AULA PRÁTICA N.º 13

Objetivos

- Implementar em VHDL e testar a unidade de controlo principal do *datapath multi-cycle*.
- Adicionar a unidade de controlo à estrutura do *datapath* desenvolvida na aula anterior.
- Realizar testes de funcionamento do *datapath multi-cycle*, executando pequenos programas de teste.

Não faça *copy/paste* do código que foi disponibilizado nas aulas teóricas. Escrever o código VHDL ajuda-o a entender a estrutura e o funcionamento do *datapath*.

Introdução

Nesta sequência de aulas práticas, está a ser implementado o *datapath multi-cycle* que foi apresentado nas aulas teóricas. Nesta aula, faremos a implementação da unidade de controlo e a respetiva integração no *datapath* montado nas aulas anteriores.

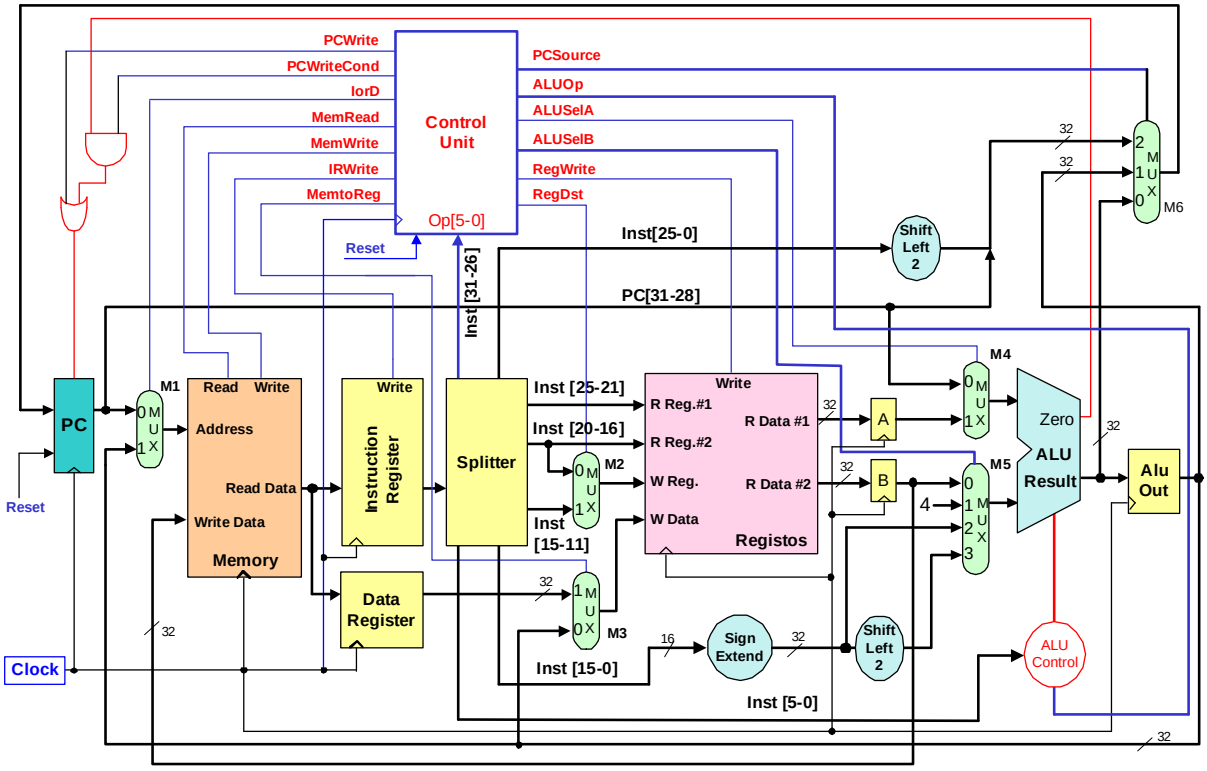

Figura 1. *Datapath multi-cycle*.

