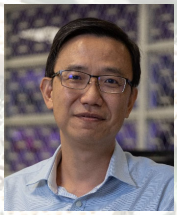
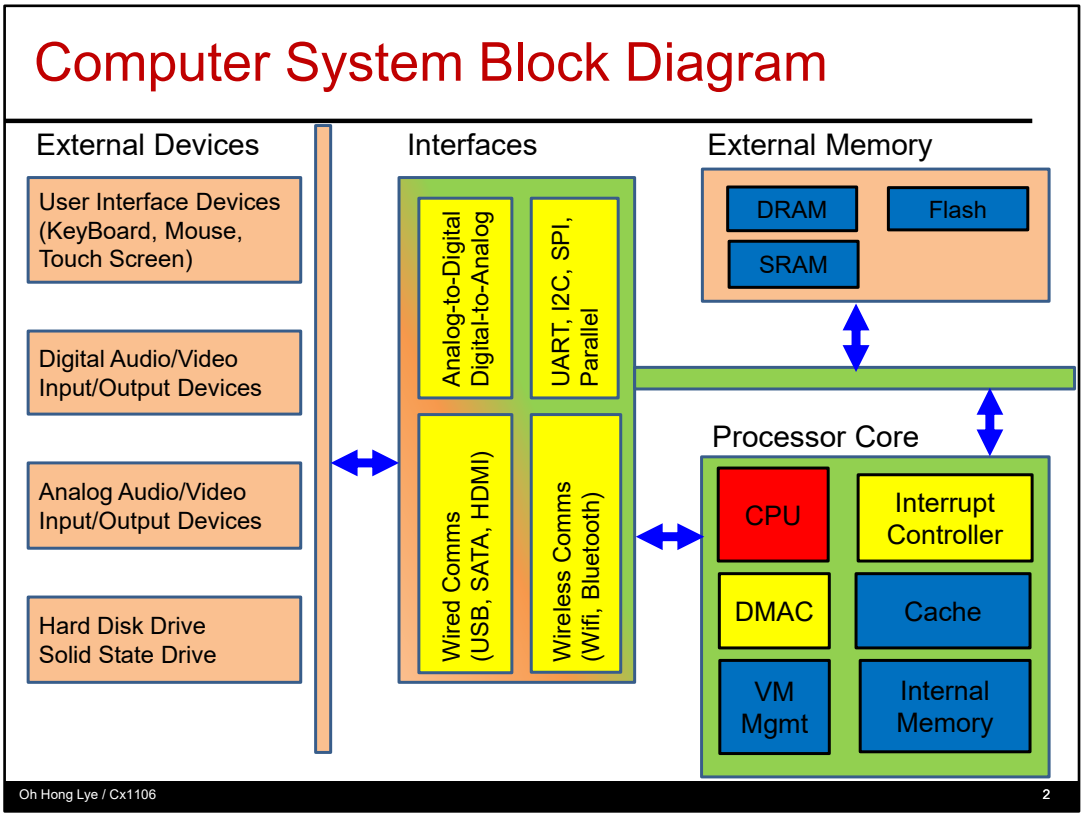
Cx1106
Computer Organization and Architecture

Computer Systems Overview



Oh Hong Lye
Senior Lecturer
School of Computer Science and Engineering, Nanyang Technological University.
Email: hloh@ntu.edu.sg

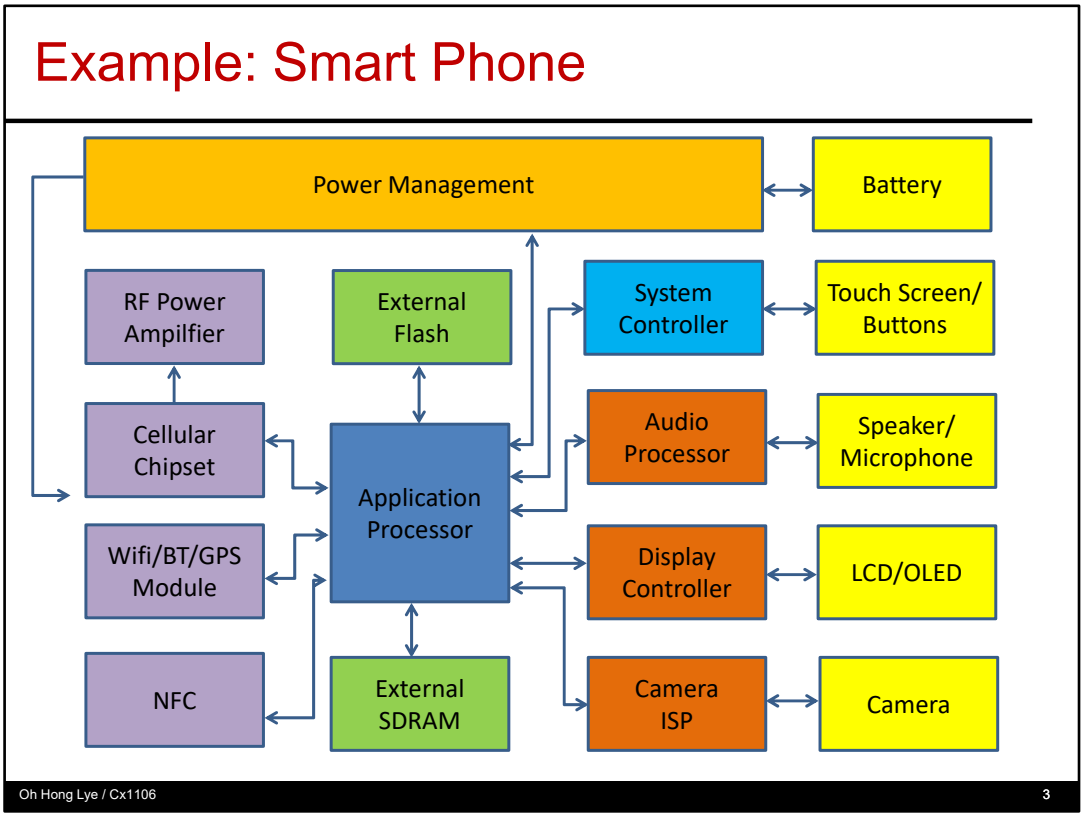
- Hi, Welcome to the second half of Cx1106 course.
- My name is Hong Lye, I'm your lecturer for this half of the course.
- If you have any questions after going through the slides and notes, please feel free to drop me an email at hloh@ntu.edu.sg



- In the first half of this course, you have been focusing mainly on the processor of a computer system.
- Now a computer system doesn't just consist of the processor alone, there are many other modules that formed up the system.
- There is the External Memory, which could be made of DRAM, SRAM or Flash memory.
    - This is where you store the code and data of your OS and applications.
- There is also the modules that enable the processor to interface to the external world.
    - It consist of external devices such as
        - Keyboard, Mouse and Touch Screen
        - Audio and Video devices
        - Mass storage such as the HDD and SSD.
    - These devices are connected to the processor via different interfaces
        - You have interface such as USB, ADC, UART, Wifi etc.
        - These functions could exist internal to the processor and is known as a peripheral, or it could exist as a separate module that is external to the processor.
- All these modules and devices are connected to the main processor via some sort of interfaces. Computer interface is a sub-topic which we will deal with in more details later.
- And the processor doesn't just consist of the CPU and internal memory as well, there are other modules such as
    - Direct Memory Controller, Interrupt controller, Cache, Virtual memory

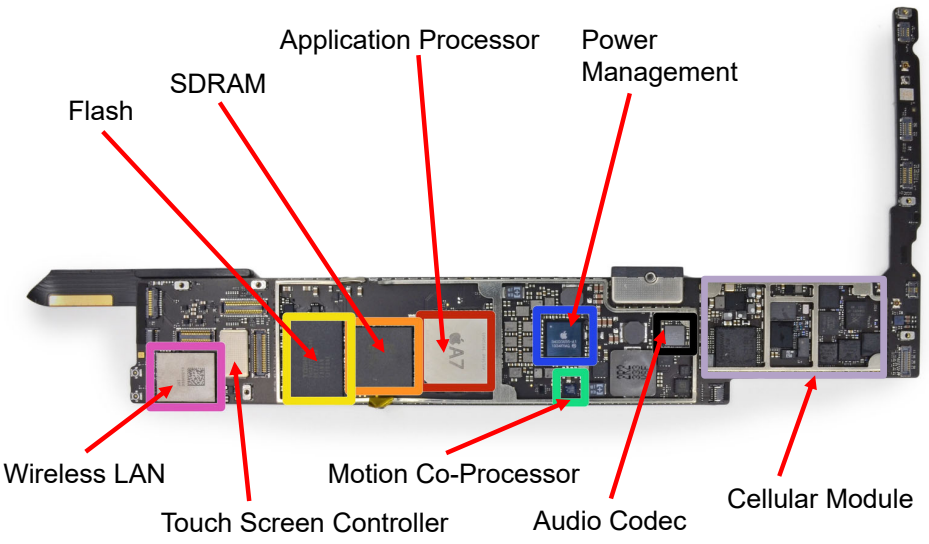
Management modules

- Essentially, we will be dealing with everything that can be found a typical computer system, topics that are beyond the CPU architecture that you learn during the first half.
- Don’t worry, we are only touching on the basics for most topics, you will be re-visiting a number of the topics again in your senior years.



- This slides shows the system block diagram of a smartphone, which is a very good example of a computer system.
- In the middle you have the application processor with its external memory
- There is a system controller which takes care of the touch screen and user inputs
- A few modules that deals with the audio, video and camera functions of the phone and the corresponding controllers for these modules.
- Over on the left side, these are the modules that is responsible for the data communication of the computer system, you have the Cellular, Wifi and NFC listed here.
- And lastly, you have the Power management module that manage the power rails and power source i.e. the battery of the system.
- If you ponder for a while, you will probably realise that modules such as the power management, wifi, camera ISP etc are actually quite complicated in design and probably would need their own processor to manage its function. You are right in that there is actually a processor in many of these modules.
- I did a count before and in a typical smart phone such as the one you see in this slide, there is easily 8-10 processors in there.

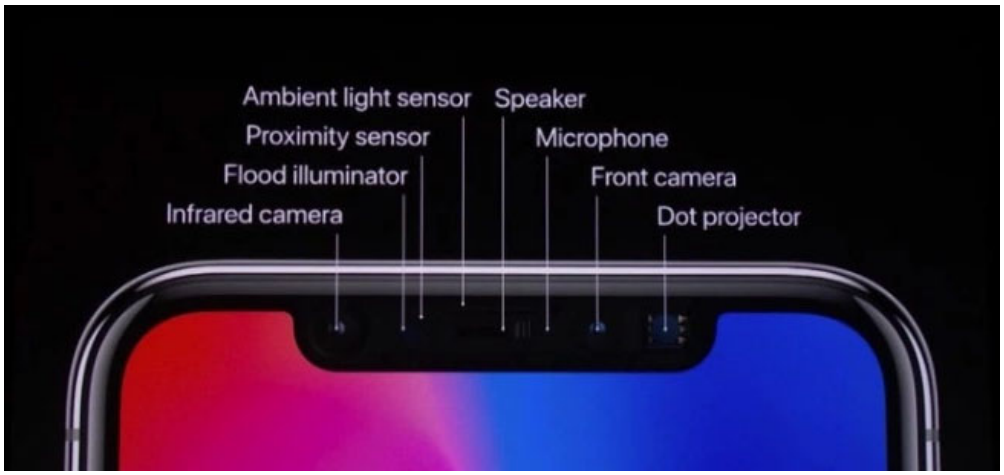
iPad Air Main PCB



Source: https://www.ifixit.com/Teardown/iPad+Air+LTE+Teardown/18907

- Translating the system block diagram to an actual Printed Circuit Board, PCB in short, of a phone.
- What you see in this slide is the main PCB of a iPAD AIR.
- This is extracted from a website know as ifixit.com, you can visit the website to see other product teardowns if you are interested.
- The photo shows you the various modules that we mention in the previous slide, not very interesting unfortunately because they are mostly black in colour.
- Sorry to say that most IC chips will be in these colours though.
- Anyway, they are mostly hidden behind product casing so I guess no one will mind.

iPhone front facing peripherals



- Smart phone these day uses many different types of sensors to enable many desired features.
- This slide here shows you a list of sensors you can find on a iphone, and this is not the latest iphone.
    - You have a proximity sensor that sense whether you face is near the phone so the screen will be blanked off
    - The ambient light sensor that would auto tune the display brightness based on the ambient lighting.
    - The Dot projector that projects thousands on IR dots in order to profile the face, and this is the basis of the FaceID feature that Apple use.
- Now the processor needs to interface to all these sensors and the various modules that we mentioned in the previous slides.
- As such, computer interface is a very important topic which we be will going into shortly under the Signal-Chain Sub-system.

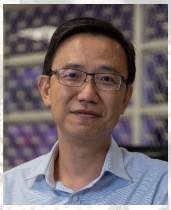
Sub-Systems in a Computer System

- 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)

- I've divided the computer into a few sub-system
- There is the processor core which you are very familiar with by now
- The signal sub system deals with interfacing the computer to the real world, and a main part of it is the computer interface topic that I mention earlier.
  - This consist of the Analog-to-Digital Convertor, ADC in short, which brings the real world signal into the computer
  - the Digital-to-Analog Converter, DAC in short, which push the information out from the computer to the real world.
  - The various digital interfaces that the processor use to communicate with various modules
  - And modules that enable different type of transfer mechanism between modules.
- You have the memory sub-system where we will touch on the different memory types and how the processor manage the code and data in the computer using cache and virtual memory management.
- And lastly, I've grouped the more complicated interface and modules in a separate sub-system called the comms and UI.
  - They actually also deal with interfacing but the mechanism behind these interfaces are more complicated so we will only be scratching the basics in this course. That's the main reason why I put them in a separate group.

Cx1106
Computer Organization and Architecture

Signal Chain Sub-System



Oh Hong Lye
Senior Lecturer
School of Computer Science and Engineering, Nanyang Technological University.
Email: hloh@ntu.edu.sg

- We will touch on the Signal Chain Sub System in the next few videos.
- I've divided the video into smaller chunks so you can have some rest in between. But some of the topics may take a longer time, in which case please feel free to exercise you right to press the pause or fast forward button.

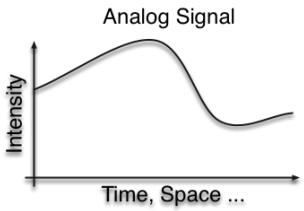
Signal Chain Sub-System

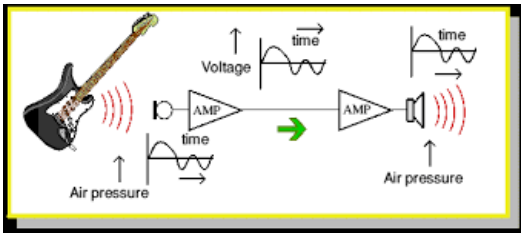
- Signal Chain Sub-System
  - ADC-DAC
  - Parallel and Serial (UART/SPI) Digital Interfaces
  - Interrupt Controller
  - Direct Memory Access Controller

- The signal chain sub-system deals with the transfer of signals between the real-world and the processor.
- The analog real world signals, passes through the ADC to the processor, which operates in the digital domain, and gets out to the real world again via the DAC.
- There are many sub-topics in the signal chain, we are going to focus on only a few of them.
  - The ADC and DAC process that deals with conversion between analog and digital domain
  - The digital interfaces between digital modules
  - And the two main types of data transfer mechanism: interrupt and DMA.

- This section is about how analog real word signals are passed into the digital processor via the analog to digital converter, and how the digital output of the processor are subsequently passed back to the real world via the digital to analog converter.

Analog and Digital Signal

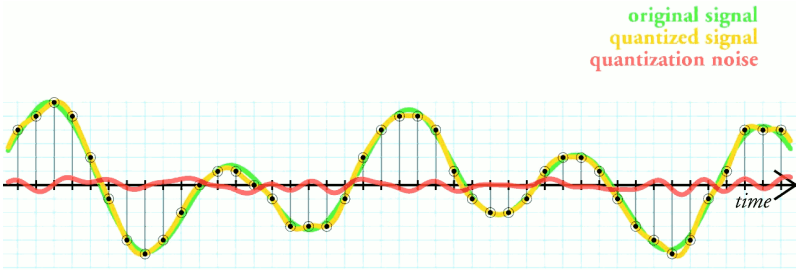




- Real world signals are Analog in nature.
  - Examples: Sound, light, heat, pressure etc
  - In the past, most processing are done in analog domain.
  - With the introduction digital processors and decreasing cost to build them, digital processing became increasingly popular as it offer more flexibility in implementation and are more tolerant to noise and component aging.
  - Analog signal has continuous voltage level and are continuous in time domain.

- Before we go into detail discussion of the various interfaces, let take some time look at the types of signal we need to deal with and their characteristics.
- All real world signals are actually analog, the sound we hear, the light we see, the wind we feel and the heat we sense.
- By analog, we mean that the signal magnitude is continuous. It is also continuous in time domain.
- So on a graph the intensity vs Time, an analog signal is represented by a continuous solid line.
- During the pre-digital processors days, analog signals are processed as it is in the analog domain.
  - For example, noise cancellation, audio equalization, TV brightness, audio loudness etc.
- With the enhancement in semiconductor technology in the last 40 years, digital processors become sufficiently economical to be deployed in large scale and overtook analog processing as the platform of choice for signal processing.
- Digital processing has the advantage of being more tolerant to noise interference and component aging.
- It also offer more flexibility in implementation as software can be easily modified to suit different scenarios.

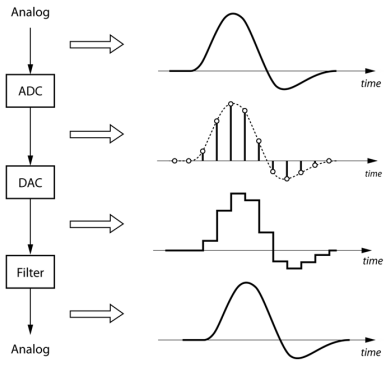
Digital Signal



- 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.

- The graph in this slide shows an analog sine wave being digitized.
- Digitization is also called analog to digital conversion. Typically done by passing the analog signal through an Analog-to-Digital Converter (ADC).
- What the ADC does is to sample the analog input at fixed interval, know as the sampling interval.
    - The value of the analog signal at each sampling instance is recorded and assigned to the closest discrete level that the particular digital system allows. More on that in the next slide.
- So these black points you see in the graph correspond to the digital equivalent of the analog sine wave.
- These digital data can be used subsequently to reconstruct the sine wave. The process is call digital-to-analog conversion and is done via a Digital-to-Analog Converter (DAC).

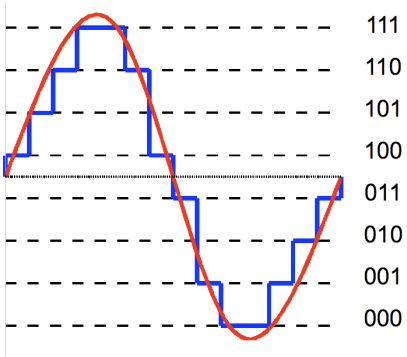
Transformation between Digital and Analog Domain



- 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.

- The diagram here shows the entire process of Analog-to-Digital followed by Digital-to-Analog conversion.
- Let's check out the Analog-to-Digital Conversion first
  - Analog signal is sampled at fixed interval known as sampling interval.
  - At each sampling point, the analog signal level is assigned a value that is closest to that allowed by the ADC used
- The digital equivalent of the analog signal is then processed in the digital processor to suit a particular purpose, for example changing the tone of the audio signal.
- In order for human to hear the processes audio, the digital signal needs to be converted back to its analog equivalent.
- This process is known as Digital to Analog Conversion
- It is achieved by passing the digital signal through a DAC followed by a filter to smooth out the edges of the DAC output.

Digital Quantization



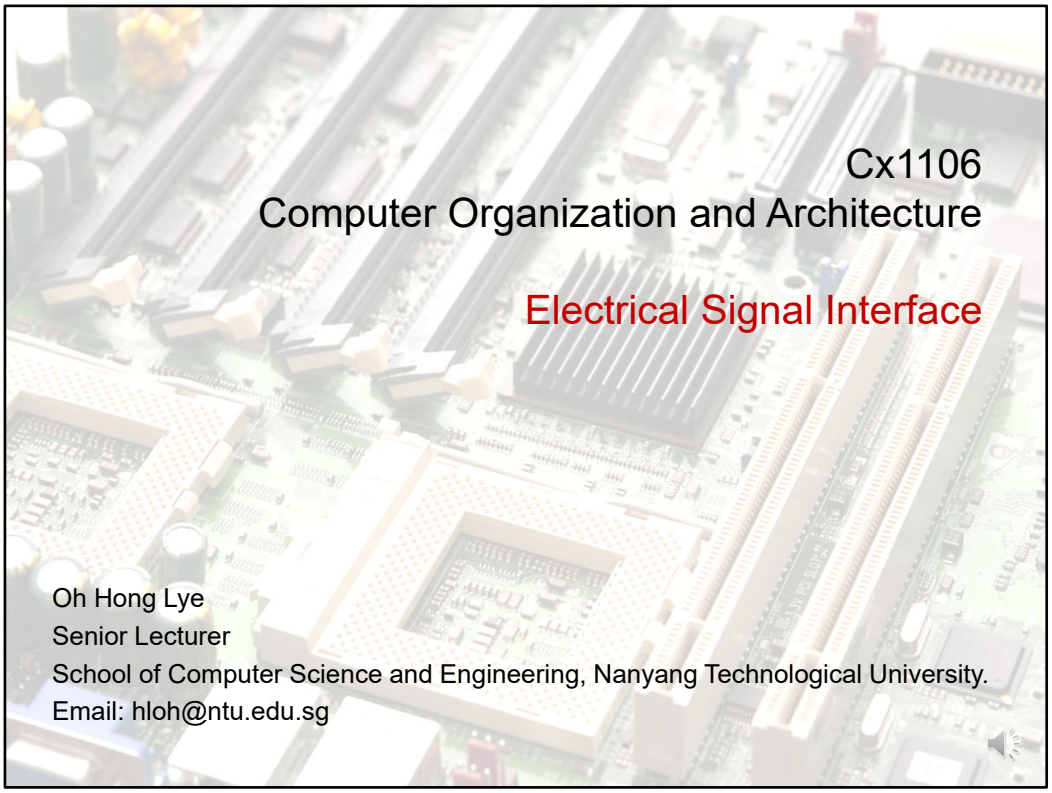
- Figure on the left shows a typical digital quantization process of a analog signal.
- A 3-bit system is used in this example. So all signal will be mapped to one of the 8 possible representations (000 to 111).
- At each sampling point, the analog signal level is approximated to the nearest digital equivalent.

- The diagram in here illustrate the details of how a sine wave is digitized using a 3-bit ADC.
- For a 3-bit ADC, there are 8 discrete output levels 0-7.
- At each sampling point, the analog signal will be assigned to one of the 8 values, the one that is closest in magnitude to the analog magnitude sampled.

Signal Chain between Processor and Real World

- Digital Processors works only in the digital domain so typical process these days is to convert the real world analog signal to digital signal, allow the processor to work on the digital data, and reconstruct back the analog signal to be output to the real world.
- So most of the interfaces we are dealing with in this course are in the digital domain.

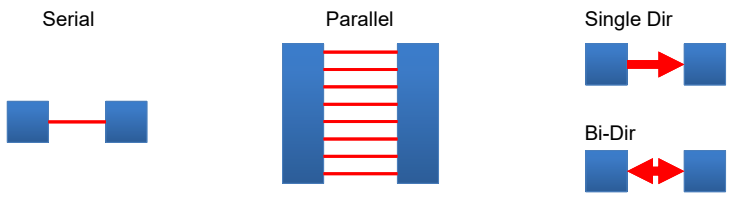
- As mentioned earlier, digital processors are the de facto platform for signal processing these days.
- Since digital processor only understand digital data, all real world signals would have to be digitized before they can be processed.
- Information exchange within the computer is typically digital in nature, they will only be converted to analog when the signals need to get into the real world.
- So naturally, most of the interface we'll touch on in this course is digital in nature.



- This section will cover the topic of electrical signal interface, where we will look at the various requirement for two devices or modules to be able to communicate with each other.

Interface

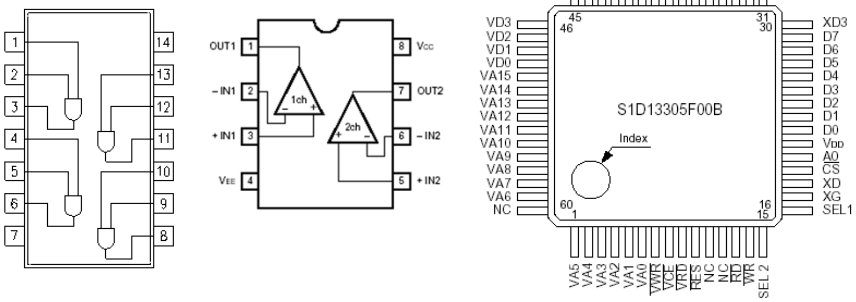
- A boundary where two or more devices meet to exchange information. Some modes of connection below
  - Single-bit Data transfer (Serial)
  - Multiple-bits Data transfer (Parallel)
- For each mode above, the connection could be Single direction or Bi-direction.



- As mentioned previously, Algorithm processing these days are predominantly done in the digital domain.
  - Which is why most of the interface we are going to discuss in this course are digital interfaces.
- Now, before that, what exactly is an interface?
  - In computer terms, interface is a boundary where two or more devices meet to exchange data.
- There as many different ways in which devices can be connected.
- The common ones are listed below.
  - Serial. In this connection, only one data line is available for each direction transfer. So data is transfer a bit at a time.
  - Parallel. Multiple data lines are available to transfer data. So multiple bits are transferred simultaneously.
  - In each way, the connection can be Single direction or Bi-Direction.

Input, Output and Bi-Directional Pins

- Semiconductor devices interface to the outside world via pins.
- Depending on the device design and configuration, these pins are either an input, output or bi-direction (input and output) pin.



Oh Hong Lye / Cx1106 11

- We talked about input and output signals in the earlier slides.
- The diagram you see here are IC pinout diagram, which illustrate the layout of the pins for a particular IC.
- Examples of actual ICs are shown in these photos.
- On the silicon chips, we often see pins or balls attached to their black casing.
- These are actually their Input/Output(I/O) interfaces to the external world.
- The pins/balls are electrical conductors and connects internally to the silicon DIE in the chip packaging.
- There are many different types of packaging, some can be as big as your palm, some smaller than your finger tip.
- In line with the types of interface connections we discussed in the previous slides, there are only 3 types of I/O where direction is concerned: Input, Output and Bi-Directional.

Interface Compatibility

- Interfacing one electronic device to another requires compatibility in
  - Electrical signal level
  - Communication protocol (Handshaking and Data signals).

- This is a very simple slide, but it delivery two important information.
- In order for two devices to talk, the criteria has to be met
  - First, the electrical signal level of the two devices has to be compatible
  - Second, the devices has to speak in the same language. For our case, they have to use the same communication protocol.
- We will discuss the two points in more details in later slides.

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.

- There are two aspects where Electrical Signal Level is concerned.
- First is the safety consideration.
    - When connecting two electrical devices, the most important thing you need to remember is safety.
    - We are not dealing with high power electronics here but if you made the wrong connection, you may still see some smokes coming out from your equipment and boards.
    - So Make sure the output voltage and input voltage of the two devices are compatible. For example, don't connect a 5V device to a 3V device unless you know what you are doing.
- Electronics devices in general do not like to be electrical stressed. So it'll either get fried (spoilt) or its reliability will be reduced if a voltage higher than what it is designed for is applied to the device.
- So do check out the input/output voltage rating of each of the devices you are connecting.

- Second consideration is the digital logic level compatibility which we will discuss in the next slide.

Electrical Signal Level (Digital Data Transfer)

- 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'

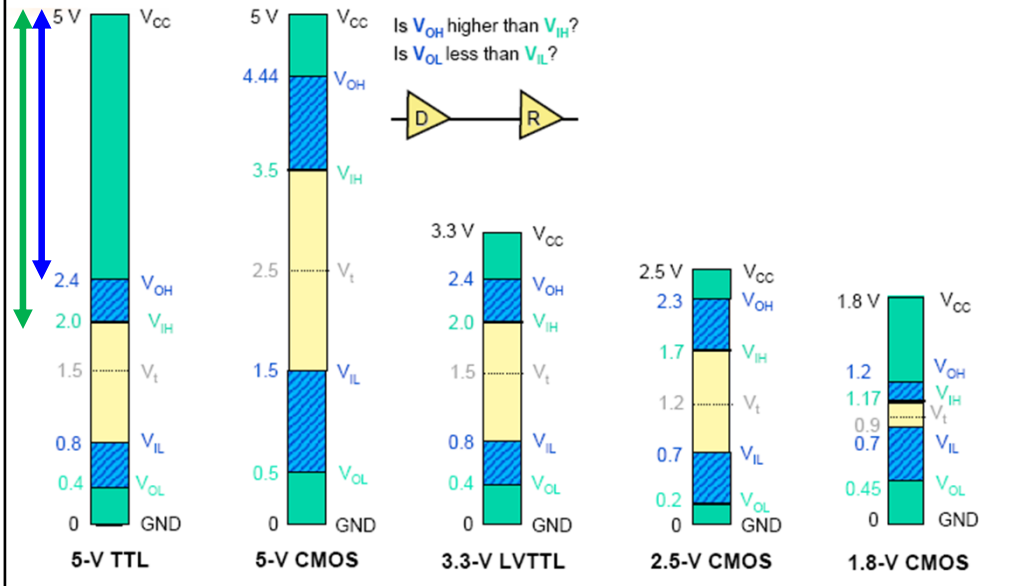
- When a device receive a 5V signal at its input pin, is this considered Logic '1' and '0'?
  - Answer could be either actually.
  - It depends on a few device parameters shown here
- These four parameters dictate whether two devices are able to communicate with each other properly or not.
- Their definition are as follows
  - VOH determine the min voltage that a device will transmit if it is transmitting a logic '1'
  - Similarly, if a device transmit a logic '0', it will transmit a signal with magnitude no larger than VOL.
  - On the receive side, we have VIH and VIL.
  - For a signal to be recognised as a logic '1', its has to at least VIH.
  - For a signal to be Logic '0', its electrical level has to be less than VIL.

Electrical Signal Level (Digital Data Transfer)

- Device #1
  - Output Voltage level for logic '1' = V1+ (e.g. 5V)
    For typical value of V1+, look for V(OH) parameter in device datasheets.
  - Output Voltage level for logic '0' = V1- (e.g. 0V)
    For typical value of V1-, look for V(OL) parameter in device datasheets.
- Device #2
  - Min Input voltage range to recognized as logic '1' = V(IH)
  - Max Input voltage range to recognized as logic '0' = V(IL)
- In order for Device #2 to sense the logic level correctly,
  - Condition 1: V1+ ≥ V(IH) (e.g. V1+ ≥ 2.0V)
  - Condition 2: V1- ≤ V(IL) (e.g. V1- ≤ 0.8V)

- This slide further illustrate what we discuss in the previous slide.
- Device #1 is the transmitter and V1+ correspond to Logic '1' and V1- correspond to Logic '0'.
  - So if we assume that VOH = 2.4V and VOL = 0.4V for Device #1, then V1+ will be greater or equal to 2.4V while V1- will be smaller or equal to 0.4V.
  - These parameters are design dependent and are specified in the device datasheets. You may have studied this in your digital logic module as well.
- When the signal reached Device #2,
  - If we assume that VIH = 2V and VIL=0.8V,
  - the signal will be recognized as a logic '1' if its signal level is larger than 2
  - It will be recognized as a logic '0' if its signal level is less than 0.8V
  - These parameters can be found in the device datasheets as well.
- In summary, two devices will be able to talk to each other if their VOH, VOL, VIH and VIL are compatible.

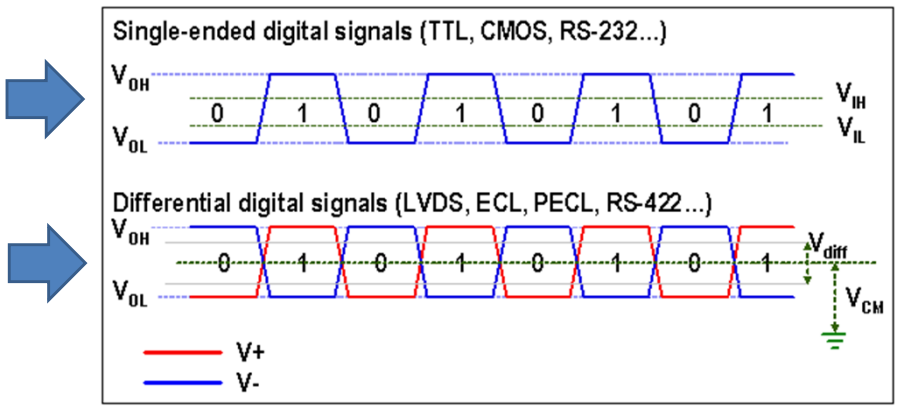
Electrical Signal Level Standards



- Depending on the voltage rating and the chip design, the four parameters (VOH, VOL, VIH and VIL) can be quite different.
- This chart shows some of the devices out there. Its not the complete list, there are many others.
- Bottomline is, please check the device datasheets before you decided to hook them up together.
- VIH and VIL usually covers a larger range compared to VOH and VOL. This is to cater for noise or losses as the electrical signal propagate thru' the circuits.

Differential Signals

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



Source: http://www.ni.com/cms/images/devzone/tut/a/07c0be3031.gif

Oh Hong Lye / Cx1106 17

- What we have seen in the previous slides are single ended signal.
- Electrical signals are also be transmitted via differential signals.
- How it works is that if V+ > V-, than it is a logic '1', if V- > V+, then it'll be a logic '0'.
- In fact, many of the common transmission standard you are familiar with are based on differential signaling.
  E.g. USB, HDMI, SATA etc.
- Differential signals has better noise tolerance in general
  - Because they have a larger margin. You can see in the diagram that the V+ and V- are further apart.and can accommodate more noise.
  - Any external interference will also have the same effect on V+ and V-, so the difference between them remains unchanged.

Communication Protocols

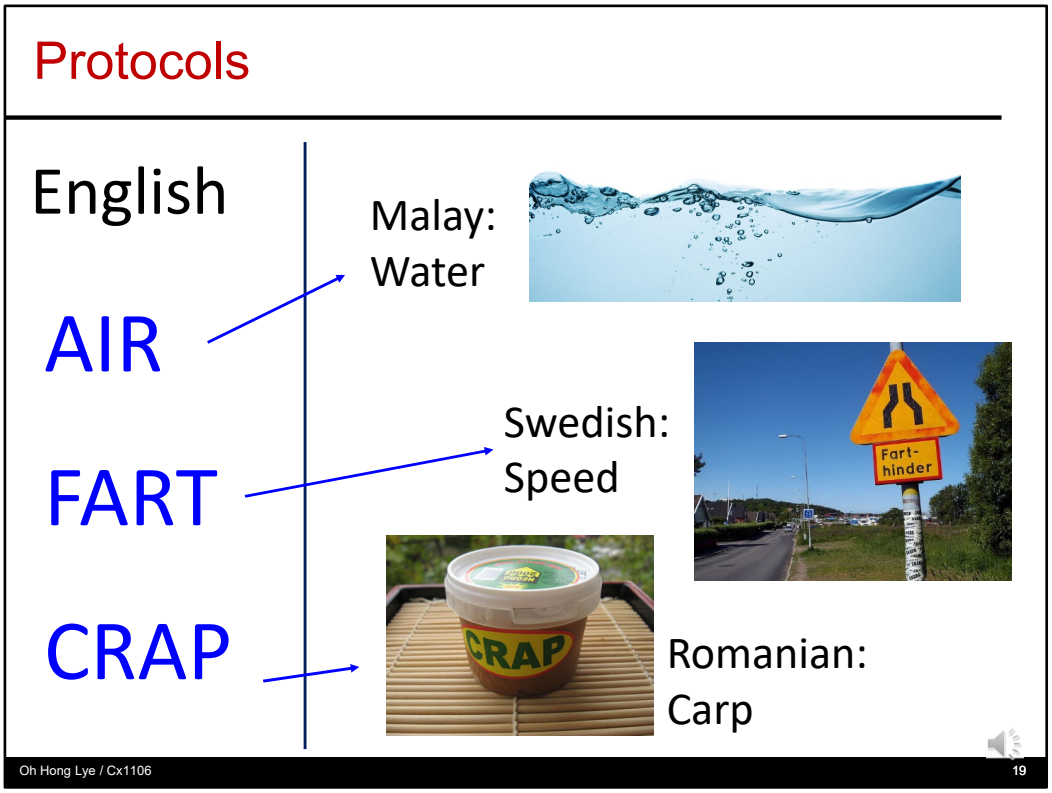
- Communication Protocols refers to how the data are formatted during transmission.
- Some examples
  - Number of bits in a transmission frame
  - What synchronization to use
  - Data width
  - Types of data and its formatting
- Examples of communication protocols are USB, UART, SPI etc.

Protocol 1	**1001001001001000**
Protocol 2	**1001001001001000**

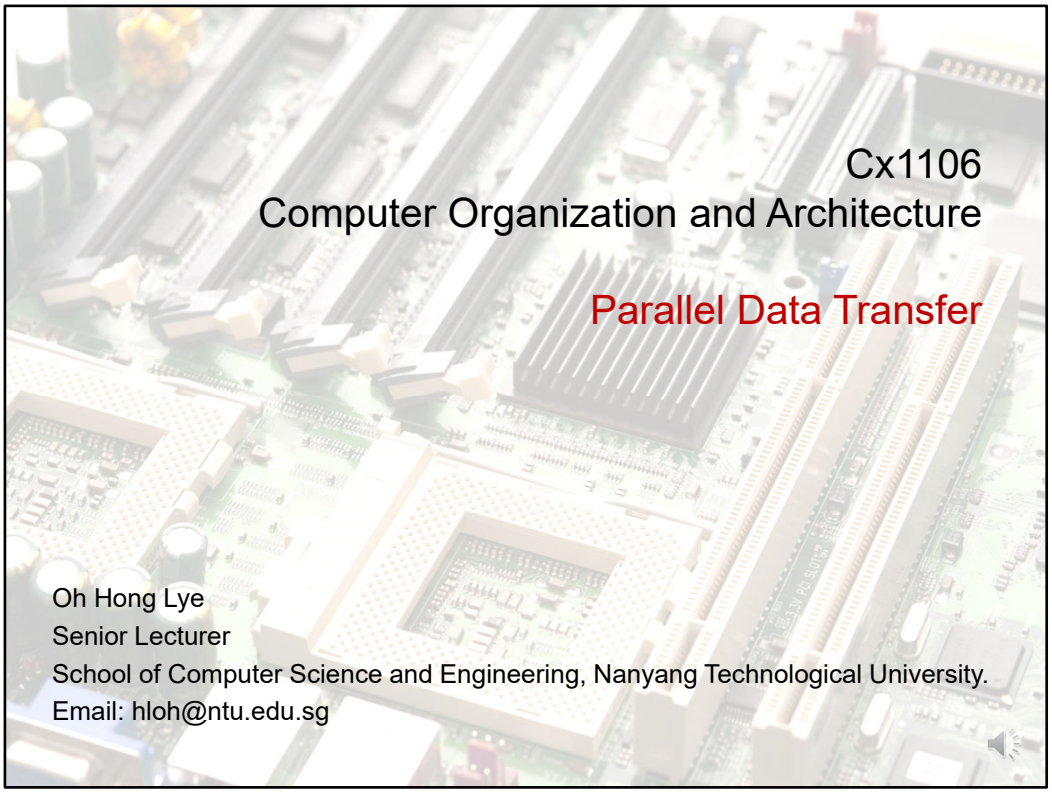
Blue: Synchronization Bits
Green: Data Bits
Red: Error Correction Bits

Oh Hong Lye / Cx1106 18

- The second criteria in order for two devices to communicate properly is that their communication protocol has to be compatible.
- Communication protocols refers to the how data are formatted during a transfer.
  - This could come in different form such as number of data bits, types of synchronisation bits used, error correction bits etc.
- In the two protocols shown here, you can see that although the information transfer is exactly the same, they are in fact two different communication with different synchronisation, data and error correction bits. Obviously, these two device will not be able to communication with each other as they speak different languages.



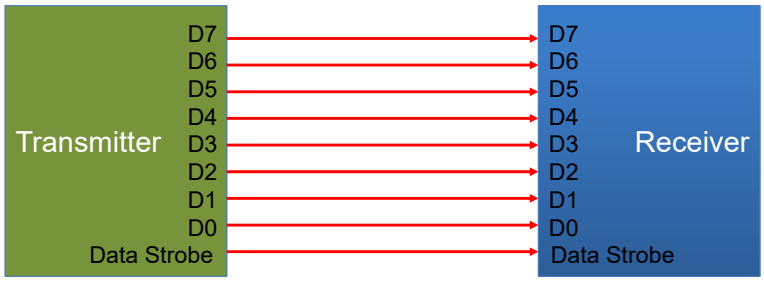
- Just a slide on some fun facts that illustrate examples of same same but different communications.
- You can skip this slide if you are tired.
- So on the left we have three English words: Air, Fart and Carp
- Now the same words in another language actually takes on a totally different meaning
    - Air in Malay is ‘A-eh’, which means water
    - Fart in Swedish is speed, so if you see a sign showing ‘Fart Hinder’ in Sweden, it just a speed bump, not asking you to hinder your fart.
    - And lastly, crap in Romanian is actually a type of fish.



- This section is about transferring data via the parallel interface.

Parallel Data Transfer

- 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).



Oh Hong Lye / Cx1106 2

- Data can be transferred either one bit or multiple bits at a time.
- When multiple bits are transferred simultaneously, it is known as a Parallel interface.
- One example is shown in this slide, where the data is transmitted on multiple lines, collectively known as a data bus..
- The transfer is typically synchronized by a strobe signal, as indicated by the pin "Data Strobe" in the diagram. For example, data could be transferred on the rising edge of the strobe signal.
- The strobe signal may also be referred to as the clock signal of the parallel bus when we are comparing the data transfer rate with respect to the serial interface.

- Some examples of devices with parallel interface are Static Random Access Memory (SRAM), Dynamic RAM (DRAM), Read-Only-Memory (ROM) etc.

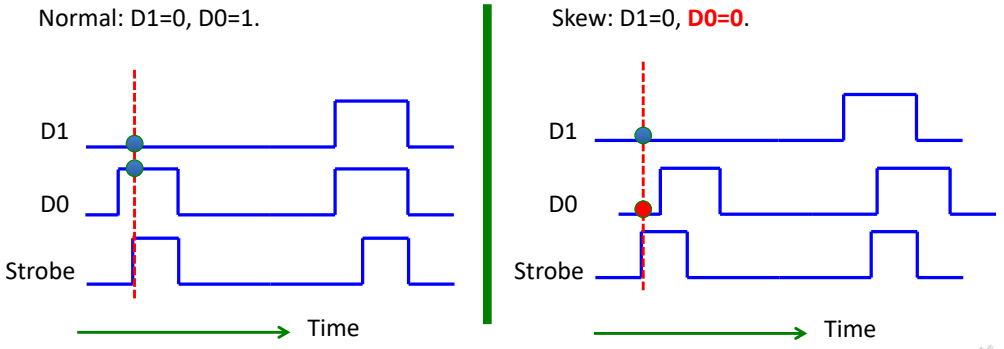
- Given the same clock rate, Parallel interface has a faster data rate than that achievable by the Serial interface.
- This is because Parallel interface has multiple data lines and therefore is able to transfer multiple data bits simultaneously.
- Serial interface, only the other hand, can only transfer one bit at a time since it has only one data line.

- However, because there are more signal lines that are closed to each other, parallel interface is more Prone to signal skew and crosstalk.

- These two phenomenon will result in wrong data being latched by the receiver.
- More details in the next few slides.

Signal Skew

- 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.



Oh Hong Lye / Cx1106 3

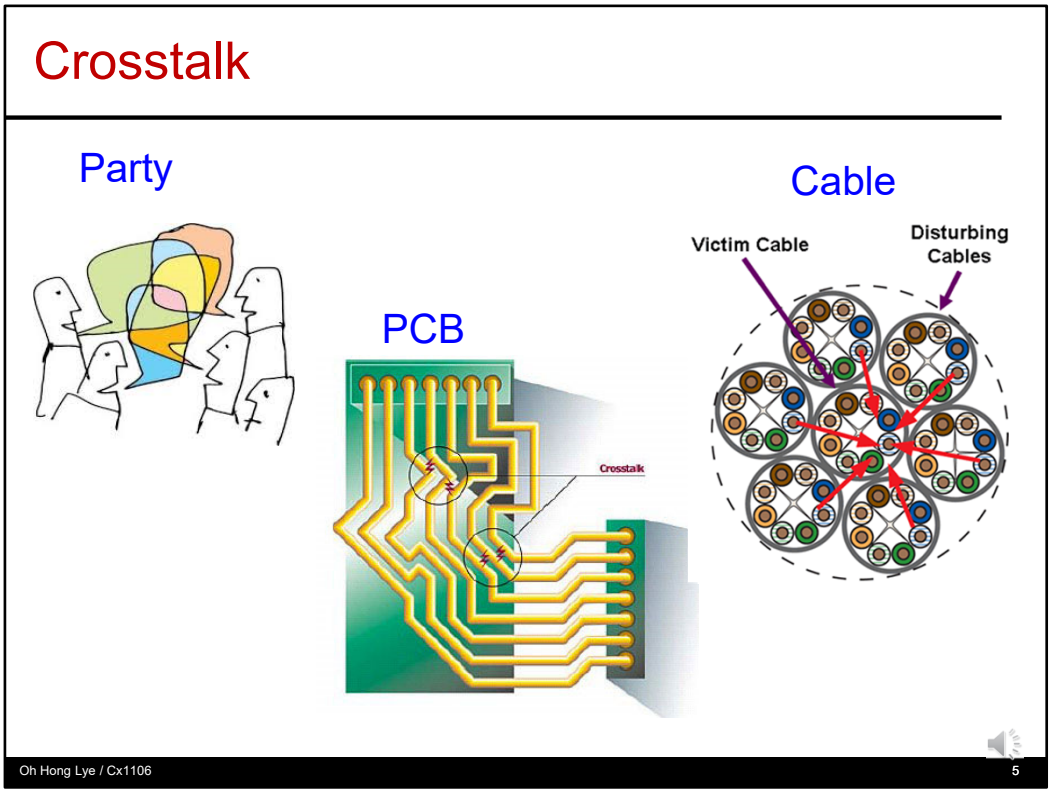
- Signal skew is the phenomenon whereby signals in different data lines of a single multi-bit data transfer travels at different speed.
- This means these signals will arrive at the receiver at different time.
- This is illustrated in the diagrams in this slide.
- Suppose D0 and D1 forms a 2-bit data bus. If the transmitter output '01', i.e. D1=0, D0=1.
- Under normal circumstances when there is no signal skew, the receiver will receive '01' at its end.
- But if for some reason, which I will elaborate later, the signal on D0 gets delayed and reach the receiver later compared to D1 and the strobe signal. Then the receiver will see a '00' instead.
- If you look at the diagram on the right, when the strobe signal goes up, the value of D1 and D0 are both '0', which means the receiver will receive the wrong data.
- There are different factors contributing to signal skews and we will be touching on more details in the next few slides.

