**Relatório**

Laboratório Sistemas Digitais

Marta Oliveira,nº 97613

1 de junho

2019/2020

# Introdução

O objetivo deste trabalho é fazer um processador que realiza operações aritméticas/lógicas sobre dois operandos(rs e rt).

Vão existir apenas 8 registos daí ter criado 8 registos em vhdl (*Reg0,Reg2….*).

Os dados vão ser guardados na memoria de dados, *DMemory*. Estes dados são transferido para os tais registos para poderem ser processados. O sentido inverso serve para armazenar.

# Fase1

A memória de instruções vai armazenar no máximo 8 instruções do tipo: {LW,SW,ADD,ADDI,XOR,EQ}.

A linguagem assembly que vai ser codificada é:

--LW $0, $1, 0

--LW $0, $2, 1

--LW $0, $3, 2

--XOR $1, $2, $4

--ADDI $4, $5, 1

--ADD $5, $3, $1

--SW $0, $1,4

O processador é capaz de executar 2 tipos de instruções:

**Tipo I:** opcode| rs |rt| rd| func => ADD,SUB,XOR

**Tipo II**: opcode| rs | rt | address =>SW,LW,ADDI

Nas instruções do tipo XOR, por exemplo, o opcode será 001(tabela1) e a func será 0100. No meu caso a instrução XOR tem o seguinte código Assembly:

$1=001

$2=010 **logo**(XOR $1, $2, $4) será "001 001 010 100 0100"

$4=100

Nas instruções LW o opcode(de acordo com a tabela1) é 111. De seguida vamos descodificar o rs,rt e o adress.

$0= 000

$1=001 **logo(** LW $0, $1, 0) irá ter o seguinte código de máquina: "1110000010000000"

0=0000000

A lógica foi a mesma para as restantes instruções.

**Simulação e validação dos resultados obtidos:**

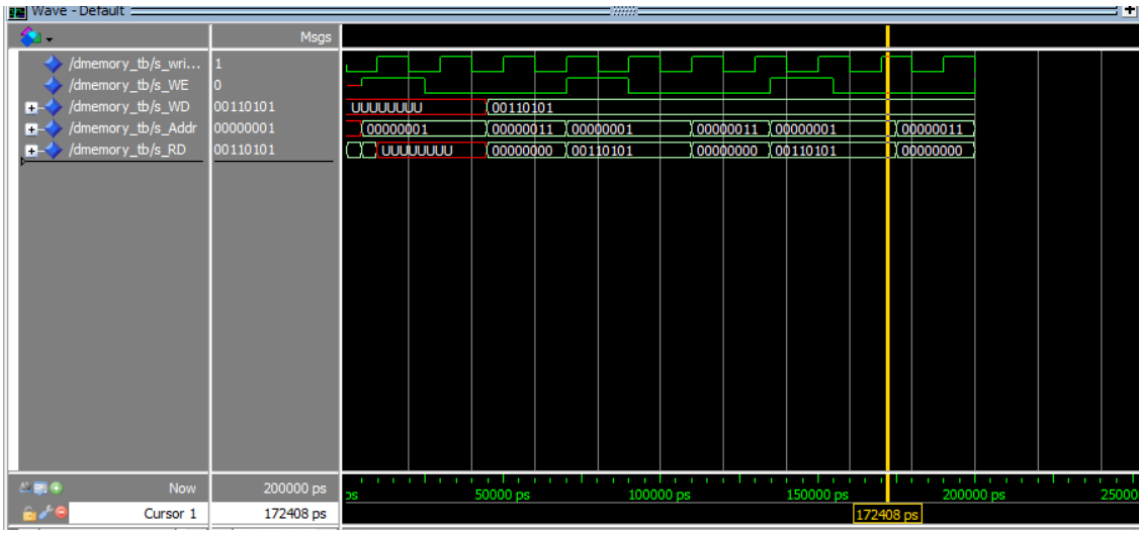


*Figura : simulação Dmemory*

A mémoria de dados deste projeto vai ser inicializada com os valores X”FF”, X"0A",X"05”.

A Dmemory só iniciara quando o enable estiver a 1. Depois disso o Addr foi declarado como 00000001 na testBench e por isso o resultado foi 0A em binário pois foi buscar a minha segunda instrução à Dmemory.

O primeiro valor RD encontra-se por omissão e será X”FF”.

Todos os outros address terão valor 0 (a não ser que atribuíssemos por opcão algum WD).

