



LUND
UNIVERSITY

EITF20: Computer Architecture

Part3.2.1: Pipeline - 3

Liang Liu
liang.liu@eit.lth.se



Outline

- Reiteration
- Dynamic scheduling - Tomasulo
- Superscalar, VLIW
- Speculation
- ILP limitations
- What we have done so far



Instruction level parallelism (ILP)

- **ILP: Overlap execution of unrelated instructions: Pipelining**
- **Two main approaches:**
 - DYNAMIC \Rightarrow hardware detects parallelism
 - STATIC \Rightarrow software detects parallelism
 - Often a mix between both
- **Pipeline CPI = Ideal CPI + Structural stalls + Data hazard stalls + Control stalls**



Loop unrolling

- **Longer sequences of straight code without branches (longer basic blocks) allows for easier compiler static rescheduling**
- **Longer basic blocks also facilitates dynamic rescheduling such as Scoreboard and Tomasulo's algorithm**



Dynamic branch prediction

❑ Branches limit performance because:

- Branch penalties
- Limit to available Instruction Level Parallelism

❑ Solution: Dynamic branch prediction to predict the outcome of conditional branches

- Branch history table
- Branch target buffer

❑ Benefits:

- Reduce the time to when the branch condition is known
- Reduce the time to calculate the branch target address



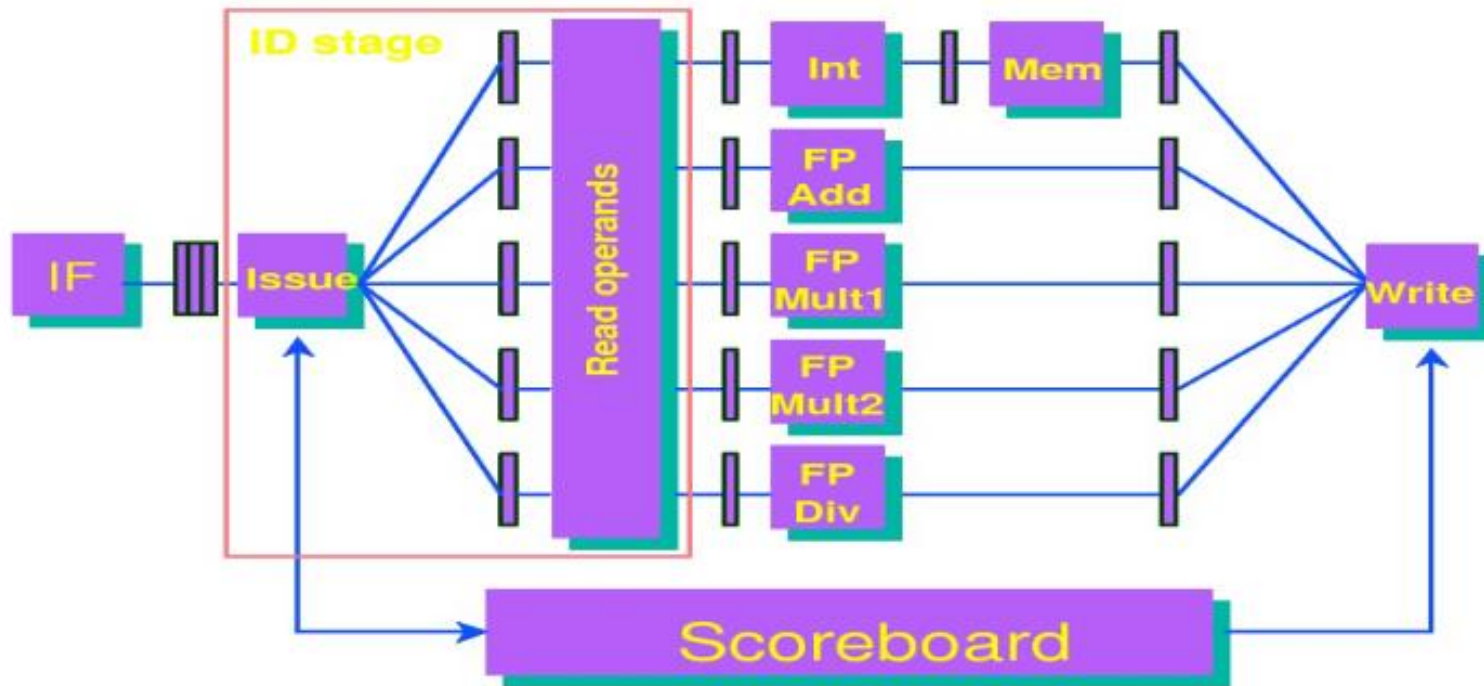
Dependencies

- ❑ **Two instructions must be independent in order to execute in parallel**
- ❑ **There are three general types of dependencies that limit parallelism:**
 - Data dependencies
 - Name dependencies
 - Control dependencies
- ❑ **Dependencies are properties of the program**
- ❑ **Whether a dependency leads to a hazard or not is a property of the pipeline implementation**



Scoreboard pipeline

- ❑ **Issue:** decode and check for structural hazards
- ❑ **Read operands:** wait until no data hazards, then read operands
- ❑ **All data hazards are handled by the scoreboard**



Scoreboard pipeline

- ❑ **Goal of scoreboarding** is to maintain an execution rate of one instruction per clock cycle by executing an instruction as early as possible
- ❑ **Instructions execute out-of-order** when there are sufficient resources and no data dependencies
- ❑ **A scoreboard is a hardware unit that keeps track of**
 - the **instructions** that are in the process of being executed
 - the **functional units** that are doing the executing
 - and the **registers** that will hold the results of those units
- ❑ **A scoreboard centrally performs** all hazard detection and instruction control



Scoreboard functionality

□ **Issue:** An instruction is issued if:

- The needed functional unit is free (there is no **structural hazard**)
- No functional unit has a destination operand equal to the destination of the instruction (resolves **WAW hazards**)

□ **Read:** Wait until no data hazards, then read operands

- Performed in parallel for all functional units
- Resolves **RAW hazards** dynamically

□ **EX:** Normal execution

- Notify the scoreboard when ready

□ **Write:** The instruction can update destination if:

- All earlier instructions have read their operands (resolves **WAR hazards**)



Factors that limits performance

The scoreboard technique is limited by:

- ❑ The amount of parallelism available in code
- ❑ The number of scoreboard entries (window size)
 - A large window can look ahead across more instructions
- ❑ The number and types of functional units
 - Contention for functional units leads to structural hazards
- ❑ The presence of anti-dependencies and output dependencies
 - These lead to WAR and WAW hazards that are handled by stalling the instruction in the Scoreboard
- ❑ Number of data-paths to registers

Tomasulo's algorithm addresses the last two limitations.



Outline

- Reiteration
- **Dynamic scheduling - Tomasulo**
- Superscalar, VLIW
- Speculation
- ILP limitations
- What we have done so far



Tomasulo algorithm

Another dynamic instruction scheduling algorithm

- For IBM 360/91, a few years after the CDC 6600 (Scoreboard)
- Goal: High performance without compiler support

R. M. Tomasulo

An Efficient Algorithm for Exploiting Multiple Arithmetic Units

Abstract: This paper describes the methods employed in the floating-point area of the System/360 Model 91 to exploit the existence of multiple execution units. Basic to these techniques is a simple common data busing and register tagging scheme which permits simultaneous execution of independent instructions while preserving the essential precedences inherent in the instruction stream. The common data bus improves performance by efficiently utilizing the execution units without requiring specially optimized code. Instead, the hardware, by 'looking ahead' about eight instructions, automatically optimizes the program execution on a local basis.

The application of these techniques is not limited to floating-point arithmetic or System/360 architecture. It may be used in almost any computer having multiple execution units and one or more 'accumulators.' Both of the execution units, as well as the associated storage buffers, multiple accumulators and input/output buses, are extensively checked.



Registr renaming

Example:

| | | | | | |
|------|------------|------|-------------|------|----------------|
| LD | F6, 34(R2) | LD | FT, 34(R2) | | |
| ... | ... | ... | | | |
| DIVD | F10,F0,F6 | DIVD | F10, F0, FT | DIVD | F10, F0, F6 |
| ADDD | F6,F8,F2 | ADDD | F6, F8, F2 | ADDD | FT, F8, F2 |
| | | | | ... | |
| | | | | SD | FT(F6), 35(R3) |

□ Potential WAR harzard on F6

- If ADDD finishes before DIVD starts

□ Register renaming

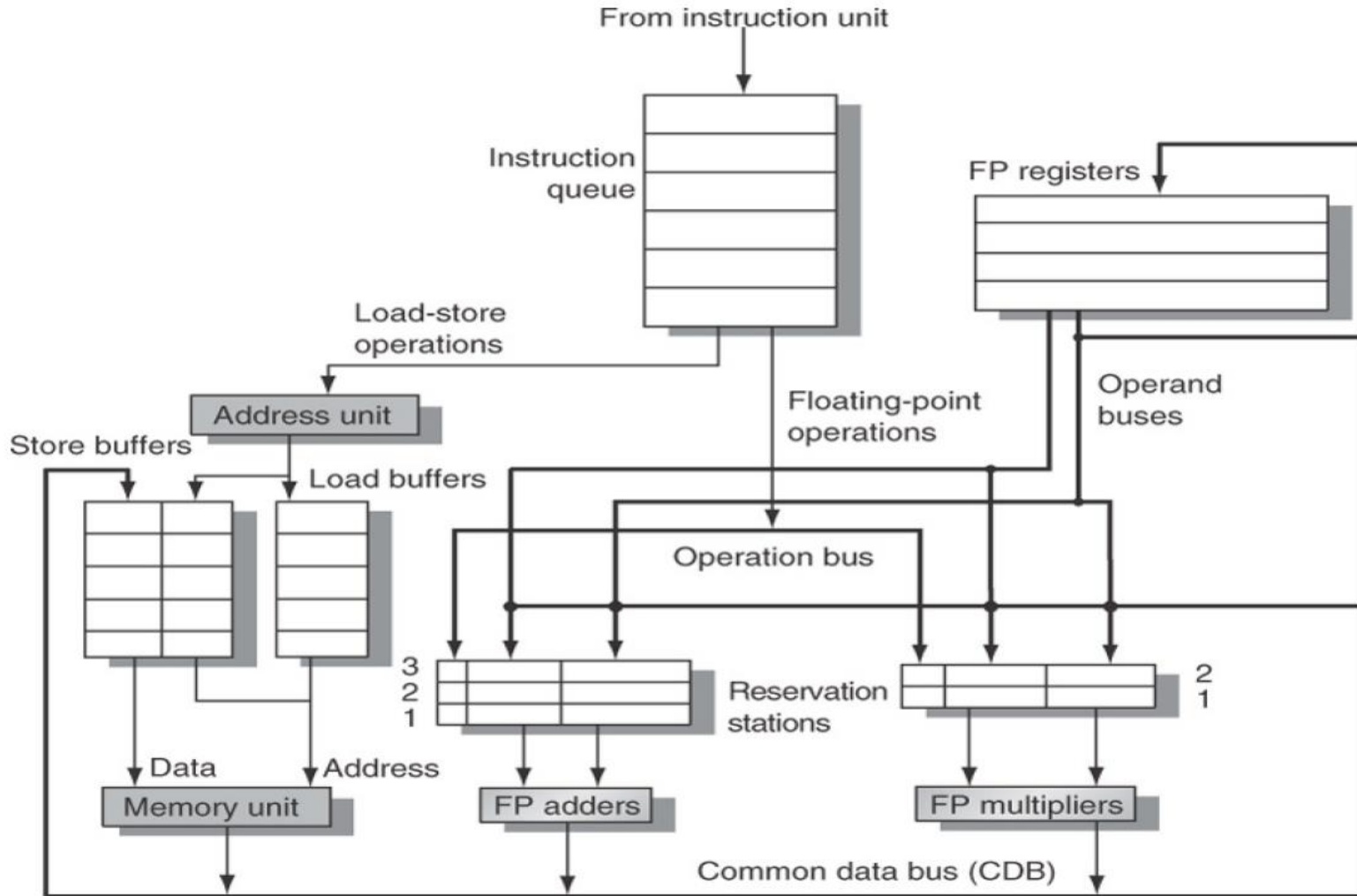
- Another temporaray register (FT) instead of F6
- Any subsequent uses of F6 should be replaced by FT

