Universidade de Brasília
Departamento de Ciência da Computação

Laboratório 2

- CPU RISC-V UNICICLO -

Alunos:

Dyesi Montagner - Matrícula: 212008544
Gabriel Castro - Matrícula: 202066571
Lucas Santana - Matrícula: 211028097
Maria Vitória Monteiro - Matrícula: 222035652
Jackson Marques - Matrícula: 170013367

Grupo B3 – OAC Unificado – 2025/1
Link do repositório GitHub: Repositório Grupo B3
Playlist no youtube com os testes em vídeo: Playlist Grupo B3

1) (10.0) Implemente o processador Uniciclo com ISA Reduzida com as instruções: add, sub, and, or, slt, lw, sw, beq, jal, e ainda as instruções jalr e addi.

1.1) (1.0) Escreva um programa de1.s que teste a corretude da implementação de todas as 9 + 2 instruções e teste no Rars. Dica: Use o registrador t0 para visualizar resultados!

Tabela 1: Tabela de Testes.

Instrução	Código	Entrada	Saída (t0)	Endereço Memória (gp)
LW	`lw t0, 0(gp)`	0x12345678	0x12345678	0x10010000
ADDI	`addi t0, zero, 1`	-	1	-
ADD	`add t0, t0, t1`	t0=1, t1=5	6	-
SUB	`sub t0, t0, t1`	t0=6, t1=5	1	-
AND	`and t0, t0, t1`	t0=0xF0, t1=0x0F	0x00	-
OR	`or t0, t0, t1`	t0=0x00, t1=0x0F	0x0F	-
SLT	`slt t0, t0, t1`	t0=5, t1=10	1	-
SW	`sw t0, 4(gp)`	t0=0x55	-	0x10010004 → 0x55

BEQ	`beq t0, t1, branch`	t0=10, t1=10	1 (confirma)	-
JAL	`jal ra, jal_test`	-	2 (confirma)	-
JALR	`jalr ra, t1, 0`	-	3 (confirma)	-

Clique para acessar o arquivo **de1.s** diretamente no repositório. A partir do qual foi escrito a Tabela 1 e as análises abaixo.

- **Banco de Registradores**: 32 registradores de 32 bits, com `gp` inicializado em `0x10010000`.

- **Memória**: Harvard (dados e instruções separados), com:

  - `.text` (instruções): Começa em `0x00400000` (padrão do RARS para código).

  - `.data` (dados): Começa em `0x10010000` (exigido pelo processador físico).

- **Instruções Testadas**: Todas as 11 exigidas, apresentadas na Tab. 1, com foco em `t0` para visualização.

- **Endereçamento explícito:** `0x10010000` força o dado para um endereço específico (exigido pelo processador físico). O teste foi robustecido: `0x12345678` testa melhor operações com bytes altos/baixos. `1` é muito simples (pode mascarar erros).

No **RARS**, todos os testes foram confirmados pelos valores em `t0`.

Se `t0` for `0x66` em qualquer momento, indica que um branch/jump falhou (valor de erro).
Se `t0` for `0xFF` no final sinaliza que todos os testes passam.

O procedimento no **RARS** é executar 'Step-by-Step' e monitorar' `t0` e `pc`.

1.2) (1.0) Implemente o Banco de Registradores com 3 leituras simultâneas: rs1, rs2 e disp. Stack Pointer (sp) inicial: 0x1001_03FC Global Pointer (gp) inicial: 0x1001_0000

Clique para acessar o arquivo **xreg.vhd** diretamente no repositório.
O banco de registradores possui 3 portas de leitura (rs1, rs2 e disp) e 1 porta de escrita, conforme exigido.

1.3) (1.0) Implemente o Gerador de Imediatos.

Clique para acessar o arquivo **genImm32.vhd** diretamente no repositório.
O Gerador de Imediatos extrai e estende imediatos de 32 bits para todos os formatos de instrução (I, S, B, U, J).

1.4) (0.5) No Rars16_Custom2, vá em File/Dump Memory e exporte (MIF 32 Format) para o arquivo de1 (sem extensão). Os arquivos de1_text.mif e de1_data.mif serão gerados. As Memórias de Instruções (1024 words) e de Dados (1024 words) já estão geradas, com conteúdo default os arquivos mif gerados. Dica: Como a memória do FPGA necessita 2 ciclos de clock para ler ou escrever um valor, a frequência de clock da CPU é a metade do clock da Memória. Endereço inicial do .text: 0x0040_0000 Endereço inicial do .data: 0x1001_0000

Clique para acessar os arquivos gerados **de1_data.mir** e **de1_text.mir** diretamente no repositório.

Tabela 2: Explicação dos arquivos .mif gerados.

Arquivo	Finalidade	Formato
`de1.s`	Contém o código assembly RISC-V que testa as 11 instruções do processador. É o programa que será executado na CPU.	Texto (assembly)
`de1_text.mif`	Armazena as instruções em binário para a memória de instruções do FPGA. Contém o código compilado de `de1.s`.	MIF (Memory Initialization File)
`de1_data.mif`	Armazena os dados inicializados (como `test_data: .word 0x12345678`) para a memória de dados do FPGA.	MIF

O `de1.s` é o código-fonte para testes, elaborado acima na questão 1.1.

Os `.mif` são carregados no FPGA para inicializar as memórias do processador físico.
Sem eles, a CPU não teria instruções nem dados para executar.

O registrador `t0` é usado como indicador visual do fluxo do programa. Assim como apresentado na tabela 1, que mostra como seu valor valida cada teste.

Se tudo funcionar, os registradores devem terminar com:
`t0 = 0xFF` (fim).
`t1 = 5`, `t2 = 3`, etc. (valores intermediários).
Memória em `0x10010004` deve conter 0x55 (teste do sw).

1.5) (0.5) Implemente a ULA mínima necessária (add, sub, and, or, slt, zero).

Clique para acessar diretamente no repositório o arquivo **alu.vhd**, que implementa a ULA mínima necessária (operações add, sub, and, or, slt e zero).

1.6) (1.0) Implemente o Controlador da ULA e o Bloco Controlador.

Clique para acessar diretamente no repositório:

- **Controlador da ULA** (alu_control.vhd).
  Controlador da ULA (Mapeia funct3 e funct7 para operações da ULA).

- **Bloco Controlador** (controle.vhd)**.**
  Bloco de controle principal (decodifica instruções e gera sinais como RegWrite, MemtoReg, Branch, etc.).

Tabela 3: Tabela resumo.

