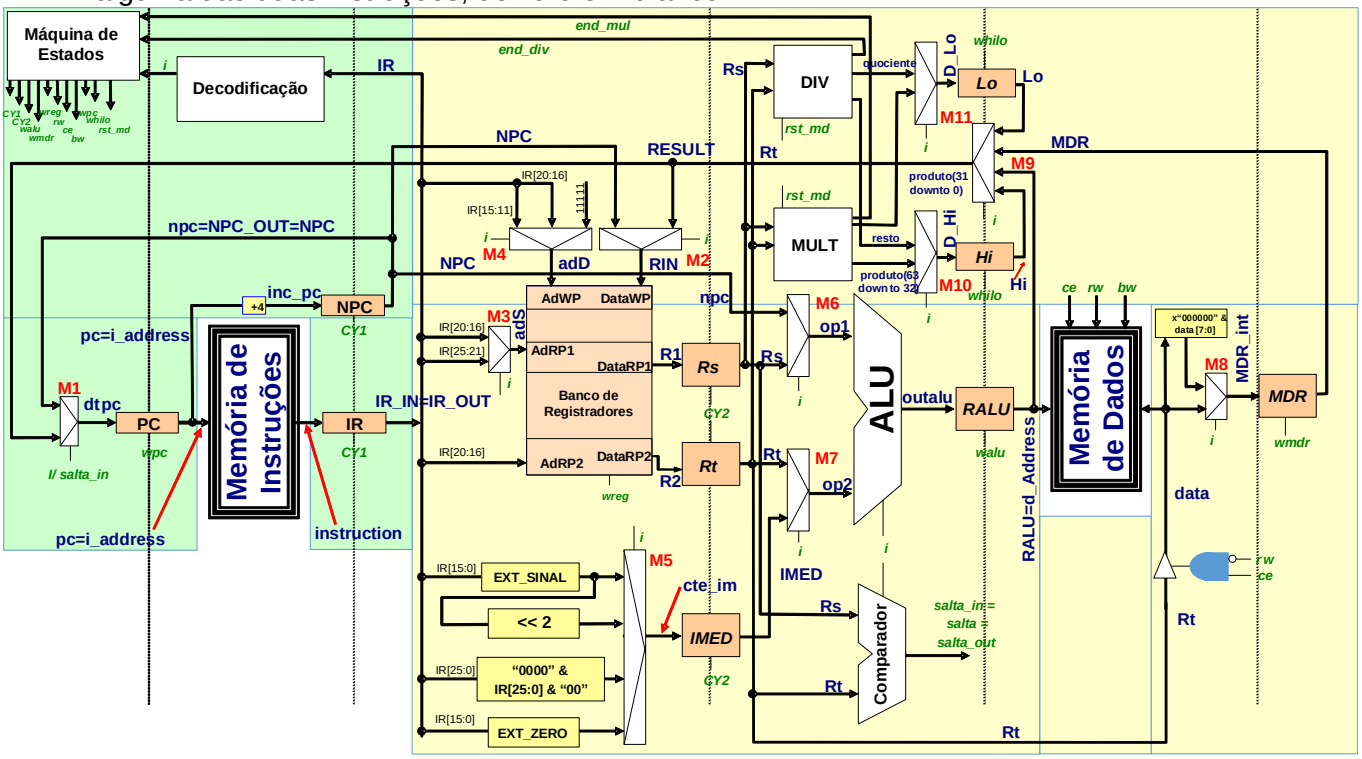
Prova P2 **Disciplina: Organização de Computadores**

Professor: Ney Laert Vilar Calazans

Aluno: 21/novembro/2017

1. [3,5 pontos]. Considere o processador multiciclo abaixo (é o mesmo visto em aula):
   a) [1,5 pontos]. Marcar sobre o desenho todos os caminhos que contém informações úteis à execução da instrução "SW Rt, Offset(Rs)" em algum ciclo de relógio durante a execução (nos multiplexadores, indicar de que entrada a saída do multiplexador vem).
   b) [2 pontos]. Use a tabela abaixo do desenho para explicar sucintamente o que ocorre em cada ciclo de relógio para as duas instruções especificadas na segunda e terceira colunas da tabela. A cada ciclo, dizer que registrador(es) é(são) escrito(s), e com qual conteúdo (semântica do valor), e a operação que a ULA executa, no ciclo relevante. Se algum ciclo não for usado em alguma das duas instruções, deixe-o em branco.



