Memory Interface

Dr. Mohammed Morsy

Memory Interface

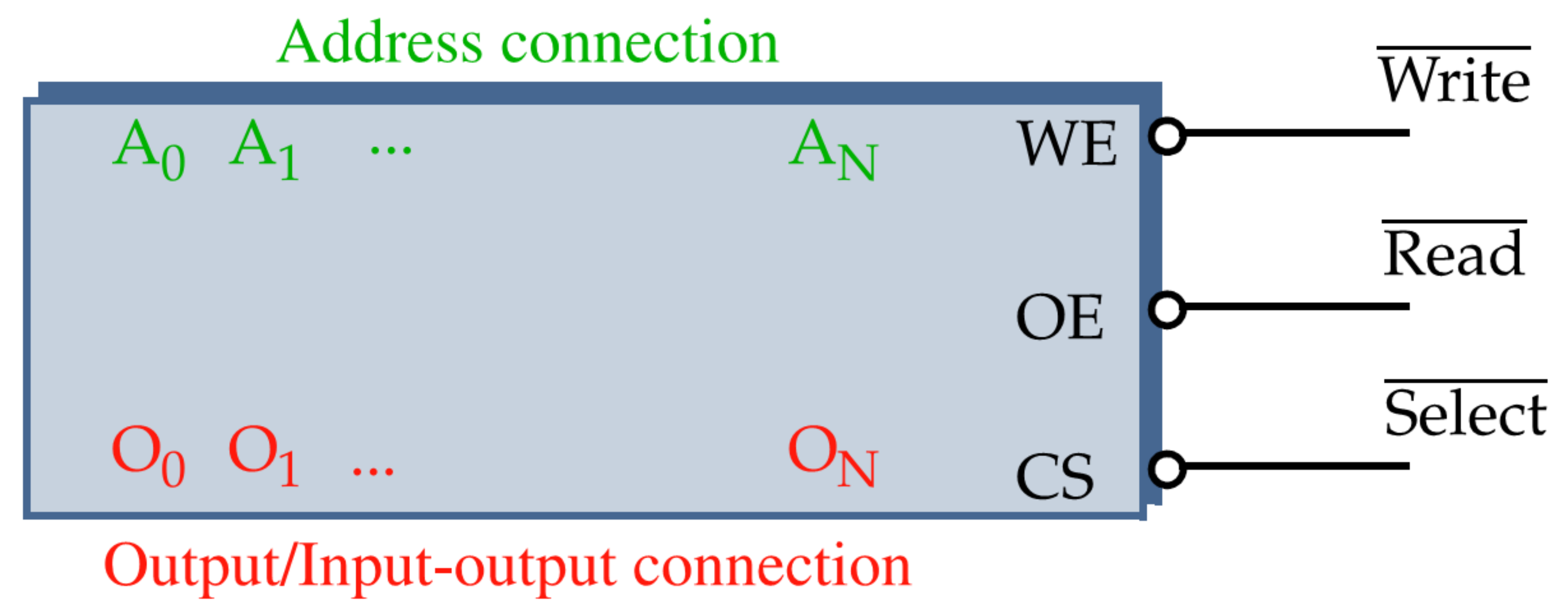
- This chapter will discuss:
  - Memory Devices:
    ROM, EEPROM, SRAM, DRAM.
  - Address Decoding.
  - 8088 Memory Interface.
  - 8086 Memory Interface.

Memory Types

- Every microprocessor-based system has a memory system
- Two basic types:
  - Read-only memory (ROM): It stores system software and permanent system data
  - Random access memory (RAM): It stores temporary data and application software (data and instructions)
- Four commonly used memories:
  - ROM
  - Flashable EEPROM
  - Static RAM (SRAM)
  - Dynamic RAM (DRAM), SDRAM, RAMBUS, DDR RAM

Memory Devices

- Generic pin configuration:



Address Pins

- All memory devices have address inputs.
- They select a memory location within the memory device.
- The number of address pins is related to the number of memory locations.
  - Common sizes today are 1K to 256M locations.
  - Therefore, between 10 and 28 address pins are present.
- Address inputs are labeled from A_0 to A_N.
- N is the total number of address pins minus 1.
- Example: The 2K memory:
  - It has 11 address lines.
  - The labels are (A_0 - A_{10}).
  - If the start address is 10000H so the end address is: 10000H + $((2*1024)_{10}$ = 800H)=107FFH

Data Pins

- All memory devices have a set of data outputs or input/outputs.
- They are used to enter the data for storage or extract the data for reading.
- The data pins are typically bi-directional in read-write memories.
  - The number of data pins is related to the size of the memory location.
  - For example, an 8-bit wide (byte-wide) memory device has 8data pins.
  - Catalog listing of 1K X 8 indicate a byte addressable 8Kbit memory with 10 address pins.
- Memory devices are defined by memory locations times bit per location.
- Examples: 1K×8, 16K×1, 64K×4.

Selection Pins

- Each memory device has at least one chip select($\overline{CS}$) or chip enable($\overline{CE}$) or select($\bar{S}$) pin that enables the memory device.
  - This enables read and/or write operations.
  - If more than one are present, then all must be 0 in order to perform a read or write.
- If they are active (logic 0), the memory device performs a read or write operation.
- If they are inactive (logic 1), the memory is disabled and do not do any operation.
- If more than one selection connection is present. All must be activated to read or write.

Control Pins

- Each memory device has at least one control pin
- For ROMs, an Output Enable ($\overline{OE}$) or Gate ($\bar{G}$) is present.
- The $\overline{OE}$ pin enables and disables a set of tri-state buffers.
- For RAMs, a read-write(R/$\overline{W}$) or write enable($\overline{WE}$) and read enable($\overline{OE}$) are present.
- For dual control pin devices, it must be hold true that both are not 0 at the same time.

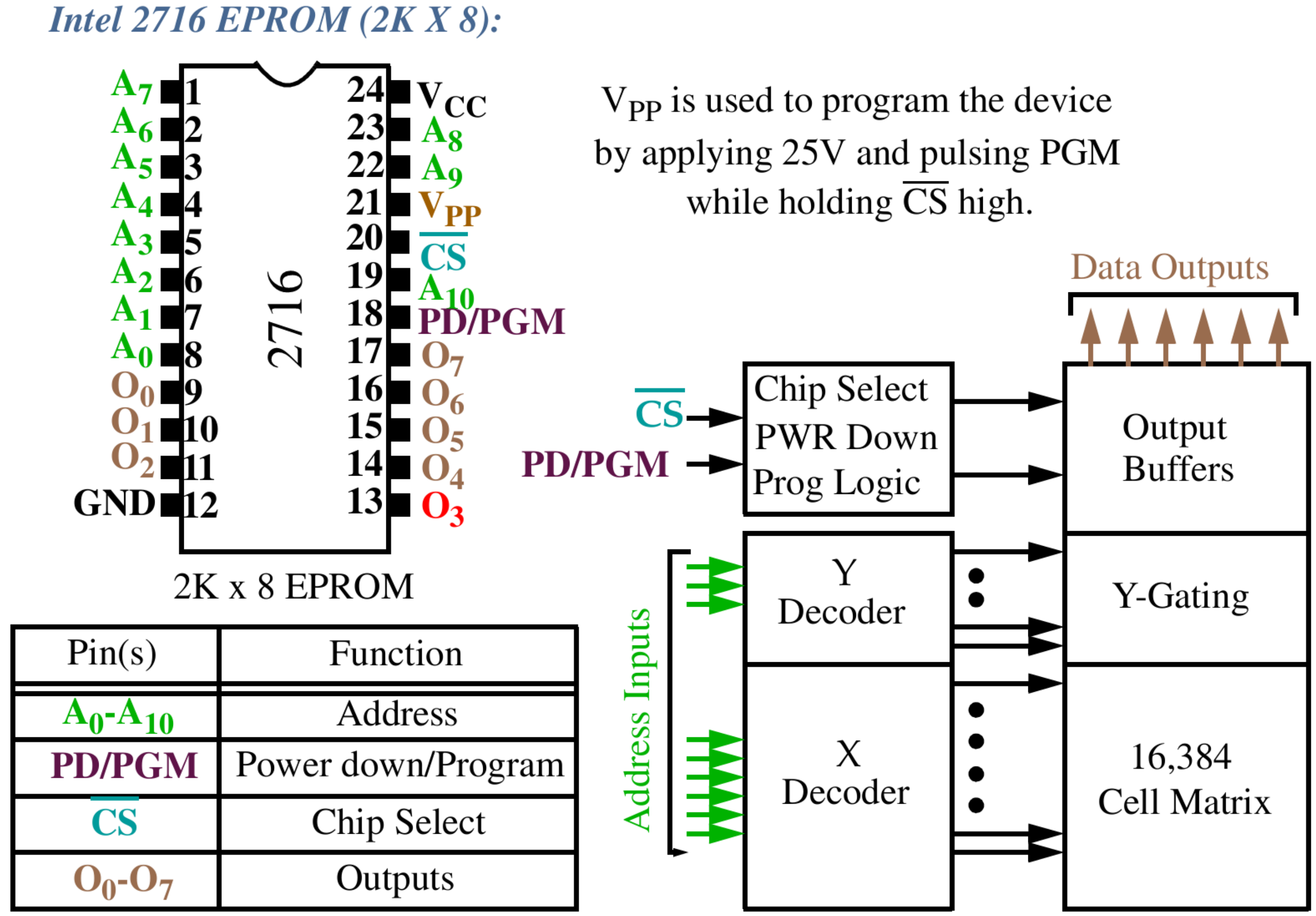
ROMs

- Non-volatile memory: Maintains its state when powered down.
- There are several forms:
  - ROM: Factory programmed, cannot be changed. Older style.
  - Programmable Read-Only Memory (PROM): Field programmable but only once. Older style.
  - Erasable Programmable Read-Only Memory (EPROM): Reprogramming requires up to 20 minutes of high-intensity UV light exposure.
  - Electrically Erasable Programmable ROM (EEPROM): Also called Electrically Alterable ROM (EAROM) and NOVRAM (Non-Volatile RAM).
    - Writing is much slower than a normal RAM.
    - Used to store setup information, e.g. video card, on computer systems. Can be used to replace EPROM for BIOS memory.

ROMs

- The 27XXX series of the EPROM includes:
  - 2704 (512 × 8).
  - 2708 (1K × 8).
  - 2716 (2K × 8).
  - 2732 (4K × 8).
  - 2764 (8K × 8).
  - 27128 (16K × 8).
  - 27256 (32K × 8).
  - 27512 (64K × 8).
  - 271024 (128K × 8).
- Each EPROM has:
  - Address pins
  - 8 data connections
  - One or more selection inputs and one output enable pin.

Intel 2716 EPROM (2K X 8)



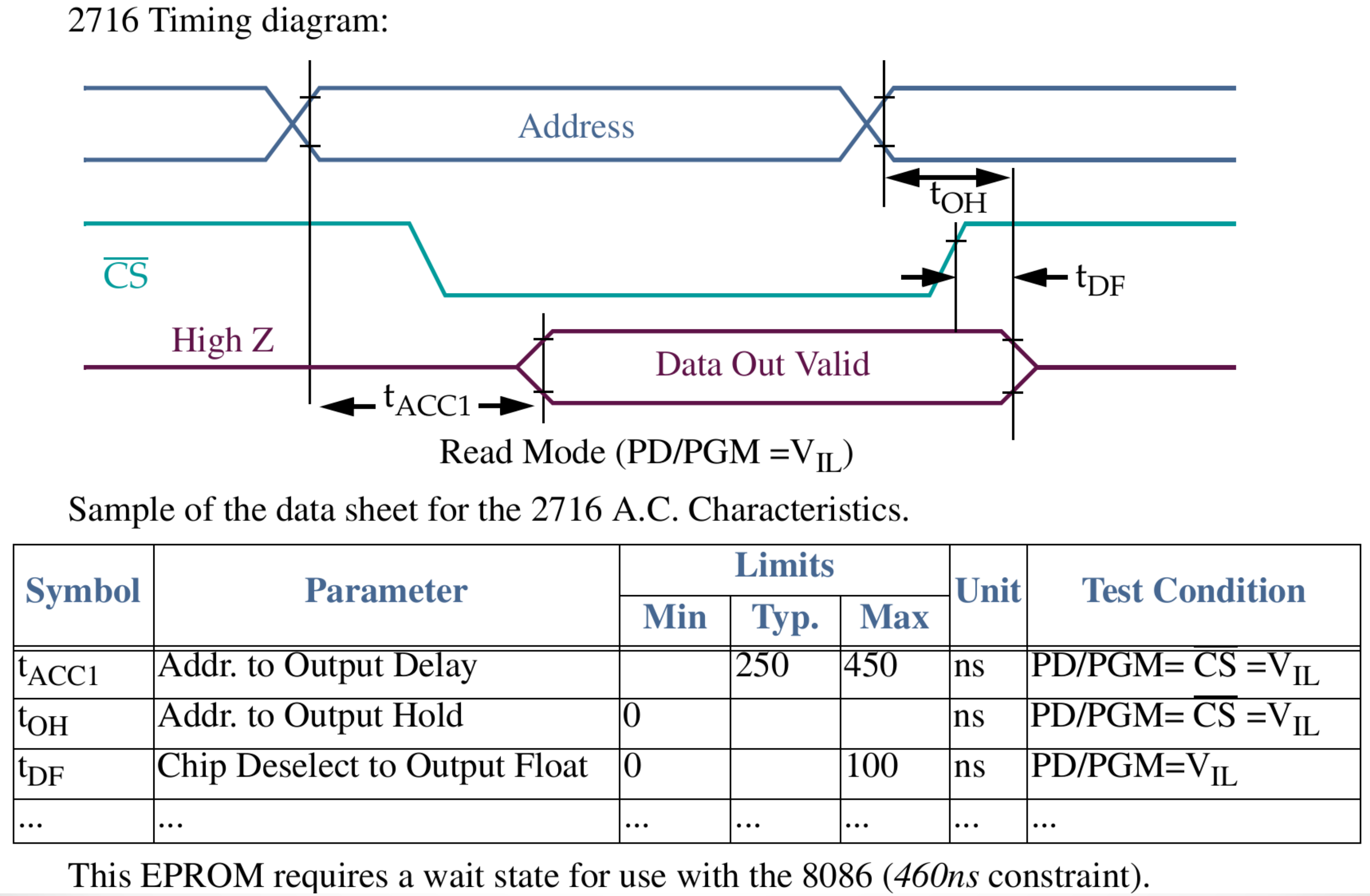
Intel 2716 EPROM (2K X 8):

2K x 8 EPROM

Pin(s)	Function
A_0-A_{10}	Address
PD/PGM	Power down/Program
$\overline{CS}$	Chip Select
O_0-O_7	Outputs

V_{PP} is used to program the device by applying 25V and pulsing PGM while holding $\overline{CS}$ high.

