As linhas ***S0, *DEN, *S1, DT/*R, *S2, M/*IO** entram em alta impedância um pouco antes da CPU emitir o reconhecimento de *hold (hlda)*. Com as linhas **AD0...AD15, A16/S3, A17/S4, A18/S5** e **A19/S6**, isto acontece durante o reconhecimento de interrupção.

2.4. Barramentos de Endereços e Dados

Como já foi visto, barramentos são conjuntos de linhas condutoras por onde trafegam sinais de uma determinada espécie. Os barramentos, normalmente, interconectam diversos dispositivos e, por isso, oferecem uma carga considerável. As CPUs, devido à baixa capacidade de corrente, necessitam de *drivers* (amplificadores lógicos de corrente) para entregar seus sinais ao barramento. Para retirar os sinais do barramento, usa-se um dispositivo similar, denominado *receiver*. Para as linhas bidirecionais, torna-se necessário um *driver* e um *receiver*, a esse conjunto dá-se o nome *transceiver*. O termo *buffer* é usado para designar qualquer dispositivo que repotencialize (amplifique) os sinais. Para guardar sinais que apareçam no barramento, por um curto período de tempo, usa-se um dispositivo denominado *latch*.

Para trabalhar corretamente, uma memória ou um dispositivo de I/O, conectados à CPU, necessitam de que os endereços fiquem estáveis durante todo o ciclo de barramento. Como os endereços aparecem apenas durante o instante T1, é necessário armazená-los, o que pode ser conseguido com o emprego de *latches*. Para amplificar e controlar o sentido dos dados, torna-se necessário o emprego de *transceiveres*. A Intel comercializa um conjunto de latches e *transceiveres*:

- 8282 → *latch* não inversor,
- 8283 → *latch* inversor,
- 8286 → *transceiver* não inversor,
- 8287 → *transceiver* inversor.

Porém, no presente estudo, adotar-se-ão os TTLs:

- 74LS373 → *latch* não inversor e
- 74LS245 → *transceiver* não inversor.

2.4.1. Como Separar o Barramento de Endereços

A forma mais usual de criar um barramento de endereços (linhas onde os endereços ficam estáveis durante todos os períodos do ciclo de barramento) é separar os endereços através de latches, controlados pelo sinal ALE. O 74LS373, como ilustrado na figura 2.11, é um *latch* de 8 bits (8 flip-flops D em paralelo), que possui um sinal de controle (***OE**), para os três-estados de saída, e um *gate* (G), para controlar a operação de *latch*. Quando G = 1, o *latch* é transparente, ou seja, a saída é igual à entrada; porém, quando G = 0, a saída fica congelada com o último valor. A figura 2.12 mostra um exemplo da demultiplexação do barramento de endereços.

*OE	G	Saída
1	X	Alta Impedância
0	1	DO = DI
0	0	DO = Anterior

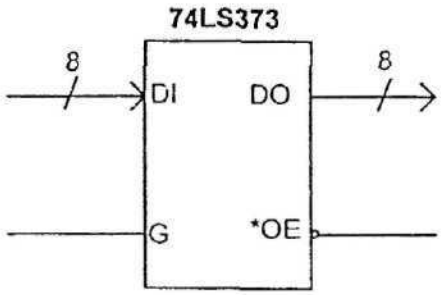


Figura 2.11. Descrição do latch 74LS373.

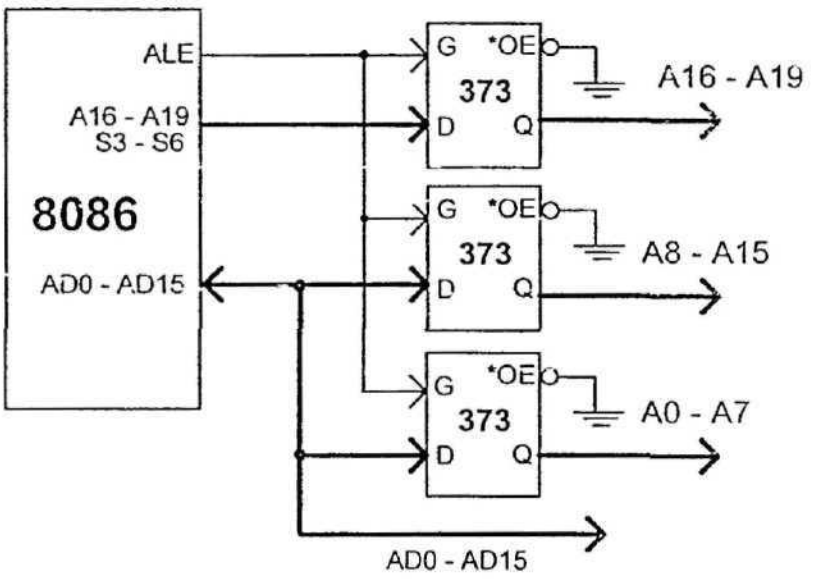


Figura 2.12. Demultiplexação do barramento de endereços.

O barramento de endereços deve ser capturado o mais tarde possível para permitir a estabilização de suas linhas. Esse instante é bem caracterizado no flanco de descida do sinal **ALE**.

2.4.2. Como Usar a Memória com o 8086

O espaço de memória endereçado pelo 8086 pode ser visualizado como uma seqüência de 1MB, na qual um byte pode conter qualquer valor de 8 bits e dois bytes consecutivos formam uma palavra *(word)* de 16 bits. O endereço de uma palavra de 16 bits é dado pelo endereço do seu byte menos significativo e isto é o que se chama de *"little endian"*. Não existe qualquer restrição sobre onde colocar um byte ou uma palavra de 16 bits. Ou seja, podem estar em qualquer endereço. O espaço de endereçamento é fisicamente conectado ao barramento de 16 bits através da sua divisão em dois bancos de 512 KB, como ilustrado na figura 2.13.

