Centro Federal de Educação Tecnológica - CEFET-MG

Laboratório de Arquitetura e Organização de Computadores

Project Base

Thiago Figueiredo Costa

20 de abril de 2017

1 Introdução

O documento a seguir descreve de forma resumida o projeto do processador nanoRisk, proposto durante as aulas de laboratório de arquitetura de computadors. O processador deve ser limitado à 8 bits, e inicialmente não deve possuir sinais de entrada ou saída e possuir programa unico gravado na memória. O programa do processador deve resolver um problema que for escolhido.

2 Programa proposto

O programa que o processador nRisk irá executar é um programa que soluciona o problema abaixo:

2.1 Problema

Um geólogo retirou uma amostra de um local onde possivelmente houve um grande deslizamento de terra, porém ele ainda tinha dúvidas quanto a isso. Ao analisar a granulometria percebeu que o material particulado não possui grande distinção, uma vez que nele possuía material pelitico, havia ainda grandes conglomerados, indicando vir de condições hipopicnais. Ainda sem certeza, resolveu levar para o laboratório de microscopia, e ao perceber que na amostragem havia grandes presenças de Plagioclásio e Olivina , que são minerais detríticos primários instáveis, ele concluiu que estes só poderiam estar presentes ali se estivessem muito perto da área fonte e ainda percebeu que o evento era recente, uma vez que estes minerais não sofreram retrabalhamento e se alteraram para minerais secundários.

Considerando a massa molar do cálcio, alumínio, silício, oxigênio como respectivamente 40, 26 ,28, 16 e $\hat{v}$ = {300,150,666,357,220,480,276,666}

a)Para fazer o cálculo daquela reserva, este geólogo precisa de calcular a massa molar do plagioclásio, sabendo a fórmula química no mineral é $CaAl_2Si_2O_8$, qual a massa molar do plagioclásio?

b)Uma estagiária distraida do 7° periodo de geologia misturou as amostras trituradas, tendo apenas o vetor $\hat{v}$ com a massa molar de cada uma ajude o géologo a achar a amostra de plagioclásio indicando a posição dela no vetor.

2.2 Solução

Multiplicando o índice do átomo pela sua massa molar e somando com o resultado dos outros átomos se obtém a massa molar da molécula como abaixo:

40+2x26+2x28+8x16=276

E pesquisando um a um no vetor $\hat{v}$ achamos esse valor na 7^a posição do vetor.

2.3 Codigo em MIPS

Para analisar melhor o programa ele foi implementado usando a arquitetura mips, foram usadas as funções la, lw, jal, addi, add, slt, beq, sub, j, li e jr. E foram usados os registradores s0, s1, t0, t1, t2, t3, t4, a0, a1, v0, ra.

```
.data
v:      .word 300,150,666,357,220,480,276,666    #v array size 8
nCa:    .word 1
nAl:    .word 2
nSi:    .word 2
nO:     .word 8
mCa:    .word 40
mAl:    .word 26
mSi:    .word 28
mO:     .word 16
```

```
.text
#main func
main:
la $t0, nCa
la $t1, mCa
lw $a0, 0($t0)
lw $a1, 0($t1)
jal mult
addi $s0,$v0,0
la $t0, nAl
la $t1, mAl
lw $a0, 0($t0)
lw $a1, 0($t1)
jal mult
add $s0,$v0,$s0
la $t0, nSi
la $t1, mSi
lw $a0, 0($t0)
lw $a1, 0($t1)
jal mult
add $s0,$v0,$s0
la $t0, nO
la $t1, mO
lw $a0, 0($t0)
lw $a1, 0($t1)
jal mult
add $s0,$v0,$s0          #massa molar de CaAl2Si2O8
li $t2, 8                #vector size
li $t1, 0
la $t3, v
mainloop:
slt $t0, $t1, $t2        #if(t1<t2) t0=true else t0=false
beq $t0, $zero, mainend
add $t4, $t1, $t1        #t1*2
add $t4, $t4, $t4        #t4*2
add $t4, $t3, $t4        #t4+base
lw $t4, 0($t4)           #vector at i value
sub $t4, $t4, $s0
addi $t1, $t1, 1
addi $s1, $t1,0          #posicao do plagioclasio(primeira posicao e 1)
beq $zero,$t4,mainend
j mainloop
mainend:
li   $v0, 10             # system call for exit
syscall                  # we are out of here.

#mult func(a*b=c)
mult:
slt $t0, $a0, $a1        #if(a0<a1) t0=true else t0=false
beq $t0, $zero, multp2
```

```
addi $t0, $a0, 0          #a0 lower
addi $t1, $a1,0
j multp3
multp2:                   #a1 lower
addi $t0, $a1, 0
addi $t1, $a0,0
multp3:                   #t0<t1
li $t2, 0
li $v0, 0
multloop:
slt $t3, $t2, $t0         #if(t2<t0) t3=true else t3=false
beq $t3, $zero, multend
add $v0, $v0, $t1
addi $t2, $t2, 1
j multloop
multend:
jr $ra
```

3 Registradores

Para pensar nas intruções e no formato delas foi necessário fazer um esboço dos registradores nos quais as funções iriam trabalhar e na quantidade deles. Por isso foram pensados 16 registradores endereçados e 1 não endereçado que podem ser representados com 4 bits.

- Zero
  Valor constante zero, representação $zero, $0. ID=0.

- Flag
  Registrador que armazena as flags e interrupts, como é 8 bits, pode armazenar até 8 flags, representação $flag, $flg. ID=1;

Tabela 1: Flags do nanoRisk

Bit(0 to 7)	Nome	Descrição
0	Overflow+	Quando uma operação com numero tem overflow positivo esse bit é definido como 1
1	Overflow-	Quando uma operação com numero tem overflow negativo esse bit é definido como 1
2	Unknow	-
3	Unknow	-
4	Unknow	-
5	Read	Flag usada para leitura de dados(e.g. switch)
6	Write	Flag usada para escrita de dados(e.g. leds)
7	Custom	O programador pode usar esse bit para definir uma flag customizada
0:7	Wake	Sai da instrução sleep se diferente de zero

- nano Temporary
  Registrador temporário usado para instruções interpretadas e calculos temporários, pode ser usado pelo programador sob o risco de ser sobrescrito caso seja chamada uma instrução interpretada ou função que usa ele, representação $nt, $nT. ID=2.

- Stack Pointer
  Registrador com o endereço da pilha, representação $sp. ID=3.

- Return address
  Registrador com o endereço de retorno, representação $ra. ID=4.

- Return value(x2)
  Registradores de retorno que podem ser usados também como output, representação $v0,$v1. ID=5,6.

- Arguments(x4)
  Registradores de argumento que podem ser usados também como input, representação $a0,$a1,$a2,$a3. ID=7,8,9,10.

- Saved Value(x4)
  Registradores com valores que podem ser usador pelo programador, representação $s0,$s1,$s2,$s3. ID=11,12,13,14.

- Base Pointer
  Registrador com o endereço base, representação $bp. ID=15.

- Next
  Registrador com o proximo endereço de execução do programa, representação $nxt. ID=Não endereçado.

4 Conjunto de Instruções

Foram definidas 32 instruções, porém 16 delas são interpretadas, então o processador executa 16 instruções que podem ser representadas com 4 bits.

4.1 Instruções

4.1.1 Controle de fluxo

- Sleep
  Instrução que mantem o registrador que aponta para a proxima instrução($nxt) nessa linha, ou seja, ele não faz nada. O nRisk deve também bloquear leitura e escrita na memória. Isso continuará acontecendo até que haja uma interrupção. A sintaxe assembly é "slp $r", onde $r é o registrador responsável pela interrupção do evento, $r será definido como $zero. Pode ser escrito como "slp", assim $r será o registrador padrão para flags e interrupções($flg). Se $r for zero ele continua o programa. Pode ser usado para finalizar o programa.

- Branch register on equal
  Vai para uma parte do programa se os registradores forem iguais. A sintaxe assembly é "brq $r1, $r2, $addr", if($r1==$r2) goto $addr.

- Branch register on flag
  Se uma flag estiver ativa no registrador usalmente de flag $flg ele desativa a flag e vai para a posição. A sintaxe assembly é "brf $flg, $r1, $addr", if($flg&$r1>0) $flg-$r1 → goto $addr.

- Branch on equal
  Vai para uma parte do programa se os registradores forem iguais. A sintaxe assembly é "beq $r1, $r2, addr", if($r1==$r2) goto addr. Precisa ser interpretada como "lc $nt, addr"→ "brq $r1, $r2, $nt"

- Branch on flag
  Se uma flag estiver ativa no registrador usalmente de flag $flg ele desativa a flag e vai para a posição. A sintaxe assembly é "bof $flg, $r1, addr", if($flg&$r1>0) $flg-$r1→goto addr. Precisa ser interpretada como "lc $nt, addr"→ "brf $r1, $r2, $nt"

- Jump register
  Vai para uma parte do programa que está em um registrador e grava no registrador $nt a proxima instrução antes do desvio. A sintaxe assembly é "jr $r", goto $r.

- Jump
  Vai para uma parte do programa. A sintaxe assembly é "j addr", goto addr. Pode ser interpretado como "lc $nt, addr"→ "jr $nt".

- Jump and link
  Vai para uma parte do programa e define o registrador de retorno ($ra) como a proxima instrução antes do desvio. A sintaxe assembly é "jal addr", $ra=$nxt → goto addr. Pode ser interpretado como "lc $nt, addr"→ "jr $nt"→ "add $ra, $nt, $0"

- Jump register and link
  Vai para uma parte do programa que está em um registrador e define o registrador de retorno ($ra) como a proxima instrução antes do desvio. A sintaxe assembly é "jrl $r", $ra=$nxt → goto $r. Pode ser interpretado como "jr $r"→ "add $ra, $nt, $0"

4.1.2 Transferência de dados

- Load Address
  Carrega o enderereço da memória RAM de uma variável declarada no programa. A sintaxe assembly é "la $r, var", o endereço de var é gravado em $r.

- Load Constant
  Carrega no registrador um valor pré-definido. A sintaxe assembly é "lc $r, value", value é gravado em $r.

- Load Word
  Carrega o conteudo da memória RAM que está em um determinado endereço. A sintaxe assembly é "lw $r, offset($a)", o valor do endereço de $a+offset é gravado em $r. Ou "lw $r, $a"nesse caso offset é tido como zero.

- Store Word
  Salva um conteudo de um registrador em um endereço da memória RAM. A sintaxe assembly é "sw $r, offset($a)", o valor de $r é gravado no endereço $a+offset. Ou "sw $r, $a"nesse caso offset é tido como zero.

- Move
  Move um valor de um registrador para outro. A sintaxe assembly é "mov $d, $r", $d:=$r ao invés de fazer um circuito para essa instrução, ela pode ser interpretada pelo montador como "add $d, $r, $zero".

- Push
  Adiciona um valor contido em um registrador na pilha padrão($sp). A sintaxe assembly é "push $r", essa instrução é interpretada como "lc $nt, ∓ 1 "→"add $sp, $sp, $nt"→"sw $r, 0($sp)". (Repare o simbolo $\mp$, ele foi usado pois ainda nao foi decidido se a pilha funcionará com incremento ou decremento)

- Pop
  Remove um valor na pilha padrão($sp) e salva ele em um registrador. A sintaxe assembly é "pop $r", essa instrução é interpretada como "lw $r, 0($sp)"→"lc $nt, ± 1"→"add $sp, $sp, $nt". (Repare o simbolo $\pm$, ele foi usado pois ainda nao foi decidido se a pilha funcionará com incremento ou decremento)

4.1.3 Aritméticas

- Add
  Soma o valor de dois registradores e salva em outro. Provavelmente zerar

as flags de overflow antes de somar para evitar lixo. A sintaxe assembly é ”add $d, $r1, $r2”, $d:=$r1+$r2

- Subtract
  Subtrai o valor de dois registradores e salva em outro. Provavelmente zerar as flags de overflow antes de somar para evitar lixo. A sintaxe assembly é ”sub $d, $r1, $r2”, $d:=$r1-$r2

- Add Constant
  Soma o valor de um registrador e uma constante e salva em um registrador. A sintaxe assembly é ”addc $d, $r, value”, $d:=$r+value, pode ser interpretada ”lc $nt, value”→ ”add $d, $r, $nt”

4.1.4 Lógicas

- And
  Faz a operação lógica and(&) entre dois registradores e salva em um terceiro o resultado. A sintaxe assembly é ”and $d, $r1, $r2”, $d:=$r1&$r2.

- Or
  Faz a operação lógica or(|) entre dois registradores e salva em um terceiro o resultado. A sintaxe assembly é ”or $d, $r1, $r2”, $d:=$r1|$r2.

- Nor
  Faz a operação lógica nor(~(|)) entre registradores e salva em outro o resultado. A sintaxe assembly é ”nor $d, $r1, $r2”, $d:= ~($r1| $r2).

- Not
  Faz a operação lógica not(~)em um registrador e salva em outro o resultado. A sintaxe assembly é ”not $d, $r1”, $d:=~$r1, pode ser interpretada como ”nor $d, $r1, $zero”.

- Set on less than
  Define o registrador de saída como 0xFF se o valor primeiro registrador for menor que o segundo, caso contrário a saída é 0x00. A sintaxe assembly é ”slt $d, $r1, $r2”, if($r1<$r2) $d:=1 else $d:=0.

- And constant
  Faz a operação lógica and(&) entre um registrador e uma constante e salva em outro registrador resultado. A sintaxe assembly é ”andc $d, $r, value”, $d:=$r&value. Pode ser interpretado como ”lc $nt, value”→ ”and $d, $r, $nt”.

- Or constant
  Faz a operação lógica or(|) entre um registrador e uma constante e salva em outro registrador resultado. A sintaxe assembly é ”orc $d, $r, value”, $d:=$r|value. Pode ser interpretado como ”lc $nt, value”→ ”or $d, $r, $nt”.

- Nor constant
  Faz a operação lógica nor(~(|)) entre um registrador e uma constante e salva em outro registrador resultado. A sintaxe assembly é ”norc $d, $r, value”, $d:= ~($r|value). Pode ser interpretado como ”lc $nt, value”→ ”nor $d, $r, $nt”.

- Set on less than constant
  Define o registrador de saída como 1 se o valor primeiro registrador for menor que uma constante, caso contrário a saída é 0. A sintaxe assembly é "sltc $d, $r, value", if($r<value) $d:=1 else $d:=0. Pode ser interpretado como "lc $nt, value"→ "slt $d, $r, $nt".

4.1.5 Deslocamento de bit

- Shift right
  Desloca para a direita os bits de um registrador n vezes, onde n é o valor de outro registrador e salva o valor em um terceiro. A sintaxe assembly é "sr $d, $r1, $r2", $d:=$r1>>$r2.

- Shift left
  Desloca para a esquerda os bits de um registrador n vezes, onde n é o valor de outro registrador e salva o valor em um terceiro. A sintaxe assembly é "sl $d, $r1, $r2", $d:=$r1<<$r2.

- Shift right constant
  Desloca para a direita os bits de um registrador n vezes, onde n é o valor de uma constante e salva o valor em outro registrador. A sintaxe assembly é "src $d, $r, value", $d:=$r>>value. Pode ser interpretado como "lc $nt, value"→ "sl $d, $r, $nt".

- Shift left constant
  Desloca para a esquerda os bits de um registrador n vezes, onde n é o valor de uma constante e salva o valor em outro registrador. A sintaxe assembly é "slc $d, $r, value", $d:=$r<<value. Pode ser interpretado como "lc $nt, value"→ "sr $d, $r, $nt".

4.1.6 ID de instrução

Os códigos em binário de cada instrução que foi definido.

- nRisk_slp="0000"0

- nRisk_brq="0001"1

- nRisk_brf="0010"2

- nRisk_add="0011"3

- nRisk_sub="0100"4

- nRisk_and="0101"5

- nRisk_or =”0110”6
- nRisk_nor=”0111”7
- nRisk_slt=”1000”8
- nRisk_sr =”1001”9
- nRisk_sl =”1010”10
- nRisk_jr =”1011”11
- nRisk_la =”1100”12
- nRisk_lc =”1101”13
- nRisk_lw =”1110”14
- nRisk_sw =”1111”15

4.2 Formato

Considerando que a memória tem 256 posições podemos endereçá la com 8 bits, isso pode ser ruim, com uma memoria tão pequena, talvez será necessário fazer duas memórias de 256 bytes(ja que a memoria armazena 8 bits), uma memória para instruções e outra para dados. Os registradores precisam de 4 bits para endereço e as instruções também. Então será necessário dois bytes por instrução. E os formatos são como os abaixo. Os registradores ficam na ordem da sintaxe assembly.

4.2.1 Formato Single

Tabela 2: Formato S

Bits	Nome	Tamanho	Descrição
		1◦ Byte	
7:4	Instrução	4b	Código da instrução
3:0	$r1	4b	Endereço do registrador

Funções que usam esse formato(x2): sleep, jump register.

4.2.2 Formato Jump

Tabela 3: Formato J

Bits	Nome	Tamanho	Descrição
		1◦ Byte	
7:4	Instrução	4b	Código da instrução
3:0	$r1	4b	Endereço do registrador
		2◦ Byte	
7:0	Constante	8b	Se for diferente de zero contém o endereço para ir, campo usado para evitar a interpretação da função jump

Funções que usam esse formato: jump, jump register. É possível ver que jump register pode ter dois formatos, isso ocorre por que na verdade, o formato jump foi criado para aproveitar o byte extra que fica vazio no formato single, com o baixo custo de não poder usar jump para o endereço 0, problema. que pode ser resolvida pelo montador substituindo esse caso por com "jr $zero". Outra alternativa para evitar esse desperdicio é fazer com que só se busque o segundo byte de instrução caso seja uma operação diferente, mas isso pode complicar mais o hardware.

4.2.3 Formato Three Registers

Tabela 4: Formato 3R

Bits	Nome	Tamanho	Descrição
		1◦ Byte	
7:4	Instrução	4b	Código da instrução
3:0	$r1	4b	Endereço do registrador 1
		2◦ Byte	
7:4	$r2	4b	Endereço do registrador 2
3:0	$r3	4b	Endereço do registrador 3

Funções que usam esse formato(x10): branch register on equal, branch register on flag, add, subtract, and, or, nor, set on less than, shift right, shift left.

4.2.4 Formato Memory

Tabela 5: Formato M

Bits	Nome	Tamanho	Descrição
		1◦ Byte	
7:4	Instrução	4b	Código da instrução
3:0	$r1	4b	Endereço do registrador 1
		2◦ Byte	
7:4	$r2	4b	Endereço do registrador 2
3:0	Offset	4b	Valor a ser somado em $r2

Funções que usam esse formato(x2): load word, store word.

4.2.5 Formato Constant

Tabela 6: Formato C

Bits	Nome	Tamanho	Descrição
		1◦ Byte	
7:4	Instrução	4b	Código da instrução
3:0	$r	4b	Endereço do registrador
		2◦ Byte	
7:0	Constante	8b	Endereço ou constante

Funções que usam esse formato(x2): load constant, load address.

5 Código nanoRisk

Para se ter ideia de como seria o código após propor a arquitetura e para começar a pensar em como tratar os problemas como overflow, foi feito um esboço do código do programa em nanoRisk.

```
.data
#v:        .word 300,  150,  666,  357,  220,  480,  276,
666          #OVERFLOW
#v array size 16 with 8 elements
#positive overflow(simple)(signed)
#v:        .word 2,44, 1,22,  5,26,  2,101, 1,92,  3,96,  2,20, 5,26
#nagative overflow(2 overflow flags ou overflow e menor que zero)(unsigned)
v:         .word 1,44, 0,150, 2,154, 1,101, 0,220, 1,224, 1,20, 2,154
nCa:       .word 1
nCa:       .word 1
nAl:       .word 2
nSi:       .word 2
nO:        .word 8
mCa:       .word 40
mAl:       .word 26
mSi:       .word 28
mO:        .word 16
.text
#main func
main:
la $a0, nCa
la $a1, mCa
lw $a0, 0($a0)
lw $a1, 0($a1)
jal mult
mov $s0,$v0
mov $s1,$v1
la $a0, nAl
la $a1, mAl
lw $a0, 0($a0)
lw $a1, 0($a1)
jal mult
add $s0,$v0,$s0
add $s1,$v1,$s1
la $a0, nSi
la $a1, mSi
lw $a0, 0($a0)
lw $a1, 0($a1)
jal mult
add $s0,$v0,$s0
add $s1,$v1,$s1
la $a0, nO
la $a1, mO
lw $a0, 0($a0)
```

```
lw $a1, 0($a1)
jal mult
add $s1,$v1,$s1          #+sig massa molar de CaAl2Si2O8
add $s0,$v0,$s0          #-sig massa molar de CaAl2Si2O8
lc $s2, 0
la $s3, v
mainloop:
sltc $nt, $s2, 8         #if(s2<8) nt=true else nt=false
beq $nt, $0, mainend     #usado nt pois e temporario
push $s2
push $s3
lw $s2, 0($s3)  #vector at i value
lw $s3, 1($s3)  #vector at i value
sub $s2,$s1,$s2          #mais significativo
sub $s3,$s0,$s3          #menos significativo
add $nt,$s2,$s3
pop $s3
pop $s2
beq $0,$nt,mainend #igual
addc $s2,$s2,1
addc $s3,$s3,2  #nao ha alinhamento de memoria
j mainloop
mainend:
addc $s2, $s2, 1                 #posicao do plagioclasio(primeira posicao e 1)
slp                      # we are out of here.

#mult func(a*b=c)
mult:
push $s0
push $s1
slt $s0, $a0, $a1        #if(a0<a1) t0=true else t0=false
beq $s0, $0, multp2
mov $s0, $a0 #a0 lower
mov $s1, $a1
j multp3
multp2:                  #a1 lower
mov $s0, $a1 #a0 lower
mov $s1, $a0
multp3:                  #always s0<s1
lc $v0, 0
lc $v1, 0
multloop:
sltc $nt, $s0, 0 #if(s0<0) nt=true else nt=false
not $nt, $nt
beq $nt, $zero, multend
add $v0, $v0, $s1
lc $a0, 2        #overflow negative
addc $s0, $s0, -1
bof $flg, $a0, multoverflow
j multloop
```

```
multend:
pop $s1
pop $s0
jr $ra
multoverflow:
addc $v1, $v1, 1
j multloop
```

6 Codigo binário em nanoRisk

O código foi gerado através do compilador, o codigo pode ser visto com quebras de linha abrindo o arquivo "compiled"em um editor de texto.

00010000 00011100 00011010 00101000 00001000 00000010 00000010 00000001
00000001 00000000 00000000 00000000 00000000 00000000 00000000 00000000
00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
00000001 00011001 11000111 00010001 11001000 00010101 11100111 00000111
11101000 00001000 11010010 10001010 10110010 00000000 00110100 00100000
00111011 01010000 00111100 01100000 11000111 00010010 11001000 00010110
11100111 00000111 11101000 00001000 11010010 10001010 10110010 00000000
00110100 00100000 00111011 01011011 00111100 01101100 11000111 00010011
11001000 00010111 11100111 00000111 11101000 00001000 11010010 10001010
10110010 00000000 00110100 00100000 00111011 01011011 00111100 01101100
11000111 00010100 11001000 00011000 11100111 00000111 11101000 00001000
11010010 10001010 10110010 00000000 00110100 00100000 00111100 01101100
00111011 01011011 11011101 00000000 11001110 00000000 11010010 00001000
10000010 11010010 11010010 10000100 00010010 00000010 11010010 00000001
01000011 00110010 11111101 00000011 11010010 00000001 01000011 00110010
11111110 00000011 11101101 00001110 11101110 00011110 01001101 11001101
01001110 10111110 00110010 11011110 11101110 00000011 11010010 00000001
00110011 00110010 11101101 00000011 11010010 00000001 00110011 00110010
11010010 10000100 00010000 00100010 11010010 00000001 00111101 11010010
11010010 00000010 00111110 11100010 10110000 01001100 11010010 00000001
00111101 11010010 00000001 00000000 11010010 00000001 01000011 00110010
11111011 00000011 11010010 00000001 01000011 00110010 11111100 00000011
10001011 01111000 11010010 10100010 00011011 00000010 00111011 01110000
00111100 10000000 10110000 10100110 00111011 10000000 00111100 01110000
11010101 00000000 11010110 00000000 11010010 00000000 10000010 10110010
01110010 00100000 11010010 11000010 00010010 00000010 00110101 01011100
11010111 00000010 11010010 00000001 00111011 10110010 11010010 11010000
00100001 01110010 10110000 10101010 11101100 00000011 11010010 00000001
00110011 00110010 11101011 00000011 11010010 00000001 00110011 00110010
10110100 00000000 11010010 00000001 00110110 01100010 10110000 10101010

