Spiral 2-9

Tri-State Gates
Memories
DMA

Learning Outcomes

- I understand how a tri-state works and the rules for using them to share a bus
- I understand how SRAM and DRAM cells perform reads and writes
- I understand the pros and cons of SRAM and DRAM
- I understand how modern memories optimize sequential access
- I know what a DMA engine is and why it is used
- I understand the difference between big- and little-endian byte ordering and the problems that might arise when transferring between different orderings

TRI-STATE GATES

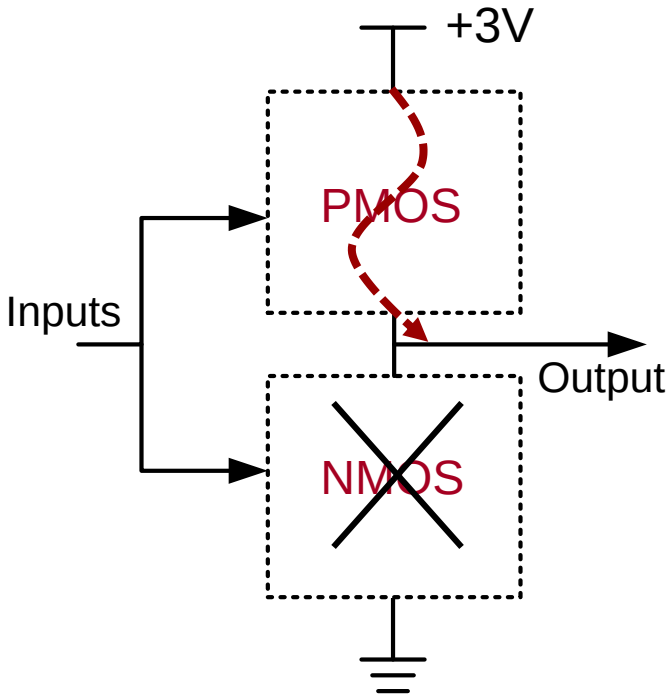
Typical Logic Gate

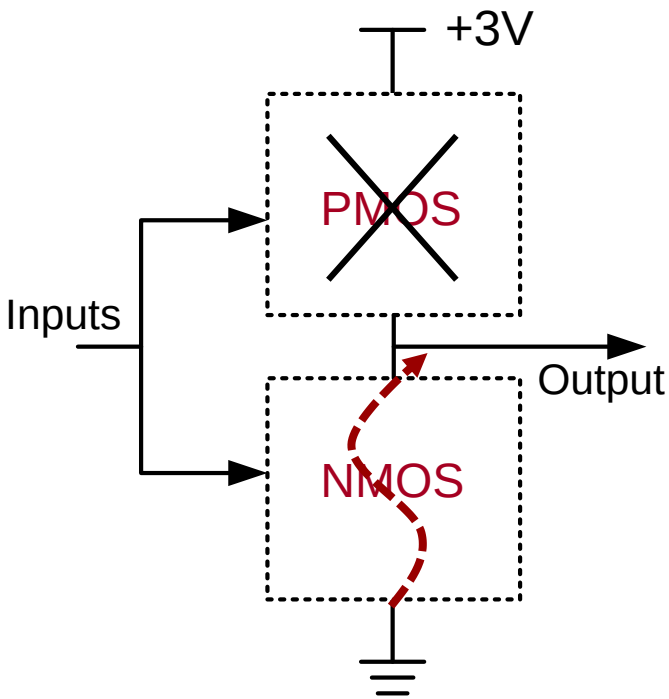
- Gates can output two values: 0 & 1
  - Logic '1' (Vdd = 3V or 5V), or Logic '0' (Vss = GND)
  - But they are ALWAYS outputting something!!!
- Analogy: a sink faucet
  - 2 possibilities: Hot ('1') or Cold ('0')
- In a real circuit, inputs cause *EITHER* a pathway from output to VDD *OR* VSS

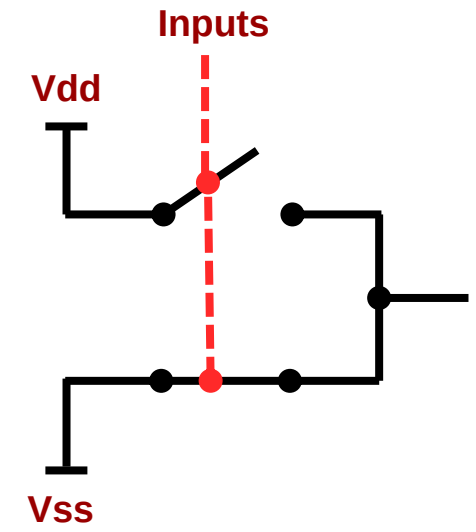
Hot Water = Logic 1

Cold Water = Logic 0

(Strapped together so always one type of water coming out)

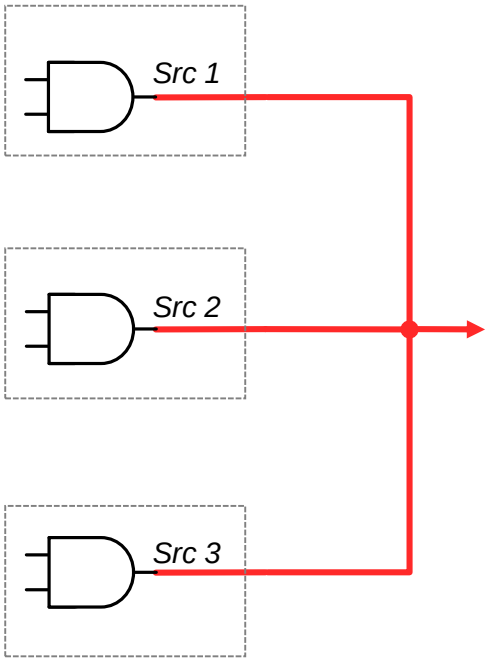


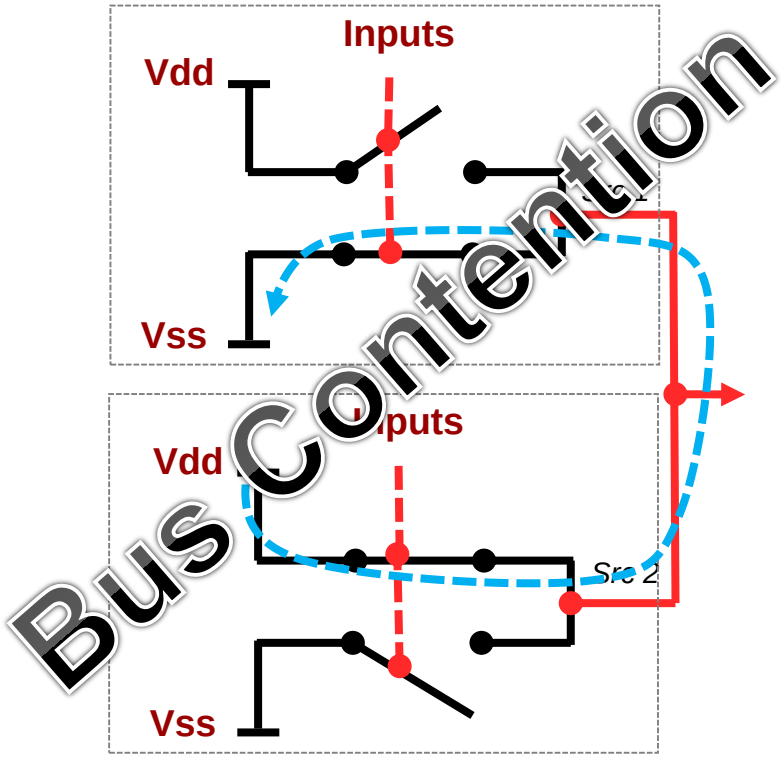




Output Connections

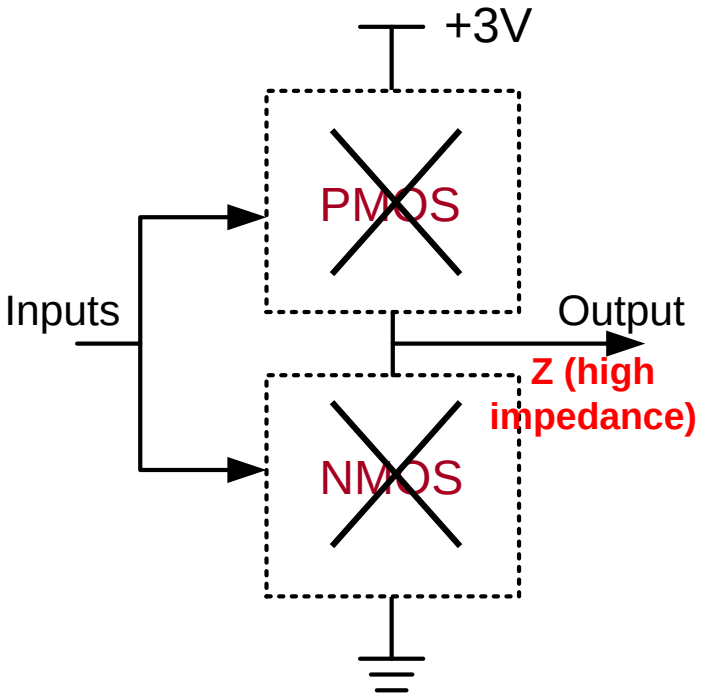
- Can we connect the output of two logic gates together?
- No! Possible short circuit (static, low-resistance pathway from Vdd to GND)
- We call this situation “bus contention”

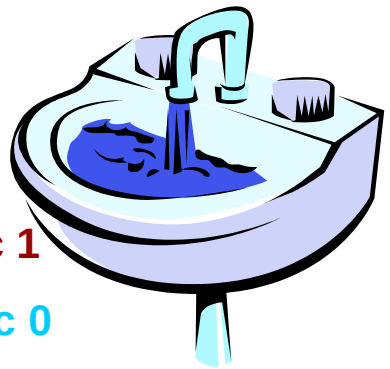




Tri-State Buffers

- Normal digital gates can output two values: 0 & 1
  1. Logic 0 = 0 volts
  2. Logic 1 = 5 volts
- Tristate buffers can output a third value:
  3. Z = High-Impedance = "Floating" (no connection to any voltage source…infinite resistance)
- Analogy: a sink faucet
  - 3 possibilities:
    1.) Hot water,
    2.) Cold water,
    3.) NO water





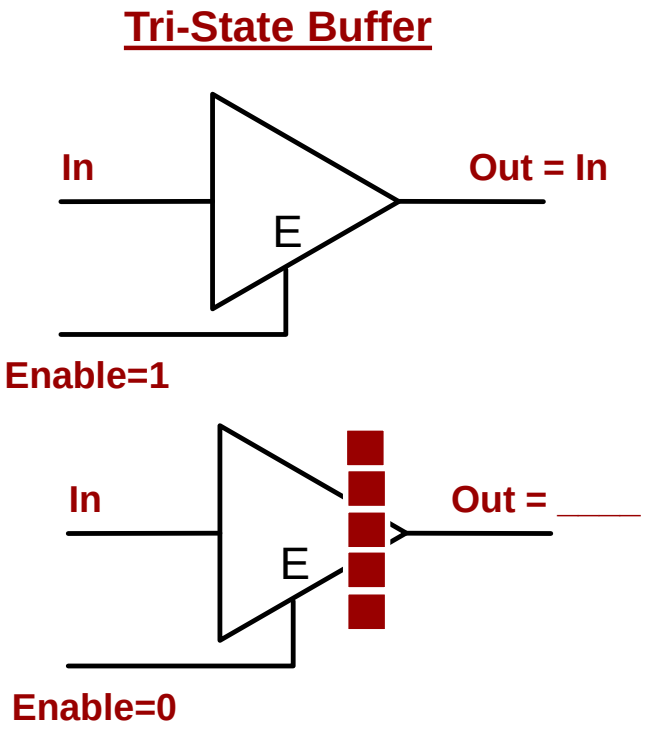
Hot Water = Logic 1

Cold Water = Logic 0

NO Water = Z (High-Impedance)

Tri-State Buffers

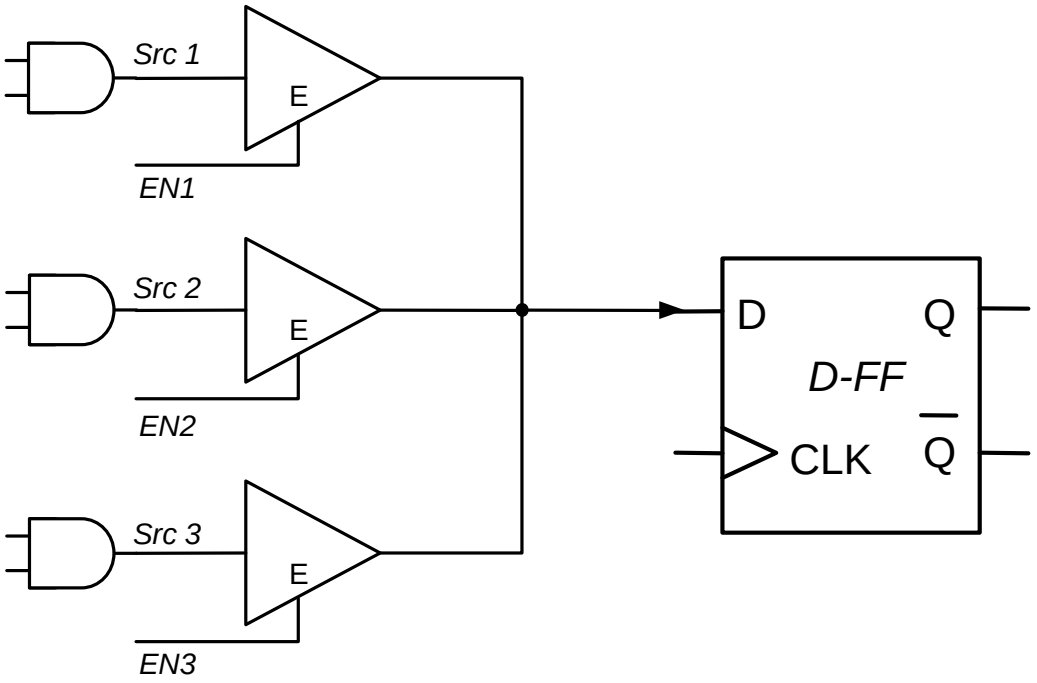
- Tri-state buffers have an extra enable input
- When disabled, output is said to be at high impedance (a.k.a. Z)
  - High Impedance is equivalent to no connection (i.e. floating output) or an infinite resistance
  - It's like a brick wall between the output and any connection to source
- When enabled, normal buffer



