Universidade de Brasília
Departamento de Ciência da Computação

- 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)

Acrescentando os Registradores de Pipeline

A transição de uma arquitetura de ciclo único para uma de pipeline é fundamentalmente definida pela segmentação do datapath em estágios independentes. Para alcançar essa segmentação, foram introduzidos quatro bancos de registradores intermediários, que atuam como "estações de parada" entre os cinco estágios do pipeline: IF (Busca), ID (Decodificação), EX (Execução), MEM (Memória) e WB (Escrita de Volta).

A função desses registradores é capturar, ao final de cada ciclo de clock, todos os dados e sinais de controle gerados em um estágio e fornecê-los de forma estável para o estágio seguinte no próximo ciclo. Isso permite que até cinco instruções diferentes sejam processadas simultaneamente, cada uma em um estágio diferente, aumentando drasticamente a vazão (throughput) do processador.

A Tabela 1 possui os links para o repositório, que levam à cada um dos registradores implementados em VHDL.

A Figura 1 mostra a visão RTL (Register-Transfer Level) do datapath em pipeline, onde os principais registradores intermediários (IF_ID_reg, ID_EX_reg, EX_MEM_reg e MEM_WB_reg) estão destacados.

A resposta da questão a seguir (1.2.a) apresenta a visão RTL destes módulos (Figuras 5(a,b,c,d)) que devem ajudar na compreensão das informações detalhadas abaixo.

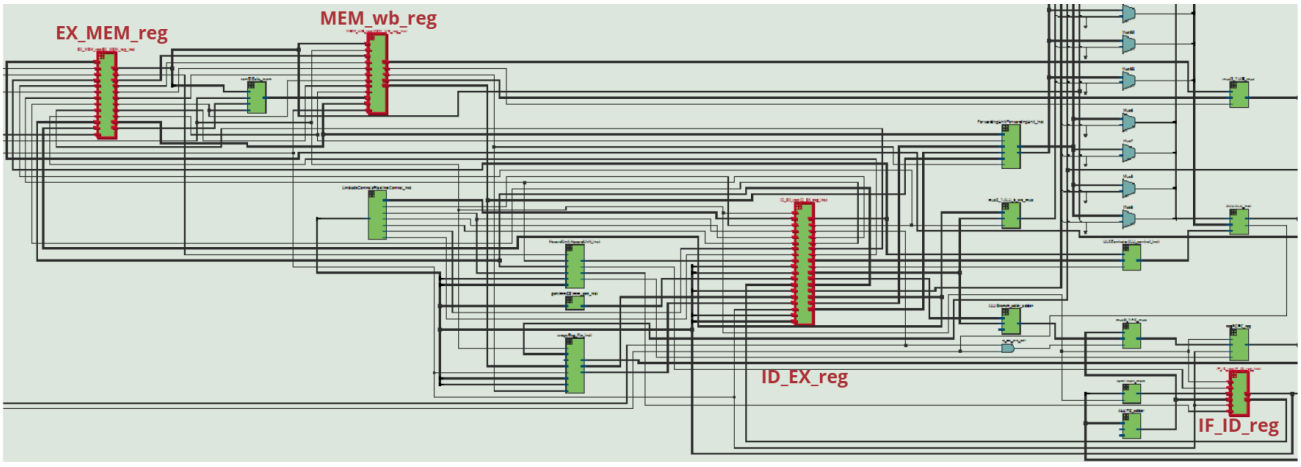

Figura 1 - Visão RTL do Datapath Pipeline com Registradores Intermediários Destacados.

A seguir, detalhamos as informações armazenadas por cada um dos registradores de pipeline:

- **Registrador IF/ID:**
  - **Função:** Armazena a instrução que acabou de ser buscada da memória e o endereço da próxima instrução sequencial.
  - **Informações Armazenadas:**
    - Instrução [31:0]: O código de máquina de 32 bits da instrução.
    - PC_plus_4 [31:0]: O endereço da próxima instrução (PC + 4), necessário para calcular desvios relativos e para ser salvo em instruções como jal.

- **Registrador ID/EX:**
  - **Função:** Este é o registrador mais "largo", pois carrega todos os resultados da decodificação. Ele passa para o estágio de Execução os operandos, o imediato e todos os sinais de controle que comandarão os estágios futuros.
  - **Informações Armazenadas:**
    - PC_plus_4 [31:0]
    - Dado do Registrador 1 [31:0] (lido de rs1)
    - Dado do Registrador 2 [31:0] (lido de rs2)
    - Imediato Estendido [31:0]
    - Endereços rs1, rs2, rd [4:0] (necessários para a Unidade de Adiantamento)
    - Sinais de Controle do WB: RegWrite, MemtoReg
    - Sinais de Controle do MEM: MemRead, MemWrite, Branch
    - Sinais de Controle do EX: ALUSrc, ALUOp

- **Registrador EX/MEM:**
  - **Função:** Armazena os resultados do estágio de Execução, que serão utilizados no acesso à memória.
  - **Informações Armazenadas:**
    - Resultado da ULA [31:0]: Contém o endereço calculado para lw/sw ou o resultado de uma operação aritmética.
    - Dado do Registrador 2 [31:0]: Necessário para ser escrito na memória em uma instrução sw.
    - Flag Zero da ULA: Indica se o resultado da ULA foi zero (essencial para o beq).
    - Endereço rd [4:0]
    - Sinais de Controle do WB e MEM.

- **Registrador MEM/WB:**
  - **Função:** É o último registrador, que segura o dado final a ser escrito no banco de registradores.
  - **Informações Armazenadas:**
    - Dado Lido da Memória [31:0]: O valor vindo da memória após uma instrução lw.
    - Resultado da ULA [31:0]: O resultado de uma operação aritmética que "atravessou" o estágio MEM.
    - Endereço rd [4:0]
    - Sinais de Controle do WB.

A implementação correta desses registradores, incluindo a lógica de flush e stall controlada pela Unidade de Hazard, é a base para o funcionamento de todo o processador pipeline.

1.2) (5.0) Implemente o Processador Pipeline completo com os registradores de pipeline.

Tabela 1 - Componentes do Projeto Pipeline

Componente	Função Principal	Arquivo Fonte (Link)
riscv_pkg	Pacote com definições globais e constantes.	riscv_pkg.vhd
TopDE	Entidade de mais alto nível do projeto.	TopDE.vhd
pipeline	Datapath principal, conectando todos os estágios.	pipeline.vhd
UnidadeControlePipeline	Gera os sinais de controle a partir do opcode.	UnidadeControlePipeline.vhd
ForwardingUnit	Resolve hazards de dados com lógica de adiantamento.	ForwardingUnit.vhd
HazardUnit	Detecta hazards de load-use e de controle, gerando stalls/flushes.	HazardUnit.vhd
IF_ID_reg	Registrador de pipeline entre os estágios IF e ID.	IF_ID_reg.vhd
ID_EX_reg	Registrador de pipeline entre os estágios ID e EX.	ID_EX_reg.vhd
EX_MEM_reg	Registrador de pipeline entre os estágios EX e MEM.	EX_MEM_reg.vhd
MEM_WB_reg	Registrador de pipeline entre os estágios MEM e WB.	MEM_WB_reg.vhd
xregs	Banco de registradores RISC-V.	xregs.vhd
ALU	Unidade Lógica e Aritmética.	alu.vhd

a) Visualização dos Blocos Funcionais com o Netlist RTL View

A visualização do netlist RTL (Register-Transfer Level), gerada pela ferramenta **Netlist Viewers** do Quartus Prime, é essencial para validar a estrutura do hardware que foi inferida a partir do código VHDL. Ao contrário de uma simples lista de componentes, a visão RTL revela a interconexão e o fluxo de dados entre os blocos funcionais, confirmando que a arquitetura pipeline foi sintetizada conforme o planejado.

Visão de Topo do Sistema (TopDE)

A Figura 3 apresenta a visão de mais alto nível, correspondente à entidade TopDE. Esta imagem demonstra a natureza hierárquica do projeto, encapsulando toda a complexidade do processador em um único bloco, Pipeline_Processor_inst. As únicas conexões visíveis são as portas de I/O globais, como CLOCK, Reset, e as saídas de depuração que foram criadas para a verificação do sistema.

