Memory Systems

The Big Picture

- Operations that can be performed by combinational circuits alone is limited
- Memory required to
  - Store large amounts of input data
  - Use results of previous computations
  - Persistent storage
  - Store instructions
- A flip-flop (register) is the simplest example of a memory element

Basic Specification

- K by N, K words of N bits each
- $M = \log_2 K$ address input signals
- So, 4096×8 memory has 32,768 bits of memory, 12 address input signals, 8 input/output data signals
- K is a power of 2, N is a multiple of 8

Memory Capacity

- What is the capacity (in bytes) of a memory with K words and N bits per word?
- The SRAM chip that you will use in Lab 4 has 16-bit word length and 256K words
- What is the capacity (in bytes)?
- How many bits are needed to address the memory?

Memory Capacity

- For a 32-bit processor, how many addresses are possible?
- For a 64-bit processor?

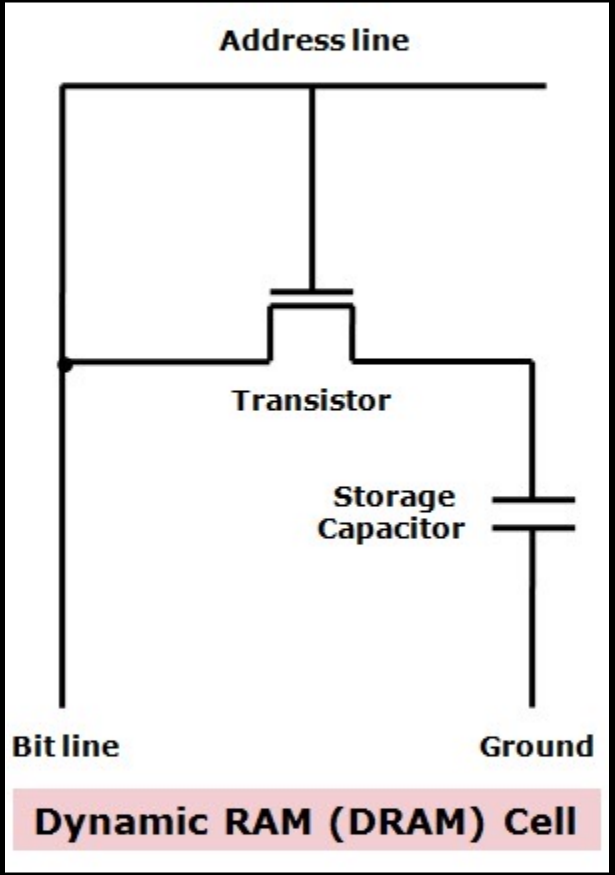
Storage permanence - ROM

- ROM: can be read from, but not written to, maintains bits “forever”
- EEPROM, Flash memory are essentially versions of ROM

Storage permanence - DRAM

Dynamic RAM (DRAM)

- Single transistor and a capacitor to store bit (transistor on – charge capacitor)
- Compact and cheap
- “refresh” required due to capacitor leak
- Refresh rate on the order of microseconds
- Slow access time (on the order of 100 ns)



Storage permanence - SRAM

Static RAM (SRAM)

- Memory cell uses flip-flop to store bit
- Requires six transistors to store bit
- Holds data as long as power supplied
- Fast access time (on the order of 10 ns)

