



- CPU RISC-V MULTICICLO -

Alunos:

Gabriel Castro - Matrícula: 202066571

Lucas Santana - Matrícula: 211028097

OAC Unificado – 2025/1

Link do repositório GitHub: [Repositório Grupo B3](#)

1.1)

O processador multiciclo implementado segue o modelo de Arquitetura Von Neumann, ou seja, a memória de instruções e a memória de dados compartilham o mesmo espaço físico, como pode ser melhor visualizado na figura abaixo.

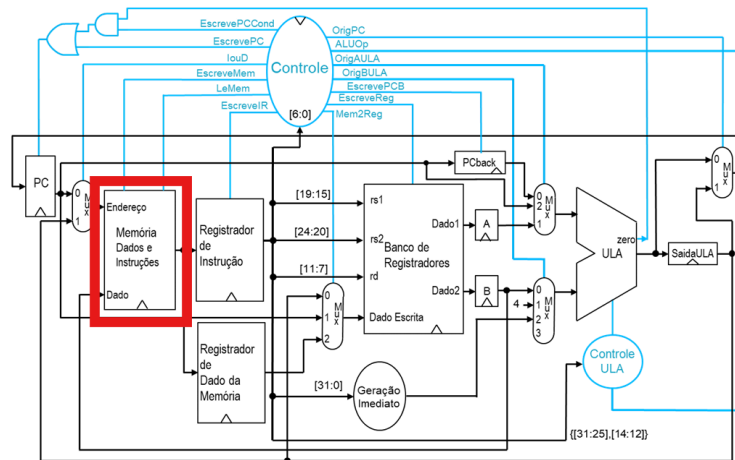


Figura 1 - Processador Multiciclo

Devido ao fato de as instruções e os dados estarem armazenados na mesma memória física, foi necessário criar um sinal de controle específico para selecionar o tipo de acesso desejado: o sinal **loud** (*Instruction or Data*).

O funcionamento desse sinal é descrito na tabela a seguir:

Tabela 1: Funcionamento do sinal **loud**.

loud	0	Acesso à memória de instruções.	O endereço vem do PC.
	1	Acesso à memória de dados.	O endereço vem da ULA, esta calcula o endereço efetivo.

1.2)

A fim de lidar com essa característica da memória IP utilizada pelo Quartus possui uma latência de 2 ciclos de clock (um ciclo para fornecer o endereço e iniciar leitura e o segundo para disponibilizar o dado na saída). Para adaptar o diagrama de estados do processador multiciclo criamos um estado intermediário. Abaixo está a explicação detalhada:

O ciclo de busca de instrução em dois estados:

- **ST_FETCH_1**: Fornece o endereço da instrução (vindo do PC) para a memória.
- **ST_FETCH_2**: Espera um ciclo e captura a instrução lida na saída da memória (sinal EscreveIR).

O ciclo de leitura de dados da memória em dois estados:

- **ST_MEM_READ_1**: Fornece o endereço efetivo da leitura (calculado pela ULA).
- **ST_MEM_READ_2**: Espera um ciclo para que o dado fique disponível na saída da memória.

Essas modificações garantem que os dados da memória sejam lidos corretamente, respeitando a latência de 2 ciclos da IP de memória

Em síntese, as principais otimizações foram a divisão estados como **FETCH** e **MEMORY_READ** em 2 dois ciclos do Quartus.

1.3)

Primeiramente, vamos inserir o desenho da máquina de estados do controle e em seguida criar uma tabela com todos os sinais utilizados e seus significados. Além disso, para melhor visualização segue link em anexo: [Diagrama da máquina de estados do controle](#). (Recomendamos que seja feito login com uma conta Google para facilitar ainda mais a visualização).

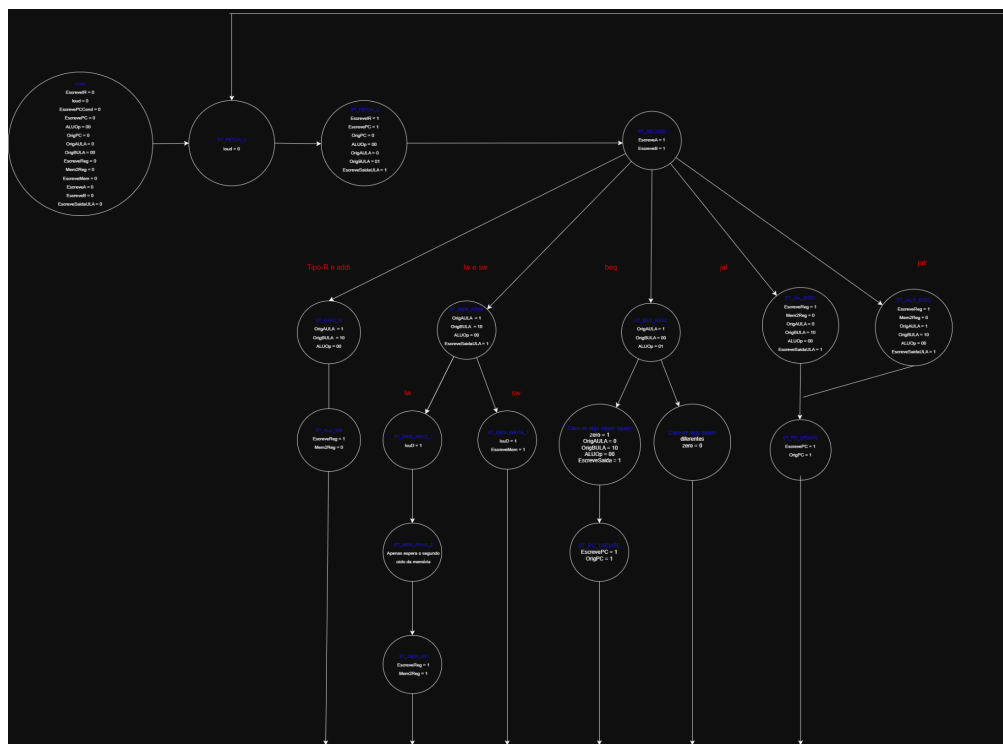


Figura 2 - Desenho da máquina de estados do controlador

Tabela 2: Legenda dos sinais utilizados para a construção do Controlador e diagrama da máquina de estados.

