CURSO: ENGENHARIA ELETROTÉCNICA E DE COMPUTADORES

DISCIPLINA: ARQUITETURA DE COMPUTADORES

TRABALHO DE LABORATÓRIO I

UNIDADE DE PROCESSAMENTO

Trabalho Realizado por: Xavier Abreu Dias Nº 87136

Diogo Martins Alves Nº 86980

Data: 24/03/2017

ANÁLISE E CARACTERIZAÇÃO DE UMA UNIDADE DE PROCESSAMENTO (1ª SEMANA)

1. [4 Val.] Caracterize completamente a unidade funcional (functional unit, FU), identificando:

a) As operações que a unidade é capaz de realizar através da análise dos esquemas elaborados pelo Vivado e dos ficheiros VHDL fornecidos. Para cada operação identificada indique o estado do sinal FS e a função na saída numa tabela. Inclua no relatório uma justificação sucinta de como chegou a estas conclusões. No caso da unidade aritmética faça uma tabela de verdade da lógica de entrada B de cada somador completo (full-adder) e da entrada de transporte (cary in) do somador.

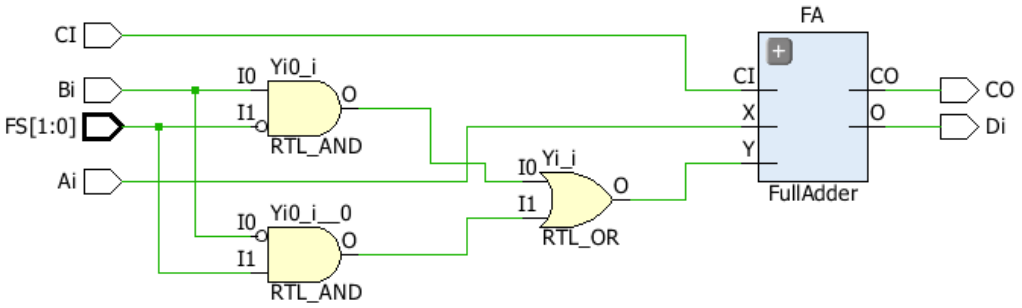
FS(3:0)	Operação	Função à saída D	Flags válidas
0000	Adição	$A + B$	Z, N, C, V
0001	Adição com carry	$A + B + 1$	Z, N, C, V
0010	Subtração menos uma unidade	$A + \overline{B} = A - B - 1$	Z, N, C, V
0011	Subtração	$A + \overline{B} + 1 = A - B$	Z, N, C, V
0100	AND	$A . B$	Z, N
0101	NAND	$\overline{A . B}$	Z, N
0110	OR	$A + B$	Z, N
0111	NOR	$\overline{A + B}$	Z, N
1000	XOR	$A \oplus B$	Z, N
1001	XNOR	$\overline{A \oplus B}$	Z, N
1010	Deslocamento lógico p/ esquerda	SHL(B)	Z, N, C
1011	Deslocamento lógico p/ direita	SHR(B)	Z, N, C
1100	Deslocamento aritmético p/ esquerda	SHLA(B)	Z, N, C, V
1101	Deslocamento aritmético p/ direita	SHRA(B)	Z, N, C, V
1110	Rotação para a esquerda	ROL(B)	Z, N, C
1111	Rotação para a direita	ROR(B)	Z, N, C

Justificações:

A saída D pode tomar o valor de 3 sinais diferentes, consoante o valor de FS(3:1), como mostra o seguinte código presente no ficheiro FunctionalUnit.vhd, onde DA corresponde à saída da Arithmetic Unit, DL à saída da Logic Unit e DS à saída do Shifter:

```
with FS(3 downto 1) select
DI <=   DA when "000" | "001",
        DL when "010" | "011" | "100",
        DS when "101" | "110" | "111",
        (others => 'X') when others;
```

Na Arithmetic Unit encontram-se 16 circuitos como o seguinte, compostos por um full adder e alguma lógica adicional.



Podemos observar que o sinal A é ligado diretamente ao operando X e o operando Y é obtido a partir dos sinais B e FS a partir da seguinte expressão:

$$Y = \left(Bi \,.\, \overline{FS(1)}\right) + \left(\overline{Bi} \,.\, FS(1)\right) = Bi \oplus FS(1)$$

Bi	FS(1)	Y
0	0	0 (= Bi)
0	1	1 (=$\overline{Bi}$)
1	0	1 (= Bi)
1	1	0 (=$\overline{Bi}$)

É também importante notar que a entrada carry in do somador correspondente ao bit de menor peso está ligado ao sinal FS(0):

```
C(0) <= FS(0);
```

A partir destas informações, é bastante fácil concluir qual será a saída da unidade aritmética para os diferentes valores de FS, como mostrado na primeira tabela.

