UNIVERSIDADE FEDERAL DE RORAIMA
CENTRO DE CIÊNCIAS E TECNOLOGIA
DEPARTAMENTO DE CIÊNCIA DA COMPUTAÇÃO
CURSO BACHAREL EM CIÊNCIA DA COMPUTAÇÃO
ARQUITETURA E ORGANIZAÇÃO DE COMPUTADORES

AMANDA DE BRITO BARBOSA
FÁBIO AURÉLIO BARROS ALEXANDRE
JOSÉ CARVALHO NETO

LABORATÓRIO DE CIRCUITOS - CODIFICAÇÃO E SIMULAÇÕES

BOA VISTA – RR
2025

AMANDA DE BRITO BARBOSA
FÁBIO AURÉLIO BARROS ALEXANDRE
JOSÉ CARVALHO NETO

LABORATÓRIO DE CIRCUITOS - CODIFICAÇÃO E SIMULAÇÕES

Relatório Científico apresentado ao Prof. Dr. Herbert Oliveira Rocha, com objetivo de obtenção de nota parcial para aprovação na disciplina DCC 301 - Arquitetura e Organização de Computadores, do Departamento de Ciência da Computação da Universidade Federal de Roraima.

BOA VISTA – RR
2025

SUMÁRIO

1 INTRODUÇÃO

Este documento apresenta uma série de circuitos digitais desenvolvidos como parte da primeira avaliação da disciplina **Arquitetura e Organização de Computadores**. O objetivo principal deste trabalho foi construir, de forma gradual e detalhada, os componentes fundamentais que formam a base estrutural de um processador.

Todos os circuitos foram **projetados e testados utilizando o software Logisim**, permitindo a análise do comportamento lógico de cada componente e a verificação prática de seu funcionamento por meio de simulações.

Seguindo as orientações da atividade, cada componente é descrito com base em quatro aspectos fundamentais: Descrição, imagem do circuito integrado, testes realizados e resultados obtidos.

O relatório técnico busca detalhar ao máximo o processo de implementação e validação de cada circuito, destacando a importância da compreensão das operações lógicas, da otimização de estruturas digitais e do funcionamento integrado dos módulos que compõem a arquitetura de um processador.

2 COMPONENTES

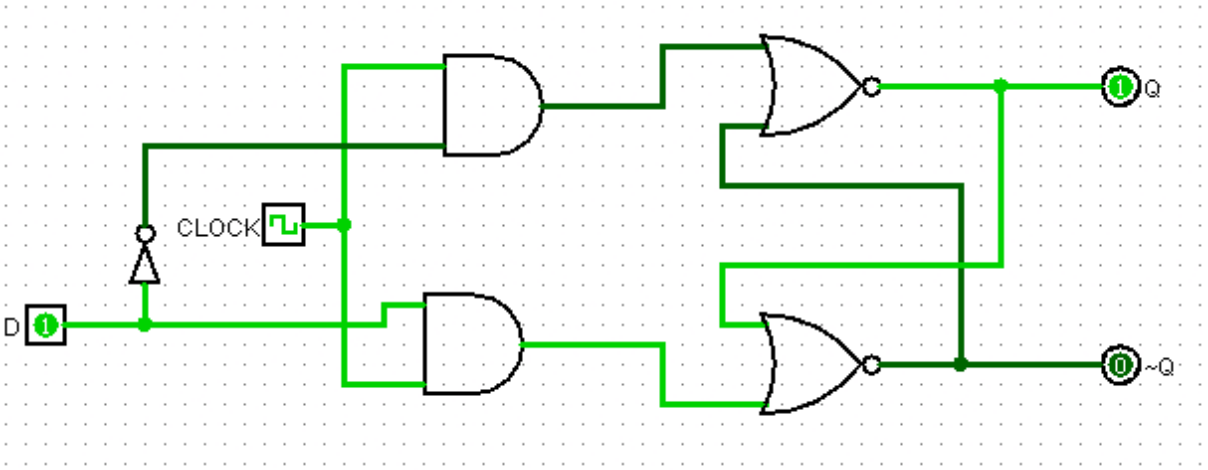
2.1 Flip-Flop

2.1.1 Flip-Flop Tipo D

O Flip-Flop D é o mais direto ao ponto. Pense nele como uma memória de "copia e cola". O que estiver na sua entrada D (Dados) é copiado para a saída Q no exato momento em que o CLOCK dá um pulso. Se o clock não pulsar, ele ignora a entrada e guarda o último valor que tinha.

Teste Prático:

1. Colocamos D = 1.
2. Enquanto o **CLOCK** está em 0, nada acontece. A saída **Q** continua como estava antes.
3. Quando o **CLOCK** pulsa (vai de 0 para 1), a saída **Q** instantaneamente se torna **1**, armazenando o valor.



2.1.2 Flip-Flop Tipo JK

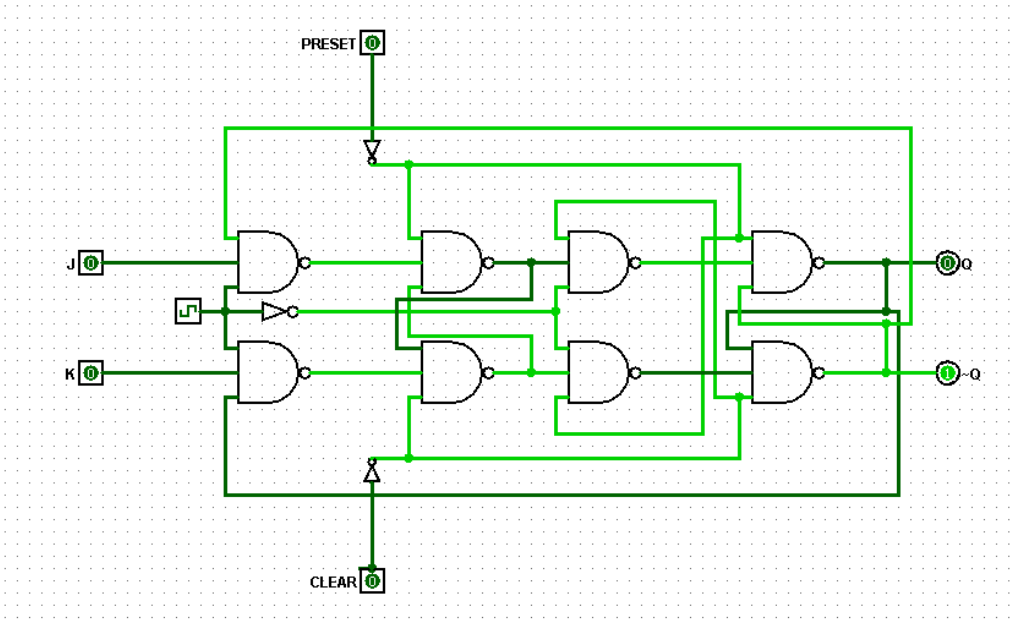
Com suas duas entradas, **J** e **K**, ele pode ser programado para fazer quatro coisas diferentes a cada pulso do clock:

J	K	Q
0	**0**	**Manter:** Não muda nada.
0	**1**	**Resetar:** Força a saída para 0.

1	**0**	**Setar:** Força a saída para 1.
1	**1**	**Bascular:** Inverte o valor atual.

Teste Prático:

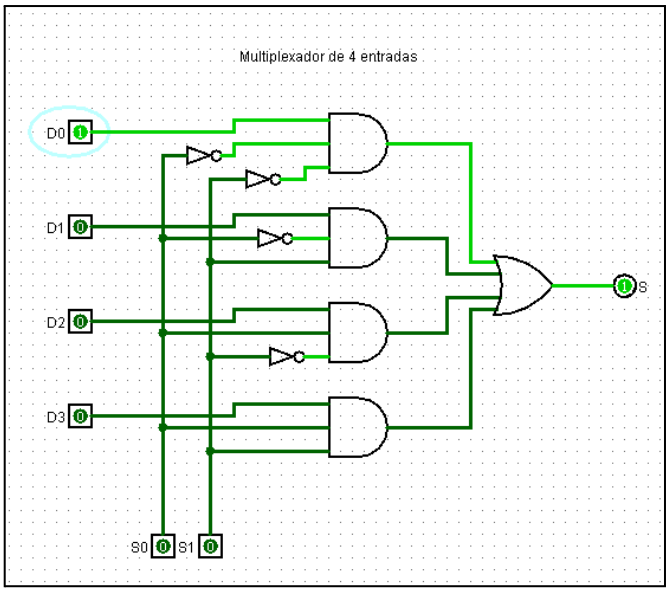
4. Imagine que a saída **Q** está em 0.
5. Configuramos **J = 1** e **K = 1.**
6. No próximo pulso do **CLOCK**, a saída **Q** inverte e vai para **1**. Se o **CLOCK** pulsar de novo, ela inverte para **0,** e assim por diante.



2. 2 Multiplexador de 4 bits

Um multiplexador é um circuito lógico que funciona permitindo que apenas uma de suas entradas seja enviada para a saída. Essa seleção é determinada por chaves seletoras, que conduzem os sinais para a saída esperada. No exemplo implementado, o multiplex apresenta 4 entradas (4 bits), 2 chaves (2 bits) e 1 saída (1 bit). Um das entradas D0, D1, D2 e D3 será enviada para a saída de acordo com as combinações das chaves seletoras. Podemos ver a possibilidades na seguinte tabela.

S0 (Chave 1)	S1 (Chave 2)	Saída
0	0	D0
0	1	D1
1	0	D2
1	1	D3



Para exemplificar, o teste será para a saída D0. Para isso, S0 é 0, passando pelo inversor, se torna 1. S1 é 0, passando pelo inversor se torna 1. A primeira porta AND recebe 1 do D0, portanto passa a receber os valores 1 1 1. Considerando a lógica da tabela verdade, uma porta AND tem saída como verdadeira quando todos os seus valores são verdadeiros, e falsa quando pelo menos um dos valores é falso. Nesse caso, a saída será verdadeira, pois todos os valores recebidos são verdadeiros, enquanto nas outras portas pelo menos 1 entrada foi 0 (falsa). A porta OR no fim do circuito, reúne as saídas das portas AND e tem saída verdadeira quando pelo menos 1 entrada for verdadeira. O mesmo acontece nas entradas D1, D2 e D3, com as respectivas combinações da chave.

