Projeto
ZeptoProcessador-III de 16 bits

Gabriel Mendes Ciriatico Guimarães, 202033202
Grupo G46

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

202033202@aluno.unb.br

Abstract. *This project describes the development of a 16-bit processor, the ZeptoProcessor-III, which implements basic arithmetic and logic operations. In the project, the various parts of the processor are developed, such as the PC register, register bank, ROM memories, control block and comparator. Programs that use processor operations to implement new operations are also designed, such as signed and unsigned multiplier and divisor and unsigned remainder of the division between 2 numbers.*

Resumo. *Neste projeto é descrito o desenvolvimento de um processador de 16 bits, o ZeptoProcessador-III, que implementa operações artiméticas e lógicas básicas. No projeto são desenvolvidas as várias partes do processador, como registrador PC, banco de registradores, memórias ROM, bloco de controle e comparador. Programas que usam as operações do processador para implementar novas operações também são elaborados, como multiplicador e divisor com e sem sinal e resto sem sinal da divisão entre 2 números.*

1. Introdução

Um processador é um circuito capaz de executar instruções, permitindo algumas operações básicas com os dados de entrada. Isso permite que programas mais complexos sejam desenvolvidos a partir dessas operações básicas. Um processador com soma e subtração, por exemplo, pode ser usado para desenvolver programas com multiplicação e divisão. [Wikipedia 2021b]

O processador executa operações aritméticas básicas e lógicas dependendo de uma entrada e entregando uma saída. Todas as operações de um computador são executadas em um processador, por exemplo.

Neste projeto, o objetivo é construir um processador simples usando os conhecimentos aprendidos ao longo da disciplina. Conceitos como operações lógicas e binárias, soma e subtração com sinal e sem sinal, MUX sendo usados para determinar condição de saída, registradores e clocks são fundamentais nessa construção. [Koike 2021a]

O processador aqui desenvolvido, o ZeptoProcessador-III, tem 16 bits e recebe instruções de 32 bits. Essas instruções são divididas em 5 partes: Opcode, Rd, Ra, Rb e Imediato (Imm). A divisão dessas partes em bits pode ser vista na figura abaixo, retirada das especificações do projeto. [Lamar and Mandelli 2021]

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Rb				**Ra**				**Rd**				**Opcode**			

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Imediato															

Figura 1. Divisão da instrução do processador por bit.

Aqui, são implementadas 13 operações especificadas através do Opcode. Essas operações vão desde operações aritméticas básicas como soma e subtração até a saltos de memória. O Opcode e a descrição de cada uma dessas operações podem ser vistos logo abaixo.

OpCode	Mnemônico	Nome	Operação
0000	addi Rd, Ra, Rb, Imm	Soma com imediato	Rd = Ra + Rb + Imm
0001	subi Rd, Ra, Rb, Imm	Subtração com imediato	Rd = Ra - Rb – Imm
0010	andi Rd, Ra, Rb, Imm	And bitwise com imediato	Rd = Ra & Rb & Imm
0011	ori Rd, Ra, Rb, Imm	Or bitwise com imediato	Rd = Ra | Rb | Imm
0100	xori Rd, Ra, Rb, Imm	Xor bitwise com imediato	Rd = Ra ⊕ Rb ⊕ Imm
0101	beq Ra, Rb, Imm	Salto se igual	Ra==Rb ? PC=PC+Imm : PC=PC+1
0110	bne Ra, Rb, Imm	Salto se diferente	Ra!=Rb ? PC=PC+Imm : PC=PC+1
0111	ble Ra,Rb,Imm	Salto se menor ou igual	Ra<=Rb ? PC=PC+Imm : PC=PC+1 Ra e Rb considerados com sinal
1000	bleu Ra, Rb, Imm	Salto se menor ou igual unsigned	Ra<=Rb ? PC=PC+Imm : PC=PC+1 Ra e Rb considerados sem sinal
1001	bgt Ra, Rb, Imm	Salto se maior	Ra>Rb ? PC=PC+Imm : PC=PC+1 Ra e Rb considerados com sinal
1010	bgtu Ra, Rb, Imm	Salto se maior unsigned	Ra>Rb ? PC=PC+Imm : PC=PC+1 Ra e Rb considerados sem sinal
1011	jal Rd,Imm	Salto incondicional ao endereço	Rd=PC+1 e PC=PC+Imm
1100	jalr Rd,Ra,Imm	Salto incondicional ao registrador	Rd=PC+1 e PC=Ra+Imm

Figura 2. Lista de operações definidas pelo Opcode.

Observe que Rd define o registrador que terá dados escritos, enquanto Ra e Rb definem os registradores que serão usados como componentes de alguma operação. Já o valor Imm é um valor imediato que pode ser usado sem precisar ser armazenado, sendo muito mais facilmente acessado.

O processador aqui desenvolvido é composto de 6 partes principais: o registrador PC; a memória de instruções; o banco de registradores; a Unidade Lógico-Aritmética (ULA); o comparador; e o bloco de controle. [Mandelli 2021]

Um conceito fundamental aqui é o de registrador. Os registradores são circuitos sequenciais capazes de guardar valores. Aqui, eles permitem guardar as diferentes variáveis e seus valores no banco de registradores. Essas variáveis estão armazenadas no banco de registradores. A seleção da instrução de memória a ser executada também é feita através de um registrador, o PC.

A memória usada no projeto também tem papel extremamente importante, já que guarda as instruções que o programa deve executar. De modo geral, a cada instrução executada passa-se para a próxima, com o registrador PC, que faz a seleção da memória, passando para PC + 1. Isso é feito em todos os casos com exceção daqueles que implementam saltos na memória, que podem ser vistos nas operações já descritas acima.

A ULA é um circuito que implementa operações aritméticas básicas, como soma e subtração. No caso deste projeto, será usada uma ULA já desenvolvida e oferecida pelo Deeds. [Wikipedia 2021a]

O comparador, por sua vez, recebe dois números A e B e checa se A $>$ B, A == B ou A $<$ B. Neste projeto, foi usado o comparador do Deeds, que precisou sofrer alterações para aceitar comparações com sinal também, além das sem sinal já implementadas. Aqui, as saídas do comparador indicam apenas se A $<=$ B e A == B, embora as 3 comparações básicas sejam executadas para se obter essas saídas.

O bloco de controle, por sua vez, recebe como entrada o Opcode e define todas as saídas que dependem do Opcode. Entre essas saídas, há a saída que define se a comparador deve ser usado com sinal ou sem sinal, se há escrita de dados em Rd, qual é a entrada de seleção da ULA (que define qual operação realizar) e quais são as entradas de seleção dos MUX que dependem do bloco.

Os multiplexadores são os grandes responsáveis por ligar todos esses blocos, definindo os dados D a serem escritos em Rd e a próxima memória a ser acessada pelo registrador PC.

Isso acontece porque multiplexadores podem ser usados em circuito para aplicar condições. As entradas de seleção definem a condição que determina qual entrada do MUX será transmitida à saída. [Koike 2021b]

São os MUX no processador aqui desenvolvido que definem qual a operação de comparação necessária, por exemplo. Se o usuário quer saber se Ra $>$ Rb, é preciso que o circuito pegue a saída A $<=$ B e conecte uma porta NOT a esse fio.

Já se o usuário quer saltar a memória e fazer PC + 5, e não PC + 1, o MUX precisa identificar pelo Opcode se essa operação é possível e qual deve ser a saída conectada a PC.

