Lab 9 – Implementação do MIPS em VHDL e teste na FPGA

Neste trabalho de laboratório vamos trabalhar com um MIPS descrito em VHDL e implementado numa FPGA.

1. Descrição da implementação do MIPS em VHDL

O arquivo FPGA_MIPS.zip fornecido para estes trabalho contém um projecto para o Quatus II[1] com a implementação *single clock cycle* do MIPS em VHDL[2]. Neste trabalho vamos utilizar a implementação dada (fig. 1), não alterando a sua estrutura.

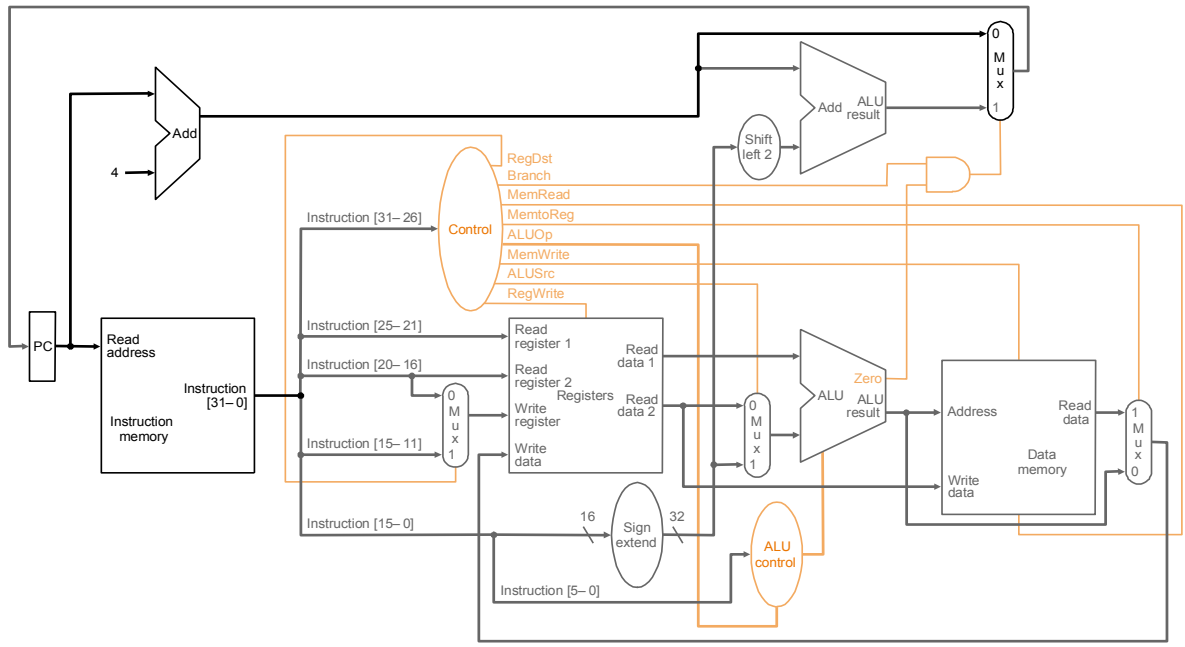

Fig. 1. Implementação *single clock cycle* do MIPS.

[1] Altera Quartus II disponível em: http://www.deec.uc.pt/~jlobo/altera/
[2] Implementação adaptada do livro "Rapid Prototyping of Digital Systems" de Jim Hamblen, baseada modelo descrito no livro "Computer Organization & Design" de Patterson and Hennessy.

