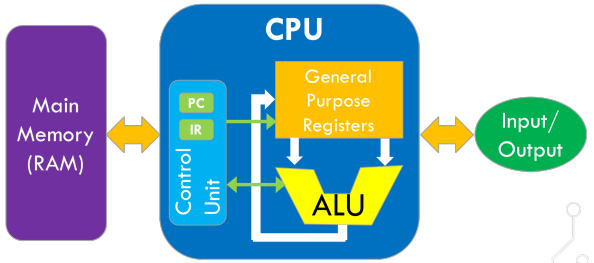
# 10 – MEMORY HIERARCHY AND PARALLELISM

COMP256 – COMPUTING ABSTRACTIONS

DICKINSON COLLEGE

## A STORED PROGRAM MACHINE

- Abstractions:
  - Transistors
  - Gates
  - Circuits
  - Micro-Instructions
  - Machine Language
    - Fetch/Decode/Execute




## IMPROVING PERFORMANCE

- Better Technology
  - Moore's Law
  - Bigger / Better CPU Designs
  - Bigger Memories
- Better Software/Algorithms
  - COMP 232 / COMP 332

## DESIGN GOALS AND TRADEOFFS

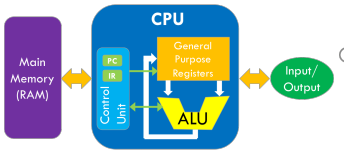
- In general… Want the biggest/fastest/cheapest/simplest thing…
- Tradeoff Rules of Thumb:
  - Hardware is faster but more complex, more expensive and harder to change than software
  - *On chip* is faster but more expensive than *off chip*.
    - Registers or ALU vs RAM or I/O devices
  - Electronic (solid state) is faster but more expensive than mechanical.
    - RAM vs Hard Disk Drive (HDD)
    - Solid State Drive (SSD) vs HDD
  - *Parallel execution* is more expensive and more complex but faster than *sequential execution*.

# BIGGER / BETTER CPU DESIGNS

- Additional transistors made it possible to:
  - Add floating point arithmetic circuits to the ALU
  - Expand CPU instruction word size from 8 to 16 to 32 to 64 bits.
  - Add machine language instructions for enhanced sound and graphics.
  - **Add *Caching* to improve memory speed.**
  - **Utilize *Parallelism* to improve computation speed.**

# MEMORY SPEED

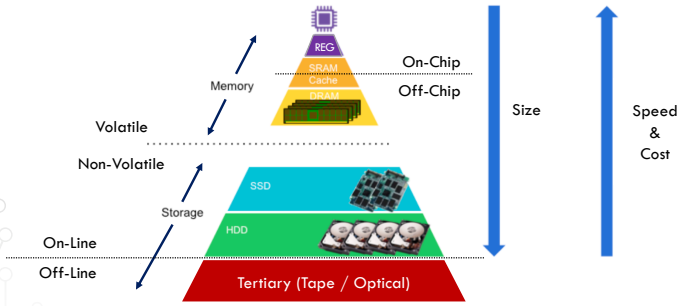
- Want to run as many/as big programs as possible as fast as possible.
  - CPU's getting faster more quickly than main memory.
  - Speed gap means CPU has to wait for instructions and data to process.
    - The "*Von Neuman Bottleneck*"




| Year | CPU Speed | Memory Speed | Ratio |
|---|---|---|---|
| 1986 | 25 MHz | 25MHz | 1:1 |
| 1998 | 350 MHz | 100 MHz | 3.5:1 |
| 2004 | 2800 MHz | 500 MHz | 5.6:1 |
| 2009 | 3200 MHz<br>x 4 cores<br>12,800 MHz | 1333 MHz | 9:1 |
| 2018 | 4500 MHz<br>x 18 cores<br>81,000 MHz | 2666 MHz | 30:1 |

# THE MEMORY HIERARCHY

- There are many types of memory in a modern computer system.
  - The amount of each is a tradeoff between speed and cost.