1.1. Objetivos

O primeiro objetivo e o mais fundamental deste projeto é desenvolver o processador, colocando em prática todos os conhecimentos vistos pelo estudante ao longo da disciplina. Questões como operações binárias, MUX, registrador, flip-flops e clocks são fundamentais para esse processo.

O projeto também tem por objetivo colocar o estudante para desenvolver programas através do processador desenvolvido. Isso permite aproximar a capacidade de programação do estudante a uma noção de programação mais baixo nível.

Para programar com esse processador, por exemplo, o estudante se vê limitado a operações através de bits, precisando pensar em memória.

Com isso, há o objetivo de se desenvolver os programas de: multiplicação sem e com sinal; divisão sem e com sinal; resto sem e com sinal; e encontrar o primeiro número

primo em um intervalo dado.

Também é um objetivo do projeto entender e especificar o limite de frequência no qual o clock do processador consegue operar. A partir desse limite, as operações passam a entregar dados não confiáveis. O estudante deve saber explicar por que isso ocorre e indicar a frequência máxima permitida.

2. Metodologia

O processador foi montado através de 7 componentes principais: o registrador PC, que armazena o endereço da instrução a ser executada; a memória de instruções que tem endereço de 16 bits e entrega instrução de 32 bits; o banco de registradores, com 16 registradores de 16 bits para armazenar os registradores com os quais lidamos (Ra, Rb e Rd).

Também há a Unidade Lógico-Aritmética (ULA), implementada pelo próprio Deeds e aqui usada para fazer operações aritméticas; o comparador, que checa se 2 números são iguais, diferentes, o primeiro é menor, igual ou maior ao segundo (com sinal ou sem sinal); o bloco de controle, que define as entradas dos outros componentes segundo o Opcode inserido; e os sinais de monitoramento, displays de 7 segmentos hexadecimais que exibem os bits de PC, da instrução, de Ra e de Rb.

A ligação entre todos esses componentes é feita principalmente através de MUX, que definem os dados D a serem escritos no banco de registradores e o próximo endereço do registrador PC.

2.1. Registrador PC

O registrador PC armazena o endereço da instrução a ser executada, tendo sua saída conectada diretamente às entradas das memórias ROM utilizadas. Aqui, esse registrador tem 16 bits, podendo armazenar até 65536 endereços. Como é um registrador com RESET síncrono, sua implementação exigiu também o uso de um MUX $2:1$ de 16 bits.

A implementação total foi encapsulada em um bloco no Deeds. Nela, é possível observar em detalhes como foi implementado o RESET. O MUX tem conectado à entrada de seleção o RESET, enquanto na sua entrada de dados 0 tem os dados D e na entrada 1 16 bits 0. Quando RESET = 1, a saída é igual à entrada 1, sendo apenas os 0. Quando RESET = 0, a entrada de dados escolhida é a entrada do fluxo normal de dados.

A saída desse MUX é conectada à entrada de um registrador de 16 bits, que tem o clock que garante o sincronismo das ações. Nesse registrador, a entrada ENABLE fica sempre em 1, já que no registrador PC sempre estamos fazendo operações de escrita. CLR' também é uma entrada disponível, embora não seja usada, já que estamos fazendo uso de RESET síncrono.

A saída S desse registrador é conectada às memórias ROM.

Veja na figura abaixo como foi feita a implementação do registrador PC.

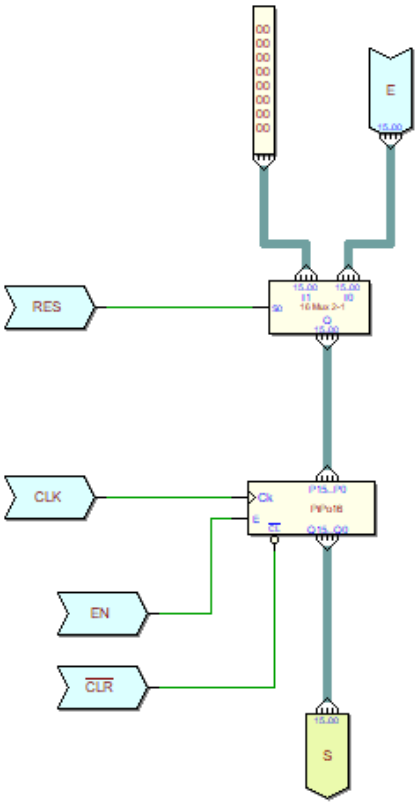

Figura 3. Implementação do registrador PC.

2.2. Memória de instruções

As instruções do processador são compostas de 32 bits, que devem ser preenchidos e armazenados em arquivo .drs. Observe, no entanto, que o Deeds não tem memória com saída de 32 bits, apenas de 16 bits. Aqui, portanto, foi preciso utilizar 2 memórias de 16 bits, sendo 1 responsável pelos 16 bits mais significativos e outra pelos 16 menos significativos. Os arquivos. drs, portanto, precisam ser divididos, um msb e outro lsb.

No projeto foram utilizadas 2 memórias ROM1Kx16. A memória é ROM porque nela não escrevemos nada, apenas lemos os dados. É 1K porque consegue armazenar 2^10 = 1024 endereços. Para os programas que vamos executar, esse é um tamanho suficiente já.

Observe que a saída do registrador PC tem 16 bits mas a entrada das memórias tem apenas 10 bits. Por isso foi necessário utilizar bus tap para separar os 10 bits menos significativos do registrador PC e conectá-los às memórias. Veja, portanto, que o máximo de endereços que podemos usar não são 2^16, e sim 2^10.

As saídas dessas memórias precisam ser depois juntadas em fiação bus para compor os 32 bits de instrução que são usados pelo processador.

Veja na figura abaixo como foi feita a implementação das memórias ROM usadas.

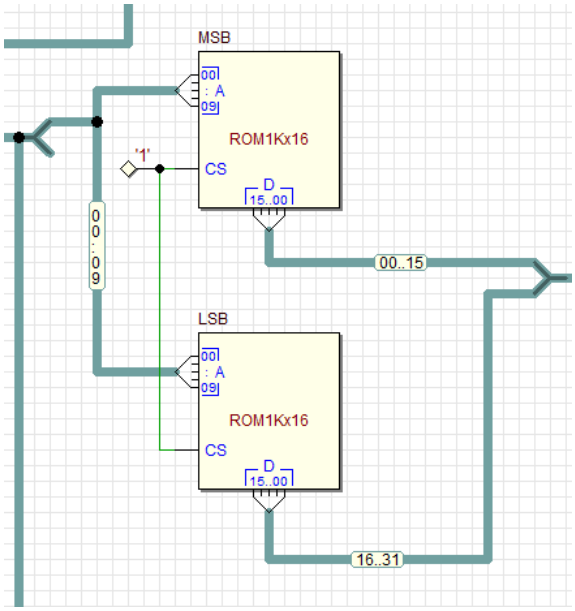

Figura 4. Implementação das memórias ROM de instruções.

2.3. Banco de registradores

O banco de registradores é composto por 16 registradores de 16 bits, todos com RESET síncrono. Esse banco é o que permite associar a um endereço um registrador e colocar nesse endereço um dado - o registrador 0x0000 recebendo o valor hexadecimal 0x0001, por exemplo.

Esse banco recebe várias entradas. A entrada RESET define se esse banco deve ser resetado, com todos os registrados indo para o valor 0 na próxima borda de subida do clock. A entrada WE informa se estão sendo realizadas operações de escrita ou apenas leitura nos registradores - quando WE = 1, a escrita é permitida, e WE = 0, apenas leitura. CLR' também é uma entrada que existe, mas serve para RESET assíncrono, que não usamos.

