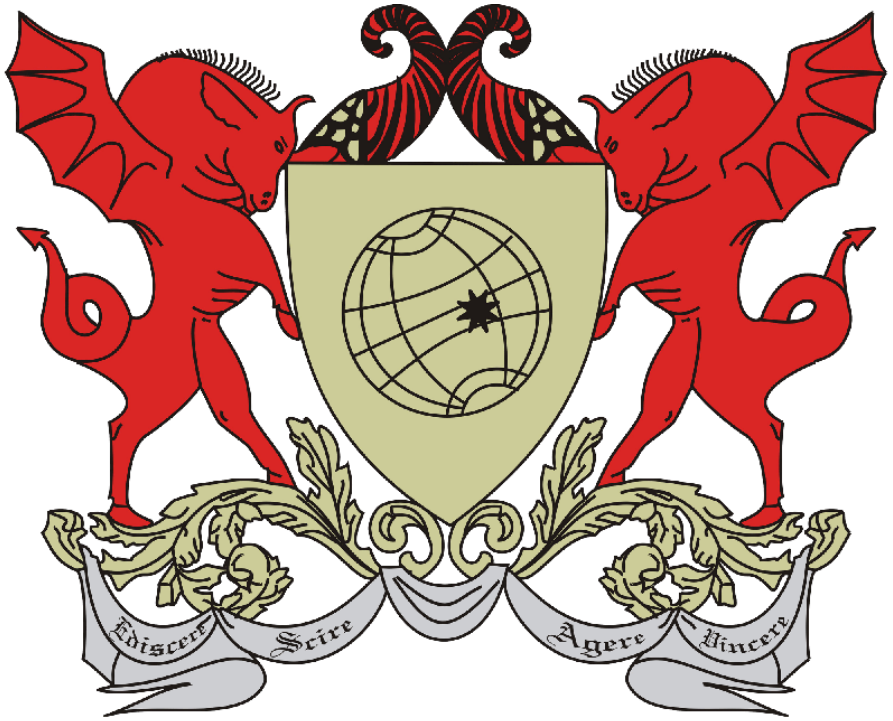


CCF 251 – Introdução aos Sistemas Lógicos

Aula 01 – Introdução e Sistemas de Numeração

Prof. José Augusto Nacif – jnacif@ufv.br

Por que estudar projeto lógico?

- Razões óbvias
  - Esta disciplina é obrigatória no curso de bacharelado em Ciência da Computação
  - É a base de qualquer dispositivo de computação moderno
    - Pequenos blocos lógicos são usados para construir sistemas complexos
    - Fornece um modelo de como um computador funciona
- Razões mais importantes
  - O paralelismo inerente do hardware é normalmente o primeiro contato com computação paralela
  - Apresenta um contraponto em relação a projeto de software sendo útil para entender computação em geral

O que será abordado na disciplina?

- A linguagem de projeto lógico
  - Álgebra booleana
  - Minimização lógica
  - Estados
  - Temporização
  - Ferramentas CAD
- O conceito de estado em sistemas digitais
  - Análogo a variáveis sistemas de software

O que será abordado na disciplina?

- Como especificar/simular/compilar projetos
  - Linguagens de descrição de hardware
  - Ferramentas para simular os projetos
  - Compiladores lógicos para sintetizar os blocos de hardware
  - Mapeamento em hardware programável
- Contraste com projeto de software
  - Implementações paralela e serial
  - Especifica o algoritmo assim como os recursos de armazenamento e computação necessários

Aplicações de Projeto Lógico

- Projeto de computadores
  - CPUs, barramentos, periféricos
- Redes e comunicações
  - Telefones, modems, roteadores
- Equipamentos embarcados
  - Em carros, brinquedos, utilitários domésticos, sistemas de entreterimento
- Equipamentos científicos
  - Teste, sensores, relatórios
- A computação é muito mais abrangente do que apenas computadores pessoais

Histórico

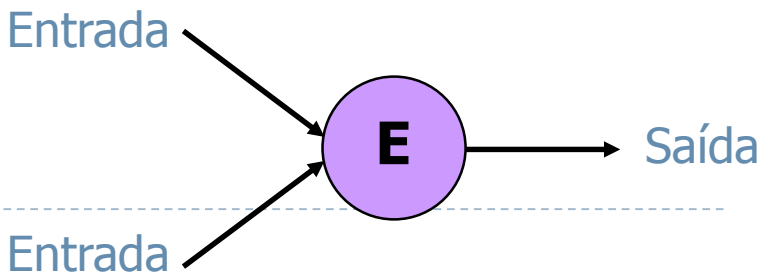
- 1850: George Boole inventa a álgebra Booleana
  - Mapeia proposições lógicas em símbolos
  - Permite a manipulação de declarações lógicas utilizando matemática
- 1938: Claude Shannon relaciona álgebra booleana com chaves
  - Dissertação de mestrado
- 1945: John von Neumann desenvolve o primeiro computador com programa armazenado
  - Elementos de chaveamento = válvulas
- 1946: ENIAC . . . Primeiro computador completamente eletrônico
  - 18.000 válvulas
  - Algumas centenas de multiplicações por minuto
- 1947: Shockley, Brittain e Bardeen inventam transistor
  - Substitui as válvulas
  - Permite a integração de múltiplos transistores em um único encapsulamento
  - Início da eletrônica moderna

O que é projeto lógico?

- O que é projeto?
  - Dada a especificação de um problema, identificar uma solução utilizando componentes disponíveis, atendendo a restrições de tamanho, custo, potência, beleza, etc…
- O que é projeto lógico?
  - Determinar o conjunto de componentes lógicos para realizar controle e/ou manipulação de dados e/ou função de comunicação e suas respectivas interconexões
  - Quais componentes lógicos escolher? – Existem muitas tecnologias de implementação (e.g., dispositivos com função fixa, programáveis, transistores em um chip, etc
  - O projeto pode precisar ser otimizado e/ou transformado para atender restrições de projeto

O que é hardware digital?

- Conjunto de dispositivos que lê e/ou controla fios que transmitem um valor digital (i.e., podem ser interpretados como “0” ou “1”)
  - Lógica digital na qual a tensão 0.8V é “0” e > 2.0V é “1”
  - Par de fios de transmissão nos quais os valores “0” ou “1” são definidos pelo maior valor de tensão (diferencial)
  - Orientação da magnetização significa “0” ou “1”
- Dispositivos primitivos de hardware digital
  - Computação lógica
    - Se os dois fios de entrada são “1”, o fio da saída se torna “1” (E)
    - Pelo menos um dos dois fios de entrada é “1”, o fio de saída se torna “1” (OU)
    - Se o fio de entrada é “1” , o fio de saída se torna “0” (NÃO)
  - Dispositivos de memória (armazenamento)
    - Armazena um valor
    - Lê um valor previamente armazenado



Mais informações sobre projeto digital

- Tendências na indústria de projeto de hardware
  - Projetos cada vez maiores
  - Tempo para o produto entrar no mercado cada vez menor
  - Produtos cada vez mais baratos
- Escala
  - Automatização do projeto utilizando ferramentas CAD
  - Múltiplos níveis de representação do projeto
- Tempo
  - Ênfase em representações mais abstratas
  - Dispositivos programáveis ao invés de dispositivos com função fixa
  - Técnicas de síntese automática
  - Importância de metodologias de projeto
- Custo
  - Níveis cada vez maiores de integração
  - Utilização de simulação para depurar projetos
  - Simular e verificar antes de fabricar

CCF 251: Conceitos/Habilidades

- Entender os fundamentos de projeto lógico
- Entender as metodologias de projeto
- Métodos modernos de especificação
- Familiaridade com ferramentas CAD
- Projetar sistemas digitais em uma tecnologia de implementação
- Apresentação das diferenças e similaridades de projeto de software e hardware

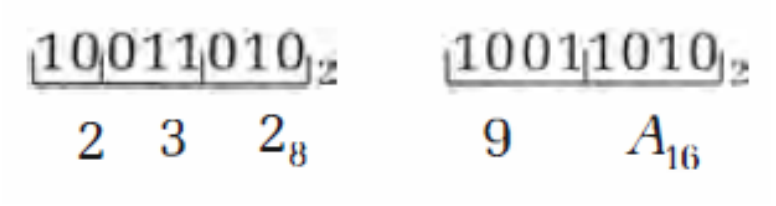
Nova habilidade: Desempenhar a tarefa de projeto lógico com a ajuda de ferramentas e mapear a descrição do problema em uma implementação com dispositivos de lógica Programável após validação via simulação entendendo as vantagens/desvantagens em comparação com implementações de software

Sistemas de Numeração

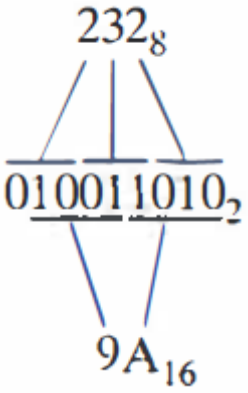
- Números decimais
  - 154_{10}
    - $1 \times 10^2 + 5 \times 10^1 + 4 \times 10^0$
- Números binários, octais e hexadecimais
  - 10011010_2
    - $1 \times 2^7 + 0 \times 2^6 + 0 \times 2^5 + 1 \times 2^4 + 1 \times 2^3 + 0 \times 2^2 + 1 \times 2^1 + 0 \times 2^0$
  - 232_8
    - $2 \times 8^2 + 3 \times 8^1 + 2 \times 8^0$
  - $9A_{16}$
    - $9 \times 16^1 + 10 \times 16^0$

Sistemas de Numeração

- Conversão entre sistemas binário, octal e hexadecimal
  - Conversão de binário para octal ou hexadecimal