Unidade de controlo

No *datapath multi-cycle*, cada instrução é decomposta num conjunto de ciclos de execução, correspondendo cada um destes a um período de relógio distinto. Na versão simplificada do *datapath* considerado nesta implementação, o número de ciclos de relógio necessário para executar cada uma das instruções é o seguinte: 3 ciclos de relógio para as instruções de *branch* e *jump*; 4 ciclos de relógio para as instruções tipo R, imediatas e de escrita na memória; 5 ciclos de relógio para a leitura da memória.

A geração dos sinais de controlo ao longo do conjunto de ciclos em que é decomposta cada instrução depende da instrução particular que está a ser executada. A decomposição da execução

da instrução num conjunto de fases torna necessário o recurso a uma máquina de estados síncrona para a geração de todos os sinais de controlo.

A Figura 2 apresenta o diagrama de estados completo, modelo de Moore, da unidade de controlo do *datapath multi-cycle*. De salientar que, nessa figura, os sinais de saída não explicitados em cada estado ou são irrelevantes (por exemplo *multiplexers*) ou encontram-se no estado não ativo, isto é, tomam o valor lógico '**0**' (controlo de elementos de estado).

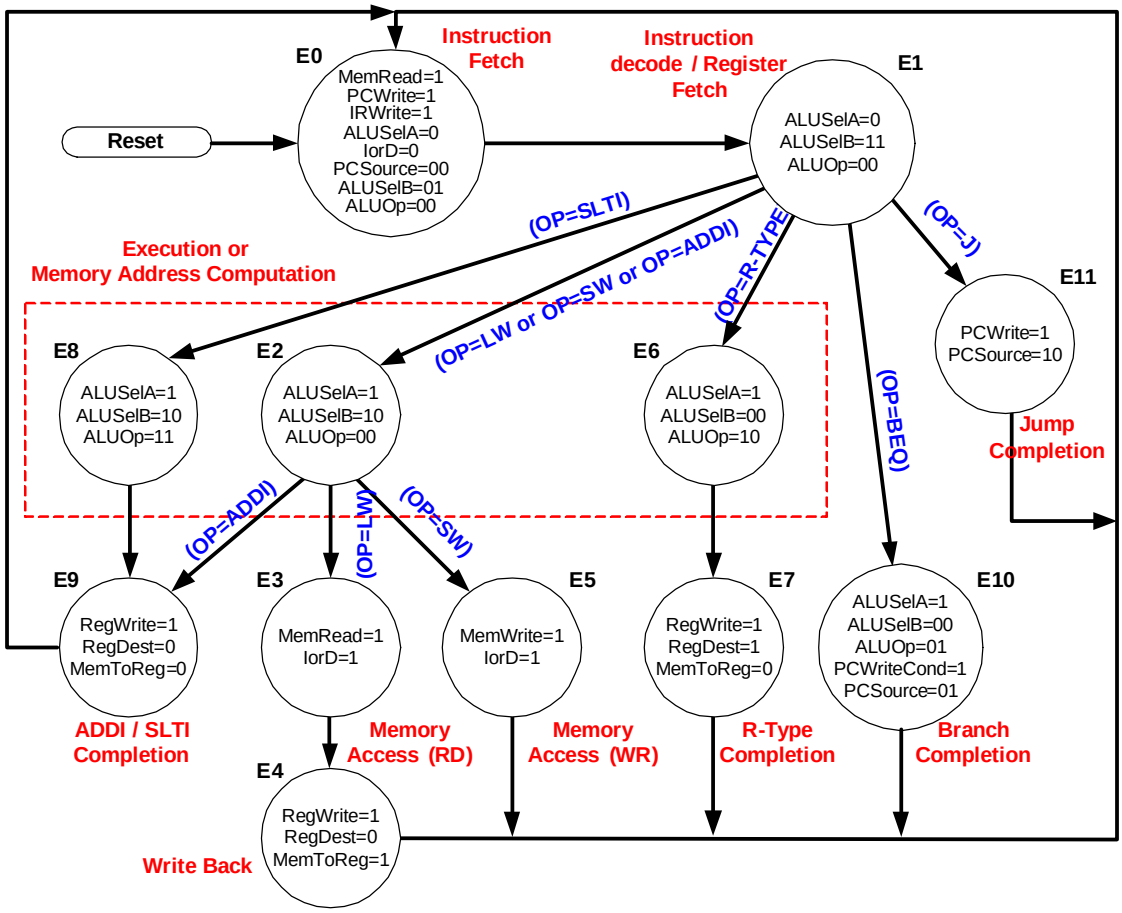

Figura 2. Diagrama de estados da unidade de controlo do *datapath multi-cycle*.

Guião

Parte I

1. Preencha a tabela abaixo com o nome de cada uma das fases de execução e com o valor que tomam, em cada uma delas, os sinais de controlo aí indicados, para todas as instruções que o *datapath multi-cycle* suporta, isto é, **`r-type`**, **`addi`**, **`slti`**, **`lw`**, **`sw`**, **`beq`** e **`j`** (uma tabela para cada instrução).

Tabela 1. Sinais de controlo e fases de execução das instruções.

	Fase 1	**Fase 2**	**Fase 3**	**Fase 4**	**Fase 5**
Nome da fase					
PCWriteCond					
PCWrite					
MemWrite					
MemRead					
IRWrite					
RegWrite					
RegDst					
ALUOp					
ALUSelA					
ALUSelB					
MemtoReg					
PCSource					
