CS510 Computer Architecture

Lecture 11: Speculation and Multiple Instruction Issue

Soontae Kim

Spring 2017

School of Computing, KAIST

Notice

- **Homework assignment#2**
  - Due on April 19 (Wednesday)
- **Midterm exam**
  - Officially scheduled on April 17 but move to 19??
- **Term project proposal**
  - April 28 (Friday)
  - Prepare less than 10 slides in 5 minutes.
  - All team members must prepare parts of presentation

Dynamic Hardware-Based Speculation

(Speculative Execution Processors)

dynamic branch prediction
- prediction만 하는 것
speculation
- 실제 실행

- **Combines:**
  - **Dynamic hardware-based branch prediction**
  - **Dynamic Scheduling: issue instructions in order and execute out of order. (Tomasulo)**
  - **Speculation to allow the execution of instructions in the predicted branch direction, before control dependences are resolved.**
    - This overcomes the ILP limitations of the basic block size.
    - Creates dynamically speculated instructions at run-time with no ISA/compiler support at all.
    - If a branch turns out as mispredicted, all such dynamically speculated instructions must be prevented from changing the state of the machine (ISA registers, memory).

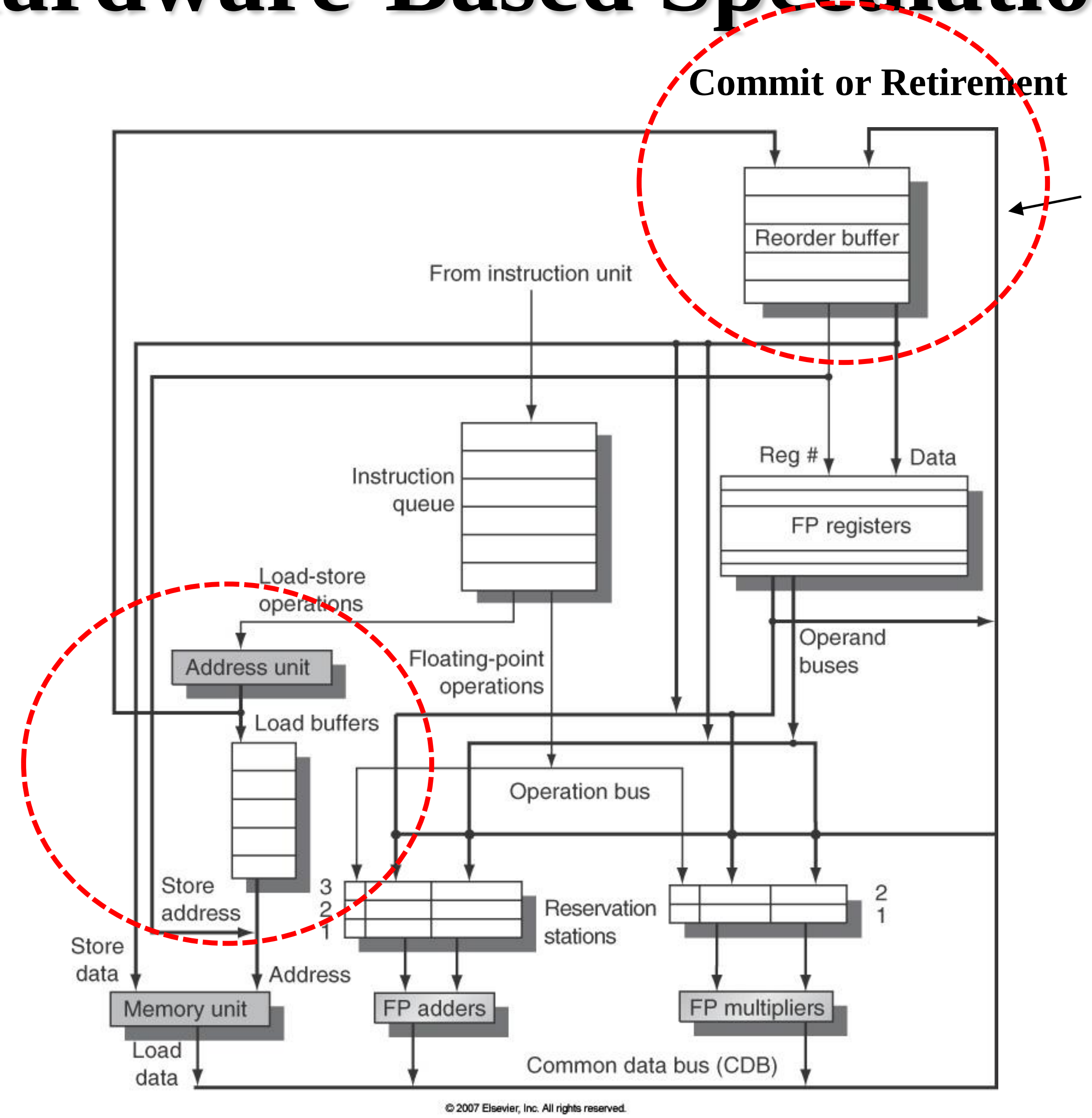
when we update register & memory, we have no way to recover that previous value

