Universidade Federal de Pernambuco

Relatório de projeto da disciplina Infraestrutura de Hardware

Professora: Edna Natividade da Silva Barros

Equipe: Douglas Brito (dba)
Filipe Medeiros (fcma)
George Alves (gvsa)
Igor Barbosa (iobs)
Lygia Helena (lhca)

Recife, 09 de Julho de 2014

Índice

1 INTRODUÇÃO

Partindo do aprendizado de arquitetura de processadores, foi possível desenvolver uma unidade central de processamento de 32 bits, baseada na arquitetura Multiciclo.

O presente relatório tem como objetivo elencar as etapas de construção de tal estrutura de hardware, baseando-se nos princípios da arquitetura MIPS, no qual foi projetada de acordo com a filosofia RISC (Reduced Instruction Set Computer, ou seja, computadores com conjunto de instruções reduzidas), conceitos estes adquiridos em sala de aula.

Com a implementação, buscou-se aprender mais sobre a arquitetura citada, observando vantagens e desvantagens. No entanto, para se chegar a um projeto consolidado de um processador faz-se necessário cumprimento de etapas como o projeto e a implementação dos seus componentes.

Nas sessões seguintes será descrito as etapas do desenvolvimento do processador acima citado, bem como os elementos que irão compô-lo, as entidades implementadas pelo grupo, o conjunto de instruções exigido, assim como a simulação de cada instrução em forma de figura (waveforms). A ferramenta utiliza para o desenvolvimento da unidade de processamento foi o QUARTUS II da Altera, utilizando a linguagem SystemVerilog.

2 DESCRIÇÃO DAS ENTIDADES IMPLEMENTADAS

Baseando-se na arquitetura do tipo MIPS, apresentada em sala de aula, foram implementados alguns componentes de hardware, para poder abranger todo o conjunto de instruções exigido no escopo do projeto. Tais entidades estão especificadas a seguir:

Entidade: Load

Esta entidade é responsável por ler um byte ou uma halfword da memória. São implementadas duas instruções a Load byte e Load half.

Entrada:

- Uma entrada de 32
- Load (1 bit): função a ser realizada pelo registrador (sinal de controle)

Saída:

Uma saída de 32 bits.

Objetivo:

Lê, um byte ou uma halfword da memória, salvando os oito bits (load byte) ou os 16 bits (load half) mais significativos em um registrador.

Algoritmo:

O algoritmo funciona da seguinte forma:

1. Load byte: os oito bits mais significativos do vetor de 32 bits são deslocados para direita e o restante do vetor é preenchido com zeros.

2. Load half: os 16 bits mais significativos do vetor de 32 bits são deslocados para a direita e o restante do vetor é preenchido com zeros.

Entidade: Store

Esta entidade é responsável por escrever na memória um byte ou uma halfword na memória. São implementadas duas instruções o Store Byte e o Store Half.

Entrada:

1. MDR (32bits):

2. B (32 bits):

3. Store (1 bit): função a ser realizada pelo registrador (sinal de controle).

Saída:

Vetor de 32 bits.

Objetivo:

Escrever na memória os 8 bits ou 16 bits menos significativos na memória.

Algoritmo:

O algoritmo funciona da seguinte forma:

1. Store byte: Os 8 bits do conteúdo do registrador (rt) são sobrescritos no vetor de endereço de memória.

2. Store Half: Os 16 bits do conteúdo do registrador (rt) são sobrescritos no vetor de endereço de memória.

Entidade Extend EPC

Estende o endereço de tratamento de exceção.

Entrada/Saída

1. Vetor de 32 bits

Objetivos:

Esta entidade é responsável por estender os oito bits mais significativos, preenchendo o restante do vetor com zeros.

Algoritmo:

O vetor de endereço vindo da memória de dados; os oito bits são deslocados para a direita e preenchendo o restante do vetor com zeros.

Entidade Shift

Nesta entidade foram implementadas cinco instruções, a saber:

1. Sll – shift left logical – Desloca os bits de um registrador para a esquerda, n vezes, e armazena o valor deslocado no reg. rd

2. Sllv – shift left logical by variable number of bits – realiza o deslocamento para a esquerda do registrador (rt), sendo que, o valor a ser deslocado é a quantidade de bits do registrador (rs) e armazena o resultado no registrador (rd)
3. Srl – shift right logical – Desloca os bits de um registrador para a direita, n vezes sem preservar o sinal, e armazena o valor deslocado em rd.
4. Srlv – shift riht logical by a variable number of bits - realiza o deslocamento para a direita do registrador (rt), sendo que, o valor a ser deslocado é a quantidade de bits do registrador (rs) e armazena o resultado no registrador (rd)
5. Sra – shift right arithmetic – Desloca os bits de um registrador para direita preservando o sinal, e armazena o valor deslocado em rd.
6. Srav - shift right arithmetic by a variable number of bits – desloca o registrador rt pela distância especificada no registrador rs e coloca o resultado no registrador rd

Entrada:

Duas entradas de 32 bits:

- Primeira entrada - vetor que sofre o deslocamento.
- Segunda entrada – vetor que indica a quantidade do deslocamento
  (usado nas instruções que utilizam o registrador rs para a quantidade de deslocamento)

Shift (3 bits): função a ser realizada pelo registrador. (sinal de controle)

Shamt (5 bits): quantidade de deslocamento

Saída:

Vetor de 32 bits deslocado.

Objetivo:

Esta unidade foi criada para ser capaz de efetuar as operações de shift a esquerda lógico e aritmético, shift a direita lógico e aritmético.

Algoritmo:
A funcionalidade desse registrador é especificada pela entrada Shift, que especifica a função e pela quantidade de deslocamento.

Entidade Sign Extented Lui

Esta unidade estende o sinal de 16 bits para 32 bits

Entrada:

Entrada de 16 bits: vetor que corresponde aos bits mais a esquerda do registrador.

Saída:

Saída de 32 bits: vetor que contém o resultado da operação

Objetivo:

Estender o vetor de entrada de 16 bits para 32 bits.

Algoritmo:

A instrução load upper immediate (lui) transfere o valor do campo de constante imediata de 16 bits para os 16 bits mais a esquerda do registrador, preenchendo os 16 bits de menor ordem (direita) com zeros.

Entidade EPC

Este registrador armazena o valor da instrução que gerou a exceção.

Saída/ Entrada:

Vetor de 32 bits.

Objetivo:
Salva o endereço da instrução problemática, e depois transfere o controle para o Sistema Operacional em alguns endereços especificados.

3 DESCRIÇÃO DAS OPERAÇÕES

O MIPS é o nome da arquitetura de processadores baseados no uso de registradores. As suas instruções têm à disposição um conjunto de 32 registradores para realizar as operações. Por esta arquitetura ser do tipo RISC, existe um conjunto bastante pequeno de instruções que o processador sabe fazer.

As instruções implementadas no desenvolvimento da unidade de processamento estão especificadas a seguir:

3.1 INSTRUÇÕES ARITMÉTICAS

Soma:

add rd, rs, rt – no estágio de decodificação o conteúdo do registrador rs e rt são carregados nos registradores de saída do banco de registradores. Um sinal de controle (ALUop) é enviado a ULA indicando que a operação a ser realizada é uma soma e o valor é armazenado em ALUout. Outro sinal de controle (Memtoreg) seleciona o resultado para ser entrada de dados no banco de registradores e o valor é gravado em rd. Uma nova instrução busca a memória. A soma pode gerar overflow

addi rt, rs, 4 – no estágio de decodificação o conteúdo do registrador rs é lido no banco de registradores. O valor lido do banco de registradores é somado aos 16 bits menos significativos da instrução com o sinal estendido. O resultado é armazenado no registrador rt do banco de registradores. A soma pode gerar overflow.

addu rd, rs, rt – no estágio de decodificação o conteúdo do registrador rs e rt são lidos no banco de registradores. A ALU realiza uma soma, de acordo com o sinal vindo da unidade de controle. O resultado é armazenado no registrador rd do banco de registradores. A soma não gera overflow.

addiu rt, rs,4 – no estágio de decodificação o conteúdo do registrador rs é lido no banco de registradores. A ALU realiza uma soma, de acordo com o sinal vindo da unidade de controle. O valor lido do banco de registradores é somado aos 16 bits menos significativos com o sinal estendido. O resultado é armazenado no registrador rt do banco de registradores. A soma não gera overflow.

Subtração:

sub rd, rs, rt - no estágio de decodificação o conteúdo do registrador rs e rt são carregados nos registradores de saída do banco de registradores. Um sinal de controle (ALUop) é enviado a ULA indicando que a operação a ser realizada é uma subtração e o valor é armazenado em ALUout. Outro sinal de controle (Memtoreg) seleciona o resultado para ser entrada de dados no banco de registradores e o valor é gravado em rd. A subtração pode gerar overflow.

subu rd, rs, rt – no estágio de decodificação o conteúdo do registrador rs e rt são lidos no banco de registradores. A ALU realiza uma subtração, de acordo com o sinal vindo da unidade de controle. O resultado é armazenado no registrador rd do banco de registradores. A subtração não gera overflow.

3.2 Instruções Lógicas/Shift

and rd, rs, rt - no estágio de decodificação dos registradores, o conteúdo dos registradores rs e rt são lidos do banco de registradores. Um sinal da unidade de controle é enviado a ALU indicando que a operação a ser realizada é um **and bit a bit.** O resultado é armazenado no registrador rd no banco de registradores.

andi rt, rs, 2 - no estágio de decodificação o conteúdo do registrador rs é lido no banco de registradores. Com o valor lido do banco de registradores, é realizado um **and bit a bit** com os 16 bits menos significativos com o sinal estendido. O resultado é armazenado no registrador rt do banco de registradores.

xor rd, rs, rt - no estágio de decodificação dos registradores, o conteúdo dos registradores rs e rt são lidos do banco de registradores. Um sinal da unidade de controle é enviado a ALU indicando que a operação a ser realizada é um **xor bit a bit.** O resultado é armazenado no registrador rd no banco de registradores.

xori rt, rs, 2 - no estágio de decodificação o conteúdo do registrador rs é lido no banco de registradores. Com o valor lido do banco de registradores, é realizado um **xor bit a bit** com os 16 bits menos significativos com o sinal estendido. O resultado é armazenado no registrador rt do banco de registradores.

sll rd, rt, shamt - No estágio de decodificação, o valor do registrador rt é carregado a partir do banco de registradores. A unidade de controle envia um sinal para a entidade shift, indicando que a operação sll será executada. Esta instrução seleciona a entrada do registrador rt e o shamt (quantidade de deslocamento) e os bits de uma word são deslocados para a esquerda, preenchendo os bits que ficaram vazios com zeros. O resultado é armazenado no registrador rd e pode gerar Overflow.

sllv rd, rt, shamt - No estágio de decodificação, o valor do registrador rt é carregado a partir do banco de registradores. A unidade de controle envia um sinal para a entidade shift, indicando que a operação sllv será executada. Esta instrução seleciona a entrada do registrador B e o shamt (quantidade de deslocamento) e os bits de uma word são deslocados para a esquerda, preenchendo os bits que ficaram vazios com zeros. O resultado é armazenado no registrador rd e não pode gerar Overflow.

srl rd, rt, shamt – No estágio de decodificação, o valor do registrador rt é carregado a partir do banco de registradores. A unidade de controle envia um sinal para a entidade shift, indicando que a operação srl será executada. Esta instrução seleciona a entrada do registrador rt e o shamt (quantidade de deslocamento) e os bits de uma word são deslocados para a direita, preenchendo os bits que ficaram vazios com zeros. O resultado é armazenado no registrador rd. A instrução pode gerar Overflow.

sra rd, rt, shamt – No estágio de decodificação, o valor do registrador rt é carregado a partir do banco de registradores. A unidade de controle envia um sinal para a entidade shift, indicando que a operação sra será executada. Esta instrução seleciona a entrada do registrador rt e o shamt (quantidade de deslocamento) e os bits de uma word são deslocados para a direita, preservando-se o sinal e preenchendo os bits que ficaram vazios com zeros (ou 1, se negativo). O resultado é armazenado no registrador rd e pode gerar Overflow.

srav rd, rt, shamt – No estágio de decodificação, o valor do registrador rt é carregado a partir do banco de registradores. A unidade de controle envia um sinal para a entidade shift, indicando que a operação sra será executada. Esta instrução seleciona a entrada do registrador rt e o shamt (quantidade de deslocamento) e os bits de uma word são deslocados para a direita, preservando-se o sinal e preenchendo os bits que ficaram vazios com zeros (ou 1, se negativo). O resultado é armazenado no registrador rd e não pode gerar Overflow.

3.3 INSTRUÇÕES DE TOMADA DE DECISÃO (BRANCH)

CONDICIONAIS:

Beq rs, rt, offset – No estágio de decodificação o offset é estendido e deslocado duas vezes à esquerda para poder somar com o valor de PC + 4. Também no estágio de decodificação é carregado do banco de registradores os valores de rs e rt, enquanto o valor da soma do offset com PC+4 é carregado no registrador ALUout. Após isso, é feita a comparação entre rs e rt e se os registradores forem iguais o sinal de "zero" (isto é, a ALU realiza uma subtração entre os registradores) da ALU é acionado e então o desvio é tomado.

bne rs, rt, offset – No estágio de decodificação o offset é estendido e deslocado duas vezes à esquerda para poder somar com o valor de PC + 4. Também no estágio de decodificação é carregado do banco de registradores os valores de rs e rt, enquanto o valor da soma do offset com PC+4 é carregado no registrador ALUout. Após isso, é feita a comparação entre rs e rt e se os registradores não forem iguais o sinal de "zero" (isto é, a ALU realiza uma subtração entre os registradores) da ALU não será acionado e então o desvio é tomado.

slt rd, rs, rt – No estágio de decodificação os valores do registrador rs e rt são carregados a partir do banco de registradores. A unidade de controle mandará um sinal para ALU indicando que uma instrução de comparação será feita. Se o valor do registrador rs for menor que o valor do registrador rt, será armazenado o valor 1 no registrador rd, caso contrário, será armazenado o valor zero.

slti rt, rs, 4 – No estágio de decodificação, o valor do registrador rs é carregado a partir do banco de registradores. Os 16 bits menos significativos da instrução (imediato) será estendido para 32 bits. A unidade de controle mandará um sinal para ALU indicando que uma instrução de comparação será feita. Se o valor do registrador rs for menor que o valor do que o valor do imediato, será armazenado o valor 1 no registrador rd, caso contrário, será armazenado o valor zero.

INCONDICIONAIS:

J offset – Após identificar a instrução jump no estágio de decodificação, os 26 bits menos significativos são deslocados de 2 a esquerda (28 bits) e concatenados com os 4 bits mais significativos do (PC + 4). Esse novo endereço é colocado no PC.

Jr rs – Após identificar a instrução jr no estágio de decodificação, o valor do registrador rs é carregado a partir do banco de registradores. A partir de um sinal do controle, o PC recebe o valor que estava no registrador rs.

Jal – O valor de PC + 4 é salvo no registrador 31, então os26 bits menos significativos da instrução será deslocado duas vezes para a esquerda e concatenado com PC + 4 para obter o novo endereço para PC.

