Laboratório 05 - Grupo 09
- CPU femtoRISC-V PIPELINE –

Antônio Vinicius de Moura Rodrigues 19/0084502,
Gabriel Pinheiro da Conceição 19/0133724,
Leandro de Sousa Monteiro 17/0060454

[1] Universidade de Brasília - Instituto de Ciências Exatas
Dep. Ciência da Computação - CIC0099 - Organização e Arquitetura de Computadores
2020.2 - Turma A - Professor Marcus Vinicius Lamar
Prédio CIC/EST - Campus Universitário Darcy Ribeiro
Asa Norte 70919-970 Brasília, DF

1. ULA

A ULA utilizada neste relatório necessitou processar operações de até 32 bits. O Deeds oferece uma opção de ULA já implementada de 16 bits, desse modo precisou-se fazer algumas adaptações de maneira que ela pudesse se adequar ao projeto.

Foram então utilizadas duas ULAs de 16 bits e as modificações adicionais podem ser observadas abaixo:

1. As entradas que definem a operação a ser realizada pelo componente foram compartilhadas entre as duas.
2. As entradas dos operandos A e B responsáveis por fornecer à ULA os argumentos para cálculo foram divididos em dois BUSes de 16 bits cada e distribuidos entre as duas entradas para cada argumento de cada ULA.
3. A saída CO da primeira ULA, ou seja, a ULA que opera com os primeiros 15 bits de cada argumento foi conectada na entrada CI da segunda ULA de modo que as operações feitas possam ter continuidade com base no *carry out* da primeira ULA.
4. A saída das duas ULAs foi concatenada de modo a compor o resultado final da operação com 32 bits

O resultado final da ULA pode ser visto na figura 1 que representa o circuito interno do CBE utilizado no processador *femto*RISC-V.

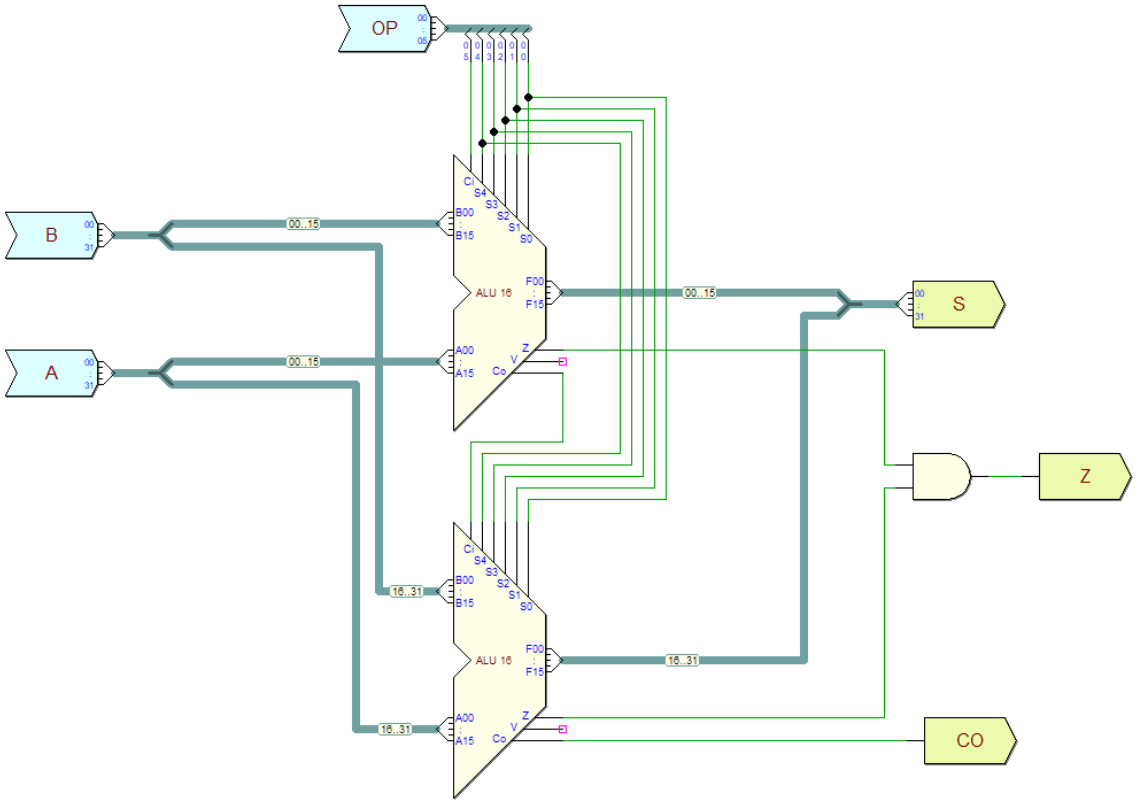

Figure 1. Circuito interno do CBE da ULA

2. Memória de Instruções

Para a memória de instruções do processador *femto*RISC-V foi necessário um componente que armazenasse os 32 bits do código de linguagem de máquina conforme os padrões da ISA RISC-V. O componente escolhido foi uma memória ROM simples que segundo [Arruda 2011] possui como função: "[...] oferecer dados apenas para leitura. Normalmente, a ROM é utilizada para armazenar firmwares, pequenos softwares que funcionam apenas no hardware para o qual foram desenvolvidos e que controlam as funções mais básicas do dispositivo.

Na ROM de uma calculadora, por exemplo, podemos encontrar as rotinas matemáticas que o estudante pode realizar ao usá-la. Já no caso de celulares, normalmente as ROMS carregam o sistema operacional e os softwares básicos do aparelho."

Escolhido o componente, foi necessário adaptá-lo de modo a suportar o armazenamento de 32 bits ante o padrão máximo das memórias ROM fornecidas pelo Deeds (16 bits). As adaptações seguiram a mesma lógica da ULA da seção 1, ou seja, dividir a entrada em dois BUSes de 16 bits cada e concatenar a saída em um BUS só de 32 bits.

Por questões de simplificação do processo de construção do processador a memória de instruções não foi implementada em um CBE dedicado e o circuito desenvolvido foi implementado diretamente no arquivo pbs do processador. O resultado final da memória de instruções pode ser visto na figura 2.

