**FACULDADE DE ENGENHARIA ELÉTRICA E DE COMPUTAÇÃO**
**EA772 - CIRCUITOS LÓGICOS**


Ana Carolina De Almeida Cardoso
Pedro Damasceno Vasconcellos

Projetos finais da disciplina

CAMPINAS
2023

## 01. MEMÓRIA TIPO READ-ONLY (ROM)

O primeiro elemento do circuito que construímos foi a estrutura do bloco de memória. Note que, conforme pedido pelo professor no enunciado do projeto, projetamos um bloco de memória de 4 linhas, onde cada linha (ou instrução) possui 4 bits. Foi dado que, para a seleção do bloco de memória, haveria duas chaves físicas, e assim, para projetarmos um seletor de blocos, construímos um decodificador, onde, para n entradas, há $2^n$ saídas. Dessa maneira, como temos 2 entradas, então há $2^2 = 4$ saídas.

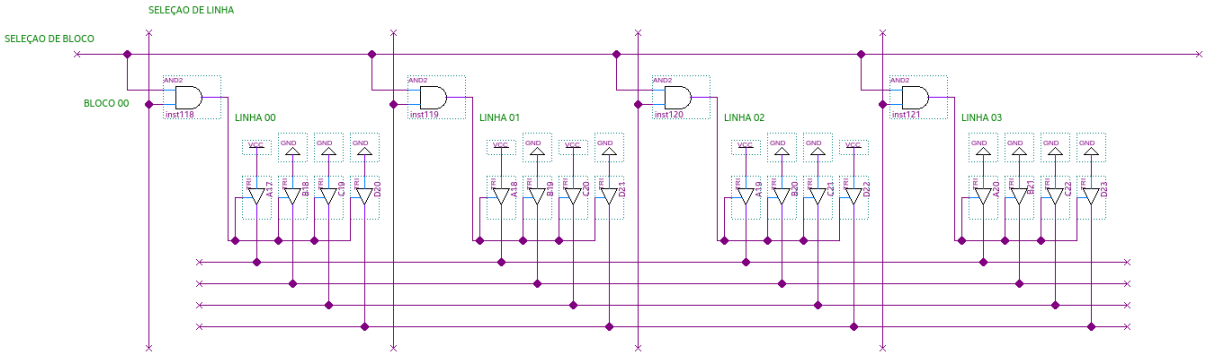


**Imagem 01** – Um dos blocos de memória do nosso circuito.

Veja que, para atendermos ao modelo proposto pelo professor, foi necessário projetarmos um total de quatro bancos de memória, todos iguais ao modelo acima. Embora tenha sido dito que apenas o primeiro bloco precisava ter as instruções de memória idênticas às da primeira página da descrição do projeto, optamos por deixar todos os demais blocos com instruções idênticas. Note que as linhas do bloco estão em ordem crescente, indo da linha 0 até a linha 3.

Para selecionarmos qual linha iríamos acessar, utilizamos um contador de 4 estados, onde cada um ativa a sua respectiva linha. Decidimos, então, construir um contador assíncrono feito a partir de FF-JK, que pode ser visto na imagem abaixo. Além disso, como a ativação do clock se dá por um botão, não há como a escala de tempo tornar um contador síncrono superior ao contador assíncrono. Utilizamos a tabela-verdade abaixo como referência para projetarmos este contador:

| NÚMERO | $Q_n$ | $Q_{n+1}$ | $J_1K_1$ | $J_0K_0$ |
|---|---|---|---|---|
| 0 | 00 | 01 | 0X | 1X |
| 1 | 01 | 10 | 1X | X1 |
| 2 | 10 | 11 | X0 | 1X |
| 3 | 11 | 00 | X1 | X1 |

**Tabela 01** – Tabela-verdade referente a um flip-flop do tipo JK.

Com isso, a partir do auxílio dessa tabela, conseguimos de fato projetar o circuito referente ao contador, que pode ser observado melhor na imagem a seguir:

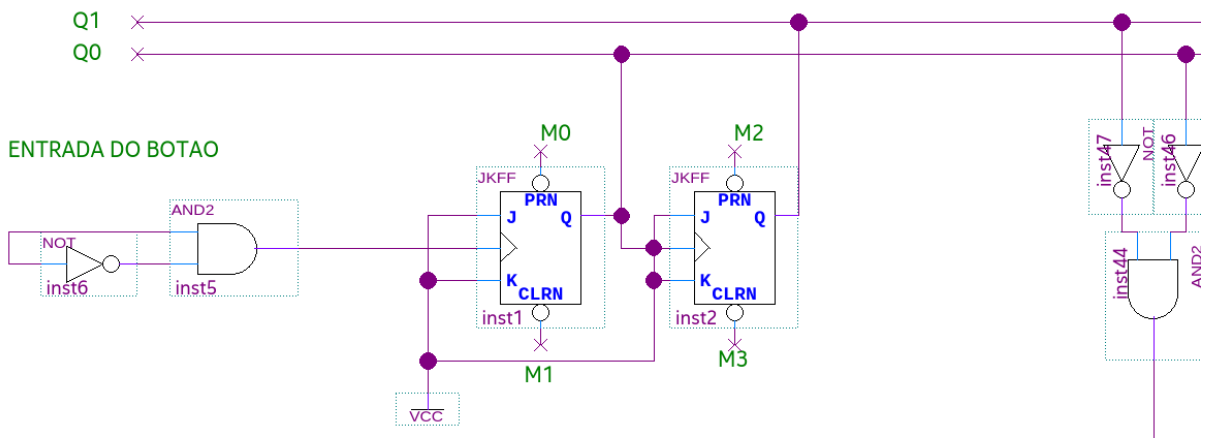


**Imagem 02** – Representação do contador assíncrono projetado para seleção de linhas.

Para utilizarmos o clock para acessar a sua respectiva linha de informação, uma estrutura de decodificador foi utilizada. Com isso, cada um dos valores do clock, de 00 a 11, ativa apenas a sua respectiva linha. Para o caso em que fosse necessário uma seleção manual de linha, foi admitido que as entradas $M_0$ a $M_3$ seriam utilizadas para selecionar as linhas, seguindo a tabela abaixo. É importante ressaltar que qualquer valor não especificado não é válido para entradas de MX.

| $M_0$ | $M_1$ | $M_2$ | $M_3$ | Linha ativa (binário) |
|---|---|---|---|---|
| 1 | 0 | 1 | 0 | 00 |
| 1 | 0 | 0 | 1 | 01 |
| 0 | 1 | 1 | 0 | 10 |
| 0 | 1 | 0 | 1 | 11 |

**Tabela 02 –** Entradas para selecionarmos as linhas.

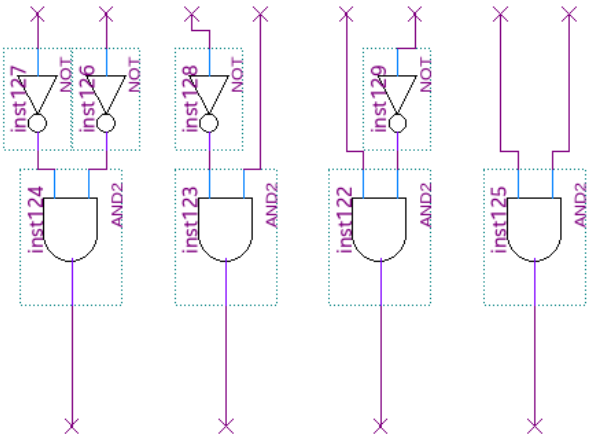


**Imagem 03** – Circuito para acessarmos a linha ou bloco de memória.

A última estrutura utilizada foi um AND, utilizando a ativação de cada linha com a ativação de cada bloco. Dessa maneira, cada instrução seria conectada ao barramento apenas quando o seu bloco e sua linha estivessem ligados, como pode ser visto abaixo. Na figura o condutor que vem da borda esquerda é o referente à ativação do bloco por completo, e o que vem da borda superior é referente a ativação de qualquer uma das primeiras linhas de cada bloco. Portanto, o AND combina

estas informações e apenas ativa uma instrução por vez. Além disso, é importante ressaltar que foi assumido que cada barramento de cada bloco será unido posteriormente, logo não seria necessário criar um curto entre eles neste projeto.