3.4 Instruções de Transferência de Dados

lw rt, address (rs) - No estágio de decodificação a instrução carrega o registrador rs para poder somar com o imediato (16 bits) estendido e deslocado duas vezes à esquerda. Esta instrução calcula o endereço da memória somando o registrador rs com o imediato através da ALU. O valor dessa soma é carregado no registrador de saída da ALU (ALUout), então é usada como endereço na entrada da memória. Após dois ciclos de clock, o dado da memória é salvo no registrador de saída da memória (MDR) para só depois poder ser escrito no banco de registradores.

lb rt, address(rs) - No estágio de decodificação a instrução carrega o registrador rs para poder somar com o imediato (16 bits) estendido e deslocado duas vezes à esquerda. Esta instrução calcula o endereço da memória somando o registrador rs com o imediato através da ALU. O valor dessa soma é carregado no registrador de saída da ALU (ALUout), então é usada como endereço na entrada da memória. Após dois ciclos de clock, o dado da memória é salvo no registrador de saída da memória (MDR). Após passar pelo MDR, o dado vai para a entidade do Load e é selecionado apenas os 8 bits mais significativos para só depois poder ser escrito no banco de registradores.

lh rt, address (rs) - No estágio de decodificação a instrução carrega o registrador rs para poder somar com o imediato (16 bits) estendido e deslocado duas vezes à esquerda. Esta instrução calcula o endereço da memória somando o registrador rs com o imediato através da ALU. O valor dessa soma é carregado no registrador de saída da ALU (ALUout), então é usada como endereço na entrada da memória. Após dois ciclos de clock, o dado da memória é salvo no registrador de saída da memória (MDR). Após passar pelo MDR, o dado vai para a entidade do Load e é selecionado apenas os 16 bits mais significativos para só depois poder ser escrito no banco de registradores.

sw rt, adress (rs) - No estágio de decodificação a instrução carrega os registradores rs e rt para poder somar rs com o imediato (16 bits) estendido e deslocado duas vezes à esquerda. Esta instrução calcula o endereço da memória de onde o dado será armazenado somando o registrador rs com o imediato através da ALU. O valor dessa soma é carregado no registrador de saída da ALU (ALUout), então é usada como endereço na entrada da memória. Feito isso, o dado do registrador rt é usado como entrada de dado na memória para poder escrever.

sb rt, address (rs) - No estágio de decodificação a instrução carrega os registradores rs e rt para poder somar rs com o imediato (16 bits) estendido e deslocado duas vezes à esquerda. Esta instrução calcula o endereço da memória de onde o dado será armazenado somando o registrador rs com o imediato através da ALU. O valor dessa soma é carregado no registrador de saída da ALU (ALUout), então é usada como endereço na entrada da memória. Feito isso, será carregado da memória o dado onde será armazenado, então passa pela entidade do Store e então o dado da memória é sobrescrito apenas o byte necessário para poder escrever de volta na memória.

sh rt, address (rs) - No estágio de decodificação a instrução carrega os registradores rs e rt para poder somar rs com o imediato (16 bits) estendido e deslocado duas vezes à esquerda. Esta instrução calcula o endereço da memória de onde o dado será armazenado somando o registrador rs com o imediato através da ALU. O valor dessa soma é carregado no registrador de saída da ALU (ALUout), então é usada como endereço na entrada da memória. Feito isso, será carregado da memória o dado onde será armazenado, então passa pela entidade do Store e então o dado da memória é sobrescrito apenas os 2 bytes necessário para poder escrever de volta na memória.

lui rt, imm – O imediato (16 bits) recebido da instrução será estendido, utilizando a entidade **Sign extend lui,** para 32 bits. Então é colocados zeros (ou um, se for negativo) na parte menos significativa do vetor.

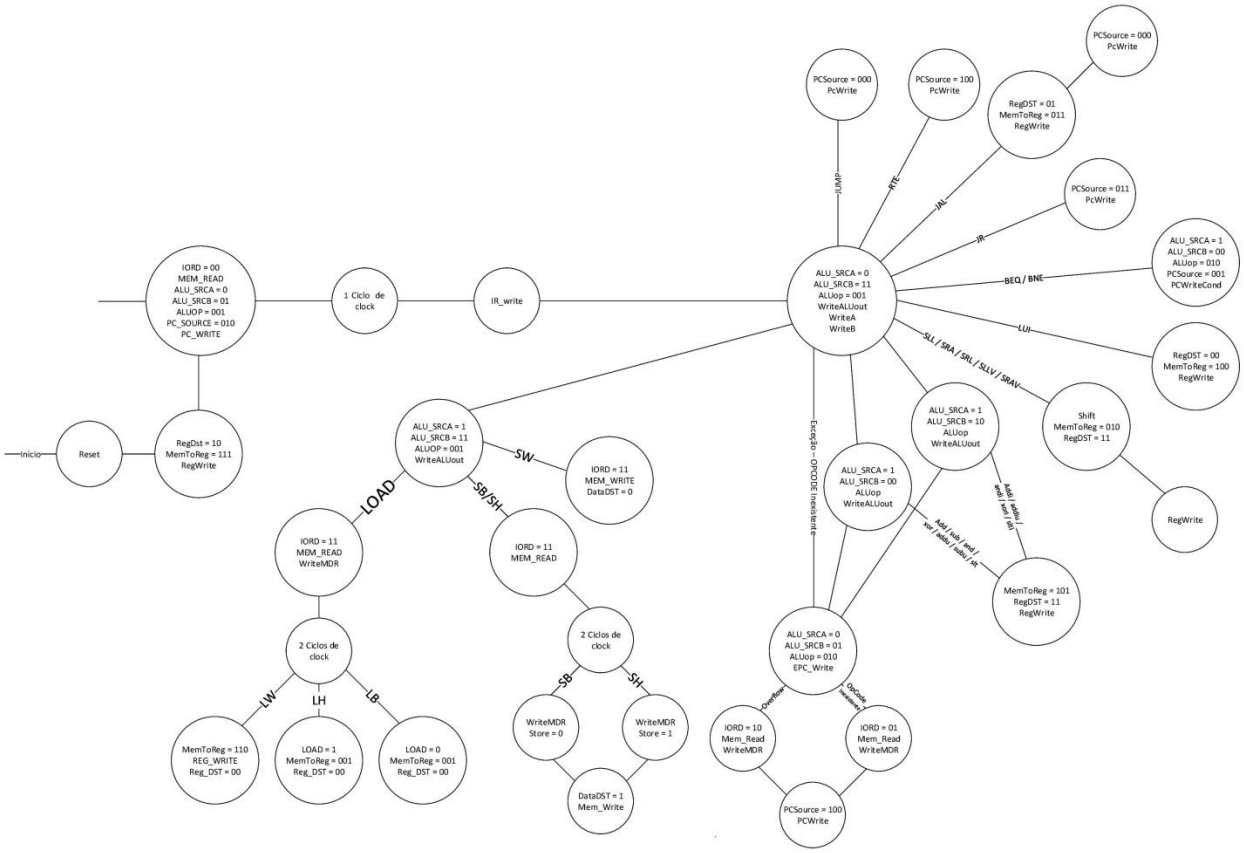
3.5 INSTRUÇÕES ESPECIAIS

break - Após identificar a instrução break no estágio de decodificação, há uma parada abrupta na execução do programa colocando o processador em um estado de inatividade contínua, do qual se retira apenas pelo reinicio das atividades de todo o processador.

rte – Essa intrução faz com que o programa volte a executar, após uma exceção, exatamente do ponto de onde ocorreu a exceção. (endereço está armazenado em EPC).

nop – Após identificar a instrução nop, o processador busca uma nova instrução. A instrução é usada apenas para gastar ciclos de clock.

4 DESCRIÇÃO DOS ESTADOS DO CONTROLE



RESET

Todos os registradores são zerados e o registrador que aponta para a pilha será inicializado.

ESTADO INICIAL

PC aponta para o endereço de memória que contém a instrução. É feita a busca na memória enquanto paralelamente pc é somado com 4 para poder buscar a próxima instrução.

IR_WRITE

Após 1 ciclo de espera, a instrução que foi buscada é escrita nos registrador.

DECODIFICAÇÃO

Decodifica a instrução e carrega nos registradores e espera o próximo ciclo para executar a instrução de acordo com seu opcode. Nesse estado também é calculado o endereço de instrução, caso a instrução seja um beq ou bne.

ESTADO COMUM LOAD/STORE

O registrador A é preparado para ser carregado na ALU e executar o cálculo do endereço junto com o deslocamento.

ESTADO COMUM LOAD

A memória é lida, de acordo com o endereço calculado, e o dado é levado para o MDR e em seguida 2 ciclos de espera são executados.

Load Word (LW)

O dado que estava na memória é carregado no banco de registradores.

Load Half (LH) / Load Byte (LB)

O dado que estava na memória passa pelo módulo “LOAD” para ‘tratamento’ e em seguida carregado no banco de registradores.

Store Word (SW)

Após o “ESTADO COMUM LOAD/STORE”, o endereço que foi calculado é usado para escrever o dado que estava no registrador, na memória.

Store Half (SH) / Store Byte (SB)

Após o “ESTADO COMUM LOAD/STORE”, o dado é buscado no banco de registradores, de acordo com o endereço calculado, passa pelo módulo “STORE” para ‘tratamento’ do dado e em seguida é carregado na memória.

Tipo-R

Registradores A e B são carregados na ALU e a operação é feita de acordo com o opcode e em seguida escrito na ALUout.

Tipo-I

Registradores A é carregado na ALU, enquanto o registrador B espera o dado a ser carregado ser estendido para 32 bits. Após isto, o valor calculado pela ALU é escrito no registrador ALUout.

WRITE Tipo-I / Tipo-R

O dado que está em ALUout é escrito no banco de registradores.

EPC Write

Se houver um opcode inexistente ou um overflow durante a execução das operações “Tipo-R” ou “Tipo-I” o módulo ‘EPC’ receberá um sinal de escrita, em seguida o módulo realizará o processo de acordo com a exceção.

EPC Overflow

Neste caso, o tratamento para overflow é buscado na posição 255 da memória, é lido e em seguida escrito na MDR.

EPC Opcode

O caso de opcode segue o mesmo padrão do ‘overflow’, sendo que o endereço de tratamento buscadona memória é o 254, que é lido e em seguida escrito na MDR.

End exceção

O endereço da instrução que ocorreu a exceção é escrito no registrador do PC.

Shift

Após o ‘Estado Inicial’, o módulo ‘Shift’ executa as operações de rotação de acordo com o opcode e em seguida é escrita no banco de registradores.

LUI

Nesta operação, o signal de 16 bits é estendido pelo módulo ‘lui extended signal’ preenchendo os 16 bits, mais significativos, restantes com zero e em seguida escrito no banco de registradores.

BEQ/BNE

O conteúdo nos registradores da ALU são comparados, e após isto é verificado sinal da ‘Flag Zero’. Caso seja zero, o ‘Branch’ é executado somando o endereço atual com o valor da ‘label’.

JR

Após execução do “Estado inicial”, o endereço desejado é carregado no registrador de PC para retorno.

JAL

Após pular para endereço calculado, o endereço de retorno é carregado no registrador $31 e em outro ciclo este endereço de retorno é escrito em PC.

RTE

O endereço calculado na ALU é retornado e escrito no PC.

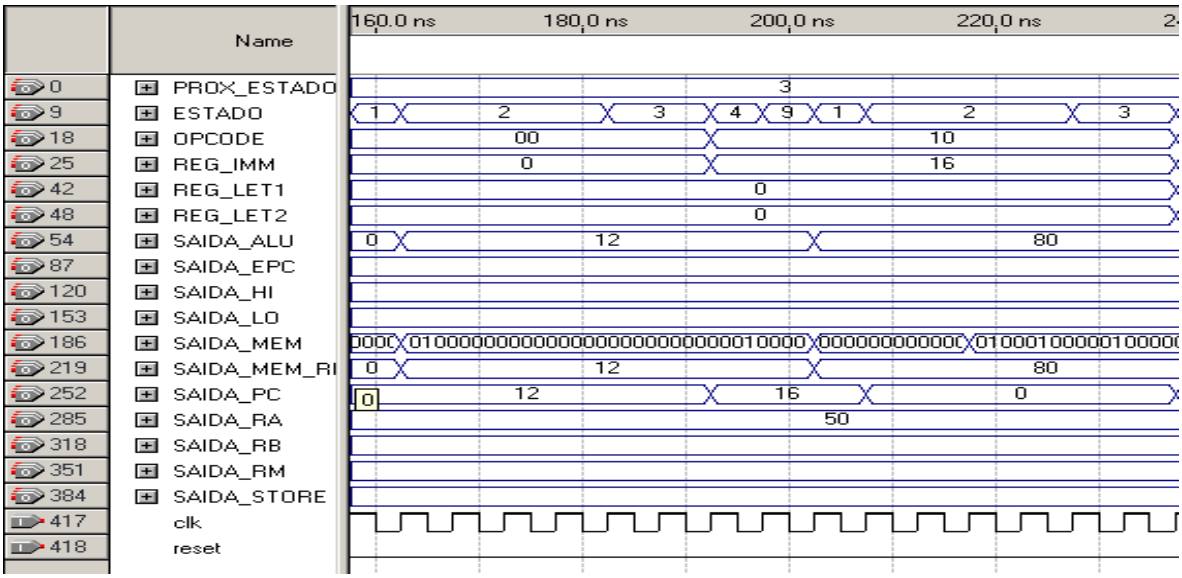
JUMP

Pula para o endereço calculado, escrevendo-o no PC.

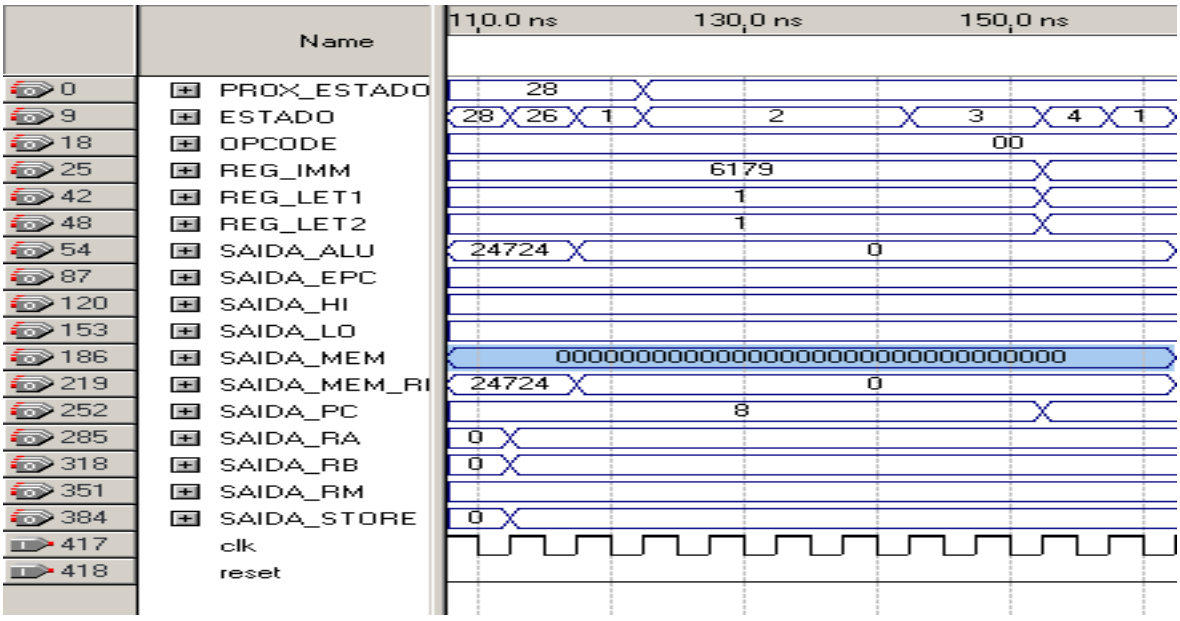
5 CONJUNTO DE SIMULAÇÕES

1. Instruções Tipo-R

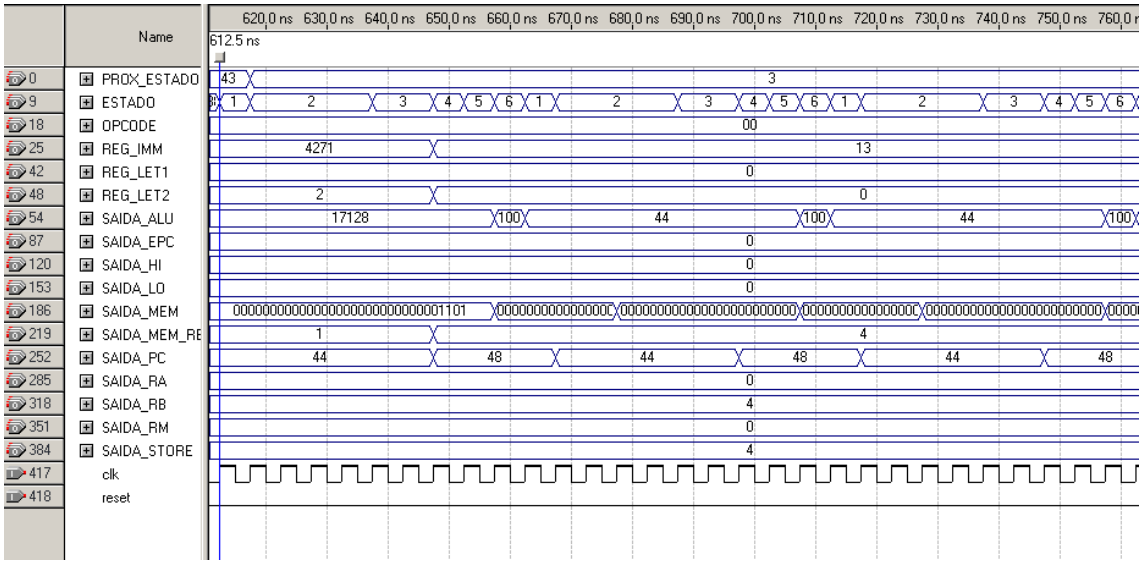
- rte:



- nop:

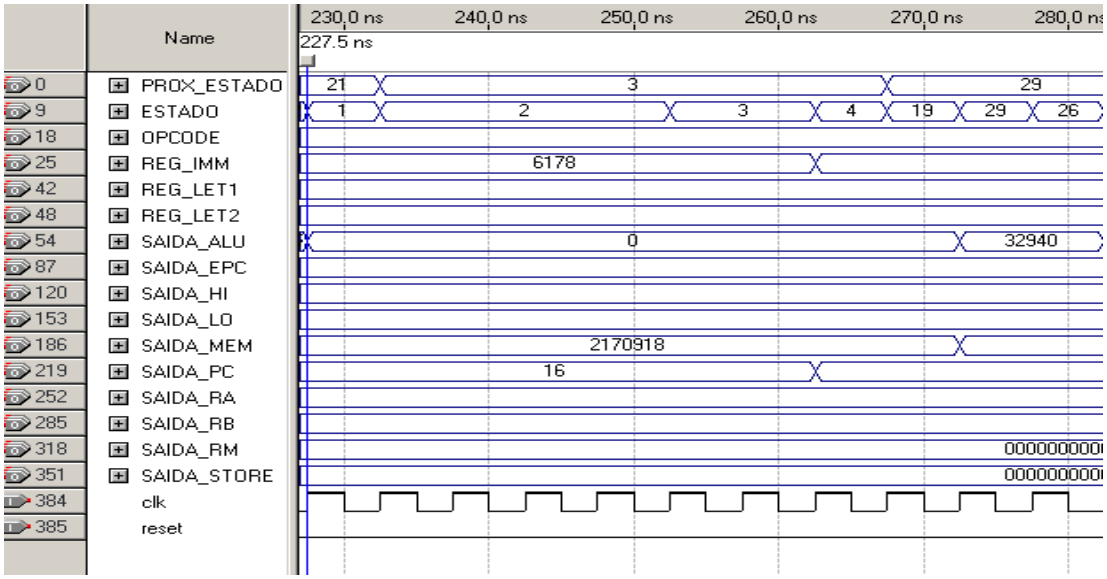


- break:

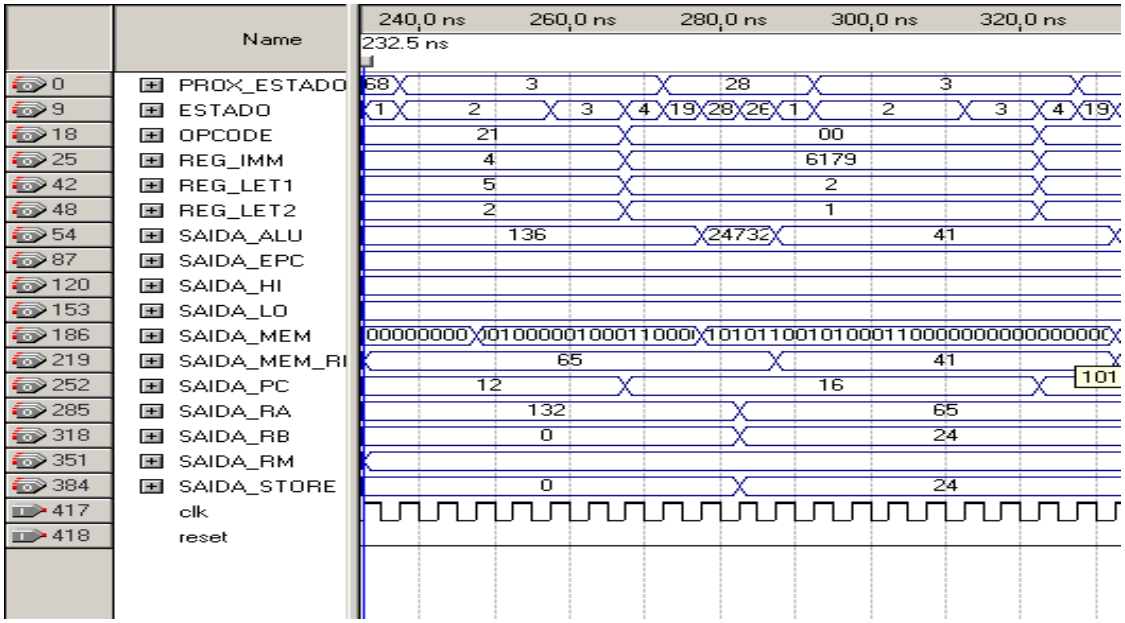


Como pode ser visto, ao executar o ‘break’, a instrução entra em um loop infinito.

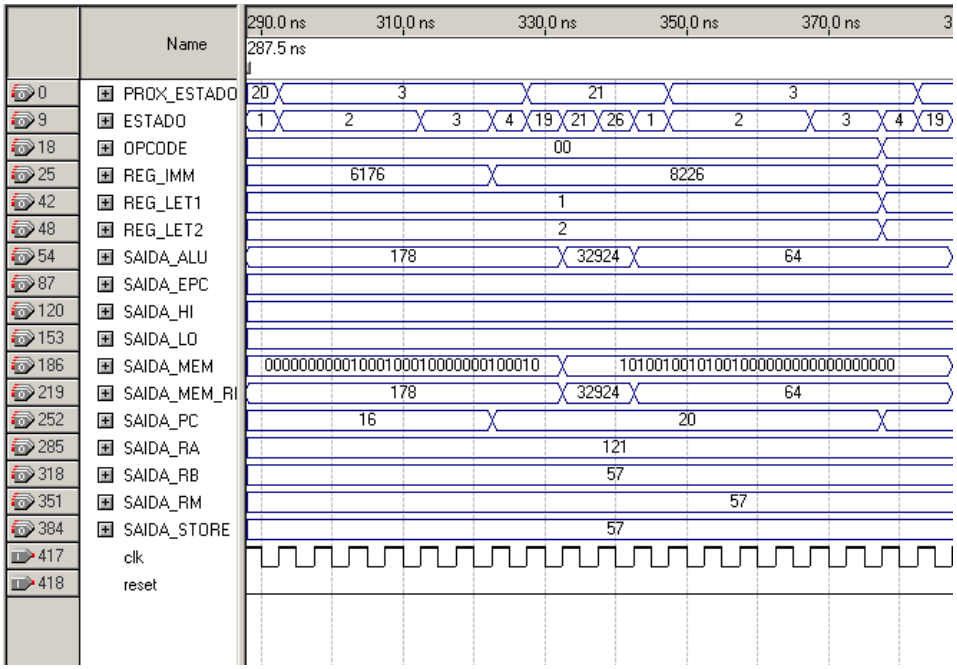
- xor:



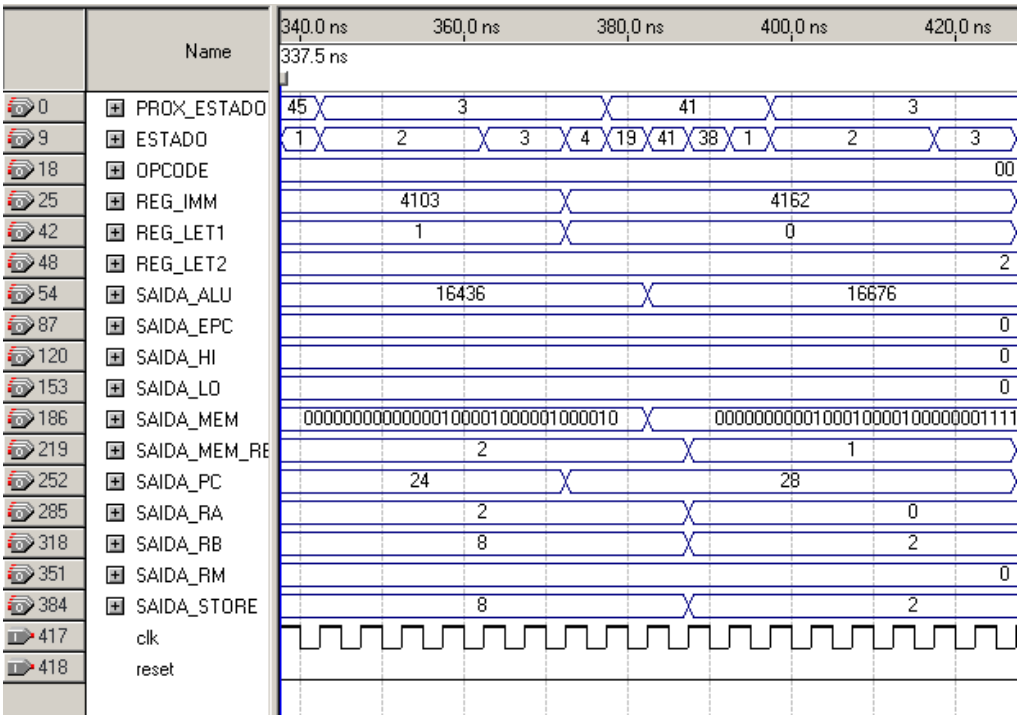
- subu;


Name
240,0 ns
260,0 ns
280,0 ns
300,0 ns
320,0 ns
232.5 ns
0 PROX_ESTADO
9 ESTADO
18 OPCODE
25 REG_IMM
42 REG_LET1
48 REG_LET2
54 SAIDA_ALU
87 SAIDA_EPC
120 SAIDA_HI
153 SAIDA_LO
186 SAIDA_MEM
219 SAIDA_MEM_RI
252 SAIDA_PC
285 SAIDA_RA
318 SAIDA_RB
351 SAIDA_RM
384 SAIDA_STORE
417 clk
418 reset

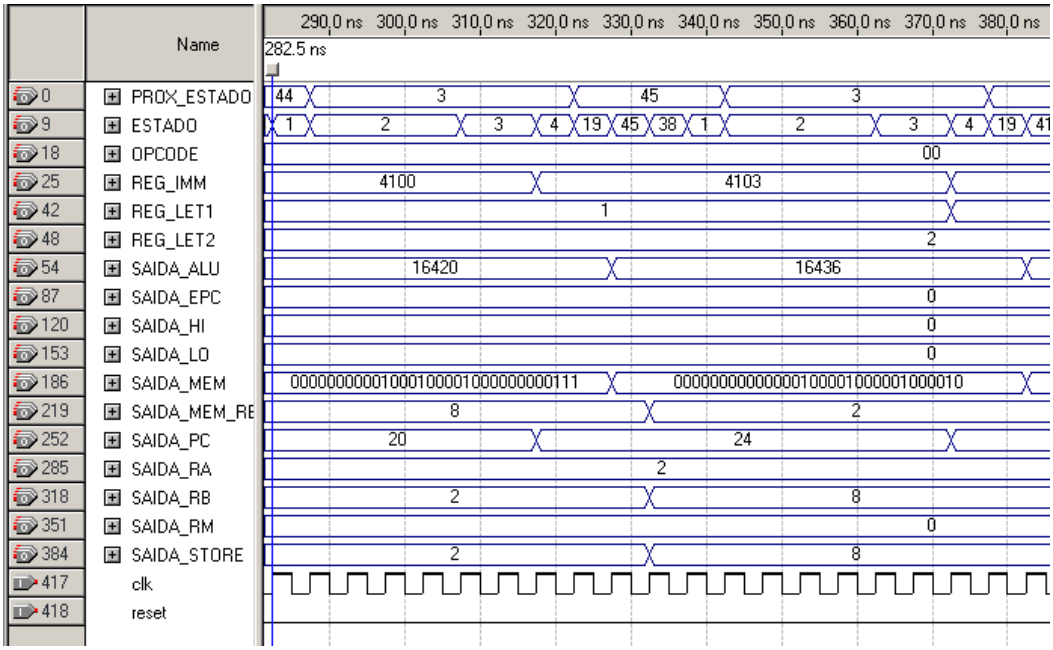
- sub:


Name
290.0 ns
310,0 ns
330,0 ns
350,0 ns
370,0 ns
287.5 ns
0 PROX_ESTADO
9 ESTADO
18 OPCODE
25 REG_IMM
42 REG_LET1
48 REG_LET2
54 SAIDA_ALU
87 SAIDA_EPC
120 SAIDA_HI
153 SAIDA_LO
186 SAIDA_MEM
219 SAIDA_MEM_RI
252 SAIDA_PC
285 SAIDA_RA
318 SAIDA_RB
351 SAIDA_RM
384 SAIDA_STORE
417 clk
418 reset

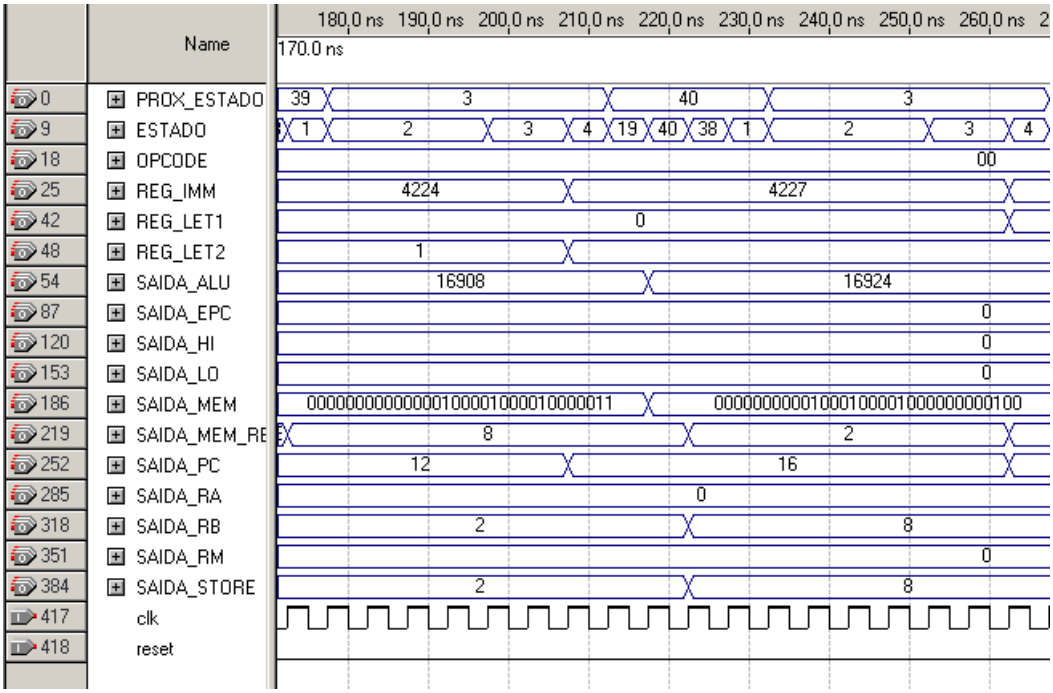
- srl:


Name
340,0 ns
360,0 ns
380,0 ns
400,0 ns
420,0 ns
337.5 ns
0 PROX_ESTADO
9 ESTADO
18 OPCODE
25 REG_IMM
42 REG_LET1
48 REG_LET2
54 SAIDA_ALU
87 SAIDA_EPC
120 SAIDA_HI
153 SAIDA_LO
186 SAIDA_MEM
219 SAIDA_MEM_RE
252 SAIDA_PC
285 SAIDA_RA
318 SAIDA_RB
351 SAIDA_RM
384 SAIDA_STORE
417 clk
418 reset

