AULAS PRÁTICAS N.º 11 e 12

Objetivos

- Montar os elementos operativos/funcionais de um *datapath multi-cycle*.
- Implementar em VHDL e testar os seguintes módulos do *datapath*: módulo de atualização do *Program Counter*, registo de 32 bits, *multiplexer* 4x1 e *Left Shifter* de 2 bits.

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, vai ser implementado o *datapath multi-cycle* que foi apresentado nas aulas teóricas. Este *datapath* terá suporte para a execução das seguintes instruções: **`ADD`**, **`SUB`**, **`AND`**, **`OR`**, **`NOR`**, **`XOR`**, **`SLT`**, **`ADDI`**, **`SLTI`**, **`LW`**, **`SW`**, **`BEQ`** e **`J`**.

A maioria dos módulos implementados para a versão *single-cycle* do *datapath* será reaproveitada para a implementação da versão *multi-cycle*. Será necessário, no entanto, desenvolver, para além da Unidade de Controlo, os seguintes módulos: registo de 32 bits com *enable*; *multiplexer* 4x1 genérico; módulo de atualização do *Program Counter* e *Left Shifter* de 2 bits.

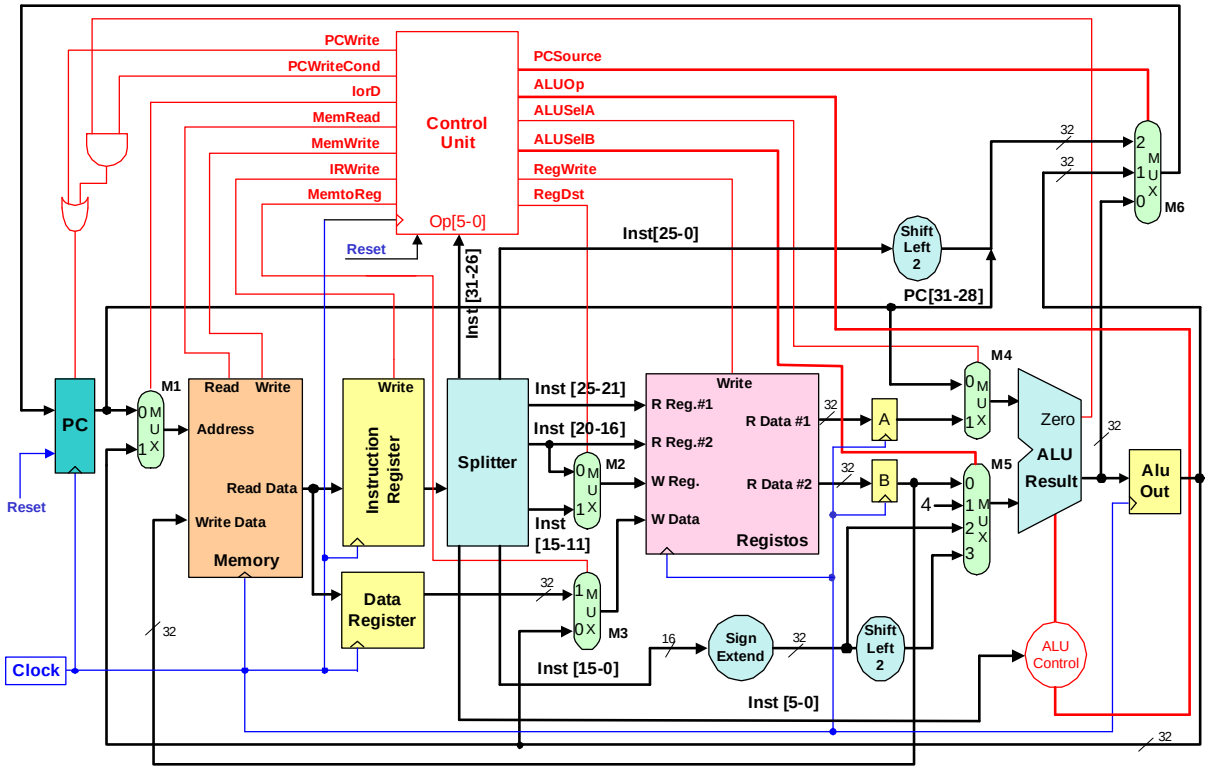

Figura 1. *Datapath multi-cycle*.