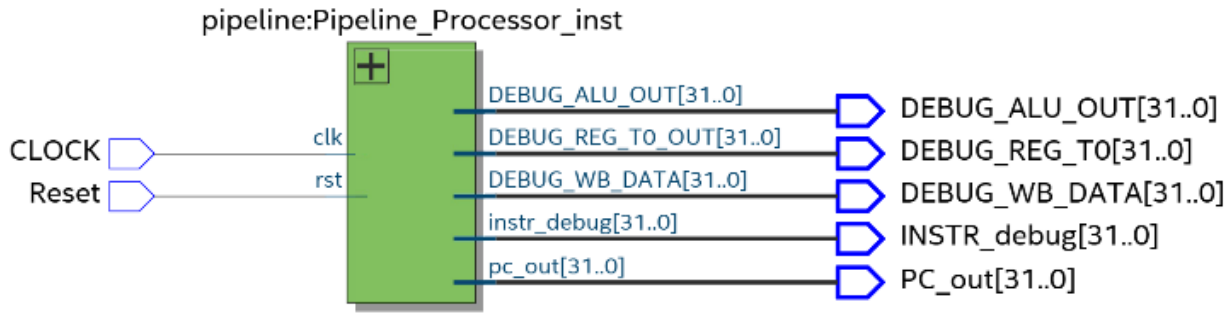

Figura 3 - Visão RTL do módulo de topo TopDE.

Visão do Datapath Pipeline (pipeline)

A visão mais importante é a do interior da entidade pipeline, apresentada na Figura 4. Para tornar o diagrama claro e legível, os inúmeros multiplexadores de 1 bit, inferidos para a lógica de adiantamento (forwarding) e seleção de operandos da ULA, ocultados no RTL Viewer.

Fica evidente o fluxo da instrução da esquerda para a direita, assim como os "atalhos" de dados criados pela Unidade de Adiantamento (as linhas que saem dos estágios posteriores e voltam para as entradas da ULA).

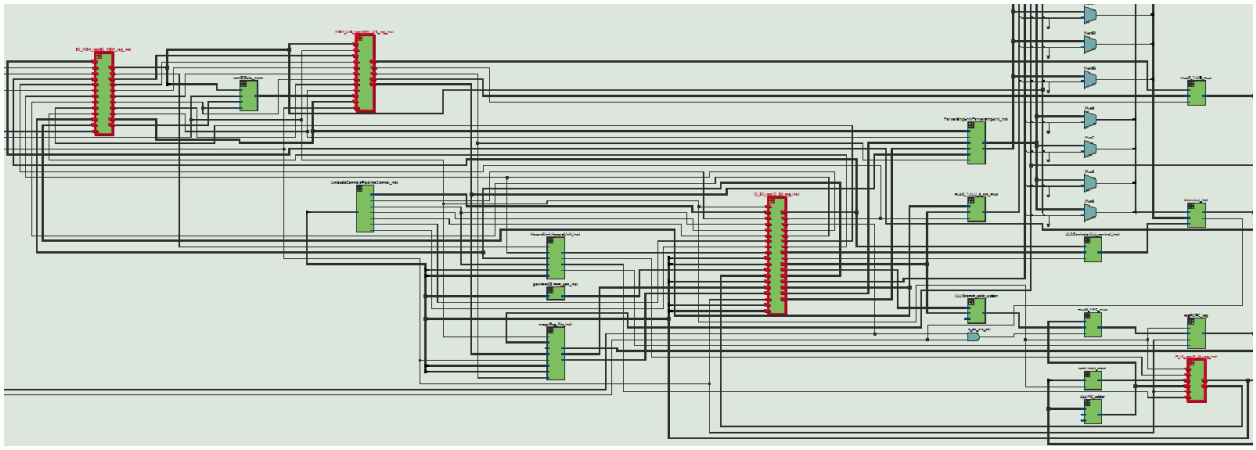

Figura 4 - Visão RTL organizada do datapath pipeline, destacando os registradores e a lógica de adiantamento ocultada.

Visão dos Registradores de Pipeline

As Figuras 5 (a, b, c, d) os principais registradores intermediários (respectivamente: IF_ID_reg, ID_EX_reg, EX_MEM_reg e MEM_WB_reg).

As informações armazenadas por cada um dos registradores de pipeline foram detalhadas na resposta da questão 1.1

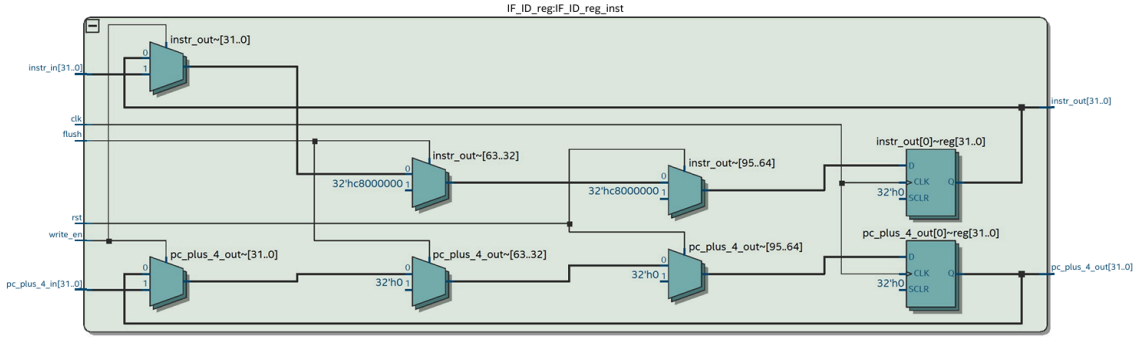

Figura 5(a) - Visão RTL do Registrador IF/ID.

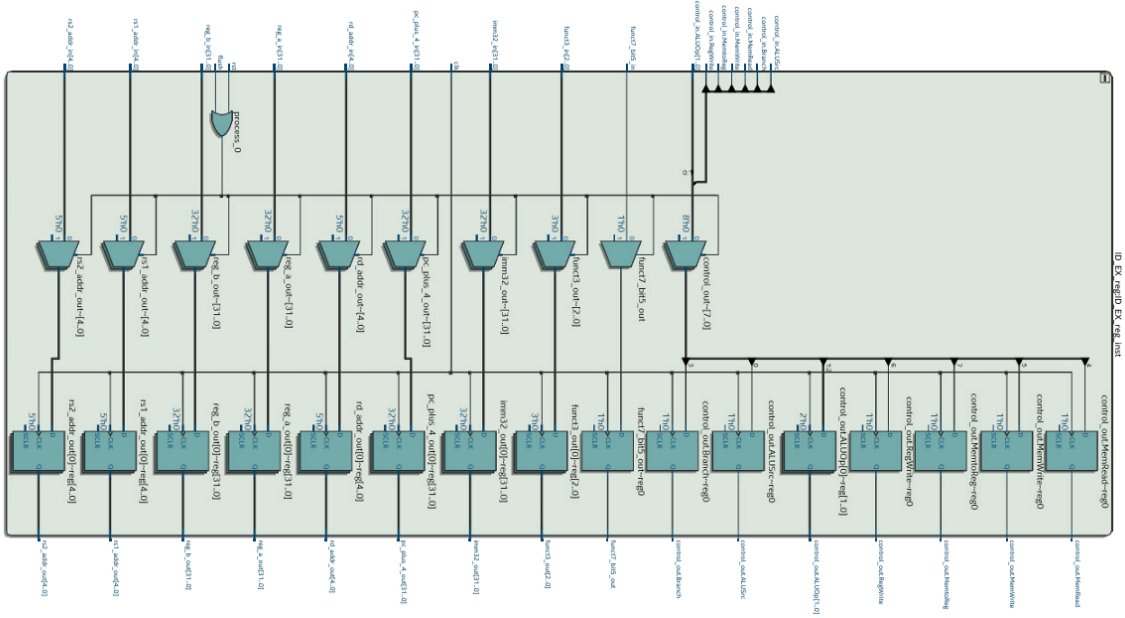

Figura 5(b) - Visão RTL do Registrador ID/EX.