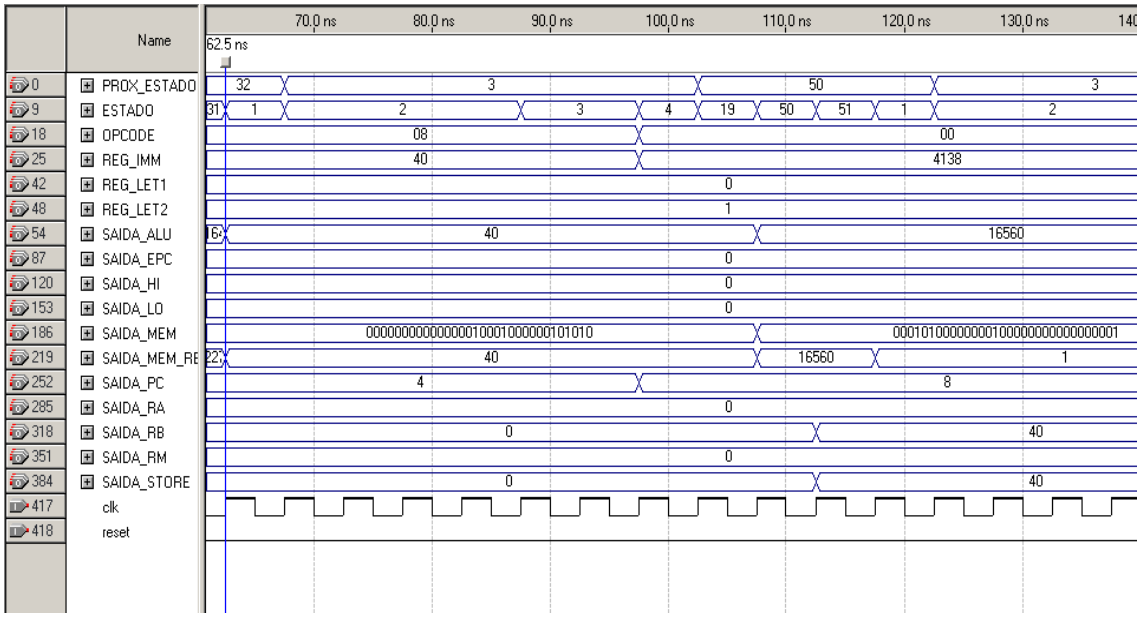
- srav:


Name
290,0 ns
300,0 ns
310,0 ns
320,0 ns
330,0 ns
340,0 ns
350,0 ns
360,0 ns
370,0 ns
380,0 ns
282.5 ns
0 PROX_ESTADO
9 ESTADO
18 OPCODE
25 REG_IMM
42 REG_LET1
48 REG_LET2
54 SAIDA_ALU
87 SAIDA_EPC
120 SAIDA_HI
153 SAIDA_LO
186 SAIDA_MEM
219 SAIDA_MEM_RE
252 SAIDA_PC
285 SAIDA_RA
318 SAIDA_RB
351 SAIDA_RM
384 SAIDA_STORE
417 clk
418 reset

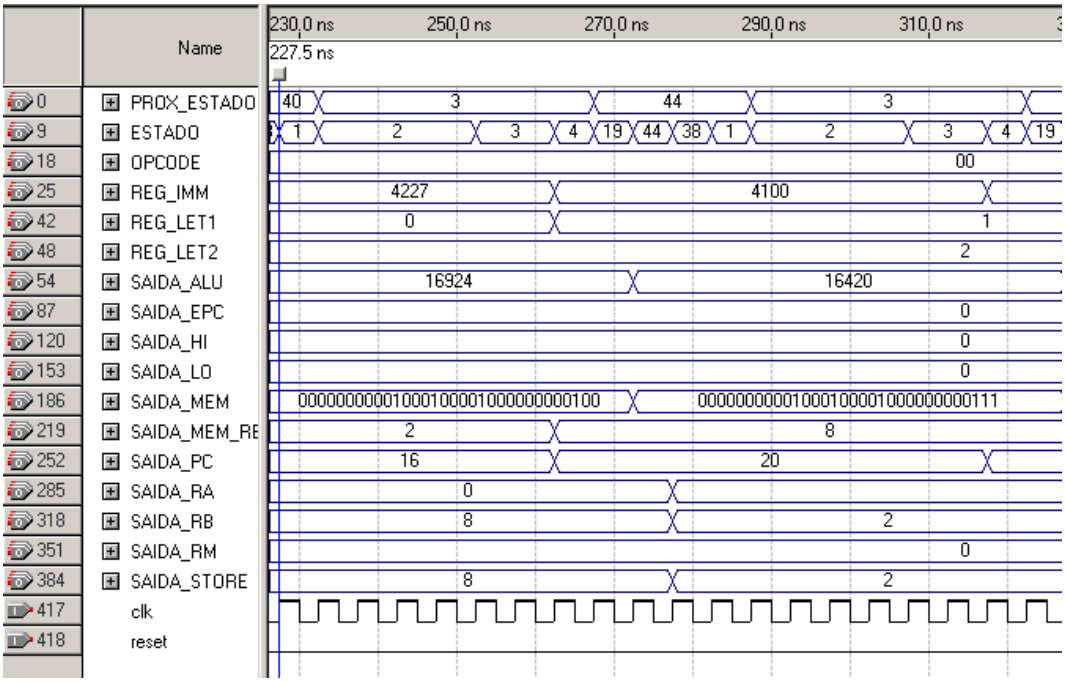
- sra:


Name
180,0 ns 190,0 ns 200,0 ns 210,0 ns 220,0 ns 230,0 ns 240,0 ns 250,0 ns 260,0 ns
170.0 ns
0 PROX_ESTADO
9 ESTADO
18 OPCODE
25 REG_IMM
42 REG_LET1
48 REG_LET2
54 SAIDA_ALU
87 SAIDA_EPC
120 SAIDA_HI
153 SAIDA_LO
186 SAIDA_MEM
219 SAIDA_MEM_RE
252 SAIDA_PC
285 SAIDA_RA
318 SAIDA_RB
351 SAIDA_RM
384 SAIDA_STORE
417 clk
418 reset

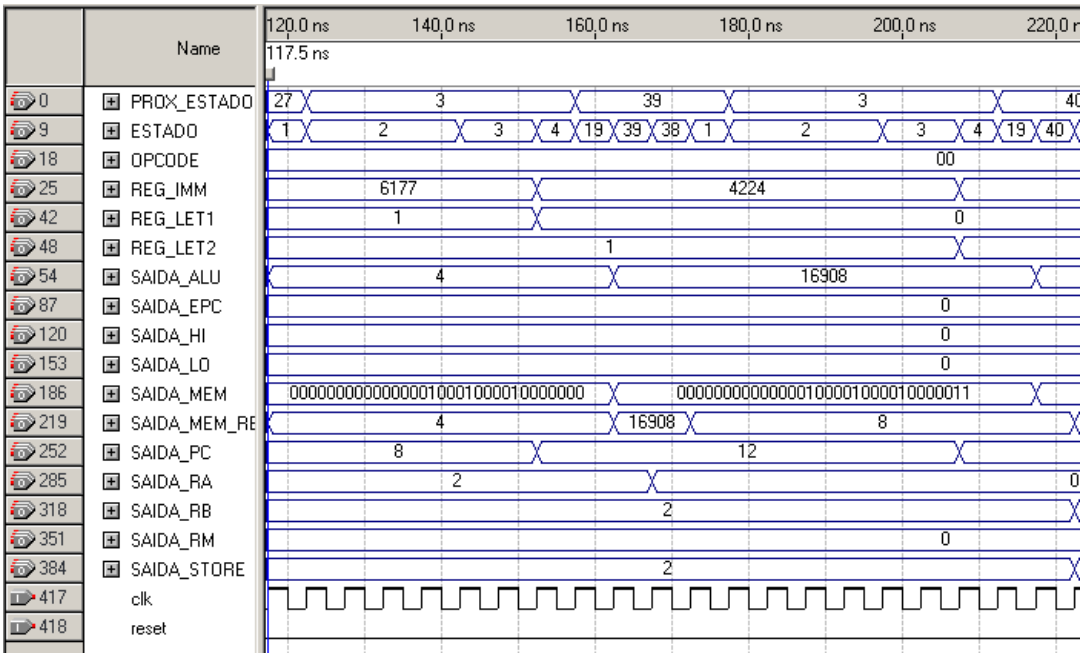
- slt:


Name
70,0 ns 80,0 ns 90,0 ns 100,0 ns 110,0 ns 120,0 ns 130,0 ns
62.5 ns
0 PROX_ESTADO
9 ESTADO
18 OPCODE
25 REG_IMM
42 REG_LET1
48 REG_LET2
54 SAIDA_ALU
87 SAIDA_EPC
120 SAIDA_HI
153 SAIDA_LO
186 SAIDA_MEM
219 SAIDA_MEM_RE
252 SAIDA_PC
285 SAIDA_RA
318 SAIDA_RB
351 SAIDA_RM
384 SAIDA_STORE
417 clk
418 reset

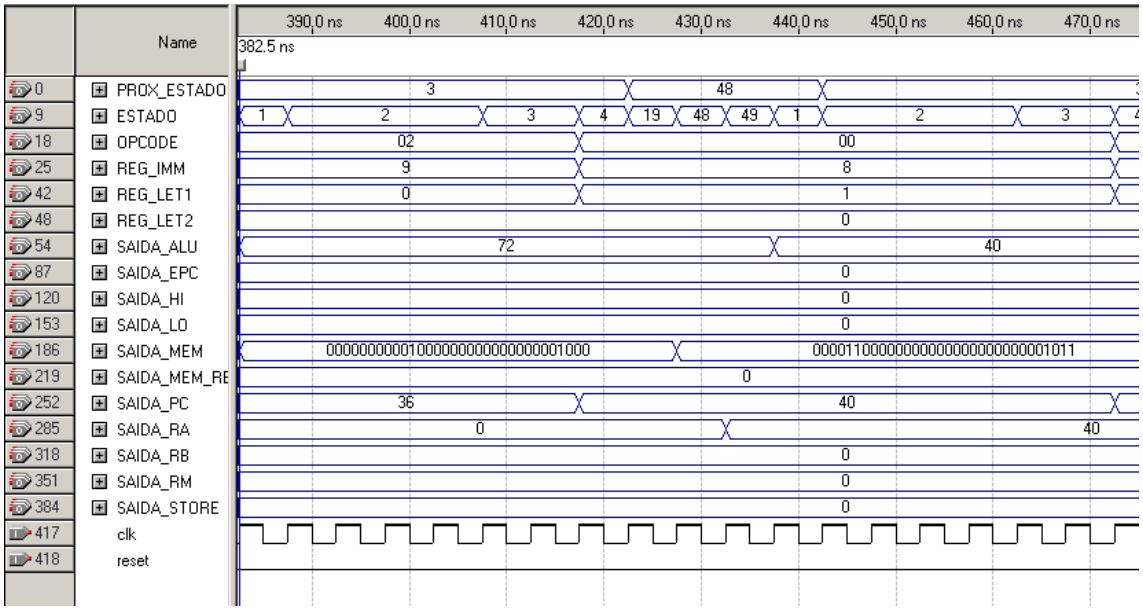
- sllv:


Name
230,0 ns
250,0 ns
270,0 ns
290,0 ns
310,0 ns
227.5 ns
0 PROX_ESTADO
9 ESTADO
18 OPCODE
25 REG_IMM
42 REG_LET1
48 REG_LET2
54 SAIDA_ALU
87 SAIDA_EPC
120 SAIDA_HI
153 SAIDA_LO
186 SAIDA_MEM
219 SAIDA_MEM_RE
252 SAIDA_PC
285 SAIDA_RA
318 SAIDA_RB
351 SAIDA_RM
384 SAIDA_STORE
417 clk
418 reset

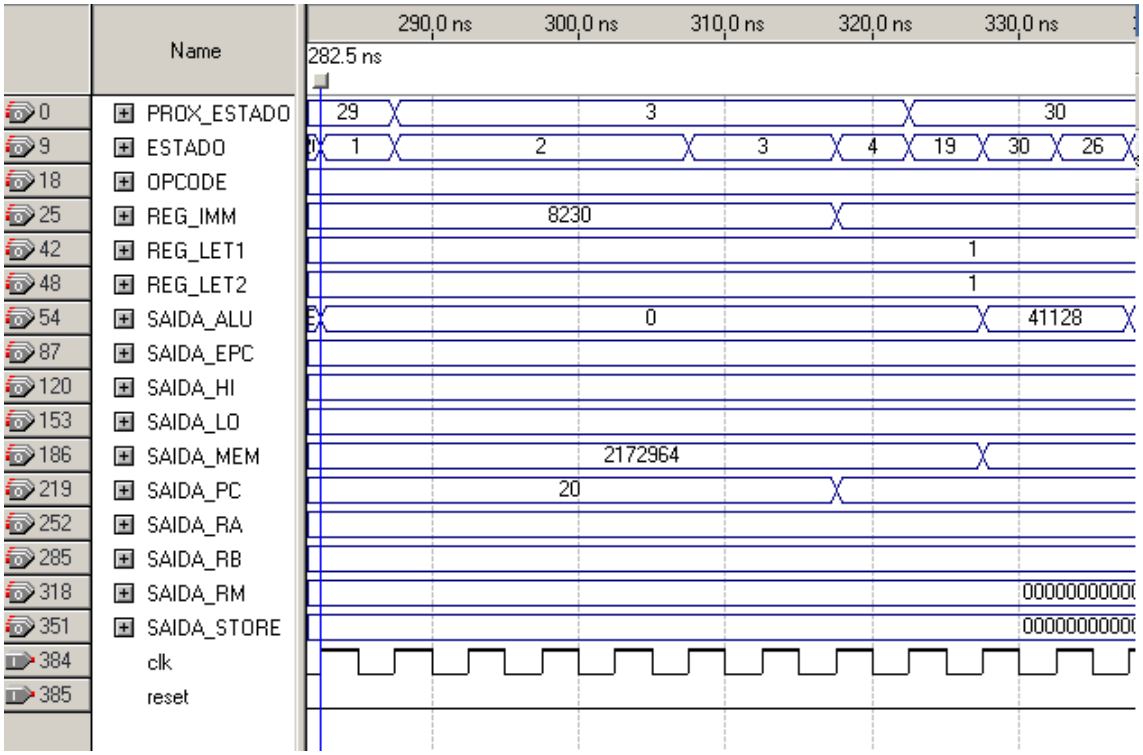
- sll:


Name
120,0 ns
140,0 ns
160,0 ns
180,0 ns
200,0 ns
117.5 ns
0 PROX_ESTADO
9 ESTADO
18 OPCODE
25 REG_IMM
42 REG_LET1
48 REG_LET2
54 SAIDA_ALU
87 SAIDA_EPC
120 SAIDA_HI
153 SAIDA_LO
186 SAIDA_MEM
219 SAIDA_MEM_RE
252 SAIDA_PC
285 SAIDA_RA
318 SAIDA_RB
351 SAIDA_RM
384 SAIDA_STORE
417 clk
418 reset

