FIT1047 Week 5 Workshop

FIT1047 Introduction to Computer Systems, Networks and Security

Assignment 2

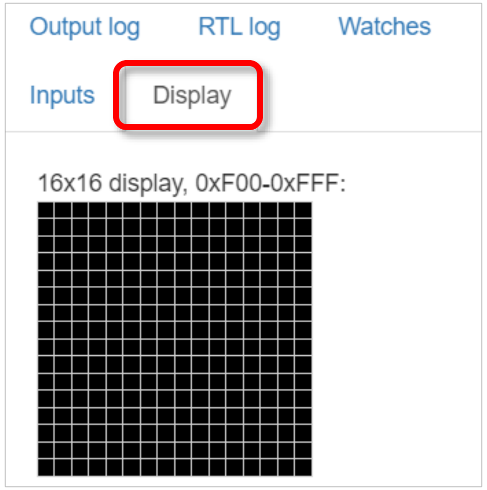
Moodle submission at the end of Week 6 **(Friday 29 November)**:

- **Part 1a**: Weekly reflection activity (Weeks 4-6).
- **Part 1b:** **Analyse and describe the processes** that are running on your computer.
- **Part 1c**: **Disassemble a MARIE program** (see exercise in this week's applied classes).
- **Part 1d**: Solve a **MARIE programming task**, demonstrating concepts such as direct/indirect addressing, conditionals and loops, subroutines, and using memory-mapped input/output.
- **Part 2:** Interview: **demonstrate** Part 1d submission to your tutor and answer some questions.

Activity 1: MARIE Memory-Mapped Graphics

In MARIE:

- Click on the Display tab (next to Output log, RTL log etc)
- This simulates a display with 16 x 16 pixels
  - Each pixel is mapped to a memory location in the range **F00-FFF**
  - Writing to memory changes the colour of the pixel (**0000** is black, **FFFF** is white)



Work in small group:

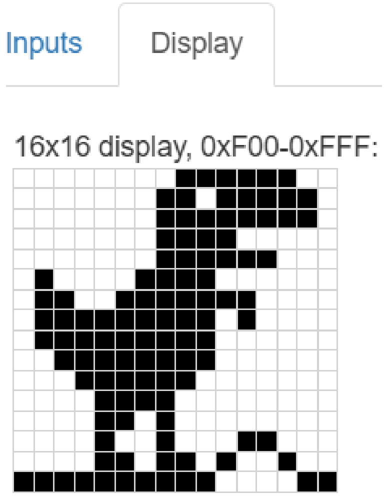
- Create a MARIE program that draws something on the graphics output display
- Share screenshots of your designs on the Ed forum!

Sample code in Notes below ↓

Activity 1: MARIE Memory-Mapped Graphics

Complete a MARIE program that prints an image to the Display.

- Use loops and indirect addressing to move existing image data from the memory to the Display with 16 x 16 pixels.



Pseudocode

1. Initialize DisplayPt with the address of the first pixel (0F00) in Display, and ImagePt with the starting address of image to draw
2. If DisplayPt exceeds the last pixel (0FFF), jump to Step 5
3. Else, move one pixel from image (ImagePt) to the Display (DisplayPt)
4. Increase ImagePt and DisplayPt to the next pixel, then jump Step 2
5. Halt the program

```
// Variables
One, DEC 1
FirstPixel,     HEX F00     / First pixel in Display
LastPixel,      HEX FFF     / Last pixel in Display
DisplayPt,      HEX 000     / Temporary Pointer for Display
ImagePt,        HEX 000     / Temporary Pointer for image
DinoPt,         ADR Dino    / Fixed Pointer for image
```

You may use the MARIE template from Moodle: Template

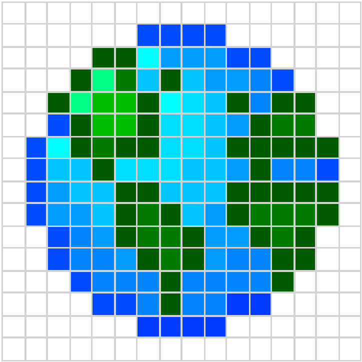
Activity 1: MARIE Memory-Mapped Graphics

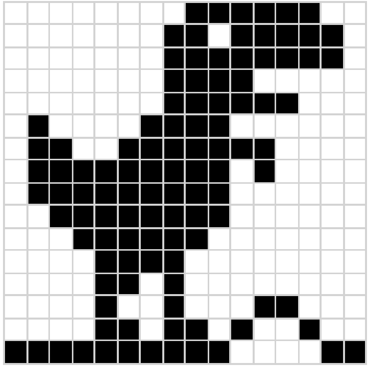
Complete a MARIE program that prints an image to the Display.

