

**ARQUITETURA DE COMPUTADORES**

***3° Projeto – CPU COM PIPELINE***

Membros: Christopher de Oliveira Souza RA: 18726430

Murilo De Paula Araujo 17747775

Leonardo Sanavio 18054395

Beatriz Morelatto Lorente 18071597

Professor: Roberto Santana De Rezende Edmar

**ÍNDICE**

1. Descrição textual do projeto com a topologia da CPU

2. Especificação e detalhes relevantes do projeto

3. Desenvolvimento e descrição da implementação

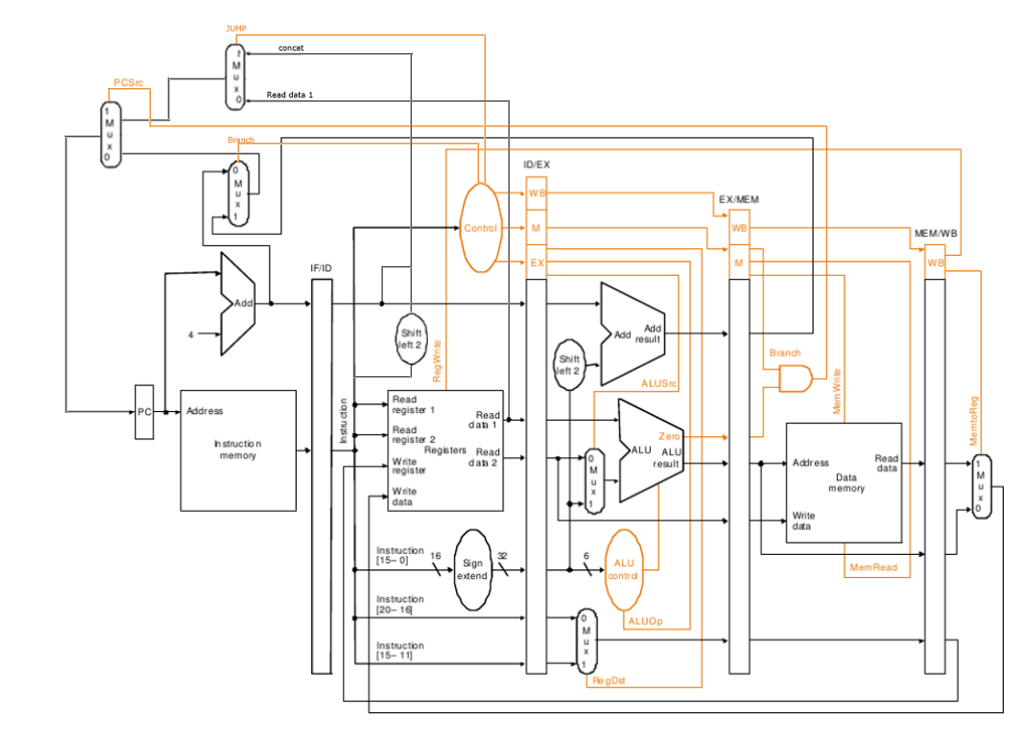
4. Resultados obtidos e funcionamento

5. Conclusões e analise dos resultados obtidos

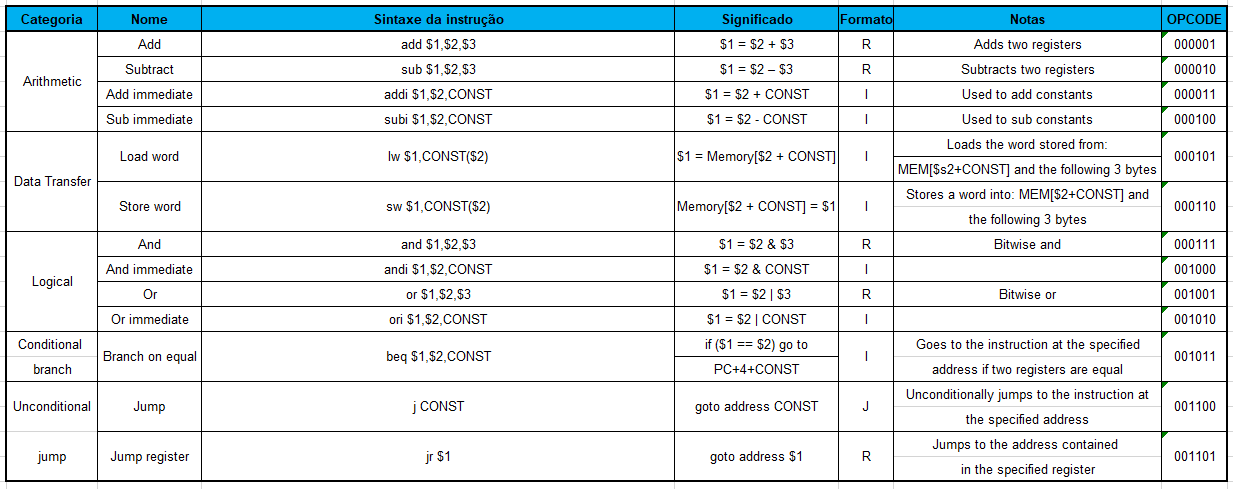
6. Bibliografia

1. **INTRODUÇÃO**

O projeto realizado é uma implementação de uma CPU com Pipeline em linguagem VHDL, o grupo partiu do seu datapath até sua real implementação no *Intel Quartus* onde foi programada, programando componente por componente, e em seguida implementada como um todo.

A implementação apresentada neste trabalho foi baseada no MIPS original de Hennessy e Patterson [HEN98] e na versão monociclo de [HAM00]. Na imagem abaixo, podemos observar o datapath com as alterações necessárias para realizar as instruções de jumps.

O MIPS possui três formatos de instruções diferentes: formato R, formato I e formato J. Onde as instruções do tipo R executam operações de lógicas e aritmética somente em dados armazenados nos registradores, as intruções do tipo I possuem dois tipos, as que fazem referência a memória como *load* e *store* e as instruções de *branch* (salto condicional).

A CPU tem que ser capaz de executar as seguintes instruções mostradas abaixo na tabela:

A memória do MIPS é endereçada a byte e, portanto, adiciona-se quatro para o cálculo do próximo endereço da palavra de 32 bits da memória, sendo os endereços de memória de 32 bits.

**2. ESPECIFICAÇÃO**

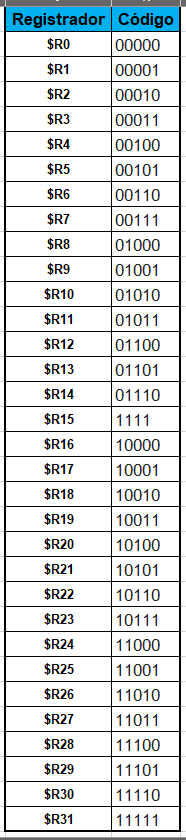
No projeto da CPU foi utilizado 5 registradores ao todo que são maiormente utilizados no armazenamento dos dados na passagem de um estágio para o outro. Formato e tipo das instruções utilizadas e aceitas pela CPU construída foram:

|  |  |  |
| --- | --- | --- |
| **Instrução do Tipo R** | | |
| **Campos** | **Bits** | **Notas** |
| OPCODE | 6 | Operação básica da instrução (opcode) |
| RS | 5 | O primeiro registrador fonte |
| RT | 5 | O segundo registrador fonte |
| RD | 5 | O registrador destino |
| SHAMT | 5 | Shift Amount, para instruções de deslocamento |
| FUNCT | 6 | Function. Seleciona variações das operação especificada pelo opcode |

|  |  |  |
| --- | --- | --- |
| **Instrução do Tipo I** | | |
| **Campos** | **Bits** | **Notas** |
| OPCODE | 6 | Operação básica da instrução (opcode) |
| RS | 5 | O primeiro registrador fonte |
| RT | 5 | O segundo registrador fonte |
| ENDEREÇO | 16 | Endereço da instrução |

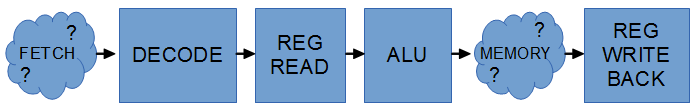
|  |  |  |
| --- | --- | --- |
| **Instrução do Tipo J** | | |
| **Campos** | **Bits** | **Notas** |
| OPCODE | 6 | Operação básica da instrução (opcode) |
| ENDEREÇO | 16 | Endereço da instrução |

Foram utilizados vários registradores para armazenamento dos dados, abaixo temos a tabela de registradores e seus códigos:



A implementação do MIPS desenvolvida neste trabalho cobre um subconjunto das instruções do MIPS. Como podemos notar, as instruções do tipo R possuem um campo apenas para determinar qual operação será executada pelo ULA (Unidade Lógica Aritmética) da CPU.

**3- DESENVOLVIMENTO**

[](http://labs.domipheus.com/blog/wp-content/uploads/2015/07/pipe.png)Funcionamento dos estágios de processamento de uma instrução:

Após o estudo da organização e arquitetura do processador, desenvolveu-se separadamente cada um dos componentes que compõem os estágios do pipeline do MIPS. Depois de testados individualmente, os estágios foram conectados e testados novamente.

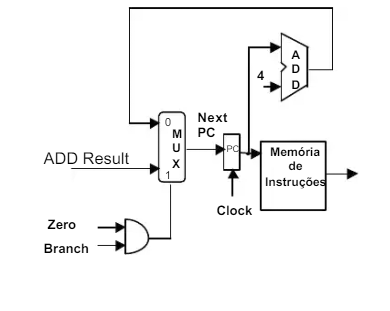
A implementação desenvolvida no trabalho trabalha com 5 estágios, o primeiro estágio de *fetch* (busca) da instrução, o estágio de decodificação da instrução, o de execução e o de acesso a memória e escrita de volta nos registradores (*Writeback*).

