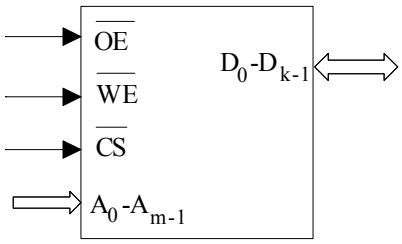
IV - Memorias (2 semanas) v1.3

4.1 Aspectos básicos

La figura muestra las señales básicas de una memoria.



A_0-A_{m-1} : Líneas de direcciones
D_0-D_{k-1}: Líneas de Datos

El número **m** de líneas de direcciones indica el número de posiciones que es 2^n.
Por encima de 2^{10} son k. De ello $2^{16}=2^6 \cdot 2^{10}=64k$.
Por encima de 2^{20} son M. De ello $2^{22}=2^2 \cdot 2^{20}=4M$.

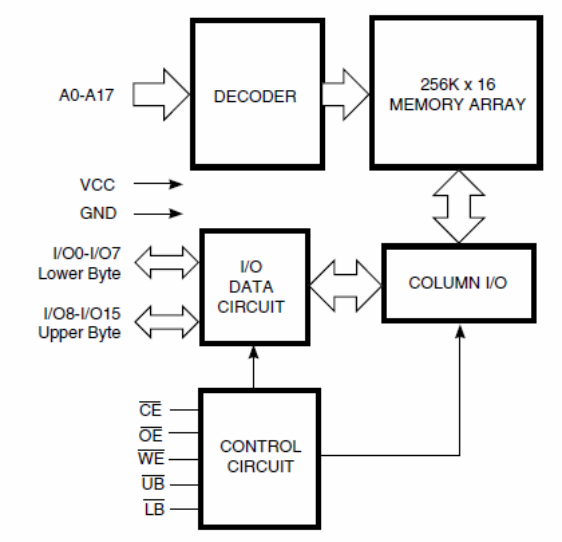
El número **k** de líneas de datos indica el acho de los datos.
La referencia es el byte (8 bits) por eso lo normal es $k=8 \cdot 2^x$.
Podría ser 8 (byte), 16 (word), 32 (dword), pero también 4 (nibble), 2, 1 (bit) tomando x como entero negativo.

La conexión de las direcciones y datos de la memoria no tiene porqué ser directa con los buses de datos y direcciones de la CPU.
Por ejemplo se pueden conectar 8 memorias de 1 bit a los 8 bits del bus de datos de una CPU, o se pueden juntar varios módulos de memoria que ocupan rangos de direcciones distintos del bus de direcciones. Eso se explica en el apartado de decodificación de direcciones.

Señales de control (todas activas bajas):

CS* (Chip Select) Activa el chip de memoria de manera general (también CE*).
OE* (Output Enable) Indica que se puede volcar un dato en el bus de datos.
WE* (Write Enable) Indica que se realiza una operación de escritura (RAM).

Cuando el bus de direcciones es de más de un byte (word, dword, ...) se añaden señales adicionales de control para posibilitar el acceso a bytes individuales tal y como ocurre en la memoria IS61LV25616 usada en prácticas (*UB, *LB).



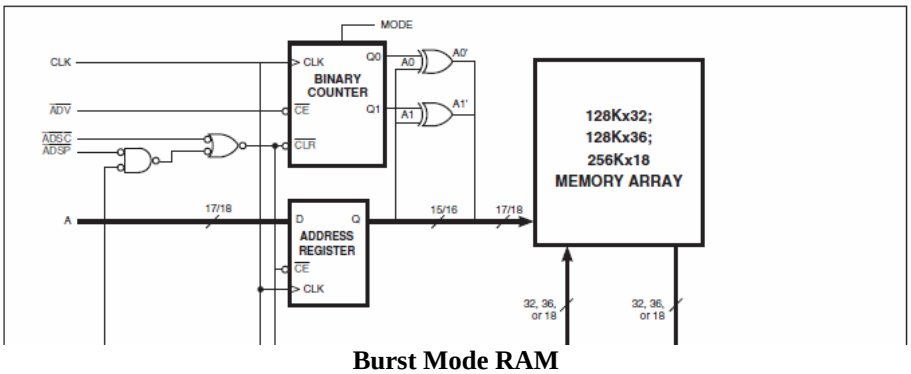
IS61LV25616 Diagrama de bloques

Obsérvese que en esta memoria las direcciones A_0 - A_{17} son en words, no en bytes, a diferencia de lo que pasaba en el µP 68000 que usaba A_1 - A_{23} para referenciar words.
Adicionalmente, en esta memoria, OE puede ser indeterminado en ciclos de escritura por lo que OE* sólo indica que hay ciclo de lectura si WE* está inactivo.*

Las memorias son típicamente asíncronas. No obstante existen memorias que tienen entrada de reloj y que admiten modos asíncronos como el funcionamiento en modo ***burst*** en el que se hacen 4 lecturas consecutivas sin cambiar el bus de direcciones.

IS61(64)LPS12832A
IS61(64)LPS12836A IS61(64)VPS12836A
IS61(64)LPS25618A IS61(64)VPS25618A

BLOCK DIAGRAM



Burst Mode RAM

Obsérvese que existen memorias de 128k x 32 (4 bytes), 128k x 36 (4,5 bytes) y 256k x 18 (3 bytes). Las memorias no siempre se organizan en bytes, words o dwords.

Clasificación de memorias

RAM (Random Access Memory)

Permiten lectura y escritura de datos.

Estáticas (SRAM)

Emplean biestables para almacenar la información, por tanto, ésta dura hasta que se desactiva la alimentación.

Dinámicas (DRAM)

Emplean condensadores MOSFET para almacenar la información.
Ocupan menos en cada celda por lo que son más densas y son, por tanto, preferidas para aplicaciones que demandan mucha memoria como son los PC ya que el precio por bit es mucho menor que las SRAM.
A parte de necesitar alimentación requieren de refresco de su información cada cierto tiempo. También son más lentas que las estáticas.
Suelen tener muchas líneas de direcciones y, en muchos casos, un único bit de datos.
Para reduir el número de pines se multiplexa temporalmente el bus de direcciones con dos señales RAS y CAS. Usando este multiplexado, una memoria de 64Kb (2^{16}) puede tener 8 filas y 8 columnas por tanto le bastarían 8 pines para el bus de direcciones en lugar de 16.

NVRAM

RAM especial estática no volátil porque lleva una batería o una EEPROM de respaldo.

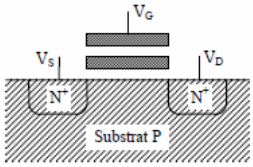
Memorias no volátiles

Mask ROM

Memoria con un contenido fijado en el proceso de fabricación.
Tiene un elevado coste fijo y un bajo coste por unidad por lo que sólo se usa en aplicaciones masivas que no requieran actualizaciones durante su vida útil.

EPROM (Erasable Programable ROM)

Memoria que emplea transistores de doble puerta (una flotante) para programar el estado de las celdas.

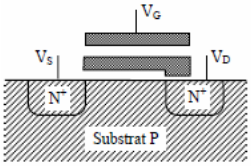


Una tensión elevada y el paso de corriente por el canal dan lugar a la aparición de electrones de alta energía que atraviesan el óxido y quedan atrapados en la puerta flotante. Ello hace que la V_T del transistor aumente (campo opuesto a Vg) y no pueda ser activado con tensiones normales de puerta.

La radiación ultravioleta (1/2 hora) elimina los electrones de la puerta y borra la memoria (El transistor vuelve a responder a la tensión de puerta).
El encapsulado es caro al ser cerámico y requerir una ventana para el borrado (que se suele tapar, en funcionamiento normal, para evitar borrados accidentales).

OTP PROM (One Time Programmable ROM)
Memoria que posee fusibles internos y que se puede programar una única vez. Muchas veces se realiza con una EPROM en encapsulado plástico sin ventana.