Ciclo de Relógio	Instrução: SW Rt, Offset(Rs)	Instrução: JALR Rs
1º		
2º		
3º		
4º		
5º		

2. [3,0 pontos]. Considere a versão simplificada da ULA (ou ALU) do processador MIPS monociclo visto em aula, cujo código VHDL é mostrado abaixo.

```
1.  architecture alu of alu is
2.  begin
3.      outalu <=
4.          op1 - op2      when  op_alu=SUBU                    else
5.          op1 and op2    when  op_alu=AAND                    else
6.          op1 or  op2    when  op_alu=OOR  or op_alu=ORI      else
7.          op1 xor op2    when  op_alu=XXOR                    else
8.          op1 nor op2    when  op_alu=NNOR                    else
9.          op1 + op2;
10. end alu;
```

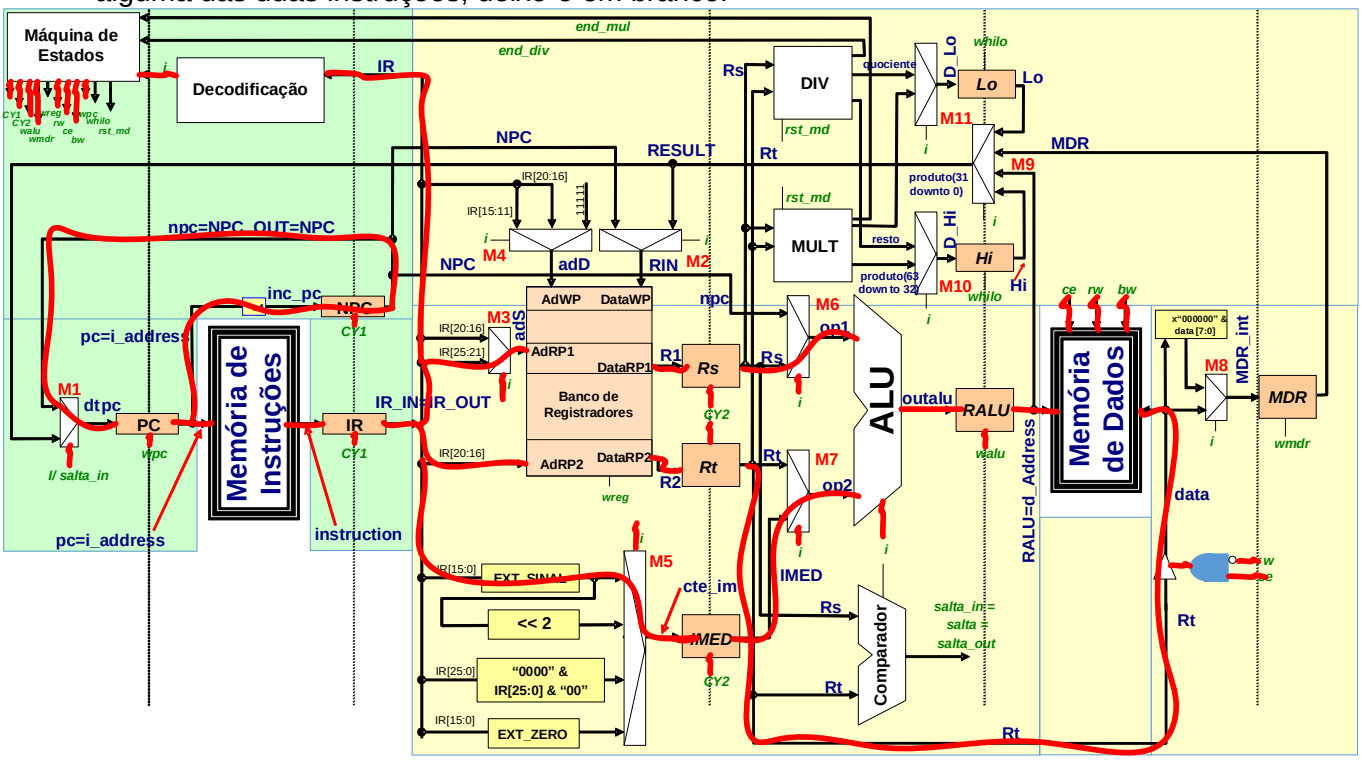
Desenhe um hardware correto que seria gerado por uma ferramenta de síntese ao processar esta linha, assumindo o seguinte: op_alu é conectado ao sinal uins.i, um sinal que codifica com o mínimo número de bits possível os símbolos associados ao nome de cada instrução do processador e o símbolo **invalid_instruction**. A codificação parte do menor valor possível, e segue em ordem crescente para os símbolos ordenados conforme aparece na declaração do tipo inst_type (ou seja, **ADDU, SUBU, AAND, OOR, XXOR, NNOR, LW, SW, ORI, invalid_instruction**). Não é necessário detalhar os operadores gerados pelas expressões **op1 – op2**, **op1 and op2**, etc., nem a implementação do hardware que toma a saída destes operadores e gera o sinal outalu. Estes podem ser apenas "caixas" com entradas, saídas e texto dizendo o que tem dentro das caixas. Contudo, quer-se que a lógica de controle que escolhe o que vai para **outalu** seja detalhada no nível de portas lógicas.