IorD					

2. Preencha a tabela seguinte com o nome de cada uma das fases de execução da instrução **`sw $7,0x20($6)`** e com o valor que tomam, em cada uma delas, os sinais e os valores do *datapath* nos elementos funcionais/operativos ali indicados. Considere que os registos, no instante em que vai iniciar-se o *instruction fetch*, têm os seguintes valores: **`$7=0x12F4C6A8`**, **`$6=0x00000004`** e **`$PC=0x00000008`**.

Tabela 2. Fases de execução e valores do *datapath*.

	Fase 1	**Fase 2**	**Fase 3**	**Fase 4**	**Fase 5**
Nome da fase					
PC					
Instr. Register					
Data Register					
A					
B					
ALU Result					
ALU Out					
ALU Zero					

3. Abra no Quartus o projeto do *datapath multi-cycle* que tem vindo a construir.

4. Implemente em VHDL a unidade de controlo do *datapath*, tomando em consideração o resultado do preenchimento das tabelas dos sinais de controlo que efetuou no exercício 1 e os códigos apresentados na Tabela 3. Poderá designar este módulo por "**ControlUnit.vhd**".

5. Acrescente à unidade de controlo o código que permita codificar, com 5 bits em BCD, o estado atual da máquina de estados. O resultado deve ser ligado ao sinal global "**DU_CState**", permitindo, assim, observar a evolução da máquina de estados no decurso da execução de uma instrução. O código seguinte fornece uma possível solução, em que "**CS**" é o estado corrente da máquina de estados:

```
DU_CState <= "00000" when CS = E0 else
             "00001" when CS = E1 else
             (...)
             "10001" when CS = E11 else
             "11111" when others;
```

   No módulo de visualização pode observar o estado em que a máquina de estados da unidade de controlo se encontra, nos displays **HEX4** e **HEX5**, selecionando a entrada PC (isto é, **InputSel="00"**) e o modo de visualização parcial (**DispMode='1'**).

6. Selecione este ficheiro como o *top-level* do projeto.

Tabela 3. Códigos do sub-conjunto de instruções suportadas pela arquitetura.