Signal Skew – Contributing Factors

- Signal Skew is due to variation in propagation delay between signals from the same data bus.
- Propagation delay is the time taken for signal to travel between two points in a circuit.
- Capacitance and Resistance of the physical data line is a major contributor to circuit propagation delay.
  - The larger the resistance and capacitance, the larger the delay. Illustrated in the equation $\tau=RC$ where τ is the time constant dictating rate of change of voltage levels in the data line concerned.
- Variation in resistance and capacitance of the signal lines can be due to
  - Variation in PCB trace length/width (lead to change in impedance).
  - Connecting active components (capacitors, inductors, IC etc) to some of the signal lines.

- Signal skew is a result of variation in propagation delay between signals.
- So what is propagation delay?
  - Its the time taken by the electrical signal to travel from one point to another within the electrical circuit.
- Many factors can affect the propagation delay. One of the biggest contributor is capacitance and resistance of the wire that the electrical signal is travelling on.
  - The product of resistance and capacitance is proportional to the time it takes for any voltage change on the wire to reach its steady state.
- What do I mean by that?
  - Say the logic '1' of the device correspond to 3V, when the device tries to output a logic '1', the signal voltage will not reach 3V immediately but will take some time to get there.
  - How long it takes is dependent on the RC product value of the circuit it is connected to.
- And these two parameters change if the circuit environment changes.
  - Such as variation in length/width of PCB traces,
  - Connection of components such as capacitor, resistors, ICs on the circuit concerned.
- So when connecting high speed devices via parallel bus, it is very important to ensure that the propagation delay of the data line within the same bus is kept the same so data can be transferred correctly.
- One common practice is to try to keep the length of the PCB traces of the same data bus to be of similar length.

- You can see that some interesting patterns formed by the PCB traces connecting between the processor and high speed DRAM in this photo.
- These method of routing the PCB trasces is known as serpentine routing.
- For this case, the main objective being to ensure that the propagation delay of the data lines are similar to prevent signal skew.

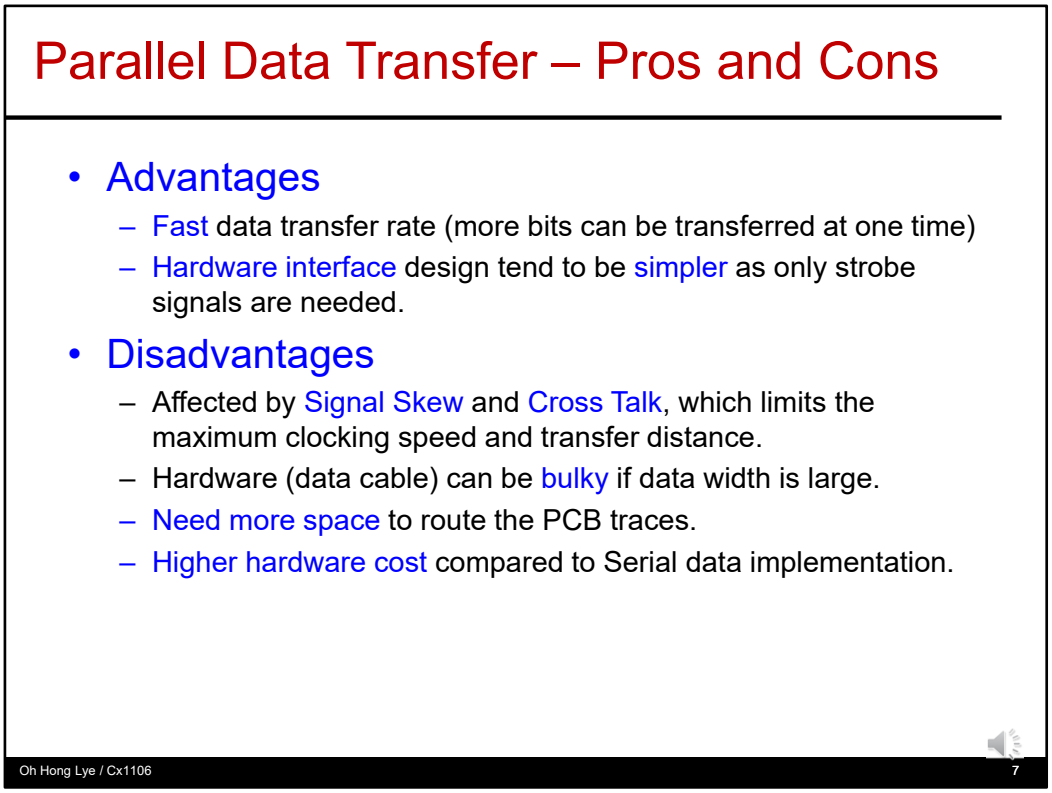


- Next is Cross Talk.
- In layman term, Crosstalk are unwanted interference from neighbours.
- In a party where many people is talking simultaneously, voices from people other than the person you are talking to will come across as Crosstalk to you and your partner.
- Similarly, in a PCB with many data lines closely packed together, there will be a lot of cross interference between these signals as they
- propagate through the circuits.
- Interference can be via
    - Electromagnetic domain where the wires behaves like antenna emitting Radio Waves.
    - Electrical domain as the electrical signal from one wire could potentially coupled to the other wires at the connection point.
      They could also be coupled via ground current returning via different paths on the PCB ground plane.

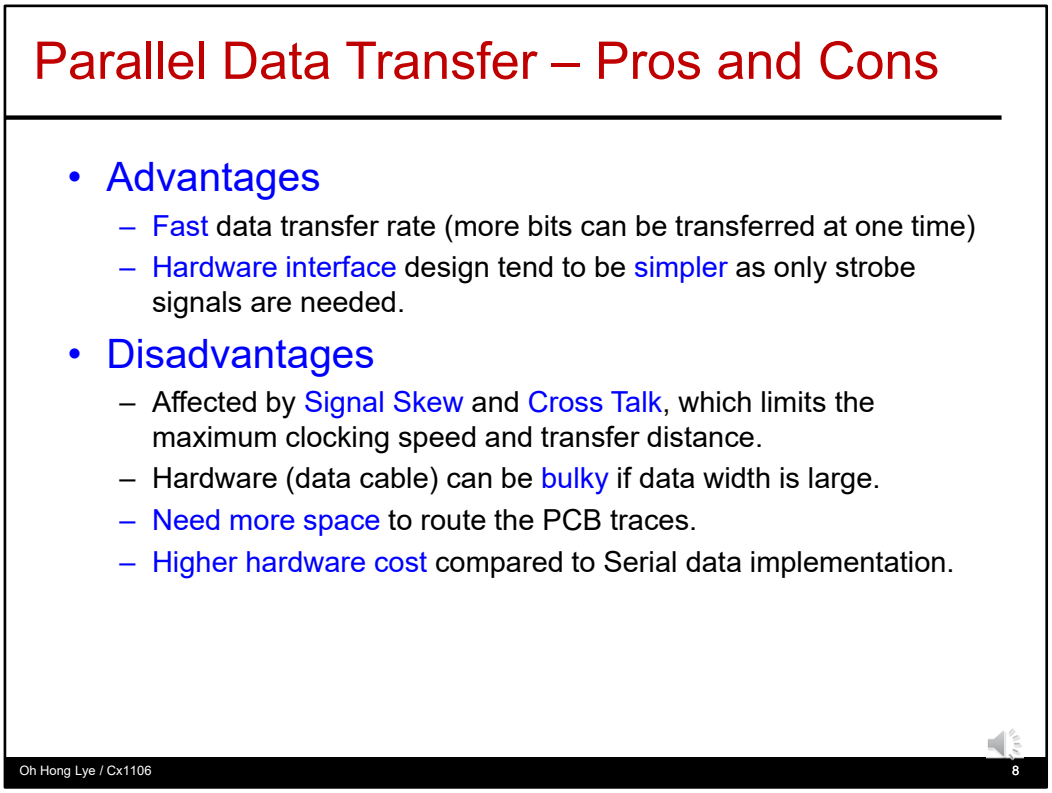
Crosstalk – Electrical Circuit

- Crosstalk are undesired coupling of signals from one circuit to another circuit.
- In a parallel bus context, the close placement of the data lines in PCB routing or cabling enables the effect of electrical signal in one trace/wire to be coupled over to the other. Creating undesired interference (crosstalk).
- Crosstalk can be transmitted electrically or via electromagnetic radiation (the trace/wire acts like an antenna).

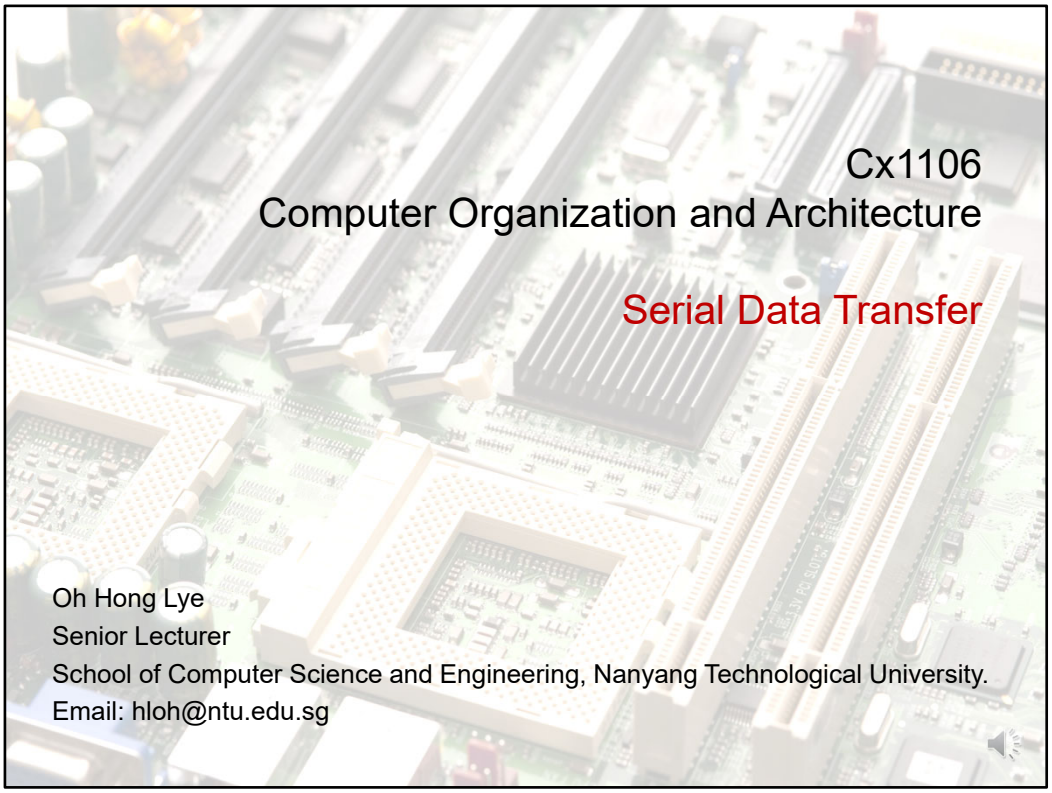
- This slide summarise what I have talked about in the previous slide.
- You can pause the video for a while to go through the information presented.



- Summarising the pros and cons of parallel interface
- Advantages
- Given the same clock rate, Parallel interface is able to transfer data at a faster rate compared to Serial Interface.
- However, because there are more signals lines (data and strobe) that are closed to each other, parallel interfaces are more prone to signal skew and crosstalk. These two phenomenon will result in wrong data being latched by the receiver. This also limit the maximum clock speed and maximum distance that the signal can travel without being affected by signal skew or cross talk.
- Parallel bus or cable also takes up more space compared to serial bus because more wires are involved.
    - As a result, it is more complex and costly to design the PCB for parallel bus.
- The cable is also more bulky so house keeping can be an issue.
- The forst photo shows difference in bulk comparing the IDE HDD cable many years back and the SATA HDD cable that we use today.
- The second photo shows you how messy it get be when you have a few of these IDE cables in your Computer Chasis. I used to have issue just trying to figure out which cable belongs to which device!

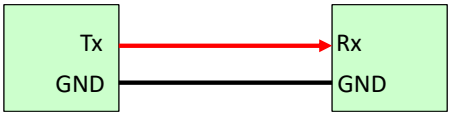


- Summarising the pros and cons of parallel interface
- Advantages
- Given the same clock rate, Parallel interface is able to transfer data at a faster rate compared to Serial Interface.
- However, because there are more signals lines (data and strobe) that are closed to each other, parallel interfaces are more prone to signal skew and crosstalk. These two phenomenon will result in wrong data being latched by the receiver. This also limit the maximum clock speed and maximum distance that the signal can travel without being affected by signal skew or cross talk.
- Parallel bus or cable also takes up more space compared to serial bus because more wires are involved.
    - As a result, it is more complex and costly to design the PCB for parallel bus.
- The cable is also more bulky so house keeping can be an issue.
- The first photo shows difference in bulk comparing the IDE HDD cable many years back and the SATA HDD cable that we use today.
- The second photo shows you how messy it get be when you have a few of these IDE cables in your Computer Chasis. I used to have issue just trying to figure out which cable belongs to which device!



- Next, we will be touching on the Serial Data Interface

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.

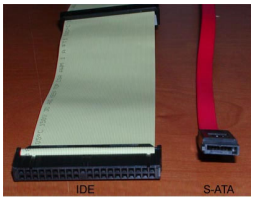
- Serial Data interface, as mentioned earlier, transfer one bit of data at one time
- So given the same clock rate, its data transfer rate will be lower than that of the parallel interface.
- However, because only one data line per direction is involved,
  - it is less affected by signal skew since less wires are involved => chances of a skew happening is less.
  - It is also less affected by crosstalk since there are less neighbouring wires or cables as compared to a parallel interface..
- With less influence from signal skew and crosstalk, a serial interface bus, if well designed, could potentially achieve a higher clock rate compared to the parallel bus, this allows its transfer rate to be closer to that of the parallel bus.

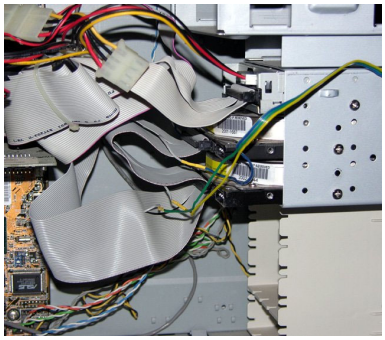
Serial Data Transfer Pros and Cons

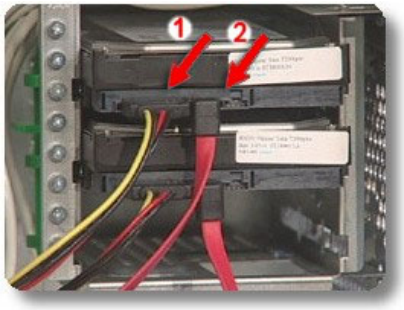
- Advantage
  - Less affected by signal skew and crosstalk because there are less electrical wires involved compared to parallel data transfer. Hence, able to support higher frequency clocking.
  - Able to transfer data reliably over a longer distance.
  - Lower cost since less wires and connectors are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

- To summarise
- The Advantages of Serial transfers are
  - It is less affected by signal skew and crosstalk as there are less wires involved in each interface
  - As the effect of signal skew and cross talk is not as significant when thre are only a few wires, serial interface can therefore be potentially clocked at a higher frequency and is able to transfer data over a longer distance.
  - It is also less costly since less wires and connectors are needed.
- The Disadvantages are
  - Given the same clock rate, Data Transfer rate for serial interface is lower than parallel interface.
  - Hardware interface design may be more complex because serialization and deserialization operation is needed to convert
    the serial bit data to multiple-bit data format such as word, byte used by the processor.

Parallel and Serial Comparison





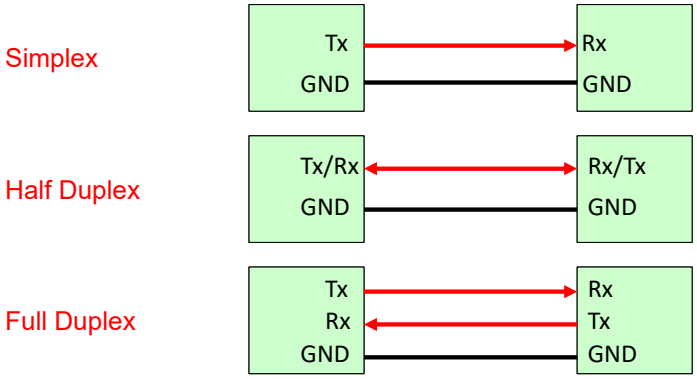


Oh Hong Lye / Cx1106 4

- This is an example of the space saving you can achieve by converting from parallel to serial interface.
- Example here shows the difference between the old IDE interface, which is a big mess and the new SATA interface for HDD.

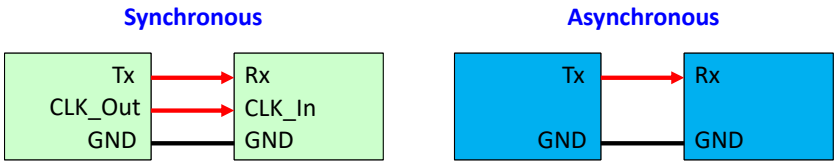
Serial Data Transfer Mode

- 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



- There are 3 types of serial transfer mode:
- In Simplex mode, One device is the transmitter while the other is the receiver. Data transfer is only in one direction.
- For Half-Duplex mode, Both devices can operate as transmitter as well as receiver. But at any point in time, there is only one transmitter and one receiver, with data flowing in one direction.
- Lastly, in Full Duplex mode, Both devices can operate as transmitter as well as receiver. There are separate data line for each direction transfer so each device can transmit and receive data at the same time.

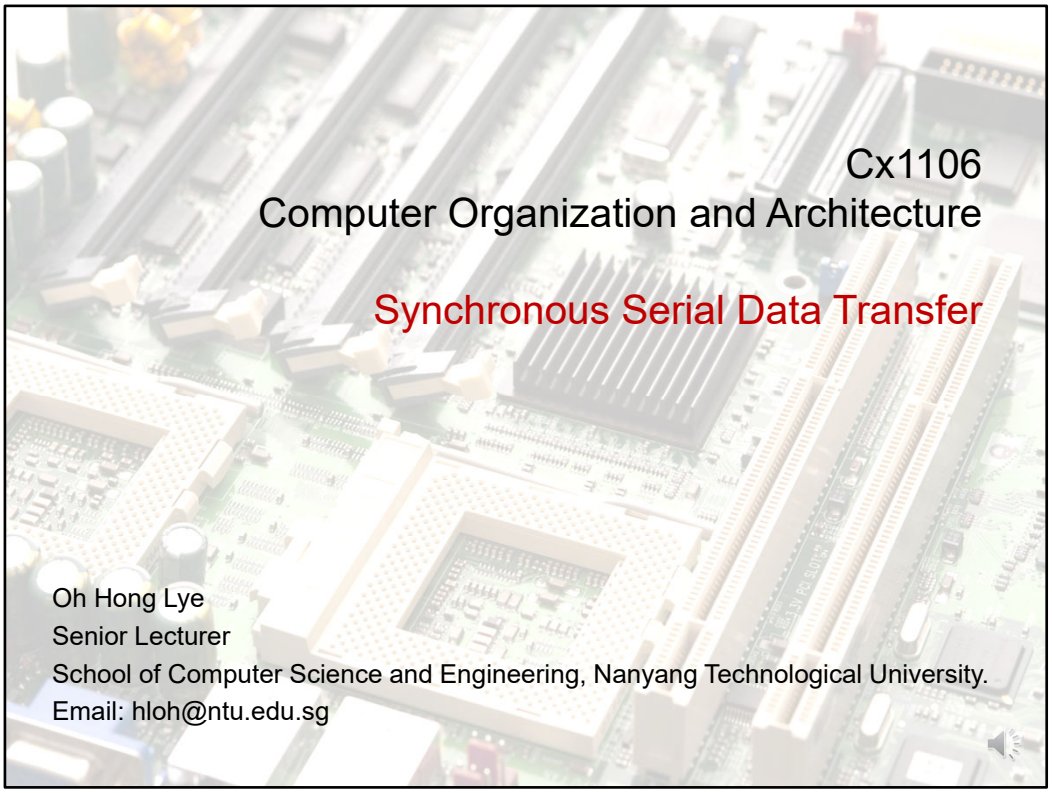
Synchronous vs Asynchronous



- If there is a common clock signal between the Transmitter and Receiver, then the communication is termed synchronous. Else, the communication is termed asynchronous.
- In synchronous transmission, there is a common clock signal to synchronize the data transfer. E.g. receiver to latch in the data at every rising edge of the clock.
- In asynchronous transmission, there is no common clock signal so devices have to agree on a pre-fixed clock frequency to use for data transfer.

Oh Hong Lye / Cx1106 6

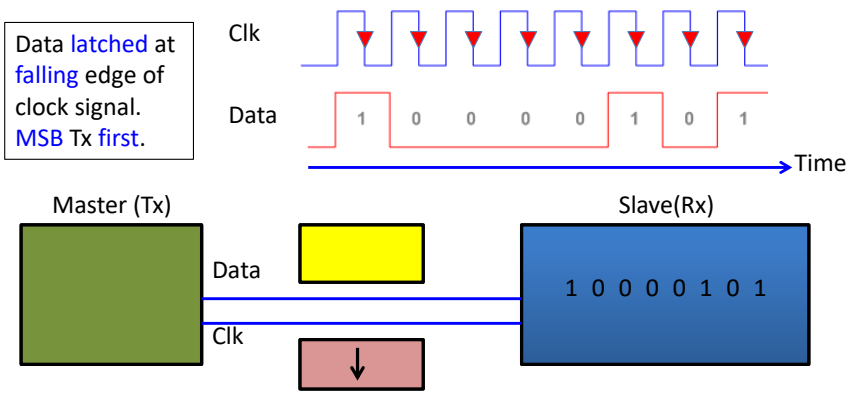
- Serial communications can be Synchronous or Asynchronous in nature and they have different design considerations during implementation.
- A serial communication is synchronous when there is a common clock signal between the transmitter and receiver.
- If a common clock signal is absent, then the transfer is asynchronous in nature.
- For synchronous transmission, the common clock signal is used to synchronize the data transfer, it tells the receiver when to latch in the data bits send by the transmitter.
- For Asynchronous transmission, since there is no common clock. But that doesn't mean that sychronisation is not required between the transmitter and receiver.
- In fact, synchronization is still needed for the data transfer to be successful.
    - Synchronization is achieved by sending synchronization information over the data line, and the devices will have to agree on the clock rate to be used for transferring data.
- This obviously will lead to some issues and we'll go into more detail later.
- Note that this concept of Synchronous and Asynchronous transfer that we discuss here is referring to the electrical signals layer.
- At higher protocol layer, the word synchronous may take on a slightly different meaning and it is possible to implement synchronous communication over an asynchronous physical link. You need not bother about that now but keep in mind that such implementation is possible.



- We will be covering synchronous serial transfer in the next few slides.

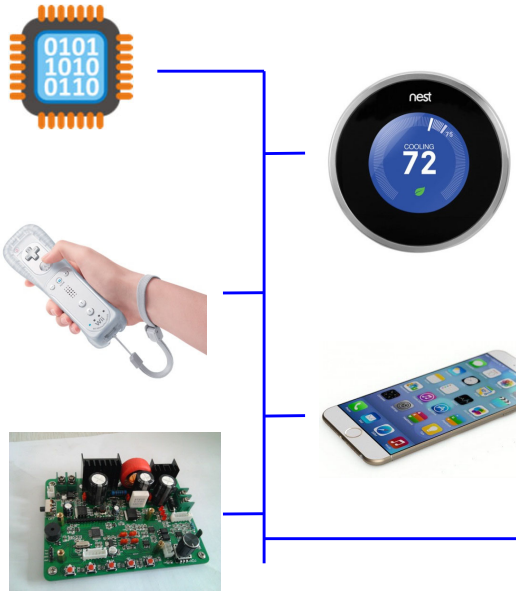
Synchronous Serial Transfer

- Common clock signal between transmitter and receiver to synchronize the data transfer.
- Master-Slave configuration. With Master providing the clock signal.
- Potentially allows faster transfer rate since no data overhead is needed to synchronize the transfer.



- As mentioned, synchronous serial interface has a common clock signal that is used to synchronize the data transfer.
- The devices are usually configured in a Master-Slave configuration, with master providing the clock.
- What is shown here is a typical transmission.
  - The signal waveform is as shown. This is the typical way a signal transmission diagram is illustrated in documents.
  - What is means is that data on the left will be sent/receive first.
  - In this example, the data is latched on the falling edge of the clock.
  - The Master will first output a '1' and data will be latched by the slave on the falling edge of the clock signal.
  - This is followed by a '0' and corresponding clock edge.
  - Master will continue to send the rest of the data bits, which is latched into the receiver on the falling edge of the clock.
- Depending on the configuration, data can also be latched in at the rising edge instead.

I2C and SPI Bus



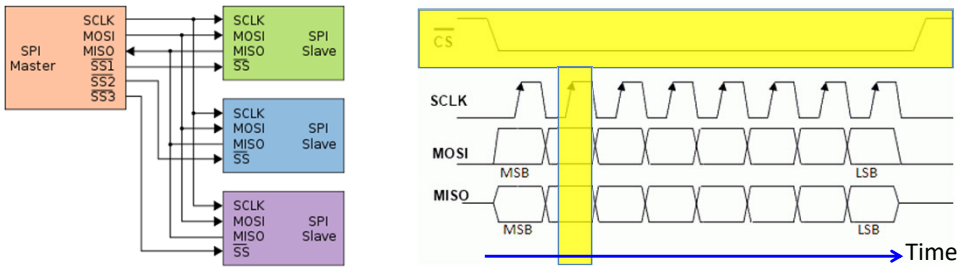
- Popular serial buses used by processor to transfer data and control many peripheral devices.
  - Accelerometers
  - Temperature Sensors
  - Touch Screen Controllers
  - Power Supply Modules Configuration
  - Audio/Video Codecs Configuration

I2C will not be covered in this module

Oh Hong Lye / Cx1106 9

- I2C ad SPI are two of the popular synchronous serial bus standards used by the processor for interfacing to other devices.
- Some of the examples are shown here.
- In this course, we will only be touching on SPI bus interface.

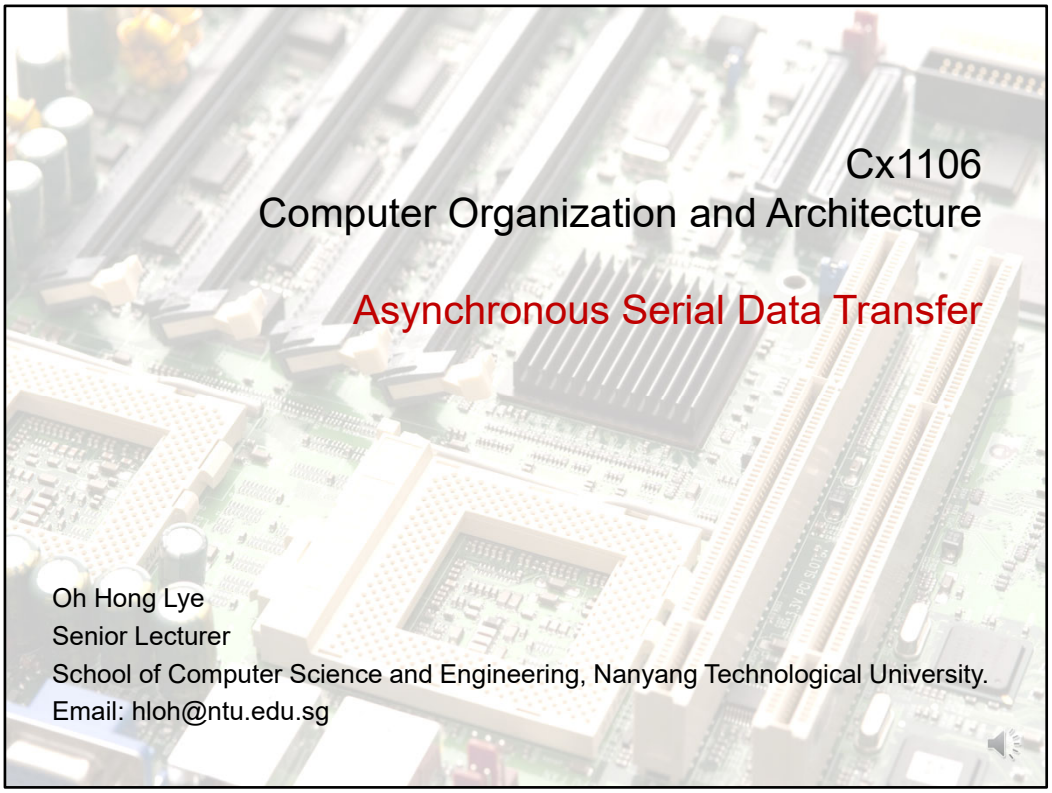
Serial Peripheral Interface (SPI) Bus



- 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.

Oh Hong Lye / Cx1106 10

- SPI bus is a synchronous serial bus. You can see the explicit clock line that the Master used to synchronize transfer over SPI interface.
- To initiate a transfer, the master will first pull the Slave Select pin low. This will enable the target SPI slave.
- Data on the MOSI and MISO pins are then latched on the rising edge of the SCLK signal.
- MOSI refers to Master Out Slave In, this allows master to output data to slave. Similarly, MISO refers to Master In Slave out.
- SPI bus allows multiple slaves to be connected to one master.



- This section is on asynchronous serial interface. This is the last section of the Serial Interface chapter.
- Yes, your agony is going to be over soon.

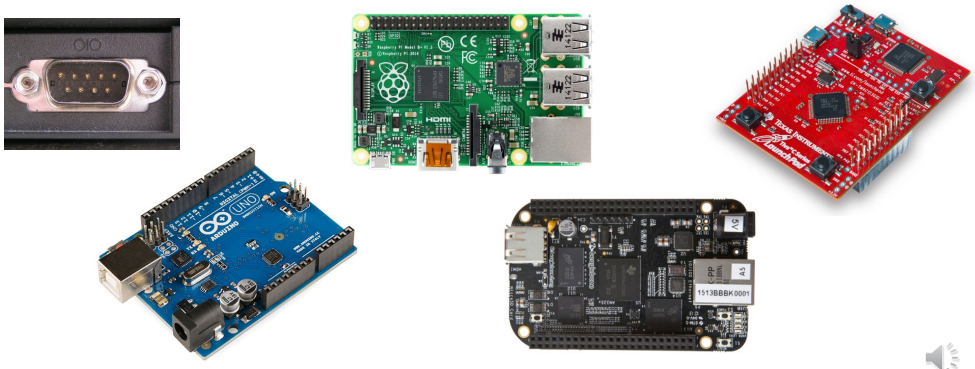
Asynchronous Serial Transfer

- 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).

- In Asynchronous transfer, as mentioned earlier, there is no common clock between the transmitter and receiver so they need to rely on other means to achieve synchronization.
- First, both devices needs to agree on a certain clock frequency and the number of data bits in a data packet.
- They also need a special SYNC word to indicate the start and stop of transmission. This SYNC word is embedded in the data packet. Remember again that there is no explicit clock line so all information has to be transferred via the data line.
- Upon receiving the Start SYNC word, the receiver will start to track the data using its own local clock. Since the transfer clock frequency is known beforehand, the receiver can roughly guess when the data bits are coming in and latch it.
- The fact that the transmitter and receiver are using two different clocks means that over time, these two clocks will drift relative to each other since there is no way these two clock will run at exactly the same frequency.
- If the clock skew kept increasing, it'll eventually lead to wrong data being latched by the receiver.
- One way to mitigate this issue is for the transmitter to send SYNC word periodically so that the receiver can use that info to re-align their local clock.
- The re-alignment of clock here means getting the clock edge of the two local clock to occur at the same time stamp.

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

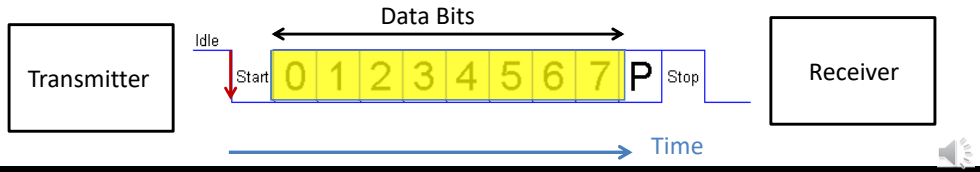


Oh Hong Lye / Cx1106 13

- One of the most commonly used Asynchronous Serial bus standard is the UART.
- It stands for Universal Asynchronous Receiver Transmitter.
- The PC serial COM Port you see here used to be found in every PC but has since been replaced by USB.
- But UART as a protocol is still a popular protocol for various processor development boards like Arudino and Rasberry Pi.
- You don't see a physical COM Port on these boards though, what you have is the USB interface.
- What happened is that a Virtual COM Port will be enumerated over the USB when processor board such as Arduino is connected to the PC. And the Virtual COM Port uses the UART protocol for communication.

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.



Oh Hong Lye / Cx1106 14

- When the UART is IDLE, its data line will be at a logic HIGH.
- When the transmitter want to send data, it'll first send a STAR bit by pulling the data line LOW.
- This falling edge of the data line will alert the receiver to get ready for data reception.
- The transmitter will then output the actual data at a known frequency, also know as the baud rate.
- At the end of the data transfer, the transmitter can optionally output a parity bit.
- Function of a Parity bit is to enable error detection at the receiver side. We discuss this in more details in the next slide.
- After that, the transmitter will send a STOP bit to end the transmission.
- If the transmitter wants to send another word, it needs to restart the whole process by transmitting the start bit again.
- You can't send multiple words with only one set of Start and Stop bit.
- For UART protocol, a few of the symbols or status has a fixed logic level
    - IDLE state is always Logic '1'
    - START bit is always Logic '0'
    - STOP bit is always Logic '1'

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.

- UART parity scheme is designed to detect errors during transmission
- There are 3 different options: Even, Odd or No Parity.
- If Even Parity Scheme is employed, the transmitter will count the number of '1's when it is transmitting the data. Depending on the number of '1's recorded, it will then transmit a '0' or '1' as the Parity bit, so that the total number of '1's in the Data field and the Parity Field is an Even number.
- For example, if the data transmitted is 1110000, which has three '1's, then the transmitter will transmit a '1' as the parity bit, so that the total number is 4, which is an even number.
- Similarly, Odd Parity scheme implies the transmitter will set or clear the Parity bit so that total number of '1's in data and parity field is ODD.
- No parity means parity bit will not be generated.
- On the receiver side, after it received the data and parity bits from the transmitter, it will count the number of '1's in there, if it doesn't match the parity scheme used, then it implies that one or more bits in the transmission is wrong. But it won't be able to tell the exact bit or bits that are wrong.
- Another type of error is the framing error. This will be flagged if the receiver detects a logic '0' in the bit position where the STOP bit is expected.

UART Tx Example

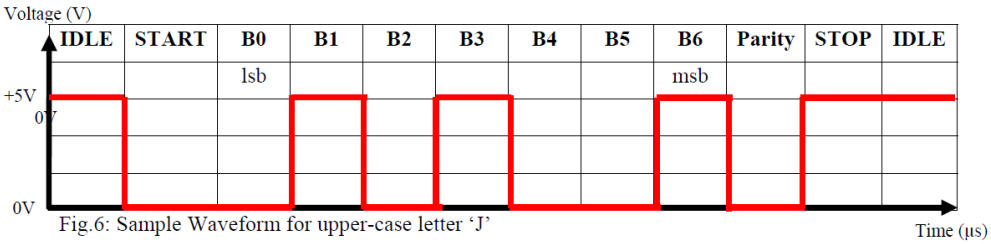


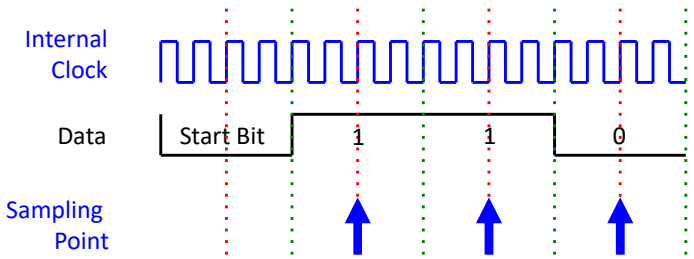
Fig.6: Sample Waveform for upper-case letter 'J'

- 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
- 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.

- One example to illustrate the UART transmit operation.
- Here we have a 7O1 configuration, which implies 7 Data bits, Odd Parity and 1 STOP bit. There is always only one START bit in each UART frame.
- Notice that for UART transmission, LSB is transmitted first.
- In this example, the data transmitted is 0x4A or 100 1010 in binary. But you'll notice that the waveform shows that the LSB is transmitted first so the logics on the UART signal waveform shows 0101001.
- There are 3 '1's in the data, so the transmitter will clear the parity bit. You can see that Parity bit = 0 in the diagram.

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.

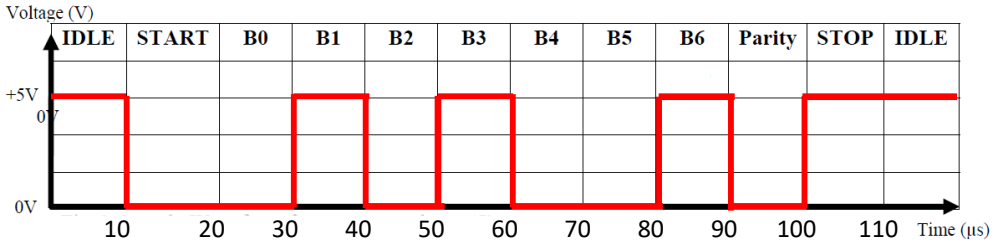


Oh Hong Lye / Cx1106 17

- Ext, let's take a look at the UART receiver operation, which is more complicated than the transmitter.
- The receiver needs to constantly monitor the data line to see if transmitter has anything for him.
- In a real world design, interrupt mechanism is used to make the process more efficient.
- The receiver needs to know the baud rate in order to sample the data correctly.
- And the internal clock of the UART receiver is typically much faster than the baud rate. One value is 16X.
- The receiver will use this faster clock to derive the best location to sample the incoming data.
- The example below illustrates the process. I'm using a 4X sampling here to simplify things. Actual internal clock is usually much faster.
- We have the first logic '0' as the start bit. This is followed by the data bits
- Since the internal clock is running at much faster frequency, the receiver will able to estimate the mid point of each data bit and perform the sample at that instance.

UART Rx Example

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



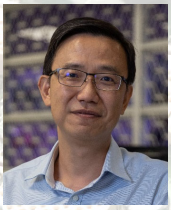
- RX Data = 1001010b
- RX Data = 0011000b ?
- **Information Sampled @ 200000 bps:**
  0000110011000011001 1

Oh Hong Lye / Cx1106 18

- Similarly, let take a look at the timing diagram example of a UART receiver to illustrate the concepts learnt.
- The first Logic '0' is the START bit,
- it is followed by the data bits and the bits that comes into the UART receiver in chronological order are 0101001.
- Since LSB is transmitted first, the actual data received is 1001010 or 0x4A.
- Odd parity scheme is used so Parity bit as transmitted by the transmitter is '0' so that total number of '1's is an odd number.
- Now, looking at the waveform, is it possible for the receiver to receive 0011000 instead?
- Answer is yes and that happen when the transmitter and receiver baud rate is different.
- In this case, the receiver baud rate is twice that of the transmitter.
- So receiver will be sampling 20 points instead of 10.
  - It'll then use the UART configuration of 7O1 to decode these 20 samples, as shown in the slide.
  - The first data received is 0001100. Its LSB first so the actual value is 0011000.

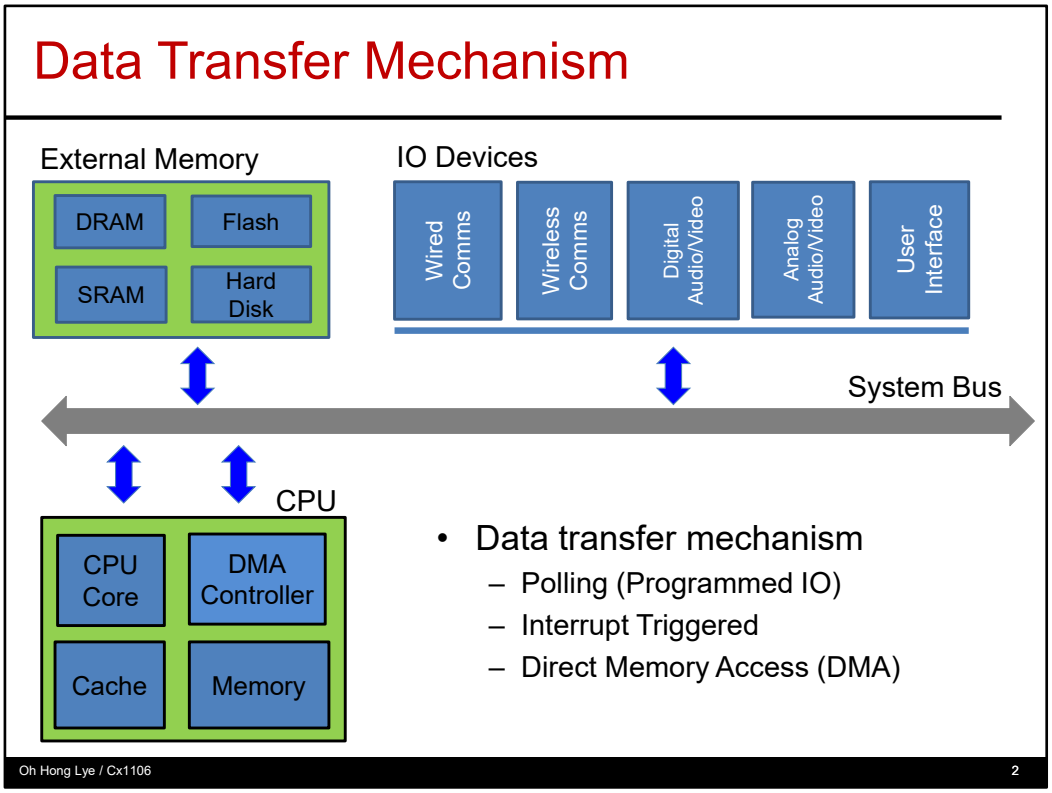
Cx1106
Computer Organization and Architecture

Data Transfer Mechanism



Oh Hong Lye
Senior Lecturer
School of Computer Science and Engineering, Nanyang Technological University.
Email: hloh@ntu.edu.sg

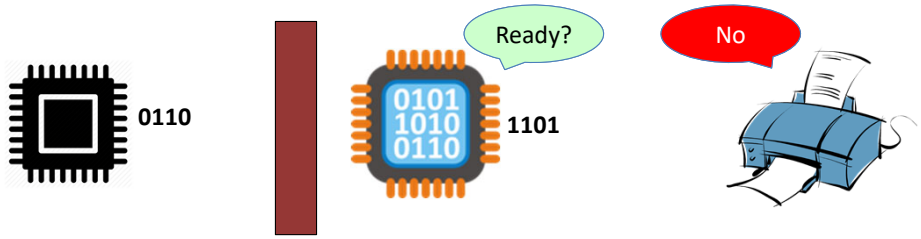
- This section is on Polling and Interrupts, which are two data transfer mechanism commonly used in computer system.



- A computer system consist of the CPU, memory and IO devices.
- These blocks are connected by a system bus, which is used to transfer data between the blocks.
- The two controllers that can initiate these transfers are the CPU core and the DMA Controller.
- The 3 transfer mechanisms that we'll touch on in this lecture are
    - Polling
    - Interrupt Triggered
    - DMA – stands for Direct Memory Access (we'll go into more details later).

Polling Technique

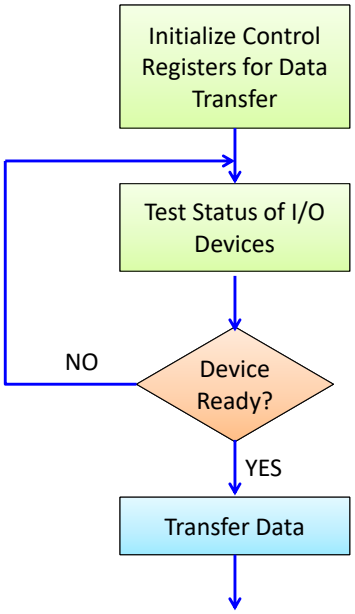
- CPU polls a certain I/O port continuously (using software) for data or readiness of the port to perform a data transaction.
  - For example, the CPU polls the printer port continuously to see if the printer is ready to accept data.
  - If it is ready, the CPU writes a data byte to the printer port.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



- First, Polling. Its also known as Programmed IO.
- In this method, the CPU poll a certain IO port continuously for data.
- One example is CPU polling the printer port to see if the printer is ready to accept data.
- If the printer is ready, the CPU will send the data to the printer for printing. Else, it'll wait.
- AS CPU uses 100% of its resource to do the polling, it is not able to entertain request from any other peripherals until the polling is done.
- The following animation explains how the process works.
  - Here we have a processor, printer and an external device
  - When the processor wanted to send some information to the printer, it'll first poll the printer to see if the printer is ready.
  - If the printer says 'No', the processor will continue to poll. Note that during this time, no other device will be able to get the processor's attention since it dedicate 100% of its attention on the polling.
  - Eventually, the printer will be ready and the processor can send the information over.
  - Once that is done, the process will remove the wall installed and allow other device to communicate with it.

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.



- This sides shows the flow chart of the process that the CPU goes through when handling a data transfer suing polling technique.
- CPU first perform some system initialisation
- It then poll the status of the device it wants to talk to
- If the device is not ready, it'll continue to poll the device status in a loop.
- Once the device is ready, CPU will transfer the data 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.

- Advantages of using polling technique to transfer data is that
  - Since it is a pure software approach, the programmer has complete control over the entire process, this makes testing and debugging simple.
- However, it result in very inefficient use of the CPU resources because
  - The CPU needs to dedicate 100% of its resource to do the polling, as such it would not be able to perform any other task.
  - Specifically, program execution will be held up while CPU is wating for the devices to be ready.

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 mechanism is another common data transfer mechanism used and will result in a more efficient usage of CPU resources.
- Interrupt is a signal that is used to alert the CPU that its attention is needed.
- This signal can be an electrical pulse or some change in register status.
- When the CPU receive an interrupt, it can choose to service it, or not to service it.
- If the CPU decides to service an interrupt request, it will proceed to execute some pre-designed sequences. We will discuss more of that in later slides.
- In general, interrupt is typically used to trigger the CPU to start some operation, which could be some data transfer to memory, registers or other devices.