2.3 Porta lógica XOR usando os componentes: AND, NOT, e OR

XOR é a porta que dá 1 só quando A e B são diferentes. Com AND/OR/NOT

Definição Lógica

A tabela verdade de uma porta XOR é a seguinte:

A	B	Y

0	0	0
0	1	1
1	0	1
1	1	0

Essa equação combina as entradas usando portas AND, OR e NOT.

Circuito simulado:

Pinos

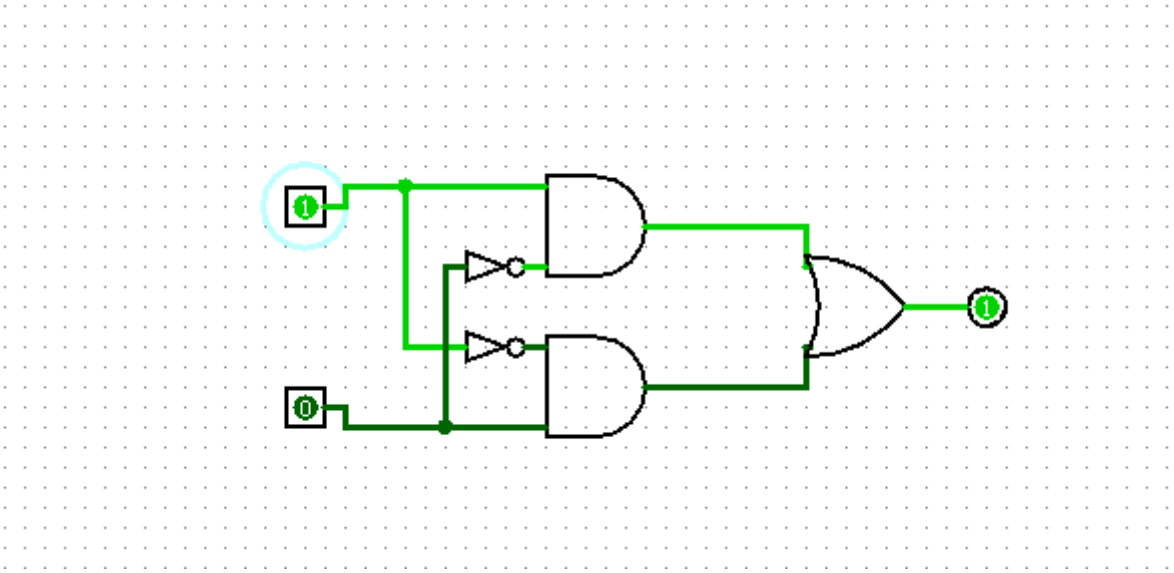
- Entradas: `A`, `B` (1 bit cada, nível lógico 0/1)
- Saída: `Y` (1 bit)

Função lógica

A saída é `1` somente quando as entradas são diferentes:
`Y = A ⊕ B`

Implementações equivalentes usando AND/OR/NOT

1. Soma de produtos:
   `Y = (A ∧ ¬B) ∨ (¬A ∧ B)`
2. Máscara do AND:
   `Y = (A ∨ B) ∧ ¬(A ∧ B)`



2.4 Somador de 8 bits que recebe um valor inteiro e soma com o valor 4

2.4.1 O Somador Completo de 1 Bit

Para entender como somamos números grandes, precisamos primeiro entender como somar apenas um dígito (ou, no nosso caso, um bit). O **Somador Completo de 1 Bit** é o bloco de construção fundamental para isso.

Pense nele como o processo de fazer uma conta de "vai um" no papel. Ele recebe três informações:

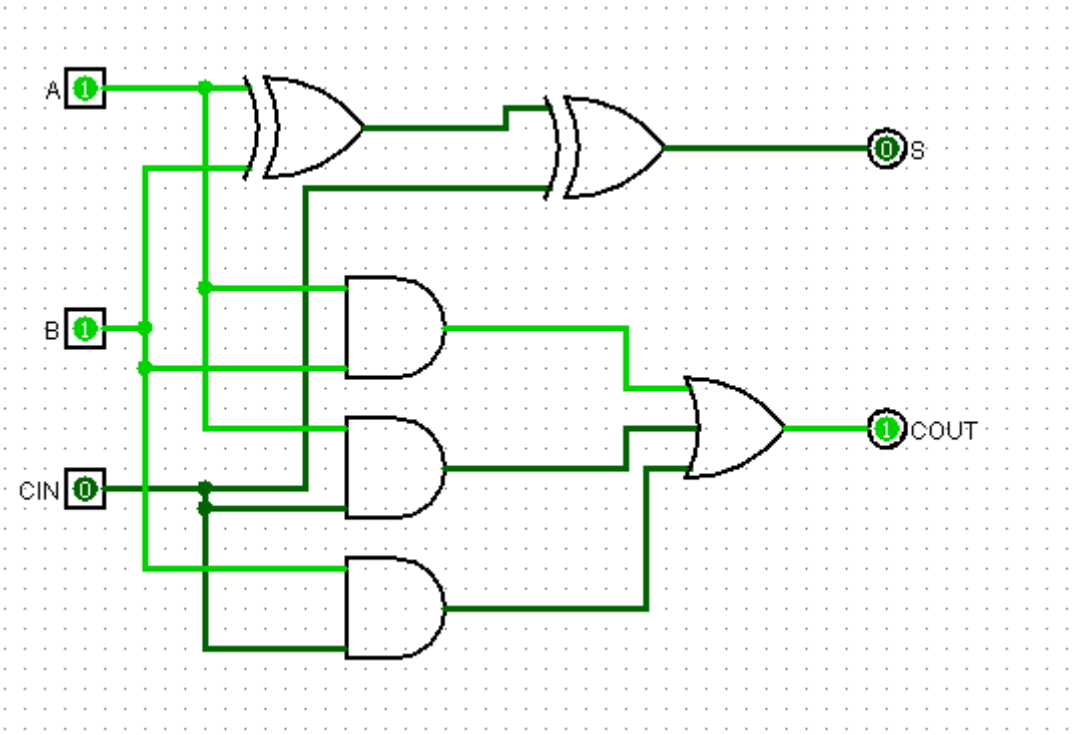
- O bit da **Entrada A**.
- O bit da **Entrada B**.
- O "vai um" que pode ter vindo da conta anterior (**Carry In**).

Com isso, ele produz duas saídas:

- O resultado da soma daquela coluna (**Soma**).
- O novo "vai um" que será enviado para a próxima coluna (**Carry Out**).

