

Prática 2

Erick Henrique
Marina Bernardes

07/2021

—
Laboratório de Arquitetura e
Organização de Computadores II

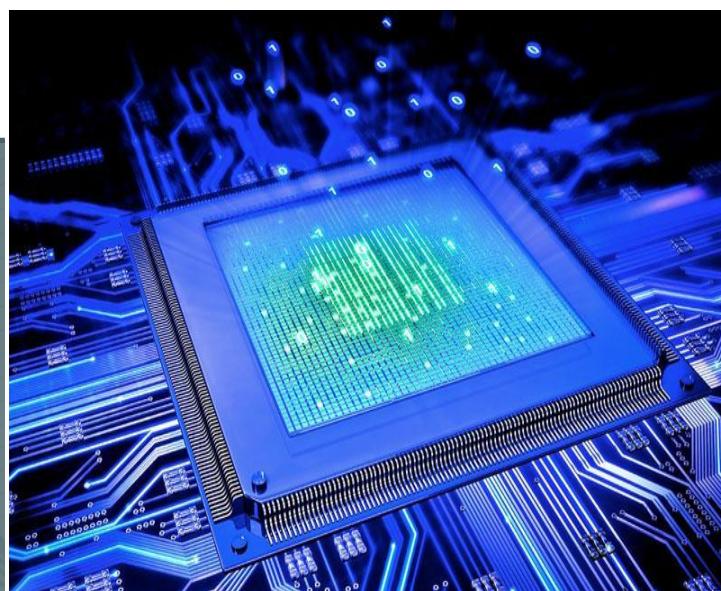
2021.1

—
Daniela Cristina Cascini Kupsch

Centro Federal de Educação Tecnológica de Minas
Gerais

OBJETIVOS

Esta prática tem a finalidade de exercitar os conceitos relacionados à implementação de um processador.



O PROCESSO

A prática 2 consistiu em projetar um processador simples capaz de realizar operações de memória e aritméticas.

O processador possui oito registradores listados em R0 a R7, os quais contém 16 bits. O R7, além de ser um registrador comum, é também um contador das instruções passado como parâmetro na biblioteca counterlpm. Há também dois registradores auxiliares para as operações de multiciclo, A e G.

O processador estava parcialmente projetado no arquivo disponibilizado nas instruções deste relatório, onde se encontra o datapath base que foi seguido para sua implementação, restando completá-lo com as demandas solicitadas.

Além disso, temos que, o bloco de instrução denominado “Addsub” foi substituído por “ULA”, em nossa implementação, que será responsável por realizar as operações aritméticas do processador.

O DataPath

Processador parcialmente implementado em material com as instruções disponibilizado para essa tarefa.