How?
- **Separate bypassing of results from actual completion of an instruction**
- **Addition of commit (retire, completion, graduation, or re-ordering) stage and forcing instructions to commit in their program order (i.e to write results to registers or memory in program order).**
  - **Precise exceptions are possible since instructions *must commit in order*.**

(in order processor)

Hardware-Based Speculation

Commit or Retirement



© 2007 Elsevier, Inc. All rights reserved.

ROB usually implemented as a circular buffer

tomasulo과의 차이점
CDB is not broadcast to FP registers and broadcast to reorder buffer

store buffer integrate into reorder buffer

Four Steps of Speculative Tomasulo Algorithm

1. <u>Issue</u> — (In-order) Get an instruction from Instruction Queue

   If <u>a RS</u> and <u>a ROB slot</u> are free, issue the instruction; send to the RS its operands, if available in ROB or register file, and also the ROB number allocated to this instruction (sometimes called "dispatch")

2. <u>Execution</u> — (out-of-order) When both operands are ready, execute; if not ready, watch CDB for result; when both operands are in RS, execute; checks RAW (sometimes called "issue")

   ROB number을 가지고 match시킨다. (기존 tomasulo는 reservation number을 가지고 match함)

3. <u>Write result</u> — (out-of-order) Write on CDB to all waiting RSs & ROB; mark the RS as available. Store value is written into ROB when available.

4. <u>Commit</u> — (In-order) Update ISA registers and memory with ROB result

   – When an instruction is at head of ROB & its result is present, update a register with the result in ROB (or memory for stores) and remove the instruction from ROB (sometimes called graduation or completion).

   – <u>A mispredicted branch at the head of ROB flushes ROB</u> (cancels the speculated instructions after the branch)

   ⟹ Instructions <u>issue in order</u>, execute (EX), write result (WB) out of order, but <u>must commit in order</u>.

Speculative Tomasulo vs. Tomasulo

- ROB entry
  - Inst. Type (branch, store, or register)
  - Destination (reg. # or mem. addr)
  - Value (result)
  - Ready (value ready?)
- Store buffer is integrated into ROB
- ROB performs register renaming
  - Tag matching using ROB number instead of RS number used in Tomasulo
- Who supplies operand values?
  - CDB on write result
  - ROB after write result and before commit
  - Register file or memory after commit

Evolution of Processor Performance

So far we examined static & dynamic techniques to improve the performance of single-issue (scalar) pipelined CPU designs including: static & dynamic scheduling, static & dynamic branch prediction. Even with these improvements, the restriction of issuing a single instruction per cycle still limits the ideal CPI = 1

Multi-cycle → ← Pipelined (single issue) ← Multiple Issue (CPI <1) Superscalar/VLIW/SMT

	1970-1980	1980-1990	1990-2000	2000-2010
Transistor Count	2K-100K	100K-1M	1M-100M	100M-1B
Clock Frequency	0.1-3MHz	3-30MHz	30M-1GHz	**1 GHz to 4 GHz**
Instruction/Cycle	< 0.1	0.1-0.9	0.9- 1.9	1.9-2.9 (?)
CPI	**> 10**	**1.1-10**	**0.5 - 1.1**	**.35 - .5 (?)**

IPC

Source: John P. Chen, Intel Labs

We next examine the two approaches to achieve a CPI < 1 by issuing multiple instructions per cycle:

- **Superscalar CPUs**
- **Very Long Instruction Word (VLIW) CPUs.**

Single-issue Processor = Scalar Processor
Instructions Per Cycle (IPC) = 1/CPI

Multiple Instruction Issue: CPI < 1

- To improve a pipeline's CPI to be less than one, and to better exploit Instruction Level Parallelism (ILP), a number of instructions have to be *issued* in the same cycle.