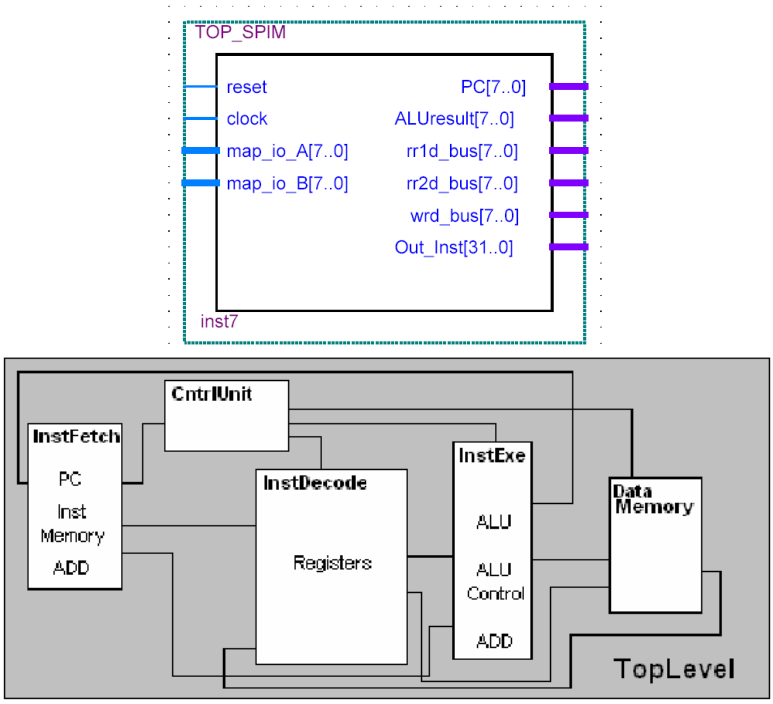

Fig. 2. Módulo de topo TOP_SPIM.VHD
e modelo hierárquico da implementação em VHDL.

A implementação do MIPS em VHDL recorre a uma hierarquia de dois níveis (fig. 2), tendo ao topo um VHDL estrutural, TOP_SPIM.VHD, que interliga os 5 modelos comportamentais que ficam no segundo nível, nomeadamente:

- IFETCH.VHD (Instruction fetch stage): memória de intruções e o PC;
- CONTROL.VHD (Control unit): lógica do controlador do datapath;
- IDECODE.VHD (Decode stage): contém o ficheiro de registos dual-port;
- EXECUTE.VHD (Execute stage): ALU de dados e de endereço de salto;
- DMEMORY.VHD (Data memory stage): memória de dados e I/O mapeado.

Nesta implementação o formato completo de instruções de 32 bits é utilizado, mas o *datapath* é limitado a 8 bits para permitir a simulação e síntese rápida, e o teste nas placas disponíveis no laboratório. Temos apenas os registos $R0..$R7 limitados a 8 bits, a memória de instruções tem apenas 16 elementos, e a memória de dados 4 posições, sendo as duas últimas de I/O mapeado. Após um *reset* cada registo fica com o seu número de índice, e as duas primeiras posições de memória de dados são inicializadas com 0x55, 0xAA, sendo as outras duas posições só de leitura, correspondendo a interruptores da placa.

Para este trabalho apenas será necessário alterar o módulo **IFETCH.VHD** para testar diferentes programas, e o esquemático de topo onde se liga o processador com os diversos módulos para trabalhar com as placas disponíveis no laboratório.

O modelo segue o formato das instruções de 32 bits para MIPS (fig. 3), embora depois o *datapath* seja limitado como referido acima.

Field Size	6-bits	5-bits	5-bits	5-bits	5-bits	6-bits
R - Format	Opcode	Rs	Rt	Rd	Shift	Function
I - Format	Opcode	Rs	Rt	Address/immediate value		
J - Format	Opcode	Branch target address				

Fig. 3. Formato das instruções MIPS de 32 bits

A implementação dada suporta apenas as instruções de Add, Sub, And, Or e Slt de formato R, e Lw, Sw e Beq de formato I, como indicado na figura 4.

Mnemonic	Format	Opcode Field	Function Field	Instruction
Add	R	0	32	Add
Sub	R	0	34	Subtract
And	R	0	36	Bitwise And
Or	R	0	37	Bitwise Or
Slt	R	0	42	Set if Less Than
Lw	I	35	-	Load Word
Sw	I	43	-	Store Word
Beq	I	4	-	Branch on Equal

Fig. 4. Instruções implementadas no modelo VHDL do MIPS.

Para escrever um programa temos que ter em conta o formato das instruções e traduzir para código máquina para colocar no módulo **IFETCH.VHD**, como no seguinte exemplo que corresponde à instrução **add $1,$2,$3** [3]:

```
constant mem2 : std_logic_vector(31 downto 0) := (
 To_Stdlogicvector(B"000000 00010 00011 00001 00000 100000"));--add $1,$2,$3
```

O módulo recorre a constantes directas em VHDL e não blocos de memória para ocupar menos espaço na FPGA, facilitando a simulação, e libertando espaço para outros elementos que serão implementados na placa de testes DE2 da altera, como a saída VGA, ou memo múltiplos MIPS na mesma FPGA.

[3] O valor constante binário indicado em VHDL não pode ter espaços. Neste enunciado foram colocados espaços apenas para facilitar a leitura.

