

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

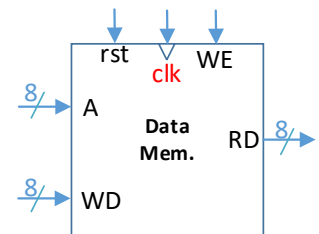


Figura 1 – Memória RAM de dados

#### 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 > MegaWizzard 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.

		ENTRADAS			SAÍDAS					
	Instr	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.

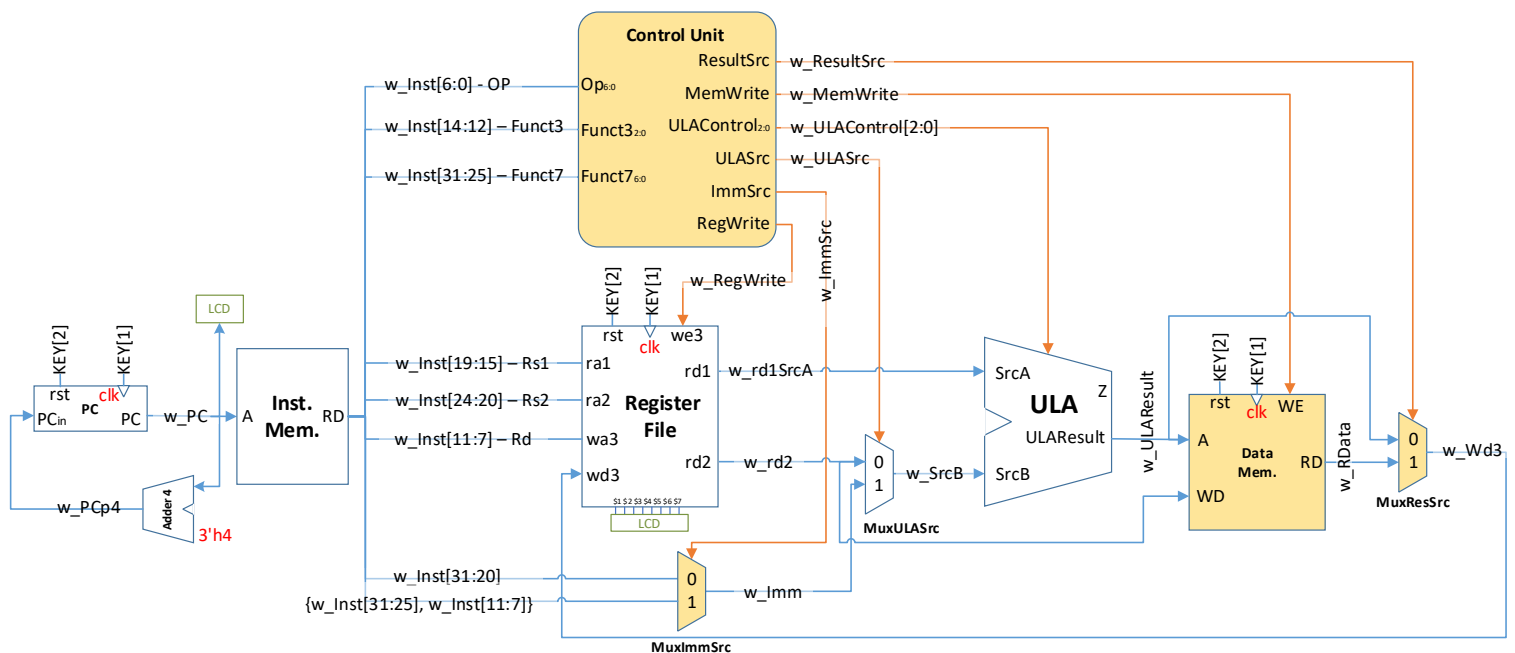


Figura 2 – Processador V0.2

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

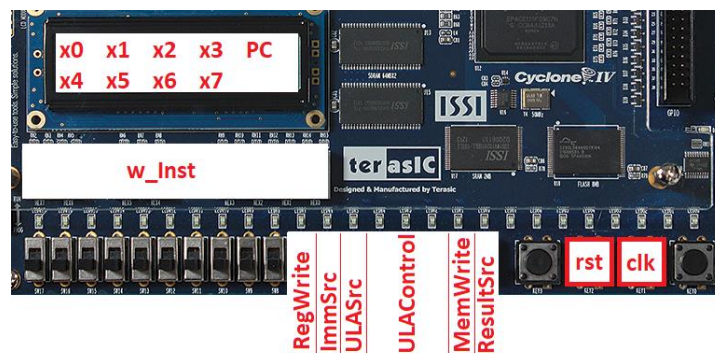


Figura 3 – Placa Altera DE2

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!