**Desenvolvimento de cada estágio:**

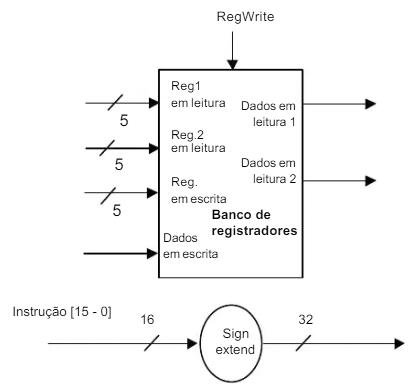
**3.1 ESTAGIO DE BUSCA**

Este estágio é responsável pela busca da instrução que será executada, a partir da ordem definida no programa. Nele também é realizado a incrementação do PC, somando 4 para já calcular o endereço da próxima instrução a ser executada.

Representação no circuito:

****

**3.2 Estagio de Decodificação**

 Neste estágio realiza-se a decodificação da instrução, e ele também é responsável pelo banco de registradores. O banco de registradores foi implementado como uma entidade separada, podendo ser determinado na instanciação o número de registradores. Representação no Circuito:

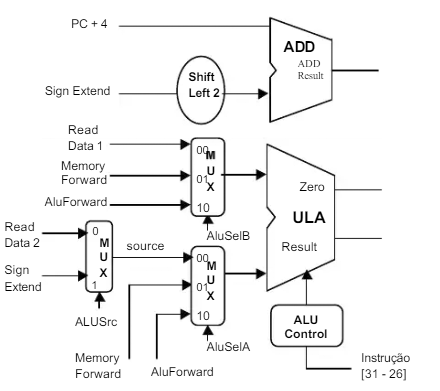
O banco de registradores implementado é um componente composto por um conjunto de registradores que podem ser acessados de forma organizada. podem ser executadas operações de leitura dos dados anteriormente gravados e de escrita de dados para modificar as informações internas. O banco de registradores é composto por 32 registradores de 32 bits cada.

**3.3 ESTÁGIO DE EXECUÇÃO**

Neste estágio, dependendo da instrução, existem duas unidades que as podem executar, a unidade para calculo de endereço de *branch* (apenas soma), e a ULA para realizar as operações aritmética e logica.

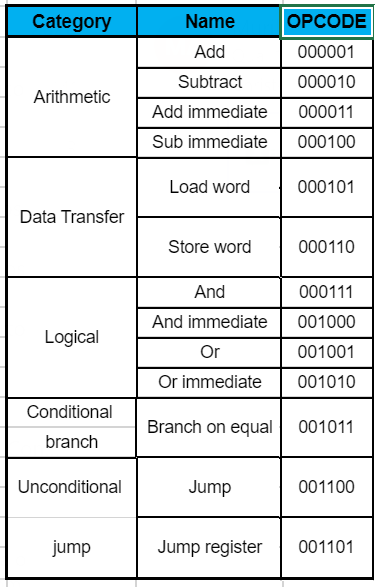
Segundo os sinais de controle gerados pela Unidade de Controle, poderá ser selecionado o segundo operando da ULA, que pode ser um dado lido de um registrador, como nos casos das instruções tipo R e de *branch,* ou um endereço, como nas instruções de *load* e *store*.

Representação no circuito:



**3.3.1 UNIDADE LÓGICA ARITMÉTICA**

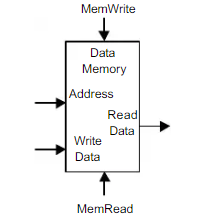
O controle da ULA tem como entrada o opcode, os 6 bits mais significativos da palavra de instrução e mais os 6 bits menos significativos da instrução, gerando a partir destes um sinal de três bits para a ULA.

Abaixo está o modo definido pelo grupo para os *opcodes* das instruções aceitas pela ULA construída no projeto.

**3.4 ESTÁGIO DE MEMÓRIA**

Neste estágio ocorre a leitura e/ou a escrita da na memória no mesmo ciclo. A escrita só é realizada no final do ciclo e se o sinal for recebido o sinal de *MemWrite*, que indica uma instrução de *store*.

Representação no circuito:



**3.5 ESTÁGIO DE *WRITEBACK***

Nesta última etapa, o resultado da ULA é escrito no registrador de destino.

Os sinais de controle, gerados na UC (unidade de controle) definem de onde virá o valor de escrita (memória de dados ou resultado da ULA) e se haverá escrita no banco de registradores.

**3.6 UNIDADE DE CONTROLE**

É responsável por gerar os sinais de controle adequados, para ser feito o que se pede na instrução que será executada. Sinais de controle da CPU com pipeline:

RegWrite - sinal que habilita a escrita no banco de registradores;

MemToReg - sinal que indica que o dado a ser escrito no registrador de destino é um dado vindo da memória;

AluSrc - sinal que indica qual o segundo operando da ULA;

MemWrite - sinal que habilita a escrita na memória;

MemRead - sinal que habilita a leitura da memória;

Branch - sinal que indica uma instrução de branch;

RegDst - sinal utilizado para selecionar o registrado destino.

AluOp1 – primeiro operando da ULA.

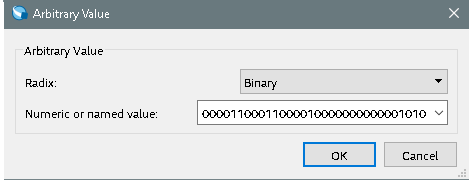
AluOp2 – segundo operando da ULA.

Jump – se a instrução for tipo J, o valor escrito no PC será o que foi incrementado em 4 no somador.

