

Aluno: \_\_\_\_\_

Matrícula: \_\_\_\_\_ Data: \_\_\_\_\_

### Sprint 1 – Primeiro projeto no Quartus II (Rev Verilog/System Verilog)

**Descrição geral do problema:** Criar o primeiro projeto no Quartus II e implementar um MUX 2x1, de 8bits, em Verilog/SystemVerilog.

#### Requisitos mínimos:

1. Setup do ambiente de trabalho individual
  - a) Crie o diretório “**D ou E:\LASD\20XX.X\SEUNOME\_MATRICULA**”. Cada aluno deverá usar o mesmo computador em todas as aulas do LASD. Sempre salvar seus arquivos nessa pasta;
  - b) Abra o Quartus II 13.0;
  - c) Menu “**File -> New Project Wizard**”;
  - d) Crie o Projeto, com nome “**Mod\_Teste**”, no seu diretório;
  - e) Ignore o pedido de inclusão de arquivos já existentes;
  - f) Selecione a família **CycloneII**, FPGA **EP2C35F672C6** e finish;
  - g) Copie os seguintes arquivos, do google classroom, para a pasta local do seu projeto: **Mod\_Teste.v**, **LCD\_TEST2.v**, **LCD\_CONTROLLER.v** e **DE2\_PIN\_ASSIGNMENT.CSV**
  - h) Use o menu “**Assignments > import assignment**” para incluir o arquivo **DE2\_PIN\_ASSIGNMENT.CSV**
  - i) Adicione, em seu projeto, os arquivos **.v** do passo g). Na janela “**Project Navigator > Files > Botão direito > Add Remove files in project**”. Selecione os arquivos, Add All e OK.



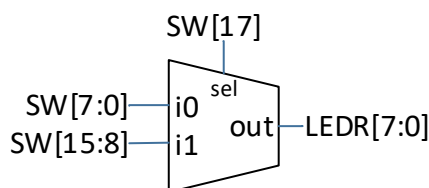
**Obs: Nas aulas subsequentes devemos abrir este projeto usando “File > Open Project” e NÃO “File > Open”.**
2. Dentro do módulo **Mod\_Teste**, faça uma atribuição contínua entre o LEDG[0] e o botão KEY[1]. Compile  , carregue na placa  e teste seu primeiro código!
3. Implemente um MUX 2x1 de 8bits:
  - a) O circuito deve possuir dois canais de entrada de 8bits, **i0** e **i1** respectivamente conectadas às chaves **SW[7:0]** e **SW[15:8]**, uma entrada de seleção, **sel**, conectada à chave **SW[17]** e uma saída, **out**, de 8bits para o resultado, conectada aos **LEDR[7:0]**. A lógica de operação do MUX é resumida na tabela 1 e o diagrama de blocos ilustrado na Figura 1.

Tabela 1

Entrada de seleção do canal ( <b>sel</b> - <b>SW[17]</b> )	Resultado da operação ( <b>out</b> - <b>LEDR[7:0]</b> )
0	<b>i0</b> - <b>SW[7:0]</b>
1	<b>i1</b> - <b>SW[15:8]</b>

Figura 1



b) **Fica a seu critério:**

- i. modelar o circuito em nível de portas lógicas ou subir o nível de abstração?
- ii. implementar com atribuição contínua ou procedural?
- iii. Criar um módulo ou implementar diretamente no Mod\_Testes?
- iv. Teste diferentes formas!

c) Deve ser um circuito combinacional.

d) **Roteiro de testes:**

- i. **i0=8'h0F, i1=8'hF0, sel=1'b0** saída esperada: **out = 8'h0F**
- ii. **i0=8'h0F, i1=8'hF0, sel=1'b1** saída esperada: **out = 8'hF0**
- iii. **i0=8'h0F, i1=8'hFA, sel=1'b1** saída esperada: **out = 8'hFA**
- iv. **i0=8'hBF, i1=8'hFA, sel=1'b1** saída esperada: **out = 8'hFA**
- v. Caso o seu circuito passe por todos os testes, chame o professor para receber sua nota.

e) **Após o professor conferir seus testes, compacte o projeto em um .zip e submeta-o no Google Classroom da disciplina**

f) **Dica: Aproveite o elevado nível de abstração que Verilog proporciona e escreva um código simples e enxuto. Daria para resolver essa sprint com quantas linhas de código?**

**Desafio (Valendo +0,1 na média geral)**

1. Crie um módulo de MUX 2x1 cuja largura das palavras possa ser escolhida por um parâmetro, no momento da instanciação.
2. Defina cenários de testes para seu novo módulo.

**Módulo topo – Mod\_Testes**

```
`default_nettype none //Comando para desabilitar declaração automática de wires
module Mod_Testes (
//Clocks
input          CLOCK_27, CLOCK_50,
//Chaves e Botoes
input  [3:0]    KEY,
input  [17:0]   SW,
//Displays de 7 seg e LEDs
output [0:6]    HEX0, HEX1, HEX2, HEX3, HEX4, HEX5, HEX6, HEX7,
output [8:0]    LEDG,
output [17:0]   LEDR,
//Serial
output          UART_TXD,
input           UART_RXD,
inout  [7:0]    LCD_DATA,
output          LCD_ON, LCD_BLON, LCD_RW, LCD_EN, LCD_RS,
//GPIO
inout  [35:0]   GPIO_0, GPIO_1
);
assign GPIO_1      = 36'hzzzzzzzz;
assign GPIO_0      = 36'hzzzzzzzz;
assign LCD_ON      = 1'b1;
assign LCD_BLON    = 1'b1;
wire  [7:0]        w_d0x0, w_d0x1, w_d0x2, w_d0x3, w_d0x4, w_d0x5,
        w_d1x0, w_d1x1, w_d1x2, w_d1x3, w_d1x4, w_d1x5;
LCD_TEST MyLCD (
.iCLK      ( CLOCK_50 ),
.iRST_N    ( KEY[0] ),
.d0x0(w_d0x0),.d0x1(w_d0x1),.d0x2(w_d0x2),.d0x3(w_d0x3),.d0x4(w_d0x4),.d0x5(w_d0x5),
.d1x0(w_d1x0),.d1x1(w_d1x1),.d1x2(w_d1x2),.d1x3(w_d1x3),.d1x4(w_d1x4),.d1x5(w_d1x5),
.LCD_DATA( LCD_DATA ),
.LCD_RW    ( LCD_RW ),
.LCD_EN    ( LCD_EN ),
.LCD_RS    ( LCD_RS )
);
//----- modifique a partir daqui -----
endmodule
```

Aluno: \_\_\_\_\_  
Matrícula: \_\_\_\_\_ Data: \_\_\_\_\_

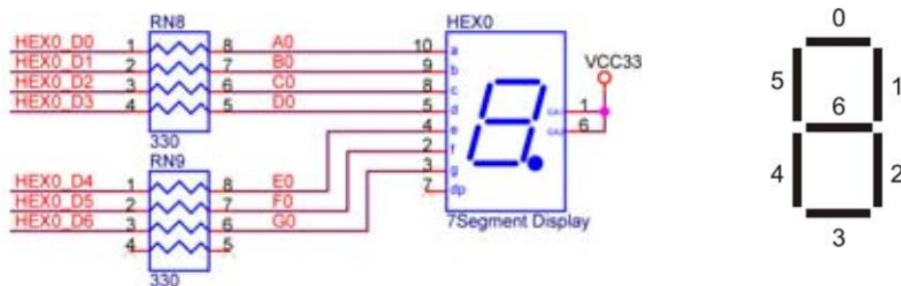
### Sprint 2 – Revisão de Verilog - Blocos construtivos

**Descrição geral do problema:** Seguindo a revisão de Verilog/SystemVerilog, implemente um circuito que conte o tempo entre 0 e 9s e mostre em um display de 7 segmentos.

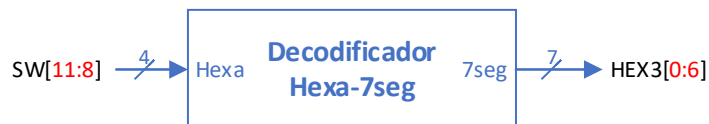
#### Requisitos mínimos:

Abra o projeto da Sprint1 e edite-o para incluir as funcionalidades dessa sprint. **Obs: “File > Open Project” e NÃO “File > Open”.**

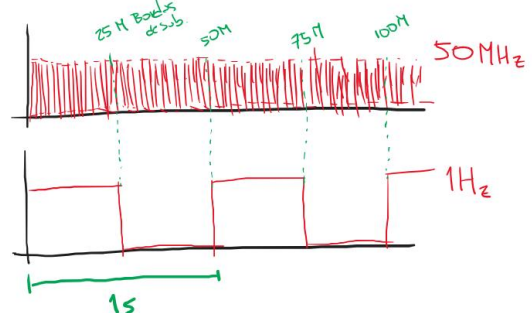
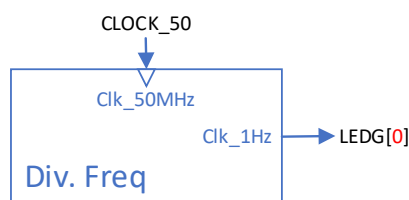
1. Faça um **assign** entre os leds do display de 7 segmentos **HEX0[0:6]** e as chaves **SW[6:0]**. Varie as chaves e observe qual segmento está associado a cada **wire** de **HEX0**.



