Rodrigo Henriky de Assis Oliveira

MICROPROCESSADOR MIPS

CUIABÁ

2024

Rodrigo Henriky de Assis Oliveira

MICROPROCESSADOR MIPS

Trabalho apresentado ao Instituto Federal de Mato Grosso como requisito para a matéria de microprocessador.

Professor: Ruy de Oliveira

CUIABÁ

2024

RESUMO

O MIPS é uma arquitetura de microprocessadores baseada no conceito RISC (Reduced Instruction Set Computer), que utiliza um conjunto reduzido de instruções simples e eficientes. Ele possui 32 registradores de propósito geral e suporta execução em pipeline para maior desempenho.

As instruções do MIPS são classificadas em três formatos:

1. **Formato R (Register)**: Para operações entre registradores. Exemplos:
   - add rd, rs, rt: Soma os valores de rs e rt, armazenando o resultado em rd.
   - sub rd, rs, rt: Subtrai rt de rs, armazenando o resultado em rd.
2. **Formato I (Immediate)**: Para operações com valores imediatos e acesso à memória. Exemplos:
   - addi rt, rs, imm: Soma o valor imediato imm ao conteúdo de rs e armazena em rt.
   - lw rt, offset(rs): Carrega na memória o valor no endereço offset + rs para rt.
3. **Formato J (Jump)**: Para desvios e saltos. Exemplos:
   - j address: Salta para o endereço especificado.
   - jal address: Salta e armazena o endereço de retorno.

Principais instruções do MIPS:

- **Aritméticas**: add, addi, sub, mult, div.
- **Lógicas**: and, andi, or, ori, xor, xori, nor.
- **Deslocamento**: sll, srl, sra.
- **Comparações**: slt, slti, sltu, sltiu.
- **Controle de Fluxo**: beq, bne.
- **Acesso à Memória**: lw, sw.
- **Saltos**: j, jal, jr, jalr.

O MIPS é amplamente utilizado em sistemas embarcados e dispositivos de alta performance devido à sua simplicidade e eficiência.

SUMÁRIO

1 INTRODUÇÃO

1.1 Contextualização

A arquitetura MIPS (Microprocessor without Interlocked Pipeline Stages) é uma das arquiteturas de conjunto de instruções mais influentes e amplamente estudadas no campo da engenharia de computadores. Desenvolvida nos anos 1980 como parte do movimento RISC (Reduced Instruction Set Computer), a MIPS foi projetada com o objetivo de simplificar as operações de hardware, melhorando a eficiência e o desempenho dos processadores.

Embora a MIPS tenha evoluído ao longo do tempo para incluir funcionalidades mais avançadas, como o pipeline e a execução fora de ordem, seu núcleo básico continua sendo uma ferramenta educacional essencial para estudantes e profissionais que desejam compreender os princípios fundamentais da arquitetura de computadores. Este trabalho explora a implementação básica da arquitetura MIPS utilizando o Logisim Evolution, uma ferramenta de simulação digital que permite a construção e análise de circuitos digitais.

1.2 Objetivos do Artigo

O principal objetivo deste artigo é documentar a implementação da arquitetura MIPS em um ambiente simulado, utilizando o Logisim Evolution. O foco está na criação de um processador MIPS básico, capaz de executar um conjunto de instruções fundamentais sem a complexidade adicional do pipeline. Especificamente, o artigo busca:

1. **Apresentar os conceitos básicos da arquitetura MIPS**, abordando sua estrutura e o conjunto de instruções utilizado.
2. **Detalhar o processo de implementação no Logisim Evolution**, incluindo a configuração da memória, banco de registradores, unidade de controle e ALU (Unidade Lógica e Aritmética).
3. **Demonstrar a execução de instruções básicas** (aritméticas, lógicas, de memória e de controle de fluxo) através de simulações no Logisim.
4. **Discutir as limitações e desafios encontrados durante a implementação**, bem como possíveis melhorias e expansões futuras do projeto.

5. **Contribuir para o aprendizado de estudantes e entusiastas** de arquitetura de computadores, fornecendo um exemplo prático de como construir um processador MIPS funcional em um ambiente de simulação.

2 ARQUITETURA E COMPONENTES

O conjunto MIPS é feito com uma arquitetura de 32 bits, porém como estaremos aqui fazendo apenas para fim didáticos com as funções simples, a arquitetura terá apenas 16 bits de largura.

Para entender sobre o MIPS é necessário saber sobre os seus componentes e de como eles estão organizados. Resumidamente ele pode ser separado por 6 grandes blocos que juntos formar a arquitetura como um todo.

2.1 Program Counter

Quando se fala de processadores no geral, o program counter sempre aparece sendo essencial e indispensável. Ele tem o papel de apontar o endereço da memória a ser usada que contém uma instrução. No final do ciclo de clock normalmente é somado 1 ao endereço que estava guardando para apontar para a próxima instrução.

2.2 Memória de instruções

Aqui é armazenado todas as instruções que o programa irá rodar, podendo ser previamente carregado em um arquivo com as instruções com código de máquina. Cada endereço dessa memória se refere a um instruções que no caso desse projeto são de 16 bits.

2.3 Banco de Registradores

No banco de registradores temos um conjunto de registradores que serve para auxiliar em operações e armazenar dados temporários dependendo de cada instrução. Pode-se colocar valores previamente em algum dos registradores, porém diferentemente da memória de instruções aqui durante a execução do programa

valores podem ser escritos dentro dos registradores do mesmo jeito que pode ser lidos.

2.4 ALU

A unidade lógica aritmética ou em inglês arithmetic logic unit (ALU), é onde é feito todas as operações lógicas e aritméticas das instruções, presente em quase todas as instruções.

2.5 Memória de Dados

Na memória de Dados é armazenado os valores em definitivo das operações a qual o usuário deseja guardar, sendo ele volátil, ou seja, caso o programa seja encerrado de maneira indevida, os dados que estavam na memória se perderão.

2.6 Controlador

A unidade de controle, é fundamental para qualquer processador. É a parte responsável por controlar entradas e saída de dados de cada bloco dependendo de cada instrução. Mesmo sendo essencial é possível o programa rodar sem ele, porém os dados teriam que ser direcionados manualmente, o que demanda mas tempo porém é melhor para o aprendizado.

