Desenvolvimento de um processador de 8 bits em VHDL

Alfredo Matos

***Resumo* – Este artigo descreve o desenvolvimento de um processador de 8 bits que executa operações aritméticas e lógicas, auxiliado por uma memória de registo e outra de armazenamento. O processador tem de executar algumas operações particulares que incluem operações entre dados residentes na memória de armazenamento, sem recorrer a memória de registos, leitura de informação da placa FPGA, e também impressão de resultados num meio visual.**

***Abstract* – This paper describes the development of an 8 bit processor that executes arithmetic and logical operations, aided by an instruction memory and a data memory. The processor has to execute some particular operations which include operations on the words stored in the data memory without recurring to the register memory, reading information from the FPGA board and displaying the results on a visual media.**

I. INTRODUÇÃO

O trabalho aqui descrito é o resultado do projecto de avaliação proposto pelo Professor Valery Sklyarov na cadeira Computação Reconfigurável [1]. Para a implementação do processador proposto utilizou-se uma placa Xilinx TE-XC2Se [2], assim como o ambiente de desenvolvimento da Xilinx ISE 5.2. A implementação foi escrita em VHDL, sem recorrer aos núcleos IP disponibilizados pela plataforma pois poderiam levar a um menor conhecimento sobre os mecanismos da linguagem. No desenvolvimento deste processador existiram duas fases claramente distintas. A primeira foi do desenvolvimento da especificação do processador que responde aos objectivos propostos, com o recurso à bibliografia especificada [3], que claramente pertence a área de Arquitectura de Computadores. A segunda fase foi a implementação do conjunto de instruções que resultaram da arquitectura criada.

A. Objectivos

O projecto consiste em implementar um processador que seja capaz de efectuar as seguintes operações:

1. Ler operando A (B) da memória no endereço N.
2. Escrever operando A (B) na memória no endereço N.
3. Realizar operação A = A # B (# = soma (+), subtracção (-), multiplicação (*), AND, OR ou XOR);
4. Realizar operação A # B e escrever o resultado na memória no endereço N;
5. Fazer operação A = A # (operando B lido da memória no endereço N);
6. Efectuar um atraso de A segundos;
7. Mostrar o valor da variável A (B) (lido da memória no endereço N) no display LCD;
8. Ler o valor (8 bits) marcado nos interruptores;
9. Fazer inversão da variável A;
10. Fazer deslocamento (shift) à direita/esquerda da variável A por B bits;
11. Trocar os valores de A e B.

Fig. 1 - Formato das instruções

II. ARQUITECTURA DO PROCESSADOR

Para atingir os objectivos propostos é preciso primeiro desenvolver um conjunto de instruções que permita realizar as operações desejadas. De seguida é necessário definir o formato das mesmas, bem como a dimensão das memórias de armazenamento e de registos.

A. Conjunto de Operações

Após cuidado estudo do problema, chegou-se Ã tabela 1, que representa todas as instruções que o processador deverá executar. No final da fase de análise do conjunto de instruções totalizamos um total 27 instruções, que incluem operações lógicas, aritméticas, com armazenamento em registos e directamente na memória de armazenamento, operações de leitura e escrita na memória de armazenamento e ainda operações entrada e saída de informação através da FPGA.

B. Formato das instruções

O próximo passo é definir o formato ideal para as instruções. Para isso é necessário considerar todos os tipos de instruções e a informação que é necessária transportar em cada instrução. Dadas as exigências das instruções o formato escolhido, representado na figura 1, opta por levar todos os três registos que serão necessários ainda com o endereço ou tempo de atraso desejado para a instrução de delay. Assim a opção recai em considerar apenas um tipo de instrução que contivesse todos os elementos necessários a todas as instruções.