2. Projete um decodificador de hexadecimal para 7 segmentos.
  - a) Crie o decodificador em uma estrutura de módulo. Para facilitar o reuso, salve-o em um arquivo **.v/.sv** separado.
  - b) O decodificador deve possuir uma entrada de 4bits, para entrar um número em hexa e uma saída de 7bits para conectar um display de 7 segmentos e poder visualizar o dígito.
  - c) Para testar seu circuito, instancie um decodificador, no Mod\_Teste, com o seguinte mapeamento de entradas e saídas:

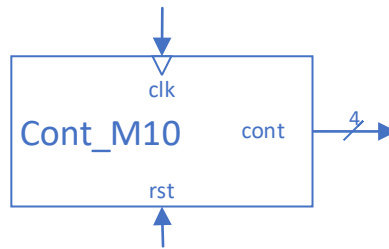


- d) Dica: pesquise sobre a estrutura **case**. Implemente a lógica de funcionamento em alto nível.
3. Implemente um módulo divisor de frequência para gerar um clock de 1Hz a partir do clock de 50MHz disponível na placa DE2 (CLOCK\_50).
    - a) Esse circuito é basicamente um contador de pulsos do clock de entrada, que inverte sua saída cada vez que uma condição é atingida.
    - b) O divisor deve possuir uma entrada de 1bit de clock (rápido, 50MHz) e uma saída de 1bit de clock (lento, 1Hz).
    - c) Para testar seu circuito, instancie um decodificador, no Mod\_Teste, com o seguinte mapeamento de entradas e saídas:

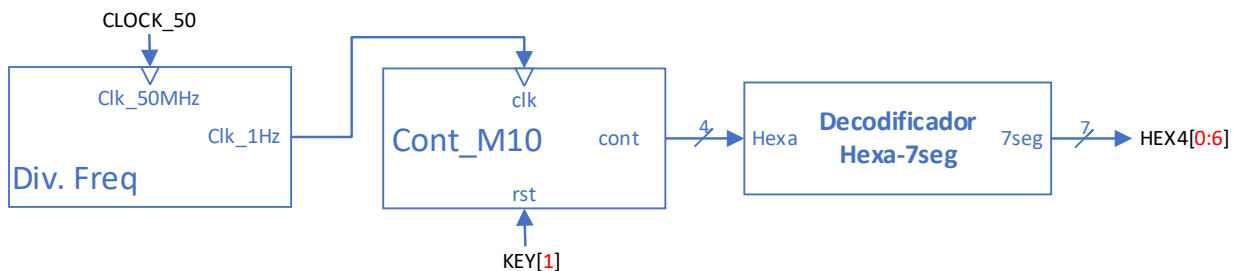


4. Implemente um contador módulo 10 para varrer, ciclicamente, os números de 0 a 9.

- a) Esse circuito deve ter uma entrada de clock, uma entrada de reset e uma saída de contagem de 4 bits. A cada borda de subida do clock, a saída de contagem deve ser incrementada em uma unidade até que seu valor seja 9. Em um clock subsequente, a saída é zerada, reiniciando a contagem. Quando a entrada de reset estiver em nível baixo, o valor de contagem também é zerado.



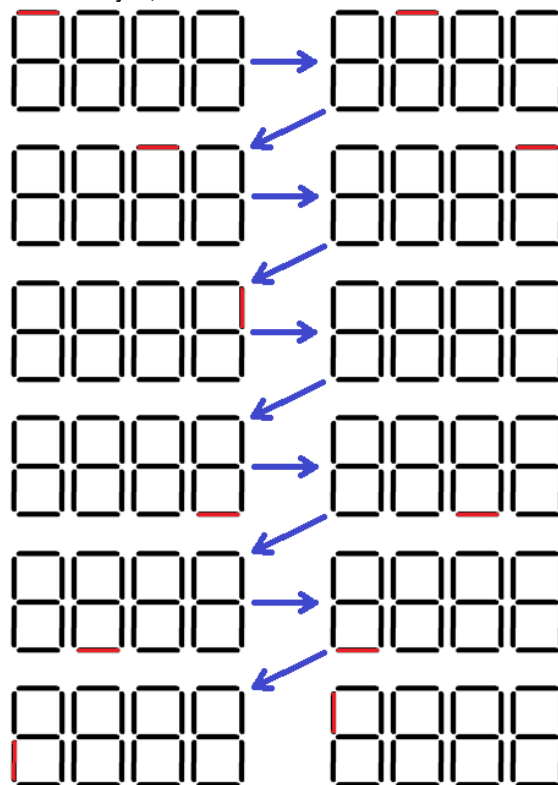
- b) Para testar seu módulo, implemente a seguinte montagem final, envolvendo todos os outros blocos implementados anteriormente:



- c) Caso o seu circuito passe por todos os testes, chame o professor para receber sua nota.
- d) Após o professor conferir seus testes, compacte o projeto em um .zip e submeta-o no Google Classroom da disciplina

#### Desafio (Valendo +0,1 na média geral)

1. Implemente uma animação nos displays HEX0, HEX1, HEX2 e HEX3, de modo que pareça que seus segmentos estejam executando uma grande rotação, a **60RPM**.



Aluno: \_\_\_\_\_

Matrícula: \_\_\_\_\_ Data: \_\_\_\_\_

### Sprint 3 – Banco de registradores – Processador RISC-V

**Descrição geral do problema:** Implemente um banco com 8 registradores de 8 bits. Esse banco deverá possuir um barramento de escrita síncrono (dados + endereço) e dois barramentos de leitura combinacionais (dados + endereços)

#### Requisitos mínimos:

Abra o projeto da Sprint2 e edite-o para incluir as funcionalidades dessa sprint. **Obs: “File > Open Project” e NÃO “File > Open”.**

1. Faça a descrição de hardware de um módulo, denominado RegisterFile, que gere o circuito indicado na Figura 1, usando a linguagem Verilog/SystemVerilog.

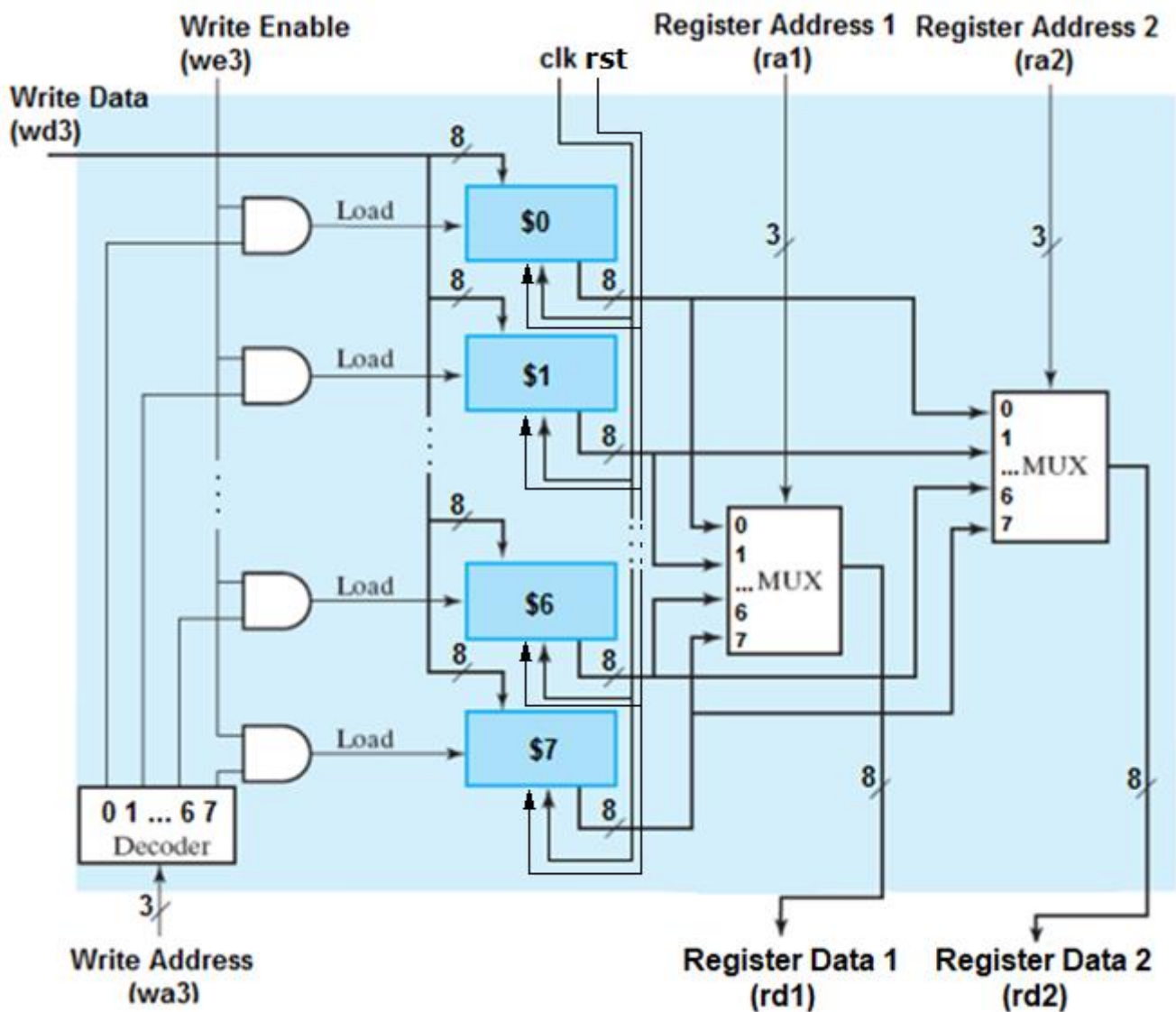


Figura 1 –Diagrama do módulo RegisterFile.

O módulo apresenta as seguintes entradas e saídas:

**Entradas:**

- Write Data (wd3)* (8 bits) – Entrada de dados;
- Write Address (wa3)* (3 bits) – Seleção do registrador que armazenará o dado proveniente de *Write Data (wd3)*;
- Write Enable (we3)* (1 bit) – Habilita (1) ou desabilita (0) a gravação de dados nos registradores de \$1 a \$7;
- clk* (1 bit) – Se o sinal *Write Enable (we3)* estiver ativo (1), na borda de subida do clock, o dado é gravado no registrador selecionado.
- Register Address 1 (ra1)* (3 bits) – Seleção de qual registrador será disponibilizado na saída rd1;
- Register Address 2 (ra2)* (3 bits) – Seleção de qual registrador será disponibilizado na saída rd2;
- Reset (rst)* (1 bit) – Reseta o valor dos registradores, em nível baixo (0);

**Saídas:**

- Register Data 1 (rd1)* (8 bits) – Barramento de saída de 8 bits;
- Register Data 2 (rd2)* (8 bits) – Barramento de saída de 8 bits;

Consideração Importante: O circuito não precisa ficar idêntico à figura, modularizado em decodificadores e portas AND. O funcionamento é que deve ser preservado. Faça a descrição em alto nível, para ganhar produtividade.

Princípio de funcionamento em três partes:

- ESCRITA:** A operação de escrita deve ser sincronizada com a borda de subida do clock (clk). O valor de 8 bits de *Write Data (wd3)* é armazenado no registrador cujo índice é igual a *Write Address (wa3)* (3 bits) se *Write Enable (we3)* for 1. O valor dos registradores se mantém caso *Write Enable (we3)* seja 0. *Circuito sequencial síncrono* → *always @(posedge clock) ou always\_ff @(posedge clock)*. **OBS: O registrador \$0 é somente de LEITURA e seu valor deve ser fixo em ZERO.**
- LEITURA:** o valor do registrador de índice *ra1* é disponibilizado de forma contínua na saída *rd1* assim como o registrador de índice *ra2* na saída *rd2*. Variando qualquer entrada, as saídas são atualizadas. *Circuito Combinacional* → *always @(\*), always\_comb ou assign*.
- RESET:** limpa o valor dos registradores quando a entrada *rst* for 0. Decida se implementará essa funcionalidade de forma síncrona (no momento da subida do clock) ou assíncrona (na descida do rst).