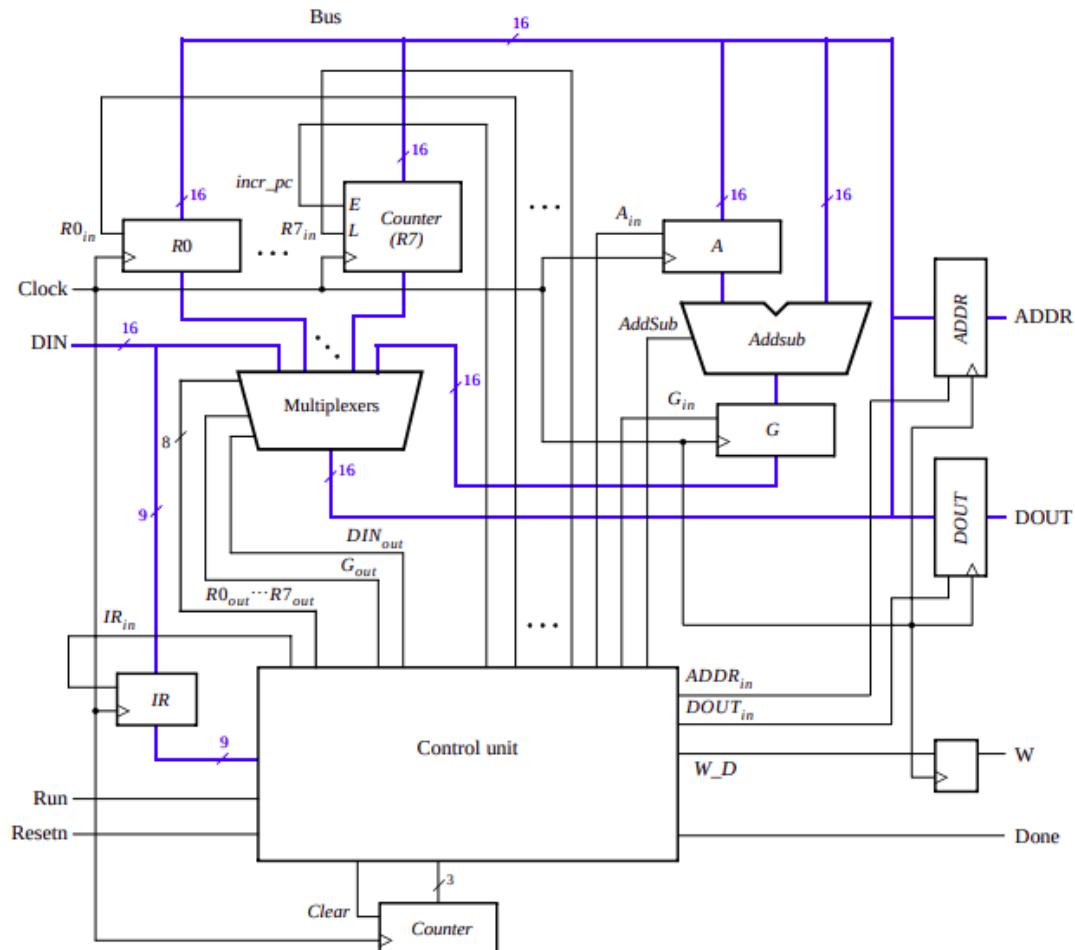


Figure 7. An enhanced version of the processor.

Sendo assim, tivemos que implementar suporte a novas instruções, como “*or*, *slt*, *sll*, *srl*”, além de implementar um módulo adicional denominado “*TLB*”.

Operações

Para iniciar a implementação do nosso processador, denominamos uma representação em binário para cada instrução que é listado a seguir:

| Operações | Significado | Bits |
|-------------|--|------|
| mv | move o conteúdo de um registrador Ry para o registrador Rx | 0ooo |
| st | armazena o valor de Rx em Ry | 0oo1 |
| mvnz | se Gin for diferente de 0, Rx armazena o valor de Ry | 0010 |
| ld | armazena o valor de Ry em Rx | 0011 |
| mvi | armazena o valor de um imediato D em Rx | 0100 |
| add | armazena o valor da soma de Rx e Ry em Rx | 0101 |
| sub | armazena o valor da subtração de Rx e Ry em Rx | 0110 |
| or | armazena o valor de Rx ou Ry em Rx | 0111 |
| slt | se Rx for menor que Ry, Rx armazena o valor 1, caso contrário, Rx armazena o valor 0 | 1000 |
| sll | Move ‘Ry’ bits para a esquerda de Rx | 1001 |
| srl | Move ‘Ry’ bits para a direita de Rx | 1010 |

Instructions Performed

As operações implementadas, estão listadas a seguir:

| Operation | Function Performed |
|--------------------|--------------------------------------|
| ld Rx,[Ry] | $Rx \leftarrow [[Ry]]$ |
| st Rx,[Ry] | $[Ry] \leftarrow [Rx]$ |
| mvnz Rx, Ry | $if G!=0, Rx \leftarrow [Ry]$ |
| mv Rx, Ry | $Rx \leftarrow [Ry]$ |
| mvi Rx, #D | $Rx \leftarrow D$ |
| add Rx, Ry | $Rx \leftarrow [Rx]+[Ry]$ |
| sub Rx, Ry | $Rx \leftarrow [Rx]-[Ry]$ |
| or Rx, Ry | $Rx \leftarrow [Rx] // [Ry]$ |
| slt Rx, Ry | $If(Rx < Ry) [Rx] = 1 else [Rx] = 0$ |
| sll Rx, Ry | $Rx = [Rx] << [Ry]$ |
| srl Rx, Ry | $Rx = [Rx] >> [Ry]$ |

Todo endereço enviado ao processador é um endereço virtual (endereço das instruções), por isso foi implementada uma TLB no estágio de IF (Instruction Fetch) totalmente associativo. Nele, as páginas virtuais contém 38 endereços sequenciais de 6 bits e as 38 páginas físicas contém as instruções em binário de 16 bits, do arquivo teste disponibilizado.

