Arquitetura de Computadores

2º Trabalho Prático

Estudo do funcionamento de um processador

49470 Ana Carolina Pereira
50562 Umera Aktar

Licenciatura em Engenharia Informática e de Computadores
Semestre de Verão 2022/2023

21/4/2023

Índice

1. Análise da microarquitetura

1.1. Tipo de microarquitetura

A afirmação é verdadeira uma vez que a Figura 1 apresenta o diagrama lógico da microarquitetura do processador – constítuida por *Control Unit*, ALU, *Register File*, *Instruction Memory* e *Data Memory*.
A arquitetura de Harvard é caracterizada por ser uma arquitetura com ciclo único e por ter uma memória para dados e programa separadas.

1.2. Bloco Ext

O bloco Ext tem como funcionalidade extender o sinal de um valor imediato.

2. Codificação das instruções

2.1. Mapa de codificação

O mapa de codificação realizado encontra-se na Figura 1.

Instrução	8	7	6	5	4	3	2	1	0
add rd, rn	rd			rn			opcode		
b rn	rn			-	-	-	opcode		
bne label	label						opcode		
cmp rn, rm	rn			rm			opcode		
ldr rd, [rn]	rd			rn			opcode		
mov rd, #imm3	rd			#imm3			opcode		
push rn	-	-	-	rn			opcode		
str rd, [rn]	rd			rn			opcode		

2.2. Código das instruções

Na tabela abaixo encontram-se os valores dos opcodes

Instrução	opcode		
add rd, rn	0	0	0
ldr rd, [rn]	0	0	1
cmp rn, rm	0	1	0
push rn	0	1	1
b rn	1	0	0
bne label	1	0	1
mov rd, #imm3	1	1	0
str rd, [rn]	1	1	1

Como sabemos, as instruções **add** e **ldr** internamente realizam somas e por isso os seus dois primeiros bits serão os mesmos, neste caso 00.

A instrução **cmp** internamente realiza uma subtração de forma a que se a flag Z der 0, esta ativar.

A instrução **str** envia o registo rd para o *data bus* (vindo do banco de registos) e o registo rn para a ALU, sendo este o endereço onde o endereço onde o registo rd vai posteriormente ser escrito em memória. Por fim, as instruções **mov**, **bne** e **b** não precisam de realizar operações na ALU. No **mov**, foram atribuidos aos dois primeiros bits os valores 11 (respetivamente) uma vez que queremos que o segundo operando seja um valor imediato. Já para as instruções **b** e **bne** foi atribuido 10 aos dois primeiros bits.

3. Descodificador de instruções

3.1. Sinais do descodificador

A tabela abaixo apresentada apresenta os sinais de cada instrução.

	nRD	nWR	EP	ER	SD	SE	SS	SI	SO
add	1	1	1	1	00	-		0	1
ldr	0	1	0	1	00	0		0	1
cmp	1	1	1	0	01	-		0	1
push	1	1	0		-	-		0	1
b	1	1	0	-	-	-		0	0
bne	1	1	0	-	-	-		1	1
mov	1	1	0	1	11	1		0	1
str	1	0	0	0	11	-		0	1

Desta forma :

- o sinal **nRD** está ativo nas instruções que não precisam de ler da memória (porque é negado);
- o sinal **nWR** está ativo nas instruções que não precisam de escrever da memória (porque é negado);
- o sinal **EP** está ativo nas instruções que atualizam a flag Z da ALU;
- o sinal **ER** está ativo nas instruções em que é preciso armazenar valores no banco de registos;
- o sinal **SE** está ativo em instruções que precisam de extender o valor de uma imediata;
- o sinal **SI** está ativo em instruções que precisam de alterar o valor do PC;
- o sinal **SO** está ativo em instruções que não alteram o PC diretamente.

3.2. Implementação baseada em ROM

	A3	A2	A1	A0
add	0	0	0	1
ldr	0	0	1	0
cmp	0	1	0	1
push	0	1	1	0
b	1	0	0	0
bne	1	0	1	0
mov	1	1	0	0
str	1	1	1	0

3.3. Capacidade de memória da ROM

Capacidade da ROM = 2^4 * 10 = 160 bits = 20 bytes

4 - porque o instruction decode tem uma entrada opcode de 3 bits e uma entrada Z (flag)
10 – porque temos 9 sinais/saídas, mas o sinal SD tem 2 bits

4. Codificação de programas em linguagem máquina

4.1. Funcionaliade do programa

```
        mov   r0, #0
        mov   r1, #0
        mov   r2, #4
loop:
        ldr   r3, [r0]
        add   r1, r3
        mov   r4, #1
        add   r0, r4
        cmp   r0, r2
        bne   loop
        str   r1, [r2]
        mov   r5, #6
        add   r5, r5
        b     r5
```

O troço de código apresentado realiza a soma dos elementos de um array. Começa-se por iniciar os registos r0, r1 e r2 com o valor zero. De seguida entra-se dentro do loop em que primeiro é carregado o valor do endereço (primeiro elemento do array) contido em r0 no registo r3. Adiciona o valor de r3 ao valor de r1, que corresponde à soma parcial dos elementos do array até ao momento. Guarda-se no registo r4 o valor 1 para incrementar o valor de r0 no loop seguinte e adiciona-se o valor de r4 ao valor de r0, que é o endereço do próximo elemento do array a ser somado. Posto isto, compara-se o valor de r0 com o valor de r2, que é o tamanho do array e se r0 for menor que r2, o programa volta para o label "loop" caso contrário guarda-se o valor contido em r1 no endereço contido em r2 (este endereço é o último elemento do array, que agora contém a soma total dos elementos do mesmo). De seguida guarda-se em r5 o valor 6 e adiciona-se o valor de r5 a si mesmo - o que corresponde a multiplicar r5 por 2. Por fim, realiza-se um salto incondicional para o endereço contido em r5.

4.1. Código máquina

Instrução	8	7	6	5	4	3	2	1	0	endereço
mov r0, #0	0	0	0	0	0	0	1	1	0	0x00
mov r1, #0	0	0	1	0	0	0	1	1	0	0x01
mov r2, #4	1	0	0	1	0	0	1	1	0	0x02
ldr r3, [r0]	0	1	1	0	0	0	0	0	1	0x03
add r1, r3	0	0	1	0	1	1	0	0	0	0x04
mov r4, #1	0	0	1	0	0	1	1	1	0	0x05
add r0, r4	0	0	0	1	0	0	0	0	0	0x06
cmp r0, r2	0	0	0	0	1	0	0	1	0	0x07
bne loop	0	0	0	0	0	0	1	0	1	0x08
str r1, [r2]	0	0	1	0	1	0	1	1	1	0x09
mov r5, #6	1	1	0	1	1	0	1	1	0	0x0A
add r5, r5	1	0	1	1	0	1	0	0	0	0x0B
b r5	1	0	1	-	-	-	0	1	1	0x0C

4. Conclusões

Com este trabalho foi possivel consolidar os conhecimentos sobre microarquiteturas e funcionamento de processadores.