Os dois multiplexadores que conectam os registradores às saídas *rd1* e *rd2* são circuitos combinacionais. Os dados gravados nos registradores selecionados, sempre estarão disponíveis nas respectivas saídas.

2. Durante a instanciação do RegisterFile, no módulo “Mod\_Teste”, faça as seguintes conexões:

Direção da Porta	Nome da porta	Conectar no fio
Entradas	clk	KEY[1]
	we3	SW[17]
	wa3	SW[16:14]
	ra1	SW[13:11]
	ra2	SW[10:8]
	wd3	SW[7:0]
	rst	KEY[2]
Saídas	rd1	w_d0x0[7:0]
	rd2	w_d0x1[7:0]

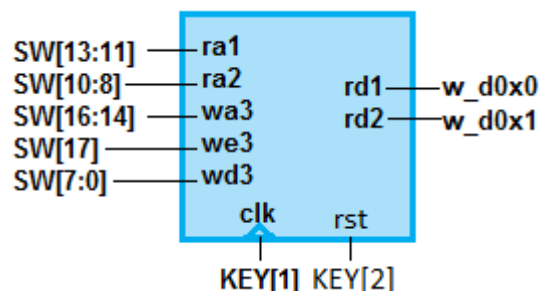
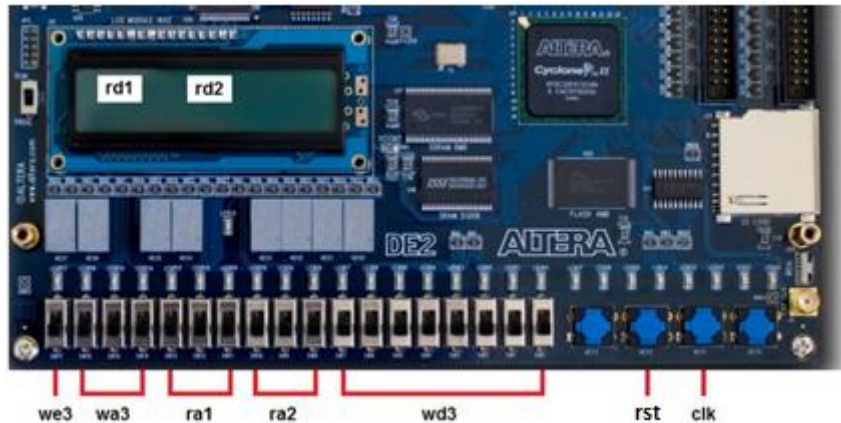


Figura 2 – Interfaces do módulo RegisterFile

Ligações auxiliares:

LEDG[8]	~KEY[1]	Dica: assign
HEX0 e HEX1	SW[7:0]	Dica: conversor <i>hex7seg</i> da sprint 2

3. Visualize a entrada *wd3* nos displays HEX0 e HEX1, assim como conecte as saídas *rd1* e *rd2* respectivamente nas posições *d0x0* e *d0x1* do LCD.
4. Para avaliar o funcionamento do circuito, realize a gravação de um conjunto de dados nos registradores. Após a gravação, verifique se os dados foram de fato gravados, selecionando *ra1* e *ra2* e observando as saídas *rd1* e *rd2*



TESTE: Escrever os seguintes valores nos registradores:

$\$1 \leftarrow CA$

$\$7 \leftarrow FE$

$\$0 \leftarrow DB$

$rst \leftarrow 0$

#### Desafio (Valendo +0,1 na média geral)

- Crie um testbench para o módulo desenvolvido e simule exaustivamente as operações de escrita e leitura em cada um dos registradores do banco. Faça testes com verificação automática para todos os 8 registradores.

Aluno: \_\_\_\_\_  
Matrícula: \_\_\_\_\_ Data: \_\_\_\_\_

### Sprint 4 – Unidade Lógica e Aritmética ULA – CPU RISC-V

**Descrição geral do problema:** Implemente uma ULA com 5 operações lógicas/aritméticas e associe-a às saídas do seu banco de registradores.

#### Requisitos mínimos:

Abra o projeto da Sprint3 e edite-o para incluir as funcionalidades dessa sprint. **Obs: “File > Open Project” e NÃO “File > Open”.**

1. Faça a descrição de hardware de um módulo, denominado ULA, que realize 5 operações lógicas/aritméticas conforme a Tabela 1. As entradas e saídas do módulo são ilustradas na Figura 1.

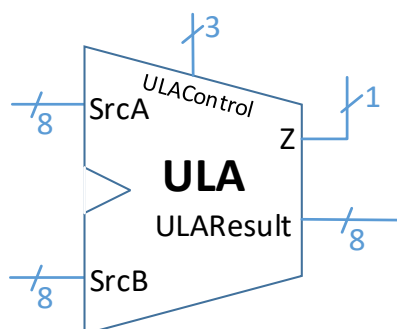


Figura 1 – Módulo ULA.

Operação	ULAControl	ULAResult
Add	3'b000	= SrcA + SrcB
Subtract	3'b001	= SrcA + $\overline{\text{SrcB}}$ + 1
And	3'b010	= SrcA & SrcB
Or	3'b011	= SrcA   SrcB
Set less than (SLT)	3'b101	1, se SrcA < SrcB 0, c.c.

Tabela 1. Operações da ULA

#### Entradas:

- SrcA (8bits): Entrada de dados do operando A;
- SrcB (8bits): Entrada de dados do operando B;
- ULAControl (3bits): Entrada para seleção da operação realizada (ver Tabela 1).

#### Saídas:

- ULAResult (8bits): Saída do resultado da operação realizada;
- Flag Z (1bit): Bit de status que indica se a saída da operação realizada é zero (resultado igual a zero: Z=1; resultado diferente de zero: Z=0).

**OBS: O módulo ULA é assíncrono (circuito combinacional).**

2. Instancie a ULA, o banco de registradores (implementado na sprint anterior) e um MUX de 2x1 de 8 bits, no seu ambiente de testes (Mod\_Teste). A montagem completa é ilustrada na Figura 2. Observe que o MuxULASrc possibilita a entrada direta de constantes de 8 bits na entrada SrcB da ULA.

Devido à quantidade limitada de chaves na placa de testes, algumas entradas serão constantes.



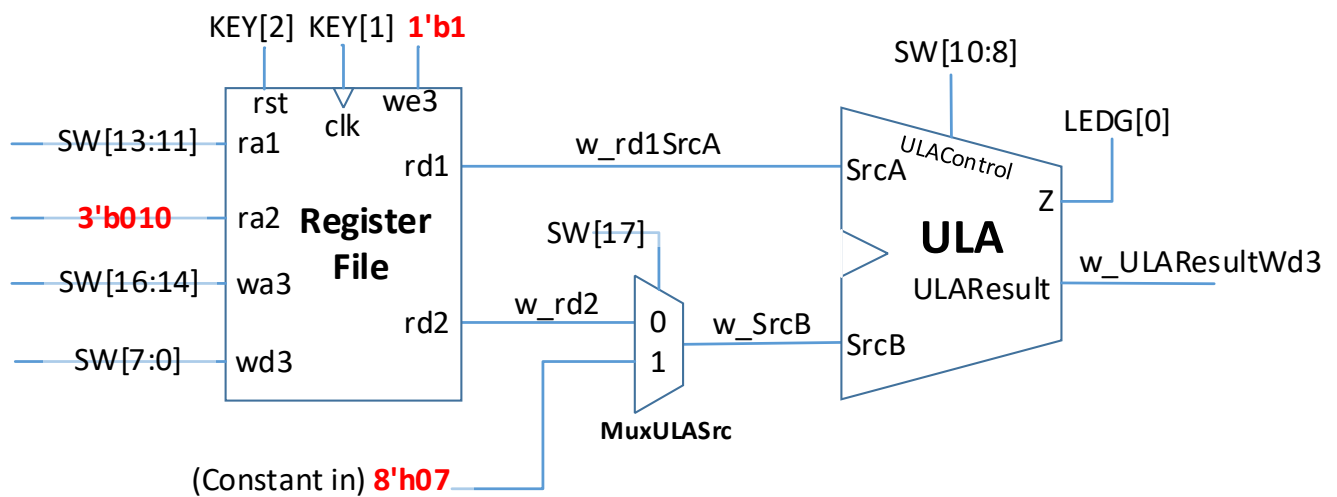


Figura 2 – Diagrama do Datapath

3. Ligações auxiliares:

- Visualize o conteúdo do fio **w\_rd1SrcA** na posição **d0x0** do LCD
- Visualize o conteúdo do fio **w\_rd2** na posição **d1x0** do LCD
- Visualize o conteúdo do fio **w\_SrcB** na posição **d1x1** do LCD
- Visualize o conteúdo do fio **w\_ULAResultWd3** na posição **d0x4** do LCD

