# Computer organization

Jeremy Iverson

College of Saint Benedict & Saint John's University

## plan for the day

- computer organization
  - central processing unit
  - main memory
  - cache memory
- case study: matrix multiplication
- using Git
- `printf("hello, world\n");`

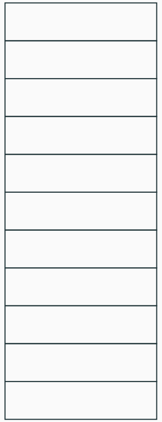
## central processing unit (CPU)

activity with Pep/9 ISA

## what does memory look like?

have students draw a picture of “memory”

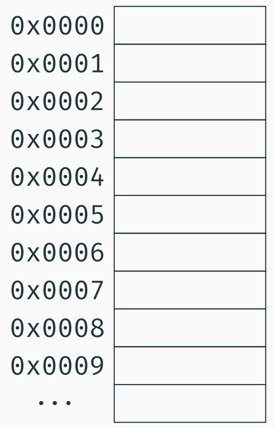
## memory organization



- memory is organized sequentially

- a process is an instance of a computer program
- a multitasking operating system is capable of running many processes despite having only one or a “small” number of physical cores
  - context switches the processes at predefined time slices
  - `top` activity

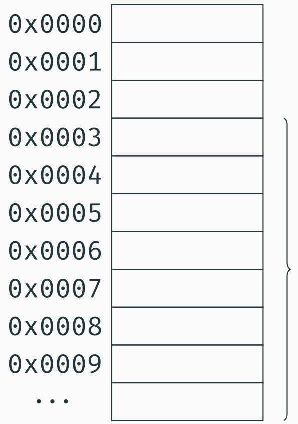
## memory organization



- memory is organized sequentially
- memory is addressed starting from the *top*
    - granularity is a byte

- a process is an instance of a computer program
- a multitasking operating system is capable of running many processes despite having only one or a “small” number of physical cores
    - context switches the processes at predefined time slices
    - `top` activity

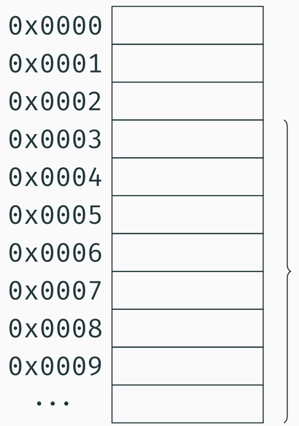
## memory organization



- memory is organized sequentially
- memory is addressed starting from the *top*
    - granularity is a byte
- a process and its data can be stored anywhere in memory

- a process is an instance of a computer program
- a multitasking operating system is capable of running many processes despite having only one or a “small” number of physical cores
    - context switches the processes at predefined time slices
    - `top` activity

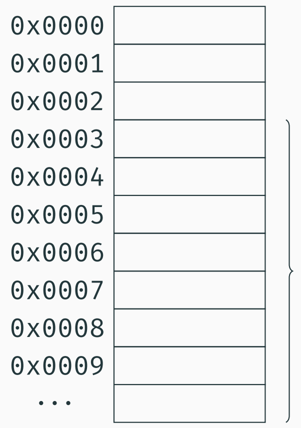
## memory organization



- memory is organized sequentially
- memory is addressed starting from the *top*
  - granularity is a byte
- a process and its data can be stored anywhere in memory
  - physical vs. logical addressing

- a process is an instance of a computer program
- a multitasking operating system is capable of running many processes despite having only one or a “small” number of physical cores
  - context switches the processes at predefined time slices
  - `top` activity

## memory organization



- memory is organized sequentially
- memory is addressed starting from the *top*
    - granularity is a byte
- a process and its data can be stored anywhere in memory
    - physical vs. logical addressing
    - each process has its own *address space*

- a process is an instance of a computer program
- a multitasking operating system is capable of running many processes despite having only one or a “small” number of physical cores
    - context switches the processes at predefined time slices
    - `top` activity

## cache memory

- what is it?

- `lscpu`
- `cat /proc/cpuinfo` activity for CPU info
- `cat /sys/devices/system/cpu/cpu0/cache/` for cache info
- L1, L2, L3
  - L1 is usually split into data and instruction
  - L2 is usually not shared, i.e., each core has its own
  - L3 is usually shared by multiple cores

## cache memory

- a high-speed memory co-located with the CPU on the chip

- `lscpu`
- `cat /proc/cpuinfo` activity for CPU info
- `cat /sys/devices/system/cpu/cpu0/cache/` for cache info
- L1, L2, L3
  - L1 is usually split into data and instruction
  - L2 is usually not shared, i.e., each core has its own
  - L3 is usually shared by multiple cores

## cache memory

- a high-speed memory co-located with the CPU on the chip
- what problem is cache addressing?

