Midterm

CS 40^ Study Guide

Ch 1 Computer System Overview

- 1.1 Basic Elements
- 1.2 Evolution of the Microprocessor
- 1.3 Instruction Execution
- 1.4 Interrupts
- 1.5 The Memory Hierarchy
- 1.6 Cache Memory
- 1.7 Direct Memory Access
- 1.8 Multiprocessor and Multicore Organization

Ch 2 Operating System Overview

- 2.1 Operating System Objectives and Functions
- 2.2 The Evolution of Operating Systems
- 2.3 Major Achievements
- 2.4 Developments Leading to Modern Operating Systems
- 2.5 Fault Tolerance
- 2.6 OS Design Considerations for Multiprocessor and Multicore
- 2.7 - 2.11 Optional

Ch 3 Process Description and Control

- 3.1 What is a process?
- 3.2 Process States
- 3.3 Process Description
- 3.4 Process Control
- 3.5 Evolution of the Operating System
- 3.6 UNIX SVR4 Process Management

Ch 9 Uniprocessor Scheduling

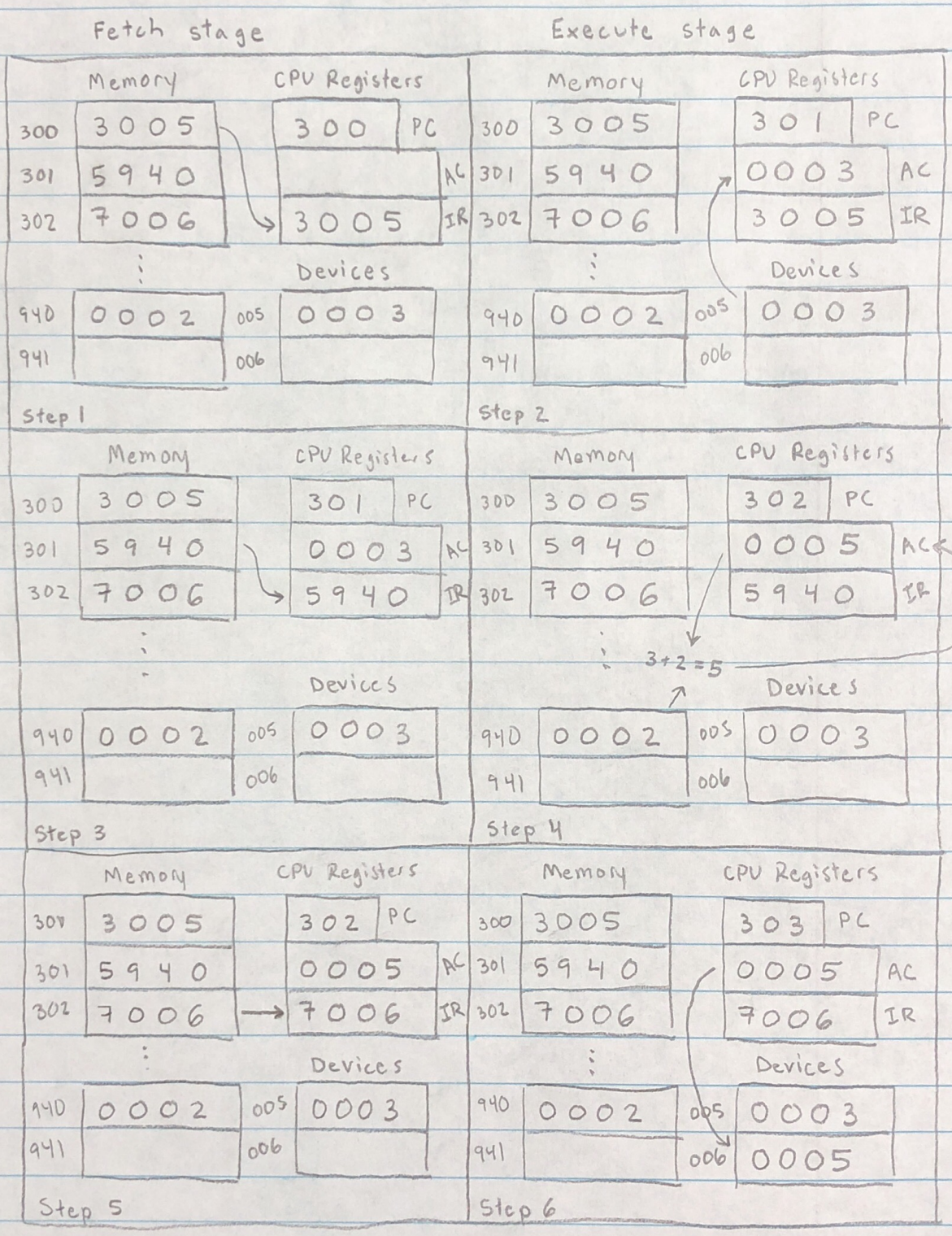
- 9.1 Types of Processor Scheduling
- 9.2 Scheduling Algorithms (through pg. 415)