O acesso se deu por cada instrução, sempre verificando se o endereço das instruções bate com o endereço da página virtual, se isso ocorre, o conteúdo é buscado na página física.

Esse conteúdo é então mandado para o processador como “DIN”, que são as instruções representadas da seguinte forma:

| Instrução: | 16'b0100000000000000 | | | | |
|------------|------------------------------------|-----------|-----|-----|--------|
| | 16'b | 0100 | 000 | 000 | 000000 |
| | Representa os 16 bits da instrução | Instrução | Rx | Ry | offset |

Simultaneamente, a memória inicializada com um arquivo .mif contendo apenas o número ‘oooooooooooooo100’ na posição ‘3’, está sendo lida através da ramlpm e sua saída é passada para o processador, assim como os dados e o wren.

Control Signals - TimeStep

O processador atua em ciclos, de 0 a 3 à depender da instrução que está sendo executada. Sendo assim, o TimeStep refere qual o ciclo atual do processador, que irá delimitar qual e como as operações serão executadas.

| Instruction | TimeStep 1 | TimeStep 2 | TimeStep 3 |
|--------------------|---------------------------------------|---|---------------------------|
| mv Io: | <i>RXin, RYout, Done</i> | | |
| st I1: | <i>RXout, DOUTin</i> | <i>RYout, ADDRin, Win</i> | |
| mvnz Rx, Ry | <i>Done, if(G!=0) RXin, RYout</i> | | |
| ld Rx, Ry | <i>RYout, ADDRin</i> | | <i>RXin, MEMout, Done</i> |
| mvi Rx, #D | | <i>RXin, DINin, Done, REGout, INCR_PC</i> | |
| add Rx, Ry | <i>RXout, Ain</i> | <i>RYout, Gin, ULA(000)</i> | <i>RXin, Gout, Done</i> |
| sub Rx, Ry | <i>RXout, Ain</i> | <i>RYout, Gin, ULA(001)</i> | <i>RXin, Gout, Done</i> |
| or Rx, Ry | <i>RXout, Ain</i> | <i>RYout, Gin, ULA(010)</i> | <i>RXin, Gout, Done</i> |
| slt Rx, Ry | <i>RXout, Ain</i> | <i>RYout, Gin, ULA(011)</i> | <i>RXin, Gout, Done</i> |
| sll Rx, Ry | <i>RXout, Ain</i> | <i>RYout, Gin, ULA(100)</i> | <i>RXin, Gout, Done</i> |
| srl Rx, Ry | <i>RXout, Ain</i> | <i>RYout, Gin, ULA(101)</i> | <i>RXin, Gout, Done</i> |

Modules

Para a execução do processador, tivemos que desenvolver alguns módulos, outros já estavam implementados, nos quais cada um é responsável por realizar uma determinada tarefa. Listamos a seguir:

| Módulos | Função |
|------------------|---|
| counterlpm | Biblioteca LPM, contador |
| dec3to8 | Extensor de sinal, estendendo registradores de 3 bits para 8 bits |
| instructionFetch | Gera uma página virtual que será utilizada na TLB |
| MUX | Recebe dez entradas e maneja sua saída |
| pratica2 | Ligaçāo dos módulos |
| proc | É o método principal, que irá designar o caminho que cada instruāo irá percorrer durante sua execuāo |
| ramlpm | Biblioteca LPM, acesso à memória |
| regn | Módulo que tem a função de inicializar, resetar um registrador passado como parâmetro ou atribuí-lo a uma próxima instruāo. |
| romlpm | Biblioteca LPM, permite apenas à leitura, guarda o conteúdo das páginas virtuais, que foram escritos no arquivo romMIF.mif |
| TLB | Permite acesso aos casos testes |
| ULA | Armazena informações de como as operações aritméticas devem funcionar |
| upcount | Conta os estágios |