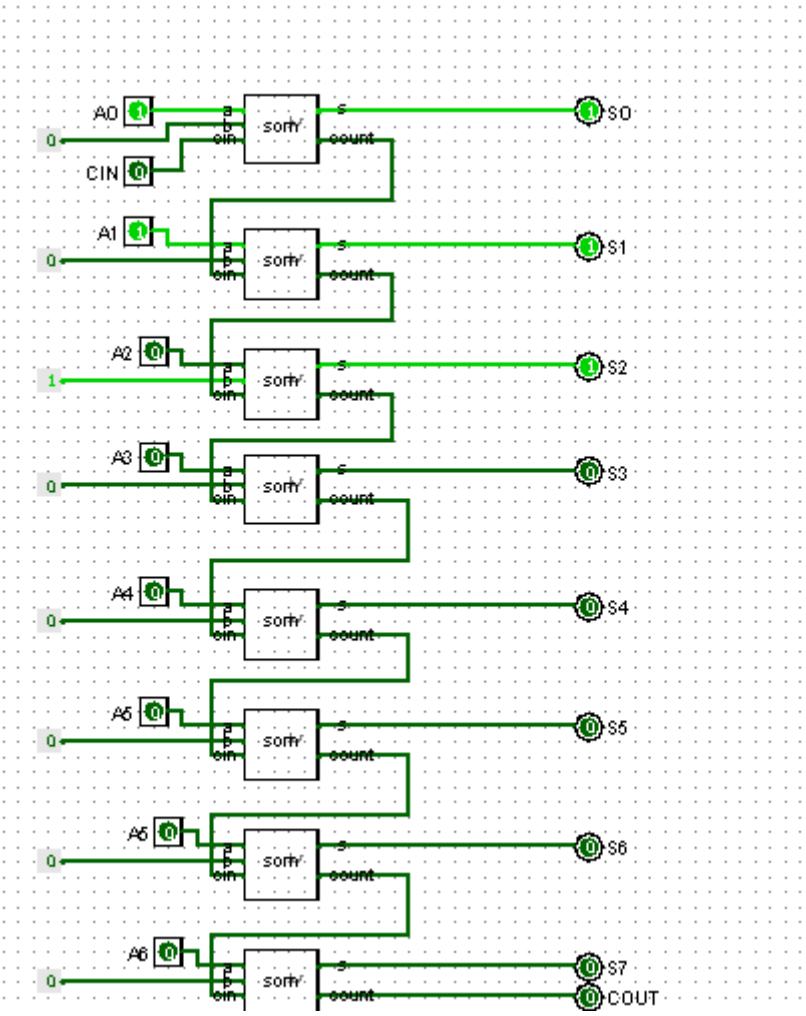
A	B	Carry-In	Soma	Carry-Out
0	0	0	0	0

0	1	0	1	0
1	0	0	1	0
1	1	0	0	1
0	0	1	1	0
0	1	1	1	1
1	0	1	0	1
1	1	1	1	1



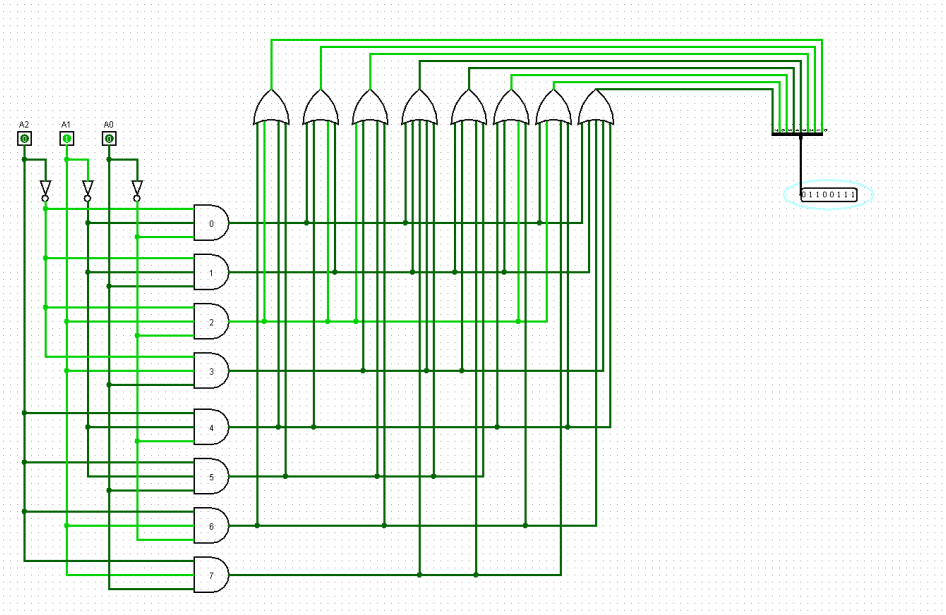
Este circuito funciona como uma calculadora com uma única missão: pegar qualquer número de 8 bits que a gente der e somar 4 a ele. Diferente de um somador comum que precisa de dois números para somar, este já vem com o número 4 "embutido" em sua estrutura. Ele tem uma entrada de 8 bits para recebermos um número (vamos chamar de A) e uma saída de 8 bits que mostra o resultado de A + 4. Para fazer a mágica acontecer, ele é construído usando oito pequenos somadores de 1 bit, colocados em fila, onde o "vai-um" de um alimenta o próximo.

A parte mais importante é que a segunda entrada (B) não é uma entrada de verdade, ela é fixa. Como o número 4 em binário é 00000100, o circuito é montado da seguinte forma:



2.5 Memória ROM de 8 bits

Uma memória ROM é um tipo de memória não volátil (não pode ser alterada por um usuário comum) usada principalmente em BIOS de computadores, cartuchos de videogames e em dispositivos móveis.
Ela é usada para armazenar dados e instruções essenciais e permanentes, que são necessários para inicialização do hardware, iniciar a sequência de boot e carregar o sistema operacional de um dispositivo.



O circuito de memória proposto tem organização de 8 palavras de 8 bits (8x8), utilizando portas lógicas OR, AND, inversores, pinos de entrada e barramento de 8 bits na saída.

Funcionamento

A memória proposta pode ser decomposta em:

- Decodificador de Endereços: responsável por selecionar qual palavra de memória será lida;
- Matriz de Memória: onde os dados são efetivamente armazenado através de conexões físicas;
- Lógica de Saída: responsável por agrupar os bits da palavra selecionada e apresentá-los;

1. **Entradas de Endereçamento**
   As linhas de endereço são representadas pelas entradas A2, A1 e A0. Com 3 bits de endereço, é possível acessar 2^3 = 8 localizações de memória distintas, variando do endereço 0 (000) ao endereço 7 (111).

2. **Decodificador de Endereços**
   Composto por:
   3 portas NOT, uma linha para cada endereço gerando sinais inversos complementares;
   8 portas AND, cada uma conectada a uma combinação únicas das linhas de endereço;

3. **Matriz de Memória e Lógica de Saída**

Composto por 8 portas OR. A saída de cada porta OR corresponde a um bit da palavra de dados de 8 bits na saída final (D7 a D0).

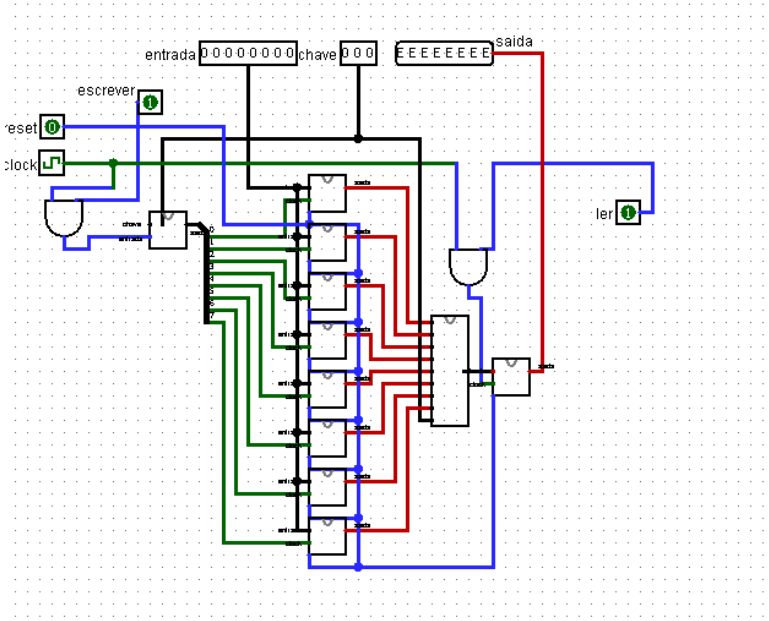
Exemplo de Funcionamento
O contexto para exemplificação será a BIOS de inicialização de um computador.
Considerando LOAD_A #val como uma instrução que carrega um valor de 4 bits no registrador A.

1. Ao ligar o computador, o endereço da CPU sempre inicia no 000. O processador coloca esse endereço nas suas linhas de endereço.
2. A ROM recebe 000 das entradas e a porta correspondente (AND 0) é ativada. As conexões dessa porta AND ativam as saídas correspondentes, colocando o valor 11001010 no barramento de dados.
3. O processador lê o valor e decodifica, interpretando a instrução LOAD_A #val representada pelo número binário trazido da ROM, posteriormente executando a instrução. Agora, o registrador A contém o valor 8.
4. O processador avança o ponteiro de instrução para o próximo endereço: 001.
5. Esses passos seguem da mesma forma: a CPU continua pedindo à ROM o conteúdo dos endereços, enquanto a ROM entrega as palavras de 8 bits, até que a CPU encontre uma instrução que pare ou pule para outro programa.

2.6 Memória RAM de 8 bits

Este circuito implementa uma Memória RAM (Random Access Memory) funcional utilizando Flip-Flops D, demultiplexador e multiplexador para leitura e escrita.
Possui interface de 8 bits para dados e sinais de clock, ler, escrever e chave de endereçamento.

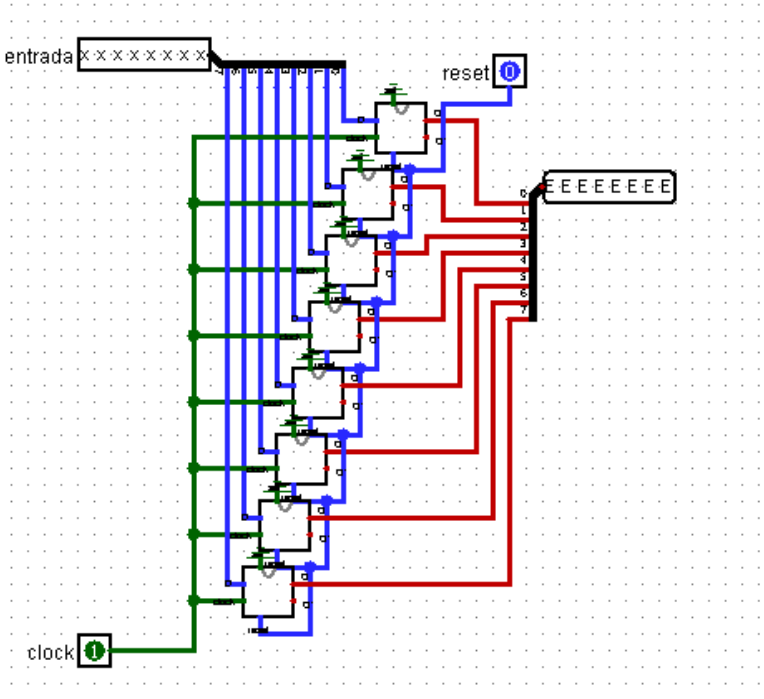
Memória RAM



Componentes Principais

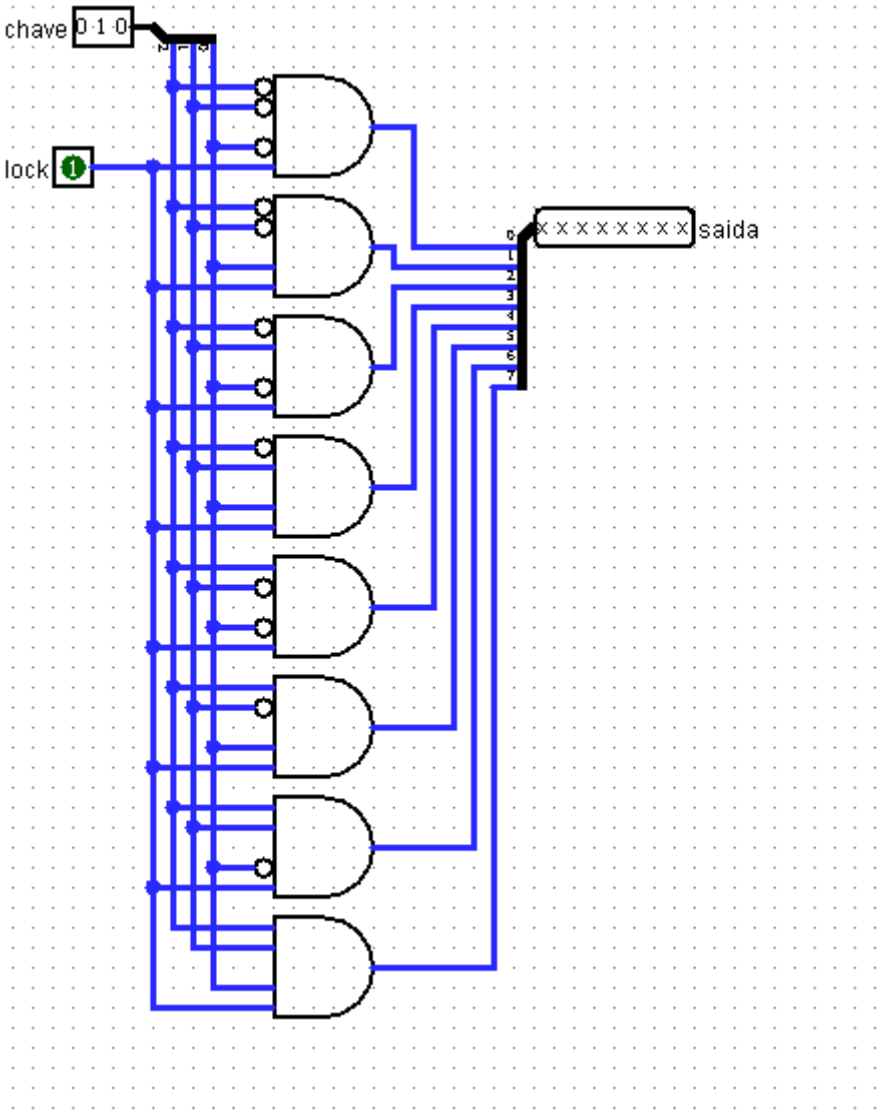
Registrador (Célula de Memória)

- As células são feitas com Flip-Flops D, que armazenam cada bit.
- Cada célula contém 8 FF-D em paralelo (um registro de 8 bits).