Intel 2716 EPROM (2K X 8)



2716 Timing diagram:

Read Mode (PD/PGM =V_{IL})

Sample of the data sheet for the 2716 A.C. Characteristics.

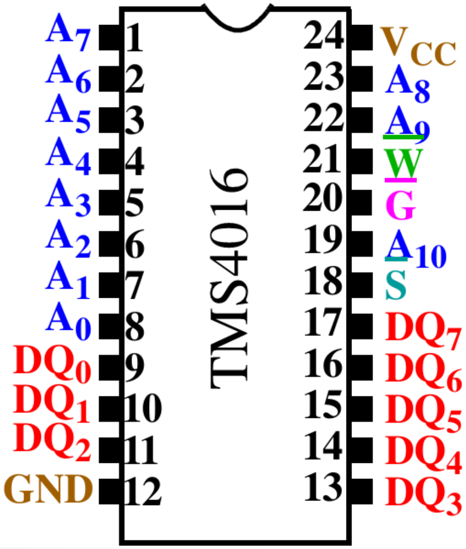
Symbol	Parameter	Limits			Unit	Test Condition
		Min	Typ.	Max		
t_{ACC1}	Addr. to Output Delay		250	450	ns	PD/PGM= $\overline{CS}$ =V_{IL}
t_{OH}	Addr. to Output Hold	0			ns	PD/PGM= $\overline{CS}$ =V_{IL}
t_{DF}	Chip Deselect to Output Float	0		100	ns	PD/PGM=V_{IL}
...	...	...	...	...	...	...

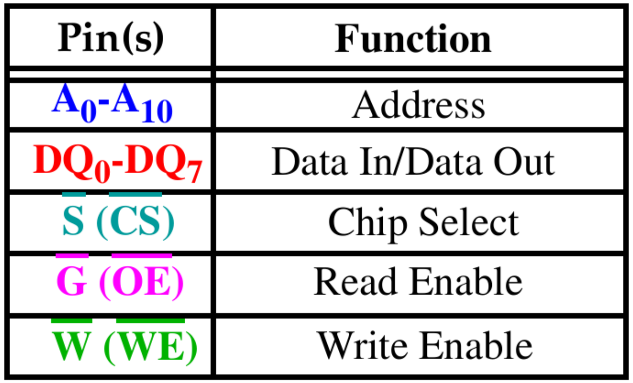
This EPROM requires a wait state for use with the 8086 (*460ns* constraint).

SRAM

- SRAMs are virtually identical to the EPROM with respect to the pinout although access time is faster (250ns).
- SRAMs used for caches have access times as low as 10ns.

TI TMS 4016 SRAM (2K X 8):





Pin(s)	Function
A_0-A_{10}	Address
DQ_0-DQ_7	Data In/Data Out
$\overline{S}$ ($\overline{CS}$)	Chip Select
$\overline{G}$ ($\overline{OE}$)	Read Enable
$\overline{W}$ ($\overline{WE}$)	Write Enable

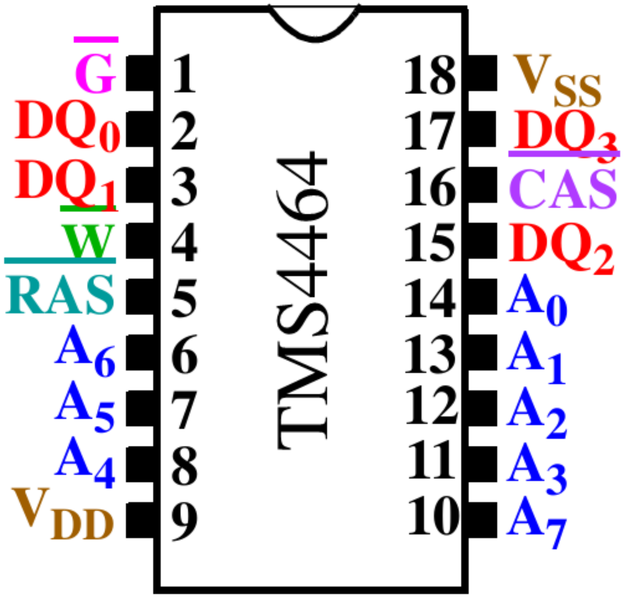
DRAM

- SRAMs are limited in size (up to about 128K X 8).
- DRAMs are available in much larger sizes, e.g., 64M X 1.
- DRAMs MUST be refreshed (rewritten) every 2 to 4 ms Since they store their value on an integrated capacitor that loses charge over time.
- This refresh is performed by a special circuit in the DRAM which refreshes the entire memory.
- Refresh also occurs on a normal read or write.
- The large storage capacity of DRAMs make it impractical to add the required number of address pins.
- Instead, the address pins are multiplexed.

TI TMS4464 DRAM (64K X 4)

- The TMS4464 can store a total of 256K bits of data.
- It has 64Kaddressable locations which means it needs 16 address inputs, but it has only 8.
- The row address (A_0 through A_7) are placed on the address pins and strobed into a set of internal latches.
- The column address (A_8 through A_{15}) is then strobed in using $\overline{CAS}$.

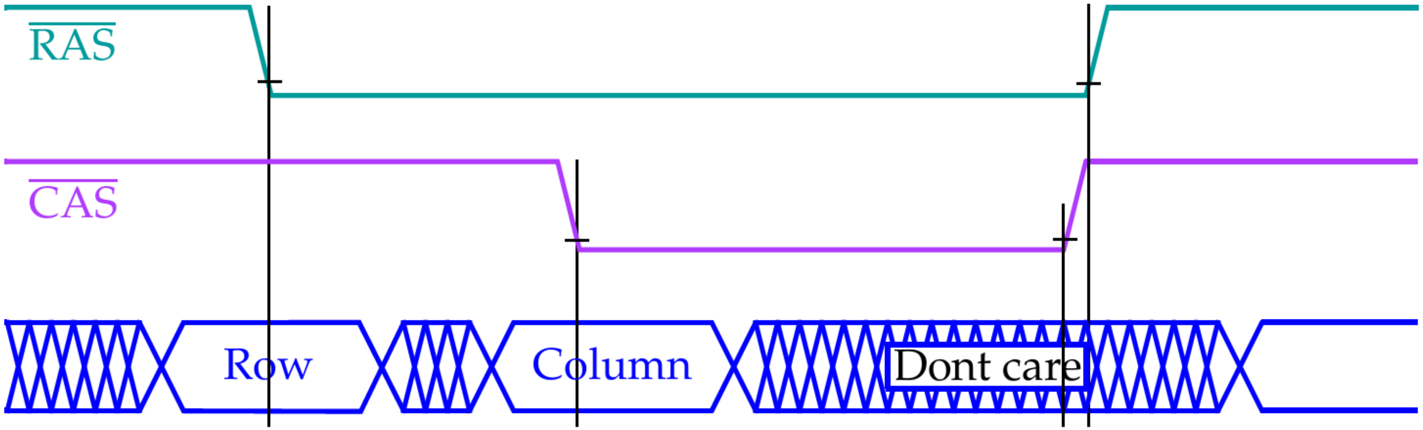
TI TMS4464 DRAM (64K X 4)



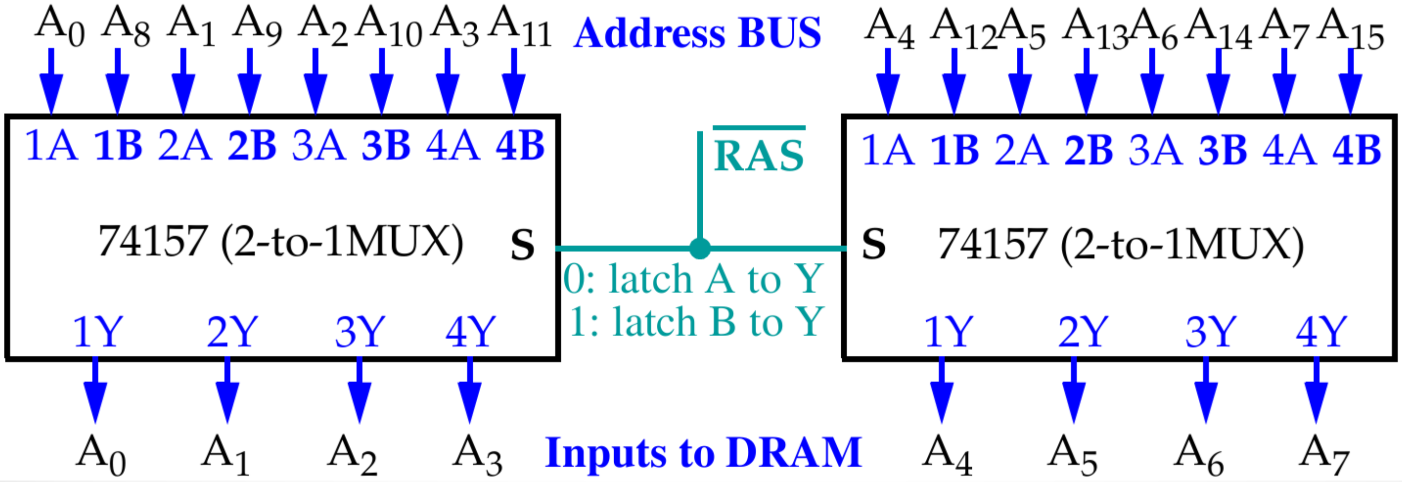
64K x 4 DRAM

Pin(s)	Function
A_0-A_7	Address
DQ_0-DQ_3	Data In/Data Out
$\overline{RAS}$	Row Address Strobe
$\overline{CAS}$	Column Address Strobe
$\overline{G}$	Output Enable
$\overline{W}$	Write Enable

TI TMS4464 DRAM (64K X 4) Timing Diagram:

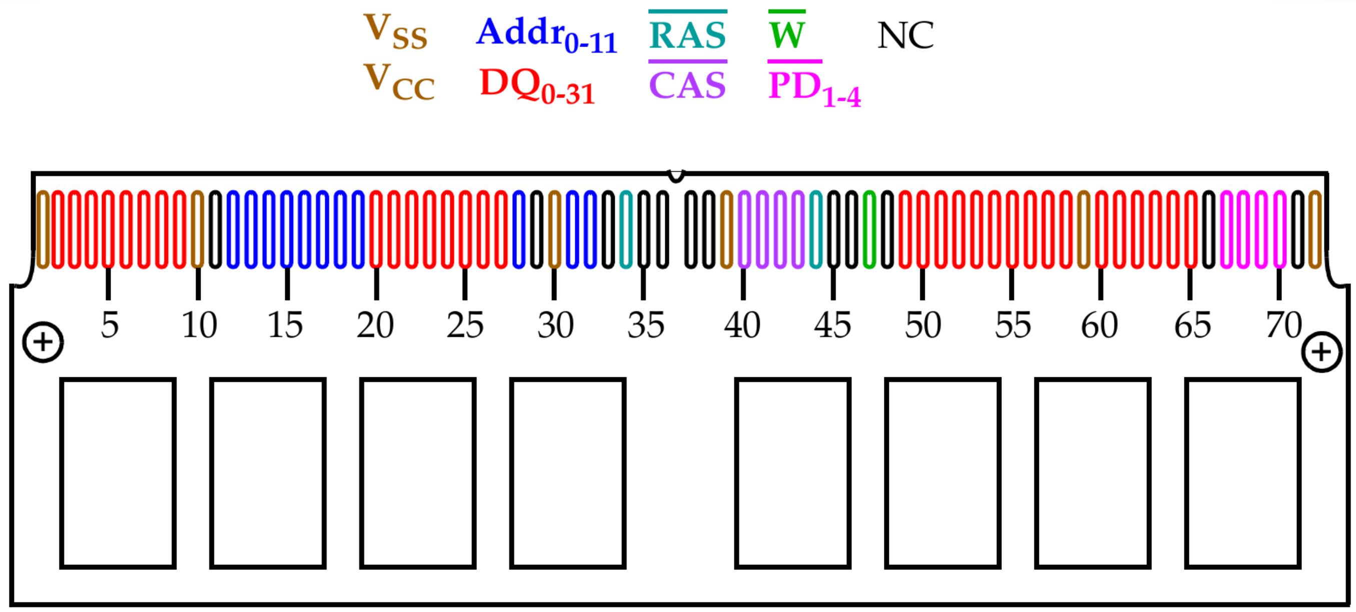


$\overline{\text{CAS}}$ also performs the function of the chip select input.