Most common = 4 instructions/cycle called 4-way superscalar processor

- Multiple instruction issue processors are of two types:
  - Superscalar: A number of instructions (0-8) are issued in the same cycle, scheduled statically by the compiler or more commonly dynamically (Tomasulo).
    - PowerPC, Sun UltraSparc, Alpha, HP 8000, Intel PII, III, 4 ...
  - VLIW (Very Long Instruction Word):

    A fixed number of instructions (3-6) are formatted as one long (independent) instruction word or packet (statically scheduled by the compiler).
    - Example: Explicitly Parallel Instruction Computer (EPIC)
      - Originally a joint HP/Intel effort.
      - ISA: Intel Architecture-64 (IA-64) 64-bit address:
      - First CPU: Itanium, Q1 2001. Itanium 2 (2003)
- Limitations of the approaches:
  - Available ILP in the program (both).
  - Specific hardware implementation difficulties (superscalar).
  - VLIW optimal compiler design issues.

Multiple Instruction Issue:

Superscalar Vs. VLIW

- Usually dynamically scheduled (Tomasulo)
- Complex scheduling hardware.
- Smaller code size.
  - Fewer registers
- Binary compatibility across generations of hardware.

- Statically scheduled.
- Complex compiler design.
- Simplified Hardware for decoding, issuing instructions.
- More ISA registers (no register renaming) , but simplified hardware for register ports.
- Usually no binary compatibility across generations

Multiple-Issue Processors

name	Issue structure	Hazard detection	Scheduling	Distinguishing characteristic	Examples
Superscalar (static)	dynamic	h/w	static	in-order execution	MIPS, ARM
Superscalar (dynamic)	dynamic	h/w	dynamic	some out-of-order execution	none
Superscalar (speculative)	dynamic	h/w	dynamic w/ speculation	out-of-order execution w/ speculation	Pentium 4, MIPS R12K, Alpha 21264,
VLIW/LIW	static	s/w	static	no hazards between issue packets	Trimedia, i860
EPIC	mostly static	s/w	mostly static	explicit dependences marked by compiler	Itanium

