Quick Review of Digital Logic

Combinational Logic

- Circuits such as simple gates up through more complex mux and demux circuits fall into this category.
- Output is purely a function of input, there is no stored state to influence output.
- This is the basis of circuits within the ALU that perform arithmetic calculations and within the control region of the CPU to route data from point to point

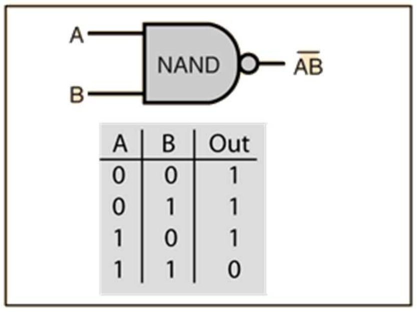
Sequential Logic

- Output is a function not only of input, but also of previous state.
- This is the basis of semiconductor memory.
- Without the ability for a circuit to remember previous state, there is no memory or storage.

SR Latch

- Simplest sequential circuit.
- Conceptually, a latch represents a circuit that stores a single bit.
- For the sake of visualization, can be built from cross-coupled NAND or NOR gates, even though as a practical matter this might not be how they are actually implemented.

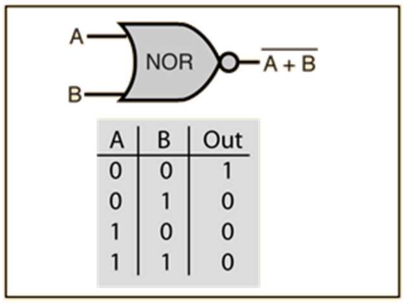
NAND Gate Truth Table



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

Figure from http://hyperphysics.phy-astr.gsu.edu/hbase/electronic/nand.html

NOR Gate Truth Table



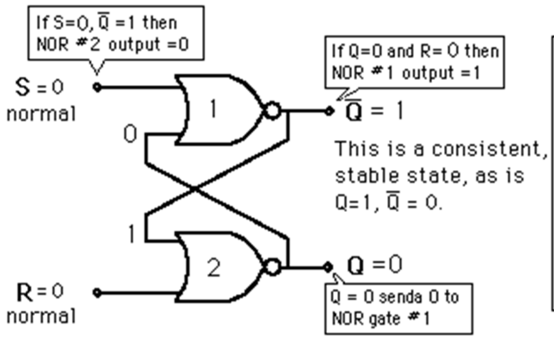
A	B	Out
0	0	1
0	1	0
1	0	0
1	1	0

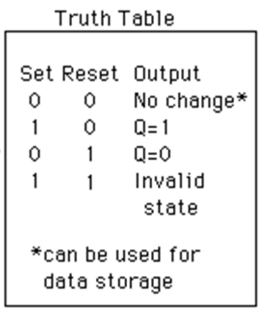
Figure taken from http://hyperphysics.phy-astr.gsu.edu/hbase/electronic/nor.html

Latches and Flip Flops

A single NAND or NOR gate are combinational circuits. But a simple design of two gates that have been **cross-coupled** changes everything, giving rise to state and storage. Latches and flip flops are the simplest forms of sequential circuit. A single latch or flip flop represents storage for a single bit. A latch is generally defined as a level triggered circuit, while flip flops are edge triggered.

Cross-Coupled NOR Gates: SR Latch


If S=0, $\overline{Q}$ =1 then NOR #2 output =0
S = 0 normal
0
1
1
2
R = 0 normal
If Q=0 and R= 0 then NOR #1 output =1
$\overline{Q}$ = 1
This is a consistent, stable state, as is Q=1, $\overline{Q}$ = 0.
Q =0
Q = 0 sends 0 to NOR gate #1


Truth Table

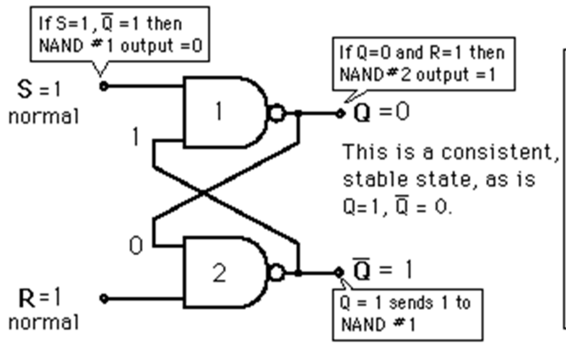
Set	Reset	Output
0	0	No change*
1	0	Q=1
0	1	Q=0
1	1	Invalid state

