INSTITUTO SUPERIOR DE ENGENHARIA DE LISBOA

Relatório do 2º trabalho
de
Arquitetura de Computadores

Estudo do Funcionamento de um Processador

Trabalho realizado por:

Nome:	Carolina Pereira	Nº 49470
Nome:	Pedro Malafaia	Nº 49506
Nome:	Roberto Petrisoru	Nº 49418

Docente: Tiago Dias

1. Resposta às perguntas

3.1 Análise da microarquitetura

1 – A afirmação é falsa. Trata-se de uma arquitetura do tipo Harvard pois não contém uma unidade de controlo. Para se tratar de uma arquitetura do tipo von Neumann, teria de apresentar uma unidade de controlo.

2 – O bloco Ext é utilizado para realizar uma extensão de sinal. A extensão de sinal é realizada para evitar que nas instruções mov e sub, o valor saia fora do domínio.

3 – O ALU realiza a soma de A com B e a soma negada de A com B. Ou seja, quando OP = 0, o ALU realiza a operação A+B. Quando OP = 1, o ALU realiza a operação -A + (-B).

O valor do sinal OP associado a cada operação é dado por:

Operação	**Valor**
ldr / str	00
sub	01
nor	10
and	11

Tabela 1 – valor do sinal OP para cada instrução

3.2 Codificação das instruções

1 –

	8	7	6	5	4	3	2	1	0
b rn				-	-	-	-	rn1	rn0
bzs label				off5	off4	off3	off2	off1	off0
ldr rd, [rn, #imm2]				rd1	rd0	rn1	rn0	imm1	imm0
mov rd, #imm4				rd1	rd0	imm3	imm2	imm1	imm0
nor rd, rn, rm				rd1	rd0	rn1	rn0	rm1	rm0
sub rd, rn, #imm2				rd1	rd0	rn1	rn0	imm1	imm0
str rd, [rn, rm]				rd1	rd0	rn1	rn0	rm1	rm0
tst rn, rm				-	-	rn1	rn0	rm1	rm0
	opcode								

Tabela 2 – Mapa de codificação das instruções

2 – De acordo com o funcionamento da ALU e com os valores atribuídos a cada operação na resposta 3.1.3, observámos que havendo instruções com o mesmo valor de operação (como por exemplo, a instrução ldr e str) o valor atribuido ao bit mais à esquerda teria que ser diferente em ambos para ser possível realizar cada operação. Assim, os dois bits da direita são dados como o valor da operação atribuidos na resposta de cima e os valores do bit mais à esquerda são atribuídos

consoante a nossa escolha tendo sempre em atenção os casos em que os dois bits da direita são iguais e por isso o bit na esquerda numa das instruções tem que ser 0 e noutra tem que ser 1.

	8	7	6
b rn	1	1	1
bzs label	0	0	1
ldr rd, [rn, #imm2]	1	0	0
mov rd, #imm4	0	1	0
nor rd, rn, rm	1	1	0
sub rd, rn, #imm2	1	0	1
str rd, [rn, rm]	0	0	0
tst rn, rm	0	1	1

Tabela 3 – Tabela com os OP codes das instruções

3.3 Projeto do descodificador de instruções

1 –

Instrução	**Endereço**				**Palavra**								
b rn	1	1	1	-	-	1	1	0	0	-	0	1	-
bzs label	0	0	1	0	-	1	1	1	1	00	0	0	0
	0	0	1	1	-	1	1	1	1	00	0	0	1
ldr rd, [rn, #imm2]	1	0	0	-	00	0	1	0	1	01	1	0	0
mov rd, #imm4	0	1	0	-	10	1	1	0	1	10	0	0	0
nor rd, rn, rm	1	1	0	-	01	1	1	1	0	-	1	0	0
sub rd, rn, #imm2	1	0	1	-	01	1	1	1	1	11	0	0	0
str rd, [rn, rm]	0	0	0	-	-	1	0	0	0	-	1	0	0
tst rn, rm	0	1	1	-	-	1	1	1	1	-	0	0	0
	opcode			**Z**	**SD**	**nRD**	**nWR**	**EF**	**SC**	**SE**	**ER**	**SI**	**SO**

2 –

Valor dos endereços	Notação hexadecimal
14 a 15	0xC2
2	0x1E0
3	0x1E1
8 a 9	0xAC
4 a 5	0x5B0
12 a 13	0x3C4
10 a 11	0x3F8
0 a 1	0x84
6 a 7	0xF0

3 – ROM – (2^n) * m [n: endereços, m: dados]

ROM: 2^4 * 11 = 176 bits

3.4 Codificação de programas em linguagem máquina

1 – O programa carrega em r1 o valor do endereço de memória r0 + 0. Faz um NOR entre r0 e r0 e armazena o valor em r3. Compara r1 com r3. Caso bzs verifique que o valor do registo é 1 ele salta para a label skip, caso contrário, carrega em r2 o valor que reside no endereço de memória r0 + 1. Armazena o NOR entre r1 e r1 em r1, faz o mesmo para r2 e de seguida armazena em r1 o NOR entre r1 e r2. Na label skip atribui o valor 8 ao r3, guarda o valor de r1 no endereço de memória r3 + r0, atribui o valor 0xC ao r2 e salta para r2. Assim, concluimos que o programa tem a funcionalidade de testar a arquitetura.

2 –

Instrução	Código máquina
mov r0, #0	010 00 0000 (0x80)
ldr r1, [r0, #0]	100 01 00 00 (0x110)
nor r3, r0, r0	110 11 00 00 (0x1B0)
tst r1, r3	011 00 01 11 (0xC7)
bzs skip	001 000 101 (0x45)
ldr r2, [r0, #1]	100 10 00 01 (0x121)
nor r1, r1, r1	110 01 01 01 (0x195)
nor r2, r2, r2	110 10 10 10 (0x1AA)
nor r1, r1, r2	110 01 01 10 (0x196)
mov r3, #8	010 11 1000 (0xB8)
str r1, [r3, r0]	000 01 11 00 (0x1C)
mov r2, #0xC	010 10 1100 (0xAC)
b r2	111 0000 10 (0x1C2)

Tabela 4 – Tradução do código para código máquina

3 – (a) – Não é possível fazer essa alteração visto que, nesta arquitetura, a ALU é usada pela instrução str para fazer a soma entre os registos rn e rm. Sendo assim, a ALU não ia ser usada para fazer a operação (soma).

(b) i) Como vantagem, o endereçamento passaria a ser em valores mais baixos, e desse modo permite aceder aos valores do registo mais rapidamente. Uma desvantagem seria o facto de o str com apenas um registo não permite chegar a todos os endereços dessa RAM, visto que o valor de um só registo vai ser menor do que a soma desse registo com outro.

ii) Quanto à densidade do código, esta aumentaria porque a parte alta da RAM estaria inalcançável. Para ser possível usá-la seria preciso mais troço de código. Porém, é importante referir que se isto não for feito, existiria desaproveitamento.

iii) Quanto ao impacto no desenho da microarquitetura, a ALU teria de ser redesenhada pois já não iria ser feita uma soma e por isso o percurso da instrução str tinha que ser diferente.

2. Conclusão

Com este trabalho conseguimos consolidar os nossos conhecimentos acerca do estudo do funcionamento de um processador, bem como aprender a trabalhar com o logisim.