DRAMs

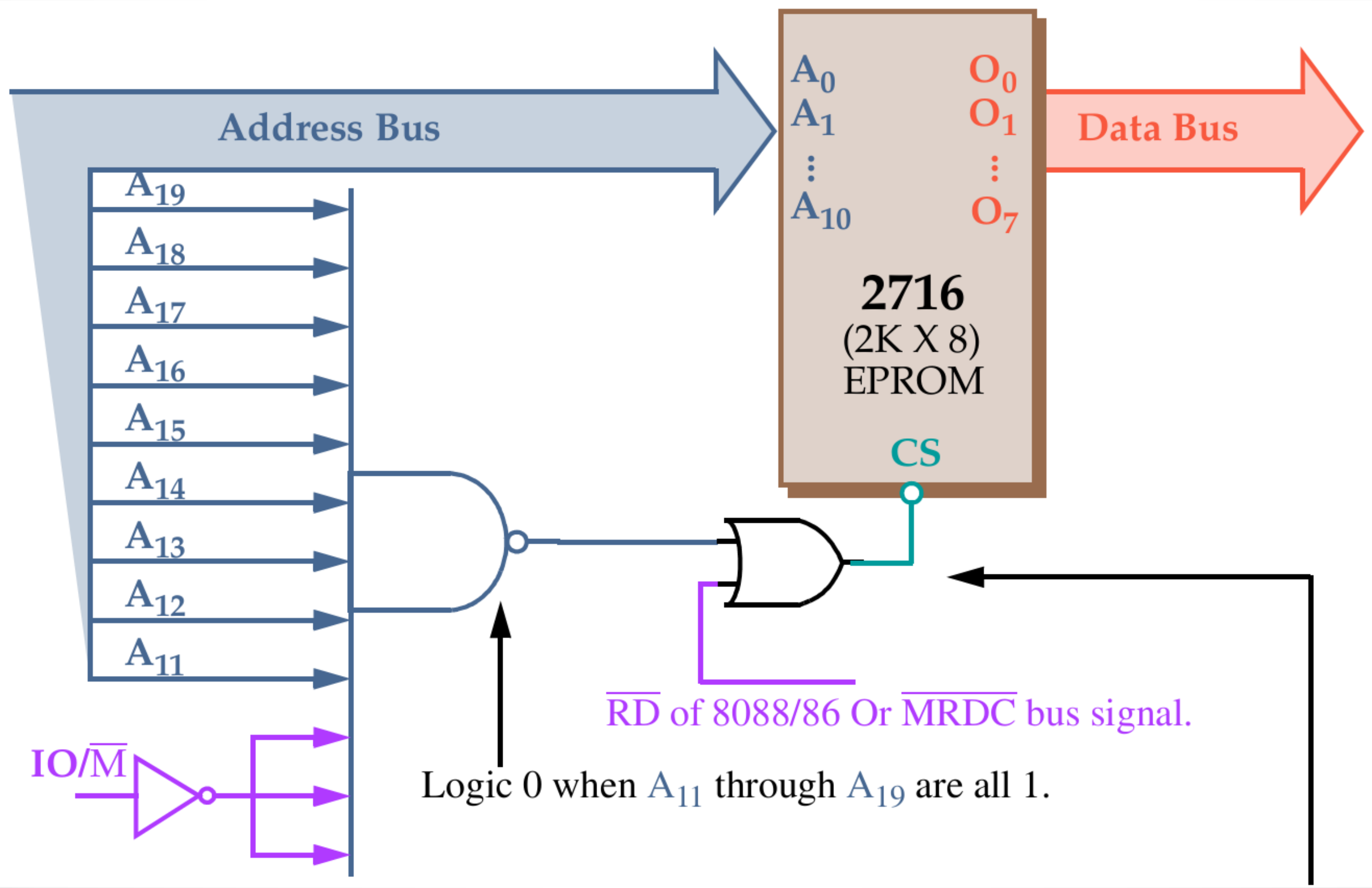
- Larger DRAMs are available which are organized as 1M X 1, 4M X 1, 16M X 1, 64M X 1, 256M X 1.
- DRAMs are typically placed on SIMM (Single In-line Memory Modules) or DIMM (Dual In-line Memory Modules) boards.



Memory Address Decoding

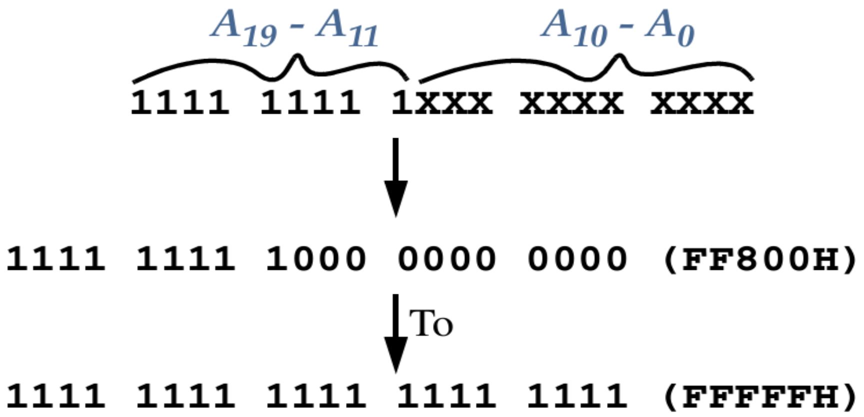
- The processor can usually address a memory space that is much larger than the memory space covered by an individual memory chip.
- In order to splice a memory device into the address space of the processor, decoding is necessary.
- For example, the 8088 issues 20-bit addresses for a total of 1MB of memory address space.
- However, the BIOS on a 2716 EPROM has only 2KB of memory and 11address pins.
- A decoder can be used to decode the additional 9 address pins and allow the EPROM to be
- placed in any 2KB section of the 1MB address space.

NAND Decoder Example



NAND Decoder Example

- To determine the address range that a device is mapped into



A_{19} - A_{11} A_{10} - A_0

1111 1111 1xxx xxxx xxxx

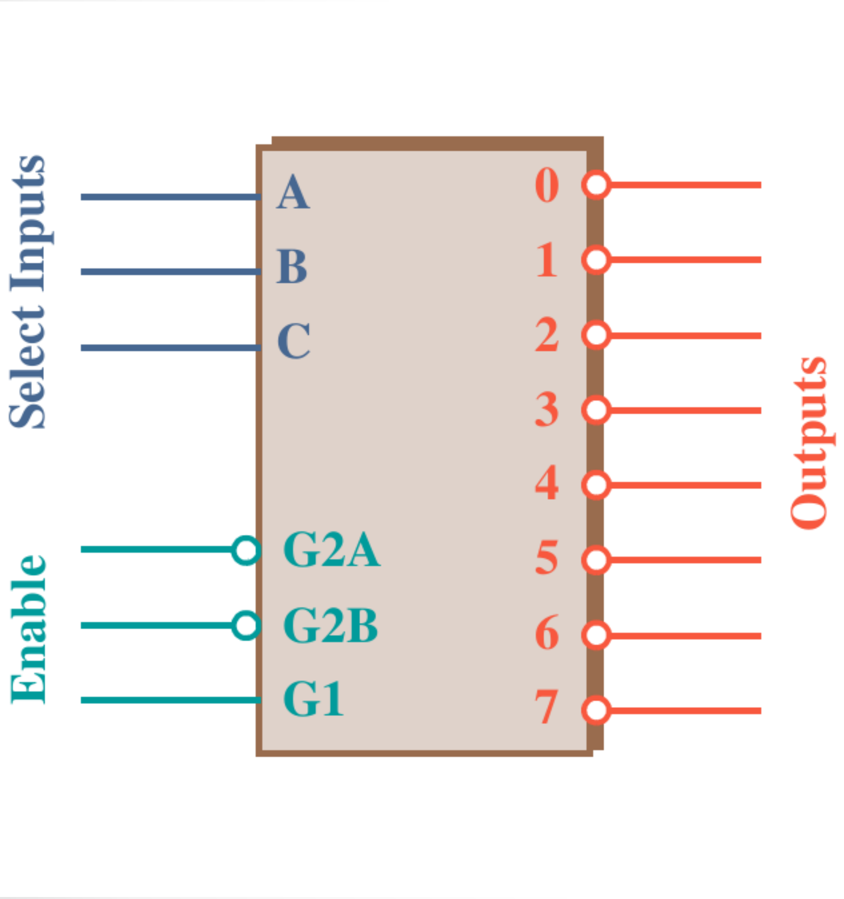
1111 1111 1000 0000 0000 (FF800H)

To

1111 1111 1111 1111 1111 (FFFFFH)

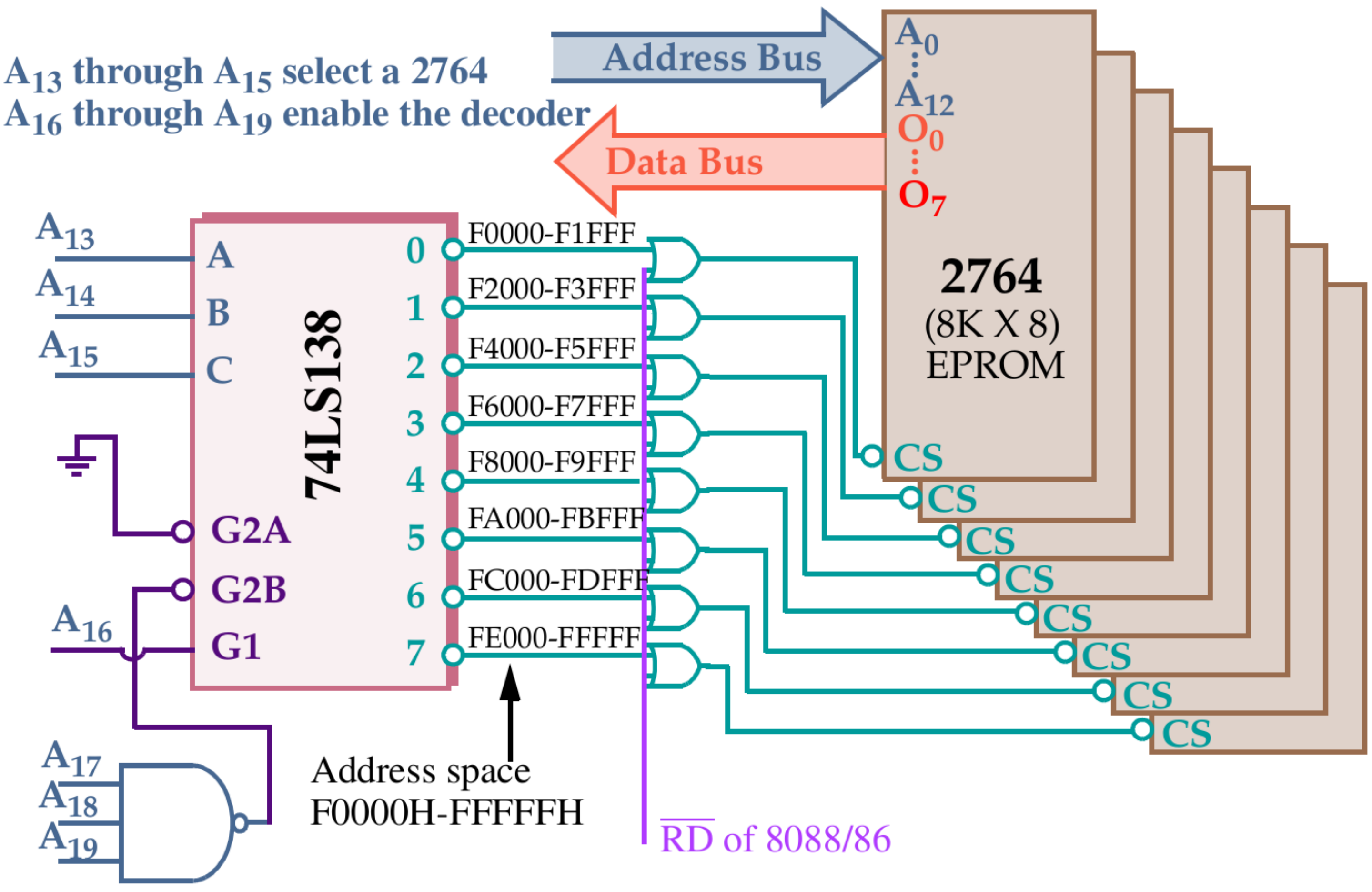
- NAND gate decoders are not often used
  - Large fan-in NAND gates are not efficient
  - Multiple NAND gate IC's might be required to perform such decoding
  - Rather the 3-to-8 Line Decoder (74LS138) is more common.

The 3-to-8 Line Decoder (74LS138)



Inputs						Output							
Enable			Select										
G2A	G2B	G1	C	B	A	$\overline{0}$	$\overline{1}$	$\overline{2}$	$\overline{3}$	$\overline{4}$	$\overline{5}$	$\overline{6}$	$\overline{7}$
1	X	X	X	X	X	1	1	1	1	1	1	1	1
X	1	X	X	X	X	1	1	1	1	1	1	1	1
X	X	0	X	X	X	1	1	1	1	1	1	1	1
0	0	1	0	0	0	0	1	1	1	1	1	1	1
0	0	1	0	0	1	1	0	1	1	1	1	1	1
0	0	1	0	1	0	1	1	0	1	1	1	1	1
0	0	1	0	1	1	1	1	1	0	1	1	1	1
0	0	1	1	0	0	1	1	1	1	0	1	1	1
0	0	1	1	0	1	1	1	1	1	1	0	1	1
0	0	1	1	1	0	1	1	1	1	1	1	0	1
0	0	1	1	1	1	1	1	1	1	1	1	1	0

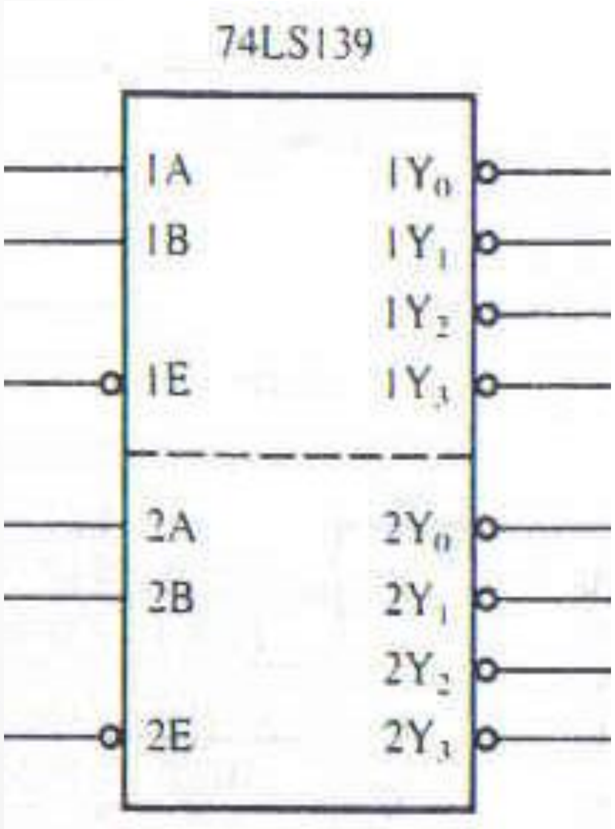
Sample Decoder Circuit

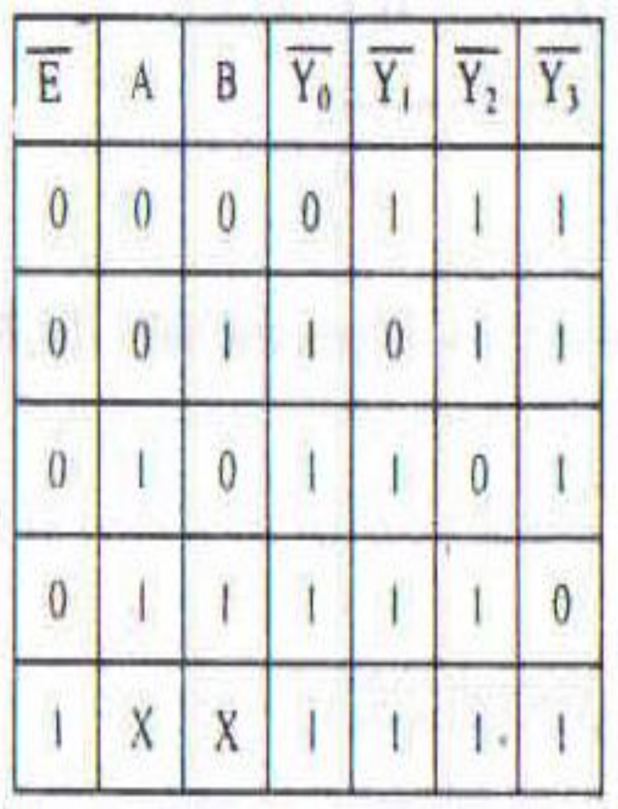


The EPROMs cover a 64KB section of memory.

