Scalable Instruction Scheduler and an example of CPU design bug

Department of Electrical Engineering
National Cheng-Kung University

Chung-Ho Chen (陳中和)

Bug First (CPU IP Design)

- Reasons for bug
  - Coding error
    - Functional errors.
  - because of not knowing what for.
A context switch is the computing process of storing and restoring the state of a CPU (the context) such that multiple processes can share a single CPU resource.

The context switch is an essential feature of a multitasking operating system.
Context Switch: Steps

- The state of the first process must be saved. All the register/data that are necessary for state is stored in one data structure, called a *switchframe* or a process control block.

- Since the operating system has effectively suspended the execution of the first process, it can now load the switchframe and context of the second process. In doing so, the program counter from the switchframe is loaded, and thus execution can continue in the new process.
## Banked Registers

- **Speedup calls and interrupt services**

### Modes

<table>
<thead>
<tr>
<th>User</th>
<th>System</th>
<th>Supervisor</th>
<th>Abort</th>
<th>Undefined</th>
<th>Interrupt</th>
<th>Fast interrupt</th>
</tr>
</thead>
<tbody>
<tr>
<td>R0</td>
<td>R0</td>
<td>R0</td>
<td>R0</td>
<td>R0</td>
<td>R0</td>
<td>R0</td>
</tr>
<tr>
<td>R1</td>
<td>R1</td>
<td>R1</td>
<td>R1</td>
<td>R1</td>
<td>R1</td>
<td>R1</td>
</tr>
<tr>
<td>R2</td>
<td>R2</td>
<td>R2</td>
<td>R2</td>
<td>R2</td>
<td>R2</td>
<td>R2</td>
</tr>
<tr>
<td>R3</td>
<td>R3</td>
<td>R3</td>
<td>R3</td>
<td>R3</td>
<td>R3</td>
<td>R3</td>
</tr>
<tr>
<td>R4</td>
<td>R4</td>
<td>R4</td>
<td>R4</td>
<td>R4</td>
<td>R4</td>
<td>R4</td>
</tr>
<tr>
<td>R5</td>
<td>R5</td>
<td>R5</td>
<td>R5</td>
<td>R5</td>
<td>R5</td>
<td>R5</td>
</tr>
<tr>
<td>R6</td>
<td>R6</td>
<td>R6</td>
<td>R6</td>
<td>R6</td>
<td>R6</td>
<td>R6</td>
</tr>
<tr>
<td>R7</td>
<td>R7</td>
<td>R7</td>
<td>R7</td>
<td>R7</td>
<td>R7</td>
<td>R7</td>
</tr>
<tr>
<td>R8</td>
<td>R8</td>
<td>R8</td>
<td>R8</td>
<td>R8</td>
<td>R8</td>
<td>R8</td>
</tr>
<tr>
<td>R9</td>
<td>R9</td>
<td>R9</td>
<td>R9</td>
<td>R9</td>
<td>R9</td>
<td>R9</td>
</tr>
<tr>
<td>R10</td>
<td>R10</td>
<td>R10</td>
<td>R10</td>
<td>R10</td>
<td>R10</td>
<td>R10</td>
</tr>
<tr>
<td>R11</td>
<td>R11</td>
<td>R11</td>
<td>R11</td>
<td>R11</td>
<td>R11</td>
<td>R11</td>
</tr>
<tr>
<td>R12</td>
<td>R12</td>
<td>R12</td>
<td>R12</td>
<td>R12</td>
<td>R12</td>
<td>R12</td>
</tr>
<tr>
<td>R13</td>
<td>R13</td>
<td>R13_svc</td>
<td>R13_abt</td>
<td>R13_und</td>
<td>R13_irq</td>
<td>R13_fiq</td>
</tr>
<tr>
<td>R14</td>
<td>R14</td>
<td>R14_svc</td>
<td>R14_abt</td>
<td>R14_und</td>
<td>R14_irq</td>
<td>R14_fiq</td>
</tr>
<tr>
<td>PC</td>
<td>PC</td>
<td>PC</td>
<td>PC</td>
<td>PC</td>
<td>PC</td>
<td>PC</td>
</tr>
</tbody>
</table>

### CPSR

- **SPSR_spc**
- **SPSR_abt**
- **SPSR_und**
- **SPSR_irq**
- **SPSR_fiq**
Interpretation

- Multiple/Load/Sw

What does this mean?