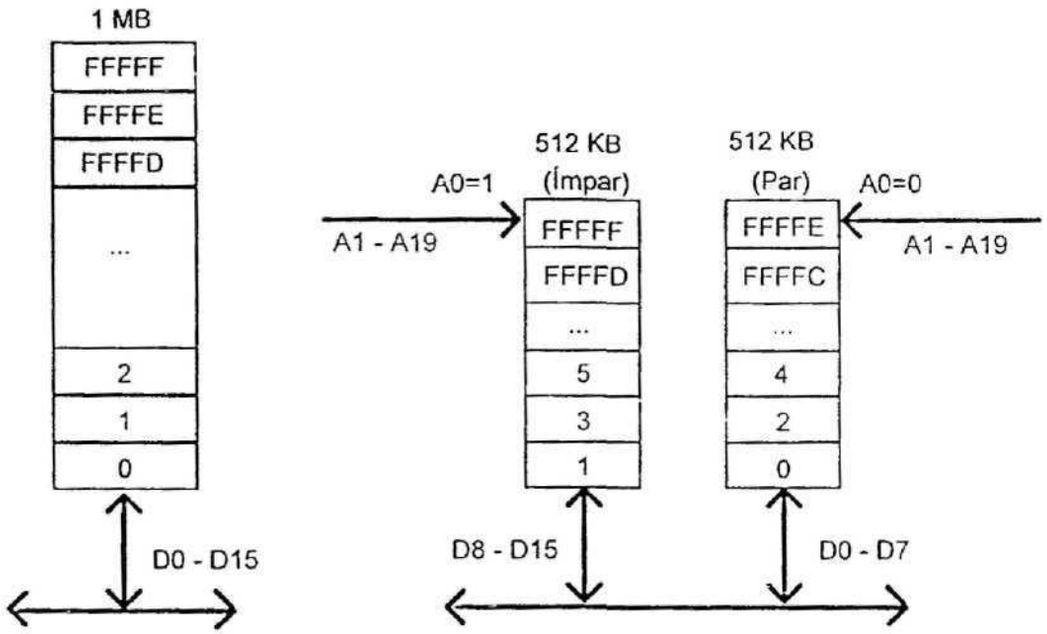


Figura 2.13. Divisão da memória de 1 MB em dois bancos de 512 KB.

Um banco está conectado à metade inferior do barramento de dados (**D0 - D7**) e contém os bytes com endereços pares (**A0** = 0). O outro banco está conectado à metade superior (**D8 - D15**) e contém os bytes com endereços ímpares (**A0** = 1). As linhas **A1 - A19** selecionam um byte específico dentro de cada banco. A idéia que surge de imediato é a de habilitar os bancos de acordo com **A0**. Se **A0** = 0, habilitar banco par e, se **A0** = 1, habilitar banco ímpar, como ilustrado na figura 2.14. Isto funciona, porém impede o uso de palavras de 16 bits, pois, para acessá-las, é necessário habilitar os dois bancos simultaneamente.

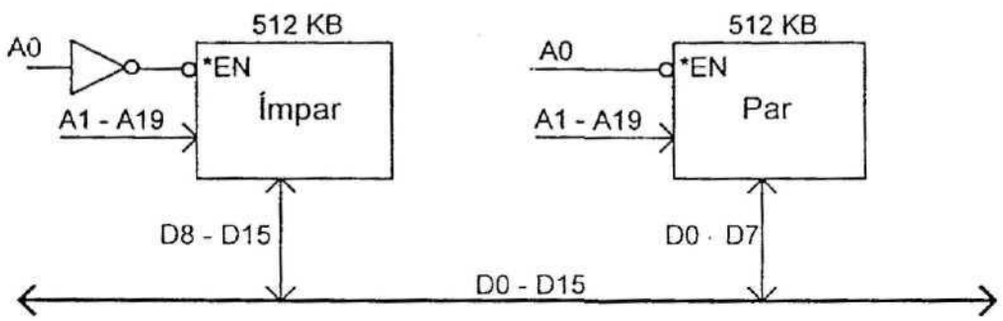


Figura 2.14. Esquema deficiente, pois não permite transferências de 16 bits.

Para permitir o trabalho com bytes e palavras de 16 bits, existe a linha ***BHE** (Bus High Enable)*, que controla o banco ímpar. Agora, cada banco possui um controle separado: **A0** para o banco par e ***BHE** para o banco ímpar, como ilustrado na figura 2.15. Se os dois controles são ativados ao mesmo tempo, os dois bancos ficam habilitados, permitindo um acesso de 16 bits.

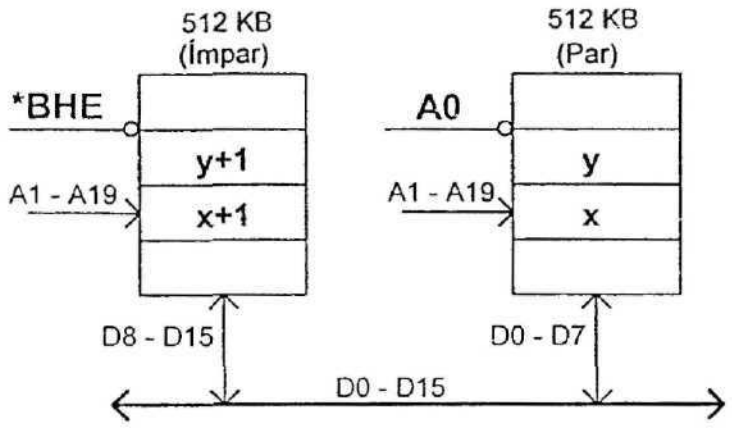


*Figura 2.15. Uso de A0 e *BHE para controlar dois bancos de 512 KB.*

Para acessar um byte que está num endereço par, a CPU ativa o banco par e desativa o banco ímpar. Portanto, **A0** é colocado em 0 (endereço par), para selecionar o banco de memória conectada à porção inferior do barramento de dados, e a linha ***BHE** é colocada em 1, para desabilitar o banco ligado à porção superior do barramento de dados. A figura 2.16 demonstra esse caso.

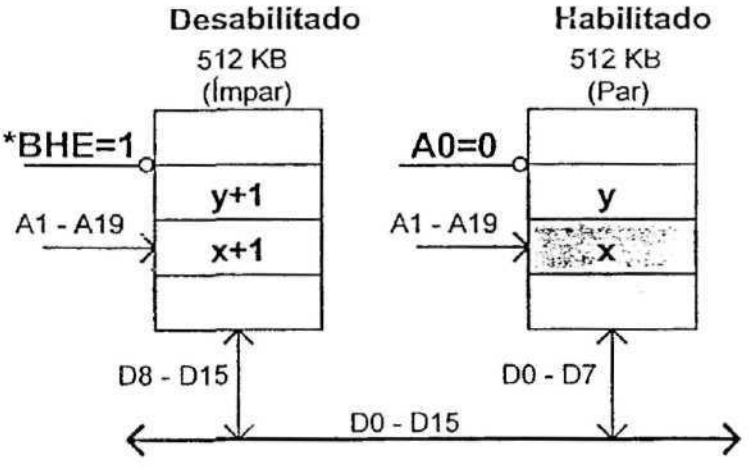


Figura 2.16. Acesso a um byte de endereço par (endereço x).