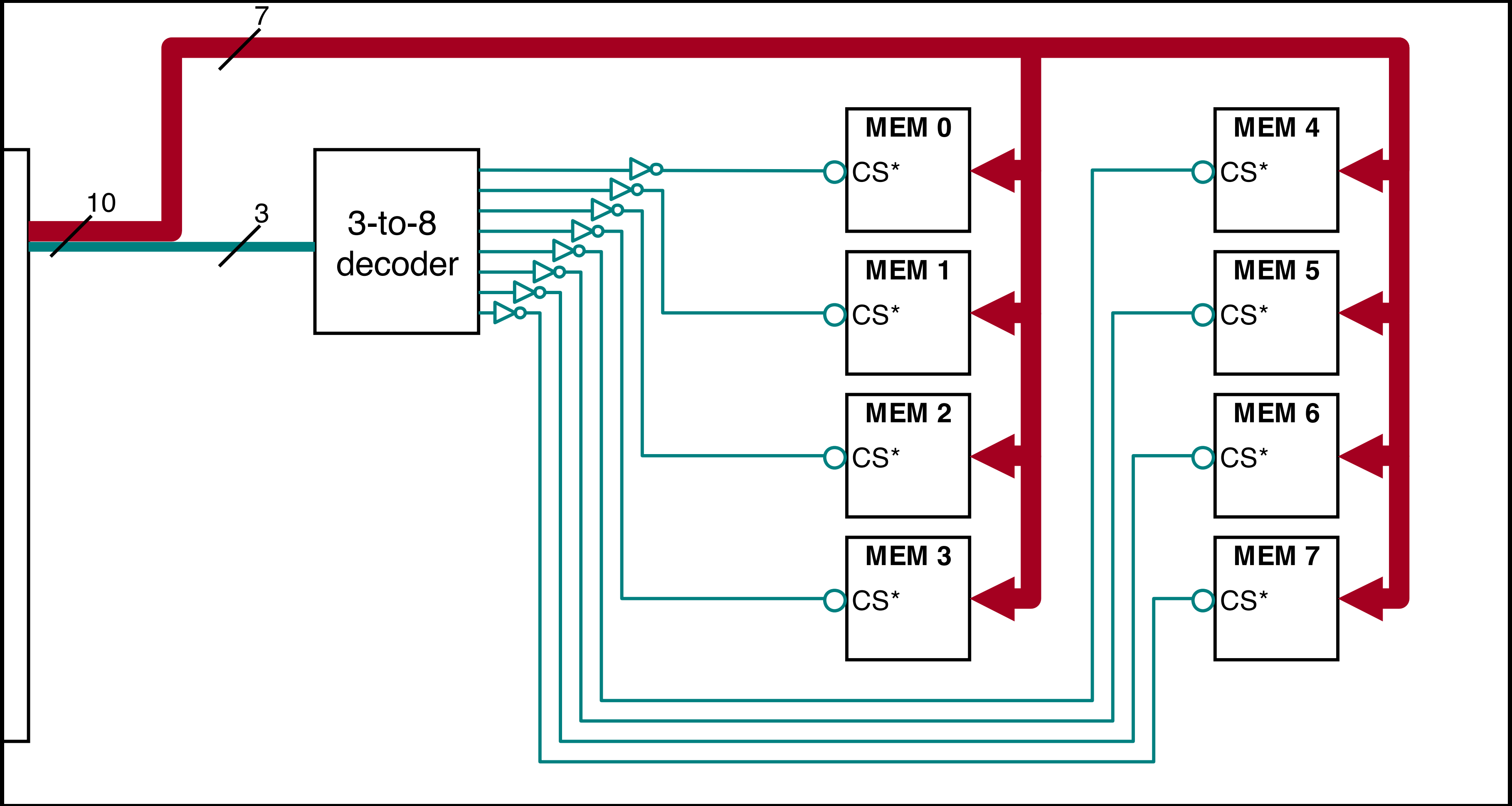
Bits and Bytes - Reminder

- Anything with bit is a power of 10, so a Kbit is 10^3 bits
- Anything with byte is a power of 2, so a Kbyte is $2^{10} = 1024$ bytes

Memory Hierarchies

- For economic reasons, may want to build up a larger memory from smaller units
- Straightforward to do, for example suppose that we have 10 address lines (1 KB of memory)
- Suppose that we use 128 by 8 memory chips
- Need 8 such chips
- 3 of the address lines choose the chip, the remaining 7 choose the address on the chip (duplicate addresses between chips)

