Universidade Federal do Rio Grande do Sul
Instituto de Informática

INF01113
Organização de Computadores B

Trabalho Prático 1 - Tarefa 09

Implementação de Instruções no MIPS

Bruno Alexandre Hofstetter Bourscheid (00550177)
Fernando Longhi de Andrade (00580366)
Luiz Augusto Ponzoni Schmidt (00580108)
Miguel Dutra Fontes Guerra (00342573)
Pedro Lubaszewski Lima (00341810)

Turma B

19 de abril de 2025

Sumário

1.1 MIPS Singlecycle

1.1.1 Modificações no Bloco Operativo

Para começar, segue uma ilustração de como estava inicialmente a parte operativa do MIPS Singleclycle:

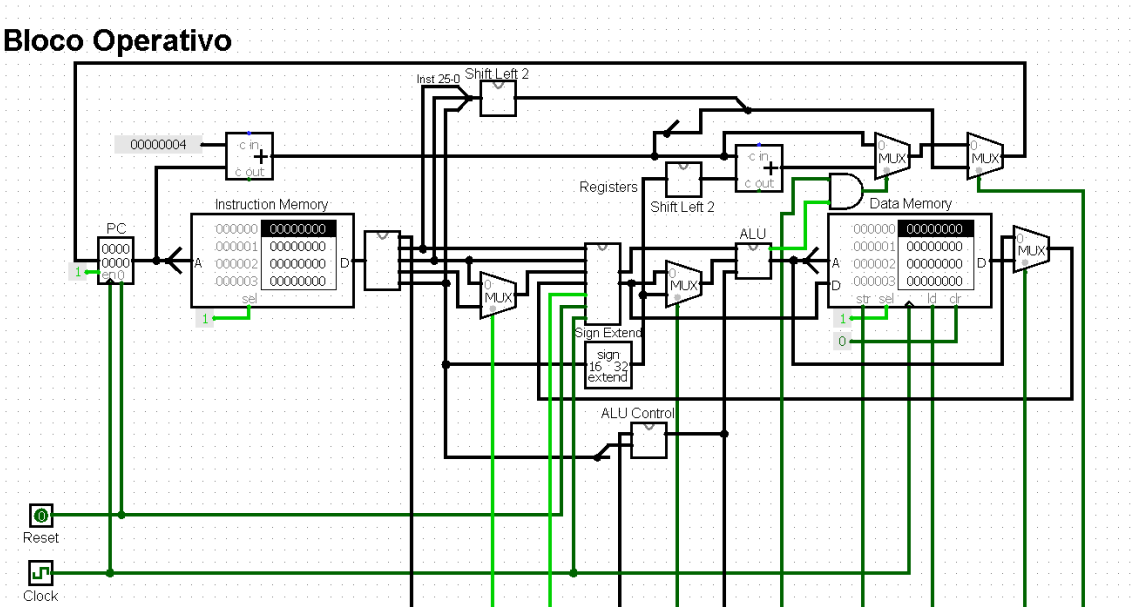

Figura 1: MIPS Singlecycle - Bloco Operativo Antes

Essa parte operativa foi modificada de acordo com as instruções adicionadas ao MIPS Singlecycle:

- Adicionada operação multiplicação na ALU;
- Adicionada saída dos bits superiores da multiplicação na ALU;
- Adicionada saída IsMULT da ALU;
- Adicionados registradores especiais LO e HI;
- Atualizada a flag 0 na ALU;
- Adicionado comparador para BLTZ e SLTIU;
- Adicionado sign extend para o BLTZ e SLTIU;
- Adicionado splitter para LB;
- Adicionado sign extend para LB;
- Atualizado o splitter para a Data Memory.

Segue uma ilustração de como ficou a parte operativa do MIPS Singlecycle após as modificações acima listadas:

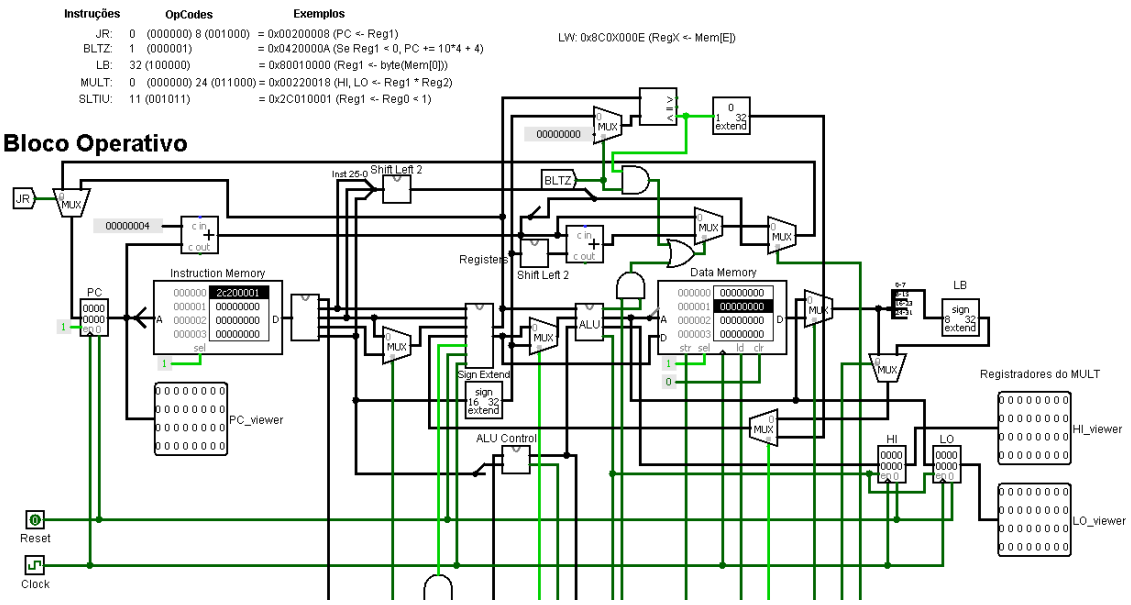

Figura 2: MIPS Singlecycle - Bloco Operativo Depois

1.1.2 Modificações no Bloco de Controle

Já para a parte de controle, segue uma representação inicial do MIPS Singlecycle:

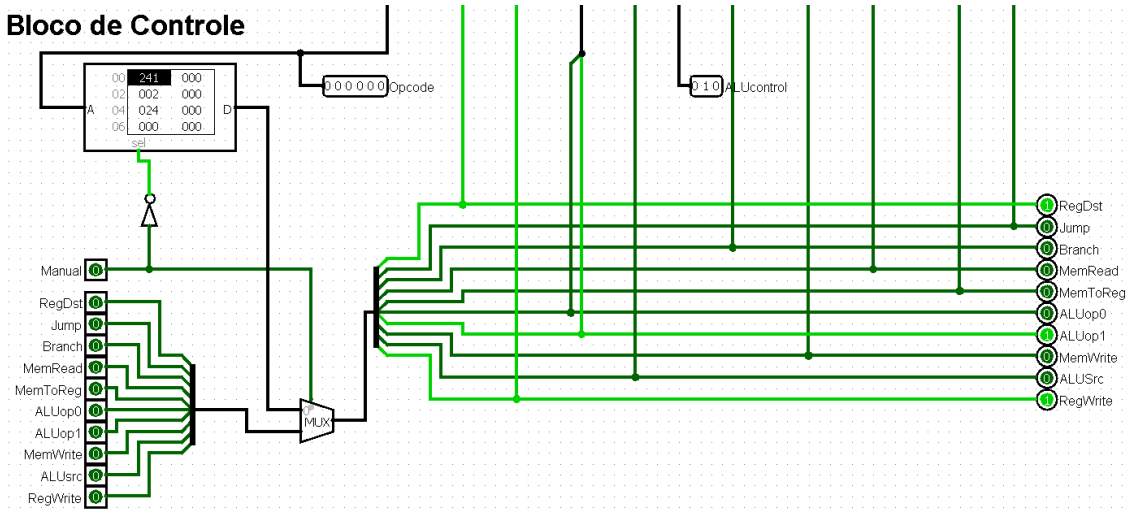

Figura 3: MIPS Singlecycle - Bloco Controle Antes

Com ela em mente, foram feitas as seguintes modificações:

- Criados os sinais de controle LB, SLTIU, BLTZ, JR e IsMult;
- Atualizado o ALU Control para as instruções de MULT e JR;
- Adicionado MUX para o BLTZ;
- Adicionado controle de branch para BLTZ;
- Adicionado controle de RegWrite usando IsMULT;
- Adicionado MUX para o JR;
- Adicionado MUX para LB;

- Adicionado MUX para SLTIU.

Abaixo segue também a codificação em memória ROM de controle das novas instruções:

- ROM para JR: 0x0241 end 0x00;
- ROM para BLTZ: 0x0404 end 0x01;
- ROM para LB: 0x0B18 end 0x20;
- ROM para MULT: 0x0241 end 0x00;
- ROM para SLTIU: 0x1300 end 0x0B.

