Capítulo 1 | Introdução

Com este trabalho, propusemo-nos a desenhar a arquitetura de um processador $\mu RISC$ com funcionamento multi-ciclo. Dividimos a arquitetura em unidades funcionais de modo a simplificar a implementação do processador.

O processador $\mu RISC$ consiste num processador com um número reduzido de instruções simples, nosso caso 42 instruções, distribuídas entre operações de salto, aritméticas, lógicas, deslocamento, "load"/"store" e uso de constantes.

Capítulo 2 | Unidade lógico-aritmética (ALU)

Breve descrição da **ALU**: Desenhou-se a **ALU** com três unidades a funcionarem em paralelo, abaixo descritas com maior detalhe.

O resultado produzido por estas unidades é introduzido num "multiplexer" que escolhe de acordo com sinais provenientes da unidade de descodificação qual o resultado e "Flags" a colocar à saída da **ALU**.

2.1 Unidade Aritmética

A unidade Aritmética é responsável pelas operações apresentadas na tabela 2.1.

OP	Operação	Mnemónica	Flags actualizadas
00000	$C = A + B$	add c, a, b	S,C,Z,V
00001	$C = A + B + 1$	addinc c, a, b	S,C,Z,V
00011	$C = A + 1$	inca c, a	S,C,Z,V
00100	$C = A - B - 1$	subdec c, a, b	S,C,Z,V
00101	$C = A - B$	sub c, a, b	S,C,Z,V
00110	$C = A - 1$	deca c, a	S,C,Z,V

Tabela 2.1: Operações Aritméticas

A unidade aritmética começa por analisar qual a operação a executar de acordo com os dados vindos da unidade de descodificação e em seguida começa por calcular o segundo membro da operação $C = A + operB$ em que

$$operB = \begin{cases} B & : OP = 00000 \\ B + 1 & : OP = 00001 \\ 1 & : OP = 00011 \\ -B - 1 & : OP = 00100 \\ -B & : OP = 00101 \\ -1 & : OP = 00110 \end{cases}$$

De seguida calcula $C = A + operB$ e as "Flags" correspondentes com base na análise do resultado e dos operandos.

2.2 Unidade Lógica

A unidade Lógica é responsável pelas operações apresentadas na tabela 2.2.

OP	Operação	Mnemónica	Flags actualizadas
10000	$C = 0$	zeros c	
10001	$C = A\&B$	and c, a, b	S,Z
10010	$C = !A\&B$	andnota c, a, b	S,Z
10011	$C = B$	passb c, b	
10100	$C = A\&!B$	andnotb c, a, b	S,Z
10101	$C = A$	passa c, a	S,Z
10110	$C = A \oplus B$	xor c, a, b	S,Z
10111	$C = A\|B$	or c, a, b	S,Z
11000	$C = !A\&!B$	nor c, a, b	S,Z
11001	$C = !(A \oplus B)$	xnor c, a, b	S,Z
11010	$C = !A$	passnota c, a	S,Z
11011	$C = !A\|B$	ornota c, a, b	S,Z
11100	$C = !B$	passnotb c, b	S,Z
11101	$C = !A\|!B$	nand c, a, b	S,Z
11111	$C = 1$	ones c	

Tabela 2.2: Operações Lógicas

2.3 Unidade de Deslocamentos

A unidade de Deslocamentos é responsável pelas operações apresentadas na tabela 2.3.

OP	Operação	Mnemónica	Flags actualizadas
01000	$C = Shift\ Lógico\ Esq.(A)$	lsl c, a	S,C,Z
01001	$C = Shift\ Aritmético\ Dir.(A)$	asr c, a	S,C,Z

Tabela 2.3: Operações de Deslocamento

No caso do "shift" lógico a saída resulta do deslocamento do sinal de entrada uma posição e preenchimento do "bit0" com 0.

No caso do "shift" aritmético a saída resulta do deslocamento do sinal de entrada uma posição e preenchimento do "bit15" com o "bit15" da entrada (extensão de sinal).

2.4 Cálculo das flags

Para o cálculo das *flags* seguimos uma técnica semelhante nas 3 sub-unidades dentro da **ALU**.

2.4.1 Flag de Sinal (S)

A *flag* de sinal é calculada em todas as sub-unidades e é feita sempre do mesmo modo, avaliando o "bit15" do resultado. Ou seja, $S = Resultado(15)$.

Isto pode se fazer deste modo por estarmos a utiliza números com sinal em complemento para 2.

2.4.2 Flag de Carry (C)

A *flag* de *Carry* é apenas calculada nas sub-unidades Aritmética e de Deslocamentos, na unidade lógica é passado o valor zero(0) e depois ignorado na **Unidade de Controlo de Saltos e Flags**.

No caso da unidade **Aritmética**, as operações são feitas com 1 *bit* extra, ou seja, o resultado é gerado com 17 bits de onde são utilizados os 16 menos significativos para o verdadeiro resultado e 17º bit para calcular a *flag* de *Carry*.

