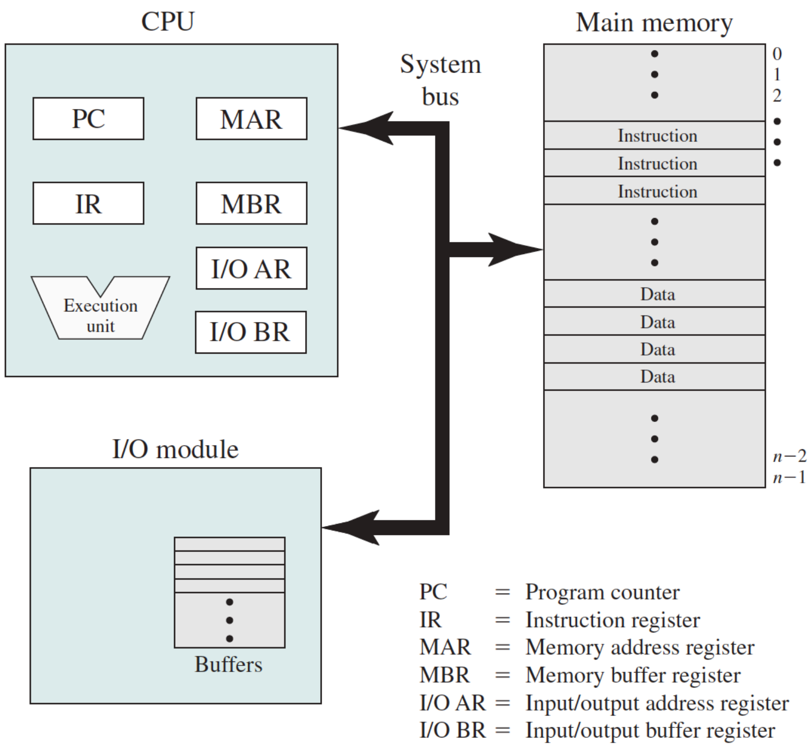
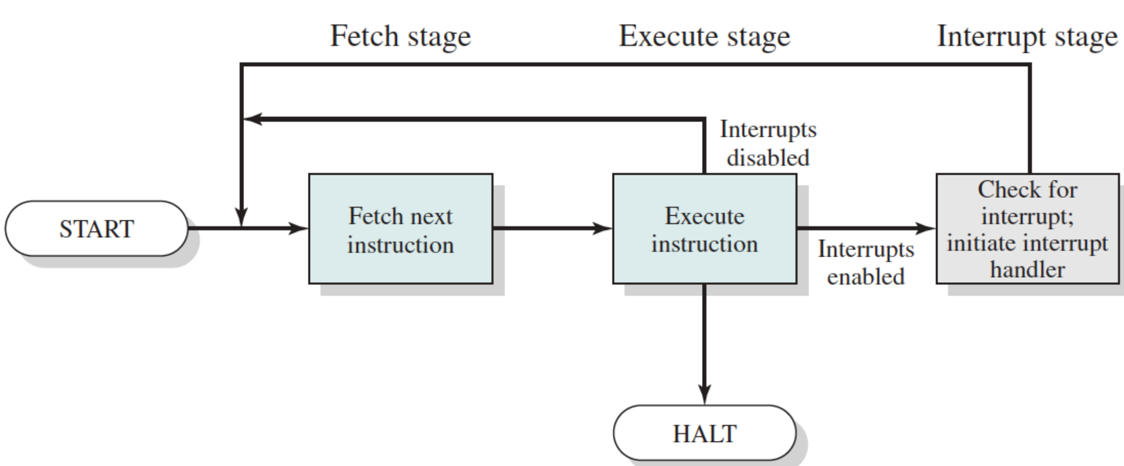
Chapter 1 (1.1 – 1.8)