A desabilitação do banco ímpar (***BHE** = 1) seria dispensável nas operações de leitura, pois o processador sabe, a priori, qual dos bytes vindos do barramento lhe seria útil. Porém, na escrita, o banco ímpar deve ser desabilitado para evitar que seja alterado indevidamente. Para endereçar um byte que está em um endereço ímpar, já que a informação será transferida pela parte alta do barramento de dados (**D8-D15**), a CPU habilita ***BHE** e, como o endereço é ímpar (**A0** = 1), o banco par fica automaticamente desabilitado. Este caso está ilustrado na figura 2.17.

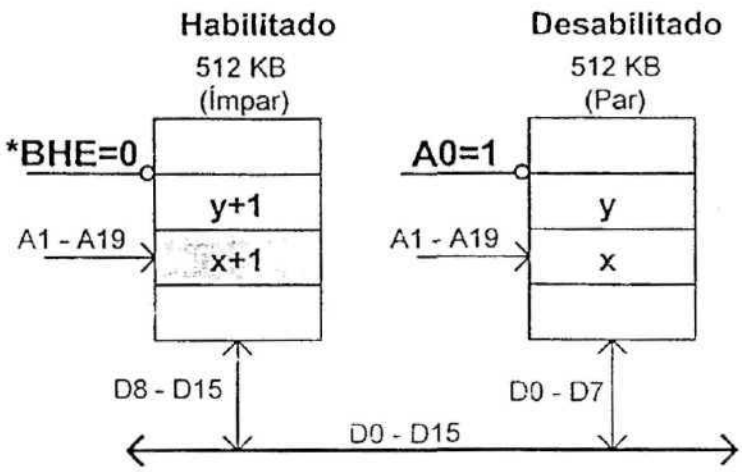


Figura 2.17. Acesso a um byte de endereço ímpar (endereço x+1).

Algumas vezes, torna-se necessário transferir um byte da metade alta para a metade baixa do barramento de dados. Por exemplo, a CPU 8086 possui um registrador de 16 bits denominado AX. Este registrador está dividido em duas partes de 8 bits: AH é a parte alta e AL é a parte baixa. Caso seja necessário carregar AL com um byte que está em um endereço ímpar, torna-se necessária essa transferência. O dado será acessado através da parte alta do barramento de dados (D8 - D15). Ao chegar no 8086, esse dado será transferido para a parte baixa do barramento interno da CPU para então ser entregue a AL. Algo semelhante acontece quando se carrega em AH um byte que está em um endereço par. Este recurso é interessante porque permite que transferências de I/O, usando o registrador AL, acessem dispositivos conectados na parte baixa ou na parte alta do barramento de dados.

O endereço de uma palavra de 16 bits, como já foi visto, é dado pelo endereço do seu byte menos significativo. Por exemplo, a palavra de 16 bits definida pelo endereço 200h ocupa os endereços 200h (menos significativo) e 201h (mais significativo). O acesso a palavras de 16 bits, localizadas em endereços pares, é feito em um único ciclo de barramento. As linhas **A1-A19** selecionam um byte em cada banco, sendo que **A0** = 0 e ***BHE** = 0 habilitam os dois bancos simultaneamente. A figura 2.18 ilustra esse acesso.

Palavras de 16 bits localizadas em endereço ímpar são problemáticas, pois os bits **A1** dos dois endereços consecutivos são diferentes. Portanto, com o esquema da figura 2.18, não é possível realizar o acesso em um único ciclo de barramento. Desta forma, tais palavras só podem ser acessadas em dois ciclos de barramento. No primeiro ciclo, o byte menos significativo é acessado e, durante o segundo ciclo, acessa-se o byte mais significativo. Durante o primeiro ciclo de barramento, **A1-A19** especificam o endereço do primeiro byte, com **A0** = 1 (endereço ímpar) e ***BHE** = 0. Assim, o banco par é desabilitado e o banco ímpar é habilitado. Durante o ciclo seguinte, o endereço é incrementado, com **A0** = 0 e ***BHE** = 1; portanto, o banco ímpar é desabilitado e o par é habilitado. Esta seqüência está na figura 2.19.

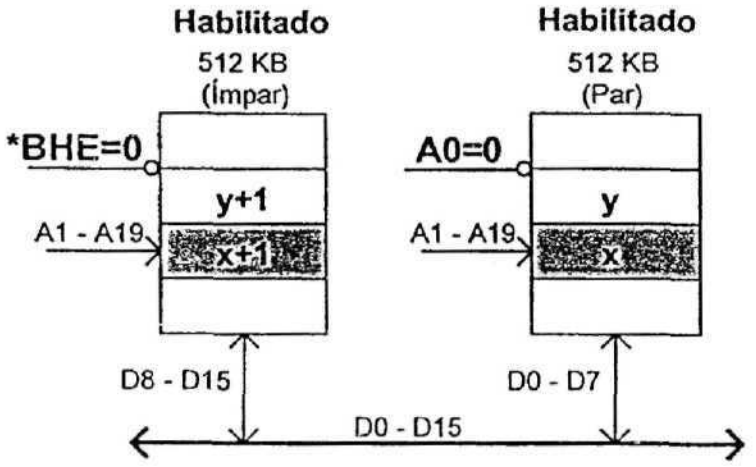


Figura 2.18. Acesso a uma palavra de 16 bits em endereço par (endereço x).