Demultiplexador 1×8 (1 bit)

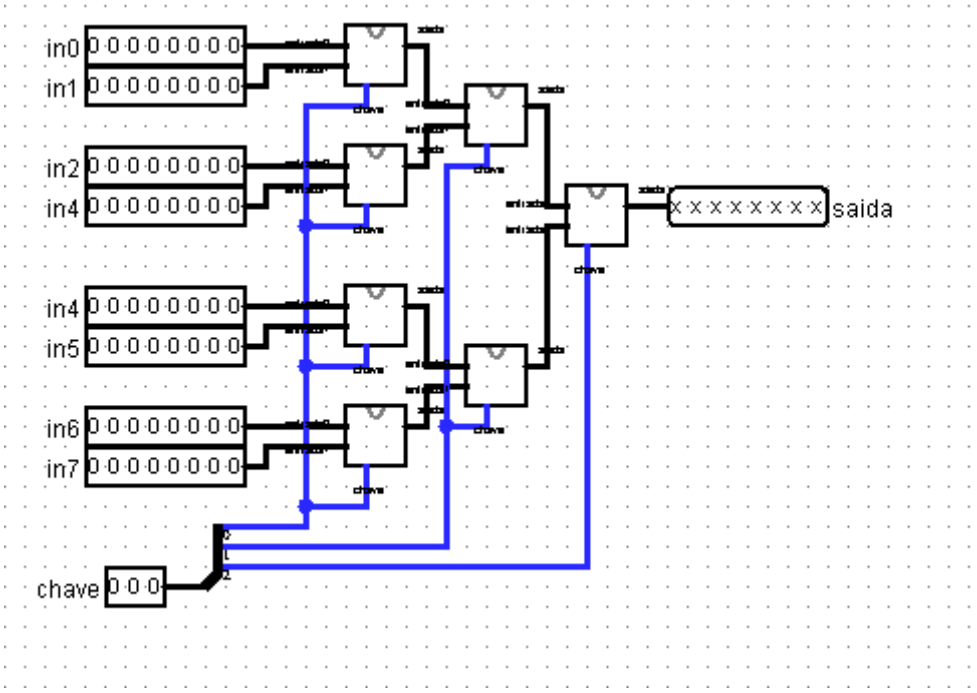
- Direciona o sinal de escrita para uma das 8 células.
- A célula ativa é escolhida pela chave de endereçamento.
- Permite escrever na célula selecionada apenas se o sinal escrever estiver ativo.
- Obs.: neste documento, a saída aparece como 8 bits para simplificar.
  No circuito principal, a saída do demultiplexador é dividida via Distribuidor (nativo do Logisim).



Multiplexador 8×1 (8 bits)

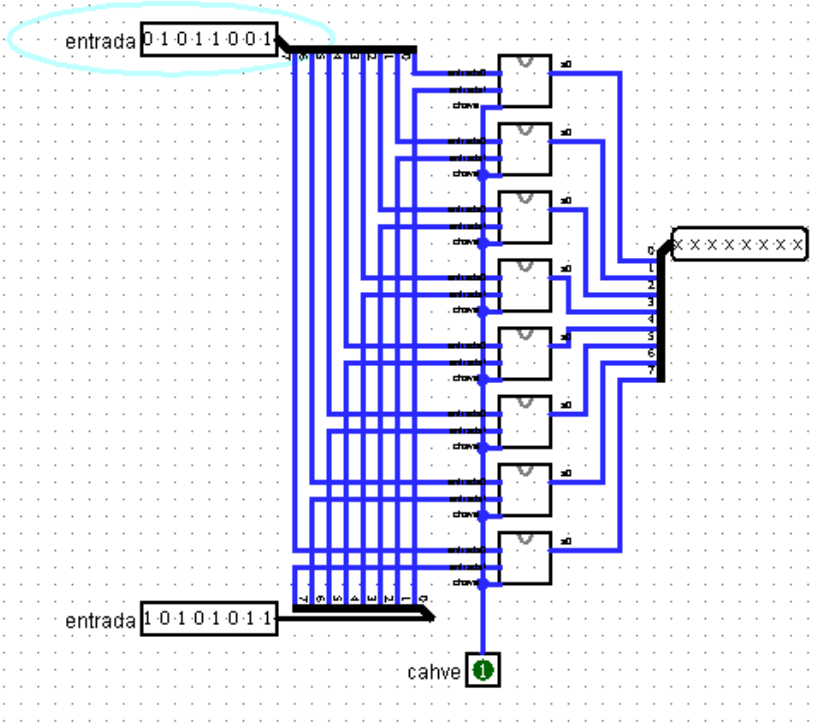
- Seleciona uma das 8 células para leitura.
- A célula lida é definida pela chave de endereçamento.

- Só transfere dados à saída se o sinal ler estiver ativado.



Modo de construção:

1. Fazer um Multiplexador 2×1 de 1 bit.
2. A partir dele, construir um Multiplexador 2×1 de 8 bits (8 cópias em paralelo).
3. Em seguida, montar uma árvore “mata-mata” com muxes 2×1 (8 bits) controlados por cada bit da chave → Multiplexador 8×1 (8 bits).



Registrador (Cache)

- Um registrador adicional armazena temporariamente os dados lidos.
- Atualiza apenas quando o sinal ler estiver ativado.

Sinais de Controle

- Clock: sincroniza todas as operações.
- Reset: zera os valores armazenados nas células.
- Escrever: ativa a escrita na célula selecionada.
- Ler: ativa a leitura da célula selecionada.

Funcionamento Geral

Escrita de Dados

1. Entrada de dados: aplicar os 8 bits na entrada.
2. Sinal de controle: escrever = 1 para permitir gravação.
3. Endereçamento: a chave seleciona qual célula receberá os dados.
4. Demux: direciona o sinal de escrita apenas à célula endereçada.
5. Armazenamento: na próxima borda de clock, a célula salva o byte de entrada.

Leitura de Dados

1. Endereçamento: a chave escolhe qual célula será lida.
2. Sinal de controle: ler = 1 para habilitar a leitura.
3. Multiplexador: seleciona os 8 bits da célula endereçada.
4. Transferência ao registrador: os dados vão para o cache (registrador).
5. Saída: o valor do registrador é enviado à saída de 8 bits.

Detalhes do Circuito

Interface de Entrada

- Dados (8 bits) a serem escritos.
- Endereço (3 bits) da chave de endereçamento.

Interface de Controle

- Clock: sincroniza o circuito.
- Escrever: habilita a escrita na célula endereçada.
- Ler: habilita a leitura da célula endereçada.
- Reset: zera os registros de memória.

Interface de Saída

- Saída (8 bits): dados lidos da memória (via registrador).

Resumo do Fluxo de Operações

Escrita:
`Dados de entrada → Demux (endereço) → Célula selecionada → (borda do clock) → Armazenado`

Leitura:
`Célula selecionada → Mux 8×1 → Registrador (cache) → Saída`

Vantagens do Circuito

- Armazenamento dinâmico: leitura e escrita em tempo real.
- Endereçamento simples: seleção por chave de 3 bits.
- Controle rigoroso: sinais separados de ler e escrever evitam conflitos.

Aplicações

- Memória volátil: para sistemas com leituras/escritas frequentes.
- Sistemas embarcados: pequenas RAMs em dispositivos de baixo custo.
- Simulação didática: entender a dinâmica de leitura/escrita em memórias digitais.

Considerações Finais

Este circuito demonstra a implementação de uma RAM simples com FF-D, multiplexadores e demultiplexadores, oferecendo um modelo prático para compreender operações fundamentais de leitura e escrita em sistemas digitais.

2.7 Banco de Registradores

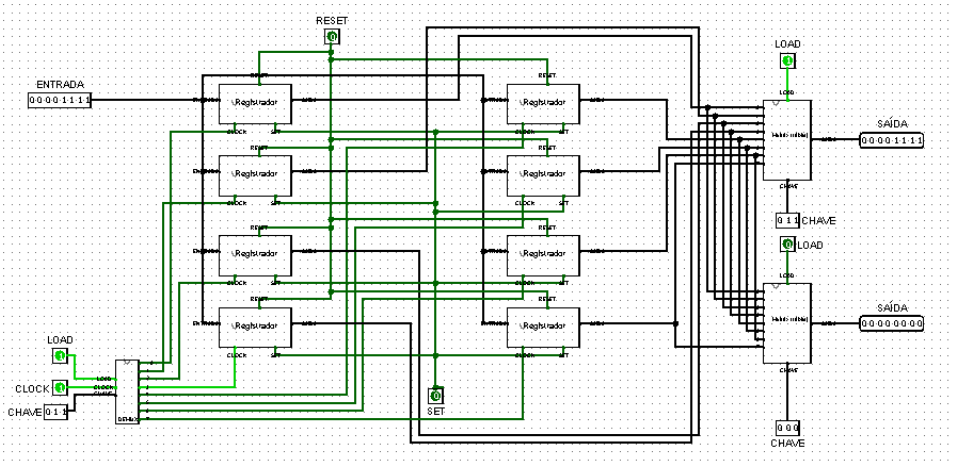
O banco de registradores é como a "mesa de trabalho" ou um pequeno "gaveteiro" super rápido que fica dentro do processador. Ele não serve para guardar arquivos grandes como a memória RAM, mas sim para segurar os poucos números que o processador precisa usar *agora*, no exato momento de um cálculo. Nosso projeto tem 8 "gavetas" (os registradores), e cada uma consegue guardar um número de 8 bits. A grande vantagem é que o processador pode escrever em uma gaveta e, ao mesmo tempo, ler o conteúdo de outras duas, tornando tudo muito rápido e eficiente.