Polling vs Interrupt

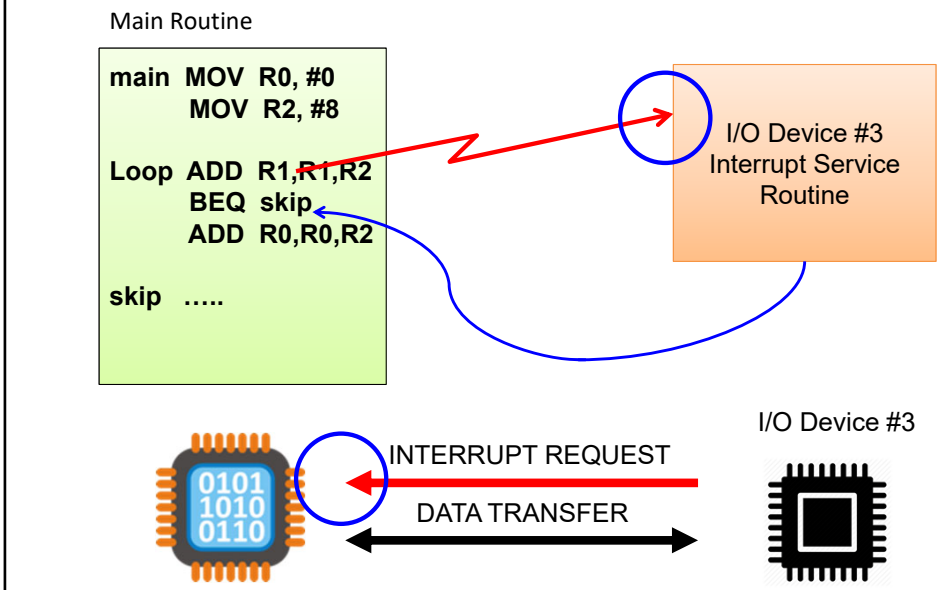
- Boiling Water Analogy

Polling:
Check every few minutes

Interrupt:
Listen for the whistle

- This slide use the example of boiling water to give a simply analogy to the difference between polling and interrupt.
- If you boil water using an open pot, you will likely need to monitor the status of the water continuously or periodically to see if the water is boiled. This is similar to the polling mechanism.
- But if you boil water in a kettle with whistle, you would not need to monitor the kettle and is free to go about doing your work until you hear the whistle from the kettle. The whistle sound is similar to the interrupt request that inform the processor, which is you in this case, that your attention is needed.

Interrupt Triggered Data Transfer



- This slide will illustrate the interrupt mechanism with actual program.
- Supposed the processor is running some code in the main routine
  - After two instructions,
  - IO device #3 interrupted the CPU.
- If the CPU decides to service the interrupt,
  - the program will then branched off to a special routine known as the interrupt service routine.
  - For this case, it's a routine to transfer data between CPU and Device #3.
- After the necessary data is transferred, the control is passed back to the main routine where the
  code will continue where it has previously interrupted.
- One key timing parameter you need to take note is delay between
  - the point where CPU receive the interrupt request
  - To the point where it enter the ISR
- This delay is known as the interrupt latency, you will need to use this in your tutorial.

Interrupt Vector Table

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.

- In the previous slide, the CPU will branch off to the ISR if it decides to service the interrupt.
- But how does the CPU known where to branch to? Especially when there are many interrupt sources in the system.
- The answer is in a table known as the Interrupt Vector Table.
- This table contains the starting address of each ISR that is present in the system.
- Each interrupt has a unique location within the table so the CPU knowns exactly where to look for it when searching for the starting address of the corresponding ISR.
- For example, in the table shown, UART interrupt is stored at index 9, so if a UART interrupt occur, CPU will be looking at location corresponding to index 9 for the starting address of the ISR.
- Note however, that the information in the table here is for illustration purpose only.

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.

- This slide summarizes the interrupt control flow we discussed in the previous slides.
- You can pause the video to go through the points again.
- Note that CPU will save a copy of the processor context before branching to the ISR. This so call 'context' is basically the status of the key registers.
- This context will be restored after the ISR complete and the control is passed back to the interrupted routine, so that the interrupted routine can proceed from where it was interrupted.

Interrupt Service Routine

- 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).

- This slides summarizes what I mentioned in the previous slide.
- You can pause the video to go through the points again.
- Some additional points that we didn't touch on in previous slides:
  - ISR tends to be a short routine so as not to hold the CPU for extended duration.
  - It is possible for CPU to receive multiple interrupt requests at the same time,
    so CPU has to be some scheme in place to decide which interrupt to service first.

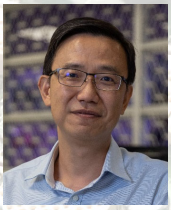
Pros/Cons of Interrupt Triggered Technique

- 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.

- And here we have the advantages and disadvantages of interrupt triggered IO.
- Advantage
  - It leads to a more efficient use of CPU resources
  - And allow prioritization and pre-emption. This is One of the key enablers for high level OS as it allows
    a more critical software task to pre-empt other tasks to get the system going. Pre-empt here mean to
    halt the current task temporarily and run the higher priority task.
- Disadvantages
  - More hardware is needed to implement an interrupt mechanism, the control flow from interrupt request to the fetching of the ISR starting address are all handled in hardware.
  - Since hardware is involved, there is less visibility to the underlying operation so resultant program is more complex and more difficult to debug.

Cx1106
Computer Organization and Architecture

Direct Memory Access



Oh Hong Lye
Senior Lecturer
School of Computer Science and Engineering, Nanyang Technological University.
Email: hloh@ntu.edu.sg

- This section is on Direct Memory Access, DMA in short. We will go through the operation of a DMA controller and how it can be used to relief the CPU from data transfer task.

Optimizing CPU resources in Data Transfer

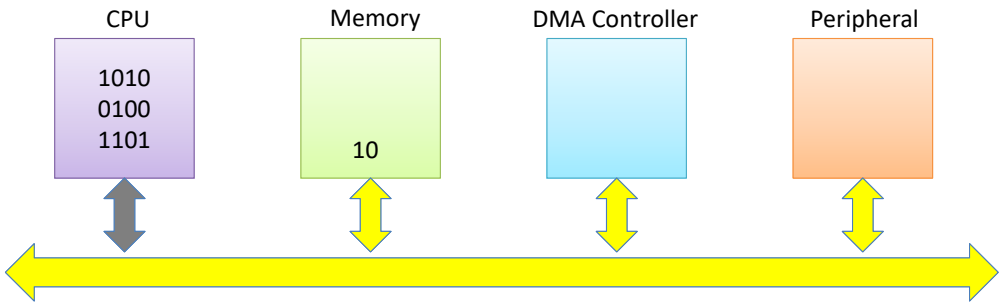
- Till now, we have been using CPU to perform the data transfer.
- This becomes increasing inefficient as amount of data increase as CPU has to spend most of its time moving data.
- As such, CPU has less time to perform algorithm processing.
- DMA controller (DMAC) is added to relief CPU of the data 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.

- Up till this point, our discussion on data transfer mechanism has always involve CPU as the controller that perform the data transfer task.
- As the amount of data increases, the time needed to transfer data increases as well. So it would appear that we are not using the CPU effectively as the main purpose of a CPU should be number crunching rather than transfer data from point A to point B.
- If we are able to relief the processor from the data transfer task and have it focus on algorithm processing, the overall system will be better utilised and efficient. This is where the DMA controller comes in.
- DMA Controller is a controller that is able to control the system bus to perform data transfer without help from CPU.
    - In doing so, it relief the CPU from doing data transfer so that it could focus on processing data.
- DMA controller has dedicated hardware that if well designed, would be more efficient than CPU when transferring data which requires complex address manipulation.
    - For example, de-interleaving the Left and Right channel of audio data.
- Operation wise, it is possible to have both the CPU execution and DMAC data transfer to happen at the same time if both of them are not using any common resources.
- But if they are using the same resource, e.g. system bus, then one of them will have to wait.
- With common resources, there will be a need for arbitration, and one way of

doing it is via priority. Whoever has the higher priority will get to use the system bus. In real word processor, this priority can be fixed, either CPU has higher priority compared to DMAC, or vice versa. Or it could very well be configurable.

- For our course, we will assume that there is only one system bus available in the processor. That means the CPU and DMA will have to fight of the usage of the system bus.

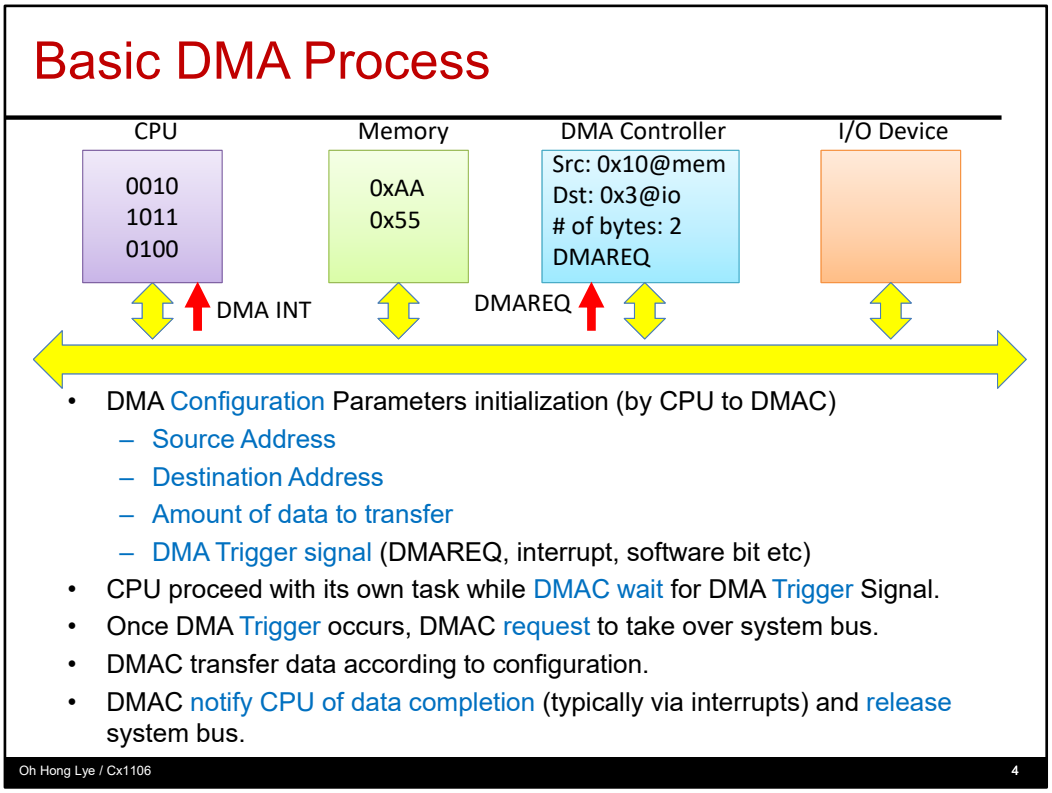
DMA Controller (DMAC)



- 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.

Oh Hong Lye / Cx1106 3

- A DMA controller, DMAC in short, is able to transfer data between various modules within a computer system.
  - It does this by generating the addresses and initiate the read/write controls.
- The DMAC shown here is known as a Fetch-and-Deposit type of DMA, where the data from the source will be transferred into the DMAC before it is transported to the destination.
- The following animation illustrates how bus is shared between CPU and DMAC.
  - The CPU may be doing some operations that doesn't require the system bus.
  - So DMAC is able to perform data transfer between the memory and IO device.
  - As and when the CPU needs the system bus, control will be given to the CPU.
  - And after the CPU completes its data transfer, the DMAC can continue its operation.
  - In real world design, the bus priority between the CPU and DMAC may differ from the illustration.
  - In this example, we are assuming that CPU has a higher priority than DMAC.



- Now that we have some idea of what a DMA is, let's walk thru a DMA process to understand
  how a DMAC is configured and the various control signals involved.
- In order for DMAC to transfer data, it needs a few key parameters. Typically, these parameters are configured by the CPU to the DMAC.
  - Source Address of Data
  - Destination Address of Data
  - Number of words of Data to transfer
  - And the trigger signal to tell DMAC when to start the transfer
- After the DMAC is configured, it'll just wait in IDLE state for the trigger signal.
- Once the trigger signal comes in, it'll request for the system bus and start the data transfer.
- When all the data has been transferred, DMAC will release the system bus and issue a DMA interrupt signal to the CPU, informing CPU that the data is ready to be processed.

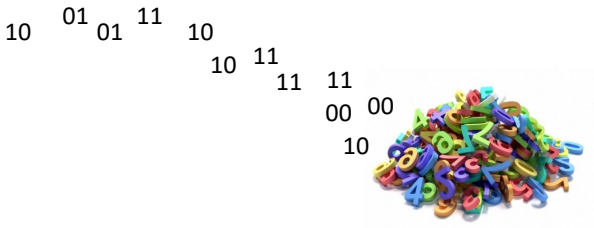
DMA Mode of Operation

- Different processors has different DMAC design, typically with slight variance in terms of how they transfer data and the data type (Audio, Video etc) they are optimized for.
- In general, DMAC could transfer the block of data they are tasked to transfer in the following mode
  - Burst
  - Cycle Stealing
  - Transparent
- Do note that you may encounter DMAC design that varies from the basic mode of operation discussed here but general concept will still apply.
- In fact, majority of the time, you'll encounter DMAC design that uses a combination of the modes described

- Different DMAC may differ in the way they transfer data.
- There are three basic data transfer modes that DMAC typically use, namely, Burst, Cycle Stealing and Transparent. We will get to the details of each mode in later slides.
- Note, however that in actual processor, it is likely that you will encounter DMAC that uses a combination of these modes.

Burst Mode

- 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.



- First, let's take a look at the Burst Mode.
- In Burst Mode, DMAC will transfer multiple units of data when they take control of the system bus.
  - This could be the entire block of data the DMAC is tasked to transfer, OR
  - It could be a subset of the block, that means it probably take the DMAC a few burst to complete the block transfer.
- CPU can continue with its operation as long as there are no conflict in resource utilisation, for example the system bus.
- However, if the CPU needs system bus during the time when DMAC is doing the transfer, the CPU may need to wait till DMAC complete its transfer.
- Burst mode would result in faster data transfer rate but may potentially cause the CPU to be suspended for extended period of time.

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

- If DMAC is using Cycle Stealing mode, it will only transfer one unit of data after taking control of the system bus, after which it will release the bus back to the CPU.
- The name cycle stealing comes about as the DMAC will try to adjust its use of system bus by virtue of the processor operation.
  - For example, it will tend to execute the data transfer
    - Between CPU instructions, OR
    - Between Processor Pipeline stages
- Any resultant suspension of CPU execution is kept short because only one unit of data will be transferred so CPU will be more responsive to other events.
- The time available for data transfer is shorter than that of the Burst Mode, therefore, data transfer rate will be lower as well.
- As this method is less disruptive to CPU operation compared to Burst Mode, it is preferred in system that required CPU to be more responsive, for example, real-time monitoring.

Transparent Mode

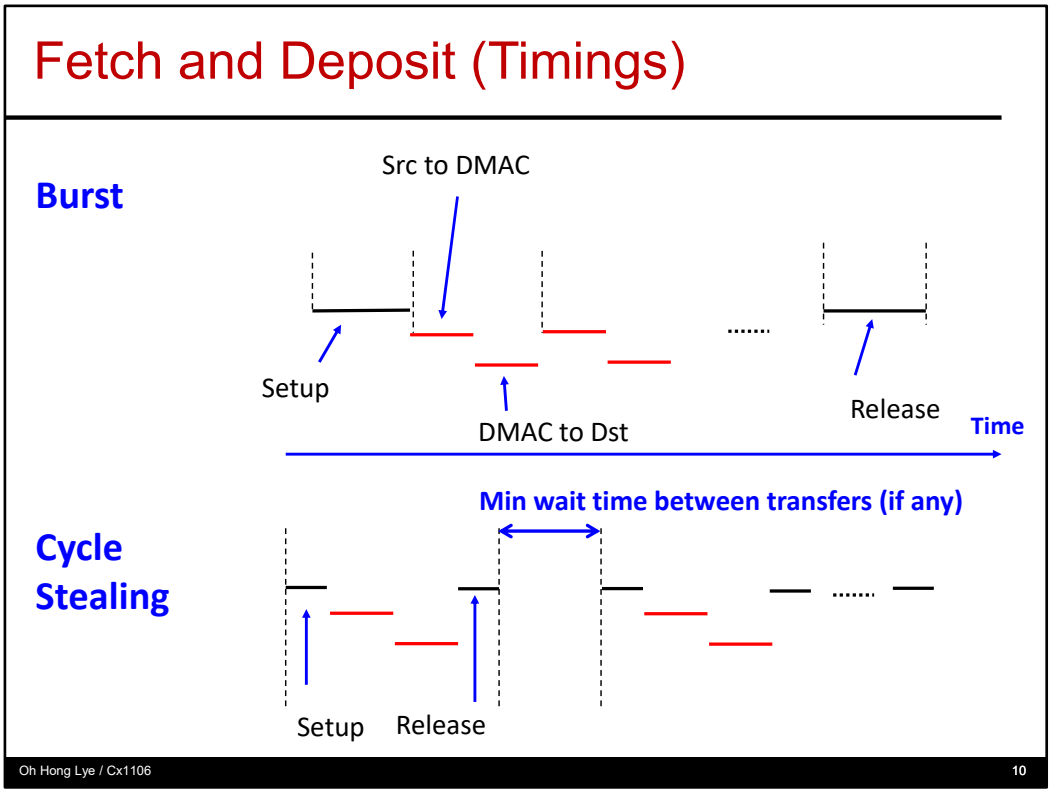
- DMAC transfer data only when CPU is not using the data bus.
- Zero impact to CPU performance in terms of data bus access.
- Potentially the slowest transfer rate among the three modes.
- More Complex hardware needed to detect when CPU is not using the data bus.
- The CPU activity detection technique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

- Lastly is the Transparent Mode, which can be view as the extreme od the cycle stealing.
- In this ode, the DMAC will only start a transfer if it detected that the system bus is not being used by the CPU.
- This incidentally result in the least disruption to the CPU but also have the slowest data transfer rate among the three DMA transfer mode.
- In addition, it would require some complex hardware that allows the DMAC to snoop the system bus in order to determine the activity level on the bus.

Comparison of DMA Transfer Modes

- **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.
- **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.
- **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.

- To summarise.
- Burst Mode allows a high transfer rate but may potentially suspend the CPU for extended period of time.
  - Suitable for application where data transfer is bursty in nature, such as HDD transfer.
- Cycle Stealing mode interleave its data transfer with CPU instructions so it has less impact to the CPU performance.
  - This allows the CPU to be more responsive, but at a trade off of lower DMA transfer rate.
- Transparent mode DMA does not affect the CPU performance at all as it only work when CPU is not using the system bus.
  - However, implementation may be more complex as it now has to predict/detect when the system bus is free.



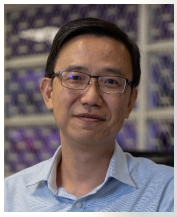
- This is an additional slide to illustrate, from timing diagram perspective, the DMA operation under different modes.
- The slide shows the timing diagram for the burst and the cycle stealing mode.
- In the burst mode
    - Recall that once the DMAC have the control for the system bus, it'll transfer the entire block of data to the destination.
    - Timing wise
        - Setup correspond to the time DMAC takes to request control of the system bus from the CPU
        - This is followed by transfer of data
            - Each unit of data is transfer in two stages,
                - first from Source to DMAC internal buffer
                - Second from DMAC to the destination.
        - After the last unit of data is transferred to the destination, the DMAC will release the bus control back to the CPU.
- In Cycle Stealing mode
    - The DMAC will request control of the system bus from the CPU, transfer one unit of data and release the bus back to the CPU.
    - The DMAC may also be required to wait for a certain time duration before it can request for the bus control again.
    - The timing is as shown in the diagram
        - The DMAC will request for the bus control, which is 'Setup' in the diagram, it will than transfer the data, release the bus back to CPU

and wait for a fixed duration before requesting for the bus again.

- Take note of these timing diagrams as you will need them to complete your tutorial questions.
- This is the last slide of the DMA section.

Cx1106
Computer Organization and Architecture

Computer Memory Introduction



Oh Hong Lye
Senior Lecturer
School of Computer Science and Engineering, Nanyang Technological University.
Email: hloh@ntu.edu.sg

- We will be touching on the topic of computer memory in the next few videos.
- Memory is an important sub system in the computer and has a big impact to the overall system performance.

Computer Memory

Semiconductor-based

Solid-State Drive

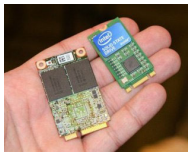
SODIMM

Memory IC Chips

CF, SD, miniSD, microSD, MMC

Solid-State Drive

Solid-State Drive



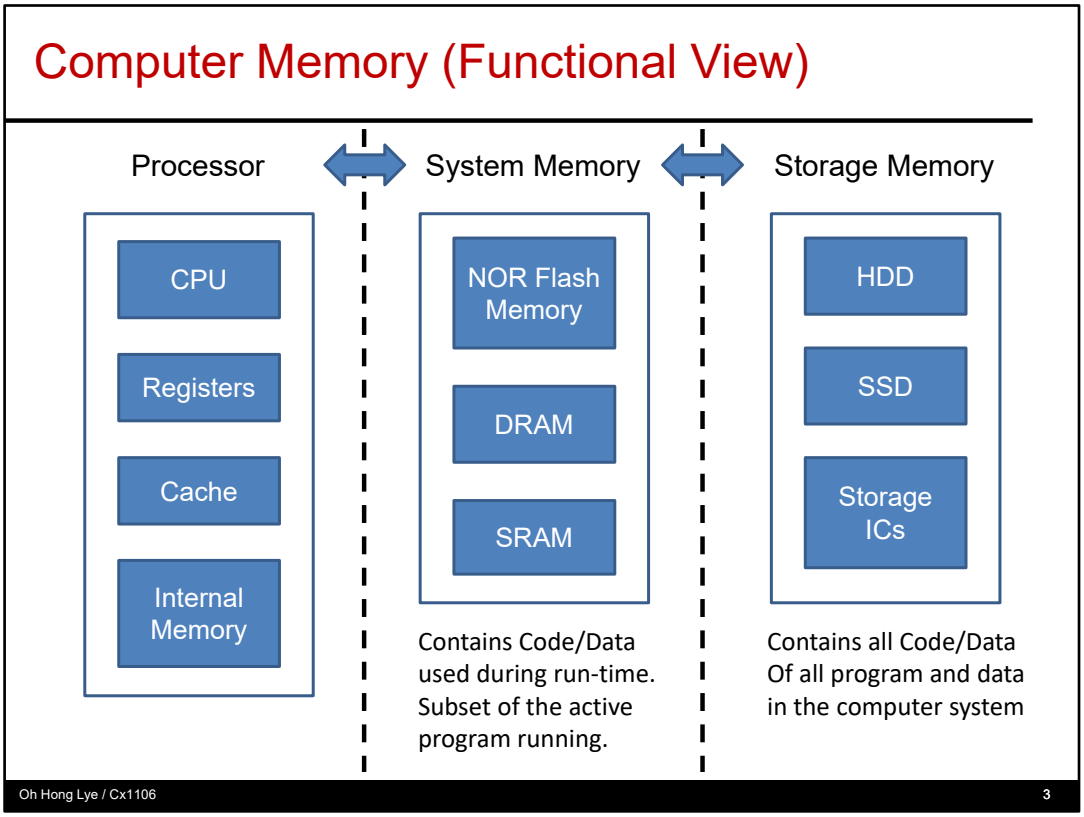
Thumb Drive

Magnetic-based

Magnetic HD

Oh Hong Lye / Cx1106 2

- Computer memory is a key component in the computer system as this is where all the information and code are stored.
- There are many types of memory
    - Semiconductor based
    - And the Good old Magnetic HDD
    - Although HDD is fast being replaced by SSD (solid state drive) in laptop these days, you may want to know that HDD is still very much involved in your daily life, in the form of cloud storage.
        - The iCloud, Google Drive, drop box cloud storge that you use stores information in Data Centers, and HDD is still the main storage element in Data Centers. We will get to more of that in later slides.
- One question, among these memories, which are volatile and which are non-volatile?
- To know the answer, we have to first understand what is volatile and non-volatile memory.

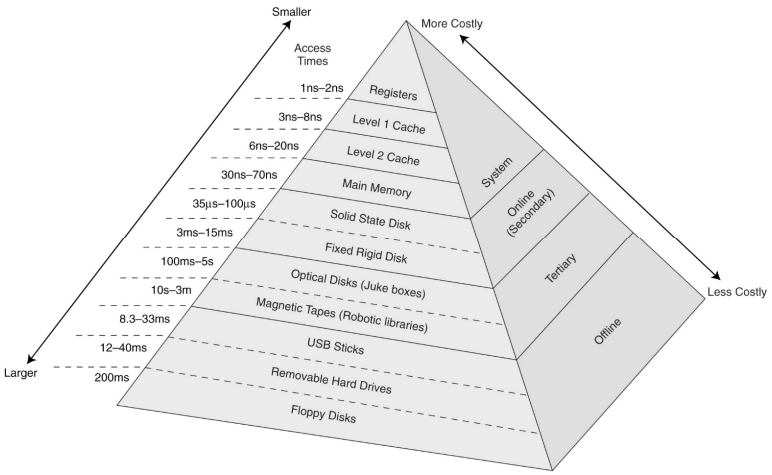


- Before we go into the memory type details, let's go through some basics on the function of a memory in a computer system.
- When we look at memory from functional point of view, there are two types of memory in a computer system.
    - System and Storage memory
        - System memory is where you run the code directly from. But it is likely that system memory will not be large enough to contain all the application program and data you have, as such you will need a larger space for these information.
        - This is where Storage memory comes in. The entire collection of Program and Data is stored in the Storage memory.
        - Another common attribute of storage memory is that it is typically non-volatile as it needs to preserve the program and data even after the computer power down.
        - For this course, where system memory is concern, we only have three memory types to select from: NOR Flash, DRAM and SRAM, the primary reason behind is that these three memories support XIP, meaning you can run code directly from these memory. No worries if you are not familiar with these names and terms, we will get to more details in later slides.
    - The processor itself will also have its own internal memory and other memory elements such as Cache and Registers.
    - Operation wise, the computer will power up with its OS and applications resided in the Storage memory, these are then transferred over to the System

memory so that the processor can execute the code. The various internal memory elements may also be used.

Memory – Cost vs Function Trade Off

- The memory and storage devices may be organized like a pyramid.
- 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.



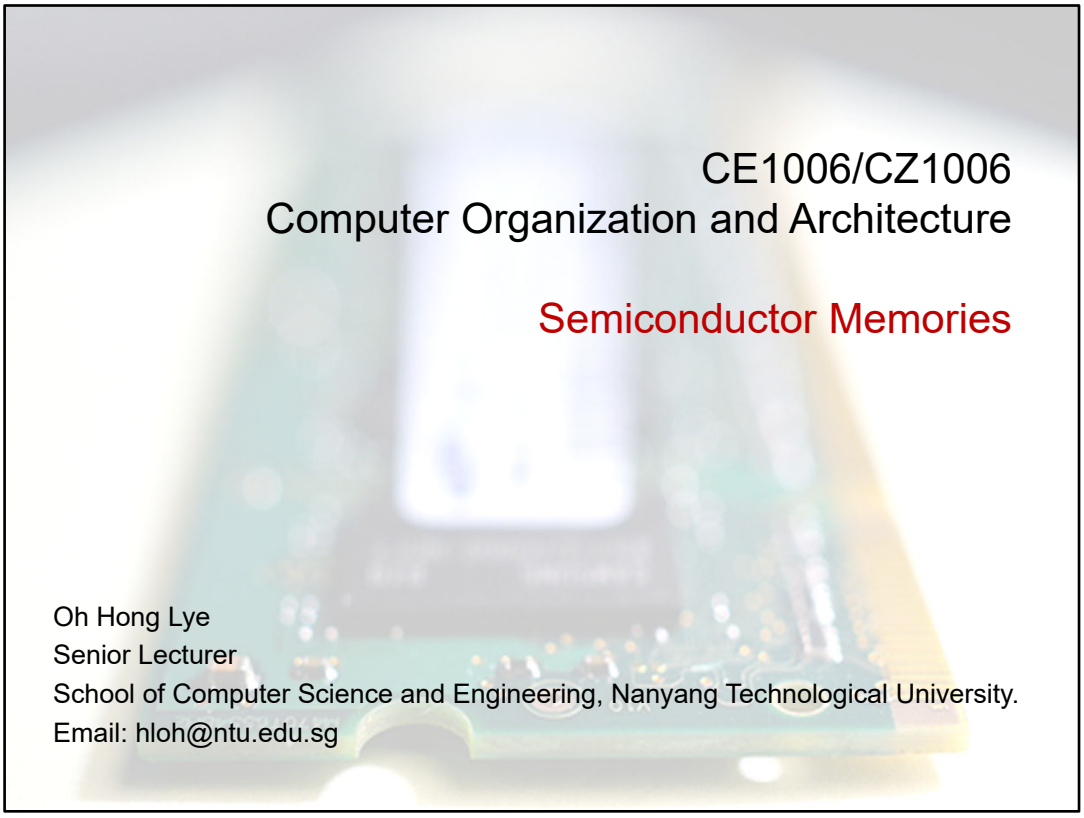
- When choosing which type of memory to use, we typically have to consider the cost vs function trade off.
- IN this slide, different memory types are organised in the form of a pyramid.
    - The fastest memories are at the top but they are also the most expensive so their size are typically smaller
      as compared to those at the bottom.
- Note that the memory access timing are for illustration only, these changes with improvement in technology.

Volatile and Non-Volatile Memory

- 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.

Oh Hong Lye / Cx1106 5

- Back to our discussion on Volatile and Non-Volatile memory
- Volatile memories has the attribute that
  - The data will be lost when power supply to the memory is cut off.
  - So the information can only last if there is power and is therefore used as a temporary storage.
  - System memory are typically implemented using volatile memories. For this course, we will look at SRAM and DRAM
- Non-volatile memory, on the other hand, is able to preserve the data even when the power is cut.
  - So it is used as a permanent storage. We will look at Flash and Magnetic Hard Disk in our course.



- The first group of memories we are going to check out is the Semiconductor memories.

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)

- Semiconductor memories are based on semiconductor IC, short for integrated circuits.
- They are used as processor internal memory elements such as registers, buffers, cache and internal system or storage memory.
- Types of semiconductor memories we will touch on in this course are SRAM, DRAM, Flash Memory and SSD, which are bulk storage memory mainly consist of Flash memory ICs.

VOLATILE MEMORY

This section is on Volatile memories, which are memories that lose its data if the power is shut down.

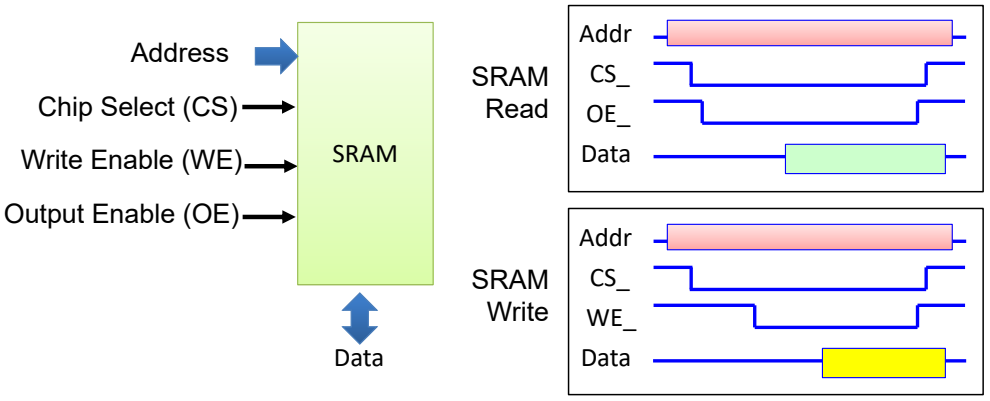
Random Access Memory (RAM)

- 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.

- In this course, we will look at two types of volatile memory, SRAM and DRAM.
- Static Random Access Memory, SRAM in short
  - Is able to retain its data integrity as long as power supply is maintained.
  - It has a relatively larger layout consisting of 4-6 transistors per cell, typically.
  - It access time is fast
  - And power consumption for both active and standby mode is low.
  - All the above are wrt DRAM
- Dynamic RAM, DRAM in short is another type of volatile memory you will frequently encounter.
  - It consist of a capacitor and one transistor.
  - This give rise to a smaller layout compared to SRAM, cost per bit of DRAM is therefore lower than SRAM
  - The use of capacitor as a storage element, however, requires the cell to be refreshed periodically in order to maintain data integrity.
  - That leads to a higher power consumption and slower access time when compared to SRAM.

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)

Oh Hong Lye / Cx1106 3

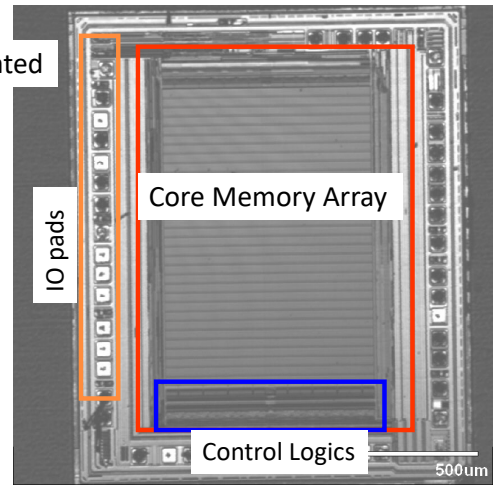
- To perform read and write operation on the SRAM, we need the following signal lines.
  - Address, which specify which memory location will be read or written
  - Chip select which is the main switch for the SRAM chip
  - The Write Enable pin which tells the SRAM that the processor is performing a write operation
  - The Output Enable pin which allows the SRAM to output data from its data bus.
  - And lastly the data bus.
- During a SRAM read, the uP will first output the address info, followed by the CS and OE. The SRAM will then output the data requested on the data bus.
- I'm showing these signaling sequentially to walk you thru the whole process. In actual case, these signaling happen simultaneously.
- Similarly for SRAM write, uP wil first output the Address, followed by the CS and WE so that SRAM can prepare for a write operation. uP will then output the data on the data bus and SRAM will latch the information in.
- And again, these signals are output simultaneously.

SRAM Chip DeCap

Commercial SRAM Chip
Cypress CY62128, 1 Mbit

Decapsulated

Decapsulated silicon die
Laser scanned image, 5x magnification

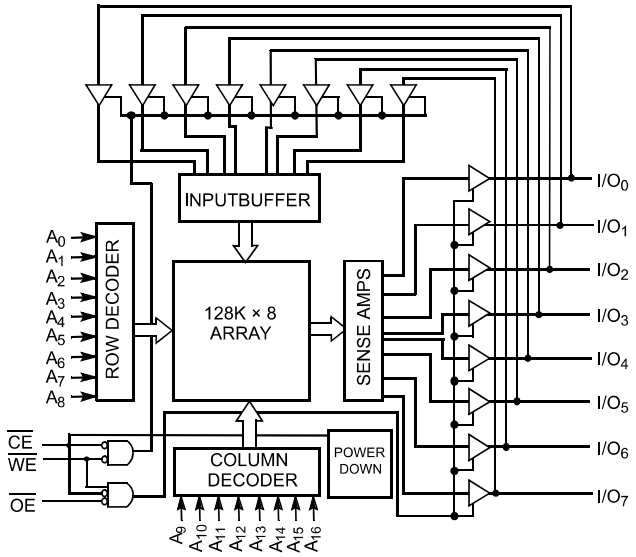


- **Memory** array takes up **most** of the **real estate**.
- Leading Edge Semiconductor Process is usually first tested with Memory Design.

Oh Hong Lye / Cx1106 4

- Next, let's dig deeper into the SRAM chip.
- This show what it looks like under the packaging. You can see rows and rows of uniform design, which is actually the SRAM cells.
- In most system or uP, memory usually takes up the most real estate.
- As such, they are usually the target candidate to test the leading edge semiconductor process technology.
- Which is why leading memory makers such as Samsung are usually the leaders in semiconductor process technology.
- The other group of companies that leads in the process technology are the processor makers such as Intel and AMD.

SRAM Internal Circuitry

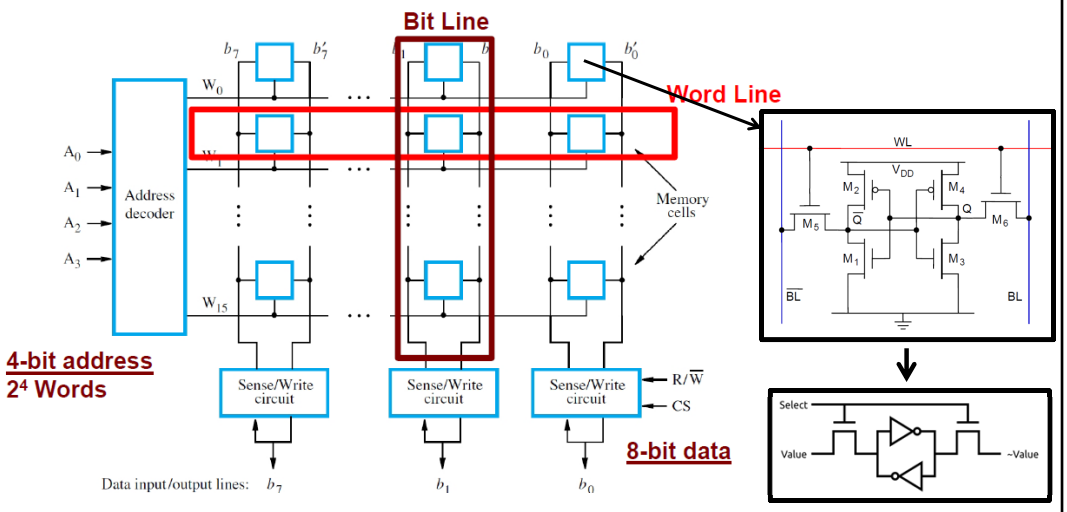


- Memory array
- 1M SRAM cells for the chip shown.
- Control circuitry
- Decoders
- Sense amplifiers
- Input/Output multiplexers

- This is how SRAM internal looks like functionally.
- The example shown here is a 128KByte SRAM chip
  - It consist of an array of 1 Mega (2^20) SRAM memory cells, each cell contains 1 bit.
  - The address are fed into decoders which will enable 8 of the cells so that read or write operation can be carried out.
- For SRAM, you don't have to be too concern about the detail design such as the role of decoders, multiplexers and sense amplifiers, its not in the syllabus.

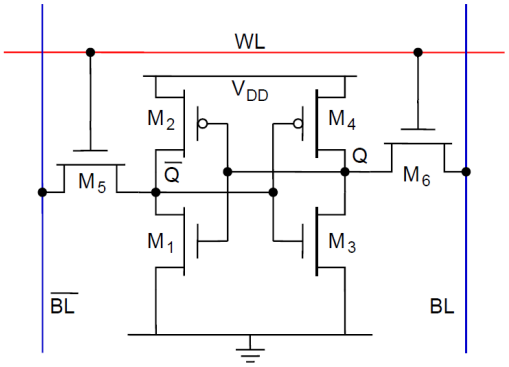
SRAM Cell

- 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).



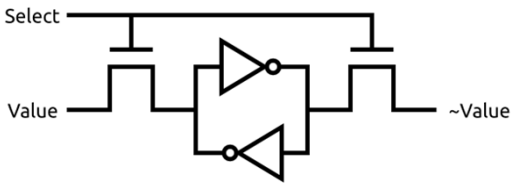
- Diving in further, this is how the SRAM cells are arranged.
- The address info is decoded to enable a particular word of data, each word consist of one row of cells, each cell corresponding to one bit of data.
- Each SRAM cell has six transistors. It has a differential output.
- The digital logic equivalence of the transistor design is shown here.
  - Its basically two inverters with output of one connected to the input of the other, and vice versa.
  - This ensure that the logic state that is stored will remain unchanged as long as the gates are powered.
- I do not expect you to know the detail transistor level layout design but you should at least know the function of the transistors and the logic equivalent of the 6-transistor design.

SRAM Cell



- 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.

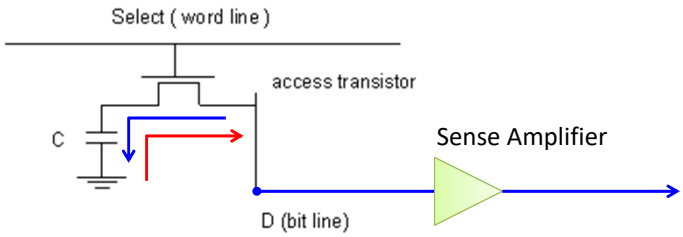
- The data bit is stored in M1, M2, M3 and M4 (which is equivalent to two invertors connected as shown on the right).
- M5 and M6 are pass transistors.



Oh Hong Lye / Cx1106 7

- As mentioned in the previous slide, SRAM cell consist of 6 transistors.
- The word line is derived from the address info and controls which sets of bit cells are enabled
- The bit line output the actual data.
- M1 to M4 forms the invertor logic we mentioned earlier.
- M5 and M6 controls the data in and 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.

- Next we are going to look at Dynamic RAM. DRAM in short.
- 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.
- First, let's look at the Write Process
  - To store a logic '1', we first enable the pass transistor and transfer some charge to the capacitor to charge it up.
  - If we want to write a logic '0' instead, then we would need to enable the pass transistor and discharge the capacitor.
- Similarly, for the Read Process
  - The pass transistor has to be enabled and the capacitor charge status is detected using a sense amplifier, in order to determine the logic stored in the DRAM cell.

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.

- DRAM is called Dynamic RAM because the charges stored in the DRAM cell tends to change, this compared to the SRAM,
  which can hold its charges indefinitely as long as there is power supplied.
- Let's take a look at this behaviour in more detail.
- In DRAM, a read process involves discharging the capacitor so in a way, the data stored is destroyed when performing the read operation.
    - That's why the original data has to be re-written after every read.
- On top of that, charges in a capacitor tend to leak with time. Hence, DRAM cells requires periodic refresh operation to hold the data.
- Note that the basic DRAM is more or less obsolete in the market today. It is replaced by the synchronous version called
  Synchronous DRAM, or SDRAM in short.
    - But basic concepts and features of the DRAM remains, such as the use of capacitor as main storage element and the need to have period refresh to maintain data integrity.
- The difference between SDRAM and DRAM is that
    - SDRAM use a clock signal from host to synchronise the data transfer
    - SDRAM also employs a pipeline architecture that allow faster, overlapping operations compared to conventional DRAM.
- The SDRAM variants you see today are enhancement over the single rate SDRAM. Known as DDR2, DDR3, DDR4 etc which is able to sustain higher data transfer rate.
- That conclude the discussion on volatile memory.

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.

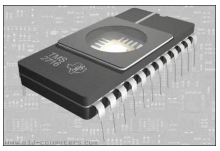
- DRAM is called Dynamic RAM because the charges stored in the DRAM cell tends to change, this versus the SRAM,
  which can hold its charges indefinitely as long as there is power supplied.
- Let take a look at this behaviour in more detail.
- In DRAM, a read process involves discharging the capacitor so in a way, the data stored is destroyed in the process.
    - That's why the original data has to be re-written after every read. This process is called pre-charging.
- Also, charges in a capacitor tend to leak with time. Hence, DRAM requires periodic refresh operation to hold the data.
- The basic DRAM is more or less obsolete in the market today. Being replaced by its synchronous version called
  Synchronous DRAM, or SDRAM in short.
- Other enhancement

NON-VOLATILE MEMORY

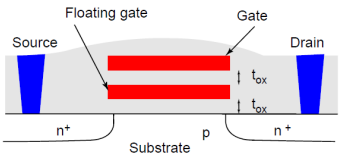
- This section is on non-volatile memory.
- These memory is able to retain the data even when its power is cut off.

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



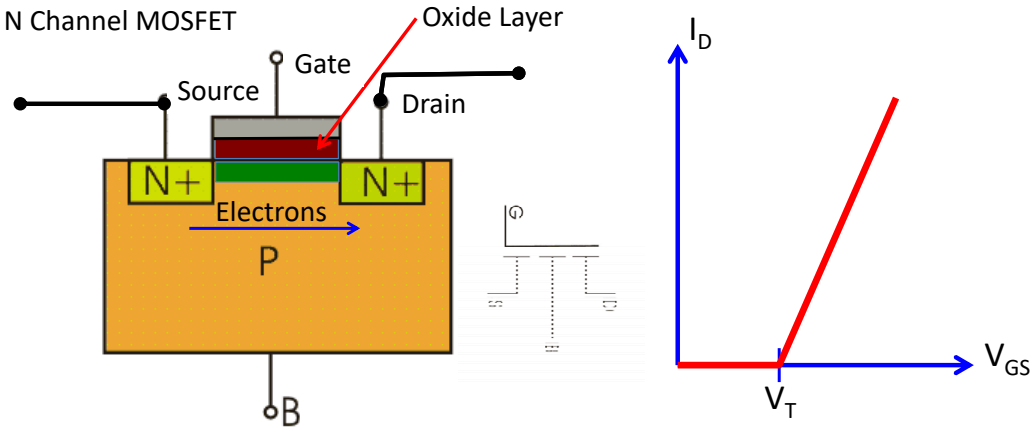
Device cross-section

- 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).

Oh Hong Lye / Cx1106 2

- Two common non-volatile memories you can find in the market are EPROM and EEPROM
  - Which are short form for
    - Erasable Programmable ROM and
    - Electrically Erasable PROM respectively
- EPROM is one of the earliest erasable non-volatile memory that uses a floating gate design.
  - This device can be programmed electrically but requires UV light to erase its data.
  - The packaging has a small transparent window that allows UV to get to the floating gate to remove the charges trapped there.
- As technology advances, we are able to reduce the thickness of the gate oxide.
- This led to the creation of EEPROM, which allows the charges to be erased electrically.

MOSFET

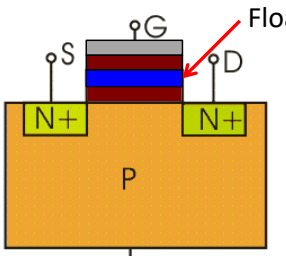


- MOSFET (Metal-Oxide-Semiconductor Field Effect Transistor)
- For N-Channel MOSFET, if Gate-Source Voltage (Vgs) > Threshold Voltage (Vt), a conductive channel of electrons (inversion layer) will be formed and current will flow if a positive voltage is applied across Drain and Source.

Oh Hong Lye / Cx1106 3

- We mentioned a number of times this term called floating gate in the previous slide.
- So what is a floating gate?
- To know that, let's start by looking at a MOSFET, or Metal-Oxide-Semicon Field Effect transistor.
- For a N-channel MOSFET, it remains non-conductive until the Gate-Source voltage exceeds a certain positive voltage threshold.
- When the gate voltage is sufficiently positive, it will be able to attract more electrons which are negatively charged.
- A layer of electrons will be formed under the oxide and current will flow between the Source and Drain terminal of the MOSFET, the electron layer is illustrated in this slide by a green bar below the oxide layer.
- You can see from the graph that current is zero till Vgs exceed Vt. Afterwhich current will flow.

Floating Gate Transistor (FGT)

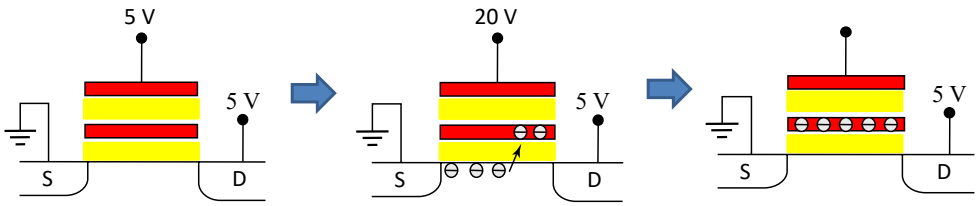


Floating Gate

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

Oh Hong Lye / Cx1106 4

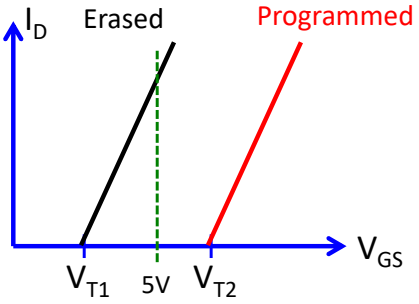
- Now that you know how a MOSFET works, let's take a look at the floating gate transistor, FGT in short.
- In the FGT, an additional layer of the metal gate and the oxide is inserted. So the gate that is sandwiched between the two oxide layers appears to be floating as its not electrically connected to anywhere.
- There are two states that the FGT may be in.
- One is the erased state.
    - This correspond to the state where there are no excess electrons in the FGT, i.e. it is electrically neutral.
- Then there is the Programmed State
    - Which is the state when there are excess electrons in the FGT.
- 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 voltage of the FGT and this attribute can be used to implement memory bit cell with non-volatile property. More details in the next slide.