3. [3,5 pontos]. Dada uma frequência de operação de 400MHz para o processador **MIPS monociclo**, assuma que a organização original foi alterada para dar suporte a todas as instruções do programa abaixo, mantendo a característica monociclo. Com estes pressupostos, calcule e diga:
   a) (1,5 pontos). O número de ciclos de relógio que leva p/ executar o programa, com a área de dados fornecida;
   b) (0,5 ponto). O tempo de execução do programa em segundos (1ns=10^{-9} segundos);
   c) (1,0 ponto). O que faz este programa, do ponto de vista semântico. Diga o que ele gera como saída;
   d) (0,5 pontos). Se este programa possui subrotina(s). Se sim, diga onde esta(s) se encontra(m), definindo o intervalo de linhas que ela(s) ocupa(m).

```
1.          .text
2.          .globl main
3.  main:   la      $s0,vt
4.          xor     $s1,$s1,$s1
5.          la      $t2,n
6.          lw      $t2,0($t2)
7.  l:      lw      $t3,0($s0)
8.          andi    $t3,$t3,1
9.          beq     $t3,$zero,ep
10. v:      addiu   $s0,$s0,4
11.         addiu   $t2,$t2,-1
12.         beq     $t2,$zero,fim
13.         j       l
14. ep:     addiu   $s1,$s1,1
15.         j       v
16. fim:    la      $t1,np
17.         sw      $s1,0($t1)
18.         li      $v0,10
19.         syscall
20.         .data
21. np:     .word   0
22. vt:     .word   345 622 80 1040 57 145 83 74 9860 610 245 925
23. n:      .word   12
```

Gabarito

1. [3,5 pontos]. Considere o processador multiciclo abaixo (é o mesmo visto em aula):
   a) [1,5 pontos]. Marcar sobre o desenho todos os caminhos que contém informações úteis à execução da instrução “SW Rt, Offset(Rs)” em algum ciclo de relógio durante a execução (nos multiplexadores, indicar de que entrada a saída do multiplexador vem).
   b) [2 pontos]. Use a tabela abaixo do desenho para explicar sucintamente o que ocorre em cada ciclo de relógio para as duas instruções especificadas na segunda e terceira colunas da tabela. A cada ciclo, dizer que registrador(es) é(são) escrito(s), e com qual conteúdo (semântica do valor), e a operação que a ULA executa, no ciclo relevante. Se algum ciclo não for usado em alguma das duas instruções, deixe-o em branco.



Ciclo de Relógio	Instrução: SW Rt, Offset(Rs)	Instrução: JALR Rs
1º	O conteúdo do registrador PC é usado para buscar o código objeto da instrução **SW Rt,Offset(Rs)** da Memória de Instruções, que é carregado no registrador IR. O valor do PC incrementado de 4 é carregado no registrador NPC. Sinal ativado CY1.	
2º	Instrução é decodificada no BC, que gera sinais de controle p/ sua execução a cada ciclo. Neste, CY2 faz com que Rs seja carregado com o valor do registrador base, IMED seja carregado com o deslocamento (offset) estendido para 32 bits e Rt seja carregado com dado a ser escrito na memória.	Instrução é decodificada no BC, que gera sinais de controle p/ sua execução a cada ciclo. Neste, CY2 faz com que Rs receba o endereço para onde saltar.
3º	A ALU produz o endereço de escrita, somando base e offset, o que é então carregado em RALU. Sinal ativo: walu.	A ALU produz o endereço de salto, simplesmente carregando sua entrada op1 em RALU. Sinal ativo: walu.
4º	Conteúdo de RALU endereça a memória de dados, sinais ce/rw/bw comandam a escrita de 4 bytes a partir deste endereço com os conteúdos de Rt, conectados à entrada da memória pelos *buffers tristate* controlados por rw e ce. Em paralelo, o valor do NPC é copiado para o PC. Sinal ativo: wpc.	Conteúdo de RALU passa para RESULT e daí para a entrada do PC e lá é carregado (saltando). Em paralelo, o banco de registradores é escrito no registrador $ra ($31 ou 11111) com o valor de NPC. Sinais ativos: wreg e wpc.
5º	-	-