A configuração proposta resulta num total de 21 bits por instrução. O campo de operador necessita de 5 bits pois é necessário codificar 27 instruções, o que implica um mínimo de 5 bits pois 4 codificam 16 possibilidades, e 5 bits codificam 32, pois $2^4 = 16$ e $2^5 = 32$. Para os campos que indicam registos, estes têm de ter 4 bits, para codificar um total de 16 registos. De notar que sendo este um processador de 8 bits, cada registo tem o tamanho fixo de 8 bits podendo assim representar valores no intervalo inteiro de [-128; 127]. O endereço necessita de 4 bits pois a memória foi reduzida a um total de 16 endereços possíveis. As configurações de memórias são especificadas em II-C. O Stall, ou tempo de atraso, codifica o número de segundos

#	Representação	Descrição	Função
1	lb r1,addr	load byte	r1=addr[8][1]
2	sb r1,addr	store byte	addr[8]=r1
3	add r1,r2,r3	adição	r3=r1+r2
4	sub r1,r3,r3	subtracção	r3=r1−r2
5	mult r1,r2,r3	multiplicação	r3=r1∗r2
6	and r1,r2,r3	and lógico	r3 = r1&r2
7	or r1,r2,r3	or lógico	r3=r1|r2
8	xor r1,r2,r3	xor lógico	r3=r1⊕r2
9	addm r1,r2,addr	adição com mem	r2=r1+addr[8]
10	subm r1,r3,addr	subtracção com mem	r2=r1−addr[8]
11	multm r1,r2,addr	multiplicação com mem	r2=r1∗addr[8]
12	andm r1,r2,addr	and lógico com mem	r2=r1&addr[8]
13	orm r1,r2,addr	or lógico com mem	r2=r1|addr[8]
14	xorm r1,r2,addr	xor lógico com mem	r2=r1⊕addr[8]
15	sadd r1,r2,addr	store add	addr[8]=r1+r2
16	ssub r1,r3,addr	store sub	addr[8]=r1−r2
17	smult r1,r2,addr	store mult	addr[8]=r1∗r2
18	sand r1,r2,addr	store and	addr[8]=r1&r2
19	sor r1,r2,addr	store or	addr[8]=r1|r2
20	sxor r1,r2,addr	store xor	addr[8]=r1⊕r2
21	delay r1	Atraso	
22	inv r1,r2	Inversão de registos	r2= inverter r1
23	sll r1,r2,r3	deslocamento lógico à esquerda	r3=r1<<r2
24	srl r1,r2,r3	deslocamento lógico à direita	r3=r1>>r2
25	swt r1,r2	trocar registos	r1=r2 & r2=r1
26	print_byte r1	imprime byte no lcd	
27	read_byte	lê byte interruptores DIP	

TABLE I
CONJUNTO DE INSTRUÇÕES A IMPLEMENTAR.

de atraso, e dado que não deverá ser necessário atrasar mais do que 16 segundos o funcionamento do processador, podemos utilizar os mesmos 4 bits que utilizamos na codificação de um endereço.

C. Memórias

Na construção de um processador necessitamos tanto de uma memória de registos, como de uma memória principal. Na secção II-B for referido que existem 16 registos para operações do processador. Estes 16 registos ocupam um total de 16 bytes. A memória principal tem um tamanho total de 8 bytes, pois aloja 8 registos. O tamanho reduzido das memórias advém directamente da natureza do processador, que é experimental, ou seja, uma prova de conceito para implementação em VHDL, portanto o interesse cai mais em testar operações do que propriamente armazenar informação em memória. Portanto, os 16 e 8 registos satisfazem as reduzidas necessidades para concretizar os objectivos estabelecidos. De referir que ambas as memórias poderiam ser facilmente aumentadas, dado que devidamente acompanhadas do aumento do tamanho das instruções.

[1]addr[8] é um apontador para uma zona de memória com 8 bits, ou seja, um registo na memória RAM.