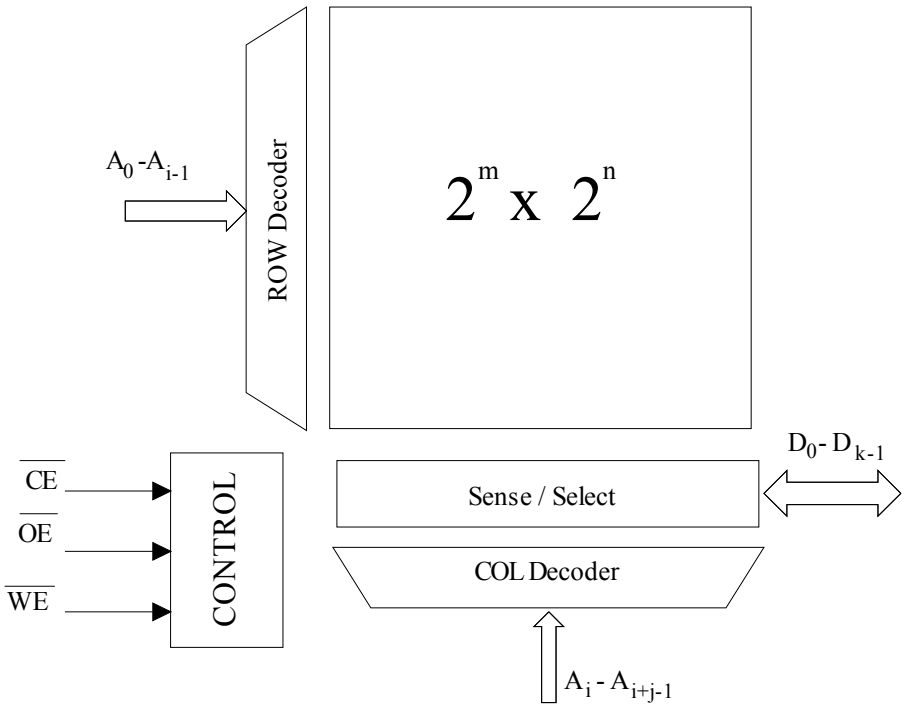
EEPROM (Electrically Erasable Programmable ROM)
Emplean también una puerta flotante, pero la carga y descarga de ésta se realiza por efecto túnel. En escritura son más lentas que las RAM y tienen mas limitado el número de ciclos de escritura.



Las memorias FLASH son una evolución de las memorias EEPROM.

4.2 Arquitectura básica

Para una RAM típica de 1bit, las celdas se agruparían en filas y columnas repartiendo las direcciones (m=i+j). Se intenta que el reparto sea homogéneo (i≈j) para evitar que un decodificador sea tenga demasiadas líneas de salida y el retardo no sea similar. En las DRAM se habría de añadir un LATCH de filas y columnas para el multiplexado de filas y columnas con las señales RAS / CAS.

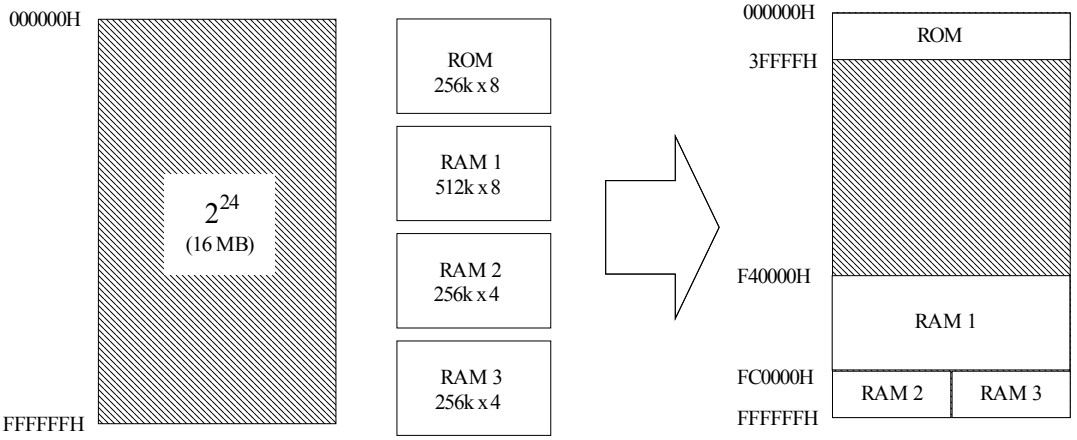


Dependiendo del caso, CE* controla o no la activación de los decodificadores y el resto de circuitos periféricos.

Dependiendo de la RAM, puede existir un acceso distinto para los datos de entrada y salida (Din, Dout) en lugar de ser un acceso bidireccional.

4.3 Decodificación

En un sistema basado en μP/μC puede haber un conjunto de distintos chips de memoria. Cada chip de memoria debe estar asociado a distintas partes del mapa de memoria del μP/μC.
La circuitería de decodificación es la responsable de activar adecuadamente las señales CS* (o CE*) de cada chip de memoria de manera que cada chip quede correctamente asociado a su región asignada en el mapa de direcciones.



La misión de la circuitería de decodificación es ubicar todos los chips de memoria dentro del mapa de memoria del μP.

Sobre las direcciones hexadecimales

Hay 24 líneas de direcciones. $2^{24} = 2^{6x4}$ Ello implica que hay 6 dígitos hexadecimales (un dígito son 4 bits). El espacio de direcciones va, por tanto, de 000000H a FFFFFFH.

Si la ROM de 256kB se pone al principio ello implica que irá de 0 a 256k-1.
$256k = 2^8 \cdot 2^{10} = 2^{18} = 2^2 \cdot 2^{16} = 4 \cdot 2^{4x4} = 40000H$. Por tanto de 0H a 3FFFFH.

RAM 2 y RAM 3 se ponen al final, por tanto llegarán a FFFFFFH, la dirección inicial será
$2^{24} - 256k = 2^{24} - 2^8 \cdot 2^{10} = 2^{24} - 2^{18} = (2^8 - 2^2)\, 2^{16} = (256-4)\, 2^{16} = 252 \cdot 2^{16} = FCH \cdot 10000H = FC0000H$

RAM 1 empieza 512k antes que RAM 2/3, por tanto, la dirección inicial será:
$252 \cdot 2^{16} - 512k = 252 \cdot 2^{16} - 2^9 \cdot 2^{10} = (252 - 2^3) 2^{16} = 244 \cdot 2^{16} = F4H \cdot F10000H = F40000H$

Un procedimiento para convertir entre las bases decimal y hexadecimal aparece en el apéndice A de este capítulo.

En general hay algunos criterios que suele ser interesante cumplir:

- Vectores de interrupción preferentemente en RAM
- Dirección (o vector) de reset preferentemente en ROM
- Mapeados continuos de memoria para RAM/ROM

Para generar la señal CS* de los chips de RAM suele usarse alguno de los Strobes (AS*, UDS*, LDS*, MREQ*) que proporciona el μP/μC para garantizar que los chips de memoria sólo están activos cuando realmente se hace un acceso a memoria.

Decodificación para bus de datos de 1 byte (8 bits) y una Memoria

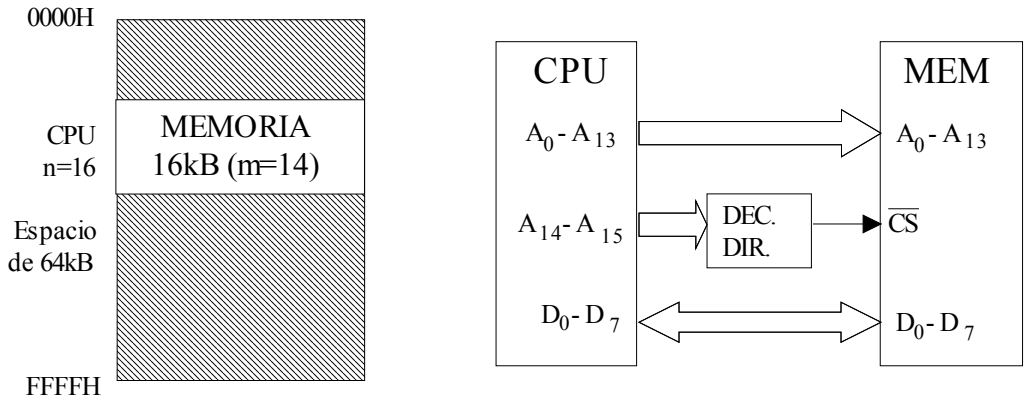
Si el bus de datos de la CPU y de las memorias es de 8 bits, se unen entre sí todas las líneas D_0 a D_7 de CPU y memorias.
Lo que queda es resolver las direcciones.

Supongamos que la CPU tiene $_n$ líneas de direcciones $A_0...A_{n-1}$.
La memoria deberá tener un número de líneas **m** menor o igual $A_0...A_{m-1}$.

Si **m** es menor que **n**, la memoria no llenará todo el espacio de direcciones de la CPU. Eso se representa en el mapa de memoria de la CPU que muestra todo su posible rango de direccionamiento.

