Implementação de um processador simples em HDL

Objetivo: Este trabalho tem como objetivo a implementação de um processador multiciclo de 16 bits com instruções de tamanho fixo, com capacidade de fazer operações aritméticas simples, negação de conjunção e com 8 registradores. Esta implementação, será feita utilizando-se uma linguagem de descrição de hardware (HDL), especificamente, a linguagem Verilog.

1.0 - Descrição do Processador

O processador implementado, possui 8 registradores de 16 bits, uma unidade de controle, um extensor de sinal, um contador, um multiplexador e uma unidade lógica aritmética (ALU) capaz de realizar operações de soma, subtração e negação de conjunção. O processador, sempre executa uma a cada intervalo de 4 ciclos de clock, assim, toda instrução deve estar pronta na entrada de dados ao final dos 4 ciclos de clock necessários para execução de uma instrução.

As instruções reconhecidas pelo processador, estão descritas na tabela abaixo:

Operação	Código	Função realizada	Explicação
rep Rx, Ry	111	Rx ← [Ry]	**Rep**laces Rx with Ry
ldi Rx, #D	101	Rx ← D	**Lo**a**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: Instruções do processador.

Essas instruções podem ser dividas em três formatos distintos:

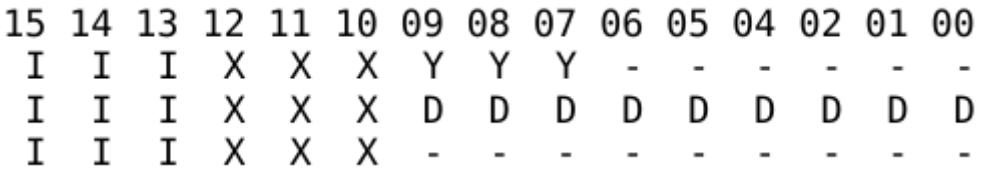

Imagem 1: Formatos de instruções.

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

Os módulos do processador são organizados, e seus dados circulam de acordo com a representação abaixo:

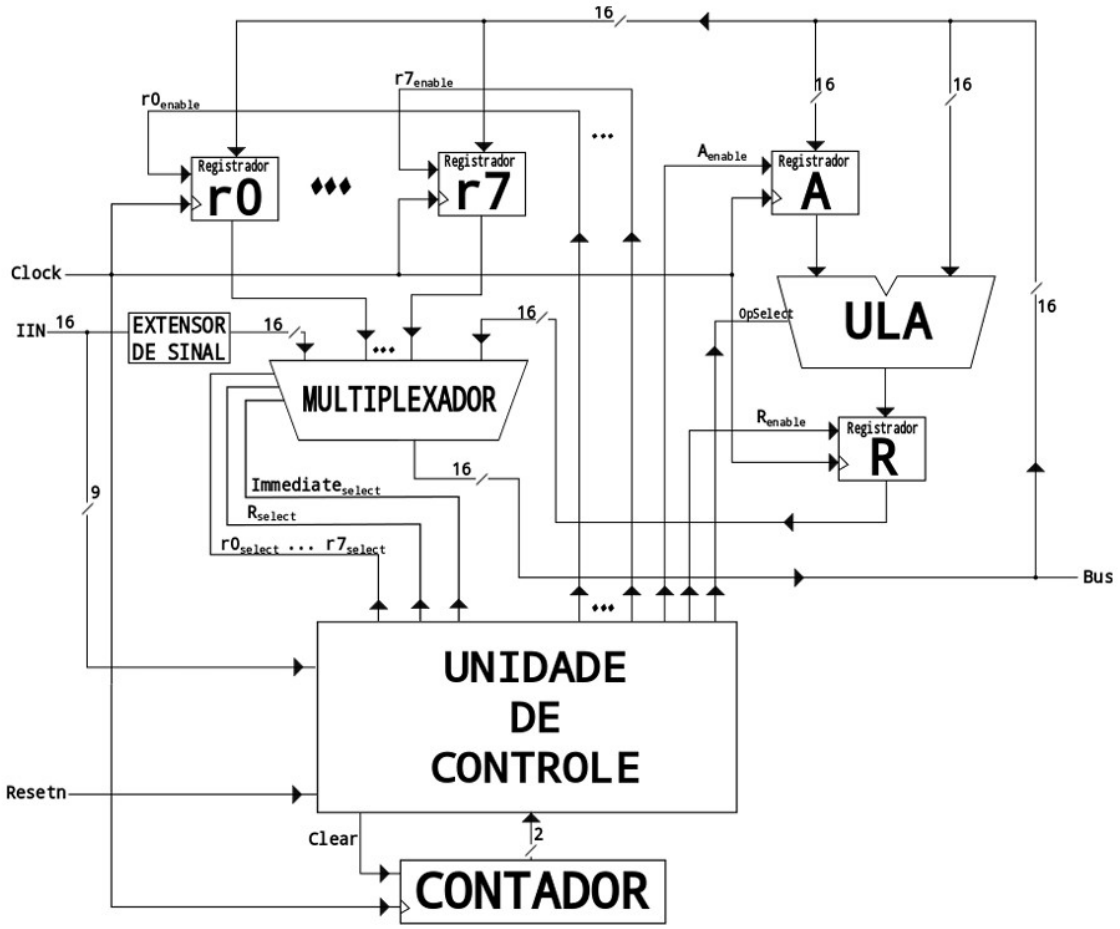

Imagem 2: Representação esquemática do processador.

A partir da representação esquemática, e do conjunto de instruções descrito acima, percebe-se a necessidade dos seguintes ciclos para sua execução:

1) **Primeiro ciclo:** Decodificação da instrução a ser executada. A **unidade de controle** recebe uma instrução vinda da **INN** para ser decodificada, ou seja, para a unidade de controle descobrir qual a instrução a ser realizada.
2) **Segundo ciclo:** Seleção do primeiro operando. Através do **multiplexador**, é selecionado o primeiro operando, que será armazenado no **registrador A** na entrada da esquerda da **unidade lógica aritmética**.
3) **Terceiro ciclo:** Seleção do segundo operando e armazenamento do resultado. Também através do **multiplexador**, o segundo operando é selecionado e vai direto para a entrada direita da **ULA**. Também neste ciclo, o resultado é guardado no **registrador G**.
4) **Quarto ciclo:** Seleção da origem e destino de um valor. O resultado armazenado no **registrador G**, **immediate** ou valor de um outro **registrador fonte** é transferido para o **registrador destino** através do **multiplexador**.

Sobre os ciclos, cabe ressaltar que:

- **a)** Os dados dos registradores estão sempre disponíveis na saída dos mesmos.
- **b)** Os sinais **enable** habilitam a escrita no registrador correspondente.

Além disso, todas as entradas do processador serão associadas a um estímulo externo. 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.

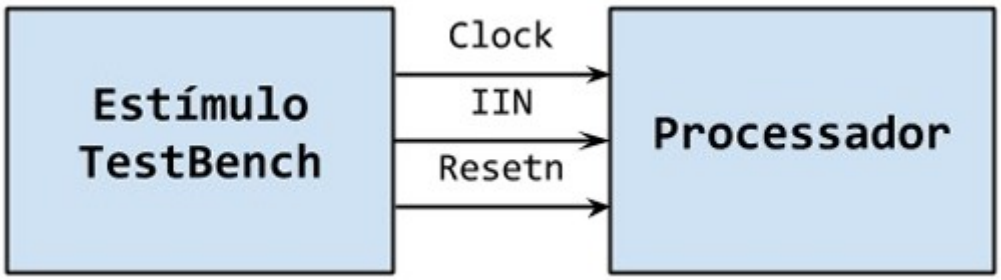

Imagem 3: Funcionamento do processador associado ao seu testbench.

2.0 - Implementação dos Módulos

A partir da descrição do processador acima, definimos os seguintes módulos 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 2: Descrição dos módulos do processador.

