Universidade de Aveiro

Departamento de Eletrónica, Telecomunicações e Informática

Laboratório de Sistemas Digitais

Proposta de Projeto Final

Ano letivo 2019/20

Projeto nº 04 – Modelação dum processador simplificado

1. Introdução

O objetivo deste trabalho é modelar em VHDL e validar por simulação um processador simplificado cujo diagrama de blocos é apresentado na Figura 1. Basicamente o processador é capaz de realizar um conjunto de operações aritméticas/lógicas sobre dois operandos guardados em registos de 8 bits (bloco *Registers*) e armazenar o resultado também num registo. Existem apenas 8 registos, por isso a maior parte dos dados está guardada na memória de dados (*DMemory*) de dimensão 256×8 (256 palavras de 8 bits). Estes dados são transferidos para registos para poderem ser processados e no sentido inverso para serem armazenados.

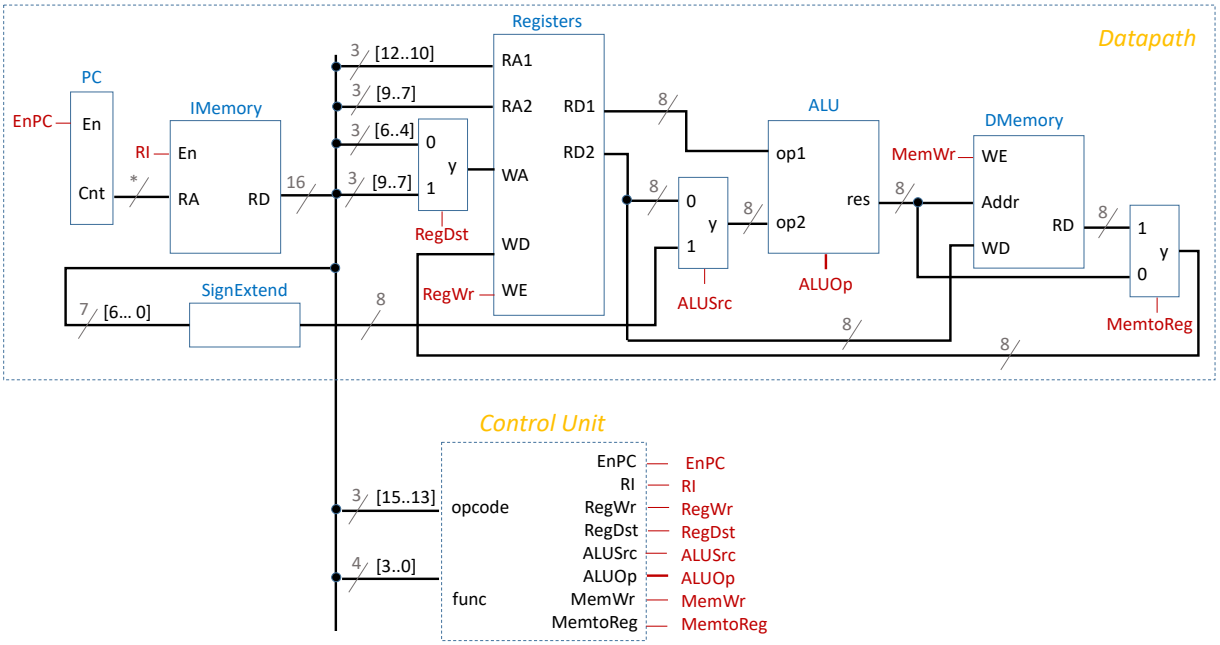

Figura 1 – Diagrama de blocos do processador.

2. Tipos de instruções

O processador é capaz de executar uma série de instruções armazenadas na memória de instruções (*IMemory*). Uma instrução codifica e indica o que se pretende fazer. Há dois tipos de instruções: I e II; o seu formato está ilustrado na Figura 2. Todas as instruções são de 16 bits.