- Convert the working program to a subroutine DisplayImage.
- User can choose to draw any hardcoded image using this subroutine.

A well-designed subroutine should (Ed Forum):

- Has a clear purpose
- Isolated from other parts of the code
- Has minimal side effects on unrelated code components
- Ensure reusability and flexibility
  - via good arguments (variables, or parameters)





You may use the MARIE template from Moodle: Template

The solution will not be provided!

Pre-Workshop Lecture Content

1. Input and output
2. I/O and CPU
3. Polling, interrupts and DMA

Explain how I/O devices communicate with the CPU

4. The boot process

Ed Lesson

5. Operating systems
6. Multi-processing

Understand how a single CPU execute multiple processes at once

Explain simple process scheduling algorithms

7. Virtual memory

Applied Session

MONASH University

Learning Outcomes

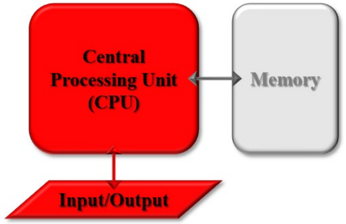
At the end of this workshop, you will be able to:

1. Explain how I/O devices communicate with the CPU

2. Understand how a single CPU can appear to execute multiple processes at once

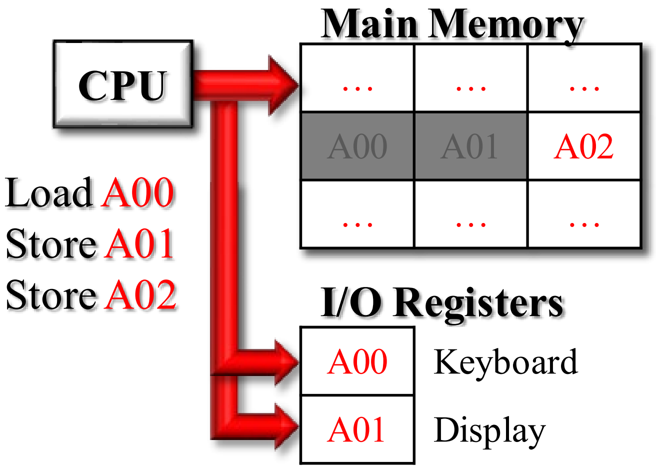
3. Explain simple process scheduling algorithms

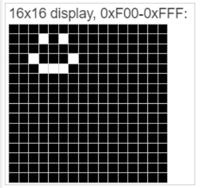
Activity 2: MMIO vs PMIO (Recap)



How does CPU access I/O devices?

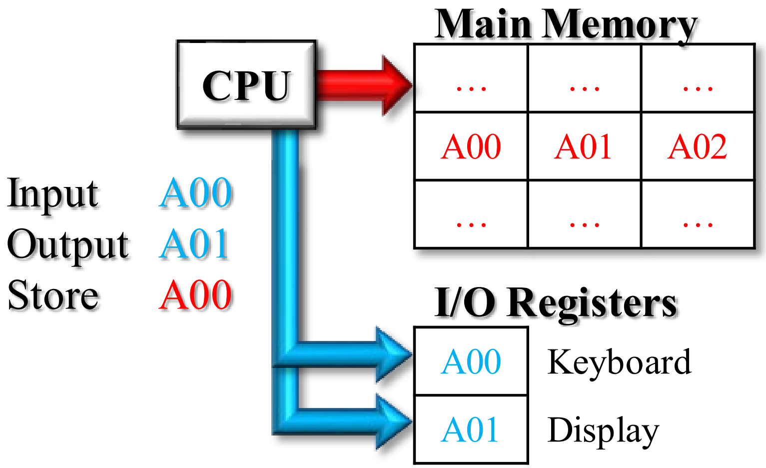
Memory-mapped I/O (MMIO)