En	In	Out
0	-	Z
1	0	0
1	1	1

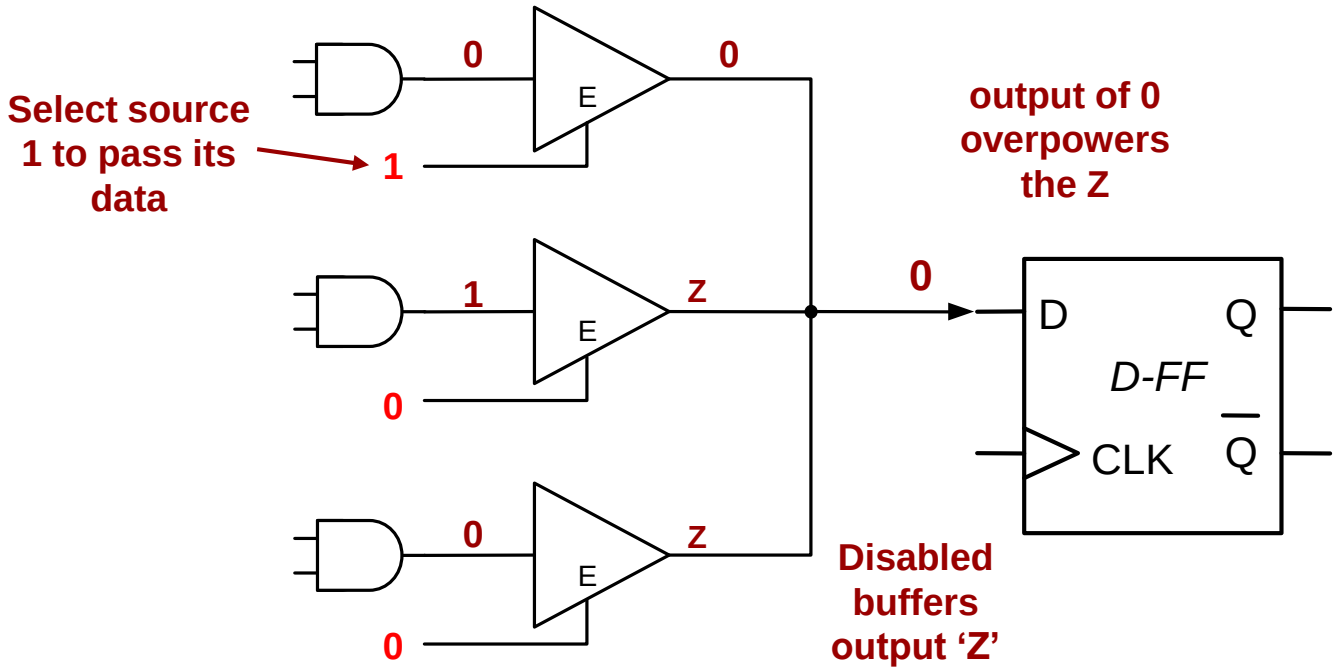
Tri-State Buffers

- We use tri-state buffers to share one output amongst several sources
- Rule: Only 1 buffer enabled at a time



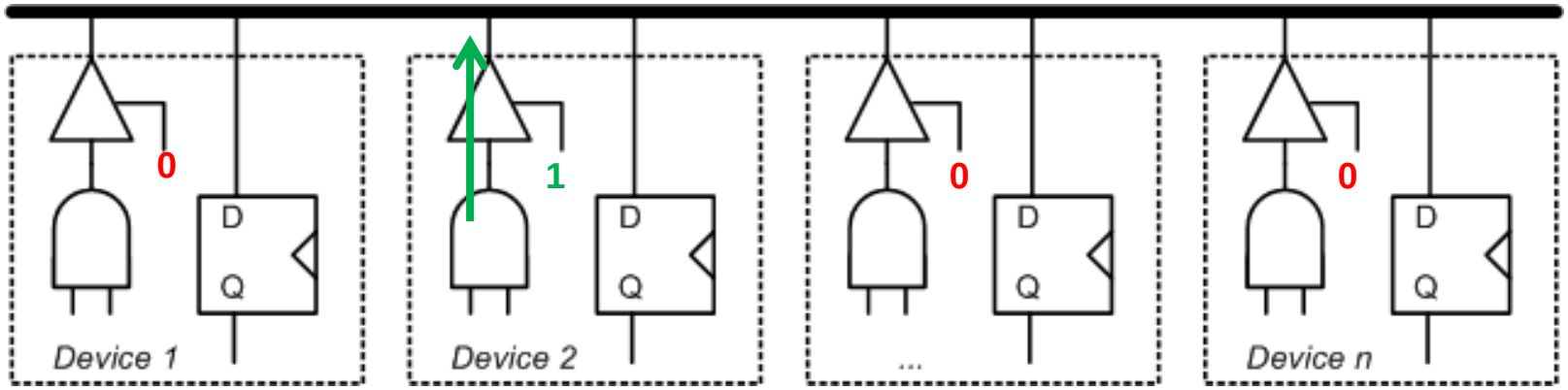
Tri-State Buffers

- We use tri-state buffers to share one output amongst several sources
- Rule: Only 1 buffer enabled at a time
- When 1 buffer enabled, its output overpowers the Z's (no connection) from the other gates



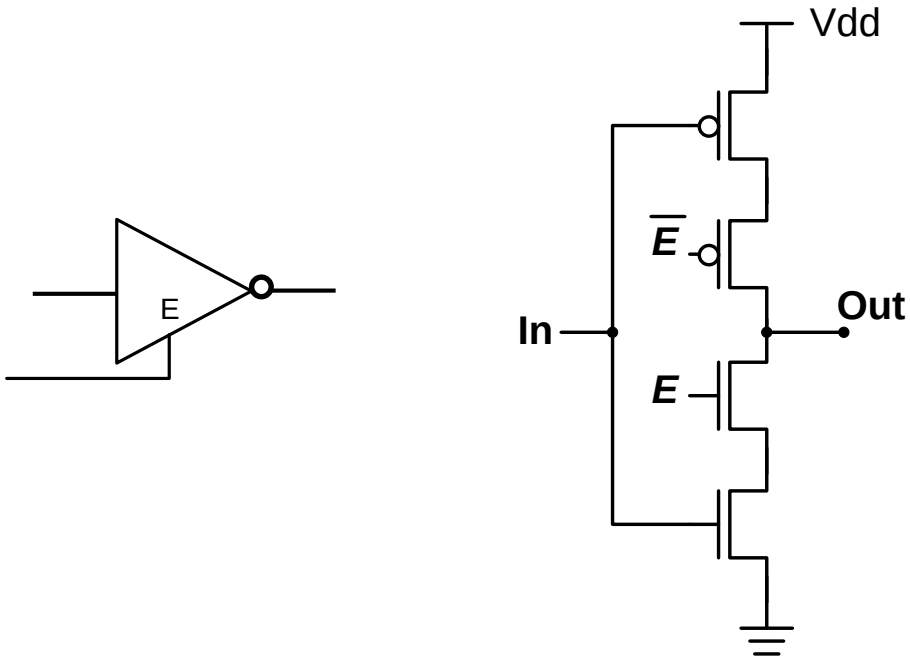
Bidirectional Bus

- 1 transmitter (otherwise bus contention)
- N receivers
- Each device can send (though 1 at a time) or receive



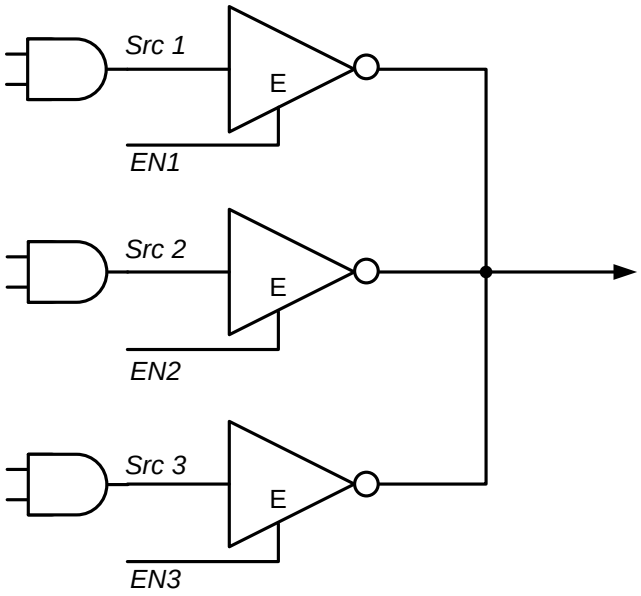
Tri-State Cell Implementation

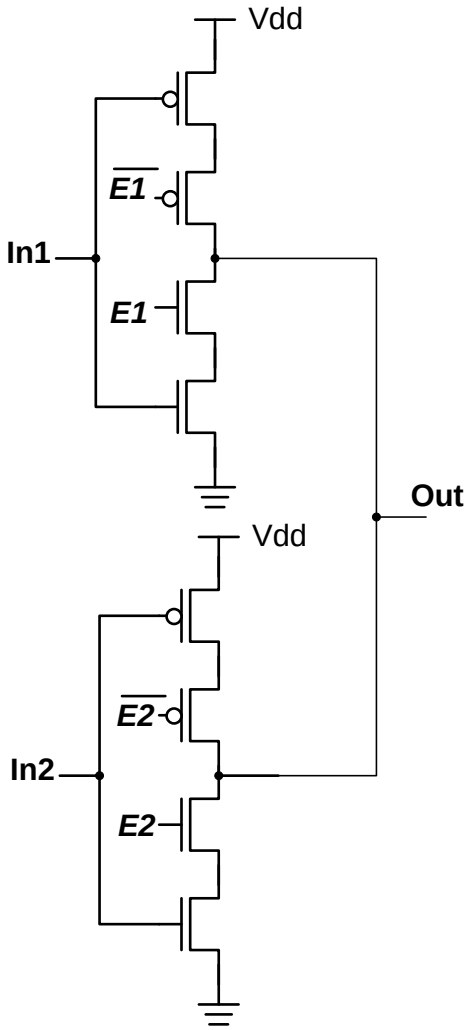
- To implement we simply add a pair of inner transistors that can BOTH be turned off
  - This would isolate the output from any Vdd or GND connection



Tri-State Cell Implementation

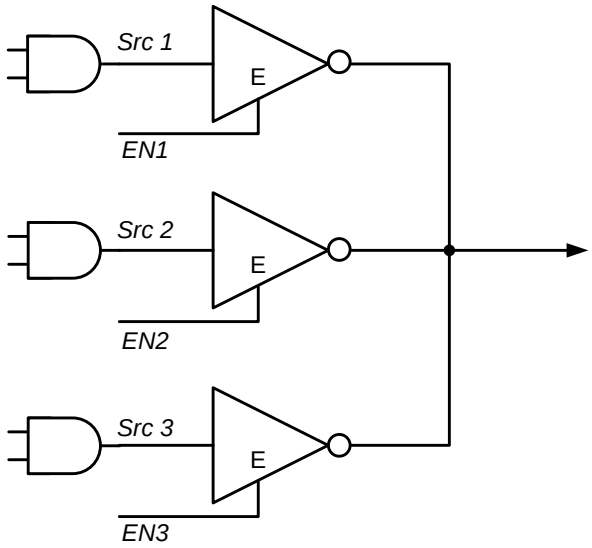
- To implement we simply add a pair of inner transistors that can BOTH be turned off
  - This would isolate the output from any Vdd or GND connection

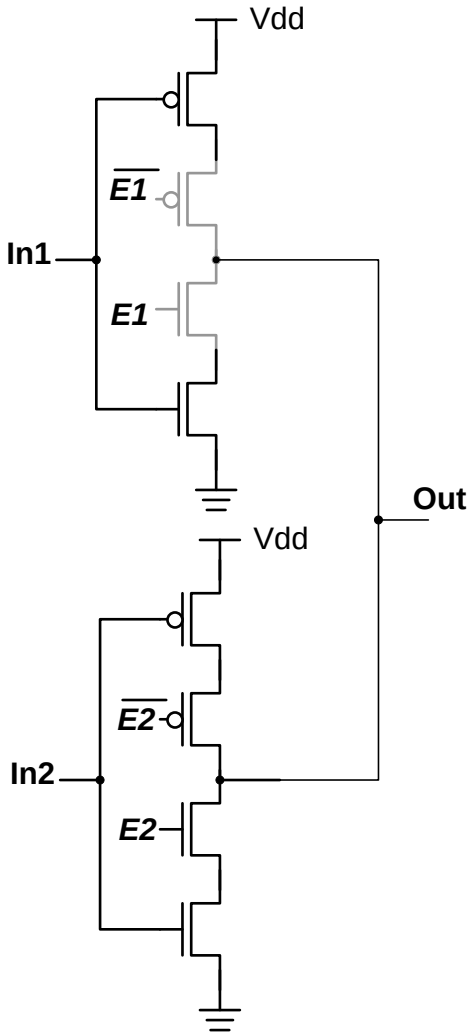




Tri-State Cell Implementation

- If E1 = 0 then the upper tri-state will isolate the output from any Vdd/GND connection
- If E2 = 1 then the lower tri-state acts as a normal inverter
- If both E1=0, E2=0 then the output is floating (Z = high impedance)

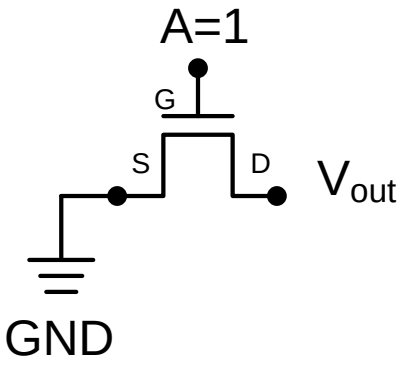


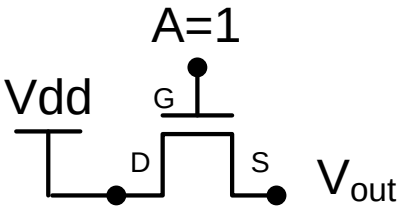


TRANSMISSION GATE & PASS TRANSISTORS

Why NMOS Pass Weak 1s

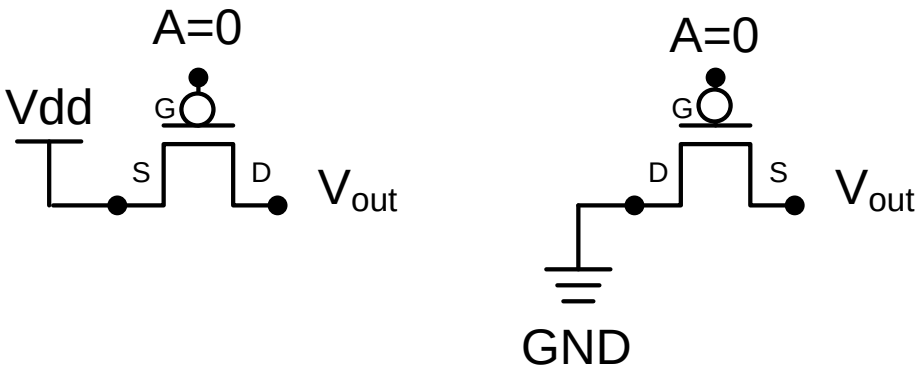
- We've said NMOS are:
  - Pass a strong 0 but a weak 1
- Let's explore why
  - In the first case one side is tied to GND and thus is the source (lower voltage side is the source)
    - Vgs = Vdd and transistor will stay on no matter what the drain voltage is (eventually Vd will equal 0)
  - In the second case one side is tied to Vdd and thus the other terminal is the source (source is the lower voltage terminal)
    - As the source node charges up eventually it will reach Vdd-Vt but at that point Vgs = Vg – Vs = Vdd – (Vdd-Vt) = Vt
    - Recall to stay on the transistor must have Vgs > Vt.
    - So the transistor will turn off and not allow the source to charge all the way to Vdd





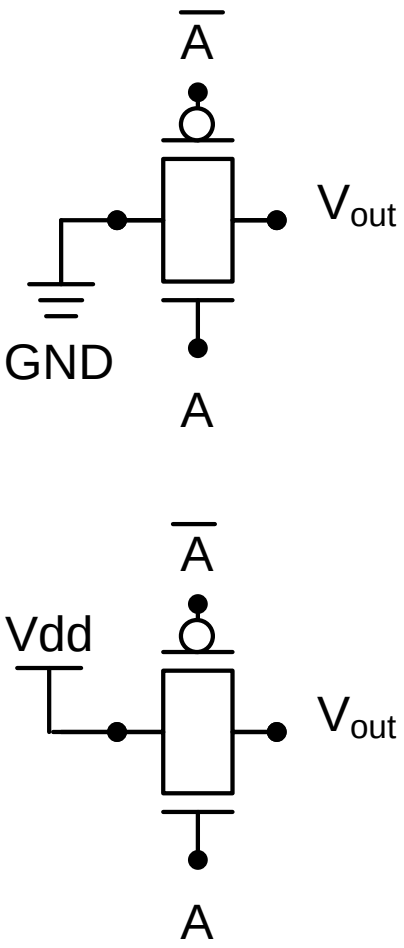
Why PMOS Pass Weak 0s

- We've said PMOS are:
  - Pass a strong 1 but a weak 0
- For similar reasons as on the previous slide when we try to pass a 0 through a PMOS transistor the transistor will turn off when Vout reaches Vt
  - This prevents the PMOS from passing a strong 0