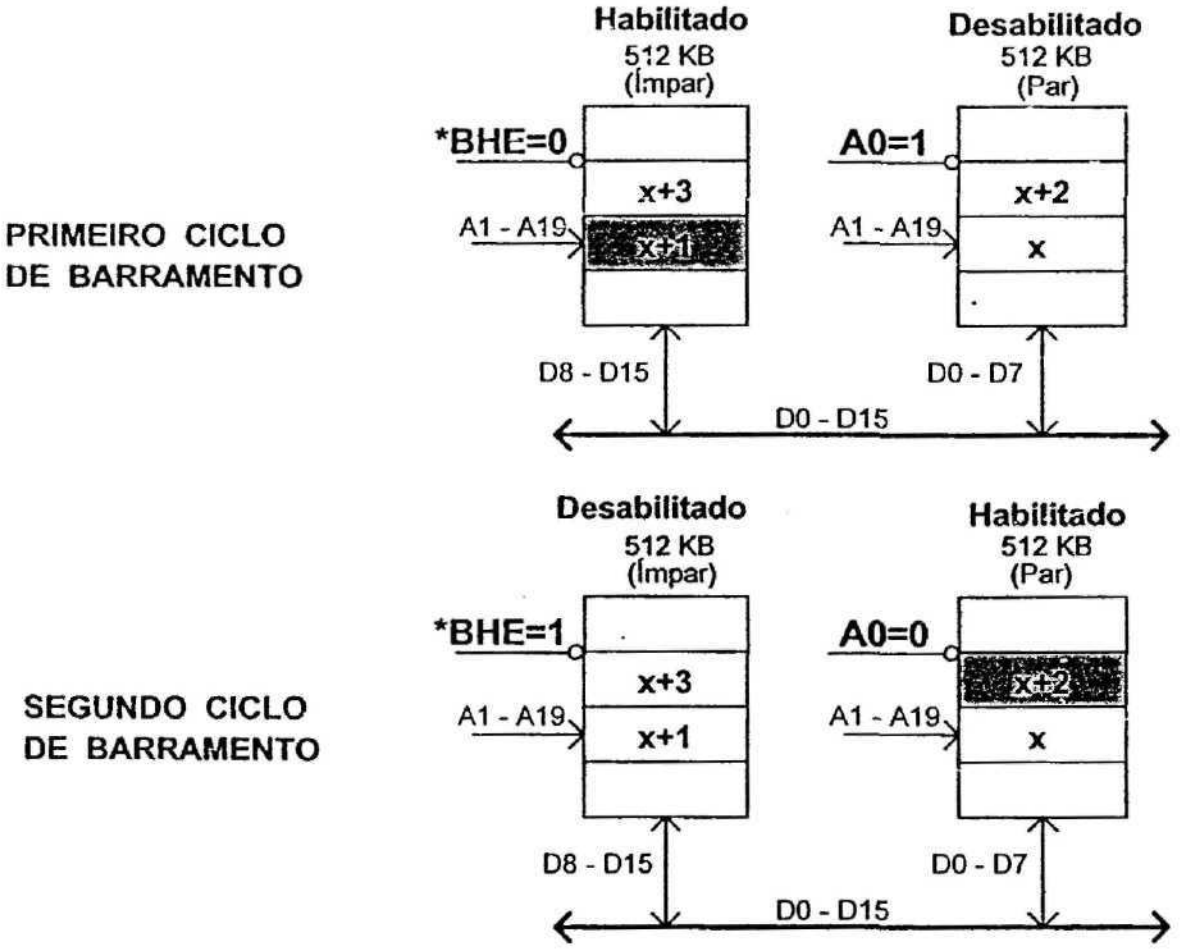


Figura 2.19. Acesso a uma palavra de 16 bits em endereço ímpar (x+1).

No caso da figura 2.19, a CPU 8086 automaticamente conecta as partes alta e baixa dos registradores de 16 bits às metades apropriadas do barramento de dados. Entretanto, acessar palavras de 16 bits em endereços ímpares requer um ciclo extra de barramento e isto degrada o desempenho do sistema. Portanto, todos os compiladores oferecem a opção de iniciar em endereço par qualquer variável de mais de 8 bits. Já a CPU 8088, como será visto adiante, por ter um barramento externo de dados de 8 bits, sempre necessita de dois ciclos para acessar palavras de 16 bits. Para ela, não importa se o endereço é par ou ímpar.

Durante a leitura, todos os 16 bits do barramento de dados são colocados em alta impedância, a partir do período T2, mesmo quando se utiliza somente a metade do barramento. Isto simplifica a decodificação das memórias. Durante a escrita de um byte, o 8086 aciona as 16 linhas de dados do barramento, mas a informação na metade que não está sendo usada é indeterminada. O mesmo vale para as operações de I/O.

Exemplificando o que foi exposto nesta seção, citam-se a seguir instruções que acessam bytes e palavras de 16 bits em endereços pares ou ímpares.

`mov al,[2322h]`	Esta instrução busca, no endereço 2322h (par), um dado de 8 bits (pois AL é um registrador de 8 bits).
`mov al,[2323h]`	Esta instrução busca, no endereço 2323h (ímpar), um dado de 8 bits (pois AL é um registrador de 8 bits).
`mov ax,[2322h]`	Esta instrução busca, nos endereços 2322h (par) e 2323h (ímpar), um dado de 16 bits (pois AX é um registrador de 16 bits). Neste caso, o acesso é feito em um único ciclo de barramento.
`mov ax,[2323h]`	Esta instrução busca, nos endereços 2323h (ímpar) e 2324h (par), um dado de 16 bits (pois AX é um registrador de 16 bits). Neste caso, o acesso é feito em dois ciclos de barramento.

Obs: é suposto que o registrador DS esteja zerado.

Comentário: Quando se constrói o banco de memória de uma CPU com barramento de dados de 16 bits, como o 8086, e que é capaz de acessar 1 mega endereços diferentes, sempre se pergunta por que não utilizar 1 mega palavras de 16 bits, ao invés de 1 megabyte ? Em outras palavras, por que não arrumar a memória em células de 16 bits, como exemplificado na figura 2.20. O grande problema é que os programas operam, na grande maioria das vezes, com bytes. Essa operação com bytes ficaria difícil em uma memória arrumada por palavras de 16 bits. Uma possível solução seria jogar fora metade da palavra de 16 bits, mas isso levaria a um grande desperdício. Não importa qual seja a CPU (16, 32 ou 64 bits), sempre haverá a necessidade de operar-se com bytes e, por isso, essas memórias são estruturadas por bytes.

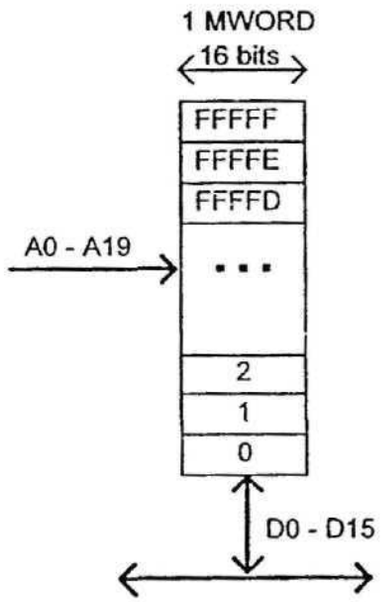


Figura 2.20. Organização em palavras de 16 bits dificultam o acesso a bytes.