Para que essa mágica aconteça, o circuito tem duas operações principais: escrever um dado e ler um dado. Para **escrever**, precisamos de três coisas: o número que queremos guardar (que vai na **ENTRADA**), o endereço da gaveta onde queremos guardá-lo (a **CHAVE SELETORA)**, e um sinal de comando (**LOAD**) para autorizar a gravação. Por dentro, um circuito chamado demultiplexador funciona como um "gerente", garantindo que apenas a gaveta certa seja aberta para receber o novo número. Para **ler**, é ainda mais fácil. O nosso circuito tem duas saídas independentes, cada uma com sua própria chave seletora. Basta escolher o endereço da gaveta na chave e, instantaneamente, o número que está lá dentro aparece na saída, pronto para ser usado.

Para exemplificar, vamos imaginar um cenário. Suponha que na gaveta de endereço 2 já está guardado o número 10. Agora, queremos guardar o número 50 na gaveta de endereço 5 e, logo em seguida, ler os dois números para fazer uma soma.

1. **Escrevendo o número 50 na gaveta 5:**
    - Primeiro, colocamos o número 50 (em binário,00110010) na **ENTRADA** de dados.
    - Depois, ajustamos a **CHAVE SELETORA** para o endereço 5 (em binário, 101).
    - Por fim, ativamos o sinal **LOAD** e damos um pulso de **CLOCK**. Pronto! A gaveta" 5 agora contém o número 50.
2. **Lendo os números 10 (gaveta 2) e 50 (gaveta 5) ao mesmo tempo:**
    - Ajustamos a **PRIMEIRA CHAVE DE LEITURA** para o endereço (010). Como num passe de mágica, o número 10 aparece na primeira **SAÍDA**.
    - Ao mesmo tempo, ajustamos a **SEGUNDA CHAVE DE LEITURA** para o endereço 5 (101). O número 50 que acabamos de guardar aparece na segunda **SAÍDA**.

Com isso, o processador agora tem os números 10 e 50 disponíveis ao mesmo tempo, prontos para serem enviados para a ULA (a calculadora do sistema) para realizar a soma. Isso mostra como o banco de registradores é essencial para organizar e agilizar as operações dentro de uma CPU.



2.8 Somador de 8 bits

O somador é um dos circuitos mais fundamentais para realizar operações aritméticas em computadores. São construídos a partir de circuitos lógicos básicos e são escalonados para manipular números binários de tamanhos variados. Ele funciona somando bits individuais e apresentando uma saída com o resultado dessa soma. Algumas somas precisam de mais bits para soma, então uma terceira entrada é adicionada, conhecida como carry in ou “vai-um”. O circuito proposto foi projetado a partir da combinação de somadores de 1 bit.

2.8.1. Somador de 1 bit

O somador de 1 bit soma um número de 1 bit com outro, também de 1 bit. Considerando a maneira como a soma manual é realizada, a0 seria o primeiro elemento, enquanto a1, o segundo elemento. S indica o resultado da soma. Como exemplo:

0 + 0 = 0

0 + 1 = 1

1 + 1 = 0 e sobe 1

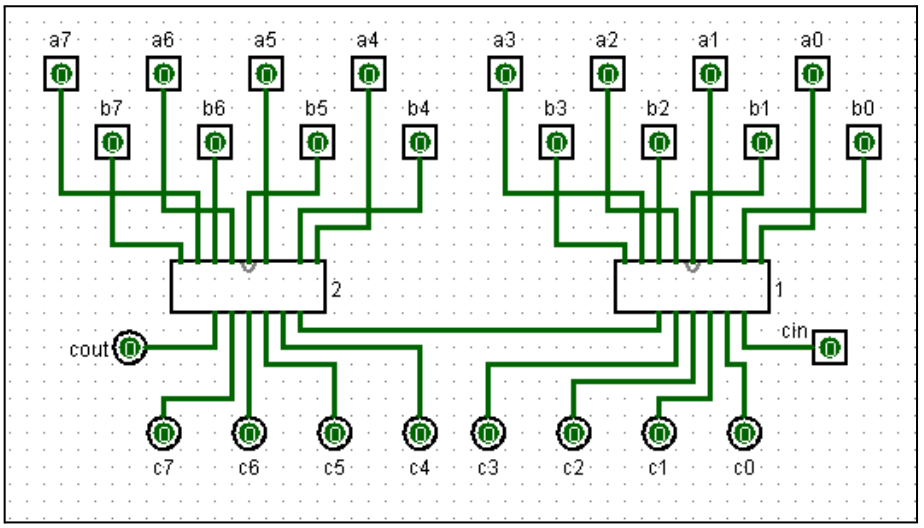
Podemos observar que a soma de 1 + 1 resulta em 10 (em decimal 2). O bit “0” é setado na saída, enquanto o bit “1” é levado para a casa esquerda (conhecido como “vai-um / carry in”), para realizar uma nova soma com os elementos da próxima etapa, se houver. Se logo no início da operação, o *carry in* for setado (ou se houver um bit de outra operação), o resultado será 1

na saída e 1 no carry out, ou seja 11 (3 em decimal), onde o bit “1” vira o carry-out, e será transportado para a próxima etapa da soma.

2.8.2. Somador de 4 bits

O somador de 4 bits consiste na composição de 4 circuitos de somadores de 1 bit, onde serão somados um número de 4 bits com outro de 4 bits. Nesse contexto, o carry-out de um circuito, se torna o carry-in do outro, seguindo a mesma lógica da operação realizada manualmente com número decimais, onde é somado aos elementos da coluna à esquerda, e “desce” para compor o resultado. O último carry-out do 4º circuito, indica se a soma excede a capacidade de representação, sinalizando um overflow.

2.8.3. Somador de 8 bits



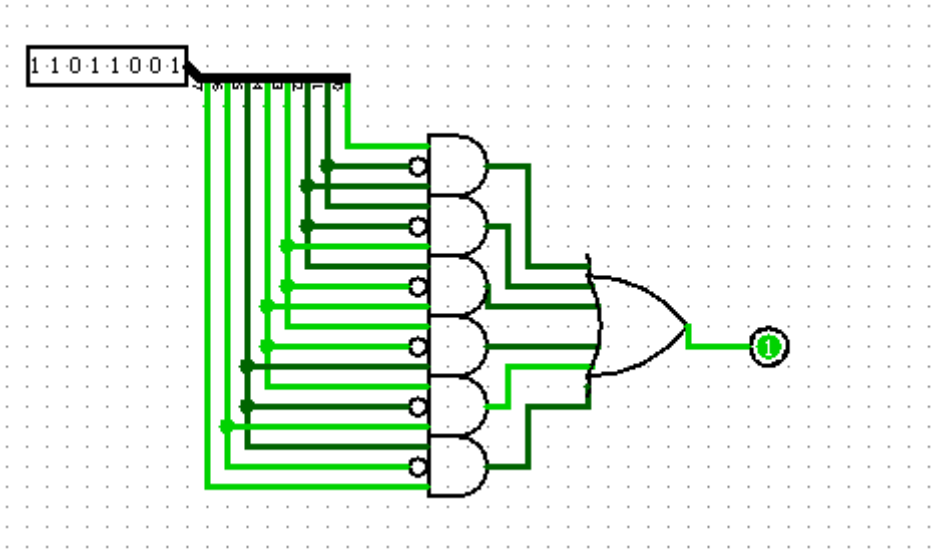
Seguindo a mesma lógica de módulo, o somador de 8 bits consiste na composição de 2 circuitos de 4 bits, somando dois números, ambos de 8 bits. De forma similar ao somador de 4 bits, o carry out do primeiro somador, se torna o carry in do segundo somador. O carry out final do segundo somador sinaliza um overflow da soma de 8 bits.

Como exemplo desse circuito, será realizado o teste de soma com os números 172 (10101100) + 40 (00101000), com resultado esperado de 212 (11010100). Os 4 bits mais à direita (LSB) do primeiro somador (bits 0-3), serão somados, e nesse gerarão um carry out, pois 1 + 1 = 10, que será encaminhado para o segundo somador (bits 4-7). O bit “0” permanecerá na

saída, e o bit “1” que veio da operação anterior como carry out, se torna o carry in da próxima etapa da soma. Esse processo é repetido durante o circuito sempre que necessário, até que encontre o último carry out, completando os 8 bits. O carry out final do segundo somar resultando em 0 indica que não houve overflow.

2.9 Detector de Sequência 101

Este circuito detecta o padrão binário 101 em um fluxo de bits de entrada.
Utiliza portas lógicas básicas para verificar e sinalizar quando essa sequência ocorre.



Descrição Geral

Entradas

- Fluxo de bits: um conjunto de bits binários é aplicado na entrada.
- Cada bit é analisado individualmente para verificar a presença da sequência 101.

Saída

- Um único bit que fica ativo (1) quando a sequência 101 é detectada.
- Permanece inativo (0) nos demais casos.

Componentes Utilizados

Portas Lógicas

- AND: verifica quando dois ou mais sinais satisfazem a condição requerida.
- NOT: inverte o valor de um bit, permitindo validar o 0 da sequência.
- OR: combina resultados parciais de portas AND.

