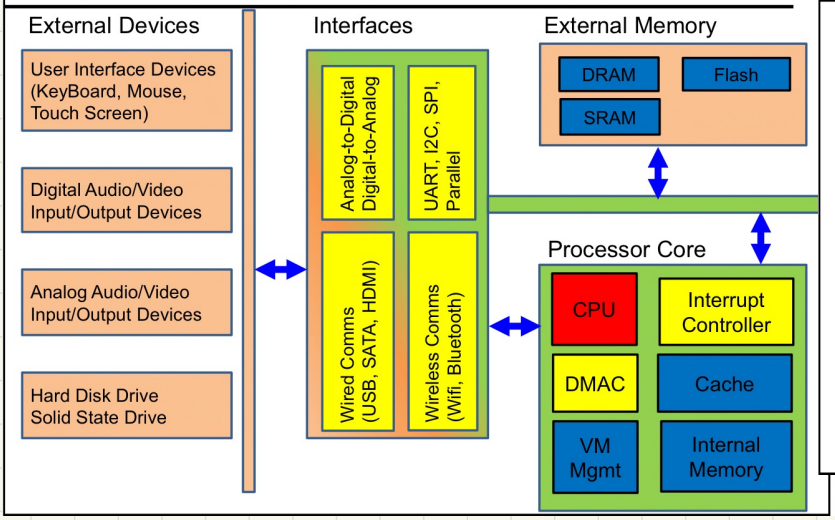
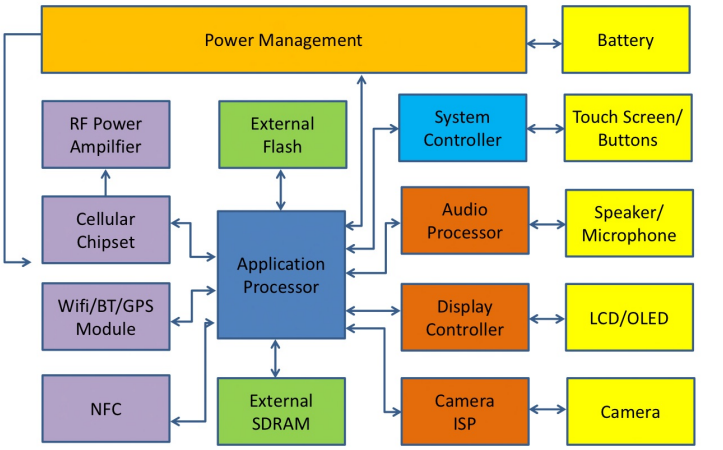
Part 2

Computer System Block Diagram



Example: Smart Phone



Sub-Systems in a Computer System

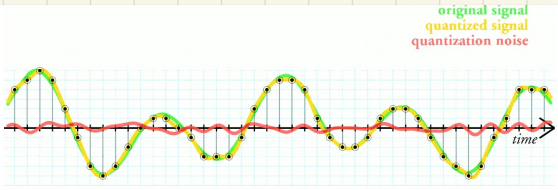
- Processor Core
- Signal Chain Sub-System
  - ADC-DAC
  - Parallel and Serial (UART/SPI) Digital Interfaces
  - Direct Memory Access Controller
  - Interrupt Controller
- Memory Sub-System
  - Semiconductor Memories (SRAM, DRAM, FLASH)
  - Flash memory based Solid State Drives
  - Magnetic Hard Disk Drives
  - Cache Memory Management
  - Virtual Memory Management
- Communication and User Interface Sub-System
  - Wired (USB, SATA, HDMI)
  - Wireless (Wifi, Bluetooth)
  - Input Devices (KBM, Capacitive Touch, Camera, Microphone)
  - Output Devices (Display, Speakers)

manage data transaction

Signal Chain Sub-System

- Signal Chain Sub-System
  - ADC-DAC
  - Parallel and Serial (UART/SPI) Digital Interfaces
  - Interrupt Controller
  - Direct Memory Access Controller

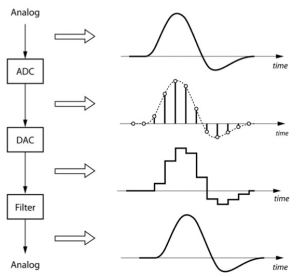
Analog, continous in time domain



- Digital signals are discrete time representation of analog signal.
- Obtained through process of digitization, commonly know as Analog to Digital Conversion.
- Analog signals are digitized to its digital equivalent using a Analog-to-Digital Converter (ADC).
- Digital signal has discrete voltage levels.
- Similarly, analog signal can be reconstructed from digital signal via digital to analog conversion.

- Samples the wave through digital processor
- ADC samples at a Sampling interval where the system assigns a discrete valve

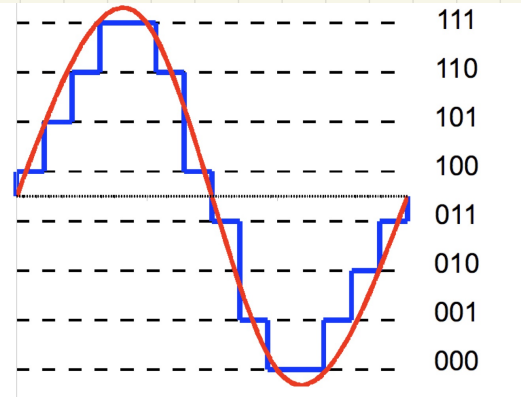
Transformation between Digital and Analog Domain



- Figure on the left shows conversion between analog and digital domain.
- Digital signal is obtained by sampling the analog signal level at discrete time (known as the sampling interval).
- Sampling frequency needs to be at least twice the signal frequency (Nyquist theorem).
- Typically, the analog signal level is assigned to the nearest discrete voltage level allowed in that particular digitization process.
- Analog signal can be reconstructed back by applying a filter on the digital signal.

Wave → ADC → Processor
↳ DAC → filter
↳ output wave

- Because we cannot "hear" bits, only analog



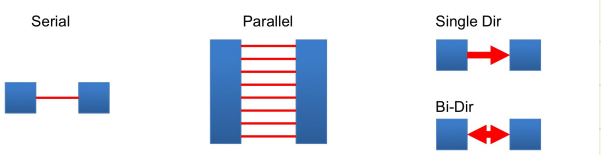
Digital Quantization

- Assign valve to nearest approx. valve from wave to discrete bits

Interface: Boundary where two devices exchange info

- Single-bit Data transfer (Serial)
- Multiple-bits Data transfer (Parallel)

For each mode above, the connection could be Single direction or Bi-direction.



Consideration

- Electrical signal level
- Communication protocol (Handshaking and Data signals).

Electrical Signal level (Safety)

- Primary consideration when connecting two electrical device together.
- Ensure that the output voltage level of output device do not exceed the maximum allowable input voltage level of the input device.
- Electronic devices will either get 'fried' i.e. spoilt or have its reliability reduced if the input voltage is higher than what they are designed for.
- So do check the voltage level which the device input/output is operating on (1.8V, 3V, 5V, 15V etc) before connecting them together.

→ output cannot > input max

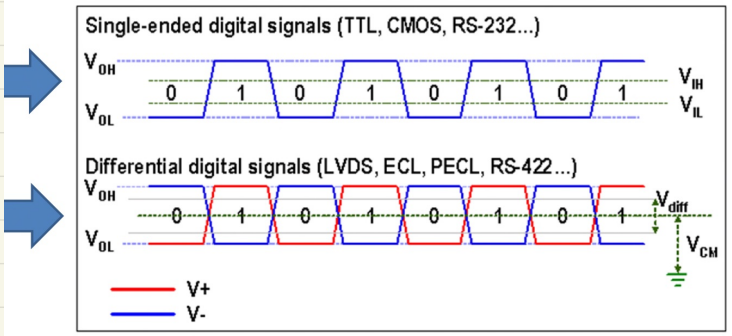
5V => Logic '1" or '0' ?

Four parameters

- VOH.
  - Transmitting a Logic '1' yield a signal level of >= VOH
- VOL.
  - Transmitting a Logic '0' yield a signal level of =< VOL
- VIH.
  - A received signal with level >= VIH will be recognized as a Logic '1'
- VIL.
  - A received signal with level =< VIL will be recognized as a Logic '0'

Differential Signals

- Differential signals has better noise tolerance so is able to be clocked at higher frequency.
- V(CM) = Common Mode Ground.



- Differential signal ↓ noise
  1) Larger margin between V+ & V- to differentiate between 1 & 0 (gap big)
  2) External interference ↑ or ↓ both V+ & V-... difference is unchanged

} V+ > V- ∴ 1
V+ < V- ∴ 0

- Communication Protocols refers to how the data are formatted during transmission.
- Some examples
  - Number of bits in a transmission frame
  - What synchronization to use
  - Data width
  - Types of data and its formatting
- Examples of communication protocols are USB, UART, SPI etc.

Protocol 1	1001001001001000
Protocol 2	1001001001001000

Blue: Synchronization Bits
Green: Data Bits
Red: Error Correction Bits

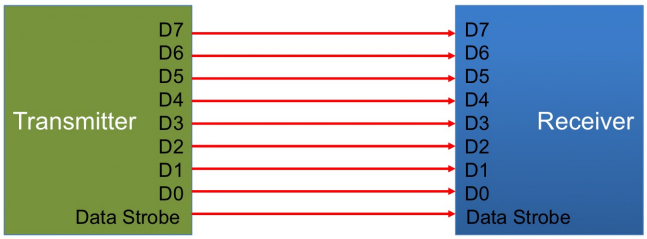
Parallel Interface

Multiple bits of data are transferred simultaneously between two devices.

Synchronous in nature as some sort of strobe signal is needed to inform the receiver when to latch in the data. E.g. rising edge of strobe signal.

Able to achieve higher transfer rate than Serial Interface (using same clock).

But more prone to Signal Skew and Crosstalk (see later slides).



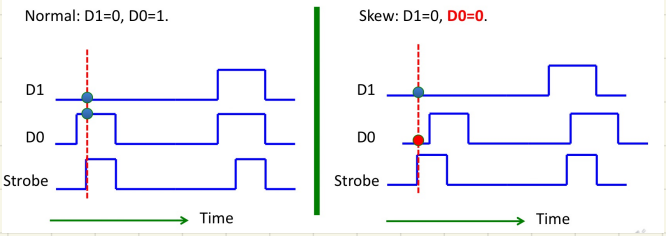
Signal Skew

Information is delayed, leading to incorrect read

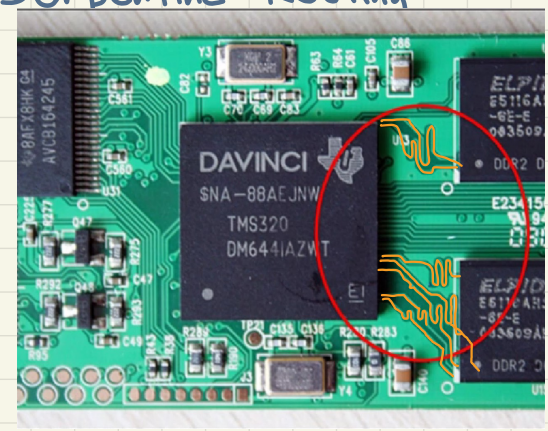
Reason of Signal Skew

1) Varying **propogation delay**, which is the time taken to reach B from A
2) **Capacitance & resistance**
   a) Capacitance is the "charge time"  Time delay is calculated through $T = RC$
   b) Can vary in **PCB length / width** or other **active components**

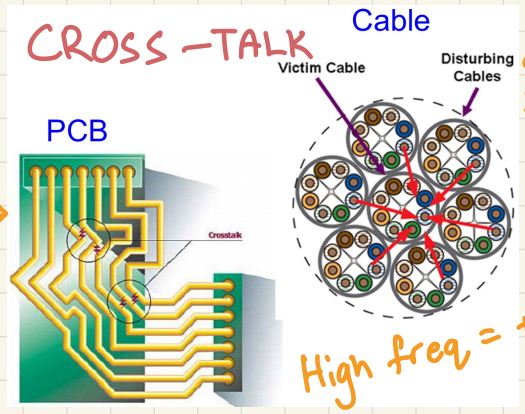
- If for some reason (due to circuit design, external interference), the signal in one or more data lines took different amount of time to reach the receiver, then there is a skew between the signals in the parallel data bus.
- Below example illustrate the effect of signal skew. Data is latched by the receiver on rising edge of the strobe signal.
- This result in wrong data (D0=0 instead of 1) being latched by the receiver.



Serpentine Routing



Also



a) Electrical
b) Electromag. radiation

High freq = ↑↑ crosstalk

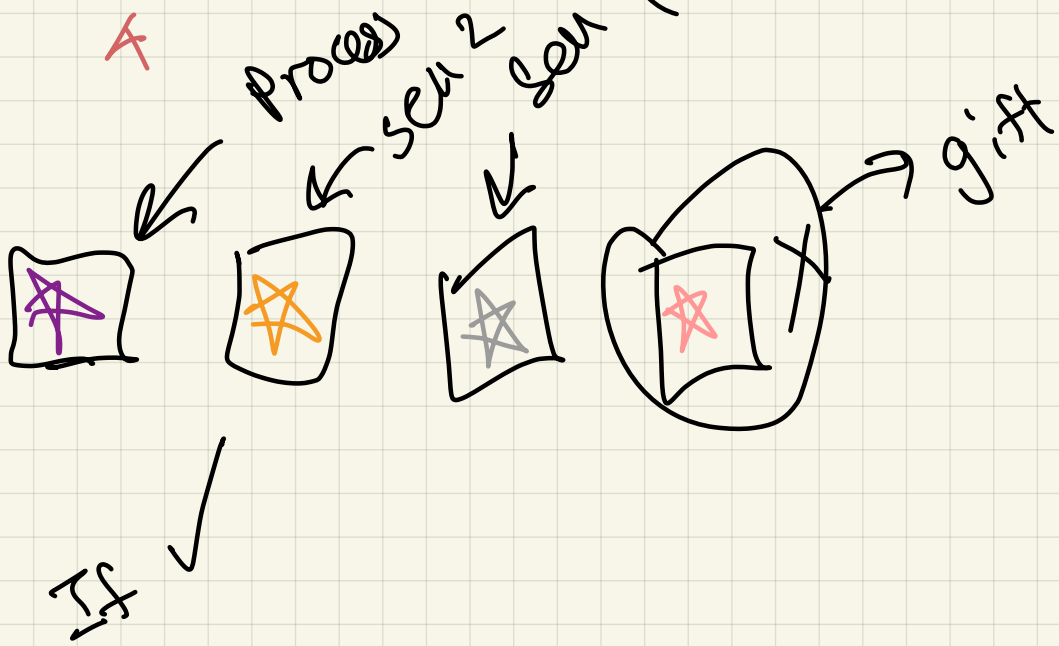
Advantages of parallel

- Fast data transfer rate (more bits can be transferred at one time)
- Hardware interface design tend to be simpler as only strobe signals are needed.

Disadvantages

- Affected by Signal Skew and Cross Talk, which limits the maximum clocking speed and transfer distance.
- Hardware (data cable) can be bulky if data width is large.
- Need more space to route the PCB traces.
- Higher hardware cost compared to Serial data implementation.

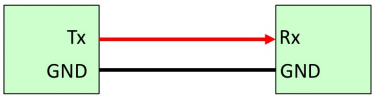
A



If ✓

However,

Serial Data Transfer



Data is transferred one bit at a time over a single data line. Comparatively, parallel data interface transfer multiple bits simultaneously.

Less affected by signal skew and crosstalk because there are less electrical wires involved compared to parallel data transfer. Hence, able to support higher frequency clocking.

Data transfer rate lower (compared to parallel interface) given the same clock rate since only one data line is available.

Advantage

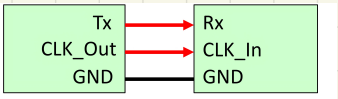
- Less affected by signal skew and crosstalk because there are less electrical wires involved compared to parallel data transfer. Hence, able to support higher frequency clocking.
- Able to transfer data reliably over a longer distance.
- Lower cost since less wires and connectors are needed.

Disadvantage

- Data transfer rate lower given the same clock rate since only one data line is available.
- Hardware interface design typically more complex as it need to handle serial to parallel conversion (Processor typically only process in bytes or multiple bytes).

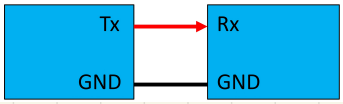
CLOCK

Synchronous



Common clock
Rx dependent on Tx,
Master - slave
-SPI

Asynchronous



Pre-fixed clock frequency

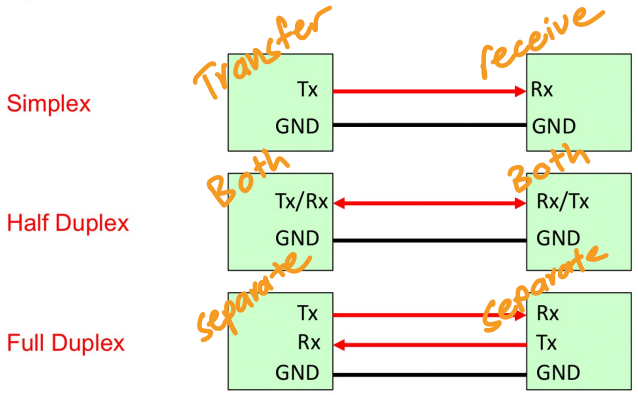
-UART
-USB
-RS232

Types / Duplex

Simplex: Data transfer in one direction only.

Half-Duplex: Data transfer in both direction, but RX and TX is mutually exclusive.

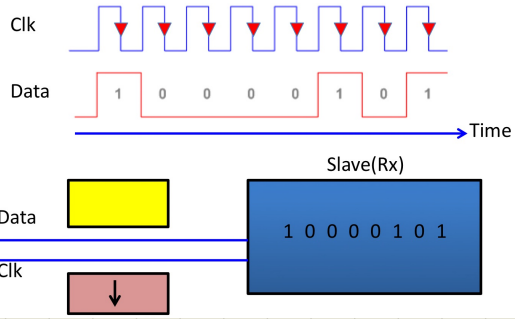
Full Duplex: Simultaneous RX and TX



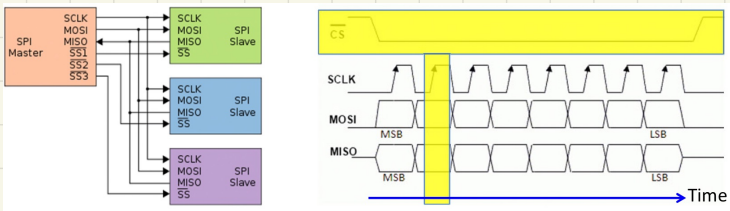
Synchronous

Potentially allows faster transfer rate since no data overhead is needed to synchronize the transfer.

Data latched at falling edge of clock signal.
MSB Tx first.



SPI BUS



To start the transfer
- Slave Select (SS) has to be pulled Low.

