Universidade Federal de São Carlos

Centro de Ciências Exatas e de Tecnologia

Departamento de Computação

Microcontroladores e Microprocessadores

Trabalho 6

Microarquitetura

Professor: Dr. Ricardo Menotti

Alexandre Strabello, 770076, Engenharia Física

Gabriel Lemes Molizane Almeida, 770097, Engenharia Física

Sarah Brennda Bispo Almeida, 770161, Engenharia Física

São Carlos

2023

1 Introdução

O objetivo dessa pratica é estudar a microarquitetura de processadores do tipo RISC (ARM e RISC-V). Para isso foram utilizadas ferramentas de acesso remoto ao Laboratório Virtual para programar uma placa Zybo z720 com um processador RISC-V simplificado e foram adicionados o módulo VGA e novos comandos para realização de programas simples com processamento de vídeo.

2 Descrição da Execução do Experimento

De modo geral, o relatório utilizou como base o repositório do professor da disciplina de um processador RISC-V reduzido [2] multi ciclo, bem como o laboratório virtual para testes de funcionamento [1].

2.1 Acrescentar o módulo VGA ao processador LightRISC-V

Para adicionar o módulo de vídeo ao processador, utilizou-se o módulo gráfico já produzido anteriormente, considerando uma resolução de 20x15 pixels, em um monitor 640x480. Algumas modificações na memória foram realizadas para permitir que esta apresentasse indicadores para acesso de vídeo.

Decidiu-se utilizar o laboratório virtual da disciplina, dessa forma, a memória também foi adicionada diretamente no arquivo verilog. O código a ser utilizado para testar o funcionamento foi a sequência de Fibonacci, apresentado a seguir:

```
.text    # 0x00000000
#   la s0, b # auipc x8, 2 + addi x8, x8, 0x0104 # riscV
   simulator
    addi x8, x0, 0x0104 # verilog simulation
    lw t0, -4(s0)
    lw t1, (s0)
loop:
    add t2, t1, t0
    mv t0, t1
    mv t1, t2
    addi s0, s0, 4
    sw t2, (s0)
    j loop

.data    # 0x00000000
a:  .word 0
b:  .word 1
```

O código foi armazenado em um arquivo de textos e foi gerada uma memória com 512 posições de 32 bits. As modificações no código gerado em verilog são dadas a seguir:

```
module top(
  input  sysclk,
  output [3:0] VGA_R, VGA_G, VGA_B,
  output VGA_HS_O, VGA_VS_O);

  wire pixel_clk,reset,memwrite;
  wire [31:0] writedata, adr, readdata, vdata,vaddr;

  power_on_reset por(sysclk, reset);
  clk_wiz_1 clockdiv(pixel_clk, sysclk);

  // microprocessor (control & datapath)
  riscvmulti riscvmulti(sysclk, reset, adr, writedata, memwrite,
      readdata);

  // memory
  mem mem(sysclk, memwrite, adr, writedata, vaddr, readdata,
     vdata);

  vga gpu(pixel_clk, reset, vdata, vaddr,VGA_R, VGA_G, VGA_B,
     VGA_HS_O, VGA_VS_O);

endmodule


module mem  //altera  o original: "memfile.hex" para "memfile.
   bin"
          (input  logic        sysclk, we,
           input  logic [31:0] a, wd, va,
           output logic [31:0] rd, vd);

  logic  [31:0] RAM[511:0];

  // initialize memory with instructions
  initial
     begin
      RAM[0] = 32'h10400413;
      RAM[1] = 32'hffc42283;
      //...
```

```
      RAM[510] = 32'h00000000;
      RAM[511] = 32'h00000000;
    end

  assign rd = RAM[a[31:2]]; // word aligned
  assign vd = RAM[va];

  always_ff @(posedge sysclk)
    if (we)
      RAM[a[31:2]] <= wd;
endmodule

module vga( // 20x15
  input sysclk, reset,
  input  [31:0] vdata,
  output [ 6:0] vaddr,
  output [3:0] VGA_R, VGA_G, VGA_B,
  output VGA_HS_O, VGA_VS_O);

  reg [9:0] CounterX, CounterY;
  reg inDisplayArea;
  reg vga_HS, vga_VS;

  wire CounterXmaxed = (CounterX == 800); // 16 + 48 + 96 + 640
  wire CounterYmaxed = (CounterY == 525); // 10 +  2 + 33 + 480
  wire [4:0] col;
  wire [3:0] row;
  wire [7:0] vbyte;

  always @(posedge sysclk or posedge reset)
    if (reset)
      CounterX <= 0;
    else
      if (CounterXmaxed)
        CounterX <= 0;
      else
        CounterX <= CounterX + 1;

  always @(posedge sysclk or posedge reset)
    if (reset)
      CounterY <= 0;
```

```
    else
      if (CounterXmaxed)
        if(CounterYmaxed)
          CounterY <= 0;
        else
          CounterY <= CounterY + 1;

  assign row = (CounterY>>5); // 32 pixels x
  assign col = (CounterX>>5); // 32 pixels (x4 bytes)
  assign vaddr = 65 + (col>>2) + (row<<2) + row; // addr =
     offset + col / 4 + row * 5
  assign vbyte = col[1] ? (col[0] ? vdata[7:0] : vdata[15:8]) :
     (col[0] ? vdata[23:16] : vdata[31:24]); // byte select

  always @(posedge sysclk)
  begin
    vga_HS <= (CounterX > (640 + 16) && (CounterX < (640 + 16 +
       96)));   // active for 96 clocks
    vga_VS <= (CounterY > (480 + 10) && (CounterY < (480 + 10 +
        2)));   // active for  2 clocks
    inDisplayArea <= (CounterX < 640) && (CounterY < 480);
  end

  assign VGA_HS_O = ~vga_HS;
  assign VGA_VS_O = ~vga_VS;

  assign VGA_R = inDisplayArea ? {vbyte[5:4], 2'b00} : 4'b0000;
  assign VGA_G = inDisplayArea ? {vbyte[3:2], 2'b00} : 4'b0000;
  assign VGA_B = inDisplayArea ? {vbyte[1:0], 2'b00} : 4'b0000;
endmodule

module power_on_reset(
  input sysclk,
  output reset);

  reg q0 = 1'b0;
  reg q1 = 1'b0;
  reg q2 = 1'b0;

  always@(posedge sysclk)
  begin
```

```
        q0 <= 1'b1;
        q1 <= q0;
        q2 <= q1;
    end

    assign reset = !(q0 & q1 & q2);
endmodule
```