Em relação aos registradores, o banco recebe 3 entradas: Ra, Rb e Rd, todas de 4 bits. Preste atenção no fato de que Ra, Rb e Rd dizem respeito a registradores, que são a mesma coisa - o que é Ra em uma operação pode ser Rb em outra e assim segue, não delimitando endereços exclusivos de cada.

Ra e Rb são os registradores que fazem as operações, enquanto Rd é o registrador que recebe o valor dessas operações. Na entrada D, também de 16 bits, entram os dados que devem ser registrados em Rd. Em uma operação de soma entre Ra, Rb e 0, por exemplo, D = Ra (o que está armazenado) + Rb (o que está armazenado). Já as saídas desse banco são justamente Ra e Rb, com as quais são feitas as operações que resultam em D.

A implementação do RESET síncrono segue a mesma lógica do RESET do registrador PC: é usado um MUX 2 : 1 de 16 bits, cuja entrada de seleção está conectada ao RESET. Quando RESET = 1, a saída do MUX é a entrada que tem 16 bits de 0, e quando RESET = 0, a saída do MUX é a entrada normal de dados.

A entrada de cada registrador está conectada, portanto, a esse MUX que tem como

entradas de dados 16 bits de 0 e os 16 bits de dados inseridos.

A saída do banco também usa registradores para definir qual é a saída a ser selecionada dos 16 registradores para Ra e Rb. Nesse caso, são necessários 2 MUX 16 : 1, já que temos 16 registradores. As 4 entradas de seleção desses MUX são conectadas aos 4 bits de Ra e Rb.

No entanto, no Deeds só há no máximo MUX 8 : 1. Portanto, para implementar um MUX 16 : 1 é preciso usar 2 MUX, um 8 : 1 conectado a um 2 : 1, com as 3 entradas de seleção menos significativas no primeiro MUX e a mais significativa no segundo. A saída desses MUX 2 : 1 que se conectam as saídas Ra e Rb, de 16 bits.

Veja na figura abaixo como foi feita a implementação do banco de registradores.

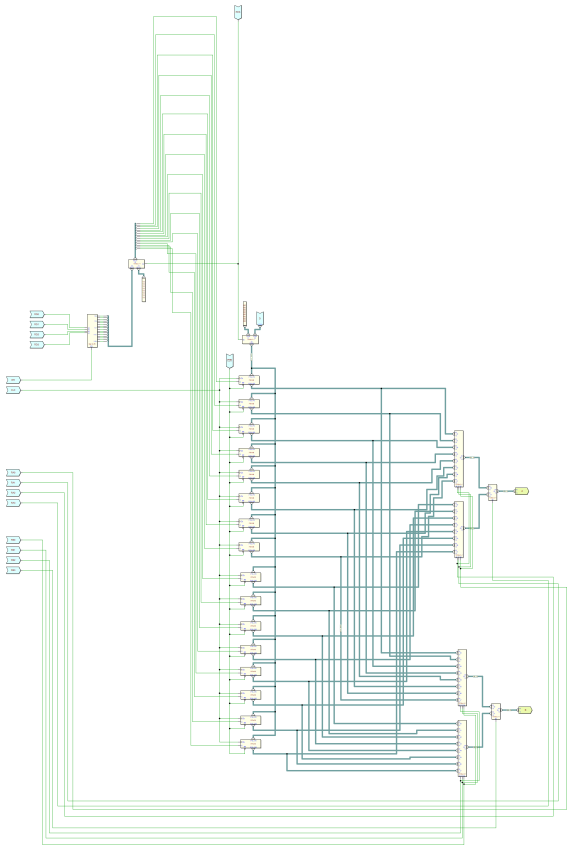

Figura 5. Implementação do banco de registradores.

2.4. Unidade Lógico-Aritmética

A ULA utilizada é fornecida diretamente pelo Deeds. No caso, foi utilizada a chamada ”ALU 16”disponibilizada pelo software. As operações utilizadas dali na implementação do processador foram: soma (11000), subtração (11011), AND bitwise (01000), OR bitwise (01100), XOR bitwise (10000).

Na ULA, há 5 entradas Si que permitem identificar qual é a operação realizada. Nas operações listadas, em parênteses estão os 5 bits que definem o que a ULA deve implementar.

Algumas entradas das ULAs foram desnecessárias neste projeto, como Z, V e Co, enquanto Ci foi fixado em 0.

Veja na figura abaixo como foi usada uma ULA para fazer a soma PC + 1.

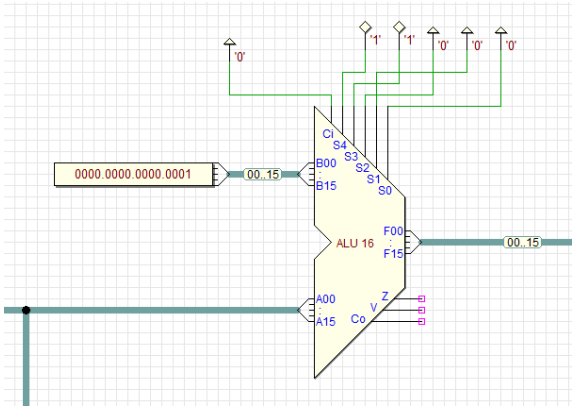

Figura 6. Uso de ULA para obter PC + 1.

2.5. Comparador

O comparador implementado recebe dois números A e B e entrega na saída se A é menor ou igual a B ou se A é igual a B - sendo considera operação com sinal ou sem sinal, o que é definido pela entrada SU = 1 (com sinal) ou SU = 0 (sem sinal).

Para esse comparador, foi aproveitado o comparador disponibilizado pelo Deeds. No entanto, esse comparador considera apenas operações sem sinal. Logo, foi necessário implementar as operações com sinal.

Para tanto, foram utilizados 2 comparadores conectados a um MUX, cuja entrada de seleção é SU, definindo se as saídas vêm do comparador com sinal ou sem sinal. No caso de operação sem sinal, nada precisa ser alterado e o MUX pode ser conectado diretamente às saídas do primeiro comparador.

Já no comparador com sinal, foi utilizado um MUX $4:1$. Aqui, há 4 casos a se considerar: A é negativo e B é positivo, sendo A e B sempre diferentes e A menor que B; A é positivo e B é negativo, sendo A e B sempre diferentes e A maior que B; A é negativo e B é negativo ou A é positivo e B é positivo, com as operações sem sinal entregando as comparações necessárias.

O que define se A e B é positivo ou negativo é o bit mais significativo das entradas A e B. Logo, esses bits podem ser conectados à entrada de seleção do MUX. Aqui foi conectado o bit mais significativo de B à entrada de seleção S1 e de A a S0. Quando S1 = 1 e S0 = 0, A é positivo e B é negativo, com o MUX retornando 001 (A $>$ B apenas).

Quando S1 = 0 e S0 = 1, A é negativo e B é positivo, com o MUX retornando 100 (A $<$ B apenas). Quando S1 = 1 e S0 = 1 ou S1 = 0 e S0 = 0, o MUX retorna as saídas do comparador sem sinal.

Observe que o comparador do Deeds tem 3 saídas: A<B; A==B; e A>B. No MUX utilizado, portanto, temos MUX $2:1$ de 4 bits. O bit mais significativo não é usado, sendo descartado na hora da seleção das saídas.