Data transfer
- Data on MOSI and MISO latched in on rising/falling clock edge (configurable)
- MOSI: Master-Out-Slave-In (Master Output, Slave Input)
- MISO: Master-In-Slave-Out (Slave Output, Master Input)

Allow multipe slaves via use of multiple Slave Select.

a) SPI is ONE to MANY

b) $\overline{SS}$ low activates slave

c) MISO & MOSI xfers data

Asynchronous

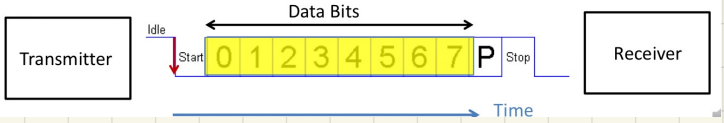
- No common clock is provided between transmitter and receiver.
- Prior to the transmission, the receiver needs to know the transmitting clock rate and the number of bits that are to be transferred with each data packet.
- Special SYNC words are used to indicate START/STOP condition.
- Upon receiving the START SYNC Word, the receiver then use its own local clock to track the timing.
- Potential skew issue between the two local clocks as transmission progress.
- Asynchronous Transmission typically also uses SYNC word/bits to provide occasional time stamp for receiver to synchronize its clock to the transmitter clock (helps to reduce clock skew between the two clocks).

- SYNC word is like a "reset"

Modern systems send UART through USB

UART Transmit

- During **IDLE** State, the data line is in 'powered' state i.e. Logic '1'.
- The transmitter send a **START** pattern (logic '0') to alert the receiver.
- Sending a logic '0' from idle state (logic '1') will result in a falling edge on the data line. This is typically used by the receiver to detect the start of transmission.
- This is followed by the actual **DATA** at a frequency known to the receiver. The transmitting clock rate is also known as the **baud rate**, and determines the number of bits transmitted per second.
- A **PARITY** bit (optional) may also be sent for the receiver to check the integrity of the data packet.
- A **STOP** bit (Logic '1') terminates the transmission.



Universal Asynchronous Receiver Transmitter (UART)

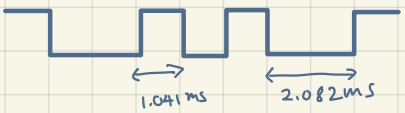
- One of the most commonly used serial interface.
  - PC Serial COM Port (RS232) uses UART protocol.
  - Many USB devices uses a Virtual COM Port implementation to connect to the PC.
  - Communication interface between PC and many processor development boards e.g. Arduino Board, TIVA-C Launchpad etc

Start | 7 data bit | Parity | stop
Start | NEXT 7 bit | Parity | stop

* Sending second word requires pull down of data line again

Baud Rate : 9600 = 9600 bit per sec

Interval : $\frac{1}{9600}$ = 1.041×10^{-4} = 1.041 ms



Parity Bit

- If parity scheme is enabled, the receiver will also sample the parity bit and checks for parity error.
- If even parity scheme is used, there should be an even number of '1's in the data field and the parity field.
- Hence, if there is an odd number of '1's in the data field, then the parity bit transmitted should be a '1' to make the total even.
- If receiver receives odd number of '1's in a even parity scheme, it'll flag a Parity Error, meaning one or more bits in the transmission is wrong.
- Vice versa for odd parity scheme.
- The receiver then samples the **STOP** bit(s), if a '0' is detected instead of '1', then receiver will flag a Framing Error.

Even parity : 10100101

Odd parity : 10100100

If 0 detected, framing error!

0 xxxxxxx 1/0 1
Start | 7 data bit | Parity | stop

UART Tx Example

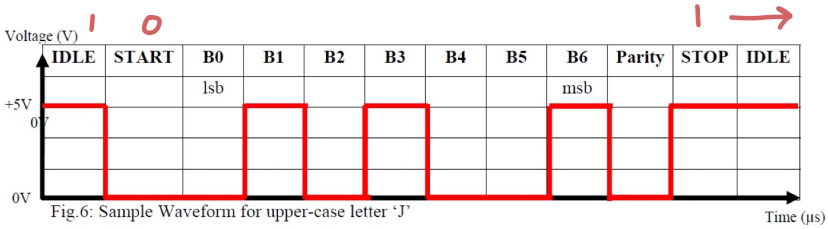


Fig.6: Sample Waveform for upper-case letter 'J'

- Figure 6 above correspond to the configuration 7O1 (7 Data, 1 STOP and Odd parity).
- Capital Letter 'J' => Ascii value = 0x4A
- 0x4A => 100 1010 (binary)
- Presented in LSB first => 0101 001 } Transmit LSB 0 first
- IDLE=1, START=0, STOP=1.
- Parity bit logic depends on number of 1's in Data Field and the Parity Scheme (Odd or Even) used.

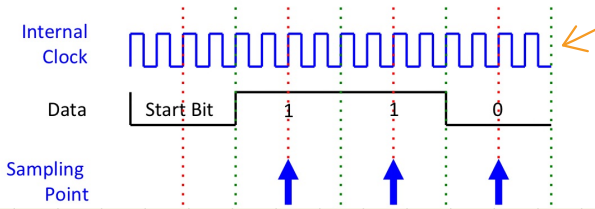
7 O 1 : Odd parity

7 E 1 : Even parity

7 N 1 . No parity

UART Receive

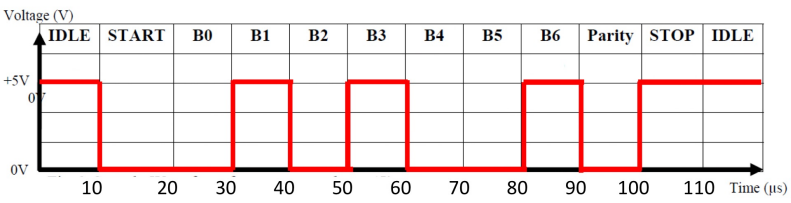
- Receiver monitors the Data line for the Start Bit. In real world design, the falling edge on the data line will trigger an interrupt in the microprocessor to start the receiving process.
- Needs to know the baud rate in order to sample the data bits correctly.
- Internal clock of the UART typically run at a multiple e.g. 16X of the baud rate so as to timed the sampling closed to the middle of each data bit.
- Below is an example of UART receive with internal clock running at 4X baud rate (for illustration only). Internal clock rate is typically faster than 4X baud rate in real world implementation.



Clock Starts at falling edge

1) Calculate middle of pulse
   ∴ 4 ÷ 2 = 2
2) Wait 2x cycle
3) Sample
4) Sample every following 4x

Tx baud rate = 1/(10 us) = 100000 bps. Configuration = 7O1.



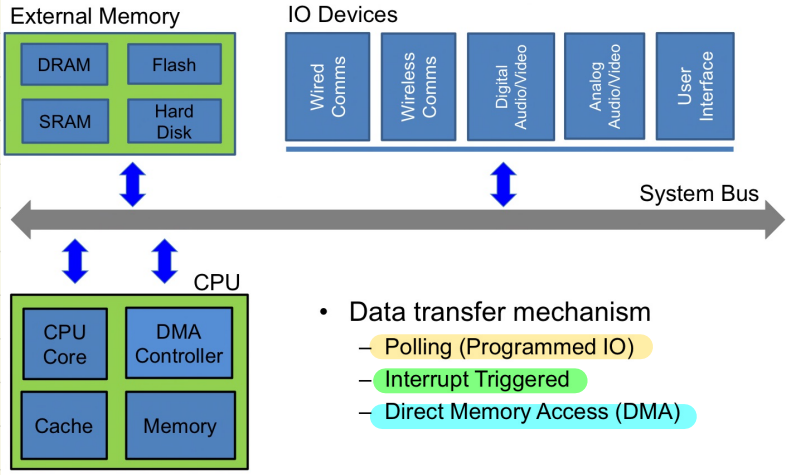
misaligned baud rate

- RX Data = 1001010b
- RX Data = 0011000b ?
- **Information Sampled @ 200000 bps:** } receiver thinks it samples twice
  00001100**1**1**0**0001100**1**1

→ Samples it according to 7O1 ... legit transmission!

Data transfer Mechanism

Data Transfer Mechanism



- Data transfer mechanism
  - Polling (Programmed IO)
  - Interrupt Triggered
  - Direct Memory Access (DMA)

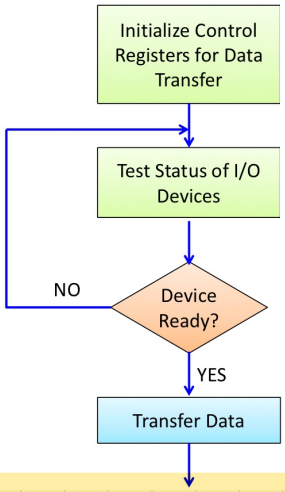
Polling Technique

During polling, if other devices want to communicate with the CPU, they are denied

- CPU polls a certain I/O port continuously (using software) for data or readiness of the port to perform a data transaction.
  - For example, the CPU polls the printer port continuously to see if the printer is ready to accept data.
  - If it is ready, the CPU writes a data byte to the printer port. Else, it waits.
- CPU has full control and dedicate 100% of its resource in the whole data transfer process and does nothing else.

Polling Technique - Flowchart

- CPU performs all necessary initialization.
- CPU polls the I/O device for its readiness to perform data transfer.
- If I/O device is not ready, CPU continue to wait in the loop to check if device is ready.
- If device is ready, CPU make the data transfer and exit the loop.



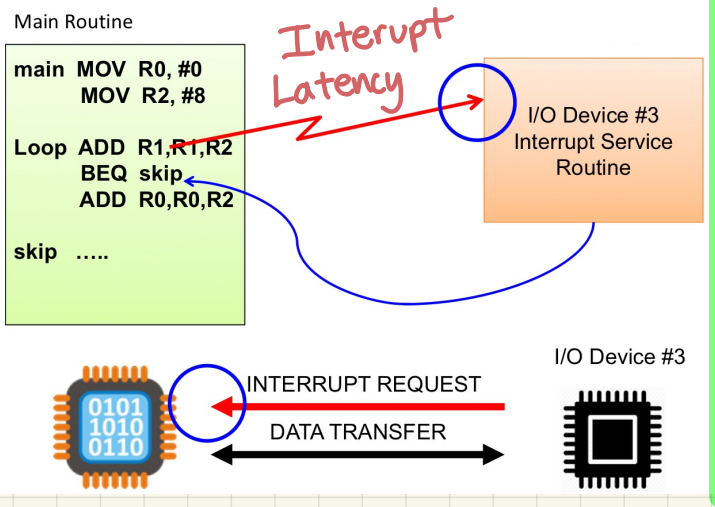
Pros/Cons of Polling Technique

- Advantages
  - Programmer has complete control over the entire process.
  - Easiest method to test and debug.
- Disadvantages
  - Since the CPU waits in a loop, it cannot perform any other task until data transfer is completed.
  - Program execution of CPU held up while waiting for I/O device to get ready.
  - Inefficient use of CPU resources.

Interrupts

- A signalling mechanism that allows internal and external peripherals to alert the CPU that attention is needed.
- Could be in the form of a external/internal electrical pulse or change in internal register status.
- Once the CPU receive the signal, known as interrupt request, it would then need to decide if it wanted to service the request.
- If CPU decides to service interrupt request, it will follow up with the series of procedures to handle the interrupt event.
- Interrupt mechanism is typically used trigger the CPU to start some operation, e.g. data transfer to memory, status registers, control registers etc.

Interrupt Triggered Data Transfer



Interrupt Vector Table (where to go?)

Interrupt Vector Table

0	Reset ISR Addr
1	...
2	...
3	...
4	...
5	...
6	...
7	Ext INT 3 ISR Addr
8	SPI ISR Addr
9	UART ISR Addr
A	

- How does the CPU know the location of the corresponding interrupt service routine for each interrupts?
- The starting address of each interrupt is stored in a table known as the interrupt vector table
- Each interrupt has a unique index to the vector table, e.g. UART interrupt could be in index 9, so if a UART interrupt occurs, CPU will know where to branch to by checking index 9 in the vector table.
- The values and interrupt source in the table on the left are for illustration purpose only, different processor has different indexing for their interrupts.

Pros/Cons of Interrupt Triggered Technique

- **Advantages**
  - Efficient use of CPU resources as it does not need to monitor I/O device status.
  - CPU can continue with other tasks between interrupts.
  - Allows prioritization and pre-emption.
- **Disadvantages**
  - More hardware interface circuitry required between I/O device and processor.
  - Program is slightly more complex and difficult to debug.

Overview of interupt

Interrupt Control Flow

- A signal from external/internal peripheral, or a change in status of some special registers notifies the CPU that some event has occur and ask for CPU's (immediate) attention.
- If the CPU decide to service the interrupt, it will suspend its current program temporarily.
- CPU look up the interrupt vector table to check the starting address of the interrupt service routine (ISR) for the corresponding interrupt.
- CPU would save a copy of the processor context, i.e. the current value of various registers, this is to allow the interrupted routine to continue its execution after returning from the ISR.
- CPU then proceed to execute the ISR linked to the interrupt that was triggered.
- Once the ISR is completed, CPU restore the save context, returns to the interrupted routine and continue from where it had left previously.

- The ISR will perform some operation e.g. Data Transfer.
- ISR is typically a very short routine so as not to suspend the main program for too long.
- After servicing the ISR, CPU will return to the previous program and continue from it branched off.
- It is possible for CPU to receive multiple interrupt requests simultaneously since it typically interface to multiple devices.
- In this case, some arbitration scheme has to be designed to decide which interrupt to service first (priority, first-come-first-serve etc).

Direct Memory Access

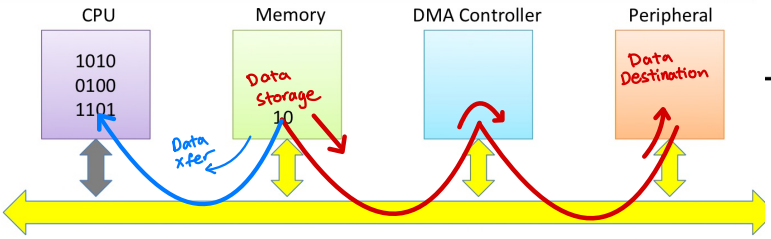
- DMA controller (DMAC) is added to relief CPU of the data transfer task.
- DMAC has dedicated hardware that could move data more efficiently than CPU in scenarios where complex address manipulation is required. E.g. de-interleaving Left and Right Channel Audio data.
- If there are no conflict in hardware resources used, it is possible for data transfer via DMA and CPU execution to occur simultaneously.
- If there is a conflict in hardware resources used, e.g. both DMAC and the CPU needs the system bus, then access will be given to the one with higher priority.
- Who has the higher priority and whether the priority is configurable depends on processor design and is thus processor specific.

— CPU should not be used for easy tasks like data x-fer.

∴ Impliment DMAC to assist

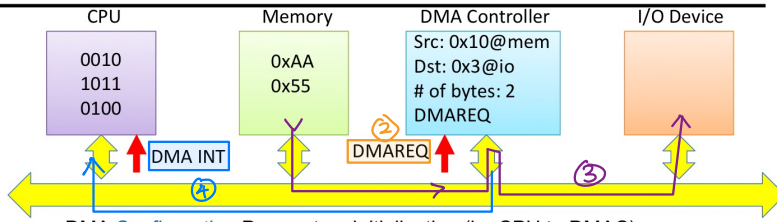
fight for system bus control

DMA Controller (DMAC)



- DMAC is a Data Bus Controller module that performs data transfer independent of CPU, between Memory and memory, I/O Peripheral and I/O peripheral, Memory and I/O Peripheral.
- Generates address and initiates read/write operation between devices mentioned above.
- Note that 'Peripheral' here could be internal or external peripherals
- The DMAC shown here uses the Fetch and Deposit DMA mechanism where the data goes into the internal buffer of the DMAC before being transferred to the Destination.

Basic DMA Process



- DMA Configuration Parameters initialization (by CPU to DMAC)
  - Source Address
  - Destination Address
  - Amount of data to transfer
  - DMA Trigger signal (DMAREQ, interrupt, software bit etc)

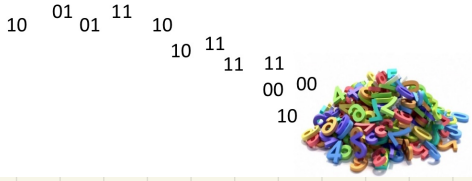
Required items to start

1. CPU proceed with its own task while DMAC wait for DMA Trigger Signal.
2. Once DMA Trigger occurs, DMAC request to take over system bus.
3. DMAC transfer data according to configuration.
4. DMAC notify CPU of data completion (typically via interrupts) and release system bus.

DMA through 3 types, can be mixture

a) Burst

- The DMA controller gains control of the system bus and transfer multiple units of data before returning control of the bus to the CPU.
- Note that the burst could be the entire block of data DMAC is tasked to transfer, or a subset of the block, i.e. it may take a few burst to complete the transfer of the entire block.
- CPU may continue to operate as long as it does not need access the particular system bus that DMAC is using.
- If CPU needs to access the ssytem bus, CPU may be suspended till DMAC has completed its data transfer.
- Fast data transfer rate but may render the CPU inactive for longer period of time.



- Burst entire block ō data or short blocks
- If CPU need data bus during burst, CPU will halt

- **Burst**
  - Allow faster data transfer
  - CPU may be suspended for longer period of time
  - May not be suitable for Real-time application
  - Suitable for application where transfer is bursty e.g. HDD file transfer.

b) Cycle Stealing

- DMAC releases the system bus after transferring one unit of data.
- Depending on processor design, DMAC tends to execute the data transfer
  - Between CPU instructions
  - Between Pipeline stages
- CPU may be suspended if it need to access the system bus but suspend time is shorter as only one unit is transferred at one time.
- Transfer rate is slower than in Burst Mode but will CPU will only be inactive for very short period of time.
- Favoured in application which requires CPU to be responsive e.g. real-time security status monitoring.

Between CPU Instr

CPU	Instr1	Instr2		Instr3		Instr4
DMAC			Data		Data	

Between Pipeline Stages

CPU	Fetch Instr	Decode Instr		Fetch Operand	Exe Instr	
DMAC			Data			Data

- Time avail for data xfer is lower
- Less data
- Faster CPU responsiveness

- **Cycle Stealing**
  - Interleaving DMA data transfer with CPU instructions allow CPU to continue executing its program while DMAC is doing the data transfer.
  - CPU performance lower due to interleaving of DMA transfers, but would still be more responsive than in Burst mode.
  - Slower data transfer rate than Burst mode.
  - Suitable for Real-time application.

c) Transparent Mode

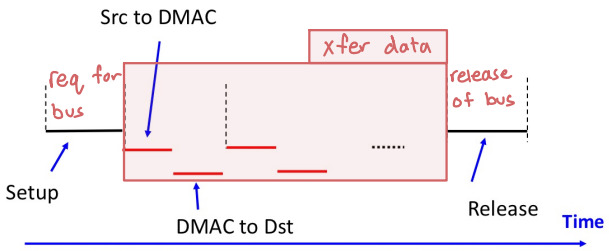
- DMAC transfer data only when CPU is not using the data bus.
- Zero impact to CPU performance in terms of data bus access.
- Potentially the slowest transfer rate among the three modes.
- More Complex hardware needed to detect when CPU is not using the data bus.
- The CPU activity detection technique could also be used by some processors as a cue to trigger a burst transfer only when CPU is not using the system bus.