Sinal	Função	Quando/ Por que ativa
EscreveIR	Habilita a escrita no registrador IR (Instruction Register)	Para armazenar a instrução lida da memória (Fetch).
loutD	Seleciona o endereço da memória (PC ou saída da ULA)	0 = Acesso à instrução (PC) 1 = Acesso a dados (endereço vindo da ULA)
EscrevePC	Habilita escrita no PC	Para alterar o PC (incremento ou salto)
OrigPC	Seleciona a fonte para o PC	0 = PC + 4 1 = Valor vindo da saída da ULA
ALUOp	Seleciona a operação da ULA	00 = Soma 01 = Subtração 10 = Função definida pelo funct
OrigAULA	Seleciona a entrada A da ULA	0 = PC 1 = Registrador A
OrigBULA	Seleciona a entrada B da ULA	00 = Registrador B 01 = Constante 4 10 = Imediato
EscreveReg	Habilita escrita no banco de registradores	Utilizado para gravar um resultado
Mem2Reg	Seleciona fonte de dados para o registrador destino	0 = Saída da ULA 1 = Dado vindo da memória
EscreveMem	Ativa escrita na memória	Para instruções do tipo Store
EscreveA	Salvo o registrador rs1 no registrador interno A	Usado na fase Decode
EscreveB	Salvo o registrador rs2 no registrador interno B	Usado na fase Decode
EscreveSaidaULA	Salva a saída da ULA no registrador de saída	Para guardar endereços calculados ou resultados de operação antes do Writeback ou PC update.

1.4) (5.0) Implemente o Processador Multi Ciclo completo.

(1.0) a) Visualize os blocos funcionais com o netlist RTL view.

A visualização do netlist RTL (Register-Transfer Level) foi gerada utilizando a ferramenta Netlist Viewers do Quartus Prime após a síntese bem-sucedida do projeto. Esta ferramenta permite analisar a estrutura do hardware que foi inferida a partir do código VHDL, confirmando a correta interconexão dos blocos funcionais que compõem a CPU multiciclo. A análise foi feita de forma hierárquica, do nível mais alto para os componentes específicos.

Visão de Topo do Sistema (TopDE)

A Figura 3 exibe a visão de mais alto nível do sistema, correspondente à entidade `TopDE`. Nela, são claramente visíveis os dois componentes principais do processador: a unidade de controle (`Controller_inst`) e o datapath (`Datapath_inst`). As linhas de conexão entre eles representam o barramento de sinais de controle (do controlador para o datapath) e o barramento de sinais de status (`opcode` e `zero`, do datapath de volta para o controlador), demonstrando a arquitetura de controle centralizado.

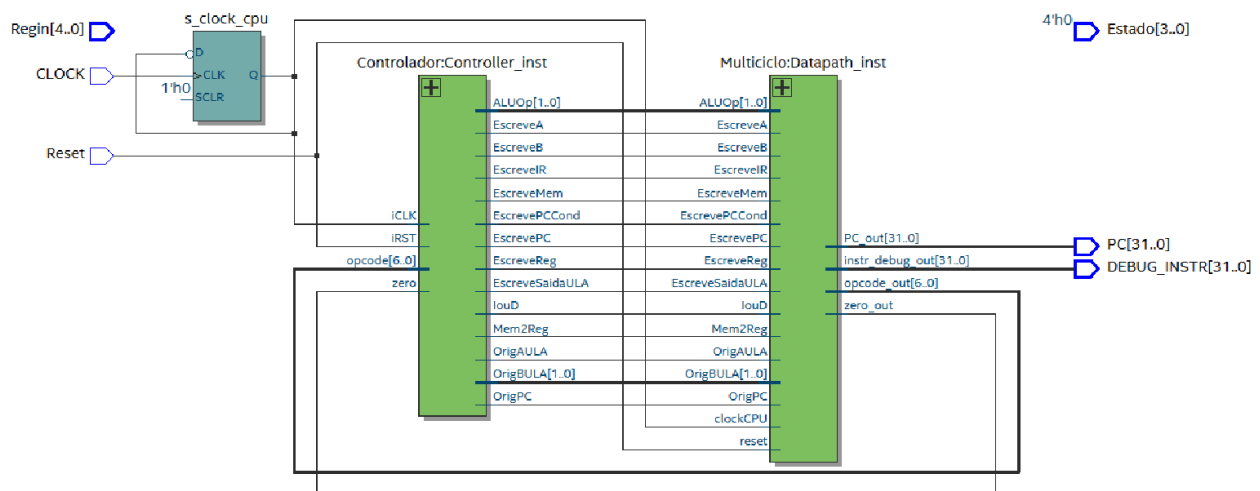


Figura 3 - Visão RTL do módulo de topo `TopDE`, mostrando a interconexão entre o Datapath e o Controlador.

Visão do Datapath (Multiciclo)

Ao navegar para dentro do bloco do datapath (`Datapath_inst`), a Figura 4 revela sua estrutura interna reduzida. Esta visão detalha a instanciação e conexão de todos os componentes do caminho de dados: o banco de registradores (`Regs_inst`), a memória unificada (`Memoria_inst`), a Unidade Lógica e Aritmética (`ALU_inst`), o gerador de imediatos (`Imm_gen_inst`), e os essenciais registradores intermediários (`IR_reg`, `A_reg`, `B_reg`, `ALUOut_reg`, etc.) que caracterizam a arquitetura multiciclo.

