Projeto Final

Marcelo Junqueira, 200023624
Vinicius Lima, 200028545
Grupo G2

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

viniciuslimapassos@gmail.com, marcelojunqueiraf@gmail.com

Abstract. *This report details the development of a 16-bit ZeptoProcessador-II by combining the structures in reports. Throughout the document, its objectives are sampled and, just like the methods and materials used. Then, the procedures for implementing the 8-instruction programmable processor, which executed programs designed to test its operation, are discussed. Subsequently there is a more detailed analysis of the interaction between the parts of the processor, in addition to other deductions.*

Resumo. *Nesse relatório há o detalhamento do desenvolvimento de um ZeptoProcessador-II de 16 bits por meio da combinação das estruturas abordas em relatórios anteriores. Ao longo do documento ocorre a amostragem de seus objetivos e finalidade, tal qual os métodos e materiais utilizados. Em seguida, discorrem-se os devidos procedimentos envolvendo implementação do processador programável de 8 instruções, o qual executou programas elaborados para testar seu funcionamento. Posteriormente há uma análise mais detalhada da interação entre as partes do processador, além de outras deduções.*

1. Introdução

Processadores são estruturas extremamente presentes no contexto da atualidade e são também o auge da complexidade dos circuitos lógicos. Um processador é responsável por executar uma série de instruções, incluindo funções aritméticas, lógicas e de controle. Existem diversos tipos de processadores, aprimorados ao longo da história. Para este relatório foi implementado um ZeptoProcessador-II de 16 bits de 8 instruções, verificando assim seu funcionamento e os métodos de construções desse elaborado circuito.

1.1. Objetivos

O experimento tem como objetivo verificar a implementação de um processador, mais especificamente um de 16 bits com as instruções addi (soma), subi (subtração), andi (and bitwise), ori (or bitwise), xori (xor bitwise), beq (branch se igual), bleu (branch se menor igual sem sinal) e bles (branch se menor igual com sinal). Instruções essas que serão posteriormente testadas por meio de programas simples. Ocorrendo dessa forma, uma análise do processador como um todo e de suas frequências máximas de funcionamento.

1.2. Materiais

Neste experimento foi utilizado o software Deeds, além do livro Sistemas Digitais Princípios e Aplicações para formulação da base teórica [2] e como base para o experimento o roteiro[1].

2. Procedimentos

Os experimentos consistiram de modelos digitais no software Deeds, o qual permite a esquematização direta de circuitos por meio de portas lógicas e outros componentes teóricos sem a preocupação de especifidades físicas, como a necessidade de usar circuitos integrados. Alguns trechos foram filmados e colocados na plataforma digital do youtube. (Clique Aqui)

2.1. REGISTRADORES

Para compor o circuito, foram utilizados registradores fornecidos pelo próprio DEEDS, o qual contém 16 bits, segue o modelo PIPO (Parallel-in to Parallel-out) e é escrito na borda de subida do clock.

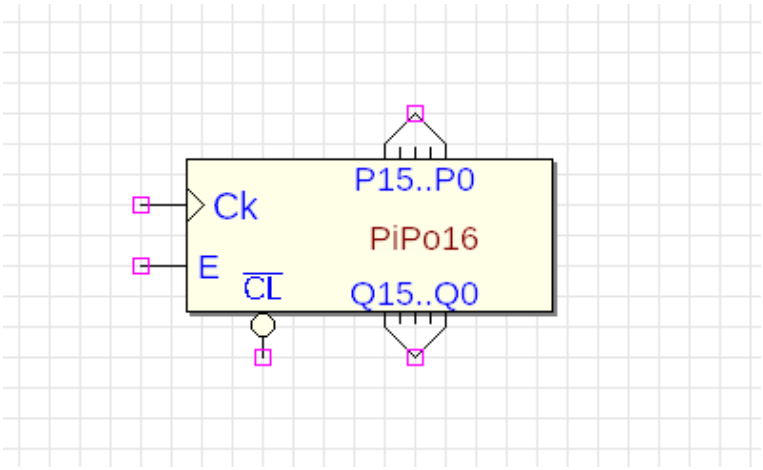

Figura 1. Registrador PIPO

2.2. MEMÓRIA DE INSTRUÇÕES

Para armazenar os programas elaborados pelo usuário, houve o uso de memórias ROMs fornecidas pelo DEEDs. Todavia, como o módulo fornecido possui apenas 16 bits por instrução, foi necessária a combinação de duas memórias para compartimentalizar as instruções em duas half-words.

