High Performance Computing
Lecture 21

Matthew Jacob

Indian Institute of Science

Semaphore Examples

- Semaphores can do more than mutex locks
- Example: Consider our concurrent program where process P1 reads 2 matrices; process P2 multiplies them & process P3 outputs the product
  - Semaphores $S_1 = 0$ $S_2 = 0$

Process P1	Process P2	Process P3
	$P(S_1)$	$P(S_2)$
Read A[ ], B[ ]	C[ ] = A[ ] * B[ ]	Write C[ ]
$V(S_1)$	$V(S_2)$	

Deadlock

Consider the following process:

P1: AcquireLock (L); AcquireLock(L);

- Suppose that the first AcquireLock(L) succeeds
- P1 is then waiting for something (release of lock that it is holding) that will never happen
- This is a simple example of a general problem called deadlock
- Caused by a cycle of processes waiting for resources held by others while holding resources needed by others

Classical Problems

Producers-Consumers Problem

- Bounded buffer problem
- Producer process makes things and puts them into a fixed size shared buffer
- Consumer process takes things out of shared buffer and uses them
- Must ensure that producer doesn't put into full buffer or consumer take out of empty buffer
- While treating buffer accesses as critical section

Producers-Consumers Problem

shared Buffer[0 .. N-1]

Producer: repeatedly

Produce x; if (buffer is full) wait for consumption

Buffer[i++] = x; signal consumer

Consumer: repeatedly

If (buffer is empty) wait for production

y = Buffer[- - i]

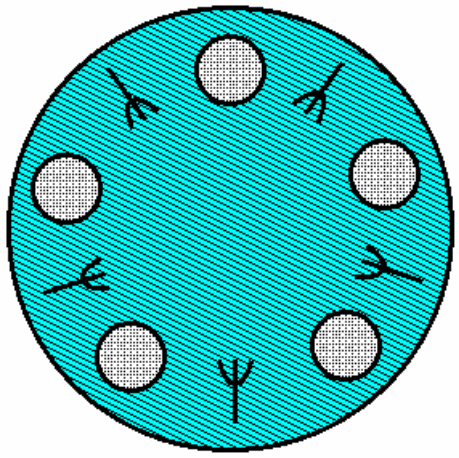
Consume y; signal producer

Dining Philosophers Problem

- N philosophers sitting around a circular table with a plate of food in front of each and a fork between each 2 philosophers
- Philosopher does: repeatedly

  Eat (using 2 forks)

  Think
- Problem: Avoid deadock; be fair



THREADS

Thread

- Thread of control in a process
- `Light weight process'

- Weight related to
  - Time for creation
  - Time for context switch
  - Size of context
- Recall: Process as a Data Structure

Process as a Data Structure.

PROCESS CONTEXT

- What is the data manipulated by these process operations?
  1. Text, Data, Stack, Heap
  2. Data stored in hardware
  3. Other information maintained by the OS
     - Process, parent and user identifiers
     - Memory management information: Page table
     - CPU time used by the process, in user/system
     - File related info: Open files, file pointers

Threads and Processes

- Thread context
  - Thread id
  - Stack
  - Stack pointer, PC, GPR values
- So, thread context switching can be much faster than process context switch
- Many threads in the same process share parts of that process context
  - Virtual address space (other than stack)
- So, threads in the same process share variables that are not stack allocated

Thread Implementation

- Could either be supported in the operating system or by a library
- Pthreads: POSIX thread library
  - int pthread_create
    - pthread_t *thread, const pthread_attr_t *attr, void *(*start_routine), void *arg
  - pthread_attr
  - pthread_join
  - pthread_exit
  - pthread_detach