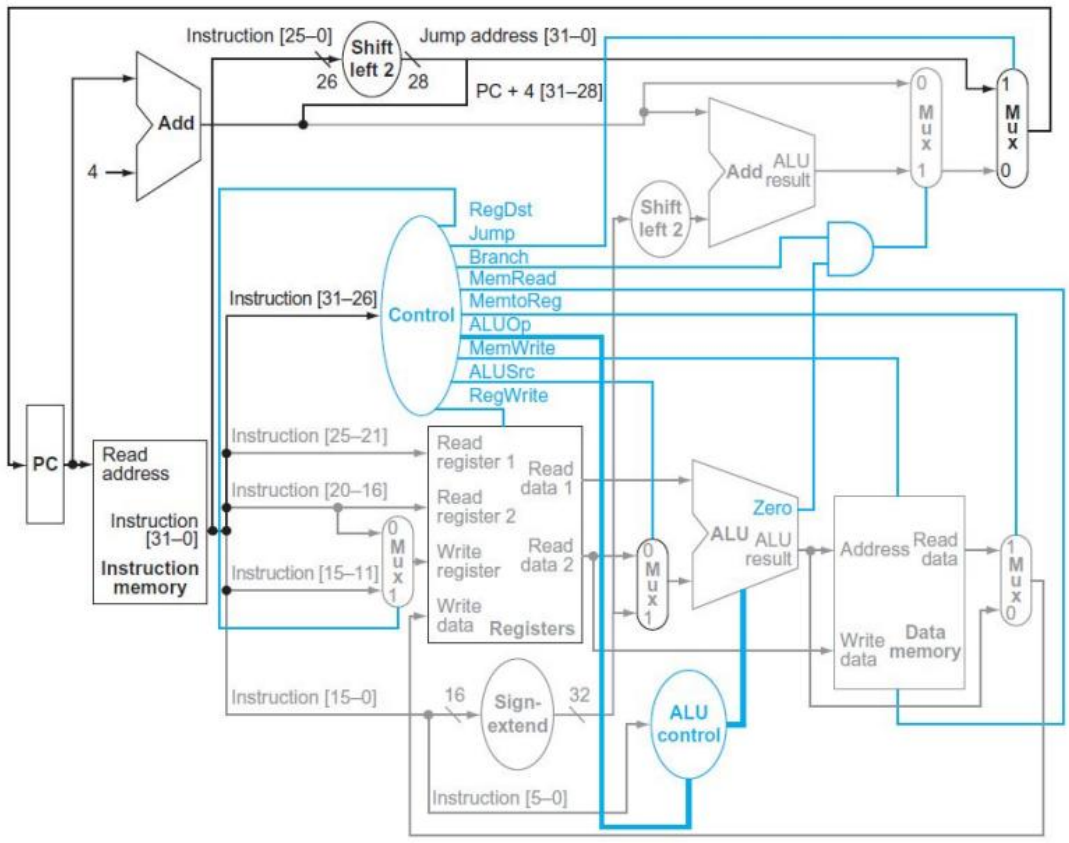


Figura 1: Arquitetura MIPS resumida

3 INSTRUÇÕES

As instruções do MIPS se dividem em três tipos (R, I e J) e cada uma tem funcionalidades diferentes e usam números de registradores diferentes.

As instruções do tipo R, são as instruções mais básicas de lógicas e aritméticas, onde pega um valor em um registrador faz a operação com o valor de outro registrador e escreve o resultado em um terceiro registrador, sendo assim utilizando três registradores dentro da sua instrução. Ao todo serão 8 tipos de instruções do tipo R: ADD, SUB, AND, OR, XOR, NAND, NOR e SLT. O que define o que será cada uma é a parte chamada funct que será explicada mais a frente.

Instruções do tipo I são aquelas que usam acesso direto a algum endereço ou usando esse valor como constante chamado também de valor imediato. No geral são 5 instruções do tipo I: LW, SW, ADDI, BEQ e SLTI. Diferente do tipo R o tipo I usam

apenas 2 registradores do banco de registradores e nos casos LW e SW também usam a memória.

Por último temos do tipo J que trás apenas uma instrução, JUMP, responsável por pular para o endereço indicado por ela mesmo, ou seja, em vez das instruções serem executadas de maneira sequencial, quando chega no JUMP ele realizará um salto na memória podendo ir para frente ou voltar na memória de instruções.

3.1 Formato das instruções

Para falar sobre cada instrução utiliza-se uma linguagem pra definir o que é cada instrução, chamamos essa linguagem de assembly. Para podermos representarmos cada instrução é usado o mnemônico (add, sub, lw, sw ...) e em seguida os endereços ou constantes a ser utilizados.

Para a arquitetura MIPS existe uma simbologia para o que estamos acessando: quando tem o símbolo '$' junto com um número trata-se de um registrador que em específico nesse trabalho possui apenas 8 ou seja os registradores vão de $0 até $7, como está sendo tratado de registradores foi acrescentado a letra r no meio ficando $r0 até $r7 os registradores.

Quando estamos falando de um valor imediato ou endereço de algum registrador é colocado apenas o esse valor sem simbologia nenhuma antes e nem depois.

Usa-se uma sintase diferente caso a instrução peça um acesso a memória, que no nosso caso refere-se as instruções LW e SW. Para falar o endereço da memória a ser acessado é colocado uma constante junto com um registrador dentro de um parênteses, com isso é falado que o endereço é o resultado da soma da constante com o valor dentro do registrador.

Para mostrar o que foi falado aqui vai um exemplo: add $r0, $r1, $r2, essa é uma instrução do tipo R que retrata a seguinte operação -> ($r0) = ($r1) + ($r2), ou seja, o registrador r0 irá receber a soma dos conteúdos dos registradores r1 e o r2 respectivamente. Outro exemplo podemos ver em ->lw $r1, 5($r3), a instrução lw carrega da memória determinado valor, portanto nessa instrução temos que no

registrador r1 receberá o valor que está na memória no endereço 5 +($r3), caso no registrador r3 estivesse o valor 8, por exemplo, então r1 iria receber o conteúdo da memória na posição 13 (5+8).