Questão	Arquivo
1.2	`xreg.vhd`
1.3	`genImm32.vhd`
1.5	`alu.vhd`
1.6	`controle.vhd` + `alu_control.vhd`

1.7) (5.0) Implemente o Processador Uniciclo completo.

(1.0) a) Visualize o netlist RTL view. Coloque print screens dos módulos no relatório.

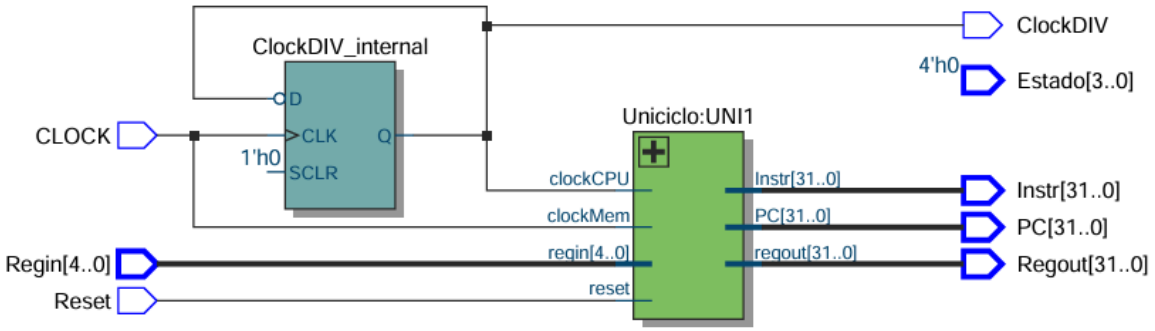

Figura 1 - Caminho de Dados (visão top level), arquivo TopDE.vhd.

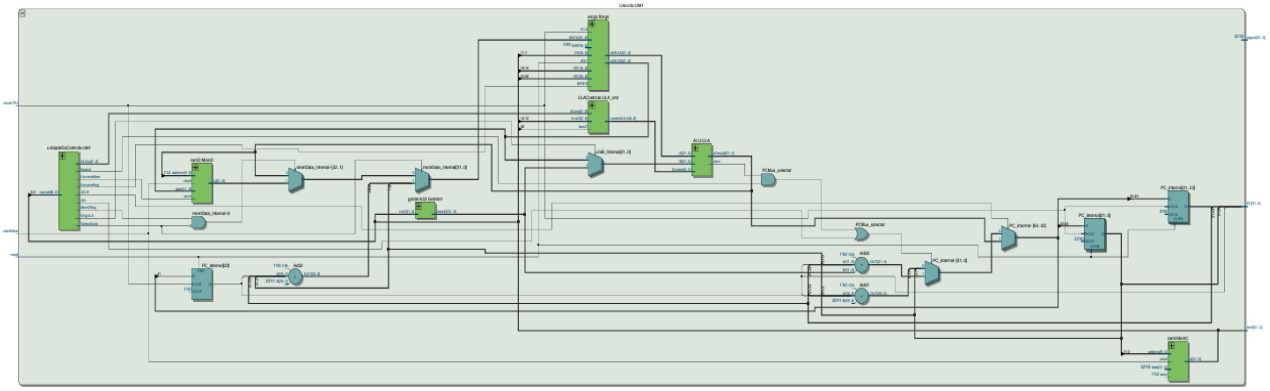

Figura 2 - Unidade Operativa, arquivo uniciclo.vhd.

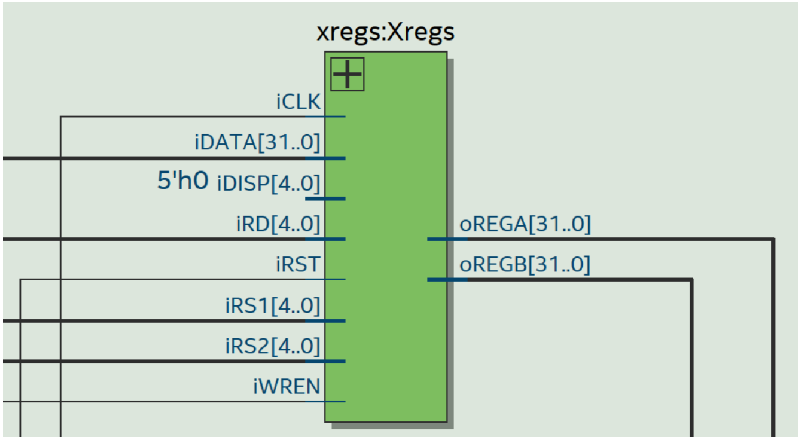

Figura 3 - Banco de Registradores, arquivo xreg.vhd.

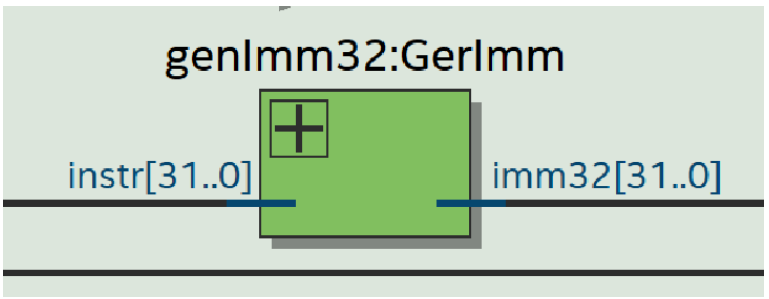

Figura 4 - Gerador de Imediatos,, arquivo ganImm32.vhd.

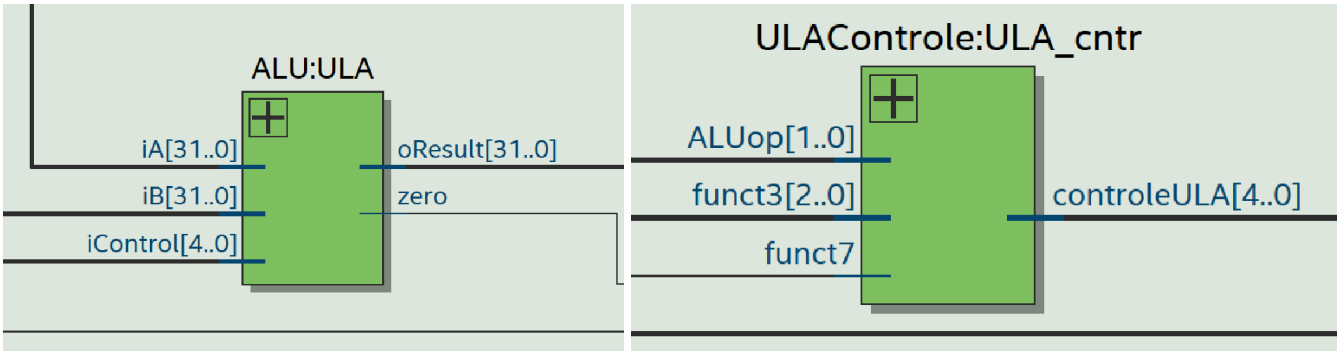

Figura 5 e 6 - ULA + Controle da ULA, alu.vhd + alu_control.vhd.

