Trabalho prático de Arquitetura de Computadores II

Professor Eliseu César Miguel

Autor: Fernando Carvalho da Silva Coelho [fccoelho@dcc.ufmg.br]

Implementação de um processador

1. Introdução

Neste trabalho, o objetivo implementar um processador multiciclo de 16 bits com instruções de tamanho fixo, com capacidade de fazer operações aritméticas simples e com apenas 8 registradores.

Dessa forma, para implementar esse processador, deverá ser utilizada a linguagem de descrição de Hardware (HDL) Verilog, cuja utilização é comum nos projetos envolvendo desenvolvimento de hardware.

Assim, o estudante terá a oportunidade de aprender a utilizar ferramentas comuns para esse propósito, além de entrar em contato com o processo de desenvolvimento de um processador simples, que reforçará os conteúdos vistos na disciplina de Organização de Computadores I e II.

2. Descrição do processador

O processador a ser implementado sempre executa uma instrução a cada intervalo de 4 ciclos de clock, possui 8 registradores de 16 bits e uma unidade lógica aritmética (ALU) capaz de fazer operações de soma e subtração. Dessa forma, todas as instruções, devem estar prontas na entrada de dados, exatamente ao final dos 4 ciclos de clock necessários para a execução de uma instrução.

As instruções reconhecidas pelo processador estão descritas na Tabela 1.

Operação	Código	Função realizada	Explicação
rep Rx, Ry	111	Rx ← [Ry]	**Rep**laces Rx with Ry
ldi Rx, #D	101	Rx ← D	**L**oa**d**s **i**mmediate #D in register Rx
add Rx, Ry	000	Rx ← [Rx] + [Ry]	**Add**s Rx and Ry and stores in Rx
sub Rx, Ry	001	Rx ← [Rx] - [Ry]	**Sub**stracts Rx and Ry and stores the result in Rx
nan Rx, Ry	010	Rx ← [Rx] ~& [Ry]	Rx **nan**d Ry and stores in Rx
out Rx	100	[Rx]	**Out**puts Rx in the Bus

Tabela 1

Essas instruções podem ser divididas em três formatos distintos:

```
15 14 13 12 11 10 09 08 07 06 05 04 02 01 00
 I  I  I  X  X  X  Y  Y  Y  -  -  -  -  -  -
 I  I  I  X  X  X  D  D  D  D  D  D  D  D  D
 I  I  I  X  X  X  -  -  -  -  -  -  -  -  -
```

Nessa representação I define os campos de opcode da instrução, X e Y os registradores envolvidos nas operações e D representa os bits de uma constante associada à instrução (imediato).

2.1. Exemplo de programa

Na Tabela 2, está um exemplo de programa reconhecido pelo processador, em que são carregados dois valores e a diferença é armazenada no registrador r0 e em seguida o resultado sai pelo Bus.

Tempo (Clocks)	Assembly	Versão binária
0	ldi r0, #28	101 000 0000011100

4	ldi r1, #10	101 001 0000001010
8	sub r0, r1	001 000 001 0000000
12	out r0	100 000 0000000000