7 Compilador (nrc)

O código do compilador desenvolvido para o nanoRisk se encontra abaixo: O código de maquina das variáveis(a partir de ".data") está sendo endereçado serapadamente o que pede uma memória separada para dados e instruções, e o segmento de dados está sendo incluso no código de máquina mas a forma como isso aconece pode mudar para que fique melhor no código/hardware. O segmento de dados está sendo incluso no código de maquina após 8 bits que dizem seu tamanho, deve ser copiado para a memória de dados enquanto o resto vai para a memória de instruções.

```
#include <iostream>
#include <bitset>
#include <algorithm>
#include <fstream>
#include <sstream>
#include <string>
#include <vector>
#include <map>

using namespace std;

string output = "a.out";
string input;
map<string,int> addresses;

int nRisk_StackSigal=-1;
//funcs
string nRisk_slp="0000";
string nRisk_brq="0001";
string nRisk_brf="0010";
string nRisk_add="0011";
string nRisk_sub="0100";
string nRisk_and="0101";
string nRisk_or ="0110";
string nRisk_nor="0111";
string nRisk_slt="1000";
string nRisk_sr ="1001";
string nRisk_sl ="1010";
string nRisk_jr ="1011";
string nRisk_la ="1100";
string nRisk_lc ="1101";
string nRisk_lw ="1110";
string nRisk_sw ="1111";
//regs
string nRisk_zro="0000";
string nRisk_flg="0001";
string nRisk_nt ="0010";
string nRisk_sp ="0011";
string nRisk_ra ="0100";
string nRisk_v0 ="0101";
string nRisk_v1 ="0110";
string nRisk_a0 ="0111";
string nRisk_a1 ="1000";
string nRisk_a2 ="1001";
string nRisk_a3 ="1010";
string nRisk_s0 ="1011";
string nRisk_s1 ="1100";
string nRisk_s2 ="1101";
string nRisk_s3 ="1110";
string nRisk_bp ="1111";
```

```
51
52 void error(string str){
53      cout<<"error - "<<str<<"."<<endl;
54 }
55
56 struct nanoArg{
57     string arg;
58     bool isRegister;
59     int pos;
60     bool err=false;
61 };
62
63 int strToInt(string str){
64     stringstream ss;
65     ss.str(str);
66     int out;
67     ss>>out;
68     return out;
69 }
70
71 bool isValidChar(char c){
72     return c=='$'||(c>=48&&c<=57)||(c>=65&&c<=90)||(c>=97&&c<=122);
73 }
74
75 bool isBinary(string str){
76     for(int i=0;i<str.size();i++){
77         if(str[i]!='0'&&str[i]!='1'){
78             return false;
79         }
80     }
81     return true;
82 }
83 nanoArg getArgFrom(string str, int from){
84     nanoArg out;
85     int start=-1;
86     for(int j=from;j<=str.size();j++){
87         if(start<0&&isValidChar(str[j])){
88             start=j;
89         }else if(start>=0){
90             if(str[j]==',||str[j]=='\0'||str[j]=='\t'||str[j]=='('
                    ||str[j]==')'||str[j]==' '){
91                 out.pos=j+1;
92                 out.arg=str.substr(start,j-start);
93                 out.isRegister=out.arg.find("$")!=string::npos;
94                 return out;
95             }
96         }
97     }
98     out.err=true;
99     return out;
100 }
101 string translateRegister(string arg){
102     arg.erase(remove(arg.begin(),arg.end(),'$'),arg.end());//remove
          $
103     if(!arg.compare("zero")||!arg.compare("0")){
104         return nRisk_zro;
105     }else if(!arg.compare("flg")||!arg.compare("flag")){
106         return nRisk_flg;
107     }else if(!arg.compare("nt")||!arg.compare("nT")){
108         return nRisk_nt;
109     }else if(!arg.compare("sp")){
110         return nRisk_sp;
```

```
111     }else if(!arg.compare("ra")){
112         return nRisk_ra;
113     }else if(!arg.compare("v0")){
114         return nRisk_v0;
115     }else if(!arg.compare("v1")){
116         return nRisk_v1;
117     }else if(!arg.compare("a0")){
118         return nRisk_a0;
119     }else if(!arg.compare("a1")){
120         return nRisk_a1;
121     }else if(!arg.compare("a2")){
122         return nRisk_a2;
123     }else if(!arg.compare("a3")){
124         return nRisk_a3;
125     }else if(!arg.compare("s0")){
126         return nRisk_s0;
127     }else if(!arg.compare("s1")){
128         return nRisk_s1;
129     }else if(!arg.compare("s2")){
130         return nRisk_s2;
131     }else if(!arg.compare("s3")){
132         return nRisk_s3;
133     }else if(!arg.compare("bp")){
134         return nRisk_bp;
135     } else return "";
136 }
137 vector<string> translateFunc(string func,string args){
138     func.erase(remove(func.begin(),func.end(),' '),func.end());//
            remove spaces
139     args.erase(remove(args.begin(),args.end(),' '),args.end());//
            remove spaces
140     func.erase(remove(func.begin(),func.end(),'\t'),func.end());//
            remove spaces
141     args.erase(remove(args.begin(),args.end(),'\t'),args.end());//
            remove spaces
142     vector<string> out;
143     nanoArg arg0,arg1,arg2;
144     string nRisk,argRegister,target0,target1,target2;
145     if(!func.compare("slp")||!func.compare("sleep")){
146         arg0=getArgFrom(args,0);
147         target0="0001";
148         if(!arg0.err){
149             if(arg0.isRegister){
150                 argRegister=translateRegister(arg0.arg);
151                 if(argRegister!="")
152                     target0=argRegister;
153                 else{
154                     error("Register \""+arg0.arg+"\" doesnt exists"
                            );
155                     return out;
156                 }
157             }
158         }
159         out.push_back(nRisk_slp+target0);
160         out.push_back("00000000");
161         return out;
162     }else if(!func.compare("brq")||!func.compare("brf")||!func.
            compare("add")||!func.compare("sub")||!func.compare("and")
163              ||!func.compare("or")||!func.compare("nor")||!func.
                 compare("slt")||!func.compare("sr")||!func.compare
                 ("sl")){
164         arg0=getArgFrom(args,0);
```

```
165         if(arg0.err){
166             error("Cannot find arg0 for \""+func+"\"");
167             return out;
168         }
169         if(!arg0.isRegister){
170             error("Arg0 for \""+func+"\" is not a register");
171             return out;
172         }
173         arg1=getArgFrom(args,arg0.pos);
174         if(arg1.err){
175             error("Cannot find arg1 for \""+func+"\"");
176             return out;
177         }
178         if(!arg1.isRegister){
179             error("Arg1 for \""+func+"\" is not a register");
180             return out;
181         }
182         arg2=getArgFrom(args,arg1.pos);
183         if(arg2.err){
184             error("Cannot find arg2 for \""+func+"\"");
185             return out;
186         }
187         if(!arg2.isRegister){
188             error("Arg2 for \""+func+"\" is not a register");
189             return out;
190         }
191         argRegister=translateRegister(arg0.arg);
192         if(argRegister==""){
193             error("Register \""+arg0.arg+"\" doesnt exists for \""+
                    func+"\"");
194             return out;
195         }
196         target0=argRegister;
197         argRegister=translateRegister(arg1.arg);
198         if(argRegister==""){
199             error("Register \""+arg1.arg+"\" doesnt exists for \""+
                    func+"\"");
200             return out;
201         }
202         target1=argRegister;
203         argRegister=translateRegister(arg2.arg);
204         if(argRegister==""){
205             error("Register \""+arg2.arg+"\" doesnt exists for \""+
                    func+"\"");
206             return out;
207         }
208         target2=argRegister;
209 
210         if(!func.compare("brq"))
211             nRisk=nRisk_brq;
212         else if(!func.compare("brf"))
213             nRisk=nRisk_brf;
214         else if(!func.compare("add"))
215             nRisk=nRisk_add;
216         else if(!func.compare("sub"))
217             nRisk=nRisk_sub;
218         else if(!func.compare("and"))
219             nRisk=nRisk_and;
220         else if(!func.compare("or"))
221             nRisk=nRisk_or;
222         else if(!func.compare("nor"))
223             nRisk=nRisk_nor;
```

```
224         else if(!func.compare("slt"))
225             nRisk=nRisk_slt;
226         else if(!func.compare("sr"))
227             nRisk=nRisk_sr;
228         else if(!func.compare("sl"))
229             nRisk=nRisk_sl;
230 
231         out.push_back(nRisk+target0);
232         out.push_back(target1+target2);
233         return out;
234     }else if(!func.compare("jr")){
235         arg0=getArgFrom(args,0);
236         if(arg0.err){
237             error("Cannot find arg0 for \""+func+"\"");
238             return out;
239         }
240         if(!arg0.isRegister){
241             error("Arg0 for \""+func+"\" is not a register");
242             return out;
243         }
244         argRegister=translateRegister(arg0.arg);
245         if(argRegister==""){
246             error("Register \""+arg0.arg+"\" doesnt exists for \""+
                    func+"\"");
247             return out;
248         }
249         target0=argRegister;
250         out.push_back(nRisk_jr+target0);
251         out.push_back("00000000");
252         return out;
253     }else if(!func.compare("j")){//SEMI INTERPRETADA
254         arg0=getArgFrom(args,0);
255         if(arg0.err){
256             error("Cannot find arg0 for \""+func+"\"");
257             return out;
258         }
259         if(arg0.isRegister){
260             error("Arg0 for \""+func+"\" is a register");
261             return out;
262         }
263         out.push_back(nRisk_jr+"0000");
264         out.push_back(arg0.arg);
265         return out;
266     }else if(!func.compare("la")||!func.compare("lc")){
267         arg0=getArgFrom(args,0);
268         if(arg0.err){
269             error("Cannot find arg0 for \""+func+"\"");
270             return out;
271         }
272         if(!arg0.isRegister){
273             error("Arg0 for \""+func+"\" is not a register");
274             return out;
275         }
276         argRegister=translateRegister(arg0.arg);
277         if(argRegister==""){
278             error("Register \""+arg0.arg+"\" doesnt exists for \""+
                    func+"\"");
279             return out;
280         }
281         target0=argRegister;
282         arg1=getArgFrom(args,arg0.pos);
283         if(arg1.err){
```

```
            error("Cannot find arg1 for \""+func+"\"");
            return out;
        }
        if(arg1.isRegister){
            error("Arg1 for \""+func+"\" is a register");
            return out;
        }
        if(!func.compare("la")){
            out.push_back(nRisk_la+target0);
            out.push_back(arg1.arg);
        }else if(!func.compare("lc")){
            if(strToInt(arg1.arg)>255){
                error("Constant overflow, your constant should be
                    less than 256");
                return out;
            }
            out.push_back(nRisk_lc+target0);
            out.push_back(bitset< 8 >(strToInt(arg1.arg)).to_string
                ());
        }
        return out;
    }else if(!func.compare("lw")||!func.compare("sw")){
        arg0=getArgFrom(args,0);
        if(arg0.err){
            error("Cannot find arg0 for \""+func+"\"");
            return out;
        }
        if(!arg0.isRegister){
            error("Arg0 for \""+func+"\" is not a register");
            return out;
        }
        argRegister=translateRegister(arg0.arg);
        if(argRegister==""){
            error("Register \""+arg0.arg+"\" doesnt exists for \""+
                func+"\"");
            return out;
        }
        target0=argRegister;
        arg1=getArgFrom(args,arg0.pos);
        if(arg1.err){
            error("Cannot find arg1 for \""+func+"\"");
            return out;
        }
        if(arg1.isRegister){
            error("Arg1 for \""+func+"\" is a register");
            return out;
        }
        arg2=getArgFrom(args,arg1.pos);
        if(arg2.err){
            error("Cannot find arg2 for \""+func+"\"");
            return out;
        }
        if(!arg2.isRegister){
            error("Arg2 for \""+func+"\" is not a register");
            return out;
        }
        argRegister=translateRegister(arg2.arg);
        if(argRegister==""){
            error("Register \""+arg2.arg+"\" doesnt exists for \""+
                func+"\"");
            return out;
        }
```

```
342         target2=argRegister;
343
344         if(!func.compare("lw"))
345             nRisk=nRisk_lw;
346         else if(!func.compare("sw"))
347             nRisk=nRisk_sw;
348         if(strToInt(arg1.arg)>15){
349             error("Offset \""+arg1.arg+"\" for \""+func+"\" should
                    be less than 16");
350             return out;
351         }
352         out.push_back(nRisk+target0);
353         out.push_back(bitset< 4 >(strToInt(arg1.arg)).to_string()+
                target2);
354         return out;
355     }//INTERPRETADAS
356     else if(!func.compare("addc")||!func.compare("andc")||!func.
            compare("orc")||!func.compare("norc")||
357             !func.compare("sltc")||!func.compare("src")||!func.
                    compare("slc")||!func.compare("beq")||
358             !func.compare("bof")){
359
360         arg0=getArgFrom(args,0);
361         if(arg0.err){
362             error("Cannot find arg0 for \""+func+"\"");
363             return out;
364         }
365         if(!arg0.isRegister){
366             error("Arg0 for \""+func+"\" is not a register");
367             return out;
368         }
369         arg1=getArgFrom(args,arg0.pos);
370         if(arg1.err){
371             error("Cannot find arg1 for \""+func+"\"");
372             return out;
373         }
374         if(!arg1.isRegister){
375             error("Arg1 for \""+func+"\" is not a register");
376             return out;
377         }
378         arg2=getArgFrom(args,arg1.pos);
379         if(arg2.err){
380             error("Cannot find arg2 for \""+func+"\"");
381             return out;
382         }
383         if(arg2.isRegister){
384             error("Arg2 for \""+func+"\" is a register");
385             return out;
386         }
387         argRegister=translateRegister(arg0.arg);
388         if(argRegister==""){
389             error("Register \""+arg0.arg+"\" doesnt exists for \""+
                    func+"\"");
390             return out;
391         }
392         target0=argRegister;
393         argRegister=translateRegister(arg1.arg);
394         if(argRegister==""){
395             error("Register \""+arg1.arg+"\" doesnt exists for \""+
                    func+"\"");
396             return out;
397         }
```

```
398         target1=argRegister;
399         if(func.compare("beq")&&func.compare("bof"))
400         if(strToInt(arg2.arg)>255){
401             error("Constant overflow, your constant should be less
                    than 256");
402             return out;
403         }
404         out.push_back(nRisk_lc+nRisk_nt);
405         if(!func.compare("beq")||!func.compare("bof")){
406             out.push_back(arg2.arg);
407         }else{
408             out.push_back(bitset< 8 >(strToInt(arg2.arg)).to_string
                    ());
409         }
410         if(!func.compare("addc"))
411             nRisk=nRisk_add;
412         else if(!func.compare("andc"))
413             nRisk=nRisk_and;
414         else if(!func.compare("orc"))
415             nRisk=nRisk_or;
416         else if(!func.compare("norc"))
417             nRisk=nRisk_nor;
418         else if(!func.compare("sltc"))
419             nRisk=nRisk_slt;
420         else if(!func.compare("src"))
421             nRisk=nRisk_sr;
422         else if(!func.compare("slc"))
423             nRisk=nRisk_sl;
424         else if(!func.compare("beq"))
425             nRisk=nRisk_brq;
426         else if(!func.compare("bof"))
427             nRisk=nRisk_brf;
428         out.push_back(nRisk+target0);
429         out.push_back(target1+nRisk_nt);
430         return out;
431     }else if(!func.compare("mov")||!func.compare("move")||!func.
            compare("not")){
432         arg0=getArgFrom(args,0);
433         if(arg0.err){
434             error("Cannot find arg0 for \""+func+"\"");
435             return out;
436         }
437         if(!arg0.isRegister){
438             error("Arg0 for \""+func+"\" is not a register");
439             return out;
440         }
441         arg1=getArgFrom(args,arg0.pos);
442         if(arg1.err){
443             error("Cannot find arg1 for \""+func+"\"");
444             return out;
445         }
446         if(!arg1.isRegister){
447             error("Arg1 for \""+func+"\" is not a register");
448             return out;
449         }
450         argRegister=translateRegister(arg0.arg);
451         if(argRegister==""){
452             error("Register \""+arg0.arg+"\" doesnt exists for \""+
                    func+"\"");
453             return out;
454         }
455         target0=argRegister;
```

```
456         argRegister=translateRegister(arg1.arg);
457         if(argRegister==""){
458             error("Register \""+arg1.arg+"\" doesnt exists for \""+
                    func+"\"");
459             return out;
460         }
461         target1=argRegister;
462 
463         if(!func.compare("mov")||!func.compare("move"))
464             nRisk=nRisk_add;
465         else if(!func.compare("not"))
466             nRisk=nRisk_nor;
467 
468 
469         out.push_back(nRisk+target0);
470         out.push_back(target1+nRisk_zro);
471         return out;
472 
473     }else if(!func.compare("jal")||!func.compare("jrl")){
474         if(!func.compare("jal")){
475             arg0=getArgFrom(args,0);
476             if(arg0.err){
477                 error("Cannot find arg0 for \""+func+"\"");
478                 return out;
479             }
480             if(arg0.isRegister){
481                 error("Arg0 for \""+func+"\" is a register");
482                 return out;
483             }
484             target0=nRisk_nt;
485             out.push_back(nRisk_lc+target0);
486             out.push_back(arg0.arg);
487         }else{
488             arg0=getArgFrom(args,0);
489             if(arg0.err){
490                 error("Cannot find arg0 for \""+func+"\"");
491                 return out;
492             }
493             if(!arg0.isRegister){
494                 error("Arg0 for \""+func+"\" is not a register");
495                 return out;
496             }
497             argRegister=translateRegister(arg0.arg);
498             if(argRegister==""){
499                 error("Register \""+arg0.arg+"\" doesnt exists for
                        \""+func+"\"");
500                 return out;
501             }
502             target0=argRegister;
503         }
504         out.push_back(nRisk_jr+target0);
505         out.push_back("00000000");
506         out.push_back(nRisk_add+nRisk_ra);
507         out.push_back(nRisk_nt+nRisk_zro);
508         return out;
509     }else if(!func.compare("push")||!func.compare("pop")){
510         arg0=getArgFrom(args,0);
511         if(arg0.err){
512             error("Cannot find arg0 for \""+func+"\"");
513             return out;
514         }
515         if(!arg0.isRegister){
```

```
516             error("Arg0 for \""+func+"\" is not a register");
517             return out;
518         }
519         argRegister=translateRegister(arg0.arg);
520         if(argRegister==""){
521             error("Register \""+arg0.arg+"\" doesnt exists for \""+
                    func+"\"");
522             return out;
523         }
524         target0=argRegister;
525         if(nRisk_StackSigal>0){
526             if(!func.compare("push")){
527                 //escreve soma
528                 out.push_back(nRisk_sw+target0);
529                 out.push_back("0000"+nRisk_sp);
530                 out.push_back(nRisk_lc+nRisk_nt);
531                 out.push_back(bitset< 8 >(1).to_string());
532                 out.push_back(nRisk_add+nRisk_sp);
533                 out.push_back(nRisk_sp+nRisk_nt);
534             }else{
535                 //subtrai le
536                 out.push_back(nRisk_lc+nRisk_nt);
537                 out.push_back(bitset< 8 >(1).to_string());
538                 out.push_back(nRisk_sub+nRisk_sp);
539                 out.push_back(nRisk_sp+nRisk_nt);
540                 out.push_back(nRisk_lw+target0);
541                 out.push_back("0000"+nRisk_sp);
542             }
543         }else{
544             if(!func.compare("push")){
545                 //subtrai escreve
546                 out.push_back(nRisk_lc+nRisk_nt);
547                 out.push_back(bitset< 8 >(1).to_string());
548                 out.push_back(nRisk_sub+nRisk_sp);
549                 out.push_back(nRisk_sp+nRisk_nt);
550                 out.push_back(nRisk_sw+target0);
551                 out.push_back("0000"+nRisk_sp);
552             }else{
553                 //le soma
554                 out.push_back(nRisk_lw+target0);
555                 out.push_back("0000"+nRisk_sp);
556                 out.push_back(nRisk_lc+nRisk_nt);
557                 out.push_back(bitset< 8 >(1).to_string());
558                 out.push_back(nRisk_add+nRisk_sp);
559                 out.push_back(nRisk_sp+nRisk_nt);
560             }
561         }
562         return out;
563     }else{
564         error("Function \""+func+"\" doesnt exists");
565         return out;
566     }
567 }
568 
569 int main(int argc, char **argv){
570      if(argc<2){
571           error("No such file or arguments");
572           return -1;
573      }
574      input=argv[1];
575      for(int i=2;i<argc;i++){
576           if(argv[i][0]=='-'&&argv[i][1]=='o'){
```

```
577                 output=argv[++i];
578             }
579         }
580         string line;
581         vector<string> code;
582         ifstream codeFile (input);
583         if (codeFile.is_open()){
584             while (getline (codeFile,line)){
585              code.push_back(line);
586             }
587             codeFile.close();
588         }else{
589              error("Unable to reach file");
590              return -1;
591         }
592         vector<string> machineCode;
593        vector<string> dataSeg;
594        string dataSegSize;
595         bool data=false;
596         bool text=false;
597        int commentedLines=0;
598        for(int i=0;i<code.size();i++){
599            if(code[i][0]=='#'||code[i][0]=='\n'){
600                code.erase(code.begin()+i);
601                i--;
602                commentedLines++;
603            }else{
604                int foundAt=code[i].find("#");//size
605                if(foundAt!=string::npos){
606                    code[i].erase(code[i].begin()+foundAt,code[i].end()
                        );
607                }
608            }
609        }
610        int currentAddress=0;
611        int currentDataAddress=0;
612        int doubleDotAt;
613        for(int i=0;i<code.size();i++){
614            if(code[i].find(".data")!=string::npos){
615                data=true;
616            }else if(code[i].find(".text")!=string::npos){
617                text=true;
618            }else if(data&&!text){//MERGE DATA WITH MACHINE CODE
619                doubleDotAt=code[i].find(":");//size
620                if(doubleDotAt==string::npos)
621                    doubleDotAt=0;
622                else{
623                    string tmp=code[i].substr(0,doubleDotAt);
624                    code[i]=code[i].substr(doubleDotAt+1);
625                    tmp.erase(remove(tmp.begin(),tmp.end(),' '),tmp.end
                        ());
626                    tmp.erase(remove(tmp.begin(),tmp.end(),'\t'),tmp.
                        end());
627                    addresses[tmp]=currentDataAddress;
628                }
629                code[i]=code[i].substr(code[i].find("."));
630                string varType=code[i].substr(0,code[i].find(" "));
631                if(!varType.compare(".word")){
632                    code[i]=code[i].substr(code[i].find(" "));
633                    code[i].erase(remove(code[i].begin(),code[i].end(),
                        ' '),code[i].end());//remove spaces
```

```
                code[i].erase(remove(code[i].begin(),code[i].end(),'\t'),code[i].end());//remove spaces
                int last=0;
                for(int j=0;j<=code[i].size();j++){
                    if(code[i][j]==','||code[i][j]=='\0'){
                        dataSeg.push_back(bitset< 8 >(strToInt(code[i].substr(last,j-last))).to_string());
                        last=j;
                        currentDataAddress++;
                    }
                }
            }
        }else if(text){
            doubleDotAt=code[i].find(":");//size
            if(doubleDotAt==string::npos)
                doubleDotAt=0;
            else{
                string tmp=code[i].substr(0,doubleDotAt);
                tmp.erase(remove(tmp.begin(),tmp.end(),' '),tmp.end());
                tmp.erase(remove(tmp.begin(),tmp.end(),'\t'),tmp.end());
                addresses[tmp]=currentAddress;
            }
            int start=-1;
            for(int j=doubleDotAt;j<=code[i].size();j++){
                if(start<0&&isValidChar(code[i][j])){
                    start=j;
                }else if(start>=0){
                    if(code[i][j]==' '||code[i][j]=='\0'||code[i][j]=='\t'){
                        vector<string> translated=translateFunc(code[i].substr(start,j-start),code[i].substr(j+1));
                        if(translated.size()!=0){
                            for(int t=0;t<translated.size();t++){
                                machineCode.push_back(translated[t]);
                                currentAddress++;
                            }
                        }else{
                            error("Translating line ("+to_string(i+commentedLines)+")");
                            return -1;
                        }
                         break;
                    }
                }
            }

        }
    }
    for(int i=1;i<machineCode.size();i+=2)
        if(machineCode[i].size()!=8||!isBinary(machineCode[i])){
            if(addresses.find(machineCode[i]) == addresses.end()){
                error("Cannot jump to \""+machineCode[i]+"\", tag doesnt exists");
                return -1;
            }else{
                if(addresses[machineCode[i]]>255){
                    error("Address overflow("+to_string(addresses[machineCode[i]])+"), your code is too big")
```

```
                        ;
685                     return -1;
686                 }
687                 machineCode[i]=bitset< 8 >(addresses[machineCode[i
                        ]]).to_string();
688             }
689         }
690     dataSegSize=bitset< 8 >(dataSeg.size()).to_string();
691     machineCode.insert(machineCode.begin(), dataSegSize);
692     for(int i=0;i<dataSeg.size();i++){
693         machineCode.insert(machineCode.begin(), dataSeg[i]);
694     }
695      ofstream machineFile (output);
696      if (machineFile.is_open()){
697          for(int i=0;i<machineCode.size();i++){
698             machineFile<<machineCode[i]<<" ";
699             if((i+1)%2==0&&i+1<machineCode.size())
700                 machineFile<<endl;
701         }
702          machineFile.close();
703      }else{
704           error("Unable to create file");
705           return -1;
706      }
707      return 0;
708 }
```

8 Caminho de dados

O desenho do primeiro esboço do caminho de dados pode ser visto abaixo:

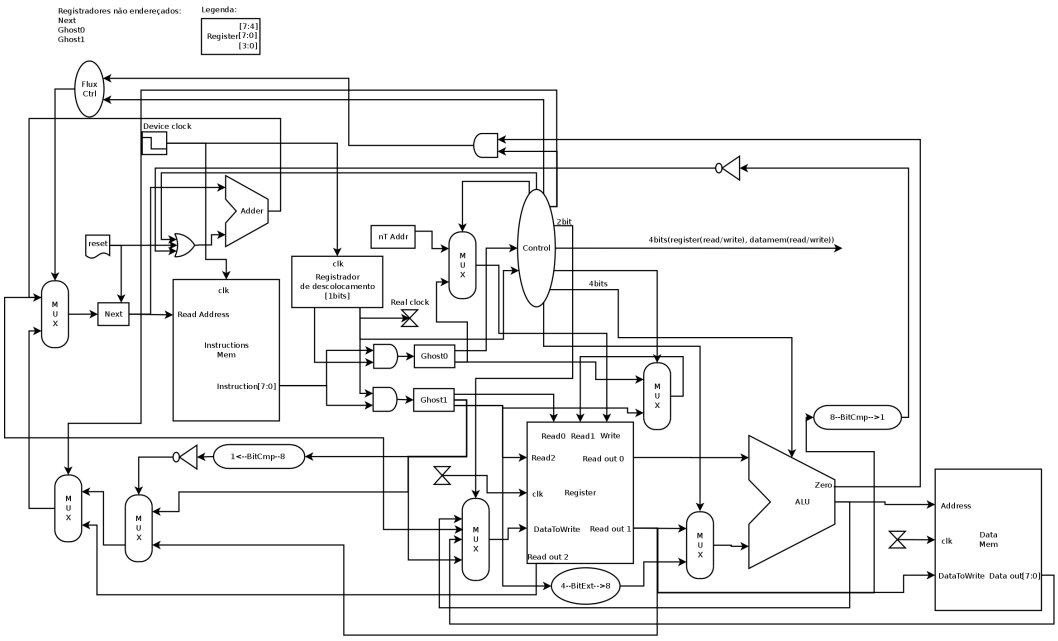


Figura 1: Caminho de dados nanoRisk

Para a implementação do projeto de um processador de 8 bits com instruções que necessitam de 16 bits foi necessário adicionar dois registradores temporarios não endereçados para armazenar cada parte da instrução e dividir o clock por dois para as intruções usando um registrador de deslocamento de 1 bit(um bit sempre é um e outro sempre é zero), esse registrador de deslocamento também serve para alternar em qual registrador "ghost"será armazenado aquele pedaço de instrução. Os Bit Compacters(BitCmp) são uma porta lógica or para 8 entradas. Durante o desenvolvimento do diagrama, foi possivel observar algumas coisas desnecessárias que foram descritas no projeto, como por exemplo o registrador bp(base pointer) e a função la(load address) que poderia ser substituida pela função load constant sem problemas já que não há calculos de alinhamento.

8.1 Bits de controle

Há 9 saídas do controle, que totalizam 16 bits de cotrole. As nove saídas são descritas abaixo seguindo a ordem de cima para baixo no diagrama do caminho de dados.

- 1-Temporary Write(tmpwr)
  Bit para definir se o registrador de escrita vai ser o temporário se for zero ou o registrador passado na instrução se for um. Em uma nova versão do projeto seria possível alterar a estrutura da instruções para que as que usam o registrador temporário não precissasem passar ele como parâmetro. As funções de link que gravam o endereço de next+1, usam esse bit para gravar no registrador nT o valor de retorno, poderia ser usado o endereço de RA também mas não é o caso devido ao projeto.

- 2-Halt(hlt)
  Bit para definir o estado sleep, se ele for zero e o registrador de flag for diferente de zero o programa continua congelado nessa instrução. Um sinal externo de reset pode reiniciar o programa, ou caso haja uma modificação no projeto para adicionar outros cores um deles pode tirar desse estado também.

- 3-Jump(jmp)
  Define se a instrução é uma instrução de jump incondicional, se for um, o proximo endereço não virá de next+1

- 4-Branch(brc)
  Define se a instrução é uma instrução de branch condicional, se for um e a operação na ula retornar verdadeiro habilita o branch, o endereço virá da leitura 2 do banco de registradores. Esse campo de leitura é usado apenas para esse caso e caso mudasse o projeto inteiro das intruções seria possivel evita-lo passando o registrador de retorno primeiro, ou mudando a ordem apenas para esse tipo de instrução.

- 5-Register Write(rgw)
  Dois bits que definem o que vai ser escrito no registrador, o valor dos bits se encontra abaixo e após está seu significado, sendo que o primeiro é o mais significativo:

  00 - Resultado da ula.

  01 - Próxima instrução.

  10 - Valor lido na memória de dados.

  11 - Constante passada pela instrução.

- 6-Input Output Control(ioc)
  Quatro bits que definem as entradas e saídas em termos de memória e registrador eles controlam escrita e leitura na memoria de dados e no banco de registradores. Sendo os bits 4,3,2 e 1 os bits desse sinal de controle do mais significativo pro menos significativo, eles repesentam:

  4 - Habilita escrita no banco de registradores se for um.

  3 - Habilita leitura no banco de registradores se for um.

2 - Habilita escrita na memória de dados se for um.

1 - Habilita leitura na memória de dados se for um.

- 7-Register Read(rgr)
  Bit que define de onde vem o segundo valor a ser lido no banco de registradores, se for zero será lido a partir do enderço passado nos ultimo quatro bits do primeiro byte da instrução, se for um será lido a partir do enderço passado nos ultimo quatro bits do segundo byte da instrução.

- 8-ALU Operation(alo)
  Quatro bits que definem qual operação será feita na ula. O códigos de cada operação estão abaixo e após está seu significado, lembrando q o primeiro bit é o mais significativo:

  0000 - Adição, a saída é a soma das entradas.

  0001 - Subtração do primeiro valor na ula pelo segundo, se o resultado for zero, define a saída zero como um.

  0010 - Teste de flag, faz um & lógico das entradas, se o resultado for verdadeiro para todos os bits onde a segunda entrada é um ele define a saída zero como um e o resultado será a subtração do primeiro pelo segundo, caso contrario a saída zero será zero e a saída sera a primeira entrada.

  0011 - And, a saída é a operação lógica and das entradas.

  0100 - Or, a saída é a operação lógica or das entradas.

  0101 - Nor, a saída é a operação lógica nor das entradas.

  0110 - Less, faz a subtração da segunda entrada pela primera, define a saída zero como um se o valor for negativo.

  0111 - Shift left, a saída é a primeira entrada deslocada para a esquerda o n vezes, onde n é o valor da segunda entrada

  1000 - Shift right, a saída é a primeira entrada deslocada para a direita o n vezes, onde n é o valor da segunda entrada

  1001 - Não usado

  1010 - Não usado

  1011 - Não usado

  1100 - Não usado

  1101 - Não usado

  1110 - Não usado

  1111 - Não usado

- 9-ALU Argument(ala)
  Bit que define segundo argumento da ula, pode ser um valor lido no banco de registradores se for zero ou uma constante/endreço passada pela função se for um.

8.2 Valores de controle especificos

A tabela abaixo contém os sinais de controle para cada função do processaodor nanoRisk, após analizar atentamente a tabela é possivel ver que podem ser feitas vária modificações para simplificar e diminuir a quantidade de bits de controle mas deixando tudo menos didatico.

Tabela 7: Controle de instruções

Instrução	tmpwr	hlt	jmp	brc	rgw	ioc	rgr	alo	ala
slp	X	0	0	0	XX	0100	0	XXXX	X
brq	X	1	0	1	XX	0100	0	0001	0
brf	1	1	0	1	00	1100	0	0010	0
add	1	1	0	0	00	1100	1	0000	0
sub	1	1	0	0	00	1100	1	0001	0
and	1	1	0	0	00	1100	1	0011	0
or	1	1	0	0	00	1100	1	0100	0
nor	1	1	0	0	00	1100	1	0101	0
slt	1	1	0	0	00	1100	1	0110	0
sr	1	1	0	0	00	1100	1	1000	0
sl	1	1	0	0	00	1100	1	0111	0
jr	0	1	1	0	01	1100	0	XXXX	X
la	1	1	0	0	11	1000	X	XXXX	X
lc	1	1	0	0	11	1000	X	XXXX	X
lw	1	1	0	0	10	1101	0	0000	1
sw	X	1	0	0	XX	0110	0	0000	1

9 Caminho das instruções

O caminho de dados especificos das instruções de acordo com o tipo pode ser visto nas imagens abaixo.

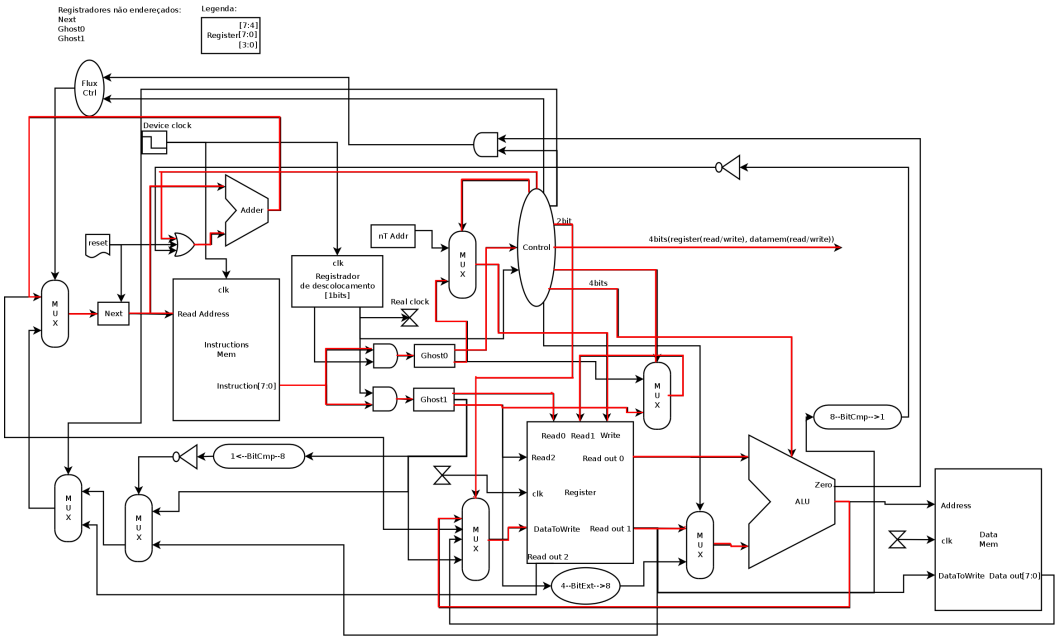


Figura 2: Caminho de dados instruções lógicas e aritméticas

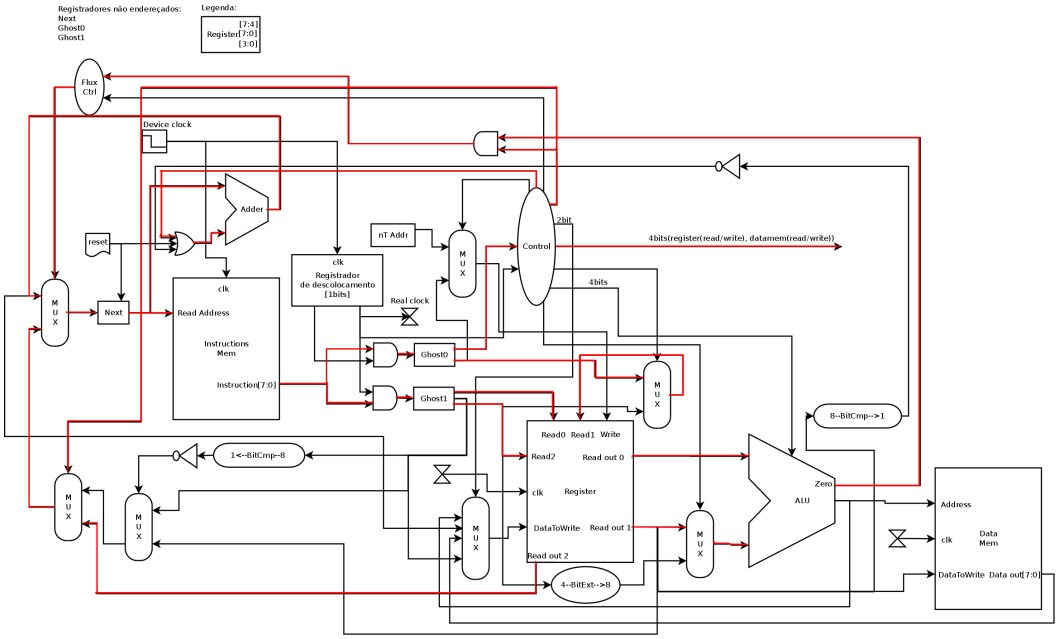


Figura 3: Caminho de dados instruções de branch

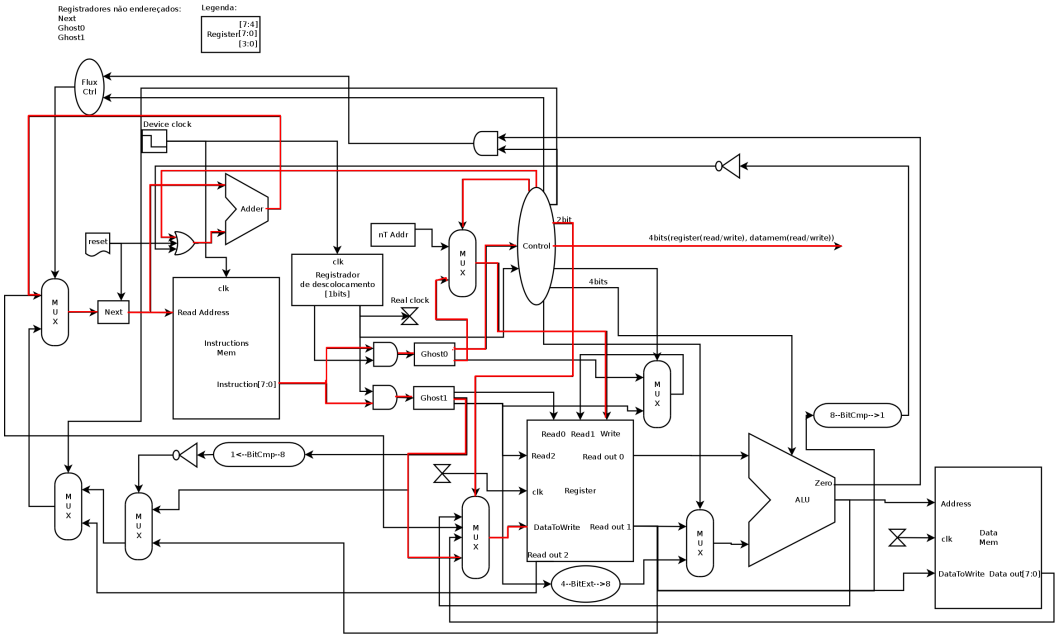


Figura 4: Caminho de dados intruções load address/constant

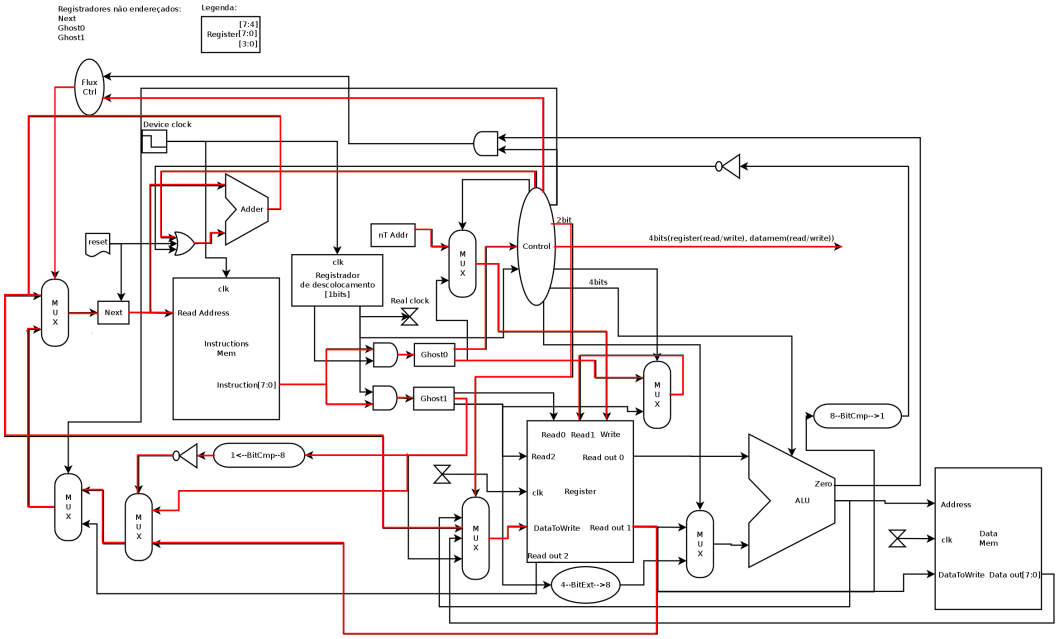


Figura 5: Caminho de dados instruções de jump

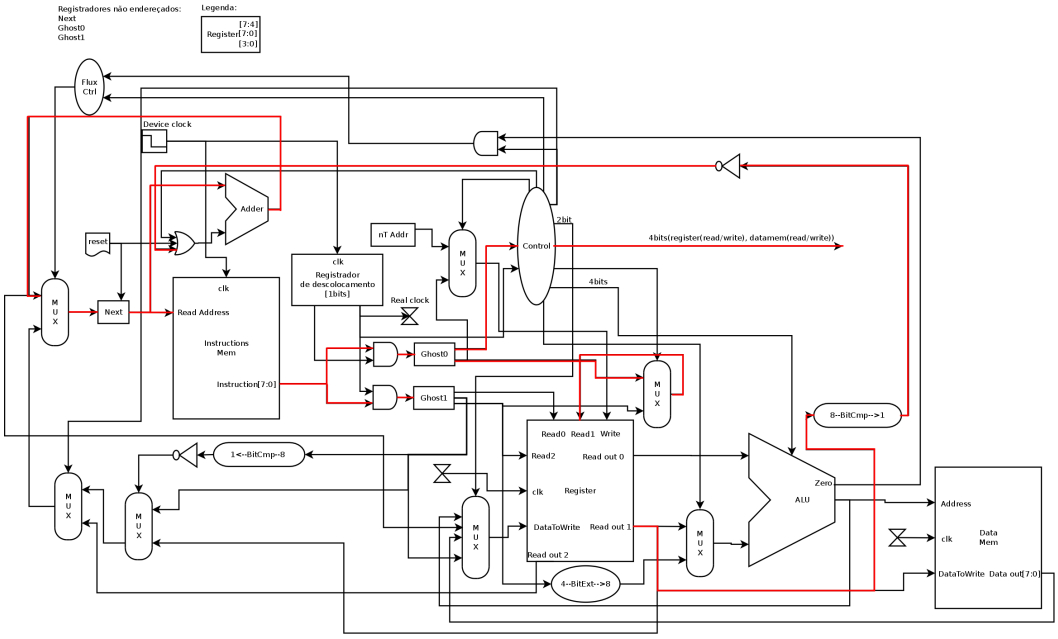


Figura 6: Caminho de dados intrução sleep

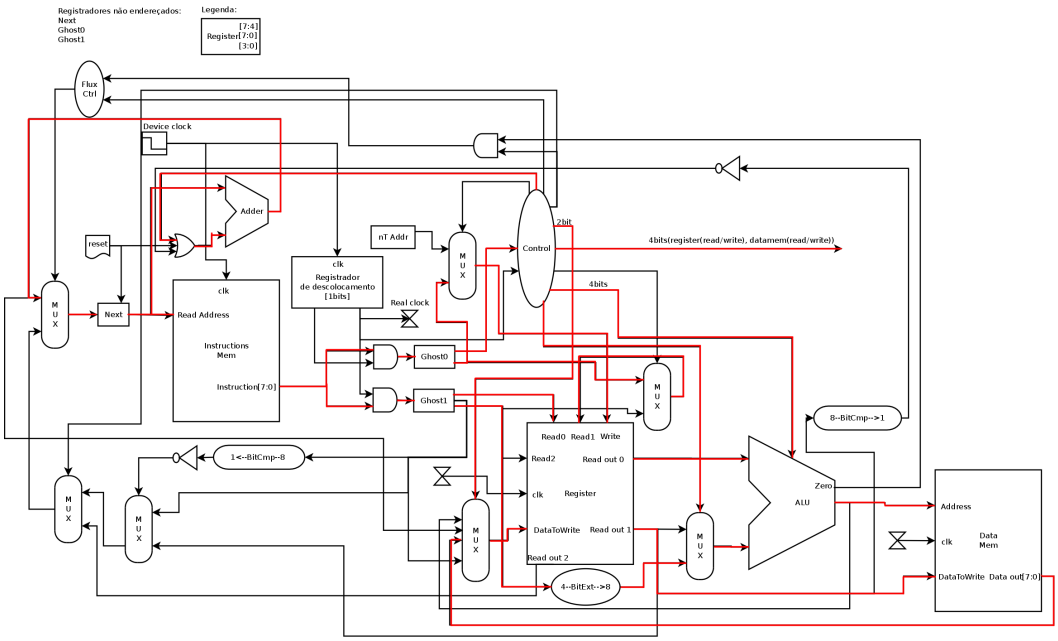


Figura 7: Caminho de dados intruções sw/lw

10 Alterações no projeto

10.1 Overflow

Durante a relização da prática foi notado que não havia sido projetado uma forma de gravar as exceções de overflow, então foi adicionado no banco de registradores a entrada "ovrflw", que sai da ALU, essa entrada de dois bits informa estados de overflow que são gravados de forma assincrona no registrador flg, o bit mais significativo é o underflow e o menos significativo é o overflow. O overflow é gravado usando o ou lógico, assim conserva os dados gravados previamente.