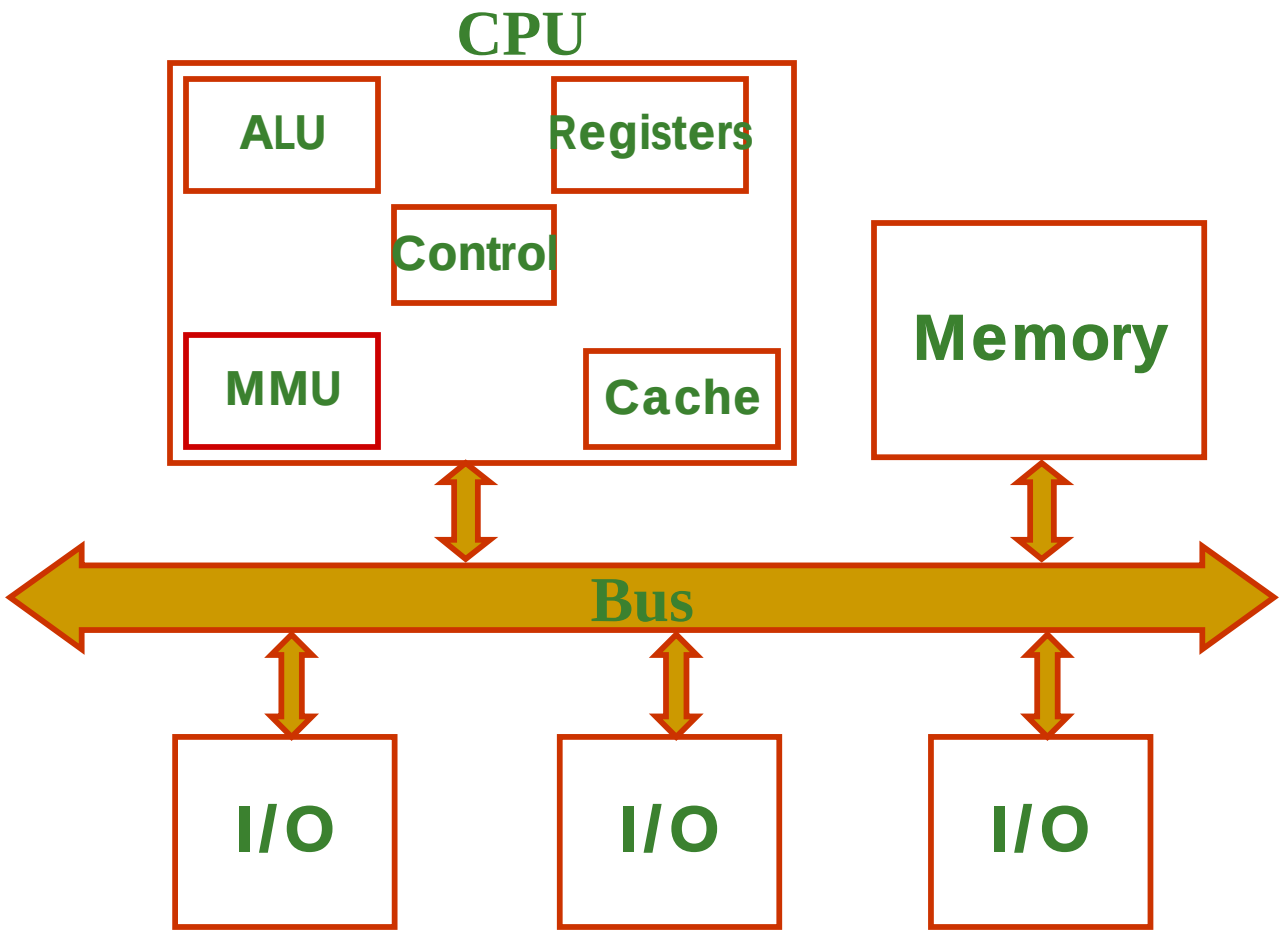
Synchronization Primitives

- Mutex locks
  - int pthread_mutex_lock(pthread_mutex_t *mutex)
    - If the mutex is already locked, the calling thread blocks until the mutex becomes available. Returns with the mutex object referenced by mutex in the locked state with the calling thread as its owner.
  - pthread_mutex_unlock
- Semaphores
  - sem_init
  - sem_wait
  - sem_post

Agenda

1. Program execution: Compilation, Object files, Function call and return, Address space, Data & its representation (4)
2. Computer organization: Memory, Registers, Instruction set architecture, Instruction processing (6)
3. Virtual memory: Address translation, Paging (4)
4. Operating system: Processes, System calls, Process management (6)
5. Pipelined processors: Structural, data and control hazards, impact on programming (4)
6. Cache memory: Organization, impact on programming (5)
7. Program profiling (2)
8. File systems: Disk management, Name management, Protection (4)
9. Parallel programming: Inter-process communication, Synchronization, Mutual exclusion, Parallel architecture, Programming with message passing using MPI (5)

