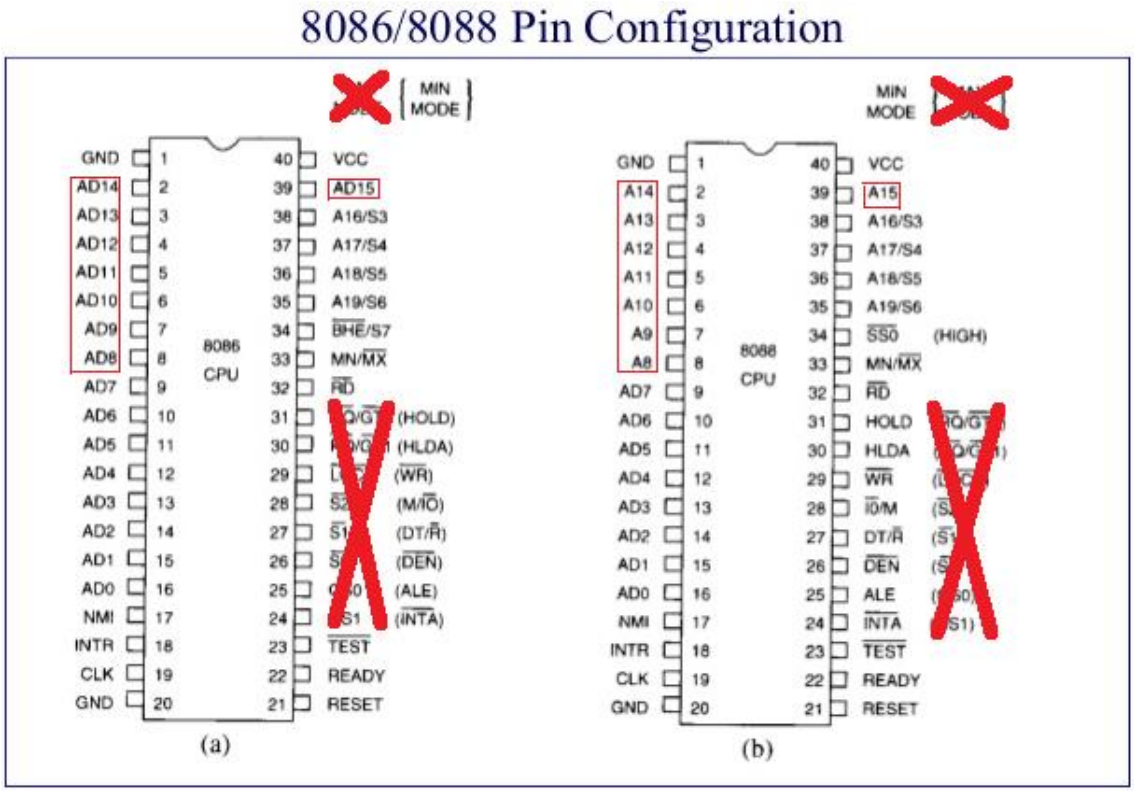
Resumo da parte de hardware de Sistemas Embarcados I

Atividade 1 – Projete um sistema mínimo com processador da família MSC86:

O primeiro passo da questão 1 é desenhar o processador que será utilizado e suas conexões. Aqui escolheremos entre os processadores 8086 ou 8088, a diferença entre eles é pouca (marcado na figura) e no fim ***sempre*** será usado o 8088. O modo máximo foi cortado pois não usaremos.



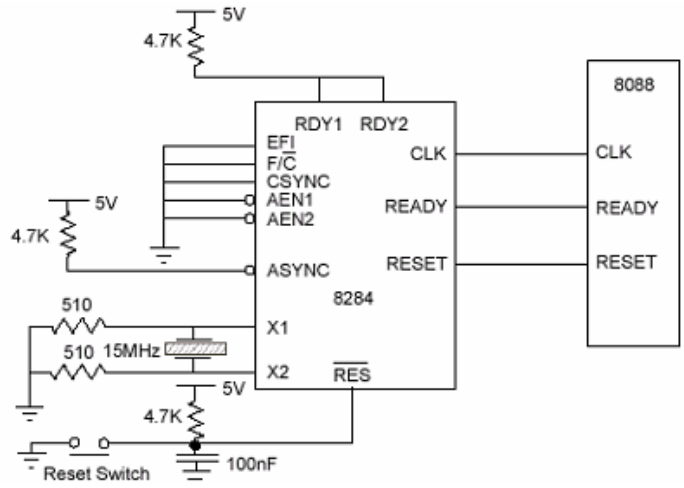
É importante saber a função de alguns pinos do processador assim será preciso decorar menor quantidade de conteúdo. Os pinos apresentados a seguir deverão ser desenhados na figura inicial.