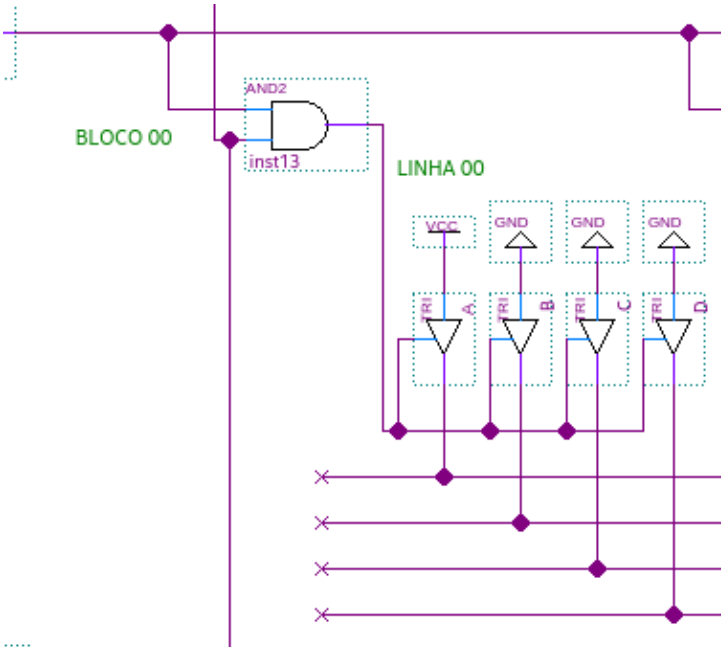


**Imagem 04 –** Estrutura representando nosso bloco.

Juntando todas essas estruturas, pudemos enfim terminar o projeto do circuito, que pode ser analisado mais detalhadamente na imagem abaixo:

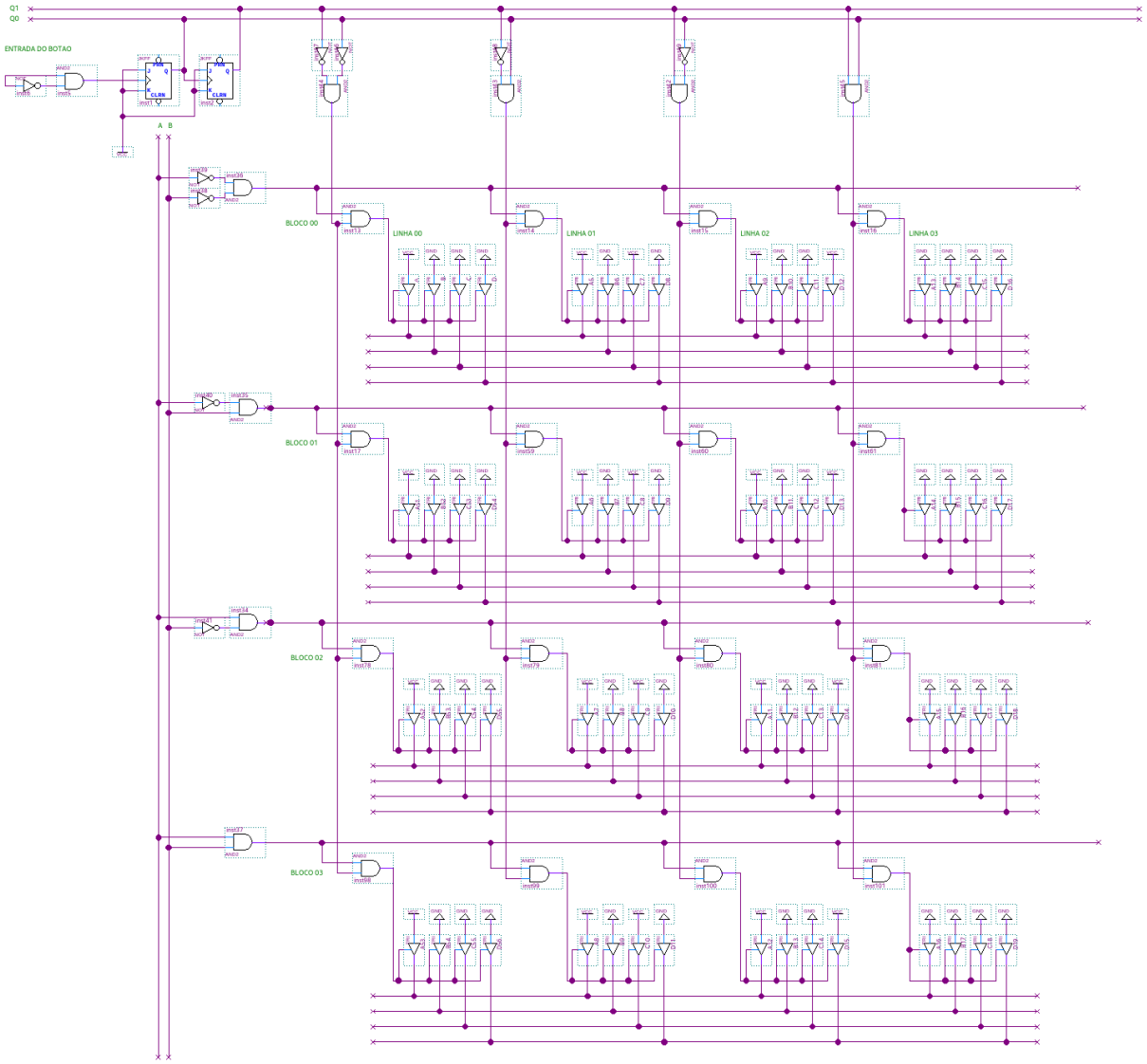


**Imagem 05** – Circuito que representa uma memória do tipo read-only.

## 02. UNIDADE LÓGICA E ARITMÉTICA (ULA)

Inicialmente, projetamos os complementadores para os números x e y. Note que, para atender às instruções passadas pelo professor, consideramos que ambos os números não poderiam ser negativos ao mesmo tempo, visto que foi entendido que os complementadores iriam apenas resultar no complemento do número e, para virar complemento de 2, há apenas uma única porta de Carry-in. Por conta disso, entendemos que não seria possível complementar na base 2 os dois números ao mesmo tempo, pois para isso seriam necessárias duas portas de Carry-in.

No design do circuito que complementa o número podemos observar a tabela abaixo, notando que, caso o complemento esteja ligado, o dígito se transforma em seu oposto e, caso contrário, permanece o mesmo. Assim, atendemos os requisitos para a operação de complemento.

| DÍGITO | COMPLEMENTO | D ⊕ C |
|---|---|---|
| 0 | 0 | 0 |
| 0 | 1 | 1 |

| | | |
|---|---|---|
| 1 | 0 | 1 |
| 1 | 1 | 0 |

**Tabela 03** - Tabela verdade para a operação XOR.

É importante notar que, como havia a possibilidade de zerarmos o número y, tivemos o cuidado adicional de, após complementamos o número, realizarmos, com cada dígito, uma operação do tipo D AND (NOT Zy). Assim, caso houvesse uma entrada de 1 na porta Zy, todos os dígitos seriam zerados. Outro detalhe importante é o fato de que a parte do circuito que zera o número deve ficar após a parte de complemento, porque, caso ela estivesse antes de complementar o número, haveria um caso onde o complemento e o zera estariam ligados e assim a saída seria o número máximo (1111). Por isso, foi necessário haver essa precaução extra, tomando o 'zera' como maior prioridade ao 'complementa'. Com isso, os nossos circuitos para complementar x e para complementar e zerar y tiveram a seguinte construção:

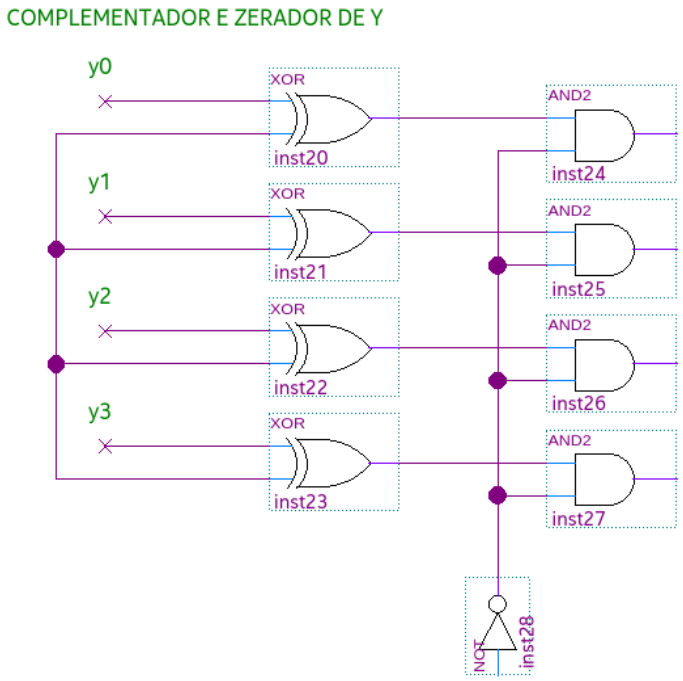


**Imagem 06** – Circuito responsável pela operação complemento e zeramento de y.

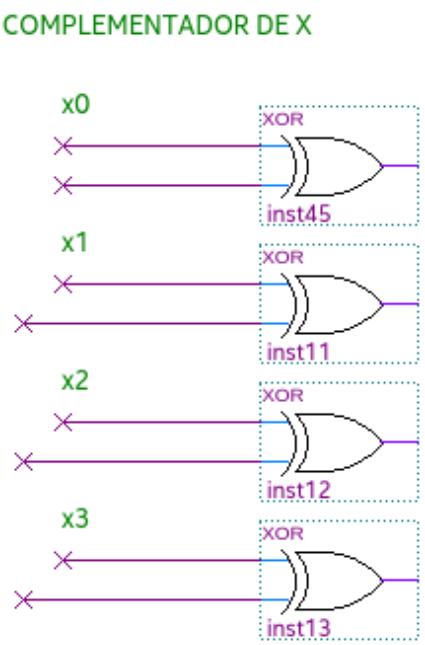


**Imagem 07** - Circuito responsável por complementar x.

Para projetarmos o circuito referente ao somador para cada dígito $x_i$ e $y_i$, inicialmente construímos a seguinte tabela-verdade para utilizarmos como base:

| $X_i$ | $Y_i$ | $C_i$ | $D_i$ | $C_{i+1}$ |
|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 1 | 1 | 0 |
| 0 | 1 | 0 | 1 | 0 |
| 0 | 1 | 1 | 0 | 1 |
| 1 | 0 | 0 | 1 | 0 |
| 1 | 0 | 1 | 0 | 1 |
| 1 | 1 | 0 | 0 | 1 |
| 1 | 1 | 1 | 1 | 1 |

**Tabela 04** – Tabela-verdade para uma soma completa entre dois dígitos.

A partir dessa tabela, foi possível construir dois Mapas de Karnaugh, que podem ser vistos abaixo. Com estes dois mapas, conseguimos extrair as seguintes funções: $D_i = X_i \oplus (Y_i \oplus C_i)$, onde $D_i$ representa a soma dos dígitos $X_i$ e $Y_i$, e $C_{i+1} = X_i \cdot Y_i + X_i \cdot C_i + Y_i \cdot C_i$, onde $C_{i+1}$ representa se haverá um carry para a soma dos dígitos $X_{i+1}$ e $Y_{i+1}$.

