# Advanced Computer Architecture

Dynamic Instruction Level Parallelism

Lecture 2



# Improving Performance

- Three ways
  - Reduce clock cycle time
    - Technology, implementation
  - Reduce number of instructions
    - Improve instruction set
    - Improve compiler
  - Reduce Cycles/Instruction
    - Improve implementation
- Can we do better than 1 cycle/instruction?

# Instruction Level Parallelism

- Property of software
  - How many instructions are independent?
  - Very dependent on compiler
- Many ways to find ILP
  - Dynamic scheduling
  - Superscalar
  - Speculation
  - Static methods (Chapter 4)



# Multiple Issue

- Issue more than 1 instruction per cycle
- 2 variants
  - Superscalar
    - Extract parallelism from single-issue instruction set
    - Run unmodified sequential programs
  - VLIW
    - Parallelism is explicit in instruction set
    - Chapter 4

# Superscalar variants

- Scheduling
  - Static
    - In order execution
    - Early superscalar processors, e.g., Sun UltraSparc II
  - Dynamic
    - Tomasulo's algorithm – out of order execution, in order issue
  - Dynamic with speculation
    - Out of order execution, out of order issue
    - Most high-end CPU's today



# Issue constraints

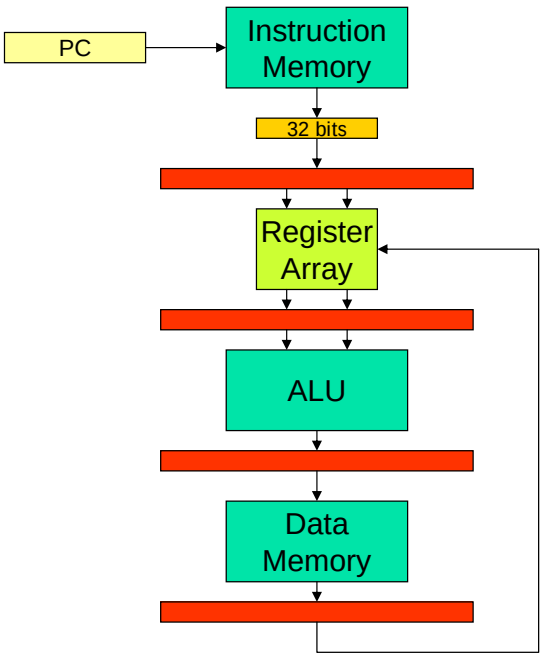
- Number of instructions to issue at once
  - Minimum 2, maximum 8
- Number of execution units
  - Can be larger than number issued
  - Usually classes of E.U's
  - Sometimes multiple instances per class
- Dependencies
  - RAW still applies

# Implementation Costs

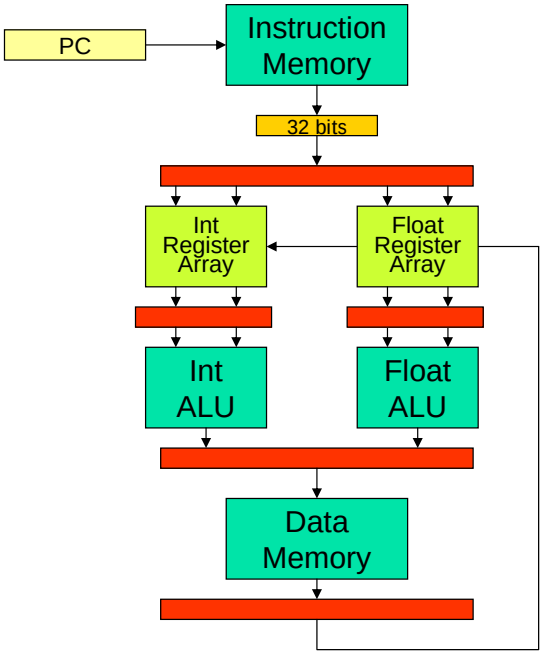
- Lookahead
  - Instruction window
  - IC alignment
- Register pressure
  - 2 extra read ports per function unit
- Branch penalties
  - 4 issue CPU, 25% of instructions are branches
- Hazard detection
- More?