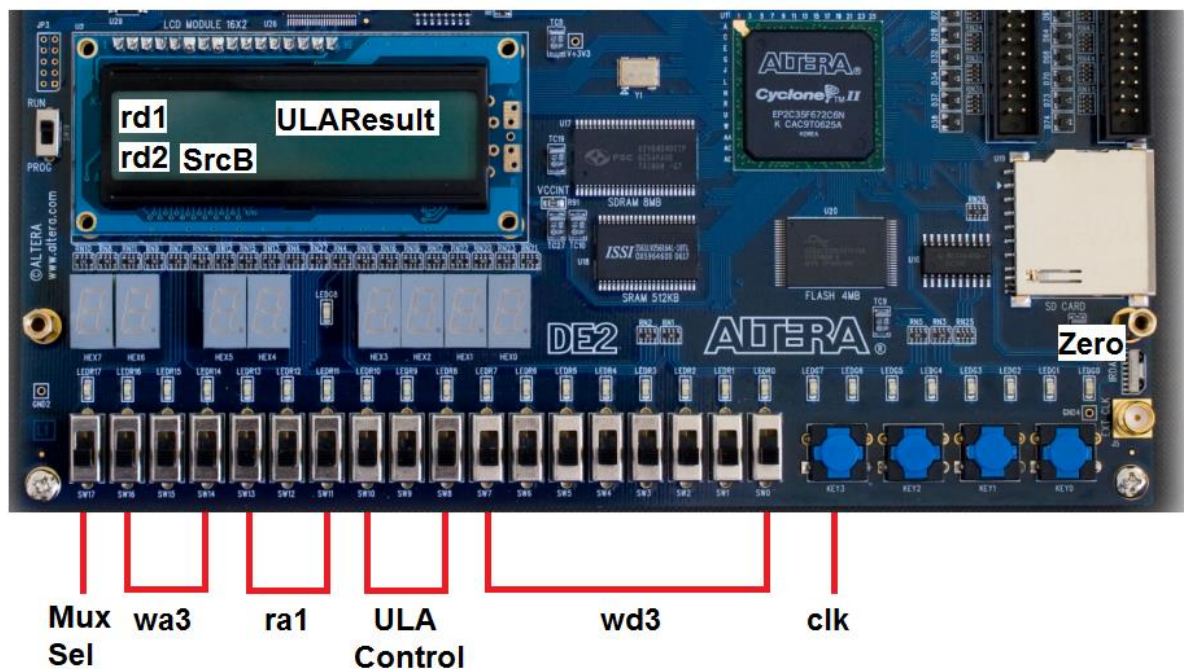


Figura 3 – Placa de testes

4. A fim de testar o funcionamento da ULA implementada realize os seguintes procedimentos:

- Carregue os seguintes valores nos registradores: **\$5 ← 06**      **\$2 ← 03**.
- Disponibilize os valores de **\$5** e **\$2** respectivamente nas entradas **SrcA** e **SrcB** da ULA
- Efetue as operações: Add \_\_\_\_, Sub \_\_\_\_, And \_\_\_\_, Or \_\_\_\_ e SLT \_\_\_\_
- Carregue o valor da **Constant in (8'h07)** na entrada **SrcB** da ULA
- Efetue novamente as operações: Add \_\_\_\_, Sub \_\_\_\_, And \_\_\_\_, Or \_\_\_\_ e SLT \_\_\_\_
- **Verifique o funcionamento do flag Z**

**Desafio (Valendo +0,1 na média geral)**

- Pesquise como transformar seus módulos .v/.sv em blocos .bdf (Block Diagram/Schematic Files) e realize a modelagem do circuito dessa sprint de forma visual, arrastando os blocos e desenhando os fios.

Aluno: \_\_\_\_\_  
Matrícula: \_\_\_\_\_ Data: \_\_\_\_\_

### Sprint 5 – Unidade de Controle – CPU RISC-V

**Descrição geral do problema:** Implementar a unidade de controle, memória de instruções e o registrador PC a fim de obter a primeira versão do nosso processador RISC-V v0.1. Esse circuito será capaz de rodar as seguintes instruções *addi*, *add*, *sub*, *and*, *or* e *slt*.

#### Requisitos mínimos:

Abra o projeto da Sprint4 e edite-o para incluir as funcionalidades dessa sprint. **Obs: “File > Open Project” e NÃO “File > Open”.**

- É ilustrado na Figura 1 o diagrama de blocos do processador RISC-V v0.1. Além do banco de registradores e da ULA implementados anteriormente, será necessário incluir:
  - Uma memória de instruções, para armazenar o programa a ser executado;
  - O registrador PC, que determina o endereço da instrução atual;
  - A unidade de controle, responsável por gerar os sinais de controle do *datapath*.

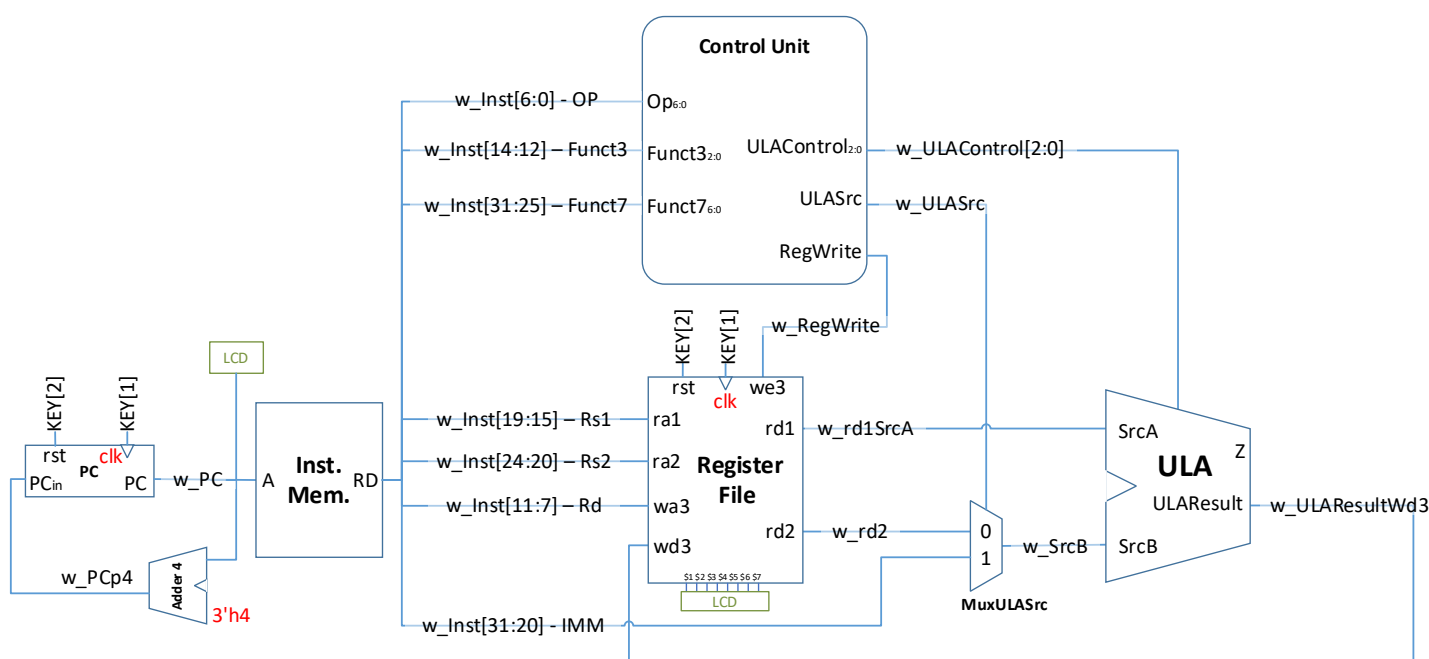


Figura 1 – CPU v0.1

Os barramentos devem ser criados com uma nomenclatura lógica para facilitar o entendimento geral do circuito e facilitar o debug. Seguem sugestões na Tabela 1

Nome	Tamanho
w_ULASrc	1 bit
w_RegWrite	1 bit
w_ULACtrl	3 bits
w_PCp4	8 bits
w_PC	8 bits

w_rd1SrcA	8 bits
w_rd2	8 bits
w_SrcB	8 bits
w_ULAResultWd3	8 bits
w_Inst	32 bits

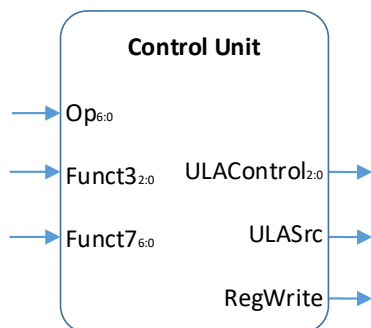
Tabela 1 –fios utilizados na montagem da Figura 1

A CPU RISC-V v0.1 será capaz de rodar as 6 instruções da Tabela 2

Instrução	Descrição	Algoritmo
ADD \$X, \$Y, \$Z	Adicionar	$\$X = \$Y + \$Z$
SUB \$X, \$Y, \$Z	Subtrair	$\$X = \$Y - \$Z$
AND \$X, \$Y, \$Z	AND Bit a bit	$\$X = \$Y \& \$Z$
OR \$X, \$Y, \$Z	OR Bit a bit	$\$X = \$Y   \$Z$
SLT \$X, \$Y, \$Z	Menor que	$\$X = 1$ se $\$Y < \$Z$ e 0 c.c.
ADDi \$X, \$Y, i	Adicionar Imediato	$\$X = \$Y + i$

Tabela 2 –Conjunto de instruções RISC-V suportadas pela CPU v0.1

2. Faça a descrição de hardware, em Verilog/SystemVerilog, da Unidade de Controle indicada na figura 2. Esse módulo gera os sinais de controle para cada uma das instruções suportadas pela CPU.



- Esse circuito é um decodificador COMBINACIONAL;
- A relação lógica entre as entradas e saídas é definida pela Tabela 3;
- Utilize a estratégia que preferir para a codificação. A título de inspiração, é possível implementar esse circuito com 1 ou 2 “cases” (Lembrar do “default”).

Figura 2 – Unidade de Controle.

Instr	ENTRADAS			SAÍDAS		
	OP	Funct3	Funct7	RegWrite	ULASrc	ULAControl
ADD	0110011	000	0000000	1	0	000
SUB	0110011	000	0100000	1	0	001
AND	0110011	111	0000000	1	0	010
OR	0110011	110	0000000	1	0	011
SLT	0110011	010	0000000	1	0	101
ADDi	0010011	000	xxxxxxx	1	1	000

Tabela 3 – Tabela do decodificador da Unidade de Controle

Nessa sprint, focaremos em instruções do tipo R (add, sub, and, or e slt) e I (addi), cujo formato está definido na Tabela 4.

	31:25	24:20	19:15	14:12	11:7	6:0
Tipo R	Funct76:0	Rs24:0	Rs14:0	Funct32:0	Rd4:0	Op6:0
Tipo I	Imm11:0		Rs14:0	Funct32:0	Rd4:0	Op6:0

Tabela 4 – Regra de formação do código de máquina das instruções MIPS

3. Faça a descrição de hardware, em Verilog/SystemVerilog, do Registrador PC (Program Counter). Tal componente é um registrador de 8 bits com uma entrada de clock (clk), uma entrada de reset (rst), uma entrada para carregamento paralelo (PCin) e uma saída paralela (PC).

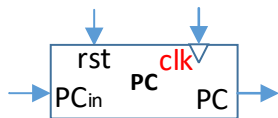


Figura 3 – PC.

