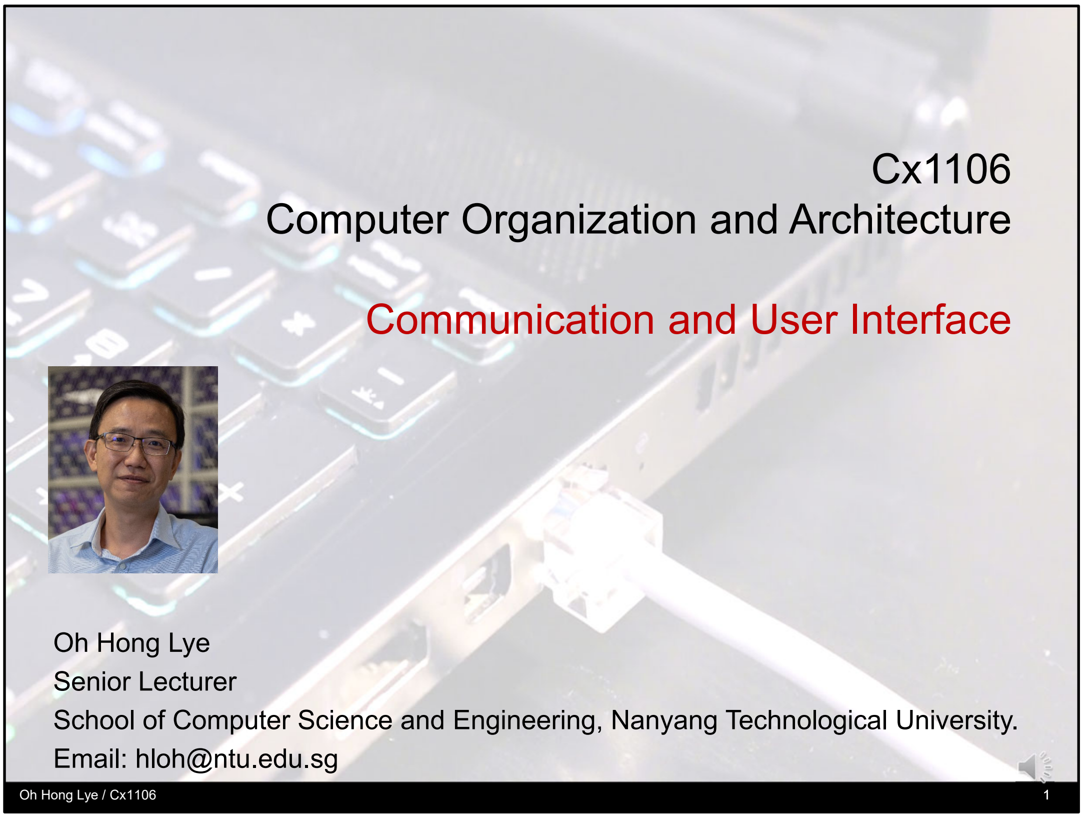


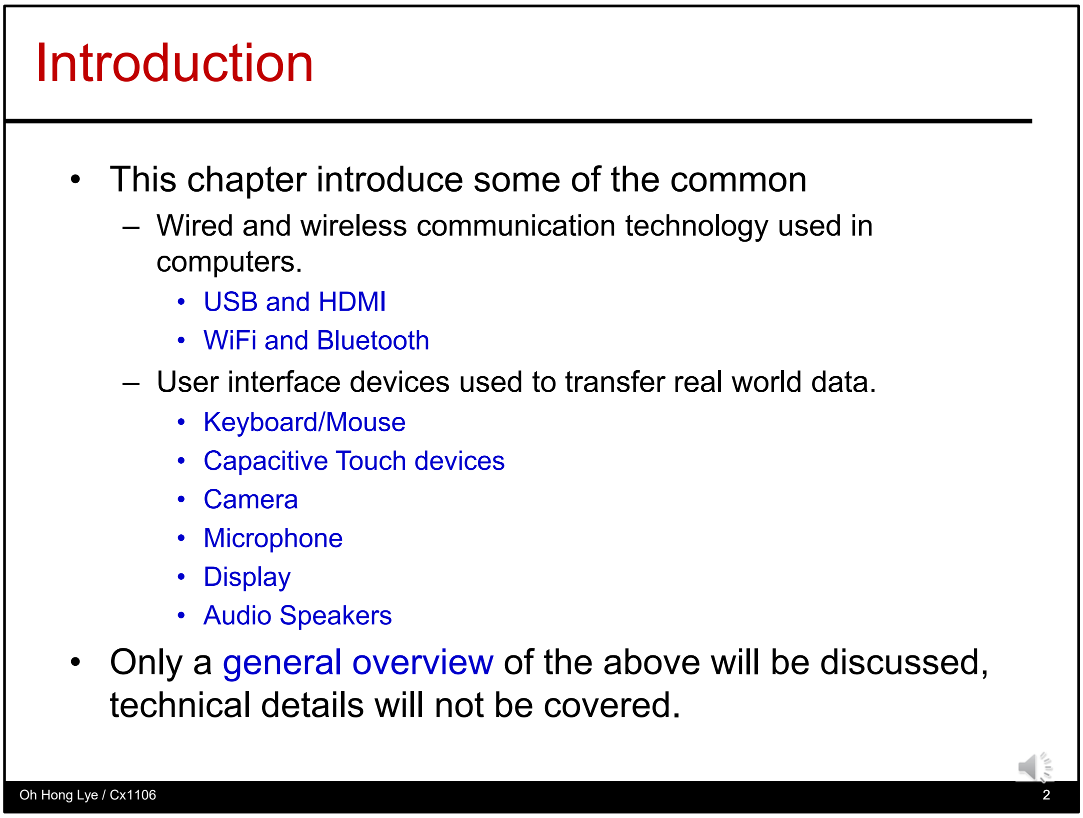
Cx1106
Computer Organization and Architecture

Communication and User Interface

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

Oh Hong Lye / Cx1106 1

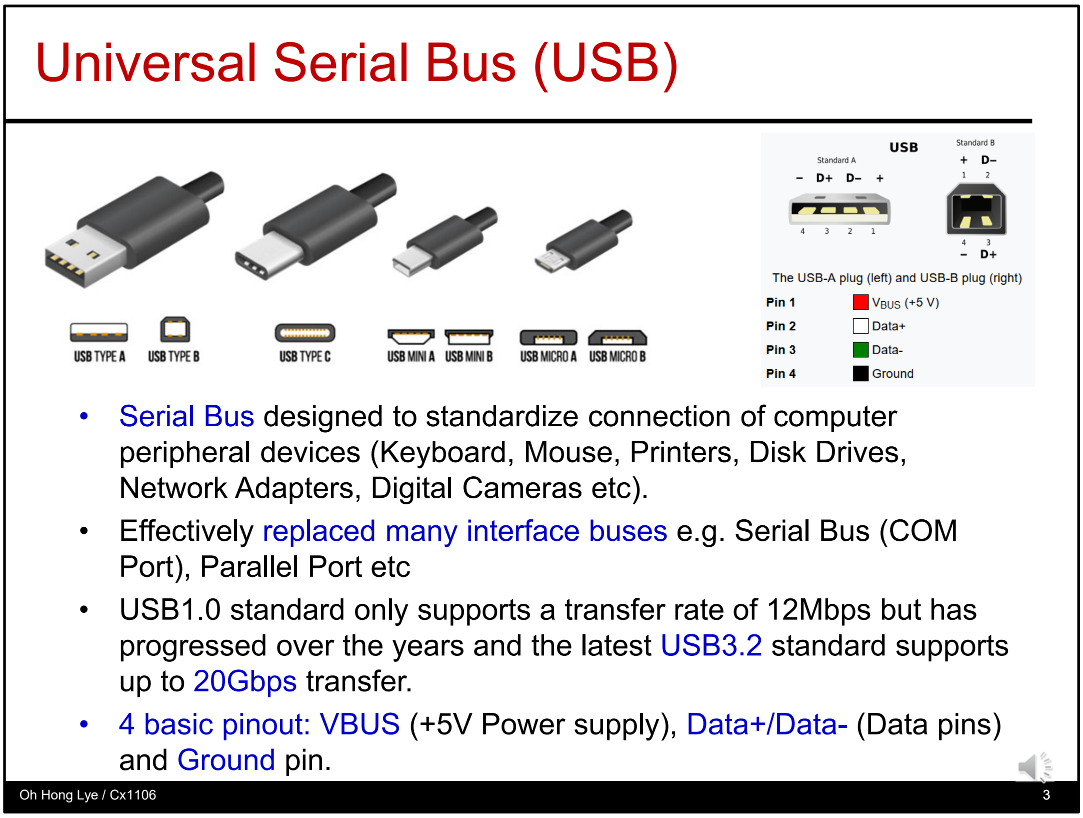
- This chapter deals with the more complex communication and user interfaces that a typical computer has.
- I have grouped them here to explicitly indicate that we will only cover the overview and basics where these interfaces are concerned.



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)

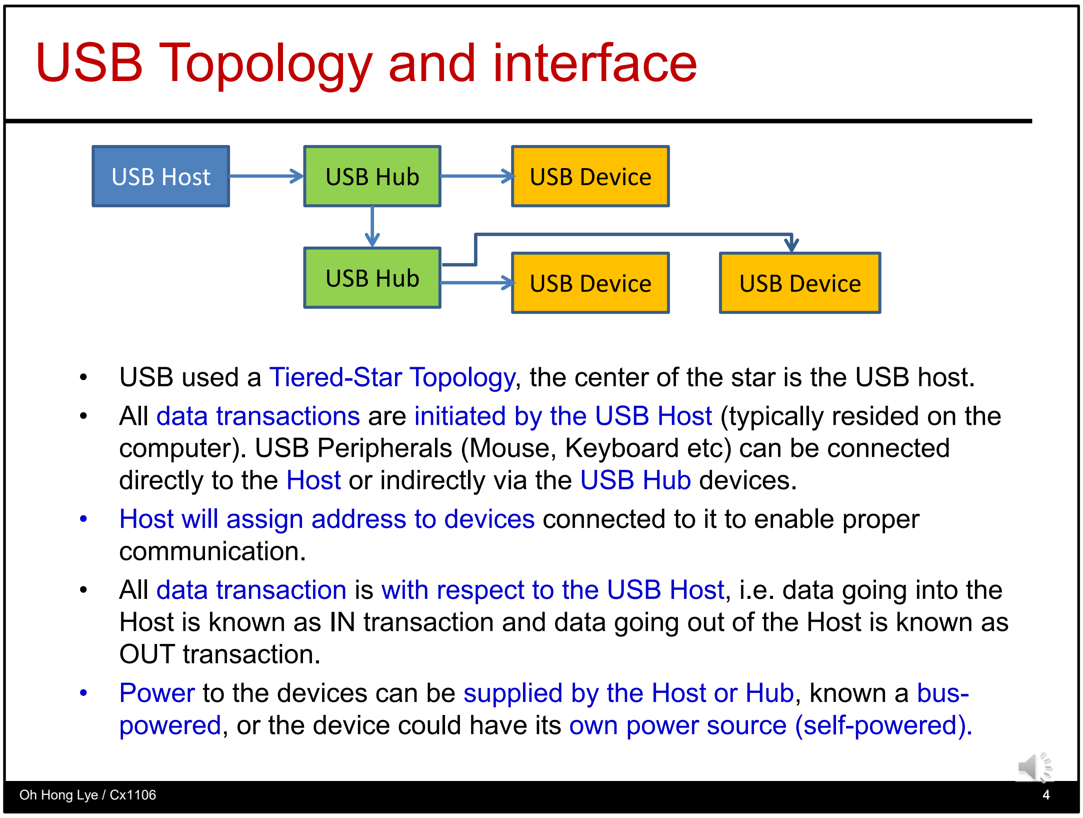
USB TYPE A
USB TYPE B
USB TYPE C
USB MINI A
USB MINI B
USB MICRO A
USB MICRO B
USB
Standard A
– D+ D– +
4 3 2 1
Standard B
+ D–
1 2
4 3
– D+

The USB-A plug (left) and USB-B plug (right)

Pin 1	🟥 V_{BUS} (+5 V)
Pin 2	□ Data+
Pin 3	🟩 Data-
Pin 4	■ Ground

- 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.

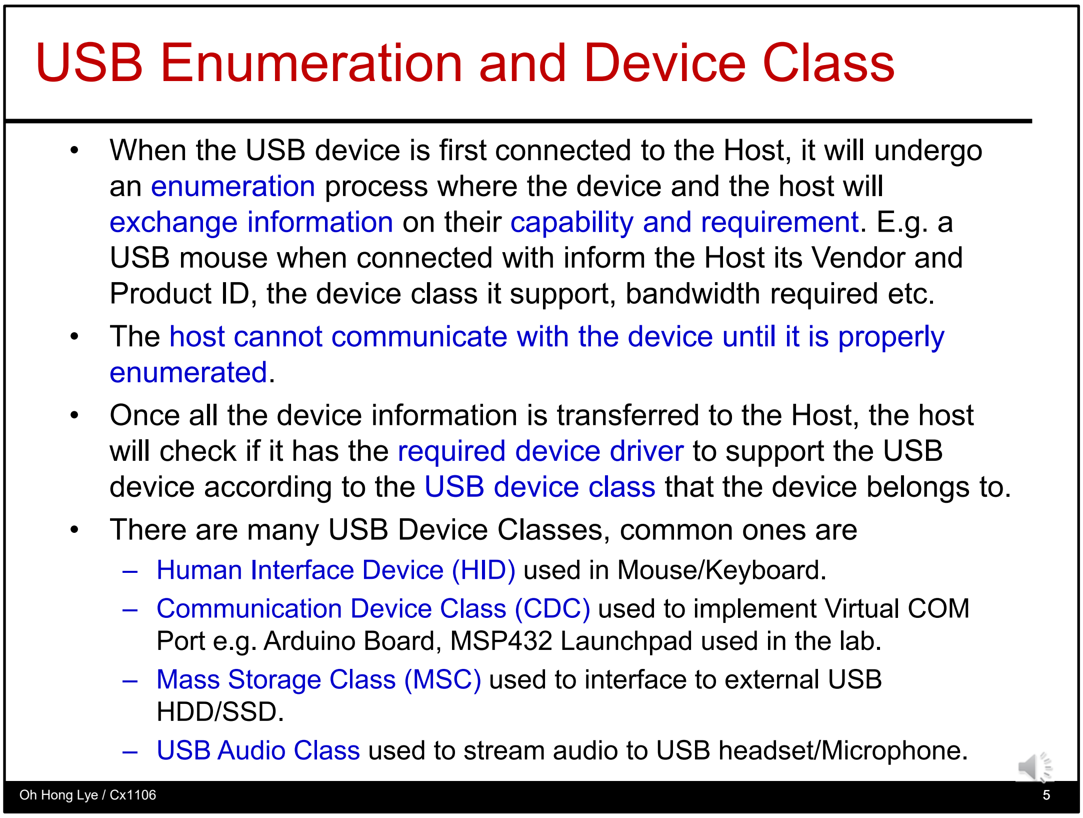
- First let’s look at the most common external bus interface in the computer world today, USB.
- USB was first proposed to replace low speed peripherals such as the PC Serial COM Port, Keyboard/Mouse PS2 etc.
    - But it quickly become the de facto standard to interface to many other devices, e.g. Printers, 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.

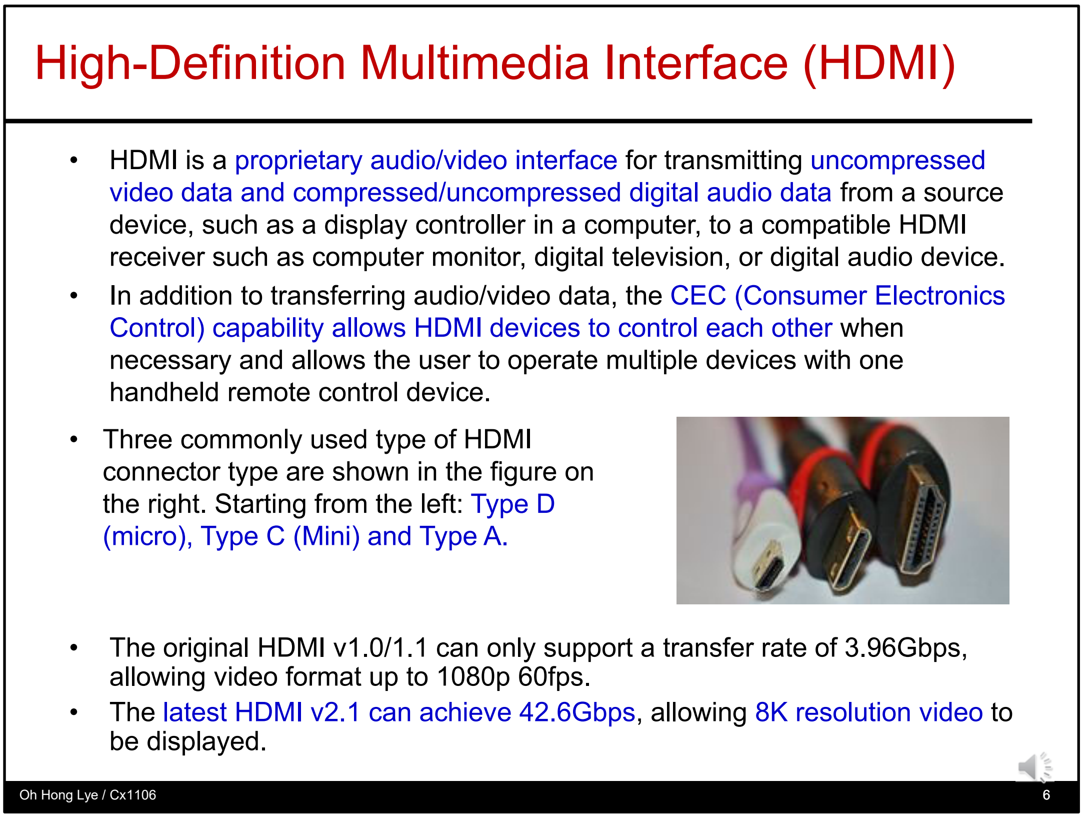


USB Enumeration and Device Class

- 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,

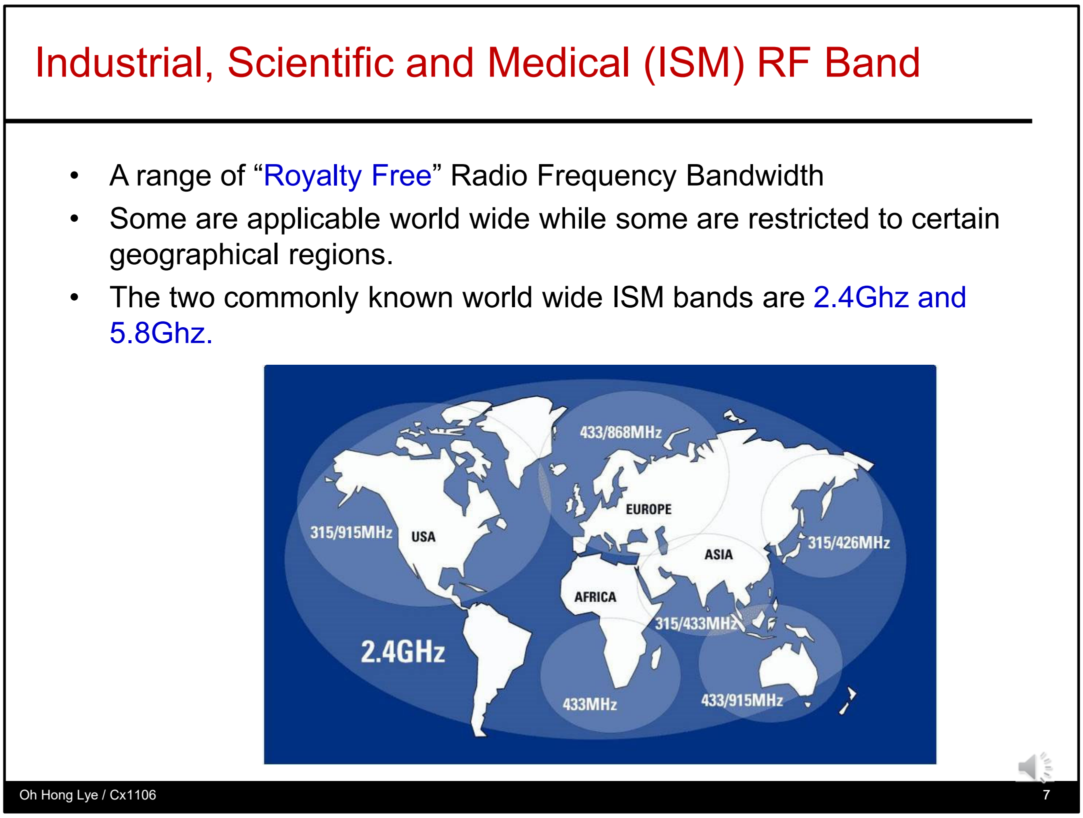


High-Definition Multimedia Interface (HDMI)

- 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.

