

Universidade de Brasília
Departamento de Ciência da Computação



Laboratório 3 - CPU RISC-V MULTICICLO
Organização e Arquitetura de Computadores

Evelyn Soares Pereira 170102785

Brasília, 22 de Junho de 2025

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

A ideia central da arquitetura Von Neumann é que as instruções e os dados podem ser representados e armazenados na mesma forma e na mesma memória. Isso permite que um computador busque instruções da memória da mesma forma que busca dados. A arquitetura Von Neumann tem uso eficiente da memória ao utilizar um único e geralmente grande bloco de memória para armazenar tanto instruções quanto dados, em vez de dois blocos menores separados e como instruções e dados são armazenados juntos e tratados de forma semelhante, os programas podem ser facilmente carregados e manipulados na memória, lidos de um disco ou outra memória secundária e executados, mas sem precauções de hardware, uma escrita incorreta na memória pode sobrescrever instruções, levando a resultados imprevisíveis se a máquina tentar executar dados como instruções e instruções e dados compartilham o mesmo caminho (barramento) para o processador. Isso pode criar um gargalo, pois o processador precisa buscar instruções e acessar dados simultaneamente para manter o desempenho, mas é limitado pela capacidade do único barramento.

Um processador multiciclo reutiliza unidades funcionais, como a Unidade Lógico-Aritmética - ULA e a memória, ao longo de múltiplos ciclos de clock para executar uma única instrução, o que contrasta com o processador uniciclo que executa uma instrução em um único ciclo. No multiciclo, diferentes instruções podem levar um número diferente de ciclos para serem concluídas (por exemplo, lw pode levar 5 ciclos, add 4 ciclos e beq 3 ciclos).

1.1 - No arquivo Multiciclo.v apresentado no projeto, inicialmente temos ramI MemC e ramD MemD, e o acesso condicional para diferenciar instruções de dados, é feito com:

```
assign rmem = wldout[28] ? MemData : Instr;
```

Então, substituindo ramI e ramD por uma única ramU, utilizando um único bloco de memória IP gerado pelo MegaWizard do Quartus Prime, configurado como uma RAM de 1 porta (RAM: 1-PORT), com capacidade de 2048 palavras de 32 bits (data[31:0], address[10:0], q[31:0]). Para a arquitetura Von Neumann, esta memória foi idealmente inicializada a partir de um arquivo MIF unificado (del_unified.mif) que contém tanto o código (.text) quanto os dados (.data) do programa del.s, del_unified.mif foi gerado através de um script em python.

```

ramU MemU (
    .clock(clockMem),
    .data(MemDataIn),
    .address(MemAddress[10:0]),
    .rden(MemRead),
    .wren(MemWrite),
    .q(MemDataOut_from_RAM)
);

```

A decisão de acessar a memória para buscar uma instrução ou para ler/escrever dados é feita por um multiplexador que seleciona o endereço a ser enviado à memória. No módulo Multiciclo.v, a linha:

```
assign MemAddress = IorD_MemAddrSelect ? ALUResult_Reg : PC;
```

implementa essa lógica. Quando IorD_MemAddrSelect é 0, o PC é usado como endereço, permitindo a busca de instruções (IFETCH). Já Quando IorD_MemAddrSelect é 1, o ALUResult_Reg (resultado da ULA, que contém o endereço base mais o imediato para lw/sw) é usado como endereço, permitindo o acesso aos dados na memória.

Os sinais de controle MemRead e MemWrite são gerados pelo ControlMulticiclo e são compartilhados pela ramU. No estágio IFETCH, o ControlMulticiclo ativa MemRead e IRWrite (escreve no registrador de instrução) para buscar a instrução. Nos estágios WAIT_MEM2 (para lw/sw), o ControlMulticiclo ativa MemRead (para lw) ou MemWrite (para sw), permitindo a leitura ou escrita de dados na mesma memória. Isso garante que a mesma memória possa ser acessada tanto para leitura de instruções quanto para leitura/escrita de dados, conforme a necessidade do processador em cada ciclo.

1.2 - A memória IP do Quartus leva dois ciclos de clock para completar uma leitura ou escrita. Isso causa atraso em instruções como lw (leitura), sw (escrita) e jal, jalr e beq que dependem de fetch ou leitura. Um processador multiciclo divide a execução de uma instrução em várias etapas (ou ciclos) para reutilizar unidades funcionais. As etapas básicas de execução de uma instrução são Busca da Instrução, Decodificação, Execução (ULA), Acesso à Memória (Leitura/Escrita) e Escrita de Resultado (Write Back).