Dual 2-to-4 Line Decoder

- 74LS139 is a dual 2-to-4 line decoder



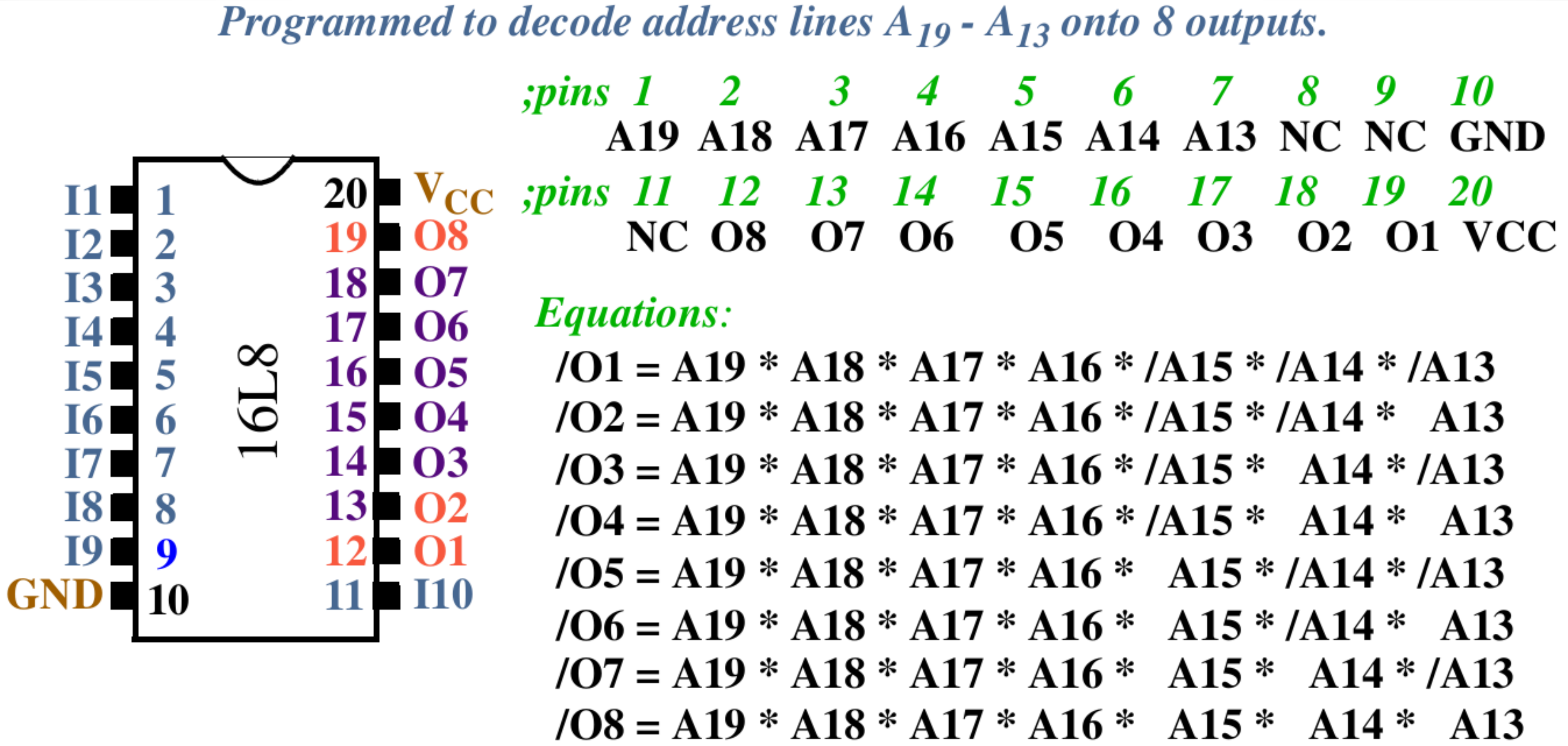


$\overline{E}$	A	B	$\overline{Y_0}$	$\overline{Y_1}$	$\overline{Y_2}$	$\overline{Y_3}$
0	0	0	0	1	1	1
0	0	1	1	0	1	1
0	1	0	1	1	0	1
0	1	1	1	1	1	0
1	X	X	1	1	1	1

Programmable Decoder

- Programmable Logic Devices (PLDs) can be used as a decoder
- PLDs come in three varieties:
  - PLA (Programmable Logic Array)
  - PAL (Programmable Array Logic)
  - GAL (Gated Array Logic)
- PLDs have been around since the mid-1970s but have only recently appeared in memory systems (PALs have replaced PROM address decoders).
- PALs and PLAs can be fuse-programmed (like the PROM) or erasable (like the EPROM).

AMD 16L8 PAL decoder Example



Programmed to decode address lines A_{19} - A_{13} onto 8 outputs.

;pins	*1*	*2*	*3*	*4*	*5*	*6*	*7*	*8*	*9*	*10*
	A19	**A18**	**A17**	**A16**	**A15**	**A14**	**A13**	**NC**	**NC**	**GND**

;pins	*11*	*12*	*13*	*14*	*15*	*16*	*17*	*18*	*19*	*20*
	NC	**O8**	**O7**	**O6**	**O5**	**O4**	**O3**	**O2**	**O1**	**VCC**

Equations:

/O1 = A19 * A18 * A17 * A16 * /A15 * /A14 * /A13

/O2 = A19 * A18 * A17 * A16 * /A15 * /A14 * A13

/O3 = A19 * A18 * A17 * A16 * /A15 * A14 * /A13

/O4 = A19 * A18 * A17 * A16 * /A15 * A14 * A13

/O5 = A19 * A18 * A17 * A16 * A15 * /A14 * /A13

/O6 = A19 * A18 * A17 * A16 * A15 * /A14 * A13

/O7 = A19 * A18 * A17 * A16 * A15 * A14 * /A13

/O8 = A19 * A18 * A17 * A16 * A15 * A14 * A13

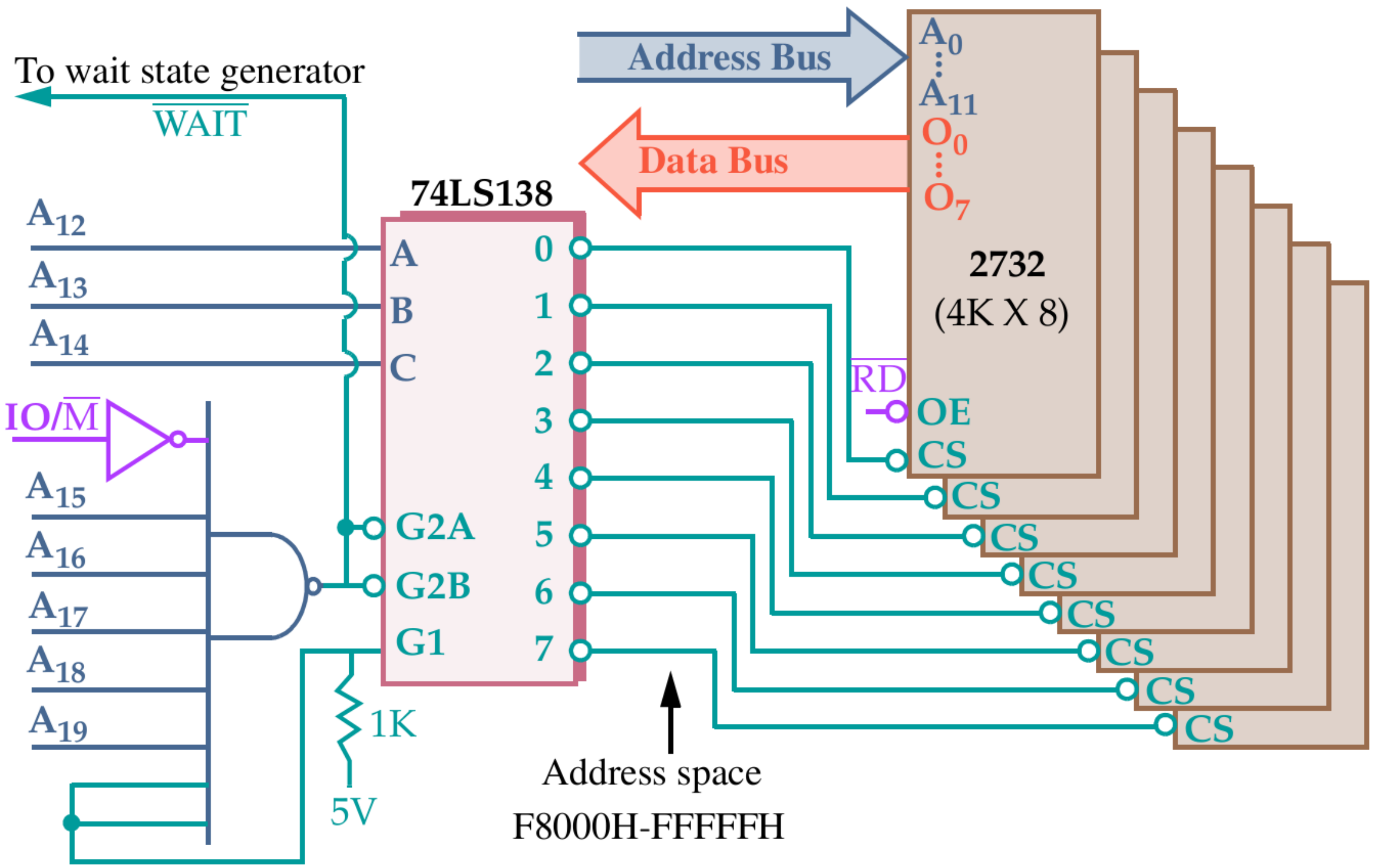
8088 and 80188 (8-bit) Memory Interface

- The memory system "sees" the 8088 as a device with:
  - 20 address connections (A_{19} to A_0).
  - 8 data bus connections (AD_7 to AD_0).
  - 3 control signals: $\mathrm{IO}/\overline{M}$, $\overline{RD}$, and $\overline{WR}$.
- We'll present examples of the 8088 interfacing with:
  - 32K of EPROM (at addresses F8000H through FFFFFH).
  - 512K of SRAM (at addresses 00000H through 7FFFFH).

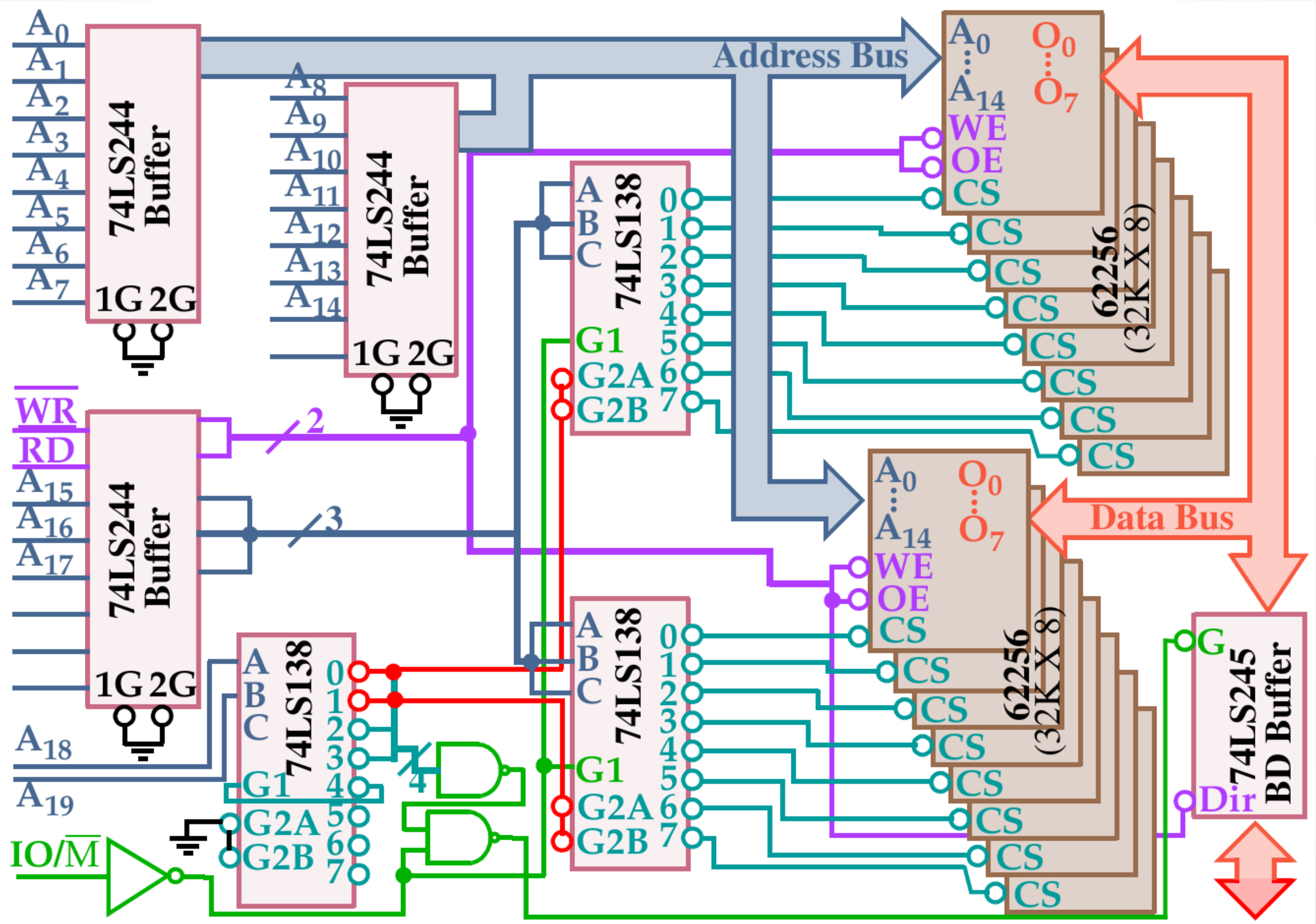
8088 and 80188 (8-bit) EPROM Memory Interface Example