Quando o sinal FS(3:1) escolhe um dos sinais DL ou DS para a saída D, basta verificarmos nos ficheiros dos respetivos componentes (Logic Unit e Shifter, respetivamente), quais são as operações executadas:

```
with FS select
Di <= Ai and Bi     when "100",
      Ai nand Bi    when "101",
      Ai or Bi      when "110",
      Ai nor Bi     when "111",
      Ai xor Bi     when "000",
      Ai xnor Bi    when "001",
      '-'           when "010" | "011",
      'X'           when others;
```

```
with FS select
D <= (others => '-')          when "000" | "001",
     B(14 downto 0) & '0'     when "010",
     '0' & B(15 downto 1)     when "011",
     B(14 downto 0) & '0'     when "100",
     B(15) & B(15 downto 1)   when "101",
     B(14 downto 0) & B(15)   when "110",
     B(0) & B(15 downto 1)    when "111",
     (others => 'X')          when others;
```

b) Quais as flags válidas para cada uma das operações identificadas. Justifique sucintamente.

As flags Zero (Z) e Negative (N) são válidas para qualquer uma das operações implementadas. Por esta razão elas não estão definidas em nenhum dos sub-blocos da functional unit, mas sim na própria functional unit.

Nas operações relativas à Arithmetic Unit, uma vez que se tratam de operações aritméticas, faz sentido haver as flags overflow e carry:

```
-- cary
CO <= C(16);
-- overflow
Yn <= (B(15) and not FS(1)) or (not B(15) and FS(1));
OV <= (A(15) and Yn and not Z(15)) or (not A(15) and not Yn and Z(15));
```

Nas operações da Logic Unit não faz sentido existirem estas duas flags uma vez que operações lógicas estão relacionadas com os vários bits de um número e não com o número que esses bits representam, por essa razão estas duas flags não estão definidas na Logic Unit.

Por fim, quanto à operações do Shifter faz sentido existir carry em todas elas, que corresponde ao bit deslocado, mas apenas faz sentido existir overflow nos deslocamentos aritméticos, embora esta flag esteja definida para todas as operações do Shifter:

```
CO <= B(15) when FS(0)='0' else
      B(0) when FS(0)='1' else
      'X';

OV <=  B(15) xor B(14) when FS(0)='0' else
       '0' when FS(0)='1' else
       'X';
```

2. [2 Val.] Identifique e caracterize a Unidade de Armazenamento através da análise dos esquemas dos ficheiros VHDL. Inclua no relatório uma tabela com a descrição e função de cada um dos sinais da unidade de armazenamento.

A unidade de armazenamento corresponde ao ficheiro RF1 – RegisterFile.h. O seu objetivo é disponibilizar dois operandos (A(15:0) e B(15:0)) à unidade funcional. Possui 16 registos de 16 bits cada um. Os valores destes registos podem ser lidos ou escritos consoante os seguintes sinais de input:

DA(3:0)	Seleciona um dos 16 registos onde será armazenado o sinal Data(15:0)
Data(15:0)	Variável que será armazenada num dos registos
WR	Controla a escrita no registo selecionado (esta apenas é ativa quando WR=1)