# Single Issue

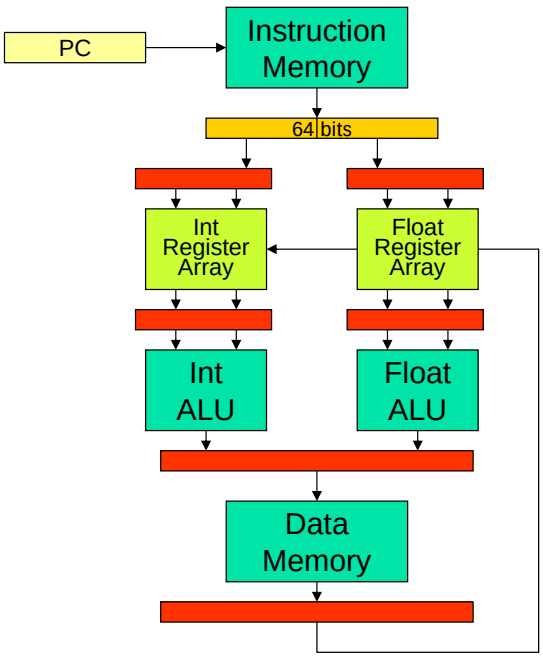


# Single Issue – separate float

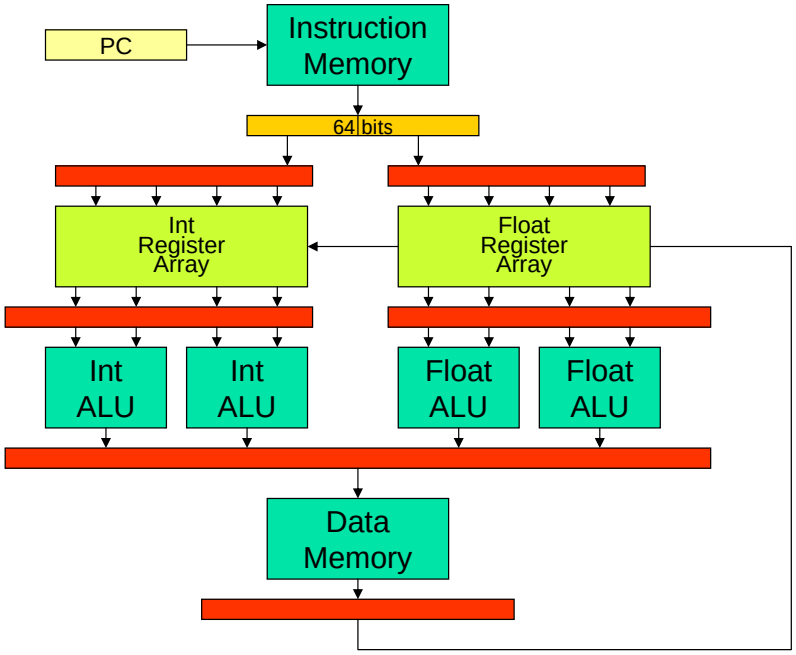





# Multiple Issue – separate float



# Multiple Issue – replication

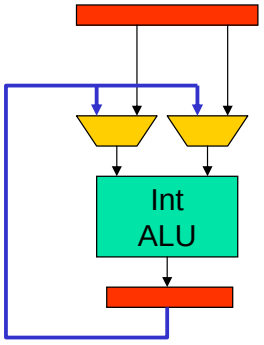





# Superscalar Function Units

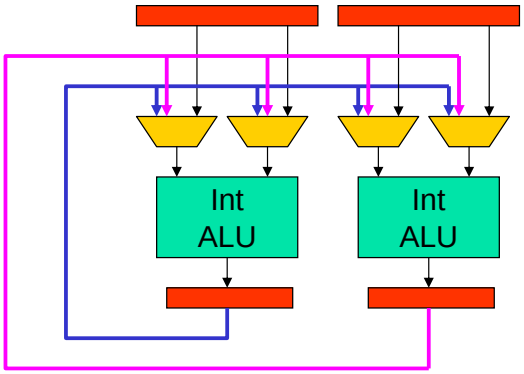
- Can be same as issue width or wider than issue width
- Varies by design
  - IBM RS/6000 (1st superscalar): 1 ALU/Load, 1 FP
  - Pentium II: 1 ALU/FP, 1 ALU, 1 Load, 1 Store, 1 Branch
  - Alpha 21264: 1 ALU/FP/Branch, 2 ALU, 1 Load/Store