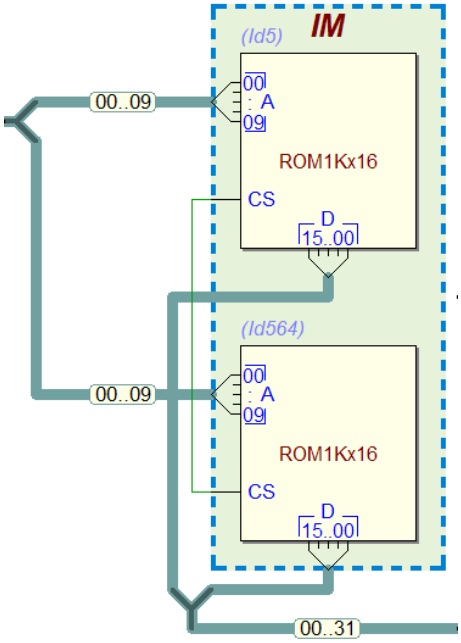

Figure 2. Memória de Instruções

3. Memória de Dados

A memória de dados precisou ser um componente versátil que pudesse fornecer os dados e também regravá-los caso necessário, além de precisar ser mais rápida. Uma memória ROM, portanto, não atende aos requisitos necessários e por isso o componente escolhido para realizar a função foi uma memória RAM, que segundo [Alves 2014] "[...] é um tipo de memória volátil que serve para rodar aplicações depois que o computador já está ligado, e cujas informações são perdidas depois do desligamento da máquina."

Assim como nos outros componentes, o Deeds fornece soluções prontas, no entanto elas não possuem a quantidade de bits necessárias a construção desse processador. Com a memória RAM não foi diferente e as adaptações padrões precisaram ser feitas, ou seja dividir os dados de entrada entre duas memórias iguais e concatenar as saídas de modo a obter um dado de 32 bits. A entrada de endereçamento da memória foi fornecida igualmente para as duas RAMs.

O resultado final do circuito interno do CBE utilizado pode ser visto na figura 10.

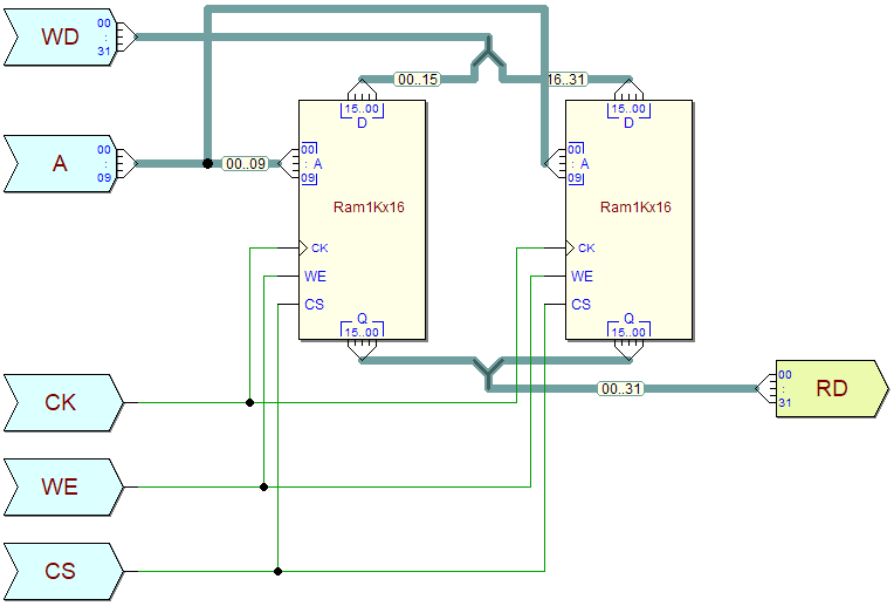

Figure 3. Memória de Dados

4. Banco de Registradores

Para a construção do banco de registradores foi usada como base a figura referência de [Pattersson and Henessy 2021] que idealiza um banco de registradores com 32 registradores de 32 bits cada. Além disso, o banco possui duas entradas de endereçamento utilizadas para seleção dos registradores, uma entrada de endereçamento para seleção do registrador de destino das operações realizadas com outros registradores, uma entrada de habilitação da escrita nos registradores e uma entrada ligada diretamente à ULA e à RAM que permite a transferência de argumentos recentemente calculados ou armazenados na RAM diretamente ao banco. A idealização de [Pattersson and Henessy 2021] inclui ainda duas saídas de dados, o que permite a leitura de dois registradores de forma simultânea. Um desenho esquemático do banco de registradores implementado pode ser visto na figura 4 também retirada de [Pattersson and Henessy 2021].

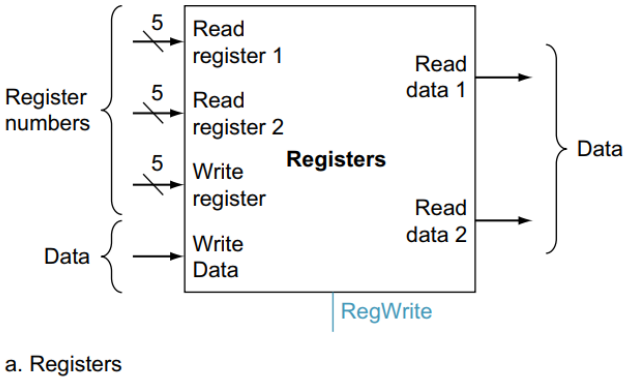

Figure 4. Desenho esquemático do banco de registadores

Para a seleção dos registradores a seguinte lógica foi aplicada: Foram utilizados 8 sets de 2 mux ($8 \rightarrow 1$) cada e 6 sets de 2 mux ($2 \rightarrow 1$) cada. Todos os mux utilizados possuem 16 bits e por isso a necessidade de se usar 2 mux para cada seleção desejada. Os mux foram dispostos de modo a fazer uma filtragem por setor de registradores.

Os setores foram divididos da seguinte forma:

1. Dois setores principais divididos entre esquerda e direita, cada um com 16 registradores.

2. Dentre de cada setor principal os registradores foram divididos em mais dois setores: Superior e Inferior, cada um com 8 registradores.

3. Dentro dos setores superior e inferior cada registrador pode ser selecionado por um número de 0 a 7.

Os 3 bits de seleção menos significativos são os responsáveis por filtrar os registradores do número correspondente em cada um dos 4 setores superior e inferior dos lados esquerdo e direito do banco de registradores. O próximo bit de seleção é o responsável por escolher entre o setor superior ou o inferior de cada lado e o último bit (MSB) seleciona o lado esquerdo ou o direito.

O número 00001, por exemplo, seleciona o registrador 1 da seguinte forma:

1. 0 seleciona o setor esquerdo ou o direito. Nesse caso será o setor esquerdo.