□ Register renaming can thus be done:

- statically by the compiler
- dynamically by the hardware



Tomasulo orgnizations



Reservation stations

- ❑ Op: Operation to perform (e.g., + or –)
- ❑ Vj, Vk: **Value (instead of reg specifier)** of Source operands
 - Buffer for Store Inst. has one V field, result to be stored
- ❑ Qj, Qk: **Reservation stations (pointer, instead of FU)** producing source registers (value to be written)
 - Note: Qj, Qk=0 => ready
 - Buffer for Store Inst. has single Q field, source of value
- ❑ **Busy:** Indicates reservation station or FU is busy
- ❑ **Register result status**—Indicates which RS will write each register, if one exists
 - Blank when no pending instructions that will write that register

| <u>Functional unit status</u> | | | | src 1 | src 2 | RS for j | RS for k |
|-------------------------------|-------|------|----|-------|-------|----------|----------|
| Time | Name | Busy | Op | Vj | Vk | Qj | Qk |
| | Add1 | No | | | | | |
| | Add2 | No | | | | | |
| | Add3 | No | | | | | |
| | Mult1 | No | | | | | |
| | Mult2 | No | | | | | |



Three stages of Tomasulo algorithm

- **Issue** – get instruction from FP Op Queue
 - If reservation station free (no structural hazard)
 - instruction is issued together with its operands values or RS point (reg rename)
- **Execution** – operate on operands (EX)
 - When both operands are ready, then execute;
 - if not ready, watch Common Data Bus (CDB) for operands (snooping)
- **Write result** – finish execution (WB)
 - Write on CDB to all awaiting functional units, regs
 - Mark reservation station available
- **Normal bus: data + destination**
- **Common Data Bus: data + source (snooping)**



Tomasulo example, cycle 0

Instruction status

| Instruction | j | k | Issue | Exec. compl. | Write result |
|-------------|-----|-----|-------|--------------|--------------|
| LD | F6 | 34+ | R2 | | |
| LD | F2 | 45+ | R3 | | |
| MULTD | F0 | F2 | F4 | | |
| SUBD | F8 | F6 | F2 | | |
| DIVD | F10 | F0 | F6 | | |
| ADD | F6 | F8 | F2 | | |

Load buffers

| | Busy | Address |
|-------|------|---------|
| Load1 | No | |
| Load2 | No | |
| Load3 | No | |

Functional unit status

| Time | Name | Busy | Op | src 1 Vj | src 2 Vk | RS for j Qj | RS for k Qk |
|------|-------|------|----|-------------|-------------|----------------|----------------|
| | Add1 | No | | | | | |
| | Add2 | No | | | | | |
| | Add3 | No | | | | | |
| | Mult1 | No | | | | | |
| | Mult2 | No | | | | | |

Register result status

| | F0 | F2 | F4 | F6 | F8 | F10 | ... | F30 |
|----|----|----|----|----|----|-----|-----|-----|
| FU | | | | | | | | |

Clock: 0



Tomasulo example, cycle 1

Instruction status

| Instruction | j | k | Exec. Write Issue compl. result |
|--------------|------------|-----------|------------------------------------|
| LD F6 | 34+ | R2 | 1 |
| LD F2 | 45+ | R3 | |
| MULTD F0 | F2 | F4 | |
| SUBD F8 | F6 | F2 | |
| DIVD F10 | F0 | F6 | |
| ADDD F6 | F8 | F2 | |

Load buffers

| Time | Busy | Address |
|-------|------------|--------------|
| Load1 | Yes | R2+34 |
| Load2 | No | |
| Load3 | No | |

Functional unit status

| Time | Name | Busy | Op | src 1 | src 2 | RS for j | RS for k |
|------|-------|------|----|-------|-------|----------|----------|
| | | | | Vj | Vk | Qj | Qk |
| | Add1 | No | | | | | |
| | Add2 | No | | | | | |
| | Add3 | No | | | | | |
| | Mult1 | No | | | | | |
| | Mult2 | No | | | | | |

Register result status

| | F0 | F2 | F4 | F6 | F8 | F10 | ... | F30 |
|----|----|----|----|--------------|----|-----|-----|-----|
| FU | | | | Load1 | | | | |

Clock: **1**



Tomasulo example, cycle 3

Instruction status

| Instruction | j | k | Issue | Exec. compl. | Write result |
|-----------------|-----------|-----------|----------|--------------|--------------|
| LD F6 | 34+ | R2 | 1 | | |
| LD F2 | 45+ | R3 | 2 | | |
| MULTD F0 | F2 | F4 | 3 | | |
| SUBD F8 | F6 | F2 | | | |
| DIVD F10 | F0 | F6 | | | |
| ADDD F6 | F8 | F2 | | | |

| Time | Load1 | Load2 | Load3 | Busy | Address |
|------|-------|-------|-------|------|---------|
| 0 | Yes | Yes | No | | R2+32 |
| 1 | Yes | Yes | No | | R3+45 |

Functional unit status

| Time | Name | Busy | Op | src 1 Vj | src 2 Vk | RS for j Qj | RS for k Qk |
|------|-------|------|------|-------------|-------------|----------------|----------------|
| | Add1 | No | | | | | |
| | Add2 | No | | | | | |
| | Add3 | No | | | | | |
| | Mult1 | Yes | Mult | | F4 | Load2 | |
| | Mult2 | No | | | | | |

Register result status

| FU | F0 | F2 | F4 | F6 | F8 | F10 | ... | F30 |
|----|-------|-------|----|-------|----|-----|-----|-----|
| | Mult1 | Load2 | | Load1 | | | | |

Clock: 3

Note:

1. Can have multiple loads
2. registers names are removed (“renamed”) in Reservation Stations



Tomasulo example, cycle 4

Instruction status

| Instruction | j | k | Issue | Exec. compl. | Write result |
|----------------|-----------|-----------|----------|--------------|--------------|
| LD F6 | 34+ | R2 | 1 | 3 | 4 |
| LD F2 | 45+ | R3 | 2 | 4 | |
| MULTD F0 | F2 | F4 | 3 | | |
| SUBD F8 | F6 | F2 | 4 | | |
| DIVD F10 | F0 | F6 | | | |
| ADDD F6 | F8 | F2 | | | |

Load buffers

| Time | Busy | Address |
|------|------|---------|
| 0 | No | |
| 0 | Yes | R3+45 |
| 0 | No | |

Functional unit status

| Time | Name | Busy | Op | src 1 Vj | src 2 Vk | RS for j Qj | RS for k Qk |
|------|-------|------|------|-------------|-------------|----------------|----------------|
| | Add1 | Yes | Sub | M(R2+34) | | | Load2 |
| | Add2 | No | | | | | |
| | Add3 | No | | | | | |
| | Mult1 | Yes | Mult | | F4 | | Load2 |
| | Mult2 | No | | | | | |

Register result status

| FU | F0 | F2 | F4 | F6 | F8 | F10 | ... | F30 |
|----|-------|-------|----|----|------|-----|-----|-----|
| | Mult1 | Load2 | | - | Add1 | | | |

Clock: 4



Tomasulo example, cycle 5

Instruction status

| Instruction | j | k |
|-----------------------|---|---|
| LD F6 34+ R2 | | |
| LD F2 45+ R3 | | |
| MULTD F0 F2 F4 | | |
| SUBD F8 F6 F2 | | |
| DIVD F10 F0 F6 | | |
| ADDD F6 F8 F2 | | |

Exec. Write
Issue compl. result

| Issue | compl. | result |
|----------|--------|----------|
| 1 | 3 | 4 |
| 2 | 4 | 5 |
| 3 | | |
| 4 | | |
| 5 | | |

Load buffers

| Time | Busy | Address |
|-------|-----------|---------|
| Load1 | No | |
| Load2 | No | |
| Load3 | No | |

Functional unit status

| Time | Name | Busy | Op | src 1 Vj | src 2 Vk | RS for j Qj | RS for k Qk |
|-----------|-------|------------|------------|-----------------|-----------------|----------------|----------------|
| 2 | Add1 | Yes | Sub | M(R2+34) | M(R3+45) | | - |
| | Add2 | No | | | | | |
| | Add3 | No | | | | | |
| 10 | Mult1 | Yes | Mult | M(R3+45) | F4 | | - |
| | Mult2 | Yes | Div | | F6 | Mult1 | |

Register result status

| | F0 | F2 | F4 | F6 | F8 | F10 | ... | F30 |
|----|-------|----|----|----|------|--------------|-----|-----|
| FU | Mult1 | - | | | Add1 | Mult2 | | |

Clock: **5**



Tomasulo example, cycle 7

Instruction status

| Instruction | j | k | Issue | Exec. compl. | Write result |
|-------------|-----|----|-------|--------------|--------------|
| LD F6 | 34+ | R2 | 1 | 3 | 4 |
| LD F2 | 45+ | R3 | 2 | 4 | 5 |
| MULTD F0 | F2 | F4 | 3 | | |
| SUBD F8 | F6 | F2 | 4 | 7 | |
| DIVD F10 | F0 | F6 | 5 | | |
| ADDD F6 | F8 | F2 | 6 | | |