- ADx: A diferença entre o 8086 e o 8088 é a quantidade de pinos AD, onde o 8086 tem 16 pinos e o 8088 tem 8 pinos. AD vem de Adress-Data, ou seja, em certas operações são usados para indicar um endereço e em outras servem para fornecer/receber dados. No 8088 temos do AD0 até o AD7.
- Ax: São pinos que apenas indicam o endereço a ser acessado. Vão do A8 ao A19. Obs.: Para endereçamento de memória (RAM e ROM) são usados 20bits (A0~A19), para portas de entrada e saída são usados 16bits (apenas um registrador, A0 ~ A15).
- INTR (Interruption Request): Este pino recebe a informação da existência de uma interrupção externa. O processador responde a requisição pelo pino $\overline{\text{INTA}}$. No caso, quem **manda** a sinalização é o 8259, chip de interrupção.

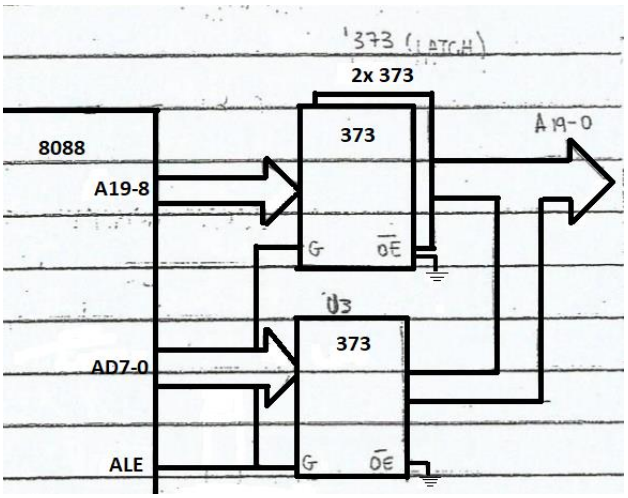
- $\overline{\text{INTA}}$ (Interruption Acknowledge): Assim que o processador estiver pronto para atender à requisição de interrupção ele sinaliza por este pino levando seu nível lógico para zero.
- MN/ $\overline{\text{MX}}$: Indica para o processador se ele está em modo mínimo ou modo máximo. Se este pino for ligado ao Vcc será modo mínimo, se ligado no terra é modo máximo. **SEMPRE** usaremos modo mínimo.
- $\overline{\text{RD}}$ (Read): Indica quando o processador quer fazer leitura.
- $\overline{\text{WR}}$ (Write): Indica quando o processador quer fazer escrita.
- IO/ $\overline{\text{M}}$ (Input Output/Memory): Neste pino o processador indica se fará um acesso à memória (RAM ou ROM) ou a um dispositivo de entrada/saída (Porta de entrada/saída). Nível lógico 1 caso o acesso seja à memória, 0 caso IO.
- DT/ $\overline{\text{R}}$ (Data Transmit/Receive): Indica se os pinos AD0~AD7 estão enviando dados ou recebendo dados. Nível lógico 1 caso enviando, 0 caso recebendo.
- $\overline{\text{DEN}}$ (Data Enable): Indica o funcionamento dos pinos ADx, se contém endereçamento (Ax) ou dados (Dx). Nível lógico 1 caso contenha endereço, 0 caso dados.
- ALE (Address Latch Enable): Indica quando um endereço válido está disponível nos pinos A0~A19, ele é responsável por avisar aos latchs (parte importante do desenho da atividade) que eles podem salvar os valores de endereçamento.
- CLK, RESET, READY: São pinos controle usados pelo gerador de clock, eles são ligados nos pinos de mesmo nome do gerador.