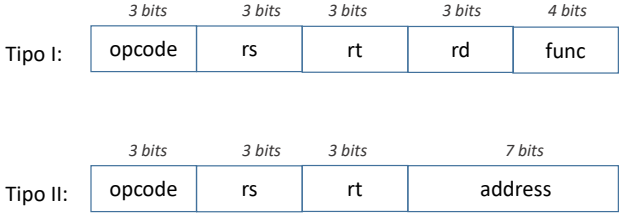

Figura 2 – Formato de instruções suportadas.

As instruções do **tipo I** permitem realizar uma operação aritmética ou lógica sobre dois operandos lidos dos registos *rs* e *rt* e guardar o resultado no registo *rd*. O campo *func* indica o tipo de operação a realizar. Exemplos de instruções do tipo I:

- **ADD** $*rs*, $*rt*, $*rd* --->>> registo$_{rd}$ = registo$_{rs}$ + registo$_{rt}$
- **SUB** $*rs*, $*rt*, $*rd* --->>> registo$_{rd}$ = registo$_{rs}$ – registo$_{rt}$
- **XOR** $*rs*, $*rt*, $*rd* --->>> registo$_{rd}$ = registo$_{rs}$ $\oplus$ registo$_{rt}$

As instruções do **tipo II** permitem:
- ler o conteúdo do registo *rt* e guardá-lo na memória de dados (*DMemory*) – instrução **SW** (*save word*);
- realizar a operação inversa, i.e. ler um dado da memória e armazená-lo no registo *rt* – instrução **LW** (*load word*);
- somar o conteúdo do registo *rs* com uma constante, especificada no campo *address*, e guardar o resultado no registo *rt* – instrução **ADDI** (*add immediate*);

Exemplos de instruções do tipo II:

- **SW** $*rs*, $*rt*, address --->>> *DMemory*[registo$_{rs}$ + address] = registo$_{rt}$
- **LW** $*rs*, $*rt*, address --->>> registo$_{rt}$ = *DMemory*[registo$_{rs}$ + address]
- **ADDI** $*rs*, $*rt*, address --->>> registo$_{rt}$ = registo$_{rs}$ + address

Diferentes instruções são distinguidas pelo código *opcode* de acordo com a Tabela 1. Notem que para além de instruções dos tipos I e II, há uma instrução especial, **NOP**, que não faz nada.

Tabela 1 – Valor do campo *opcode* para várias instruções.

opcode	**Instrução**
000	**NOP** – não fazer nada
001	Todas as instruções aritméticas ou lógicas (a operação a executar é definida pelo campo *func*) – tipo I
100	**ADDI** - soma o conteúdo dum registo com uma constante – tipo II
110	**SW** – transferir dados dum registo para a memória de dados – tipo II
111	**LW** – transferir dados da memória de dados para um registo – tipo II

O campo *func* indica o tipo de operação a realizar de acordo com a Tabela 2:

Tabela 2 – Operações suportadas pela ALU.

func	**Operação**
0000	**ADD** – soma
0001	**SUB** – subtração
0010	**AND**
0011	**OR**
0100	**XOR**
0101	**NOR**
0110	**MUU** - multiplicação sem sinal (só é usada a metade menos significativa do resultado)
0111	**MUS** - multiplicação com sinal (só é usada a metade menos significativa do resultado)
1000	**SLL** - deslocamento lógico do registo$_{rs}$ à esquerda registo$_{rt}$ bits
1001	**SRL** - deslocamento lógico do registo$_{rs}$ à direita registo$_{rt}$ bits
1010	**SRA** - deslocamento aritmético do registo$_{rs}$ à direita registo$_{rt}$ bits
1011	**EQ** – *equal* - o resultado é 1 se registo$_{rs}$ = registo$_{rt}$
1100	**SLS** – *set less than signed* – o resultado é 1 se *signed*(registo$_{rs}$) < *signed*(registo$_{rt}$)
1101	**SLU** – *set less than unsigned* – o resultado é 1 se *unsigned*(registo$_{rs}$) < *unsigned*(registo$_{rt}$)
1110	**SGS** – *set greater than signed* – o resultado é 1 se *signed*(registo$_{rs}$) > *signed*(registo$_{rt}$)
1111	**SGU** – *set greater than unsigned* – o resultado é 1 se *unsigned*(registo$_{rs}$) > *unsigned*(registo$_{rt}$)