Images adapted from: https://www.rambus.com/blogs/mid-when-memory-and-storage-converge/

# MEMORY SIZE ABBREVIATIONS

- Memory uses "metric-like" prefixes for size:

| | | | |
|---|---|---|---|
| • Byte (B) | $2^0$ | 1 byte | Registers |
| • Kilobyte (KB) | $2^{10}$ | 1,024 bytes | Cache |
| • Megabyte (MB) | $2^{20}$ | 1,048,576 bytes | |
| • Gigabyte (MB) | $2^{30}$ | 1,073,741,824 bytes | RAM/SDD |
| • Terabyte (TB) | $2^{40}$ | 1,099,511,627,776 bytes | HDD/Off-line |
| • Petabyte (PB) | $2^{50}$ | 1,125,899,906,842,624 bytes | |

# CHEF CACHE ANALOGY

- Without cache:
  - All ingredients (data and instructions) are in the pantry (main memory).
  - When needed ingredients are moved from the pantry to a small counter (registers) closer to the stove (CPU).
  - Small counter fills up and some ingredients must be moved back to pantry to make room for what is needed next.
  - Pantry is far away making it slow and inconvenient.

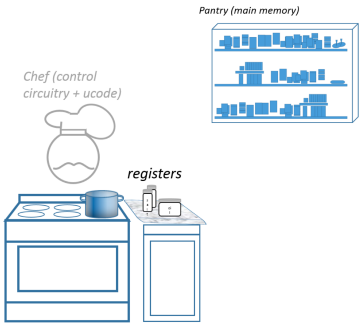



Image from: https://software.intel.com/en-us/blogs/2015/06/11/advanced-computer-concepts-for-the-not-so-common-chef-memory-hierarchy-of-registers

# CHEF CACHE ANALOGY

- Caching is a common practice in the real world.
  - We add space to keep items that we use frequently closer at hand than those we use less frequently.
  - Cached items can be retrieved more quickly than those not cached.

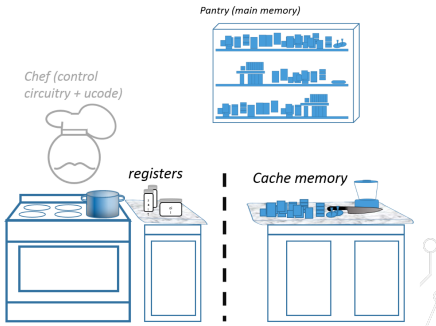
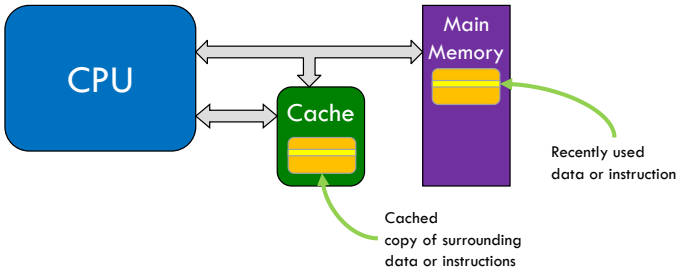



Image from: https://software.intel.com/en-us/blogs/2015/06/11/advanced-computer-concepts-for-the-not-so-common-chef-memory-hierarchy-of-registers

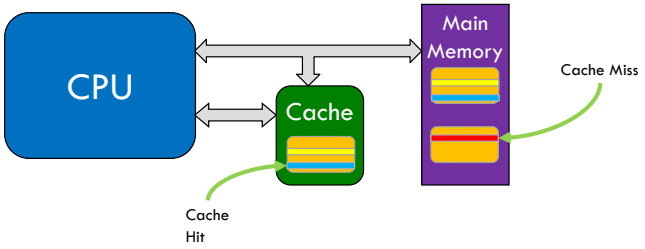
# CPU CACHE

- **CPU Cache**: A small, very fast (SRAM), memory that *stores copies of data and instructions* from main memory *(DRAM)* that have recently been used as well as some nearby data and instructions.




# CACHE HITS AND MISSES

- A *cache miss* occurs when a requested data or instruction is not in the cache.
  - Retrieved from main memory and a copy of the surrounding block is placed in cache
- A *cache hit* occurs when a requested data or instruction is in the cache.
  - Retrieved from cache without accessing main memory