Oh Hong Lye / Cx1106 6

- 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.

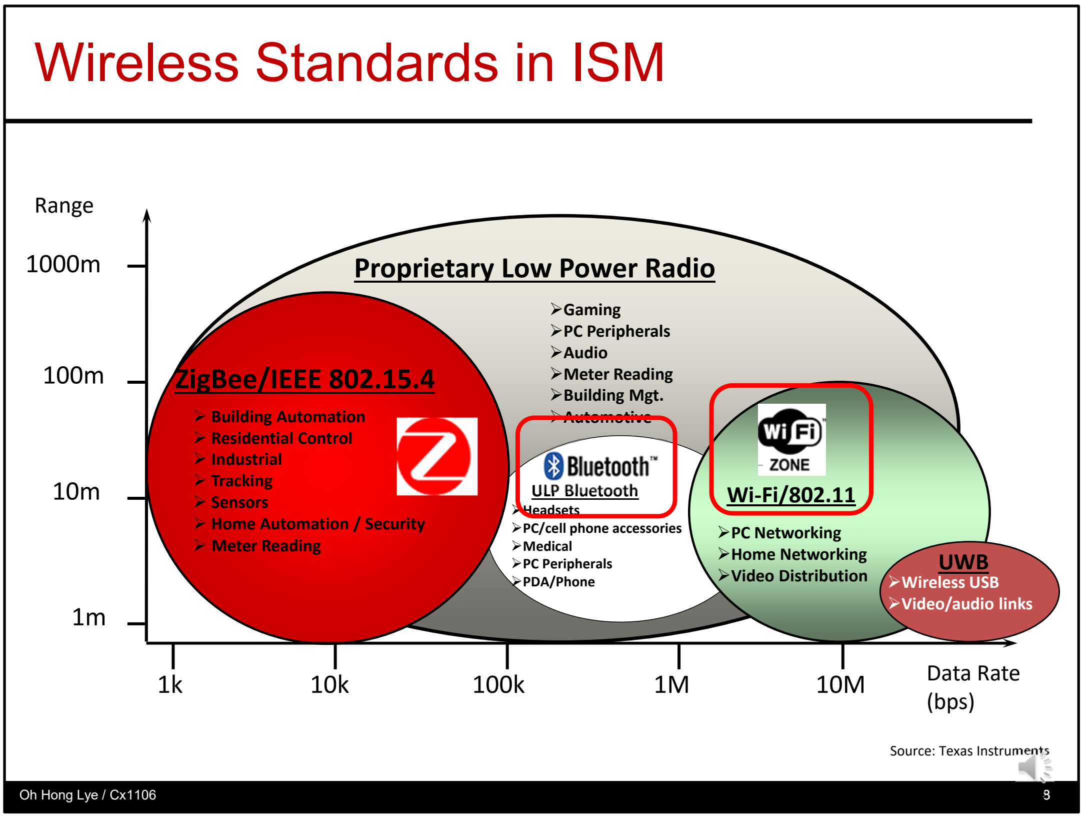


Industrial, Scientific and Medical (ISM) RF Band

- 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.



Wireless Standards in ISM

Proprietary Low Power Radio
- Gaming
- PC Peripherals
- Audio
- Meter Reading
- Building Mgt.
- Automotive

ZigBee/IEEE 802.15.4
- Building Automation
- Residential Control
- Industrial
- Tracking
- Sensors
- Home Automation / Security
- Meter Reading

Bluetooth™

ULP Bluetooth
- Headsets
- PC/cell phone accessories
- Medical
- PC Peripherals
- PDA/Phone

Wi Fi ZONE

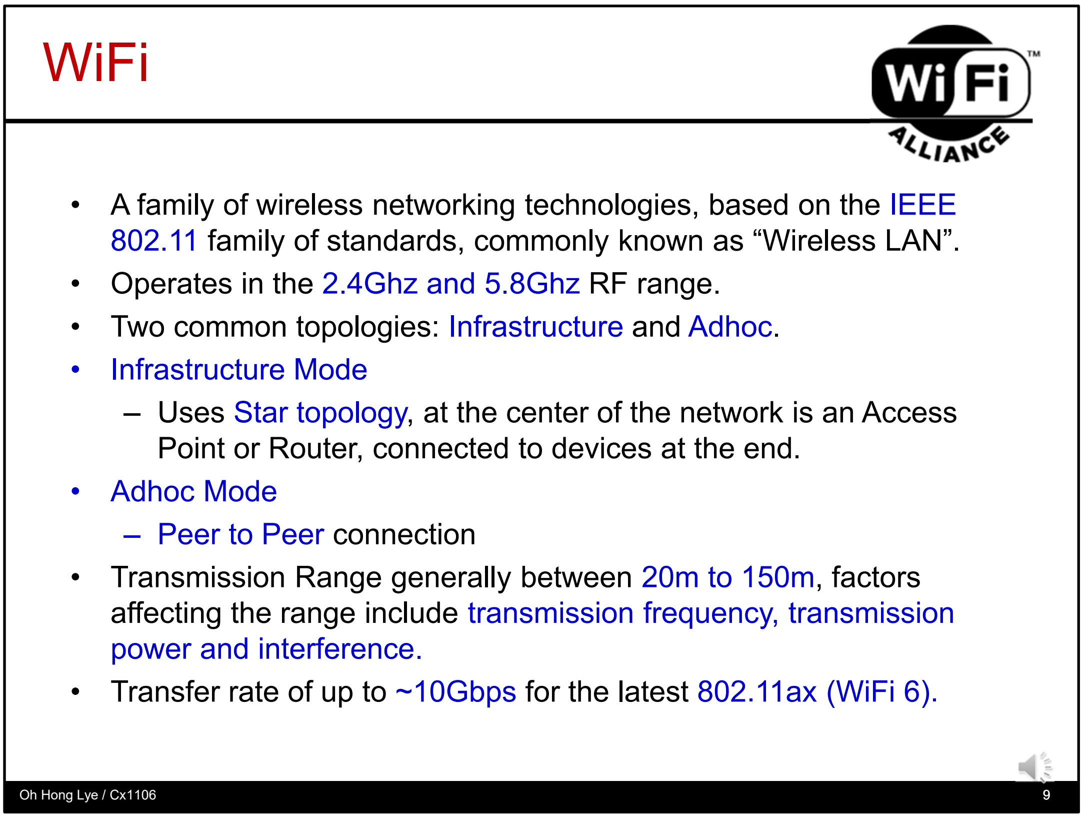
Wi-Fi/802.11
- PC Networking
- Home Networking
- Video Distribution

UWB
- Wireless USB
- Video/audio links

Source: Texas Instruments

Oh Hong Lye / Cx1106 8

- 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.

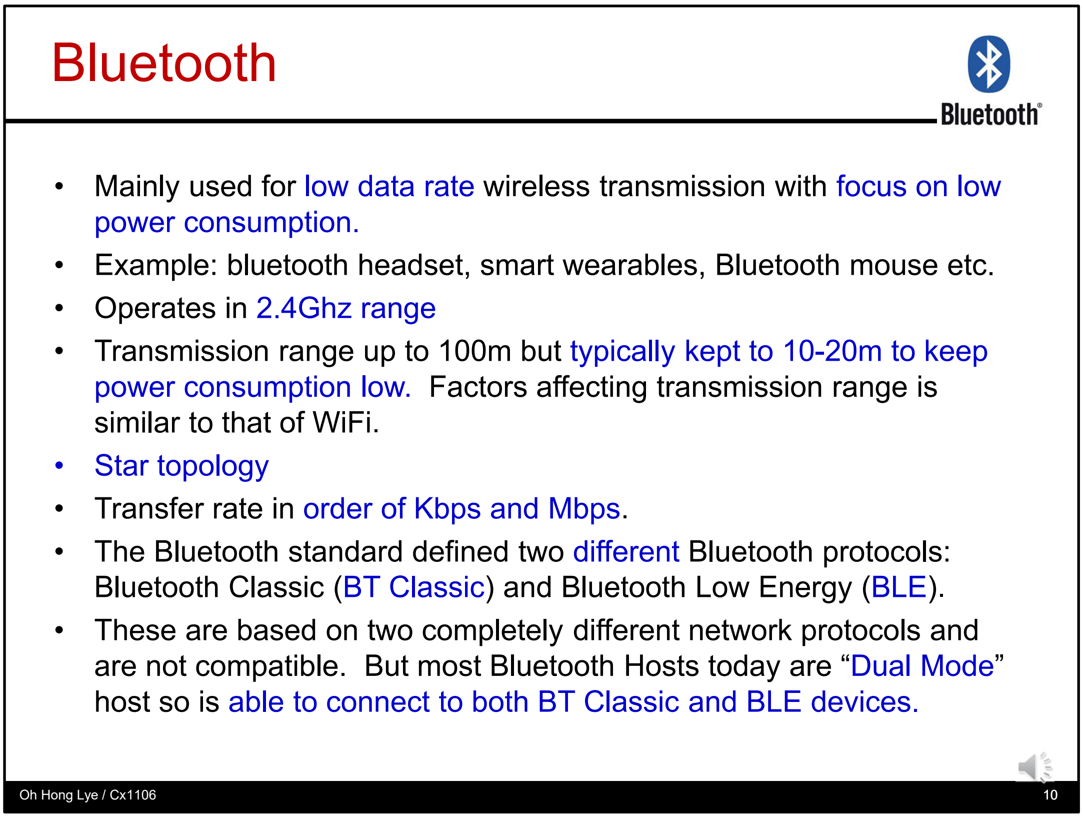


WiFi

- 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).

Oh Hong Lye / Cx1106 9

- 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.

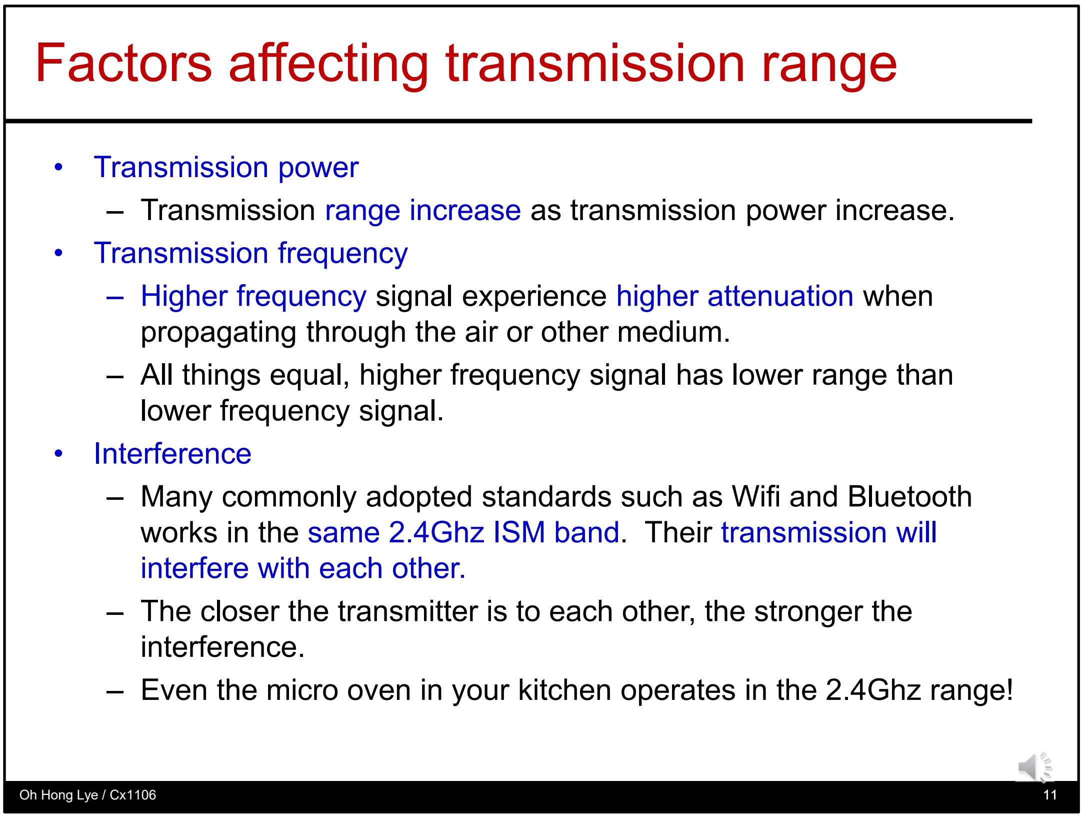


Bluetooth

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

Oh Hong Lye / Cx1106 10

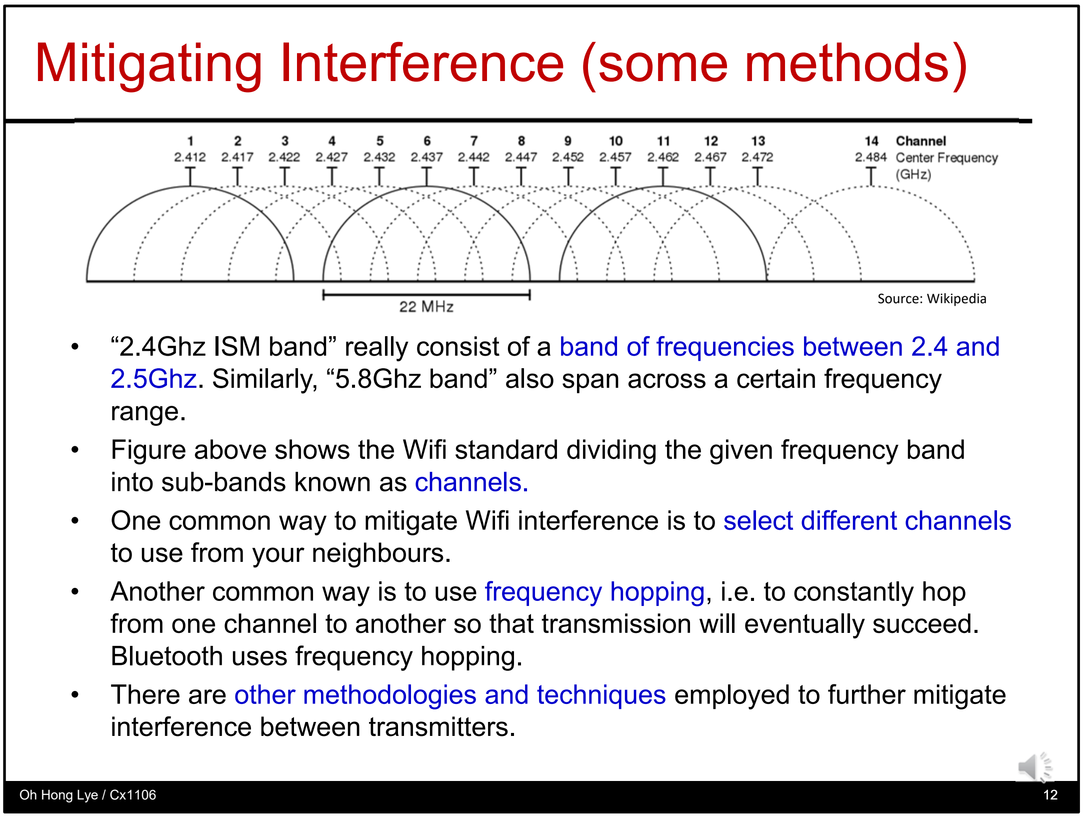
- Bluetooth transmission standard targets low power applications, compared to Wifi transmission standard.
    - Incidentally, the transmission rate is much lower than Wifi as well.
- Example of Bluetooth applications includes headset, keyboard, mouse, smart wearables etc.
- Bluetooth devices operates 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.



Factors affecting transmission range

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

- 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.



1 2.412
2 2.417
3 2.422
4 2.427
5 2.432
6 2.437
7 2.442
8 2.447
9 2.452
10 2.457
11 2.462
12 2.467
13 2.472
14 2.484
Channel
Center Frequency (GHz)
22 MHz

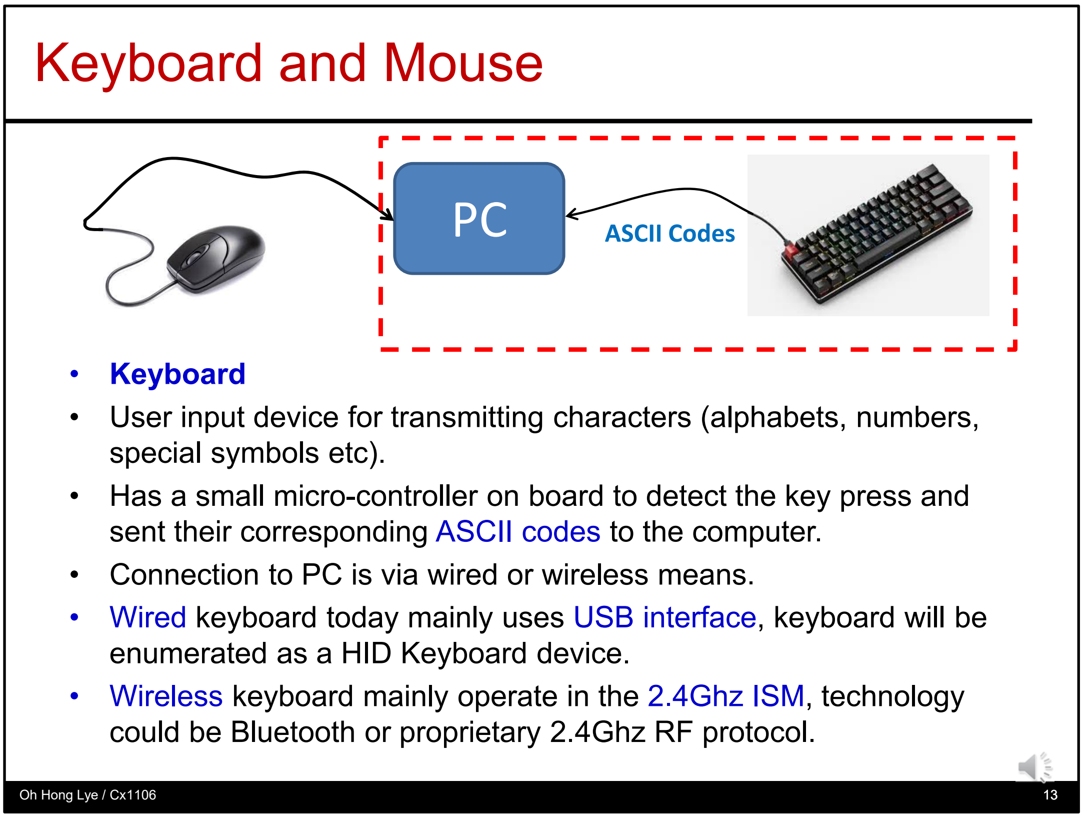
Mitigating Interference (some methods)

Source: Wikipedia

- "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.