- Esse circuito é SEQUENCIAL. Depende da borda de subida do clock.
- A cada clock, o novo *PCin* é carregado no registrador e disponibilizado na saída *PC*
- Caso o *rst* esteja em nível baixo, o valor do PC deve ser zerado

4. O último elemento a ser implementado é a memória de instruções. Nessa sprint, deverá ser implementada uma memória puramente combinacional, com até 256 posições de 32 bits.

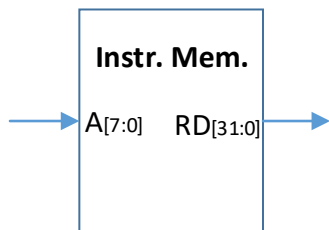


Figura 4 – Instruction Memory.

- Esse circuito é COMBINACIONAL;
- Essa memória será somente de LEITURA (ROM);
- Seguem algumas sugestões de implementação, para inspiração:
  - Um decodificador usando um *case*;
  - Um array bidimensional inicializado por um *.txt*, *.mif* ou *.hex*;
  - Um IP de ROM, disponível no Quartus (Tools > MegaWizard Plug-in Manager);
- O conteúdo da memória será definido pelo código de máquina da sequência de instruções a serem executadas (programa). Nessa sprint, deve-se rodar o programa de testes da Tabela 5.

Endereço	Assembly	Código de máquina (hexa)	Código de máquina (binário)
8'h00	addi x1, x0, 3	00300093	00000000011_0000_000_0001_0010011
8'h04	addi x2, x0, 9	00900113	00000001001_0000_000_0010_0010011
8'h08	add x2, x1, x2	00208133	0000000_00010_00001_000_00010_0110011
8'h0C	and x3, x1, x2	0020f1b3	0000000_00010_00001_111_00011_0110011
8'h10	or x4, x1, x2	0020e233	0000000_00010_00001_110_00100_0110011
8'h14	slt x5, x3, x4	0041a2b3	0000000_00100_00011_010_00101_0110011
8'h18	sub x6, x4, x5	40520333	0100000_00101_00100_000_00110_0110011

Tabela 5 –programa teste

5. Ligações auxiliares para Debug:

- Faça uma pequena alteração no módulo RegisterFile. Crie 8 saídas auxiliares para visualizar externamente os valores de cada um dos registradores. Ao instanciar o módulo, faça as seguintes ligações: *.x0(w\_d0x0)*, *.x1(w\_d0x1)*, *.x2(w\_d0x2)*, *.x3(w\_d0x3)*, *.x4(w\_d1x0)*, *.x5(w\_d1x1)*, *.x6(w\_d1x2)*, *.x7(w\_d1x3)*. Nesse caso, o LCD deverá ficar conforme a Figura 6.
- Mostre também o PC, na posição *w\_d0x4* do LCD. (assign)
- Visualize o código de máquina da instrução sendo executada (*w\_Inst*) nos displays HEX0-HEX7
- Visualize os 3 sinais de controle gerados pelo módulo "Control Unit" nos LEDs vermelhos. LEDR[4:0]. (assign)

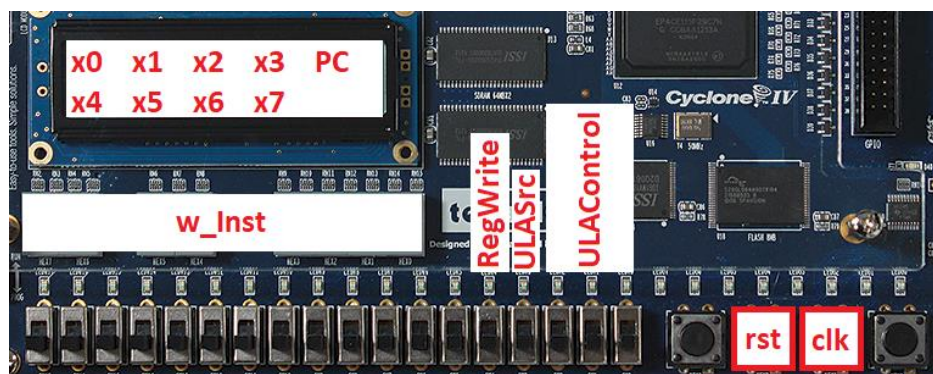


Figura 6 – Placa

6. Roteiro de testes

- O circuito proposto tem quantas entradas externas?
- Simule o programa da Tabela 5 no software [RARS](#)
- Carregue no FPGA e compare os resultados
- Qual o conteúdo final dos registradores?

**x0:\_\_\_\_, x1:\_\_\_\_, x2:\_\_\_\_, x3:\_\_\_\_, x4:\_\_\_\_, x5:\_\_\_\_, x6:\_\_\_\_, x7:\_\_\_\_**

**Desafio (Valendo +0,1 na média geral)**

- Escreva e rode nesse processador v0.1, uma rotina em assembly para detectar se o conteúdo do registrador x1 é múltiplo de 4.
- Caso o número seja múltiplo de 4, retorne 1 no registrador x7, caso contrário, 0.
- Utilize somente as 6 instruções da Tabela 3. Não inclua nenhum hardware adicional no processador.

**Desafio EXTRA (Valendo +0,5 na média geral)**

- O aluno que fizer o desafio utilizando menos instruções, receberá uma pontuação extra!
- Em caso de empate, ganha quem submeter o código antes;
- [LINK](#) para concorrer ao desafio extra!

Aluno: \_\_\_\_\_

Matrícula: \_\_\_\_\_ Data: \_\_\_\_\_

### Sprint 6 – Memórias – Processador RISC-V

**Descrição geral do problema:** Implemente uma memória RAM de dados e atualize o *datapath* e a unidade de controle para dar suporte às instruções LB e SB.

#### Requisitos mínimos:

Abra o projeto da Sprint5 e edite-o para incluir as funcionalidades dessa sprint. **Obs: “File > Open Project” e NÃO “File > Open”.**

1. Implemente uma nova memória RAM de 8bits dados e 8bits de endereço (256 posições). Dessa forma, será possível armazenar e carregar o conteúdo de registradores, através das instruções Sb e Lb.

**A:** Entrada de endereço – 8bits

**WD:** Entrada de dados (Escrita) – 8bits

**RD:** Saída de dados (Leitura) – 8bits

**WE:** Enable de escrita – 1bit

**rst:** *reset* da memória – 1bit

**clk:** *clock* de escrita – 1bit

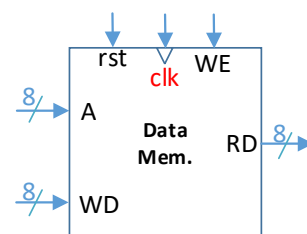


Figura 1 – Memória RAM de dados

#### Funcionamento:

- Leitura: Combinacional. Depende somente do endereço **A**
- Escrita: Sequencial. O dado **WD** é escrito no endereço **A**, na subida do **clk**, caso o **WE** esteja em nível alto.
- Reset: O conteúdo da memória deve ser apagado, quando ocorrer uma borda de descida no **rst**.

#### Sugestão para implementação:

- Um array bidimensional;
- Um IP de RAM, disponível no Quartus (Tools > MegaWizard Plug-in Manager);

2. Atualize a unidade de controle para gerar os sinais relativos às instruções LB e SB. Segue na Tabela 1 a lógica do novo decodificador.

	Instr	ENTRADAS			SAÍDAS					
		OP	Funct3	Funct7	RegWrite	ImmSrc	ULASrc	ULAControl	MemWrite	ResultSrc
R	ADD	0110011	000	0000000	1	x	0	000	0	0
	SUB	0110011	000	0100000	1	x	0	001	0	0
	AND	0110011	111	0000000	1	x	0	010	0	0
	OR	0110011	110	0000000	1	x	0	011	0	0
	SLT	0110011	010	0000000	1	x	0	101	0	0
I	ADDi	0010011	000	xxxxxxx	1	0	1	000	0	0
	LB	0000011	000	xxxxxxx	1	0	1	000	0	1
S	SB	0100011	000	xxxxxxx	0	1	1	000	1	x

Tabela 1 – Tabela do decodificador da Unidade de Controle

Lembrando que agora serão suportadas instruções do tipo R (add, sub, and, or e slt), I (addi e lb) e S (sb)

	31:25	24:20	19:15	14:12	11:7	6:0
<b>Tipo R</b>	Func7 <sub>6:0</sub>	Rs2 <sub>4:0</sub>	Rs1 <sub>4:0</sub>	Func3 <sub>2:0</sub>	Rd <sub>4:0</sub>	Op <sub>6:0</sub>
<b>Tipo I</b>	Imm <sub>11:0</sub>		Rs1 <sub>4:0</sub>	Func3 <sub>2:0</sub>	Rd <sub>4:0</sub>	Op <sub>6:0</sub>
<b>Tipo S</b>	Imm <sub>11:5</sub>	Rs2 <sub>4:0</sub>	Rs1 <sub>4:0</sub>	Func3 <sub>2:0</sub>	Imm <sub>4:0</sub>	Op <sub>6:0</sub>

Tabela 2 – Regra de formação do código de máquina das instruções RISC-V

3. Atualize o conteúdo da memória de instruções, com o código de máquina do programa definido na Tabela 3. Dica: utilize o RARs para converter o assembly em código de máquina.

Endereço	Assembly
8'h00	addi x1, x0, 0xAB
8'h04	sb x1, 0xA(x0)
8'h08	lb x2, 0xA(x0)
8'h0C	sb x2, 0xB(x0)
8'h10	lb x3, 0xB(x0)
8'h14	sb x3, 0xC(x0)
8'h18	lb x4, 0xC(x0)

Tabela 3 – Regra de formação do código de máquina das instruções RISC-V

4. A fim de completar a próxima versão da CPU v0.2, todos os módulos desenvolvidos até agora devem ser **instanciados** e **conectados** conforme o circuito da Figura 2.