Já quanto às saídas, MOI = 1 quando A <=B e IEI = 1 quando A == B. Para MOI, foi necessário usar uma porta OR conectada às saídas A¡B e A==B. IEI pôde ser conectado diretamente ao fio da saída do MUX representando A==B.

Veja na figura abaixo como foi implementado o comparador usado.

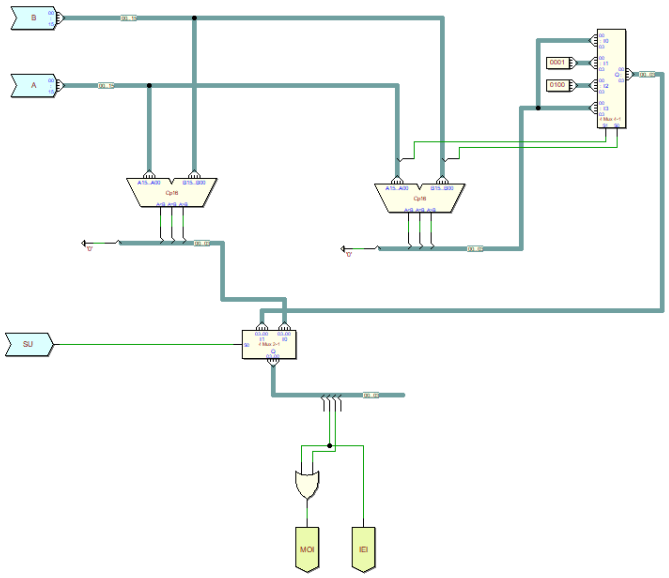

Figura 7. Implementação de comparador, que considera comparação com sinal e sem sinal dependendo da entrada SU.

2.6. Multiplexadores no processador

O projeto completo é conectado principalmente por MUX, que definem as condições de cada parte. Para entender o bloco de controle e suas saídas, é primeiro preciso ter noção de todos os MUX utilizados.

São utilizados 6 MUX para ligar todo o projeto. O MUX 1 tem sua saída diretamente ligada à entrada D do banco de registrados, tendo como entradas de seleção saídas do ponto de controle.

Esse MUX tem por função definir se o que vai ser registrado no banco de registradores é a saída da ULA (soma, subtração, AND, OR e XOR) ou é PC + 1 (casos de jal e jalr). O MUX 1 é MUX 2 : 1, em 0 sendo selecionada a entrada que vem da ULA e em 1 temos PC + 1.

Para a definição de qual operação obter do comparador, é utilizado o MUX COMP, cujas entradas de seleção vêm do ponto de controle. Aqui, usamos um MUX 4 : 1, em que as entradas são conectadas às saídas do comparador.

Com as entradas de seleção em 00, obtemos Ra <= Rb, quando 01 checamos Ra > Rb, quando 10 checamos Ra == Rb e quando 11 checamos Ra != Rb. Observe que para fazer essas entradas de dados em alguns casos é preciso conectar portas lógicas às saídas do comparador, já que este só retorna se A <= B e A == B.

Para fazer Ra <= Rb e Ra == Rb podemos usar as saídas dos comparadores sem alteração. Já para Ra > Rb, precisamos usar a saída com NOT de MOI, e para Ra != Rb,

a saída com NOT de IEI.

A saída desse MUX COMP é conectada às entradas de seleção do MUX 2, que define o próximo endereço de PC segundo instrução condicional (operações que necessitam do comparador). Aqui, é usado um MUX 2 : 1, em que 0 é conectado a PC + 1 e 1 a PC + Imm.

Já o MUX 3 define o próximo endereço de PC segundo instrução que não segue alguma condição (soma e subtração, por exemplo). É um MUX 2 : 1 cujas entradas de seleção vêm do ponto de controle. Nesse caso, em 0 temos a entrada PC + Imm e em 1 temos Ra + Imm.

O MUX COND define se o próximo endereço de PC alterado é condicional ou não. É um MUX 2 : 1 cujas entradas de seleção estão conectadas ao ponto de controle. Na entrada 0 temos a saída do MUX 3 (que define não condicional), enquanto em 1 temos a saída do MUX 2 (que define condicional).

Por fim, temos o MUX 4, que define se o próximo endereço PC é o padrão (PC + 1) ou alguma alteração. As entradas de seleção vêm do ponto de controle. Na entrada de dados 0, temos PC = PC + 1, enquanto na entrada 1 temos a saída do MUX COND, que entrega o próximo PC que tenha sofrido alguma alteração específica. A saída do MUX 4 é conectada à entrada do registrador PC.

2.7. Bloco de controle

O bloco de controle define todas as entradas que dependem do Opcode da instrução. Isto é, o bloco de controle recebe os 4 bits do Opcode da instrução e retornam na saída o WE do banco de registradores (que define se a operação é de escrita ou somente leitura), o Op da ULA (que define qual operação usar), o SU do comparador (se é com sinal ou sem) e as entradas de seleção dos MUX (somente os MUX que dependem do bloco de controle, como já explicado).

Essas saídas puderam ser feitas através de tabela verdade, com cada saída tendo sua equação simplificada obtida através de mapa de Karnaugh. Ao fim, chegamos em uma tabela de 4 entradas e 13 saídas. Veja abaixo a tabela resultante.

E3	E2	E1	E0	O4	O3	O2	O1	O0	W	S	M1	C1	C0	M3	MC	M4
0	0	0	0	1	1	0	0	0	1	X	0	X	X	X	X	0
0	0	0	1	1	1	0	1	1	1	X	0	X	X	X	X	0
0	0	1	0	0	1	0	0	0	1	X	0	X	X	X	X	0
0	0	1	1	0	1	1	0	0	1	X	0	X	X	X	X	0
0	1	0	0	1	0	0	0	0	1	X	0	X	X	X	X	0
0	1	0	1	X	X	X	X	X	0	X	X	1	0	X	1	1
0	1	1	0	X	X	X	X	X	0	X	X	1	1	X	1	1
0	1	1	1	X	X	X	X	X	0	1	X	0	0	X	1	1
1	0	0	0	X	X	X	X	X	0	0	X	0	0	X	1	1
1	0	0	1	X	X	X	X	X	0	1	X	0	1	X	1	1
1	0	1	0	X	X	X	X	X	0	0	X	0	1	X	1	1
1	0	1	1	X	X	X	X	X	1	X	1	X	X	0	0	1
1	1	0	0	X	X	X	X	X	1	X	1	X	X	1	0	1
1	1	0	1	X	X	X	X	X	X	X	X	X	X	X	X	X
1	1	1	0	X	X	X	X	X	X	X	X	X	X	X	X	X
1	1	1	1	X	X	X	X	X	X	X	X	X	X	X	X	X

Figura 8. Tabela de entradas e saídas do bloco de controle.

Nessa tabela, Ei são os valores de Opcode, inseridos na entrada. Oi diz respeito ao Opcode da ULA; W representa a saída WE; S representa a saída SU; M1 é a entrada de seleção do MUX 1; Ci são as entradas de seleção do MUX COMP; M3 é a entrada de seleção do MUX 3; MC é a entrada de seleção do MUX COND; e M4 é a entrada de seleção do MUX 4.

Observe que boa parte dessa tabela é preenchida por valores ”don’t care”. Isso está ligado ao fato de algumas decisões no MUX já descartarem várias operações.

Se o próximo endereço PC for o simples PC + 1, por exemplo em uma operação de adição, todos os MUX que alteram PC de forma diferente podem ter qualquer valor nas entradas de seleção, já que suas saídas não serão usadas.

Observe abaixo a implementação do bloco de controle usado.

