**Multiple Issue**

\[
\text{CPI} = \text{CPI}_{\text{ideal}} + \text{Stalls}_{\text{structural}} + \text{Stalls}_{\text{RAW}} + \text{Stalls}_{\text{WAR}} + \text{Stalls}_{\text{WAW}} + \text{Stalls}_{\text{control}}
\]

Προσοχή να διατηρουνταί
1. Data flow
2. Exception Behavior

Έχουμε μελετήσει θα μελετήσουμε σήμερα

**Δυναμικές δρομολόγησης εντολών (hardware)**
- Scoreboard (ελάττωση RAW stalls)
- Register Renaming
  α) Tomasulo (ελάττωση WAR και WAW stalls)
  β) Reorder Buffer
- Branch prediction (ελάττωση Control stalls)
- Multiple Issue (CPI < 1)
- Multithreading (CPI < 1)

**Στατικές (shoftware/compiler)**
- Loop Unrolling
- Software Pipelining
- Trace Scheduling
Common way of Designing any Architecture

- Networking, multi-core processor, single processor, virtually any design:
  - **Broadcasting**: Use Common Data Bus or Point to point! For example CDM in Tomasulo.
  - **Asynchronous communication** between “processing stages” with different throughputs (a processing stage can be a whole system, for example router, switch, processor, or a simple block, for example IF, ID stages): Use Elastic Buffer & Flow Control. For example instruction buffer, reservation stations and reorder buffer.
  - **Faster clock**: Pipelining. Split a stage in multiple stages. For example split Issue stage(supерpipelining).
  - **Higher Throughput**: Parallel processing. For example superscalar.
  - **Less Latency**: Forwarding/Bypassing

- A processor is a sophisticated design that follows the “unwritten” design rules every architect should follow.
Multithreading

• Difficult to continue to extract ILP from a single thread
• Many workloads can make use of thread-level parallelism (TLP)
  – TLP from multiprogramming (run independent sequential jobs)
  – TLP from multithreaded applications (run one job faster using parallel threads)
• Multithreading uses TLP to improve utilization of a single processor
Pipeline Hazards

- Each instruction may depend on the next

What can be done to cope with this?
Solution with Multithreading

How can we guarantee no dependencies between instructions in a pipeline?

-- One way is to interleave execution of instructions from different program threads on same pipeline

Interleave 4 threads, T1-T4, on non-bypassed 5-stage pipe

T1: LW r1, 0(r2)
T2: ADD r7, r1, r4
T3: XORI r5, r4, #12
T4: SW 0(r7), r5
T1: LW r5, 12(r1)

Prior instruction in a thread always completes write-back before next instruction in same thread reads register file
Multithreaded DLX

- Have to carry thread select down pipeline to ensure correct state bits read/written at each pipe stage
- Appears to software (including OS) as multiple, albeit slower, CPUs
Multithreading Cost

• Each thread requires its own user state. Many CPU resources are split or shared!
  – PC
  – GPRs & Physical/HW registers
  – Prefetch & Instruction buffers
  – Reorder buffer
  – Load/Store buffer
  – Issue buffers

• Also, needs its own system state
  – virtual memory page table base register
  – exception handling registers

• Other costs?
• Watch out for performance when executing in Single Thread (ST) mode…
Thread Scheduling Policies

• Fixed interleave (*CDC 6600 PPU*s, 1964)
  – each of N threads executes one instruction every N cycles
  – if thread not ready to go in its slot, insert pipeline bubble

• Software-controlled interleave (*TI ASC PPU*s, 1971)
  – OS allocates S pipeline slots amongst N threads
  – hardware performs fixed interleave over S slots, executing whichever thread is in that slot

• Hardware-controlled thread scheduling (*HEP, 1982*) (*Power 5*)
  – hardware keeps track of which threads are ready to go
  – picks next thread to execute based on hardware priority scheme
Fine-Grain Multithreading