Oh Hong Lye / Cx1106 12

- 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,.

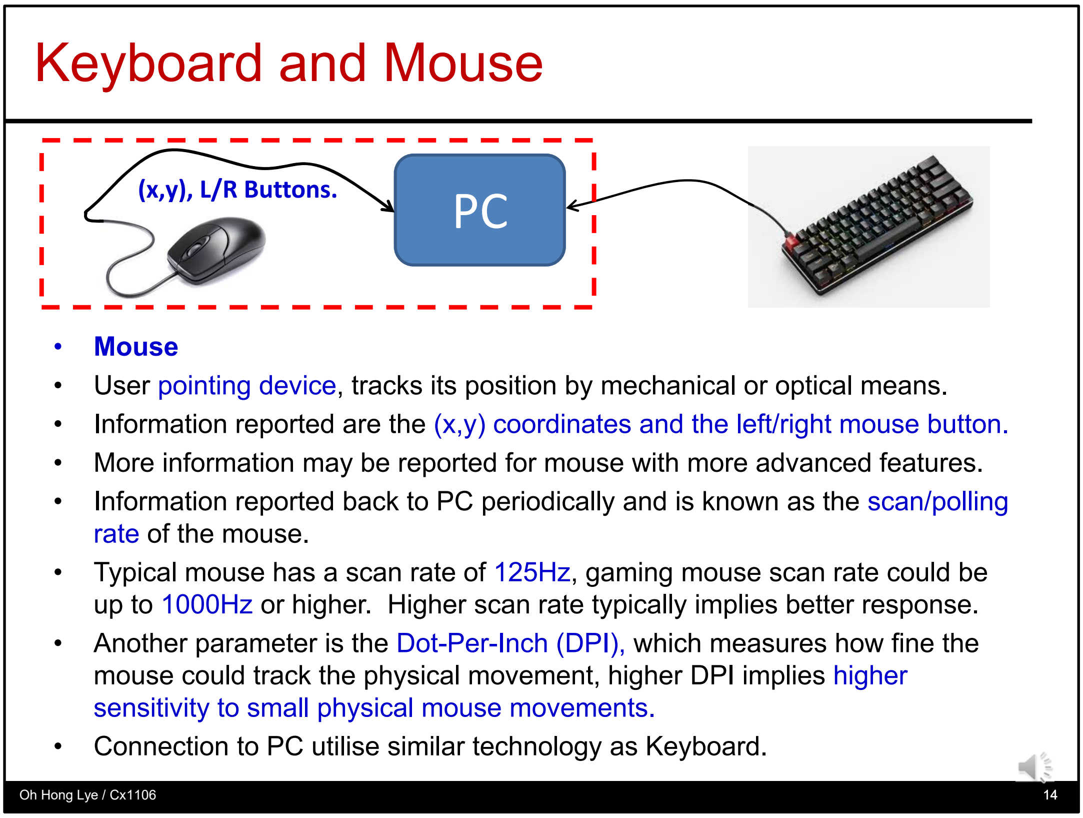

PC
ASCII Codes

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.

Oh Hong Lye / Cx1106 13

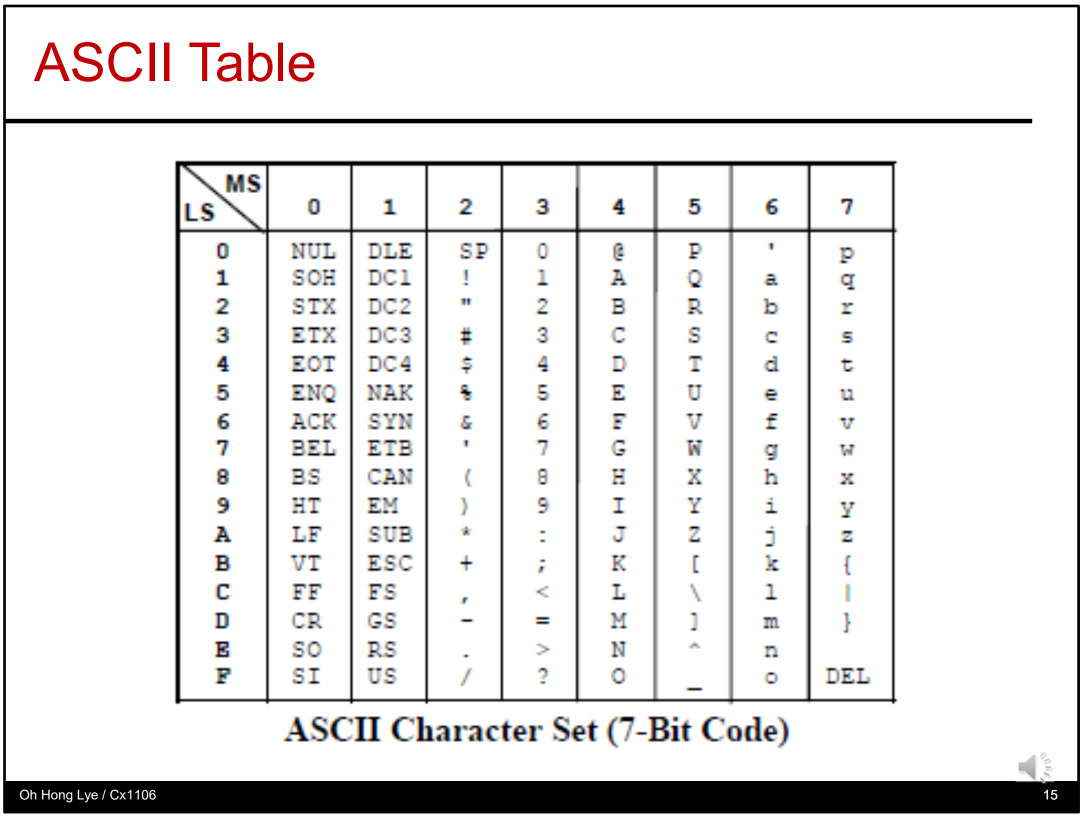
- Keyboard and Mouse are the two important user input devices in a computer system.
- Both device has a micro-controller in them that sent the main processor in the computer information based on the keys, buttons and mouse movement status.
- For Key Board, note that a coded form of the key status is 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.


(x,y), L/R Buttons.
PC

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.

- 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,

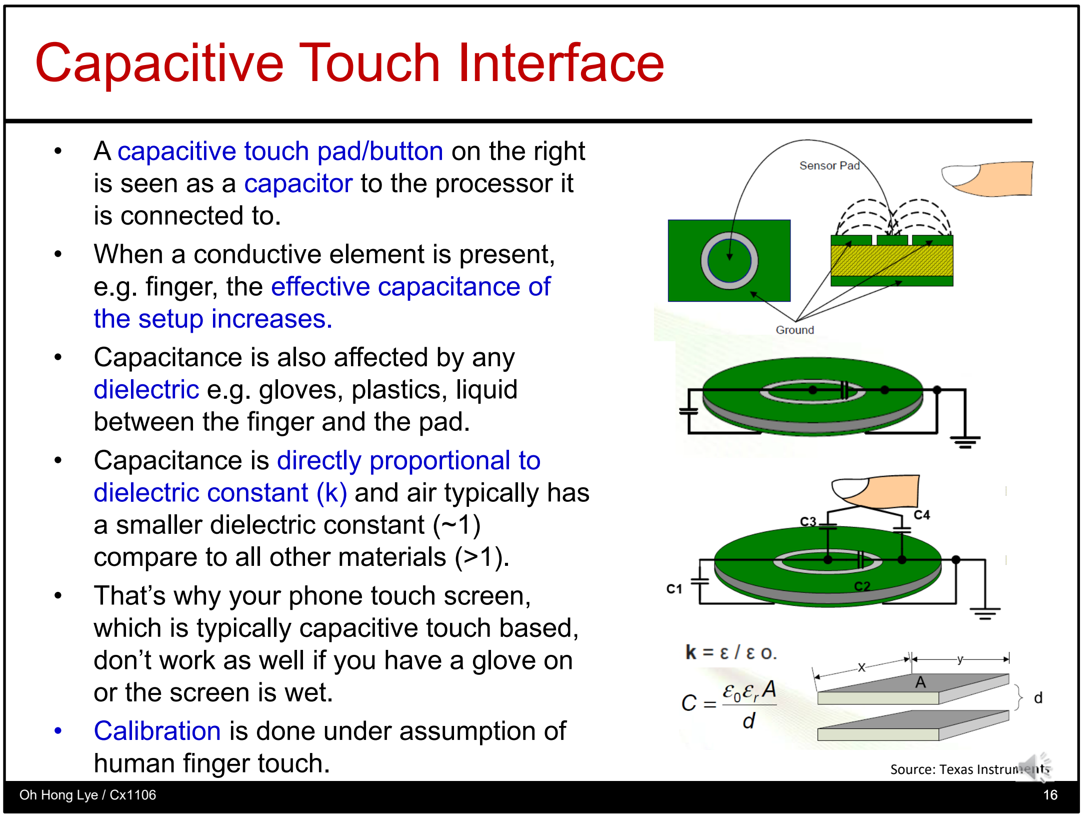


ASCII Table

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.

k = ε / ε o.

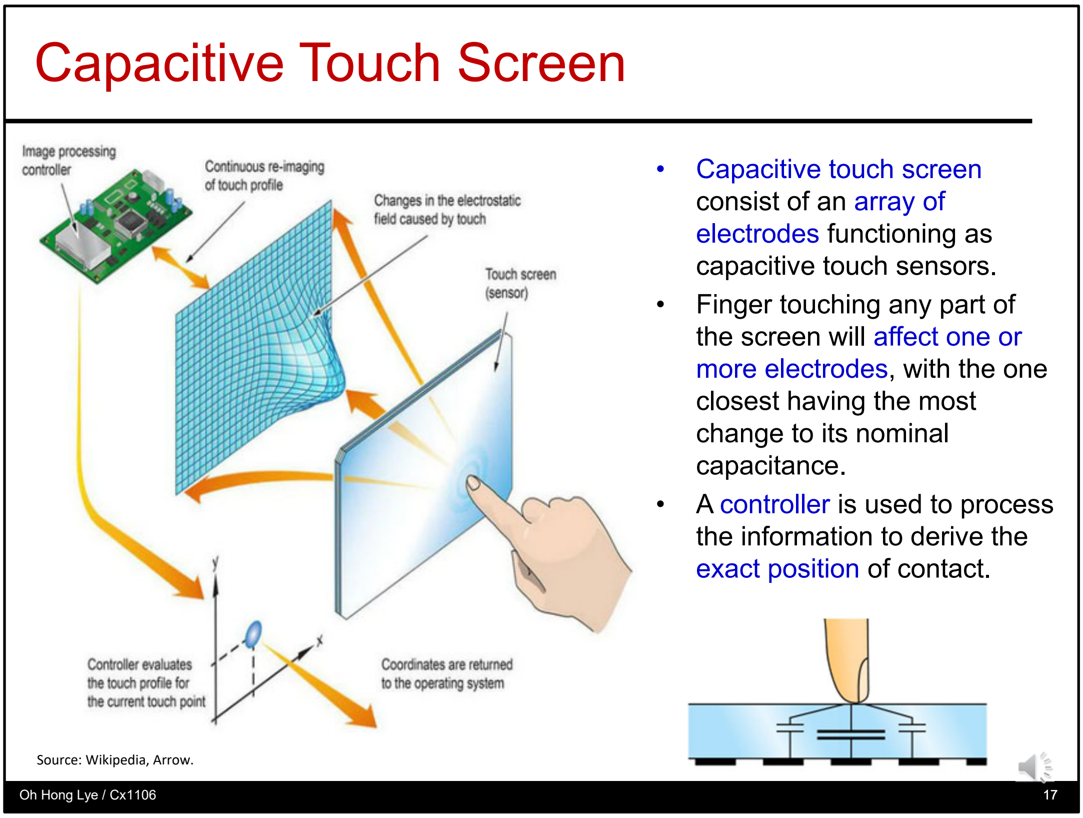
$$C = \frac{\varepsilon_0 \varepsilon_r A}{d}$$

Source: Texas Instruments

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.



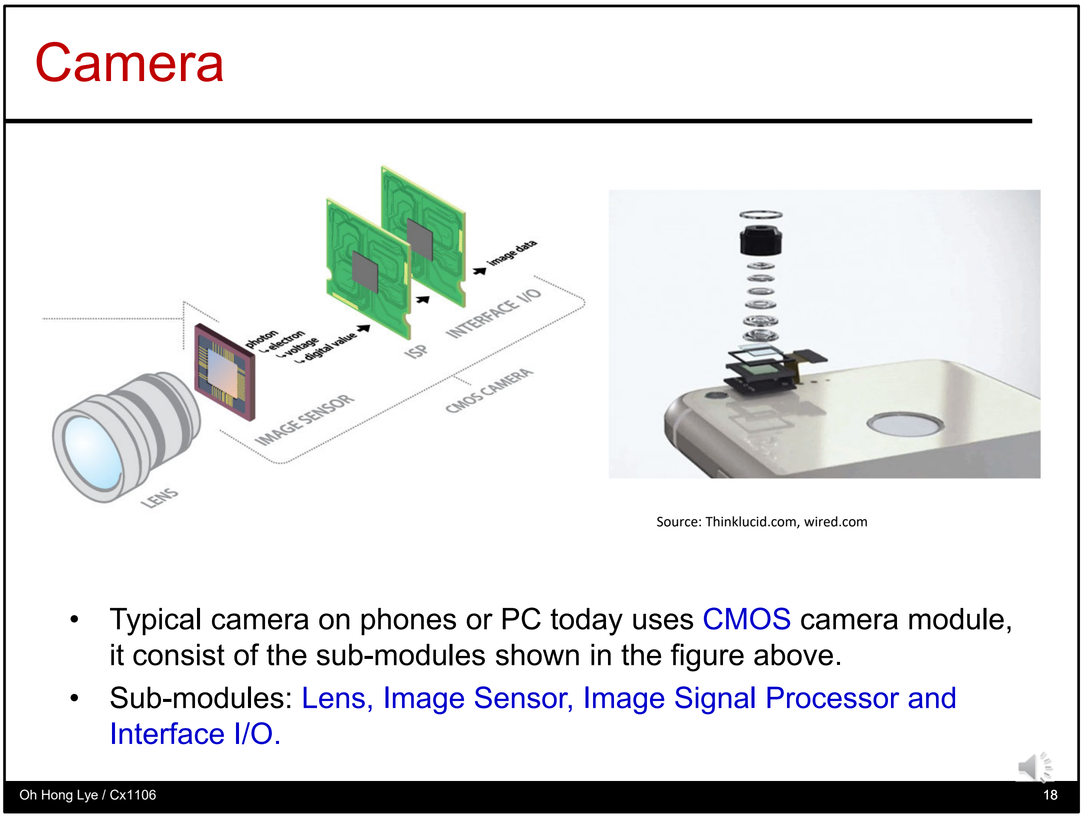
Capacitive Touch Screen

Image processing controller
Continuous re-imaging of touch profile
Changes in the electrostatic field caused by touch
Touch screen (sensor)
y
x
Controller evaluates the touch profile for the current touch point
Coordinates are returned to the operating system

- 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.

Source: Wikipedia, Arrow.

- 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.



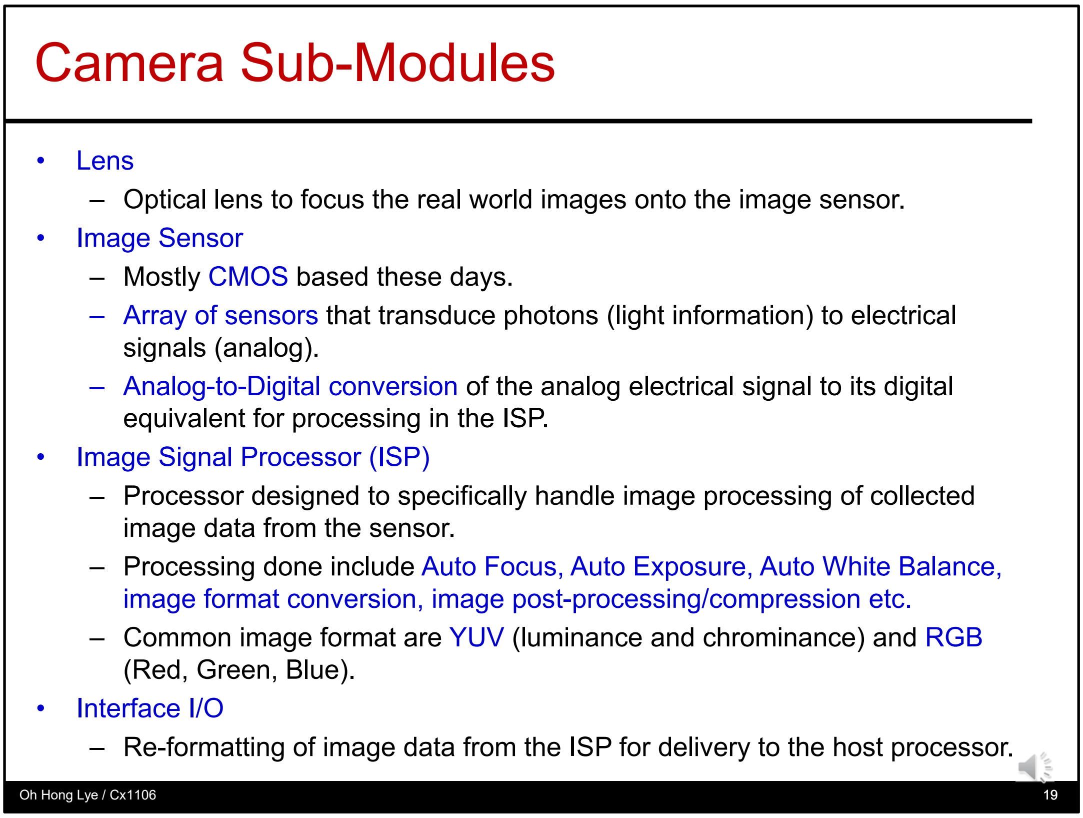
Camera

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.

