AULA PRÁTICA N.º 10

Objetivos

- Implementar em VHDL e testar os seguintes módulos do *datapath single-cycle*: memória de dados e unidade de controlo principal.
- Adicionar os novos módulos à estrutura do *datapath* desenvolvida nas aulas anteriores.
- Realizar testes de funcionamento do *datapath single-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 single-cycle* que foi apresentado nas aulas teóricas. Nesta aula, faremos a implementação dos dois últimos módulos: memória de dados e unidade de controlo principal. Para a implementação destes módulos tome como referência o código apresentado nas aulas teóricas.

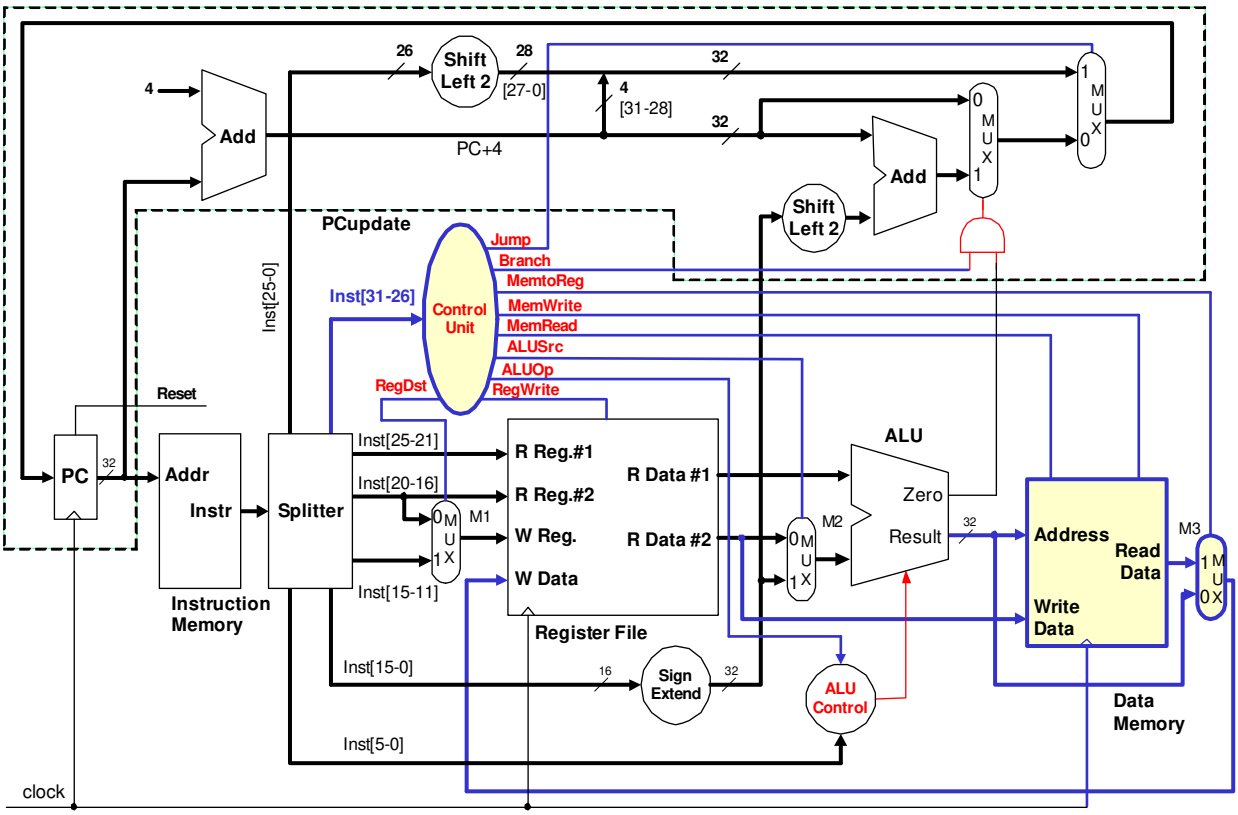

Figura 1. *Datapath single-cycle* com os módulos a implementar nesta aula desenhados com fundo colorido e as ligações a azul.