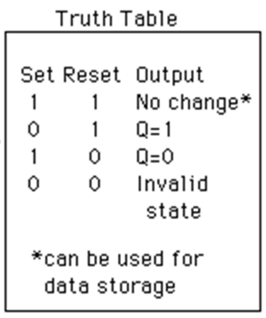
*can be used for data storage

Figure taken from http://hyperphysics.phy-astr.gsu.edu/hbase/electronic/norlatch.html

- Normal operation is for both S and R to be held low (0).
- When S becomes high (1) then returns to low (0) (while R remains low), this triggers the “set” operation, and the Q output becomes high (1). The set operation stores a 1 into Q.
- When R becomes high then returns to low (while S remains low), this triggers the “reset” operation, and the Q output becomes low (0). The reset operation stores a 0 into Q.
- S and R should never become high at the same time. Additional circuitry is required to enforce this.

Cross-Coupled NAND Gates: S-bar R-bar Latch


If S=1, $\overline{Q}$ =1 then NAND #1 output =0
S = 1 normal
1
1
0
2
R = 1 normal
If Q=0 and R=1 then NAND#2 output =1
Q =0
This is a consistent, stable state, as is Q=1, $\overline{Q}$ = 0.
$\overline{Q}$ = 1
Q = 1 sends 1 to NAND #1


Truth Table

Set	Reset	Output
1	1	No change*
0	1	Q=1
1	0	Q=0
0	0	Invalid state

*can be used for data storage

Figure taken from http://hyperphysics.phy-astr.gsu.edu/hbase/electronic/nandlatch.html

The difference in this circuit from the previous one is that S and R are normally high and are active low. Also note the relative positions of Q and Q-bar are reversed.

Types of RAM

Static RAM: SRAM

- Each bit is a latch or flip flop.
- Multiple transistors per bit.
- Each bit takes up more space on chip.
- As long as power is applied to chip, bit retains its value, no refresh circuitry required.
- Fast switching (time required to read or write the bit).
- Used when speed is more important than cost: cache

Dynamic RAM: DRAM

- Different technology based on capacitor holding charge to represent bit value
- Not based on latch or flip flop circuits
- Takes up less space, more bits can be packed onto a chip, so cheaper
- Capacitor loses charge over time so it must be periodically refreshed indefinitely, even when power is applied.
- Switching time is longer, so it's slower.
- Used when large volumes at a low price are more important than speed: main memory RAM.

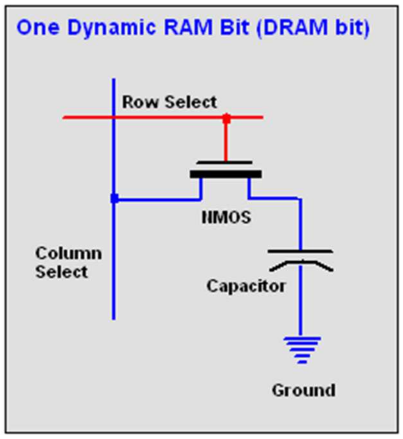


Figure taken from http://www.pcmag.com/encyclopedia/term/42192/dynamic-ram

Synchronous DRAM: SDRAM

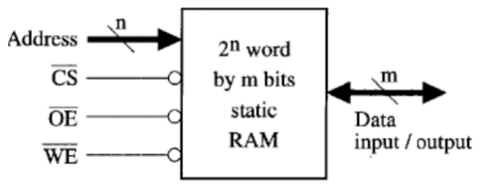
- Operation is synchronized with a clock signal
- Allows higher data transfer rates between RAM and CPU than asynchronous DRAM

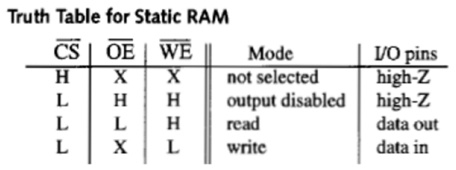
Double Data Rate SDRAM: DDR SDRAM (DDR1, DDR2, DDR3, DDR4 Generations)

- Data is transferred on both the rising and falling edge of the clock (double pumping)
- Doubles the data transfer rate without changing the clock frequency
- See https://en.wikipedia.org/wiki/DDR_SDRAM
- Example: 64 bit data bus, 100 MHz clock implies 1600 MB/sec transfer rate