Load buffers

| Time | Busy | Address |
|-------|------|---------|
| Load1 | No | |
| Load2 | No | |
| Load3 | No | |

Functional unit status

| Time | Name | Busy | Op | src 1 Vj | src 2 Vk | RS for j Qj | RS for k Qk |
|------|-------|------|------|-------------|-------------|----------------|----------------|
| 0 | Add1 | Yes | Sub | M(R2+34) | M(R3+45) | | |
| | Add2 | Yes | Add | | F2 | Add1 | |
| | Add3 | No | | | | | |
| 8 | Mult1 | Yes | Mult | M(R3+45) | F4 | | |
| | Mult2 | Yes | Div | | F6 | Mult1 | |

Register result status

| FU | F0 | F2 | F4 | F6 | F8 | F10 | ... | F30 |
|----|-------|----|----|------|------|-------|-----|-----|
| | Mult1 | | | Add2 | Add1 | Mult2 | | |

Clock: 7



Tomasulo example, cycle 10

Instruction status

| Instruction | j | k | Issue | Exec. compl. | Write result |
|-------------|-----|--------|-------|--------------|--------------|
| LD | F6 | 34+ R2 | 1 | 3 | 4 |
| LD | F2 | 45+ R3 | 2 | 4 | 5 |
| MULTD | F0 | F2 F4 | 3 | | |
| SUBD | F8 | F6 F2 | 4 | 7 | 8 |
| DIVD | F10 | F0 F6 | 5 | | |
| ADDD | F6 | F8 F2 | 6 | 10 | |

| Time | Load buffers | Busy | Address |
|------|--------------|------|---------|
| | Load1 | No | |
| | Load2 | No | |
| | Load3 | No | |

Functional unit status

| Time | Name | Busy | Op | src 1 V _j | src 2 V _k | RS for j Q _j | RS for k Q _k |
|----------|-------|------|------|-------------------------|-------------------------|----------------------------|----------------------------|
| | Add1 | No | | | | | |
| 0 | Add2 | Yes | Add | F6-F2 | F2 | | |
| | Add3 | No | | | | | |
| 5 | Mult1 | Yes | Mult | M(R3+45) | F4 | | |
| | Mult2 | Yes | Div | | F6 | Mult1 | |

Register result status

| | F0 | F2 | F4 | F6 | F8 | F10 | ... | F30 |
|----|-------|----|----|------|----|-------|-----|-----|
| FU | Mult1 | | | Add2 | | Mult2 | | |

Clock: **10**



Elimination of WAR hazard

LD F6, 34(R2)

Example:
DIVD F10,F0,F6
ADDD F6,F8,F2

□ **ADDD can safely finish before DIVD has read register F6 because:**

- DIVD has renamed register F6 to point at the reservation station
- LD broadcasts its result on the Common Data Bus
- Even **WAW** hazard can be solved if LD is delayed



Tomasulo example, cycle 11

Instruction status

| Instruction | j | k | Issue | Exec. compl. | Write result |
|-------------|-----|----|-------|--------------|--------------|
| LD F6 | 34+ | R2 | 1 | 3 | 4 |
| LD F2 | 45+ | R3 | 2 | 4 | 5 |
| MULTD F0 | F2 | F4 | 3 | | |
| SUBD F8 | F6 | F2 | 4 | 7 | 8 |
| DIVD F10 | F0 | F6 | 5 | | |
| ADD F6 | F8 | F2 | 6 | 10 | 11 |

Load buffers

| Time | Busy | Address |
|-------|------|---------|
| Load1 | No | |
| Load2 | No | |
| Load3 | No | |

Functional unit status

| Time | Name | Busy | Op | src 1 Vj | src 2 Vk | RS for j Qj | RS for k Qk |
|------|-------|------|------|-------------|-------------|----------------|----------------|
| | Add1 | No | | | | | |
| | Add2 | No | | | | | |
| | Add3 | No | | | | | |
| 4 | Mult1 | Yes | Mult | M(R3+45) | F4 | | |
| | Mult2 | Yes | Div | | F6 | Mult1 | |

Register result status

| FU | F0 | F2 | F4 | F6 | F8 | F10 | ... | F30 |
|----|-------|----|----|----|----|-------|-----|-----|
| | Mult1 | | | - | | Mult2 | | |

Clock: 11



Tomasulo example, cycle 16

Instruction status

| Instruction | j | k | Issue | Exec. compl. | Write result |
|-------------|-----|----|-------|--------------|--------------|
| LD F6 | 34+ | R2 | 1 | 3 | 4 |
| LD F2 | 45+ | R3 | 2 | 4 | 5 |
| MULTD F0 | F2 | F4 | 3 | 15 | 16 |
| SUBD F8 | F6 | F2 | 4 | 7 | 8 |
| DIVD F10 | F0 | F6 | 5 | | |
| ADD F6 | F8 | F2 | 6 | 10 | 11 |

Load buffers

| Time | Busy | Address |
|-------|------|---------|
| Load1 | No | |
| Load2 | No | |
| Load3 | No | |

Functional unit status

| Time | Name | Busy | Op | src 1 Vj | src 2 Vk | RS for j Qj | RS for k Qk |
|------|-------|------|-----|-------------|-------------|----------------|----------------|
| | Add1 | No | | | | | |
| | Add2 | No | | | | | |
| | Add3 | No | | | | | |
| | Mult1 | No | | | | | |
| 40 | Mult2 | Yes | Div | F0 | F6 | - | |

Register result status

| FU | F0 | F2 | F4 | F6 | F8 | F10 | ... | F30 |
|----|----|----|----|----|----|-------|-----|-----|
| | - | | | | | Mult2 | | |

Clock: 16



Tomasulo example, cycle 57

Instruction status

| Instruction | j | k | Issue | Exec. compl. | Write result |
|-------------|-----|----|-------|--------------|--------------|
| LD F6 | 34+ | R2 | 1 | 3 | 4 |
| LD F2 | 45+ | R3 | 2 | 4 | 5 |
| MULTD F0 | F2 | F4 | 3 | 15 | 16 |
| SUBD F8 | F6 | F2 | 4 | 7 | 8 |
| DIVD F10 | F0 | F6 | 5 | 56 | 57 |
| ADD F6 | F8 | F2 | 6 | 10 | 11 |

| Time | Load buffers | Busy | Address |
|------|--------------|------|---------|
| | Load1 | No | |
| | Load2 | No | |
| | Load3 | No | |

Functional unit status

| Time | Name | Busy | Op | src 1 Vj | src 2 Vk | RS for j Qj | RS for k Qk |
|------|-------|-----------|----|-------------|-------------|----------------|----------------|
| | Add1 | No | | | | | |
| | Add2 | No | | | | | |
| | Add3 | No | | | | | |
| | Mult1 | No | | | | | |
| | Mult2 | No | | | | | |

Register result status

| FU | F0 | F2 | F4 | F6 | F8 | F10 | ... | F30 |
|----|----|----|----|----|----|-----|-----|-----|
| | | | | | | - | | |

Clock: **57**



Tomasulo vs Scoreboard

Differences between Tomasulo Algorithm and Scoreboard

- ❑ Control and buffers **distributed** with Function Units versus **centralized** in scoreboard
- ❑ Registers in instructions replaced by **pointers** to reservation stations
 - this is called register renaming
 - renaming helps avoid WAR and WAW hazards
 - more reservation stations than registers; so allow optzns compilers can't do
 - Operands stays in reg in Scoreboard
- ❑ **Common Data Bus** broadcasts results to all FUs (forwarding!)
- ❑ Load and Stores treated as FUs as well, with reservation stations for stores



Tomasulo summary

□ **Instructions:** move from decoder to reservation stations

- in program order
- dependences can be correctly recorded
- Distributed hazard detection

□ **Significant increase in HW cost**

□ **Benefits**

- tolerates unpredictable delays
- compile for one pipeline - run effectively on another
- register renaming, remove WAW, WAR hazard
- out-of-order execution, completion



Outline

- Reiteration
- Dynamic scheduling - Tomasulo
- **Superscalar, VLIW**
- Speculation
- ILP limitations
- What we have done so far



Getting CPI <1 !?

Issuing multiple (**independent**) instructions per clock cycle

□ **Superscalar: varying number of instructions/cycle (1-8) scheduled by compiler or HW**

- IBM Power5, Pentium 4, Sun SuperSparc, DEC Alpha
- Simple hardware, complicated compiler (static) or...
- Very complex hardware but simple for compiler (dynamic)

□ **Very Long Instruction Word (VLIW): fixed number of instructions (3-5) scheduled by the compiler**

- HP/Intel IA-64, Itanium
- Simple hardware, difficult for compiler
- High performance through extensive compiler optimization



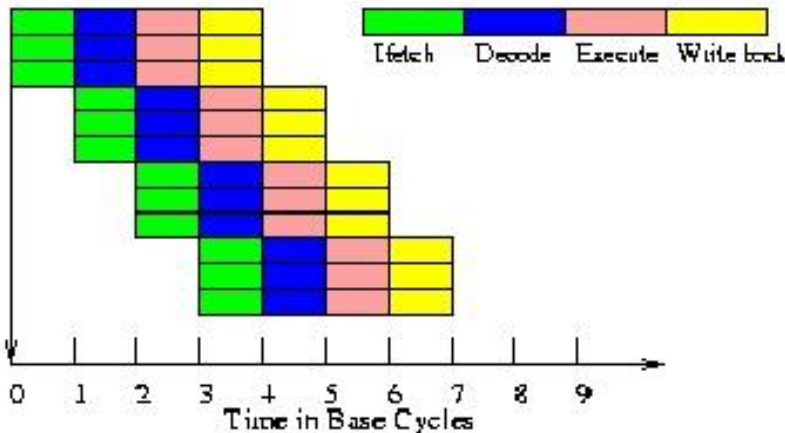
Approches for multiple issuing

| | Issue | Hazard detection | Scheduling | Characteristics /examples |
|-------------|--------------|-----------------------------|-------------------|--|
| Superscalar | dynamic | HW | static | in-order execution ARM |
| Superscalar | dynamic | HW | dynamic | out-of-order execution |
| Superscalar | dynamic | HW | dynamic | speculation Pentium 4 IBM power5 |
| VLIW | static | compiler | static | TI C6x |
| EPIC | static | compiler | mostly static | Itanium |