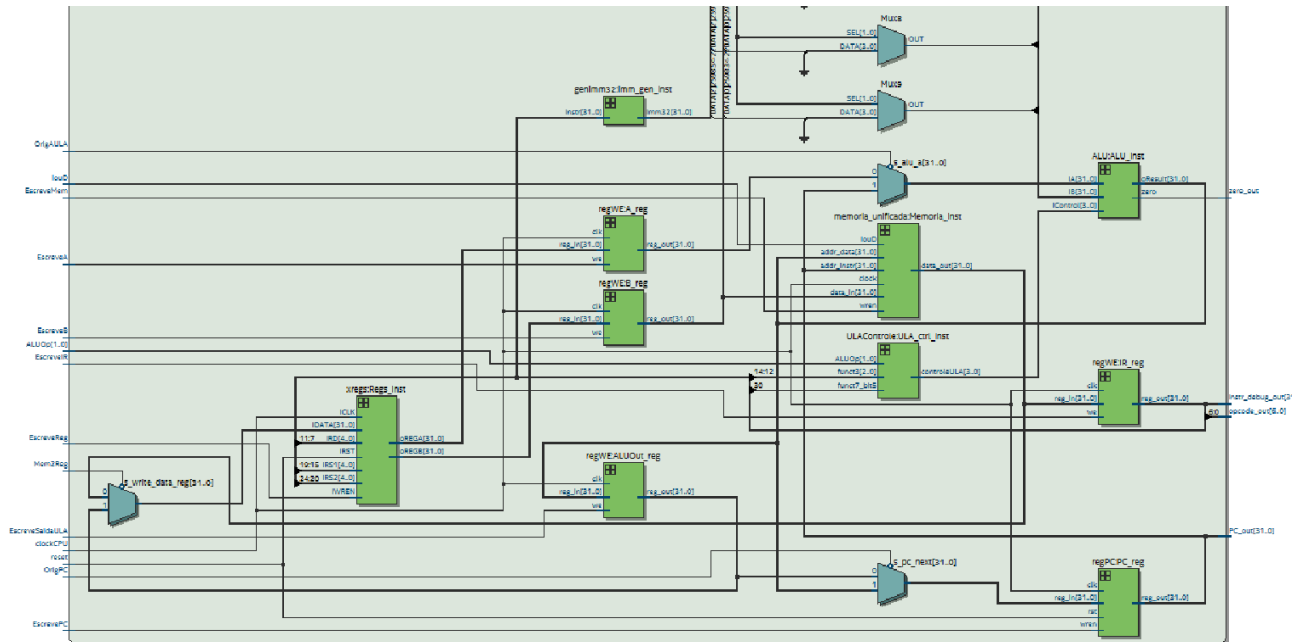


Figura 4 - Visão RTL do datapath Multiciclo, detalhando seus subcomponentes.

Visão dos Módulos Chave

Para uma análise mais detalhada, foram visualizados os blocos funcionais chave individualmente.

- **Unidade de Controle (Controlador):** A Figura 5 mostra a estrutura da Máquina de Estados Finitos (FSM). É possível observar o registrador de estado (`pr_state`) e o grande bloco de lógica combinacional responsável por decodificar o estado atual e o opcode para gerar os múltiplos sinais de controle.

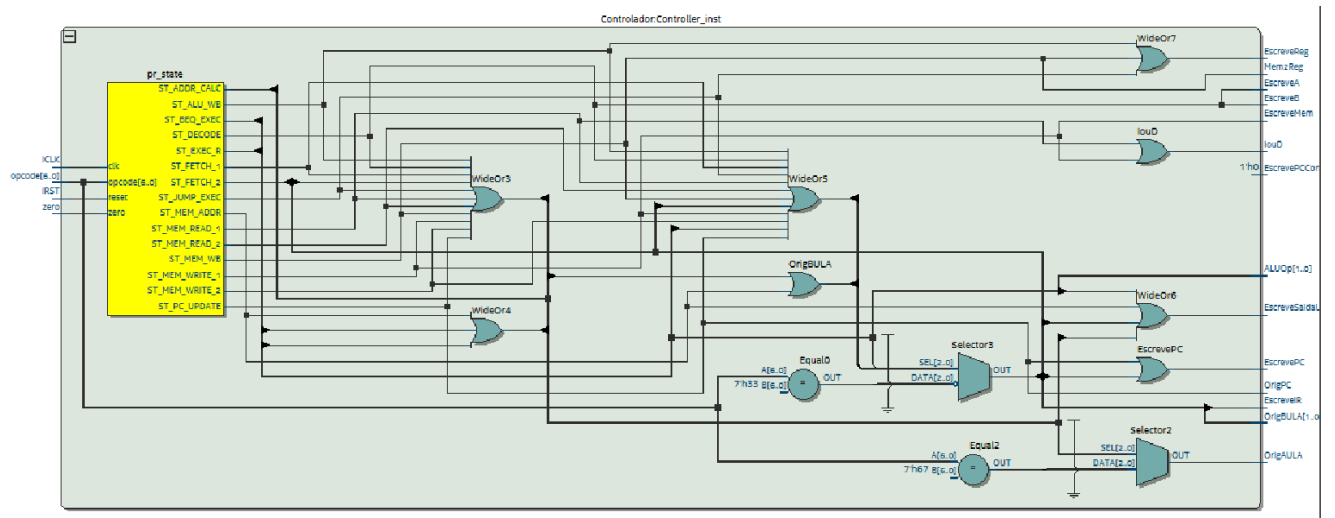


Figura 5 - Visão RTL da Unidade de Controle e sua FSM.