2. 0 seleciona o setor superior ou o inferior de cada lado. Nesse caso será o setor superior.

3. 001 = 3 bits menos significativos, selecionam o registrador 1 em cada um dos 4 setores superior e inferior.

Com essa lógica foi possível então implementar o banco de registradores no Deeds. O resultado final pode ser visto na figura 5. Recomenda-se que para uma visualização mais detalhada do banco de registradores, o arquivo cbe correpondente seja consultado. Na imagem os setores foram separados por cor de forma a facilitar a visualização da lógica de seleção.

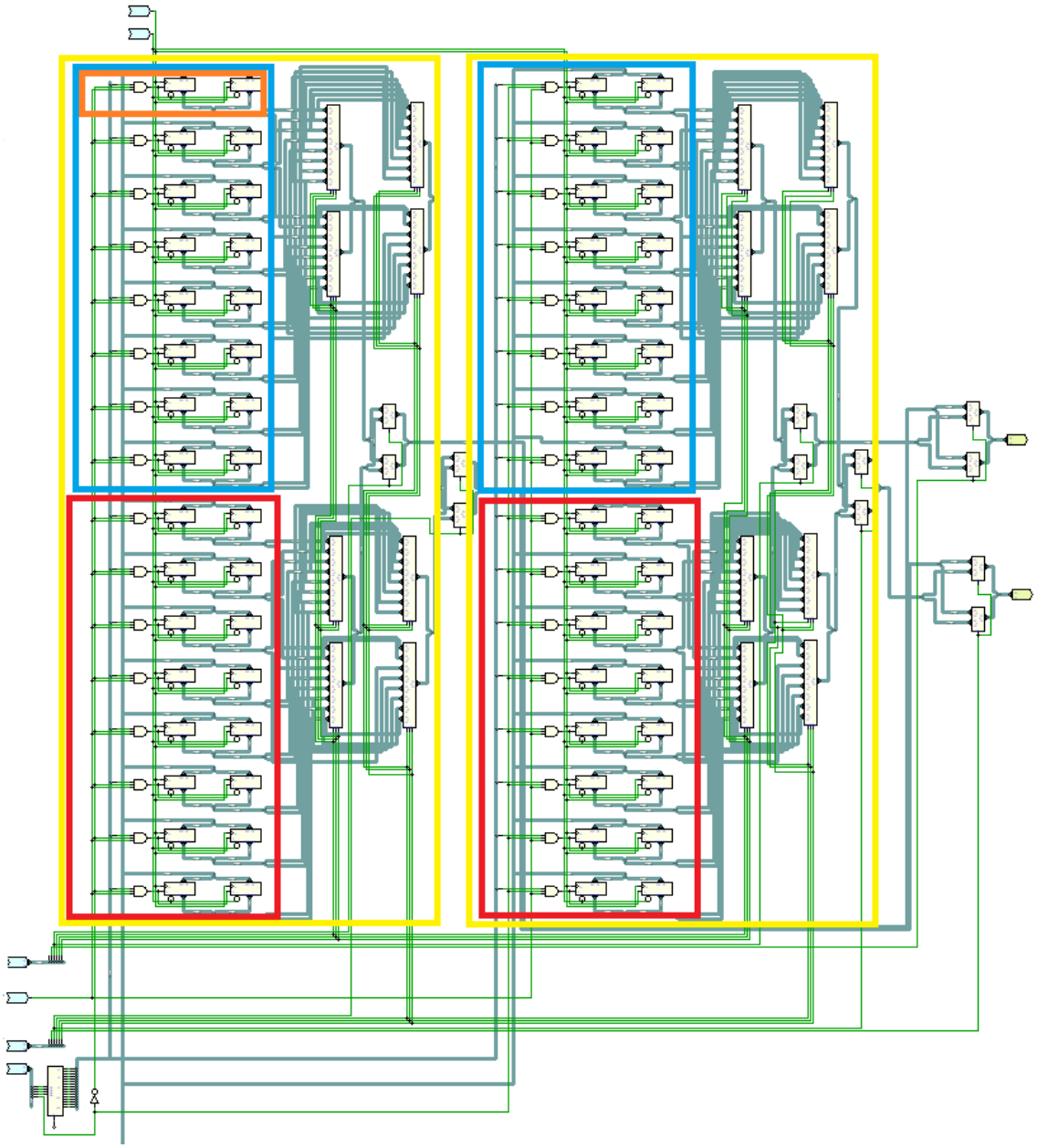

Figure 5. Banco de Registradores

- Amarelo: setores esquerdo e direito
- Azul: setores superiores
- Vermelho: setores inferiores
- Laranja: registrador individual de cada setor selecionado por 3 bits

5. Datapath e bloco de controle

O datapath utilizado para a construção do processador *femto*RISC-V foi retirado de [Pattersson and Henessy 2021] e seguido à risca de modo que se assemelhasse de maneira mais fiel possível à versão do autor. No entanto, algumas modificação foram feitas de modo a implementar as instruções sll e jal. A figura 6 apresenta um desenho esquemático do caminho de dados padrão sem modificações utilizado.

