LABORATÓRIO DE ARQUITETURA DE COMPUTADORES

Experimento 3

Memória de Dados

GRUPO: 1	TURMA: A
Antonio Jorge Medeiros	RA: 620521
Daniel Souza Bertoldi	RA: 620548
Guilherme Gonçalves	RA: 620386
Lucas Buzzo	RA: 620408

1 Resumo (1/2 a 1 página)

Nesse experimento aprendemos um pouco sobre a relação entre memória e processador, como fazer o load word (lw), que carrega uma palavra da memória para o registrador, e o store word (sw), que grava o conteúdo de um registrador para a memória. Aprendemos também a fazer um salto condicional, o beq (branch if equal), que compara o conteúdo de dois registradores e, se forem iguais, faz com que o PC aponte para um determinado endereço de memória, que não necessariamente seu sucessor. O experimento foi ótimo para entender também como é feita a seleção do tipo de instrução (R, I, J) e ajudou a esclarecer um pouco mais o funcionamento da leitura de instruções, pelo processador.

Conseguimos fazer tudo, sem muita dificuldade em entender a proposta e a lógica do que era para ser feito. Nossa maior dificuldade foi relacionar sinais de diferentes componentes, alguns erros de compilação e exibição.

2 Código

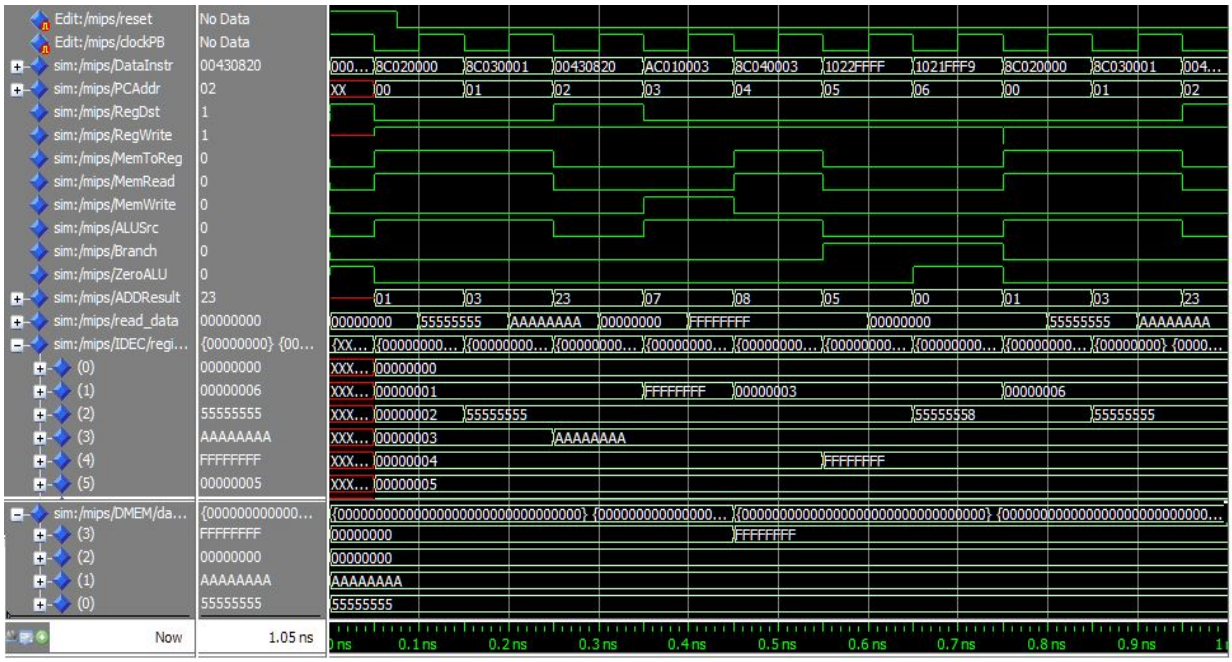
```
.data
mem:    .word   0x55555555, 0xAAAAAAAA

.text
lb2:
        lw      $2, mem
        lw      $3, mem + 4
        add     $1, $2, $3
        sw      $1, mem + 12
        lw      $4, mem + 12
lb1:
        beq     $1, $2, lb1
        beq     $1, $1, lb2
        nop
```

Utilizamos novamente o MARS para descrever e compilar o código em assembly contido no *program.mif*. "mem" é o label que contém o endereço da memória pré definida, com valores mem[0] = 0x55555555 e mem[1] = 0xAAAAAAAA. Não foi necessário inicializar o restante (mem[02..FF] = 0x00000000), pois o próprio MARS inicializa toda a memória com zeros. $2 recebe 0x55555555 (mem[0]) e $3 recebe 0xAAAAAAAA (mem[1]), então $1 recebe 0xFFFFFFFF, que é a soma de $2 e $3, e é guardado em mem[3]. Em seguida, $4 lê da memória mem[3] e recebe o valor 0xFFFFFFFF.