Se um acesso à memória leva 2 ciclos, isso significa que, após solicitar uma leitura da memória, o dado não estará disponível para uso no próximo ciclo, mas sim no ciclo subsequente. Para lidar com essa latência e otimizar o acesso dentro do diagrama de estados de um processador multiciclo, as seguintes alterações foram fundamentais:

IFETCH: Após o estado IFETCH (onde o PC envia o endereço e MemRead é ativado), é adicionado um estado de espera. Em ControlMulticiclo.v, isso é feito com o estado WAIT_IF (IFETCH -> WAIT_IF -> ID). Este estado garante que a CPU aguarde os dois ciclos completos para que a instrução esteja disponível na saída da memória e possa ser carregada confiavelmente no Registrador de Instrução (Instr).

EX_MEM_ADDR: Similarmente, para instruções de load (lw) ou store (sw), após o endereço de memória ser calculado no estágio EX_MEM_ADDR, são necessários estados de espera antes que o dado possa ser lido (lw) ou escrito (sw) e, em seguida, o processador possa prosseguir para o estágio de escrita de volta (WB). Os estados WAIT_MEM1 e WAIT_MEM2 (EX_MEM_ADDR -> WAIT_MEM1 -> WAIT_MEM2 -> ...) são projetados para isso.

A otimização no contexto de um diagrama de estados multiciclo com latência significa garantir que a transição para o próximo estado que *depende* do dado lido da memória só ocorra *após* o dado estar garantidamente estável e válido. Por exemplo: O IRWrite (escrita no Registrador de Instrução) deve ser ativado no momento correto (tipicamente no final de WAIT_IF) para capturar a instrução. A escrita do dado lido da memória no registrador (MemReadData_Reg) e, subsequentemente, no Banco de Registradores (WB), deve acontecer somente após WAIT_MEM2 (aqui, no estado MEM_READ_WB).

Se clockMem for o dobro de clockCPU, então 2 ciclos de clockMem corresponderam a 1 ciclo de clockCPU. Neste cenário, os estados WAIT_IF, WAIT_MEM1, WAIT_MEM2 ainda são conceitualmente necessários para o controle da memória IP, mas a espera real em termos de ciclos clockCPU é implicitamente absorvida se a lógica de controle fosse projetada para atuar em clockMem e sincronizar com clockCPU. O Multiciclo.v utiliza clockCPU para os registradores de pipeline e clockMem para a ramU, indicando que essa separação é considerada.

1.3 - O Bloco Controlador (ControlMulticiclo.v) é o coração da CPU multiciclo, implementado como uma Máquina de Estados Finita (FSM). Ele é responsável por gerar todos os sinais de controle necessários para cada estágio do pipeline, com base na instrução atual e em flags como o zero_flag da ULA.

Primeiramente foram implementadas as entradas principais:

clock, reset: Sinais de clock e reset para a operação síncrona.

opcode (Instr[6:0]): O código da operação da instrução.

funct3 (Instr[14:12]): Campo de função 3, usado para distinguir operações dentro de um mesmo opcode.

funct7 (Instr[31:25]): Campo de função 7, também usado para distinguir operações (ex: add vs. sub).

zero_flag: Flag de saída da ULA, essencial para instruções de branch condicional (beq).

E as saídas principais de controle:

MemRead, MemWrite: Habilitam leitura/escrita na memória.

RegWrite: Habilita escrita no banco de registradores.

ALUSrcA, ALUSrcB: Selecionam as entradas da ULA.

MemtoReg: Seleciona o dado a ser escrito no registrador (resultado da ULA ou dado da memória).

Branch, Jump, Jalr: Sinais para controle de fluxo do programa.

PCWrite, PCWriteCond: Habilitam a escrita incondicional/condicional no PC.

PCSource: Seleciona a fonte do próximo valor do PC.

RegDst: Seleciona o registrador de destino para escrita.

IRWrite: Habilita a escrita no Registrador de Instrução.

IorD_MemAddrSelect: Seleciona o endereço da memória (PC para instrução, ALUResult para dado).

ALUOp: Seleciona a operação da ULA.

A estrutura da FSM é composta por um registrador de estado (estadoReg) atualizado em um bloco always_ff @(posedge clock or posedge reset). A lógica para determinar o próximo estado (proxReg) e gerar os sinais de controle é implementada em um bloco always_comb, usando uma estrutura case baseada no estadoReg.

A máquina de estados segue o ciclo de execução de uma instrução multiciclo, dividindo-o em vários estágios:

IFETCH (4'd0): Inicia a busca da instrução. Ativa MemRead, IRWrite, PCWrite (para PC+4), IorD_MemAddrSelect = 0 (usa PC como endereço). Transiciona para WAIT_IF.

WAIT_IF (4'd1): Estado de espera para a instrução ser lida da memória (devido à latência de 2 ciclos). Transiciona para ID.