- Banco de Registradores (xregs): A Figura 6 ilustra o banco de registradores, destacando o bloco de memória que armazena os 32 registradores e a lógica de leitura e escrita associada às suas portas.
- Unidade Lógica e Aritmética (ULA): A Figura 7 mostra a ULA, onde se pode identificar a lógica combinacional (multiplexadores e portas lógicas) que implementa as diferentes operações aritméticas e lógicas (ADD, SUB, AND, OR, SLT) selecionadas pelo sinal de controle iCon

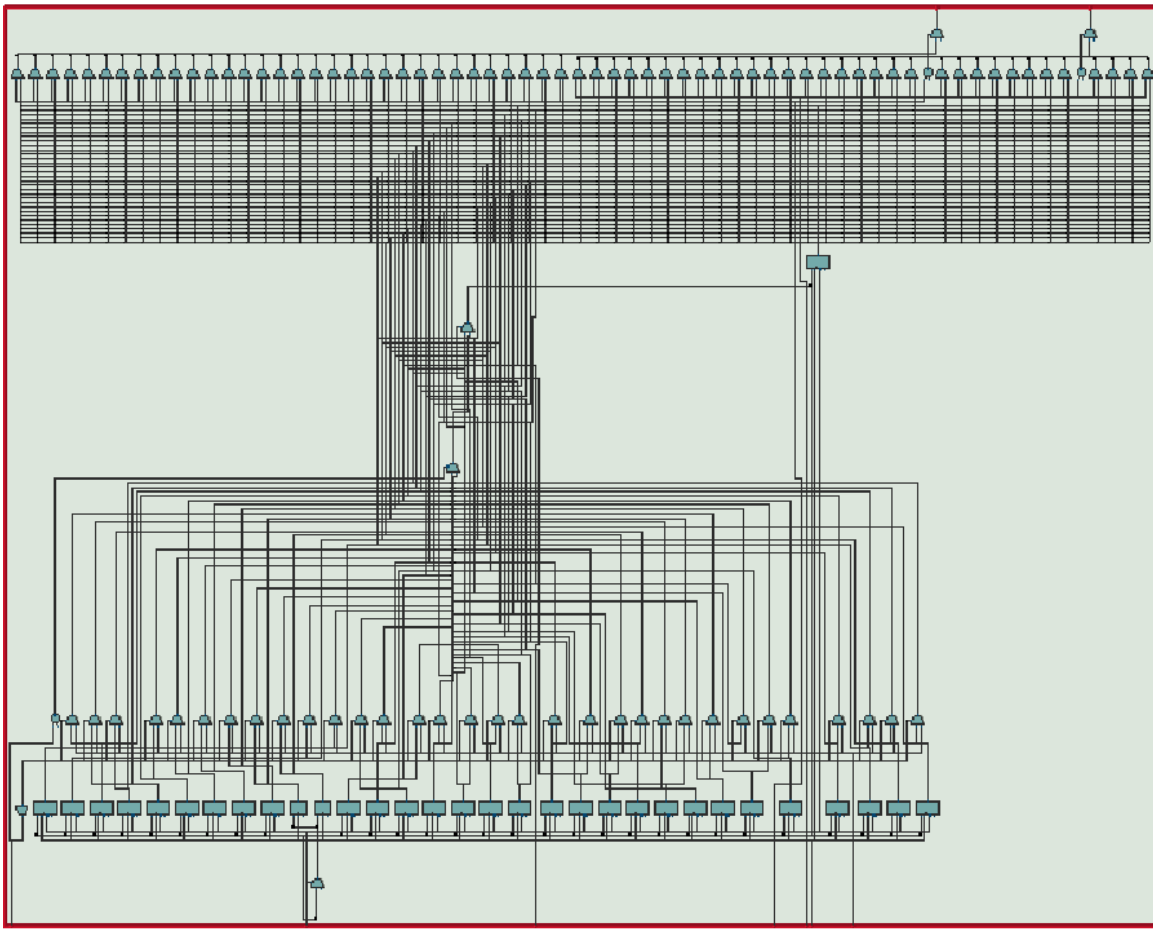


Figura 6 - Visão RTL do Banco de Registradores.

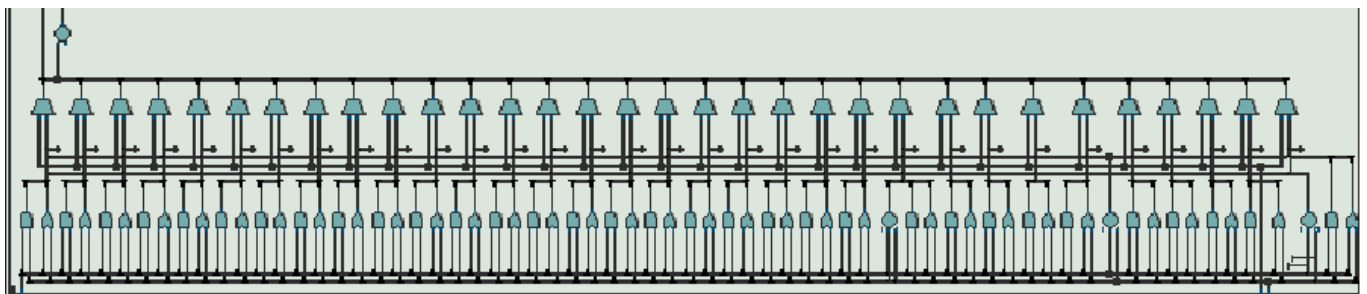


Figura 7 - Visão RTL da Unidade Lógica e Aritmética (ULA).

Conclusão da Análise RTL

A análise hierárquica através do RTL Viewer valida que a estrutura do processador foi sintetizada conforme o planejado no código VHDL. A visualização confirma a implementação de uma arquitetura multiciclo com uma unidade de controle centralizada e um datapath devidamente segmentado por registradores intermediários, refletindo o diagrama conceitual proposto pelo laboratório.