Módulo de atualização do *Program Counter*

Relativamente à versão *single-cycle*, este módulo tem que ser alterado, uma vez que agora quer o incremento do PC quer o cálculo do *Branch Target Address* são feitos na ALU (na primeira e na segunda fase de execução da instrução, respetivamente). Deverá, no entanto, continuar a calcular o *Jump Target Address* (JTA). Assim, este módulo deve ter as seguintes entradas (ver Figura 2):

- os 26 bits menos significativos do código máquina da instrução, necessários para calcular o *jump target address*;
- a saída da ALU, necessária para obter o valor de PC+4, calculado na primeira fase de execução da instrução;
- a saída do registo AluOut, necessária para obter o valor do *Branch Target Address* (BTA), calculado na segunda fase de execução da instrução;
- o sinal "**`Zero`**" gerado pela ALU;
- os sinais gerados pela Unidade de Controlo, "**`PCSource`**", "**`PCWrite`**" e "**`PCWriteCond`**".

Para além destes é necessário ainda incluir o sinal de "**`Reset`**" necessário para repor a zero o valor do *Program Counter*.

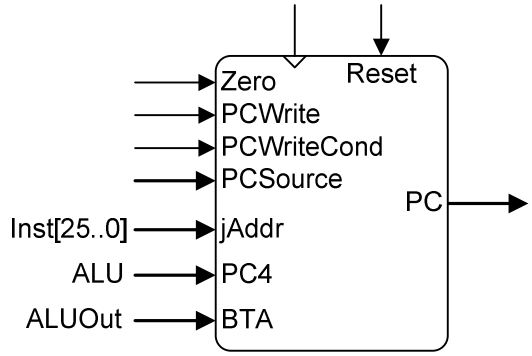

Figura 2. Módulo de atualização do PC.

A tabela seguinte apresenta o valor que a saída do módulo de atualização do PC irá tomar na próxima transição ativa do relógio, em função dos sinais gerados pela unidade de controlo ("**`PCSource`**", "**`PCWrite`**" e "**`PCWriteCond`**") e pela saída "**`Zero`**" da ALU.

Tabela 1. Valor que a saída do módulo de atualização do PC vai tomar na próxima transição ativa do relógio.

PCWrite	PCWriteCond	Zero	PCSource	PC
0	0	X	XX	PC (não alterado)
0	1	0	XX	PC (não alterado)
0	1	1	01	BTA
1	X	X	00	PC4
1	X	X	10	JTA

Multiplexers

O *multiplexer* M6 deverá estar incluído na descrição comportamental do módulo de atualização do *Program Counter*. Para o *multiplexer* M5 é necessário desenvolver um novo módulo com 4 entradas de 32 bits (ou genérico com N bits) e 2 bits de seleção.

Left Shifter

O *Left Shifter* recupera os 2 bits menos significativos do *offset* da instrução de *branch* que são desprezados no momento da codificação da instrução.

Registos de 32 bits

Na saída de cada elemento operativo é colocado um registo de 32 bits. Este registo armazena o valor calculado durante o ciclo de relógio corrente, deixando-o disponível para ser usado, noutro elemento operativo, no ciclo de relógio seguinte. Para este módulo deve ser incluído um sinal de *enable*, que será colocado a '**1**' em todos os casos em que a escrita é incondicional (Data Register, A, B e AluOut).

Memória de dados e instruções

A memória RAM armazena os dados e as instruções do programa. Para esse efeito pode ser reutilizado o módulo desenvolvido para o *datapath single-cycle*, isto é, uma memória RAM de 64 posições de 32 bits cada (256 bytes). Na divisão do espaço de armazenamento, as primeiras 32 posições de memória de 32 bits (endereços **0x00** a **0x7F**, considerando a memória *byte-addressable*) serão utilizadas para o armazenamento do código máquina das instruções do programa e a capacidade restante (endereços **0x80** a **0xFF**) para o armazenamento de dados.