10.2 Sincronização

Um caso de erro que pode ocorrer devido alguma falha eletrica, lógica ou humana é o estado onde o endereço de next é um numero impar e o registrador ghost atual é o 0, isso faria com que o programa não funcionasse, então uma forma de forçar para que isso não ocorra de forma alguma é adicionar uma porta logica AND, na saída 1 do registrador de deslocamento(real clock) e a outra entrada dessa AND seria o bit menos significativo de next, assim impediariamos o clock de funcionar na hora errada. Seria necessário também colocar uma AND na entrada de clock de ghost0 juntamente com o bit menos significativo de next invertido, para evitar a gravação em um momento errado.

11 Componentes de memória

Foram implementados em verilog registrador, registrador de deslocamento, banco de registradores e memória, abaixo podemos ver o código verilog desses componentes, acima do código de cada há um comentario explicando suas entradas e saídas.

```
//Modulo dos registrador de deslocamento:
//Entradas:
//        clk: clock
//        set: define o valor inicial caso seja 1
//Saidas:
//        out0: primeiro bit
//        out1: segundo bit
module nR_BitShifter(clk,s_out0,s_out1,set);
     input clk,set;
     output reg s_out0,s_out1;

     initial
     begin
          s_out0<=1;
          s_out1<=0;
     end

     always @ (posedge clk | set) begin
          if(~set) begin
               s_out0<=s_out1;
               s_out1<=s_out0;
          end
     end
     always @ (set) begin
          s_out0<=1;
          s_out1<=0;
```

```
     end
endmodule

//Modulo dos registradores: ghost0,ghost1,next
//Entradas:
//         clk: clock(escrita apenas em borda de subida)
//         clr: limpa a memoria caso seja 1
//         in0: valor a ser gravado
//Saidas:
//         out: valor registrado
module nR_Register(clk,in0,r_out0,clr);
     input clk,clr;
     input[7:0] in0;
     output reg[7:0] r_out0;

     initial
     begin
          r_out0<=8'b00000000;
     end

     always @ (posedge clk) begin
          if(~clr)
               r_out0<=in0;
     end

     always @ (clr) begin
          r_out0<=0;
     end
endmodule

//Modulo memory: dataMem, instrMem
//Entradas:
//         clk: clock(escrita=borda de subida,leitura=borda de
    descida)
//         clr: limpa a memoria caso seja 1
//         adrIn0: endereco onde deve gravar um valor
//         in0: valor a ser gravado
//         adrOut0: endereco de leitura do valor 0
//         canWrt: habilita escrita
//         canRd: habilita leitura

//Saidas:
//         out0: valor lido em 0
module nR_Memory(clk,in0,m_out0,adrIn0,adrOut0,canWrt,canRd,clr);
     input clk,clr,canWrt,canRd;
     input[7:0] in0,adrIn0,adrOut0;
     output reg[7:0] m_out0;

     reg [7:0]m_mem[255:0];

     integer m_i;

     initial
     begin
          for (m_i=0;m_i<256;m_i=m_i+1)
                    m_mem[m_i]<=0;
               m_out0<=0;
     end

     always @ ( posedge clk & canWrt ) begin
          if(~clr)
               m_mem[adrIn0]<=in0;
```

```
     end

     always @ (clr) begin
          for (m_i=0;m_i<256;m_i=m_i+1)
               m_mem[m_i]<=0;
          m_out0<=0;
     end

     always @ (negedge clk & canRd) begin
          m_out0<=m_mem[adrOut0];
     end
endmodule

//Modulo registradores: dataMem, instrMem
//Entradas:
//         clk: clock(escrita=borda de subida,leitura=borda de
    descida)
//         clr: limpa a memoria caso seja 1
//         adrIn0: endereco onde deve gravar um valor
//         in0: valor a ser gravado
//         adrOut0: endereco de leitura do valor 0
//         adrOut1: endereco de leitura do valor 1
//         adrOut2: endereco de leitura do valor 1
//         canWrt: habilita escrita
//         canRd: habilita leitura
//       ovrflw: define overflow positivo ou negativo

//Saidas:
//         out0: valor lido em 0
//         out1: valor lido em 1
//         out2: valor lido em 2
module nR_RegisterBank(clk,in0,rb_out0,rb_out1,rb_out2,adrIn0,
    adrOut0,adrOut1,adrOut2,canWrt,canRd,clr,ovrflw);
     input clk,clr,canWrt,canRd;
     input[1:0] ovrflw;
     input[7:0] in0;
     input[3:0] adrIn0,adrOut0,adrOut1,adrOut2;
     output reg[7:0] rb_out0,rb_out1,rb_out2;

     reg[7:0]rb_mem[15:0];

     integer rb_i;

     initial
     begin
          for (rb_i=0;rb_i<16;rb_i=rb_i+1) begin
                    rb_mem[rb_i]<=0;
               end
          rb_out0<=0;
          rb_out1<=0;
          rb_out2<=0;
     end

     always @ (posedge clk & canWrt) begin
          if(~clr)
               rb_mem[adrIn0]<=in0;
     end

     always @ (negedge clk & canRd) begin
          rb_out0<=rb_mem[adrOut0];
          rb_out1<=rb_mem[adrOut1];
          rb_out2<=rb_mem[adrOut2];
```

```
    end

    always @ (clr) begin
        for (rb_i=0;rb_i<16;rb_i=rb_i+1) begin
            rb_mem[rb_i]<=0;
        end
        rb_out0<=0;
        rb_out1<=0;
        rb_out2<=0;
    end

    always @ (ovrflw[0]|ovrflw[1])begin//[0]=positive overflow,
        [1]=negative overflow
        rb_mem[1][0]<=rb_mem[1][0]|ovrflw[0];
        rb_mem[1][1]<=rb_mem[1][1]|ovrflw[1];
    end

endmodule
```

11.1 Simulação

Para verificar o funcionamento dos componentes foi criado um código extra para simulações que é o código abaixo:

```
//MODULOS DE TESTE

module nR_MemoryComponents_BS_Test();
    reg bs_clk;
    reg bs_set;
    wire bs_out0,bs_out1;
    always
    begin
        #1 bs_clk <= ~bs_clk;
    end

    initial
    begin
        bs_clk <= 1'b0;
        bs_set <= 1'b0;
        $monitor("BitShifter: Clock %b Out0 = %0d Out1 = %0d Set
            = %b",bs_clk,bs_out0,bs_out1,bs_set);
        #15 bs_set <= 1'b1;
        #2 bs_set <= 1'b1;
        #10 $stop;
    end

    nR_BitShifter Deslocador(bs_clk,bs_out0,bs_out1,bs_set);
endmodule

module nR_MemoryComponents_Reg_Test();
    reg reg_clk;
    integer reg_in=0;
    wire[7:0] reg_out;
    reg reg_clr;
    always
    begin
        #1 reg_in<=reg_in+1; if(reg_in>15) reg_in<=0;
        #1 reg_clk <= ~reg_clk;
    end

    initial
```

```
    begin
        reg_clk <= 1'b0;
        reg_clr <= 1'b0;
        $monitor("Register: Clock = %b In = %0d Out  = %0d Clr =
            %b",reg_clk,reg_in,reg_out,reg_clr);
        #27 reg_clr <= 1'b1;
        #5 reg_clr <= 1'b0;
        #10 $stop;
    end

    nR_Register Register(reg_clk,reg_in,reg_out,reg_clr);
endmodule

module nR_MemoryComponents_RegBank_Test();
    reg rb_clk;
    reg rb_canWr;
    reg rb_canRd;
    reg rb_clr;
    reg[1:0] rb_over;
    integer rb_in=1;
    integer rb_Adrin=2;
    integer rb_Adrout0=2;
    integer rb_Adrout1=0;
    integer rb_Adrout2=0;
    wire[7:0] rb_out0;
    wire[7:0] rb_out1;
    wire[7:0] rb_out2;
    always
    begin
        #1 rb_in<=rb_in+1; if(rb_in>255) rb_in=0;
        #1 rb_Adrin<=rb_in;
        #1 rb_Adrout0<=rb_Adrout0+1; if(rb_Adrout0>255)
            rb_Adrout0<=2;
        #1 rb_Adrout1<=rb_Adrout0-1;
        #1 rb_Adrout2<=rb_Adrout0-2;
        #1 rb_clk <= ~rb_clk;
    end

    initial
    begin
        rb_clk <= 1'b0;
        rb_canWr<= 1'b1;
        rb_canRd<= 1'b1;
        rb_clr<= 1'b0;
        rb_over<=0;
        $monitor("Register Bank: Clock = %b Out0 = %0d Out1 = %0d
            Out2 = %0d AdrIn = %0d Adr(out0/out1+1/out2+2) = %0d
            CWr = %b CRd = %b Clr = %b OverFlow = %b",rb_clk,
            rb_out0,rb_out1,rb_out2,rb_Adrin,rb_Adrout0,rb_canWr,
            rb_canRd,rb_clr,rb_over);
        #27 rb_clr <= 1'b1;
        #1 rb_clr <= 1'b0;

        #5 rb_Adrout0<=2; rb_over<=3;
        #5 rb_Adrout0<=2; rb_over<=2;
        #5 rb_Adrout0<=2; rb_over<=1;
        #1 rb_over<=0;

        #3 rb_canWr <= 1'b0;
        #1 rb_canWr <= 1'b1;

        #10 rb_canRd <= 1'b0;
```

```
            #1 rb_canRd <= 1'b1;

            #4 $stop;
        end

        nR_RegisterBank Bank(rb_clk,rb_in,rb_out0,rb_out1,rb_out2,
            rb_Adrin,rb_Adrout0,rb_Adrout1,rb_Adrout2,rb_canWr,
            rb_canRd,rb_clr,rb_over);
endmodule

module nR_MemoryComponents_RAM_Test();
        reg mem_clk;
        reg mem_canWr;
        reg mem_canRd;
        reg mem_clr;
        integer mem_in=1;
        integer mem_Adrin=0;
        integer mem_Adrout=0;
        wire[7:0] mem_out;
        always
        begin
            #1 mem_in<=mem_in+1; if(mem_in>255) mem_in<=0;
            #1 mem_Adrin<=mem_in;
            #1 mem_Adrout<=mem_Adrout+1; if(mem_Adrout>255)
                mem_Adrout<=0;
            #1 mem_clk <= ~mem_clk;
        end

        initial
        begin
            mem_clk <= 1'b0;
            mem_canWr<= 1'b1;
            mem_canRd<= 1'b1;
            mem_clr<= 1'b0;
            $monitor("Memory: Clock = %b Out = %0d AdrIn = %0d AdrOut
                = %0d CWr = %b CRd = %b Clr = %b",mem_clk,mem_out,
                mem_Adrin,mem_Adrout,mem_canWr,mem_canRd,mem_clr);
            #27 mem_clr <= 1'b1;
            #1 mem_clr <= 1'b0;

            #10 mem_canWr <= 1'b0;

            #10 mem_canRd <= 1'b0;

            #7 $stop;
        end

        nR_Memory RAM(mem_clk,mem_in,mem_out,mem_Adrin,mem_Adrout,
            mem_canWr,mem_canRd,mem_clr);
endmodule
```

11.1.1 Resultados

As imagens abaixo são prints das simulações e serão usadas para analisar o funcionamento do código.

- Registrador de Deslocamento

```
# Loading work.nR_MemoryComponents_BS_Test
# Loading work.nR_BitShifter
VSIM 5> run
# BitShifter: Clock 0 Out0 = 1 Out1 = 0 Set = 0
# BitShifter: Clock 1 Out0 = 0 Out1 = 1 Set = 0
# BitShifter: Clock 0 Out0 = 0 Out1 = 1 Set = 0
# BitShifter: Clock 1 Out0 = 1 Out1 = 0 Set = 0
# BitShifter: Clock 0 Out0 = 1 Out1 = 0 Set = 0
# BitShifter: Clock 1 Out0 = 0 Out1 = 1 Set = 0
# BitShifter: Clock 0 Out0 = 0 Out1 = 1 Set = 0
# BitShifter: Clock 1 Out0 = 1 Out1 = 0 Set = 0
# BitShifter: Clock 0 Out0 = 1 Out1 = 0 Set = 0
# BitShifter: Clock 1 Out0 = 0 Out1 = 1 Set = 0
# BitShifter: Clock 0 Out0 = 0 Out1 = 1 Set = 0
# BitShifter: Clock 1 Out0 = 1 Out1 = 0 Set = 0
# BitShifter: Clock 0 Out0 = 1 Out1 = 0 Set = 0
# BitShifter: Clock 1 Out0 = 0 Out1 = 1 Set = 0
# BitShifter: Clock 0 Out0 = 0 Out1 = 1 Set = 0
# BitShifter: Clock 1 Out0 = 1 Out1 = 0 Set = 1
# BitShifter: Clock 0 Out0 = 1 Out1 = 0 Set = 1
# BitShifter: Clock 1 Out0 = 1 Out1 = 0 Set = 1
# BitShifter: Clock 0 Out0 = 1 Out1 = 0 Set = 1
# BitShifter: Clock 1 Out0 = 1 Out1 = 0 Set = 1
# BitShifter: Clock 0 Out0 = 1 Out1 = 0 Set = 1
# BitShifter: Clock 1 Out0 = 1 Out1 = 0 Set = 1
# BitShifter: Clock 0 Out0 = 1 Out1 = 0 Set = 1
# BitShifter: Clock 1 Out0 = 1 Out1 = 0 Set = 1
# BitShifter: Clock 0 Out0 = 1 Out1 = 0 Set = 1
# BitShifter: Clock 1 Out0 = 1 Out1 = 0 Set = 1
# BitShifter: Clock 0 Out0 = 1 Out1 = 0 Set = 1
```

Figura 8: Simulação do registrador de deslocamento

Podemos ver que quando o clock altera seu estado de zero para um, o circuito faz o deslocamento dos bits e de forma assincrona, quando set tem nivel logico alto o registrador vai para o estado inicial, logo o circuito funciona corretamente.

- Registrador

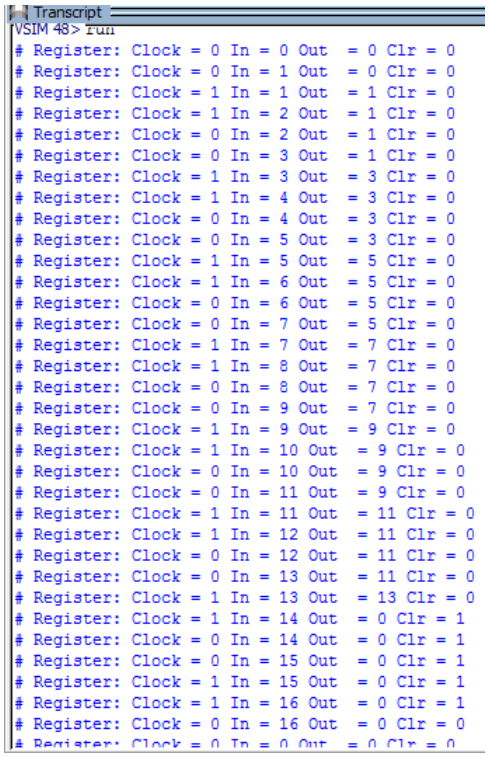


Figura 9: Simulação do registrador de deslocamento

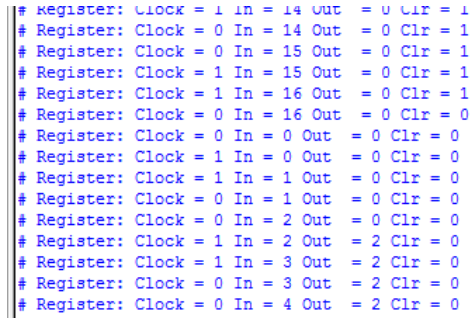


Figura 10: Simulação do registrador de deslocamento

Quando o clock tem borda de subida, o valor na entrada é gravado, quando temos borda de subida o valor é lido corretamente. Podemos ver tambem que o controle de leitura e escrita tambem funciona, permitindo ou nao leitura e escrita. O sinal assincrono de clear também está funcionado, zerando os valores quando tem nivel logico alto.

- Banco de Registradores

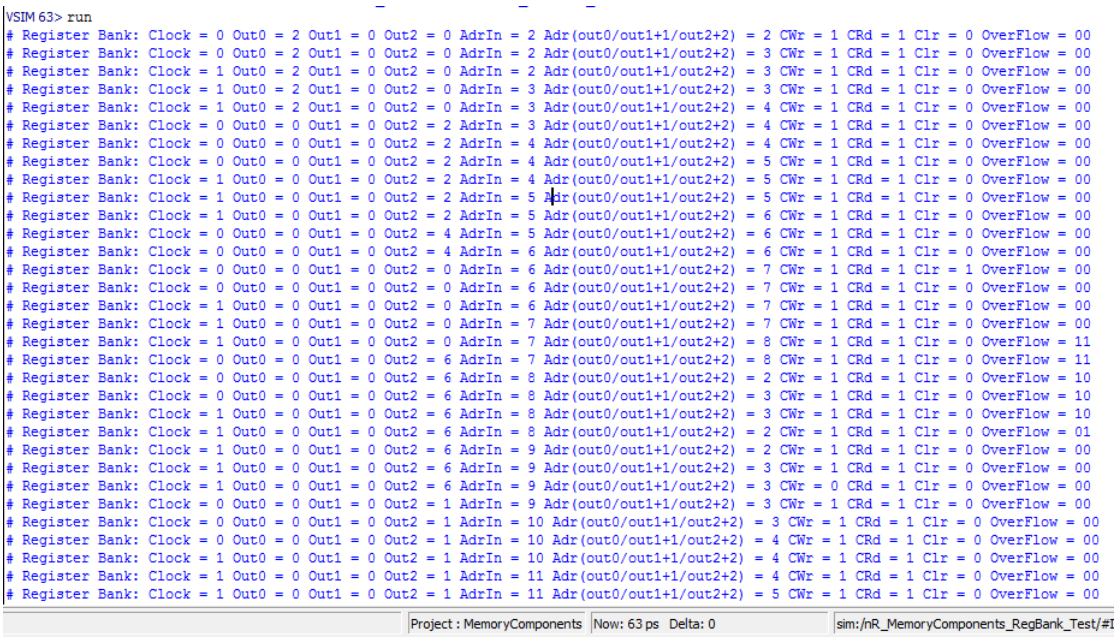
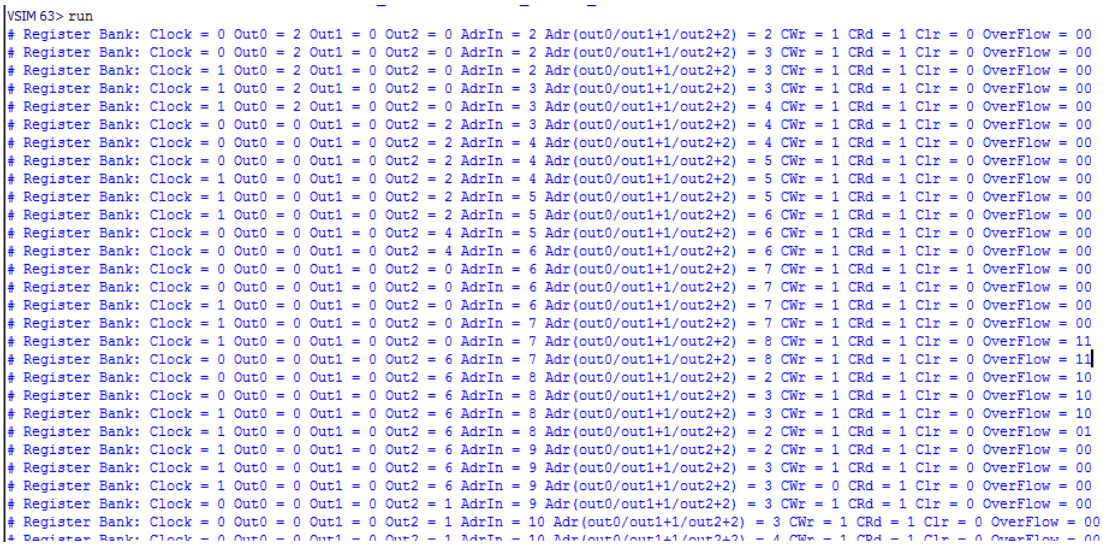


Figura 11: Simulação do banco de registradores, valor inicial zero

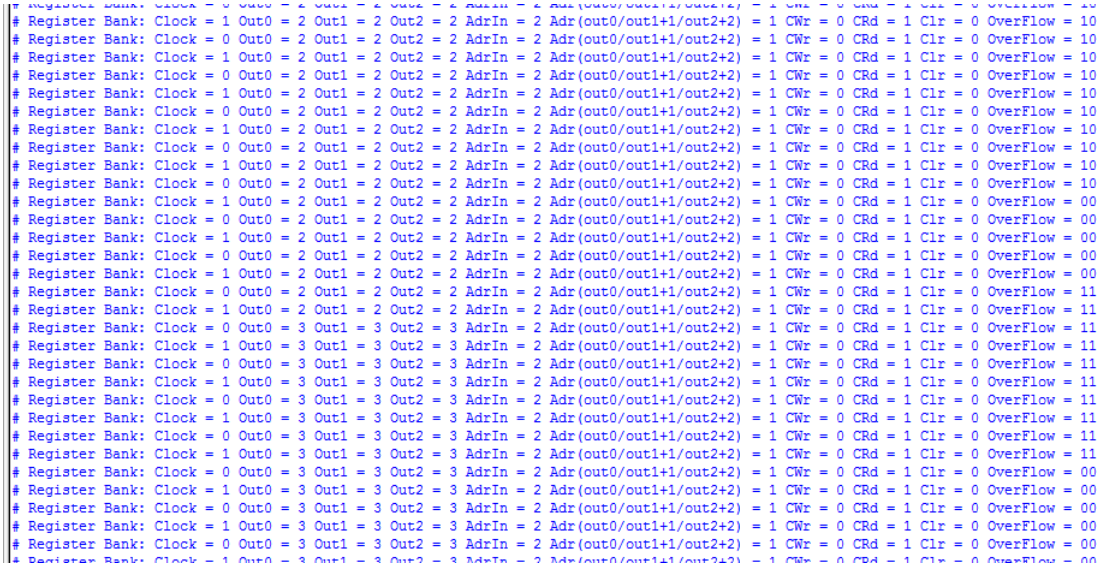
Quando o clock tem borda de subida, o valor no endereco de entradada(que é o mesmo da entrada nesse caso) é gravado, quando temos borda de subida o valor é lido corretamente. Podemos ver tambem que o controle de leitura e escrita tambem funciona, permitindo ou nao leitura e escrita. O sinal assincrono de clear também está funcionado, zerando os valores quando tem nivel logico alto.



```
VSIM 63> run
# Register Bank: Clock = 0 Out0 = 2 Out1 = 0 Out2 = 0 AdrIn = 2 Adr(out0/out1+1/out2+2) = 2 CWr = 1 CRd = 1 Clr = 0 OverFlow = 00
# Register Bank: Clock = 0 Out0 = 2 Out1 = 0 Out2 = 0 AdrIn = 2 Adr(out0/out1+1/out2+2) = 3 CWr = 1 CRd = 1 Clr = 0 OverFlow = 00
# Register Bank: Clock = 1 Out0 = 2 Out1 = 0 Out2 = 0 AdrIn = 2 Adr(out0/out1+1/out2+2) = 3 CWr = 1 CRd = 1 Clr = 0 OverFlow = 00
# Register Bank: Clock = 1 Out0 = 2 Out1 = 0 Out2 = 0 AdrIn = 3 Adr(out0/out1+1/out2+2) = 3 CWr = 1 CRd = 1 Clr = 0 OverFlow = 00
# Register Bank: Clock = 1 Out0 = 2 Out1 = 0 Out2 = 0 AdrIn = 3 Adr(out0/out1+1/out2+2) = 4 CWr = 1 CRd = 1 Clr = 0 OverFlow = 00
# Register Bank: Clock = 0 Out0 = 0 Out1 = 0 Out2 = 2 AdrIn = 3 Adr(out0/out1+1/out2+2) = 4 CWr = 1 CRd = 1 Clr = 0 OverFlow = 00
# Register Bank: Clock = 0 Out0 = 0 Out1 = 0 Out2 = 2 AdrIn = 4 Adr(out0/out1+1/out2+2) = 4 CWr = 1 CRd = 1 Clr = 0 OverFlow = 00
# Register Bank: Clock = 0 Out0 = 0 Out1 = 0 Out2 = 2 AdrIn = 4 Adr(out0/out1+1/out2+2) = 5 CWr = 1 CRd = 1 Clr = 0 OverFlow = 00
# Register Bank: Clock = 1 Out0 = 0 Out1 = 0 Out2 = 2 AdrIn = 4 Adr(out0/out1+1/out2+2) = 5 CWr = 1 CRd = 1 Clr = 0 OverFlow = 00
# Register Bank: Clock = 1 Out0 = 0 Out1 = 0 Out2 = 2 AdrIn = 5 Adr(out0/out1+1/out2+2) = 5 CWr = 1 CRd = 1 Clr = 0 OverFlow = 00
# Register Bank: Clock = 1 Out0 = 0 Out1 = 0 Out2 = 2 AdrIn = 5 Adr(out0/out1+1/out2+2) = 6 CWr = 1 CRd = 1 Clr = 0 OverFlow = 00
# Register Bank: Clock = 0 Out0 = 0 Out1 = 0 Out2 = 4 AdrIn = 5 Adr(out0/out1+1/out2+2) = 6 CWr = 1 CRd = 1 Clr = 0 OverFlow = 00
# Register Bank: Clock = 0 Out0 = 0 Out1 = 0 Out2 = 4 AdrIn = 6 Adr(out0/out1+1/out2+2) = 6 CWr = 1 CRd = 1 Clr = 0 OverFlow = 00
# Register Bank: Clock = 0 Out0 = 0 Out1 = 0 Out2 = 0 AdrIn = 6 Adr(out0/out1+1/out2+2) = 7 CWr = 1 CRd = 1 Clr = 1 OverFlow = 00
# Register Bank: Clock = 0 Out0 = 0 Out1 = 0 Out2 = 0 AdrIn = 6 Adr(out0/out1+1/out2+2) = 7 CWr = 1 CRd = 1 Clr = 0 OverFlow = 00
# Register Bank: Clock = 1 Out0 = 0 Out1 = 0 Out2 = 0 AdrIn = 6 Adr(out0/out1+1/out2+2) = 7 CWr = 1 CRd = 1 Clr = 0 OverFlow = 00
# Register Bank: Clock = 1 Out0 = 0 Out1 = 0 Out2 = 0 AdrIn = 7 Adr(out0/out1+1/out2+2) = 7 CWr = 1 CRd = 1 Clr = 0 OverFlow = 00
# Register Bank: Clock = 1 Out0 = 0 Out1 = 0 Out2 = 0 AdrIn = 7 Adr(out0/out1+1/out2+2) = 8 CWr = 1 CRd = 1 Clr = 0 OverFlow = 11
# Register Bank: Clock = 0 Out0 = 0 Out1 = 0 Out2 = 6 AdrIn = 7 Adr(out0/out1+1/out2+2) = 8 CWr = 1 CRd = 1 Clr = 0 OverFlow = 11
# Register Bank: Clock = 0 Out0 = 0 Out1 = 0 Out2 = 6 AdrIn = 8 Adr(out0/out1+1/out2+2) = 2 CWr = 1 CRd = 1 Clr = 0 OverFlow = 10
# Register Bank: Clock = 0 Out0 = 0 Out1 = 0 Out2 = 6 AdrIn = 8 Adr(out0/out1+1/out2+2) = 3 CWr = 1 CRd = 1 Clr = 0 OverFlow = 10
# Register Bank: Clock = 1 Out0 = 0 Out1 = 0 Out2 = 6 AdrIn = 8 Adr(out0/out1+1/out2+2) = 3 CWr = 1 CRd = 1 Clr = 0 OverFlow = 10
# Register Bank: Clock = 1 Out0 = 0 Out1 = 0 Out2 = 6 AdrIn = 8 Adr(out0/out1+1/out2+2) = 2 CWr = 1 CRd = 1 Clr = 0 OverFlow = 01
# Register Bank: Clock = 1 Out0 = 0 Out1 = 0 Out2 = 6 AdrIn = 9 Adr(out0/out1+1/out2+2) = 2 CWr = 1 CRd = 1 Clr = 0 OverFlow = 00
# Register Bank: Clock = 1 Out0 = 0 Out1 = 0 Out2 = 6 AdrIn = 9 Adr(out0/out1+1/out2+2) = 3 CWr = 1 CRd = 1 Clr = 0 OverFlow = 00
# Register Bank: Clock = 1 Out0 = 0 Out1 = 0 Out2 = 6 AdrIn = 9 Adr(out0/out1+1/out2+2) = 3 CWr = 0 CRd = 1 Clr = 0 OverFlow = 00
# Register Bank: Clock = 0 Out0 = 0 Out1 = 0 Out2 = 1 AdrIn = 9 Adr(out0/out1+1/out2+2) = 3 CWr = 1 CRd = 1 Clr = 0 OverFlow = 00
# Register Bank: Clock = 0 Out0 = 0 Out1 = 0 Out2 = 1 AdrIn = 10 Adr(out0/out1+1/out2+2) = 3 CWr = 1 CRd = 1 Clr = 0 OverFlow = 00
```