Assignments

- 1.5a Machine Language Program Execution
  (Problem 1.1 / Fig 1.4)
- 1.5b Cache Memory cost/effective access time
  (Problem 1.12)
- 2.5 Binary and Hexadecimal Addition
- 3.5 Assemble/Link/Run an Assembly Language Program
  (3+2=5 like Fig 1.4)
- 9 Process Scheduling Exercise
  (Problem 9.2 / Table 9.5 / Figure 9.5)

Problem 1.1

Fig. 1.4 Example of Program Execution
(machine language program that computes 3 + 2 = 5)



Program counter (PC) = Address of instruction

Instruction register (IR) = Instruction being executed

Accumulator (AC) = Temporary storage

Instructions of a Hypothetical Machine:

0001 = Load AC from memory
0010 = Store AC to memory
0101 = Add to AC from memory

0011 = Load AC from I/O
0111 = Store AC to I/O

= Multiply AC by memory content
= Divide AC by memory content

↓

Note that contents of memory and registers are in hexadecimal

Fig. 1.5 Program Flow of Control Without and With Interrupts

Step		Units of time
1	-	6
2a	-	8
2b	-	12
3a	-	8
3b	-	10
4	-	4
5	-	4
I/O	-	8

Fig 1.5a - No Interrupts

6 + 4 + 8 + 4 + 20 + 4 + 8 + 4 + 18

10 + 12 + 24 + 12 + 18

22 + 36 + 18

58 + 18

76

Fig 1.5b - With Interrupts

6 + 4 + 8 + 4 + 12 + 4 + 8 + 4 + 10

10 + 12 + 16 + 12 + 10

22 + 28 + 10

50 + 10

60

↳ For Fig 1.5b you don't add a term for the I/O command b/c it executes concurrently with steps in the user program (specifically, 2a and 3a)