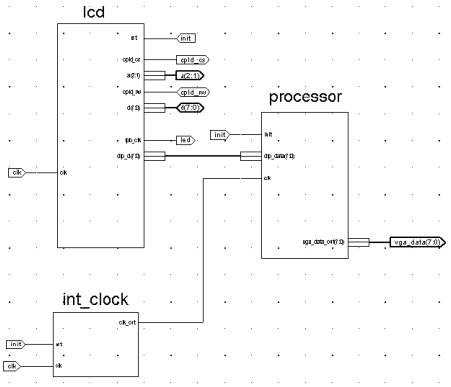

Fig. 2 - Processador e unidades auxiliares

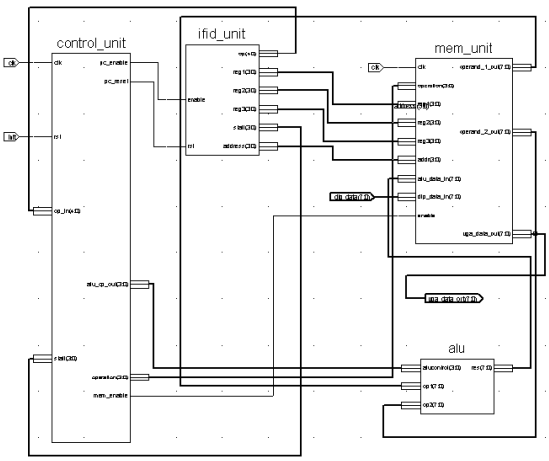

Fig. 3 - Blocos funcionais do processador

III. IMPLEMENTAÇÃO

Dado que definimos concretamente a especificação do processador, passamos à prática e a como implementar o que foi previamente descrito. Para a descrição da solução obtida fazemos uma análise que segue de um maior nível de abstracção até chegarmos aos componentes básicos do processador.

Como podemos ver pela figura 2 existem para além do processador duas unidades. Estas definem a interacção com as funções da FPGA (saÃda vga e visor LCD), e relógios para o funcionamento do processador (int_clock).

Avançando na implementação, passamos a analisar o processador mais concretamente. Podemos ver pela figura 3 que este se encontra dividido em quatro grandes áreas funcionais: a unidade de controlo (control_unit), a unidade de processamento de instruções (ifid_unit), a unidade de memória (mem_unit) e a ALU (alu).

A. Bloco IFID_UNIT

A ifid_unit (Instruction Fetch, Instruction decode) é um bloco que contém um Program Counter que nos indica a instrução a executar, e que de seguida vai Ã memória de instruções buscar a instrução necessária e a divide pe-

Nome	Código	Descrição
lb	0001	Operação de leitura da memória principal para registo (load byte)
sb	0010	Operação de escrita de um registo para memória principal (store byte)
arit_reg	0011	Operação aritmética só com registos
arit_memreg	0100	Operação aritmética com um registo e memória
arit_memout	0101	Operação aritmética com escrita do resultado para memória
delay	0110	Operação de atraso
switch	0111	Operação de trocas de registos
print	1000	Operação de impressão de valor no vga
read	1001	Operação de leitura dos DIP Switches

TABLE II
FAMILIA DE OPERAÇÕES

los sinais necessários que são passados aos demais componentes. São eles o código da operação e os segundos de atraso que vão para a unidade de controlo, e os 3 registos e endereço de memória que vão para a unidade de memória. Apenas de referir que a memória de instrução é uma ROM com capacidade para 16 instruções, valor suficiente para pequenas rotinas de teste. Esta unidade é activada por um sinal de enable, que quando posto a 1, executa o processamento descrito atrás.

B. Bloco ALU

A alu é um componente bastante simples que executa as operações aritméticas e lógicas necessárias a este processador. Para o efeito aceita os dois operandos vindos da unidade de memória e a operação a realizar da unidade de controlo. As operações possíveis são as mais vulgares: Soma, subtracção, multiplicação, ”e”lógico, ”ou”lógico, ”ou”exclusivo lógico, negação lógica, deslocamento lógico à esquerda e à direita.

