# Projeto Final
# ZeptoProcessador-III de 16 Bits

**Matheus Cardoso de Souza, 202033507**
**Ualiton Ventura da Silva, 202033580**
**Grupo G42**

[1]Dep. Ciência da Computação – Universidade de Brasília (UnB)
CIC0231 - Laboratório de Circuitos Lógicos

matheus-cardoso.mc@aluno.unb.br, 202033580@aluno.unb.br

***Abstract.*** *The present report will show a detailed description of all the needed process to build a processor, one of the most important logic circuits today. Also, the processors' functions will be assessed when submitted to execution of arbitrary code, therefore showing their generality in a wide variety of activities.*

***Resumo.*** *O presente relatório apresentará uma descrição detalhada de todos os processos que abrangem a construção de um processador, um dos circuitos lógicos mais importantes na atualidade. Também será abordado o comportamento do processador quando submetido à execução de códigos arbitrários inseridos pelo usuário, dessa forma demonstrando a generalidade para uso em diversas atividades.*

## 1. Introdução

Na atualidade, com o advento dos computadores, um dos circuitos mais improtantes para a sociedade moderna se tornou os processadores. Tais circuitos apresentam-se como um dos mais complexos na área de circuitos lógicos. Para o presente relatório, entretanto, abordaremos a construção de um relativamente simples, contendo apenas um registrador de instruções, uma memória de instruções que armazena instruções de $32$ *bits*, e um banco de registradores de $16$ registradores contendo $16$ bits cada.

### 1.1. Objetivos

Temos como objetivo neste projeto final desenvolvermos um processador (chamado *ZeptoProcessador-III*), com as características de ser um processador de $16$ *bits*, com memória de instruções própria, suportando instruções de $32$ *bits*. O processador terá um conjunto de $13$ instruções, a saber: $\{$*add*, *subi*, *andi*, *ori*, *xori*, *beq*, *bne*, *ble*, *bleu*, *bgt*, *bgtu*, *jal* e *jalr*$\}$

### 1.2. Materiais

Em função da natureza do ensino a distância, os presentes experimentos não foram realizados usando-se materiais e equipamentos físicos, mas sim emulados por meio do software Deeds.

A seguir estão enumerados os materiais utilizados:

- Software Deeds
- Portas lógicas
    - Portas Lógicas *AND*
    - Portas Lógicas *NAND*
    - Portas Lógicas *NOR*
    - Portas Lógicas *OR*
- *Clocks*
- Display de saída de 1 bit
- Display de saída de 8 Segmentos
- Memórias *ROM*
- Multiplexadores
- Unidades Lógico-Aritméticas - *ULA*s

## 2. Procedimentos

Como o processador é um circuito complexo, optou-se por abordar a implementação de cada bloco lógico de forma separada, e, posteriormente, mostrar como esses blocos individuais podem ser integrados para resultar em um circuito funcional. Começaremos demonstrando a montagem do circuito do principal registrador do processador, o registrador *PC*.

### 2.1. Montagem do Circuito do Processador *PC*

Todo processador deve possuir um registrador principal que sirva de "maestro" do circuito como um todo. Este processador, também chamado de processador de instruções, tem a funcionalidade de armazenar a instrução que está atualmente sendo decodificada e/ou executada. Como o processador ZeptoProcessador-III é bem simples, precisamos de apenas um registrador desse tipo para obtermos os resultados esperados. Entretanto, em processadores mais modernos e complexos (vide [Wikipedia 2021a]), vários processadores de instrução podem ser acoplados para formarem uma *pipeline* de instruções, onde cada parte da *pipeline* é responsável por parte do processo de decodificação, preparação ou execução das instruções passadas.

Como é possível observar pela imagem 1, o registrador de instruções (no centro da imagem), tem como input o resultado do último ciclo de instruções. Além disso, está diretamente ligado ao sinal de clock que sincroniza todos os componentes do processador, e também a um sinal de reset (na imagem com o nome de *Start*). Seu output é utilizado como entrada para a memória de instruções (para obter a próxima intrução que deve ser processada), e também utilizada em um somador (aqui não mostrado devido à distância entre o registrador e o somador), para calcular o próximo endereço de instrução a ser processado.

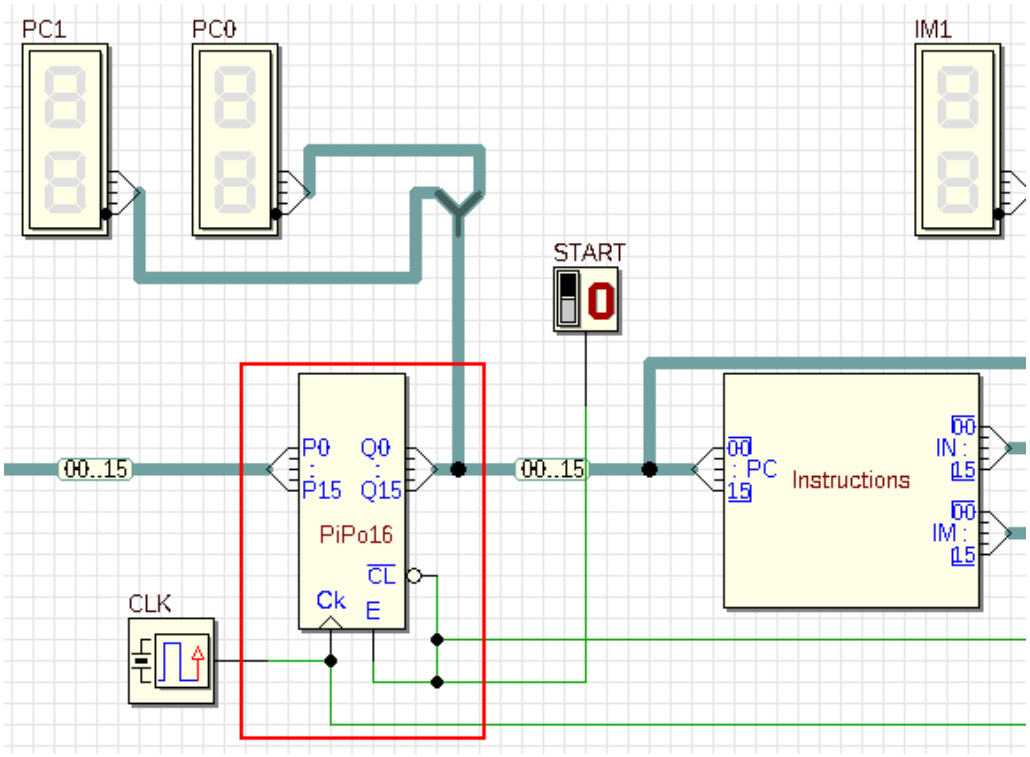


**Figura 1. Zoom no Registrador de Instruções**

O processador ZeptoProcessador-III segue a mesma ideia de execução do processador *MIPS* apresentado na figura 2 (fonte: [Harris and Harris 2012]). Então é possível ver todas as ligações relevantes relacionadas ao registrador de instruções.

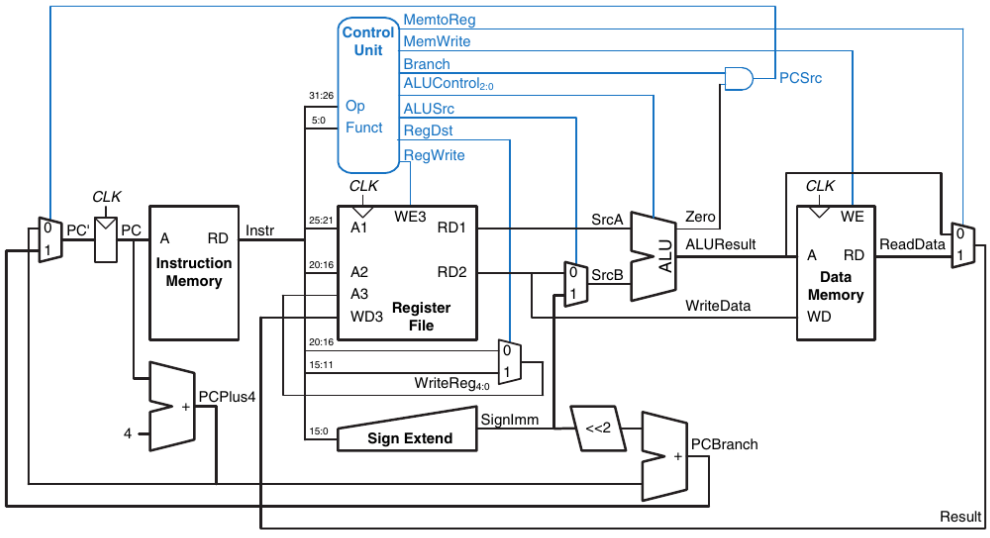


**Figure 7.11 Complete single-cycle MIPS processor**

**Figura 2. Overview esquemático do processador *MIPS***

## 2.2. Montagem do Circuito da Memória de Instruções

Como foi possível observar no circuito do registrador de instruções, o output daquele circuito se ligava à memória de instruções. Tal componente de um processador server para armazenar as instruções que um usuário do processador deseja que este execute. Dessa forma, o processador se torna *programável*.