# Forwarding – Standard Pipeline

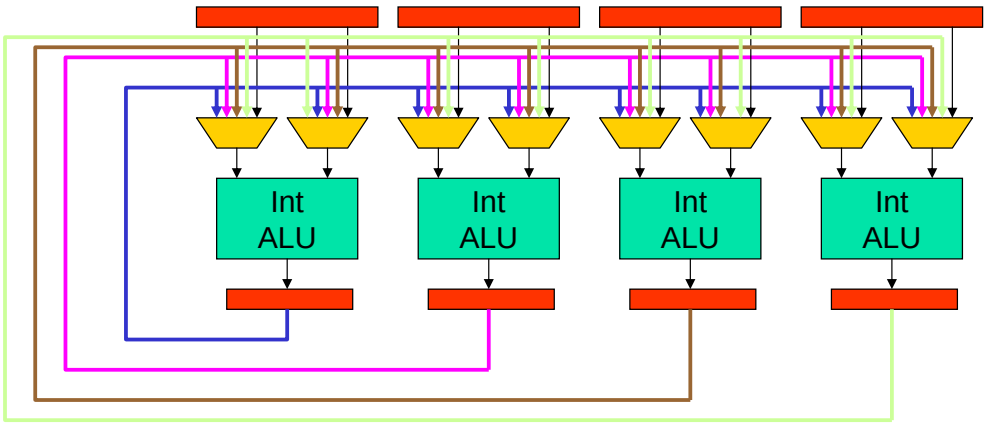





# Forwarding – 2-way Superscalar



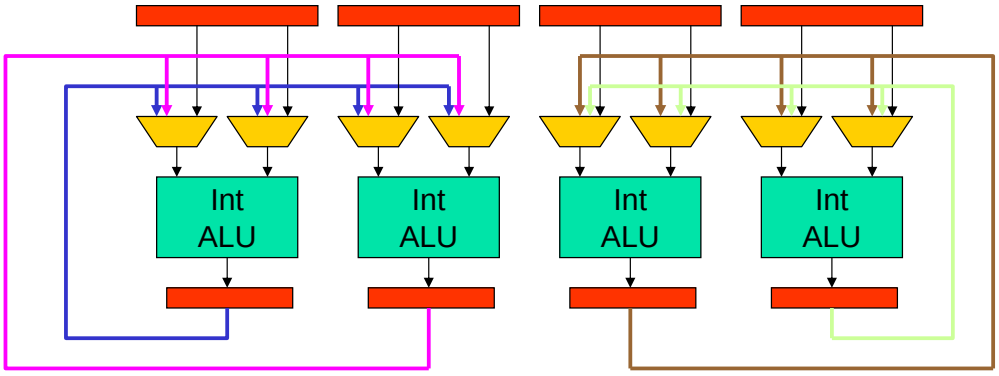
# Forwarding – 4-way Superscalar






# Clustering

Alpha 21264 an example



# Forwarding costs

- RAW detection
  - $N^2$ growth in detection logic
    - Relatively localized
  - $N^2$ growth in input muxes
    - Not so good
    - Wide buses (64-bit data)
    - Probably limits growth



# Impact on Caches

- Need to fetch *n* times as many bytes for *n* instructions issue
  - Branches a problem
    - Not-Taken can be issued, but must be checked
- Data Cache pressure
  - More reads in parallel
  - More writes in parallel

# Instruction Window

- Must fetch enough memory from Icache to issue all instructions
  - Wider fetch at same clock rate
- Must be able to analyze all instructions in window at once
  - Looking for branches



# Trace Caches

- Combine instruction cache with Branch Target Buffer
  - Tag: PC plus directions of branches
  - Instruction fetches come from trace cache before IC
- Still need regular IC for backup
- Used in high end superscalar
  - Pentium 4 (actually micro-ops)

# Trace Cache example

Assume inst0 is a branch to inst4

icache

| Addr | Data |
|---|---|
| 0 | inst 0, inst 1 |
| 2 | inst 2, inst 3 |
| 4 | inst 4, inst 5 |

tcache

| Addr | Taken? | Data |
|---|---|---|
| 0 | Yes | inst 0, inst 4 |
| 2 | - | inst 2, inst 3 |
| 4 | - | inst 4, inst 5 |

| | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|---|
| inst0 | F | D | X | M | W | | |
| inst4 | | F | D | X | M | W | |
| inst5 | | F | D | X | M | W | |
| inst6 | | | F | D | X | M | W |
| inst7 | | | F | D | X | M | W |

| | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|---|
| inst0 | F | D | X | M | W | | |
| inst4 | F | D | X | M | W | | |
| inst5 | | F | D | X | M | W | |
| inst6 | | F | D | X | M | W | |
| inst7 | | | F | D | X | M | W |



