Instituto Superior de Engenharia de Lisboa

ARQUITETURA DE COMPUTADORES

Relatório do Primeiro Trabalho Prático

Docente

Prof. Hernâni Mergulhão

Alunos

43552 Samuel Costa
43884 Pedro Rocha

Março 2017

Índice

Introdução

Este relatório, referente ao primeiro de três trabalhos práticos, foi elaborado no âmbito da unidade curricular de Arquitetura de Computadores e acompanha o estudo de um processador de ciclo único que implementa uma arquitetura de Harvard a 8 bits, conforme descrito no enunciado. Este exercício constituiu uma aplicação de alguns aspetos introdutórios à Arquitetura de Computadores que o presente relatório aprofunda e expande.

Ao longo do documento apresentam-se os principais objetivos metodológicos e práticos delineados pelo enunciado; a metodologia adotada; uma explicação dos procedimentos aplicados na execução das tarefas propostas; e, por fim, uma breve reflexão sobre o trabalho realizado em que se discutem os resultados obtidos, explicitando-se as conclusões suscitadas pelo trabalho realizado.

Objetivos

O trabalho prático visa, em termos gerais, aplicar os conteúdos introduzidos em contexto de sala de aula, em particular, os conceitos introdutórios de Arquitetura de Computadores, aprofundando as suas potencialidades de utilização, por meio do estudo de um processador cuja arquitetura é descrita no enunciado do trabalho.

Definem-se os seguintes objetivos:

- do ponto de vista metodológico: compreender a relação entre a codificação do conjunto de instruções que compõe o ISA de um processador e as saídas produzidas pelo respetivo descodificador de instruções;
- do ponto de vista prático, empregar e dominar os seguintes meios (i) identificar a correspondência entre a arquitetura de um processador e o operation code que realiza uma dada instrução que aquele disponibilize; (ii) reconhecer a diferença entre sinais de encaminhamento e sinais de controlo; (iii) compreender o contributo das saídas do descodificador para a realização de uma instrução; (vi) interpretar criticamente uma sequência de instruções a executar; e por último (v) efetuar trace a uma sequência de instruções realizadas, fornecendo breves comentários sobre cada comando que permitam saber, a dado momento, o estado de diferentes sinais.

Métodos e Recursos Utilizados

Para a realização deste trabalho foi utilizada a aplicação Logisim Versão 2.7.1.

O processador estudado é de ciclo único, ie, executa uma instrução por ciclo de instrução. A execução de cada instrução dá-se em duas fases: uma primeira fase fetch, de preparação, em que os sinais necessários à execução da instrução são disponibilizados e encaminhados; e uma segunda fase, execute, em que a instrução é propriamente executada.

Trabalho Realizado

Codificação das instruções do ISA

opcode

Analisou-se o diagrama de blocos da ALU presente na 3 do Anexo A do enunciado.

Constatou-se que o bit de maior peso do sinal OP_ALU seleciona o resultado de uma operação lógica ou aritmética (OP_ALU1 =1 é selecionado o resultado de uma operação lógica, OP_ALU1=0 é selecionado o resultado de uma operação aritmética).

Por outro lado, o bit de menor peso permite selecionar entre o resultado de uma operação and-lógico (OP_ALU0=0) ou o resultado da operação xor-lógico(OP_ALU0=1). O sinal OP_ALU0, se ativo, efetua uma transformação do operando B de uma operação aritmética, por forma a realizar uma subtração.

Constatou-se ainda, por análise do circuito disponibilizado, que os dois bits d10 e d9 do código de instruções são encaminhados diretamente para OP_ALU.

Assim, tendo em conta que OP_ALU corresponde diretamente aos dois bits à direita no opcode apresentado a seguir, é possível completar o mapa de codificação de instruções da seguinte forma:

Instrução	Opcode
add rx, ry, rz	0000
sub rx, ry, rz	0001
anl rx, ry, rz	0010
xrl rx, ry, rz	0011

Tabela 1 – detalhe do mapa de codificação de instruções

Confirmação por simulação

Foi utilizada a aplicação Logisim para confirmar a solução anteriormente descrita. Para tal, foram afetados o sinal OP_ALU e os operandos A e B com diferentes valores de modo a testar o funcionamento da ALU nas quatro operações disponíveis. A tabela 2 mostra, para cada uma das operações da ALU, os valores de teste e as saídas obtidas.