Dessa forma, a parte de controle do processador terminou da seguinte forma:

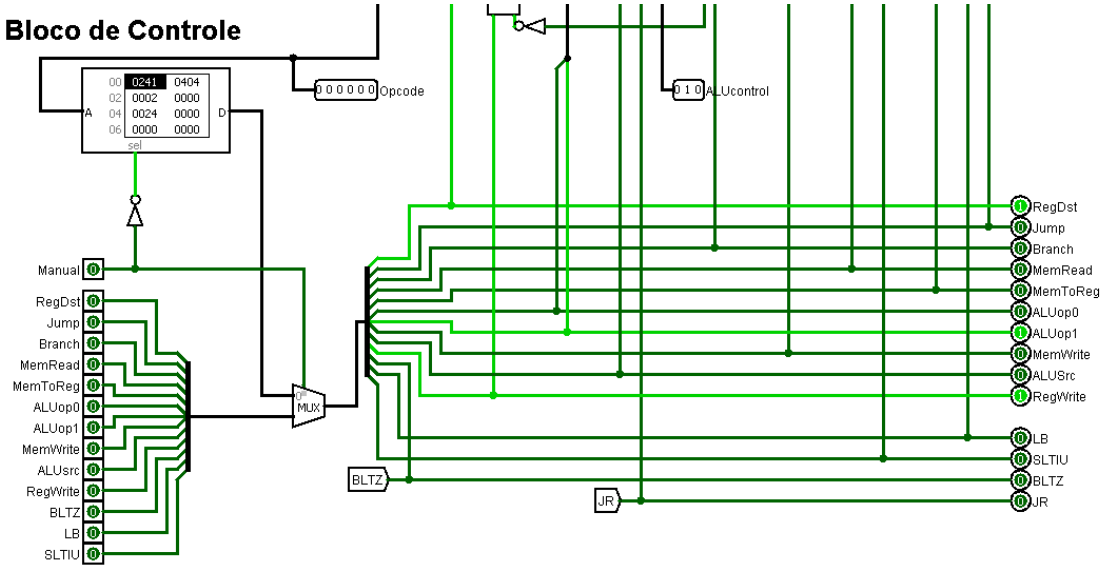

Figura 4: MIPS Singlecycle - Bloco Controle Depois

Algumas dessas modificações aqui listadas foram mostradas em mais detalhes na subseção anterior.

1.1.3 Testes de Implementação

Para comprovar que essas mudanças acima trouxeram a correta implementação das instruções especificados neste trabalho, abaixo seguem os testes realizados com imagens e a respectiva sequência de instruções:

- Teste da Instrução JR:

  JR: PC ← Reg1

  0x8C010000; Reg1 ← Mem[0] (3)

  0x00200008; PC ← Reg1 (3)

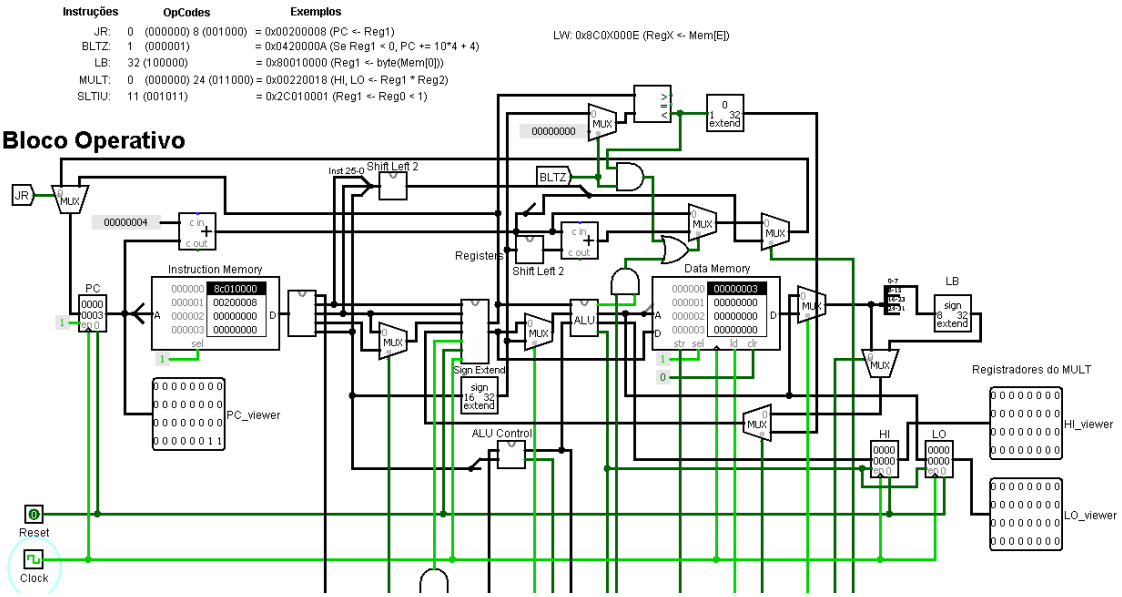

Figura 5: MIPS Singlecycle - Teste JR

- Teste da Instrução BLTZ:

  BLTZ: if Reg1 < 0 then PC += IMED·4 + 4 (IMED = 10)

  0x8C010000; Reg1 ← Mem[0] (-1 (0xFFFFFFFF))

  0x0420000A; PC += IMED·4 + 4 (PC = 48)

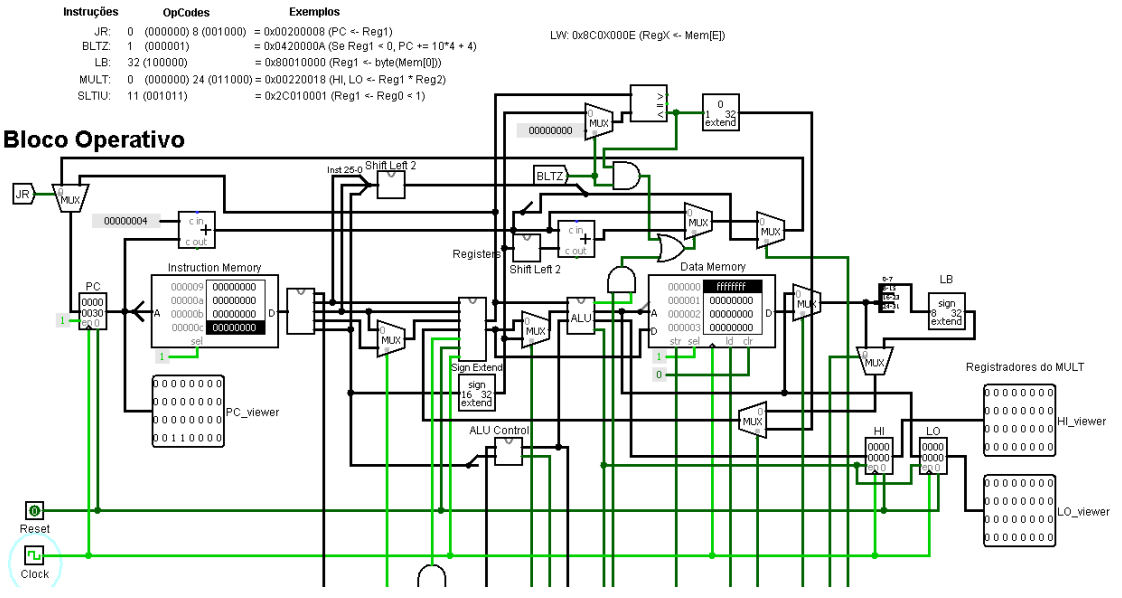

Figura 6: MIPS Singlecycle - Teste BLTZ

- Teste da Instrução LB:

  LB: Reg1 ← byte(Mem[0]) (3)

  0x80010000; Reg1 ← byte(Mem[0])

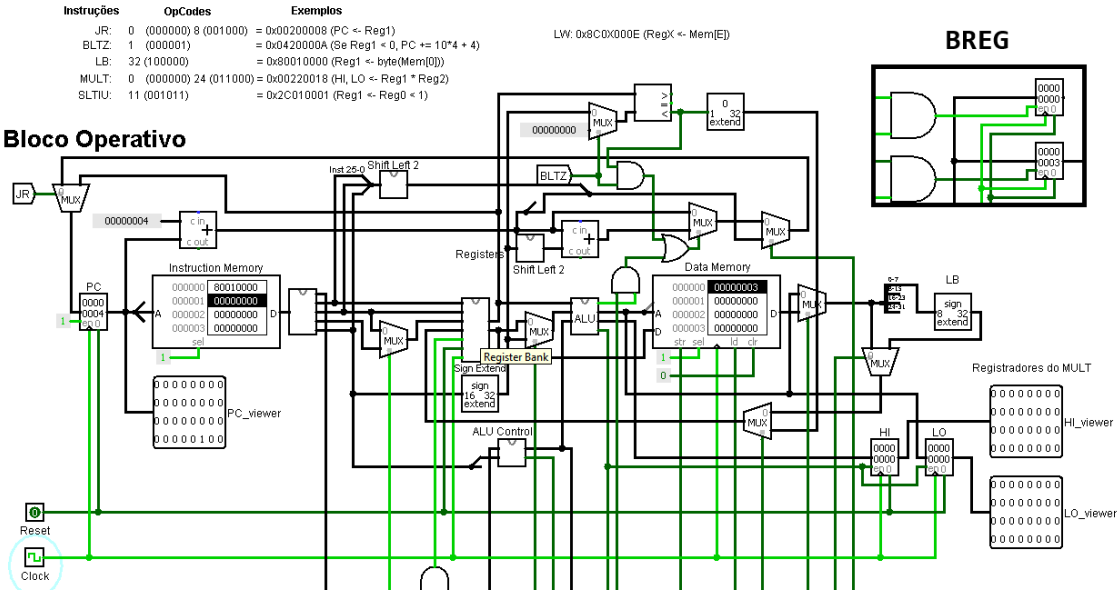

Figura 7: MIPS Singlecycle - Teste LB

- Teste da Instrução MULT:

  MULT: HI, LO ← Reg1 · Reg2

  0x8C010000; Reg1 ← Mem[0] (3)

  0x8C020001; Reg2 ← Mem[1] (4)

  0x00220018; HI, LO ← Reg1 · Reg2 (HI = 0, LO = 12)

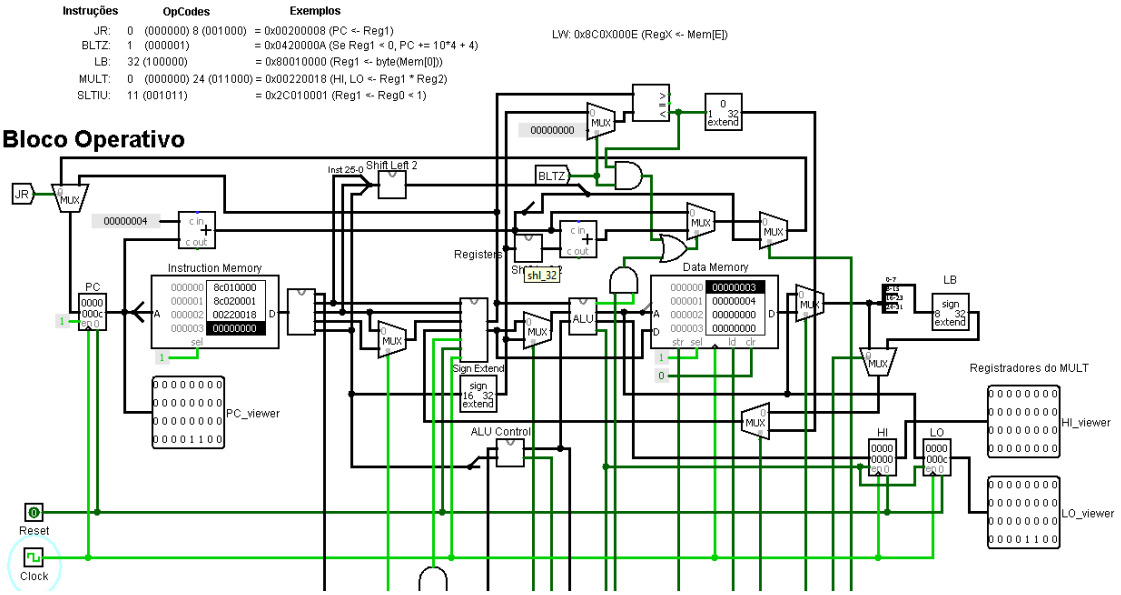

Figura 8: MIPS Singlecycle - Teste MULT

- Teste da Instrução SLTIU:

  SLTIU: Reg1 ← Reg0 < IMED (1)

  0x2C010001; Reg1 ← Reg0 < 1 (Reg1 = 1)

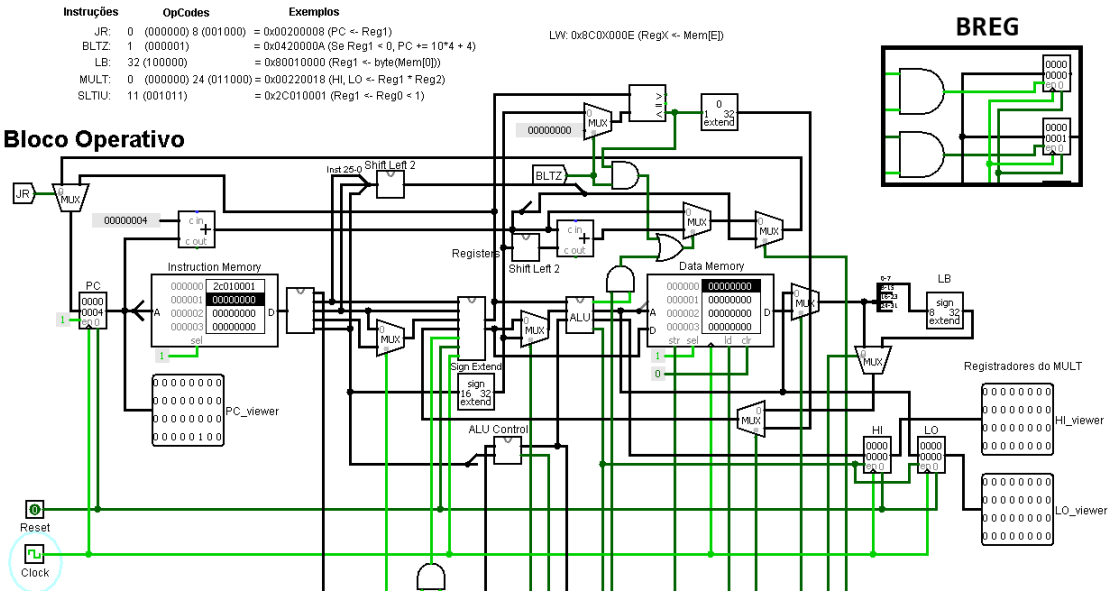

Figura 9: MIPS Singlecycle - Teste SLTIU

2.1 MIPS Multicycle

2.1.1 Modificações no Bloco Operativo

Como a implementação das instruções sorteadas possuiam uma similaridade parcial com as instruções bases, a maioria das mudanças aconteceram para integrar os novos sinais controle com o circuito. Ainda assim, casos especificos foram tratados:

- JR: Como se tratava de uma instrução R-Type não houve alterações nas ROMs, apenas na ULA-Control para lidar com o Funct e gerar o sinal isJR, este sendo utilizado como seletor no Mux que lida com o update do PC.
- BLTZ: Aqui foi feita uma implementação dos estados B e C, tendo novamente o sinal utilizado como seletor para a parte operativa, cuja qual a partir dos blocos adicionados (conferir Figura 11) realiza a comparação para testar a validade do branch, prossegue-se assim similar ao BEQ já implementado
- LB: Implementação muito parecida com o LW, aqui foi utilizado o sinal criado no controle como seletor para uma lógica MUX que separa a busca em duas opções, o dado completo ou apenas um Byte com Sign Extension
- Mult: O multi por se tratar de uma instrução Especial teve seu controle trabalhado no ALU-Control, e sua implementação foi caracterizada pela criação dos registradore HI e LO, junto com um multiplicador.
- SLTIU: Foi adicionado um registrador para armazenamento da saída da instrução SLTIU, um MUX com constantes para escrita do resultado de SLTIU, além de um signal extension e os multiplexadores que tratam a lógica do novo sinal de controle isMult

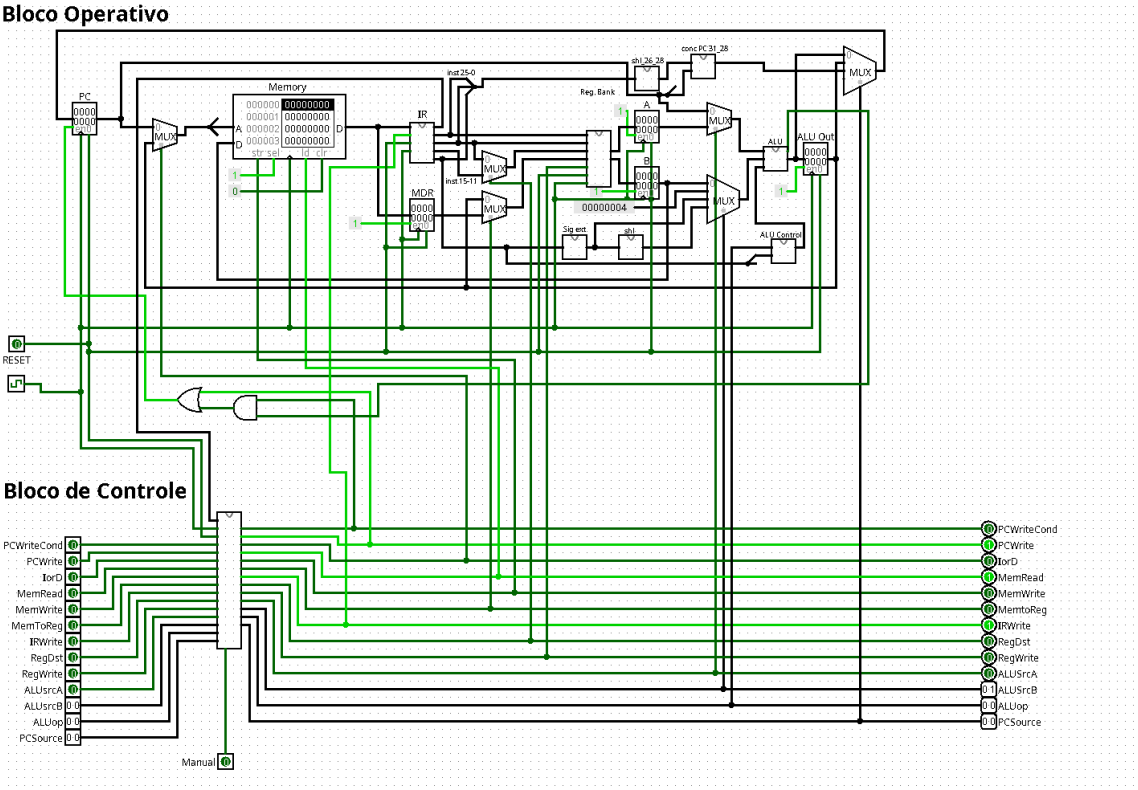

Figura 10: MIPS Multicycle - Bloco Operativo Inicial

Observe as mundanças a seguir:

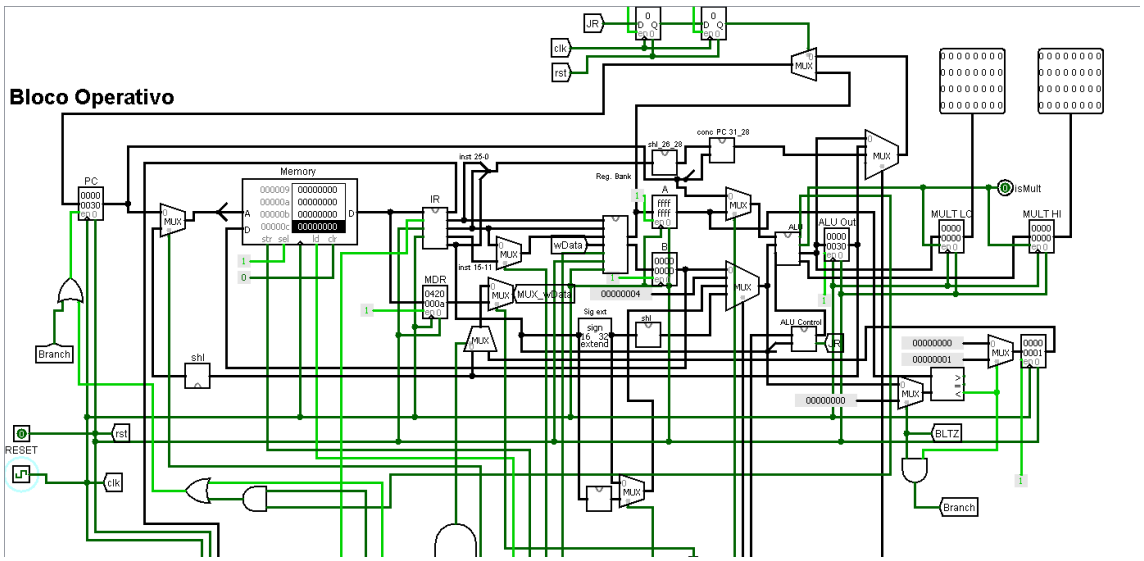

Figura 11: MIPS Multicycle - Bloco Operativo Final

2.1.2 Modificações no Bloco de Controle

Além da adição de três sinais (isBLTZ/isSLTIU/isLB). Foram modificadas as ROMs de controle de NextState e Saída, na primeira as alterações se deram pelas implementações da lógica de estado para as novas instruções, todas descritas na FSM no tópico a seguir. Na segunda foi adicionado as saídas para os estado A, B e C; 60, 30 e 31 respectivamente. Ademais bastou-se as alterações na ULA-Control, onde foi feito a lógica para o isMult e o isJR, ambas similares as implementações do singlecycle.

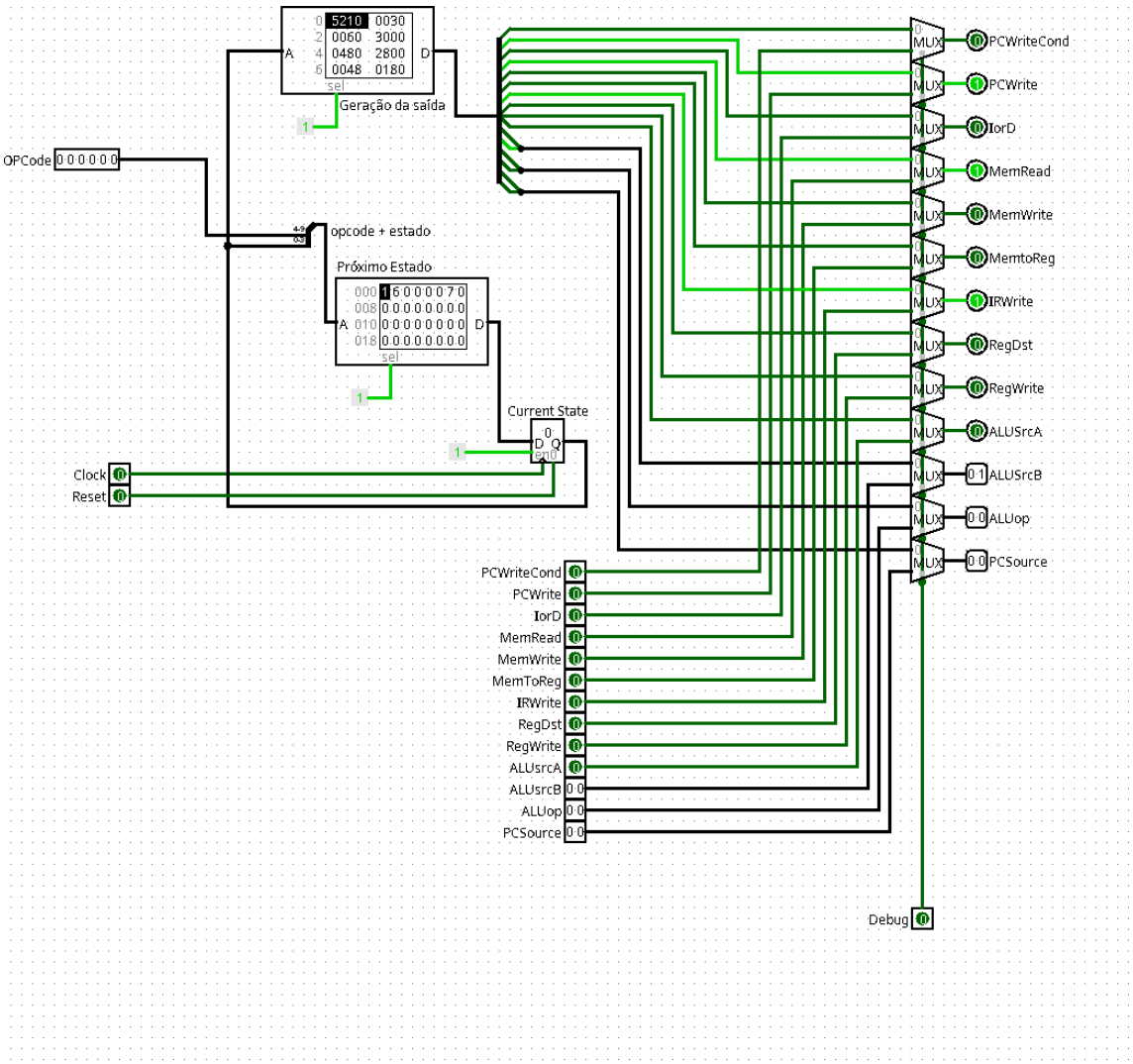

Figura 12: MIPS Multicycle - Controle Inicial

A próxima subseção mostra de forma mais detalhada as atualizações das ROMs e lógicas implementadas, mas por enquanto observe as mundanças a seguir:

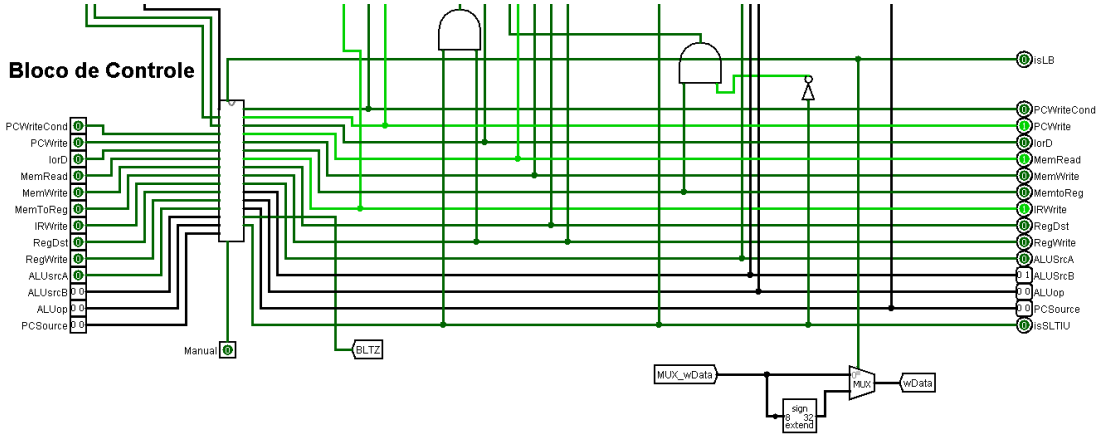

Figura 13: MIPS Multicycle - Controle Final

2.1.3 Estados de Controle

São utilizados no projeto 12 estados diferentes, contendo instruções de 3 a 5 ciclos, todas representadas na FSM anexada a seguir: Nota-se que os nodos representam os estados de controle,