Oh Hong Lye / Cx1106 18

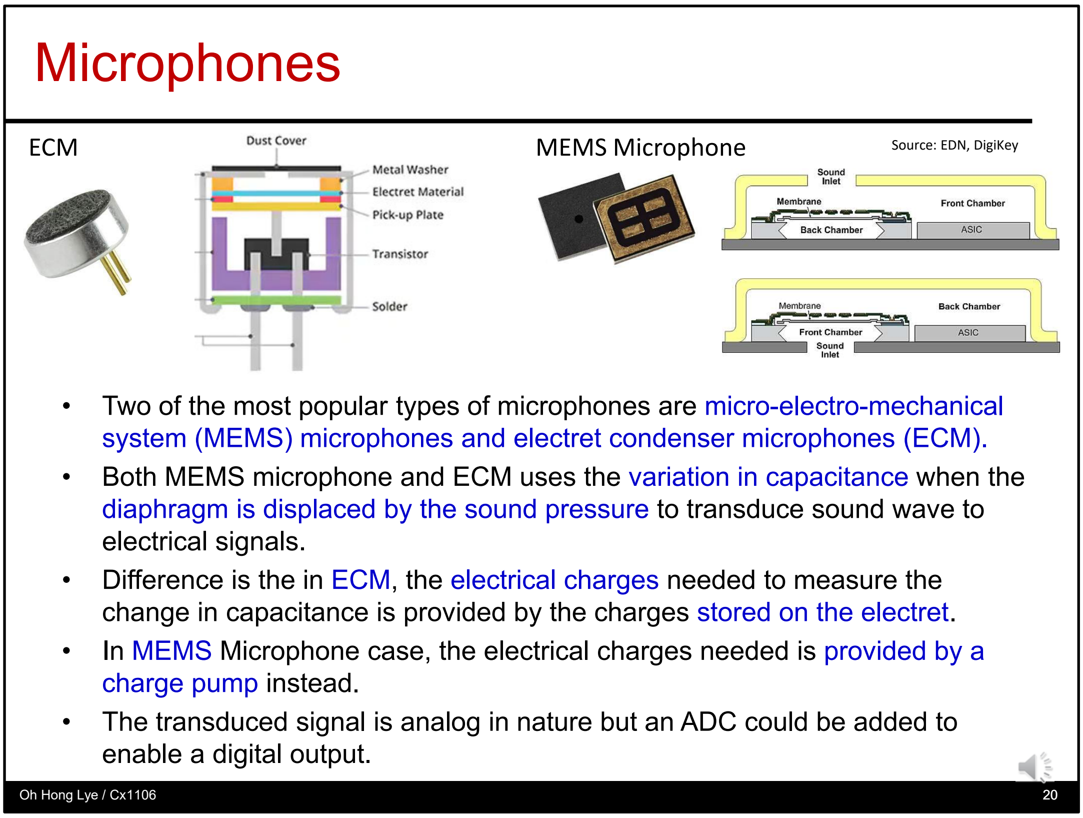
- Most phones, tablets and laptop these days are equipped with one or more cameras.
- There are two main types of camera sensors used in the market, CCD and CMOS.
  - Most products with thin mechanical ID profile uses the CMOS camera.
  - The camera module consist of the following main components
  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

Dust Cover
Metal Washer
Electret Material
Pick-up Plate
Transistor
Solder

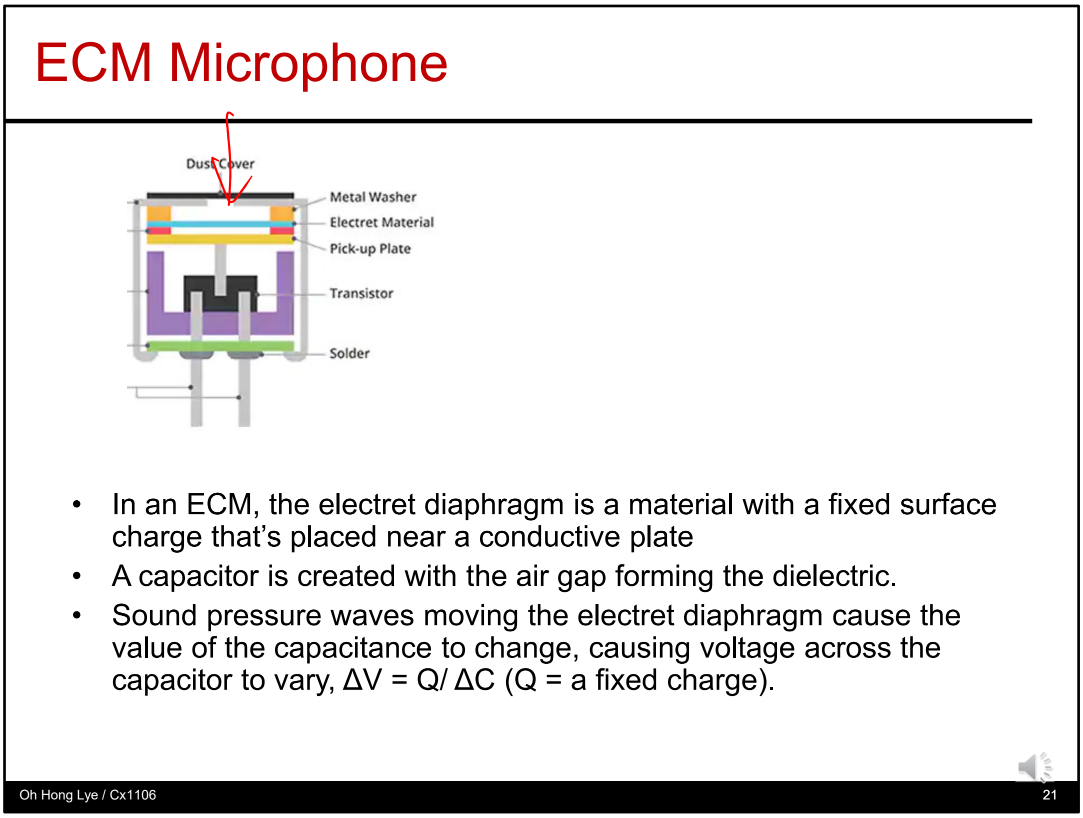
MEMS Microphone

Source: EDN, DigiKey

Sound Inlet
Membrane
Front Chamber
Back Chamber
ASIC
Membrane
Back Chamber
Front Chamber
ASIC
Sound Inlet

- 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.

- 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.



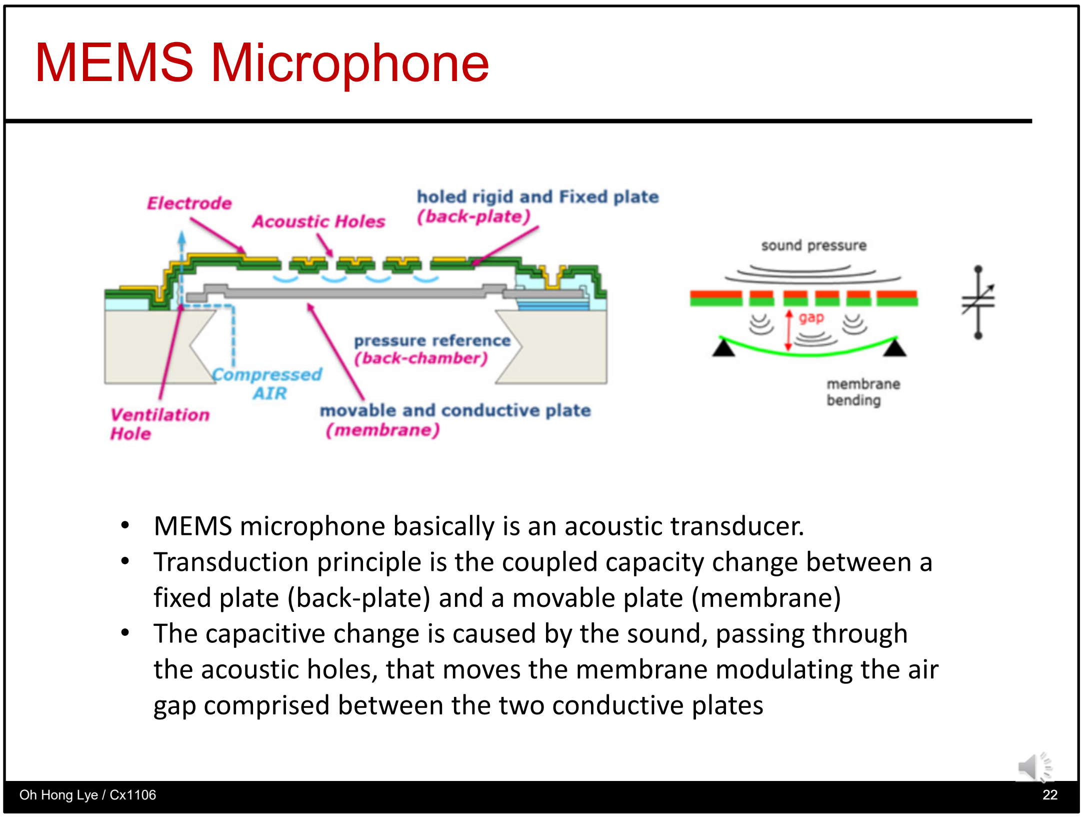
ECM Microphone

Dust Cover
Metal Washer
Electret Material
Pick-up Plate
Transistor
Solder

- 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).

Oh Hong Lye / Cx1106 21

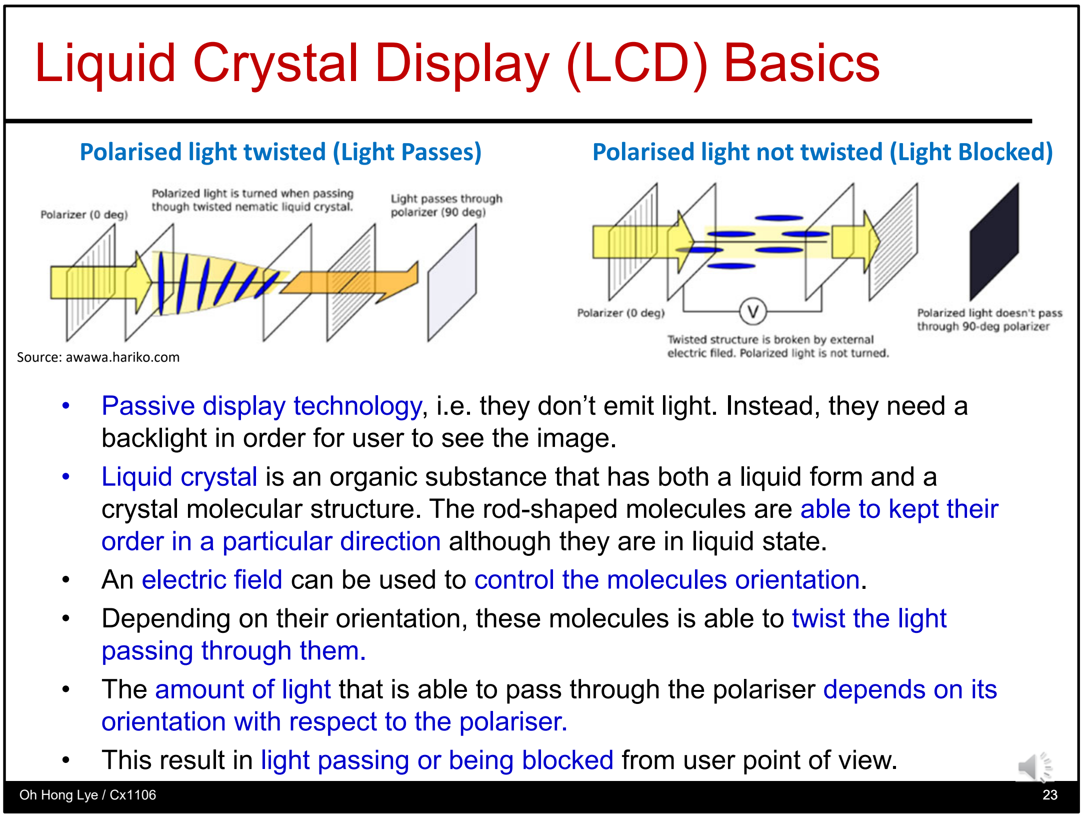
- In the ECM microphone, the two capacitor plates are formed by
  - an electret, which stores some excess charges permanently and form the movable plate of the capacitor.
  - A fixed conductive plate connected to the rest of the circuitry of the microphone.
- The pressure created by the sound wave causes 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

- 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)

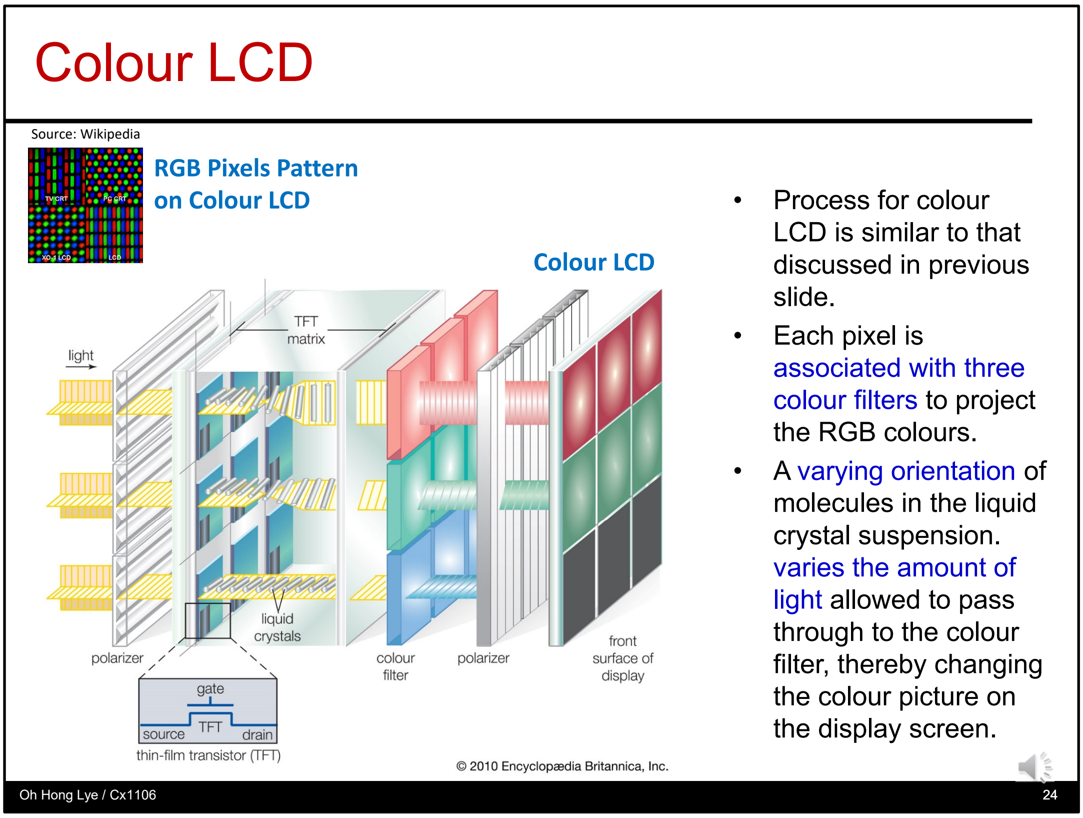
Polarised light not twisted (Light Blocked)

Polarizer (0 deg)
Polarized light is turned when passing though twisted nematic liquid crystal.
Light passes through polarizer (90 deg)
Polarizer (0 deg)
V
Twisted structure is broken by external electric filed. Polarized light is not turned.
Polarized light doesn't pass through 90-deg polarizer

Source: awawa.hariko.com

- 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.

- 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

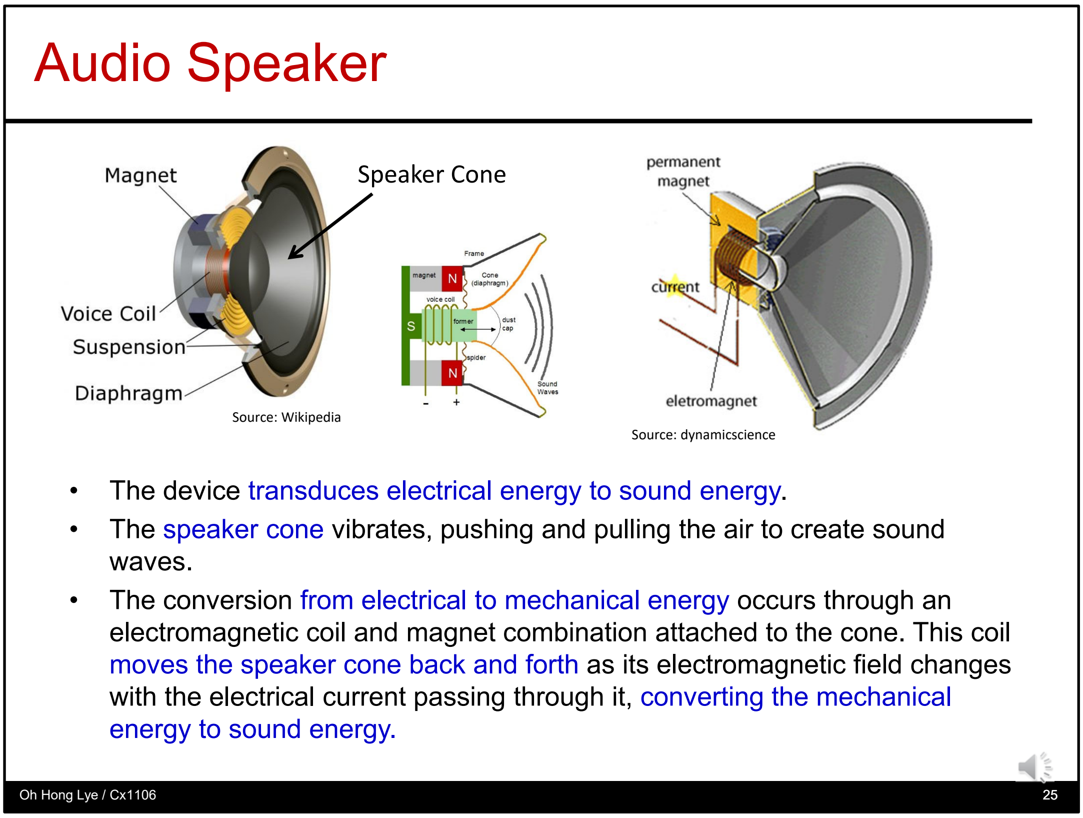
thin-film transistor (TFT)

© 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.

Oh Hong Lye / Cx1106 24

- 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.