Instrução	Formato	opcode (6 bits)
r-type	oper rd, rs, rt	0x00
lw	lw rt, imm(rs)	0x23
sw	sw rt, imm(rs)	0x2B
addi	addi rt, rs, imm	0x08
slti	slti rt, rs, imm	0x0A
beq	beq rs, rt, imm	0x04
j	j target	0x02

7. Simule funcionalmente a unidade de controlo para todas as instruções que o *datapath* suporta, isto é, verifique se os sinais de controlo gerados são corretos para todos os códigos da Tabela 3, em cada uma das fases que a execução da respetiva instrução se decompõe.

Parte II

1. Abra o ficheiro "**mips_multi_cycle.vhd**" e selecione-o como o *top-level* do projeto.

2. Ligue os sinais gerados pela Unidade de Controlo a LEDs da placa (sugestão: sinais da esquerda da unidade de controlo a LEDs verdes **LEDG[6..0]**, restantes a LEDs vermelhos **LEDR[8..0]**).

3. Efetue a síntese e a implementação do projeto.

4. Com o *datapath multi-cycle* concluído podem agora ser feitos testes de funcionamento, executando pequenos programas que permitam verificar o correto funcionamento do processador. Para tal, podemos usar os programas que foram já usados para teste da versão *single-cycle*. Assim, para cada um dos exemplos que se seguem, deve inicializar a memória RAM com o código máquina das instruções, fazer a síntese e implementação do projeto e reprogramar a FPGA.

   Note que o espaço de armazenamento em RAM disponível terá agora que ser partilhado por instruções e dados. A divisão poderá ser a seguinte: instruções entre o endereço **0x00** e o endereço **0x7F**; dados entre os endereços **0x80** e **0xFF**.

Programa de teste 1:

Address	Code	Instr	Comment
0x00000000		main: addi $3,$0,0x55	
0x00000004		sw $3,0x80($0)	
0x00000008		lw $4,0x80($0)	
0x0000000C		nop	

Programa de teste 2:

Address	Code	Instr	Comment
		main: addi $3,$0,0x8F	
		addi $4,$0,0x36	
		add $4,$3,$4	
		sw $4,0x84($0)	
		lw $5,-65($4)	
		nop	

Programa de teste 3:

```
void main(void)
{
    int i, soma;
    int *p = 0x80;

    for(i=4, soma=0; i > 0; i--)
    {
        *p = 2 * i;
        soma = soma + 2 * i;
        p++;
    }
    *p = soma;
}
```

Address	Code	Instr	Comment
		main: addi $5,$0,4	i = 4;
		addi $6,$0,0x80	p = 0x80;
		add $7,$0,$0	soma = 0;
		for: slt $1,$0,$5	while(i > 0)
		beq $1,$0,endf	{
		addi $8,$5,$5	
		sw $8,0($6)	*p = 2 * i;
		add $7,$7,$8	soma += 2 * i;
		addi $6,$6,4	p++;
		addi $5,$5,-1	i--;
		j for	}
		endf: sw $7,0($6)	*p = soma;
		w1: beq $0,$0,w1	while(1);
		nop	

Após a execução do programa verifique o conteúdo da memória nos endereços **0x80**, **0x84**, **0x88**, **0x8C** e **0x90**.

5. O sinal de relógio do *datapath* que usou até agora é gerado manualmente premindo a tecla **KEY[0]** da placa de desenvolvimento (o módulo de *debouncing* gera um pulso, com a duração de 1 ciclo do seu relógio de referência, sempre que é premida a tecla **KEY[0]**). Desse modo foi possível observar a execução de cada instrução, ciclo a ciclo.

Pretende-se agora usar como relógio do *datapath* um sinal com uma frequência de 8 Hz, obtido por divisão de frequência do relógio de referência da placa de desenvolvimento (**CLOCK_50**, 50 MHz). Para isso instancie um divisor de frequência (pode obter o código no *moodle* da UC) e ligue o sinal de saída ao sinal que usava para ligar o relógio aos diversos pontos do *datapath* (desligue a saída do *debouncer*: "**pulsedOut => open**"). Deve ter em atenção que o programa em teste deverá terminar com um ciclo infinito, tal como é feito no "programa de teste 3" apresentado no ponto anterior.