3. Arquitetura

Regressando à Figura 1, pode-se observar que o processador inclui uma unidade de execução (*Datapath*) e uma unidade de controlo (*Control Unit*).

A unidade de execução (*Datapath*) deste processador é composta pelos blocos seguintes:

- PC (*Program Counter*) – é um contador que guarda o endereço da próxima instrução a ser executada.
- IMemory (*Instruction Memory*) – memória que guarda instruções dum programa.
- Registers – bloco de 8 registos de 8 bits cada. O registo 0 é um registo especial – guarda sempre o valor 0 e não pode ser escrito.
- ALU – unidade aritmética e lógica que realiza as operações da Tabela 2.
- DMemory (*Data Memory*) – memória 256×8 que guarda os dados dum programa.
- SignExtend – bloco que realiza extensão de sinal num vetor de 7 bits produzindo o resultado de 8 bits.
- 3 *multiplexers* 2:1.

A unidade de controlo (*Control Unit*) descodifica o código de instrução *opcode* e o código de operação *func* e gera sinais de controlo (escritos a vermelho na Figura 1) de acordo com o diagrama de estados da Figura 3 (para a instrução **NOP** todos os sinais de controlo devem ser inativos). Note que o diagrama não está completo pois não especifica os sinais de controlo ativos para cada um dos estados. O diagrama ilustra que o ciclo de execução de cada instrução inclui 4 fases principais:

- *Fetch* – ler instrução da memória de instruções e atualizar o *Program Counter*
- *Decode* – ler conteúdos de registos
- *Execute* – realizar cálculos na ALU (por exemplo, operação aritmética/lógica para instruções do tipo I ou o cálculo de endereço da memória de dados para instruções **SW** e **LW**)
- *Update* (*RegUpdate* ou *WriteMem*) – guardar o resultado num registo (para instruções do tipo I ou **LW/ADDI**) ou na memória (para **SW**)

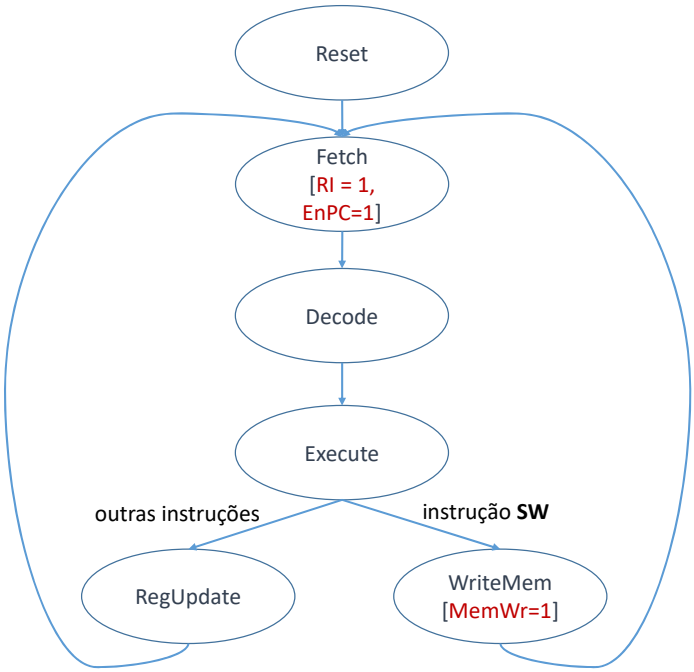

Figura 3 – Diagrama de estados (incompleto) da unidade de controlo.