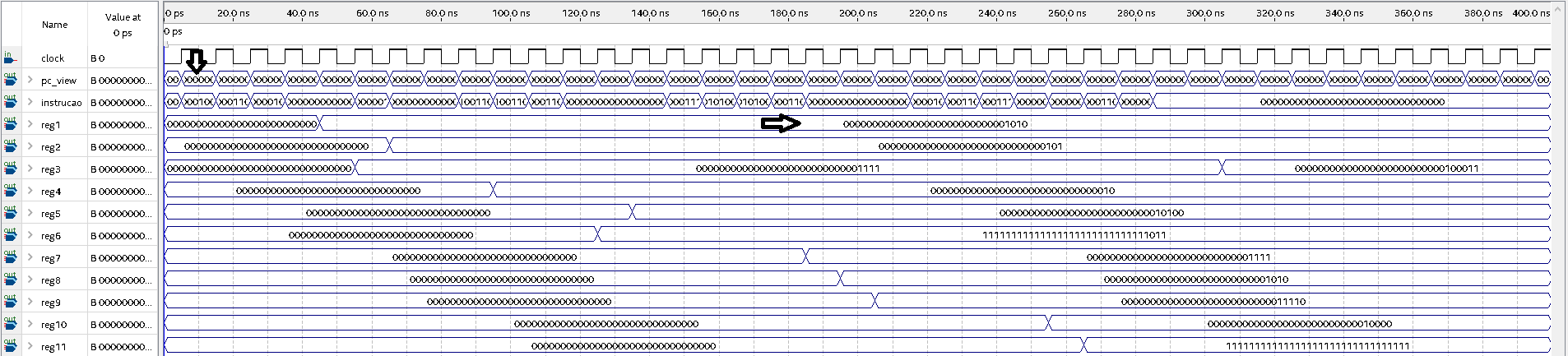
1. **RESULTADOS**

**4.1 TESTES REALIZADOS**

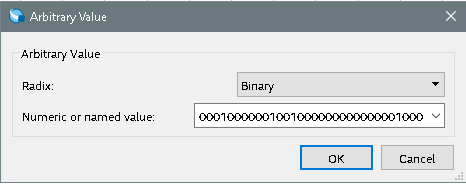
**--Instrução ADDI**



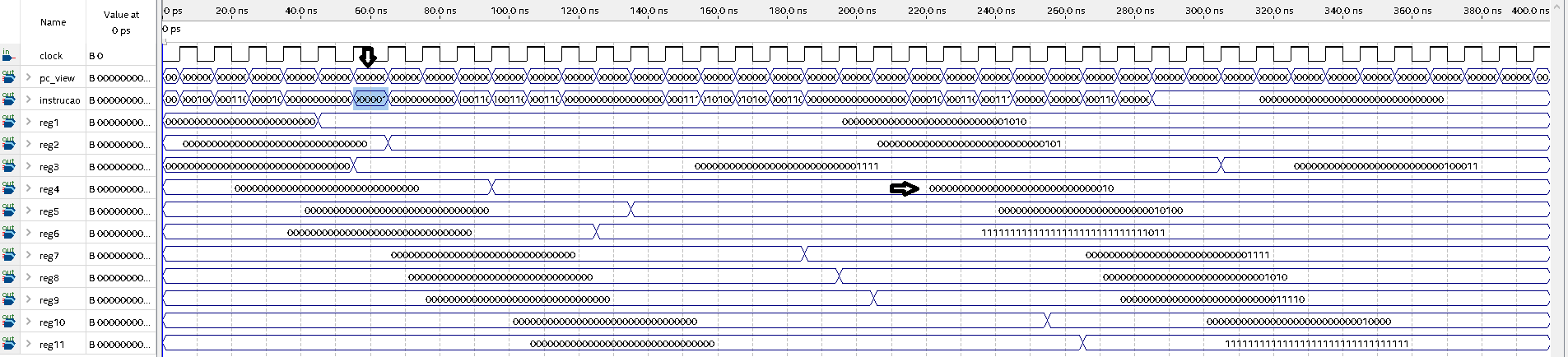
**-- ADDi $1, $3, 10**



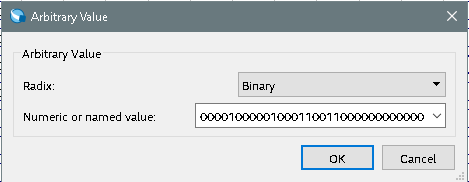
**-- Instrução SUBI**



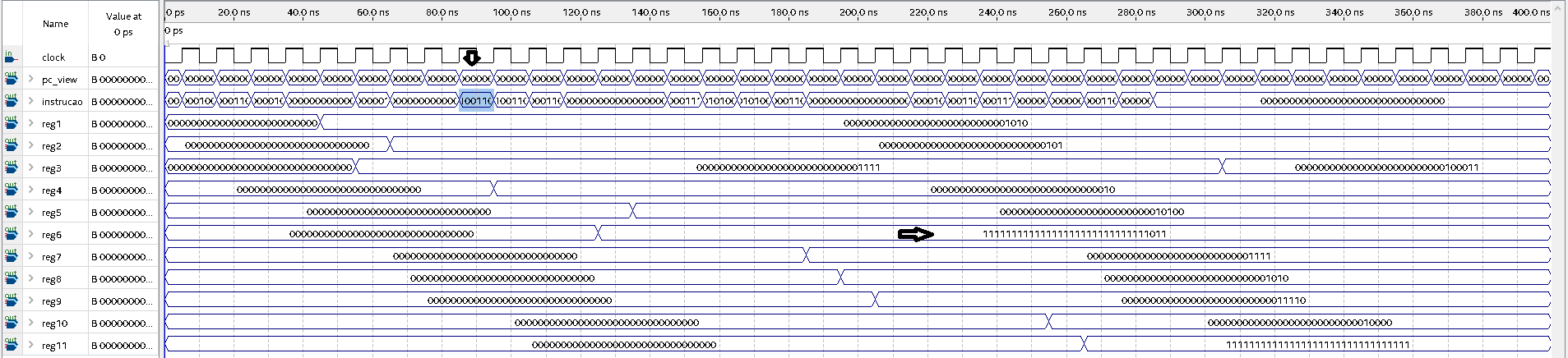
-- **SUBi $4, $1, 8**



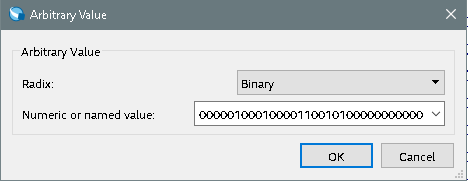
**-- Instrução SUB**



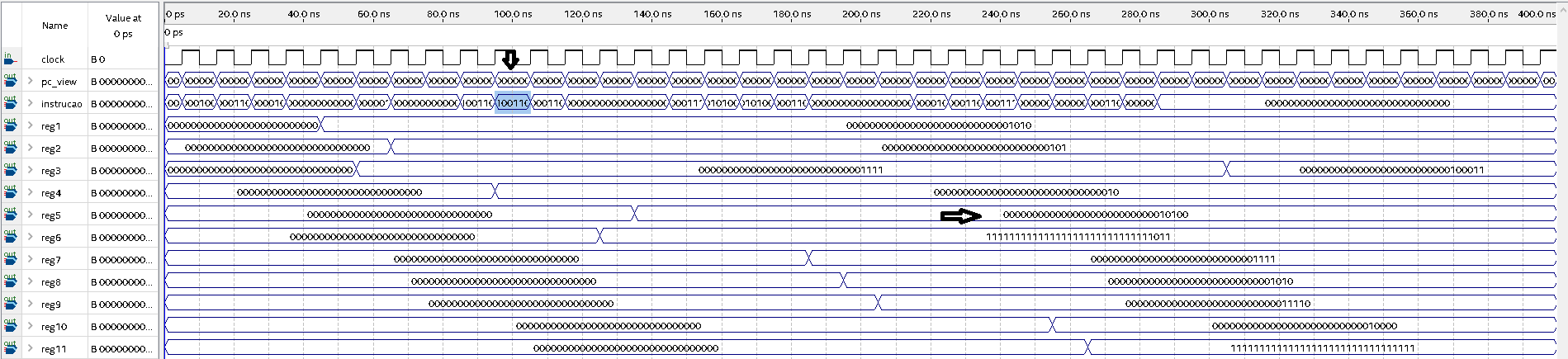
**-- SUB $6, $1, $3**



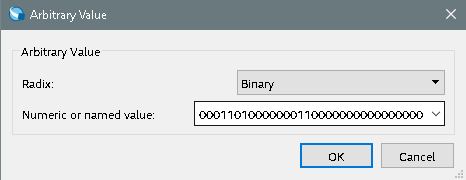
**--Instrução ADD**



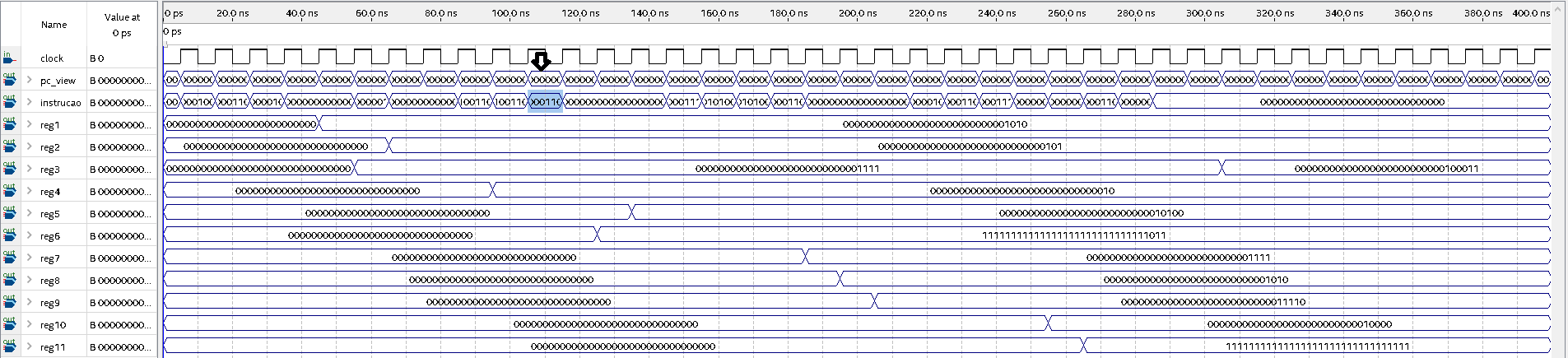
**-- ADD $5, $2, $3**



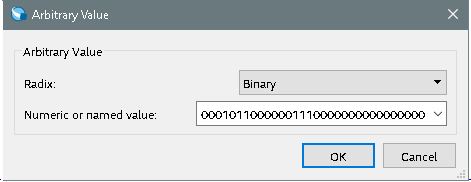
**--Instrução SW**



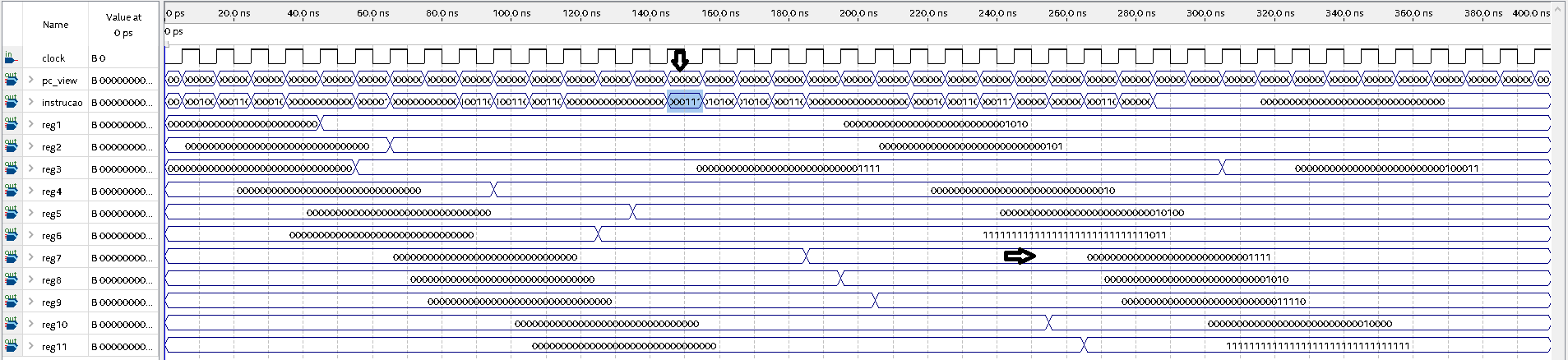
**-- SW $3, 0($16)**



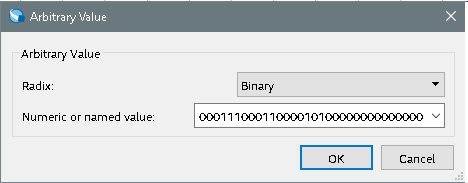
**-- Instrução LW**



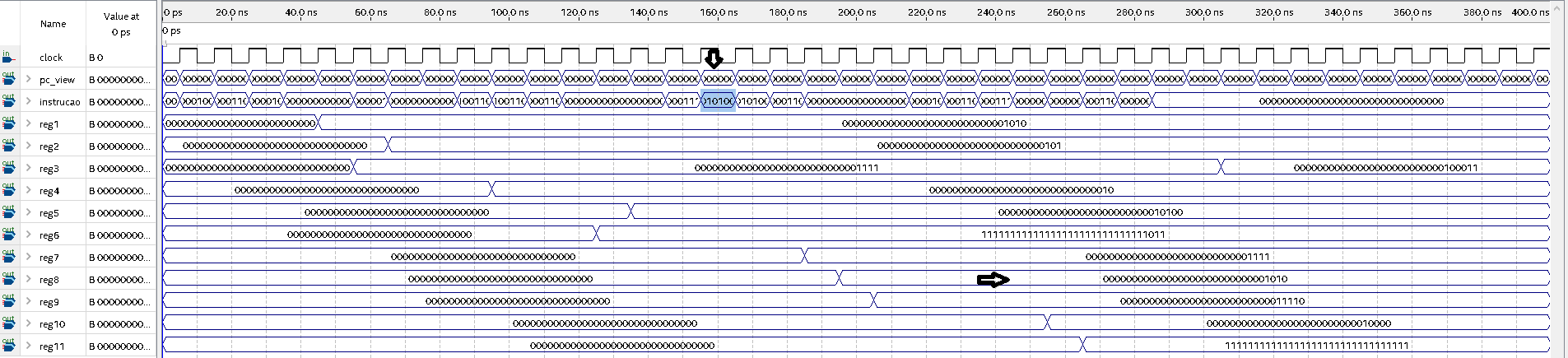
**-- LW $7, 0($16)**



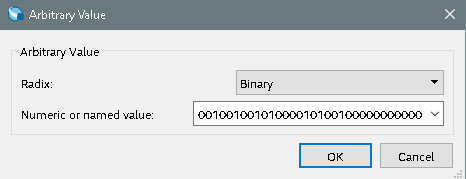