- Only when CPU not using bus
- Complex hardware to detect when bus is free

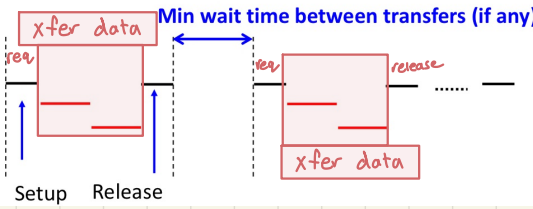
- **Transparent**
  - Potentially would not affect CPU performance at all.
  - Slowest data transfer rate but best CPU respond
  - More complex hardware needed to detect when CPU is not using the bus.

Fetch and Deposit (Timings)

Burst

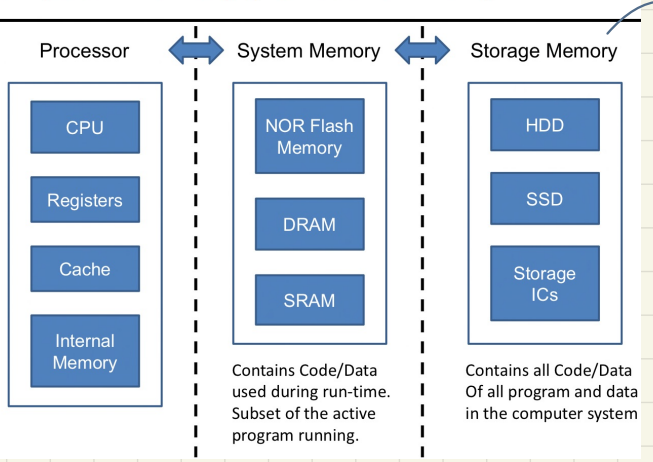


Cycle Stealing



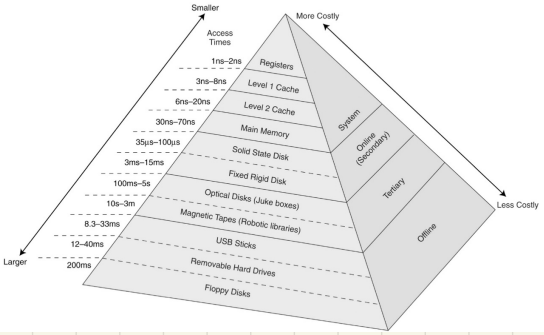
Memory

Computer Memory (Functional View)



Non-volatile, when shutdown memory maintains

The pinnacle has the fastest access time, but is also more costly.
Memory Access Time below are only for illustration. These changes with improvement in technology.



Semiconductor Memories

- Memories based on semiconductor integrated circuits (IC).
- Used as processor internal memories and system memory in a computer system
- Processor internal memories
  - Registers
  - Buffers
  - Cache
  - Internal System and Storage Memory (SRAM, DRAM, Flash based)
- Types of memory
  - SRAM
  - DRAM
  - Flash Memory
  - Solid State Drives (based on Flash Memories)

Volatile

- Data is lost when electric power is removed.
- Temporary storage.
- Typically used as system memory.
- We will look at Random Access Memories such as Static-RAM (SRAM) and Dynamic-RAM (DRAM).

Non-volatile

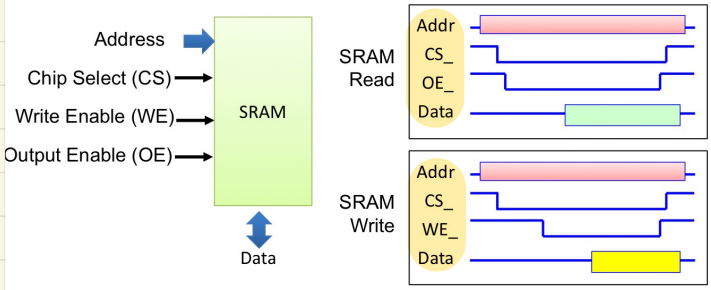
- Data is retained even if electric power is removed.
- Permanent storage.
- Typically used as main storage.
- We will look at FLASH, magnetic hard-disk specifically.

Random Access Memory (RAM)

- Static RAM (SRAM)
  - Static Random Access Memory
  - Data stored as long as supply is applied.
  - Large (4 to 6 transistors per cell).
  - Fast.
  - Low power consumption (active and standby)

- Dynamic Random Access Memory (DRAM)
  - Periodic refresh required.
  - Small (1 to 3 transistors per cell).
  - Slower.
  - Higher power consumption due to need for periodic refresh operation to maintain data integrity of memory cells.

Static RAM (SRAM) Access

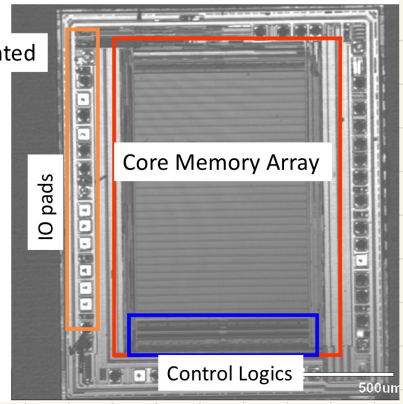


Addr	Specifies Address of memory location.	
Data	Data to be read/written.	Typically 8,16 or 32 bits.
CS	Control signals	Chip Select (Enables or disables the chip).
WE		Write Enable (Allows data to to be written)
OE		Output Enable (Allows SRAM to output data)

Commercial SRAM Chip
Cypress CY62128, 1 Mbit

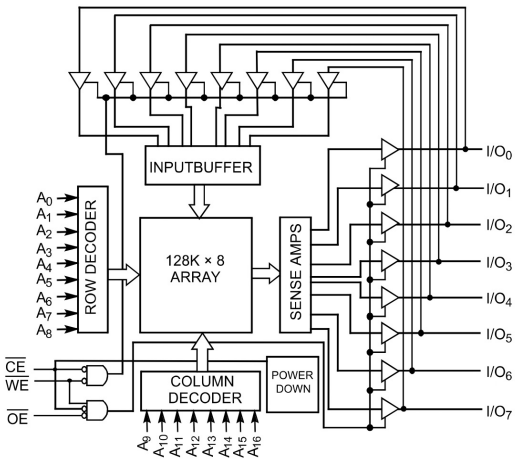
Decapsulated

Decapsulated silicon die
Laser scanned image, 5x magnification



- **Memory** array takes up **most** of the **real estate**.
- Leading Edge Semiconductor Process is usually first tested with Memory Design.

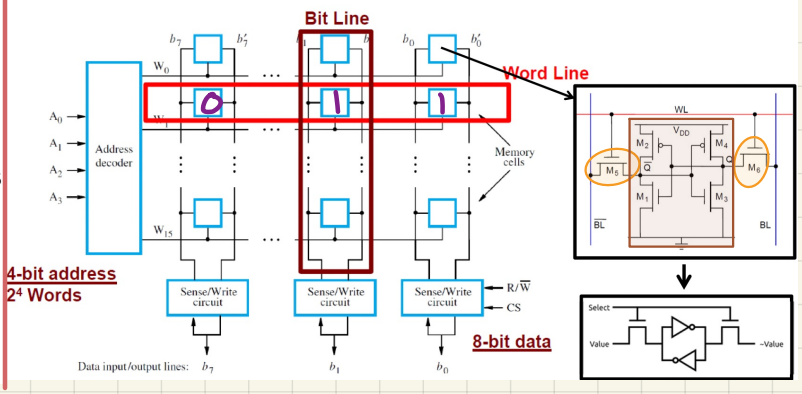
SRAM Internal Circuitry



- Memory array
- 1M SRAM cells for the chip shown.
- Control circuitry
- Decoders
- Sense amplifiers
- Input/Output multiplexers

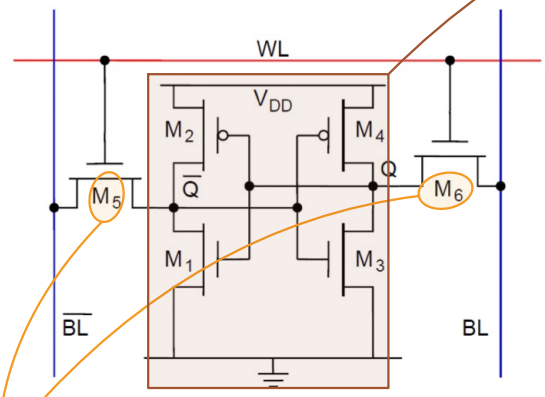
SRAM Cell

- Memory cells are organized in arrays (rows), and are accessed via the word lines and bit lines.
- Each individual SRAM Bit consist of 6 Transistors (typical design).



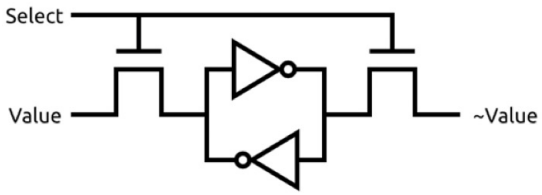
SRAM Cell

This transistor is used to store the data

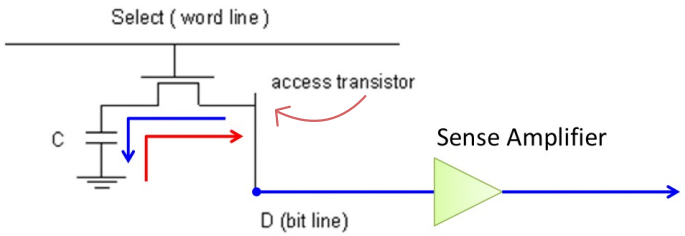


- There are six transistors
- M1, M2, M3, M4, M5 and M6
- Word line (WL) is derived from Address Decoder Output. It controls the Read/Write process
- The actual data are placed on the differential bit lines (BL and BLB). This is connected to the Data Bus.

- The data bit is stored in M1, M2, M3 and M4 (which is equivalent to two invertors connected as shown on the right).
- M5 and M6 are pass transistors.
  Control into in & out of the cell



Dynamic RAM (DRAM)



- Single Transistor Design
- DRAM uses a capacitor as its storage element.
- The Transistor is used to control charges flowing in and out of the capacitor during the Read and Write process.
- Write Process
  - To store a Logic '1': Enable the Transistor, transfer charge into capacitor.
  - To store a Logic '0': Enable the Transistor, discharge the capacitor.
- Read Process
  - Enable the Transistor. Measure the capacitor charge using a sense amplifier.

- DRAM cell only consist of one single transistor and a capacitor.
- The capacitor is used to store the charge while the transistor controls the flow of charges in and out of the capacitor.

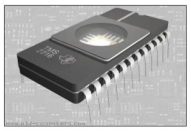
DRAM – Maintaining Data Integrity

- DRAM Read Process destroys information stored on capacitor
- The process of measuring charges on a capacitor also effectively discharged it i.e. data is destroyed.
- Hence, the original data has to be re-written back after every read.
- Periodic refresh is needed as the stored charge "leaks" with time.
- The basic DRAM is more or less obsolete in the market today. It is replaced by its synchronous version called Synchronous DRAM (SDRAM).
- Difference between SDRAM and DRAM is that the former make use of a clock signal from the host to synchronize data transfer, enabling faster transfer rate. SDRAM also has a pipeline architecture that allow faster, overlapping operations.
- Other enhancements of SDRAM includes its double date rate versions DDR, DDR2, DDR3 SDRAM, which could reach transfer speed of more than 2G transfers per second.

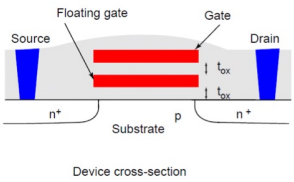
Non-Volatile Memory (EEPROM)

EPROM and EEPROM

- EPROM (Erasable Programmable ROM)
  - Earliest floating gate transistors are implemented as Erasable Programmable ROM (EPROM) devices.
  - Need to put device under ultra-violet (UV) light to erase the stored program.



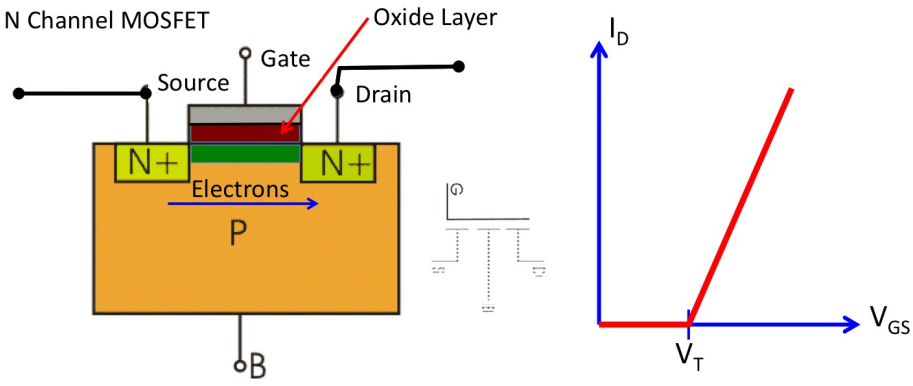
[Source] www.old-computers.com



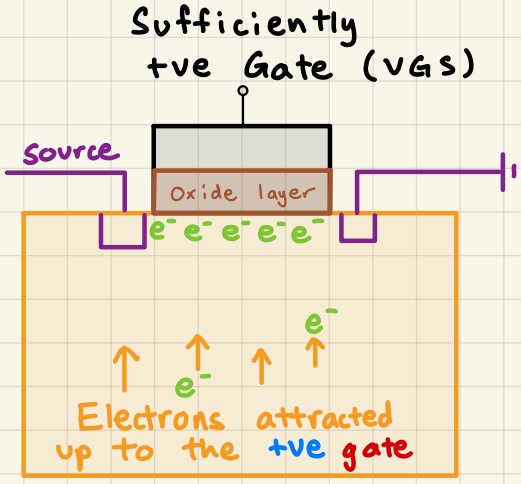
Device cross-section

- EEPROM (Electrically Erasable PROM)
  - Advancement in process technologies make it possible to reduce the oxide thickness (t_{ox}).
  - Can electrically program or erase device.
  - Hence, named Electrically Erasable Programmable ROM (E^2PROM).

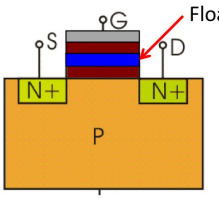
MOSFET



- MOSFET (Metal-Oxide-Semiconductor Field Effect Transistor)
- For N-Channel MOSFET, if Gate-Source Voltage (Vgs) > Threshold Voltage (Vt), a conductive channel of electrons (inversion layer) will be formed and current will flow if a positive voltage is applied across Drain and Source.



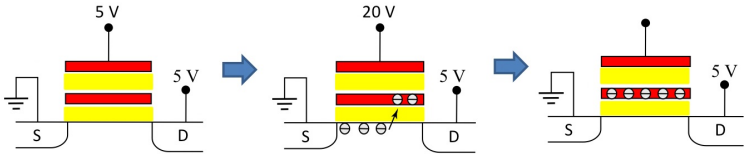
Floating Gate Transistor (FGT)



Erased State
No Excess Electrons in Floating Gate

Programming
A large positive voltage applied at the gate, causing excess electrons to tunnel through the oxide and embed in the floating gate

Programmed State
Excess Electrons are trapped in Floating Gate even after power off



Erased State — Programming — Programmed State

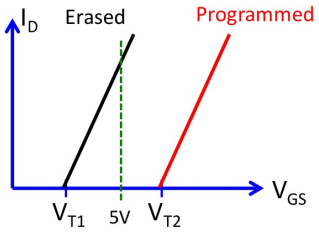
- In order to program the FGT, a large positive voltage e.g. 20V is applied to the gate.
  - This will cause some electrons in the substrate of the FGT to tunnel through the oxide and embed into the floating gate.
  - Note that the voltage value used in this slide are for illustration purpose only, actual value may differ depending on the semiconductor process.
- After programming, the FGT is in the programmed state and the excess electrons in the floating gate will continued to be trapped there even if the power is removed.
- The presence of excess electrons in the floating gate has effect on the threshold

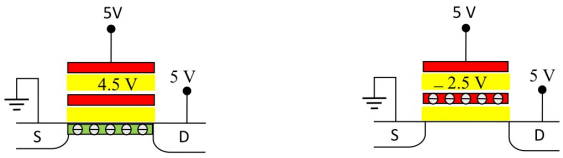
FGT Threshold Voltage

As an illustration, let's say a 5V input at the Gate of a FGT in erased state is sufficient to turn ON the FGT.

"Programming" increases the threshold voltage of Floating Gate Transistor so the same 5V is now unable to turn ON the FGT.

Voltage values used are for illustration purpose only.



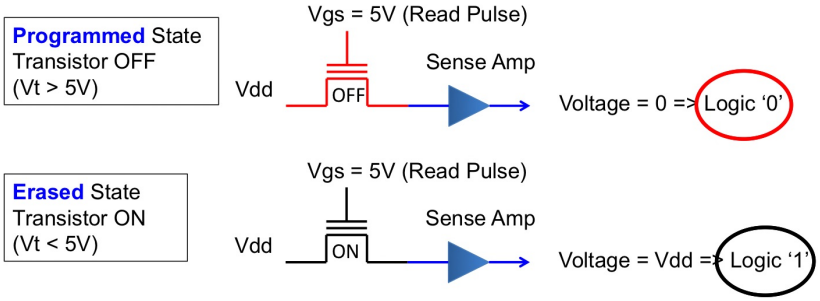


Erased State — Programmed State

if the FGT is in the programmed state instead,
Which means excess electrons are present in the floating gate, and these negatively charged electrons will lower the resultant voltage potential, meaning user now has to apply a larger voltage in order to attract the same number of electrons to form the conductive layer.
Applying the same 5V will therefore not be able to do the job, as illustrated in the diagram, the same 5V input at the gate will result in a much lower voltage potential at the floating gate when the FGT is in programmed state. This as mentioned, is due to effect of the negative charges of the excess electrons.

Reading of Stored Data in Floating Gate

- With a positive Gate Voltage (Vgs) = 5V
  - Transistor OFF if floating gate is programmed (contains charges).
  - Transistor ON if floating gate is erased (no charges).
- 5V value below is for illustration only. Actual Vt value in real world may varies depending on the doping of the transistors.