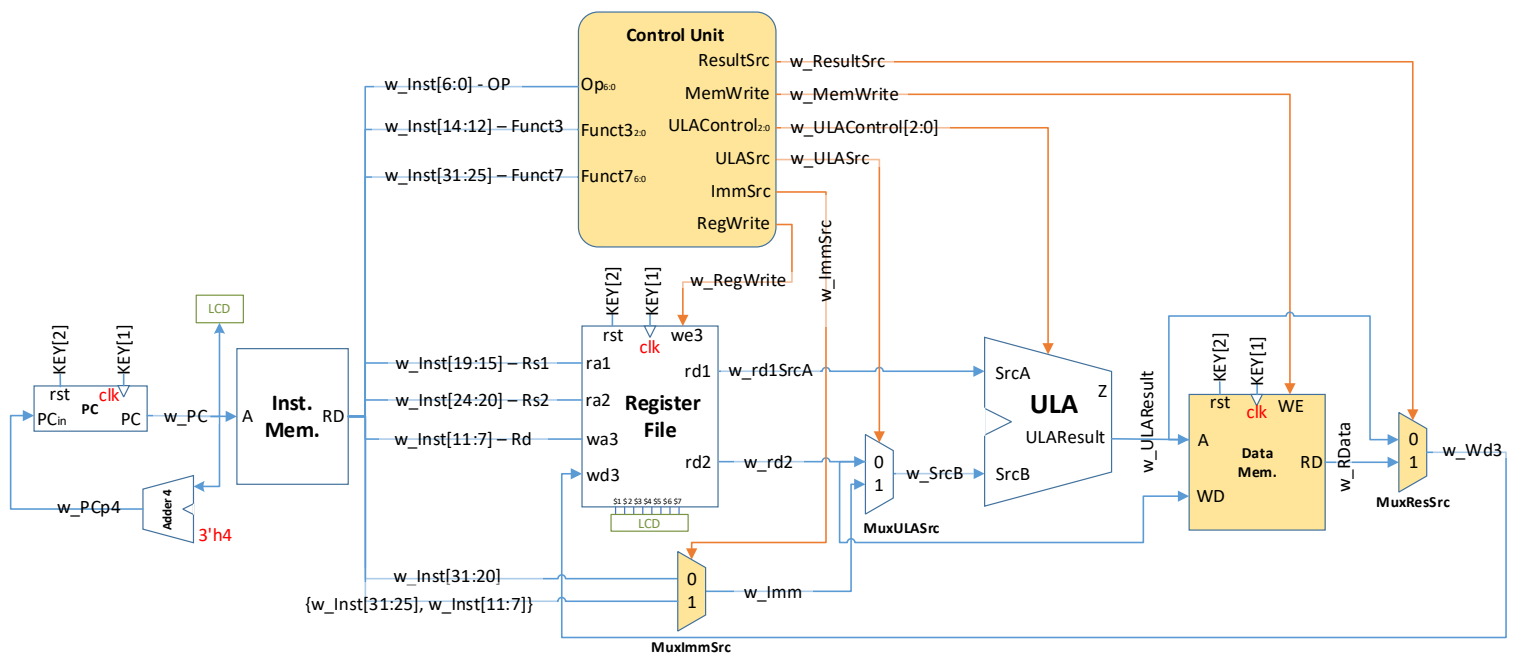


Figura 2 – Processador V0.2

Nome	Tamanho
w_ImmSrc	1 bit
w_MemWrite	1 bit
w_ResultSrc	1 bit

w_Wd3	8 bits
w_Imm	8 bits
w_RData	8 bits

Tabela 4 – novos fios, utilizados na montagem

OBS: O barramento anteriormente chamado de **w\_ResultWd3**, agora é chamado de **w\_Result**



5. Após o debug preliminar, substitua o clock manual do processador (KEY[1]) por um sinal de 1Hz gerado pelo divisor de frequência implementado na Sprint2.
6. Ligações auxiliares para Debug:
  - Visualize os 6 sinais de controle gerados pelo módulo “Control Unit” nos LEDs vermelhos. LEDR[7:0]. (assign)

Relembrando o conjunto de instruções suportadas pela CPU

Instrução	Descrição	Algoritmo
ADD \$X, \$Y, \$Z	Adicionar	$\$X = \$Y + \$Z$
SUB \$X, \$Y, \$Z	Subtrair	$\$X = \$Y - \$Z$
AND \$X, \$Y, \$Z	AND Bit a bit	$\$X = \$Y \& \$Z$
OR \$X, \$Y, \$Z	OR Bit a bit	$\$X = \$Y   \$Z$
SLT \$X, \$Y, \$Z	Menor que	$\$X = 1$ se $\$Y < \$Z$ e 0 c.c.
LB \$X, i(\$Y)	Carregar da memória	$\$X \leftarrow \text{end}[\$Y + i]$
SB \$X, i(\$Y)	Armazenar na memória	$\text{End}[\$Y + i] \leftarrow \$X$
ADDi \$X, \$Y, i	Adicionar Imediato	$\$X = \$Y + i$

Tabela 6 –Conjunto de instruções MIPS suportadas pela CPU do LASD

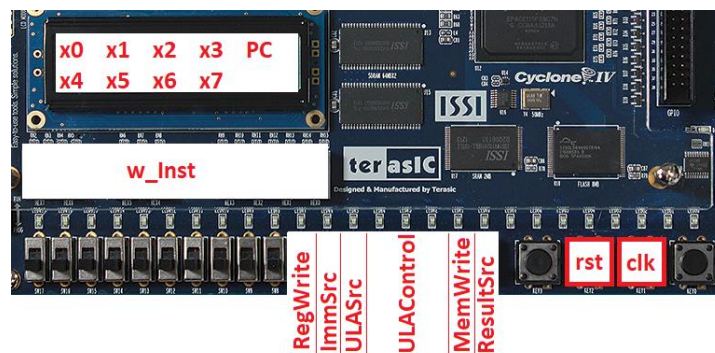


Figura 3 – Placa Altera DE2

2. Rode o programa da Tabela 1 e diga qual o conteúdo dos registradores e da memória de dados, ao finalizá-lo:

Registradores:

Registrador	x0	x1	x2	x3	x4	x5	x6	x7
Dado								

Memória de dados:

Endereço	0x00	0x01	0x02	0x03	...	0x0A	0x0B	0x0C	0x0D
Dado									

#### Desafio (Valendo +0,1 na média geral)

- Crie uma rotina, em assembly, que retorne a quantidade de dígitos 1s em um número de 4bits, previamente carregado no registrador x1.
- Retorne a quantidade de 1s no registrador x7.
- Utilize somente as 8 instruções da Tabela 6. Não inclua nenhum hardware adicional, porém sinta-se livre para iniciar a memória de dados com os valores que quiser.
- OBS: Essa rotina pode ser utilizada em problemas de paridade e de criptografia.

#### Desafio EXTRA (Valendo +0,5 na média geral)

- O aluno que fizer o desafio utilizando menos instruções, receberá uma pontuação extra!
- Em caso de empate, ganha quem submeter o código antes;
- [LINK](#) para concorrer ao desafio extra!



Aluno: \_\_\_\_\_

Matrícula: \_\_\_\_\_ Data: \_\_\_\_\_

### Sprint 7 – Desvios – Processador RISC-V

**Descrição geral do problema:** Modifique o processador da sprint anterior para dar suporte a instrução de desvio BEQ. Atualize o *datapath* e a unidade de controle.

#### Requisitos mínimos:

Abra o projeto da Sprint6 e edite-o para incluir as funcionalidades dessa sprint. **Obs: “File > Open Project” e NÃO “File > Open”.**

- Atualize a unidade de controle para gerar os sinais relativos à instrução BEQ. Segue na Tabela 1 a lógica do novo decodificador.

	Instr	ENTRADAS			SAÍDAS						
		OP	Funct3	Funct7	RegWrite	ImmSrc	ULASrc	ULAControl	MemWrite	ResultSrc	Branch
R	ADD	0110011	000	0000000	1	xx	0	000	0	0	0
	SUB	0110011	000	0100000	1	xx	0	001	0	0	0
	AND	0110011	111	0000000	1	xx	0	010	0	0	0
	OR	0110011	110	0000000	1	xx	0	011	0	0	0
	SLT	0110011	010	0000000	1	xx	0	101	0	0	0
I	ADDi	0010011	000	xxxxxxx	1	00	1	000	0	0	0
	LB	0000011	000	xxxxxxx	1	00	1	000	0	1	0
S	SB	0100011	000	xxxxxxx	0	01	1	000	1	x	0
B	BEQ	1100011	000	xxxxxxx	0	10	0	001	0	x	1

Tabela 1 – Tabela do decodificador da Unidade de Controle

Lembrando que agora serão suportadas instruções do tipo R (add, sub, and, or e slt), I (addi e lb), S (sb) e B (beq)

	31:25	24:20	19:15	14:12	11:7	6:0
<b>Tipo R</b>	Funct7 <sub>6:0</sub>	Rs2 <sub>4:0</sub>	Rs1 <sub>4:0</sub>	Funct3 <sub>2:0</sub>	Rd <sub>4:0</sub>	Op <sub>6:0</sub>
<b>Tipo I</b>	Imm <sub>11:0</sub>		Rs1 <sub>4:0</sub>	Funct3 <sub>2:0</sub>	Rd <sub>4:0</sub>	Op <sub>6:0</sub>
<b>Tipo S</b>	Imm <sub>11:5</sub>	Rs2 <sub>4:0</sub>	Rs1 <sub>4:0</sub>	Funct3 <sub>2:0</sub>	Imm <sub>4:0</sub>	Op <sub>6:0</sub>
<b>Tipo B</b>	Imm <sub>12,10:5</sub>	Rs2 <sub>4:0</sub>	Rs1 <sub>4:0</sub>	Funct3 <sub>2:0</sub>	Imm <sub>4,1:11</sub>	Op <sub>6:0</sub>

Tabela 2 – Regra de formação do código de máquina das instruções RISC-V

- Atualize o conteúdo da memória de instruções, com o código de máquina do programa definido na Tabela 3. Dica: utilize o RARs para converter o assembly em código de máquina.

```

addi x1, x0, 7      #inicializa x1 com 7
addi x3, x0, 3      #inicializa x3 com 3

init:
addi x2, x0, -1     #inicializa o contador x2 com -1

incremento:
addi x2, x2, 1      #incrementa x2
slt x7, x2, x3       #atualiza a saída x7
beq x1, x2, init     #se a condição de parada for atingida, reinicia a rotina
beq x0, x0, incremento #caso contrário, volta para incrementar x2
  
```

Alguma  
ideia de um  
possível uso  
para esse  
código?

Tabela 3 – Programa de testes dessa sprint

3. A fim de completar a próxima versão da CPU v0.3, todos os módulos desenvolvidos até agora devem ser **instanciados** e **conectados** conforme o circuito da Figura 2.

