CMPE - 310

Lecture 07 – Memory II

Outline

Memory Address Decoding (*NAND, 3-to-8 Line Decoder, PLD*)

Memory Interfacing

Error Detection Methods (Parity, BCC, CRC)

Error Correction (Hamming Distance)

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.

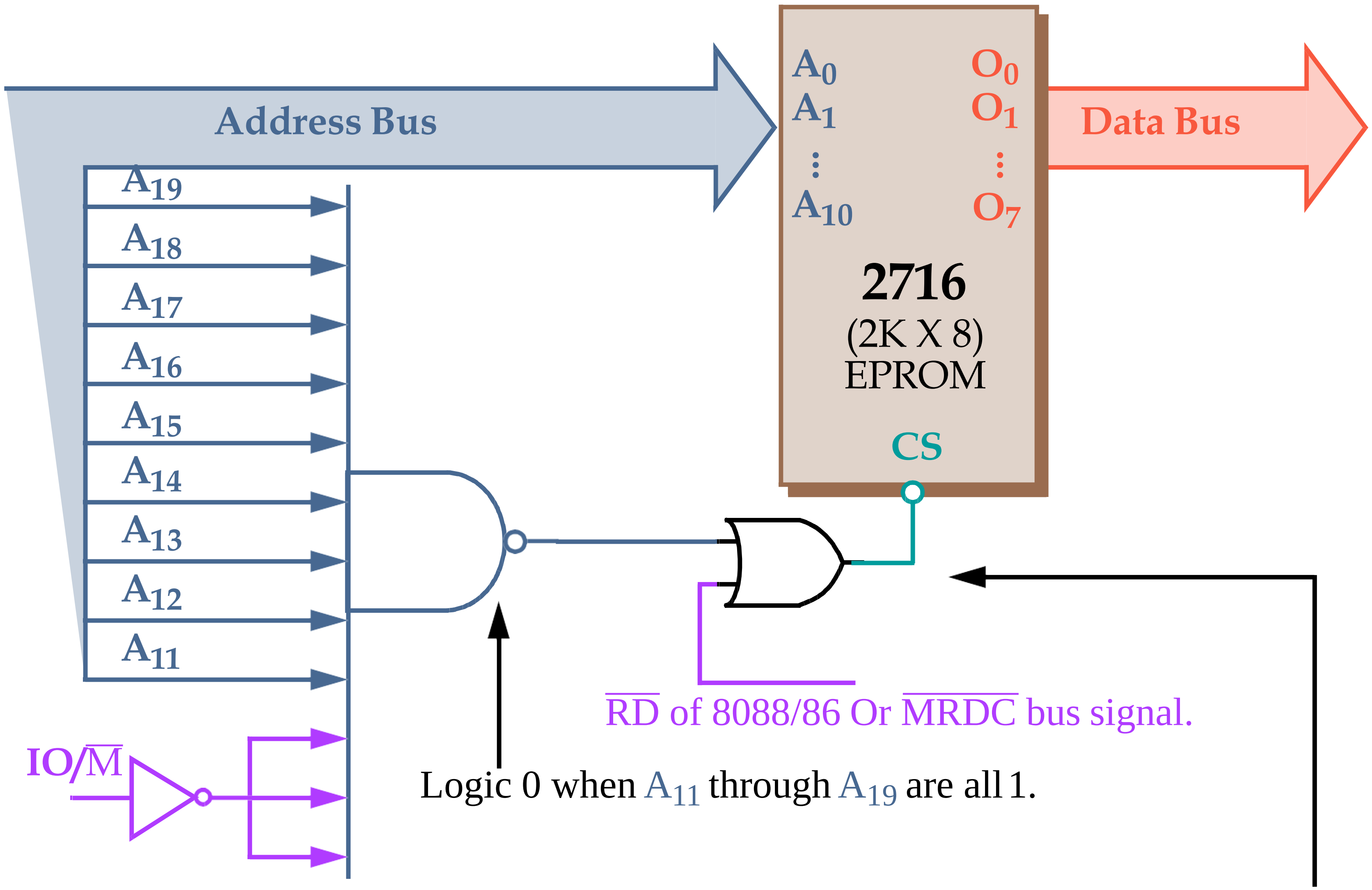
In order to splice a memory device into the address space of the processor, decoding is necessary.

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 *11* address 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.

