UNIVERSIDADE DO VALE DO RIO DOS SINOS

Ciência da Computação

Projetos de Sistemas Digitais

Professor: João Elison

GUILHERME DOS SANTOS MOREIRA

LUIZ ALFREDO TOMAZHINI

São Leopoldo, RS, abril de 2020

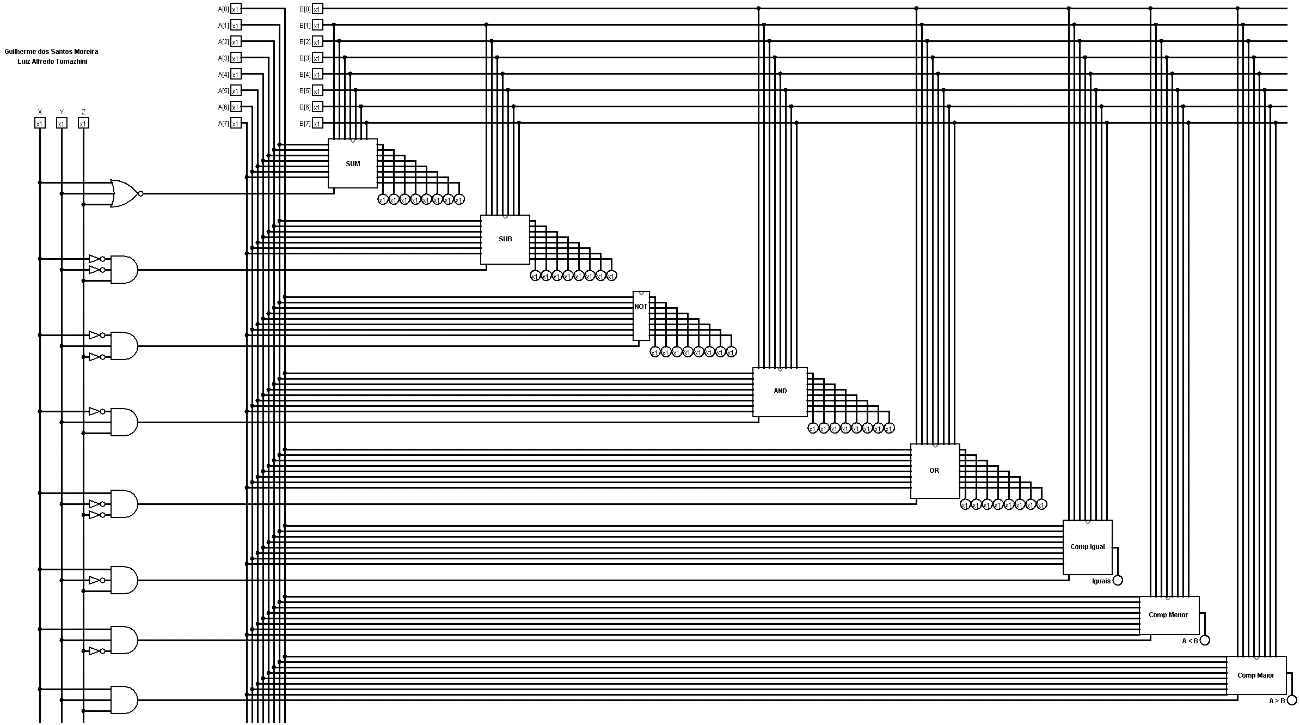
**Unidade Lógica Aritmética**

**Descrição:**

O circuito de Unidade Lógica Aritmética (ULA) é uma calculadora eletrônica capaz de realizar operações lógicas (booleanas) e aritméticas, conversando com o baixo nível de máquina. É uma peça fundamental da Unidade Central de Processamento (UCP) e até de microprocessadores.

Com a evolução dos semicondutores, a criação dos transistores dos circuitos integrados (CI), as ULAs poderam ser implementadas em larga escalas, exponencializando o poder de processamento na computação.

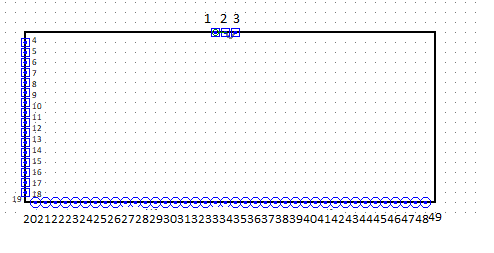
**Circuito Final:**

****

Desenvolvemos o circuito simulado pelo software LogiSim, contendo 1 circuito principal e 8 subcircuitos. Esta plataforma permite modularizar circuitos, permitindo reutilizá-los em outros na forma de Cis.

Neste circuito final, temos 19 entradas e 43 saídas. 3 entradas são utilizadas como chave seletora para escolher qual das 8 operações a ULA irá realizar, 16 entradas servem como duas palavras de 8 bits para que as operações sejam realizadas entre elas. Os circuitos aritméticas e os lógicos de NOT, AND e OR têm 8 saídas cada, já os outros 3 têm apenas 1 saída.

**Pinagem:**

****

**Pinos:**

**1 – Entrada X (combinação da chave seletora)**

**2 – Entrada Y (combinação da chave seletora)**

**3 – Entrada Z (combinação da chave seletora)**

**4 – Entrada A0**

**5 – Entrada B0**

**6 – Entrada A1**

**7 – Entrada B1**

**8 – Entrada A2**

**9 – Entrada B2**

**10 – Entrada A3**

**11 – Entrada B3**

**12 – Entrada A4**

**13 – Entrada B4**

**14 – Entrada A5**

**15 – Entrada B5**

**16 – Entrada A6**

**17 – Entrada B6**

**18 – Entrada A7**

**19 – Entrada B7**

**20 – Saída S0 (saída do circuito somador)**

**21 – Saída S1 (saída do circuito somador)**

**22 – Saída S2 (saída do circuito somador)**

**23 – Saída S3 (saída do circuito somador)**

**24 – Saída S4(saída do circuito somador)**

**25 – Saída S5 (saída do circuito somador)**

**26 – Saída S6 (saída do circuito somador)**

**27 – Saída S7 (saída do circuito somador)**

**28 – Saída S8 (saída do circuito subtrator)**

**29 – Saída S9 (saída do circuito subtrator)**

**30 – Saída S10 (saída do circuito subtrator)**

**31 – Saída S11 (saída do circuito subtrator)**

**32 – Saída S12 (saída do circuito subtrator)**

**33 – Saída S13 (saída do circuito subtrator)**

**34 – Saída S14 (saída do circuito subtrator)**

**35 – Saída S15 (saída do circuito subtrator)**

**36 – Saída S16 (saída do circuito lógico NOT)**

**37 – Saída S17 (saída do circuito lógico NOT)**

**38 – Saída S18 (saída do circuito lógico NOT)**

**39 – Saída S19 (saída do circuito lógico NOT)**

**40 – Saída S20 (saída do circuito lógico NOT)**

**41 – Saída S21 (saída do circuito lógico NOT)**

**42 – Saída S22 (saída do circuito lógico NOT)**

**43 – Saída S23 (saída do circuito lógico NOT)**

**44 – Saída S24 (saída do circuito lógico AND)**

**45 – Saída S25 (saída do circuito lógico AND)**

**46 – Saída S26 (saída do circuito lógico AND)**

**47 – Saída S27 (saída do circuito lógico AND)**

**48 – Saída S28 (saída do circuito lógico AND)**

**49 – Saída S29 (saída do circuito lógico AND)**

**50 – Saída S30 (saída do circuito lógico AND)**

**51 – Saída S31 (saída do circuito lógico AND)**

**52 – Saída S32 (saída do circuito lógico OR)**

**53 – Saída S33 (saída do circuito lógico OR)**

**54 – Saída S34 (saída do circuito lógico OR)**

**55 – Saída S35 (saída do circuito lógico OR)**

**56 – Saída S36 (saída do circuito lógico OR)**

**57 – Saída S37 (saída do circuito lógico OR)**

**58 – Saída S38 (saída do circuito lógico OR)**

**59 – Saída S39 (saída do circuito lógico OR)**

**60 – Saída S40 (saída do circuito lógico Comparador de Igualdade)**

**61 – Saída S41 (saída do circuito lógico Comparador de Menor Que)**

**62 – Saída S42 (saída do circuito lógico Comparador de Maior Que)**

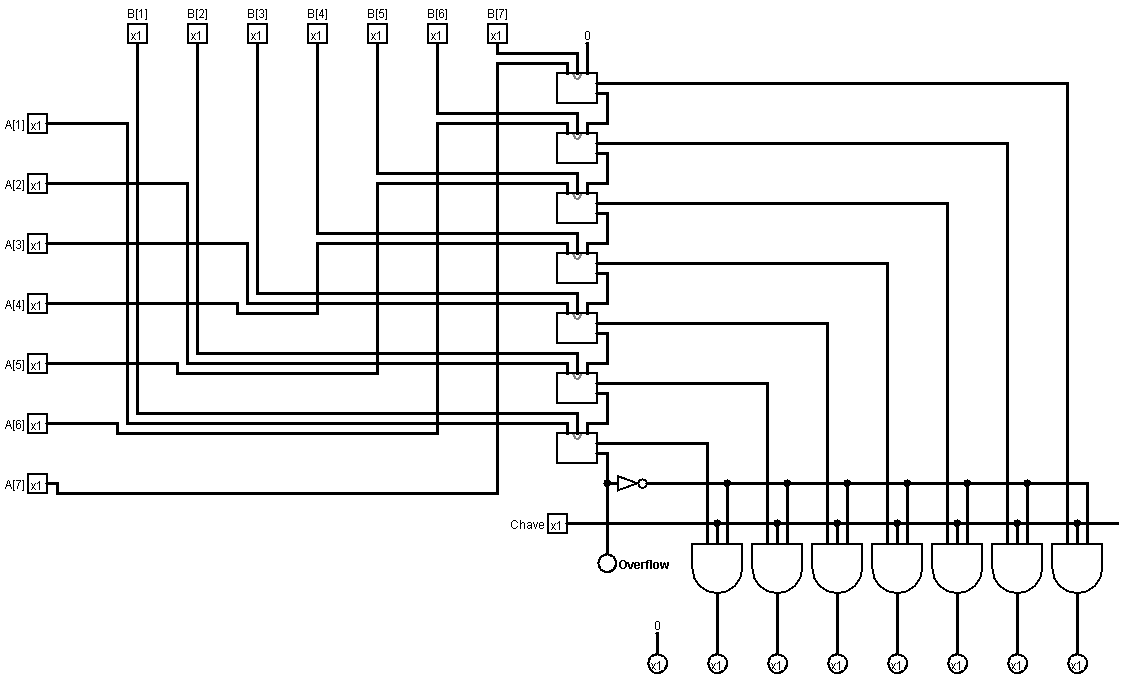
**Os circuito somadores e subtratores que foram desenvolvidos para 8 bits, operam internamente com um circuito de 1 bit encapsulado e possuem 7 entradas ‘A’ e 7 entradas ‘B’, pois um bit de saída é usado para mostrar o sinal da operação (0 = positivo ou 0 e 1 = negativo). Os demais circuitos de 8 bits apresentam 16 entradas, 8 ‘A’ e 8 ‘B’.**

**Todos os circuitos de 8 bits constam também com uma entrada extra de chave seletora, que fará um AND com a operação final antes da saída, para que a saída sempre seja nula se a chave não tiver ativada para este circuito. Em um componente real, poderíamos colocar o resultado lógico desta chave no VCC do componente, ou seja, cortando a energia do componente em ‘0’ e saturando positivamente em ‘1’.**

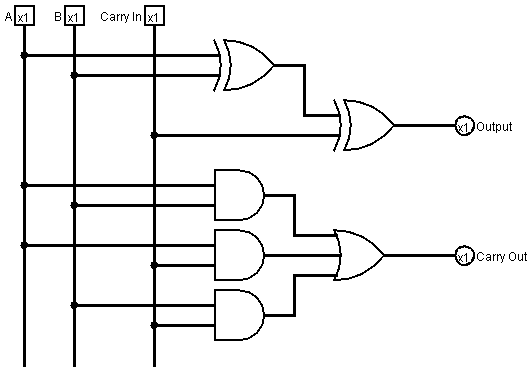
**O circuito subtrator realiza ‘A – B’ e os circuitos lógicos ‘Maior Que’ e ‘Menor Que’ são em relação a ‘A’.**

**Circuito Somador:**

**8 Bits**

****

**1 Bit**

****

O circuito somador consiste em somar o código binário das duas palavras. Ele faz uma operação XOR entre o bit ‘A’ e o bit ‘B’, e em seguida, faz outro XOR com o Carry-In. Existe também uma saída Carry Out, que será ‘1’ quando pelo menos duas das três entradas forem ‘1’. O Carry-Out tornará o Carry-In do somador implementado em cascata no seu próprio estado.

O circuito ligará um LED no caso de OVERFLOW no tamanho da palavra resultante e tornará todas as saídas em ‘0’ para proteção de dados.

Vale lembrar que o sinal desta operação sempre será positivo (0).

**Pinagem 1 Bit:**

****

**1 – Entrada A**

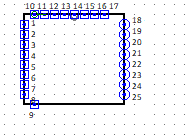
**2 – Entrada B**

**3 – Carry In**

**4 – Saída**

**5 – Carry Out**

**Pinagem 8 Bits:**

****

**1 – Entrada A0**

**2 – Entrada A1**

**3 – Entrada A2**

**4 – Entrada A3**

**5 – Entrada A4**

**6 – Entrada A5**

**7 – Entrada A6**

**8 – Chave Seletora**

**9 – Entrada B0**

**10 – Entrada B1**

**11 – Entrada B2**

**12 – Entrada B3**

**13 – Entrada B4**

**14 – Entrada B5**

**15 – Entrada B6**

**16 – Saída S0**

**17 – Saída S1**

**18 – Saída S2**

**19 – Saída S3**

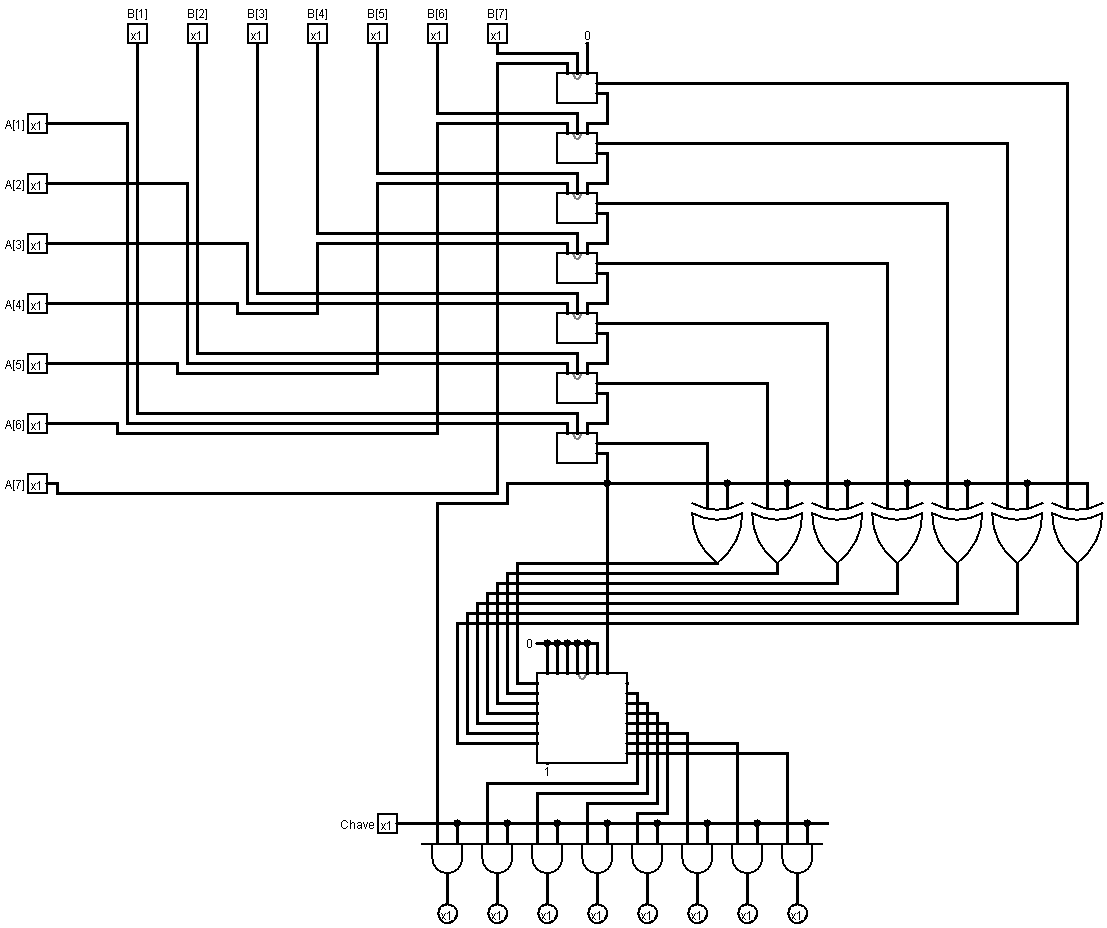
**20 – Saída S4**

**21 – Saída S5**

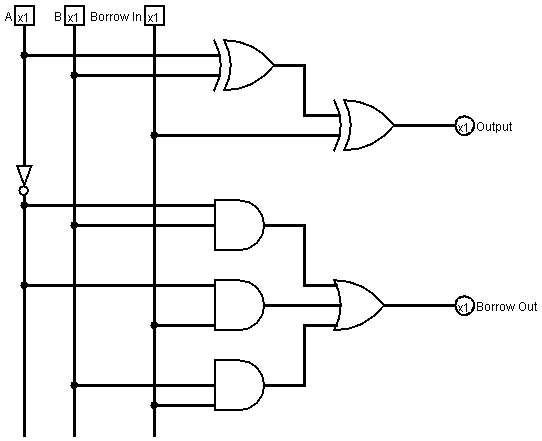
**22 – Saída S6**

**23 – Saída S7**

**Circuito Subtrator:**

****

**1 Bit:**

****

O circuito subtrator faz a subtração das duas palavras. As entradas realizam um XOR entre si e em seguida um XOR com o Borrow In para configurar a saída. O Borrow In é alimentado pelo Borrow Out do circuito subtrator implementado em cascata anteriormente. O Borrow Out é configurado para ‘1’ quando ‘B’ e o Borrow In estiverem em ‘1’ ou quando ‘A’ estiver em ‘0’ e o Borrow In ou o ‘B’ estiverem em 1.

Vale lembrar que o último Borrow Out está ligado na saída que representa o sinal. Se o sinal for negativo (‘1’), aplicamos complemento de 2, ou seja, invertemos todas as outras saídas com um NOT e usamos o circuito somador para somar com +1.

**Pinagem 1 Bit:**

****

**1 – Entrada A**

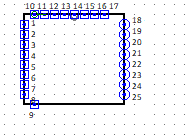
**2 – Entrada B**

**3 – Borrow In**

**4 – Saída**

**5 – Borrow Out**

**Pinagem 8 Bits:**

****

**1 – Entrada A0**

**2 – Entrada A1**

**3 – Entrada A2**

**4 – Entrada A3**

**5 – Entrada A4**

**6 – Entrada A5**

**7 – Entrada A6**

**8 – Chave Seletora**

**9 – Entrada B0**

**10 – Entrada B1**

**11 – Entrada B2**

**12 – Entrada B3**

**13 – Entrada B4**

**14 – Entrada B5**

**15 – Entrada B6**

**16 – Saída S8**

**17 – Saída S9**

**18 – Saída S10**

**19 – Saída S11**

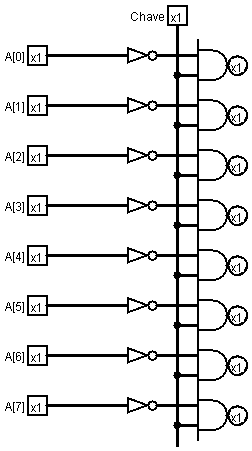
**20 – Saída S12**

**21 – Saída S13**

**22 – Saída S14**

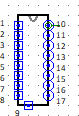
**23 – Saída S15**

**Circuito lógico NOT:**

****

Este circuito só opera com as entradas ‘A’ (de 0 a 7) e inverte o estado lógico das mesmas.

**Pinagem:**

****

**1 – Entrada A0**

**2 – Entrada A1**

**3 – Entrada A2**

**4 – Entrada A3**

**5 – Entrada A4**

**6 – Entrada A5**

**7 – Entrada A6**

**8 – Entrada A7**

**9 – Chave Seletora**

**10 – Saída S16**

**11 – Saída S17**

**12 – Saída S18**

**13 – Saída S19**

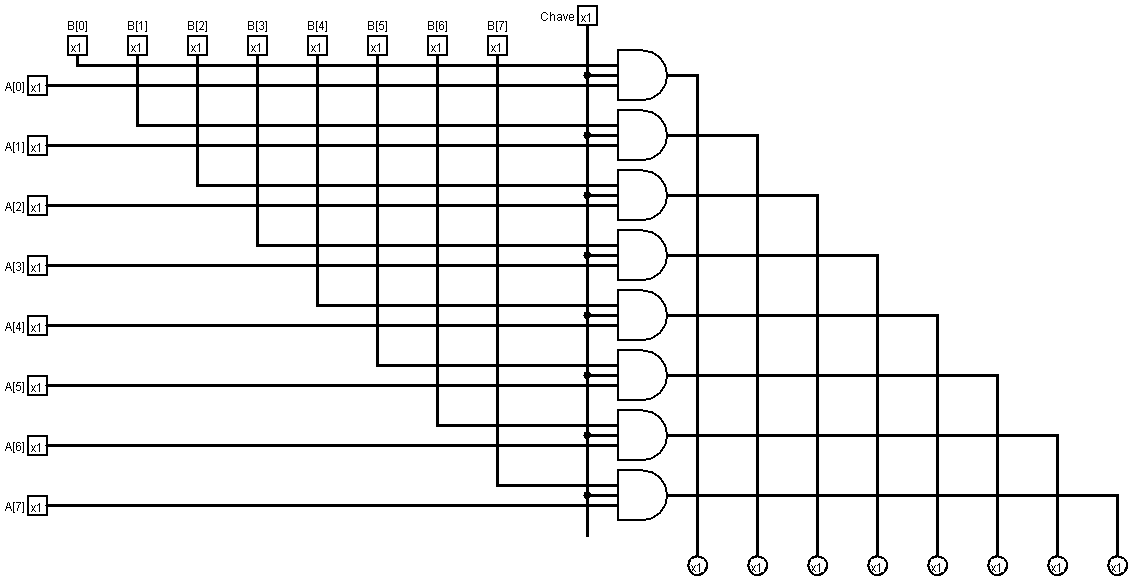
**14 – Saída S20**

**15 – Saída S21**

**16 – Saída S22**

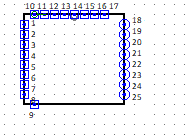
**17 – Saída S23**

**Circuito lógico AND:**

****

Este circuito faz uma operação de AND bit a bit entre as entradas ‘A’ e as ‘B’. Equivale ao operador ‘&’ usado em várias linguagens de programação, como C e Java. Este operando pode ser utilizado como uma forma otimizada de verificar se um número é par, por exemplo. O comando: if(n&1 == 0) é mais otimizado, quando n > 1 do que if(n%2 == 0), visto que o operador irá mascarar todos os bits do número que não forem o bit menos significativo.

**Pinagem:**

****

**1 – Entrada A0**

**2 – Entrada A1**

**3 – Entrada A2**

**4 – Entrada A3**

**5 – Entrada A4**

**6 – Entrada A5**

**7 – Entrada A6**

**8 – Entrada A7**

**9 – Chave Seletora**

**10 – Entrada B0**

**11 – Entrada B1**

**12 – Entrada B2**

**13 – Entrada B3**

**14 – Entrada B4**

**15 – Entrada B5**

**16 – Entrada B6**

**17 – Entrada B7**

**18 – Saída S24**

**19 – Saída S25**

**20 – Saída S26**

**21 – Saída S27**

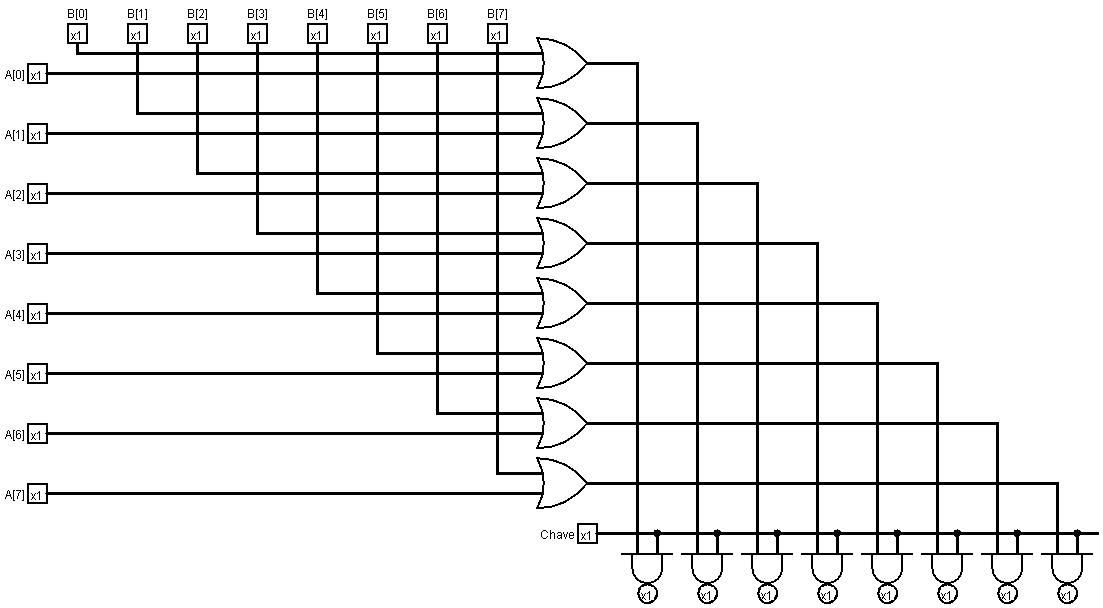
**22 – Saída S28**

**23 – Saída S29**

**24 – Saída S30**

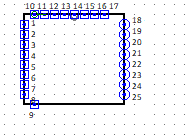
**25 – Saída S31**

**Circuito lógico OR:**

****

Este circuito faz a operação OR bit a bit das duas palavras. Equivale ao operador ‘|’ presente em várias linguagens de programação como C e Java.

**Pinagem:**

****

**1 – Entrada A0**

**2 – Entrada A1**

**3 – Entrada A2**

**4 – Entrada A3**

**5 – Entrada A4**

**6 – Entrada A5**

**7 – Entrada A6**

**8 – Entrada A7**

**9 – Chave Seletora**

**10 – Entrada B0**

**11 – Entrada B1**

**12 – Entrada B2**

**13 – Entrada B3**

**14 – Entrada B4**

**15 – Entrada B5**

**16 – Entrada B6**

**17 – Entrada B7**

**18 – Saída S32**

**19 – Saída S33**

**20 – Saída S34**

**21 – Saída S35**

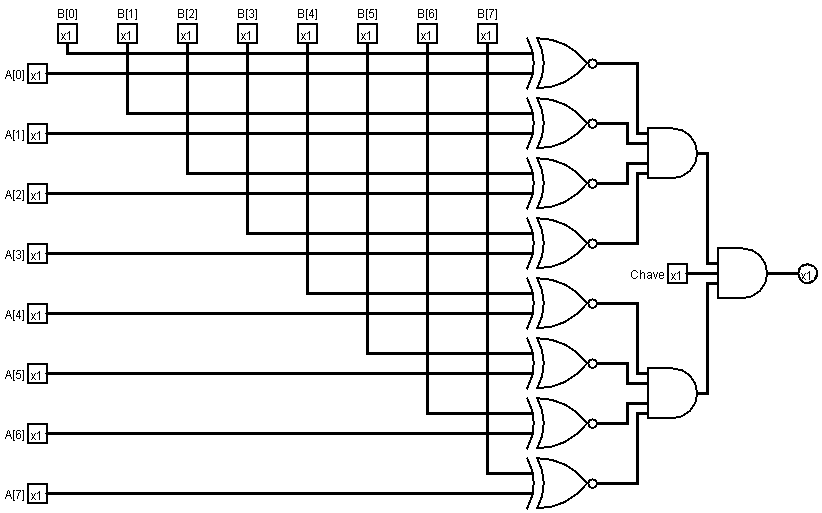
**22 – Saída S36**

**23 – Saída S37**

**24 – Saída S38**

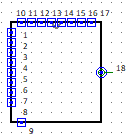
**25 – Saída S39**

**Circuito lógico Comparador de Igualdade:**

****

Este circuito compara se as duas palavras são iguais. O bit ‘A0’ compara se seu estado é igual ao de ‘B0’ através de um XNOR, o bit ‘A1’ compara com ‘B1’ e assim por diante. Tudo é operado com um AND para verificar bit a bit se todos de ‘A’ são iguais aos correspondentes de ‘B’.

**Pinagem:**

****

**1 – Entrada A0**

**2 – Entrada A1**

**3 – Entrada A2**

**4 – Entrada A3**

**5 – Entrada A4**

**6 – Entrada A5**

**7 – Entrada A6**

**8 – Entrada A7**

**9 – Chave Seletora**

**10 – Entrada B0**

**11 – Entrada B1**

**12 – Entrada B2**

**13 – Entrada B3**

**14 – Entrada B4**

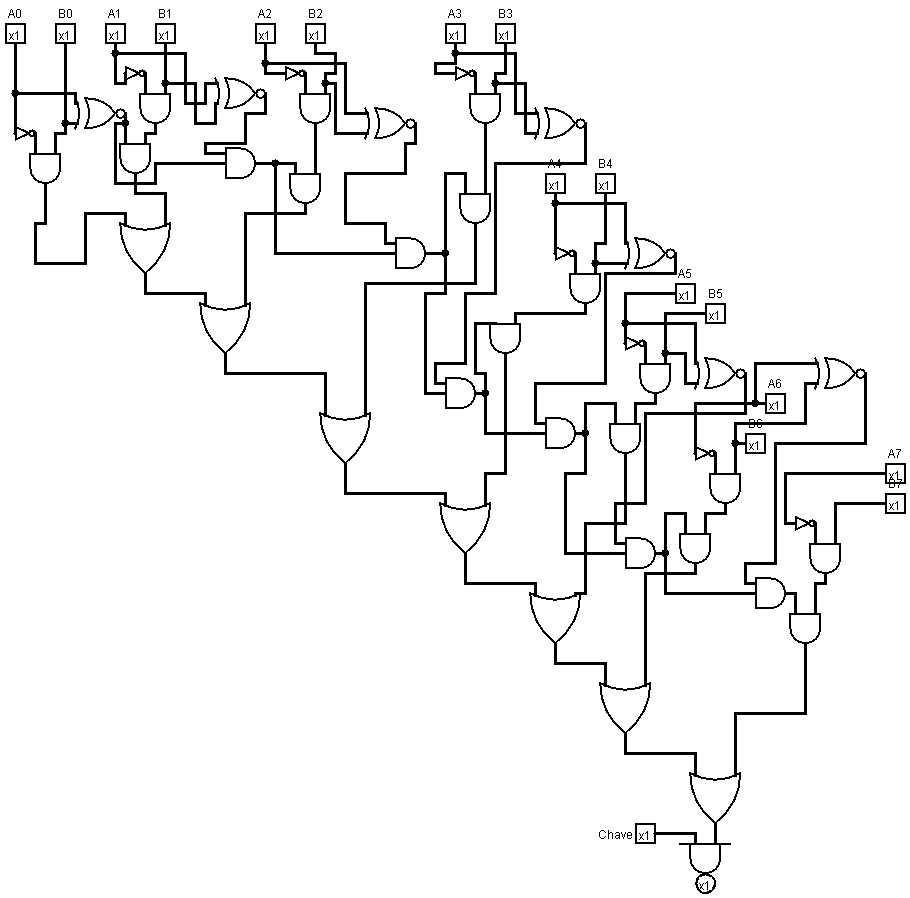
**15 – Entrada B5**

**16 – Entrada B6**

**17 – Entrada B7**

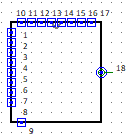
**18 – Saída S40**

**Circuito lógico Comparador de Menor Que:**

****

Neste circuito, comparamos o bit mais significativo ‘A’ com o mais significativo ‘B’. Se o bit de ‘B’ for maior que o de ‘A’, o circuito já saberá que o resultado será verdade (1). Caso ‘A’ for maior, o circuito já saberá que é falso (0). Mas no caso dos bits serem iguais, usando um XNOR, conseguimos passar a comparação para os bits conseguintes e assim por diante. O circuito consegue sempre voltar recursivamente para verificar se os bits anteriores são iguais de fato, visto que para comparar os bits ‘An ‘ e ‘Bn ‘, utilizamos um AND com o resultado da comparação de XNOR dos bits anteriores, e estes por sua vez, executam um AND com os antes destes.

**Pinagem:**

****

**1 – Entrada A0**

**2 – Entrada A1**

**3 – Entrada A2**

**4 – Entrada A3**

**5 – Entrada A4**

**6 – Entrada A5**

**7 – Entrada A6**

**8 – Entrada A7**

**9 – Chave Seletora**

**10 – Entrada B0**

**11 – Entrada B1**

**12 – Entrada B2**

**13 – Entrada B3**

**14 – Entrada B4**

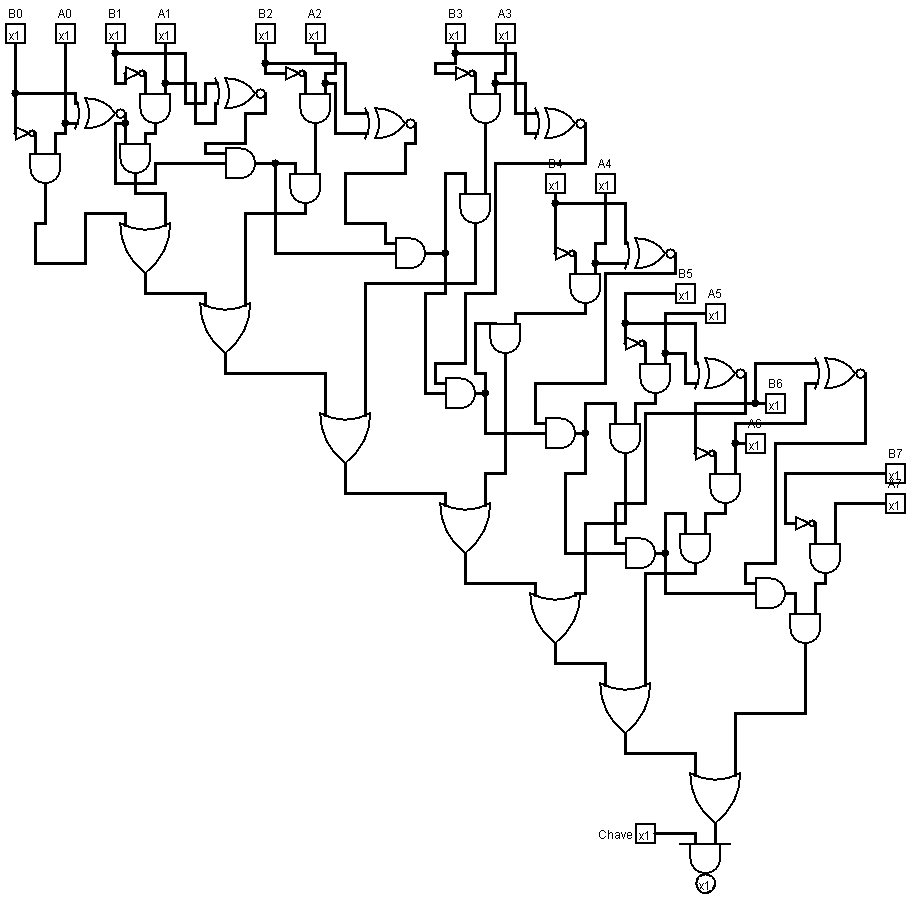
**15 – Entrada B5**

**16 – Entrada B6**

**17 – Entrada B7**

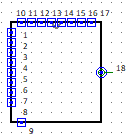
**18 – Saída S41**

**Comparador de Maior Que:**

****

Mesma coisa que o anterior, só invertemos ‘A’ e ‘B’.

**Pinagem:**

****

**1 – Entrada B0**

**2 – Entrada B1**

**3 – Entrada B2**

**4 – Entrada B3**

**5 – Entrada B4**

**6 – Entrada B5**

**7 – Entrada B6**

**8 – Entrada B7**

**9 – Chave Seletora**

**10 – Entrada A0**

**11 – Entrada A1**

**12 – Entrada A2**

**13 – Entrada A3**

**14 – Entrada A4**

**15 – Entrada A5**

**16 – Entrada A6**

**17 – Entrada A7**

**18 – Saída S42**