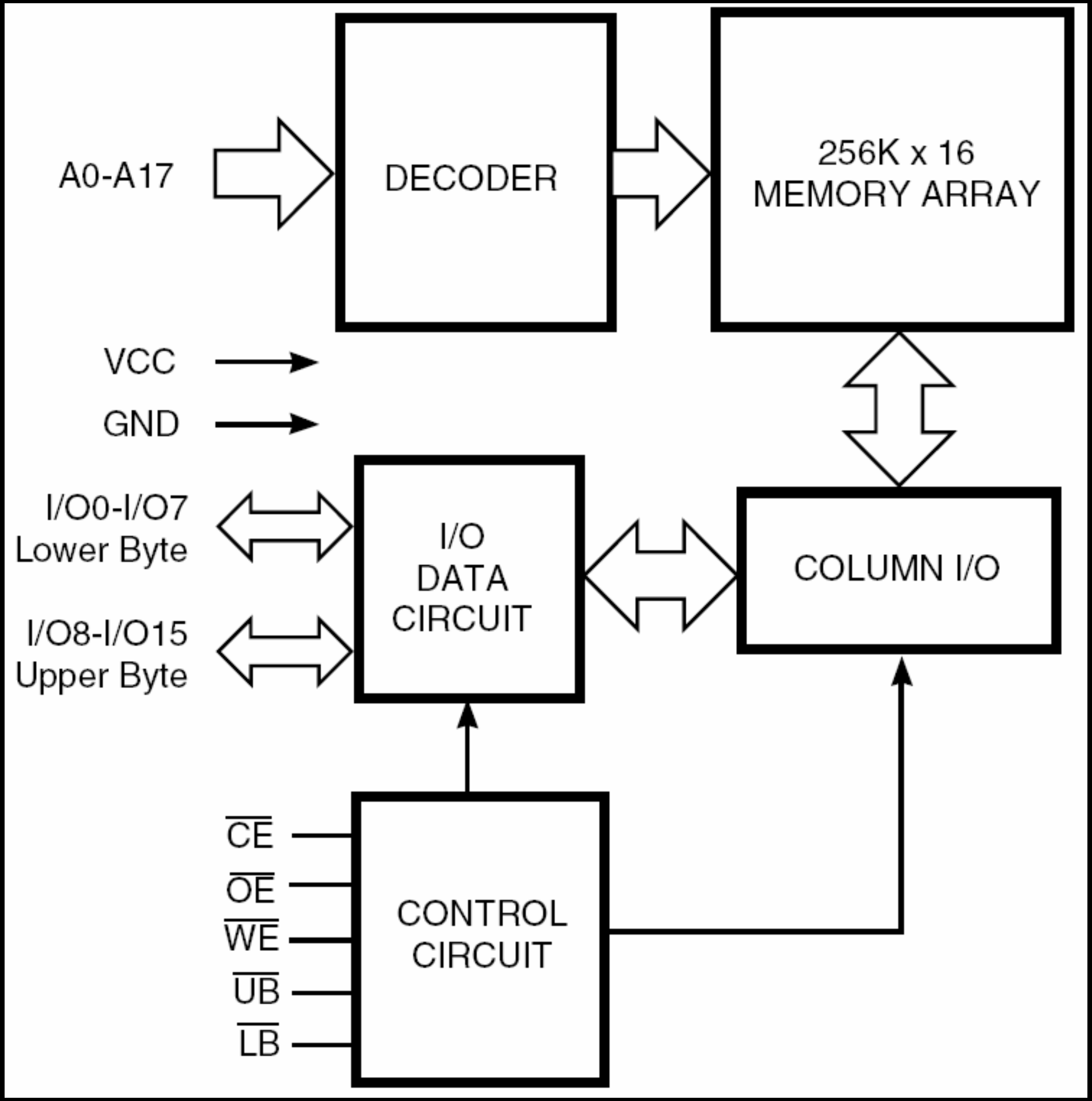
Memory Hierarchies



Addressing Bytes

- Although memory chips may have 16 or 32 bits per word, processors typically require byte addressability (characters are one byte)
- To read one byte is easy: read the whole word and discard what you do not need
- For writing only one byte, memory chips must provide byte addressability through additional signals
- byteenable signals, if the corresponding line is active, write that byte
- Why byteenable for reads?

Memory in Lab 4



byteenable signals

- $\overline{UB}, \overline{LB}$
- Active low (like other control signals)

Data Pins

- Both input and output
- When output enable $\overline{OE}$ is active (low), memory drives values to I/O pins, otherwise memory will not drive anything (Z)
- Need to use tri-state buffer to drive these pins for writing, as we want to drive the values on those pins when writing, never when reading