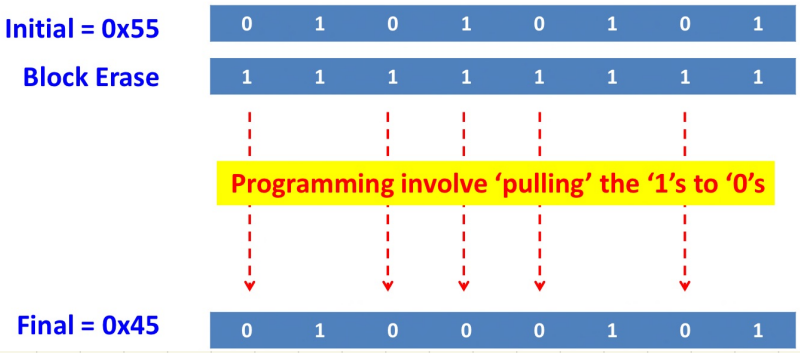


TL: OR, how to use

- Input charge ½ way inbtwn V_{T_1} & V_{T_2}
- If erased, $I_D = 0$
- If programmed, $I_D = 1$

Example of Programming a FGT based memory

- Flash 'programming' can only modify the cell content from '1' to '0'
- To modify the cell content from '0' to '1', an erase operation has to be done at block level
- Example: To Program a value of '0x45' when initial value in flash memory is '0x55'



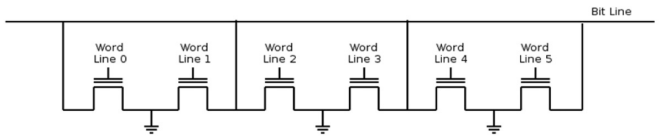
— To erase, charge all cells

- Zap the cells you don't want to use

Non - Volatile Memory (flash)

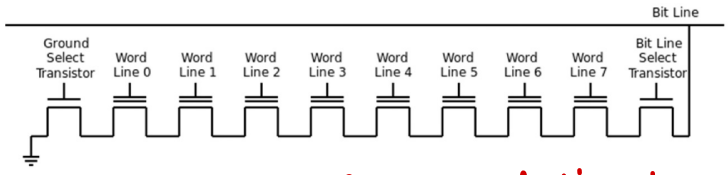
- Can be erased in larger blocks size compared to EEPROM (which typically has smaller page/block size).
- Since Erase cycle is comparatively slower than other operations (Read/Write), Flash memory has a faster speed than EEPROM when performing write operations for large block of data.
- Flash also cost less than EEPROM.
- Suitable for system requiring large amount of non-volatile memory.
- Two main types of Flash in the market
  - NAND Flash
  - NOR Flash

NOR Flash



- Cell behaves like a NOR gate.
  When any one of the Word Line > Vt(Prog), Bit Line output = 0.
- Supports Execute in Place, i.e. Program code stored in Parallel NOR Flash can be executed directly without the need to transfer to internal RAM first.
  - Allows Random Reading of memory data using only Address information (no additional Commands needed).
- Need Special Commands (issue in Write mode) in order to perform operations other than Data Read. E.g. Program, Erase etc.
- Allows random word/byte programming. But erasure has to be done at block level. Typical Block size: 64KByte, 128KByte, 256KByte
- Use as system memory to store program code or general storage memory

NAND Flash



All word line!

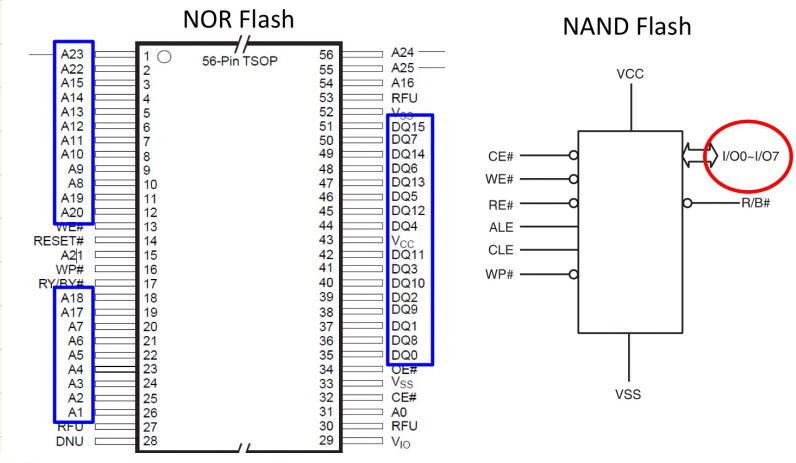
- Cell behaves like NAND gate.
  - Bit line output = '0' only when ALL Word Line > Vt(Prog).
- Does not support execute in place operation.
  - Data has to be accessed one page at a time.
  - Command issued to open a particular page, followed by which byte(s) is/are needed in the page.
  - NAND chips uses a single bus to carry Address and Data.
- Lower Cost per Bit than NOR. Used mainly as Main Storage Memory.
  - On Board Main Storage, USB Flash Drive, SSD etc

more compact also!

NOR

- Processor provide the address of target location it wants to access
- Nor flash output data on dataline

NOR and NAND Flash chip pin-out



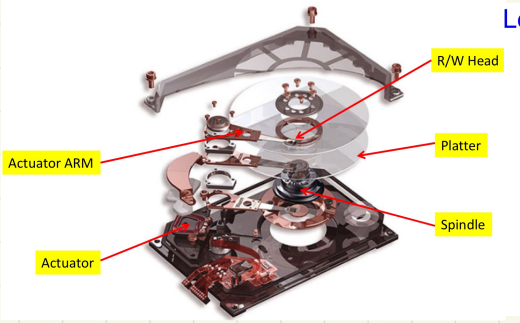
NAND

- Only one data bus
- All command, address & data information through one bus
- All operations are multi - step

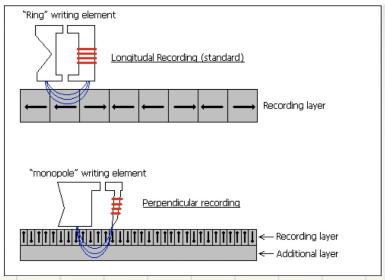
System vs Storage Memory

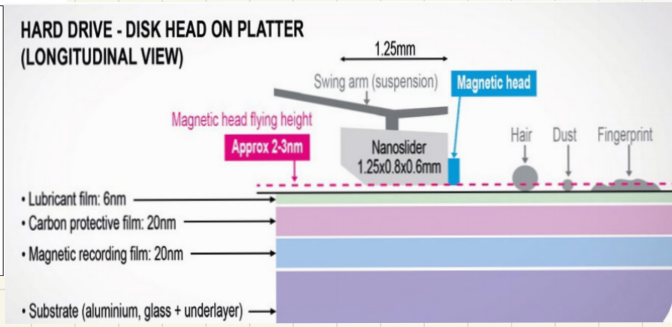
- System Memory
  - Used to store runtime program and code is executed directly from the system memory
  - This typically refers to
    - Internal SRAM/DRAM or NOR Flash
    - External memory that supports XIP (NOR Flash, SRAM, DRAM)
  - DRAM technically speaking does not support XIP by itself but processors that has a DRAM interface will have a DRAM controller that handle the necessary translation to allow execution of code directly from the DRAM.
- Storage memory
  - Used to store all program and data in a computer system
  - Cannot run code directly from storage memory, needs to be transferred to system memory before code execution
  - All memory types can be used as storage memory

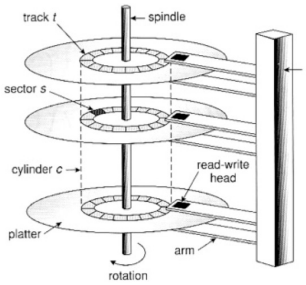
Hard Drive



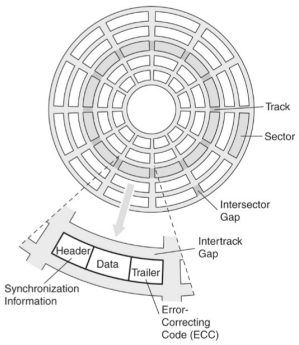
Longitudinal vs Perpendicular Recording





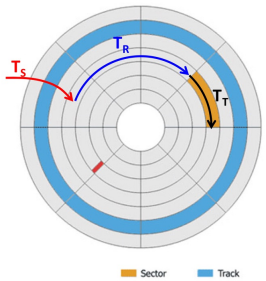


- Computers often use magnetic hard disks for large secondary storage devices.
  - One or more platters on a common spindle.
  - Platters are covered with thin magnetic film.
  - Platters rotate on spindle at constant rate.



- Data is stored on the surface of the platter in concentric rings called tracks.
  - Gaps between tracks to minimize interferences from adjacent tracks.
- Tracks are divided into sectors.
  - Minimum data block size is a sector.
- Stored data consists header, data and trailer.

- Seek time (T_S)
  - Time taken for the head to move to the correct track.
- Rotational Delay (T_R)
  - Time taken for the disk to rotate until the read/write head reaches the starting position of the target sector.
- Access Time (T_A)
  - Time from request to the time the head is in position ($T_S + T_R$)
- Transfer Time (T_T)
  - Time required to transfer the required data after the head is positioned.



HDD Transfer Rate

- T_R is dependent on the rotational speed, Revolutions Per Minute (RPM), of the disk. For calculations, RPM is usually converted to Revolutions Per Second (RPS). i.e. RPS = RPM/60
- For a random section, average rotational delay $T_{R,AV}$ may be calculated as

$$T_{R,AV} = \frac{0.5}{RPS} \text{ seconds}$$

- T_T is dependent on the rotational speed of the disk, the Track Density D_T (number of sectors per track), Sector Density D_S (number of bytes per sector) and the number of bytes N for the transfer.

$$T_T = \frac{N}{RPS * D_T * D_S}$$

ratio of data to information in one track

Start from random sector

Chance that instant hit

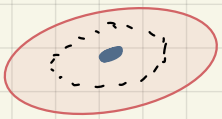
Time = $\frac{0}{RPS}$ s

Chance that barely miss

Time = $\frac{1}{RPS}$ s

(need wait full revolution)

$D_T * D_S$ = Number of bits in one



- If data is spread across multiple sectors, the seek time repeated many times

Example

- A magnetic hard disk rotates at 15000 RPM, with the following properties:
  - Average Seek Time, T_S = 4ms
  - Track density, D_T = 500 sectors per track
  - Sector density, D_S = 512 bytes per sector
- Calculate the total time T_{TOTAL} it takes to read a 3 KB file stored in consecutive sectors on the same track?

3 × 2^{10}

[Solution]

RPS = 15000/60 = 250 per second

Access time $T_A = T_S + T_R$ = 4 ms + (0.5/250) = 6 ms

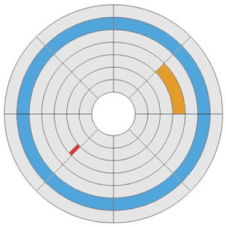
Transfer time T_T = 3072/(250*500*512) = 48 µs

Total time, $T_{TOTAL} = T_A + T_T$ = 6.048 ms

Notice T_A >> T_T. This example shows that accessing file whose data is distributed across sectors on different tracks on the magnetic hard disk wou potentially incur more time.

Revise this!

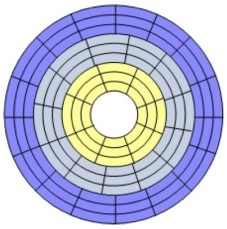
Physical Layout



- Early HDD
  - Physical Layout refers to the actual layout design on the HDD.
  - Equal number of sectors per track and each sector has the same data size.
  - Same-numbered Tracks from different surfaces formed a cylinder.
  - The early hard disks were implemented using this topology to simplify the controller design.

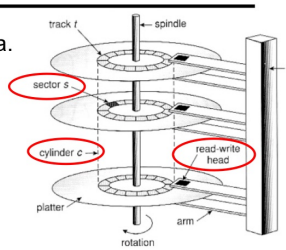
Modern HDD

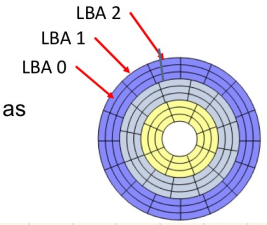
- Having equal number of sectors per track means that sectors at the outer tracks are wider.
- Waste physical space as bit density of those sectors are not optimal.
- Zone bit recording technique addresses space wastage.
- Tracks are divided into zones, with differing number of sectors per track for different zones.



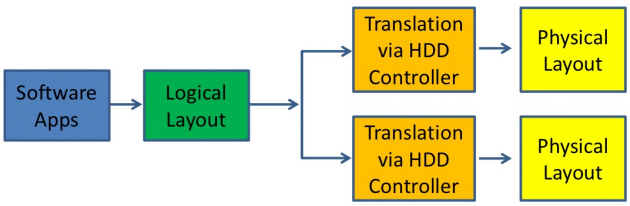
Logical Layout

- How the software see and address the HDD data.
- Address translations are needed to map the physical to logical locations. Two addressing scheme are CHS and LBA.
- Cylinder-Head-Sector (CHS)
  - Legacy scheme using the old HDD physical structure.
  - Made Obsolete in recent standards due to addressing limitation and more complex formatting.
- Logical Block Addressing (LBA)
  - Simple linear addressing starting from LBA=0 as first block, LBA=1 as second block and so on.
  - 48-bit LBA standard allows addressing up to 128PByte. 1PByte=2^50 Bytes.





Logical vs Physical Layout

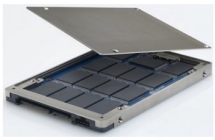


- Logical layout is needed in order to provide a common interface to all software developer
- If only physical layout is available, software has to be tuned for every HDD that uses a different physical layout
- Having a standard such as LBA allows the software developer to write application that can be used in any HDD.
- HDD vendor will provide the necessary firmware in their controller to perform the logical to physical layout address translation.

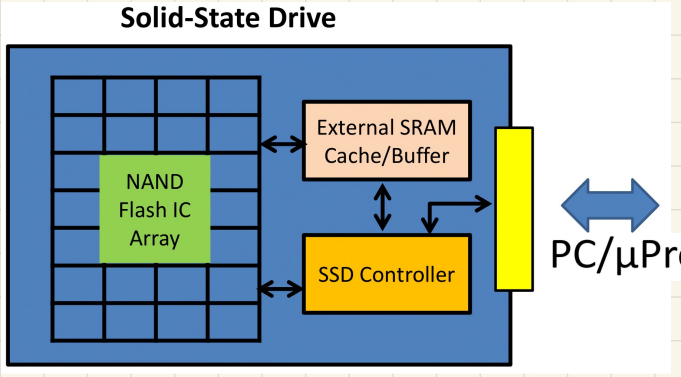
Modern Day HDD

- Detail physical layout not given due to complex zone bit recording.
- Only the average transfer rate is given these days.
- Use of Cache and Buffers to speed up data transfer.
- However, concept and limitation of the magnetic HDD's physical design still valid
  - Actuator ARM is fixed and access has to be sequential, once data is missed, it has to wait for a disc revolution.
  - Seeking from track to track takes time as the R/W head needs to align to the starting sector.
  - More efficient to read in blocks rather than random access
  - Vulnerable to motion

Solid State Drive (SSD)



- Solid-state drives (SSD) are becoming popular
  - Memory array based on NAND-FLASH or NOR-FLASH.
  - Still more expensive than magnetic hard-disk.
- Flash memory has limited program-erase cycles (3,000 to 1,000,000).
- Various techniques used to extend the life of SSD
  - Wear levelling is a technique used to extend the life of the SSD disk by distributing data erase/write operations evenly over the entire disk.
  - Use External RAM as buffer to minimize the number of writes to Flash in SSD.
  - Error Correction Code. Ability to recover from one or more bits of error in media.
- MTBF (Mean Time To Failure) of recent SSD is comparable to that of HDD.



HARD DISK

- Pros
  - Lower Cost Per Bit,
  - Almost infinite Erasure cycles
- Cons
  - Consist of Moving Mechanical Parts so more prone to crashing if HDD is dropped or shaken.
  - Heavier and larger physical profile.
  - Slower transfer rate

↳ Can use raid for redundancy

SSD

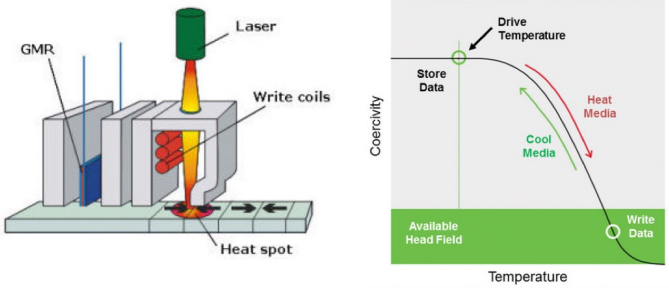
- Pros
  - No moving parts. More robust to movement.
  - Lighter and occupy less space
  - Higher Transfer rate
- Cons
  - More Costly compared to HDD
  - Finite number of Erasure cycles

Data Centre Consideration

- Power consumption
  - Data centers consumed a lot of power, which not only translate to direct cost in utility and cooling measures, but also limit its choice of location.
  - Centers are commonly found near natural cooling elements such as large natural water bodies which offer a low cost and reliable source of cooling.
- Speed
  - Beneficial for caching databases and other data affecting overall application or system performance.
- Robustness
  - Tolerance to various form of mechanical movement/interference increases reliability and reduces need and cost of maintenance.
  - Drive housing structure shock absorption requirement is also reduced.
- Heat production
  - The less heat generated the less cooling and power required in the data center.
- Size
  - Data centers will be able to store more data in less space, which increases efficiency in all areas (power, cooling, etc.)

HAMR (Heat Assisted Magnetic Recording)

→ manufactures trying to ↑ HDD

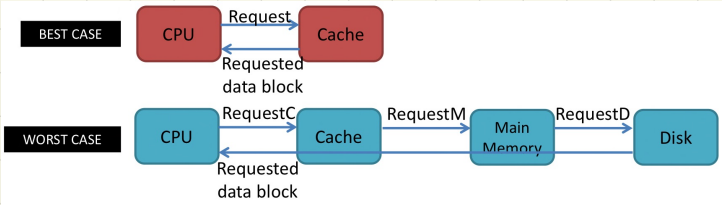


- The areal density of the HDD was stagnant for a while and manufactures had been shipping drives with 2TByte/Platter.
- But with the HAMR, the areal density is expected to increase again.
- A small laser is attached to a recording head, designed to heat a tiny spot on the disk where the data will be written. This allows a smaller bit cell to be written as either a 0 or a 1.
- Current projections are that HAMR can achieve 5 Tbpsi, enabling hard drives with capacities higher than 100 TB.

- The major problem with packing bits so closely together on conventional magnetic media is that the data bits become unstable and may flip (0 → 1 or 1 → 0).
- To make the media maintain their stability to store bits over a long period of time, the recording media needs to have a higher coercivity.
- Higher coercivity implies the media is magnetically more stable during storage, but it would also be more difficult to change the magnetic characteristics of the media when writing.
- For that, a laser is employed to heat a tiny region of several magnetic grains for a very short time (~1 ns) to a temperature high enough to lower the media's coercive field to below that of the write head's magnetic field. This is the write process.
- Immediately after the heat pulse, the region quickly cools down and the bit's magnetic orientation is frozen in place. The data is stored in the media and would be stable due to the high coercivity of the recording media.
- See the graph in the previous slide for the visual illustration.

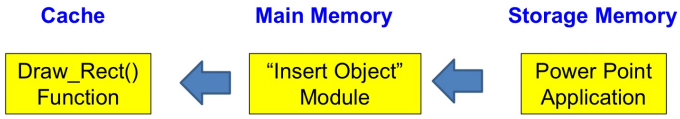
Cashe

- Issue: **CPU** speed is typically much **faster** than **external memory**.
  - CPU speed in Ghz region
  - External Memory Speed in 100s of Mhz region.
- **Need a fast memory** to act as a **buffer** between the Main memory and CPU.
- The purpose of cache memory is to speed up accesses by storing/fetching **recently used data** closer to the CPU instead of the main memory (slower access).
- Potentially able to **improve the overall system performance** drastically.