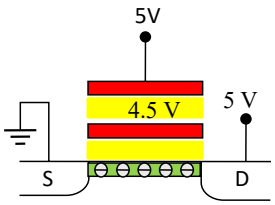
FGT Threshold Voltage

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.

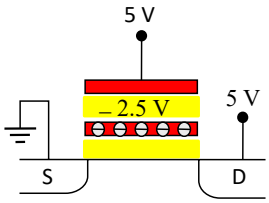
"Programming" increases the threshold voltage of Floating Gate Transistor so the same 5V is now unable to turn ON the FGT.

Voltage values used are for illustration purpose only.





Erased State



Programmed State

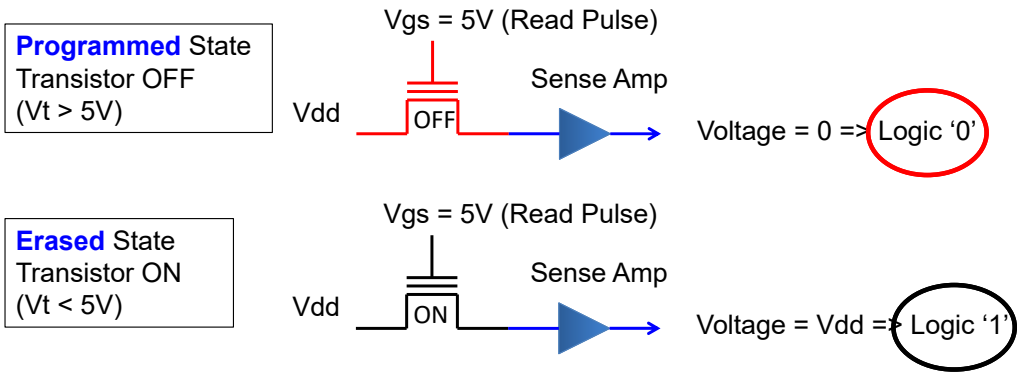
Oh Hong Lye / Cx1106 5

- First, let's look at how the presence of excess electrons affects the threshold voltage of the FGT.
- Remember that a MOSFET becomes conductive when the gate source voltage or Vgs is larger than a certain threshold voltage Vt.
- Similarly, for FGT, It'll be conductive when its gate voltage crosses a certain threshold.
    - As an illustration, let's say that when the FGT is in erased state, i.e. the floating gate has no excess electrons, a 5V input at the gate is able to attract enough electrons to form the conductive layer. So the FGT is turned ON.
- Now, 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.
- So to summarise, FGT in erase state has a lower Vt than when it is in programmed state.
- This attribute of the FGT can be used to implement a memory element.
    - We have previously discussed about the way to detect whether a FGT is in erased or programmed state.
        - That can be done by injecting a voltage between the two threshold, VT1 and VT2 in the graph.

        - If the FGT in ON, its in erased state, if it is OFF, it is in programmed state.
    - If we further defined that the FGT stores a Logic ‘1’ and Logic ‘0’ when it is in erased and programmed state respectively,
        - then we have a 1-bit memory cell in the form of the FGT.
            - To write a ‘1’, we erase the FGT
            - To write a ‘0’, we programmed the FGT.
            - To read the content, we apply a 5V.
- Note again the 5V here and other values are for illustration purpose only.
- This concept forms the basis of many of the non-volatile memory discussed in this course, e.g. EPROM, EEPROM and Flash.

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.

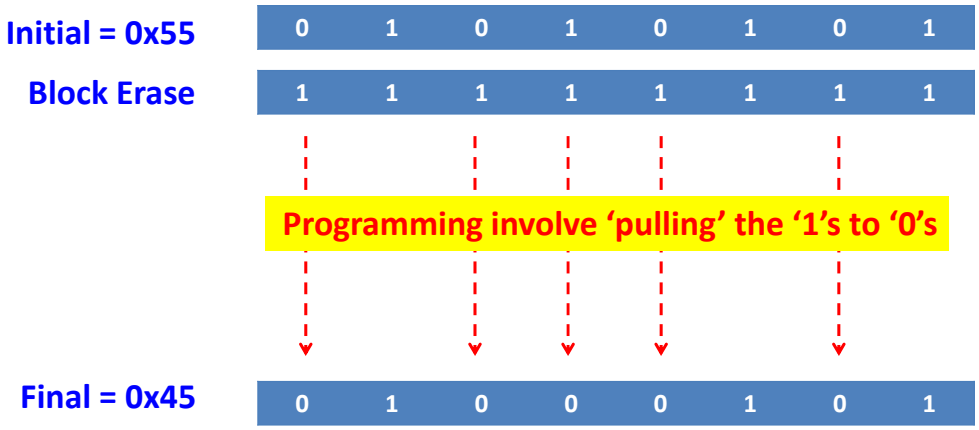


Oh Hong Lye / Cx1106 6

- Just to recap from the discussion in previous slide,
  - With a positive gate voltage, e.g. 5V, you find that FGT will remain OFF if it is in the programmed state.
  - Since FGT is off, the sense amplifier will detect a logic '0'.
- But if the FGT is in erased state, the threshold voltage will be lower and the FGT will be turned ON, leading to sensing of logic '1' at the output.
- Which is why, after a flash is erased, it'll contain all '1's in its memory. During programming, we are actually storing '0's into the various memory bits.
- Note that the 5V value here is just for illustration, actual Vt value may change depending on the doping of the transistors.

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'



Oh Hong Lye / Cx1106 7

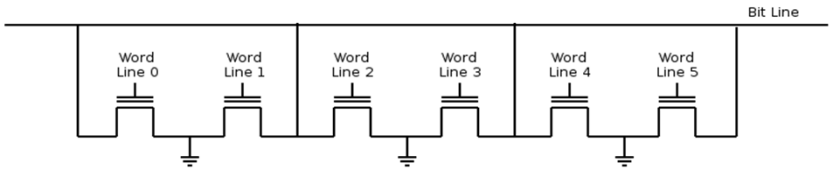
- This slide runs through the process of programming a number into a FGT based memory such as Flash memory.
- Programming in FGT based memory is really modifying the cell content from '1' to '0'.
- If you need to write a '1' instead, that correspond to the erase operation.
- In FGT based memory, erasure are done at block level, where one block consist of multiple bytes.
- For example, if we want to write a value '0x45' to a particular memory location whose initial value was '0x55' in a Flash memory, we first have a erased the block in which the byte location is.
- This will set all the bits to '1's.
- After which, the actual programming involves setting to some of the '1's to '0's to form the value 0x45.

Flash

- Based on similar floating gate technology as EEPROM.
- 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

- One of the most popular non volatile memory based on FGT is the Flash Memory.
- Flash memory can be erased in larger block size compared to EEPROM
- Since erase operation is typically slower than other operations, Flash memory has a faster access speed compared to EEPROM, especially when writing large blocks of data.
- Flash memory also cost less since each overhead module for erasing or programming cater for a larger block of memory.
- There are two main types of Flash Memory in the market today, namely the NOR and NAND flash. We will discuss in more details in later slides.

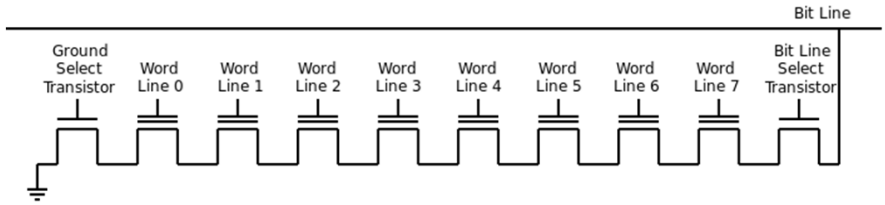
NOR Flash



- 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

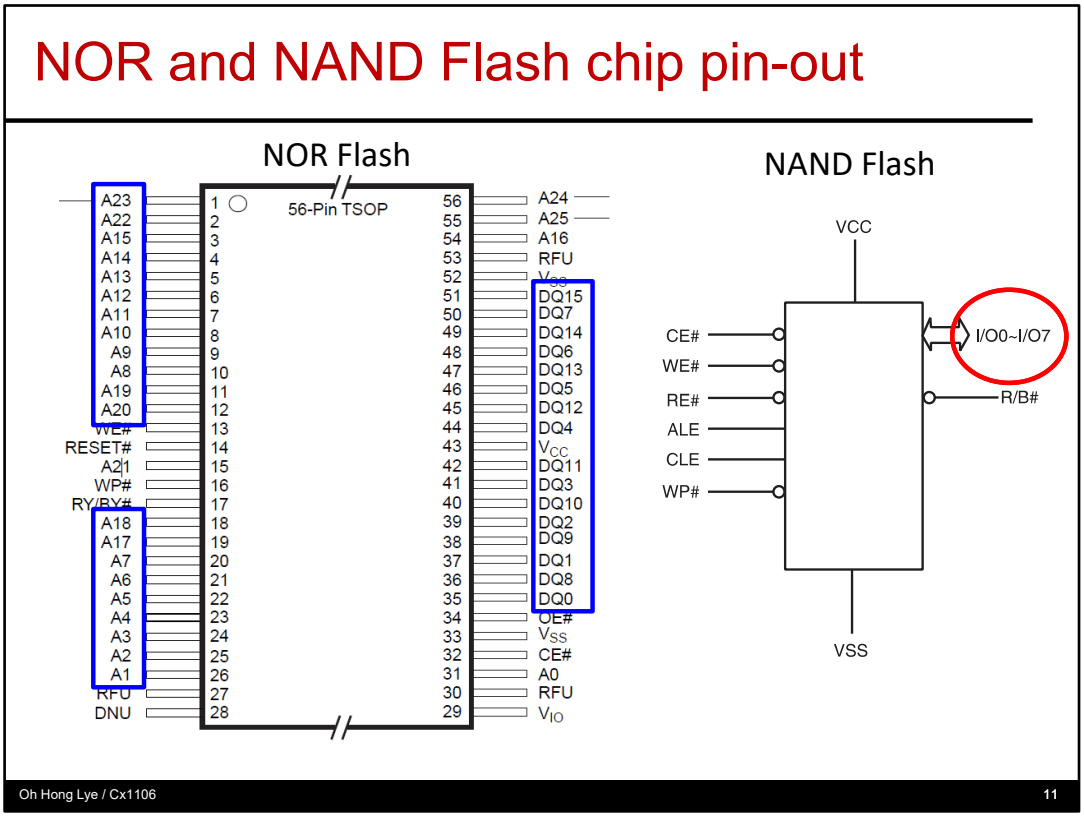
- In NOR flash, the individual cell are connected with logics similar to that of the NOR gates.
- Meaning that if any of the wordline is higher than the threshold voltage, the bit line output will be LOW.
  - Similar to the NOR gate property that if any of the input is a '1', then output will be a '0'.
- NOR flash supports Execture-In-Place, XIP in short.
- This means you can store your program code in the NOR flash and execute the instruction directly from there without having to transfer the code to RAM.
  - This is possible because the NOR flash allows random reading of its data using only Address information.
  - Recall that in a processor, it only uses the Program Counter as the pointer to access instruction code, no other commands are involved.
- Note that the previous discussion refers only to NOR Read operation.
- For other operations such as write operation, procedure is more complicated and will require special commands and multiple steps operations.
- Note that even though NOR flash allows random word or byte programming, erase operation needs to be done at block level.
- Lastly, its ability to support XIP enable NOR Flash to be used as the system memory of a product. It could still be as a storage memory if user wants to.

NAND Flash



- 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

- Similarly, Nand flash got its name from the fact that the behaviour of its design has a lot similarilty with the property of a NAND gate.
  - The Bit line output is '0' only when all Word Line input > Vt.
- Unlike NOR Flash, NAND flash does not support XIP.
  - Data is accessed at page level. Each page consist of multiple bytes of data, multiple pages formed a block.
  - NAND flash uses a single IO bus to transfer Command, Address and Data information.
    - It's a multi-step process where the Processor will first need to send a COMMAND to the NAND Flash telling the memory what operation will be carried out, e.g. Read. It then followed up transferring the Address information and lastly the target data will be transferred.
    - As seen, the process is much more complicated does not support XIP.
- NAND flash's layout, however, has less connecting wires and therefore can be compact to a higher density, compared to NOR Flash.
  - This lower cost per bit advantage result in NAND flash being the memory of choice for mass storage devices.
  - Examples include USB Flash Drive, SSD, Phones and Tablets.



- This slide illustrate the difference between the NOR and NAND Flash interface.
- NOR Flash has a very simple interface consisting of Address and Data lines. The processor provide the Address of the target location and the NOR Flash supplies the data.
- NAND Flash, on the other hand, has only one IO bus so all the COMMANDS, ADDRESS and DATA information have to go through this bus.

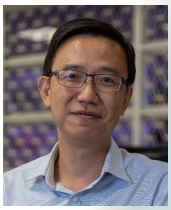
System vs Storage Memory

- 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

- This is the last slide of the semiconductor memory chapter.
- I'll use this slide to recap the key points for System and Storage Memory.
- System memory
  - Store runtime program code and data that the processor use.
    - Note that the code is executed directly from system memory.
  - System memory could be internal or external memory.
    - For this course, there are three memory types that supports XIP and can be used to as a system memory, namely, SRAM, DRAM and NOR Flash.
    - DRAM, technically speaking does not support XIP as its access is also multi-steps, but typically processor that can be connected to a DRAM will have a DRAM controller module that handles the necessary translation to allow executing code directly from DRAM.
- Storage memory, on the other hand, stores all the program code/data in a computer system.
  - However, the processor do not run code directly from the storage memory, these needs to be transferred to the system memory for execution.
  - Technically speaking, all memory types can be used as storage memory.
  - If you need the storage memory to be able to retain information when the power is cut off, then non-volatile memory has to be used.
- That comes to the end of the semiconductor memory chapter.
  - We will touch on the HDD and SDD in next video.

Cx1106
Computer Organization and Architecture

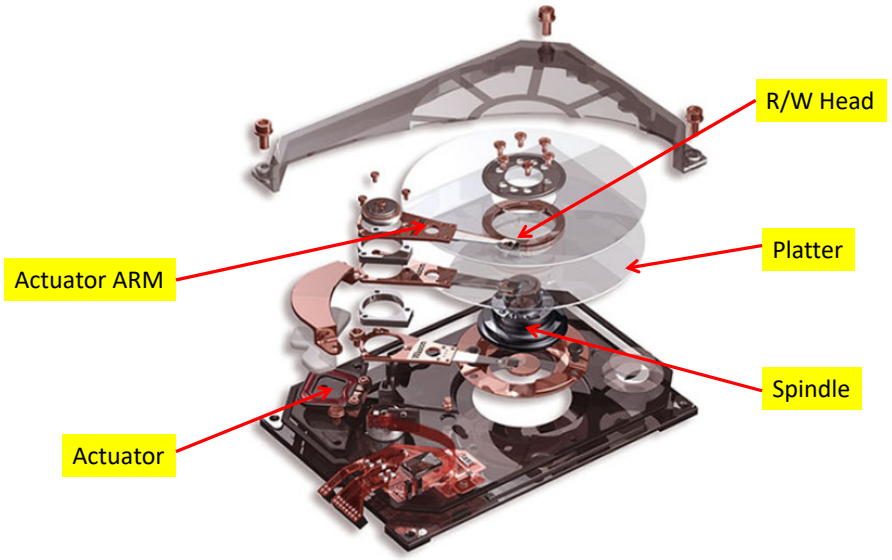
HDD and SSD



Oh Hong Lye
Senior Lecturer
School of Computer Science and Engineering, Nanyang Technological University.
Email: hloh@ntu.edu.sg

- This chapter is on the two main storage devices for bulk storage, the Magnetic Hard disk drives (HDD) and the Solid State Drives (SSD).

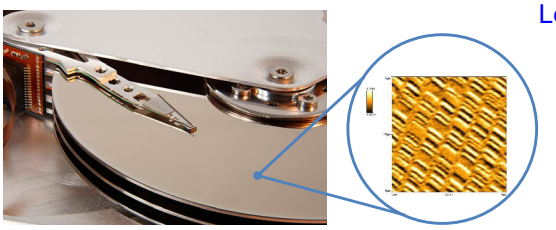
Magnetic Hard Disk



Source: Backblaze

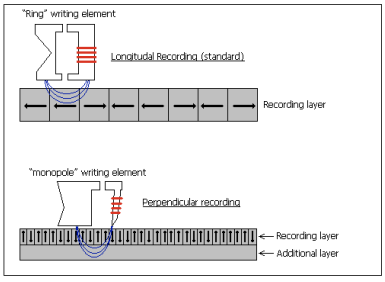
- This diagram shows an exploded view of the hard disk internal.
  - The Actuator ARM which carries the R/W head around different parts of the HDD media
  - The Actuator is an assembly of strong magnets and electrical windings. The resultant force produced when electric current flow through the wires is used to propel the actuator ARM.
  - The R/W head which picks up the changes in magnetic field on the media and converts these fluctuation into data bits.
  - The Platter which is the storage media. Each Platter consist of two surfaces, top and bottom surfaces. These day, data is stored on both surfaces.
  - Lastly, the spindle which is a motor used to spin the platter at very high speed of up to 15K rpm.

Magnetic Hard Disk

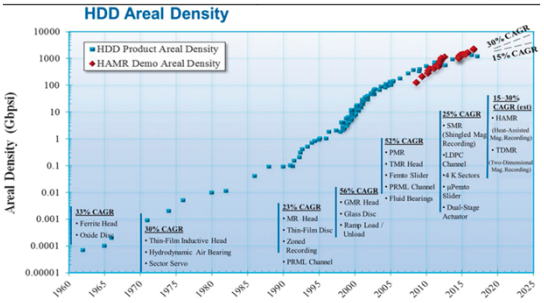


[Source] http://en.wikipedia.org/wiki/Disk_read-and-write_head

Longitudinal vs Perpendicular Recording



[Source] Robert Fontana et al, IBM Areal Density Comparison Paper, 2010

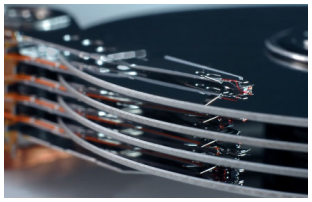


Source: Cambridge

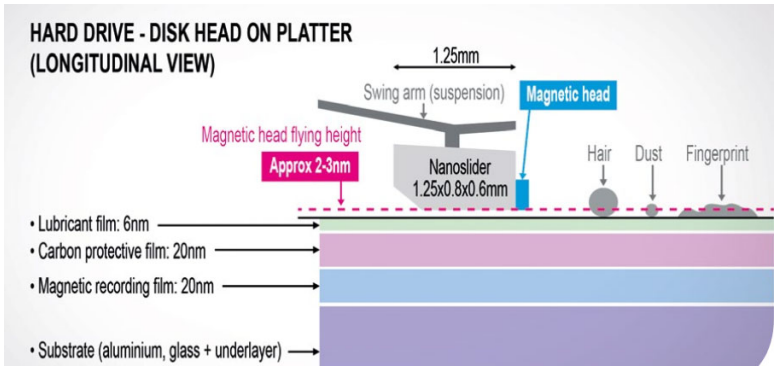
- Stores data by magnetizing a thin film of ferromagnetic media on the circular disk known as platter.
- Video on Introduction to Hard Disk. https://www.youtube.com/watch?v=kdmLvl1n82U

- Magnetic hdd is still the main mass storage for Data Centers, Servers, Desktop PC. However, it is fast being replaced by semiconductor memories such as SSD in the notebook products.
- Magnetic HDD stores data by changing the orientation of the tiny magnetic elements in the circular disc platter.
- There are Two types of recording method: Longitudinal and the more advanced perpendicular recording.
    - Difference between these two types of recording is in the orientation of the magnetic elements: as the name implies, longitudinal is along the surface of the platter while the other is perpendicular to the surface (which offers higher data storage density)
    - The different orientation of the magnetic element will be interpreted as ones and zeroes by the RW head.
- On the bottom left corner, we have a plot of the HDD areal density vs time. The graph shows how hard the HDD maker is working to enable a continuous improvement in HDD technology. Which is essential in enabling HDD to continue to be the memory element used in huge storage use cases such as the Data Centers.

HDD Technology



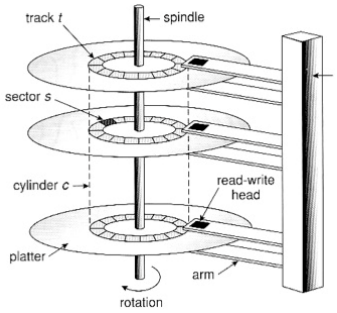
- The magnetic head uses the wind pressure generated by the high speed rotation of the disk to fly above the disk surface, similar to the principles employed by the aircraft.
- Fly height is lower than a finger print mark.



Source: datarecoverydublin

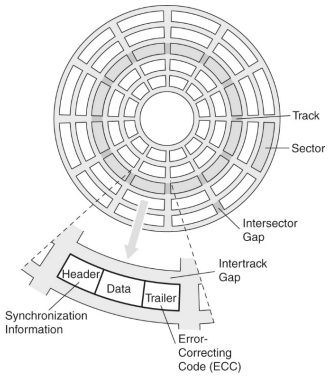
- Flying the Magnetic head is often compared to flying a jumbo jet.
- On the web, you'll find many different figures when people made this comparison. This is because it was done at different timeline, the technology they reference to is therefore different. And since HDD technology changed so quickly, you'll find that the number quoted differ by a large amount.
- There is one source quoting that flying the RW head is like flying the jumbo jet a few mm above the ground.
- With the actual height the RW head flies, even a finger print mark is taller than the fly height and will crash the RW head.
- Bottomline, it shows the huge amount of technology and precision that has gone into the tiny HDD.

Magnetic HDD Data Organization



- 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.

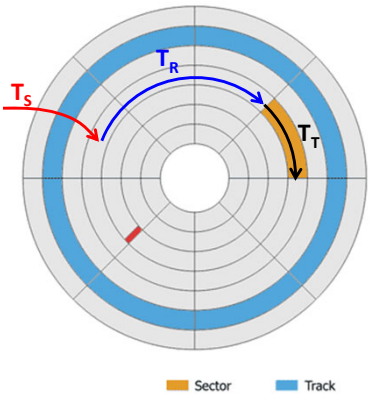
- Data is stored on the surface of the platter in concentric rings called tracks.
  - Gaps between tracks to minimize interferences from adjacent tracks.
- Tracks are divided into sectors.
  - Minimum data block size is a sector.
- Stored data consists header, data and trailer.



- Now let's go thru some definition of parts in detail.
- HDD stores its data in the circular disk known as platters.
- The reading and writing of info go thru the read/write head. There is one head to each surface of the platter.
  - RW head is attached to an actuator arm whose motion is initiated by electromagnetic forces.
- In terms of data organisation, the smallest container is known as a sector.
  - Multiple sectors will form a track, which is basically a circular track around the disc.
- Within each sector, the data is formatted to provide synchronisation info, actual data payload and error correction code.

HDD Transfer Rate

- 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.



Oh Hong Lye / Cx1106 6

- Now for some definition of timing parameters used in HDD performance measurement.
- First the Seek Time
  - This is the time it takes for the ARM to move from its resting position to the track that contains the target sector to be read.
    - The HDD vendor will typically supply a parameter known as the Track-to-Track seek time, since each movement from one track to another really takes different time, the parameter supplied by the HDD vendor is the average seek time to move from one track to another. That is, it is a statistical average value based on some test cases.
- Then we have the Rotational Delay
  - Which is the time it takes to move to the start of the targeted sector, after the RW head reaches the correct track.
- Adding the Seek Time and Rotational Delay will give you the Access Time.
- Lastly, we have the Transfer Time, which is the time it takes to read the target data after the RW head reached the start of the sector.

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.

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

Oh Hong Lye / Cx1106 7

- Rotational delay depends on the rotation speed. The faster the rotation speed, the shorter the rotational delay.
- Assuming we start from any random section, then the following formula will give the average rotational delay.
- So how did we derive this formula?
    - Since the RW head could land on any sector when it reaches the correct track,
        - Under the best case scenario, the RW head is just above the target sector, which means we have zero delay.
        - Under the worst case scenario, the RW head had just past the target sector, which means it need to wait for a full revolution of the disk before it can read the data. In this case the delay is equal to time it take to rotate one round, which is 1/RPS.
        - So if we take the average, it'll be 0.5/RPS, the formula given in the slide.
- For the formula on Data Transfer Time,
    - The assumption here is that the RW head will be able to read the data once it past over the corresponding sectors, so transfer time is equal to the time needed to transverse the targeted sectors.
    - Dt*Ds gives the total number of bytes per track and N is the number of bytes to read.
        - So N/(Dt*Ds) gives you the ratio of data needed vs total data in one track.

- Since the angular velocity of the disk is a constant, this ratio is also the ratio of the time it takes to transverse the targeted data vs the time it take to revolve one round, which is 1/RPS.
- That is how we end up with the expression shown in the slide for Tt.

HDD Transfer Rate 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?

[Solution]

RPS = 15000/60 = 250 per second

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

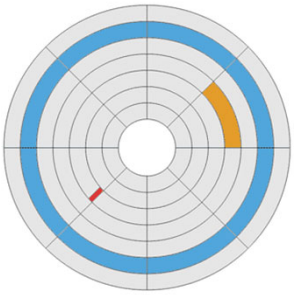
Transfer time T_T = 3072/(250*500*512) = 48 µs

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

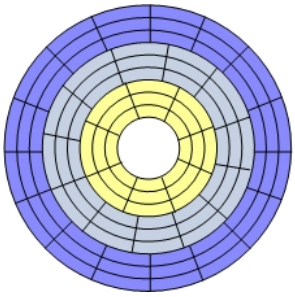
Notice $T_A >> T_T$. This example shows that accessing file whose data is distributed across sectors on different tracks on the magnetic hard disk would potentially incur more time.

- Let's look at one work example to apply what we have learnt so far.
  - With a HDD having the parameters shown in the slide, what is the total time taken of read a 3KByte file stored in consecutive sectors?
- From the previous slide, we know that the total time taken is equal to Ta+Tt.
  - Where Ta = Ts +Tr, the seek time and the rotational delay
  - Tt is obtained using the expression in the previous slide.
  - With that, we have a final value of 6.048ms
- One point, notice that Ta>>Tt in this case.
- What this means is that if the data of one file is spread across different sectors in different track, then you'll find that the effective HDD transfer rate will be much lower compared to the case where the data are all within the same track. This is because every time the RW head completes reading one sector, it has to move to the start of the next sector which is located in a different track, and that is going to take another Ta duration.
- Not sure if you people still do disk defragmentation or not, but that is a process where you get the computer to organise the data so that data from the same file is allocate to consecutive sectors/track. Effect is that the HDD transfer rate should increase compared to before defragmentation.

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.

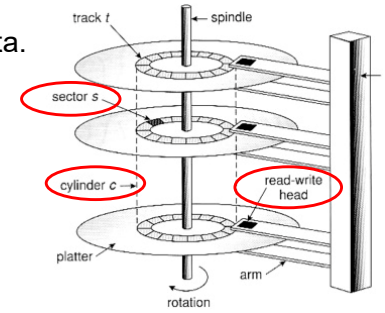


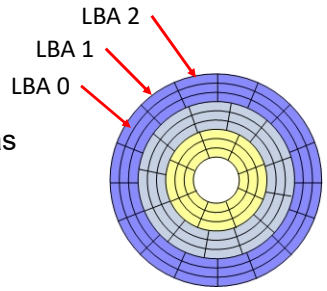
Oh Hong Lye / Cx1106 9

- Physical layout refers to how data is organized and formatted on the HDD media.
- In the past,
  - Every track has equal number of sectors and each sector contain the same amount of data.
  - This constant sector per track type of layout is similar to what you saw in earlier slides when we do the access time computation.
  - Early HDD design used this type of formatting for both logical and physical layout to simplify the controller design.
- But the physical layout of the HDD today has adopted a slightly different format.
  - Primary reason is that the old method of formatting results in a lot of space wastage since the sector length on the outer track are much larger than those in the inner tracks, meaning they could potentially store more data.
  - Hence HDD today uses the zone bit recording formatting. Where the sector length is kept approximately the same to fully utilize the recording media.
  - Outcome of this is that the number of sectors on the outer track is now more than those on the inner tracks.

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.

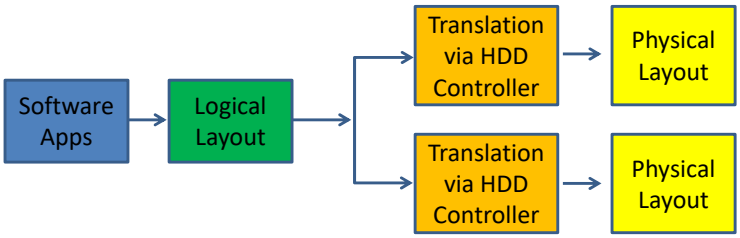




Oh Hong Lye / Cx1106 10

- Logical layout and the corresponding addressing, is what your program code used when accessing the HDD.
- There are two types of logical layout addressing used,
  - CHS, which stands for Cylinder-Head-Sector, and
  - LBA, short form for Logical Block Addressing.
- CHS is the legacy scheme that uses the old constant sector per track layout and formatting convention i.e. Cylinder, Head and Sector.
  - This scheme is more or less obsolete these days due to limit in the HDD capacity it can support.
- CHS is replaced by the LBA, which uses a simple linear addressing scheme.
  - As shown in the diagram, the first block in the HDD is given the address LBA0, next block = LBA1 and so on until the last block of the HDD.
  - 48 bits is allocated to specify the LBA number so this scheme have a lot of headroom, it can address up till 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.

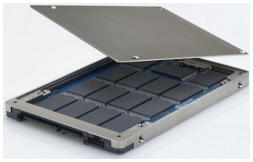
- So why do we need two different layout format?
- If only physical layout is available, a piece of software has to be tweaked everytime it needs to interface to a new HDD.
    - Because there is not standard way of designing a physical layout.
    - In fact, the physical layout of HDD from different vendors will be different, and is a way these vendors could differentiate their products.
- So in order to resolve this issue, we need a standard way for a software program to talk to the HDD, and that is where logical layout comes in.
- With a standard such as LBA, the software only needs to communicate according to this standard.
- Whenever a HDD vendor develop a new product, they will need to ensure that proper translation of logical to physical layout can be done via their HDD controller.

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

- If you buy any HDD these days, you will realise that not much information of the HDD is provided beyond the capacity of the HDD, RPM valaue and the Average Data Transfer Rate.
- The detail physical layout information is not given because of the complexity of Zone Bit Recording, giving detail physical layout information will cause more confusion than useful information.
- The HDD these days employ use of Cache and Buffers to speed up the data transfer.
- However, the concepts and limitation we learn here is still applicable
  - The mechanics of the HDD and its associated limitation remains, e.g. the actuator ARM is fixed and access has to be sequential.
  - Seeking from track to track consume more time
    - So it is more efficient to read/write in blocks rather than random sectors
    - Which means HDD defragmentation is still applicable and helps to improve performance
  - And HDD, due to its mechanical nature, is vulnerable to motion.

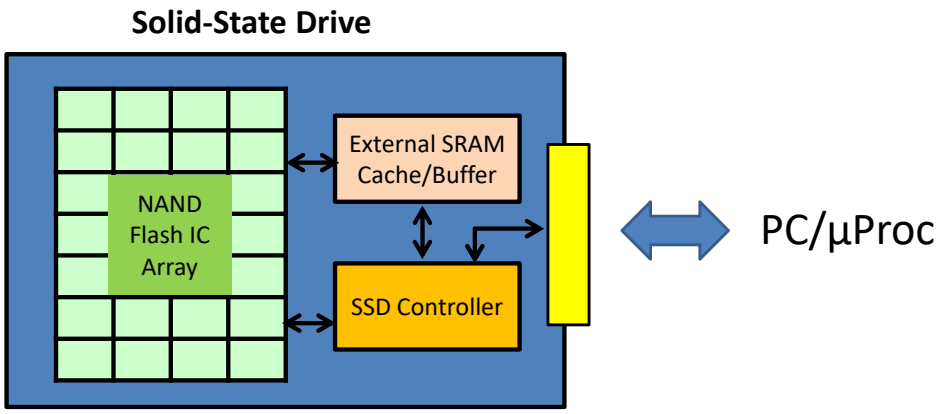
Solid State Drive (SSD)




- 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.

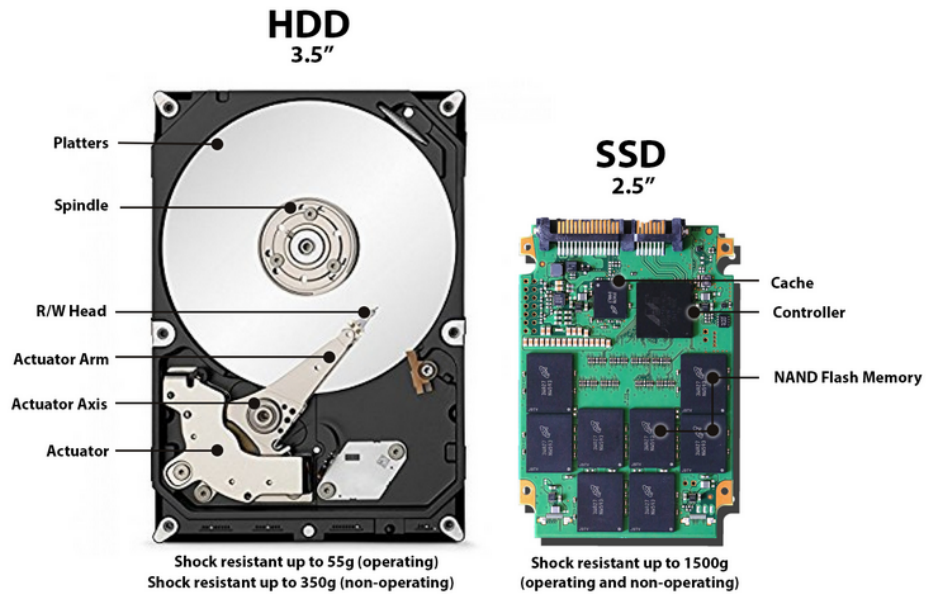
- Next we come to the solid state drives.
- These drive are getting more and more popular as they are more robust to handling, since they don't have any moving parts compared to magnetic HDD. And also, primarily because the cost of NAND flash memory, which is the main memory used in SSD, has dropped drastically over the last few years.
- It is, however still more expensive than magnetic HDD so many of the computers, especially the desktops and servers, are still using magnetic HDD. HDD is also still the main storage element found in Data Centers.
- One of the main issue with SSD is the limited program-erase cycles of flash memory that it uses.
  - What it means is that there is only limited number of times user can perform erase operation on a particular memory block location.
- But there are many techniques to extend the life of SSD
  - Wear levelling which spreads out the write/erase operation evenly over the entire disk
  - Using external RAM as buffer to reduce the number of memory access to Flash memory.
- With these enhancement and mitigation, the mean time to failure benchmark of SSD these days are comparable to magnetic HDD.

SSD System Block Diagram



- This slides shows the system block diagram of a SSD.
- The main storage elements are NAND flash.
- There is a SSD controller which a basically a microprocessor that manages the operation of the SSD, this includes
  - Addressing translation between Logical to Physical Layout
  - Caching and buffering of data in and out of the SSD
  - Wear levelling algorithm
- There are external SRAM and Cache to improve the transfer rate for both read and write operation
- So in a way, SSD itself is a computer system.

HDD vs SSD



- This is a photo comparison between a HDD and a SSD.
- SSD form factor can be made smaller actually, such as the mSATA and NVME form factor which is much smaller than the legacy 2.5inch HDD form factor.
- The earlier SDD adopted the 2.5 inch form factor so that they can be a drop in replacement for the 2.5inch HDD found on notebook computers.

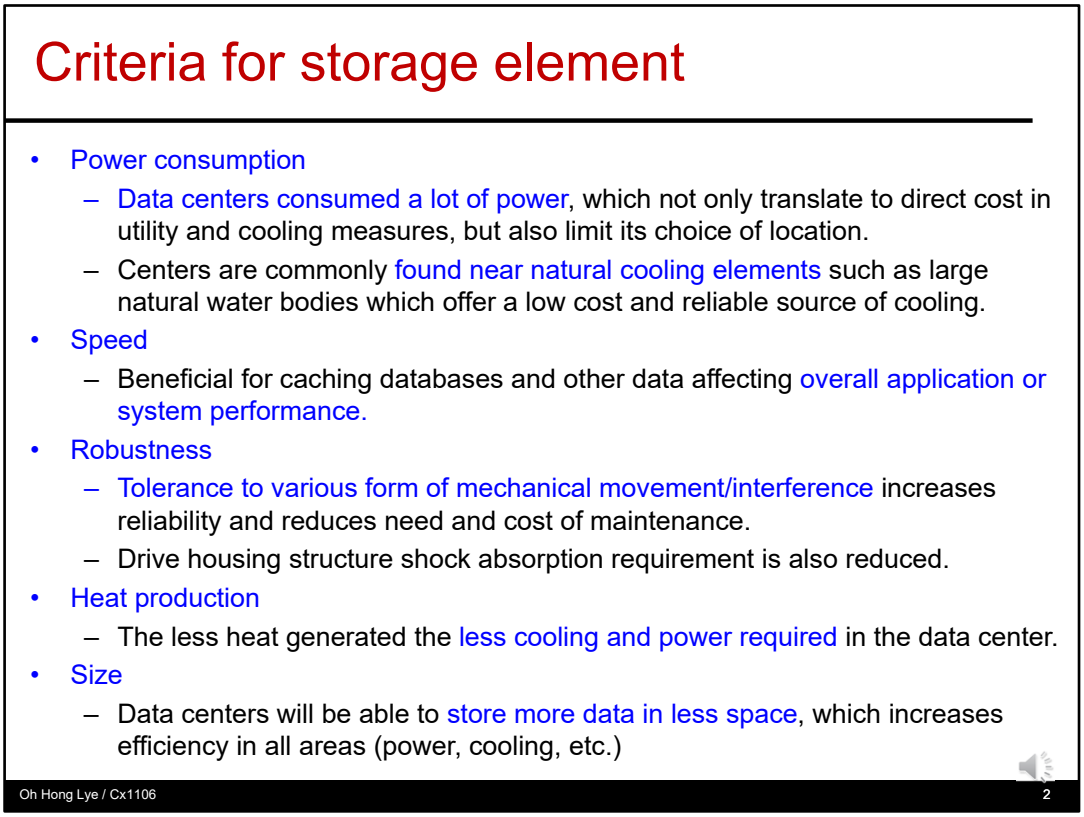
HDD vs SSD

- HDD
  - 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
- 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

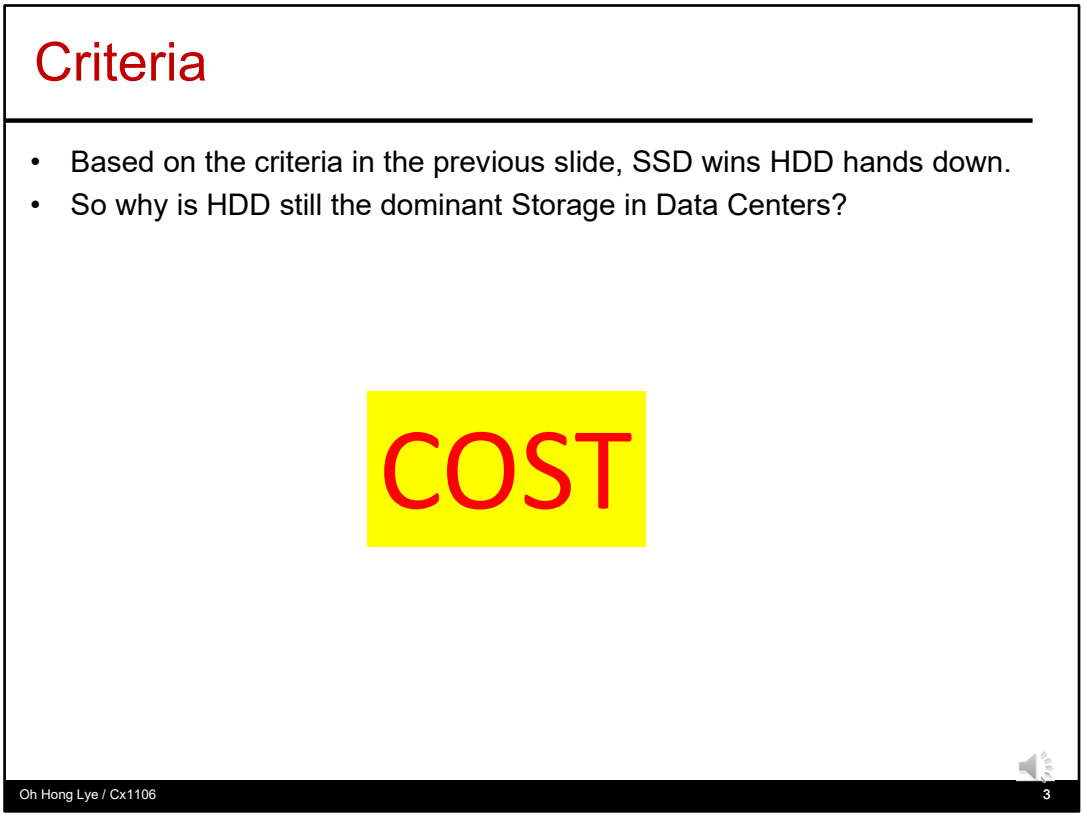
- Last slide of the HDD/SSD video.
- Advantage of the HDD
  - It has lower cost per bit and
  - Almost infinite erasure cycles.
- Disadvantage of HDD
  - There are many moving parts in HDD, it is therefore prone to crashing if there are huge movements
  - Its form factor are larger and more bulky
  - And has a slower transfer rate.
- Advantage of SSD
  - Opposite of HDD, there are no moving part so it is very robust against movement.
  - It is very light, can be made really small and
  - Has a higher transfer rate
- Disadvantages of SSD
  - Cost. Its more expensive cost per bit wise when compared with HDD
  - There is a limit to the number of erasure cycles user can apply on the SSD.
- We have come to the end of this video, next video will be on Data Center related information.

DATA CENTER STORAGE

- In this section, we will touch on some HDD specific considerations under the context of Data Center Operation.

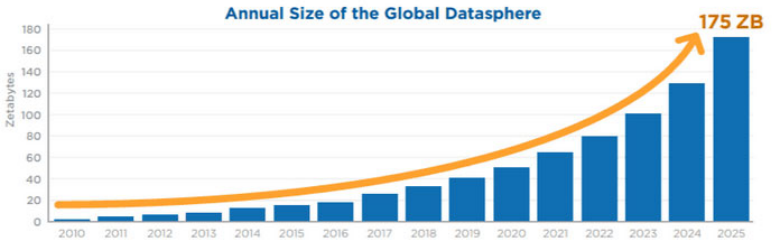


- This slide shows some of the key consideration when selecting storage elements to use in a Data Center.
- First is the Power Consumption
    - Data Center uses many storage elements so power consumption of these storage elements is very important.
    - Its translate to direct cost of electrical bill and also the indirect cost of requirement for cooling measures.
    - The need for cooling measures may also limit the choice of Data Center Locations, with many of the Centers located natural cooling sources such as natural water bodies.
- Next is the speed factor
    - Having storage element with higher speed will improve the overall performance of the system hosted in the Data Centers. Data Centers these days are increasingly being used to host virtual machines and other processing related functions, rather than just a simple cloud storage.
- The storage element needs to be robust as well too since reliability is one of the most important attribute of a Data Center.
    - One of the desired attributes are tolerance of various form of mechanical movement and interference. This will improve the reliability and reduces the need and cost for maintenance as the storage element is less likely to fail.
- Cooling measures, as mentioned above, incur cost and limits the location choices. So having low power consumption will be very beneficial.
- Lastly, would be good for the size to be small to be able to pack into a smaller building.

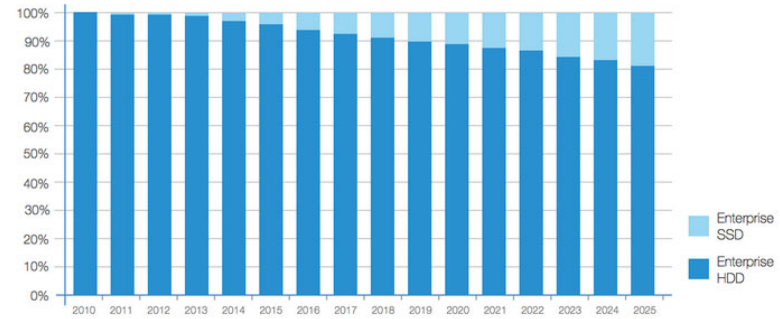


- Based on the selection criteria discussed in the previous slide, the obvious choice of storage element is the SSD.
    - But HDD is still the dominant storage for Data Center because of one most important reason – COST.
- The cost per bit of SSD today is still higher than that of HDD, this translate to huge cost difference due to the huge storage size of a Data Center.

Global Storage Consumption and Shipment Projection



1 ZB = 10^{21} bytes

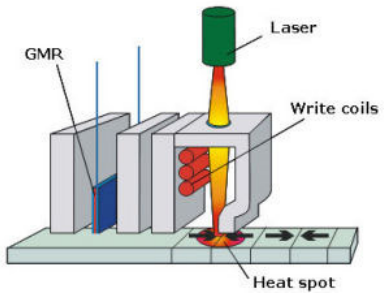


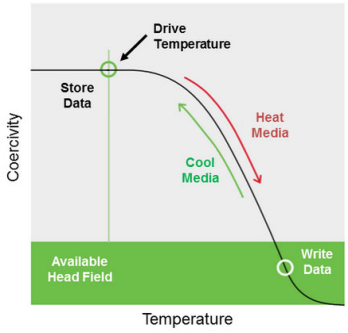
Source: IDC, Seagate.

Oh Hong Lye / Cx1106 4

- Other the Data Centers, other use case than requires huge storage also see HDD as the preferred choice, such as in the enterprise servers.
- As seen in the chart here, although SSD is slowing creeping into the market share of HDD, the total size of shipment for HDD is still much larger than that of SSD.

HAMR (Heat Assisted Magnetic Recording)





- 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.

Oh Hong Lye / Cx1106 5

- Cost is the single most important reason why HDD is still surviving to this day, so naturally, the HDD maker is always looking at new technologies to increase their cost advantage.
- One of these technologies is the HAMR, which stands for Heat Assisted Magnetic Recording.
    - There was a time where the area density of HDD was stagnant for a while and manufacturers had been shipping drives which has 2TB/platter.
- But with HAMR, the areal density is expected to increase again.
    - The recording media used in HAMR drive allows more data to be packed into the same area reliably, but it is also more difficult to change the orientation of the magnetic element on these media.
    - So for HAMR, a smaller laser is attached to the recording head.
    - The laser is designed to heat a tiny spot on the disk, increasing the temperature allows data to be written more easily to the disk.
- Current projection are that the HAMR drives can achieve 5Tbpsi, enabling drive with capacities higher than 100TB.