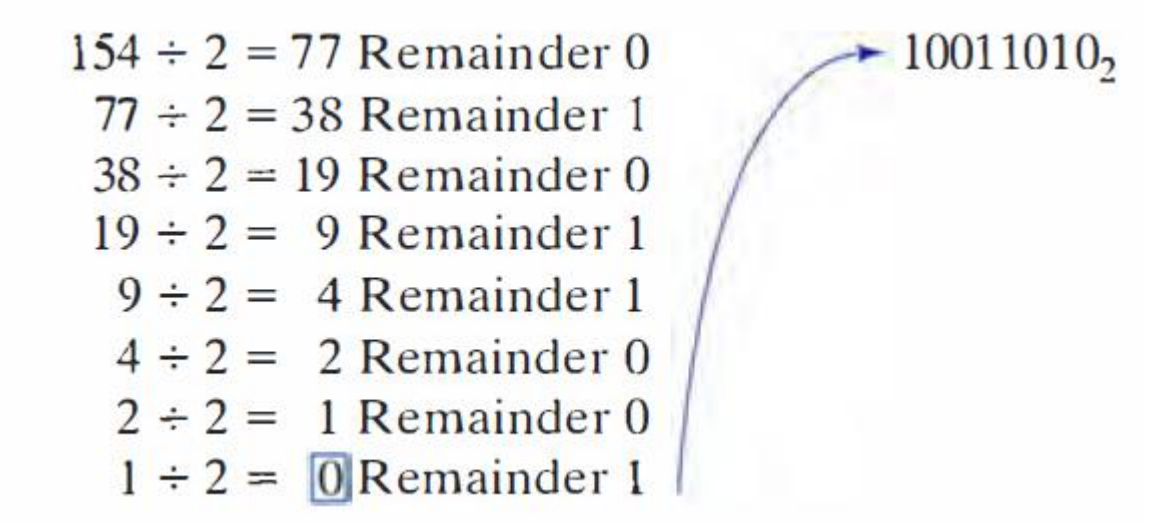


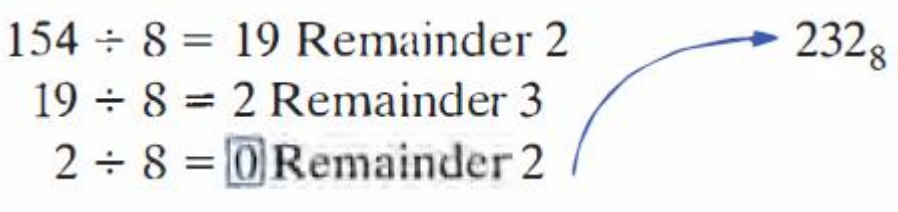
  - Conversão de octal para hexadecimal e vice-versa

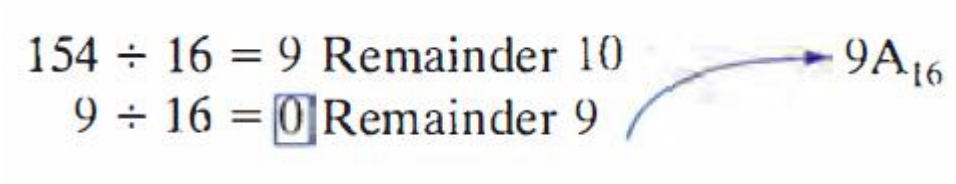


Sistemas de Numeração

- Conversão da base 10 para a base 2
  - Divisões sucessivas







Sistemas de Numeração

- Operações aritméticas binárias
  - Adição em notação posicional

$$\begin{array}{r} 95_{10} \\ +\ 16_{10} \\ \hline 111_{10} \end{array}$$

$$\begin{array}{r} 9 \times 10^1 + 5 \times 10^0 \\ +\ 1 \times 10^1 + 6 \times 10^0 \\ \hline 10 \times 10^1 + 11 \times 10^0 \\ \downarrow \\ 1 \times 10^2 + (0+1) \times 10^1 + 1 \times 10^0 \end{array}$$

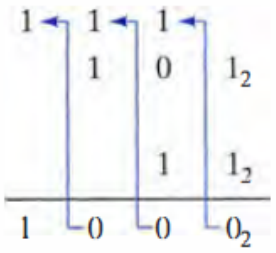
$$\begin{array}{r} 11 \leftarrow \text{Vai um} \\ 95_{10} \\ 16_{10} \\ \hline 111_{10} \end{array}$$

  - Adição na base 2

$$\begin{array}{l} 0 + 0 = \boxed{0} \\ 0 + 1 = \boxed{1} \\ 1 + 0 = \boxed{1} \\ 1 + 1 = \boxed{0} \end{array} \quad \text{Vai um}$$

Sistemas de Numeração

- Exemplos de adição binária
  - $5_{10} + 3_{10} = 8_{10}$



$$1 \times 2^2 + 0 \times 2^1 + 1 \times 2^0$$
$$1 \times 2^1 + 1 \times 2^0$$
$$1 \times 2^2 + 1 \times 2^1 + 10 \times 2^0$$
$$\downarrow$$
$$1 \times 2^2 + (1 + 1) \times 2^1 + 0 \times 2^0$$
$$\downarrow$$
$$(1 + 1) \times 2^2 + 0 \times 2^1 + 0 \times 2^0$$
$$\downarrow$$
$$1 \times 2^3 + 0 \times 2^2 + 0 \times 2^1 + 0 \times 2^0$$

  - $95_{10} + 16_{10} = 111_{10}$

$$\begin{array}{rccccccc} & 1 & & & & & & \\ 95_{10} = & 1 & 0 & 1 & 1 & 1 & 1 & 1 \\ +\ 16_{10} = & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ \hline & 1 & 1 & 0 & 1 & 1 & 1 & 1 = 111_{10} \end{array}$$

Sistemas de Numeração

- Operações aritméticas binárias
  - Subtração em notação posicional

$$\begin{array}{rr} & 10 \\ 9 & 5_{10} \\ -1 & 6_{10} \\ -1 & \\ \hline 7 & 9_{10} \end{array}$$

$$\begin{array}{rcl} 95 = 9 \times 10^1 + 5 \times 10^0 & = & 8 \times 10^1 + 15 \times 10^0 \\ -16 = 1 \times 10^1 + 6 \times 10^0 & = & 1 \times 10^1 + 6 \times 10^0 \\ \hline & & 7 \times 10^1 + 9 \times 10^0 \end{array}$$

  - Subtração na base 2

$0 - 0 = 0$

$0 - 1 = 1$ Um "emprestado"

$1 - 0 = 1$

$1 - 1 = 0$

Sistemas de Numeração

- Exemplos de subtração binária
  - $95_{10} - 16_{10} = 79_{10}$

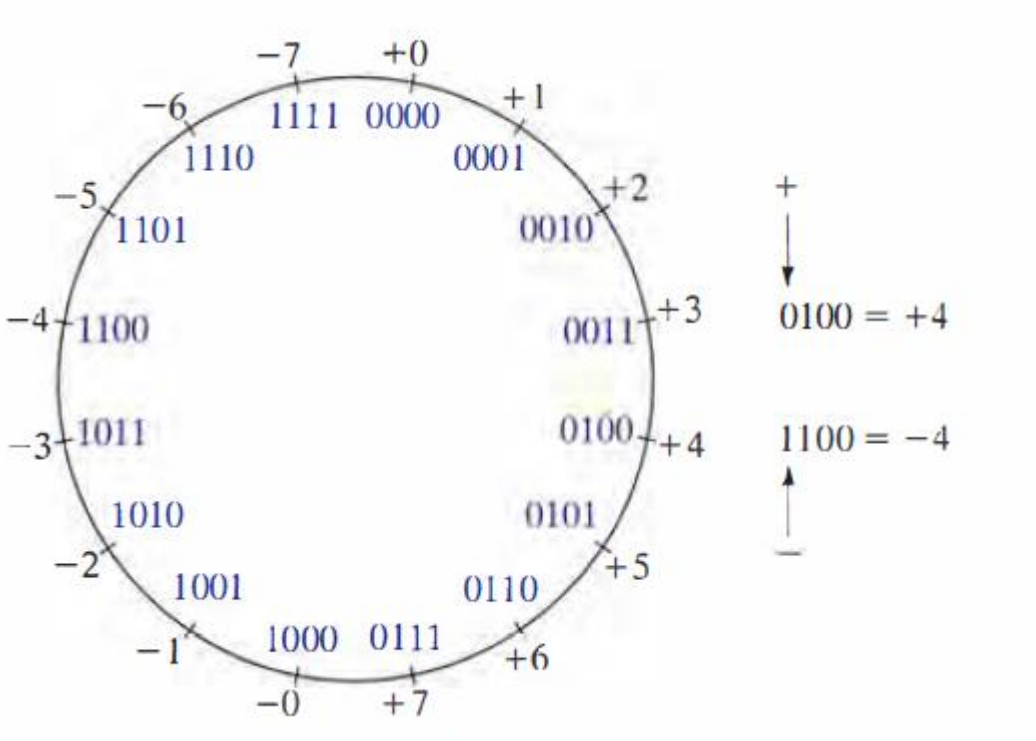
$$
\begin{array}{rrrrrrrrl}
95_{10} = & 1 & 0 & 1 & 1 & 1 & 1 & 1 & \\
-16_{10} = & -0 & 0 & 1 & 0 & 0 & 0 & 0 & \\
\hline
 & 1 & 0 & 0 & 1 & 1 & 1 & 1 & = 79_{10}
\end{array}
$$

  - $16_{10} - 1_{10} = 15_{10}$

$$
\begin{array}{rrrrrrrrl}
 & & 10 & & 10 & & 10 & & 10 \\
16_{10} = & 1 & & 0 & & 0 & & 0 & 0 \\
-\ 1_{10} = & 0 & & 0 & & 0 & & 0 & 1 \\
 & -1 & & -1 & & -1 & & -1 & \\
\hline
 & 0 & & 1 & & 1 & & 1 & 1 = 15_{10}
\end{array}
$$