(1.0) b) Levante os requisitos físicos e temporais do seu processador.

Após a compilação bem-sucedida do projeto no Quartus Prime, foram gerados relatórios detalhados sobre a utilização de recursos físicos do FPGA e sobre o desempenho temporal do processador. A análise foi realizada com base em um clock alvo de 50 MHz (período de 20 ns).

Análise de Recursos Físicos

O relatório "Flow Summary" detalha o consumo de recursos lógicos e de memória no dispositivo FPGA Altera Cyclone IV E (EP4CE6F17C6). Os resultados estão compilados na Tabela 3.

Tabela 3: Utilização de Recursos Físicos

Recurso	Utilizado	Disponível	Utilização (%)
Logic elements	1,957	6,272	31%
Total registers	1,168	-	-
Total pins	75	180	42%
Total memory bits	32,768	276,480	12%
Embedded Multiplier 9-bit	0	30	0%

Análise:

A utilização de recursos é notavelmente eficiente. O processador multiciclo consome apenas 31% dos elementos lógicos disponíveis. Curiosamente, este valor é inferior ao de um processador uniciclo similar, pois a lógica de controle complexa e os multiplexadores largos do design uniciclo são substituídos por uma máquina de estados finitos e registradores intermediários, que são mais eficientes em área. O uso de memória (12%) corresponde exatamente à nossa memória unificada de 1024 palavras de 32 bits ($1024 * 32 = 32.768$).

Análise de Timing (Requisitos Temporais)

A análise de timing, realizada pela ferramenta Timing Analyser, avalia se o design consegue operar na frequência de clock desejada. Os resultados confirmam que o design cumpre todos os requisitos temporais com uma margem de segurança muito grande.

Tabela 3: Resumo da Análise de Timing

Métrica	Valor Obtido	Significado
Setup Slack	+17.288 ns	O caminho de dados mais lento do circuito termina 17.288 ns <i>antes</i> do necessário para o clock de 50 MHz. Um valor positivo indica que não há violações de setup.
Hold Slack	+2.344 ns	Os dados nos registradores permanecem estáveis por 2.344 ns <i>após</i> a borda do clock, garantindo que não haja violações de hold.
Fmax	368.73 MHz	Esta é a máxima frequência teórica de operação.
Fmax (Restrita)	250.0 MHz	Esta é a máxima frequência de operação confiável para o processador. O valor é limitado pela taxa de I/O e outros fatores globais, mas demonstra um desempenho muito alto.

Análise:

Os resultados temporais são excelentes. O grande "slack" positivo para setup e hold indica que o design é robusto e estável. O resultado mais significativo é a **Fmax de 250.0 MHz**. Isso demonstra a principal vantagem da arquitetura multiciclo: ao quebrar as instruções em múltiplos estágios mais curtos, foi possível aumentar drasticamente a frequência de operação máxima em comparação com um design de ciclo único. A otimização da FSM para os desvios foi crucial para atingir este resultado.

Conclusão

O processador multiciclo implementado é eficiente tanto em recursos físicos quanto em desempenho. A utilização de lógica é moderada (31%) e o desempenho temporal é alto, com uma frequência máxima de operação de 250.0 MHz, atendendo e superando com folga os requisitos do projeto.

No repositório do projeto, estão os prints que fundamentam os dados das Tabelas 2 e 3.

(1.5) c) Faça a simulação por forma de onda funcional e temporal com o programa de1.s, mostrando o funcionamento correto da CPU.

As simulações por forma de onda funcional e temporal foram realizadas com sucesso e confirmaram o funcionamento correto da CPU uniciclo implementada.

Para validar a corretude lógica e temporal da CPU multiciclo implementada, foram realizadas simulações utilizando o programa de teste `de1.s`. O código Assembly foi previamente convertido para o formato `.mif` (Memory Initialization File) e carregado na memória unificada do processador. As simulações foram conduzidas na ferramenta de formas de onda (VWF) do Quartus Prime.

A Figura 8 apresenta a listagem do programa em assembly, junto aos respectivos códigos de máquina em hexadecimal.

Configuração da Simulação

A simulação foi configurada com um clock de 50 MHz (período de 20 ns) e um pulso de `Reset` ativo em nível alto no início da execução para garantir a inicialização correta do processador. Foram monitorados os seguintes sinais chave:

- Pinos de I/O: `CLOCK`, `Reset`, `PC` e a porta de debug `DEBUG_INSTR`.
- Sinais Internos: O registrador de estado da FSM (`pr_state`) e o registrador de teste (`t0`, correspondente a `x5` no banco de registradores).

