NEANDER

Ronaldo Drecksler Farias Pachico [O1P2]
Marlon Fabichacki Pereira [O1P2]

Módulo ULA:

Módulo responsável pelas operações lógicas e aritméticas, composto pela

- ULA
  Módulo Ops

  Entradas: x, y $_{8bits}$
  Saídas: madd, mnot, mand, mor $_{8bits}$
  Comportamento: recebe os valores e realiza as operações NOT, AND, OR e ADD.

f_adder8bits construído em aulas práticas anteriores.

```
u_add: f_adder8bits port map(x,y, '0', scout, madd);

   mnot <= not(x);

   mand <= x and y;

   mor <= x or y;
```

A divergência em relação à documentação do neander é a de que aqui foram usados apenas dois módulos: um para as operações vetoriais existentes no VHDL – AND, NOT e OR – e outro que foi instanciado neste primeiro para a ADD.

Detector NZ

Entrada: saída da ULA $_{8bits}$
Saída: NZ $_{2bits}$
Comportamento: recebe o resultado do cálculo da ULA e ativa (‘1’) o bit mais significativo caso seja um número negativo ou o bit menos significativo caso seja zero.

```
nz(1) <= data_in(7); -- Bit mais significativo em complemento de 2, significa negativo
   nz(0) <= not(data_in(7) or data_in(6) or data_in(5) or data_in(4) or data_in(3) or
data_in(2) or data_in(1) or data_in(0)); -- Not de uma or de todos os bits, retorna 1,
se e somente se, todos forem 0
```

- 1 Registrador(AC) de 8 bits
  Registrador com carga, idêntico ao construído durante aulas práticas anteriores.
- 1 Registrador(FLAGS) 2 bits
  Registrado com carga, onde o reset mantém o bit mais significativo em ‘0’ e o menos significativo em ‘1’.

- 1 $MUX_{2x8}z$
  Mux especial, usado para tratar o problema do inout do barramento.

```
barramento<= s_ac2ula when mem_nrw='1' else (others =>'Z');
```

Módulo Memória:

Módulo responsável pelo armazenamento e leitura de dados e instruções na memória, composto por:

- MUX_{2x8}
  Seleciona entre o recebimento de dados do PC ou do barramento.
- Registrador REM
- Memória
  ## Diferente da memória presente na documentação, essa memória possui uma entrada de clock.
- Registrador RDM
- 2 Mux_{2x8z}
  Assim como no Módulo ULA, resolvem o problema do inout

```
interface_barramento <= s_rdm2barr when MEM_nrw = '0' else (others => 'Z');
s_mem2rdm <= interface_barramento when MEM_nrw = '1' else (others => 'Z');
```

Módulo UC:
Módulo responsável por calcular e emitir os sinais no barramento de controle no neander, composto por:

- PC
  Construído usando um MUX_{2x8}, um registrador com carga e um f_adder_{8bits}, é responsável por incrementar a instrução atual e mandar a próxima instrução para a memória, ou então receber um pulo vindo do barramento e dar sequência a partir dele.
- Registrador RI
- Decodificador
  Recebe a instrução armazenada pelo RI com 8 bits e liga seu respectivo sinal em um barramento de 11 bits representando cada uma das operações do NEANDER, funcionando como um gerador de produto canônico.
- Control Unit
  Responsável por emitir os sinais de controle para os demais módulos do NEANDER, com base na instrução atual.
  Tabela verdade das instruções do NEANDER:
  **todas as entidades das operações se encontram dentro do arquivo “instrucoes.vhdl”.

NOP									
Contador	barr/inc	barr/pc	ULA_op	PC_nrw	AC_nrw	MEM_nrw	REM_nrw	RDM_nrw	RI_nrw
000	1	1	000	0	0	0	1	0	0
001	1	1	000	1	0	0	0	1	0
010	1	1	000	0	0	0	0	0	1
011	1	1	000	0	0	0	0	0	0
100	1	1	000	0	0	0	0	0	0
101	1	1	000	0	0	0	0	0	0
110	1	1	000	0	0	0	0	0	0
111	1	1	000	0	0	0	0	0	0

LDA	**AND**	**OR**	**ADD**						
Contador	barr/inc	barr/pc	ULA_op	PC_nrw	AC_nrw	MEM_nrw	REM_nrw	RDM_nrw	RI_nrw
000	1	1	xxx	0	0	0	1	0	0
001	1	1	xxx	1	0	0	0	1	0
010	1	1	xxx	0	0	0	0	0	1
011	1	1	xxx	0	0	0	1	0	0
100	1	1	xxx	1	0	0	0	1	0
101	1	0	xxx	0	0	0	1	0	0
110	1	1	xxx	0	0	0	0	1	0
111	1	1	xxx	0	1	0	0	0	0