Instrução	OP_ALU	A	B	R	Z	C
Add	00	10000100	10000101	00001001	0	1
Sub	01	00001001	00001000	00000001	0	0
Anl	10	10101010	01010101	00000000	1	-
Xrl	11	10100000	01011111	1111111	0	-

Tabela 2 – teste da solução proposta

Os resultados obtidos encontram-se em conformidade com o esperado.

Mapa de codificação

O código de instruções do processador estudado é um sinal a 13 bits que pode ser decomposto em 4 bits de código de operação (opcode) e 9 bits de encaminhamento.

Assim, a instrução realizada é codificada pelos quatro bits de maior peso do código de instruções, ie, pelo opcode, e pelas flags C e Z, provenientes do módulo PSW.

Os 2 bits de maior peso do opcode codificam, dentre 3 grandes grupos ou famílias de instruções qual o tipo de instrução efetuada.

- Instruções de afetação da memória de dados (*data memory*) ou do banco de registos;

- Operações da alu;

- Instruções de afetação do *Program Counter* (PC).

Portanto, de acordo com a descrição das instruções e o diagrama de blocos do processador, temos a tabela 3, que completa o mapa de codificação das instruções.

Instrução	**Z**	**C**	**d12**	**d11**	**d10**	**d9**	**d8**	**d7**	**d6**	**d5**	**d4**	**d3**	**d2**	**d1**	**d0**
ldi rx, #const6	_	_	1	0	0	0	rx2	rx1	rx0	c5	c4	c3	c2	c1	c0
ld rx, [ry]	_	_	1	0	0	1	rx2	rx1	rx0	-	-	-	ry2	ry1	ry0
ld rx, direct6	_	_	1	0	1	0	rx2	rx1	rx0	dr5	dr4	dr3	dr2	dr1	dr0
st rx, [ry]	_	_	1	0	1	1	-	-	-	rx2	rx1	rx0	ry2	ry1	ry0
add rx, ry, rz	_	_	0	0	0	0	rx2	rx1	rx0	ry2	ry1	ry0	rz2	rz1	rz0
sub rx, ry, #const3	_	_	0	0	0	1	rx2	rx1	rx0	ry2	ry1	ry0	c2	c1	c0
anl rx, ry, rz	_	_	0	0	1	0	rx2	rx1	rx0	ry2	ry1	ry0	rz2	rz1	rz0
xrl rx, ry, rz	_	_	0	0	1	1	rx2	rx1	rx0	ry2	ry1	ry0	rz2	rz1	rz0
jz offset6	0	_	1	1	0	0	-	-	-	of5	of4	of3	of2	of1	of0
	1	_	1	1	0	0	-	-	-	of5	of4	of3	of2	of1	of0
jnc offset6	_	0	1	1	0	1	-	-	-	of5	of4	of3	of2	of1	of0
	_	1	1	1	0	1	-	-	-	of5	of4	of3	of2	of1	of0
jmp offset6	_	_	1	1	1	0	-	-	-	of5	of4	of3	of2	of1	of0
jmp rx	_	_	1	1	1	1	-	-	-	rx2	rx1	rx0	-	-	-

Tabela 3 – mapa de codificação das instruções

Projeto do descodificador de instruções

Descodificador de instruções

Pela análise da figura 1 do Anexo A do enunciado, foram obtidas as saídas do descodificador, que vêm representadas na tabela 4.

opcode	Z	C	SI	SO	SD	SC	SA	ER	EP	WR	RD	OP_ALU	OffSet6	Const3	Const6	Direct6	AD	AA	AB
1000	-	-	0	0	00	-	-	1	0	0	0	-	-	-	#cont6	-	rx	-	-
1001	-	-	0	0	01	-	1	1	0	0	1	-	-	-	-	-	rx	-	ry
1010	-	-	0	0	01	-	0	1	0	0	1	-	-	-	-	direct6	rx	-	-
1011	-	-	0	0	-	-	1	0	0	1	0	-	-	-	-	-	-	rx	ry
0000	-	-	0	0	10	1	-	1	1	0	0	00	-	-	-	-	rz	rx	ry
0001	-	-	0	0	10	0	-	1	1	0	0	01	-	cont3	-	-	rx	ry	-
0010	-	-	0	0	10	1	-	1	1	0	0	10	-	-	-	-	rx	ry	rz
0011	-	-	0	0	10	1	-	1	1	0	0	11	-	-	-	-	rx	ry	rz
1100	0	-	0	0	-	-	-	0	0	0	0	-	-	-	-	-	-	-	-
1100	1	-	0	1	-	-	-	0	0	0	0	-	Offset6	-	-	-	-	-	-
1101	-	0	0	1	-	-	-	0	0	0	0	-	Offset6	-	-	-	-	-	-
1101	-	1	0	0	-	-	-	0	0	0	0	-	-	-	-	-	-	-	-
1110	-	-	0	1	-	-	-	0	0	0	0	-	Offset6	-	-	-	-	-	-
1111	-	-	1	-	-	-	-	0	0	0	0	-	-	-	-	-	-	rx	-