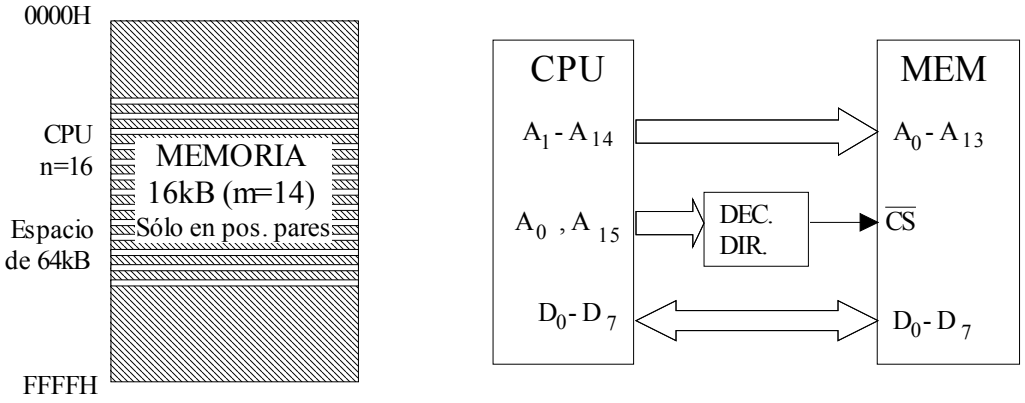
A la hora de decodificar, el hecho de que deseamos que la memoria sea contigua hace que un chip de memoria con **m** líneas de direcciones (2^m posiciones), deba emplear las direcciones de la CPU desde A_0 hasta A_{m-1} como líneas de direcciones.

Las líneas de la CPU desde m en adelante se usarán para generar CS* de manera que sólo una combinación de estas líneas genere un señal CS* baja.



Ejemplo de CPU capaz de direccionar 65kB (A_0-A_{15}) conectada a una memoria de 16kB (A_0-A_{13}) Las líneas A_{14} y A_{15} de la CPU se usan para general la señal CS*.

Si no se conectaran las direcciones A_0-A_m de CPU a las direcciones A_0-A_m de la memoria, el espacio de memoria estaría fragmentado como en el caso siguiente.



Ejemplo de memoria fragmentada debido a que A_0 de la CPU se ha empleado en la generación de CS* en lugar de conectarla a A_0 de la memoria.

Queda por resolver como se genera el CS* a partir de las líneas restantes del bus de direcciones. La solución más sencilla y habitual es colocar el inicio de la memoria en una posición del mapa de direcciones que sea múltiplo de su tamaño.

Si la CPU tiene n líneas de direcciones y la memoria tiene m:

El espacio total de direcciones es 2^n bytes. ($2^{16}=2^6 \cdot 2^{10}$=64k en el ejemplo)
La memoria es de 2^m bytes. ($2^{14}=2^4 \cdot 2^{10}$=16k en el ejemplo)

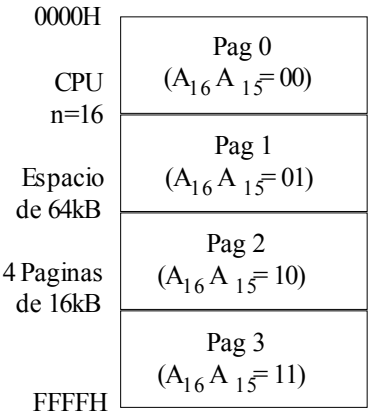
Podemos dividir el espacio de direcciones en **p** páginas donde $p=2^{n-m}$ cada una de las cuales será justamente del tamaño de la memoria. ($p=2^{16-14}=2^2=4$).

Cada página **p** empieza en $p \cdot 2^m$ y acaba antes de empezar la siguiente en $(p+1) \cdot 2^m - 1$.

A15	A14	$A_{13}...A_0$	CPU
Página		$A_{13}...A_0$	Memoria

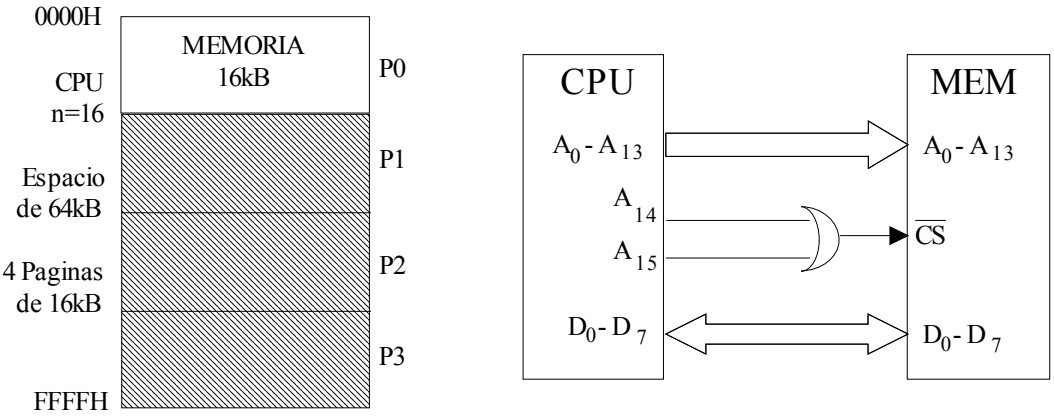
Tabla de Decodificación

La memoria se puede situar en cualquiera de estas páginas eligiendo la combinación adecuada de las líneas de direcciones no usadas directamente en la memoria.



Páginas seleccionables por la circuitería de decodificación.

Si deseamos poner la memoria en la página 0:



Ejemplo de decodificación que sitúa la memoria de 16kB al principio del espacio de direcciones de la CPU (Página 0).
Strobes no mostrados por simplicidad.

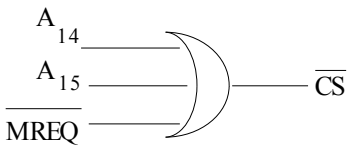
La tabla de direcciones reflejará esta asigación dado que A_{15} y A_{14} han de valer 0 (página 0) para que se active CS* en la memoria.

A15	A14	$A_{13}...A_0$	CPU
0	0	$A_{13}...A_0$	Memoria

Tabla de Decodificación

Negando o no cada una de las líneas A14 y A15 antes de la puerta OR (AND en lógica negada), podemos poner la memoria en cualquiera de las 4 páginas posibles.

La adición de ***strobes*** a la lógica de decodificación de memoria no se ha puesto por simplicidad, pero podría ser como se muestra:



En este caso CS* se activará únicamente cuando A_{14}=A_{15}=0 y MREQ* se halle activa (nivel bajo).

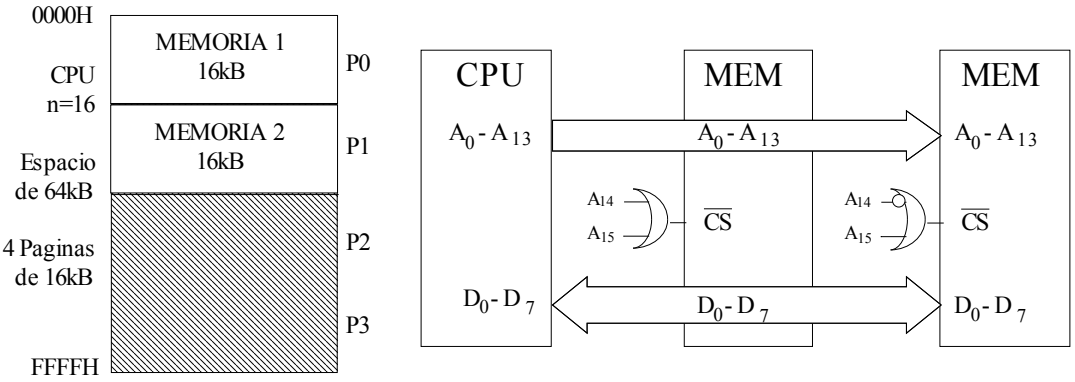
Decodificación para bus de datos de 1 byte (8 bits) y varias Memorias

Cuando tenemos varias memorias, el procedimiento es el mismo que para una memoria pero repitiéndolo en cada una de las memorias.
Lo único que se ha de vigilar es que los espacios de los diferentes chips de memoria no se solapen entre si.

Si, por ejemplo, hemos de poner 2 chips de memoria de 16kB es un mapa de memoria de CPU de 64kB que tiene soporte para 4 páginas de 16kB, bastaría asociar cada chip de memoria a una página distinta.