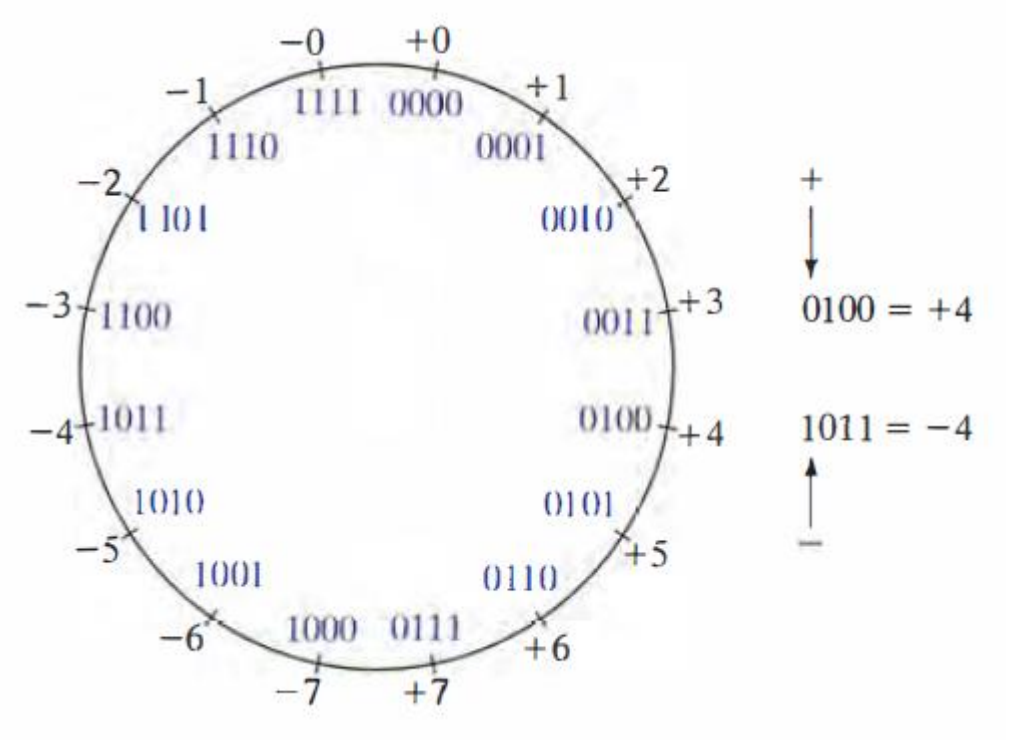
Sistemas de Numeração

- Representação de números negativos
  - Sinal e magnitude



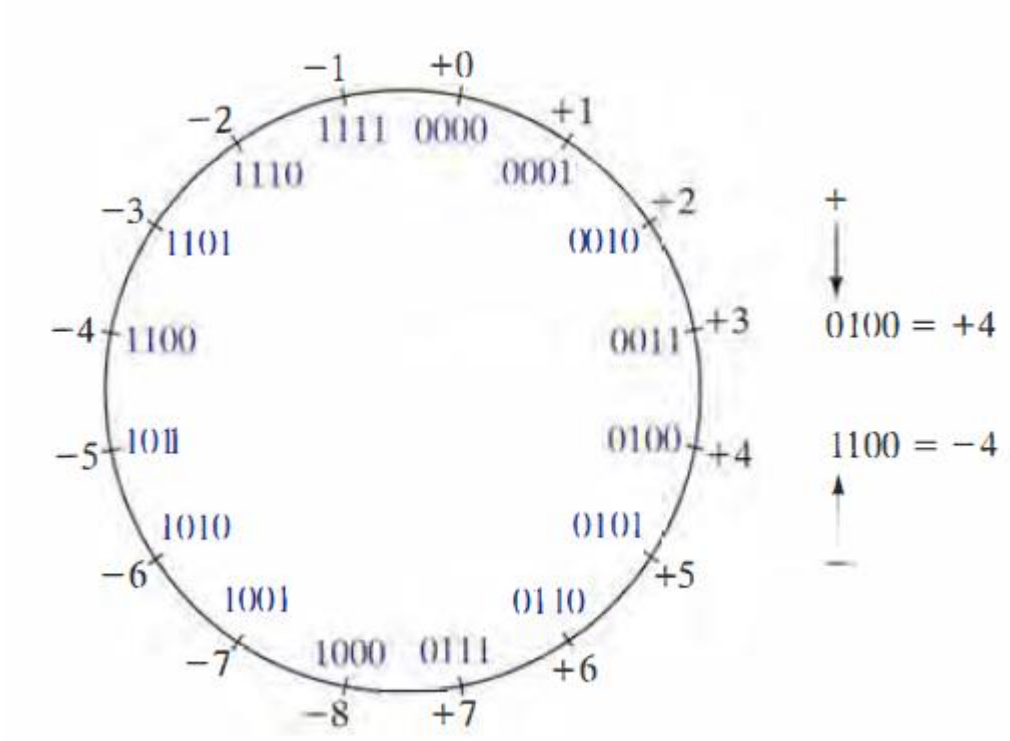
Sistemas de Numeração

- Representação de números negativos
  - Complemento de 1



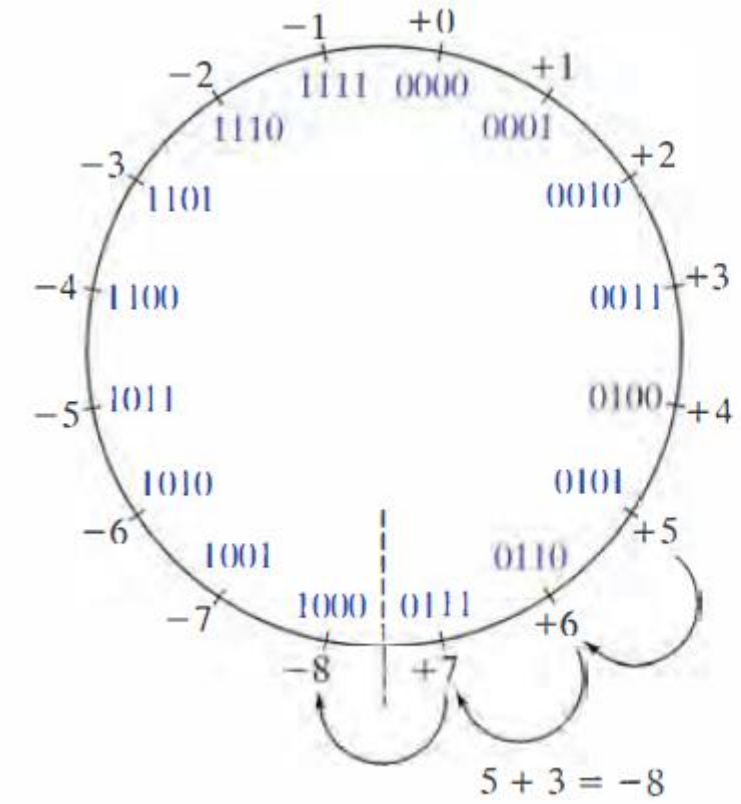
Sistemas de Numeração

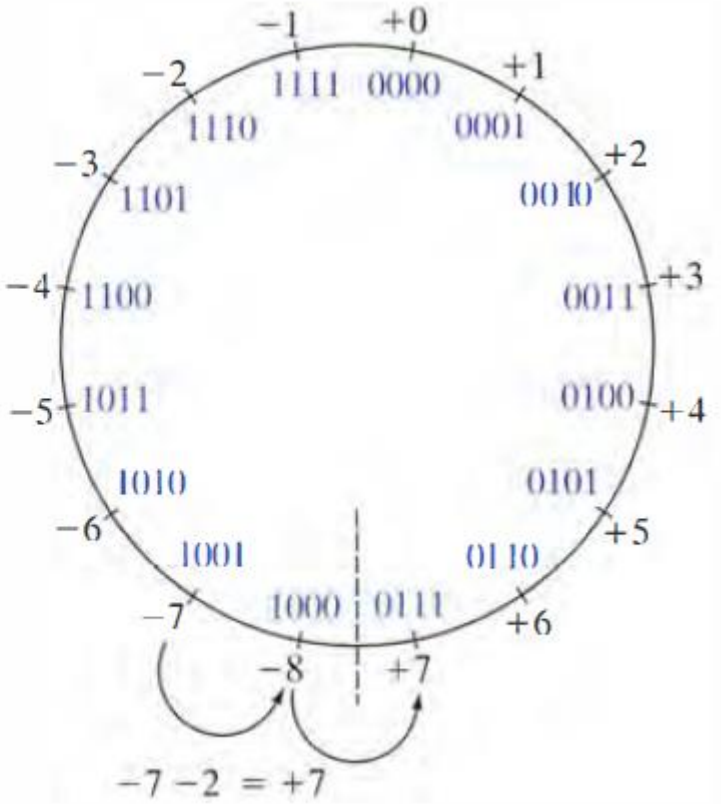
- Representação de números negativos
  - Complemento de 2



Sistemas de Numeração

- Condições de *overflow*





Sistemas de Numeração

- Representação BCD (Binário Codificado em Decimal)

$$\begin{array}{rl} 5 = & 0101 \\ 3 = & 0011 \\ \hline & 1000 \end{array} = 8 \qquad \begin{array}{rl} 5 = & 0101 \\ 8 = & 1000 \\ \hline & 1101 \end{array} = 13!$$

- Quando soma excede 9, soma-se 6

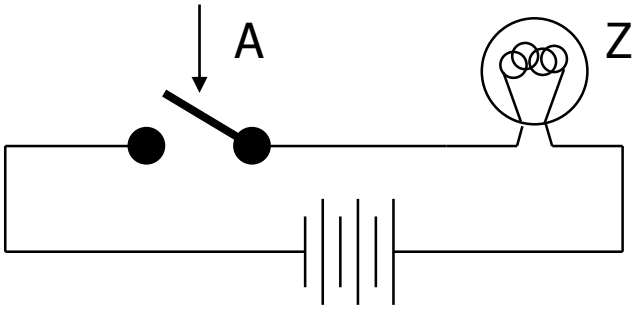
$$\begin{array}{rl} 5 = & 0101 \\ 8 = & 1000 \\ \hline & 1101 \\ + & 0110 \\ \hline & 10011 \end{array} \begin{array}{l} = 13 \text{ in decimal} \\ \\ = 13 \text{ in BCD} \end{array} \qquad \begin{array}{rl} 9 = & 1001 \\ 7 = & 0111 \\ \hline & 1\ 0000 \\ + & 0110 \\ \hline & 10110 \end{array} \begin{array}{l} = 16 \text{ in decimal} \\ \\ = 16 \text{ in BCD} \end{array}$$