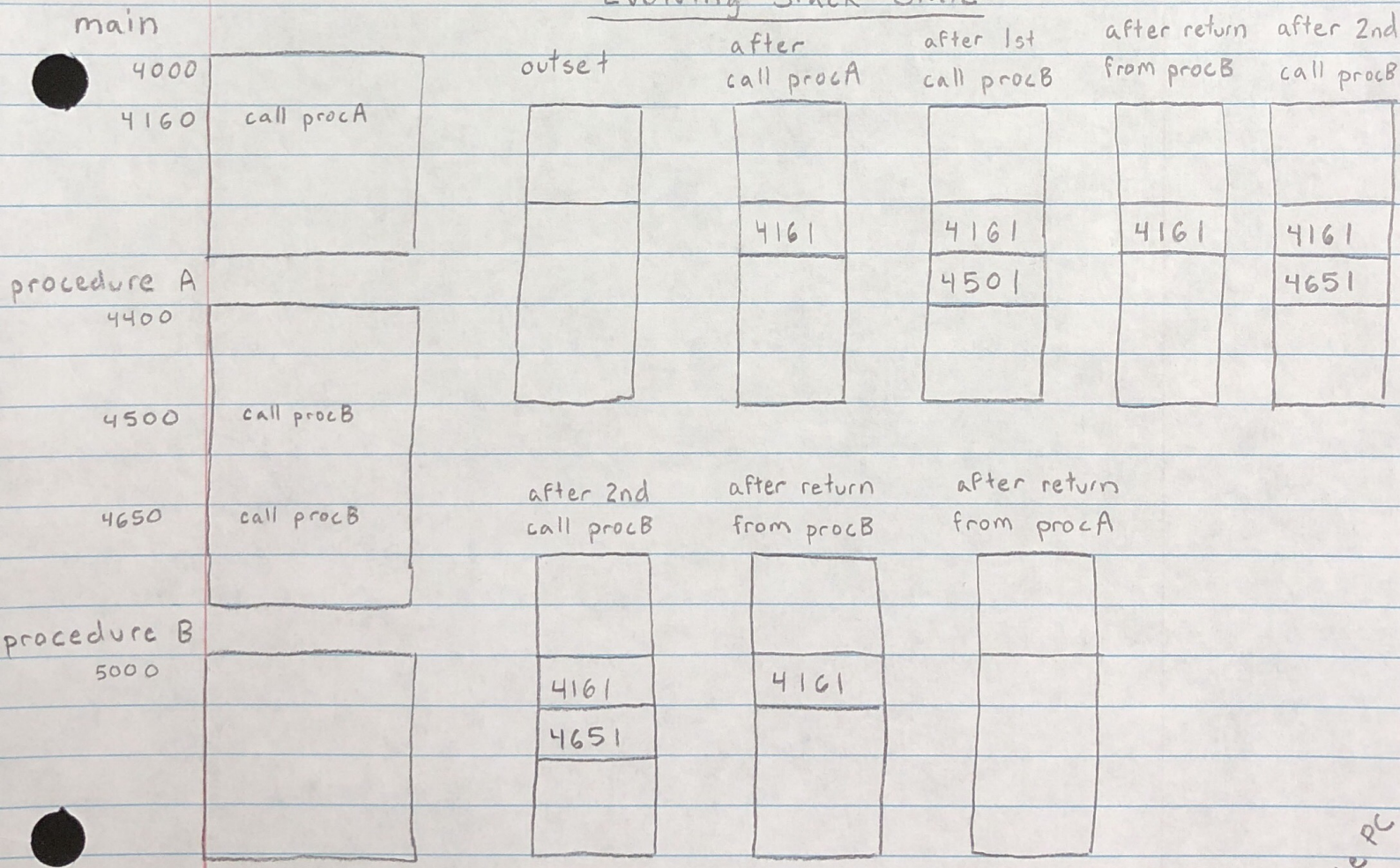
How a Stack works / Intra-program Flow of control
(Data structures slides)

Stack base register : bottom of stack

Stack pointer register: top of stack (incremented as elements are pushed on the stack)

push - stack node insertion
pop - stack node deletion
Use of Stack
- the OS uses a stack to keep track of return addresses when function calls occur

Evolving Stack State



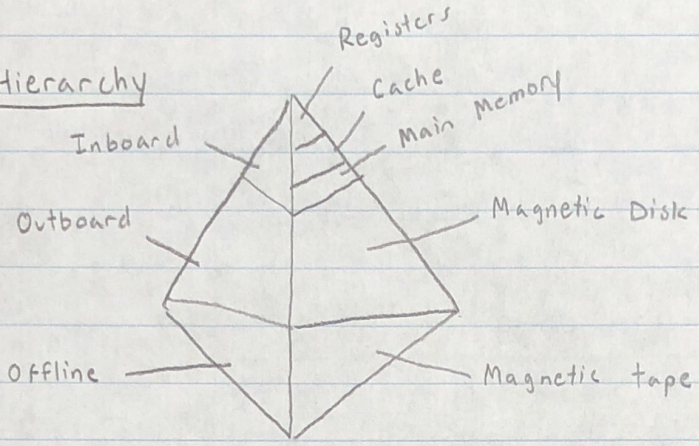
- when returning from a procedure, pop the stack and put that address in the PC
- when the instruction is a procedure call, push the address of the next instruction (current address + 1) onto the stack

Cache Memory

- Section 1.6

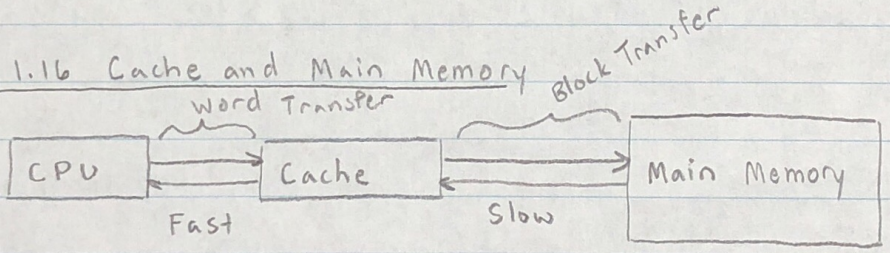
- caching: moving things into an intermediate memory that we think we will need

