Instituto Superior Técnico

MEEC

2º Semestre 2014/2015

Arquitecturas Avançadas de Computadores

1º Projecto

Simulacao processador μRISC com funcionamento multi-ciclo

João Baúto **Nº 72856**
João Severino **Nº 73608**

Docente: Prof.Leonel Sousa

28 de Março de 2015

Conteúdo

1. Introdução

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Aliquam interdum libero a eros posuere porta. Praesent a risus id enim consectetur facilisis. Nulla ut euismod tortor. Cras in ipsum tempus, vestibulum lorem nec, posuere odio. Proin mollis, lectus non lacinia cursus, odio lacus gravida enim, non rutrum turpis dolor eu nunc. Praesent cursus semper lorem, commodo elementum nisi ullamcorper et. Donec eu ligula diam. Ut nunc ante, viverra eget posuere ut, accumsan in diam. Aliquam erat volutpat. Lorem ipsum dolor sit amet, consectetur adipiscing elit. Lorem ipsum dolor sit amet, consectetur adipiscing elit. Suspendisse a augue est. Quisque ut neque lorem.

Fusce id neque at urna viverra faucibus a ac enim. Aenean porttitor ex et vehicula fermentum. Praesent sit amet vehicula ex, vitae mattis quam. Praesent sit amet magna a sem suscipit cursus. Ut blandit finibus elit sed interdum. Praesent dignissim nulla ut lacus dictum congue. Quisque maximus nibh nunc, vitae consequat quam eleifend ut. Etiam vel ultrices lectus, sit amet imperdiet eros. Class aptent taciti sociosqu ad litora torquent per conubia nostra, per inceptos himenaeos. Proin consequat justo ut massa imperdiet fermentum. Integer eget velit felis. Interdum et malesuada fames ac ante ipsum primis in faucibus. Nullam imperdiet leo.

2. Arquitectura do μRISC

2.1 Unidade de descodificação - Decoder

Por decisão própria, na unidade de descodificação foi efectuado o máximo possível de descodificação de operações, selectores de *multiplexers* e unidades funcionais. Desta forma é nos possível generalizar as restantes unidades funcionais centralizando toda a descodificação numa só unidade. Uma consequência desta metodologia é o aumento da complexidade da unidade e o número de sinais de *output*.

2.2 Unidade de Armazenamento - Memória RAM/ROM partilhada

De forma a facilitar o endereçamento da memória optou-se por uma unidade de armazenamento partilhado. No início da simulação esta é inicializada a partir de um ficheiro .txt graças à utilização de uma *impure function* que introduz as instruções a primeira posição de memória incrementando o endereço para a seguinte instrução. Esta unidade apresenta três entradas, Din para o armazenamento de dados através para instrução *store*, Addr_Instr que indica o endereço da próxima instrução a ser enviada para o *Decoder* e Addr_Dados que endereça a posição para onde será feito uma instrução de *load*. Como saídas tem-se Dout_Dados proveniente da instrução *load* e Instr que indica a próxima instrução.

As vantagens deste tipo de memória é a facilidade de endereçamento uma vez que não é necessário fornecer um offset para o caso em que é necessário aceder a um array *por exemplo*. Como desvantagem tem se o facto de o programador necessitar de uma extra atenção às posições de memória onde guarda dos dados podendo substituir futuras instruções.

Com duas memórias independentes seria possível evitar este problema caso ambas as memórias fossem inicializadas com os mesmos dados (instruções e *arrays*).

2.3 Unidade lógico-aritmética - 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 selecciona de acordo com sinais provenientes da unidade de descodificação qual o resultado e as *Flags* a colocar à saída da ALU.

2.3.1 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.

2.3.2 Unidade Lógica

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

OP	Operação	Mnemónica	Flags actualizadas
10000	$C = 0$	zeros c	Nenhuma
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	Nenhuma
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	Nenhuma

Tabela 2.2: Operações de Deslocamento

2.3.3 Unidade de Deslocamentos

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

OP	Operação	Mnemónica	Flags actualizadas
01000	$C = ShiftLgicoEsq.(A)$	lsl c, a	S,C,Z
01001	$C = ShiftAritmticoDir.(A)$	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.

2.4 Unidade de Constantes

A unidade de Constantes é responsável pelas operações apresentadas na tabela 2.4.

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

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 2.4: Operações com Constantes

2.5 Unidade de controlo de saltos e *flags*

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

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

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

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

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