- The EPROM interface uses a 74LS138 (3-to-8 line decoder) plus 8 2732 (4K X 8) EPROMs.
- The EPROM will also require the generation of a wait state.
  - The EPROM has an access time of 450ns .
  - The 74LS138 requires 12ns to decode.
  - The 8088 runs at 5MHz and only allows 460ns for memory to access data.
  - A wait state adds 200ns of additional time

8088 and 80188 (8-bit) EPROM Memory Interface Example



8088 and 80188 (8-bit) SRAM Memory Interface Example



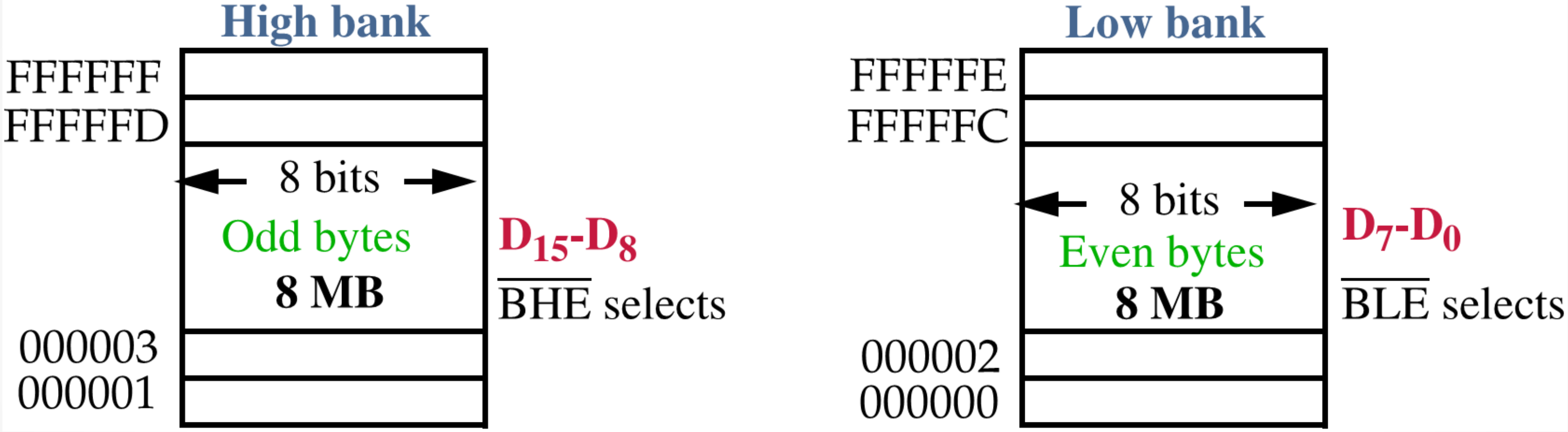
8086,80186, 80286 and 80386 Memory Interface

- These machines differ from the 8088/80188 in several ways:
  - The data bus is 16-bitswide.
  - The IO/$\overline{M}$ pin is replaced with M/$\overline{IO}$ (8086/80186) and $\overline{MRDC}$ and $\overline{MWTC}$ for 80286 and 80386SX.
  - $\overline{BHE}$, Bus High Enable, control signal is added.
  - Address pin A_0 (or $\overline{BLE}$, Bus Low Enable) is used differently.

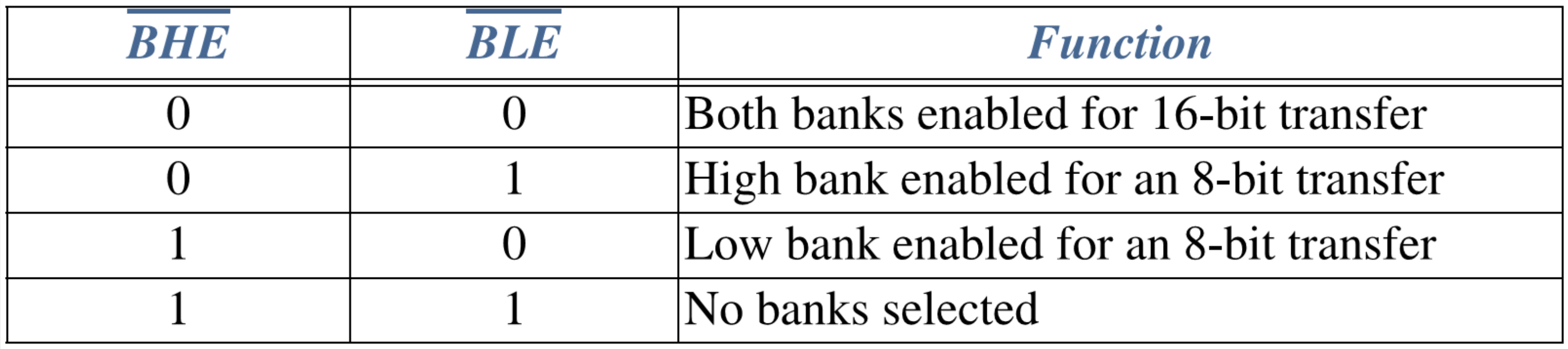
- The 16-bit data bus presents a new problem:
  - The microprocessor must be able to read and write data to any 16-bit location in addition to any 8-bit location.

8086,80186, 80286 and 80386 Memory Interface

- The data bus and memory are divided into banks:



- BHE and BLE are used to select one or both:



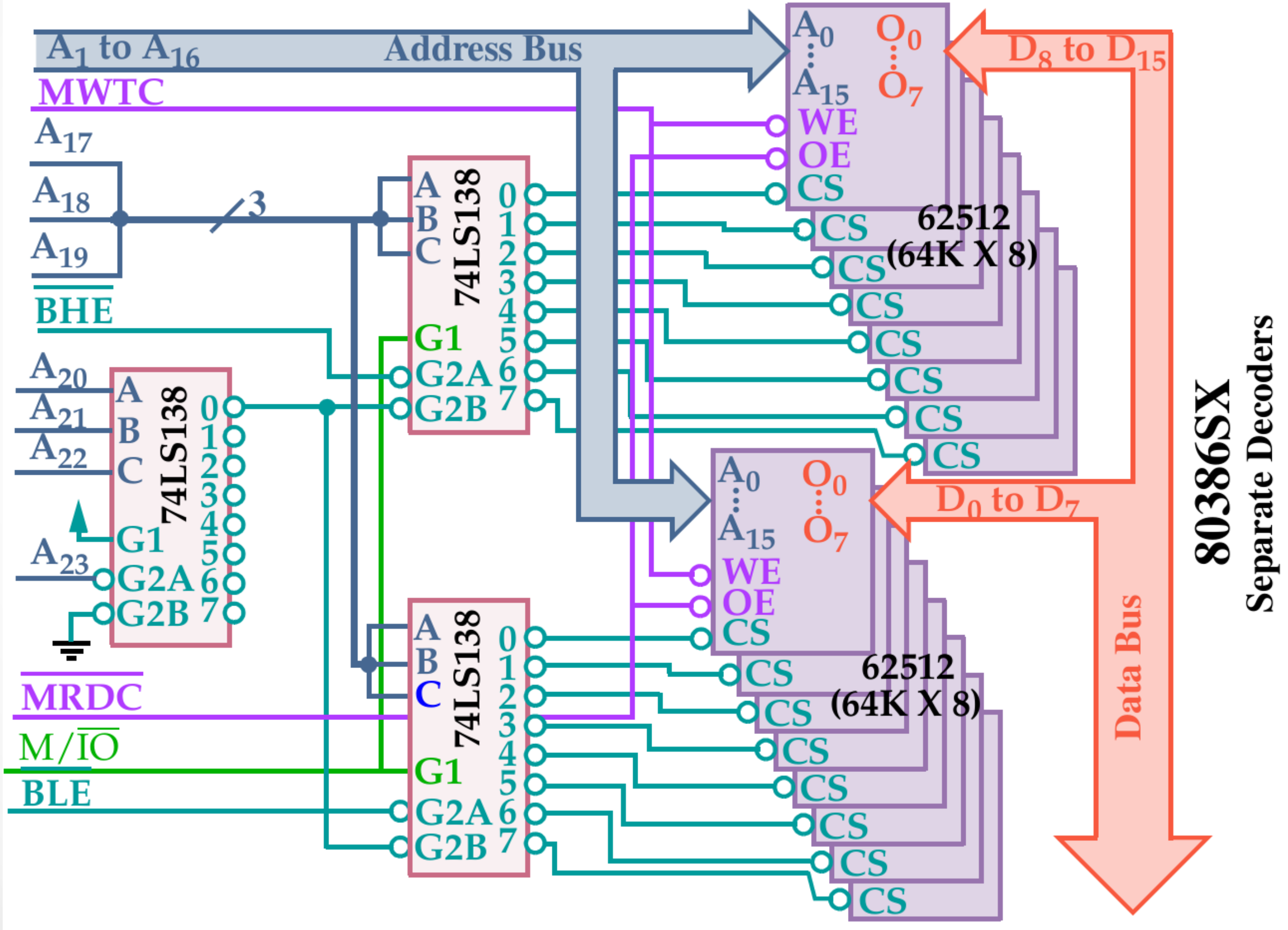
$\overline{BHE}$	***$\overline{BLE}$***	***Function***
0	0	Both banks enabled for 16-bit transfer
0	1	High bank enabled for an 8-bit transfer
1	0	Low bank enabled for an 8-bit transfer
1	1	No banks selected

8086,80186, 80286 and 80386 Memory Interface

- Bank selection can be accomplished in two ways:
  - Separate write decoders for each bank (which drive $\overline{CS}$).
  - A separate write signal (strobe) to each bank (which drive $\overline{WE}$).

  Note that 8-bit read requests in this scheme are handled by the microprocessor (it selects the bits it wants to read from the 16-bits on the bus).
- It does not seem to be a big difference between these methods.
- Note in either method that A_0 does not connect to memory and bus wire A_1 connects to memory pin A_0, A_2 to A_1, etc.

80386SX 16-bit Memory Interface Example (Separate Decoders)



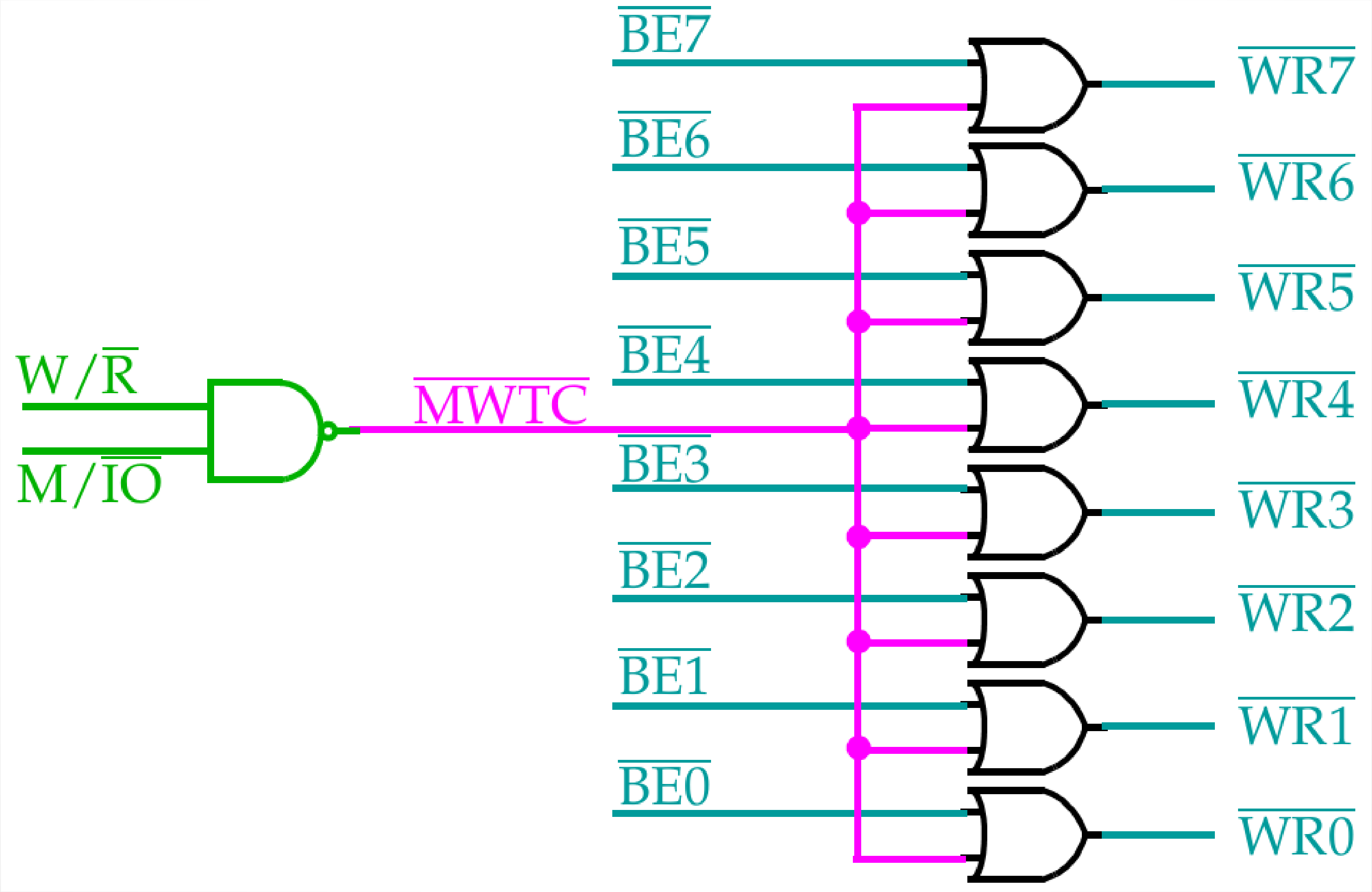
80386DX and 80486 Memory Interface

- 80386DX and 80486 have 32-bit data buses and therefore 4 banks of memory.
- 32-bit, 16-bitand 8-bit transfers are accomplished by different combinations of the bank selection signals $\overline{BE3}, \overline{BE2}, \overline{BE1}, \overline{BE0}$.
- The Address bits A_0 and $A_1$are used within the microprocessor to generate these signals.
- They are don't cares in the decoding of the 32-bit address outside the chip (using a PLD such as the PAL 16L8).
- The high clock rates of these processors usually require wait states for memory access.