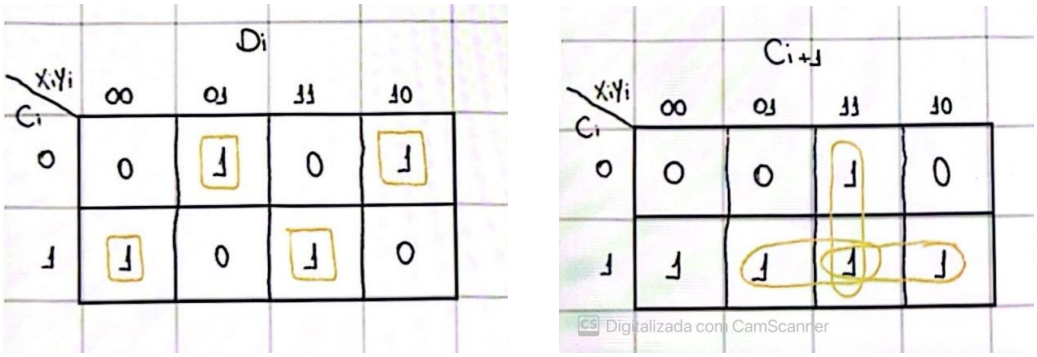


**Imagem 08** – Mapas de Karnaugh auxiliares para a montagem do circuito responsável pela soma.

Com isso, o circuito resultante para a soma dos dígitos assumiu o seguinte formato:

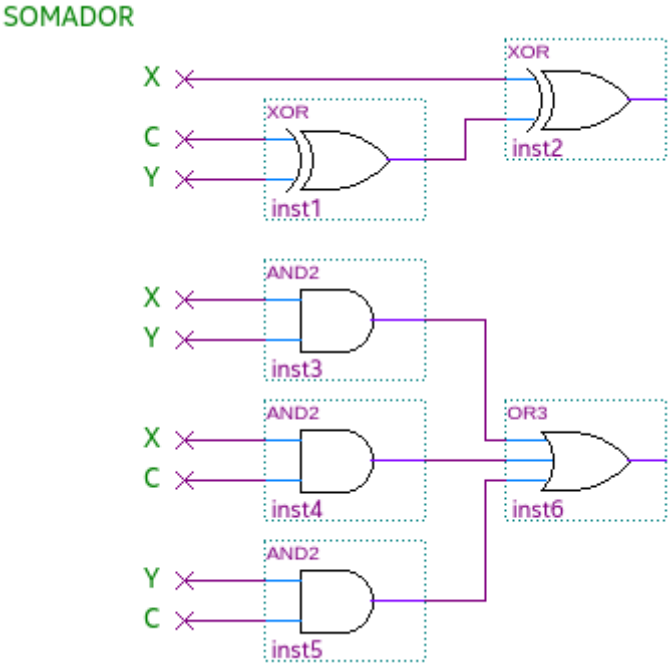


**Imagem 09** – Circuito para a operação de soma completa dos dígitos $X_i$ e $Y_i$.

Para o decodificador, que recebe as entradas f[3…0], decidimos assumir que cada uma das quatro variáveis controla cada uma das quatro saídas. Além disso, também assumimos que as entradas seriam válidas para as possíveis operações do circuito, visto que não é possível realizarmos a operação (–x) + (–y) devido à ausência de uma segunda porta de carry-in e que o circuito "Complementa" era responsável por um complemento simples. Ou seja, a ULA projetada tem a capacidade de lidar apenas com o oposto de uma das entradas por vez

Por fim, sabemos que nosso projeto possui um total de oito saídas. Quatro delas correspondem aos dígitos do número resultante da soma entre x e y. Já as outras quatro saídas dirão se houve overflow, carry e se o número resultante da soma é negativo ou zero. A lógica que construímos para essas quatro saídas está explicada mais detalhadamente abaixo:

- Overflow: checamos se houve carry para os dois dígitos mais significativos $D_3$ e $D_2$. Colocamos as informações como entradas de um XOR e, caso a saída fosse 1, saberíamos que havia ocorrido overflow.
- Negativo: se não houve carry para fora e há uma subtração (ou seja, a entrada do carry é 1) então sabemos que o número obtido é negativo.