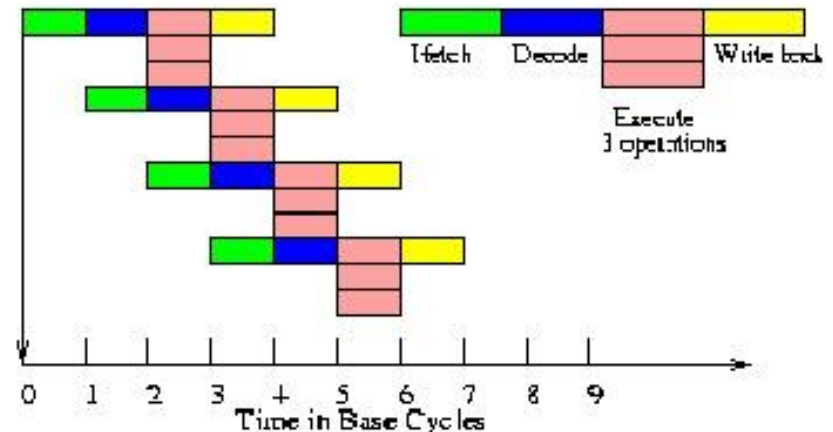


VLIW

- A number of functional units that independently execute instructions in parallel.
- The compiler decides which instructions can execute in parallel
- No hazard detection needed



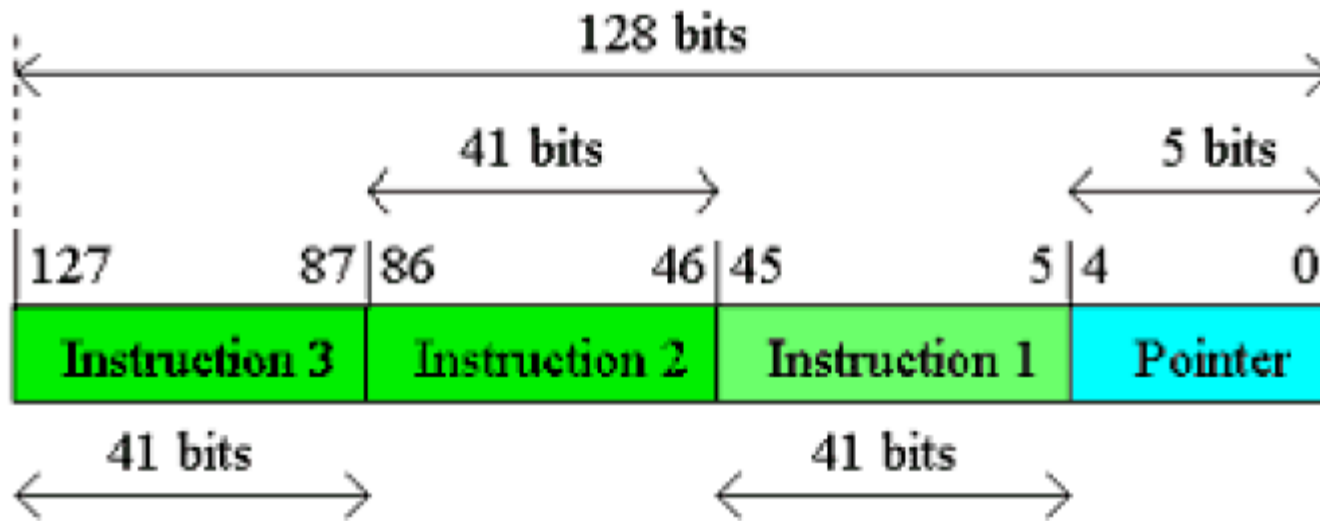
A Superscalar Processor of Degree 3



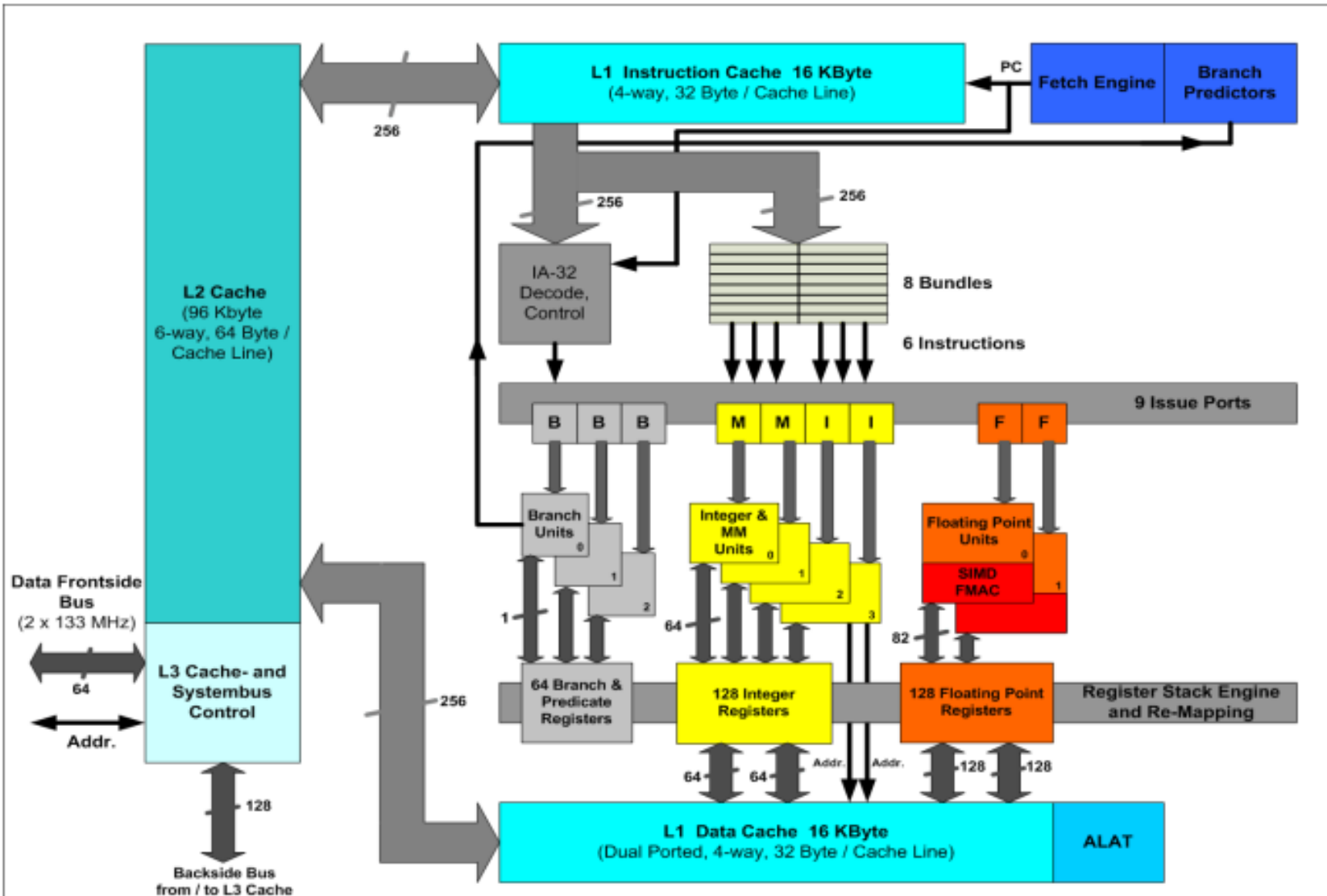
VLIW Execution with Degree 3



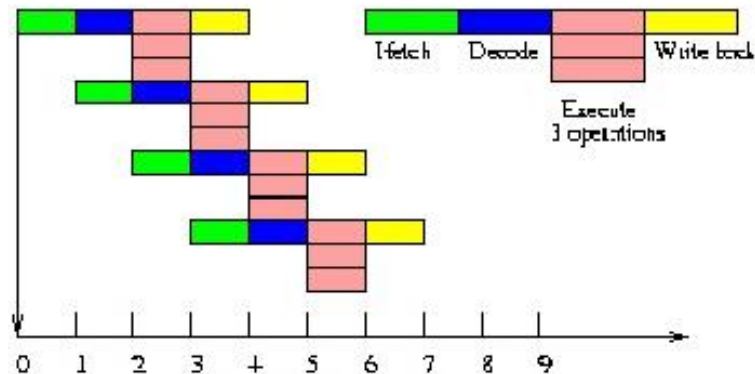
VLIW instruction format - Itanium



VLIW architecture- Itanium



VLIW limitations



❑ Limited Instruction Level Parallelism

- With n functional units and k pipeline stages we need $n * k$ independent instructions to utilize the hardware

❑ Memory and register bandwidth

- With increasing number of functional units, the number of ports needed at the memory or register file must increase to prevent structural hazards

❑ Code size

- Compiler scheduled pipeline “bubbles” take up space in the instruction
- Need more aggressive loop unrolling to work well which also increases code size

❑ No binary code compatibility

- Different pipeline stages and # function units requires different code

❑ Synchronization

- Stall in one FU cause the entire processor to stall



Outline

- Reiteration
- Dynamic scheduling - Tomasulo
- Superscalar, VLIW
- **Speculation**
- ILP limitations
- What we have done so far



Hardware-base speculation

- Trying to exploit more ILP (e.g., multiple issue) while maintaining **control dependencies** becomes a burden
- Overcome control dependencies by **speculating** on the outcome of branches and executing the program as if our guesses were correct
 - Need to handle incorrect guesses
- Key ideas:
 - Dynamic branch prediction: allows instruction scheduling across branches (not exe)
 - Dynamic scheduling: take advantage of ILP (wait for clear branch)
 - Speculation: **execute** instructions before all control dependencies are resolved
- Essentially a **data flow execution model**: Operations execute as soon as their operands are available



Hardware vs software speculation

□ Advantages:

- Dynamic runtime disambiguation of memory addresses
- Dynamic branch prediction is often better than static which limits the performance of SW speculation
- HW speculation can maintain a precise exception model
- Can achieve higher performance on older code (without recompilation)

□ Main disadvantage:

- Extremely complex implementation and extensive need for hardware resources



Implementing speculation

□ Key idea

- Allow instructions to execute **out of order**
- Force instructions to **commit in order**
- Prevent any **irrevocable action** (such as updating state or taking an exception) until an instruction commits.