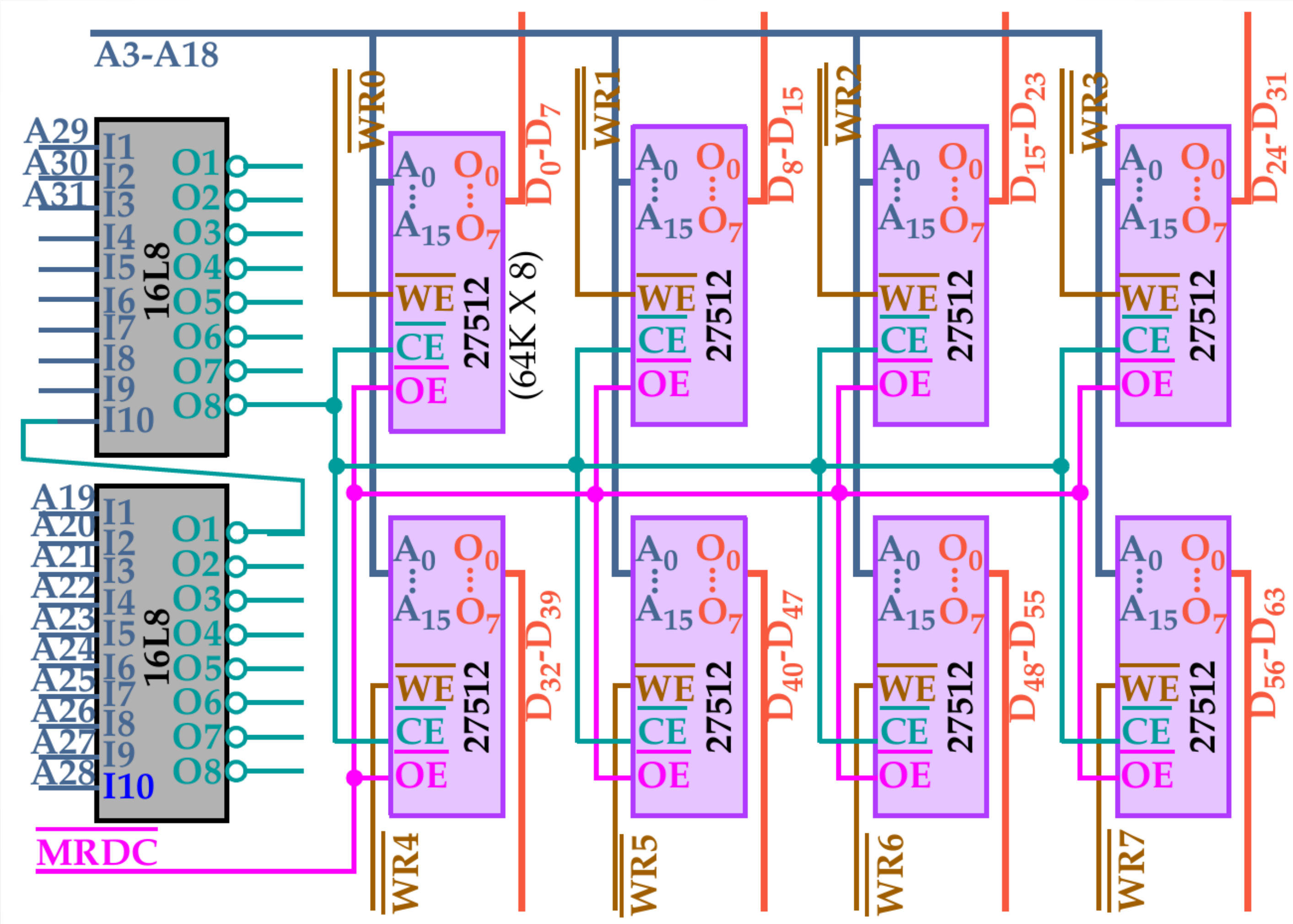
Pentium Memory Interface

- The Pentium, Pentium Pro, Pentium II and III contain a 64-bit data bus.
- Therefore, 8 decoders or 8 write strobes are needed as well as 8 memory banks.
- The write strobes are obtained by combining the bank enable signals ($\overline{BEx}$) with the $\overline{MWTC}$ signal.
- $\overline{MWTC}$ is generated by combining the $\mathrm{M}/\overline{IO}$ and $\mathrm{W}/\bar{R}$ signals.

Pentium Memory Interface

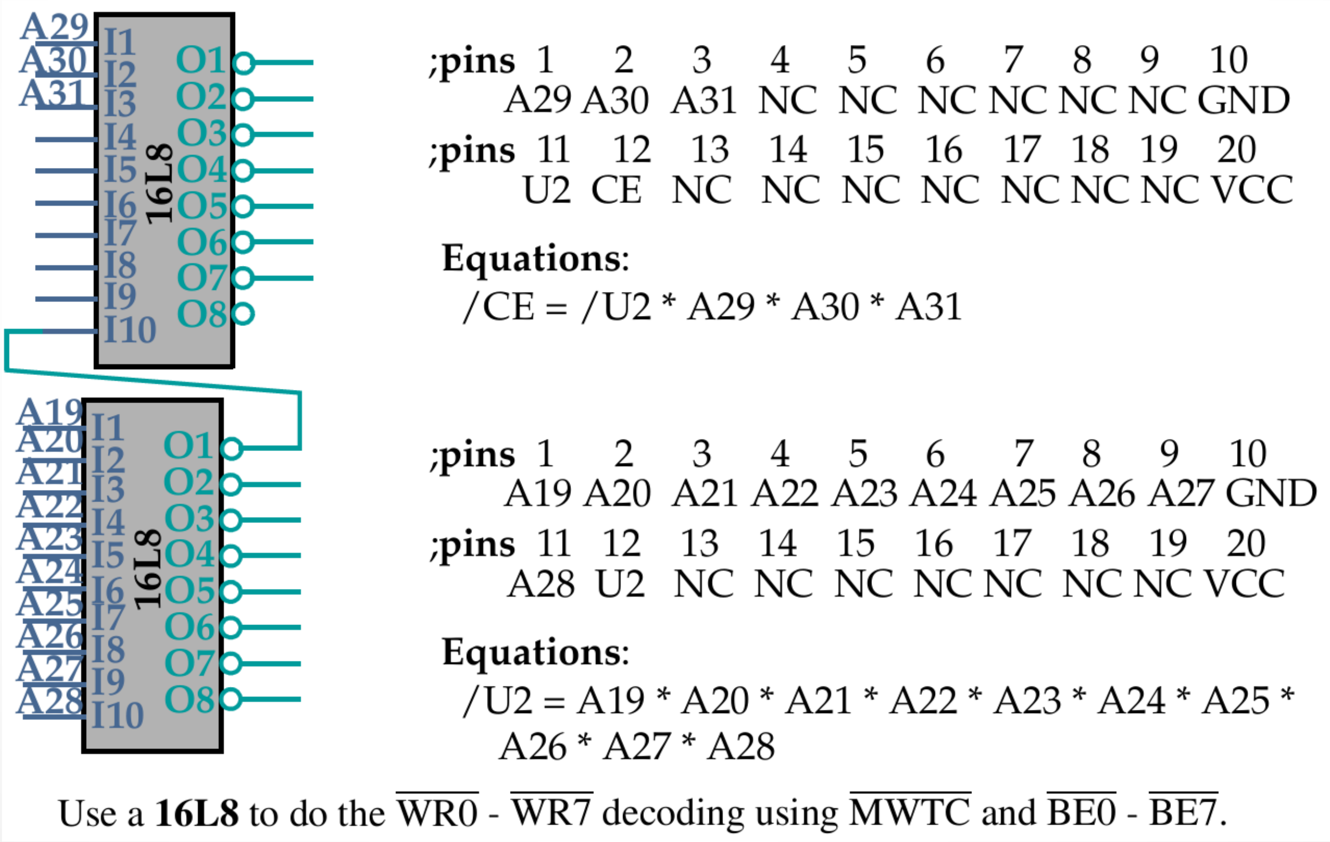


Pentium Memory Interface Example



Pentium Memory Interface Example

- In order to map the previous memory interface into address space FFF80000H-FFFFFFFFH



;pins	1	2	3	4	5	6	7	8	9	10
	A29	A30	A31	NC	NC	NC	NC	NC	NC	GND
;pins	11	12	13	14	15	16	17	18	19	20
	U2	CE	NC	NC	NC	NC	NC	NC	NC	VCC

Equations:

/CE = /U2 * A29 * A30 * A31

;pins	1	2	3	4	5	6	7	8	9	10
	A19	A20	A21	A22	A23	A24	A25	A26	A27	GND
;pins	11	12	13	14	15	16	17	18	19	20
	A28	U2	NC	NC	NC	NC	NC	NC	NC	VCC

Equations:

/U2 = A19 * A20 * A21 * A22 * A23 * A24 * A25 *
A26 * A27 * A28

Use a **16L8** to do the $\overline{\mathrm{WR0}}$ - $\overline{\mathrm{WR7}}$ decoding using $\overline{\mathrm{MWTC}}$ and $\overline{\mathrm{BE0}}$ - $\overline{\mathrm{BE7}}$.

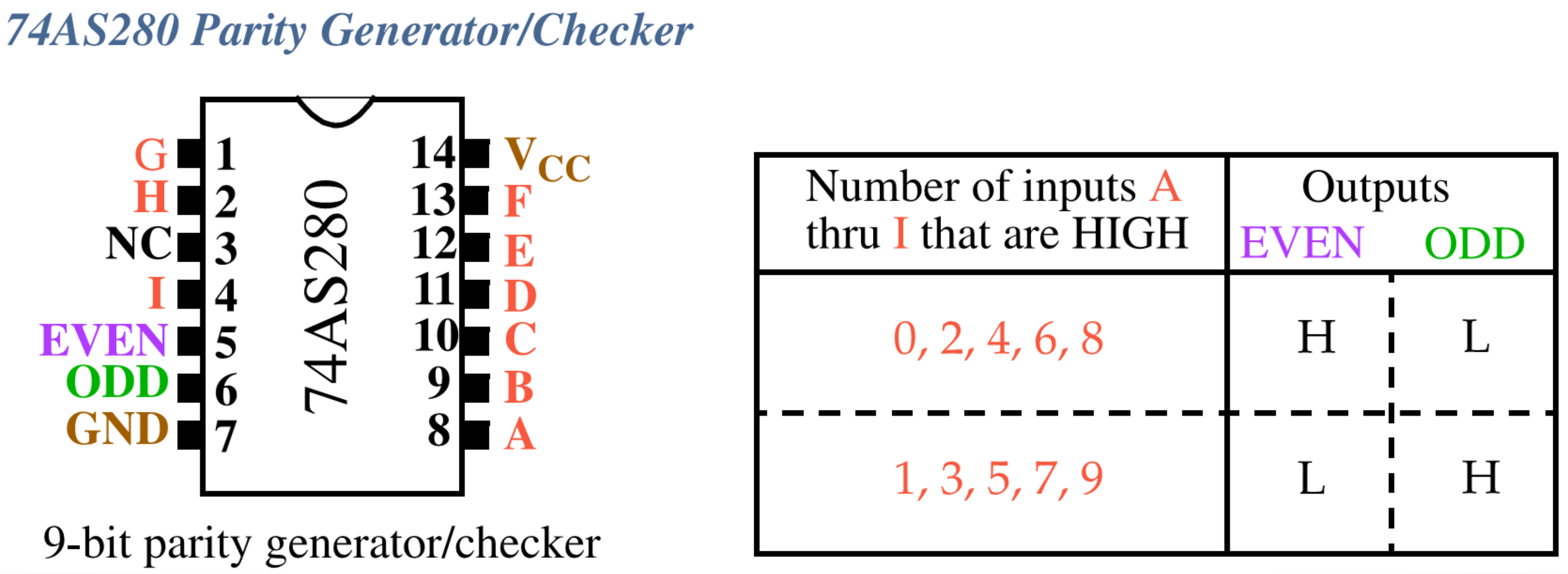
Error Detection and Correction in Memory Devices

- Memory devices use error detection and correction methods to detect and correct memory storing errors.
- Error detection and correction methods attach extra bits to data strings which can help in error detection and correction.
- In this section we will discuss:
    - Parity checking
    - Checksum tests
    - Cyclic Redundancy Check
    - Hamming code error correction

Parity Checking

- Parity checking is used to detect single bit errors in the memory.
- The current trend is away from parity checking.
- Parity checking adds 1bit for every 8 data bits.
  - For EVEN parity, the 9th bit is set to yield an even number of 1's in all 9 bits.
  - For ODD parity, the 9th bit is set to make this number odd.
- For 72-pin SIMMs, the number of data bits is 32 + 4 = 36 (4parity bits).