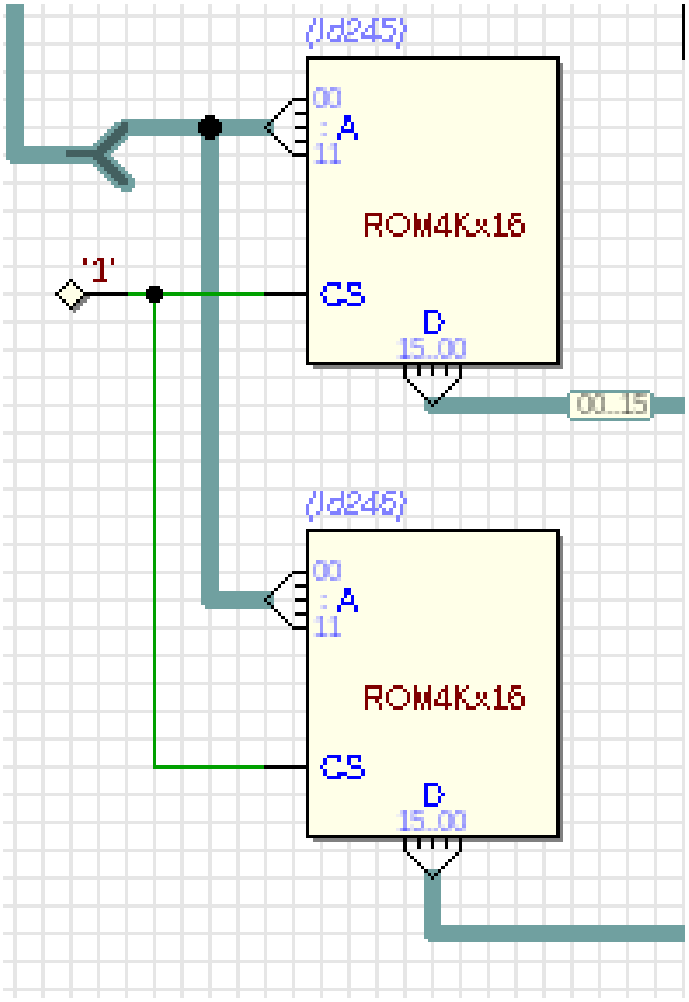

Figura 2. Memória de instruções

2.3. BANCO DE REGISTRADORES

Para o funcionamento geral do processador e processamento das instruções, foi implementado um banco de 16 registradores que permite a leitura simultânea de dois registradores e a escrita em um terceiro, além da presença do registrador zero apenas para leitura com o valor fixo em 0, funcionando a partir da borda de subida do clock. A estrutura funciona a partir de multiplexadores que selecionam os registradores corretos a partir dos valores fornecidos no opcode.

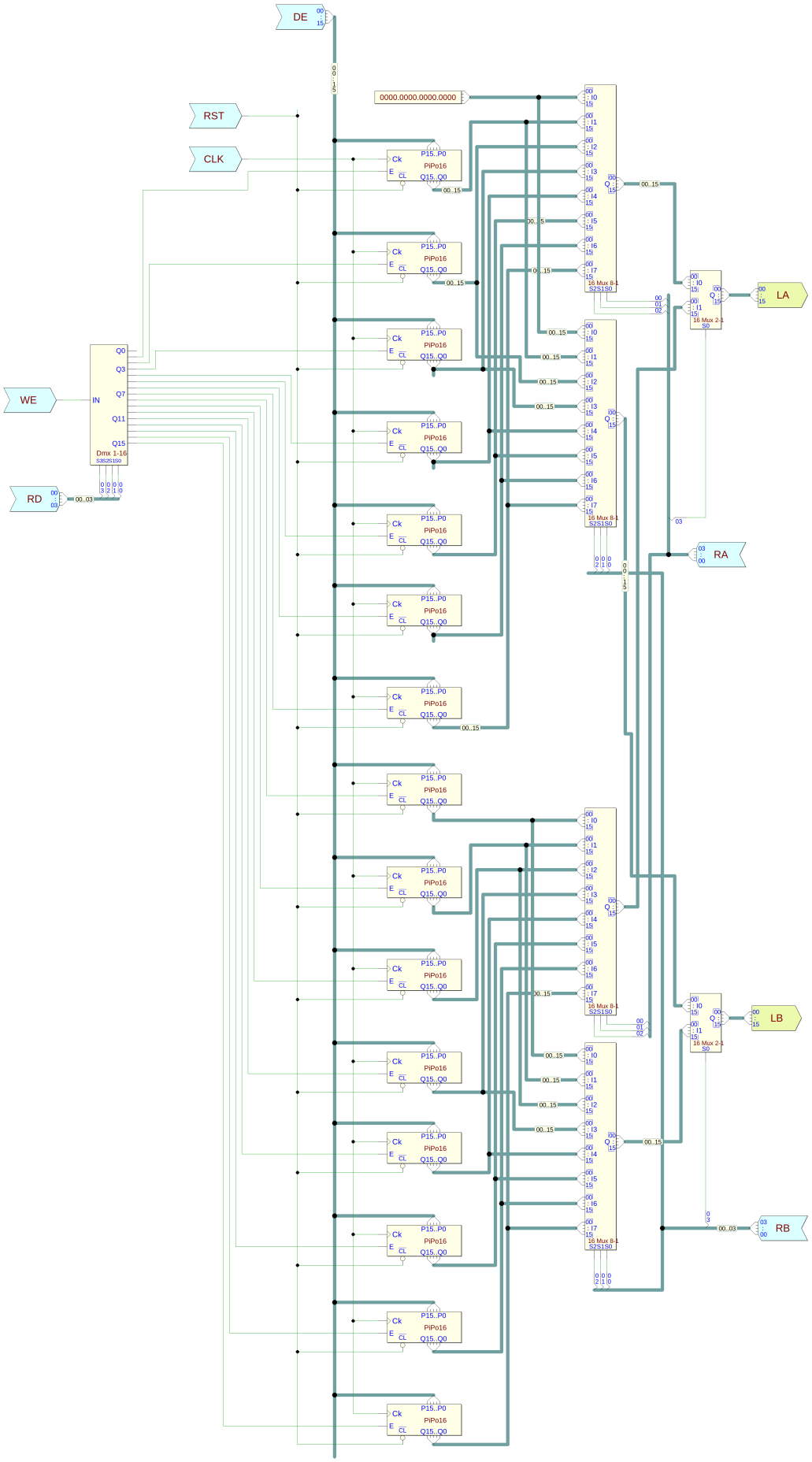

Figura 3. Banco de Registradores

2.4. UNIDADE LÓGICO-ARITMÉTICA

Para realizar as operações aritméticas, foi projetada uma ULA utilizando a unidade também provida pelo DEEDs como base. A ULA recebe os opcodes gerados pelo bloco de controle para decidir suas operações e atuar por meio de dois módulos, o primeiro para os dois registradores A e B e o segunda para interfacear com o valor do imediato.