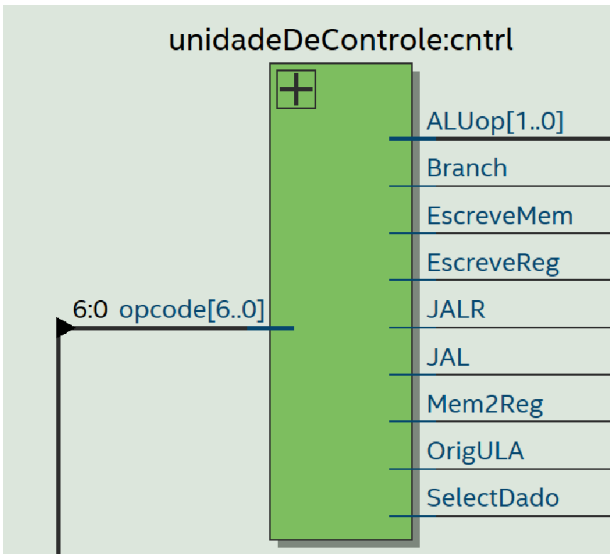

Figura 7 - Unidade de Controle, arquivo controle.vhd.

As figuras 1-7 são auto-explicativas. Em resumo:

A Fig. 1 mostra a visão Top Level do DataPath, ou caminho de dados. A Fig. 2 mostra conexões entre PC, memória, ULA e banco de registradores. Na Fig. 3 podemos ver as portas de leitura (rs1, rs2, disp) e escrita. E assim por diante.

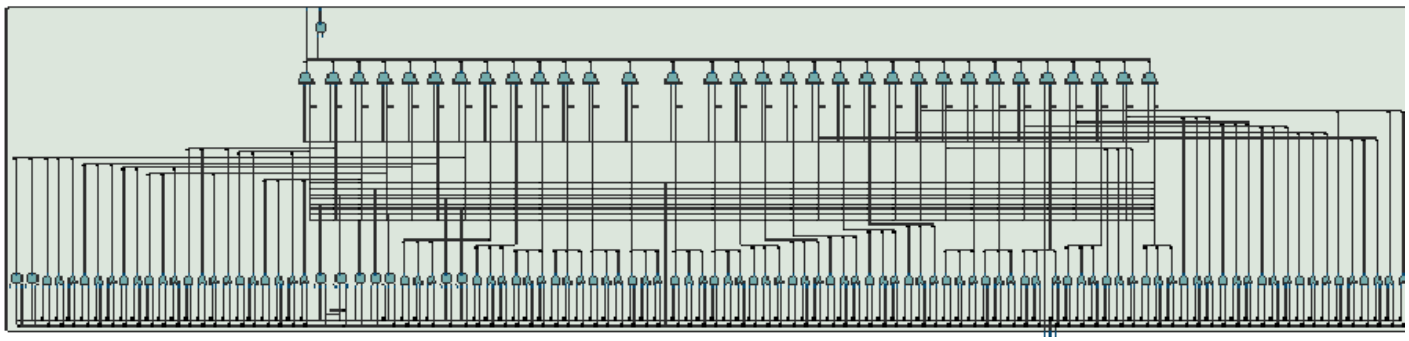

Figura 8 - Dentro do módulo da ULA, arquivo alu.vhd.

Na Figura 8, podemos visualizar por meio da netlist RTL view, a lógica da ULA. Nela, é possível observar a implementação das operações add, sub, and, etc. por meio das conexões entre as várias portas lógicas.

Outros exemplos a seguir (Figuras 9-11).

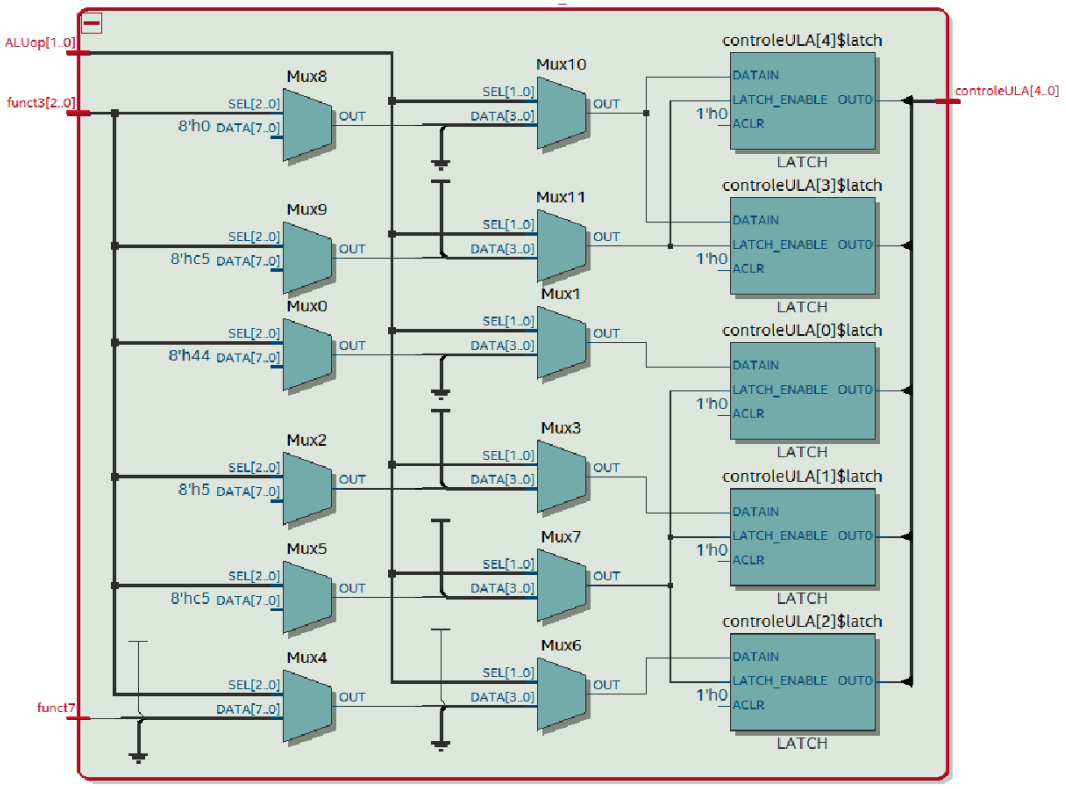

Figura 9 - Dentro do módulo Controle da ULA, arquivo alu_control.vhd.