A tabela a seguir mostra todas as instruções com os seus respectivos formatos

Tipo	Instrução	Assembly	descrição
R	ADD	add $ra,$rb, $rc	($ra) = ($rb) + ($rc)
	SUB	sub $ra,$rb, $rc	($ra) = ($rb) - ($rc)
	AND	and $ra,$rb, $rc	($ra) = ($rb) ^ ($rc)
	OR	or $ra,$rb, $rc	($ra) = ($rb) ᵛ ($rc)
	NAND	nand $ra,$rb, $rc	($ra) = ($rb) $\bar{\wedge}$ ($rc)
	NOR	nor $ra,$rb, $rc	($ra) = ($rb) $\bar{\vee}$ ($rc)
	XOR	xor $ra,$rb, $rc	($ra) = ($rb) $\oplus$ ($rc)
	SLT	slt $ra,$rb, $rc	($rb) < ($rc), ($ra) = 1; ($ra) = 0
I	LW	lw $ra, k ($rb)	$ra = (M(k + rb))
	SW	sw $ra, k ($rb)	M(k + rb) = ($ra)
	ADDI	addi $ra,$rb, k	($ra) = ($rb) + k
	BEQ	beq $ra,$rb, k	($rb) == ($ra) , (pc + 1) + k
	SLTI	sltii $ra,$rb, k	($rb) < k, ($ra) = 1; ($ra) = 0
J	JUMP	J k	Pc = k

Tabela 1: Instruções a serem usadas no processador

3.2 Convertendo para o código de máquina

Se rodássemos um programa contendo apenas as instruções em assembly, ele não rodaria, pois não há nada que traduza o código para o computador entender. Portanto, iremos transformar a instrução em executável para o computador, o que muda para cada função, tendo em comum o tamanho de 16 bits. Para cada instrução formada, também se forma uma palavra, como é chamado.

A tabela a seguir mostra como cada palavra é separada de acordo com o seu tipo:

Tipo	Opcode	Rs	Rt	Rd	funct
R	3 bits	3 bits	3 bits	3 bits	4 bits

	Opcode	**Rs**	**Rt**	**Immediate/address**
I	3 bits	3 bits	3 bits	7 bits
	Opcode	**addresss**		
J	3 bits	13 bits		

Tabela 2: organização das instruções

As instruções do tipo R são utilizadas para operações aritméticas e lógicas que envolvem apenas registradores. Este tipo de instrução é composto pelos seguintes campos:

- **Opcode (4 bits)**: Identifica o tipo geral de operação a ser realizada. No caso das instruções do tipo R, o opcode especifica que a instrução é do tipo R.
- **rs (3 bits)**: Registrador fonte. Contém o primeiro operando para a operação.
- **rt (3 bits)**: Registrador fonte. Contém o segundo operando para a operação.
- **rd (3 bits)**: Registrador de destino. Onde o resultado da operação será armazenado.
- **funct (3 bits)**: Campo de função. Especifica a operação exata (como adição, subtração, etc.) a ser realizada pela ALU (Unidade Lógica e Aritmética).

As instruções do tipo I são utilizadas para operações que envolvem um registrador e um valor imediato, ou para operações de acesso à memória. Este formato é composto por:

- **Opcode (3 bits)**: Identifica o tipo de operação a ser realizada.
- **rs (3 bits)**: Registrador fonte. Contém o operando que será usado na operação ou o endereço base para operações de carga/armazenamento.
- **rt (3 bits)**: Registrador de destino. Onde o resultado da operação será armazenado ou o registrador que contém o dado a ser armazenado na memória.
- **Immediate/address (7 bits)**: Um valor imediato usado diretamente na operação ou um endereço utilizado para operações de acesso à memória.

As instruções do tipo J são utilizadas para saltos incondicionais, onde o fluxo de execução é desviado para um novo endereço. Este formato é composto por:

- **Opcode (3 bits)**: Identifica que a instrução é um salto (jump).
- **Address (13 bits)**: Especifica o endereço para onde o controle deve saltar.

Para exemplificar a próxima tabela trará um exemplo de cada instrução e de como ele ficaria em código de máquina

TIPO	Assembly		Código de máquina				
	Operação	**Código**	**Opcode**	**Rs**	**Rt**	**Rd**	**funct**
R	ADD	$r1, $r2, $r3	0000	010	011	001	0000
	SUB	$r1, $r2, $r3	0000	010	011	001	0001
	AND	$r1, $r2, $r3	0000	010	011	001	0010
	OR	$r1, $r2, $r3	0000	010	011	001	0011
	NAND	$r1, $r2, $r3	0000	010	011	001	0100
	NOR	$r1, $r2, $r3	0000	010	011	001	0101
	XOR	$r1, $r2, $r3	0000	010	011	001	0110
	SLT	$r1, $r2, $r3	0000	010	011	001	0111
I	LW	$r1, 10($r2)	0100	010	001	0001010 (address)	
	SW	$r1, 10($r2)	0101	010	001	0001010 (address)	
	ADDI	$r1, $r2, 10	0111	010	001	0001010 (immediate)	
	BEQ	$r1, $r2, 10	0110	010	001	0001010 (immediate)	
	SLTI	$r1, $r2, 10	0001	010	001	0001010 (immediate)	
J	JUMP	J 10	0010	1010			

Tabela 3: exemplo de cada instrução em código de máquina

4 CONSTRUINDO O MIPS NO LOGISIM

Agora que ja se sabe como funciona as instruções desse nosso projeto, iremos implementar usando um software chamado Logisim evolution. Para isso é necessário retomar sobre cada componente.

A começar com o program counter que nada mais é que um registrador de 6 bits que guarda o endereço da posição da memória de instruções a ser usada, o que é necessário para acessar uma memória de 32 registradores.