Basic Computer Organization

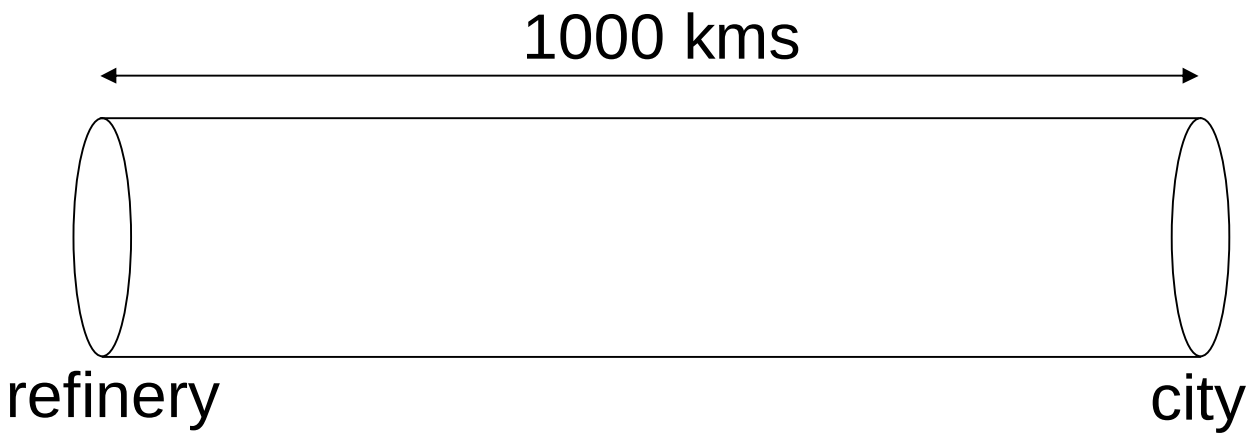


Performance of Processor

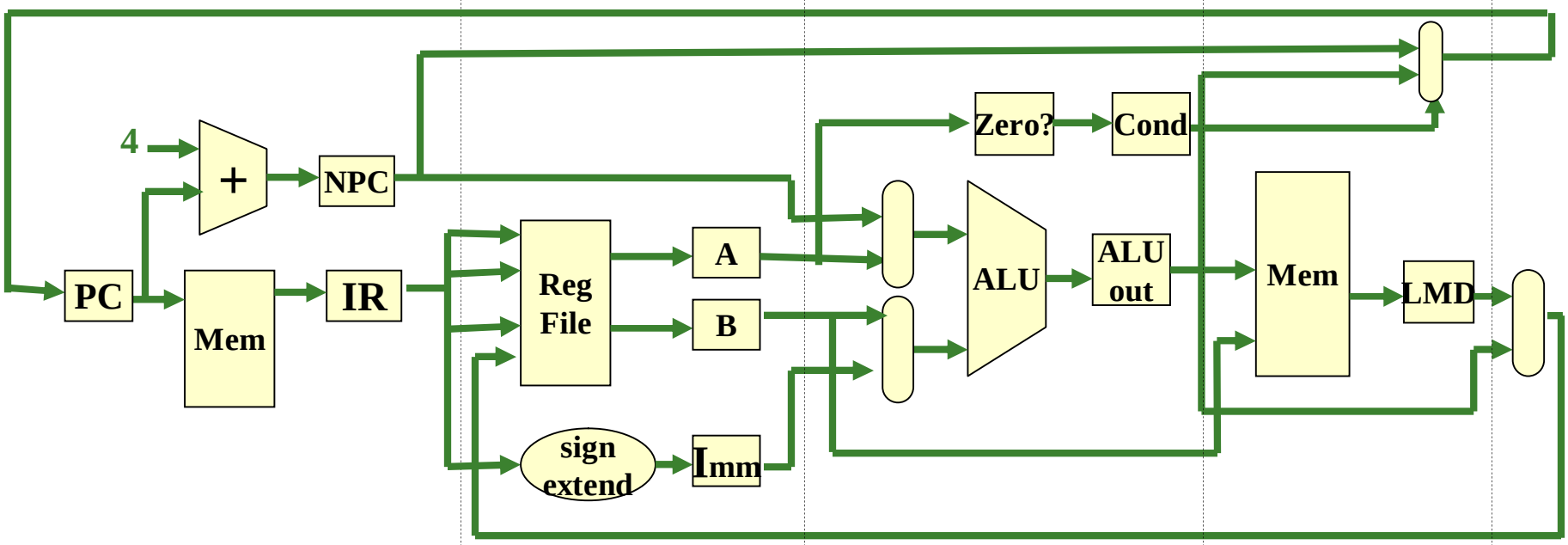
- Which is more important?
  - execution time of a single instruction
  - throughput of instruction execution
    i.e., number of instructions executed per unit time
- Cycles Per Instruction (CPI)
- Current ideas: CPI between 3 and 5
- Pipelining
  - Why keep Fetch hardware idle while instruction is being decoded
  - Inspired by petroleum pipelines?