Tabela 2: Exemplo de programa

2.2. Caminho de dados

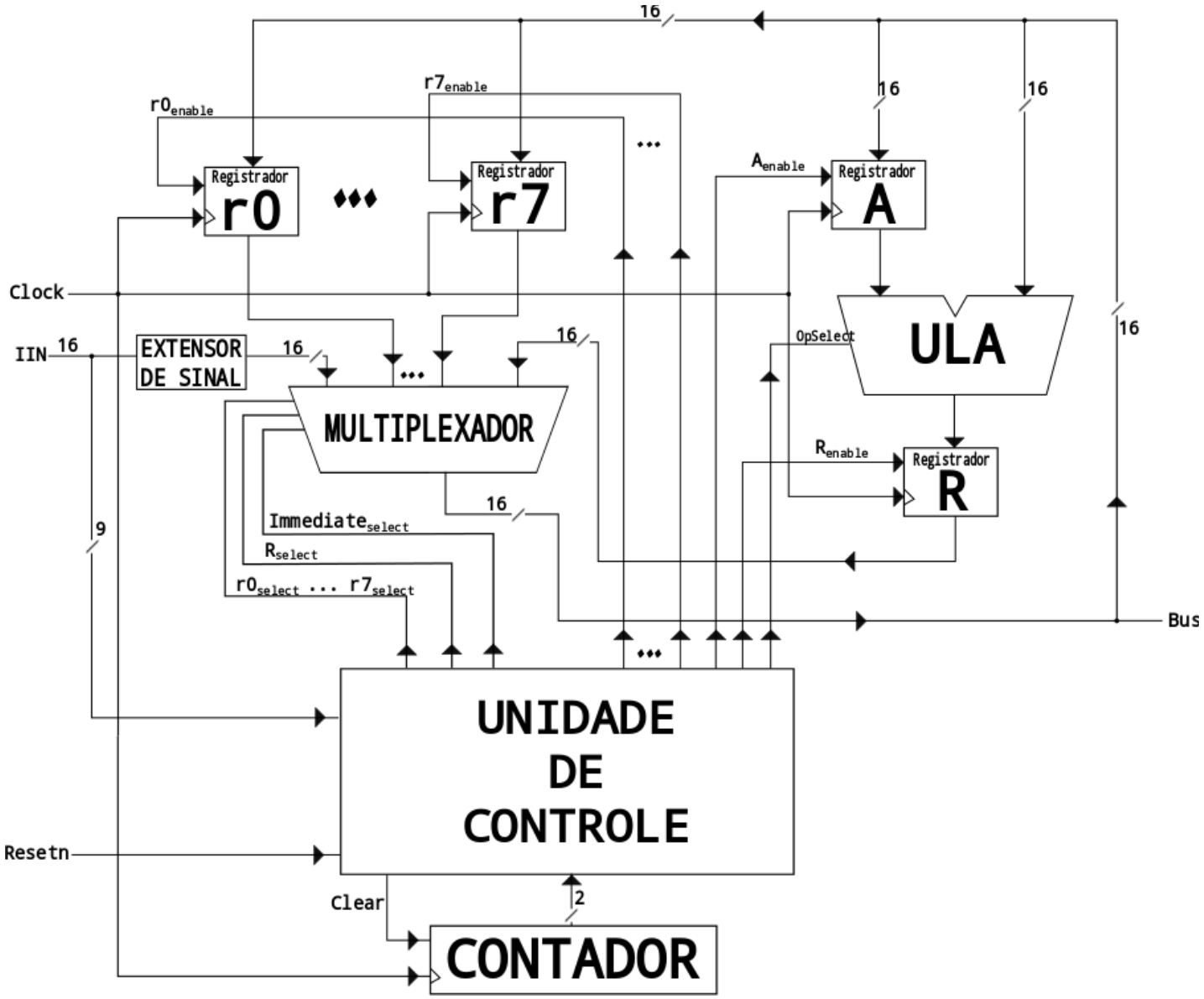

Figura 1: Representação esquemática da arquitetura do processador.

A partir desse caminho de dados, para a execução das instruções definidas na seção 2, são necessários os seguintes ciclos:

Primeiro ciclo: a instrução vinda do IIN entrará na UNIDADE DE CONTROLE, para que possa ser decodificada, ou seja, nesse ciclo, o processador descobre qual instrução será executada.

Segundo ciclo: ocorre a seleção do primeiro operando da instrução através do MULTIPLEXADOR. Esse primeiro operando é armazenado no registrador A na entrada da esquerda da Unidade Lógico-Aritimética (ULA).

Terceiro ciclo: ocorre a seleção do segundo operando da instrução através do MULTIPLEXADOR. Esse segundo operando vai direto para a entrada da direita da ULA. Também nesse ciclo, o resultado é guardado no registrador G.

Quarto ciclo: o resultado presente no registrador G é transferido para o registrador destino através do MULTIPLEXADOR.

Aqui, cabe ressaltar os seguintes pontos:

- os dados dos registradores, estão sempre disponíveis na saída de dados dos mesmos.

- os sinais enable habilitam a escrita no registrador correspondente.

Todas as entradas do processador serão associadas a um estímulo externo, conforme ilustração da Figura 2. Assim, os sinais de clock, a entrada de instruções e o sinal de reset deverão ser geradas no testbench e ligadas às interfaces do processador. Um exemplo desse esquema, se encontra na seção 5.

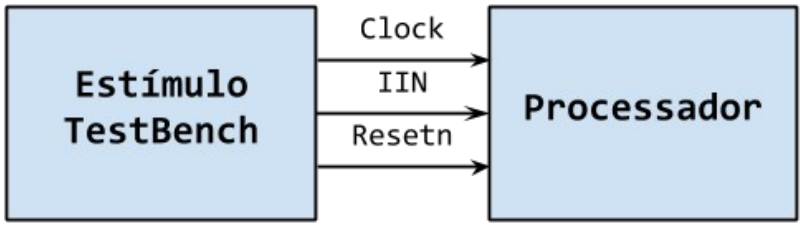

Figura 2: Funcionamento do processador associado ao seu testbench.