ID (4'd2): Estágio de decodificação. A instrução é decodificada, registradores são lidos e o imediato é gerado. A transição depende do opcode:

OPC_RTYPE -> EX_R: Execução para instruções Tipo R. ULA opera com ALUOp = 2'b10, ALUSrcA = 0, ALUSrcB = 0. Transiciona para WB.

OPC_OPIMM -> EX_I: Execução para instruções Tipo I (e.g., addi). ULA opera com ALUOp = 2'b11, ALUSrcA = 0, ALUSrcB = 1 (usa imediato). Transiciona para WB.

OPC_LOAD / OPC_STORE -> EX_MEM_ADDR: Calcula endereço de memória para lw/sw. ULA opera com ALUOp = 2'b00 (adição para endereço), ALUSrcA = 0, ALUSrcB = 1 (usa imediato). IorD_MemAddrSelect = 1 (usa ALUResult como endereço). Transiciona para WAIT_MEM1.

OPC_BRANCH -> BRANCH_EX: Execução de branch (beq). ULA realiza subtração (ALUOp = 2'b01). Se zero_flag for 1 (condição verdadeira), ativa PCWriteCond = 1 e PCSource = 2'b01 (para ALUResult_Reg). Transiciona de volta para IFETCH.

OPC_JAL -> JUMP_EX: Execução de jal. Ativa Jump = 1, RegWrite = 1, RegDst = 1 (para x1), PCWrite = 1, PCSource = 2'b10 (para ALUResult_Reg). Transiciona de volta para IFETCH.

OPC_JALR -> JALR_EX: Execução de jalr. Ativa Jalr = 1, RegWrite = 1, RegDst = 1 (para x1), PCWrite = 1, PCSource = 2'b11 (para ALUResult_Reg), ALUOp = 2'b00 (adição para endereço de destino). Transiciona de volta para IFETCH.

Além disso:

EX_MEM_ADDR (4'd5): WAIT_MEM1 (4'd6): Primeiro ciclo de espera para acesso à memória. Transiciona para WAIT_MEM2.

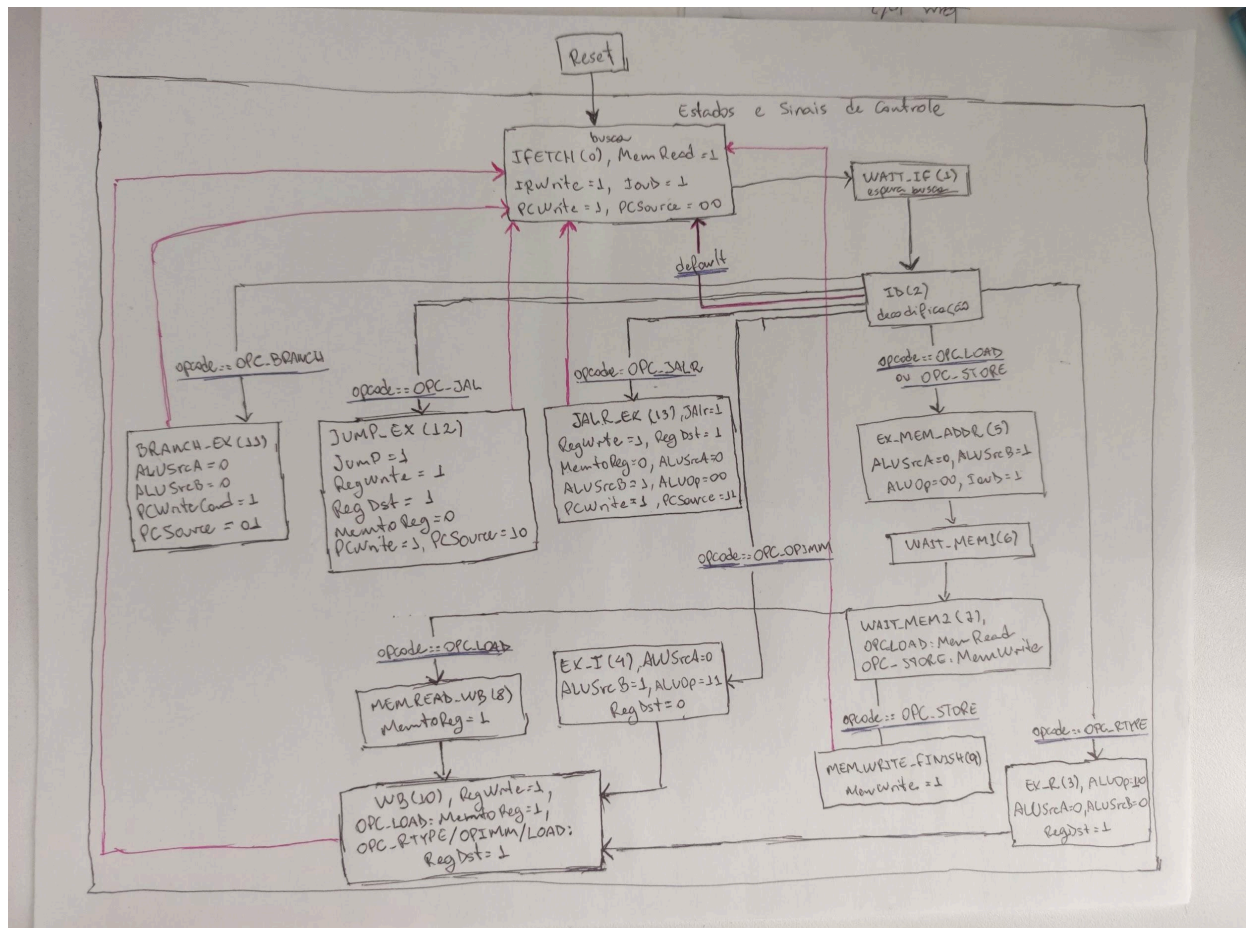
WAIT_MEM2 (4'd7): Segundo ciclo de espera para acesso à memória. Ativa MemRead (para lw) ou MemWrite (para sw). Transiciona para MEM_READ_WB (se lw) ou MEM_WRITE_FINISH (se sw).