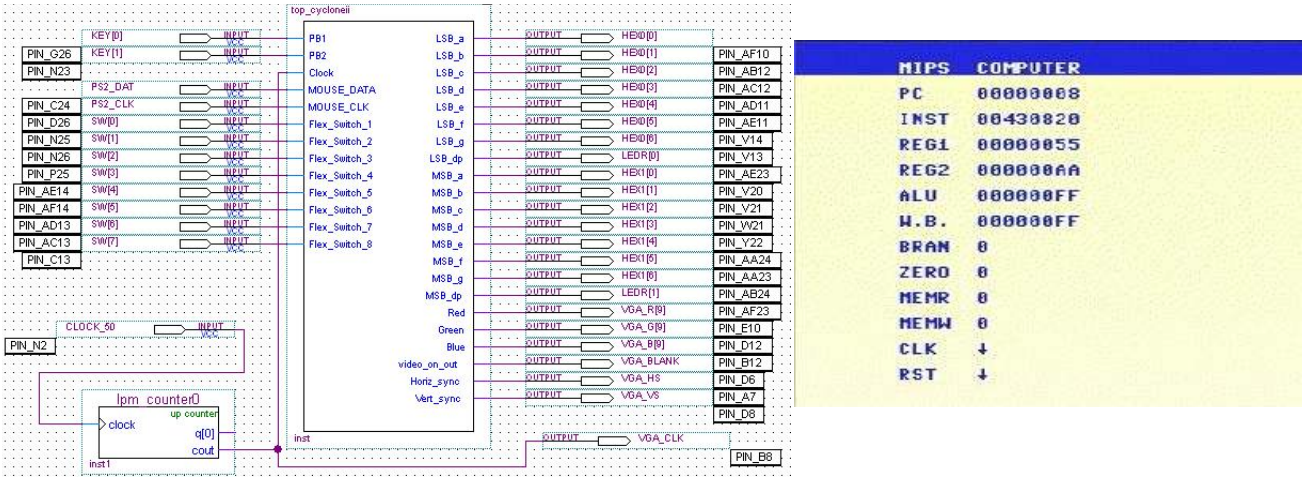

Fig. 5. Módulo TOP_CYCLONEII.VHD e saída VGA correspondente.

O modulo de topo, **TOP_SPIM**, e a sua variante com VGA, **TOP_ CYCLONEII**, colocam nas saídas sinais adicionais para ver o funcionamento do processador. A saída VGA pode assim mostrar os valores hexadecimais dos principais *BUSes* do processador, nomeadamente:

PC: (32bits) *program counter* com endereço da próxima instrução

INST: (32bits) instrução

REG1: (32 bits) porto1 do ficheiro de registos

REG2: (32 bits) porto2 do ficheiro de registos

ALU: (32 bits) saída da ALU

W.B.: (32 bits) porto de escrita do ficheiro de registos

BRAN: (1bit) controlo de salto

ZERO: (1bit) zero da ALU

MEMR: (1bit) ler de memória

MEMW: (1bit) escrever em memória

CLK: (1bit) *clock*

RST: (1bit) *reset*

Para controlar o funcionamento do processador, temos as teclas de *reset* (KEY[1]) e o *clock* (KEY[0]). Pulsando o *clock* uma vez vamos executar uma nova instrução. O *reset* vai por o PC a zero, coloca 0x55 e 0xAA nas duas primeiras localizações da memória, e cada registo com o valor do seu índice.

2. Testar implementação na FPGA

O projecto incluído no arquivo **FPGA_MIPS.zip** tem como entidade de topo um esquemático **MIPS_DE2.bdf** com o módulo **TOP_ CYCLONEII.VHD** ligado aos pinos da DE2. Deve verificar se o *device* corresponde à placa que estiver a utilizar, e recorrer ao ficheiro **DE2_pin_assignments.csv** para importar as respectivas atribuição de pinos. O projecto tem outros esquemáticos que irá utilizar mais à frente neste trabalho, que utilizam uma biblioteca de módulos UP-Core da Altera para trabalhar com algumas das funcionalidades da placa DE2 da Altera, como VGA, teclado e rato. Estes módulos estão disponíveis na página da cadeira no woc, **de2.zip**. Os respectivos ficheiros devem ficar nas directoria **C:\altera\##\quartus\libraries\de2** [4], devendo no projecto indicar a localização da biblioteca para a placa pretendida em **Project > Add/Remove Files in Project > User Libraries (Current Project)**.