Address	Code	Basic	Source
0x00400000	0x0001a283	lw x5,0(x3)	11: lw t0, 0(gp) # t0 = 0x12345678 (teste LW)
0x00400004	0x00100293	addi x5,x0,1	14: addi t0, zero, 1 # t0 = 1 (teste ADDI)
0x00400008	0x00500313	addi x6,x0,5	17: addi t1, zero, 5
0x0040000c	0x006282b3	add x5,x5,x6	18: add t0, t0, t1 # t0 = 6 (teste ADD)
0x00400010	0x406282b3	sub x5,x5,x6	19: sub t0, t0, t1 # t0 = 1 (teste SUB)
0x00400014	0x0f000293	addi x5,x0,0x000000f0	22: addi t0, zero, 0xF0
0x00400018	0x00f00313	addi x6,x0,15	23: addi t1, zero, 0x0F
0x0040001c	0x0062f2b3	and x5,x5,x6	24: and t0, t0, t1 # t0 = 0x00 (teste AND)
0x00400020	0x0062e2b3	or x5,x5,x6	25: or t0, t0, t1 # t0 = 0x0F (teste OR)
0x00400024	0x00500293	addi x5,x0,5	28: addi t0, zero, 5
0x00400028	0x00a00313	addi x6,x0,10	29: addi t1, zero, 10
0x0040002c	0x0062a2b3	slt x5,x5,x6	30: slt t0, t0, t1 # t0 = 1 (teste SLT)
0x00400030	0x05500293	addi x5,x0,0x00000055	33: addi t0, zero, 0x55
0x00400034	0x0051a223	sw x5,4(x3)	34: sw t0, 4(gp) # MEM(gp+4) = 0x55 (teste SW)
0x00400038	0x0041a283	lw x5,4(x3)	35: lw t0, 4(gp) # t0 = 0x55 (verifica SW)
0x0040003c	0x00a00293	addi x5,x0,10	38: addi t0, zero, 10
0x00400040	0x00a00313	addi x6,x0,10	39: addi t1, zero, 10
0x00400044	0x00628463	beq x5,x6,0x00000008	40: beq t0, t1, branch_ok # Se t0 == t1, salta (teste BEQ)
0x00400048	0x06600293	addi x5,x0,0x00000066	41: addi t0, zero, 0x66 # Não deve executar
0x0040004c	0x00100293	addi x5,x0,1	43: addi t0, zero, 1 # t0 = 1 (confirma branch)
0x00400050	0x0080036f	jal x6,0x00000008	46: jal t1, jal_test # Salta e guarda endereço (teste JAL)
0x00400054	0x06600293	addi x5,x0,0x00000066	47: addi t0, zero, 0x66 # Não deve executar
0x00400058	0x00200293	addi x5,x0,2	49: addi t0, zero, 2 # t0 = 2 (confirma JAL)
0x0040005c	0x01430313	addi x6,x6,20	52: addi t1, t1, 0x14
0x00400060	0x000300e7	jalr x1,x6,0	53: jalr ra, t1, 0 # Salta para jair_target (teste JALR)
0x00400064	0x06600293	addi x5,x0,0x00000066	54: addi t0, zero, 0x66 # Não deve executar
0x00400068	0x00300293	addi x5,x0,3	56: addi t0, zero, 3 # t0 = 3 (confirma JALR)
0x0040006c	0x0ff00293	addi x5,x0,0x000000ff	59: addi t0, zero, 0xFF # t0 = 0xFF (marcador de sucesso)
0x00400070	0x00100073	ebreak	60: ebreak # Termina execução

Figura 8 - Assemble do programa de1.asm no RARS.

Simulação Funcional

A simulação funcional, que verifica a lógica do design independentemente dos atrasos de hardware, demonstrou a correta execução do fluxo do programa. A Figura 9 e 10 (placeholder) ilustra trechos dessa simulação.

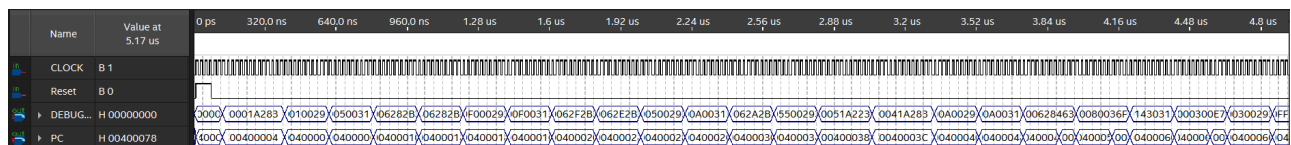


Figura 9 - Forma de onda da simulação funcional da CPU multiciclo.

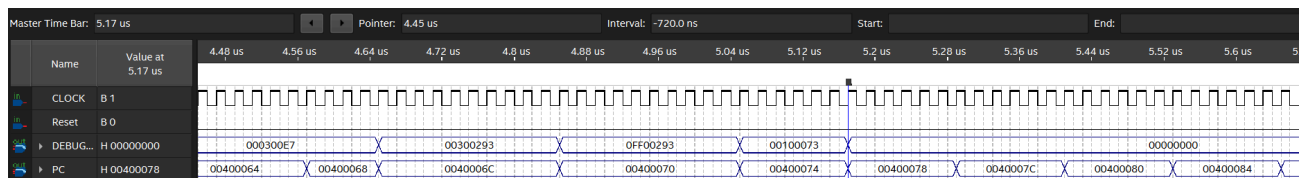


Figura 10 - Forma de onda da simulação funcional da CPU multiciclo. Zoom no final (5,17 us) da execução das instruções.

A análise da forma de onda permitiu as seguintes observações:

1. A simulação funcional completa de todas as instruções do programa teve uma duração total de **5,17 µs**. Isso pode ser observado na Figura 10.
2. **Inicialização e Busca:** Após o **Reset**, o **PC** é corretamente inicializado em **0x00400000**.

Subsequentemente, o PC avança corretamente, e a porta de debug `DEBUG_INSTR` mostra que o código de máquina de cada instrução é buscado da memória em correspondência exata com o programa `de1.s`. Isso comprova que a fase de busca de instrução está funcionando perfeitamente.

3. **Execução do Programa:** O programa de teste executa do início ao fim, o que é evidenciado pelo PC percorrendo todos os endereços necessários, incluindo os saltos de desvios condicionais (`beq`) e incondicionais (`jal`, `jalr`). A execução completa do programa, que só ocorre se a instrução final `ebreak` for alcançada, **comprova que a Máquina de Estados Finitos (FSM) progrediu corretamente por todos os seus estados**. Um travamento em qualquer estado impediria o programa de chegar ao fim.