RAM Logical View Versus Physical DRAM Packaging

A high-level logical view of the function of RAM treats RAM as a single large chip with address and data busses, plus a couple of control signals like the R/W line. A typical figure is the following:





Truth Table for Static RAM

$\overline{CS}$	$\overline{OE}$	$\overline{WE}$	Mode	I/O pins
H	X	X	not selected	high-Z
L	H	H	output disabled	high-Z
L	L	H	read	data out
L	X	L	write	data in

Figures taken from
http://www.allsyllabus.com/aj/note/ECE/Digital_System_Design_Using_VHDL/Unit8/index.php

The chip requires n address lines to pick one word out of a total of 2^n words. Each word is m bits, so there are m data lines for both read and write operations. The control signals are for chip select (CS), output enable (OE) and write enable (WE). The truth table shows that normal operation is for chip select to be enabled, then either output enable becomes active for read, or write enable becomes active for write.

DRAM Chip Architecture

Actual RAM (both SRAM and DRAM) as installed in a typical PC looks quite different internally. Commercial DRAM chips are grouped together into DIMMs (dual inline memory modules) for installation onto typical motherboards. Let’s spell out some of the details that show how a high level view of RAM can be built up from individual DRAM chips and DIMMs.

A single DRAM chip provides storage in the form of an array, where a small group of bits (usually 4 to 8) is uniquely addressed by a row and column. In the Stallings book, figure 5.3 shows the schematic for a typical 16 Mbit DRAM chip (emphasis on megabit, not megabyte).

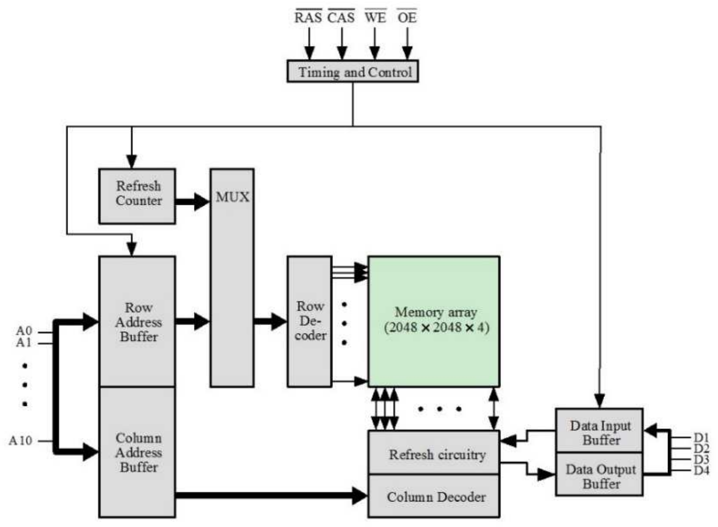
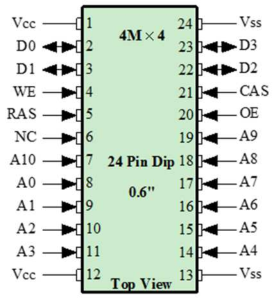


Figure 5.3 Typical 16 Megabit DRAM (4M × 4)

The notation "2048 X 2048 X 4" means "2048 rows, 2048 columns, 4 bits per address." Do the math to confirm that this adds up to 16 megabits:

$$2048 \text{ X } 2048 \text{ X } 4 = 2^{11} \text{ X } 2^{11} \text{ X } 2^{2} = 2^{24} = 16 \text{ M}$$

Another way to look at this chip: it uses a 22-bit address (11 bit row address and 11 bit column address) to select 4 bits for either read or write. Pinout looks like this (also from Stallings):



(b) 16 Mbit DRAM

Note that there are 4 data pins (D0 – D3) but only 11 address pins (A0 – A10), which seems to be only half of what is needed. But note the row address select (RAS) and column address select (CAS) pins. The RAS/CAS pin values determine the meaning of the values currently on the address lines. Using these signals, the 22-bit address is transmitted across the 11-bit address bus in two parts: the row address in one step (with the RAS signal active) and the column address in the next step (with the CAS signal active).

Banks

The chip described above consists of a single 2048 X 2048 X 4 **bank**. Larger DRAM chips take the above design and replicate it to create **4 or more identical banks within the same chip**. A chip with 4 banks will have two additional signals **BA0 and BA1 for bank address.** The 2-bit bank address then becomes part of the overall RAM address.