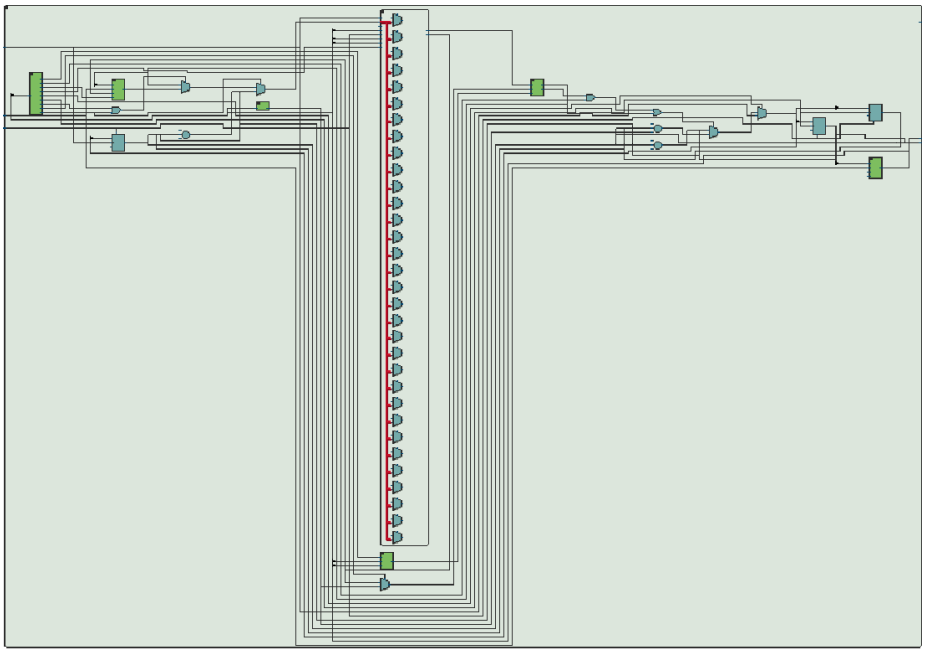

Figura 10 - Dentro do módulo Banco de Registradores, arquivo xreg.vhd.

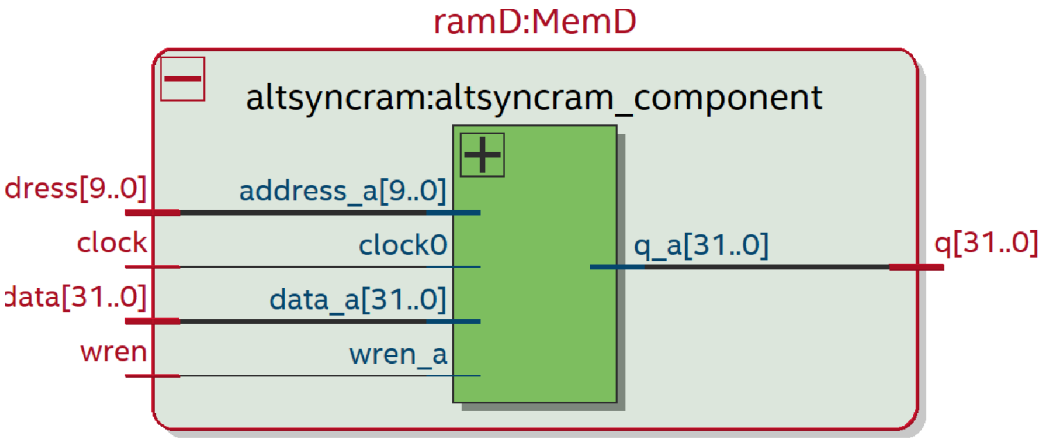

Figura 11 - Módulo ram.

(1.0) b) Levante os requisitos físicos e temporais do seu processador completo. Verifique se os slacks estão sendo cumpridos.

Para levantar os requisitos físicos e temporais do processador, foi definidos 20ns de clock.

Tabela 4: Análise de Recursos.

Recurso	**Utilizado**	**Total Disponível**	**% Utilização**
Logic Elements (LEs)	3.230	6.272	51%
Registradores	1.057	6.272	~17%
Pinos I/O	108	180	60%
Memória (bits)	65.536	276.480	24%
Multiplicadores 9-bit	0	30	0%

Segundo Patterson e Waterman [1], em CPUs RISC-V Uniciclo típicas em Cyclone IV: LEs: 30-60%; Memória: 20-40%. Portanto, os resultados estão dentro da faixa ideal, indicando margem para funcionalidades adicionais.

Conclusão:

- Registradores (17%) e LEs (51%) com boa margem para expansão.
- Nenhum recurso está acima de 60%, indicando bom dimensionamento.

Tabela 5 (a): Análise de Timing - Slack (Folga Temporal).

Métrica	**Valor Obtido**	**O que Significa?**
Setup Slack	+7.502 ns	O circuito opera 7.502 ns mais rápido que o necessário para o clock de 20 ns (50 MHz).
Hold Slack	+2.707 ns	Os dados estão estáveis por 2.707 ns após a borda do clock, evitando metaestabilidade.

Conclusão:

- Setup Slack > 0 ns: O processador funciona corretamente em 50 MHz (20 ns).
- Hold Slack > 0 ns: Não há risco de violação de hold (dados não mudam rápido demais).

Tabela 5 (b): Análise de Timing - Frequência Máxima (Fmax).

Métrica	Valor Obtido	O que Significa?
Fmax	80.01 MHz	Máxima frequência que o circuito suporta sem violar timing.
Restricted Fmax	80.01 MHz	Fmax considerando restrições de projeto (ex.: atrasos de I/O).

Conclusão:

- O processador pode operar até 80,01 MHz (12.5 ns), mas foi projetado para 50 MHz (20 ns), garantindo margem de segurança.

Tabela 5 (c) e (d): Análise de Timing - Clock-to-Output Times e Minimum Clock-to-Output Times.

