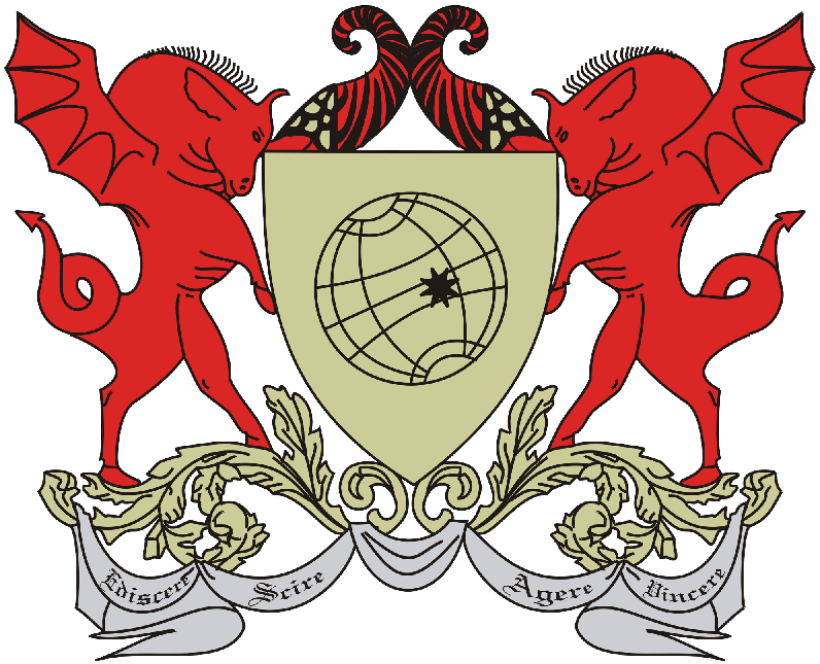


CCF 251 – Introdução aos Sistemas Lógicos

Aula 06 – Estudos de caso de projeto de lógica combinacional

Prof. José Augusto Nacif – jnacif@ufv.br

Projeto lógica combinacional estudo de casos

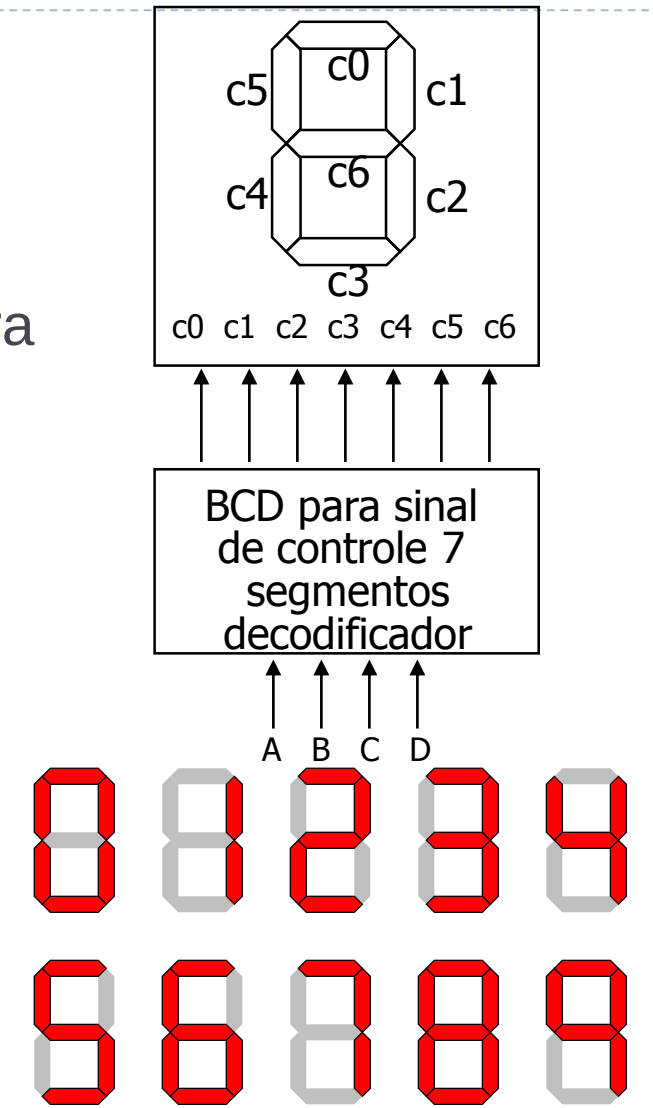
- Procedimento de projeto geral
- Estudo de casos
  - BCD para controlador display 7 segmentos
  - Unidade de função lógica
  - Controlador de linha de processo
  - Subsistema calendário
- Circuitos aritméticos
  - Representações inteiras
  - Adição/subtração
  - Unidades aritméticas/lógicas

Procedimento de projeto geral para a lógica combinacional

- 1. Entenda o problema
  - O que o circuito deveria fazer?
  - Anotar entradas (dados, controle) e saídas
  - Desenhar diagrama de blocos ou outra imagem
- 2. Formular o problema usando uma representação de projeto apropriado
  - Tabela verdade ou diagrama de formas de onda são típicos
  - Pode exigir a codificação de entradas e saídas simbólicas
- 3. Escolha o alvo da implementação
  - ROM, PAL, PLA
  - mux, decodificador e porta OR
  - Portas discretas
- 4. Siga o procedimento de execução
  - K-maps para dois níveis, multi níveis
  - Ferramentas de projeto e linguagem de descrição de hardware (por exemplo, Verilog)

BCD para controlador display 7 segmentos

- Compreender o problema
  - Entrada é um dígito BCD de 4 bits (A, B, C, D)
  - Saída são os sinais de controle para o display (7 saídas C0 – C6)
- Diagrama de bloco



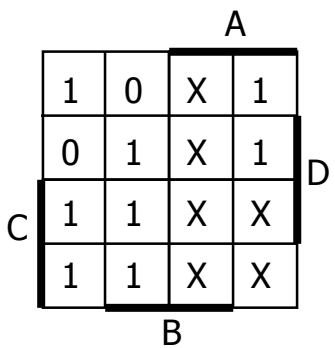
Formalizar o problema

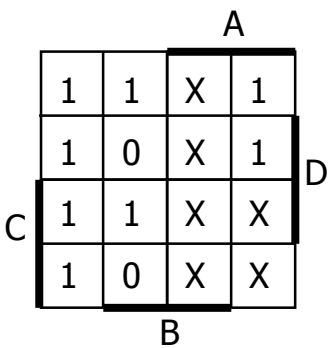
- Tabela verdade
  - Mostra don't cares
- Escolha do alvo da implementação
  - se ROM, nós fazeremos
  - don't cares implica em PAL/PLA, pode ser atrativo
- Siga o procedimento de implementação
  - Minimização usando K-maps

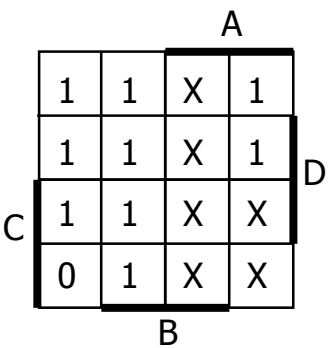
A	B	C	D	C0	C1	C2	C3	C4	C5	C6
0	0	0	0	1	1	1	1	1	1	0
0	0	0	1	0	1	1	0	0	0	0
0	0	1	0	1	1	0	1	1	0	1
0	0	1	1	1	1	1	1	0	0	1
0	1	0	0	0	1	1	0	0	1	1
0	1	0	1	1	0	1	1	0	1	1
0	1	1	0	1	0	1	1	1	1	1
0	1	1	1	1	1	1	0	0	0	0
1	0	0	0	1	1	1	1	1	1	1
1	0	0	1	1	1	1	0	0	1	1
1	0	1	–	–	–	–	–	–	–	–
1	1	–	–	–	–	–	–	–	–	–

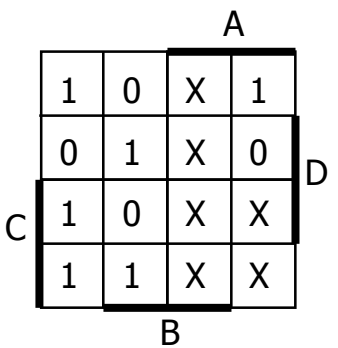
Implementação com soma de produtos minimizado

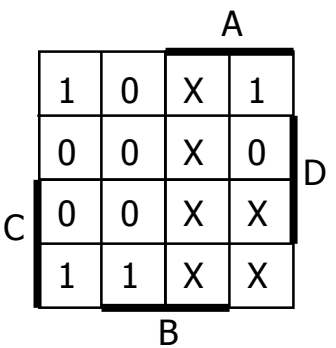
- 15 termos de produtos exclusivos quando minimizado individualmente