STM (2)

This form of STM stores a subset (or possibly all) of the User mode general-purpose registers to sequential memory locations.

Syntax

STM (<cond>) <addressing_mode> <Rn>, <registers>

where:

<cond>
Is the condition under which the instruction is executed. The conditions are defined in The condition field on page A3-5. If <cond> is omitted, the AL (always) condition is used.

<addressing_mode>
Is described in Addressing Mode 4 - Load and Store Multiple on page A5-48. It determines the P and U bits of the instruction. Only the forms of this addressing mode with W == 0 are available for this form of the STM instruction.

<Rn>
Specifies the base register used by <addressing_mode>. If R15 is specified as the base register <Rn>, the result is UNPREDICTABLE.

<registers>
Is a list of registers, separated by commas and surrounded by { and }. It specifies the set of registers to be stored by the STM instruction.

The registers are stored in sequence, the lowest-numbered register to the lowest memory address (start_address), through to the highest-numbered register to the highest memory address (end_address).

For each of i=0 to 15, bit[i] in the register_list field of the instruction is 1 if Ri is in the list and 0 otherwise. If bits[15:0] are all zero, the result is UNPREDICTABLE.

If R15 is specified in <registers> the value stored is IMPLEMENTATION DEFINED. For more details, see Reading the program counter on page A2-7.

For an STM instruction, indicates that User mode registers are to be stored.
Special Form of Store/Load Multiple Instructions

STM{<cond>}{<addressing mode>} <Rn>, <regs>^  

When the processor is in privileged mode, this instruction stores a subset (or possibly all) of the User mode general-purpose registers to sequential memory locations.

LDM{<cond>}{<addressing mode>} <Rn>, <regs_without_pc>^  

When the processor is in privileged mode, this instruction loads a non-empty subset of the User mode general-purpose registers from sequential memory locations.
Instruction Scheduler Outline

- Introduction & Motivation
- Dynamic Scheduler for Superscalar Processors
- Wakeup Logic Designs
- Experimental Evaluation
- Conclusions
Outline

- Introduction & Motivation
- Dynamic Scheduler for Superscalar Processors
- Wakeup Logic Designs
- Experimental Evaluation
- Conclusions
Instruction Wake Up

- Wake up
- LD  R1, 100(R3)
- ADD, R2, R1, R4
- OR R2, R1, R6

Issue window

<table>
<thead>
<tr>
<th>I</th>
<th>LD</th>
</tr>
</thead>
<tbody>
<tr>
<td>J</td>
<td>ADD</td>
</tr>
<tr>
<td>K</td>
<td>OR</td>
</tr>
</tbody>
</table>
Power Consideration for Dynamic Scheduler

Power consumption associated with scheduler constitutes a significant portion of processor’s total power consumption.
Latency Consideration for Dynamic Scheduler

- Dynamic schedulers become increasingly complex.
- Wakeup and selection should be an atomic operation to avoid significant performance degradation.
- The complex scheduler becomes a burden of clock cycle time.
Motivation: Inefficient Wakeup Design

Matching all the tags in the issue window only to wake up a few instructions is inefficient in terms of both time and power.
Goal

- A scheduler comes with lower power consumption and faster operation speed.
- Scalable.
Presentation Outline

- Introduction & Motivation
- Dynamic Scheduler for Superscalar Processor
- Wakeup Logic Designs
- Experimental Evaluation
- Conclusions
Baseline Processor Model

The wakeup and select logics form the dynamic scheduler.

HP PA8000, Intel Pentium 4, MIPS R10000, Alpha 21264
Decoupled Processor Model

- Intel P6, PowerPC 604, HAL SPARC64.
- Wakeup is triggered by the result tags from the functional unit; successive instructions cannot be executed cycle by cycle.