Computação: abstração vs. implementação

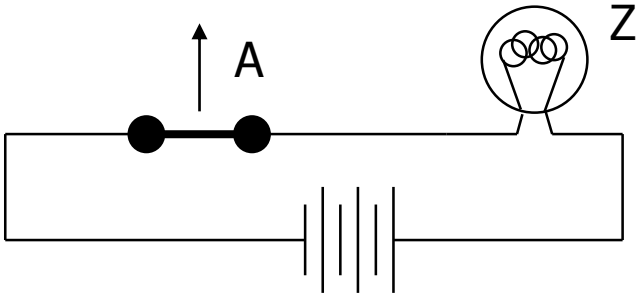
- Implementar fisicamente computação utilizando dispositivos reais que utilizam tensão para representar valores lógicos
- Unidades básicas de computação:
  - Representação:
    - "0", "1" em um fio ou conjunto de fios (inteiros binários)
  - Assinalmento: x = y
  - Operações sobre dados: x + y – 5
  - Controle:
    - Declarações sequenciais: A; B; C
    - Condicionais: se x == 1 então y
    - Laços de repetição: para (i = 1 ; i == 10, i++)
    - Procedimentos: A; proc(...); B;
- Vamos estudar como implementar estas estruturas em um hardware composto por estruturas computacionais

Chaves: Elemento básico de implementações físicas

- Implementação de um circuito simples:
  - Setas indicam ação se o fio mudar para “1”



Fecha a chave (se A é “1”) e liga a lâmpada (Z)



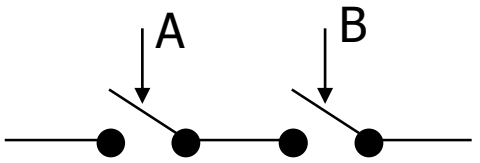
Abre a chave (se A é “0”) e desliga a lâmpada (Z)

$Z \equiv A$

Chaves

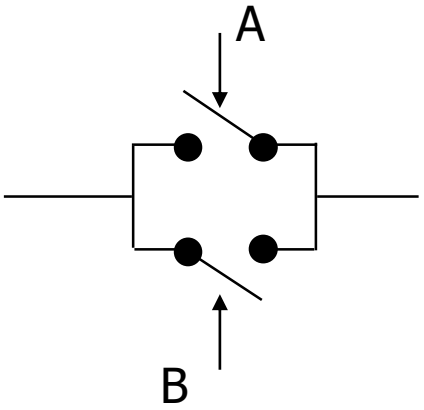
- Composição de chaves em circuitos mais complexos (funções booleanas)

E



Z ≡ A e B

OU



Z ≡ A ou B

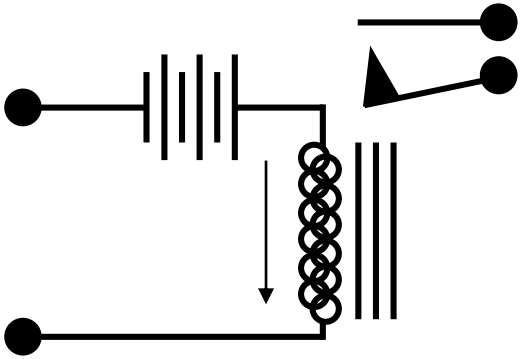
Redes de chaves

- Configurações das chaves
  - Determinar se existe ou não um caminho condutor que alimente a lâmpada
- Computações maiores
  - Utilizar a lâmpada (saída da rede) para configurar outras chaves (entradas de outra rede).
- Conectar redes de chaves
  - Construir redes de chaves mais complexas

Redes de relés

- Uma forma simples de converter caminhos condutores em chaveamento é utilizando relés eletromecânicos
- O que é um relé?

Caminho condutor composto por chaves fecha o circuito



Fluxo de corrente na bobina magnetiza o núcleo, resultando na abertura de um contato normalmente fechado (NF)

Quando não existe fluxo de corrente, a mola do contato faz com que ele retorne a sua posição normal

O que determina a velocidade de chaveamento de uma rede de relés?

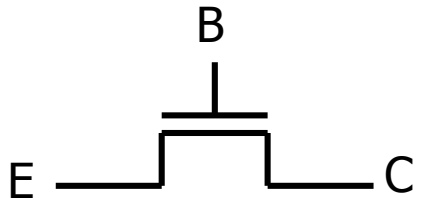
Redes de transistores

- Relés não são mais comumente utilizados
  - Alguns controladores de semáforos ainda são eletromecânicos
- Sistemas digitais modernos são implementados em tecnologia CMOS
  - MOS significa Óxido-Metal em Semiconductor
  - C é complementar pois existem tanto chaves normalmente abertas quanto chaves normalmente fechadas
- Transistores MOS funcionam como chaves controladas por tensão
  - Similar, mas mais fáceis de trabalhar do que relés.

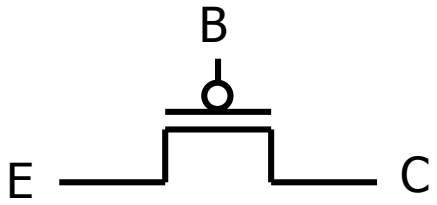
Transistores MOS

- Transistores MOS são compostos por três terminais: base, emissor e coletor.
  - Estes terminais se comportam como chaves, da seguinte maneira:

    Se a tensão no terminal base é superior/inferior ao terminal emissor, então o caminho de condução será estipulado entre os terminais coletor e emissor.

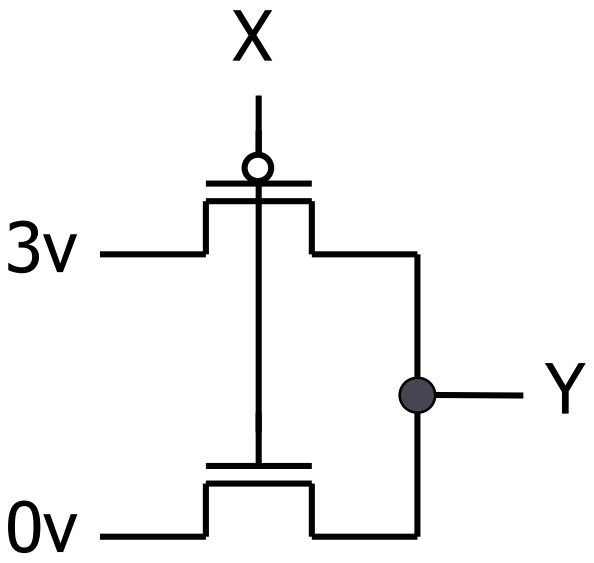


Canal-n
Aberto quando tensão de B é baixo
Fecha quando:
Tensão (B) > Tensão (E) + ε



Canal-p
Fechado quando tensão de B é baixo
Aberto quando:
Tensão (B) < Tensão (E) – ε

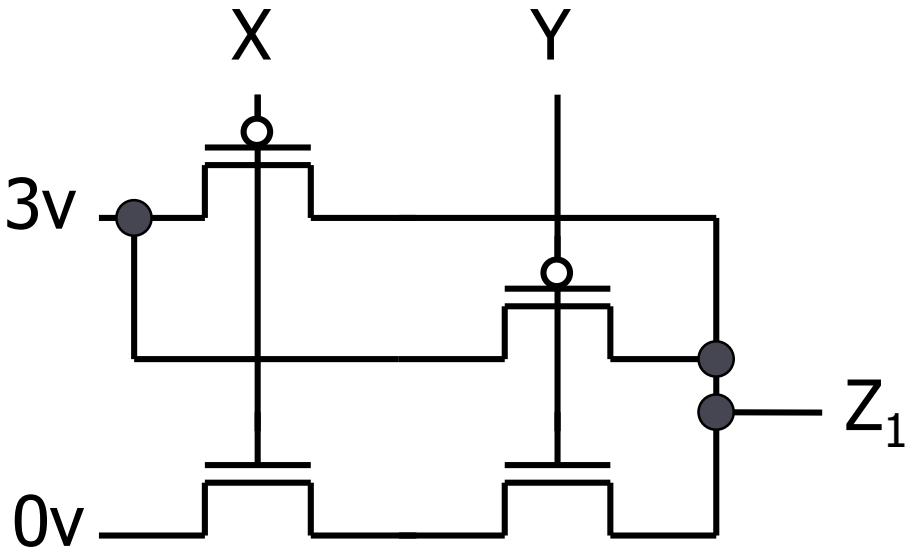
Redes MOS

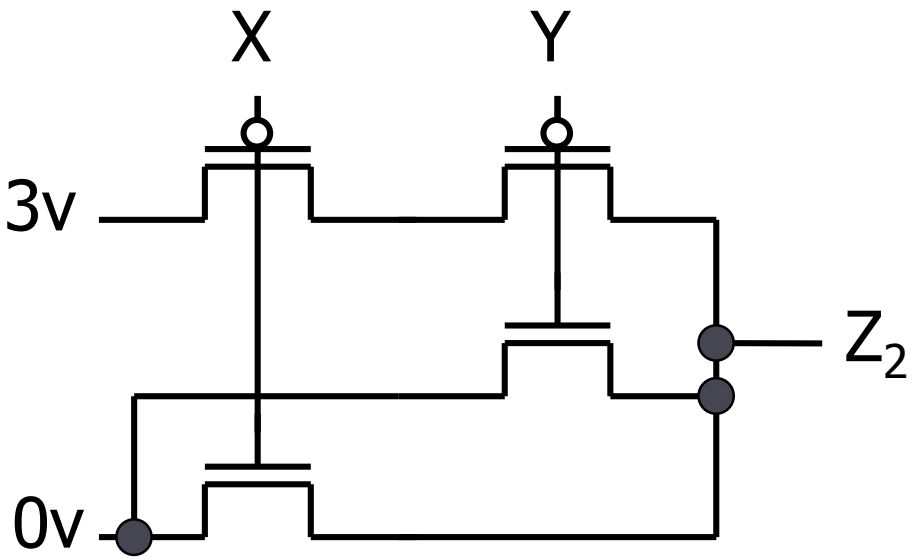


Qual é a relação entre x e y?

x	y
0 volts	
3 volts	

Redes com duas entradas





Qual é a relação entre x, y, e z?