## Relembrando o conjunto de instruções suportadas pela CPU

Instrução	Descrição	Algoritmo
ADD \$X, \$Y, \$Z	Adicionar	$\$X = \$Y + \$Z$
SUB \$X, \$Y, \$Z	Subtrair	$\$X = \$Y - \$Z$
AND \$X, \$Y, \$Z	AND Bit a bit	$\$X = \$Y \& \$Z$
OR \$X, \$Y, \$Z	OR Bit a bit	$\$X = \$Y   \$Z$
SLT \$X, \$Y, \$Z	Menor que	$\$X = 1 \text{ se } \$Y < \$Z \text{ e } 0 \text{ c.c.}$
LB \$X, i(\$Y)	Carregar da memória	$\$X \leftarrow \text{end}[\$Y + i]$
SB \$X, i(\$Y)	Armazenar na memória	$\text{End}[\$Y + i] \leftarrow \$X$
ADDi \$X, \$Y, i	Adicionar Imediato	$\$X = \$Y + i$
BEQ \$X, \$Y, i	Desviar se igual	Se $\$X == \$Y$ , $PC = PC + i$

Tabela 6 –Conjunto de instruções RISC-V suportadas pela CPU do LASD

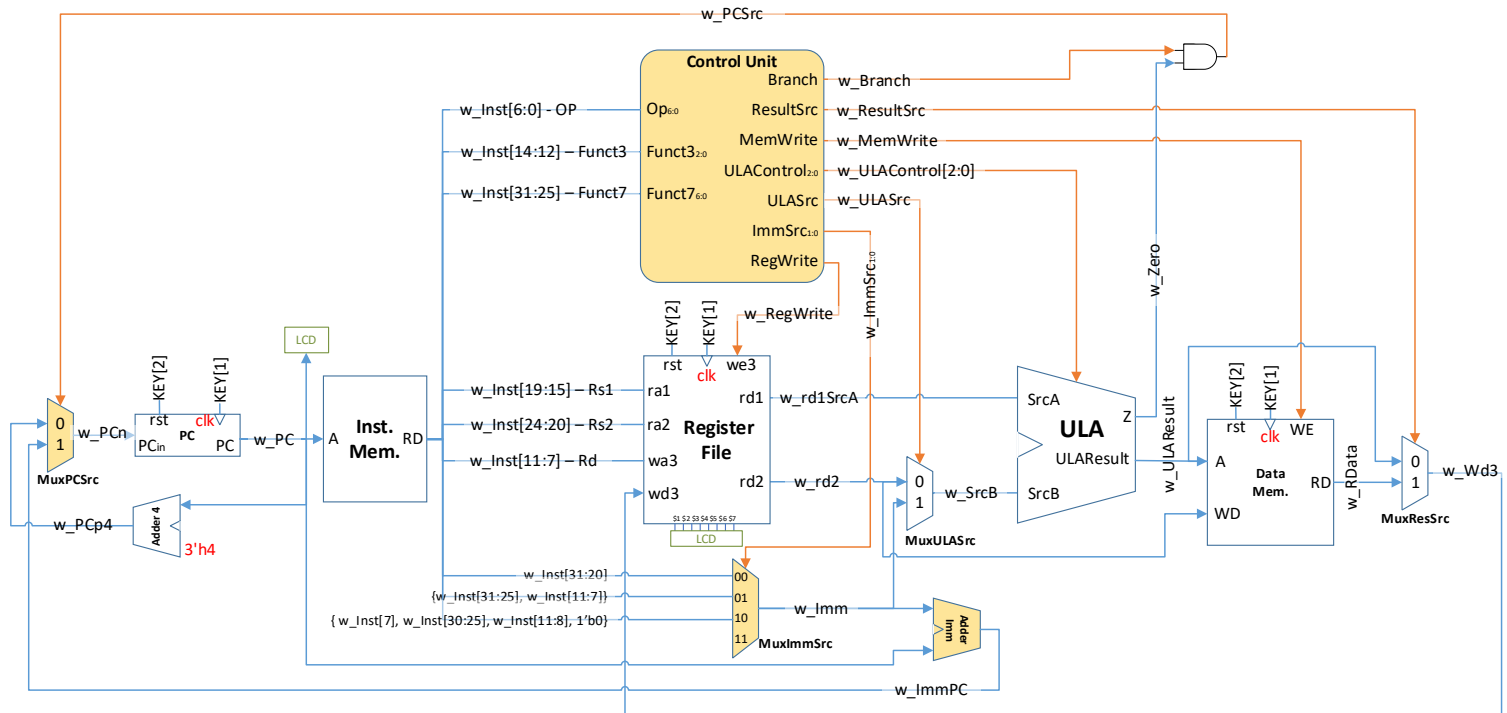


Figura 2 – Processador V0.3

Nome	Tamanho
w_PCSrc	1 bit
w_Zero	1 bit
w_Branch	1 bit

w_ImmSrc	2 bits
w_ImmPC	8 bits
w_PCn	8 bits

Tabela 4 – novos fios, utilizados na montagem

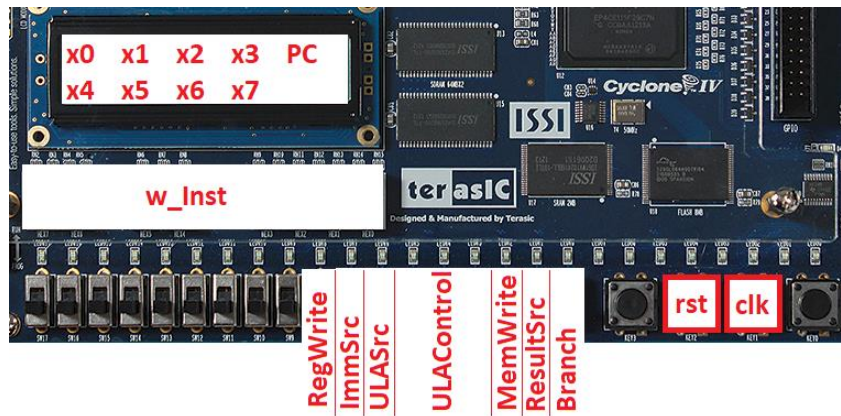


Figura 3 – Placa Altera DE2

5. Rode o programa da Tabela 1 e diga qual o conteúdo dos registradores e da memória de dados, ao finalizá-lo:

Registradores:

Registrador	x0	x1	x2	x3	x4	x5	x6	x7
Dado								

### Desafio (Valendo +0,5 na média geral)

- Adicione suporte para as instruções JAL e JALR. Isso possibilitará chamar sub-rotinas (JAL) e retornar das sub-rotinas (JALR);
- Teste seu projeto, escrevendo uma sub-rotina de delay, com duração de 500ms (Assuma que o clock da CPU é 100Hz);
- Utilize sua sub-rotina para criar uma onda quadrada de aproximadamente 1Hz no registrador \$6;

op	funct3	funct7	Type	Instruction	Description	Operation
1100111 (103)	000	-	I	jalr rd, rs1, imm	jump and link register	PC = rs1 + SignExt(imm), rd = PC + 4
1101111 (111)	-	-	J	jal rd, label	jump and link	PC = PC + SignExt({imm20:1,1'b0}), rd = PC + 4

Aluno: \_\_\_\_\_

Matrícula: \_\_\_\_\_ Data: \_\_\_\_\_

### Sprint 8 – Entrada e saída paralela – processador RISC-V

**Descrição geral do problema:** Incluir uma entrada e uma saída paralela mapeada em memória, de 8bits. Isso finalizará o conjunto mínimo de funcionalidades do processador.

#### Requisitos mínimos:

Abra o projeto da Sprint7 e edite-o para incluir as funcionalidades dessa sprint. **Obs: “File > Open Project” e NÃO “File > Open”.**

- Até esse momento, o processador v0.3 não tinha nenhuma forma de trocar dados com o mundo externo. Além das interfaces de debug, as únicas entradas externas da montagem eram o clock e o reset. A fim de completar a versão v1.0 do processador, inclua uma porta de entrada e uma de saída paralela, mapeada em memória.
  - O endereço 8'hFF da memória de dados será inutilizado e ressignificado para as portas de entrada e saída paralelas, mapeadas em memória.
  - Ao armazenar o conteúdo de algum registrador  $\$X$  no endereço 8'hFF da memória de dados, **SB**  $\$X$ , **FF(\$0)**, o bloco **ParallelOUT** redirecionará o conteúdo de  $\$X$  para a saída paralela **w\_DataOut**. A especificação lógica do circuito de saída está ilustrada na Figura 2
  - Ao carregar o conteúdo do endereço 8'hFF da memória de dados, para um registrador  $\$X$ , **LB**  $\$X$ , **FF(\$0)**, o bloco **ParallelIN** redirecionará o conteúdo da entrada paralela **w\_DataIn** para o registrador  $\$X$ . A especificação lógica do circuito de saída está ilustrada na Figura 3
  - A sugestão de montagem final da CPU v1.0 está representada na Figura 1.

Perceba que não foi necessário criar mais nenhuma instrução para manipular as portas. Somente **SB** e **LB**