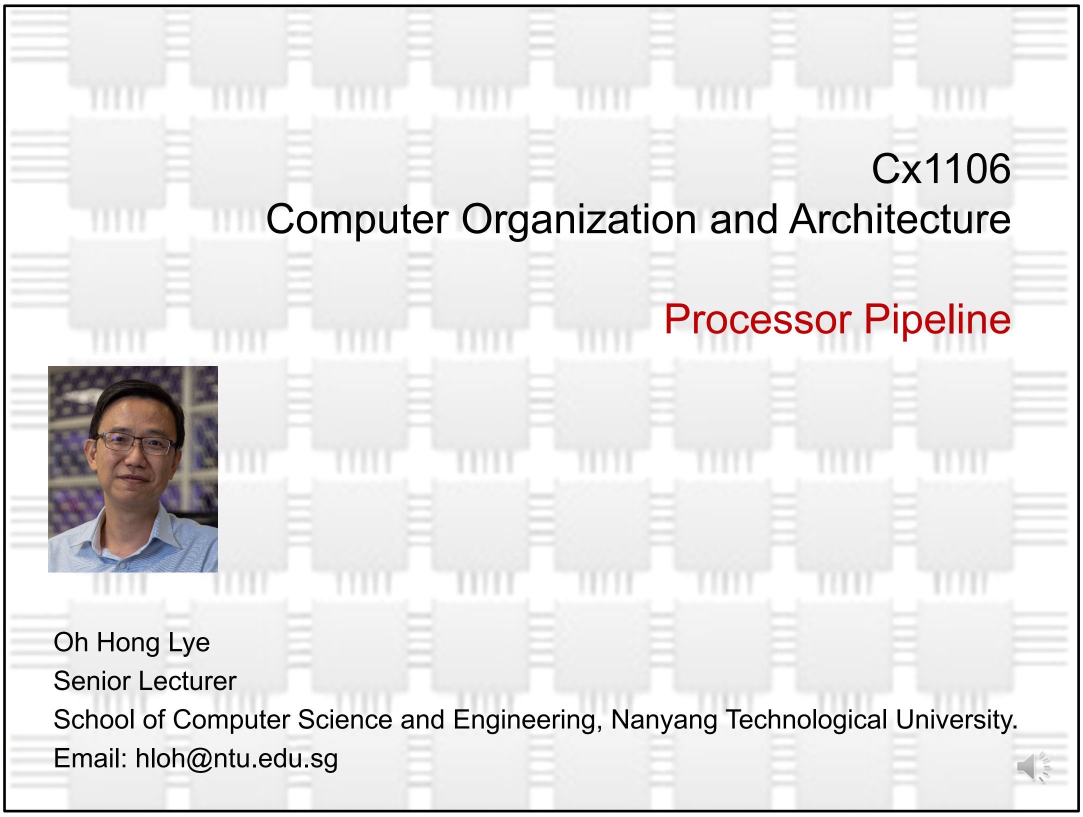
Audio Speaker

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.

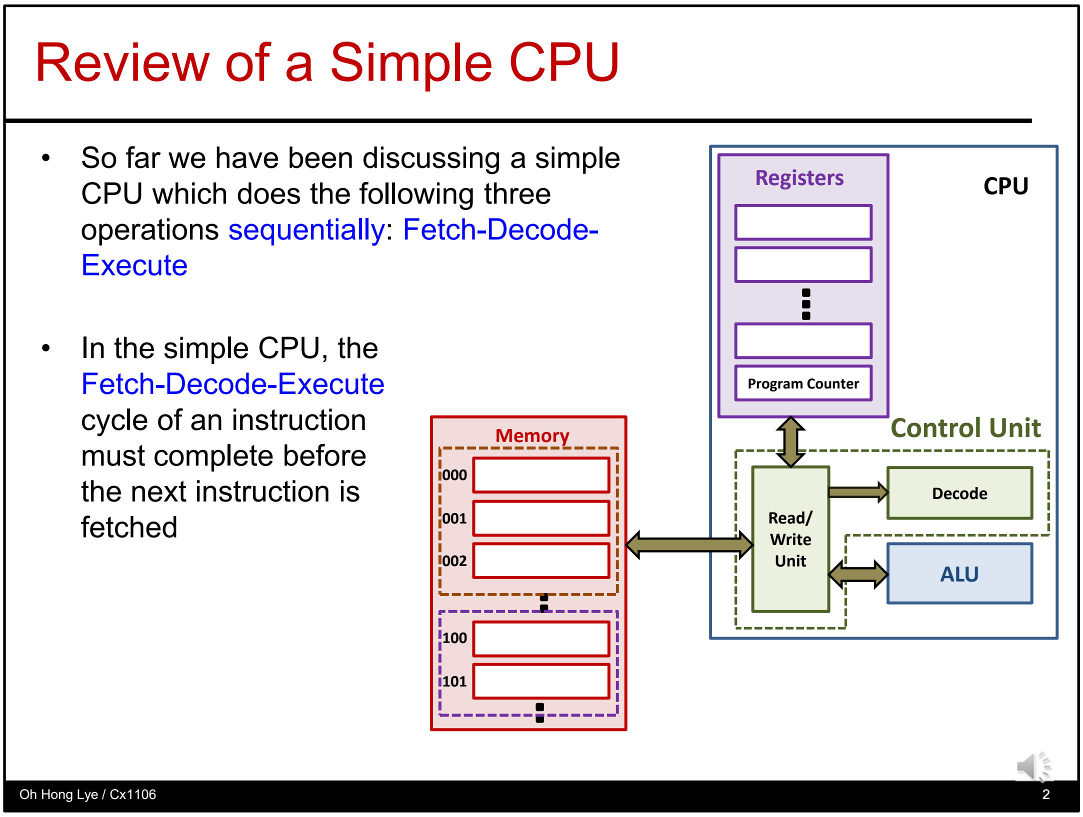


Cx1106
Computer Organization and Architecture

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.



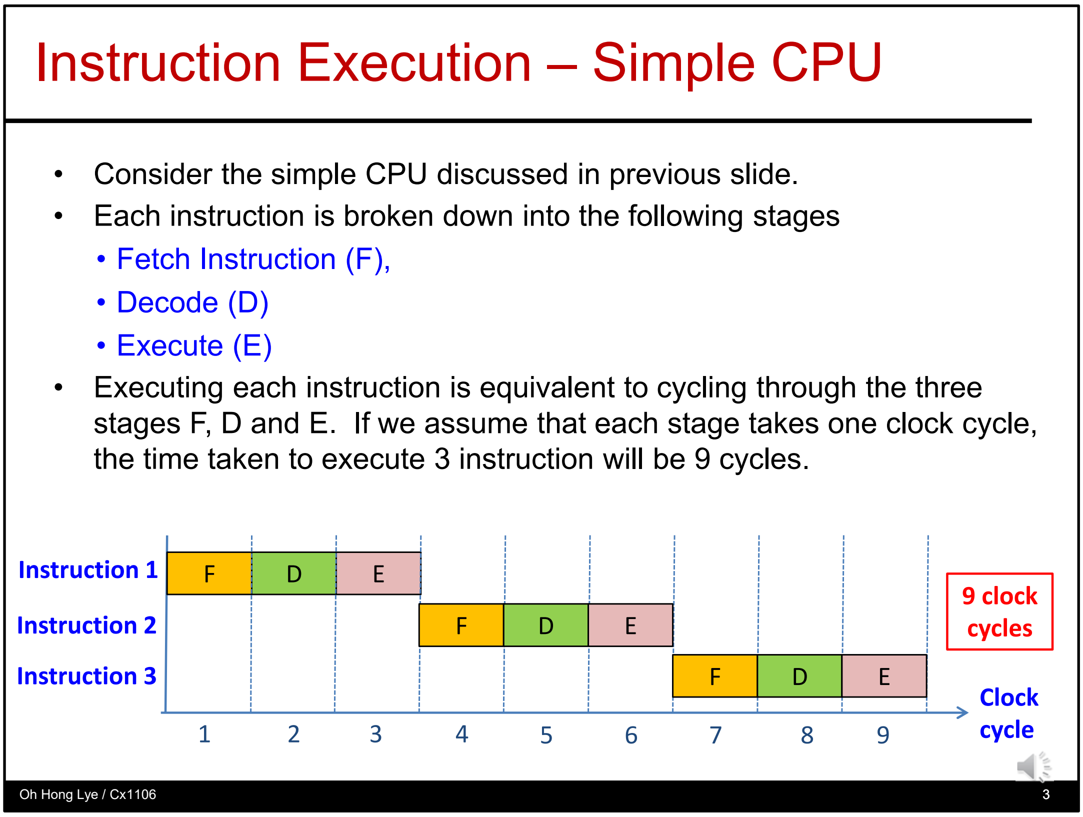
Review of a Simple CPU

- 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.

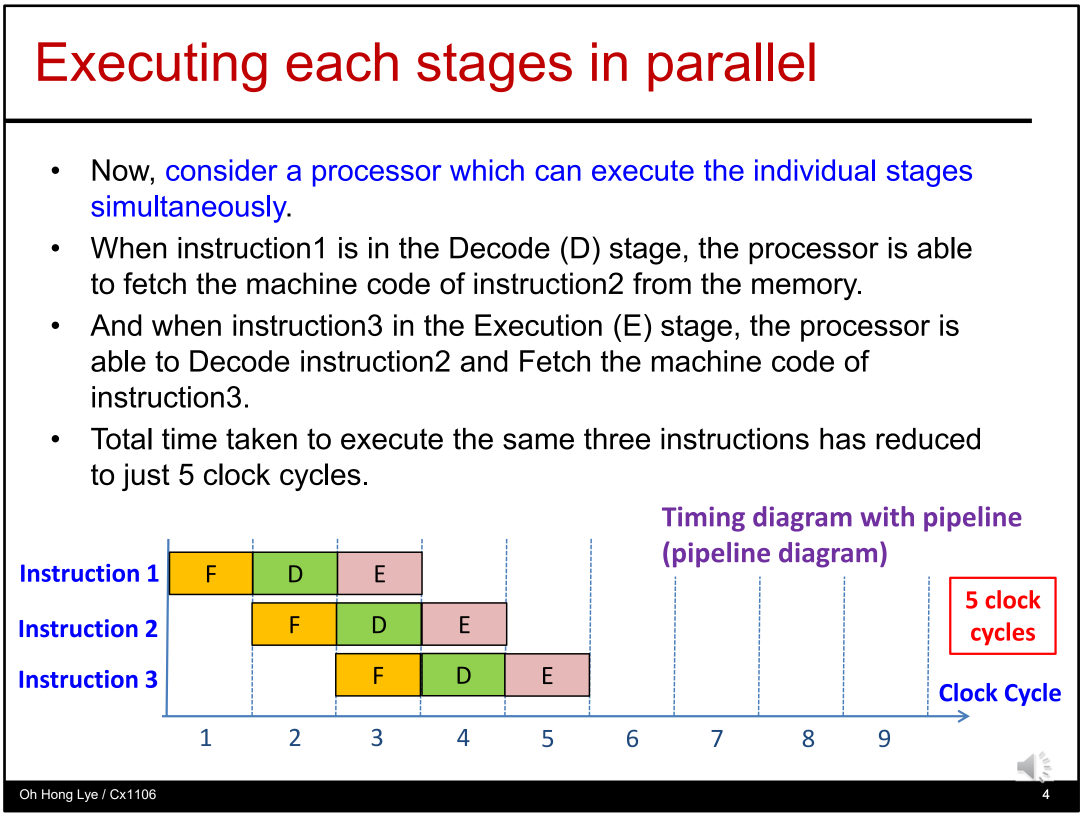


Instruction Execution – Simple CPU

- 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.

	1	2	3	4	5	6	7	8	9
Instruction 1	F	D	E						
Instruction 2				F	D	E			
Instruction 3							F	D	E

- 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.



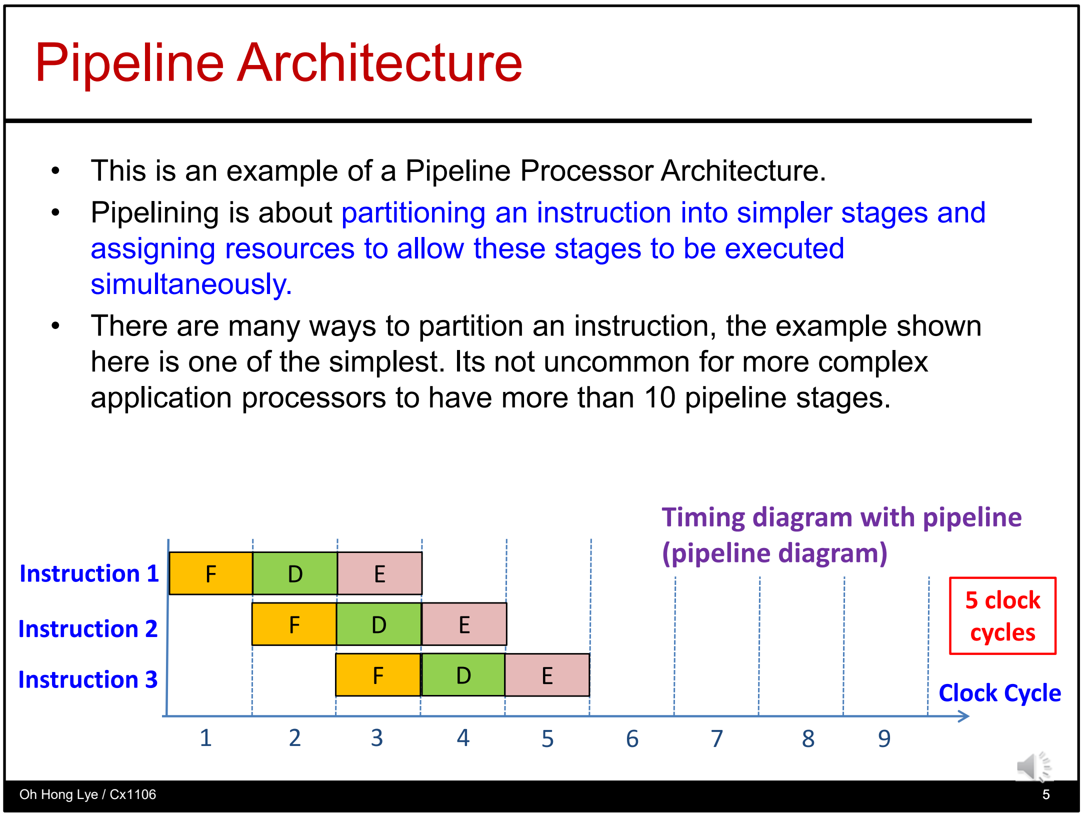
Executing each stages in parallel

- 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

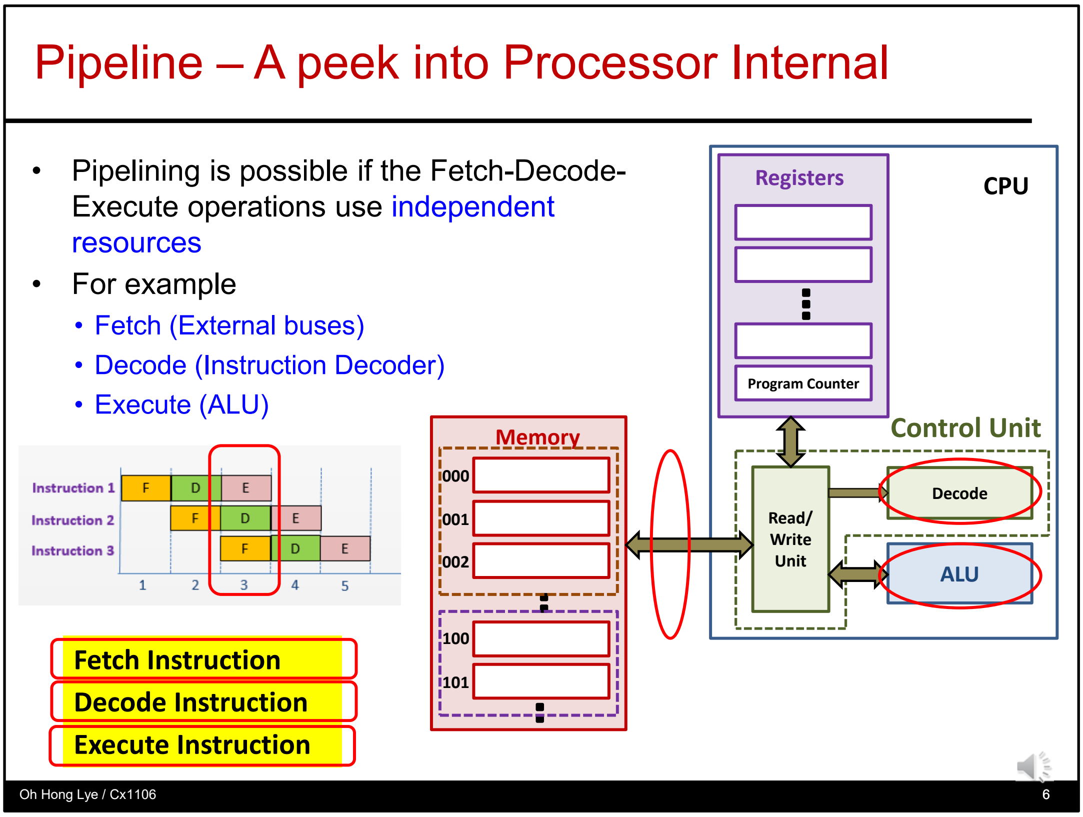
- Now, if a processor is able to execute the Fetch, Decode and Execute stage simultaneously, then the outcome will be very different.
- The following animation will illustrate the difference.
- The machine code for instruction 1 is fetched in the first clock cycle
- In the 2nd clock cycle, the processor 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.

- 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.



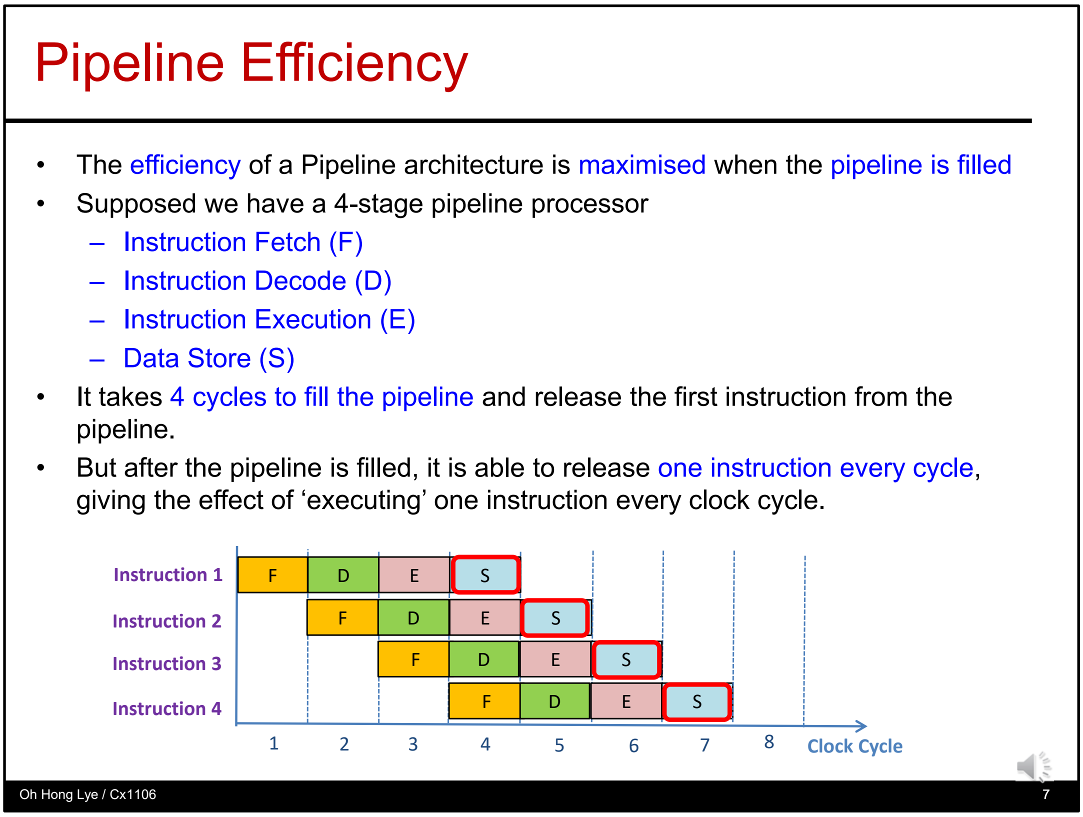
Pipeline – A peek into Processor Internal

