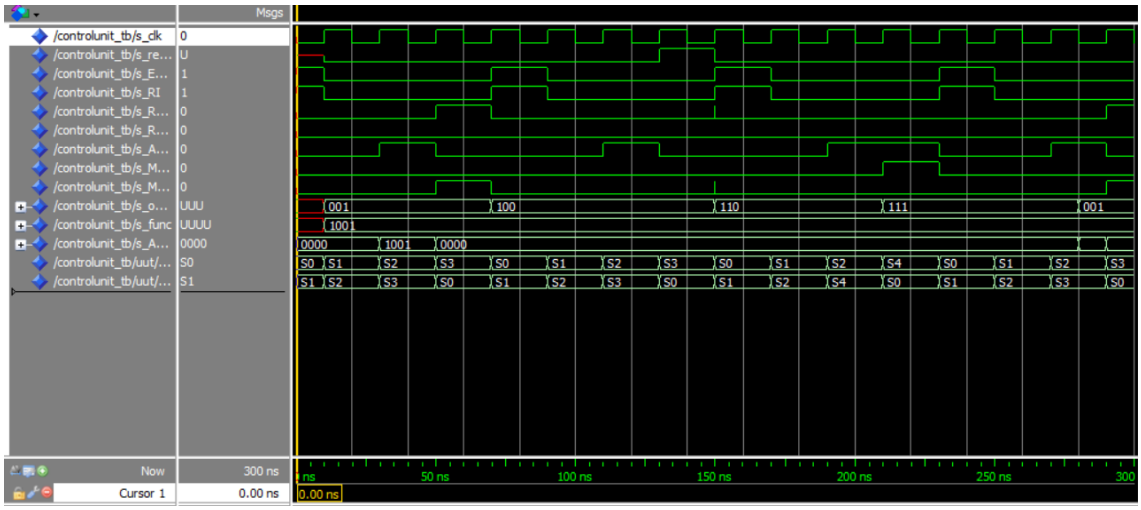
- RegUpdate – Guarda o resultado no Registers (funciona com qualquer instrução, exceto a SW). Estão ativos os sinais de escrita do register e do mux que levam a informação da RAM para ser escrita no Registers.

```
when S3 =>--RegUpdate
    EnPc <= '0';
    RI <= '0';
    RegWr <= '1';
    RegDst <= '0';
    ALUSrc <= '0';
    ALUOp <= "0000";
    MemWr <= '0';
    MemtoReg <= '1';
    reg_fstate <= S0;
```

- WriteMem – Guarda o resultado na DMemory. Este estado apenas é ativado se a instrução usada no opcode for SW (“110”). Estão ativos os sinais de escrita da RAM e do mux que levam o address até ao ALU, depois de ter passado pelo SingExtend.

```
when S4 =>--WriteMem
    EnPc <= '0';
    RI <= '0';
    RegWr <= '0';
    RegDst <= '0';
    ALUSrc <= '1';
    ALUOp <= "0000";
    MemWr <= '1';
    MemtoReg <= '0';
    reg_fstate <= S0;
```