Memory Address Decoding



(Book shows $\overline{OE}$ connection for $\overline{RD}$ but chip definition does NOT have this pin)

Memory Address Decoding

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

$$\overbrace{\texttt{1111 1111 1}}^{A_{19} - A_{11}}\overbrace{\texttt{XXX XXXX XXXX}}^{A_{10} - A_{0}}$$

$$\downarrow$$

1111 1111 1000 0000 0000 (FF800H)

$$\downarrow \text{To}$$

1111 1111 1111 1111 1111 (FFFFFH)

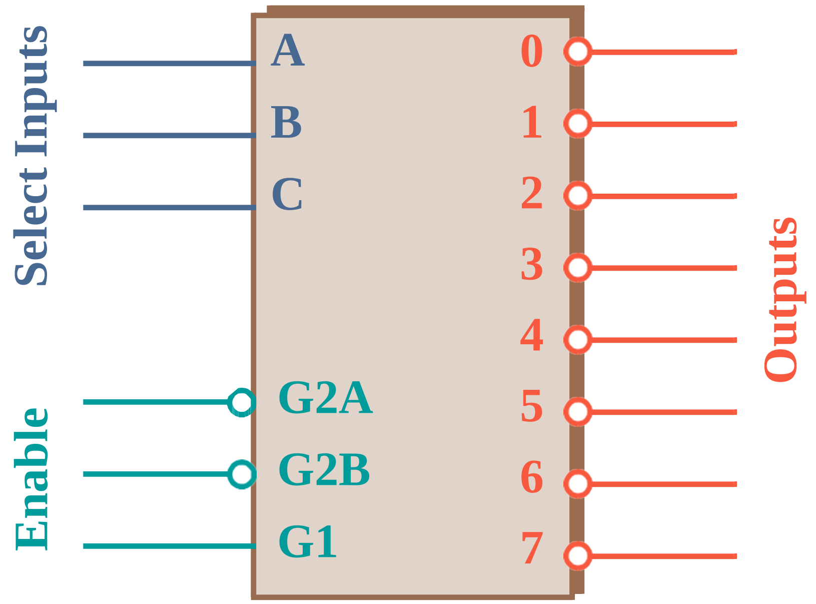
This 2KB memory segment maps into the ***reset location*** of the 8086/8088 (FFFF0H).

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.