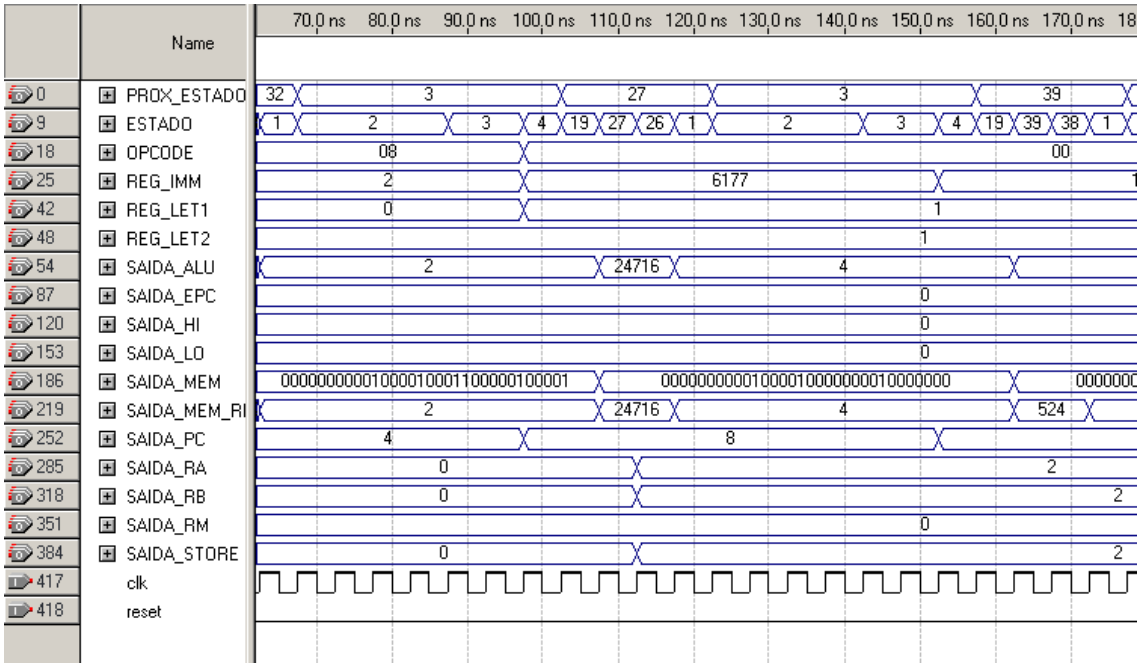
- jr:


Name
390,0 ns
400,0 ns
410,0 ns
420,0 ns
430,0 ns
440,0 ns
450,0 ns
460,0 ns
470,0 ns
382.5 ns
0 PROX_ESTADO
9 ESTADO
18 OPCODE
25 REG_IMM
42 REG_LET1
48 REG_LET2
54 SAIDA_ALU
87 SAIDA_EPC
120 SAIDA_HI
153 SAIDA_LO
186 SAIDA_MEM
219 SAIDA_MEM_RE
252 SAIDA_PC
285 SAIDA_RA
318 SAIDA_RB
351 SAIDA_RM
384 SAIDA_STORE
417 clk
418 reset

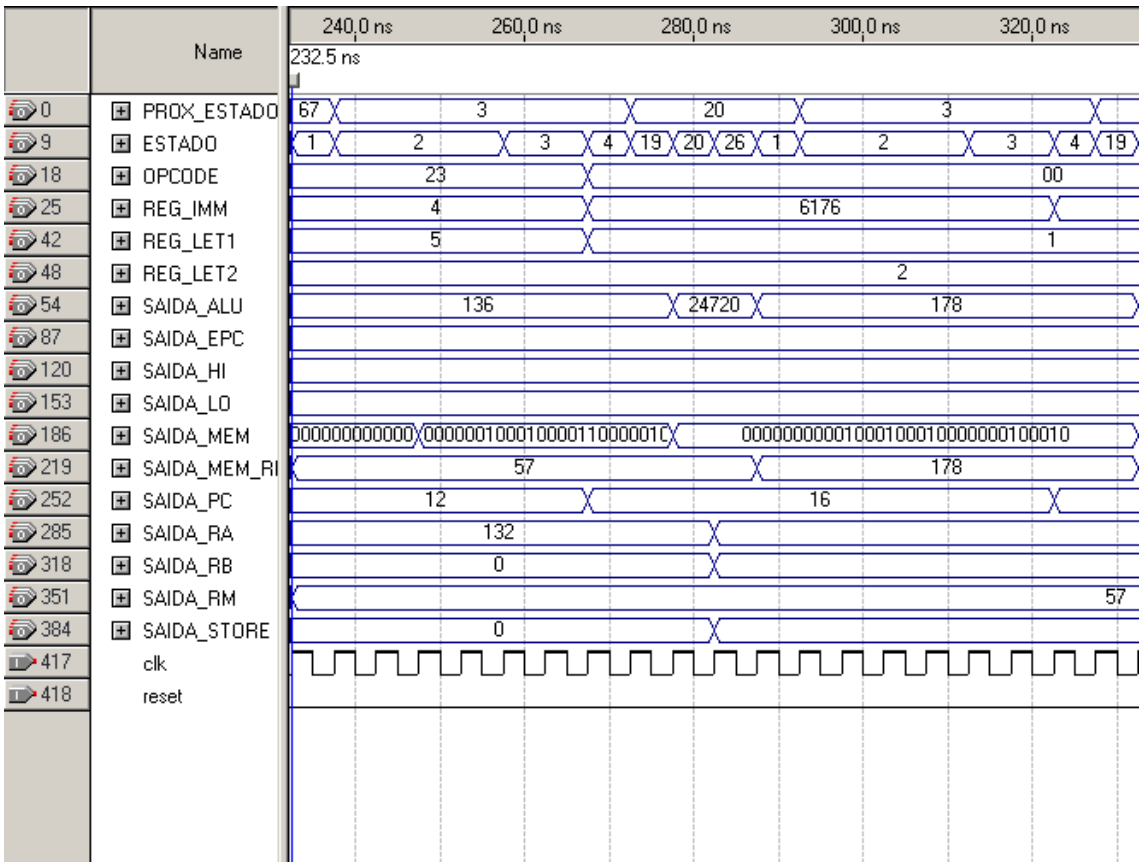
- and:


Name
290,0 ns
300,0 ns
310,0 ns
320,0 ns
330,0 ns
282.5 ns
0 PROX_ESTADO
9 ESTADO
18 OPCODE
25 REG_IMM
42 REG_LET1
48 REG_LET2
54 SAIDA_ALU
87 SAIDA_EPC
120 SAIDA_HI
153 SAIDA_LO
186 SAIDA_MEM
219 SAIDA_PC
252 SAIDA_RA
285 SAIDA_RB
318 SAIDA_RM
351 SAIDA_STORE
384 clk
385 reset

- addu:

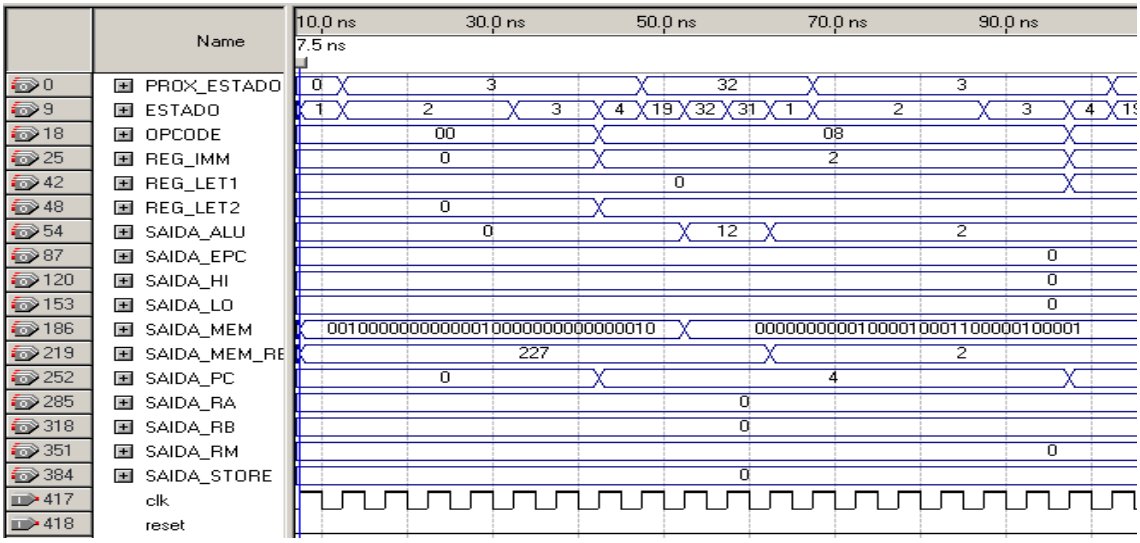

Name
70,0 ns
80,0 ns
90,0 ns
100,0 ns
110,0 ns
120,0 ns
130,0 ns
140,0 ns
150,0 ns
160,0 ns
170,0 ns
0 PROX_ESTADO
9 ESTADO
18 OPCODE
25 REG_IMM
42 REG_LET1
48 REG_LET2
54 SAIDA_ALU
87 SAIDA_EPC
120 SAIDA_HI
153 SAIDA_LO
186 SAIDA_MEM
219 SAIDA_MEM_RI
252 SAIDA_PC
285 SAIDA_RA
318 SAIDA_RB
351 SAIDA_RM
384 SAIDA_STORE
417 clk
418 reset

- add:

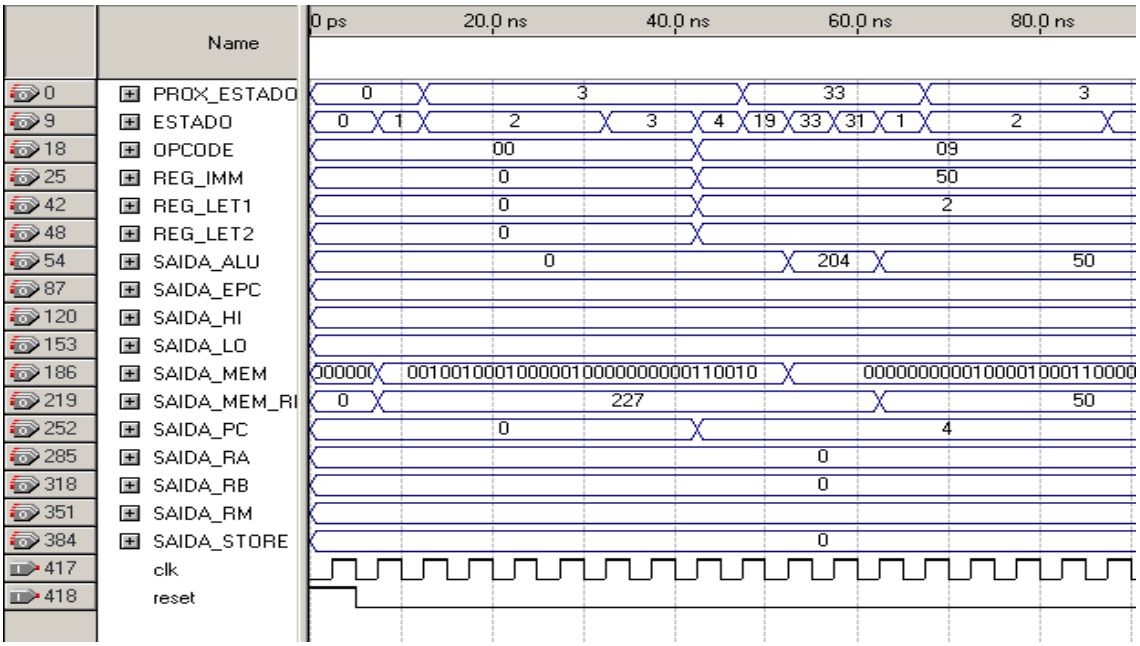

Name
240,0 ns
260,0 ns
280,0 ns
300,0 ns
320,0 ns
232.5 ns
0 PROX_ESTADO
9 ESTADO
18 OPCODE
25 REG_IMM
42 REG_LET1
48 REG_LET2
54 SAIDA_ALU
87 SAIDA_EPC
120 SAIDA_HI
153 SAIDA_LO
186 SAIDA_MEM
219 SAIDA_MEM_RI
252 SAIDA_PC
285 SAIDA_RA
318 SAIDA_RB
351 SAIDA_RM
384 SAIDA_STORE
417 clk
418 reset

2. Instruções Tipo-I

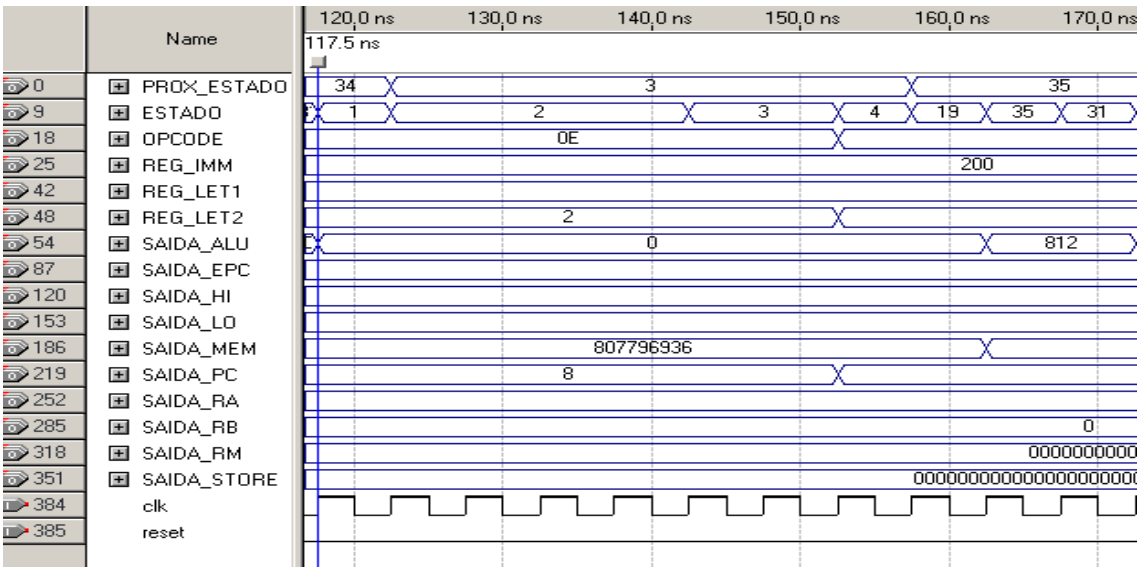
- addi:



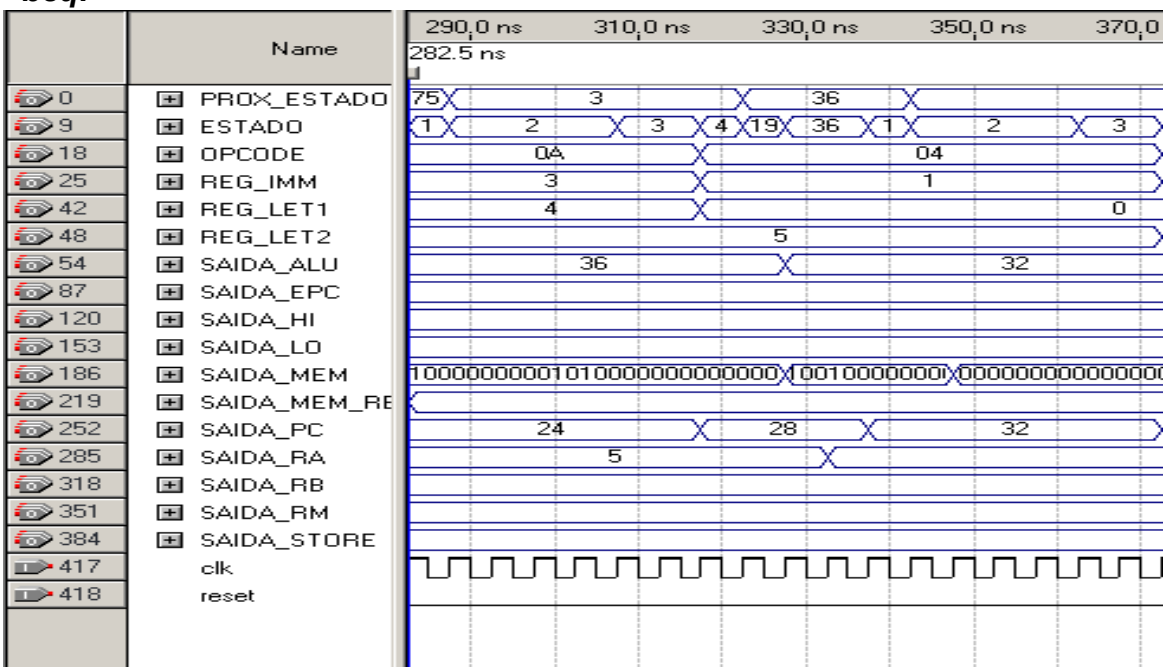
- addiu:



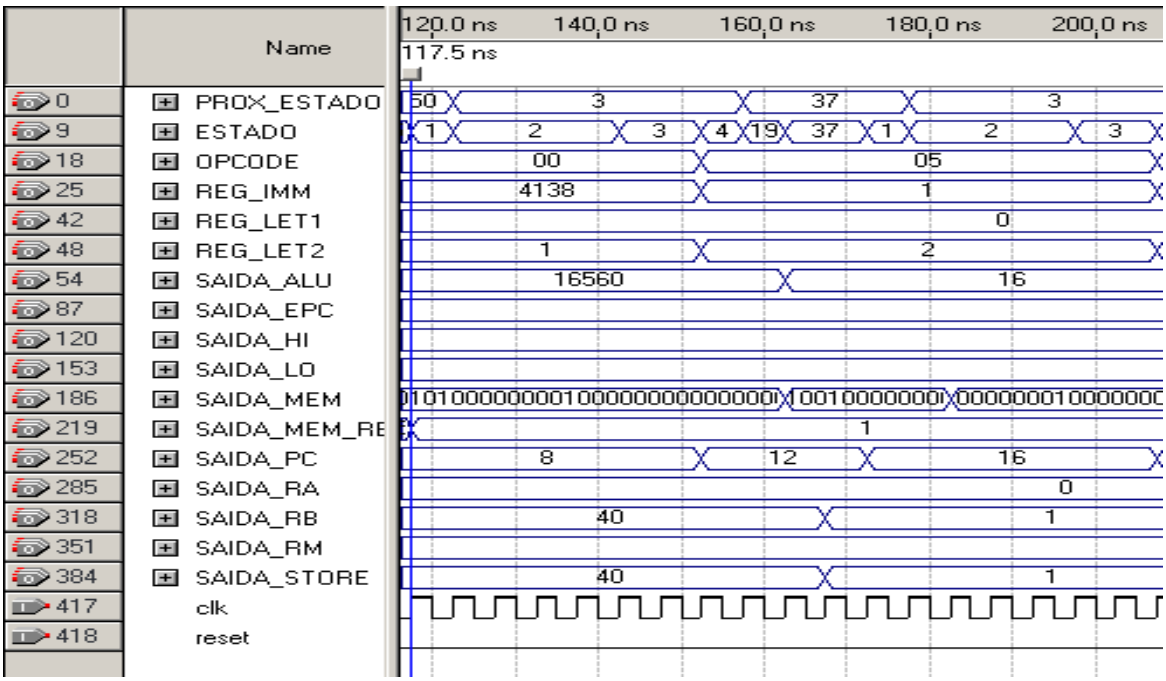
- andi:


Name
120,0 ns
130,0 ns
140,0 ns
150,0 ns
160,0 ns
170,0 ns
117.5 ns
0 PROX_ESTADO
9 ESTADO
18 OPCODE
25 REG_IMM
42 REG_LET1
48 REG_LET2
54 SAIDA_ALU
87 SAIDA_EPC
120 SAIDA_HI
153 SAIDA_LO
186 SAIDA_MEM
219 SAIDA_PC
252 SAIDA_RA
285 SAIDA_RB
318 SAIDA_RM
351 SAIDA_STORE
384 clk
385 reset
34
3
35
1
2
3
4
19
35
31
0E
200
2
0
812
807796936
8
0
0000000000
00000000000000000000

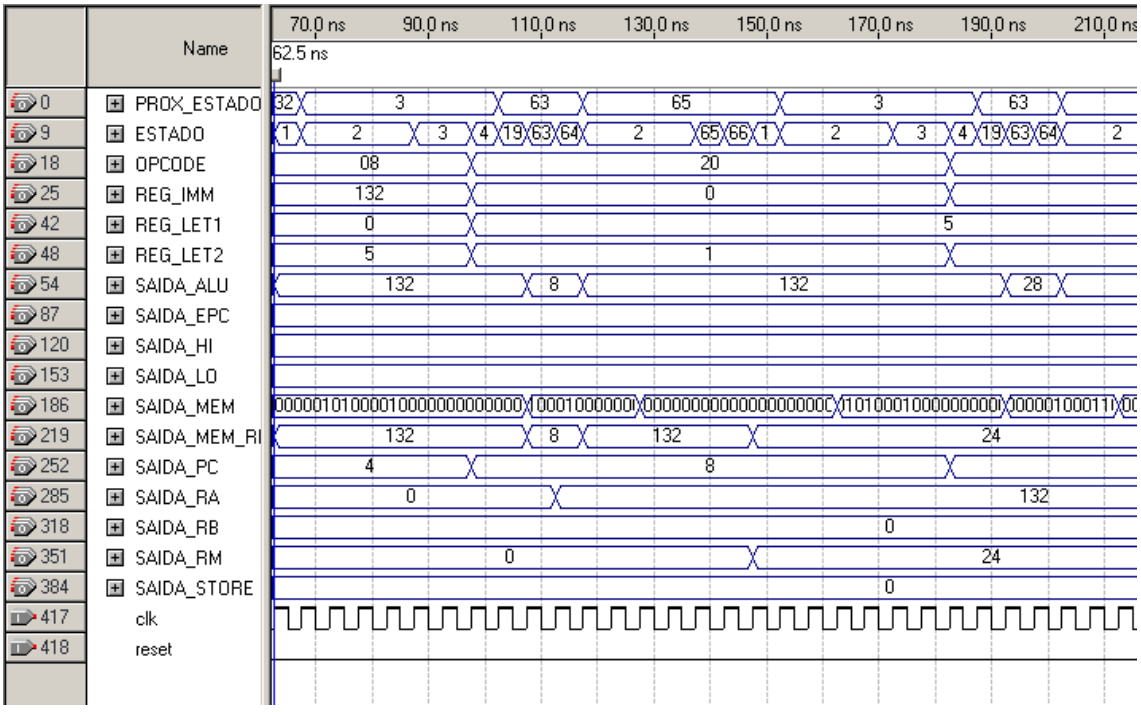
- beq:


Name
290,0 ns
310,0 ns
330,0 ns
350,0 ns
370,0
282.5 ns
0 PROX_ESTADO
9 ESTADO
18 OPCODE
25 REG_IMM
42 REG_LET1
48 REG_LET2
54 SAIDA_ALU
87 SAIDA_EPC
120 SAIDA_HI
153 SAIDA_LO
186 SAIDA_MEM
219 SAIDA_MEM_RE
252 SAIDA_PC
285 SAIDA_RA
318 SAIDA_RB
351 SAIDA_RM
384 SAIDA_STORE
417 clk
418 reset
75
3
36
1
2
3
4
19
36
1
2
3
0A
04
3
1
4
0
5
36
32
24
28
32
5

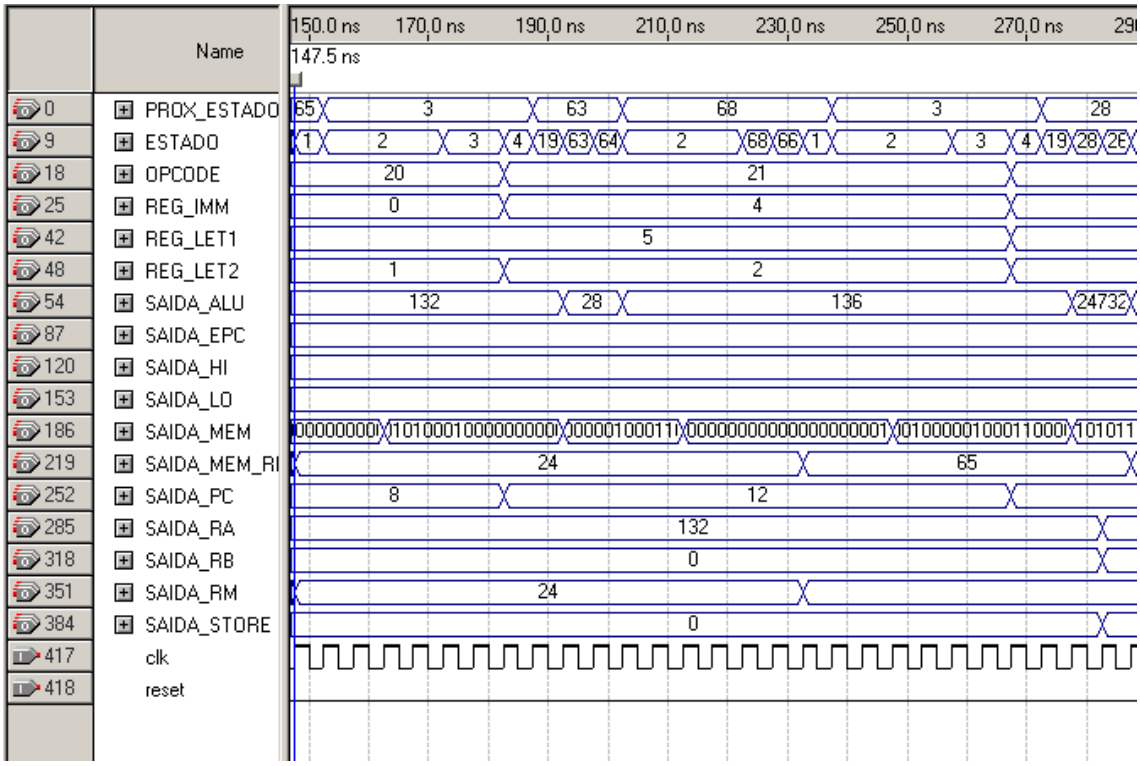
- bne:


Name
120,0 ns
140,0 ns
160,0 ns
180,0 ns
200,0 ns
117.5 ns
0 PROX_ESTADO
9 ESTADO
18 OPCODE
25 REG_IMM
42 REG_LET1
48 REG_LET2
54 SAIDA_ALU
87 SAIDA_EPC
120 SAIDA_HI
153 SAIDA_LO
186 SAIDA_MEM
219 SAIDA_MEM_RE
252 SAIDA_PC
285 SAIDA_RA
318 SAIDA_RB
351 SAIDA_RM
384 SAIDA_STORE
417 clk
418 reset

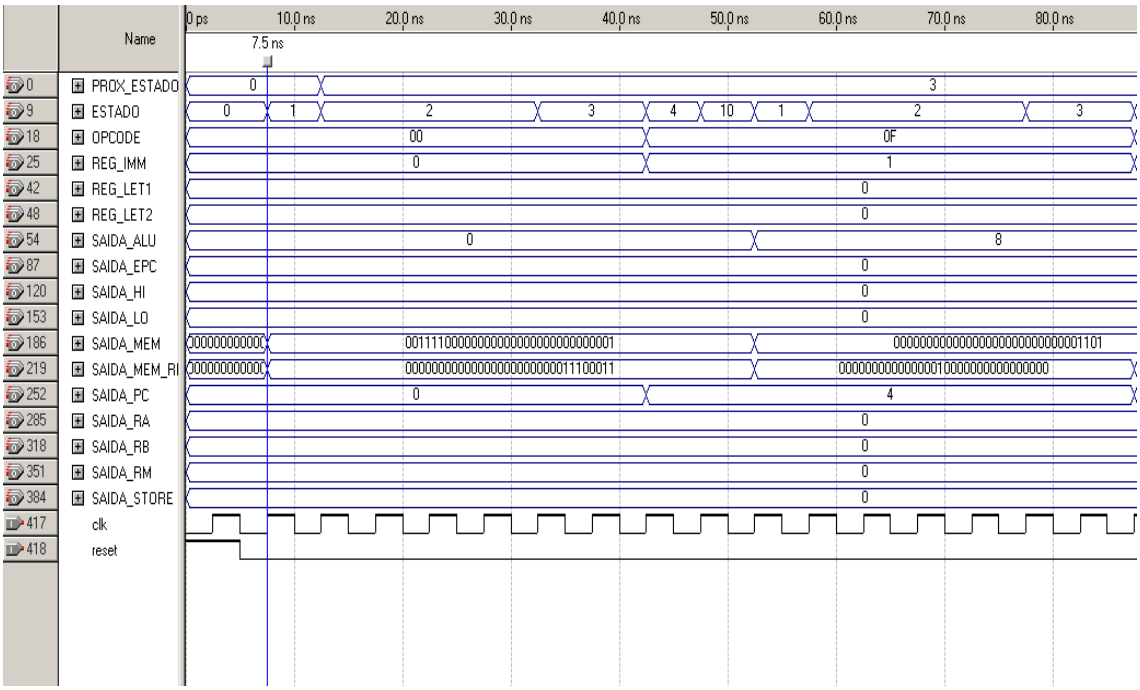
- lb:


Name
70,0 ns
90,0 ns
110,0 ns
130,0 ns
150,0 ns
170,0 ns
190,0 ns
210,0 ns
62.5 ns
0 PROX_ESTADO
9 ESTADO
18 OPCODE
25 REG_IMM
42 REG_LET1
48 REG_LET2
54 SAIDA_ALU
87 SAIDA_EPC
120 SAIDA_HI
153 SAIDA_LO
186 SAIDA_MEM
219 SAIDA_MEM_RI
252 SAIDA_PC
285 SAIDA_RA
318 SAIDA_RB
351 SAIDA_RM
384 SAIDA_STORE
417 clk
418 reset

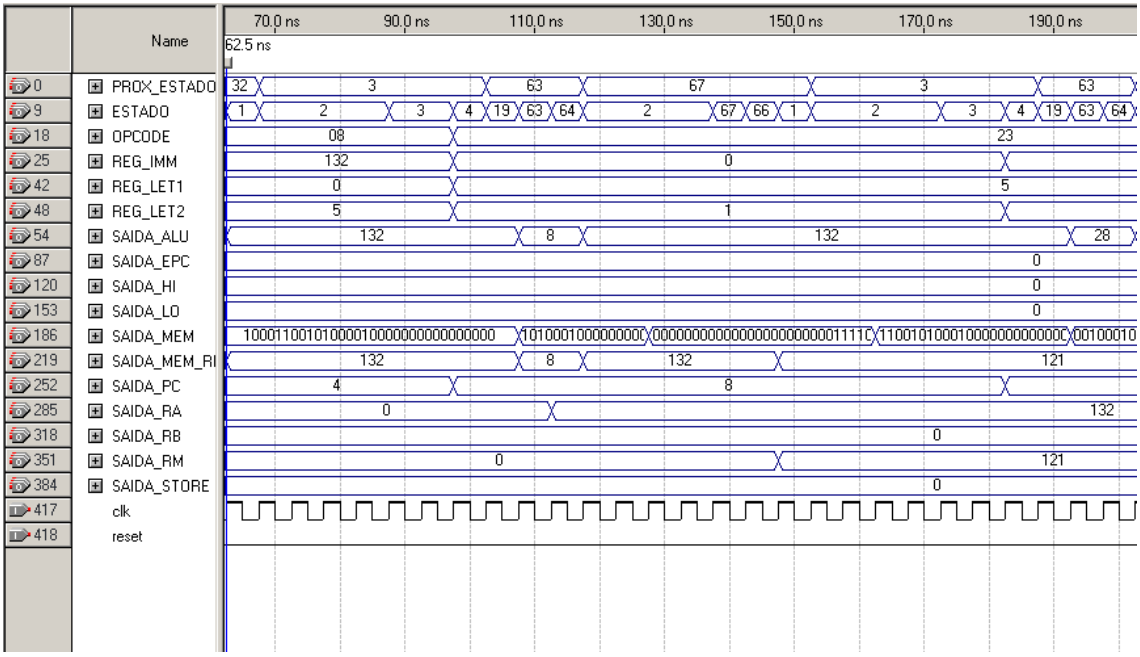
- lh:


Name
150.0 ns
170,0 ns
190,0 ns
210,0 ns
230,0 ns
250,0 ns
270,0 ns
147.5 ns
0 PROX_ESTADO
9 ESTADO
18 OPCODE
25 REG_IMM
42 REG_LET1
48 REG_LET2
54 SAIDA_ALU
87 SAIDA_EPC
120 SAIDA_HI
153 SAIDA_LO
186 SAIDA_MEM
219 SAIDA_MEM_RI
252 SAIDA_PC
285 SAIDA_RA
318 SAIDA_RB
351 SAIDA_RM
384 SAIDA_STORE
417 clk
418 reset