Exercício Resolvido 2.4.2.1

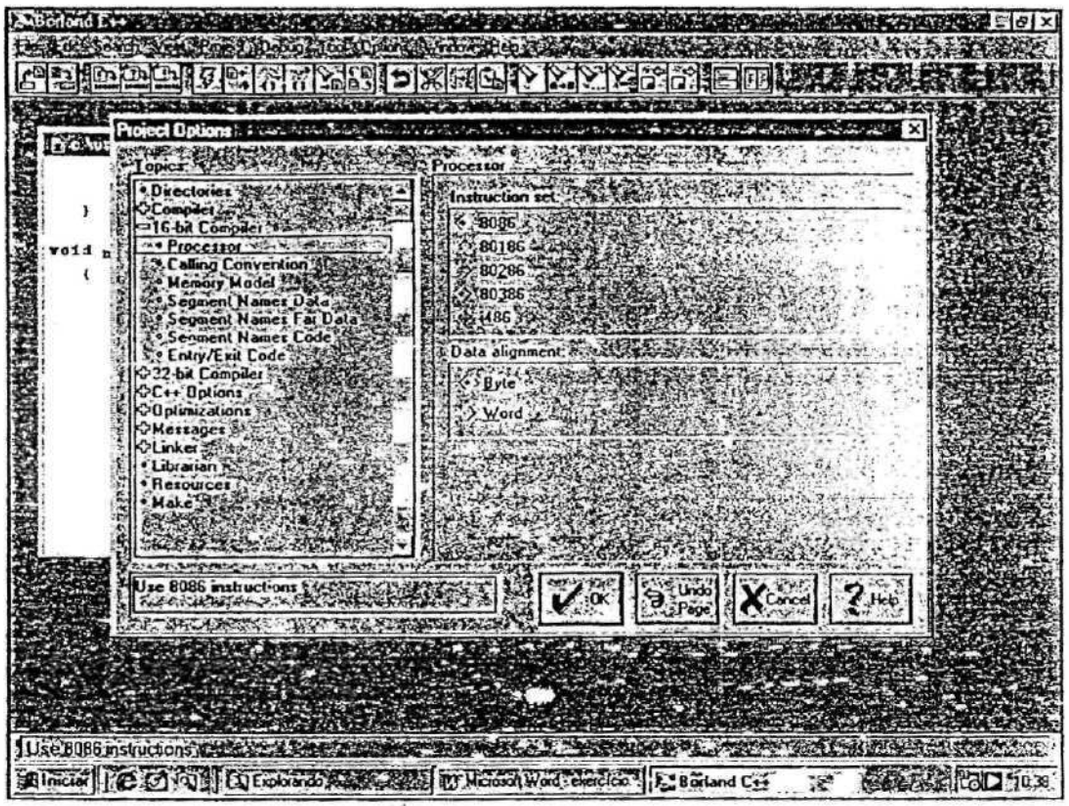
Supondo que se esteja usando um compilador para DOS, calcule o tamanho, em bytes, da seguinte estrutura em linguagem C:

```
struct exemplo
        {
             int a;
             char b;
             int c;
        };
```

Solução:

À primeira vista, a estrutura teria 5 bytes de tamanho: 2 bytes para cada inteiro e 1 byte para o caracter. Contudo, se isto ocorresse, um dos inteiros ficaria alocado a partir de um endereço ímpar, o que degradaria o desempenho da CPU quando fosse feito um acesso a ele.

Para evitar este problema de degradação de desempenho, os compiladores oferecem uma opção para permitir a alocação de variáveis de 16 bits apenas a partir de endereços pares, através do alinhamento. Como no exemplo da figura, um alinhamento por **byte** não garante o melhor desempenho, mas garante o menor tamanho da estrutura (5 bytes). Já o alinhamento por **word** faz com que todas as variáveis de 16 bits sejam alocadas em endereços pares, de forma que a estrutura passe a ter o tamanho de 6 bytes.



Escolha do alinhamento pelo compilador.

Exercício Resolvido 2.4.2.2

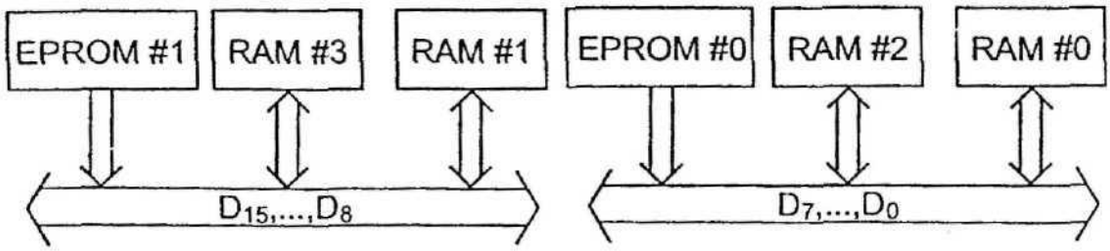
Conecte 4 memórias RAM 6116 (2 KB cada) e 2 memórias EPROM 2716 (2 KB cada) aos barramentos de dados e de endereços do 8086, de forma que elas constituam toda a memória de programa e de dados de um computador baseado neste 8086.

Solução:

Como o 8086 utiliza um barramento de dados de 16 bits, implementando assim um banco par e um banco ímpar, os *chips* de memória devem ser organizados de 2 em 2. Assim, uma opção é utilizar o seguinte mapeamento:

EPROM #0 e #1	FFFFFh FF000h	⇐	A19 = A18 = ... = A12 = 1
sem dispositivos mapeados			
RAM # 2 e #3	01FFFh 01000h	⇐	A19 = A18 = ... = A13 = 0 A12 = 1
RAM #0 e #1	00FFFh 00000h	⇐	A19 = A18 = ... = A13 = 0 A12 = 0

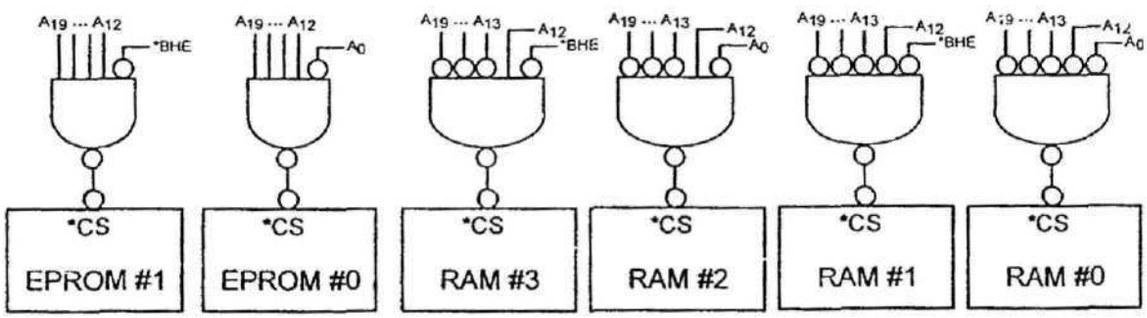
Mapeamento.