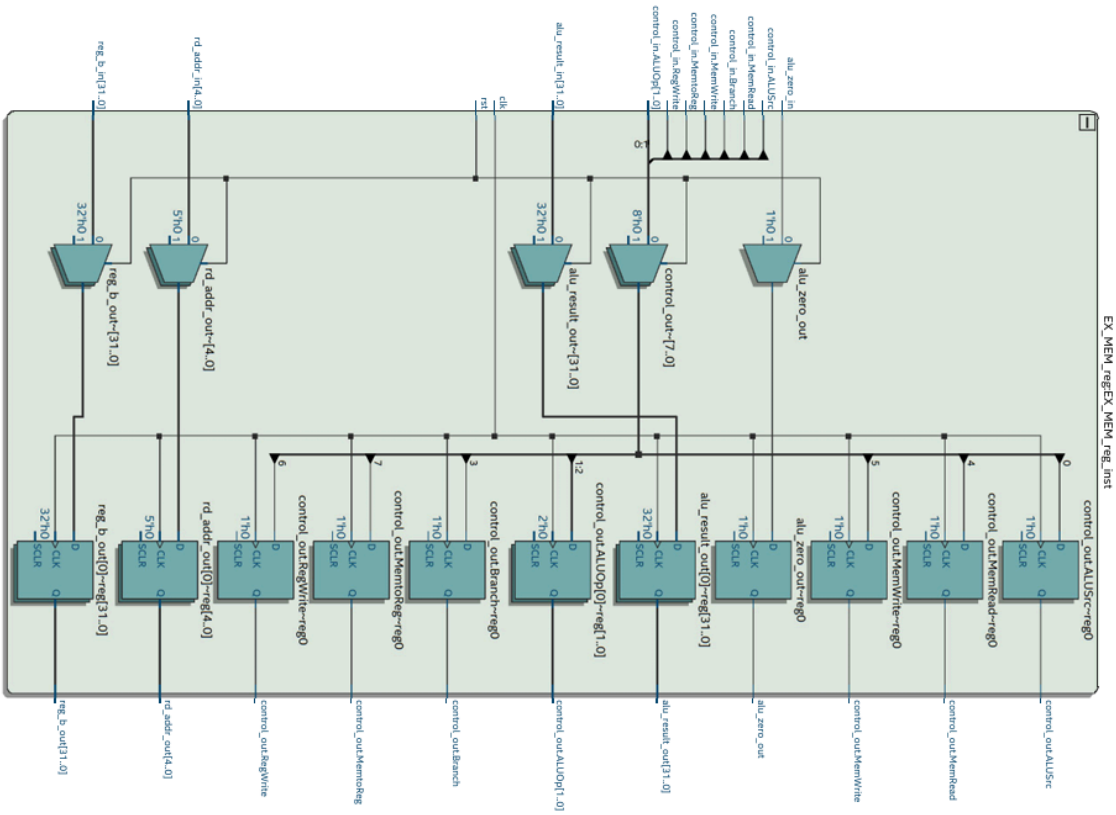

Figura 5(c) - Visão RTL do Registrador EX/MEM.

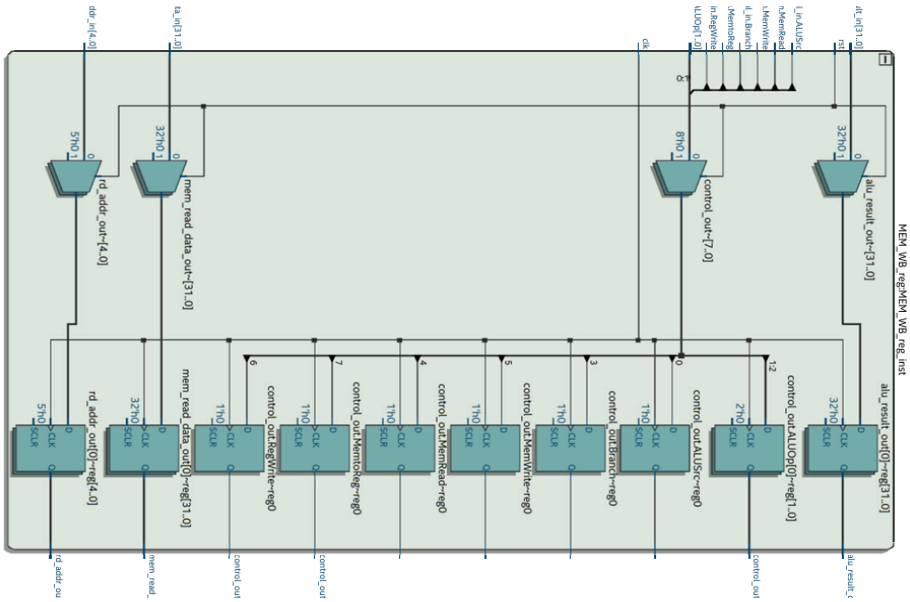

Figura 5(d) - Visão RTL do Registrador MEM/WB.

Visão das Unidades de Controle

Finalmente, as Figuras 6 (a, b, c) mostram os "cérebros" do processador. Os diagramas exibem a UnidadeControlePipeline (Fig. 6a), um bloco combinacional que decodifica o opcode. Então, a ForwardingUnit (Fig. 6b), e a HazardUnit (Fig. 6c), recebem informações dos estágios do pipeline para tomar decisões críticas sobre adiantamento de dados, stalls (paradas) e flushes (limpezas), garantindo a execução correta do programa.

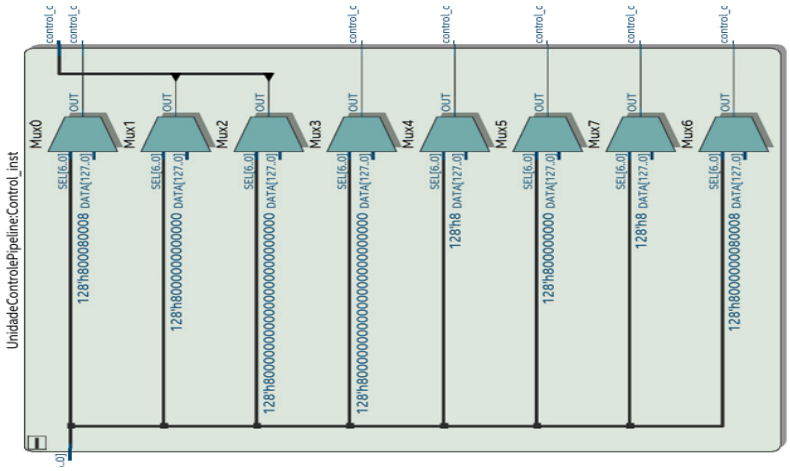

Figura 6(a) - Visão RTL das unidades de controle.

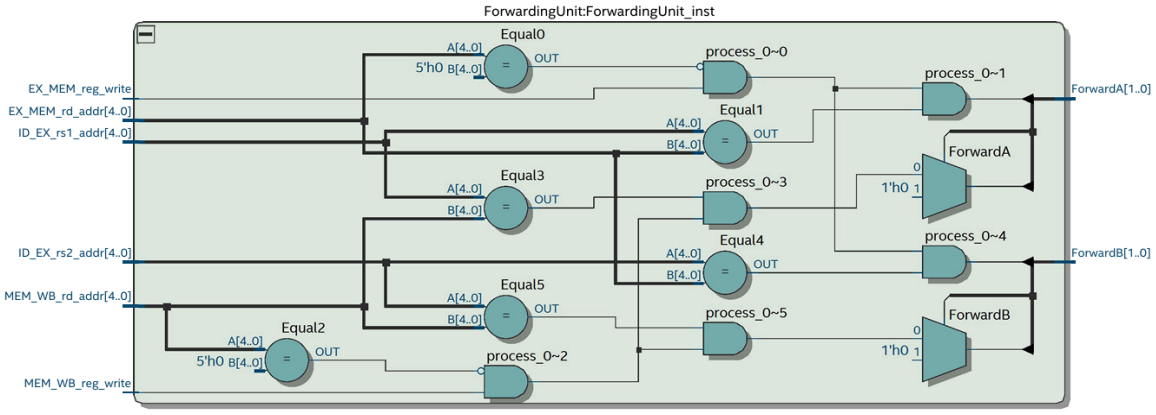

Figura 6(b) - Visão RTL das unidades adiantamento.

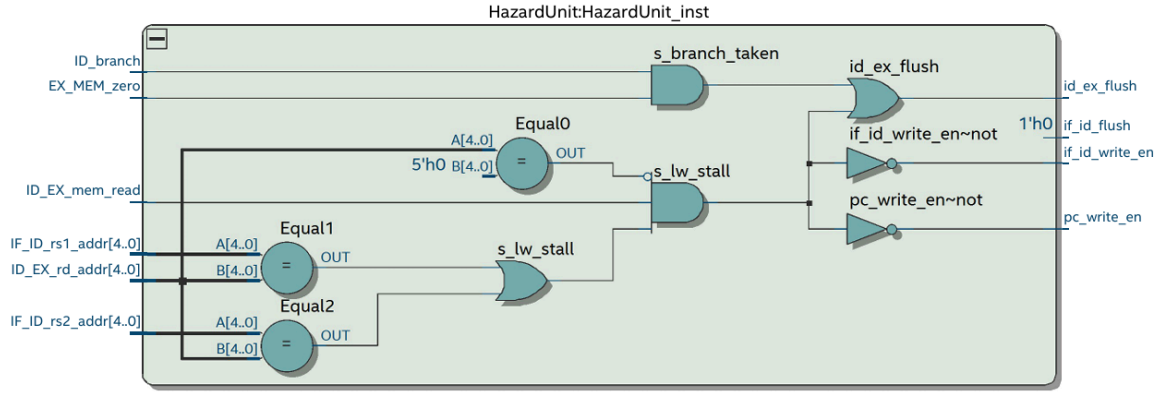

Figura 6(c) - Visão RTL das unidades de detecção de hazards.