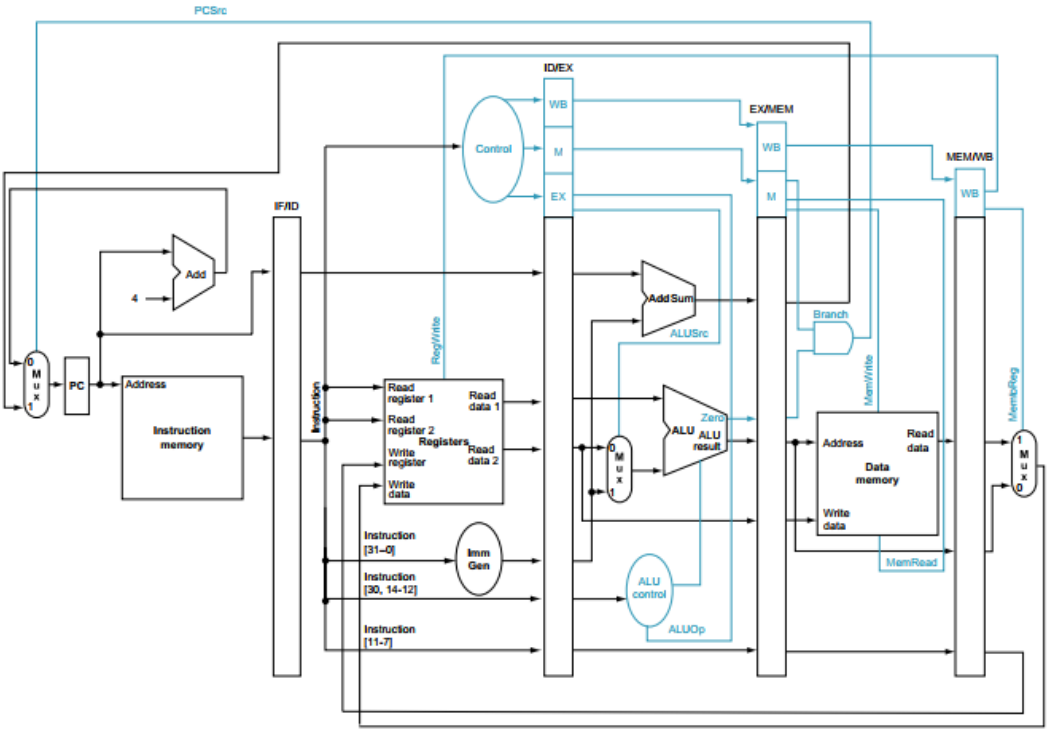

Figure 6. Caminho de Dados

5.1. Gerador de Imediato

Na geração do imediato de cada instrução foi utilizado um circuito decodificador que atribui um número a cada opcode da instrução submetida ao circuito. O número foi então utilizado para fazer a seleção de dois mux ($8 \rightarrow 1$) que contem em cada entrada o imediato gerado para cada instrução. Os imediatos foram gerados com base na tabela da figura 7 retirada de [Pattersson and Henessy 2021] manualmente, ou seja, reorganizando bit a bit e completando o restante dos bits com o valor 0 para que o imediato possua os devidos 32 bits necessários à configuração do processador.

CORE INSTRUCTION FORMATS

	31 27	26 25	24 20	19 15	14 12	11 7	6 0
R	funct7		rs2	rs1	funct3	rd	opcode
I	imm[11:0]			rs1	funct3	rd	opcode
S	imm[11:5]		rs2	rs1	funct3	imm[4:0]	opcode
SB	imm[12|10:5]		rs2	rs1	funct3	imm[4:1|11]	opcode
U	imm[31:12]					rd	opcode
UJ	imm[20|10:1|11|19:12]					rd	opcode

Figure 7. Formatos das instruções da ISA RISC-V

A figura 8 apresenta o resultado final do circuito interno do CBE implementado para a geração de imediatos. Recomenda-se o exame do arquivo CBE original para a observação de mais detalhes quanto ao circuito em questão.

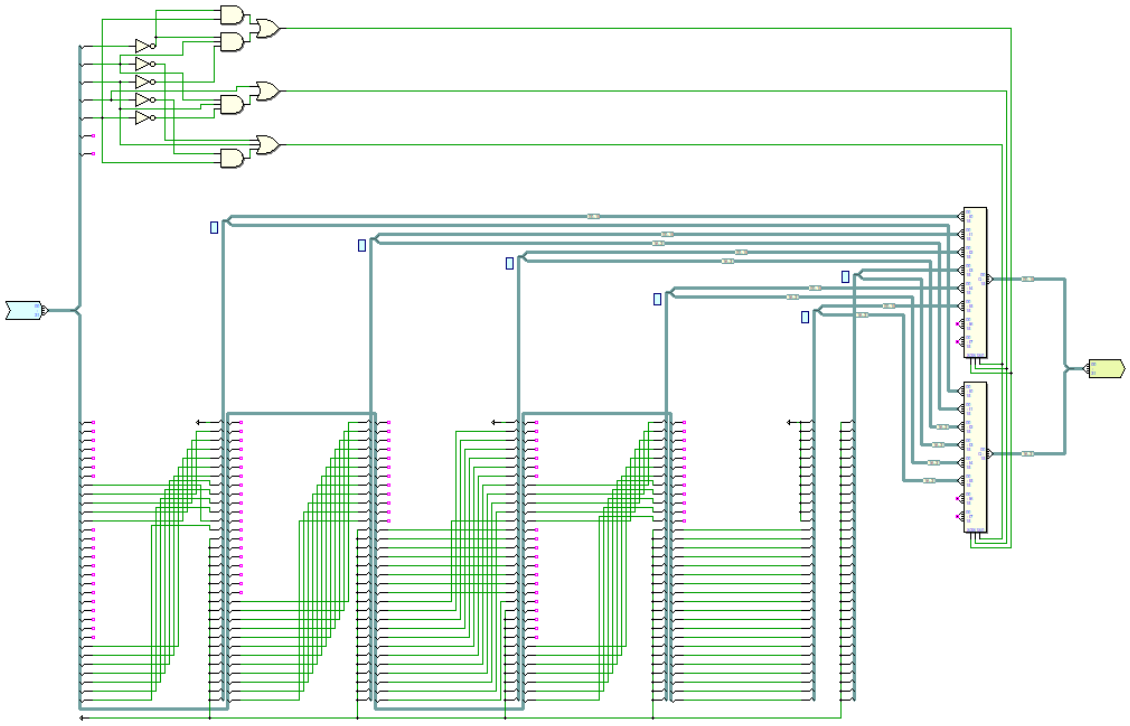

Figure 8. Circuito interno do CBE gerador de imediato

5.2. Bloco de Controle

Para a implementação do bloco de controle foi desenvolvido um decoder que fornece um número 3 bits correspondente a cada tipo de instrução a partir do opcode. Esses 3 bits são conectados em um outro encoder responsável por filtrar se a instrução recebida necessita ou não da análise dos campos de funct3 e funct7. Se a entrada 3 do encoder for igual a 1 ela habilita um novo encoder que analisará o funct3 e para a análise do funct7 um mux ($2 \rightarrow 1$) utilizando a saída 0 do encoder que analisa o funct3 para seleção foi suficiente visto que só há um bit de diferença entre as duas únicas instruções que necessitam da análise de funct3 e funct7.

A figura 9 apresenta de forma gráfica o circuito interno do CBE utilizado como bloco de controle nesse relatório. Recomenda-se que para uma melhor visualização do circuito, o arquivo CBE correspondente seja consultado.