- Memory Hierarchy

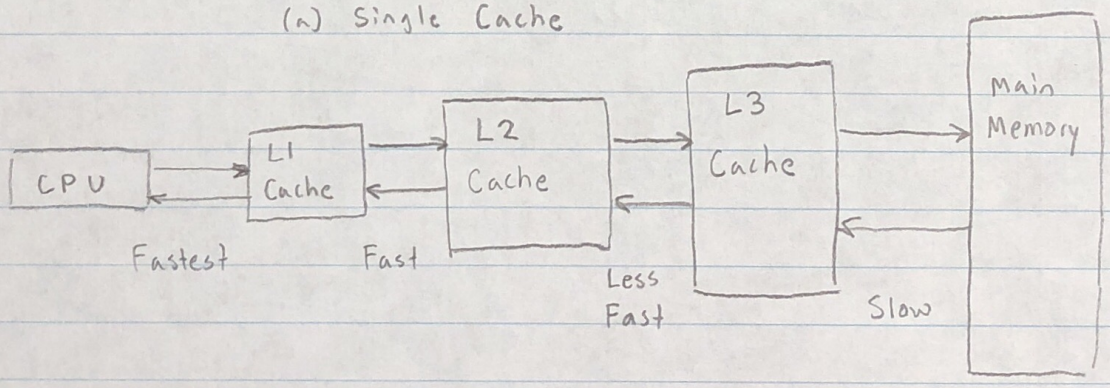


- Principal of Locality: during the execution of a program, memory references by the processor, for both instructions and data, tend to cluster
  - program counter → successive instructions
  - loops

- Fig 1.16 Cache and Main Memory



(a) Single Cache



(b) Three-level cache organization

- Caching writeup: Imagine you have two kinds of memory: fast and slow. Strategizing to predetermine where to-be-accessed data shall reside when needed, seeking to preposition it in the fastest memory available within the computer, is called caching

$$H = \frac{(T_{slow} - T_{ave})}{(T_{slow} - T_{fast})}$$

The percentage of time the data sought lies in the faster memory is called the "hit ratio"

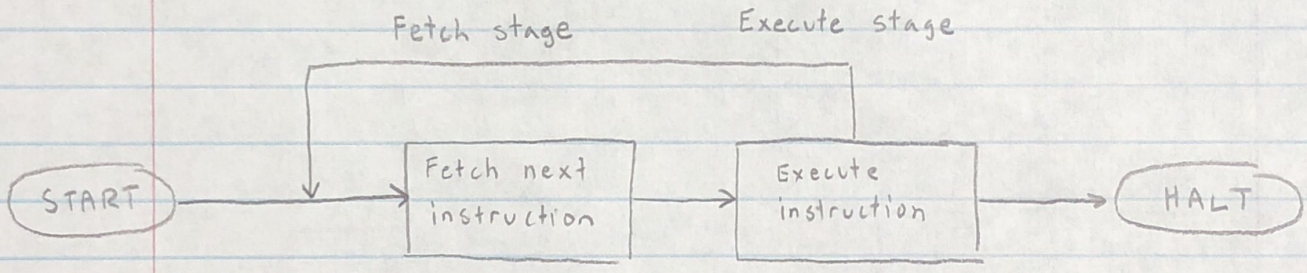
$$T_{ave} = (1 - H)\,T_{slow} + H\,T_{fast}$$

Registers and what they do

- Instruction Register (IR): fetched instruction is loaded into this register

- Program Counter (PC): holds the address of the next instruction to be fetched
  (eip/rip/ip)
  aka Instruction Pointer Register