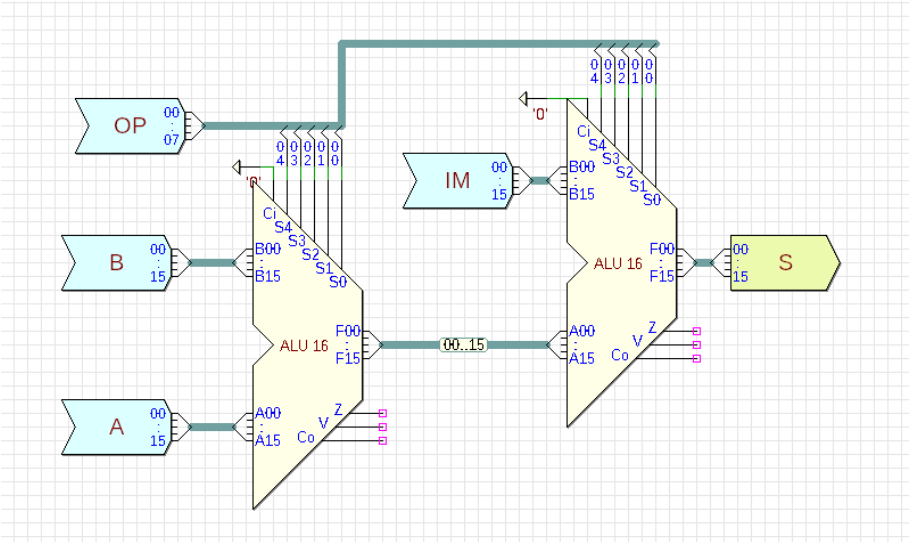

Figura 4. Unidade lógico-aritmética

2.5. COMPARADOR

Para avaliar as comparações pedidas pelas instruções de salto, foi necessário construir um comparador. Também foi utilizado o comparador fornecido pelo DEEDs, porém, o módulo pré-pronto realiza comparações apenas entre números não sinalizados, o que fez com que houvesse a necessidade de adaptação construindo circuitos paralelos para que a comparação sinalizada pudesse ser executada corretamente, além da junção das saídas ”menor”e ”igual”para a ”menor ou igual”.

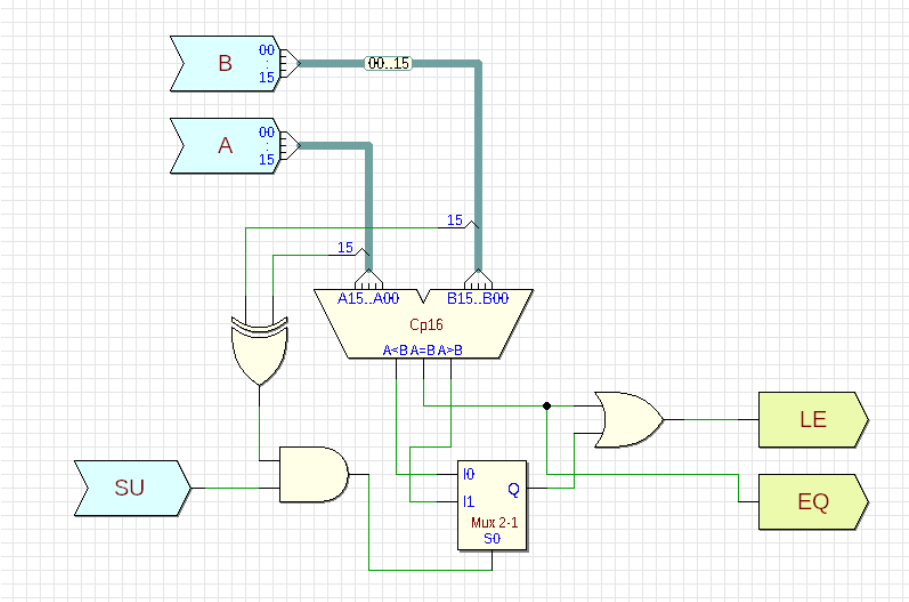

Figura 5. Comparador

2.6. BLOCO DE CONTROLE

Foi construído um bloco de controle para gerar os sinais de controle, sendo esses, o sinal de habilitação de escrita nos registradores, o de definição entre comparação sinalizada e não sinalizada no comparador, para operações na ULA e estados de multiplexadores que definem como o PC irá se comportar. Para mandar sinais de operação corretos para ula, foi utilizado um multiplexador que seleciona o sinal adequada a partir dos bits selecionados do opcode.