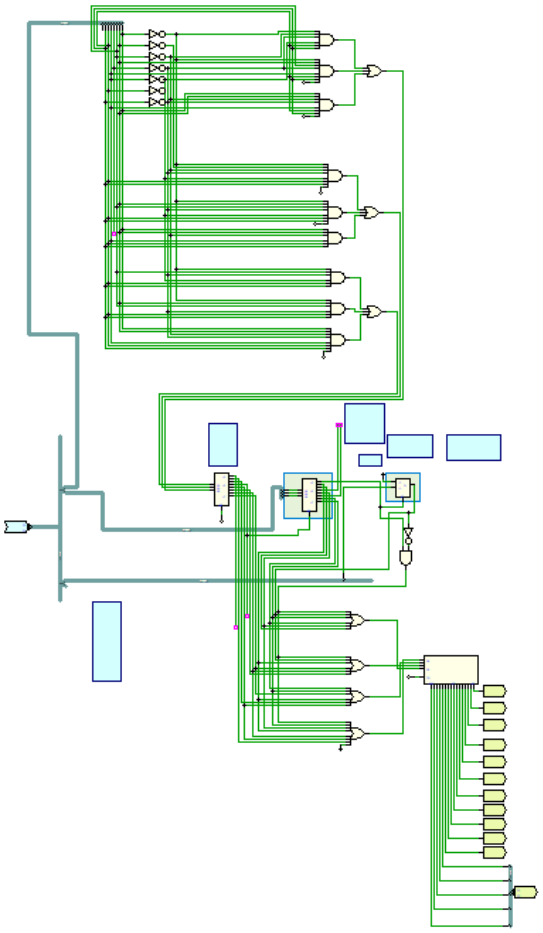

Figure 9. Circuito interno do CBE do bloco de controle

A partir das saídas dos encoders foi desenvolvido um decoder que seleciona um endereço de memória de uma ROM de acordo com a instrução solicitada. Por exemplo, quando o opcode da instrução beq for inserido no circuito do controle, o endereço selecionado na ROM será o 0000. Com isso foi possível definir cada bit dentro dos endereços da ROM como um sinal de controle diferente de acordo com a tabela 1.

Bit da ROM	Sinal
0	Branch (Bra)
1	MemWrite (MW)
2	MemRead (MR)
3	RegWrite (RW)
4	Mem2Reg (M2R)
5	ALUSrc
6	ALUouPC4 (AP4)
7	Bra2

Table 1. Relação dos bits de controle dentro de cada endereço da memória ROM

Para a facilitação da construção do processador *femto*RISC-V a implementação do

componente ALU Control, responsável por fornecer à ULA o código necessário à seleção de cada operação aritmética, foi incorporado ao controle. A memória ROM utilizada possui capacidade armazenamento de até 16 bits em cada endereço e somente foram utilizados 8 bits para armazenar os sinais necessários, desse modo foram dedicados mais 5 bits dos que sobraram da memória para definir ALU Control de acordo com a instrução específica. Os bits escolhidos foram os 5 mais significativos.

O restante dos bits da memória foram *setados* como 0 e não serão utilizados no processador *femto*RISC-V uniciclo. É interessante notar que caso seja necessário incluir mais sinais de controle futuramente essa tarefa será facilitada, visto que só será necessário *setar* um ou mais bits dos que sobraram.

Com isso cada endereço da memória ROM ficou dividido da seguinte forma:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
ALUControl					Bits Livres			Bra2	AP4	ALUSrc	M2R	RW	MR	MW	Bra

Table 2. Divisão dos bits nos endereços da ROM

A tabela 3 relaciona os endereços da ROM às respectivas instruções implementadas pelo processador *femto*RISC-V. A ROM utilizada implementa até 16 funções, como somente 14 instruções foram utilizadas nesse processador, dois espaços ficaram vagos e poderão ser utilizados futuramente para a implementação de outras instruções da ISA RISC-V.

Endereço	Instrução
0000	beq
0001	addi
0010	jal
0011	sw
0100	lui
0101	lw
0110	jalr
0111	sll
1000	slt
1001	xor
1010	or
1011	and
1100	sub
1101	add
1110	(não utilizado)
1111	(não utilizado)

Table 3. Instruções relacionadas a cada endereço da ROM

6. Shift Left Logical Immediate

Para a realização do Shift Left Logical Immediate (SLLI) foi preciso pegar a entrada de 32 bits que vem do registrador ”A” e desloca-lo para a esquerda a quantidade de vezes do

imediato que vem no registrador ”B”. Para isso é necessário pegar o bus da entrada ”A” e manipular manualmente os bits para que eles fiquem deslocados, isso é realizado para todas as 32 opções de deslocamento possíveis. Depois disso basta utilizar multiplexadores que recebem os 5 primeiros bits do registrador ”B” para escolher qual das 32 opção de deslocamento que devem estar na saída

O resultado final do circuito interno do CBE utilizado pode ser visto na figura 10.

