



ARQUITECTURAS AVANÇADAS DE COMPUTADORES

LABORATÓRIO I

Simulação de um microprocessador μ RISC com funcionamento multi-ciclo

Gonçalo Ribeiro
73294

Miguel Costa
73359

Rafael Gonçalves
73786

27 de Março de 2015

Conteúdo

1	Introdução	2
2	Arquitectura	2
2.1	IF	2
2.1.1	ROM	2
2.1.2	PC	2
2.2	IDRF	2
2.2.1	RF	3
2.3	EX	3
2.3.1	ALU	3
2.3.2	RAM	3
2.3.3	Flag Tester	4
2.4	WB	4
2.5	Máquina de Estados	4
3	Testes e Simulações	4
3.1	Fibonacci	4
4	Conclusão	4

1 Introdução

Nesta actividade laboratorial pretende-se desenvolver um processador μ RISC com descrição em VHDL. Pretende-se que este processador com arquitectura RISC de 16 bits e funcionamento multi-ciclo execute 42 instruções diferentes, demorando 4 ciclos para completar cada uma. Ao primeiro ciclo (Instruction Fetch - IF), a próxima instrução a ser executada é carregada da memória e armazenada no registo de instruções. No segundo ciclo (Instruction Decode - ID) a instrução anteriormente carregada é decodificada e os operandos são lidos do Register File (RF). O terceiro ciclo (Execution - EX) consiste na execução da instrução e cálculo das condições dos resultados. Por fim, no quarto ciclo (Write Back - WB) os resultados são escritos no RF.

Tal como já foi referido, é um processador de 16 bits que contém 8 registos de uso geral, cada um com 16 bits de largura. Existem 42 instruções possíveis de executar, sendo estas compostas por até 3 operandos. Quanto à memória esta é do tipo big endian, sendo endereçável ao nível da palavra.

2 Arquitectura

2.1 IF

2.1.1 ROM

2.1.2 PC

2.2 IDRF

Neste módulo são gerados vários sinais de controlo com base na instrução obtida do IF. Para além do mais, com base nessa instrução é também feita a leitura assíncrona dos registos envolvidos na operação (operandos).

Entre os sinais de controlo gerados neste bloco encontram-se:

- A operação da ALU, que corresponde a uma cadeia de bits da instrução, salvo no caso de se tratar de um salto ou de carregamento de uma constante (nesse caso é passada a operação necessária);
- Os sinais de controlo dos multiplexers das duas entradas da ALU, que variam consoante a operação;
- A condição e o tipo de salto, bem como um sinal a indicar se a instrução é, de facto um salto, ou se estes valores devem ser ignorados;
- Um sinal que só permite a alteração das *flags* quando a instrução é executada ao nível da ALU (por oposição a uma leitura ou escrita da memória);
- Sinais relativos ao carregamento de constantes para a parte *high* ou *low* de um registo, ou para o carregamento de imediatos;
- Um sinal de controlo para a escrita na memória;
- Sinais de controlo a ser passados para o Write Back com o endereço do registo de destino do resultado, bem como os *enables* relevantes.

2.2.1 RF

O bloco de registos contém 8 registos de 16 bits cada, com dois portos de leitura assíncrona e um porto de escrita síncrona.

Os endereços de leitura provêm dos mesmos bits de instrução, excepto quando é feito o carregamento de um só byte para as partes *high* ou *low* da palavra. Nesse caso, o endereço do porto de leitura A é substituído pelo do registo de leitura e escrita.

O endereço de escrita vem do Write Back, sendo que o *enable* de escrita deste bloco está ligado ao *enable* do Write Back, e não ao do módulo IDRF.

2.3 EX

Neste módulo são executadas as instruções e calculadas as condições de resultados. Podemos então considerar 3 grandes componentes que funcionam neste ciclo: ALU, Flag Tester e RAM.

Tal como é visível na Figura 1, podemos ainda considerar lógica adicional, correspondente ao *load* de constantes. Como o processador permite o carregamento de constantes para a parte *high* ou parte *low*, é feita aqui essa concatenação, sendo que depois é feita uma multiplexagem para os restantes casos de carregamento de constantes (que provêm do *Signal Extender* em IDRF).

2.3.1 ALU

O componente da ALU é responsável por receber 2 operandos (*A* e *B*), efectuar uma dada operação sobre estes (aritmética, deslocamento lógico ou lógica), dando um resultado *C*.

Este componente é divisível em 3 subcomponentes: componente aritmética, lógica e shifts. É neste componente que são geradas as *flags*, consoante a operação a executar. Existem então 4 *flags*: *signal* (*S*), *carry* (*C*), *zero* (*Z*) e *overflow* (*V*). Mais à frente são descritas as funcionalidades destas e quais as suas utilidades.

O primeiro sub-componente é responsável por executar qualquer função aritmética: soma, subtração e as suas variantes. É de notar que nestas operações todas as *flags* são actualizadas neste bloco.