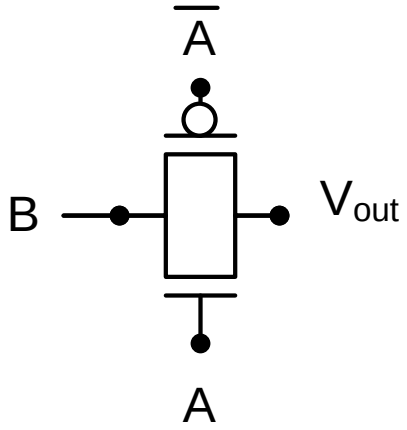
Transmission Gates

- Well if one is good at passing 0's and one is good at passing 1's why not use them together
- A Transmission gate puts a PMOS and NMOS transistors in parallel
  - Depending on the input voltage at the source either the NMOS or PMOS can pull the drain up/down to that voltage
  - Use complementary inputs at each gate to ensure only one transistor is on



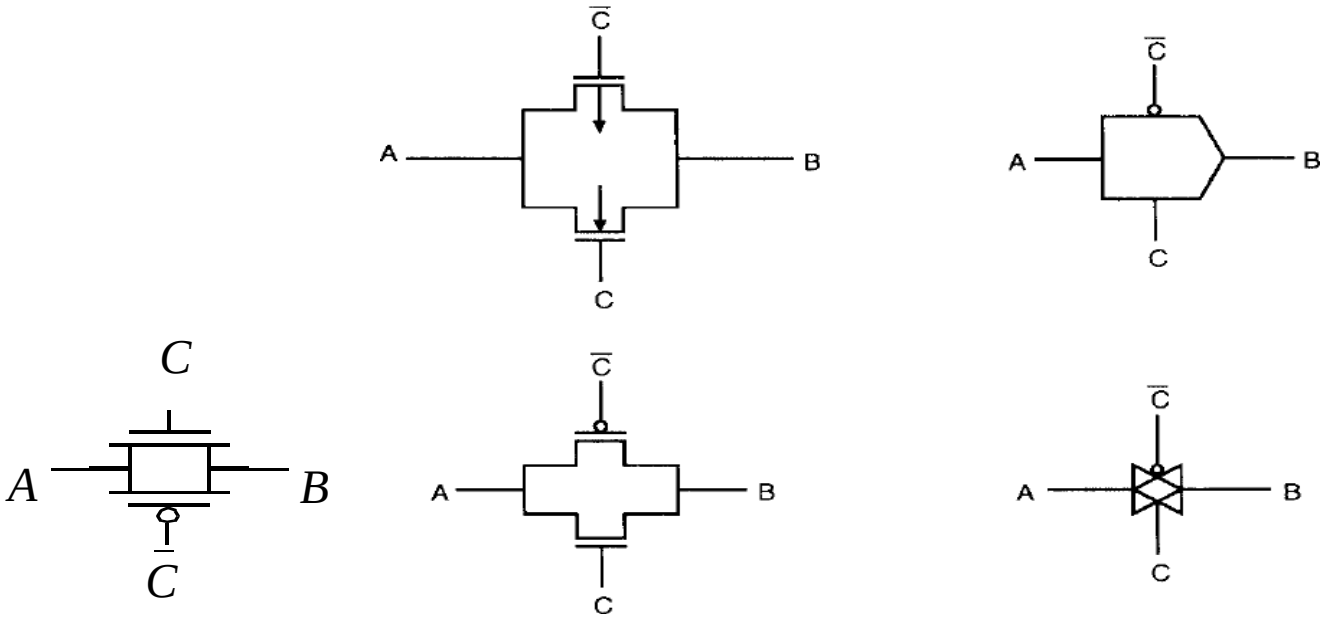
Transmission Gates

- We can even connect the source to some other signal rather than a constant voltage
- In the example to the right, if A = 1 (A' = 0) then B is passed
- This gate produces
  - AB when A=1 (i.e. pass B when A=1)
  - Z (floating) when A=0



Transmission Gate Symbols

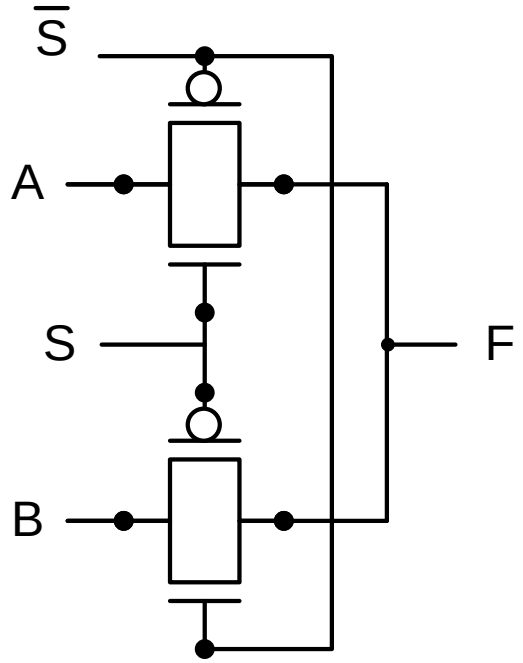
- The CMOS TG operates as a bidirectional switch between nodes A and B, which is controlled by signal C



Different representations (symbols) of the CMOS TGs

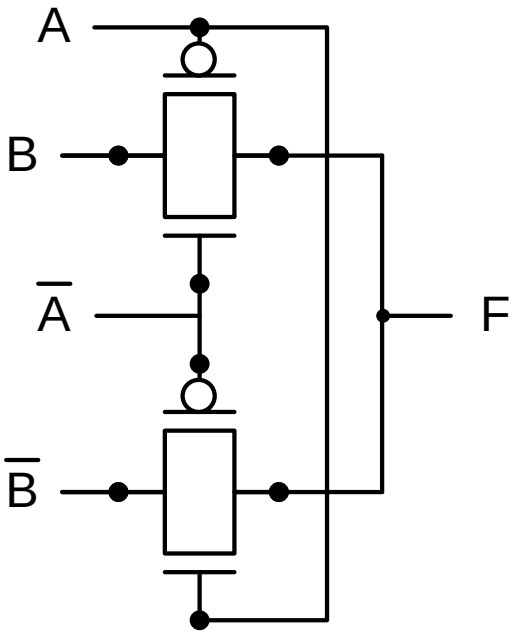
More Transmission Gates

- By making complementary cases of transmission gates we can build some useful structures
- What is this structure?
- A 2-to-1 mux



More Transmission Gates

- What is this structure?
- An XOR gate

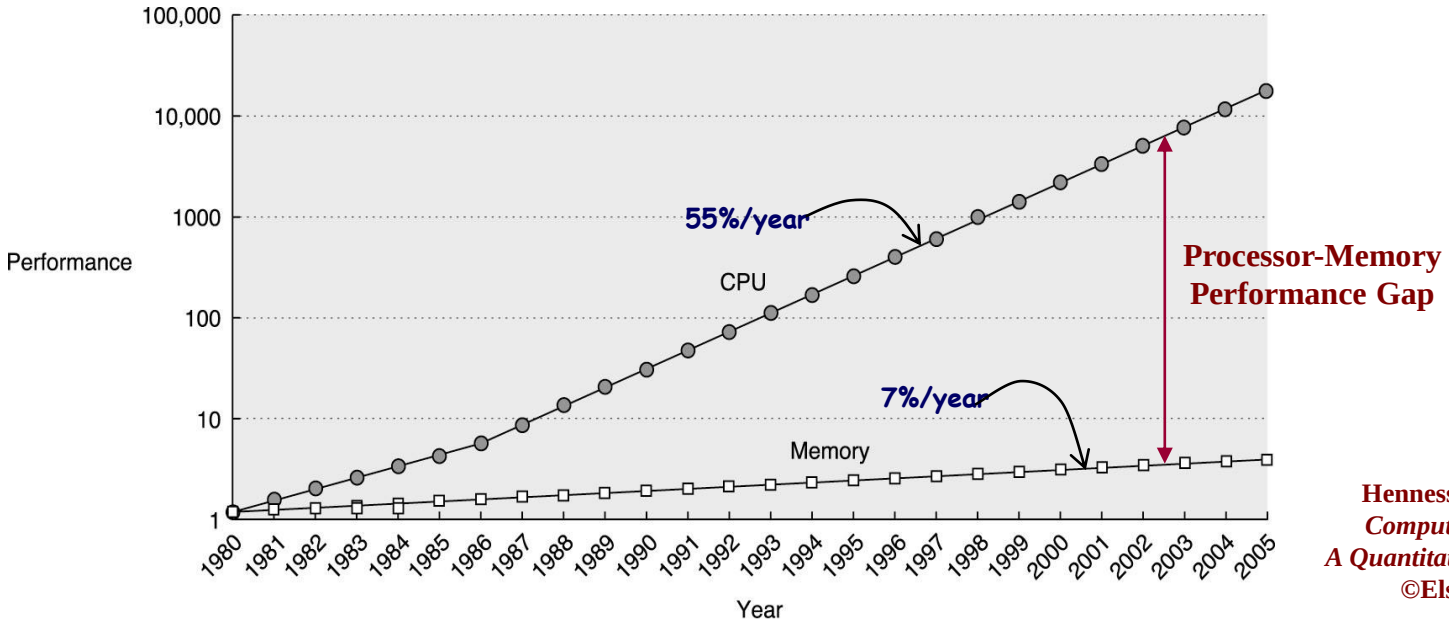


MEMORY

MEMORY TECHNOLOGIES

The Memory Wall

- Problem: The Memory Wall
  - Processor speeds have been increasing much faster than memory access speeds (Memory technology targets density rather than speed)
  - Large memories yield large address decode and access times
  - Main memory is physically located on separate chips and inter-chip signals have much higher propagation delay than intra-chip signals



Hennessy and Patterson, *Computer Architecture – A Quantitative Approach* (2003)

Improving Memory Performance

- Possibilities for improvement
  - Technology
    - Can we improve our transistor-level design to create faster RAMs
    - Can we integrate memories on the same chip as our processing logic
  - Architectural
    - Can we organize memory in a more efficient manner (this is our focus)

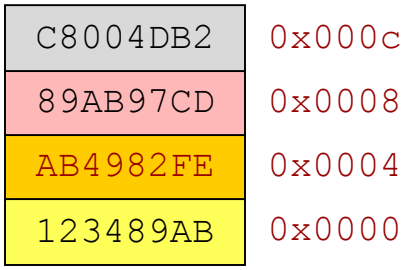
DRAM & SRAM

Memory Technology

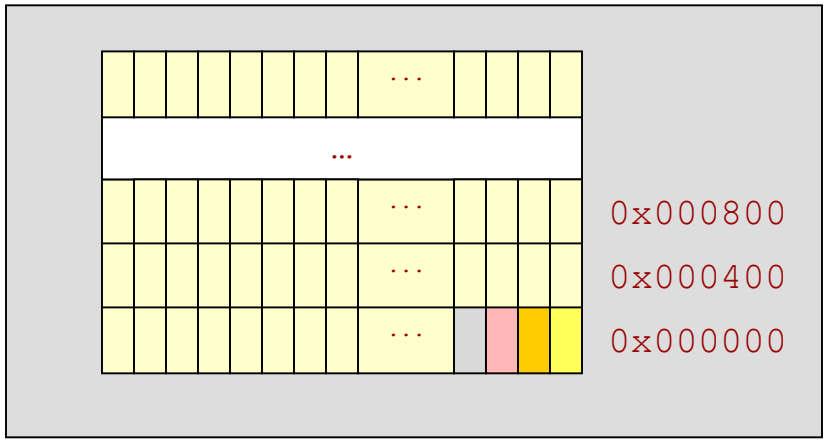
- Static RAM (SRAM)
  - Will retain values indefinitely (as long as power is on)
  - Stored bit is actively “remembered” (driven and regenerated by circuitry)
- Dynamic RAM (DRAM)
  - Will lose values if not refreshed periodically
  - Stored bit is passively “remembered” and needs to be regenerated by external circuitry

Memory Array

- Logical View = 1D array of rows (words)
  - Already this is 2D because each word is 32-bits (i.e. 32 columns)
- Physical View = 2D array of rows and columns
  - Each row may contain 1000's of columns (bits) though we have to access at least 8- (and often 16-, 32-, or 64-) bits at a time



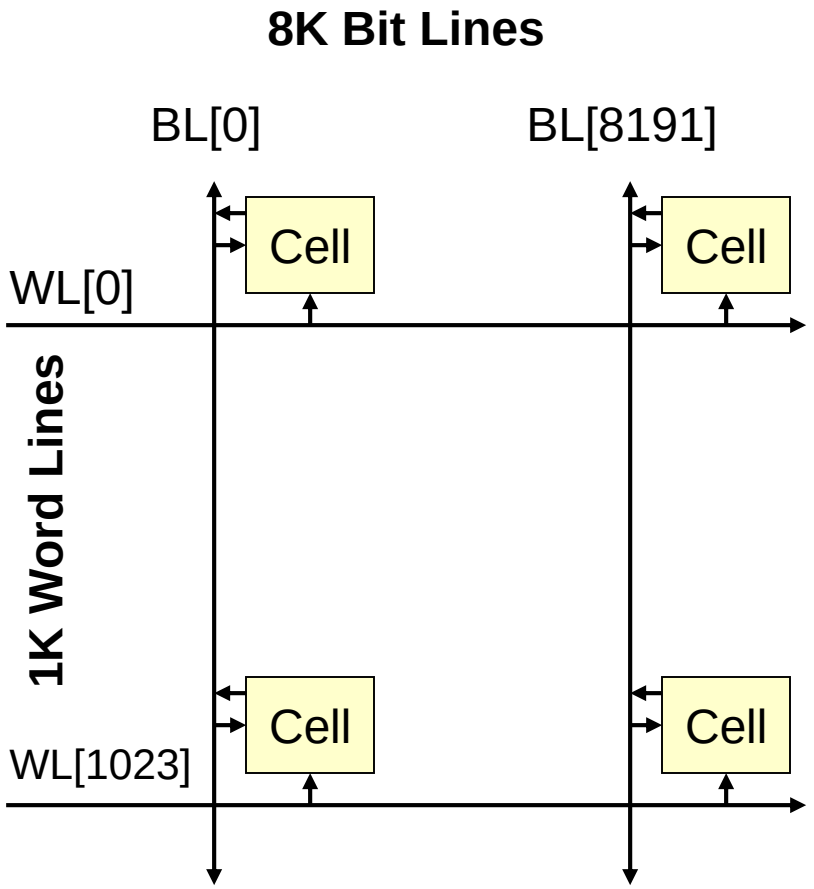
1D Logical View (Each row is a single word = 32-bits)



2D Physical View (e.g. a row is 1KB = 8Kb)