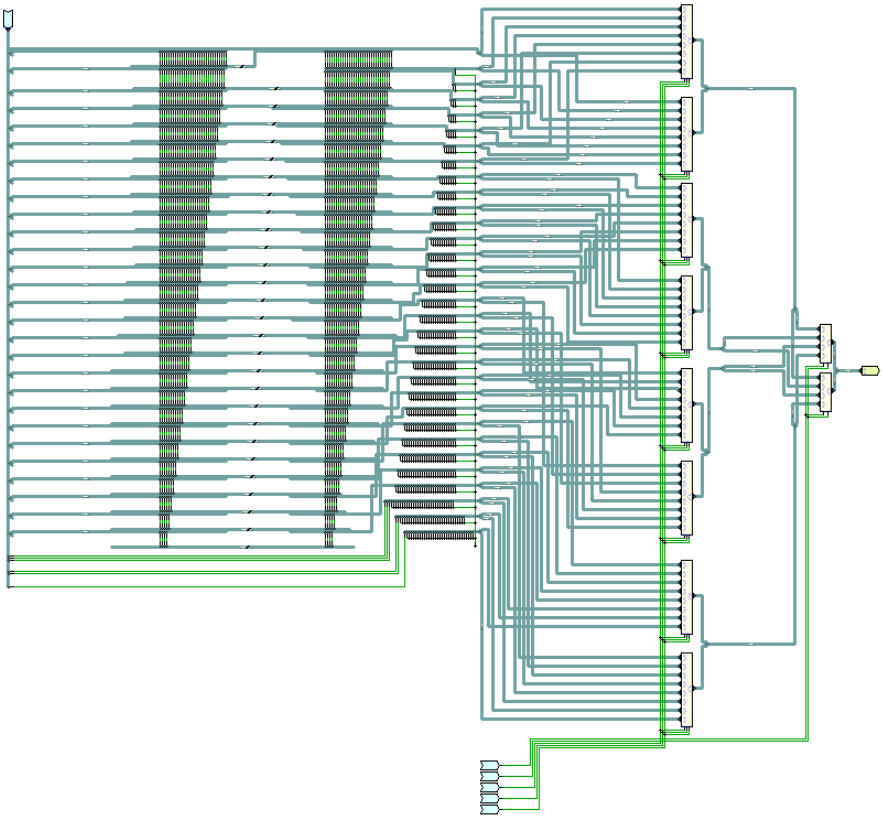

Figure 10. Shift Left Logical Immediate

6.1. Multiplexador de 32 bits

Na implementação do processador em questão foi necessária a utilização de diversos multiplexadores ($2 \rightarrow 1$) de 32 bits cada. No entanto, o Deeds não fornece um mux que seja capaz de multiplexar tantos bits e desse modo foi necessário implementar um CBE que realizasse a tarefa.

O circuito foi feito utilizando a divisão e concatenação de bits dos BUSes de entrada e saída. A figura 15 apresenta o resultado final do circuito interno do CBE implementado.

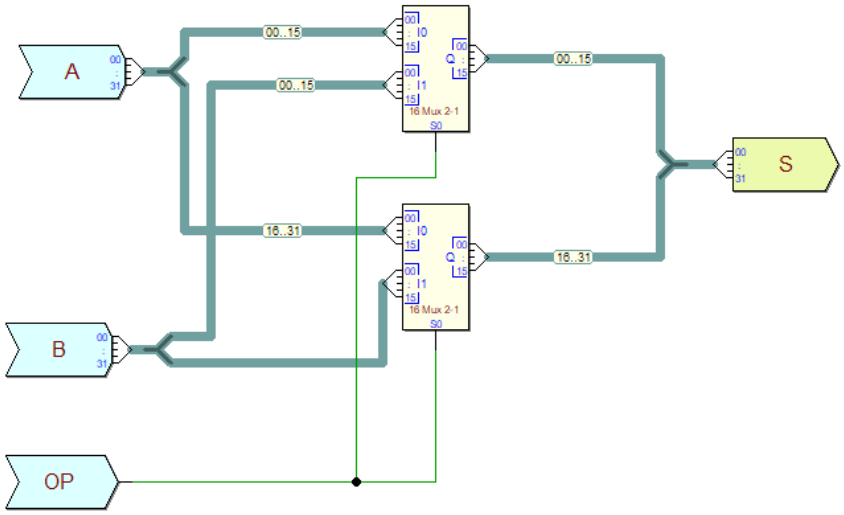

Figure 11. Circuito interno do CBE do Mux de 32 bits

6.2. Registradores de Estado

A principal lógica por trás do funcionamento de processadores que implementam organizações do tipo pipeline é a capacidade de os *outputs* dos elementos serem salvos logo após sua disponibilização na saída. Para isso, foram construídos 4 registradores de estado responsáveis por seccionar o funcionamento do processador e permitir que os outros elementos sejam utilizados mais de uma vez por instrução.

Os registradores desenvolvidos estão relacionados abaixo e seus circuitos internos estão apresentados nas seções de 6.2.1 a 6.2.4 adicionalmente é interessante notar que à exceção do registrador IF/ID todos os registradores de estagio armazenam sinais de controle.

- IF/ID: A instrução é lida da memória nesse estágio, desse modo o registrador é responsável por armazenar o endereço de PC e o código da instrução.
- ID/EX: Esse registrador armazena os dados lidos do banco de registradores, o valor atual de PC e o imediato gerado em componente dedicado à isso.
- EX/MEM: Armazena os resultados disponibilizados por operações da ULA e o próximo valor de PC, ou seja, uma escolha entre PC+4 e PC+Imm. Além disso, o registrador em questão também armazena o segundo dado consultado no banco de registrador para eventuais armazenamentos na memória de dados a depender da instrução sendo executada.
- MEM/WB: Responsável pela etapa final da execução da instrução: ler o dado do registrador MEM/WB e escrevê-lo no banco de registradores.