O código não sofreu alteração significativa frente ao disponibilizado inicialmente, apenas no módulo de vídeo foi adicionado um offset para o endereçamento, garantindo que o endereço lido fosse parte da memória reservada para esse fim.

2.2 Implementar uma animação com funções trigonométricas usando software e/ou hardware

A ideia para esse exercício consistia em complementar o primeiro exercício obrigatório, implementando as novas funções ao processador e realizando uma animação de um ponto descrevendo uma trajetória senoidal. Para esse exercício, desenvolveu-se o código em Risc-V apresentado abaixo:

```
.data
image:
.space 300
sen_data:
.byte #Dados dos senos

.text
#s0 - Numero de linhas
#s1 - Numero de colunas
#s2 - Tempo
#s3 - Celula atual
#S4 - Intensidade maxima
#s5 - Um
#s6 - Valor para delay
li s1,20
li s2,0 #Tempo inicial eh zero
li s3,0 #Celula inicial eh zero
li s4,255
li s5,1
li s6,10000 #numero de interacoes para delay

animation:
```

```
    la a0,sen_data
    mv a1,s2
    jal ra,sen #Determina qual a posicao da celula a ser
       substituida
    mv a1,s1
    mv a2,a0
    mv a0,s2
    jal ra, position #Acha qual a posicao da celula a ser
       selecionada
    mv a1,s3
    mv a2,s4
    jal ra, replace #Substitui a celula na imagem
    mv a0,s6
    jal ra, delay #Atraso para visualizacao no monitor
    addi s2,s2,1
    bne s2,s0, animation
    mv s2,zero
    b animation

mult:
#Entrada: a0 - valor a ser multiplicado; a1 - segundo numero a
   ser multiplicado;
#Saida: a0 - valor multiplicado
    mv t0,a0
    loop_mult:
    add a0,a0,t0
    sub a1,a1,s5
    bne a1,s5,loop_mult
    ret

sen:
#Entrada: a0 - Banco de dados da funcao seno; a1 - Em que
   instante de tempo estah a animacao
#Saidas: a0 - valor do seno calculado naquele tempo
    add a0,a0,a1
    lb a0,(a0)
    li a1,7
    mv a2,ra
    jal ra,mult
    addi a0,a0,7
    srli a0,a0,8
```

```
    mv ra,a2
    ret

position:
#Entrada: a0 - instante de tempo; a1 - numero de linhas; a2 -
   valor do seno;
#Saida: a0 - posicao do pixel a ser substituido
    mv a3,ra
    jal ra,mult
    add a0,a0,a2
    mv ra,a3
    ret

replace:
#Entrada: a0 - nova posicao; a1 - posicao atual; a2 -
   intensidade
    la t0,image
    mv s3,a0
    add a0,a0,t0
    add a1,a1,t0
    sb zero, (a1)
    sb a2, (a0)
    ret

delay:
#Entrada> a0 - Delay
    delay_loop:
    sub a0,a0,s5
    bne a0,zero,delay_loop
    ret
```

O código funciona basicamente considerando cada coluna da imagem como um tempo diferente. Com base nos dados da função seno para determinado ponto, seria gerado um incremento na posição da imagem, além da adição do offset com base no tempo selecionado. Em seguida, o ponto seria substituído por uma cor branca, enquanto a posição anterior retornaria para a cor preta. Dessa forma, o ponto se moveria conforme uma onda senoidal. Os valores da função seno seriam carregados como números de 8 bits, estes sofreriam uma alteração para se adequar entre valores de 0 até 14, sinalizando qual a linha a ser preenchida com a cor branca.

