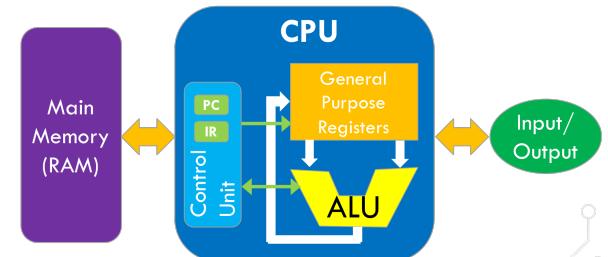


# 10 – MEMORY HIERARCHY AND PARALLELISM

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

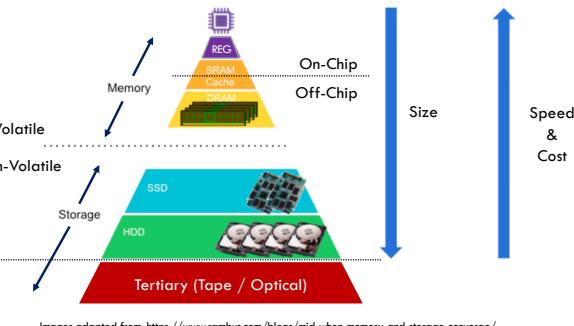
- In general... Want the biggest/fastest/cheapest/simplest
- Tradeoff Rules of Thumb:
  - Hardware is faster but more complex, more expensive and harder to change than software
  - On chip is faster but more expensive than off chip.
    - Registers or ALU vs RAM or I/O devices
  - Electronic (solid state) 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 units 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 Cache to improve memory speed.**
  - Utilize Parallelism to improve computation speed.**

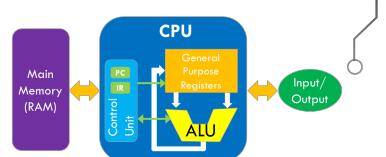
## THE MEMORY HIERARCHY

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



## MEMORY CHALLENGES

- Want to run as many/as big programs as possible as fast as possible.
  - Program number and size demands getting larger more quickly than main memory.
    - Memory size limits size and number of programs.
  - CPU's getting faster more quickly than main memory.
    - Slower memory speed means CPU has to wait.
      - 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  |

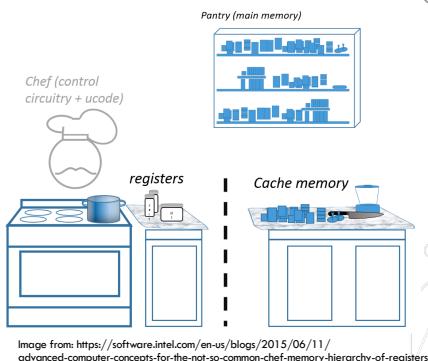
## MEMORY SIZE ABBREVIATIONS

- Memory uses "metric-like" prefixes:

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

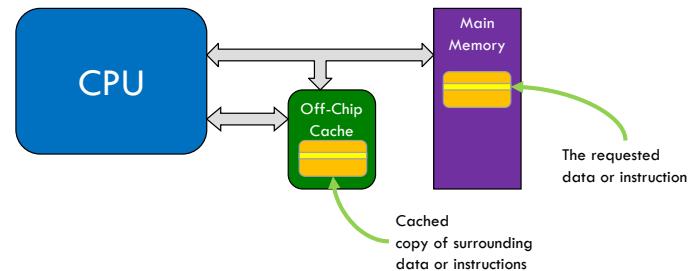
## CACHE MEMORY ANALOGY

- Caching is a common practice in the real world.
  - We 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.



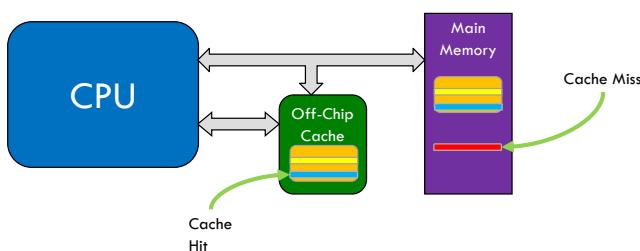
## CPU CACHE

- **CPU Cache:** A small, very fast (SRAM), memory that stores copies of data and instructions, 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



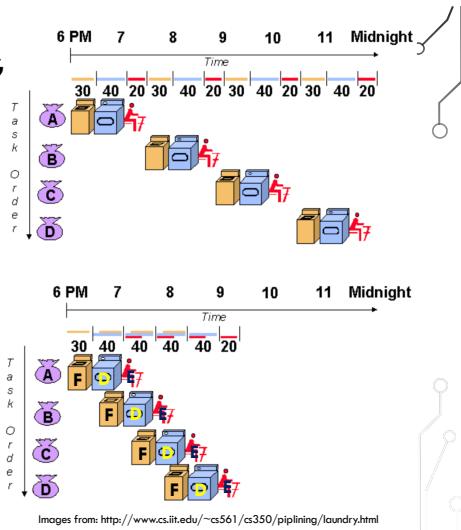
## WHY CACHE WORKS

- **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.

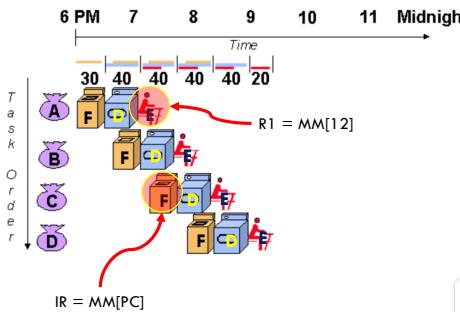
- Each task uses distinct hardware
  - Wash 30 min (Fetch)
  - Dry 40 min (Decode)
  - Fold 20 min (Execute)



## PIPELINE RESOURCE HAZARD

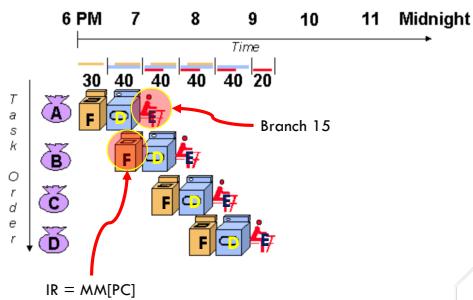
- A resource hazard occurs when two pipeline stages require the same hardware at the same time.

- E.g. Fetch and Execute may both need to access main memory at the same time.



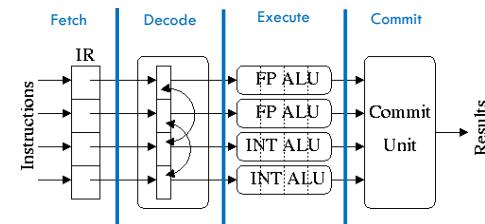
## PIPELINE CONTROL HAZARD

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



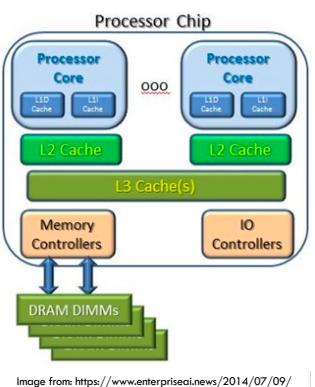
## SUPER-SCALAR PROCESSORS

- 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



## 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).