O componente responsável por fazer os deslocamentos lógicos (shifts) apenas tem que efectuar duas operações: shift aritmético direito e shift lógico esquerdo. Neste bloco são actualizadas as *flags* de signal, zero e carry.

Por último, a unidade lógica é responsável por efectuar todas as operações lógicas: and, or, xor, pass, nor, nand, entre outras. Neste componente, por a actualização das *flags* ser independente de operação para operação, é apenas feita a actualização das que forem necessárias (zero, signal ou nenhuma).

2.3.2 RAM

O bloco de memória RAM, tal como já foi referido, é endereçável ao nível da palavra, sendo que cada endereço de memória corresponde então a uma palavra de 2 bytes. Se o processador tem 16 bits de endereço, então a capacidade total de memória do processador é de 32 KB. Esta memória funciona com leitura assíncrona e escrita síncrona caso o enable de escrita se encontre activo (*we* = '1').

Quanto à descrição *VHDL*, a memória tem como sinal de entrada $data_{in}$, de saída $data_{out}$ e de endereço $addr$, ou seja, para as instruções existentes de acesso à memória temos que podem ser feitas as operações de **LOAD** e **STORE**. Quanto à primeira é carregado para um registo C o valor da memória endereçável pelo conteúdo do registo A ; quanto à segunda instrução é armazenado o conteúdo do registo B no endereço de memória apontado pelo conteúdo do registo A .

2.3.3 Flag Tester

Quanto à actualização das *flags*, tal como já foi referido, o método utilizado baseia-se em actualizar apenas as *flags* necessárias, ou seja, todas as *flags* guardadas entram dentro da ALU, mas dependendo da instrução a executar, é feita uma concatenação das *flags* a actualizar com as que não serão actualizadas. No entanto, no caso de ser uma operação aritmética, todas as *flags* são actualizadas.

O componente de Flag Tester serve então para indicar se se deve efectuar um determinado salto ou não dependendo se é instrução de salto ou se as condições de flag são válidas para que se efectue esse salto. Temos então como sinais de entrada as quatro *flags*, o código correspondente à condição de salto, o código da operação a efectuar (para distinguir os vários tipos de salto), um *enable* e como sinais de saída os valores das *flags* armazenados em registos e ainda um sinal s que indicará qual valor de PC a utilizar, se PC+1 ou se um outro valor de PC. Em primeiro lugar é verificada a condição de salto de acordo com os sinais presentes nos registos das *flags*, a operação de salto, e caso se verifique que a condição é verdadeira para o salto indicado, então é colocado s com valor lógico '1'. Caso a condição não seja cumprida, é colocado o valor lógico '0'.

À saída deste bloco temos ainda lógica adicional para indicar caso não seja uma operação de salto (*mux* regulado por *is_jump*), sendo que se for um jump **True** ou **False**, é mandado o sinal proveniente do *Flag Tester*, mas caso seja um outro tipo de salto (por exemplo, **Inconditional jump**), seja carregado na mesma o valor do salto e não PC+1.

2.4 WB

2.5 Máquina de Estados

3 Testes e Simulações

3.1 Fibonacci

4 Conclusão

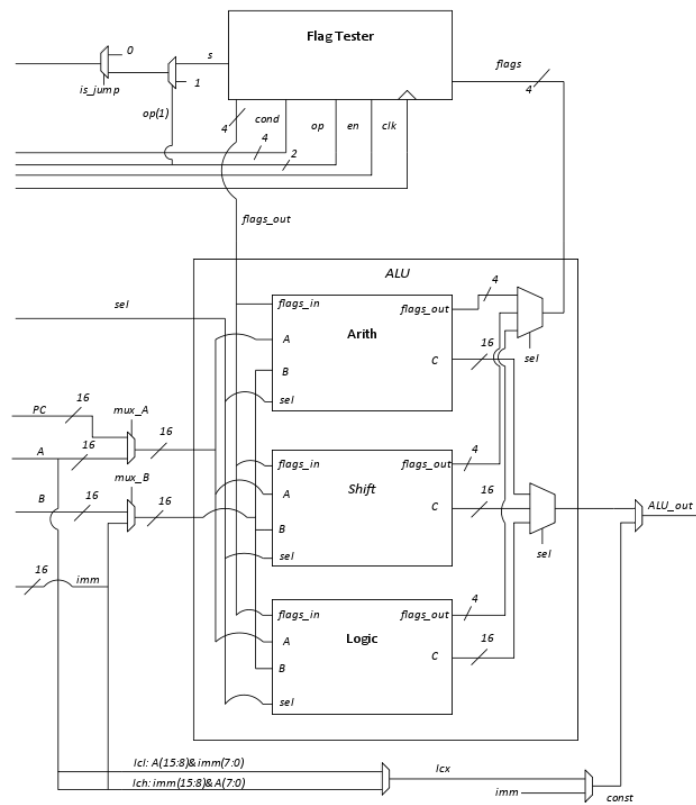


Figura 1: Bloco de Execução e acesso à memória