No ZeptoProcessador-III, a memória de instruções é implementada utilizando-se a memória ROM. Ela é uma memória somente de leitura, de rápido acesso. Tal tipo de memória foi muito utilizada por exemplo em cartuchos de *video-games*, *CD*s e afins (para maiores informações sobre essa tecnologia, consultar [Wikipedia 2021b])

Sua implementação no circuito integrado do processador é dada conforme a imagem 3. Pela figura, é possível observar que existem duas memórias ROM separadas, que recebem como input o endereço produzido pelo registrador de instruções. O uso de duas memórias ROM separadas se dá devido ao fato de que, segundo as especificações do ZeptoProcessador-III, a memória de instruções deve ser capaz de armazenar instruções de 32 *bits*. Entretanto, como a maior memória ROM disponível no *Deeds* é de 16 *bits*, foi utilizado uma memória ROM para armazenar os 16 *bits* menos significativos, e outra ROM para os outros 16 *bits* mais significativos.

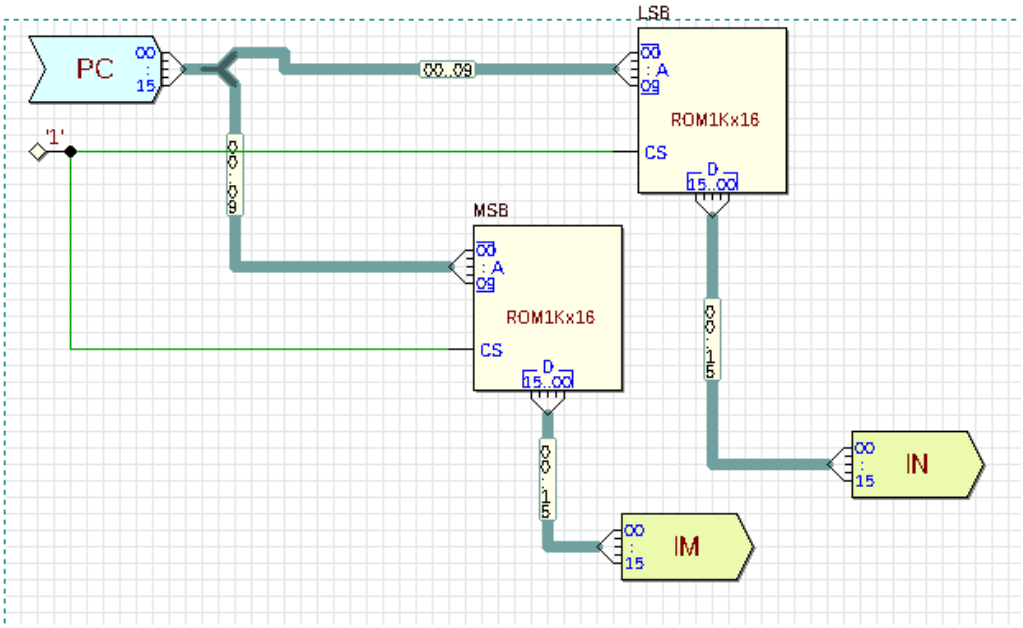


**Figura 3. Overview da Memória de Instruções**

## 2.3. Montagem do Circuito do Banco de Registradores

O próximo bloco de circuito lógico necessário para um processador funcional é o *Banco de Registradores*. Tal circuito possibilita o armazenamento de dados durante a operação do processador, servindo como memória para manipulação de variáveis e controle de estados.

O ZeptoProcessador-III possui como especificação um banco de registradores contendo 16 registradores de 16 *bits* cada. Além disso, esquematicamente o banco de registradores funciona da sequinte forma: 3 canais de input sinalizam respectivamente o endereço

de leitura dos registradores *Ra*, *Rb* e o endereço de escrita do registrador *Rd*. O banco de registradores também possui um canal com 16 *bits* que corresponde aos dados de escrita, o sinal de *clock*, um sinal para informar se o banco de registradores deve escrever os dados de escrita no registrador *Rd*, um sinal de *reset* (que faz com que todos os registradores no banco assumam o valor 0), e por fim também possui dois canais de saída de dados *Sa* e *Sb* que são os dados de leitura dos registradores nos endereços *Ra* e *Rb*, respectivamente. A figura 4 (fonte: [Lamar 2021]) mostra esquematicamente como opera um banco de registradores.

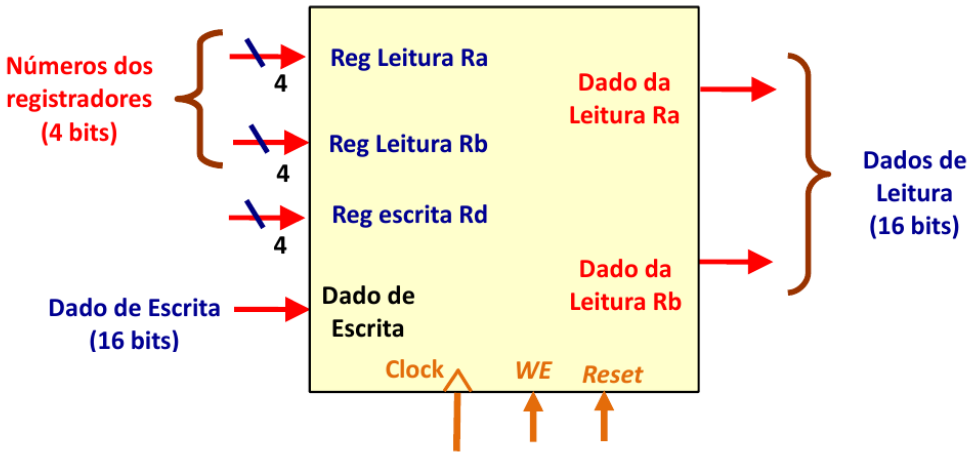


**Figura 4. Esquema do funcionamento de um Banco de Registradores**

E, finalmente, na figura 5 é possível ver como foi organizado o banco de registradores do ZeptoProcessador-III. Por ser um circuito muito grande, as figuras 6 e 7 mostram, respectivamente, mais detalhadamente os circuito dos registradores e dos selecionadores de canais valendo-se de multiplexadores. Novamente, por uma limitação dos blocos pré-existentes no *Deeds*, a seleção de canais foi feita com dois multiplexadores de 8 canais e posteriormente com um multiplexador de 2 canais devido a ausência de um multiplexador que contenha diretamente 16 canais.