	Data Port	Clock Port	Rise	Fall	Clock E
1	ClockDIV	CLK1	5.542	5.520	Rise
2	Instr[*]	CLK1	10.705	10.768	Rise
1	Instr[0]	CLK1	8.362	8.343	Rise
2	Instr[1]	CLK1	8.798	8.812	Rise
3	Instr[2]	CLK1	8.686	8.635	Rise
4	Instr[3]	CLK1	8.483	8.447	Rise
5	Instr[4]	CLK1	8.376	8.361	Rise
6	Instr[5]	CLK1	8.475	8.442	Rise
7	Instr[6]	CLK1	8.218	8.182	Rise
8	Instr[7]	CLK1	9.009	9.000	Rise
9	Instr[8]	CLK1	9.639	9.615	Rise
10	Instr[9]	CLK1	9.077	9.059	Rise
11	Instr[10]	CLK1	10.277	10.282	Rise
12	Instr[11]	CLK1	9.804	9.787	Rise
13	Instr[12]	CLK1	9.474	9.454	Rise
14	Instr[13]	CLK1	9.612	9.687	Rise
15	Instr[14]	CLK1	9.156	9.148	Rise
16	Instr[15]	CLK1	9.384	9.382	Rise
17	Instr[16]	CLK1	9.431	9.484	Rise
18	Instr[17]	CLK1	9.369	9.412	Rise
19	Instr[18]	CLK1	9.822	9.832	Rise
20	Instr[19]	CLK1	9.333	9.352	Rise
21	Instr[20]	CLK1	10.081	10.094	Rise
22	Instr[21]	CLK1	9.739	9.817	Rise
23	Instr[22]	CLK1	9.292	9.384	Rise
24	Instr[23]	CLK1	9.614	9.663	Rise
25	Instr[24]	CLK1	10.705	10.768	Rise
26	Instr[25]	CLK1	9.894	9.844	Rise
27	Instr[26]	CLK1	9.284	9.266	Rise
28	Instr[27]	CLK1	10.076	10.044	Rise
29	Instr[28]	CLK1	9.948	10.026	Rise
30	Instr[29]	CLK1	8.499	8.473	Rise
31	Instr[30]	CLK1	8.680	8.688	Rise
32	Instr[31]	CLK1	8.713	8.693	Rise
3	Regout[*]	CLK1	16.484	16.483	Rise
1	Regout[0]	CLK1	15.069	14.932	Rise
2	Regout[1]	CLK1	15.254	15.177	Rise
3	Regout[2]	CLK1	15.630	15.502	Rise
4	Regout[3]	CLK1	15.752	15.616	Rise

	Data Port	Clock Port	Rise	Fall	Clock
1	ClockDIV	CLK1	5.366	5.345	Rise
2	Instr[*]	CLK1	7.938	7.903	Rise
1	Instr[0]	CLK1	8.074	8.055	Rise
2	Instr[1]	CLK1	8.498	8.510	Rise
3	Instr[2]	CLK1	8.387	8.336	Rise
4	Instr[3]	CLK1	8.192	8.156	Rise
5	Instr[4]	CLK1	8.087	8.072	Rise
6	Instr[5]	CLK1	8.185	8.152	Rise
7	Instr[6]	CLK1	7.938	7.903	Rise
8	Instr[7]	CLK1	8.699	8.689	Rise
9	Instr[8]	CLK1	9.302	9.278	Rise
10	Instr[9]	CLK1	8.763	8.745	Rise
11	Instr[10]	CLK1	9.913	9.917	Rise
12	Instr[11]	CLK1	9.462	9.444	Rise
13	Instr[12]	CLK1	9.142	9.122	Rise
14	Instr[13]	CLK1	9.275	9.345	Rise
15	Instr[14]	CLK1	8.839	8.829	Rise
16	Instr[15]	CLK1	9.058	9.055	Rise
17	Instr[16]	CLK1	9.104	9.154	Rise
18	Instr[17]	CLK1	9.045	9.086	Rise
19	Instr[18]	CLK1	9.477	9.486	Rise
20	Instr[19]	CLK1	9.011	9.028	Rise
21	Instr[20]	CLK1	9.729	9.740	Rise
22	Instr[21]	CLK1	9.399	9.472	Rise
23	Instr[22]	CLK1	8.971	9.058	Rise
24	Instr[23]	CLK1	9.278	9.324	Rise
25	Instr[24]	CLK1	10.374	10.437	Rise
26	Instr[25]	CLK1	9.545	9.496	Rise
27	Instr[26]	CLK1	8.962	8.943	Rise
28	Instr[27]	CLK1	9.724	9.692	Rise
29	Instr[28]	CLK1	9.648	9.726	Rise
30	Instr[29]	CLK1	8.211	8.184	Rise
31	Instr[30]	CLK1	8.379	8.387	Rise
32	Instr[31]	CLK1	8.414	8.393	Rise
3	Regout[*]	CLK1	10.699	10.636	Rise
1	Regout[0]	CLK1	13.034	12.888	Rise
2	Regout[1]	CLK1	13.213	13.124	Rise
3	Regout[2]	CLK1	13.571	13.433	Rise
4	Regout[3]	CLK1	13.689	13.543	Rise

O Clock-to-Output (tCO) indica o atraso entre a borda do clock → Saída estável nos pinos do FPGA.

Tabela 5 (e): Análise de Timing - Propagation Delay.

	Input Port	Output Port	RR	RF	FR	FF
1	Regin[1]	Regout[13]	16.645	16.722	17.311	17.388
2	Regin[1]	Regout[22]	15.991	16.038	16.576	16.623
3	Regin[0]	Regout[13]	15.303	15.380	15.944	16.021
4	Regin[0]	Regout[31]	15.280	15.428	15.701	15.849
5	Regin[0]	Regout[10]	15.093	15.184	15.595	15.686
6	Regin[1]	Regout[1]	15.034	15.121	15.559	15.646
7	Regin[1]	Regout[31]	14.861	15.009	15.455	15.603
8	Regin[0]	Regout[22]	14.787	14.834	15.324	15.371
9	Regin[1]	Regout[15]	14.506	14.530	15.081	15.105
10	Regin[0]	Regout[19]	14.561	14.629	15.079	15.148

Os valores nas colunas RR, RF, FR e FF representam tempos de propagação medidos em nanossegundos (ns), mostrando:

- RR: Tempo de propagação quando tanto a entrada quanto a saída mudam de baixo para alto (Rising)
- RF: Tempo de propagação quando a entrada muda de baixo para alto (Rising) e a saída muda de alto para baixo (Falling)
- FR: Tempo de propagação quando a entrada muda de alto para baixo (Falling) e a saída muda de baixo para alto (Rising)
- FF: Tempo de propagação quando tanto a entrada quanto a saída mudam de alto para baixo (Falling)

Observamos que os tempos de propagação variam de aproximadamente 16,645 ns (mais lento) até 7,000 ns (mais rápido). Estes dados são usados para identificar caminhos críticos: Localizar as combinações input/output com maiores tempos; Otimizar o projeto: Focar nas conexões mais lentas para melhorias; Verificar constraints: Comparar com requisitos de frequência máxima de operação; Balancear tempos: Garantir que diferenças entre caminhos não causem problemas de sincronismo.

