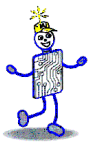


CHAPTER 8: CPU and Memory
Design, Enhancement, and Implementation

The Architecture of Computer Hardware, Systems Software & Networking:
An Information Technology Approach

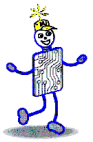
5th Edition, Irv Englander

John Wiley and Sons ©2013

PowerPoint slides authored by Angela Clark, University of South Alabama

PowerPoint slides for the 4th edition were authored by Wilson Wong, Bentley University

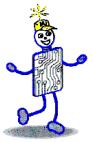
PowerPoint slides modified by Gianluca Amato, Univ. di Chieti-Pescara



Traditional Modern Architectures

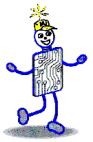
Problems with early CPU Architectures and solutions:

- Large number of specialized instructions were rarely used but added hardware complexity and slowed down other instructions
- Slow data memory accesses could be reduced by increasing the number of general purpose registers
- Using general registers to hold addresses could reduce the number of addressing modes and simplify architecture design
- Fixed-length, fixed-format instruction words would allow instructions to be fetched and decoded independently and in parallel



Current CPU Architectures

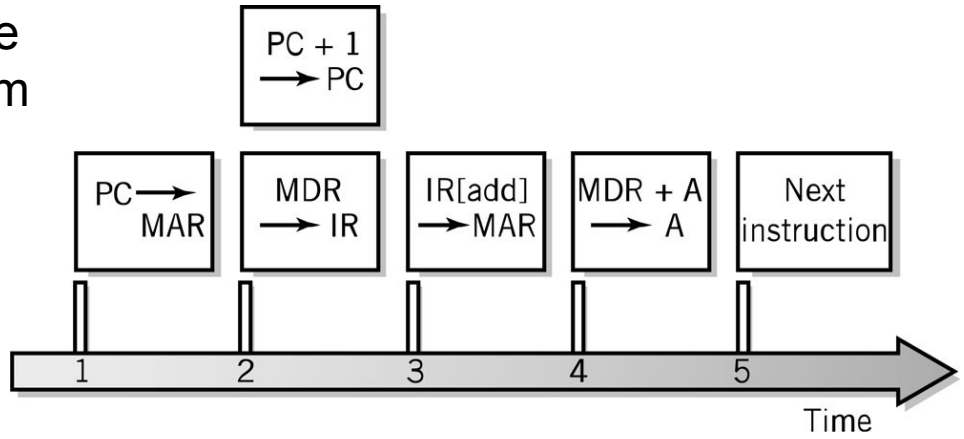
- Current CPU Architecture Design
  - Complex Instruction Set Computers (CISC)
  - Reduced Instruction Set Computers (RISC)
  - Recently difference has become blurred
- Current CPU Architectures
  - IBM Mainframe series (CISC)
  - Intel x86 family (CISC)
  - IBM POWER/PowerPC family (RISC)
  - ARM architecture (RISC)
  - Oracle SPARC family (RISC)

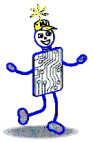


Fetch-Execute Cycle Timing Issues

- Computer clock is used for timing purposes for each step of the instruction cycle
  - Originally in the order of magnitude of MHz
  - GHz (gigahertz) – billion steps per second
- Instructions can (and often) take more than one step

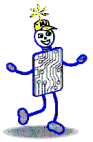
Fetch-execute timing diagram





CPU Features and Enhancements

- Separate Fetch/Execute Units
- Pipelining
- Multiple, Parallel Execution Units
- Scalar Processing
- Superscalar Processing
- Branch Instruction Processing

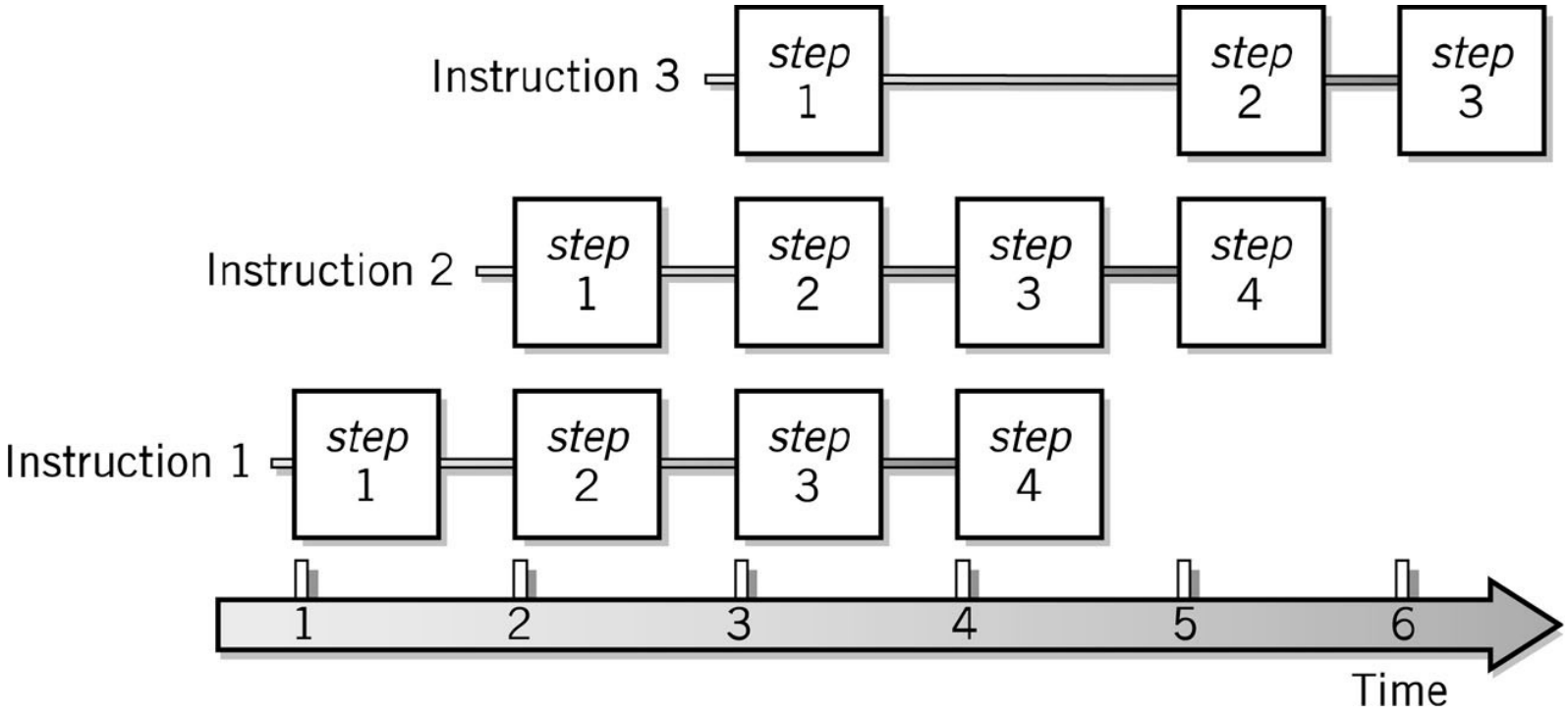


Separate Fetch-Execute Units

- Fetch Unit
  - Instruction fetch unit
  - Instruction decode unit
    - Determine opcode
    - Identify type of instruction and operands
  - Several instructions are fetched in parallel and held in a buffer until decoded and executed
- Execution Unit
  - Receives instructions from the decoder unit
  - Appropriate execution unit services the instruction

Instruction Pipelining

- Assembly-line technique to allow overlapping between fetch-execute cycles of sequences of instructions



Pipelining Problems (1)

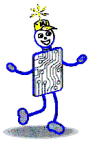
- Problems from stalling
  - Instructions have different numbers of steps
- Problems from data-dependencies
  - An instruction needs the result of another
    - `add 1 to R5`
      `copy R5 to R6`
  - The copy instruction cannot be executed if add is not terminated

Pipelining Problems (2)

- Problems from branching
  - `add 1 to R5`
    `branch to label if R5 is zero`
    `copy 0 to r6`
    …
    …
    `label:`
  - Instruction `copy 0 to r6` should executed only if branch does not occur

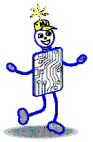
Branch Problem Solutions (1)

- Stall pipeline until branch condition may be computed
- Speculative execution
  - Continue execution, but be prepared to rollback
  - Require a very sophisticated control unit



Branch Problem Solutions (2)

- Always execute instructions after branches (and tell programmer….)
  - In this case the “no operation” is executed in every case, but it does nothing
    - ```
      add 1 to R5
      branch if R5 is zero
      nop
      ```
      ```



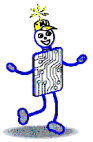
Branch Problem Solutions (3)

- In this case code which should always be executed is moved after the jump.
  - **Before:**

```
add 1 to R5
add 1 to R6
branch if R5 is zero
```

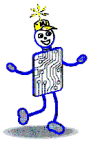
  - **After:**

```
add 1 to R5
branch if R5 is zero
add 1 to R6
```



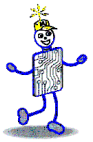
Branch Problem Optimization

- Separate pipelines for both possibilities
- Branch prediction
  - Branch History Table
  - Hint from the programmer:
    - Example: two BRZ instruction which tell programmer what is more likely, branch taken or not
- Instruction reordering



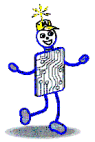
Multiple, Parallel Execution Units

- Different instructions have
  - different numbers of steps in their cycle
  - differences in each step
- Use different execution unit for classes of instructions
- Multiple execution units permit simultaneous execution of several instructions

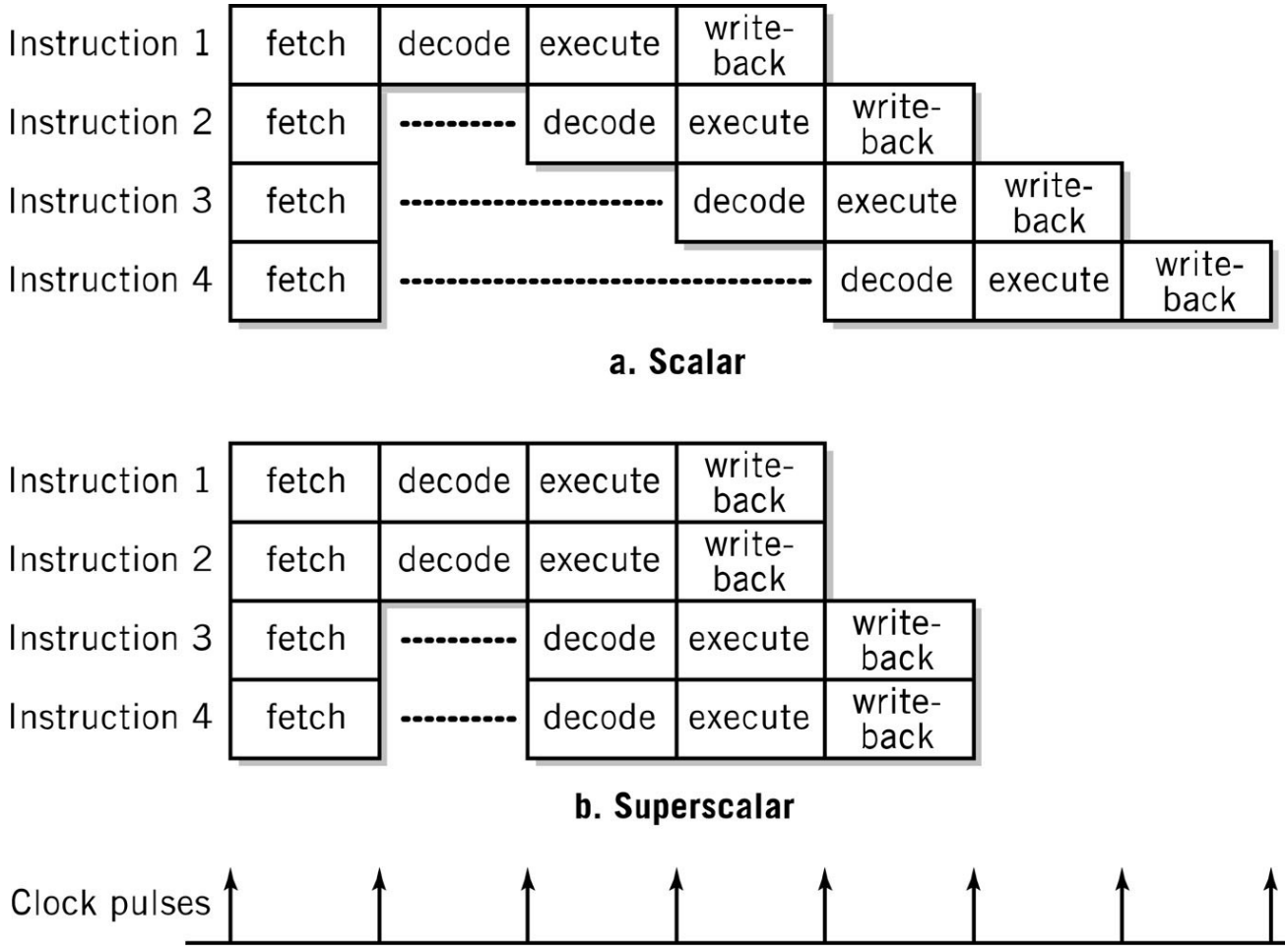


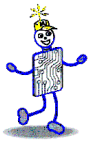
Scalar and Superscalar procesing

- Scalar processing
  - One instruction is started each clock cycle.
- Superscalar processing
  - More the one instruction is started for each clock cycle.
  - Requires multiple pipelines



Scalar vs. Superscalar Processing

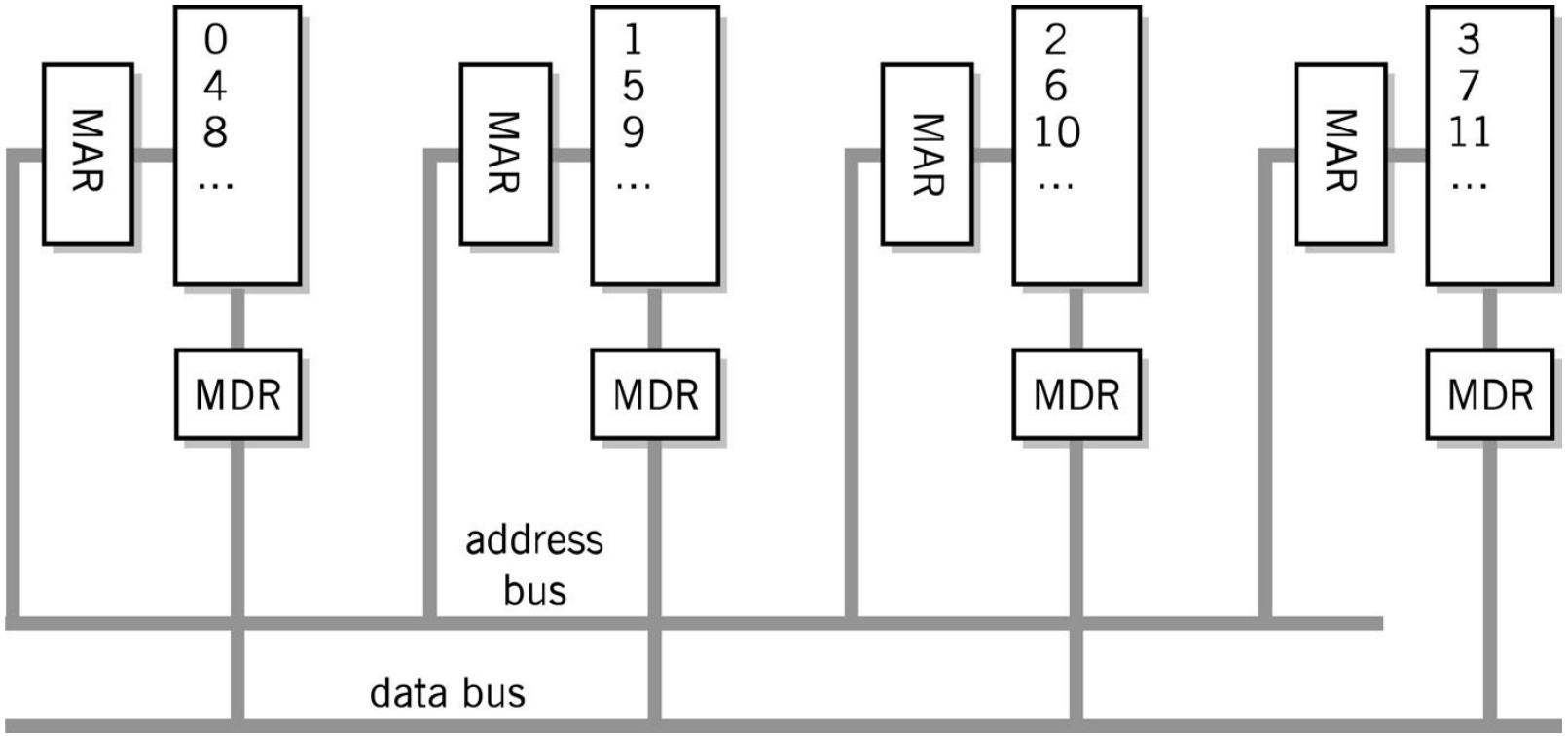


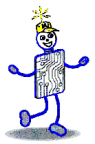


Memory Enhancements

- Memory is slow compared to CPU processing speeds!
  - 2Ghz CPU = 1 cycle in ½ of a billionth of a second
  - 70ns DRAM = 1 access in 70 billionth of a second
- Methods to improvement memory accesses
  - Wide Path Memory Access
    - Retrieve multiple bytes instead of 1 byte at a time
  - Memory Interleaving
    - Partition memory into subsections, each with its own address register and data register
  - Cache Memory

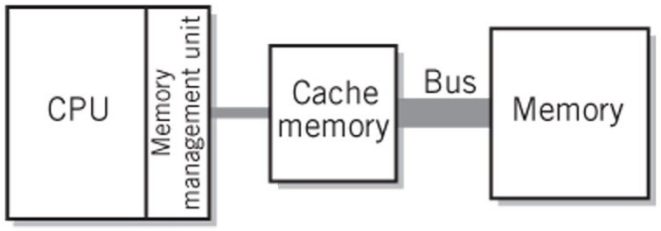
Memory Interleaving



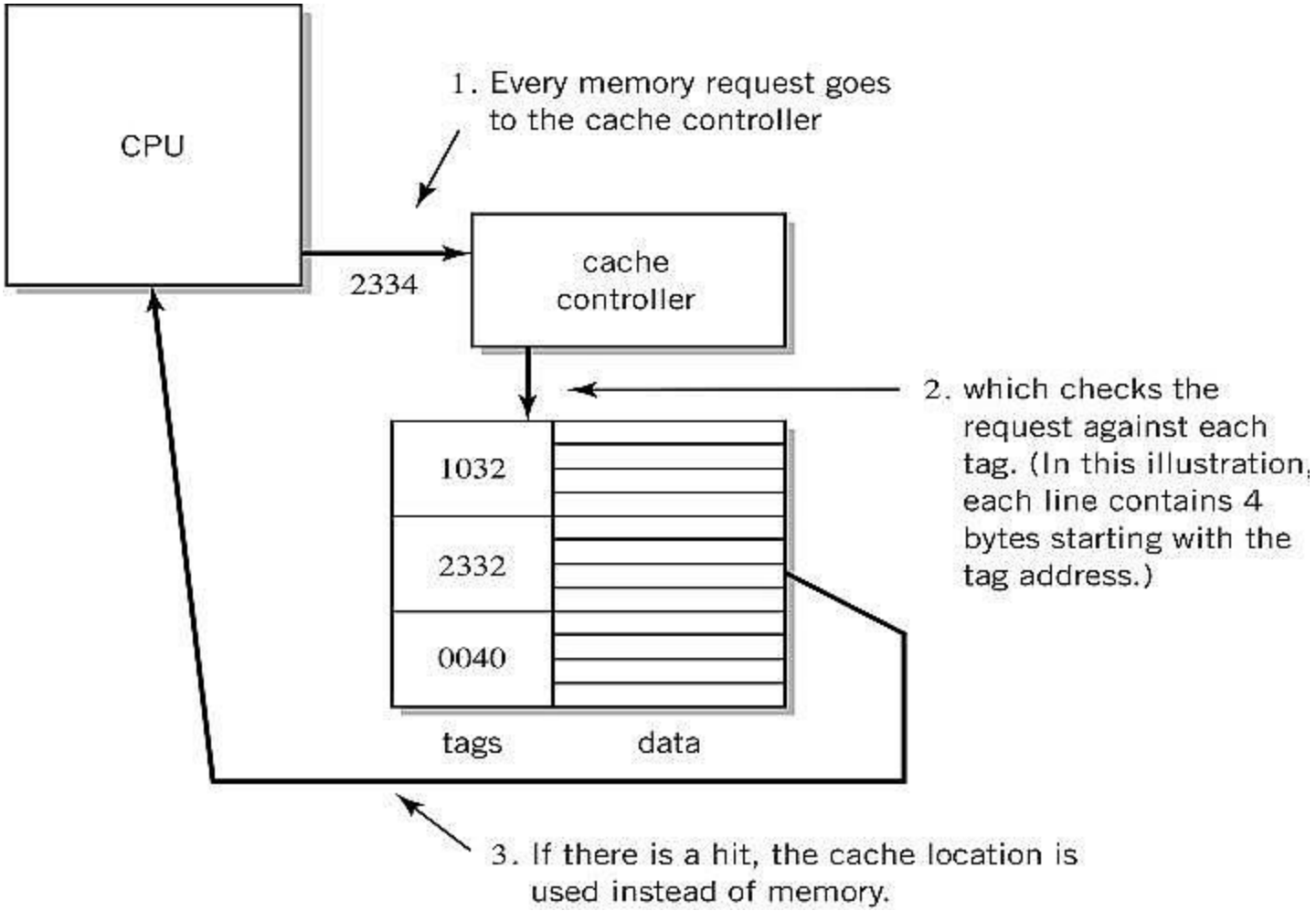


Cache Memory

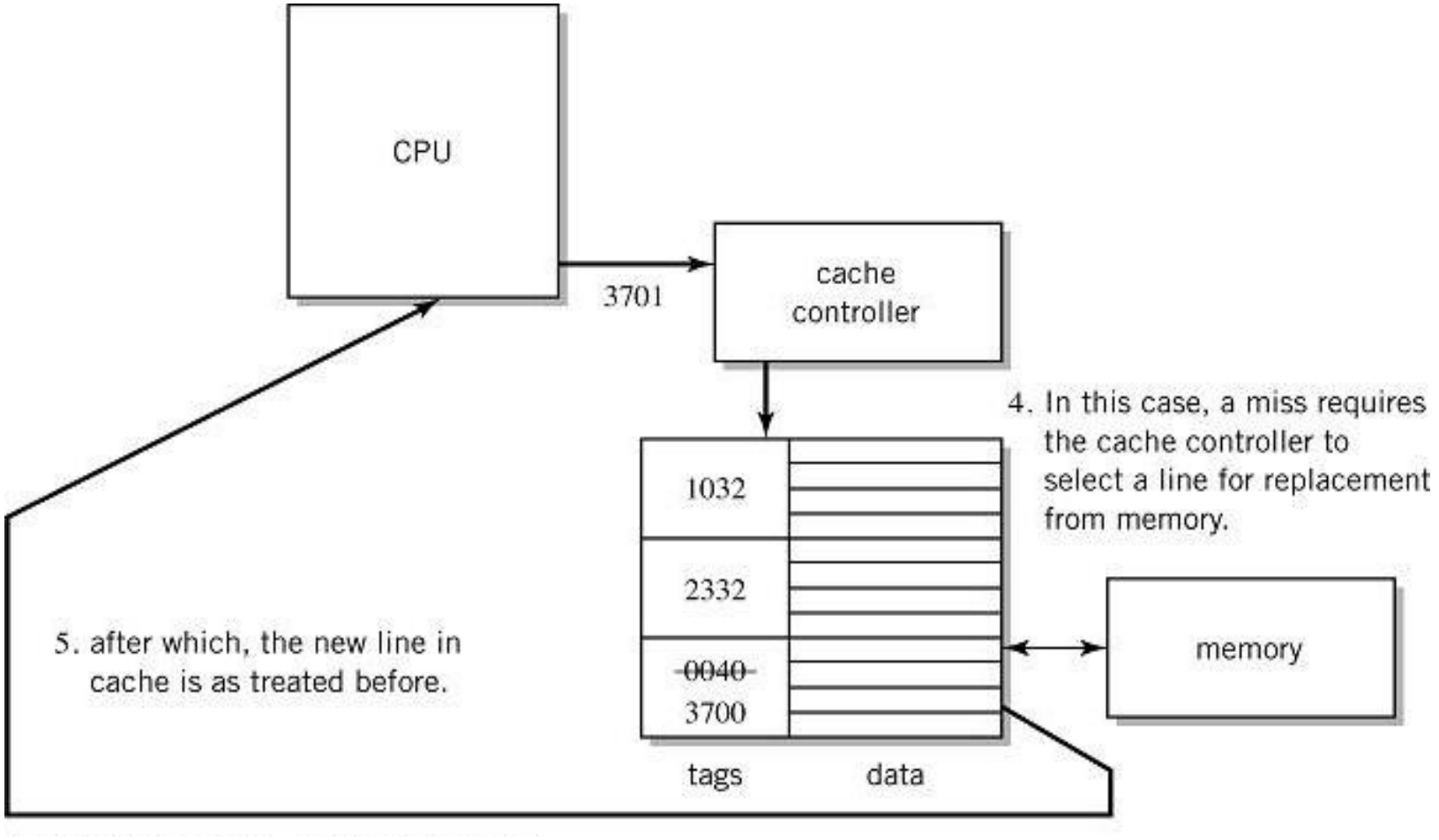
- Divided in blocks called Cache Lines
  - Between 8 and 64 bytes
  - Unit of transfer between storage and cache memory
- Tags: pointer to location in main memory
- Cache controller
  - hardware that checks tags
- Hit Ratio: ratio of hits out of total requests
- Synchronizing cache and memory
  - Write through
  - Write back



Step-by-Step Use of Cache



Step-by-Step Use of Cache

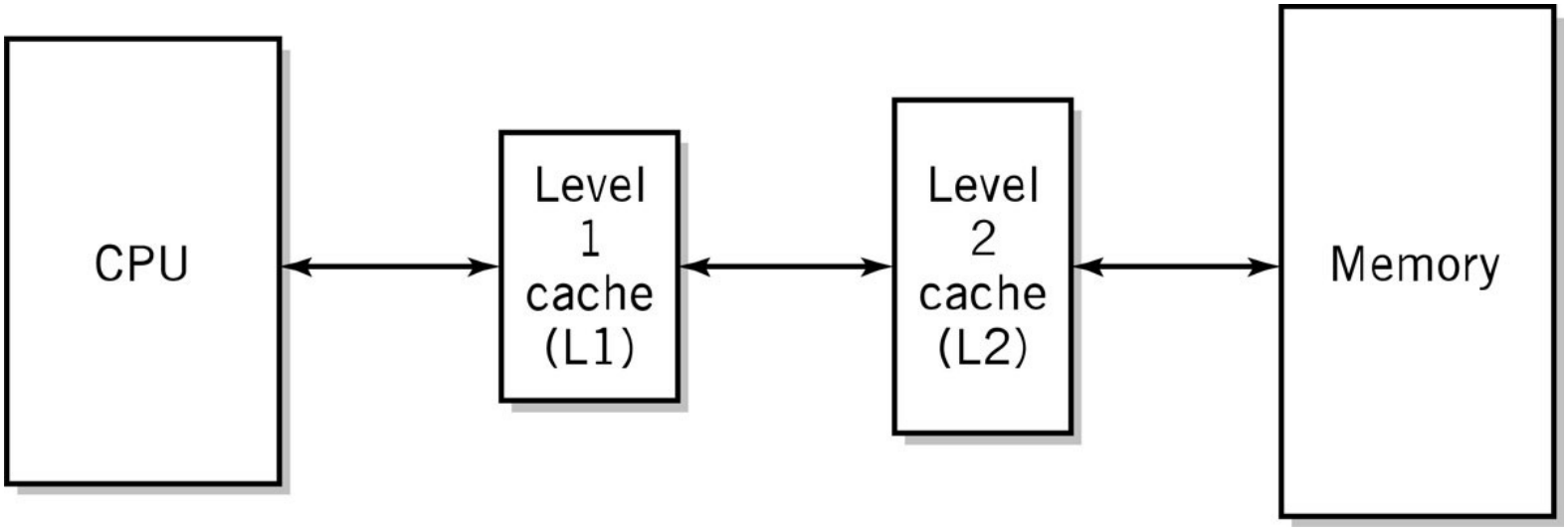


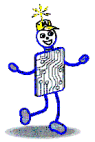
Performance Advantages

- Hit ratios of 90% common
- 50%+ improved execution speed
- **Locality of reference** is why caching works
  - Spatial and temporal locality: most memory references confined to small region of memory at any given time
  - Well-written program in small loop, procedure or function
    - Data likely in array
    - Variables stored together

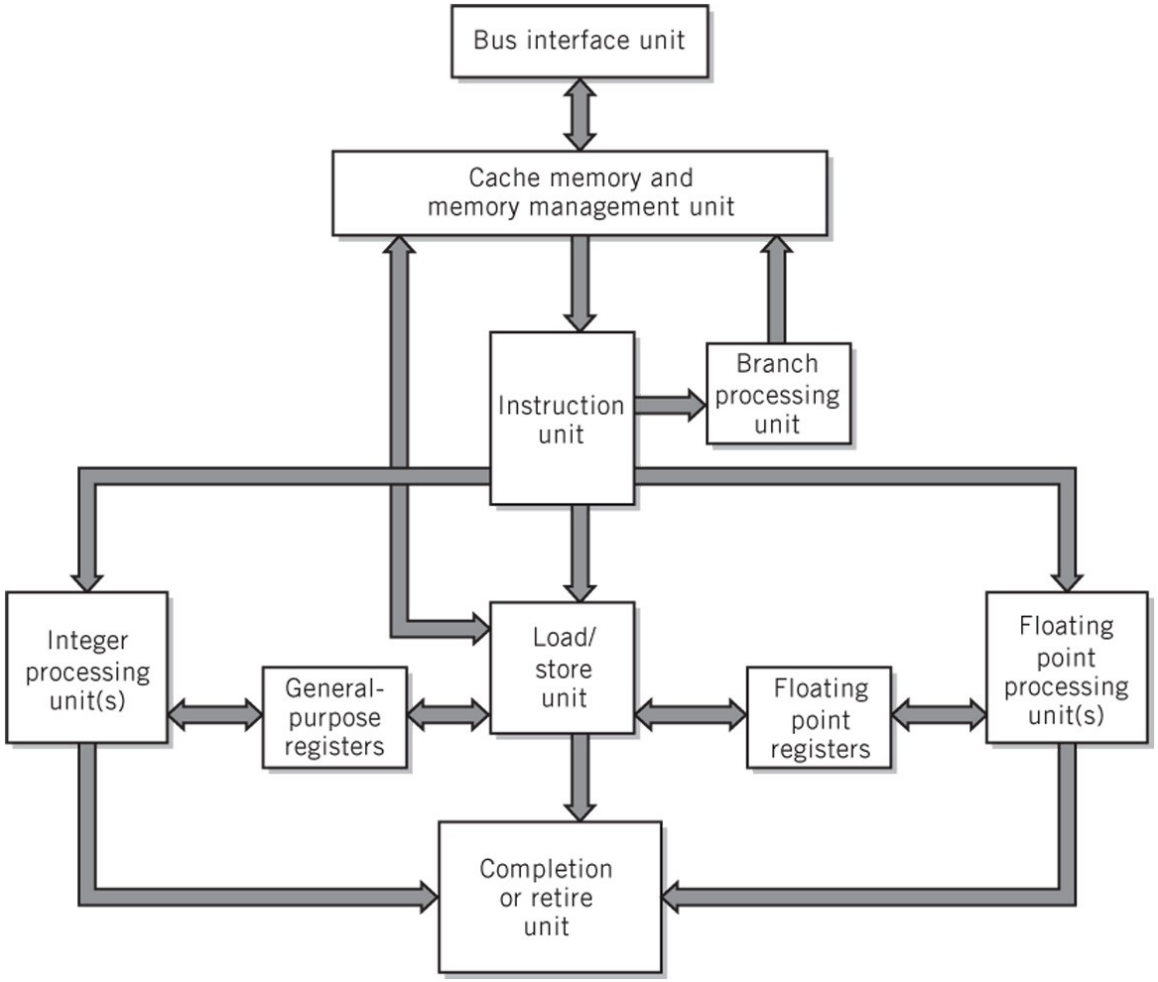
Two-level Caches

- L2 cache is bigger (and slower)
- Still faster than Memory



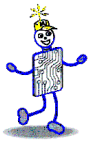


Modern CPU Block Diagram

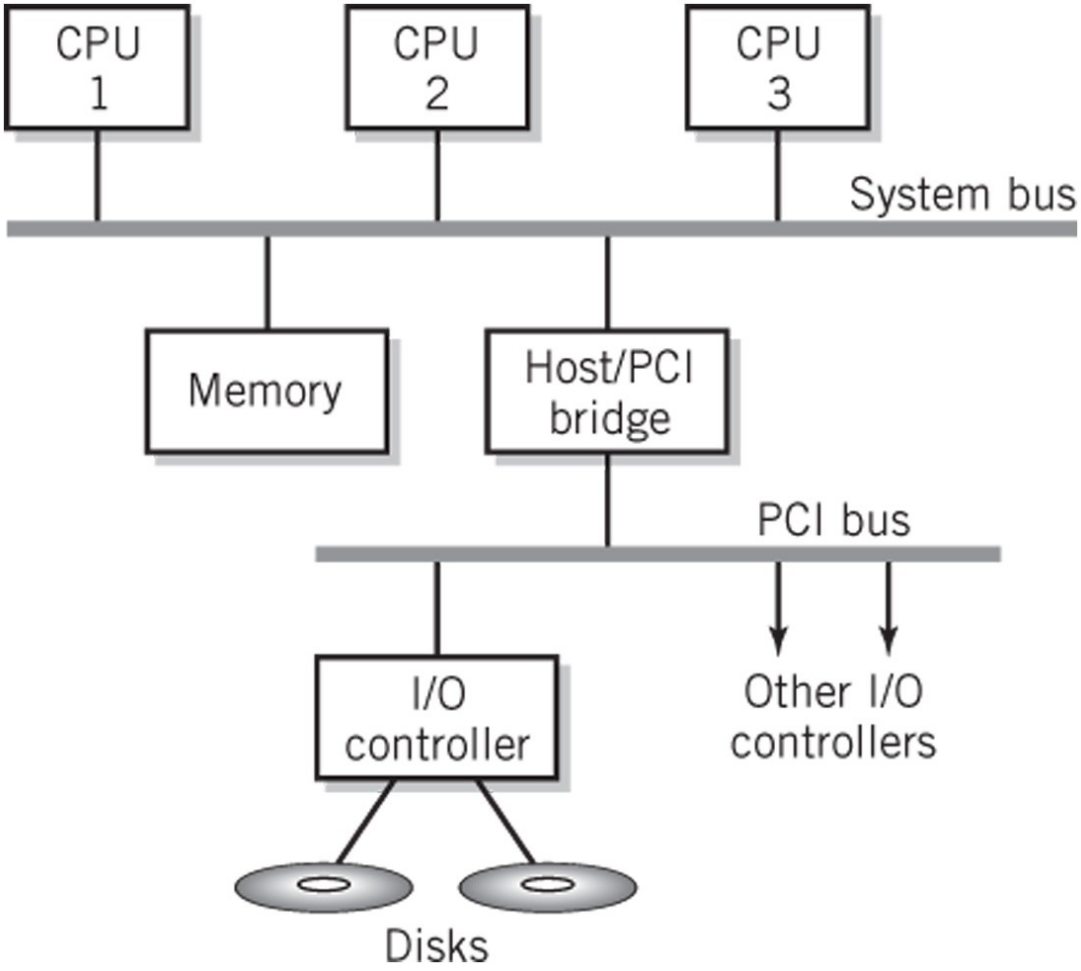


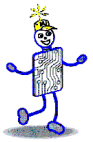
Multiprocessing

- Multiprocessor system
  - Multiple CPU within a computer, sharing some or all system memory and I/O
  - Multicore processors - when CPUs are on a single integrated circuit
- Reasons
  - Increase the processing power of a system
  - Parallel processing



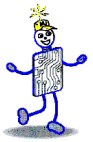
Typical Multiprocessing System Configuration





Multiprocessor Systems

- Advantages of multiprocessors
  - Different CPU may execute differents programs in a multi-tasking operating-system
  - Different CPU may cooperate on different threads of a single program
- Two ways to configure
  - Master-slave multiprocessing
  - Symmetrical multiprocessing (SMP)

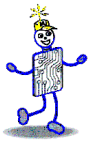


Symmetrical Multiprocessing

- Each CPU has equal access to resources
- Each CPU determines what to run using a standard algorithm
- Disadvantages
  - Resource conflicts – memory, i/o, etc.
  - Complex implementation
- Advantages
  - High reliability
  - Fault tolerant support is straightforward
  - Balanced workload

Master-Slave Multiprocessing

- Master CPU
  - Manages the system
  - Controls all resources and scheduling
  - Assigns tasks to slave CPUs
- Advantages
  - Simplicity
  - Protection of system and data
- Disadvantages
  - Master CPU becomes a bottleneck
  - Reliability issues – if master CPU fails entire system fails



Copyright 2013 John Wiley & Sons