Conexões

- Os bits de entrada são ligados a combinações de portas lógicas para validar cada posição da sequência (1, 0, 1).

Funcionamento do Circuito

Sequência Detectada

Para reconhecer 101, as condições são:

1. O primeiro bit deve ser 1.
2. O segundo bit deve ser 0.
3. O terceiro bit deve ser 1.

Processo de Detecção

- Bit 1: passa diretamente por uma porta AND para validar o valor 1.
- Bit 2: é invertido por uma porta NOT para validar que seja 0.
- Bit 3: passa diretamente por outra entrada de AND para validar o valor 1.

Combinação final:
As portas AND combinam os três bits consecutivos (1, 0, 1).
Se as condições forem atendidas, a saída da AND resulta em 1, que é enviada para a saída principal.

Casos de Teste

Entrada	Saída
`000`	0
`101`	1
`110`	0

01010101	1

Aplicações do Detector de Sequência

- Comunicação Digital: identificação de padrões específicos em fluxos de dados.
- Controladores de Estado Finito: monitoramento de eventos em sistemas digitais.
- Filtros de Sinal: extração de informações relevantes de sequências binárias.

2.10 Unidade Lógica e Aritmética (ULA)

A Unidade Lógica e Aritmética (ULA) é um circuito digital combinacional que constitui o núcleo computacional de uma Unidade Central de Processamento (CPU). Sua função é executar operações aritméticas (soma, subtração) e operações lógicas bit a bit (AND, OR, XOR, etc.) sobre dois operandos de entrada. Este projeto implementa uma ULA de 8 bits, capaz de realizar um conjunto de 10 funções distintas, determinadas por um sinal de controle de 4 bits.

A arquitetura da ULA opera com base no princípio da computação paralela. Internamente, ela é composta por múltiplas unidades funcionais dedicadas: um somador de 8 bits, um subtrator, circuitos para cada operação lógica e unidades para operações de deslocamento (shift). Todas estas unidades recebem os mesmos operandos de entrada (A e B) e processam seus respectivos resultados simultaneamente. A seleção do resultado final é gerenciada por um multiplexador de 8 bits na saída, que é controlado por um sinal de seleção de 4 bits (opcode). Este código determina qual das saídas das unidades funcionais será efetivamente a rota para a saída principal da ULA.

Para exemplificar o funcionamento, será demonstrada a execução de duas operações sobre os mesmos operandos: uma soma e uma operação lógica AND.

Operandos de Entrada:

- **A:00001010** (10 em decimal)
- **B:00000101** (5 em decimal)

1. Execução da Soma (A + B):

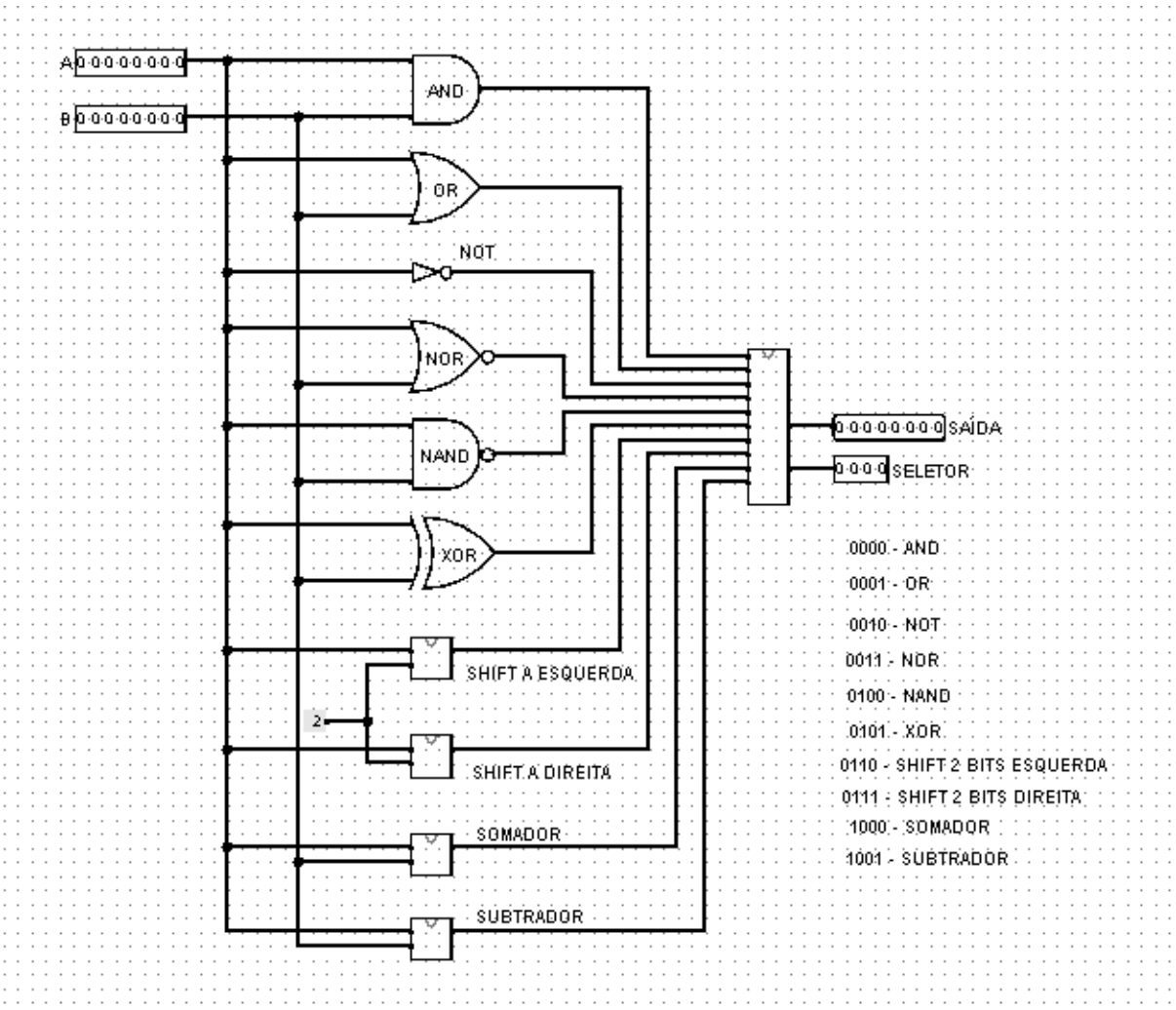
- Primeiro, o sinal de seleção (opcode) é configurado com o valor 1000, que corresponde à instrução de SOMA.

- Neste instante, todas as unidades internas já calculam seus resultados. O somador produziu o valor 00001111 (15 em decimal).
- O multiplexador de saída, ao receber o seletor 1000, seleciona o barramento de dados proveniente do somador.
- Consequentemente, o resultado na saída principal da ULA é 00001111.

2. Execução da Operação AND (A E B):

- Mantendo os mesmos operandos A e B, o sinal de seleção é alterado para 0000, o opcode da operação AND.
- O multiplexador de saída reage a esta mudança e agora seleciona o resultado da unidade lógica AND, que calculou o valor 0000000000.
- A saída da ULA é imediatamente atualizada para 0000000000.

Este método permite que a ULA execute um vasto repertório de operações de forma eficiente, mudando sua função dinamicamente apenas pela alteração do sinal de controle, o que é fundamental para a execução de instruções em um processador.

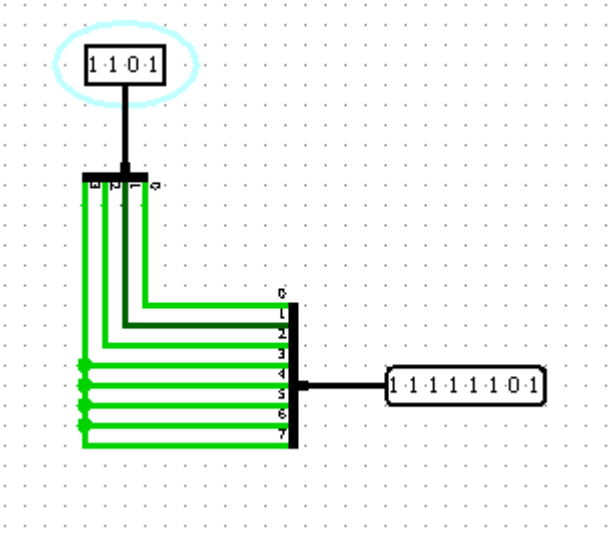


2.11 Extensor de Sinal de 4 Bits para 8 bits

Um extensor de sinal é um circuito criado para converter um número com sinal n bits para um número com sinal de n bits, sem que seu valor e sinal sejam alterados no processo.

No esquema proposto a seguir, o circuito converte um número de 4 bits para 8 bits, organizado da seguinte maneira: A entrada com um barramento de 4 bits, onde o fio 3 representa o bit mais significativo, e é o bit que determina o sinal do número. A saída com barramento de 8 bits, onde os fios 4, 5, 6 e 7, estão ligados ao bit mais significativo da entrada (3), para replicar o valor/sinal, e assim estender o número sem que haja perda de significado.

Por exemplo, o número -5 é representado usando o complemento de dois por 1101. Na conversão, os bits sem mantém, porém os 4 bits adicionais serão a repetição do bit mais esquerda (o MSB), que resultará em 11111011.



