PCS3225 - Sistemas Digitais II
Projeto 6 - Processador Monociclo PoliLEG

Edson S. Gomi

16/11/2020

Nos projetos anteriores, você construiu a memória de instruções (`rom`), a memória de dados (`ram`), o banco de registradores (`regfile`), a Unidade Lógico-Aritmética (`alu`), a unidade funcional `signExtend` e a Unidade de Controle (`controlunit`) do processador monociclo PoliLEG. O objetivo deste trabalho é completar o PoliLEG, construindo o fluxo de dados (`datapath`) e interligá-lo com a unidade de controle. O teste do processador será feito através da execução de um programa instalado na `rom`, junto com os dados colocados na `ram`.

Introdução

Conforme foi explicado no trabalho anterior, o processador PoliLEG usa o paradigma de Unidade de Controle e Fluxo de Dados. Na Figura 1 podemos observar o diagrama de blocos, com as interligações entre a unidade de controle e o fluxo de dados. É importante lembrar que as memórias de instruções e de dados são componentes externos ao processador.

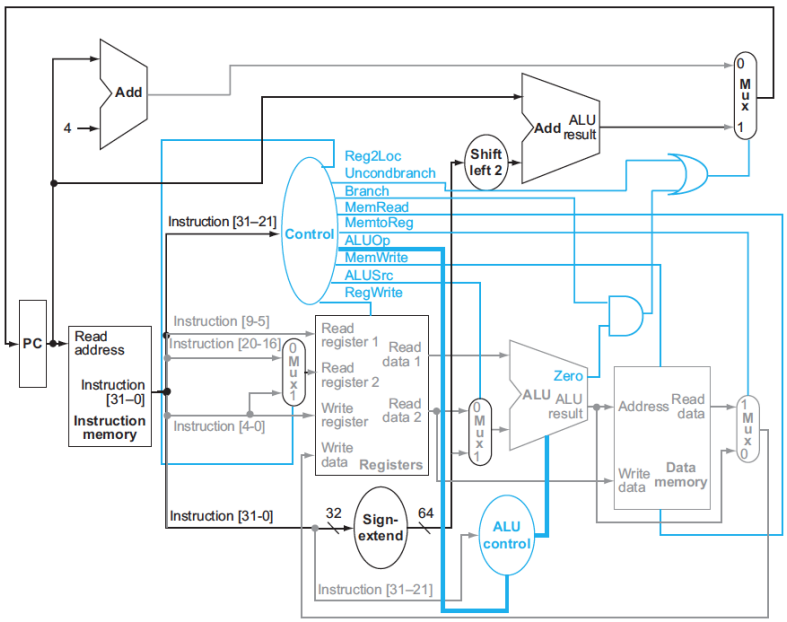

Figura 1: Diagrama do PoliLEG

Para facilitar o projeto dos circuitos deste trabalho, será bom rever o funcionamento e o formato das instruções do PoliLEG. A leitura recomendada são os Capítulos 2 (Seções 2.1 a 2.10), 4 (Seções 4.1 a 4.4) e o Apêndice A5 do livro texto da disciplina (*Computer Organization and Design - The Hardware/Software Interface, ARM Edition*, de David Patterson & John Hennessy).

Atividades

P6A1 (8 pontos) Implementação do Processador Monociclo PoliLEG.

Projeto 6, Atividade 1

Para implementar o PoliLEG, siga os passos explicados a seguir.