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

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

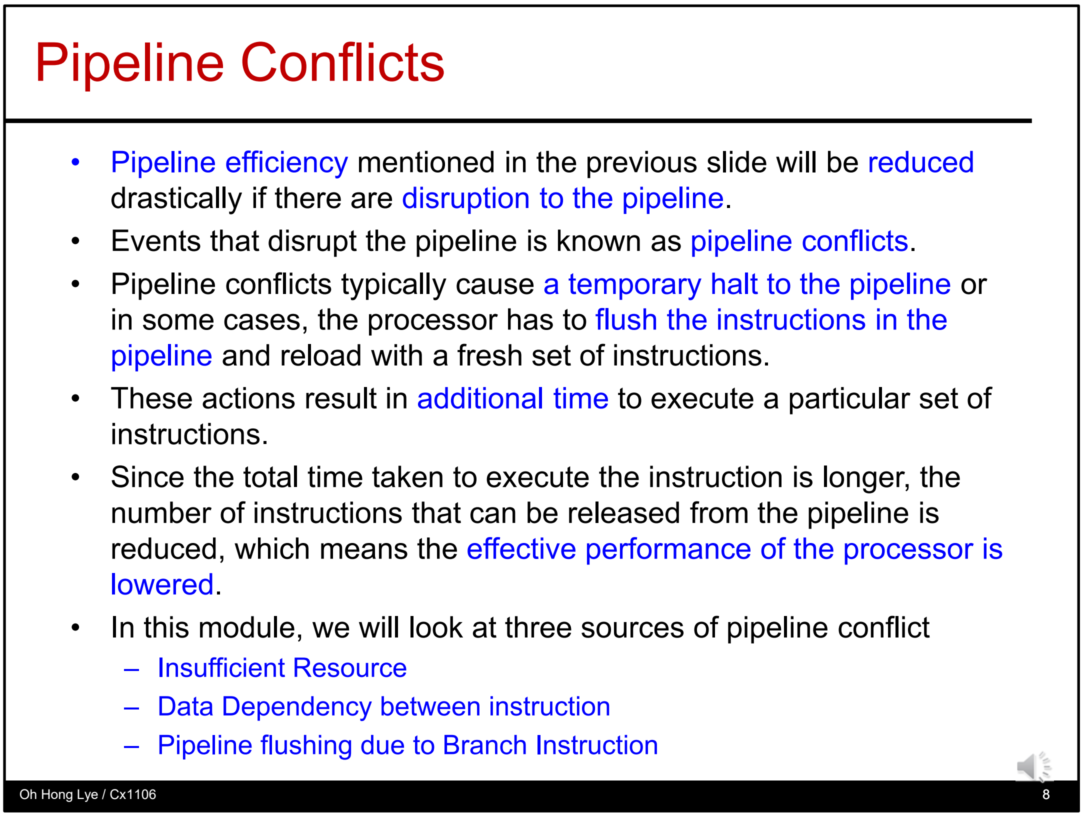
- The efficiency of a Pipeline architecture is maximised when the pipeline is filled
- Supposed we have a 4-stage pipeline processor
  - Instruction Fetch (F)
  - Instruction Decode (D)
  - Instruction Execution (E)
  - Data Store (S)
- It takes 4 cycles to fill the pipeline and release the first instruction 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.

	1	2	3	4	5	6	7	8
Instruction 1	F	D	E	S				
Instruction 2		F	D	E	S			
Instruction 3			F	D	E	S		
Instruction 4				F	D	E	S	

Clock Cycle

Oh Hong Lye / Cx1106 7

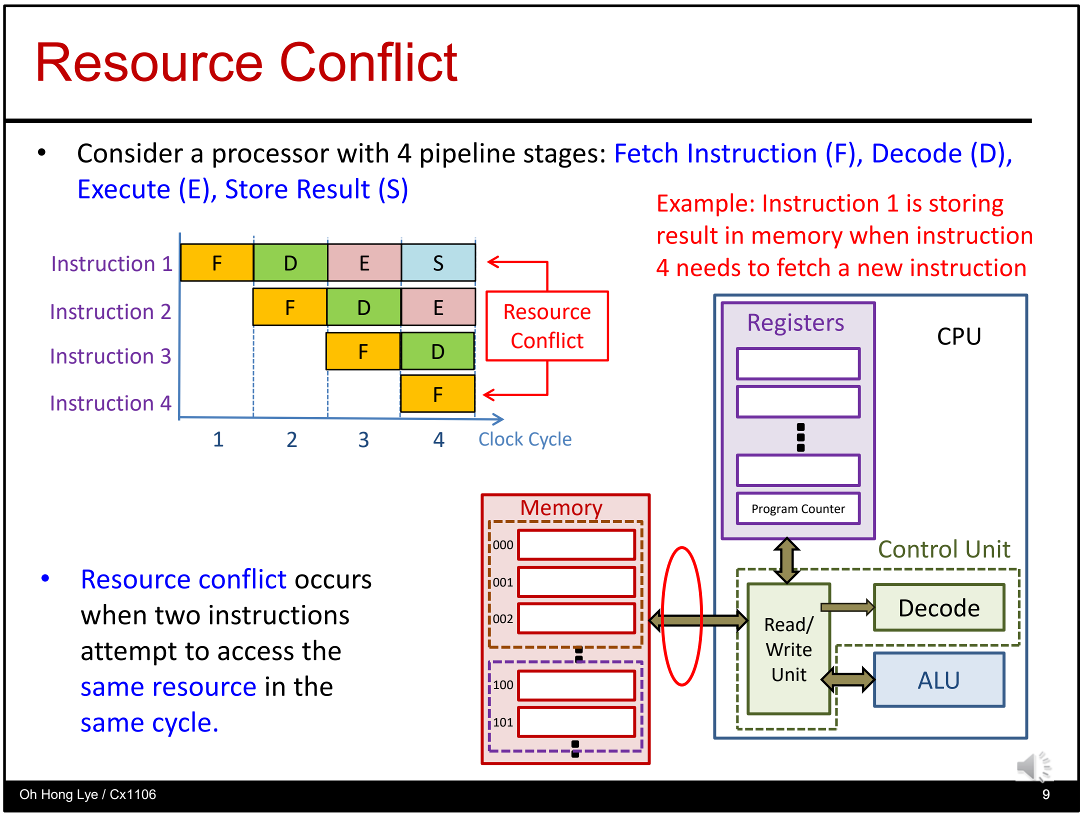
- Now, a pipeline's efficiency is maximised when the pipeline is filled, i.e. all the stages are in use.
- But it takes time to fill up a processor pipeline, the longer the pipe line i.e. number of stages, the longer it takes to fill it up.
- For example, in a processor with 4 pipeline stages, Fetch, 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)

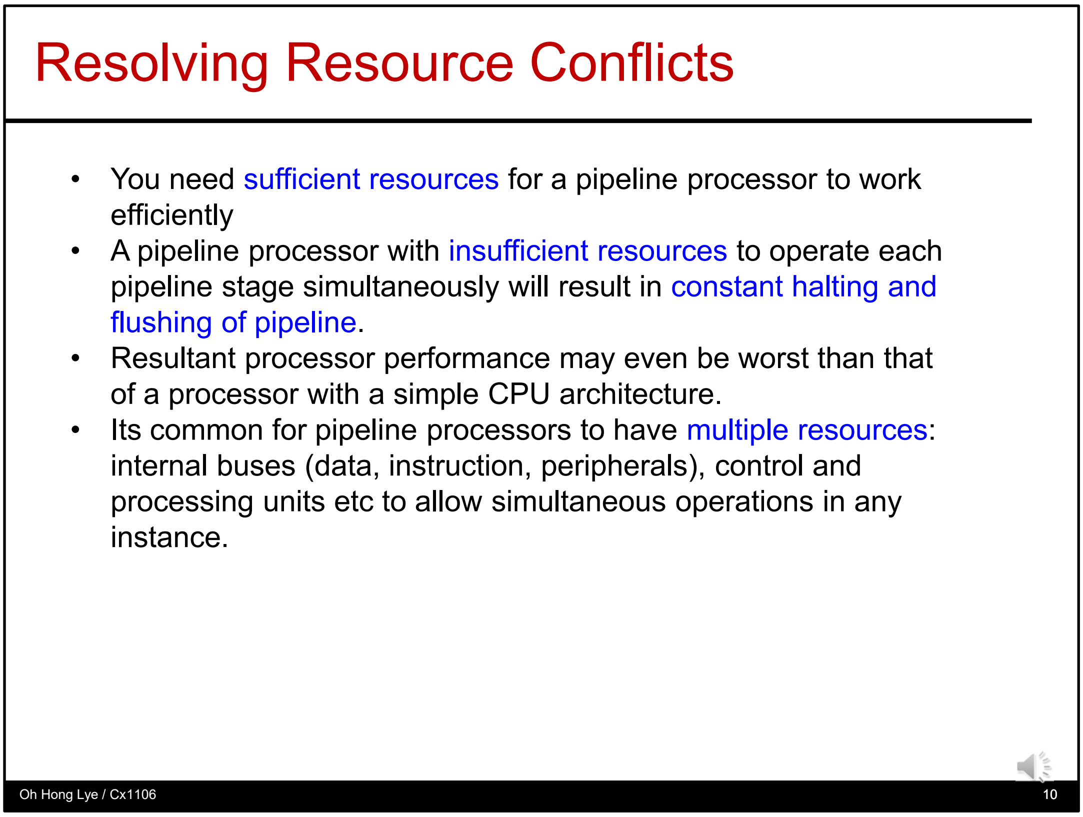
Example: Instruction 1 is storing result in memory when instruction 4 needs to fetch a new instruction

- 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.

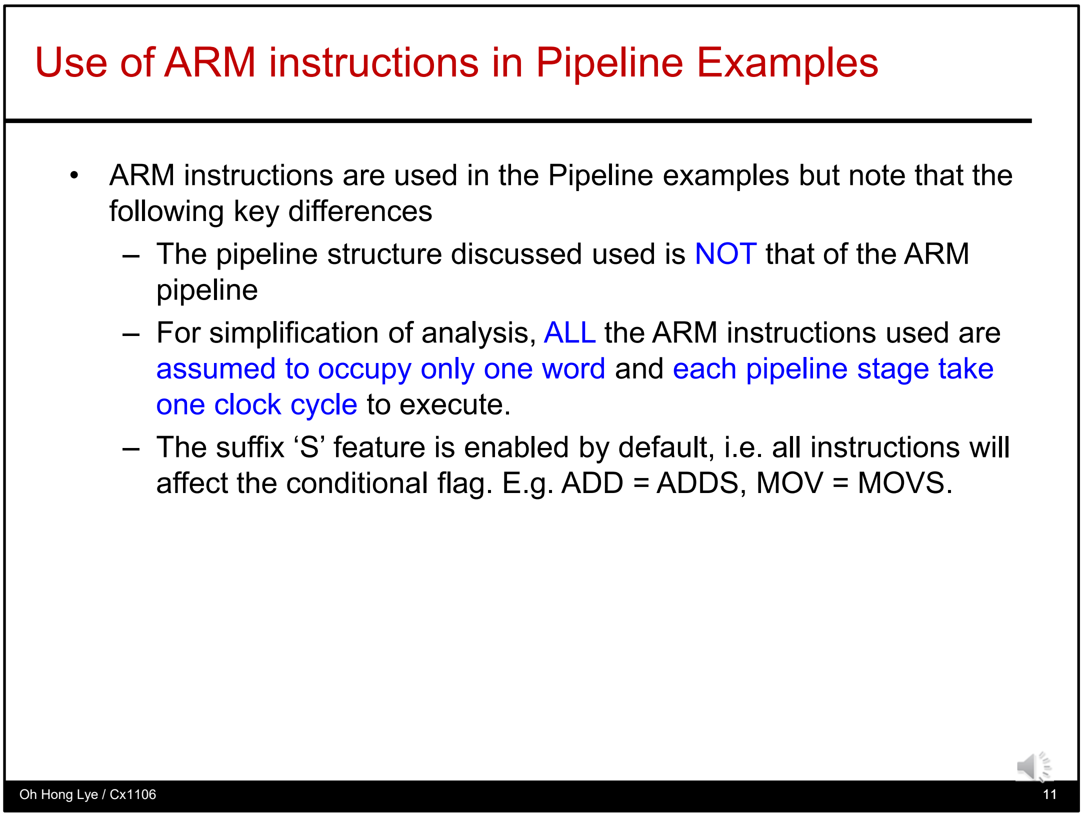


Resolving Resource Conflicts

- 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.

Oh Hong Lye / Cx1106 10

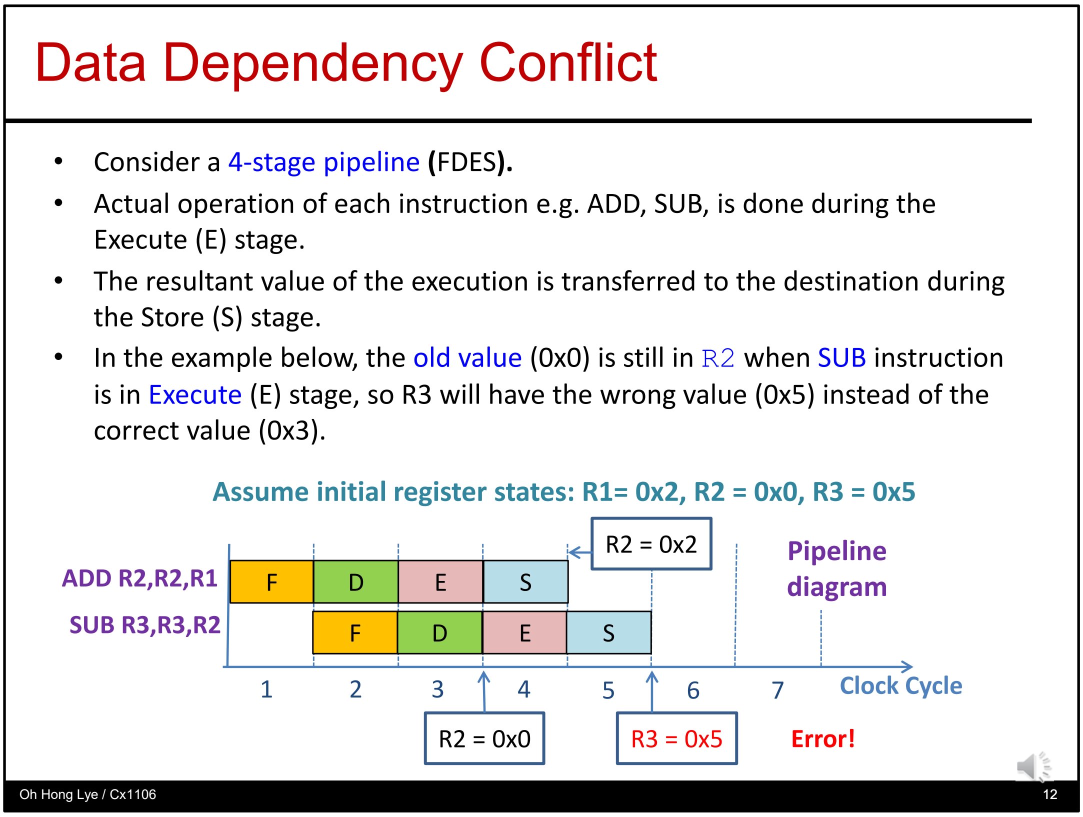
- There is really no other alternative in resolving resource conflict other than giving the processor what it needs to do its job.
- So you need to provide sufficient resources for a pipeline processor to work properly and efficiently.
- If the processor doesn’t have the resources it needs to operate 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.

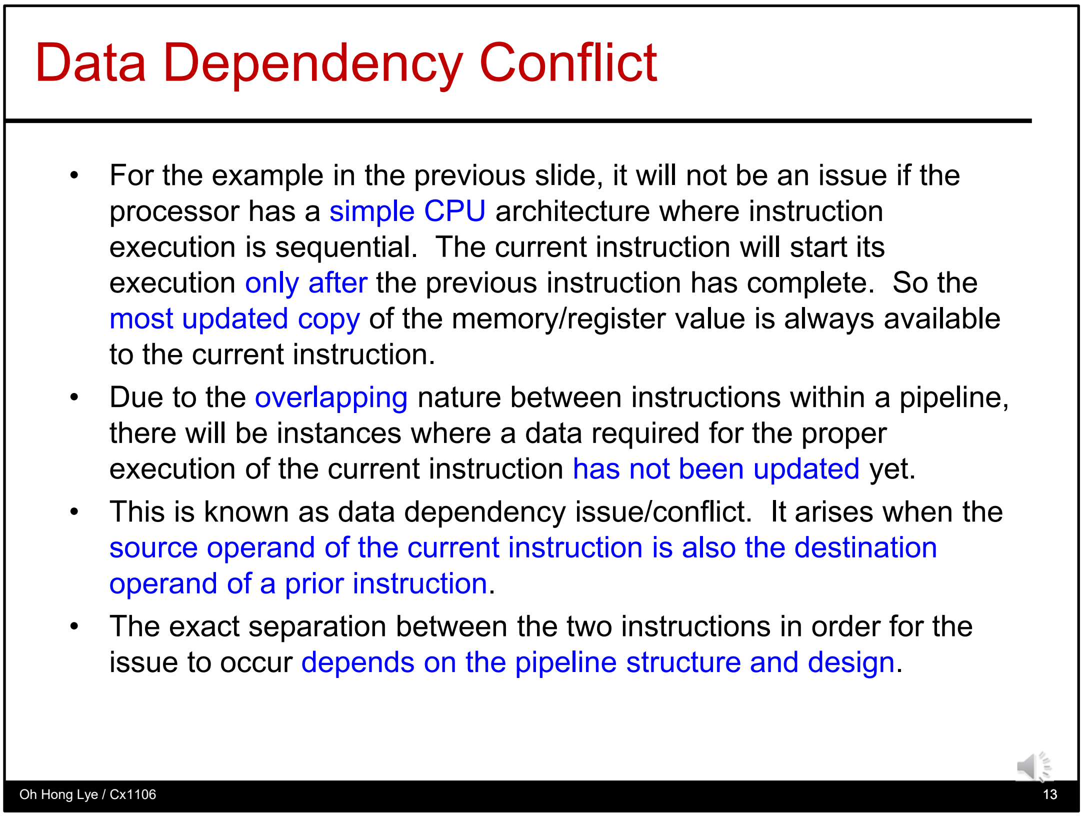


Data Dependency Conflict

- 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).