- `lscpu`
- `cat /proc/cpuinfo` activity for CPU info
- `cat /sys/devices/system/cpu/cpu0/cache/` for cache info
- L1, L2, L3
  - L1 is usually split into data and instruction
  - L2 is usually not shared, i.e., each core has its own
  - L3 is usually shared by multiple cores

## cache memory

- a high-speed memory co-located with the CPU on the chip
- CPU can compute faster than memory can be fetched
  - many problems reuse data or use data that is stored near each other in memory, known as *temporal* and *spatial* locality, respectively

- `lscpu`
- `cat /proc/cpuinfo` activity for CPU info
- `cat /sys/devices/system/cpu/cpu0/cache/` for cache info
- L1, L2, L3
  - L1 is usually split into data and instruction
  - L2 is usually not shared, i.e., each core has its own
  - L3 is usually shared by multiple cores

## cache memory

- a high-speed memory co-located with the CPU on the chip
- CPU can compute faster than memory can be fetched
    - many problems reuse data or use data that is stored near each other in memory, known as *temporal* and *spatial* locality, respectively
- how does it work?

- `lscpu`
- `cat /proc/cpuinfo` activity for CPU info
- `cat /sys/devices/system/cpu/cpu0/cache/` for cache info
- L1, L2, L3
    - L1 is usually split into data and instruction
    - L2 is usually not shared, i.e., each core has its own
    - L3 is usually shared by multiple cores

## cache memory

- a high-speed memory co-located with the CPU on the chip
- CPU can compute faster than memory can be fetched
    - many problems reuse data or use data that is stored near each other in memory, known as *temporal* and *spatial* locality, respectively
- when memory load is issued, cache is checked to see if the data exists there. if not, it is loaded from main memory and stored in cache. when cache becomes full, old data is evicted to make room for new.

- `lscpu`
- `cat /proc/cpuinfo` activity for CPU info
- `cat /sys/devices/system/cpu/cpu0/cache/` for cache info
- L1, L2, L3
    - L1 is usually split into data and instruction
    - L2 is usually not shared, i.e., each core has its own
    - L3 is usually shared by multiple cores

## cache memory

- a high-speed memory co-located with the CPU on the chip
- CPU can compute faster than memory can be fetched
    - many problems reuse data or use data that is stored near each other in memory, known as *temporal* and *spatial* locality, respectively
- when memory load is issued, cache is checked to see if the data exists there. if not, it is loaded from main memory and stored in cache. when cache becomes full, old data is evicted to make room for new.
- how much is there?

- `lscpu`
- `cat /proc/cpuinfo` activity for CPU info
- `cat /sys/devices/system/cpu/cpu0/cache/` for cache info
- L1, L2, L3
    - L1 is usually split into data and instruction
    - L2 is usually not shared, i.e., each core has its own
    - L3 is usually shared by multiple cores

## cache memory

- a high-speed memory co-located with the CPU on the chip
- CPU can compute faster than memory can be fetched
    - many problems reuse data or use data that is stored near each other in memory, known as *temporal* and *spatial* locality, respectively
- when memory load is issued, cache is checked to see if the data exists there. if not, it is loaded from main memory and stored in cache. when cache becomes full, old data is evicted to make room for new.
- multiple levels of cache, each larger than the previous, but all much smaller than main memory

- `lscpu`
- `cat /proc/cpuinfo` activity for CPU info
- `cat /sys/devices/system/cpu/cpu0/cache/` for cache info
- L1, L2, L3
    - L1 is usually split into data and instruction
    - L2 is usually not shared, i.e., each core has its own
    - L3 is usually shared by multiple cores

## cache performance

From Intel Performance Analysis Guide:

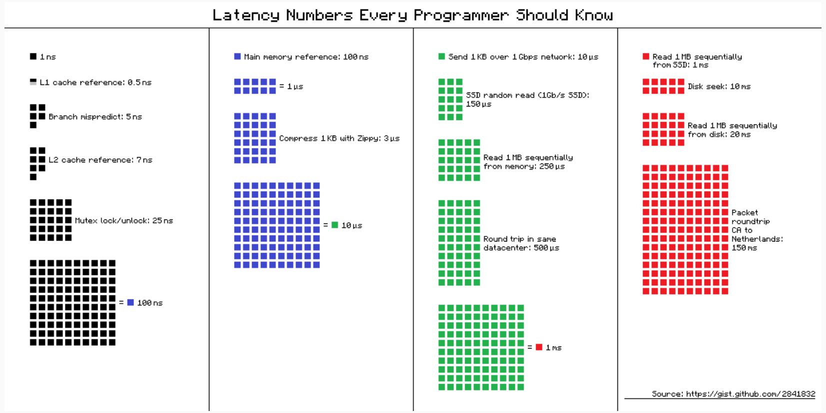
```
Core i7 Xeon 5500 Series Data Source Latency (approximate)              [Pg. 22]

local  L1 CACHE hit,                              ~4 cycles (   2.1 -  1.2 ns )
local  L2 CACHE hit,                             ~10 cycles (   5.3 -  3.0 ns )
local  L3 CACHE hit, line unshared               ~40 cycles (  21.4 - 12.0 ns )
local  L3 CACHE hit, shared line in another core ~65 cycles (  34.8 - 19.5 ns )
local  L3 CACHE hit, modified in another core    ~75 cycles (  40.2 - 22.5 ns )

remote L3 CACHE (Ref: Fig.1 [Pg. 5])        ~100-300 cycles ( 160.7 - 30.0 ns )

local  DRAM                                                  ~60 ns
remote DRAM                                                 ~100 ns
```

## more about latency



Latency Numbers Every Programmer Should Know