Parte III – Exemplo de Aplicação

Neste capítulo descreve-se a implementação de um processador básico com as seguintes características:

- Capacidade de endereçamento de memória de dados até 16x8
- Capacidade de endereçamento de memória de código até 32x8
- Dois registos genéricos de 8 bits
- É capaz de interpretar e executar o conjunto de instruções da tabela 4.

Tabela 4 - Conjunto de instruções do processador básico

Instrução	**Sintaxe**	**Função**
LD Rx, direct	Rx ← M[direct]	Move conteúdo da posição *direct* da memória RAM para o registo Rx
ST direct, Rx	M[direct] ← Rx	Move conteúdo do registo Rx para a posição *direct* da memória RAM
MOV Rx, #const	Rx ← #const	Move constante *const* para o registo Rx
Add Rx, Ry	Rx ← Rx + Ry	Adiciona Rx com Ry e coloca o resultado em Rx
Sub Rx, Ry	Rx ← Rx - Ry	Subtrai Ry a Rx e coloca o resultado em Rx
JMP addr	PC ← addr1	O registo PC fica com o endereço *addr1*
JZ addr	If (Z = 1) PC ← addr	O registo PC fica com o endereço *addr* se o conteúdo da ALU for 0
JP addr	If (P = 1) PC ← addr	O registo PC fica com o endereço *addr* se o conteúdo da ALU for positivo

1.1 Conjunto de Instruções e Programação

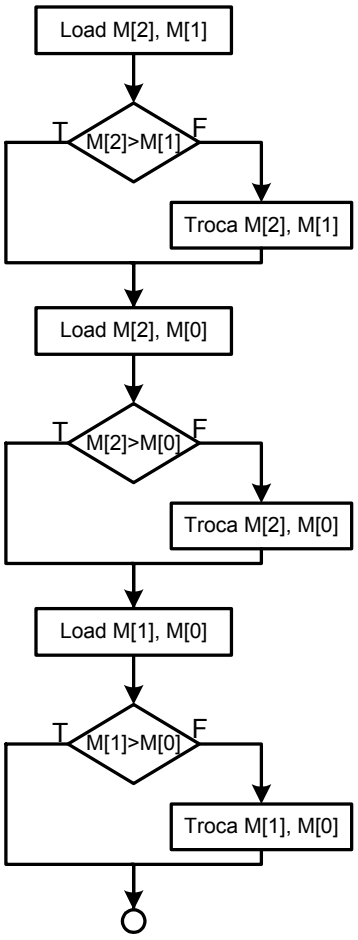
O conjunto tem três primeiras instruções de transferência de dados, duas delas para transferir dados entre a memória e os registos e a terceira para carregar constantes em registos, duas de manipulação de dados, uma soma e uma subtracção, e três de controlo.

Antes do projecto da arquitectura, vamos considerar um exemplo de um programa com o conjunto de instruções. O programa deverá ordenar por ordem crescente os valores que se encontram nas posições 0, 1 e 2 da memória RAM. Considera-se que os valores a ordenar são positivos e representados com o código dos complementos.

A solução considerada irá começar pelo valor na posição 2 e compara-o com as outras duas. Sempre que for menor, troca-se os conteúdos. O processo repete-se com os conteúdos das posições 0 e 1. A comparação é feita à custa da subtracção bastando depois verificar o sinal do resultado (ver programa 1).

Programa 1 – Programa de ordenação de números

```
0 – LD R0, 2
1 – LD R1, 1
2 – SUB R0, R1
3 – JP 7
4 – LD R0, 2
5 – ST 2, R1
6 – ST 1, R0
7 – LD R0, 2
8 – LD R1, 0
9 – SUB R0, R1
10 – JP 14
11 – LD R0, 2
12 – ST 2, R1
13 – ST 0, R0
14 - LD R0, 1
15 – LD R1, 0
16 – SUB R0, R1
17 – JP 21
18 – LD R0, 1
19 – ST 1, R1
20 – ST 0, R0
21 – JMP 21
```



Exercício – Multiplique os conteúdos positivos dos endereços 2 e 3 da memória RAM e coloque o resultado no endereço 1 da mesma RAM.

1.2 Codificação das Instruções

Tendo em conta o conjunto de instruções do processador e os requisitos de endereçamento de memória, podemos realizar o passo de codificação das instruções.

Uma vez que o conjunto tem oito instruções, podem-se considerar três bits para o *opcode*. Na atribuição dos códigos, vamos ter o cuidado de nas operações aritméticas considerar garantir que os códigos apenas diferem num bit. Assim, na implementação, o sinal respectivo pode ser ligado directamente à entrada de controlo de operação da ALU.

Tendo em conta as necessidades de endereçamento, conclui-se que o endereço *direct* terá de ter 4 bits e o *addr* 5 bits. O tamanho da constante não está especificado, mas vamos usar 4 bits para que a codificação total não ultrapasse os 8 bits.

Com estas considerações, obteve-se a codificação final da tabela 5.

Tabela 5 – Codificação do conjunto de instruções do processador básico

Instrução	Opcode	Operandos		
LD Rx, direct0	0 0 0	Rx	direct0	
	7 5	4	3	0
ST direct0, Rx	0 0 1	Rx	direct0	
	7 5	4	3	0
MOV Rx, #const	0 1 0	Rx	const	
	7 5	4	3	0
Add Rx, Ry	1 0 0	Rx	Ry	- - -
	7 5	4	3 2	0
Sub Rx, Ry	1 0 1	Rx	Ry	- - -
	7 5	4	3 2	0
JMP addr	0 1 1	addr1		
	7 5	4		0
JZ addr	1 1 0	addr1		
	7 5	4		0
JP addr	1 1 1	addr1		
	7 5	4		0

No processo de codificação houve o cuidado de alinhar os parâmetros para simplificar o descodificador de instruções.

