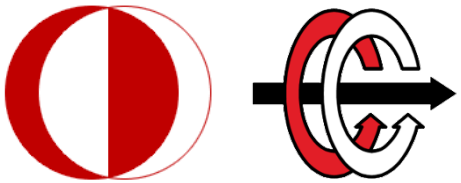


EE 446 Computer Architecture II

Course Coverage: EE445

In **EE-445** we studied:

- Computer System components
- Instruction Set Architecture (ISA) design and tradeoffs
- Hardwired and microprogrammed control for basic multi-cycle machines
- Arithmetic algorithms and implementation in hardware

Course Coverage: EE446

- Will complete the missing theoretical pieces to obtain a solid Computer Architecture/Organization background:
  - Single-Cycle datapath/controller design
  - Multi-cyle datapath/controller design
  - Pipelining, superscalar operation, and parallel processing
  - Advanced memory hierarchies, and multiprocessor buses
  - Software interactions
- In addition you will get some practical experience in the lab by applying what you learnt in EE-445/446 sequence.

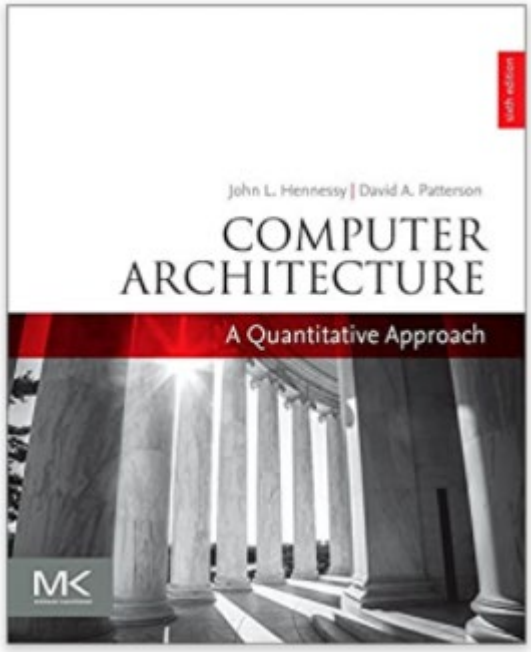
Course Outline

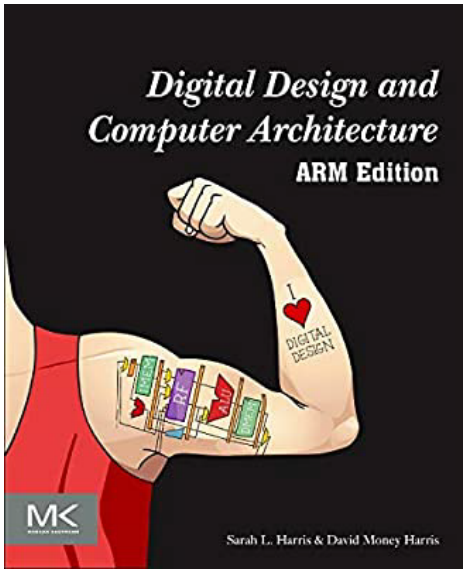
- Introduction to Computer Architecture
- Part I: Implementations of ARM Microarchitecture: Single cycle, multi cycle, pipelined. A more comprehensive coverage of pipelining, branch prediction
- Part II: Memory Hierarchy: Memory, Virtual Memory, Cache
- Part III: Advanced Topics: Superscalar Processors, possibly more topics

Copyright notice: Lecture Note Slides are compiled from the teaching material of these books, previous lecture notes of EE446 and additional resources. Part of the slides are entirely created by the Instructors

Text Books

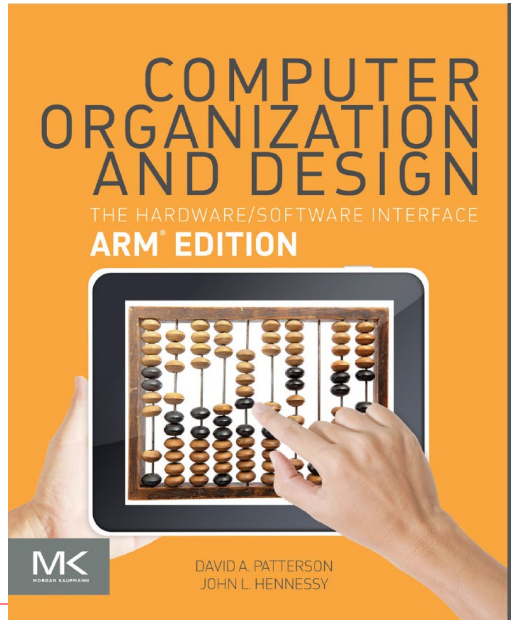
Computer Architecture, A Quantitative Approach, 6th Edition, John Hennessy, David Patterson





Harris & Harris, “Digital Design and Computer Architecture. ARM Edition”, 1st Ed., Kaufmann, 2015.

Computer Organization and Design ARM Edition: The Hardware Software Interface (The Morgan Kaufmann Series in Computer Architecture and Design) 1st Edition
by David A. Patterson, John L. Hennessy