![Diagram of decoupled processor model]

<table>
<thead>
<tr>
<th>Fetch</th>
<th>Decode &amp; Allocate</th>
<th>Rename</th>
<th>REG Read &amp; Insert</th>
<th>Wakeup &amp; Select</th>
<th>EXE &amp; Bypass</th>
<th>D Cache Access</th>
<th>Commit &amp; REG Write</th>
</tr>
</thead>
</table>
Presentation Outline

- Introduction & Motivation
- Dynamic Scheduler for Superscalar Processors
- **Wakeup Logic Designs**
  - CAM-Based Scheme
  - Bit-Map RAM Scheme
  - Proposed Schemes
- Experimental Evaluation
- Conclusions
Conventional CAM-Based Wakeup Design

This design employs a RAM structure and two CAM structures.

CAM structures match all the source tags with the result tags.

D. Folegnani, ISCA, 2001
D. Ernst, ISCA, 2002
I. Kim, ISCA, 2003
The Grant signals read the corresponding destination tags from the RAM structure.
Wakeup Process (2/3)

- Destination tags are read and forwarded to the CAM structure.
- All the source tags match with each destination tag.
Wakeup Process (3/3)

- If a match occurs, the corresponding ready bit is set.
- If both ready bits are set, a request is sent for execution.
A bit represents a data dependence between two instructions.

- LD R1, R2
- ADD x, R1,x
- SUB x, R1,x
Conventional RAM-Based Wakeup Design

Wakeup operation can be finished by reading a data from this RAM.

M. Goshima, *MICRO-34*, 2001
Proposed Wakeup Logic

- The grant lines directly select the matching circuit.
- Predecode the source tags for comparisons.
Basic Wakeup Unit

This circuit matches 5-bit tag with 32 lines of grant signal.

Only one decoded line is asserted to wait for the corresponding grant line.
Proposed Wakeup Logic for 128-Entry

128-entry wakeup logic is built up by using four lanes of the basic wakeup units.
Features of the Proposed Design (1/2)

- Compared to the CAM-based scheme
  - The proposed wakeup logic does not need to read the destination tags.
  - At most, only $w$ grant lines are driven rather than $x$ tag bits are driven. Where
    - $w = \text{issue width}$,
    - $x = \text{issue width} \times \text{tag length}$.
  - Only the match lines that depend on the execution result is active.
Features of the Proposed Design (2/2)

Compared to the RAM-based scheme

- The match circuits are separated from the tag cells so that
  - the area of the RAM cells only grows linearly with the increasing of the issue window size
  - while the area of the match circuits grows with the square of the window size.
Presentation Outline

- Introduction & Motivation
- Dynamic Scheduler for Superscalar Processors
- Wakeup Logic Designs
- Experimental Evaluation
  - Experimental Methodology
  - Results of Power, Timing, and Area
- Conclusions
Experimental Methodology

- Wattch (for power) & SimpleScalar (for IPC) were used for architectural simulation.
- Avant! Hspice was used for circuit-level simulation. (for latency)

<table>
<thead>
<tr>
<th>Dispatch, issue, commit width</th>
<th>4</th>
<th>8</th>
<th>16</th>
</tr>
</thead>
<tbody>
<tr>
<td>Issue window size/ LSQ size</td>
<td>32/ 8</td>
<td>64/ 16</td>
<td>128/ 32</td>
</tr>
<tr>
<td>Functional units</td>
<td>4 IALU, 1 IMUL, 2 FALU, 1 FMUL, 2 LSU</td>
<td>8 IALU, 2 IMUL, 4 FALU, 2 FMUL, 4 LSU</td>
<td>16 IALU, 4 IMUL, 8 FALU, 4 FMUL, 8 LSU</td>
</tr>
<tr>
<td>L1 I-cache/ L1 D-cache</td>
<td>4-way, 64KB, 32-Byte line, 1-cycle latency/ line, 1-cycle latency</td>
<td></td>
<td>4-way, 64KB, 32-Byte</td>
</tr>
<tr>
<td>L2 cache/ TLB</td>
<td>4-way, 256KB, 64-byte line, 10-cycle latency/</td>
<td>4-way, 128-entry, 4KB page size</td>
<td></td>
</tr>
<tr>
<td>Memory width and latency</td>
<td>64-bit wide, 75 cycle latency, 4-cycle burst</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Branch predictor</td>
<td>Combination of bimodal and 2-level global predictor/ 2048-entry bimodal 8-bit history, 2048-entry level 2 1024-entry chooser/ 4-way, 1024-entry BTB/ 16-entry RAS(return address stack)/ 8-cycle penalty</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Experimental Parameters