A análise hierárquica através do RTL Viewer, com a devida organização dos seus elementos, não só valida que a estrutura foi sintetizada conforme o planejado, mas também serve como uma poderosa ferramenta visual para compreender o fluxo de dados e controle da complexa arquitetura pipeline.

b) Requisitos físicos e temporais do processador.

Após a compilação bem-sucedida do projeto no Quartus Prime e a configuração de um clock alvo de 50 MHz (período de 20 ns) através do arquivo de restrições pipeline.sdc e também pelo Timing Analysis, foram gerados os relatórios de síntese e análise de timing. Estes relatórios detalham o consumo de recursos físicos no FPGA e o desempenho temporal do processador.

Análise de Recursos Físicos

O relatório "Flow Summary" detalha o consumo de recursos lógicos e de memória no dispositivo FPGA Altera Cyclone IV E (EP4CE15F23C6). Os resultados estão consolidados na Tabela 2.

Tabela 2: Utilização de Recursos Físicos no Pipeline

Recurso	Utilizado	Disponível	Utilização (%)
Logic elements	2,312	15,408	15%
Total registers	1,374	-	-
Total pins	162	344	47%
Total memory bits	65,536	516,096	13%

Análise:
A utilização de recursos pelo processador pipeline é significativamente maior em comparação com um design de ciclo único. O aumento no número de elementos lógicos (15%) e, principalmente, de registradores (1,374), é uma consequência direta da arquitetura:

1. **Registradores de Pipeline:** A adição dos quatro grandes registradores (IF/ID, ID/EX, EX/MEM, MEM/WB) para separar os estágios consome uma quantidade considerável de registradores.
2. **Lógica de Hazard e Adiantamento:** As unidades ForwardingUnit e HazardUnit, juntamente com os multiplexadores associados nas entradas da ULA, adicionam lógica combinacional para gerenciar os conflitos do pipeline.

O uso de memória de 65.536 bits corresponde exatamente à soma das memórias de instrução (ramI) e dados (ramD), cada uma com 32k bits, refletindo a arquitetura Harvard implementada para evitar hazards estruturais.

Análise de Timing (Requisitos Temporais)

A análise de timing, realizada pela ferramenta TimeQuest, valida se o design consegue operar na frequência de clock especificada. Os relatórios (Setup_Summary, Hold_Summary, Fmax_Summary), sintetizados na Tabela 3, confirmam que o design cumpre os requisitos.

Tabela 3: Resumo da Análise de Timing do Pipeline

Métrica	Valor Obtido	Significado
Setup Slack	+6.608 ns	O caminho de dados mais lento do circuito termina 6.608 ns *antes* do necessário para o clock de 50 MHz. Um slack positivo indica que o design é rápido o suficiente e não há violações de setup.
Hold Slack	+0.295 ns	Os dados nos registradores permanecem estáveis por tempo suficiente *após* a borda do clock, garantindo que não haja violações de hold e evitando metaestabilidade. O valor é pequeno, mas positivo, o que é suficiente para a correção.
Fmax (Restrita)	**74.67 MHz**	Esta é a máxima frequência de operação confiável do processador, limitada pelo seu estágio mais lento (o caminho crítico).

Os resultados temporais demonstram o sucesso do design. O slack de setup positivo de +6.608 ns confirma que o projeto atende confortavelmente à meta de 50 MHz. O Fmax de 74.67 MHz indica que o período de clock mínimo é de aproximadamente 13.4 ns. Este valor é determinado pelo estágio mais lento do pipeline, que, após a síntese, define a velocidade máxima de todo o sistema.

Conclusão

O processador pipeline implementado é funcional e robusto. Embora exija mais recursos físicos que arquiteturas mais simples, ele cumpre os requisitos temporais do projeto, operando com uma frequência máxima de 74.67 MHz, o que valida a correção do design e da lógica de tratamento de hazards.

No repositório, encontra-se ainda os relatórios do ‘’Report_Datasheet”: Clock_to_Output_Times; Minimum_Clock_to_Output_Times; Hold_Times; Setup_Times.

c) Simulação Funcional e Temporal com de1.s

Para validar a corretude lógica e o comportamento temporal do processador pipeline, foram realizadas simulações utilizando o programa de teste de1.s. O código Assembly foi modificado com a inserção de instruções nop para tratar os casos de hazard que não são resolvidos pelo hardware de adiantamento.

Alterações no Programa de Teste de1.s

A implementação de uma ForwardingUnit por hardware elimina a grande maioria dos hazards de dados do tipo Leitura-Após-Escrita (RAW). No entanto, dois cenários específicos ainda requerem a inserção manual de uma bolha (nop) no software para garantir a execução correta:

1. **Hazard de Load-Use:** Ocorre quando uma instrução tenta usar o resultado de um lw *imediatamente* a seguir. Como o dado da memória só está disponível no final do estágio MEM, mesmo a lógica de adiantamento não consegue enviá-lo a tempo para o estágio EX da instrução seguinte. A HazardUnit detecta essa condição e insere um ciclo de stall (parada), mas a inserção de um nop no software é a solução mais robusta para garantir que o pipeline processe uma instrução inofensiva enquanto espera o dado.

2. **Hazard de Controle:** A decisão de um desvio beq só é conhecida no final do estágio EX. Nesse momento, a instrução seguinte (PC+4) já foi buscada especulativamente. O nop é inserido para preencher a "ranhura de atraso do desvio" (branch delay slot), garantindo que a instrução que é anulada (flushed) pela Unidade de Hazard seja uma operação nula.

A Figura 7 apresenta o assembly do programa de.1 no RARS, que são os resultados esperados para a forma de onda nas simulações funcionais e temporais do processador pipeline, que comprovarão a corretude da implementação.

0x00400000	0x0001a283	lw x5,0(x3)	8:	lw t0, 0(gp)	# t0 = 0x12345678
0x00400004	0x00000013	addi x0,x0,0	9:	nop	# HAZARD: Necessário 1 NOP para o caso de lw-use
0x00400008	0x00100293	addi x5,x0,1	12:	addi t0, zero, 1	# t0 = 1
0x0040000c	0x00500313	addi x6,x0,5	16:	addi t1, zero, 5	
0x00400010	0x006282b3	add x5,x5,x6	17:	add t0, t0, t1	# t0 = 6
0x00400014	0x406282b3	sub x5,x5,x6	19:	sub t0, t0, t1	# t0 = 1
0x00400018	0x0f000293	addi x5,x0,0x000000f0	22:	addi t0, zero, 0xF0	
0x0040001c	0x00f00313	addi x6,x0,15	23:	addi t1, zero, 0x0F	
0x00400020	0x0062f2b3	and x5,x5,x6	25:	and t0, t0, t1	# t0 = 0x00
0x00400024	0x0062e2b3	or x5,x5,x6	27:	or t0, t0, t1	# t0 = 0x0F
0x00400028	0x00500293	addi x5,x0,5	30:	addi t0, zero, 5	
0x0040002c	0x00a00313	addi x6,x0,10	31:	addi t1, zero, 10	
0x00400030	0x0062a2b3	slt x5,x5,x6	33:	slt t0, t0, t1	# t0 = 1
0x00400034	0x05500293	addi x5,x0,0x00000055	36:	addi t0, zero, 0x55	
0x00400038	0x0051a223	sw x5,4(x3)	38:	sw t0, 4(gp)	# MEM[gp+4] = 0x55
0x0040003c	0x0041a283	lw x5,4(x3)	39:	lw t0, 4(gp)	# Carrega o valor de volta
0x00400040	0x00000013	addi x0,x0,0	40:	nop	# HAZARD: Necessário 1 NOP para o lw-use (verifica...
0x00400044	0x00a00293	addi x5,x0,10	43:	addi t0, zero, 10	
0x00400048	0x00a00313	addi x6,x0,10	44:	addi t1, zero, 10	
0x0040004c	0x00628663	beq x5,x6,0x0000000c	46:	beq t0, t1, branch_ok	
0x00400050	0x00000013	addi x0,x0,0	47:	nop	# Bolha para o HAZARD DE CONTROLE
0x00400054	0x06600293	addi x5,x0,0x00000066	48:	addi t0, zero, 0x66	# Não deve executar
0x00400058	0x00100293	addi x5,x0,1	50:	addi t0, zero, 1	
0x0040005c	0x00c0036f	jal x6,0x0000000c	53:	jal t1, jal_test	
0x00400060	0x00000013	addi x0,x0,0	54:	nop	# Bolha para o HAZARD DE CONTROLE
0x00400064	0x06600293	addi x5,x0,0x00000066	55:	addi t0, zero, 0x66	# Não deve executar
0x00400068	0x00200293	addi x5,x0,2	57:	addi t0, zero, 2	
0x0040006c	0x01430313	addi x6,x6,20	60:	addi t1, t1, 0x14	
0x00400070	0x000300e7	jalr x1,x6,0	62:	jalr ra, t1, 0	
0x00400074	0x00000013	addi x0,x0,0	63:	nop	# Bolha para o HAZARD DE CONTROLE
0x00400078	0x06600293	addi x5,x0,0x00000066	64:	addi t0, zero, 0x66	# Não deve executar
0x0040007c	0x00300293	addi x5,x0,3	66:	addi t0, zero, 3	
0x00400080	0x0ff00293	addi x5,x0,0x000000ff	69:	addi t0, zero, 0xFF	
0x00400084	0x00100073	ebreak	70:	ebreak	