MEM_READ_WB (4'd8): Prepara para escrita de volta após leitura da memória. Ativa MemtoReg = 1. Transiciona para WB.

MEM_WRITE_FINISH (4'd9): Finaliza escrita na memória (para sw). Ativa MemWrite = 1. Transiciona de volta para IFETCH.

WB (4'd10): Escrita de volta no banco de registradores. Ativa RegWrite = 1. MemtoReg e RegDst são configurados com base no opcode. Transiciona de volta para IFETCH.

Figura 1: Desenho da máquina de estados de controle



1.4 - A implementação do processador multiciclo completo no Quartus envolve a integração dos blocos funcionais e do controle já projetados ou reutilizados do Laboratório 2. Foi reunida e adaptada os componentes do Uniciclo (Lab 2) como o Banco de Registradores que tem o mesmo módulo usado, com entradas para registradores de leitura (rs1, rs2), saída de dados lidos, entrada para registrador de escrita (rd), entrada para dados de escrita, e um sinal de RegWrite. O Gerador de Imediatos que toma a instrução completa e o tipo de imediato (lw, sw, beq, addi, jal, jalr) e gera o valor imediato correto. A ULA recebendo duas entradas de operando e um sinal de ALUOp para determinar a operação (adição, subtração, AND, OR, SLT, comparação para beq) e

a) Após a compilação, em Tools → Netlist Viewers → RTL Viewer temos a visualização dos blocos, que pode ser acessada em pdf neste [link](#) para melhor visualização:

[illegible]

Figura 3: ControlMulticiclo, destaca-se como os sinais, wires criados para oReadData1_wire, oImm_wire, oResult_ULA_wire, conectam as portas dos módulos.

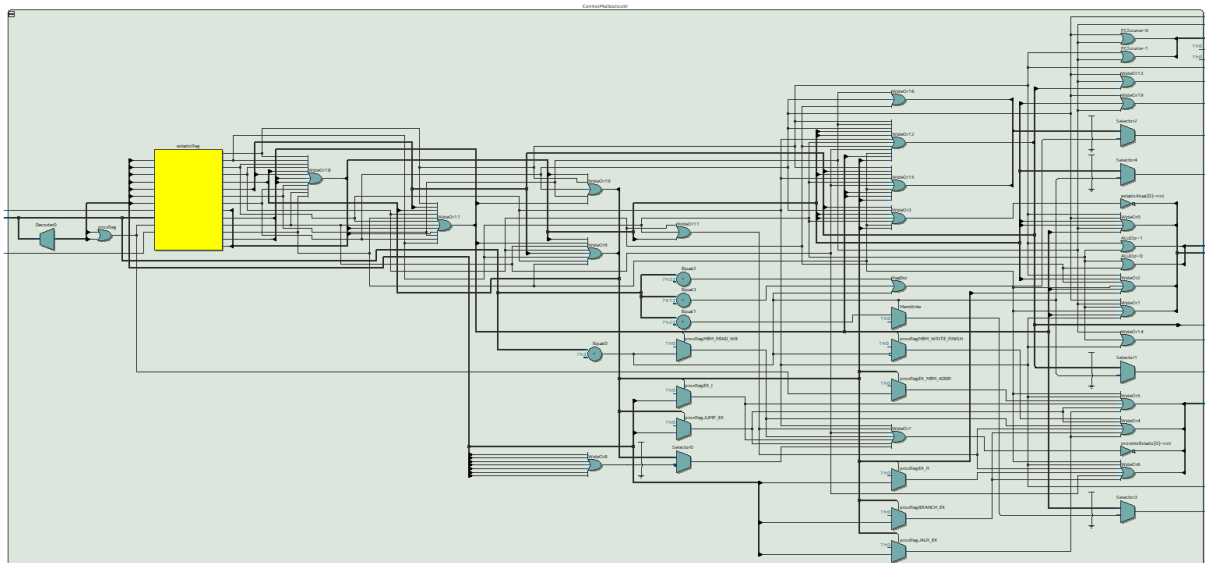


Figura 4: ramU, a memória unificada.