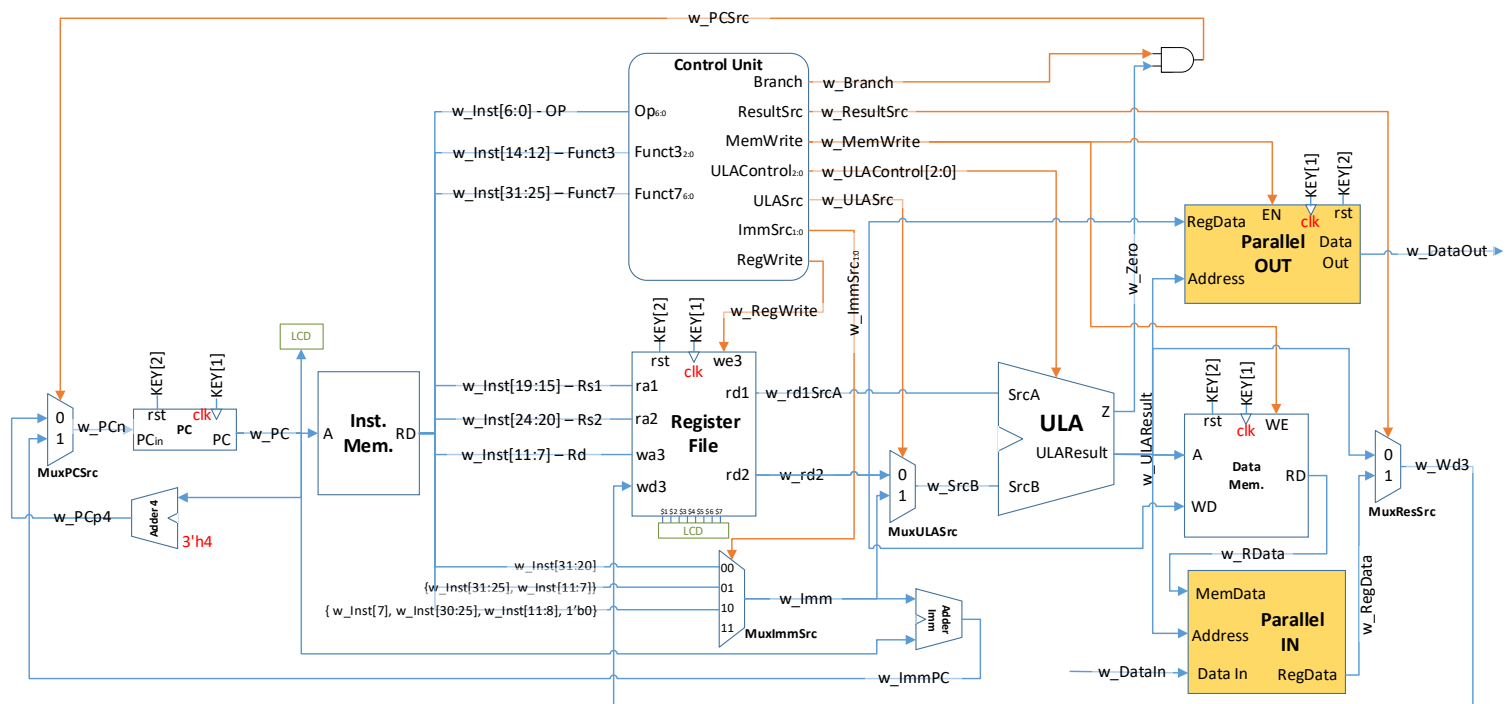


Figura 1 – CPU V1.0, memórias e porta de IO paralela de 8bits

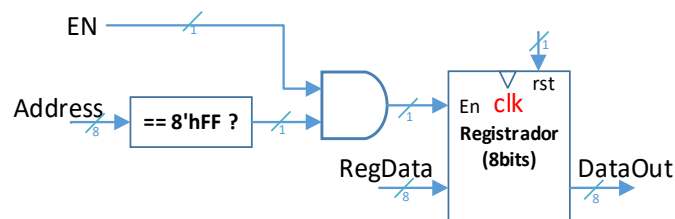


Figura 2 – Saída paralela mapeada no endereço 8'hFF da memória

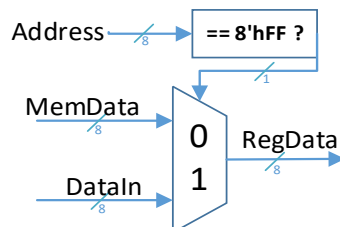


Figura 3 – Entrada paralela mapeada no endereço 8'hFF da memória

## 2. Ligações auxiliares para Debug:

- Conecte a saída paralela (DataOut) no display **w\_d1x4**,
- Conecte a entrada paralela (DataIn) nas chaves **SW[7:0]**

## 3. Roteiro de testes:

- Rode o programa da Tabela 1

```
init:
lb x1, 0xFF(x0)      #Carrega a entrada paralela no registrador 1
sb x1, 0xFF(x0)      #Salva o registrador 1 na saída paralela
beq x0, x0, init     #Reinicia o laço
```

Tabela 1 – Programa de testes A

- Escreva e rode na sua CPU, **um programa em assembly** que receba um número de 8bits na entrada paralela e calcule se ele é PAR ou ÍMPAR. Retorne, na saída paralela, **1** caso o número seja PAR e **0** caso seja ÍMPAR. Resolva esse problema por software, não é necessário nenhum hardware extra, que não tenha sido descrito previamente. Aumente o clock principal da CPU para 10Hz.

## Relembrando o conjunto de instruções suportadas pela CPU

Instrução	Descrição	Algoritmo
ADD \$X, \$Y, \$Z	Adicionar	$\$X = \$Y + \$Z$
SUB \$X, \$Y, \$Z	Subtrair	$\$X = \$Y - \$Z$
AND \$X, \$Y, \$Z	AND Bit a bit	$\$X = \$Y \& \$Z$
OR \$X, \$Y, \$Z	OR Bit a bit	$\$X = \$Y   \$Z$
SLT \$X, \$Y, \$Z	Menor que	$\$X = 1$ se $\$Y < \$Z$ e 0 c.c.
LB \$X, i(\$Y)	Carregar da memória	$\$X \leftarrow \text{end}[\$Y + i]$
SB \$X, i(\$Y)	Armazenar na memória	$\text{End}[\$Y + i] \leftarrow \$X$
ADDi \$X, \$Y, i	Adicionar Imediato	$\$X = \$Y + i$
BEQ \$X, \$Y, i	Desviar se igual	Se $\$X == \$Y$ , $\text{PC} = \text{PC} + i$

Tabela 2 –Conjunto de instruções RISC-V suportadas pela CPU do LASD

## Desafio 1 (Valendo +0,2 na média geral)

- Escreva um código, em assembly, para gerar um sinal PWM de 8bits no pino menos significativo da saída paralela;
- Configure o  $t_{on}$  (0-255) do PWM através da entrada paralela;
- Conecte as chaves SW[7:0] na entrada paralela, um LED na saída paralela e aumente o clock do processador para 5kHz.
- Altere o valor do  $t_{on}$  e veja o brilho do LED variar!

Aluno: \_\_\_\_\_  
Matrícula: \_\_\_\_\_ Data: \_\_\_\_\_

### Sprint 9 – Programação em Assembly – Processador RISC-V

**Descrição geral do problema:** Agora que o hardware do seu processador RISC-V já está pronto, você pode programá-lo, em assembly, como qualquer outro microcontrolador. Escreva códigos, em assembly, para resolver os problemas propostos e em seguida rode-os no seu próprio processador.

#### Requisitos mínimos:

Abra o projeto da Sprint8 e edite-o para incluir as funcionalidades dessa sprint. **Obs: “File > Open Project” e NÃO “File > Open”.**

1. Implemente uma rotina, em assembly de RISC-V, para gerar uma animação nos LEDs da placa.
  - Conecte as chaves SW[7:0] na entrada paralela do seu processador;
  - Conecte os LEDR[7:0] na saída paralela do seu processador;
  - Escreva uma rotina para gerar uma animação nos LEDs
    - i. Caso a entrada paralela seja igual a 1, os leds devem acender e apagar, em sequência, da direita para esquerda (ver o vídeo);
    - ii. Caso a entrada paralela seja igual a 0, a animação deve ser desligada;
2. Carregue seu código na memória de instruções e teste sua rotina variando a entrada paralela.

Relembrando o conjunto de instruções suportadas pela CPU

Instrução	Descrição	Algoritmo
ADD \$X, \$Y, \$Z	Adicionar	$\$X = \$Y + \$Z$
SUB \$X, \$Y, \$Z	Subtrair	$\$X = \$Y - \$Z$
AND \$X, \$Y, \$Z	AND Bit a bit	$\$X = \$Y \& \$Z$
OR \$X, \$Y, \$Z	OR Bit a bit	$\$X = \$Y   \$Z$
SLT \$X, \$Y, \$Z	Menor que	$\$X = 1 \text{ se } \$Y < \$Z \text{ e } 0 \text{ c.c.}$
LB \$X, i(\$Y)	Carregar da memória	$\$X \leftarrow \text{end}[\$Y + i]$
SB \$X, i(\$Y)	Armazenar na memória	$\text{End}[\$Y + i] \leftarrow \$X$
ADDi \$X, \$Y, i	Adicionar Imediato	$\$X = \$Y + i$
BEQ \$X, \$Y, i	Desviar se igual	Se $\$X == \$Y$ , $PC = PC + i$

Tabela 1 –Conjunto de instruções RISC-V suportadas pela CPU do LASD

#### Desafio (Valendo +0,5 na média geral) – JOGO do REFLEXO

- Para iniciar o jogo, deve-se acionar uma chave/botão;
- Um led deve acender e após um delay de 2 segundos apagar;
- Após o led apagar, o jogador deve pressionar uma chave/botão, o mais rápido possível;
- O jogo irá calcular o tempo de reflexo do jogador e mostrará a quantidade de ms de sua reação, nos displays de 7 segmentos. O valor deve estar entre 0-1023ms, com resolução de 4ms.
- Após o fim do jogo, o mesmo deve reiniciar automaticamente.

Aluno: \_\_\_\_\_

Matrícula: \_\_\_\_\_

Data: \_\_\_\_\_

### **Sprint 10 – Novas funcionalidades – Processador RISC-V**

**Descrição geral do problema:** Nessa sprint final, cada aluno deverá criar novas funcionalidades para seu processador RISC-V e utiliza-lo em uma aplicação prática. **A criatividade será extremamente valorizada!**

#### **Requisitos mínimos:**

1. Acrescente, no mínimo, 3 novos componentes de hardware no circuito do seu processador.
  - Seguem alguns exemplos para inspiração:
    - i. Suporte a novas instruções: JAL, JR, ANDi, Ori, BNE, NOR, SRL;
    - ii. Suporte a interrupções;
    - iii. Novos periféricos: outras GPIO paralelas, Timers/counters, RS232, PS/2, Ethernet, IrDA, I2C, SPI (ver datasheet da placa);
    - iv. Expansão do número de registradores;
    - v. Aumento da largura dos barramentos;
    - vi. Expansão da capacidade das memórias de dados e instruções;
    - vii. Arquitetura multiciclo ou com pipeline;
    - viii. etc...
  - Documente as novas funcionalidades no formato de um *help*
2. Projete uma aplicação prática, com potencial de uso real, baseada no seu novo processador RISC-V melhorado. Seja criativo.
  - Documente o seu projeto;
  - Implemente seu código com o máximo possível de boas práticas. Desacoplamento, otimização de recursos, reuso, etc;
  - Crie uma conta no GitHub e faça o upload do seu projeto;
  - Grave um vídeo de até 5 minutos explicando as novas funcionalidades, descrevendo o código e realizando uma demonstração de funcionamento. A demonstração pode ser totalmente simulada, de modo que você possa fazer tudo em casa. Caso deseje testar na placa, poderá fazê-lo nos horários de atendimento. Pode utilizar o próprio Quartus ou qualquer outra ferramenta de simulação, como o <https://www.edaplayground.com/>.
  - Compacte todos os arquivos do seu projeto e submeta no Classroom até o deadline.