x	y	z1	z2
0 volts	0 volts		
0 volts	3 volts		
3 volts	0 volts		
3 volts	3 volts		

Aceleração de redes MOS

- O que influencia a aceleração de redes CMOS ?
  - Carregamento e descarregamento da tensão nos fios e portas dos transistores
- Capacitores retém carga
  - Capacitância está nas portas dos transistores e no material do fio
- Resistores com movimento lento de elétrons
  - Resistência principalmente devido aos transistores

Representação de projetos digital

- Dispositivos físicos (transistores, relés)
- Chaves
- Tabela verdade
- Álgebra booleana
- Portas
- Forma de ondas
- Comportamento de estado finito
- Comportamento transferência-registrador
- Especificações abstratas simultâneas

CCF 251

Digital vs. analógico

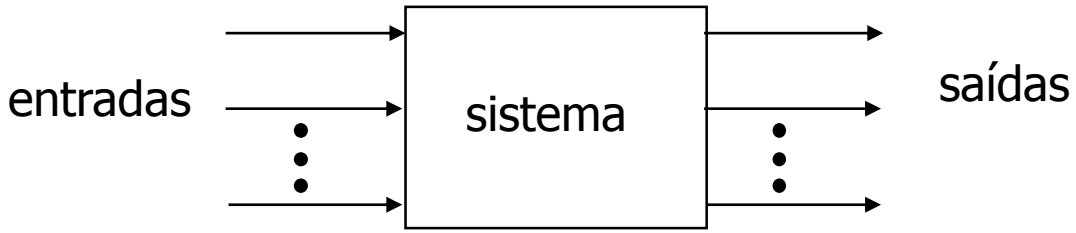
- Conveniente pensar em sistemas digitais como tendo somente valores discretos, digital e entrada/saída
- Na realidade, componentes eletrônicos reais apresentam comportamento contínuo e analógico
- Por que fazer a abstração digital de qualquer maneira?
  - Chaves operam desta maneira
  - Mais fácil pensar sobre um número pequeno de valores discretos
- Por que trabalhar com abstração digital?
  - Não propaga erros pequenos nos valores
  - Sempre inicializa com 0 ou 1

Mapeamento a partir do mundo físico para o mundo binário

Tecnologia	Estado 0	Estado 1
Lógica relé	Circuito aberto	Circuito fechado
Lógica CMOS	0.0-1.0 volts	2.0-3.0 volts
Lógica transistor transistor (TTL)	0.0-0.8 volts	2.0-5.0 volts
Fibra óptica	luz desligada	Luz acesa
RAM dinâmica	Capacitor descarregado	Capacitor carregado
Memória não volátil	Elétrons presos	Nenhum elétron preso
ROM programável	Fusível queimado	Fusível intacto
Disco magnético	Nenhuma reversão de fluxo	Reversão de fluxo

Circuito digital
Combinacional vs. sequencial

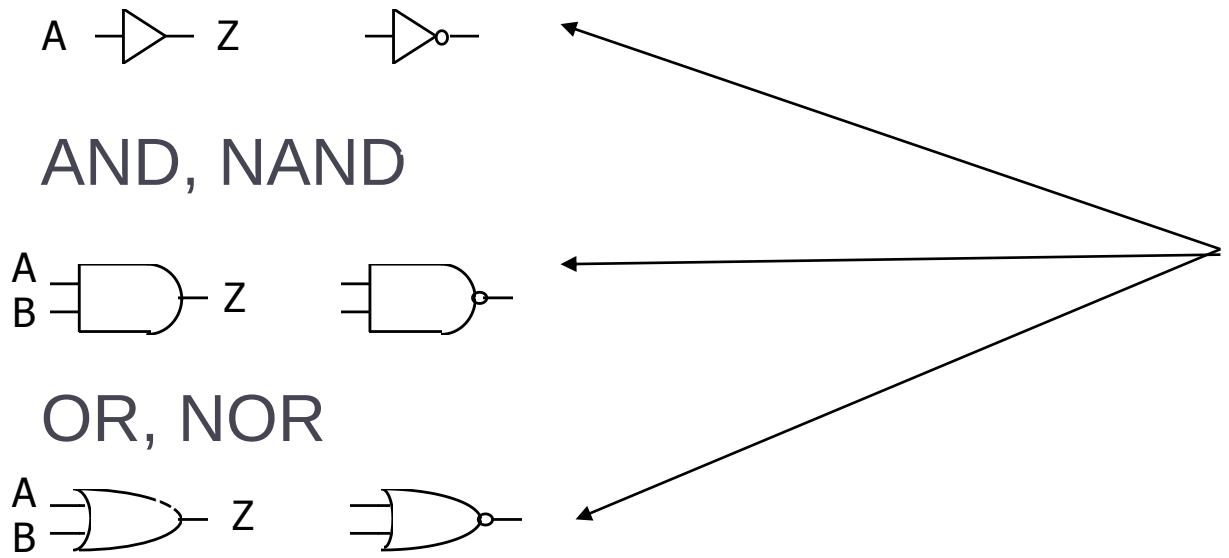
- Um modelo simples de um sistema digital é uma unidade com entradas e saídas:



- Combinacional significa “menos memória”
  - Um circuito digital é combinacional se o valor da saída somente depende do valor da entrada

Símbolo lógico combinacional

- Sistemas lógico combinacional comuns tem símbolos padronizados denominados como portas lógicas
  - Buffer, NOT
  - AND, NAND
  - OR, NOR



Fácil de implementar com transistores CMOS

Lógica sequencial

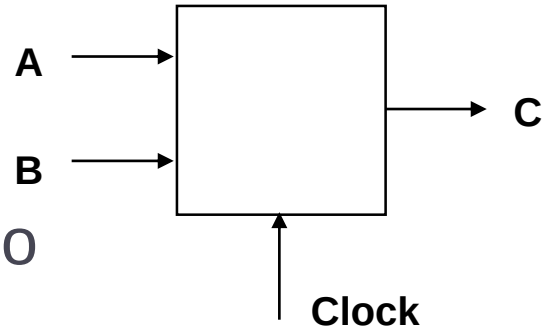
- Sistemas sequenciais
  - Apresentam comportamentos (saídas) que dependem não somente dos valores da entrada atual, mas também dos valores da entrada anterior
- Na realidade, todos os circuitos reais são sequenciais
  - Porque as saídas não se alteram instantaneamente depois de uma alteração na entrada
  - Por que não? E por que eles são então sequenciais?
- Uma abstração muito importante de projeto digital é a razão sobre os comportamentos de estado-estável
  - Olhar para as saídas somente após ter decorrido o tempo suficiente para que o sistema faça as alterações necessárias e se estabeleça

Sistemas digitais sequenciais síncronos

- Saídas de um circuito combinacional dependem somente das entradas atuais
  - Após ter decorrido o tempo suficiente
- Circuitos sequenciais tem memória
- A abstração de estado-estável é tão útil que a maioria dos designers utilizam uma forma de ao construir circuitos sequenciais

Exemplo de lógica combinacional e sequencial

- Combinacional
  - Entrada A, B
  - Espera por borda de clock
  - Observa C
  - Espera por outra borda de clock
  - Observa C novamente: ficará no mesmo
- Sequencial
  - Entrada A, B
  - Espera por borda de clock
  - Observa C
  - Espera por outra borda de clock
  - Observa C novamente: pode ser diferente



Abstrações

- Algumas já foram vistas
  - Interpretação digital de valores analógicos
  - Transistores como chaves
  - Chaves como portas lógicas
  - Uso de um *clock* para compreender um circuito sequencial síncrono
- Alguns outros nós veremos
  - Tabelas verdade e álgebra booleana para representar lógica combinacional
  - Codificação de sinais com mais de dois valores lógicos em forma binária
  - Diagrama de estado para representar lógica sequencial
  - Linguagem de descrição de hardware para representar lógica digital
  - Formas de onda para representar comportamento digital

Um exemplo

- Subsistema de calendário: número de dias em um mês (controlar via display do relógio)
  - Entradas: mês, flag ano bissexto
  - Saídas: número de dias

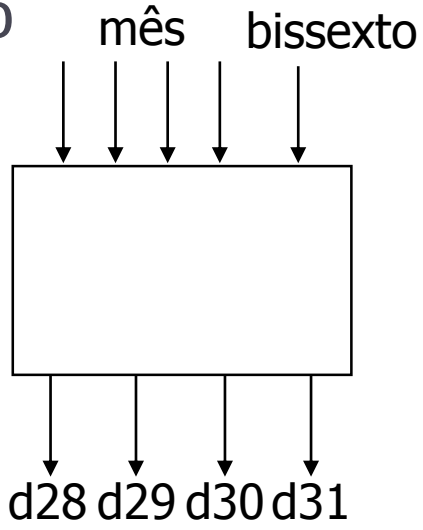
Implementação em software

```
integer number_of_days ( month,
  leap_year_flag) {
  switch (month) {
    case 1: return (31);
    case 2: if (leap_year_flag == 1) then return
      (29)
                                    else return
      (28);
    case 3: return (31);
    ...
    case 12: return (31);
    default: return (0);
  }
}
```

Implementação com um sistema digital combinacional

- Codificação:
  - Quantos bits para cada entrada/saída?
  - Número binário para mês
  - Quatro fios para 28, 29, 30, e 31
- Comportamento:
  - Combinacional
  - Especificação

tabela verdade



Mês	bissexto	d28	d29	d30	d31
0000	–	–	–	–	–
0001	–	0	0	0	1
0010	0	1	0	0	0
0010	1	0	1	0	0
0011	–	0	0	0	1
0100	–	0	0	1	0
0101	–	0	0	0	1
0110	–	0	0	1	0
0111	–	0	0	0	1
1000	–	0	0	0	1
1001	–	0	0	1	0
1010	–	0	0	0	1
1011	–	0	0	1	0
1100	–	0	0	0	1
1101	–	–	–	–	–
111–	–	–	–	–	–

Exemplo combinacional

- Tabela-verdade para lógica, chaves e portas
  - d28 = 1 quando mês=0010 e bissexto=0
  - d28 = m8'•m4'•m2•m1'•bissexto'
  - d31 = 1 quando mês=0001 ou mês=0011 or ... mês =1100
  - d31 = (m8'•m4'•m2'•m1) + (m8'•m4'•m2•m1) + ... (m8•m4•m2'•m1')
  - d31 = Nós podemos simplificar mais?