Solução:

a) O desenho acima foi anotado com os caminhos usados pela instrução. Note que foram salientados todos os sinais de controle efetivamente relevantes para a instrução. O Item b) detalha o processo que ocorre, ciclo de relógio a ciclo de relógio.

b) Ver conteúdo da tabela acima para as duas instruções. Observar que houve uma falha na especificação deste item, pois a instrução JALR Rótulo deveria de fato ser JALR Rs, pois seu operando é um registrador e não um endereço de instrução (como ocorre nas instruções JAL e J). Isto será considerado durante a correção, de forma a não prejudicar os alunos de nenhuma forma.

2. [3,0 pontos]. Considere a versão simplificada da ULA (ou ALU) do processador MIPS monociclo visto em aula, cujo código VHDL é mostrado abaixo

```
1.  architecture alu of alu is
2.  begin
3.      outalu <=
4.          op1 - op2      when  op_alu=SUBU                    else
5.          op1 and op2    when  op_alu=AAND                    else
6.          op1 or  op2    when  op_alu=OOR  or op_alu=ORI      else
7.          op1 xor op2    when  op_alu=XXOR                    else
8.          op1 nor op2    when  op_alu=NNOR                    else
9.          op1 + op2;
10. end alu;
```

Desenhe um hardware correto que seria gerado por uma ferramenta de síntese ao processar esta linha, assumindo o seguinte: op_alu é conectado ao sinal uins.i, um sinal que codifica com o mínimo número de bits possível os símbolos associados ao nome de cada instrução do processador e o símbolo **`invalid_instruction`**. A codificação parte do menor valor possível, e segue em ordem crescente para os símbolos ordenados conforme aparece na declaração do tipo inst_type (ou seja, **`ADDU, SUBU, AAND, OOR, XXOR, NNOR, LW, SW, ORI, invalid_instruction`**). Não é necessário detalhar os operadores gerados pelas expressões op1 – op2, op1 and op2, etc., nem a implementação do hardware que toma a saída destes operadores e gera o sinal outalu. Estes podem ser apenas "caixas" com entradas, saídas e texto dizendo o que tem dentro das caixas. Contudo, quer-se que a lógica de controle que escolhe o que vai para outalu seja detalhada no nível de portas lógicas.

Solução: A ALU executa a cada instante uma de 6 operações binárias (ou seja, de dois operandos) distintas sobre valores de 32 bits (veja a declaração de **`op1`** e de **`op2`**). Logo sua implementação consiste em um multiplexador que recebe 6 vetores de 32 bits (candidatos a passar para sua saída **`outalu`**) e produz o vetor de 32 bits **`outalu`**. A lógica de controle é o que se quer que seja detalhado no nível de portas lógicas. Para tanto, usando o esquema de codificação proposto, cada instrução será codificada em 4 bits (ADDU→0000, SUBU→0001, ..., **`invalid_instruction`**→1001). Por outro lado, o multiplexador deve ser controlado por um vetor de no mínimo 3 bits (para gerar as 6 possibilidades distintas de seleção de entrada). Ordenando as entradas do multiplexador de acordo com sua posição no texto VHDL, pode-se associar as entradas do mux assim: 0(000)→**`SUBU`**, 1(001)→**`AND`**, 2(010)→**`OR/ORI`**, 3(011)→**`XOR`**, 4(100)→**`NOR`**. 5(101)→demais instruções e **`invalid_instruction`**. A partir destas codificações pode-se gerar o hardware completo. O hardware consiste em três funções Booleanas de 4 variáveis, assim definidas:

Cód. instrução	uins.i(3)	uins.i(2)	uins.i(1)	uins.i(0)	mux(2)	mux(1)	mux(0)	Cód. operação
ADDU	0	0	0	0	1	0	1	+
SUBU	0	0	0	1	0	0	0	-
AND	0	0	1	0	0	0	1	and
OR	0	0	1	1	0	1	0	or
XOR	0	1	0	0	0	1	1	xor
NOR	0	1	0	1	1	0	0	nor
LW	0	1	1	0	1	0	1	+
SW	0	1	1	1	1	0	1	+
ORI	1	0	0	0	0	1	0	or
Invalid_instruction	1	0	0	1	-	-	-	
-	1	0	1	0	-	-	-	
-	1	0	1	1	-	-	-	
-	1	1	0	0	-	-	-	
-	1	1	0	1	-	-	-	
-	1	1	1	0	-	-	-	
-	1	1	1	1	-	-	-	

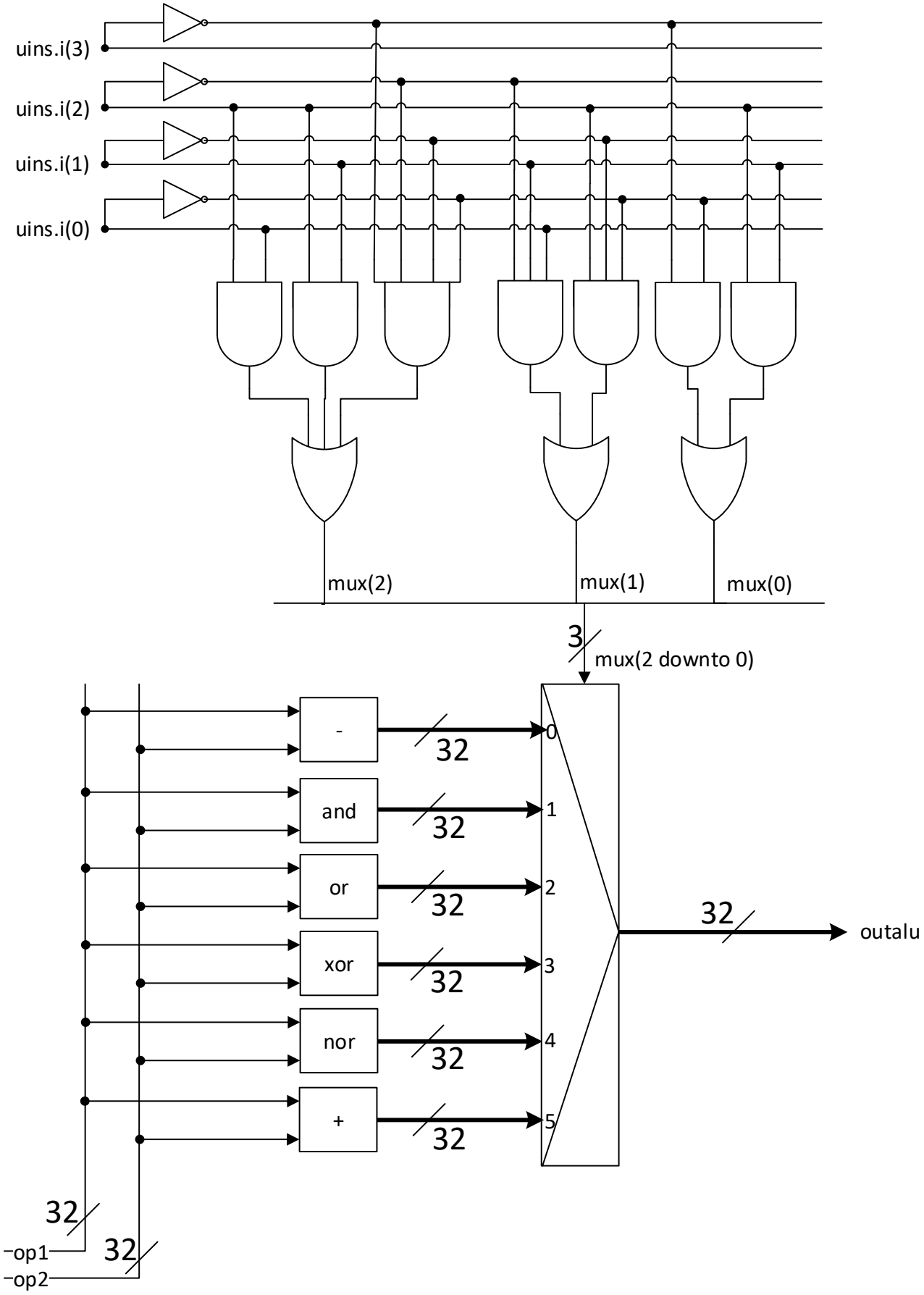
De posse desta tabela verdade, usa-se qualquer método adequado de implementação das funções. Veja por exemplo a aplicação web disponível em http://www.32x8.com/. O resultado é (. representa função **`and`**, + representa função **`or`** e ~ representa função **`not`**):