4. Requisitos

- O *reset* global do sistema só deve afetar (anular) o valor do *Program Counter* e o conteúdo de todos os registos. As memórias (quer de dados quer de instruções) não são afetadas pelo *reset*.
- O sincronismo deve ser assegurado por um só sinal de relógio.
- O *top-level* do sistema deve ser implementado, preferencialmente, com recurso a representação estrutural em VHDL.

5. Implementação

Sugere-se um desenvolvimento faseado, de acordo com a descrição que se segue.

Fase 1 – implementação e teste da unidade de execução
Modele em VHDL, sintetize e valide por simulação todos os blocos da unidade de execução (*datapath*). Tente criar blocos parametrizáveis pois isto facilitará ajustes futuros no sistema. Assuma que a memória de instruções pode armazenar no máximo 16 instruções do grupo seguinte: {**LW**, **SW**, **ADDI**, **ADD**, **MUU**}. Inicialize a memória de dados com os valores: `(X"05",X"03", others => X"00")`. Inicialize a memória de instruções com o programa de teste seguinte (codifique as instruções de acordo com a Figura 2 e as Tabelas 1 e 2):

```
LW   $0, $1, 0  -- carregar no registo 1 dados que se encontram no endereço [registo0+0] na memória de dados
ADDI $0, $2, 1  -- realizar a operação registo2 = registo0 + 1
MUU  $1, $2, $3 -- realizar a operação registo3 = registo1 MUU registo2
ADD  $0, $3, $4 -- realizar a operação registo4 = registo0 + registo3
ADDI $2, $2, 1  -- realizar a operação registo2 = registo2 + 1
MUU  $1, $2, $3 -- realizar a operação registo3 = registo1 MUU registo2
ADD  $4, $3, $4 -- realizar a operação registo4 = registo4 + registo3
ADDI $2, $2, 1  -- realizar a operação registo2 = registo2 + 1
MUU  $1, $2, $3 -- realizar a operação registo3 = registo1 MUU registo2
ADD  $4, $3, $4 -- realizar a operação registo4 = registo4 + registo3
SW   $0, $2, 2  -- escrever no endereço [registo0+2] na memória de dados o conteúdo do registo2
SW   $0, $4, 3  -- escrever no endereço [registo0+3] na memória de dados o conteúdo do registo4
```

Interligue todos os blocos entre si, conforme a Figura 1, e simule o comportamento de toda a unidade de execução.

Fase 2 – Implementação e teste da unidade de controlo
Complete o diagrama de estados da unidade de controlo da Figura 3. Especifique em VHDL e valide por simulação a unidade de controlo.

Fase 3 – Interligação e teste do processador completo (unidade de execução + unidade de controlo)
Interligue a unidade de controlo e a de execução e teste, no simulador, todo o sistema.

Fase 4
Adicione uma instrução nova: **BNE** (*branch on not equal*), do tipo II (cujo *opcode* é 011). Esta instrução compara o conteúdo dos registos *rs* e *rt* e, caso sejam diferentes, "salta" *address* instruções.

Exemplo da instrução **BNE**:
- **BNE** $*rs*, $*rt*, *3* --->>> se *(*registo$_{rs}$ /= registo$_{rt}$*) avançar (saltar por cima de) 3 instruções no código*

Note que deverá fazer alterações quer na unidade de execução, quer na unidade de controlo.
Escreva um programa que ordene, por ordem crescente, um *array* de 3 números com sinal, armazenados nos endereços 0, 1, e 2 na memória de dados e escreve o resultado na mesma zona na memória. Comente cada instrução do seu código. Traduza o programa no código de máquina e teste-o. Note que pode ser necessário aumentar a profundidade da memória de instruções.

PDF gerado em 04/05/2020