Símbolo para not

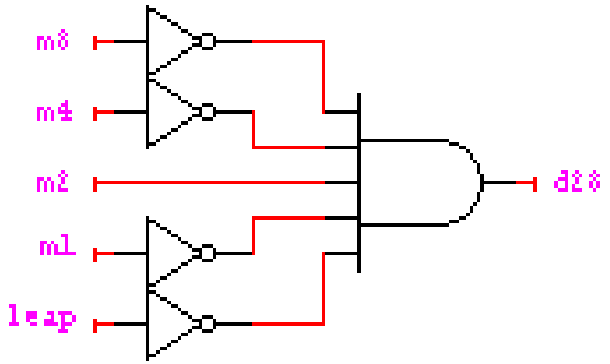
Símbolo para and

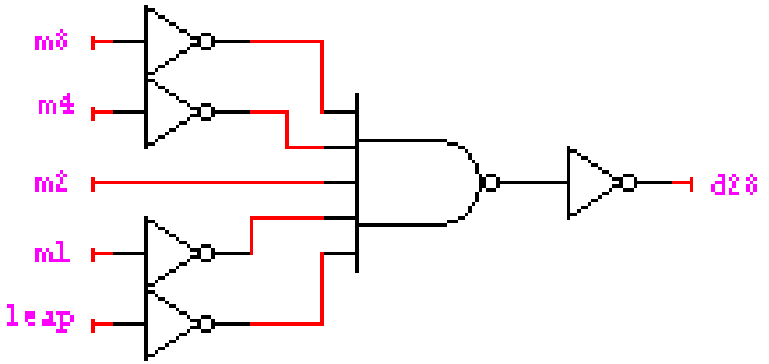
Símbolo para or

Mês	bissexto	d28	d29	d30	d31
0001	–	0	0	0	1
0010	0	1	0	0	0
0010	1	0	1	0	0
0011	–	0	0	0	1
0100	–	0	0	1	0
...					
1100	–	0	0	0	1
1101	–	–	–	–	–
111–	–	–	–	–	–
0000	–	–	–	–	–

Exemplo combinacional

- d28 = m8'•m4'•m2•m1'•bissexto’
- d29 = m8'•m4'•m2•m1'• bissexto
- d30 = (m8'•m4•m2'•m1') + (m8'•m4•m2•m1') +
  (m8•m4'•m2'•m1) + (m8•m4'•m2•m1)
  = (m8'•m4•m1') + (m8•m4'•m1)
- d31 = (m8'•m4'•m2'•m1) + (m8'•m4'•m2•m1) +
  (m8'•m4•m2'•m1) + (m8'•m4•m2•m1) +
  (m8•m4'•m2'•m1') + (m8•m4'•m2•m1') +
  (m8•m4•m2'•m1')





Atividade

- Quanto nós podemos simplificar d31?

- E se nós iniciarmos os meses com 0 ao invés de 1? (Ex: Janeiro é 0000 e Dezembro é 1011)

Atividade

- Quanto nós podemos simplificar d31?

d31 é verdade se: o mês é 7 ou menor e ímpar (1, 3, 5, 7), ou mês é 8 ou maior e igual (8, 10, 12, incluindo o 14)

d31 é verdade se: m8 é 0 e m1 é 1, ou m8 é 1 e m1 é 0

d31 = m8′m1 + m8m1′

- E se nós iniciarmos os meses com 0 ao invés de 1? (Ex: Janeiro é 0000 e Dezembro é 1011)

Expressão mais complexa (0, 2, 4, 6, 7, 9, 11):

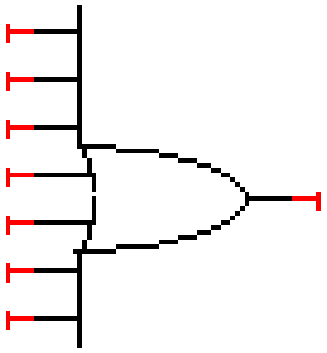
d31 = m8′m4′m2′m1′ + m8′m4′m2m1′ + m8′m4m2′m1′ + m8′m4m2m1′ + m8′m4m2m1 + m8m4′m2′m1 + m8m4′m2m1

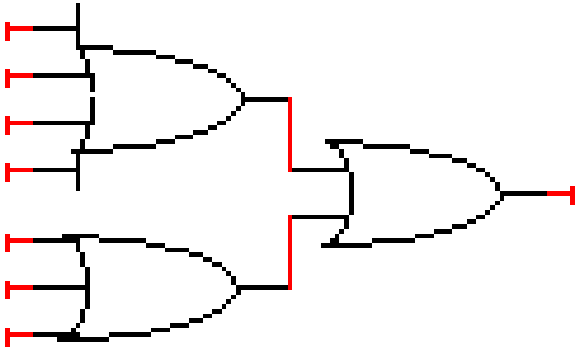
d31 = m8′m1′ + m8′m4m2 + m8m1 (incluindo 13 e 15)

d31 = (d28 + d29 + d30)′

Exemplo combinacional

- d28 = m8'•m4'•m2•m1'•bissexto’
- d29 = m8'•m4'•m2•m1'•bissexto
- d30 = (m8'•m4•m2'•m1') + (m8'•m4•m2•m1') +
  (m8•m4'•m2'•m1) + (m8•m4'•m2•m1)
- d31 = (m8'•m4'•m2'•m1) + (m8'•m4'•m2•m1) +
  (m8'•m4•m2'•m1) + (m8'•m4•m2•m1) +
  (m8•m4'•m2'•m4') + (m8•m4'•m2•m1') +
  (m8•m4•m2'•m1')





Outro exemplo

- Travar porta com combinação
  - Colocando 3 valores em sequência a porta se abre. Se houver um erro o bloqueio deve ser redefinido. Uma vez a porta aberta o bloqueio também deve ser redefinido.

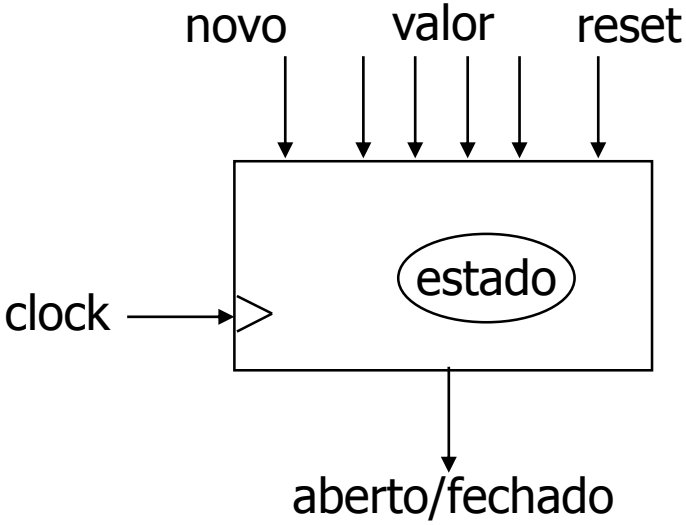
  - entradas: sequência de valores de entrada, reset
  - saídas: porta aberta/fechada
  - memória: precisa lembrar a combinação ou sempre tê-la disponível como uma entrada

Implementação em software

```
integer combination_lock ( ) {
   integer v1, v2, v3;
   integer error = 0;
   static integer c[3] = 3, 4, 2;

   while (!new_value( ));
   v1 = read_value( );
   if (v1 != c[1]) then error = 1;

   while (!new_value( ));
   v2 = read_value( );
   if (v2 != c[2]) then error = 1;

   while (!new_value( ));
   v3 = read_value( );
   if (v3 != c[3]) then error = 1;

   if (error == 1) then return(0); else return (1);
}
```

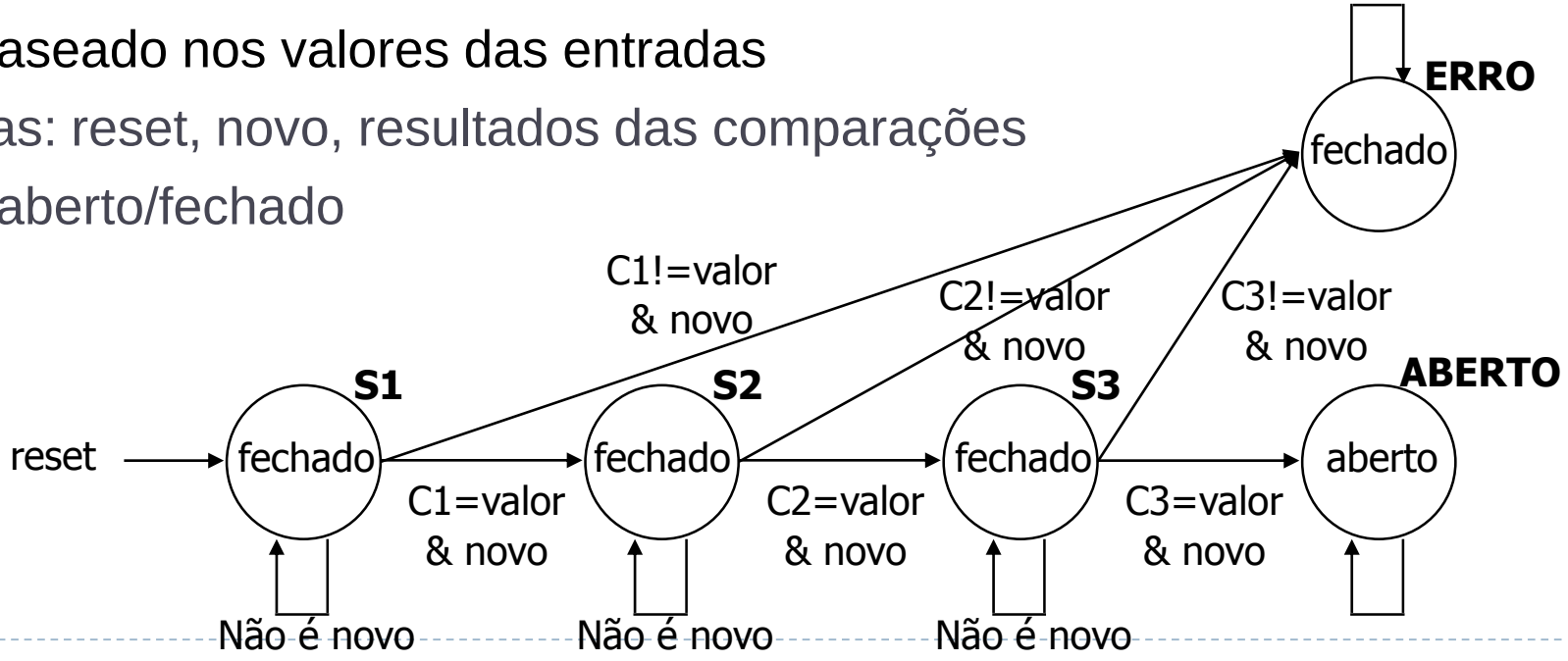
Implementação como um sistema digital sequencial