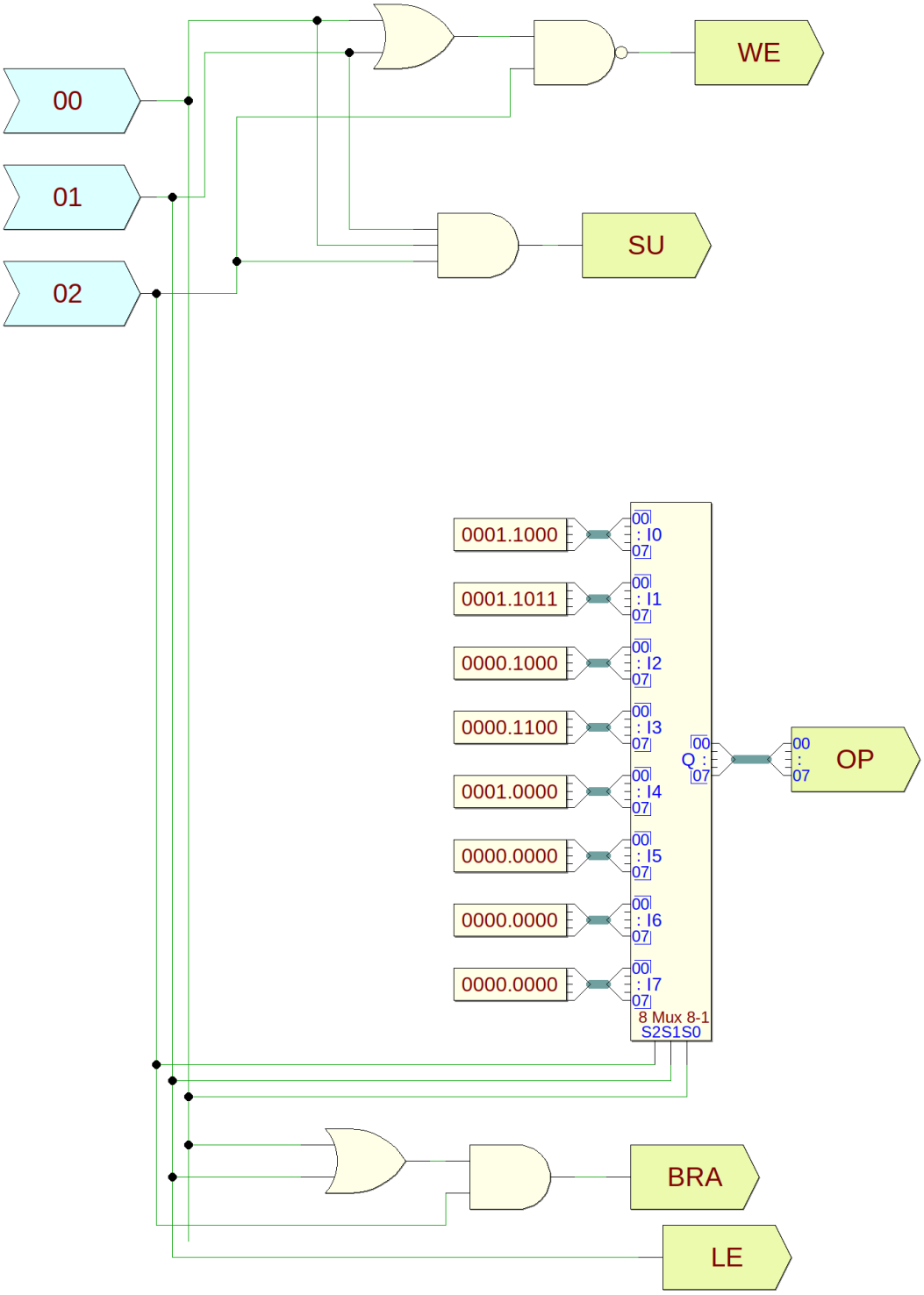

Figura 6. Bloco de controle

2.7. PROGRAM COUNTER

Para que a posição em programas pudesse ser manuseada e monitorada, foi elaborada a estrutura do Program counter (PC), utilizando um registrador de 16 bits para guardar o

endereço da memória e alterá-lo. A estrutura utiliza os sinais enviados pelo bloco de controle para decidir se endereço avançará normalmente ou se será somado com o imediato, permitindo dessa forma saltos na execução.

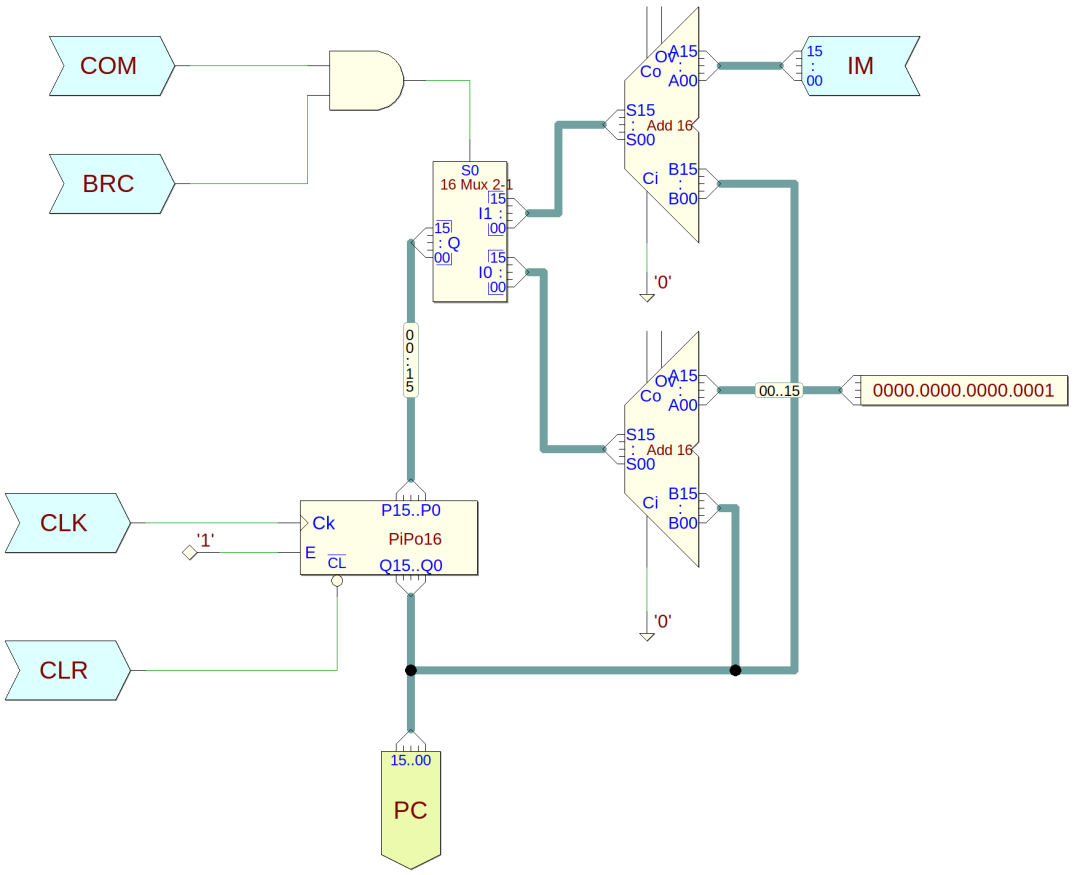

Figura 7. Program Counter

2.8. ESTRUTURA GERAL E FREQUÊNCIA MÁXIMA

Para manter a organização, os componentes supracitados foram compartimentalizados em blocos de circuito, adicionando também diversos displays de monitoramento, permitindo a visualização dos estados do circuito, bem como a facilitação da correção de erros.