Parity for Memory Error Detection



74AS280 Parity Generator/Checker

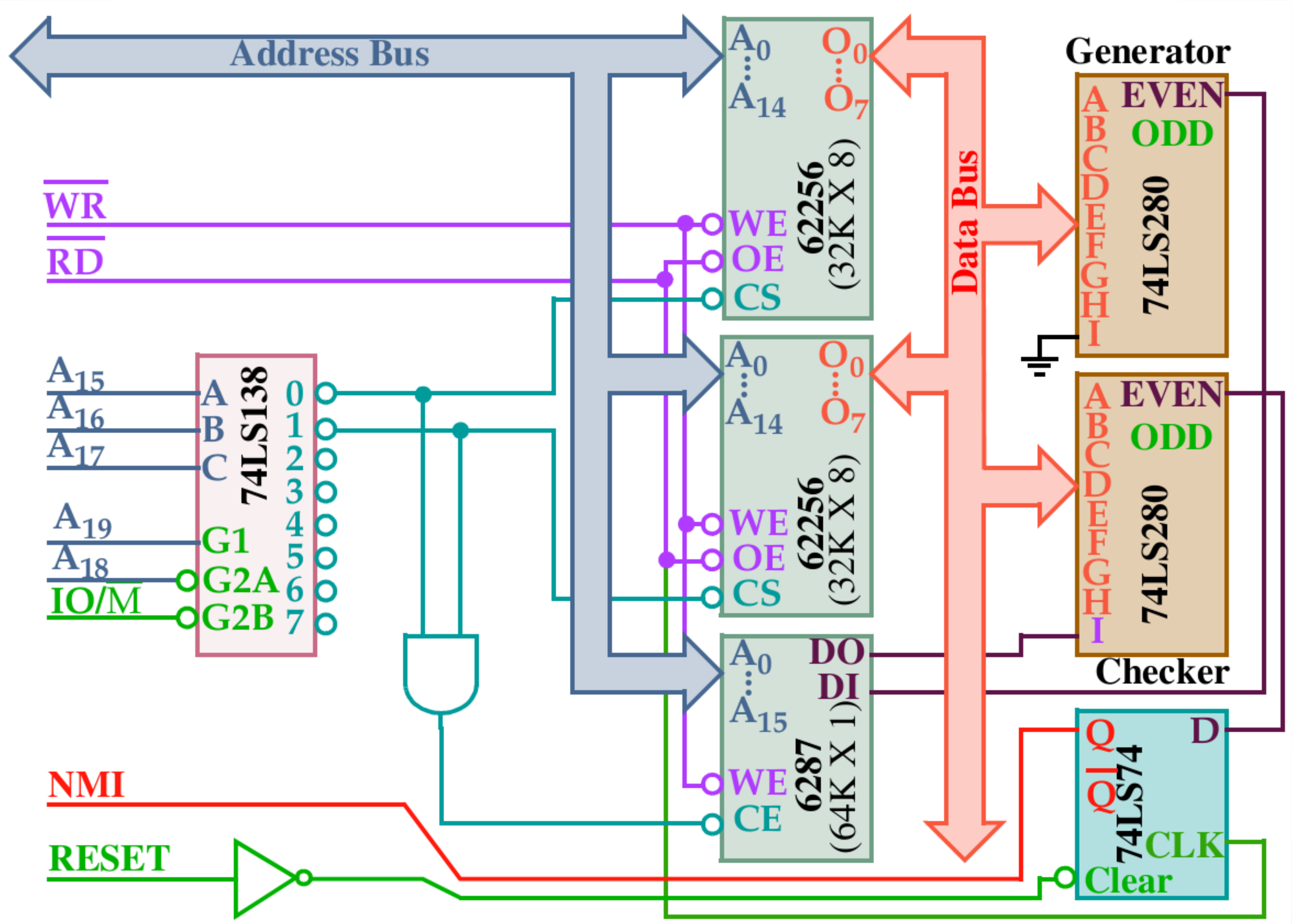
9-bit parity generator/checker

Number of inputs A thru I that are HIGH	Outputs EVEN	Outputs ODD
0, 2, 4, 6, 8	H	L
1, 3, 5, 7, 9	L	H

Parity for Memory Error Detection

- This circuit generates EVEN or ODD parity for the 9-bit number placed on its inputs.
- Typically, for generation, the 9th input bit is set to 0.
- This circuit also checks EVEN or ODD parity for the 9-bit number.
- In this case, the 9th input bit is connected to the 9th bit of memory.
- For example, if the original byte has an even # of 1's (with 9th bit at GND), the parity bit is set to 1 (from the EVEN output).
- If the EVEN output goes high during the check, then an error occurred.

Parity for Memory Error Detection

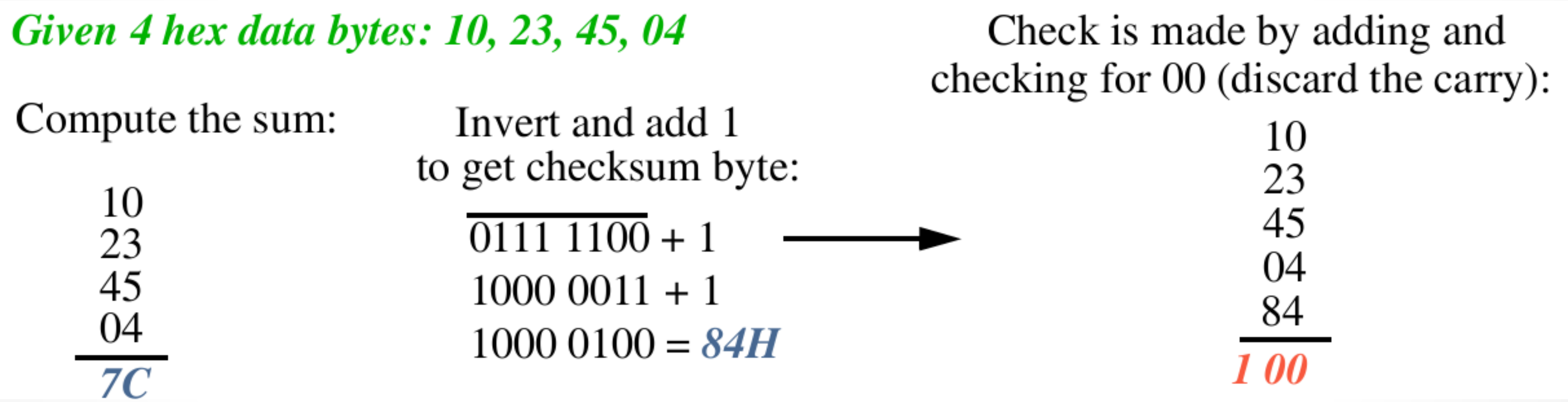


Checksum Error Detection

- This parity scheme can only detect a single bit error.
- Block-Check Character (BCC) or Checksum can detect multiple bit errors.
- This is simply the two's complement sum (the negative of the sum) of the sequence of bytes.
- No error occurred if adding the data values and the checksum produces a 0.

Checksum Error Detection

- For example



Given 4 hex data bytes: 10, 23, 45, 04

Compute the sum:

10
23
45
04

7C

Invert and add 1
to get checksum byte:

$\overline{0111\;1100} + 1$

$1000\;0011 + 1$

$1000\;0100 =$ ***84H***

Check is made by adding and
checking for 00 (discard the carry):

10
23
45
04
84

1 00

- If 45 changes to 44 AND 04 changes to 05, the error is missed.

CRC Error Detection

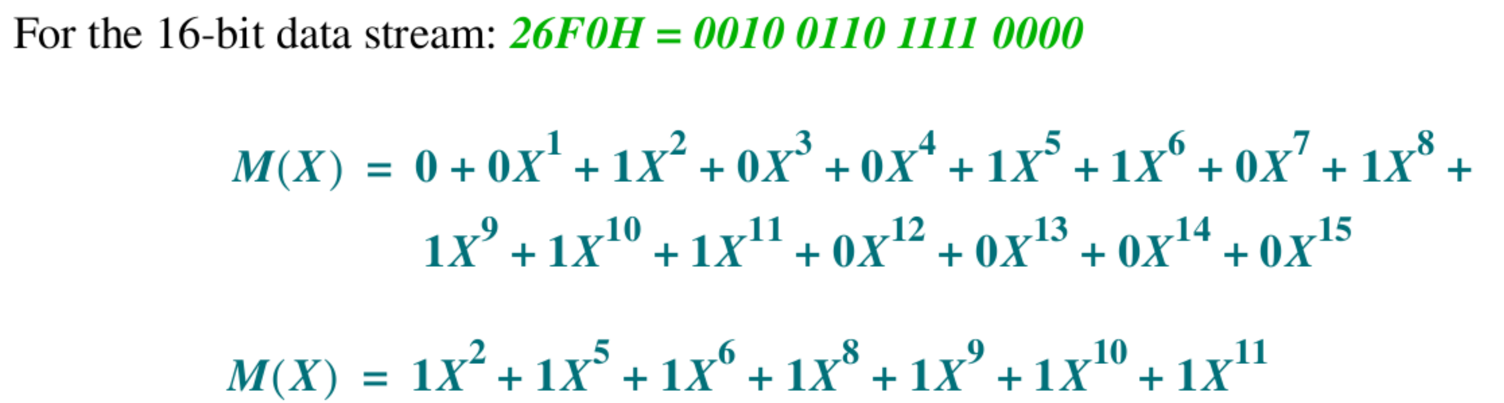
- Cyclic Redundancy Check (CRC) is commonly used to check data transfers in hardware such as hard drives.
- Treats data as a stream of serial data n-bits long.
- The bits are treated as coefficients of a characteristic polynomial, M(X) of the form:

$$M(X) = b_n + b_{n-1}X + b_{n-2}X^2 + \ldots + b_1X^{n-1} + b_0X^n$$

where b_0 is the least significant bit while b_n is the most significant bit.

CRC Error Detection

- For example

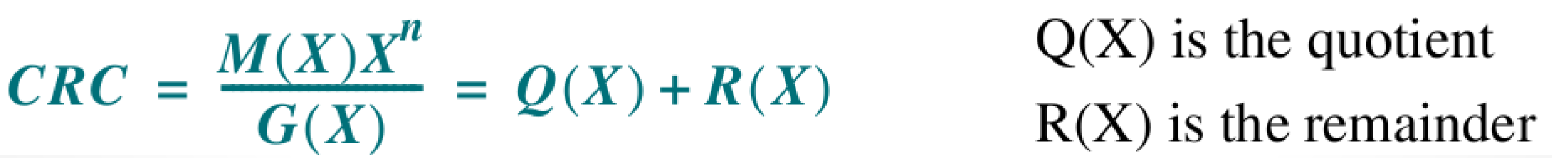


For the 16-bit data stream: *26F0H = 0010 0110 1111 0000*

$$M(X) = 0 + 0X^1 + 1X^2 + 0X^3 + 0X^4 + 1X^5 + 1X^6 + 0X^7 + 1X^8 + 1X^9 + 1X^{10} + 1X^{11} + 0X^{12} + 0X^{13} + 0X^{14} + 0X^{15}$$

$$M(X) = 1X^2 + 1X^5 + 1X^6 + 1X^8 + 1X^9 + 1X^{10} + 1X^{11}$$

- The CRC is found by applying the following equation



$$CRC = \frac{M(X)X^n}{G(X)} = Q(X) + R(X)$$

Q(X) is the quotient

R(X) is the remainder

- G(X)is the called the generator polynomial and has special properties.

CRC Error Detection

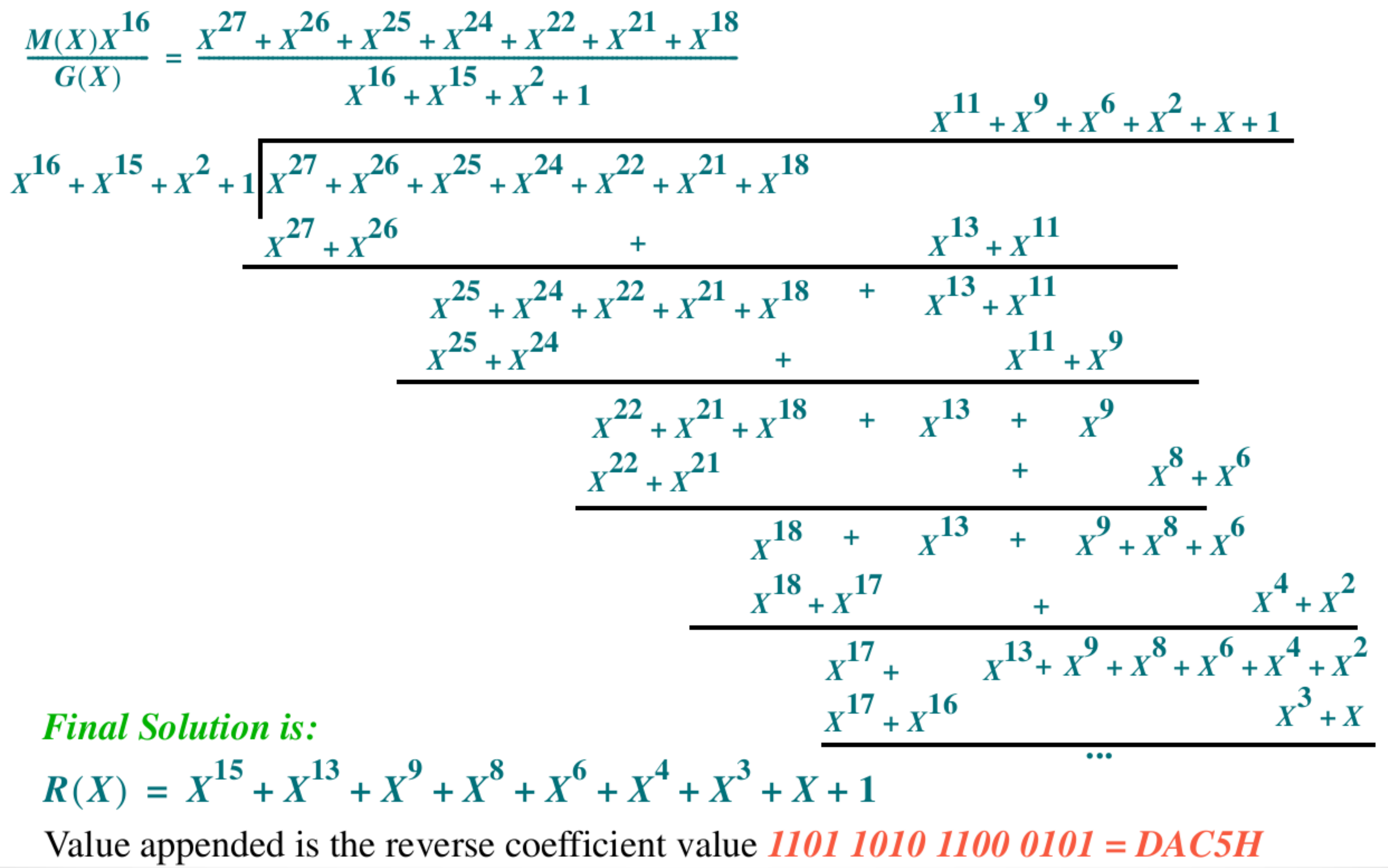
- A commonly used polynomial is:

$$G(X) = X^{16} + X^{15} + X^{2} + 1$$

- The remainder R(X) is appended to the data block.
- When the CRC and R(X)is computed by the receiver, R(X)should be zero.
- Since G(X)is of power 16, the remainder, R(X), cannot be of order higher than 15.
- Therefore, no more than 2 bytes are needed independent of the data block size.