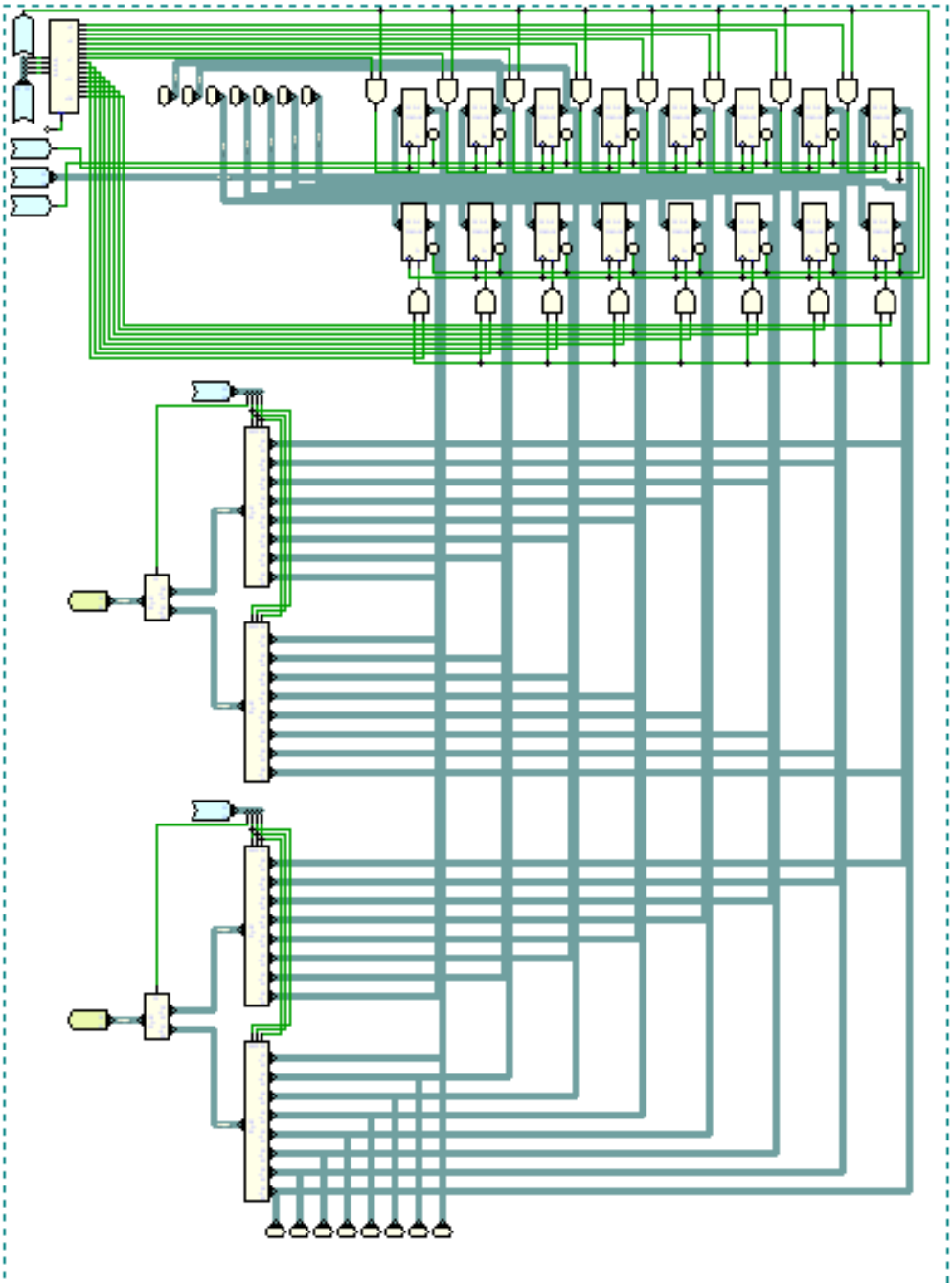

**Figura 5. Montagem do Banco de Registradores**

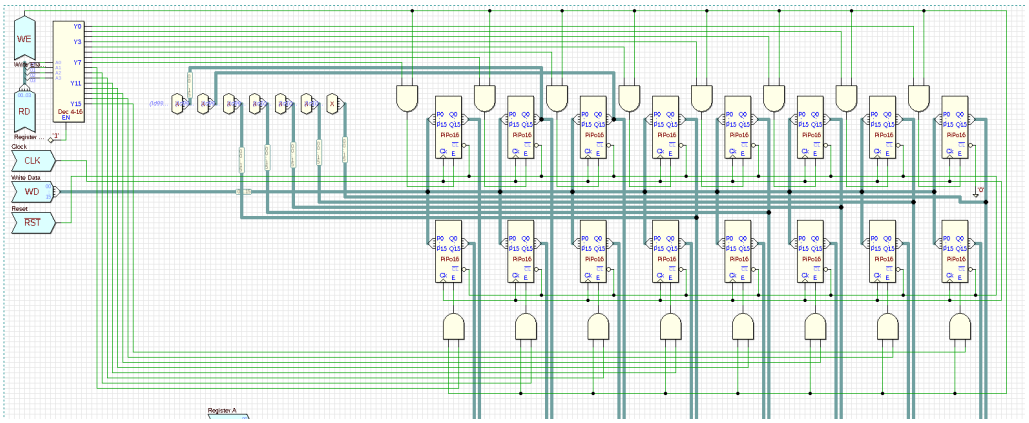


**Figura 6. Zoom nos registradores do banco**

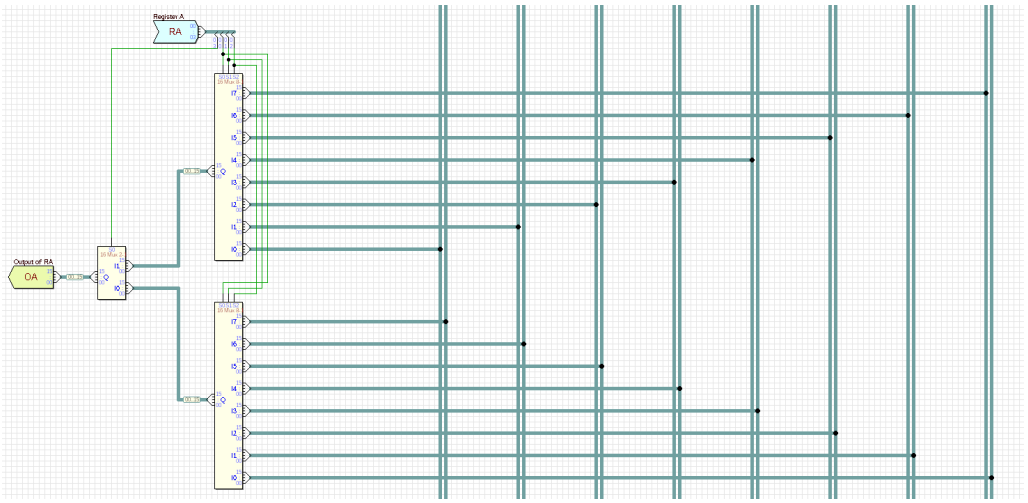


**Figura 7. Zoom no seletor de canais**

## 2.4. Montagem do Unidade Lógico Aritmética

A Unidade Lógico Aritmética (*ULA*), é um circuito indispensável a qualquer processador, pois é ela que permite por exemplo operações de soma, subtração, desvios condicionais e saltos incondicionais.

No ZeptoProcessador-III, a ULA foi implementada da seguinte forma (que pode ser visualizada na figura 8)

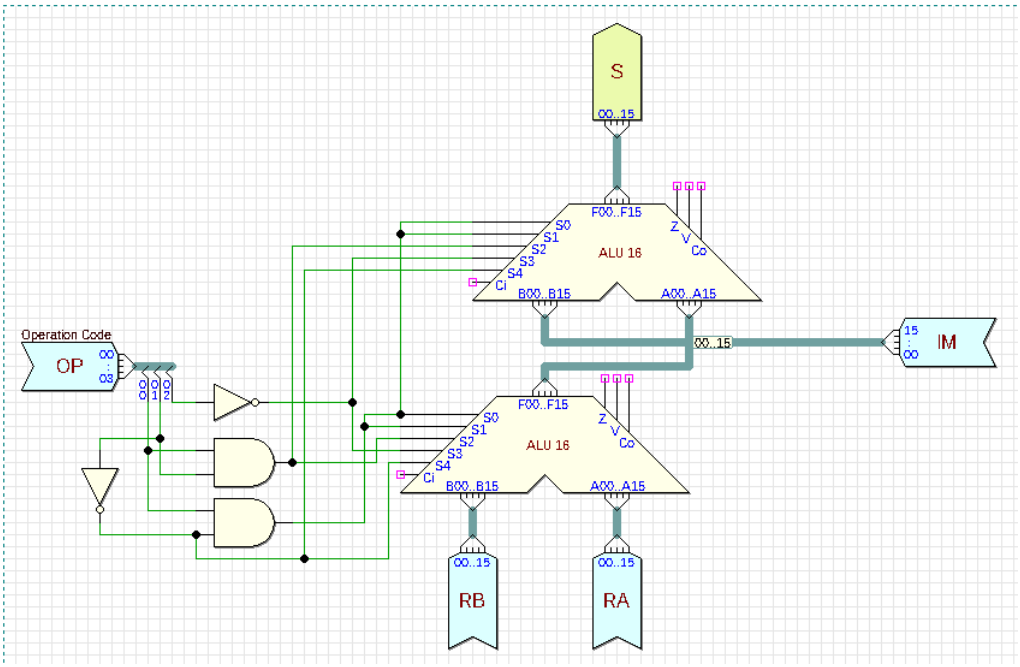


**Figura 8. Montagem do circuito comparador de números**

## 2.5. Montagem do Circuito do Comparador de Números

Uma das funcionalidades mais importantes de se ter em um processador é a capacidade de chegagem de (des)igualdade entre números. Para implementarmos tal funcionalidade em um processador faz-se necessário a criação de um bloco que permita comparar, a nível de *bits*, se um determinado número é igual, maior ou menor que outro. Como existem duas possibilidades de tratamento de números (números *com* e *sem* sinal), o circuito necessita ainda de um tratamento especial para tal condição.

Dessa forma, no ZeptoProcessador-III foi implementado um circuito com o seguinte esquema (que pode ser visualizado na figura 9):

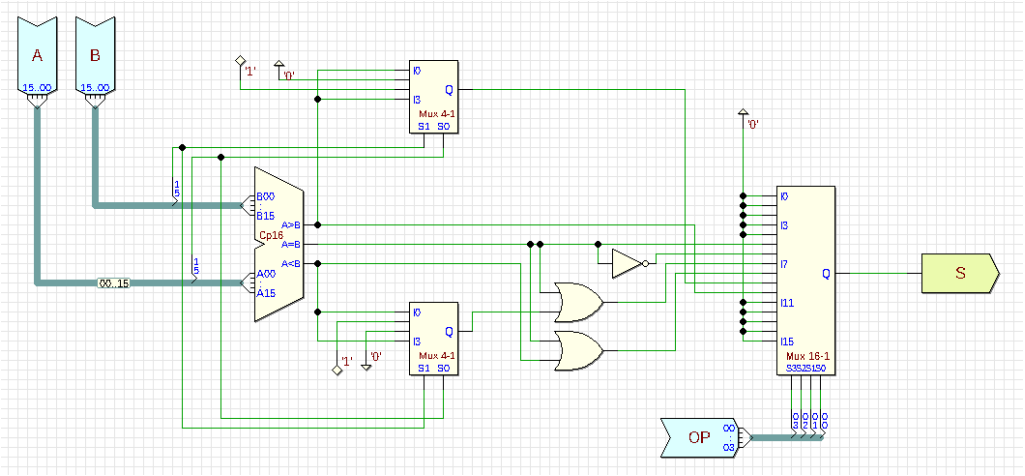


**Figura 9. Montagem do circuito comparador de números**

## 2.6. Montagem do Circuito do Bloco de Controle

Como é possível perceber na figura 2, um dos circuitos lógicos importantes do processador é o bloco de controle. Ele é responsável por interpretar as instruções lidas da memória de instruções. Dessa forma, há um mapeamento de determinada instrução para uma execução específica do processador.