- Fig 1.2 Basic Instruction Cycle



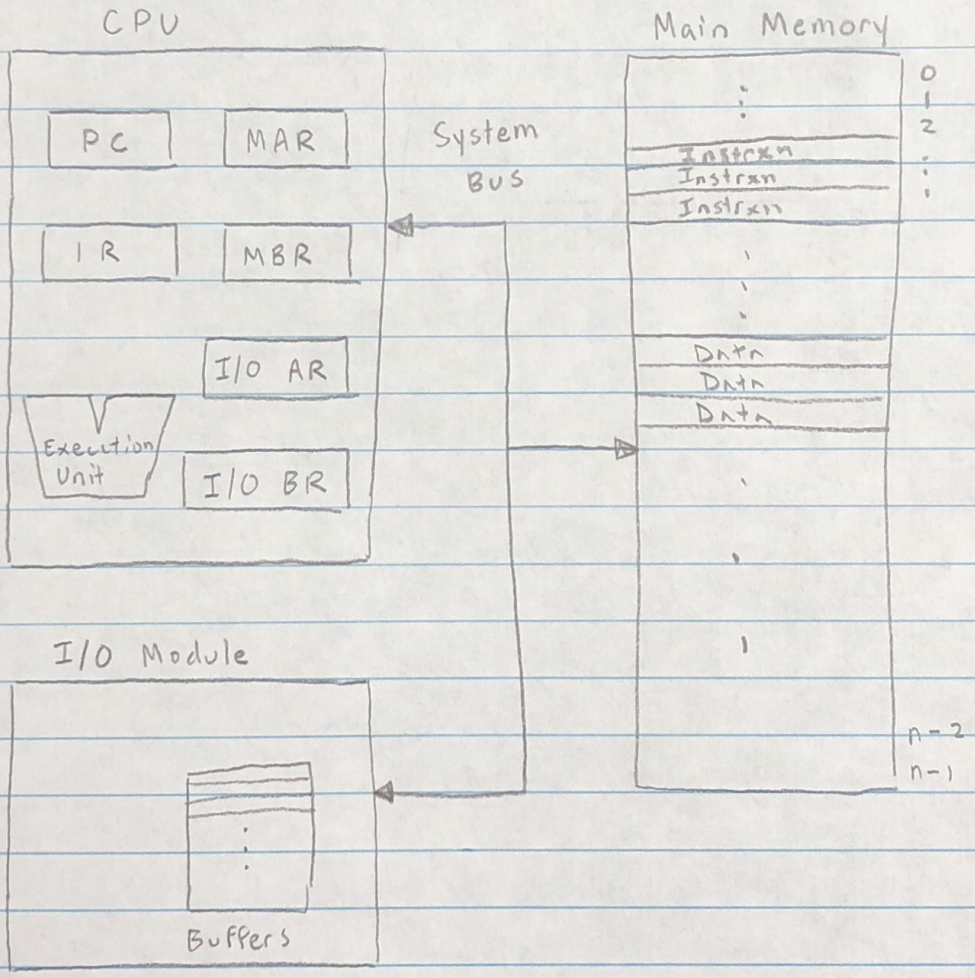
- Accumulator (AC): temporary storage, single data register

- EFLAGS Register: reflects outcomes; read, not write; status/control/system flags
  (Intel's version of Program Status Word (PSW))

Fig 1.1 Computer Components : Top-Level View

- composition of machine
- bus

- Fig 1.1 Computer Components : Top-Level View



System bus : provides for communication among processors, main memory, and I/O modules

Processor : controls the operation of the computer and performs its data processing functions

Main memory: stores data and programs; this memory is typically volatile
aka real/primary memory

I/O modules: move data between the computer and its external environment (disk, communications equipment, terminals)

What is an interrupt?

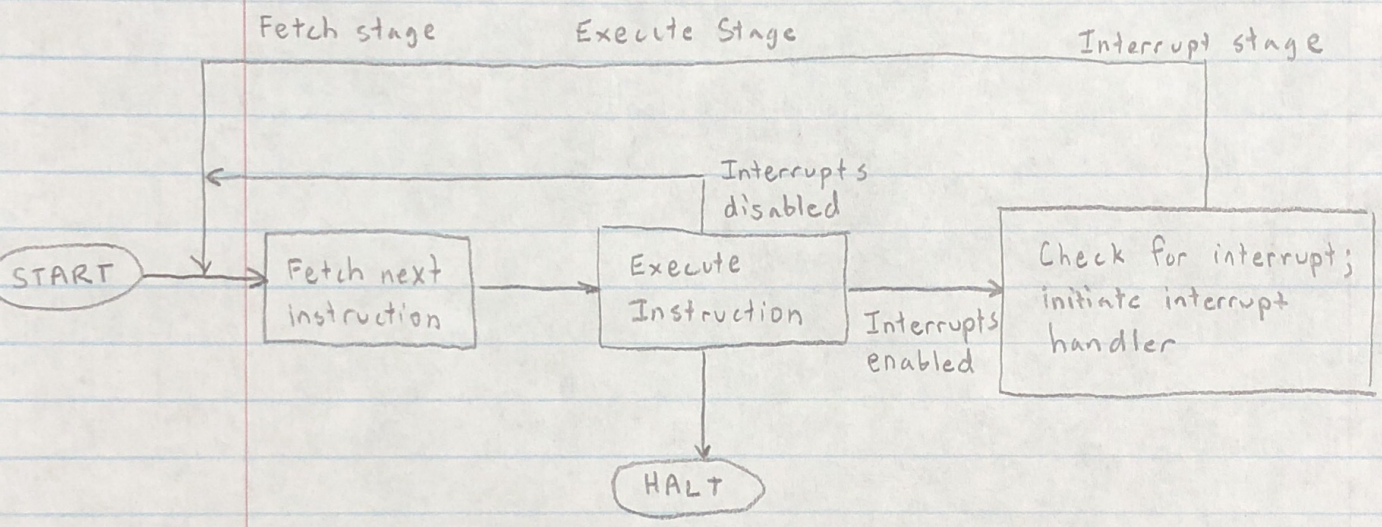
Interrupt: mechanism by which other modules may interrupt the normal sequencing of the processor; provided to improve processor utilization