A implementação destes módulos, utilizando-se a linguagem de descrição de hardware Verilog, foi feita da seguinte forma:

2.1 - Registradores

O módulo registrador recebe como entrada um sinal de clock, um sinal enable e um valor de 16 bits. O registrador armazena um valor de 16 bits. A cada transição positiva do clock, caso o sinal enable seja verdadeiro, o registrador armazenará o valor de entrada.

```
module registrador(
    input wr_enable,
    input clock,
    input [15:0] entrada_reg,
    output reg [15:0] saida_reg
);

always @(posedge clock)
begin
    if(wr_enable)
```

```
        saida_reg <= entrada_reg;
end

endmodule
```

2.2 - Extensor de sinal

O módulo extensor recebe os 10 bits menos significativos da instrução, que corresponde ao campo imediato. O bit de sinal do valor imediato é replicado 6 vezes e concatenado com o valor imediato na saída.

```
module extensor(
    input  [9:0] iin,
    output [15:0] imediato
);

assign imediato[9:0] = iin;
assign imediato[15:10] = {6{iin[9]}};

endmodule
```

2.3 - Multiplexador

O módulo multiplexador possui dez entradas de 16 bits, os oito registradores gerais, o registrador R da ULA e o immediate. Além disso, ele também recebe um valor de 4 bits select, que serve como sinal controlador do multiplexador. De acordo com o valor de select, a saída receberá o valor de uma determinada entrada, conforme visto na tabela a seguir:

Valor do sinal select	Entrada selecionada
0000	Registrador 0
0001	Registrador 1
0010	Registrador 2
0011	Registrador 3
0100	Registrador 4
0101	Registrador 5
0110	Registrador 6
0111	Registrador 7
1000	Immediate
1001	Registrador R

Tabela 3: Valor Select x Entrada Selecionada para Saída.

Note que sinal select de 4 bits permite um multiplexador de até 16 entradas, para qualquer valor que não corresponda a nenhuma das 10 entradas descritas na tabela, o módulo desligará a saída.

```
module multiplexador (
    input  wire [3:0] select,
    input  wire [15:0] immediate,
    input  wire [15:0] r0, r1, r2, r3, r4, r5, r6, r7,
    input  wire [15:0] r,
    output reg  [15:0] saida
);
    always @(*)
    case (select)
        4'd0: saida = r0;
        4'd1: saida = r1;
        4'd2: saida = r2;
        4'd3: saida = r3;
        4'd4: saida = r4;
        4'd5: saida = r5;
        4'd6: saida = r6;
        4'd7: saida = r7;
        4'd8: saida = immediate;
        4'd9: saida = r;
        default: saida = 16'dz;
    endcase

endmodule
```

2.4 - Unidade Lógico Aritmética

O módulo ula recebe dois operandos de 16 bits e um sinal de controle de 2 bits chamado select. De acordo com o valor de select, o módulo realizará uma determinada operação conforme a tabela a seguir:

Valor do sinal select	Operação realizada
0	Soma
01	Subtração
10	Negação da Conjunção
11	no_op

Tabela 4: Valor Select x Operação Realizada.

Ao receber o sinal da operação no_op (não operação), a saída do módulo será desligada.

```
module ula (
    input wire [15:0] A,
    input wire [15:0] B,
    input wire [1:0] op_select,
    output reg [15:0] r
);

    always @(*)
    case (op_select)
        2'd0: r = A + B;
        2'd1: r = A - B;
        2'd2: r = ~(A & B);
        2'd3: r = 16’dz;
    endcase

endmodule
```