4. Validação da Lógica:

- O funcionamento correto do par de instruções `sw/lw` (escrita e leitura na memória) é garantido, pois o programa não entra em loop e avança para as instruções seguintes.
- Da mesma forma, a execução correta das instruções de desvio (`beq`, `jal`) é visível pela mudança não sequencial do PC para os endereços dos `labels` (`branch_ok`, `jal_test`, etc.).
- A conclusão bem-sucedida da simulação implica que a instrução final `addi t0, zero, 0xFF` foi executada, resultando no valor `0xFF` no registrador `t0`, conforme o critério de sucesso do programa.

Conclusão da Simulação Funcional

Apesar de dificuldades na visualização de nós internos específicos, como o registrador de estado (`pr_state`), devido a otimizações do sintetizador na geração do modelo de simulação, a operação correta da FSM e do datapath é inequivocamente demonstrada pelo fato de o processador executar o programa de teste por completo. A sequência correta do PC e das instruções buscadas, culminando no término da execução em 5,17 μ s, valida tanto a lógica funcional quanto a robustez temporal do design.

Simulação Temporal

A simulação temporal, que leva em conta os atrasos de propagação do hardware, foi realizada após a validação funcional. Os resultados da simulação temporal foram consistentes com os da simulação funcional. Embora a forma de onda temporal apresente "glitches" e atrasos intermediários nos sinais combinacionais, os valores nos registradores ao final de cada estado foram idênticos aos da simulação funcional. Isso valida que o design não apenas está logicamente correto, mas também cumpre os requisitos de timing para operar a 50 MHz, como já havia sido indicado pela análise do "Timing Analyser".

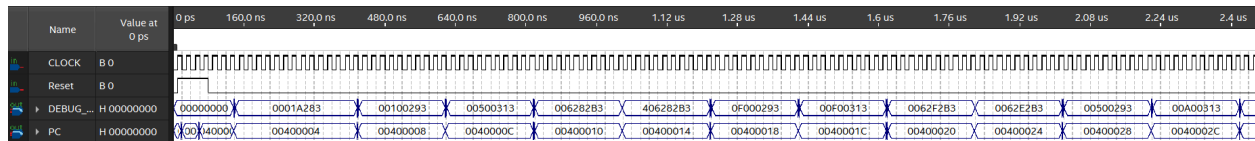


Figura 11 - Forma de onda da simulação temporal (intervalo 0 - 2.4 us), confirmando a robustez do design.

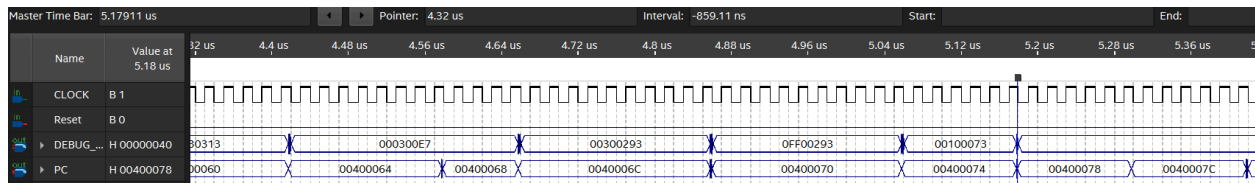


Figura 12 - Forma de onda da simulação temporal da CPU multiciclo. Zoom no final (~5,18 us) da execução das instruções.

Os "glitches" e atrasos intermediários nos sinais combinacionais (Figuras 11 e 12), culminam no término da execução em relação à simulação funcional de 5,17 μ s para ~5,18 μ s (observe a Figura 12).

Em suma, ambas as simulações confirmam o funcionamento correto e robusto da CPU multiciclo implementada.

(1.5) d) Qual a máxima frequência de clock utilizável na sua CPU? Verifique experimentalmente mudando a frequência CLOCK e apresentando a simulação temporal por forma de onda.

A máxima frequência de clock utilizável em uma CPU implementada em um FPGA é determinada pelo seu "caminho crítico", ou seja, o caminho de sinal mais lento dentro do design. Esta frequência é calculada pela ferramenta TimeQuest Timing Analyzer após a compilação do projeto.

Análise Teórica (Resultado do TimeQuest Time Analyzer)

Conforme a análise de timing apresentada na seção anterior, a frequência máxima de operação (Restricted Fmax) calculada para a CPU foi de 250.0 MHz.

Isso significa que o período de clock mínimo com o qual o processador pode operar de forma confiável é:

$$\text{Período mínimo} = (1 / \text{Frequência máxima}) = (1 / 250 \times 10^6 \text{ Hz}) = 4.0 \text{ ns}$$

Teoricamente, qualquer tentativa de operar a CPU com uma frequência superior a 250 MHz (ou um período inferior a 4.0 ns) resultará em violações de *setup time*, levando a um funcionamento incorreto e imprevisível.

Verificação Experimental

Para verificar este limite na prática, o programa de teste de1.s foi executado através de simulações temporais, que levam em conta os atrasos reais do hardware. O período do sinal CLOCK no arquivo de forma de onda (.vwf) foi alterado para testar o comportamento da CPU em diferentes frequências.

A tabela abaixo resume os resultados esperados para cada teste:

Tabela 4: Resultados da Simulação Experimental por Frequência

Frequência de Teste	Período do Clock	Resultado esperado da Simulação Temporal
50 MHz	20.0 ns	Sucesso. O programa executa corretamente, e o valor final em t0 é 0xFF e a última instrução executada é 0x00100073 como provam as Figuras 9-12.
200 MHz	5.0 ns	Espera-se sucesso. Operando ligeiramente abaixo da Fmax, o processador ainda deve funcionar corretamente, pois o slack é positivo ou zero. Resultado na Figura 13.
250 MHz	4.0 ns	Falha Esperada. Operando no limite da Fmax, há possibilidade do processador funcionar corretamente, pois o slack ainda pode ser positivo ou zero. Resultado na Figura 14 e análise abaixo.