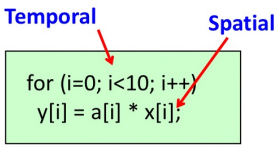
- To access a particular piece of data, the CPU first sends a request to its nearest memory, usually cache.
- If the data is not in cache, a query is then sent to the main memory.
- If the data is not in main memory, then the request goes to disk.
- Once the data is located, the **required data and a number of its nearby data elements** are fetched into cache memory simultaneously.

Cache Design - Introduction



- Example: Power Point Application
- Entire Application is stored in Storage Memory
- The user is trying to insert some object into the powerpoint slide
  - The corresponding code for object insert is transferred to Main memory for execution
- At the instance, the user is drawing a rectangle
  - The corresponding code for the Draw_Rect() function finds its way into the cache eventually.

Principle of Locality



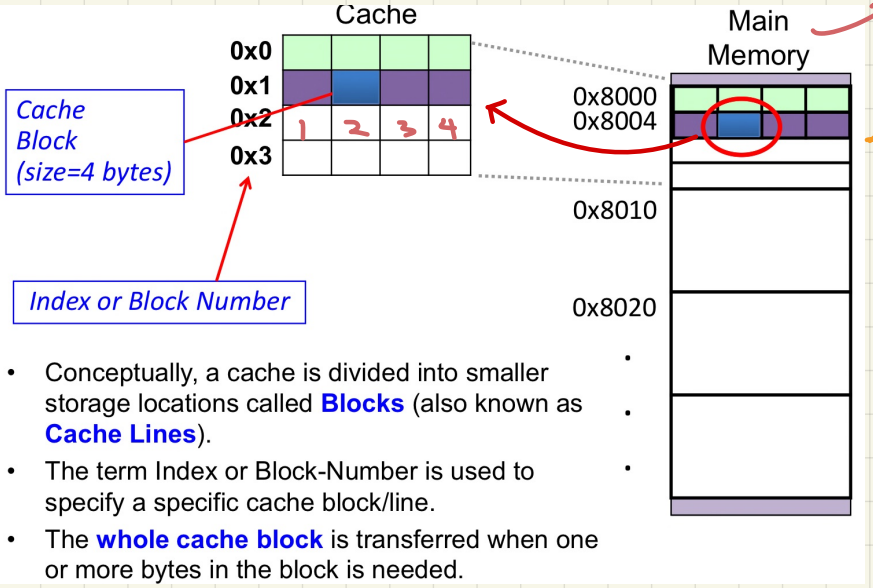
- The **principle of locality** tells us that once a byte in a program is accessed, it is likely a **nearby data element** will be needed soon.
- There are two principle of locality governing this behaviour
- Locality of **Space** (or Spatial Locality)
  - **Code/Data that is nearby each other** is likely to be accessed together
  - Transfer of data between main memory and cache is done in blocks to leverage on this behaviour
- Locality of **Time** (or Temporal Locality)
  - **Recently accessed code/data** is more likely to be accessed again
  - Used to decide which item to replace in the Cache

} Spatial Block Transfer

} Reuse of previously executed code

Cache Mapping

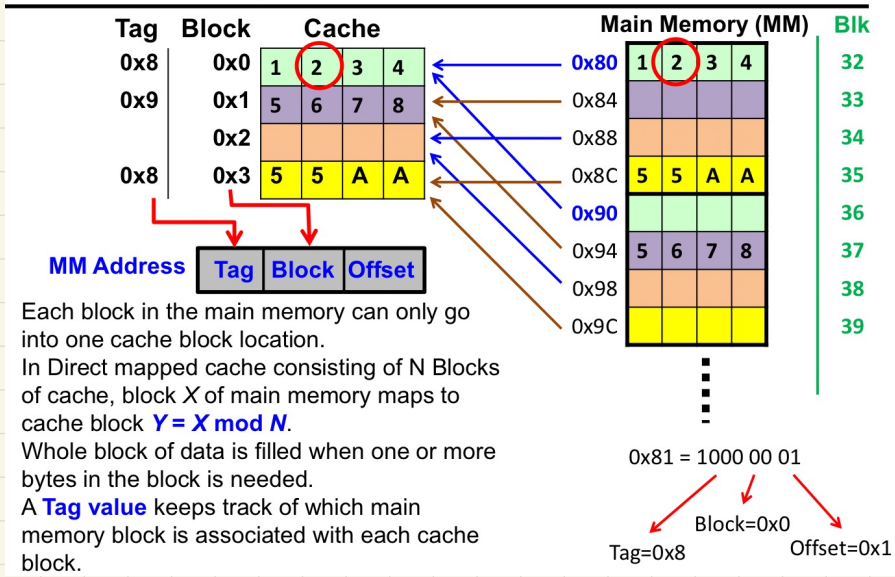
also 4 byte



- Conceptually, a cache is divided into smaller storage locations called **Blocks** (also known as **Cache Lines**).
- The term Index or Block-Number is used to specify a specific cache block/line.
- The **whole cache block** is transferred when one or more bytes in the block is needed.

Cache trashing when one cache block is always being used, leading to frequent replacement

Direct Mapped Cache



Each block in the main memory can only go into one cache block location.
In Direct mapped cache consisting of N Blocks of cache, block *X* of main memory maps to cache block ***Y = X mod N***.
Whole block of data is filled when one or more bytes in the block is needed.
A **Tag value** keeps track of which main memory block is associated with each cache block.

Tag is what cache uses to find which section data comes from

- Cache mapping
  - Allocates the data in the entire main memory into the cache which is of a much smaller size.
  - Able to uniquely identify each and every main memory location within the cache via the cache way of addressing: Block Index, Offset of the data within the block and the corresponding Tag Value of the block.
- To start doing the mapping, we need an attribute of the target information that is unique to it within the entire main memory, for that, the main memory address of the data is chosen as each target information's MM address is unique to itself.
- So the target information's MM address is partitioned into the three fields: TAG, BLOCK and OFFSET so that proper allocation to cache can be done.

- The **number of bits allocated to each field** is a result of the structure of the cache.
  - **Offset** refers to the targeted data's location offset from the start of the cache block. Something like an address within the cache block. So if the size of a cache block is 16, one would need 4 bits in the OFFSET field to address these 16 locations.
  - **Block** refers to the index of the cache block that the targeted data will be mapped to. If there are 8 cache blocks for example, then one would need 3 bits in the BLOCK field to fully represent the cache block index.
  - **Tag** bits are the left over of target data's main memory address after partitioning for Offset and Block Index. Having this information will allow the cache system to **uniquely identify** the data in the cache.

Example:

Given a system with following attribute
- **Main/System Memory size = 64KBytes**
- **Cache Size = 256Bytes**
- **Cache Block Size = 16Bytes**

Where would Data at main memory address **0x1106** be mapped to?

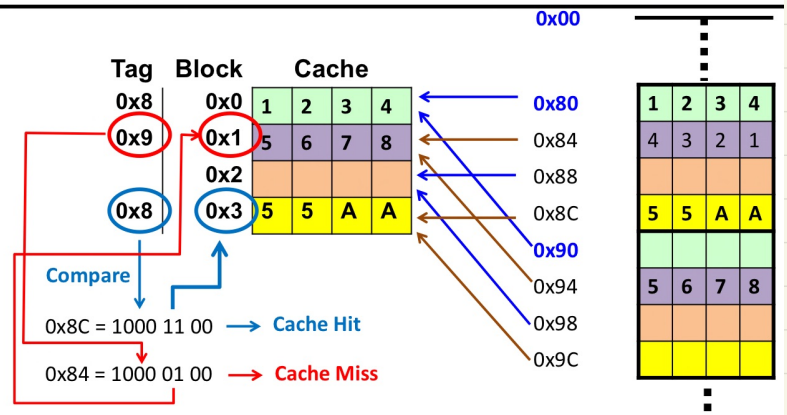
Cache Block Size of 16 Bytes ... need $\log_2(16) = 4$ bits to represent 16 value

How many Blocks? : $256 \div 16 = 16$ Blocks, need 4 bits

Main memory : $\log_2(2^{10} \times 64) = 16$ bit address

∴ Data = 0x1106 = 0001 0001 | 0000 | 0110
tag = 0x11 | BLK = 0x0 | OFFSET = 0x6

Data Retrieval Example (Direct-Mapped Cache)



Total Memory = 0xFF = 256 bytes => 8 address bits.
Cache Size = 16Bytes => 4 address bits.
Cache Block Size = 4 Bytes => 2 address bits for offset.
# of Blocks in Cache = 16/4 = 4 Blocks => 2 address bits (Block index)
# of TAG bits = 8-2-2 = 4 bits.

Steps...

1) Process Mem address
2) Find Cache block
3) If Tag is same, cache hit
4) If Tag is diff, look at MM

Cache Memory Replacement Policy

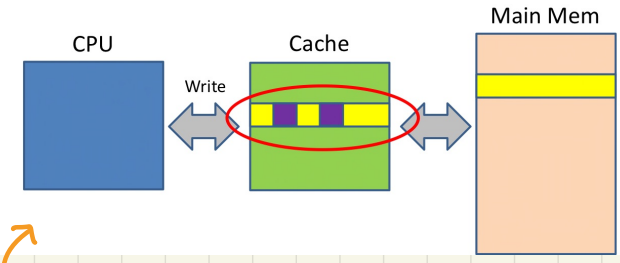
- when Cache fully occupied, need to purge

1) Least recently Used (LRU)
   → Can remove useless items
   → However, complicated as need to rem access history
2) First in, First out (FIFO)
   → Purge oldest block

- **Cache Hit rate (H)**
  - Percentage of time data found in the cache
- **Cache Miss rate**
  - Percentage of time data not found in cache.
- **Miss rate = 1 - Hit rate = (1 – H).**

Cache Write Policy

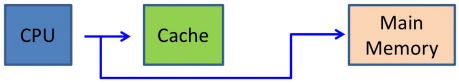
- Locations in cache that are written into by CPU is known as **Dirty Block**
- Cache replacement must take into account **dirty blocks**, those blocks that have been updated while they were in the cache.
- Dirty blocks must be written back to memory. A **write policy** determines how this will be done.
- There are two types of write policies: **write through** and **write back**.



Cache Coherency

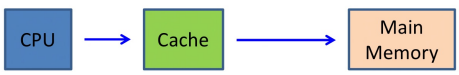
Write hit

Write through updates cache and main memory **simultaneously** on every write.



* Data Bus usage

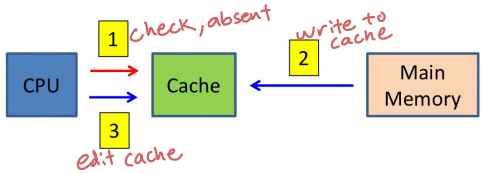
Write back (also called copyback) updates memory **only** when the block is selected for replacement.



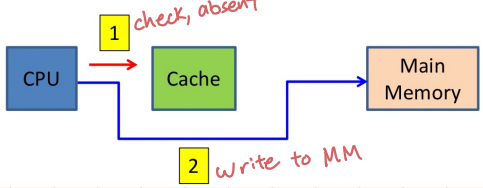
* less traffic but chance of cache coherency

Write miss

Write allocate => fetch on write. A write miss will cause the data block at the write-miss address to be **loaded to the cache**.



Write-no-allocate => A write miss will not cause the data block to be loaded to the cache. Data **Write** is done **directly** to its location in **main memory**.



Effective Access Time

Sequential Access of Cache and Main Memory, i.e. access do not overlap.

- The performance of hierarchical memory is measured by its effective access time (EAT).
- EAT is a weighted average that takes into account the hit ratio and relative access times of successive levels of memory.
- The EAT for a two-level memory is given by

$$EAT = H \times Access_C + (1-H) \times Miss\ Penalty$$

- $Access_C$ = Access times for cache
- Miss Penalty = Time need to access the data when there is Cache Miss
- If we assume that data access to Cache and Main memory do not overlap, then Miss Penalty = $Access_C + Access_{MM}$, where $Access_{MM}$ is the access times for main memory

$$EAT = H \times Access_C + (1-H) \times (Access_C + Access_{MM})$$

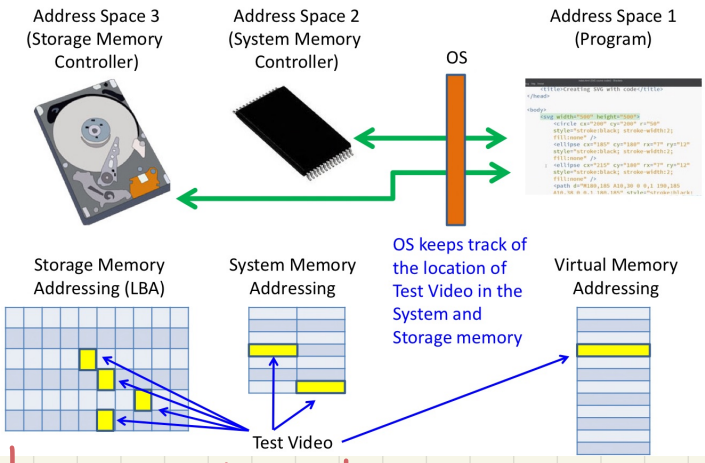
Example:

Consider a system with a **main memory access time of 200ns** supported by a **cache** having a **10ns access time** and a **hit rate of 99%.**

$EAT = 0.99 \times 10ns + 0.01 \times (10+200)ns$
$= 12ns$

Virtual Memory

Memory Address Space (Computing)

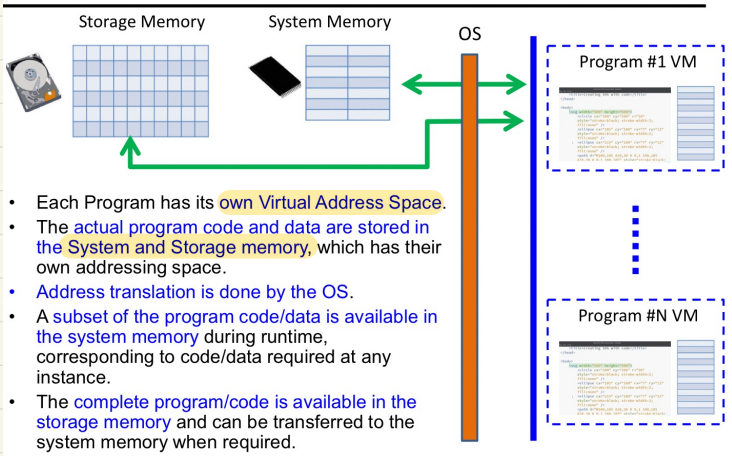


↳ OS handles translation btwn virtual & physical mem

Basic Concepts

- In computing, address space is a range of discrete addresses corresponding to some physical or logical entity, e.g. program virtual memory, physical system memory, logical layout of a HDD etc.
- Every piece of data/code in a program is assigned an address by the compiler during the program compilation.
- When executing a program, the information it requires is obtained by issuing a request to the operating system (OS) using the addresses assigned by the compiler.
- The program doesn't need to know how the required information is organised in the system memory and rely on a middle man to do the corresponding address translation in order to fetch the correct information.
- In this case, the operating system is the middle man, translating the compiler generated address to the system memory address where the required information is stored.
- Program will still work as long as it gets the code/data it requested.

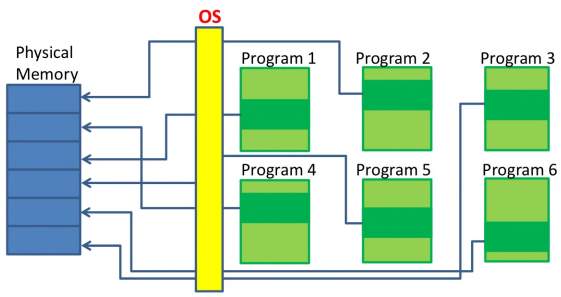
Virtual Memory (VM) Management



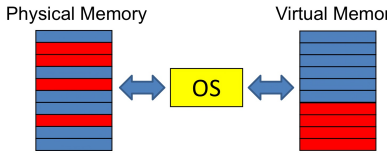
- Each Program has its own Virtual Address Space.
- The actual program code and data are stored in the System and Storage memory, which has their own addressing space.
- Address translation is done by the OS.
- A subset of the program code/data is available in the system memory during runtime, corresponding to code/data required at any instance.
- The complete program/code is available in the storage memory and can be transferred to the system memory when required.

Virtual vs Physical Memory Address Space

- The addresses used by the program is generated by the compiler and is known as the virtual address.
- The virtual address of the code/data is typically different from the Physical Memory Address that they will be resided.
- Address Translation is thus required and is done in hardware and/or software, managed by the Operating System (OS).
- Note that Physical, System and Main memory refers to the same memory for this course.
- OS is able to isolate each virtual memory space and prevent corruption between these spaces.
- Allow efficient and safe sharing among different programs within the shared physical memory.
- Allow one or more programs to run in the physical memory simultaneously even if the total size of all the programs is larger than the actual physical memory size.



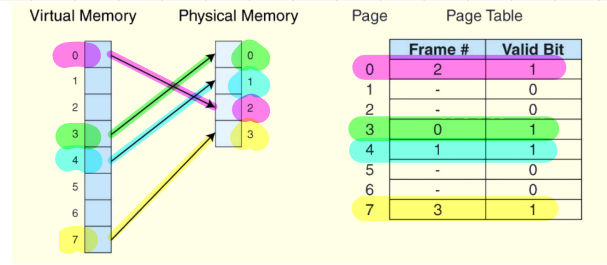
OS is able to relocate virtual memory blocks to any physical memory blocks. This allows the virtual address space seen by the application to appear to be contiguous when it is actually spread across fragmented blocks in the physical memory.



↳ Software does not need to boundary check for error

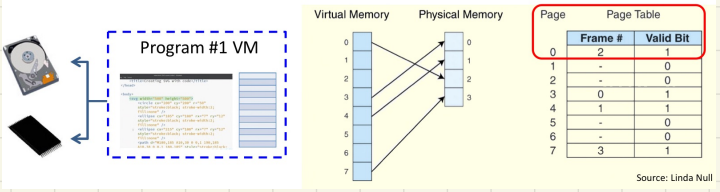
✱ Advantage: Multiple programs at once!

Paging

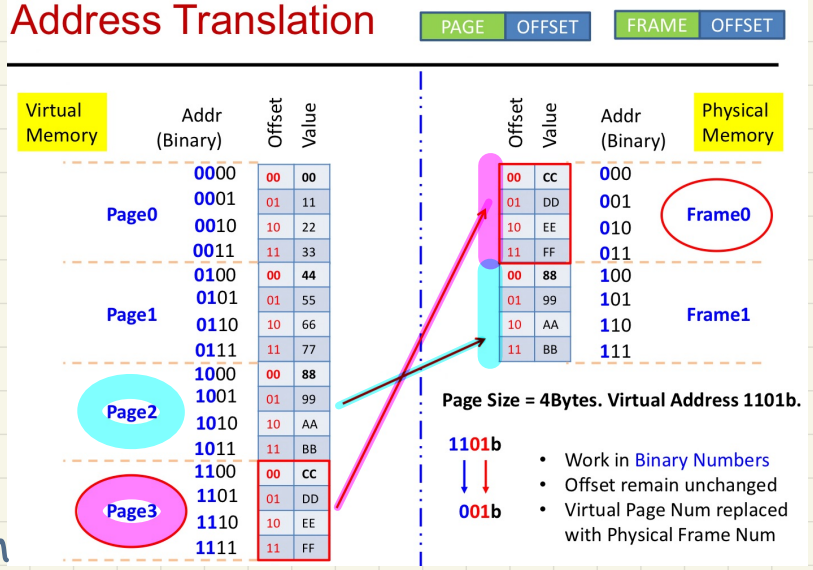


✱ Imagine Hashing!