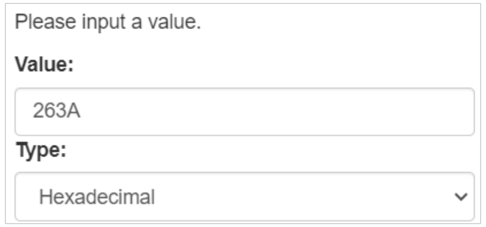


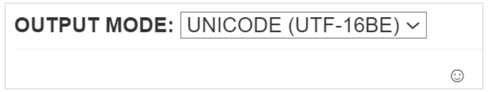


- ➢ Reserve memory address to (as) I/O registers
- ➢ Do not require new instruction (**Load** and **Store**)
- ✓ Simple ISA (do not need new instructions)
- ✓ Cheaper (e.g. shared buses)
- × Risk of unauthorized or unintended access between memory and I/O devices
- × Reduce usable memory space

Port-mapped I/O (PMIO)



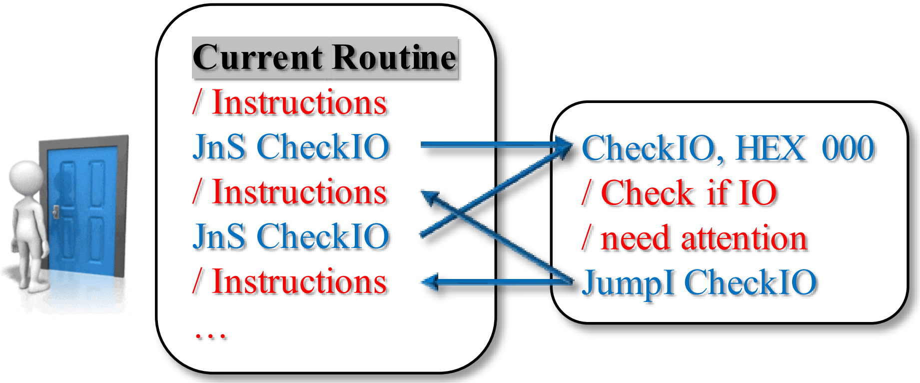


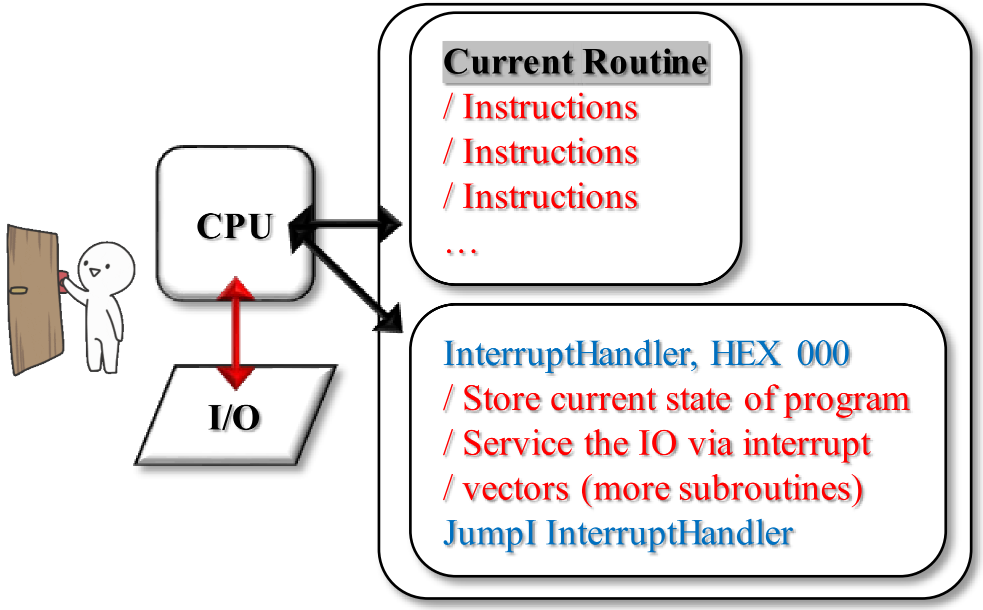


- ➢ Separate I/O registers from memory
- ➢ Require new instructions (**Input** and **Output**)
- ✓ Better isolation between memory and I/O devices
- ✓ Memory fully utilized
- × Complex ISA (new instructions)
- × Costly (extra instructions → extra hardware)

Activity 2: Programmed vs Interrupt-Based I/O (Recap)

When does CPU access I/O devices?