Simulação

- Caso Teste 1

- Dados

Para a simulação do processador implementado, utilizamos dois casos de testes listados a seguir, com sua implementação em binário:

| Caso Teste 1 | | | | | |
|-----------------------|-------------|-------|-------|----|----|
| Entrada | | Saída | | | |
| Representation | Instruction | Ro | R1 | R2 | R3 |
| 16'b0100000000000000; | MVI Ro, #2 | 0 | 0 | 0 | 0 |
| 16'b0000000000000010; | #2 | 2 | 0 | 0 | 0 |
| 16'b0100010000000000; | MVI R1, #3 | 2 | 0 | 0 | 0 |
| 16'b0000000000000011; | #3 | 2 | 3 | 0 | 0 |
| 16'b0101001000000000; | ADD R1,R0 | 2 | 5 | 0 | 0 |
| 16'b0100010000000000; | MVI R2, #6 | 2 | 5 | 0 | 0 |
| 16'b0000000000000010; | #6 | 2 | 5 | 6 | 0 |
| 16'b0110010001000000; | SUB R2, R1 | 2 | 5 | 1 | 0 |
| 16'b0000011001000000; | MV R3, R2 | 2 | 5 | 1 | 1 |
| 16'b0101000011000000; | ADD Ro, R3 | 3 | 5 | 1 | 1 |
| 16'b0110010000000000; | OR R1, Ro | 3 | 7 | 1 | 1 |
| 16'b0110001000000000; | SUB R1, Ro | 3 | 4 | 1 | 1 |
| 16'b0101001011000000; | ADD R1, R3 | 3 | 5 | 1 | 1 |
| 16'b1001001011000000; | SLL R1, R3 | 3 | A(10) | 1 | 1 |
| 16'b1010001011000000; | SRL R1, R3 | 3 | 5 | 1 | 1 |
| 16'b0100000000000000; | MVI Ro, #0 | 3 | 5 | 1 | 1 |
| 16'b0000000000000000; | #0 | 0 | 5 | 1 | 1 |
| 16'b1000000001000000; | SLT Ro, R1 | 1 | 5 | 1 | 1 |