2.3. Módulos e sinais do processador

Módulo	Descrição
Registradores r0 ... r7, A, R e IR	Armazenam dos dados da entrada de dados, quando o sinal de enable estiver ativado e o sinal de clock estive na borda de subida.
Extensor de sinal	Pega a entrada de instruções (DIN) no formato IIIXXXDDDDDDDDD e a transforma em 000000DDDDDDDDD , separando assim o imediato dos valores da instrução.
Multiplexador	Utiliza os sinais de select para selecionar qual entrada será enviada aos registradores, à ULA e ao Bus.
ULA	Realiza as operações aritméticas e lógicas do processador.
Unidade de controle	Seleciona quais sinais devem estar ativos em cada estágio do processador.
Contador	Determina qual o estágio atual do processador a partir do estímulo da borda de subida do clock.

Tabela 3: Descrição dos módulos do processador.

Sinal	Descrição
Resetn	Reinicia o contador, assim que for liberado, ou seja, na borda de descida do sinal.
OpSelect	Determina qual a instrução será executada pela ALU.

Tabela 4: Sinais mais importantes do processador.

3. Recomendações para a implementação

A fim de facilitar a decodificação dos registradores, recomenda-se a criação de um módulo que decodifique os 3 bits que representam o número do registrador, em uma sequência de 8 bits, em que cada posição represente se o registrador está ativo ou não, de acordo com a Tabela 5.

Número do registrador	Código do registrador na instrução	Decodificação
0	000	10000000
1	001	01000000
2	010	00100000
3	011	00010000

4	100	00001000
5	101	00000100
6	110	00000010
7	111	00000001

Tabela 5: Decodificação dos sinais de ativação dos registradores

Para visualizar o comportamento do processador, recomenda-se a utilização do aplicativo GTKWave. Assim, a partir desse aplicativo, foi gerada a Figura 2 cuja saida representa a execução do programa de teste apresentado na seção 2.1.