□ Strategies:

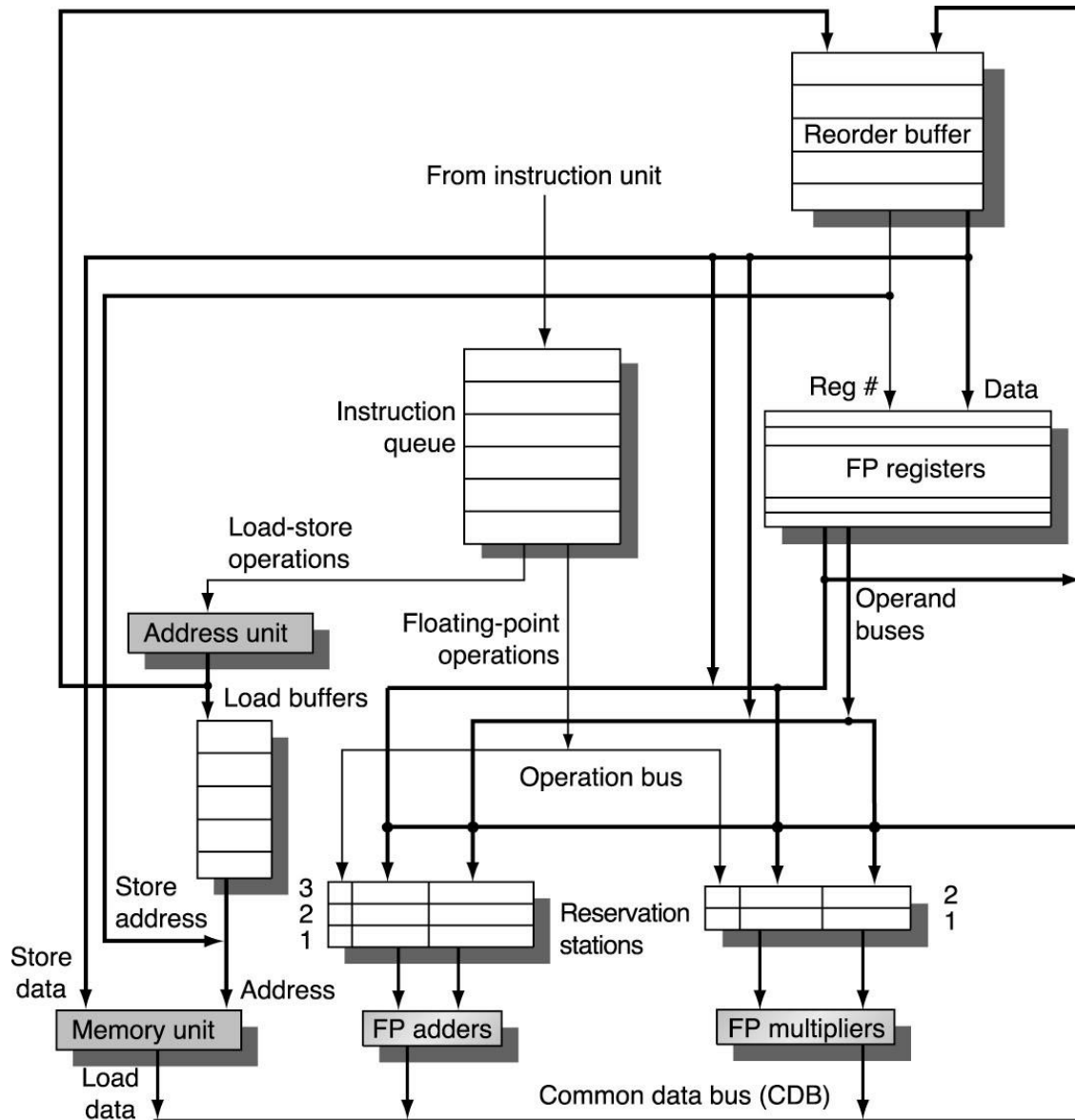
- Must separate **bypassing of results** among instructions from **actual completion** (write-back) of instructions
- Cannot allow updates to be performed that can't be undone
- Instruction commit updates register or memory when instruction no longer speculative

□ Need to add **re-order buffer**

- Hold the results of inst. that have finished exe but have not committed



Tomasulo extended to support speculation



ROB (reorder buffer)

| entry | instruction type | destination | value | ready |
|-------|------------------|-------------|-------|-------|
| 1 | | | | |
| 2 | | | | |
| ... | | | | |
| n | | | | |

□ Contains 4 fields:

- Instruction type indicates whether branch, store, or register op
- Destination field memory or register
- Value field hold the inst. result until commit
- Ready flag indicates instruction has completed operation

□ The renaming function of the reservation stations is replaced by the ROB

□ Every instruction has a ROB entry until it commits

- Therefore tag results using ROB entry number



ROB (reorder buffer)

□ When MUL.D is ready to commit

| | |
|-------|-------------|
| L.D | F6, 32 (R2) |
| L.D | F2, 44 (R3) |
| MUL.D | F0, F2, F4 |
| SUB.D | F8, F2, F6 |
| DIV.D | F10, F0, F6 |
| ADD.D | F6, F8, F2 |

| Reorder buffer | | | | | | |
|----------------|------|-------------|-------------|--------------|-------------|--------------------|
| Entry | Busy | Instruction | | State | Destination | Value |
| 1 | No | L.D | F6, 32 (R2) | Commit | F6 | Mem[32 + Regs[R2]] |
| 2 | No | L.D | F2, 44 (R3) | Commit | F2 | Mem[44 + Regs[R3]] |
| 3 | Yes | MUL.D | F0, F2, F4 | Write result | F0 | #2 × Regs[F4] |
| 4 | Yes | SUB.D | F8, F2, F6 | Write result | F8 | #2 − #1 |
| 5 | Yes | DIV.D | F10, F0, F6 | Execute | F10 | |
| 6 | Yes | ADD.D | F6, F8, F2 | Write result | F6 | #4 + #2 |

| Reservation stations | | | | | | | | | |
|----------------------|------|-------|--------------------|--------------------|----|----|------|---|--|
| Name | Busy | Op | Vj | Vk | Qj | Qk | Dest | A | |
| Load1 | No | | | | | | | | |
| Load2 | No | | | | | | | | |
| Add1 | No | | | | | | | | |
| Add2 | No | | | | | | | | |
| Add3 | No | | | | | | | | |
| Mult1 | No | MUL.D | Mem[44 + Regs[R3]] | Regs[F4] | | | #3 | | |
| Mult2 | Yes | DIV.D | | Mem[32 + Regs[R2]] | #3 | | #5 | | |



Four-step speculation

□ Issue:

- Get instruction from instruction queue and issue if reservation station and **ROB slots** available – sometimes called dispatch
- Send operands or **ROB entry #** for destination

□ Execution – operate on operands (EX)

- If both operands ready: execute; if not, watch CDB for result;
- When both operands are in reservation station: execute

□ Write result – complete execution

- Write on CDB to all awaiting **FUs (RSs) & ROB** (tagged by **ROB entry #**)
- Mark reservation station available

□ Commit – update register with reorder result

- When instr. is at head of reorder buffer & result is present; update register with result (or store to memory) and remove instr. from reorder buffer;
- (handle misspeculations and precise exceptions)



Four-step speculation

□ Commit – branch prediction wrong

- When branch instr. is at head of reorder buffer & incorrect prediction: remove all instr. from reorder buffer (flush); restart execution at correct instruction
- Expensive \Rightarrow try to recover as early as possible (delay in ROB)
- Performance sensitive to branch prediction/speculation (waste computation power)

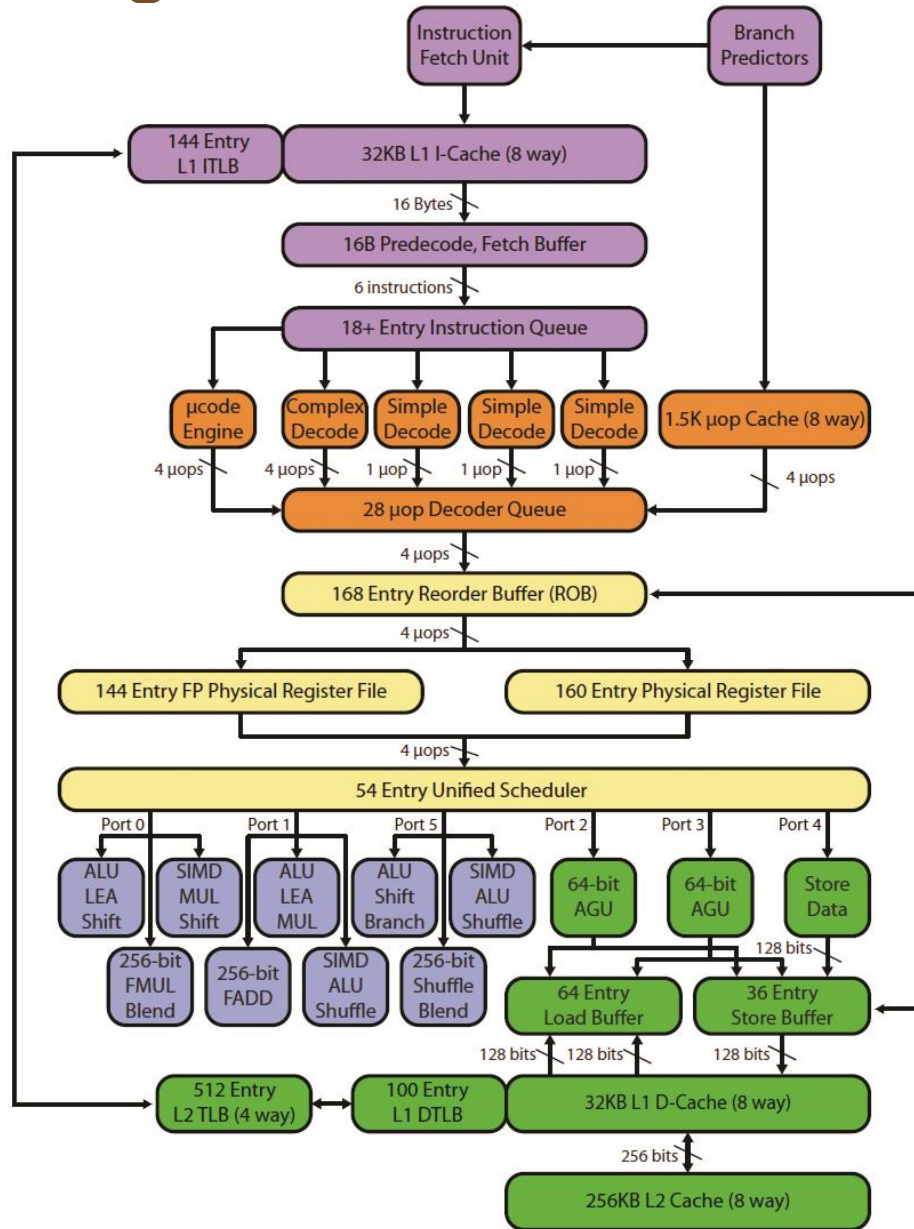


Dynamic scheduling, speculation summary

- ❑ **Tolerates unpredictable delays**
- ❑ **Compile for one pipeline - run effectively on another**
- ❑ **Allows speculation**
 - Multiple branches
 - In-order commit
 - Precise exceptions
 - Time, energy; recovery
- ❑ **Significant increase in HW complexity**
- ❑ **Out-of-order execution, completion**
- ❑ **Register renaming**



Sandy bridge microarchitecture



Outline

- Reiteration
- Dynamic scheduling - Tomasulo
- Superscalar, VLIW
- Speculation
- **ILP limitations**
- What we have done so far



How much performance can be get from instruction-level parallelism?