The accompanying schematic explains the internal architecture of a larger 4-bank chip, a 128 megabit DRAM. The chip comes in 3 configurations (all with 4 banks):

- 32 M X 4: each bank is 4096 X 2048 X 4 (4096 X 2048 X 4 X 4 = 128 megabits)
- 16 M X 8: each bank is 4096 X 1024 X 8 (4096 X 1024 X 8 X 4 = 128 megabits)
- 8 M X 16: each bank is 4096 X 512 X 16 (4096 X 512 X 16 X 4 = 128 megabits)

Bank address and row address is the same in all configurations

- Bank address: 2 bits ($2^2 = 4$)
- Row address: 12 bits ($2^{12} = 4096$).

Column address depends on the configuration

- 32 M X 4: 2048 columns = 11 bit column address
- 16 M X 8: 1024 columns = 10 bit column address
- 8 M X 16: 512 columns = 9 bit column address

Number of data bits selected is indicated by the 3rd dimension for that configuration

- 32 M X 4: 4 bits per address
- 16 M X 8: 8 bits per address
- 8 M X 16: 16 bits per address

The chip will need to provide the number data pins that matches the number of data bits per address.

Depending on how the bank address bits are used, the banks can be used to store data sequentially or interleaved. If the bank address bits are MSBs (most significant bits), sequentially addressed data will be stored sequentially across the banks, filling bank 0, then moving on to bank 1, etc. If the bank address bits are used as LSBs (least significant bits), sequentially address data will be interleaved across the banks, with one address in bank 0, the next address in bank 1, etc. The chip does not enforce one addressing approach over the other, the user of the chip can build a circuit around the chip to use one or the other.

So far, the design hierarchy of memory looks like this:

Flip-flop (1 bit) ➔ DRAM bank (e.g., 2048 X 2048 X 4 bits) ➔ DRAM chip (with 4 to 8 banks)

DIMM: Dual Inline Memory Module

Organization of storage within a single DRAM chip is based on multiple banks, each with the bits for that bank selected by row and column addresses. In the past, DRAM chips might have been plugged into IC sockets directly on the motherboard. But today, it's usually provided to the end user in the form of DIMMs (dual inline memory module). The DIMM is a separate small card that plugs into slots on the motherboard. The individual DRAM chips are attached on the sides of the DIMM, and the DIMM has connection pins on one edge and snaps into place in a specially designed socket on the motherboard. Single Inline Memory Modules (SIMMs) have DRAM chips mounted on one side of the card, DIMMs have DRAM chips on two sides.

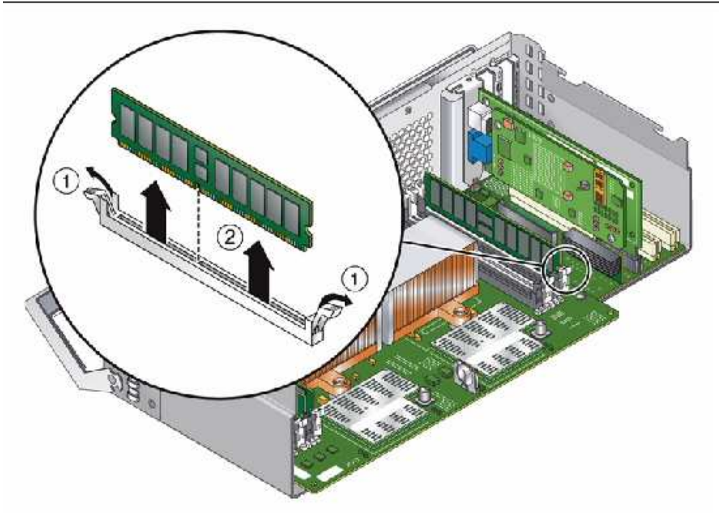


Figure from https://docs.oracle.com/cd/E19469-01/819-4359-19/figures/CH3-maint-61.jpg

DIMMs provide the interface between the small set of bits provided by each DRAM chip (4 to 8 bits) and the data bus, which is usually 64 bits currently. For example, if a DRAM chip provides 8 bits per read, but the motherboard has a 64-bit data bus, then ideally we should install 8 DRAM chips together to act as a single larger "virtual" DRAM chip with 8 X 8 = 64 bits for each read or write. Now the capacity of the DRAM chips matches the capacity of the data bus. If we send the same address to all chips simultaneously and access their selected bits simultaneously, then we have effectively created a larger storage word spread across multiple DRAM chips, and we can take advantage of the wide data path provided by the data bus. There are many ways to combine these individual DRAM chips to provide a complete logical memory function to our computer system, but one of the most common today is called the DIMM. Previously, DRAM was supplied on a simpler module called a SIMM (single inline memory module).