Simple Statically Scheduled Superscalar Pipeline

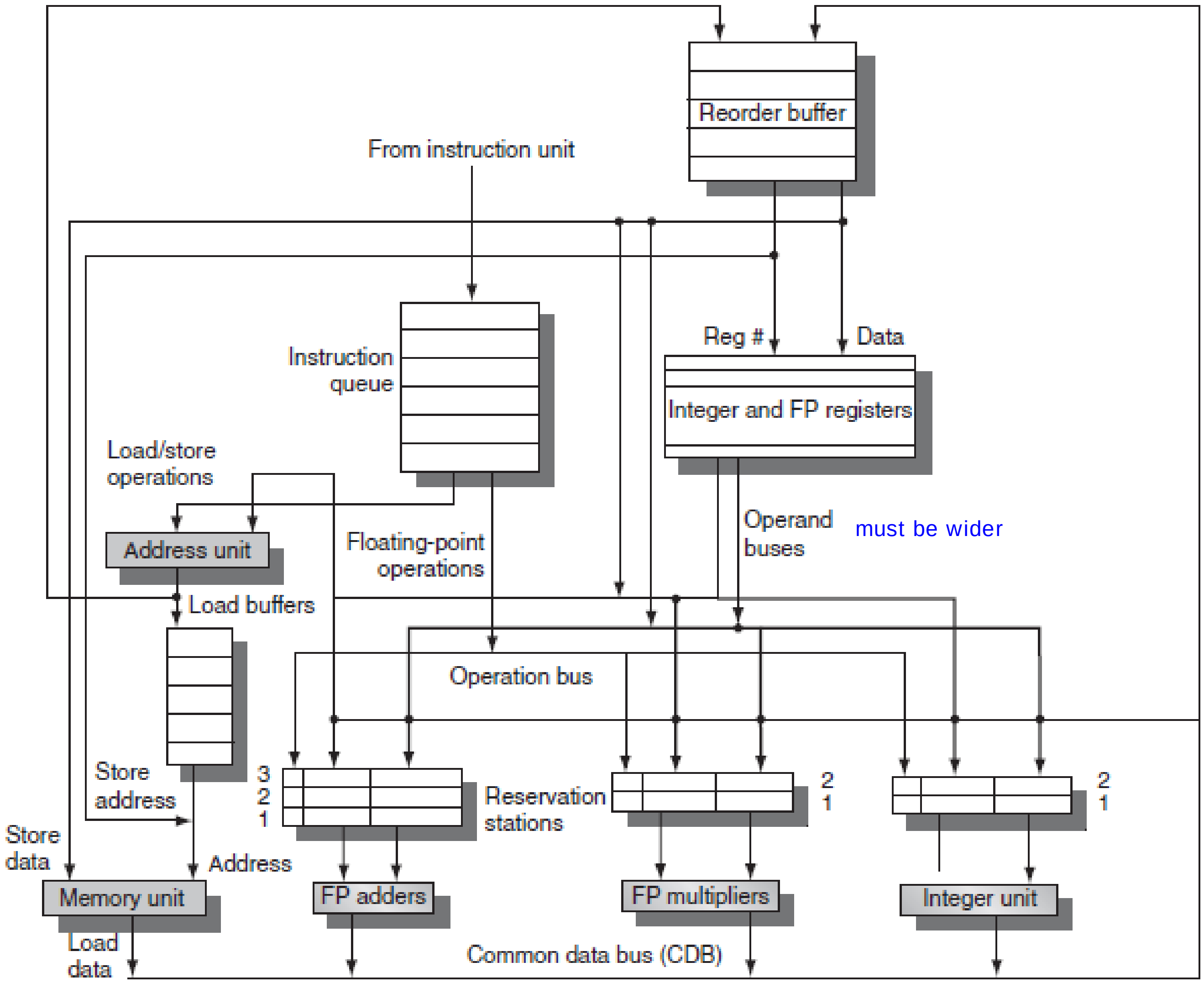
- Two instructions can be issued per cycle (**static two-issue or 2-way superscalar**).
- One of the instructions is integer (including load/store, branch). The other instruction is a floating-point operation.
  - This restriction reduces the complexity of hazard checking.
- Hardware must fetch and decode two instructions per cycle.
- Then it determines whether zero (a stall), one or two instructions can be issued (in decode stage) per cycle.

Instruction Type	1	2	3	4	5	6	7	8
Integer Instruction	IF	ID	EX	MEM	WB			
FP Instruction	IF	ID	EX	EX	EX	WB		
Integer Instruction		IF	ID	EX	MEM	WB		
FP Instruction		IF	ID	EX	EX	EX	WB	
Integer Instruction			IF	ID	EX	MEM	WB	
FP Instruction			IF	ID	EX	EX	EX	WB
Integer Instruction				IF	ID	EX	MEM	WB
FP Instruction				IF	ID	EX	EX	EX

Two-issue statically scheduled pipeline in operation
FP instructions assumed to be adds (EX takes 3 cycles)

Instructions assumed independent (no stalls)

Ideal CPI = 0.5 Ideal Instructions Per Cycle (IPC) = 2



must be wider

must be wider

Figure 3.17 The basic organization of a multiple issue processor with speculation. In this case, the organization could allow a FP multiply, FP add, integer, and load/store to all issues simultaneously (assuming one issue per clock per functional unit). Note that several datapaths must be widened to support multiple issues: the CDB, the operand buses, and, critically, the instruction issue logic, which is not shown in this figure. The last is a difficult problem, as we discuss in the text.