6.2.1. Registrador IF/ID

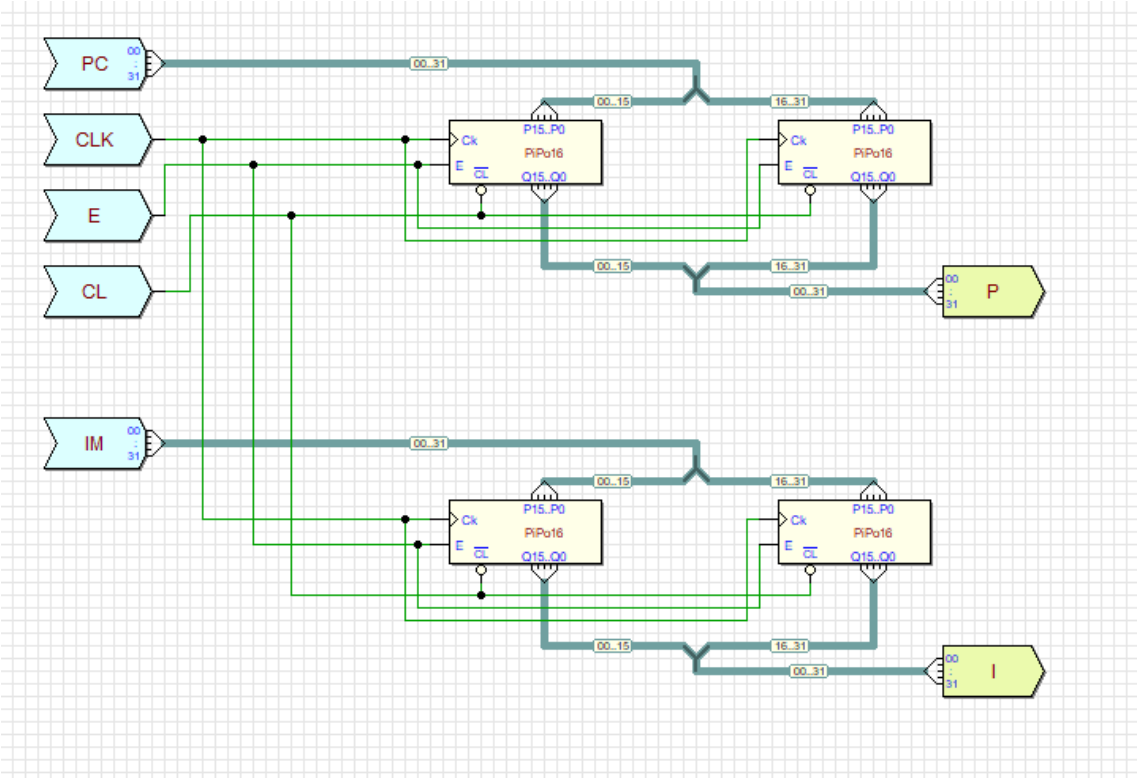

Figure 12. Registrador IF/ID

6.2.2. Registrador ID/EX

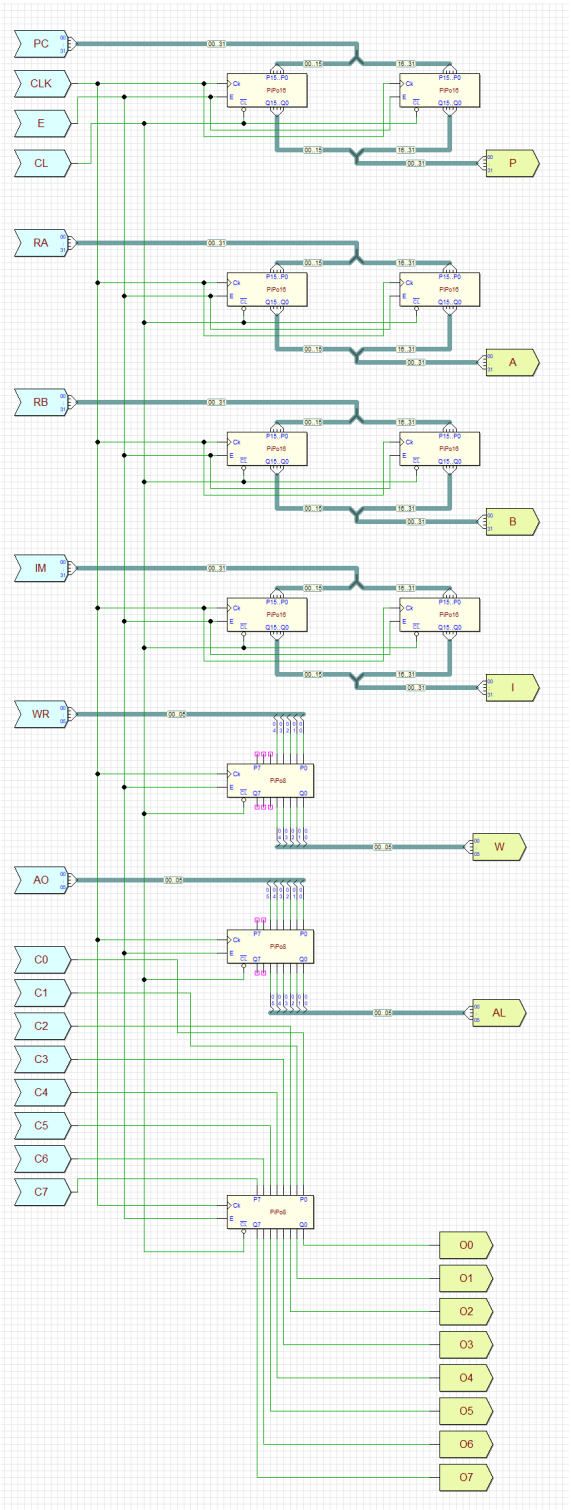

Figure 13. Registrador ID/EX

6.2.3. Registrador EX/MEM

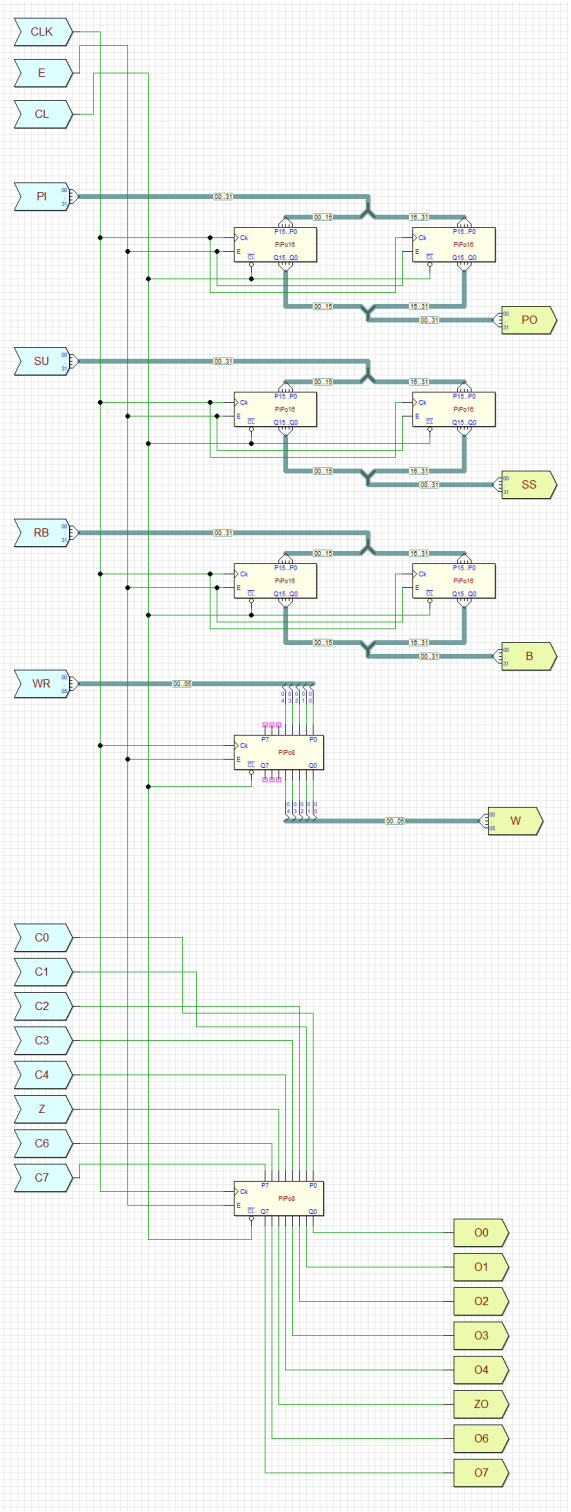

Figure 14. Registrador EX/MEM

6.2.4. Registrador MEM/WB

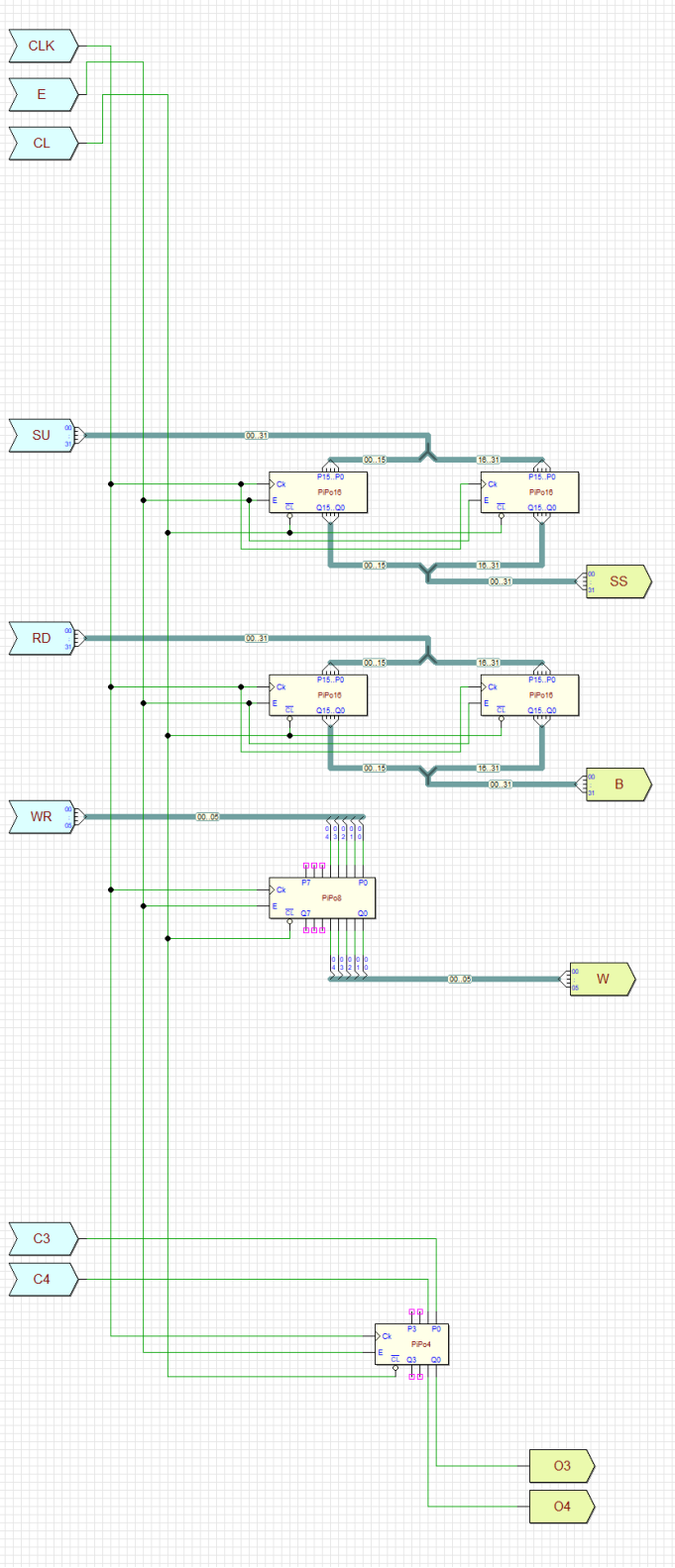

Figure 15. Registrador MEM/WB

6.3. *femto*Risc-V completo

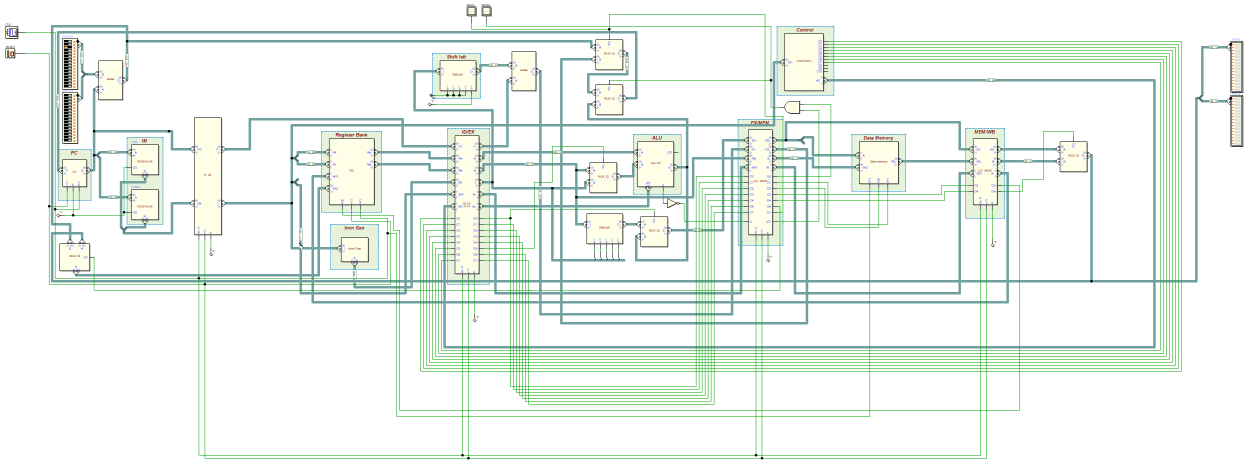

Figure 16. Circuito do *femto*Risc-V

O processador Pipeline completo não foi capaz de executar todas as instruções, todos os blocos funcionaram isoladamente, porém a montagem completa do processador apresentou problemas que não foram possíveis de serem corrigidos a tempo. Entretanto analisamos que sua ocorrência tem relação com as saídas do controle, elas que são responsáveis por administrar os registradores do pipeline. Em razão desses fatos não foi possível analisar o gráfico de ondas e nem o tempo necessário para a execução de todas as instruções. As demais explicações quanto à parte de implementação que envolvem o processador podem ser vistos em vídeo neste link.

References

[Alves 2014] Alves, P. (2014). Entenda a diferença entre memória RAM e memória ROM.

[Arruda 2011] Arruda, F. (2011). O que é memória ROM?

[Pattersson and Henessy 2021] Pattersson, D. A. and Henessy, J. L. (2021). *Computer Organization and Design RISC-V Edition*. Elsevier Inc.: Katey Birtcher, 2nd edition.