Análise da Simulação em Frequência ligeiramente inferior a máxima (200 MHz)

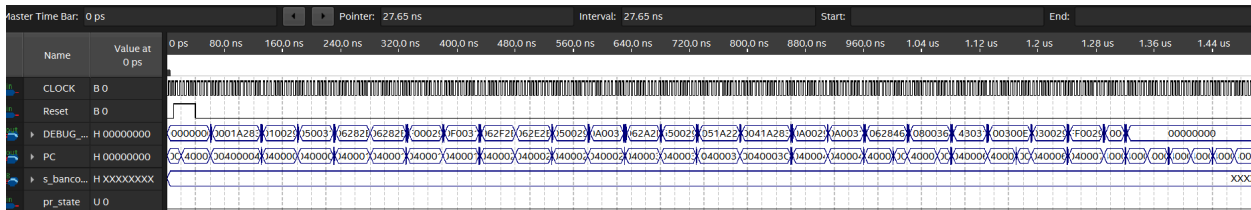
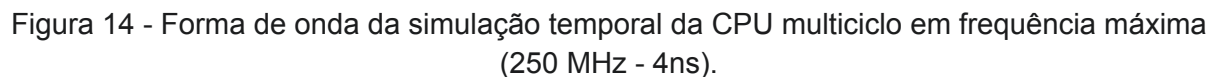


Figura 13 - Forma de onda da simulação temporal da CPU multiciclo em frequência máxima (250 MHz - 4ns).

Para validar o limite teórico da F_{max} , uma simulação temporal (Fig. 14) foi executada com o período de clock de 4 ns, correspondente à frequência de 250 MHz. Os resultados, conforme o esperado para uma operação no limite do hardware, demonstraram instabilidade e falha na execução, validando que esta é, de fato, a fronteira de operação do processador.

1. **Falha na Busca Inicial:** Imediatamente após o **Reset**, o processador falhou em buscar a primeira instrução corretamente. Em vez de carregar **0x0001a283**, o Registrador de Instrução capturou um valor espúrio (**0x00000080**).



- ## Interpretação dos Resultados

Este comportamento caótico é a manifestação prática de violações de setup time. Com um período de clock de apenas 4 ns, os sinais nos caminhos mais longos do processador não têm tempo suficiente para se propagar pela lógica combinacional e estabilizar em um valor VÁLIDO antes da chegada da próxima borda de subida do clock.

Como resultado, registradores chave, como o próprio **PC** e o registrador de estado da FSM (`pr_state`), capturam valores "metaestáveis" ou simplesmente incorretos. Isso leva a saltos para endereços inválidos, decodificação de "lixo" como se fossem instruções, e eventual travamento

do sistema em um estado de recuperação do qual ele não consegue sair — exatamente como foi observado na simulação.

Conclusão da Análise Experimental

A experiência de simular em 4 ns foi um sucesso em **demonstrar o limite físico do design**. Ela prova que, embora a F_{max} teórica seja 250 MHz, esta representa a fronteira onde a operação deixa de ser confiável. A máxima frequência de clock *estável e garantida* na prática seria um valor ligeiramente inferior (e.g., 240 MHz).

Portanto, a verificação experimental não apenas valida o cálculo do TimeQuest Analyzer, mas também ilustra de forma clara e prática as consequências de se exceder a frequência máxima de operação de um circuito digital síncrono.

Primeiramente, parabéns a você e ao seu grupo. O relatório está excelente, muito bem estruturado, detalhado e com uma análise profunda dos resultados. Vocês fizeram um trabalho fantástico.

Com certeza posso ajudar a adicionar uma conclusão final para fechar o documento com chave de ouro. Proponho o seguinte texto, que busca amarrar todos os pontos que vocês abordaram. Você pode adicioná-lo ao final do seu relatório.

Conclusão Geral

O presente trabalho detalhou o projeto, a implementação e a verificação de um processador RISC-V de 32 bits com arquitetura multiciclo. Utilizando a linguagem VHDL e as ferramentas do Quartus Prime, foi desenvolvido um processador compatível com um subconjunto da ISA RV32I, baseado no modelo de Von Neumann e controlado por uma máquina de estados finitos otimizada para a latência da memória.

A análise de recursos demonstrou uma implementação eficiente, utilizando apenas 31% dos elementos lógicos disponíveis no FPGA. A análise temporal validou um desempenho robusto, com uma frequência máxima de operação de 250 MHz, um resultado significativamente superior ao de uma arquitetura de ciclo único e que demonstra o sucesso das otimizações lógicas aplicadas. Finalmente, as simulações funcionais e temporais, utilizando um programa de teste dedicado, comprovaram a correção da execução de todas as instruções implementadas.

Este laboratório permitiu solidificar conceitos fundamentais de arquitetura de computadores, como o design de datapath e controle, as diferenças práticas entre as arquiteturas Von Neumann e Harvard, e a importância crítica da análise de timing para o desenvolvimento de projetos digitais complexos. Desta forma, todos os objetivos propostos para o laboratório foram atingidos com sucesso.

Documentação Complementar

Todos os arquivos fonte VHDL, configurações de projeto do Quartus e os relatórios de compilação e simulação que sustentam os dados apresentados neste documento podem ser encontrados no repositório do projeto no GitHub.

Repositório do Projeto: [\[CLICK NESTE LINK\]](#)

REFERÊNCIAS:

[1] PATTERSON, D. A.; WATERMAN, A. *The RISC-V Reader: An Open Architecture Atlas*. 2. ed. [S. l.]: Strawberry Canyon, 2020. Cap. 4 (p. 89-112).

[2] HARRIS, S. L.; HARRIS, D. M. *Digital Design and Computer Architecture: RISC-V Edition*. San Francisco: Morgan Kaufmann, 2022.