A model of an idea processor

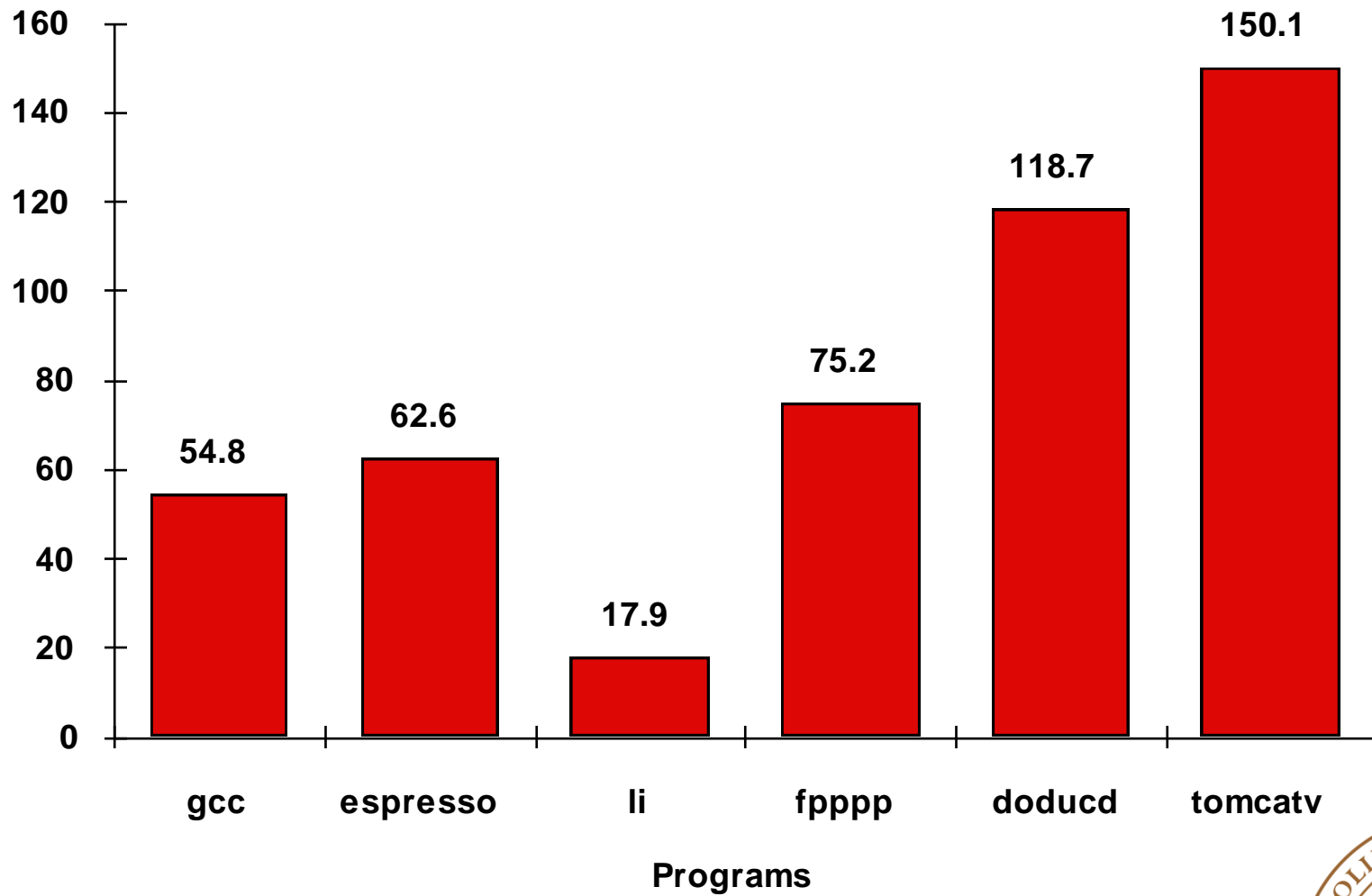
□ Assumptions for ideal/perfect machine to start:

- **Register renaming**: infinite virtual registers => all register WAW & WAR hazards are avoided
- **Branch prediction**: perfect; no mispredictions
- **Jump prediction**: all jumps perfectly predicted
- **2 & 3** => machine with perfect speculation & an unbounded buffer of instructions available
- **Memory-address alias analysis**: addresses are known & a store can be moved before a load provided addresses not equal, perfect caches;

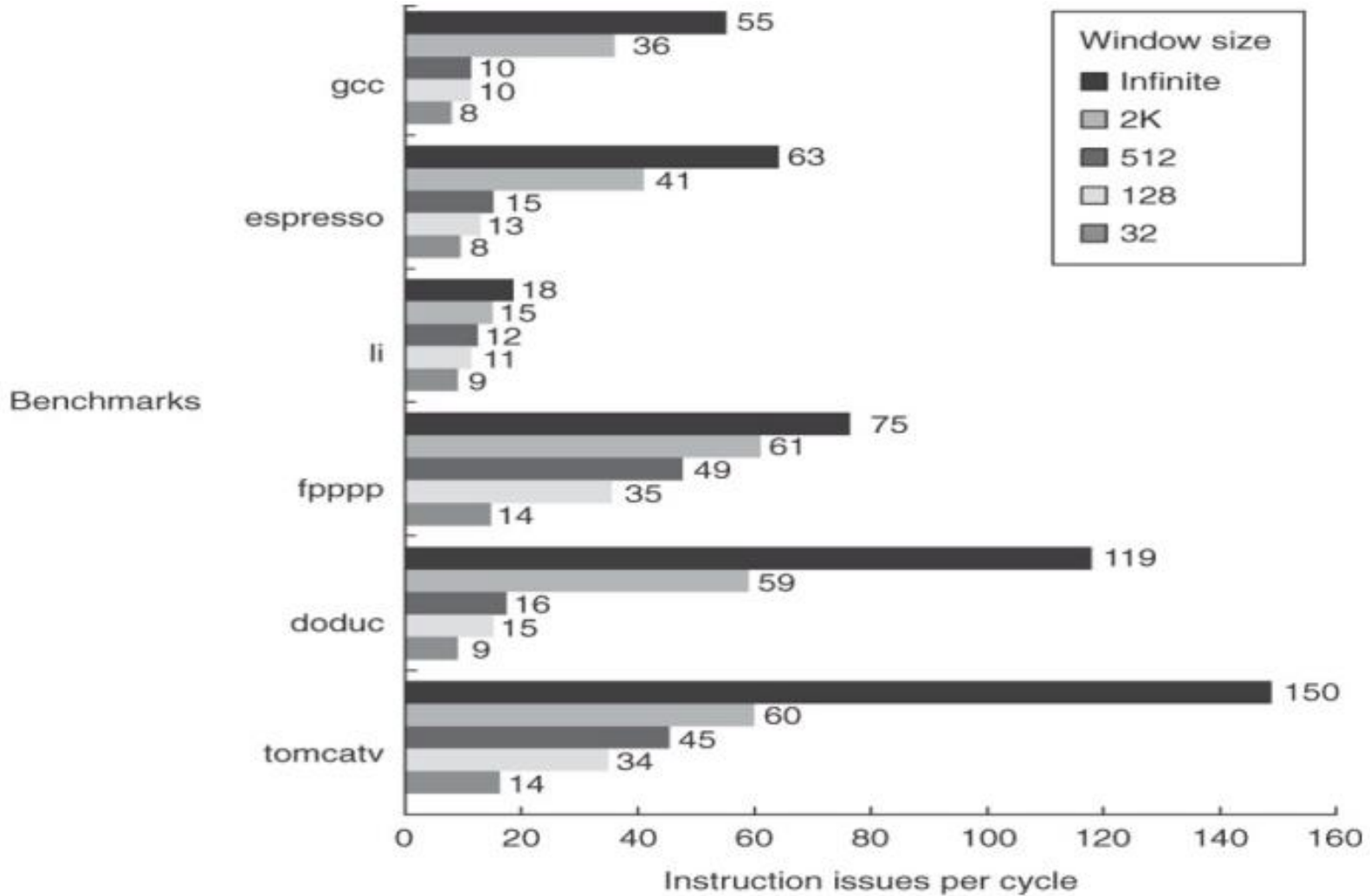
□ There are only true data dependencies left!