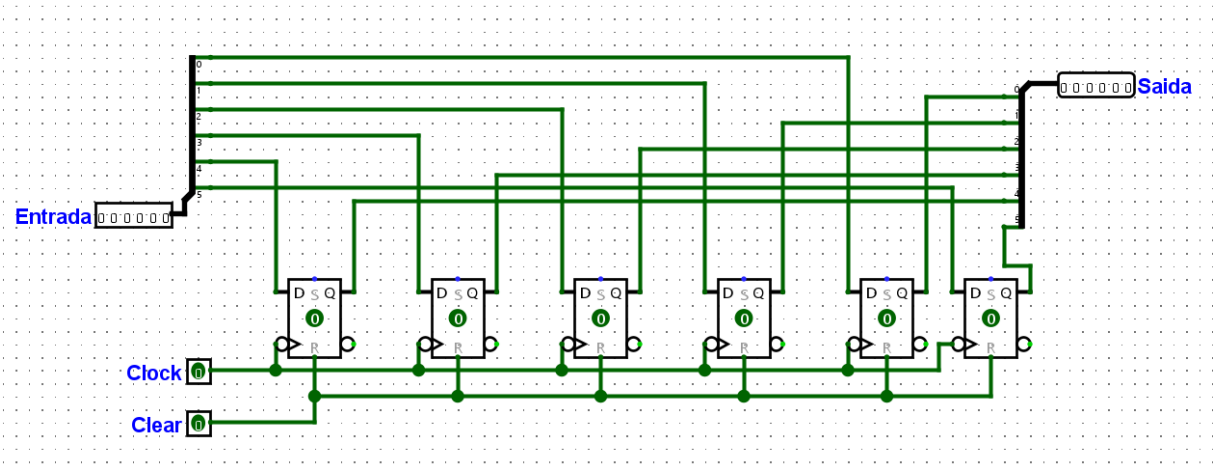


Figura 2: estrutura do PC ou program counter no Logisim Evolution.

A figura acima demonstra a implementação do program counter onde foi usado flips flops do tipo d (ou transparente) para armazenar os bits do endereço a ser acessado. É necessário entender também cada pino que possuem ai seja de entrada ou saida:

Entrada: é um pino de entrada 5 bits que contém o endereço da próxima instrução a ser acessada.

Saída: é um pino de saída de 5 bits que libera o conteúdo que tem nos flips flops e que externamente é conectada com a memória de instruções.

Clock: é um pino de entrada onde é conectado o clock, o qual serve para habilitar o conteúdo dos registradores para a saida, toda vez que sair de nivel alto para nivel baixo o conteúdo é liberado.

Clear: é um pino de entrada o qual, enquanto estiver em nivel alto irá deixar os flips flops deixando o conteúdo em 0.

Externamente temos algo parecido com a imagem logo abaixo:

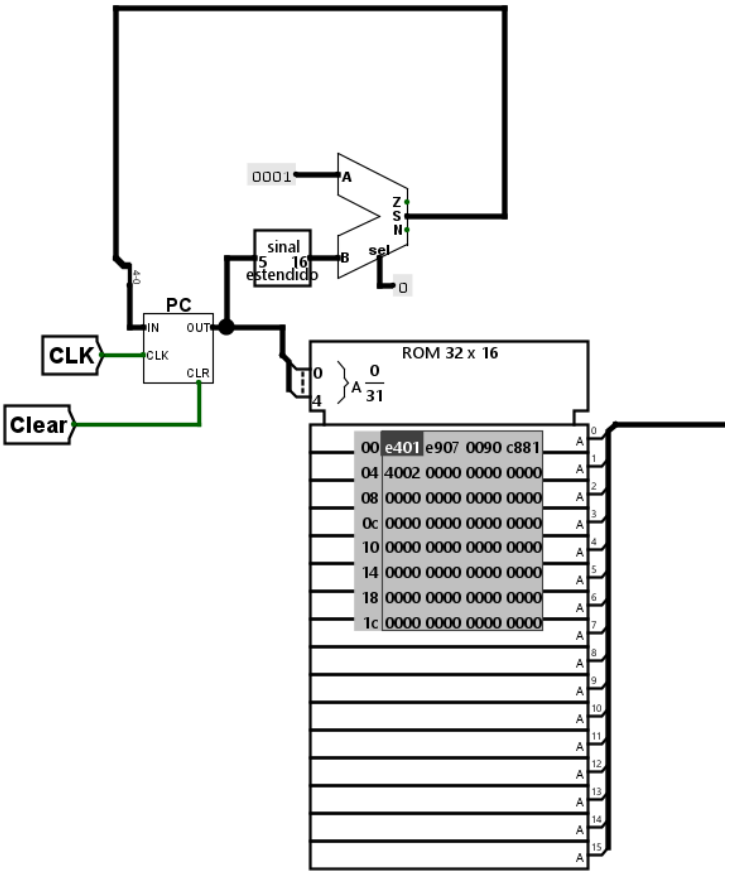


Figura 3: Implementação do program counter junto com a memória de instruções

Para incrementar o program counter utilizamos a ULA (elemento que veremos mais a frente) usado para somar mais um e voltar para entrada o valor do próximo endereço, assim toda vez que tivermos um clock teremos um endereço novo a ser acessado.

É possível ver também que a saída do PC está conectado com a memória de instruções.

A seguir veremos o funcionamento do banco de registradores:

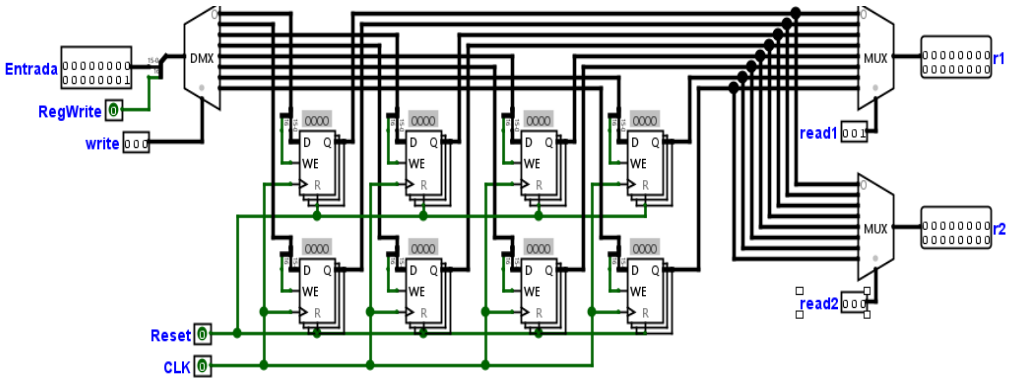


Figura 4: estrutura do banco de registradores 8x16, 8 registradores de 16 bits

Entrada: é um pino de entrada de 16 bits que contém a instrução a ser guardada em um dos 8 registradores.

RegWrite: é um pino de entrada que define se o registrador selecionado irá ativar a função de escrever nele ou não.