Memória de dados

A memória RAM armazena os dados do programa. Nesta implementação a capacidade da RAM será limitada a 64 posições de 32 bits cada (6 bits de endereço). A escrita é síncrona e a leitura é assíncrona, sendo estas operações dependentes dos sinais de controlo *write enable* e *read enable*, respetivamente. A dimensão do barramento de endereços desta memória está definida na *package*

"**MIPS_pkg.vhd**" na constante "**RAM_ADDR_SIZE**", que deverá ser usada para definir a respetiva contante genérica de parametrização.

O endereço de acesso à memória de dados é calculado na ALU e tem uma dimensão de 32 bits. Desses 32 bits apenas vão ser necessários 6. Do mesmo modo que já vimos para a memória de instruções, como cada posição de memória armazena o equivalente a 4 bytes (2^2), para formar o endereço de acesso à memória são ignorados os 2 bits menos significativos da ALU e usados os 6 seguintes (A_7 a A_2).

Multiplexer

O *multiplexer* M3 permite encaminhar para o porto de escrita do banco de registos o valor calculado pela ALU ou o valor lido da memória. Este *multiplexer* é controlado pelo sinal *MemtoReg* gerado pela unidade de controlo.

Unidade de controlo

A unidade de controlo principal do *datapath* é uma unidade combinatória que gera os sinais de controlo em função do campo *opcode* da instrução (bits 31 a 26). Esta unidade deverá reconhecer e gerar os sinais de controlo do seguinte sub-conjunto de instruções do MIPS: **ADD**, **SUB**, **AND**, **OR**, **NOR**, **XOR**, **SLT**, **ADDI**, **SLTI**, **LW**, **SW**, **BEQ** e **J**.

Guião

Parte I

1. Abra no Quartus o projeto do *datapath single-cycle* que tem vindo a construir.
2. Implemente em VHDL a memória RAM 64x32 com escrita síncrona e leitura assíncrona, com barramentos de dados independentes. A escrita e a leitura são dependentes dos sinais de controlo *write enable* e *read enable*, respetivamente. Poderá designar este módulo por "**DataMemory.vhd**". Selecione este ficheiro como o *top-level* do projeto.
3. Usando o simulador *University Program VWF*, simule funcionalmente o módulo "**DataMemory.vhd**", efetuando operações de escrita em pelo menos 5 posições de memória e operações de leitura dessas posições. Use na simulação os sinais de controlo de escrita e de leitura.
4. Implemente na memória RAM um ponto de leitura assíncrona que deve ser ligado ao módulo de visualização através dos sinais globais "**DU_DMaddr**" e "**DU_DMdata**".
5. Preencha cuidadosamente a Tabela 1, onde devem ser definidos todos os sinais de controlo para as instruções consideradas.

Tabela 1. Sinais de controlo para o sub-conjunto de instruções implementado.

Instr	RegWrite	MemRead	MemWrite	RegDst	ALUSrc	MemtoReg	ALUOp	Branch	Jump
r-type									
lw									
sw									
addi									
slti									
beq									
j									

6. Implemente em VHDL a unidade de controlo do *datapath*, tomando em consideração o resultado do preenchimento da Tabela 1 e os códigos apresentados na Tabela 2. Poderá

designar este módulo por "**ControlUnit.vhd**". Selecione este ficheiro como o *top-level* do projeto.

Tabela 2. 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 2.

Parte II

1. Abra o ficheiro "**MIPS_SingleCycle.vhd**" e selecione-o como o *top-level* do projeto. Instancie os módulos que desenvolveu nesta aula e faça as respetivas ligações ao restante diagrama. Não se esqueça de eliminar a ligação da saída da ALU ao banco de registos feita na aula anterior.

2. Ligue também os sinais gerados pela Unidade de Controlo a LEDs da placa (**LEDR[0]** a **LEDR[9]**).

3. Efetue a síntese e a implementação do projeto.

4. Com o *datapath* concluído podem agora ser feitos testes de funcionamento, executando pequenos programas que permitam verificar o correto funcionamento do processador. Para tal, deve, para cada um dos exemplos que se seguem, obter o código máquina de cada uma das instruções e inicializar a memória de instruções com esses códigos. Naturalmente que, para essa operação ter efeito, tem que efetuar a síntese e implementação do projeto e reprogramar a FPGA.