**-- Instrução AND**



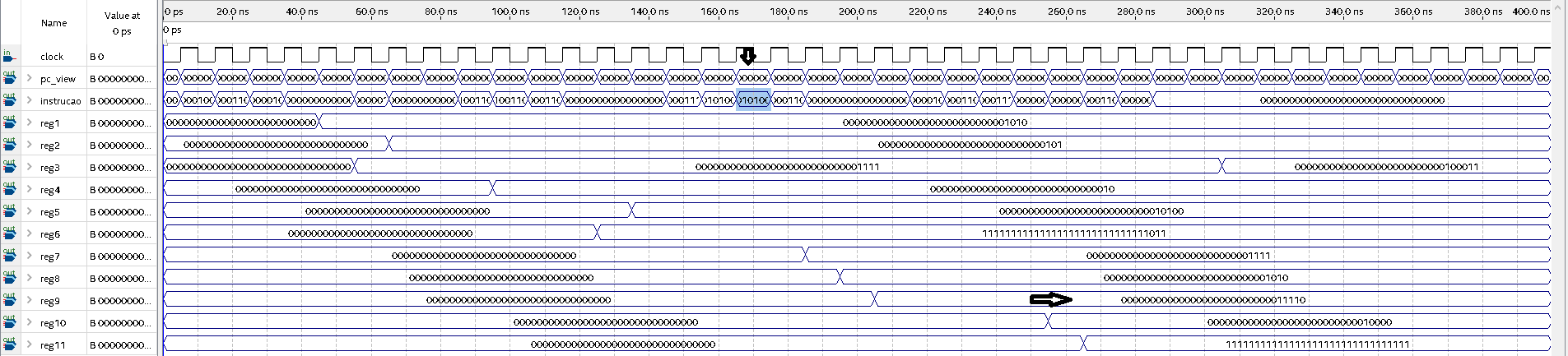
**-- AND $8, $3, $1**



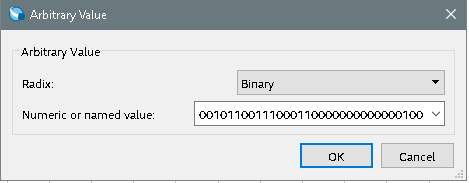
**-- Instrução OR**



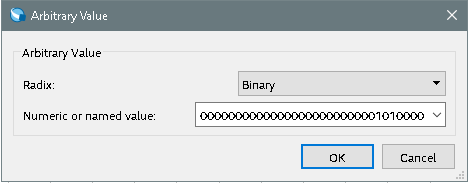
**-- OR $9, $5, $1**



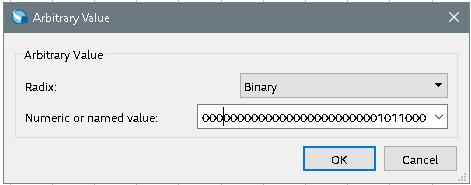
**-- Instrução BEQ**



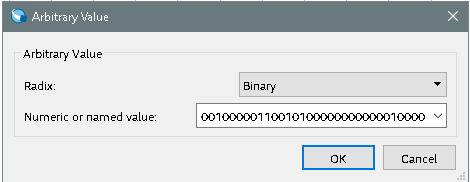
**-- BEQ $7,$3 antes do salto**



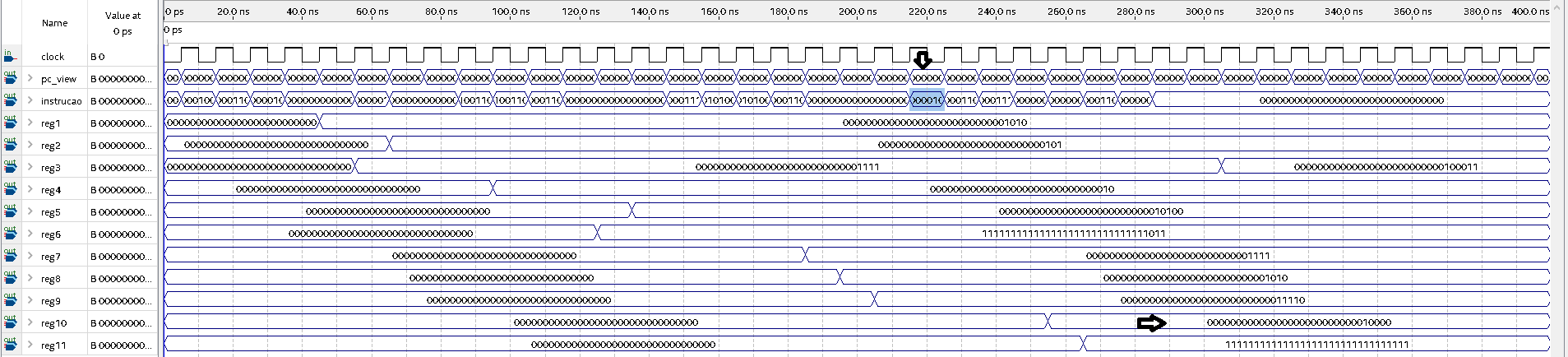
**-- BEQ $7,$3 depois do salto**



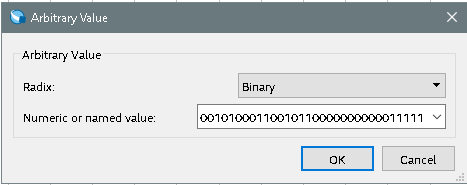
**-- Instrução ANDi**



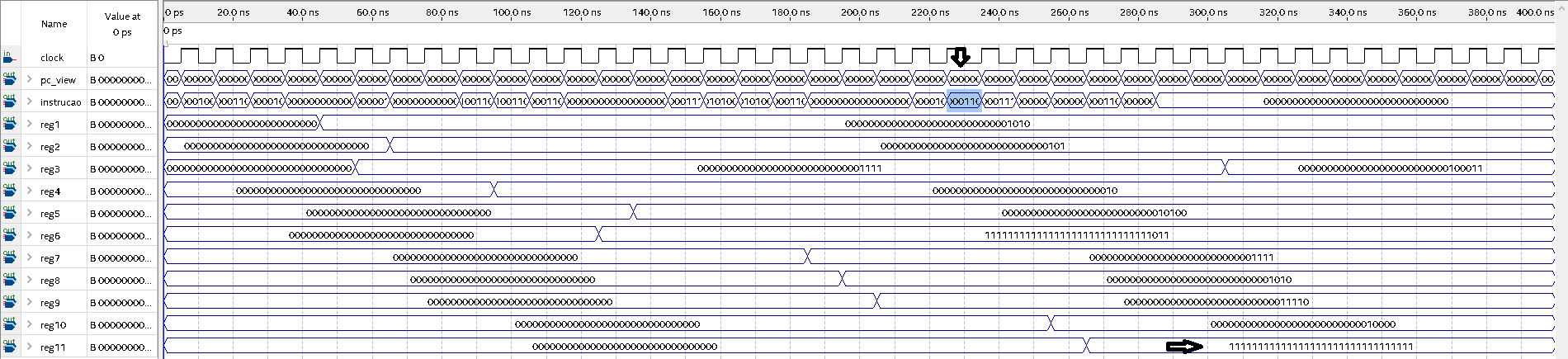
**-- ANDi $10, $6, 16**



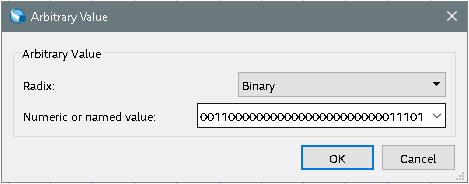
**-- Instrução ORi**



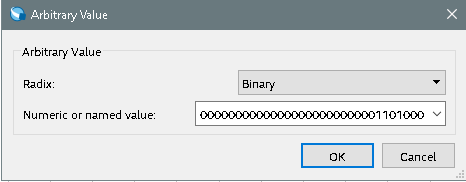
**-- ORi $11, $6, 31**



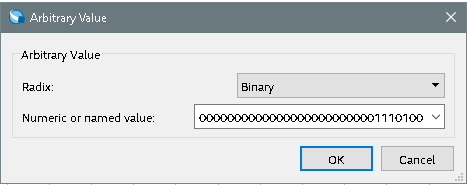
**-- Instrução Jump**



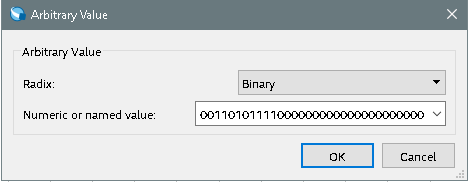
**-- Jump (116) antes do salto**



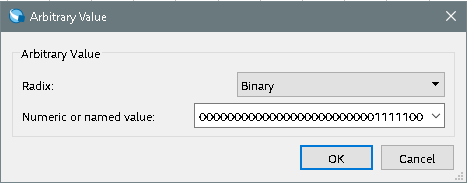