Depois de ajustar o projecto para a placa que estiver a utilizar, recompile e programe a placa. Se analisar o conteúdo de IFETCH.VHD encontra o seguinte código VHDL:

```
-- Insert SPIM Machine Language Test Program Here
          constant mem0 : std_logic_vector(31 downto 0) := (
--          Field    | op | rs | rt | rd ?addr/immed|
To_Stdlogicvector(B"100011 00000 00010 0000000000000000"));  -- lw $2,0($0)
          constant mem1 : std_logic_vector(31 downto 0) := (
To_Stdlogicvector(B"100011 00000 00011 0000000000000001"));  -- lw $3,1($0)
          constant mem2 : std_logic_vector(31 downto 0) := (
To_Stdlogicvector(B"000000 00010 00011 00001 00000 100000"));-- add $1,$2,$3
          constant mem3 : std_logic_vector(31 downto 0) := (
To_Stdlogicvector(B"101011 00000 00001 0000000000000011"));  -- sw $1,3($0)
          constant mem4 : std_logic_vector(31 downto 0) := (
To_Stdlogicvector(B"000100 00001 00010 1111111111111111"));  -- beq $1,$2,-1
          constant mem5 : std_logic_vector(31 downto 0) := (
To_Stdlogicvector(B"000100 00001 00001 1111111111111010"));  -- beq $1,$1,-6
          constant mem6 : std_logic_vector(31 downto 0) := (
To_Stdlogicvector(B"00000000000000000000000000000000"));     -- nop
          constant mem7 : std_logic_vector(31 downto 0) := (
To_Stdlogicvector(B"00000000000000000000000000000000"));     -- nop
```

que corresponde ao seguinte programa:

```
[00] lw   $2, 0($0)     # rs=00000 = $0
[04] lw   $3, 1($0)
[08] add  $1, $2, $3    # 100000 = 32 = function field do Add
[12] sw   $1, 3($0)
[16] beq  $1, $2, -1    # 1111111111111111=-1  para ter  -4 = -1*4 (ver fig.1)
[20] beq  $1, $1, -6    # 1111111111111010=-6  para ter -24 = -6*4 (ver fig.1)
[24] nop
```

[4] Se não tiver permissões de escrita/leitura pode colocar os ficheiros de2.zip directamente na pasta do seu projecto, não sendo necessário configurar a biblioteca.

Para testar deve primeiro fazer reset, i.e. carregar em reset (KEY[1]) e pulsar o clock (KEY[0]) uma vez, no ecrã deve ver as setas do sinal de clock e reset a variar. O reset vai por o PC a zero, coloca 0x55 e 0xAA nas duas primeiras localizações da memória, e cada registo com o valor do seu índice. Siga passo a passo a execução do programa, tendo em conta o diagrama da figura 1 na interpretação dos diversos sinais visualizados no ecrã.

Para a realização das alíneas seguintes, pode utilizar o mesmo projecto e ir alterando os ficheiros, se criar um novo ficheiro do desenho esquemático do seu circuito, não esquecer de o indicar como o principal do seu projecto (top-level entity), seleccionando essa opção com o botão do lado direito do rato sobre o ficheiro na janela do Project Navigator.

Para a atribuição de pinos da FPGA às entradas e saídas do circuito, utilize nomes de acordo coma lista standard da placa DE2, de forma a recorrer ao ficheiro **DE2_pin_assignments.csv** para importar a atribuição de pinos.

3. Passar de assembly para código máquina

Converta o seguintes programas para código máquina, edite o ficheiro IFETCH.VHD (guardando a versão anterior com outro nome), recompile e teste na FPGA o seu funcionamento, recordando que ao fazer reset fica com 0x55 e 0xAA nas duas primeiras localizações da memória, e cada registo com o valor do seu índice.

1)

```
init:
    lw    $2, 1($0)
    sub   $3, $7, $3
    and   $4, $4, $5
    or    $5, $6, $7
    add   $6, $2, $1
    beq   $1, $1, init
```

2)

```
init:
    sub   $2, $1, $3
    and   $4, $2, $5
    or    $5, $2, $6
    add   $3, $4, $2
    slt   $1, $7, $6
    beq   $1, $0, init
```

4. Codificar e testar um ciclo *for*

Implemente o seguinte ciclo *for* com o modelo do MIPS em VHDL dado, tendo em conta as instruções disponíveis.

```
for($6=10;$6!=0;$6--){
      $7=$6;
}
```

Para conseguir o valor inicial 10 pode ter que somar dois registos, tendo em conta que após o *reset* eles ficam com o valor do índice.

Compile e teste na placa, vendo na saída VGA o funcionamento passo a passo do programa.