Tri-State Buffers in Verilog

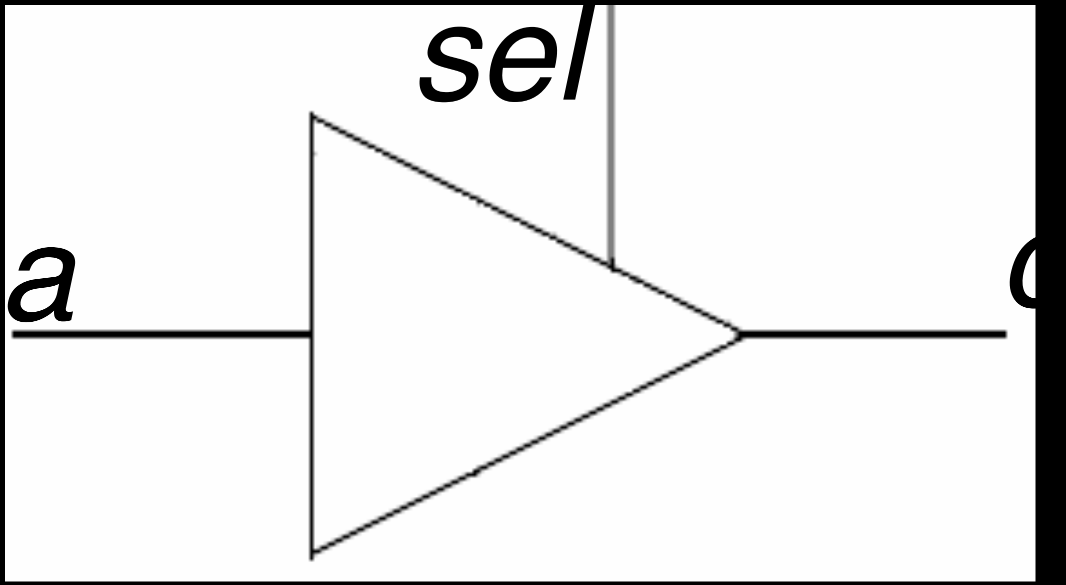
- In Verilog, there is a special port for tri-stated pins called inout
- They can drive the special value z

```
module tristates (input [7:0] data, input enable, inout [7:0]
data_out);

assign data_out = {8{1'bz}};

endmodule
```

However, that is not really what we want to do

Verilog `select` operator

```
assign c = sel ? a:1’bz;
```



Data Sheets

- Usually has everything you need to know: definitions of signals, etc.
- Key information not discussed here: timing diagrams

Principle of Locality

- Programs tend to reuse data and instructions near those they have used recently, or that were recently reference themselves
- *Temporal locality* refers to the recently referenced items being likely to be referenced in the near future
- *Spatial locality* refers to items with nearby addresses tend to be reference close together in time

Example

Where can one find spatial and temporal locality in this piece of code?