Figura 12: Simulação do banco de registradores, valor inicial igual à posição

Nesse caso o banco de registradores ja estava inicializado, o valor de cada posição era o indice da mesma, isso é perciptivel antes da primeira vez que clear é '1'.



```
# Register Bank: Clock = 1 Out0 = 2 Out1 = 2 Out2 = 2 AdrIn = 2 Adr(out0/out1+1/out2+2) = 1 CWr = 0 CRd = 1 Clr = 0 OverFlow = 10
# Register Bank: Clock = 0 Out0 = 2 Out1 = 2 Out2 = 2 AdrIn = 2 Adr(out0/out1+1/out2+2) = 1 CWr = 0 CRd = 1 Clr = 0 OverFlow = 10
# Register Bank: Clock = 1 Out0 = 2 Out1 = 2 Out2 = 2 AdrIn = 2 Adr(out0/out1+1/out2+2) = 1 CWr = 0 CRd = 1 Clr = 0 OverFlow = 10
# Register Bank: Clock = 0 Out0 = 2 Out1 = 2 Out2 = 2 AdrIn = 2 Adr(out0/out1+1/out2+2) = 1 CWr = 0 CRd = 1 Clr = 0 OverFlow = 10
# Register Bank: Clock = 1 Out0 = 2 Out1 = 2 Out2 = 2 AdrIn = 2 Adr(out0/out1+1/out2+2) = 1 CWr = 0 CRd = 1 Clr = 0 OverFlow = 10
# Register Bank: Clock = 0 Out0 = 2 Out1 = 2 Out2 = 2 AdrIn = 2 Adr(out0/out1+1/out2+2) = 1 CWr = 0 CRd = 1 Clr = 0 OverFlow = 10
# Register Bank: Clock = 1 Out0 = 2 Out1 = 2 Out2 = 2 AdrIn = 2 Adr(out0/out1+1/out2+2) = 1 CWr = 0 CRd = 1 Clr = 0 OverFlow = 10
# Register Bank: Clock = 0 Out0 = 2 Out1 = 2 Out2 = 2 AdrIn = 2 Adr(out0/out1+1/out2+2) = 1 CWr = 0 CRd = 1 Clr = 0 OverFlow = 10
# Register Bank: Clock = 1 Out0 = 2 Out1 = 2 Out2 = 2 AdrIn = 2 Adr(out0/out1+1/out2+2) = 1 CWr = 0 CRd = 1 Clr = 0 OverFlow = 10
# Register Bank: Clock = 0 Out0 = 2 Out1 = 2 Out2 = 2 AdrIn = 2 Adr(out0/out1+1/out2+2) = 1 CWr = 0 CRd = 1 Clr = 0 OverFlow = 10
# Register Bank: Clock = 1 Out0 = 2 Out1 = 2 Out2 = 2 AdrIn = 2 Adr(out0/out1+1/out2+2) = 1 CWr = 0 CRd = 1 Clr = 0 OverFlow = 00
# Register Bank: Clock = 0 Out0 = 2 Out1 = 2 Out2 = 2 AdrIn = 2 Adr(out0/out1+1/out2+2) = 1 CWr = 0 CRd = 1 Clr = 0 OverFlow = 00
# Register Bank: Clock = 1 Out0 = 2 Out1 = 2 Out2 = 2 AdrIn = 2 Adr(out0/out1+1/out2+2) = 1 CWr = 0 CRd = 1 Clr = 0 OverFlow = 00
# Register Bank: Clock = 0 Out0 = 2 Out1 = 2 Out2 = 2 AdrIn = 2 Adr(out0/out1+1/out2+2) = 1 CWr = 0 CRd = 1 Clr = 0 OverFlow = 00
# Register Bank: Clock = 1 Out0 = 2 Out1 = 2 Out2 = 2 AdrIn = 2 Adr(out0/out1+1/out2+2) = 1 CWr = 0 CRd = 1 Clr = 0 OverFlow = 00
# Register Bank: Clock = 0 Out0 = 2 Out1 = 2 Out2 = 2 AdrIn = 2 Adr(out0/out1+1/out2+2) = 1 CWr = 0 CRd = 1 Clr = 0 OverFlow = 11
# Register Bank: Clock = 1 Out0 = 2 Out1 = 2 Out2 = 2 AdrIn = 2 Adr(out0/out1+1/out2+2) = 1 CWr = 0 CRd = 1 Clr = 0 OverFlow = 11
# Register Bank: Clock = 0 Out0 = 3 Out1 = 3 Out2 = 3 AdrIn = 2 Adr(out0/out1+1/out2+2) = 1 CWr = 0 CRd = 1 Clr = 0 OverFlow = 11
# Register Bank: Clock = 1 Out0 = 3 Out1 = 3 Out2 = 3 AdrIn = 2 Adr(out0/out1+1/out2+2) = 1 CWr = 0 CRd = 1 Clr = 0 OverFlow = 11
# Register Bank: Clock = 0 Out0 = 3 Out1 = 3 Out2 = 3 AdrIn = 2 Adr(out0/out1+1/out2+2) = 1 CWr = 0 CRd = 1 Clr = 0 OverFlow = 11
# Register Bank: Clock = 1 Out0 = 3 Out1 = 3 Out2 = 3 AdrIn = 2 Adr(out0/out1+1/out2+2) = 1 CWr = 0 CRd = 1 Clr = 0 OverFlow = 11
# Register Bank: Clock = 0 Out0 = 3 Out1 = 3 Out2 = 3 AdrIn = 2 Adr(out0/out1+1/out2+2) = 1 CWr = 0 CRd = 1 Clr = 0 OverFlow = 11
# Register Bank: Clock = 1 Out0 = 3 Out1 = 3 Out2 = 3 AdrIn = 2 Adr(out0/out1+1/out2+2) = 1 CWr = 0 CRd = 1 Clr = 0 OverFlow = 11
# Register Bank: Clock = 0 Out0 = 3 Out1 = 3 Out2 = 3 AdrIn = 2 Adr(out0/out1+1/out2+2) = 1 CWr = 0 CRd = 1 Clr = 0 OverFlow = 11
# Register Bank: Clock = 1 Out0 = 3 Out1 = 3 Out2 = 3 AdrIn = 2 Adr(out0/out1+1/out2+2) = 1 CWr = 0 CRd = 1 Clr = 0 OverFlow = 11
# Register Bank: Clock = 0 Out0 = 3 Out1 = 3 Out2 = 3 AdrIn = 2 Adr(out0/out1+1/out2+2) = 1 CWr = 0 CRd = 1 Clr = 0 OverFlow = 00
# Register Bank: Clock = 1 Out0 = 3 Out1 = 3 Out2 = 3 AdrIn = 2 Adr(out0/out1+1/out2+2) = 1 CWr = 0 CRd = 1 Clr = 0 OverFlow = 00
# Register Bank: Clock = 0 Out0 = 3 Out1 = 3 Out2 = 3 AdrIn = 2 Adr(out0/out1+1/out2+2) = 1 CWr = 0 CRd = 1 Clr = 0 OverFlow = 00
# Register Bank: Clock = 1 Out0 = 3 Out1 = 3 Out2 = 3 AdrIn = 2 Adr(out0/out1+1/out2+2) = 1 CWr = 0 CRd = 1 Clr = 0 OverFlow = 00
# Register Bank: Clock = 0 Out0 = 3 Out1 = 3 Out2 = 3 AdrIn = 2 Adr(out0/out1+1/out2+2) = 1 CWr = 0 CRd = 1 Clr = 0 OverFlow = 00
```

Figura 13: Simulação do banco registradores em casos de overflow, endereço de saida fixo em '1'

Esse caso testa as condições de overflow, fazendo um OR logico com o registrador[1](flg, ou, flag) com o valor de overflow, e funciona corretamente.

- Memória

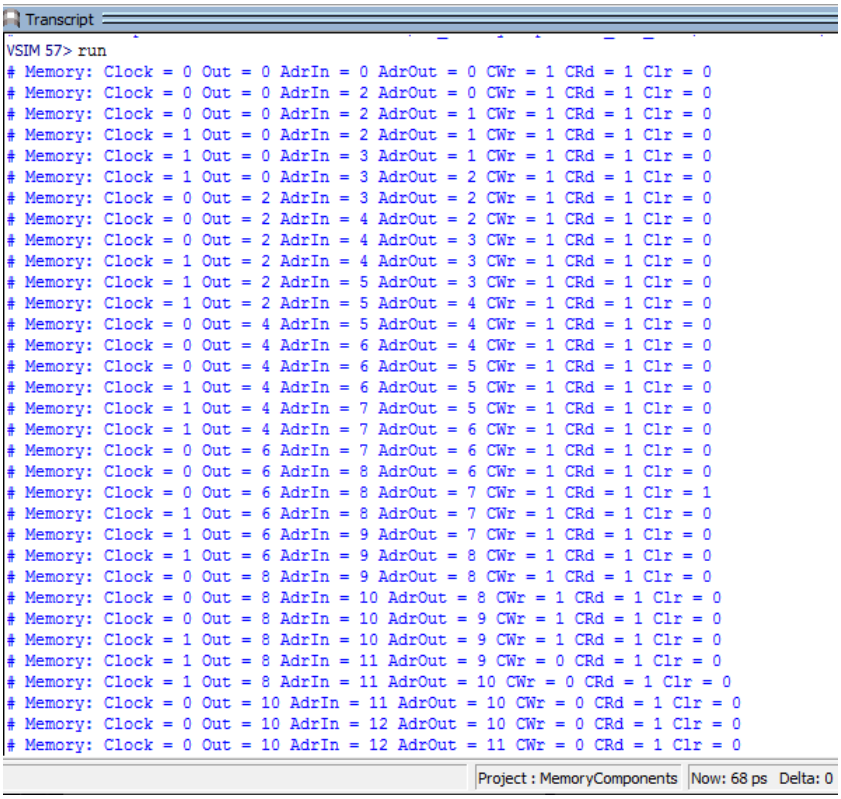


Figura 14: Simulação do da memória

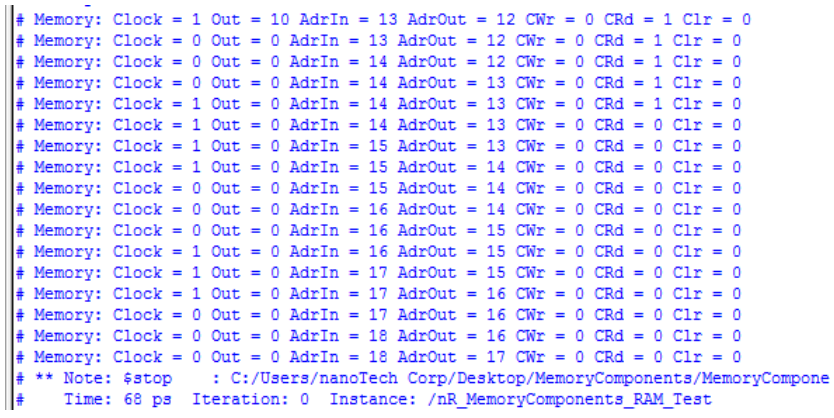


Figura 15: Simulação do da memória

Quando o clock tem borda de subida, o valor no endereco de entradada(que é o mesmo da entrada nesse caso) é gravado, quando temos borda de subida

o valor é lido corretamente. Podemos ver tambem que o controle de leitura e escrita tambem funciona, permitindo ou nao leitura e escrita. O sinal assincrono de clear também está funcionado, zerando os valores quando tem nivel logico alto.

12 Compilador

o código do compilador sofreu diversas alterações, o segmento de dados estava sendo interpretado e anexado corretamente, agora, com maior conhecimento sobre o funcionamento do mesmo do mesmo tudo foi corrigido e funciona corretamente. Também foram adicionadas várias flags de compilalção que tem seu funcionamento descrito abaixo:

12.1 Flags de compilador

- -o
  Define o nome do arquivo de saída compilado, por padrão o nome é "a.out"
- -s
  Separa a saída em dois arquivos compilados, deixando um arquivo para instruções e outro para dados.
- -c
  Adiciona comentarios no arquivo compilado indicando onde começa cada segmento da memoria
- -v
  Após essa flag é passado o nome da variavél para dados e o nome da variável para instruções, então o compilador gera um código verilog atribuindo a essas variaveis o programa, para simplemente colar no verilog
- -b
  Compila para um arquivo binário

12.2 Novo código do compilador

```
#include <iostream>
#include <bitset>
#include <algorithm>
#include <fstream>
#include <sstream>
#include <string>
#include <vector>
#include <map>

using namespace std;

string output = "a.out";
string input;
map<string,int> addresses;

int nRisk_StackSigal=-1;
//funcs
string nRisk_slp="0000";
string nRisk_brq="0001";
string nRisk_brf="0010";
string nRisk_add="0011";
string nRisk_sub="0100";
string nRisk_and="0101";
string nRisk_or ="0110";
string nRisk_nor="0111";
```

```
26 string nRisk_slt="1000";
27 string nRisk_sr ="1001";
28 string nRisk_sl ="1010";
29 string nRisk_jr ="1011";
30 string nRisk_la ="1100";
31 string nRisk_lc ="1101";
32 string nRisk_lw ="1110";
33 string nRisk_sw ="1111";
34 //regs
35 string nRisk_zro="0000";
36 string nRisk_flg="0001";
37 string nRisk_nt ="0010";
38 string nRisk_sp ="0011";
39 string nRisk_ra ="0100";
40 string nRisk_v0 ="0101";
41 string nRisk_v1 ="0110";
42 string nRisk_a0 ="0111";
43 string nRisk_a1 ="1000";
44 string nRisk_a2 ="1001";
45 string nRisk_a3 ="1010";
46 string nRisk_s0 ="1011";
47 string nRisk_s1 ="1100";
48 string nRisk_s2 ="1101";
49 string nRisk_s3 ="1110";
50 string nRisk_bp ="1111";
51
52 void error(string str){
53     cout<<"error - "<<str<<"."<<endl;
54 }
55
56 struct nanoArg{
57     string arg;
58     bool isRegister;
59     int pos;
60     bool err=false;
61 };
62
63 int strToInt(string str){
64     str.erase(remove(str.begin(),str.end(),','),str.end());//remove
          $
65     stringstream ss;
66     ss.str(str);
67     int out;
68     ss>>out;
69     return out;
70 }
71
72 bool isValidChar(char c){
73     return c=='$'||(c>=48&&c<=57)||(c>=65&&c<=90)||(c>=97&&c<=122);
74 }
75
76 bool isBinary(string str){
77     for(int i=0;i<str.size();i++){
78         if(str[i]!='0'&&str[i]!='1'){
79             return false;
80         }
81     }
82     return true;
83 }
84 nanoArg getArgFrom(string str, int from){
85     nanoArg out;
86     int start=-1;
```

```
87      for(int j=from;j<=str.size();j++){
88          if(start<0&&isValidChar(str[j])){
89              start=j;
90          }else if(start>=0){
91              if(str[j]==',',||str[j]=='\0'||str[j]=='\t'||str[j]=='('
                    ||str[j]==')'||str[j]==' '){
92                  out.pos=j+1;
93                  out.arg=str.substr(start,j-start);
94                  out.isRegister=out.arg.find("$")!=string::npos;
95                  return out;
96              }
97          }
98      }
99      out.err=true;
100     return out;
101 }
102 string translateRegister(string arg){
103     arg.erase(remove(arg.begin(),arg.end(),'$'),arg.end());//remove
            $
104     if(!arg.compare("zero")||!arg.compare("0")){
105         return nRisk_zro;
106     }else if(!arg.compare("flg")||!arg.compare("flag")){
107         return nRisk_flg;
108     }else if(!arg.compare("nt")||!arg.compare("nT")){
109         return nRisk_nt;
110     }else if(!arg.compare("sp")){
111         return nRisk_sp;
112     }else if(!arg.compare("ra")){
113         return nRisk_ra;
114     }else if(!arg.compare("v0")){
115         return nRisk_v0;
116     }else if(!arg.compare("v1")){
117         return nRisk_v1;
118     }else if(!arg.compare("a0")){
119         return nRisk_a0;
120     }else if(!arg.compare("a1")){
121         return nRisk_a1;
122     }else if(!arg.compare("a2")){
123         return nRisk_a2;
124     }else if(!arg.compare("a3")){
125         return nRisk_a3;
126     }else if(!arg.compare("s0")){
127         return nRisk_s0;
128     }else if(!arg.compare("s1")){
129         return nRisk_s1;
130     }else if(!arg.compare("s2")){
131         return nRisk_s2;
132     }else if(!arg.compare("s3")){
133         return nRisk_s3;
134     }else if(!arg.compare("bp")){
135         return nRisk_bp;
136     } else return "";
137 }
138 vector<string> translateFunc(string func,string args){
139     func.erase(remove(func.begin(),func.end(),' '),func.end());//
            remove spaces
140     args.erase(remove(args.begin(),args.end(),' '),args.end());//
            remove spaces
141     func.erase(remove(func.begin(),func.end(),'\t'),func.end());//
            remove spaces
142     args.erase(remove(args.begin(),args.end(),'\t'),args.end());//
            remove spaces
```

```
143     vector<string> out;
144     nanoArg arg0,arg1,arg2;
145     string nRisk,argRegister,target0,target1,target2;
146     if(!func.compare("slp")||!func.compare("sleep")){
147         arg0=getArgFrom(args,0);
148         target0="0001";
149         if(!arg0.err){
150             if(arg0.isRegister){
151                 argRegister=translateRegister(arg0.arg);
152                 if(argRegister!="")
153                     target0=argRegister;
154                 else{
155                     error("Register \""+arg0.arg+"\" doesnt exists"
                            );
156                     return out;
157                 }
158             }
159         }
160         out.push_back(nRisk_slp+target0);
161         out.push_back("00000000");
162         return out;
163     }else if(!func.compare("brq")||!func.compare("brf")||!func.
            compare("add")||!func.compare("sub")||!func.compare("and")
164              ||!func.compare("or")||!func.compare("nor")||!func.
                 compare("slt")||!func.compare("sr")||!func.compare
                 ("sl")){
165         arg0=getArgFrom(args,0);
166         if(arg0.err){
167             error("Cannot find arg0 for \""+func+"\"");
168             return out;
169         }
170         if(!arg0.isRegister){
171             error("Arg0 for \""+func+"\" is not a register");
172             return out;
173         }
174         arg1=getArgFrom(args,arg0.pos);
175         if(arg1.err){
176             error("Cannot find arg1 for \""+func+"\"");
177             return out;
178         }
179         if(!arg1.isRegister){
180             error("Arg1 for \""+func+"\" is not a register");
181             return out;
182         }
183         arg2=getArgFrom(args,arg1.pos);
184         if(arg2.err){
185             error("Cannot find arg2 for \""+func+"\"");
186             return out;
187         }
188         if(!arg2.isRegister){
189             error("Arg2 for \""+func+"\" is not a register");
190             return out;
191         }
192         argRegister=translateRegister(arg0.arg);
193         if(argRegister==""){
194             error("Register \""+arg0.arg+"\" doesnt exists for \""+
                    func+"\"");
195             return out;
196         }
197         target0=argRegister;
198         argRegister=translateRegister(arg1.arg);
199         if(argRegister==""){
```

```
200             error("Register \""+arg1.arg+"\" doesnt exists for \""+
                    func+"\"");
201             return out;
202         }
203         target1=argRegister;
204         argRegister=translateRegister(arg2.arg);
205         if(argRegister==""){
206             error("Register \""+arg2.arg+"\" doesnt exists for \""+
                    func+"\"");
207             return out;
208         }
209         target2=argRegister;
210 
211         if(!func.compare("brq"))
212             nRisk=nRisk_brq;
213         else if(!func.compare("brf"))
214             nRisk=nRisk_brf;
215         else if(!func.compare("add"))
216             nRisk=nRisk_add;
217         else if(!func.compare("sub"))
218             nRisk=nRisk_sub;
219         else if(!func.compare("and"))
220             nRisk=nRisk_and;
221         else if(!func.compare("or"))
222             nRisk=nRisk_or;
223         else if(!func.compare("nor"))
224             nRisk=nRisk_nor;
225         else if(!func.compare("slt"))
226             nRisk=nRisk_slt;
227         else if(!func.compare("sr"))
228             nRisk=nRisk_sr;
229         else if(!func.compare("sl"))
230             nRisk=nRisk_sl;
231 
232         out.push_back(nRisk+target0);
233         out.push_back(target1+target2);
234         return out;
235     }else if(!func.compare("jr")){
236         arg0=getArgFrom(args,0);
237         if(arg0.err){
238             error("Cannot find arg0 for \""+func+"\"");
239             return out;
240         }
241         if(!arg0.isRegister){
242             error("Arg0 for \""+func+"\" is not a register");
243             return out;
244         }
245         argRegister=translateRegister(arg0.arg);
246         if(argRegister==""){
247             error("Register \""+arg0.arg+"\" doesnt exists for \""+
                    func+"\"");
248             return out;
249         }
250         target0=argRegister;
251         out.push_back(nRisk_jr+target0);
252         out.push_back("00000000");
253         return out;
254     }else if(!func.compare("j")){//SEMI INTERPRETADA
255         arg0=getArgFrom(args,0);
256         if(arg0.err){
257             error("Cannot find arg0 for \""+func+"\"");
258             return out;
```

```
259         }
260         if(arg0.isRegister){
261             error("Arg0 for \""+func+"\" is a register");
262             return out;
263         }
264         out.push_back(nRisk_jr+"0000");
265         out.push_back(arg0.arg);
266         return out;
267     }else if(!func.compare("la")||!func.compare("lc")){
268         arg0=getArgFrom(args,0);
269         if(arg0.err){
270             error("Cannot find arg0 for \""+func+"\"");
271             return out;
272         }
273         if(!arg0.isRegister){
274             error("Arg0 for \""+func+"\" is not a register");
275             return out;
276         }
277         argRegister=translateRegister(arg0.arg);
278         if(argRegister==""){
279             error("Register \""+arg0.arg+"\" doesnt exists for \""+
                    func+"\"");
280             return out;
281         }
282         target0=argRegister;
283         arg1=getArgFrom(args,arg0.pos);
284         if(arg1.err){
285             error("Cannot find arg1 for \""+func+"\"");
286             return out;
287         }
288         if(arg1.isRegister){
289             error("Arg1 for \""+func+"\" is a register");
290             return out;
291         }
292         if(!func.compare("la")){
293             out.push_back(nRisk_la+target0);
294             out.push_back(arg1.arg);
295         }else if(!func.compare("lc")){
296             if(strToInt(arg1.arg)>255){
297                 error("Constant overflow, your constant should be
                        less than 256");
298                 return out;
299             }
300             out.push_back(nRisk_lc+target0);
301             out.push_back(bitset< 8 >(strToInt(arg1.arg)).to_string
                    ());
302         }
303         return out;
304     }else if(!func.compare("lw")||!func.compare("sw")){
305         arg0=getArgFrom(args,0);
306         if(arg0.err){
307             error("Cannot find arg0 for \""+func+"\"");
308             return out;
309         }
310         if(!arg0.isRegister){
311             error("Arg0 for \""+func+"\" is not a register");
312             return out;
313         }
314         argRegister=translateRegister(arg0.arg);
315         if(argRegister==""){
316             error("Register \""+arg0.arg+"\" doesnt exists for \""+
                    func+"\"");
```

```
317             return out;
318         }
319         target0=argRegister;
320         arg1=getArgFrom(args,arg0.pos);
321         if(arg1.err){
322             error("Cannot find arg1 for \""+func+"\"");
323             return out;
324         }
325         if(arg1.isRegister){
326             error("Arg1 for \""+func+"\" is a register");
327             return out;
328         }
329         arg2=getArgFrom(args,arg1.pos);
330         if(arg2.err){
331             error("Cannot find arg2 for \""+func+"\"");
332             return out;
333         }
334         if(!arg2.isRegister){
335             error("Arg2 for \""+func+"\" is not a register");
336             return out;
337         }
338         argRegister=translateRegister(arg2.arg);
339         if(argRegister==""){
340             error("Register \""+arg2.arg+"\" doesnt exists for \""+
                    func+"\"");
341             return out;
342         }
343         target2=argRegister;
344 
345         if(!func.compare("lw"))
346             nRisk=nRisk_lw;
347         else if(!func.compare("sw"))
348             nRisk=nRisk_sw;
349         if(strToInt(arg1.arg)>15){
350             error("Offset \""+arg1.arg+"\" for \""+func+"\" should
                    be less than 16");
351             return out;
352         }
353         out.push_back(nRisk+target0);
354         out.push_back(bitset< 4 >(strToInt(arg1.arg)).to_string()+
                target2);
355         return out;
356     }//INTERPRETADAS
357     else if(!func.compare("addc")||!func.compare("andc")||!func.
            compare("orc")||!func.compare("norc")||
358             !func.compare("sltc")||!func.compare("src")||!func.
                    compare("slc")||!func.compare("beq")||
359             !func.compare("bof")){
360 
361         arg0=getArgFrom(args,0);
362         if(arg0.err){
363             error("Cannot find arg0 for \""+func+"\"");
364             return out;
365         }
366         if(!arg0.isRegister){
367             error("Arg0 for \""+func+"\" is not a register");
368             return out;
369         }
370         arg1=getArgFrom(args,arg0.pos);
371         if(arg1.err){
372             error("Cannot find arg1 for \""+func+"\"");
373             return out;
```

```
374         }
375         if(!arg1.isRegister){
376             error("Arg1 for \""+func+"\" is not a register");
377             return out;
378         }
379         arg2=getArgFrom(args,arg1.pos);
380         if(arg2.err){
381             error("Cannot find arg2 for \""+func+"\"");
382             return out;
383         }
384         if(arg2.isRegister){
385             error("Arg2 for \""+func+"\" is a register");
386             return out;
387         }
388         argRegister=translateRegister(arg0.arg);
389         if(argRegister==""){
390             error("Register \""+arg0.arg+"\" doesnt exists for \""+
                    func+"\"");
391             return out;
392         }
393         target0=argRegister;
394         argRegister=translateRegister(arg1.arg);
395         if(argRegister==""){
396             error("Register \""+arg1.arg+"\" doesnt exists for \""+
                    func+"\"");
397             return out;
398         }
399         target1=argRegister;
400         if(func.compare("beq")&&func.compare("bof"))
401         if(strToInt(arg2.arg)>255){
402             error("Constant overflow, your constant should be less
                    than 256");
403             return out;
404         }
405         out.push_back(nRisk_lc+nRisk_nt);
406         if(!func.compare("beq")||!func.compare("bof")){
407             out.push_back(arg2.arg);
408         }else{
409             out.push_back(bitset< 8 >(strToInt(arg2.arg)).to_string
                    ());
410         }
411         if(!func.compare("addc"))
412             nRisk=nRisk_add;
413         else if(!func.compare("andc"))
414             nRisk=nRisk_and;
415         else if(!func.compare("orc"))
416             nRisk=nRisk_or;
417         else if(!func.compare("norc"))
418             nRisk=nRisk_nor;
419         else if(!func.compare("sltc"))
420             nRisk=nRisk_slt;
421         else if(!func.compare("src"))
422             nRisk=nRisk_sr;
423         else if(!func.compare("slc"))
424             nRisk=nRisk_sl;
425         else if(!func.compare("beq"))
426             nRisk=nRisk_brq;
427         else if(!func.compare("bof"))
428             nRisk=nRisk_brf;
429         out.push_back(nRisk+target0);
430         out.push_back(target1+nRisk_nt);
431         return out;
```

```
432     }else if(!func.compare("mov")||!func.compare("move")||!func.
            compare("not")){
433         arg0=getArgFrom(args,0);
434         if(arg0.err){
435             error("Cannot find arg0 for \""+func+"\"");
436             return out;
437         }
438         if(!arg0.isRegister){
439             error("Arg0 for \""+func+"\" is not a register");
440             return out;
441         }
442         arg1=getArgFrom(args,arg0.pos);
443         if(arg1.err){
444             error("Cannot find arg1 for \""+func+"\"");
445             return out;
446         }
447         if(!arg1.isRegister){
448             error("Arg1 for \""+func+"\" is not a register");
449             return out;
450         }
451         argRegister=translateRegister(arg0.arg);
452         if(argRegister==""){
453             error("Register \""+arg0.arg+"\" doesnt exists for \""+
                    func+"\"");
454             return out;
455         }
456         target0=argRegister;
457         argRegister=translateRegister(arg1.arg);
458         if(argRegister==""){
459             error("Register \""+arg1.arg+"\" doesnt exists for \""+
                    func+"\"");
460             return out;
461         }
462         target1=argRegister;
463 
464         if(!func.compare("mov")||!func.compare("move"))
465             nRisk=nRisk_add;
466         else if(!func.compare("not"))
467             nRisk=nRisk_nor;
468 
469 
470         out.push_back(nRisk+target0);
471         out.push_back(target1+nRisk_zro);
472         return out;
473 
474     }else if(!func.compare("jal")||!func.compare("jrl")){
475         if(!func.compare("jal")){
476             arg0=getArgFrom(args,0);
477             if(arg0.err){
478                 error("Cannot find arg0 for \""+func+"\"");
479                 return out;
480             }
481             if(arg0.isRegister){
482                 error("Arg0 for \""+func+"\" is a register");
483                 return out;
484             }
485             target0=nRisk_nt;
486             out.push_back(nRisk_lc+target0);
487             out.push_back(arg0.arg);
488         }else{
489             arg0=getArgFrom(args,0);
490             if(arg0.err){
```

```
                error("Cannot find arg0 for \""+func+"\"");
                return out;
            }
            if(!arg0.isRegister){
                error("Arg0 for \""+func+"\" is not a register");
                return out;
            }
            argRegister=translateRegister(arg0.arg);
            if(argRegister==""){
                error("Register \""+arg0.arg+"\" doesnt exists for
                    \""+func+"\"");
                return out;
            }
            target0=argRegister;
        }
        out.push_back(nRisk_jr+target0);
        out.push_back("00000000");
        out.push_back(nRisk_add+nRisk_ra);
        out.push_back(nRisk_nt+nRisk_zro);
        return out;
    }else if(!func.compare("push")||!func.compare("pop")){
        arg0=getArgFrom(args,0);
        if(arg0.err){
            error("Cannot find arg0 for \""+func+"\"");
            return out;
        }
        if(!arg0.isRegister){
            error("Arg0 for \""+func+"\" is not a register");
            return out;
        }
        argRegister=translateRegister(arg0.arg);
        if(argRegister==""){
            error("Register \""+arg0.arg+"\" doesnt exists for \""+
                func+"\"");
            return out;
        }
        target0=argRegister;
        if(nRisk_StackSigal>0){
            if(!func.compare("push")){
                //escreve soma
                out.push_back(nRisk_sw+target0);
                out.push_back("0000"+nRisk_sp);
                out.push_back(nRisk_lc+nRisk_nt);
                out.push_back(bitset< 8 >(1).to_string());
                out.push_back(nRisk_add+nRisk_sp);
                out.push_back(nRisk_sp+nRisk_nt);
            }else{
                //subtrai le
                out.push_back(nRisk_lc+nRisk_nt);
                out.push_back(bitset< 8 >(1).to_string());
                out.push_back(nRisk_sub+nRisk_sp);
                out.push_back(nRisk_sp+nRisk_nt);
                out.push_back(nRisk_lw+target0);
                out.push_back("0000"+nRisk_sp);
            }
        }else{
            if(!func.compare("push")){
                //subtrai escreve
                out.push_back(nRisk_lc+nRisk_nt);
                out.push_back(bitset< 8 >(1).to_string());
                out.push_back(nRisk_sub+nRisk_sp);
                out.push_back(nRisk_sp+nRisk_nt);
```

```
                out.push_back(nRisk_sw+target0);
                out.push_back("0000"+nRisk_sp);
            }else{
                //le soma
                out.push_back(nRisk_lw+target0);
                out.push_back("0000"+nRisk_sp);
                out.push_back(nRisk_lc+nRisk_nt);
                out.push_back(bitset< 8 >(1).to_string());
                out.push_back(nRisk_add+nRisk_sp);
                out.push_back(nRisk_sp+nRisk_nt);
            }
        }
        return out;
    }else{
        error("Function \""+func+"\" doesnt exists");
        return out;
    }
}

char strToChar(string a){
    char out=0;
    int pow2[8]={1,2,4,8,16,32,64,128};
    if(a.size()<8)
        return 0;
    for(int i=0;i<8;i++)
        out+=pow2[7-i]*a[i];
    return out;
}

string appendName(string a, string b){
    string p1="";
    string p2="";
    bool dot=false;
    for(int i=0;i<a.size();i++){
        if(a[i]!='.'&&!dot)
            p1+=a[i];
        else{
            dot=true;
            p2+=a[i];
        }
    }
    return p1+b+p2;
}

int main(int argc, char **argv){
    bool separateFiles=false;
    bool coment=false;
    bool verilog=false;
    bool binary=false;
    string verilog0,verilog1;
    if(argc<2){
        error("No such file or arguments");
        return -1;
    }
    input=argv[1];
    for(int i=2;i<argc;i++){
        if(argv[i][0]=='-'&&tolower(argv[i][1])=='o'){
            if(i+1>=argc){
                error("Few arguments for -o");
                return -1;
            }
            output=argv[++i];
```

```
        }
        if(argv[i][0]=='-'&&tolower(argv[i][1])=='s'){
            separateFiles=true;
        }
        if(argv[i][0]=='-'&&tolower(argv[i][1])=='c'){
            coment=true;
        }
        if(argv[i][0]=='-'&&tolower(argv[i][1])=='b'){
            binary=true;
        }
        if(argv[i][0]=='-'&&tolower(argv[i][1])=='v'){
            if(i+2>=argc){
                error("Few arguments for -v");
                return -1;
            }
            verilog=true;
            verilog0=argv[++i];
            verilog1=argv[++i];
        }
    }
    string line;
    vector<string> code;
    ifstream codeFile (input);
    if (codeFile.is_open()){
        while (getline (codeFile,line)){
            code.push_back(line);
        }
        codeFile.close();
    }else{
        error("Unable to reach file");
        return -1;
    }
    vector<string> machineCode;
    vector<string> dataSeg;
    string dataSegSize;
    bool data=false;
    bool text=false;
    int commentedLines=0;
    for(int i=0;i<code.size();i++){
        if(code[i][0]=='#'||code[i][0]=='\n'){
            code.erase(code.begin()+i);
            i--;
            commentedLines++;
        }else{
            int foundAt=code[i].find("#");//size
            if(foundAt!=string::npos){
                code[i].erase(code[i].begin()+foundAt,code[i].end()
                    );
            }
        }
    }
    int currentAddress=0;
    int currentDataAddress=0;
    int doubleDotAt;
    for(int i=0;i<code.size();i++){
        if(code[i].find(".data")!=string::npos){
            data=true;
        }else if(code[i].find(".text")!=string::npos){
            text=true;
        }else if(data&&!text){//DATA PART
            doubleDotAt=code[i].find(":");//size
            if(doubleDotAt==string::npos)
```

```
674                 doubleDotAt=0;
675             else{
676                 string tmp=code[i].substr(0,doubleDotAt);
677                 code[i]=code[i].substr(doubleDotAt+1);
678                 tmp.erase(remove(tmp.begin(),tmp.end(),' '),tmp.end
                        ());
679                 tmp.erase(remove(tmp.begin(),tmp.end(),'\t'),tmp.
                        end());
680                 addresses[tmp]=currentDataAddress;
681             }
682             code[i]=code[i].substr(code[i].find("."));
683             string varType=code[i].substr(0,code[i].find(" "));
684             if(!varType.compare(".word")){
685                 code[i]=code[i].substr(code[i].find(" "));
686                 code[i].erase(remove(code[i].begin(),code[i].end(),
                        ' '),code[i].end());//remove spaces
687                 code[i].erase(remove(code[i].begin(),code[i].end(),
                        '\t'),code[i].end());//remove spaces
688                 int last=0;
689                 for(int j=0;j<=code[i].size();j++){
690                     if(code[i][j]==','||code[i][j]=='\0'){
691                         dataSeg.push_back(bitset< 8 >(strToInt(code
                                [i].substr(last,j-last))).to_string());
692                         last=j;
693                         currentDataAddress++;
694                     }
695                 }
696             }
697         }else if(text){//INSTRUCTIONS PART
698             doubleDotAt=code[i].find(":");//size
699             if(doubleDotAt==string::npos)
700                 doubleDotAt=0;
701             else{
702                 string tmp=code[i].substr(0,doubleDotAt);
703                 tmp.erase(remove(tmp.begin(),tmp.end(),' '),tmp.end
                        ());
704                 tmp.erase(remove(tmp.begin(),tmp.end(),'\t'),tmp.
                        end());
705                 addresses[tmp]=currentAddress;
706             }
707             int start=-1;
708             for(int j=doubleDotAt;j<=code[i].size();j++){
709                 if(start<0&&isValidChar(code[i][j])){
710                     start=j;
711                 }else if(start>=0){
712                     if(code[i][j]==' '||code[i][j]=='\0'||code[i][j
                            ]=='\t'){
713                         vector<string> translated=translateFunc(
                                code[i].substr(start,j-start),code[i].
                                substr(j+1));
714                         if(translated.size()!=0){
715                             for(int t=0;t<translated.size();t++){
716                                 machineCode.push_back(translated[t
                                        ]);
717                                 currentAddress++;
718                             }
719                         }else{
720                             error("Translating line ("+to_string(i+
                                    commentedLines)+")");
721                             return -1;
722                         }
723                          break;
```

```
724                     }
725                 }
726             }
727 
728         }
729     }
730     for(int i=1;i<machineCode.size();i+=2){
731         if(machineCode[i].size()!=8||!isBinary(machineCode[i])){
732             if(addresses.find(machineCode[i]) == addresses.end()){
733                 error("Cannot jump to \""+machineCode[i]+"\", tag
                        doesnt exists");
734                 return -1;
735             }else{
736                 if(addresses[machineCode[i]]>255){
737                     error("Address overflow("+to_string(addresses[
                            machineCode[i]])+"), your code is too big")
                            ;
738                     return -1;
739                 }
740                 machineCode[i]=bitset< 8 >(addresses[machineCode[i
                        ]]).to_string();
741             }
742         }
743     }
744     if(binary){
745         if(separateFiles){
746             ofstream dataFile(appendName(output,"_data"),ios::
                    binary|ios::out);
747             if(dataFile.is_open()){
748                 for(int i=0;i<dataSeg.size();i++){
749                     char c=strToChar(dataSeg[i]);
750                     dataFile.write(&c, sizeof(c));
751                 }
752                 dataFile.close();
753             }else{
754                 error("Unable to create file");
755                 return -1;
756             }
757             ofstream machineFile(appendName(output,"_inst"),ios::
                    binary|ios::out);
758             if(machineFile.is_open()){
759                 for(int i=0;i<machineCode.size();i++){
760                     char c=strToChar(machineCode[i]);
761                     machineFile.write(&c, sizeof(c));
762                 }
763                 machineFile.close();
764             }else{
765                 error("Unable to create file");
766                 return -1;
767             }
768         }else{
769             ofstream outFile(output,ios::binary|ios::out);
770             if(outFile.is_open()){
771                 for(int i=0;i<dataSeg.size();i++){
772                     char c=strToChar(dataSeg[i]);
773                     outFile.write(&c, sizeof(c));
774                 }
775                 for(int i=0;i<machineCode.size();i++){
776                     char c=strToChar(machineCode[i]);
777                     outFile.write(&c, sizeof(c));
778                 }
779                 outFile.close();
```

```
780             }else{
781                 error("Unable to create file");
782                 return -1;
783             }
784         }
785     }else{
786         if(verilog){
787             if(separateFiles){
788                 ofstream dataFile (appendName(output,"_data"));
789                 if(dataFile.is_open()){
790                     if(coment)
791                         dataFile<<"//data\n";
792                     for(int i=0;i<dataSeg.size();i++)
793                         dataFile<<verilog0<<"["<<i<<"]<=8\'b"<<
                                dataSeg[i]<<";"<<endl;
794                     dataFile.close();
795                 }else{
796                     error("Unable to create file");
797                     return -1;
798                 }
799                 ofstream machineFile (appendName(output,"_inst"));
800                 if(machineFile.is_open()){
801                     if(coment)
802                         machineFile<<"//instructions\n";
803                     for(int i=0;i<machineCode.size();i++)
804                         machineFile<<verilog1<<"["<<(i)<<"]<=8\'b"
                                <<machineCode[i]<<";"<<endl;
805                     machineFile.close();
806                 }else{
807                     error("Unable to create file");
808                     return -1;
809                 }
810             }else{
811                 ofstream outFile (output);
812                 if(outFile.is_open()){
813                     if(coment)
814                         outFile<<"//data\n";
815                     for(int i=0;i<dataSeg.size();i++)
816                         outFile<<verilog0<<"["<<i<<"]<=8\'b"<<
                                dataSeg[i]<<";"<<endl;
817                     if(coment)
818                         outFile<<"//instructions\n";
819                     int offset=0;
820                     if(verilog0==verilog1)
821                         offset=dataSeg.size();
822                     for(int i=0;i<machineCode.size();i++)
823                         outFile<<verilog1<<"["<<(i+offset)<<"]<=8\'
                                b"<<machineCode[i]<<";"<<endl;
824                     outFile.close();
825                 }else{
826                     error("Unable to create file");
827                     return -1;
828                 }
829             }
830         }else{
831             if(separateFiles){
832                 if(coment){
833                     machineCode[0]+="//instructions\n";
834                     dataSeg[0]+="//data\n";
835                 }
836                 ofstream dataFile (appendName(output,"_data"));
837                 if(dataFile.is_open()){
```

```
838                     for(int i=0;i<dataSeg.size();i++){
839                         dataFile<<dataSeg[i]<<" ";
840                         if((i+1)%2==0&&i+1<dataSeg.size())
841                             dataFile<<endl;
842                     }
843                     dataFile.close();
844                 }else{
845                     error("Unable to create file");
846                     return -1;
847                 }
848 
849                 ofstream machineFile (appendName(output,"_inst"));
850                 if(machineFile.is_open()){
851                     for(int i=0;i<machineCode.size();i++){
852                         machineFile<<machineCode[i]<<" ";
853                         if((i+1)%2==0&&i+1<machineCode.size())
854                             machineFile<<endl;
855                     }
856                     machineFile.close();
857                 }else{
858                     error("Unable to create file");
859                     return -1;
860                 }
861             }else{
862                 dataSegSize=bitset< 8 >(dataSeg.size()).to_string()
                        ;
863                 if(coment){
864                     dataSegSize+="//data seg size\n";
865                     machineCode[0]+="//instructions\n";
866                     dataSeg[0]+="//data\n";
867                 }
868                 for(int i=0;i<dataSeg.size();i++){
869                     machineCode.insert(machineCode.begin()+i,
                            dataSeg[i]);
870                 }
871                 machineCode.insert(machineCode.begin(), dataSegSize
                        );
872                 ofstream outFile (output);
873                 if(outFile.is_open()){
874                     for(int i=0;i<machineCode.size();i++){
875                         outFile<<machineCode[i]<<" ";
876                         if((i+1)%2==0&&i+1<machineCode.size())
877                             outFile<<endl;
878                     }
879                     outFile.close();
880                 }else{
881                     error("Unable to create file");
882                     return -1;
883                 }
884             }
885         }
886     }
887     return 0;
888 }
```