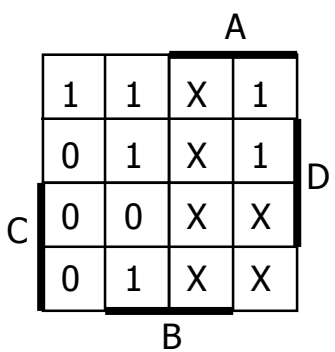


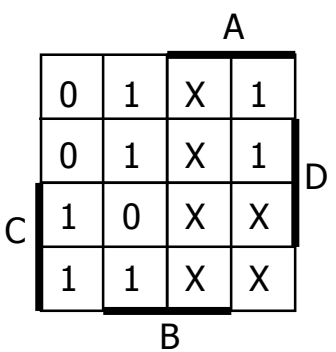








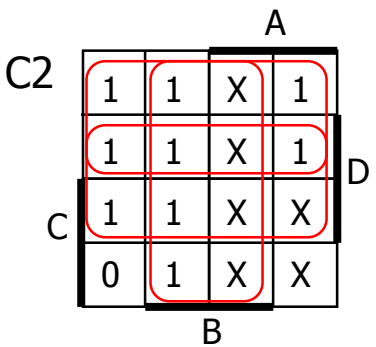




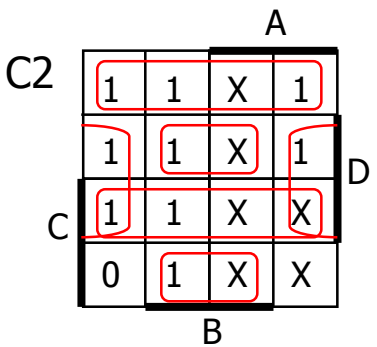
C0 = A + B D + C + B' D'
C1 = C' D' + C D + B'
C2 = B + C' + D
C3 = B' D' + C D' + B C' D + B' C
C4 = B' D' + C D'
C5 = A + C' D' + B D' + B C'
C6 = A + C D' + B C' + B' C

Implementação com S-o-P minimizado

- Pode fazer melhor
  - 9 termos de produtos exclusivos (em vez de 15)
  - Compartilhar termos entre saídas
  - Cada saída não está necessariamente na forma minimizada

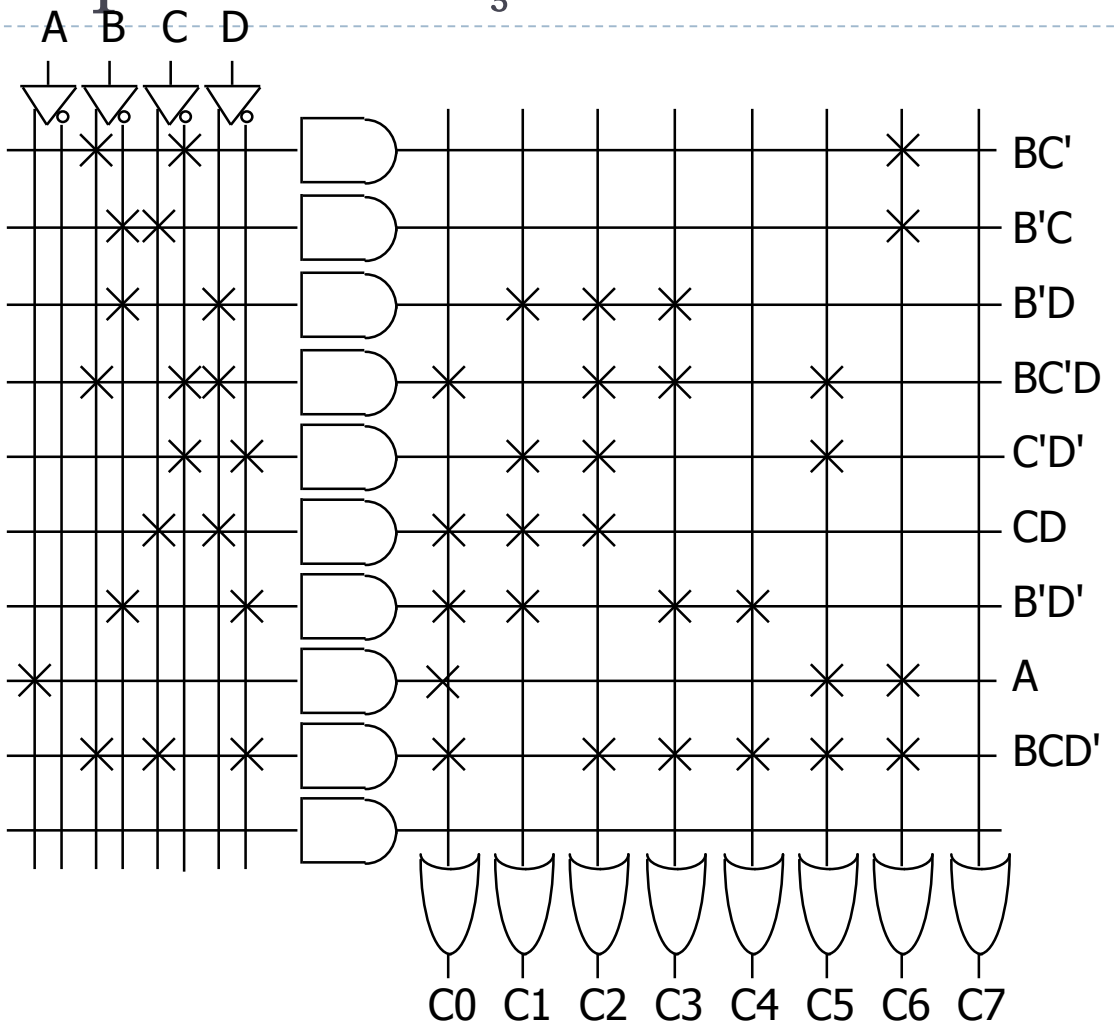


C0 = A + B D + C + B' D'
C1 = C' D' + C D + B'
C2 = B + C' + D
C3 = B' D' + C D' + B C' D + B' C
C4 = B' D' + C D'
C5 = A + C' D' + B D' + B C'
C6 = A + C D' + B C' + B' C



C0 = B C' D + C D + B' D' + B C D' + A
C1 = B' D + C' D' + C D + B' D'
C2 = B' D + B C' D + C' D' + C D + B C D'
C3 = B C' D + B' D + B' D' + B C D'
C4 = B' D' + B C D'
C5 = B C' D + C' D' + A + B C D'
C6 = B' C + B C' + B C D' + A

Implementação PLA



Implementação
PAL vs. porta discreta

- Limite de 4 termos de produto por saída
  - Decomposição de funções com maior número de termos
  - Não compartilham termos em PAL de qualquer maneira (embora existam alguns com termos comuns)

    C2 = B + C' + D

    C2 = B' D + B C' D + C' D' + C D + B C D'

    C2 = B' D + B C' D + C' D' + W ← Necessário outra entrada e outra saída
    W = C D + B C D'

- Decomposição em lógica multi níveis (suporte CAD)
  - Encontrar subexpressões comuns entre funções

    C0 = C3 + A' B X' + A D Y
    C1 = Y + A' C5' + C' D' C6
    C2 = C5 + A' B' D + A' C D
    C3 = C4 + B D C5 + A' B' X'
    C4 = D' Y + A' C D'
    C5 = C' C4 + A Y + A' B X
    C6 = A C4 + C C5 + C4' C5 + A' B' C

    X = C' + D'
    Y = B' C'

Unidade de função lógica

- Bloco de função Multi propósitos
  - 3 entradas de controle para especificar a operação a realizar em operandos
  - 2 entradas de dados para operandos
  - 1 saída do mesmo bit de largura como os operandos

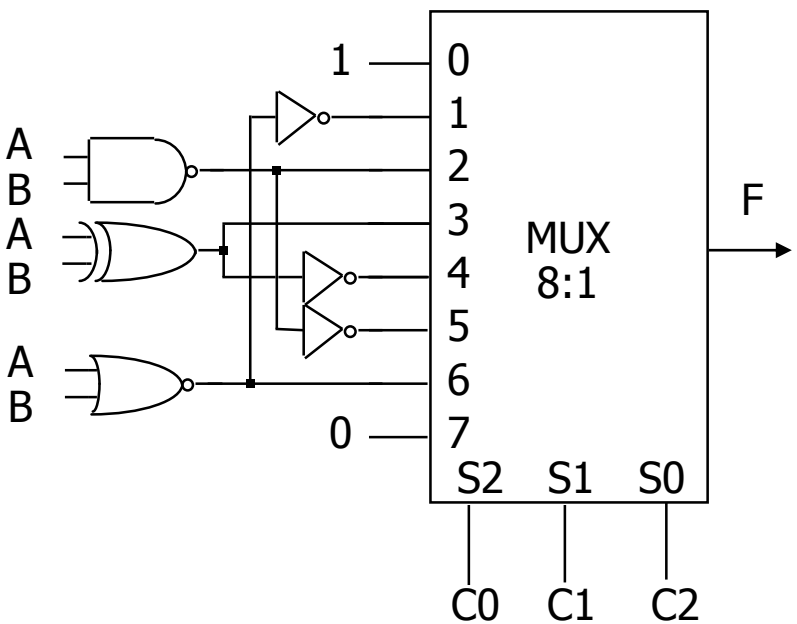
C0	C1	C2	Função	Comentários
0	0	0	1	sempre 1
0	0	1	A + B	lógica OR
0	1	0	(A • B)'	lógica NAND
0	1	1	A xor B	lógca xor
1	0	0	A xnor B	lógica xnor
1	0	1	A • B	lógica AND
1	1	0	(A + B)'	lógica NOR
1	1	1	0	sempre 0