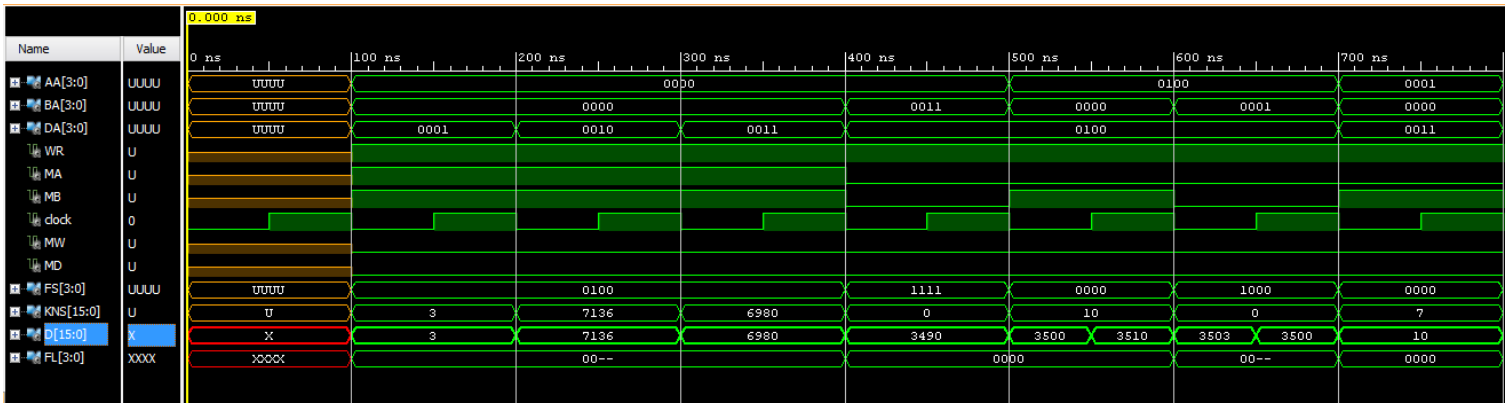
Os seguintes sinais servem para selecionar os registos de onde serão lidos os sinais de output A(15:0) e B(15:0):

AA(3:0)	Seleciona um dos 16 registos do qual será lida a saída A(15:0)
BA(3:0)	Seleciona um dos 16 registos do qual será lida a saída B(15:0)

3. [3 Val.] Faça um teste do funcionamento da unidade de processamento de forma a realizar as operações indicadas na tabela; para cada operação indique numa tabela a palavra de controlo respetiva, i.e., o valor de todos os sinais de controlo necessários à execução da operação. Faça uma única simulação com os diferentes micro operações em sequência, e com uma micro operação por ciclo de relógio. Apresente ainda na tabela os resultados esperados e compare com o resultado da simulação. Siga o exemplo apresentado na Tabela 2. Para realizar as simulações modifique o ficheiro SimulDataPath.vhd fornecido junto com o projeto.

Operação	Palavra de controlo										Valor esperado	
	AA	BA	DA	WR	MA	MB	$KNS_{(10)}$	FS	MD	MW	$D_{(10)}$	Flags(ZNCO)
R1←3	X	X	0001	1	1	1	3	0100	0	X	3	00--
R2←7136	X	X	0010	1	1	1	7136	0100	0	X	7136	00--
R3←6980	X	X	0011	1	1	1	6980	0100	0	X	6980	00--
R4←ROR(R3)	X	0011	0100	1	X	0	X	1111	0	X	3490	0000
R4←R4+10	0100	X	0100	1	0	1	10	0000	0	X	3500	0000
R4←R4 xor R1	0100	0100	1	0	0	0	X	1000	0	X	3503	00--
R3←R1+7	0001	X	0011	1	0	1	7	0000	0	X	10	0000

Nota: Na simulação do Vivado, os "don't cares" (X) foram substituídos por zeros.



Como se pode observar na simulação, os valores esperados coincidem com os resultados obtidos.

4. [1 Val.] Considere a realização da 5ª operação da tabela anterior. Verifique na simulação qual o valor do sinal D(31:0) à saída da FU antes e depois do flanco de relógio. Comente o que observa.

Durante a realização da 5ª operação (R4←R4+10) o sinal D muda de 3500 para 3510 depois de se dar o flanco ascendente de relógio. Isto acontece porque quando se dá o flanco de relógio, R4 toma o valor que o sinal D tem antes do relógio (3500) e a unidade aritmética volta a executar a operação R4+10, que neste caso tem o valor de 3510, que corresponde ao valor de D depois do flanco de relógio.

SÍNTESE DE NOVAS FUNCIONALIDADES PARA A UNIDADE DE PROCESSAMENTO (2ª SEMANA)

Alterar a unidade aritmética 1

Altere a unidade aritmética para reduzir o consumo para operações com números de valor absoluto reduzido colocando os bits de maior peso a zero, mesmo no caso em que os números são negativos. Adicione lógica que deteta que a parte alta do A e B (os 8 bits mais significativos) é igual ao bit de sinal, A(7) e B(7) ativando o sinal small_number. Se small_number estiver a um então coloque a parte alta (8 bits) de A e B a zero e faça extensão de sinal do resultado (8 bits).