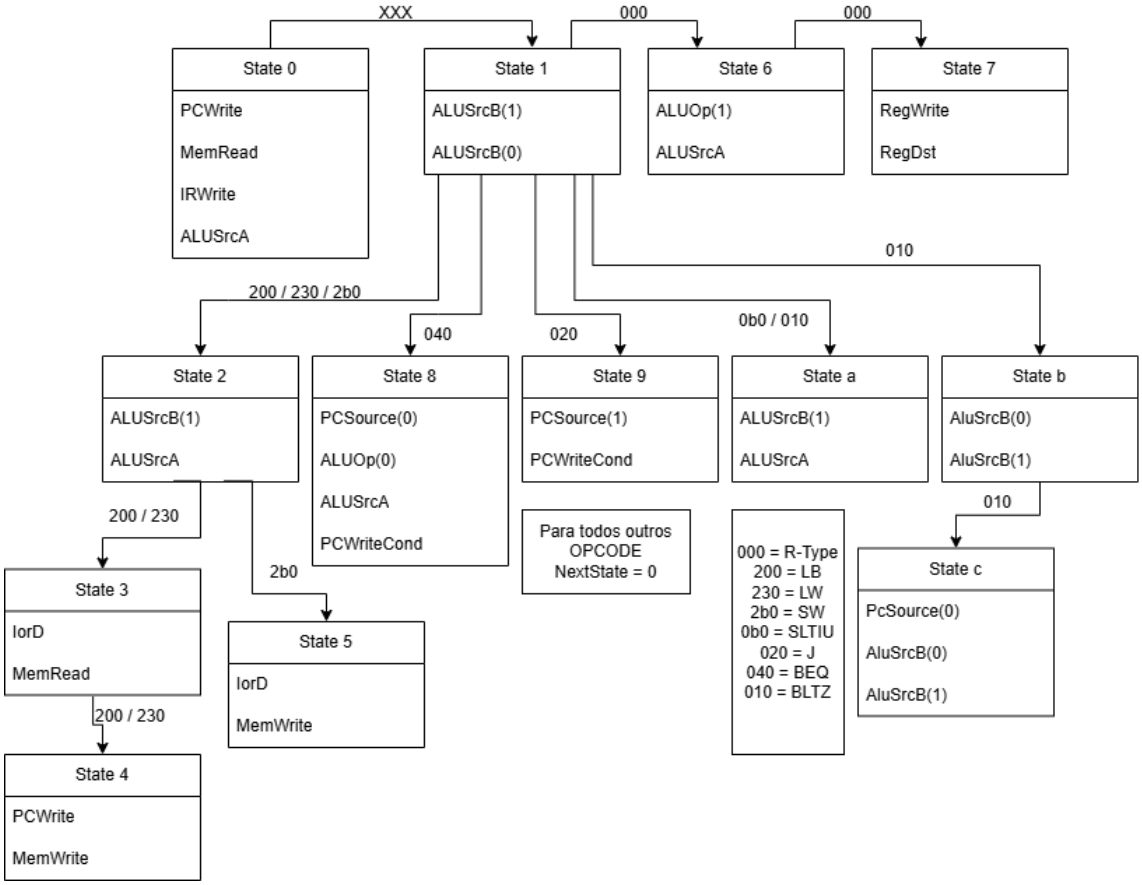

Figura 14: MIPS Multicycle - Maquina de estados Multicycle

enquanto as arestas os OPCodes, assim formando uma máquina de Moore com as saídas em "1"de cada estado notadas em cada nodo.

2.1.4 Testes de Implementação

Segue a seguir os testes realizados acompanhados do código utilizado:

- Teste da Instrução JR:

  JR: PC ← Reg1

  0x8C010000; Reg1 ← Mem[0] (3)

  0x00200008; PC ← Reg1 (3)

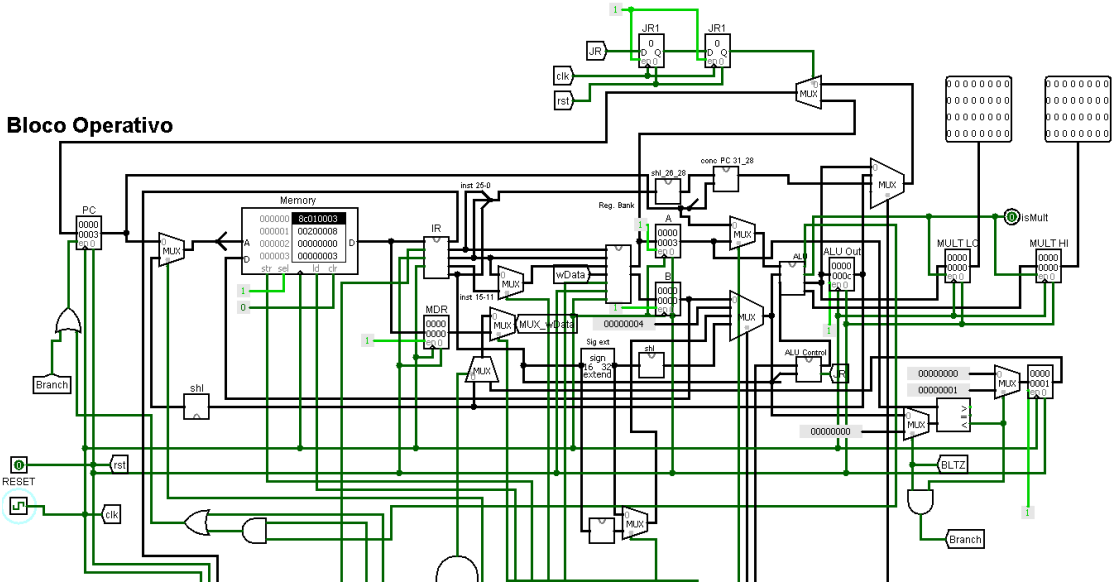

Figura 15: MIPS Multicycle - Teste JR

- Teste da Instrução BLTZ:

  BLTZ: if Reg1 < 0 then PC += IMED·4 + 4 (IMED = 10)

  0x8C010000; Reg1 ← Mem[0] (-1 (0xFFFFFFFF))

  0x0420000A; PC += IMED·4 + 4 (PC = 48)

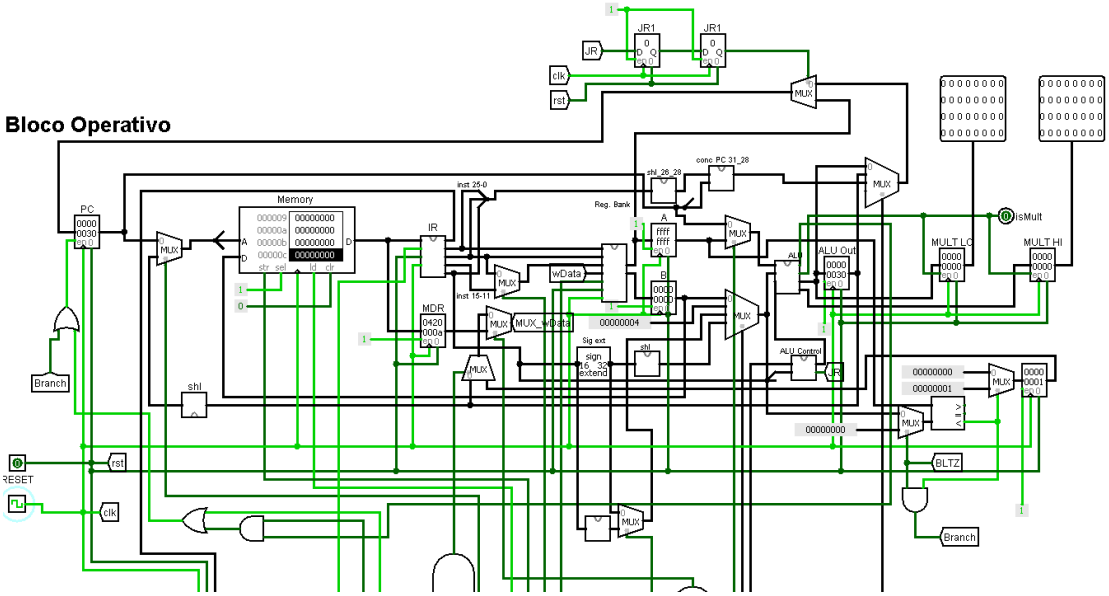

Figura 16: MIPS Multicycle - Teste BLTZ

- Teste da Instrução LB:

  LB: Reg1 ← byte(Mem[0]) (3)

0x80010000; Reg1 ← byte(Mem[0])

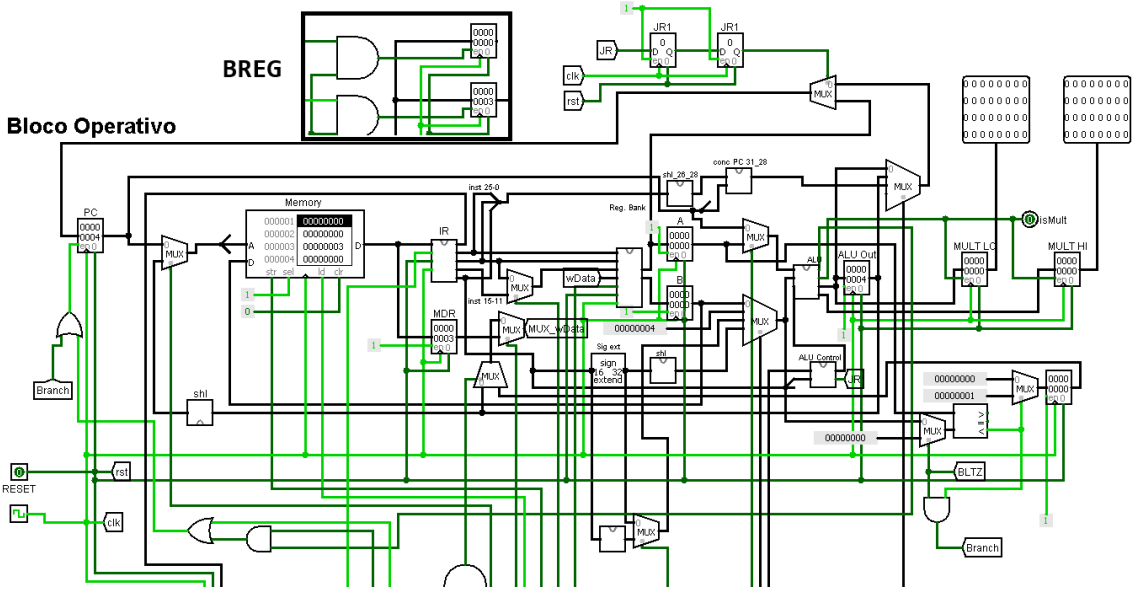

Figura 17: MIPS Multicycle - Teste LB

- Teste da Instrução MULT:

  MULT: HI, LO ← Reg1 · Reg2

  0x8C010000; Reg1 ← Mem[0] (3)

  0x8C020001; Reg2 ← Mem[1] (4)

  0x00220018; HI, LO ← Reg1 · Reg2 (HI = 0, LO = 12)

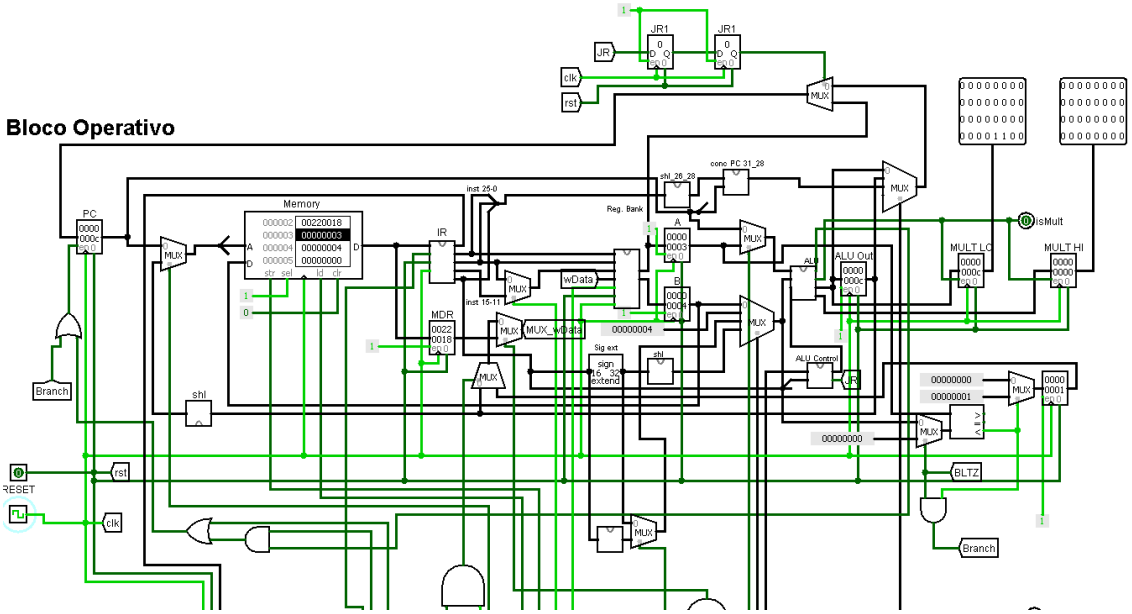

Figura 18: MIPS Multicycle - Teste MULT

- Teste da Instrução SLTIU:

  SLTIU: Reg1 ← Reg0 < IMED (1)

  0x2C010001; Reg1 ← Reg0 < 1 (Reg1 = 1)

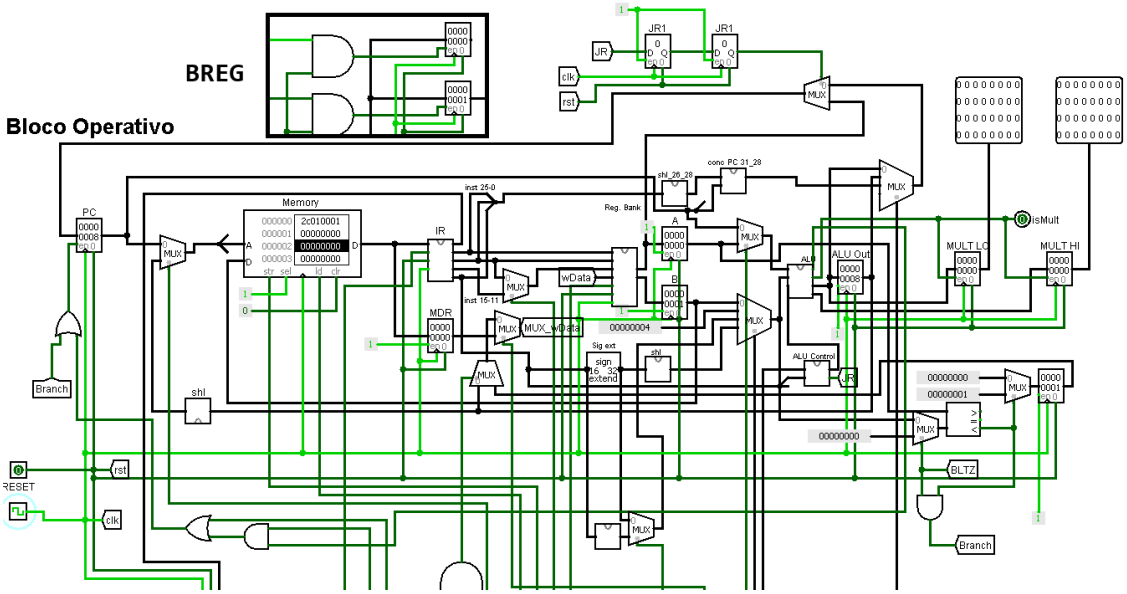

Figura 19: MIPS Multicycle - Teste SLTIU

3.1 MIPS Pipeline

3.1.1 Modificações no Bloco Operativo

Primeiramente, segue uma visão de como estava a implementação do MIPS com pipeline antes do acrescimo das novas instruções:

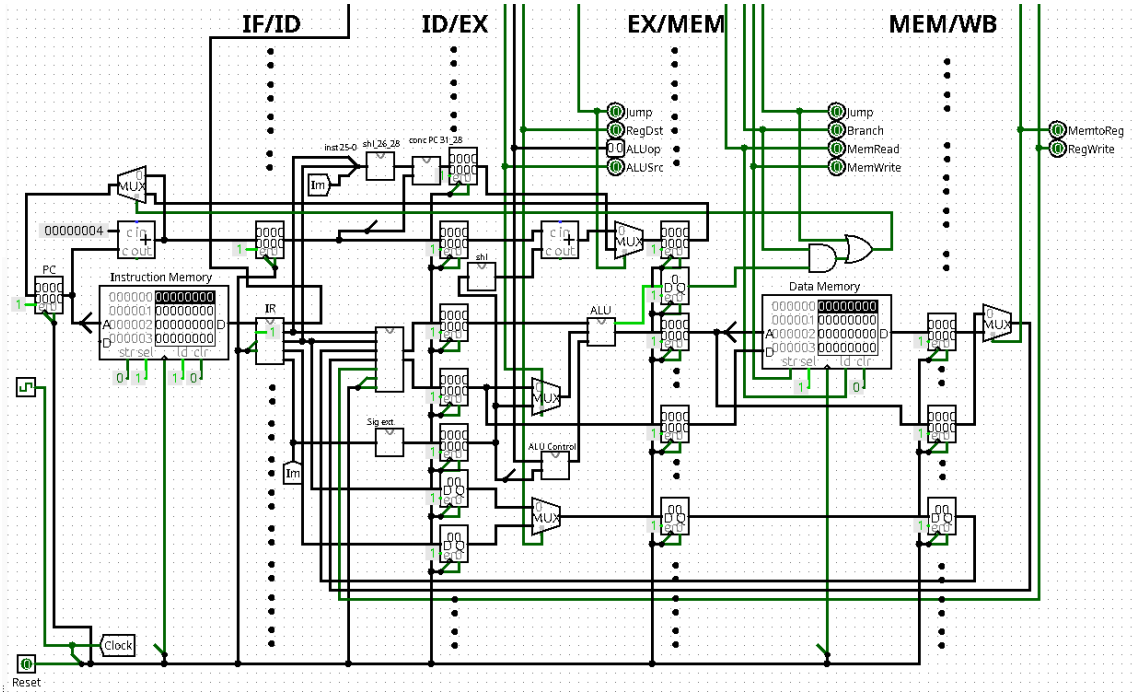

Figura 20: MIPS Pipeline - Bloco Operativo Antes

Sobre essa versão do MIPS com pipeline, foram feitas as modificações listadas à seguir:

- Adicionada operação multiplicação na ALU;
- Adicionada saída dos bits superiores da multiplicação na ALU;
- Adicionada barreira temporal para capturar valor de RS;
- Adicionado MUX para o JR na entrada do PC;
- Corrigidos os timings entre barreiras temporais para o sinal de IsMult;
- Utilizado o sinal IsMult para não habilitar a escrita nos registradores;
- Propagado o sinal de imediato com 32 bits para a etapa de MEM;
- Colocado comparador de menor que 0 ou menor que RS para instruções de BLTZ e SLTIU em MEM;
- Conectado o BLTZ ao mux de branches;
- Propagado o bit de comparação RS ¡ Imed para o estágio de MEM;
- Adicionado bit extender para o resultado da comparação acima;
- Conectado o resultado desse bit extender na entrada de escrita dos GPRs;
- Adicionado splitter para pegar os 8 bits menos significativos de um dado da memória;
- Extendido esses 8 bits acima;
- Conectado esse resultado controlado pelo sinal LB nos GPRs.