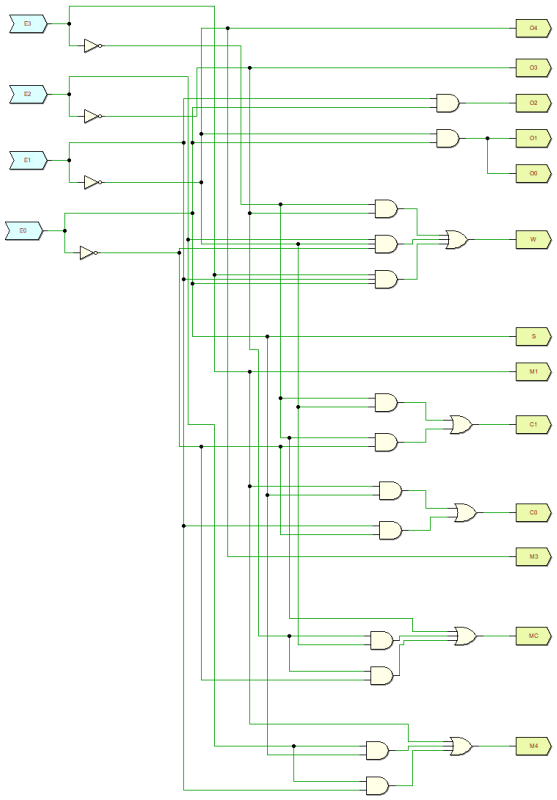

Figura 9. Implementação do bloco de controle.

2.8. Sinais de monitoramento

Os sinais de monitoramento são displays de 7 segmentos hexadecimais. Observe que cada display recebe 8 bits na entrada. Assim, foi necessário mais de um display para exibir cada um dos dados pedidos.

Para exibir o registrador PC, foram necessários 2 displays, já que PC tem 16 bits. Já para a instrução completa, foram 4 displays para exibir os 32 bits. Para Ra e Rb também foram necessários 2 displays cada, com cada registrador tendo 16 bits.

Preste atenção na ordem do que é exibido: o primeiro dígito de cima para baixo é o MSB, e o último, mais embaixo, é o LSB.

Observe abaixo como ficaram os sinais de monitoramento implementados.

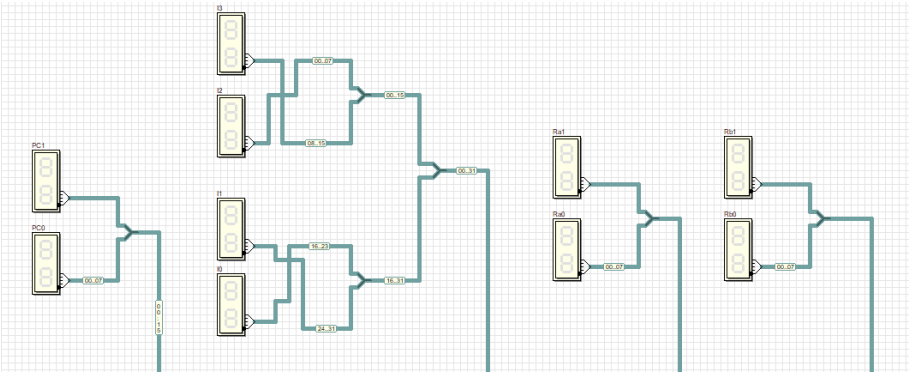

Figura 10. Sinais de monitoramento do processador, onde temos por ordem: PC, instrução, Ra e Rb.

3. Resultados Obtidos

Foram elaborados programas para testar o funcionamento do processador. Cada um desses programas pode ser testado com uma memória já preenchida e anexada junto ao projeto, com arquivos .drs.

A explicação detalhada do funcionamento de cada programa e da sua elaboração pode ser vista abaixo, tendo sido implementados os programas de: multiplicação sem e com sinal; divisão sem e com sinal; e resto sem divisão.

3.1. Multiplicador sem signal

Para fazer a multiplicação utilizando o processador montado, basta entender a lógica da multiplicação feita a partir de soma. Se temos: [Mathonline 2021]

$$R1 = R2 \times R3 \tag{1}$$

Temos o multiplicando R2 e o multiplicador R3. O resultado da multiplicação é o valor de R2 somado consigo mesmo R3 vezes. 2x4, por exemplo, pode ser entendido como:

$$2 \times 4 = 2 + 2 + 2 + 2 \tag{2}$$

No processador, isso pode ser executado fazendo um laço a partir do qual R3 é decrementado até chegar ao valor 0. A cada decréscimo, o valor de R2 é somado a R1, que no fim tem o valor da multiplicação.

Na tabela abaixo, é possível ver o exemplo fazendo R2 = 6 e R3 = 4, onde chegamos ao resultado:

$$6 \times 4 = 24 \tag{3}$$

Endereço	Código hexadecimal	Instrução	Comentário
0x0000	0x0006 3000	addi R2, R2, R0, 6	R2 = 6
0x0001	0x0004 3110	addi R3, R3, R0, 4	R3 = 4
0x0002	0x0000 0220	addi R1, R1, R2, 0	Inicia loop
0x0003	0x0001 3111	subi R3, R3, R0, 1	R3 = R3 - 1
0x0004	0xFFFE 3116	bne R3, R0, -2	R3 != R0 ? Loop : Next
0x0005	0x0000 2225	beq R1, R1, 0	Mostra R1

Figura 11. Tabela com endereço de memória e instrução de multiplicação sem sinal, com R2 = 6 e R3 = 4.

O multiplicador sem sinal pode ser executado no processador desenvolvido com frequência máxima de 5MHz. O exemplo desenvolvido na tabela acima pode ser visto detalhadamente na simulação temporal abaixo, na frequência máxima.

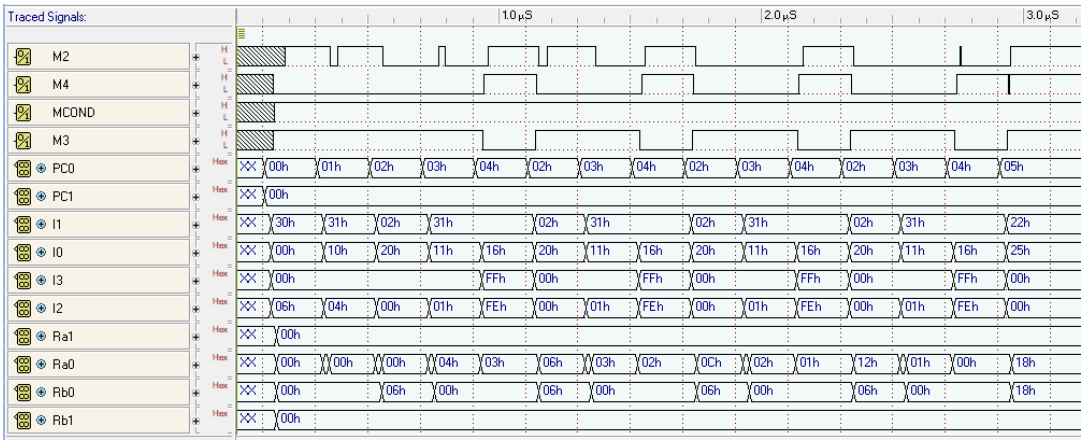

Figura 12. Simulação temporal de multiplicador sem sinal implementado, com R2 = 6 e R3 = 4.

Veja neste vídeo a simulação mostrando o funcionamento do multiplicador sem sinal.