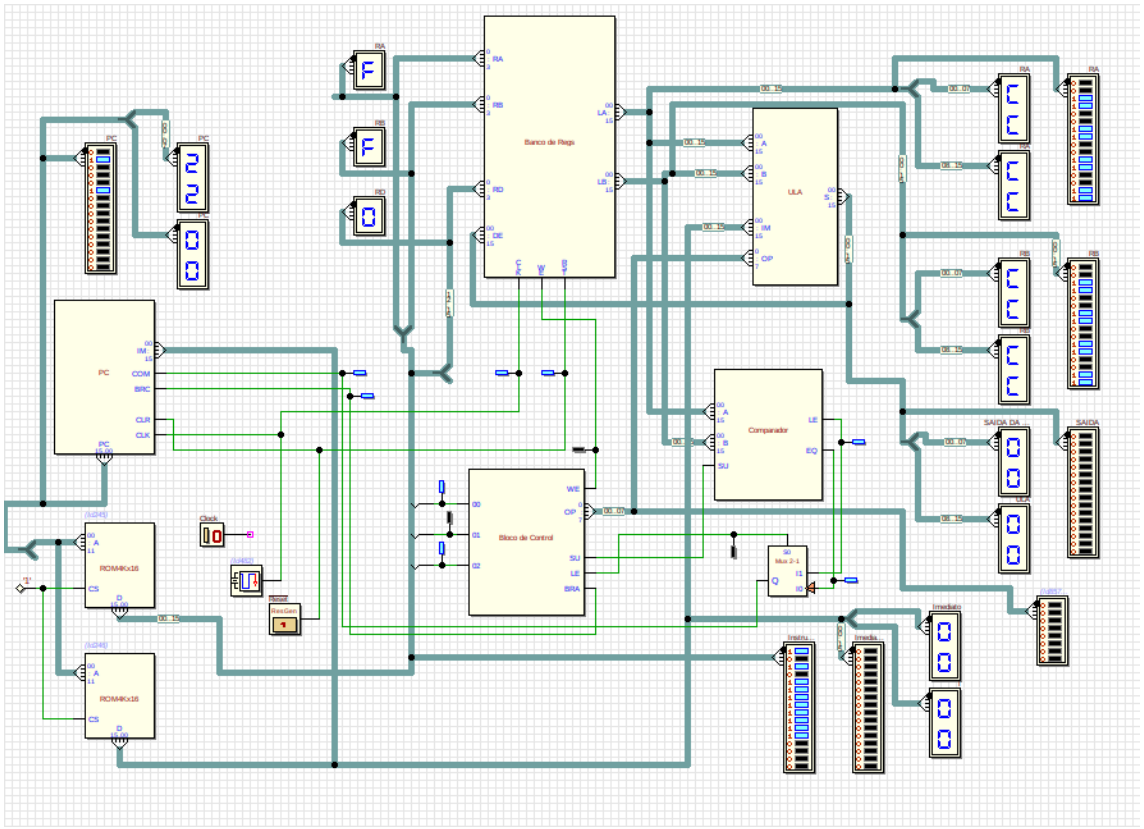

Figura 8. Estrutura Geral

Além disso, a partir da tentativa e erro, foi observado que a frequência máxima de funcionamento do processador está em torno de 4,76 MHz. Todos os diagramas de onda exibidos nesse relatório encontram-se nessa frequência.

2.9. CONTADOR DE -16 A 16

O primeiro programa a ser testado no processador foi um contador de -16 a 16, o qual segue o seguinte código abaixo:

Endereço	Código hexadecimal	Instrução	Comentário
0x0000	0xFFF0 1001	subi R1,R0,R0,16	R1=-16
0x0001	0x0010 2000	addi R2,R0,R0,16	R2=16
0x0002	0x0001 2201	subi R2,R2,R0,1	R2=R2-1 devido ao =
0x0003 Loop:	0x0001 1100	addi R1,R1,R0,1	R1=R1+1
0x0004	0xFFFF 0127	jles R1,R2,-1	R0<=R1? Loop : Next
0x0005 Fim:	0x0000 0115	beq R1,R1,0	J Fim

Figura 9. Código fonte do contador

E sua forma de onda ao ser executado:

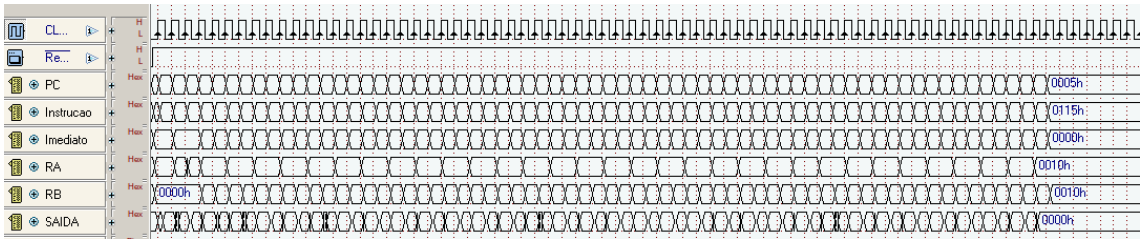

Figura 10. diagrama de onda contador de -16 a 16

2.10. SOMA DOS NÚMEROS ÍMPARES DE 0 A 15

Em seguida, foi executado um somador dos números ímpares de 0 a -15, que segue o seguinte código fonte:

Endereço	Código hexadecimal	Instrução	Comentário
0x0000	0x0001 1000	addi R1,R0,R0,1	R1=1 primeiro ímpar
0x0001	0x000F 2000	addi R2,R0,R0,15	R2=15 valor final
0x0002	0x0001 2200	addi R2,R2,R0,1	R2=R2+1 (devido ao =)
0x0003 Loop:	0x0004 0216	jleu R2,R1,4	R2<=R1 ? Fim : Next
0x0004	0x0000 3310	addi R3,R3,R1,0	R3=R3+R1 somatório
0x0005	0x0002 1100	addi R1,R1,R0,2	R1=R1+2 próximo ímpar
0x0006	0xFFFD 0005	beq R0,R0,-3	J Loop
0x0007 Fim:	0x0000 0335	beq R3,R3,0	J Fim