Programa de teste 4:

O objetivo deste programa é ordenar um *array* de 6 *words*, armazenado no segmento de dados da memória RAM a partir do endereço **0x80**. Para executar e testar este programa terá que adicionar à inicialização da memória as 6 *words* com os valores iniciais do *array*, a partir da posição **0x20** (a que corresponde o endereço **0x80**). As posições não ocupadas entre o final do programa e o início dos dados terão que ser preenchidas (por exemplo com **X"00000000"**), de modo a garantir que os dados ficam armazenados nos endereços pretendidos.

Sugerem-se os seguintes valores iniciais para o *array*: **20**, **17**, **-2**, **25**, **5**, **-1**. O trecho de código VHDL seguinte (incompleto) mostra a inicialização da memória com estes valores.

```
subtype TDataWord is std_logic_vector(31 downto 0);
type TMemory is array(0 to 31) of TDataWord;
signal s_memory : TMemory := (
            --          .text (address = 0x00)
                        X"20050000", --  addi $5,$0,0
                        (...)
                        X"00000000",
                        (...)
                        X"00000000",

            --          .data (address = 0x80)
            -- array:   .word 20, 17, -2, 25, 5, -1
                        X"00000014", --  .word  20
                        X"00000011", --  .word  17
                        X"FFFFFFFE", --  .word  -2
                        X"00000019", --  .word  25
                        X"00000005", --  .word  5
                        X"FFFFFFFF", --  .word  -1
                        others => X"00000000");
```

```
#define  SIZE   6
#define  TRUE   1
#define  FALSE  0

void main(void)
{
     static int array[]={20, 17, -2, 25, 5, -1};
     int flag, i, j, aux;

     j = SIZE – 1;
     do {
          flag = TRUE;
          for (i=0; i < j; i++) {
               if (array[i] > array[i+1])
               {
                  aux = array[i];
                  array[i] = array[i+1];
                  array[i+1] = aux;
                  flag = FALSE;
               }
          }
          j--;
     } while (flag == FALSE);
}
```

```
# Mapa de registos
  $2: flag
  $3: i
  $4: j
  $5: &array[0]
```

Address	Code	Instr	Comment
			.text
0x00		main: addi $5,$0,0x80	$5=&array[0]
0x04		addi $4,$0,5	j = SIZE – 1
		do:	do {
0x08		addi $2,$0,1	flag = 1
0x0C		addi $3,$0,0	i = 0
0x10		for: slt $1,$3,$4	while(i < j)
0x14		beq $1,$0,endfor	{
0x18		add $6,$3,$3	
0x1C		add $6,$6,$6	aux = i << 2
0x20		add $6,$6,$5	$6 = array + i
0x24		lw $7,0($6)	$7=array[i]
0x28		lw $8,4($6)	$8=array[i+1]
0x2C		slt $1,$8,$7	if($7 > $8)
0x30		beq $1,$0,endif	{
0x34		sw $7,4($6)	array[i+1]=$7
0x38		sw $8,0($6)	array[i]=$8
0x3C		addi $2,$0,0	flag = 0
		endif:	}
0x40		addi $3,$3,1	i++
0x44		j for	}
0x48		endfor:addi $4,$4,-1	j--
0x4C		beq $2,$0,do	} while(flag == 0)
0x50		nop	
0x54		w1: beq $0,$0,w1	while(1);
0x58		nop	
(...)		nop	
0x7C		nop	
			.data
0x80	0x00000014		.word 20
0x84	0x00000011		.word 17
0x88	0xFFFFFFFE		.word -2
0x8C	0x00000019		.word 25
0x90	0x00000005		.word 5
0x94	0xFFFFFFFF		.word -1
0x98	0x00000000		
(...)	0x00000000		
0xFC	0x00000000		

Observe o conteúdo da memória, no segmento de dados, e verifique se o *array* ficou corretamente ordenado.

PDF gerado em 27/10/2017