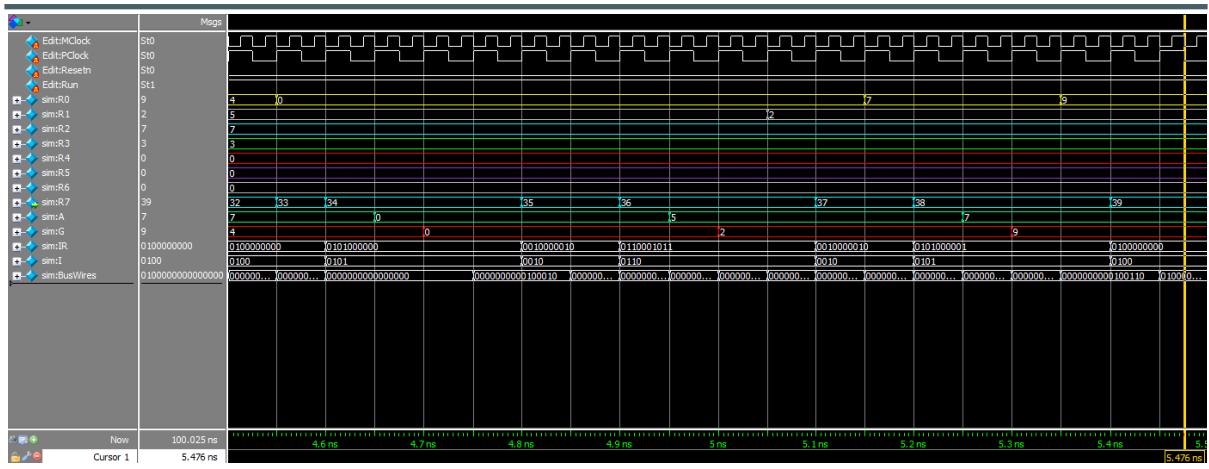
Caso Teste 1

| Entrada | | Saída | | | |
|----------------------------|--------------------------------------|----------------|----------------|----------------|----------------|
| Representation | Instruction | R ₀ | R ₁ | R ₂ | R ₃ |
| 16'b1000001001000000 | SLT R1, R1 | 1 | 0 | 1 | 1 |
| 16'b0100011000000000; | MVI R ₃ , #3 | 1 | 0 | 1 | 1 |
| 16'b0oooooooooooo000011; | #3 | 1 | 0 | 1 | 3 |
| 16'b0100010000000000; | MVI R ₁ , #5 | 1 | 5 | 1 | 3 |
| 16'b0oooooooooooo0000101; | #5 | 4 | 5 | 1 | 3 |
| 16'b0101000011000000; | ADD R ₀ , R ₃ | 4 | 5 | 1 | 3 |
| 16'b0100000000000000; | MVI R ₀ , #0 | 4 | 5 | 1 | 3 |
| 16'b0oooooooooooo00000000; | #0 | 0 | 5 | 1 | 3 |
| 16'b0011010011000000; | LD R ₂ , R ₃ | 0 | 5 | 4 | 3 |
| 16'b0101010011000000; | ADD R ₂ , R ₃ | 0 | 5 | 7 | 3 |
| 16'b0001010000000000; | SD R ₂ , R ₀ | 0 | 5 | 7 | 3 |
| 16'b0011000000000000; | LD R ₀ , R ₀ | 7 | 5 | 7 | 3 |
| 16'b0110000011000000; | SUB R ₀ , R ₃ | 4 | 5 | 7 | 3 |
| 16'b0100000000000000; | MVI R ₀ , #0 | 4 | 5 | 7 | 3 |
| 16'b0oooooooooooo00000000; | #0 | 0 | 5 | 7 | 3 |
| 16'b0101000000000000; | ADD R ₀ , R ₀ | 0 | 5 | 7 | 3 |
| 16'b0010000010000000; | MVNZ R ₀ , R ₂ | 0 | 5 | 7 | 3 |
| 16'b0110001011000000; | SUB R ₁ , R ₃ | 0 | 2 | 7 | 3 |
| 16'b0010000010000000; | MVNZ R ₀ , R ₂ | 7 | 2 | 7 | 3 |
| 16'b0101000001000000; | ADD R ₀ , R ₁ | 9 | 2 | 7 | 3 |

- Simulação

A partir da simulação do processador implementado em Verilog HDL, utilizamos o ModelSim para sua simulação, e o resultado é demonstrado da seguinte forma:





As variáveis contidas nestas simulações representam:

| Simulation | | |
|---|-----------------------------|---|
| Registradores | Inputs | Variáveis de confirmação de resultado |
| Ro,R1,R2,R3,R4,R5, R6,R7, A, G | MClock, PClock, Resetn, Run | IR, I, BusWires |
| Sendo que R7 é um counter. A e G são auxiliares | | Confirmam pelos seus resultados que a simulação ocorreu como o esperado |

Analizando a simulação executada por meio das suas ondas, percebemos que o algoritmo teste foi executado com sucesso, observando-se por meio dos registradores contidos nessa simulação.