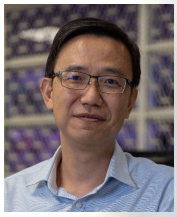
HAMR

- 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.

- This slides describe the principle between the HAMR drives.
- One main problem of increasing the areal density of a HDD media is that the data bits will become unstable, meaning it has higher chance of flipping from 1 to 0 or vice versa.
- To make the media more stable, the recoding media needs to have high coercivity.
    - Higher coercivity means the magnetic media is more stable but it also means it is more difficult to change its magnetic characteristics, i.e. more difficult to write to the media.
    - To counter that, a laser is used to heat up the spot where the head is writing to, this increases the temperature and lower the coercivity of that surrounding area. The reduced coercivity allow data write to be carried out properly.
    - After the write operation, the laser is turned off, the area cools down quickly and regain its coercivity and stability.
    - You can revisit the graph in the previous slide for illustration of this process.

Cx1106
Computer Organization and Architecture

Cache



Oh Hong Lye
Senior Lecturer
School of Computer Science and Engineering, Nanyang Technological University.
Email: hloh@ntu.edu.sg

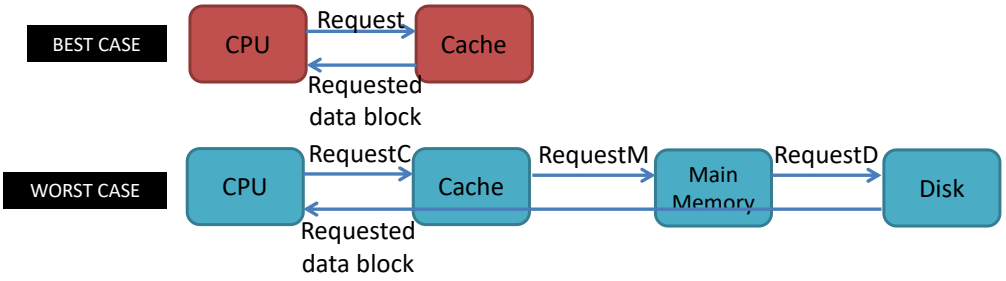
- This chapter is on Cache Memory Management. This is a module in the processor that is able to improve the overall system performance significantly by leveraging on the principle of locality exhibit by most application program. More details in this video.

Cache

- 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.

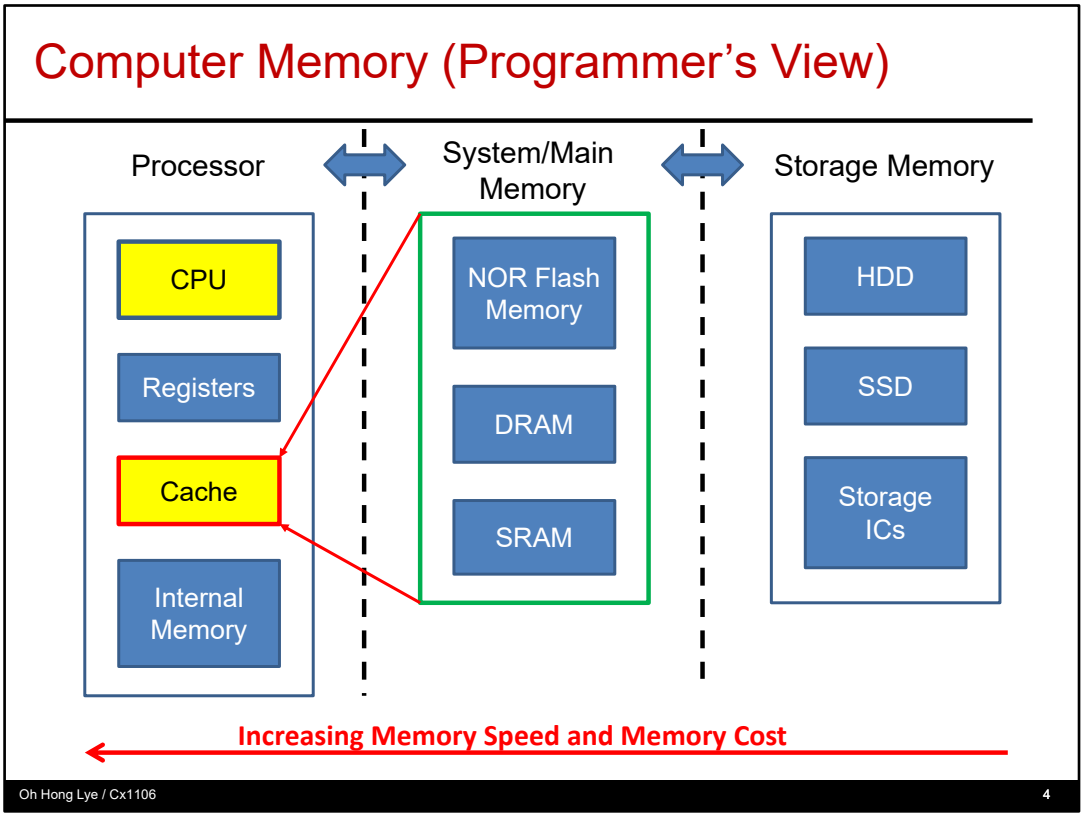
- Now, CPU typically run at a higher speed compared to the external memory.
  - So the external memory becomes the bottleneck and slows down the system performance.
- To increase the overall system performance, we need a fast memory to act as a buffer between the CPU and External Memory.
  - And that piece of fast memory is known as Cache.
- We'll see later that a well designed cache could potentially improved the system performance significantly.

CPU Memory Access



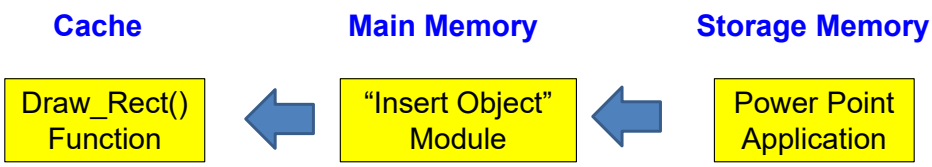
- To access a particular piece of data, the CPU first sends a request to its nearest memory, usually cache.
- If the data is not in cache, a query is then sent to the main memory.
- If the data is not in main memory, then the request goes to disk.
- Once the data is located, the **required data and a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

- This slide shows you what really happen between the CPU, cache and main memory.
- In a processor with cache, the CPU will first visit the cache to see if it has the data the CPU needs.
    - If the data is there, then the requested data is transfer and the processor process with other operations.
    - If data is not in the cache, then CPU will need to look in the main memory or the HDD for the data.
        - Once data is found, it'll be fetched into the cache, together with some other nearby data elements.
- How the main memory data is stored in the cache depends on the mapping scheme used. And this is what we'll be going through in the next few slides.



- Let's re-visit a previous slide on computer memory from the perspective of a programmer.
- A program's code and data is stored entirely in the Storage memory.
- The specific sections of code and data is transferred to the System memory during runtime.
- A subset of the code and data which is recently or frequently used is stored in the cache for fast access by the CPU.
    - Different subset of code/data will be loaded to the cache as the program execute
        - The objective is to have the subset that CPU need most at that instance to minimise the time need to retrieve these information.
- Just to recap, CPU will first visit the cache to look for the information it needs, it will proceed to the System memory if it can't find the required information, and further check out the storage memory if it can't find the information in the System memory.
- You will see the system memory being reference by the name "main memory" in the rest of chapter. In this course, system memory and main memory are referring to the same piece of memory.

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.

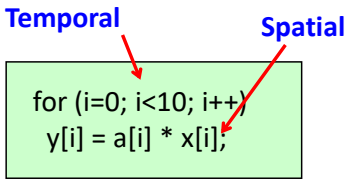
- This slide gives you an illustration of what happen in a Processor with cache when running a power point application.
- The entire powerpoint application is too large to be stored in the main memory so it is stored in the storage memory instead.
- User will only be using one of the many features so only those needed is loaded to the main memory.
  - In this case, it's the insert object module that is loaded.
- Even within the insert object module, user will only be using a sub-set of what the module provide
  - In the case, the user is trying to draw a rectangle.
  - The particular function that he needs will eventually find its way into the cache since it is the most recently and frequently used group of code.

Principle of Locality

- Now that we know cache only needs to contain a subset of the main memory to work
- In what way should the data be transferred between main memory and cache in order to maximise the efficiency of a cache?
- Efficiency meaning how much improvement can the cache brings to the system. This is related to the probability the CPU is able to find the information that it needs in the cache.
- For that, we need to understand the attributes of how information is organised and accessed when a program is executed.
- This is summarised in the concept called Principle of Locality

- From previous slides, we know that CPU only needs a subset of the entire program during runtime.
- Which is why cache size can be very small compared to system and storage memory.
- So the next question we need to answer is what data should be stored in the cache in order to maximise the resultant system performance?
    - To maximise the system performance, we need to maximise the cache efficiency.
    - The efficiency here is directly related to the probability of CPU finding what it needs in the cache.
- To enable that, we need to understand the attributes of how code and data is organised and accessed when a program is executed.
- This is summarised in the concept called principle of locality.

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

Oh Hong Lye / Cx1106 7

- When a byte of code or data is access within a program, it is very likely that its nearby code or data will be needed soon.
  - This attribute is known as the principle of locality
- There are two types of locality
  - Locality of space and Locality of time
  - Also known as Spatial and Temporal Locality
- Spatial locality refers to the scenario where the code/data that are located near to each other are likely to be accessed together
  - There leads to the cache design where if a particular byte required by the CPU is not in the cache, then the cache module will transfer that byte together with its neighbours to the cache. That is, any transfer between the cache and main memory always happen in blocks and not single byte. This is so that subsequent access will result in cache hit. Cache hit means CPU is able to find the required information in the cache.
- Temporal locality, on the other hand, refers to the scenario where the recently accessed code/data is likely to be accessed again.
  - This sort of justify why a cache will be effective even though its size is small.
  - It is also one of the considerations used when designing the cache replacement policy
- The small for loop shown here illustrate the two principles.
  - The for loop had 10 iterations, with each iteration performing a sum of product between array a and x, and storing the result to array y.
  - The sequential access of the arrays illustrate the concept of spatial locality.
    - When one element of an array is used, the chances of subsequent

elements being access is high.

- The 10 iterations implies the code for the sum of product will be executed 10 times, an illustration of temporal locality in operation.

Cache Memory Replacement Policy (need to move to after mapping)

- When there is a need to transfer a block of data to the cache but the cache is fully occupied, there is a need to decide which cache block to evict/purge in order to free up space for the new data.
- The algorithm used to decide which block gets evicted is designed based on some Cache Replacement Policy. Two examples illustrated below
- **Least recently used (LRU)** algorithm keeps track of the last time that a block was assessed and evicts the block that has been unused for the longest period of time. The disadvantage of this approach is its complexity: LRU has to maintain an access history for each block, which ultimately slows down the cache.
- **First-in, first-out (FIFO)** is a popular cache replacement policy. In FIFO, the block that has been in the cache the longest, regardless of when it was last used.

- The other part of cache storage design deals with replacement policy.
- While the direct mapped cache doesn't really need a cache replacement policy as its mapping is one-to-one, its good to understand the general concept behind replacement policy. Set associate or fully associative cache, which you will cover in Advanced Comp Arch module, will need cache replacement policy to decide which blocks gets replaced when the cache is full.
- In general, if the cache is full and a new block needs to be introduced into the cache, then the cache module will need to decide which cache block to evict to make space for the new block.
    - For Direct Mapped Cache, since the mapping is one-to-one, the block to evict is known before hand so there isn't a need to have any algorithm to manage the block replacement.
    - For Set Associate and Fully Associate Cache, which is not covered in this course, there is an option to choose from two or more blocks to evict. Hence, a replacement policy has to be put in place.
- Two of these methods are described here
    - Least Recently used algor. This scheme will evict the cache block that has not been used for the longest time. It's a very efficient scheme since the algor conform well to the locality principles. But implementation is more complex and costly as the cache needs to keep a access history for each block.
    - FIFO. Another popular scheme with simpler implementation. It basically replace the first block that comes into the cache. It conform approximately to the locality principles, just that it doesn't take into account of the scenario where a cache block is used multiple times. This typically result in lower

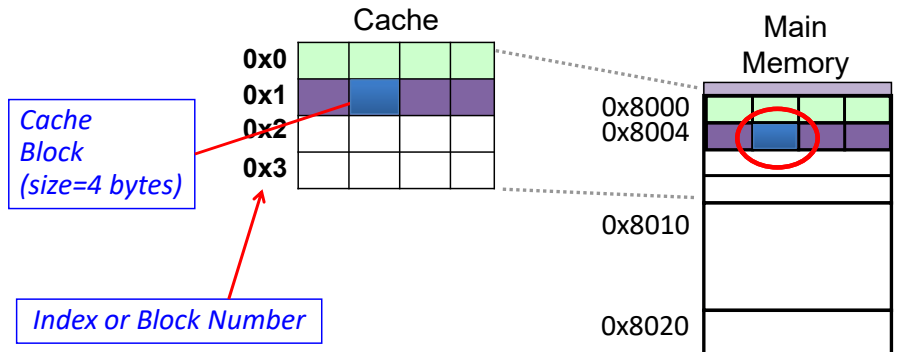
efficiency compared to LRU.

Cache Mapping Scheme

- We understand now that data transfer between main memory and cache is done in blocks instead of individual bytes to leverage on the principle of locality.
- Another attribute of cache design that affects the efficiency of the cache is the cache mapping scheme.
- Cache mapping scheme deals with how each main memory block is mapped to the a particular cache block, e.g. Main memory block #0x80 is mapped to cache block #0 (index 0).
- There are three basic cache mapping schemes
  - Direct Mapped
  - Set Associative
  - Fully Associative
- We will only touch on Direct Mapped Cache in this course, rest of the mapping scheme will be discussed in Advanced Computer Architecture Course.

- Another attribute of the cache design that will affect the efficiency is the cache mapping scheme
  - This deals with how each block in the main memory is mapped to the cache
- There are three basic types of mapping scheme
  - Direct Mapped
  - Set Associative and
  - Fully Associative
- For this course, we will only touch on the Direct Mapped Cache. The other two mapping scheme will be discussed in the Advanced Comp Arch module.

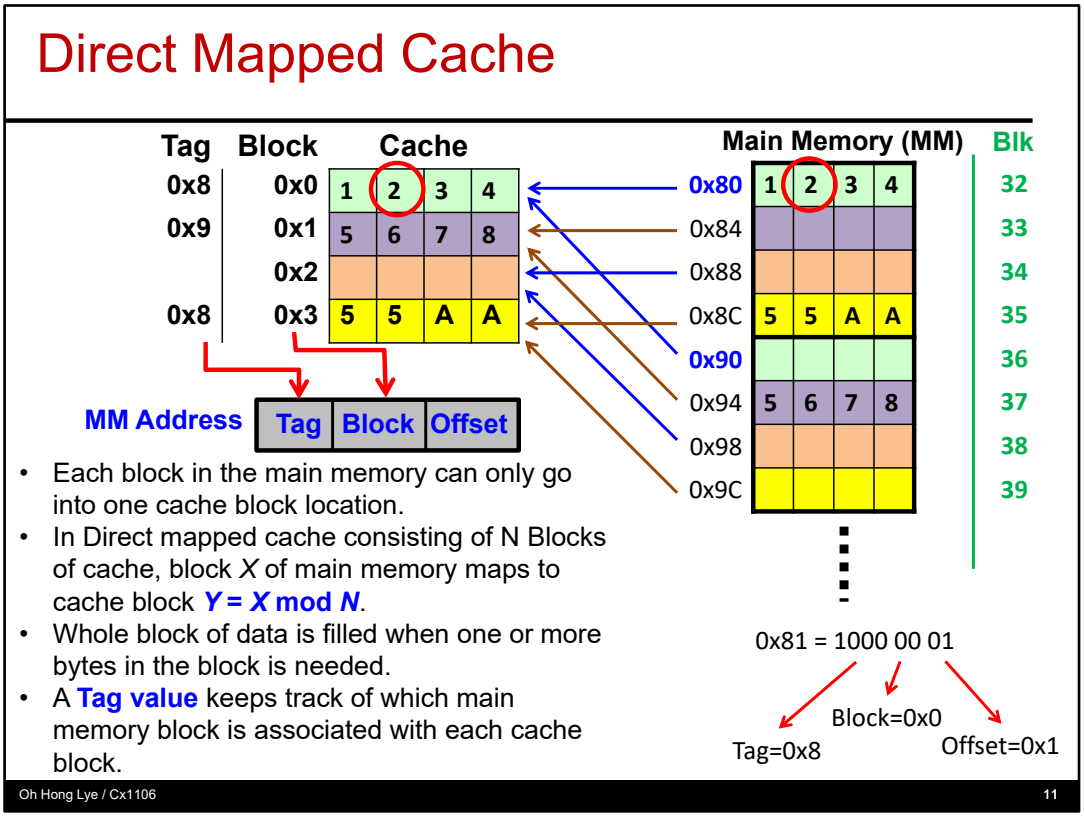
Terms used in Cache Mapping



- 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.

Oh Hong Lye / Cx1106 10

- Before we go into the details of cache mapping scheme, let's go through some basic terms that we will be using.
- First is the Cache Block or Cache Lines
  - Cache is divided into blocks consisting of multiple bytes
  - For this example, the cache block contains 4 bytes each.
- Each Cache block in the cache is numbered. This is known the cache block index or block number.
  - First cache block of the cache has a index 0.
- In cache analysis, the main memory is typically organised in the same format as the cache, i.e if the cache has a block size of 4 bytes, the main memory will also be arranged in blocks of 4 bytes each.
- When a particular byte in the main memory is needed, the entire block is transferred to the cache instead of the particular byte.
  - This as mentioned earlier, is to leverage on the principle of locality.



- For Direct mapped cache, each main memory block can only go into one specific block location in the cache.
- The way they are allocated is given by the formula X mod N, where X is the main memory block number and N is the total number of cache blocks available.
    - So MM block 0 goes to cache block 0, MM block 1 goes to cache block 1 and so on. MM Block 4 for this cache structure will wrap around and go to cache block 0.
    - In this example, MM BLK32 goes to Cache BLK 0, the mapping continue sequentially until MM BLK36 where it wraps around and get mapped to Cache BLK 0.
- As mentioned in the previous slides, although only one data byte is needed, the whole block of data is copied to the cache.
- Question here would be: How do we know which main memory block does the data in the cache correspond to?
    - For that, we need to look at the tag value, which is used to uniquely identify which MM block is in the cache.
        - For example, block starting 0x80 will be place in cache block 0 and have a tag value of 0x8, 0x8C will be placed in block 3 and have tag value of 0x8 as well.
        - 0x94, on the other hand, will be placed in block 1 and has a tag value 0x9.
- By now, you must be wondering how I derive these values.
    - The values are derived from the MM address.
    - Take for example 0x81.

        - The first step is to partition the MM address into three fields consisting of TAG, BLK and OFFSET.
        - To know how many bits the 'offset' field supposed to have, we need to understand the definition of the offset field.
            - An offset specify the position of the target byte from the start of the cache block.
            - For this example, the cache block size is 4, the offset field will consist of 2 bits as that is the number of bits needed to support 4 different address representation. For ease of calculation, you just need to use LOG2 of 4.
        - Next is the 'BLK' field. Similarly, this field describe which block the target data is transferred to, so if the cache contains 4 block, 2 bits is required for the 'BLK' field.
        - Lastly, the TAG field, which is used to uniquely identify each MM block, is derived from subtracting the bits allocated to offset and BLK. Fro this case, MM address range is 8 bits, so number of TAG bits = 8-2-2 = 4.
- After we partition the MM address according to the TAG, BLK and OFFSET field, we can easily determine which cache block this particular MM block will be transferred to by looking at the BLK field.
    - In this example, we can see that the MM block which contains 0x81 will be mapped to Cache Block 0 and the corresponding TAG value is 0x8.
    - We can also tell that the actual byte of interest is one byte away from the start of the cache block.

Cache Mapping (Elaboration on basic principle)

- 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.

- This slides elaborate on what we have learn so far regarding cache mapping scheme.
- Cache mapping involves the following tasks
  - Design a systematic way to map the main memory to a cache whose size is smaller than the main memory.
  - Allow the CPU to retrieve the information it needs from the cache, based on the MM address of the target information.
- The name 'target information' refers to the target code or data that the CPU needs.
- Since the CPU will use the MM address as the reference to retrieve the information, it is only natural that the MM address is used as the basis to enable the mapping process.

Cache Mapping (Elaboration on basic principle)

- 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.

- This slide contain detail description of each of the TAG, BLK and OFFSET field.
- The content has been discussed in previous slide so I will not elaborate further.
- You can pause the video here to review through the points again.

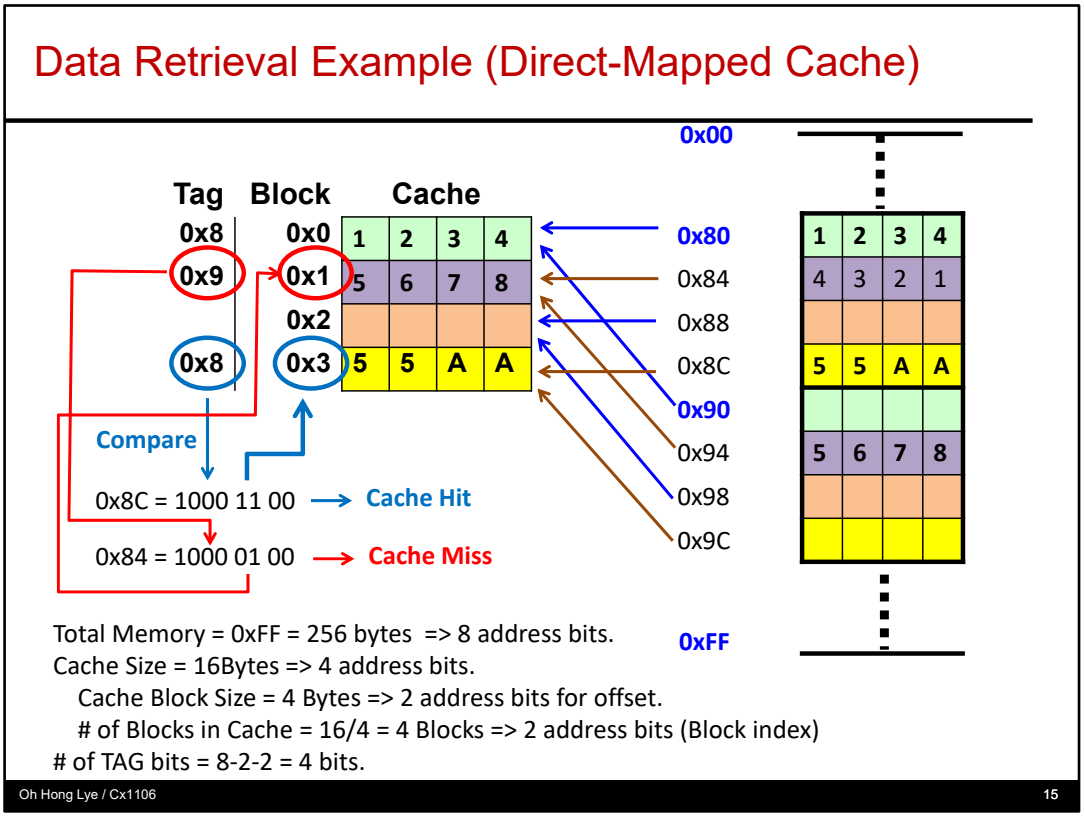
Direct-Mapped Cache Mapping Example

- 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?
- Derivation of cache mapping format
  - Cache Block Size = 16Bytes => #**Offset** bits = log2(16) = **4 bits**.
  - Number of Cache Blocks = 256/16 = 16 Blocks
    => #**Blk** bits = log2(16) = **4 bits**.
  - Main memory size = 64KBytes => log2(64*1024) = 16bits address.
    #**Tag** bits = 16-4-4 = **8 bits**
  - Mapping Format = 8:4:4
- Applying to the main memory address 0x1106
  - 0x1106 = **0001 0001 0000 0110b**
  - **BLK = 0x0**, **OFFSET=0x6**, **TAG=0x11**

- Next, let's go through a work example to check our understanding of the concepts we have learn so far for direct mapped cache mapping scheme.
- Given a system with
  - MM size 64KBytes,
  - Cache Size 256 Bytes
  - Cache Block Size 16Bytes
- Where would the Data at MM address 0x1106 be mapped to?
- Using the concept we learnt on derivation of the TAG:BLK:OFFSET fields for direct mapped cache,
  - Starting with the offset first, cache block size = 16 implies offset field is LOG2(16) = 4 bits.
  - Number of cache block is not given but that can be derived by dividing the cache size by the cache block size.
    - So there are 16 cache blocks in the cache. That implies BLK field is also 4 bits.
  - The TAG field is obtained by subtracting the MM address bit with OFFSET and BLK bits.
    - So number of TAG bits = 8.
  - That means the Mapping Format is 8:4:4
- When doing the cache mapping analysis, always remember to first convert the MM address to binary, performing analysis with hexadecimal MM address will very likely result in careless mistakes.
  - After converting 0x1106 to binary and applying the mapping format
  - We can see that the data at 0x1106 will be mapped to cache BLK 0 and has a

TAG value of 0x11.

- The target byte is located at byte 6 from the start of cache block.
- Byte 0 is the first byte of the cache block.

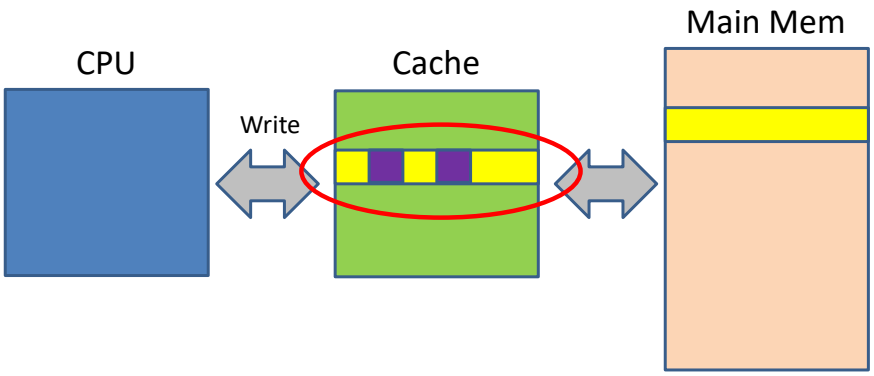


- After the work example, let's go through the data retrieval process to se how CPU evaluate whether a particular access is a cache hit or cache miss.
- Using the procedure mentioned in earlier slides, we can derive the mapping format for this cache structure
    - 4:2:2
    - Note that mapping format will change if the cache structure change e.g. change in cache size, cache block size and/or MM address range.
- Supposed after some data exchange, CPU wanted to retrieve data stored at location 0x8C.
    - Applying the 4:2:2 format on the 0x8C, BLK value is '3' so CPU proceed to check out cache block 3.
    - To find out whether the data stored at cache block 3 comes MM block that contains 0x8C's data, which btw is MM BLK 35.
        - It compare the TAG value of cache block 3 and the TAG value derived from 0x8C MM address.
        - Result is a match, that implies the data in cache block 3 does indeed comes from MM BLK 35. This is known as a Cache Hit.
- The CPU then proceed to access 0x84.
    - Applying the same procedure again, we realise that this time, the TAG value of cache block 1 is different from the TAG value derived from the 0x84 MM address. This means the data in cache block 1 does not come from MM block where 0x84's data is resided. This is known as a Cache Miss.
- If there is a Cache Hit, CPU will retrieve the data and proceed with other operations.
- If there is a Cache Miss, the corresponding MM block will be transferred into the

cache.

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**.

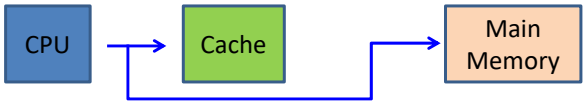


Oh Hong Lye / Cx1106 16

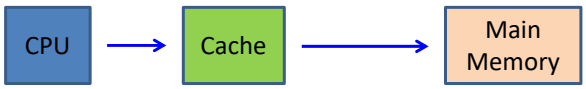
- So far, our discussion has always been related to CPU read operation.
    - In the next few slides, we will be looking at effects on cache when CPU perform write operation.
- A special marking is done to the cache block which the CPU writes into and these blocks are known as Dirty Blocks.
- Cache replacement needs to take into account of dirty blocks.
- E.g one main memory block was cached. And CPU write into some location in this block within the cache.
    - The content of the block in cache and main memory is now different.
    - Meaning that if we want to evict this block, we'll need to first write back this block to the main memory to maintain data coherency.
- There are two types of write policies: write through and write back.

Cache Write Policy (Write Hit)

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



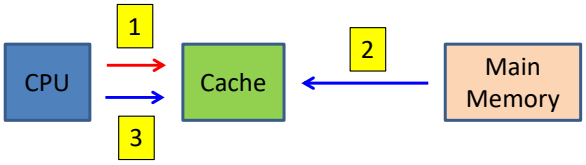
- **Write back** (also called copyback) updates memory **only** when the block is selected for replacement.



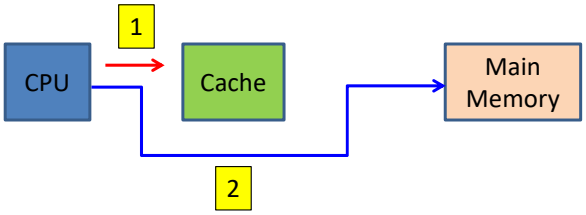
- Write through means if CPU write to a cache location, it'll also update the main memory at the same time.
- But for a Write back scheme, the main memory will be updated only when the cache block Is evicted.

Cache Write Policy (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**.



Oh Hong Lye / Cx1106 18

- Correspondingly, there are two ways in which CPU can handle write miss.
  - Write miss here refers to cases where CPU want to write to a certain main memory location but found that it is not in the cache.
  - A cache miss could be either a read miss or a write miss.
- The two ways in which CPU handle write miss are
  - Write allocate.
    - Under this scheme, the corresponding MM block will be loaded to the Cache when there is a write miss.
    - CPU will follow with a read to the cache to retrieve the data
  - Write no-allocate
    - With this scheme, CPU will write directly to the corresponding MM block if there is a Write Miss.

Cache Performance Related Terminologies

- **Cache Hit**
  - Data is found in the cache.
- **Cache Miss**
  - Data is not found in the cache.
- **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).**

- Some definition of the key parameters used for calculating Effective Access Time.
- We have encountered most of them previously, so I won't go through them again.
- You can pause the video briefly to revise the definitions.

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

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

- Effective Access Time, EAT in short, is the weighted average of the access time during cache hit and cache miss.
- Now, CPU always visit the cache first when it needs information.
    - If it can find the information in the cache, it's a cache hit and access time incurred will be the Cache access time.
    - If CPU cannot find the information in the cache, it's a cache miss and CPU will need to visit the Main Memory to retrieve the information.
        - If we further assume that access to cache and main memory happen sequentially and do not overlap,
            - Then access time during cache miss will be cache access time + main memory access time.
                - The cache access time is incurred when CPU first visit the cache to look for information.
                - The main memory access time is incurred when CPU realised that it is a cache miss and need an additional MM access time to retrieve the data.
        - As mentioned, the effective access time is a weighted average of the two access timing and is calculated by applying the probability of each of the scenario to the two access time component. Resulting in the expression you see in the slide.

Effective Access Time (Example)

Sequential Access of Cache and Main Memory

- 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%.**
- If the **accesses do not overlap**,

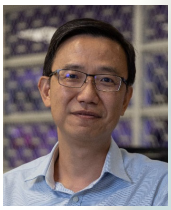
  The EAT is:

  0.99(10ns) + 0.01(10ns + 200ns) = 9.9ns + 2.1ns = 12ns.
- This equation for determining the effective access time can be extended to any number of memory levels.

- Applying what we learn about calculation of the EAT in one work example.
- System has a MM access time of 200ns, cache access time of 10ns and cache hit rate of 99%
- Remember the assumption that access to memories do not overlap.
- EAT is obtained by using the expression in the previous slide.
  - We will obtain 12ns.
- Here is a good example showing how a small cache can realise a significant improvement in Effective Access Time of the memory system.
- With that, we come to the end of the Cache Memory System module. Next lecture will be on Virtual memory management.

Cx1106
Computer Organization and Architecture

Virtual Memory



Oh Hong Lye
Senior Lecturer
School of Computer Science and Engineering, Nanyang Technological University.
Email: hloh@ntu.edu.sg

- This chapter touches on the concept of Virtual Memory Management in a Computer System.
- This is an important concept that is used widely in the design of a Operating System.
- We will cover the basics in this course and you will re-visit these concepts again when you take the Operating System Module.

MEMORY ADDRESS SPACE

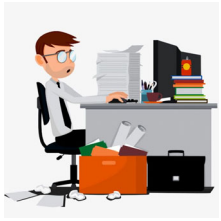
- Before we start on the Virtual Memory Management Topic, I would like to use a few slides to introduce the concept of memory address space in computing.

Memory Address Space (Analogy)

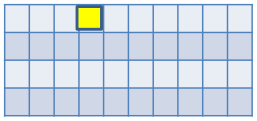
Address Space 3 (Employee)

Address Space 2 (Manager)

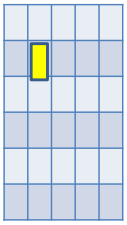
Address Space 1 (Boss)



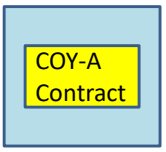
Cabinet (Company Name)



Cabinet (Market Segment)

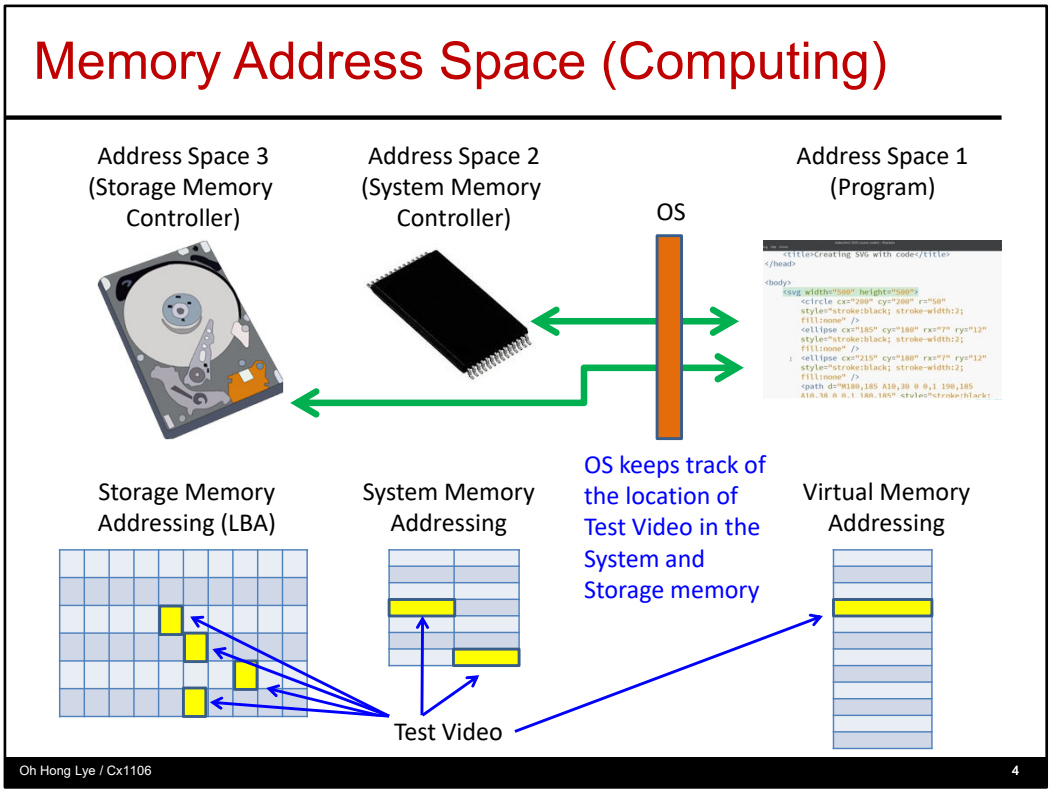


Drawer (all-in-one)



Source: Google, Baby Boss.

- Let's start with an analog.
- Memory space are how a sub-system or module keep its information.
- Take for example, in a company,
    - the big boss will keep the most important documents in his drawer.
    - His manager will probably have a larger cabinet consisting of the key contracts and information specific to the market segments they supervise.
    - The subordinates would likely have their own cabinet which they kept documents corresponding to the specific customers they call on.
- The three parties do not know how each other store their information. But the company operation will still run smoothly, as long as, when the boss needs a specific piece of information, his managers or the subordinates under the managers can provide him the required data.
- As long as the boss is happy, everything is ok.



- Coming back to computing.
- Different entities within the computer stores information differently.
    - Each will have their own memory space and own addressing scheme.
- The program user developed uses the addresses generated by the compiler and further allocated by the Operating System,
    - the memory space correspond to the virtual memory address space of the program.
    - Note that each program has their own virtual memory address space.
- The actual program code and data are stored in the storage and system memory.
    - As mentioned previously, the entire program code/data resides in the storage memory while the system memory stores the subset of the program that is needed during runtime.
- As illustrated, the same piece of information, e.g. a test video clip, is stored in a different manner in the three entities.
    - In the program's virtual memory space, the test video could reside in contiguous piece of memory
    - In the System memory, it could be distributed across different frames in the system memory.
    - While in the storage memory, it may be allocated in random LBA blocks in the HDD.
- Similar to the analog in the previous slide, the program will still be able to function properly, as long as when it needs the test video, the corresponding data would be made available to the program.

- Since the test video is stored in different manner across the entities, that mean some form of address translation have to occur in order for the program to retrieve the data correctly. The address translation is done by the OS. And we will go into details of one of the scheme employed.

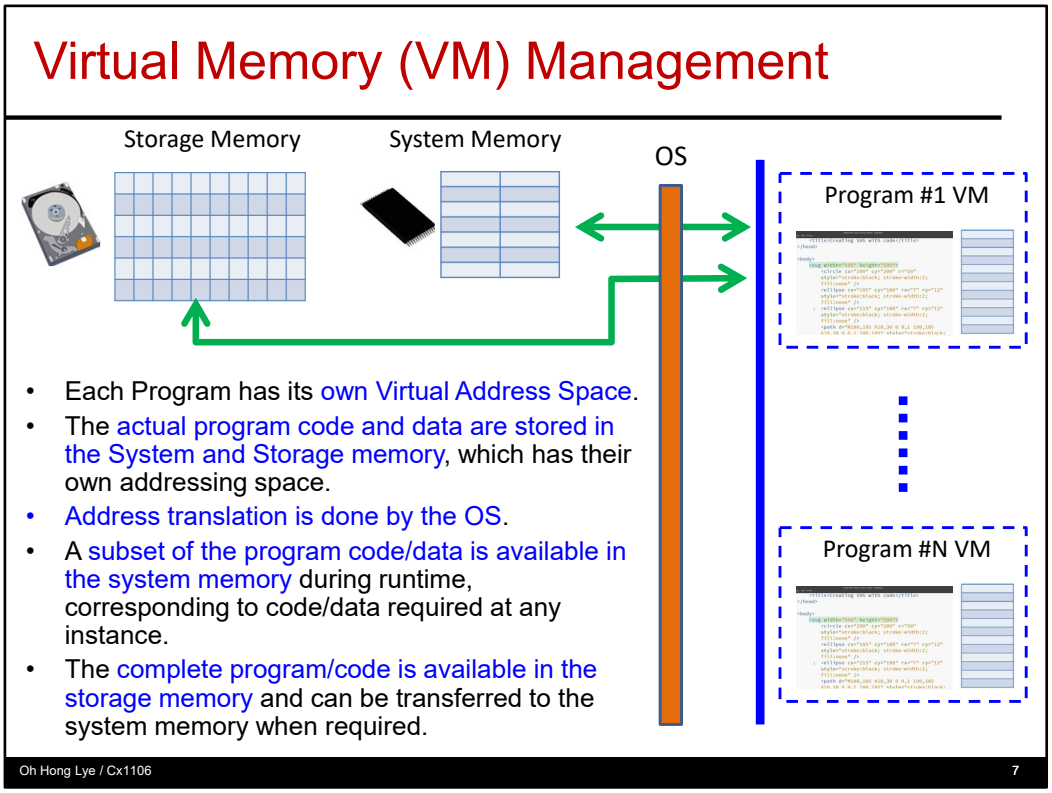
Basic Concepts

- 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.

- To recap, address space is a range of address corresponding to a physical or logical entity.
- The address used by a program is generated by the compiler.
- During program execution, OS will translate the compiler generated address to the system memory address where the information is physically stored.
- The program doesn't need to know where the actual information is stored.
- It will work properly as long as it gets the information it needs when it request for it.

VIRTUAL MEMORY MANAGEMENT

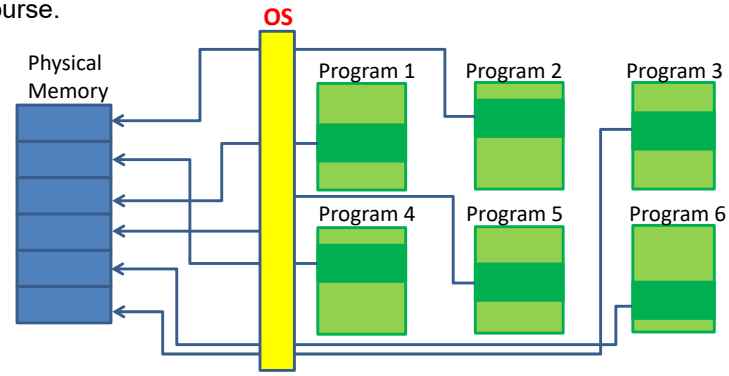
- Next, we get into the Virtual Memory Management discussion.



- As mentioned in previous slides, each program has its own virtual address space.
  - But the actual program code/data is stored in the system or storage memory, which has their own address space.
- Therefore, address translation needs to be done in order for the program to retrieve the information it needs, and this translation is done by the OS.
- Functionality wise, system memory stores the code/data required during any instance at runtime while the storage memory stores the entire program code/data.

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.

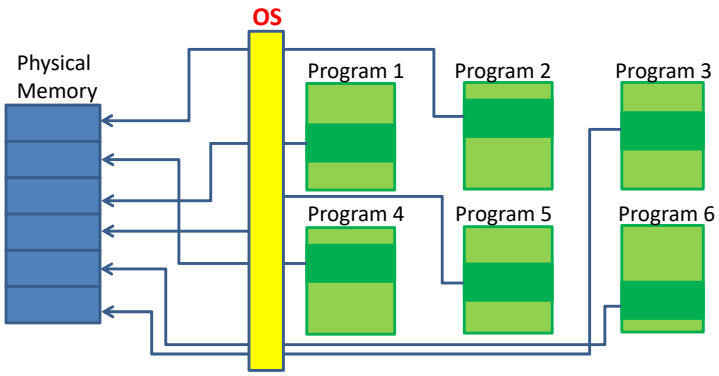


Oh Hong Lye / Cx1106 8

- Similar to the use of "Main Memory" in the Cache Chapter to refer to the System memory, in this chapter, System memory is sometimes referred to as the "Physical Memory", basically means the actual runtime memory the program code/data is stored in, this versus the Virtual Memory which is really a logical entity.
- The OS is responsible in managing the virtual memory, which includes allocating the subset of the program to the physical memory and performing the address translation from virtual to physical during program execution.
- The slide here shows the scenario where subset from multiple programs are allocated to the physical program simultaneously, with the OS handling the corresponding address translation.

Advantages of Virtual Memory

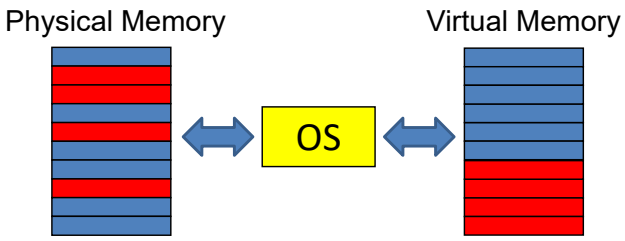
- OS is able to isolate each virtual memory space and prevent corruption between these spaces.
- Allow efficient and safe sharing among different programs within the shared physical memory.
- Allow one or more programs to run in the physical memory simultaneously even if the total size of all the programs is larger than the actual physical memory size.



- Some of the advantages of having a virtual memory management system are
  - Virtual Memory space of each program can be isolated from each other to prevent intended or unintended corruption between programs.
    - This allows efficient sharing among different programs with the shared physical memory
  - Virtual Memory management allows multiple programs to run simultaneously in the physical memory even if the total size of all the programs are larger than the physical memory size. This is illustrated in the diagram below where 6 programs shared a physical memory whose size is smaller than the total of the 6 programs.

Advantages of Virtual Memory

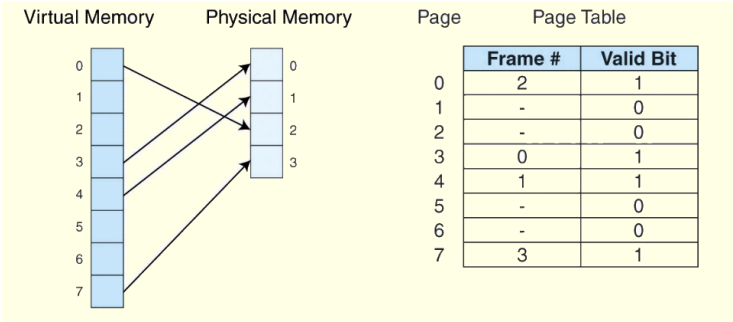
- OS is able to relocate virtual memory blocks to any physical memory blocks. This allows the virtual address space seen by the application to appear to be contiguous when it is actually spread across fragmented blocks in the physical memory.



- Another advantage of Virtual Memory Management is that the virtual address space seen by the program can appear to be contiguous when it is actually spread across different segments of physical memories.
- Having a contiguous piece of memory greatly simplify coding as the software does not need to do complicated boundary condition checking and management.

Address Mapping

- There are two common schemes used in the industry for address mapping
  - Paging. Memory space partitioned into Fixed sized blocks
  - Segmentation. Memory space partitioned into variable sized segments.
- For our course, we will only deal with paging with single level page table.