1.3 Arquitectura do processador

A arquitectura do processador básico baseia-se nas arquitecturas introduzidas nos capítulos anteriores. A unidade de processamento tem um banco de dois registos e uma ALU com duas operações (adição e subtracção) (ver figura 18).

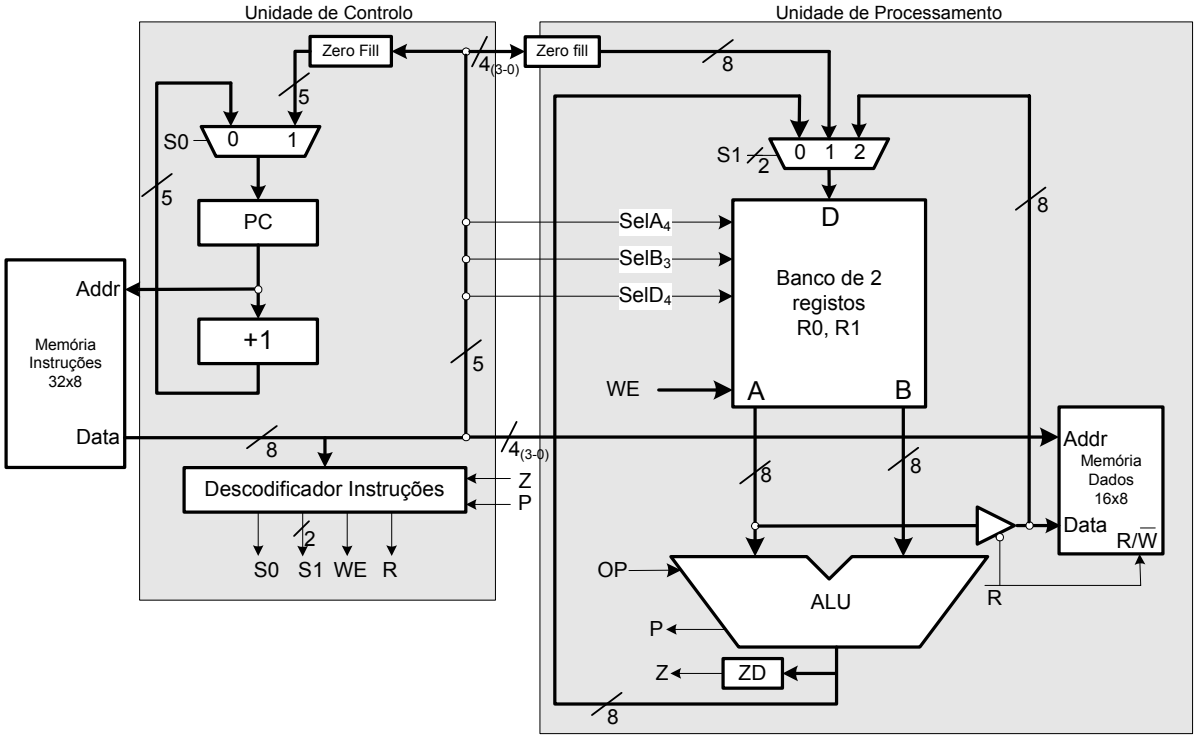


Figura 18 – Arquitectura do processador básico

Atente-se na interacção entre a unidade de controlo e a de processamento. O endereço da memória de dados vem directamente da saída de dados da memória de instruções consistindo nos 4 bits de menor peso correspondentes ao parâmetro endereço da instrução. Os selectores do banco de registos também vêm directamente da saída de dados da memória de instruções. Daí a vantagem em alinhar as referências aos registos na instrução. O *multiplexer* à entrada do banco de registos escolhe a origem dos dados a escrever nos registos, nomeadamente, da memória, da ALU ou uma constante vinda da instrução.

O descodificador de instruções recebe o opcode e os bits de estado da ALU e gera a palavra de controlo formada pelos sinais S0, S1, WE e R. Na sua implementação usou-se uma ROM (ver figura 19).

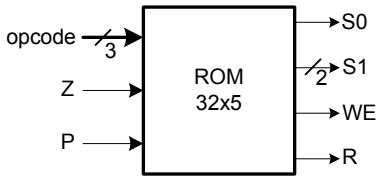


Figura 18 – Especificação da ROM usada na implementação do descodificador de instruções

O conteúdo da ROM está de acordo com a execução das instruções (ver tabela 6).

Tabela 6 – Conteúdo da ROM de descodificação de instruções

Inputs			**Outputs**			
opcode	**Z**	**P**	**S0**	**S1**	**R**	**WE**
0 0 0	–	–	0	1 0	1	1
0 0 1	–	–	0	– –	0	0
0 1 0	–	–	0	0 1	1	1
0 1 1	–	–	1	– –	1	0
1 0 0	–	–	0	0 0	1	1
1 0 1	–	–	0	0 0	1	1
1 1 0	0	–	0	– –	1	0
1 1 0	1	–	1	– –	1	0
1 1 1	–	0	0	– –	1	0
1 1 1	–	1	1	– –	1	0

‘–’ → indiferenças