A more close up figure:

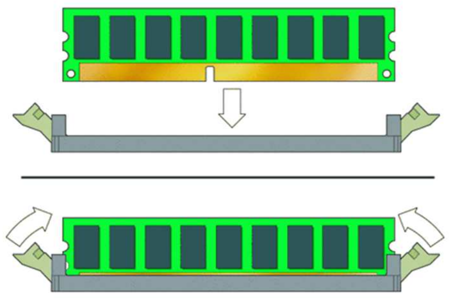


Figure from https://docs.oracle.com/cd/E19127-01/ultra27.ws/820-6776/images/24051.gif

Note the notch on the bottom of the DIMM. One or more notches imply different DIMM generations and characteristics that are not forward or backward compatible with other kinds of DIMM. To add DIMMs to a motherboard requires careful research of what kind of DIMM is compatible with that motherboard. Some motherboards have requirements that multiple DIMMS be installed in specific capacity combinations. For example, some older motherboards with 4 DIMM slots required that all DIMM slots be filled, you could not install a DIMM into a couple of slots and leave the other slots empty.

Ranks

The next level in the hierarchy of memory storage is the **rank**. A rank consists of some subset of the DRAM chips attached to the DIMM. Most commonly these days, all the DRAM chips on one side of the DIMM form a **rank**. Rank 0 is on one side of the DIMM, and rank 1 is on the other. A typical arrangement is for 8 DRAM chips to form one rank. Two ranks imply 16 chips per DIMM.

Hierarchy: flip-flop ➔ bank ➔chip ➔rank ➔DIMM

DIMM Capacity

Using the DIMM design with 8 chips per side or 16 chips total, the capacity of a single DIMM in megabytes as a function of DRAM capacity in megabits can be calculated as follows:

DRAM Size	**DIMM Capacity**
128 Mb	2^{27} X 2^4 = 2^{31} bits; $2^{31}/2^3$ bytes = 2^{28} bytes = 2^8 * 2^{20} = 256 MB
256 Mb	512 MB
512 Mb	1 GB

ECC

Error correcting code DRAM will typically add a 9th chip for SEC/DED codes for 8 data chips.

Exercise: how many SEC bits are needed for 64 data bits?

Error Detecting and Correcting Codes

- Imagine a 4-bit data item 1001.
- Store it into RAM or transmit it along a communication channel.
- Suppose during storage or transmission the data is corrupted in one bit and becomes 1011.
- What happens on read or receive?

Parity Codes

- We make the assumption that no more than one bit can change during storage or transmission.
- When the data is first examined, we calculate a parity bit, so the total number of bits becomes 5
    - 4 data bits
    - 1 parity bit
- The parity bit is redundant information that uses extra resources to maintain, but it is worth it because it gives us the ability to detect a single-bit error.
    - Original Data: 1001
    - Common parity scheme: parity bit is
    - 0 if there are an even # of 1s in the data
    - 1 if there are an odd # of 1s in the data
    - In this scheme, the total # of 1s for the data + parity bit will always be even.
- We can now store or transmit all 5 bits.
- When the data is read or received, we recalculate the parity bit.
- If the parity bits match, there are two possibilities:
    - There are no errors.
    - There are two or more errors.
- The parity bit only works if we assume a maximum of one error, so in this case we will go forward assuming the data bits are correct.
- If the parity bits don't match, there is one possibility:
    - There are one or more errors.
- In this scheme, we can't correct the error, but simply detecting the error is better than not knowing there was an error at all.

Checksum

- A generalization of the parity bit is the checksum.
- A checksum is a function of the data that produces a new value that then accompanies the original data.
- $C_{i-j} = f(d_{k-l})$: some number of check bits are calculated as a function of the data bits.
- When the data and originally calculated checksum are received, the checksum is recalculated.
- If the two checksums match, we have higher confidence that the received data has not been modified or corrupted.
- The checksum itself could be corrupted, but since the checksum is small compared to the data, any errors are more likely to be within the data.