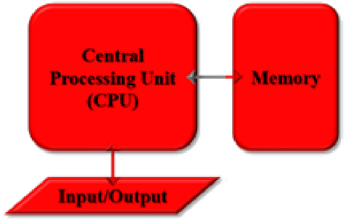
Programmed I/O (e.g. Polling)

- ➢ Codes periodically check I/O (software)
- ➢ CPU pause the current routine & check I/O
  - ➢ If needed, service the I/O register before resume
- ✓ Efficient when many I/O access needed

Interrupt-based I/O

- ➢ CPU is notified when it should service I/O (hardware)
- ➢ I/O sends interrupt signal to notify CPU
  - ➢ CPU pause the current routine and service I/O register through interrupt handler
- ✓ Efficient when less I/O interaction needed

Activity 2: Direct Memory Access (DMA) (Recap)



Briefly discuss the working principle of DMA.

I/O ↔ CPU ↔ Memory: Both I/O access need involvement of CPU

× CPU stuck when dealing with large data transfer or slow I/O devices

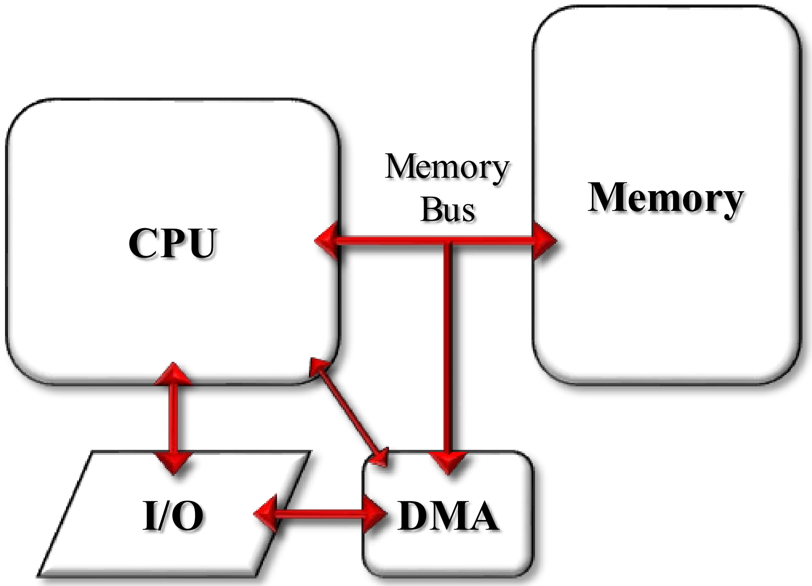
- Image, audio, video, network data, etc.

DMA (hardware)

Controller to handle the transfer of data between memory and I/O registers

1. CPU initiates DMA for I/O services (programmed I/O)
2. DMA replaces CPU to handle data transfer
   - ✓ CPU can have more time to execute the main program
3. DMA interrupts CPU at the end of the service (interrupt-based I/O)

*CPU and DMA share same memory bus, need take turn to access memory



Learning Outcomes

At the end of this workshop, you will be able to:

1. Explain how I/O devices communicate with the CPU

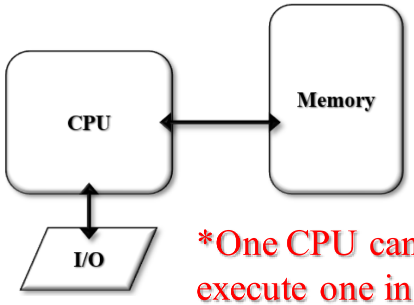
2. Understand how a single CPU can appear to execute multiple processes at once

3. Explain simple process scheduling algorithms

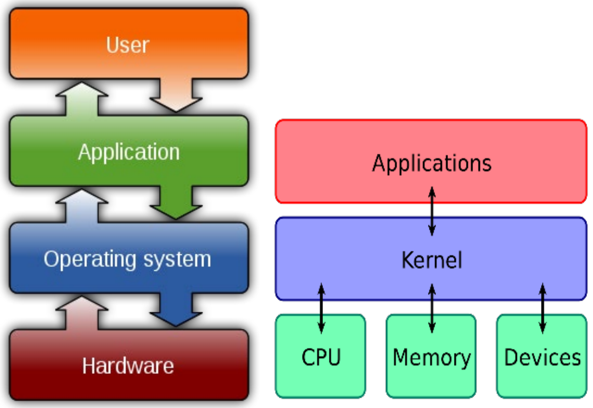
Activity 3: Operating System (OS)