Segment

Address	Value (+0)	Value (+4)	Value (+8)	Value (+c)	Value (+10)	Value (+14)	Value (+18)	Value (+1c)
0x00400000	0x0001a283	0x00000013	0x00100293	0x00500313	0x006282b3	0x406282b3	0x0f000293	0x00f00313
0x00400020	0x0062f2b3	0x0062e2b3	0x00500293	0x00a00313	0x0062a2b3	0x05500293	0x0051a223	0x0041a283
0x00400040	0x00000013	0x00a00293	0x00a00313	0x00628663	0x00000013	0x06600293	0x00100293	0x00c0036f
0x00400060	0x00000013	0x06600293	0x00200293	0x01430313	0x000300e7	0x00000013	0x06600293	0x00300293
0x00400080	0x0ff00293	0x00100073	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000

Figura 7 - Visão 'data and text segments' do programa de1.asm no RARS.

Análise da Forma de Onda

A simulação temporal com o código de1.s corrigido foi executada, inicialmente com um cock de 50 MHz (20ns), e a forma de onda resultante, apresentada nas Figura 8 (a,b), comprova o funcionamento correto do processador.

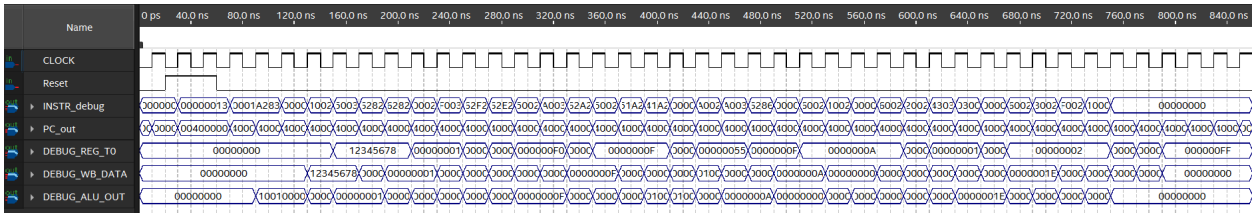

Figura 8.1 - Simulação funcional do processador pipeline (clock de 50 MHz) - executando o programa de teste completo em ~0.750µs.

A simulação funcional executa o programa de1.s por completo em ~0.750. A prova definitiva do sucesso é o sinal DEBUG_REG_T0, que monitora o registrador t0. Ao final da execução, este sinal mostra o valor **0x000000FF**, que é o marcador de sucesso definido na última instrução do programa, confirmando que todas as operações foram realizadas corretamente.

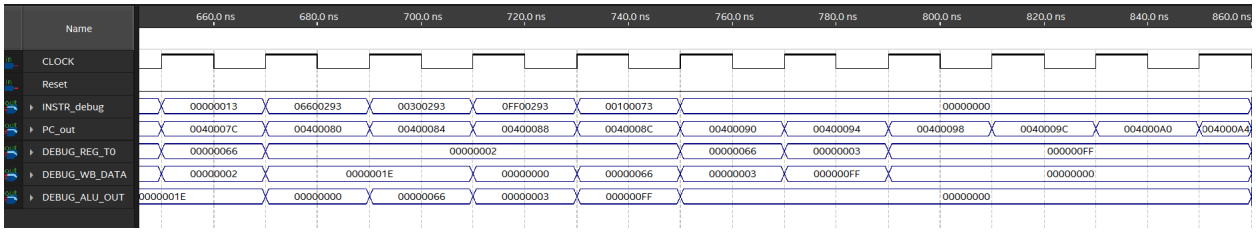

Figura 8.2 - Detalhe da simulação mostrando o tratamento de um hazard de controle.

A Figura 8.2 acima exibe um trecho da simulação temporal que captura o momento exato em que um desvio condicional (beq) é tomado e como a Unidade de Hazard age para garantir a corretude do fluxo de execução. A execução pode ser analisada ciclo a ciclo:

1. **Ciclo 1 (Primeira Coluna):**
    - **INSTR_debug mostra 00000013 (nop):** Esta é a instrução nop que foi inserida manualmente no código assembly, e que está atualmente no estágio **ID**.
    - **PC_out mostra 0040007C:** O PC já avançou para o endereço da instrução seguinte, que é um addi.
2. **Ciclo 2:**
    - **INSTR_debug mostra 06600293 (addi t0, zero, 0x66):** A instrução "proibida" que vem logo após o beq foi buscada e agora está no estágio **ID**.
    - **PC_out mostra 00400080:** O PC avançou sequencialmente para PC+4.
    - **O Evento Chave:** Neste mesmo ciclo, a instrução beq original está no estágio **EX**. A ULA confirma que o desvio deve ser tomado. A HazardUnit detecta isso (s_branch_taken = '1').
3. **Ciclo 3 (A Magia Acontece):**
    - **INSTR_debug mostra 00300293 (addi t0, zero, 3):** O PC foi atualizado para o endereço do label de destino do beq (jair_target), e a instrução correta (addi t0, zero, 3) foi buscada e agora está no estágio **ID**.
    - **A "Bolha":** A HazardUnit, que detectou o desvio no ciclo anterior, gerou um sinal flush. Esse sinal fez com que a instrução "proibida" (addi ... 0x66) fosse **anulada** e substituída por uma bolha (nop) no registrador de pipeline ID/EX. O DEBUG_ALU_OUT mostrando 00000000 (um dos possíveis resultados de um nop) pode ser uma evidência disso.

Esta sequência prova que o hardware está funcionando perfeitamente:

- O processador busca instruções especulativamente (PC+4).
- A HazardUnit detecta corretamente a condição do desvio.
- Ela anula (flush) a instrução incorreta que já estava no pipeline.
- Ela comanda o PC a carregar o endereço de destino correto, garantindo que o fluxo do programa continue sem erros.

Em suma, um sumário de análises interessantes para se observar na forma de onda:

1. **Fluxo de Pipeline Visível:** É possível rastrear o ciclo de vida de uma instrução através dos estágios. Por exemplo, pode-se observar a instrução addi t0, zero, 0xFF (código 0x0FF00293) aparecer em INSTR_debug (estágio ID), seu resultado (0xFF) aparecer em DEBUG_ALU_OUT no ciclo seguinte (estágio EX), e, por fim, o mesmo valor aparecer em DEBUG_WB_DATA dois ciclos depois, pronto para a escrita no registrador (estágio WB).
2. **Prova do Hazard de Controle:** Um dos momentos mais interessantes da simulação ocorre durante a execução da instrução beq. Observa-se que a instrução seguinte, addi t0, zero, 0x66 (código 0x06600293), chega a ser buscada e aparece no sinal INSTR_debug. Este é o comportamento esperado de um fetch especulativo. No entanto, quando o beq é tomado, a HazardUnit anula (flushes) esta instrução "proibida" antes que ela possa ser executada. A prova disso é que o valor do registrador de teste (DEBUG_REG_T0) nunca é alterado para 0x66, validando o funcionamento correto da unidade de controle de hazards.
3. **Resultado Final Correto:** A simulação executa o programa de1.s por completo. A prova definitiva do sucesso é o sinal DEBUG_REG_T0, que monitora o registrador t0. Ao final da execução, este sinal mostra o valor **0x000000FF**, que é o marcador de sucesso definido na última instrução do programa, confirmando que todas as operações foram realizadas corretamente.

Análise da Simulação Temporal

A simulação temporal leva em conta os atrasos de propagação reais do hardware. A Figura 9 mostra o trecho final desta simulação, operando com um clock de 50 MHz.