A15	A14	$A_{13}...A_0$	CPU	Rango
0	0	$A_{13}...A_0$	Memoria 1	0 a 2^{14}-1
0	1	$A_{13}...A_0$	Memoria 2	2^{14} a $2\cdot2^{14}$-1

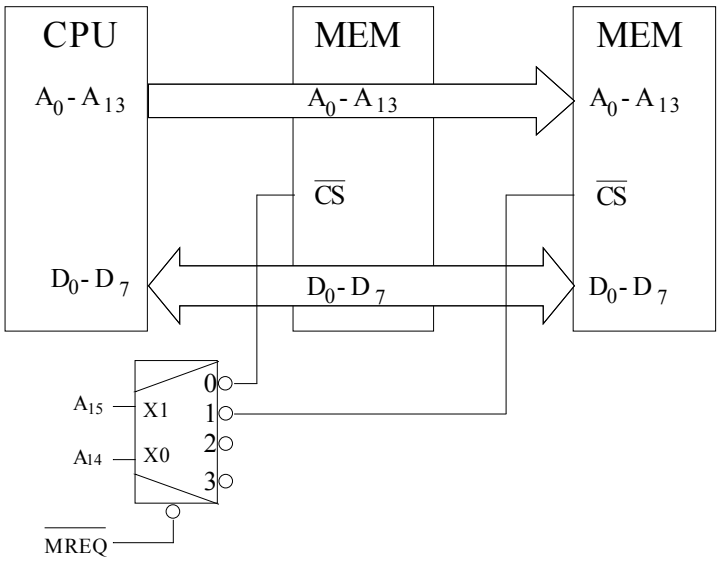
Tabla de Decodificación para dos memorias



Ejemplo de decodificación que sitúa dos memorias de 16kB al principio del espacio de direcciones de la CPU (Páginas 0 y 1).
Strobes no mostrados por simplicidad.

En el ejemplo anterior las líneas A_{14} y A_{15} se han decodificado de manera independiente en cada memoria. Lo normal es agrupar la lógica usando decodificadores.

En efecto, si juntamos damos A_{14} y A_{15} como entrada de un decodificador binario de 2 a 4, las 4 salidas serán las correspondientes a las 4 páginas.
Dado que las señales CS* son de lógica negada, emplearemos un decodificador con salidas negadas.
Adicionalmente, usaremos un decodificador con entrada de habilitación para añadir la línea de *strobe* usada para acceder a la memoria.



Ejemplo de uso de decodificadores.

MREQ*	A15	A14	O_3*	O_2*	O_1*	O_0*
H	X	X	H	H	H	H
L	0	0	H	H	H	L
L	0	1	H	H	L	H
L	1	0	H	L	H	H
L	1	1	L	H	H	H

Tabla de verdad del decodificador

Cuando tenemos memorias del mismo tamaño, podemos agrupar un conjunto de ellas que sea potencia de 2 de manera que accesos a posiciones consecutivas den lugar a accesos a chips distintos de memoria.
En el caso de chips RAM puede mejorar la velocidad de acceso a direcciones consecutivas. En el caso de ROM no suele ser conveniente porque obliga a programar los chips con valores que no son contiguos en el mapa de memoria.

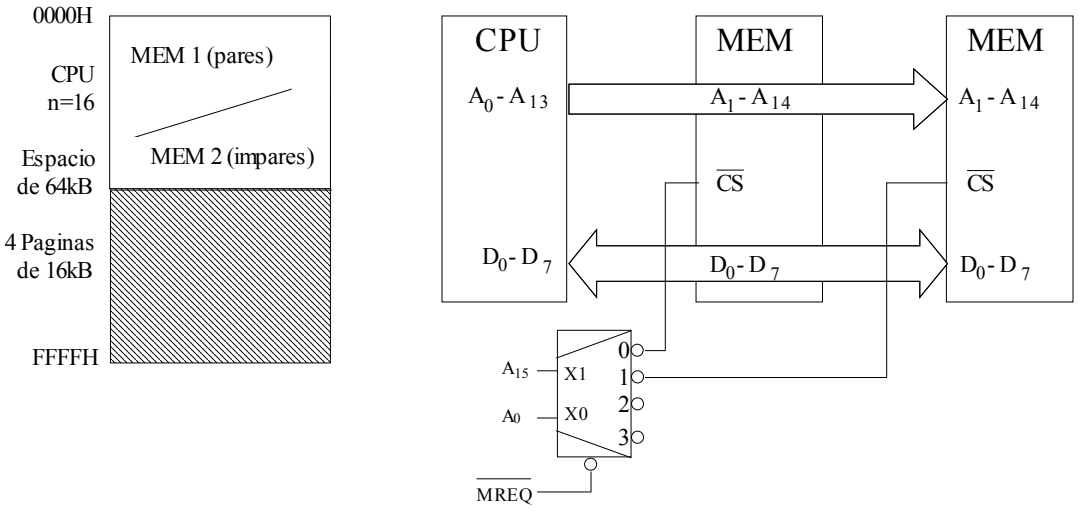
A modo de ejemplo, usando las 2 memorias anteriores se podrían agrupar con lo que resultaría:

A_{15}	$A_{14}...A_1$	A_0	CPU	Rango
0	$A_{13}...A_0$	0	Memoria 1	0 a 2^{15}-2 pares (A_0=0)
0	$A_{13}...A_0$	1	Memoria 2	0 a 2^{15}-1 impares (A_0=1)

Tabla de decodificación de 2 chips memoria con direcciones intercaladas

Lo que es fundamental es que el conjunto de memorias que ocupa 2·16k = 32kB = 2^{15} asigne todas las posibles combinaciones de los bits A_0 a A_{14} de manera que no haya huecos en el espacio total asignado de 32kB.

El mapa de memoria y el conexionado resultante sería:



Ejemplo de decodificación para memorias intercaladas.

Con 4 memorias las combinaciones posibles serían algunas más dependiendo de si las ponemos consecutivas o intercaladas:

A15	A14	$A_{13}...A_0$	CPU	Rango
0	0	$A_{13}...A_0$	Memoria 1	0 a 2^{14}-1
0	1	$A_{13}...A_0$	Memoria 2	2^{14} a $2\cdot2^{14}$-1
1	0	$A_{13}...A_0$	Memoria 3	$2\cdot2^{14}$ a $3\cdot2^{14}$-1
1	1	$A_{13}...A_0$	Memoria 4	$3\cdot2^{14}$ a $4\cdot2^{14}$-1

Tabla de decodificación 1: 4 Memorias consecutivas

A_{15}	$A_{14}...A_1$	A_0	CPU	Rango
0	$A_{13}...A_0$	0	Memoria 1	0 a 2^{15}-1 pares (A_0=0)
0	$A_{13}...A_0$	1	Memoria 2	0 a 2^{15}-1 impares (A_0=1)
1	$A_{13}...A_0$	0	Memoria 3	2^{15} a 2^{16}-1 pares (A_0=0)
1	$A_{13}...A_0$	1	Memoria 4	2^{15} a 2^{16}-1 impares (A_0=1)

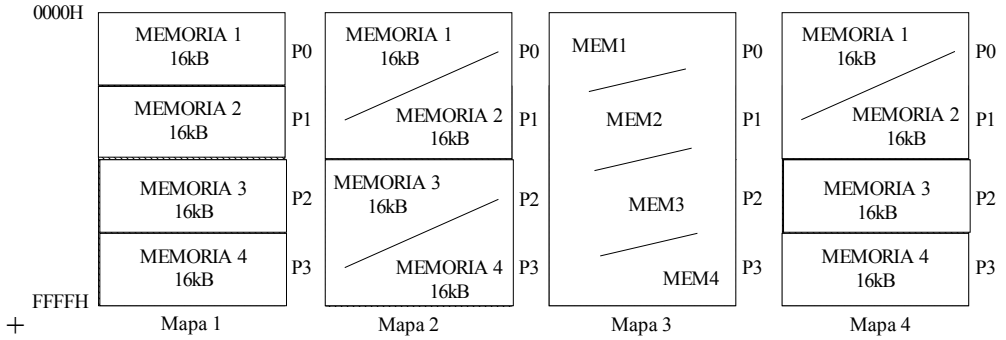
Tabla de decodificación 2: Memorias intercaladas 2 a 2

$A_{15}...A_2$	A_1	A_0	CPU	Rango
$A_{13}...A_0$	0	0	Memoria 1	0 a 2^{15}-1 (A_0,A_1=00)
$A_{13}...A_0$	0	1	Memoria 2	0 a 2^{15}-1 (A_0,A_1=01)
$A_{13}...A_0$	1	0	Memoria 3	0 a 2^{15}-1 (A_0,A_1=10)
$A_{13}...A_0$	1	1	Memoria 4	0 a 2^{15}-1 (A_0,A_1=11)