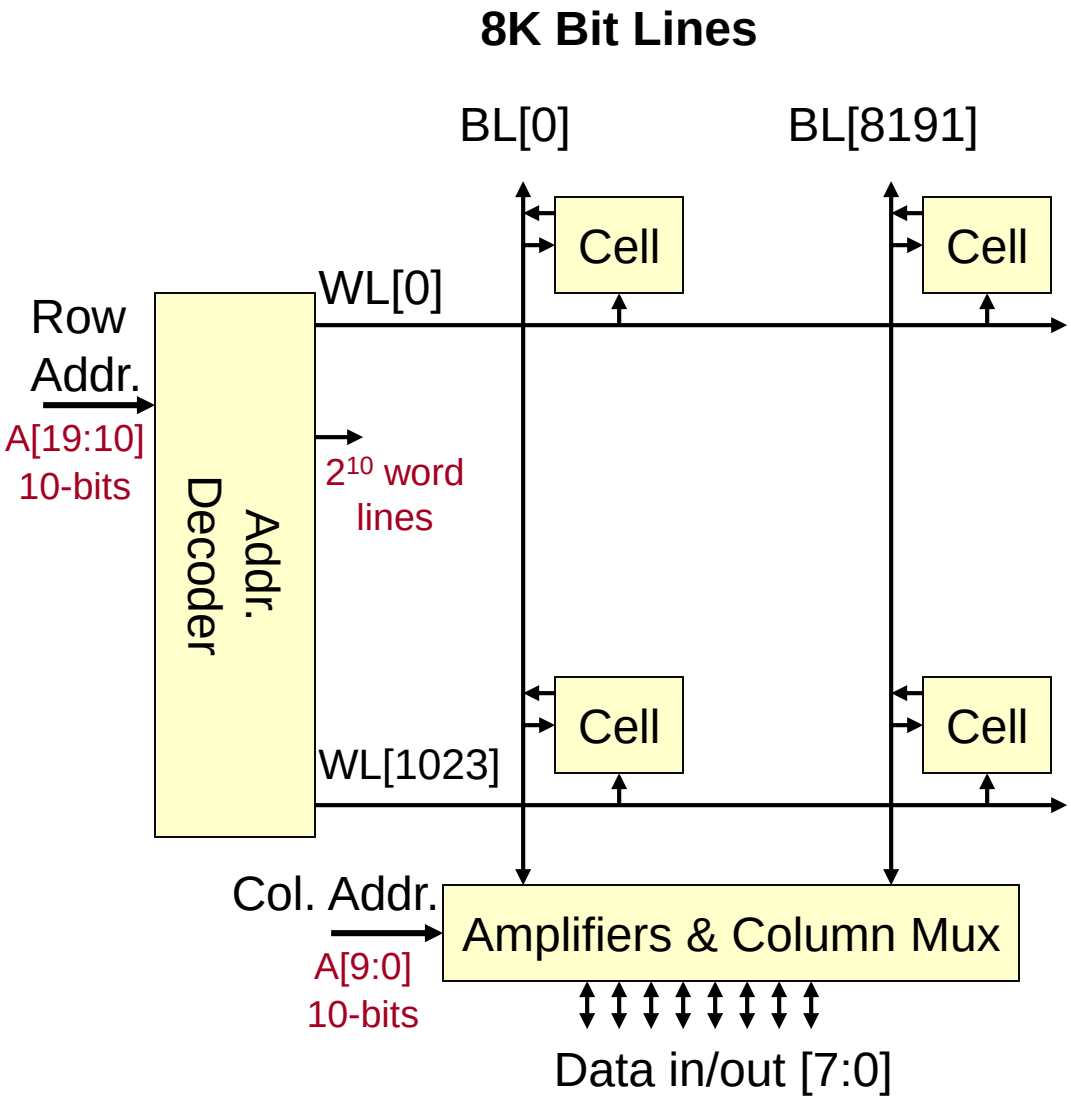
1Mx8 Memory Array Layout

- Start with array of cells that can each store 1-bit
- 1 MB = 8 Mbit = 2^{23} total cells
- This can be broken into a 2D array of cells (2^{10} x 2^{13})
- Each row connects to a WL = word line which selects that row
- Each column connects to a BL = bit line for read/write data



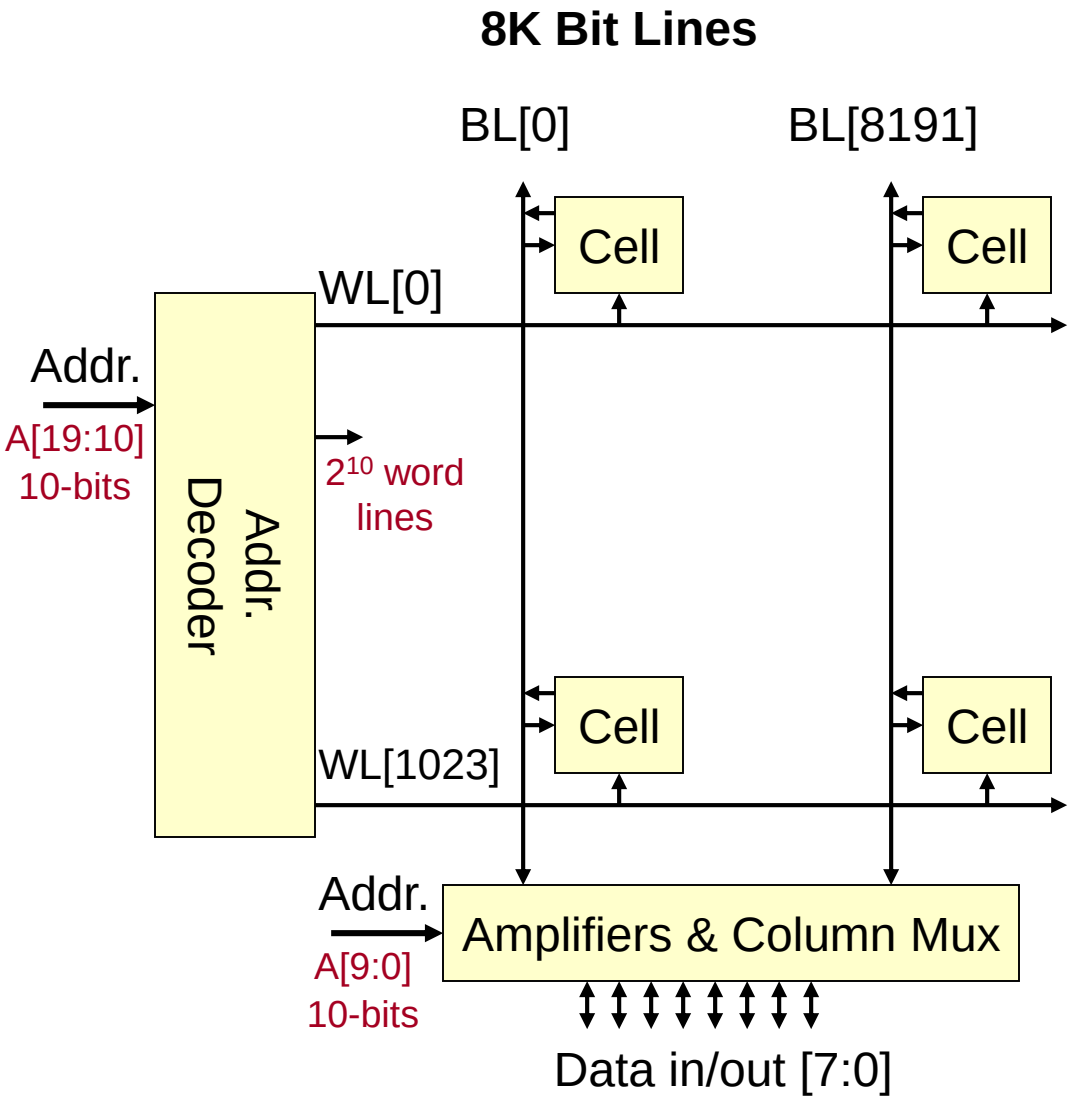
Row and Column Address

- For 1MB we need 20 address bits
- 10 upper address bits can select one of the 1024 rows
- Suppose we always want to read/write an 8-bits (byte), then we will group each set of 8-bits into 1024 byte columns (1024*8=8K)
- 10 lower address bits will select the column



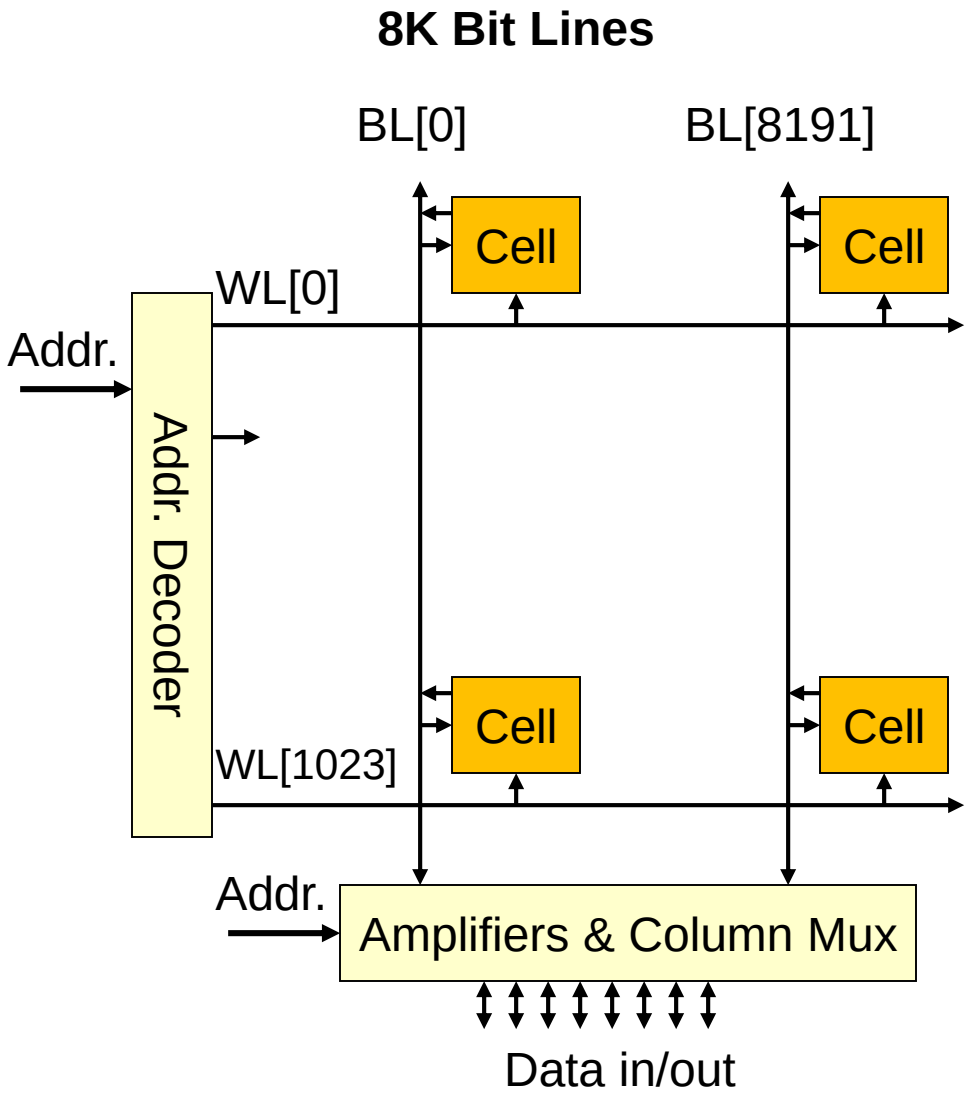
Periphery Logic

- Address decoders selects one row based on the input address number
- Bit lines are bidirectional lines for reading (output) or writing (input)
- Column multiplexers use the address bits to select the right set of 8-bits from the 8K bit lines



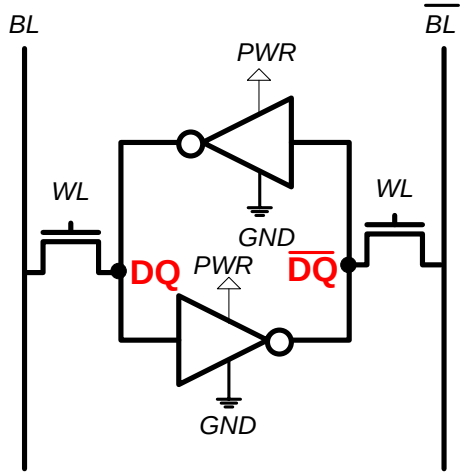
Memory Technologies

- Memory technologies share the same layout but differ in their cell implementation
- Static RAM (SRAM)
  - Will retain values indefinitely (as long as power is on)
  - When read, the stored bit is actively driven onto the bit lines
- Dynamic RAM (DRAM)
  - Will lose values if not refreshed periodically
  - When read, the stored bit passively pulls the bit line voltage up or down slightly and needs to be amplified

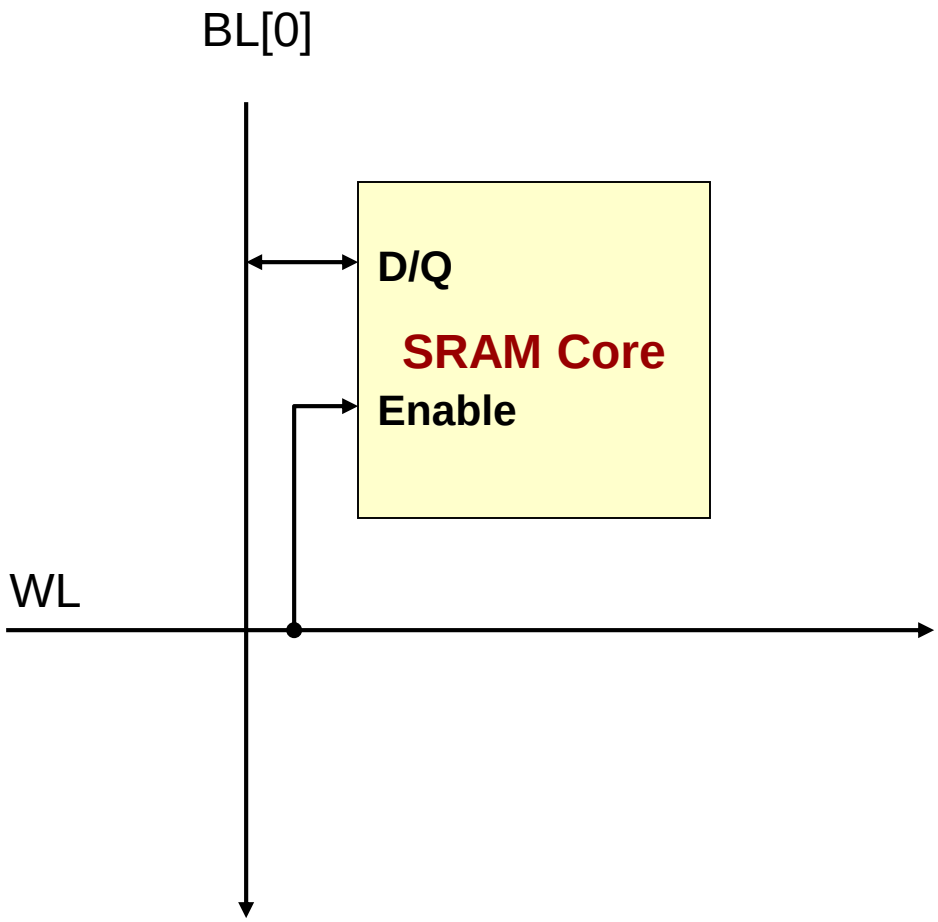


SRAM Cell

- Each memory cell requires **6 transistors**
- Each cell consists of a D-Latch is made from cross connected inverters which have active connections to PWR and GND.
  - Thus, the signal is *remembered* and *regenerated* as long as power is supplied