Tal bloco, no ZeptoProcessador-III, foi implementado da seguinte forma (possível de ser visualizado na figura 10)

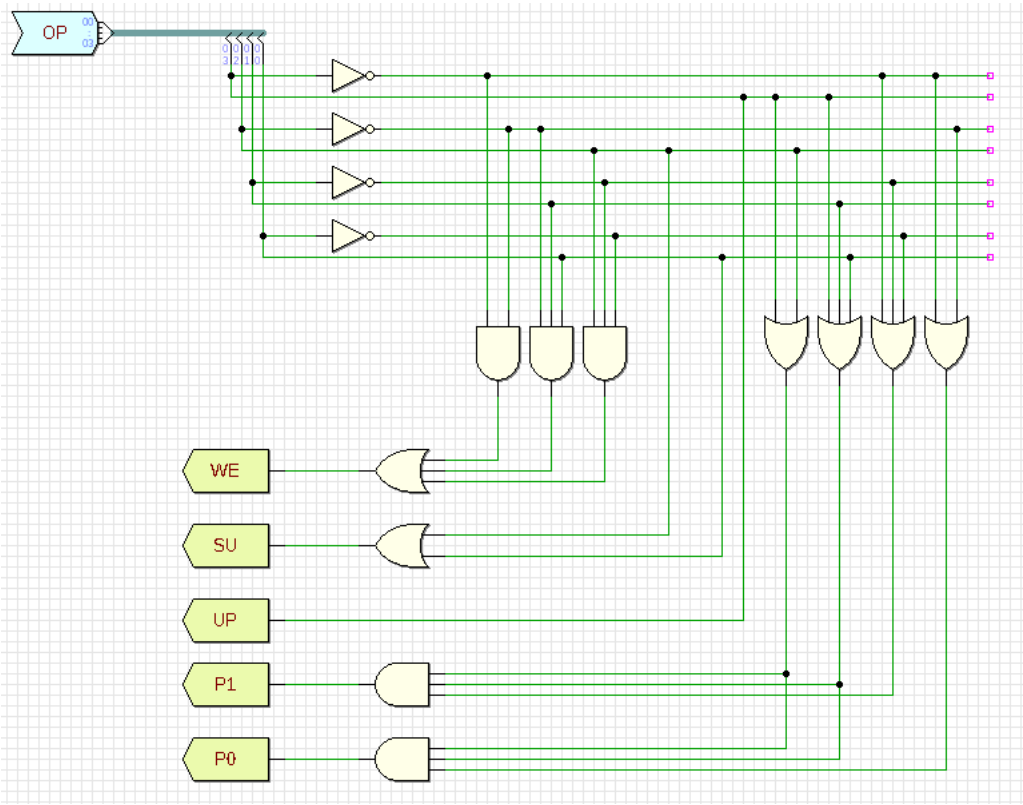


**Figura 10. Montagem do Bloco de Controle**

## 2.7. Montagem dos Sinais de Monitoramento

Por fim, uma característica importante de se adicionar ao ZeptoProcessador-III são sinais de monitoramento, para uma melhor capacidade de avaliar se o circuito se comporta de fato como esperado. Na figura 11 é possível ver os diversos sinais de monitoramento inseridos no circuito para uma melhor visualização de seu comportamento durante operação.

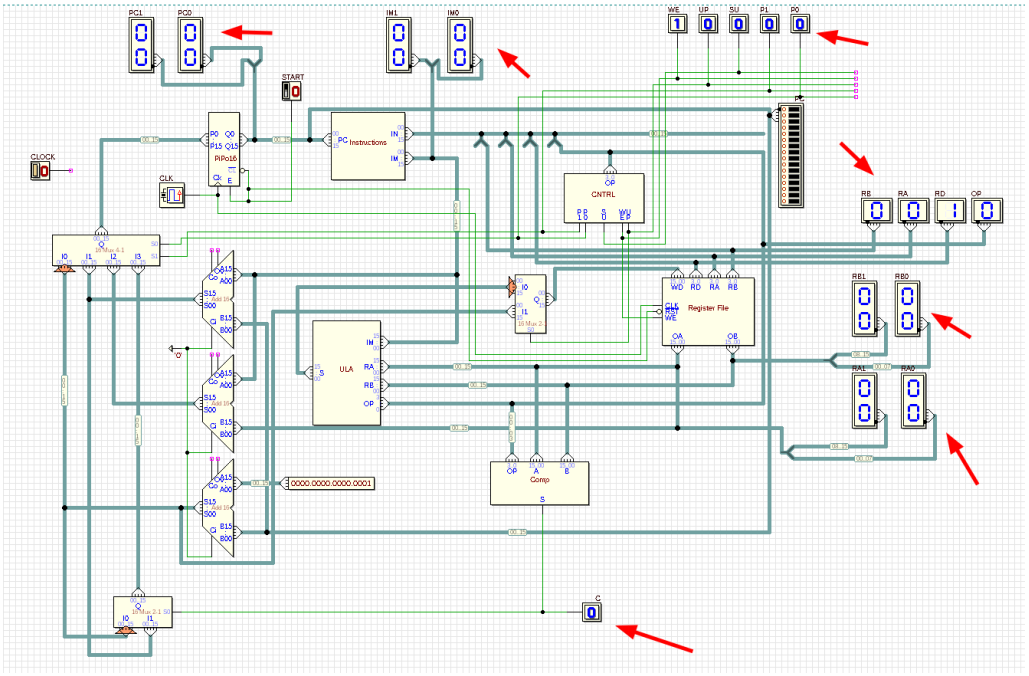


**Figura 11. Montagem dos Sinais de Monitoramento**

## 3. Execução de Programas no ZeptoProcessador-III

### 3.1. Multiplicação de Números sem Sinal

Para o primeiro programa proposto no projeto, faremos um que multiplique dois números sem sinal, e retorne o valor da multiplicação no registrador *R1*. Os valores de input para o programa serão carregados nos registradores *R2* e *R3*, com valores de $8$ e $5$, respectivamente. O resultado esperado da multiplicação será $8 \cdot 5 = 40$, que, em hexadecimal, corresponde a $0x0028$.

O código do programa, e seus respectivos comentários, podem ser encontrados na tabela 1.

**Tabela 1. Código para o programa Multu**

| Endereço | Código Hexadecimal | Instrução | Comentário |
|---|---|---|---|
| 0x0000 | 0x0000 0010 | addi R1,R0,R0,0 | R1 = 0 ;; Resultado |
| 0x0001 | 0x0008 0020 | addi R2,R0,R0,8 | R2 = 8 |
| 0x0002 | 0x0005 0030 | addi R3,R0,R0,5 | R3 = 5 |
| 0x0003 | 0x0002 00FB | jal R15,Proc | R15 = 0x0004; PC=Proc |
| 0x0004 Fim: | 0x0000 1105 | beq R1,R1,0 | j Fim ;; mostra R1 |
| 0x0005 Proc: | 0x0000 0040 | addi R4,R0,R0,0 | R4 = 0 ;; instantiate counter |
| 0x0006 Loop: | 0x0000 2110 | addi R1,R1,R2,0 | R1 += R2 |
| 0x0007 | 0x0001 0440 | addi R4,R4,R0,1 | R4 += 1 |
| 0x0008 | 0xFFFE 3406 | bne R4,R3,Loop | if (R4= R3) goto Loop |
| 0x0009 | 0x0000 0F0C | jalr R0,R15,0 | Retorna resultado em R1 |

Para a demonstração do programa funcionando, por favor conferir o vídeo no link:

https://youtu.be/NygIUTYyUNY

E, como requisitado, o diagrama de onda do programa pode ser visto na figura 12. O processador conseguiu executar o presente programa na máxima frequência de $4.35MHz$. Qualquer outra frequência mais rápida que essa resulta em falha no funcionamento, e toda frequência menor que essa resulta em um resultado sub-ótimo.