- In a system that uses paging, the memory space is partitioned into fixed size blocks known as a Page/Frame.
- Information concerning the location of each page, whether on disk or in memory, is maintained in a data structure called a page table (shown below).
- In the Page Table below
  - Frame refers to the physical frame number in the main memory.
  - Page refers to the virtual page number used by program code.
  - Valid Bit (VB) indicates whether the Virtual Page is in the main memory (VB=1) or not (VB=0).



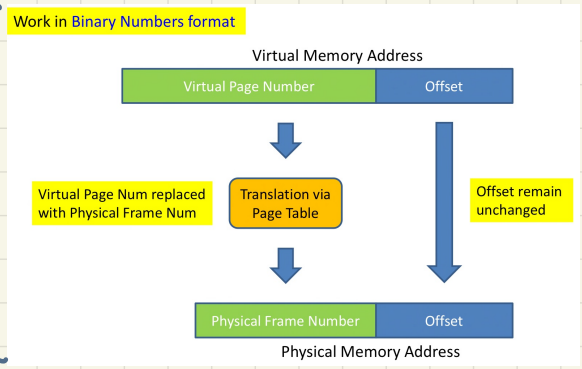
Address Translation



Steps: 1) Look at virtual address & Page size to partition the address

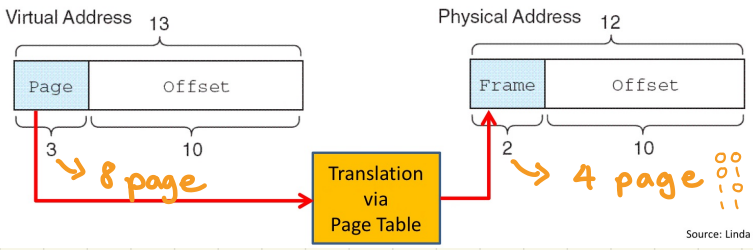
2) Look at page table to find which frame the page corr. to

3) Replace page no. with frame no.

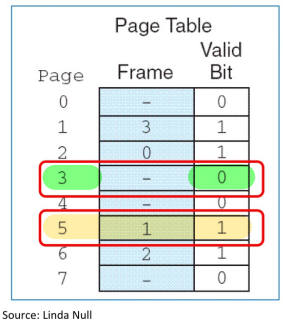


Example:

- Consider a system with a virtual address space of 8K and a physical address space of 4K, and the system uses byte addressing.
- If page size is 1KByte, we have 8 virtual pages mapping to 4 physical frames.
- Note that Virtual Page Size is always equal to Physical Frame Size.
- A virtual address has 13 bits ($8K = 2^{13}$) with 3 bits for the page field and 10 for the offset, because the page size is 1024.
- A physical memory address requires 12 bits, the first two bits for the page frame and the trailing 10 bits the offset.



- Suppose we have the page table shown below.
- What happened when CPU access virtual address location
  - 0x1553
  - 0x0FA0



- 0x1553 = 1010101010011b
  - Data resides in Virtual Page 5
  - From Page Table, Virtual Page 5 is mapped to Physical Frame 1 and Valid bit is 1.
  - Physical Address = 010101010011b = **0x0553**
- 0x0FA0 = 0111110100000b
  - Data resides in Virtual Page 3
  - From Page Table, Valid bit is 0
  - Data not in Physical Memory
  - **Page Fault**

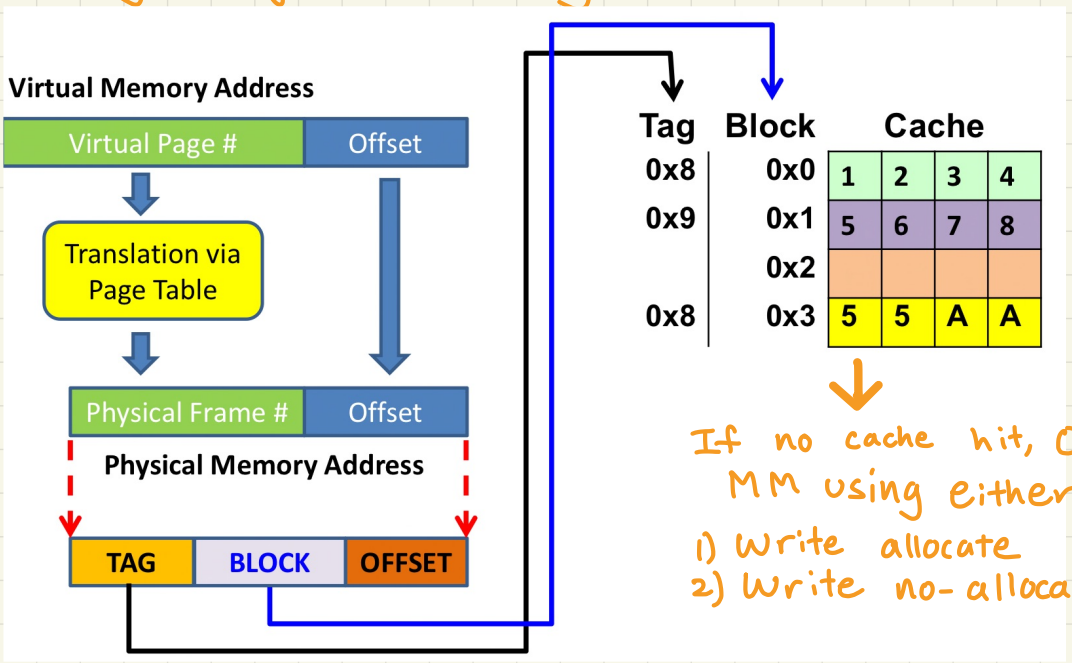
Page Fault? (Not in Page table)

→ OS needs to xfer info from storage to System memory & update page table

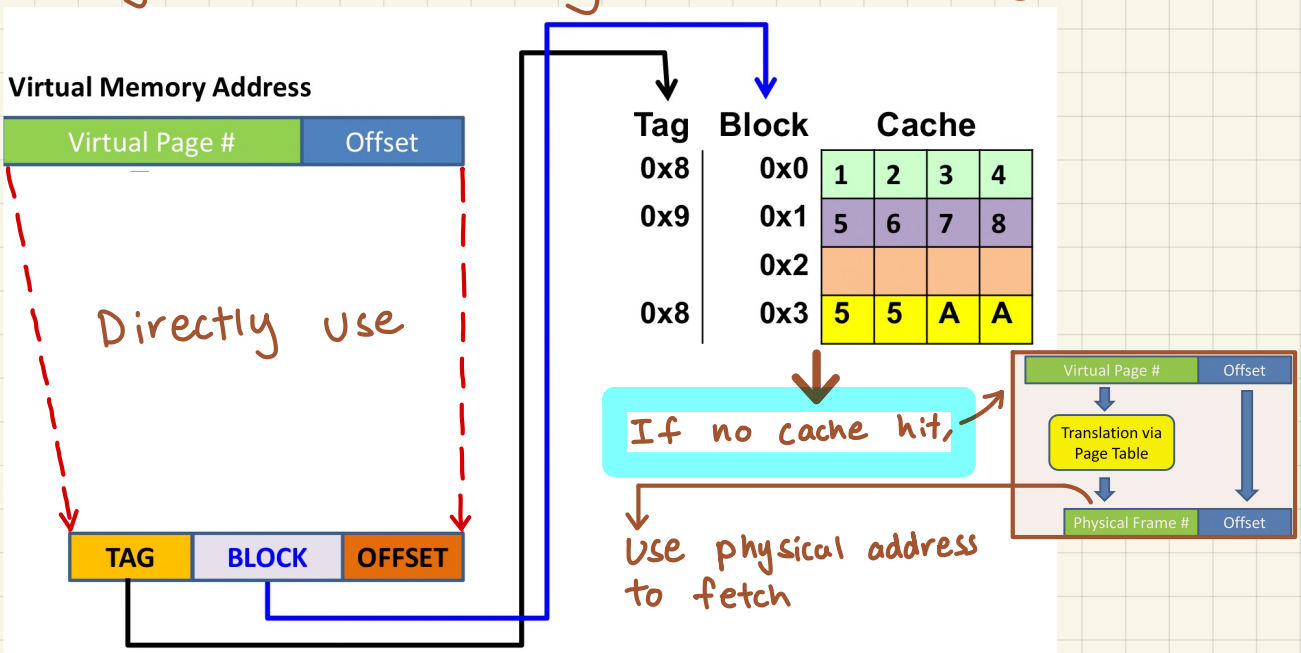
If necessary, a page is evicted from memory and is replaced by the page retrieved from storage memory, and the valid bit is set to 1.

Cache ⇄ VM

Using Physical Memory Address to page

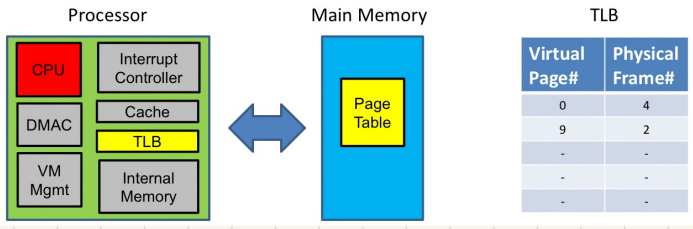


Using Virtual Memory Address to page

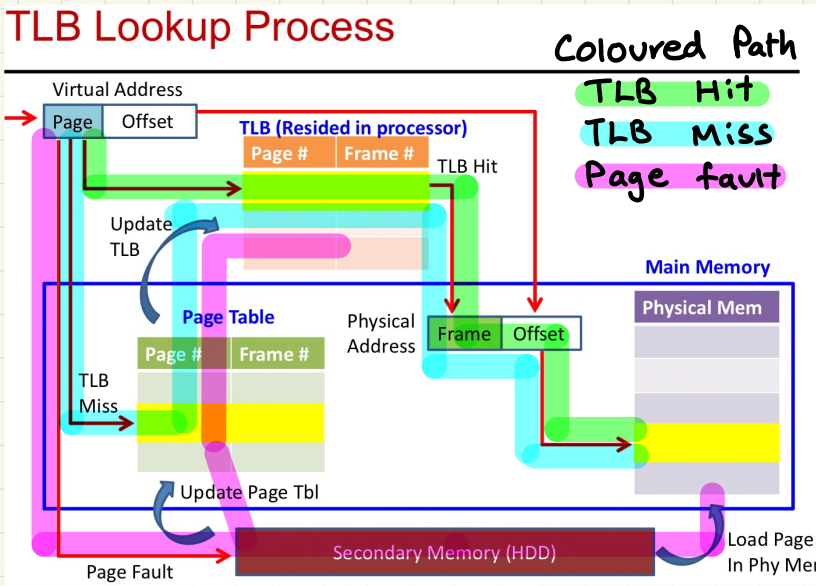


Translation Lookaside Buffer (TLB)

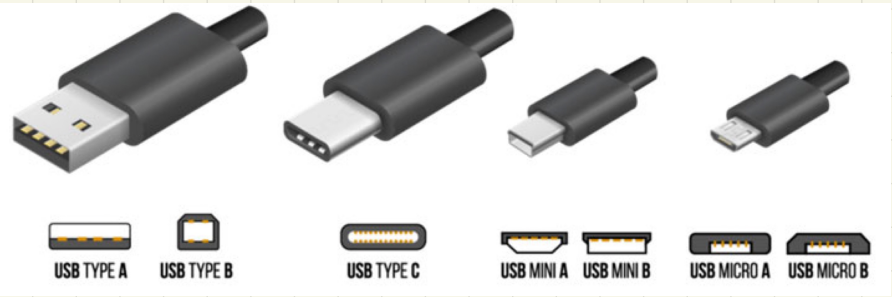
- Page Table is located in Main Memory so access is relatively slower than internal memory access.
- To speed up the page translation process, a page table cache (TLB) is used to stored a list of most recently/frequently used address translation entries in the page table.
- TLB is implement with fast memory such as SRAM and reside within the processor.
- Each TLB entry includes a Virtual Page number and its corresponding Frame Number.

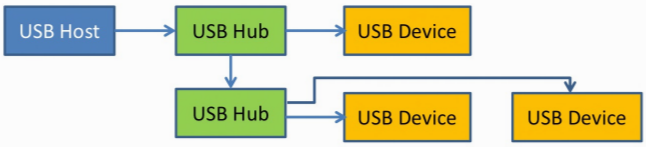


TLB Lookup Process



Computer UI





- USB used a Tiered-Star Topology, the center of the star is the USB host.
- All data transactions are initiated by the USB Host (typically resided on the computer). USB Peripherals (Mouse, Keyboard etc) can be connected directly to the Host or indirectly via the USB Hub devices.
- Host will assign address to devices connected to it to enable proper communication.
- All data transaction is with respect to the USB Host, i.e. data going into the Host is known as IN transaction and data going out of the Host is known as OUT transaction.
- Power to the devices can be supplied by the Host or Hub, known a bus-powered, or the device could have its own power source (self-powered).

USB Enumeration and Device Class

- When the USB device is first connected to the Host, it will undergo an enumeration process where the device and the host will exchange information on their capability and requirement. E.g. a USB mouse when connected with inform the Host its Vendor and Product ID, the device class it support, bandwidth required etc.
- The host cannot communicate with the device until it is properly enumerated.
- Once all the device information is transferred to the Host, the host will check if it has the required device driver to support the USB device according to the USB device class that the device belongs to.
- There are many USB Device Classes, common ones are
  - Human Interface Device (HID) used in Mouse/Keyboard.
  - Communication Device Class (CDC) used to implement Virtual COM Port e.g. Arduino Board, MSP432 Launchpad used in the lab.
  - Mass Storage Class (MSC) used to interface to external USB HDD/SSD.
  - USB Audio Class used to stream audio to USB headset/Microphone.

HDMI

High-Definition Multimedia Interface (HDMI)

- HDMI is a proprietary audio/video interface for transmitting uncompressed video data and compressed/uncompressed digital audio data from a source device, such as a display controller in a computer, to a compatible HDMI receiver such as computer monitor, digital television, or digital audio device.
- In addition to transferring audio/video data, the CEC (Consumer Electronics Control) capability allows HDMI devices to control each other when necessary and allows the user to operate multiple devices with one handheld remote control device.
- Three commonly used type of HDMI connector type are shown in the figure on the right. Starting from the left: Type D (micro), Type C (Mini) and Type A.

- The original HDMI v1.0/1.1 can only support a transfer rate of 3.96Gbps, allowing video format up to 1080p 60fps.
- The latest HDMI v2.1 can achieve 42.6Gbps, allowing 8K resolution video to be displayed.

Industrial, Scientific and Medical (ISM) RF Band

- A range of "Royalty Free" Radio Frequency Bandwidth
- Some are applicable world wide while some are restricted to certain geographical regions.
- The two commonly known world wide ISM bands are 2.4Ghz and 5.8Ghz.

WIFI

WiFi

- A family of wireless networking technologies, based on the IEEE 802.11 family of standards, commonly known as “Wireless LAN”.
- Operates in the 2.4Ghz and 5.8Ghz RF range.
- Two common topologies: Infrastructure and Adhoc.
- Infrastructure Mode
  - Uses Star topology, at the center of the network is an Access Point or Router, connected to devices at the end.
- Adhoc Mode
  - Peer to Peer connection
- Transmission Range generally between 20m to 150m, factors affecting the range include transmission frequency, transmission power and interference.
- Transfer rate of up to ~10Gbps for the latest 802.11ax (WiFi 6).

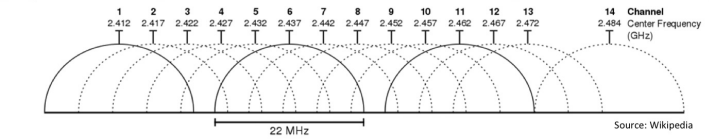
Bluetooth

- Mainly used for low data rate wireless transmission with focus on low power consumption.
- Example: bluetooth headset, smart wearables, Bluetooth mouse etc.
- Operates in 2.4Ghz range
- Transmission range up to 100m but typically kept to 10-20m to keep power consumption low. Factors affecting transmission range is similar to that of WiFi.
- Star topology
- Transfer rate in order of Kbps and Mbps.
- The Bluetooth standard defined two different Bluetooth protocols: Bluetooth Classic (BT Classic) and Bluetooth Low Energy (BLE).
- These are based on two completely different network protocols and are not compatible. But most Bluetooth Hosts today are “Dual Mode” host so is able to connect to both BT Classic and BLE devices.

Factors affecting transmission range

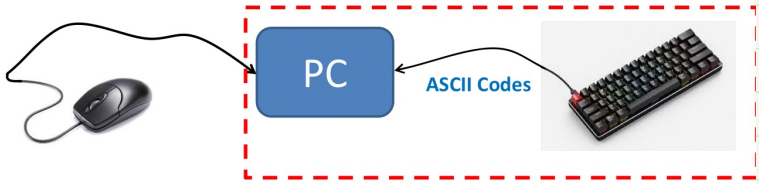
- Transmission power
  - Transmission range increase as transmission power increase.
- Transmission frequency
  - Higher frequency signal experience higher attenuation when propagating through the air or other medium.
  - All things equal, higher frequency signal has lower range than lower frequency signal.
- Interference
  - Many commonly adopted standards such as Wifi and Bluetooth works in the same 2.4Ghz ISM band. Their transmission will interfere with each other.
  - The closer the transmitter is to each other, the stronger the interference.
  - Even the micro oven in your kitchen operates in the 2.4Ghz range!

Mitigating Interference (some methods)



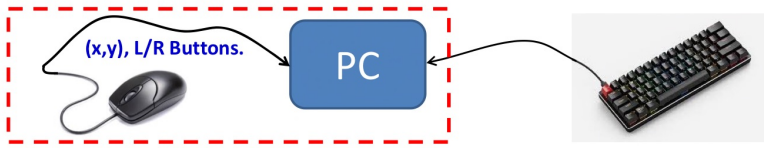
- “2.4Ghz ISM band” really consist of a band of frequencies between 2.4 and 2.5Ghz. Similarly, “5.8Ghz band” also span across a certain frequency range.
- Figure above shows the Wifi standard dividing the given frequency band into sub-bands known as channels.
- One common way to mitigate Wifi interference is to select different channels to use from your neighbours.
- Another common way is to use frequency hopping, i.e. to constantly hop from one channel to another so that transmission will eventually succeed. Bluetooth uses frequency hopping.
- There are other methodologies and techniques employed to further mitigate interference between transmitters.

Keyboard and Mouse



- **Keyboard**
- User input device for transmitting characters (alphabets, numbers, special symbols etc).
- Has a small micro-controller on board to detect the key press and sent their corresponding ASCII codes to the computer.
- Connection to PC is via wired or wireless means.
- Wired keyboard today mainly uses USB interface, keyboard will be enumerated as a HID Keyboard device.
- Wireless keyboard mainly operate in the 2.4Ghz ISM, technology could be Bluetooth or proprietary 2.4Ghz RF protocol.