2.5 - Unidade de Controle

O módulo logica_de_controle recebe uma instrução, um sinal resetn e o valor do clock. Além disso, ele possui diversas saídas, sinais de controle para os outros módulos do processador. Os valores destes sinais são dados de acordo com a instrução e com o ciclo de clock.

```
`include "decodificador.v"

module logica_de_controle(
    input            resetn,
    input      [1:0] counter,
    input      [8:0] iin,
    output reg [3:0] mux_select,
    output     [7:0] regs_enable,
    output     [1:0] alu_op_select,
    output           a_reg_enable,
    output           alu_reg_enable,
    output reg       clear
);

`define REP 3'b111
`define ADD 3'b000
`define SUB 3'b001
`define NAN 3'b010
`define LDI 3'b101
`define OUT 3'b100
`define ALU_OPERATION (opcode == `ADD || opcode == `SUB ||
opcode == `NAN)
```

```
`define RX_SELECT      {1'b0,rx}
`define RY_SELECT      {1'b0,ry}
`define IMM_SELECT      4'b1000
`define ALU_OUT_SELECT  4'b1001
`define NO_OUTPUT       4'b1111
`define NO_OP            2'b11

reg  [2:0] opcode, rx, ry;
wire [7:0] regs_select;
decodificador dec(rx, regs_select);

// Habilita a escrita no registrador de entrada da ULA.
assign a_reg_enable   = (counter == 2'b01 && `ALU_OPERATION);

// Habilita a escrita no registrador de saída da ULA.
assign alu_reg_enable = (counter == 2'b10 && `ALU_OPERATION);

// Determina qual operação deve ser realizada pela ULA.
assign  alu_op_select   =  (`ALU_OPERATION)  ?  opcode[1:0]  :
`NO_OP;

// Cada elemento desse array de 8 bits é ligado ao campo
// 'wr_enable' de um dos 8 registradores endereçáveis.
assign regs_enable    = (counter == 2'b11 && opcode != `OUT) ?
regs_select : 8'h00;

always @(counter)
begin
    case(counter)

        2'b00:
            mux_select <= `NO_OUTPUT;

        2'b01:
            begin
                opcode = iin[8:6];
                rx     = iin[5:3];
                ry     = iin[2:0];

                if(`ALU_OPERATION)
                    mux_select <= `RX_SELECT;
            end

        2'b10:
            if(`ALU_OPERATION)
                mux_select <= `RY_SELECT;

        2'b11:
```

```
            case(opcode)
                `OUT   : mux_select <= `RX_SELECT;
                `REP   : mux_select <= `RY_SELECT;
                `LDI   : mux_select <= `IMM_SELECT;
                default: mux_select <= `ALU_OUT_SELECT;
            endcase

    endcase
end

always @(negedge resetn)
begin
    clear = 1'b1;
    clear = 1'b0;
end

endmodule
```

O módulo da unidade de controle utiliza-se de um decodificador, cuja implementação pode ser vista abaixo. O módulo decodificador recebe um valor chave de 3 bits como chave, e como saída, retorna um valor de 8 bits onde, o bit na posição do valor da chave tem o valor 1, e os demais tem valor 0.

```
module decodificador(
    input      [2:0] chave,
    output reg [7:0] saida
);

always @(chave)
    saida = 8'h01 << chave;

endmodule
```

A geração dos sinais a partir da unidade de controle para os demais módulos se dá de acordo com as seguintes expressões:

Sinal op_select da ULA: Se for uma instrução lógica aritmética, então op_select recebe os valores dos dois bits menos significativos do opcode. Senão, recebe o código de no_op.

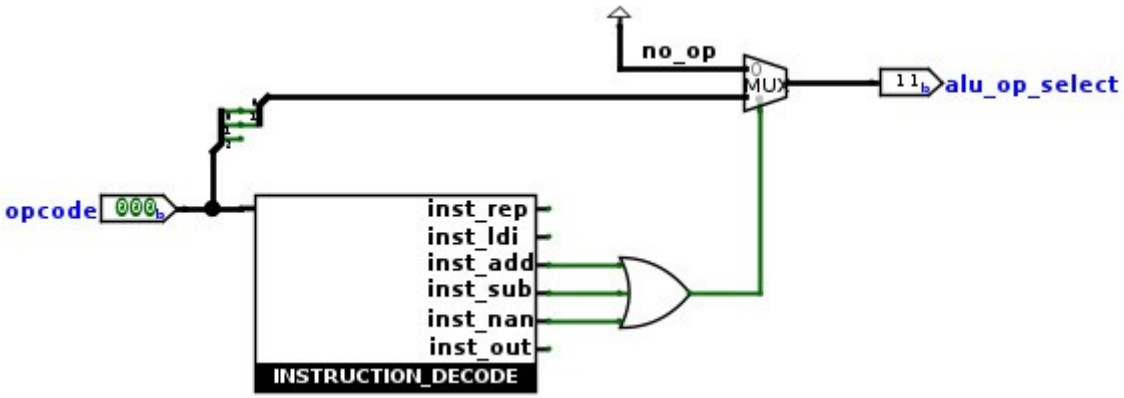

Imagem 4: Representação em circuito da expressão lógica.

Sinal enable do registrador A: Se estiver no ciclo 2 e for uma operação lógico aritmética, o sinal recebe verdadeiro. Senão, o sinal recebe falso.