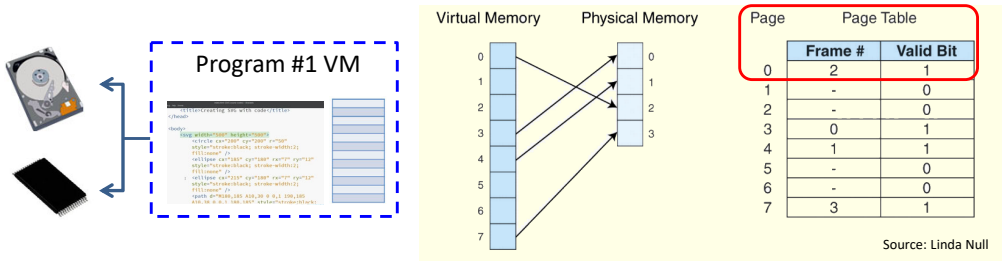
Page	Frame #	Valid Bit
0	2	1
1	-	0
2	-	0
3	0	1
4	1	1
5	-	0
6	-	0
7	3	1

Source: Linda Null

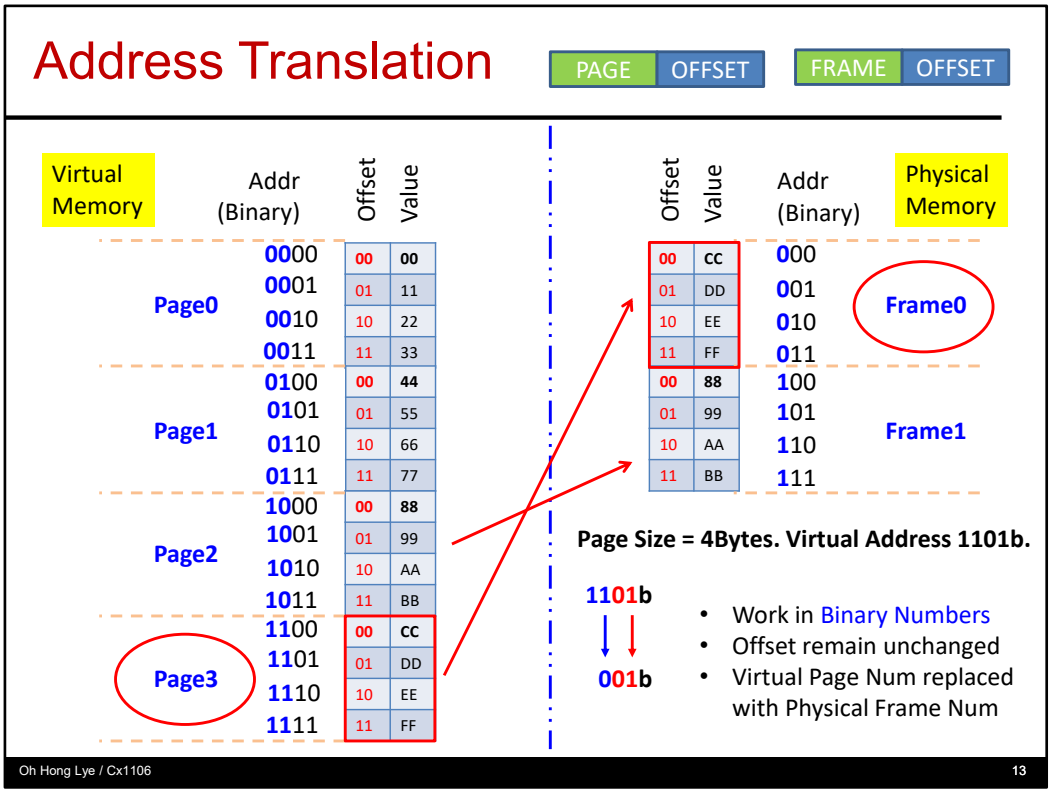
- There are two common scheme used in the industry for address mapping.
  - Paging and Segmentation.
- We will only cover Paging in the course.
- For Paging, the virtual and physical memory space are divided into fixed size pages and frames respectively.
  - We use the name 'Page' for Virtual Memory and 'Frame' for Physical Memory.
- The OS will map a virtual page to a particular physical frame and the translation information is store in a data structure known as a Page Table. More on this in later slides.
- Our course will only cover paging with single page table, you will progress with multi-level page table in your Operating System Course.

Paging Method

- 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).



- The size of a Virtual Page is always the same as the size of a Physical Frame in Paging Scheme of address mapping.
- The OS will allocate a particular virtual page used by the program to be stored in a particular Frame in the physical memory during runtime.
  - The mapping information is kept in the Page Table shown.
- Within the Page table, you will see at least three parameters
  - Virtual Page column containing the virtual page number of the target virtual page.
  - Frame column containing the corresponding frame that the virtual page is mapped to.
  - And the Valid column showing whether the particular entry is valid or not.
    - If the valid bit is '1', this implies that corresponding mapping is valid and the information in that particular page is in the physical memory at that instance. E.g. in the page table shown, Virtual Page 0 information is resided in the Physical Frame 2 at that instance.

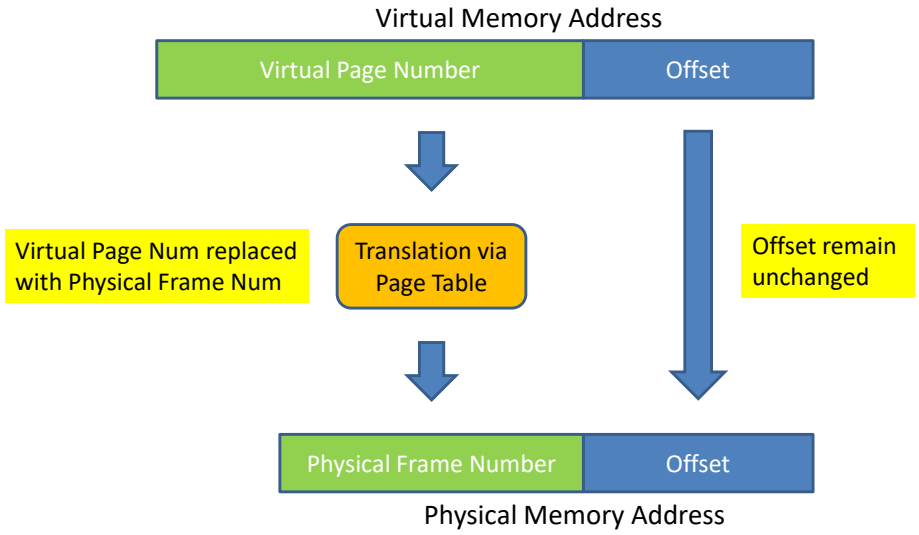


- This slide illustrate the process of address translation from Virtual to Physical Memory Space via a work example.
- The system shows here has the following attributes
    - Virtual Memory Space of 16 bytes.
    - Physical Memory Space of 8 bytes.
    - Virtual Page size of 4 bytes, meaning the Physical Frame size is also 4 bytes.
- First point to note is that when we map a virtual page to a physical frame, the relative position of the data in the page does not change.
    - You can see that when Page 3 is mapped to Frame 0, the offset of the 4 bytes of data in the Page does not change. E.g. offset of the data '0xEE' is 2 or 10 in binary in the Virtual Page3, the same data "0xEE" has an offset of 2 as well in the Physical Frame 0.
- With that, let's start the address mapping process.
    - First, convert the address to binary format. This is an important step, never work in hexadecimal format, there is a high possibility that you will make careless mistake.
    - Next, we need to partition the Virtual Address to two fields – Page and Offset.
        - This is done starting with the offset
            - Using the fact that page size is 4, then number of bits allocated to the Offset Field is 2. LOG2(4) = 2.
        - We can deduce the virtual page number from the upper address

bits of the virtual address and use that to look up the page table to find any matching Physical Frame Number.
        - If the entry for the particular page number is valid, the Corresponding Frame Number is used to replace the upper address bits to give the Physical Memory Address the virtual address is mapped to.
- Let's put some real values to illustrate the translation.
    - Starting with a virtual address of 1101 in binary.
    - Offset field is 2 bits so the address 1101 is stored in virtual page 3, 11b are the upper bits.
    - Page 3 is mapped to Frame 0 from the diagram.
        - So to derive the physical memory address, we replace '11' in binary with '0'.
        - This gives the Physical Memory address 001 in binary.
- You can try out other Virtual Memory Addresses in the is slide to reinforce your concept.

Address Translation

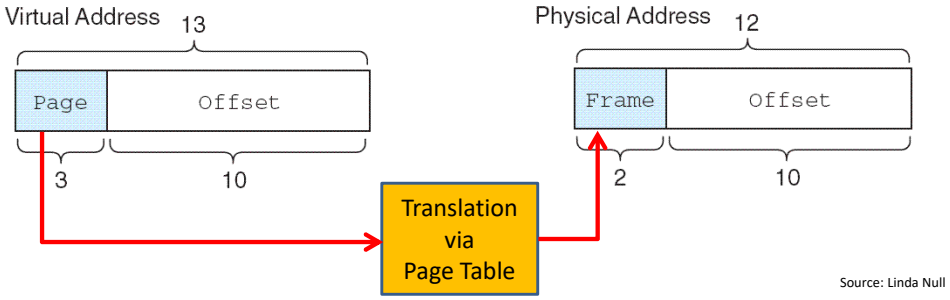
Work in Binary Numbers format



- A graphics illustration of the address translation process we discussed in previous slide.
- Convert the addresses to binary.
- Derive the number of bits in the offset field from the virtual page size.
- Partition the Virtual Memory Address to two fields: Virtual Page Number and Offset.
- The Offset remains unchanged during the translation.
- Use the Virtual Page Number to look up the Page Table for a valid mapping
- If a valid mapping is found, extract the corresponding Frame Number to replace the Virtual Page Number with the Physical Frame Number to have a Physical Memory Address.

Paging 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.

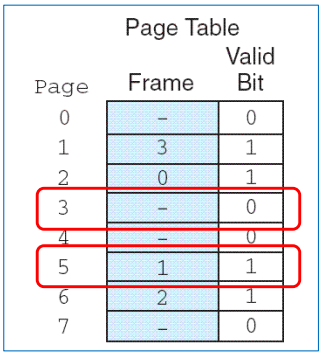


Source: Linda Null

- More work example.
- Consider a system with
  - Virtual Address Space 8KByte
  - Physical Address Space 4Kbyte
  - Page size of 1KByte
- From the attributes, we know that the system has 8 virtual pages and 4 physical frames.
- The page size is 1KByte, which is 2^10 bytes. That implies the OFFSET field is 10 bits.
- Virtual address is 13 bits and Physical Address is 12bits.
- So the translation process involves replacing the Upper 3 bits of the Virtual Address with the 2 bit Frame Number from the Page Table entries.

Paging Example

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



Page Table

Page	Frame	Valid Bit
0	–	0
1	3	1
2	0	1
3	–	0
4	–	0
5	1	1
6	2	1
7	–	0

Source: Linda Null

- 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**

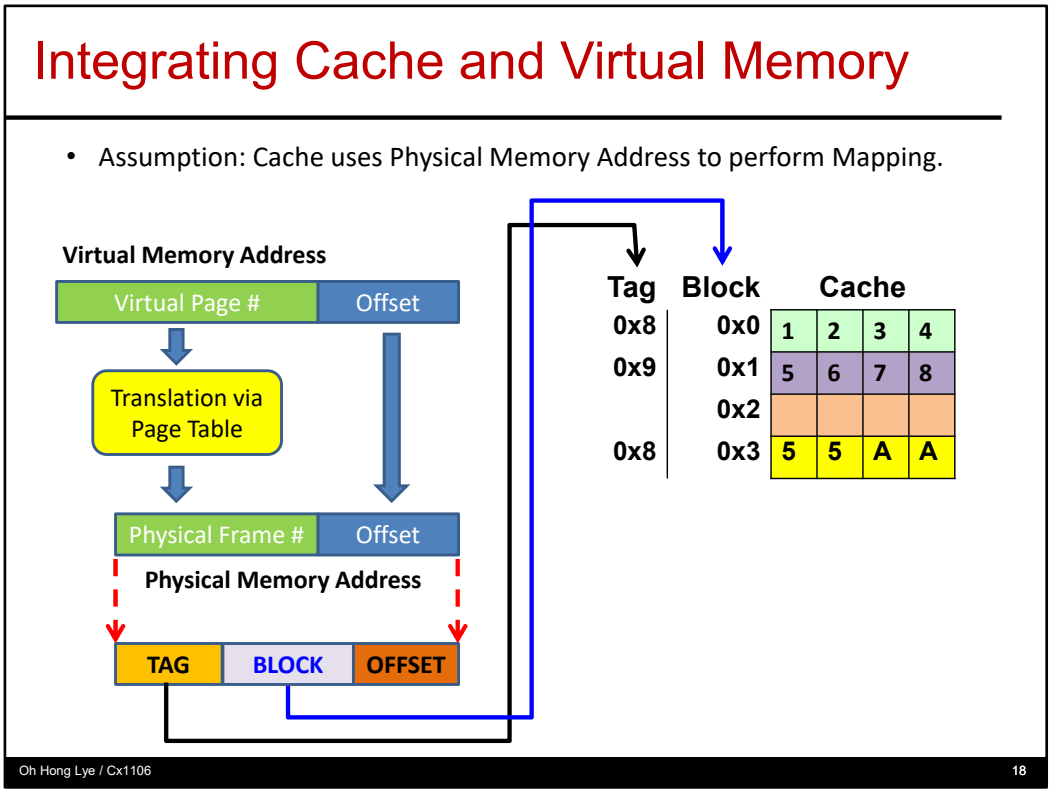
Oh Hong Lye / Cx1106 16

- Putting in some values.
- 0x1553 correspond to virtual page number 5.
  - Looking at the page table, we see that virtual page 5 has its valid bit set, meaning that it is mapped to some physical memory and from the same table, we see that the its frame #1.
  - Translation process involves replacing the upper 3 bits (101b) in Virtual Memory Address with 1.
- For 0x0FA0, it corresponds to virtual page number 3.
  - When we look at the page table, we can see that its valid bit is 0. Meaning no physical memory is mapped to this virtual page.
  - This generates a page fault that triggers the OS to load the required page from storage memory to the Physical Memory.

Paging Example

- In the previous slide, when accessing virtual address 0x1553 (which is in virtual page 5), the page table shows the valid bit is 1. From the page table, physical frame 1 will be used for subsequent address translation needed to access the actual data.
- When accessing virtual address 0xFA0 (virtual page 3), the page table shows that the valid bit is zero. If the valid bit is zero in the page table entry for the virtual address, this means that the page is not in memory and must be fetched from Storage Memory.
  - This is a page fault.
  - 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.
- For both cases above, the data is then accessed by appending the offset to the physical frame number (address translation is simply replacing the virtual page number with the corresponding physical frame number in the page table).

- This slide summarises the discussion we did in the previous slide.
- You can pause the video for a while to review the points again.

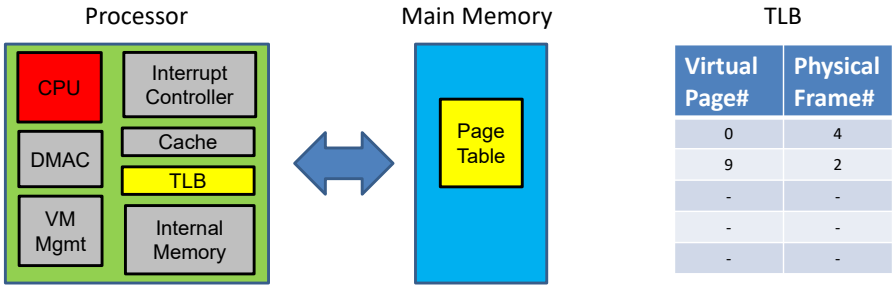


- In a typical computer system, especially one with an operating system, both cache and virtual memory management will be implemented.
- Therefore, there is a need to look at integration of these two modules.
- This slide shows one of the ways in which the processor may retrieve the information required when executing a program
- With a software program, we would always start with the virtual memory address.
- If we assume that the cache in this processor uses physical memory address to perform the indexing and mapping, then we following procedure will be carried out within this computer system.
    - First, the virtual memory address will be translated to its corresponding physical memory address using the methods we discussed in previous slides.
    - The CPU then anaylse if there is a cache hit or cache miss using the target's physical address.
    - If it is a cache Hit, then CPU will proceed to retrieve the information from the cache and proceed.
    - If it is a cache miss, then cache management module will proceed to perform the transfer of data from System/Storage memory.
- Note that the sequence will be slightly different if the Cache is using Virtual Memory Address to do the indexing and mapping.
    - Starting with the virtual address used by the program again, the CPU will visit the cache first to check if it is a cache hit or cache miss.
    - If it is a cache hit, CPU will retrieve the data and proceed.

    - If it is a cache miss, that means the block containing the required data needs to be fetched from the Main Memory.
        - But the OS will have to translate the virtual memory address to physical memory address before the data can be fetched.
- As seen, the sequence of access and translation is different but the concept behind each of the cache mapping and virtual memory translation is the same.

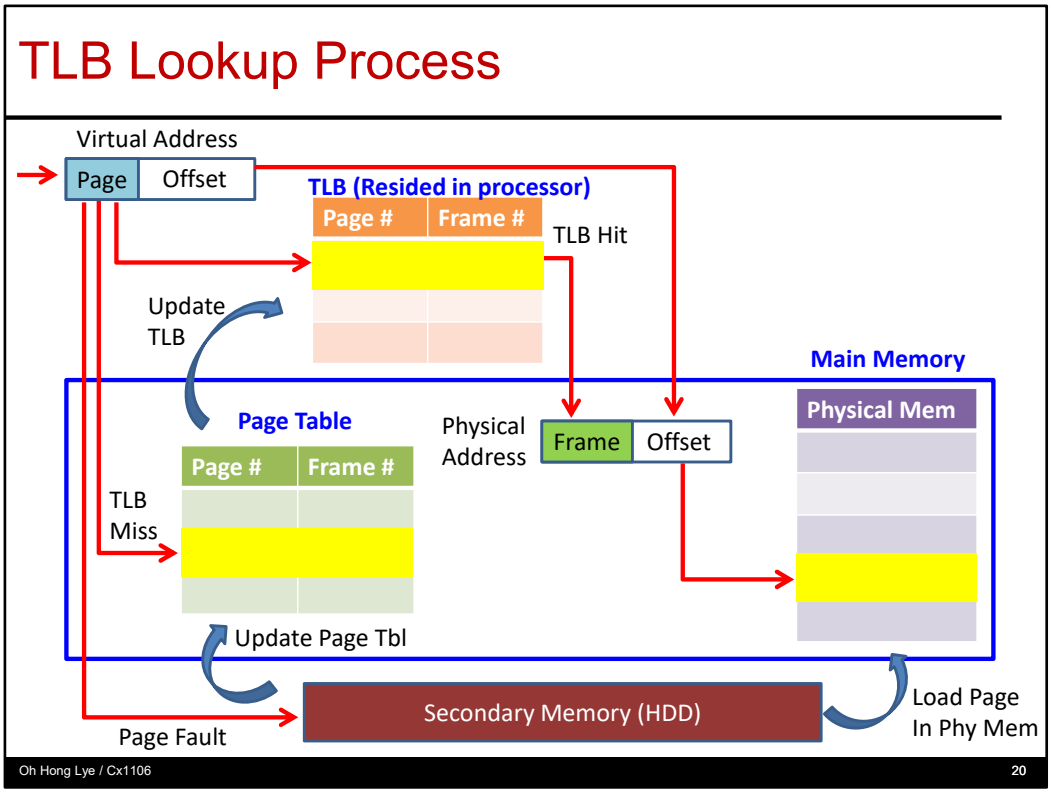
Translation Lookaside Buffer (TLB)

- 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.



Oh Hong Lye / Cx1106 19

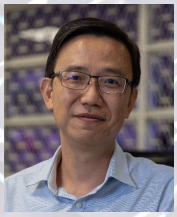
- Due to its size, which could go up to a few MBs, Page table is located in the Main Memory, which is relatively slower compared to the internal memory.
- Since the process of translating Virtual Memory Address to Physical Memory Address happen very frequently, it makes sense to optimise this process.
    - One way is to use a fast internal memory, called a TLB (Translation Lookaside Buffer) to store frequently used page table information.
    - And get the OS to refer to the TLB first when they are looking for address translation information.
    - If the hit rate of the TLB is high, the overall speed of the address translation will speed up significantly.
- TLB is implemented with fast memory like SRAM and resides in the processor.
    - It stores a subset of the Page Table information.
    - Each TLB entry includes the Virtual Page Number and the corresponding Physical Frame it is mapped to.
        - All entries in the TLB are Valid entries.
- So in some way, TLB function like a cache to the Page Table.
- But the difference between a TLB and the Cache we discussed previously is that a TLB stores address translation information i.e. which virtual page gets mapped to which physical frame. While a regular cache stores the actual code and data of a program.



- The slide here illustrate the 3 ways that the virtual to physical memory address translation could be carried out.
- Starting with the Virtual Memory Address.
    - The OS will check the TLB for a matching entry.
        - If the virtual page being access has an entry in the TLB, then it is a TLB hit.
        - The OS can extract the corresponding Physical Frame Number the virtual page is mapped to and proceed to derive the Physical Memory Address and access the data.
    - If the require entry is not in the TLB, then it is a TLB miss.
        - OS will proceed to check the Page Table resided in the Main Memory for the information.
        - If a valid entry if found in the Page Table, it is a Page Table Hit, the OS proceed to update the TLB, compute the physical address and access the data.
    - Under the worst case scenario, the entry in the Page Table is invalid, that means the required data is not in the Physical memory and the OS needs to transfer the data from the Storage memory to the Physical Memory and update the page table.
    - Whether the TLB gets updated or not will depend on the Virtual Memory Management design.
    - That is the end of the chapter on Virtual memory management.

Cx1106
Computer Organization and Architecture

Communication and User Interface



Oh Hong Lye
Senior Lecturer
School of Computer Science and Engineering, Nanyang Technological University.
Email: hloh@ntu.edu.sg

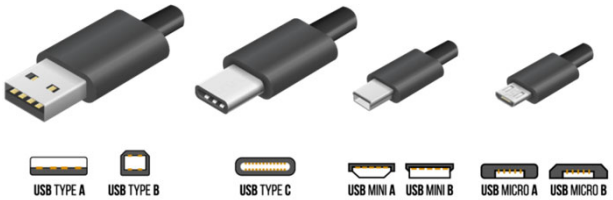
- This chapter deals with the more complex communication and user interfaces that a typical computer has.
- I have grouped them here to explicitly indicate that we will only cover the overview and basics where these interfaces are concerned.

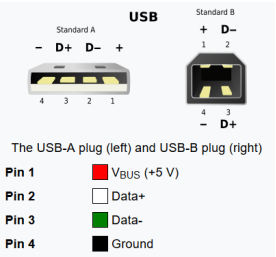
Introduction

- This chapter introduce some of the common
  - Wired and wireless communication technology used in computers.
    - USB and HDMI
    - WiFi and Bluetooth
  - User interface devices used to transfer real world data.
    - Keyboard/Mouse
    - Capacitive Touch devices
    - Camera
    - Microphone
    - Display
    - Audio Speakers
- Only a general overview of the above will be discussed, technical details will not be covered.

- This chapter gives a basic introduction to the common interfaces seen on a computer system, particularly the Smartphone and the Lap Top.
- These includes the wired and wireless communication technology such as the USB, HDMI, Wifi and Bluetooth
- And other user interfaces such as the
  - Key Board and Mouse
  - Capacitive Touch Devices
  - Camera
  - Microphone
  - Display, and
  - Audio Speakers
- Note again that we will only touch on the basics for the above. You can reference the lecture notes for the required depth you need to get into.

Universal Serial Bus (USB)



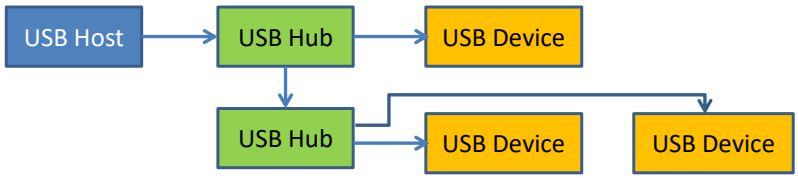


- Serial Bus designed to standardize connection of computer peripheral devices (Keyboard, Mouse, Printers, Disk Drives, Network Adapters, Digital Cameras etc).
- Effectively replaced many interface buses e.g. Serial Bus (COM Port), Parallel Port etc
- USB1.0 standard only supports a transfer rate of 12Mbps but has progressed over the years and the latest USB3.2 standard supports up to 20Gbps transfer.
- 4 basic pinout: VBUS (+5V Power supply), Data+/Data- (Data pins) and Ground pin.

Oh Hong Lye / Cx1106 3

- First let's look at the most common external bus interface in the computer world today, USB.
- USB was first proposed to replace low speed peripherals such as the PC Serial COM Port, Keyboard/Mouse PS2 etc.
    - But it quickly become the de facto standard to interface to many other devices, e.g. Printers, Cameras, Disk Drives etc.
- The fast adoption is partly due to the continuous improvement in performance brought about by each newer version of the USB standard
    - It start with USB1.0 which only supports up to a data rate of 12Mbps,
    - But has progress over the years and the latest USB 3.2 standard is able to support up to 20Gbps transfer, sufficient to support real time video streaming.
- There are 4 basic pins in a USB interface
    - VBUS which is the power supply rail of the USB bus. This is also the pin that the USB devices can tap on for power supply to their system.
    - Data+/Data- pins which carries data information with a differential signal electrical interface.
    - Ground Pin for the VBUS supply.

USB Topology and interface



- 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 uses a Tiered-Star Topology
  - The USB Host is at the center of the star, connecting to either USB Hub or USB Devices.
    - USB Hub can further connect to more USB devices, with itself at the center.
- All data transaction are initiated by the USB Host.
  - Each device is identified by its unique address which is assigned by the USB Host.
  - All data transaction is with respect to the Host, i.e. an IN transaction refers to data going into the Host while an OUT transaction transfer data out of the Host.
- As mentioned earlier, power for USB devices can be supplied via the VBUS.
  - If a device draws its power from VBUS, then it is known as a Bus-Powered Device.
  - Else, it is known as 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.

- In order for USB host and Device to communicate with each other, they need to go through the USB enumeration process.
- When the USB device first connects to the Host, it will go through a process know as USB enumeration where the Host and Device will exchange information on their capability and requirement.
  - E.g. when a USB mouse is connected to the PC, it will inform the host its Vendor and Product ID, the polling rate it supports, the USB device class drivers it supports and need, how many buttons it has, whether it is bus or self powered etc.
- The host can only start communicating with the new USB devices after it has gone through the enumeration process successfully.
- Once the host understand the USB device's requirement, it'll check and load the required device drivers to support its operation.
- Device drivers are software doing the first level interface with the USB device's hardware.
- There are many different types of USB device class. Some of the more common ones are listed here.
  - Human Interface Device. HID in short. This device class is used in many of the user input devices such as Keyboard. Muse, touch pad etc.
  - Communication Device Class., which is used to implement the Virtual COM Port used in many embedded system boards. E.g. Arduino, Raspberry Pi and the MSP432 Launchpad used in your lab sessions.
  - Mass Storage Class device class which handles external mass storage devices such as the HDD, SSD and USB thumb drives.

- USB Audio Class which supports streaming audio via the USB bus. One example of such devices is the USB Headphone,

High-Definition Multimedia Interface (HDMI)

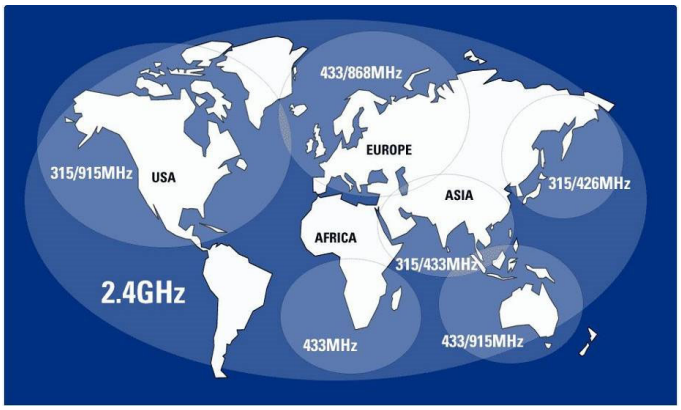
- HDMI is a proprietary audio/video interface for transmitting uncompressed video data and compressed/uncompressed digital audio data from a source device, such as a display controller in a computer, to a compatible HDMI receiver such as computer monitor, digital television, or digital audio device.
- 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.

- HDMI is a complex interface standard that is able to transfer high quality uncompressed audio and video data.
- In addition to the media data, it also support the CEC standards which allows HDMI devices to control each other and allow user to operate multiple HDMI devices with only one remote controller.
- Hardware interface wise, there are three commonly used HDMI connector, Type D, C and A. You can see these connectors in the photo, Type D is the micro version, which is the smallest connector on the left, this it followed by type C, which is the mini, and type A, which iis the regular HDMI connector.
- Similar to USB, HDMI has progressed through time, the first version HDMI1.0 can only support up to 3.96Gbps data transfer rate, but it has increased to 42.6Gbps in the HDMI v2.1. With this data transfer rate, the HDMI2.1 compliant device is able to transfer and display 8K high resolution video.

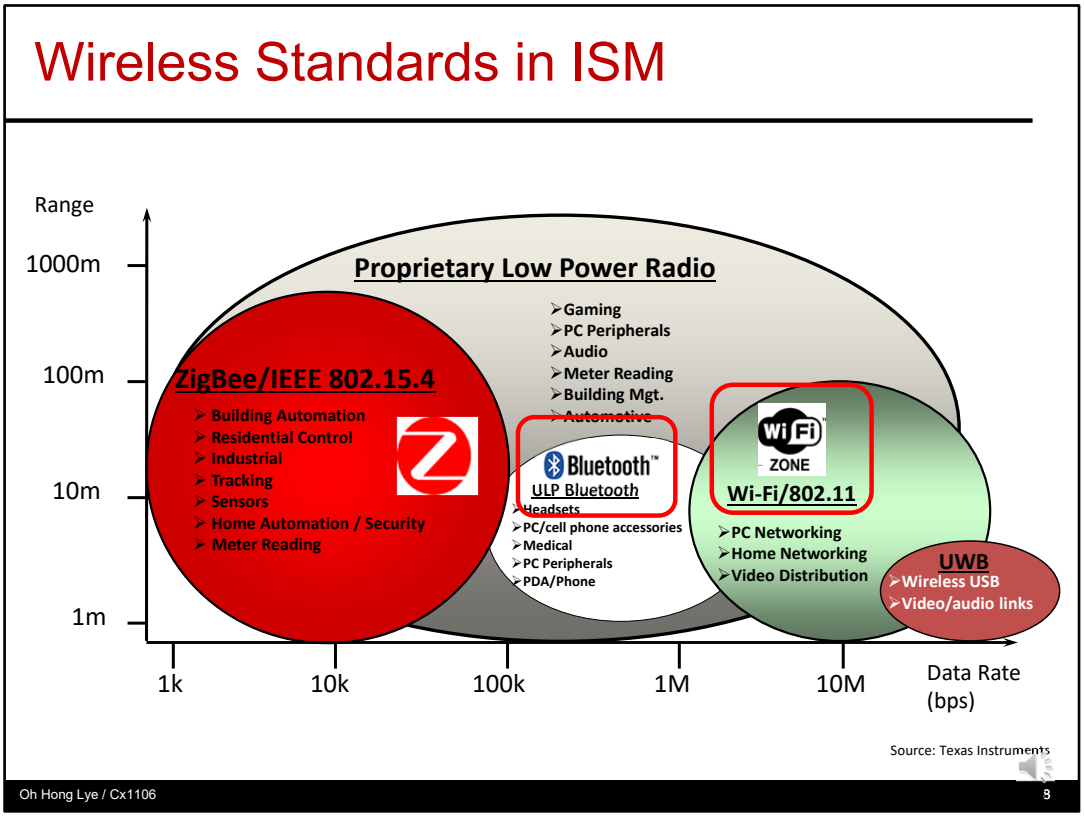
Industrial, Scientific and Medical (ISM) RF Band

- 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.



Oh Hong Lye / Cx1106 7

- If you haven't realised, the use of various frequency bands in the air to transmit data is typically not free, i.e. you need to pay the government or some organisation in order to transmit data in most frequency bands.
    - E.g. the telecommunication companies like Singtel, Starbhub, M1 has to pay the Singapore government licensing fees in order to transmit data via the 3G/4G cellular bands.
- There are however, some frequency bands that are free for all to use.
    - These are known as ISM Bands, which stands for Industrial, Scientific and Medical RF Band.
        - ISM frequencies varies with the geographical regions, but there are two frequency bands that are standard world wide.
            - 2.4Ghz and 5.8Ghz bands.
            - Incidentally, these are also the two bands that are used by the Wifi transmission standards.

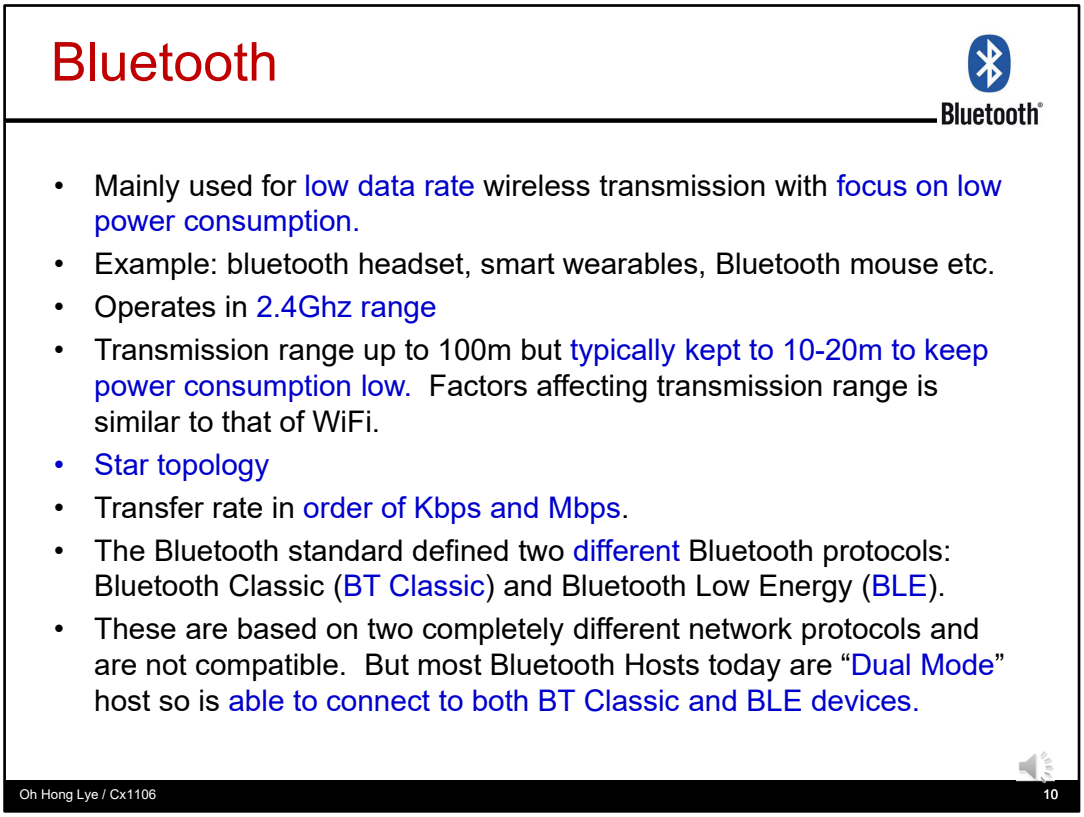


- One of the most commonly used ISM band is the 2.4Ghz band, this slides shows some of the RF standard that uses this frequency band for data transmission.
- We will be touching on two of these standards, which is the Wifi and Bluetooth.

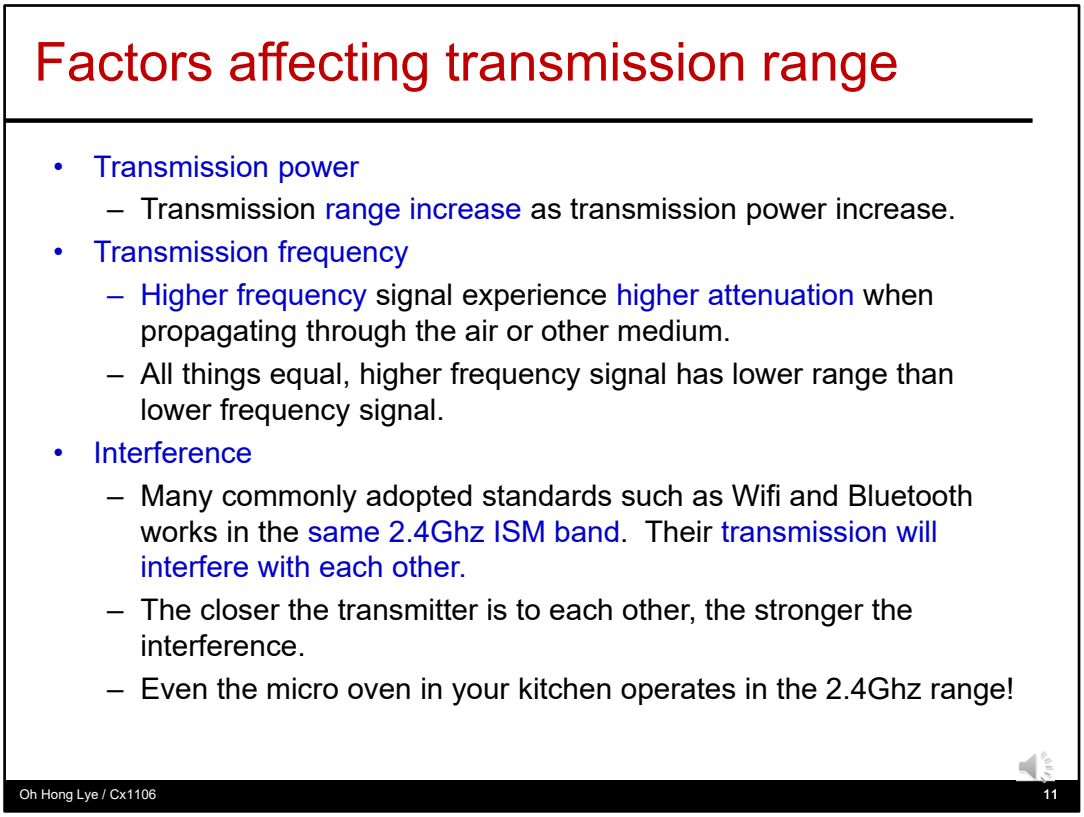
WiFi

- 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).

- Wifi is based on the IEEE802.11 family of wireless transmission standard.
- It initially only utilise the 2.4Ghz RF Band but has progress in recent year to use both 2.4Ghz and 5.8Ghz bands.
- Two common topologies used in Wifi are the Infrastructure and Adhoc mode.
- Infrastructure mode use a Star Topology.
    - The Access Point or Router is at the center and terminals such as you phones, tablets and lap tops are the end devices.
- Adhoc mode is a peer to peer connection between two wifi enabled devices.
    - One example is the Wifi Direct technology.
- Transmission range for wifi is generally around 20 to 150m.
    - Factors that affects transmission range includes
        - Transmission frequency
        - Transmission power
        - RF interference
    - Wifi data transfer rate of 10Gbps is possible with the latest 802.11ax, also call Wifi6.

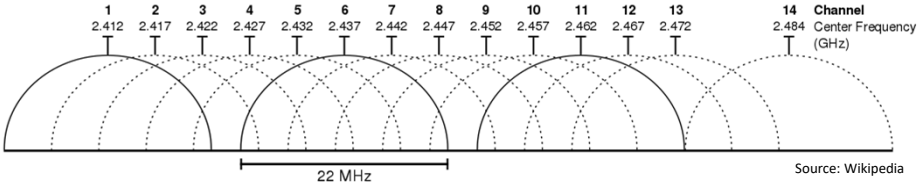


- Bluetooth transmission standard targets low power applications, compared to Wifi transmission standard.
    - Incidentally, the transmission rate is much lower than Wifi as well.
- Example of Bluetooth applications includes headset, keyboard, mouse, smart wearables etc.
- Bluetooth devices operates in the 2.4Ghz band
    - Transmission range can be up to 100m but is typically kept to 10-20m to keep the power consumption low.
    - Factors affecting the transmission range are similar to Wifi, in fact, the factors discussed are applicable to all RF transmission standards.
- Bluetooth also used a star topology
    - The center of the network is known as a Bluetooth Central, the end device known as device or peripheral.
- Transfer rate are typically lower than wifi and is in the order of Kbps and Mbps, in fact typically more Kbps than Mbps.
- Note that the Bluetooth standard defines two completely different protocols
    - One is the protocol defined in the original Bluetooth standard, this is known as the Bluetooth Classic.
    - The other is a new protocol adopted later to further improve the power consumption, this is known as Bluetooth Low Energy. BLE in short.
    - These two protocol are not compatible with each other.
- But most Bluetooth Host on Smart Phone, Tablets or Lap Top these days are "Dual Mode" Host and is able to talk to both BT Classic and BLE devices.



- We will go into a little detail on the factors affecting wireless transmission range and the mitigating methods, in the next two slides.
- As mentioned in previous slides, the discussion here is applicable to all RF transmission standard.
- First factor that affects the transmission range is the transmission power.
    - Larger transmission power will give rise to larger range.
- Rate of attenuation increases if the transmission frequency increases.
    - i.e. a 2.4Ghz RF signal will suffer less attenuation compared to a 5.8Ghz signal.
- One of the key factor that affect transmission range is the interference from other transmission sources.
    - This is especially so for 2.4Ghz which is used by many RF transmission standard.
        - E.g. Wifi, Bluetooth, Cordless Phones, Baby Monitor etc
        - Even your Microwave Oven uses 2.4Ghz!
    - In general, the closer the transmitter is from each other, the stronger the interference.

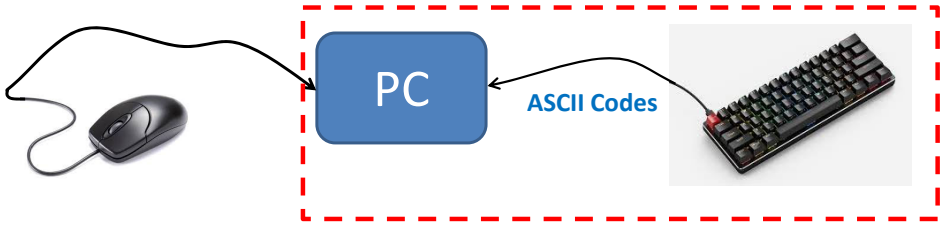
Mitigating Interference (some methods)



- "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.

- As interference is one of the most common factor affecting the RF transmission range, a lot of effort is put in to mitigate the effect of interference.
- Before we go into the actual detail, it is useful to know that the so call 2.4Ghz that wifi works on is actually a band of frequencies rather than exactly 2.4Ghz.
- The 2.4Ghz band that wifi or Bluetooth works on is shown in the diagram in the slide.
  - It is roughly between 2.4 to 2.5Ghz.
  - The 802.11 standards that Wifi and Bluetooth use divides this frequency band into multiple sub-bands known as channels.
  - At any point in time, the wifi or Bluetooth will use one or more of these bands for transmission.
- So incidentally, one way of mitigation is to have different device use a different channel or channels for transmission.
- Another way, which is used in Bluetooth, is to hop from one channel to the other periodically so as to reduce the chances of colliding into each other's operating band. This is known as Frequency Hopping.
- There are other methodologies and techniques employed to further enhance the transmission,.

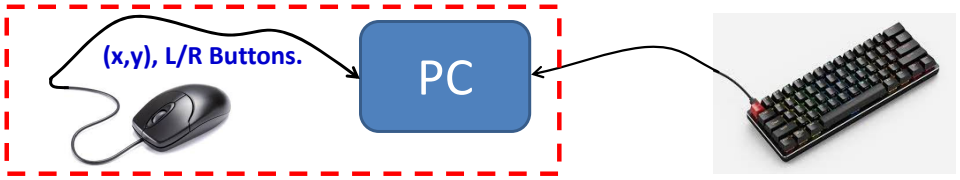
Keyboard and Mouse



- **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 are the two important user input devices in a computer system.
- Both device has a micro-controller in them that sent the main processor in the computer information based on the keys, buttons and mouse movement status.
- For Key Board, note that a coded form of the key status is sent rather than the actual value.
    - E.g. when the Key Pad '1' is pressed, the ASCII code of '1' is sent to the main processor.
- Connection to the PC can be wired or wireless
    - Wired Keyboard typically use the USB interface
    - The devices will be enumerated as a HID device class under USB standard.
- Wireless KeyBoard mainly operate in 2.4Ghz ISM band, it could either be using Bluetooth technology or some proprietary protocol designed by the vendors.

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.

Oh Hong Lye / Cx1106 14

- Mouse is the main pointing device use in computer system.
- It tracks the cursor movement on the screen and is the key enabler for icon-based GUI.
- The tracking is done via mechanical or optical means.
    - For mechanical, there is a ball at the base of the mouse that will rotate as user slide the mouse, the ball will in turn move two rollers that track the X and Y axis movement.
    - For optical, the information transferred between the transmitter-receiver pair underneath the mouse tracks the movement of the mouse.
- Basic information reported by the mouse are the X/Y coordinates and the left/right mouse button press.
    - There may be more information reported for more advanced mouse, e.g. scrolling, panning and customised key codes etc.
- How fast these information gets reported back to the PC is known as the polling rate of the mouse.
    - Typical mouse has a polling rate of 125Hz but gaming mouse could have polling rate of up to 1000Hz or higher.
    - Higher polling rate mean faster response.
- Another Parameter is the DPI, which stands for Dot-Per-Inch.
    - This measures how fine the mouse is able to track the physical movement.
    - Higher DPI means higher sensitivity to small physical movement.
- Connection to PC is either wired or wireless and technology used is similar to the KeyBoard,

ASCII Table

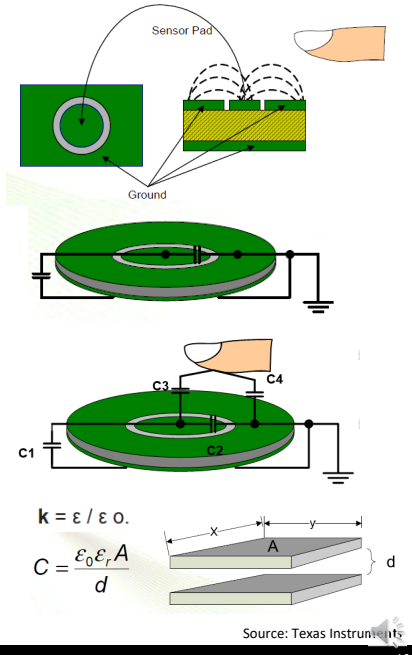
LS \ MS	0	1	2	3	4	5	6	7
0	NUL	DLE	SP	0	@	P	'	p
1	SOH	DC1	!	1	A	Q	a	q
2	STX	DC2	"	2	B	R	b	r
3	ETX	DC3	#	3	C	S	c	s
4	EOT	DC4	$	4	D	T	d	t
5	ENQ	NAK	%	5	E	U	e	u
6	ACK	SYN	&	6	F	V	f	v
7	BEL	ETB	'	7	G	W	g	w
8	BS	CAN	(	8	H	X	h	x
9	HT	EM	)	9	I	Y	i	y
A	LF	SUB	*	:	J	Z	j	z
B	VT	ESC	+	;	K	[	k	{
C	FF	FS	,	<	L	\	l	|
D	CR	GS	-	=	M	]	m	}
E	SO	RS	.	>	N	^	n	
F	SI	US	/	?	O	_	o	DEL

ASCII Character Set (7-Bit Code)

- This is the ASCII table which has been introduced to you previously during the first half.
- Keypad ‘1’ has an ASCII code = 0x31 (MS = 3, LS =1 in the table).
- So 0x31 is sent to the main processor in the computer when the keypad ‘1’ on the keyboard is pressed.