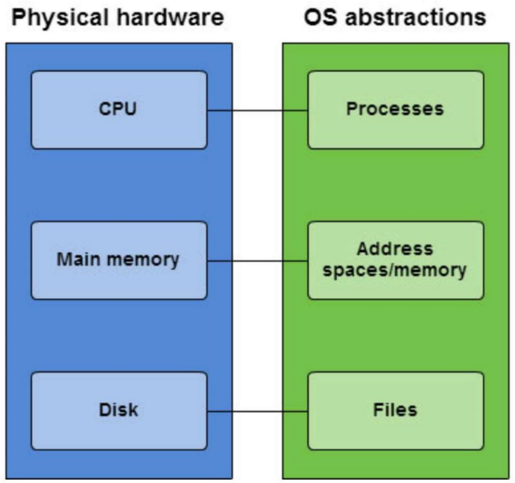
What does OS do?

- A program that provides an abstraction layer between hardware and user
- Make the computer easier to be used by end users and programmers
- Create illusion of multiple processes running simultaneously
  - Process switching
- OS Kernel: Core functionality
  - Process management
  - Memory management
  - I/O management



*One CPU can only execute one instruction at any instance!





Activity 3: OS Kernel

Explain how kernel works and how user processes access I/O.

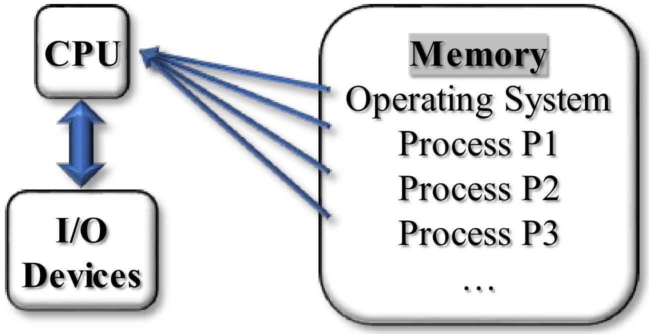
!! Dangerous if buggy or malicious process run in Kernel

Kernel Mode

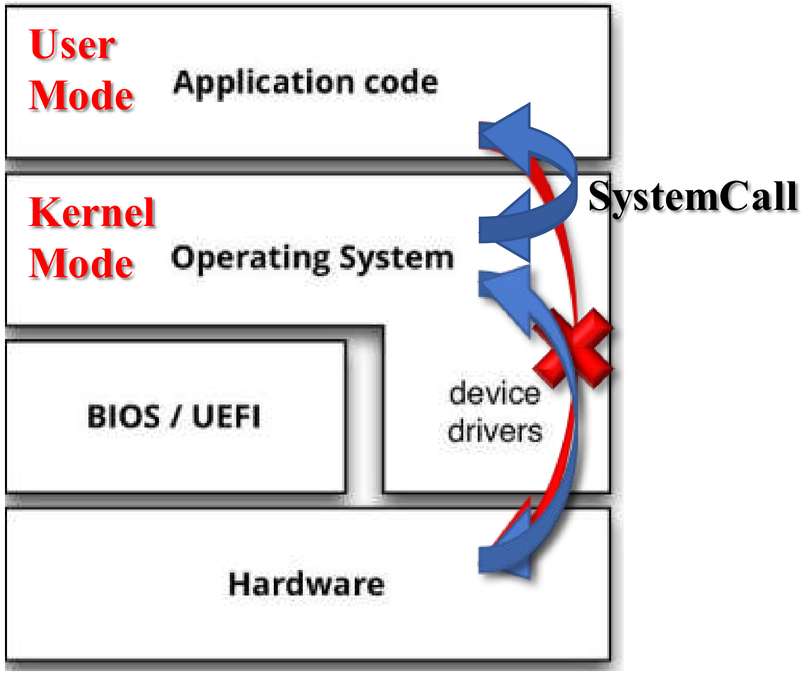
- OS and most drivers
- Unrestricted code
- I/O access