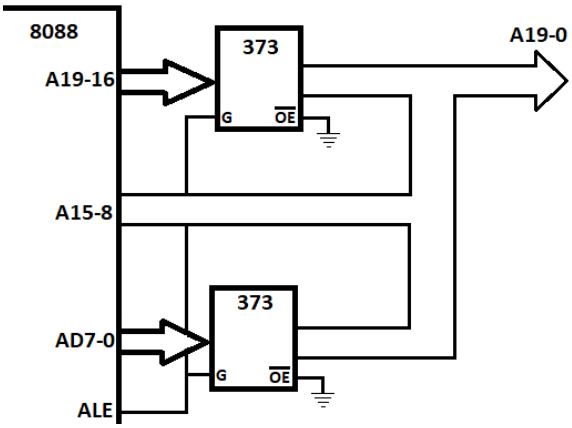
A primeira parte do desenho de hardware é o microprocessador 8088 ligado ao gerador de clock 8284. **LEMBREM-SE DE DESENHAR O 8284 INTEIRO!** (ele desconta 0,1 caso não façam).

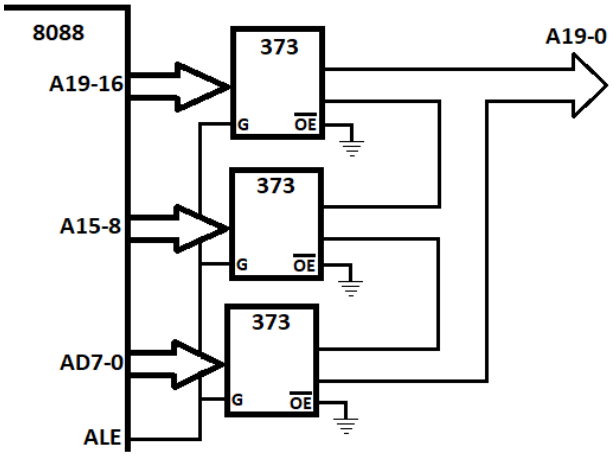
Esta parte não tem mistério, repete o desenho dado em prova do gerador de clock e ligue os pinos que estão sobrando no 8088, os pinos tem o mesmo nome.



Depois temos que desenhar o acesso do 8088 ao barramento de dados e de memória. Para isso usaremos os pinos AD0-7, A8-19, ALE, $\overline{DEN}$ e DT/$\overline{R}$. Ao acessar memória o endereço é salvo em latches, no caso usaremos o 373 (um dos chips Ux que ele passa na prova), cada 373 trabalha com 8 bits então usaremos 3 chips para salvar os 20 bits de endereçamento.

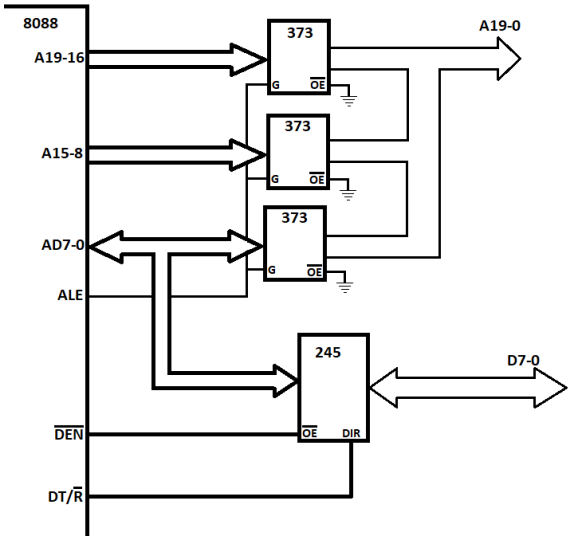






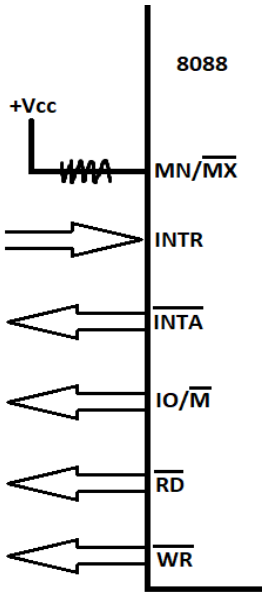
As figuras mostram três formas diferentes de representar como o processador endereça os dispositivos. O pino **G** do 373 indica quando ele deve salvar o valor que vem do processador, o $\overline{\textbf{OE}}$ disponibiliza para o barramento o valor armazenado (ele está sempre ativo).