- Caso Teste2

- Dados

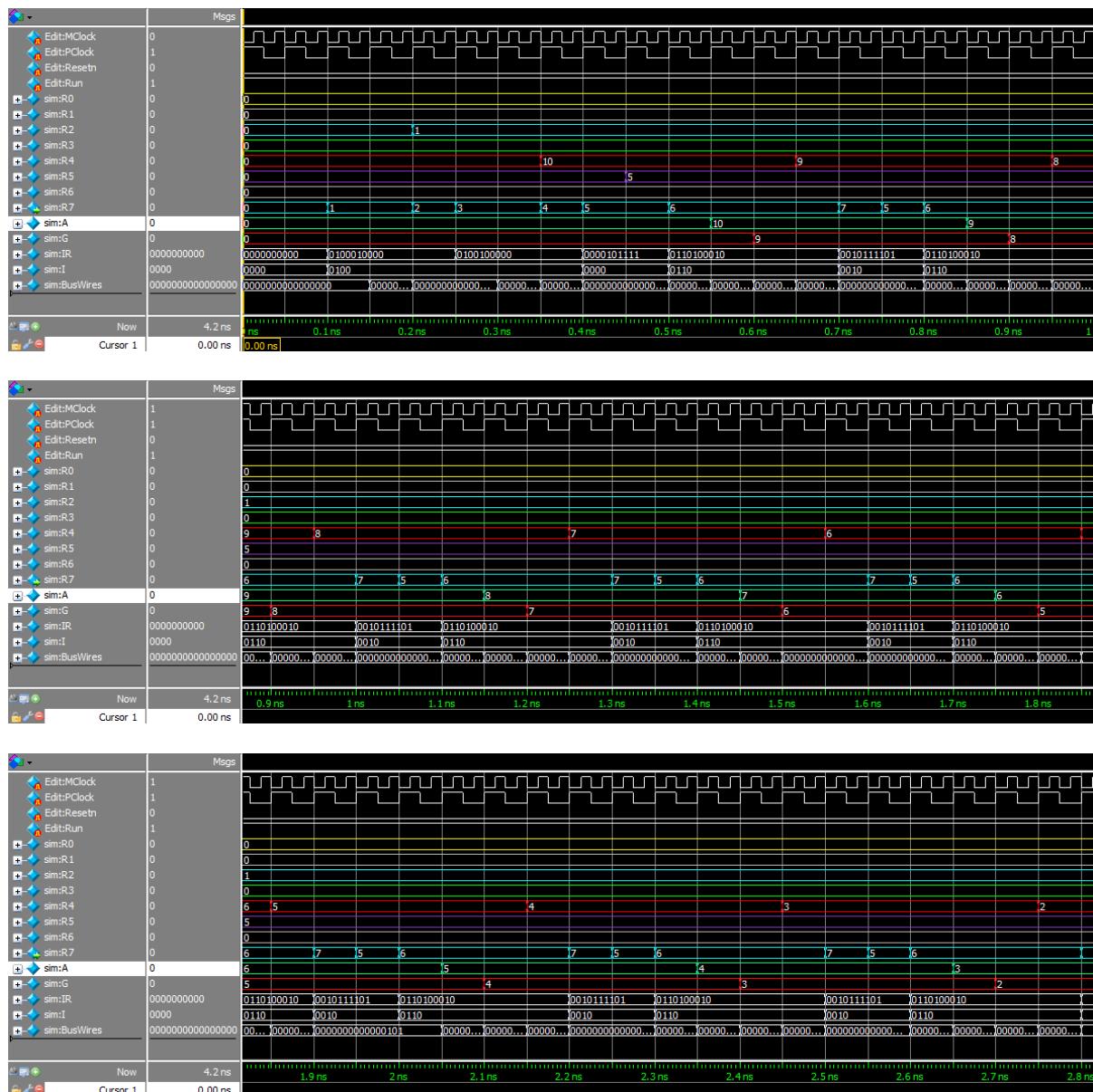
| Caso Teste 2 | | | | | |
|-----------------------|-------------|-------|----|----|---------|
| Entrada | | Saída | | | |
| Representation | Instruction | R2 | R4 | R5 | R7 |
| 16'b0100010000000000; | MVI R2, #1 | 0 | 0 | 0 | counter |
| 16'b0000000000000001; | #1 | 1 | 0 | 0 | counter |
| 16'b0100100000000000; | MVI R4, #10 | 1 | 0 | 0 | counter |
| 16'b0000000000001010; | #10 | 1 | 10 | 0 | counter |
| 16'b0000101111000000; | MV R5,R7 | 1 | 10 | 5 | counter |
| 16'b0110100010000000; | SUB R4,R2 | 1 | 9 | 5 | counter |
| 16'b0010111010000000; | MVNZ R7,R5 | 1 | 9 | 5 | counter |
| 16'b0110100010000000; | SUB R4,R2 | 1 | 8 | 5 | counter |
| 16'b0010111010000000; | MVNZ R7,R5 | 1 | 9 | 5 | counter |
| 16'b0110100010000000; | SUB R4,R2 | 1 | 7 | 5 | counter |
| 16'b0010111010000000; | MVNZ R7,R5 | 1 | 9 | 5 | counter |
| 16'b0110100010000000; | SUB R4,R2 | 1 | 6 | 5 | counter |
| 16'b0010111010000000; | MVNZ R7,R5 | 1 | 9 | 5 | counter |
| 16'b0110100010000000; | SUB R4,R2 | 1 | 5 | 5 | counter |
| 16'b0010111010000000; | MVNZ R7,R5 | 1 | 9 | 5 | counter |
| 16'b0110100010000000; | SUB R4,R2 | 1 | 3 | 5 | counter |
| 16'b0010111010000000; | MVNZ R7,R5 | 1 | 9 | 5 | counter |
| 16'b0110100010000000; | SUB R4,R2 | 1 | 2 | 5 | counter |
| 16'b0010111010000000; | MVNZ R7,R5 | 1 | 9 | 5 | counter |
| 16'b0110100010000000; | SUB R4,R2 | 1 | 1 | 5 | counter |
| 16'b0010111010000000; | MVNZ R7,R5 | 1 | 9 | 5 | counter |
| 16'b0110100010000000; | SUB R4,R2 | 1 | 0 | 5 | counter |