C. Bloco mem_unit

A mem_unit contém a memória de registos e a memória principal. Aceita os registos e endereço vindos da ifid_unit e um código de operação enviado pela unidade de controlo. A cada sinal de relógio do processador e consoante o sinal de activação da unidade (enable) executa a operação desejada. Destas operações destacam-se a capacidade de ler e escrever registos no mesmo ciclo, assim como o acesso a recursos da placa FGPA (DIP Switches, e registo de output).

D. Bloco control_unit

A control_unit é o ”cérebro”do processador e controla todo o seu funcionamento. Para isso apoia-se numa máquina de estados finitos (FSM). A FSM utilizada tem como estados as diferentes fases do processador, e é definida a evolução pelo grupo da instrução a executar.

Isto implica que tivesse sido feito um estudo prévio das fases comuns a cada instrução, e o posterior agrupamento em famílias, descritas na tabela III-D que têm as mesmas fases. Após serem claramente definidos os grupos, resta

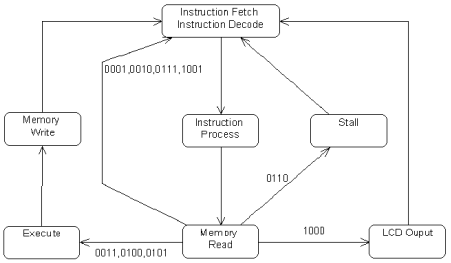

Fig. 4 - Diagrama de estados da Finite State Machine da unidade de controlo

apenas definir o fluxo do diagrama de estados, para cada grupo de instrução. Por fim a máquina de estados traduziu-se no diagrama apresentado na figura 4.

IV. CONCLUSÃO

Uma vez desenvolvida a arquitectura do processador, a implementação de cada unidade não é muito complexa. O grande problema coloca-se na sincronização de todos os blocos, uma vez que o funcionamento de cada um dos blocos foi testado exaustivamente no simulador disponibilizado pelo software [2]. O facto da unidade de sinal de relógio se encontrar fora do bloco do processador também foi benéfico, uma vez que facilita a alteração da frequência dos componentes do processador simultaneamente, que necessitam de sincronismo, especialmente a unidade de controlo. Os objectivos propostos foram integralmente atingidos, sendo de notar que houve uma tentativa de desenvolver uma arquitectura plausível e não simplesmente algo que resolvesse o problema em causa. O maior período de desenvolvimento foi gasto na sincronização entre diferentes componentes, de maneira a todas as fases conjugarem correctamente, e também na planificação da arquitectura. De futuro seria interessante promover a detecção de *overflows* na ALU. O objectivo superlativo deste trabalho, que foi plenamente atingido, foi o de integrar conhecimentos adquiridos na cadeira de Computação Reconfigurável, assim como a utilização de uma FPGA, nomeadamente a placa TE-XC2Se da Trenz.

V. AGRADECIMENTOS

Os agradecimentos vão para o Professor Valery Sklyarov pelo acompanhamento do trabalho, assim como a oportunidade de escrita deste artigo, para a Professora Iouliia Skliarova, pela ajuda na revisão do artigo, e por fim para os colegas que ajudaram a solucionar alguns dos problemas que surgiram no decorrer do trabalho.

REFERENCES

[1] Valery Sklyarov, *DocumentaÃ§Ã£o da disciplina ComputaÃ§Ã£o ReconfigurÃ¡vel, 2Âº semestre, LECT*, Universidade de Aveiro, 2004.
URL: *http://elearning.ua.pt*

[2] Xilinx, *ISE 5.2, Xilinx series FPGA.*
URL: *http://www.xilinx.com*

[3] David A. Patterson e John L. Hennessy, *Computer Organization Design: The Hardware/Software Interface*, Morgan Kaufman, 2Âª edição, Ag. de 1998.