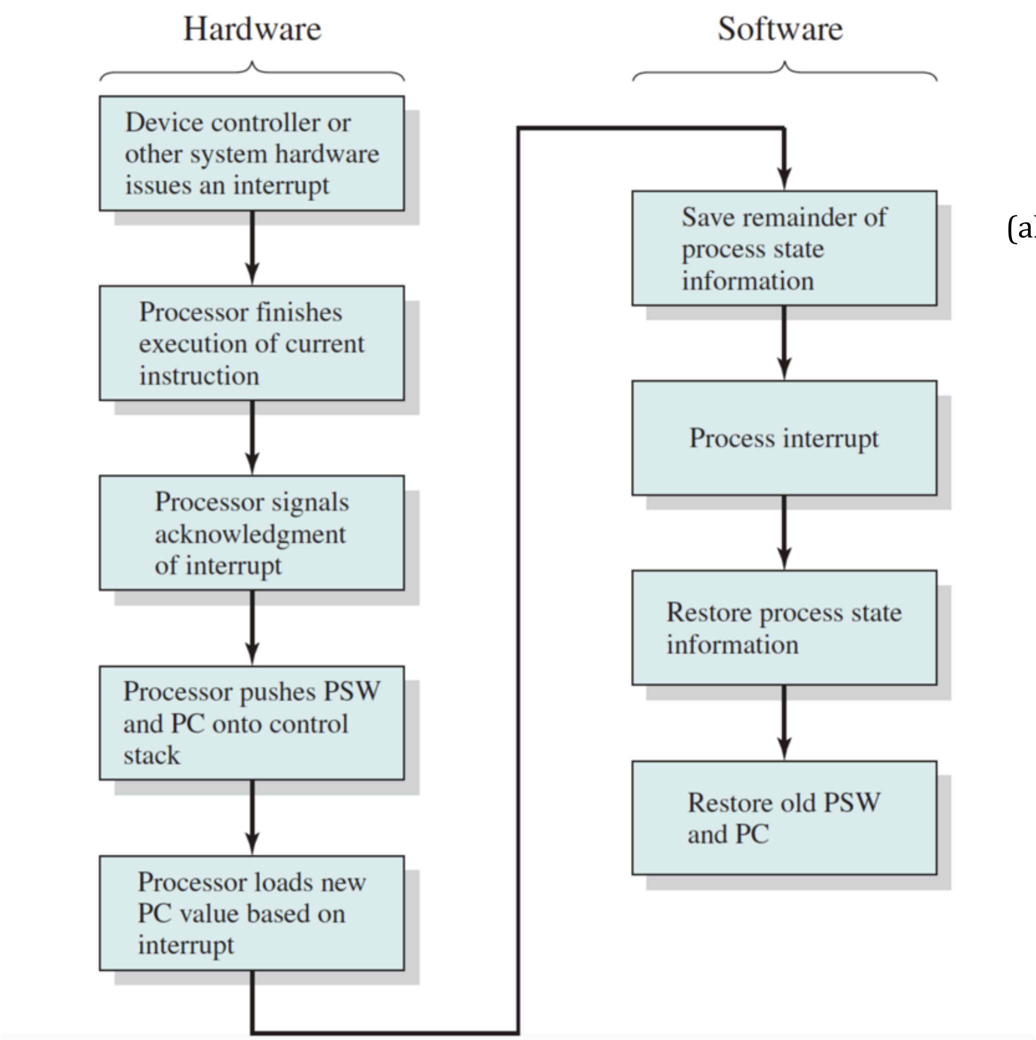
- Overview:



- Generally 4 types of instructions:
  - Processor ↔ memory
  - Processor ↔ I/O
  - Data processing (e.g. arithmetic/logic)
  - Control (e.g. branching)
- **Instruction cycle**
  - Fetch next instruction (address pointed to by PC) place into IR (via MAR & MBR)
  - Execute instruction (instruction contains opcode & target memory address)
  - (*If interrupts enabled*) check for interrupts; if present, initiate interrupt handler



- **Interrupts**
  - Types of interrupts:
    - Program, timer, I/O, hardware failure
  - Increases processor utilization because <u>I/O devices are much slower than the processor</u>
  - Steps of an interrupt:

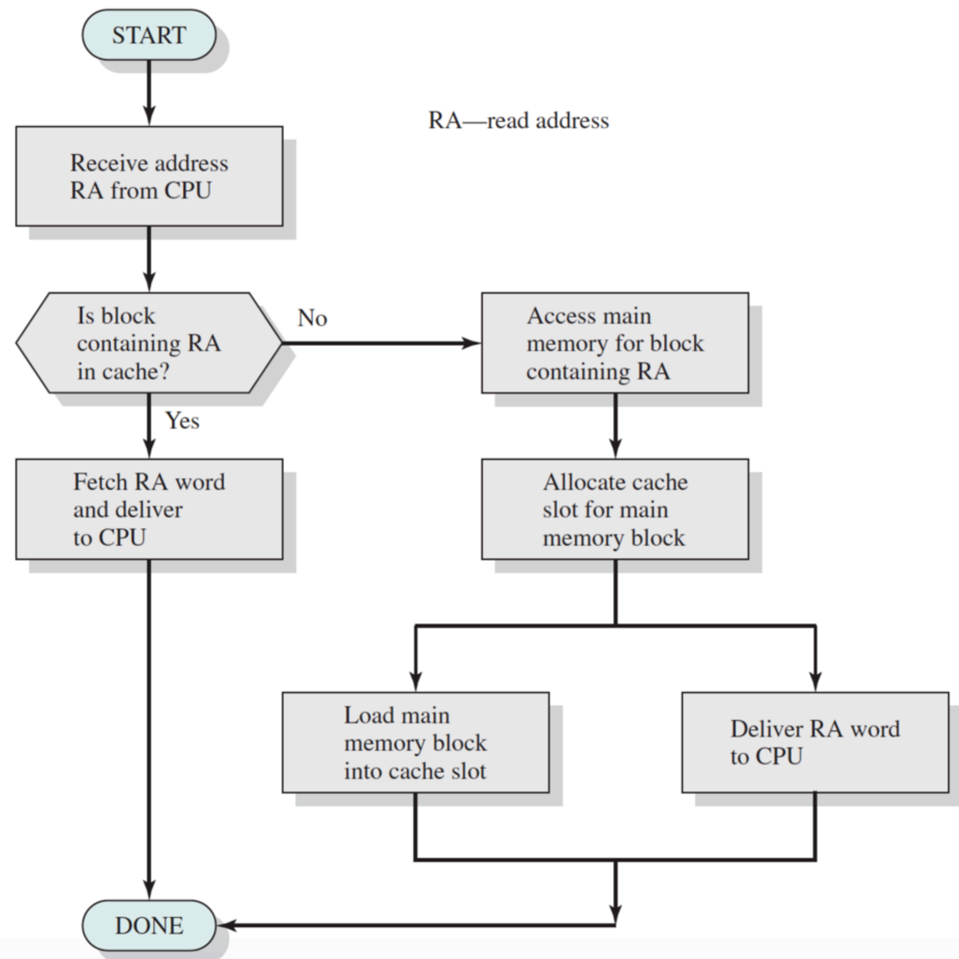


(aka. registers)

  - Multiple interrupts
    - Sequential processing – disable interrupts during interrupt
      - Con: no priority
    - Nested processing – high priority call can interrupt low priority interrupt call

- **Caching**
  - Useful because of <u>locality of reference</u>
    - *Tendency for memory references by a program to cluster in the same region*
  - Goal: organize data so that accesses on each level is faster than the on the next level
  - Main memory contains many <u>blocks</u> (size = K words)
  - Cache contains <u>lines</u> (size = K words) – much fewer than the # of blocks in memory
  - Design strategies:
    - Cache size

- Block size
- Mapping function (where in cache to place new blocks)
- Replacement algorithm (e.g. least recently used/LRU)
- Write policy – when to update changes in cache to memory
  - Every time block is updated (<u>write through</u>)
  - Only when block is replaced (<u>write back</u>)
- Number of cache levels