3 entradas de controle: C0, C1, C2
2 entradas de dados: A, B
1 saída: F

Formalizar o problema

C0	C1	C2	A	B	F
0	0	0	0	0	1
0	0	0	0	1	1
0	0	0	1	0	1
0	0	0	1	1	1
0	0	1	0	0	0
0	0	1	0	1	1
0	0	1	1	0	1
0	0	1	1	1	1
0	1	0	0	0	1
0	1	0	0	1	1
0	1	0	1	0	1
0	1	0	1	1	0
0	1	1	0	0	0
0	1	1	0	1	1
0	1	1	1	0	1
0	1	1	1	1	0
1	0	0	0	0	1
1	0	0	0	1	0
1	0	0	1	0	0
1	0	0	1	1	1
1	0	1	0	0	0
1	0	1	0	1	0
1	0	1	1	0	0
1	0	1	1	1	1
1	1	0	0	0	1
1	1	0	0	1	0
1	1	0	1	0	0
1	1	0	1	1	0
1	1	1	0	0	0
1	1	1	0	1	0
1	1	1	1	0	0
1	1	1	1	1	0

Escolher a tecnologia de implementação
5 variáveis K-map para portas discretas
Implementação multiplexador

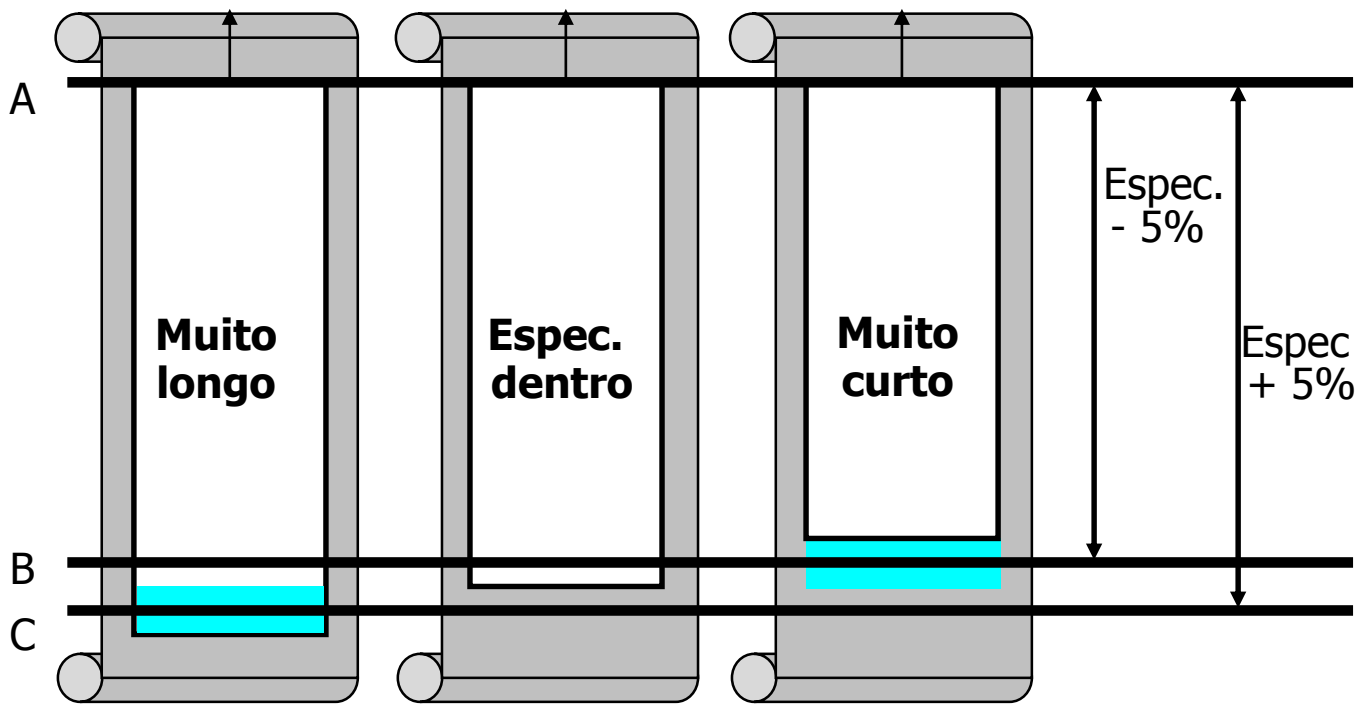


Controle de linha de produção

- Variando o comprimento das hastes (+/-10%). Viajar na correia transportadora
  - Braço mecânico empurra hastes dentro das especificações (+/- 5%) para um lado
  - Segundo braço empurra hastes muito tempo para outro lado
  - Hastes que são curtas ficam na correia
  - 3 barreiras de luz (fonte de luz + fotocélulas) como sensores
  - Projetar lógica combinacional para ativação dos braços
- Compreender o problema
  - Entradas são três sensores
  - Saídas são dois sinais de controle do braço
  - Assumir sensor lê "1" quando desarmado, "0" caso contrário,
  - Chama sensore A, B, C

Esboço do problema

- Posição de sensores
  - Distância A para B = especificação – 5%
  - Distância A para C = especificação + 5%



Formalizar o problema

- Tabela verdade
  - mostra don't cares

A	B	C	Função
0	0	0	faz nada
0	0	1	faz nada
0	1	0	faz nada
0	1	1	faz nada
1	0	0	muito curto
1	0	1	don't care
1	1	0	Em espec.
1	1	1	muito longo

implementação lógica, agora simples
usar apenas três 3 portas AND de 3 entradas

"Muito curto" = AB'C'
(somente o primeiro sensor disparado)

"Em especificação" = A B C'
(primeiros dois sensores desarmados)

"Muito longo" = A B C
(todos os três sensores desarmados)

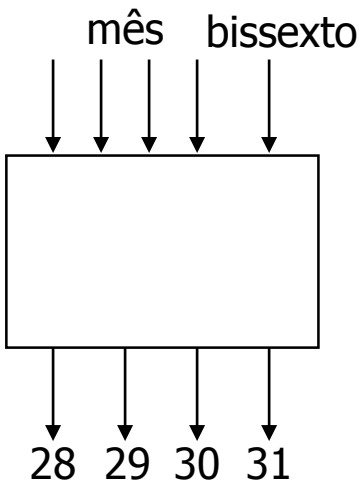
Subsistema Calendário

- Determinar o número de dias em um mês (para controlar o display do relógio)
  - Usado no controle do display de um screen LCD de relógio de pulso
  - Entradas: mês, flag ano bissexto
  - Saídas: número de dias
- Uso de implementação software para ajudar a entender o problema

```
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  4: return (30);
          case  5: return (31);
          case  6: return (30);
          case  7: return (31);
          case  8: return (31);
          case  9: return (30);
          case 10: return (31);
          case 11: return (30);
          case 12: return (31);
          default: return (0);
     }
}
```

Formalizar o problema

- Codificação:
  - Número binário para mês: 4 bits
  - 4 fios para 28, 29, 30, e 31 one-hot – Somente 1 verdadeiro em qualquer tempo
- Diagrama de bloco:



Mês	bissexto	28	29	30	31
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–	–	–	–	–	–

Escolher a implementação alvo e realizar o mapeamento

- Portas discretas
  - 28 = m8′ m4′ m2 m1′ bissexto′
  - 29 = m8′ m4′ m2 m1′ bissexto
  - 30 = m8′ m4 m1′ + m8 m1
  - 31 = m8′ m1 + m8 m1′
- Pode se traduzir para SoP ou PoS

Mês	bissexto	28	29	30	31
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–	–	–	–	–	–

Flag ano bissexto

- Determinar o valor da flag ano bissexto de acordo com o respectivo ano
  - Para anos depois de 1582 (reformulação calendário Gregoriano),
  - Anos bissextos são todos os anos divisíveis por 4,
  - Exceto os anos divisíveis por 100, não são anos bissextos,
  - Mas os anos divisíveis por 400 são anos bissextos.
- Codificação ano:
  - Binário – fácil para anos divisíveis por 4, mas difícil para 100 e 400 (não para potências de 2)
  - BCD – fácil para 100, mais difícil para 4, sobre 400?
- Partes:
  - Construir um circuito que determina se o ano é divisível por 4
  - Construir um circuito que determina se o ano é divisível por 100
  - Construir um circuito que determina se o ano é divisível por 400
  - Combinar os resultados das três etapas anteriores para se obter a flag do ano bissexto

Atividade: Circuito divisível por 4

Circuitos divisíveis por 100 e por 400

- Divisível por 100 requer apenas verificar se todos os dois dígitos de ordem baixa são todos 0:

  YT8’ YT4’ YT2’ YT1’ • YO8’ YO4’ YO2’ YO1’

- Divisível por 400 combina com os circuitos divisível por 4 (aplicado tanto nos dígitos do milhar e centena) e divisível por 100.