Como o pino AD7-0 hora funciona para indicar um endereço, hora para receber/transmitir dados devemos completar o desenho para o acesso ao barramento de dados, para isso é adicionado um transceiver (transmiter-receiver) que é capaz de receber dados do processador e coloca-los no barramento como também de pegar dados do barramento e coloca-los no processador:

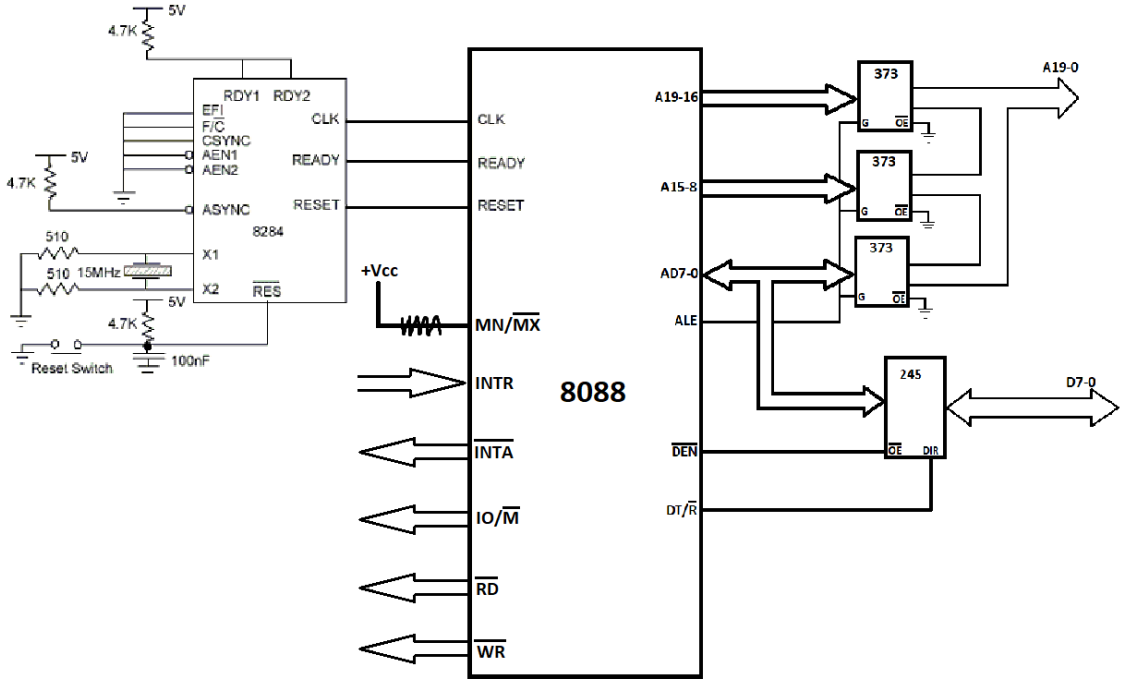


O pino **$\overline{\text{DEN}}$** (Data Enable) indica se será feito acesso aos dados no pino **$\overline{\text{OE}}$** (ativa o 245) e **DT/ $\overline{\text{R}}$** (Data Transmit/Receive) indica o fluxo de dados, se 1 os dados **saem** do processador, se 0 os dados **entram** no processador.

Para finalizar devemos desenhar o restante do 8088. Os pinos mostrados são para controle e apenas o **INTR** é recebido.



A primeira parte da questão 1 é os desenhos mostrados juntos, o que fica desta forma:



Agora vamos ao acesso as memórias. O processador para indicar qual segmento de memória irá acessar usa os pinos **A19-0,** $\overline{\textbf{RD}}$ **,** $\overline{\textbf{WR}}$ e **IO/** $\overline{\textbf{M}}$. Como o processador usar 20 bits para endereçamento de memória a capacidade máxima é de $2^{20} = 1\text{Mb}$ e o primeiro segmento tem o endereço de 0x00000 enquanto o último é de 0xFFFFF (20 bits = 5 hex).

Endereço de cada segmento de memória possível de ser endereçado pelo processador

Hexadecimal	Binário
0xFFFFF	1111 1111 1111 1111 1111
0xFFFFE	1111 1111 1111 1111 1110
0xFFFFD	1111 1111 1111 1111 1101
0xFFFFC	1111 1111 1111 1111 1100
● ● ●	● ● ●
0x00003	0000 0000 0000 0000 0011
0x00002	0000 0000 0000 0000 0010
0x00001	0000 0000 0000 0000 0001
0x00000	0000 0000 0000 0000 0000

A19 A18 A17 ... A2 A1 A0

Nos endereços mais altos são alocadas as memórias ROMs

Nos endereços mais baixos são alocadas as memórias RAMs

Obs.: Explicação não relevante para prova.