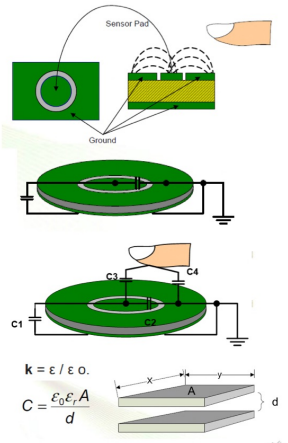
Keyboard and Mouse



- **Mouse**
- User pointing device, tracks its position by mechanical or optical means.
- Information reported are the (x,y) coordinates and the left/right mouse button.
- More information may be reported for mouse with more advanced features.
- Information reported back to PC periodically and is known as the scan/polling rate of the mouse.
- Typical mouse has a scan rate of 125Hz, gaming mouse scan rate could be up to 1000Hz or higher. Higher scan rate typically implies better response.
- Another parameter is the Dot-Per-Inch (DPI), which measures how fine the mouse could track the physical movement, higher DPI implies higher sensitivity to small physical mouse movements.
- Connection to PC utilise similar technology as Keyboard.

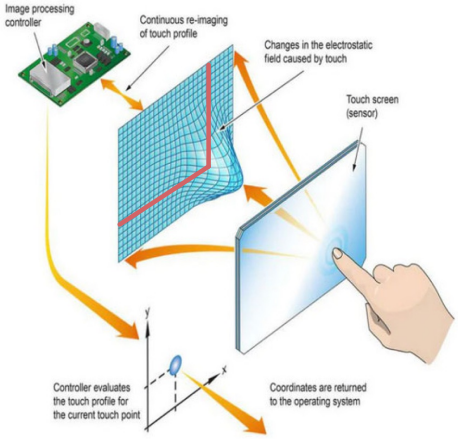
Capacitive Touch Interface

- A capacitive touch pad/button on the right is seen as a capacitor to the processor it is connected to.
- When a conductive element is present, e.g. finger, the effective capacitance of the setup increases.
- Capacitance is also affected by any dielectric e.g. gloves, plastics, liquid between the finger and the pad.
- Capacitance is directly proportional to dielectric constant (k) and air typically has a smaller dielectric constant (~1) compare to all other materials (>1).
- That's why your phone touch screen, which is typically capacitive touch based, don't work as well if you have a glove on or the screen is wet.
- Calibration is done under assumption of human finger touch.



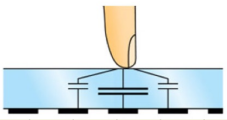
Source: Texas Instruments

Capacitive Touch Screen

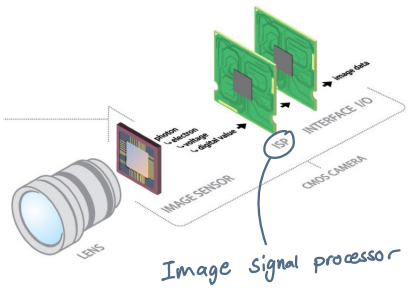


Source: Wikipedia, Arrow.

- Capacitive touch screen consist of an array of electrodes functioning as capacitive touch sensors.
- Finger touching any part of the screen will affect one or more electrodes, with the one closest having the most change to its nominal capacitance.
- A controller is used to process the information to derive the exact position of contact.



Camera



Source: Thinklucid.com, wired.com

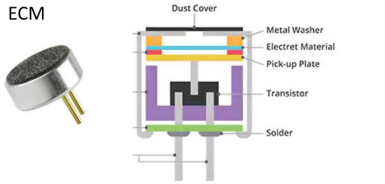
- Typical camera on phones or PC today uses CMOS camera module, it consist of the sub-modules shown in the figure above.
- Sub-modules: Lens, Image Sensor, Image Signal Processor and Interface I/O.

Camera Sub-Modules

- Lens
  - Optical lens to focus the real world images onto the image sensor.
- Image Sensor
  - Mostly CMOS based these days.
  - Array of sensors that transduce photons (light information) to electrical signals (analog).
  - Analog-to-Digital conversion of the analog electrical signal to its digital equivalent for processing in the ISP.
- Image Signal Processor (ISP)
  - Processor designed to specifically handle image processing of collected image data from the sensor.
  - Processing done include Auto Focus, Auto Exposure, Auto White Balance, image format conversion, image post-processing/compression etc.
  - Common image format are YUV (luminance and chrominance) and RGB (Red, Green, Blue).
- Interface I/O
  - Re-formatting of image data from the ISP for delivery to the host processor.

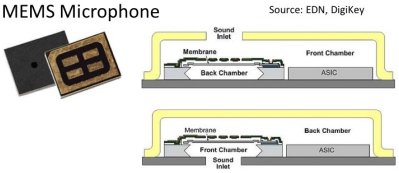
Microphones

ECM



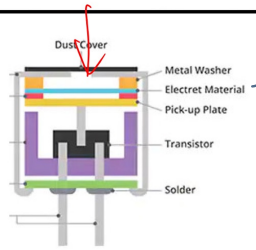
MEMS Microphone

Source: EDN, DigiKey



- Two of the most popular types of microphones are micro-electro-mechanical system (MEMS) microphones and electret condenser microphones (ECM).
- Both MEMS microphone and ECM uses the variation in capacitance when the diaphragm is displaced by the sound pressure to transduce sound wave to electrical signals.
- Difference is the in ECM, the electrical charges needed to measure the change in capacitance is provided by the charges stored on the electret.
- In MEMS Microphone case, the electrical charges needed is provided by a charge pump instead.
- The transduced signal is analog in nature but an ADC could be added to enable a digital output.

ECM Microphone

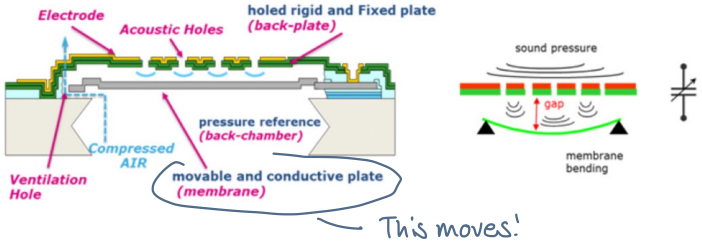


Always Charged, when sound hit, will separate from conductive plate, △ capacitance

- In an ECM, the electret diaphragm is a material with a fixed surface charge that's placed near a conductive plate
- A capacitor is created with the air gap forming the dielectric.
- Sound pressure waves moving the electret diaphragm cause the value of the capacitance to change, causing voltage across the capacitor to vary, $\Delta V = Q/\Delta C$ (Q = a fixed charge).

↳ If supply votage △, micro still can record!

MEMS Microphone

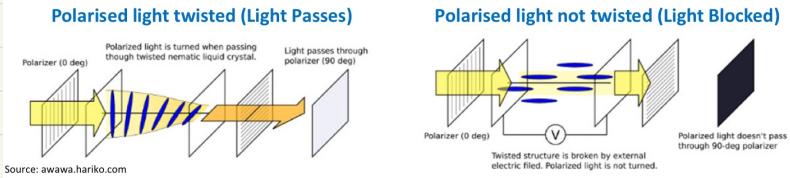


This moves!

- MEMS microphone basically is an acoustic transducer.
- Transduction principle is the coupled capacity change between a fixed plate (back-plate) and a movable plate (membrane)
- The capacitive change is caused by the sound, passing through the acoustic holes, that moves the membrane modulating the air gap comprised between the two conductive plates

↳ Less power
↳ Quality
↳ Smaller

Liquid Crystal Display (LCD) Basics



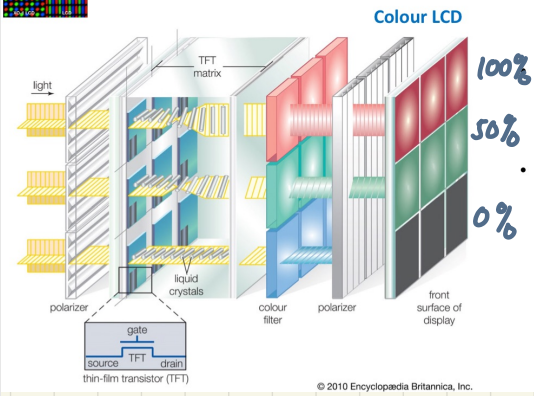
- Passive display technology, i.e. they don't emit light. Instead, they need a backlight in order for user to see the image.
- Liquid crystal is an organic substance that has both a liquid form and a crystal molecular structure. The rod-shaped molecules are able to kept their order in a particular direction although they are in liquid state.
- An electric field can be used to control the molecules orientation.
- Depending on their orientation, these molecules is able to twist the light passing through them.
- The amount of light that is able to pass through the polariser depends on its orientation with respect to the polariser.
- This result in light passing or being blocked from user point of view.

↳ Must change 90° or else blocked

Colour LCD

Source: Wikipedia

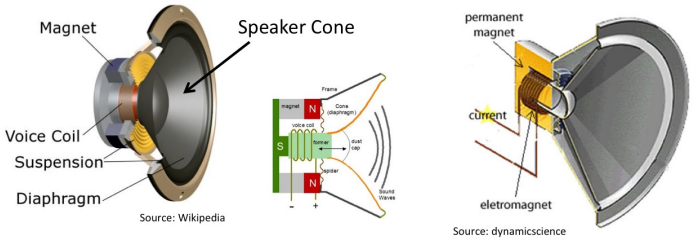
RGB Pixels Pattern on Colour LCD



© 2010 Encyclopædia Britannica, Inc.

- Process for colour LCD is similar to that discussed in previous slide.
- Each pixel is associated with three colour filters to project the RGB colours.
- A varying orientation of molecules in the liquid crystal suspension. varies the amount of light allowed to pass through to the colour filter, thereby changing the colour picture on the display screen.

Audio Speaker

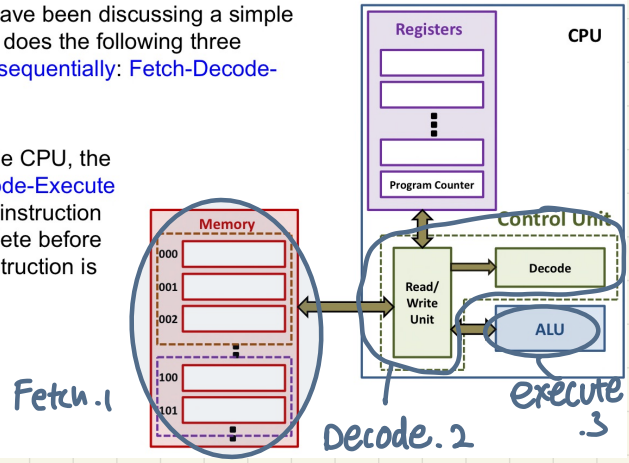


- The device transduces electrical energy to sound energy.
- The speaker cone vibrates, pushing and pulling the air to create sound waves.
- The conversion from electrical to mechanical energy occurs through an electromagnetic coil and magnet combination attached to the cone. This coil moves the speaker cone back and forth as its electromagnetic field changes with the electrical current passing through it, converting the mechanical energy to sound energy.

Processor Pipeline

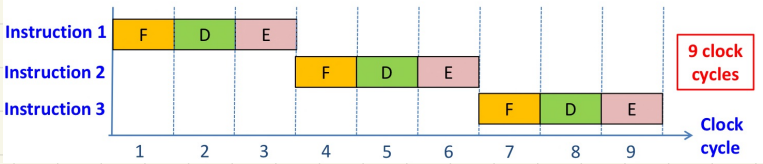
Review of a Simple CPU

- So far we have been discussing a simple CPU which does the following three operations sequentially: Fetch-Decode-Execute
- In the simple CPU, the Fetch-Decode-Execute cycle of an instruction must complete before the next instruction is fetched



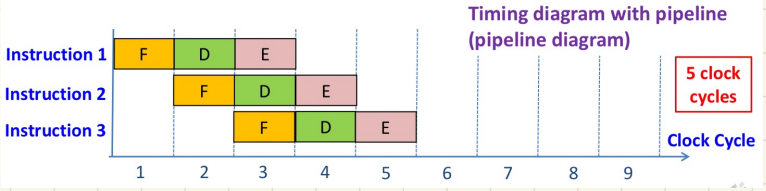
Instruction Execution – Simple CPU

- Consider the simple CPU discussed in previous slide.
- Each instruction is broken down into the following stages
  - Fetch Instruction (F),
  - Decode (D)
  - Execute (E)
- Executing each instruction is equivalent to cycling through the three stages F, D and E. If we assume that each stage takes one clock cycle, the time taken to execute 3 instruction will be 9 cycles.



Executing each stages in parparallel

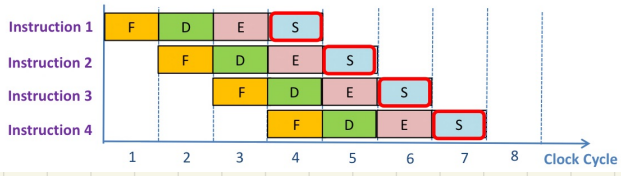
- Now, consider a processor which can execute the individual stages simultaneously.
- When instruction1 is in the Decode (D) stage, the processor is able to fetch the machine code of instruction2 from the memory.
- And when instruction3 in the Execution (E) stage, the processor is able to Decode instruction2 and Fetch the machine code of instruction3.
- Total time taken to execute the same three instructions has reduced to just 5 clock cycles.



Pipeline Efficiency

→ Like Satisfactory Pipes!
↳ Need to fill first

- The efficiency of a Pipeline architecture is maximised when the pipeline is filled
- Supposed we have a 4-stage pipeline processor
  - Instruction Fetch (F)
  - Instruction Decode (D)
  - Instruction Execution (E)
  - Data Store (S)
- It takes 4 cycles to fill the pipeline and release the first instruction from the pipeline.
- But after the pipeline is filled, it is able to release one instruction every cycle, giving the effect of 'executing' one instruction every clock cycle.

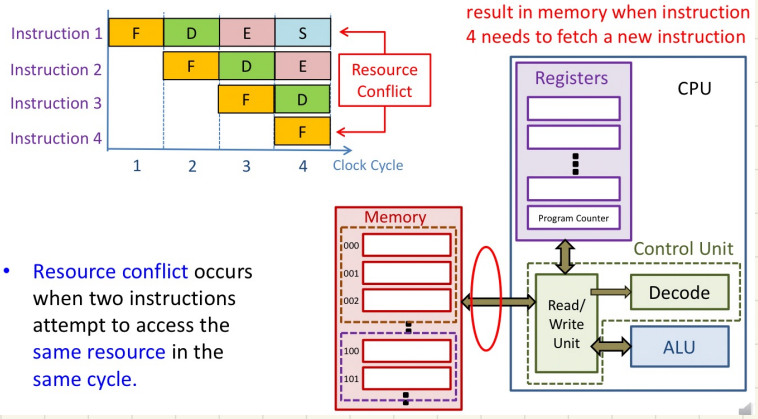


Pipeline Conflicts

- Pipeline efficiency mentioned in the previous slide will be reduced drastically if there are disruption to the pipeline.
- Events that disrupt the pipeline is known as pipeline conflicts.
- Pipeline conflicts typically cause a temporary halt to the pipeline or in some cases, the processor has to flush the instructions in the pipeline and reload with a fresh set of instructions.
- These actions result in additional time to execute a particular set of instructions.
- Since the total time taken to execute the instruction is longer, the number of instructions that can be released from the pipeline is reduced, which means the effective performance of the processor is lowered.
- In this module, we will look at three sources of pipeline conflict
  - Insufficient Resource
  - Data Dependency between instruction
  - Pipeline flushing due to Branch Instruction

Resource Conflict

- Consider a processor with 4 pipeline stages: Fetch Instruction (F), Decode (D), Execute (E), Store Result (S)



- Resource conflict occurs when two instructions attempt to access the same resource in the same cycle.

Resolving Resource Conflicts

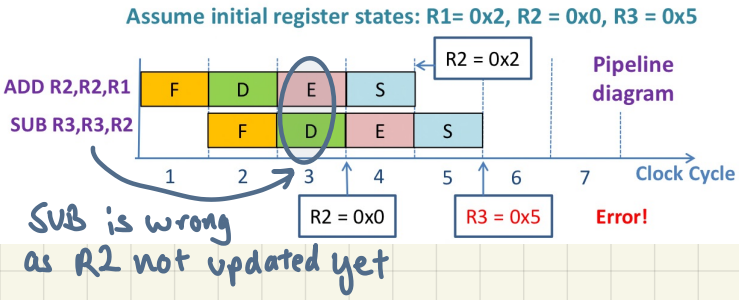
- You need sufficient resources for a pipeline processor to work efficiently
- A pipeline processor with insufficient resources to operate each pipeline stage simultaneously will result in constant halting and flushing of pipeline.
- Resultant processor performance may even be worst than that of a processor with a simple CPU architecture.
- Its common for pipeline processors to have multiple resources: internal buses (data, instruction, peripherals), control and processing units etc to allow simultaneous operations in any instance.

Use of ARM instructions in Pipeline Examples

- ARM instructions are used in the Pipeline examples but note that the following key differences
  - The pipeline structure discussed used is NOT that of the ARM pipeline
  - For simplification of analysis, ALL the ARM instructions used are assumed to occupy only one word and each pipeline stage take one clock cycle to execute.
  - The suffix 'S' feature is enabled by default, i.e. all instructions will affect the conditional flag. E.g. ADD = ADDS, MOV = MOVS.

Data Dependency Conflict

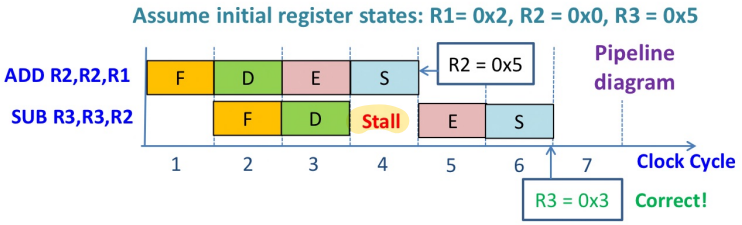
- Consider a 4-stage pipeline (FDES).
- Actual operation of each instruction e.g. ADD, SUB, is done during the Execute (E) stage.
- The resultant value of the execution is transferred to the destination during the Store (S) stage.
- In the example below, the old value (0x0) is still in R2 when SUB instruction is in Execute (E) stage, so R3 will have the wrong value (0x5) instead of the correct value (0x3).



SUB is wrong as R2 not updated yet

Resolving Data Conflict – Stall the Pipeline

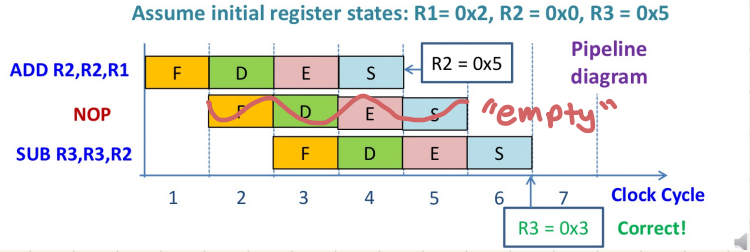
- Hardware circuitry can be used to detect data dependency between instructions
  - Compare destination identifier in the Execute Stage with source(s) ir the Decode stage.