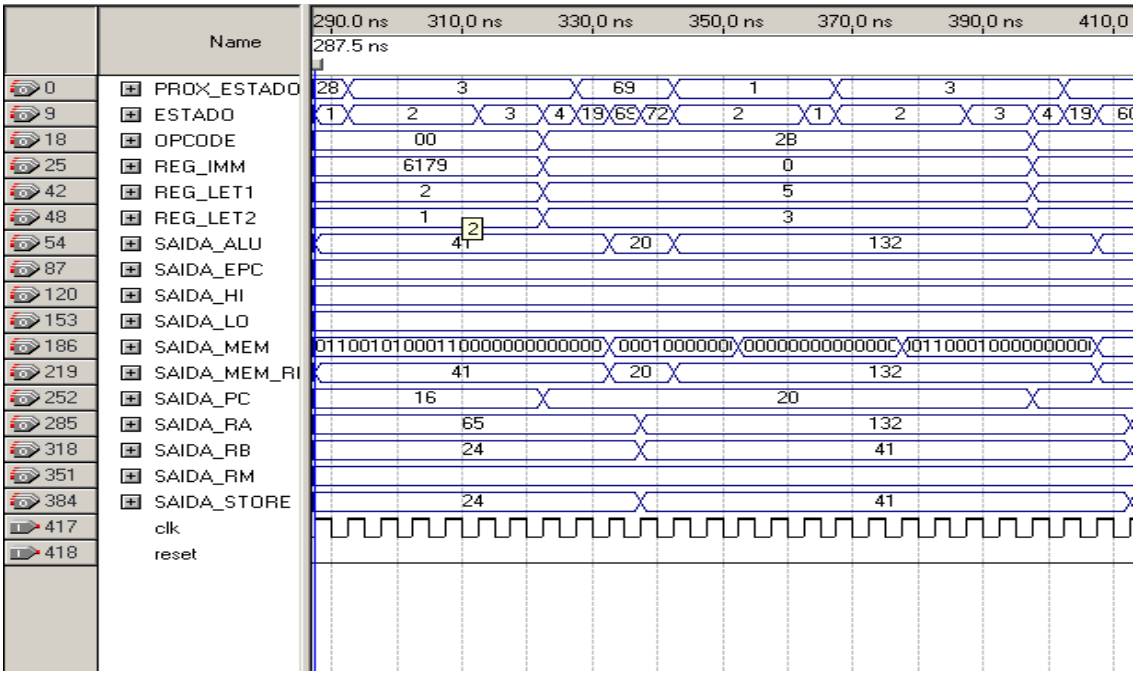
- lui:


Name
0 ps
10,0 ns
20,0 ns
30,0 ns
40,0 ns
50,0 ns
60,0 ns
70,0 ns
80,0 ns
7.5 ns
0 PROX_ESTADO
9 ESTADO
18 OPCODE
25 REG_IMM
42 REG_LET1
48 REG_LET2
54 SAIDA_ALU
87 SAIDA_EPC
120 SAIDA_HI
153 SAIDA_LO
186 SAIDA_MEM
219 SAIDA_MEM_RI
252 SAIDA_PC
285 SAIDA_RA
318 SAIDA_RB
351 SAIDA_RM
384 SAIDA_STORE
417 clk
418 reset

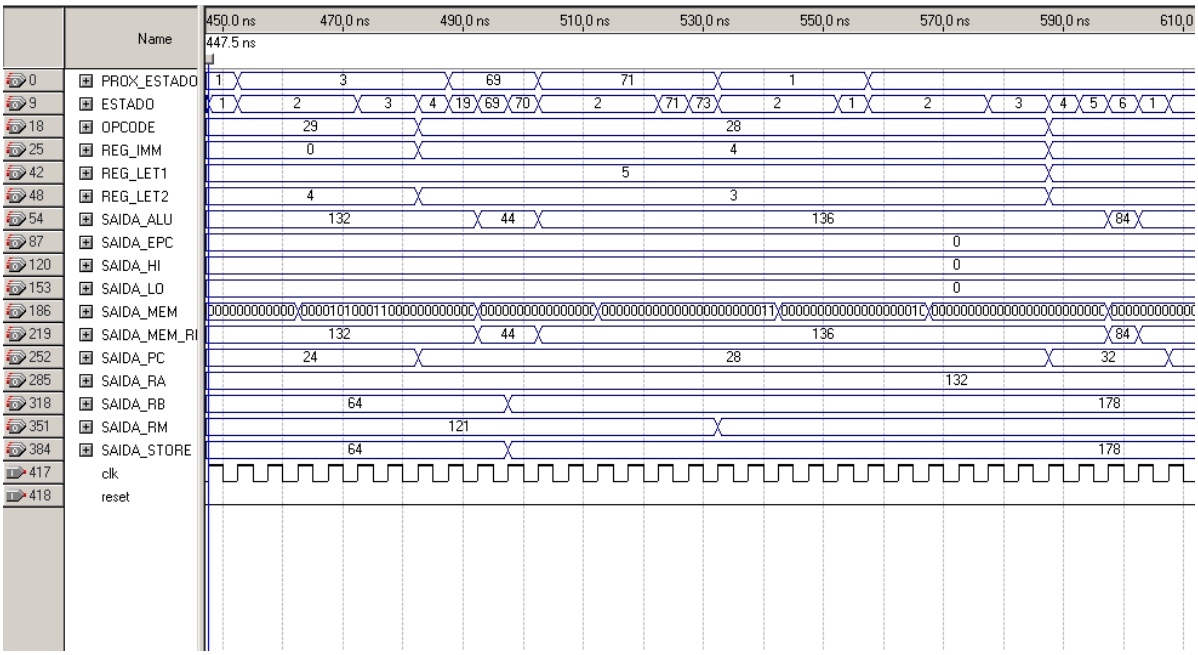
- lw:


Name
70,0 ns
90,0 ns
110,0 ns
130,0 ns
150,0 ns
170,0 ns
190,0 ns
62.5 ns
0 PROX_ESTADO
9 ESTADO
18 OPCODE
25 REG_IMM
42 REG_LET1
48 REG_LET2
54 SAIDA_ALU
87 SAIDA_EPC
120 SAIDA_HI
153 SAIDA_LO
186 SAIDA_MEM
219 SAIDA_MEM_RI
252 SAIDA_PC
285 SAIDA_RA
318 SAIDA_RB
351 SAIDA_RM
384 SAIDA_STORE
417 clk
418 reset

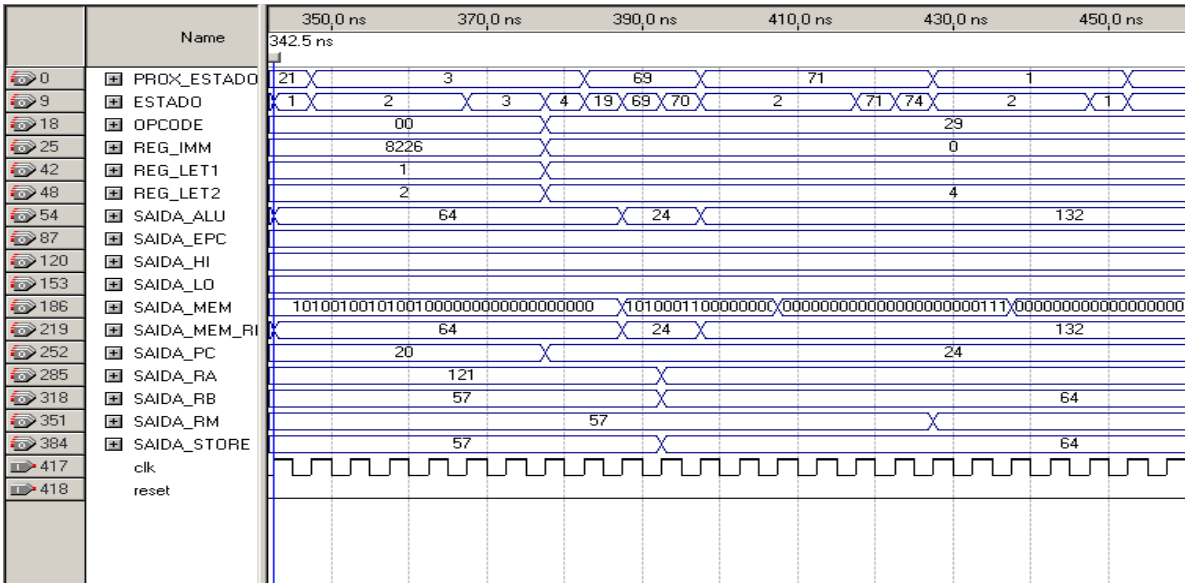
- sw:


Name
290.0 ns
310,0 ns
330,0 ns
350,0 ns
370,0 ns
390,0 ns
410,0
287.5 ns
0 PROX_ESTADO
9 ESTADO
18 OPCODE
25 REG_IMM
42 REG_LET1
48 REG_LET2
54 SAIDA_ALU
87 SAIDA_EPC
120 SAIDA_HI
153 SAIDA_LO
186 SAIDA_MEM
219 SAIDA_MEM_RI
252 SAIDA_PC
285 SAIDA_RA
318 SAIDA_RB
351 SAIDA_RM
384 SAIDA_STORE
417 clk
418 reset

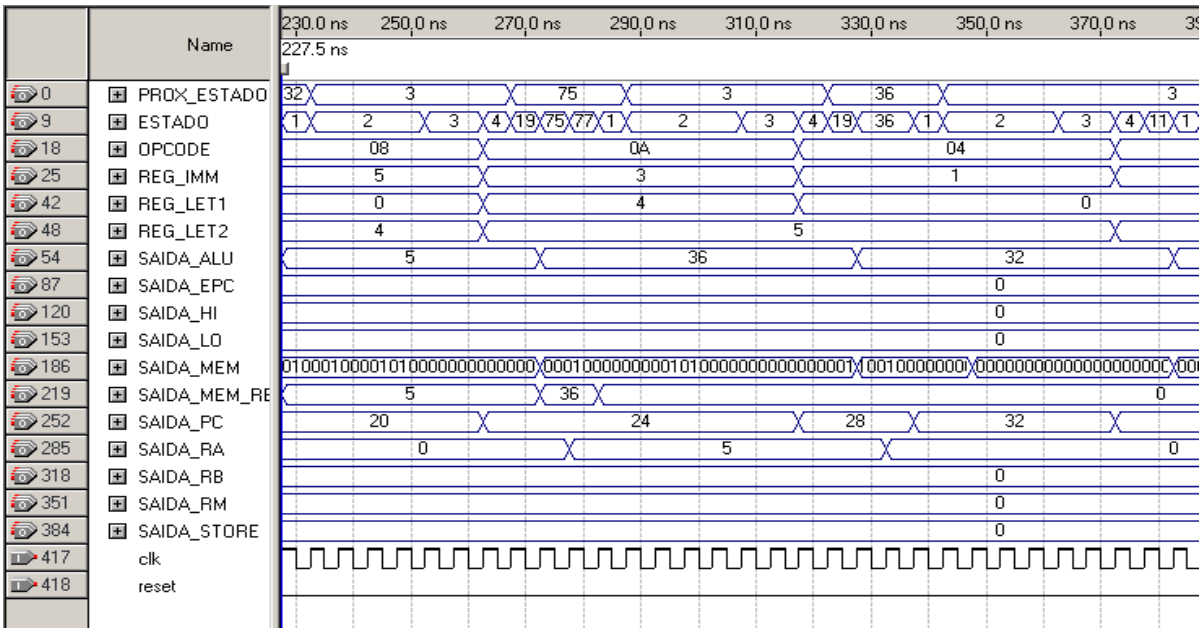
- sb:


Name
450,0 ns
470,0 ns
490,0 ns
510,0 ns
530,0 ns
550,0 ns
570,0 ns
590,0 ns
610,0
447.5 ns
0 PROX_ESTADO
9 ESTADO
18 OPCODE
25 REG_IMM
42 REG_LET1
48 REG_LET2
54 SAIDA_ALU
87 SAIDA_EPC
120 SAIDA_HI
153 SAIDA_LO
186 SAIDA_MEM
219 SAIDA_MEM_RI
252 SAIDA_PC
285 SAIDA_RA
318 SAIDA_RB
351 SAIDA_RM
384 SAIDA_STORE
417 clk
418 reset

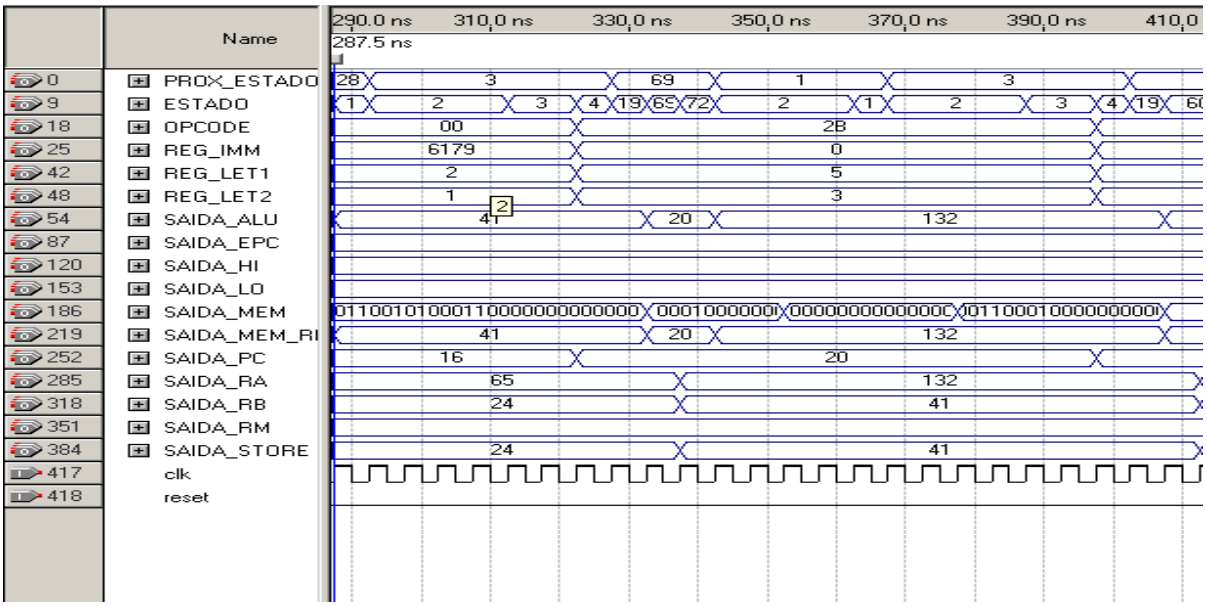
- sh:


Name
350,0 ns
370,0 ns
390,0 ns
410,0 ns
430,0 ns
450,0 ns
342.5 ns
0 PROX_ESTADO
9 ESTADO
18 OPCODE
25 REG_IMM
42 REG_LET1
48 REG_LET2
54 SAIDA_ALU
87 SAIDA_EPC
120 SAIDA_HI
153 SAIDA_LO
186 SAIDA_MEM
219 SAIDA_MEM_RI
252 SAIDA_PC
285 SAIDA_RA
318 SAIDA_RB
351 SAIDA_RM
384 SAIDA_STORE
417 clk
418 reset