(a) Inicialmente implemente o componente correspondente ao fluxo de dados (`datapath`). O fluxo de dados é composto pelo banco de registradores (`regfile`), a Unidade Lógico-Aritmética (`alu`), a unidade funcional `signExtend` e o `Shiftleft2`. A entidade `datapath` é mostrada na Listagem 2.

```
entity datapath is
  port(
    -- Common
    clock : in bit;
    reset : in bit;
    -- From Control Unit
    reg2loc : in bit;
    pcsrc: in bit;
    memToReg: in bit;
    aluCtrl: in bit_vector(3 downto 0);
    aluSrc: in bit;
    regWrite: in bit;
    -- To Control Unit
    opcode: out bit_vector(10 downto 0);
    zero:   out bit;
    -- IM interface
    imAddr: out bit_vector(63 downto 0);
    imOut:  in  bit_vector(31 downto 0);
    -- DM interface
    dmAddr: out bit_vector(63 downto 0);
    dmIn:   out bit_vector(63 downto 0);
    dmOut:  in  bit_vector(63 downto 0)
  );
end entity datapath;
```

Figura 2: Entidade datapath

(b) No próximo passo, implemente o componente do PoliLEG, cuja entidade é denominada `polilegsc` e cuja descrição está na Figura 3. O `polilegsc` essencialmente contém a interligação do fluxo de dados `datapath` com a unidade de controle `controlunit`. Observe que as interligações entre o fluxo de dados e a unidade de controle correspondem aos sinais com cor azul na Figura 1. Também observe que as memórias `ROM` e `RAM` não fazerm parte do processador e, portanto, são interligadas externamente ao `polilegsc`.

São fornecidos os arquivos `rom.dat` e `ram.dat` que contém as instruções e os dados do programa em Assembly do PoliLEG que calcula o Máximo Divisor Comum (MDC). Este programa implementa o algoritmo de Euclides, que está descrito na Figura 4. Use o conteúdo desses arquivos `dat` para testar a sua implementação do `polilegsc`. Cuidado: observe que esta implementação não tem proteção contra o uso de zero e inteiros negativos.

```
entity polilegsc is
    port (
      clock, reset: in bit;
      -- Data Memory
      dmem_addr:        out     bit_vector(63 downto 0);
      dmem_dati:        out     bit_vector(63 downto 0);
      dmem_dato:        in      bit_vector(63 downto 0);
      dmem_we:   out     bit;
      -- Instruction Memory
      imem_addr:        out     bit_vector(63 downto 0);
      imem_data:        in      bit_vector(31 downto 0)
    );
end entity;
```

Figura 3: Entidade polilegsc

```
int MDC(int a, int b){
    while(a!=b){
        if(a>b) a = a-b;
        else b = b-a;
    }
    return a;
}
```

Figura 4: Algoritmo de Euclides

O conteúdo da `rom` corresponde ao conteúdo binário da implementação do algoritmo de Euclides em Assembly do PolLEG, conforme é mostrado na Figura 5. Esta implementação assume que a memória de dados contém os valores 8000000000000000_{16}, a e b, respectivamente nas posições 0, 8 e 16 da memória de dados. No arquivo `ram.dat` fornecido para testes, os valores de a e b são 9 e 15 respectivamente.

```
                LDUR X1,[X31,0]         -- X1 = 8000000000000000
                LDUR X2,[X31,8]         -- X2 = a
                LDUR X3,[X31,16]        -- X3 = b
L1:             SUB X4,X2,X3            -- X4 = temp1
                CBZ X4,L0
                AND X5,X4,X1            -- X5 = temp2
                CBZ X5,L3
                SUB X3,X3,X2
                B L1
L3:             SUB X2,X2,X3
                B L1
L0:             STUR X2,[X31,8]
L2:             B L2                    -- Loop eterno.
```

Figura 5: Algoritmo de Euclides em Assembly do PoliLEG

Prepare o seu testbench para o testar e depurar o funcionamento do `polilegsc`. Também faça uso do GTKWave para analisar o funcionamento do seu projeto (ver Figura 6).

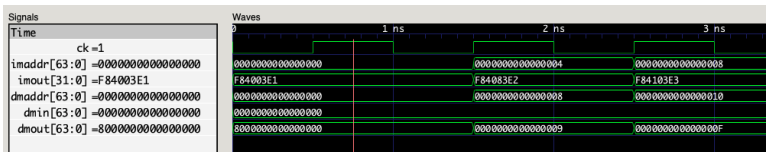

Figura 6: Carta de Tempos no GTKWave

Você deverá submeter ao Juiz o arquivo VHDL da descrição (entidade e arquitetura) do `polilegsc`. O arquivo `polilegsc.vhd` deverá conter todos os componentes que utilizou para implementar o `polilegsc`, mas você não deve incluir os arquivos `rom.dat` e `ram.dat` que foram fornecidos para o teste do circuito. Os arquivos `rom.dat` e `ram.dat` já estão presentes no Juiz, que executará o MDC com outros valores de a e b para avaliar a sua implementação.