Write: pino de entrada de 3 bits que seleciona o registrador a ser escrito.

Read1: pino de entrada de 3 bits que seleciona qual registrador será selecionado na saída r1.

Read2: pino de entrada de 3 bits que seleciona qual registrador será selecionado na saída r2.

R1 e R2: pinos de saída de 16 bits que as instruções dos registradores selecionados para fazer a operação desejada externamente.

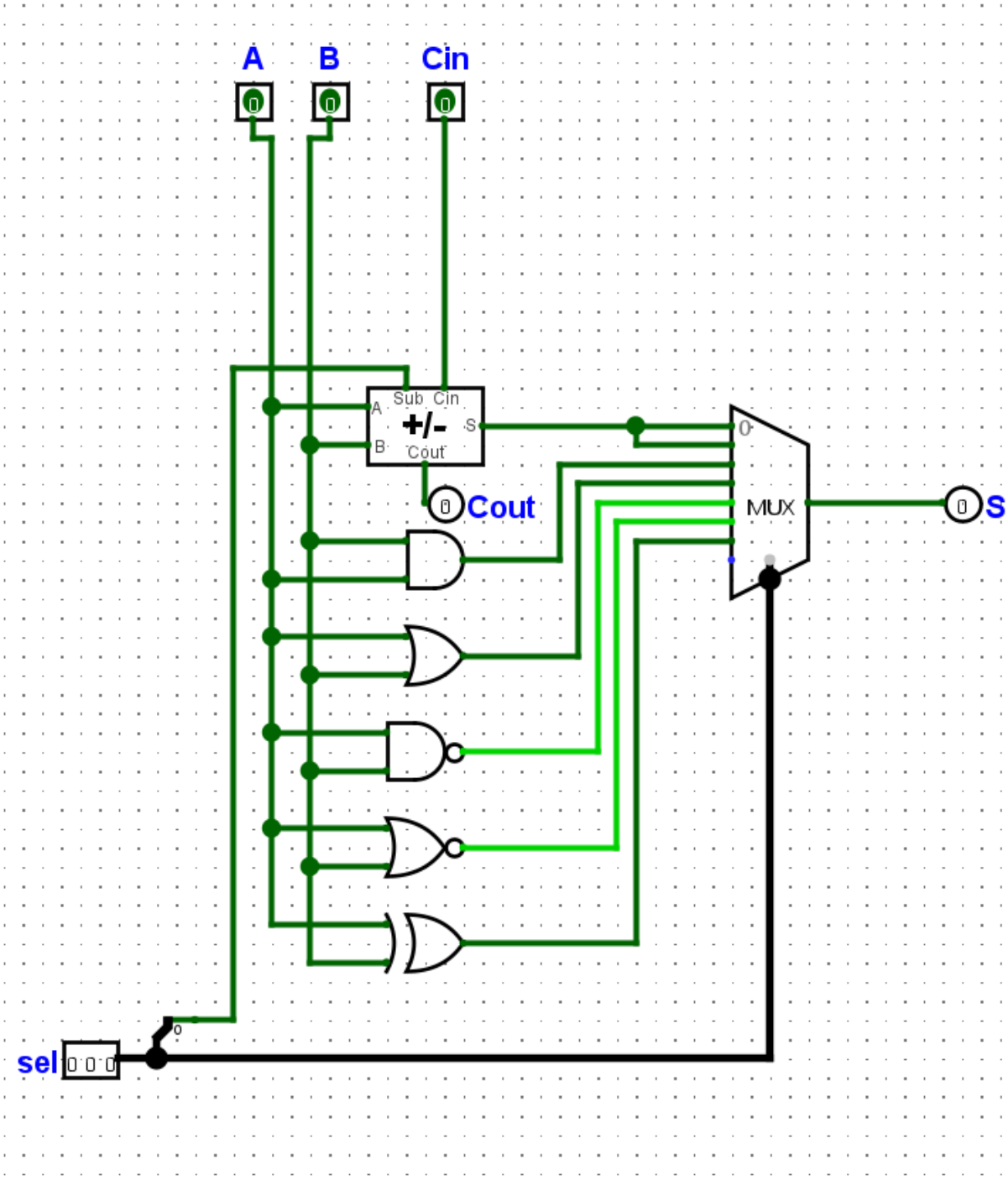


Figura 5: estrutura da ALU de um bit

Dando continuidade na construção do processador vamos implementar um dos principais componentes para o funcionamento geral que é a ULA ou ALU como é conhecido também. Na figura 5 temos a implementação da ALU de 1 bit que é necessário para fazer uma de 16 bits essa estruturação está de acordo com a

Tabela 3 na coluna funct que é definida qual operação é feita. Como é possível ver também temos também um multiplex que seleciona qual vai ser a saída ele possui de entrada de seleção de 4 bits que é exatamente o numero de bits que tem o funct, com isso pode-se ter até 16 operações lógicas e aritméticas, porém temos apenas 8, por isso nas entradas do multiplex está preenchido apenas de 0 a 7 (0000 a 0111), podendo então ter até mais 8 operações se necessárias. Com isso temos o necessário para fazer a ALU de 16 bits como mostra a figura a seguir:

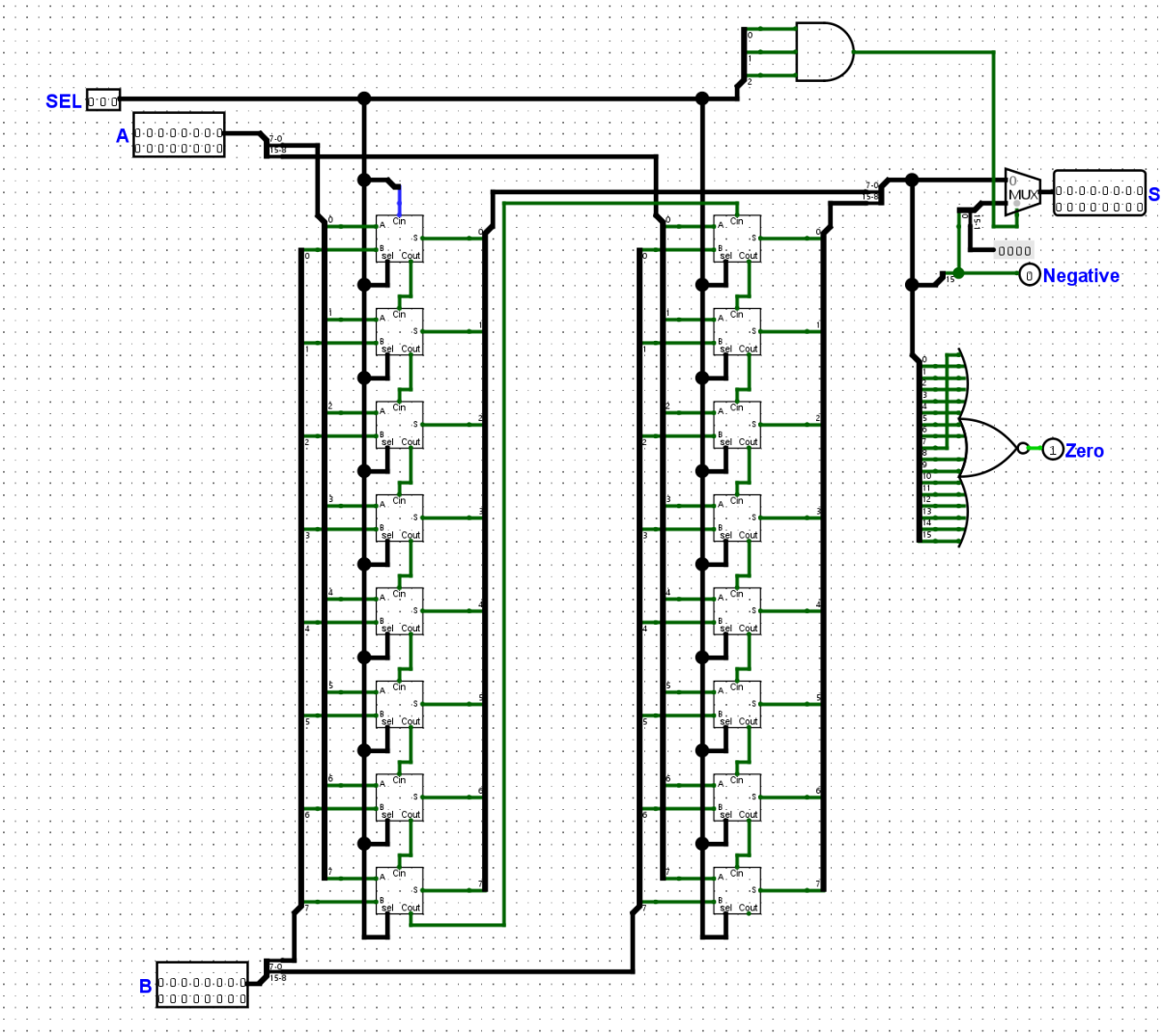


Figura 6: estrutura da ALU de 16 bits

Aqui na figura 6 além das ALUs de um bit tem uma porta and em cima ela serve exclusivamente para a função slt que reflete na saída, onde em vez de sair o resultado

da operação vai sair 1 caso o resultado de negativo e 0 caso não. Além disso temos as entradas e saídas:

Sel: pino de entrada de 4 bits que seleciona a operação desejada.

A: pino de entrada pino de entrada de 16 bits que contém o primeiro dado da operação.

B: pino de entrada pino de entrada de 16 bits que contém o segundo dado da operação.

Negative e Zero: pinos de saídas indicadores para os resultados da operação ser negativo ou zero respectivamente.

S: pino de saída de 16 bits que contém o resultado da operação correspondente.

É importante ressaltar que o que estiver na função funct não será conectado diretamente na ALU pelo fato de essa parte funct ter apenas nas instruções do tipo R, visto isso tem que ter um controle antes da entrada de seleção da ALU porque tem instruções que usam a ALU sem ser do tipo R, como por exemplo LW, entre outros. Assim temos o controlador da ALU na imagem a seguir:

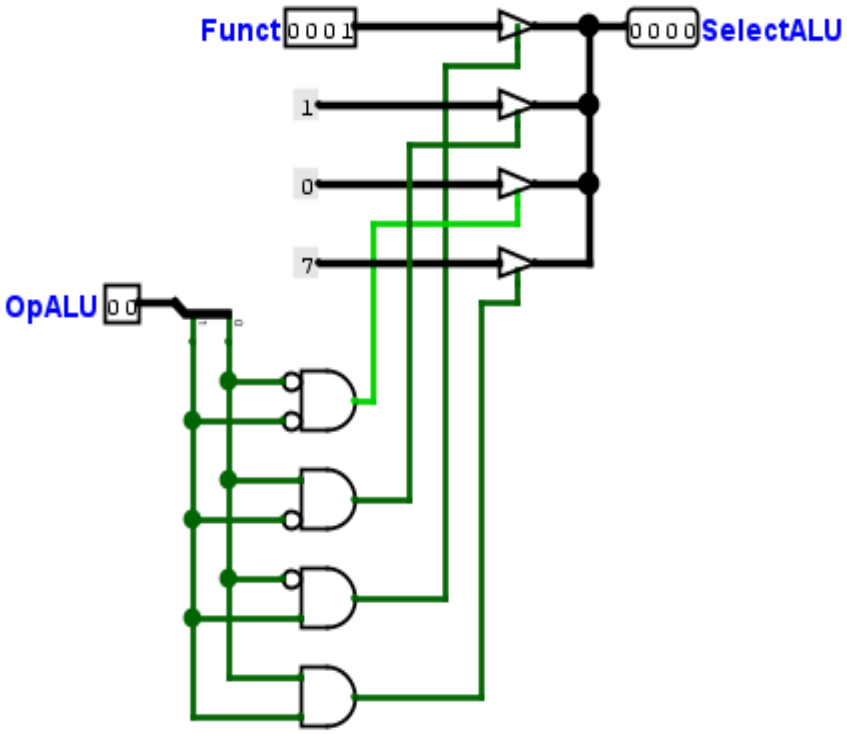


Figura 7: estrutura do Controle da ALU

Na figura 7 temos o controle da ALU, para definir a operação a ser feita, o que irá decidir isso será a OpALU que controla qual a saída. De acordo com a seguinte tabela:

Op	Op ALU1	Op ALU0	Op desejada	Entrada ALU
Tipo R	1	0	Tipo R	Funct
Lw	0	0	Add	0000
Sw	0	0	Add	0000
Addi	0	0	Add	0000
Beq	0	1	Sub	0001
slti	1	1	slt	0111

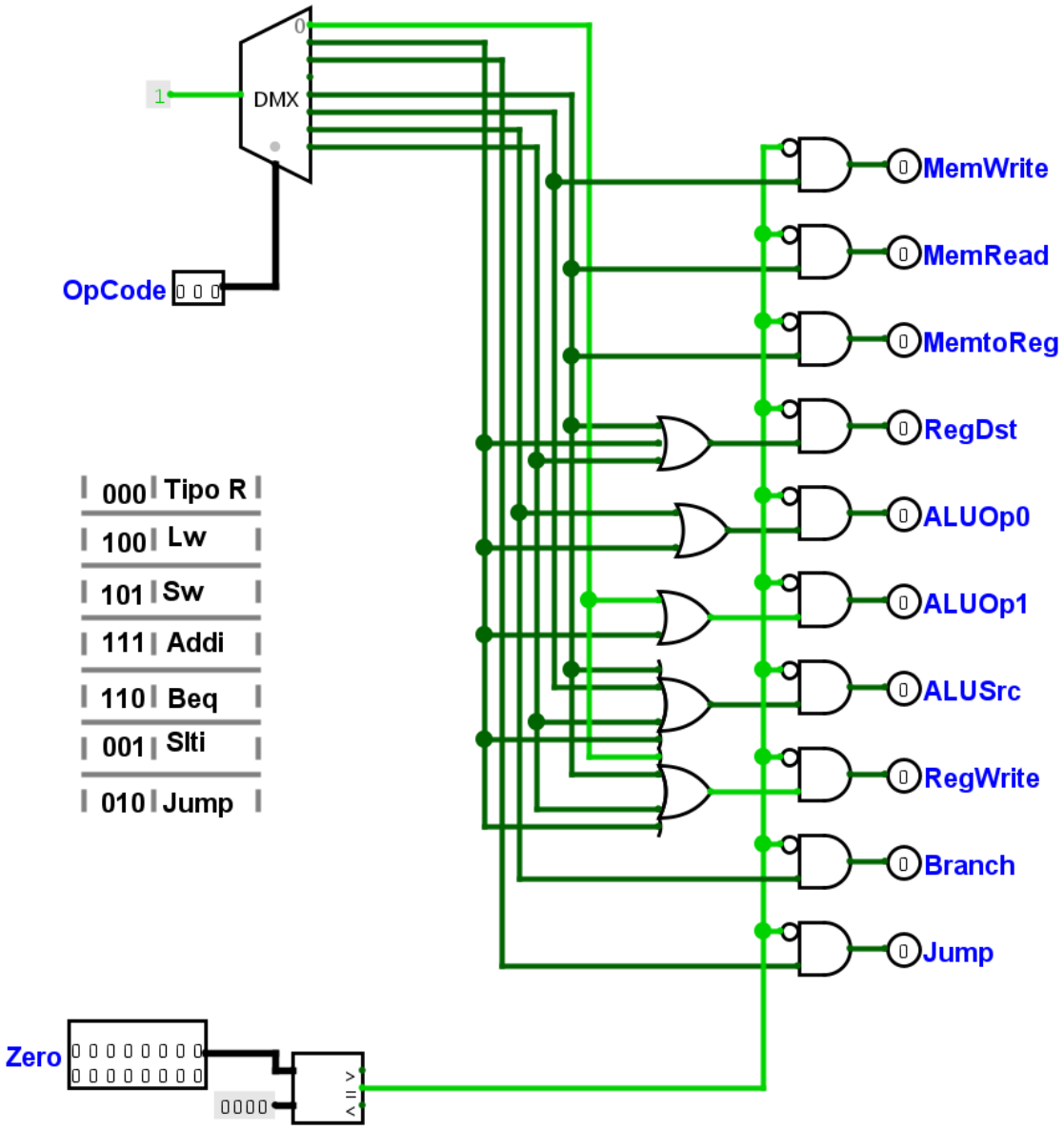
Tabela 4: tabela que define o que faz cada opALU e a sua respectiva saída

De acordo com a tabela 4 cada instrução vai ativar uma saída diferente do controle da ALU no caso da instrução jump não importa a entrada pois a ALU não é usada

Depois de dito tudo isso vamos para o controle que define o que é acionado ou não de cada entrada, no total temos 10 saídas que o controlador controla:

Controle	Definição
MemRead	Habilita leitura da memória
MemWrite	Habilita escrita na memória
MemtoReg	Caso 0 passa a saída da ALU para os registradores, caso 1 a saída da memória
RegDst	Caso 0 passa os bits 4-6 para selecionar o registrador da escrita dos registradores, caso 1 passa os bits 7-9
Regwrite	Habilita escrita dos registradores
OpALU0	Define a operação no controlador ALU
OpALU1	Define a operação no controlador ALU
ALUSrc	Define a segunda entrada de dados da ALU, caso 0 a entrada vai vir dos registradores, caso 1 seleciona da própria instrução
Branch	Seleciona a saída para o PC, caso 0 tudo normal, caso 1 vai ser adicionado o valor que esta na instrução
Jump	Seleciona se terá um salto no PC ou não, caso 0 tudo normal, caso 1 haverá salto

E assim fica o controlador depois de todas as mudanças:


0
1
DMX
OpCode
000 Tipo R
100 Lw
101 Sw
111 Addi
110 Beq
001 Slti
010 Jump
MemWrite
MemRead
MemtoReg
RegDst
ALUOp0
ALUOp1
ALUSrc
RegWrite
Branch
Jump
Zero
0000
> = <

