Universidade Federal De Campina Grande Departamento De Engenharia Elétrica
Laboratório De Arquitetura De Sistemas Digitais – LASD
Prof. Rafael B. C. Lima

Aluno: _____

Matrícula: _____ Data: _____

Sprint 5 – Unidade de Controle – Processador 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".**

1. 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;
   - Um extensor de sinal, para operações com o Imediato;
   - A unidade de controle, responsável por gerar os sinais de controle do *datapath*.

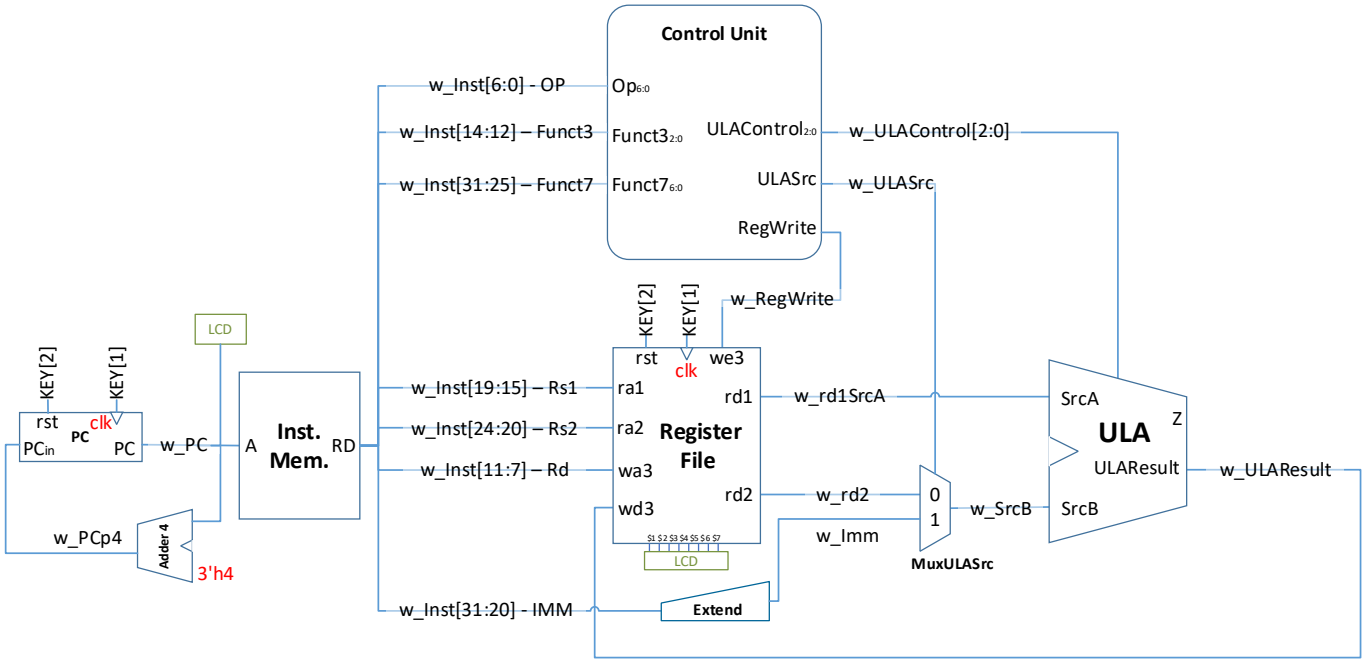


Figura 1 – Processador v0.1

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_ULAControl	3 bits
w_PCp4	32 bits
w_PC	32 bits

w_Imm	32 bits
w_rd1SrcA	32 bits
w_rd2	32 bits
w_SrcB	32 bits
w_ULAResult	32 bits
w_Inst	32 bits

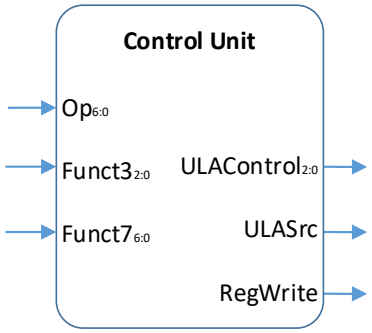
Tabela 1 –fios utilizados na montagem da Figura 1

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.