Uma versão alternativa do programa foi elaborada conforme as instruções iniciais apresentadas no processador. O código é apresentado abaixo:

```
.data
image:
.space 1200
sen_data:
.word 0x7,0x9,0xb,0xd,0xe,0xe,0xd,0xb,0x9,0x7,0x5,0x3,0x1,0x0,0
   x0,0x1,0x3,0x5,0x7,0x9

.text
#s0 - Numero de linhas
#s1 - Numero de colunas
#s2 - Tempo
#s3 - Celula atual
#S4 - Intensidade maxima
#s5 - Um
#s6 - Valor para delay
addi s0,zero,15
addi s1,zero,20
addi s2,zero,0 #Tempo inicial eh zero
addi s3,zero,0 #Celula inicial eh zero
addi s4,zero,0xff
addi s5,zero,1

animation:
    la a0,sen_data
    #addi x10, x0, 0x0104 #Verilog
    lw a0,(a0)
    add a1,zero,s2
    #-------------
    #Acha qual a posicao da celula a ser selecionada
    add t0,zero,s2
    addi a1,zero,0
    addi a2,zero,0
    loopmult:
    beq t0,a2,exitmult
    add a1,a1,s0
    sub t0,t0,s5
    j loopmult
    exitmult:
    add a0,a0,a1
    #-------------
    add a1,zero,s3
```

```
    add a2,zero,s4
    #Substitui a celula na imagem
    la t0,image
    #addi t0,zero,0x0350 #Verilog
    add a0,a0,a0
    add a0,a0,a0
    add t1,a1,t0
    add t0,a0,t0
    addi t2,zero,0
    sw t2, (t1)
    sw a2, (t0)
    #-------------
    add s3,zero,a0
    addi t0,zero,5
    addi a2,zero,0
    #Atraso para visualizacao no monitor
    delayloop:
    beq t0,a2,exitloop
    sub t0,t0,s5
    j delayloop
    exitloop:
    addi s2,s2,1
    beq s2,s1, reset_time
    j animation
    reset_time:
    addi s2,zero,0
    j animation
```

As duas linhas com a pseudoinstrução *la* foram utilizadas apenas para simulação no RARS, devendo ser substituídas pelas imediatamente posteriores na simulação em verilog ou na direta implementação do mesmo no laboratório virtual. Os dados da função seno foram inseridos já modificados, retirando a necessidade do tratamento com as funções de deslocamento. As instruções de JAL também foram retiradas, já que a instrução de retorno não estava disponível.

Através de um *assembler*, o programa foi implementado no processador do item 1, simulado no ambiente virtual EDA Playground, e posteriormente compilado no laboratório remoto. Houve necessidade de alterar o offset no endereçamento de vídeo, que assumi o valor de 211. Dessa forma, a memória possuía 64 linhas reservadas para o código, a partir da linha 65 se iniciava a memória de dados, e na linha 211 até a 511 estava o espaço reservado para o vídeo.

3 Avaliação dos Resultados do Experimento

3.1 Acrescentar o módulo VGA ao processador LightRISC-V

O código produzido foi implementado no laboratório virtual da disciplina, cujos resultados são apresentados na Figura 1.

Figura 1: Padrão de cores obtido através da sequência de Fibonacci

Os resultados da sequência de Fibonacci foram utilizados como sinal de vídeo, tal que o número 34 por exemplo é escrito como 100010 em binário, e portanto representa 10 no canal vermelho, 00 no verde, e 10 no verde. Para comparação, foi feita uma simulação de como a imagem deveria ficar utilizando essa lógica, e o resultado está apresentado na Figura 2.