- Zero: colocamos um OR com quatro entradas. Dessa forma, se qualquer um dos dígitos for 1, nossa saída será 1. Com o not, invertemos esse resultado. Assim, garantimos que a saída será 1 no caso em que a soma resultar em zero.
- Carry: apenas pegamos o resultado do carry do algarismo mais significativo, $D_3$. Caso essa saída fosse 1, então saberíamos que havia ocorrido carry.
  Explicada a lógica utilizada, o os circuitos resultantes tiveram a seguinte forma:

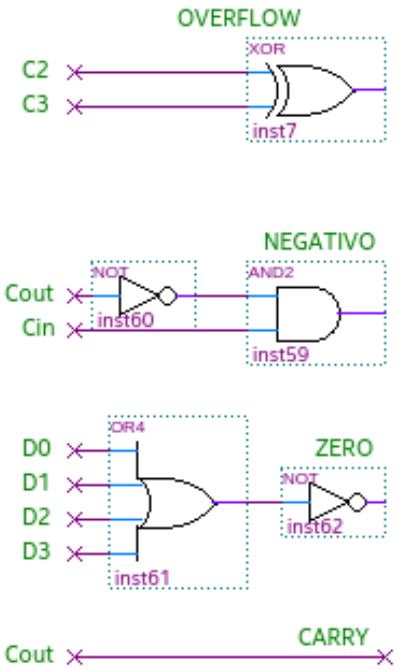


**Imagem 10** – Circuitos representando as saídas O, N, Z e C.

Juntando todos esses quatro blocos lógicos anteriormente explicados, pudemos enfim construir o circuito que simula nossa ULA, que pode ser visto melhor na imagem abaixo. Note que, com o intuito de poluirmos menos a imagem do circuito final, os somadores de $X_i$ e $Y_i$, do complemento de X e do complemento e zeramento de Y foram representados apenas como blocos lógicos com suas respectivas entradas e saídas.

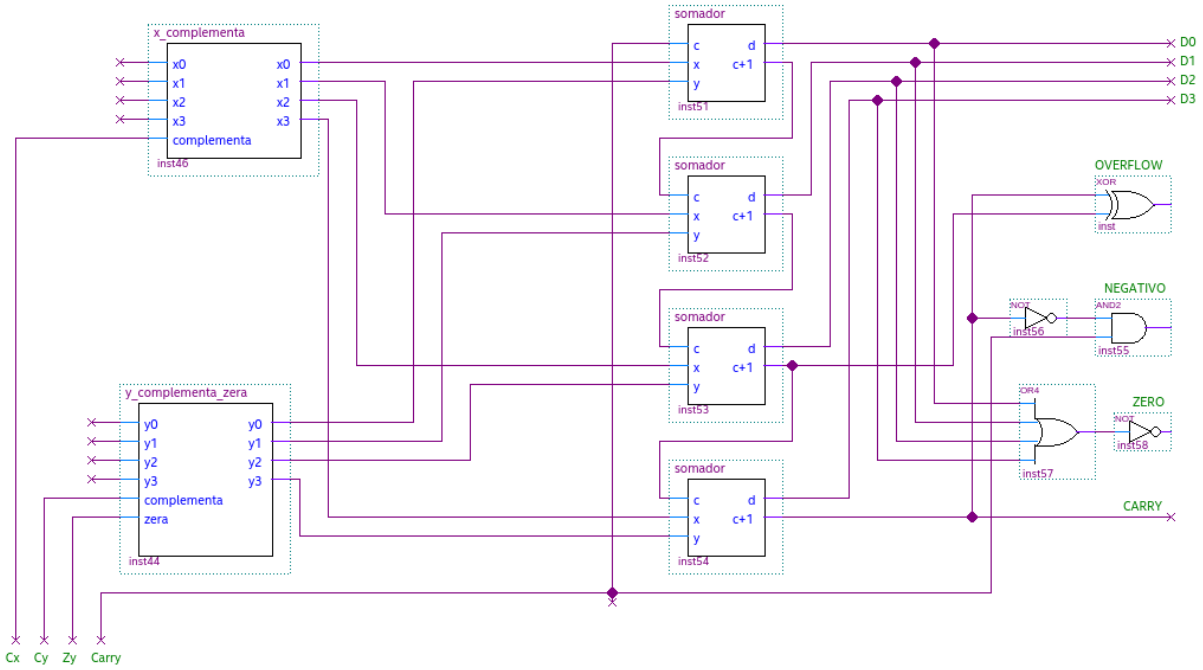


**Imagem 11** – Projeto final do circuito.