  (YM1’ YH2’ YH1’ + YM1 YH2 YH1’)

  (YT8’ YT4’ YT2’ YT1’ • YO8’ YO4’ YO2’ YO1’)

Combinando para determinar flag ano bissexto

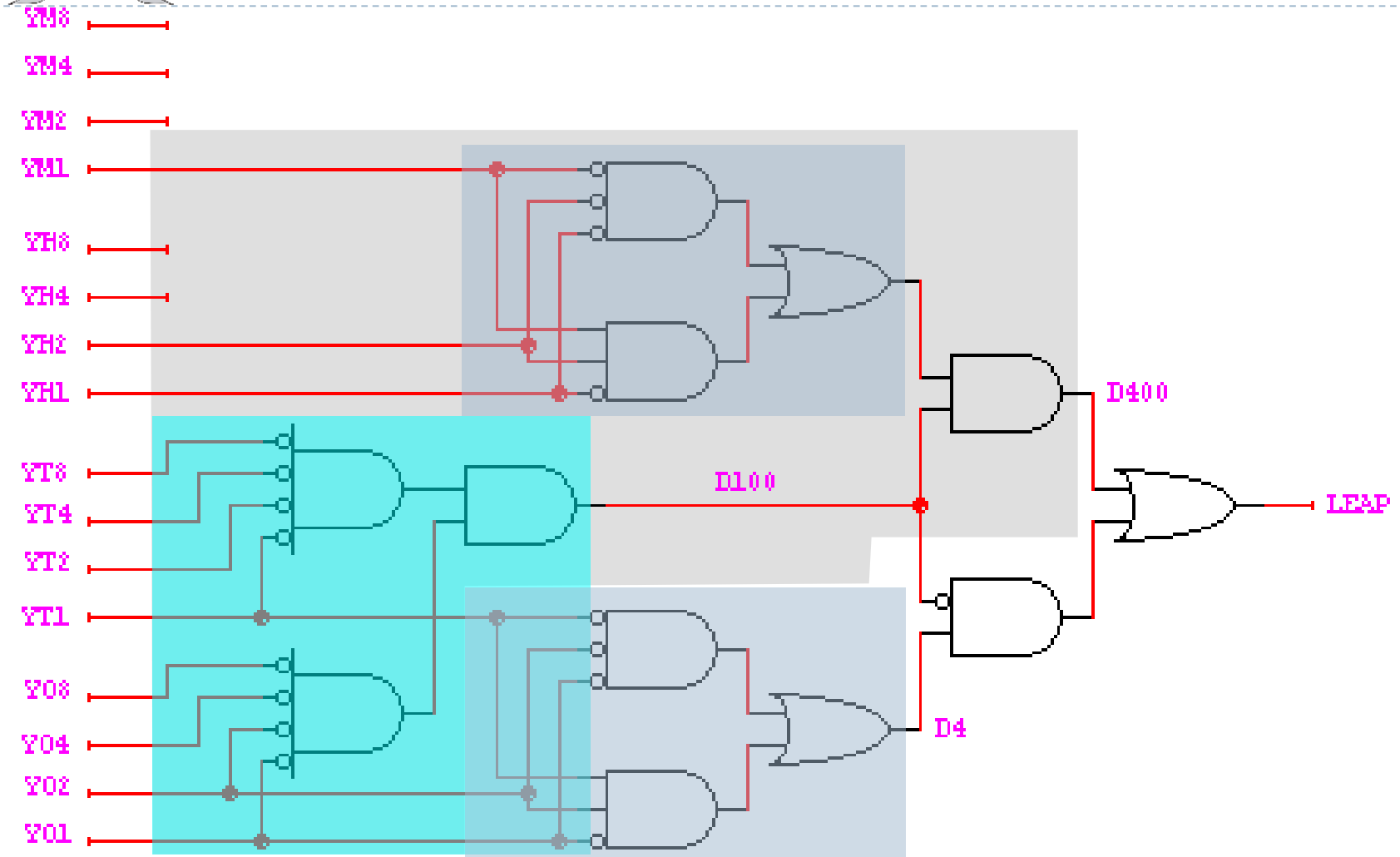
- Rótulo resultados dos três circuitos anteriores: D4, D100, e D400

leap_year_flag = D4 (D100 • D400') '

= D4 • D100' + D4 • D400

= D4 • D100' + D400

Implementação da flag ano bissexto

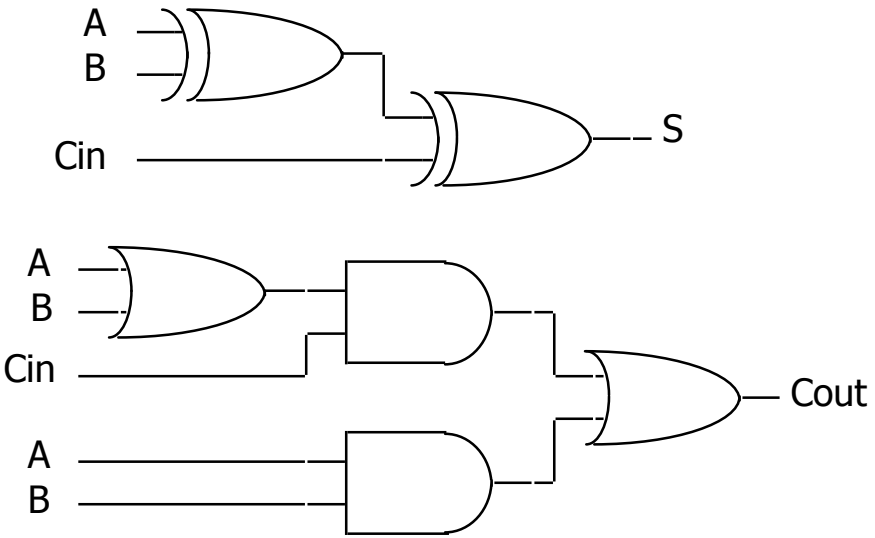


Circuitos aritméticos

- Excelentes exemplos de projeto de lógica combinacional
- Compromisso entre velocidade e tamanho
  - Fazer coisas mais rápido pode exigir mais lógica e, portanto, mais espaço
  - Exemplo: Lógica *carry lookahead*
- Unidades aritméticas e lógicas
  - Blocos construídos como propósito geral
  - Componentes críticos do caminho de dados do processador
  - Usado dentro da maioria das instruções do computador