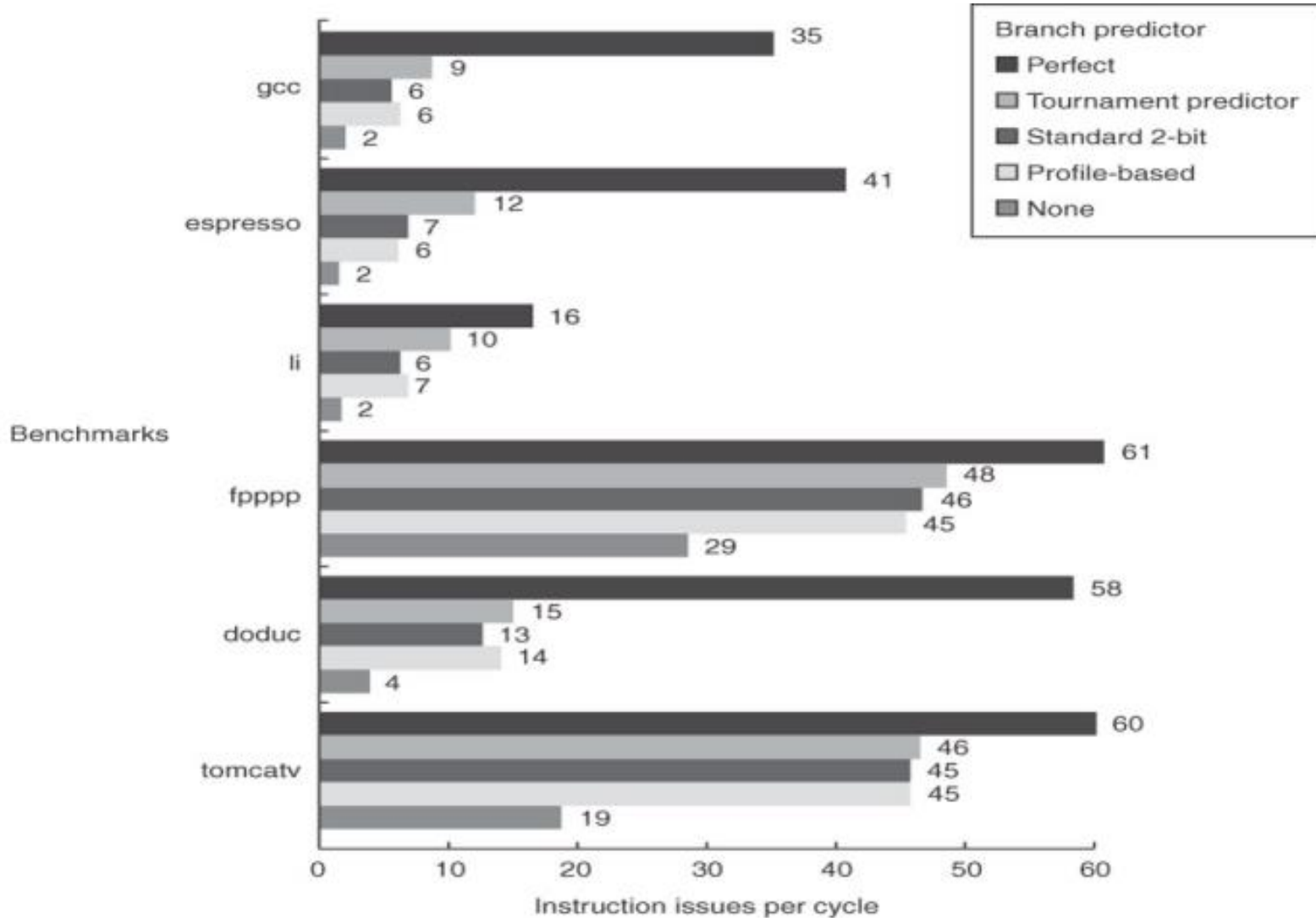
Upper limit to ILP



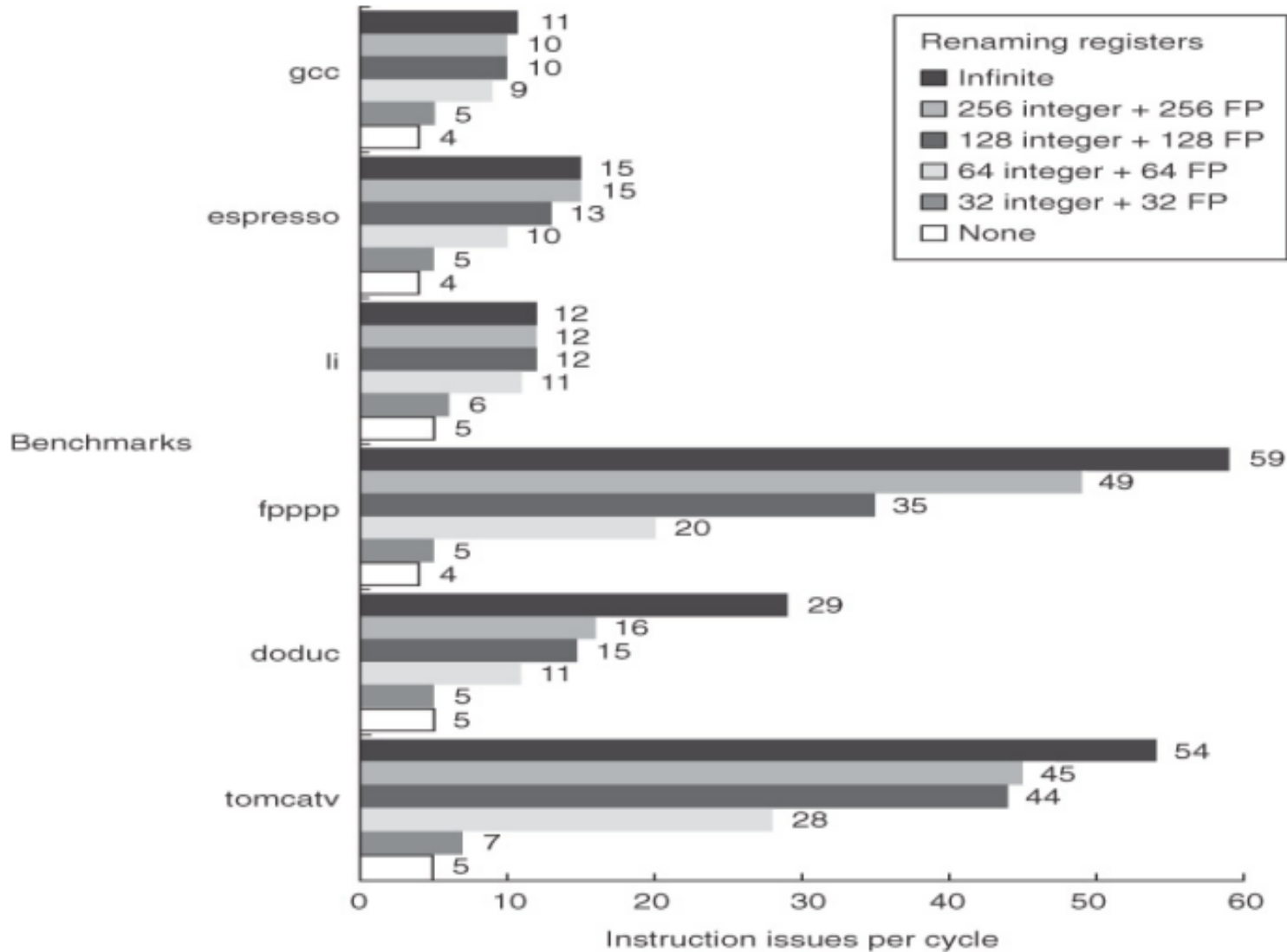
Window size impact



Branch impact



Register # impact



Summary

Software (*compiler*) tricks:

- Loop unrolling
- Static instruction scheduling (with *register renaming*)
- ... and more

Hardware tricks:

- Dynamic instruction scheduling
- Dynamic branch prediction
- Multiple issue – Superscalar, VLIW
- Speculative execution
- ... and more



Outline

- Reiteration
- Dynamic scheduling - Tomasulo
- Superscalar, VLIW
- Speculation
- ILP limitations
- **What we have done so far**

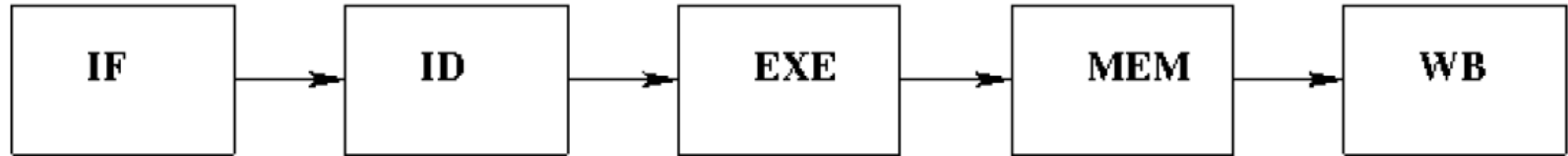


Summary pipeline - method

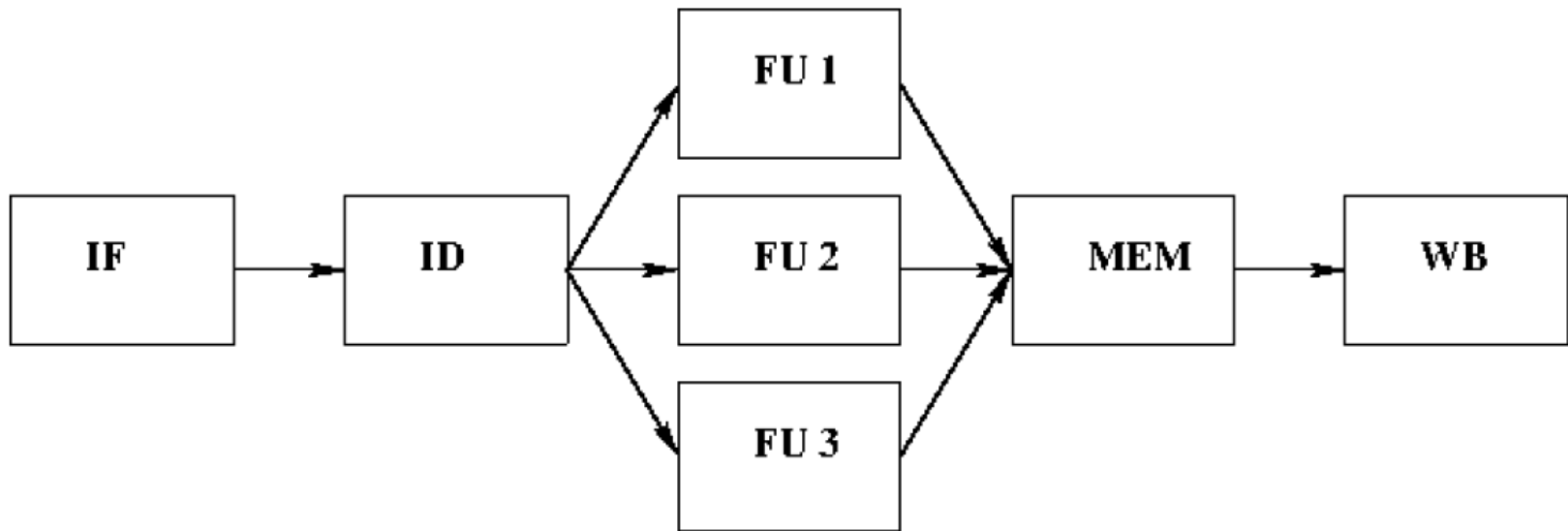
| Dependency | Hazard | Method |
|--------------------|---------------|---|
| Data | RAW | Forwarding, Scheduling, |
| Name | WAR, WAW | Register Renaming |
| Control | Control | Branch Prediction, Speculation, Delayed branch |
| Precise exceptions | | in-order commit |
| ILP | | Scheduling, Loop unrolling |