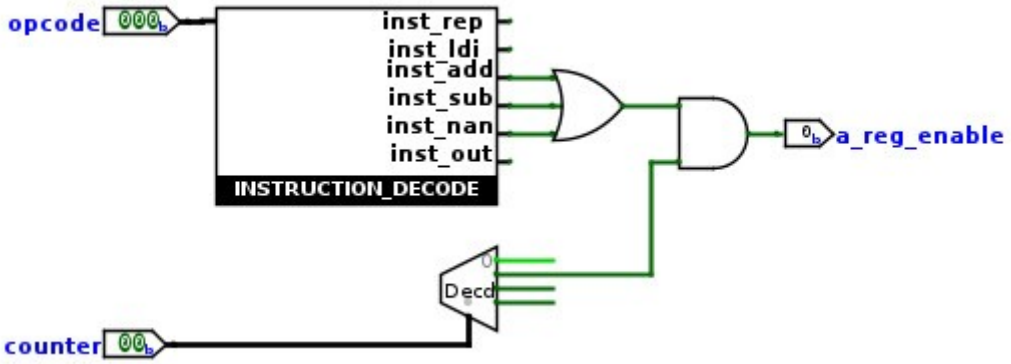

Imagem 5: Representação em circuito da expressão lógica.

Sinal enable do registrador R: Se estiver no ciclo 3 e for uma operação lógico aritmética, o sinal recebe verdadeiro. Senão, o sinal recebe falso.

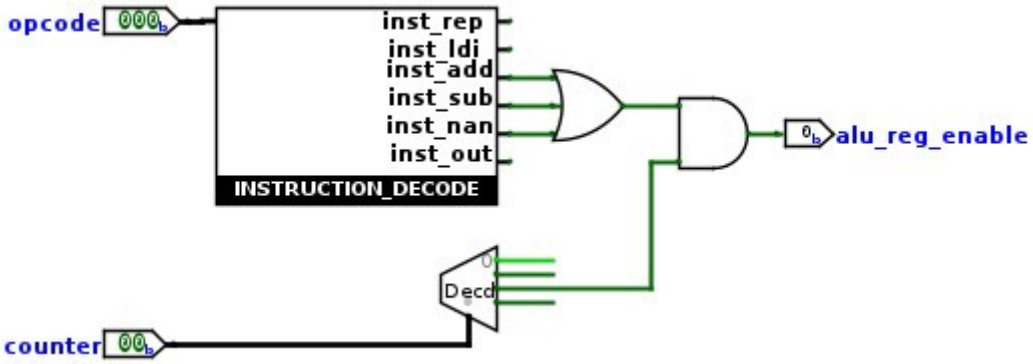

Imagem 6: Representação em circuito da expressão lógica.

Sinal enable dos demais registradores: Se estiver no ciclo 4 e a instrução não for a instrução ‘OUT’, então o sinal enable recebe a saída do decodificador. Senão, todos os seus bits serão 0.