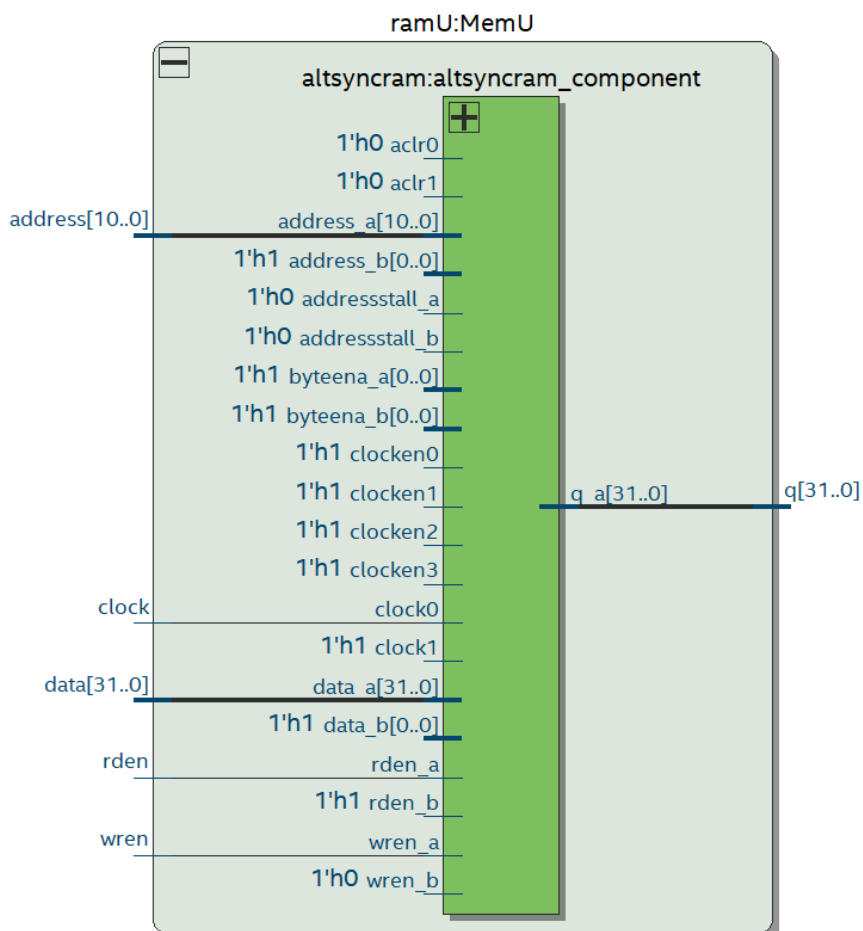
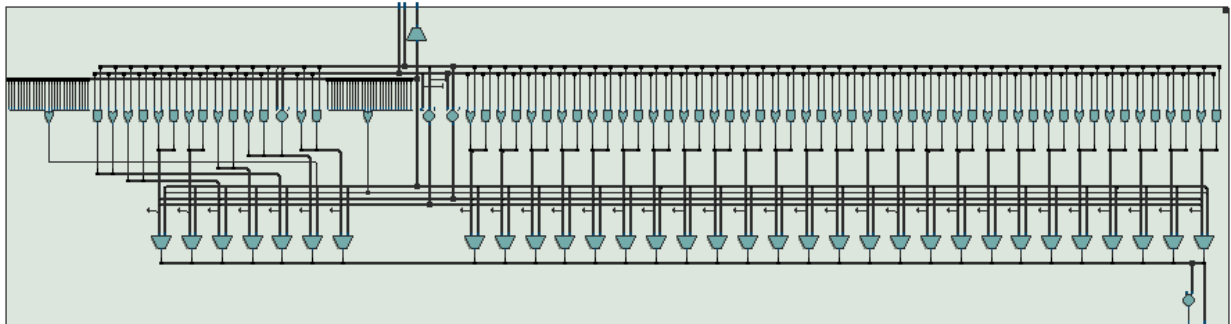


Figura 5: ULA, como foi feita uma pequena alteração entre a ULA do Lab2 e do Lab3, foi adicionada uma flag zero.