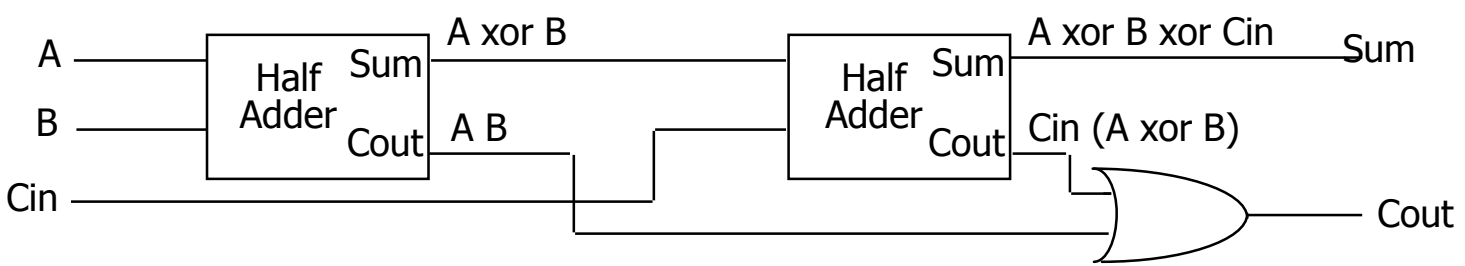
Implementação do somador completo

- Abordagem padrão
  - 6 portas
  - 2 XORs, 2 ANDs, 2 ORs



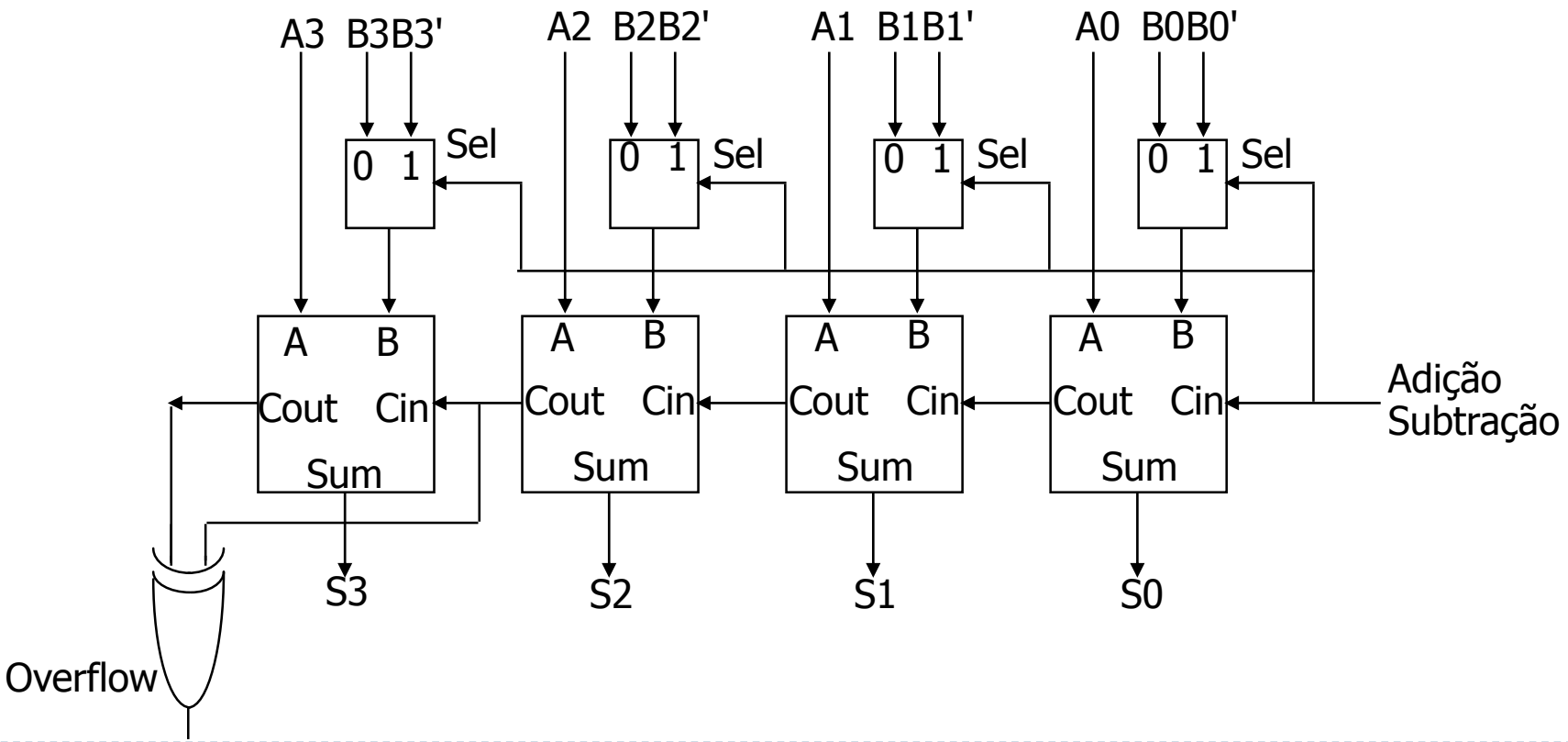
- Alternativa de implementação
  - 5 portas
  - Meio somador é uma porta XOR e uma porta AND
  - 2 XORs, 2 ANDs, 1 OR

Cout = A B + Cin (A xor B) = A B + B Cin + A Cin



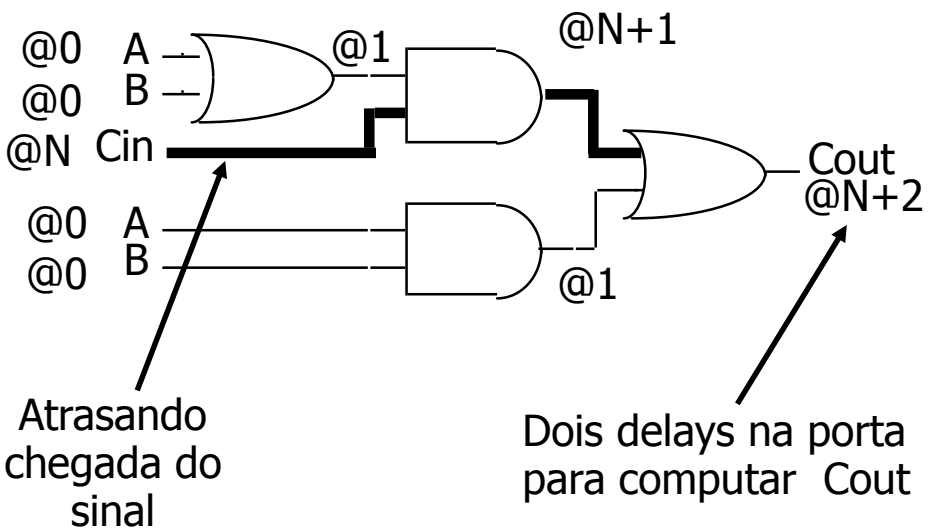
Somador/subtrador

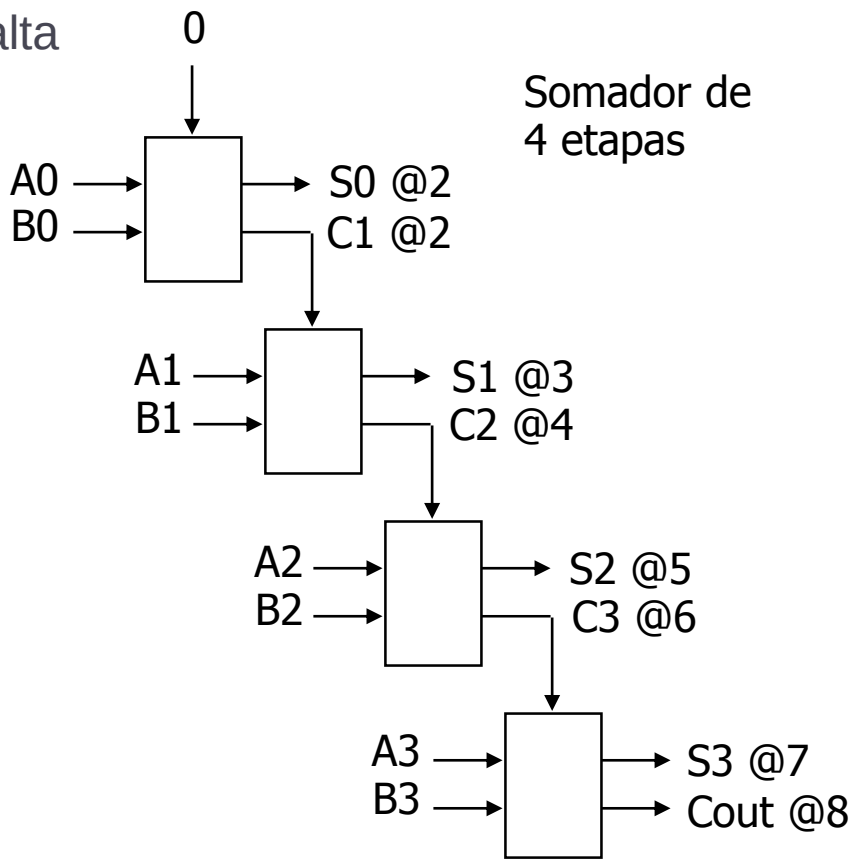
- Usa um somador para fazer uma subtração graças a representação do complemento de 2
  - A – B = A + (– B) = A + B' + 1
  - Sinal de controle seleciona B ou o complemento de 2 de B