	ENTRADAS			SAÍDAS		
Instr	**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	$Funct7_{6:0}$	$Rs2_{4:0}$	$Rs1_{4:0}$	$Funct3_{2:0}$	$Rd_{4:0}$	$Op_{6:0}$
Tipo I	$Imm_{11:0}$		$Rs1_{4:0}$	$Funct3_{2:0}$	$Rd_{4:0}$	$Op_{6:0}$

Tabela 4 – Regra de formação do código de máquina das instruções RISC-V

Utilize o testbench fornecido (UC_SP5_TB.sv e test_vector.txt) para simular seu módulo da Unidade de Controle no ambiente https://edaplayground.com/ . Certifique-se que todos os testes rodaram sem falhas (“Passou”), antes de prosseguir para a próxima etapa. Alguns exemplos, podem ser encontrados na seguinte videoaula sobre Testbenches no EDAplayground: https://www.youtube.com/watch?v=VsP6zHarUSM .

3. Faça a descrição de hardware, em Verilog/SystemVerilog, do Registrador PC (Program Counter). Tal componente é um registrador de 32 bits com uma entrada de clock (clk), uma entrada de reset (rst), uma entrada para carregamento paralelo (*PCin)* e uma saída paralela (PC).

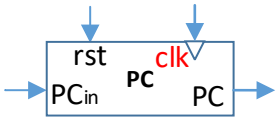


Figura 3 – PC.

- 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. Devido ao fato das instruções do tipo I possuírem operandos imediatos de 12bits e a ULA operar em 32bits, é necessário incluir um módulo extensor de sinal para viabilizar operações com imediatos negativos.

Figura 4 – Extensor de sinal.

- Esse circuito é COMBINACIONAL;
- Deverá existir uma entrada de 12bits no formato: 12'b_**sxxx_xxxx_xxxx**
- A saída deverá ser uma palavra de 32bits, cujos 20 bits mais significativos serão cópias do bit mais significativo da entrada (extensão de sinal): 32'b_ssss_ssss_ssss_ssss_ssss_**sxxx_xxxx_xxxx**

5. O último elemento a ser implementado é a memória de instruções. Nessa sprint, deverá ser implementada uma memória puramente combinacional, com endereços parametrizáveis entre 10bits e 32bits. Serão palavras de 32bits, endereçadas de 8 em 8 bits (ou seja, cada instrução de 32bits, ficará armazenada em 4 endereços consecutivos).

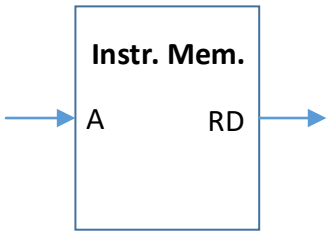


Figura 5 – Instruction Memory.

- 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; (**Preferencial!**)
  - Um IP de ROM, disponível no Quartus (Tools > MegaWizzard 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, 0xF3	0f300093	000011110011_00000_000_00001_0010011
8'h04	addi x2, x0, 9	00900113	000000001001_00000_000_00010_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 x6, x3, x4	0041a333	0000000_00100_00011_010_00110_0110011
8'h18	sub x7, x4, x6	406203b3	0100000_00110_00100_000_00111_0110011

Tabela 5 –programa teste

6. 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)

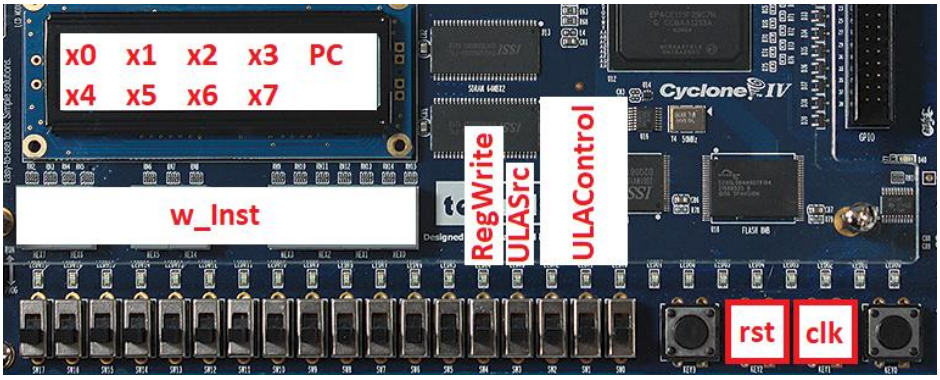


Figura 6 – Placa

7. 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 8.
- Caso o número seja múltiplo de 8, retorne 1 no registrador x7, caso contrário, 0.
- Utilize somente as 7 instruções da Tabela 3. Não inclua nenhum hardware adicional no processador.
- Inclua comentários, para explicar sua rotina.

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!