P6A2 (2 pontos) Implementação do programa Fibonacci para o PoliLEG. Projeto 6, Atividade 2

Na Figura 8 temos a listagem de uma implementação em assembly do PoliLEG de um programa para o cálculo dos valores da sequência de Fibonacci (`https://www.mathsisfun.com/numbers/fibonacci-sequence.html`). Esta implementação assume que nas posições 0,8,16 e 24 da memória de dados temos inicialmente o primeiro número da sequência, o segundo número, a quantidade de números a serem calculados e a constante 1 (um), respectivamente. Sugerimos calcular ao menos 13 números da sequência de Fibonacci nos testes com o testbench.

```
        LDUR X0,[X31,0]         -- 1o. numero da sequencia
        LDUR X1,[X31,8]         -- 2o. numero da sequencia
        LDUR X2,[X31,16]        -- Quantidade de numeros a serem calculados
        LDUR X3,[X31,24]        -- Constante 1
Lo:     ADD X5,X1,X0
        ADD X0,X1,X31
        ADD X1,X5,X31
        SUB X2,X2,X3
        CBZ X2,L1
        B Lo
L1:     STUR X1,[X31,32]
L2:     B L2    -- Loop eterno (por que?)
```

Figura 7: Algoritmo Fibonacci em Assembly do PoliLEG

A tarefa a ser feita nesta Atividade é traduzir as instruções do programa da Figura 8 para a forma binária e colocar as instruções no arquivo `rom.vhd` (não confundir com o arquivo `rom.dat` fornecido para a Atividade 1. O arquivo template `rom.vhd` contém o código do programa MDC. Você deve substituir as instruções pelo código do programa Fibonacci. Para os seus testes, você também deverá adaptar os seguintes itens do seu processador monociclo:

- O conteúdo da memória de dados no arquivo `ram.dat`;
- A descrição do componente ROM no testbench para

```
component rom is
  port (
    addr : in  bit_vector(3 downto 0);
    data : out bit_vector(31 downto 0)
  );
end component;
```

Figura 8: Algoritmo Fibonacci em Assembly do PoliLEG

- A instanciação da ROM. Observe que na Atividade 1, a ROM tinha 256 posições de memória e nesta Atividade a ROM tem apenas 16 posições. Isso significa que o tamanho da via de endereçamento mudou de 8 bits para 4 bits.

Você deverá submeter ao Juiz apenas o arquivo `rom.vhd`. Não é preciso submeter a sua implementação do processador monociclo (o arquivo `polilegsc.vhd`), assim como o arquivo `ram.dat`. Esses arquivos estarão presentes no Juiz.

Modelo de Memória do PoliLEG

O objetivo desta Seção é explicar em detalhe alguns aspectos do modelo de memória do PoliLEG, para facilitar o entendimento para a implementação do processador, os testes e a depuração do circuito.

O PoliLEG possui dois alinhamentos de memória: o de instruções (IM) e o de dados (DM). As memórias possuem palavras sempre de 8bits, mas só se pode ler instruções de 32 em 32 bits, e dados de 64 em 64 bits. Note que o ARM real não possui alinhamento para a memória de dados.