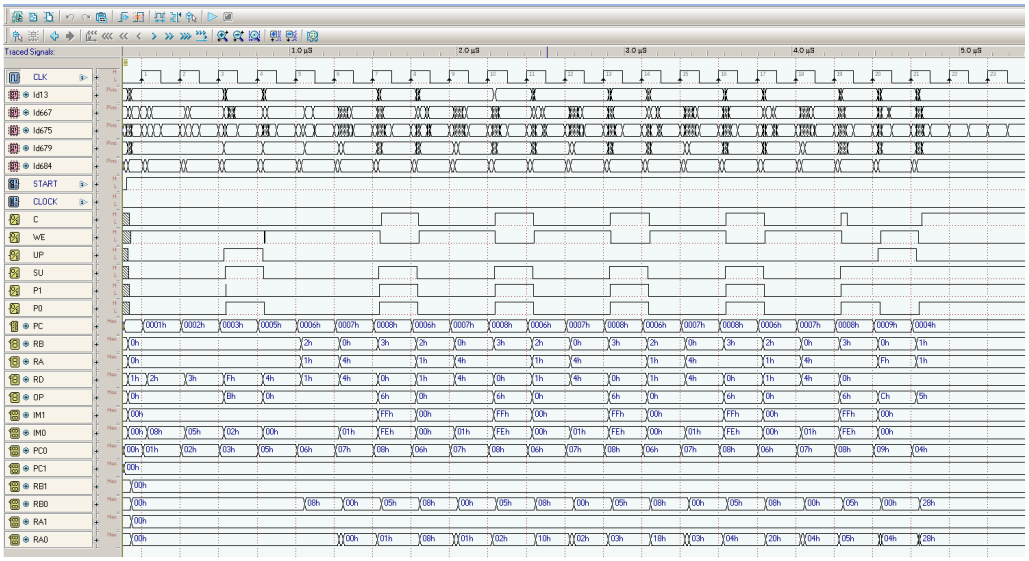


**Figura 12. Diagrama de onda para o programa Multu**

## 3.2. Multiplicação de Números com Sinal

Agora, para o programa de multiplicação de números com sinal, o programa proposto realiza a multiplicação de dois números com sinal, e retorna o valor da multiplicação no registrador *R1*. Os valores de input para o programa serão carregados nos registradores *R2* e *R3*, com valores de $-2$ e $5$, respectivamente. O resultado esperado da multiplicação será $-2 \cdot 5 = -10$, que, em hexadecimal, corresponde a $0xFFF6$.

O código do programa, e seus respectivos comentários, podem ser encontrados na tabela 2.

**Tabela 2. Código para o programa Mult**

| Endereço | Código Hexadecimal | Instrução | Comentário |
|---|---|---|---|
| 0x0000 | 0x0000 0010 | addi R1,R0,R0,0 | R1 = 0 ;; Resultado |
| 0x0001 | 0xFFFE 0020 | addi R2,R0,R0,-2 | R2 = -2 |
| 0x0002 | 0x0005 0030 | addi R3,R0,R0,5 | R3 = 5 |
| 0x0003 | 0x0002 00FB | jal R15,Proc | R15 = 0x0004; PC=Proc |
| 0x0004 Fim: | 0x0000 1105 | beq R1,R1,0 | j Fim ;; mostra R1 |
| 0x0005 Proc: | 0x0000 0040 | addi R4,R0,R0,0 | R4 = 0 ;; counter |
| 0x0006 | 0x8000 0050 | addi R5,R0,R0,0x8000 | R5 (is a mask) |
| 0x0007 | 0xFFFF 5262 | andi R6,R2,R5,0xFFFF | R6 == 0x08000 ? |
| 0x0008 | 0x0004 5606 | bne R6,R5,R2end | R6 is a result of a mask |
| 0x0009 | 0x0001 4040 | addi R4,R0,R4,1 | Increment count of odd numbers |
| 0x000A | 0xFFFF 0224 | xori R2,R2,R0,0xFFFF | One's compelemnt for R2 |
| 0x000B | 0x0001 0220 | addi R2,R2,R0,1 | Two's compelemnt for R2 |
| 0x000C R2end: | 0xFFFF 5362 | andi R6,R3,R5,0xFFFF | R6 == 0x08000 ? |
| 0x000D | 0x0004 5606 | bne R6,R5,R3end | R6 is a result of a mask |
| 0x000E | 0x0001 0440 | addi R4,R4,R0,1 | Increment count of odd numbers |
| 0x000F | 0xFFFF 0334 | xori R3,R3,R0,0xFFFF | One's compelemnt for R3 |
| 0x0010 | 0x0001 0330 | addi R3,R3,R0,1 | Two's compelemnt for R3 |
| 0x0011 R3end: | 0x0000 0070 | addi R7,R0,R0,0 | R7 = 0 ;; loop counter |
| 0x0012 Loop: | 0x0000 2110 | addi R1,R1,R2,0 | R1 += R2 |
| 0x0013 | 0x0001 0770 | addi R7,R7,R0,1 | R7 += 1 |
| 0x0014 | 0xFFFE 3706 | bne R7,R3,Loop | if (R7= R3) goto Loop |
| 0x0015 | 0x0001 0060 | addi R6,R0,R0,1 | R6 = 1 |
| 0x0016 | 0x0003 6406 | bne R4,R6,Ok | if (R4= R6) goto Ok |
| 0x0017 | 0xFFFF 0114 | xori R1,R1,R0,0xFFFF | One's complement for R1 |
| 0x0018 | 0x0001 0110 | addi R1,R1,R0,1 | Two's complement for R1 |
| 0x0019 Ok: | 0x0000 0F0C | jalr R0,R15,0 | Retorna resultado em R1 |

Para a demonstração do programa funcionando, por favor conferir o vídeo no link: https://youtu.be/IQj2BHXhxUk

E, como requisitado, o diagrama de onda do programa pode ser visto na figura 13. O processador conseguiu executar o presente programa na máxima frequência de $4.35MHz$. Qualquer outra frequência mais rápida que essa resulta em falha no funcionamento, e toda frequência menor que essa resulta em um resultado sub-ótimo.

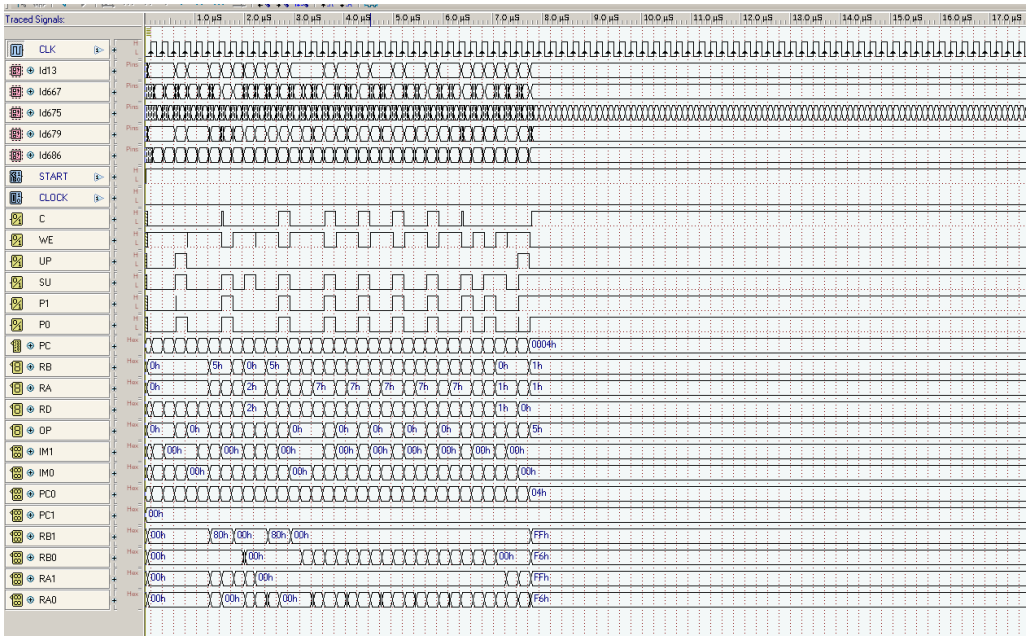


**Figura 13. Diagrama de onda para o programa Mult**

### 3.3. Divisão de Números sem Sinal

Agora, para o programa de divisão de números sem sinal, o programa proposto realiza a divisão de um número pelo outro, ambos representados sem sinal, e retorna o valor da divisão no registrador *R1*. Os valores de input para o programa serão carregados nos registradores *R2* e *R3*, com valores de $60$ e $12$, respectivamente. O resultado esperado da divisão será $\frac{60}{12} = 5$, que, em hexadecimal, corresponde a $0x0005$.

O código do programa, e seus respectivos comentários, podem ser encontrados na tabela 3.

**Tabela 3. Código para o programa Divu**

| Endereço | Código Hexadecimal | Instrução | Comentário |
|---|---|---|---|
| 0x0000 | 0x0000 0010 | addi R1,R0,R0,0 | R1 = 0 ;; Resultado |
| 0x0001 | 0x003C 0020 | addi R2,R0,R0,60 | R2 = 60 |
| 0x0002 | 0x000C 0030 | addi R3,R0,R0,12 | R3 = 12 |
| 0x0003 | 0x0002 00FB | jal R15,Proc | R15 = 0x0004; PC=Proc |
| 0x0004 Fim: | 0x0000 1105 | beq R1,R1,0 | j Fim ;; mostra R1 |
| 0x0005 Proc: | 0x0000 0040 | addi R4,R0,R0,0 | R4 = 0 ;; store the quotient |
| 0x0006 | 0x0007 230A | bgtu R3,R2,End | Make sure R2 >= R3 |
| 0x0007 Loop: | 0x0000 3221 | subi R2,R2,R3,0 | R2 -= R3 |
| 0x0008 | 0x0001 0440 | addi R4,R4,R0,1 | R4 += 1 |
| 0x0009 | 0xFFFE 320A | bgtu R2,R3,Loop | if (R2 > R3) goto Loop |
| 0x000A | 0x0003 3206 | bne R2,R3,End | if (R2= R3) goto End |
| 0x000B | 0x0001 0440 | addi R4,R4,R0,1 | R4 += 1 |
| 0x000C | 0x0000 0020 | addi R2,R0,R0,0 | R2 = 0 |
| 0x000D End: | 0x0000 0410 | addi R1,R4,R0,0 | R1 = R4 ;; get the quotient |
| 0x000E | 0x0000 0F0C | jalr R0,R15,0 | Retorna resultado em R1 |

Para a demonstração do programa funcionando, por favor conferir o vídeo no link: https://youtu.be/ZgymrJHToj8

E, como requisitado, o diagrama de onda do programa pode ser visto na figura 14. O processador conseguiu executar o presente programa na máxima frequência de $4.35MHz$. Qualquer outra frequência mais rápida que essa resulta em falha no funcionamento, e toda frequência menor que essa resulta em um resultado sub-ótimo.