Para alterarmos a Unidade Aritmética conforme pedido, resolvemos fazer o seguinte:

Criámos um componente Detect_Low.vhd, que deteta se ambos os operandos cumprem os requisitos de "small_number", com base na lógica fornecida no enunciado. Este componente foi então introduzido dentro da Arithmetic Unit:

```
--adiçao do componente detect_low
component Detect_low is
    port(A: in std_logic_vector(15 downto 0);
         B: in std_logic_vector(15 downto 0);
         small_number: out std_logic);
end component;
```

Adicionámos alguns sinais internos à Arithmetic unit, que correspondem ao sinal "small_number" e aos operandos A e B transformados no caso de small_number=1:

```
--adiçao de sinais internos à arithmetic unit
signal small_number : std_logic;
signal A_inter, B_inter, D_inter: std_logic_vector(15 downto 0);
```

Declarámos os sinais A_inter e B_inter de tal forma que seriam iguais a A e B, respetivamente, no caso de small_number=0 e no caso de small number=1 teriam os seus 8 bits mais significativos transformados em '0':

```
--logica que transforma os bits de maior peso em 0 caso small_number = 1
A_inter <= ( A(15 downto 8) and (7 downto 0 => not(small_number))) & A(7 downto 0);
B_inter <= ( B(15 downto 8) and (7 downto 0 => not(small_number))) & B(7 downto 0);
```

Alterámos as entradas dos somadores, para que estes recebecem os operandos A_inter e B_inter em vez de A e B e ligamos os sinais A, B e small_number ao componente Detect_Low:

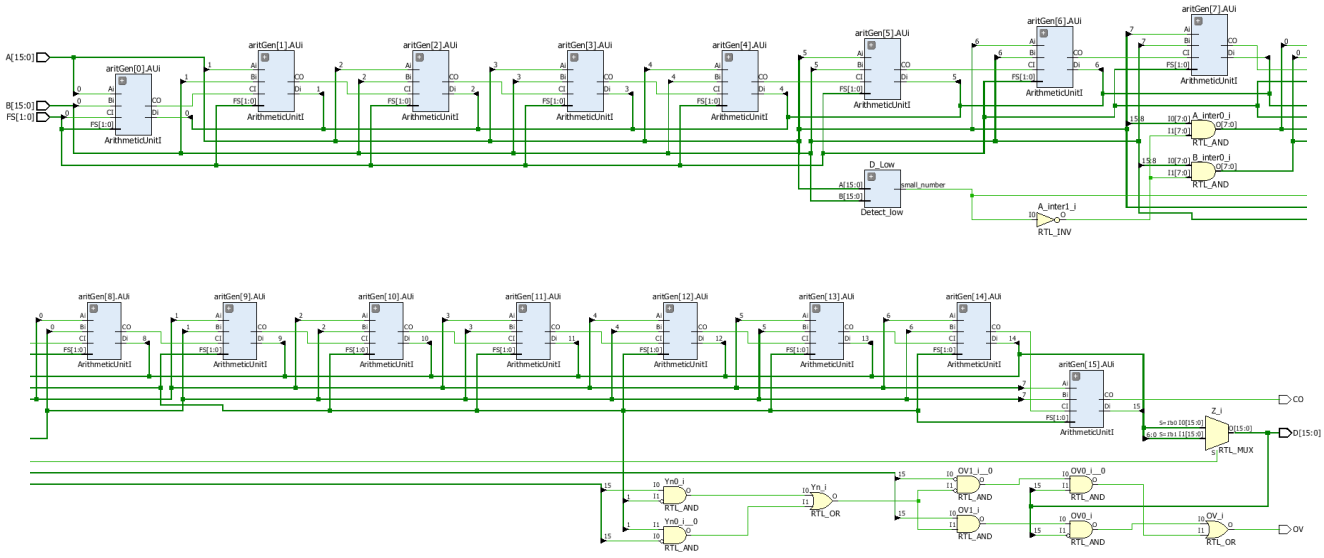
```
--alteração das entradas dos somadores
aritGen: for I in 15 downto 0 generate
    AUi: ArithmeticUnitI port map (Ai=>A_inter(I), Bi=>B_inter(I), FS=>FS, CI=>C(I), Di=>D_inter(I), CO=>C(I+1));
end generate aritGen;

--mapeamento das entradas e saídas do componente detect_low
D_Low: Detect_low port map( A=>A, B=>B, small_number=>small_number);
```