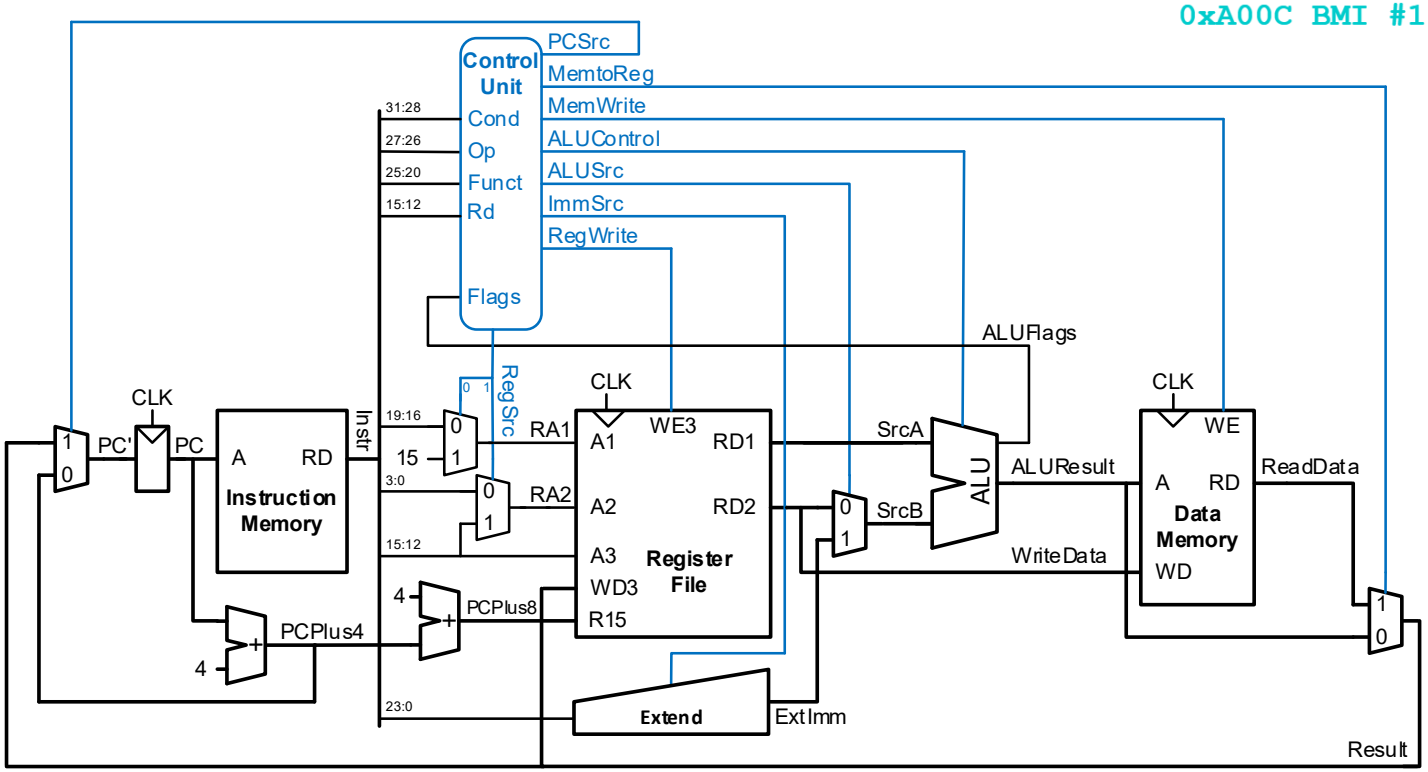


Grading

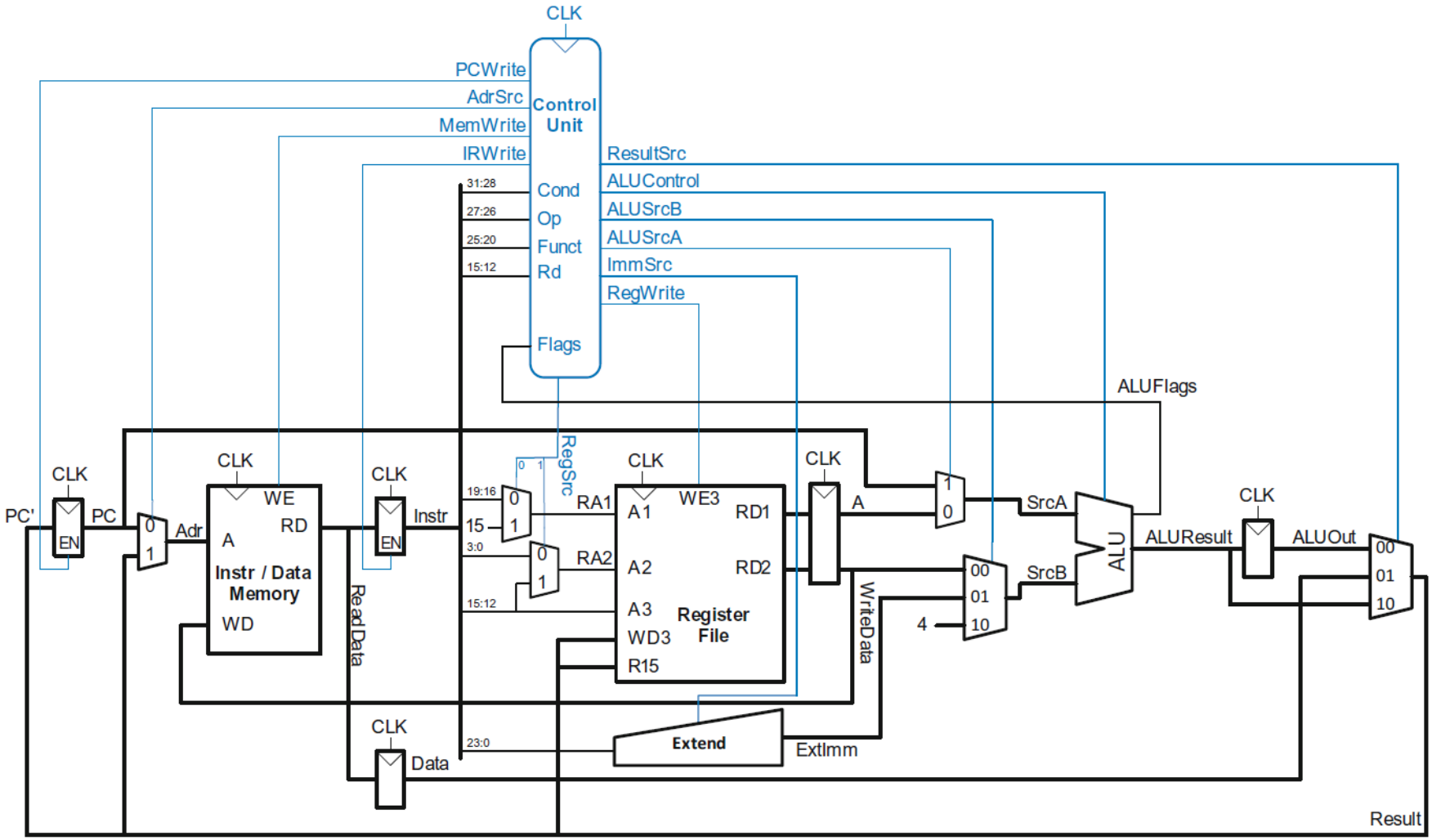
- Laboratory Work+ Class Project: 40%
- 3 Short Exams: 30%
- Final exam: 30%
- 5% bonus for attendance >=80%.
- Getting 0 from LAB 2 AND LAB3 AND LAB4➔ NA from the course

LAB 2 Single Cycle Processor

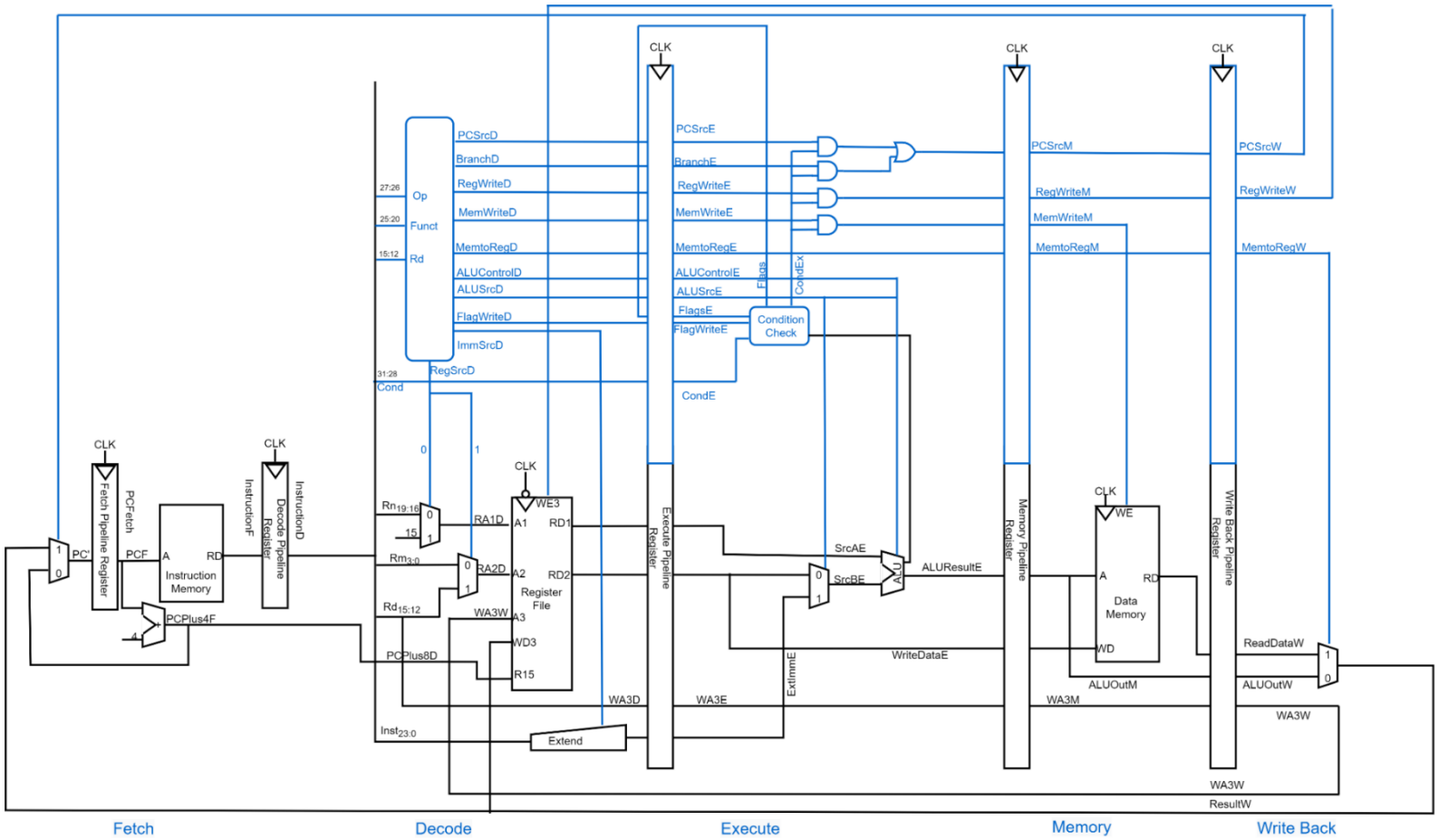
```
0xA000 LDR R2, R0, #40
0xA004 AND R3, R9, R10
0xA008 STR R4, R1, #20
0xA00C BMI #10
```

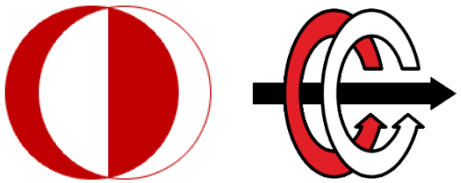


LAB 3 Multi Cycle Processor



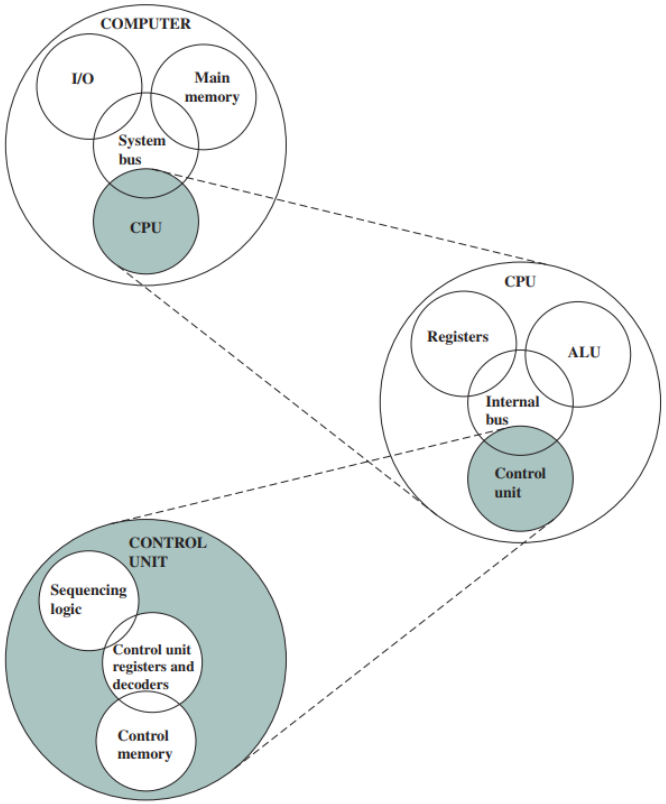
LAB 4 Pipelined Processor



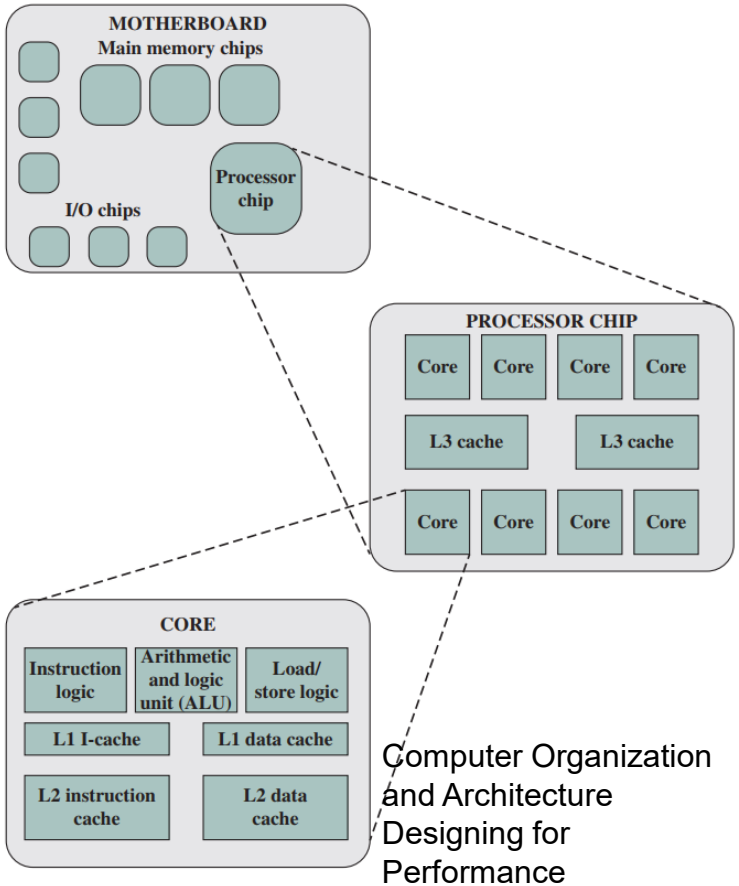


An Overview of Computer Architecture

Basic Computer Organization



Simple Single Processor Computer
(μProgram controlled)

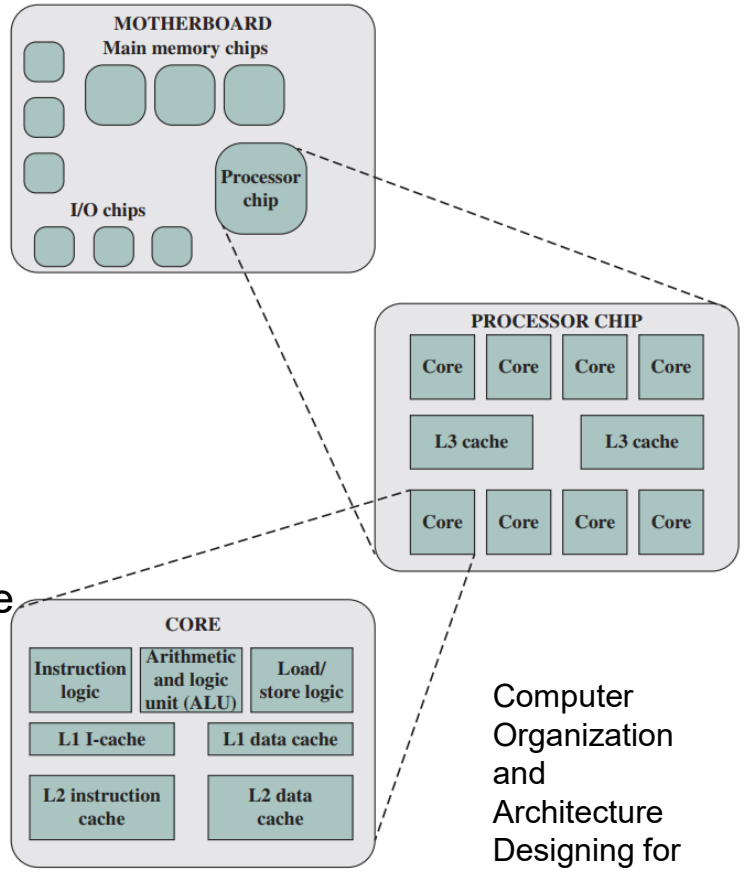


Multi-core Computer

Computer Organization and Architecture Designing for Performance
William Stallings, 2019
Eleventh Edition

Basic Computer Organization

- **Core:** An individual processing unit on a processor chip.
  - Equivalent in functionality to a CPU on a single-CPU system.
  - Other specialized processing units, such as one optimized for vector and matrix operations, are also referred to as cores.
- **Cache memory:** multiple layers of memory between the processor and main memory.
  - smaller and faster than main memory
  - used to speed up memory access, by placing in the cache data from main memory, that is likely to be used in the near future.
  - Multiple levels of cache, with level 1 (L1) closest to the core and additional levels (L2, L3, and so on) progressively farther from the core.
  - instruction cache (I-cache) that is used for the transfer of instructions to and from main memory,
  - data cache, for the transfer of operands and results.

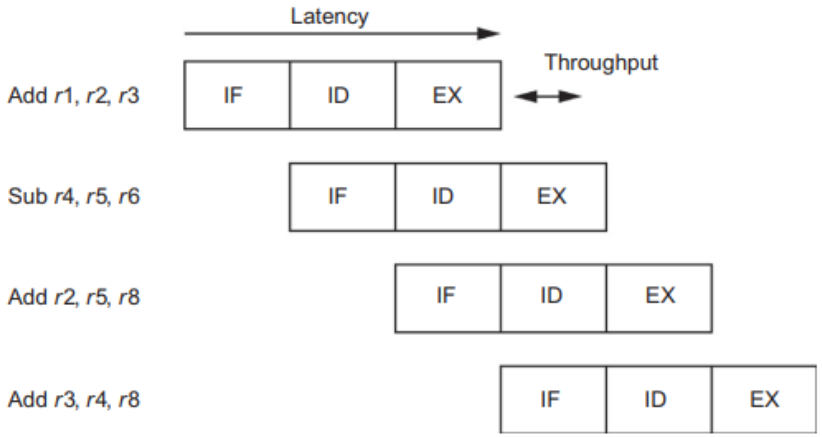


Multi-core Computer

Computer Organization and Architecture Designing for Performance William Stallings, 2019 Eleventh Edition

Performance Overview

- Microscopically: a window of a few instructions



simple pipeline diagram that shows the execution of several instructions

- Latency: Time required to execute an instruction from start to finish
- Throughput: the rate at which instructions are finished.
- Even if it takes several clock cycles to execute an instruction, the processor may still be able to finish one instruction per cycle
- Execution time
  - Wall clock time: includes all system overheads
  - CPU time: only computation time
- Speedup of X relative to Y
  - Execution time$_Y$ / Execution time$_X$

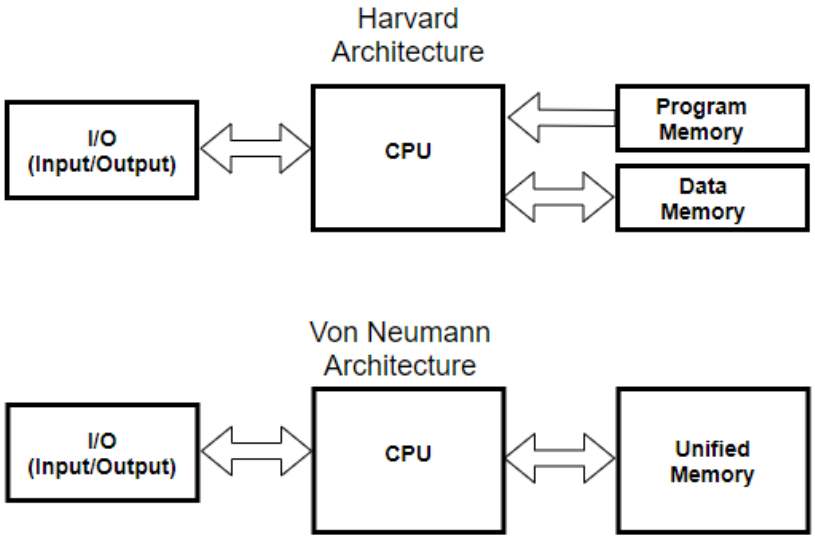
Performance Overview

- Macroscopically: over large programs

- Peak performance:
  - instruction throughput proceeds at its maximum rate
  - all processor resources are fully utilized.
- Average performance: generally measured by executing a set of benchmarks on sample data
  - Kernels (e.g. matrix multiply)
  - Toy programs (e.g. sorting)
  - Synthetic benchmarks (e.g. Dhrystone)
  - Benchmark suites (e.g. SPEC06fp, TPC-C)
- Worst case performance:
  - embedded system metric
  - determined for a particular program running on a given processor.
  - generally determined by analysis because of the difficulty of determining an input set that can be used to cause the worst-case execution.

Taxonomy of Computer Architecture

- Two flavors of computer architectures
- Based on if “data” and “instructions” should be mixed.

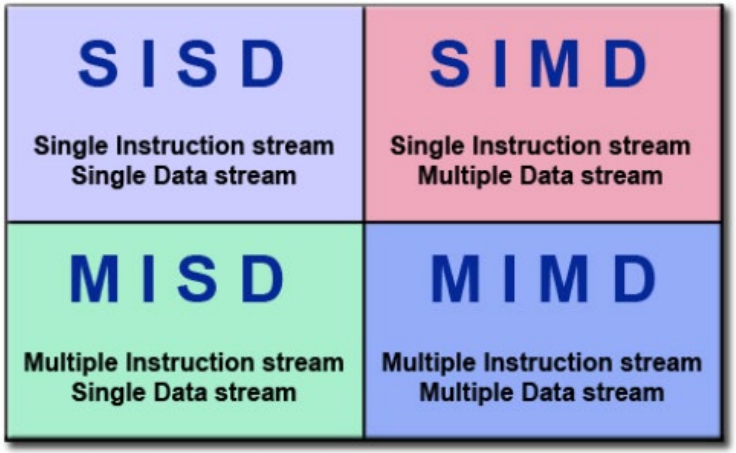


http://www.csbio.unc.edu/mcmillan/index.py?run=Courses.Comp411F17

Taxonomy of Processors: Focus on Parallelism

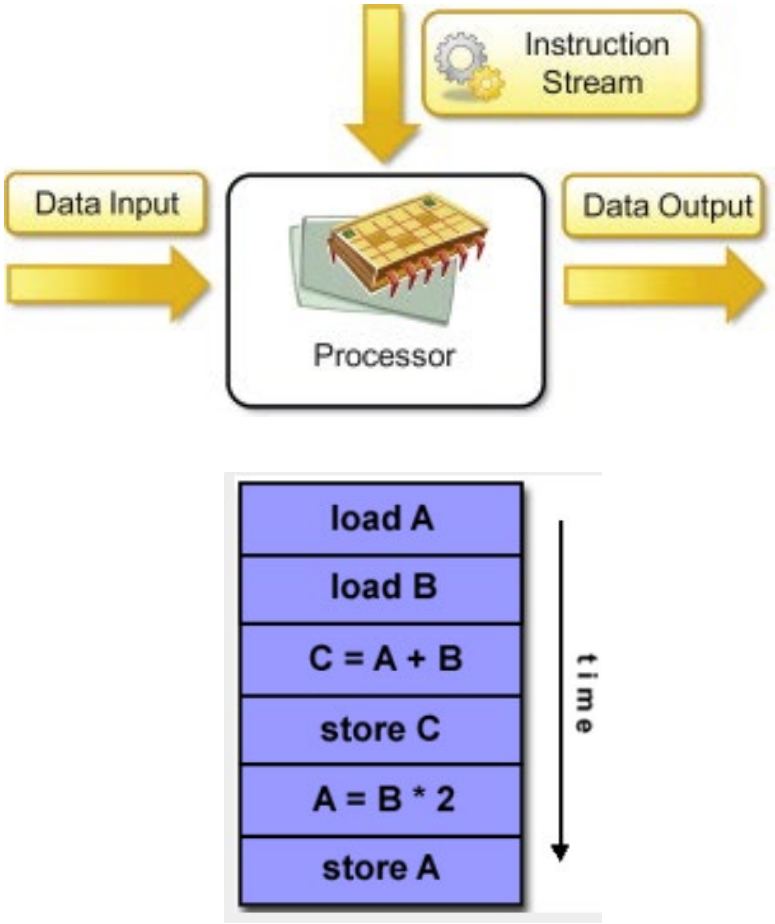
- Flynn's taxonomy
- Instruction stream (executed code), Data stream, Multiple processors
- Possible cases:
  - Each processor has the same instruction stream, execute the same code.
  - Each processor has a distinct instruction stream, each can execute a different code.
  - Each processor receives the same data stream
  - Each process receives data from a distinct data stream

Combinations



Taxonomy of Processors

- Single instruction, single data (SISD).
  - A serial (non-parallel) computer
  - Single Instruction: Only one instruction stream is being acted on by the CPU during any one clock cycle
  - Single Data: Only one data stream is being used as input during any one clock cycle
  - Deterministic execution
  - This is the oldest type of computer



Taxonomy of Processors: ISA

CISC **Complex Instruction Set Computer**	**RISC** **Reduced Instruction Set Computer**
Emphasis on hardware	Emphasis on software
Includes multi-clock complex instructions	Single-clock, reduced instruction only
Small code sizes, high cycles per second	Low cycles per second, large code sizes

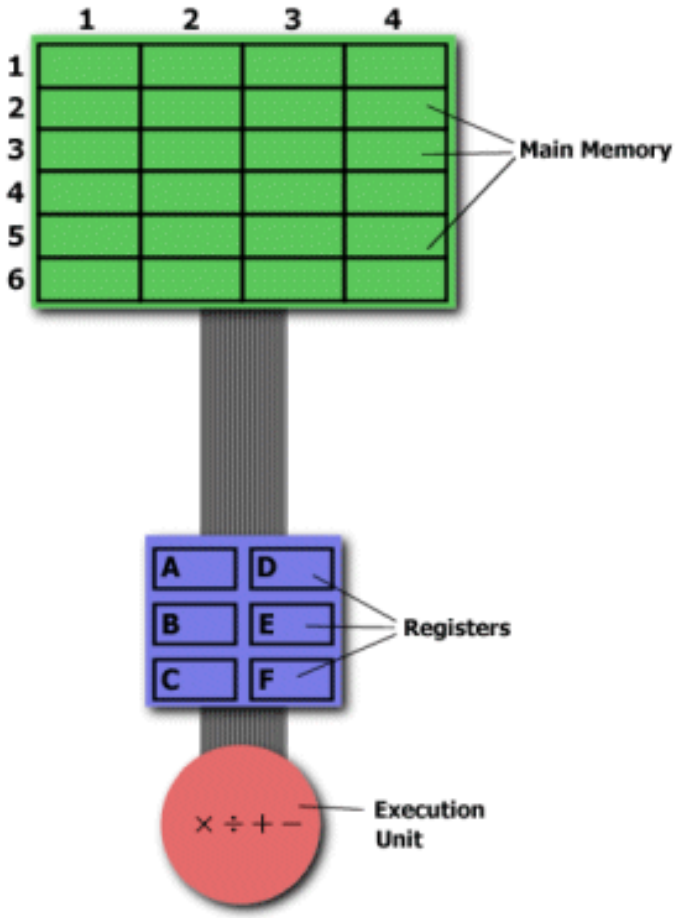
- Multiplying Two Numbers in Memory locations 2:3 and 5:2, storing in location 2:3
  - CISC : `MULT 2:3, 5:2`
  - RISC:

```
LOAD A, 2:3
LOAD B, 5:2
PROD A, B
STORE 2:3, A
```

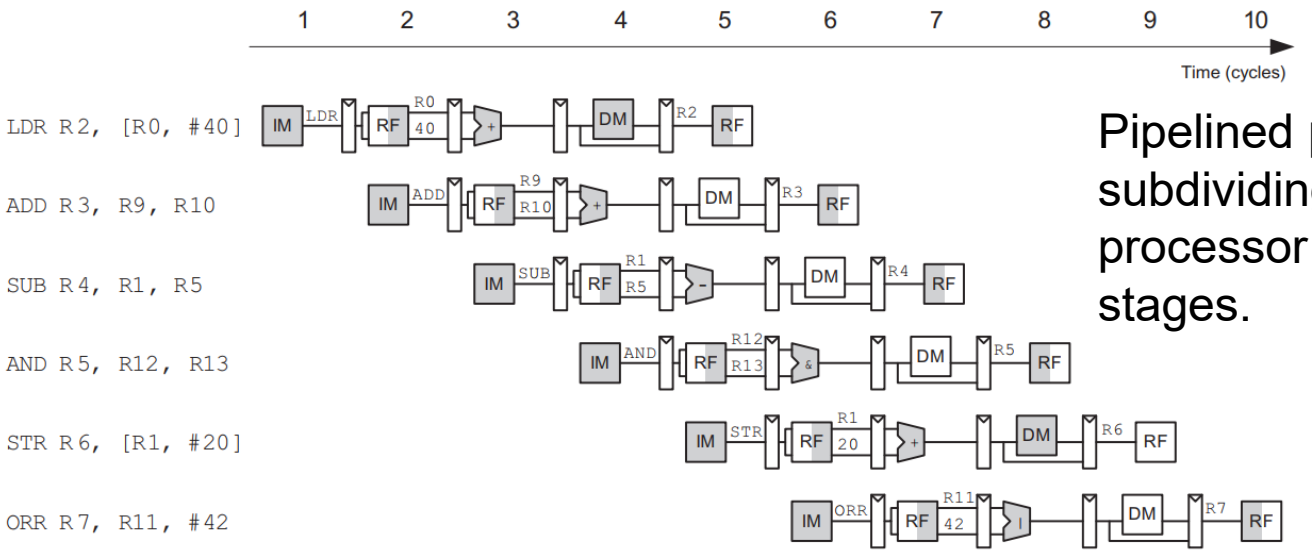
Example RISC vs CISC

- Multiplying Two Numbers in Memory locations 2:3 and 5:2, storing in location 2:3
- CISC : `MULT 2:3, 5:2`
- RISC:

```
LOAD A, 2:3
LOAD B, 5:2
PROD A, B
STORE 2:3, A
```



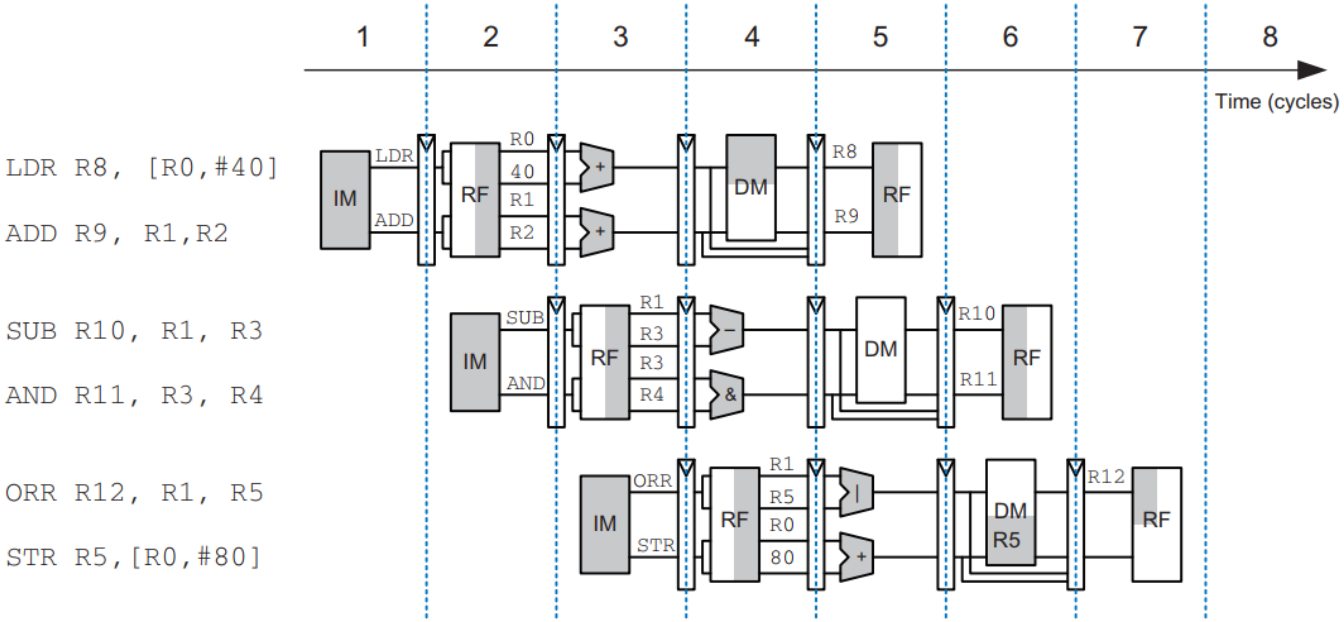
Instruction Level Parallelism (ILP): Pipelining



Pipelined processor by subdividing the single-cycle processor into five pipeline stages.

- Overlapping the execution of multiple instructions
- An instruction is partitioned into a number of functional stages

Instruction Level Parallelism (ILP): Superscalarity



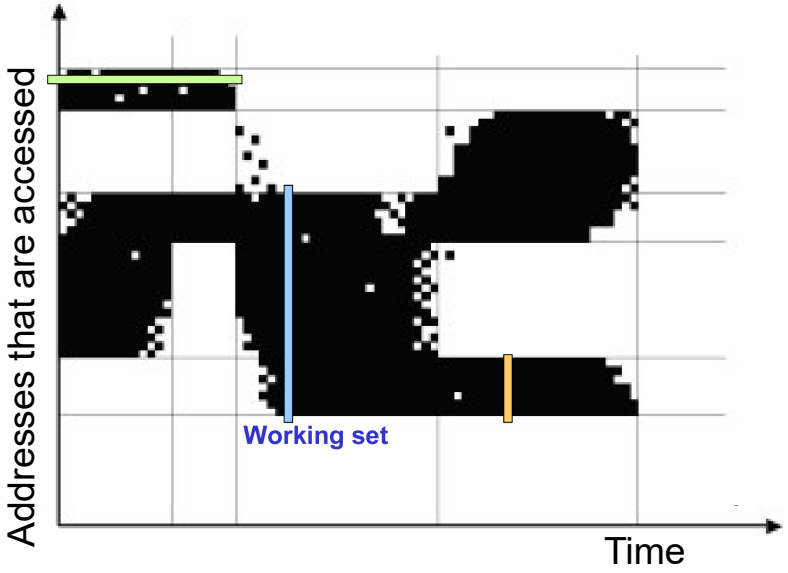
- Pipelining is a case of temporal parallelism.
- Multiple execution units is a case of spatial parallelism.
- Superscalar processors exploit both forms of parallelism

Designing for Performance

- Pipelining: The processor moves data or instructions into a conceptual pipe with all stages of the pipe processing simultaneously
- Superscalar execution: The ability to issue more than one instruction in every processor clock cycle with multiple parallel pipelines.

Principle of Locality

- Programs tend to reuse data and instructions they have used recently.
  - Temporal locality: recently accessed items are likely to be accessed soon.
  - Spatial locality: items whose addresses are near one another tend to be referenced close together in time.



- Results:
  - Prediction of what instructions and data a program will use in the near future based on its accesses in the recent past.
  - Memory Hierarchy

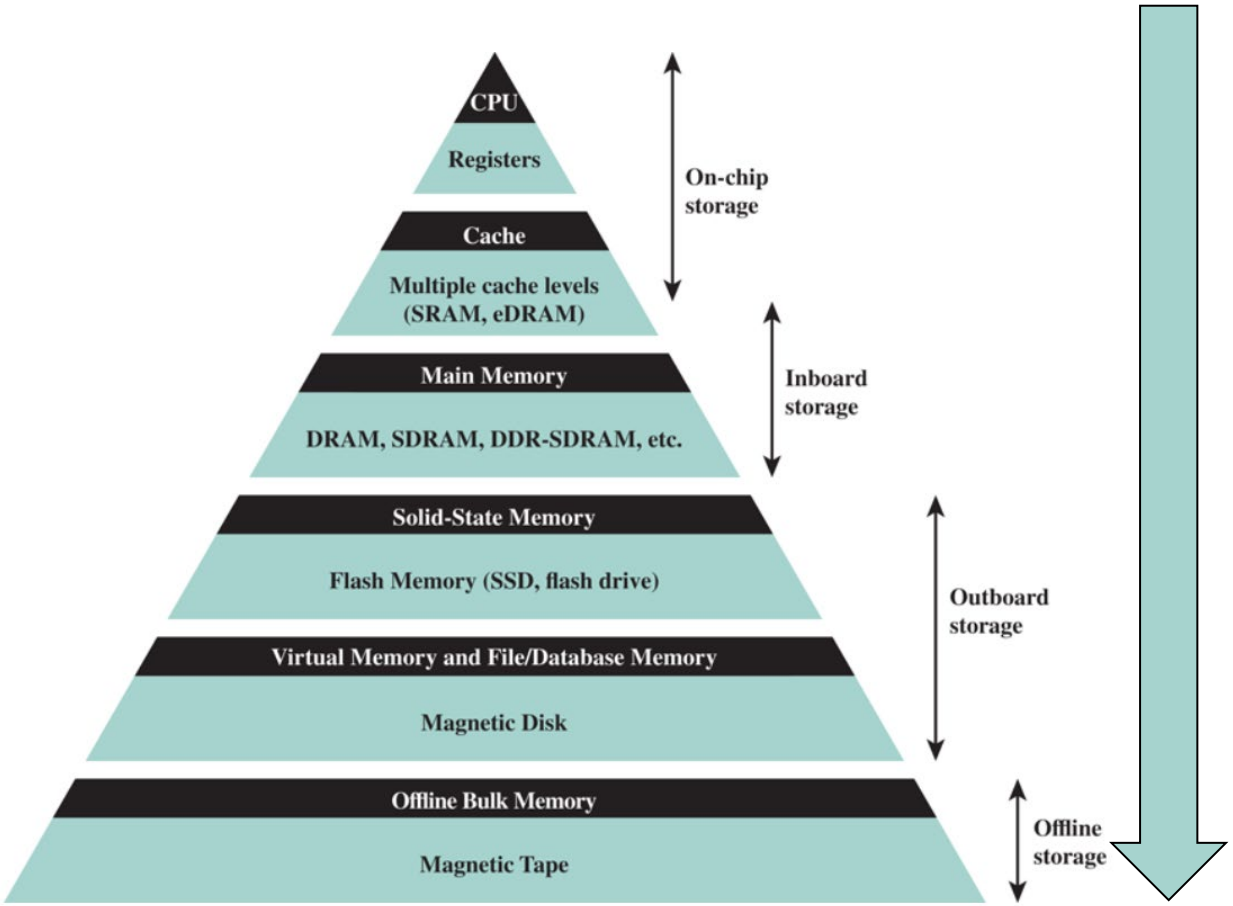
Processor Design with Locality

- Branch prediction:
  - Look ahead in the instruction code fetched from memory
  - Predict which branches, or groups of instructions, are likely to be processed next
- Data flow analysis:
  - Analyze which instructions are dependent on each other's results, or data, to create an optimized schedule of instructions
- Speculative Execution:
  - Using branch prediction and data flow analysis
  - Speculatively execute instructions ahead of their actual appearance in the program execution
  - holding the results in temporary locations, keeping execution engines as busy as possible

Memory Hierarchy Design with Locality

- Programmers want unlimited amounts of memory with low latency
- Fast memory technology is more expensive per bit than slower memory
- Solution: organize memory system into a hierarchy
  - Entire addressable memory space available in largest, slowest memory
  - Incrementally smaller and faster memories getting closer to the processor
  - Each memory contains a subset of the memory below
- Temporal and spatial locality insures that nearly all references can be found in smaller memories
  - Gives the allusion of a large, fast memory being presented to the processor

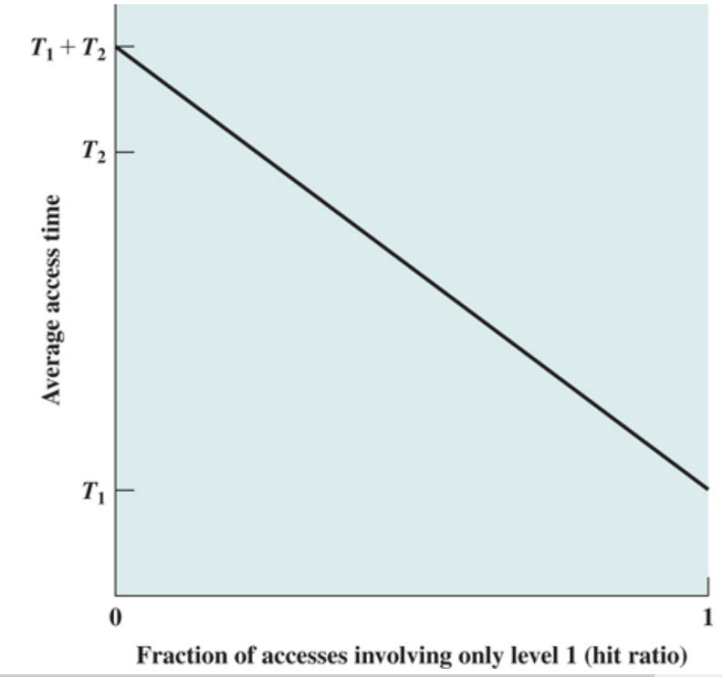
Memory Hierarchy



- Cost per byte decreases
- Average access time increases
- Average data transfer rate decreases
- Total memory size increases
- Frequency of access decreases ➔Principle of locality
- Data contained in a lower level are a superset of the next higher level ➔ Inclusion property

Memory Hierarchy: Two level example

- If a word to be accessed is in level 1, then the processor accesses it directly.
- If it is in level 2, then the word is first transferred to level 1 and then accessed by the processor.
- If the accessed word is found in the faster memory, that is defined as a hit.
- A miss occurs if the accessed word is not found in the faster memory.
- Ignore the time required for the processor to determine whether the word is in level 1 or level 2.



Level 1: Access time =T1 µs
Level 2: Access time= T2 µs.

Quantitative Principles of Computer Design

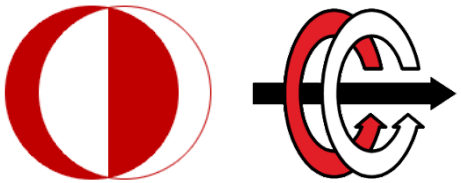
- Take Advantage of Locality
  - Branch prediction
  - Data flow analysis
  - Speculative Execution
  - Memory hierarchy, average memory access time with cache
- Focus on the common case
  - Impact of the improvement is higher if the improved case occurs frequently
  - Amdahl's law

Application of Amdahl's Law

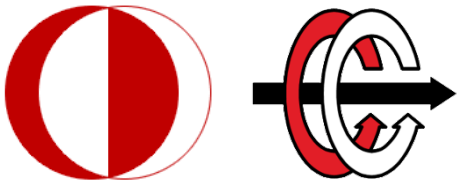
- Hardware Accelerators
  - Specialized hardware instead of general-purpose hardware
  - Performance and energy-efficiency improvements
  - FPGA, GPU
  - See: https://www.xilinx.com/developer/articles/acceleration-basics.html

- Example: A processor spends 40% of its time on computing Conjugate Gradient Algorithm. We employ an FPGA hardware accelerator which has a speed-up of 2 to run this algorithm,
- This speed-up of 2 is a real benchmark result: https://xilinx.github.io/Vitis_Libraries/hpc/2021.2/benchmark.html
- Execution time increases 25%

$$\frac{ET}{ET_{new}} = \frac{1}{(1 - F_p) + F_p/SU_p} = \frac{1}{(1 - 0.4) + 0.4/2} = 1.25$$



An Overview of Computer Architecture



EE 446 Computer Architecture II