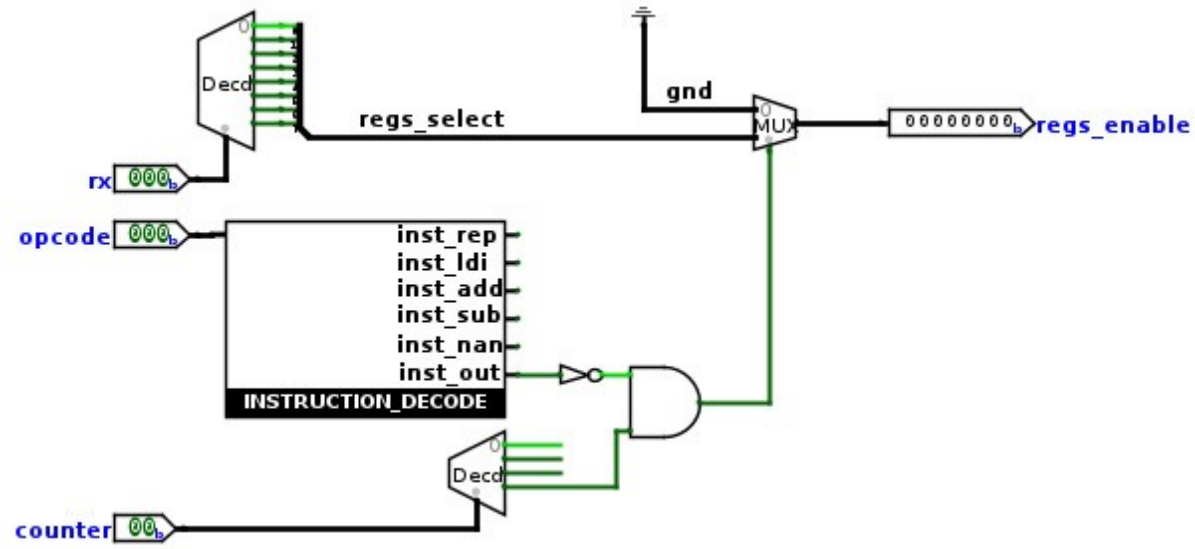

Imagem 7: Representação em circuito da expressão lógica.

2.6 - Contador

O módulo contador recebe como entrada um sinal de clock e um sinal assíncrono clear. A cada iteração positiva do clock, o valor do contador é incrementado. Ao receber o sinal clear, o contador vai para zero.

```
module contador(
    input clock,
    input clear,
    output reg [1:0] saida_cont);

always @(clear)
    saida_cont <= 2'b00;

always @(posedge clock)
    saida_cont <= saida_cont + 2'b01;


endmodule
```

Os módulos descritos acima foram organizados em um arquivo processador.v de acordo com a representação esquemática apresentada na figura 2.

```
`include "registrador.v"
`include "logica_de_controle.v"
`include "multiplexador.v"
`include "contador.v"
`include "extensor.v"
`include "ula.v"
```

```
module processador(
    input         clock,
    input         resetn,
    input  [15:0] iin,
    output [15:0] bus
);

wire [3:0]  mux_select;
wire [7:0]  regs_enable;
wire [1:0]  alu_op_select;
wire [1:0]  saida_contador;
wire [15:0] imm, a_reg_out, alu_out, alu_reg_out;
wire [15:0] r0_out, r1_out, r2_out, r3_out, r4_out, r5_out,
r6_out, r7_out;

contador cont(clock, clear, saida_contador);

extensor ext(iin[9:0], imm);

ula alu(a_reg_out, bus, alu_op_select, alu_out);

multiplexador mux(mux_select, imm, r0_out, r1_out, r2_out,
r3_out,
    r4_out, r5_out,r6_out, r7_out, alu_reg_out, bus);

logica_de_controle log_ctl(resetn, saida_contador, iin[15:7],
mux_select,
               regs_enable,   alu_op_select,   a_reg_enable,
alu_reg_enable,clear);

registrador r0(regs_enable[0], clock, bus, r0_out);
registrador r1(regs_enable[1], clock, bus, r1_out);
registrador r2(regs_enable[2], clock, bus, r2_out);
registrador r3(regs_enable[3], clock, bus, r3_out);
registrador r4(regs_enable[4], clock, bus, r4_out);
registrador r5(regs_enable[5], clock, bus, r5_out);
registrador r6(regs_enable[6], clock, bus, r6_out);
registrador r7(regs_enable[7], clock, bus, r7_out);
registrador a(a_reg_enable, clock, bus, a_reg_out);
registrador r(alu_reg_enable, clock, alu_out, alu_reg_out);

endmodule
```

3.0 - Testes

Após implementados todos os módulos do processador, sua funcionalidade foi verificada através de testes realizados em um testbench e visualizada através do GTKwave. Abaixo, seguem-se três exemplos de testes realizados.

3.1 - Inversão de sinal (utilizando-se NAND)

A seguinte sequência de instruções que foi realizada:

```
ldi r0, -3
ldi r1, 1
nan r0, r0
add r0, r1
out r0
```

Cuja representação em binário é:

```
processador p(clock, resetn, iin, bus);

initial begin
     # 0 resetn <= 1'b0;
     # 0 iin = 16'b101_000_11111111101;
     # 8 iin = 16'b101_001_0000000001;
     # 8 iin = 16'b010_000_000_0000000;
     # 8 iin = 16'b000_000_001_0000000;
     # 8 iin = 16'b100_000_000_0000000;
     # 8 $finish;
end
```

Esta sequência de instruções traduz-se para: 1) Carregar o valor (-3) no registrador 0; 2) Carregar o valor (1) no registrador 1; 3) Realizar a negação da conjunção do valor do registrador 0 com ele mesmo; 4) Somar o valor do registrador 0 com o valor do registrador 1; 5) Exibir o valor do registrador 0.