3.2. Multiplicador com sinal

Na multiplicação com sinal, a lógica sofre uma mudança dependendo do sinal do multiplicador: se o multiplicador é negativo, ele incrementa até chegar a 0 (não decresce até 0) e R1 tem seu sinal invertido ao fim das operações.

Caso o multiplicador seja positivo, as operações realizadas são as mesmas das de multiplicação sem sinal.

Na prática, é acrescentado um bloco de código novo, que satisfaz essa condição se... então que verifica o sinal do multiplicador.

Nesse caso, 2x-4 aconteceria em um laço com R3 que passaria de -4 para -3 (com R1 = 2), -3 para -2 (R1 = 4), -2 para -1 (R1 = 6) e -1 para 0 (R1 = 8). A inversão do sinal então ocorre, com R1 = -8.

A inversão de sinal satisfaz mesmo os casos onde multiplicando e multiplicador são negativos. Isso porque quando o multiplicador R3 chega em 0, chegamos a uma soma com valor negativo. No entanto, quando 2 valores de sinal igual são multiplicados, o resultado é positivo.

Na tabela abaixo, é possível ver o exemplo fazendo R2 = -6 e R3 = 4, onde chegamos ao resultado:

$$-6 \times 4 = -24 \tag{4}$$

Endereço	Código hexadecimal	Instrução	Comentário
0x0000	0xFFFA 3010	addi R2, R1, R0, -6	R2 = -6
0x0001	0x0004 3220	addi R3, R3, R0, 4	R3 = 4
0x0002	0x0005 3227	ble R3, R0, 5	R3 <= 0 ? Next : Loop (multiplicador positivo)
0x0003	0x0000 1000	addi R1, R1, R2, 0	R1 = R1 + R2 (loop)
0x0004	0x0001 3221	subi R3, R3, R0, 1	R3 = R3 - R0 - 1
0x0005	0xFFFE 3226	bne R3, R0, -2	R3 != R0 ? PC - 2 : PC + 1 (fim)
0x0006	0x0000 0005	beq R1, R1, 0	R1 == R1 ? PC + 0 : PC + 1 (mostra R1)
0x0007	0x0000 1000	addi R1, R1, R2, 0	R1 = R1 + R2 (loop - multiplicador negativo)
0x0008	0x0001 3220	addi R3, R3, R0, 1	R3 = R3 + 1
0x0009	0xFFFE 3226	bne R3, R0, -2	R3 != R0 ? PC - 2 : PC + 1 (fim)
0x000A	0x0000 0301	subi R1, R0, R1, 0	R1 = R0 - R1
0x000B	0x0000 0005	beq R1, R1, 0	Mostra R1

Figura 13. Tabela com endereço de memória e instrução de multiplicação com sinal, com R2 = -6 e R3 = 4.

O multiplicador com sinal pode ser executado no processador desenvolvido com frequência máxima de 5MHz. O exemplo desenvolvido na tabela acima pode ser visto detalhadamente na simulação temporal abaixo, na frequência máxima.

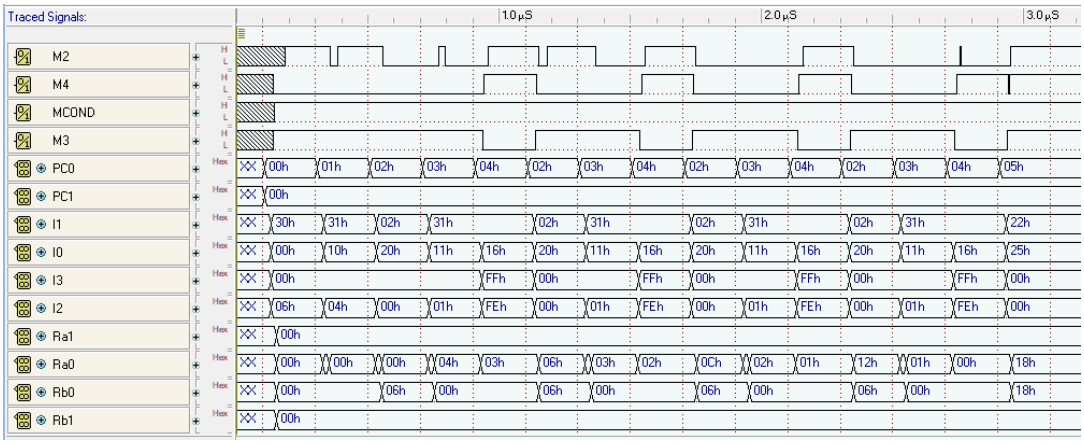

Figura 14. Simulação temporal de multiplicador com sinal implementado, com R2 = -6 e R3 = 4.

Veja neste vídeo a simulação mostrando o funcionamento do multiplicador com sinal.

3.3. Divisor sem sinal

Para a divisão sem sinal, foi necessário revisar a lógica da divisão, onde: [Mathonline 2021]

$$R1 = R2 \div R3 \tag{5}$$

Uma divisão pode ser entendida como uma soma imaginando-se um R4 que começa com 0 e é somado com o divisor a cada iteração, com o quociente R1 incrementando em 1 a cada soma. O laço de soma R4 = R4 + R3 para quando R4 é maior ou igual R2. Se R4 é igual a R3, tem-se uma divisão exata, mas se R4 é maior que R3, a divisão não é exata, sobrando um resto.

No programa desenvolvido, são feitas 2 checagens dentro do laço: se R4 $>$ R2, então a divisão tem resto, e nessa última iteração R1 não é incrementado; ou se R4 ==R2, então a divisão não tem resto e na última iteração R1 é incrementado.

Observe que essa última iteração é diferente para cada caso porque para se chegar em R4 $>$ R2 é preciso fazer uma soma a mais do que o necessário. Em 9/2, por exemplo, temos que o quociente é 4 (com 2 + 2 + 2 + 2 = 8 sendo o valor mais próximo de 9).

No entanto, só sabemos que de fato essa divisão tem resto quando chegamos em 10, que é maior que 9. A última soma, portanto, é descartada na contagem de R1.

Veja que é uma divisão inteira, então se o resultado não é exato, o quociente é somente a parte inteira do quociente real (9/2 resulta em 4,5 e não 4).

Com 8/2, por sua vez, são necessárias 4 somas de 2 para se chegar exatamente em 8. Aqui, a lógica de incrementar R1 a cada soma com o divisor não muda em nada.

Na tabela abaixo, é possível ver o exemplo fazendo R2 = 9 e R3 = 2, onde:

$$4 = 9 \div 2 \tag{6}$$

Endereço	Código hexadecimal	Instrução	Comentário
0x0000	0x0009 3110	addi R2, R2, R0, 9	R2 = 9
0x0001	0x0002 3220	addi R3, R3, R0, 2	R3 = 2
0x0002	0x0000 2440	addi R4, R4, R3, 0	R4 = R4 + R3 (loop)
0x0003	0x0003 1449	bgt R4, R2, 3	R4 > R2 ? PC + 3 : PC + 1 (fim - divisão com resto)
0x0004	0x0001 3000	addi R1, R1, R0, 1	R1 = R1 + 1
0x0005	0xFFFD 1446	bne R4, R2, -3	R4 != R2 ? PC - 3 : PC + 1 (fim - divisão exata)
0x0006	0x0000 0005	beq R1, R1, 0	Mostra R1

Figura 15. Tabela com endereço de memória e instrução de divisão sem sinal, com R2 = 9 e R3 = 2.