Figura 11. Código fonte do somador de ímpares

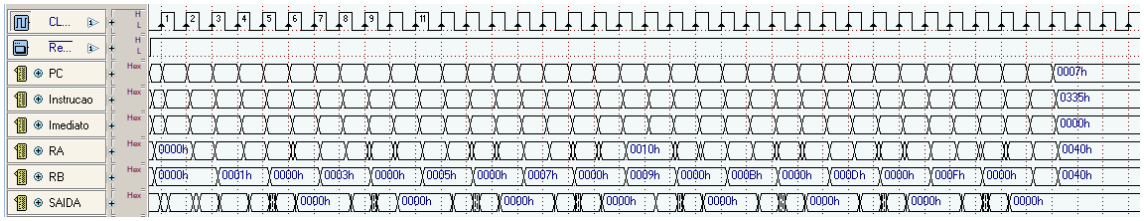

Figura 12. diagrama de onda da soma dos ímpares de 0 a 15

2.11. Assembler

Para facilitar o desenvolimento dos programas e a geração do código binário (Nesse caso hexadecimal) para as ROMs, foi desenvolvido um assembler em python. Ele recebe o códido em assembly pela entrada padrão ou lendo de um arquivo e gera dois arquivos de saída: lsh.drs e msh.drs, que podem ser lidos pelas ROMs.
Segue o código fonte do assembler:

```
opCode = {"addi":"0", "subi":"1", "andi":"2", "oi":"3", "xoi":"4",
"beq":"5", "bleu":"6", "bles":"7"}
padrao = "#R ROM4Kx16, id 0001\n#- Deeds Rom Source
Format (*.drs)\n\n#A 0000h\n#H\n"
lsh = padrao
msh = padrao
while True:
```

```
inst = input()
inst = inst.lower()
if inst == "s":
break
inst = inst.replace("r", "").replace(","," ")
inst = inst.split()
op = opCode[inst[0]]
if int(op) >= 5: #branch
rd = 0
ra = int(inst[1])
rb = int(inst[2])
imm = int(inst[3])
if imm < 0:
imm = (-imm ^ 0xffff)+0x1
lsh +=  hex(rd)[2:] + hex(ra)[2:] + hex(rb)[2:] + op + " "
msh += hex(imm)[2:].zfill(4) + " "
else:
rd = int(inst[1])
ra = int(inst[2])
rb = int(inst[3])
imm = int(inst[4])
if imm < 0:
imm = (-imm ^ 0xffff)+0x1
lsh +=  hex(rd)[2:] + hex(ra)[2:] + hex(rb)[2:] + op + " "
msh += hex(imm)[2:].zfill(4) + " "
f = open("lsh.drs", "w")
f.write(lsh)
f.close()

f = open("msh.drs", "w")
f.write(msh)
f.close()
```

2.12. MULTIPLICAÇÃO DE DOIS NÚMEROS SEM SINAL

Após isso, foram elaborados programas próprios, sendo esse um multiplicador entre dois números sem o uso de sinal. Foi feito um diagrama de onda para melhor analisar o funcionamento do citcuito, nesse caso, foi feita uma multiplicação entre 3 e 4, resultando em 12, ou C em hexadecimal.
Foi usado o seguinte código fonte em assembly:

```
addi r1 r0 r0 0 # count = 0
addi r2 r0 r0 3 # x = 3
addi r3 r0 r0 4 # y = 4
addi r1 r1 r2 0 # count += x
subi r3 r3 r0 1 # y -= 1
beq r0 r3 2 # branch if y == 0
beq r0 r0 -3 # recomeça o loop
beq r1, r1, 0 #show result
```

Diagrama de Ondas:

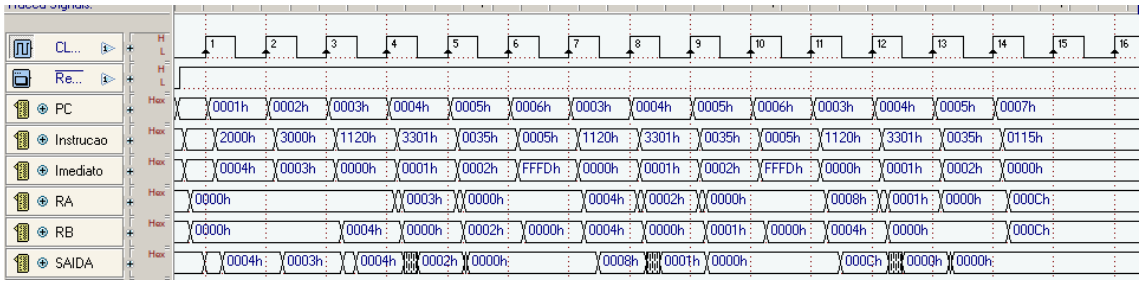

Figura 13. diagrama de onda da multiplicação sem sinal

Por motivos de simplicidade e melhor entendimento dos diagramas de ondas, em todo o relatório, foram usados valores de entrada pequenos para os programas. No video de teste, porém, foram utilizados os valores máximos indicados no roteiro.

2.13. MULTIPLICAÇÃO DE DOIS NÚMEROS COM SINAL

Posteriormente, houve também a elaboração de um multiplicador considerando os sinais da operação. Os números escolhidos foram 8 e -2 (FFFE em hexadecimal), resultando em -16 (FFF0 em hexadecimal)
Código Fonte:

```
addi r1 r0 r0 0 # count = 0
addi r2 r0 r0 8 # x = 8
addi r3 r0 r0 -2 # y = -2
bles r0 r3 3 # branch if y>0
subi r2 r0 r2 0 # x = -x
subi r3 r0 r3 0 # y = -y
addi r3 r3 r0 -1 # y -= 1 (Compensar pelo <=)
addi r1 r1 r2 0 # count += x
addi r3 r3 r0 -1 # y -= 1
bles r0 r3 -2 # branch if y > 0
beq r1, r1, 0 #show result
```

Diagrama de Ondas:

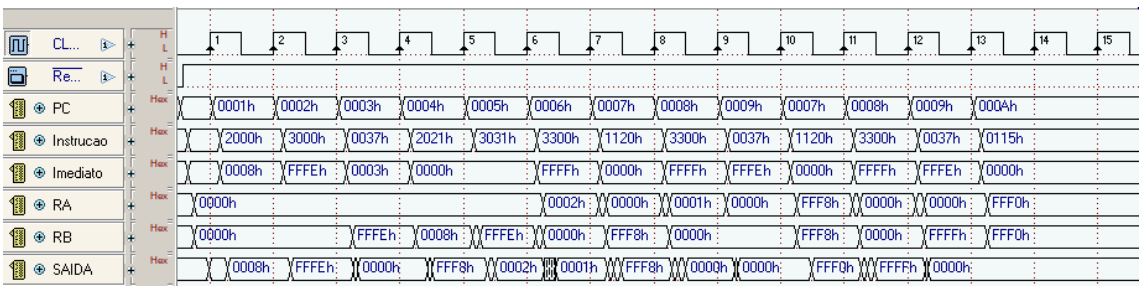

Figura 14. diagrama de onda da multiplicação com sinal

2.14. DIVISÃO INTEIRA DE DOIS NÚMEROS SEM SINAL

O próximo programa trata-se de um divisor inteiro de dois números desconsiderando o sinal. Os números escolhidos para o diagrama foram 6 e 2, cuja divisão resultou em 3.
Código Fonte:

```
addi r1 r0 r0 0 # saida = 0
addi r2 r0 r0 6 # R2 x = 6
addi r3 r0 r0 2 #R3 y = 2
subi r3 r3 r0 1   # subtrai um para checar
bleu r2 r3 5      # Checagem se dividendo é menor que diivsor
addi r3 r3 r0 1   # adiciona um para voltar ao normal
subi r2 r2 r3 0 # r2-r3,um pedaço de divisão
addi r1 r1 r0 1 # saida += 1
bleu r3 r2 -2 # volta
beq  r1, r1, 0 #show result
```

Diagrama de Ondas:

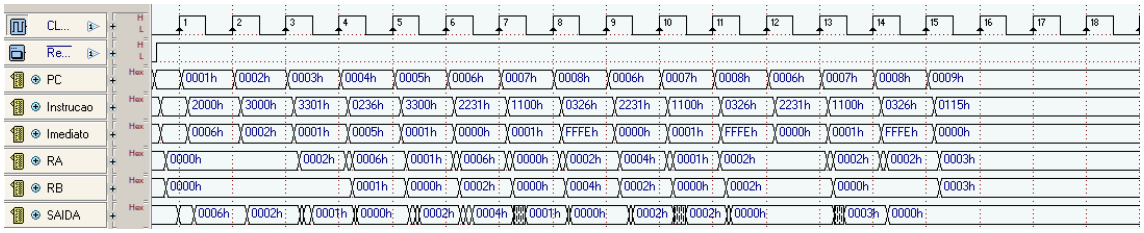

Figura 15. diagrama de onda divisão inteira sem sinal

2.15. RESTO DA DIVISÃO INTEIRA DE DOIS NÚMEROS INTEIROS SEM SINAL

Logo após, também foi projetado um programa que calcula o resto da divisão inteira desconsiderando o sinal. Os números escolhidos foram 7 e 3, cujo resto da divisão inteira é 1.
Código Fonte:

```
addi r1 r0 r0 0
addi r2 r0 r0 7  # R2 x = 7
addi r3 r0 r0 3  # R3 y = 3
subi r3 r3 r0 1  #subtrai 1 para checar
bleu r2 r3 4     #checagem se dividendo é menor que divisor
addi r3 r3 r0 1  #adiciona 1 para checar
subi r2 r2 r3 0 # r2-r3,um pedaço de divisão
bleu r3 r2 -1 # volta
addi r1 r2 r0 0
beq  r1, r1, 0 #show result
```

Diagrama de Ondas:

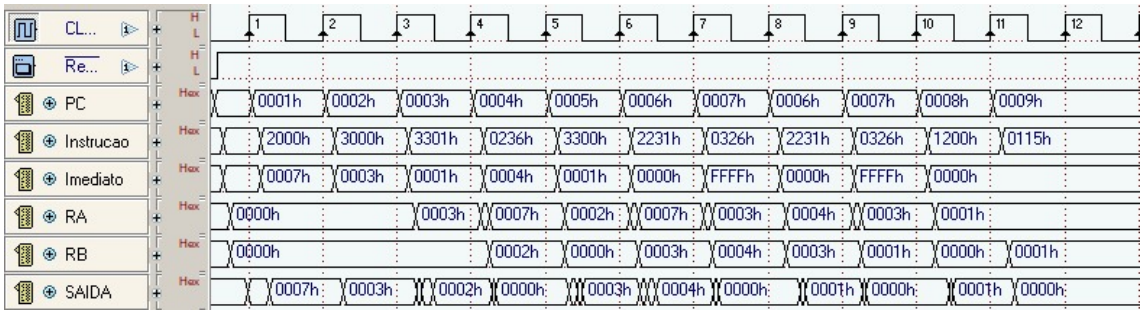

Figura 16. diagrama de onda resto da divisão inteira sem sinal

2.16. PROGRAMA DE TESTES FORNECIDO

Por fim, foi executado o programa de testes fornecido, verificando o total funcionamento das diversas funcionalidades do circuito.
Código fonte:

```
# Programa Teste zepto-II
#End MSH  LSH
0000 0002 1000 addi R1,R0,R0,0x0002 # R1=2
0001 FFFE 2000 addi R2,R0,R0,0xFFFE # R2=-2
0002 0000 3120 addi R3,R1,R2,0     # R3=R1+R2
0003 0002 0305 beq  R3,R0,OK1 # R3==0? OK1
0004 001D 0005 beq  R0,R0,ERRO # j ERRO
0005 0000 4202 OK1: andi R4,R2,R0,0 # R4=R2&0x0000
0006 0002 0405  beq  R4,R0,OK2 # R4==0? OK2
0007 001A 0005  beq  R0,R0,ERRO # j ERRO
0008 FFFF 5003 OK2: ori  R5,R0,R0,0xFFFF # R5=0xFFFF
0009 0000 6503  ori  R6,R5,R0,0 # R6=R5|0x0000
000A 0002 0655  beq  R6,R5,OK3 # R6=R5?OK3
000B 0016 0005   beq  R0,R0,ERRO # j ERRO
000C 5555 7000 OK3: addi R7,R0,R0,0x5555 # R7=0x5555
000D FFFF 8704  xori R8,R7,R0,0xFFFF # R8=R7 xor 0xFFFF
000E 0002 0877  bles R8,R7,OK4 # Sig R8<=R7? OK4
000F 0012 0005  beq  R0,R0,ERRO # j ERRO
0010 8000 9000 OK4: addi R9,R0,R0,0x8000 # R9=0x80000
0011 7FFF A000  addi R10,R0,R0,0x7FFF # R10=0x7FFFF
0012 0002 09A7  bles R9,R10,OK5 # Sig R9<=R10? OK5
0013 000E 0005  beq R0,R0,ERRO # j ERRO
0014 0002 0A96 OK5: bleu R10,R9,OK6 # unsig R10<=R9? OK6
0015 000C 0005  beq R0,R0,ERRO # j ERRO
0016 0080 B000 OK6: addi R11,R0,R0,128 # R11=128
0017 0000 C0B1  subi R12,R0,R11,0 # R12=R0-R11
0018 0000 DBC0  addi R13,R11,R12,0 # R13=R11+R12
0019 0002 0C05  beq  R12,R0,OK7 # R12==0? OK7
001A 0007 0005  beq R0,R0,ERRO # j ERRO
001B 7FFE D000 OK7: addi R13,R0,R0,0x7FFFE     # R13=0x7FFFE
001C 0004 ED00  addi R14,R13,R0,4 # R14=R13+4    Overflow
```

```
001D 0002 0E07  bles R14,R0,OK8 # sig R14<=0? OK8
001E 0003 0005  beq R0,R0,ERRO # j ERRO
001F CCCC F000 OK8: addi R15,R0,R0,0xCCCC # R15=0xCCCC
0020 0002 0005  beq R0,R0,FIM # j FIM
0021 EEEE F000 ERRO: addi R15,R0,R0,0xEEEE # R15=0xEEEE
0022 0000 0FF5 FIM: beq R15,R15,FIM # j FIM
```

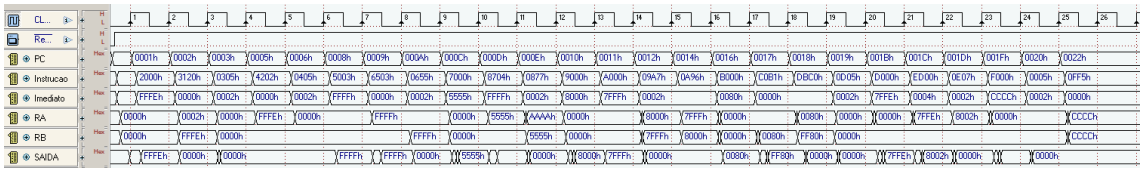

Figura 17. diagrama de onda

3. Análise dos Resultados

Foi possível observar que a criação de um comparador de numeros com sinal adicionando apenas algumas portas lógicas e um multiplexador a um comparador sem sinal. Por meio da checagem do último bit, verificando a sinalização das entradas e alterando a combinação de saídas para manter a comparação correta, sendo controlada por um sinal externo, emitido pelo bloco de controle, o qual decide o processo de comparação.
Além disso, observou-se que a frequência de funcionamento, menor que 5 MHz, é muito baixa, principalmente se comparada aos processadores modernos que funcionam na ordem dos Ghz. Essa frequência é limitada pelo tempo gasto para todos os circuitos se estabilizarem após o clock e, consequentemente, a alteração da instrução e do imediato fornecidos pela ROM.
Assim, foi visto que Nesse processador os endereços incrementam de 1 em 1, com 2 bytes por endereço em cada ROM, diferente do padrão usado por exemplo em Risc-V, que incrementa de 4 em 4, 1 byte por endereço e uma word a cada 4 endereços. O circuito de manipulação do Program Counter utiliza dois somadores, que operam no valor atual de endereço, um avançando normalmente para o próximo endereço e outro somando o valor do imediato, esses valores calculados depois são escolhidos por um multiplexador operado a partir dos sinais emitidos pelo bloco de controle e comparadores, definindo se há uma instrução de branch e se a comparação da instrução está correta, permitindo dessa forma um salto de endereçamento.
Devido ao tamanho dos registradores, o processador é capaz de:
Somar números de 15 bits, sem sina ($n < 32768$) e com sinal ($-16384 < n < 16384$)
Multiplicar números de 8 bits, sem sinal ($n < 256$) e com sinal ($-128 < n < 128$)
Dividir e achar o resto entre números de 16 bits, sem sinal ($n < 65536$)
Ademais, com a construção de programas, foi notado que é possível elaborar multiplicações com somas sucessivas e divisões com subtrações sucessivas.

4. Conclusão

O desenvolvimento do zeptoprocessador de 16 bits e 8 instruções foi um sucesso e seu funcionamento foi de acordo com o planejado e descrito no roteiro. Denota-se importância da integração entre as diversas partes do processador e o respeito de sua frequência máxima

para que o circuito se estabilize e funcione corretamente. Além do respeito aos valores limites das operações aritméticas, para que não ocorra overflows. É um processador simples, porém poderoso, sendo capaz de solucionar diversos problemas computacionais. Foi possível fixar e aplicar os conhecimentos adquiridos ao longo das disciplinas de Circuitos Lógicos e Laboratório de Circuitos Lógicos.

Referências

[1] Marcus Vinicius Lamar. *Roteiro*. URL: `https://aprender3.unb.br/mod/resource/view.php?id=208040`.

[2] Ronald J Tocci. *Sistemas Digitais Princípios e Aplicações*. 1977.