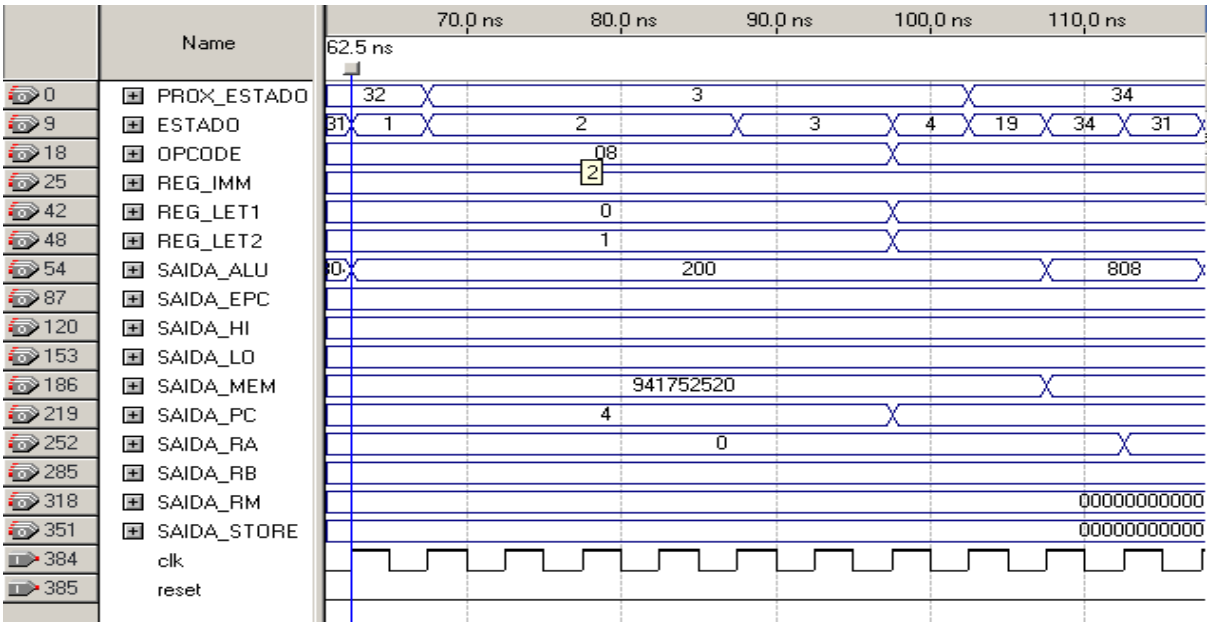
- slti:


Name
230,0 ns
250,0 ns
270,0 ns
290,0 ns
310,0 ns
330,0 ns
350,0 ns
370,0 ns
227.5 ns
0 PROX_ESTADO
9 ESTADO
18 OPCODE
25 REG_IMM
42 REG_LET1
48 REG_LET2
54 SAIDA_ALU
87 SAIDA_EPC
120 SAIDA_HI
153 SAIDA_LO
186 SAIDA_MEM
219 SAIDA_MEM_RE
252 SAIDA_PC
285 SAIDA_RA
318 SAIDA_RB
351 SAIDA_RM
384 SAIDA_STORE
417 clk
418 reset

- sw:

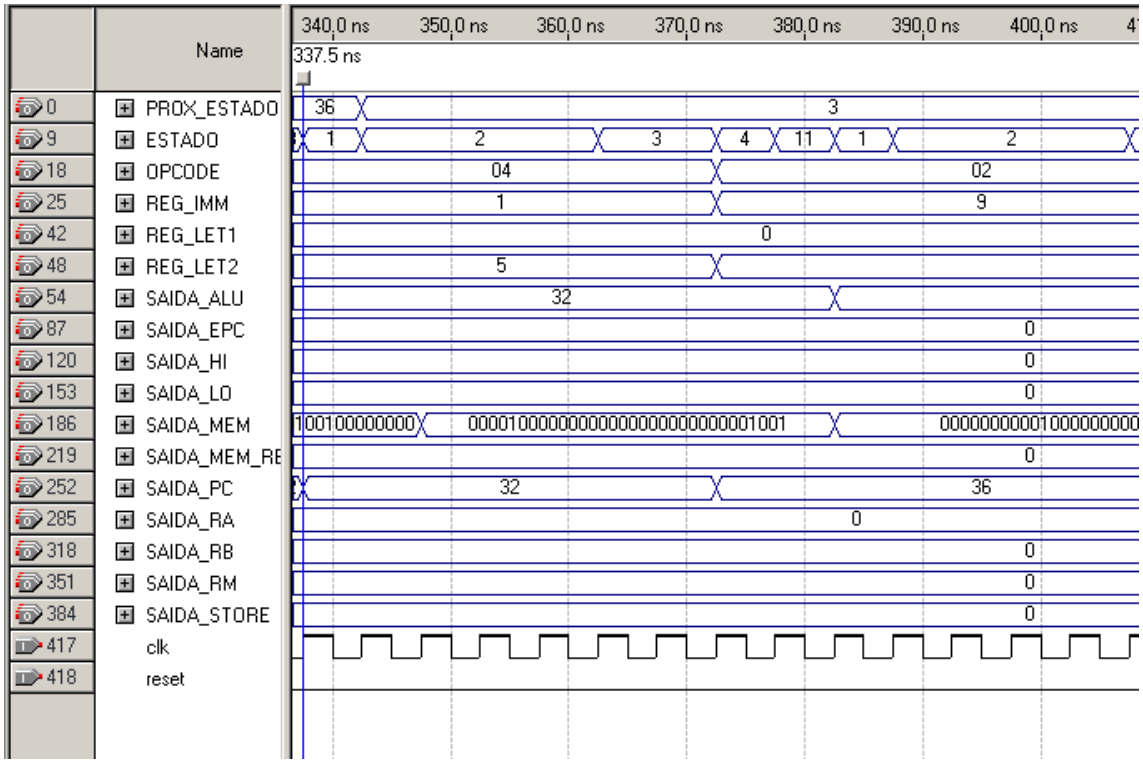

Name
290,0 ns
310,0 ns
330,0 ns
350,0 ns
370,0 ns
390,0 ns
287.5 ns
0 PROX_ESTADO
9 ESTADO
18 OPCODE
25 REG_IMM
42 REG_LET1
48 REG_LET2
54 SAIDA_ALU
87 SAIDA_EPC
120 SAIDA_HI
153 SAIDA_LO
186 SAIDA_MEM
219 SAIDA_MEM_RI
252 SAIDA_PC
285 SAIDA_RA
318 SAIDA_RB
351 SAIDA_RM
384 SAIDA_STORE
417 clk
418 reset

- xori:

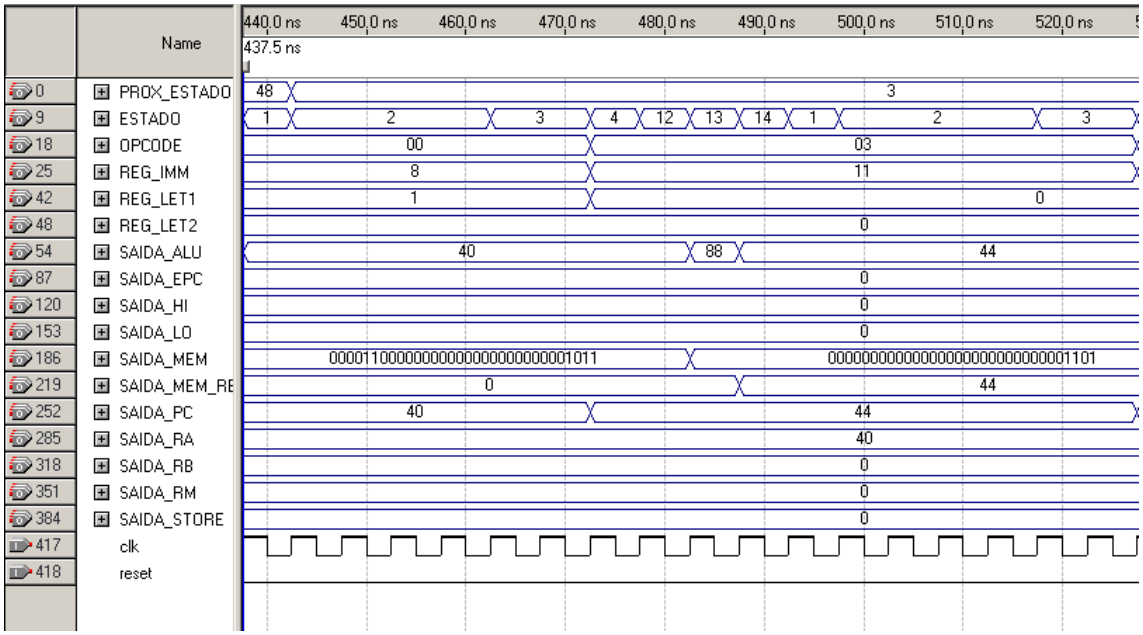

Name
70,0 ns
80,0 ns
90,0 ns
100,0 ns
110,0 ns
62.5 ns
0 PROX_ESTADO
9 ESTADO
18 OPCODE
25 REG_IMM
42 REG_LET1
48 REG_LET2
54 SAIDA_ALU
87 SAIDA_EPC
120 SAIDA_HI
153 SAIDA_LO
186 SAIDA_MEM
219 SAIDA_PC
252 SAIDA_RA
285 SAIDA_RB
318 SAIDA_RM
351 SAIDA_STORE
384 clk
385 reset
941752520
00000000000
00000000000

3. Instruções Tipo-J

- jump ('j'):

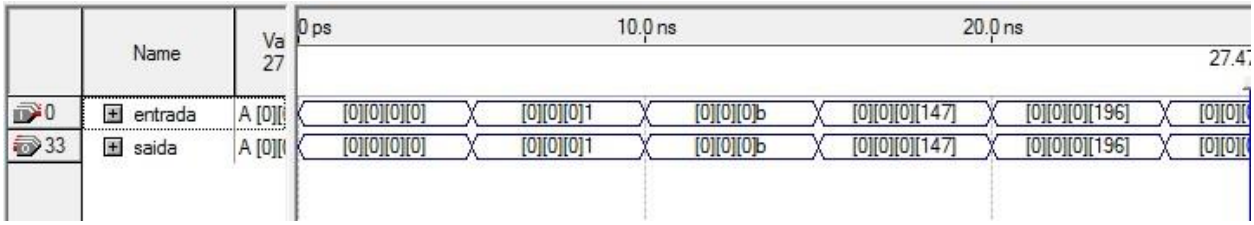


- jal:



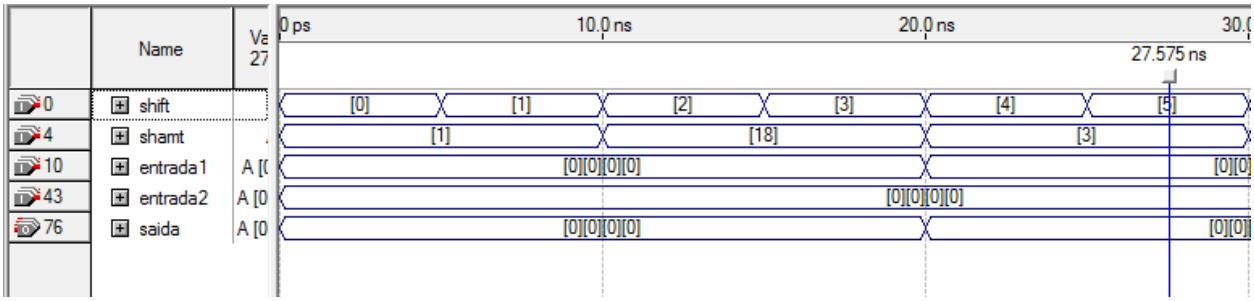
4. Extended_EPC

No extensor do EPC, são utilizados os 8 bits mais significativos com o dado recebido e os outro 24 bits menos significativos são preenchidos com 0.



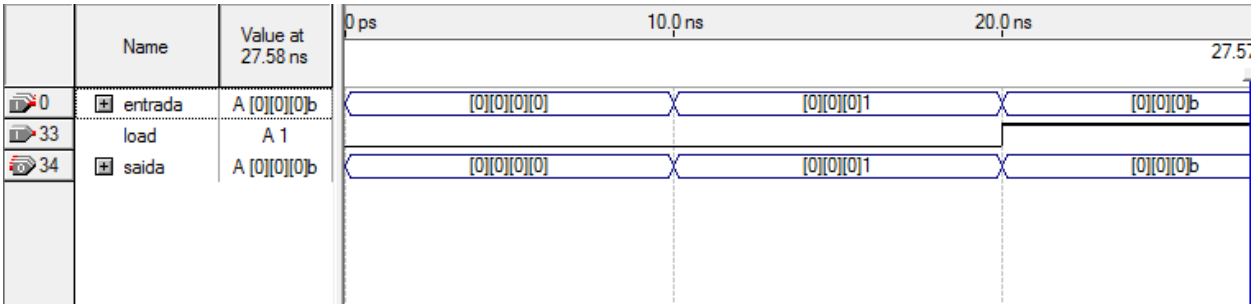
5. Shift

Neste modulo serão tratadas todas as instruções de ‘shift’.



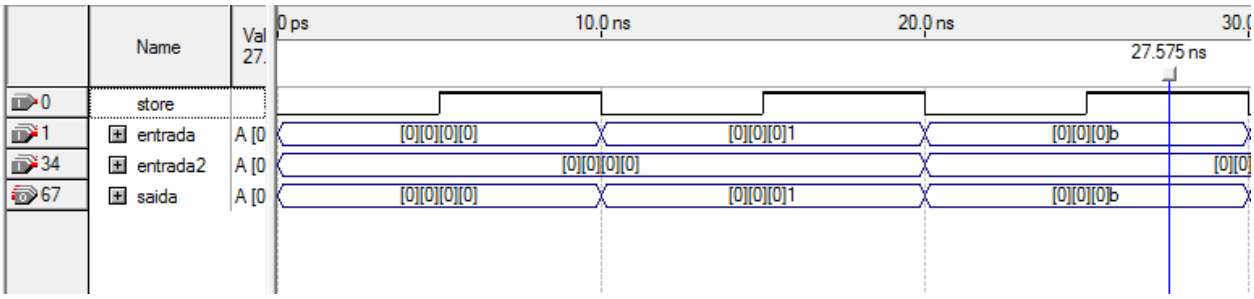
6. Load

A instrução de load é dividida em load byte, load halfword e load word. As opções de load byte e load halfword são computadas dentro de uma unidade responsável por zerar os bits que são insignificantes para a instrução. No caso de Load Byte e Load Halfword, são lidos 32 bits da memória e 32 bits provenientes do registrador B. Dentro da caixa, é feita a seleção da instrução através de um seletor de 1 bit e então computada a instrução correspondente. No caso do Load Byte, 24 bits são descartados, adicionando zeros nos bits mais significativos. No caso do Load Halfword, são 16 bits descartados da mesma forma que o LB. No caso da Load Word, o conteúdo de 32 bits passa direto, sem passar pela LoadBox.



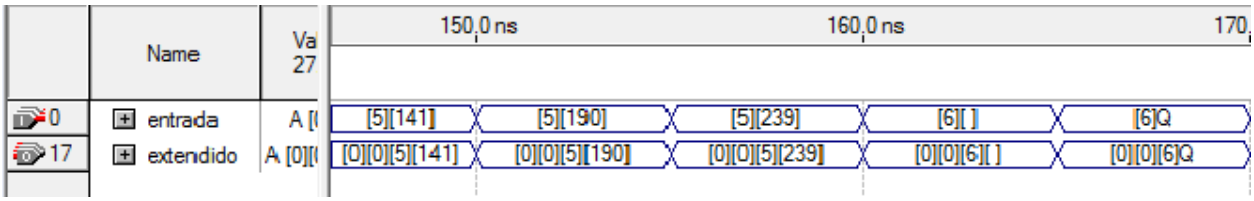
7. Store

A instrução de Store é dividida em store word, store byte e store halfword. No primeiro caso, o conteúdo lido é enviado diretamente para um Mux com um seletor que seleciona a instrução que está sendo executada. Nos outros casos, os dados da MDR e do Registrador B passam pela StoreBox a fim de realizar a operação. No caso do store byte, são escritos 8 bits do registrador B, nos bits mais significativos da saída, e 24 bits do conteúdo de memória, no resto. Já o store halfword funciona de forma parecida, sendo 16 bits do registrador B escritos nos bits mais significativos da saída juntamente com 16 bits preenchendo o resto.



8. Signal_Extended_LUI

No extensor de sinal do lui a extensão é feita a fim de preservar o sinal do dado. Dessa forma, é feita uma verificação e de acordo com o sinal do conteúdo são preenchidos 16 bits de 1 (no caso de ser negativo) ou 16 bits de 0 (no caso de ser positivo).



6 CONCLUSÃO

A partir do aprendizado em sala de aula, utilizando-se livros recomendados para o estudo da Infraestrutura de Hardware, conseguimos projetar uma arquitetura de um processador baseada no ‘MIPS’.

Com todos os módulos implementados e todas as instruções testadas, terminamos o nosso projeto satisfeitos com o nosso trabalho, pois conseguimos o mais importante que foi a experiência e o aprendizado ganho durante o período com erros, acertos e contratempos durante os estudos.