Note que a inversão é feita no sistema complemento de 2.

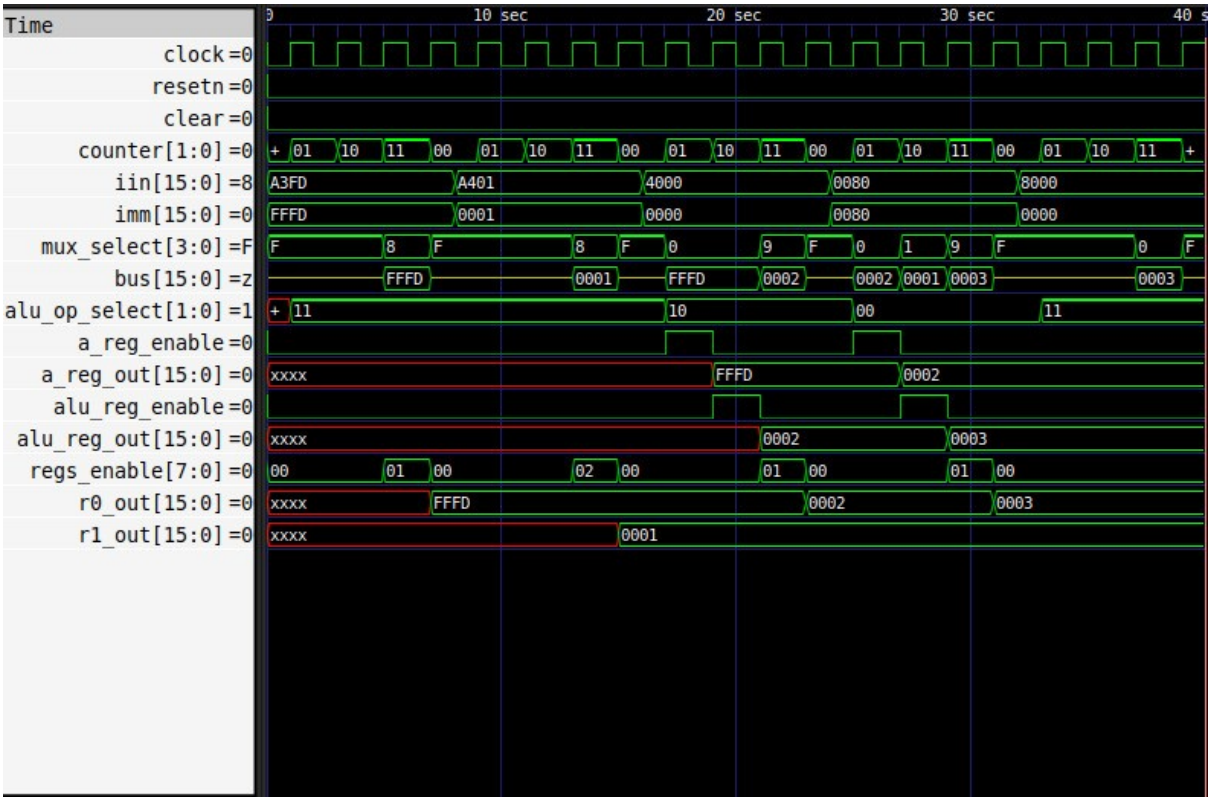

Imagem 8: Saída do primeiro teste no GTKwave.

3.2 - Inversão de sinal (utilizando-se subtração)

A seguinte sequência de instruções que foi realizada:

```
ldi r0, -3
ldi r1, 0
sub r1, r0
rep r2, r1
out r2
```

Cuja representação em binário é:

```
processador p(clock, resetn, iin, bus);

initial begin
     # 0 resetn <= 1’b0;
     # 0 iin = 16’b101_000_11111111101;
     # 8 iin = 16’b101_001_0000000000;
     # 8 iin = 16’b001_001_000_0000000;
     # 8 iin = 16’b111_010_001_0000000;
     # 8 iin = 16’b100_010_000_0000000;
     # 8 $finish;
end
```

Esta sequência de instruções traduz-se para: 1) Carregar o valor (-3) no registrador 0; 2) Carregar o valor (0) no registrador 1; 3) Subtrair o valor do registrador 1 pelo valor do registrador 0; 4) Copiar o valor do registrador 1 no registrador 2; 5) Exibir o valor do registrador 2.