User Mode

- Normal user processes
- Limited instruction set
- No I/O access



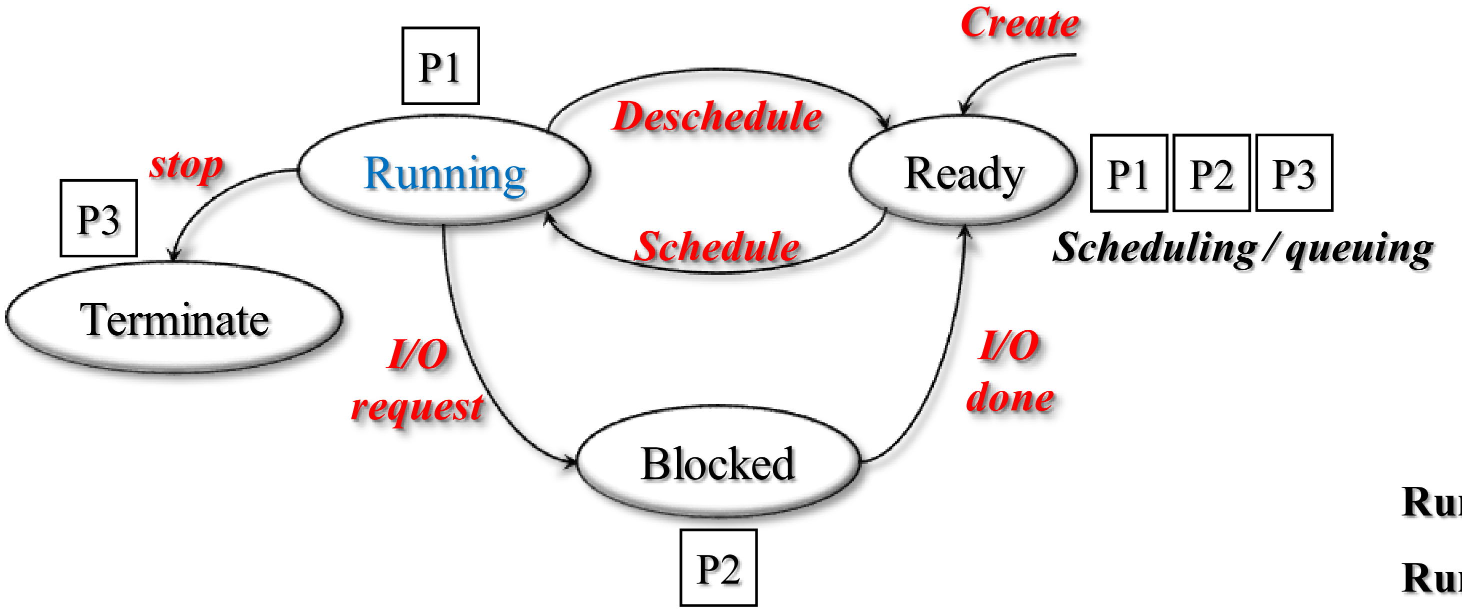
- One CPU can only run one process at any instance
- Switching of processes (including via System Call) will trigger **context switch**.

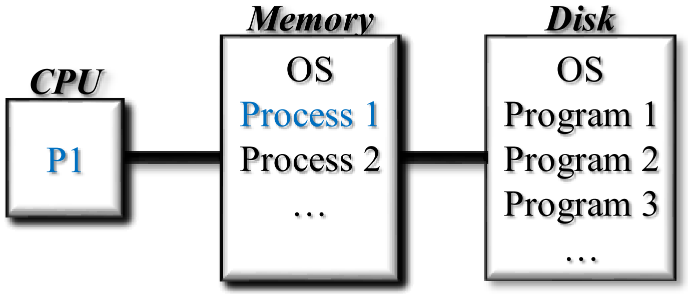


Context Switching:
A process of storing the state of a switched process (eg: AC, PC) so that it can be restored and resume execution at a later point.

Activity 4: State of Processes

Explain the state of the processes and show how process switching can be done.





	Process 1	Process 2	Process 3
Run P1 in CPU	Running	Ready	Ready
Run P2 in CPU	Ready	Running	Ready
P2 requests for I/O access	Ready	Blocked	Running
I/O access done by OS	Ready	Ready	Running
P3 done and terminated	Running	Ready	Terminate

- Processes can be one of three states
- Only one process in running state (per CPU)
- Blocked state cannot revert back to running state

MONASH University