Classes of Interrupts:
- Program
- Timer
- I/O
- Hardware failure

What is an interrupt handler

- interrupt handler routine : generally part of the OS; this routine determines the nature of the interrupt and performs whatever actions are needed

- Fig 1.7 Instruction Cycle with Interrupts



- In the interrupt stage, the processor checks to see if any interrupts have occurred
- If an interrupt is pending, the processor suspends execution of the current program and executes an interrupt handler routine

Distinguish between machine language vs. others

- In particular, machine language is the only language
- Other languages are just conveniences layered on top of machine language; they must be translated / reinterpreted to machine language when they are compiled

assembly language: mnemonics are 1-to-1 with the instruction set; need to run assembly code through an assembler and a linker in order to produce machine code that can run on the processor

high level languages (i.e. C): high level programs need to be run through a compiler to produce assembly code which is in turn translated to machine code that can run on the processor

Data Structures

- linked lists
- ★ queues
- stacks

Special linked list: a queue

- all insertions occur on one end (the "rear")
- all deletions at the other end (the "front")

Special linked list: a stack

- all insertions and deletions occur at one end, the "top"

Why does caching work?

Imagine you have 2 kinds of memory: fast and slow. The more frequently you can arrange to have wanted data waiting for you in the fast place, the faster your machine's overall or global average access to data will be. This works because of locality of reference.

Number Bases

BINARY:

1) 0101 + 1010 = (e) 1111

2) 0101 + 1011 = (b) 10000

3) 0111 + 0111 = (d) 1110

HEXADECIMAL:

4) 4817 + 3172 = (g) 7989

5) 4817 + 3173 = (e) 798A

6) 4817 + 3179 = (a) 7990

7) 4817 + 317F = (f) 7996

8) 4817 + B172 = (c) F989

9) 4817 + B972 = (d) 10189

10) ^{11}B 4 A^{10} C^{12} + ^{15}F D^{13} 8 6 = (b) 1B232

Why does deadlock occur?

deadlock:

It is possible for two or more programs to be hung up waiting for each other. For example, two programs may each require two I/O devices to perform some operation. One of the programs has seized control of one of the devices and the other program has control of the other device. Each is waiting for the other program to release the desired resource.

Other causes of errors:

- improper synchronization:

- failed mutual exclusion: more than one user attempts to make use of a shared resource at the same time
  (ex) integer n is shared between processes P and Q

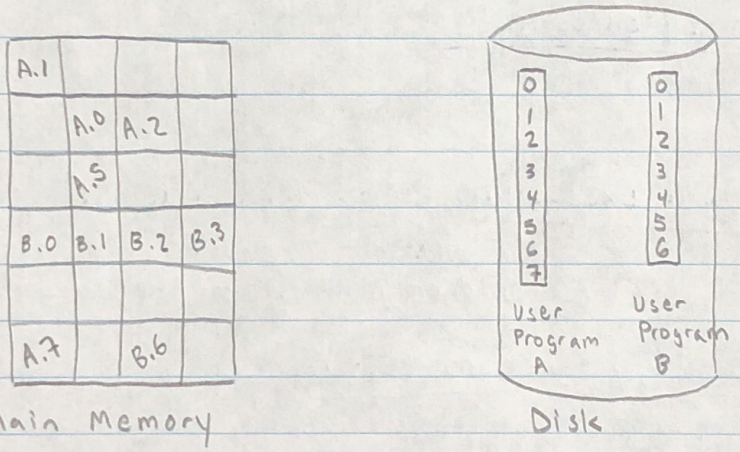
- nondeterminate program operation:

Virtual Memory Management

Virtual memory: virtual memory addresses are resolved to real addresses

Paging: pages are the unit loaded between disk and memory, not a full program
- virtual address consists of a page number and an offset within the page
- fixed size segments of memory = pages

Fig 2.9 Virtual Memory Concepts



Main Memory consists of a number of fixed-length frames, each equal to the size of a page. For a program to execute, some or all of its pages must be in main memory.