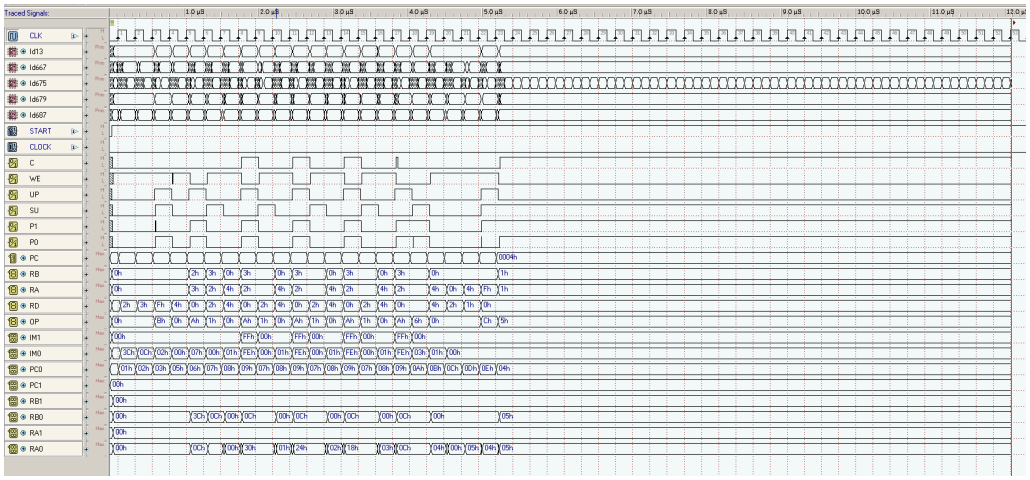

**Figura 14. Diagrama de onda para o programa Divu**

### 3.4. Divisão de Números com Sinal

Agora, para o programa de divisão de números com sinal, o programa proposto realiza a divisão de um número pelo outro, ambos representados com sinal, e retorna o valor da divisão no registrador *R1*. Os valores de input para o programa serão carregados nos registradores *R2* e *R3*, com valores de $1337$ e $-42$, respectivamente. O resultado esperado da divisão será $\frac{1337}{-42} = -31$, que, em hexadecimal, corresponde a $0xFFE1$. É importante fazer a observação de que optamos por seguir o padão ANSI99 para a divisão de números negativos. (A mesma divisão, utilizando o algotimo implementado na linguagem *Python*, por exemplo, retornaria $-32$).

O código do programa, e seus respectivos comentários, podem ser encontrados na tabela 4.

**Tabela 4. Código para o programa Div**

| Endereço | Código Hexadecimal | Instrução | Comentário |
|---|---|---|---|
| 0x0000 | 0x0000 0010 | addi R1,R0,R0,0 | R1 = 0 ;; Resultado |
| 0x0001 | 0x0539 0020 | addi R2,R0,R0,64 | R2 = 1337 |
| 0x0002 | 0xFFD6 0030 | addi R3,R0,R0,12 | R3 = -42 |
| 0x0003 | 0x0002 00FB | jal R15,Proc | R15 = 0x0004; PC=Proc |
| 0x0004 Fim: | 0x0000 1105 | beq R1,R1,0 | j Fim ;; mostra R1 |
| 0x0005 Proc: | 0x0000 0040 | addi R4,R0,R0,0 | R4 = 0 ;; store the quotient |
| 0x0006 | 0x0000 0050 | addi R5,R0,R0,0 | R5 = 0 ;; division flag |
| 0x0007 | 0x0003 0209 | bgt R2,R0,Apos | if (R2 ¿ R0) goto Apos |
| 0x0008 | 0x0000 2021 | subi R2,R0,R2,0 | R2 = -R2 |
| 0x0009 | 0x0001 0550 | addi R5,R5,R0,1 | R5 += 1 |
| 0x000A Apos: | 0x0003 0309 | bgt R3,R0,Div | if (R3 ¿ R0) goto Bpos |
| 0x000B | 0x0000 3031 | subi R3,R0,R3,0 | R3 = -R3 |
| 0x000C | 0x0001 0550 | addi R5,R5,R0,1 | R5 += 1 |
| 0x000D Div: | 0x0000 3221 | subi R2,R2,R3,0 | R2 -= R3 |
| 0x000E | 0x0003 2009 | bgt R0,R2,Enddiv | if (R2 ¿ R0) goto Enddiv |
| 0x000F | 0x0001 0440 | addi R4,R4,R0,1 | R4 += 1 |
| 0x0010 | 0xFFFD 000B | jal R0,Div | goto Div |
| 0x0011 Enddiv: | 0x0000 3220 | addi R2,R2,R3,0 | R2 += R3 |
| 0x0012 | 0x0001 0660 | addi R6,R6,R0,1 | R6 = 1 |
| 0x0013 | 0x0002 6506 | bne R5,R6,End | if (R5= R6) goto End |
| 0x0014 | 0x0000 4041 | subi R4,R0,R4,0 | R4 = -R4 |
| 0x0015 End: | 0x0000 0410 | addi R1,R4,R0,0 | R1 = R4 ; save quotient |
| 0x0016 | 0x0000 0F0C | jalr R0,R15,0 | Retorna resultado em R1 |

Para a demonstração do programa funcionando, por favor conferir o vídeo no link: https://youtu.be/BimxtoXOGsk

E, como requisitado, o diagrama de onda do programa pode ser visto na figura 15. O processador conseguiu executar o presente programa na máxima frequência de $4.35MHz$. Qualquer outra frequência mais rápida que essa resulta em falha no funcionamento, e toda frequência menor que essa resulta em um resultado sub-ótimo.

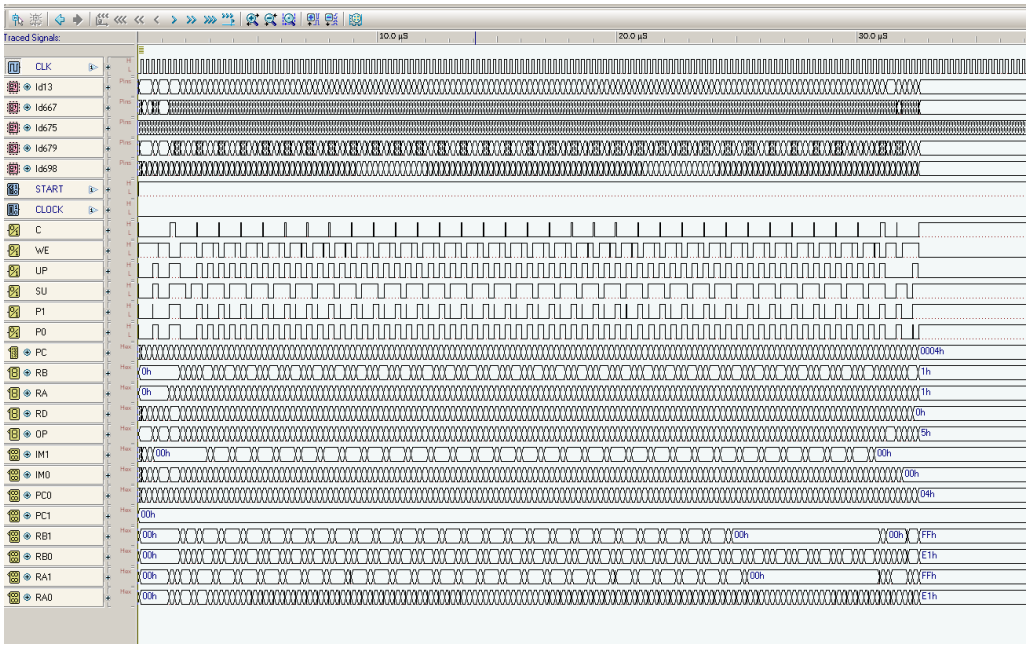


**Figura 15. Diagrama de onda para o programa Div**

## 3.5. Resto da Divisão de Números sem Sinal

Agora, para o programa que computa do resto da divisão de números sem sinal, o programa proposto realiza a divisão de um número pelo outro, ambos representados sem sinal, e retorna o valor do resto da divisão no registrador *R1*. Os valores de input para o programa serão carregados nos registradores *R2* e *R3*, com valores de $64$ e $12$, respectivamente. O resultado esperado do resto da divisão será $4$, pois $64 = 12 * 5 + 4$. Como o resultado é dado em hexadecimal, esse valor será de $0x0004$.