- Fine-grain multithreading switches processor context every thread cycle
- Context belongs to same address space
HW Multithreading alternatives: Fine-Grain Multithreading

Fine-Grain Multithreading

Switch every clock cycle

- Need fast HW switch between contexts
  - Multiple PCs and register files
  - Alternatively, thread ID attached to each GP register
  - Implemented with round-robin scheduling, skipping stalled threads
- Hides both short and long stalls
- Delays all threads, even if they have no stalls
HW Multithreading alternatives: Fine-Grain Multithreading

Coarse-Grain Multithreading

- Coarse-grain multithreading switches processor context upon long-latency event
- Context may belong to different address space

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Coarse-grain multithreading</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Processor context</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Thread-1</td>
<td>Thread-1</td>
<td>Thread-1</td>
<td>Thread-2</td>
<td>Thread-2</td>
<td>Thread-3</td>
<td>Thread-3</td>
<td>Thread-3</td>
</tr>
<tr>
<td>Cache miss-1</td>
<td></td>
<td></td>
<td>Syscall-2</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
HW Multithreading alternatives: Coarse-Grain Multithreading

Coarse-Grain Multithreading

Switch upon long upon long-latency events

- Can afford slower context switch than fine-grain multithreading
- Threads are not slowed down
  - Other thread runs when current thread stalls
- Pipeline startup cost upon thread switching
  - Processor issues instructions from one thread (address space)
Techniques presented so far have all been “vertical” multithreading where each pipeline stage works on one thread at a time.

SMT uses fine-grain control already present inside an OoO superscalar to allow instructions from multiple threads to enter execution on same clock cycle. Gives better utilization of machine resources.
For most apps, most execution units lie idle in an OoO superscalar.

For an 8-way superscalar.

Superscalar Machine Efficiency

Issue width

Instruction issue

Completely idle cycle (vertical waste)

Partially filled cycle, i.e., IPC < 4 (horizontal waste)
What is the effect of cycle-by-cycle interleaving?
- removes vertical waste, but leaves some horizontal waste
Chip Multiprocessing (CMP)

• What is the effect of splitting into multiple processors?
  – reduces horizontal waste,
  – leaves some vertical waste, and
  – puts upper limit on peak throughput of each thread => single thread execution is slower
Ideal Superscalar Multithreading: SMT

[Tullsen, Eggers, Levy, UW, 1995]

- Interleave multiple threads to multiple issue slots with no restrictions
O-o-O Simultaneous Multithreading
[Tullsen, Eggers, Emer, Levy, Stamm, Lo, DEC/UW, 1996]

- Add multiple contexts and fetch engines and allow instructions fetched from different threads to issue simultaneously
- Utilize wide out-of-order superscalar processor issue queue to find instructions to issue from multiple threads
- OOO instruction window already has most of the circuitry required to schedule from multiple threads
- Any single thread can utilize whole machine

**Shared HW mechanisms**

- Large set of virtual registers can hold register sets of independent threads
- Renaming provides unique register identifiers to different threads
- Out-of-order completion of instructions from different threads allowed
  - No cross-thread RAW, WAW, WAR hazards
  - Separate reorder buffer per thread
Summary: Multithreaded Categories

- **Superscalar**
- **Fine-Grained**
- **Coarse-Grained**
- **Multiprocessing**
- **Simultaneous Multithreading**

Time (processor cycle):

- Thread 1
- Thread 2
- Thread 3
- Thread 4
- Thread 5
- Idle slot
Power 4

Single-threaded predecessor to Power 5. 8 execution units in out-of-order engine, each may issue an instruction each cycle.
Power 4