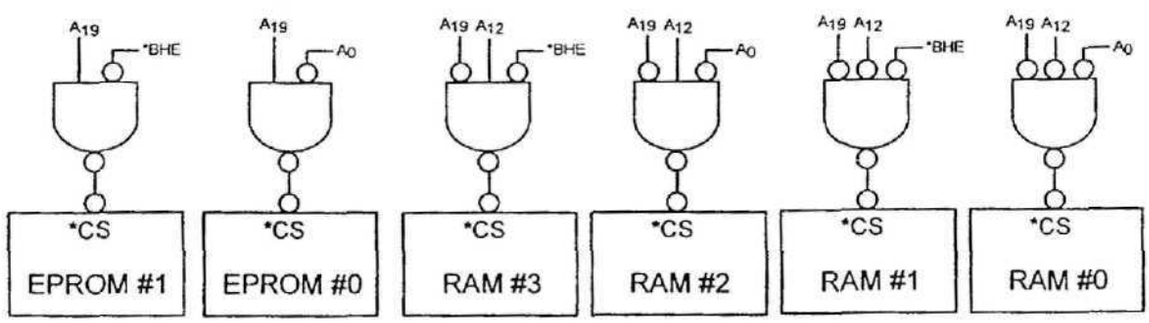
Conexão das memórias ao barramento de dados.

A EPROM, que deverá conter o programa de inicialização (BIOS), deve ocupar os últimos endereços de memória, pois a primeira instrução executada é a de endereço FFFF0h. Já as memórias RAM podem ocupar os primeiros endereços. Os *chips* de índice par vão conectados à parte baixa do barramento de dados ($D_7,...,D_0$), enquanto que os de índice ímpar vão conectados à parte alta ($D_{15},...,D_8$).

Repare que a coluna mais à direita da tabela de mapeamento ressalta os valores que algumas linhas de endereços devem ter para que o acesso àquela porção de memória seja logrado. Com base nesta coluna, pode-se então calcular a lógica que ativa o *Chip Select* de cada componente, como no esquema (a) a seguir. Já o esquema (b) usa um hardware menor, mas não admite a conexão futura de outras memórias.

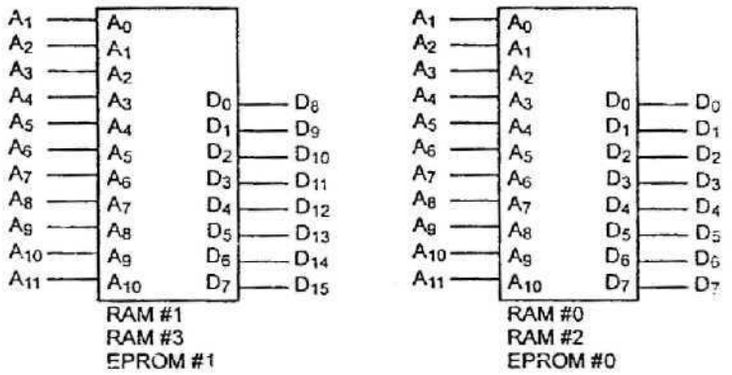


(a) Hardware expansível.



(b) Hardware mínimo.

Para finalizar, resta conectar as linhas de endereços. Como os *chips* mapeam endereços de 2 em 2, as linhas $A_{10}...A_0$ de cada memória são ligadas às linhas $A_{11}...A_1$ da CPU, como mostra a figura.



Conexão das memórias aos barramento de dados e de endereços.

2.4.3. Como Separar o Barramento de Dados

Com relação ao barramento de dados do sistema, duas alternativas de implementação devem ser consideradas:

- barramento de endereços e dados multiplexados (vide figura 2.21),
- barramento de dados com *buffers* ou *transceiveres* (vide figura 2.23).

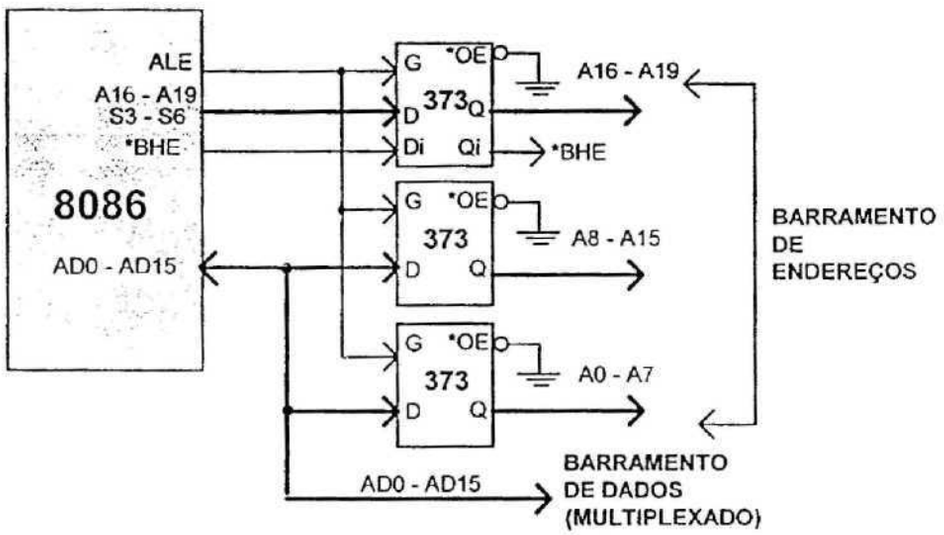


Figura 2.21. Barramento de dados multiplexado.

Em projetos pequenos, não existe problema em usar o barramento de dados multiplexado. É necessário apenas garantir que as memórias e os dispositivos de I/O, conectados a esse barramento, não interfiram com os endereços durante o período T1. Para ter-se essa garantia, é necessário usar dispositivos com três-estados na saída, que só são ativados pela linha de leitura (*RD) da CPU. Dispositivos que não possuem três-estados na saída só poderão ser conectados se for adicionado um circuito extra para suprir o terceiro estado necessário.