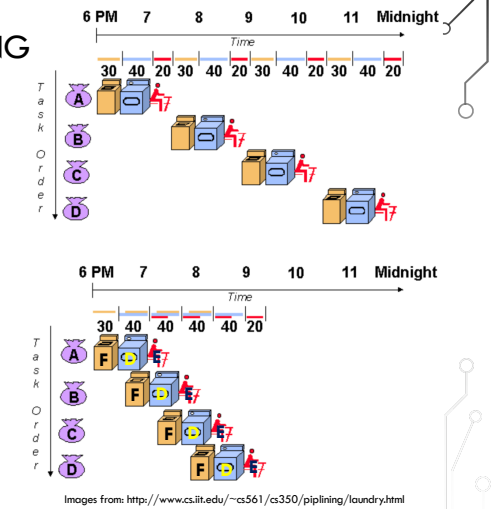


# WHY CACHE WORKS

- Locality of Reference:
  - **Spatial Locality**: If a program uses an instruction/data, then instructions/data that are close by will tend to be used soon.
  - **Temporal Locality**: If a program uses an instruction/data, then that instruction/datum will tend to be used again soon.
  - 90/10 Rule:
    - Typical programs spend 90% of their time executing 10% of their instructions.
    - Result: With only 512 KB cache, the hit rate is typically above 99.9%
      - Modern processors typically have several times that much cache.

# INSTRUCTION PIPELINING

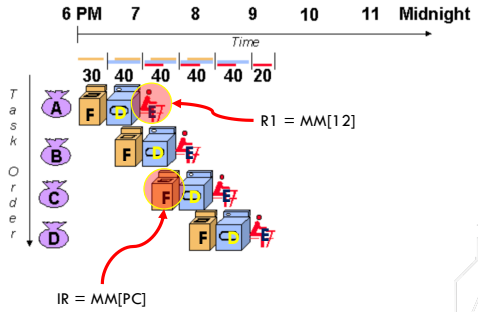
- *Pipelining* breaks a task (e.g. laundry / instruction cycle) into distinct sub-tasks that can be done in parallel.
  - Each task uses distinct hardware
    - Wash 30 min (Fetch)
    - Dry 40 min (Decode)
    - Fold 20 min (Execute)




Images from: http://www.cs.iit.edu/~cs561/cs350/piplining/laundry.html

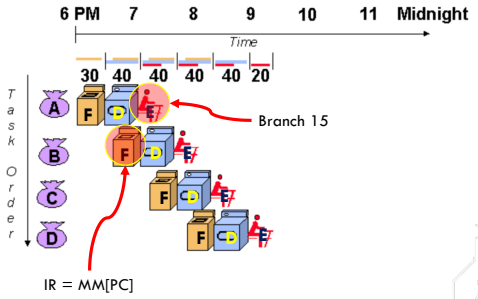
# PIPELINE RESOURCE HAZARD

- A *resource hazard* occurs when two pipeline stages require the same hardware at the same time.
  - E.g. Fetch and Execute may both need to access main memory at the same time.



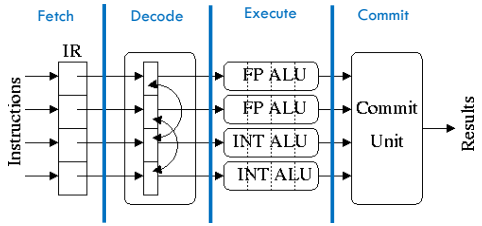

# PIPELINE CONTROL HAZARD

- A *control hazard* occurs when a branch results in an instruction that has alreadybeen fetched to not be executed.



# SUPER-SCALAR PROCESSORS

- A *Super-Scalar Processor* has multiple functional units (e.g. ALUs).
  - Multiple instructions are fetched at the same time.
  - Instructions are matched to available functional units during decoding.
  - Multiple instructions are executed in parallel.
  - Results are put back into order by a "commit unit"




# MULTICORE PROCESSORS

- A *multicore processor* has multiple complete (super-scalar pipelined) CPU *cores* on a single processor chip.
- Multiple levels of cache.
  - Individual Level 1 (L1) and Level 2 (L2) cache
    - Separate data and instructions caches at L1.
  - Shared (L3) cache

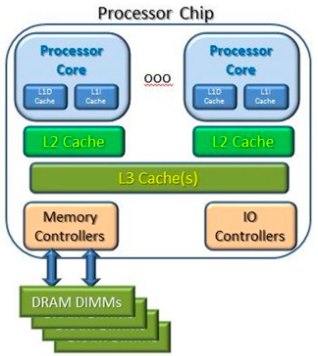



Image from: https://www.enterpriseai.news/2014/07/09/shared-memory-clusters-numa-cache-latencies/

# ADDING ABSTRACTIONS

- Main Memory – caching is (mostly) transparent to programs.
  - Programs use main memory addresses to access data and instructions.
  - Hardware handles the cache operation in the background.
- Pipelining & Super-Scalar – parallel execution is (mostly) transparent to programs.
  - Programs contain individual instructions.
  - Hardware handles the overlapped Fetch / decode / execute / (commit).