Capacitive Touch Interface

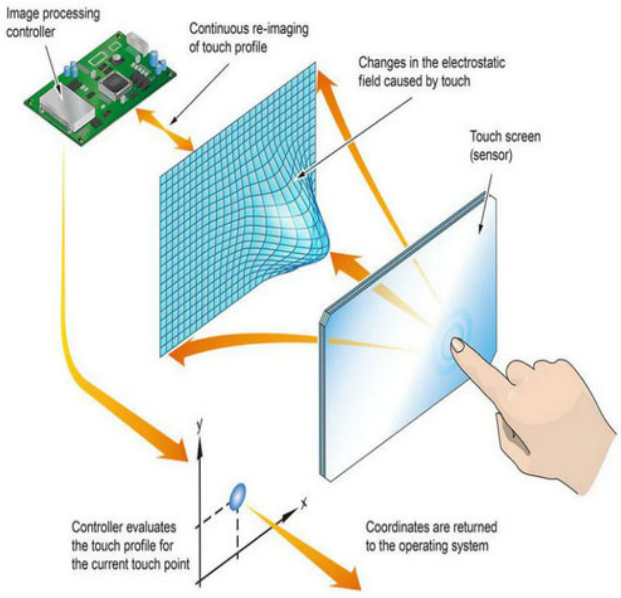
- A capacitive touch pad/button on the right is seen as a capacitor to the processor it is connected to.
- When a conductive element is present, e.g. finger, the effective capacitance of the setup increases.
- Capacitance is also affected by any dielectric e.g. gloves, plastics, liquid between the finger 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.



Oh Hong Lye / Cx1106 16

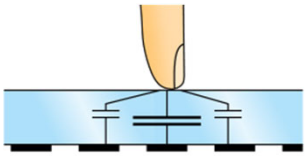
- Capacitive touch technology is widely used to implement touch button, pads and screen for smartphones, tablets and laptops.
- The working principles behind this technology is the change in capacitance of the hardware setup in the presence of human finger.
- The diagram on the right illustrates this principle
  - The electrodes that form the capacitive touch button is seen as a capacitor to the processor connected to it.
  - When an electrically conductive element such as the human finger is present in the vicinity, the effective capacitance of the setup changes.
    - The diagram illustrates the effect of putting a finger near a capacitive touch button, additional Capacitors are formed between the fingers and the electrodes of the button.
  - This change result in a change in electrical behaviour which can be picked up by the processor.
- Changes in capacitance is also affected by the dielectric of the capacitor.
  - This dielectric could be the glove, plastic sheet or a film of water between the finger and the capacitive button.
  - Capacitance is directly proportional to the dielectric constant of the dielectric material and air typically has a lower dielectric constant than most materials.
  - Which is why your phone's touch screen doesn't work or is not as responsive if your hand is wet or you wore your gloves.
- As the change in capacitance for a setup differs with different conductive medium interacting, calibration is needed to ensure a correct and reliable result.

Capacitive Touch Screen



Source: Wikipedia, Arrow.

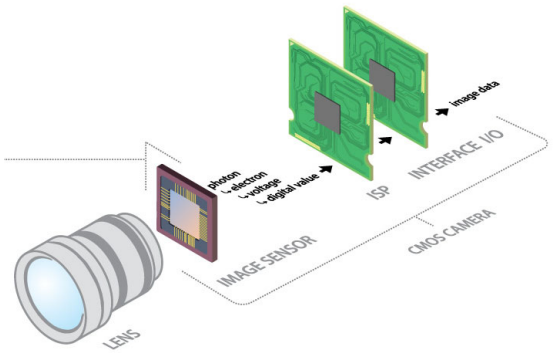
- Capacitive touch screen consist of an array of electrodes functioning as capacitive touch sensors.
- Finger touching any part of the screen will affect one or more electrodes, with the one closest having the most change to its nominal capacitance.
- A controller is used to process the information to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



Oh Hong Lye / Cx1106 17

- A capacitive touch screen is an array of electrodes functioning as individual capacitive button.
- When a finger touches any part of the touch screen, it interacts with the electric field of the electrodes in the vicinity.
    - The electrode nearest to the finger will experience the largest change in its capacitance.
- These changes in capacitances are fed back to a controller, processed to derived the exact position where the finger touches the screen
- This position information is typically in the (x, y) coordinate form.

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.

- Most phones, tablets and laptop these days are equipped with one or more cameras.
- There are two main types of camera sensors used in the market, CCD and CMOS.
  - Most products with thin mechanical ID profile uses the CMOS camera.
  - The camera module consist of the following main components
    - Lens
    - Image Sensor
    - Image Signal Processor
    - Interface I/O

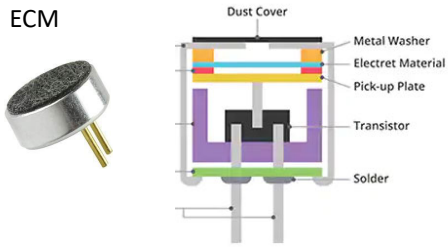
Camera Sub-Modules

- 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.

- Camera Lens is a transparent medium used to focus the real world images onto the image sensor.
- Image Sensor IC are mostly CMOS based
  - CMOS stands for Complementary Metal Oxide Semiconductor, which is a semiconductor process technology based on MOSFET transistors.
  - A sensor IC consist of an array of light sensors that transduce photons to analog electrical signals
  - These analog signals are then passed into ADC to be converted to digital signal that can be processed by the digital processor.
- The processor on the camera module is known as the Image Signal Processor, ISP in short.
  - Digital data of the images detected by the sensor IC is sent to the ISP for further processing. Some of these processing are
    - Auto Focus, Auto Exposure and Auto White Balance, commonly known as '3A'.
    - Conversion between different image format, e.g. YUV, RGB etc.
    - Image post processing such as noise reduction and lens distortion adjustment etc
    - Image or Video compression to various formats such as JPEG, H.264 etc.
- The camera module needs to talk to the main processor in the computer system and that it done via the interface I/O controller on the module.
  - The I/O controller encapsulate the image and video data in a format that the processor support.

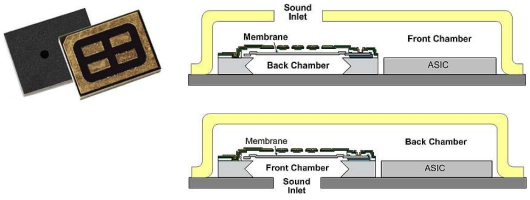
Microphones

ECM



MEMS Microphone

Source: EDN, DigiKey

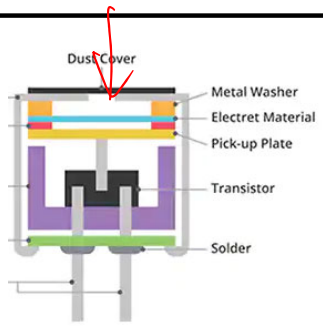


- Two of the most popular types of microphones are micro-electro-mechanical system (MEMS) microphones and electret condenser microphones (ECM).
- Both MEMS microphone and ECM uses the variation in capacitance when the diaphragm is displaced by the sound pressure to transduce sound wave to electrical 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.

Oh Hong Lye / Cx1106 20

- Microphones are present in almost all computer system that supports voice communications.
- Two of the most popular types of microphone are the MEMS and ECM.
- Operating principles of both types of microphone are similar. Both has a diaphragm that can be displaced by sound waves, movement of the diaphragm in turn changes the capacitance of the setup, and these capacitance changes can be picked up by the electrical circuits connected and sent to the processor for further processing.
- The difference between MEMS and ECM is that
  - in the ECM, the electrical charges needed to measure the change in capacitance is stored permanently in the electret. The electret here forms one of the capacitor plate.
  - While for MEMS microphone, the same electrical charges for the capacitor plate is provided by a charge pump instead.
- The transduced signal is analog but an ADC can be included in the microphone to enable a digital output.

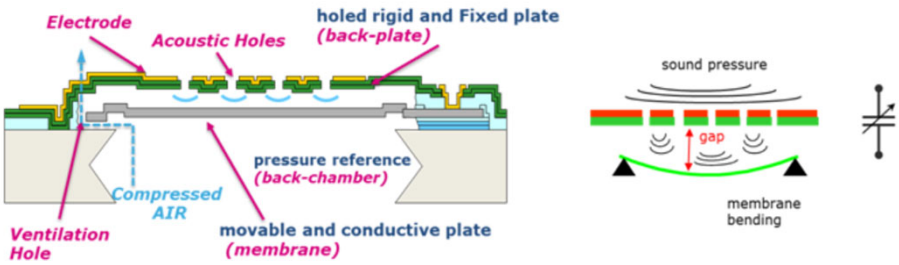
ECM Microphone



- 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).

- In the ECM microphone, the two capacitor plates are formed by
  - an electret, which stores some excess charges permanently and form the movable plate of the capacitor.
  - A fixed conductive plate connected to the rest of the circuitry of the microphone.
- The pressure created by the sound wave causes the electret to move, changing the gap of the capacitor and hence the capacitance.
- Advantage of the ECM microphone is that its performance is more robust under scenario of fluctuation in supply voltage, because the charges embedded on the electret are permanent and not affected by variation in supply voltage.

MEMS Microphone



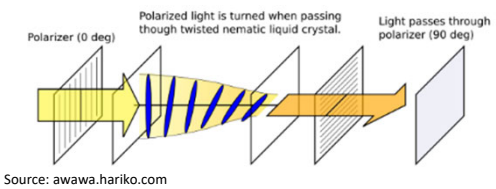
- MEMS microphone basically is an acoustic transducer.
- Transduction principle is the coupled capacity change between a fixed plate (back-plate) and a movable plate (membrane)
- The capacitive change is caused by the sound, passing through the acoustic holes, that moves the membrane modulating the air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

Oh Hong Lye / Cx1106 22

- MEMS microphone also employs the design of one fixed and one movable capacitor plate.
  - The back plate is charged up by an internal charge pump.
  - The pressure from sound wave pass through the acoustic holes and move the movable conductive plate, this changes the capacitor plate gap and therefor the capacitance of the setup.
- MEMS microphones are more popular these days because compared to ECM microphone, MEMS microphone
  - Has a much smaller form factor
  - Consume less power
  - Better signal to noise ratio, i.e. better recording quality.

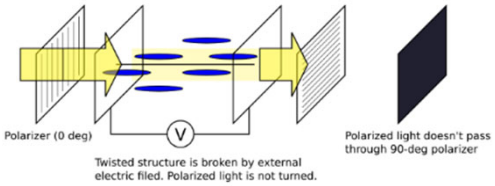
Liquid Crystal Display (LCD) Basics

Polarised light twisted (Light Passes)



Source: awawa.hariko.com

Polarised light not twisted (Light Blocked)



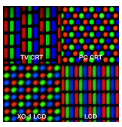
- Passive display technology, i.e. they don't emit light. Instead, they need a backlight in order for user to see the image.
- Liquid crystal is an organic substance that has both a liquid form and a crystal molecular structure. The rod-shaped molecules are able to kept their order in a particular direction 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.

Oh Hong Lye / Cx1106 23

- The discussion here is the basic principle behind a passive LCD, the LCD technology you see in the market has various enhancement applied over this basic design.
- Passive LCD, as the name implies, does not emit light and requires a backlight in order for user to see the image.
- Liquid crystal is a substance that has both liquid and crystal property.
    - The rod-shaped molecules are able to keep their order in a particular direction although they are in a liquid state.
    - And the orientation of these molecules can be controller with an electric field.
- These molecules are able to twist the orientation property of the light passing through them.
- Next we need to introduce another important module of the LCD panel, the light polariser.
    - A light polariser allows light that has the same orientation as the polariser to pass through
        - Lights that has orientation that are 90 degree with respect to the polariser will be blocked completely
        - Any other orientation will have partial component of the light passing
- With a setup consisting of two polariser oriented 90 degrees from each other, we will be able to control the amount of light passing through the setup by applying electric field to change the orientation of the LCD molecules.
    - If the LCD molecules twist the light by 90 degrees, then all the light will pass through the second polariser.
    - If the LCD molecules keep the orientation of the light unchanged, then the light will be blocked by the second polariser.

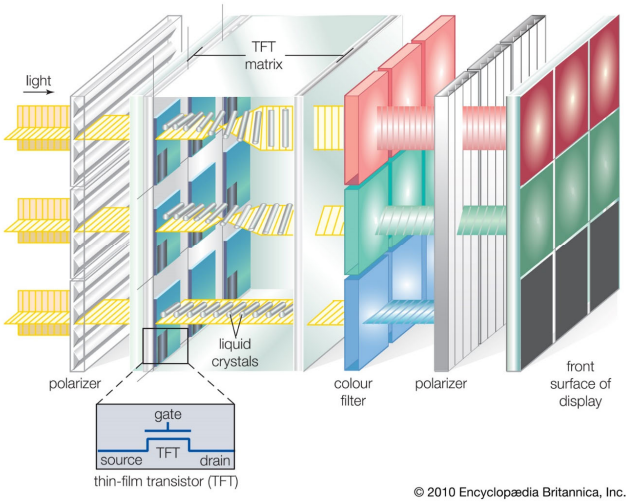
Colour LCD

Source: Wikipedia



RGB Pixels Pattern on Colour LCD

Colour LCD

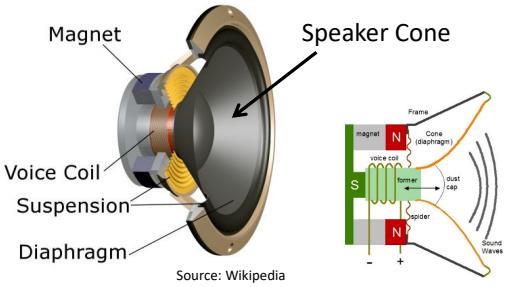


© 2010 Encyclopædia Britannica, Inc.

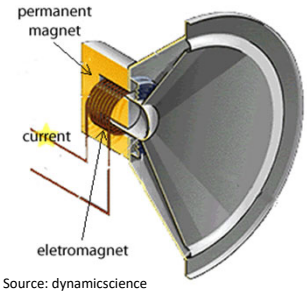
- Process for colour LCD is similar to that discussed in previous slide.
- Each pixel is associated with three colour filters to project the RGB colours.
- A varying orientation of molecules in the liquid crystal suspension. varies the amount of light allowed to pass through to the colour filter, thereby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

- Applying the LCD operating principles we learnt in the previous slide for the case of a colour LCD.
- The primary colours are Red, Green and Blue.
  - Any colour can be derived by mixing R, G, B in different proportions.
- Using this concept, we can have three light sources, one each for R. G and B, enabled by passing the light source through the respective colour filter.
- Each of these colour component will pass through the first polariser, LCD and second polariser setup that we discussed in previous slide.
  - By varying the electric field of each of the light source's LCD suspension, we can vary the amount of light for each colour component and hence derive the intended colour.

Audio Speaker



Source: Wikipedia



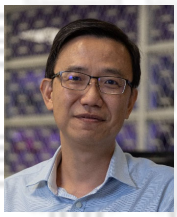
Source: dynamicscience

- 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.

- Audio speaker transduces electrical energy to sound energy.
- The speaker cone is a moveable structure attached to an electrical coil called the Voice Coil.
- The voice coil is surrounded by a permanent magnet and moves according to the amount of electric current passing through coil.
- This in turn causes the speaker cone to move back and forth to create the sound waves which is heard as sound by the human ear.
- This same principle is used in the headphones as well.

Cx1106
Computer Organization and Architecture

Processor Pipeline

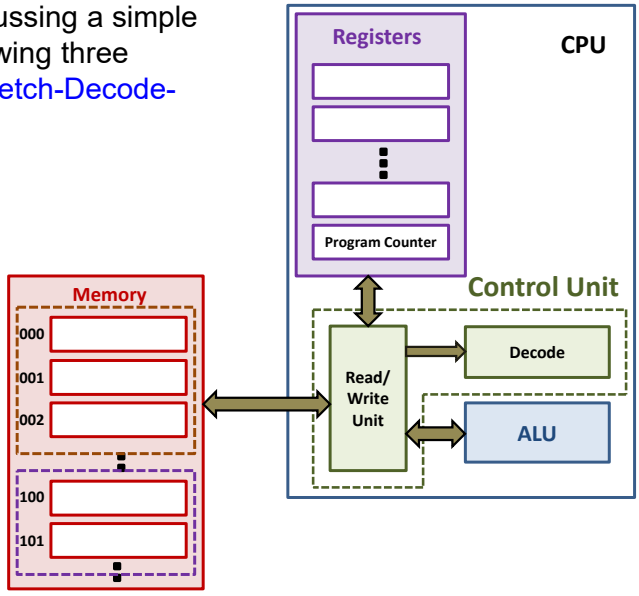


Oh Hong Lye
Senior Lecturer
School of Computer Science and Engineering, Nanyang Technological University.
Email: hloh@ntu.edu.sg

- This chapter is on processor pipeline.
- Pipeline is a processor architecture that split an instruction into a few stages, each performing an operation that is simpler than a typical CPU instruction.
- The pipeline allows CPU to be clock at a faster rate and have a higher performance. More details in later slides.

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

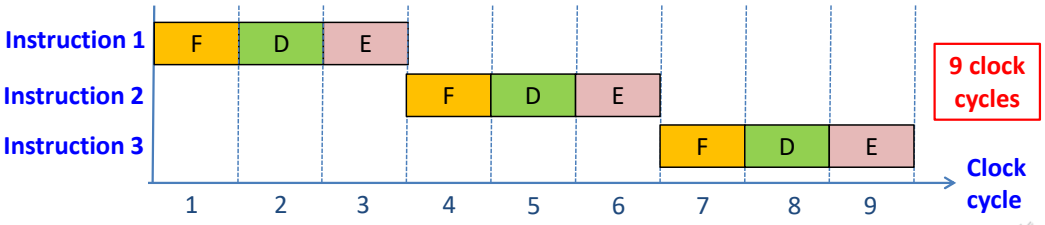


Oh Hong Lye / Cx1106 2

- So far, when we discuss execution of CPU instructions, we use a simple CPU architecture where the current instruction is executed completely before the next instruction is fetched from the memory to be executed.
- If we looked at the execution of a CPU instruction, we could roughly split it into 3 operations,
  - the machine code of the instruction needs to be fetched from the memory,
  - the CPU will decode the machine code to find out what instruction doe the machine code correspond to
  - After the CPU know what the instruction is, it can proceed to perform the actual execution.
- This correspond to the Fetch, Decode and Execute operations.
- In a simple CPU architecture, the Fetch, Decode and Execute operations of a particular instruction needs to 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.

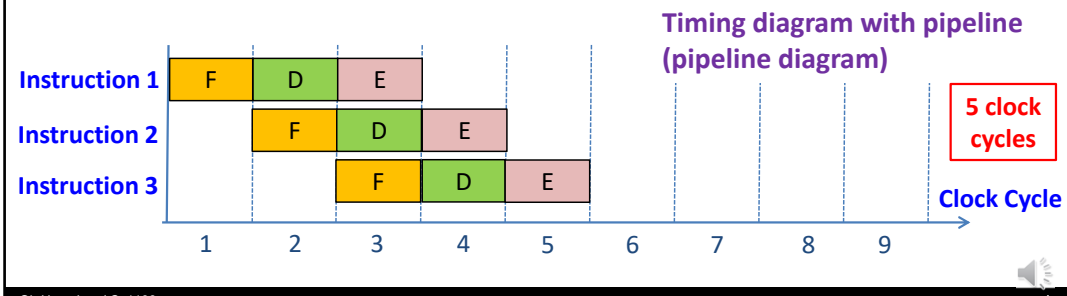


Oh Hong Lye / Cx1106 3

- Extending on the discussion we have in previous slide, when a simple CPU execution an instruction, it goes through 3 stages:
  - Fetch
  - Decode
  - Execute
- If each of these stages takes one clock cycle to execute, and since the fetch for the next instruction can only be executed after completing the current instruction,
  - then three CPU instructions will take 9 clock cycles.

Executing each stages in parallel

- 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.

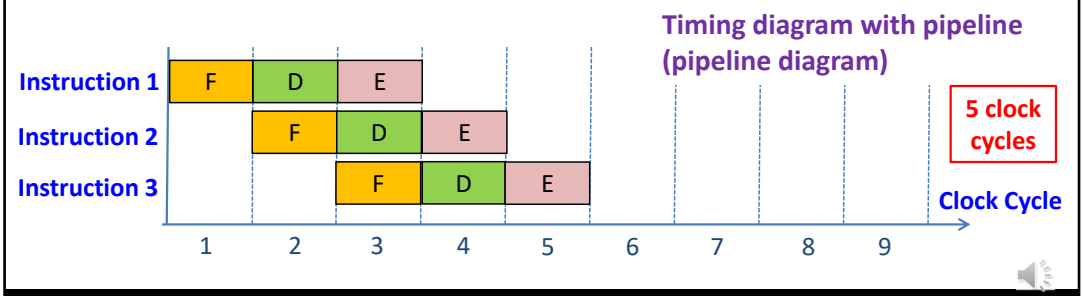


Oh Hong Lye / Cx1106 4

- Now, if a processor is able to execute the Fetch, Decode and Execute stage simultaneously, then the outcome will be very different.
- The following animation will illustrate the difference.
- The machine code for instruction 1 is fetched in the first clock cycle
- In the 2nd clock cycle, the processor will be decoding the machine code of instruction 1 and fetching the machine code of instruction 2 simultaneously.
- In the 3rd cycle, processor will be executing instruction 1, decoding instruction 2 and fetching instruction 3, all at the same time.
- The result is that this processor only needs 5 clock cycle to execute 3 instructions. This vs 9 clock cycles for simple CPU architecture.

Pipeline Architecture

- This is an example of a Pipeline Processor Architecture.
- Pipelining is about partitioning an instruction into simpler stages and assigning resources to allow these stages to be executed simultaneously.
- There are many ways to partition an instruction, the example shown here is one of the simplest. Its not uncommon for more complex application processors to have more than 10 pipeline stages.

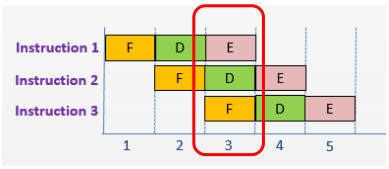


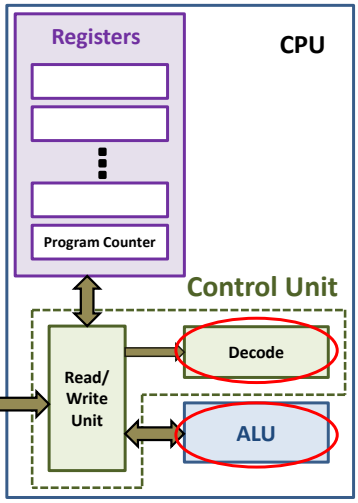
Oh Hong Lye / Cx1106 5

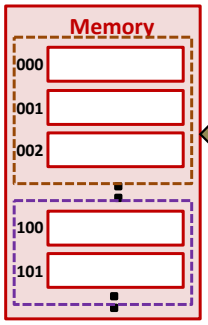
- Pipeline processor split an instruction into simpler stages and assign resources to execute these individual stages simultaneously.
- Splitting into simpler stages allows the CPU to be clock at a faster rate, since the operation at each stage is now easier.
- There are many ways to split an instructions into simpler stages. The Fetch, Decode and Execute stages discussed is only one of the ways.
    - It is not uncommon to have more than 10 pipeline stages in the more complex processors.

Pipeline – A peek into Processor Internal

- Pipelining is possible if the Fetch-Decode-Execute operations use independent resources
- For example
  - Fetch (External buses)
  - Decode (Instruction Decoder)
  - Execute (ALU)







Fetch Instruction

Decode Instruction

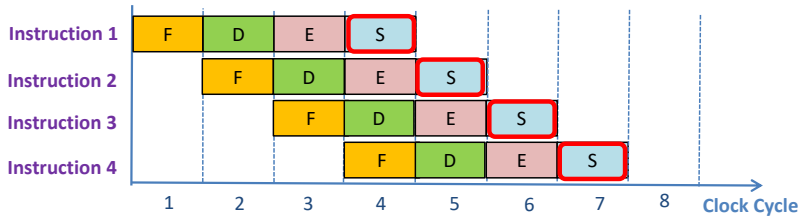
Execute Instruction

Oh Hong Lye / Cx1106 6

- This slide shows the processor internal operations when handling the pipeline operation.
- During the fetch stage, the processor use the system bus to read the instructions from the memory
- The Decode stage uses the instruction decoder in the processor to decode the machine code.
- During the execute stage, the processor use the ALU for computation.
- Notice that the hardware resources used in this example are independent from one another, i.e. none of the stages use the same hardware resource.
  - This allows all 3 stages of the pipeline to be executed simultaneously, as shown at the 3rd clock in the pipeline diagram.
- This illustrates one important criteria in order to maximise the efficiency of a pipeline, i.e. each of the pipeline stages should use independent resources, in other words, these stages should not be using the same resources.

Pipeline Efficiency

- 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.



Oh Hong Lye / Cx1106 7

- Now, a pipeline's efficiency is maximised when the pipeline is filled, i.e. all the stages are in use.
- But it takes time to fill up a processor pipeline, the longer the pipe line i.e. number of stages, the longer it takes to fill it up.
- For example, in a processor with 4 pipeline stages, Fetch, Decode, Execute and Store, where
  - Fetch refers to fetching the instruction from the memory
  - Decode refers to decoding the machine code to know what instruction was fetched
  - Execute refers to performing the actual execution such as ADD, SUB etc
  - Store here refers to the storing of the results from the Execute Stage to registers or memory.
- From the pipeline diagram, we can see that it take 4 cycles to fill up the pipeline,
  - But once the pipeline is filled, the processor is able to release one instruction every clock cycle.
    - This gives the effect of processor executing one instruction every clock cycle.
    - Using the above scenario, executing one instruction in a simple CPU is equivalent to executing 4 pipeline stages in a pipeline processor above so takes 4 times as long compared to the pipeline processor.

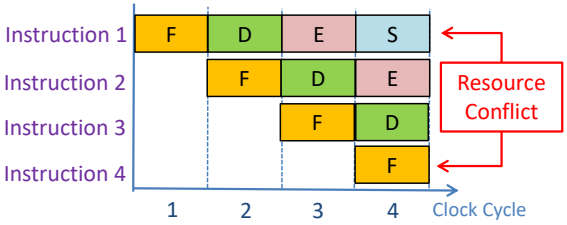
Pipeline Conflicts

- 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

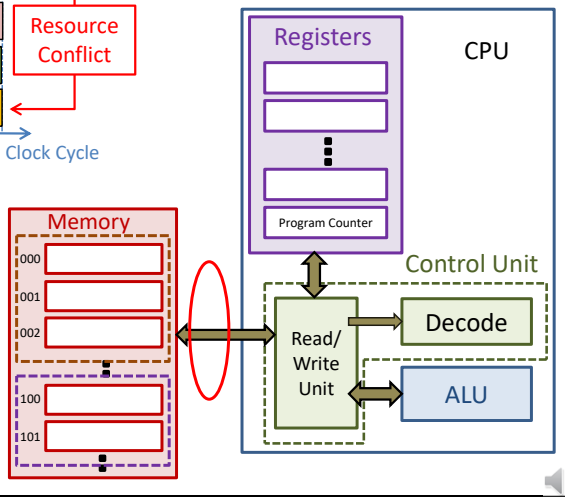
- From previous slides, we saw that pipeline efficiency is maximised when all the stages in the pipeline are filled or occupied.
- There are however, events that may disrupt the pipeline, causing
  - some stages to be halted temporarily, or
  - Some instructions to be flushed or discarded, meaning the processor will need to refill the pipeline stages again.
- These events are known as pipeline conflicts
- The pipeline conflicts as described above will result in more time needed to execute one or more instructions.
  - Since more time is needed to execute a set of instructions, it implies that the number of instructions that can be released from the pipeline is reduced.
  - As a comparison, when the pipeline efficiency is maximised, it would be able to release one instruction every clock cycle.
- In this module, we will look into three sources of pipeline conflict
  - Conflict due to Insufficient Resource to operate each pipeline stages simultaneously
  - Conflict due to dependency of data between instructions, which leads to additional cycles needed to mitigate the dependency.
  - Additional cycles needed to execute a branch operation because some instructions that were pre-fetched into the processor has to be discarded..

Resource Conflict

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



Example: Instruction 1 is storing result in memory when instruction 4 needs to fetch a new instruction



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

Oh Hong Lye / Cx1106 9

- First, let's look at the resource conflict, which happen if the processor do not have sufficient resources to operate all the pipeline stages simultaneously.
- The diagram in the slide illustrate this point clearly.
  - Consider a 4-stage pipeline processor with Fetch, Decode, Execute and Store stages,
  - At clock 4, the processor is fetching instruction 4 and performing a write to the memory.
    - Now instruction fetch involves reading the memory via the system bus, and
    - Store involves writing to the memory via the same system bus.
  - So if the processor only has one system bus, there will be a conflict between the two parties trying to use the system bus at clock 4.
- In summary, a resource conflict occurs when two instructions attempts to access the same resource in the same clock 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.

- There is really no other alternative in resolving resource conflict other than giving the processor what it needs to do its job.
- So you need to provide sufficient resources for a pipeline processor to work properly and efficiently.
- If the processor doesn't have the resources it needs to operate all the stage simultaneously, then at some point in time, one pipeline stage will have to wait for another and that will disrupt the pipeline operation.
- In the worst case, performance of the pipeline processor may be worse than the simple CPU architecture, due to the constant halting or flushing and refilling of pipeline stages.
- It is common for pipeline processors to have multiple resources of the same type. E.g.
    - the processor may have multiple internal buses for transferring address, data and instruction information.
    - It may also have multiple processing units to allowing different arithmetic operations to be carried out simultaneously.

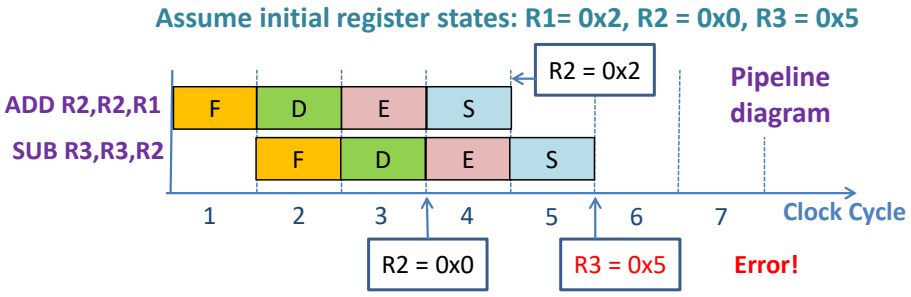
Use of ARM instructions in Pipeline Examples

- 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.

- For the pipeline and the computer arithmetic topic, I'll be using the ARM assembly instructions that you have learn during the first half to illustrate some of the concepts in these two chapters.
- But note that the pipeline structure discussed in this chapter is not that of the ARM processor.
- To simplify the analysis, all instructions used in the pipeline discussion are assumed to occupy only one word and each pipeline stage take one clock cycle to execute.
  - The pipeline operation will be more complicated if multi-word instructions are involved as these instructions may required multiple cycles to clear one or more of the pipeline stages.
- Lastly, the suffix 'S' of ARM instructions will not be used here, or rather you can considered it to be enabled by default. That means MOV = MOVS, ADD = ADDS etc. All instructions will affect the conditional flags.
  - Our focus in this chapter is pipeline, not ARM assembly instruction operation.

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).



Oh Hong Lye / Cx1106 12

- The second conflict we will be discussing is data dependency.
  - Data dependency occurs when the current instruction has an operand that has not been updated in time for the current instruction's execution, and that resulted in wrong result being generated.
- I'll illustrate this with an example.
  - Consider a 4-stage pipeline processor that we discussed in previous slide, Fetch-Decode-Execute-Store.
  - The initial state of the registers are as indicated in the slide.
  - The processor is executing two instructions as shown. ADD R2, R2, R1 followed by SUB R3, R3, R2.
  - At clock cycle 3, SUB instruction is about to enter into the E stage where it will perform subtraction of R2 from R3. But notice that at this point in time, the content in R2 is still the old value, i.e. 0, instead of the updated value of 2.
    - This is because the ADD instruction will only update R2 at its S stage, which is at clock 4.
  - Therefore, the result of the SUB instruction will be wrong, it'll be 5 instead of 3.
  - This is a classic example of data dependency.
    - Here, SUB instruction uses R2 as one of its operand, but R2 is the destination register of an earlier instruction.
    - And because the update of R2 is not in time, the old value of R2 is used in the SUB instruction and the result is wrong.

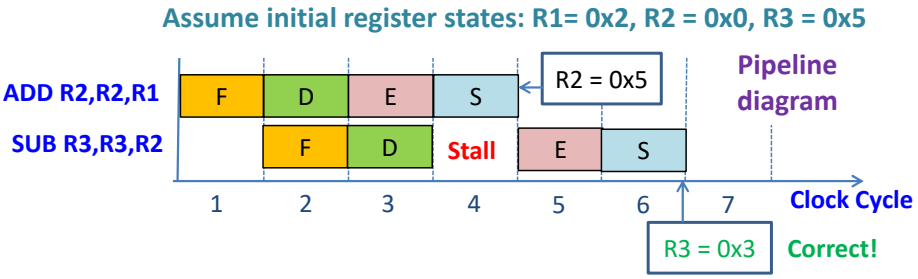
Data Dependency Conflict

- For the example in the previous slide, it will not be an issue if the processor has a simple CPU architecture where instruction execution is sequential. The current instruction will start its execution only after the previous instruction has complete. So the most updated copy of the memory/register value is always available to the current instruction.
- Due to the overlapping nature between instructions within a pipeline, there will be instances where a data required for the proper execution of the current instruction has not been updated yet.
- This is known as data dependency issue/conflict. It arises when the source operand of the current instruction is also the destination operand of a prior instruction.
- The exact separation between the two instructions in order for the issue to occur depends on the pipeline structure and design.

- If the processor in the previous slide has a simple CPU architecture instead of a 4-stage pipeline, then data dependency issue will not occur because a simple CPU will execute ADD instruction completely, including the storage of result to R2, before it starts to execute the SUB instruction.
- But this is not the case for pipeline processor as it overlap the execution of adjacent instructions within the pipeline.
- Which means there will be scenarios where data required by current instruction has not been updated yet.
- This as discussed in previous slides, is known as data dependency.
- Note that the exact separation between two instructions in order for data dependency to occur will change depending on the pipeline structure and design.

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) in the Decode stage.

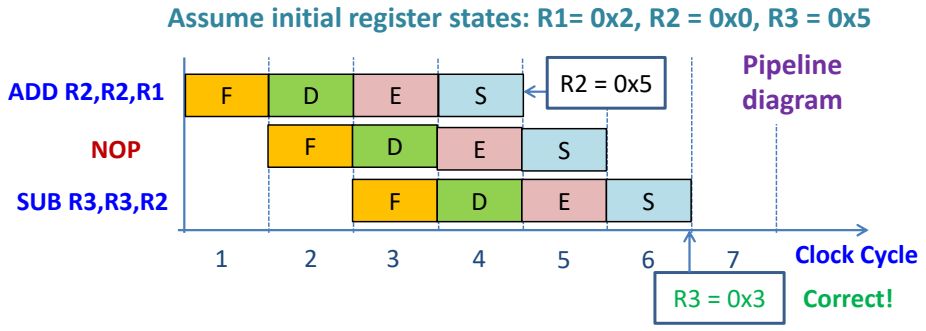


- If data dependency is detected (i.e. R2 matches), allow ADD to continue normally, but stall the Decode stage of SUB.
- After ADD completes, SUB is allowed to resume.

- There are two ways to resolve the data dependency.
- First method is to design the hardware to detect data dependency between instructions and stall one of the pipeline stage temporarily to allow the required data to be updated before executing the current instruction.
- Using the previous example, when hardware detected the data dependency between the ADD and SUB instructions, it will stall the D stage of the SUB instruction for one cycle to allow R2 to be updated, then proceed with execution.

Resolving Data Conflict – Insert NOP Instructions

- 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



Oh Hong Lye / Cx1106 15

- Another way to resolve data dependency issue is to have the compiler insert additional NOP instructions between instructions with data dependency.
- This will delay the execution of the later instructions so that its operands will be the updated values.
- As shown in the diagram, the additional NOP instruction between ADD and SUB instruction causes the E stage of the SUB instruction to be delayed to clock 5, this enable R2 to be updated to the correct value before E-stage of the SUB instruction.
- Note that NOP instruction is an assembly instruction that does nothing, it is typically used as a place holder to delay the operation of the processor by one cycle.

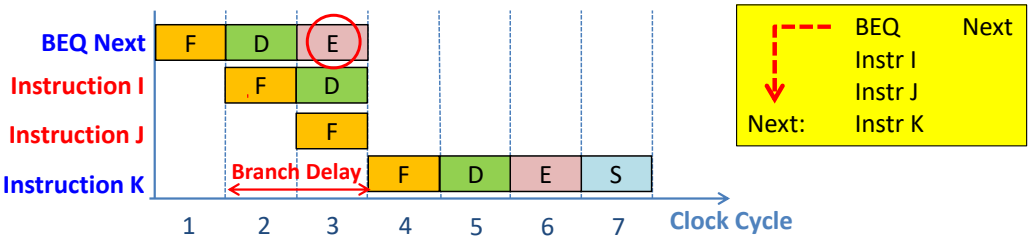
Branch Instruction

- 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.

- Next we come to the last pipeline conflict we will be discussing for this course, the branch delay.
- A typical conditional branch instruction need to perform two operations
  - Evaluate the branch condition to decide whether the branch is to be taken or not.
  - Calculate the branch target
- Both operations require the use of an ALU to perform some computation so a natural stage in which these operations are carried out is at the E stage.
- However, due to the overlapping nature in which instructions are processed within a pipeline, unnecessary instructions may have been introduced into the pipeline before branch decision is made.
  - This means the processor will need to discard these unnecessary instructions so as to maintain proper execution of the program code.
  - Such flushing of pipeline introduce delays as some cycles are wasted.
- The number of cycles lost is known as the branch delay.

Branch Delay

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

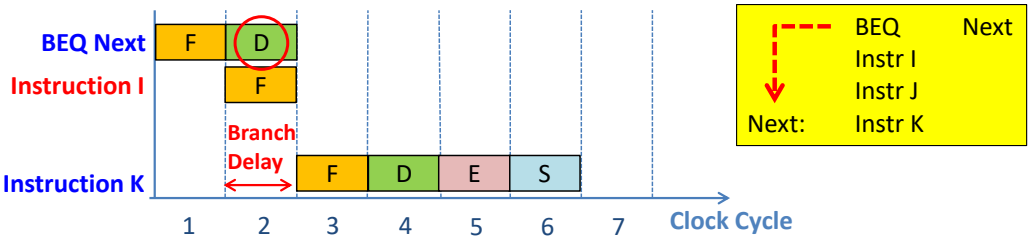


- The branch target is only known after the Execute stage, but by this time, Instructions I and J have already been fetched.
- Instructions I and J will be discarded, resulting in two-cycle branch delay.
- The two slots that are discarded is known as the Delay Slots
- Instruction I and J is known as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

- This slide illustrate the concept of branch delay discussed in the previous slide.
- Consider the code on the right, the processor pipeline will fetch these instruction sequentially into the pipeline as shown.
- At clock 3, BEQ instruction will be at its E-stage where the branch decision is made.
  - If the branch is true, then instruction K will be fetch in the next cycle as it is the next instruction to be executed after BEQ.
  - However, instruction I and J would already been fetched into the pipeline at clock 3.
  - I and J have to be discarded in order to maintain the proper execution of the program.
  - This incurred a 2-cycle branch delay.
- I and J are known as delay slot instructions.

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.

- In the previous slide, the branch decision is made at E stage and that resulted in 2-cycle branch delay.
- We can reduce the branch delay to one cycle by bringing forward the branch decision making to the D stage.
- Since performing branch decision requires an adder, one additional adder needs to be introduced into the processor to support this feature.
- With the branch decision done at D stage, we can see from the pipeline diagram that only one cycle branch delay is incurred.

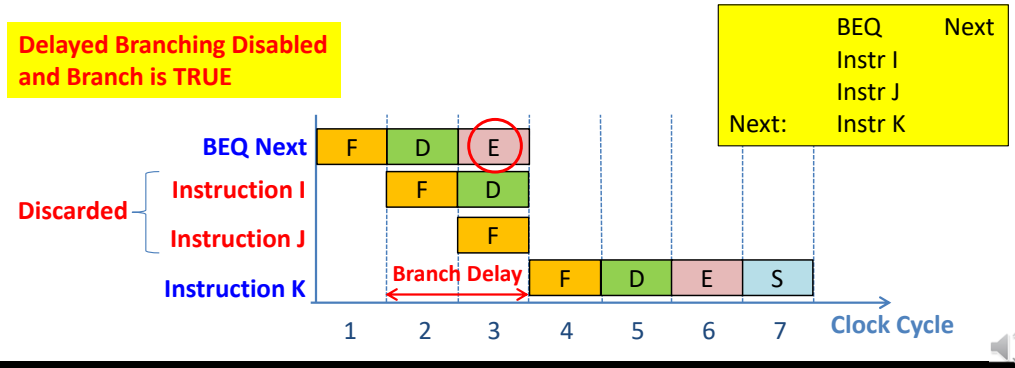
Delayed Branching

- In the previous implementation, the instruction(s) immediately following a branch is always fetched, regardless of the branch decision.
- If branch is taken, these instruction(s) will be discarded resulting in branch delay.
- Delayed Branching is a method that ensures no instructions are discarded after the branch. That means delay slot instructions are always executed.

- Branch delay reduces pipeline efficiency as delay slot instructions are discarded.
- To reclaim back this efficiency, delayed branching can be used.
    - When delayed branching is enabled, the processor will not discard the delay slot instructions.
- But if the original order of instructions is used with delayed branching, the program execution will be wrong. In other word, we resolved the pipeline inefficiency issue by getting the processor to execute delay slot instructions, but in doing so, we broke the logic flow of the original program.
- We will illustrate this point again in the next slide.

Delayed Branching

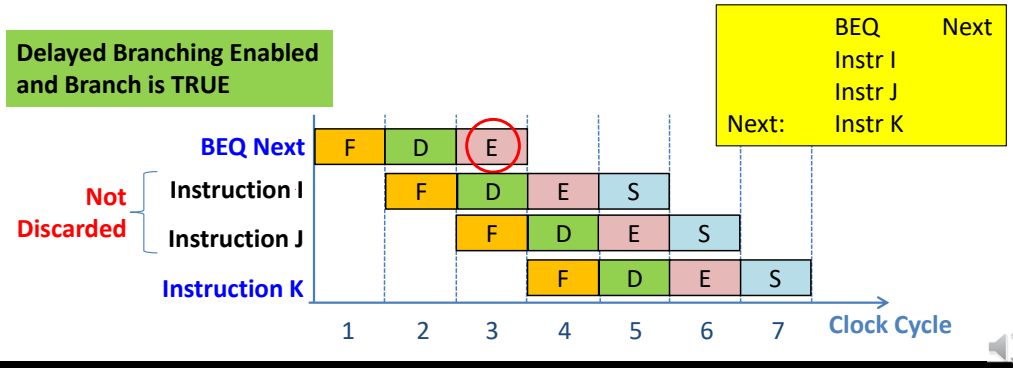
- For simplification sake, we can view Delayed branching as a feature in the processor
- If Delayed branching is disabled, the processor pipeline will discard the delay slot instructions if the branch is True to preserve the correctness of the program logic, at the expense of cycle wastage.



- The pipeline diagram here illustrate the case where Delayed branching is disabled, i.e. delay slot instructions are discarded when branch is true.
- The instructions are discarded to maintain the proper program logic, i.e. after BEQ, instruction K will be executed. I and J should not be executed.

Delayed Branching

- If the Delayed branching is enabled, the processor will execute the delay slots instructions regardless of true or false branch decision so no cycles are wasted.
- However, the program logic would be wrong if instruction I and J are executed, when the branch is True.
- So, some other instructions (other than I and J) need to fill the delay slots.

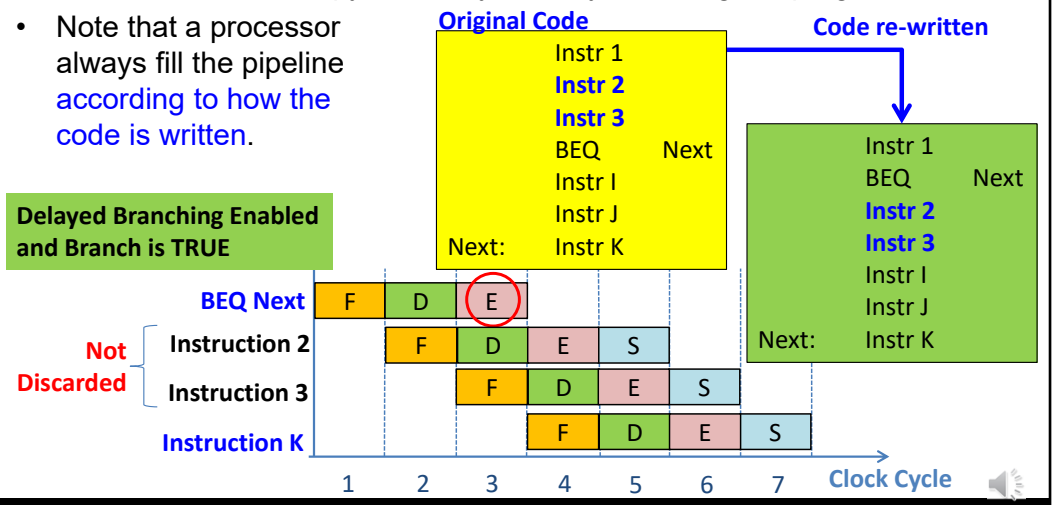


Oh Hong Lye / Cx1106 21

- With delayed branching enabled, the processor will execute delay slot instructions to completion.
- If we kept everything as per the previous slide, that means instruction I and J will be executed even when branch is true, which is obviously wrong from program logic point of view.
- That means there is a need to re-arrange the program code so that some other instructions will get loaded into the delay slots instead of I and J, and yet allow a proper program logic flow.
- Note again that processor will load the pipeline according to how the code is written.

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.



Oh Hong Lye / Cx1106 22

- If we expand the code a little to include more instructions, we have instruction 1, 2, 3 followed by BEQ followed by instruction I, J, K. As shown in the yellow box.
- This code needs to be re-written in order to use the delayed branching feature.
- The re-written code is shown in the green box. You can see that we have brought two instructions to below BEQ.
    - So would this code behave similar to the original program logic flow?
- Let's go back to the yellow box which has the original code. The program logic will have two paths corresponding to case where branch is true and false.
    - If branch is true, execution sequence will be instruction 1, 2, 3, BEQ and K.
    - If branch is false, execution sequence will be instruction 1, 2, 3, BEQ and I
- Moving over to the green box and with delayed branching enabled,
    - If branch is true, execution sequence will be instruction 1, BEQ, 2, 3 and K
    - If branch is false, execution sequence will be instruction 1, BEQ, 2, 3 and I
- You can see that other than the execution sequence of instruction 2 and 3, basically the same group of instructions are executed for the case of true and false branch.
    - The fact that instruction 2 and 3 is executed after BEQ means that instruction 2 and 3 has to be independent instructions that does not affect the branch decision making process.