O divisor sem sinal pode ser executado no processador desenvolvido com frequência máxima de 5MHz. O exemplo desenvolvido na tabela acima pode ser visto detalhadamente na simulação temporal abaixo, na frequência máxima.

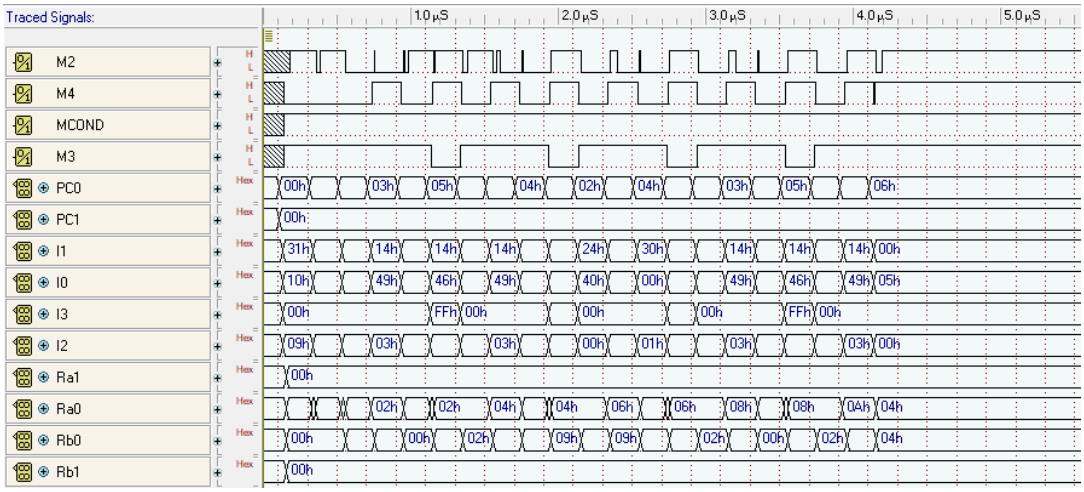

Figura 16. Simulação temporal de divisor sem sinal implementado, com R2 = 9 e R3 = 2.

Veja neste vídeo a simulação mostrando o funcionamento do divisor sem sinal.

3.4. Divisor com sinal

Na divisão com sinal implementada utiliza-se a mesma lógica da divisão sem sinal. A grande diferença se dá no fato de que no começo do programa os números negativos têm seu sinal invertido, sendo a divisão feita como se tivéssemos divisor e dividendo positivos.

Ao fim do programa, caso os sinais do divisor e do dividendo sejam diferentes, o quociente R1 é negativado. Do contrário, continua positivo.

No programa, o sinal de cada número é guardado em RI2 (sinal do dividendo) e RI3 (sinal do divisor). Quando RI = 0, o sinal é positivo, e quando RI = 1, o sinal é negativo.

Com isso, é possível entender a divisão como o número de somas do divisor até chegar em um número igual ou maior que o dividendo. Também aqui é aplicada a condição que verifica se a divisão é exata ou não, o que afeta quantas vezes o quociente é incrementado.

Com -9/2, primeiro guardamos o sinal de -9 em RI2 = 1, depois seguimos (RI3 = 0 automaticamente, já que o projeto inicia com os registradores zerados) para a soma R4 = 2 + 2 + 2 + 2 + 2 = 10, que é maior que 9, logo R1 = 4 (número de 2 na soma que resulta em R4, com exceção do último 2). Ao fim, é checado que RI2 != RI3, logo 4 é invertido através da subtração R1 = R0 - R1 - 0.

Na tabela abaixo é possível ver as instruções para fazer -9/2, em que:

$$-4 = -9 \div 2 \tag{7}$$

Endereço	Código hexadecimal	Instrução	Comentário
0x0001	0xFFF7 3110	addi R2, R2, R0, -9	R2 = -9
0x0002	0x0003 3119	bgt R2, R0, 3	R2 > R0 ? PC + 3 : PC + 1
0x0003	0x0001 3550	addi RI2, RI2, R0, 1	RI2 = RI2 + 1 (guarda sinal de R2)
0x0004	0x0000 1311	subi R2, R0, R2, 0	R2 = R0 - R2 (inverte R2 negativo)
0x0005	0x0002 3220	addi R3, R3, R0, 2	R3 = 2
0x0006	0x0003 3229	bgt R3, R0, 3	R3 > R0 ? PC + 3 : PC + 1
0x0007	0x0001 3660	addi RI3, RI3, R0, 1	RI3 = RI3 + 1 (guarda sinal de R3)
0x0008	0x0000 2321	subi R3, R0, R3, 0	R3 = R0 - R3 (inverte R3 negativo)
0x0009	0x0000 2440	addi R4, R4, R3, 0	R4 = R4 + R3 (loop)
0x000A	0x0003 1449	bgt R4, R2, 3	R4 > R2 ? PC + 3 : PC + 1 (fim - divisão com resto)
0x000B	0x0001 3000	addi R1, R1, R0, 1	R1 = R1 + 1
0x000C	0xFFFD 1446	bne R4, R2, -3	R4 != R2 ? PC - 3 : PC + 1 (fim - divisão exata)
0x000D	0x0002 6555	beq RI2, RI3, 2	RI2 == RI3 ? PC + 2 : PC + 1
0x000E	0x0000 0301	subi R1, R0, R1, 0	R1 = R0 - R1 (inverte caso R2 ou R3 negativo)
0x000F	0x0000 0005	beq R1, R1, 0	Mostra R1

Figura 17. Tabela com endereço de memória e instrução de divisão com sinal, com R2 = -9 e R3 = 2.

O divisor com sinal pode ser executado no processador desenvolvido com frequência máxima de 5MHz. O exemplo desenvolvido na tabela acima pode ser visto detalhadamente na simulação temporal abaixo, na frequência máxima.

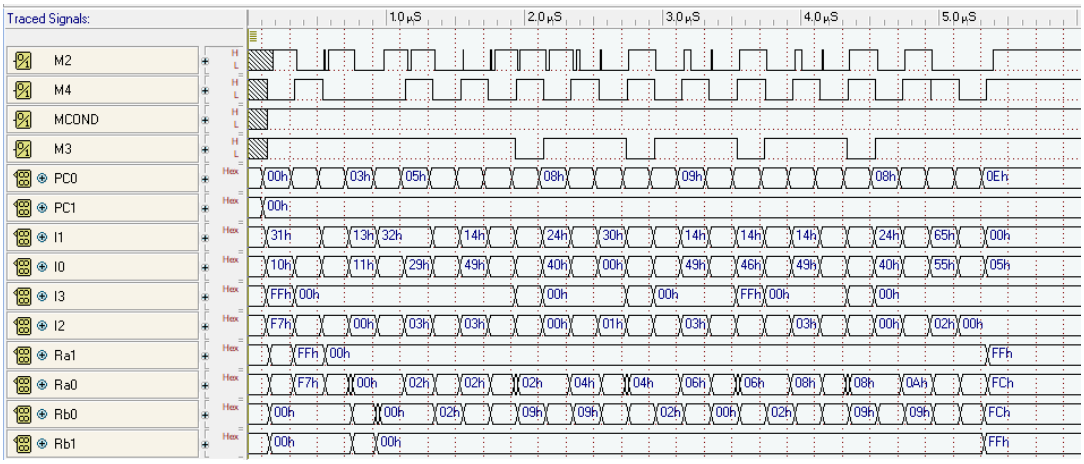

Figura 18. Simulação temporal de divisor com sinal implementado, com R2 = -9 e R3 = 2.