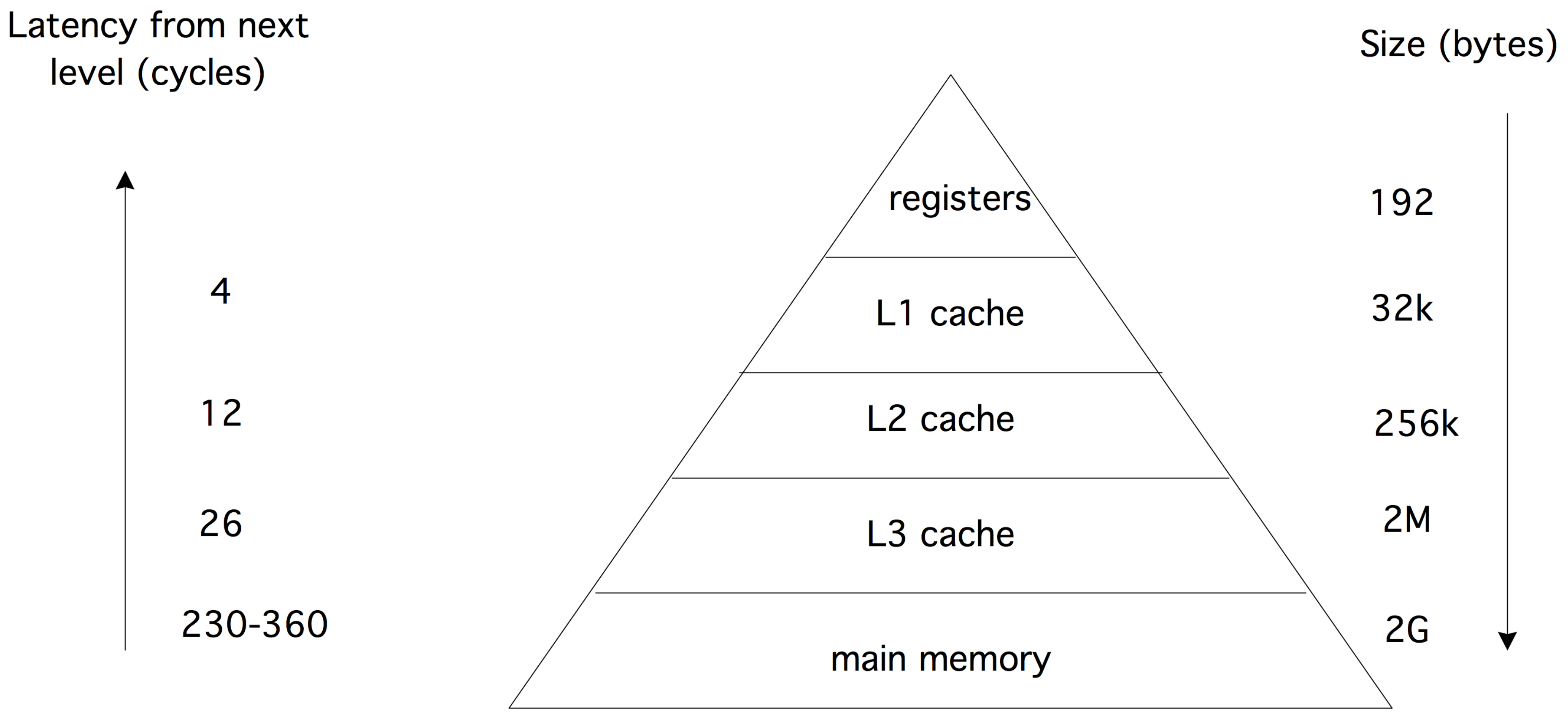
```
sum=0;
for (i=0; i<n; i++)
sum += a[i];
return sum;
```