Pipelines

- Used for transportation of liquids or gases over long distances
  - 1000s of kms
  - Built with periodic pump/compressor stations to keep the fluid flowing

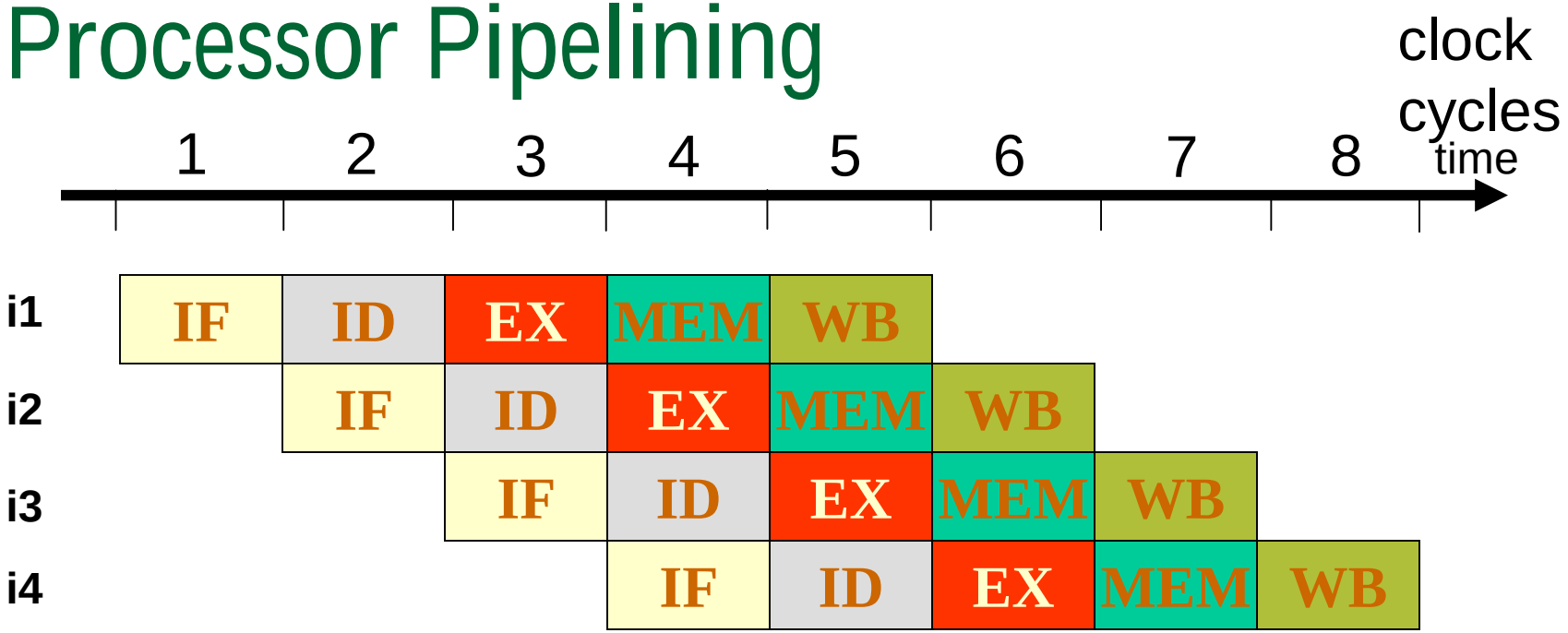


Inside the Processor



Inst Fetch IF | **Inst Decode ID** | **Execution EX** | **Memory MEM** | **WB**

Processor Pipelining



- Execution time of each instruction is still 5 cycles, but the throughput is now 1 instruction per cycle
- Initial pipeline fill time (4 cycles), after which 1 instruction completes every cycle