Tabela 4 - saídas do descodificador de instruções

	Instrução de controlo
	Saídas de controlo
	Sinais da palavra de controlo obtidos diretamente do código da instrução

Os sinais da palavra de controlo obtidos diretamente do código da instrução, poderiam não passar pelo descodificador de instruções, pois não interferem com as saídas de controlo, e simplificariam a construção do mesmo.

Confirmação por simulação

Para testar a solução proposta, foi utilizada a aplicação Logisim, e carregada em memória de código uma instrução de cada uma das instruções disponíveis. Foram também preenchidos os endereços 0,3 e 7 da memória de dados com 0x00, 0x07 e 0x0D respetivamente.

Instrução	Código de instrução	Palavra carregada em *code memory*	Ação realizada
ldi r2, #7	1.0000.1000.0111	1087	Foi transferida para r2 a palavra binária 00000111
ld r1, [r2]	1.0010.0110.1010	126A	Foi transferida para r1 o conteúdo de *data_mem*[7] = 00001101
ld r3, 13	1.0100.1100.1101	14CD	Foi transferida para r3 o conteúdo de *data_mem*[13] = 00000011
st r2, r4	1.0111.1101.0100	17D4	Foi transferido para *data_mem*[0]=7 o conteúdo de r2
add r1, r1, r2	0.0000.0100.1010	004A	Foi transferido para r1 o resultado da adição de r1 com r2. Assume-se que o conteúdo de r1 é 1111.1111 e o conteúdo de r2 é 0000.0001.
sub r2, r4, #5	0.0010.1010.0101	02A5	Foi transferido para r2 o resultado da subtração da constante 5 ao contéudo de r4. Assume-se que r4 tem o valor 0000.0000. r2 toma o valor 1111011.
jz +3	1.1000.0000.0011	1803	Verificou-se que PC foi incrementado em 3 unidades se a flag zero estiver ativa. Caso contrário, é incrementado em 1.
jnc 2	1.1010.0000.0010	1A02	PC foi incrementado em 2 unidades com a flag carry inativa. Verificou-se que na presença de carry, PC é incrementado em 1.
jmp 24	1.1100.0001.1000	1C18	Com PC=0, na execução deste comando, PC toma o valor 0x18.
jmp r2	1.1110.0001.0000	1E10	Assumiu-se que r2 tem valor 0x00 e que PC tem valor 0x18. Na execução deste comando, PC toma valor 0x00.

Tabela 5 - confirmação por simulação

Os resultados obtidos apresentaram-se coerentes com a solução proposta.

Teste da arquitetura

Codificação de instruções

Foram carregados para as cinco primeiras posições de *data memory* os cinco dígitos do numero de aluno do Samuel (43552). O código da tabela 6 foi carregado para os 9 primeiros endereços de *code memory*.

Program	Codigo máquina		Adr
	bin	Hex	
Ldi r0,#4	1.0000.0000.0100	1004	0
Xrl r1,r1,r1	0.0110.0100.1001	0649	1
Ld r2,[r0]	1.0010.1000.0000	1280	2
Add r1,r1,r2	0.0000.0100.1010	004A	3
Sub r0,r0,#1	0.0010.0000.0001	0201	4
Jnc -3	1.1010.0011.1101	1A3D	5
Ldi r0,#20	1.0000.0001.0100	1014	6
St r1,[r0]	1.0110.0000.1000	1608	7
Jmp 0	1.1100.0011.1000	1C38	8

Tabela 6 - código maquina

Trace

O código foi executado no Logisim e foram registadas as alterações que constam na tabela 7.