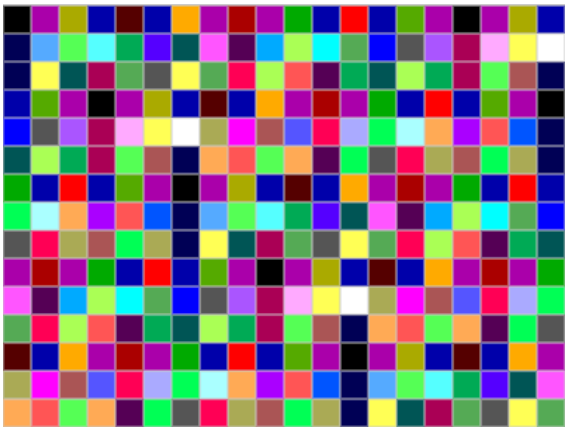

Figura 2: Padrão teórico obtido pela sequência de Fibonacci

Nota-se que o resultado obtido diverge completamente daquele esperado teoricamente para o padrão da sequência de Fibonacci.

3.2 Implementar uma animação com funções trigonométricas

O código desenvolvido com as equações disponíveis no processador foi testado no simulador RARS, onde não foram identificados erros ou problemas no código. O resultado

obtido na implementação no laboratório virtual é dado na Figura 3.

Figura 3: Resultados obtidos para animação trigonométrica durante simulação.

Observa-se que os resultados não chegam nem próximo do que deveria ser executado. A imagem não ao menos é carregada por completa, mantendo uma parte com cores desconhecidas.

4 Análise Crítica e Discussão

4.1 Acrescentar o módulo VGA ao processador LightRISC-V

A implementação do módulo VGA ao processador LightRISC-V foi feita com sucesso. Foi possível testá-lo no laboratório virtual e o mesmo apresentou resultados. Porém, ao simular o programa da sequência de Fibonacci, o resultado obtido foi distinto do esperado, e o motivo para isso segue desconhecido. Verificou-se que o endereçamento dos dados estava na posição correta. O código em assembly foi testado no simulador e garantiu-se que apenas as funções já implementadas fossem utilizadas para desenvolver o programa.

4.2 Implementar uma animação com funções trigonométricas usando software e/ou hardware

Houveram grandes dificuldades na execução desse item. Apesar do programa em *assembly* funcionar como esperado no simulador RARS, o mesmo não funcionou ao utilizar o processador descrito em Verilog. Como discutido anteriormente, não se observou uma animação da função seno no monitor do laboratório virtual, mas sim uma imagem estática de difícil compreensão.

Para compreender o problema ocorrido, escolheu-se utilizar o EDA Playground para realizar mais testes. Os resultados obtidos estão apresentados nas figuras 4 e 5.

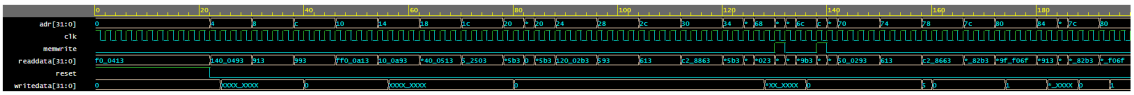

Figura 4: Diagrama de tempo da simulação

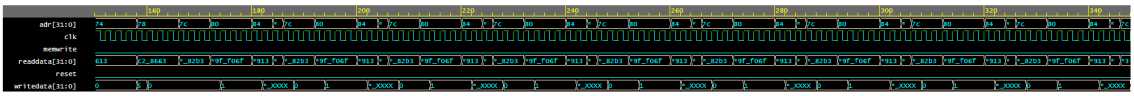

Figura 5: Diagrama de tempo da simulação

Na Figura 4, nota-se que após o endereço ser 0x34 ocorre um salto para 0x68. Porém, o endereço 0x34 é apenas uma instrução de *add*. Além disso, 0x30 corresponde a uma instrução *beq*, cujo salto deveria ser realizado nessa situação, o que também não foi observado, dada a execução de 0x34.

Na Figura 5, ainda é possível perceber outro problema, e talvez o principal: o loop que acontece nas instruções dos endereços 0x78 e 0x84. Isso mostra que a instrução de *beq* no endereço 0x78 nunca era executada, tal como no parágrafo anterior. Mesmo retirando essa parte, o erro persistiu, porém agora ocorria no outro loop utilizado para implementação no atraso de visualização. Como última medida, o programa teve todos os loops removidos, porém o erro persistia, dessa vez na instrução *beq*, utilizada para verificar qual a coluna selecionada.

Dessa forma, verificou-se que a instrução o *beq*, supostamente implementada no processador, não estava funcionando corretamente. Tentou-se analisar a instrução do código verilog, mas não foi possível identificar problemas na escrita da instrução. Ademais, se fazia necessário ao menos uma instrução de salto condicional para a formulação do programa, de modo que a ausência dessa inviabilizava a implementação da animação com o limite de linhas para código.

5 Referências

[1] *Lab. Remoto de Embarcados - DC/UFSCar*. URL: `https://vlab.dc.ufscar.br/`.

[2] Ricardo Menotti. *LightRISC-V*. URL: `https://www.edaplayground.com/x/cTAA`.