mux(2) = uins.i(2).uins.i(0) + uins.i(2).uins.i(1) + (~uins.i(3)).(~uins.i(2)).(~uins.i(1)).(~uins.i(0))

mux(1) = (~uins.i(2)).uins.i(1).uins.i(0) + uins.i(2).(~uins.i(1)).(~uins.i(0))

mux(0) = (~uins.i(3)).(~uins.i(0)) + uins.i(2).uins.i(1)

O desenho abaixo ilustra a estrutura completa da ALU



3. [3,5 pontos]. Dada uma frequência de operação de 400MHz para o processador **MIPS monociclo**, assuma que a organização original foi alterada para dar suporte a todas as instruções do programa abaixo, mantendo a característica monociclo. Com estes pressupostos, calcule e diga:
   a) (1,5 pontos). O número de ciclos de relógio que leva p/ executar o programa, com a área de dados fornecida;
   b) (0,5 ponto). O tempo de execução do programa em segundos (1ns=10^{-9} segundos);
   c) (1,0 ponto). O que faz este programa, do ponto de vista semântico. Diga o que ele gera como saída;
   d) (0,5 pontos). Se este programa possui subrotina(s). Se sim, diga onde esta(s) se encontra(m), definindo o intervalo de linhas que ela(s) ocupa(m).

```
1.        .text                #
2.        .globl main          # Ciclos
3.  main: la     $s0,vt        # (2) $s0 recebe ponteiro para início do vetor vt
4.        xor    $s1,$s1,$s1   # (1) zera $sa, o contador de pares
5.        la     $t2,n         # (2) toma endereço do número de elementos de vt
6.        lw     $t2,0($t2)    # (1) $t2 fica com tamanho de vt inicialmente
7.  l:    lw     $t3,0($s0)    # (1) $t3 recebe elemento do vetor
8.        andi   $t3,$t3,1     # (1) $t3 ← 1 se elemento do vetor é ímpar, 0 se par
9.        beq    $t3,$zero,ep  # (1) salta para ep se achou par, senão segue
10. v:    addiu  $s0,$s0,4     # (1) incrementa ponteiro para próximo elemento
11.       addiu  $t2,$t2,-1    # (1) decrementa contador de elementos a processar
12.       beq    $t2,$zero,fim# (1) salta para o fim se acabou
13.       j      l             # (1) senão, volta para processar novo elemento
14. ep:   addiu  $s1,$s1,1     # (1) chega aqui qdo elemento é par, incrementa contador
15.       j      v             # (1) depois volta para atualizar ponteiro, contador, teste
16. fim:  la     $t1,np        # (2) chega aqui ao fim do processamento. Pega end var np
17.       sw     $s1,0($t1)    # (1) escreve valor do contador de pares em np
18.       li     $v0,10        # (1) e sai fora do programa
19.       syscall              # (1) com syscall 10
20.       .data
21. np:   .word  0
22. vt:   .word  345 622 80 1040 57 145 83 74 9860 610 245 925
23. n:    .word  12
```

Solução:

a) O programa possui um ciclo apenas, que executa uma vez para cada elemento do vetor vt, ou seja, 12 vezes. O laço ocupa as linhas 7-13, mas executa opcionalmente as linhas 14-15, quando acha um elemento par de vt. As instruções dos trechos das linhas 3-6 e 16-19 são executadas exatamente uma vez, gastando 6+5=11 ciclos. vt possui 6 elementos pares e 6 elementos ímpares. Assim, a execução do laço toma 6*9=54 ciclos (para pares) + 6*7-1=41 ciclos (para ímpares). O -1 ao final da expressão para ímpares ocorre porque o êxito no teste do beq da linha 12 evita executar a linha 13 apenas. Assim, o número de ciclos total para executar o programa é 11+54+41=**106 ciclos**.

b) Como f=400MHz, o período do relógio é T=(1/(400*10^6))s ou 2,5x10^{-9}s. Para executar o programa leva-se então 106*2,5x10^{-9}s=**2,65x10^{-7}s**.

c) O programa computa quantos elementos pares o vetor vt possui e escreve este número na variável np em memória.

d) O programa não possui subrotinas (não usa a instrução jal seguida de jr $ra).