1. Instruction fetch
   - IF
   - IC
   - BP
   - D0
   - D1
   - D2
   - D3
   - Xfer
   - GD

2. Instruction crack and group formation

3. Out-of-order processing
   - BR
   - LD/ST
   - WB
   - Xfer
   - CP

4. Interrupts and flushes

Power 5

1. Instruction fetch
   - IF
   - IC
   - BP
   - D0
   - D1
   - D2
   - D3
   - Xfer
   - GD

2. Group formation and instruction decode

3. Out-of-order processing
   - MP
   - ISS
   - RF
   - EX
   - DC
   - Fmt
   - WB
   - Xfer
   - CP

4. Fixed-point pipeline
   - F6
   - Floating-point pipeline

5. Branch redirects

6. Load/store pipeline

7. Branch pipeline

8. 2 fetches (PC), 2 initial decodes

9. 2 commits (architected register sets)
Why only 2 threads? With 4, one of the shared resources (physical registers, cache, memory bandwidth) would be prone to bottleneck.
### Rename Registers and issue queue sizes

<table>
<thead>
<tr>
<th>Resource type</th>
<th>Logical size (per thread)</th>
<th>Physical size</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>POWER4</td>
</tr>
<tr>
<td>GPRs</td>
<td>32 (+4)</td>
<td>80</td>
</tr>
<tr>
<td>FPRs</td>
<td>32</td>
<td>72</td>
</tr>
<tr>
<td>CRs(^\d)</td>
<td>8 (+1) 4-bit fields</td>
<td>32</td>
</tr>
<tr>
<td>Link/count registers</td>
<td>2</td>
<td>16</td>
</tr>
<tr>
<td>FPSCR(^\d)</td>
<td>1</td>
<td>20</td>
</tr>
<tr>
<td>XER(^\d)</td>
<td>Four fields</td>
<td>24</td>
</tr>
<tr>
<td>Fixed-point and load/store issue queue</td>
<td>Shared by both threads</td>
<td>36</td>
</tr>
<tr>
<td>Floating-point issue queue</td>
<td>Shared by both threads</td>
<td>20</td>
</tr>
<tr>
<td>Branch execution issue queue</td>
<td>Shared by both threads</td>
<td>12</td>
</tr>
<tr>
<td>CR logical issue queue</td>
<td>Shared by both threads</td>
<td>10</td>
</tr>
</tbody>
</table>
Changes in Power 5 to support SMT

- Two separate program counters are used, one for each thread.
- Added per thread load and store queues. Added virtual entries.
- The size of the BIQ (Branch Information Queue) remains at 16 entries but split in two, with eight entries per thread.
- Added separate instruction prefetch and buffering per thread.
- Each logical register number has a thread bit appended and mapped as usual. Increased the number of physical registers from 152 to 240.
- Increased the size of FP issue queue.
- Shared global completion table (GCT). Two linked lists in order to in order commit instructions from the two threads.
- The Power5 core is about 24% larger than the Power4 core because of the addition of SMT support.
Power 5 thread performance ...

- Priority is set by SW and enforced by HW.
- Relative priority of each thread controllable in hardware.
- For balanced operation, both threads run slower than if they “owned” the machine.
- Hyper-Threading Technology is SMT introduced by Intel. HTT has two logical processors, with its own processor architectural state.
- HTT duplicates the architectural state but not the main execution resources.
- Transparent to OS: min. required is symmetric multiprocessing (SMP) support.
- SMP involves two or more identical processors connect to a single, shared main memory, all I/O devices, controlled by single OS.
Pentium-4 Hyperthreading (2002)

- First commercial SMT design (2-way SMT)
  - Hyperthreading == SMT
- Logical processors share nearly all resources of the physical processor
  - Caches, execution units, branch predictors
- Die area overhead of hyperthreading ~ 5%
- When one logical processor is stalled, the other can make progress
  - No logical processor can use all entries in queues when two threads are active
- Processor running only one active software thread runs at approximately same speed with or without hyperthreading
Pentium-4 Hyperthreading
Front End