Hamming Codes

- Richard Hamming invented a coding method in the 1950s to detect 1 or 2 errors and correct 1 error. It relies on the concept of code words and distances.
- A **code word** is a **valid** set of bits for the coding scheme. It contains a mix of data and checksum bits.
- An arbitrary bit pattern may or may not be a valid code word.
- Code distance = number of bits that must be modified (flipped) to convert one bit pattern into a second bit pattern.
- For **error detection**, we require any two valid code words to be at least **distance two** apart: a single bit error must not convert one valid code word into another code word.
- For **error correction**, we further require that valid code words be at least **distance three** apart: a single bit error will move to a bit pattern that is distance one from the original valid code word. When the error code is encountered later, there is only one valid code word that is distance one away, so the error can be corrected.
- Some useful abbreviations
    - SED: single error detection code
    - DED: double error detection code
    - SEC: single error correction code

SEC: How Many Bits Are Needed?

Let's go through the SEC (single error correction) Hamming code algorithm. Before looking at the detailed algorithm, we have to figure out the minimum number of extra bits would be required to provide error correction capability. We have already seen that a single parity bit is not enough.

- Suppose we have n data bits and will add k check bits.
- We plan to use the k check bits to distinguish between all the possible error conditions.
- The number of bit patterns in k bits is 2^k.
- We reserve 1 bit pattern to indicate that there is no error
- This leaves $2^k – 1$ patterns for all error conditions
- If we have n data bits and k check bits, one complete word is n+k bits.
- How many single bit errors can there be?
- Each bit could be in error, so there are n+k possible errors.
- So we need the check bit patterns to be at least as big as the number of possible errors:
- $2^k – 1 >= n+k$

Here are a few representative values for n and k that satisfy this relation

n	**k**	**n+k**	**$2^k – 1$**	**$2^k – 1 >= n+k$?**
8	2	10	$2^2 – 1 = 3$	3 >= 10 ? No
8	3	11	$2^3 – 1 = 7$	7 >= 11 ? No
8	4	12	$2^4 – 1 = 15$	15 >= 12 ? Yes

Now that we know how many bits we need, we can work out the details for the (8,4) SEC Hamming code.

- Number the 12 bits from 1 to 12 going from right to left.
- The 4 check bits are associated with the positions that represent powers of 2: 1, 2, 4, and 8.
- The 8 data bits are the rest: 3, 5, 6, 7, 9, 10, 11, 12.

	12	11	10	9	8	7	6	5	4	3	2	1
Data	D_8	D_7	D_6	D_5		D_4	D_3	D_2		D_1		
Check					C_8				C_4		C_2	C_1

We'll see why it's useful to put bits in this order shortly.

Now take an 8-bit data item – $39_{16} = 00111001_2$ – and place into this table.

	12	11	10	9	8	7	6	5	4	3	2	1
Bits	0	0	1	1		1	0	0		1		
Data	D_8	D_7	D_6	D_5		D_4	D_3	D_2		D_1		
Check					C_8				C_4		C_2	C_1

How to Calculate Check Bits

- Each check bit will be a function of a subset of the data bits.
- We will carefully pick which data bits are checked by each check bit to guarantee that each data bit is checked by a unique combo of check bits.
- Each check bit occupies a position that represents a power of 2: $1=2^0$, $2=2^1$, $4=2^2$, $8=2^3$.
- Each data bit occupies the other positions 3, 5, 6, 7, 9, 10, 11, 12.
- For each of these data bit positions, express them as sums of powers of 2:
  - $3 = 0011_2 = 2^1 + 2^0$ (bit 3 = data bit 1)
  - $5 = 0101_2 = 2^2 + 2^0$ (bit 5 = data bit 2)
  - $6 = 0110_2 = 2^2 + 2^1$ (bit 6 = data bit 3)
  - $7 = 0111_2 = 2^2 + 2^1 + 2^0$ (bit 7 = data bit 4)
  - $9 = 1001_2 = 2^3 + 2^0$ (bit 9 = data bit 5)
  - $10 = 1010_2 = 2^3 + 2^1$ (bit 10 = data bit 6)
  - $11 = 1011_2 = 2^3 + 2^1 + 2^0$ (bit 11 = data bit 7)
  - $12 = 1100_2 = 2^3 + 2^2$ (bit 12 = data bit 8)

Now reverse this list to show for each power of two, which data bit positions include it in their sum:

- 2^0: check bit 1 checks data bits 1, 2, 4, 5, 7
- 2^1: check bit 2 checks data bits 1, 3, 4, 6, 7
- 2^2: check bit 4 checks data bits 2, 3, 4, 8
- 2^3: check bit 8 checks data bits 5, 6, 7, 8

The check bits are calculated as the XOR (exclusive OR) of the selected data bits.