A FPGA usada permite a inicialização de uma RAM, de modo similar ao que fez para a memória ROM. O código seguinte exemplifica o procedimento:

```
architecture Behavioral of RAM is
  constant NUM_WORDS : positive := (2 ** ADDR_BUS_SIZE );
  subtype TData is std_logic_vector(DATA_BUS_SIZE-1 downto 0);
  type TMemory is array(0 to NUM_WORDS - 1) of TData;
  signal s_memory : TMemory := ( X"8C010000",  -- lw    $1,0x0000($0)
                                 X"20210004",  -- addi  $1,$1,4
                                 X"AC010004",  -- sw    $1,4($0)
                                 others => X"00000000");
begin
  (...)
end Behavioral;
```

O endereço de acesso à memória é proveniente do módulo de atualização do PC (acesso durante a fase *Instruction Fetch*), ou é o resultado do valor calculado na ALU na fase *Execute* da instrução (acesso durante a fase *Memory*). Em ambos os casos esse endereço tem uma dimensão de 32 bits. Como já visto anteriormente, desses 32 bits apenas vão ser necessários 6 (2^6=64): são ignorados os 2 bits menos significativos e usados os 6 seguintes (A_7 a A_2). O *multiplexer* M1 deverá, assim, ter uma dimensão de 6 bits (i.e., 2x1 de 6 bits).

Guião

Parte I

1. Crie no Quartus um projeto, selecionando como FPGA o dispositivo Altera Cyclone IV EP4CE115F29C7. Pode designar o projeto e a entidade *top-level* por "**mips_multi_cycle**".
2. Implemente em VHDL o módulo de atualização do *Program Counter*. Poderá designar este módulo por "**PCupdate.vhd**". Selecione este ficheiro como o *top-level* do projeto.
3. Usando o simulador *University Program VWF*, simule funcionalmente o módulo "**PCupdate.vhd**". Verifique o valor de saída do módulo nos seguintes casos (não se esqueça de selecionar, para os sinais correspondentes a barramentos, a opção "radix=hexadecimal"):
   a) **PCWrite='1'; PCWriteCond='0'; PCSource="00"**
   b) **PCWrite='1'; PCWriteCond='0'; PCSource="10"; instr[25..0]=0x05**
   c) **PCWrite='0'; PCWriteCond='1'; Zero='0'; PCSource="01"; BTA=0x0C**
   d) **PCWrite='0'; PCWriteCond='1'; Zero='1'; PCSource="01"; BTA=0x0C**
4. Implemente em VHDL um *multiplexer* 4x1 genérico. Poderá designar este módulo por "**Mux4N.vhd**".
5. Implemente em VHDL o *Left Shifter* de 2 bits. Poderá designar este módulo por "**LeftShifter.vhd**".
6. Implemente em VHDL um registo de 32 bits com *enable*, isto é, a escrita só é realizada se, quando ocorrer uma transição de '**0**' para '**1**' no relógio, o sinal de *enable* estiver ativo.
7. Crie o ficheiro para a unidade de controlo. Defina a entidade com todos os portos de entrada e de saída e deixe, por agora, a implementação da arquitetura em branco. Deste modo poderá já ligar todos os sinais provenientes da unidade de controlo aos vários elementos operativos / funcionais.

Parte II

1. Crie um novo ficheiro "**mips_multi_cycle.vhd**" onde deverá instanciar e interligar todos os módulos que constituem o *datapath multi-cycle*, incluindo a unidade de controlo ainda sem descrição comportamental. O sinal de *clock* deve ser ligado, através do *debouncer*, a uma tecla (**KEY[0]**) da placa de desenvolvimento. O sinal de *reset* deve ser ligado à tecla **KEY[1]** (não se esqueça de inverter este sinal).
2. Na instanciação do módulo de visualização defina a constante genérica "**datapathType**" com a constante "**MULTI_CYCLE_DP**":
   ```
   generic map (datapathType => MULTI_CYCLE_DP)
   ```
3. Verifique as ligações dos sinais globais do módulo de visualização, de modo a poder observar:
   - o valor à saída do módulo de atualização do PC (sinal "**DU_PC**");
   - o conteúdo dos registos do *Register File* (sinais "**DU_RFdata**" e "**DU_RFaddr**");
   - o conteúdo da memória (sinais "**DU_DMdata**" e "**DU_DMaddr**");
   - o valor à saída da ALU (sinal "**DU_IMdata**" anteriormente usado para observar a memória de instruções);
   - o estado atual da máquina de estados da unidade de controlo (sinal "**DU_CState**").
4. Efetue a síntese e implementação do projeto e elimine todos os erros que forem detetados pela ferramenta.

PDF gerado em 28/11/2017