Note que a inversão é feita no sistema complemento de 2 e que a instrução replace não é realmente necessária para a inversão de sinal, feita apenas a fim de teste da funcionalidade.

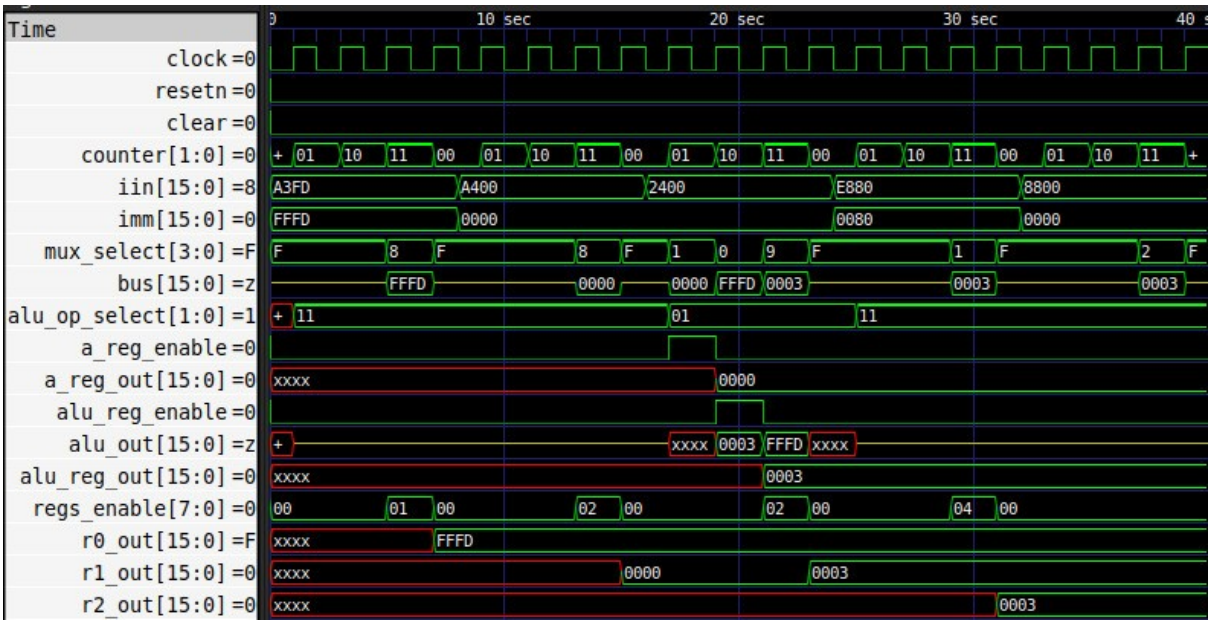

Imagem 9: Saída do segundo teste no GTKwave.

3.3 - Operação AND

A seguinte sequência de instruções que foi realizada:

```
ldi r4, 0xFFF0
ldi r5, 0x00FF
nan r4, r5
nan r4, r4
out r4
```

Cuja representação em binário é:

```
processador p(clock, resetn, iin, bus);

initial begin
     # 0 resetn <= 1’b0;
     # 0 iin = 16’b101_100_11111110000;
     # 8 iin = 16’b101_101_0011111111;
     # 8 iin = 16’b010_100_101_0000000;
     # 8 iin = 16’b010_100_100_0000000;
     # 8 iin = 16’b100_100_000_0000000;
     # 8 $finish;
end
```

Esta sequência de instruções traduz-se para: 1) Carregar o valor (FFF0) no registrador 4; 2) Carregar o valor (00FF) no registrador 5; 3) Realizar a negação da conjunção do valor do registrador 4 com o valor do registrador 5; 4) Realizar a negação da conjunção do valor do registrador 4 com ele mesmo; 5) Exibir o valor do registrador 4.

Note que os valores passados estão em hexadecimal.