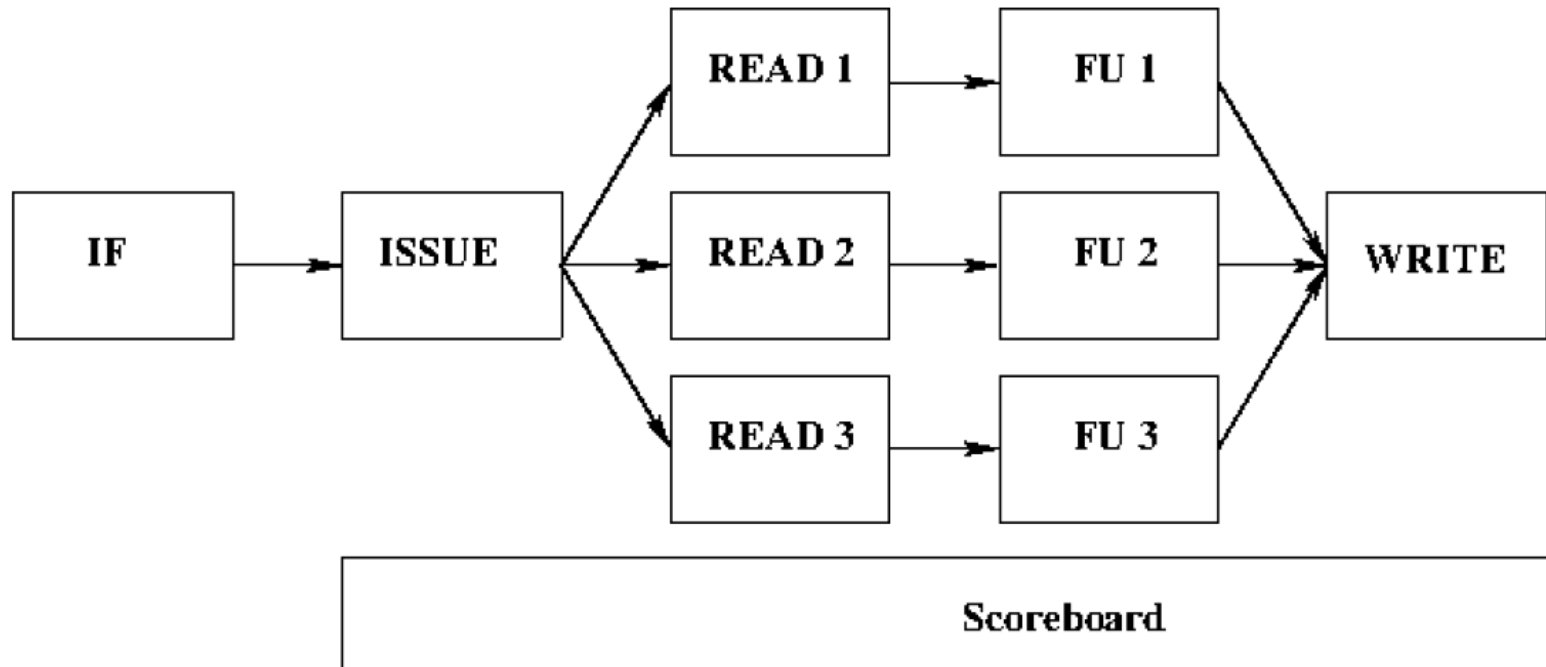
Basic 5-stage pipeline



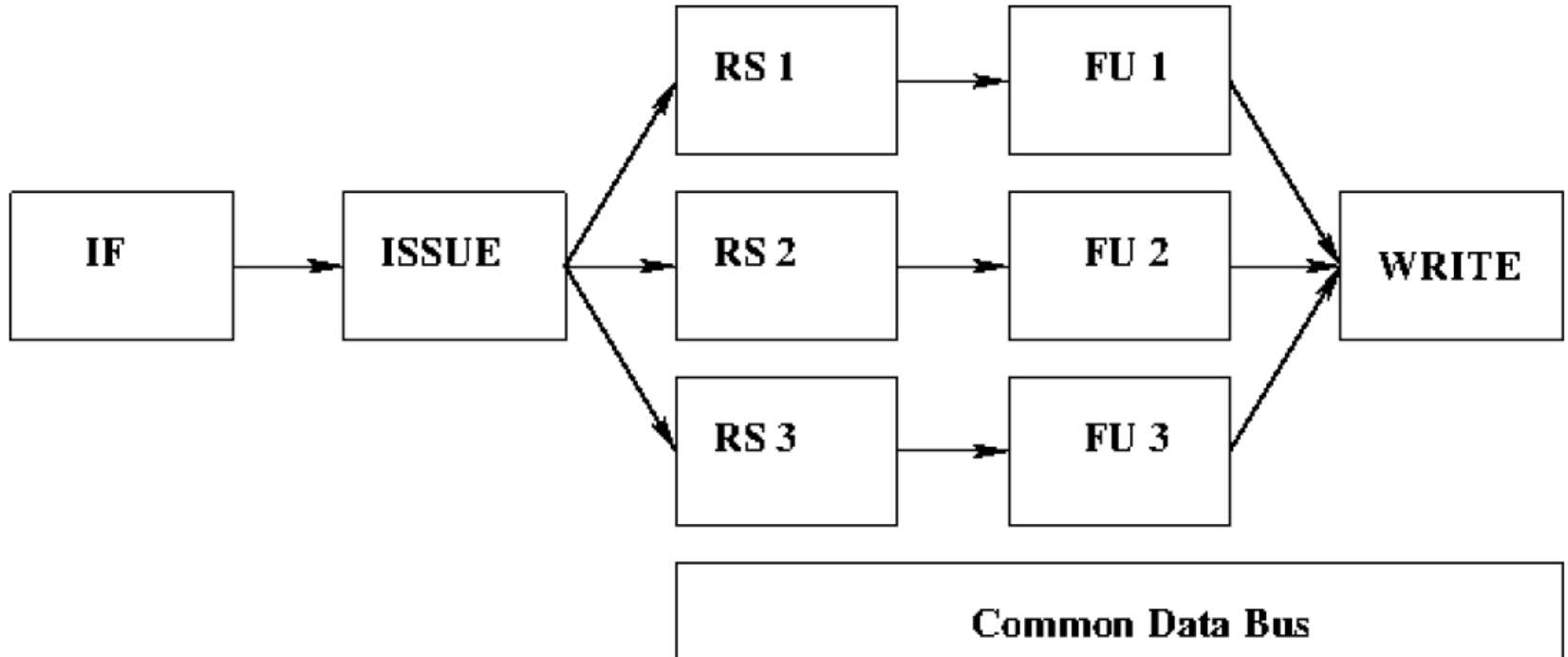
Pipeline with several FUs



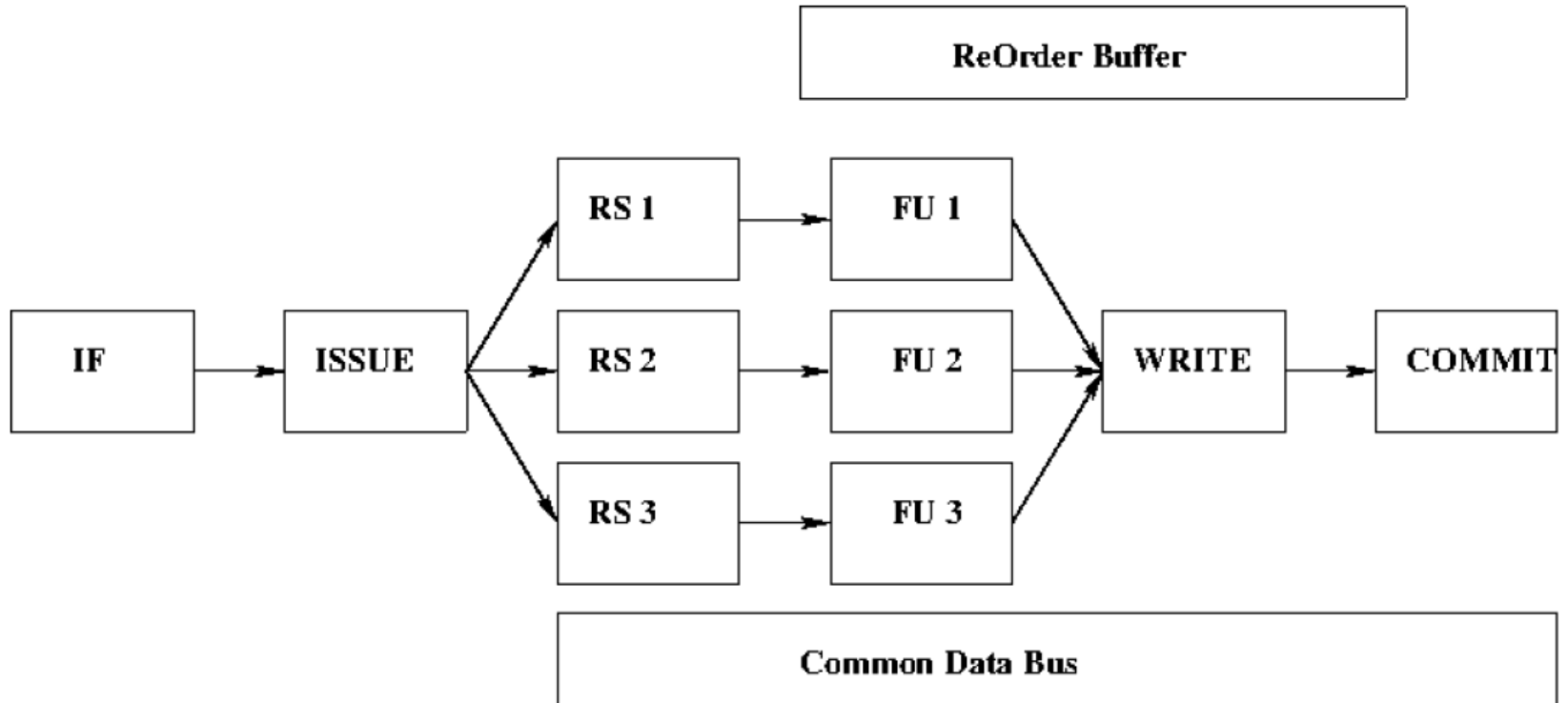
Scoreboard pipeline



Tomasulo pipeline



Tomasulo pipeline with speculation



Summary pipeline - implementation

| Problem | Simple | Scoreboard | Tomasulo | Tomasulo + Speculation |
|-------------------|--------------------|--------------------|---------------------|------------------------|
| | Static Sch | Dynamic Scheduling | | |
| RAW | forwarding stall | wait (Read) | CDB stall | CDB stall |
| WAR | - | wait (Write) | Reg. rename | Reg. rename |
| WAW | - | wait (Issue) | Reg. rename | Reg. rename |
| Exceptions | precise | ? | ? | precise, ROB |
| Issue | in-order | in-order | in-order | in-order |
| Execution | in-order | out-of-order | out-of-order | out-of-order |
| Completion | in-order | out-of-order | out-of-order | in-order |
| Structural hazard | - | many FU stall | many FU, CDB, stall | many FU, CDB, stall |
| Control hazard | Delayed br., stall | Branch prediction | Branch prediction | Br. pred, speculation |