Assim pode acionar todos os controles conforme o programa for solicitado

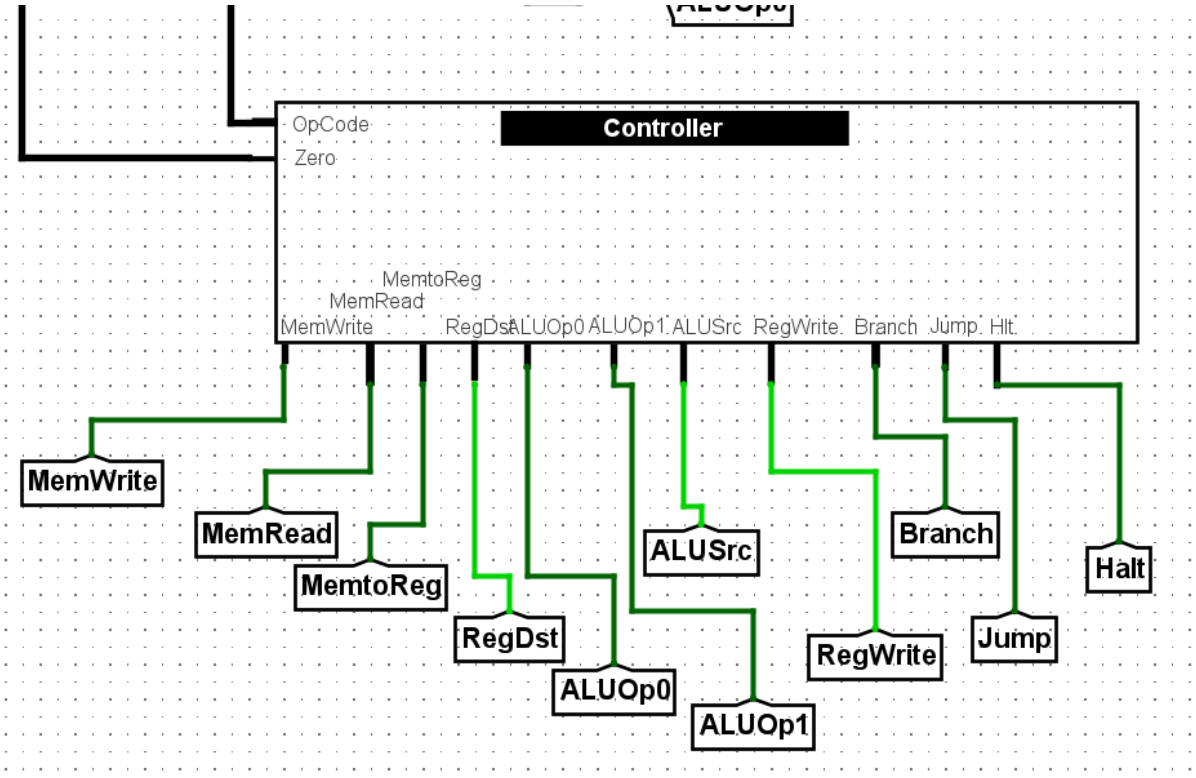


Tabela 7: Instruções Realizada no Mips com controle Automático

5 CONCLUSÃO

No final temos:

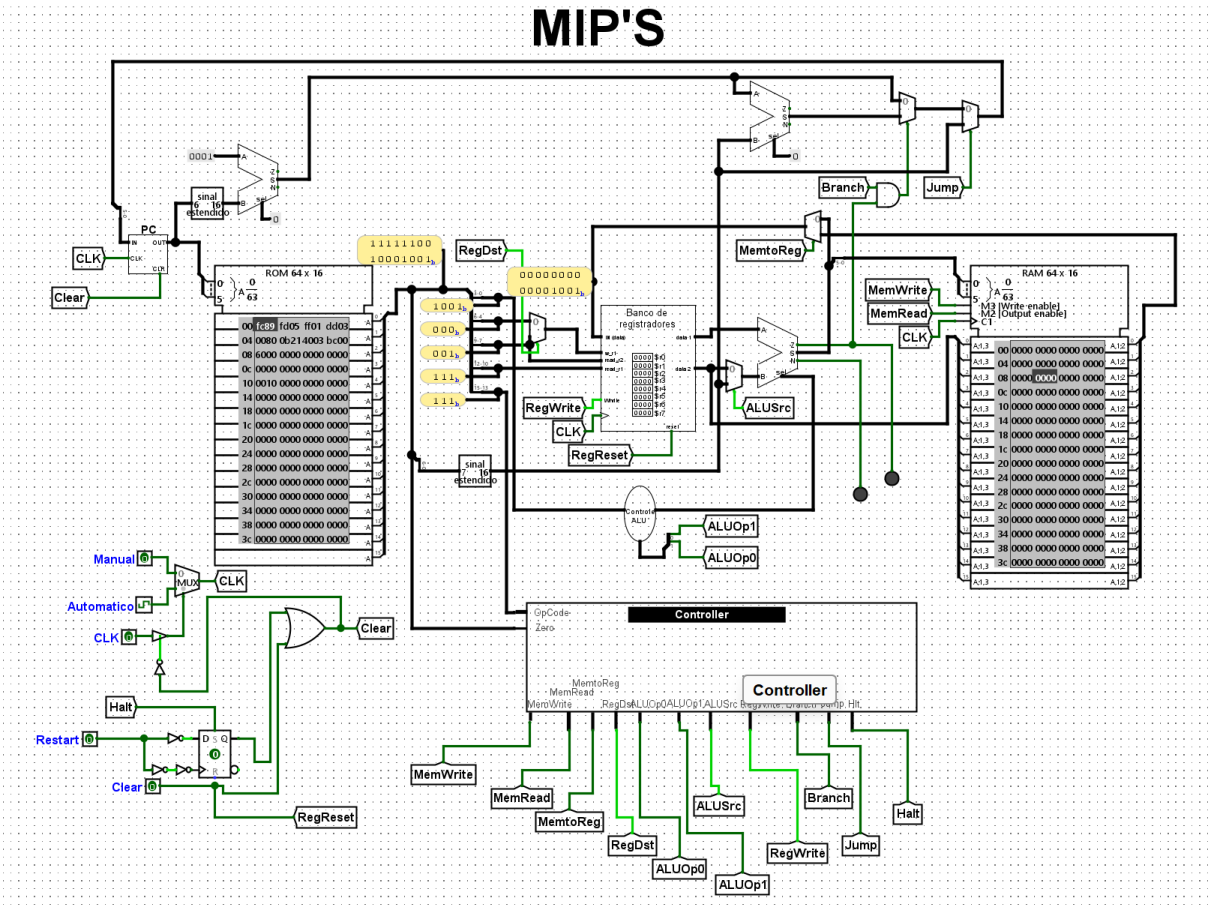


Tabela 7: Mips com Controle Automático