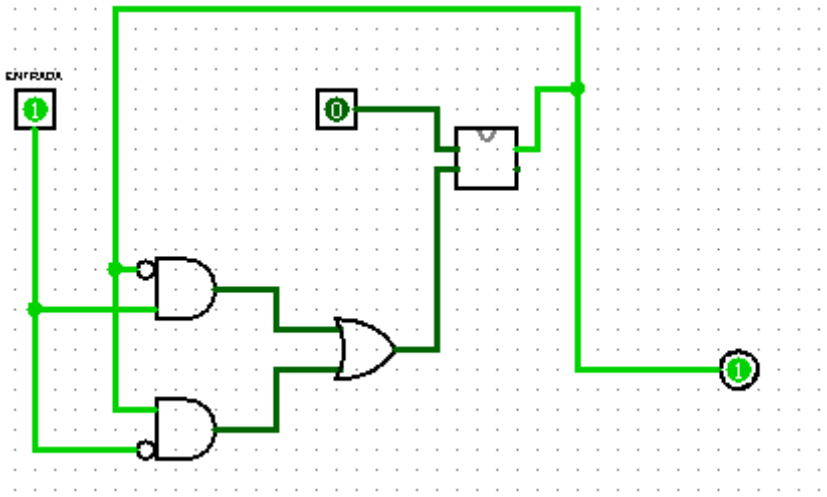
Número Decimal	Entrada	Saída
0	0000	0000 0000
1	0001	0000 0001
2	0010	0000 0010
3	0011	0000 0011
4	0100	0000 0100
5	0101	0000 0101
6	0110	0000 0110
7	0111	0000 0111
-8	1000	1111 1000
-7	1001	1111 1001
-6	1010	1111 1010
-5	1011	1111 1011
-4	1100	1111 1100

-3	1101	1111 1101
-2	1110	1111 1110
-1	1111	1111 1111

2.12 Máquina de Estado

O circuito é um modelo matemático usado para representar e controlar sistemas que podem estar em um número finito de estados distintos.
A transição entre estados ocorre com base em entradas e/ou eventos externos.
É amplamente utilizado em sistemas digitais, controle automático, programação e outros domínios.

Máquina de Estado



Componentes do Circuito

Entrada (ENTRADA)

- Representa o sinal externo que influencia o comportamento da máquina de estado.
- Pode assumir valores binários (0 ou 1) para condicionar a lógica do circuito.

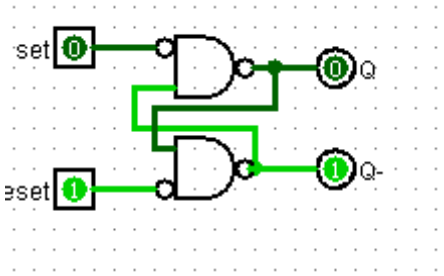
Clock (CLOCK)

- Sincroniza as transições de estado no Flip-Flop D.
- Mudanças de estado ocorrem apenas nas bordas do clock, garantindo operação síncrona.

Flip-Flop D

- Elemento de memória que armazena o estado atual da máquina.

- A entrada D recebe o próximo estado calculado pela lógica combinacional.
- A saída do FF fornece o estado atual para a lógica combinacional.



Lógica Combinacional

- Composta por portas AND, OR e NOT.
- Calcula o próximo estado a partir da entrada e do estado atual.
- Determina a saída com base no estado atual e/ou na entrada.

Saída (SAÍDA)

- Representa o resultado da máquina de estado, podendo ativar ou controlar outros sistemas.

Funcionamento do Circuito

Estado Atual

- O Flip-Flop D mantém o estado atual, atualizado a cada ciclo de clock.

Cálculo do Próximo Estado

- A lógica combinacional processa o estado atual e a entrada para determinar o próximo estado, que é aplicado à entrada D.

Saída do Circuito

- A saída é gerada pela lógica combinacional.
- Pode depender apenas do estado atual (FSM de Moore) ou do estado e da entrada (FSM de Mealy).

Aplicações Possíveis

- Controle de Semáforo: cada estado representa uma luz (vermelho, amarelo, verde), com transições condicionadas pela entrada.
- Contadores Sequenciais: contagem/seqüenciamento de estados de forma síncrona.

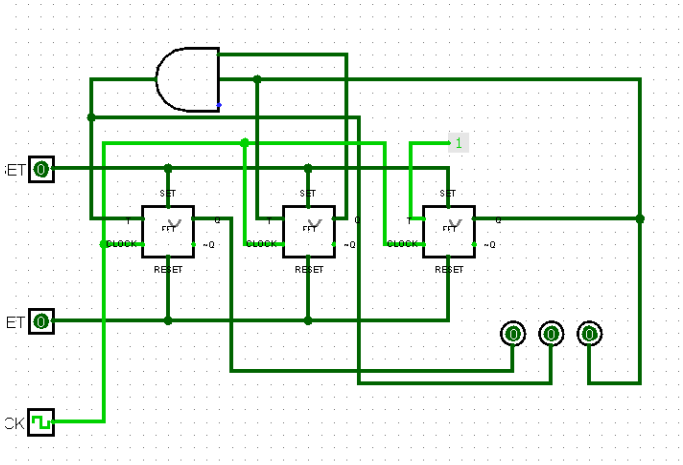
- Sistemas de Controle: implementação de lógicas de controle em máquinas industriais e automação.

2.13 Contador Síncrono

Um contador faz exatamente o que o nome diz: conta. Nosso contador usa Flip-Flops T para avançar a contagem em binário a cada pulso do **CLOCK**. O termo "síncrono" é importante: significa que todos os flip-flops recebem o sinal do **CLOCK** ao mesmo tempo e mudam de estado em perfeita sincronia.

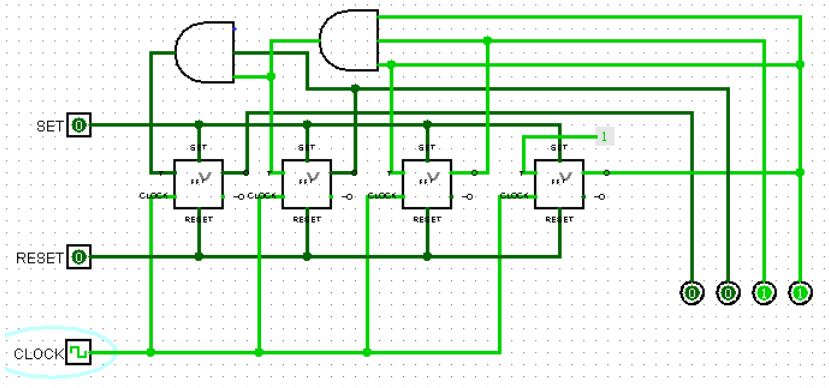
Teste Prático(Contador de 3 bits):

1. Inicialmente, a saída é 000.
2. Primeiro pulso do clock: a saída muda para 001.
3. Segundo pulso: a saída muda para 010.
4. E assim sucessivamente até chegar em 111, e retorna para 000.



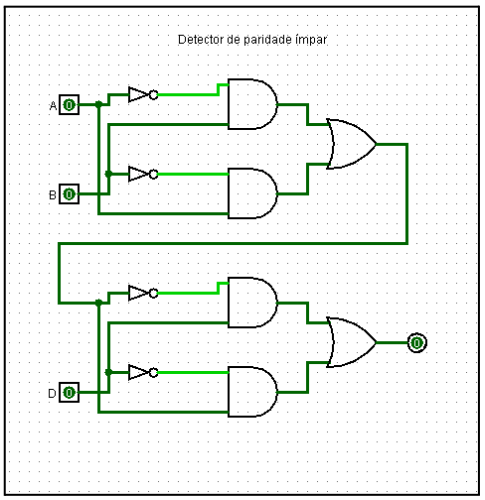
Teste Prático(Contador de 4 bits):

5. Inicialmente, a saída é 0000.
6. Primeiro pulso do clock: a saída muda para 0001.
7. Segundo pulso: a saída muda para 0010.
8. E assim sucessivamente até chegar em 1111, e retorna para 0000.

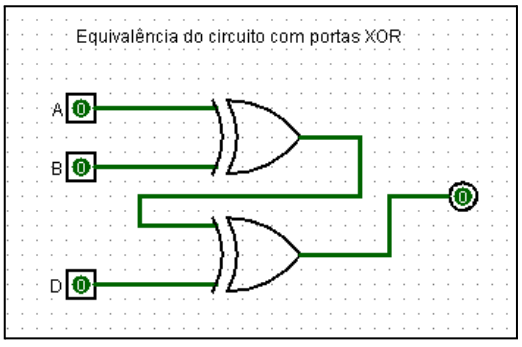


2.14 Detector de Paridade Ímpar

O detector de paridade ímpar é um circuito projetado para verificar se o número de bits com valor 1 em um conjunto de entrada é par ou ímpar. Por exemplo, o número 12 possui 2 números “1” na sua representação em binário (1100). A quantidade (2) é par, logo a saída será 0. O número 7 possui 3 números “1” em binário (0111), logo a quantidade de números “1” é ímpar, então a saída do circuito será 1.



O circuito está organizado com portas AND e OR, com 3 entradas A, B e D. Na parte superior, duas entradas A e B estão conectadas a portas AND, na qual o resultado é reunido por uma porta OR. O resultado desse primeiro bloco, se torna a entrada “A” da parte inferior, que se configura da mesma maneira e resulta em uma saída S final. Esse circuito nada mais é do que a representação em módulos da porta XOR, que apresenta o mesmo comportamento, de forma mais simplificada.