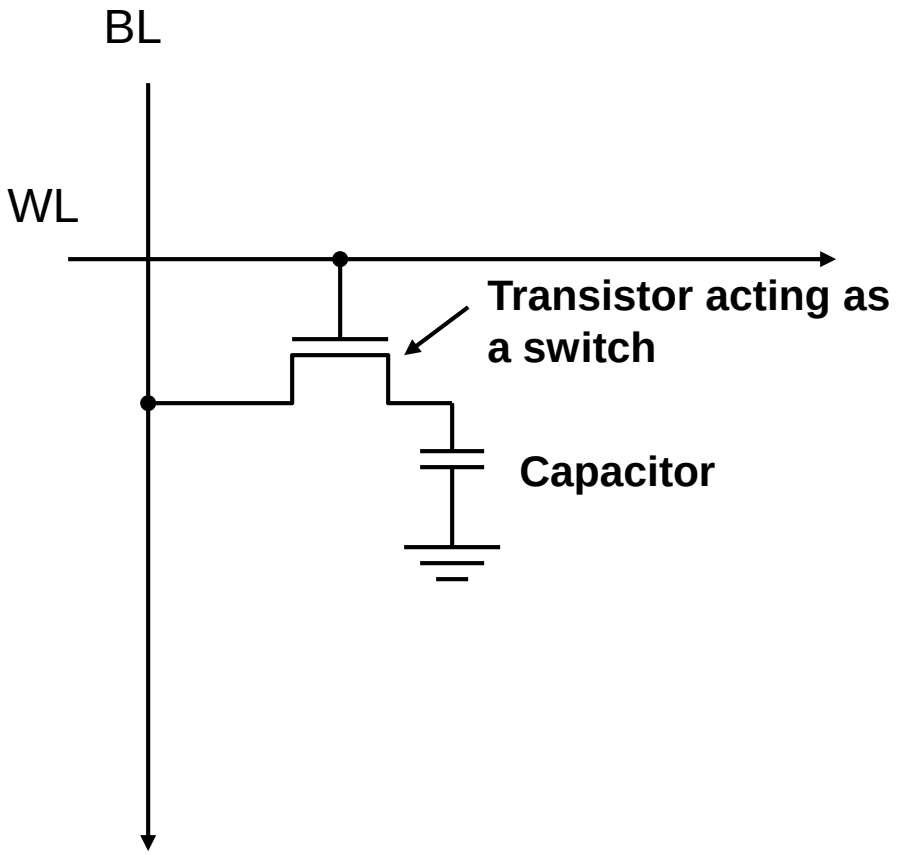


SRAM core implementation



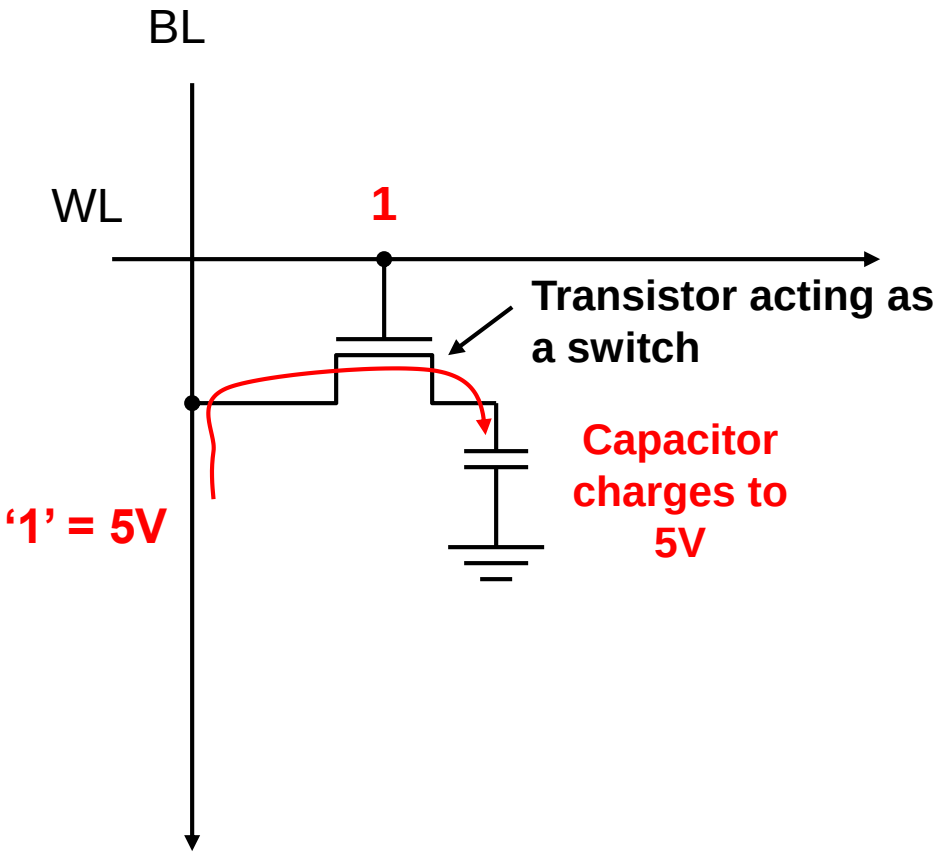
DRAM Cell

- Bit is stored on a capacitor and requires only **1 transistor and a capacitor**



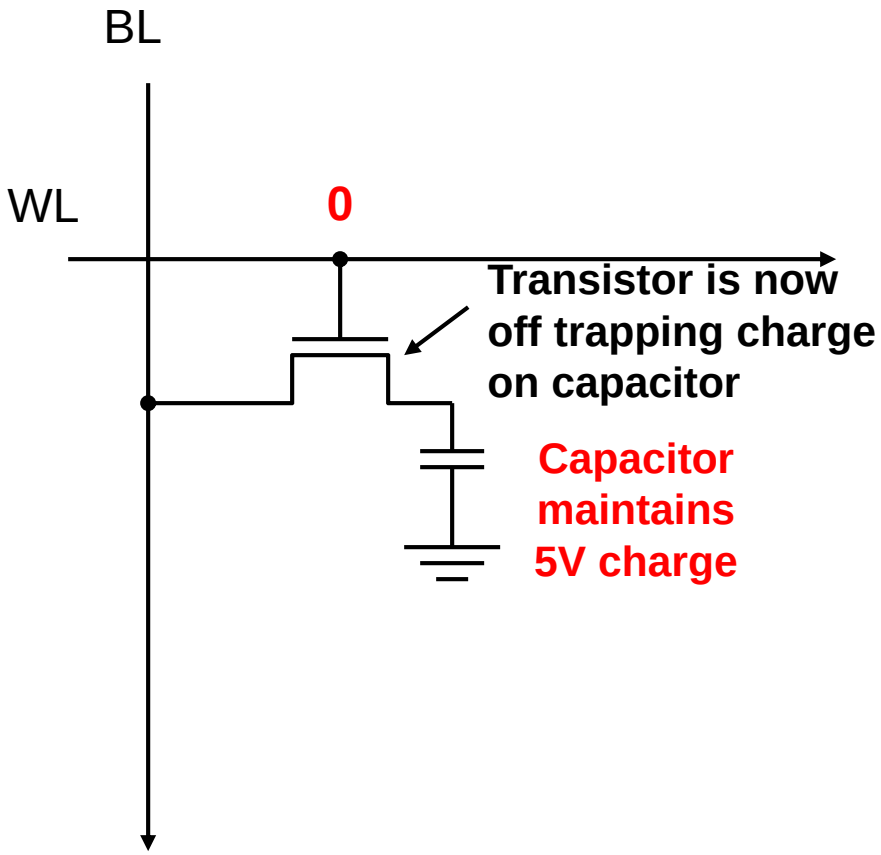
DRAM Cell

- Bit is stored on a capacitor and requires only **1 transistor**
- Write
  - WL=1 connects the capacitor to the BL which is driven to 1 or 0 and charges/discharges capacitor based on the BL value



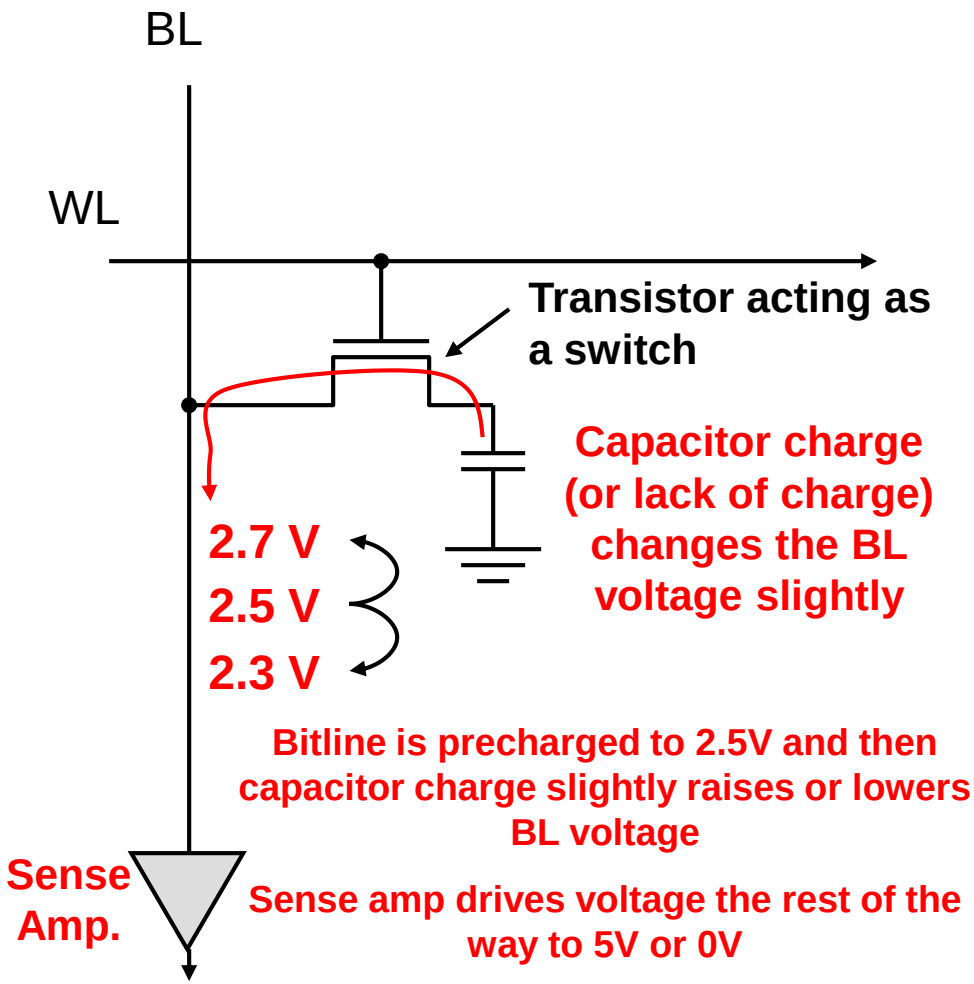
DRAM Cell

- Bit is stored on a capacitor and requires only **1 transistor**
- Write
  - WL=1 connects the capacitor to the BL which is driven to 1 or 0 and charges/discharges capacitor based on the BL value
- With WL=0 transistor is closed and value stored on the capacitor



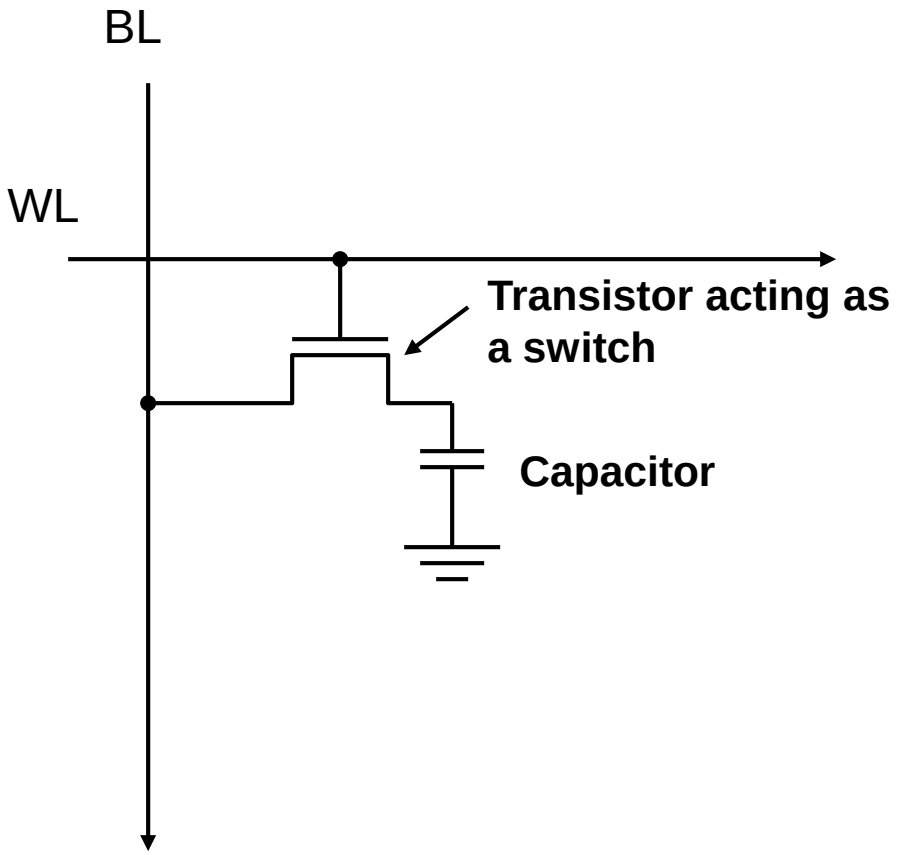
DRAM Cell

- Bit is stored on a capacitor and requires only **1 transistor**
- Write
  - WL=1 connects the capacitor to the BL which is driven to 1 or 0 and charges/discharges capacitor based on the BL value
- Read
  - BL is precharged to 2.5 V
  - WL=1 connects the capacitor to the BL allowing charge on capacitor to change the voltage on the BL



DRAM Issues

- Destructive Read
  - Charge is lost when capacitor value is read
  - Need to write whatever is read back to the cap.
- Leakage Current
  - Charge slowly leaks off the cap.
  - Value must be refreshed periodically



SRAM vs. DRAM Summary

- SRAM
  - Faster because bit lines are actively driven by the D-Latch
  - Faster, simpler interface due to lack of refresh
  - Larger Area for each cell which means less memory per chip
  - Used for cache memory where speed/latency is key
- DRAM
  - Slower because passive value (charge on cap.) drives BL
  - Slower due to refresh cycles
  - Small area means much greater density of cells and thus large memories
  - Used for main memory where density is key

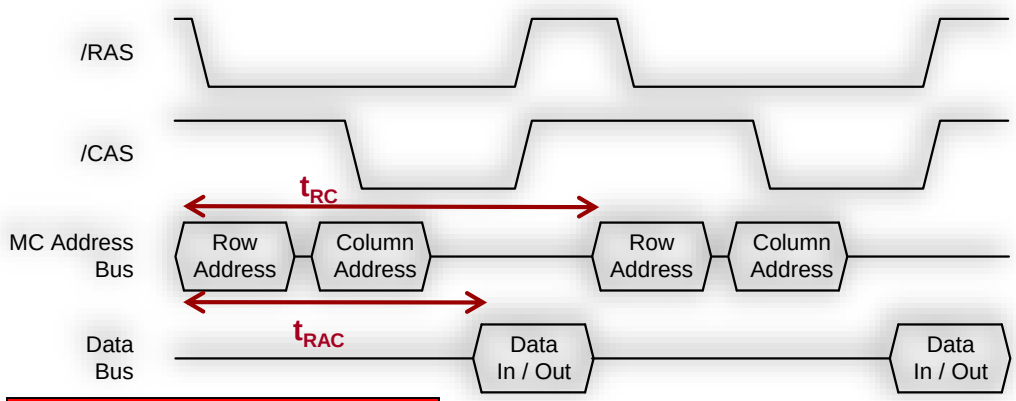
MEMORY ARCHITECTURES

Implications of Memory Technology

- Memory latency of a single access using current DRAM technology will be slow
- We must improve bandwidth
  - Idea 1: Access more than just a single word at a time
  - Technology: EDO, SDRAM, etc.
  - Idea 2: Increase number of accesses serviced in parallel (in-flight accesses)
  - Technology: Banking