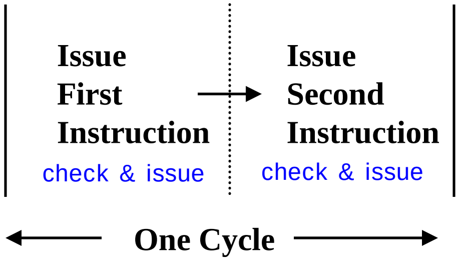
Superscalar Dynamic Scheduling

Three techniques can be used to support multiple instruction issue in Tomasulo without putting restrictions on the type of instructions issued per cycle:

<scalability issue ! : 2이상 issue 불가능 >

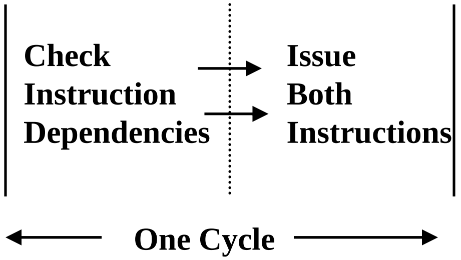
1 **Issue at a higher clock rate so that issue remains in order.**

- **For example for a 2-Issue supercalar issue at 2X Clock Rate.**



2 **Widen the issue logic to handle multiple instruction issue**

- **All possible dependences between instructions to be issued are detected at once and the result of the multiple issue matches in-order issue**



2-Issue superscalar

0, 1 or 2 instructions issued per cycle for either method

Superscalar Dynamic Scheduling

3 **To avoid increasing the CPU clock cycle time in the last two approaches, multiple instruction issue can be spilt into <u>two pipelined issue stages</u>:**

- **<u>Issue Stage One:</u> Decide how many instructions can issue simultaneously checking dependencies within the group of instructions to be issued + available RSs, <u>ignoring instructions already issued</u>.**
- **<u>Issue Stage Two:</u> Examine hazards among the selected instructions from the group and the those already issued.**

- **This approach is usually used in dynamically-scheduled wide superscalars that can issue four or more instructions per cycle.**
- **Splitting the issue into two pipelined stages increases the CPU pipeline depth and increases branch penalties**
  - **This increases the importance of accurate dynamic branch prediction methods.**
- **Further pipelining of issue stages beyond <u>two stages</u> may be necessary as CPU clock rates are increased.**

Superscalar Architecture Limitations:

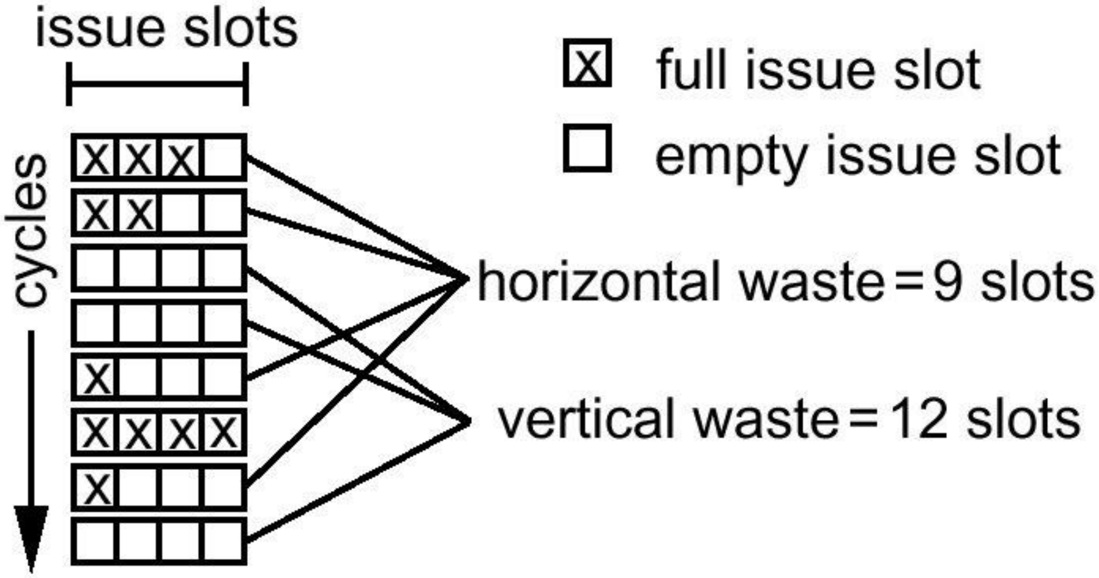
Issue Slot Waste Classification

- **Empty or wasted issue slots can be defined as either vertical waste or horizontal waste:**
  - **<u>Vertical waste</u> is introduced when the processor issues no instructions in a cycle.**
  - **<u>Horizontal waste</u> occurs when not all issue slots can be filled in a cycle.**

Example:

4-Issue Superscalar

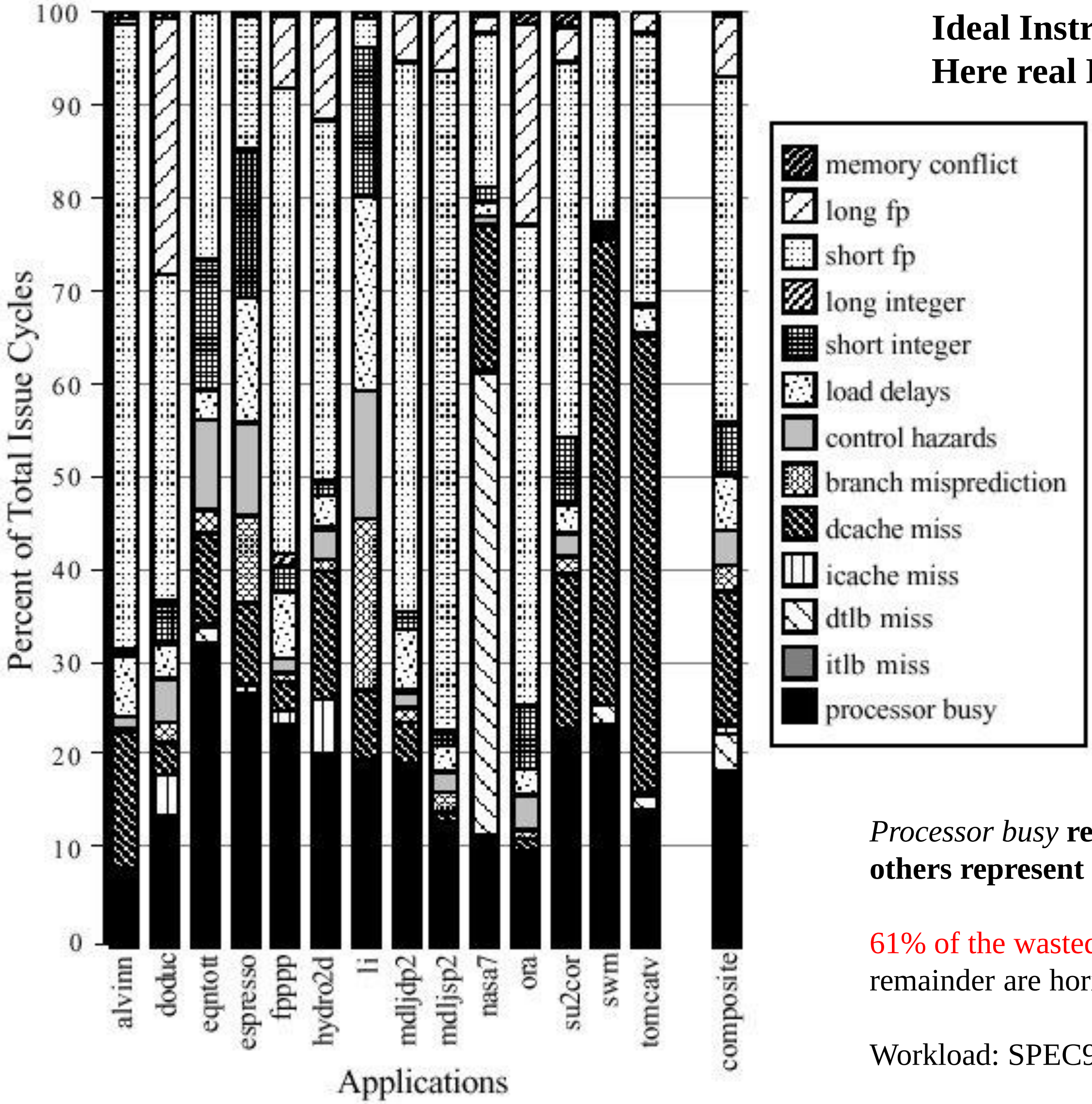
Ideal IPC =4
Ideal CPI = .25



Also applies to VLIW

<u>Result of issue slot waste</u>: Actual Performance << Peak Performance

Sources of Unused Issue Cycles in an 8-issue Superscalar Processor.



Ideal Instructions Per Cycle, IPC = 8
Here real IPC about 1.5

Real IPC << Ideal IPC

1.5 << 8

Processor busy **represents the utilized issue slots; all others represent wasted issue slots.**

61% of the wasted cycles are vertical waste, the remainder are horizontal waste.