PC	opcode	mmemonic	Action	comment	
00	1004	ldi r0,#4	r0 = 4		
01	0649	xrl r1,r1,r1	r1 = 0	r1 = 00000000 ⊕ 00000000	
02	1280	ld r2,[r0]	r2 = 2	r2 = memdata[4] = 2	
03	004A	add r1,r1,r2	r1 = 2	r1 = 0 + 2	z=0;c=0
04	0201	sub r0,r0,#1	r0 = 3	r0 = 4 - 1	z=0;c=0
05	1A3D	jnc -3	PC = 2	PC = 5 - 3	c=0
02	1280	ld r2,[r0]	r2 = 5	r2 = memdata[3] = 5	
03	004A	add r1,r1,r2	r1 = 7	r1 = 2 + 5	z=0;c=0
04	0201	sub r0,r0,#1	r0 = 2	r0 = 3 - 1	z=0;c=0
05	1A3D	jnc -3	PC = 2	PC = 5 – 3 <= c=0	
02	1280	ld r2,[r0]	r2 = 5	r2 = memdata[2] = 5	
03	004A	add r1,r1,r2	r1 = 12	r1 = 7 + 5 = r1 + r2	z=0;c=0
04	0201	sub r0,r0,#1	r0 = 1	r0 = 2 - 1 = r0 – 1	z=0;c=0
05	1A3D	jnc -3	PC = 2	PC = 5 – 3 <= c=0	
02	1280	ld r2,[r0]	r2 = 3	r2 = memdata[1] = 3	
03	004A	add r1,r1,r2	r1 = 15	r1 = 12 + 3 = r1 + r2	z=0;c=0
04	0201	sub r0,r0,#1	r0 = 0	r0 = 1 – 1 = r0 - 1	z=1;c=0
05	1A3D	jnc -3	PC = 2	PC = 5 – 3 <= c=0	
02	1280	ld r2,[r0]	r2 = 4	r2 = memdata[0] = 4	
03	004A	add r1,r1,r2	r1 = 19	r1 = 15 + 4	z=0;c=0
04	0201	sub r0,r0,#1	r0 = -1	r0 = 0 – 1	z=0;c=1
05	1A3D	jnc -3	PC = 6	PC = 5 + 1 <= c=1	
06	1014	ldi r0,#20	r0 = 20		
07	1608	st r1,[r0]	memdata[20] = 19		
08	1C38	jmp 0	PC = 0	PC = 8 - 8 <= PC=8	

Tabela 7- trace

Verificou-se que as alterações ocorridas nos registos do processador (r0-r7, PC e PSW) corresponderam ao esperado, dadas as instruções a executar.

Conclusões

Com a realização deste trabalho (tarefas práticas e relatório) foi possível aplicar as ferramentas introduzidas nas aulas de Arquitetura de Computadores, consolidando e aprofundando as suas potencialidades de utilização, nomeadamente através do projeto do descodificador de instruções.

Salienta-se que os dispositivos de memória constantes do ficheiro de simulação disponibilizado (memória de dados e memória de código) apresentam simplificações ao nível da sua implementação, como seja a temporização por clock. De facto, esta característica não é comum no desenho de processadores, tratando-se de uma opção pedagógica. Assim, durante a elaboração do trabalho não foi necessário levar em conta parâmetros de temporização, críticos na interação com a memória, apesar de estes terem sido estudados no mesmo período.

Levando em conta essa opção didática, o facto de a orientação apontada pelo enunciado se concentrar no descodificador de instruções, e a separação entre memória de instruções e de dados característica da arquitetura de Harvard, não foram identificadas alterações ao circuito que não pusessem em causa estes pressupostos do trabalho. Ainda assim, verificou-se que os sinais da palavra de controlo obtidos diretamente do código da instrução, poderiam não passar pelo descodificador de instruções, pois não interferem com as saídas de controlo. Esta alteração representaria uma diminuição da complexidade na construção do descodificador de instruções.

Referências

[1] Logisim - Documentação oficial da versão 2.7.x – disponível em http://www.cburch.com/logisim/pt/docs.html - acedido a 1/04/2017

[2] Textos de apoio às aulas teóricas – CPU v1, Prof. João Pedro Patriarca – disponível em http://pwp.net.ipl.pt/cc.isel/ezeq/arquitetura/textos_apoio/patriarca/CPU_v1.pdf - acedido a 1/4/2017