Uma outra limitação à utilização do barramento de dados multiplexado é relativa à capacidade de corrente do 8086, que é de 2 mA, e à carga capacitiva de 100 pF (para garantir as características AC). Sejam os seguintes valores típicos:

- dispositivo de I/O: 20 pF,
- *latch* de endereços: 12 pF,
- memória: 5 a 12 pF.

Um sistema com 3 periféricos e 2 a 3 memórias já estaria fora do limite especificado. Para permitir sistemas grandes, deve-se aumentar a capacidade de carga do 8086. Isto é conseguido com uma "amplificação" do barramento de dados. O 8086, em modo mínimo, gera dois sinais para controle dos *buffers*: ***DEN** e **DT/*R**. ***DEN** *(Data Enable)* indica a existência de tráfego de dados pelo barramento, enquanto que **DT/*R** *(Data TX or not RX)* indica a direção (o sentido) desse tráfego.

É muito comum o uso do transceiver 74LS245 para realizar essa função. A figura 2.22 apresenta um diagrama resumido deste CI, enquanto que a figura 2.23 ilustra o caso do barramento de dados "amplificado".

Como o sinal ***DEN**, no pior dos casos, é habilitado ao final do período T1, ele garante a contenção do barramento. Assim, durante T1, ninguém, exceto a CPU, estará colocando dados no barramento multiplexado.

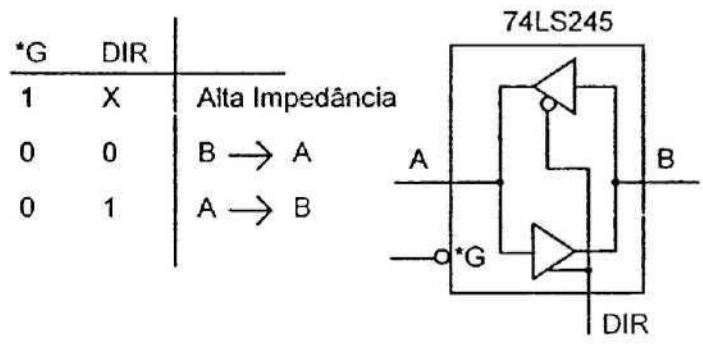


Figura 2.22. Transceiver 74LS245.

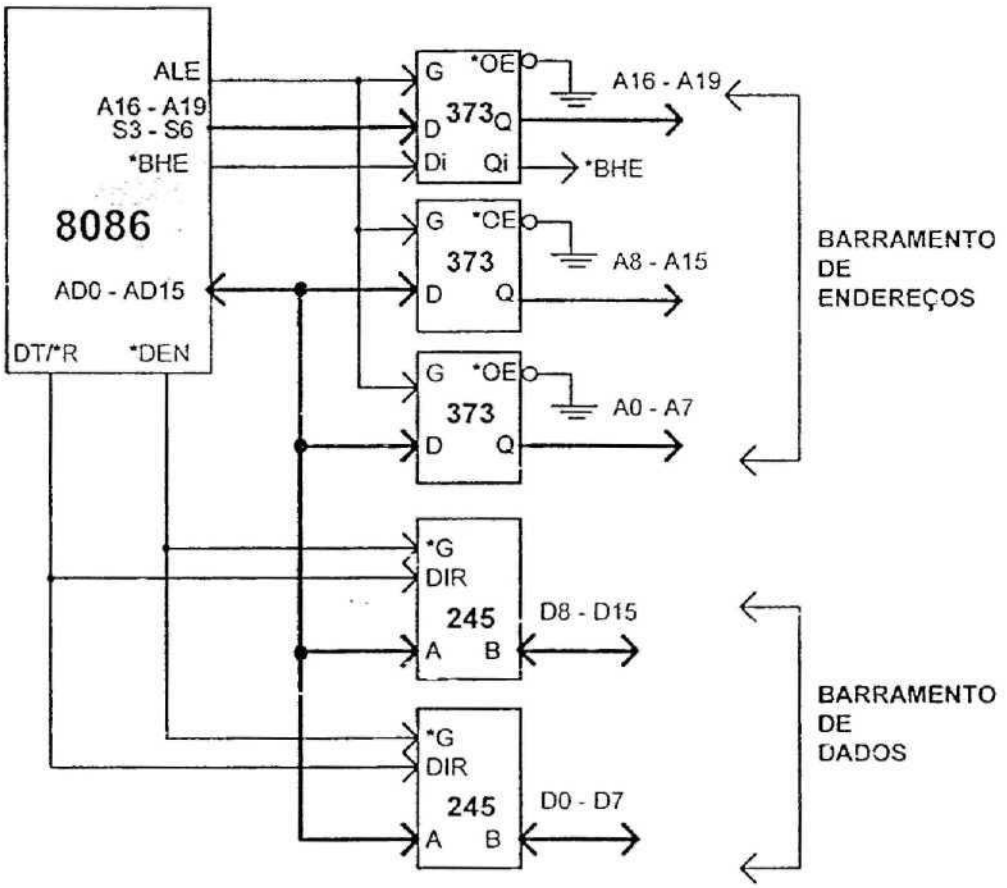


Figura 2.23. Barramento de dados "amplificado".

2.5. Projeto Modo Mínimo

No modo mínimo, o pino **MN/*MX** deve estar conectado a Vcc. Este modo deve ser usado para sistemas com uma única CPU. Pode-se endereçar 1 MB de memória e 64 K dispositivos de I/O. O barramento de dados é de 16 bits. Os sinais para controle do barramento (DT/*R, *DEN, ALE, **M/*IO**, ***RD**, ***WR**, ***INTA**) são fornecidos pela própria CPU. Um mecanismo de *hold* e *hlda* permite compatibilidade com os controladores de DMA existentes. Uma CPU 8086 em modo mínimo está mostrada na figura 2.24.