Construímos a lógica para estender o sinal do operando de saída dos somadores, caso os operandos de entrada sejam números pequenos em valor absoluto:

```
--sign extend
with small_number select
Z <= D_inter when '0',
     (7 downto 0 => D_inter(7)) & D_inter(7 downto 0) when '1',
     (others => 'X') when others;
```

Obtivemos então o seguinte esquema gerado pelo Vivado:

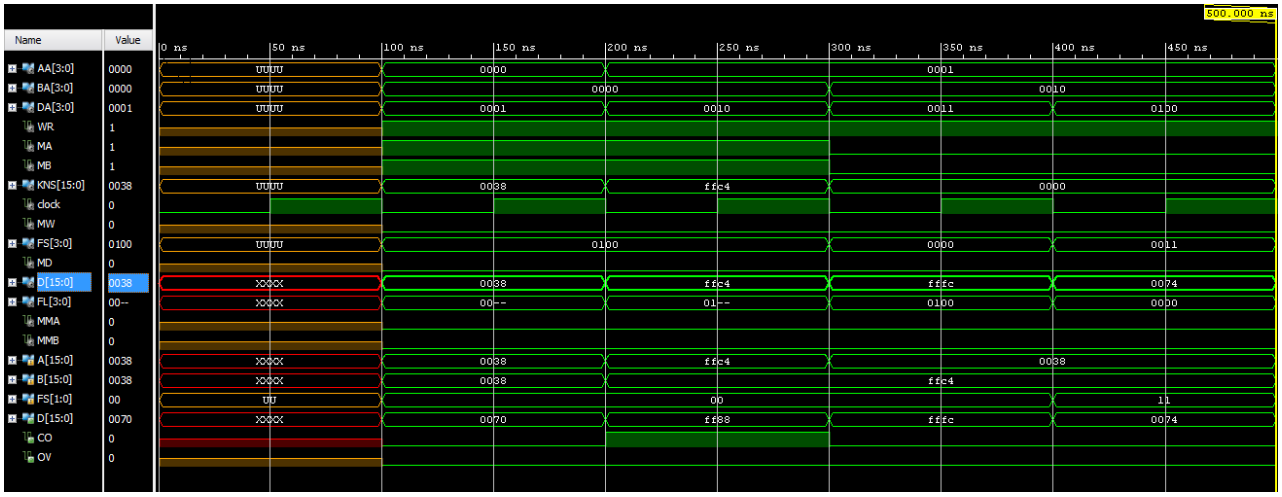


a) Teste as alterações ao data path, nomeadamente implementando as seguintes operações:
R1 ← 56 (0038h)
R2 ← -60 (FFC4h)
R3 ← R1 + R2
R4 ← R1 - R2
Acrescente na simulação os sinais internos com os valores das entradas e das saídas da unidade aritmética antiga

Para realizar a simulação alteramos o ficheiro SimulDataPath.vhd da seguinte forma:

```
wait until clock'event and clock='0';
-- R1 <- 38h
AA<="0000" ; BA<="0000"; DA<="0001"; WR<='1'; MA<='1'; MB<='1'; KNS<=x"0038"; MW<='0'; FS<="0100"; MD<='0'; MMB<='0'; MMA<='0';
wait until clock'event and clock='0';
-- R2 <- FFC4h
AA<="0001" ; BA<="0000"; DA<="0010"; WR<='1'; MA<='1'; MB<='1'; KNS<=x"FFC4"; MW<='0'; FS<="0100"; MD<='0'; MMB<='0'; MMA<='0';
wait until clock'event and clock='0';
-- R3 <- R1 + R2
AA<="0010" ; BA<="0001"; DA<="0011"; WR<='1'; MA<='0'; MB<='0'; KNS<=x"0000"; MW<='0'; FS<="0000"; MD<='0'; MMB<='0'; MMA<='0';
wait until clock'event and clock='0';
-- R4 <- R1 - R2
AA<="0001" ; BA<="0010"; DA<="0100"; WR<='1'; MA<='0'; MB<='0'; KNS<=x"0000"; MW<='0'; FS<="0011"; MD<='0'; MMB<='0'; MMA<='0';
```

A simulação obtida foi a seguinte:



R1 ← $56_{(10)}$=0038h
R2 ← $-60_{(10)}$ = FFC4h
R3 ← R1 + R2 = $56_{(10)}$ + $-60_{(10)}$ = $-4_{(10)}$ = FFFCh
R4 ← R1 - R2 = $56_{(10)}$ – ($-4_{(10)}$) = $52_{(10)}$ = 0074h

Concluímos então que a simulação devolveu os valores corretos.