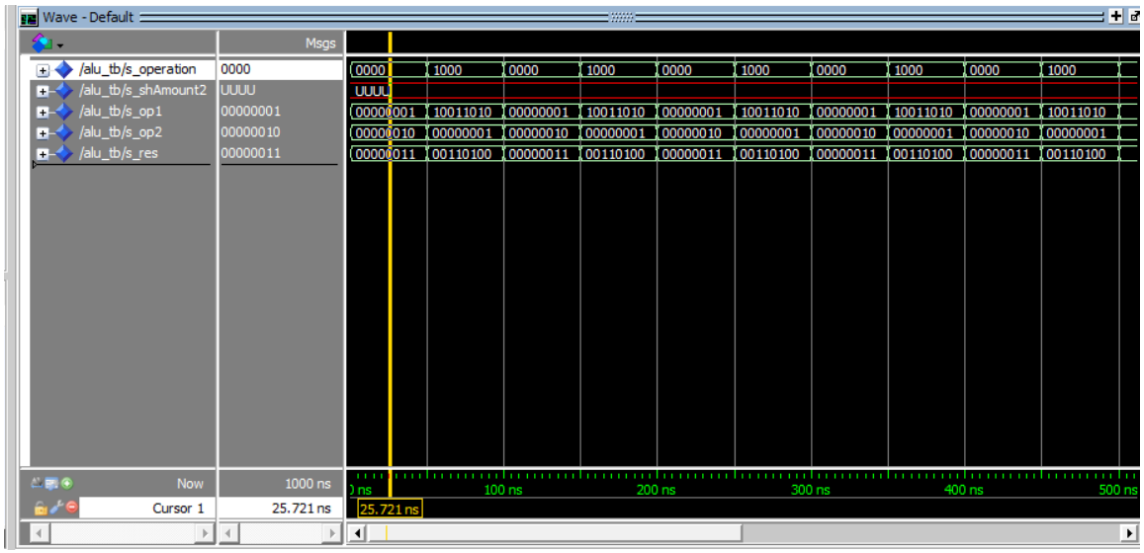


*Figura : TestBench Alu*

Quando a operation está a 0000 irá ser um add entre o op1(00000001) e o op2(00000010) que irá dar 00000011 mostrando o seu correto funcionamento para este tipo de operação.

Quando a operation é 1000 vai ser um deslocamento lógico do registo rs à esquerda registort bits. Se op1 é 10011010 e o op 2 00000001 (o shAmount2 irá ser de 0001 bits) e por isso o resultado é 00110100.

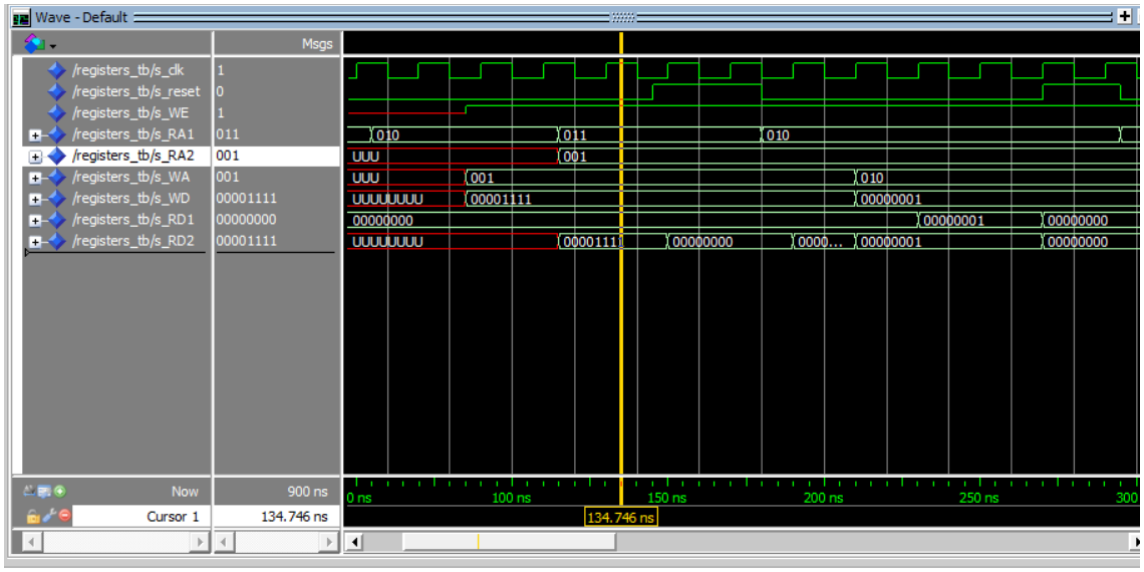


*Figura: Test Bench Registers*

Para fazer a simulação dos Registos tivemos de compilar os blocos pela sua ordem hierárquica (do mais “baixo” para o mais alto”).