L2 Access
Queue
Decode
Queue
Cache Fill
Uop Queue

Resource divided between logical CPUs

Resource shared between logical CPUs
Pentium-4 Hyperthreading Execution Pipeline

[ Intel Technology Journal, Q1 2002 ]
Initial Performance of SMT

Multi-program workloads

► Pentium 4 Extreme SMT achieves 1.01 speedup for SPECint_rate benchmark and 1.07 for SPECfp_rate
  ► Pentium 4 is dual-threaded SMT
  ► SPECRate requires that each SPEC benchmark be run against a vendor-selected number of copies of the same benchmark
► Running on Pentium 4 each of 26 SPEC benchmarks paired with every other (26^2 runs) speed-ups from 0.90 to 1.58; average is 1.20
► Power 5, 8 processor server 1.23 faster for SPECint_rate with SMT, 1.16 faster for SPECfp_rate
► Power 5 running 2 copies of each app speedup between 0.89 and 1.41 Most gained some FP apps had cache conflicts and least gains
Comparison between multiple-issue processors

<table>
<thead>
<tr>
<th>Processor</th>
<th>Microarchitecture</th>
<th>Fetch/issue/execute</th>
<th>Func. units</th>
<th>Clock rate (GHz)</th>
<th>Transistors and die size</th>
<th>Power</th>
</tr>
</thead>
<tbody>
<tr>
<td>Intel Pentium 4 Extreme</td>
<td>Speculative dynamically scheduled; deeply pipelined; SMT</td>
<td>3/3/4</td>
<td>7 int. 1 FP</td>
<td>3.8</td>
<td>125M 122 mm²</td>
<td>115 W</td>
</tr>
<tr>
<td>AMD Athlon 64 FX-57</td>
<td>Speculative dynamically scheduled</td>
<td>3/3/4</td>
<td>6 int. 3 FP</td>
<td>2.8</td>
<td>114M 115 mm²</td>
<td>104 W</td>
</tr>
<tr>
<td>IBM Power5 1 processor</td>
<td>Speculative dynamically scheduled; SMT; two CPU cores/chip</td>
<td>8/5/8</td>
<td>6 int. 2 FP</td>
<td>1.9</td>
<td>200M 300 mm² (estimated)</td>
<td>80 W</td>
</tr>
<tr>
<td>Intel Itanium 2</td>
<td>EPIC style; primarily statically scheduled</td>
<td>6/5/11</td>
<td>9 int. 2 FP</td>
<td>1.6</td>
<td>592M 423 mm²</td>
<td>130 W</td>
</tr>
</tbody>
</table>
Comparison between ILP processors

SPEC INT rate

The graph compares the SPEC INT rate of different processors, including Itanium 2, Pentium 4, AMD Athlon 64, and Power 5. The SPEC INT rate is measured for various benchmarks such as gzip, vpr, gcc, mips, cray, parser, econ, mk, perf, gap, vortex, bzip2, and twolf. The bars represent the performance of each processor in each benchmark, allowing for a comparison of their efficiency and speed.
Comparison between ILP processors

SPEC FP rate

![SPEC FP Rate Graph](image-url)
Measuring processor efficiency

Area- and power-efficiency

- Processor performance gain comes at an area/power budget cost
  - Weigh performance again against power and area increase
- Area-efficiency
  - Performance / transistor (e.g. SPECrate/million transistors)
- Power-efficiency
  - Performance / watt (e.g. SPECRate/watt)
Comparison between ILP processors

Power and area efficiency

[Graph showing comparison between Itanium 2, Pentium 4, AMD Athlon 64, and POWER 5 for SPECInt/M Transistors, SPECFP/M Transistors, SPECInt/mm\(^2\), SPECFP/mm\(^2\), SPECInt/Watt, and SPECFP/Watt]
Best ILP approach?

Results with commercial processors

- AMD Athlon most performance-efficient in INT programs
- Power5 most performance-efficient in FP programs
- Power5 most power-efficient overall
- Itanium VLIW least power-efficient and area-efficient overall

The power is proportional to the peak rate, but performance will be at the sustained rate.