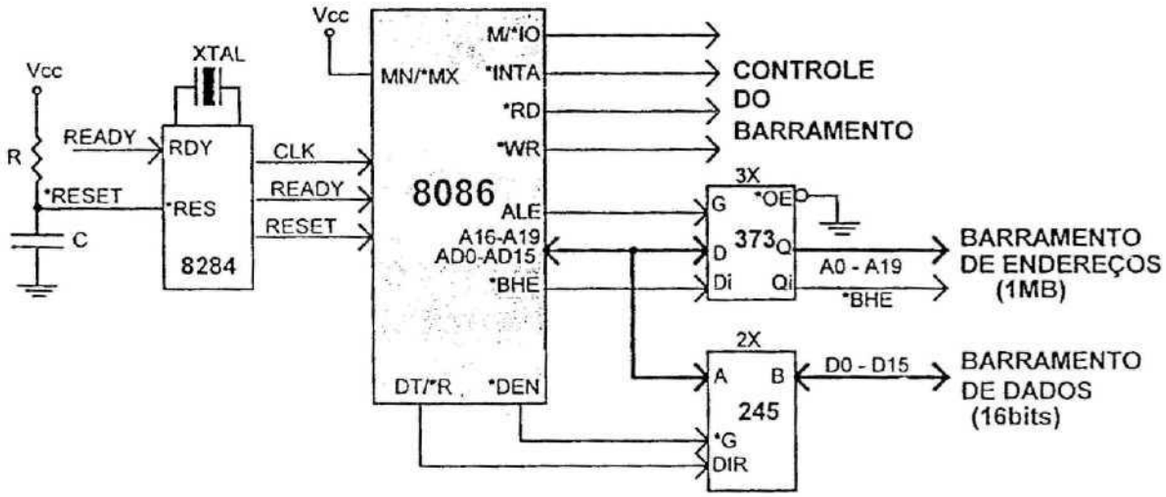


Figura 2.24. Projeto Modo Mínimo.

2.6. Projeto Modo Máximo

No modo máximo, o pino **MN/*MX** deve estar conectado à terra. O modo máximo é usado para sistemas multiprocessados ou coprocessados. Neste modo, o 8288 (controlador de barramento) recebe sinais de estado do 8086 que, decodificados, geram os sinais de controle do barramento. Como já foi visto, alguns pinos mudam de função quando em modo máximo:

- o estado da fila interna é informado por **QS0** e **QS1**, permitindo que outros processadores acompanhem a execução da CPU; o estado da fila é válido durante o período de relógio seguinte à operação (vide figura 2.10);
- o estado do barramento é obtido com ***S0**, ***S1** e ***S2** (vide figura 2.8);
- existe o mecanismo de *lock*, ou tranca, do barramento, para permitir o controle de recursos compartilhados em sistemas multiprocessados;
- dois controles priorizados (***RQ/*GT0** e ***RQ/*GT1**) permitem que diversas CPU compartilhem o mesmo barramento.

Os sinais ***S0**, ***S1**, ***S2** são fornecidos ao 8288 de forma a gerar os sinais de controle do barramento. O 8288 amostra as linhas de estado no início de cada período de relógio (↓) da CPU. No início de um ciclo de barramento, a CPU leva as linhas do estado ocioso (***S0** = ***S1** = ***S2** = 1) para um dos sete estados possíveis. O 8288, então, inicia um ciclo de barramento, gerando o sinal **ALE**, acompanhado pelos controles de direção do *buffer*. A figura 2.25 apresenta um exemplo de uma 8086 no modo máximo.

O 8288 gera os seguinte sinais de controle:

- **ALE** *Address Latch Enable*
- **DEN** *Data Enable*
- **DT/*R** *Data Transmite or Receive*
- ***INTA** *Interrupt Acknowledge*
- ***MRDC** *Memory Read Command*
- ***MWTC** *Memory write command*
- ***IORC** *I/O Read Command*
- ***IOWC** *I/O Write Command*
- ***AMWC** *Advanced Memory Write Command*
- ***AIOWC** *Advanced I/O Write Command*

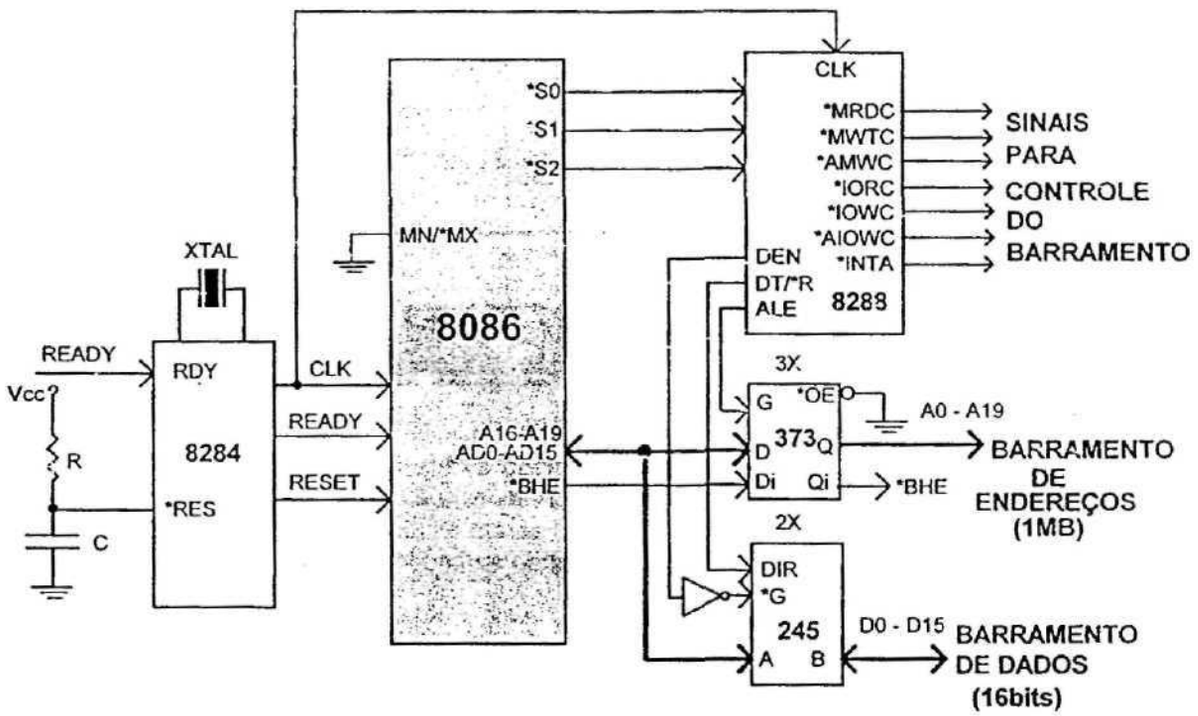


Figura 2.25. Projeto Modo Máximo.

2.7. Unidade de Execução (EU) e Unidade de Interface com o Barramento (BIU)

Um dado muito importante para compreender o funcionamento do 8086 é o fato da lógica de controle do barramento ser separada da lógica de execução. Isto significa que o 8086 tem duas entidades independentes: uma Unidade de Execução (EU) e uma Unidade de Interface com Barramento (BIU).