Somador ripple-carry

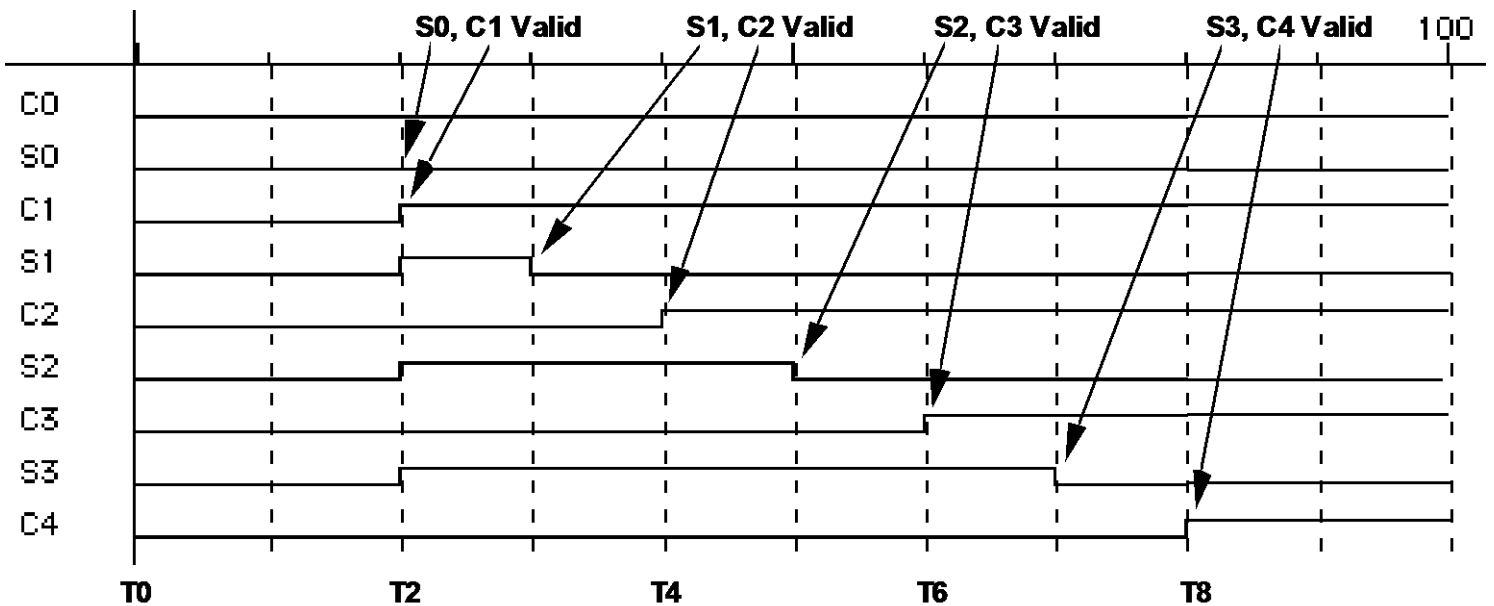
- Atraso crítico
  - A propagação de carry de baixa para alta em ordem de etapas





Somador ripple-carry

- Atraso crítico
  - A propagação de carry de baixa para alta em ordem de etapas
  - 1111 + 0001 é o pior caso na adição
  - carry precisa propagar através de todos os bits



Lógica carry-lookahead

- Geração do carry: Gi = Ai Bi
  - Precisa gera carry quando A = B = 1
- Propagação do carry: Pi = Ai xor Bi
  - carry-in será igual a carry-out aqui
- Sum e Cout podem ser re-expressos em termos da geração/propagação:

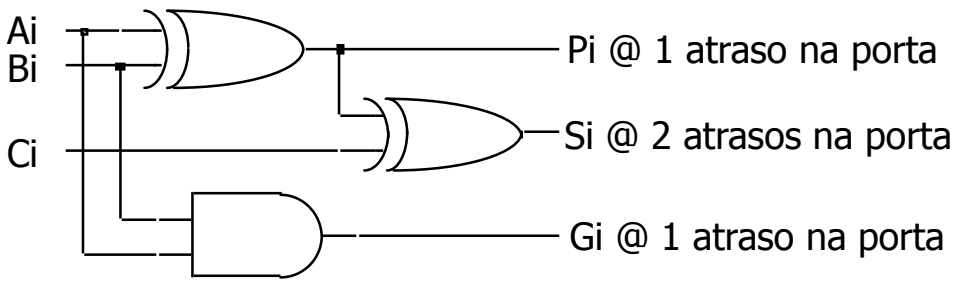
  - Si = Ai xor Bi xor Ci
    = Pi xor Ci
  - Ci+1 = Ai Bi + Ai Ci + Bi Ci
    = Ai Bi + Ci (Ai + Bi)
    = Ai Bi + Ci (Ai xor Bi)
    = Gi + Ci Pi

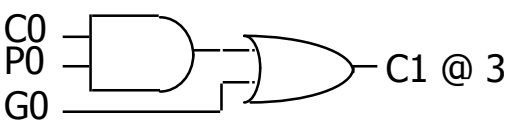
Lógica carry-lookahead

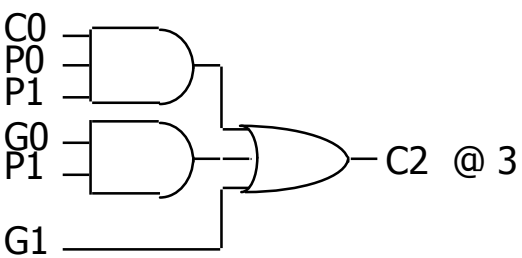
- Re-expressar a lógica carry da seguinte maneira:
  - C1 = G0 + P0 C0
  - C2 = G1 + P1 C1 = G1 + P1 G0 + P1 P0 C0
  - C3 = G2 + P2 C2 = G2 + P2 G1 + P2 P1 G0 + P2 P1 P0 C0
  - C4 = G3 + P3 C3 = G3 + P3 G2 + P3 P2 G1 + P3 P2 P1 G0 + P3 P2 P1 P0 C0
- Cada uma das equações carry pode ser implementada com lógica de dois níveis
  - Todas as entradas são agora diretamente derivadas de entradas de dados e não a partir de carry intermediários
  - Isto permite computar todas as saídas da soma para um procedimento em paralelo