Workload: SPEC92 benchmark suite.

Source: Simultaneous Multithreading: Maximizing On-Chip Parallelism Dean Tullsen et al.,
Proceedings of the 22nd Annual International Symposium on Computer Architecture, June 1995, pages 392-403.

Mulithreaded Execution for Servers

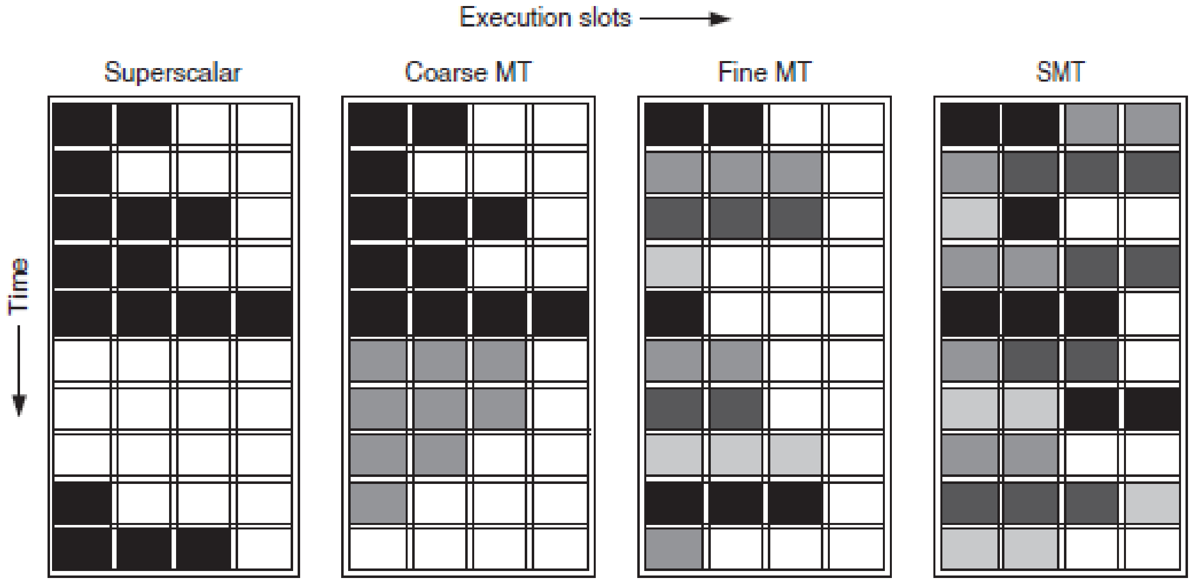
- **Thread: process with own instructions and data**
  - thread may be a process, part of a parallel program of multiple processes, or it may be an independent program
  - Each thread has all the state (instructions, data, PC, register state, and so on) necessary to allow it to execute
- **Multithreading: multiple threads to share the functional units of 1 processor via overlapping**
  - processor must duplicate indepedent state of each thread e.g., a separate copy of register file and a separate PC
  - memory shared through the virtual memory mechanisms
- **Threads execute overlapped, often interleaved**
  - When a thread is stalled, perhaps for a cache miss, another thread can be executed, improving throughput

Multithreaded Example: IBM AS/400

- **IBM Power III processor, “ Pulsar”**
  - PowerPC microprocessor that supports 2 IBM product lines: the RS/6000 series and the AS/400 series
  - Both aimed at commercial servers and focus on throughput in common commercial applications
  - such applications encounter high cache and TLB miss rates and thus degraded CPI
- **include a multithreading capability to enhance throughput and make use of the processor during long TLB or cache-miss stall**
- **Pulsar supports 2 threads: little clock rate, silicon impact**
- **Thread switched only on long latency stalls**

Simultaneous Multithreading (SMT)

- **Simultaneous multithreading (SMT): insight that dynamically scheduled processor already has many HW mechanisms to support multithreading**
  - large set of virtual registers that can be used to hold the register sets of independent threads (assuming separate renaming tables are kept for each thread)
  - out-of-order completion allows the threads to execute out of order, and get better utilization of the HW


Execution slots
Superscalar
Coarse MT
Fine MT
SMT
Time