13 Controle

O controle define os bits de controle para cada instrução, abaixo podemos ver o código verilog desse componente, acima do código há um comentario explicando suas entradas e saídas.

```
//Modulo de Controle de instrucoes:
//Entradas:
//        clk: clocks
//        inst: codigo da instrucao
//Saidas:
//        tmpwr: Bit de controle escrita temporaria
//        hlt: Bit de controle sleep
//        jmp: Bit de controle de desvio incondicional
//        brc: Bit de controle de desvio condicional
//        rgw: Bits de controle de escrita em registrador
//        ioc: Bits de controle de I/O de dados(banco e memoria)
//        rgr: Bit de controle de leitura em registrador
//        alo: Bits de controle da ALU
//        ala: Bit de controle de entrada da ALU
module nR_Control(clk,inst,tmpwr,hlt,jmp,brc,rgw,ioc,rgr,alo,ala);
     input clk;
     input[3:0]inst;
     output reg tmpwr,hlt,jmp,brc,rgr,ala;
     output reg[1:0] rgw;
     output reg[3:0] ioc,alo;

     initial
     begin
          tmpwr<=0;
          hlt<=1;
          jmp<=0;
          brc<=0;
          rgw<=0;
          ioc<=0;
          rgr<=0;
          alo<=0;
          ala<=0;
     end

     always @ (posedge clk) begin
          case (inst)
            4'b0000: begin//slp
               hlt<=0;
               jmp<=0;
               brc<=0;
               ioc<=4'b0100;
               rgr<=0;
               end
            4'b0001: begin//brq
               hlt<=1;
               jmp<=0;
               brc<=1;
               ioc<=4'b0100;
               rgr<=0;
               alo<=4'b0001;
               ala<=0;
               end
            4'b0010: begin//brf
               tmpwr<=1;
               hlt<=1;
               jmp<=0;
               brc<=1;
               rgw<=2'b00;
               ioc<=4'b1100;
               rgr<=0;
               alo<=4'b0010;
               ala<=0;
```

```
    end
4'b0011: begin//add
    tmpwr<=1;
    hlt<=1;
    jmp<=0;
    brc<=0;
    rgw<=2'b00;
    ioc<=4'b1100;
    rgr<=1;
    alo<=4'b0000;
    ala<=0;
    end
4'b0100: begin//sub
    tmpwr<=1;
    hlt<=1;
    jmp<=0;
    brc<=0;
    rgw<=2'b00;
    ioc<=4'b1100;
    rgr<=1;
    alo<=4'b0001;
    ala<=0;
    end
4'b0101: begin//and
    tmpwr<=1;
    hlt<=1;
    jmp<=0;
    brc<=0;
    rgw<=2'b00;
    ioc<=4'b1100;
    rgr<=1;
    alo<=4'b0011;
    ala<=0;
    end
4'b0110: begin//or
    tmpwr<=1;
    hlt<=1;
    jmp<=0;
    brc<=0;
    rgw<=2'b00;
    ioc<=4'b1100;
    rgr<=1;
    alo<=4'b0100;
    ala<=0;
    end
4'b0111: begin//nor
    tmpwr<=0;
    hlt<=1;
    jmp<=0;
    brc<=0;
    rgw<=2'b00;
    ioc<=4'b1100;
    rgr<=0;
    alo<=4'b0101;
    ala<=0;
    end
4'b1000: begin//slt
    tmpwr<=1;
    hlt<=1;
    jmp<=0;
    brc<=0;
    rgw<=2'b00;
```

```
    ioc<=4'b1100;
    rgr<=1;
    alo<=4'b0110;
    ala<=0;
    end
4'b1001: begin//sr
    tmpwr<=1;
    hlt<=1;
    jmp<=0;
    brc<=0;
    rgw<=2'b00;
    ioc<=4'b1100;
    rgr<=1;
    alo<=4'b1000;
    ala<=0;
    end
4'b1010: begin//sl
    tmpwr<=1;
    hlt<=1;
    jmp<=0;
    brc<=0;
    rgw<=2'b00;
    ioc<=4'b1100;
    rgr<=1;
    alo<=4'b0111;
    ala<=0;
    end
4'b1011: begin//jr
    tmpwr<=0;
    hlt<=1;
    jmp<=1;
    brc<=0;
    rgw<=2'b01;
    ioc<=4'b1100;
    rgr<=0;
    end
4'b1100: begin//la
    tmpwr<=1;
    hlt<=1;
    jmp<=0;
    brc<=0;
    rgw<=2'b11;
    ioc<=4'b1000;
    end
4'b1101: begin//lc
    tmpwr<=1;
    hlt<=1;
    jmp<=0;
    brc<=0;
    rgw<=2'b11;
    ioc<=4'b1000;
    end
4'b1110: begin//lw
    tmpwr<=1;
    hlt<=1;
    jmp<=0;
    brc<=0;
    rgw<=2'b10;
    ioc<=4'b1101;
    rgr<=0;
    alo<=4'b0000;
    ala<=1;
```

```
                end
            4'b1111: begin//sw
                hlt<=1;
                jmp<=0;
                brc<=0;
                ioc<=4'b0110;
                rgr<=0;
                alo<=4'b0000;
                ala<=1;
                end
            endcase
    end
endmodule
```

13.1 Simulação

Para verificar o funcionamento do componente foi criado um código extra para simulações que é o código abaixo:

```
//MODULOS DE TESTE

module nR_Control_Test();
    reg cc_clk;
    integer cc_inst=0;
    wire cc_tmpwr,cc_hlt,cc_jmp,cc_brc,cc_rgr,cc_ala;
    wire[1:0] cc_rgw;
    wire[3:0] cc_ioc,cc_alo;
    always
    begin
        #1 cc_inst<=cc_inst+1; if(cc_inst>15) cc_inst<=0;
        #1 cc_clk <= ~cc_clk;
    end

    initial
    begin
        cc_clk <= 1'b0;
        $monitor("Control: Clock = %b Inst = %0d Tmpwr = %b Hlt =
            %b Jmp = %b Brc = %b Rgw = %b Ioc = %b Rgr = %b Alo
            = %b Ala = %b",cc_clk,cc_inst,cc_tmpwr,cc_hlt,cc_jmp,
            cc_brc,cc_rgw,cc_ioc,cc_rgr,cc_alo,cc_ala);
        #60 $stop;
    end

    nR_Control CC(cc_clk,cc_inst,cc_tmpwr,cc_hlt,cc_jmp,cc_brc,
        cc_rgw,cc_ioc,cc_rgr,cc_alo,cc_ala);
endmodule
```

13.1.1 Resultados

O controle simplesmente obedece a tabela de bits de controle atualizando seus valores na borda de subida do clock e seu funcionamento pode ser observado abaixo:

```
VSIM 17> run -all
# Control: Clock = 0 Inst = 0 Tmpwr = 0 Hlt = 1 Jmp = 0 Brc = 0 Rgw = 00 Ioc = 0000 Rgr = 0 Alo = 0000 Ala = 0
# Control: Clock = 0 Inst = 1 Tmpwr = 0 Hlt = 1 Jmp = 0 Brc = 0 Rgw = 00 Ioc = 0000 Rgr = 0 Alo = 0000 Ala = 0
# Control: Clock = 1 Inst = 1 Tmpwr = 0 Hlt = 1 Jmp = 0 Brc = 1 Rgw = 00 Ioc = 0100 Rgr = 0 Alo = 0001 Ala = 0
# Control: Clock = 1 Inst = 2 Tmpwr = 0 Hlt = 1 Jmp = 0 Brc = 1 Rgw = 00 Ioc = 0100 Rgr = 0 Alo = 0001 Ala = 0
# Control: Clock = 0 Inst = 2 Tmpwr = 0 Hlt = 1 Jmp = 0 Brc = 1 Rgw = 00 Ioc = 0100 Rgr = 0 Alo = 0001 Ala = 0
# Control: Clock = 0 Inst = 3 Tmpwr = 0 Hlt = 1 Jmp = 0 Brc = 1 Rgw = 00 Ioc = 0100 Rgr = 0 Alo = 0001 Ala = 0
# Control: Clock = 1 Inst = 3 Tmpwr = 1 Hlt = 1 Jmp = 0 Brc = 0 Rgw = 00 Ioc = 1100 Rgr = 1 Alo = 0000 Ala = 0
# Control: Clock = 1 Inst = 4 Tmpwr = 1 Hlt = 1 Jmp = 0 Brc = 0 Rgw = 00 Ioc = 1100 Rgr = 1 Alo = 0000 Ala = 0
# Control: Clock = 0 Inst = 4 Tmpwr = 1 Hlt = 1 Jmp = 0 Brc = 0 Rgw = 00 Ioc = 1100 Rgr = 1 Alo = 0000 Ala = 0
# Control: Clock = 0 Inst = 5 Tmpwr = 1 Hlt = 1 Jmp = 0 Brc = 0 Rgw = 00 Ioc = 1100 Rgr = 1 Alo = 0000 Ala = 0
# Control: Clock = 1 Inst = 5 Tmpwr = 1 Hlt = 1 Jmp = 0 Brc = 0 Rgw = 00 Ioc = 1100 Rgr = 1 Alo = 0011 Ala = 0
# Control: Clock = 1 Inst = 6 Tmpwr = 1 Hlt = 1 Jmp = 0 Brc = 0 Rgw = 00 Ioc = 1100 Rgr = 1 Alo = 0011 Ala = 0
# Control: Clock = 0 Inst = 6 Tmpwr = 1 Hlt = 1 Jmp = 0 Brc = 0 Rgw = 00 Ioc = 1100 Rgr = 1 Alo = 0011 Ala = 0
# Control: Clock = 0 Inst = 7 Tmpwr = 1 Hlt = 1 Jmp = 0 Brc = 0 Rgw = 00 Ioc = 1100 Rgr = 1 Alo = 0011 Ala = 0
# Control: Clock = 1 Inst = 7 Tmpwr = 0 Hlt = 1 Jmp = 0 Brc = 0 Rgw = 00 Ioc = 1100 Rgr = 0 Alo = 0101 Ala = 0
# Control: Clock = 1 Inst = 8 Tmpwr = 0 Hlt = 1 Jmp = 0 Brc = 0 Rgw = 00 Ioc = 1100 Rgr = 0 Alo = 0101 Ala = 0
# Control: Clock = 0 Inst = 8 Tmpwr = 0 Hlt = 1 Jmp = 0 Brc = 0 Rgw = 00 Ioc = 1100 Rgr = 0 Alo = 0101 Ala = 0
# Control: Clock = 0 Inst = 9 Tmpwr = 0 Hlt = 1 Jmp = 0 Brc = 0 Rgw = 00 Ioc = 1100 Rgr = 0 Alo = 0101 Ala = 0
# Control: Clock = 1 Inst = 9 Tmpwr = 1 Hlt = 1 Jmp = 0 Brc = 0 Rgw = 00 Ioc = 1100 Rgr = 1 Alo = 1000 Ala = 0
# Control: Clock = 1 Inst = 10 Tmpwr = 1 Hlt = 1 Jmp = 0 Brc = 0 Rgw = 00 Ioc = 1100 Rgr = 1 Alo = 1000 Ala = 0
# Control: Clock = 0 Inst = 10 Tmpwr = 1 Hlt = 1 Jmp = 0 Brc = 0 Rgw = 00 Ioc = 1100 Rgr = 1 Alo = 1000 Ala = 0
# Control: Clock = 0 Inst = 11 Tmpwr = 1 Hlt = 1 Jmp = 0 Brc = 0 Rgw = 00 Ioc = 1100 Rgr = 1 Alo = 1000 Ala = 0
# Control: Clock = 1 Inst = 11 Tmpwr = 0 Hlt = 1 Jmp = 1 Brc = 0 Rgw = 01 Ioc = 1100 Rgr = 0 Alo = 1000 Ala = 0
# Control: Clock = 1 Inst = 12 Tmpwr = 0 Hlt = 1 Jmp = 1 Brc = 0 Rgw = 01 Ioc = 1100 Rgr = 0 Alo = 1000 Ala = 0
# Control: Clock = 0 Inst = 12 Tmpwr = 0 Hlt = 1 Jmp = 1 Brc = 0 Rgw = 01 Ioc = 1100 Rgr = 0 Alo = 1000 Ala = 0
# Control: Clock = 0 Inst = 13 Tmpwr = 0 Hlt = 1 Jmp = 1 Brc = 0 Rgw = 01 Ioc = 1100 Rgr = 0 Alo = 1000 Ala = 0
# Control: Clock = 1 Inst = 13 Tmpwr = 1 Hlt = 1 Jmp = 0 Brc = 0 Rgw = 11 Ioc = 1000 Rgr = 0 Alo = 1000 Ala = 0
# Control: Clock = 1 Inst = 14 Tmpwr = 1 Hlt = 1 Jmp = 0 Brc = 0 Rgw = 11 Ioc = 1000 Rgr = 0 Alo = 1000 Ala = 0
# Control: Clock = 0 Inst = 14 Tmpwr = 1 Hlt = 1 Jmp = 0 Brc = 0 Rgw = 11 Ioc = 1000 Rgr = 0 Alo = 1000 Ala = 0
# Control: Clock = 0 Inst = 15 Tmpwr = 1 Hlt = 1 Jmp = 0 Brc = 0 Rgw = 11 Ioc = 1000 Rgr = 0 Alo = 1000 Ala = 0
# Control: Clock = 1 Inst = 15 Tmpwr = 1 Hlt = 1 Jmp = 0 Brc = 0 Rgw = 11 Ioc = 0110 Rgr = 0 Alo = 0000 Ala = 1
# Control: Clock = 1 Inst = 16 Tmpwr = 1 Hlt = 1 Jmp = 0 Brc = 0 Rgw = 11 Ioc = 0110 Rgr = 0 Alo = 0000 Ala = 1
# Control: Clock = 0 Inst = 16 Tmpwr = 1 Hlt = 1 Jmp = 0 Brc = 0 Rgw = 11 Ioc = 0110 Rgr = 0 Alo = 0000 Ala = 1
```

Figura 16: Simulação módulo de controle

14 ALU

O ALU ou ULA faz as operações lógicas e aritméticas do processador, abaixo podemos ver o código verilog desse componente, acima do código há um comentário explicando suas entradas e saídas.

```
//Modulo de ALU:
//Entradas:
//         in0: entrada0 da ALU
//         in1: entrada1 da ALU
//         alo: Bits de controle da ALU
//Saidas:
//         zero: saida de controle da ALU
//         out0: saida da ALU
//         ovrflw: overflow da ALU
module nR_ALU(in0,in1,alo,u_zero,u_out0,ovrflw);
      input[7:0] in0,in1,alo;
      output reg [1:0] ovrflw;
      output reg u_zero;
      output reg [7:0] u_out0;

      initial begin
            ovrflw<=0;
            u_zero<=0;
            u_out0<=0;
      end

      always@(*)
            case(alo)
```

```
 4'b0000: begin//add-0
    u_zero<=0;
    if(in0+in1>8'b11111111) begin//negative overflow
          u_out0<=128+(in0+in1);
          ovrflw<=2'b10;
    end
    else if(in0+in1>8'b01111111 & in0<=8'b0111111 & in1
        <=8'b0111111) begin//positive overflow
          u_out0<=(128-(in0+in1));
          ovrflw<=2'b01;
    end
    else begin
          u_out0<=in0+in1;
          ovrflw<=0;
    end
   end
 4'b0001: begin//sub-1
   u_zero<=0;
    if(in0-in1>8'b11111111) begin//negative overflow
          u_out0<=128+(in0-in1);
          ovrflw<=2'b01;
    end else if(in0-in1>8'b01111111 & in0>8'b01111111 &
        in1>8'b01111111) begin//positive overflow
          u_out0<=128-(in0-in1);
          ovrflw<=2'b10;
    end
    else begin
          u_out0<=in0-in1;
          ovrflw<=0;
    end
    end
 4'b0010: begin//flag test-2
    ovrflw<=0;
    if((in0&in1)==in1 & in0>0) begin
          u_out0<=in0-in1;
          u_zero<=1;
    end  else u_zero=0;
    end
 4'b0011: begin//and-3
    ovrflw<=0;
     u_zero<=0;
     u_out0<=in0&in1;
end
 4'b0100: begin//or-4
    ovrflw<=0;
     u_zero<=0;
     u_out0<=(in0|in1);
 end
 4'b0101: begin//nor-5
    ovrflw<=0;
     u_zero<=0;
     u_out0<=~(in0|in1);
     end
 4'b0110: begin//less-6
    ovrflw<=0;
    u_out0<=0;
     u_zero<=(in0>in1);
     end
 4'b0111: begin//sl-7
    ovrflw<=0;
     u_zero<=0;
     u_out0<=in0<<in1;
```

```
                end
            4'b1000: begin//sr-8
                ovrflw<=0;
                u_zero<=0;
                 u_out0<=in0>>in1;
                 end
        endcase
endmodule

//Modulo de Somador:
//Entradas:
//        in0: entrada0 do somador
//        in1: entrada1 do somador
//Saidas:
//        out0: saida do somador

module nR_Adder(in0,in1,add_out0);
     input[7:0] in0,in1;
     output[7:0] add_out0;

     assign add_out0=in0+in1;
endmodule
```

14.1 Simulação

Para verificar o funcionamento do componente foi criado um código extra para simulações que é o código abaixo:

```
//MODULOS DE TESTE

 module nR_ALU_Test();
     integer u_in0=0;
     integer u_in1=0;
     integer u_alo=0;
     wire zero;
     wire[7:0] out0;
     wire[1:0] ovrflw;
     always
     begin
          #1 u_in0<=u_in0+1; if(u_in0>3) begin u_in0<=0; u_in1=
              u_in1+1; end if(u_in1>3) begin u_in1<=0; u_alo=u_alo
              +1; end if(u_alo>4'b0001) $stop;
     end

     initial
     begin
          $monitor("ALU: Alo = %0d In0 = %0d In1 = %0d Out = %0d
              Zero = %b Overflow = %b",u_alo,u_in0,u_in1,out0,zero,
              ovrflw);
     end

     nR_ALU ULA(u_in0,u_in1,u_alo,zero,out0,ovrflw);
endmodule

module nR_Adder_Test();
     integer a_in0=0;
     integer a_in1=0;
     wire[7:0] a_out;
     always
     begin
```

```
        #1 a_in0<=a_in0+1; if(a_in0>15) begin a_in0<=0; a_in1=
            a_in1+1; end if(a_in1>15) $stop;
    end

    initial
    begin
        $monitor("Adder: In0 = %0d In1 = %0d Out = %0d",a_in0,
            a_in1,a_out);
        #80 $stop;
    end

    nR_Adder Add(a_in0,a_in1,a_out);
endmodule
```

14.1.1 Resultados

A ULA funciona de forma assíncrona, então devemos observar apenas as entradas e o código de cada operação, para essa simulação foram testadas todas as possibilidades de operações, porém so algumas serão mostradas. É facilmente visível que o módulo está funcional. A imagens das simulações estão abaixo.

```
# ALU: Alo = 0 In0 = 3 In1 = 0 Out = 3 Zero = 0 Overflow = 00
# ALU: Alo = 0 In0 = 4 In1 = 0 Out = 4 Zero = 0 Overflow = 00
# ALU: Alo = 0 In0 = 0 In1 = 1 Out = 1 Zero = 0 Overflow = 00
# ALU: Alo = 0 In0 = 1 In1 = 1 Out = 2 Zero = 0 Overflow = 00
# ALU: Alo = 0 In0 = 2 In1 = 1 Out = 3 Zero = 0 Overflow = 00
# ALU: Alo = 0 In0 = 3 In1 = 1 Out = 4 Zero = 0 Overflow = 00
# ALU: Alo = 0 In0 = 4 In1 = 1 Out = 5 Zero = 0 Overflow = 00
# ALU: Alo = 0 In0 = 0 In1 = 2 Out = 2 Zero = 0 Overflow = 00
# ALU: Alo = 0 In0 = 1 In1 = 2 Out = 3 Zero = 0 Overflow = 00
# ALU: Alo = 0 In0 = 2 In1 = 2 Out = 4 Zero = 0 Overflow = 00
# ALU: Alo = 0 In0 = 3 In1 = 2 Out = 5 Zero = 0 Overflow = 00
# ALU: Alo = 0 In0 = 4 In1 = 2 Out = 6 Zero = 0 Overflow = 00
# ALU: Alo = 0 In0 = 0 In1 = 3 Out = 3 Zero = 0 Overflow = 00
# ALU: Alo = 0 In0 = 1 In1 = 3 Out = 4 Zero = 0 Overflow = 00
# ALU: Alo = 0 In0 = 2 In1 = 3 Out = 5 Zero = 0 Overflow = 00
# ALU: Alo = 0 In0 = 3 In1 = 3 Out = 6 Zero = 0 Overflow = 00
# ALU: Alo = 0 In0 = 4 In1 = 3 Out = 7 Zero = 0 Overflow = 00
# ALU: Alo = 1 In0 = 0 In1 = 0 Out = 0 Zero = 0 Overflow = 00
# ALU: Alo = 1 In0 = 1 In1 = 0 Out = 1 Zero = 0 Overflow = 00
# ALU: Alo = 1 In0 = 2 In1 = 0 Out = 2 Zero = 0 Overflow = 00
# ALU: Alo = 1 In0 = 3 In1 = 0 Out = 3 Zero = 0 Overflow = 00
# ALU: Alo = 1 In0 = 4 In1 = 0 Out = 4 Zero = 0 Overflow = 00
# ALU: Alo = 1 In0 = 0 In1 = 1 Out = 255 Zero = 0 Overflow = 00
# ALU: Alo = 1 In0 = 1 In1 = 1 Out = 0 Zero = 0 Overflow = 00
# ALU: Alo = 1 In0 = 2 In1 = 1 Out = 1 Zero = 0 Overflow = 00
# ALU: Alo = 1 In0 = 3 In1 = 1 Out = 2 Zero = 0 Overflow = 00
# ALU: Alo = 1 In0 = 4 In1 = 1 Out = 3 Zero = 0 Overflow = 00
# ALU: Alo = 1 In0 = 0 In1 = 2 Out = 254 Zero = 0 Overflow = 00
# ALU: Alo = 1 In0 = 1 In1 = 2 Out = 255 Zero = 0 Overflow = 00
# ALU: Alo = 1 In0 = 2 In1 = 2 Out = 0 Zero = 0 Overflow = 00
# ALU: Alo = 1 In0 = 3 In1 = 2 Out = 1 Zero = 0 Overflow = 00
```

Figura 17: Simulação ALU, operação de soma e subtração

```
# ALU: Alo = 2 In0 = 2 In1 = 1 Out = 0 Zero = 0 Overflow = 00
# ALU: Alo = 2 In0 = 3 In1 = 1 Out = 2 Zero = 1 Overflow = 00
# ALU: Alo = 2 In0 = 4 In1 = 1 Out = 2 Zero = 0 Overflow = 00
# ALU: Alo = 2 In0 = 0 In1 = 2 Out = 2 Zero = 0 Overflow = 00
# ALU: Alo = 2 In0 = 1 In1 = 2 Out = 2 Zero = 0 Overflow = 00
# ALU: Alo = 2 In0 = 2 In1 = 2 Out = 0 Zero = 1 Overflow = 00
# ALU: Alo = 2 In0 = 3 In1 = 2 Out = 1 Zero = 1 Overflow = 00
# ALU: Alo = 2 In0 = 4 In1 = 2 Out = 1 Zero = 0 Overflow = 00
# ALU: Alo = 2 In0 = 0 In1 = 3 Out = 1 Zero = 0 Overflow = 00
# ALU: Alo = 2 In0 = 1 In1 = 3 Out = 1 Zero = 0 Overflow = 00
# ALU: Alo = 2 In0 = 2 In1 = 3 Out = 1 Zero = 0 Overflow = 00
# ALU: Alo = 2 In0 = 3 In1 = 3 Out = 0 Zero = 1 Overflow = 00
# ALU: Alo = 2 In0 = 4 In1 = 3 Out = 0 Zero = 0 Overflow = 00
# ALU: Alo = 3 In0 = 0 In1 = 0 Out = 0 Zero = 0 Overflow = 00
# ALU: Alo = 3 In0 = 1 In1 = 0 Out = 0 Zero = 0 Overflow = 00
# ALU: Alo = 3 In0 = 2 In1 = 0 Out = 0 Zero = 0 Overflow = 00
# ALU: Alo = 3 In0 = 3 In1 = 0 Out = 0 Zero = 0 Overflow = 00
# ALU: Alo = 3 In0 = 4 In1 = 0 Out = 0 Zero = 0 Overflow = 00
# ALU: Alo = 3 In0 = 0 In1 = 1 Out = 0 Zero = 0 Overflow = 00
# ALU: Alo = 3 In0 = 1 In1 = 1 Out = 1 Zero = 0 Overflow = 00
# ALU: Alo = 3 In0 = 2 In1 = 1 Out = 0 Zero = 0 Overflow = 00
# ALU: Alo = 3 In0 = 3 In1 = 1 Out = 1 Zero = 0 Overflow = 00
# ALU: Alo = 3 In0 = 4 In1 = 1 Out = 0 Zero = 0 Overflow = 00
# ALU: Alo = 3 In0 = 0 In1 = 2 Out = 0 Zero = 0 Overflow = 00
# ALU: Alo = 3 In0 = 1 In1 = 2 Out = 0 Zero = 0 Overflow = 00
# ALU: Alo = 3 In0 = 2 In1 = 2 Out = 2 Zero = 0 Overflow = 00
# ALU: Alo = 3 In0 = 3 In1 = 2 Out = 2 Zero = 0 Overflow = 00
# ALU: Alo = 3 In0 = 4 In1 = 2 Out = 0 Zero = 0 Overflow = 00
# ALU: Alo = 3 In0 = 0 In1 = 3 Out = 0 Zero = 0 Overflow = 00
# ALU: Alo = 3 In0 = 1 In1 = 3 Out = 1 Zero = 0 Overflow = 00
# ALU: Alo = 3 In0 = 2 In1 = 3 Out = 2 Zero = 0 Overflow = 00
# ALU: Alo = 3 In0 = 3 In1 = 3 Out = 3 Zero = 0 Overflow = 00
```