Tabla de decodificación 3: 4 Memorias intercaladas

A_{15}	A_{14}	$A_{13}...A_1$	A_0	CPU	Rango
0	$A_{13}...A_0$		0	Memoria 1	0 a 2^{15}-2 pares (A_0=0)
0	$A_{13}...A_0$		1	Memoria 2	1 a 2^{15}-1 impares (A_0=1)
1	0	$A_{13}...A_0$		Memoria 3	$2 \cdot 2^{14}$ a $3 \cdot 2^{14}$-1
1	1	$A_{13}...A_0$		Memoria 4	$3 \cdot 2^{14}$ a $4 \cdot 2^{14}$-1

Tabla de decodificación 4: 2 Intercaladas y 2 Consecutivas



Mapas de memoria correspondientes a las tablas anteriores

Caso de memorias de distinto tamaño

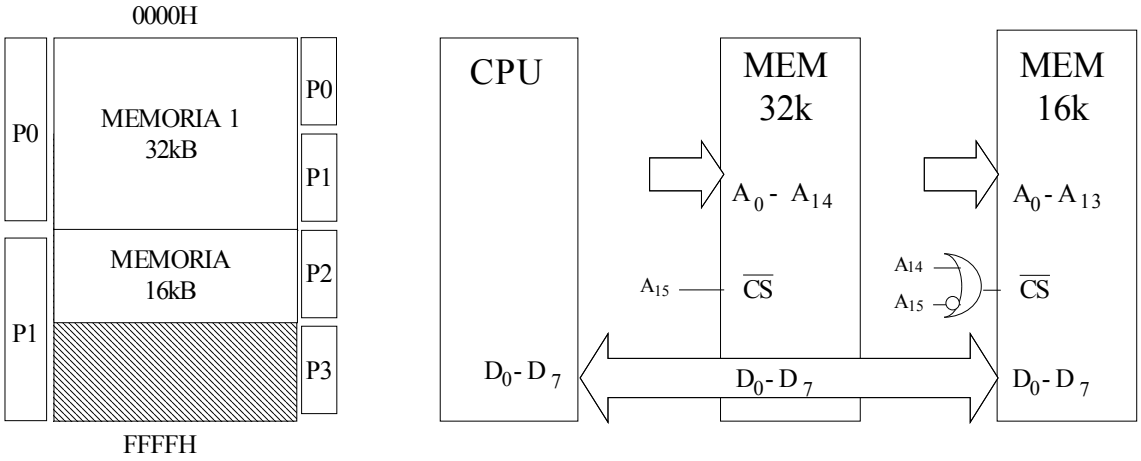
La decodificación para memorias de distinto tamaño es similar.

Consideremos, por ejemplo, el caso en que deseamos poner una memoria de 16kB y otra de 32kB conectadas a una CPU con un espacio de direcciones de 64kB.

La CPU tendrá 4 páginas de 16kB. Situaremos la memoria de 16kB en una de ellas. Adicionalmente la CPU tendrá 2 páginas de 32kB. Situaremos la memoria de 32kB en una de ellas.
Se habrá de verificar que no se den solapamientos, ninguna dirección puede ser asignada a ambos chips de memoria ya que ello daría lugar a una colisión en el bus de datos.

Elegimos poner la memoria de 32kB en la primera de las 2 páginas de 32 kB (P_0^{32k}). También elegimos poner la memoria de 16kB en la tercera de las 4 páginas de 16kB (P_2^{16k}).
De esta manera la memoria total es continua.

Si hubiéramos puesto la memoria de 16kB en la página P_0^{16k} o en la P_3^{16k} no nos sería posible situar la memoria de 32k sin tener un hueco en el mapa de memoria.



Ejemplo de decodificación para dos memorias de distinto tamaño

La tabla de uso de direcciones sería:

A_{15}	A_{14}	$A_{13}...A_0$	CPU	Rango
0	$A_{14}...A_0$		Mem 32k	0 a 2^{15}-1
1	0	$A_{13}...A_0$	Mem 16k	$2 \cdot 2^{14}$ a $3 \cdot 2^{14}$-1

Ejemplo de tabla de decodificación para dos memorias de distinto tamaño

Nuevamente pueden usarse decodificadores. La manera mas fácil, en este caso, es decodificar las 4 páginas de 16kB. La conexión de la memoria de 16kB es directa conectándola a la salida de la página elegida (P_2^{16k}).
Para la memoria de 32kB, la señal CS* no puede depender de la línea A_{14} que entra en el decodificador.
Dos posibles opciones son:

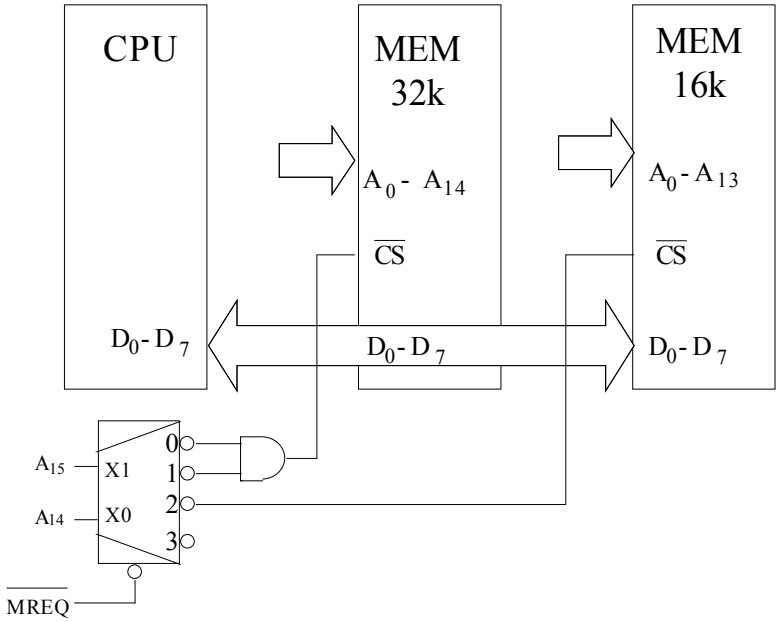
A) Conectar A15 a la entrada CS* de la memoria de 32kB igual que en la figura anterior.

B) Elegir las dos salidas del decodificador asociadas a la página 0 de 32kB (corresponden a las páginas 0 y 1 de 16kB).

La figura siguiente muestra la realización si se opta por la opción B.
Obsérvese que se usa una puerta AND que corresponde a la función OR en lógica negada.
Probablemente en este caso concreto sea mejor la opción A, ya que evita el uso de la puerta AND.

Ejemplo de decodificación no standard

Uno podría pensar en la posibilidad de poner la memoria de 16k en la primera página de 16k, P_0^{16k} y la memoria de 32k ocupando las dos páginas P_1^{16k} y P_2^{16k}.
Aunque esta opción garantiza la continuidad de la memoria, no es fácil de implementar dado que las dos páginas consecutivas P_1^{16k} y P_2^{16k} no equivalen a ninguna página de 32k.
Ello no implica que no se pueda hacer, pero es más laborioso. Queda como ejercicio para el lector determinar la circuitería que sería necesaria en este caso.

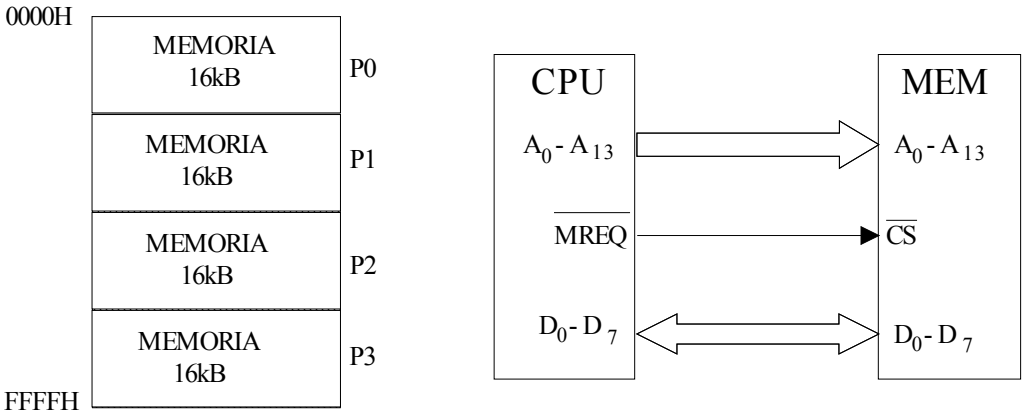


Implementación con un decodificador

Decodificación incompleta y zonas imagen

Si la aplicación diseñada no requiere usar todo el espacio direccionable por el procesador, se puede optar por realizar una decodificación incompleta.

A modo de ejemplo, si deseamos conectar una memoria de 16kB (2^{14}) a un procesador con 16 líneas de direcciones (2^{16}=64kB), podemos prescindir de la decodificación de las líneas A_{14} y A_{15}.