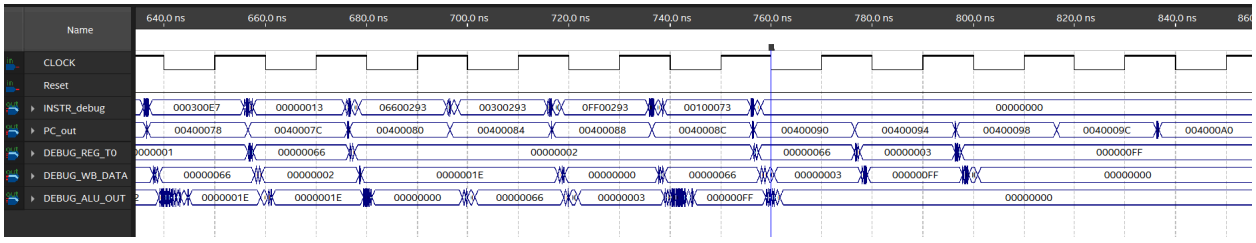

Figura 9 - Detalhe da simulação mostrando o tratamento de um hazard de controle por meio da análise de Simulação Temporal.

Embora a forma de onda temporal apresente "glitches" (ruídos transitórios) nos sinais combinacionais entre as bordas de clock, os valores nos registradores (PC_out, INSTR_debug, etc.) são capturados de forma limpa e correta a cada borda de subida do clock. Isso valida que o design síncrono é robusto e que os tempos de setup e hold são respeitados nesta frequência de operação, confirmando os resultados da simulação funcional.

Os "glitches" e atrasos intermediários nos sinais combinacionais culminam no término da execução em relação à simulação funcional de 750 ns para quase 760 ns (observe a Figura 8.2 em comparação com a Figura 9) - um atraso de quase 0,01 µs.

Em suma, a simulação temporal válida que o processador pipeline, com sua lógica de adiantamento e tratamento de hazards, está funcionalmente correto e executa o programa de teste conforme o esperado.

d) Máxima frequência de clock utilizável na CPU.

A máxima frequência de clock utilizável em uma CPU implementada em um FPGA é determinada pelo seu "caminho crítico", ou seja, o caminho de propagação de sinal mais lento dentro do design. Esta frequência é primeiramente calculada pela ferramenta de análise de timing estático e, em seguida, validada através de simulações temporais para determinar o limite prático de operação.

Análise Teórica (Resultado do TimeQuest Timing Analyzer)

Conforme a análise de timing apresentada na seção anterior (1.2 b), a frequência máxima de operação (Restricted Fmax) calculada pela ferramenta TimeQuest para a CPU pipeline foi de **74.67 MHz**. Isso significa que o período de clock mínimo com o qual o processador pode operar de forma confiável, segundo a análise estática, é:

Período Mínimo = (1 / Frequência máxima) (1 / 74.67×10^6 Hz) ≈ **13.4 ns**

Teoricamente, qualquer tentativa de operar a CPU com um período de clock inferior a 13.4 ns resultará em violações de *setup time*, levando a um funcionamento incorreto e imprevisível do hardware.

Verificação Experimental

Para validar este limite na prática, o programa de teste de1.s foi executado através de simulações temporais. O período do sinal CLOCK no arquivo de forma de onda (.vwf) foi alterado para testar o comportamento da CPU em diferentes frequências, acima e abaixo do limite teórico. A Tabela 4 resume os resultados observados em cada teste:

Tabela 4: Resultados da Simulação Experimental por Frequência

Frequência de Teste	Período do Clock	Resultado da Simulação Temporal
50 MHz	20.0 ns	**Sucesso.** O programa executa corretamente, e o valor final no registrador de teste é o esperado.
~77.0 MHz	13.0 ns	**Sucesso.** Operando muito próximo limite da Fmax teórica, o processador ainda funciona corretamente, como esperado de um slack positivo.
~83.3 MHz	12.0 ns	**Falha.** Acima da Fmax, o design começa a apresentar falhas de timing.

Análise dos Resultados da Simulação:

- **A 50 MHz e 77 MHz:** As simulações temporais (**Figuras 9 e 10**) foram bem-sucedidas. Apesar dos "glitches" normais em sinais combinacionais, os valores nos registradores de pipeline e no banco de registradores foram capturados corretamente em cada borda de clock, e o programa executou até o fim com o resultado correto.
- **A 80 MHz (Período de 12.5 ns):** Incrementamos a operação da CPU acima de sua Fmax, até a falha da simulação temporal. O período de clock de 12.0 ns se mostrou insuficiente para que os sinais nos caminhos críticos se propagassem e estabilizassem. O comportamento observado foi a captura de valores incorretos nos registradores de pipeline, corrompendo o fluxo de execução e fazendo com que o processador buscasse instruções erradas e travasse, não completando o programa. A **Figura 11** ilustra este comportamento.

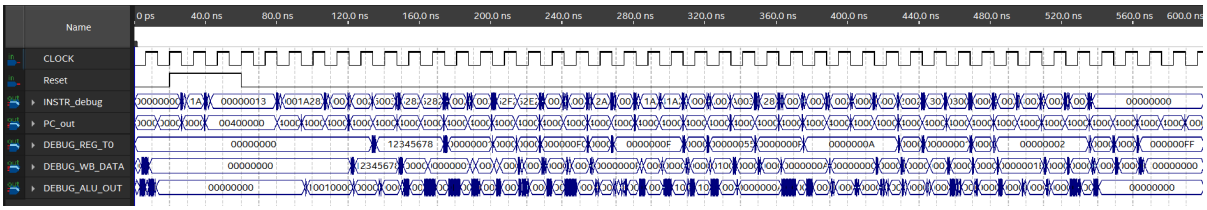

Figura 10 - Simulação temporal - Fmax teórica ~77,0 MHz (período de 13,0 ns).

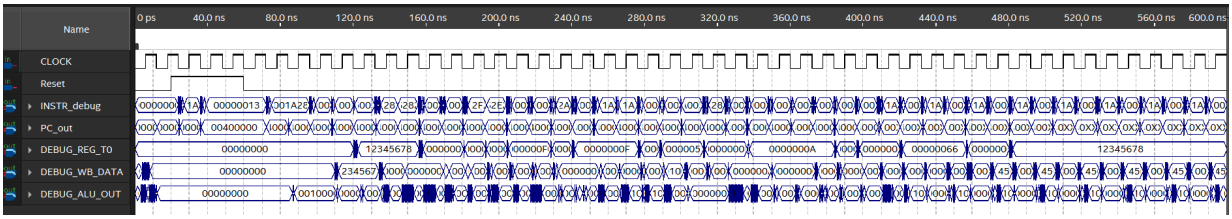

Figura 11 - Simulação temporal em 83,33 MHz (12.0 ns) - a falha de timing.

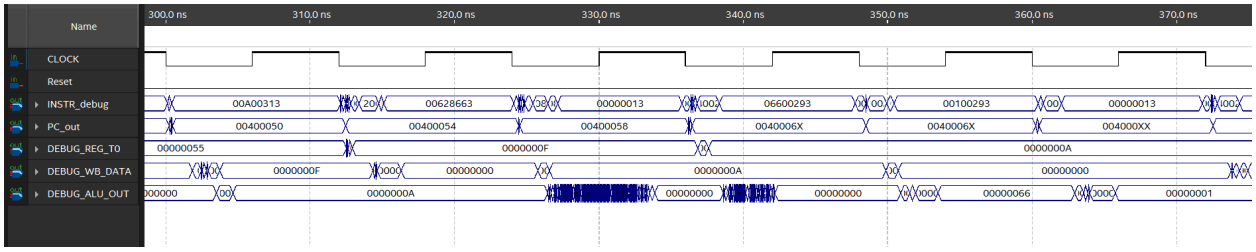

Figura 11.1 - Trecho da Simulação Temporal em 83,33 MHz - a falha de timing.

Neste trecho (Figura 11.1), podemos observar o seguinte cenário de falha:

1. **Evento:** Uma instrução de desvio (beq - 00628663) está no estágio EX, e a condição para o salto é satisfeita (o sinal s_ex_alu_zero torna-se '1').
2. **Caminho Crítico:** A informação (s_ex_alu_zero) precisa se propagar da ULA até a lógica de controle (s_pc_src_sel) para comandar o multiplexador do PC a selecionar o novo endereço de desvio.
3. **Violação de Setup:** Com um período de apenas 12 ns, este caminho crítico não tem tempo suficiente para se estabilizar antes da próxima borda de subida do clock. A borda de subida do clock chega **enquanto o glitch ainda está acontecendo**. O sinal na entrada do registrador `EX/MEM` ainda não se estabilizou. O registrador é forçado a capturar um valor "lixo", um dos valores intermediários e instáveis do glitch.
4. **Resultado:** O registrador PC captura um valor incorreto ou "metaestável" (representado por 'X's ou um valor inesperado na simulação). A partir deste ponto, o processador começa a buscar instruções de endereços errados, corrompendo completamente o fluxo de execução, o que caracteriza a falha de timing.