O código do programa, e seus respectivos comentários, podem ser encontrados na tabela 5.

**Tabela 5. Código para o programa Remu**

| Endereço | Código Hexadecimal | Instrução | Comentário |
|---|---|---|---|
| 0x0000 | 0x0000 0010 | addi R1,R0,R0,0 | R1 = 0 ;; Resultado |
| 0x0001 | 0x0040 0020 | addi R2,R0,R0,64 | R2 = 64 |
| 0x0002 | 0x000C 0030 | addi R3,R0,R0,12 | R3 = 12 |
| 0x0003 | 0x0002 00FB | jal R15,Proc | R15 = 0x0004; PC=Proc |
| 0x0004 Fim: | 0x0000 1105 | beq R1,R1,0 | j Fim ;; mostra R1 |
| 0x0005 Proc: | 0x0000 0040 | addi R4,R0,R0,0 | R4 = 0 ;; store the quotient |
| 0x0006 | 0x0007 230A | bgtu R3,R2,End | Make sure R2 >= R3 |
| 0x0007 Loop: | 0x0000 3221 | subi R2,R2,R3,0 | R2 -= R3 |
| 0x0008 | 0x0001 0440 | addi R4,R4,R0,1 | R4 += 1 |
| 0x0009 | 0xFFFE 320A | bgtu R2,R3,Loop | if (R2 ¿ R3) goto Loop |
| 0x000A | 0x0003 3206 | bne R2,R3,End | if (R2= R3) goto End |
| 0x000B | 0x0001 0440 | addi R4,R4,R0,1 | R4 += 1 |
| 0x000C | 0x0000 0020 | addi R2,R0,R0,0 | R2 = 0 |
| 0x000D End: | 0x0000 0210 | addi R1,R2,R0,0 | R1 = R2 ;; get the remainder |
| 0x000E | 0x0000 0F0C | jalr R0,R15,0 | Retorna resultado em R1 |

Para a demonstração do programa funcionando, por favor conferir o vídeo no link: https://youtu.be/OMpzPPJSW2Q

E, como requisitado, o diagrama de onda do programa pode ser visto na figura 16. O processador conseguiu executar o presente programa na máxima frequência de $4.35MHz$. Qualquer outra frequência mais rápida que essa resulta em falha no funcionamento, e toda frequência menor que essa resulta em um resultado sub-ótimo.

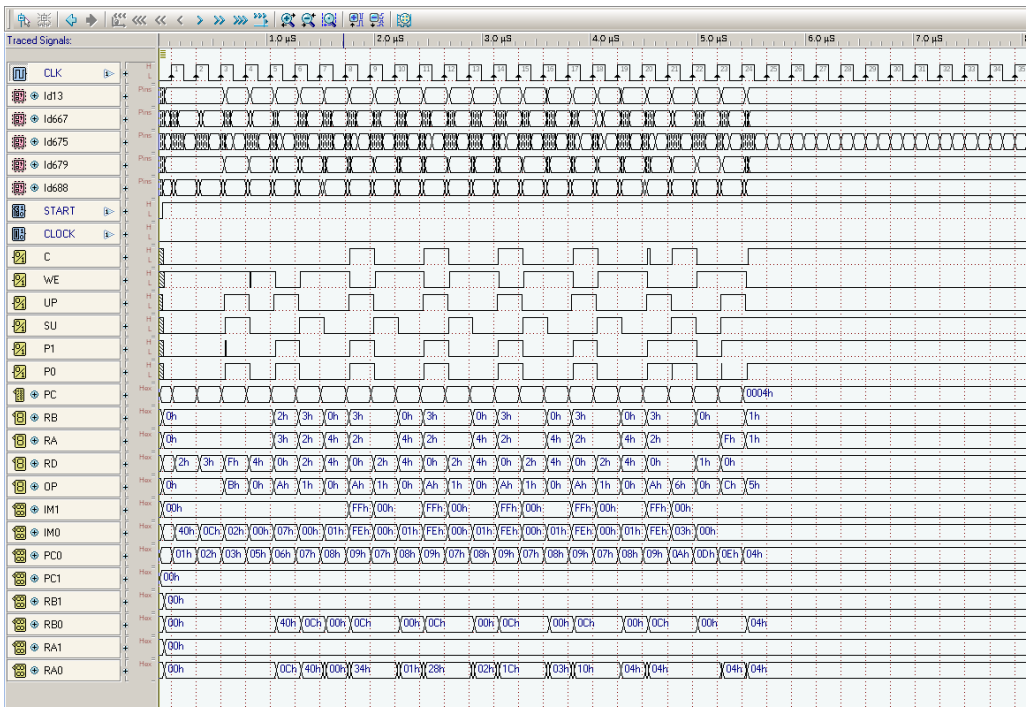


**Figura 16. Diagrama de onda para o programa Remu**

### 3.6. Resto da Divisão de Números com Sinal

Agora, para o programa que computa do resto da divisão de números com sinal, o programa proposto realiza a divisão de um número pelo outro, ambos representados com sinal, e retorna o valor do resto da divisão no registrador *R1*. Os valores de input para o programa serão carregados nos registradores *R2* e *R3*, com valores de $1337$ e $-42$, respectivamente. O resultado esperado do resto da divisão será $35$, pois estamos usando o padrão ANSI99 para realizarmos as divisões. Como o resultado é dado em hexadecimal, esse valor será de $0x0023$.

O código do programa, e seus respectivos comentários, podem ser encontrados na tabela 6.

**Tabela 6. Código para o programa Rem**

| Endereço | Código Hexadecimal | Instrução | Comentário |
|---|---|---|---|
| 0x0000 | 0x0000 0010 | addi R1,R0,R0,0 | R1 = 0 ;; Resultado |
| 0x0001 | 0x0539 0020 | addi R2,R0,R0,64 | R2 = 1337 |
| 0x0002 | 0xFFD6 0030 | addi R3,R0,R0,12 | R3 = -42 |
| 0x0003 | 0x0002 00FB | jal R15,Proc | R15 = 0x0004; PC=Proc |
| 0x0004 Fim: | 0x0000 1105 | beq R1,R1,0 | j Fim ;; mostra R1 |
| 0x0005 Proc: | 0x0000 0040 | addi R4,R0,R0,0 | R4 = 0 ;; store the quotient |
| 0x0006 | 0x0000 0050 | addi R5,R0,R0,0 | R5 = 0 ;; division flag |
| 0x0007 | 0x0003 0209 | bgt R2,R0,Apos | if (R2 ¿ R0) goto Apos |
| 0x0008 | 0x0000 2021 | subi R2,R0,R2,0 | R2 = -R2 |
| 0x0009 | 0x0001 0550 | addi R5,R5,R0,1 | R5 += 1 |
| 0x000A Apos: | 0x0003 0309 | bgt R3,R0,Div | if (R3 ¿ R0) goto Bpos |
| 0x000B | 0x0000 3031 | subi R3,R0,R3,0 | R3 = -R3 |
| 0x000C | 0x0001 0550 | addi R5,R5,R0,1 | R5 += 1 |
| 0x000D Div: | 0x0000 3221 | subi R2,R2,R3,0 | R2 -= R3 |
| 0x000E | 0x0003 2009 | bgt R0,R2,Enddiv | if (R2 ¿ R0) goto Enddiv |
| 0x000F | 0x0001 0440 | addi R4,R4,R0,1 | R4 += 1 |
| 0x0010 | 0xFFFD 000B | jal R0,Div | goto Div |
| 0x0011 Enddiv: | 0x0000 3220 | addi R2,R2,R3,0 | R2 += R3 |
| 0x0012 | 0x0001 0660 | addi R6,R6,R0,1 | R6 = 1 |
| 0x0013 | 0x0002 6506 | bne R5,R6,End | if (R5= R6) goto End |
| 0x0014 | 0x0000 4041 | subi R4,R0,R4,0 | R4 = -R4 |
| 0x0015 End: | 0x0000 0210 | addi R1,R2,R0,0 | R1 = R2 ; save remainder |
| 0x0016 | 0x0000 0F0C | jalr R0,R15,0 | Retorna resultado em R1 |

Para a demonstração do programa funcionando, por favor conferir o vídeo no link: https://youtu.be/-2xW0z4dV00

E, como requisitado, o diagrama de onda do programa pode ser visto na figura 17. O processador conseguiu executar o presente programa na máxima frequência de $4.35MHz$. Qualquer outra frequência mais rápida que essa resulta em falha no funcionamento, e toda frequência menor que essa resulta em um resultado sub-ótimo.