Assume initial register states: R1= 0x2, R2 = 0x0, R3 = 0x5

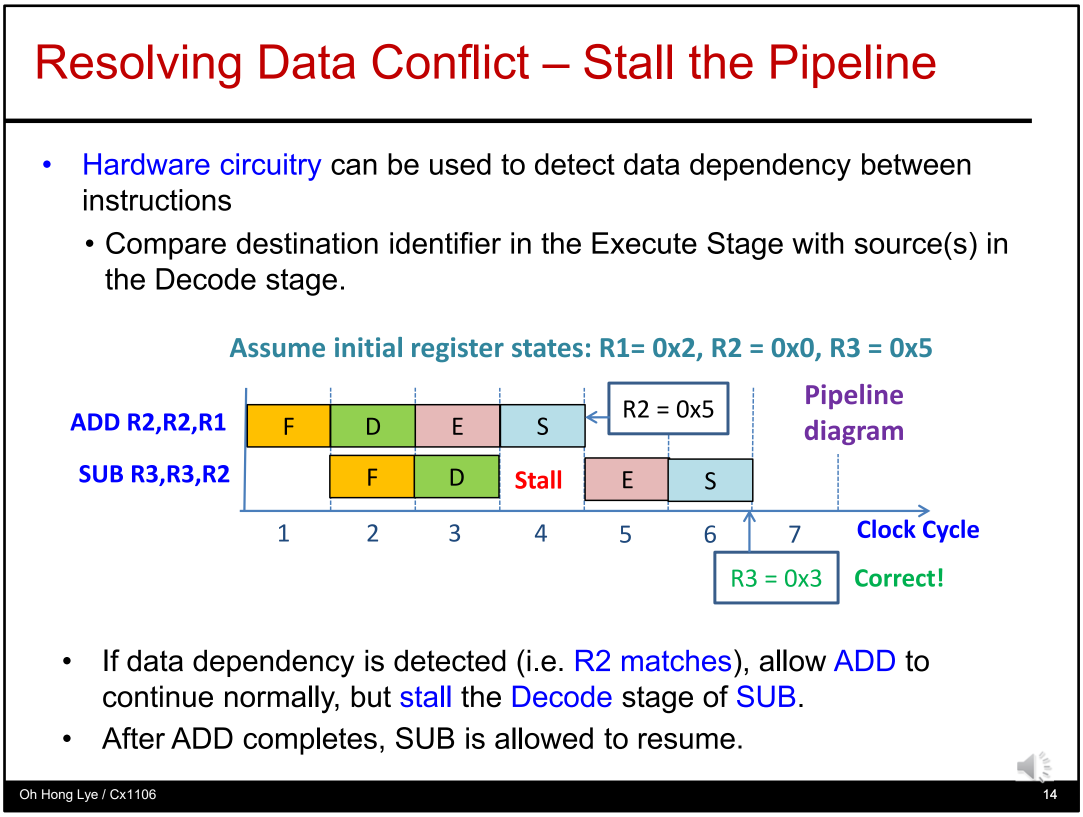
- The second conflict we will be discussing is data dependency.
  - Data dependency occurs when the current instruction has an operand that has not been updated in time for the current instruction's execution, and that resulted in wrong result being generated.
- I'll illustrate this with an example.
  - 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.

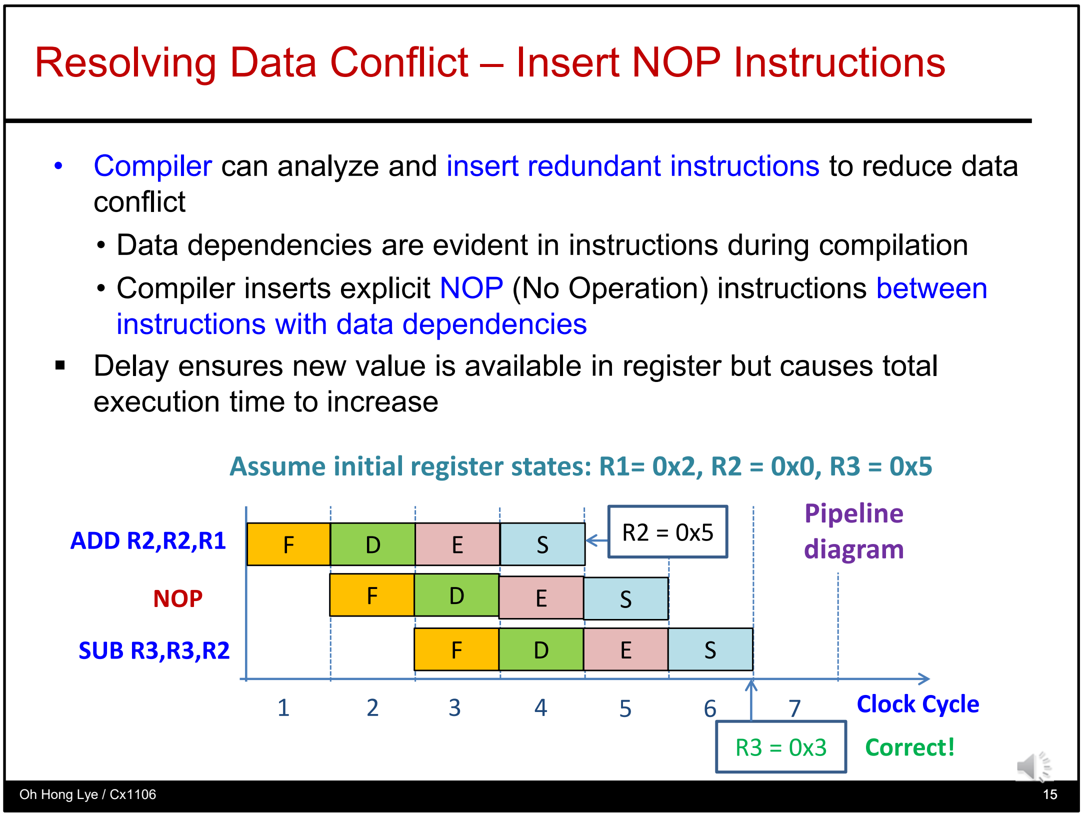
Assume initial register states: R1= 0x2, R2 = 0x0, R3 = 0x5

ADD R2,R2,R1
SUB R3,R3,R2
F
D
E
S
R2 = 0x5
Pipeline diagram
F
D
Stall
E
S
1
2
3
4
5
6
7
Clock Cycle
R3 = 0x3
Correct!

- 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.

Oh Hong Lye / Cx1106 14

- 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

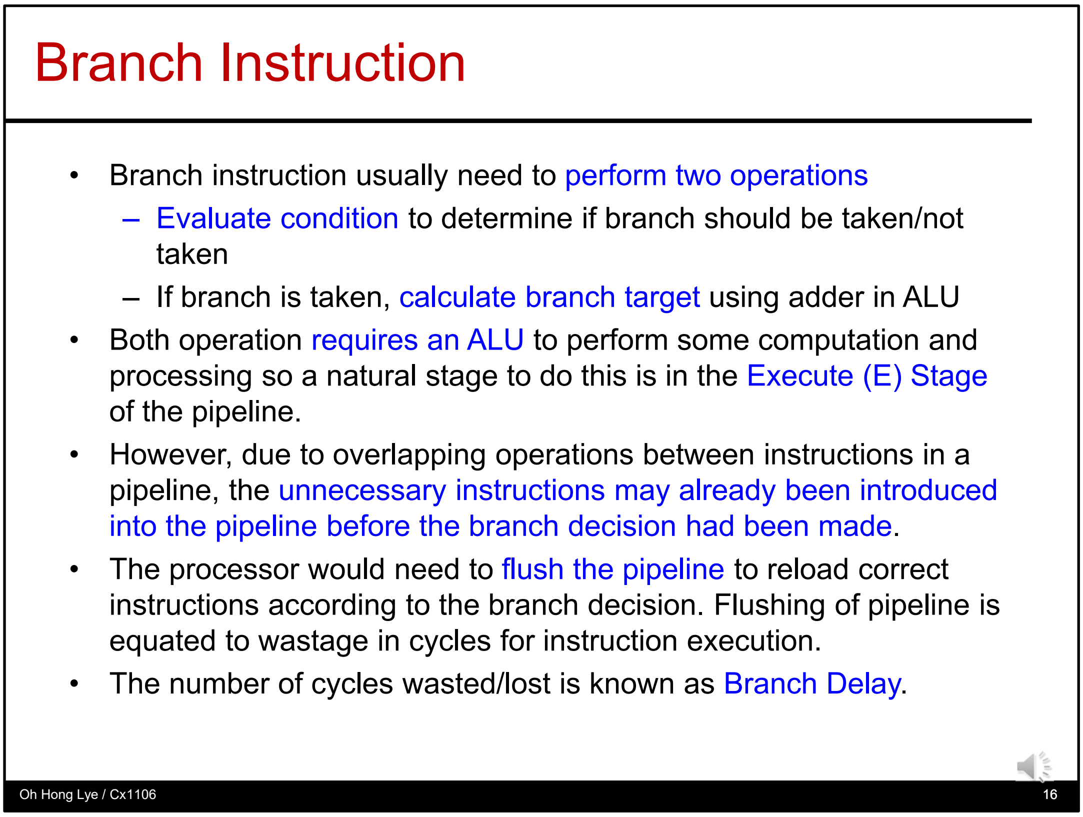
- Compiler can analyze and insert redundant instructions to reduce data conflict
  - Data dependencies are evident in instructions during compilation
  - Compiler inserts explicit NOP (No Operation) instructions between instructions with data dependencies
- Delay ensures new value is available in register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

Assume initial register states: R1= 0x2, R2 = 0x0, R3 = 0x5

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.

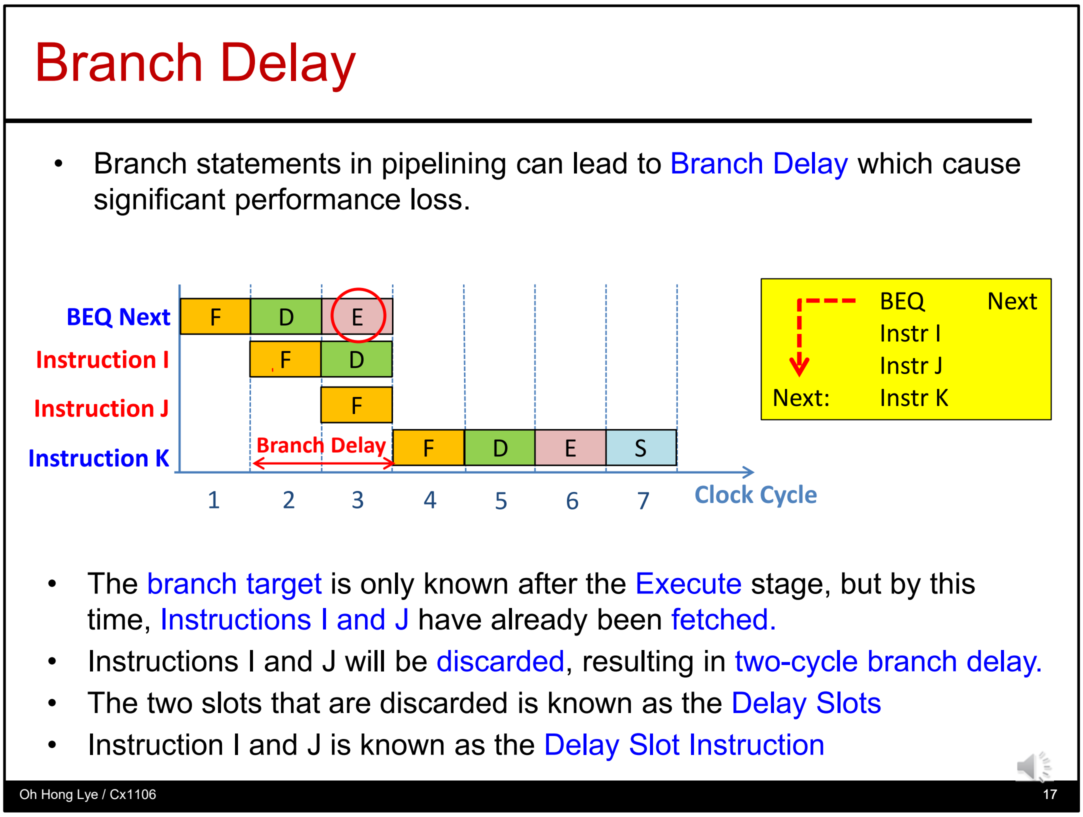


Branch Instruction

- 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.

Oh Hong Lye / Cx1106 16

- 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.



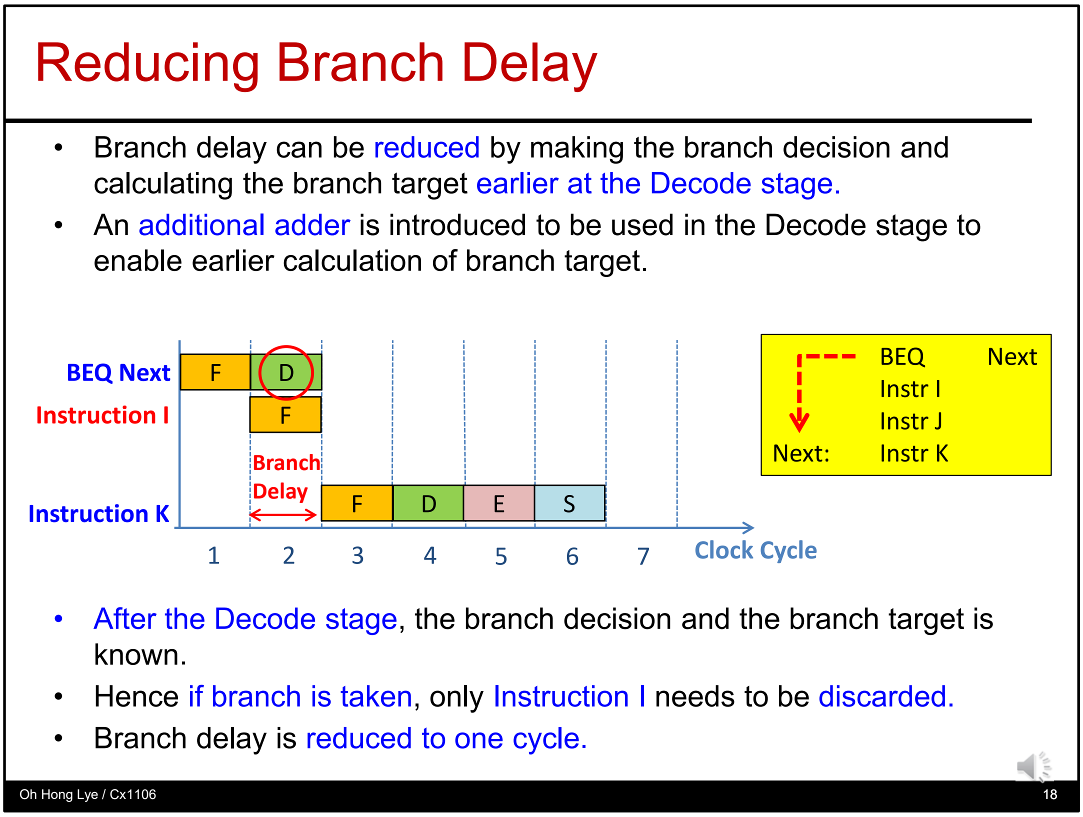
Branch Delay

- 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

Oh Hong Lye / Cx1106 17

- 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.



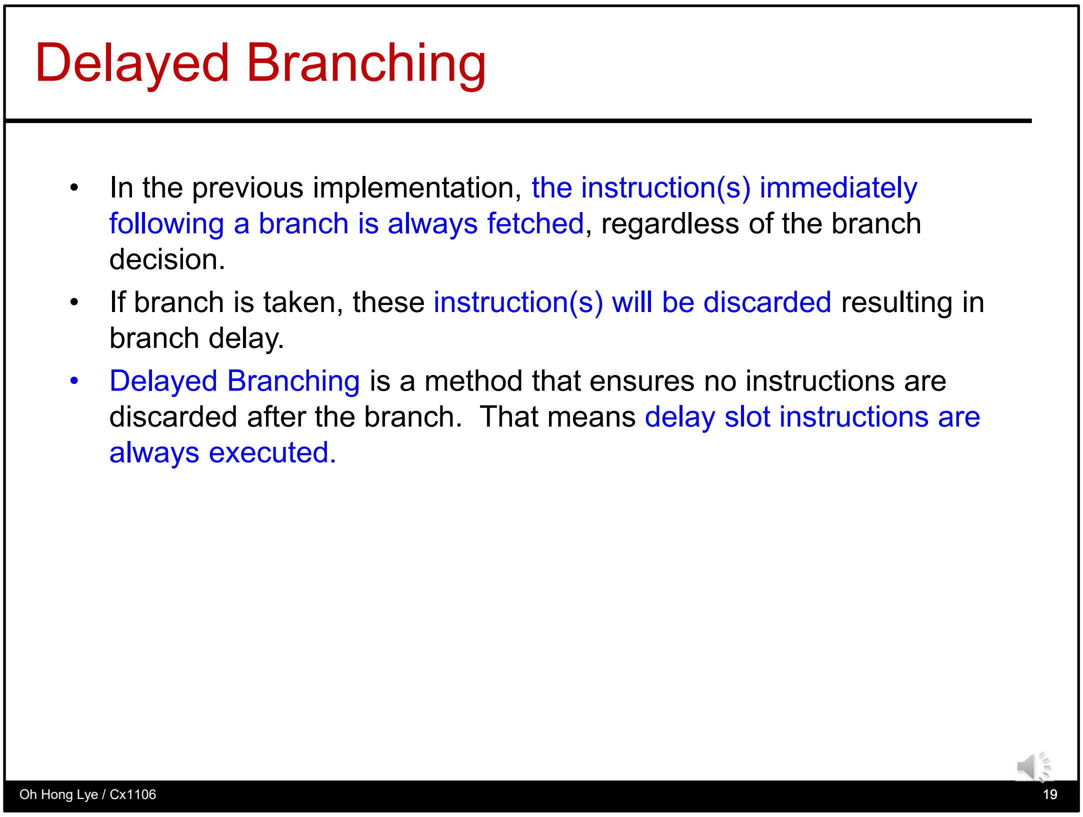
Reducing Branch Delay

- 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.

Oh Hong Lye / Cx1106 18

- 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.

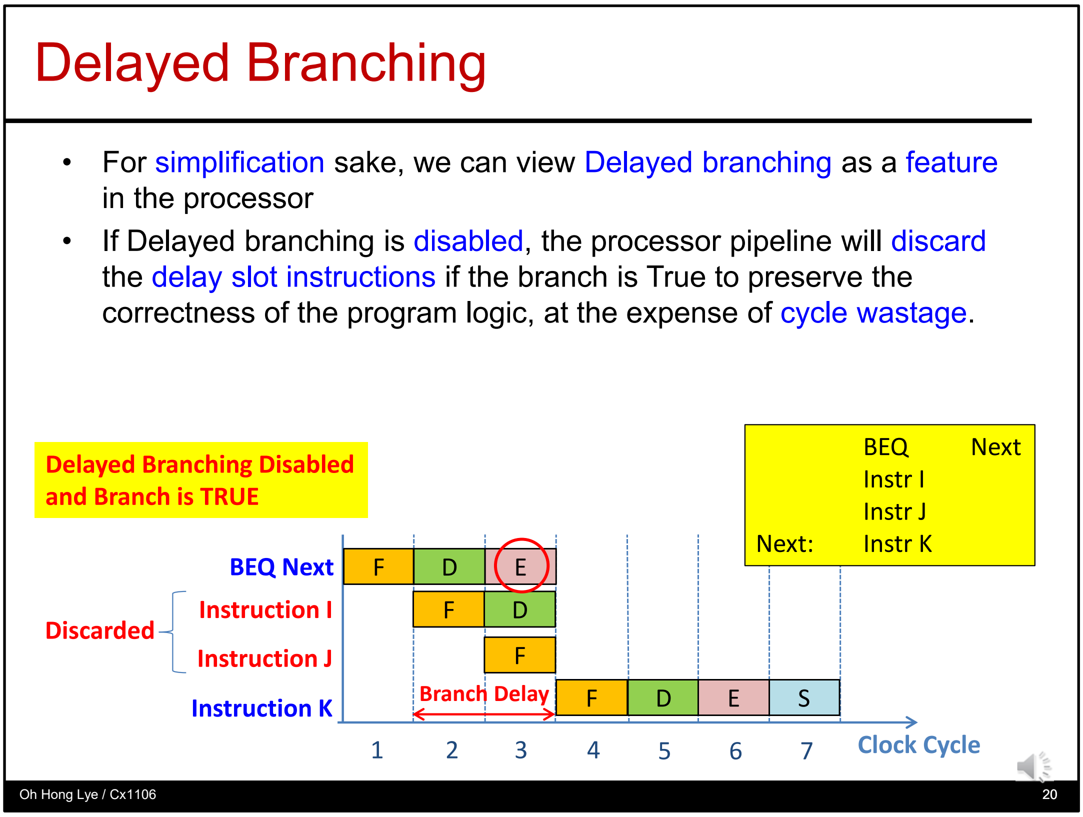


Delayed Branching

- 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.