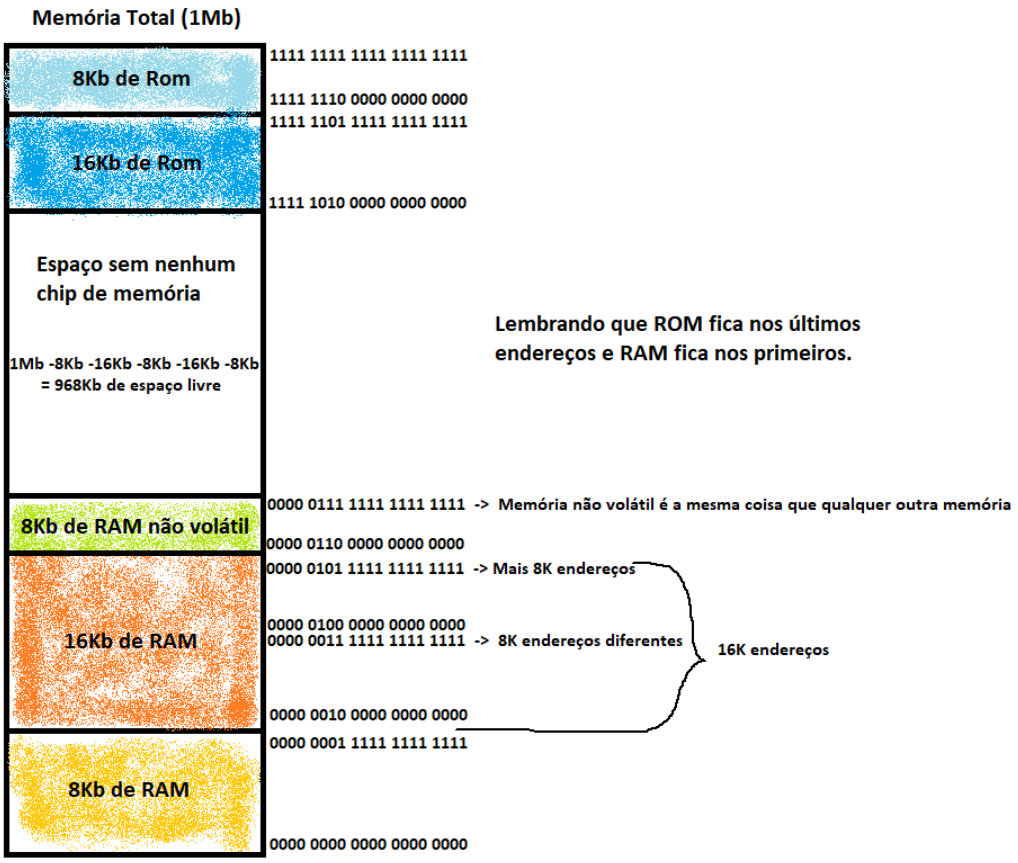
Para compor um endereço de 20 bits o processador usa dois registradores (CS:IP por exemplo)

AX:BX = shift left(AX)+BX

AX -> 0xAAAA
BX -> 0xBBBB

shl(AX) -> 0xAAAA0
BX + 0x BBBB
Endereço -> 0xB665B

Na prova é pedido para usar chips individuais de RAM e ROM com tamanhos que variam de prova para prova. Para entender melhor como usar esses chips vamos supor que temos que alocar 1 chip de 8kb e outro de 16kb de memória RAM, 1 chip de 8kb de memória RAM não volátil e 1 chip de 8kb e outro de 16kb de memória ROM. Poderíamos alocar estas memórias da seguinte forma:



A ideia da figura é mostrar que cada chip é alocado individualmente e o que há para ser feito é determinar o primeiro e o ultimo endereço que será atribuído a cada chip para que assim possamos fazer a decodificação correta de acesso à memória.

Vamos simplificar um pouco, supondo que usaremos 2 chips de 8kb de memória RAM, 2 de 16kb de RAM, 1 de 16kb de RAM não volátil e 3 chips de 8kb de ROM.

Temos que alocar 2*8kb e 3*16kb, para os chips de 8kb temos que é preciso de 13 bits para representar todos os segmentos da memória ($8 = 2^3$ e $kb = 2^{10} \rightarrow 10 + 3\ bits$), como temos 2 chips precisaremos de 1 bit para diferenciá-los, os bits restantes ($13 + 1 = 14\ bits \rightarrow faltam\ 6\ bits\ para\ preencher$) são preenchidos com '0'.