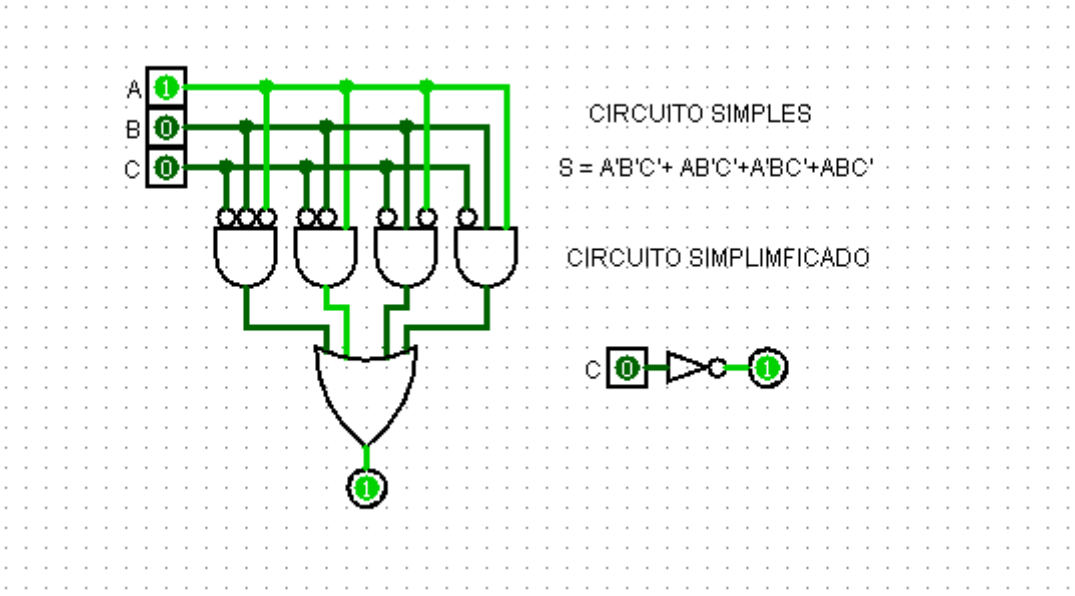
Número Decimal	Entrada ABCD	Nº de 1s	Saída
0	0000	0 (par)	0
1	0001	1 (ímpar)	1
2	0010	1 (ímpar)	1
3	0011	2 (par)	0
4	0100	1 (ímpar)	1

5	0101	2 (par)	0
6	0110	2 (par)	0
7	0111	3 (ímpar)	1
8	1000	1 (ímpar)	1
9	1001	2 (par)	0
10	1010	2 (par)	0
11	1011	3 (ímpar)	1
12	1100	2 (par)	0
13	1101	3 (ímpar)	1
14	1110	3 (ímpar)	1
15	1111	4 (par)	0

2.15 Circuitos com Mapa de Karnaugh

O objetivo deste documento é mostrar por que dois circuitos diferentes produzem a mesma saída, usando o Mapa de Karnaugh para simplificar a expressão booleana do circuito original e validá-la contra um circuito simplificado.

Circuito simplificado



Circuito Original (Não Simplificado)

Expressão booleana dada:

S = A'B'C + ABC' + A'BC' + ABC

Montagem do Mapa de Karnaugh (3 variáveis: A, B, C)

Para três variáveis existem 8 células (todas as combinações de A, B, C).
Preenchimento a partir dos mintermos:

- `A'B'C` → (A=0, B=0, C=1)
- `ABC'` → (A=1, B=1, C=0)
- `A'BC'` → (A=0, B=1, C=0)
- `ABC` → (A=1, B=1, C=1)

Mapa (AB em ordem Gray: 00, 01, 11, 10):

AB \ C	0	1
00	0	1
01	1	0
11	1	1
10	0	0

Simplificação pelo Karnaugh

Agrupamentos válidos

1. Par em C=0: células (01, 0) e (11, 0) → elimina A
   ⇒ B · C'

2. Par em AB=11: células (11, 0) e (11, 1) → elimina C
⇒ A · B

3. Célula isolada (00, 1): não tem adjacente 1 para agrupar
⇒ A' · B' · C

Expressão simplificada

Combinando os implicantes:

S = B·C' + A·B + A'·B'·C

Observação: a forma acima é equivalente à expressão original e resulta do agrupamento mais econômico possível dado o preenchimento do mapa.

Comparação dos Circuitos

- Circuito não simplificado
  Implementa a soma completa:
  `S = A'B'C + ABC' + A'BC' + ABC`
  (mais portas e interconexões).
- Circuito simplificado
  Implementa a forma reduzida:
  `S = B·C' + A·B + A'·B'·C`
  (menos portas, mesmo comportamento).

Saída

Ambas as realizações geram exatamente a mesma saída S para todas as combinações de A, B, C.

Conclusão

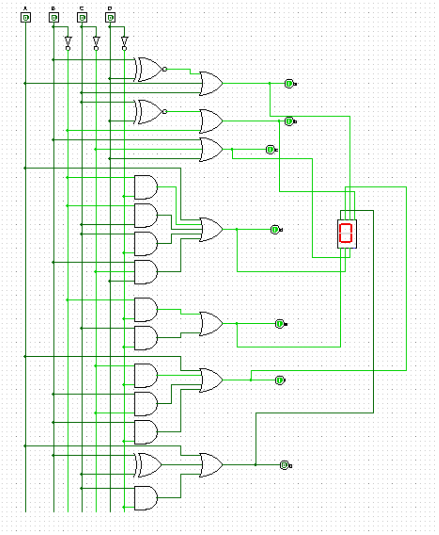
O uso do Mapa de Karnaugh reduz o número de componentes necessários, mantendo o comportamento funcional idêntico. Isso resulta em um circuito mais eficiente e mais fácil de implementar/manter.

2.16 Decodificador de 7 Segmentos

Este circuito é um "tradutor". Computadores entendem números em binário (como **0101** para o número **5**), mas nós lemos em displays com segmentos iluminados. O trabalho do decodificador é receber o número em binário e acender os segmentos corretos para formar o dígito que conseguimos ler.

Teste Prático:

1. Enviamos o código binário **0111** (que representa o número **3**) para as entradas do decodificador.
2. O circuito ativa as saídas correspondentes aos segmentos **a,b,c,d e g.**
3. No display, esses segmentos se acendem, formando perfeitamente o número **3**.



Dígito	D	C	B	A	a	b	c	d	e	f	g
0	0	0	0	0	1	1	1	1	1	1	0
1	0	0	0	1	0	1	1	0	0	0	0
2	0	0	1	0	1	1	0	1	1	0	1
3	0	0	1	1	1	1	1	1	0	0	1
4	0	1	0	0	0	1	1	0	0	1	1
5	0	1	0	1	1	0	1	1	0	1	1
6	0	1	1	0	1	0	1	1	1	1	1
7	0	1	1	1	1	1	1	0	0	0	0
8	1	0	0	0	1	1	1	1	1	1	1
9	1	0	0	1	1	1	1	1	0	1	1

2.17 Detector de Número Primo

O circuito lógico a seguir foi construído para ser capaz de identificar se um número binário de 4 bits é um número primo ou não. O circuito possui 4 entradas (A, B, C, D) correspondente aos 4 bits, uma saída S, e foi construído com a combinação

das portas lógicas NOT, AND e OR. A entrada A é o bit mais significativo (MSB), enquanto a entrada D é o bit de menor significância (LSB). A saída somente terá valor lógico alto (será 1), se o número de entrada for primo, e terá valor lógico baixo (será 0) caso contrário.

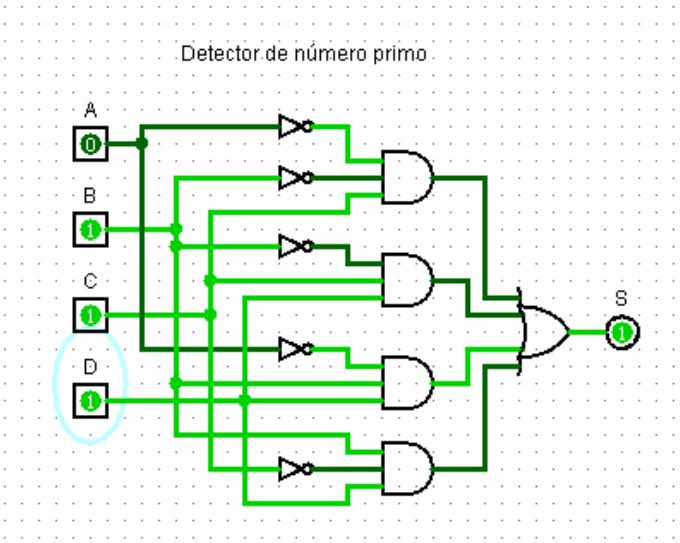
O circuito, por ser de 4 bits, identifica os números primos num intervalo de 0 (0000) à 15 (1111), sendo eles 2, 5, 7, 11 e 13. Cada porta AND é projetada para identificar um ou mais desses números, em quando a porta OR é usada para reunir os resultados.

1 AND = 2 (0010) e 3 (0011)

2 AND = 3 (0011) e 11 (1011)

3 AND = 5 (0101) e 7 (0111)

4 AND = 5 (0101) e 13 (1101)



ABCD (Número Binário)	Número Decimal	Saída
0000	0	0
0001	1	0
0010	2	1
0011	3	1
0100	4	0
0101	5	0
0110	6	0
0111	7	1
1000	8	0
1001	9	0
1010	10	0
1011	11	1
1100	12	0
1101	13	1
1110	14	0
1111	15	0

3 CONCLUSÃO

A realização deste projeto, abrangendo desde portas lógicas fundamentais até componentes complexos como o Banco de Registradores, Unidade Lógica e Aritmética e Memórias, foi uma etapa crucial para solidificar nosso conhecimento prático na disciplina. Utilizando a ferramenta de simulação Logisim, foi possível não apenas projetar e construir, mas também visualizar e depurar o comportamento de cada circuito, transformando conceitos teóricos em sistemas funcionais. Através deste processo, aprendemos a importância da modularidade na engenharia de hardware e como a integração entre circuitos combinacionais e sequenciais dá origem às estruturas complexas que formam a base de qualquer processador.

4 REFERÊNCIAS

BR-ELETRONICA. **Flip-Flop D - Circuitos Digitais**. [S. l.]: BR-Eletronica, 2016. 1 vídeo (5 min 49 s). Disponível em: https://youtu.be/2_UE7Pl-1yE. Acesso em: 15 out. 2025.

BR-ELETRONICA. **Flip-Flop JK - Circuitos Digitais**. [S. l.]: BR-Eletronica, 2016. 1 vídeo (13 min 05 s). Disponível em: https://youtu.be/6fEfOZJTQcQ. Acesso em: 15 out. 2025.

LUIS MEIRA. **Circuito Somador 1 4 e 8 bits.** Disponível em: <https://www.youtube.com/watch?v=eiFXNVBd8LQ>. Acesso em: 15 out. 2025.

DERIG ALMEIDA VIDAL. **Introdução sobre memórias ROM.** Disponível em: <https://www.youtube.com/watch?v=CXv1uArbSV8>. Acesso em: 15 out. 2025.

PATTERSON, David A.; HENNESSY, John L. **Organização e Projeto de Computadores**: A Interface Hardware/Software. 5. ed. Rio de Janeiro: LTC, 2017.

STALLINGS, William. **Arquitetura e Organização de Computadores**. 10. ed. São Paulo: Pearson Education do Brasil, 2017.