Figura 18: Simulação ALU, operação de flag e and

```
# ALU: Alo = 6 In0 = 0 In1 = 0 Out = 0 Zero = 0 Overflow = 00
# ALU: Alo = 6 In0 = 1 In1 = 0 Out = 0 Zero = 1 Overflow = 00
# ALU: Alo = 6 In0 = 2 In1 = 0 Out = 0 Zero = 1 Overflow = 00
# ALU: Alo = 6 In0 = 3 In1 = 0 Out = 0 Zero = 1 Overflow = 00
# ALU: Alo = 6 In0 = 4 In1 = 0 Out = 0 Zero = 1 Overflow = 00
# ALU: Alo = 6 In0 = 0 In1 = 1 Out = 0 Zero = 0 Overflow = 00
# ALU: Alo = 6 In0 = 1 In1 = 1 Out = 0 Zero = 0 Overflow = 00
# ALU: Alo = 6 In0 = 2 In1 = 1 Out = 0 Zero = 1 Overflow = 00
# ALU: Alo = 6 In0 = 3 In1 = 1 Out = 0 Zero = 1 Overflow = 00
# ALU: Alo = 6 In0 = 4 In1 = 1 Out = 0 Zero = 1 Overflow = 00
# ALU: Alo = 6 In0 = 0 In1 = 2 Out = 0 Zero = 0 Overflow = 00
# ALU: Alo = 6 In0 = 1 In1 = 2 Out = 0 Zero = 0 Overflow = 00
# ALU: Alo = 6 In0 = 2 In1 = 2 Out = 0 Zero = 0 Overflow = 00
# ALU: Alo = 6 In0 = 3 In1 = 2 Out = 0 Zero = 1 Overflow = 00
# ALU: Alo = 6 In0 = 4 In1 = 2 Out = 0 Zero = 1 Overflow = 00
# ALU: Alo = 6 In0 = 0 In1 = 3 Out = 0 Zero = 0 Overflow = 00
# ALU: Alo = 6 In0 = 1 In1 = 3 Out = 0 Zero = 0 Overflow = 00
# ALU: Alo = 6 In0 = 2 In1 = 3 Out = 0 Zero = 0 Overflow = 00
# ALU: Alo = 6 In0 = 3 In1 = 3 Out = 0 Zero = 0 Overflow = 00
# ALU: Alo = 6 In0 = 4 In1 = 3 Out = 0 Zero = 1 Overflow = 00
# ** Note: $stop    : C:/Users/nanoTech Corp/Desktop/ALU/ALU_Test
#    Time: 100 ps  Iteration: 0  Instance: /nR_ALU_Test
```

Figura 19: Simulação ALU, operação de less

Figura 20: Simulação ALU, operação de or e nor

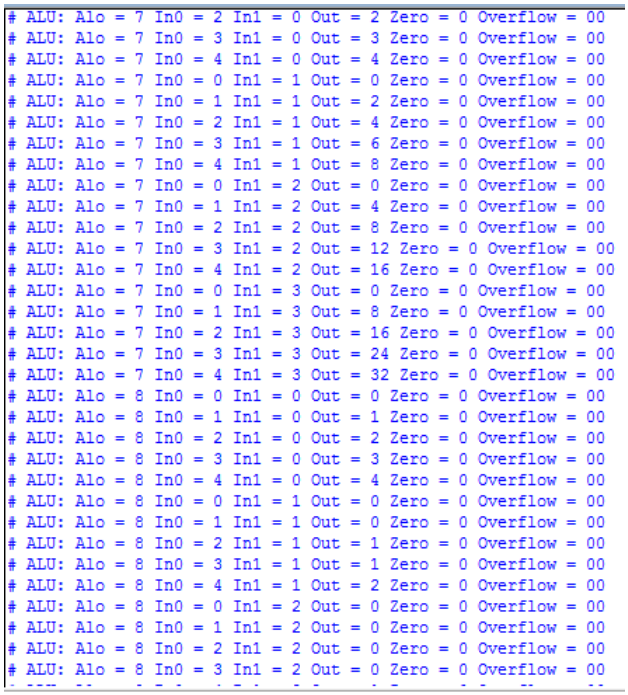


Figura 21: Simulação ALU, operação de shift esquerda e direita

A imagem abaixo contem a simulação do modulo somador, que adiciona 1 ou 0 ao endereço de next. A saída é a soma das entradas, seu funcionamento está correto.

```
# Adder: In0 = 12 In1 = 2 Out = 14
# Adder: In0 = 13 In1 = 2 Out = 15
# Adder: In0 = 14 In1 = 2 Out = 16
# Adder: In0 = 15 In1 = 2 Out = 17
# Adder: In0 = 16 In1 = 2 Out = 18
# Adder: In0 = 0 In1 = 3 Out = 3
# Adder: In0 = 1 In1 = 3 Out = 4
# Adder: In0 = 2 In1 = 3 Out = 5
# Adder: In0 = 3 In1 = 3 Out = 6
# Adder: In0 = 4 In1 = 3 Out = 7
# Adder: In0 = 5 In1 = 3 Out = 8
# Adder: In0 = 6 In1 = 3 Out = 9
# Adder: In0 = 7 In1 = 3 Out = 10
# Adder: In0 = 8 In1 = 3 Out = 11
# Adder: In0 = 9 In1 = 3 Out = 12
# Adder: In0 = 10 In1 = 3 Out = 13
# Adder: In0 = 11 In1 = 3 Out = 14
# Adder: In0 = 12 In1 = 3 Out = 15
# Adder: In0 = 13 In1 = 3 Out = 16
# Adder: In0 = 14 In1 = 3 Out = 17
# Adder: In0 = 15 In1 = 3 Out = 18
# Adder: In0 = 16 In1 = 3 Out = 19
# Adder: In0 = 0 In1 = 4 Out = 4
# Adder: In0 = 1 In1 = 4 Out = 5
# Adder: In0 = 2 In1 = 4 Out = 6
# Adder: In0 = 3 In1 = 4 Out = 7
# Adder: In0 = 4 In1 = 4 Out = 8
# Adder: In0 = 5 In1 = 4 Out = 9
# Adder: In0 = 6 In1 = 4 Out = 10
# Adder: In0 = 7 In1 = 4 Out = 11
# Adder: In0 = 8 In1 = 4 Out = 12
# Adder: In0 = 9 In1 = 4 Out = 13
# Adder: In0 = 10 In1 = 4 Out = 14
# Adder: In0 = 11 In1 = 4 Out = 15
```

Figura 22: Simulação módulo de soma

15 Código compilado

O código do programa compilado para nanoRisk com a novas alterações do compilador é o código abaixo, espera-se que ele não sofra mais alterações

```
00011001//data seg size
 00000001//data

00101100 00000000
10010110 00000010
10011010 00000001
01100101 00000000
11011100 00000001
11100000 00000001
00010100 00000010
10011010 00000001
00000001 00000010
00000010 00001000
00101000 00011010
00011100 00010000
11000111//instructions
 00010001
11001000 00010101
11100111 00000111
11101000 00001000
11010010 10001010
10110010 00000000
00110100 00100000
00111011 01010000
00111100 01100000
11000111 00010010
11001000 00010110
11100111 00000111
11101000 00001000
11010010 10001010
10110010 00000000
00110100 00100000
00111011 01011011
00111100 01101100
11000111 00010011
11001000 00010111
11100111 00000111
11101000 00001000
11010010 10001010
10110010 00000000
00110100 00100000
00111011 01011011
00111100 01101100
11000111 00010100
11001000 00011000
```

11100111 00000111
11101000 00001000
11010010 10001010
10110010 00000000
00110100 00100000
00111100 01101100
00111011 01011011
11011101 00000000
11001110 00000000
11010010 00001000
10000010 11010010
11010010 10000100
00010010 00000010
11010010 00000001
01000011 00110010
11111101 00000011
11010010 00000001
01000011 00110010
11111110 00000011
11101101 00001110
11101110 00011110
01001101 11001101
01001110 10111110
00110010 11011110
11101110 00000011
11010010 00000001
00110011 00110010
11101101 00000011
11010010 00000001
00110011 00110010
11010010 10000100
00010000 00100010
11010010 00000001
00111101 11010010
11010010 00000010
00111110 11100010
10110000 01001100
11010010 00000001
00111101 11010010
00000001 00000000
11010010 00000001
01000011 00110010
11111011 00000011
11010010 00000001
01000011 00110010
11111100 00000011
10001011 01111000
11010010 10100010
00011011 00000010
00111011 01110000

```
00111100  10000000
10110000  10100110
00111011  10000000
00111100  01110000
11010101  00000000
11010110  00000000
11010010  00000000
10000010  10110010
01110010  00100000
11010010  11000010
00010010  00000010
00110101  01011100
11010111  00000010
11010010  00000001
00111011  10110010
11010010  11010000
00100001  01110010
10110000  10101010
11101100  00000011
11010010  00000001
00110011  00110010
11101011  00000011
11010010  00000001
00110011  00110010
10110100  00000000
11010010  00000001
00110110  01100010
10110000  10101010
```

16 Multiplexador

Há no projeto multiplexadores de 2 e de 4 entradas. Os multiplexadores são importantes pois trabalham junto com o controle para definir o caminho dos dados, abaixo podemos ver o código verilog desse componente, acima do código há um comentario explicando suas entradas e saídas.

```
//Modulo de Multiplexacao:
//Entradas:
//        in0: entrada 0
//        in1: entrada 1
//        sel: selecao
//Saidas:
//        out: saida
module nR_MUX2(in0,in1,sel,m_out);
     input[7:0] in0,in1;
     input sel;
     output [7:0] m_out;

     assign m_out = (in0&~sel) | (in1&sel);
endmodule

//Modulo de Multiplexacao:
//Entradas:
```

```
//         in0: entrada 0
//         in1: entrada 1
//         in2: entrada 2
//         in3: entrada 3
//         sel: selecao
//Saidas:
//         out: saida
module nR_MUX4(in0,in1,in2,in3,sel,out);
     input[7:0] in0,in1,in2,in3;
     input[1:0] sel;
     output reg[7:0] out;
     always@(*)
     case(sel)
          0: out<=in0;
          1: out<=in1;
          2: out<=in2;
          3: out<=in3;
     endcase
endmodule
```

16.1 Simulação

Para verificar o funcionamento do componente foi criado um código extra para simulações que é o código abaixo:

```
//MODULOS DE TESTE

module nR_MUX_Test();
     integer mux_in0=2;
     integer mux_in1=3;
     wire[7:0] mux_out;
     reg mux_sel0;
     always
     begin
          #1 mux_sel0 <= ~mux_sel0;
     end

     initial
     begin
          mux_sel0 <= 1'b0;
          $monitor("MUX2: In0 = %0d In1 = %0d Sel = %b Out = %0d",
              mux_in0,mux_in1,mux_sel0,mux_out);
          #60 $stop;
     end

     nR_MUX2 M2(mux_in0,mux_in1,mux_sel0,mux_out);
endmodule

module nR_MUX4_Test();
     integer mux_in0=11;
     integer mux_in1=12;
     integer mux_in2=13;
     integer mux_in3=14;
     wire[7:0] mux1_out;
     integer mux_sel1=0;
     always
     begin
          #1 mux_sel1<=mux_sel1+1; if(mux_sel1>=3) mux_sel1<=0;
     end

     initial
```

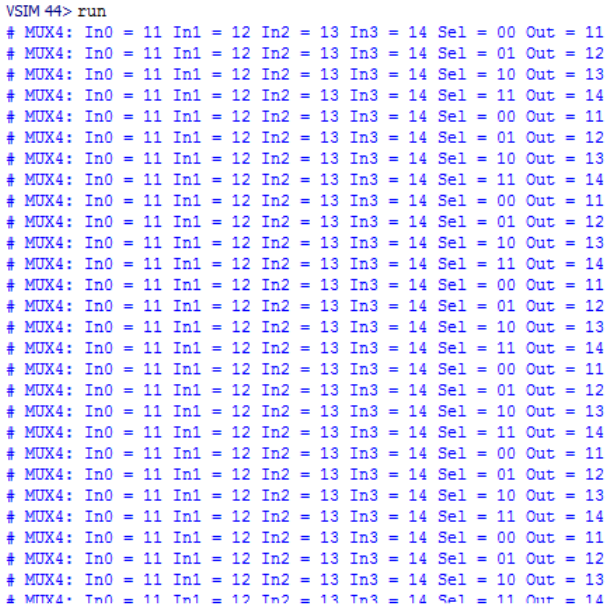
```
    begin
        $monitor("MUX4: In0 = %0d In1 = %0d In2 = %0d In3 = %0d
            Sel = %2b Out = %0d",mux_in0,mux_in1,mux_in2,mux_in3,
            mux_sel1,mux1_out);
        #60 $stop;
    end

    nR_MUX4 M4(mux_in0,mux_in1,mux_in2,mux_in3,mux_sel1,mux1_out);
endmodule
```

16.1.1 Resultados

É possível verificar seu funcionando comparando a simulação abaixo com a tabela da verdade de um MUX.

```
VSIM 60> run -all
# MUX2: In0 = 0 In1 = 1 Sel = 0 Out = 0
# MUX2: In0 = 0 In1 = 1 Sel = 1 Out = 1
# MUX2: In0 = 0 In1 = 1 Sel = 0 Out = 0
# MUX2: In0 = 0 In1 = 1 Sel = 1 Out = 1
# MUX2: In0 = 0 In1 = 1 Sel = 0 Out = 0
# MUX2: In0 = 0 In1 = 1 Sel = 1 Out = 1
# MUX2: In0 = 0 In1 = 1 Sel = 0 Out = 0
# MUX2: In0 = 0 In1 = 1 Sel = 1 Out = 1
# MUX2: In0 = 0 In1 = 1 Sel = 0 Out = 0
# MUX2: In0 = 0 In1 = 1 Sel = 1 Out = 1
# MUX2: In0 = 0 In1 = 1 Sel = 0 Out = 0
# MUX2: In0 = 0 In1 = 1 Sel = 1 Out = 1
# MUX2: In0 = 0 In1 = 1 Sel = 0 Out = 0
# MUX2: In0 = 0 In1 = 1 Sel = 1 Out = 1
# MUX2: In0 = 0 In1 = 1 Sel = 0 Out = 0
# MUX2: In0 = 0 In1 = 1 Sel = 1 Out = 1
# MUX2: In0 = 0 In1 = 1 Sel = 0 Out = 0
# MUX2: In0 = 0 In1 = 1 Sel = 1 Out = 1
# MUX2: In0 = 0 In1 = 1 Sel = 0 Out = 0
# MUX2: In0 = 0 In1 = 1 Sel = 1 Out = 1
# MUX2: In0 = 0 In1 = 1 Sel = 0 Out = 0
# MUX2: In0 = 0 In1 = 1 Sel = 1 Out = 1
# MUX2: In0 = 0 In1 = 1 Sel = 0 Out = 0
# MUX2: In0 = 0 In1 = 1 Sel = 1 Out = 1
# MUX2: In0 = 0 In1 = 1 Sel = 0 Out = 0
# MUX2: In0 = 0 In1 = 1 Sel = 1 Out = 1
# MUX2: In0 = 0 In1 = 1 Sel = 0 Out = 0
# MUX2: In0 = 0 In1 = 1 Sel = 1 Out = 1
```

Figura 23: Simulação MUX duas entradas



```
VSIM 44> run
# MUX4: In0 = 11 In1 = 12 In2 = 13 In3 = 14 Sel = 00 Out = 11
# MUX4: In0 = 11 In1 = 12 In2 = 13 In3 = 14 Sel = 01 Out = 12
# MUX4: In0 = 11 In1 = 12 In2 = 13 In3 = 14 Sel = 10 Out = 13
# MUX4: In0 = 11 In1 = 12 In2 = 13 In3 = 14 Sel = 11 Out = 14
# MUX4: In0 = 11 In1 = 12 In2 = 13 In3 = 14 Sel = 00 Out = 11
# MUX4: In0 = 11 In1 = 12 In2 = 13 In3 = 14 Sel = 01 Out = 12
# MUX4: In0 = 11 In1 = 12 In2 = 13 In3 = 14 Sel = 10 Out = 13
# MUX4: In0 = 11 In1 = 12 In2 = 13 In3 = 14 Sel = 11 Out = 14
# MUX4: In0 = 11 In1 = 12 In2 = 13 In3 = 14 Sel = 00 Out = 11
# MUX4: In0 = 11 In1 = 12 In2 = 13 In3 = 14 Sel = 01 Out = 12
# MUX4: In0 = 11 In1 = 12 In2 = 13 In3 = 14 Sel = 10 Out = 13
# MUX4: In0 = 11 In1 = 12 In2 = 13 In3 = 14 Sel = 11 Out = 14
# MUX4: In0 = 11 In1 = 12 In2 = 13 In3 = 14 Sel = 00 Out = 11
# MUX4: In0 = 11 In1 = 12 In2 = 13 In3 = 14 Sel = 01 Out = 12
# MUX4: In0 = 11 In1 = 12 In2 = 13 In3 = 14 Sel = 10 Out = 13
# MUX4: In0 = 11 In1 = 12 In2 = 13 In3 = 14 Sel = 11 Out = 14
# MUX4: In0 = 11 In1 = 12 In2 = 13 In3 = 14 Sel = 00 Out = 11
# MUX4: In0 = 11 In1 = 12 In2 = 13 In3 = 14 Sel = 01 Out = 12
# MUX4: In0 = 11 In1 = 12 In2 = 13 In3 = 14 Sel = 10 Out = 13
# MUX4: In0 = 11 In1 = 12 In2 = 13 In3 = 14 Sel = 11 Out = 14
# MUX4: In0 = 11 In1 = 12 In2 = 13 In3 = 14 Sel = 00 Out = 11
# MUX4: In0 = 11 In1 = 12 In2 = 13 In3 = 14 Sel = 01 Out = 12
# MUX4: In0 = 11 In1 = 12 In2 = 13 In3 = 14 Sel = 10 Out = 13
# MUX4: In0 = 11 In1 = 12 In2 = 13 In3 = 14 Sel = 11 Out = 14
# MUX4: In0 = 11 In1 = 12 In2 = 13 In3 = 14 Sel = 00 Out = 11
# MUX4: In0 = 11 In1 = 12 In2 = 13 In3 = 14 Sel = 01 Out = 12
# MUX4: In0 = 11 In1 = 12 In2 = 13 In3 = 14 Sel = 10 Out = 13
# MUX4: In0 = 11 In1 = 12 In2 = 13 In3 = 14 Sel = 11 Out = 14
```

Figura 24: Simulação MUX quatro entradas

17 Portas lógicas e controle de fluxo

A portas lógicas usadas são: NOT, AND(2 entradas) e OR(4 entradas), além do controle de fluxo que é uma OR de duas entradas. O código desses componentes está abaixo.

```
//Modulo de Controle de fluxo:
//Entradas:
//        branch: sinal de controle de branch
//        jump: sinal de controle de jump
//Saidas:
//        out: saida
module nR_FluxCtrl(branch,jump,out);
     input branch,jump;
     output out;
     assign out = branch | jump;
endmodule


//Modulo de NOT:
//Entradas:
//        in: entrada
//Saidas:
//        out: saida
module nR_NOT(in,out);
     input in;
     output out;
     assign out = ~in;
endmodule


//Modulo de AND2:
//Entradas:
```

```
//         in0: entrada 0
//         in1: entrada 1
//Saidas:
//         out: saida
module nR_AND2(in0,in1,out);
     input in0,in1;
     output out;
     assign out = in0&in1;
endmodule

//Modulo de AND4:
//Entradas:
//         in0: entrada 0
//         in1: entrada 1
//         in2: entrada 2
//         in3: entrada 3
//Saidas:
//         out: saida
module nR_AND4(in0,in1,in2,in3,out);
     input in0,in1,in2,in3;
     output out;
     assign out = in0&in1&in2&in3;
endmodule

//Modulo de OR4:
//Entradas:
//         in0: entrada 0
//         in1: entrada 1
//         in2: entrada 2
//         in3: entrada 3
//Saidas:
//         out: saida
module nR_OR4(in0,in1,in2,in3,out);
     input in0,in1,in2,in3;
     output out;
     assign out = in0|in1|in2|in3;
endmodule
```

17.1 Simulação

Por serem portas lógicas, a simulação escolhida foi por ondas, que é mais simples e de fácil visualização.

17.1.1 Resultados

Os resultados das simulações está abaixo e condizem com os resultados esperados.

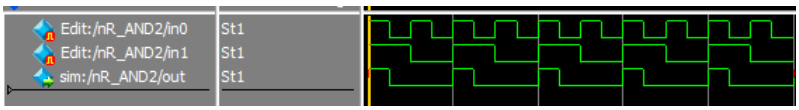


Figura 25: Simulação AND duas entradas

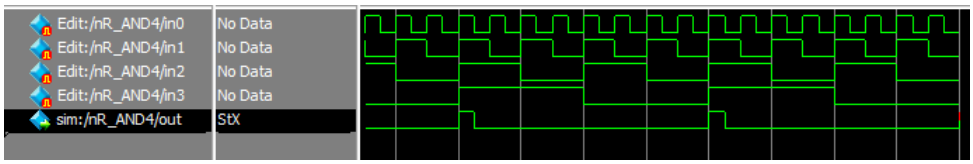


Figura 26: Simulação porta AND quatro entradas

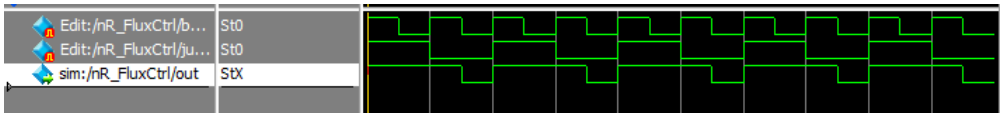


Figura 27: Simulação módulo de controle de fluxo

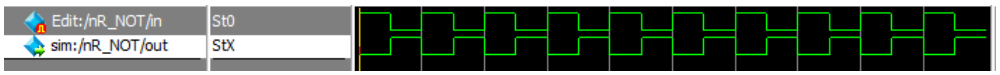


Figura 28: Simulação porta NOT

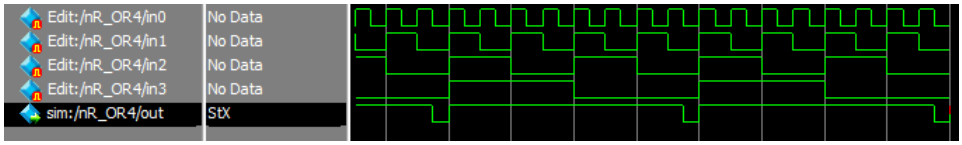


Figura 29: Simulação porta OR quatro entradas

18 BitCmp & BitExt

No projeto o BitExt é de 4 para 8 e o BitCmp é de 8 para 1. Esses componentes ligam vias de diferentes larguras, o BitExt coloca zeros não significativos na entrada e o BitCmp compacta os bits fazendo operações OR entre eles.

```
//Modulo de Compactador de Bit:
//Entradas:
//         in: entrada
//Saidas:
//         out: saida
module nR_BitCmp(in,out);
      input[7:0] in;
      output out;
      assign out =in[0]|in[1]|in[2]|in[3]|in[4]|in[5]|in[6]|in[7];
endmodule

//Modulo de Extensor de Bit:
//Entradas:
//         in: entrada
//Saidas:
//         out: saida
module nR_BitExt(in,out);
      input[3:0] in;
```

```
    output[7:0] out;
    assign out = {0,in};
endmodule
```

18.1 Simulação

Por ser mais simples, a simulação escolhida foi por ondas, que é mais simples e de fácil visualização.

18.1.1 Resultados

Os resultados foram o esperado.

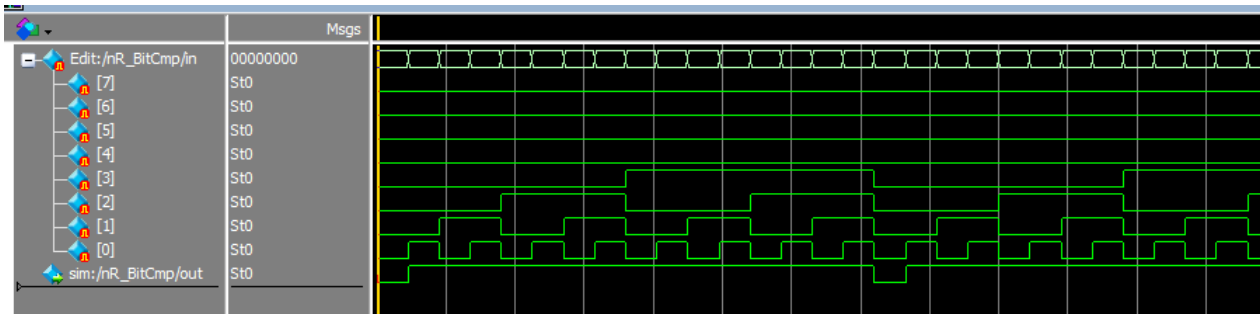


Figura 30: Simulação compactador de bits

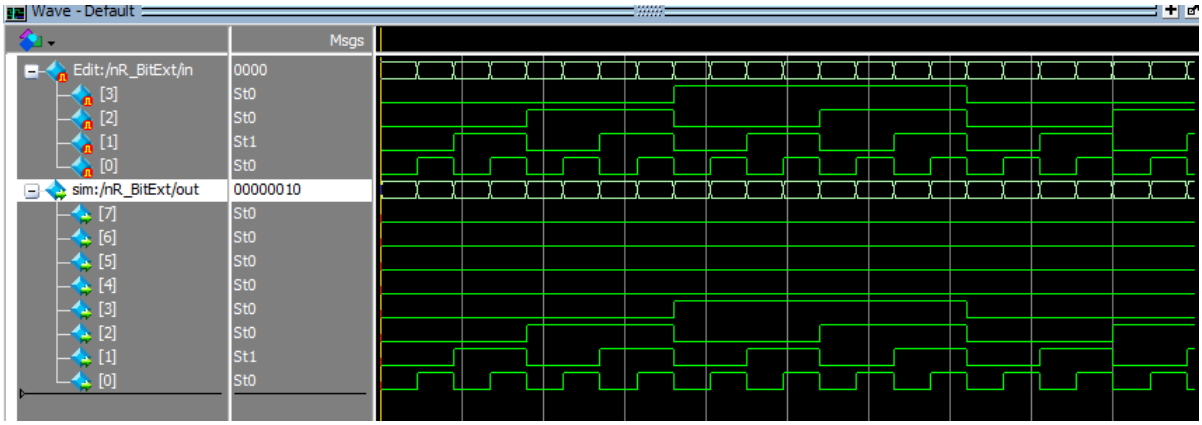


Figura 31: Simulação extensor de bits

19 Correções

Algumas modificações e adições foram feitas no código do nanoRisk.

19.1 Nova Porta lógica

foi feita uma nova porta lógica AND, essa com largura de entrada e saída de 8 bits, ela não foi feita antes devido a uma falta de atenção, se viu necessário o uso dela no caminho de dados. O código dessa porta está abaixo:

```
//Modulo de 8AND2:
//Entradas:
//        in0: entrada 0
//        in1: entrada 1
//Saidas:
//        out: saida
module nR_8AND2(in0,in1,out);
     input[7:0] in0;
     input in1;
     output reg[7:0] out;
     always@(*)
      if(in1)
        out<=in0;
      else
        out<=0;
endmodule
```

19.2 Alteração no código de Memória

O código de memória foi alterado também para evitar problemas futuros, na inicialização ele, agora define como zero a saída mas não zera todos os espaços de memória como fazia antes. Com as alterações o novo código é:

```
//Modulo memory: dataMem, instrMem
//Entradas:
//        clk: clock(escrita=borda de subida,leitura=borda de
    descida)
//        clr: limpa a memoria caso seja 1
//        adrIn0: endereco onde deve gravar um valor
//        in0: valor a ser gravado
//        adrOut0: endereco de leitura do valor 0
//        canWrt: habilita escrita
//        canRd: habilita leitura

//Saidas:
//        out0: valor lido em 0
module nR_Memory(clk,in0,m_out0,adrIn0,adrOut0,canWrt,canRd,clr);
     input clk,clr,canWrt,canRd;
     input[7:0] in0;
     input[3:0]adrIn0,adrOut0;
     output reg[7:0] m_out0;

     reg [7:0]m_mem[255:0];

     integer m_i;

     initial
     begin
               m_out0<=0;
     end

     always @ ( posedge clk & canWrt ) begin
          if(~clr)
               m_mem[adrIn0]<=in0;
     end
```

```
    always @ (clr) begin
        for (m_i=0;m_i<256;m_i=m_i+1)
            m_mem[m_i]<=0;
        m_out0<=0;
    end

    always @ (negedge clk & canRd) begin
        m_out0<=m_mem[adrOut0];
    end
endmodule
```

19.3 Alteração no código do Extensor de Bits

O código do extensor de bits não estava compilando em todos os computadores/versões do modelSIM, por isso, foram feitas alterações na sintaxe do código para deixar tudo mais explicito evitando erros. A lógica manteve a mesma, o novo código é:

```
//Modulo de Extensor de Bit:
//Entradas:
//        in: entrada
//Saidas:
//        out: saida
module nR_BitExt(in,out);
    input[3:0] in;
    output[7:0] out;
    assign out = {4'b0000,in};
endmodule
```