Veja neste vídeo a simulação mostrando o funcionamento do divisor com sinal.

3.5. Resto de divisão sem sinal

No programa que implementa resto de divisão sem sinal, foi aproveitado o programa desenvolvido para a divisão sem sinal. Isso porque R4 mostra a multiplicação do quociente pelo divisor, número que quando subtraído ao dividendo é igual ao resto na divisão sem sinal. [Cuemath 2021]

Dessa forma, aqui precisamos apenas aplicar a lógica já descrita, interrompendo o laço quando R4 é maior ou igual ao dividendo R2. R1 já não é incrementado aqui, sendo apenas a subtração R1 = R2 - R4.

Observe que aqui a checagem se a divisão é exata ou com resto é novamente aplicada, com a soma R1 = R2 - R4 não sendo feita na última iteração quando a divisão não é exata.

No caso de 9/2, por exemplo, isso acontece porque se subtraíssemos a última iteração teríamos 9 - 10 = -1, um resto incorreto. Parando a subtração antes da última iteração conseguimos 9 - 8 = 1.

Na tabela abaixo é possível ver as instruções para fazer 9%2, em que:

$$2 = 17\%3 \tag{8}$$

Endereço	Código hexadecimal	Instrução	Comentário
0x0000	0x0011 3110	addi R2, R2, R0, 17	R2 = 17
0x0001	0x0003 3220	addi R3, R3, R0, 3	R3 = 3
0x0002	0x0000 2440	addi R4, R4, R3, 0	R4 = R4 + R3 (loop)
0x0003	0x0003 1449	bgt R4, R2, 3	R4 > R2 ? PC + 3 : PC + 1 (fim - divisão com resto 0)
0x0004	0x0000 4101	subi R1, R2, R4, 0	R1 = R2 - R4
0x0005	0xFFFD 1446	bne R4, R2, -3	R4 != R2 ? PC - 3 : PC + 1 (fim - divisão com resto diferente de 0)
0x0006	0x0000 0005	beq R1, R1, 0	Mostra R1

Figura 19. Tabela com endereço de memória e instrução de resto sem sinal, com R2 = 17 e R3 = 3.

O programa que obtém o resto de divisão sem sinal pode ser executado no processador desenvolvido com frequência máxima de 5MHz. O exemplo desenvolvido na tabela acima pode ser visto detalhadamente na simulação temporal abaixo, na frequência máxima.

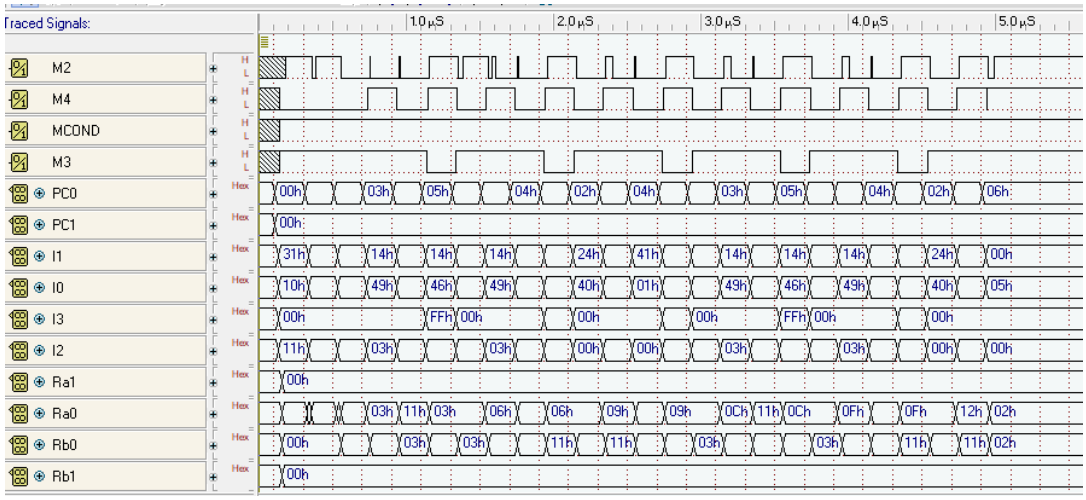

Figura 20. Simulação temporal de programa que exibe resto de divisão sem sinal implementado, com R2 = 17 e R3 = 3.

Veja neste vídeo a simulação mostrando o funcionamento do programa que mostra o resto de divisão sem sinal.

3.6. Frequência máxima do processador

O limite de frequência máxima que o processador consegue responder está ligado aos limites dos registradores. Os registradores são, na verdade, flip-flops, com n bits sendo

implementados por n flip-flops. Nesse processador, utilizamos um total de 17 registradores de 16 bits (16 no banco de registradores e 1 para o registrador PC).

Como para cada bit é necessário um flip-flop, temos que para esse processador foram necessários 272 flip-flops apenas em registradores. Assim, todos os programas desenvolvidos aqui têm uma frequência máxima de cerca de 5 MHz, a partir de onde o processador não entrega mais os dados de forma confiável.

Observe que esse limite é independente do tamanho de operações necessárias para o programa executar. Assim, o programa de multiplicação sem sinal pode ser executado em no máximo 5 MHz de frequência, o mesmo que o programa de divisão com sinal, que é mais complexo em números de linhas.

4. Conclusão

O processador pôde ser desenvolvido com sucesso, com todas suas operações sendo implementadas, assim como todos seus componentes. O projeto permitiu colocar em práticas vários conteúdos estudados ao longo da disciplina em um único circuito, bastante útil em aplicações da vida cotidiana.

Os programas desenvolvidos para serem executados no processador também puderam ser desenvolvidos em quase sua totalidade, permitindo ao estudante uma aproximação com a programação em nível mais baixo. Os programas de multiplicação sem sinal e com sinal, divisão sem sinal e com sinal, e resto sem sinal puderam ser implementados com sucesso.

O estudo do impacto da frequência no ZeptoProcessador-III também pôde ser feito, identificando-se a frequência máxima para os programas desenvolvidos e sendo definida uma causa para esse limite.

Referências

[Cuemath 2021] Cuemath (2021). Remainder. https://www.cuemath.com/numbers/remainder/. [Online; accessed 5-November-2021].

[Koike 2021a] Koike, C. C. (2021a). Aula de circuitos sequenciais: Registradores, contador em anel. https://www.youtube.com/watch?v=WbDEeDVp-5I. [Online; accessed 5-November-2021].

[Koike 2021b] Koike, C. C. (2021b). Aula de multiplexadores. https://www.youtube.com/watch?v=qqg3FRDokTY. [Online; accessed 5-November-2021].

[Lamar and Mandelli 2021] Lamar, M. V. and Mandelli, M. G. (2021). Projeto final - zeptoprocessador-iii. [Online; accessed 5-November-2021].

[Mandelli 2021] Mandelli, M. G. (2021). Aula de lcl - 08/10/2021. https://www.youtube.com/watch?v=e6MjyRlYCeg. [Online; accessed 5-November-2021].

[Mathonline 2021] Mathonline (2021). The multiplication and division principles. http://mathonline.wikidot.com/the-multiplication-and-division-principles. [Online; accessed 5-November-2021].

[Wikipedia 2021a] Wikipedia (2021a). Arithmetic logic unit. https://en.wikipedia.org/wiki/Arithmetic_logic_unit. [Online; accessed 5-November-2021].

[Wikipedia 2021b] Wikipedia (2021b). Central processing unit. https://en.wikipedia.org/wiki/Central_processing_unit. [Online; accessed 5-November-2021].