O alinhamento impõe uma restrição no acesso a ambas memórias (ROM - Instruções e RAM - Dados). No caso da memória ROM, quando um acesso a uma instrução é feito, devido ao alinhamento, os dois bits menos significativos do endereço são ignorados! Isso acontece pois a memória precisa de 4 palavras de 8bits para formar uma palavra de 32bits. Dessa forma, quando acessa-se a posição 0x0 da memória de instruções (IM), a memória sempre retornará uma palavra de 32 bits formada pelas palavras 0x0, 0x1, 0x2 e 0x3. Essas quatro palavras de 8bits formam a instrução de 32bits esperada pelo processador, sendo que os bits menos significativos da instrução estão na posição 0x0 e os mais significativos na posição 0x3. A próxima instrução começa na posição de memória 0x4, a seguinte na 0x8, depois 0xC e assim por diante. Note que todos os endereços do PoliLEG sempre terminam em 00 (os bits "ignorados" pelo processador). Se o seu processador tentar um acesso de memória que não termina em 00, tem algo errado! Eis um exemplo de conteúdo da ROM de instruções:

- 0x3 ...
- 0x4 11111000
- 0x5 01000000
- 0x6 10000011
- 0x7 11100010
- 0x8 ...

Ao acessar a posição 0x4, a memória retornará a palavra: 11111000 01000000 10000011 11100010. Esta é a segunda instrução do MDC, correspondente ao LDUR.

O mesmo acontece com a memória de dados (DM), mas neste caso o alinhamento é de 64bits, então os 3bits menos significativos do endereço serão desconsiderados pois precisamos de 8 palavras de 8bits para formar uma palavra de 64bits. Nessa memória, para acessarmos a primeira palavra de 64bits, usamos o endereço 0x0, mas a segunda está no endereço 0x8! A seguinte estará na posição 0x10, depois 0x18 e assim por diante.

O alinhamento está tão enraizado no PoliLEG, que algumas instruções nem mesmo guardam os últimos bits. É o caso do CBZ e do B, por exemplo. No campo de endereços, o endereço de salto é relativo ao PC atual, então o conteúdo do campo de endereço dessas instruções nem possui os dois últimos bits pois eles sempre são zero! Quando for somar o salto ao PC, o componente shiftLeft fará o trabalho de colocar os dois zeros faltantes (veja o diagrama no enunciado). Isso foi feito pelos projetistas do ARM pois se armazenássemos a quantidade de endereços a saltar completa na instrução, os dois últimos bits sempre seriam zero, desperdiçando espaço e limitando o espaço de salto. A idéia então é armazenar somente a parte importante. E.g. se deseja saltar uma instrução para frente, o campo de salto da instrução será 0x1 pois o shiftLeft preencherá os bits restantes e somará na verdade 0x4 ao PC, ficando PC+0x4, que é o endereço da próxima instrução (não faz muito sentido saltar para a próxima instrução, foi só um exemplo).

Instruções para Entrega

Há um link específico no e-Disciplinas para submissão deste projeto. Acesse-o somente quando estiver confortável para enviar sua solução. Em cada atividade, você pode enviar apenas um único arquivo codificado em UTF-8. O nome do arquivo não importa, mas sim a descrição VHDL que está dentro. A entidade da síntese (implementação) que você submeterá precisa estar, obrigatoriamente, de acordo com o que foi definido/especificado no enunciado e deve ser idêntica na sua solução ou o juiz não irá processar seu arquivo.

Qualquer editor de código moderno suporta UTF-8 (e.g. Atom, Sublime, Notepad++, etc)

O juiz corrigirá imediatamente sua submissão e retornará com a nota. Caso não esteja satisfeito com a nota, você pode enviar novamente e somente a última nota para aquele problema será válida. A nota para este trabalho é composta pela soma ponderada das notas dadas pelo juiz para cada atividade. Faça seu *testbench* e utilize um simulador de VHDL para validar sua solução antes de postá-la para o juiz.

A quantidade de submissões para estes problemas é limitada a 5 por problema.

Não use a biblioteca std_logic_1164, a textio e qualquer outra biblioteca não padronizada, para minimizar possível fonte de problemas. Também se certifique que não imprime nada na saída da simulação.

Atenção: não atualize a página de envio e não envie a partir de conexões instáveis (e.g. móveis) para evitar que seu arquivo chegue corrompido no juiz.