- If data dependency is detected (i.e. R2 matches), allow ADD to continue normally, but stall the Decode stage of SUB.
- After ADD completes, SUB is allowed to resume.

OR ↓

- Compiler can analyze and insert redundant instructions to reduce data conflict
  - Data dependencies are evident in instructions during compilation
  - Compiler inserts explicit NOP (No Operation) instructions between instructions with data dependencies
- Delay ensures new value is available in register but causes total execution time to increase



Branch Instruction

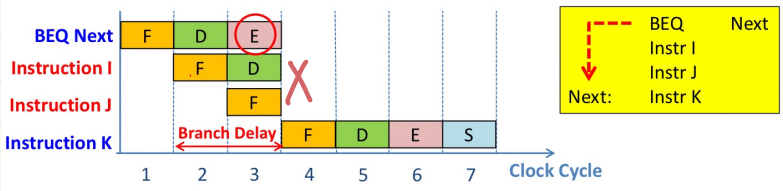
- Branch instruction usually need to perform two operations
  - Evaluate condition to determine if branch should be taken/not taken
  - If branch is taken, calculate branch target using adder in ALU
- Both operation requires an ALU to perform some computation and processing so a natural stage to do this is in the Execute (E) Stage of the pipeline.
- However, due to overlapping operations between instructions in a pipeline, the unnecessary instructions may already been introduced into the pipeline before the branch decision had been made.
- The processor would need to flush the pipeline to reload correct instructions according to the branch decision. Flushing of pipeline is equated to wastage in cycles for instruction execution.
- The number of cycles wasted/lost is known as Branch Delay.

↳ Start one branch
↳ If choose other branch, flush.

OR this

Branch Delay

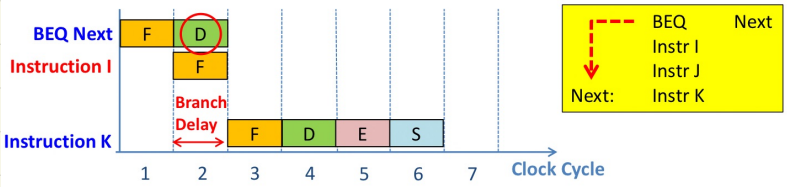
- Branch statements in pipelining can lead to Branch Delay which cause significant performance loss.



- The branch target is only known after the Execute stage, but by this time, Instructions I and J have already been fetched.
- Instructions I and J will be discarded, resulting in two-cycle branch delay.
- The two slots that are discarded is known as the Delay Slots
- Instruction I and J is known as the Delay Slot Instruction

Reducing Branch Delay

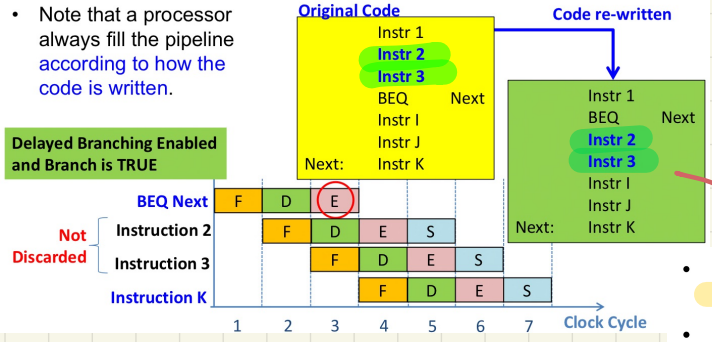
- Branch delay can be reduced by making the branch decision and calculating the branch target earlier at the Decode stage.
- An additional adder is introduced to be used in the Decode stage to enable earlier calculation of branch target.



- After the Decode stage, the branch decision and the branch target is known.
- Hence if branch is taken, only Instruction I needs to be discarded.
- Branch delay is reduced to one cycle.

Delayed Branching

- The user or compiler needs to fill the delay slots with Independent instructions.
- For example, if instruction 2 and 3 are independent instructions, they can be made to occupy the delay slots by re-writing the program code.
- Note that a processor always fill the pipeline according to how the code is written.



- User or Compiler can schedule independent instructions to be filled in the delay slots after the branch.
- If such independent instruction exists, they will always be executed, leading to zero branch delay, since no cycles is wasted.
- If an independent instruction cannot be found or if there are insufficient number of independent instructions to fill the delay slots, NOP instruction(s) should be used to populate the delay slots to preserve the correctness of the original program logics.

Dynamic Branch Prediction

- Hardware circuitry to guess outcome of a conditional branch
  - **Branch history table is implemented to store the predicted target addresses of branch instructions in the program**
- If prediction is **correct**
  - **Continue normal execution – no wasted cycles**

BEQ Next
Instr I
Instr J
Next: Instr K

- If prediction is **incorrect**
  - **Flush instructions that were incorrectly fetched – wasted cycles**
  - **Update prediction bit and target address for future use**

Address of Branch Instruction	Predicted Target address	Prediction Result (T/F)
Address of "BEQ Next"	Next	T

Connecting to the Real World

- ARM Cortex M3/M4 Processor
  - 3-stage pipeline. Instruction Fetch, Instruction Decode and Instruction Execute)
- Branch speculation.
  - When a branch instruction is encountered, the decode stage also includes a speculative instruction fetch that could lead to faster execution.
  - The processor fetches the branch destination instruction during the decode stage itself.
  - During the execute stage, the branch is resolved and it is known which instruction is to be executed next.
  - If the branch is not taken, the next sequential instruction is already available.
  - If the branch is taken, the branch instruction is made available at the same time as the decision is made.

Hexadecimal

Positional Numbering System

- Position of each numeric digit is associated with a weight.
- Each numeric value is represented through increasing powers of a **radix** (or base)
- Examples for decimal (base 10), hexadecimal (base 16) and binary (base 2) positional number notation

Base 10: $765.43_{10} = (7 \times 10^2) + (6 \times 10^1) + (5 \times 10^0) + (4 \times 10^{-1}) + (3 \times 10^{-2})$

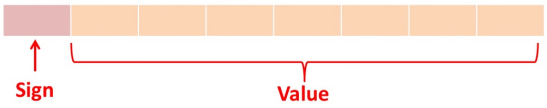
Base 16: $1234_{16} = (1 \times 16^3) + (2 \times 16^2) + (3 \times 16^1) + (4 \times 16^0) = 4660_{10}$

Base 2: $10101_2 = (1 \times 2^4) + (0 \times 2^3) + (1 \times 2^2) + (0 \times 2^1) + (1 \times 2^0) = 21_{10}$

- **Binary numbers** are the basis for all data representation in digital computer systems.

Positive and Negative Numbers

- To represent signed integers, computer systems allocate the Most Significant Bit (MSB) to indicate the sign of a number
  - MSB = 0 indicates a positive number
  - MSB = 1 indicates a negative number
- Remaining bits contain the value of the number, which can be interpreted in different ways



- Signed binary integers can be expressed using different number format, for this course, we will touch on the most commonly used format in computing
  - Two's Complement Representation

Two's Complement To Decimal Conversion

Example: What is the decimal representation for 10001110_2

Table of weights

-2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0
-128	64	32	16	8	4	2	1
1	0	0	0	1	1	1	0
-128	0	0	0	8	4	2	0

-128 + 8 + 4 + 2 = -114

The decimal representation for 10001110_2 = -114

Two's Complement Representation

- Positive numbers are represented as normal binary
- Negative numbers are created by negation
  - Invert the positive value of the number (flip the bits)
  - Add one to the inverted result (ignoring any overflow)

Example:

+106	0	1	1	0	1	0	1	0

Negate

+106	0	1	1	0	1	0	1	0
Invert	1	0	0	1	0	1	0	1
Add 1	0	0	0	0	0	0	0	1
-106	1	0	0	1	0	1	1	0

- Most Significant Bit (MSB) indicates the sign of a number (same as Signed Magnitude)
  - MSB = 0 indicates a positive number
  - MSB = 1 indicates a negative number

Detecting Overflow in Two's Complement Numbers

- Overflow can be easily detected by checking the Most Significant Bit (MSB) of the operands and result
- Conditions for overflow
  - In addition (Result = A + B)
    - If MSB(A) = MSB(B), and MSB(Result) ≠ MSB(A)
  - In subtraction (Result = A – B)
    - If MSB(A) ≠ MSB(B) and MSB(Result) ≠ MSB(A)

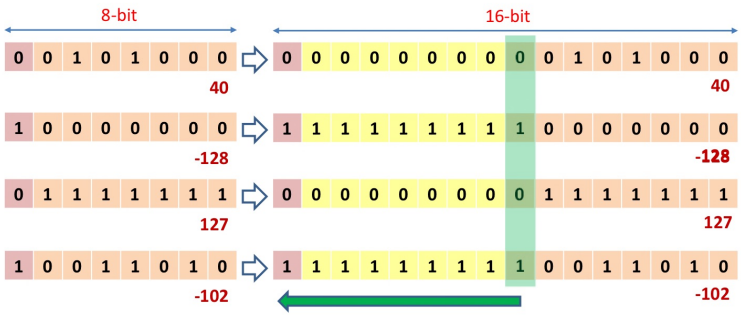
Over
Over
over
over

Operation	Conditions		Result
A + B	A > 0	B > 0	< 0
A + B	A < 0	B < 0	> 0
A – B	A > 0	B < 0	< 0
A – B	A < 0	B > 0	> 0

V=1

Sign Extension

- In two's complement, sign extension is needed to convert a smaller size operand to a larger size operand



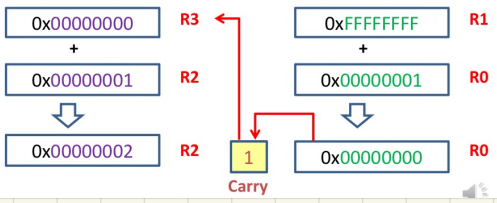
- Sign extension simply copies the sign bit (MSB) into the higher order bits

Multi-Precision Arithmetic

- How can we add operands that are larger than 32-bits (e.g. 64 bit operands) if we have only a single 32-bit ALU?
- The solution is to reuse the 32-bit adder for multi-precision addition
- Multi-precision arithmetic involves the computation of numbers whose precision is larger than what is supported by the maximum size of the processor register (Single-Precision)

```
  00000000 FFFFFFFF
+ 00000001 00000001
  00000002 00000000
```

```
ADD  R0,R0,R1 ; add lower word with carry out
ADC  R2,R2,R3 ; add upper word with carry in
```



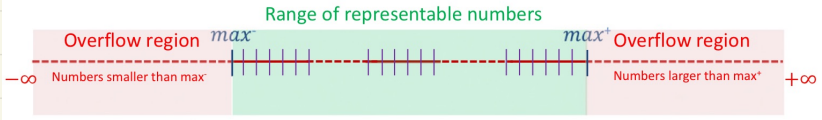
1) lower order first

2) carry bit + Next one

ADC ☆

Range and Precision

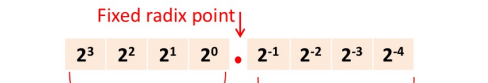
- Range: Interval between smallest (max^-) and largest (max^+) representable number
  - Example: Range of two's complement is $-(2^{(N-1)})$ to $(2^{(N-1)}-1)$
  - Each tick mark is a representable number in the range
- Precision: Amount of information used to represent each number
  - Example: 1.666 has higher precision than 1.67
  - The number of tick marks provides an indication of precision



Fixed-Point Representation

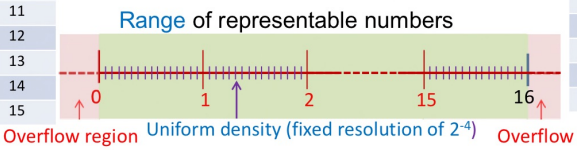
- Fixed-point format can represent integer and/or fractional values

Unsigned Integer	
0000	0
0001	1
0010	2
0011	3
0100	4
0101	5
0110	6
0111	7
1000	8
1001	9
1010	10
1011	11
1100	12
1101	13
1110	14
1111	15



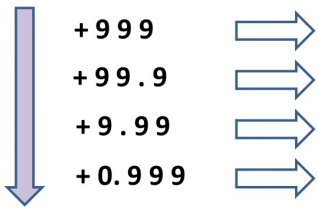
- Limitations
  - Precision is limited by the range of integer values
  - Bits are not fully utilised for certain values (e.g. 15.5 or 1111.1000_2)

Fractional Values	
0000	0.0000
0001	0.0625
0010	0.1250
0011	0.1875
0100	0.2500
0101	0.3125
0110	0.3750
0111	0.4375
1000	0.5000
1001	0.5625
1010	0.6250
1011	0.6875
1100	0.7500
1101	0.8125
1110	0.8750
1111	0.9375



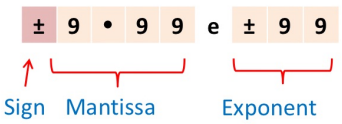
Floating Point Representation

Precision increases for smaller numbers



+999	$+9.99 \times 10^{+02}$
+99.9	$+9.99 \times 10^{+01}$
+9.99	$+9.99 \times 10^{+00}$
+0.999	$+9.99 \times 10^{-01}$

Simple decimal (Radix = 10) floating point format



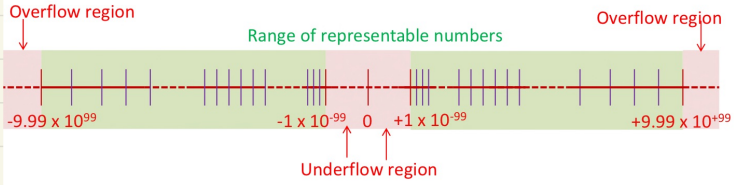
- Three main fields needed for floating-point representation:
  - Sign – denote positive/negative number
  - Mantissa – base value
  - Exponent – specifies position of radix point

Underflow

- Normalisation results in underflow regions where values close to zero cannot be represented

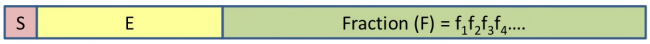
Smallest positive normalised number + 1 • 0 0 e - 9 9

Smallest negative normalised number - 1 • 0 0 e - 9 9



- Underflow occurs when a value is too small to be represented
- Floating-point overflow and underflow can cause programs to crash if not handled properly.

IEEE 754 Normalised Numbers



$$(-1)^S \times (1.F)_2 \times 2^{E - Bias}$$

- Sign bit
  - S = 0 (positive); S = 1 (negative)
- Exponent
  - Biased representation (00000001 to 11111110) 254
  - Value of exponent = E - **Bias**
  - **Bias** = **127** (Single Precision) and **1023** (Double Precision)
- Fraction
  - Assumes hidden **1.** (not stored) for normalised numbers
  - Value of normalised floating point number is:
    $(-1)^S \times (1 + f_1x2^{-1} + f_2x2^{-2} + f_3x2^{-3} + f_4x2^{-4} + ...)_2 \times 2^{E-Bias}$

Converting Single Precision To Decimal

- Find the decimal value of these single precision number:

0 1 0 1 1 0 0 1 0 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

Sign = 0 (positive)

Exponent = 10110010_2 = 178; E – Bias = 178 – 127 = 51

1 + Fraction = $(1.111)_2 = 1 + 2^{-1} + 2^{-2} + 2^{-3} = 1.875$

Value in decimal = $+1.875 \times 2^{51}$

1 0 0 0 0 1 1 0 0 0 1 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

Sign = 1 (negative)

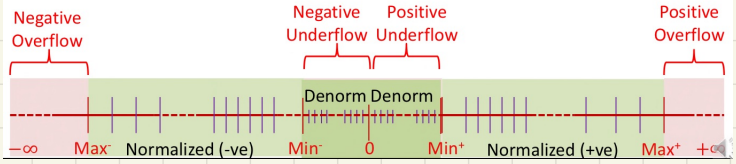
Exponent = 00001100_2 = 12; E – Bias = 12 – 127 = -115

1 + Fraction = $(1.0101)_2 = 1 + 2^{-2} + 2^{-4} = 1.3125$

Value in decimal = -1.3125×2^{-115}

IEEE 754 Encoding

Mode	Sign	Exponent	Fraction
Normalized	1 / 0	00000001 to 11111110	Anything
Denormalized	1 / 0	00000000	Non zero
Zero	1 / 0	00000000	0000 ... 0000
Infinity	1 / 0	11111111	0000 ... 0000
Not a Number (NaN)	1 / 0	11111111	Non zero



Effects of Rounding

- Rounding refers to removing of LSB(s) so that the result can fit into the representable bits.
- The limits imposed in the width of the representable bits could be from the registers width, data type etc.
- Rounding can be round-up, round-down or round to nearest representable number.
- As the rounded number is an approximation of the raw result, a certain amount of rounding error is incurred.
- Below an example of rounding off a floating point number to 2 decimal points, incurring an error of $0.004*10^1$

1 • 6 8 6 e + 0 1 ⟹ 1 • 6 9 e + 0 1

- Common to have intermediate register with width larger than regular data registers to allow intermediate processing to be done at higher precision and thus reducing the amount of rounding error.

Handing numbers with different magnitudes

- Suppose we want to compute the following :

$1.23x10^3 + 1.00x10^0 +$
$1.00x10^0 + 1.00x10^0 +$
$1.00x10^0 + 1.00x10^0$

```
  1 • 2 3 0 e + 0 3
+ 0 • 0 0 1 e + 0 3   Align exponent
  1 • 2 3 0 e + 0 3   Rounding
+ 0 • 0 0 1 e + 0 3   Align exponent
  1 • 2 3 0 e + 0 3   Rounding
        .
        .
  1 • 2 3 0 e + 0 3   Rounding
```

Handing numbers with different magnitudes

Example:
$1.00x10^0 + 1.00x10^0 +$
$1.00x10^0 + 1.00x10^0 +$
$1.00x10^0 + 1.23x10^3$

```
  1 • 0 0 0 e + 0 0
+ 1 • 0 0 0 e + 0 0
  2 • 0 0 0 e + 0 0
+ 1 • 0 0 0 e + 0 0
  3 • 0 0 0 e + 0 0
+ 1 • 0 0 0 e + 0 0
  4 • 0 0 0 e + 0 0
+ 1 • 0 0 0 e + 0 0
  0 • 0 0 5 e + 0 3   Align
+ 1 • 2 3 0 e + 0 3
  1 • 2 4 0 e + 0 3   Rounding
```

- The order of evaluation can affect accuracy of result
  - Add/subtract operands with similar size of magnitude first.

Add smaller tgt first !

Maximising Accuracy during computation

- Two issues: overflow and precision.
- From previous slides, addition, subtraction and multiplication may potentially lead to result overflowing the range of the representable number used.
- Division will lead to loss in precision as bits are loss due to truncation of LSB(s).
- Some rule of thumb
  - Accumulate/subtract numbers with small magnitude first to allow their magnitude to be comparable to big magnitude numbers.
  - Take note of the range of the number system used, which, depending on the usage scenario, can be a factor of the data type, number format, register/memory width etc.
  - Apply threshold to check for overflow if possible.
  - If no overflow checks are done, take note of the number/value of accumulation/multiplication that can be done without triggering overflow.
  - To preserve as much precision as possible, always do division last as far as possible.

→ arith right shift !