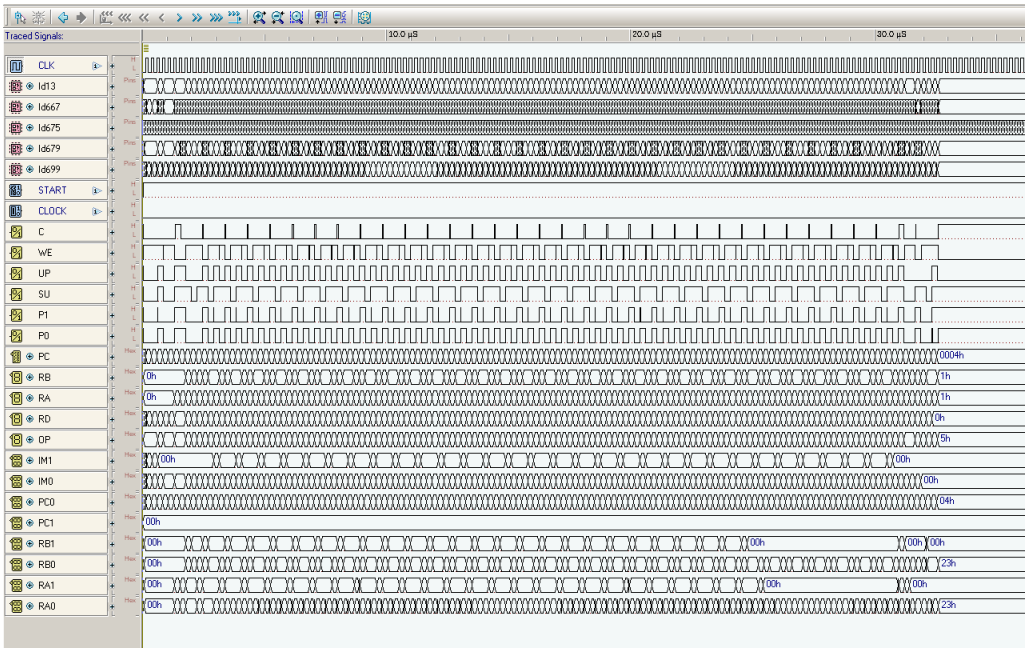


**Figura 17. Diagrama de onda para o programa Rem**

## 3.7. Encontrando o primeiro primo em um intervalo

Para a descoberta de números primos em um intervalo $0 < a < b$, podemos dividir a análise e partes mais específicas, uma delas seria primeiro buscar uma forma de verificar se o número analisado em questão é um número primo, para isto basta então dividir este número por todos os números naturais anteriores que são maiores do que 1 e menores do que o número analisado. Feito esta analise, caso o número não seja primo, basta analisar o próximo.

**Tabela 7. Código para o programa Primo**

| Endereço | Código Hexadecimal | Instrução | Comentário |
|---|---|---|---|
| 0x0000 | 0x0000 0010 | addi r1 r0 r0 0 | R1 = 0 ;; Resultado- |
| 0x0001 | 0x0036 0020 | addi r2 r0 r0 54 | R2 = 54 |
| 0x0002 | 0x0064 0030 | addi r3 r0 r0 100 | R3 = 100 |
| 0x0003 | 0x0005 00FB | jal r15 5 | j para 0x0008 |
| 0x0004 | 0x0000 2010 | addi r1 r0 r2 0 | R1 = R2 |
| 0x0005 | 0x0000 1105 | beq r1 r1 0 | j Fim ;; mostra R1 |
| 0x0006 | 0x0001 0220 | addi r2 r2 r0 1 | R2 += 1 |
| 0x0007 | 0xfffe 3209 | bgt r2 r3 -2 | if (R2 ¿ R3) goto 0x0005 |
| 0x0008 | 0x0000 2040 | addi r4 r0 r2 0 | R4 = R2 |
| 0x0009 | 0x0002 0050 | addi r5 r0 r0 2 | R5 = 2 |
| 0x000A | 0x0005 000B | jal r0 5 | pula para 0x000f |
| 0x000b | 0xfffb 0405 | beq r4 r0 -5 | if (R4 == 0) goto 0x0006 |
| 0x000c | 0x0001 0550 | addi r5 r5 r0 1 | R5 += 1 |
| 0x000d | 0x0000 2040 | addi r4 r0 r2 0 | R4 = R2 |
| 0x000e | 0xfff6 4505 | beq r5 r4 -10 | if (R5 == R4) goto 0x0004 |
| 0x000f | 0x0000 5441 | subi r4 r4 r5 0 | R4 -= R5 |
| 0x0010 | 0xfffb 4009 | bgt r0 r4 -5 | if (0 ¿ R4) goto 0x000b |
| 0x0011 | 0xfffa 0405 | beq r4 r0 -6 | if (r4 == 0) foto 0x000b |
| 0x0012 | 0xfffd 000B | jal r0 -3 | pula para 0x000f |
| 0x0013 | 0x0000 0F0C | jalr r0 r15 0 | Retorna resultado em R1 |

Para a demonstração do programa funcionando, por favor conferir o vídeo no link: https://youtu.be/nBpRLZitRZc

E, como requisitado, o diagrama de onda do programa pode ser visto na figura 18. O processador conseguiu executar o presente programa na máxima frequência de $4.35MHz$. Qualquer outra frequência mais rápida que essa resulta em falha no funcionamento, e toda frequência menor que essa resulta em um resultado sub-ótimo.

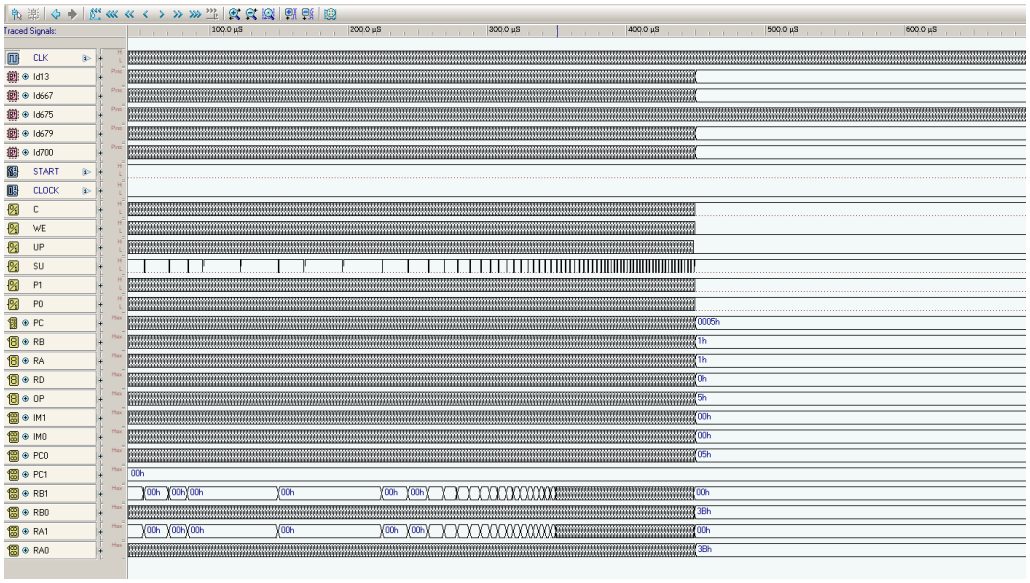


**Figura 18. Diagrama de onda para o programa Primo**

### 3.8. Maior frequência do ZeptoProcessador-III

Como requisitado no enunciado do projeto, abordaremos agora a questão da frequência máxima de operação do ZeptoProcessador-III. O resultado encontrado para a maior frequência operação correra do ZeptoProcessador-III foi de $4.35MHz$ (um comprimento de onda de $230nS$).

O limitante para uma maior frequência de operação é o tempo para a obtenção dos resultados da operação atual sendo processada, pois, para completar um ciclo de instrução, pode ser levado um tempo "X". Entretanto, caso a frequência ultrapasse as limitações do processador teremos que antes mesmo da finalização de uma instrução outra já estará sendo executada. Isso acarretará em um estado inconsistente do circuito lógico do processador e, por consequência, em um funcionamento incorreto do mesmo.

## 4. Conclusão

Com esse projeto pudemos ver todo o processo para desenvolvimento de um processador, mesmo que um simples. O circuito lógico necessário para tal construção provou-se complexo, como mencionado no início desse relatório. Por meio da utilização de portas lógicas pudtemos desenvolver abstrações mais complexas, como multiplexadores, somadores, *Flip-flops*, e, subindo no nível de abstrações, finalmente pudemos implementar registradores de instrução, uma memória de instrução suficientemente grande para comportar $32$ *bits* de instrução, um banco de registradores de $16$ *bits*, e toda uma lógica para interpretação de códigos de operação para funcionamento do processador condicionado ao código inserido pelo usuário.

Como pudemos constatar na seção 3.8, o ZeptoProcessador-III é limitado a uma frequência máxima de operação de $4.35MHz$, demonstrando com o atraso de portas lógicas acabam se somando e impactando na velocidade do circuito final.

## Referências

[Harris and Harris 2012] Harris, D. and Harris, S. (2012). Digital design and computer architecture. https://www.amazon.com/Digital-Design-Computer-Architecture-Harris/dp/0123944244.

[Lamar 2021] Lamar, M. V. (2021). Projeto 2021 - zeptoprocessador-iii.

[Wikipedia 2021a] Wikipedia (2021a). Instrunction register. https://en.wikipedia.org/wiki/Instruction_register.

[Wikipedia 2021b] Wikipedia (2021b). Rom memory. https://en.wikipedia.org/wiki/Read-only_memory.