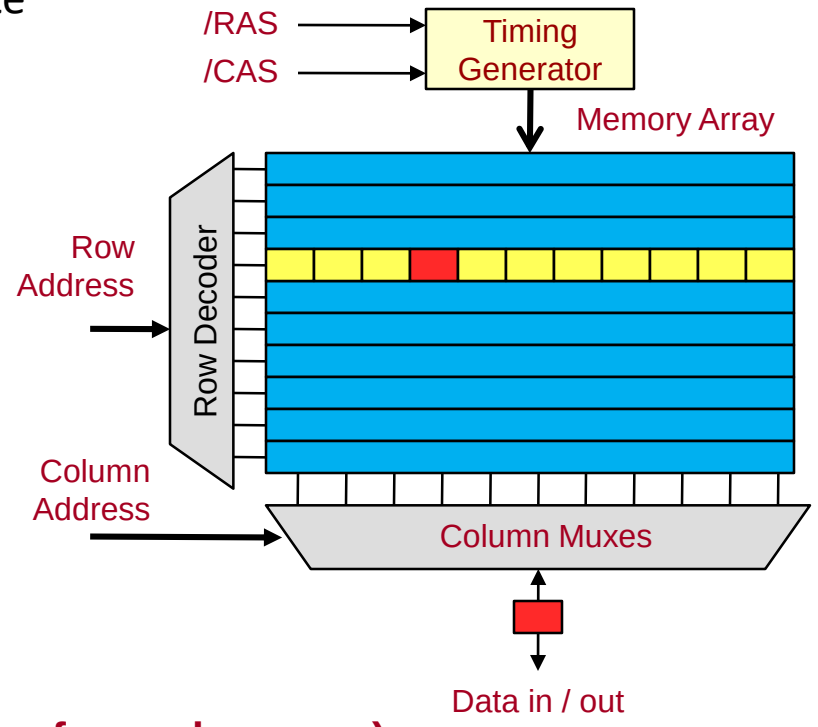
Legacy DRAM Timing

- Can have only a single access "in-flight" at once
- Memory controller must send row and column address portions for each access
  - RAS active holds the row open for reading/writing
  - CAS active selects the column to read/write



t_{RC}= Cycle Time (110ns) = Time before next access can start

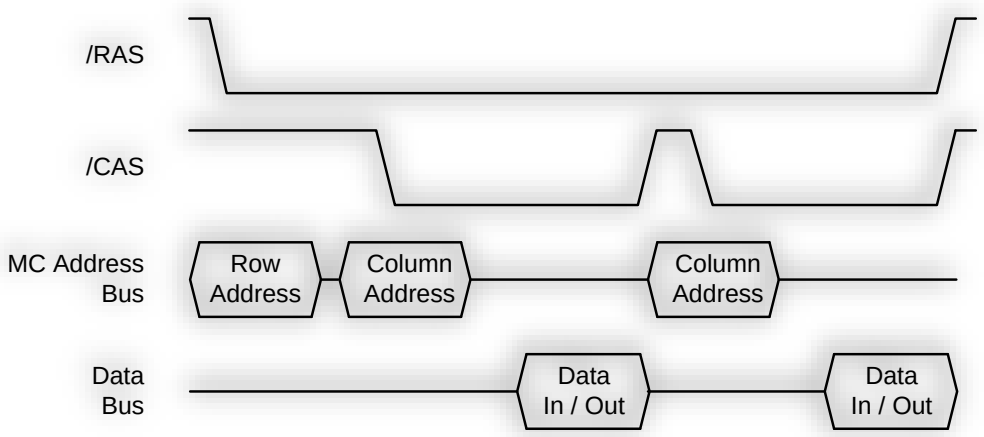
t_{RAC}=Access Time (60ns) = Time until data is valid

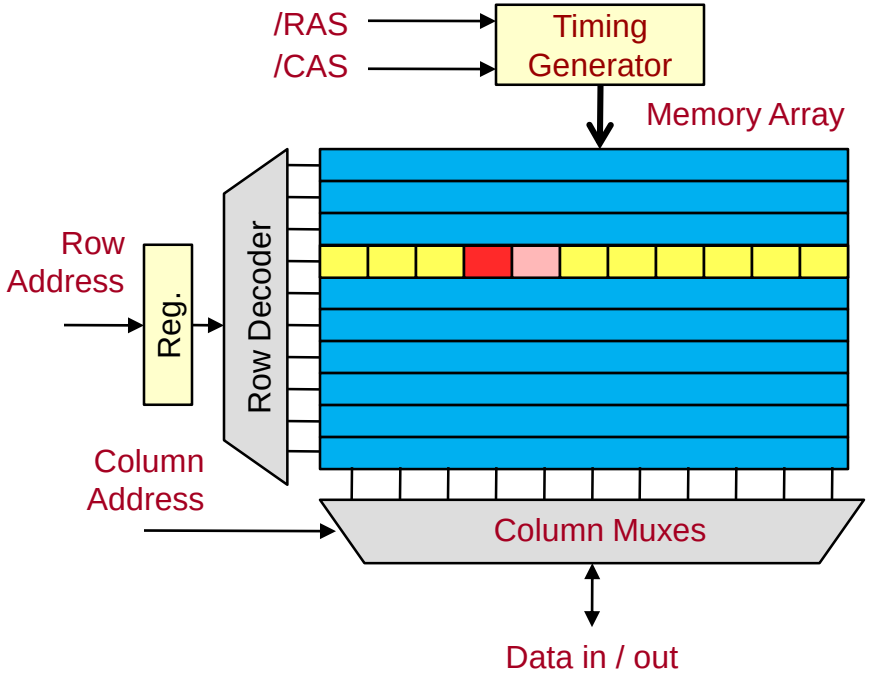


Legacy DRAM
(Must present new Row/Column address for each access)

Fast Page Mode DRAM Timing

- Can provide multiple column addresses with only one row address

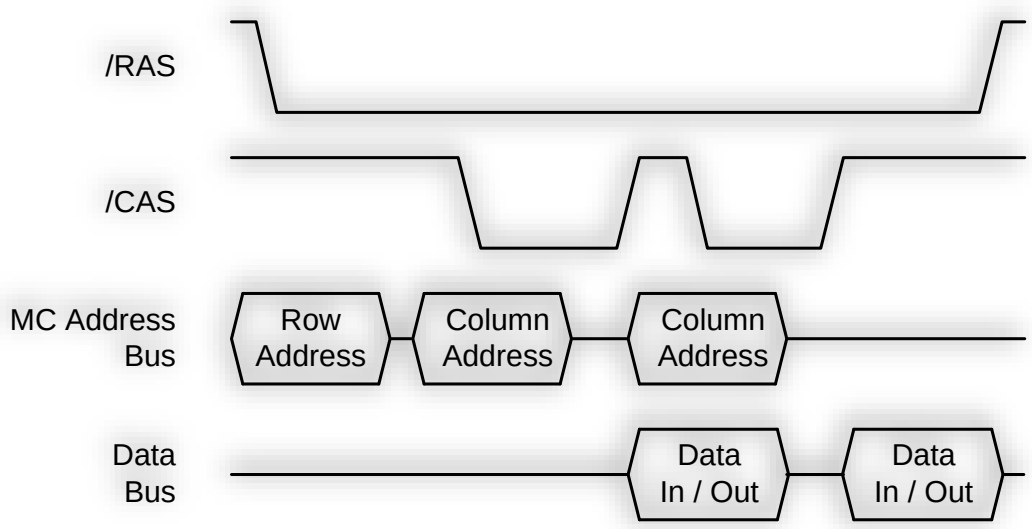


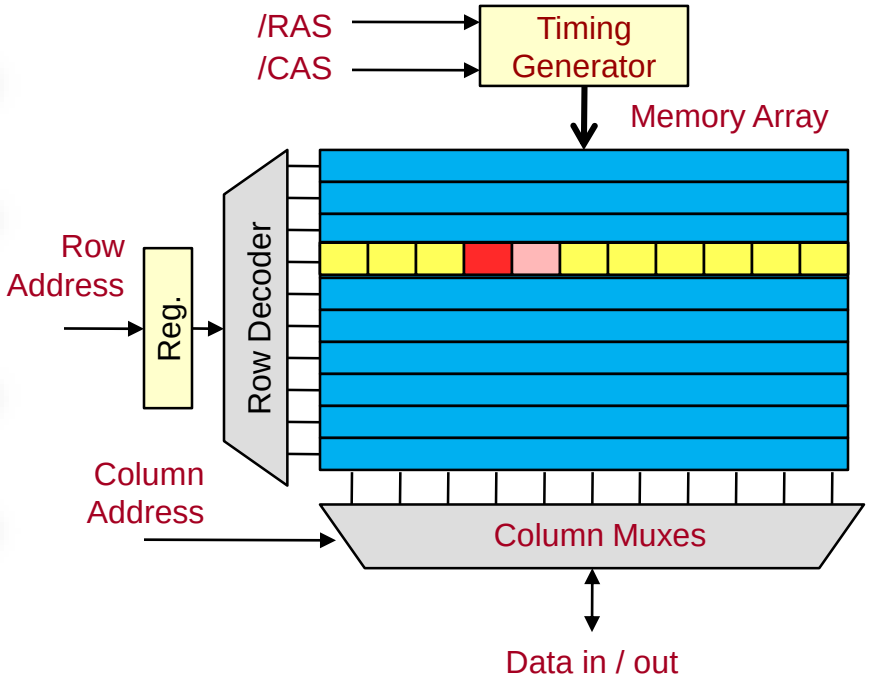


Fast Page Mode
(Future address that fall in same row can pull data from the latched row)

EDO DRAM Timing

- Similar to FPM but overlaps data i with column address i+1

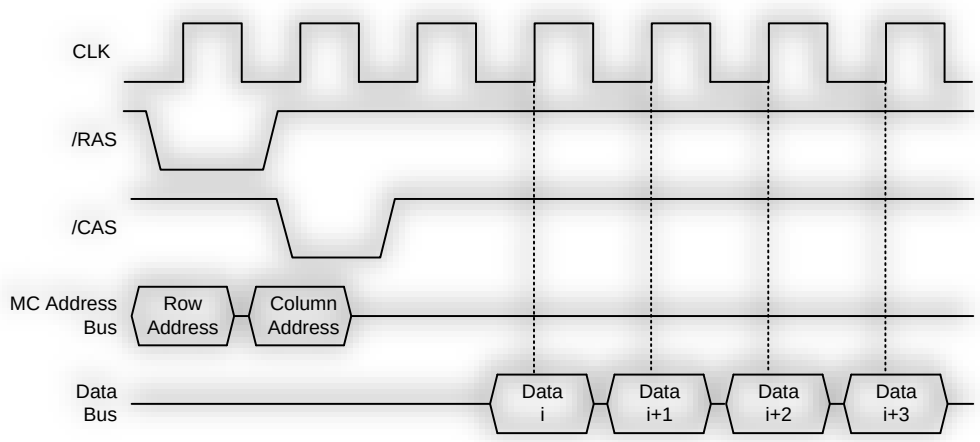


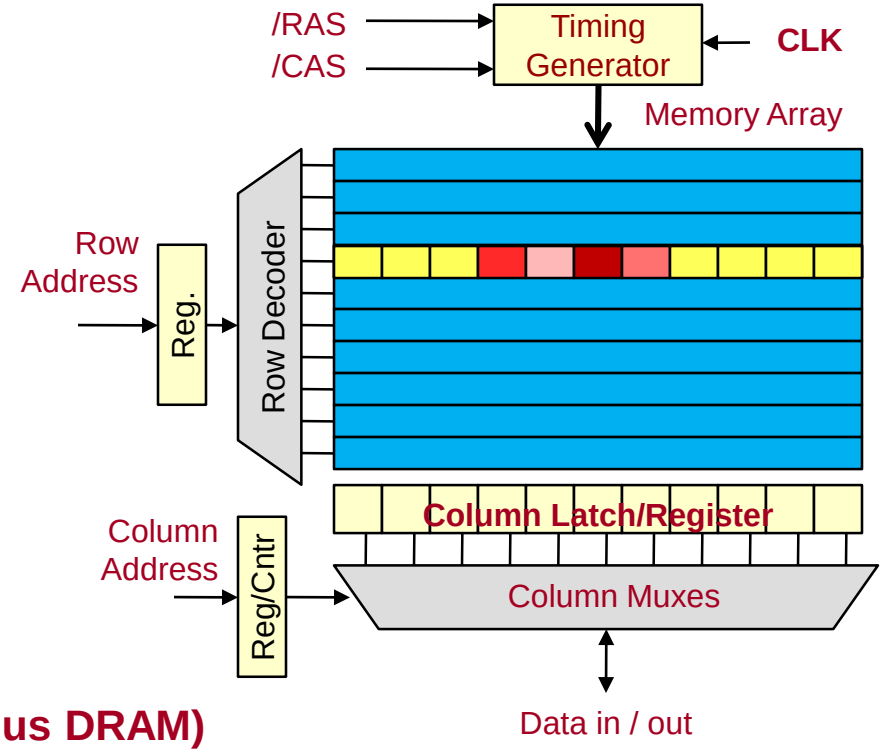


EDO (Extended Data Out)
Column address i+1 is sent while data i is being transferred on the bus

Synchronous DRAM Timing

- Registers the column address and automatically increments it, accessing *n* sequential data words in *n* successive clocks called bursts… n=4 or 8 usually)

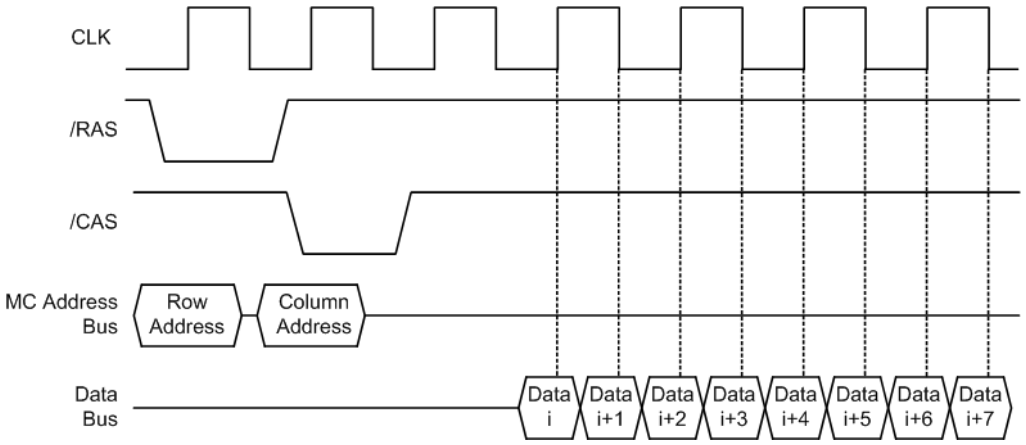


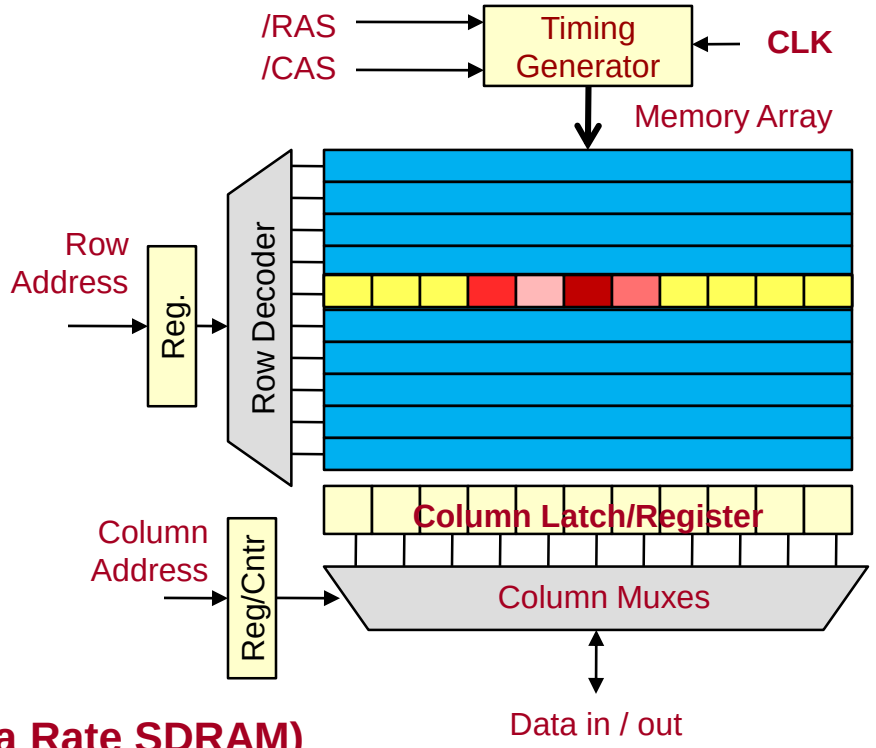


SDRAM (Synchronous DRAM)
Addition of clock signal. Will get up to 'n' consecutive words in the next 'n' clocks after column address is sent