A conclusão é de que o processador atende aos requisitos de timing com folga, operando em 50 MHz (20 ns) com um setup slack de +7.502 ns e hold slack de +2.707 ns. A Fmax de 80,01 MHz indica margem para aumento de desempenho. A utilização de recursos está abaixo de 20%, demonstrando eficiência na implementação.

Além disso, segundo Patterson e Waterman [1], em CPUs RISC-V Uniciclo típicas em Cyclone IV: LEs: 30-60%; Memória: 20-40%; Portanto, os resultados estão dentro da faixa ideal, indicando margem para funcionalidades adicionais.

Se os resultados fossem ruins (ex.: slack negativo), possíveis soluções seriam:

- Reduzir a frequência do clock (ex.: usar 40 MHz em vez de 50 MHz).
- Pipeline (dividir estágios críticos).
- Otimizar lógica combinacional (ex.: simplificar operações na ULA).

Clique **AQUI** para acessar o repositório do projeto, onde podem ser encontrados os ‘prints’ em ‘.pdf’ e ‘.png’, que sustentam os dados expostos nas Tabelas 4 e 5 (requisitos físicos e temporais do processador).

(1.5) c) Faça as simulações 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 isso, utilizamos como entrada o programa `de1.s`, previamente montado no **RARS**. O código em assembly foi convertido em `.mif` e carregado nas memórias de instrução e dados do projeto no Quartus, conforme descrito anteriormente.

A Figura 12 apresenta a listagem do programa em assembly, junto aos respectivos códigos de máquina em hexadecimal.

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 12 - Assemble do programa de1.asm no RARS.

Simulação Funcional

Nas formas de onda funcionais (Figuras 13 e 14), a simulação foi observada entre 0 ns e 1,1 µs, suficiente para a execução correta de todas as instruções. Durante a simulação funcional, foi possível observar:

- A correta execução de todas as instruções da ISA reduzida (add, sub, and, or, slt, lw, sw, beq, jal, jalr e addi);
- A alteração dos valores nos registradores, especialmente no `t0`, utilizado como registrador de verificação dos resultados intermediários e finais;
- O salto condicional e incondicional (beq, jal e jalr) ocorrendo de forma adequada, com os respectivos endereços sendo carregados no PC;
- A interação com a memória de dados para carregamento (`lw`) e armazenamento (`sw`), com os dados corretos sendo lidos/escritos nos endereços esperados.

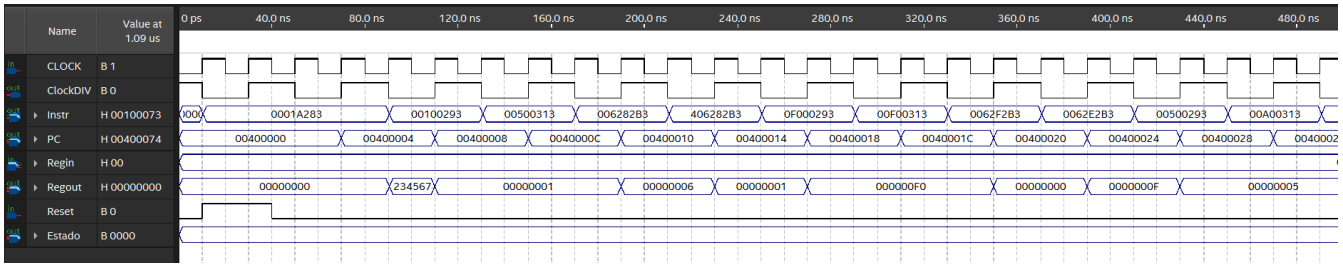

Figura 13 - Simulação por forma de onda funcional (0 - 480 ns) - Clock 20ns (50 MHz).

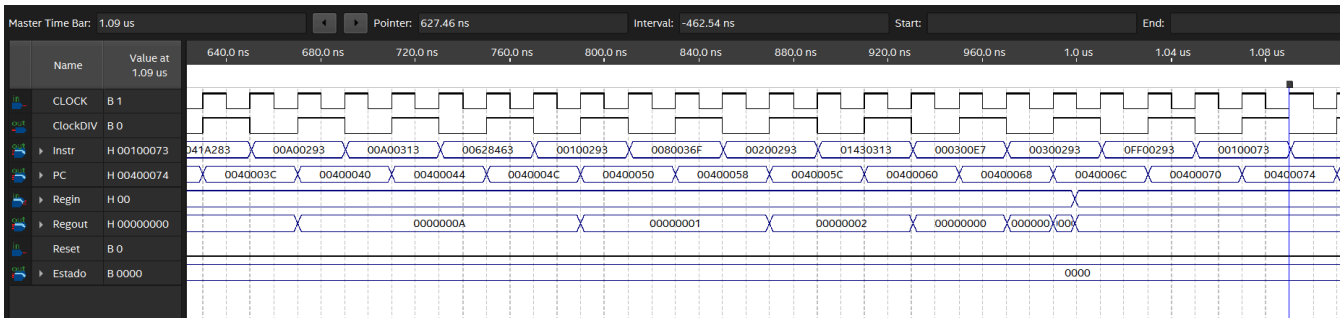

Figura 14 - Simulação por forma de onda funcional (640ns - 1.09us) - Clock 20ns (50 MHz).

Simulação Temporal

A simulação temporal (Figura 15) foi realizada considerando as restrições físicas e o clock real da CPU, operando a 50 MHz (20 ns por ciclo). A execução completa do programa também foi finalizada dentro de 1,1 µs, validando o sincronismo correto do hardware. Respeitando a restrição de dois ciclos de clock por acesso de memória, como especificado no enunciado.

A simulação temporal demonstrou:

- Correta sincronização entre os sinais de controle, registradores e memória;
- Propagação estável dos sinais internos da CPU, sem violação de tempo (slack positivo);
- Os mesmos resultados observados na simulação funcional foram reproduzidos na simulação temporal, validando o comportamento da CPU também sob restrições físicas e temporais.
- Oscilações intermediárias nos sinais de controle ou dados antes de se estabilizarem.
- Transições múltiplas de valores dentro de um mesmo ciclo de clock **(glitches)**.