Memory Address Decoding

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



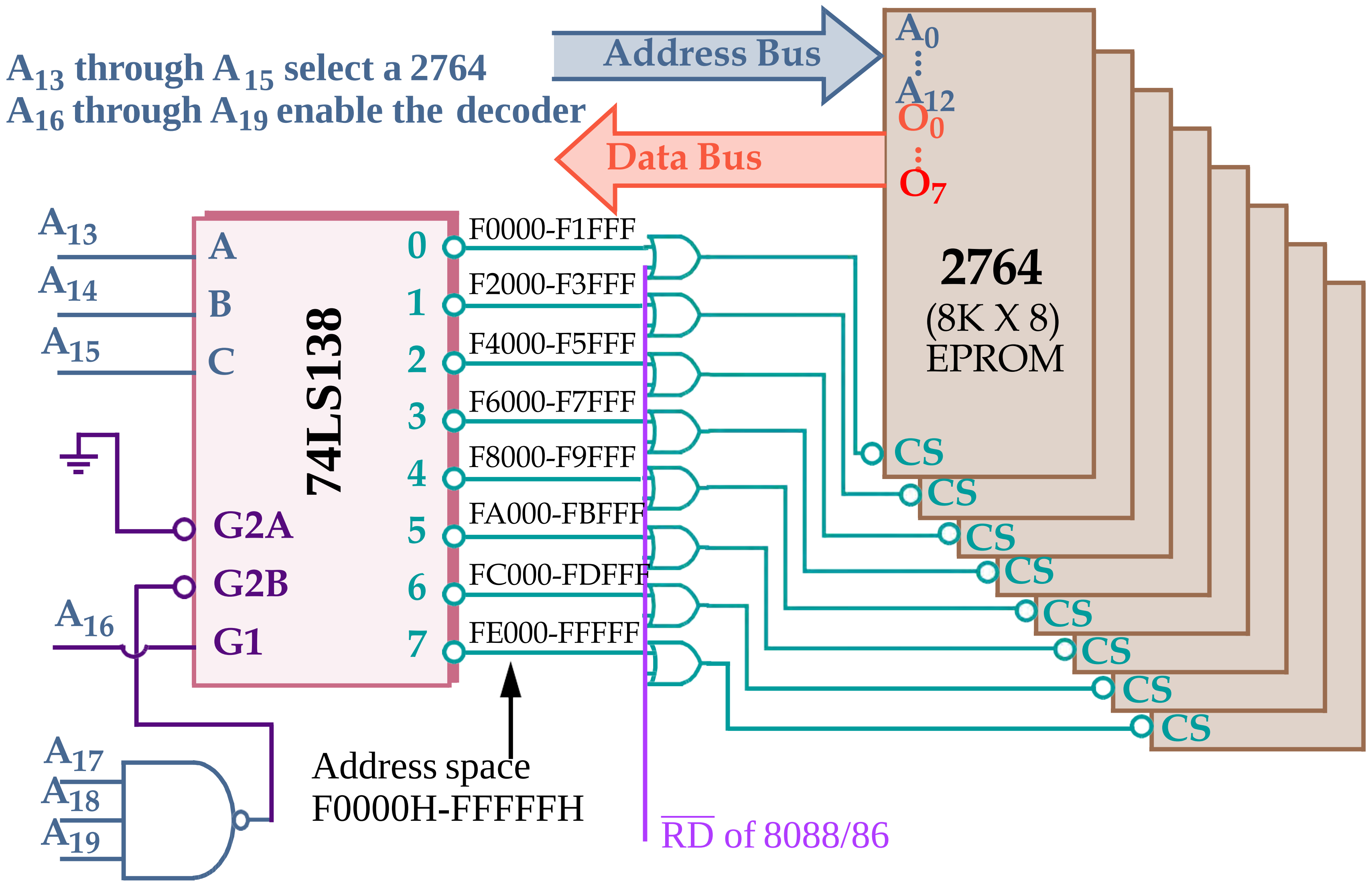
Inputs						**Output**							
Enable			**Select**										
G2A	**G2B**	**G1**	**C**	**B**	**A**	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**
1	X	X	X	X	X	1	1	1	1	1	1	1	1
X	1	X	X	X	X	1	1	1	1	1	1	1	1
X	X	0	X	X	X	1	1	1	1	1	1	1	1
0	**0**	**1**	**0**	**0**	**0**	**0**	1	1	1	1	1	1	1
0	**0**	**1**	**0**	**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**

Note that all *three* Enables (G2A, G2B, and G1) must be active, e.g. low, low and high, respectively.

Each output of the decoder can be attached to an 2764 EPROM (*8K X 8*).

Memory Address Decoding

A_{13} through A_{15} select a 2764
A_{16} through A_{19} enable the decoder



The EPROMs cover a 64KB section of memory.

Memory Address Decoding

Yet a third possibility is a ***PLD (Programmable Logic Device)***.

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 are ***fuse-programmed*** (like the PROM).

Some are erasable (like the EPROM).

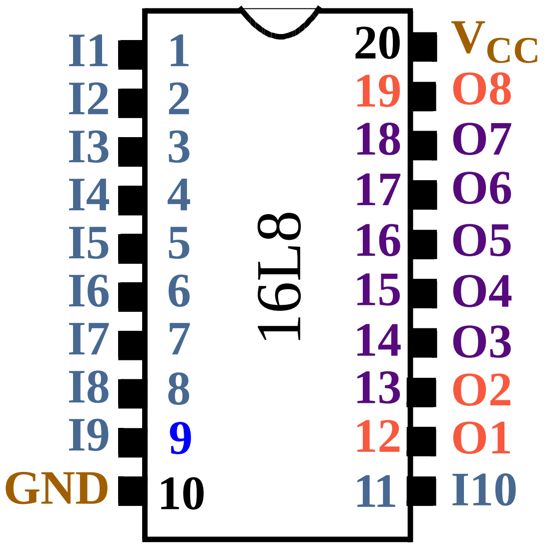
A PAL example (16L8) is shown in the text and is commonly used to decode the memory address, particularly for 32-bit addresses generated by the 80386DX and above.

Memory Address Decoding

AMD 16L8 PAL decoder.

It has 10 fixed inputs (Pins 1-9, 11), two fixed outputs (Pins 12 and 19) and 6 pins that can be either (Pins 13-18).

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

AND/NOR device with logic expressions (outputs) with up to 16 ANDed inputs and 7 ORed product terms.