- Codificação:
  - Quantos bits por valor de entrada?
  - Quantos valores em sequência?
  - Como nós sabemos que um novo valor de entrada é introduzido?
  - Como nós representamos os estados do sistema?
- Comportamento:
  - O fio de *clock* nos diz quando está ok, para olhar as entradas
  - Sequencial: sequência de valores que precisam ser inseridos
  - Sequencial: lembrar se ocorreu um erro
  - Especificação do estado-finito



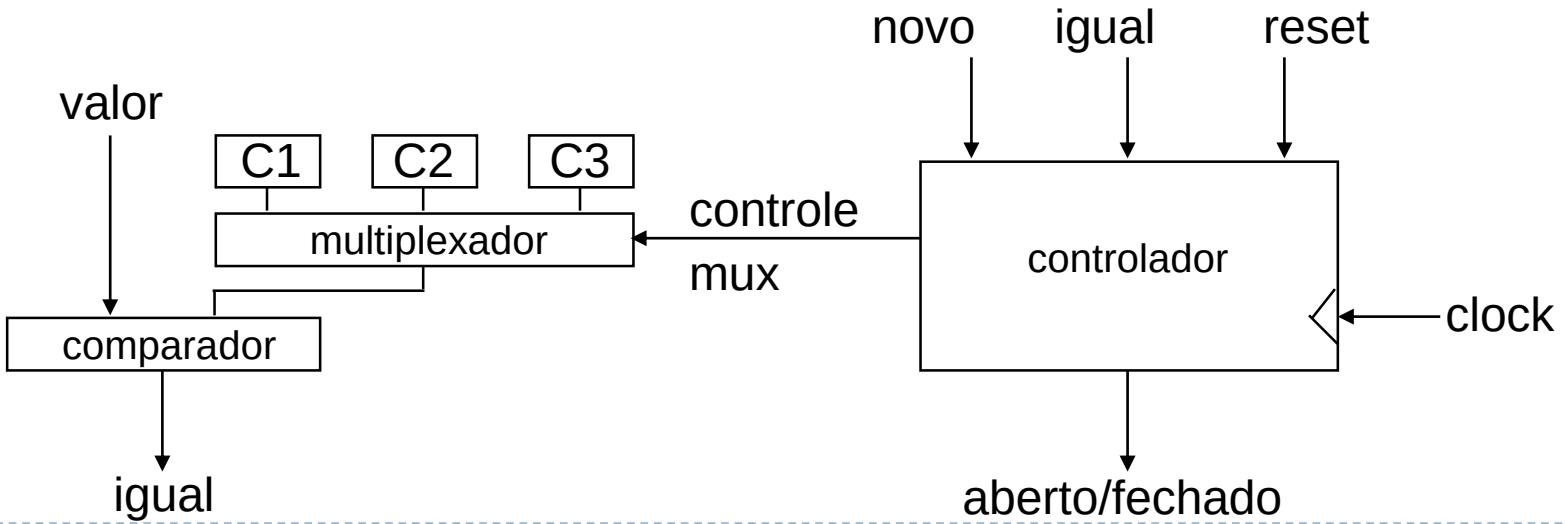
Exemplo sequencial – Controle abstrato

- Diagrama estado-finito
  - estados: 5 estados
    - Representa o ponto em execução da máquina
    - Cada estado tem saídas
  - transições: 6 de estado para estado, 5 transições para o próprio estado, 1 global
    - As trocas de estado ocorrem quando *clock* diz ok
    - Baseado nos valores das entradas
  - entradas: reset, novo, resultados das comparações
  - saída: aberto/fechado



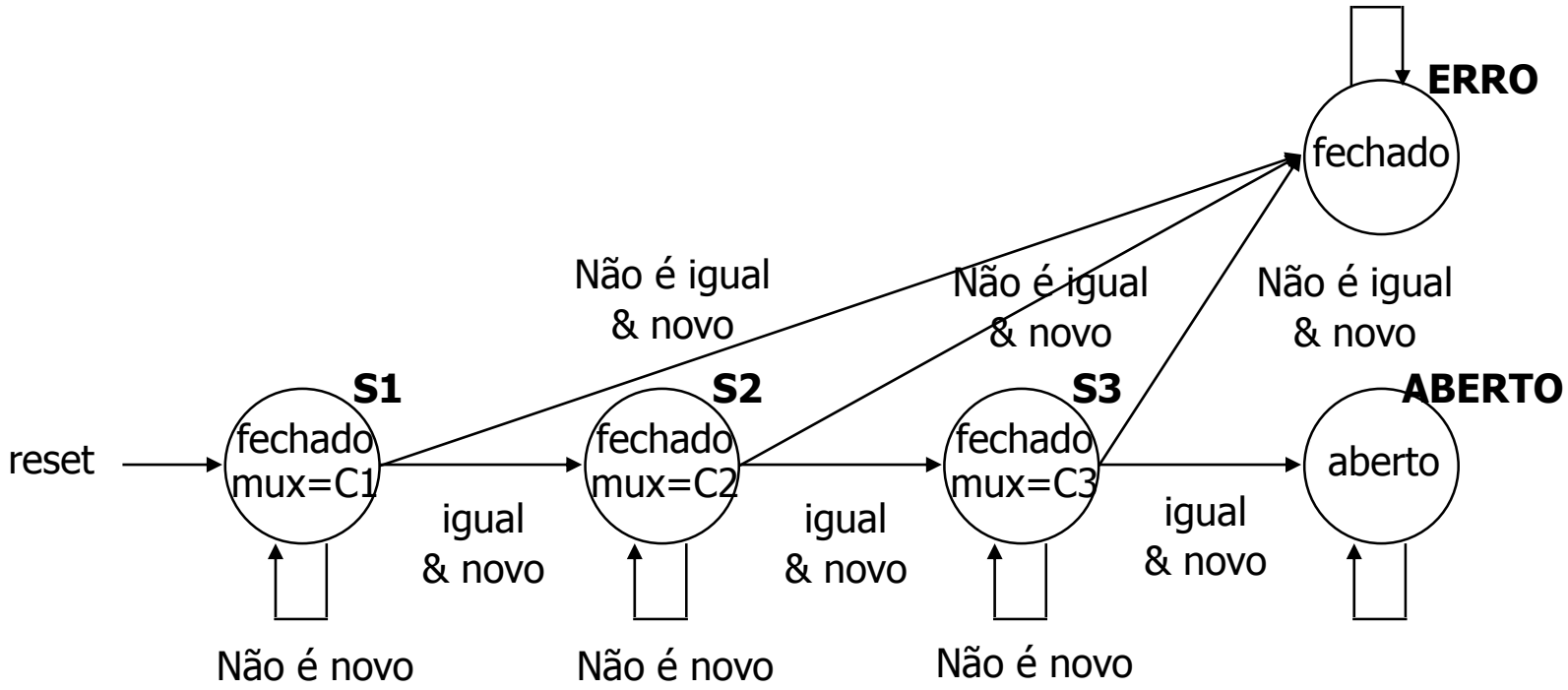
Exemplo sequencial – Caminho de dados vs. controle

- Estrutura interna
  - Caminho de dados
    - Armazenamento por combinação
    - Comparadores
  - Controle
    - Controlador máquina de estado-finito
    - Controle para o caminho de dados
    - Alterações de estado são controlado por clock



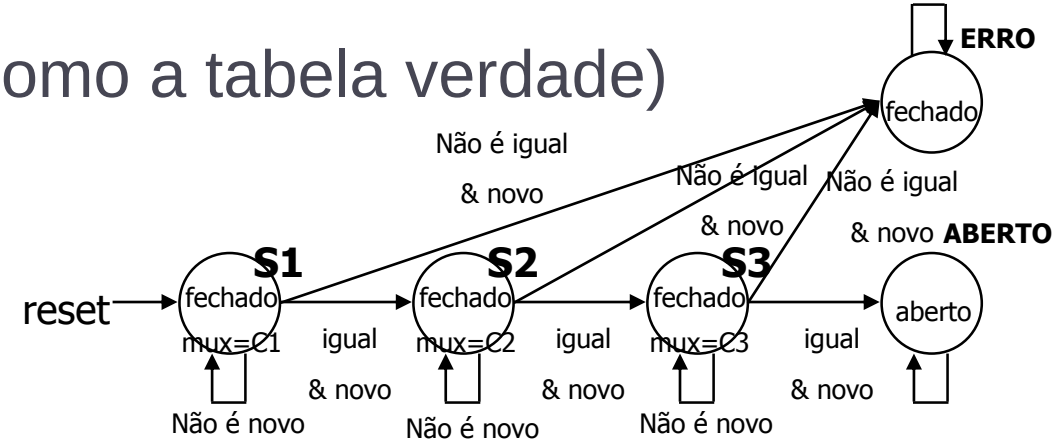
Exemplo sequencial – Máquina de estado-finito

- Máquina de estado-finito
  - Diagrama de estado aperfeiçoado para incluir estrutura interna



Exemplo sequencial – Máquina de estado-finito

- Máquina de estado-finito
  - Tabela de estado gerada (como a tabela verdade)



reset	novo	igual	estado	próximo estado	mux	aberto/fechado
1	–	–	–	S1	C1	fechado
0	0	–	S1	S1	C1	fechado
0	1	0	S1	ERRO	–	fechado
0	1	1	S1	S2	C2	fechado
0	0	–	S2	S2	C2	fechado
0	1	0	S2	ERRO	–	fechado
0	1	1	S2	S3	C3	fechado
0	0	–	S3	S3	C3	fechado
0	1	0	S3	ERRO	–	fechado
0	1	1	S3	ABERTO	–	aberto
0	–	–	ABERTO	ABERTO	–	aberto
0	–	–	ERRO	ERRO	–	fechado