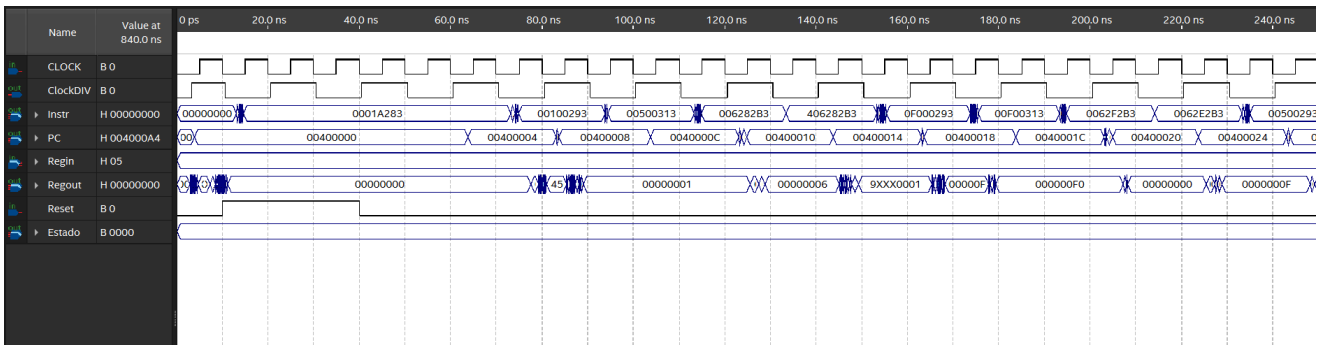

Figura 15 - Simulação por forma de onda temporal (0ns - 204ns) - Clock 20ns (50 MHz).

Dessa forma, as simulações demonstraram que a CPU uniciclo está funcionando conforme o esperado, tanto em termos lógicos quanto temporais, validando o design e a integração dos módulos da arquitetura com os arquivos de programa fornecidos.

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

De acordo com a análise de **timing** realizada após a síntese do projeto no Quartus Prime (Tabela 5(b)), a frequência máxima de operação da CPU foi determinada como:

- **Fmax = 80,01 MHz**, o que equivale a um período mínimo de clock de **12,5 ns**.

Ao configurar o clock com uma frequência **acima da Fmax (80,01 MHz)**, ou seja, com período menor que 12,5 ns, podem ocorrer:

- **Violação de setup time:** os dados não chegam a tempo nos registradores antes do próximo pulso de clock;
- **Instabilidade nos sinais:** a CPU pode começar a apresentar **resultados incorretos**, como valores errados nos registradores ou saltos imprevistos no PC;
- **Falhas de operação** intermitentes ou constantes, principalmente em instruções que exigem propagação por caminhos críticos mais longos (ex: `lw`, `sw`, `jalr`).

Em outras palavras, o circuito pode se comportar de forma imprevisível se a frequência ultrapassar os limites físicos garantidos pelo relatório de temporização.

Para verificar esses efeitos, o projeto foi simulado com diferentes frequências de clock no arquivo `.vwf`, alterando o período da onda de clock:

Tabela 6: Expectativa para verificação experimental da máxima frequência de clock da CPU.

Frequência (MHz)	Período (ns)	Esperado
60 MHz	16,67 ns	Provável funcionamento correto
80 MHz	12,50 ns	Fmax teórica, deve funcionar
100 MHz	10,00 ns	Risco alto de falha
110 MHz	9,00 ns	Muito acima da Fmax → falhas prováveis

As Figuras 16-19 apresentam a forma de onda temporal para cada uma das frequências na tabela 6. Essas simulações ajudam a validar experimentalmente o limite do processador e identificar margens reais de operação além da especificação.

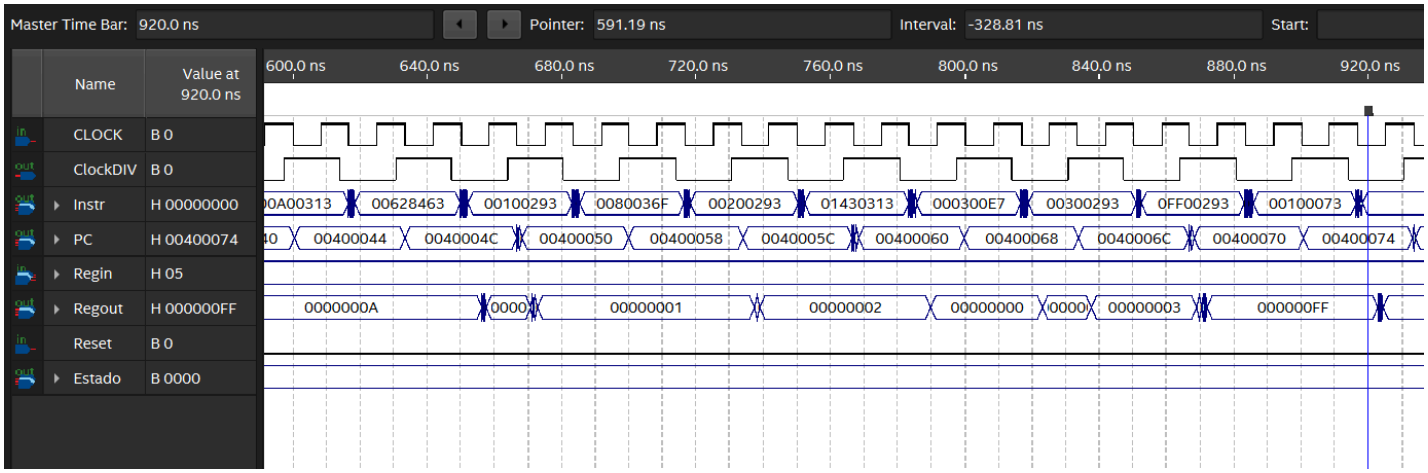

Figura 15 - Simulação por forma de onda temporal - Clock 16,67ns (60 MHz).