DDR SDRAM Timing

- Double data rate access data every half clock cycle

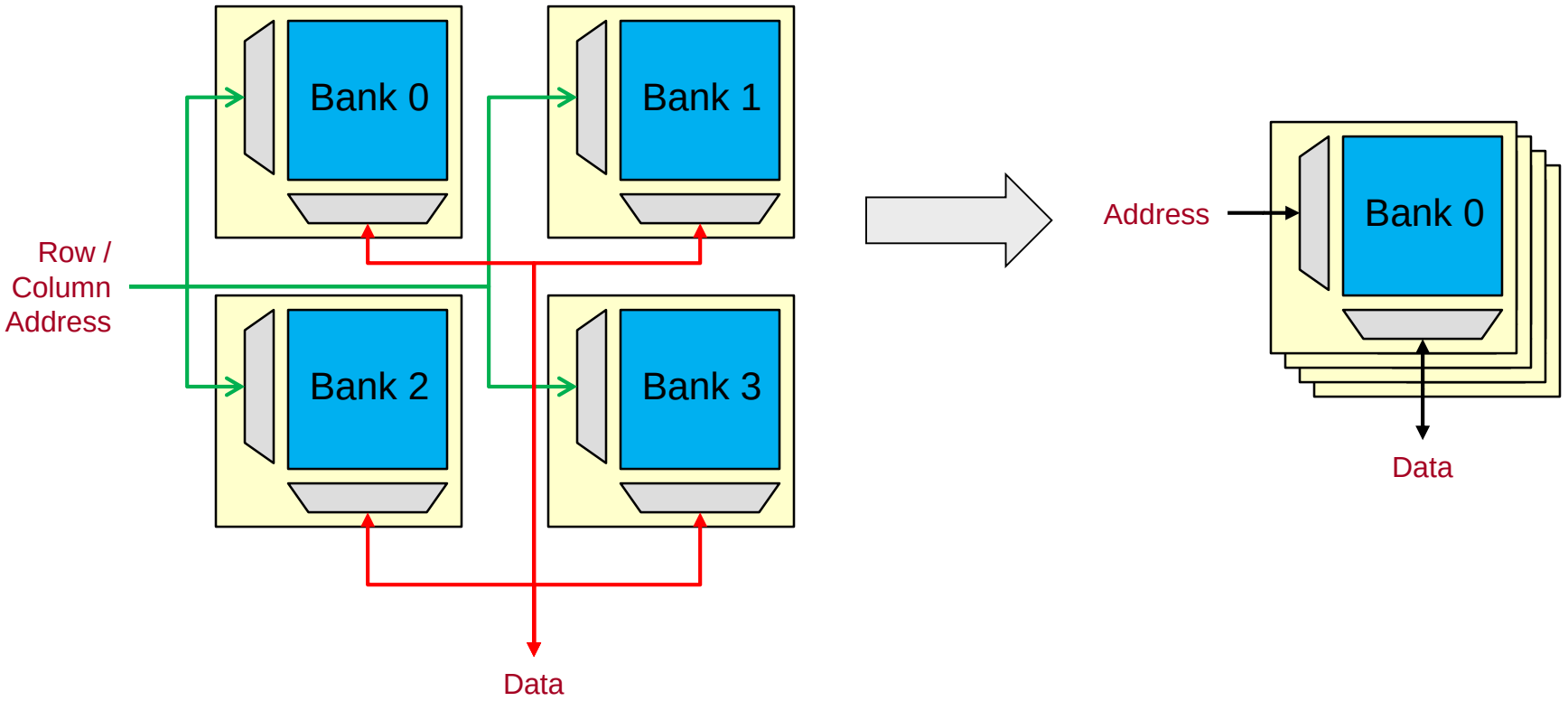




DDR SDRAM (Double-Data Rate SDRAM)
Addition of clock signal. Will get up to '2n' consecutive words in the next 'n' clocks after column address is sent

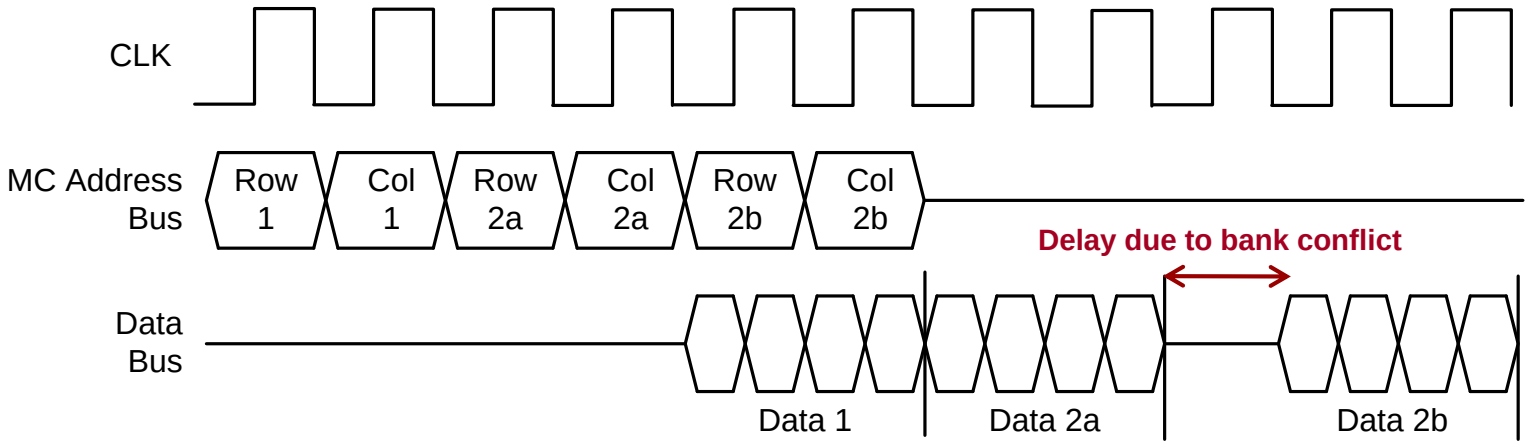
Banking

- Divide memory into "banks" duplicating row/column decoder and other peripheral logic to create independent memory arrays that can access data in parallel
  - uses a portion of the address to determine which bank to access



Bank Access Timing

- Consecutive accesses to different banks can be overlapped and hide the time to access the row and select the column
- Consecutive accesses within a bank (to different rows) exposes the access latency

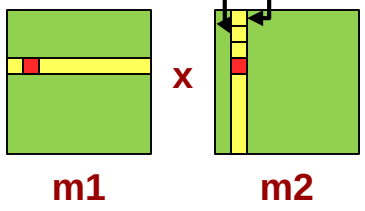


Access 1 maps to bank 1 while access 2a maps to bank 2 allowing parallel access. However, access 2b immediately follows and maps to bank 2 causing a delay.

Programming Considerations

- For memory configuration given earlier, accesses to the same bank but different row occur on an 32KB boundary
- Now consider a matrix multiply of 8K x 8K integer matrices (i.e. 32KB x 32KB)
- In code below…m2[0][0] @ 0x10010000 while m2[1][0] @ 0x10018000

	Unused	Rank	Row	Bank	Col.	Unused
	A31,A30	A29	A28…A15	A14,A13	A12…A3	A2..A0
0x10010000	00	0	1 0000 0000 0001 0	00	0000000000	000
0x10018000	00	0	1 0000 0000 0001 1	00	0000000000	000

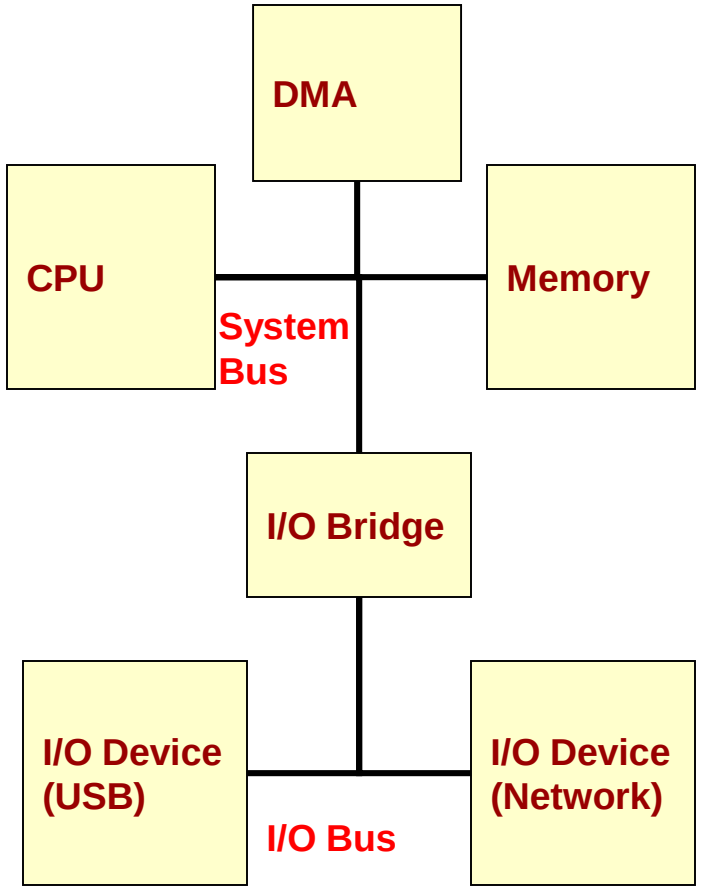



```
int m1[8192][8192], m2[8192][8192], result[8192][8192];
int i,j,k;
...
for(i=0; i < 8192; i++){
  for(j=0; j < 8192; j++){
    result[i][j]=0;
    for(k=0; k < 8192; k++){
      result[i][j] += matrix1[i][k] * matrix2[k][j];
} } }
```

DMA & ENDIAN-NESS

Direct Memory Access (DMA)

- Large buffers of data often need to be copied between:
  - Memory and I/O (video data, network traffic, etc.)
  - Memory and Memory (OS space to user app. space)
- DMA devices are small hardware devices that copy data from a source to destination freeing the processor to do “real” work

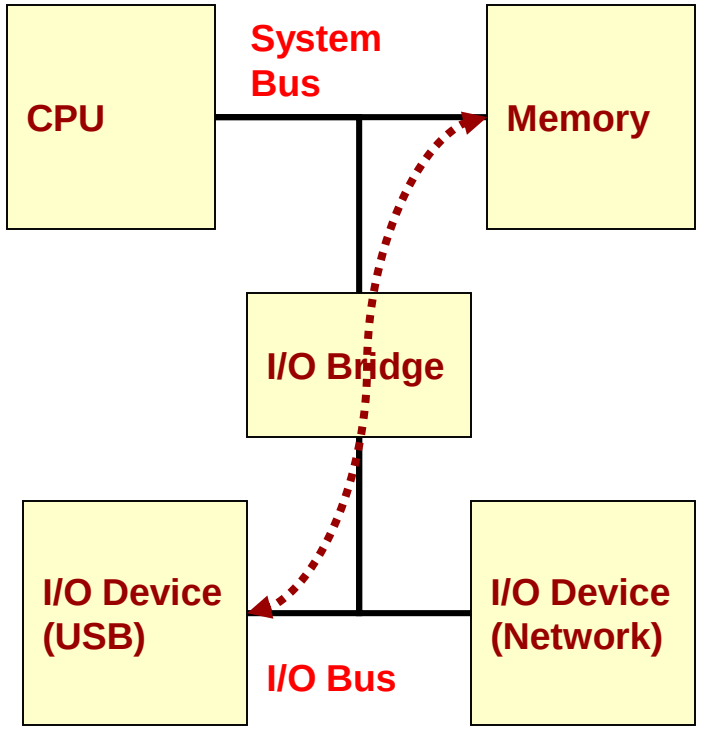


Data Transfer w/o DMA

- Without DMA, processor would have to move data using a loop
- Move 16Kwords pointed to by ($s1) to ($s2)

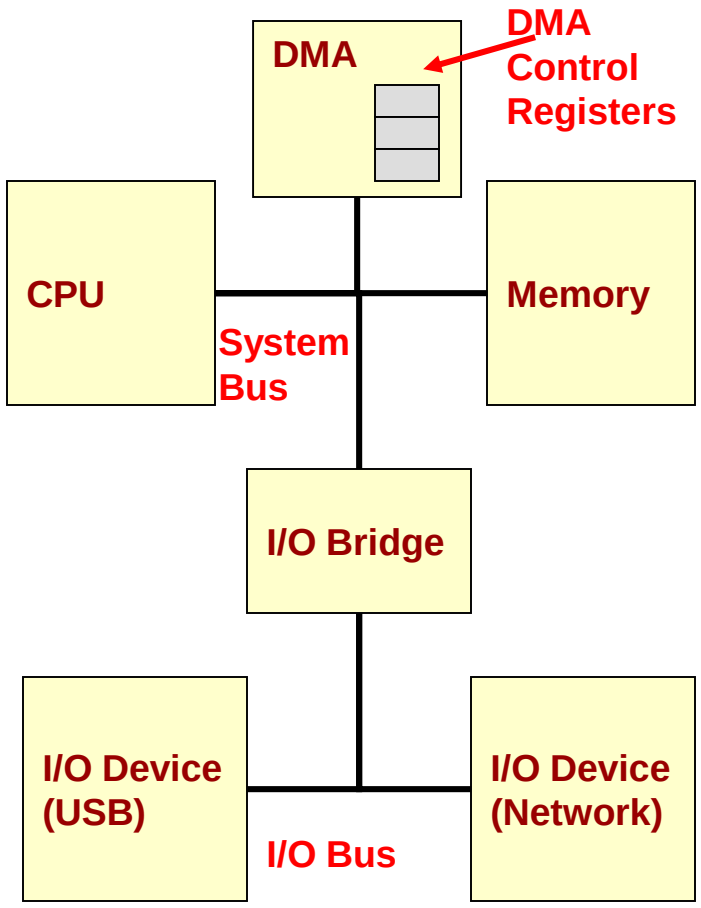
```
        li    $t0,16384
AGAIN:  lw    $t1,0($s1)
        sw    $t1,0($s2)
        addi  $s1,$s1,4
        addi  $s2,$s2,4
        subi  $t0,$t0,1
        bne   $t0,$zero,AGAIN
```