Exemplo sequencial – codificação

- Tabela de estado codificada
  - Estado pode ser: S1, S2, S3, ABERTO, ou ERRO
    - Precisa ter no mínimo 3 bits para codificação: 000, 001, 010, 011, 100
    - e para iguais ou maiores que 5: 00001, 00010, 00100, 01000, 10000
    - Escolher 4 bits: 0001, 0010, 0100, 1000, 0000
  - Saída mux pode ser: C1, C2, or C3
    - Precisa de 2 a 3 bits para codificação
    - Escolher 3 bits: 001, 010, 100
  - Saída aberto/fechado pode ser: aberto ou fechado
    - Precisa de 1 ou 2 bits para codificação
    - Escolher 1 bit: 1, 0

Exemplo sequencial – codificação

- Tabela de estado codificada
  - Estado pode ser: S1, S2, S3, ABERTO, ou ERRO
    - Escolher 4 bits: 0001, 0010, 0100, 1000, 0000
  - Saída mux pode ser: C1, C2, ou C3
    - Escolher 3 bits: 001, 010, 100
  - Saída aberto/fechado pode ser: aberto ou fechado
    - Escolher 1 bit: 1, 0

reset	novo	igual	estado	próximo estado	mux	aberto/fechado
1	–	–	–	0001	001	0
0	0	–	0001	0001	001	0
0	1	0	0001	0000	–	0
0	1	1	0001	0010	010	0
0	0	–	0010	0010	010	0
0	1	0	0010	0000	–	0
0	1	1	0010	0100	100	0
0	0	–	0100	0100	100	0
0	1	0	0100	0000	–	0
0	1	1	0100	1000	–	1
0	–	–	1000	1000	–	1
0	–	–	0000	0000	–	0

Boa escolha de codificação!

mux é idêntico aos últimos 3 bits do estado

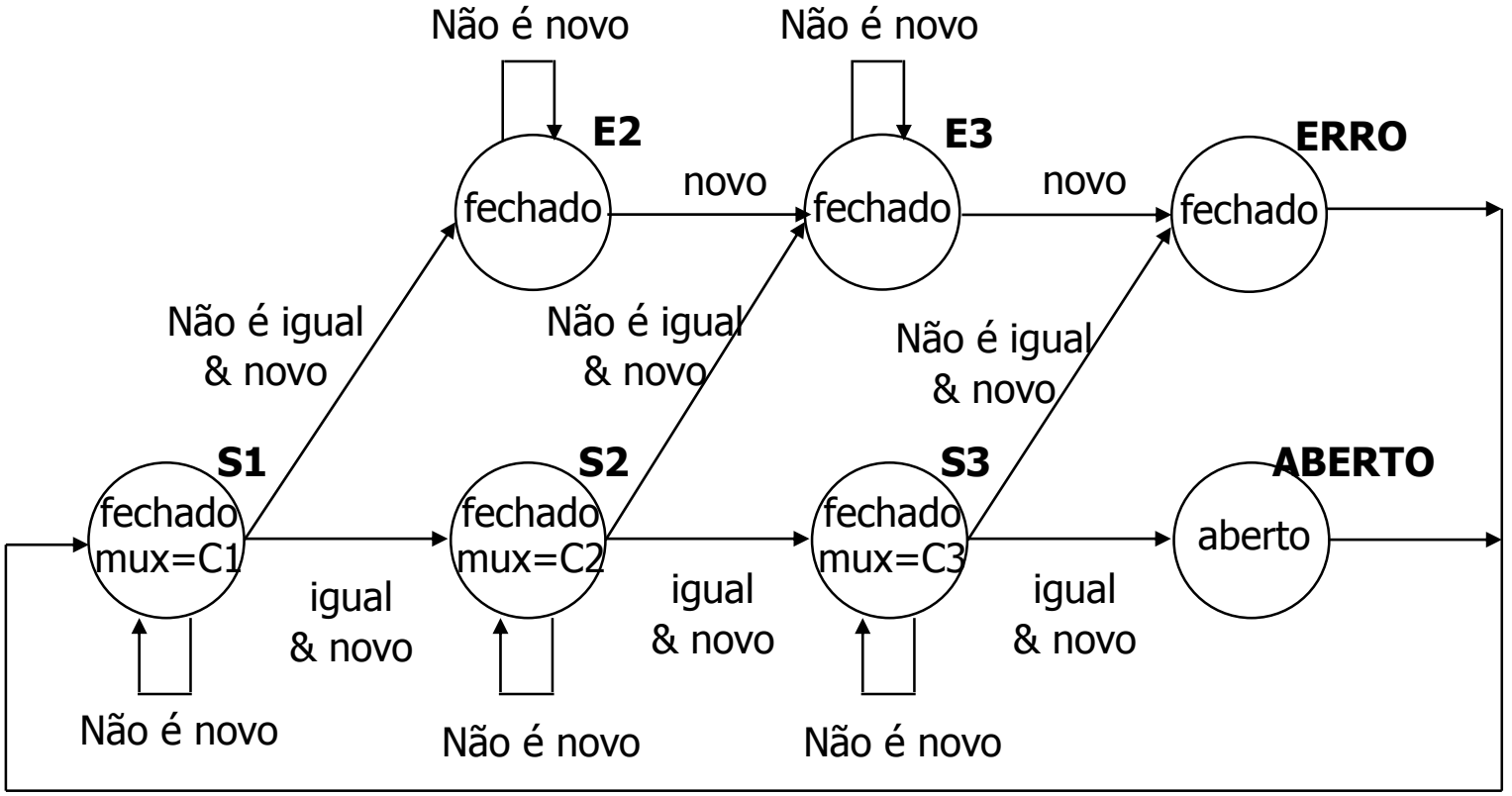
aberto/fechado é idêntico ao primeiro bit do estado

Atividade

- Ter a trava sempre esperando por 3 chaves pressionadas antes de tomar uma decisão

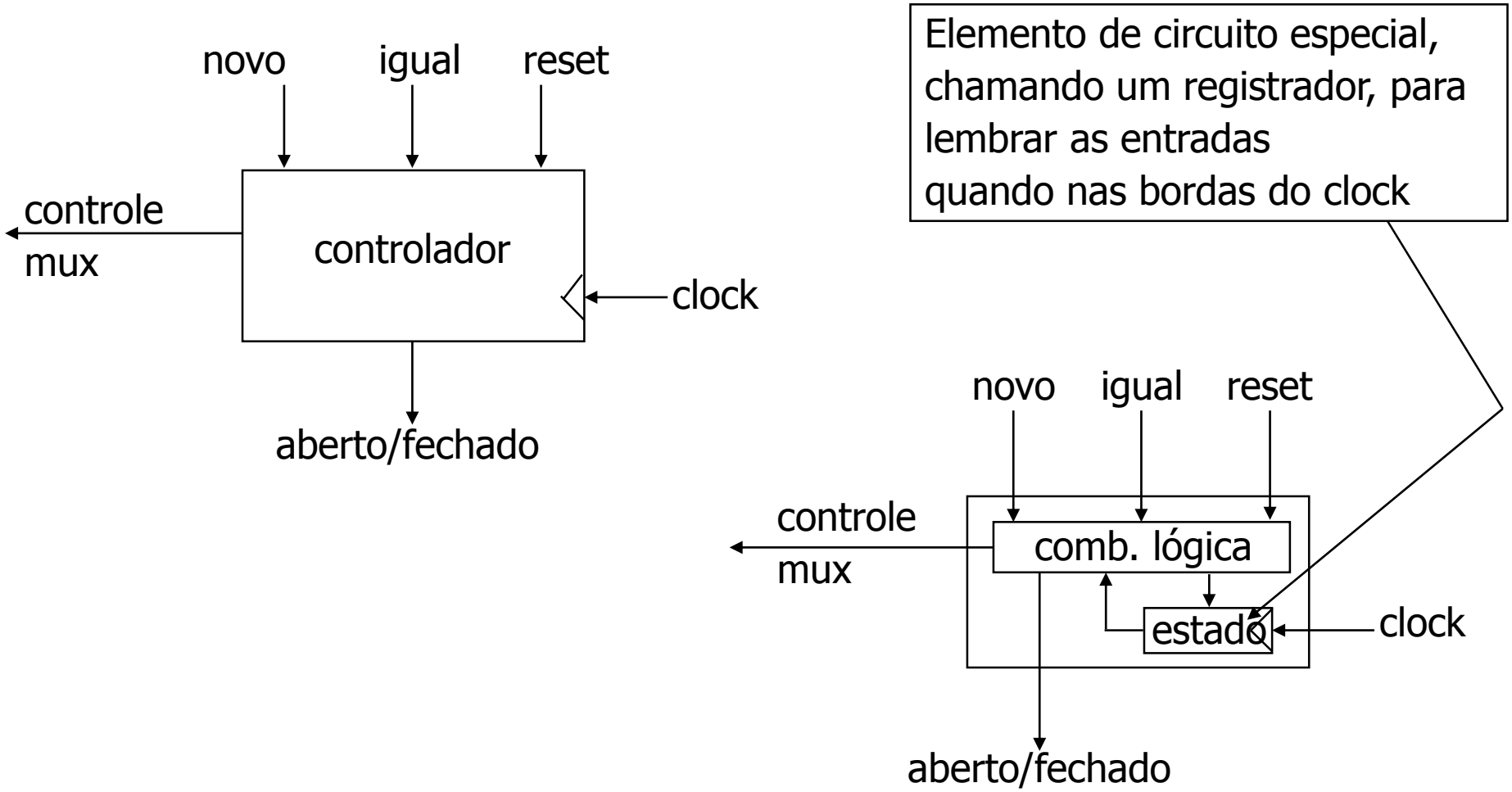
Atividade

- Ter a trava sempre esperando por 3 chaves pressionadas antes de tomar uma decisão
  - Remover reset



Exemplo sequencial – implementação controlador

- Implementação do controlador



Hierarquia de projeto