Oh Hong Lye / Cx1106 19

- 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.

Delayed Branching Disabled and Branch is TRUE

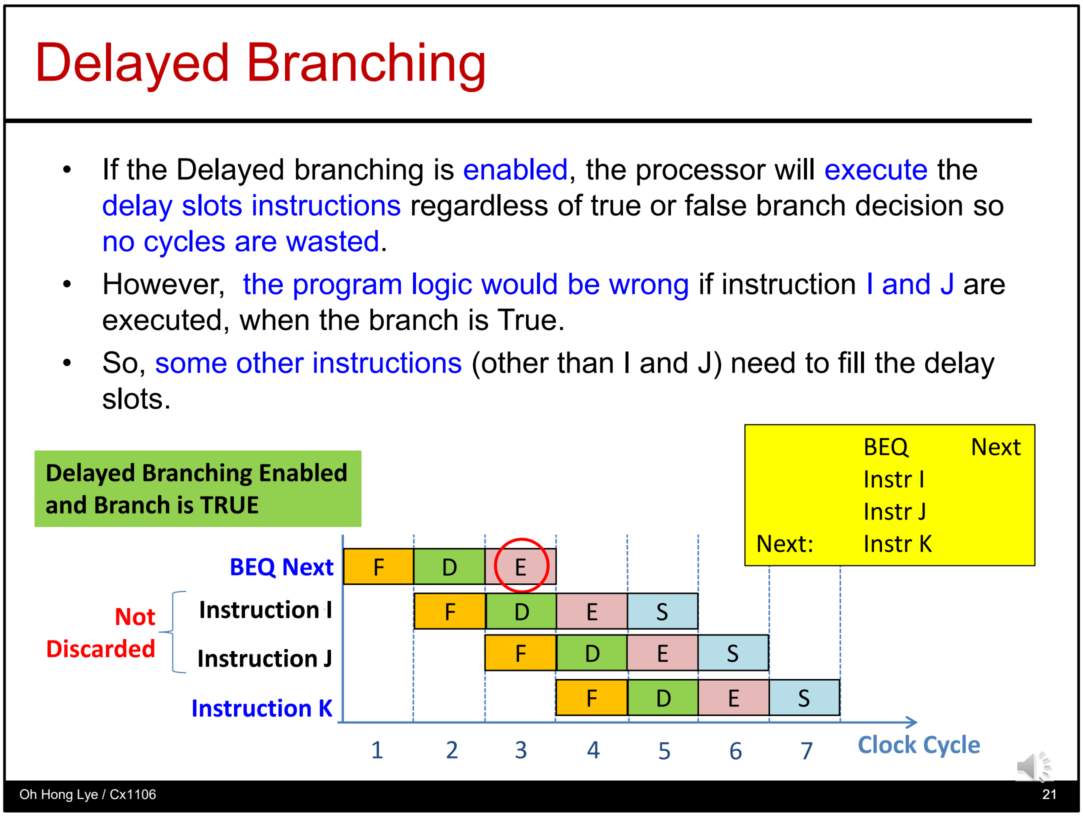
```
        BEQ      Next
        Instr I
        Instr J
Next:   Instr K
```

	1	2	3	4	5	6	7
BEQ Next	F	D	E				
Instruction I		F	D				
Instruction J			F				
Instruction K				F	D	E	S

Oh Hong Lye / Cx1106

20

- 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.

Delayed Branching Enabled and Branch is TRUE

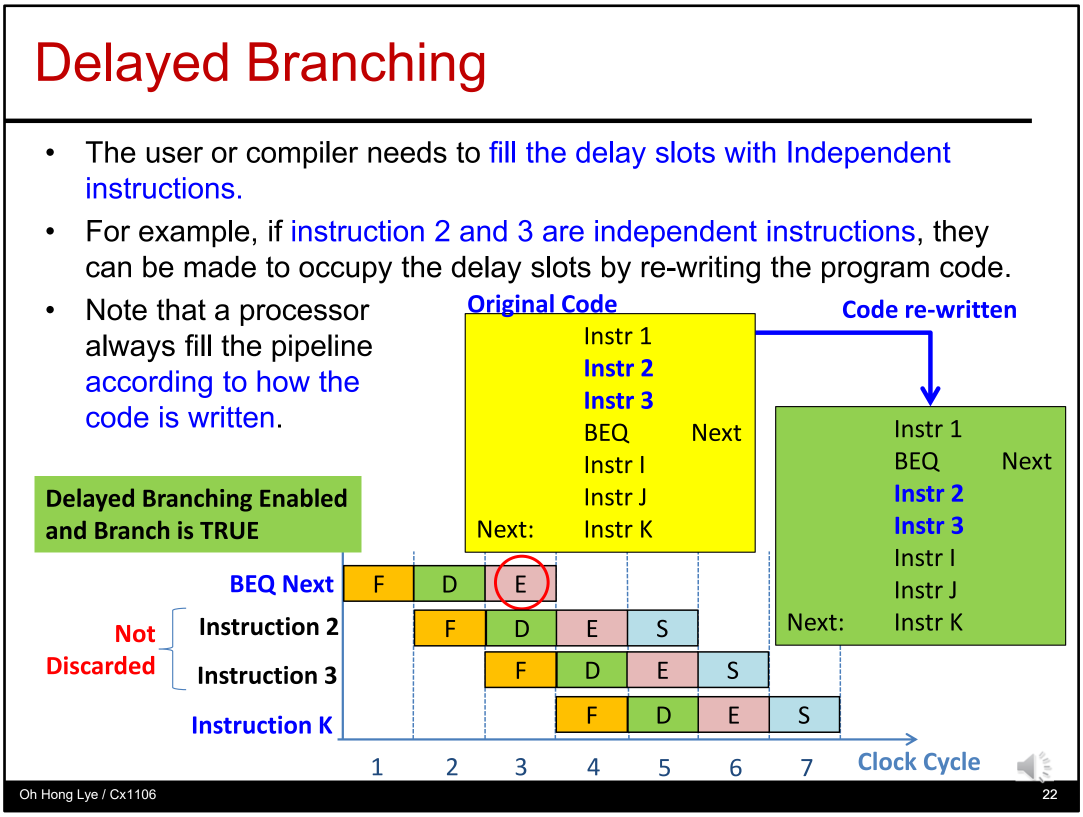
BEQ Next
Instr I
Instr J
Next: Instr K

	1	2	3	4	5	6	7
BEQ Next	F	D	E				
Instruction I		F	D	E	S		
Instruction J			F	D	E	S	
Instruction K				F	D	E	S

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.

Original Code

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

Code re-written

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

Delayed Branching Enabled and Branch is TRUE

	1	2	3	4	5	6	7
BEQ Next	F	D	E				
Instruction 2		F	D	E	S		
Instruction 3			F	D	E	S	
Instruction K				F	D	E	S

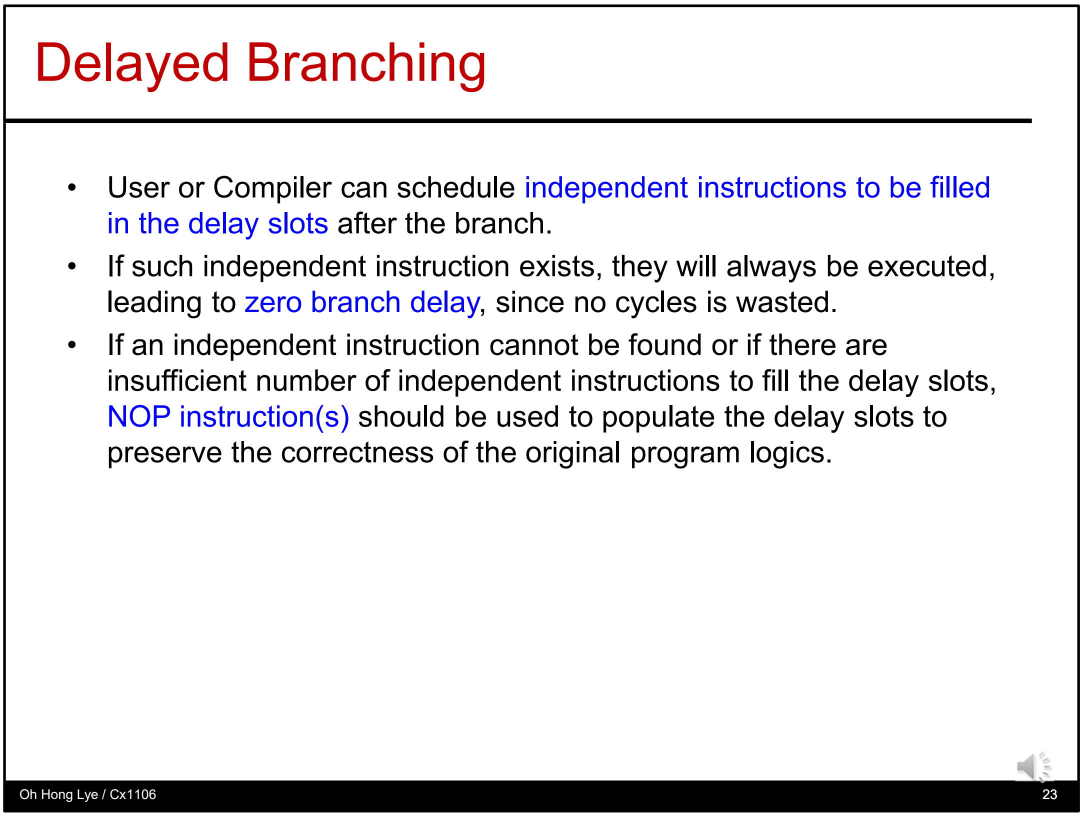
Not Discarded

Clock Cycle

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.

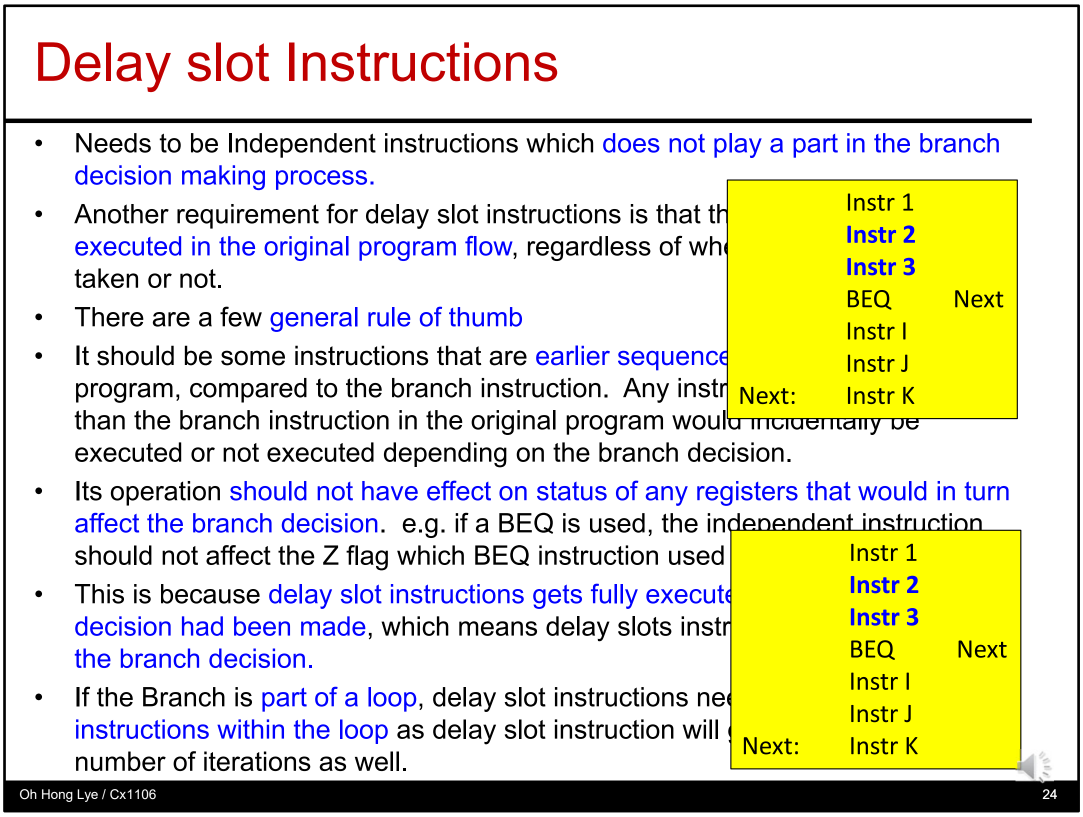


Delayed Branching

- 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.

Oh Hong Lye / Cx1106 23

- 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.

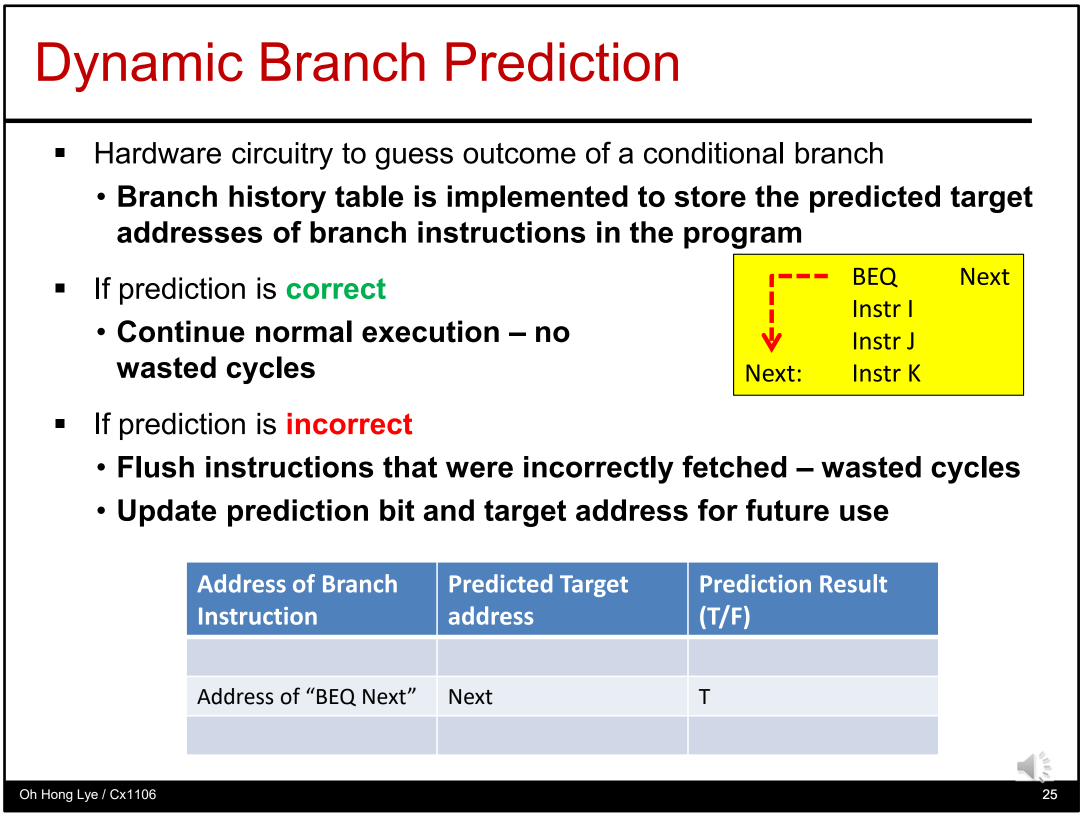
	Instr 1	
	Instr 2	
	Instr 3	
	BEQ	Next
	Instr I	
	Instr J	
Next:	Instr K	

	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

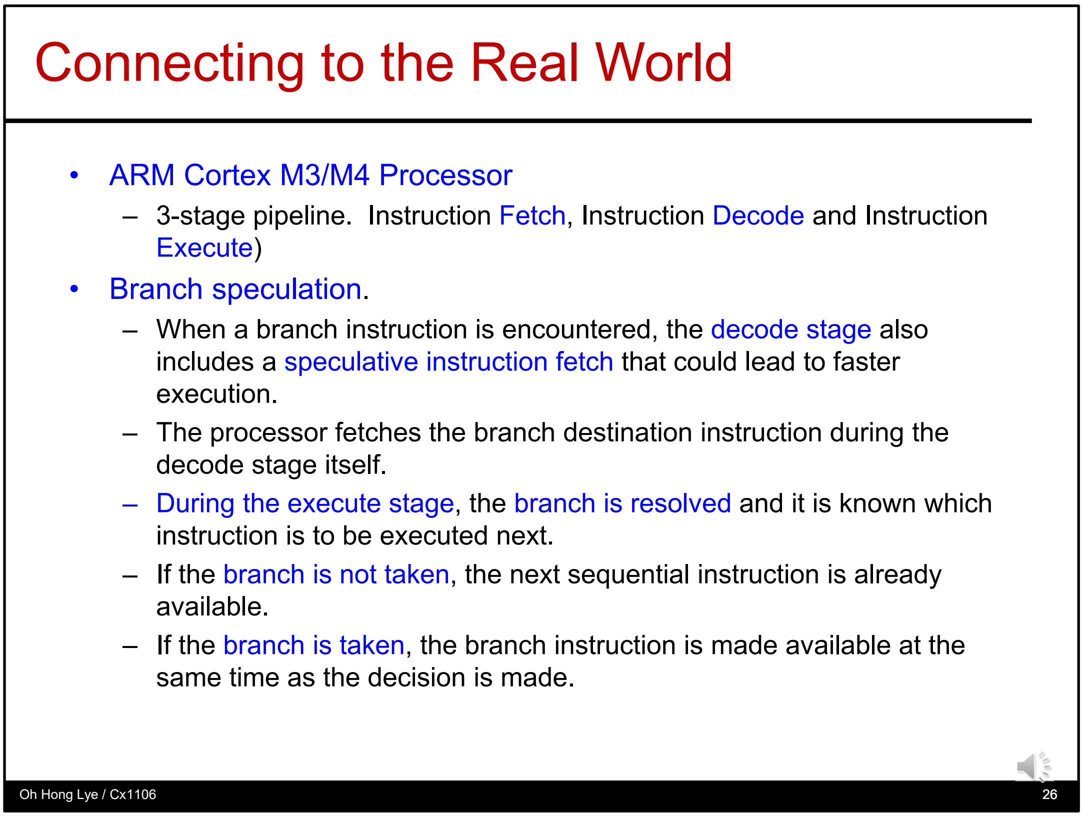
- Hardware circuitry to guess outcome of a conditional branch
  - **Branch history table is implemented to store the predicted target addresses of branch instructions in the program**
- If prediction is **correct**
  - **Continue normal execution – no wasted cycles**
- If prediction is **incorrect**
  - **Flush instructions that were incorrectly fetched – wasted cycles**
  - **Update prediction bit and target address for future use**

```
        BEQ      Next
        Instr I
        Instr J
Next:   Instr K
```

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.

Oh Hong Lye / Cx1106 26

- 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.