8088 and 80188 (8-bit) Memory Interface

The memory systems *sees* the 8088 as a device with:

- *20* address connections (A19 to A0).
- *8* data bus connections (AD7 to AD0).
- *3* control signals, IO/M, RD, and WR.

We'll look at interfacing the 8088 with:

- *32K* of EPROM (at addresses F8000H through FFFFFH).
- *512K* of SRAM (at addresses 00000H through 7FFFFH).

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.

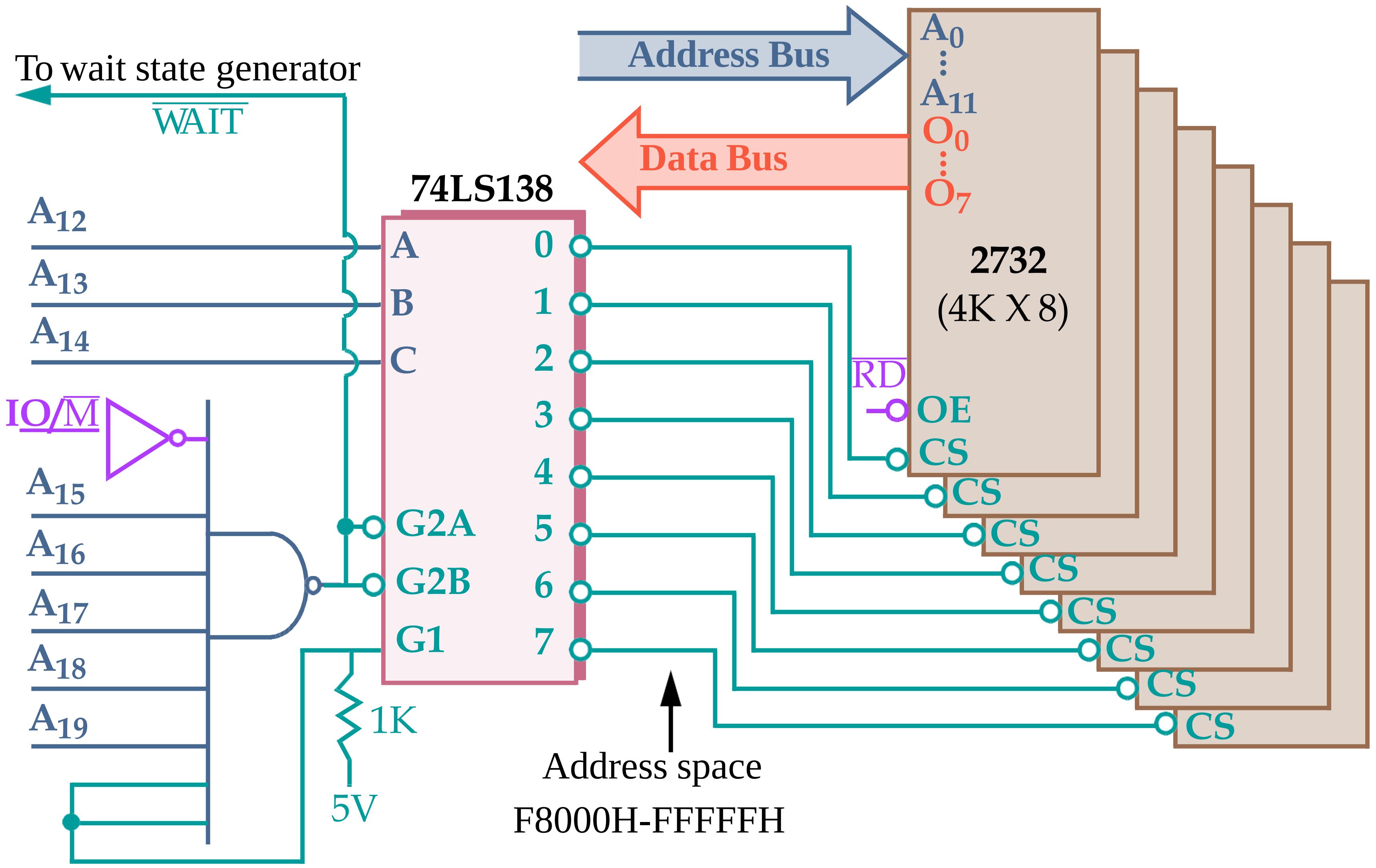
The EPROM has an access time of *450ns*.

The 74LS138 requires *12ns* to decode.

The 8088 runs at 5MHz and only allows *460ns* for memory to access data.