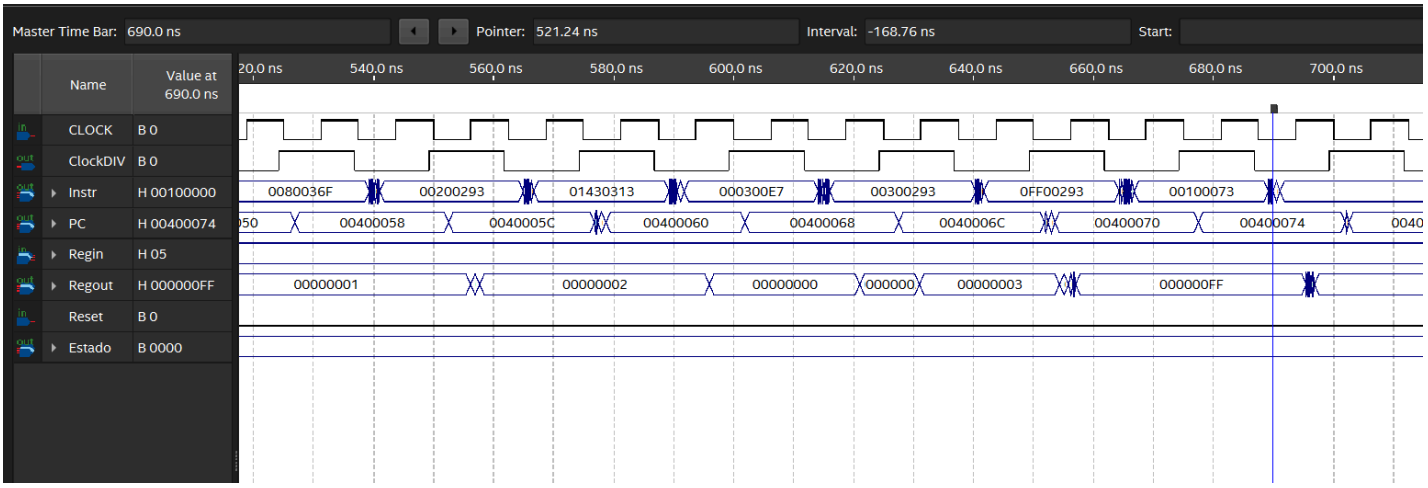

Figura 17 - Simulação por forma de onda temporal - Clock 12,50ns (80 MHz).

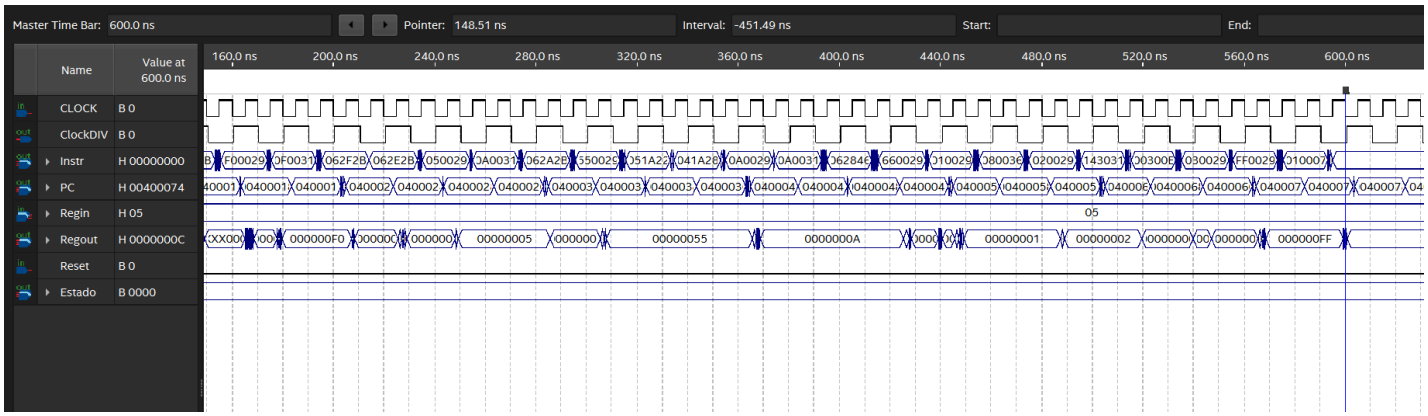

Figura 18 - Simulação por forma de onda temporal - Clock 10,00ns (100 MHz).

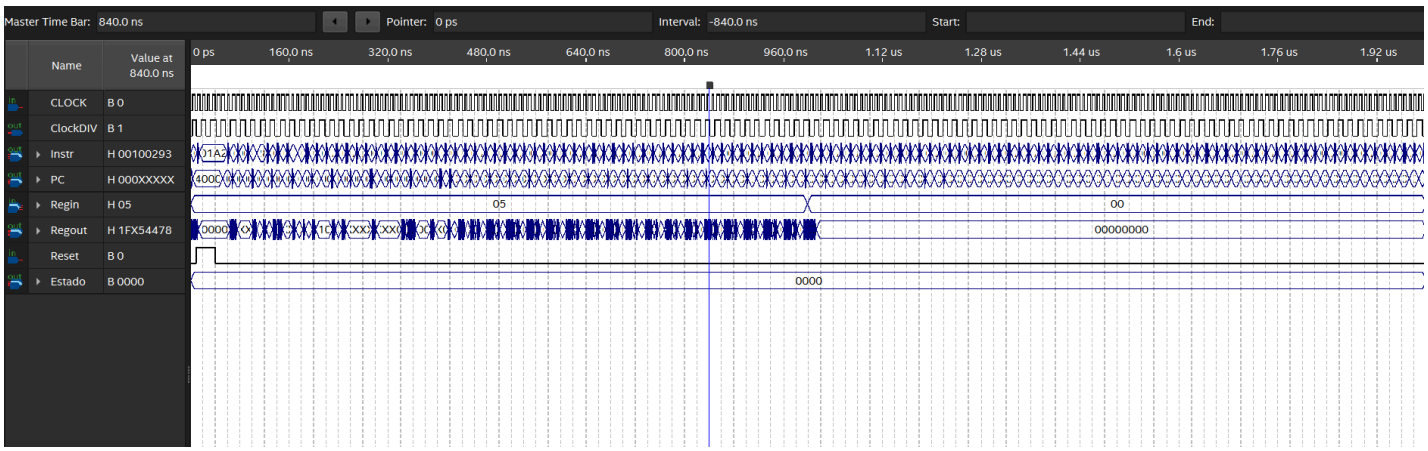

Figura 19 - Simulação por forma de onda temporal - Clock 9,00ns (110 MHz).

A CPU continuou funcionando corretamente, implementado integralmente todas as instruções, até 100 MHz, sugerindo que a margem de segurança prática é um pouco superior à estimativa da Fmax. No entanto, ao atingir 110 MHz, 9 ns (Figura 19), erros começaram a aparecer nas simulações temporais — valores incorretos em registradores e falhas no fluxo de controle foram observadas, caracterizando violação de tempo de setup e comportamento instável do circuito.

Conclusão:

Esses testes demonstram que, embora a Fmax estimada seja 80,01 MHz, a CPU mantém funcionamento correto até cerca de 100 MHz em simulações. Ainda assim, para operação confiável em FPGA físico, recomenda-se seguir a frequência de projeto abaixo de 80 MHz, garantindo robustez contra variações físicas e de temperatura.

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.