CRC Error Detection Example

- For the data stream 26F0H:



$$\frac{M(X)X^{16}}{G(X)} = \frac{X^{27}+X^{26}+X^{25}+X^{24}+X^{22}+X^{21}+X^{18}}{X^{16}+X^{15}+X^{2}+1}$$

$$\begin{array}{r|l}
 & X^{11}+X^{9}+X^{6}+X^{2}+X+1 \\ \hline
X^{16}+X^{15}+X^{2}+1 & X^{27}+X^{26}+X^{25}+X^{24}+X^{22}+X^{21}+X^{18} \\
 & X^{27}+X^{26} \quad + \quad X^{13}+X^{11} \\ \cline{2-2}
 & X^{25}+X^{24}+X^{22}+X^{21}+X^{18} \quad + \quad X^{13}+X^{11} \\
 & X^{25}+X^{24} \quad + \quad X^{11}+X^{9} \\ \cline{2-2}
 & X^{22}+X^{21}+X^{18} \quad + \quad X^{13} \quad + \quad X^{9} \\
 & X^{22}+X^{21} \quad + \quad X^{8}+X^{6} \\ \cline{2-2}
 & X^{18} \quad + \quad X^{13} \quad + \quad X^{9}+X^{8}+X^{6} \\
 & X^{18}+X^{17} \quad + \quad X^{4}+X^{2} \\ \cline{2-2}
 & X^{17} \quad + \quad X^{13}+X^{9}+X^{8}+X^{6}+X^{4}+X^{2} \\
 & X^{17}+X^{16} \quad + \quad X^{3}+X \\ \cline{2-2}
 & \ldots
\end{array}$$

Final Solution is:

$$R(X) = X^{15}+X^{13}+X^{9}+X^{8}+X^{6}+X^{4}+X^{3}+X+1$$

Value appended is the reverse coefficient value *1101 1010 1100 0101 = DAC5H*

Error Correction

- Parity, BCC and CRC are only mechanisms for error detection.
- The system is halted if an error is found in memory.
- Error correction is starting to show up in new systems.
- SDRAM has ECC (Error Correction Code).
- Correction will allow the system to continue its operation.
- If two errors occur, they can be detected but not corrected.
- Error correction will of course cost more in terms of extra bits.

Hamming Codes

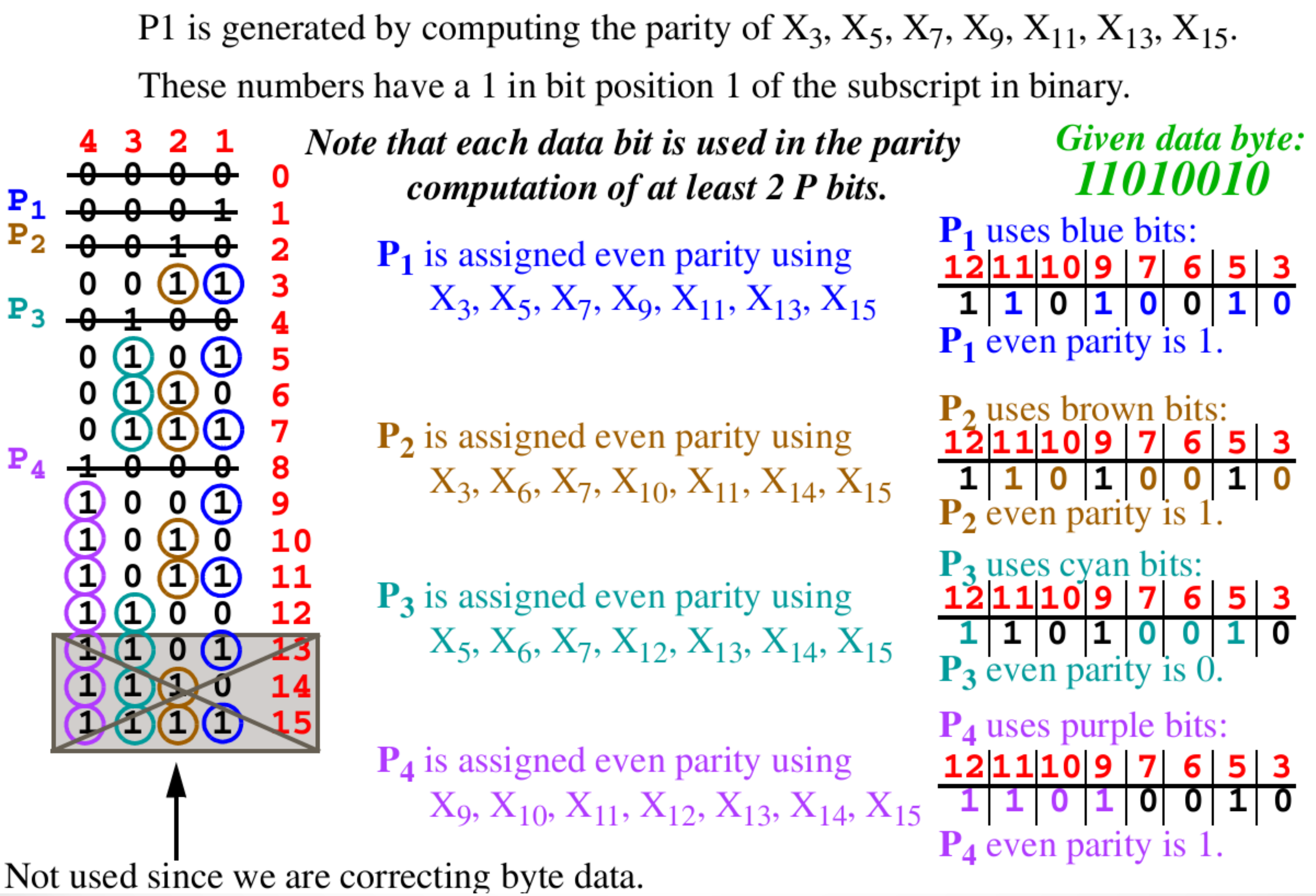
- Error correction is based on Hamming Codes.
- There is lots of theory here but our focus will be on implementation.
- The objective is to correct single bit errors in an 8-bit data byte.
- We need 4 parity bits to correct single bit errors.
- Note that the parity bits are at bit positions that are powers of 2.
- The data bits of the byte are labeled X_3, X_5, X_6, X_7, X_9, X_{10}, $X_{11}$and X_{12}.
- The parity bits are labeled P_1, P_2, P_4 and P_8.

Calculating the Hamming Code

The key to the Hamming Code is the use of extra parity bits to allow the identification of a single error. Create the code word as follows:

- Mark all bit positions that are powers of two as parity bits.(positions 1, 2, 4, 8, 16, etc)
- All other bit positions are for the data to be encoded. (positions 3, 5, 6, 7, 9, 10, 11, 12, 13, 14, 15, 17, etc.)
- Each parity bit calculates the parity for some of the bits in the code word. The position of the parity bit determines the sequence of bits that it alternately checks and skips.
  Position 1: check 1 bit, skip 1 bit, check 1 bit, skip 1 bit, etc. (1, 3, 5, 7, 9, 11, 13, 15,...)
  Position 2: check 2 bits, skip 2 bits, check 2 bits, skip 2 bits, etc.(2, 3, 6, 7, 10, 11, 15,...)
  Position 4: check 4 bits, skip 4 bits, check 4 bits, skip 4 bits, etc. (4, 5, 6, 7, 12, 13, 14, 15, 20, 21, 22, 23, ...)
  Position 8: check 8 bits, skip 8 bits, check 8 bits, skip 8 bits, etc. (8-15, 24-31, 40-47,...)
- Set a parity bit to 1 if the total number of ones in the positions it checks is odd. Set a parity bit to 0 if the total number of ones in the positions it checks is even.

Hamming Code Example



P1 is generated by computing the parity of $X_3, X_5, X_7, X_9, X_{11}, X_{13}, X_{15}$.

These numbers have a 1 in bit position 1 of the subscript in binary.

	4	3	2	1	
	~~0~~	~~0~~	~~0~~	~~0~~	0
P_1	~~0~~	~~0~~	~~0~~	~~1~~	1
P_2	~~0~~	~~0~~	~~1~~	~~0~~	2
	0	0	1	1	3
P_3	~~0~~	~~1~~	~~0~~	~~0~~	4
	0	1	0	1	5
	0	1	1	0	6
	0	1	1	1	7
P_4	~~1~~	~~0~~	~~0~~	~~0~~	8
	1	0	0	1	9
	1	0	1	0	10
	1	0	1	1	11
	1	1	0	0	12
	1	1	0	1	13
	1	1	1	0	14
	1	1	1	1	15

Not used since we are correcting byte data.

Note that each data bit is used in the parity computation of at least 2 P bits.

Given data byte:
11010010

P_1 is assigned even parity using $X_3, X_5, X_7, X_9, X_{11}, X_{13}, X_{15}$

P_1 uses blue bits:

12	11	10	9	7	6	5	3
1	1	0	1	0	0	1	0

P_1 even parity is 1.

P_2 is assigned even parity using $X_3, X_6, X_7, X_{10}, X_{11}, X_{14}, X_{15}$

P_2 uses brown bits:

12	11	10	9	7	6	5	3
1	1	0	1	0	0	1	0

P_2 even parity is 1.

P_3 is assigned even parity using $X_5, X_6, X_7, X_{12}, X_{13}, X_{14}, X_{15}$

P_3 uses cyan bits:

12	11	10	9	7	6	5	3
1	1	0	1	0	0	1	0

P_3 even parity is 0.

P_4 is assigned even parity using $X_9, X_{10}, X_{11}, X_{12}, X_{13}, X_{14}, X_{15}$

P_4 uses purple bits:

12	11	10	9	7	6	5	3
1	1	0	1	0	0	1	0

P_4 even parity is 1.

Hamming Code Example

Parity encoded data:

110110010011

If X_{10} flips from 0 -> 1, then the check gives the location of the bit error as:

P	12	11	10	9	7	6	5	3
1	1	1	1	1	0	0	1	0
1	1	1	1	1	0	0	1	0
0	1	1	1	1	0	0	1	0
1	1	1	1	1	0	0	1	0

Flipped

P_1 even parity is 0.

P_2 even parity is now 1.

P_3 even parity is 0.

P_4 even parity is now 1.

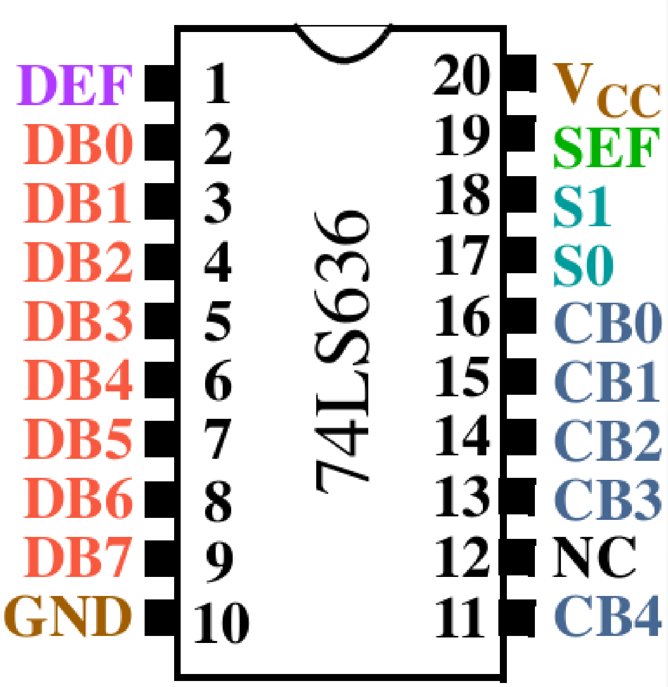
Since these are NOT 0, there was an error.

The position of the bit flip is given by:

$P_4P_3P_2P_1$,which is 1010 or 10 decimal.

Parity for Memory Error Correction

- The 74LS636 corrects errors by storing 5 parity bits with each byte of data.
- The pinout consists of: 8 data I/O pins, 5 check bit I/O pins, 2 control pins, 2 error outputs (Single error flag (SEF), Double error flag (DEF)).


74LS636
DEF 1
DB0 2
DB1 3
DB2 4
DB3 5
DB4 6
DB5 7
DB6 8
DB7 9
GND 10
20 V_{CC}
19 SEF
18 S1
17 S0
16 CB0
15 CB1
14 CB2
13 CB3
12 NC
11 CB4