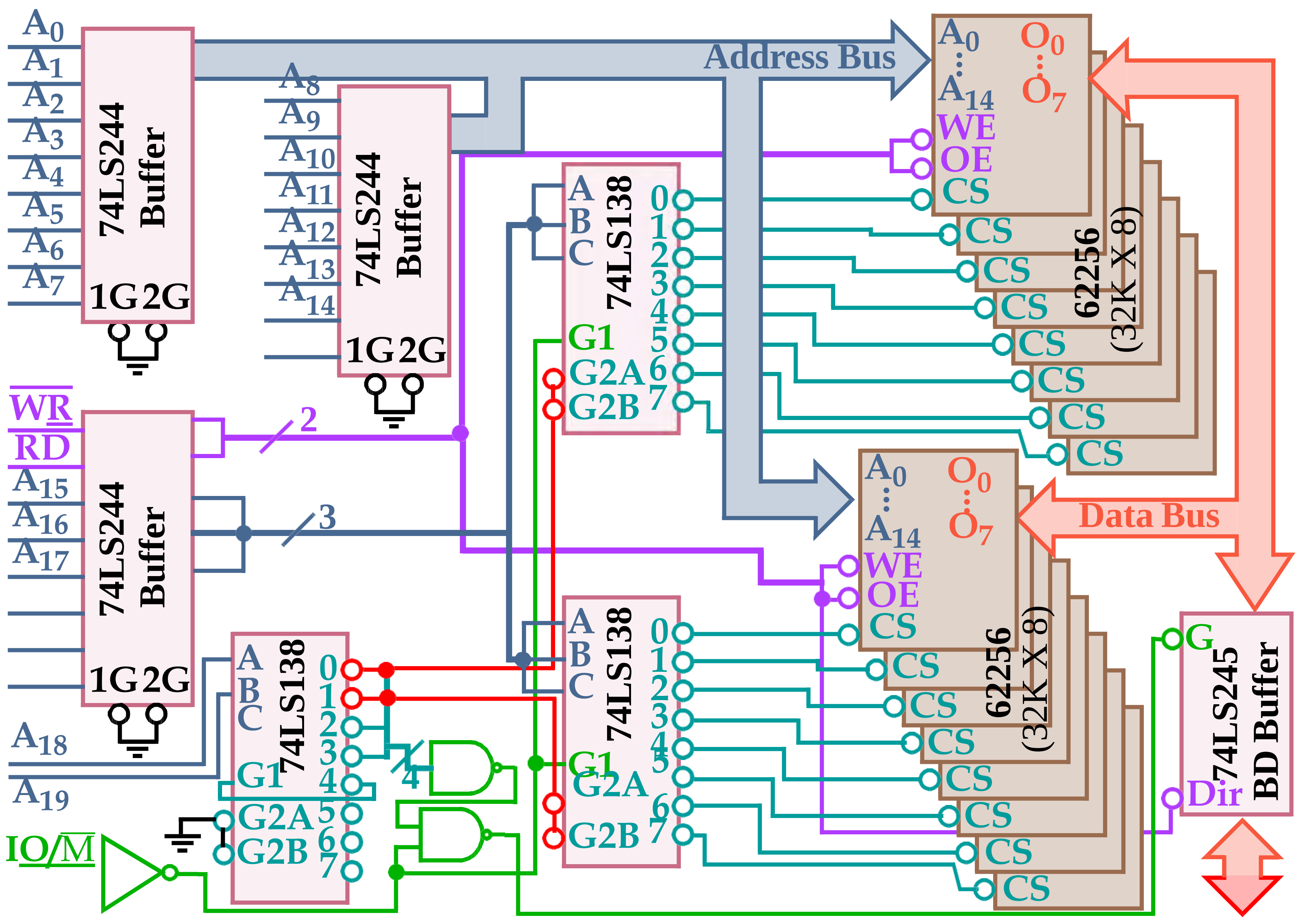
A wait state adds *200ns* of additional time.

8088 and 80188 (8-bit) EPROM Memory Interface



The 8088 cold starts execution at **FFFF0H**. JMP to F8000H occurs here.

8088 and 80188 (8-bit) RAM Memory Interface



8088 and 80188 (8-bit) RAM Memory Interface

The **16** 62256s on the previous slide are actually SRAMs.

Access times are on order of *10ns*.

Flash memory can also be interfaced to the 8088 (see text).

However, the write time (*400ms*!) is too slow to be used as RAM (as shown in the text).