	Bits restantes para completar os 20 bits (A19 – A14)	Bit de indicação de chip (A13)	Range de endereços de segmentos (A12-A0)
Chip 1 – 8k	0000 00	0	0 0000 0000 0000
	0000 00	0	1 1111 1111 1111
Chip 2 – 8k	0000 00	1	0 0000 0000 0000
	0000 00	1	1 1111 1111 1111

Falta representar os chips de 16kb e a memória não volátil, as memórias de 16kb precisam de 14 bits para endereçar todos os segmentos ($16 = 2^4$ e $kb = 2^{10} \rightarrow 10 + 4\ bits$), e como temos 3 chips serão necessários 2 bits para diferenciá-los, preenchendo o restante com '0' ($14 + 2 = 16\ bits \rightarrow faltam\ 4$).

	Bits restantes para completar os 20 bits (A19 – A16)	Bits de indicação de chip (A15 – A14)	Range de endereços de segmentos (A13 – A0)
Chip 3 – 16k	0000	01	00 0000 0000 0000
	0000	01	11 1111 1111 1111
Chip 4 – 16k	0000	10	00 0000 0000 0000
	0000	10	11 1111 1111 1111
Chip 5 – 16k RAM não volátil	0000	11	00 0000 0000 0000
	0000	11	11 1111 1111 1111

O bit A14 começa em '1' porque as o chip 3 está no endereço logo após as RAMs de 8k, se os bits de indicação de chip fossem '00' os endereços do chip 3 seriam conflitantes com os do chip 1.

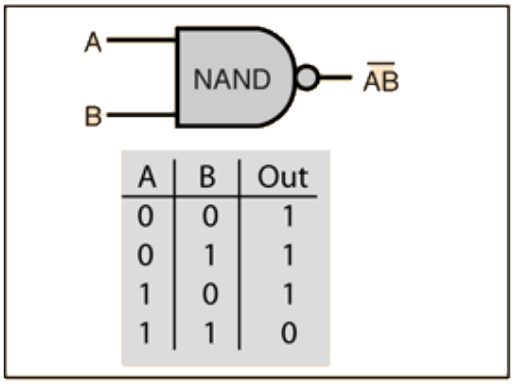
Para as ROMs é bem parecido, mas como elas são inseridas no fim da memória disponível os bits que preenchem até completar os 20 bits são '1' ao contrário das RAMs que preenchem com '0'.

Para os chips de 8kb das ROM temos que é preciso de 13 bits, como temos 3 chips precisaremos de 2 bits para diferenciá-los, os bits restantes ($13 + 2 = 15\ bits \rightarrow faltam\ 5\ bits$) são preenchidos com '1'.

	Bits restantes para completar os 20 bits (A19 – A15)	Bit de indicação de chip (A14 – A13)	Range de endereços de segmentos (A12-A0)
Chip 6 – 8k	1111 1	01	0 0000 0000 0000
	1111 1	01	1 1111 1111 1111
Chip 7 – 8k	1111 1	10	0 0000 0000 0000
	1111 1	10	1 1111 1111 1111
Chip 8 – 8k	1111 1	11	0 0000 0000 0000
	1111 1	11	1 1111 1111 1111

Reparem que os bits de indicação não começaram do '00', isso porque o chip 8 tem que estar no fim dos endereços (0xFFFFF -> 20 bits '1'). Então chip 8 -> '11' para os bits de indicação, chip 7 -> '10' e chip 6 -> '01'. (Se houvesse uma quarta ROM ela seria '00')

É possível fazer o circuito com portas **OR**, mas elas são um pouco mais difíceis de explicar, então usaremos as portas **NAND** já que são mais intuitivas. Na lógica de decodificação usaremos os bits de complemento, os bits de indicação de chip e o **IO/** $\overline{\textbf{M}}$ para indicar qual chip estamos querendo acessar.

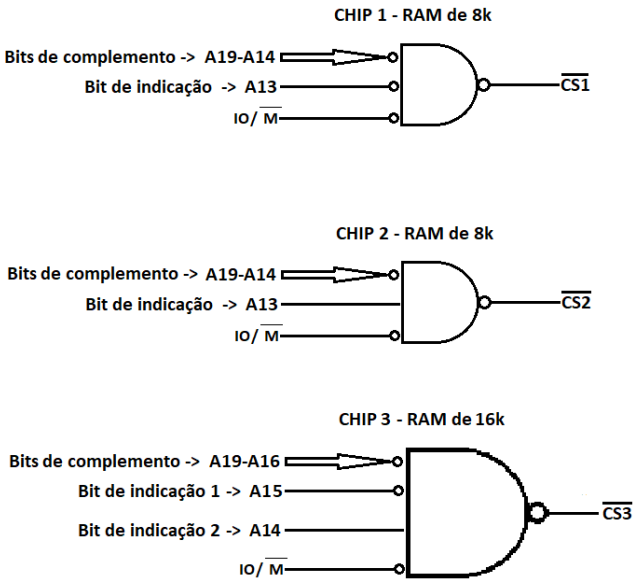


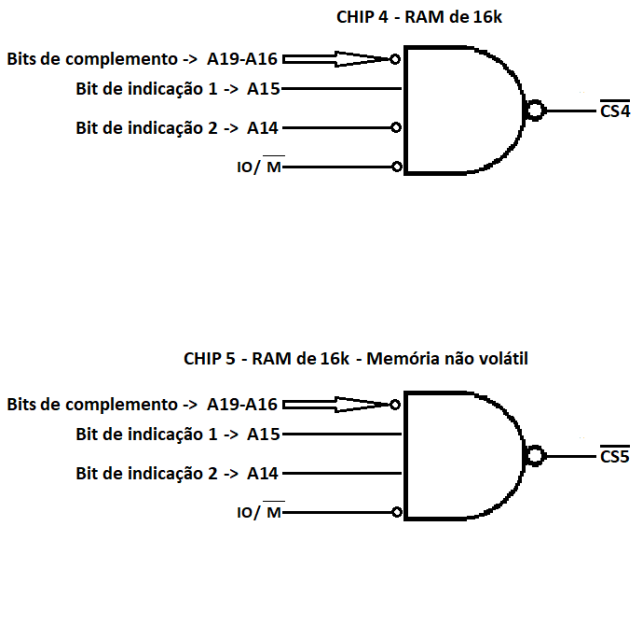
A	B	Out
0	0	1
0	1	1
1	0	1
1	1	0

Quando fazemos a faixa de endereço de cada chip tomamos o cuidado de nenhum chip ter o mesmo endereço de outro, pois quando há acesso à memória queremos garantir que somente um chip coloque informações no barramento de dados para que não haja conflito. A lógica de decodificação deve garantir que não exista acesso simultâneo.

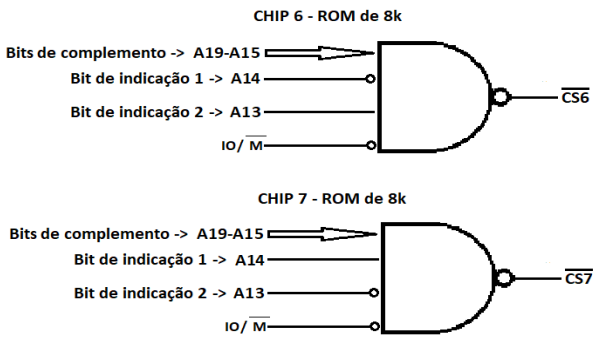
A porta **NAND** é usada para isso por sua saída ser '0' apenas quando todas as condições forem atendidas, lembrando que o enable (Chip Select) da memória ocorre em nível lógico zero, logo o chip só será ativo quando as entradas forem todas '1' (por isso negamos as entradas que queremos que sejam '0').

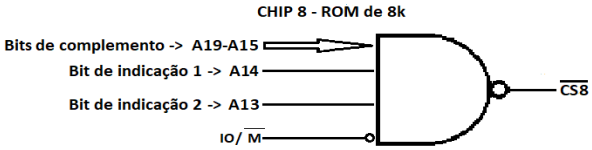
Assim a lógica de decodificação para a memória RAM fica:





E a lógica das ROMs fica:





A memória RAM permite tanto a escrita de dados como a leitura do seus dados, enquanto a ROM só permite leitura dos dados. Sendo assim precisamos informar à RAM qual operação será feita usando os pinos $\overline{\textbf{RD}}$ e $\overline{\textbf{WR}}$ para isso. A ROM usará apenas o $\overline{\textbf{RD}}$.