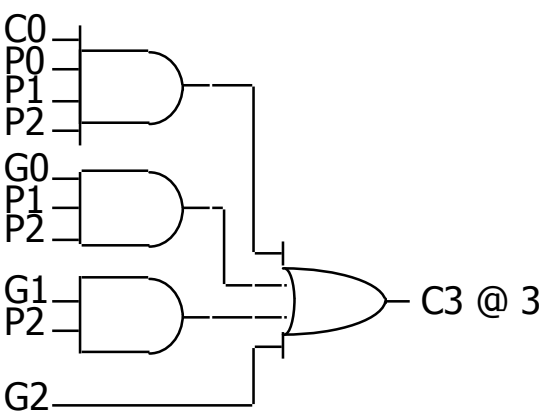
Implementação carry-lookahead

- Somador com saídas propagação e geração

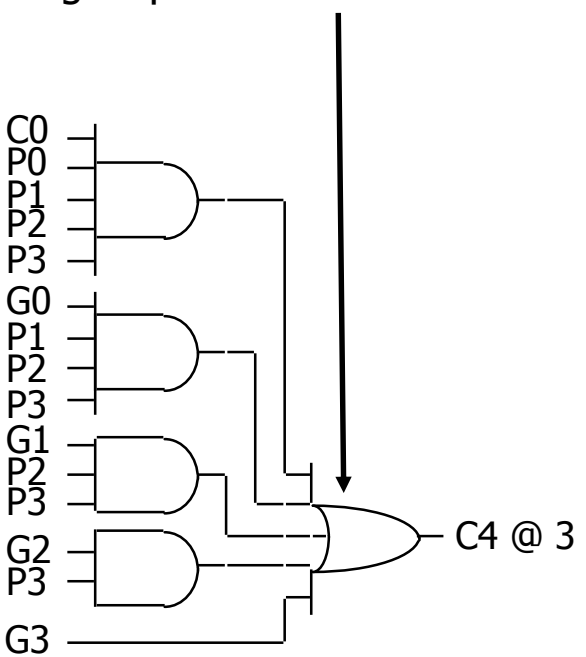






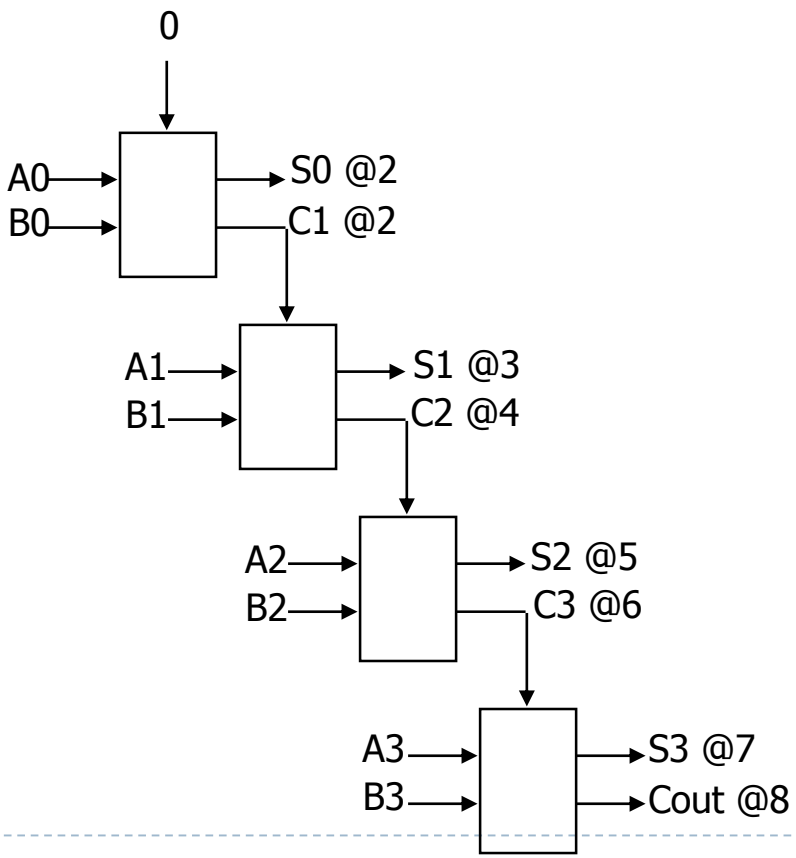


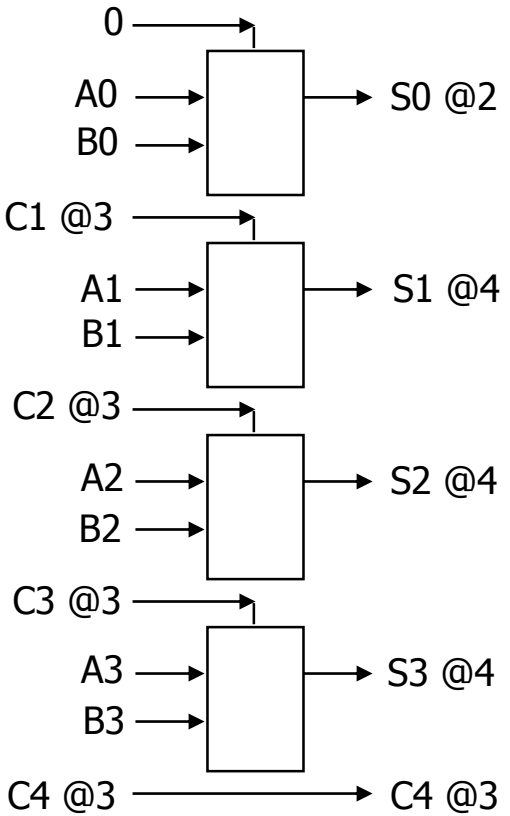
Cada vez mais complexo lógica para carries



Implementação carry-lookahead

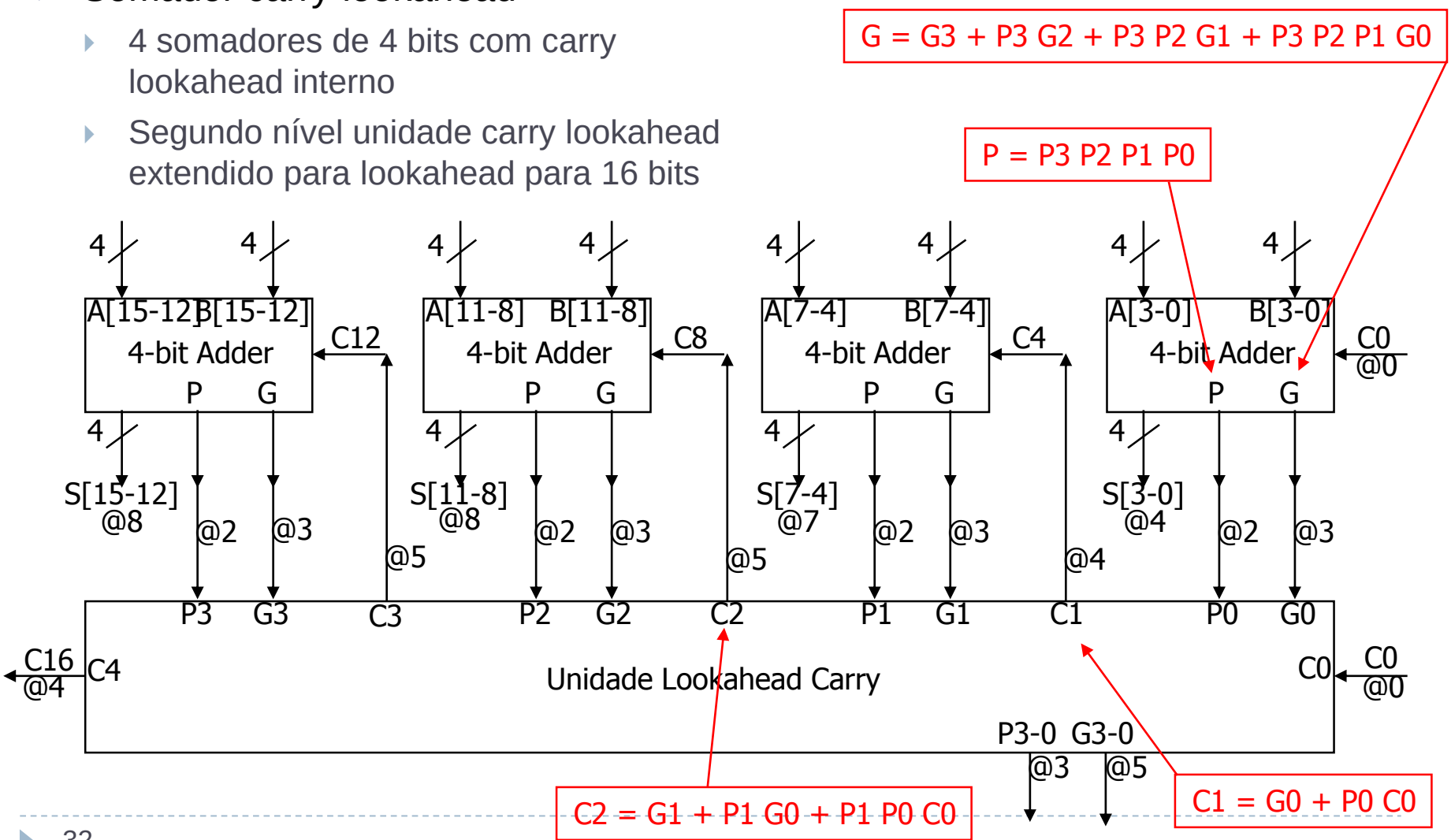
- Lógica carry-lookahead gera carries individuais
  - Somas computadas muito mais rapidamente em paralelo
  - No entanto, o custo da lógica carry aumenta com mais estágios





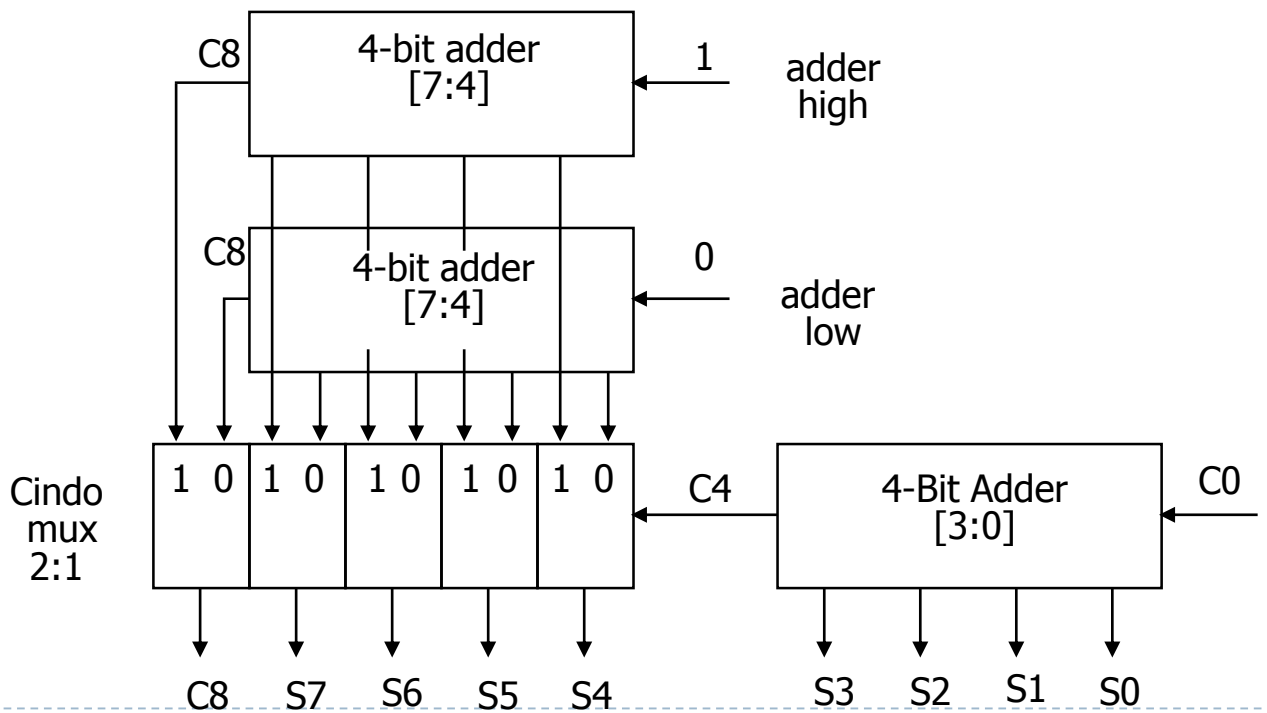
Somador carry-lookahead com lógica carry-lookahead cascateada

- Somador carry-lookahead
  - 4 somadores de 4 bits com carry lookahead interno
  - Segundo nível unidade carry lookahead extendido para lookahead para 16 bits



