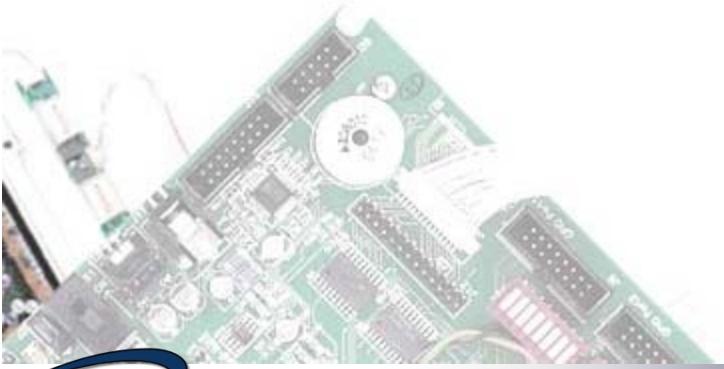
Basic Computing Concept and Instruction Level Parallelism

2010.11.29

Nam Eyee Hyun

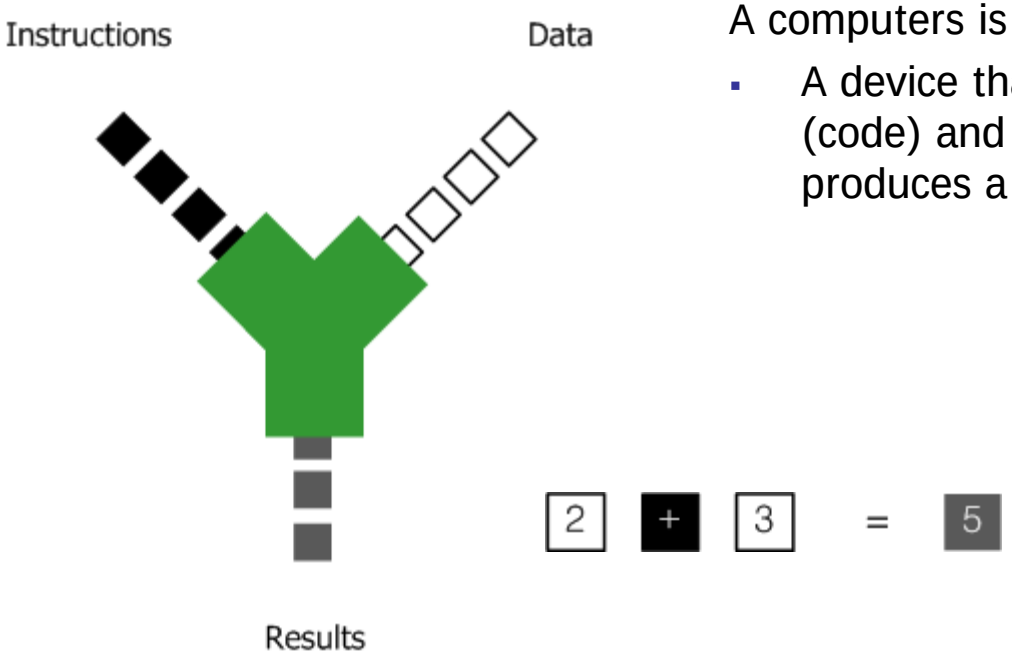


Contents

- Basic computing concepts
- Instruction level parallelism
- Hazards
- Considerations and summary

Basic model of computing

- The Calculator Model of Computing

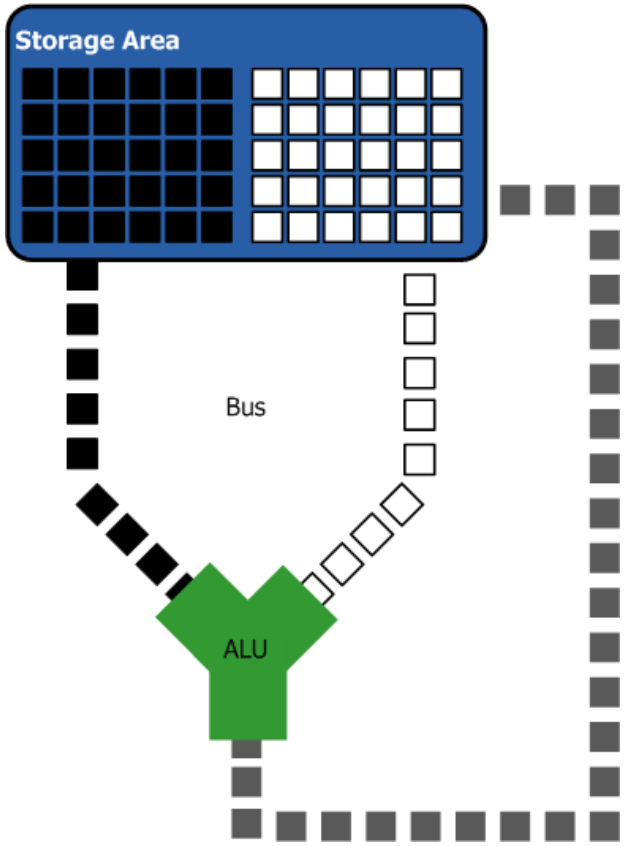


A computers is

- A device that takes a stream of instructions (code) and a stream of data(data) as input and produces a stream of results as output

Basic model of computing

- The File-Clerk Model of Computing



A computers is

- A device that reads, modifies and writes sequences of numbers according to the program

The program is

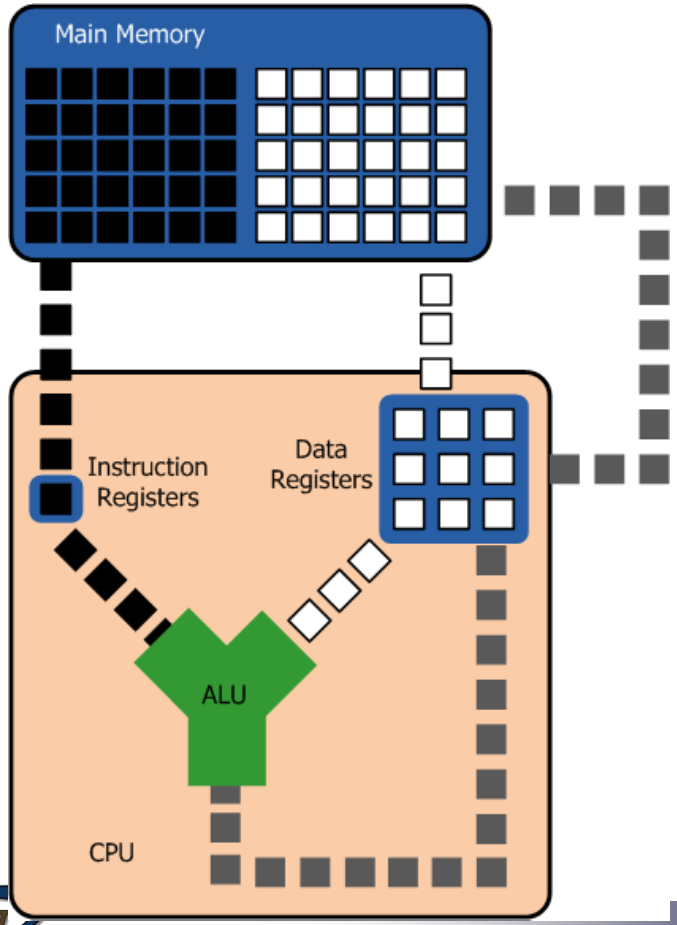
- An ordered sequences of instructions, stored in storage area

The stored-program computer

- Paradigm shift
  - from “manual input” to “automatic fetch”
- The modern computer’s ability to store and reuse prerecorded sequences of instructions make it fundamentally different from the simple calculating machine

Basic model of computing

- The File-Clerk Model of Computing



The register file

- Stores only a small subset of the data that the code stream needs
- Internal structure of a processor

The Main Memory

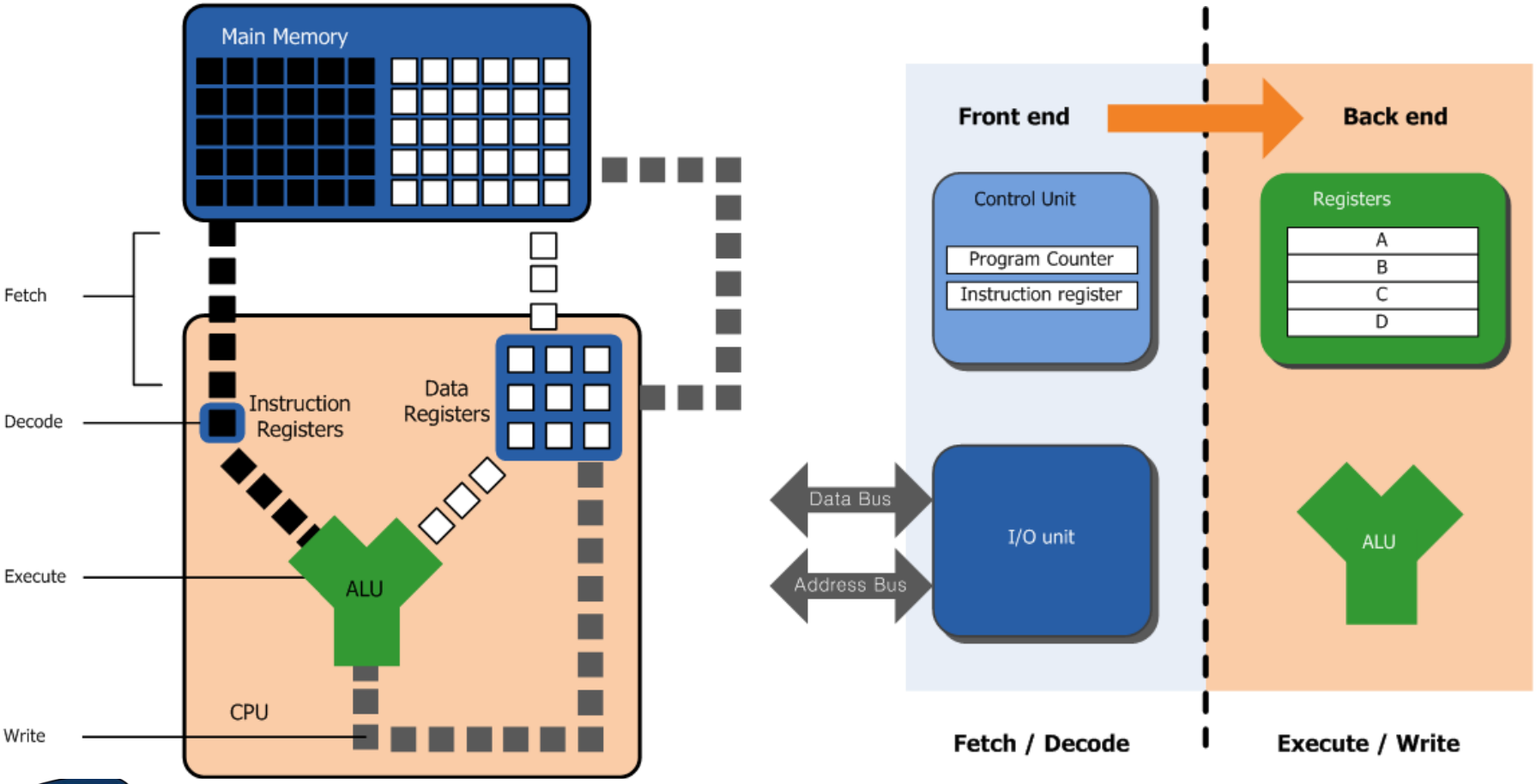
- Stores the data and the instruction set on which the computer operates
- External structure of a processor

The Instruction register and Program counter

- Stores instruction that fetched from memory
- Internal structure of a processor

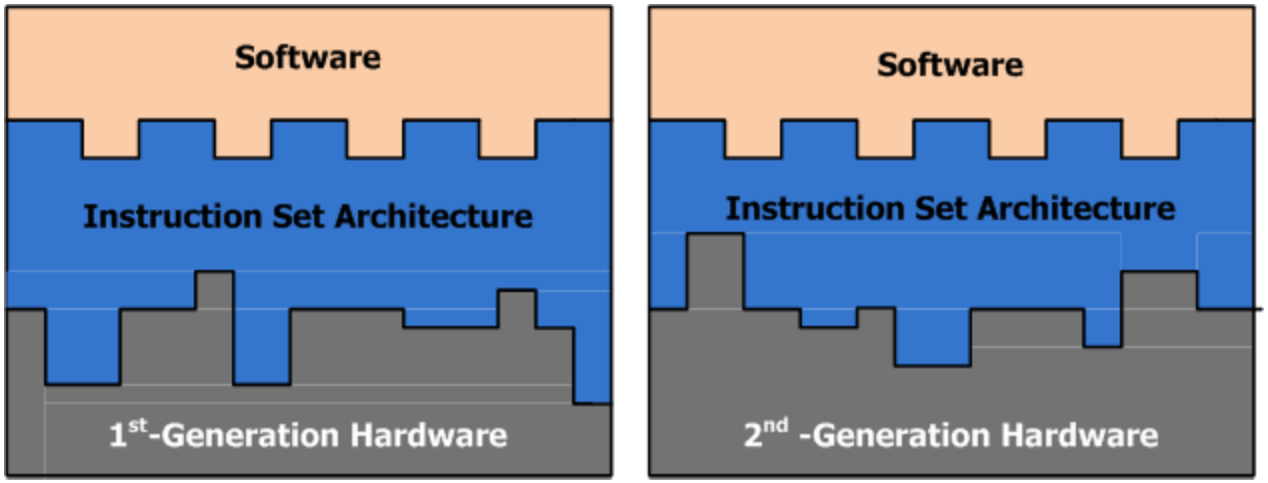
Basic model of computing

- The Lifecycle of an instruction



The programming model and the ISA

- Programming model
  - An abstract representation of the microprocessor that exposes to the programmer the microprocessor's functionality
- ISA
  - Programmer-centric combination of programming model and instruction set



Contents

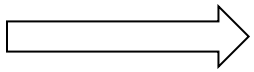
- Basic computing concepts
- Instruction level parallelism
- Hazards
- Considerations and summary

Instruction level parallelism

- ILP
  - A measure of how many of the instructions in a computer program can be performed simultaneously

- Example

1. e = a + b
2. f = c + d
3. g = e * f



ILP of 3 / 2

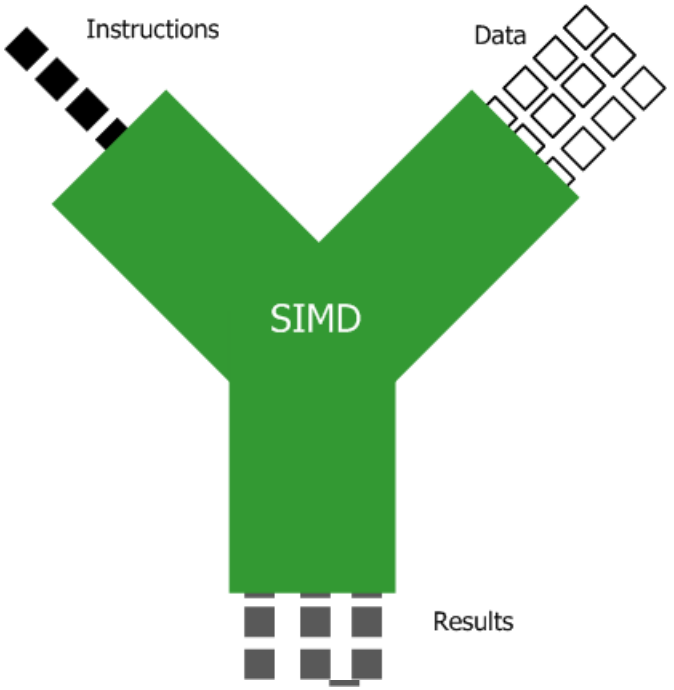
1 and 2 do not depend on any other operation
3 depends on 1 and 2

Instruction level parallelism

- Micro-architecture techniques used to exploit ILP
  - Pipelined execution
  - Superscalar execution
  - Out of order execution
  - Speculative execution
  - Branch prediction
  - VLIW

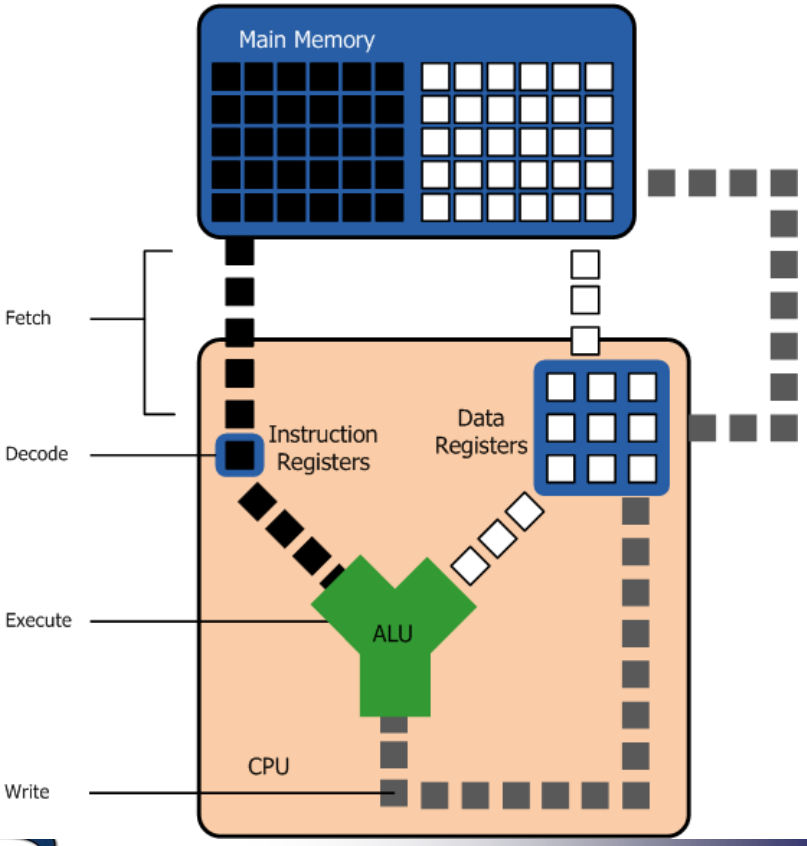
Data level parallelism (vector computing)

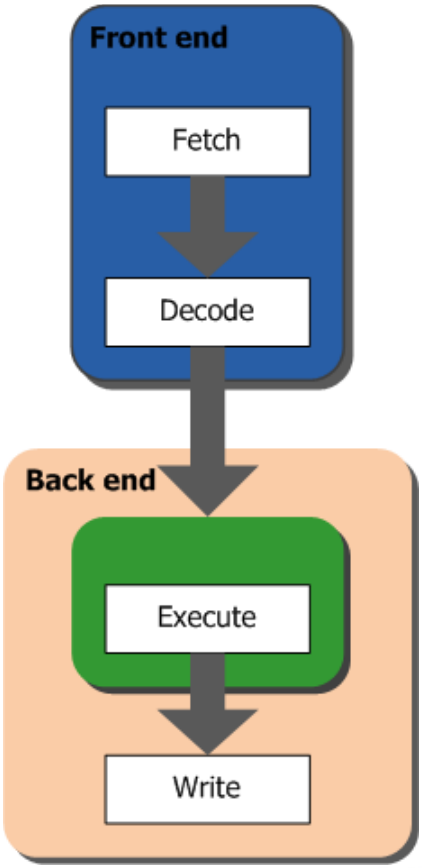
- A brief overview of vector computing
  - SIMD machine
    - Exploits data parallelism
    - Small chunks of code apply the same sequence of operation to every elements of a large dataset
  - Data parallelism
    - Is present in a dataset when its elements processed in parallel and out of order
  - Application
    - Image processing, streaming media, 3D rendering ..



Pipelined execution

- “Partially overlapped, in-order execution of multiple instructions in a lock-step manner”





Pipelined execution

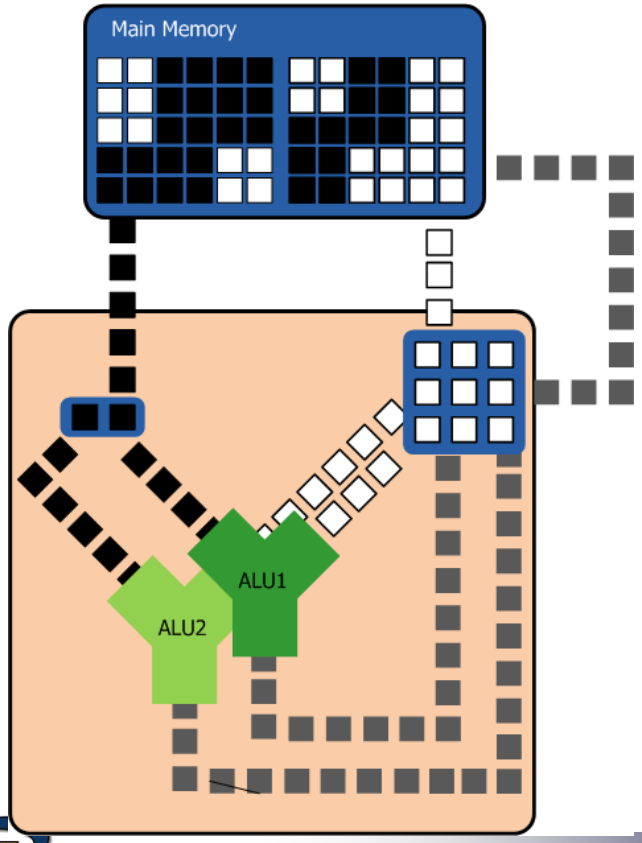
- Performance improvement
  - Decreases clock cycle time by inserting flip flops between pieces of logic, thereby decreases CPU time
    - It is important to balance the length of pipeline stages
  - Decreases CPI by using various techniques such as forwarding and branch prediction, thereby alleviates the effect of various hazard

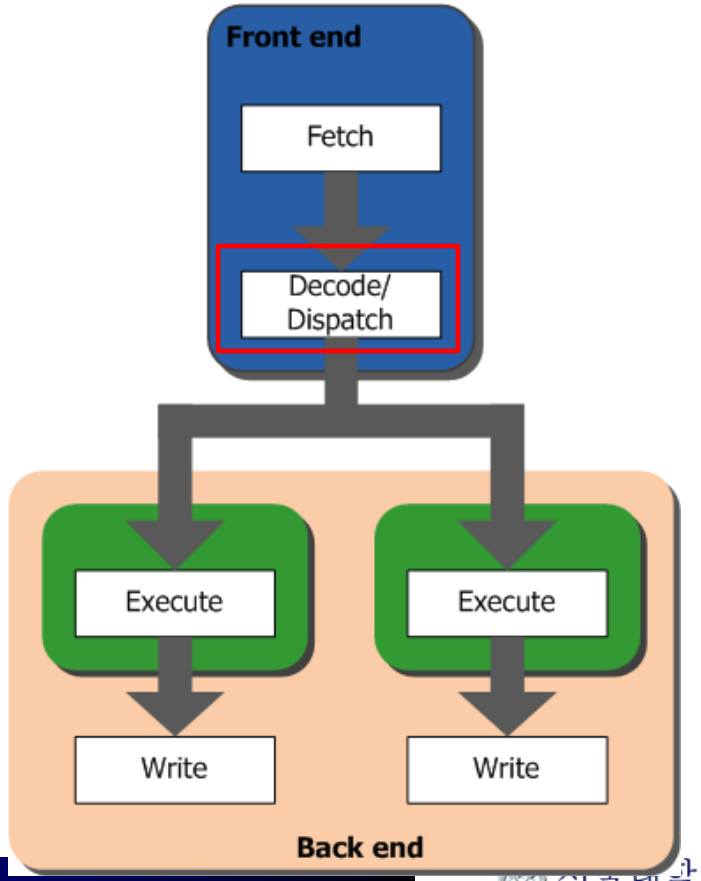
Pipelined execution

- Limitations
  - In-order execution
    - Various type of hazard and memory access introduce pipeline stall therefore increase CPI
    - Pipelined execution cannot effectively avoid or utilize such latency factors due to its in-order execution semantic
  - Partially overlapped execution
    - Pipelining requires a nontrivial amount of extra logic to implement and this overhead cost increases with pipelining depth
  - Lock-step manner
    - Front end and back end are tightly coupled with each other

Superscalar execution

- “Statically scheduled parallel execution of multiple instructions in a lock-step manner”





Superscalar execution

- Features
  - Multiple instruction fetch and decode
    - Fetch and decode stages must be able to fetch and decode multiple instruction on each cock cycle
  - Static scheduling
    - Dispatch stage determines whether or not multiple instructions can be executed in parallel *based on a set of fixed rules*
    - The illusion of sequential execution machine must be maintained for the sake of the programmer

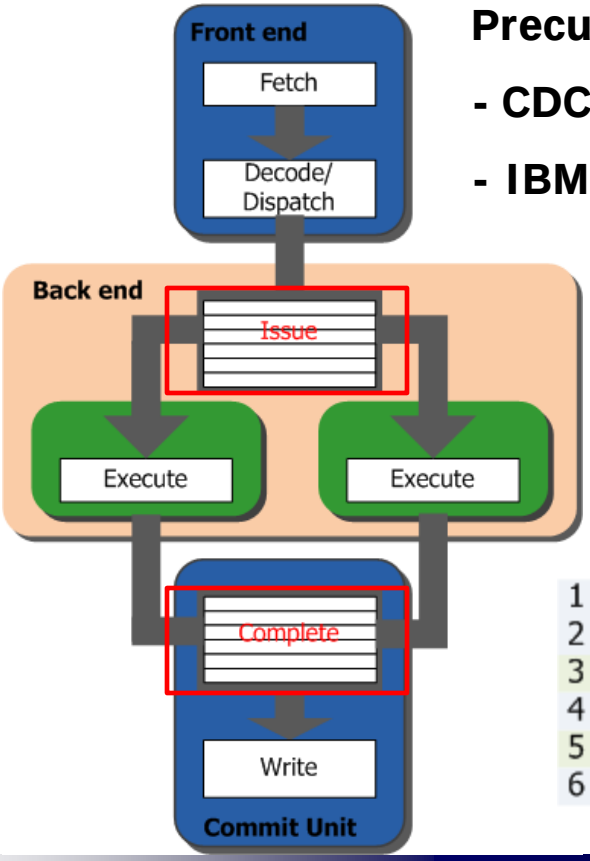
- Performance improvement
  - Decreases CPI by dispatching multiple instruction and executing them parallel, thereby decreases CPU time
    - From “CPI” to “IPC”

Superscalar execution

- Limitations
  - Static scheduling
    - Adapts poorly to the dynamic and ever-changing code stream
    - Makes poor use of wider superscalar hardware
  - Parallel but not out of order execution
    - Various type of hazard and memory access introduce pipeline stall therefore increase CPI
    - Pipelined execution cannot effectively avoid or utilize such latency factors due to its in-order execution semantic
    - The complexity and time cost of the dispatcher and associated dependency checking logic increases with # of ALU
  - Lock-step manner
    - Front end and back end are tightly coupled with each other

Out of order execution

- "Out of order execution of multiple instructions in a decoupled manner"



Precursors

- CDC 6600 Scoreboard

- IBM 360/91 Tomasulo's Algorithm

1 Fetch	In order
2 Decode/Dispatch	In order
3 Issue	Reorder
4 Execute	Out of order
5 Complete	Reorder
6 Write-back(Commit)	In order

Out of order execution

- Dynamic scheduling
  - Goal
    - Executing instructions in the most optimal order
      - Specifically utilizing unavoidable latencies
  - Constraints
    - Maintaining the illusion of sequential execution
  - Key components
    - Issue buffer (Reservation station)
    - Complete buffer (Reorder buffer)
    - Dynamic scheduling logic

Out of order execution

- Dynamic scheduling
  - Mechanism
    - Issue phase
      - Examine the instructions considering the state of the processor and the available resources in issue buffer
      - Reorder the code stream in issue buffer
      - Issue instructions from the buffer to the execution units
    - Complete phase
      - Instruction that have finished executing wait in complete buffer to have its result written back to the register file
      - Reorder the code stream in completion buffer
    - Commit phase
      - Write the instruction's result back to register file and alter the programmer-visible machine state permanently

Out of order execution

- Decoupled architecture
  - Decoupling the front end from the back end
    - Issue buffer(The Reservation station) decouple the front end's fetch/decode bandwidth from the back end's execution bandwidth
  - Decoupling the back end from the commit phase
    - Complete buffer(The Reorder buffer) resolves issues such as branch misprediction and exceptions/traps

Contents

- Basic computing concepts
- Instruction level parallelism
- Hazards
- Considerations and summary

Hazard

- Assumption upon sequential execution model
  - Programs are written based on the assumption that instructions execute one after the other, atomically and in the order specified by the program including exceptions
  - Processors must guarantee this sequential execution semantic whatever parallel jobs they do beneath the ISA

Hazard

- Definition
  - A hazard is a potential problem that can happen in a overlapped execution processor, which refers to the possibility of erroneous computation when a CPU tries to simultaneously execute multiple instructions

- Types
  - Data hazards
  - Structural hazards
  - Control hazards

Data hazard

- Overview
  - Data hazards occur when the same data is accessed in different stages of a pipeline
  - Ignoring data hazards can result in race condition

- Three situations
  - Read after Write (RAW) hazard
    - Flow (True) dependence
  - Write after Read (WAR) hazard
    - Anti dependence (False name dependence)
  - Write after Write (WAW) hazard
    - Output dependence (False name dependence)

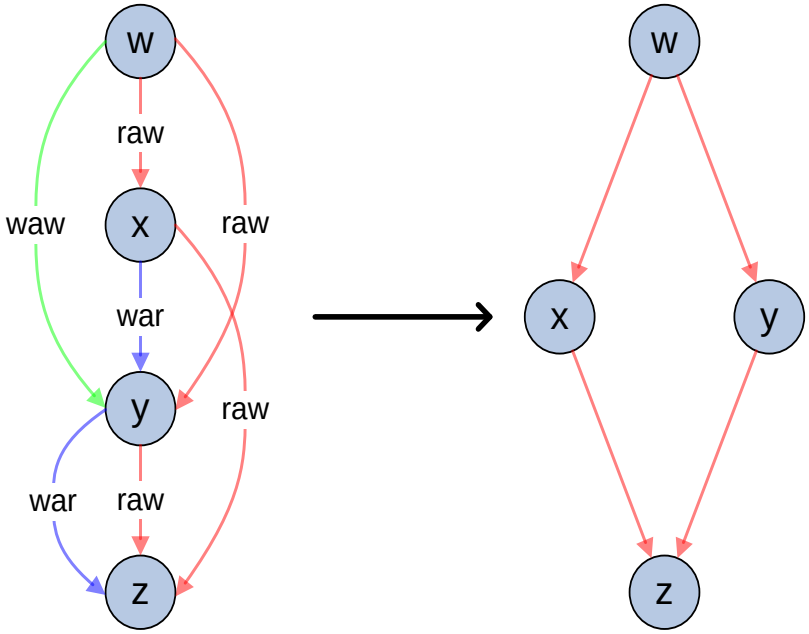
Data hazard

Flow dependence (RAW)

Anti-dependence (WAR)

Output dependence (WAW)

```
w :  R4 ← R0 + R8
x :  R2 ← R0 x R4
y :  R4 ← R4 + R8
z :  R8 ← R4 x R2
```



Structural & control hazard

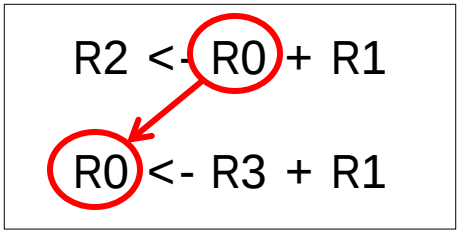
- Structural hazards
  - occurs when a part of processor's hardware is needed by two or more instructions at the same time
- Control hazards
  - arises when the processor arrives at a conditional branch and has to decide which instruction to fetch next
  - Incurs severe instruction load latency

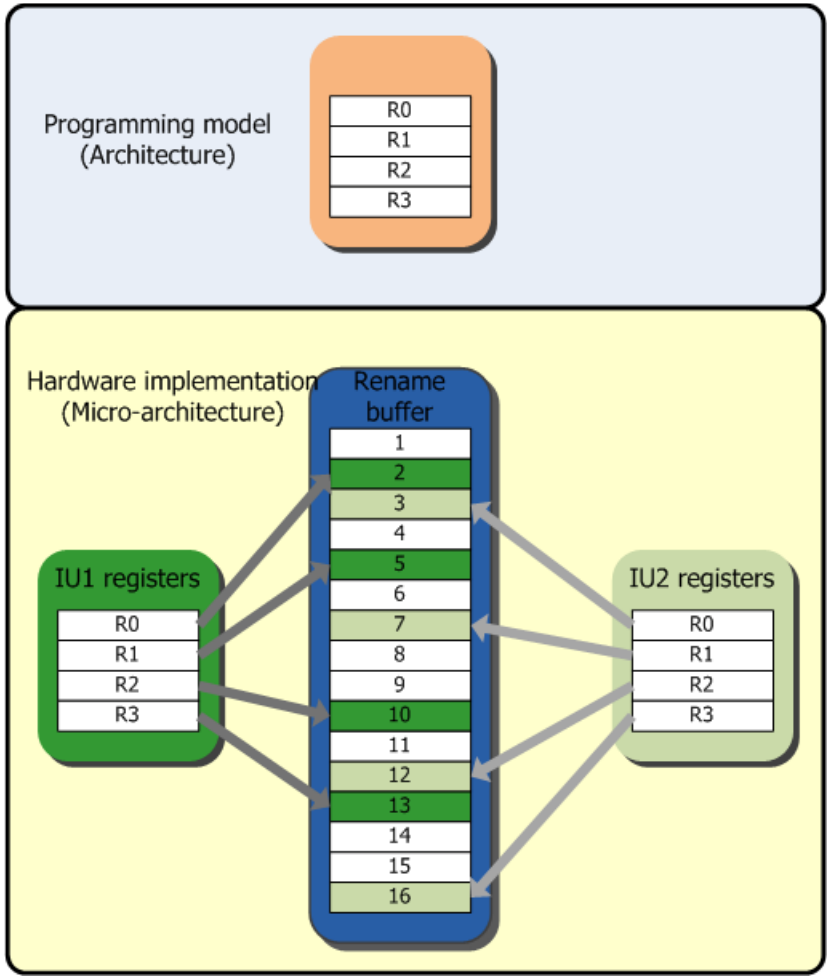
Hazard elimination

- Compiler techniques
  - NOP insertion
  - Instruction re-ordering
- Processor techniques
  - Primitive solution for all hazards
    - Bubbling the pipeline
  - Data hazard
    - Forwarding
    - Register renaming
  - Structural hazard
    - Adding more resources
  - Control hazard
    - Branch prediction

Register renaming

- A technique that used to avoid unnecessary serialization of program operations imposed by the reuse of registers by those operations





Contents

- Basic computing concepts
- Instruction level parallelism
- Hazards
- Considerations and summary

Considerations on ILP

- Motivation of ILP changes
  - From overcoming the limitations imposed by a small register file in IBM 360 in 1960
  - To narrowing the growing speed disparity between processor and memory access

- Limitations of ILP
  - Essentially difficult to tolerate all the cache miss penalty
  - Complexity and the latency of the underlying structures results
    - Reduced operating frequency ➔ further reducing ILP benefits

Recent trends

- Paradigm shift
  - From “complex and single processor”
  - To “Relatively simple and multiple processors”
- Headed towards higher level parallelism
  - Multi-threading (SMT)
  - Multi-processing (SMP)

Summary

- A computer is
  - A device that reads, modifies and writes sequences of numbers according to the program

	Key concepts	Scheduling Mechanism	Main approach for Performance Enhancement	Relation between Frontend & Backend
Pipelined execution	In-order, partially overlapped execution	X	Increasing bandwidth	Lock step
Superscalar execution	In-order, parallel execution	Static scheduling	Increasing bandwidth	Lock step
Out of order execution	Out of order, parallel execution	Dynamic scheduling	Exploiting latency	Decoupled

Summary

- Out-of-order execution processors
  - Requires no special compiler
  - Schedules instructions to multiple functional units
  - Resolves false dependencies while maintaining true dependencies