**-- Jump (116) depois do salto**



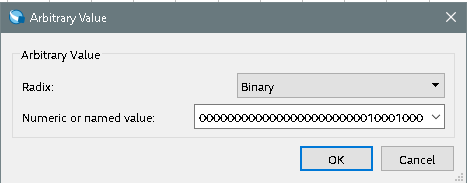
**--Instrução Jump Register**



**-- Jump Register (15) antes do salto**



**-- Jump Register (15) depois do salto**

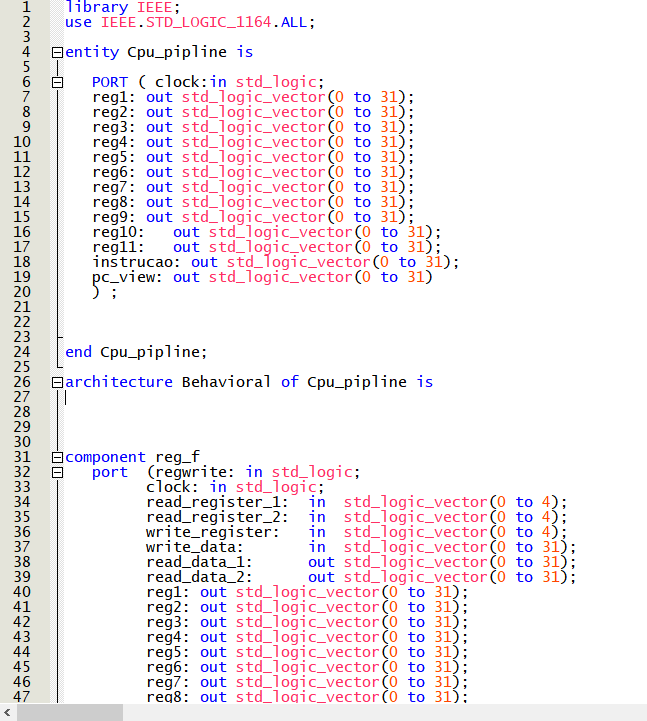


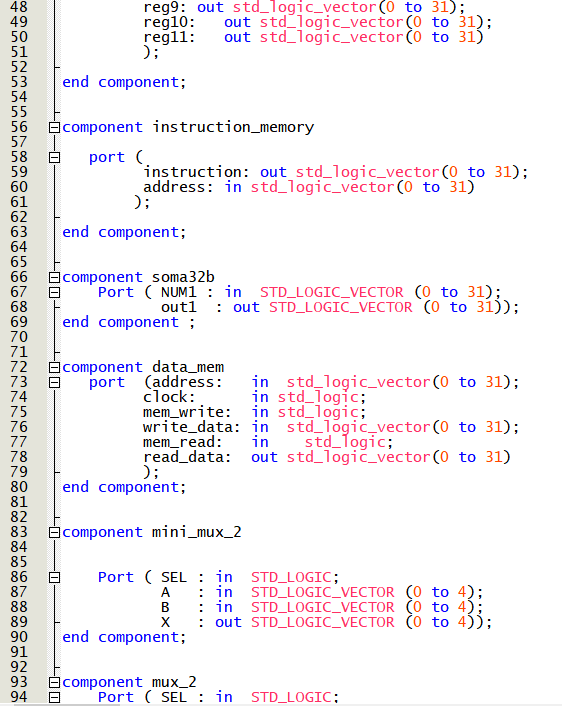
1. **CONCLUSÃO**

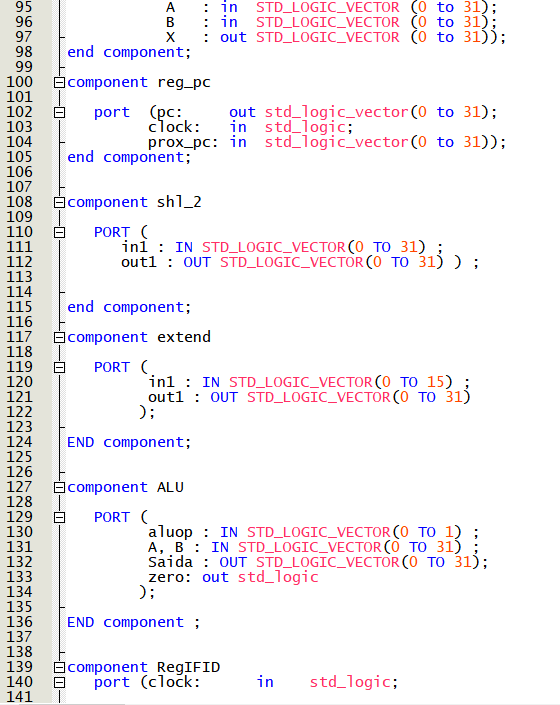
Neste trabalho foi possível estar aprimoramento os conhecimentos e manipulação com a linguagem de programação de VHDL. Podendo conhecer mais a fundo a aplicação de uma CPU e seus principais componentes, como ele funciona e como se comporta, podendo analisar as instruções e seus formatos e as suas execuções, observando o funcionamento e armazenamento nos registradores passo a passo no ciclo em pipeline com ajuda de “debug” criando no programa no waveform, observando também as saídas de cada componente e o que estará sendo salvo nos registradores que forem solicitados.

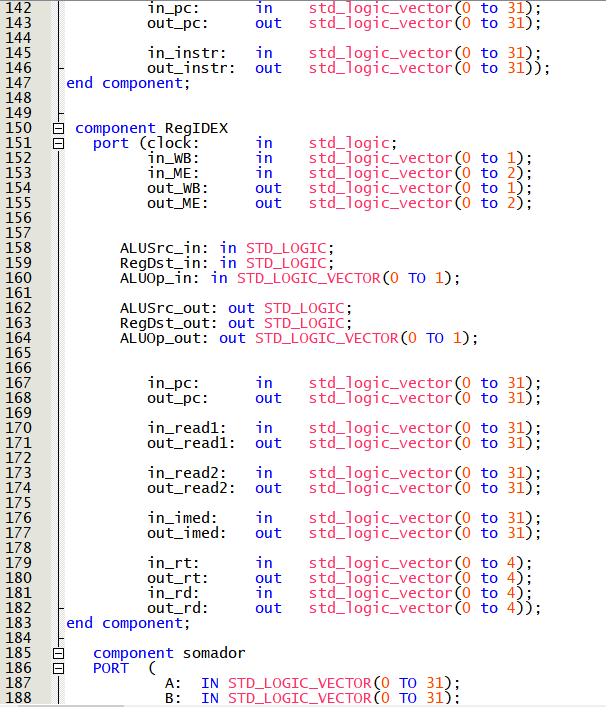
1. **ANEXO: CÓDIGO VHDL**

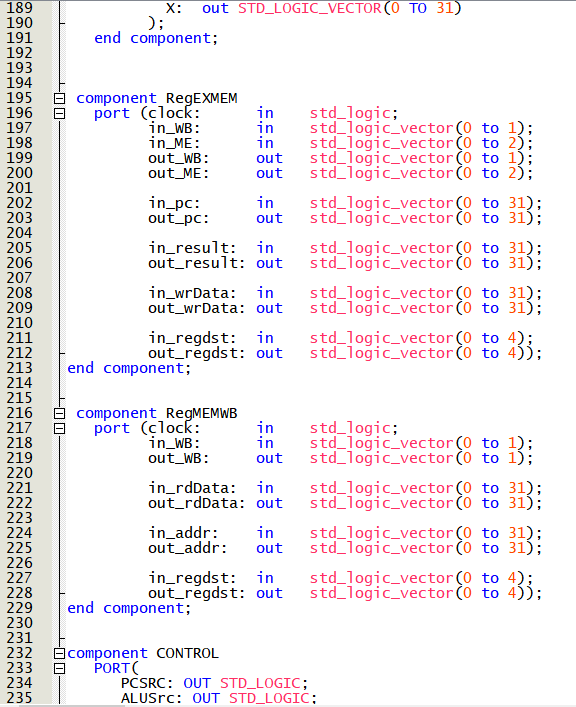
**CPU:**

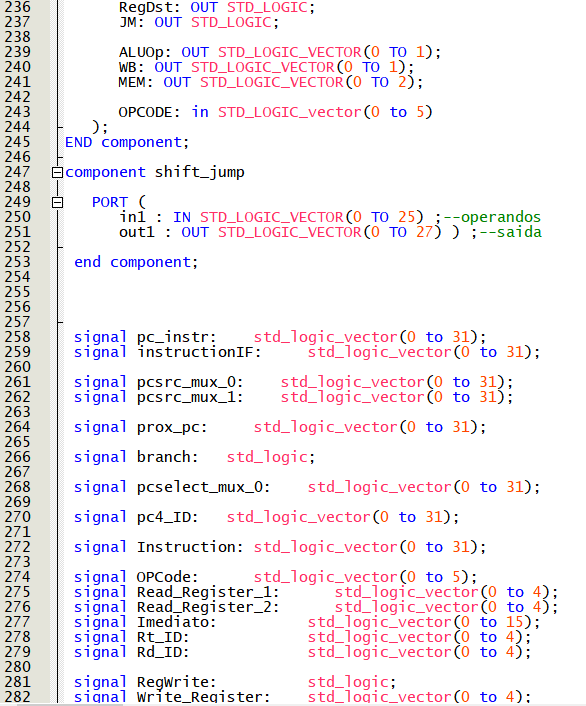


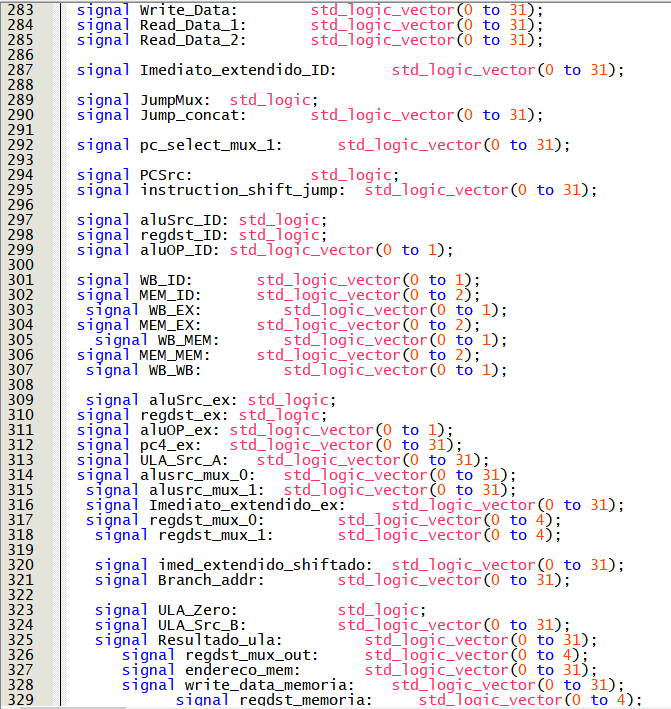




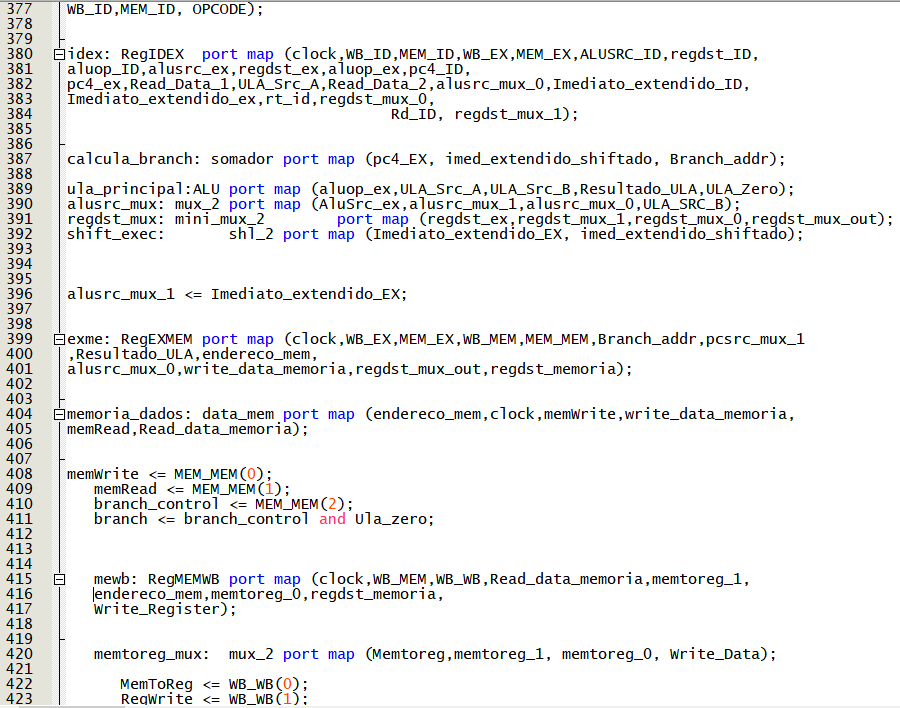


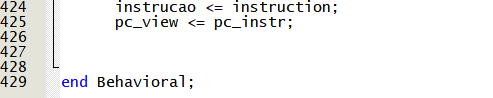




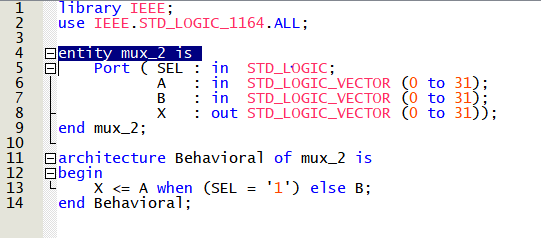




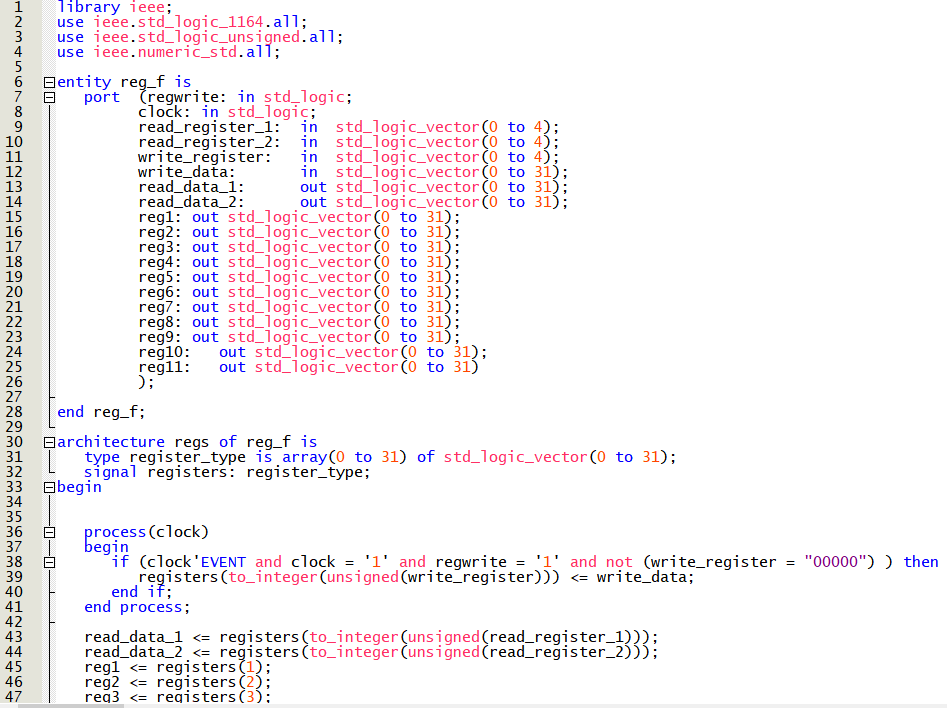




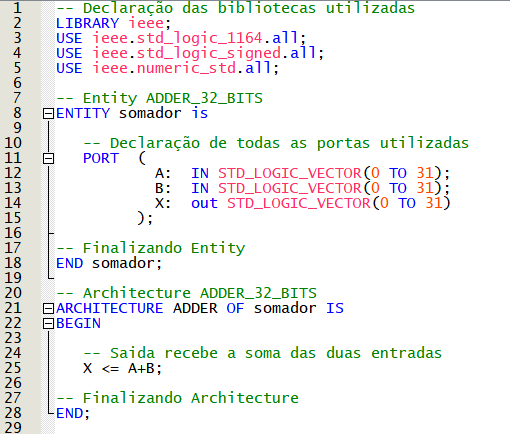
**MUX\_2:**



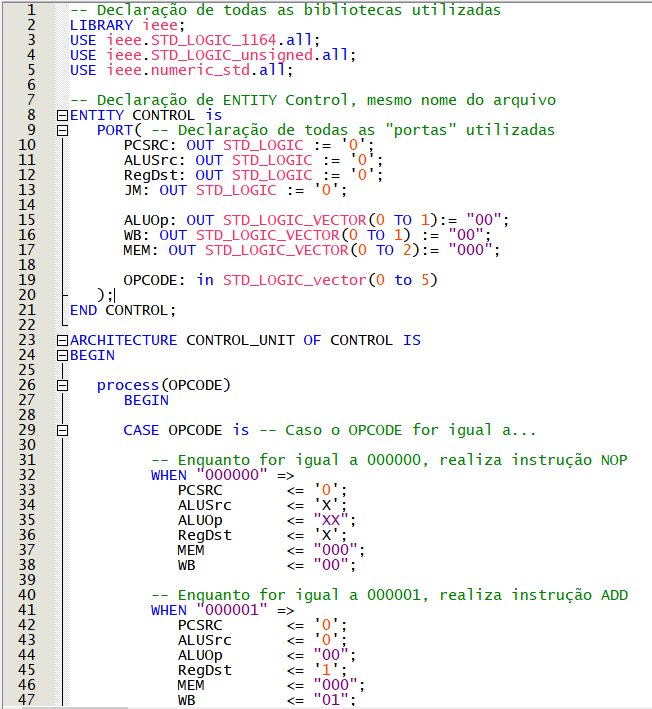
**REG\_F:**

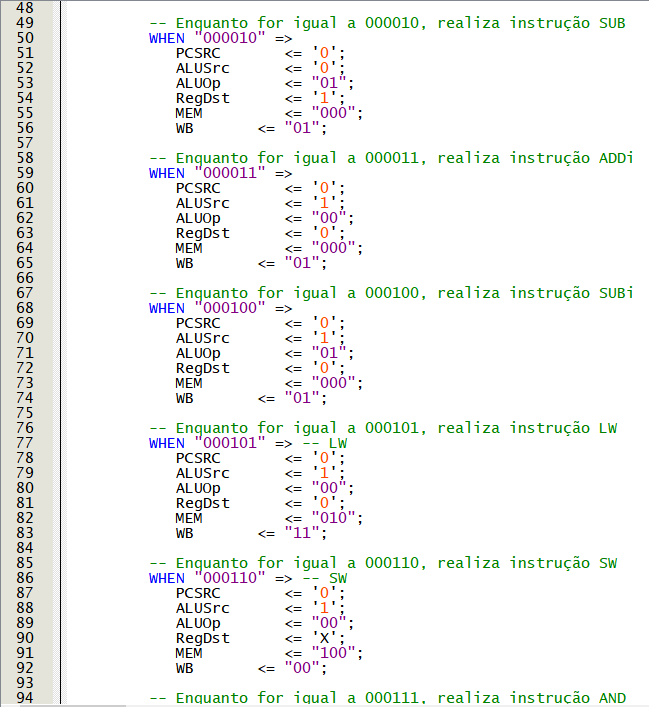


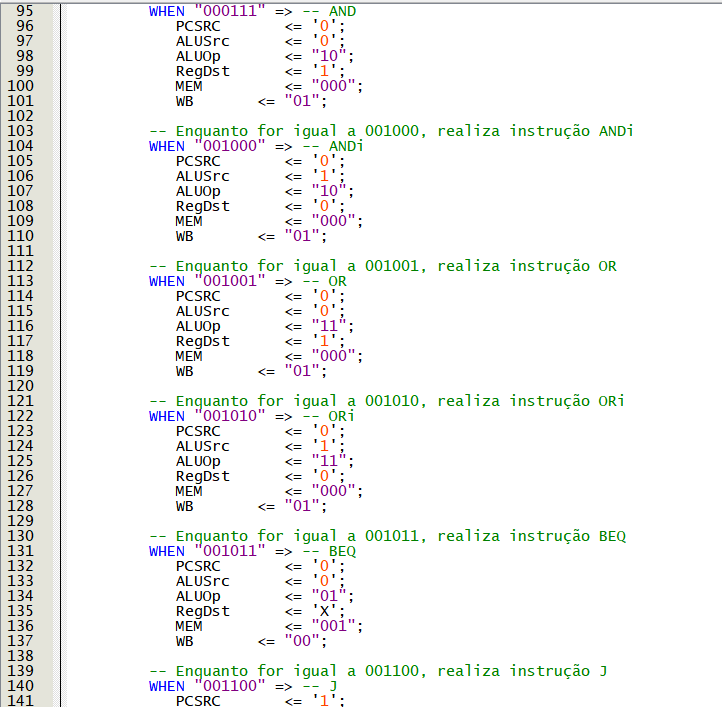
**SOMADOR:**

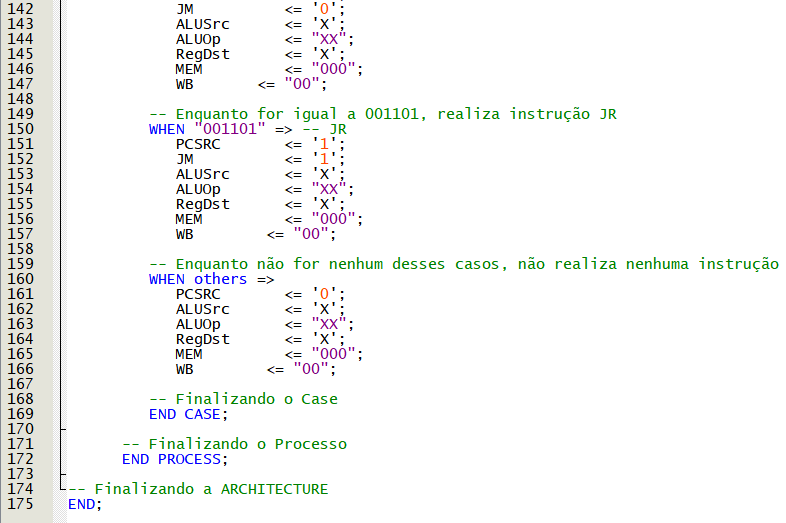


**CONTROL:**

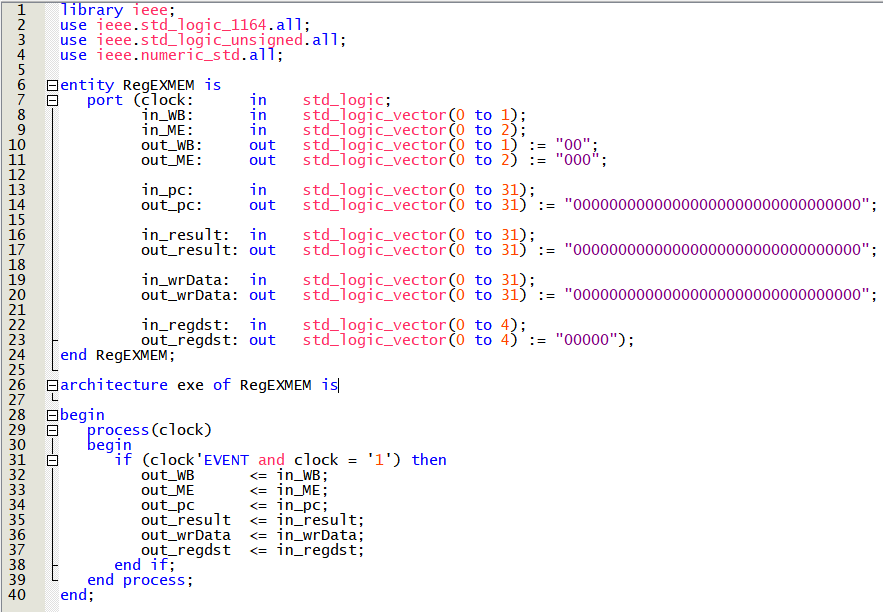




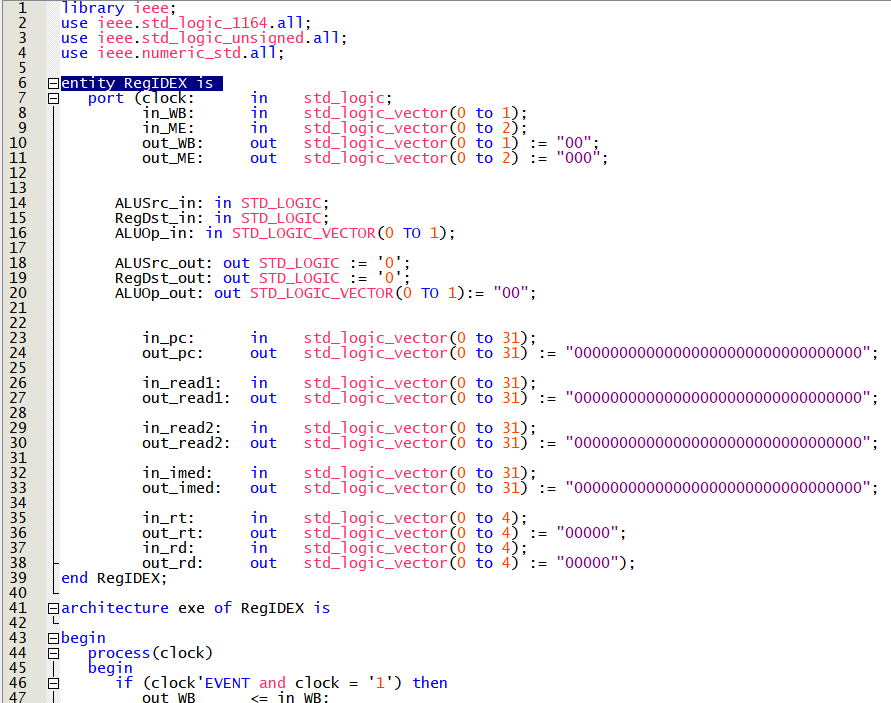


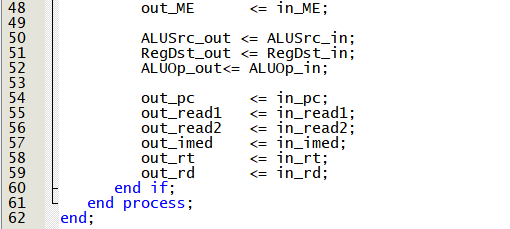


**RegEXMEM:**

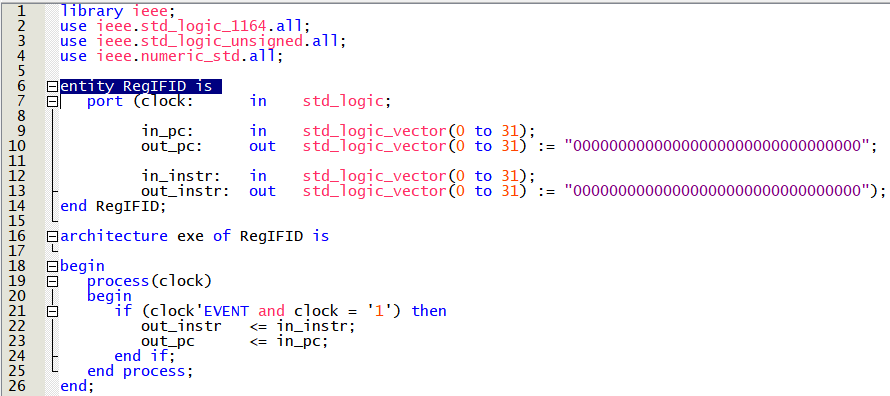


**RegIDEX:**

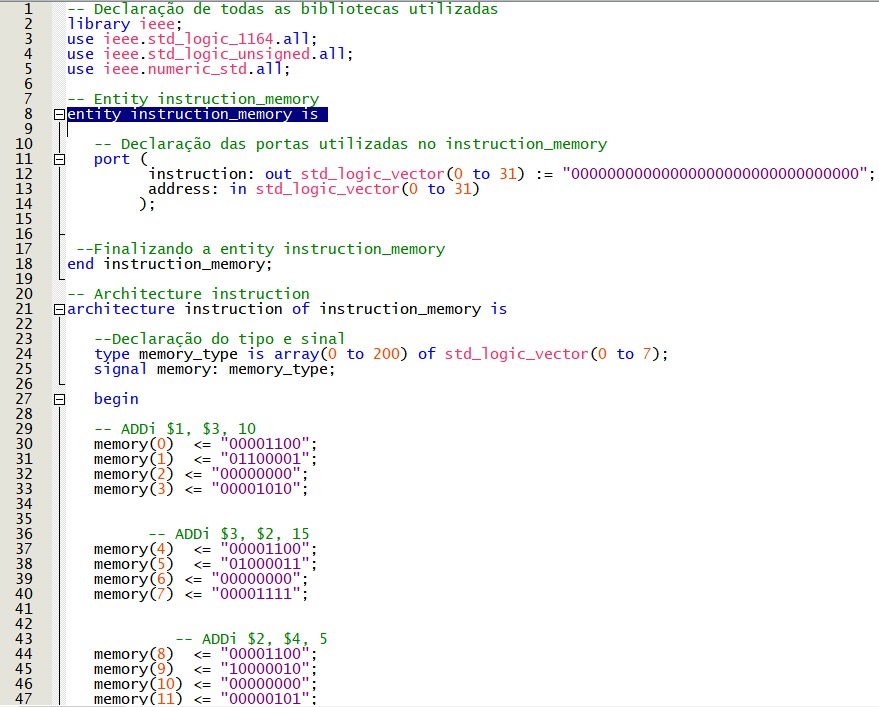


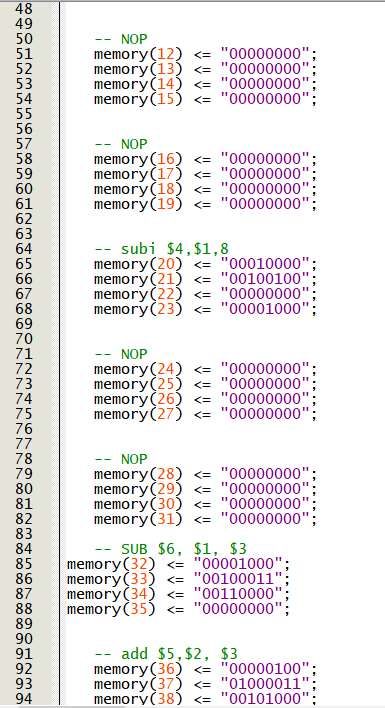


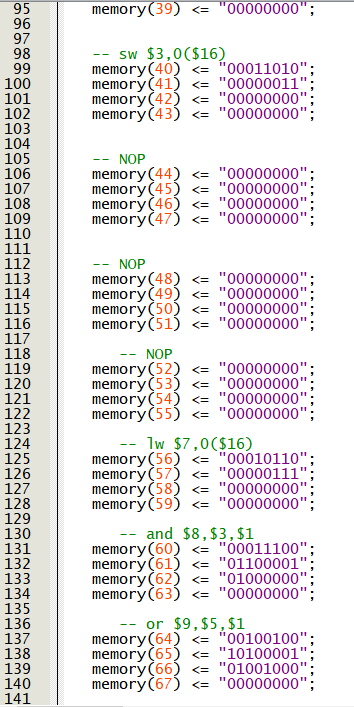
**RegIFID:**

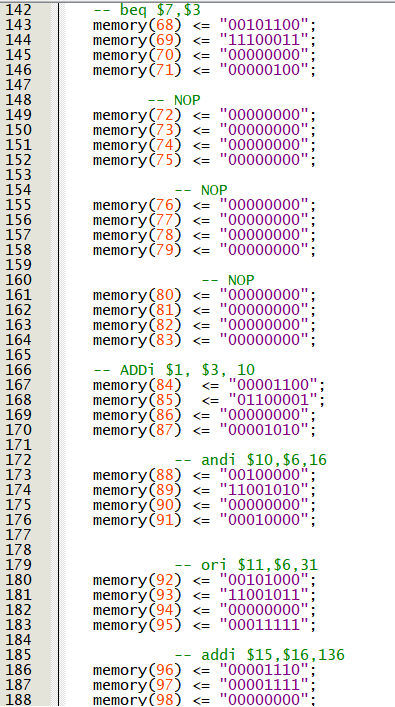