Somador carry-select

- Hardware redundante para fazer cálculo carry ir mais rápido
  - Computar duas somas de alta ordem em paralelo enquanto espera por carry-in
  - Assumindo um carry-in de 0 e outro carry-in de 1
  - Selecionar o resultado correto apenas quando carry-in é finalmente calculado



Especificação projeto unidade lógica aritmética

M = 0, Operações bitwise lógicos

S1	S0	Função	Comentários
0	0	Fi = Ai	entrada Ai transferida para saída
0	1	Fi = not Ai	complemento de Ai transferida para saída
1	0	Fi = Ai xor Bi	computação XOR de Ai, Bi
1	1	Fi = Ai xnor Bi	computação XNOR de Ai, Bi

M = 1, C0 = 0, operações aritméticas

0	0	F = A	entrada A passada para saída
0	1	F = not A	complemento de A passado para saída
1	0	F = A mais B	soma de A e B
1	1	F = (not A) mais B	soma de B e complemento de A

M = 1, C0 = 1, operações aritméticas

0	0	F = A mais 1	incrementa A
0	1	F = (not A) mais 1	complemento de dois de A
1	0	F = A mais B mais 1	incrementa soma de A e B
1	1	F = (not A) mais B mais 1	B menos A

Operações lógicas e aritméticas
Nem todas as operações parecem úteis, mas "fall out" de lógica interna

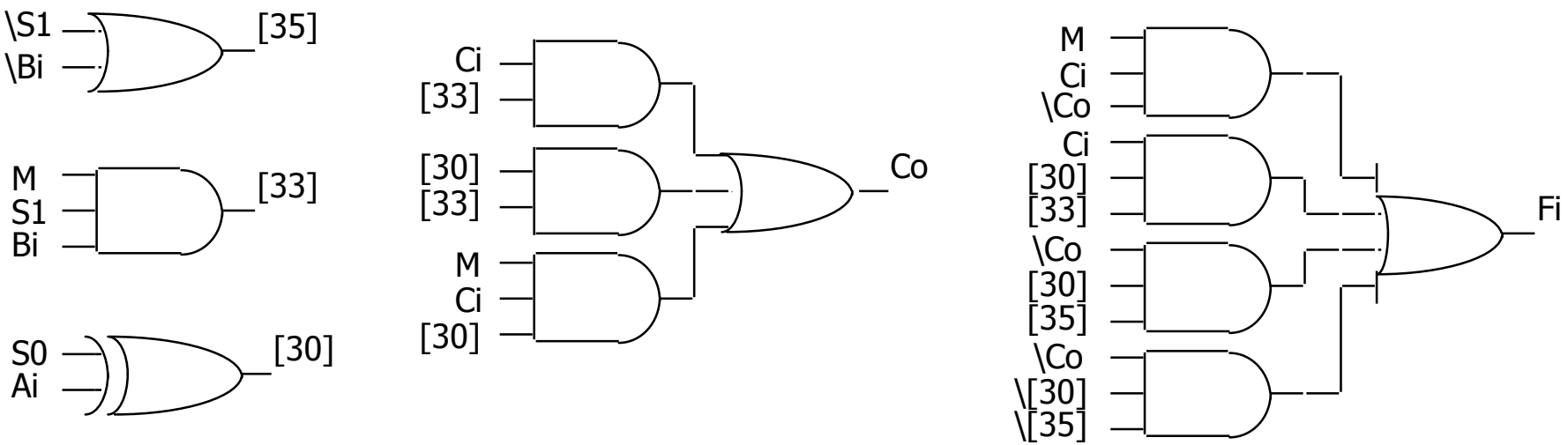
Projeto unidade lógica aritmética

- Amostra ALU – Tabela verdade

M	S1	S0	Ci	Ai	Bi	Fi	Ci+1
0	0	0	X	0	X	0	X
			X	1	X	1	X
	0	1	X	0	X	1	X
			X	1	X	0	X
	1	0	X	0	0	0	X
			X	0	1	1	X
			X	1	0	1	X
			X	1	1	0	X
	1	1	X	0	0	1	X
			X	0	1	0	X
			X	1	0	0	X
			X	1	1	1	X
1	0	0	0	0	X	0	X
			0	1	X	1	X
	0	1	0	0	X	1	X
			0	1	X	0	X
	1	0	0	0	0	0	0
			0	0	1	1	0
			0	1	0	1	0
			0	1	1	0	1
	1	1	0	0	0	1	0
			0	0	1	0	1
			0	1	0	0	0
			0	1	1	1	0
1	0	0	1	0	X	1	0
			1	1	X	0	1
	0	1	1	0	X	0	1
			1	1	X	1	0
	1	0	1	0	0	1	0
			1	0	1	0	1
			1	1	0	0	1
			1	1	1	1	1
	1	1	1	0	0	0	1
			1	0	1	1	1
			1	1	0	1	0
			1	1	1	0	1

Projeto unidade lógica aritmética

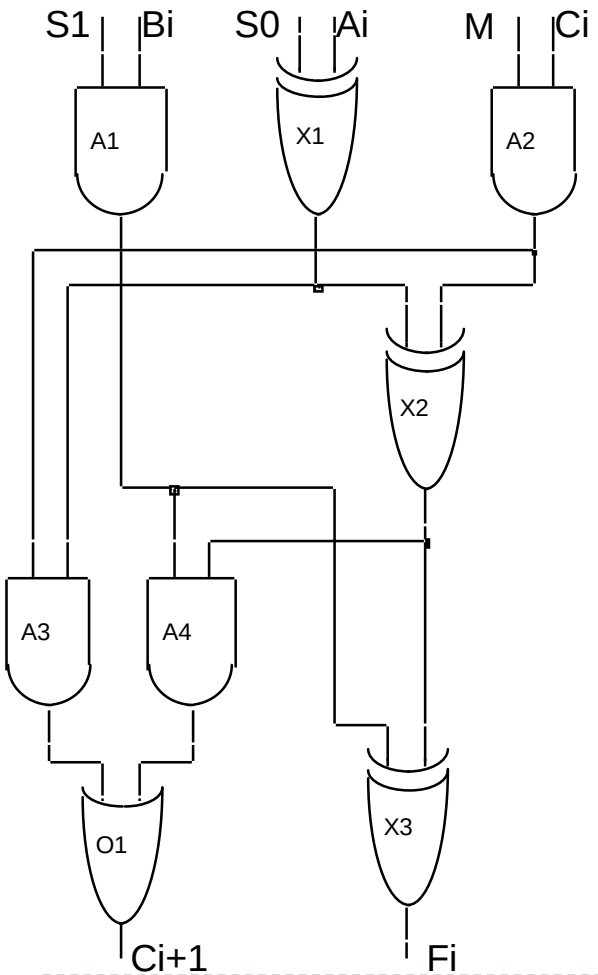
- Amostra ALU – implementação porta lógica multi nível discreta



12 portas

Projeto unidade lógica aritmética

- Amostra ALU – implementação multi nível inteligente



Primeiro nível de portas
  Usa S0 para complementar Ai
    S0 = 0 caso a porta X1 passe por Ai
    S0 = 1 caso a porta X1 passe por Ai'
  Usa S1 para bloquear Bi
    S1 = 0 caso a porta A1 faça Bi encaminhar o valor 0
            (Não quer Bi para operações com somente A)
    S1 = 1 caso a porta A1 passe para Bi
  Usa M para bloquear Ci
    M = 0 caso a porta A2 faça Ci encaminhar o valor 0
           (Não quer Ci para operações lógicas)
    M = 1 caso a porta A2 passe para Ci

Outras portas
  Para M=0 (operações lógicas, Ci é ignorado)
    Fi = S1 Bi xor (S0 xor Ai)
       = S1'S0' (Ai) + S1'S0 (Ai') +
          S1 S0' (Ai Bi' + Ai' Bi) + S1 S0 (Ai' Bi' + Ai Bi)
  Para M=1 (operações aritméticas)
    Fi = S1 Bi xor ((S0 xor Ai) xor Ci) =
    Ci+1 = Ci (S0 xor Ai) + S1 Bi ((S0 xor Ai) xor Ci) =
    Somente um somador completo com entradas S0 xor Ai, S1 Bi, e Ci

Resumo para exemplos de lógica combinacional

- Processo de projeto de lógica combinacional
  - Formalizar problema: codificação, tabela verdade, equações
  - Escolher tecnologias de implementação (ROM, PAL, PLA, portas discreta)
  - Implementar seguindo o procedimento de projeto para que a tecnologia
- Representação número binário
  - Números positivos
  - Diferença na forma como os números negativos são representados
  - Complemento de 2 mais fácil de manusear: Uma representação para zero, complementação pouco complicado, simples adição
- Circuitos para adição binária
  - Básico half-adder (meio somador) e full-adder (somador completo)
  - Lógica carry lookahead
  - Carry-select
- Projeto ALU
  - Especificação, implementação