- **Wattch simulator**: 1 Ghz, 1.8 Vdd, 0.18um
  - For power model

- SPEC2000: 7 integer and 9 floating point benchmarks
Power Consumption Statistics (1/2)

- The power consumption is only 20% of the CAM scheme and 80% of the bit-map RAM scheme.
- 8-issue 64-entry processor

![Graph showing power consumption comparison between CAM, Bit map RAM, and Proposed design for various benchmarks.]
The proposed design consumes only 14% of CAM scheme and 45% of the bit map RAM scheme for 16-issue 128-entry processor.

Improve more for wider issue and larger window

<table>
<thead>
<tr>
<th></th>
<th>4-issue 32-entry processor</th>
<th>8-issue 64-entry processor</th>
<th>16-issue 128-entry processor</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>CAM Bit-map RAM Proposed design</td>
<td>CAM Bit-map RAM Proposed design</td>
<td>CAM Bit-map RAM Proposed design</td>
</tr>
<tr>
<td>Integer avg. (mw)</td>
<td>199 43 52</td>
<td>839 214 172</td>
<td>4457 1392 635</td>
</tr>
<tr>
<td>FP avg. (mw)</td>
<td>196 41 52</td>
<td>833 210 169</td>
<td>4439 1373 632</td>
</tr>
<tr>
<td>Media avg. (mw)</td>
<td>204 43 52</td>
<td>865 216 173</td>
<td>4570 1393 635</td>
</tr>
<tr>
<td>Total avg. (mw)</td>
<td>199 42 52</td>
<td>842 213 171</td>
<td>4474 1385 634</td>
</tr>
</tbody>
</table>
Circuit Latency Analysis (1/2): Wakeup Delay Comparison

The proposed scheme is 20% to 50% faster than the bit-map RAM and 60% to 90% faster than the CAM scheme.
The IPns is 2.5 times of the CAM scheme and 1.2 times of the bit map RAM scheme.

IPns: # of instructions per ns. (IPC from simulator; clock period from Hspice assuming wakeup+select is the critical path.)
Area Trade-off

For a 128-entry processor, the area of the proposed scheme is only about 20% to 50% of the RAM scheme.

Larger issue rate needs more read ports; hence CAM cells get larger.
Other Optimizations

- Segmented CAMs using wakeup locality
- Many other possibilities for segmenting the CAM
  - Program order prefix
  - Source rename identifier prefix
  - Source rename identifier postfix
Wakeup Locality

The graph shows the percentage of wakeups for different distances, with bars representing different instruction distances:

- 1 - 16 instructions
- 17 - 32 instructions
- 33 - 64 instructions
- 65 - 127 instructions

The applications listed from top to bottom are:
- gzip
- place
- route
- vpr
- gcc
- mcf
- parser
- vortex
- bzip2
- wupwise
- mgrid
- applu
- mesa
- art
- quake
- ammp
- apic
- adpcm
- epic
- g721
- mpeg2
- pegwit
Activate two reduced + two Full

Segmented CAMs
Presentation Outline

- Introduction & Motivation
- Dynamic Scheduler for Superscalar Processors
- Wakeup Logic Designs
- Experimental Evaluation
- Conclusions
Conclusions

- Trade expensive tag RAM with combinational circuit by predecoding the source tag.
  - Faster
  - Less power used
  - Scalable design

- Other segmented CAM schemes
  - Wakeup locality
  - Rename identifier prefix/postfix
The End