Ejemplo de decodificación incompleta

Dado que la información de A14 y A15 que permite elegir las páginas de 16kB dentro del mapa de memoria no se usa en la decodificación, resulta que la memoria se halla activa para todas las páginas. O lo que es lo mismo, se cumple:

MREQ*	A15	A14	Página 16k	CS*
0	0	0	0	0
0	0	1	1	0
0	1	0	2	0
0	1	1	3	0

Tabla de decodificación incompleta

En realidad, cuando se accede a cada página se está accediendo a la misma memoria, ya que CS* está activo en todas las páginas.
Por tanto, para x desde 0 a 2^{14}-1 todas las direcciones:

$$X \qquad X+2^{14} \qquad X+2\cdot 2^{14} \qquad X+3\cdot 2^{14}$$

Son equivalentes.

Se puede decir que la memoria de cada una de las páginas es una imagen exacta de la de las otras. Es por ello que a todas ellas se les denomina **Zonas Imagen**.

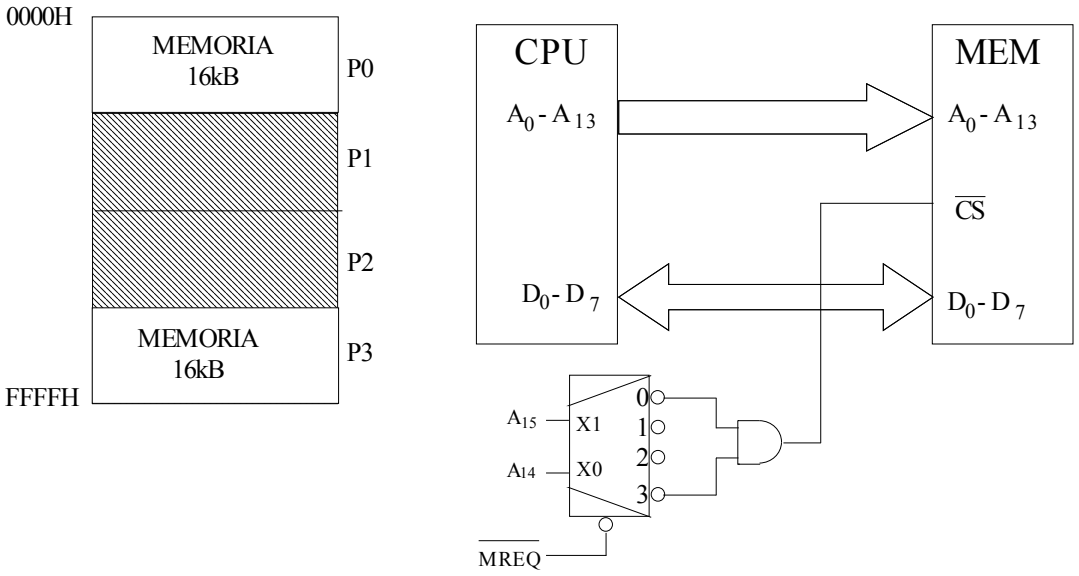
En general, cuando no se usa un número **m** de direcciones del bus de direcciones para realizar la decodificación, el número de zonas imagen mínimo es 2^m.
En nuestro caso no hemos usado 2 líneas A_{14} y A_{15} y tenemos $2^2 = 4$ zonas imagen.

Es fácil saber el número de zonas imagen a partir de la tabla de decodificación. Basta para ello ver el número combinaciones binarias para los indeterminados de la tabla de direcciones.
Los rangos se obtienen para todas las posibles combinaciones binarias de las X de la tabla.

A_{15}	A_{14}	$A_{13}...A_0$	CPU	N img.	Rangos
X	X	$A_{13}...A_0$	Memoria	2^2=4	0 a 2^{14}-1 2^{14} a $2\cdot 2^{14}$-1 $2\cdot 2^{14}$ a $3\cdot 2^{14}$-1 $3\cdot 2^{14}$ a $4\cdot 2^{14}$-1

Tabla de decodificación incompleta con X en las líneas no usadas

Es interesante obtener el número de imágenes de un análisis como el de la tabla anterior, ya que el hecho de usar todas las líneas de direcciones no garantiza la ausencia de zonas imagen. En el siguiente ejemplo usamos todas las direcciones de la CPU y, por la manera de hacer la decodificación, tenemos 2 zonas imagen.



Ejemplo con zonas imagen usando todas las direcciones en la decodificación

Evaluar el número de zonas imagen es fácil a partir de la tabla de direcciones:

A_{15}	A_{14}	$A_{13}...A_0$	CPU	N img.	Rangos
0	0	$A_{13}...A_0$	Memoria	2	0 a 2^{14}-1
1	1				$3 \cdot 2^{14}$ a $4 \cdot 2^{14}$-1

Evaluación de las zonas imagen a partir de la tabla de decodificación

Obsérvese que no hay ninguna X en la tabla. De hecho, el número de zonas imagen no tiene porque ser una potencia de 2. En efecto, si en lugar de tomarse las salidas 0 y 3 del decodificador se tomaran las salidas 0, 1 y 2 tendríamos:

A_{15}	A_{14}	$A_{13}...A_0$	CPU	N img.	Rangos
0	0	$A_{13}...A_0$	Memoria	3	0 a 2^{14}-1
0	1				2^{14} a $2 \cdot 2^{14}$-1
1	0				$2 \cdot 2^{14}$ a $3 \cdot 2^{14}$-1

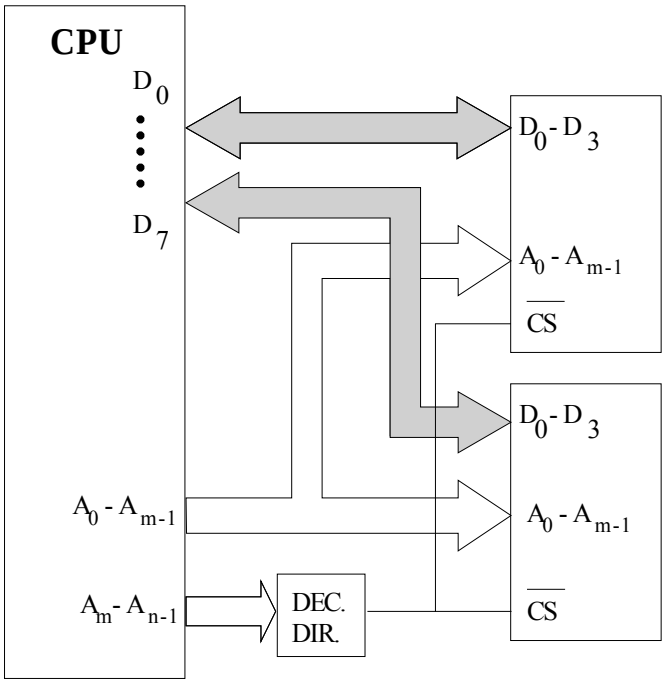
Decodificación con 3 zonas imagen

Memorias de menos de 8 bits

Si la memoria es de menos de 8 bits, podemos juntar varias de ellas con el mismo ancho de direcciones hasta obtener una memoria equivalente de 8 bits.

Para ello juntaremos las líneas de direcciones de todas las memorias junto con CS* y otras señales de control.

Luego juntaremos agruparemos las líneas de datos de las memorias hasta llegar a las 8 líneas necesarias para completar un byte.



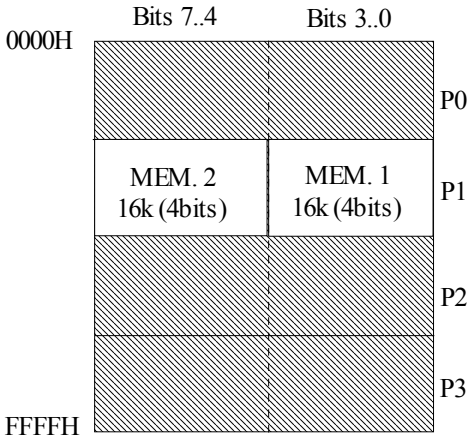
Ejemplo de decodificación para CPU de 8 bits y memorias de 4 bits.

Es posible poner esta distribución de memoria en la tabla de reparto de direcciones. Por ejemplo, si deseáramos poner 2 chips de 16k posiciones de 4 bits en la segunda página de 16k de un procesador con 16 líneas de direccionamiento (64kB), la tabla resultaría:

A_{15}	A_{14}	$A_{13}...A_0$	Bits 7..0	CPU	Rangos
0	1	$A_{13}...A_0$	Bits 3..0	Mem. 1	10^{14} a $2{\cdot}10^{14}$-1 (bits 3..0)
0	1	$A_{13}...A_0$	Bits 7..4	Mem. 2	10^{14} a $2{\cdot}10^{14}$-1 (bits 7..4)

Tabla de decodificación con una columna para la distribución de bits.