Nota: Pode fazer a codificação das instruções de forma manual ou usando o *assembler* do MARS. No caso da instrução *jump* (j), uma vez que a secção de código usada no MARS (início em **0x00400000**) é diferente da usada neste *datapath* (início em **0x00000000**), terá que fazer a codificação de forma manual. Recorde que a instrução "**j**" utiliza endereçamento direto.

Programa de teste 1:

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

Programa de teste 2:

Address	Code	Instr	Comment
		main: addi $3,$0,0x1F	
		addi $4,$0,0x24	
		add $4,$3,$4	
		sw $4,4($0)	
		lw $5,-63($4)	
		nop	

Programa de teste 3:

```
void main(void)
{
    int i, soma;
    static int array[4];  // Considera-se que o "Data segment"
                          // tem início no endereço 0x00000000
                          // da memória de dados (RAM)
    int *p = array;
    for(i=4, soma=0; i > 0; i--)
    {
        *p = 2 * i;
        soma = soma + 2 * i;
        p++;
    }
    *p = soma;
}
```

```
# Mapa de registos
  $5: i
  $6: p
  $7: soma
  $8: aux
```

Address	Code	Instr			Comment
		main:	addi	$6,$0,0	p = 0x00000000;
			addi	$5,$0,4	i = 4;
			add	$7,$0,$0	soma = 0;
		for:	slt	$1,$0,$5	while(i > 0)
			beq	$1,$0,endf	{
			add	$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		

Observe o conteúdo da memória, no segmento de dados (endereço **0x00000000**).

Quantos ciclos de relógio demora a execução completa do programa ?

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 4 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 (memória RAM), a partir do endereço **0x00000000**. Para executar e testar este programa será então necessário alterar o código VHDL da memória de modo a inicializar as primeiras 6 *words* (que correspondem às primeiras 6 posições da memória) com os valores do *array*.

Sugerem-se os seguintes valores: **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 := (
              --            .data
              -- 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");
```

Programa de ordenação de um *array* de inteiros:

```
#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);
}
```

Preencha a tabela seguinte com os valores do *array* no final de cada iteração do ciclo principal do programa de ordenação (do… while()).

Tabela 3. Valores do *array* no final de cada iteração do ciclo principal da ordenação.

Memory Address	Valores iniciais	Fim da 1ª iteração	Fim da 2ª iteração	Fim da 3ª iteração	Fim da 4ª iteração	Fim da 5ª iteração
0x00	20					
0x04	17					
0x08	−2					
0x0C	25					
0x10	5					
0x14	−1					

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

Address	Code	Instr	Comment
		main: addi $5,$0,0	$5=&array[0]
		addi $4,$0,5	j = SIZE − 1
		do:	do {
		addi $2,$0,1	flag = 1
		addi $3,$0,0	i = 0
		for: slt $1,$3,$4	while(i < j)
		beq $1,$0,endfor	{
		add $6,$3,$3	
		add $6,$6,$6	aux = i << 2
		add $6,$6,$5	$6 = array + i
		lw $7,0($6)	$7=array[i]
		lw $8,4($6)	$8=array[i+1]
		slt $1,$8,$7	if($7 > $8)
		beq $1,$0,endif	{
		sw $7,4($6)	array[i+1]=$7
		sw $8,0($6)	array[i]=$8
		addi $2,$0,0	flag = 0
		endif:	}
		addi $3,$3,1	i++
		j for	}
		endfor:addi $4,$4,−1	j−−
		beq $2,$0,do	} while(flag == 0)
		nop	
		w1: beq $0,$0,w1	while(1);

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

6. Reponha a geração do relógio através da tecla **KEY[0]** e do *debouncer* (desligue a saída do divisor de frequência: "**clkOut => open**"). Execute novamente o programa, instrução a instrução, e verifique, no final de cada iteração do ciclo principal, se os valores do *array* correspondem aos que encontrou anteriormente (Tabela 3).

PDF gerado em 17/10/2019