20 Código do Processador

O código do processador é a união de todos os componentes até então desenvolvidos seguindo os moldes do diagrama do processador.

```
module nRProcessor(clk,reset);

  input clk,reset;
  wire[7:0] im_addr,im_out;
  wire[7:0] dm_out;
  wire[7:0] nxtmux_in0,nxtmux_in1;
  wire[7:0] ctrlmux1_in0;
  wire nxtmux_sel,ctrlmux2_sel;
  wire ctrl2Cmp_out;
  wire adder_input;
  wire ULA_Zer0,branchAND_out;
  wire[1:0] regs_ovrflw;
  wire[7:0] regs_in0;
  wire[3:0] regs_adrIn0,regs_adrOut1;
  wire[7:0] regs_out0,regs_out1,regs_out2;
  wire[7:0] nxt_in;
  wire[7:0] ghost0_in,ghost1_in,ghost0_out,ghost1_out;
  wire ctrl_tmpwr,ctrl_hlt,ctrl_jmp,ctrl_brc,ctrl_rgr,ctrl_ala;
  wire[1:0] ctrl_rgw;
  wire[3:0] ctrl_alo,ctrl_ioc;
  wire[7:0] ULA_in1,ULA_out;
  wire haltNOTscape,haltScape;
  wire realClk,bitshift_out0;
```

```
nR_Memory InstructionsMemory(clk,8'b00000000,im_out,8'b00000000,
    im_addr,1'b0,1'b1,1'b0);
nR_Register Next(clk,nxt_in,im_addr,reset);
nR_Memory DataMemory(realClk,regs_out1,dm_out,ULA_out,ULA_out,
    ctrl_ioc[1],ctrl_ioc[0],1'b0);
nR_MUX2 NextMUX(nxtmux_in0,nxtmux_in1,nxtmux_sel,nxt_in);
nR_FluxCtrl FluxControl(branchAND_out,ctrl_jmp,nxtmux_sel);
nR_AND2 FluxAND(ctrl_brc,ULA_Zer0,branchAND_out);
nR_OR4 AdderOr(reset,ctrl_hlt,haltScape,1'b0,adder_input);
nR_Adder Adder(im_addr,{7'b0,adder_input},nxtmux_in0);
nR_BitShifter BitShift(clk,bitshift_out0,realClk,1'b0);
nR_8AND2 Ghost0_And(im_out,bitshift_out0,ghost0_in);
nR_8AND2 Ghost1_And(im_out,realClk,ghost1_in);
nR_Register Ghost0(clk,ghost0_in,ghost0_out,reset);
nR_Register Ghost1(clk,ghost1_in,ghost1_out,reset);
nR_MUX2 CtrlMUX1(ctrlmux1_in0,regs_out2,ctrl_brc,nxtmux_in1);
nR_MUX2 CtrlMUX2(ghost1_out,regs_out1,ctrlmux2_sel,ctrlmux1_in0);
nR_BitCmp Ctrl2Cmp(ghost1_out,ctrl2Cmp_out);
nR_NOT Ctrl2Not(ctrl2Cmp_out,ctrlmux2_sel);
nR_Control Control(realClk,ghost0_out[7:4],ctrl_tmpwr,ctrl_hlt,
    ctrl_jmp,ctrl_brc,ctrl_rgw,ctrl_ioc,ctrl_rgr,ctrl_alo,
    ctrl_ala);
nR_MUX2 RegWriteMUX(8'b00000010,{4'b0,ghost0_out[3:0]},ctrl_tmpwr
    ,regs_adrIn0);
nR_RegisterBank Registers(realClk,regs_in0,regs_out0,regs_out1,
    regs_out2,regs_adrIn0,ghost1_out[7:4],regs_adrOut1,ghost1_out
    [3:0],ctrl_ioc[3],ctrl_ioc[2],reset,regs_ovrflw);
nR_MUX2 AdrRead1MUX({4'b0,ghost0_out[3:0]},{4'b0,ghost1_out
    [3:0]},ctrl_rgr,regs_adrOut1);
nR_MUX4 WriteMUX(ULA_out,nxtmux_in0,dm_out,ghost1_out,ctrl_rgw,
    regs_in0);
nR_MUX2 ALUMUX(regs_out1,{4'b0,ghost1_out[3:0]},ctrl_ala,ULA_in1)
    ;
nR_ALU ULALA(regs_out0,ULA_in1,ctrl_alo,ULA_Zer0,ULA_out,
    regs_ovrflw);
nR_BitCmp HaltCmp(regs_out1,haltNOTscape);
nR_NOT HaltNot(haltNOTscape,haltScape);

always@(reset) begin
  DataMemory.m_mem[0]<=8'b00000001;
   DataMemory.m_mem[1]<=8'b00101100;
   DataMemory.m_mem[2]<=8'b00000000;
   DataMemory.m_mem[3]<=8'b10010110;
   DataMemory.m_mem[4]<=8'b00000010;
   DataMemory.m_mem[5]<=8'b10011010;
   DataMemory.m_mem[6]<=8'b00000001;
   DataMemory.m_mem[7]<=8'b01100101;
   DataMemory.m_mem[8]<=8'b00000000;
   DataMemory.m_mem[9]<=8'b11011100;
   DataMemory.m_mem[10]<=8'b00000001;
   DataMemory.m_mem[11]<=8'b11100000;
   DataMemory.m_mem[12]<=8'b00000001;
   DataMemory.m_mem[13]<=8'b00010100;
   DataMemory.m_mem[14]<=8'b00000010;
   DataMemory.m_mem[15]<=8'b10011010;
   DataMemory.m_mem[16]<=8'b00000001;
   DataMemory.m_mem[17]<=8'b00000001;
   DataMemory.m_mem[18]<=8'b00000010;
   DataMemory.m_mem[19]<=8'b00000010;
   DataMemory.m_mem[20]<=8'b00001000;
```

```
      DataMemory.m_mem[21]<=8'b00101000;
      DataMemory.m_mem[22]<=8'b00011010;
      DataMemory.m_mem[23]<=8'b00011100;
      DataMemory.m_mem[24]<=8'b00010000;
  end

initial begin
#1 begin
//data seg
      DataMemory.m_mem[0]<=8'b00000001;
      DataMemory.m_mem[1]<=8'b00101100;
      DataMemory.m_mem[2]<=8'b00000000;
      DataMemory.m_mem[3]<=8'b10010110;
      DataMemory.m_mem[4]<=8'b00000010;
      DataMemory.m_mem[5]<=8'b10011010;
      DataMemory.m_mem[6]<=8'b00000001;
      DataMemory.m_mem[7]<=8'b01100101;
      DataMemory.m_mem[8]<=8'b00000000;
      DataMemory.m_mem[9]<=8'b11011100;
      DataMemory.m_mem[10]<=8'b00000001;
      DataMemory.m_mem[11]<=8'b11100000;
      DataMemory.m_mem[12]<=8'b00000001;
      DataMemory.m_mem[13]<=8'b00010100;
      DataMemory.m_mem[14]<=8'b00000010;
      DataMemory.m_mem[15]<=8'b10011010;
      DataMemory.m_mem[16]<=8'b00000001;
      DataMemory.m_mem[17]<=8'b00000001;
      DataMemory.m_mem[18]<=8'b00000010;
      DataMemory.m_mem[19]<=8'b00000010;
      DataMemory.m_mem[20]<=8'b00001000;
      DataMemory.m_mem[21]<=8'b00101000;
      DataMemory.m_mem[22]<=8'b00011010;
      DataMemory.m_mem[23]<=8'b00011100;
      DataMemory.m_mem[24]<=8'b00010000;

//instructions seg
      InstructionsMemory.m_mem[0]<=8'b11000111;
      InstructionsMemory.m_mem[1]<=8'b00010001;
      InstructionsMemory.m_mem[2]<=8'b11001000;
      InstructionsMemory.m_mem[3]<=8'b00010101;
      InstructionsMemory.m_mem[4]<=8'b11100111;
      InstructionsMemory.m_mem[5]<=8'b00000111;
      InstructionsMemory.m_mem[6]<=8'b11101000;
      InstructionsMemory.m_mem[7]<=8'b00001000;
      InstructionsMemory.m_mem[8]<=8'b11010010;
      InstructionsMemory.m_mem[9]<=8'b10001010;
      InstructionsMemory.m_mem[10]<=8'b10110010;
      InstructionsMemory.m_mem[11]<=8'b00000000;
      InstructionsMemory.m_mem[12]<=8'b00110100;
      InstructionsMemory.m_mem[13]<=8'b00100000;
      InstructionsMemory.m_mem[14]<=8'b00111011;
      InstructionsMemory.m_mem[15]<=8'b01010000;
      InstructionsMemory.m_mem[16]<=8'b00111100;
      InstructionsMemory.m_mem[17]<=8'b01100000;
      InstructionsMemory.m_mem[18]<=8'b11000111;
      InstructionsMemory.m_mem[19]<=8'b00010010;
      InstructionsMemory.m_mem[20]<=8'b11001000;
      InstructionsMemory.m_mem[21]<=8'b00010110;
      InstructionsMemory.m_mem[22]<=8'b11100111;
      InstructionsMemory.m_mem[23]<=8'b00000111;
      InstructionsMemory.m_mem[24]<=8'b11101000;
      InstructionsMemory.m_mem[25]<=8'b00001000;
```

```
InstructionsMemory.m_mem[26]<=8'b11010010;
InstructionsMemory.m_mem[27]<=8'b10001010;
InstructionsMemory.m_mem[28]<=8'b10110010;
InstructionsMemory.m_mem[29]<=8'b00000000;
InstructionsMemory.m_mem[30]<=8'b00110100;
InstructionsMemory.m_mem[31]<=8'b00100000;
InstructionsMemory.m_mem[32]<=8'b00111011;
InstructionsMemory.m_mem[33]<=8'b01011011;
InstructionsMemory.m_mem[34]<=8'b00111100;
InstructionsMemory.m_mem[35]<=8'b01101100;
InstructionsMemory.m_mem[36]<=8'b11000111;
InstructionsMemory.m_mem[37]<=8'b00010011;
InstructionsMemory.m_mem[38]<=8'b11001000;
InstructionsMemory.m_mem[39]<=8'b00010111;
InstructionsMemory.m_mem[40]<=8'b11100111;
InstructionsMemory.m_mem[41]<=8'b00000111;
InstructionsMemory.m_mem[42]<=8'b11101000;
InstructionsMemory.m_mem[43]<=8'b00001000;
InstructionsMemory.m_mem[44]<=8'b11010010;
InstructionsMemory.m_mem[45]<=8'b10001010;
InstructionsMemory.m_mem[46]<=8'b10110010;
InstructionsMemory.m_mem[47]<=8'b00000000;
InstructionsMemory.m_mem[48]<=8'b00110100;
InstructionsMemory.m_mem[49]<=8'b00100000;
InstructionsMemory.m_mem[50]<=8'b00111011;
InstructionsMemory.m_mem[51]<=8'b01011011;
InstructionsMemory.m_mem[52]<=8'b00111100;
InstructionsMemory.m_mem[53]<=8'b01101100;
InstructionsMemory.m_mem[54]<=8'b11000111;
InstructionsMemory.m_mem[55]<=8'b00010100;
InstructionsMemory.m_mem[56]<=8'b11001000;
InstructionsMemory.m_mem[57]<=8'b00011000;
InstructionsMemory.m_mem[58]<=8'b11100111;
InstructionsMemory.m_mem[59]<=8'b00000111;
InstructionsMemory.m_mem[60]<=8'b11101000;
InstructionsMemory.m_mem[61]<=8'b00001000;
InstructionsMemory.m_mem[62]<=8'b11010010;
InstructionsMemory.m_mem[63]<=8'b10001010;
InstructionsMemory.m_mem[64]<=8'b10110010;
InstructionsMemory.m_mem[65]<=8'b00000000;
InstructionsMemory.m_mem[66]<=8'b00110100;
InstructionsMemory.m_mem[67]<=8'b00100000;
InstructionsMemory.m_mem[68]<=8'b00111100;
InstructionsMemory.m_mem[69]<=8'b01101100;
InstructionsMemory.m_mem[70]<=8'b00111011;
InstructionsMemory.m_mem[71]<=8'b01011011;
InstructionsMemory.m_mem[72]<=8'b11011101;
InstructionsMemory.m_mem[73]<=8'b00000000;
InstructionsMemory.m_mem[74]<=8'b11001110;
InstructionsMemory.m_mem[75]<=8'b00000000;
InstructionsMemory.m_mem[76]<=8'b11010010;
InstructionsMemory.m_mem[77]<=8'b00001000;
InstructionsMemory.m_mem[78]<=8'b10000010;
InstructionsMemory.m_mem[79]<=8'b11010010;
InstructionsMemory.m_mem[80]<=8'b11010010;
InstructionsMemory.m_mem[81]<=8'b10000100;
InstructionsMemory.m_mem[82]<=8'b00010010;
InstructionsMemory.m_mem[83]<=8'b00000010;
InstructionsMemory.m_mem[84]<=8'b11010010;
InstructionsMemory.m_mem[85]<=8'b00000001;
InstructionsMemory.m_mem[86]<=8'b01000011;
InstructionsMemory.m_mem[87]<=8'b00110010;
```

```
InstructionsMemory.m_mem[88]<=8'b11111101;
InstructionsMemory.m_mem[89]<=8'b00000011;
InstructionsMemory.m_mem[90]<=8'b11010010;
InstructionsMemory.m_mem[91]<=8'b00000001;
InstructionsMemory.m_mem[92]<=8'b01000011;
InstructionsMemory.m_mem[93]<=8'b00110010;
InstructionsMemory.m_mem[94]<=8'b11111110;
InstructionsMemory.m_mem[95]<=8'b00000011;
InstructionsMemory.m_mem[96]<=8'b11101101;
InstructionsMemory.m_mem[97]<=8'b00001110;
InstructionsMemory.m_mem[98]<=8'b11101110;
InstructionsMemory.m_mem[99]<=8'b00011110;
InstructionsMemory.m_mem[100]<=8'b01001101;
InstructionsMemory.m_mem[101]<=8'b11001101;
InstructionsMemory.m_mem[102]<=8'b01001110;
InstructionsMemory.m_mem[103]<=8'b10111110;
InstructionsMemory.m_mem[104]<=8'b00110010;
InstructionsMemory.m_mem[105]<=8'b11011110;
InstructionsMemory.m_mem[106]<=8'b11101110;
InstructionsMemory.m_mem[107]<=8'b00000011;
InstructionsMemory.m_mem[108]<=8'b11010010;
InstructionsMemory.m_mem[109]<=8'b00000001;
InstructionsMemory.m_mem[110]<=8'b00110011;
InstructionsMemory.m_mem[111]<=8'b00110010;
InstructionsMemory.m_mem[112]<=8'b11101101;
InstructionsMemory.m_mem[113]<=8'b00000011;
InstructionsMemory.m_mem[114]<=8'b11010010;
InstructionsMemory.m_mem[115]<=8'b00000001;
InstructionsMemory.m_mem[116]<=8'b00110011;
InstructionsMemory.m_mem[117]<=8'b00110010;
InstructionsMemory.m_mem[118]<=8'b11010010;
InstructionsMemory.m_mem[119]<=8'b10000100;
InstructionsMemory.m_mem[120]<=8'b00010000;
InstructionsMemory.m_mem[121]<=8'b00100010;
InstructionsMemory.m_mem[122]<=8'b11010010;
InstructionsMemory.m_mem[123]<=8'b00000001;
InstructionsMemory.m_mem[124]<=8'b00111101;
InstructionsMemory.m_mem[125]<=8'b11010010;
InstructionsMemory.m_mem[126]<=8'b11010010;
InstructionsMemory.m_mem[127]<=8'b00000010;
InstructionsMemory.m_mem[128]<=8'b00111110;
InstructionsMemory.m_mem[129]<=8'b11100010;
InstructionsMemory.m_mem[130]<=8'b10110000;
InstructionsMemory.m_mem[131]<=8'b01001100;
InstructionsMemory.m_mem[132]<=8'b11010010;
InstructionsMemory.m_mem[133]<=8'b00000001;
InstructionsMemory.m_mem[134]<=8'b00111101;
InstructionsMemory.m_mem[135]<=8'b11010010;
InstructionsMemory.m_mem[136]<=8'b00000001;
InstructionsMemory.m_mem[137]<=8'b00000000;
InstructionsMemory.m_mem[138]<=8'b11010010;
InstructionsMemory.m_mem[139]<=8'b00000001;
InstructionsMemory.m_mem[140]<=8'b01000011;
InstructionsMemory.m_mem[141]<=8'b00110010;
InstructionsMemory.m_mem[142]<=8'b11111011;
InstructionsMemory.m_mem[143]<=8'b00000011;
InstructionsMemory.m_mem[144]<=8'b11010010;
InstructionsMemory.m_mem[145]<=8'b00000001;
InstructionsMemory.m_mem[146]<=8'b01000011;
InstructionsMemory.m_mem[147]<=8'b00110010;
InstructionsMemory.m_mem[148]<=8'b11111100;
InstructionsMemory.m_mem[149]<=8'b00000011;
```

```
InstructionsMemory.m_mem[150]<=8'b10001011;
InstructionsMemory.m_mem[151]<=8'b01111000;
InstructionsMemory.m_mem[152]<=8'b11010010;
InstructionsMemory.m_mem[153]<=8'b10100010;
InstructionsMemory.m_mem[154]<=8'b00011011;
InstructionsMemory.m_mem[155]<=8'b00000010;
InstructionsMemory.m_mem[156]<=8'b00111011;
InstructionsMemory.m_mem[157]<=8'b01110000;
InstructionsMemory.m_mem[158]<=8'b00111100;
InstructionsMemory.m_mem[159]<=8'b10000000;
InstructionsMemory.m_mem[160]<=8'b10110000;
InstructionsMemory.m_mem[161]<=8'b10100110;
InstructionsMemory.m_mem[162]<=8'b00111011;
InstructionsMemory.m_mem[163]<=8'b10000000;
InstructionsMemory.m_mem[164]<=8'b00111100;
InstructionsMemory.m_mem[165]<=8'b01110000;
InstructionsMemory.m_mem[166]<=8'b11010101;
InstructionsMemory.m_mem[167]<=8'b00000000;
InstructionsMemory.m_mem[168]<=8'b11010110;
InstructionsMemory.m_mem[169]<=8'b00000000;
InstructionsMemory.m_mem[170]<=8'b11010010;
InstructionsMemory.m_mem[171]<=8'b00000000;
InstructionsMemory.m_mem[172]<=8'b10000010;
InstructionsMemory.m_mem[173]<=8'b10110010;
InstructionsMemory.m_mem[174]<=8'b01110010;
InstructionsMemory.m_mem[175]<=8'b00100000;
InstructionsMemory.m_mem[176]<=8'b11010010;
InstructionsMemory.m_mem[177]<=8'b11000010;
InstructionsMemory.m_mem[178]<=8'b00010010;
InstructionsMemory.m_mem[179]<=8'b00000010;
InstructionsMemory.m_mem[180]<=8'b00110101;
InstructionsMemory.m_mem[181]<=8'b01011100;
InstructionsMemory.m_mem[182]<=8'b11010111;
InstructionsMemory.m_mem[183]<=8'b00000010;
InstructionsMemory.m_mem[184]<=8'b11010010;
InstructionsMemory.m_mem[185]<=8'b00000001;
InstructionsMemory.m_mem[186]<=8'b00111011;
InstructionsMemory.m_mem[187]<=8'b10110010;
InstructionsMemory.m_mem[188]<=8'b11010010;
InstructionsMemory.m_mem[189]<=8'b11010000;
InstructionsMemory.m_mem[190]<=8'b00100001;
InstructionsMemory.m_mem[191]<=8'b01110010;
InstructionsMemory.m_mem[192]<=8'b10110000;
InstructionsMemory.m_mem[193]<=8'b10101010;
InstructionsMemory.m_mem[194]<=8'b11101100;
InstructionsMemory.m_mem[195]<=8'b00000011;
InstructionsMemory.m_mem[196]<=8'b11010010;
InstructionsMemory.m_mem[197]<=8'b00000001;
InstructionsMemory.m_mem[198]<=8'b00110011;
InstructionsMemory.m_mem[199]<=8'b00110010;
InstructionsMemory.m_mem[200]<=8'b11101011;
InstructionsMemory.m_mem[201]<=8'b00000011;
InstructionsMemory.m_mem[202]<=8'b11010010;
InstructionsMemory.m_mem[203]<=8'b00000001;
InstructionsMemory.m_mem[204]<=8'b00110011;
InstructionsMemory.m_mem[205]<=8'b00110010;
InstructionsMemory.m_mem[206]<=8'b10110100;
InstructionsMemory.m_mem[207]<=8'b00000000;
InstructionsMemory.m_mem[208]<=8'b11010010;
InstructionsMemory.m_mem[209]<=8'b00000001;
InstructionsMemory.m_mem[210]<=8'b00110110;
InstructionsMemory.m_mem[211]<=8'b01100010;
```

```
      InstructionsMemory.m_mem[212]<=8'b10110000;
      InstructionsMemory.m_mem[213]<=8'b10101010;
      end
end
endmodule
```

20.1 Módulo de Teste

Para testar o processador foi feito um módulo de teste que é apenas uma instanciação do processador que lê o banco de registradores na posição da resposta final para verificar o resultado, o código desenvolvido foi:

```
module nRProcessor_Test();
      reg deviceClock;
      initial
            deviceClock<=1'b0;

      always@(*)
            #1 deviceClock<=~deviceClock;

      always@(*)
            $monitor("Posicao do plagioclasio %0d",nanoRisk.Registers
                .rb_mem[4'b1101]);
      nRProcessor nanoRisk(deviceClock,1'b0);
endmodule
```

20.2 Simulação

Durante a simulação se obteve na resposta o número 0 e as vezes 20, quando deveria ser 7. E analisando em tempo real a execução da simulação foi possível ver que alguns componentes funcionavam e outro não, aparentando ser um problema de sincronia ou clock.

21 Versão 2

21.1 Compilador

```
#include <iostream>
#include <bitset>
#include <algorithm>
#include <fstream>
#include <sstream>
#include <string>
#include <vector>
#include <map>

using namespace std;

string output = "a.out";
string input;
map<string,int> addresses;

int nRisk_StackSigal=-1;
//funcs
string nRisk_slp="0000";
string nRisk_brq="0001";
string nRisk_brf="0010";
string nRisk_add="0011";
string nRisk_sub="0100";
string nRisk_and="0101";
string nRisk_or ="0110";
string nRisk_nor="0111";
string nRisk_slt="1000";
string nRisk_sr ="1001";
string nRisk_sl ="1010";
string nRisk_jr ="1011";
string nRisk_la ="1100";
string nRisk_lc ="1101";
string nRisk_lw ="1110";
string nRisk_sw ="1111";
//regs
string nRisk_zro="0000";
string nRisk_flg="0001";
string nRisk_nt ="0010";
string nRisk_sp ="0011";
string nRisk_ra ="0100";
string nRisk_v0 ="0101";
string nRisk_v1 ="0110";
string nRisk_a0 ="0111";
string nRisk_a1 ="1000";
string nRisk_a2 ="1001";
string nRisk_a3 ="1010";
string nRisk_s0 ="1011";
string nRisk_s1 ="1100";
string nRisk_s2 ="1101";
string nRisk_s3 ="1110";
string nRisk_bp ="1111";

void error(string str){
    cout<<"error - "<<str<<"."<<endl;
}

struct nanoArg{
    string arg;
    bool isRegister;
```

```
59      int pos;
60      bool err=false;
61  };
62
63  #ifdef _WIN32
64      std::string to_string(int i){
65          std::stringstream ss;
66          ss << i;
67          return ss.str();
68      }
69  #endif
70  int strToInt(string str){
71      str.erase(remove(str.begin(),str.end(),','),str.end());//remove
            $
72      stringstream ss;
73      ss.str(str);
74      int out;
75      ss>>out;
76      return out;
77  }
78
79  bool isValidChar(char c){
80      return c=='$'||(c>=48&&c<=57)||(c>=65&&c<=90)||(c>=97&&c<=122)
            ||c=='-';
81  }
82
83  bool isBinary(string str){
84      for(int i=0;i<str.size();i++){
85          if(str[i]!='0'&&str[i]!='1'){
86              return false;
87          }
88      }
89      return true;
90  }
91  nanoArg getArgFrom(string str, int from){
92      nanoArg out;
93      int start=-1;
94      for(int j=from;j<=str.size();j++){
95          if(start<0&&isValidChar(str[j])){
96              start=j;
97          }else if(start>=0){
98              if(str[j]==','||str[j]=='\0'||str[j]=='\t'||str[j]=='('
                    ||str[j]==')'||str[j]==' '){
99                  out.pos=j+1;
100                 out.arg=str.substr(start,j-start);
101                 out.isRegister=out.arg.find("$")!=string::npos;
102                 return out;
103             }
104         }
105     }
106     out.err=true;
107     return out;
108 }
109 string translateRegister(string arg){
110     arg.erase(remove(arg.begin(),arg.end(),'$'),arg.end());//remove
            $
111     if(!arg.compare("zero")||!arg.compare("0")){
112         return nRisk_zro;
113     }else if(!arg.compare("flg")||!arg.compare("flag")){
114         return nRisk_flg;
115     }else if(!arg.compare("nt")||!arg.compare("nT")){
116         return nRisk_nt;
```

```
117     }else if(!arg.compare("sp")){
118         return nRisk_sp;
119     }else if(!arg.compare("ra")){
120         return nRisk_ra;
121     }else if(!arg.compare("v0")){
122         return nRisk_v0;
123     }else if(!arg.compare("v1")){
124         return nRisk_v1;
125     }else if(!arg.compare("a0")){
126         return nRisk_a0;
127     }else if(!arg.compare("a1")){
128         return nRisk_a1;
129     }else if(!arg.compare("a2")){
130         return nRisk_a2;
131     }else if(!arg.compare("a3")){
132         return nRisk_a3;
133     }else if(!arg.compare("s0")){
134         return nRisk_s0;
135     }else if(!arg.compare("s1")){
136         return nRisk_s1;
137     }else if(!arg.compare("s2")){
138         return nRisk_s2;
139     }else if(!arg.compare("s3")){
140         return nRisk_s3;
141     }else if(!arg.compare("bp")){
142         return nRisk_bp;
143     } else return "";
144 }
145 vector<string> translateFunc(string func,string args){
146     func.erase(remove(func.begin(),func.end(),' '),func.end());//
          remove spaces
147     args.erase(remove(args.begin(),args.end(),' '),args.end());//
          remove spaces
148     func.erase(remove(func.begin(),func.end(),'\t'),func.end());//
          remove spaces
149     args.erase(remove(args.begin(),args.end(),'\t'),args.end());//
          remove spaces
150     vector<string> out;
151     nanoArg arg0,arg1,arg2;
152     string nRisk,argRegister,target0,target1,target2;
153     if(!func.compare("slp")||!func.compare("sleep")){
154         arg0=getArgFrom(args,0);
155         target0="0001";
156         if(!arg0.err){
157             if(arg0.isRegister){
158                 argRegister=translateRegister(arg0.arg);
159                 if(argRegister!="")
160                     target0=argRegister;
161                 else{
162                     error("Register \""+arg0.arg+"\" doesnt exists"
                          );
163                     return out;
164                 }
165             }
166         }
167         out.push_back(nRisk_slp+target0);
168         out.push_back("00000000");
169         return out;
170     }else if(!func.compare("brq")||!func.compare("brf")||!func.
          compare("add")||!func.compare("sub")||!func.compare("and")
171              ||!func.compare("or")||!func.compare("nor")||!func.
                  compare("slt")||!func.compare("sr")||!func.compare
```

```
                ("sl")){
172         arg0=getArgFrom(args,0);
173         if(arg0.err){
174             error("Cannot find arg0 for \""+func+"\"");
175             return out;
176         }
177         if(!arg0.isRegister){
178             error("Arg0 for \""+func+"\" is not a register");
179             return out;
180         }
181         arg1=getArgFrom(args,arg0.pos);
182         if(arg1.err){
183             error("Cannot find arg1 for \""+func+"\"");
184             return out;
185         }
186         if(!arg1.isRegister){
187             error("Arg1 for \""+func+"\" is not a register");
188             return out;
189         }
190         arg2=getArgFrom(args,arg1.pos);
191         if(arg2.err){
192             error("Cannot find arg2 for \""+func+"\"");
193             return out;
194         }
195         if(!arg2.isRegister){
196             error("Arg2 for \""+func+"\" is not a register");
197             return out;
198         }
199         argRegister=translateRegister(arg0.arg);
200         if(argRegister==""){
201             error("Register \""+arg0.arg+"\" doesnt exists for \""+
                    func+"\"");
202             return out;
203         }
204         target0=argRegister;
205         argRegister=translateRegister(arg1.arg);
206         if(argRegister==""){
207             error("Register \""+arg1.arg+"\" doesnt exists for \""+
                    func+"\"");
208             return out;
209         }
210         target1=argRegister;
211         argRegister=translateRegister(arg2.arg);
212         if(argRegister==""){
213             error("Register \""+arg2.arg+"\" doesnt exists for \""+
                    func+"\"");
214             return out;
215         }
216         target2=argRegister;
217 
218         if(!func.compare("brq"))
219             nRisk=nRisk_brq;
220         else if(!func.compare("brf"))
221             nRisk=nRisk_brf;
222         else if(!func.compare("add"))
223             nRisk=nRisk_add;
224         else if(!func.compare("sub"))
225             nRisk=nRisk_sub;
226         else if(!func.compare("and"))
227             nRisk=nRisk_and;
228         else if(!func.compare("or"))
229             nRisk=nRisk_or;
```

```
230         else if(!func.compare("nor"))
231             nRisk=nRisk_nor;
232         else if(!func.compare("slt"))
233             nRisk=nRisk_slt;
234         else if(!func.compare("sr"))
235             nRisk=nRisk_sr;
236         else if(!func.compare("sl"))
237             nRisk=nRisk_sl;
238 
239         out.push_back(nRisk+target0);
240         out.push_back(target1+target2);
241         return out;
242     }else if(!func.compare("jr")){
243         arg0=getArgFrom(args,0);
244         if(arg0.err){
245             error("Cannot find arg0 for \""+func+"\"");
246             return out;
247         }
248         if(!arg0.isRegister){
249             error("Arg0 for \""+func+"\" is not a register");
250             return out;
251         }
252         argRegister=translateRegister(arg0.arg);
253         if(argRegister==""){
254             error("Register \""+arg0.arg+"\" doesnt exists for \""+
                    func+"\"");
255             return out;
256         }
257         target0=argRegister;
258         out.push_back(nRisk_jr+target0);
259         out.push_back("00000000");
260         return out;
261     }else if(!func.compare("j")){//SEMI INTERPRETADA
262         arg0=getArgFrom(args,0);
263         if(arg0.err){
264             error("Cannot find arg0 for \""+func+"\"");
265             return out;
266         }
267         if(arg0.isRegister){
268             error("Arg0 for \""+func+"\" is a register");
269             return out;
270         }
271         out.push_back(nRisk_jr+"0000");
272         out.push_back(arg0.arg);
273         return out;
274     }else if(!func.compare("la")||!func.compare("lc")){
275         arg0=getArgFrom(args,0);
276         if(arg0.err){
277             error("Cannot find arg0 for \""+func+"\"");
278             return out;
279         }
280         if(!arg0.isRegister){
281             error("Arg0 for \""+func+"\" is not a register");
282             return out;
283         }
284         argRegister=translateRegister(arg0.arg);
285         if(argRegister==""){
286             error("Register \""+arg0.arg+"\" doesnt exists for \""+
                    func+"\"");
287             return out;
288         }
289         target0=argRegister;
```

```
290         arg1=getArgFrom(args,arg0.pos);
291         if(arg1.err){
292             error("Cannot find arg1 for \""+func+"\"");
293             return out;
294         }
295         if(arg1.isRegister){
296             error("Arg1 for \""+func+"\" is a register");
297             return out;
298         }
299         if(!func.compare("la")){
300             out.push_back(nRisk_la+target0);
301             out.push_back(arg1.arg);
302         }else if(!func.compare("lc")){
303             if(strToInt(arg1.arg)>255){
304                 error("Constant overflow, your constant should be
                        less than 256");
305                 return out;
306             }
307             out.push_back(nRisk_lc+target0);
308             out.push_back(bitset< 8 >(strToInt(arg1.arg)).to_string
                    ());
309         }
310         return out;
311     }else if(!func.compare("lw")||!func.compare("sw")){
312         arg0=getArgFrom(args,0);
313         if(arg0.err){
314             error("Cannot find arg0 for \""+func+"\"");
315             return out;
316         }
317         if(!arg0.isRegister){
318             error("Arg0 for \""+func+"\" is not a register");
319             return out;
320         }
321         argRegister=translateRegister(arg0.arg);
322         if(argRegister==""){
323             error("Register \""+arg0.arg+"\" doesnt exists for \""+
                    func+"\"");
324             return out;
325         }
326         target0=argRegister;
327         arg1=getArgFrom(args,arg0.pos);
328         if(arg1.err){
329             error("Cannot find arg1 for \""+func+"\"");
330             return out;
331         }
332         if(arg1.isRegister){
333             error("Arg1 for \""+func+"\" is a register");
334             return out;
335         }
336         arg2=getArgFrom(args,arg1.pos);
337         if(arg2.err){
338             error("Cannot find arg2 for \""+func+"\"");
339             return out;
340         }
341         if(!arg2.isRegister){
342             error("Arg2 for \""+func+"\" is not a register");
343             return out;
344         }
345         argRegister=translateRegister(arg2.arg);
346         if(argRegister==""){
347             error("Register \""+arg2.arg+"\" doesnt exists for \""+
                    func+"\"");
```

```
348             return out;
349         }
350         target2=argRegister;
351 
352         if(!func.compare("lw"))
353             nRisk=nRisk_lw;
354         else if(!func.compare("sw"))
355             nRisk=nRisk_sw;
356         if(strToInt(arg1.arg)>15){
357             error("Offset \""+arg1.arg+"\" for \""+func+"\" should
                    be less than 16");
358             return out;
359         }
360         out.push_back(nRisk+target0);
361         out.push_back(target2+bitset< 4 >(strToInt(arg1.arg)).
                to_string());
362         return out;
363     }//INTERPRETADAS
364     else if(!func.compare("addc")||!func.compare("andc")||!func.
            compare("orc")||!func.compare("norc")||
365             !func.compare("sltc")||!func.compare("src")||!func.
                    compare("slc")||!func.compare("beq")||
366             !func.compare("bof")){
367 
368         arg0=getArgFrom(args,0);
369         if(arg0.err){
370             error("Cannot find arg0 for \""+func+"\"");
371             return out;
372         }
373         if(!arg0.isRegister){
374             error("Arg0 for \""+func+"\" is not a register");
375             return out;
376         }
377         arg1=getArgFrom(args,arg0.pos);
378         if(arg1.err){
379             error("Cannot find arg1 for \""+func+"\"");
380             return out;
381         }
382         if(!arg1.isRegister){
383             error("Arg1 for \""+func+"\" is not a register");
384             return out;
385         }
386         arg2=getArgFrom(args,arg1.pos);
387         if(arg2.err){
388             error("Cannot find arg2 for \""+func+"\"");
389             return out;
390         }
391         if(arg2.isRegister){
392             error("Arg2 for \""+func+"\" is a register");
393             return out;
394         }
395         argRegister=translateRegister(arg0.arg);
396         if(argRegister==""){
397             error("Register \""+arg0.arg+"\" doesnt exists for \""+
                    func+"\"");
398             return out;
399         }
400         target0=argRegister;
401         argRegister=translateRegister(arg1.arg);
402         if(argRegister==""){
403             error("Register \""+arg1.arg+"\" doesnt exists for \""+
                    func+"\"");
```

```
404             return out;
405         }
406         target1=argRegister;
407         if(func.compare("beq")&&func.compare("bof"))
408         if(strToInt(arg2.arg)>255){
409             error("Constant overflow, your constant should be less
                    than 256");
410             return out;
411         }
412         out.push_back(nRisk_lc+nRisk_nt);
413         if(!func.compare("beq")||!func.compare("bof")){
414             out.push_back(arg2.arg);
415         }else{
416             out.push_back(bitset< 8 >(strToInt(arg2.arg)).to_string
                    ());
417         }
418         if(!func.compare("addc"))
419             nRisk=nRisk_add;
420         else if(!func.compare("andc"))
421             nRisk=nRisk_and;
422         else if(!func.compare("orc"))
423             nRisk=nRisk_or;
424         else if(!func.compare("norc"))
425             nRisk=nRisk_nor;
426         else if(!func.compare("sltc"))
427             nRisk=nRisk_slt;
428         else if(!func.compare("src"))
429             nRisk=nRisk_sr;
430         else if(!func.compare("slc"))
431             nRisk=nRisk_sl;
432         else if(!func.compare("beq"))
433             nRisk=nRisk_brq;
434         else if(!func.compare("bof"))
435             nRisk=nRisk_brf;
436         out.push_back(nRisk+target0);
437         out.push_back(target1+nRisk_nt);
438         return out;
439     }else if(!func.compare("mov")||!func.compare("move")||!func.
        compare("not")){
440         arg0=getArgFrom(args,0);
441         if(arg0.err){
442             error("Cannot find arg0 for \""+func+"\"");
443             return out;
444         }
445         if(!arg0.isRegister){
446             error("Arg0 for \""+func+"\" is not a register");
447             return out;
448         }
449         arg1=getArgFrom(args,arg0.pos);
450         if(arg1.err){
451             error("Cannot find arg1 for \""+func+"\"");
452             return out;
453         }
454         if(!arg1.isRegister){
455             error("Arg1 for \""+func+"\" is not a register");
456             return out;
457         }
458         argRegister=translateRegister(arg0.arg);
459         if(argRegister==""){
460             error("Register \""+arg0.arg+"\" doesnt exists for \""+
                    func+"\"");
461             return out;
```

```
462         }
463         target0=argRegister;
464         argRegister=translateRegister(arg1.arg);
465         if(argRegister==""){
466             error("Register \""+arg1.arg+"\" doesnt exists for \""+
                    func+"\"");
467             return out;
468         }
469         target1=argRegister;
470 
471         if(!func.compare("mov")||!func.compare("move"))
472             nRisk=nRisk_add;
473         else if(!func.compare("not"))
474             nRisk=nRisk_nor;
475 
476 
477         out.push_back(nRisk+target0);
478         out.push_back(target1+nRisk_zro);
479         return out;
480 
481     }else if(!func.compare("jal")||!func.compare("jrl")){
482         if(!func.compare("jal")){
483             arg0=getArgFrom(args,0);
484             if(arg0.err){
485                 error("Cannot find arg0 for \""+func+"\"");
486                 return out;
487             }
488             if(arg0.isRegister){
489                 error("Arg0 for \""+func+"\" is a register");
490                 return out;
491             }
492             target0=nRisk_nt;
493             out.push_back(nRisk_lc+target0);
494             out.push_back(arg0.arg);
495         }else{
496             arg0=getArgFrom(args,0);
497             if(arg0.err){
498                 error("Cannot find arg0 for \""+func+"\"");
499                 return out;
500             }
501             if(!arg0.isRegister){
502                 error("Arg0 for \""+func+"\" is not a register");
503                 return out;
504             }
505             argRegister=translateRegister(arg0.arg);
506             if(argRegister==""){
507                 error("Register \""+arg0.arg+"\" doesnt exists for
                        \""+func+"\"");
508                 return out;
509             }
510             target0=argRegister;
511         }
512         out.push_back(nRisk_jr+target0);
513         out.push_back("00000000");
514         out.push_back(nRisk_add+nRisk_ra);
515         out.push_back(nRisk_nt+nRisk_zro);
516         return out;
517     }else if(!func.compare("push")||!func.compare("pop")){
518         arg0=getArgFrom(args,0);
519         if(arg0.err){
520             error("Cannot find arg0 for \""+func+"\"");
521             return out;
```

```
522         }
523         if(!arg0.isRegister){
524             error("Arg0 for \""+func+"\" is not a register");
525             return out;
526         }
527         argRegister=translateRegister(arg0.arg);
528         if(argRegister==""){
529             error("Register \""+arg0.arg+"\" doesnt exists for \""+
                    func+"\"");
530             return out;
531         }
532         target0=argRegister;
533         if(nRisk_StackSigal>0){
534             if(!func.compare("push")){
535                 //escreve soma
536                 out.push_back(nRisk_sw+target0);
537                 out.push_back(nRisk_sp+"0000");
538                 out.push_back(nRisk_lc+nRisk_nt);
539                 out.push_back(bitset< 8 >(1).to_string());
540                 out.push_back(nRisk_add+nRisk_sp);
541                 out.push_back(nRisk_sp+nRisk_nt);
542             }else{
543                 //subtrai le
544                 out.push_back(nRisk_lc+nRisk_nt);
545                 out.push_back(bitset< 8 >(1).to_string());
546                 out.push_back(nRisk_sub+nRisk_sp);
547                 out.push_back(nRisk_sp+nRisk_nt);
548                 out.push_back(nRisk_lw+target0);
549                 out.push_back(nRisk_sp+"0000");
550             }
551         }else{
552             if(!func.compare("push")){
553                 //subtrai escreve
554                 out.push_back(nRisk_lc+nRisk_nt);
555                 out.push_back(bitset< 8 >(1).to_string());
556                 out.push_back(nRisk_sub+nRisk_sp);
557                 out.push_back(nRisk_sp+nRisk_nt);
558                 out.push_back(nRisk_sw+target0);
559                 out.push_back(nRisk_sp+"0000");
560             }else{
561                 //le soma
562                 out.push_back(nRisk_lw+target0);
563                 out.push_back(nRisk_sp+"0000");
564                 out.push_back(nRisk_lc+nRisk_nt);
565                 out.push_back(bitset< 8 >(1).to_string());
566                 out.push_back(nRisk_add+nRisk_sp);
567                 out.push_back(nRisk_sp+nRisk_nt);
568             }
569         }
570         return out;
571     }else{
572         error("Function \""+func+"\" doesnt exists");
573         return out;
574     }
575 }
576 
577 char strToChar(string a){
578     char out=0;
579     int pow2[8]={1,2,4,8,16,32,64,128};
580     if(a.size()<8)
581         return 0;
582     for(int i=0;i<8;i++)
```

```
583         out+=pow2[7-i]*a[i];
584     return out;
585 }
586 
587 string appendName(string a, string b){
588     string p1="";
589     string p2="";
590     bool dot=false;
591     for(int i=0;i<a.size();i++){
592         if(a[i]!='.'&&!dot)
593             p1+=a[i];
594         else{
595             dot=true;
596             p2+=a[i];
597         }
598     }
599     return p1+b+p2;
600 }
601 
602 int main(int argc, char **argv){
603     bool separateFiles=false;
604     bool coment=false;
605     bool verilog=false;
606     bool binary=false;
607     string verilog0,verilog1;
608     if(argc<2){
609         error("No such file or arguments");
610         return -1;
611     }
612     input=argv[1];
613     for(int i=2;i<argc;i++){
614         if(argv[i][0]=='-'&&tolower(argv[i][1])=='o'){
615             if(i+1>=argc){
616                 error("Few arguments for -o");
617                 return -1;
618             }
619             output=argv[++i];
620         }
621         if(argv[i][0]=='-'&&tolower(argv[i][1])=='s'){
622             separateFiles=true;
623         }
624         if(argv[i][0]=='-'&&tolower(argv[i][1])=='c'){
625             coment=true;
626         }
627         if(argv[i][0]=='-'&&tolower(argv[i][1])=='b'){
628             binary=true;
629         }
630         if(argv[i][0]=='-'&&tolower(argv[i][1])=='v'){
631             if(i+2>=argc){
632                 error("Few arguments for -v");
633                 return -1;
634             }
635             verilog=true;
636             verilog0=argv[++i];
637             verilog1=argv[++i];
638         }
639     }
640     string line;
641     vector<string> code;
642     ifstream codeFile (input);
643     if (codeFile.is_open()){
644         while (getline (codeFile,line)){
```

```
645             code.push_back(line);
646         }
647         codeFile.close();
648     }else{
649         error("Unable to reach file");
650         return -1;
651     }
652     vector<string> machineCode;
653     vector<string> dataSeg;
654     string dataSegSize;
655     bool data=false;
656     bool text=false;
657     int commentedLines=0;
658     for(int i=0;i<code.size();i++){
659         if(code[i][0]=='#'||code[i][0]=='\n'){
660             code.erase(code.begin()+i);
661             i--;
662             commentedLines++;
663         }else{
664             int foundAt=code[i].find("#");//size
665             if(foundAt!=string::npos){
666                 code[i].erase(code[i].begin()+foundAt,code[i].end()
                        );
667             }
668         }
669         int fix0=code[i].find("slp");
670         int fix1=code[i].find("sleep");
671         if(fix0!=string::npos||fix1!=string::npos){
672             int rg=code[i].find("$");
673             if(rg==string::npos){
674                 code[i]="slp $flg";
675             }
676         }
677         /* TODO remover linhas inuteis
678         bool onlySpace=true;
679         for(int j=0;j<code[i].size();j++){
680             if(!(j==' '||j=='\t'||j=='\n'||j=='\b')){
681                 onlySpace=false;
682                 break;
683             }
684         }
685         if(onlySpace){
686             code.erase(code.begin()+i);
687         }
688         */
689     }
690     int currentAddress=0;
691     int currentDataAddress=0;
692     int doubleDotAt;
693     for(int i=0;i<code.size();i++){
694         if(code[i].find(".data")!=string::npos){
695             data=true;
696         }else if(code[i].find(".text")!=string::npos){
697             text=true;
698         }else if(data&&!text){//DATA PART
699             doubleDotAt=code[i].find(":");//size
700             if(doubleDotAt==string::npos)
701                 doubleDotAt=0;
702             else{
703                 string tmp=code[i].substr(0,doubleDotAt);
704                 code[i]=code[i].substr(doubleDotAt+1);
```

```
705                 tmp.erase(remove(tmp.begin(),tmp.end(),' '),tmp.end
                        ());
706                 tmp.erase(remove(tmp.begin(),tmp.end(),'\t'),tmp.
                        end());
707                 addresses[tmp]=currentDataAddress;
708             }
709             code[i]=code[i].substr(code[i].find("."));
710             string varType=code[i].substr(0,code[i].find(" "));
711             if(!varType.compare(".word")){
712                 code[i]=code[i].substr(code[i].find(" "));
713                 code[i].erase(remove(code[i].begin(),code[i].end(),
                        ' '),code[i].end());//remove spaces
714                 code[i].erase(remove(code[i].begin(),code[i].end(),
                        '\t'),code[i].end());//remove spaces
715                 int last=0;
716                 for(int j=0;j<=code[i].size();j++){
717                     if(code[i][j]==','||code[i][j]=='\0'){
718                         dataSeg.push_back(bitset< 8 >(strToInt(code
                                [i].substr(last,j-last))).to_string());
719                         last=j;
720                         currentDataAddress++;
721                     }
722                 }
723             }
724         }else if(text){//INSTRUCTIONS PART
725             doubleDotAt=code[i].find(":");//size
726             if(doubleDotAt==string::npos)
727                 doubleDotAt=0;
728             else{
729                 string tmp=code[i].substr(0,doubleDotAt);
730                 tmp.erase(remove(tmp.begin(),tmp.end(),' '),tmp.end
                        ());
731                 tmp.erase(remove(tmp.begin(),tmp.end(),'\t'),tmp.
                        end());
732                 addresses[tmp]=currentAddress;
733             }
734             int start=-1;
735             for(int j=doubleDotAt;j<code[i].size();j++){
736                 if(start<0&&isValidChar(code[i][j])){
737                     start=j;
738                 }else if(start>=0){
739                     if(code[i][j]==' '||code[i][j]=='\0'||code[i][j
                            ]=='\t'){
740                         vector<string> translated=translateFunc(
                                code[i].substr(start,j-start),code[i].
                                substr(j+1));
741                         if(translated.size()!=0){
742                             for(int t=0;t<translated.size();t++){
743                                 machineCode.push_back(translated[t
                                        ]);
744                                 currentAddress++;
745                             }
746                         }else{
747                             error("Translating line ("+to_string(i+
                                    commentedLines)+")");
748                             return -1;
749                         }
750                          break;
751                     }
752                 }
753             }
754
```

```
755         }
756     }
757     for(int i=1;i<machineCode.size();i+=2){
758         if(machineCode[i].size()!=8||!isBinary(machineCode[i])){
759             if(addresses.find(machineCode[i]) == addresses.end()){
760                 error("Cannot jump to \""+machineCode[i]+"\", tag
                        doesnt exists");
761                 return -1;
762             }else{
763                 if(addresses[machineCode[i]]>255){
764                     error("Address overflow("+to_string(addresses[
                            machineCode[i]])+"), your code is too big")
                            ;
765                     return -1;
766                 }
767                 //cout<<"addresses:"<<machineCode[i]<<" = "<<
                        addresses[machineCode[i]]<<endl;
768                 machineCode[i]=bitset< 8 >(addresses[machineCode[i
                        ]]).to_string();
769             }
770         }
771     }
772     if(binary){
773         if(separateFiles){
774             ofstream dataFile(appendName(output,"_data"),ios::
                    binary|ios::out);
775             if(dataFile.is_open()){
776                 for(int i=0;i<dataSeg.size();i++){
777                     char c=strToChar(dataSeg[i]);
778                     dataFile.write(&c, sizeof(c));
779                 }
780                 dataFile.close();
781             }else{
782                 error("Unable to create file");
783                 return -1;
784             }
785             ofstream machineFile(appendName(output,"_inst"),ios::
                    binary|ios::out);
786             if(machineFile.is_open()){
787                 for(int i=0;i<machineCode.size();i++){
788                     char c=strToChar(machineCode[i]);
789                     machineFile.write(&c, sizeof(c));
790                 }
791                 machineFile.close();
792             }else{
793                 error("Unable to create file");
794                 return -1;
795             }
796         }else{
797             ofstream outFile(output,ios::binary|ios::out);
798             if(outFile.is_open()){
799                 for(int i=0;i<dataSeg.size();i++){
800                     char c=strToChar(dataSeg[i]);
801                     outFile.write(&c, sizeof(c));
802                 }
803                 for(int i=0;i<machineCode.size();i++){
804                     char c=strToChar(machineCode[i]);
805                     outFile.write(&c, sizeof(c));
806                 }
807                 outFile.close();
808             }else{
809                 error("Unable to create file");
```

```
                return -1;
            }
        }
    }else{
        if(verilog){
            if(separateFiles){
                ofstream dataFile (appendName(output,"_data"));
                if(dataFile.is_open()){
                    if(coment)
                        dataFile<<"//data\n";
                    for(int i=0;i<dataSeg.size();i++)
                        dataFile<<verilog0<<"["<<i<<"]=8\'b"<<
                            dataSeg[i]<<";"<<endl;
                    dataFile.close();
                }else{
                    error("Unable to create file");
                    return -1;
                }
                ofstream machineFile (appendName(output,"_inst"));
                if(machineFile.is_open()){
                    if(coment)
                        machineFile<<"//instructions\n";
                    for(int i=0;i<machineCode.size();i++)
                        machineFile<<verilog1<<"["<<(i)<<"]=8\'b"<<
                            machineCode[i]<<";"<<endl;
                    machineFile.close();
                }else{
                    error("Unable to create file");
                    return -1;
                }
            }else{
                ofstream outFile (output);
                if(outFile.is_open()){
                    if(coment)
                        outFile<<"//data\n";
                    for(int i=0;i<dataSeg.size();i++)
                        outFile<<verilog0<<"["<<i<<"]=8\'b"<<
                            dataSeg[i]<<";"<<endl;
                    if(coment)
                        outFile<<"//instructions\n";
                    int offset=0;
                    if(verilog0==verilog1)
                        offset=dataSeg.size();
                    for(int i=0;i<machineCode.size();i++)
                        outFile<<verilog1<<"["<<(i+offset)<<"]=8\'b
                            "<<machineCode[i]<<";"<<endl;
                    outFile.close();
                }else{
                    error("Unable to create file");
                    return -1;
                }
            }
        }else{
            if(separateFiles){
                if(coment){
                    machineCode[0]+="//instructions\n";
                    dataSeg[0]+="//data\n";
                }
                ofstream dataFile (appendName(output,"_data"));
                if(dataFile.is_open()){
                    for(int i=0;i<dataSeg.size();i++){
                        dataFile<<dataSeg[i]<<" ";
```

```
868                         if((i+1)%2==0&&i+1<dataSeg.size())
869                             dataFile<<endl;
870                     }
871                     dataFile.close();
872                 }else{
873                     error("Unable to create file");
874                     return -1;
875                 }
876 
877                 ofstream machineFile (appendName(output,"_inst"));
878                 if(machineFile.is_open()){
879                     for(int i=0;i<machineCode.size();i++){
880                         machineFile<<machineCode[i]<<" ";
881                         if((i+1)%2==0&&i+1<machineCode.size())
882                             machineFile<<endl;
883                     }
884                     machineFile.close();
885                 }else{
886                     error("Unable to create file");
887                     return -1;
888                 }
889             }else{
890                 dataSegSize=bitset< 8 >(dataSeg.size()).to_string()
                        ;
891                 if(coment){
892                     dataSegSize+="//data seg size\n";
893                     machineCode[0]+="//instructions\n";
894                     dataSeg[0]+="//data\n";
895                 }
896                 for(int i=0;i<dataSeg.size();i++){
897                     machineCode.insert(machineCode.begin()+i,
                            dataSeg[i]);
898                 }
899                 machineCode.insert(machineCode.begin(), dataSegSize
                        );
900                 ofstream outFile (output);
901                 if(outFile.is_open()){
902                     for(int i=0;i<machineCode.size();i++){
903                         outFile<<machineCode[i]<<" ";
904                         if((i+1)%2==0&&i+1<machineCode.size())
905                             outFile<<endl;
906                     }
907                     outFile.close();
908                 }else{
909                     error("Unable to create file");
910                     return -1;
911                 }
912             }
913         }
914     }
915     return 0;
916 }
```