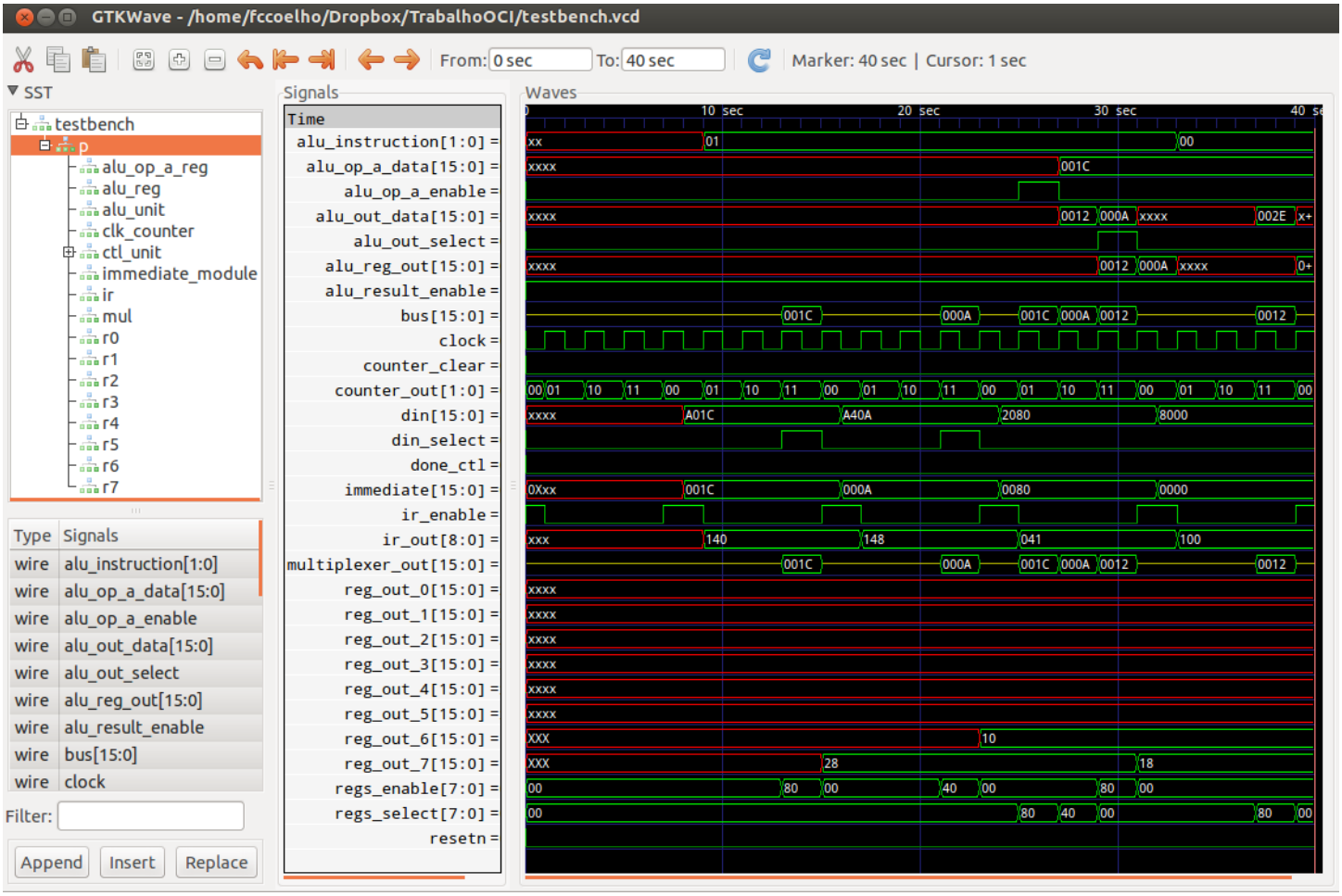

Figura 2: Uso do GTKWave a partir do programa de teste.

Conforme a figura 2, é importante notar que o sinal de *reset* deverá ser ativado, **somente**, após o momento em que a primeira instrução já estiver na entrada de dados, pronta para ser executada. Dessa forma, todas as demais instruções devem estar prontas na entrada de dados imediatamente após o término da última instrução executada.

Uma outra observação crucial para o funcionamento do sistema é o fato de que todos os registradores só são ativados para armazenamento de dados, apenas nas bordas de subida do sinal de clock (posedge). Nessas condições, a partir da Figura 2, pode-se notar que um ciclo de clock possui dois ticks de tempo.

4. Exemplo de referência

Para facilitar a apresentação da linguagem Verilog aos alunos, nessa seção há alguns trechos de código de exemplo.

Dessa forma, crie um arquivo `counter.v` para o módulo contador, com o seguinte código:

```
module counter(clock, clear, out);
input       clear;
input       clock;
```

```
output reg [1:0] out;

always @(posedge clock)
begin
	if(clear == 1)
		out <= 2'b00;
	else
		out <= out + 1'b1;
end
endmodule
```

Depois, crie um arquivo testbench.v para o módulo que servirá de testbench.

```
`include "counter.v"
module testbench;

reg	clock = 0;
reg	clear = 0;
wire [1:0] out;

always #1 clock = !clock;

initial $dumpfile("testbench.vcd");
initial $dumpvars(0, testbench);

counter c(clock, clear, out);

initial begin
	//These events must be in chronological order.
	# 5  clear = 1;
	# 1  clear = 0;
	# 20 $finish;
end
endmodule
```

Compile os módulos utilizando o iverilog:

```
iverilog testbench.v
```

Execute o programa gerado utilizando o aplicativo vvp:

```
vvp a.out
```

Utilize o GTKwave para visualizar a forma de onda das saidas do seu programa:

```
gtkwave testbench.vcd
```

5. Considerações finais

Todo o código será corrigido utilizando o compilador iverilog em ambiente Linux. Assim, será construido um testbench similar ao código a seguir, em que no início do código é incluido somente o arquivo processor.v:

```
`include "processor.v"

module testbench;

reg	clock  = 0;
reg  [15:0] iin;
reg	resetn = 1;

wire [15:0] bus;

always #1 clock = !clock;
```

```
initial $dumpfile("testbench.vcd");
initial $dumpvars(0, testbench);

processor p(clock, iin, resetn, bus);

initial begin
        # 0  resetn = 1'b0;
        # 8  iin = 16'b1010000000011100;
        # 8  iin = 16'b1010010000001010;
        # 8  iin = 16'b0010000010000000;
        # 8  iin = 16'b1000000000000000;
        # 8 $finish;
end
endmodule
```

É extremamente importante que os estudantes entreguem junto ao código, uma documentação contendo a descrição das decisões de projeto e testes realizados. Assim, *testes propostos para o processador serão levados em conta na hora da correção e avaliação dos trabalhos. Dessa forma, os estudantes deverão propor e executar testes, mostrando, **na documentação**, que seus testes foram executados corretamente. Nessas condições, é recomendado que sejam documentados pelo menos 3 testes*.

Manter-se rigoroso à especificação do caminho de dados apresentado na seção 2.2 colabora com o processo avaliativo, caso alguma parte do processador não funcione corretamente, já que a avaliação poderá ser feita em função de outros módulos. Assim, uma análise da saída da forma de onda, conforme apresentado na Figura 2, poderá esclarecer quais as funções e módulos estão corretos. Nessas condições, os sinais mais importantes são os que definem a saída dos registradores e os dados que passarem pelo BUS.

6. Referências

http://www.asic-world.com/verilog/veritut.html

http://iverilog.wikia.com/wiki/GTKWAVE