Conclusão

A verificação experimental valida com precisão a análise teórica do TimeQuest. O processador opera de forma estável com um período de 13 ns (~77,0 MHz), mas falha com 12 ns (~83.3 MHz). Portanto, a máxima frequência de clock utilizável de forma **confiável** na CPU implementada está em total concordância com o **Fmax de 74.67 MHz** calculado pela ferramenta de análise, confirmando a robustez do design e a precisão da análise de timing.

1.3) (2.0) Comparando os requerimentos físicos e temporais dos seus 3 processadores (Uni, Multi e Pipe).

A implementação das três arquiteturas de processador — Uniciclo, Multiciclo e Pipeline — permite uma análise comparativa direta sobre os trade-offs fundamentais entre custo de hardware (recursos físicos) e desempenho (requisitos temporais). Os dados para cada design foram extraídos dos relatórios de compilação do Quartus Prime.

Tabela Comparativa de Métricas de Arquitetura

Métrica	Processador Uniciclo (Lab 2)	Processador Multiciclo (Lab 3)	Processador Pipeline (Lab 4)
Arquitetura de Memória	Harvard	Von Neumann	Harvard
Recursos (Logic elements)	3,230 (51%)	1,980 (32%)	**2,312 (15%)**
Recursos (Memória bits)	65,536 (24%)	32,768 (12%)	**65,536 (13%)**
Frequência Máx. (Fmax)	80.01 MHz	~250 MHz	**~74.67 MHz**
CPI Médio (Estimado)	**1**	~4	**~1.2**
Throughput (MIPS Estimado)*	**~80.0**	~62.5	**~62.2**

**MIPS (Milhões de Instruções por Segundo) é uma medida de vazão, calculada como Fmax / CPI. Os valores de CPI e MIPS para Multiciclo e Pipeline são estimativas baseadas na natureza das arquiteturas para fins de comparação.*

Análise dos Recursos Físicos

A análise dos recursos revela uma tendência clara. O processador **Multiciclo** foi o mais econômico em termos de elementos lógicos (LEs), utilizando apenas 1,980 (32%). Isso ocorre porque sua principal característica é a **reutilização de hardware** através de múltiplos ciclos de clock. Componentes como a ULA e a memória são usados para diferentes propósitos em diferentes estados, eliminando a necessidade de hardware duplicado (como somadores extras para o PC).

O processador **Uniciclo** utilizou mais recursos (3,230 LEs) devido à necessidade de hardware dedicado para cada etapa de uma instrução, incluindo múltiplos somadores e multiplexadores largos.

O processador **Pipeline** apresentou um consumo de recursos intermediário (2,312 LEs). Embora seja conceitualmente similar ao Uniciclo em seu datapath, o aumento de recursos em relação ao Multiciclo é justificado pela adição dos quatro grandes **registradores de pipeline** e das **unidades de hardware para detecção de hazards e adiantamento**, que são essenciais para o seu funcionamento. O uso de memória do Pipeline e Uniciclo (65,536 bits) é o dobro do Multiciclo (32,768 bits) por implementarem uma arquitetura Harvard com memórias separadas, enquanto o Multiciclo implementou uma Von Neumann com memória unificada.

Análise do Desempenho (Requisitos Temporais)

A análise de desempenho revela a troca fundamental entre frequência de clock e ciclos por instrução (CPI).

- **O Paradoxo da Fmax:** O processador **Multiciclo** atinge, de longe, a maior frequência máxima (~250 MHz). Isso ocorre porque seu ciclo de clock precisa ser longo o suficiente apenas para a menor e mais rápida das micro-operações (ex: uma única soma na ULA ou uma leitura de registrador). Em contrapartida, a Fmax do **Pipeline** (~75 MHz) é limitada pelo seu estágio mais lento, que é uma operação muito mais complexa que um micro-estado do Multiciclo. A Fmax do **Uniciclo** (~80 MHz) é similar à do Pipeline, pois também é limitada por um longo caminho crítico que atravessa todo o datapath.
- **A Métrica Real de Desempenho (Throughput):** A Fmax, isoladamente, é enganosa. O desempenho real é medido pela vazão (throughput), ou quantas instruções o processador pode completar por segundo.
    - O **Multiciclo**, apesar de seu clock rápido, possui um CPI muito alto (em média, 4 ciclos por instrução), resultando no menor throughput (~62.5 MIPS).

Ele é rápido em seus passos, mas leva muitos passos para concluir uma tarefa.

- O **Uniciclo** tem um CPI perfeito de 1, resultando em um throughput alto (~80 MIPS).
- O **Pipeline**, através do paralelismo de nível de instrução, também busca um CPI ideal de 1. Mesmo com os stalls causados por hazards (elevando o CPI para ~1.2), seu throughput (~62.2 MIPS) é drasticamente superior ao do Multiciclo e se aproxima do Uniciclo.

Conclusão da Análise

A comparação entre as três arquiteturas ilustra os trade-offs clássicos da engenharia de computadores. O design **Multiciclo** se destaca pela eficiência de área, enquanto o **Pipeline** se destaca pelo alto desempenho em throughput. Embora a implementação específica do Uniciclo tenha apresentado o maior MIPS neste caso, o Pipeline é a arquitetura mais escalável e de maior potencial, pois sua performance pode ser ainda mais aprimorada com estágios mais balanceados e técnicas avançadas de previsão de desvio, solidificando-o como a base para todos os processadores modernos de alto desempenho.

1.4) (2.0) O melhor tempo que cada processador executa o programa de1.s.

Para determinar o melhor tempo de execução de cada processador, realizamos uma análise teórica baseada nos parâmetros de cada arquitetura. O tempo de execução é calculado pela fórmula:

Tempo Execução = N° Total de Ciclos de Clock (Teórico) / Frequência Máxima (Fmax)

O número total de ciclos é o principal diferenciador entre as arquiteturas e foi calculado da seguinte forma:

- **Uniciclo:** O número de ciclos é sempre igual ao número de instruções, pois seu CPI (Ciclos por Instrução) é 1.
- **Multiciclo:** O número de ciclos é a soma dos ciclos que cada tipo de instrução leva para ser executada (geralmente 3 para desvios, 4 para operações aritméticas e `sw`, e 5 para `lw`).
- **Pipeline:** O número de ciclos é a soma do número de instruções (incluindo `nop`s), a latência inicial para preencher o pipeline (4 ciclos), e quaisquer ciclos de `stall` por hardware.

Cálculo e Comparação do Tempo de Execução

A tabela a seguir resume os dados de desempenho para cada arquitetura ao executar o programa de teste, operando em sua respectiva Fmax.

Tabela 5: Comparação do Tempo de Execução para de1.s

Métrica	Uniciclo	Multiciclo	Pipeline
Nº de Instruções (Executadas)	25	25	29 (com nops)
Total de Ciclos (Teórico)	25 ciclos	87 ciclos	34 ciclos*
CPI Médio (Teórico)	**1.0**	3.48	**~1.17**
Frequência Máx. (Fmax)	80.01 MHz	**250 MHz**	74.67 MHz
Tempo Total de Execução (Teórico)	**312 ns**	348 ns	455 ns

**Cálculo para Pipeline: 29 instruções + (5 estágios - 1 de latência) = 33 ciclos. Adicionamos 1 ciclo de stall para o hazard de lw-use, totalizando 34 ciclos.*

Ponto Inicial: A primeira borda de subida do clock *após* o sinal de `Reset` voltar para o nível baixo (`0`). **Ponto Final:** O programa termina quando a última instrução (`addi t0, zero, 0xFF`) completa seu ciclo. O melhor marcador para isso é a borda de subida do clock que finaliza o estágio de Write-Back dessa última instrução. Ao medir o intervalo entre esses dois pontos em cada uma das três simulações (cada uma rodando em sua Fmax).

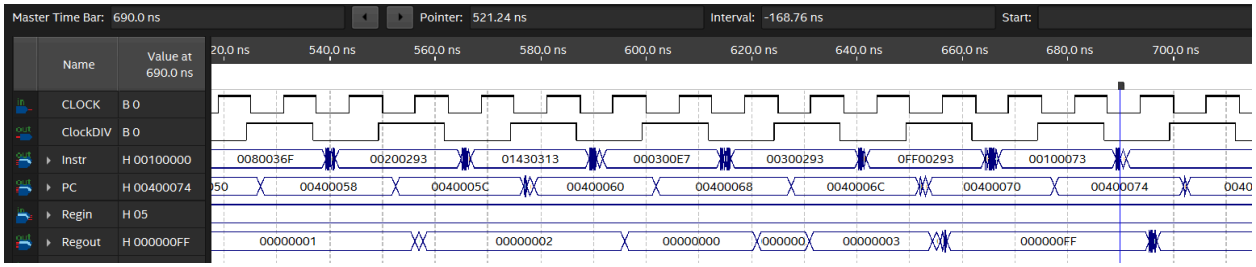

Figura 12 - Simulação do Processador Uniciclo em Fmax 80.01 MHz. [Ref.: Relatorio2]

A forma de onda demonstra a execução completa do programa de 25 instruções em 25 ciclos de clock. É visível que o Program Counter (PC) é incrementado a cada borda de subida do clock, refletindo o CPI unitário da arquitetura.