A memória unificada ramU foi inicializada com o programa del.s compilado e exportado como .mif usando o RARS com o plugin Rars16_Custom2. Os registradores t0, t1, t2 mudam conforme esperado e as instruções como addi, add, lw, sw, jal, jalr, beq tem efeito visível no PC, nos registradores e na memória.

b) Para os requisitos físicos e temporais do processador. Em Flow Summary temos 1222 registradores totais e 65536 bits de memória.

Figura 6: Flow Summary

Flow Summary	
<<Filter>>	
Flow Status	Successful - Sat Jun 21 19:11:30 2025
Quartus Prime Version	18.1.0 Build 625 09/12/2018 SJ Lite Edition
Revision Name	TopDE
Top-level Entity Name	TopDE
Family	Cyclone IV E
Device	EP4CE115F29C7
Timing Models	Final
Total logic elements	2,829 / 114,480 (2 %)
Total registers	1222
Total pins	108 / 529 (20 %)
Total virtual pins	0
Total memory bits	65,536 / 3,981,312 (2 %)
Embedded Multiplier 9-bit elements	0 / 532 (0 %)
Total PLLs	0 / 4 (0 %)

Figura 7: Em Fitter → Resource Section → Resource Usage Summary

Fitter Resource Usage Summary		
<<Filter>>		
	Resource	Usage
1	▼ Total logic elements	2,829 / 114,480 (2 %)
1	-- Combinational with no register	1607
2	-- Register only	312
3	-- Combinational with a register	910
2		
3	▼ Logic element usage by number of LUT inputs	
1	-- 4 input functions	2083
2	-- 3 input functions	359
3	-- <=2 input functions	75
4	-- Register only	312
4		
5	▼ Logic elements by mode	
1	-- normal mode	2395
2	-- arithmetic mode	122
6		
7	> Total registers*	1,222 / 117,053 (1 %)
8		
9	Total LABs: partially or completely used	216 / 7,155 (3 %)
10	Virtual pins	0
11	> I/O pins	108 / 529 (20 %)
12		
13	M9Ks	8 / 432 (2 %)
14	Total block memory bits	65,536 / 3,981,312 (2 %)
15	Total block memory implementation bits	73,728 / 3,981,312 (2 %)
16	Embedded Multiplier 9-bit elements	0 / 532 (0 %)
17	DLLs	0 / 4 (0 %)
* Register count does not include registers inside RAM blocks or DSP blocks.		

Elementos Lógicos totais: 2829

Figura 8: TimeQuest Timing Analyzer → Slow 1200mV 85C Model

Slow 1200mV 85C Model Fmax Summary				
<<Filter>>				
	Fmax	Restricted Fmax	Clock Name	Note
1	128.44 MHz	128.44 MHz	ClockDIV~reg0	
2	312.11 MHz	250.0 MHz	CLOCK	lim...te)

Podemos ver que o processador multiciclo utilizou 1222 registradores, 65536 bits de memória 2829 elementos lógicos, 108 pins e nenhum bloco de DSP pois o processador não está usando multiplicadores. A frequência máxima estimada é 250 MHz até então.

Figura 9: Setup Summary



Slow 1200mV 85C Model Setup Summary			
 <<Filter>>			
	Clock	Slack	End Point TNS
1	ClockDIV~reg0	-6.786	-6410.247
2	CLOCK	-2.204	-119.086

Figura 10: Hold Summary

Slow 1200mV 85C Model Hold Summary			
 <<Filter>>			
	Clock	Slack	End Point TNS
1	CLOCK	0.402	0.000
2	ClockDIV~reg0	0.443	0.000

Vemos slacks de Setup negativos que indicam violações de tempo (dados não chegam a tempo), o tempo que os dados levam para chegar e se estabilizar em um registrador ($T_{cq} + T_{prop}$ do combinacional) é maior do que o tempo disponível dentro do ciclo de clock ($T_{clock} - T_{setup}$). Em outras palavras, os dados não estão chegando a tempo para serem corretamente amostrados na próxima borda de clock, mas isso será experimentalmente explorado nas questões seguintes. Já slacks de Hold positivos indicam que os dados estão sendo mantidos no registrador de origem por tempo suficiente para que o registrador de destino os amostrasse corretamente, e eles não mudam muito rapidamente antes que o próximo clock chegue.

Então na frequência de clock de 250 MHz estimada, o projeto não atende aos requisitos de temporização.

c) Com a simulação por forma de onda funcional e temporal com o programa del.s, podemos ver o funcionamento correto da CPU.