NOT									
Contador	barr/inc	barr/pc	ULA_op	PC_nrw	AC_nrw	MEM_nrw	REM_nrw	RDM_nrw	RI_nrw
000	1	1	100	0	0	0	1	0	0
001	1	1	100	1	0	0	0	1	0
010	1	1	100	0	0	0	0	0	1
011	1	1	100	0	0	0	0	0	0
100	1	1	100	0	0	0	0	0	0
101	1	1	100	0	0	0	0	0	0
110	1	1	100	0	0	0	0	0	0
111	1	1	100	0	1	0	0	0	0

STA									
Contador	barr/inc	barr/pc	ULA_op	PC_nrw	AC_nrw	MEM_nrw	REM_nrw	RDM_nrw	RI_nrw
000	1	1	000	0	0	0	1	0	0
001	1	1	000	1	0	0	0	1	0
010	1	1	000	0	0	0	0	0	1
011	1	1	000	0	0	0	1	0	0
100	1	1	000	1	0	0	0	1	0
101	1	0	000	0	0	0	1	0	0
110	1	1	000	0	0	1	0	0	0
111	1	1	000	0	0	0	0	0	0

JMP									
Contador	barr/inc	barr/pc	ULA_op	PC_nrw	AC_nrw	MEM_nrw	REM_nrw	RDM_nrw	RI_nrw
000	1	1	000	0	0	0	1	0	0
001	1	1	000	1	0	0	0	1	0
010	1	1	000	0	0	0	0	0	1
011	1	1	000	0	0	0	1	0	0
100	1	1	000	0	0	0	0	1	0
101	0	0	000	1	0	0	0	0	0
110	1	1	000	0	0	0	0	0	0
111	1	1	000	0	0	0	0	0	0

JMPcond FALSO									
Contador	barr/inc	barr/pc	ULA_op	PC_nrw	AC_nrw	MEM_nrw	REM_nrw	RDM_nrw	RI_nrw
000	1	1	000	0	0	0	1	0	0
001	1	1	000	1	0	0	0	1	0
010	1	1	000	0	0	0	0	0	1
011	1	1	000	1	0	0	0	0	0
100	1	1	000	0	0	0	0	0	0
101	1	1	000	0	0	0	0	0	0
110	1	1	000	0	0	0	0	0	0
111	1	1	000	0	0	0	0	0	0

HLT									
Contador	barr/inc	barr/pc	ULA_op	PC_nrw	AC_nrw	MEM_nrw	REM_nrw	RDM_nrw	RI_nrw
000	0	0	000	0	0	0	0	0	0
001	0	0	000	0	0	0	0	0	0
010	0	0	000	0	0	0	0	0	0
011	0	0	000	0	0	0	0	0	0
100	0	0	000	0	0	0	0	0	0
101	0	0	000	0	0	0	0	0	0
110	0	0	000	0	0	0	0	0	0
111	0	0	000	0	0	0	0	0	0

Na tabela do LDA, AND, OR e ADD, os valores de ULA_op assumem o respectivo valor das instruções, ou seja, 000, 001, 010 e 011.

Para o funcionamento do JN e JZ, foi criado um módulo chamado JMPCondicional que, além das entradas padrão de todos os outros módulos, também recebe um seletor (sendo o bit mais significativo de NZ no caso do JN e o menos significativo no caso do JZ) que, por meio de cinco MUX_{2x1} é selecionada a operação de JMP quando o seletor é igual a ‘1’, ou então apenas o incremento do PC quando o seletor é igual a ‘0’.

PC_NRW	$\overline{C_1}$		C_1	
$\overline{C_2}$	0	1	0	0
C_2	1	0	0	0
STA LDA...	$\overline{C_0}$	C_0		$\overline{C_0}$

RDM_NRW	$\overline{C_1}$		C_1	
$\overline{C_2}$	0	1	0	0
C_2	1	0	0	1
LDA...	$\overline{C_0}$	C_0		$\overline{C_0}$

REM_NRW	$\overline{C_1}$		C_1	
$\overline{C_2}$	1	0	1	0
C_2	0	1	0	0
STA LDA...	$\overline{C_0}$	C_0		$\overline{C_0}$

PC_NRW	$\overline{C_1}$		C_1	
$\overline{C_2}$	0	1	0	0
C_2	0	1	0	0
JMP	$\overline{C_0}$	C_0		$\overline{C_0}$

RDM_NRW	$\overline{C_1}$		C_1	
$\overline{C_2}$	0	1	0	0
C_2	1	0	0	0
STA JMP	$\overline{C_0}$	C_0		$\overline{C_0}$

REM_NRW	$\overline{C_1}$		C_1	
$\overline{C_2}$	1	0	1	0
C_2	0	0	0	0
JMP	$\overline{C_0}$	C_0		$\overline{C_0}$

PC_NRW	$\overline{C_1}$		C_1	
$\overline{C_2}$	0	1	1	0
C_2	0	0	0	0
JMP Cond FALSO	$\overline{C_0}$	C_0		$\overline{C_0}$