We're working on data item $39_{16} = 00111001_2$.

	12	11	10	9	8	7	6	5	4	3	2	1
Bits	0	0	1	1		1	0	0		1		
Data	D_8	D_7	D_6	D_5		D_4	D_3	D_2		D_1		
Check					C_8				C_4		C_2	C_1

Now calculate the check bits:

- $C_1 = D_1 \oplus D_2 \oplus D_4 \oplus D_5 \oplus D_7 = 1 \oplus 0 \oplus 1 \oplus 1 \oplus 0 = 1$
- $C_2 = D_1 \oplus D_3 \oplus D_4 \oplus D_6 \oplus D_7 = 1 \oplus 0 \oplus 1 \oplus 1 \oplus 0 = 1$
- $C_4 = D_2 \oplus D_3 \oplus D_4 \oplus D_8 = 0 \oplus 0 \oplus 1 \oplus 0 = 1$
- $C_8 = D_5 \oplus D_6 \oplus D_7 \oplus D_8 = 1 \oplus 1 \oplus 0 \oplus 0 = 0$

Inserting the check bits into the code word gives:

	12	11	10	9	8	7	6	5	4	3	2	1
Bits	0	0	1	1	0	1	0	0	1	1	1	1
Data	D_8	D_7	D_6	D_5		D_4	D_3	D_2		D_1		
Check					C_8				C_4		C_2	C_1

The completed code word is now the 12 bit value $001101001111_2 = 34F_{16}$

Injecting an Error and Correcting It

To demonstrate how an SEC Hamming code works, let’s inject an error into a data bit. Let’s say D_3 (data bit 3, bit 6 in the code word counting from the right) which was originally 0, was flipped during storage to 1. The code word started off as

$001101001111_2 = 34F_{16}$

but is now

$001101101111_2 = 36F_{16}$

Looking at the code word layout with the new erroneous code word we have:

	12	11	10	9	8	7	6	5	4	3	2	1
Bits	0	0	1	1	0	1	1	0	1	1	1	1
Data	D_8	D_7	D_6	D_5		D_4	D_3	D_2		D_1		
Check					C_8				C_4		C_2	C_1

First we recalculate the Check Bits

- $C_1 = D_1 \oplus D_2 \oplus D_4 \oplus D_5 \oplus D_7 = 1 \oplus 0 \oplus 1 \oplus 1 \oplus 0 = 1$
- $C_2 = D_1 \oplus D_3 \oplus D_4 \oplus D_6 \oplus D_7 = 1 \oplus 1 \oplus 1 \oplus 1 \oplus 0 = 0$ $\Leftarrow$
- $C_4 = D_2 \oplus D_3 \oplus D_4 \oplus D_8 = 0 \oplus 1 \oplus 1 \oplus 0 = 0$ $\Leftarrow$
- $C_8 = D_5 \oplus D_6 \oplus D_7 \oplus D_8 = 1 \oplus 1 \oplus 0 \oplus 0 = 0$

D_3 is in position 6 and is checked only by C_2 and C_4, so only those check bits change

	12	11	10	9	8	7	6	5	4	3	2	1
Bits	0	0	1	1	0	1	1	0	1	1	1	1
Data	D_8	D_7	D_6	D_5		D_4	D_3	D_2		D_1		
Check					C_8				C_4		C_2	C_1
Recalc					0				0		0	1

We now take the XOR of the old check bits and the new check bits, which results in a 0 if the bit hasn’t changed, or a 1 if it has.

	12	11	10	9	8	7	6	5	4	3	2	1
Bits	0	0	1	1	0	1	1	0	1	1	1	1
Data	D_8	D_7	D_6	D_5		D_4	D_3	D_2		D_1		
Check					C_8				C_4		C_2	C_1
Recalc					0				0		0	1
Old XOR New					0				1		1	0

Note that the bits that are different form the number $0110_2 = 6$, which is the position of the bit in error. As explained above, the reason this works is that

- Each check bit represents a power of 2
- Each check bit checks those bit positions for positions that have that power of 2 in their sum.
- Each data bit is checked by a unique combination of check bits. If that data bit is wrong, only the check bits that include that data bit will be different. And that unique combination of check bits treated as powers of two will add up to the position of the bit that is in error.