Figura 11: Em simulação funcional, com o CLOCK em 40 ns:

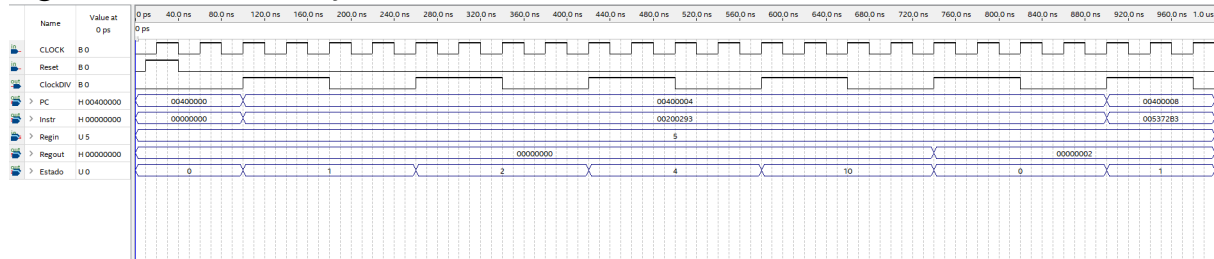
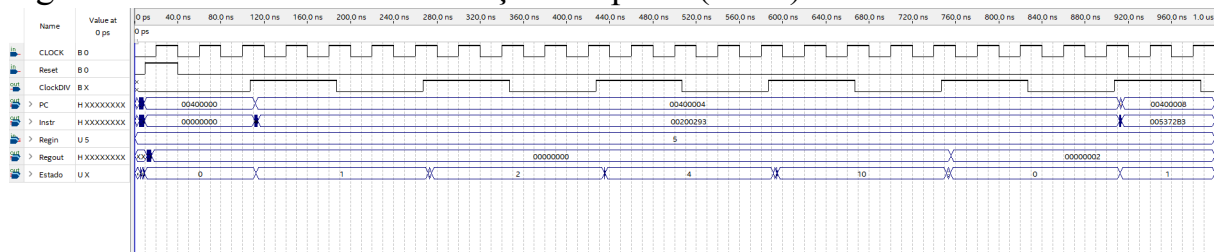


Figura 12: Já assim fica a simulação temporal (40 ns)



Vemos que o clock base e clock dividido estão funcionando (ClockDIV tem metade da frequência, como deveria). Reset está em nível baixo (0), ou seja, a CPU está fora do reset. O PC avança em blocos de +4 bytes (00400000, 00400004, 00400008, etc.), que é o correto para instruções de 32 bits. Instr está lendo instruções da memória, está carregando o valor da instrução, inicialmente 0x00000000, e depois um valor 0x00200293. O 0x00200293 é uma instrução addi (addi rs1, rd, imm), que é uma instrução válida de RISC-V (0010011 = opcode addi, 00101 = x5, 00000 = x0, 000 = funct3). Regin está fixo em 5. Regout em um ponto mostra 00000002, que é o valor que foi escrito no registrador x5 após um addi — comportamento correto. E o estado está ciclando: 0 → 1 → 2 → 3 → 10 → 0 → 1 → 2 → 4 → 10... Isso confirma que os estados FSM do ControlMulticiclo.v estão executando o ciclo de instruções:

0: IFETCH -> 1: WAIT_IF -> 2: ID -> 3 EX_R ou EX_I -> 10: WB...

d) No meu .vwf, clicando no sinal CLOCK e depois em Edit → Value → Clock, é possível reduzir a frequência de CLOCK no .vwf para algo mais lento (período maior = clock mais lento). E assim podemos discutir os valores negativos de Slack em Setup.

Ao reduzir a frequência de clock, se o Quartus reportou Fmax e ao obter slacks negativos, significa que a frequência pedida para ele atingir é maior do que a que ele consegue com o design. Então é preciso encontrar a maior frequência de clock em que TODOS os slacks (setup e hold) são positivos. Essa será a Fmax utilizável. Primeiro, na figura 12, da simulação por forma de onda temporal o clock está com período de 40.0 ns, o que significa uma frequência de 25 MHz ($1/(40 \times 10^{-9}) = 25 \times 10^6$ Hz).