DE2_pin_assignments.csv

```
# Altera's DE2 board
# Cyclone II
# EP2C35F672C6
#
To,Location
# push-buttons
KEY[0],PIN_G26
KEY[1],PIN_N23
KEY[2],PIN_P23
KEY[3],PIN_W26
# Switches
SW[0],PIN_N25
SW[1],PIN_N26
SW[2],PIN_P25
SW[3],PIN_AE14
SW[4],PIN_AF14
SW[5],PIN_AD13
SW[6],PIN_AC13
SW[7],PIN_C13
SW[8],PIN_B13
SW[9],PIN_A13
SW[10],PIN_N1
SW[11],PIN_P1
SW[12],PIN_P2
SW[13],PIN_T7
SW[14],PIN_U3
SW[15],PIN_U4
SW[16],PIN_V1
SW[17],PIN_V2
# Red LEDs
LEDR[0],PIN_AE23
LEDR[1],PIN_AF23
LEDR[2],PIN_AB21
LEDR[3],PIN_AC22
LEDR[4],PIN_AD22
LEDR[5],PIN_AD23
LEDR[6],PIN_AD21
LEDR[7],PIN_AC21
LEDR[8],PIN_AA14
LEDR[9],PIN_Y13
LEDR[10],PIN_AA13
LEDR[11],PIN_AC14
LEDR[12],PIN_AD15
LEDR[13],PIN_AE15
LEDR[14],PIN_AF13
LEDR[15],PIN_AE13
LEDR[16],PIN_AE12
LEDR[17],PIN_AD12
# Green LEDs
LEDG[0],PIN_AE22
LEDG[1],PIN_AF22
LEDG[2],PIN_W19
LEDG[3],PIN_V18
LEDG[4],PIN_U18
LEDG[5],PIN_U17
LEDG[6],PIN_AA20
LEDG[7],PIN_Y18
LEDG[8],PIN_Y12
#
# 7 segments
# hex0
HEX0[0],PIN_AF10
HEX0[1],PIN_AB12
HEX0[2],PIN_AC12
HEX0[3],PIN_AD11
HEX0[4],PIN_AE11
HEX0[5],PIN_V14
HEX0[6],PIN_V13
# hex1
HEX1[0],PIN_V20
HEX1[1],PIN_V21
HEX1[2],PIN_W21
HEX1[3],PIN_Y22
HEX1[4],PIN_AA24
HEX1[5],PIN_AA23
HEX1[6],PIN_AB24
# hex2
HEX2[0],PIN_AB23
HEX2[1],PIN_V22
HEX2[2],PIN_AC25
HEX2[3],PIN_AC26
HEX2[4],PIN_AB26
HEX2[5],PIN_AB25
HEX2[6],PIN_Y24
# hex3
HEX3[0],PIN_Y23
HEX3[1],PIN_AA25
HEX3[2],PIN_AA26
HEX3[3],PIN_Y26
HEX3[4],PIN_Y25
HEX3[5],PIN_U22
HEX3[6],PIN_W24
# hex4
HEX4[0],PIN_U9
HEX4[1],PIN_U1
HEX4[2],PIN_U2
HEX4[3],PIN_T4
HEX4[4],PIN_R7
HEX4[5],PIN_R6
HEX4[6],PIN_T3
# hex5
HEX5[0],PIN_T2
HEX5[1],PIN_P6
HEX5[2],PIN_P7
HEX5[3],PIN_T9
HEX5[4],PIN_R5
HEX5[5],PIN_R4
HEX5[6],PIN_R3
# hex6
HEX6[0],PIN_R2
HEX6[1],PIN_P4
HEX6[2],PIN_P3
HEX6[3],PIN_M2
HEX6[4],PIN_M3
HEX6[5],PIN_M5
HEX6[6],PIN_M4
# hex7
HEX7[0],PIN_L3
HEX7[1],PIN_L2
HEX7[2],PIN_L9
HEX7[3],PIN_L6
HEX7[4],PIN_L7
HEX7[5],PIN_P9
HEX7[6],PIN_N9
# 50MHz clock
CLOCK_50,PIN_N2
# PS2
PS2_CLK,PIN_D26
PS2_DAT,PIN_C24
# VGA
VGA_R[9],PIN_E10
VGA_G[9],PIN_D12
VGA_B[9],PIN_B12
VGA_HS,PIN_A7
VGA_VS,PIN_D8
VGA_CLK, PIN_B8
VGA_BLANK, PIN_D6
```