Resultado é internamente representado com 17 bits, $bit(16\ downto\ 0)$.

$$\begin{aligned} flagC &= bit(16) \\ Resultado &= bit(15\ downto\ 0) \end{aligned}$$

No caso da unidade de **Deslocamentos** a *flag* de *Carry* é o bit mais significativo do valor de entrada no caso da operação $lsl\ c,\ a$ e zero no caso da operação $asr\ c,\ a$.

$$flagC = \begin{cases} a(15) & : \ lsl\ c,\ a \\ 0 & : \ asr\ c,\ a \end{cases}$$

2.4.3 Flag de Zero (Z)

A *flag* de zero é calculada em todas as sub-unidades e é feita sempre do mesmo modo, é efetuado a operação de **nor** com todos os bits do resultado.

$$Resultado = c$$

$$flagZ = \overline{(c15 + c14 + c13 + c12 + c11 + c10 + c9 + c8 + c7 + c6 + c5 + c4 + c3 + \overline{+c2 + c1 + c0)}}$$

2.4.4 Flag de Overflow (V)

A *flag* de *Overflow* é apenas calculada na sub-unidade **aritmética**, nas restantes é passado o valor zero(0) e depois ignorado na **Unidade de Controlo de Saltos e Flags**.

Para calcular se existe *Overflow* analisam-se os bits de sinal dos operandos e o do resultado, ou seja, numa operação de adição ou subtração se os 2 operandos tiverem o mesmo bit de sinal então o resultado terá que preservar esse mesmo bit de sinal.

$$\begin{aligned} Resultado = c &= a + b \\ flagV &= \overline{(a15 \oplus b15)} \cdot (a15 \oplus c15) \end{aligned}$$

Capítulo 3 | Unidade de controlo de saltos e flags

Breve descrição da unidade: Desenhou-se a unidade de modo a controlar o próximo endereço a enviar ao *Program Counter* (**PC**).
Esta unidade guarda os valores das *flags* provenientes da **ALU** em registos e depois usa esses registos para calcular as condições de salto.

Juntámos os registos das *flags* com a unidade de controlo de saltos de modo a que consoante a condição de salto vinda do **Decoder** se pudesse calcular se se deveria executar um salto ou se permitíamos que o valor do **PC** fosse incrementado normalmente.

O cálculo do próximo endereço do **PC** é feito em 4 fases.

1. Cálculo da condição de salto
2. Cálculo do *offset* para o caso de saltos no Formato I ou do Formato II
3. Determinar se o salto é para um *offset* ou para um registo
4. Determinar o próximo endereço do **PC** com base no tipo de *jump* (condicional ou incondicional) e a condição

$$\text{Condição} = \begin{cases} 1 & : \ COND = 0000 \\ flagV & : \ COND = 0011 \\ flagS & : \ COND = 0100 \\ flagZ & : \ COND = 0101 \\ flagC & : \ COND = 0110 \\ flagS + flagZ & : \ COND = 0111 \\ 0 & : \ others \end{cases}$$

$$\text{Offset} = \begin{cases} Destino(11)\&Destino(11)\&Destino(11)\&Destino(11) \\ \quad \&Destino(11 \ downto \ 0) & : \ OP = 10 \\ Destino(7)\&Destino(7)\&Destino(7)\&Destino(7)\&Destino(7) \\ \quad \&Destino(7)\&Destino(7)\&Destino(7)\&Destino(7 \ downto \ 0) & : \ others \end{cases}$$

$$\text{Jump Address} = \begin{cases} RB & : \ OP = 11 \\ PC + 1 + Offset & : \ others \end{cases}$$

$$\text{Próximo PC} = \begin{cases} Jump\ Address & : & enable\_jump = 1 \ \cdot \ (\text{Condição} \oplus OP(0)) = 1 \\ Jump\ Address & : & enable\_jump = 1 \ \cdot \ OP(1) = 1 \\ PC + 1 & : & others \end{cases}$$

Capítulo 4 | Unidade de Constantes

Breve descrição da unidade: A unidade de Constantes é responsável pelas operações apresentadas na tabela 4.1.

Formato	Operação	Mnemónica
I	$C = Constante$	loadlit c, Const
II	$C = Const8 \mid (C\&0xff00)$	lcl c, Const8
II	$C = (Const8 << 8) \mid (C\&0x00ff)$	lch c, Const8

Tabela 4.1: Operações com Constantes

Optámos por separar estas operações das restantes da **ALU** de modo a facilitar a descodificação das instruções por parte do *decoder*.

Capítulo 5 | Conclusão

No decorrer do trabalho deparámo-nos com algumas dificuldades, nomeadamente o carregamento dos dados de um ficheiro de entrada para a memória do processador.

Para ultrapassarmos esta dificuldade recorremos à utilização de uma *impure function* tal como descrito na secção da **Unidade de Armazenamento**.

Outra unidade que nos apresentou dificuldades na sua execução foi a **Unidade de Descodificação** que pela sua complexidade demorou mais tempo a ser executada.

Apesar das dificuldades que nos apresentou, a **Unidade de Descodificação** poderia ser melhorada, fazendo com que a descodificação feita na **Unidade de Saltos** se juntasse ao resto da descodificação.

No geral, consideramos que o trabalho prosseguiu de acordo com o planeado.