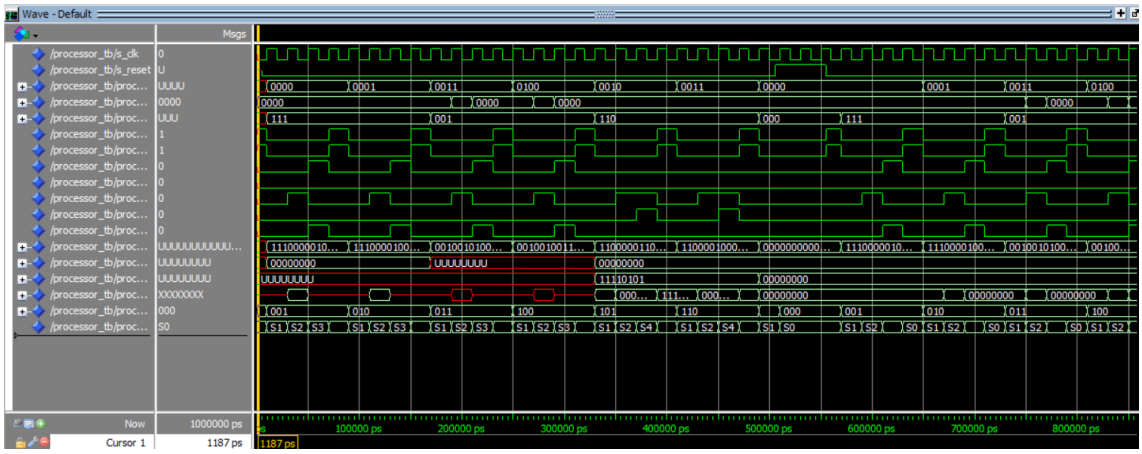
Execução do ControlUnit



Fase 3

- Nesta terceira fase, procedi à junção do bloco DataPath com o bloco ControlUnit, finalizando assim o processador.
- O bloco do processador terá apenas duas entradas, o clock e o reset global. O resto dos sinais são apenas as interligações entre os blocos ControlUnit e DataPath.
- O reset global elimina o conteúdo dos registos e coloca o PC a 0.

Execução do Processador



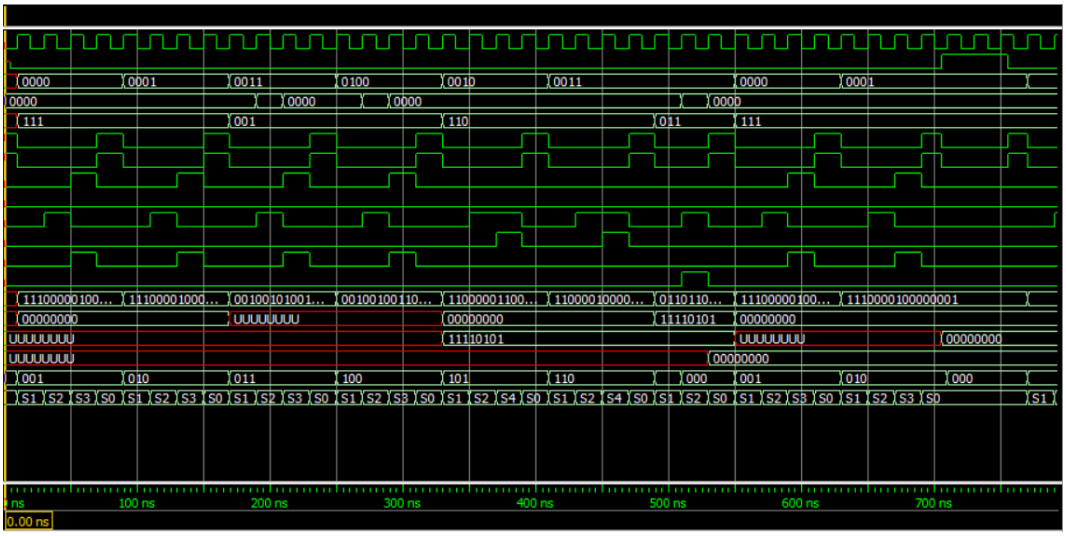
Fase4

- Nesta última fase, adicionei uma nova instrução, a BNE (branch or not equal)
- Esta instrução é ativada quando o opcode for 011 e é uma instrução do Tipo II.
- Esta instrução irá ativar a instrução “1011” do func para comparar os dois registos escolhidos.
- Se os registos escolhidos forem iguais, ou seja, se o “res” do ALU for igual a “00000001”, então, o contador irá saltar o número igual de instruções dado pelo address.

Planeamento

Nesta última fase, comecei por criar um bloco ao qual dei nome de PC_acre. Este bloco ira receber o res (output do ALU), o output do SignExtend e um enable que será ativado pelo ControlUnit assim que for ativada esta instrução. Como saída apenas possuí uma que irá estar ligada ao PC, permitindo assim fazer o “salto” de “address” instruções. O PC irá receber um aumento “address” se a saída “res” do ALU for igual a “00000001” e se o enable do PC_acre estiver ativo. Caso contrário o PC irá receber “0” como valor do PC_acre.

Execução da fase 4



Conclusão

A meu ver, este projeto foi ao encontro de todos os objetivos propostos. Instanciei os blocos pedidos, testei cada bloco individualmente para ter a certeza que todos estavam a funcionar devidamente. Finalmente juntei todos os blocos como demonstra a figura dada no enunciado do projeto finalizando a fase 1. Na fase 2 concebi uma máquina de estados que compila e funciona como era esperado. Na fase 3 fiz a junção do Datapath da fase 1 com a máquina de estados da fase 2. Finalmente na fase 4 modifiquei o processador de forma a aceitar a nova instrução pedida no enunciado. Nesta última parte alguns resultados não deram como era esperado.

Em relação à autoavaliação, acho que mereço um 16 pois houve resultados que não deram conforme o esperado.