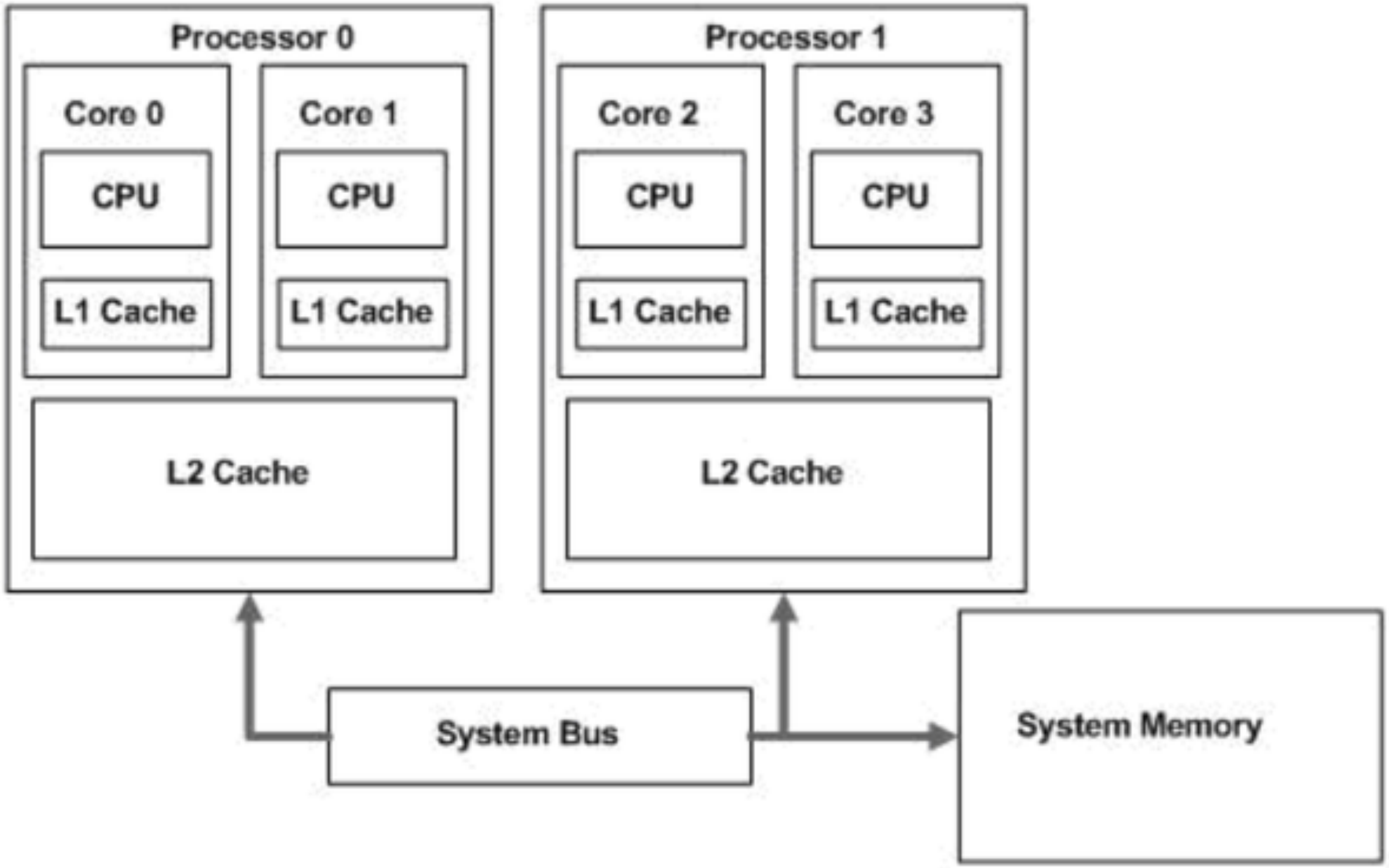
Memory Hierarchies

- Some fundamental and enduring properties of hardware and software:
  - Fast storage technologies cost more per byte, have less capacity, and require more power (and hence generate more heat)
  - The gap between CPU and main memory speed is widening
  - Well-written programs tend to exhibit good locality
- These fundamental properties complement each other well
- Suggest an approach for memory organization called *memory hierarchy*

Caches

- A cache is a smaller, faster storage device that acts as a staging area for a subset of the data in a larger, slower device
- Fundamental idea: for each k, the faster, smaller device at level k serves as a cache for the larger, slower device at level $k + 1$
- Why does this work?
  - Programs tend to access the data at level k more often than they access data at level $k + 1$
  - Thus, the storage at level $k + 1$ can be slower, and thus larger and cheaper per bit
  - The net effect is a large pool of memory that costs as much as the cheap storage near the bottom, but that serves data to programs at the rate of the fast storage near the top


Latency from next level (cycles)
4
12
26
230-360
registers
L1 cache
L2 cache
L3 cache
main memory
Size (bytes)
192
32k
256k
2M
2G


Processor 0
Core 0
CPU
L1 Cache
Core 1
CPU
L1 Cache
L2 Cache
Processor 1
Core 2
CPU
L1 Cache
Core 3
CPU
L1 Cache
L2 Cache
System Bus
System Memory

Caching in Embedded Processors

Often have L1 and L2 caches

Placement Policies

- Direct-mapping: each block of data in level $k + 1$ is mapped to exactly one block in level k
- Fully associative: each block of data in level $k + 1$ can be anywhere in level k
- n-way associative: each block of data in level $k + 1$ can be in one of n blocks in level k

Replacement Policies

- Random
- LRU (Least Recently Used)
- FIFO

Lots of work on customizing cache replacement algorithm for particular applications

Performance Calculations

Larger caches achieve lower miss rates but higher access cost. Some example calculations:

- 2K cache: hit rate 0.85, hit access 2 cycles, miss access 20 cycles, yields an average of (0.85)(2)+(0.15)(20) = 4.7 cycles per memory access
- 4K cache: hit rate 0.935, hit access 3 cycles, miss access 20 cycles, yields an average of (0.935)(3)+(0.065)(20) = 4.105 cycles per memory access
- 8K cache: hit rate 0.94435, hit access 4 cycles, miss access 20 cycles, yields an average of (0.94435)(4)+(0.05565)(20) = 4.8904 cycles per memory access