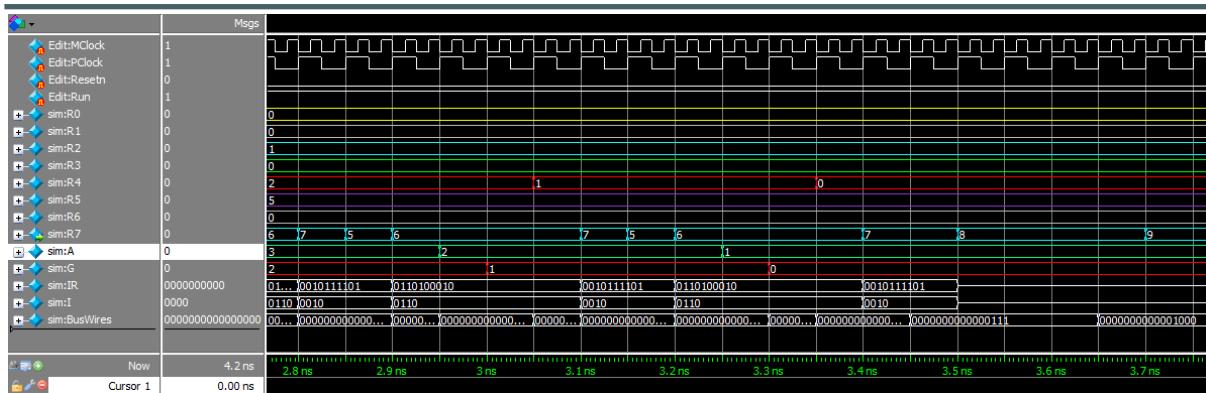
Caso Teste 2

| Entrada | | Saída | | | |
|----------------------|-------------|-------|----|----|---------|
| Representation | Instruction | R2 | R4 | R5 | R7 |
| 16'b001011101000000; | MVNZ R7,R5 | 1 | 9 | 5 | counter |

- Simulação

A partir da simulação do processador implementado em Verilog HDL, utilizamos o ModelSim para sua simulação, e o resultado é demonstrado da seguinte forma:





As variáveis contidas nestas simulações representam:

| Simulation | | |
|---|-----------------------------|---|
| Registradores | Inputs | Variáveis de confirmação de resultado |
| Ro,R1,R2,R3,R4,R5,R6,R7, A, G | MClock, PClock, Resetn, Run | IR, I, BusWires |
| Sendo que R7 é um counter. A e G são auxiliares | | Confirmam pelos seus resultados que a simulação ocorreu como o esperado |

Analisando a simulação executada por meio das suas ondas, percebemos que o algoritmo teste foi executado com sucesso, observando-se por meio dos registradores contidos nessa simulação.

Além disso, após a finalização da simulação, como não existe mais nenhuma instrução, o programa é encerrado.