Note that if a check bit itself is in error, the code also works and will indicate a bad check bit in positions 1, 2, 4, or 8. But since check bits are discarded after the data bits are read, we are only interested in correcting data bits, a check bit in error can be effectively ignored.

Remember at the beginning of this discussion, we concluded that 4 check bits were needed to check 8 data bits. Now it should be clear why. Focusing on the 4 check bits and their possible values, we have:

Old Check XOR New Check	Meaning
0000	All bits correct
0001	Bit 1 incorrect (check bit, ignore it)
0010	Bit 2 incorrect (check bit, ignore it)
0011	Bit 3 incorrect (data bit 1, correct it)
0100	Bit 4 incorrect (check bit, ignore it)
0101	Bit 5 incorrect (data bit 2, correct it)
0110	Bit 6 incorrect (data bit 3, correct it)
0111	Bit 7 incorrect (data bit 4, correct it)
1000	Bit 8 incorrect (check bit ignore it)
1001	Bit 9 incorrect (data bit 5, correct it)
1010	Bit 10 incorrect (data bit 6, correct it)
1011	Bit 11 incorrect (data bit 7, correct it)
1100	Bit 12 incorrect (data bit 8, correct it)

- In the left column, you can see we can generate $2^k = 2^4 = 16$ patterns with our k=4 check bits, even though in this case we only need 13 of them.
- In the right column, you can see we need at least n+k+1 = 8+4+1 = 13 different messages
  - 1 message for each possible bit error
  - 1 message for no error.
- So $2^k >= n + k + 1$, or $2^k – 1 >= n + k$
- From this we concluded that the SEC for n=8 data bits requires k=4 check bits.

SEC/DED: single error correction plus double error detection

By adding one additional parity bit to the SEC scheme, we get double error detection. Note that two errors can be detected but not corrected.

Actual RAM In Your PC

The letters ECC for error correcting code appear on RAM chips that implement a scheme such as Hamming SEC/DED.

Exercise: Create 12 bit Hamming Code Word from 8 bit Data Word

Data word in hex (2 digits): ____________________

Fill in the “Data” and “Check” rows with the positions of those bits, and fill in the “Value” row with the data word expanded into binary and placed in the positions reserved for data bits. For now, enter 0s for the check bit positions.

Value												
Data												
Check												
Bits	12	11	10	9	8	7	6	5	4	3	2	1

At this point, note the resulting intermediate code word in hex (3 digits): _________________

In the first row, indicate which data bit position is represented by each code word position. Then for each check bit, place a check mark ✓ in the column if the check bit should check that data bit.

	12 (1100) = 8	11 (1011) = 7	10 (1010) = 6	9 (1001) = 5	7 (0111) = 4	6 (0110) = 3	5 (0101) = 2	3 (0011) = 1
C_1								
C_2								
C_4								
C_8								

Calculate the Check Bits for the Code Word

- XOR of all bits, or equivalently, calculate the bit to enforce even parity

Check Bit	Value
C_1	
C_2	
C_4	
C_8	

Insert check bits back into code word along with data bits.

Value												
Data												
Check												
Bits	12	11	10	9	8	7	6	5	4	3	2	1

Note the final code word in hex: ____

Exercise: Recover 8 data bits from 12 bit Code Word with Single Bit Error

Code word with single data bit error in hex (3 digits): ____________________

Completed table from previous worksheet showing how to calculate check bits from data bits:

	12 (1100) = 8	11 (1011) = 7	10 (1010) = 6	9 (1001) = 5	7 (0111) = 4	6 (0110) = 3	5 (0101) = 2	3 (0011) = 1
C_1		✓		✓	✓		✓	✓
C_2		✓	✓		✓	✓		✓
C_4	✓				✓	✓	✓	
C_8	✓	✓	✓	✓				

- Fill in the "Values" row with the binary bits
- Copy the current check bits to "Check #1" row
- Recalculate the check bits from the data bits in "Check #2" row
- Form the XOR of each pair of bits from "Check #1" and "Check #2" row in "XOR" row

	12	11	10	9	8	7	6	5	4	3	2	1
Values												
Data	8:	7:	6:	5:		4:	3:	2:		1:		
Check #1					8:				4:		2:	1:
Check #2					8:				4:		2:	1:
XOR					8:				4:		2:	1:

Table for recalculating check bits for "Check #2" row.

Check Bit	Value
C_1	
C_2	
C_4	
C_8	

Code Word Bit in error: _____________

Data Bit in error: ____________

Corrected 8 bit data in hex (2 digits): _____________