Parity Checking

Parity checking is used to detect single bit errors in the memory.

The current trend is away from parity checking.

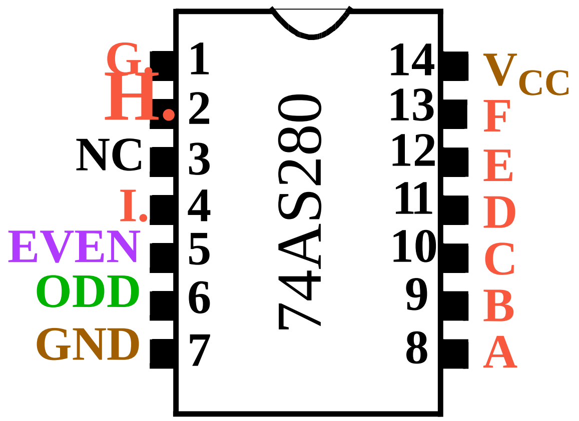
Parity checking adds **1** bit for every **8** data bits.

- 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 (**4** parity 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

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.

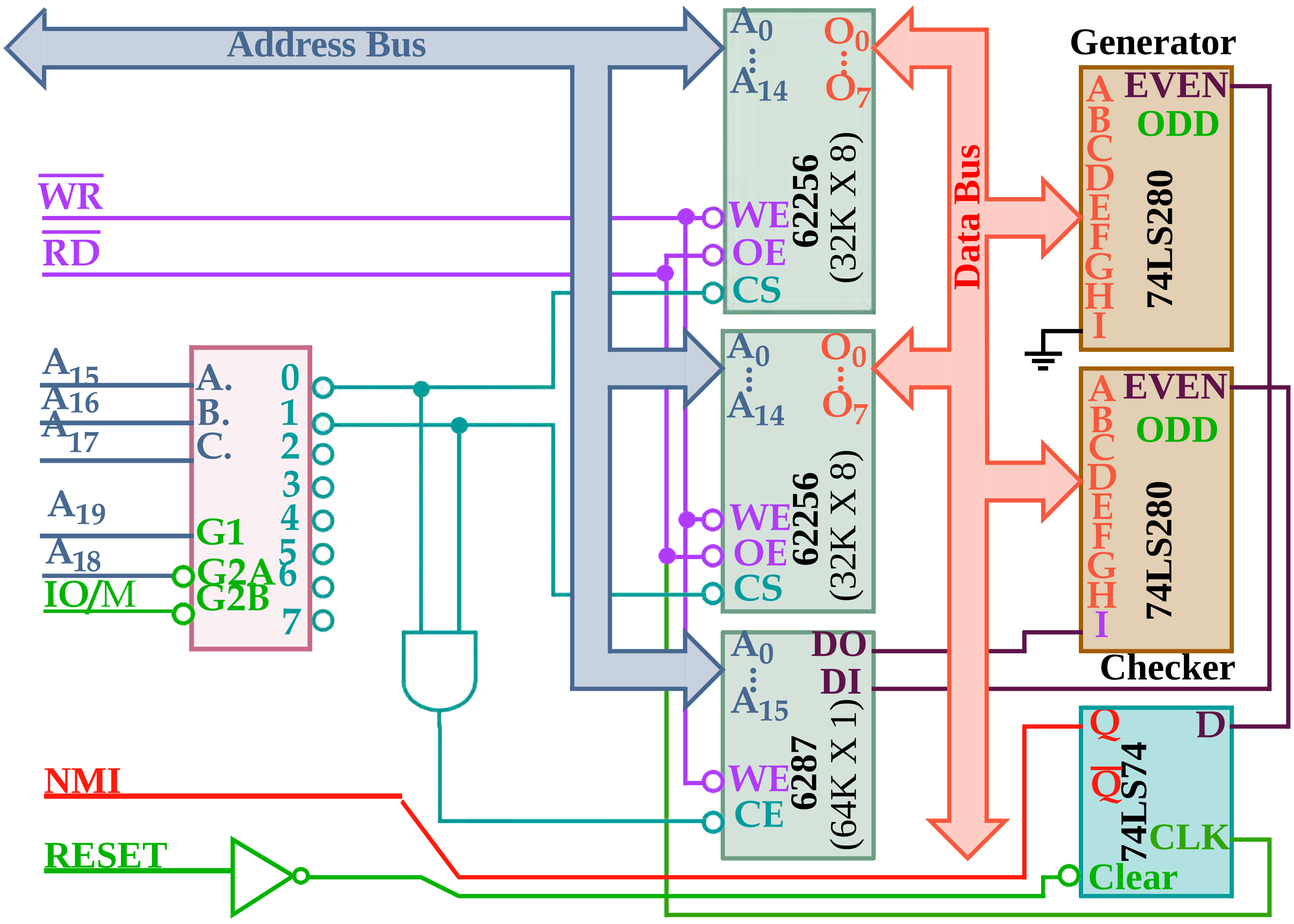
This circuit also checks *EVEN* or *ODD* parity for the 9-bit number.