Secondary memory (disk) can hold many fixed-length pages. A user program consists of some number of pages. Pages for all programs plus the operating system are on disk, as are files.

In Class:

Evolution of Operating Systems:

- What were they driving at?
- What were they trying to accomplish in early OS's?

They were trying to maximize processor utilization b/c early computers were very expensive.

Serial Processing
- no OS – user interacted directly w/ hardware
- hard copy sign up sheet
- toggle switches
- two main problems: scheduling, setup time

↓

Simple Batch Systems
- resident monitor (simple OS)
- job control language (JCL)
- uniprogramming

↓

Multiprogrammed Batch Systems
- multiprogramming aka multitasking
- when one program needs to wait for I/O, the processor can switch to one of the other programs already in memory

↓

Time Sharing Systems
- can be used to handle multiple interactive jobs
- processor time is shared among multiple users
- multiple users simultaneously access the system through terminals, with the OS interleaving the execution of each user program in a short burst or quantum of computation

↳ no PC's in the 1960's when computers were big and costly

↳ so time sharing was developed to achieve an interactive computing facility

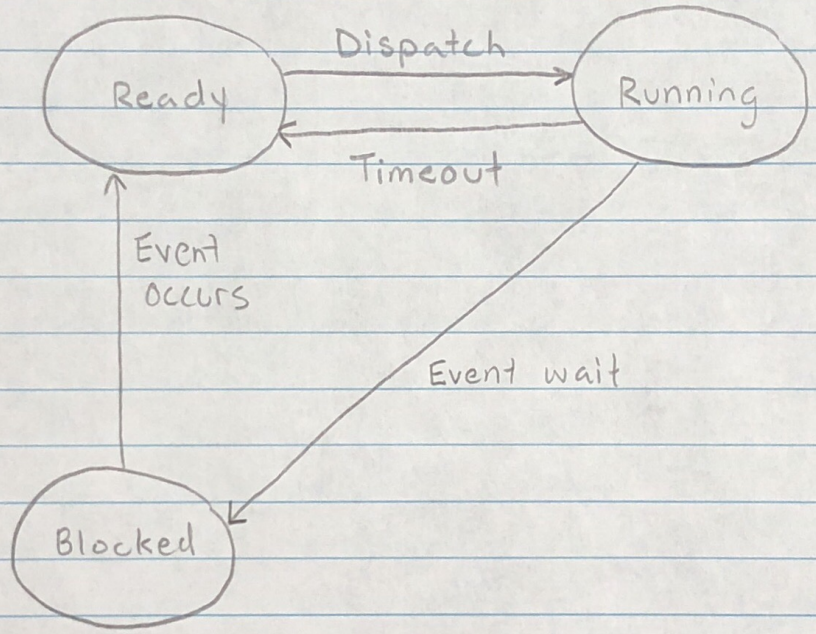
Linux Process Scheduler

- Why did Linux replace the process scheduling component in 2009?
  A: Needed to support real time processing?
  (covered day of exam)

In Class:

Three-State Process Diagram
(not the more complex five state one)
- six possible transitions
- a couple might never occur
- for those that do occur, when and why does the task make that transition?

Fig 3.6 Three-State Process Model



Ready → Running (Dispatch): when it is time to select a process to run, the OS chooses one of the processes in the Ready state. This is the job of the scheduler or dispatcher.
Running → Ready (Timeout): (1) the running process has reached the maximum allowable time for uninterrupted execution (2) the process has been preempted by another w/ higher priority (3) a process voluntarily releases control of the processor
Running → Blocked (Event wait): a process requests something for which it must wait, such as a system service call or I/O operation
Blocked → Ready (Event occurs): a process in the Blocked state is moved to the Ready state when the event for which it has been waiting occurs

Know what System Calls are

Ordinary user processes are offered a variety of services by operating systems. The programs go through the operating system to access disks and other devices, open files, and other system operations. All the program has to do is ask. That's called making a system call, i.e. to the "file open" system service.