- Processor wastes valuable execution time moving data



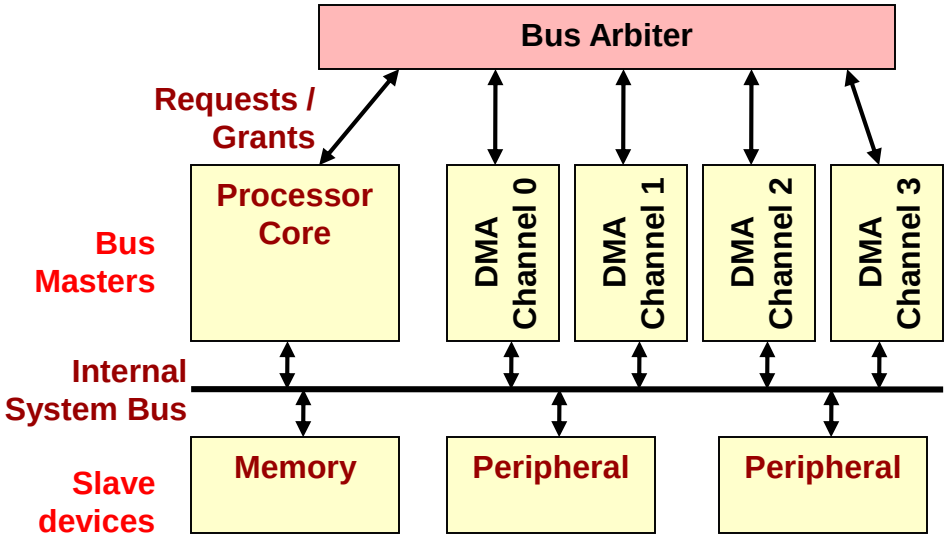
Data Transfer w/ DMA

- Processor sets values in DMA control registers
  - Source Start Address
  - Dest. Start Address
  - Byte Count
  - Control & Status (Start, Stop, Interrupt on Completion, etc.)
- DMA becomes "bus-master" (controls system bus to generate reads and writes) while processor is free to execute other code
  - Small problem: Bus will be busy
  - Hopefully, data & code needed by the CPU will reside in the processor's cache



DMA Engines

- Systems usually have multiple DMA engines/channels
- Each can be configured to be started/controlled by the processor or by certain I/O peripherals
  - Network or other peripherals can initiate DMA's on their behalf
- Bus arbiter assigns control of the bus
  - Usually winning requestor has control of the bus until it relinquishes it (turns off its request signal)

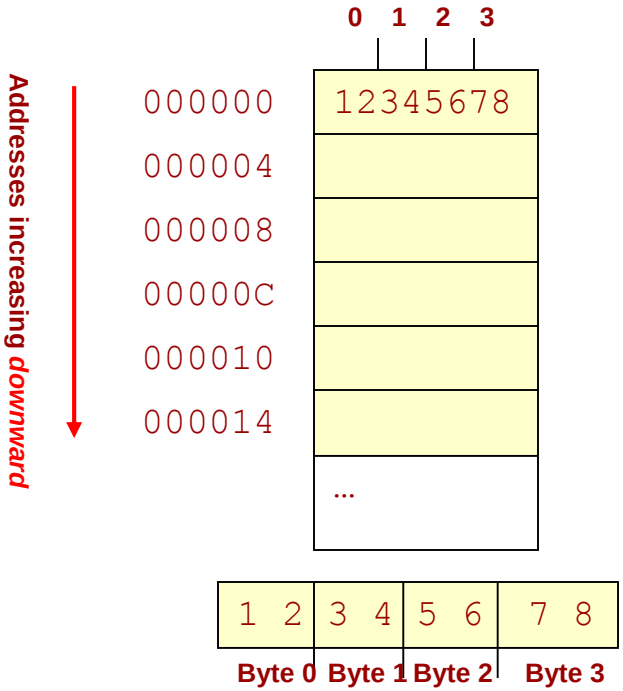


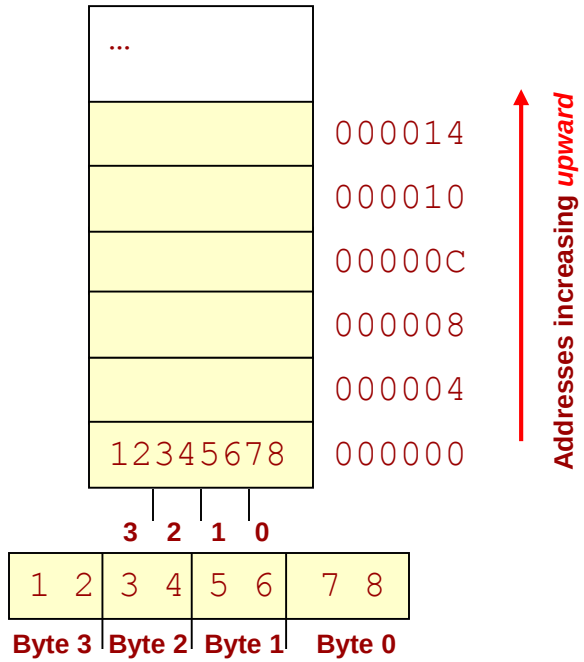
Endian-ness

- **Endian-ness** refers to the two alternate methods of ordering the **bytes** in a larger unit (word, long, etc.)
  - Big-Endian
    - PPC, Sparc
    - *MS byte* is put at the starting address
  - Little-Endian
    - used by Intel processors / PCI bus
    - *LS byte* is put at the starting address

Big-endian vs. Little-endian

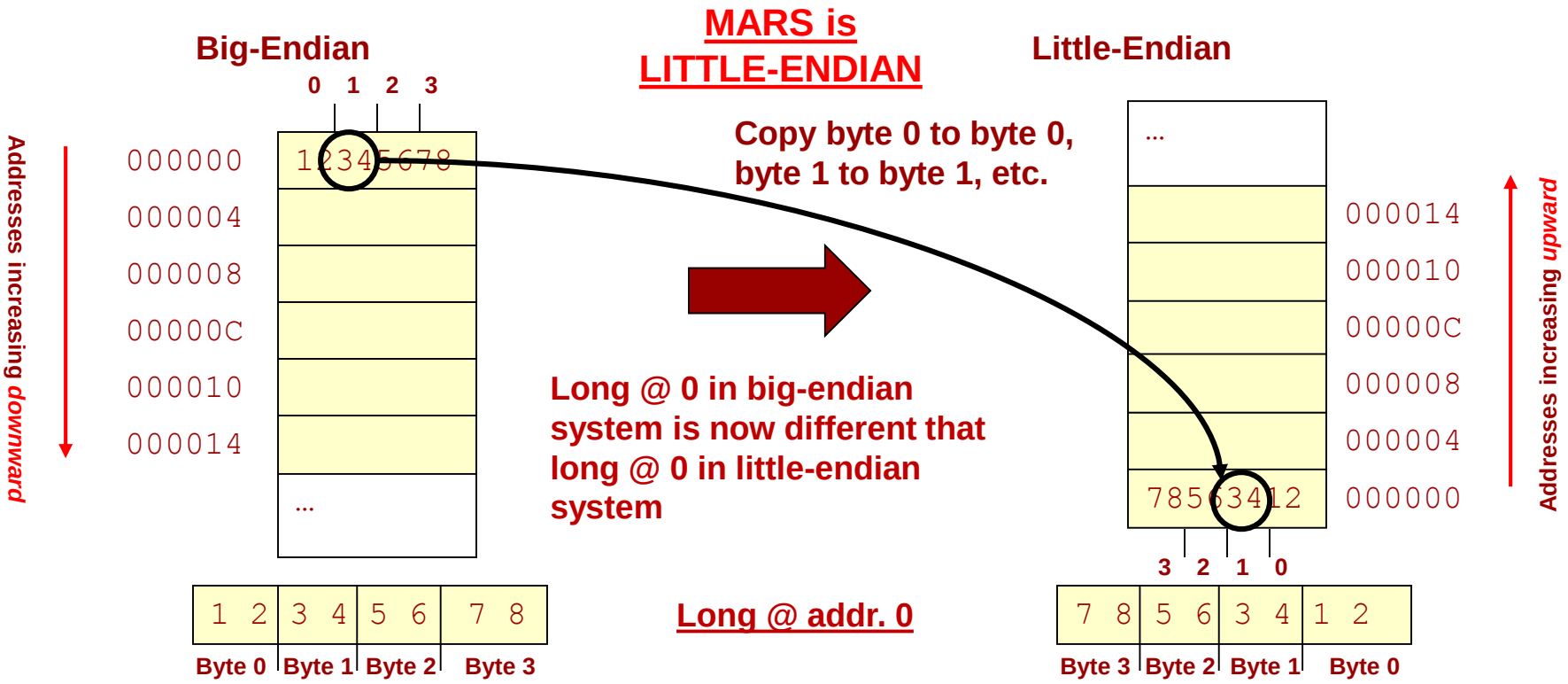
- Big-endian
  - makes sense if you view your memory as starting at the top-left and addresses increasing as you go down
- Little-endian
  - makes sense if you view your memory as starting at the bottom-right and addresses increasing as you go up





Big-endian vs. Little-endian

- Issues arise when transferring data between different systems
  - Byte-wise copy of data from big-endian system to little-endian system
  - Major issue in networks (little-endian computer => big-endian computer) and even within a single computer (System memory => Peripheral (PCI) device)

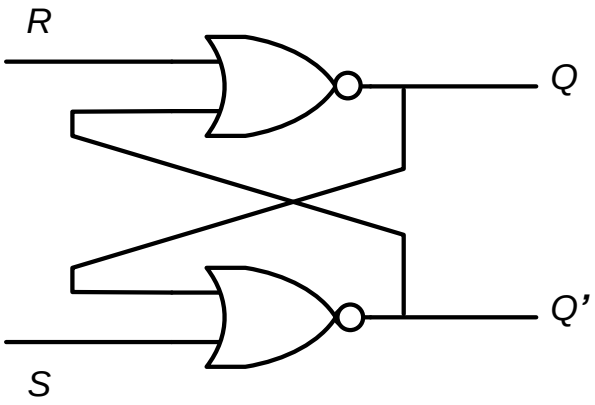


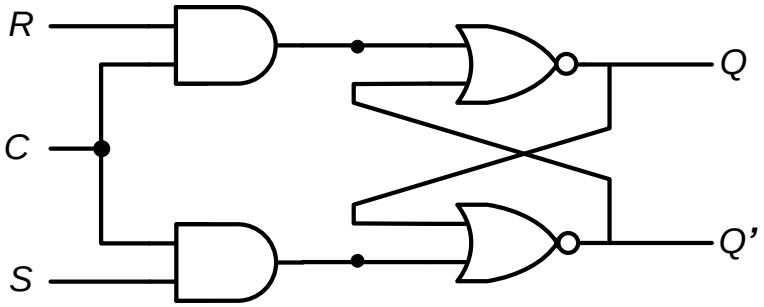
Sequential Cells

BACKUP

Sequential Cell Review

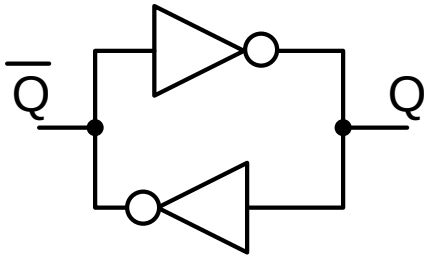
- We've seen how to build basic combinational gates but what about sequential circuits
- We can build the bistables and latches we've seen at the gate level with direct CMOS substitutions
  - Replace each NOR or AND gate with it's CMOS transistor level equivalent

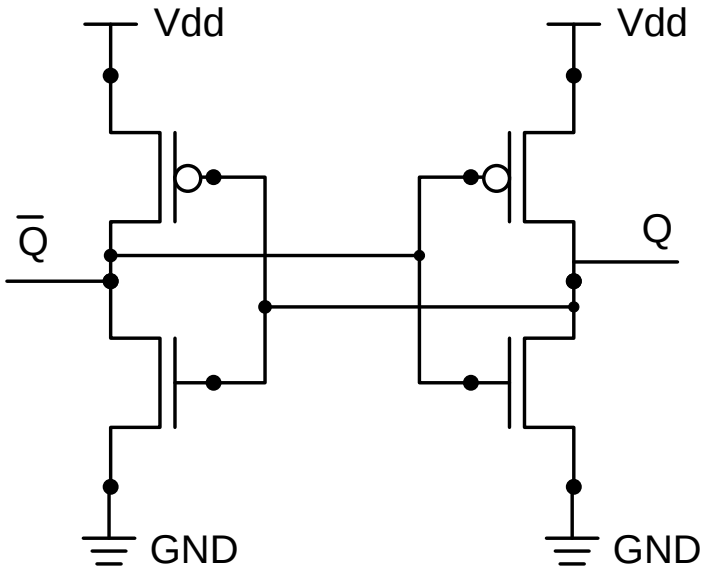




Sequential Cell Review

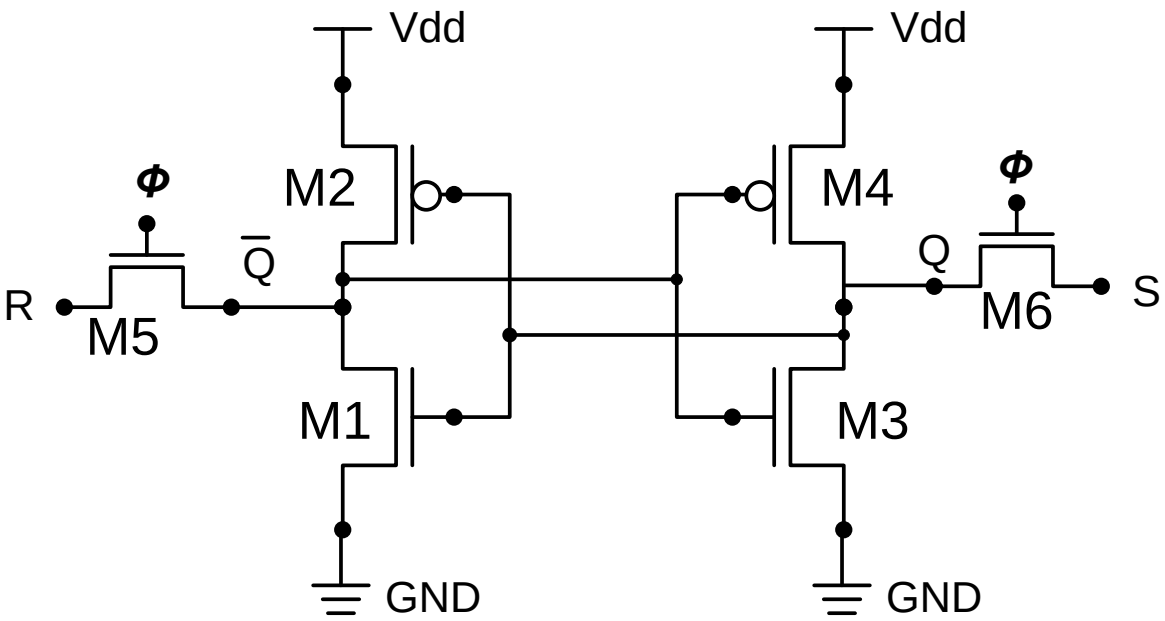
- We can also take advantage of our circuit level understanding of transistors, capacitance, and sizing to build the bistables and latches in other (possibly more efficient) ways
- The simplest sequential cell is just a feedback loop of inverters
  - Problem…how do we change the value, Q?





An SR-Latch

- We can use pass transistors to pull the output Q or Q' to a new value
  - Note: ϕ is the CLK signals



An SR-Latch

- Consider the case when Q(t=0)=0 and we then apply the set input
  - After a short time the pass transistor connected to S and the pull down transistor connected from Q to GND will be in linear mode and form a voltage divider
  - Some analysis will show us that we must make $(W/L)_{M6} > (W/L)_{M3}$ by some appropriate factor to get Q to switch