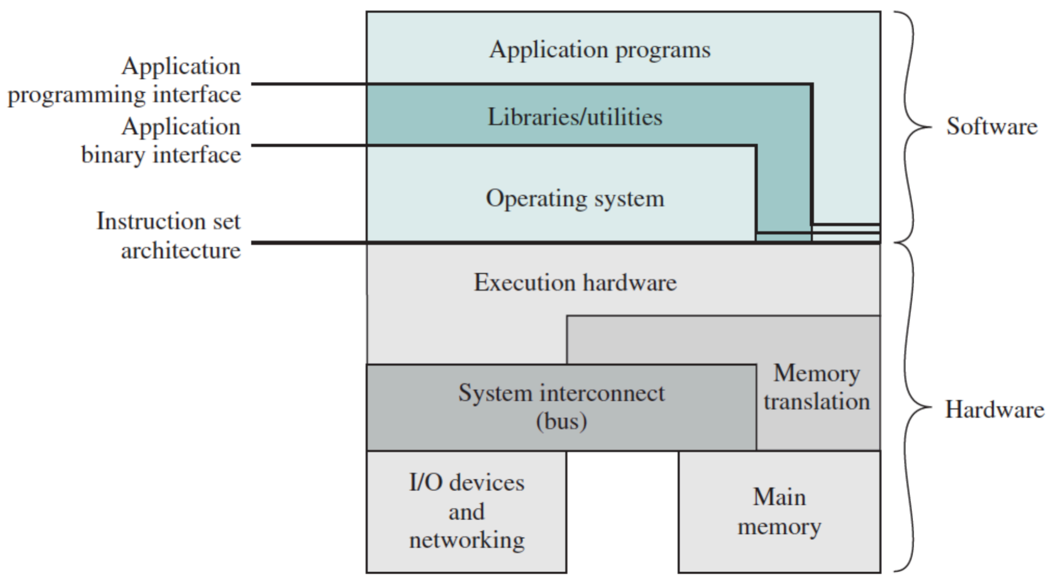


- Programmed I/O – aka. busy waiting
  - Processor checks status bit in I/O module until complete
- Interrupt-driven I/O – uses interrupts
  - Processor must handle I/O transfer – every word read/written needs to go through processor
- Direct memory access (more in Chapter 11)
  - More efficient for bulk data transfers
  - Processor delegates I/O operation to DMA module
- **Symmetric multiprocessor (SMP)**
  - ≥ 2 similar processors of comparable capability, connected by <u>bus</u>
  - Share the <u>same memory and I/O access</u>
  - All processors can perform the same functions (symmetric)

        - Controlled by an integrated OS that provides interaction between processors
        - Advantages of SMP:
            - Performance
            - Availability (redundancy against failures)
            - Incremental growth (adding more processors)
            - Scaling (vendors can offer a range of products)
        - Disadvantage:
            - Each processor has private cache – each cache invalidation has to happen in multiple places
- Multicore processor
        - Multiple processors on the same silicon chip

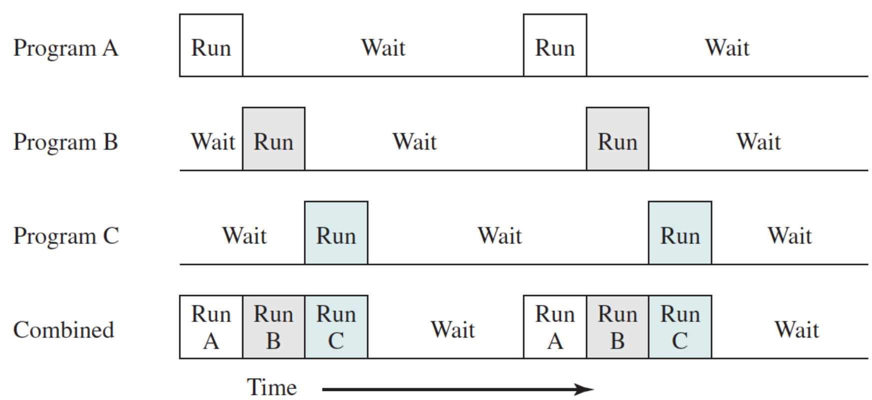
Chapter 2 (2.1 – 2.10)

- Operating system – a program that controls execution of application programs and acts as an standardized interface between applications and hardware
- **Objectives of an OS:**
  - Convenience (as a user/computer interface)
  - Efficiency (as a resource manager)
  - Ability to evolve
- An OS provides services for:
  - Program development
  - Program execution
  - Access to I/O
  - Control of file access
  - Control of system access
  - Error detection & response
  - Accounting & usage statistics
- Kernel – portion of OS that's in main memory
- The OS is a control mechanism that often gives control away for the processor to do "useful work", and then has control returned to it by the processor
- Key interfaces in a computer system:



- **Evolution of the OS**
  - Serial processing
    - Each job is run one at a time and one after another
    - Disadvantages:
      - Manual scheduling results in processing time wasted
      - Setup time associated with each job takes too long
  - Batch OS
    - Monitor – software that stays in main memory & controls the sequence of events

        - Jobs are batched together and executed; processor control is returned to monitor after every job is done
        - <u>Job control language</u> gives special instructions to the monitor
        - Hardware features:
            - Memory protection
            - Timer
            - Privileged instructions (<u>kernel mode</u> vs. <u>user mode</u>)
            - Interrupts
    - Uniprogramming vs. multiprogramming
        - Uni – process only one job at a given time
        - Multi – processor can run other jobs while waiting
            - Requires memory management that can handle multiple jobs in main memory

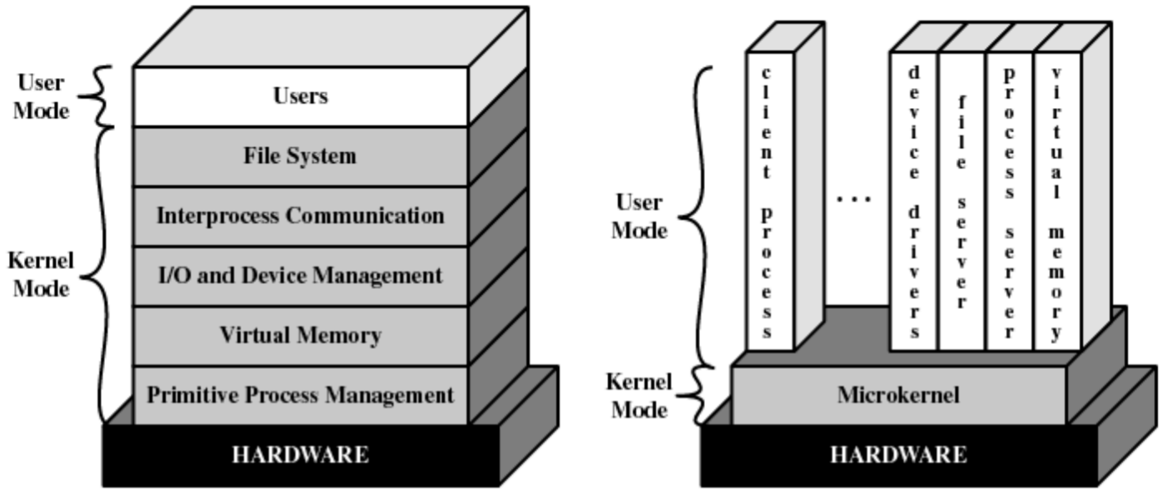


    - Time sharing
        - Share processor time among many simultaneous users
        - Time slicing – use system clock to interrupt and reassign processor control to different users

	Batch Multiprogramming	Time Sharing
Principal objective	Maximize processor use	Minimize response time
Source of directives to operating system	Job control language commands provided with the job	Commands entered at the terminal

- Major achievements:
    - <u>Process</u> (more in Chapter 3)
        - Problems:
            - Improper synchronization
            - Failed mutual exclusion
            - Non-determinate program operation
            - Deadlocks
    - <u>Memory management</u> (more in Chapter 7-8)
    - <u>Information protection & security</u> (more in Chapter 15)
    - <u>Scheduling & resource management</u> (more in Chapter 9-10)
- Monolithic kernel – provides most OS functionalities

- Microkernel – only a few essential functions are in kernel, other services provided by processes