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



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.

For example:

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

Compute the sum:

$$\begin{array}{r} 10 \\ 23 \\ 45 \\ \underline{04} \\ \mathit{7C} \end{array}$$

Invert and add 1
to get checksum byte:

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

$1000\ 0011 + 1$

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

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

$$\begin{array}{r} 10 \\ 23 \\ 45 \\ 04 \\ \underline{84} \\ \mathit{1\ 00} \end{array}$$

This is not fool proof.

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

Error Detection

Cyclic Redundancy Check (CRC)

Commonly used to check data transfers in hardware such as harddrives.

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.

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

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

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

Error Detection

Cyclic Redundancy Check (CRC) (cont.)

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.

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.

Error Detection

Cyclic Redundancy Check (CRC) (cont.)

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

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

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 operating.

- If *two* errors occur, they can be *detected* but not *corrected*.
- Error correction will of course cost more in terms of extra bits.

Error correction is based on ***Hamming Codes***.

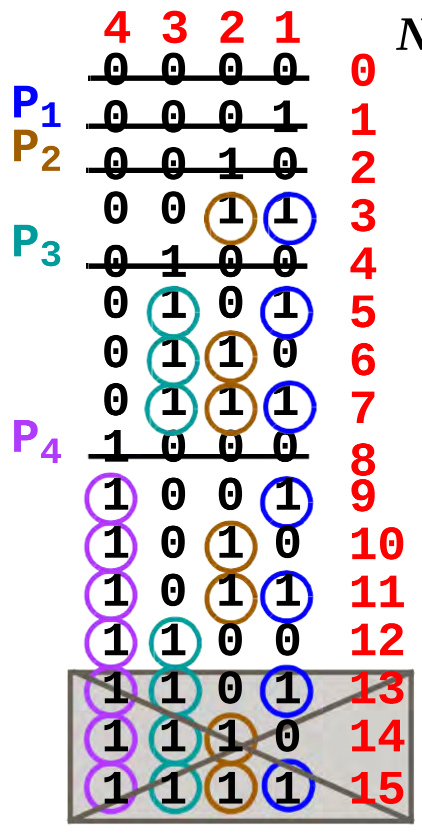
- There is lots of theory here but our focus will be on implementation.
- The objective is to correct any single bit errors in an 8-bit data byte.
  - ***The data bits of the byte are labeled*** X_3, X_5, X_6, X_7, X_9, X_{10}, X_{11} ***and*** X_{12}.
  - ***The parity bits are labeled*** P_1, P_2,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

Error Correction

Hamming Codes (cont).

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.

Error Correction

Hamming Codes (cont).

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:

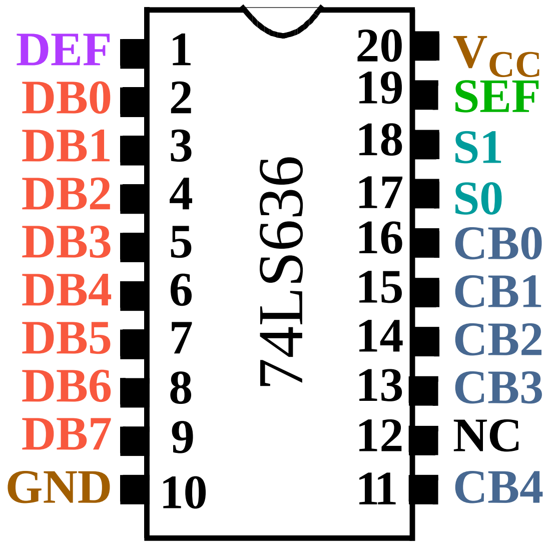
$P_4P_3P_2P_1$,which is **1010** or **10** decimal.

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**).



See the text for an example of its use in a circuit.