Por fim, temos dois beq, se $1 = $2, então o PC vai para o label lb1, equivalente a voltar uma palavra, e o programa entra em loop, pois irá continuar executando a mesma instrução. Se $1 != $2, então vai para o próximo teste, que sempre será verdadeiro, então o PC vai para o label lb2, equivalente a voltar seis palavras, reiniciando o programa.

3 Simulação e teste



3.1 Discussão

O clockPB é o clock da placa, que é invertido, por isso começa em nível lógico alto, no código VHDL, o sinal de clock interno recebe o inverso de clockPB. O reset fica em nível lógico alto até 75ps, pegando a subida do clock de 0 para 1 e assim inicializando os registradores, cada um com o seu valor, correspondente ao seu índice. Essa inicialização fica clara nas linhas 68~74 da componente Idecode que, sempre que ocorrer um pulso de clock e reset estiver em 1, os registradores serão resetados e atribuídos os valores de seus índices.

PCAddr é o apontador para qual instrução será executada, PCAddr recebe o valor do PC_out, da componente Ifetch, que foi desenvolvida no Experimento 1. Para cada ciclo de clock, PCAddr é incrementado de 1 e irá apontar para a próxima instrução. Nesse caso, quando PCAddr é 5, ele volta para 0, porque é quando o beq é executado.

RegDst recebe 1 quando a instrução que será executada é do tipo R, que usa três registradores. Ele é ativado no terceiro ciclo de clock, quando PCAddr = 02, que é logo após terem sido lidos a memória[0] e memória[1] nos registradores 2 e 3, respectivamente, sua ativação é devido o fato de que a instrução realizada é de somar (add - 00430820) o conteúdo dos registradores 2 e 3 e armazenar o resultado no registrador 1. Note que no mesmo instante, RegWrite também está em nível lógico alto, porque é o responsável por informar se haverá escrita em algum registrador, nesse caso, a escrita é no registrador 1. Ambos fazem parte da componente Control e são usados na Top Level.

Analogamente, MemToReg, MemRead e MemWrite são bem fáceis de entender. MemRead e MemToReg são ativados quando a instrução executada é um load word (lw), o que ocorre nos dois primeiros ciclos de clock ou quando PCAddr é 1 e 2, respectivamente, é

feito um lw (8C020000; 8C030001) da memória[0] e memória [1] para os registradores 2 e 3, e informam à Top Level que será feita uma leitura da memória, e que o conteúdo lido será armazenado em um registrador. MemWrite recebe 1 quando se deseja escrever na memória, que é o que é feito quando a instrução executada é um sw (store word - AC010003), que é feito exatamente nesse instante em que MemWrite é 1, escrevemos o conteúdo do registrador 1 na posição de memória[3]. Fica evidente a escrita quando olhamos a posição de memória 3 na simulação e vemos exatamente 0xFFFFFFFF, que era o conteúdo do registrador 1. Logo após, no 5º ciclo de clock, quando PCAddr = 04, é feito um load word da memória[3] para o registrador 4 e, na simulação, dá para ver claramente que o registrador 4 recebe o valor que estava na memória 3 exatamente um ciclo de clock depois. Assim como o controle dos registradores, o controle da memória também é feito na componente Control.

Finalizando nosso código, temos a última instrução implementada, o beq, que também é bem simples: quando executado, se a condição para o salto for verificada, o sinal Branch recebe 1, e isso informa à Top Level de que um salto será feito, essa informação é processada na componente Ifetch, que faz com que, em vez de PC receber normalmente o seu imediato sucessor (PC + 4), na verdade receberá o endereço da memória de instrução definido pelos bits menos significativos da instrução. Nesse caso em particular, há dois testes sendo feitos, primeiro (1022FFFF), se o registrador 1 e 2 são iguais , então PC iria recuar uma posição e entrar em loop, o que não ocorre. No segundo teste (1021FFF9), tanto Rs quanto Rt recebem o registrador 1, o que obviamente é sempre verdade, então o salto é feito, recuando o PC em 6 posições (PC - 28), o -28 é porque sempre que termina uma instrução, PC vai para a próxima, então na verdade são recuadas 7 posições, retornando ao início do código.