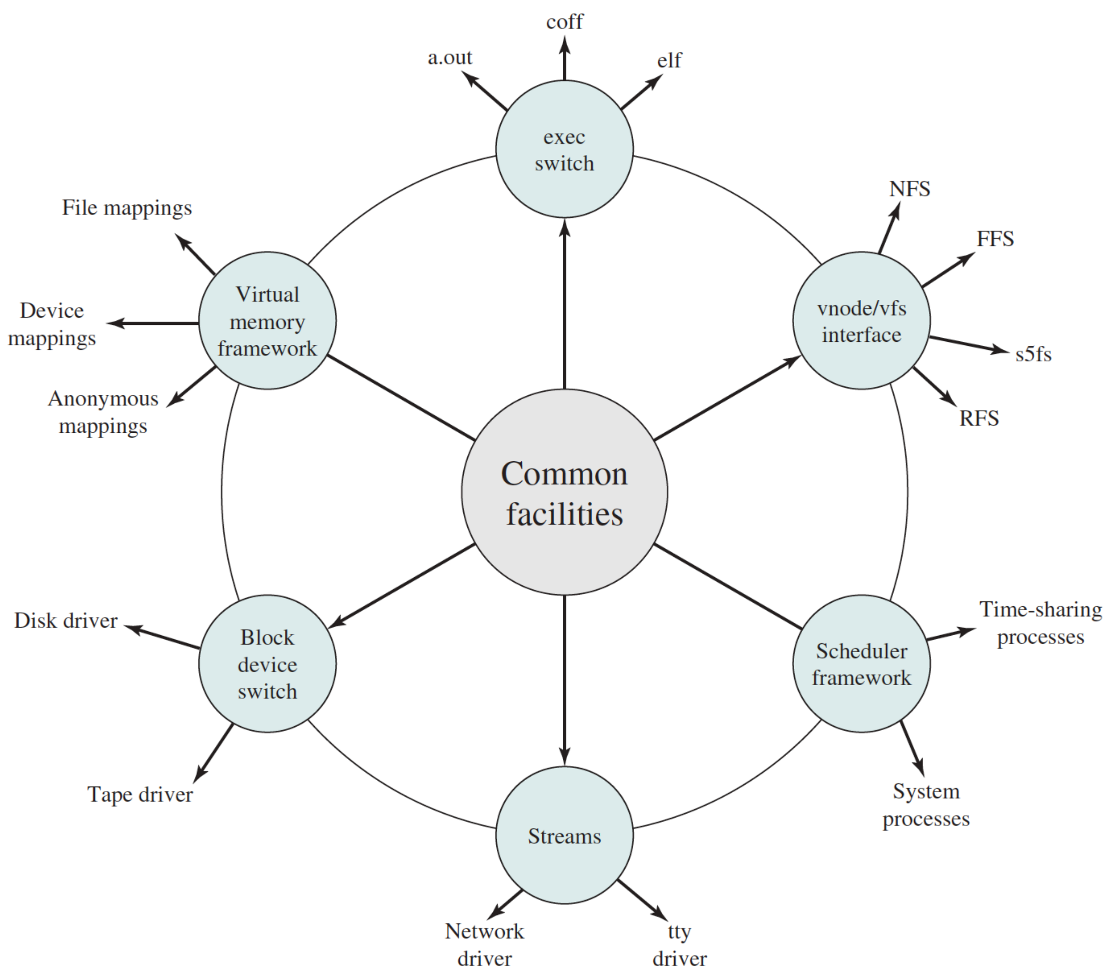


(a) Layered kernel (b) Microkernel

- **Fault tolerance**
  - Reliability = probability of correct operation up to time *t*
  - Mean time to failure (MTTF) = average uptime
  - Mean time to repair (MTTR) = average downtime
  - Types of faults:
    - Permanent, temporary (transient/intermittent)
  - Methods of redundancy:
    - Spatial/physical, temporal, information
- Design issues of multiprocessor OS:
  - Concurrency
  - Scheduling
  - Synchronization
  - Memory management
  - Reliability & fault tolerances
- **Windows architecture**
  - Kernel-mode components
    - Executive – core OS services; e.g. memory mgmt., process/thread mgmt., I/O etc.
    - Kernel – controls execution; e.g. thread scheduling, process switching etc.
    - Hardware abstraction layer
    - Device drivers
    - Windowing & graphics system
  - User-mode processes
    - Special system processes
    - Service processes
    - Environment subsystems
    - User applications
  - Windows services use the client/server model
- **Traditional UNIX architecture**

- Hardware → kernel → system call interface → UNIX commands & libraries

- **Modern UNIX architecture**



- **Linux architecture**
  - Kernel is structured as <u>loadable modules</u> (not microkernel, but modularized)
  - Dynamic linking of kernel modules (at runtime)
  - Stackable modules (modules can act as libraries or clients)
  - Kernel components:
    - Signals – kernel → process
    - System calls – process → kernel service
    - Processes & scheduler