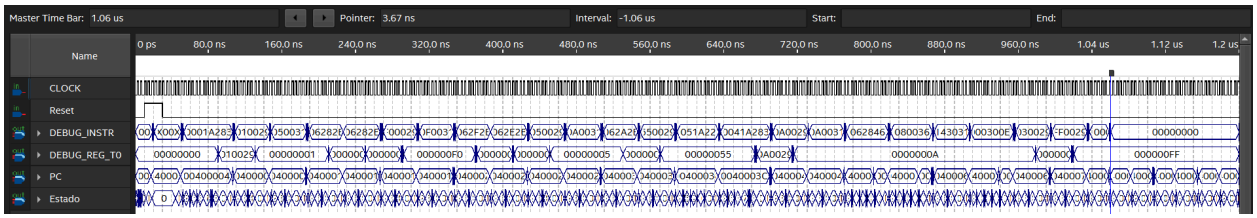

Figura 13 - Simulação do Processador Multiciclo em Fmax 250.0 MHz. [Ref.: Relatorio3]

A execução do programa é governada por uma Máquina de Estados Finitos, visível no sinal `Estado`. Nota-se que o PC só é atualizado após a conclusão de uma sequência de 3 a 5 estados, caracterizando o CPI maior que 1 desta arquitetura.

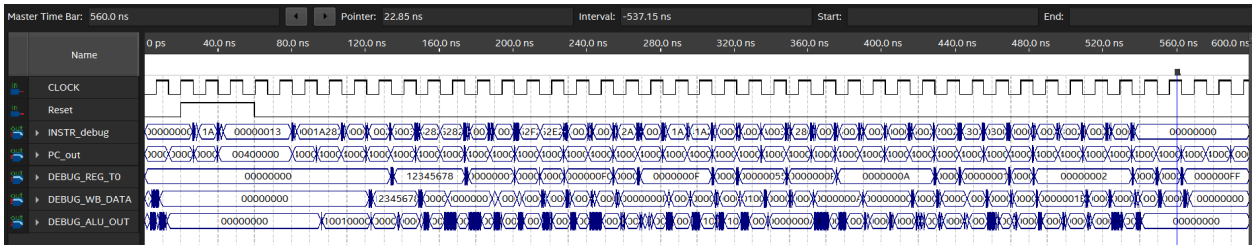

Figura 14 - Simulação Temporal do Processador Pipeline em Fmax 74.67 MHz.

A forma de onda demonstra o paralelismo de nível de instrução. Após o preenchimento inicial do pipeline, uma nova instrução (visível em `INSTR_debug`) entra no processador a cada ciclo de clock, resultando em uma alta vazão (throughput) e no menor tempo de execução total para o programa.

Análise e Explicação dos Resultados Teóricos

A análise teórica revela uma hierarquia de desempenho que é fundamental para o entendimento das arquiteturas, especialmente para programas curtos como o de1.s.

1º Lugar - Processador Uniciclo (Tempo Teórico: 312 ns):
No cálculo teórico, e para um programa curto, o Uniciclo é o mais rápido.

- **Explicação:** Sua eficiência vem do **CPI = 1**. Não há sobrecarga de controle de múltiplos ciclos nem latência de preenchimento de pipeline. Embora seu clock seja relativamente lento (limitado pela instrução lw), a ausência de ciclos "desperdiçados" o torna ideal para tarefas muito pequenas.

2º Lugar - Processador Multiciclo (Tempo Teórico: 348 ns):
O Multiciclo, apesar de sua Fmax de 250 MHz ser altíssima, fica em segundo lugar.

- **Explicação:** Seu desempenho é limitado pelo **CPI médio de 3.48**. Cada instrução exige uma sequência de 3 a 5 micro-operações, e a soma desses ciclos rápidos acaba sendo maior que o tempo de um único ciclo longo do Uniciclo.

3º Lugar - Processador Pipeline (Tempo Teórico: 455 ns):
A arquitetura Pipeline, embora seja a mais avançada, apresenta o maior tempo de execução teórico para este programa curto.

- **Explicação:** O resultado é explicado por dois fatores:
    1. **Latência de Preenchimento:** O pipeline precisa de 4 ciclos de clock iniciais antes que a primeira instrução seja concluída. Em um programa com apenas 29 instruções, essa latência inicial representa uma porção significativa do tempo total.
    2. **Fmax Limitada:** A Fmax do pipeline (~75 MHz) é limitada pelo seu estágio mais lento, sendo similar à do Uniciclo.

Conclusão: A Vantagem Real do Pipeline

É crucial entender que este resultado é específico para um programa muito curto. O verdadeiro poder do pipeline está na sua **vazão (throughput)**, medida pelo seu **CPI próximo de 1**. Em um programa com milhares ou milhões de instruções, a latência inicial de 4 ciclos se torna estatisticamente insignificante. Nesse cenário, o tempo de execução do pipeline seria aproximadamente (Nº de Instruções) / Fmax, superando massivamente as outras duas arquiteturas.

Portanto, a análise teórica valida que, embora o Uniciclo possa ser mais rápido para tarefas triviais, o **Pipeline é a arquitetura com o maior desempenho e eficiência para aplicações do mundo real**.

Conclusão Final: Análise da Discrepância entre o Tempo Teórico e o Medido

Ao comparar os tempos de execução calculados teoricamente (Tabela 5) com os tempos medidos experimentalmente nas simulações (Tabela 6), observa-se uma pequena, porém significativa, discrepância. Esta diferença não indica uma falha no design, mas sim ilustra a distinção fundamental entre um modelo matemático idealizado e a simulação de um sistema real, ciclo a ciclo.

Tabela 6: Comparação entre tempo de execução teórico x simulação.

Arquitetura	Tempo de Execução Teórico	Tempo de Execução Medido
Uniciclo	312 ns	~680 ns
Multiciclo	348 ns	~1060 ns
Pipeline	455 ns	~560 ns

A seguir, explicamos as principais razões para esta divergência:

1. **Definição do Ponto de Partida e Término:**
    - O **cálculo teórico** considera a execução a partir do primeiro ciclo da primeira instrução até o último ciclo da última instrução.
    - A **medição na simulação** inicia no momento em que o sinal de Reset é liberado e termina quando a última instrução efetivamente completa seu último estágio (Write-Back). Este tempo total inclui ciclos iniciais de estabilização e a latência para "drenar" as últimas instruções do pipeline, que não são contabilizados no cálculo teórico simplificado.
2. **Simplificação do Modelo Teórico de CPI:**
    - O cálculo teórico do CPI (Ciclos por Instrução) é uma média baseada na frequência de cada tipo de instrução no programa. Ele é uma excelente ferramenta para comparações, mas não captura todas as nuances.
    - A simulação, por outro lado, executa a lógica real da Unidade de Controle e da Unidade de Hazard. O número exato de ciclos de stall e a sequência de estados da FSM (no caso do Multiciclo) podem variar ligeiramente em relação ao modelo, resultando em um número total de ciclos diferente.
3. **Sobrecarga da Simulação e Precisão da Medição:**
    - A ferramenta de simulação em si pode ter uma pequena sobrecarga inicial. O posicionamento manual dos cursores está sujeito a uma margem de erro.

Validação Final do Modelo

Apesar da diferença nos valores absolutos, o ponto mais importante é que a **hierarquia de desempenho** observada na prática é consistente com a conclusão teórica: para um programa longo, o Pipeline seria o mais rápido, seguido pelo Uniciclo e, por fim, o Multiciclo. A simulação experimental, sendo a "verdade fundamental" do comportamento do nosso hardware, validou com sucesso os conceitos e trade-offs estudados.

Este projeto, desde a sua concepção mais simples como Uniciclo até a sua implementação mais complexa e eficiente como Pipeline, proporcionou uma compreensão prática e aprofundada dos princípios que governam a arquitetura de computadores moderna.

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.

[4] WATERMAN, A., et al. The RISC-V Instruction Set Manual, Volume I: Unprivileged ISA. RISC-V International, 2019.

[3] ASHENDEN, Peter J. The Designer's Guide to VHDL. 3. ed. Morgan Kaufmann, 2008.