# Dynamic Scheduling

- Apply Tomasulo to Superscalar
- Reservation stations to each function unit
  - Difference: simultaneous assignments
- Pipeline management more complex
  - Sometimes extend pipeline to calculate
- May need even more function units
  - Out of order execution may result in resource conflicts
  - May need extra CDB (allows more than 1 completion per cycle)

# Speculation

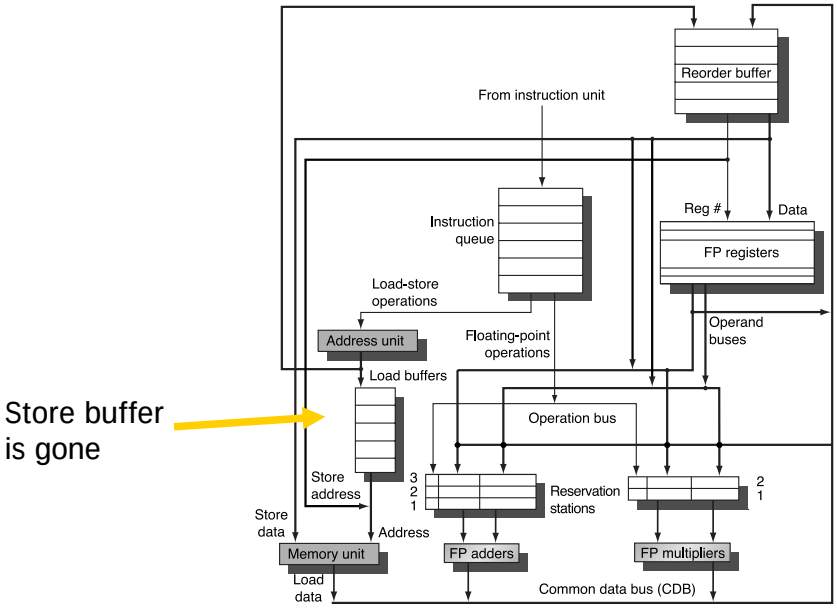
- Branches halt dynamic in-order issue
  - Speculate on result of branches and issue anyway
  - Fix up later if you guess wrong
  - *Really* need good branch prediction
  - Likewise, need dynamic scheduling
- Used in all modern high-performance processors
  - Pentium III, Pentium 4, Pentium M
  - PowerPC 603/604/G3/G4/G5
  - MIPS R10000/R12000
  - AMD K5/K6/Athlon/Opteron



# How?

- Allow out-of-order issue
- Allow out-of-order execution
- Allow out-of-order writeback
- *Commit* writes only when branches are resolved
  - Prevent speculative executions from changing state
  - Keep pending writes in a *reorder buffer*
  - Like adding even more registers than dynamic scheduling

# Adding a reorder buffer








# Stages of Speculative Execution

- Issue
  - Issue if reservation and reorder buffer slot are free
- Execute
  - When both operands are available, execute (check CDB)
- Write Result
  - Write to reservation stations and reorder buffer
- Commit
  - When instruction at head of buffer is ready, update registers/memory
  - If mispredicted branch, flush reorder buffer

# Different implementations

- Register Renaming (e.g., MIPS 10K)
  - Architectural registers replaced by larger physical register array
    - Registers dynamically mapped to correpsond to reservation stations, reorder buffer
  - Removes hazards
    - No name conflicts, since no reuse of names
  - Needs a free list!
  - Deallocating is complex
    - Do we still need a register after commit?
  - Permits a lot of parallelism



# Advantages of Reorder Buffer

- No more imprecise interrupts
  - Instructions complete out of order, but retire in order
  - Keeps dynamic schedule working through branches
    - Function as no-ops with respect to pipeline (unless mispredicted)
  - Extends register space
    - Even more instructions can be in flight

# Disadvantages of speculation

- Complexity
  - Handling variable length instructions
- A lot of data copying internally
- Even more load on CDB
- Even larger instruction window
  - Looking past branches
- Pressure on clock frequency
  - Sometimes simpler/faster beats complex/slower for performance
  - Simpler can be replicated more (see IBM's Blue Gene)



# How much further?

- How far past a branch to speculate?
- How many branches to speculate past?
  - Quadratic increase in complexity
  - Pressure on pipeline depth
  - No one does it very far