Inseriram-se valores No RA1 e no RA2 e quando se fornece valores ao WA e ao WD (que permitem a escrita de informação) e com o WE ativado(caso contrário não era permitido a escrita) no adress”001” é escrito 00001111.

Quando o RA2 é invocado o valor instanciado(WD) vai estar impresso no RD2.

WA => escreve se o Address pretendido

WD => escreve se a informação que se passa para o address

## Fase 2

Nesta figura encontra-se o diagrama de estados da controlUnity.

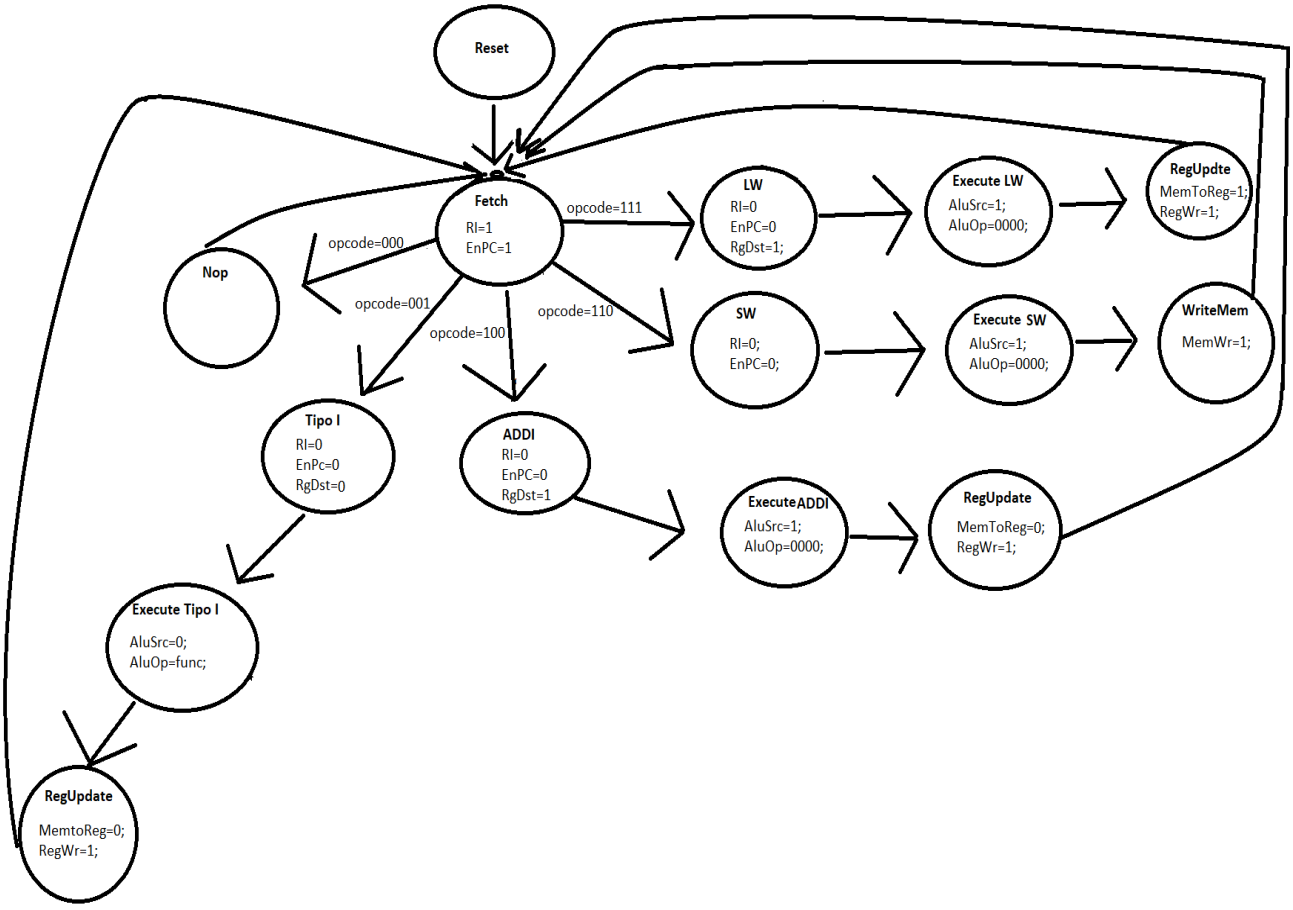


*Figura 2: Diagrama de estados*

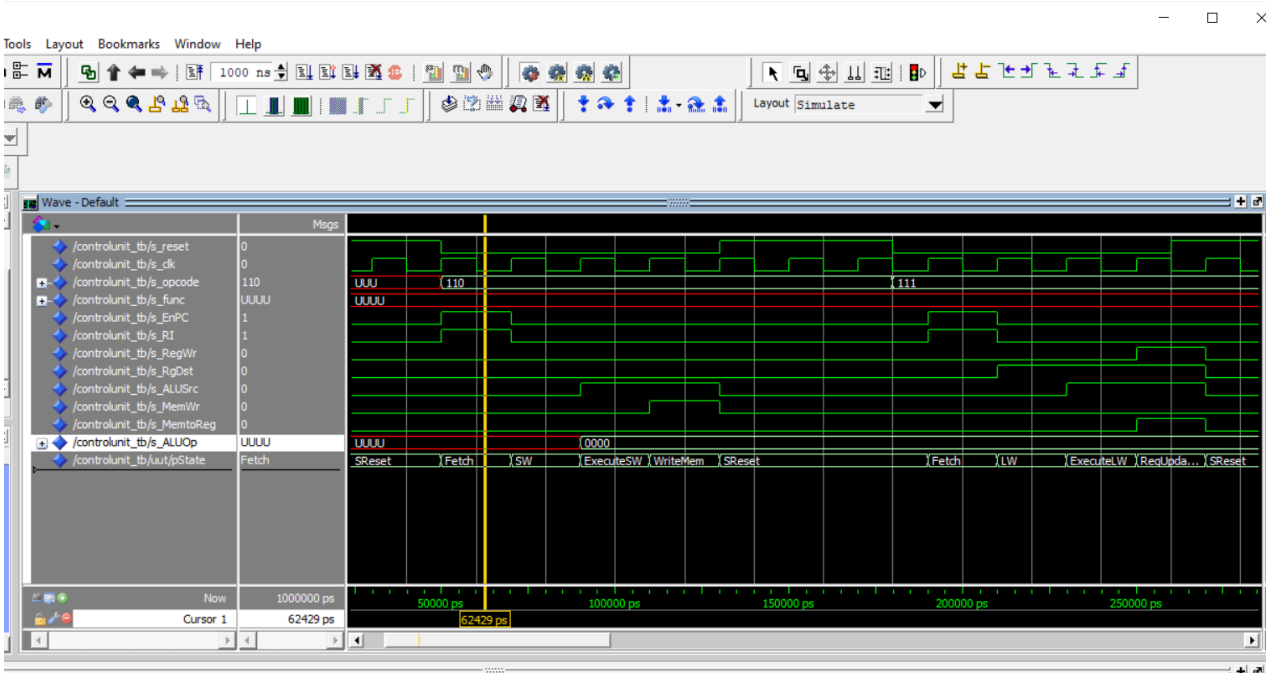


*Figura 3: Simulação da ControlUnity*

Fiz esta simulação com os opcodes 110 e 111 para verificar o funcionamento da controlunity.

Quando o opcode é 110 o primeiro estado para que vai após o reset é o Fetch onde RI=1 e EnPC=1. O estado SW vai desligar os estados RI e EnPC (pois a instrução vai ser/está a ser realizada e não estamos a precisar do progam counter para contar a próxima instrução).De seguida vamos realizar os cálculos de endereço da memória de dados (executeSW) o AluSrc=1 e a AluOp vai ser 0000 pois SW= DMemory[registors **+** address] = registort. A seguir vai haver um update para guardar o resultado num registo. Neste caso MemWr=1. A justificação deste fluxo encontra-se na imagem a seguir(figura4) ou no próprio diagrama de estados.

De seguida há um reset em que todos os estados vão para 0.

O raciocínio é o mesmo para o 111.

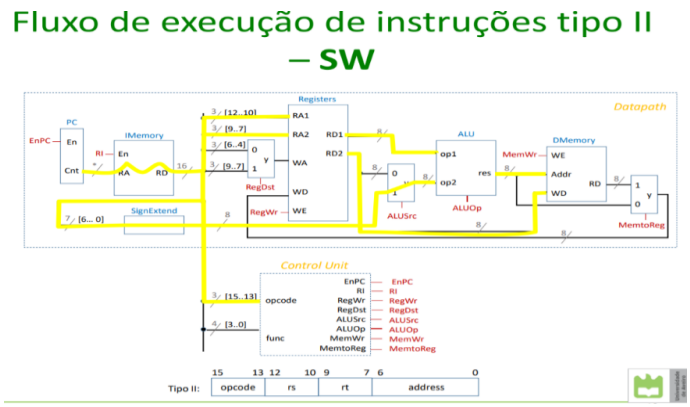


*Figura 4: Fluxo SW*

# Fase 3

O código da fase 3 implementa um código que interliga a unidade de execução e a unidade de controlo, ou seja, um código que liga a DataPath à ControlUnit.

No DataPath o EnPc ,RI ,RegDst , RegWr , ALUSrc, ALUOp, MemWr, MemtoReg são portos de entrada enquanto que na ControlUnit representam saídas. Ao mesmo tempo o Opcode e o func são saídas no Datapath e entradas na ControlUnit. Para isso criou-se sinais para interligar os blocos.

Não consegui completar a simulação mas o cógido VHDL vai resolvido com correta análise e síntese.

# Observações

Ao longo do projeto fui realizando simulações com a *university program vwf* para verificar o correto funcionamento de certos blocos.

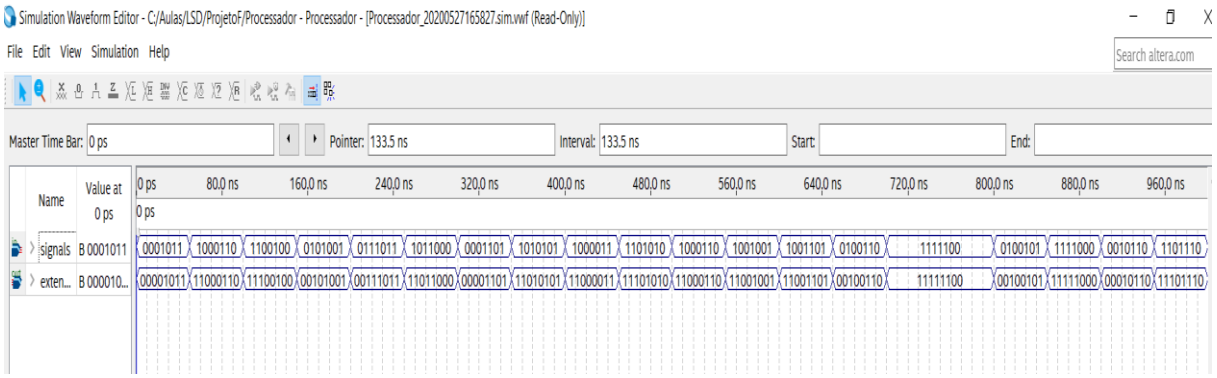


*Figura: simulação signextend*

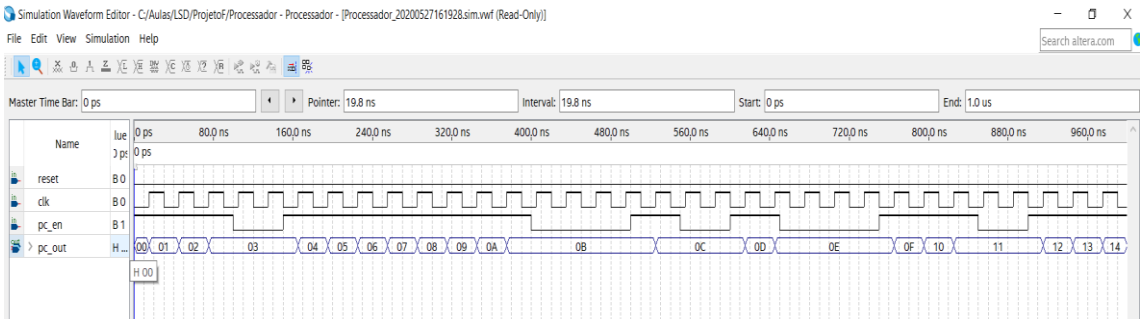


*Figura : simulação program counter*

A pasta ProjetoF contém os ficheiros/blocos da fase1.

A pasta fase2 contém o projeto da fase 2(inclui o controlUnity e a sua testBench)

A pasta fase 3 contém o projeto da fase 3 (código que interliga a unidade de execução e a unidade de controlo).

# Conclusão

Autoavalio-me em 10.

Apesar de não ter conseguido realizar todas as fases houve um esforço para completar o que se encontra feito do trabalho.