Delayed Branching

- 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.

- As discussed in the previous slide, when delayed branching is enabled, delay slots has to be filled with independent instructions that doesn't affect the branch decision making process.
- If such independent instructions cannot be found, then the delay slot has to be filled with NOP instruction.

Delay slot Instructions

- Needs to be Independent instructions which does not play a part in the branch decision making process.
- Another requirement for delay slot instructions is that th executed in the original program flow, regardless of wh taken or not.
- There are a few general rule of thumb
- It should be some instructions that are earlier sequence program, compared to the branch instruction. Any instr than the branch instruction in the original program would incidentally be executed or not executed depending on the branch decision.
- Its operation should not have effect on status of any registers that would in turn affect the branch decision. e.g. if a BEQ is used, the independent instruction should not affect the Z flag which BEQ instruction used
- This is because delay slot instructions gets fully execute decision had been made, which means delay slots instr the branch decision.
- If the Branch is part of a loop, delay slot instructions ne instructions within the loop as delay slot instruction will number of iterations as well.

```
        Instr 1
        Instr 2
        Instr 3
        BEQ      Next
        Instr I
        Instr J
Next:   Instr K
```

```
        Instr 1
        Instr 2
        Instr 3
        BEQ      Next
        Instr I
        Instr J
Next:   Instr K
```

Oh Hong Lye / Cx1106 24

- Some details on delay slots instruction selection.
- These are instructions that do not affect the branch decision. On top of that, these instructions will always be executed regardless of the outcome of the branch decision.
- Some rule of thumb in choosing delay slot instructions
    - They should be instructions that are earlier in the sequence of the program code, compare to the Branch instruction.
    - Instructions which are after the branch depends on the branch decision and may not always be executed.
    - Delay slot instructions should not affect the branch decision making process. This is because when they are brought over to occupy the delay slots, they will be executed after the branch decision has been made, which means they will not be able to influence the branch decision.
        - E.g. if a BEQ is used, and instruction 3 affect the Z flag used in the BEQ instruction. This will be ok in the original program code since instruction 3 is executed before BEQ. However, if delayed branching is enabled and instruction 3 is selected to occupy the delay slots, it will be executed after BEQ and will no longer be able to affect the BEQ decision making process, which is wrong from program logic perspective.

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**

- 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

Oh Hong Lye / Cx1106 25

- One other way to mitigate branch delay is to employ dynamic branch prediction.
- There are many different prediction algorithms
- For example, the processor can maintain a branch history table to store the predicted target address of various branch instructions in the program.
- The processor will load the instructions according to the prediction,
  - e.g. using code in the yellow box, if it predict that BEQ is true, then it'll fetch instruction K after BEQ and not instruction I.
  - If it predict BEQ to be false, it will load instruction I instead of instruction K.
- If the prediction is correct, no cycles will be wasted.
- If prediction is incorrect, the same flushing of pipeline will occur and similar cycle wastage as the branch delay will be incurred.
- It is actually not too difficult to have a fairly effective branch prediction algorithm due to the nature of a typical program code.
- E.g. Loops are commonly found in programs, if we have a simple prediction algorithm that assume that the first branch is true and to follow the previous prediction result for subsequent iterations, then applying it on a 100 iteration loop will yield a 99% prediction accuracy.

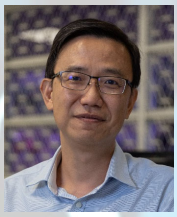
Connecting to the Real World

- 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.

- The pipeline architecture we discussed here is not the one in the ARM processor.
- Different ARM processor sub-family has different types of pipeline architecture, the one used in the MSP432 ARM processor you encounter in your Lab4 has a 3-stage pipeline: Fetch, Decode and Execute.
- On top of that, it has a branch speculation feature that fetches instructions in both the false and true branch of the program flow, one of them will be used depending on the branch decision. In other words, no branch delay incurred.
- That concludes our discussion on the chapter of pipeline processor. You will revisit this again in your Advanced Comp Arch Module.

Cx1106
Computer Organization and Architecture

Computer Arithmetic



Oh Hong Lye
Senior Lecturer
School of Computer Science and Engineering, Nanyang Technological University.
Email: hloh@ntu.edu.sg

- This chapter is on computer arithmetic.
- You have been introduced many computer arithmetic concepts in Cx1105 and also during the first half of this course so this chapter serves as a supplement of what was taught.

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

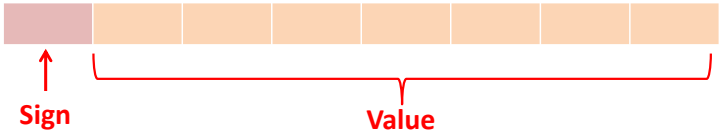
Base 2: $10101_2 = (1 \times 2^4) + (0 \times 2^3) + (1 \times 2^2) + (0 \times 2^1) + (1 \times 2^0) = 21_{10}$

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

- The decimal, binary, hexadecimal number system that you have been using are members of a number representation system known as the positonal numbering system.
- In the positional numbering system, each digit is associated with a weight.
- Let's take a look at the example below.
- For a base 10 system, the radix is 10 so each position is associated with some powers of 10.
    - Numbers to the left of the radix point, in this case, the decimal point, has an integer weightage, while those to the right of the decimal point has a fractional weightage.
- Similarly for base 16, which is the hexadecimal system you have been using.
- And lastly the base 2, which is the binary system.
- In particular, binary is the basis of computer memory. Primarily because it can be easily represented using transistor ON/OFF state.

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

- The rest of the discussion in this chapter will focus mainly on the binary system.
- First, how do we represent signed integers in a binary system?
- In computer system, the MSB of a binary number is typically used to indicate the sign of a number.
- 1=>negative, 0=> positive.
- There are a few ways in which binary numbers can represented, we will focus on the most commonly used representation system which is the 2's complement number representation system.

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

Oh Hong Lye / Cx1106 4

- In a 2's complement number system, the most significant bit is not just a sign bit, it also has a negative weightage as shown in the slide.
- The value of the number is similarly derived by calculating the Sum of Product of the individual bits and their corresponding weightage.
- Note again that the weightage of the MSB is negative, which is why if the MSB is a '1', the final value will always be negative.

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:

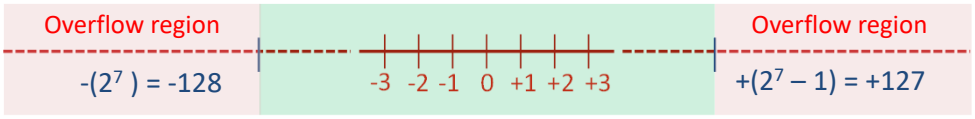
+106	**0**	**1**	**1**	**0**	**1**	**0**	**1**	**0**
+106	0	1	1	0	1	0	1	0
Invert	1	0	0	1	0	1	0	1
Add 1	0	0	0	0	0	0	0	1
-106	**1**	**0**	**0**	**1**	**0**	**1**	**1**	**0**

Negate

- 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

- As mentioned earlier, the MSB in a 2's complement number is the signed bit, MSB='1' implies the number is negative and '0' implies the number is positive.
- You can use the negation process to compute the negative equivalent of a positive number and vice versa.
- The negation process involves the following steps
  - First invert all the bits (also known as 1's complement).
  - Then add 1 to the LSB of the result and ignore any bits to the left of the MSB.
- The same process is applied when you want the convert the negative number back to positive.

Carry vs Overflow Example



Example: 48 – 19 = 48 + (-19) = 29
First, negate 19 (00010011 -> 11101101)
Then add the two numbers and discard any carries emitting from the high order bit.

1		1	1					
	0	0	1	1	0	0	0	0
+	1	1	1	0	1	1	0	1
	0	0	0	1	1	1	0	1

- Carry does not always mean we have an error
- So how do we detect overflow?

Oh Hong Lye / Cx1106 6

- We will spend the next few slides talking about the difference between the Carry and Overflow Flag in a typical processor.
- Both Carry and Overflow conditional flags were discussed in the first half.
- We understand that Carry Flag is set when there is a ‘1’ that gets carried out of the MSB of the result, as shown in the example in the slide.
- But notice that in this example, 48-19=29 and the results of computation is correct even though the carry bit is set, i.e. a ‘1’ gets carried out of the MSB.
- This means that Carry bit set does not imply that there is an error in computation for this case.
    - This case here really apply to the scenario where signed numbers are involved.
    - So Carry bit set in a signed number system doesn’t mean there is an error, it also doesn’t mean there is an overflow condition in the result.
- Which brings up the next question: how do we detect overflow condition in result? An overflow condition would imply that there is an error in the result.

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)

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

- To detect overflow condition in a signed number system, we need to check the overflow flag.
- Condition required to set the overflow flag is more complex than that of the Carry flag.
  - To detect if the overflow flag needs to be set, the processor needs to evaluate the sign bit of the result and the operands.
- The table illustrates the testing needed to detect overflow when performing addition and subtraction of two numbers A and B.
- Basically, these conditions correspond to the scenario where the results are not feasible. E.g. adding two positive numbers but getting a negative result, adding two negative numbers but getting a positive result.
- Under such condition, the overflow flag is set which implies that there is an error with the result if the system used is a signed number system.

Carry vs. Overflow

- Unsigned Numbers
  - Carry = 1 always indicates an overflow (new value is too large to be stored in the given number of bits)
  - The overflow flag means nothing in the context of unsigned numbers
- Signed numbers
  - Overflow = 1 indicates an overflow
  - Carry flag can be set for signed numbers, but this does not necessarily means an overflow has occurred.

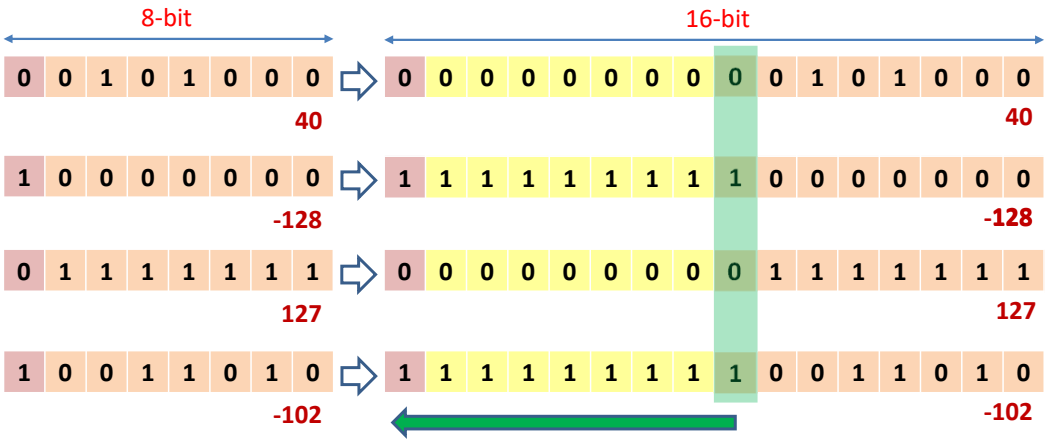
Expression	Result	Carry?	Overflow?	Corrected Result?
0100 (+4) + 0010 (+2)	0110 (+6)	No	No	Yes
0100 (+4) + 0110 (+6)	1010 (-6)	No	Yes	No
1100 (-4) + 1110 (-2)	1010 (-6)	Yes	No	Yes
1100 (-4) + 1010 (-6)	0110 (+6)	Yes	Yes	No

Oh Hong Lye / Cx1106 8

- A recap on the meaning of carry and overflow.
- In an unsigned number system,
  - Carry Flag set indicates an error as it means that the result is too large to be stored in the given number of bits.
  - While overflow flag has no meaning here.
- For a signed number system,
  - Overflow flag set implies error.
  - Carry flag may be set but it may not imply error has occur.
- The table below illustrate how the situation described could occur, e.g. In the third row, the Carry flag is set but the results is still correct. -4-2 = -6. Notice that for this case. Overflow Flag is not set.

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

Oh Hong Lye / Cx1106 9

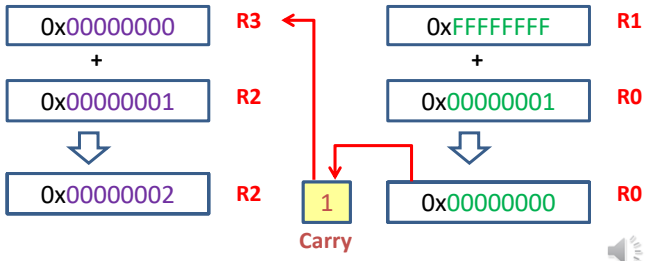
- You have also learnt about sign extension previously
- Its basically the method used when converting a smaller size to a larger size data.
- For example to convert a number stored in a 8 bit data type to a 16 bit data type variable, sign extension is used to ensure that the value in the two variables are the same.

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)

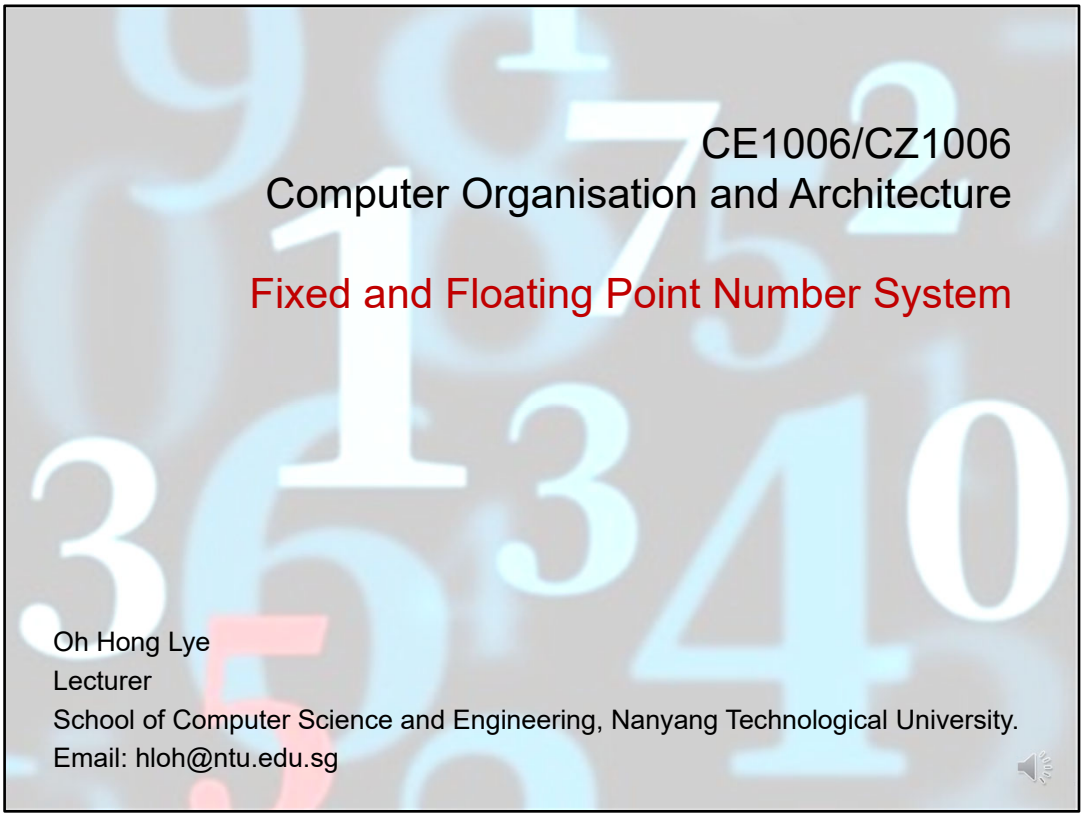
```
  00000000 FFFFFFFF
+ 00000001 00000001
  00000002 00000000
```

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



Oh Hong Lye / Cx1106 10

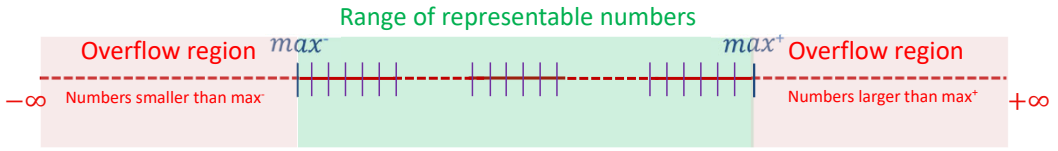
- What we have been doing are know as single-precision arithmetic, where operands are kept to the processor data width. For the 32 bit ARM processor, that would mean 32 bit operands.
- So what happen if the operands are larger than 32 bits?
- For example, what do we need to do to add two 64 bits operand?
- Solution is to perform 32bit addition multiple times, this is illustrated in the slide.
  - The ARM registers are 32bit wide so at any instance, it can only store 32bit data.
  - To add two 64bit operands, we need to add the lower order word first, followed by the higher order word.
  - Note however that when we add the higher order word, the C bit has to be added as well. Carry bit is only set if there are any '1' overflowing from the MSB of the lower order word. This is the same as the carry operation we always do with our paper caculation.
  - ARM assembly instructions wise, the instruction that add the operands and the carry bit is ADC.



- Next topic in this chapter is the fixed and floating point number system, which is the two main type of number system we typically deal with.

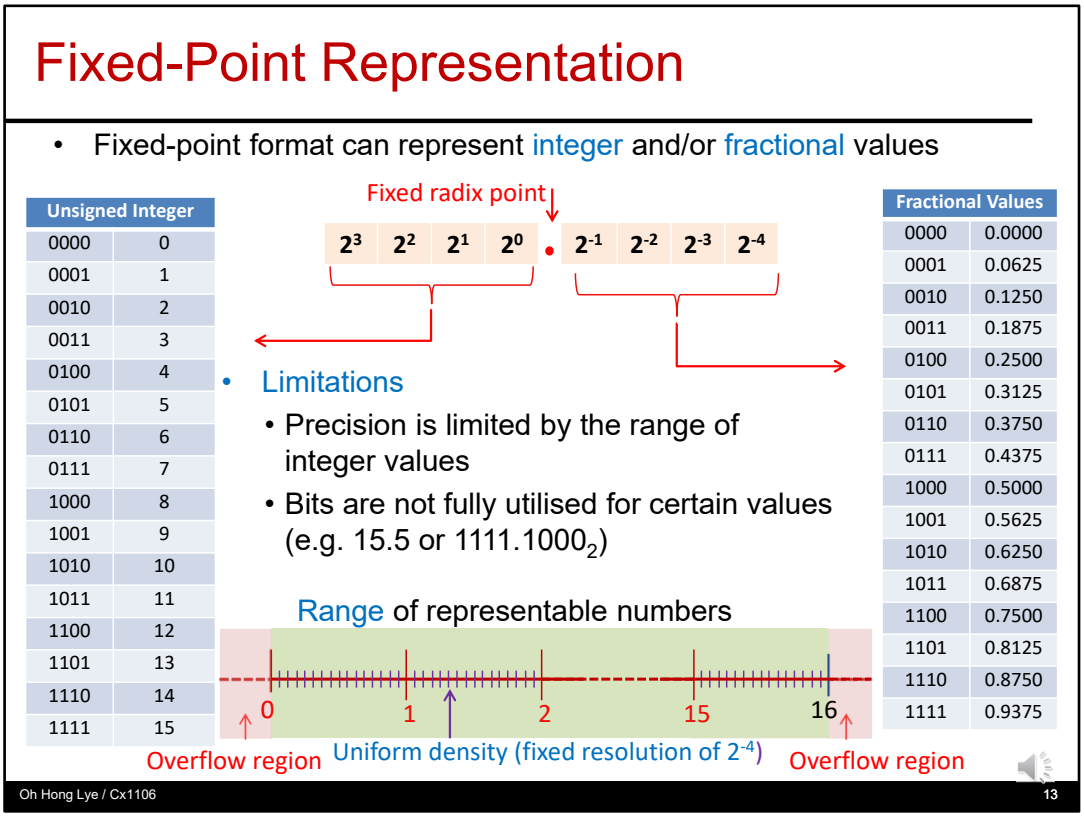
Range and Precision

- 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



Oh Hong Lye / Cx1106 12

- A few definition before we go into details of this topic.
- Range is the interval between smallest to largest representable number in a system.
- Precision, on the other hand, is the amount of information used to represent each number. E.g. 1.666 represent information in a higher precision compared to 1.67.
- Precision correspond to the interval between adjacent tick marks in the number line shown. Each tick mark correspond to a representable number in the number system used.
- In a binary number system, it correspond to increasing the LSB by 1.

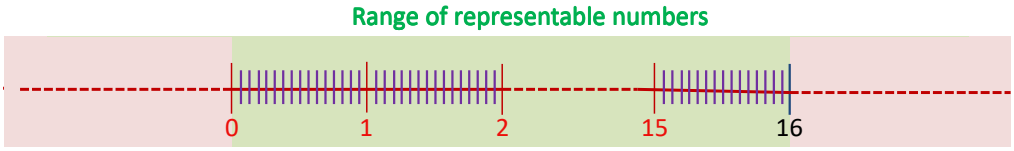


- Let's start with the numbering system which you have been using so far, the fixed point system.
- So far, we have been using fixed point number to represent integers.
    - The 4 bit integer you see here has a range of 0 to 15. Any number that is outside this range goes into the overflow region.
    - Similar to the decimal system, binary system has a radix point too and it is known as the binary point.
- And similarly, where digits after the decimal point has a fractional weightage, the bits after the binary points also has a fractional weightage.
- The smallest resolution that this system can support determine its precision, and this as mentioned earlier, correspond to its LSB.
- Fixed point system has a fixed resolution as its radix point position is fixed.
    - For this example, the resolution is 2**-4.
- Some limitation of fixed point system are
    - Given a fixed total number of bits allocated for a number, precision is limited by the range of integer.
        - If you allocate more bits for integer, the range will increase but the precision will reduce.
    - Memory used to store these data bits are not fully utilized in some cases, eg.15.5, where there are some trailing zeroes which doesn't really need to be stored.

Range and Precision Trade-off

- What happen when the radix point moves/float from one end of a number to the other end?

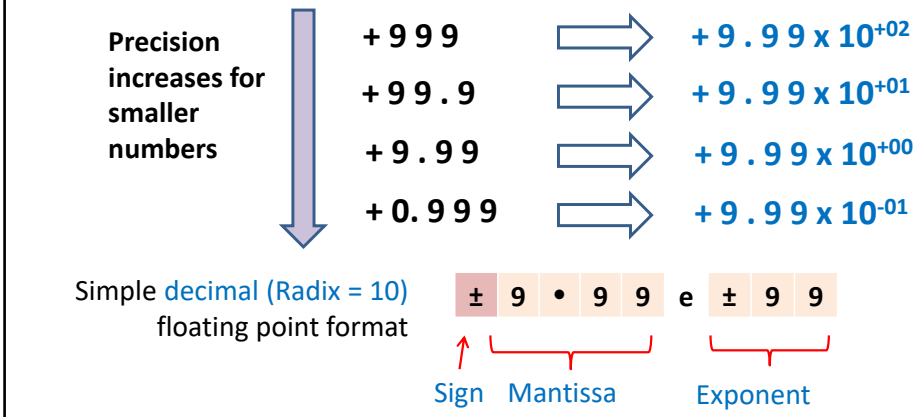
2^3	2^2	2^1	2^0	•	2^{-1}	2^{-2}	2^{-3}	2^{-4}



- When radix point floats from LSB to MSB
  - Range of representable numbers reduces
  - Precision increases
- The example above illustrates the concept of floating-point, but how do we represent a floating-point number?

- From the previous slide, we can deduce that when the radix point float from one end of the number to the other, the representable range and precision will change accordingly.
- Specifically, when the radix point move from LSB to MSB, the representable range reduces and the precision increases.
- When the range is large, we may not be too concern with the precision as the numbers we deal with are relatively large.
- But when the numbers we use gets smaller, we would be more concern with the precision now as these fractions are more significant now when compared with the numbers on the integer side.
- The above example illustrate the concept of a floating point number, in which we can dynamically choose between having a large range or high precision by shifting the radix point.
- But how can we realize such a system?

Floating Point Representation



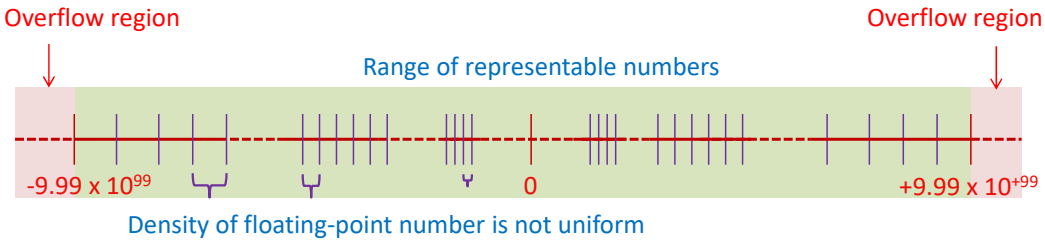
Simple decimal (Radix = 10) floating point format

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

Oh Hong Lye / Cx1106 15

- The slide here shows how we can realise the floating number system that we discussed in the previous slide.
- Take these decimal numbers with different precision for example.
- We can express these numbers in a format which has three fields called sign, mantissa and Exponent.
- By representing number in this manner, we are able to 'move' the position of the radix point by changing the exponent value.
- This is known as floating Point Number system.
- Comparing with a fixed point number, we can see that it doesn't belong to a positional numbering system, actual value is calculated by evaluating the sign, mantissa and exponent value.

Floating Point Representation



- Floating point representation can represent values across a wide range ($-9.99*10^{+99}$ to $+9.99*10^{+99}$).
- Size of exponent determines range of representable numbers.
- Size of mantissa and value of exponent determines the representable precision.
- Small numbers can be represented with good precision. When representing large numbers, precision can be sacrificed to achieve greater range.

- For a floating point number, the size of the exponent determine the range
- While the size of the mantissa, together with the exponent value, determines the precision.
- With this representation format, we are able to have good precision when number value is small, and yet could achieve large range, of course with the trade-off in precision.

Normalisation

- In the simple decimal floating-point format, there are multiple representations for the same value

±	1	•	0	0	e	±	0	1	**Normalised**
±	0	•	1	0	e	±	0	2	
±	0	•	0	1	e	±	0	3	

- Normalisation is necessary to avoid synonymous representation by maintaining one non-zero digit before the radix-point
  - In decimal number, this digit can be from 1 to 9
  - In binary number, this digit should be 1
- Normalisation can maximise number of bits of precision

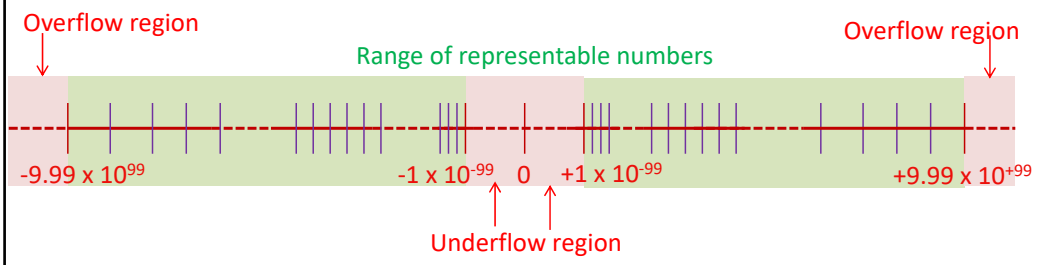
±	5	•	4	2	e	±	0	3	**Normalised**
±	0	•	5	4	e	±	0	4	

- One issue with floating point numer is that you can have multiple representations for the same number value.
- And this may potentially create confusion.
- So normalization is needed to standardise the representation.
- Normalisaton means that there should be a non-zero digit to the left of the radix point. As shown in the slide.
- Choosing this method of normalization also has an advantage of maximizing the number of bits of representable precision since the number of leading zeroes are minimised.

Underflow

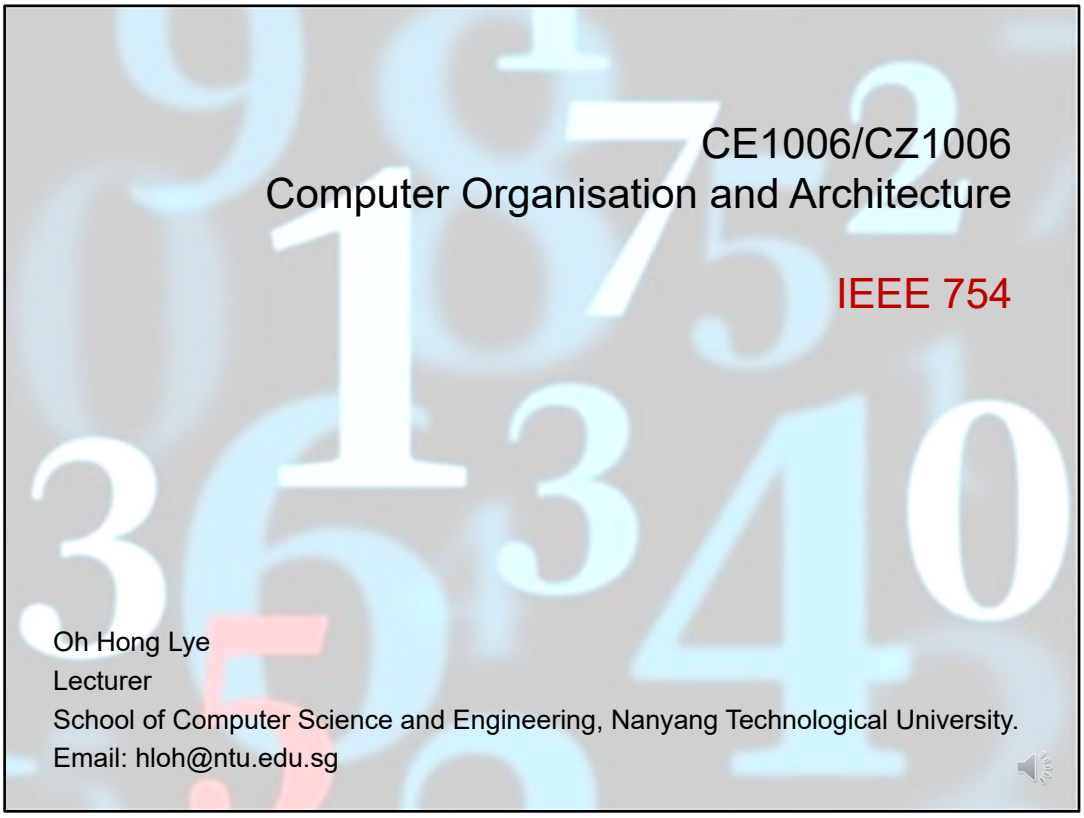
- Normalisation results in underflow regions where values close to zero cannot be represented

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.

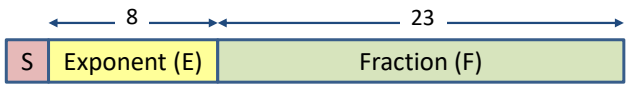
- But normalization has another side effect of creating underflow region.
- Underflow region refers to region close to zero which cannot be represented by the floating point number.
- In the example here, since the digit to the left of the decimal point has to be non zero for normalised number, that means the smallest positive normalized number is 1*10^-99, very closed to zero but not zero.
- Floating point overflow and underflow may cause the program to crash if not handled properly.
- But you'll see later that typical floating point format standard has some provision to handle the underflow scenario.



- This section is on the IEEE754 floating point number standard used in the industry.
- It is also the standard behind the floating point data type you declared in programming languages such as C.

IEEE 754 Floating Point Standard

- Found in virtually every computer invented since 1980
  - Simplified porting of floating-point numbers
  - Unified the development of floating-point algorithms
- Single Precision Floating-Point Numbers (32-bits)
  - 1-bit sign + 8-bit exponent + 23-bit fraction

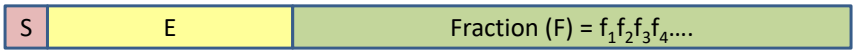


- Double Precision Floating-Point Numbers (64-bits)
  - 1-bit sign + 11-bit exponent + 52-bit fraction

Oh Hong Lye / Cx1106 20

- Because of the more complicated format, floating point system needs to be standardised.
- Else program designed by one party may not work with another.
- One of the most commonly used floating point number system is the IEEE754 standard.
- It defined both a single precision and double precision format.
- Single precision format is 32bit wide, has 1 sign bit, 8 Exponent and 23 Fractional bits. This correspond to the Float data type in C.
- Double precision format is 64bit wide, has 1 sign bit, 11 Exponent and 52 Fractional bits. It correspond to the Double data type.
- Note however that the 'Exponent' and 'Fractional' field specify in the IEEE754 format is not the Exponent and Mantissa value of a floating point number. We will touch on the details in the next slide.

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)
  - 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:
    $(-1)^S \times (\mathbf{1} + f_1x2^{-1} + f_2x2^{-2} + f_3x2^{-3} + f_4x2^{-4} + \ldots)_2 \times 2^{E-Bias}$

Oh Hong Lye / Cx1106 21

- This slide shows how the exponent and fraction field in the IEEE754 number is used to derive the actual value of the floating point number.
- Specifically
  - The Mantissa is derived by attaching a leading '1' to the left of the fractional bits. Giving the expression 1.F.
  - The actual exponent value is given by the expression E-Bias where E is the value from the Exponent Field in a IEEE754 number representation and Bias is = 127 and 1023 for Single and Double precision IEEE754 number respectively.
  - The E field in IEEE754 is a positive number, Bias allows the actual exponent to span across positive and negative number ranges.

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}

- Some working example for student to check their understanding in deriving the value of a IEEE754 floating point number.
- The first example has a sign bit = 0 so it's a positive number
  - Exponent is 178, so E-127 gives you 51
  - Plug into the expression 1.F*2^(E-Bias) gives the value 1.875*2^51.
- For the second example, sign bit is '1' so the number is negative. Final value is –1.3125*2^(-115).

Representable Range for Normalised Single Precision

- In normalised mode, exponent is from 00000001 to 11111110
- Smallest magnitude normalised number

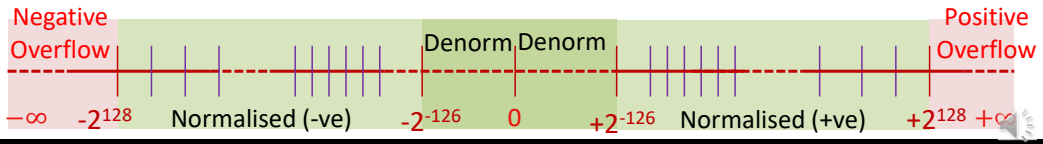
X 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

Exponent = $(00000001)_2$ = 1; E - Bias = 1 – 127 = -126
1 + Fraction = $(1.000\ldots000)_2$ = 1
Value in decimal = **1×2^{-126}**

- Largest magnitude normalised number

X 1 1 1 1 1 1 1 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1

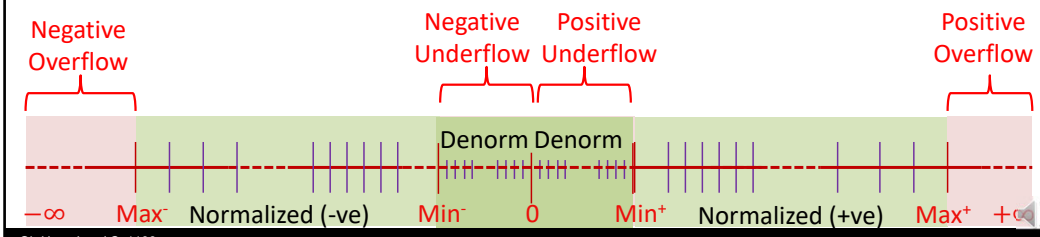
Exponent = $(11111110)_2$ = 254; E - Bias = 254 – 127 = 127
1 + Fraction = $(1.111\ldots111)_2 \approx 2$
Value in decimal = $2 \times 2^{127} \approx 2^{128}$



- The smallest normalised number for single precision IEEE754 standard correspond to E=1 and Fraction=0.
  - That gives you a value of 1*2^(-126)
- The largest normalised floating point number correspond to E=11111110 and Fraction=1111…1111 (all ones).
  - That gives a mantissa value of approximately 2 so final value is approximately 2^128.
- We can derive the range for normalised IEEE754 number using the min/max magnitude. As show in the diagram in the slide.
- Note that we didn't use the number 1111 1111 as the largest value of E, neither did we use 0 as its smallest value. Both these numbers are used for special cases in IEEE754. More on that in the next slide.

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



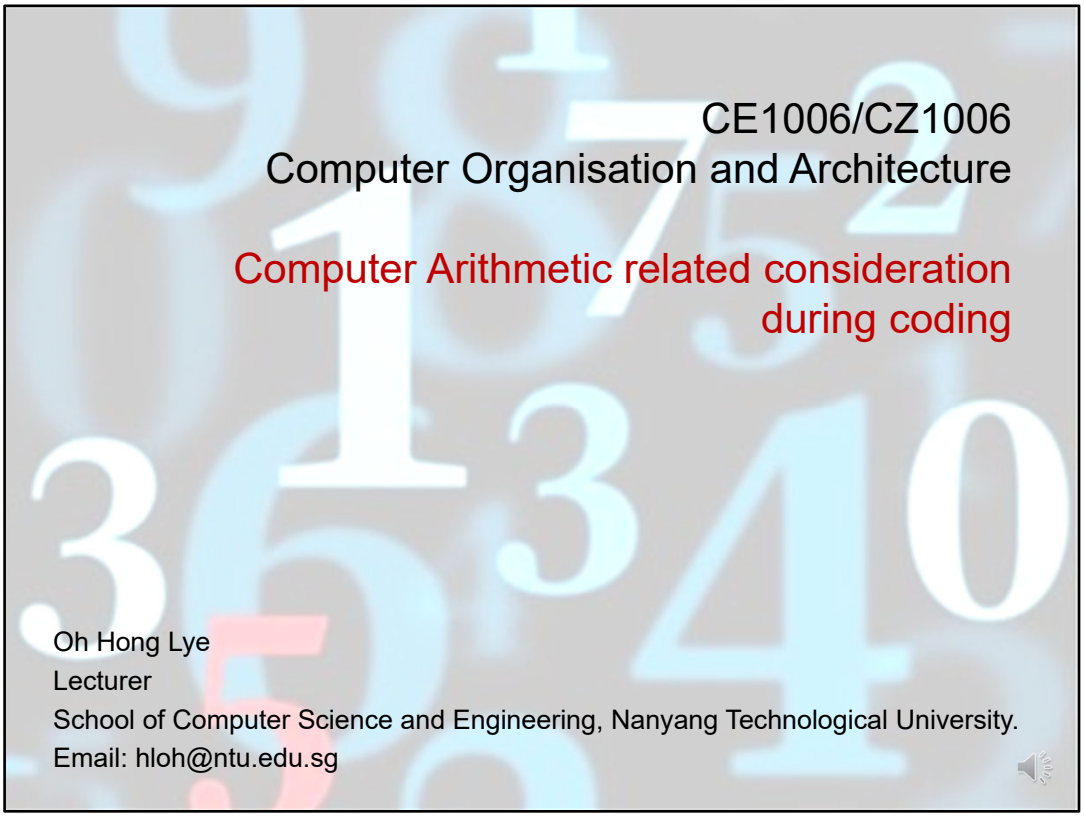
Oh Hong Lye / Cx1106 24

- As mentioned in the previous slide, some numbers are reserved for special use case in IEEE754.
- The table in this slide illustrate the various use cases.
- In normalised mode, the E value ranges from 1 to 1111 1110 while fraction can take on any value.
- Zero, Infinity and NaN are special use cases and is represented by special numbers in E and Fraction.
- IEEE754 has a special Denormalised mode that allows it to represent numbers in its underflow region.
    - In denorm mode, the MSB of the mantissa is a 0 and not a 1, i.e. 0.F instead of 1.F. Exponent is given a value zero and Fraction can be any non-zero value. Take note however that the exponent value used in the denorm mode is 2^(-126) and not 2^(-127).
    - In any case, this course will not dealt into details of denormalised mode.

Fixed Point vs Floating Point Number System

- Given the same number of bits to represent a data, e.g. 32 bits.
- Floating Point (IEEE754)
  - Max Range ≈ $2*2^{128}$ (-2^{128} to ~2^{128}).
  - Max Precision (near to zero) less than 2^{-126}
- Fixed Point
  - Max Range (Radix right of LSB, unsigned) ≈ 2^{32}
  - Max Precision (Radix left of MSB) 2^{-32}
- Floating point yield a larger range and better precision at small numbers with the same number of bits representation.
- One usually needs the best precision when the numbers are small.
- However, Fixed point number has the advantage of having uniform precision across entire range.
- Floating point number's precision changes across the range and the very coarse precision at the two end of the range may not be desirable to the intended algorithm.

- Summarising the pros and cons of fixed point vs floating point number representation
- Given the same number of data bits e.g. 32bits
  - The single precision IEEE754 floating point number, when compared to a 32bit fixed point number
    - Has a larger range
    - Is able to achieve a higher precision. This occurs when the number is near to zero.
- So floating point number yield a larger range and better precision compared to a fixed point number occupying the same amount of memory (32 bit for this case).
- However, the fixed point number has the advantage in that the precision value for a fixed point number is uniform across the entire range while that of floating point number varies across the range.
- So depending on the application requirement, you may find one is more suitable over another.



- This last section on computer arithmetic talks about some considerations to take note during coding.

Effects of four operators

- Addition/Subtraction
  - Addition/Subtraction will cause the result to increase/decrease
  - When done sufficient number of times, overflow at Min or Max end of the representable range will eventually occur
  - Take note of the range of the data type used when coding in High level language, or the width of registers and memory when coding in assembly
- Multiplication
  - Multiplication in binary is similar to an arithmetic left shift
  - Overflow at Min or Max end of the representable range
  - Similar consider as Add/Sub.
- Division
  - Division in binary is similar to an arithmetic right shift
  - Truncation of LSB leads to loss in precision
  - Reduces the magnitude of the result so it get further away from the Min/Max of the range.

- First is on the effects of the four basic operations add, subtract, multiply and divide, specifically dealing with overflow and accuracy of result.
- Addition and subtraction, if done multiple times, may potentially lead to result overflowing beyond one end of the range.
  - When that occurs, the result obtained will be wrong.
  - One example is when the result is larger than what a particular data type could hold, such as storing a value larger than 255 to the unsigned character data type,
  - Or trying to store a value larger than 255 to a 8-bit register.
- Similarly, multiplication will result in a larger number if the multiplier is larger than one, which means potentially may lead to overflow condition.
- Division with a divisor larger than one, on the other hand, gave raise to a different issue, which is the truncation of the lower order bits or digits.
  - This is easily illustrated by the fact division is similar to arithmetic right shift.
  - Any truncation of lower order bits mean a loss in accuracy of the data.
- So take note of the effects of these operators so as to have the best accuracy without running into overflow condition.

Effects of Rounding

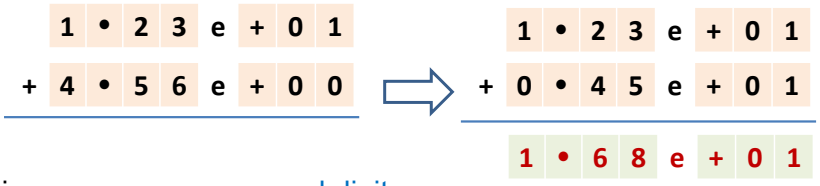
- Rounding refers to removing of LSB(s) so that the result can fit into the representable bits.
- The limits imposed in the width of the representable bits could be from the registers width, data type etc.
- Rounding can be round-up, round-down or round to nearest representable number.
- As the rounded 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 $0.004*10^1$

- 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.

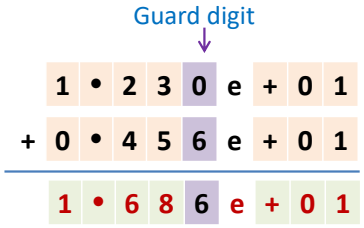
- Next is the rounding process.
- Rounding is a process in which the lower order bits of a data is truncated in order to fit the data into a register or memory location of a smaller data width.
    - Some approximation needs to be done when performing the rounding and this is by rounding up, down or to the nearest representable number.
- Since the rounded number is just an approximation of the actual data, a certain amount of rounding error is incurred in the process.
    - For example, rounding 1.686*10^01 to 1.69*10^01 result in rounding error of 0.004*10^01.
- It is common to have intermediate registers that have larger data width compared to the regular data registers in a processor. This is to preserve the intermediate data without truncating the lower order bits. Reducing the rounding error.

Rounding Error Mitigation

- When adding/subtracting two numbers, the exponents must be aligned such that they are the same



- To improve accuracy, guard digits (bits) are used to maintain precision during floating point computations. An intermediate register with a wider width could also be used.



- Rounding is then performed to ensure that the result fits into the three significant digits in the mantissa

Oh Hong Lye / Cx1106 29

- Floating point arthimetic is different from the fixed point arthimetic you have been doing previously.
- When adding/subtracting, you need to make sure the exponent has the same value in order to compute the correct result.
- Guard bits are used to maintain the precision during computation before doing rounding at the last step.
- The additional Guard digit in the example allows the least order digit 6 to be preserved.
- This in turn allows a more accurate result to be obtained after performing the rounding. For this case, the final result 1.69*10^1 is closer to the actual result of 1.686*10^1, compared to the previous result 1.68*10^1 obtained when there is no guard digit to store the least order digit ‘6’.
- Actually, the same method is applicable to fixed point numbers as well, always try to maintain as many bits of information during computation and perform any rounding or any operation that will result in bit truncation last.
    - For the simple reason that any bit truncation means losing data bits and losing data bits means losing precision.

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 • 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

Oh Hong Lye / Cx1106 30

- As data width is finite, so computer typically face limitation
  - when dealing with very large number as it may not have enough bits for required range.
  - Or very small numbers where there may be insufficient bits to achieve the required precision.
- On top of that, we often need to do rounding so as to fit the data into the limited number of bits available.
  - This introduces rounding error.
- In this example, we are performing addition of a few small numbers to a large number.
  - Recall that we need to align the exponent for floating point number addition.
  - If we perform a rounding after adding the first data, the '1' at the end would be truncated.
- This sequence will happen for the rest of the additions. Which means we are going to get the same 1.23*e^3 at the end of the day and this is not the answer we wanted.
- What should we do instead?

Handing numbers with different magnitudes

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

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

```
  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
```

Oh Hong Lye / Cx1106 31

- A different result would be obtained if we change the sequence of accumulation.
- If we allow the small number to accumulate first before doing any truncation, it will allow these small numbers to accumulate to a significant value and not be truncated off during rounding process.
- In this example, a raw intermediate value 1.235*10^3 is obtained.
- After rounding, we get 1.24*10^3, i.e. the significance of the '1's are not truncated as in the case of the previous slide.
- In general, when performing arithmetic operation between numbers with huge difference in magnitude, always add/subtract between numbers that are of similar magnitude before combining them to other larger numbers.

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.

- To summarise what we have discuss in this section of overflow and accuracy.
- Some general rule of thumb that we can use when designing algorithm
  - Accumulate numbers with small magnitude first before merging them to calculation involving large numbers so that they would not be truncated off prematurely.
  - Take note of the range of the various data type, number format, register, memory etc
  - And apply threshold to check for overflow condition. Remember that if an overflow occur during computation, that automatically implies that the result obtain is wrong.
  - Try to perform division operation last to preserve as much precision as possible for the intermediate data.
- This is the last slide for the computer arithmetic chapter.