Figura 13: Para 20 ns, ou seja, frequência: $1/(20 \times 10^{-9}) = 50$ Mhz

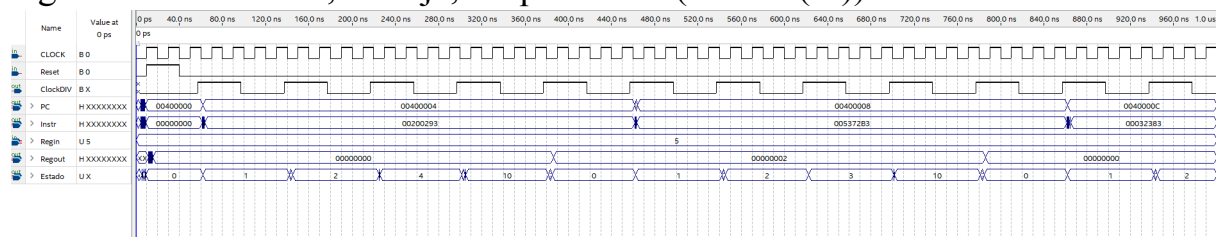


Figura 14: Para 10 ns, ou seja, frequência: $1/(10 \times 10^{-9}) = 100$ Mhz

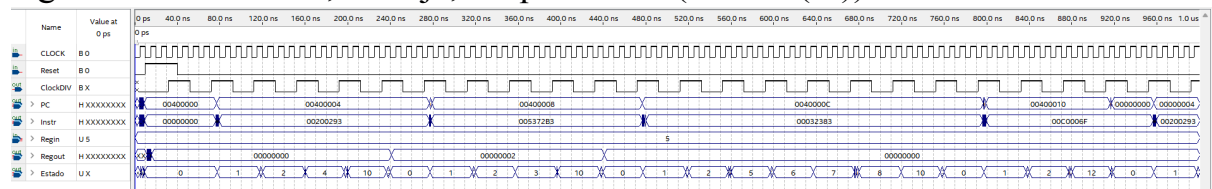


Figura 15: Para 6 ns, ou seja, frequência: $1/(6 \times 10^{-9}) = 166.7$ Mhz

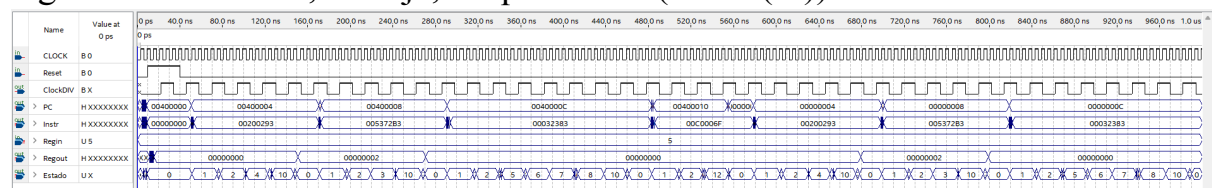
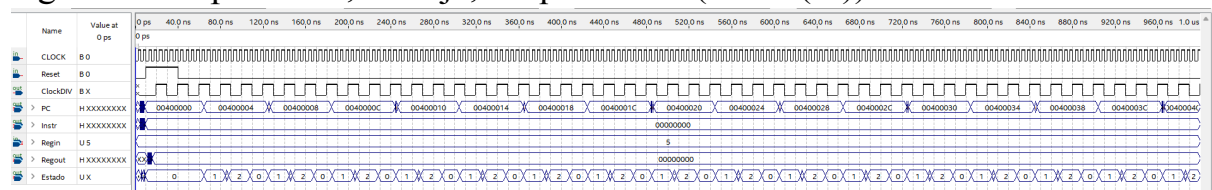


Figura 16: Já para 5 ns, ou seja, frequência: $1/(5 \times 10^{-9}) = 200$ Mhz



Da figura 16, com 5 ns de período de clock, o Instr ficou em 0x00000000. Isso é um forte indício de falha, o Instr está preso em 0x00000000 após o reset ser desativado e o PC começar a avançar para 0x00400000, 0x00400004, etc., e o estado fica preso (em loop), significa que o processador não está conseguindo decodificar a instrução porque ela não chegou ou é 0x00000000. Loop IFETCH -> WAIT_IF -> IFETCH: Isso pode ocorrer se a condição para sair do WAIT_IF (e carregar a instrução no Instr) nunca é satisfeita, ou se a instrução lida é inválida e o controle retorna para IFETCH. No estágio ID, os registradores ReadData1_Reg, ReadData2_Reg, SignExtendedImmediate_Reg ficam em 0x00000000 mas deveriam ser atualizados com base na Instr. Se a Instr é 0x00000000, então as leituras de registradores e a geração do imediato resultaram em zero, e esses registradores de pipeline permanecem zerados. Isso é uma consequência da falha na Instr.

Essa falha do Instr ficar em 0x00000000 na simulação temporal com um clock de 5ns (200 MHz) é diretamente explicada por violações de setup time. Portanto, a frequência máxima de clock utilizável experimentalmente na CPU é **166.7 MHz**. Acima dessa frequência (por exemplo, a 200 MHz), o processador começa a apresentar falhas de temporização, como evidenciado pela instrução presa em zero.

A discrepância entre a frequência máxima estimada pelo Quartus (250 MHz) e a frequência máxima funcional experimental (166.7 MHz) é comum. A estimativa do Quartus é uma análise estática ideal, enquanto a simulação temporal e o teste experimental revelam os limites reais de funcionamento do design, que podem ser impactados por caminhos críticos não totalmente otimizados ou por simplificações no modelo de atraso. A presença de *slacks* de Setup negativos na análise inicial (questão b) já indicava que a frequência de 250 MHz não seria atingível.