21.2 Software

```
.data
#real data
#v:      .word 300,  150,  666,  357,  220,  480,  276,
666          #OVERFLOW
#v array size 16 with 8 elements
#overflow(simple)(signed)
```

```
#v:      .word 2,44, 1,22,  5,26,  2,101, 1,92,  3,96,  2,20, 5,26
#overflow(unsigned second byte) doesnt allow neg numbers
v:       .word 1,44, 0,150, 2,154, 1,101, 0,220, 1,224, 1,20, 2,154
nCa:     .word 1
nAl:     .word 2
nSi:     .word 2
nO:      .word 8
mCa:     .word 40
mAl:     .word 26
mSi:     .word 28
mO:      .word 16
.text
#main func
main:
la $a0, nCa
la $a1, mCa
lw $a0, 0($a0)
lw $a1, 0($a1)
jal mult
mov $s0,$v0
mov $s1,$v1
la $a0, nAl
la $a1, mAl
lw $a0, 0($a0)
lw $a1, 0($a1)
jal mult
add $s0,$v0,$s0
add $s1,$v1,$s1
la $a0, nSi
la $a1, mSi
lw $a0, 0($a0)
lw $a1, 0($a1)
jal mult
add $s0,$v0,$s0
add $s1,$v1,$s1
la $a0, nO
la $a1, mO
lw $a0, 0($a0)
lw $a1, 0($a1)
jal mult
add $s1,$v1,$s1          #+sig massa molar de CaAl2Si2O8
add $s0,$v0,$s0          #-sig massa molar de CaAl2Si2O8
lc $a0, 2        #overflow negative
bof $flg, $a0, over
j overend
over:
addc $s1, $s1, 1
overend:
mov $flg, $0
```

```
lc $s2, 0
la $s3, v
mainloop:
sltc $a0, $s2, 8          #if(s2<8) nt=true else nt=false
beq $a0, $0, mainend      #erro por usar nt

lw $a2, 0($s3)  #vector at i value
lw $a1, 1($s3)  #vector at i value
sub $a2,$a2,$s1           #mais significativo
sub $a1,$a1,$s0           #menos significativo
add $a0,$a2,$a1

beq $0,$a0,mainend #igual
addc $s2,$s2,1
addc $s3,$s3,2  #nao ha alinhamento de memoria
j mainloop
mainend:
addc $s2, $s2, 1                  #posicao do plagioclasio(primeira posicao e 1)
slp                       # we are out of here.

#mult func(a*b=c)
mult:
push $s0
push $s1
slt $s0, $a0, $a1         #if(a0<a1) t0=true else t0=false
beq $s0, $0, multp2
mov $s0, $a0 #a0 lower
mov $s1, $a1
j multp3
multp2:                   #a1 lower
mov $s0, $a1 #a0 lower
mov $s1, $a0
multp3:                   #always s0<s1
lc $v0, 0
lc $v1, 0
multloop:

mov $flg, $0

        # erro
        # sltc $nt, $s0, 0 #if(s0<0) nt=true else nt=false
        # not $nt, $nt
        # beq $nt, $zero, multend
        # end erro

        beq $s0, $zero, multend

add $v0, $v0, $s1
```

```
lc $a0, 2          #overflow negative
addc $s0, $s0, -1
bof $flg, $a0, multoverflow
j multloop
multend:
pop $s1
pop $s0
jr $ra
multoverflow:
addc $v1, $v1, 1
j multloop
```

21.3 Hardware

```
module nanoRiskProcessor2(clk,rst,finish,plagioclasio);
     input clk, rst;
     output finish;
     output[7:0] plagioclasio;
     nanoRiskProcessor nRp(clk,rst,finish,plagioclasio);

endmodule

module nanoRiskProcessor(RealClock, Reset, Done,plagioclasio);
     integer i;
     input RealClock, Reset;
     output Done;
     output[7:0] plagioclasio;
     reg[7:0] registers[0:15];
     reg[7:0] data_memory[0:255];
     reg[7:0] inst_memory[0:255];

     reg Clock;

     reg[7:0] next;
     reg[7:0] ghost[1:0];


     reg[3:0] instruction;
     reg[3:0] R0_id,R1_id,R2_id;
     reg[7:0] Extra,Memaddr;

     assign plagioclasio = registers[13];
  assign Done = instruction==0;
     always@(posedge RealClock)
     begin
          ghost[Clock]=inst_memory[next];
          Clock=~Clock;
          next=next+1;



          if(next[0]==0)
          begin
          instruction=ghost[0][7:4];
          R0_id=ghost[0][3:0];
          R1_id=ghost[1][7:4];
          R2_id=ghost[1][3:0];
          Extra=ghost[1];
          Memaddr=registers[R1_id]+R2_id;
          case(instruction)
```

```
4'b0000://SLP
begin
     if(registers[R0_id]==0)
          next=next-2;
end


4'b0001://BRQ
begin
     if(registers[R0_id]==registers[R1_id])
          next=registers[R2_id];
end

4'b0010://BRF
begin
     if(registers[R0_id]&registers[R1_id]>0)
     begin
          registers[R0_id]=registers[R0_id]-
              registers[R1_id];
          next=registers[R2_id];
     end
end

4'b0011://ADD
begin
     if(registers[R1_id]>=8'b10000000 && registers[
         R2_id]>=8'b10000000 && registers[R1_id]+
         registers[R2_id]<8'b10000000)
          registers[1][1]=1;//negative overflow
     if(registers[R1_id]<8'b10000000 && registers[
         R2_id]<8'b10000000 && registers[R1_id]+
         registers[R2_id]>=8'b10000000)
          registers[1][0]=1;//positive overflow

     registers[R0_id]=registers[R1_id]+registers[
         R2_id];
end


4'b0100://SUB
begin
     if(registers[R1_id]<8'b10000000 && registers[
         R2_id]>=8'b10000000 && registers[R1_id]-
         registers[R2_id]>=8'b10000000)
          registers[1][1]=1;//negative overflow

     registers[R0_id]=registers[R1_id]-registers[
         R2_id];
end

4'b0101://AND
begin
     registers[R0_id]=registers[R1_id]&registers[
         R2_id];
end


4'b0110://OR
begin
     registers[R0_id]=registers[R1_id]|registers[
         R2_id];
end
```

```
4'b0111://NOR
begin
     registers[R0_id]=~(registers[R1_id]|registers[
         R2_id]);
end

4'b1000://SLT
begin
     if(registers[R1_id]<registers[R2_id])
          registers[R0_id]=8'b11111111;
     else
          registers[R0_id]=8'b00000000;
end


4'b1001://SR
begin
     registers[R0_id]=registers[R1_id]>>registers[
         R2_id];
end

4'b1010://SL
begin
     registers[R0_id]=registers[R1_id]<<registers[
         R2_id];
end

4'b1011://JR
begin
     if(inst_memory[next][7:4]==4'b0011&&inst_memory
         [next+1]==8'b00100000)//jal
     begin
          registers[inst_memory[next][3:0]]<=next+2;
          if(R0_id==0)
               next<=Extra;
          else
               next<=registers[R0_id];
     end
     else
     begin
          registers[2]<=next;
          if(R0_id==0)
               next<=Extra;
          else
               next<=registers[R0_id];
     end
end


4'b1100://LA
begin
     registers[R0_id]=Extra;
end

4'b1101://LC
begin
     registers[R0_id]=Extra;
end


4'b1110://LW
```

```
                begin
                    registers[R0_id]=data_memory[Memaddr];
                end

                4'b1111://SW
                begin
                    data_memory[Memaddr]=registers[R0_id];
                end

            endcase
            end
        end

        initial //or always@(Reset)
        begin
            next=0;
            ghost[0]=0;
            ghost[1]=0;
            Clock=0;
            instruction=0;
            R0_id=0;
            R1_id=0;
            R2_id=0;
            Extra=0;
            Memaddr=0;
            for(i=0;i<16;i=i+1)
                registers[i]=0;
            for(i=0;i<256;i=i+1)
                data_memory[i]=0;
            for(i=0;i<256;i=i+1)
                inst_memory[i]=0;
            // code down here

data_memory[0]=8'b00000001;
data_memory[1]=8'b00101100;
data_memory[2]=8'b00000000;
data_memory[3]=8'b10010110;
data_memory[4]=8'b00000010;
data_memory[5]=8'b10011010;
data_memory[6]=8'b00000001;
data_memory[7]=8'b01100101;
data_memory[8]=8'b00000000;
data_memory[9]=8'b11011100;
data_memory[10]=8'b00000001;
data_memory[11]=8'b11100000;
data_memory[12]=8'b00000001;
data_memory[13]=8'b00010100;
data_memory[14]=8'b00000010;
data_memory[15]=8'b10011010;
data_memory[16]=8'b00000001;
data_memory[17]=8'b00000010;
data_memory[18]=8'b00000010;
data_memory[19]=8'b00001000;
data_memory[20]=8'b00101000;
data_memory[21]=8'b00011010;
data_memory[22]=8'b00011100;
data_memory[23]=8'b00010000;
```

```
inst_memory[0]=8'b11000111;
inst_memory[1]=8'b00010000;
inst_memory[2]=8'b11001000;
inst_memory[3]=8'b00010100;
inst_memory[4]=8'b11100111;
inst_memory[5]=8'b01110000;
inst_memory[6]=8'b11101000;
inst_memory[7]=8'b10000000;
inst_memory[8]=8'b11010010;
inst_memory[9]=8'b10000000;
inst_memory[10]=8'b10110010;
inst_memory[11]=8'b00000000;
inst_memory[12]=8'b00110100;
inst_memory[13]=8'b00100000;
inst_memory[14]=8'b00111011;
inst_memory[15]=8'b01010000;
inst_memory[16]=8'b00111100;
inst_memory[17]=8'b01100000;
inst_memory[18]=8'b11000111;
inst_memory[19]=8'b00010001;
inst_memory[20]=8'b11001000;
inst_memory[21]=8'b00010101;
inst_memory[22]=8'b11100111;
inst_memory[23]=8'b01110000;
inst_memory[24]=8'b11101000;
inst_memory[25]=8'b10000000;
inst_memory[26]=8'b11010010;
inst_memory[27]=8'b10000000;
inst_memory[28]=8'b10110010;
inst_memory[29]=8'b00000000;
inst_memory[30]=8'b00110100;
inst_memory[31]=8'b00100000;
inst_memory[32]=8'b00111011;
inst_memory[33]=8'b01011011;
inst_memory[34]=8'b00111100;
inst_memory[35]=8'b01101100;
inst_memory[36]=8'b11000111;
inst_memory[37]=8'b00010010;
inst_memory[38]=8'b11001000;
inst_memory[39]=8'b00010110;
inst_memory[40]=8'b11100111;
inst_memory[41]=8'b01110000;
inst_memory[42]=8'b11101000;
inst_memory[43]=8'b10000000;
inst_memory[44]=8'b11010010;
inst_memory[45]=8'b10000000;
inst_memory[46]=8'b10110010;
inst_memory[47]=8'b00000000;
inst_memory[48]=8'b00110100;
inst_memory[49]=8'b00100000;
inst_memory[50]=8'b00111011;
inst_memory[51]=8'b01011011;
inst_memory[52]=8'b00111100;
inst_memory[53]=8'b01101100;
inst_memory[54]=8'b11000111;
inst_memory[55]=8'b00010011;
inst_memory[56]=8'b11001000;
inst_memory[57]=8'b00010111;
```

```
inst_memory[58]=8'b11100111;
inst_memory[59]=8'b01110000;
inst_memory[60]=8'b11101000;
inst_memory[61]=8'b10000000;
inst_memory[62]=8'b11010010;
inst_memory[63]=8'b10000000;
inst_memory[64]=8'b10110010;
inst_memory[65]=8'b00000000;
inst_memory[66]=8'b00110100;
inst_memory[67]=8'b00100000;
inst_memory[68]=8'b00111100;
inst_memory[69]=8'b01101100;
inst_memory[70]=8'b00111011;
inst_memory[71]=8'b01011011;
inst_memory[72]=8'b11010111;
inst_memory[73]=8'b00000010;
inst_memory[74]=8'b11010010;
inst_memory[75]=8'b01010000;
inst_memory[76]=8'b00100001;
inst_memory[77]=8'b01110010;
inst_memory[78]=8'b10110000;
inst_memory[79]=8'b01010100;
inst_memory[80]=8'b11010010;
inst_memory[81]=8'b00000001;
inst_memory[82]=8'b00111100;
inst_memory[83]=8'b11000010;
inst_memory[84]=8'b00110001;
inst_memory[85]=8'b00000000;
inst_memory[86]=8'b11011101;
inst_memory[87]=8'b00000000;
inst_memory[88]=8'b11001110;
inst_memory[89]=8'b00000000;
inst_memory[90]=8'b11010010;
inst_memory[91]=8'b00001000;
inst_memory[92]=8'b10000111;
inst_memory[93]=8'b11010010;
inst_memory[94]=8'b11010010;
inst_memory[95]=8'b01111010;
inst_memory[96]=8'b00010111;
inst_memory[97]=8'b00000010;
inst_memory[98]=8'b11101001;
inst_memory[99]=8'b11100000;
inst_memory[100]=8'b11101000;
inst_memory[101]=8'b11100001;
inst_memory[102]=8'b01001001;
inst_memory[103]=8'b10011100;
inst_memory[104]=8'b01001000;
inst_memory[105]=8'b10001011;
inst_memory[106]=8'b00110111;
inst_memory[107]=8'b10011000;
inst_memory[108]=8'b11010010;
inst_memory[109]=8'b01111010;
inst_memory[110]=8'b00010000;
inst_memory[111]=8'b01110010;
inst_memory[112]=8'b11010010;
inst_memory[113]=8'b00000001;
inst_memory[114]=8'b00111101;
inst_memory[115]=8'b11010010;
inst_memory[116]=8'b11010010;
inst_memory[117]=8'b00000010;
inst_memory[118]=8'b00111110;
inst_memory[119]=8'b11100010;
```

```
inst_memory[120]=8'b10110000;
inst_memory[121]=8'b01011010;
inst_memory[122]=8'b11010010;
inst_memory[123]=8'b00000001;
inst_memory[124]=8'b00111101;
inst_memory[125]=8'b11010010;
inst_memory[126]=8'b00000001;
inst_memory[127]=8'b00000000;
inst_memory[128]=8'b11010010;
inst_memory[129]=8'b00000001;
inst_memory[130]=8'b01000011;
inst_memory[131]=8'b00110010;
inst_memory[132]=8'b11111011;
inst_memory[133]=8'b00110000;
inst_memory[134]=8'b11010010;
inst_memory[135]=8'b00000001;
inst_memory[136]=8'b01000011;
inst_memory[137]=8'b00110010;
inst_memory[138]=8'b11111100;
inst_memory[139]=8'b00110000;
inst_memory[140]=8'b10001011;
inst_memory[141]=8'b01111000;
inst_memory[142]=8'b11010010;
inst_memory[143]=8'b10011000;
inst_memory[144]=8'b00011011;
inst_memory[145]=8'b00000010;
inst_memory[146]=8'b00111011;
inst_memory[147]=8'b01110000;
inst_memory[148]=8'b00111100;
inst_memory[149]=8'b10000000;
inst_memory[150]=8'b10110000;
inst_memory[151]=8'b10011100;
inst_memory[152]=8'b00111011;
inst_memory[153]=8'b10000000;
inst_memory[154]=8'b00111100;
inst_memory[155]=8'b01110000;
inst_memory[156]=8'b11010101;
inst_memory[157]=8'b00000000;
inst_memory[158]=8'b11010110;
inst_memory[159]=8'b00000000;
inst_memory[160]=8'b00110001;
inst_memory[161]=8'b00000000;
inst_memory[162]=8'b11010010;
inst_memory[163]=8'b10110100;
inst_memory[164]=8'b00011011;
inst_memory[165]=8'b00000010;
inst_memory[166]=8'b00110101;
inst_memory[167]=8'b01011100;
inst_memory[168]=8'b11010111;
inst_memory[169]=8'b00000010;
inst_memory[170]=8'b11010010;
inst_memory[171]=8'b11111111;
inst_memory[172]=8'b00111011;
inst_memory[173]=8'b10110010;
inst_memory[174]=8'b11010010;
inst_memory[175]=8'b11000010;
inst_memory[176]=8'b00100001;
inst_memory[177]=8'b01110010;
inst_memory[178]=8'b10110000;
inst_memory[179]=8'b10100000;
inst_memory[180]=8'b11101100;
inst_memory[181]=8'b00110000;
```

```
inst_memory[182]=8'b11010010;
inst_memory[183]=8'b00000001;
inst_memory[184]=8'b00110011;
inst_memory[185]=8'b00110010;
inst_memory[186]=8'b11101011;
inst_memory[187]=8'b00110000;
inst_memory[188]=8'b11010010;
inst_memory[189]=8'b00000001;
inst_memory[190]=8'b00110011;
inst_memory[191]=8'b00110010;
inst_memory[192]=8'b10110100;
inst_memory[193]=8'b00000000;
inst_memory[194]=8'b11010010;
inst_memory[195]=8'b00000001;
inst_memory[196]=8'b00110110;
inst_memory[197]=8'b01100010;
inst_memory[198]=8'b10110000;
inst_memory[199]=8'b10100000;
```

```
    end
endmodule
```

22 Conclusões

Por mais que houvessem limitações no hardware que deve ser proposto foi possível projetar muita coisa e ainda tem espaço para projetar, definir, melhorar muito. A maior limitação é a quantidade de bits, que por mais que facilite na hora da elaboração do hardware vem com muitos problemas, como por exemplo, o maior valor armazenável com 8 bits que é 256, sendo que o problema proposto possui números maiores, a solução então será usar double words(palavras duplas) e tratamento de overflow, este por sua vez não foi efetivo em sua implementação causando vários problemas, não funcionando como o esperado e limitando palavras duplas à números positivos. Foi definido duas flags de overflow para que o programador pudesse trabalhar mais facilmente com numeros unsigned sem que houvesse necessidade de operações especificas para isso. A quantidade menor de registradores também complica um pouco.

Na definição da prática há restrição de não haver entradas nem saídas, porém caso haja tempo e seja aprovado, é possível adicionar entradas e saidas no projeto, já foi definida estrutura para isso através de flags e dos registradores de função(arg e return). É importante lembrar que primeiro de tudo será desenvolvido o projeto como foi proposto, para usar entradas e saídas seria necessário modificar o programa para esperar as leituras e adicionar no hardware essa função. Além disso não foi necessário modificar, as instruções ou estruturas definidas originalmente para colocar conceitos de entrada e saida. O mais interessante desse conceito além de ampliar a interação e aplicação é por exemplo a possibilidade de fazer um outro programa no futuro que funcionaria como um

sistema operacional para gravar programas extras na memoria e alternar entre eles, algo que não seria tão complicado devido a simplicidade do projeto.

Tudo que foi proposto está sujeito a modificações na hora de implementar, mas o conceito original será mantido, por exemplo, o registrador base pointer $bp pode não ser necessário na hora de criar o hardware ou outro registrador pode ser necessário, ou, a mémoria RAM pode ser dividida ou não, ou, definição de flags e interrupts.

Foram achados diversos erros no compilador e na lógica do projeto, as instruções lc e la fazem a mesma coisa, faz falta uma instrução de link e o registrador temporário não deveria ser indexado pois o seu uso no software pode causar vários problemas.

Não foi possível ver o projeto funcionando na versão 1, por causa de diversos erros. Uma falha de sincronismo entre os componentes e erros na detecção de bordas do clock podem estar acontecendo além de erros de projeto e implementação.

Entretanto foi desenvolvida uma versão 2 do mesmo projeto em verilog de forma mais intuitiva, sem os circuitos para minimizar variáveis e corrigir erros de projeto e implementação, essa versão foi bem sucedida e é possível observar seu funcionamento na imagem abaixo. Mesmo com as falhas muito foi aprendido durante o desenvolvimento, implementação e correção do projeto sobre projetos, processadores, organização, testes, programação, etc.

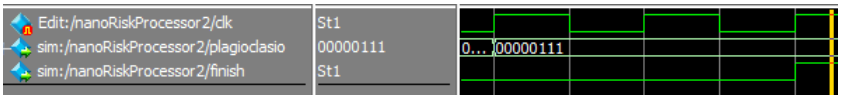


Figura 32: Simulação do nanoRisk Processor 2