Nota:

Ao implementarmos esta funcionalidade, pensámos em alterar as saídas carry e overflow de modo a que tomassem valores corretos face à nova situação. No entanto, chegámos à conclusão de que estes sinais iriam tomar os mesmos valores caso os alterássemos ou não.

Alterar entradas da memória

Altere o data path de forma se possam utilizar como entradas de endereço da memória os resultados da unidade funcional. Mais concretamente acrescente dois MUXs na entrada desta controlados pelas entradas MMA e MMB, tal como representado na figura.

Para fazermos as alterações pedidas, foi necessário alterar o ficheiro DataPath.vhd, nomeadamente:

Adicionar duas entradas MMA e MMB, ambas de 1 bit, para selecionar os sinais que irão entrar nas entradas Data e Address da Data Memory, respetivamente:

```
--adição de entradas da memoria
MMA : in std_logic;
MMB : in std_logic);
```

Adicionar dois sinais internos DataK e AddressK, que correspondem às saídas dos dois multiplexers adicionados:

```
--adição de sinais internos para as entradas de memória
signal DataK, AddressK: std_logic_vector (15 downto 0);
```

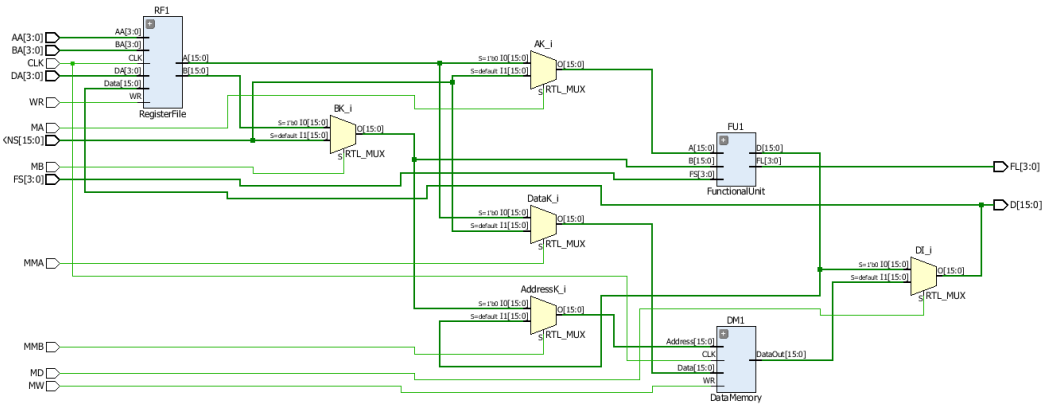
Alterar as ligações às entradas da Data Memory de modo a que o sinal AddressK se ligasse à entrada Address e DataK à entrada Data:

```
--alteração ligações à memoria
DM1 : DataMemory port map (Address=>AddressK, Data=>DataK, WR=>MW, CLK=>CLK, DataOut=>DM);
```

Construir os multiplexers cujas saídas se irão ligar às entradas da Data Memory:

```
--multiplexers que ligam às entradas de memória
AddressK <= BK when MMB='0' else
            DF when MMB='1' else (others => 'X');
DataK <= A when MMA='0' else
         KNS when MMA='1' else (others => 'X');
```

Feitas todas estas alterações, acabámos com o seguinte esquema:



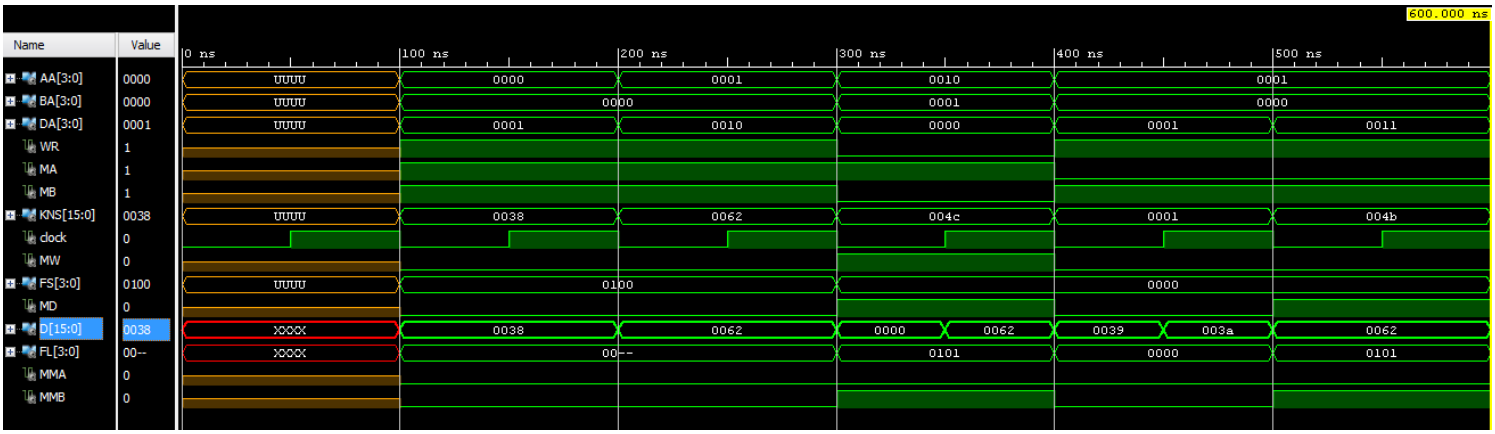
Através da análise deste esquema, concluímos que a implementação da nova funcionalidade foi feita com sucesso.

a) Teste as alterações ao data path, nomeadamente implementando as seguintes operações:
   R1 ← 38h
   R2 ← 62h
   M[4Ch + R1] ← R2
   R1 ← R1 + 1
   R3 ← M[4Bh + R1]

Para a simulação foi necessário alterar o ficheiro SimulDataPath.vhd da seguinte forma:

```
wait until clock'event and clock='0';
-- R1 <- 38h
AA<="0000" ; BA<="0000"; DA<="0001"; WR<='1'; MA<='1'; MB<='1'; KNS<=x"0038"; MW<='0'; FS<="0100"; MD<='0'; MMB<='0'; MMA<='0';
wait until clock'event and clock='0';
-- R2 <- 62h
AA<="0001" ; BA<="0000"; DA<="0010"; WR<='1'; MA<='1'; MB<='1'; KNS<=x"0062"; MW<='0'; FS<="0100"; MD<='0'; MMB<='0'; MMA<='0';
wait until clock'event and clock='0';
-- M[4Ch + R1] <- R2
AA<="0010" ; BA<="0001"; DA<="0000"; WR<='0'; MA<='1'; MB<='0'; KNS<=x"004C"; MW<='1'; FS<="0000"; MD<='1'; MMB<='1'; MMA<='0';
wait until clock'event and clock='0';
-- R1 <- R1 + 1
AA<="0001" ; BA<="0000"; DA<="0001"; WR<='1'; MA<='0'; MB<='1'; KNS<=x"0001"; MW<='0'; FS<="0000"; MD<='0'; MMB<='0'; MMA<='0';
wait until clock'event and clock='0';
-- R3 <- M[4Bh + R1]
AA<="0001" ; BA<="0000"; DA<="0011"; WR<='1'; MA<='0'; MB<='1'; KNS<=x"004B"; MW<='0'; FS<="0000"; MD<='1'; MMB<='1'; MMA<='0';
```

O resultado da simulação é o seguinte:



R1 ← 38h
R2 ← 62h
M[4Ch + R1 = 4Ch + 38h = 84h] ← R2 = 62h
R1 ← R1 + 1 = 38h + 1h = 39h
R3 ← M[4Bh + R1 = 4Bh + 39h = 84h] = 62h

Na terceira operação, o valor 0000h de D corresponde ao valor que estava armazenado no endereço 84h. Depois do flanco ascendente de relógio, este valor é alterado para 0062h, que corresponde ao valor que foi escrito nessa posição de memória.

Na quarta operação, a existência de dois valores distintos para o sinal D é explicada da mesma maneira que a questão 4 da 1º semana.

Conclusão:

Podemos afirmar que o trabalho foi concluído com sucesso, uma vez que conseguimos implementar as operações pedidas e as simulações efetuadas no vivado deram os resultados que estávamos à espera.