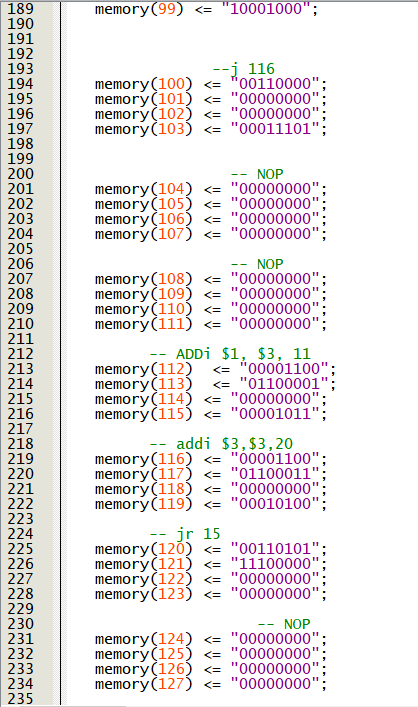
**INSTRUCTION MEMORY:**









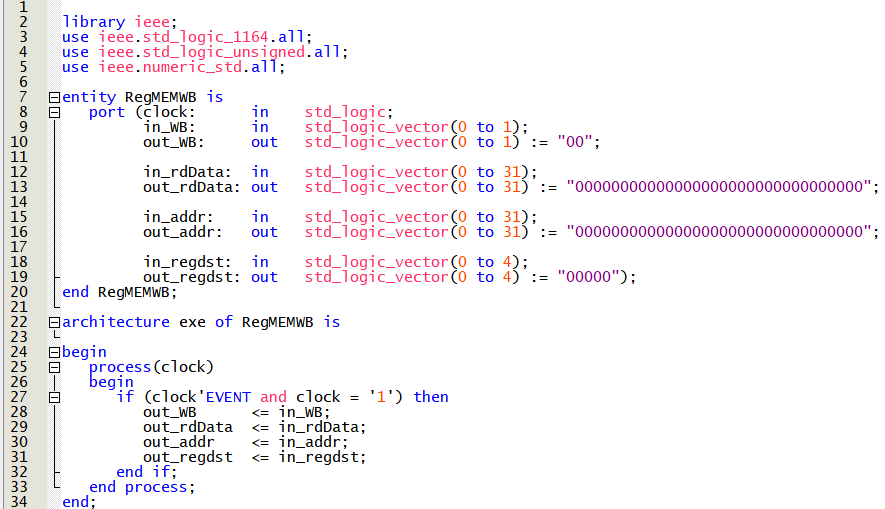




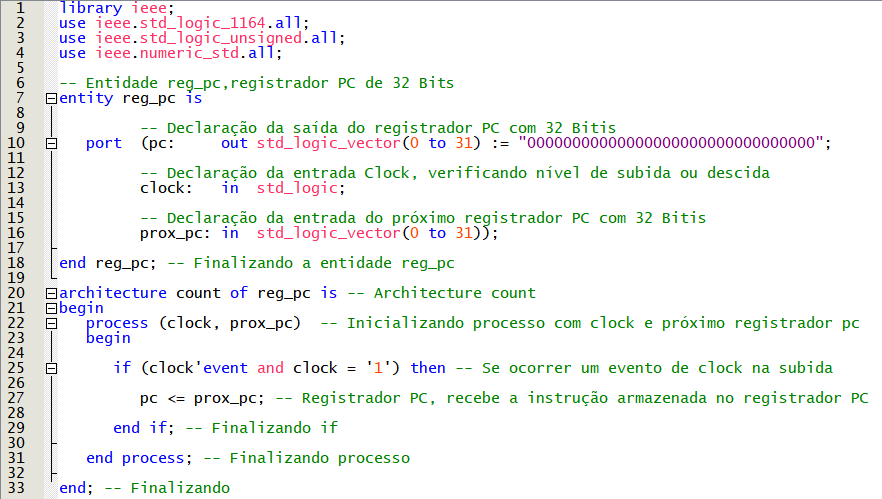
**DATA\_MEM:**



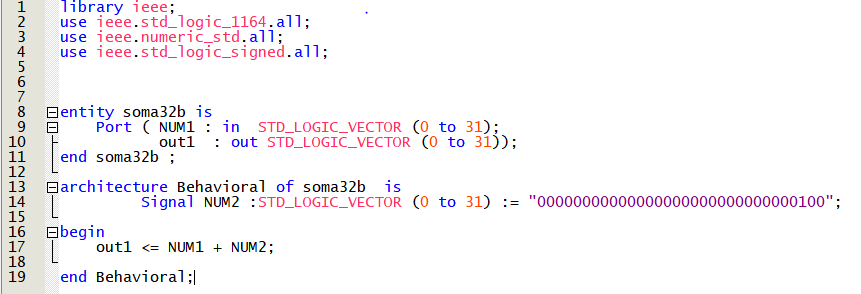
**REGMEMWB:**



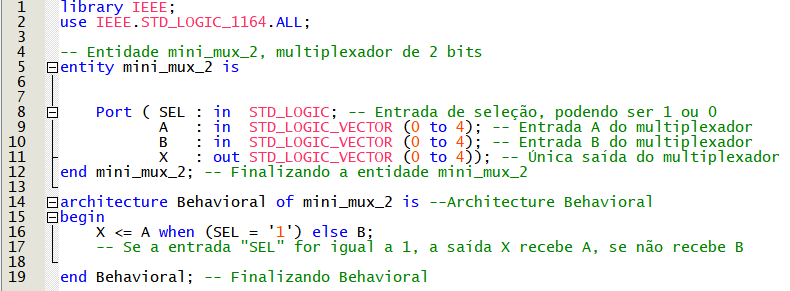
**REGPC:**



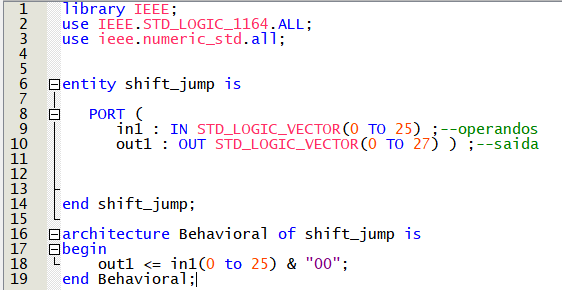
**SOMA32B:**



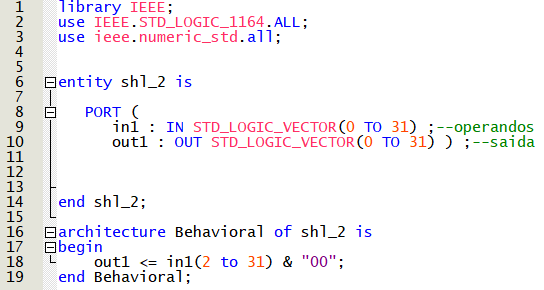
**MINI\_MUX:**



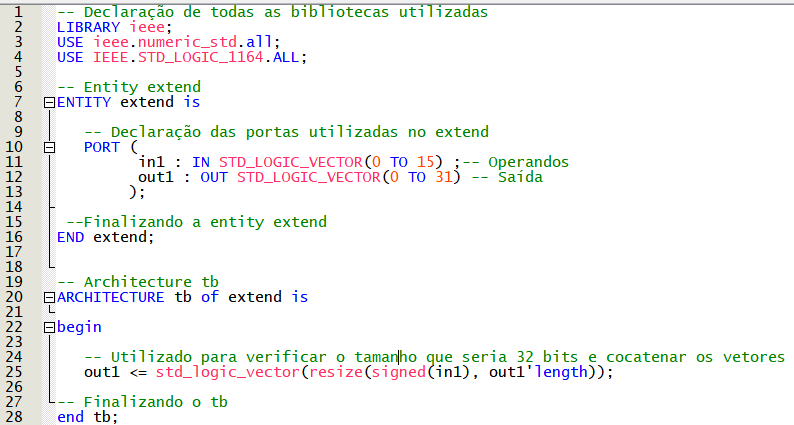
**SHIFT\_JUMP:**



**SHIFTL\_2:**



**EXTEND:**



**ALU:**