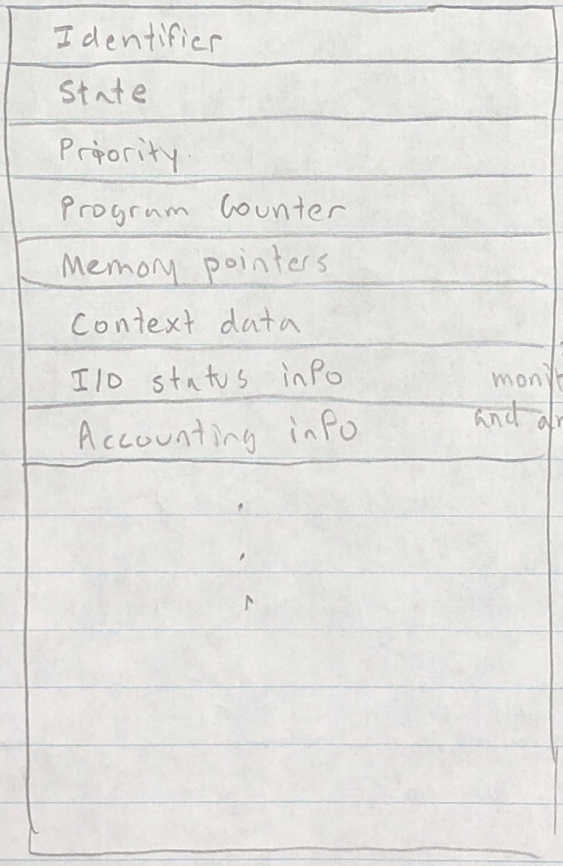
Process control block aka Process descriptor

- T/F question

Process Control Block — data structure

- contains the process elements
- it is possible to interrupt a running process and later resume execution as if the interruption had not occurred
- created and managed by the OS (data structure)
- key tool that allows support for multiple processes

Fig 3.1 Simplified Process Control Block



Process Descriptor's Role

- the process descriptor is the most important data structure in the OS
- each one contains all of the info about a process needed by the OS
- the blocks are read and/or modified by virtually every module of the OS (scheduling, resource allocation, interrupt processing, and performance monitoring and analysis)

Process Descriptor

- identifiers
- state
- resources

Process Identification
Processor State Information
Process Control Information
} Process Control Block

Timesharing vs. Batch processing

- read what the book says about this

Time-Sharing Systems

- can be used to handle multiple interactive jobs
- processor time is shared among multiple users
- multiple users simultaneously access the system through terminals, with the OS interleaving the execution of each user program in a short burst or quantum of computation

Simple Batch systems

- monitor
  - user no longer has direct access to the processor
  - job is submitted to the computer operator who batches them together and places them on an input device
  - program branches to the monitor when finished
- monitor point of view
  - monitor controls the sequence of events
  - resident monitor is software always in memory
  - monitor reads in job and gives control
  - job returns control to monitor
  - Job Control Language (JCL)

Table 2.5 Batch Multiprogramming versus Time Sharing

	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

Process Scheduling Problem like HW

- given 3 processes w/ different processing times
- which scenarios occur in which order?
- know the different scheduling policies and which are preemptive vs. nonpreemptive

Preemptive: every time slice is occasion for a new decision of which process will run next

RR1 (round robin, quantum = 1):

- uses preemption based on a clock
- aka time slicing

- fair treatment

SRT (shortest remaining time):

- preemptive version of SPN
- scheduler always chooses the process that has the shortest expected remaining processing time

- penalizes long processes

Nonpreemptive:
- once any process starts, you let it run to conclusion
- choosing which process will run next is not performed every time slice
- only on those slices right after the running process has finished

FCFS (first come first served):

- aka FIFO
- when the current process finishes executing, the process that has been in the Ready queue the longest is selected
- nonpreemptive

- penalizes short processes
- penalizes I/O bound processes

SPN (shortest process next):

- nonpreemptive policy in which the process w/ the shortest expected processing time is selected next

- penalizes long processes

Maintaining the queue:

On any time slice that has both a bumped-but-unfinished and a brand new process the queue will consist of

1. whatever processes were in the queue already
2. the brand new process
3. the one that just got bumped

Linux Process Scheduler

- and its priorities (covered day of exam)

In Class: