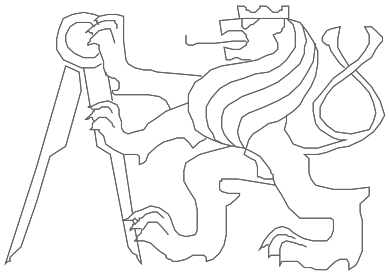
Advanced Computer Architectures

02

Superscalar organization - Introduction (Diversified dynamic pipelines)

Michal Stepanovsky



Czech Technical University in Prague, Faculty of Electrical Engineering
Slides authors: Michal Štepanovský, update Pavel Píša

Scalar pipeline

- Limitations of scalar pipelines:
  - The maximal throughput is bounded by 1 IPC
  - Unification – all instruction types into one pipeline => inefficient design.. (i.e., memory phase even for ALU ops.)
  - The stalling leads to waste cycles in previous pipeline stages which induces unnecessary pipeline bubbles (rigid pipeline) when these unused slots move through rest of pipeline

Program throughput:

$W = 1 / T = IPC.f_{CLK} / IC$

- A deeper pipeline has fewer logic gate levels in each pipeline stage (which leads to a shorter cycle time), but… (hardware overhead, instruction hazards, …)

Scalar pipeline – problem of unification

Problem of unification

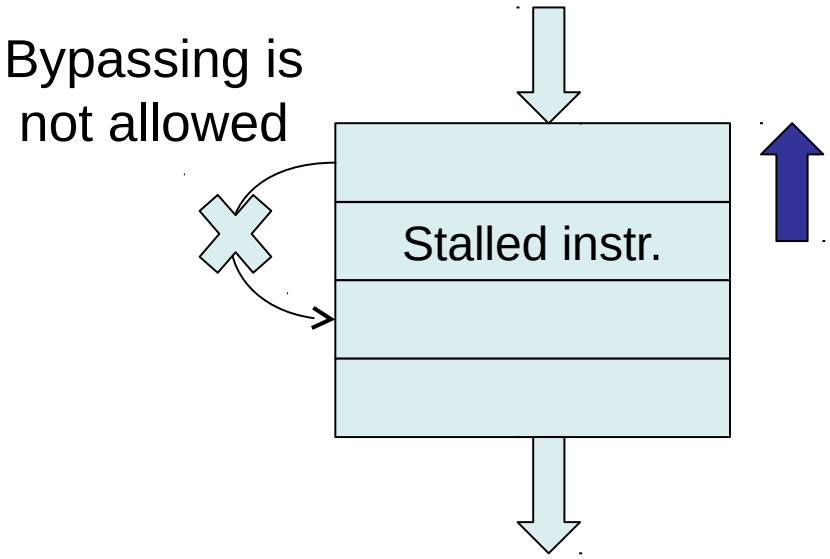
- Different instruction types require different hardware resources and processing. It is difficult and simultaneously inefficient implement all requirements in the unified pipeline.
- Different latencies: Some instruction types (i.e., floating-point instructions, multiply and divide) require multiple/more execution cycles/stages to be processed. Instructions that require long and possibly variable latencies are difficult to unify with simple instructions that require only a single cycle latency.

Scalar pipeline

- Requirements to stall pipeline for some instruction sequences

  the pipeline is not adaptable
  - scalar pipelines are "rigid" – instructions advance through pipeline stages one after the other in program order (lockstep fashion) and the need of stall in a certain stage is "spread" to the previous stages

div **R0**, R1, R2
sub R3, **R0**, R4 - blocked
add R5, R6, R7 - blocked



Solution – superscalar pipeline

- The natural solution is to create a parallel pipeline
- Superscalar pipelines – descendants of scalar pipelines, but not only parallel pipelines, but **diversified** parallel pipelines!
- Each parallel branch of the pipeline can be specialized to perform another function. Sometimes some of the function pipelines are doubled.

Superscalar organization

Superscalar pipeline:

- Multiple functional units in parallel (allows to process simultaneously multiple instructions across pipeline stages)
- Significant hardware resources are needed for implementing parallel pipelines
- **Out-of-order** execution is an important feature, instructions are executed in the order that differs to original program – **dynamic pipelines**
- Multiple instructions can be initiated in the same clock
- Combination of spatial and temporal parallelism
- **The width (*w*)** is equal to a number of parallel pipeline branches (number of instructions which can be fetched, decoded or completed in every cycle) – which can lead to a potential speedup of *w* over the scalar pipeline

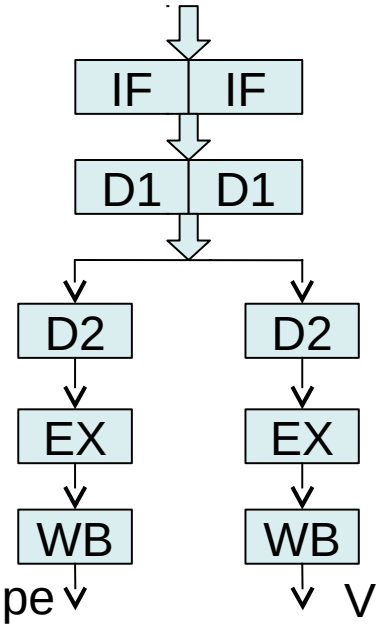
Superscalar organization

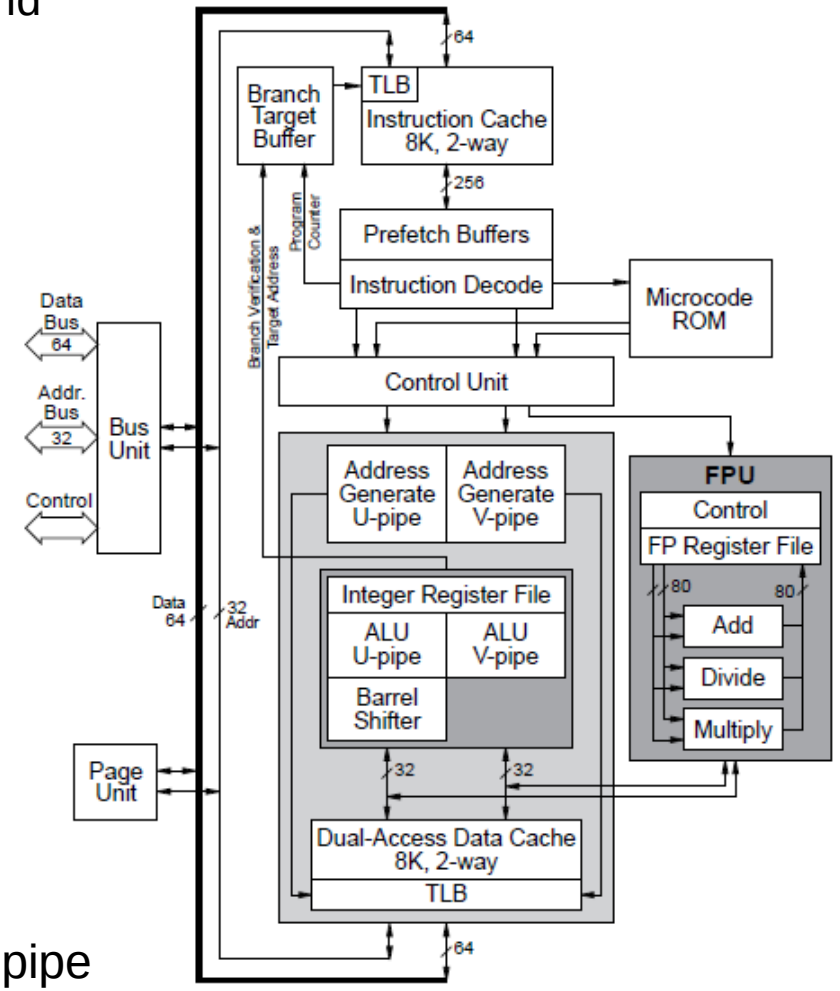
- Implementation requires additional hardware resources
- Each pipeline stage can potentially process and advance up to *w* instructions in every cycle.

The logic complexity in every pipeline stage can increase by a factor of *w*. Circuicity for interconnections can increase by a factor of w^2. The number of read and write ports of register file has to be also increased. Similarly, additional I-cache and D-cache ports…

5-stage pipeline in Intel Pentium (*w*=2): (two i486 pipelines)

Two ALU operations – both read from the register file. Two memory operations – dual access. But? (serialization – in case of banks conflict)





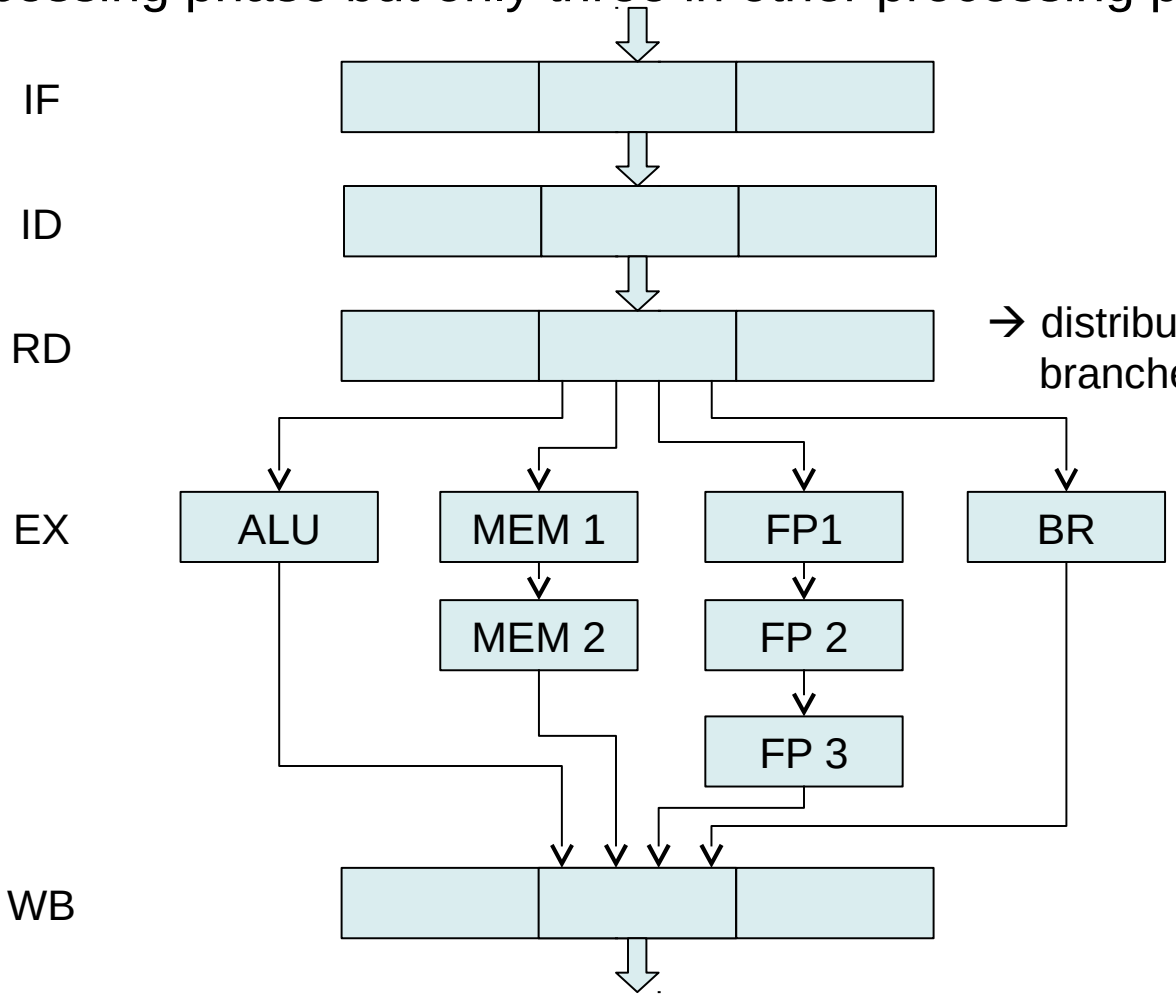
Brian Case: Intel Reveals Pentium Implementation Details, Vol. 7, No. 4, March 29, 1993

Superscalar organization – **diversified pipelining**

- The hardware required for different instruction types can vary significantly. For a scalar pipeline, all the diverse requirements must be unified into a single pipeline. But each instruction type requires only a related subset of scalar pipeline stage HW resources.
- Instead of implementing *w* identical pipes in a *w*-wide parallel pipeline, diversified execution pipes are typically implemented
- It results in:
  - Efficiency in HW resources,
  - Different latency in different pipeline branches (add vs. div – “faster” instructions can advance from shorter latencies)
  - Elimination of stall of independent instructions in other branches (**Independence** of parallel branches) – inter-instructions dependencies have to be solved before distribution/dispatch into parallel branches
  - Centralized control can be replaced by local independent and distributed control

Superscalar organization – **diversified pipelining**

A diversified parallel pipeline with four execution branches in the EX processing phase but only three in other processing phases (stages)

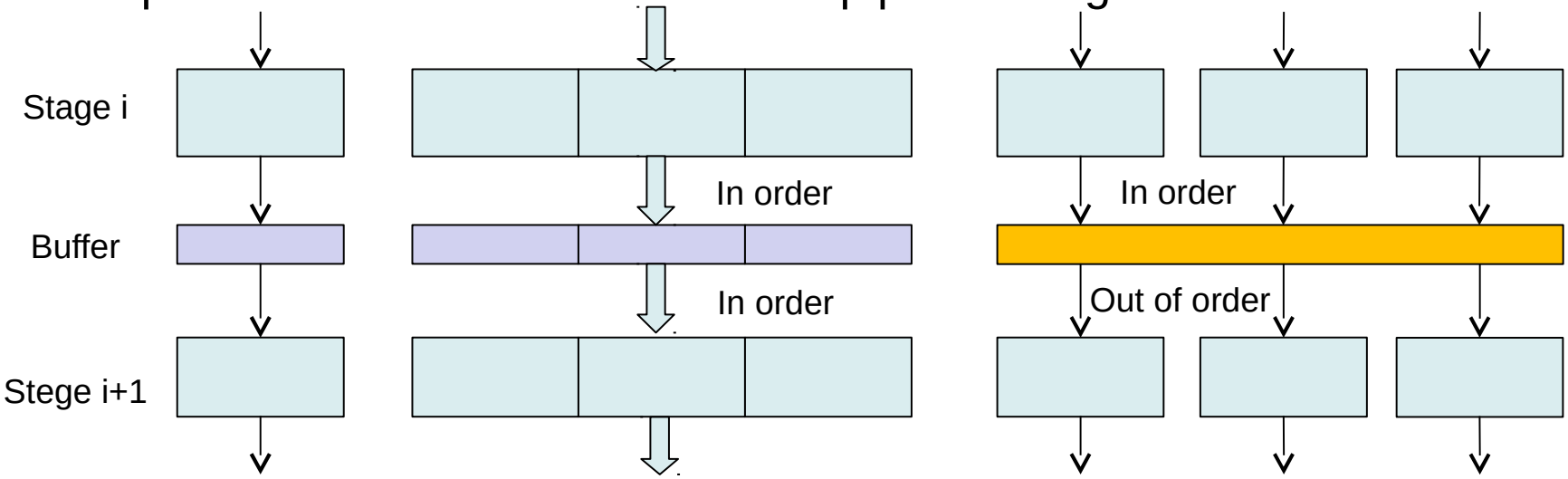


→ distribution of instructions into branches (**dispatching**)

Pipeline Width = 3

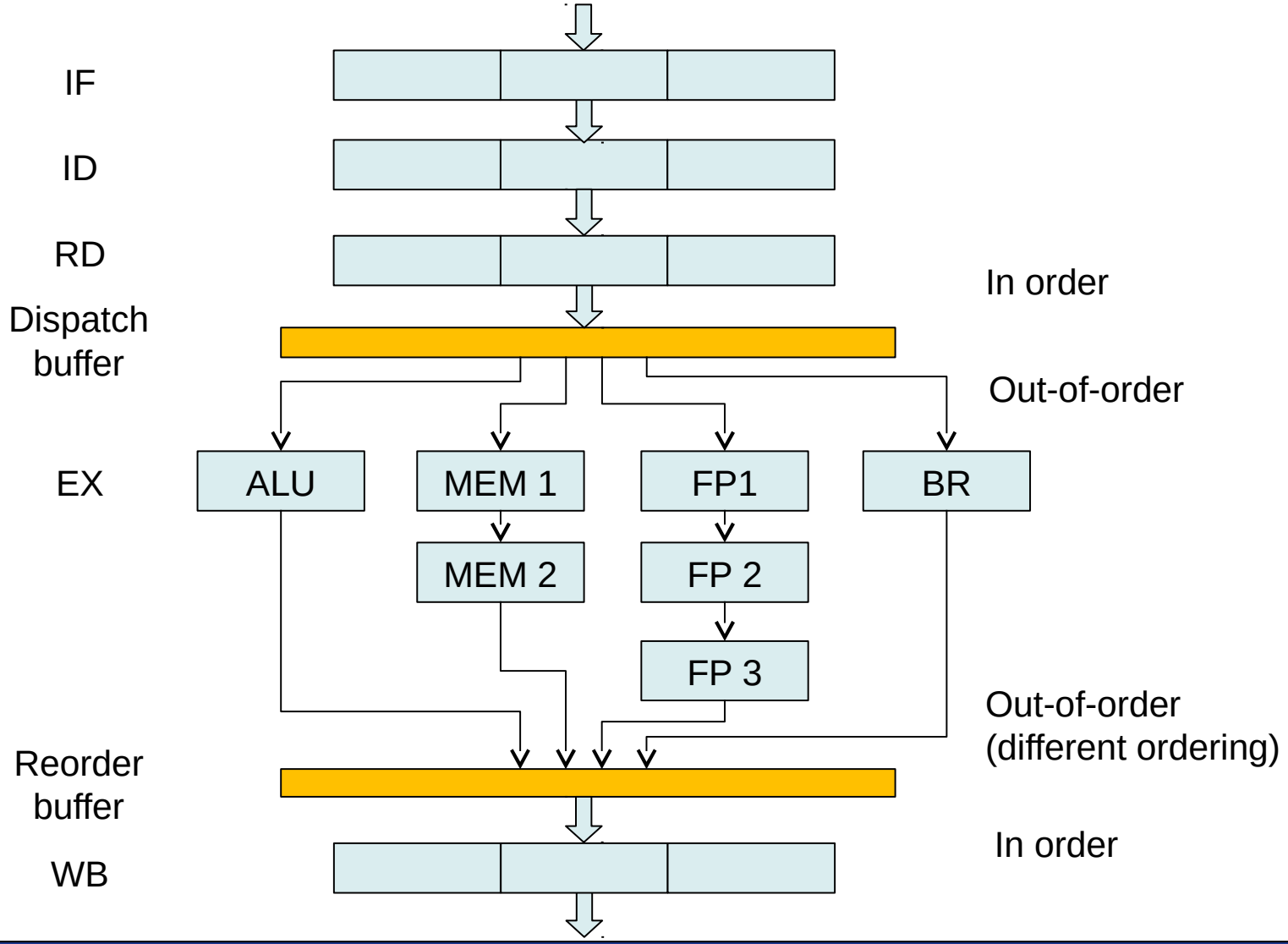
Superscalar organization – **dynamic pipelining**

- In any pipelined design (except asynchronous pipeline), buffers are required between two consecutive pipeline stages.



- **multi-entry buffers**: usually small multi-ported RAM, independent access (addressing) for write an read ports - an instruction can remain in an entry of the buffer for many machine cycles and can be updated or modified while resident in that buffer
- **dynamic pipelining** utilizes multi-entry buffers for **out-of-order** instruction execution, an instruction can be stored and dispatched from the buffer in a different order

Superscalar organization – **dynamic pipelining**



Superscalar organization – **dynamic pipelining**

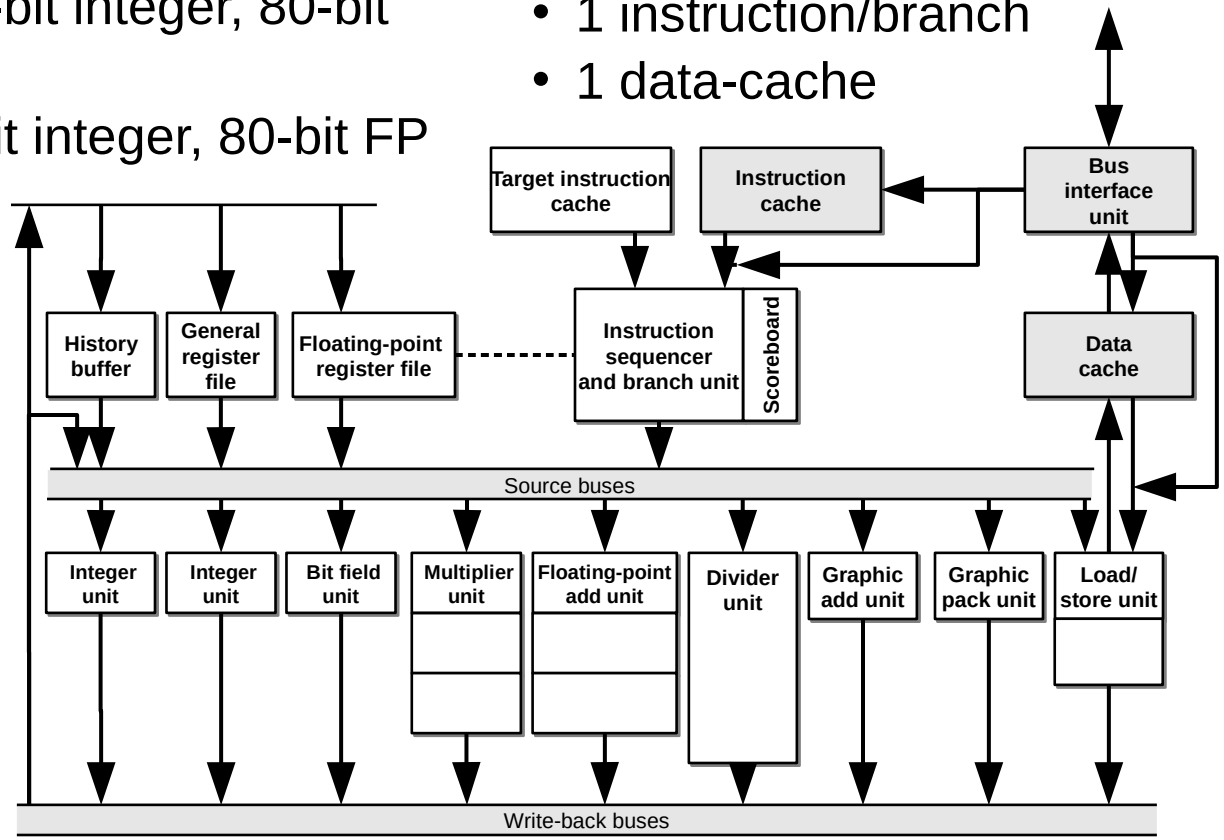
- **Dispatch buffer** – is loaded with decoded instructions according to program order and then they are dispatched to the functional units potentially in an order different from the program order

- Execution units (diversified pipeline) can exhibit different latencies

- **Completion buffer** (or **Reorder buffer**) – retires (finishes) the instructions in program order – required for precise exception support (updating the machine state according to the program order.)

Superscalar organization

- 2 integer ALUs (32-bit operands)
- 1 FP-add (80-bit operands)
- 1 multiply (64-bit integer, 80-bit FP operands)
- 1 divide (64-bit integer, 80-bit FP operands)
- 1 bit-field(32-bit operands)
- 2 graphics (64-bit operands)
- 1 instruction/branch
- 1 data-cache

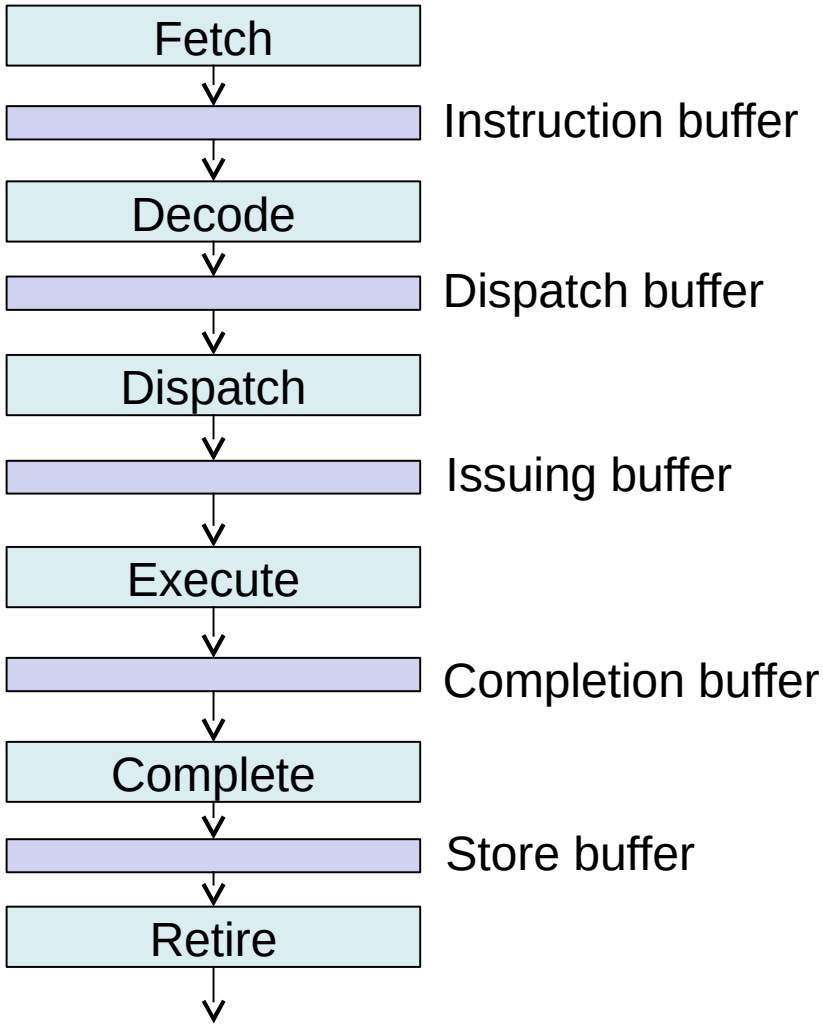


Motorola 88110, year 1992.

Shen, J.P., Lipasti, M.H.: Modern Processor Design : Fundamentals of Superscalar Processors, First Edition, New York, McGraw-Hill Inc., 2004

Superscalar organization – execution phases and problems

- Phases of instruction processing:
  - Fetch
  - Decode
  - Dispatch
  - Execute
  - Complete
  - Retire



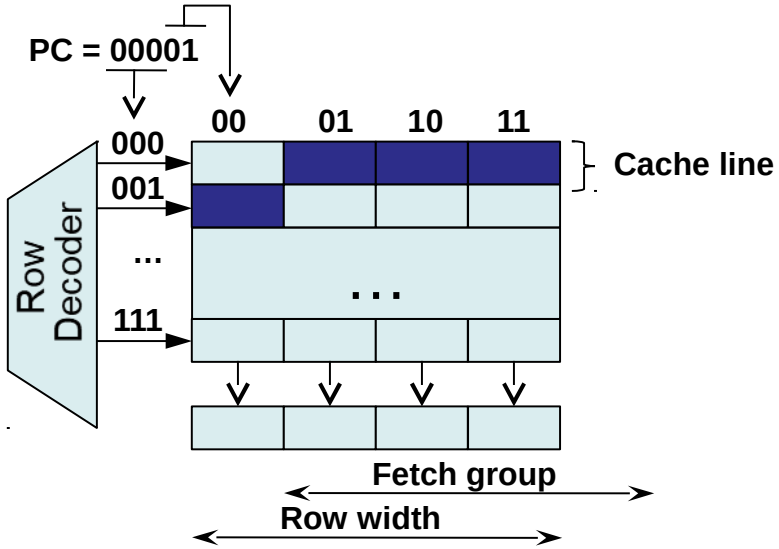
Superscalar organization – **Fetch**

- For superscalar pipelining of width *w*, fetch stage should be able to fetch *w* instructions from the I-cache in every machine cycle
- The program counter (PC, sometimes named instruction pointer IP) is used to fetch *w* instructions (Fetch group) in every cycle
- I-cache must be wide enough that each row can store w instructions and that an entire row can be accessed at one time (another option is to compose row from more simultaneously accessible blocks)
- The throughput of other stages cannot exceed the throughput of fetch stage – fetch throughput is degraded by unaligned instructions in I-cache and by branch/jump instructions (PC is changed) inside fetch group or targets to the middle of group

Superscalar organization – **Fetch of misaligned groups**

Solutions:

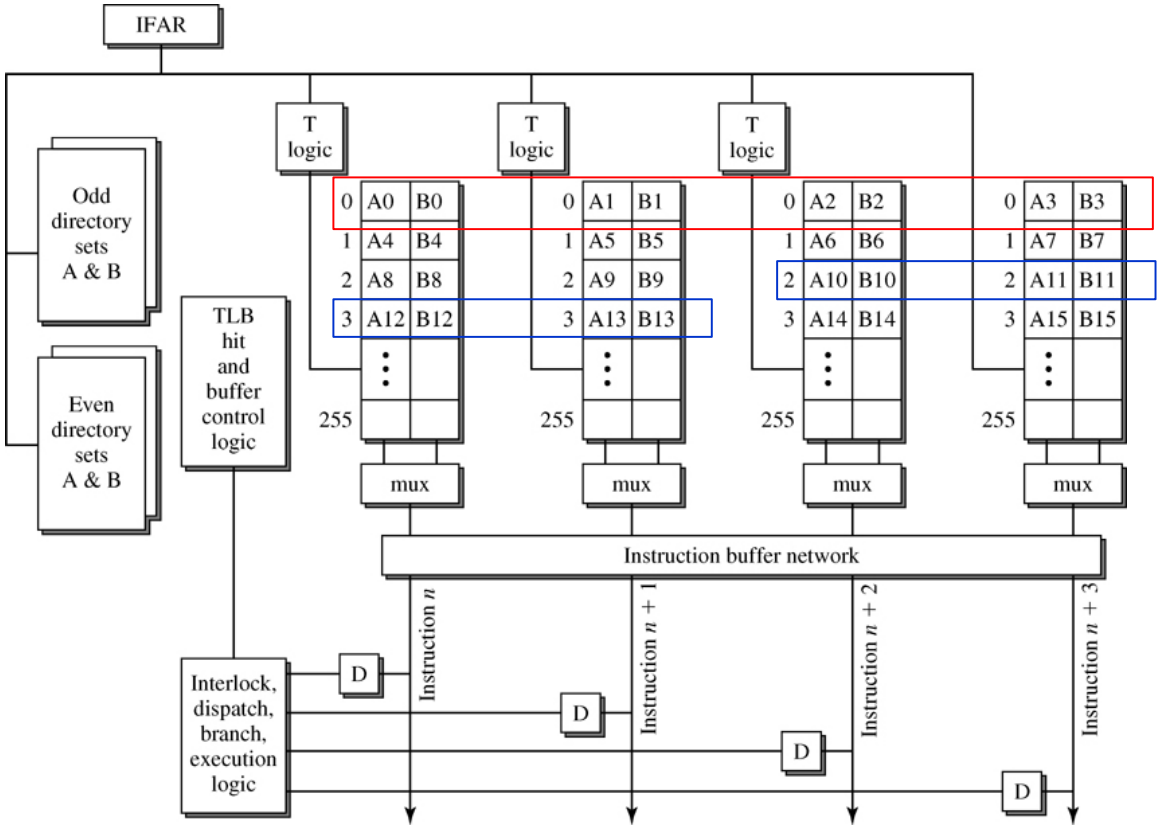
- Static – during compilation – the compiler receives I-cache parameters, chooses optimal instruction placement; target of branches are placed in proper memory locations
- Dynamic – by hardware at run time – modified internal organization of the cache …



Superscalar organization – **Fetch T-logic**

Dynamic: If fetch group span over row boundary, but not over line/block boundary, it is still possible to fetch the whole fetch group in one cycle … IBM RS/6000 (two-way set associative I-cache with Auto-Realignment)

Program:
Instr. A0
Instr. A1
Instr. A2
Instr. A3
Instr. A4
Instr. A5
…



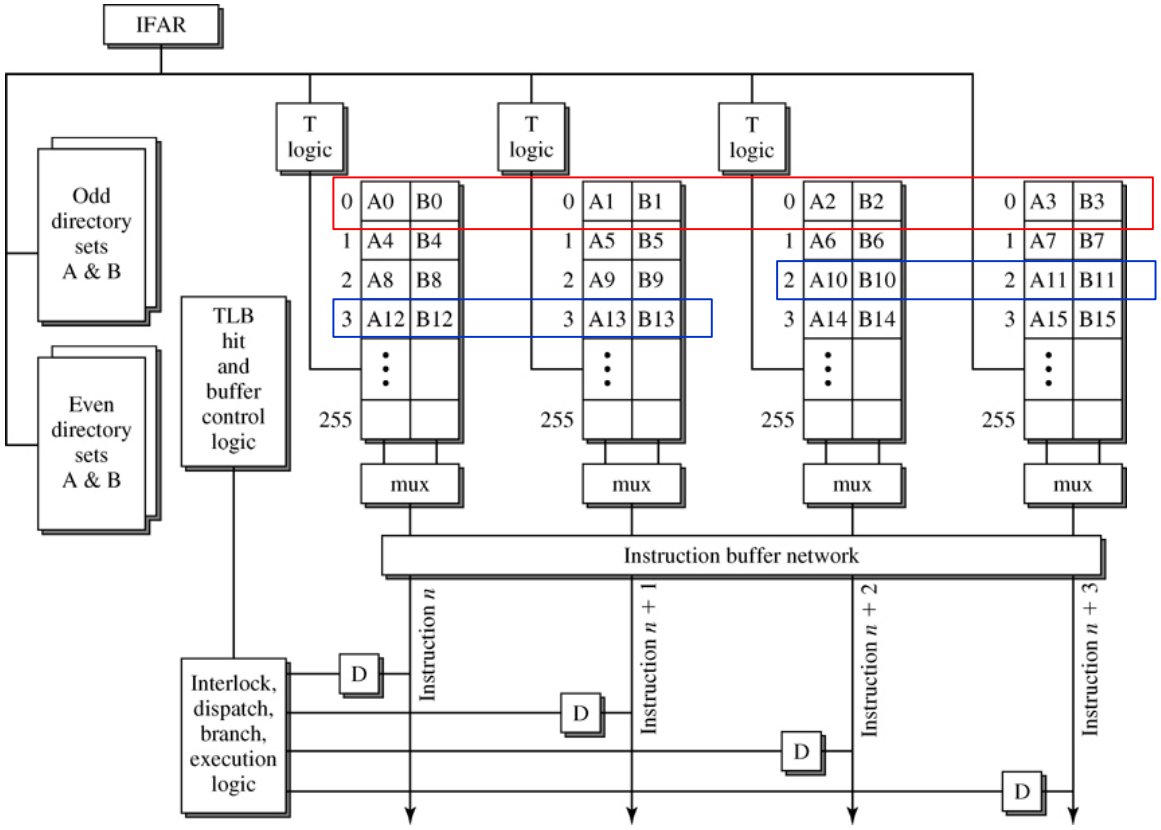
Line size: 16 instructions (64B), Fetch group 4 instructions

If PC addresses A0 then instructions A0,1,2,3 can be fetched without problems (they belong to same row)

If PC addresses A10, then instructions A10 and A11 belong to one row, A12 and A13 to the next one. The task of T-logic is to detect this case and increment row index for these instructions in respective cache sections.

Superscalar organization – **Fetch T-logic**

Dynamic: If fetch group span over row boundary, but not over line/block boundary, it is still possible to fetch the whole fetch group in one cycle …
IBM RS/6000 (two-way set associative I-cache with Auto-Realignment)

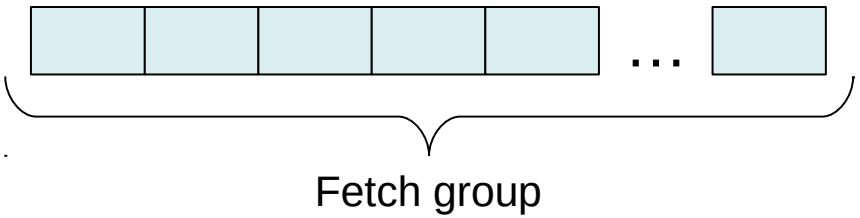


Line size: 16 instructions (64B), Fetch group 4 instructions

If line size boundary is not crossed then T-logic results in complete fetch group read in single cycle independently to PC address. If the boundary is crossed, then memory access can be necessary. For 16 instructions cache line size, 16 start offsets of the fetch group are possible. Average throughput with T-logic is (13*4+1*3+1*2+1)/16=3,63 instr/cycle, or 1,188 cycle/fetch group instead of (4*4+4*3+4*2+4)/16=2,5 instr/cycle (or 1,75 cycle/fetch group)

Superscalar organization – **Decoding**

- Complexity strongly influenced by pipeline width and ISA => RISC vs. CISC (instructions of different length with even 1 byte increment)
- Decode tasks: 1. identification of the individual instructions (even lengths for CISC), 2. determination of the instruction types, 3. detection of inter-instruction dependencies and determine set of independent instructions to dispatch to next stages
- fetch group – centralized decoding..



CISC – instruction identification
RISC – inter-instruction dependencies

- A large number of comparators are needed for determining register dependences between instructions
- Register files must be multi-ported to allow multiple simultaneous accesses
- Multiple buses – to route the accessed operands to their destination buffers

Superscalar organization – **Decoding**

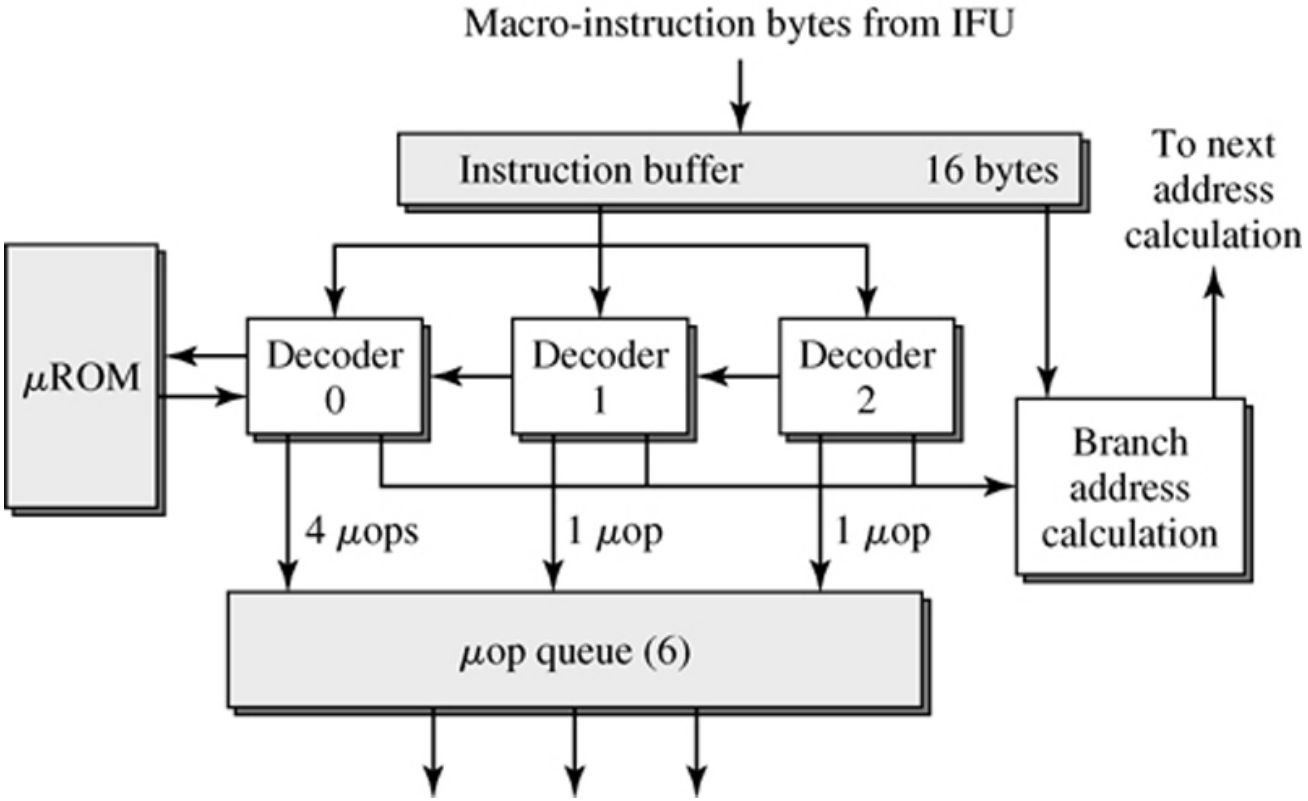
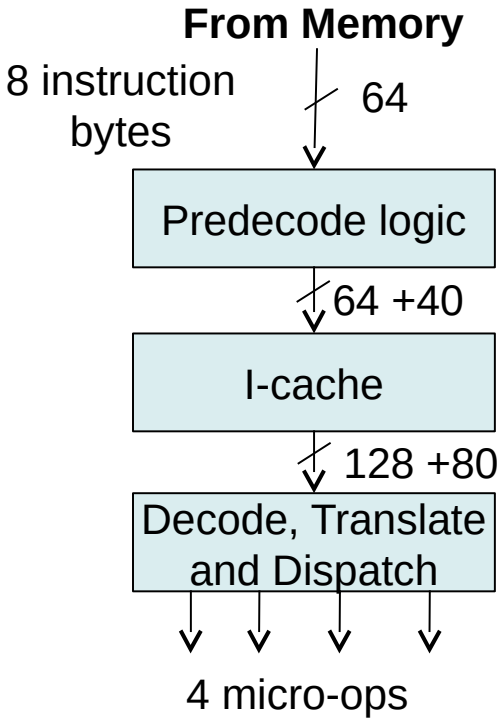


Fig. The Fetch/Decode Unit of the Intel P6 Superscalar Pipeline

Shen, J.P., Lipasti, M.H.: Modern Processor Design : Fundamentals of Superscalar Processors, First Edition, New York, McGraw-Hill Inc., 2004

Superscalar organization – **Decoding**

- Predecoding
  - I-cache miss – partial decoding of the cache block when it is transferred from the memory
  - Both CISC and RISC (PowerPC, MIPS R10000) – identification of branches, independent instructions, …



AMD K5

Additional bits contain information about identify
control-flow changing branch instructions within the fetch group, beginning (and end) of instructions (mainly for CISC), corresponding microinstructions count, a location of instruction operation code…
Advantages?
Disadvantages?

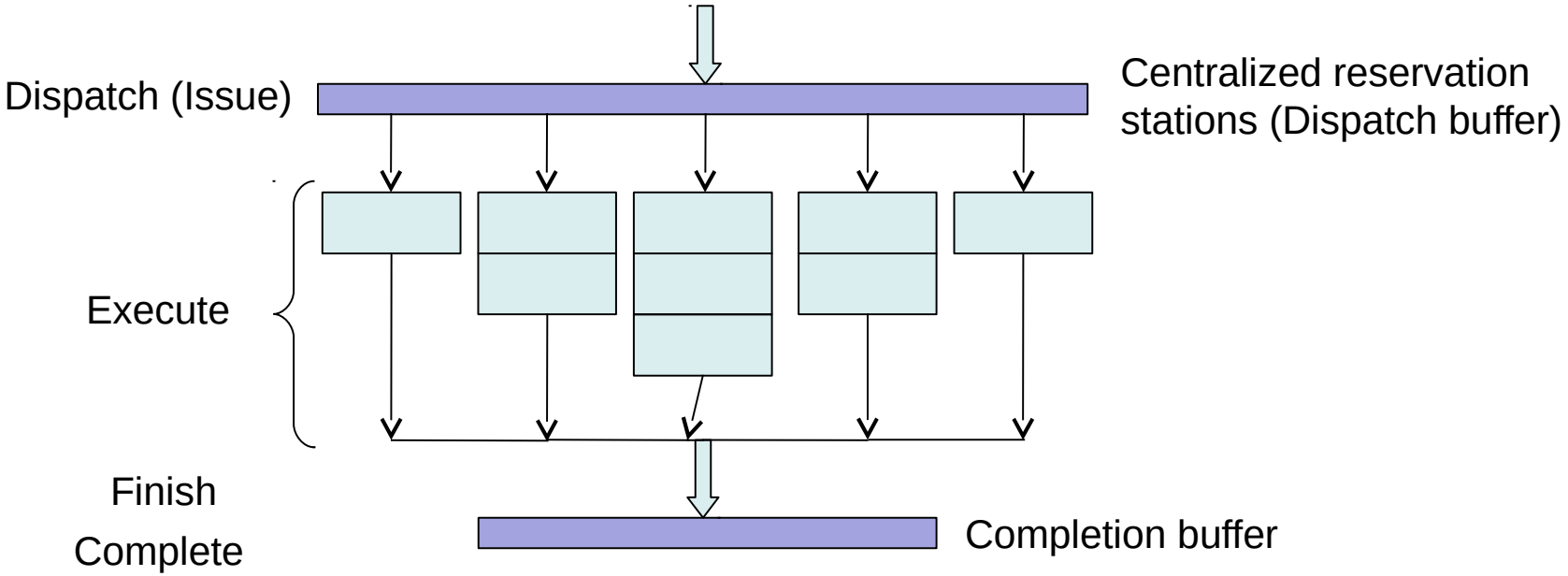
Predecode bits significantly simplify instruction decoding and allow the simultaneous decoding of multiple IA32 instructions.

Superscalar organization – **Dispatching**

- The primary goal of **dispatching** is to route instruction to the appropriate functional unit for execution (diversified pipelines: multiple heterogeneous functional units used according to instruction type)
- Operands values are not necessarily ready for some instructions. Solved by dispatch stall or by reservation stations (buffers) where instruction waits for previous instructions providing operands. Following instructions with all operands ready can be dispatched → **data flow** concept on the lowest level – **Tomasulo algorithm** and registers renaming to solve WAW hazards
- We distinguish:
  - Centralized reservation stations
  - Distributed reservation stations
  - Hybrid reservation stations (or clustered)
- Definition refinement:
  - dispatching - instruction association to the functional unit for execution
  - issuing - initializing execution in a functional unit

Superscalar organization – **Dispatching**

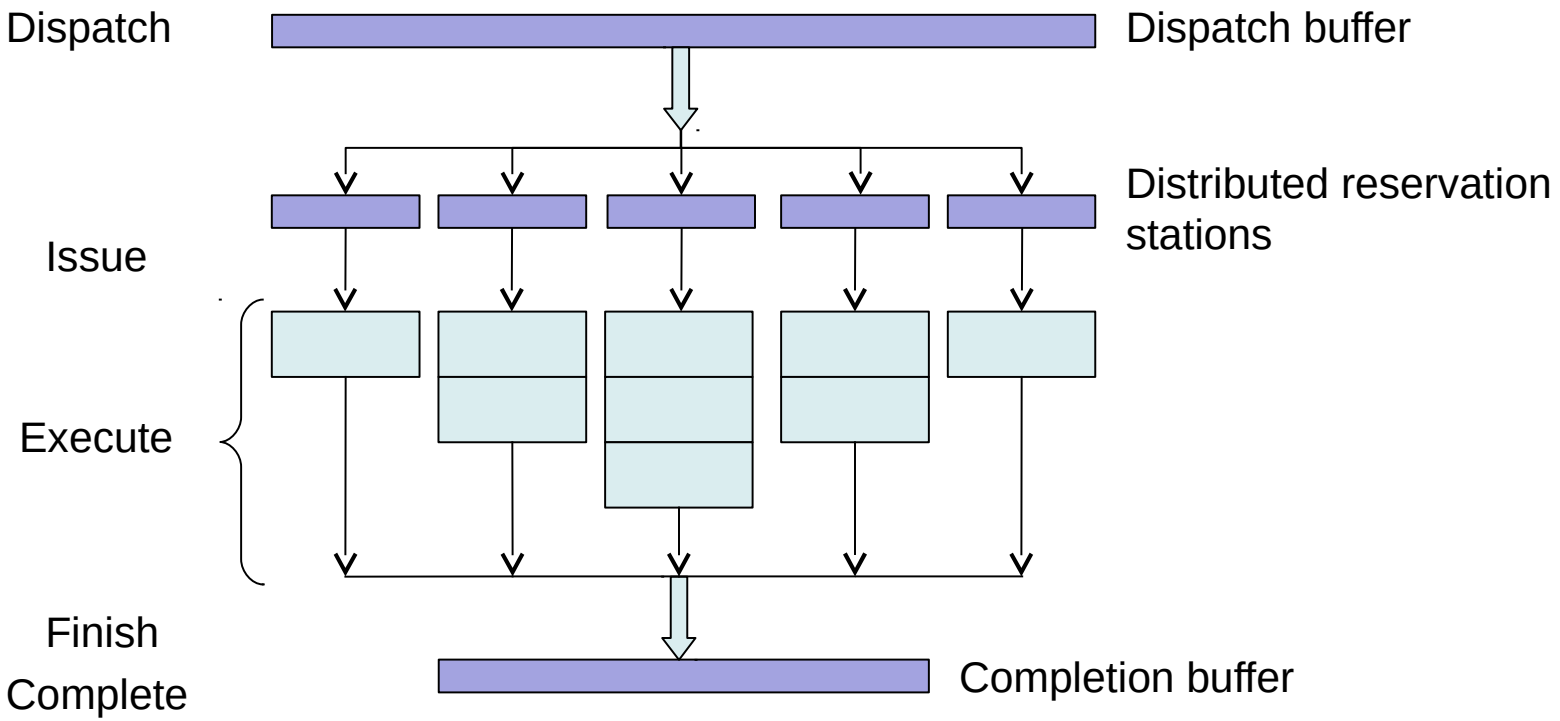
Centralized reservation stations:



Consumes more hardware resources, but effectively utilized

Superscalar organization – **Dispatching**

Distributed reservation stations:



Less HW resources, reservation stations with one write and one read port

Superscalar organization – **Execution**

- Current trend – more parallel and more diversified pipelines
  (more functional units and having these functional units be more specialized, older generation -- branch for integer operations and one for floating point)
- One pipeline branch – can process more operations on different operands
- Number of functional units (EX phase) exceeds pipeline width
- With an increased number of functional units demand on hardware resources increases as well

  - caused by need of forwarding from functional units outputs to their inputs, increased number of buses and match logic (mechanism of operands routing requires to solve new structural hazards), reservation stations need to monitor availability (ready state) of operands values (tag matching).
  Each waited for operand in station needs to monitor all buses where result can appear. Number of tags/ways of the bus corresponds to the number of ready results in a single cycle

Superscalar organization – **Execution**

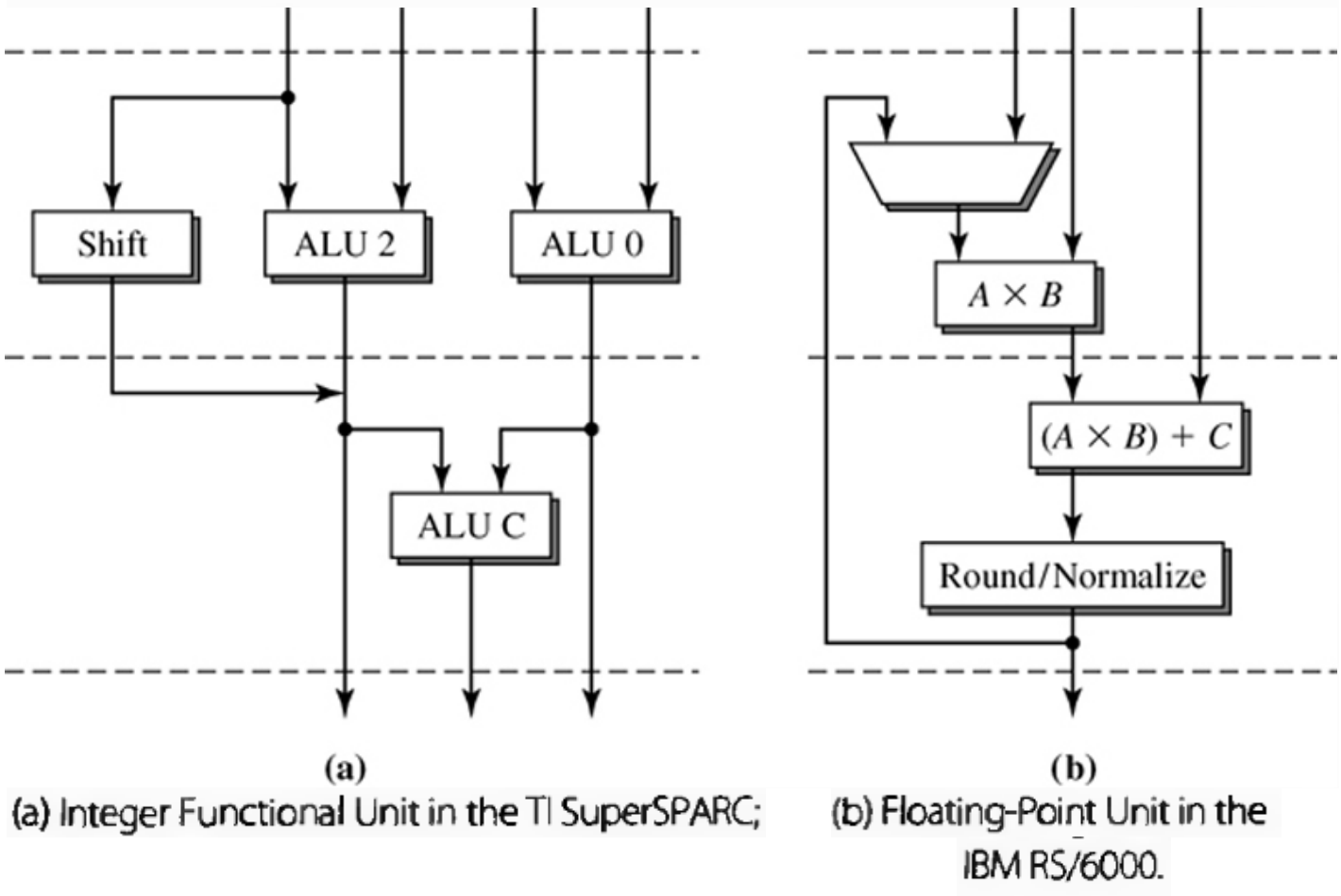
What is the best mix of functional units for a superscalar pipeline???

- It depends on application domain… and HW complexity…
  (For example, typical programs having 40% ALU instructions, 20% branches, and 40% load/store instructions would require 2:1:2 rule => two ALU units, 1 branch unit, 2 Load/Store units)

- The total number of functional units very often exceeds the actual width of the pipeline. **Why???**

- Bottleneck due to structural dependences (mismatch of instruction mix and functional unit mix) and a high level of diversification

Superscalar organization – **Execution**



(a) Integer Functional Unit in the TI SuperSPARC; (b) Floating-Point Unit in the IBM RS/6000.

Shen, J.P., Lipasti, M.H.: Modern Processor Design : Fundamentals of Superscalar Processors, First Edition, New York, McGraw-Hill Inc., 2004

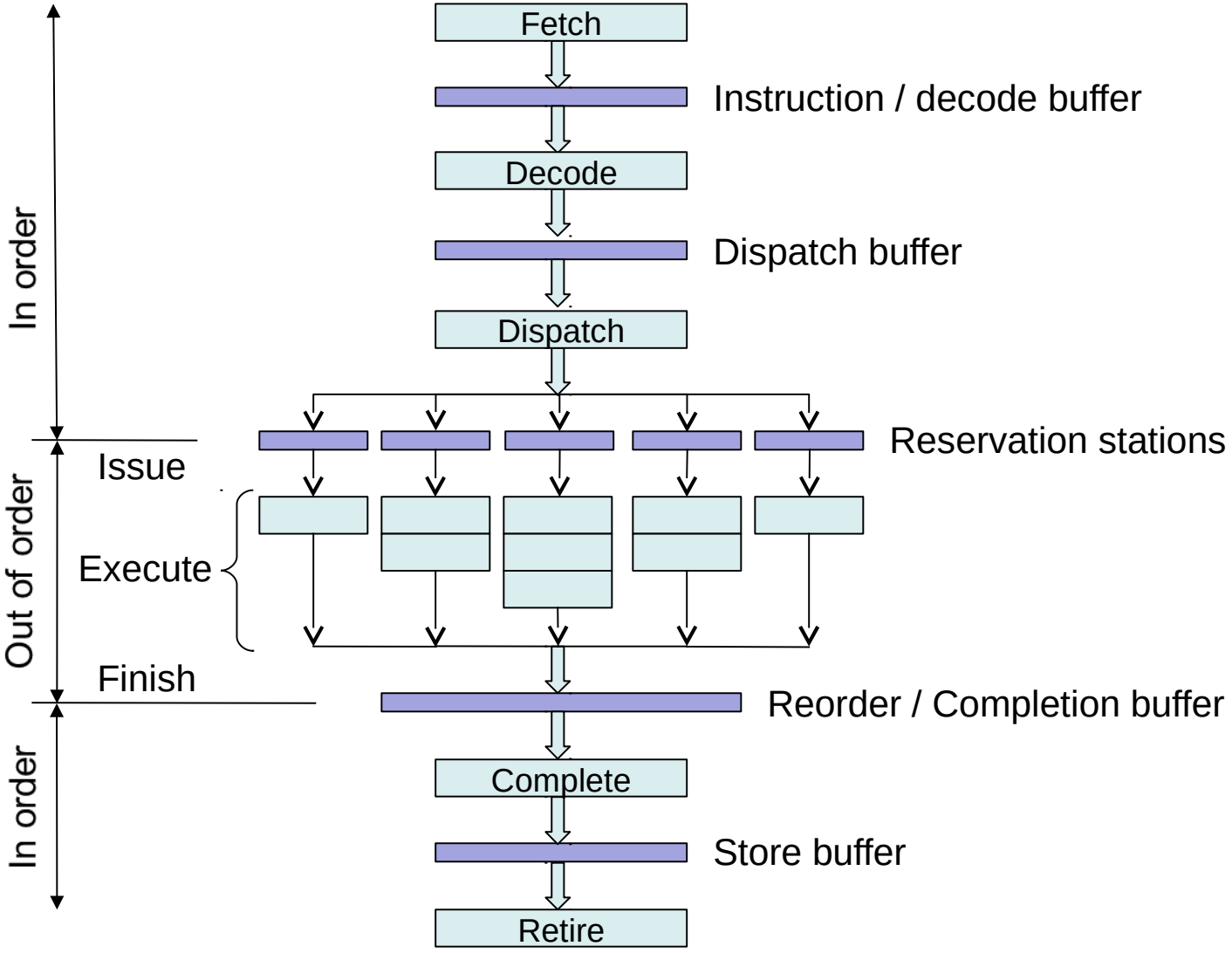
Superscalar organization – **Completion and Retiring**

- An instruction is considered **completed** when it finishes execution and updates the machine state. An instruction finishes execution when it exits the functional unit (execution phase) and enters the completion buffer.
- Additional cycles can be required before an instruction result is available in D-cache. The instruction is considered **retired** when it exits the store buffer and updates the D-cache.
- These terms (completed and retired) are sometimes used interchangeable ... (as well as dispatching and issuing, seldom the Completion and Retiring concepts are shifted to a higher level/phase, i.e., to Finishing and Completion)

Superscalar organization – **Completion and exceptions**

- Precise exception support requires reorder/completion buffer
- Instruction completion must occur in program order
- When an exception occurs (in functional units, ...), the excepting instruction is tagged in the completion buffer
- The completion stage checks each instruction before that instruction is completed
- When a tagged instruction is detected, it is not allowed to be completed before it is not the oldest instruction in the completion buffer. Then this instruction is not completed, and preceding processor state is saved. Following instructions and in progress operations in pipelines are discarded.

Superscalar organization – **Completion and Retiring**



Superscalar organization – Summary

- Out-of-order superscalar pipeline supports out-of-order execution (processing) of instructions only between **Dispatch** and **Complete**. Instructions remain in reservation stations for one or more cycles while waiting for their source operands. When operands are available, they are **issued** into the functional units (**data flow** principle). After execution (when leaving functional unit), the instruction enters into the reorder/completion buffer. Instructions leave the reorder/completion buffer in program order (completion) when all previous instructions are completed. The architectural state of CPU respects the sequential semantic of the program, corresponds to last completed instruction.

Literature

References:

- Shen, J.P., Lipasti, M.H.: Modern Processor Design : Fundamentals of Superscalar Processors, First Edition, New York, McGraw-Hill Inc., 2005
- Brian Case: Intel Reveals Pentium Implementation Details, Vol. 7, No. 4, March 29, 1993