La siguiente figura muestra como quedaría el mapa de memoria.



Mapa de memoria para 2 memorias de menos de 1 byte.

Decodificación para bus de datos de varios bytes

Los buses de datos de los μP/μC no siempre son de 8 bits (1 byte).

El número de bytes (**NB**) del bus de datos suele ser una potencia de 2.
La anchura **w**, en bits, del bus de datos suele ser por tanto del tipo $w=8\cdot 2^x=8\cdot NB$.
Los valores habituales son 8 (byte), 16 (word) y 32 (dword).

Con independencia de la anchura del bus de datos, las líneas de direcciones de la CPU $A_0..A_{n-1}$ siempre referencian bytes (8 bits). El mapa de memoria también se da en bytes puesto que usa la codificación de las líneas del bus de direcciones.
Las direcciones de las memorias, sin embargo, suelen ser siempre desde A0 con independencia de que sean bytes, words o dwords.

Si el bus de datos es de 16 bits, no tiene sentido que la línea A_0 salga de la CPU puesto que cuando A_0 sea 0 (direcciones pares), los datos vendrán por las líneas D_0 a D_7 y cuando A_0 sea 1 (direcciones impares), los datos vendrán por las líneas D_8 a D_{15}. De hecho, la información de A_0 saldrá de la CPU no por una línea con ese nombre sino a través de la activación de los *strobes* de byte.

Ancho del bus de datos	Dir. salientes CPU	Ejemplo Strobes
8 bits (Byte)	$A_0...A_{n-1}$	MREQ*
16 bits (Word)	$A_1..A_{n-1}$	UDS*,LDS*
32 bits (Dword)	$A_2..A_{n-1}$	BE0*,BE1*,BE2*,BE3*

Lineas de direcciones y Strobes

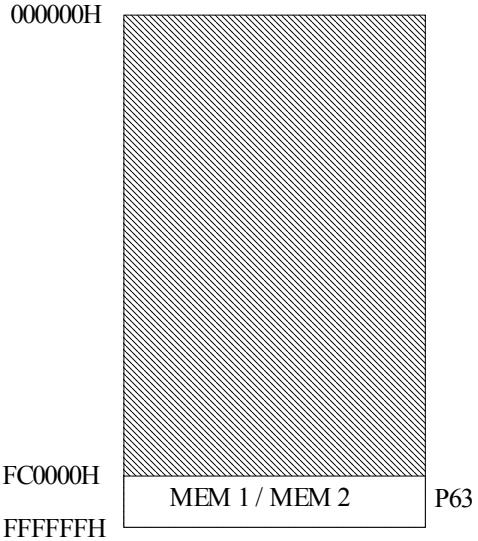
A la hora de hacer la decodificación, la manera de proceder, a priori, es la misma que cuando decodificamos con buses de datos de 8 bits.
Si las memorias son de 8 bits (1 Byte), los bits de direcciones mas bajos que no salen del procesador, A_0 en 16 bits y A_0, A_1 en 32 bits deberán ser decodificados, puesto que pertenecen a chips distintos. La única diferencia con el caso de buses de 8 bits es que valores, por ejemplo, de A_0=0 y A_0=1 pueden ser accedidos a la vez usando los strobes.

A modo de ejemplo mostraremos como poner 2 chips de 128kB (2^{17}) en un procesador 68000 que tiene un bus de datos de 16 bits.
Los dos chips de memoria usarán A_0 en la decodificación por lo que uno empleará direcciones pares y el otro impares. En conjunto direccionarán 256kB (2^{18}). El 68000 tiene 24 líneas de direcciones por lo que tiene $2^{24-18}=2^6=64$ páginas de 268kB.
Situaremos, a modo de ejemplo, la memoria en la última página, la número 63.

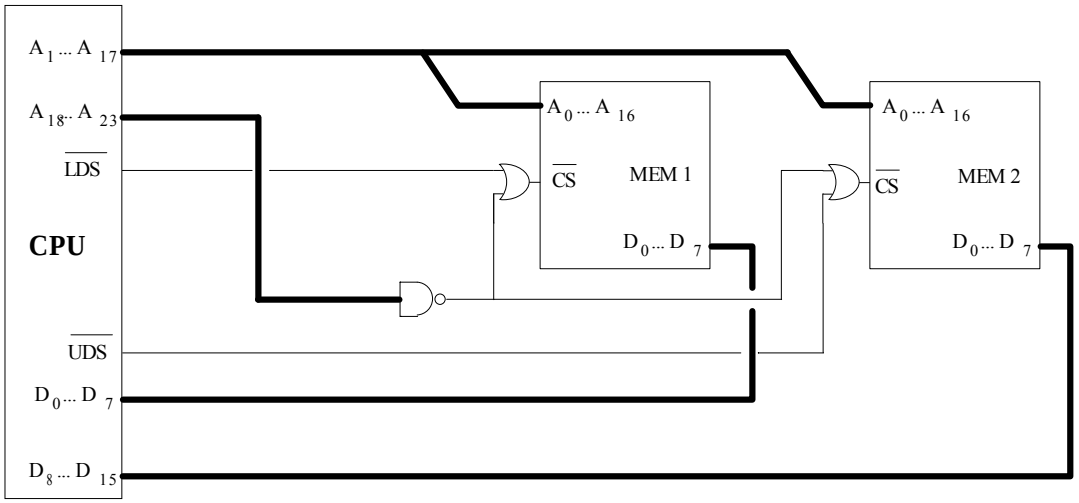
A_{23}	A_{22}	A_{21}	A_{20}	A_{19}	A_{18}	$A_{17}..A_1$	Strobes (A_0)	CPU	Rango
1	1	1	1	1	1	$A_{16}..A_0$	LDS* (A_0=0)	MEM 1	$63\cdot2^{18}$ a $64\cdot2^{18}$-1 (Pares)
1	1	1	1	1	1	$A_{16}..A_0$	UDS* (A_0=1)	MEM 2	$63\cdot2^{18}$ a $64\cdot2^{18}$-1 (Impares)

Ejemplo de tabla de decodificación en el μP 68000

El mapa de memoria resultante será:



Un diagrama circuital que implementa esta decodificación es:



Caso de que se tratara de una CPU de 32 bits, el número de chips de memoria de 8 bits habría de ser múltiplo de 4.
A continuación se muestra un ejemplo que conecta una CPU con 24 líneas en el bus de direcciones y con bus de datos de 32 bits con 4 memorias de 128kB para un total de 512kB en la primera página de memoria (de 512kB).

A_{23}	A_{22}	A_{21}	A_{20}	A_{19}	$A_{18}..A_2$	Strobes (A_1A_0)	CPU	Rango
0	0	0	0	0	$A_{16}..A_0$	BE0* (00)	Mem 1	0 a 2^{19}-1 Dir. mod 4 = 0
0	0	0	0	0	$A_{16}..A_0$	BE1* (01)	Mem 2	0 a 2^{19}-1 Dir. mod 4 = 1
0	0	0	0	0	$A_{16}..A_0$	BE2* (10)	Mem 3	0 a 2^{19}-1 Dir. mod 4 = 2
0	0	0	0	0	$A_{16}..A_0$	BE3* (11)	Mem 4	0 a 2^{19}-1 Dir. mod 4 = 3

Ejemplo de tabla de decodificación para una CPU de 32bits

Memorias de más de 8 bits

Para μP/μC con bus de datos de más de 8 bits, también existe la posibilidad de emplear memorias con el mismo ancho de bus de datos que el microprocesador.

Si la CPU puede hacer accesos con ancho menor que el bus de datos, es necesario que la memoria, para operaciones de escritura, cuente con *strobes* de byte igual que la CPU. En operaciones de lectura no es necesario dado que la CPU sencillamente ignorará los bytes no solicitados con los strobes.