Com essas modificações listadas acima, obteve-se a seguinte nova representação do bloco operativo do MIPS:

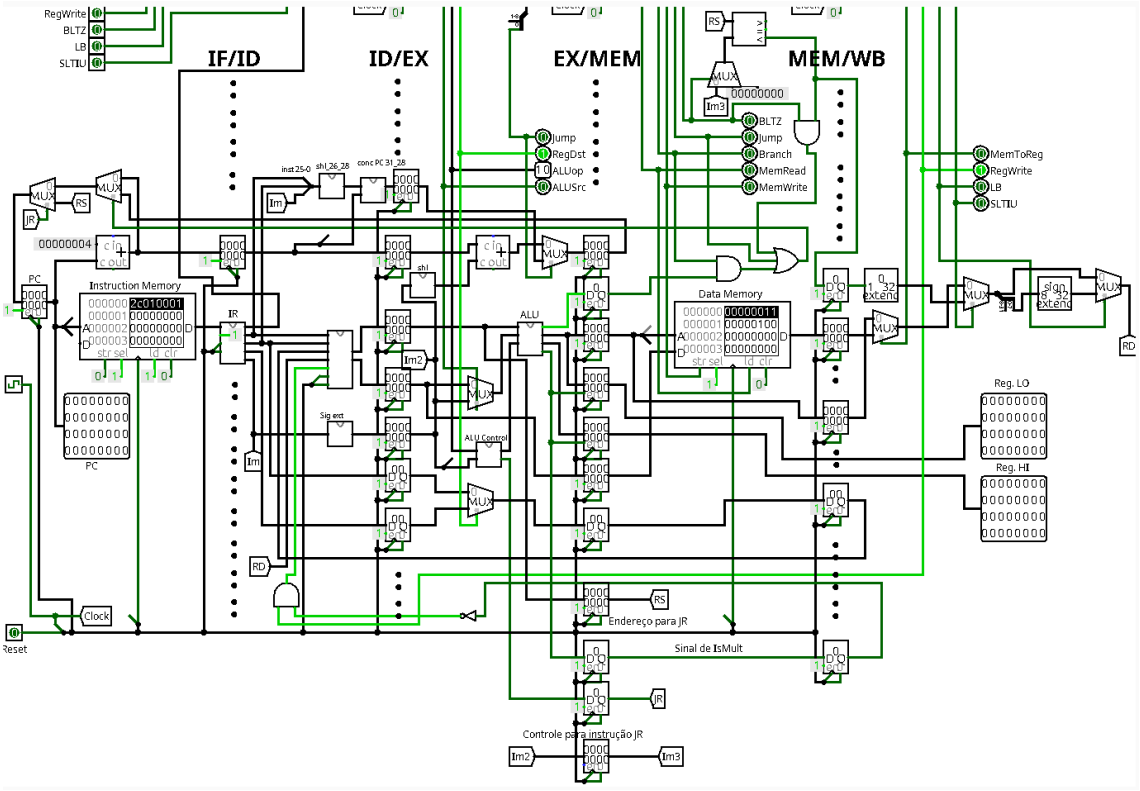

Figura 21: MIPS Pipeline - Bloco Operativo Depois

3.1.2 Modificações no Bloco de Controle

Já na parte de controle, anteriormente, tinha-se a seguinte representação:

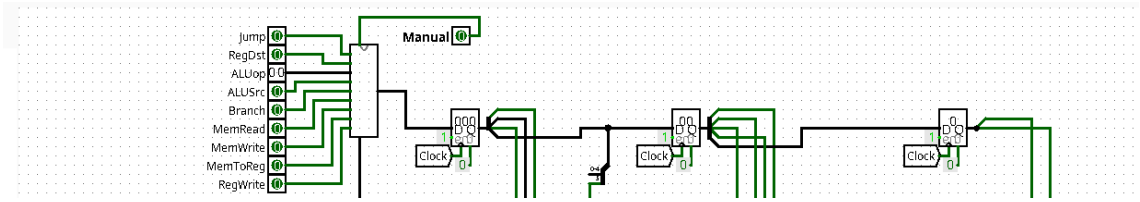

Figura 22: MIPS Pipeline - Bloco Controle Antes

A partir dessa parte de controle, é possível listar as seguintes modificações:

- Criados os sinais de controle LB, SLTIU, BLTZ, JR e IsMult;
- Atualizado o ALU Control para as instruções de MULT e JR;
- Adicionada saída IsMULT da ALU;
- Adicionada barreira temporal para o estágio de EX/MEM do JR;
- Propagados todos os sinais de controle para as respectivas etapas do circuito:

  - BLTZ em MEM;

  - LB e SLTIU em WB.

Além dessas modificações, foram reordenados os sinais de controle para serem equivalentes aos sinais de controle do MIPS Singlecycle. Assim, as modificações da ROM de controle foram as mesmas listadas na seção do MIPS Singlecycle:

- ROM para JR: 0x0241 end 0x00;
- ROM para BLTZ: 0x0404 end 0x01;
- ROM para LB: 0x0B18 end 0x20;

- ROM para MULT: 0x0241 end 0x00;
- ROM para SLTIU: 0x1300 end 0x0B.

Com tudo isso, obteve-se o seguinte novo esquemático externo da parte de controle do MIPS com pipeline:

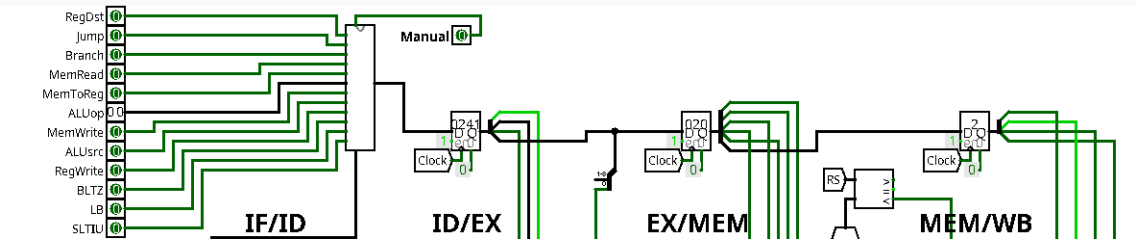

Figura 23: MIPS Pipeline - Bloco Controle Depois

Algumas dessas modificações aqui listadas foram mostradas em mais detalhes na subseção anterior.

3.1.3 Testes de Implementação

Para comprovar que essas mudanças acima trouxeram a correta implementação das instruções especificados neste trabalho, abaixo seguem os testes realizados com imagens e a respectiva sequência de instruções:

É importante notar, em todos os testes, que há uma grande quantidade de NOPs espalhados entre as instruções. Isso é devido a não terem sido implementadas soluções para lidar com dependências de dados e de controle.

- Teste da Instrução JR:

  0x8C010000; LW: Reg1 ← Mem[0] (64 ou 0x40)

  0x00000000; NOP

  0x00000000; NOP

  0x00000000; NOP

  0x00000000; NOP

  0x00200008; JR: PC ← Reg1 (64 ou 0x40)

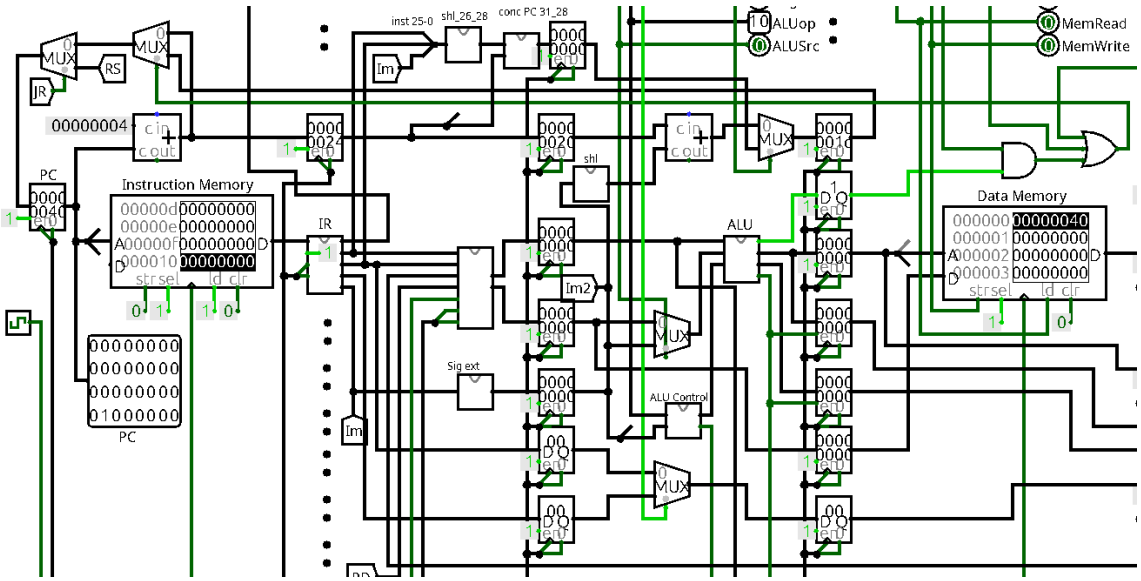

Figura 24: MIPS Pipeline - Teste JR

- Teste da Instrução MULT:

  MULT: HI, LO ← Reg1 · Reg2

  0x8C010000; Reg1 ← Mem[0] (17 ou 0x11)

  0x8C020001; Reg2 ← Mem[1] (256 ou 0x100)

  0x00000000; NOP

  0x00000000; NOP

  0x00000000; NOP

  0x00000000; NOP

  0x00220018; HI, LO ← Reg1 · Reg2 (HI = 0, LO = 4352 ou 0x10400)

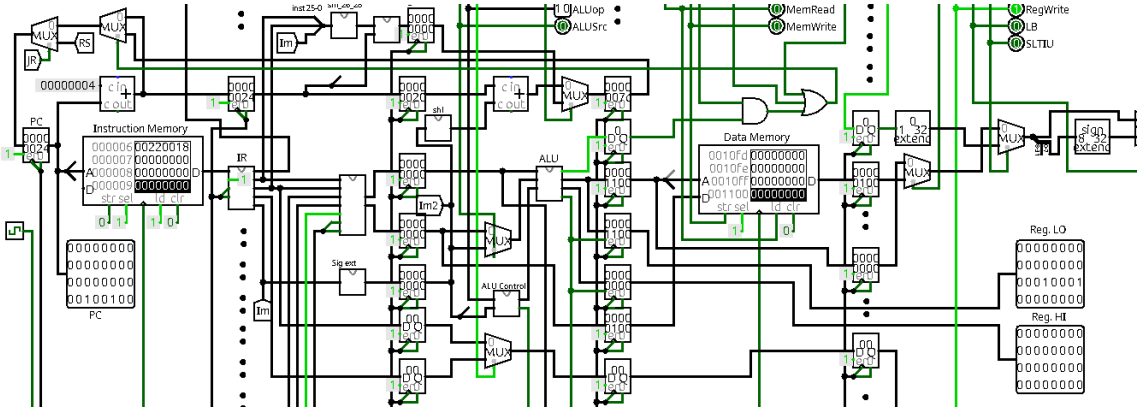

Figura 25: MIPS Pipeline - Teste MULT

- Teste da Instrução BLTZ:

  BLTZ: if Reg1 < 0 then PC += IMED·4 + 4 (IMED = 10)

  0x8C010000; Reg1 ← Mem[0] (-1 (0xFFFFFFFF))

  0x00000000; NOP

  0x00000000; NOP

  0x00000000; NOP

  0x00000000; NOP

  0x0420000A; PC += IMED·4 + 4 (PC = 64 ou 0x40)

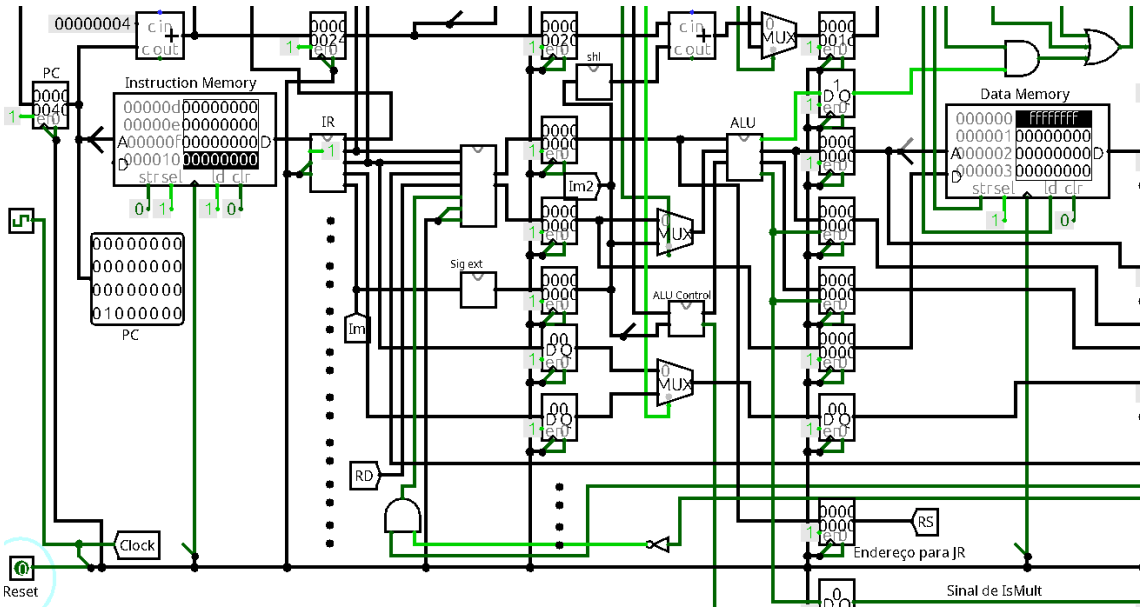

Figura 26: MIPS Pipeline - Teste BLTZ

- Teste da Instrução LB:

  LB: Reg1 ← byte(Mem[0]) (0x00000FFF)

  0x80010000; Reg1 ← byte(Mem[0]) (0xFFFFFFFF)

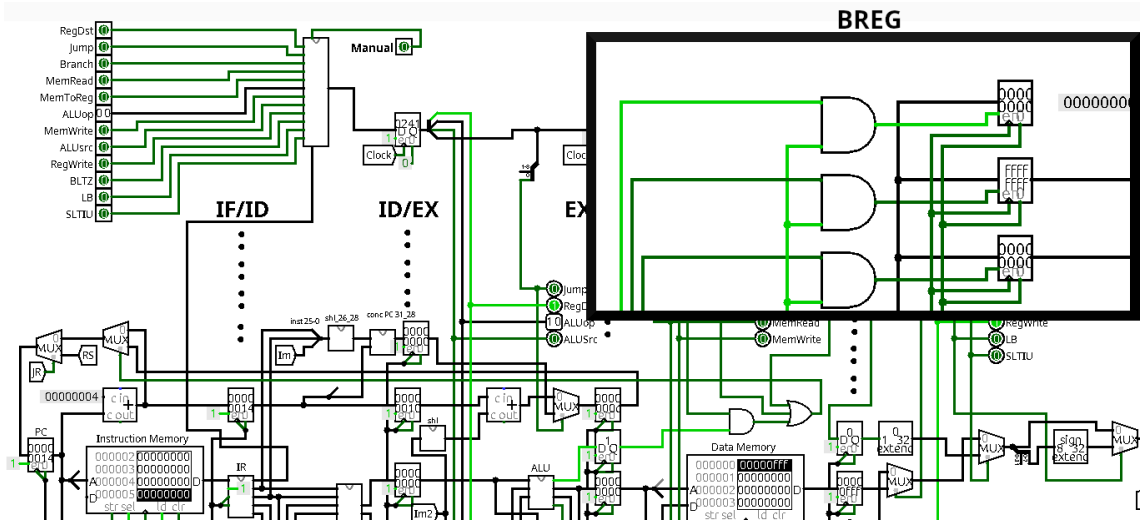

Figura 27: MIPS Pipeline - Teste LB

- Teste da Instrução SLTIU:

  SLTIU: Reg1 ← Reg0 < IMED (1)

  0x2C010001; Reg1 ← Reg0 < 1 (Reg1 = 1)

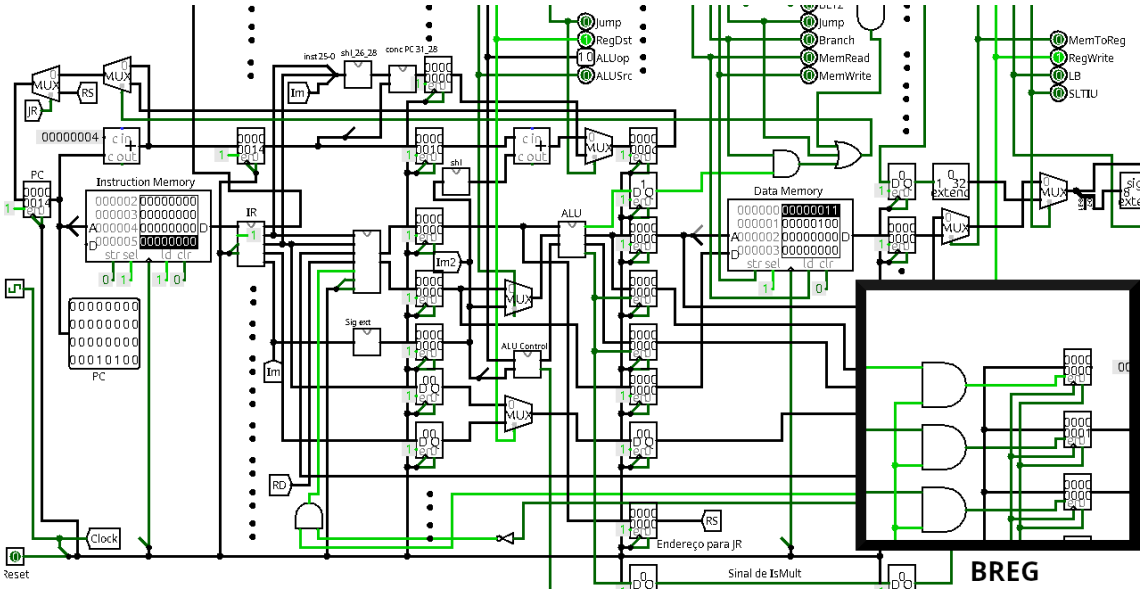

Figura 28: MIPS Pipeline - Teste SLTIU