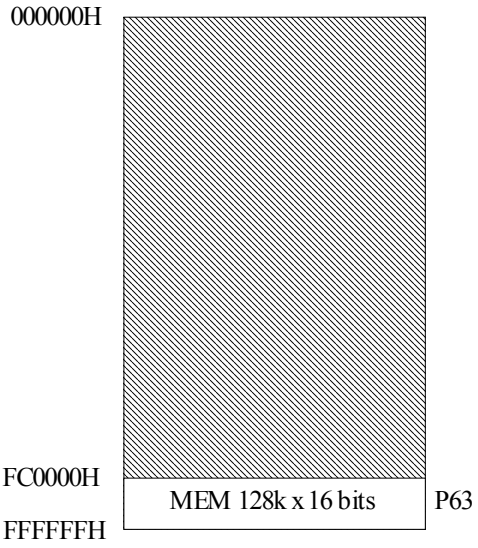
El procedimiento de decodificación es similar al caso anterior. La diferencia se halla en que los *strobes* de byte de la CPU no se usarán para generar las señales CS* de los chips de memoria sino que se conectarán, para memorias RAM, a los *strobes* de entrada de la memoria. Si la memoria es de tipo ROM y no tiene entradas de *strobe* de byte, lo que si se empleará es un *strobe* general de la CPU dentro de la generación de la señal CS*.

La siguiente tabla muestra como conectar un chip de memoria de 128k (2^{17}) posiciones de 16 bits (total de 256kB) en un procesador 68000 que tiene también un bus de datos de 16 bits. Situaremos, igual que en caso anterior, la memoria en la última página, la número 63.

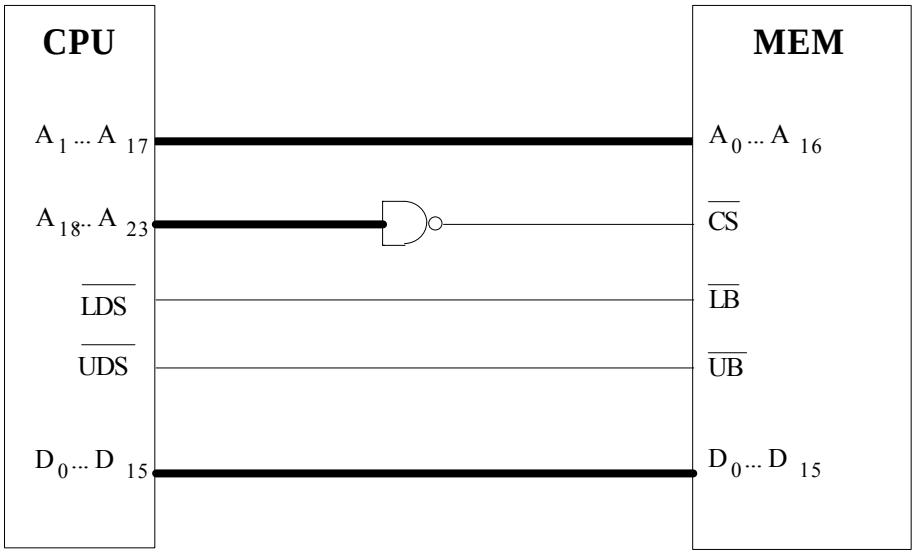
A_{23}	A_{22}	A_{21}	A_{20}	A_{19}	A_{18}	$A_{17}..A_1$	A_0	CPU	Rango
1	1	1	1	1	1	$A_{16}..A_0$	--	MEM	$63 \cdot 2^{18}$ a $64 \cdot 2^{18}-1$

Ejemplo de tabla de decodificación para CPU y memoria de 16 bits

El mapa de memoria resultante será:



Un circuito que realiza la conexión entre la CPU y la memoria sería:

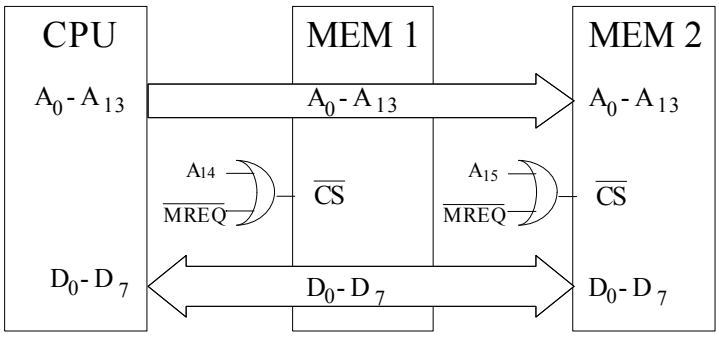


Si la memoria es de tipo ROM, podríamos prescindir de las señales LB* y UB*.

Decodificación con solapamientos

Tal y como se ha indicado anteriormente, no suele ser conveniente que una dirección de memoria de se asocie al mismo tiempo a dos chips distintos dado que daría lugar a una colisión en el bus de datos.
En algunas ocasiones, no obstante, puede ser interesante aprovechar el solapamiento y las zonas imagen para simplificar la circuitería de decodificación.

A modo de ejemplo se mostrará la conexión de dos memorias de 16kB en una CPU con un espacio de direcciones de 64kB.



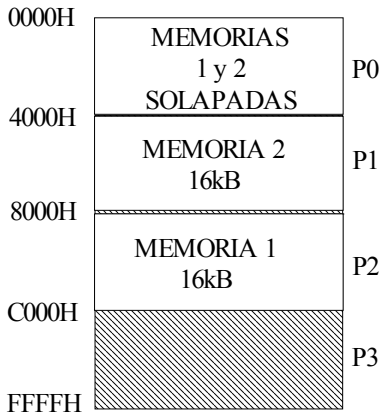
Ejemplo de decodificación con solapamientos

La tabla de decodificación resultante será:

A15	A14	$A_{13}...A_0$	CPU	Rango
0	0	$A_{13}...A_0$	Mem. 1 y 2	0 a 2^{14}-1
0	1	$A_{13}...A_0$	Memoria 2	2^{14} a $2 \cdot 2^{14}$-1
1	0	$A_{13}...A_0$	Memoria 1	$2 \cdot 2^{14}$ a $3 \cdot 2^{14}$-1
1	1	$A_{13}...A_0$	No usado	$3 \cdot 2^{14}$ a $4 \cdot 2^{14}$-1

Ejemplo de tabla de decodificación con solapamientos

El mapa de memoria será:



La CPU podrá acceder a la memoria 1 en el rango de 8000H a BFFFH.
También podrá acceder a la memoria 2 en el rango de 4000H a 7FFFH.
Pero se deberá cuidar mucho la programación para que nunca se intente acceder en lectura a direcciones dentro del rango 0000H a 3FFFFH.
Por otro lado, el acceso en escritura este rango solapado podría ser interesante en aplicaciones en las que interese escribir a la vez en los dos bloques de memoria.

Comentarios adicionales

En general, este tipo de decodificación no suele ser muy útil porque desaprovecha mucho el espacio de memoria disponible debido a los rangos en los que se dan solapamientos.
Si el espacio de direccionamiento de la CPU es de 2^n y las memorias son de 2^m bytes, podrán conectarse un máximo de n-m memorias de este modo (Un CS a cada línea del bus de direcciones de la CPU que no va al bus de direcciones de las memorias).*
El número de páginas disponibles de 2^m bytes es, sin embargo 2^{n-m}.
De ello, la eficiencia de uso del espacio de direcciones es $(n-m)/2^{n-m}$.
Para el ejemplo n=16 y m=14, la eficiencia máxima es $2/2^2 = 50\%$.
Pero para n=24 y m=14, se podrían poner como máximo 10 chips de 16kB y la eficiencia bajaría hasta $10/2^{10} \approx 1\%$

Apendice A : Conversión Decimal ↔ Hexadecimal

La numeración hexadecimal emplea 16 posibles valores para cada dígito. Dado que en notación decimal empleamos sólo 10 valores (0...9), los valores restantes en hexadecimal se extraen del abecedario. De ello:

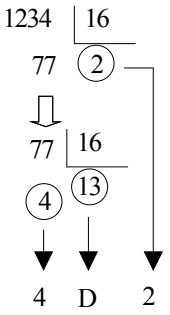
Decimal	Hexadecimal
0	0
1	1
2	2
3	3
4	4
5	5
6	6
7	7

Decimal	Hexadecimal
8	8
9	9
10	A
11	B
12	C
13	D
14	E
15	F

La conversión de hexadecimal a decimal es directa teniendo en cuenta que cada dígito hexadecimal tiene un peso 16x donde x es la posición de este dígito empezando por cero y de derecha a izquierda:

$$4D2H = 4\cdot16^2+13\cdot16^1+2\cdot16^0 = 2 + 16\,(\,13 + 16\,(\,4\,)\,) = 1234$$

La conversión inversa, de decimal a hexadecimal, se puede hacer por divisiones sucesivas del número decimal:



Para el número 1234 decimal, se divide por 16 y se obtiene 2 de cociente y 77 de resto. Reservamos el cociente como dígito hexadecimal de menor peso.
El resto obtenido se vuele a dividir y volvemos a reservar el cociente como siguiente dígito hexadecimal más significativo que el anterior.
El proceso se repite hasta que el resto sea menor que 16.
Este último resto constituye el dígito más significativo.