Instruction-Level Parallel Processors

Objective:
- Executing two or more instructions in parallel

4.1 Evolution and overview of ILP-processors
4.2 Dependencies between instructions
4.3 Instruction scheduling
4.4 Preserving sequential consistency
4.5 The speed-up potential of ILP-processing

How do we increase the number of instructions to be executed?

Traditional von Neumann processors (sequential issue, sequential execution)
Scalar ILP-processors (sequential issue, parallel execution)
Superscalar ILP-processors (parallel issue, parallel execution)

Parallelism of instruction execution
Parallelism of instruction issue

Typical implementation
Non-parallel processors
Processors with multiple functional units and pipelined processors
VLIM and superscalar processors containing multiple pipelined EU's

Processor performance

Improve CPU performance by
- Increasing clock rates
  - (CPU running at 2GHz)
- Increasing the number of instructions to be executed in parallel
  - (executing 6 instructions at the same time)

Time and Space parallelism

Pipeline (assembly line)

Result of pipeline (e.g.)

<table>
<thead>
<tr>
<th>t1</th>
<th>t2</th>
<th>t3</th>
<th>t4</th>
<th>t5</th>
<th>t6</th>
<th>t7</th>
</tr>
</thead>
<tbody>
<tr>
<td>F1</td>
<td>D1</td>
<td>E1</td>
<td>W1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>F2</td>
<td>D2</td>
<td>E2</td>
<td>W2</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>F3</td>
<td>D3</td>
<td>E3</td>
<td>W3</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>F4</td>
<td>D4</td>
<td>E4</td>
<td>W4</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>F5</td>
<td>D5</td>
<td>E5</td>
<td>W5</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>D6</td>
<td>E6</td>
<td>W6</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>D7</td>
<td>E7</td>
<td>W7</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>F8</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
From Sequential instructions to parallel execution

- Dependencies between instructions
- Instruction scheduling
- Preserving sequential consistency

4.2 Dependencies between instructions

- Instructions often depend on each other in such a way that a particular instruction cannot be executed until a preceding instruction or even two or three preceding instructions have been executed.
  - 1 Data dependencies
  - 2 Control dependencies
  - 3 Resource dependencies

4.2.1 Data dependencies

- Read after Write
- Write after Read
- Write after Write
- Recurrences

Data dependencies in straight-line code (RAW)

- RAW dependencies
  - r1: load r1, a
  - r2: add r2, r1, r1
- flow dependencies
- true dependencies
- cannot be abandoned
Data dependencies in straight-line code (WAR)

- WAR dependencies
  - i1: mul r1, r2, r3
  - r2: add r2, r4, r5
- anti-dependencies
- false dependencies
- can be eliminated through register renaming
  - i2: mul r1, r2, r3
  - r2: add r6, r4, r5
- by using compiler or ILP-processor

Data dependencies in straight-line code (WAW)

- WAW dependencies
  - i1: mul r1, r2, r3
  - r2: add r1, r4, r5
- output dependencies
- false dependencies
- can be eliminated through register renaming
  - i2: mul r1, r2, r3
  - r2: add r6, r4, r5
  - by using compiler or ILP-processor

Data dependencies in loops

```java
for (int i=2; i<10; i++) {
    x[i] = a*x[i-1] + b
}
```

- cannot be executed in parallel

Data dependency graphs

- i1: load r1, a
- i2: load r2, b
- i3: load r3, r1, r2
- i4: mul r1, r2, r4;
- i5: div r1, r2, r4

4.2.2 Control dependencies

- mul r1, r2, r3
- jz  zproc
- :
- zproc: load r1, x
- :
- actual path of execution depends on the outcome of multiplication
- impose dependencies on the logical subsequent instructions

Control Dependency Graph
Branches?
Frequency and branch distance

- Expected frequency of (all) branch
  - general-purpose programs (non scientific): 20-30%
  - scientific programs: 5-10%
- Expected frequency of conditional branch
  - general-purpose programs: 20%
  - scientific programs: 5-10%
- Expected branch distance (between two branch)
  - general-purpose programs: every 3rd-5th instruction, on average, to be a conditional branch
  - scientific programs: every 10th-20th instruction, on average, to be a conditional branch

Impact of Branch on instruction issue

- fig. 4.14

4.2.3 Resource dependencies

- An instruction is resource-dependent on a previously issued instruction if it requires a hardware resource which is still being used by a previously issued instruction.
  - e.g.
    - div r1, r2, r3
    - div r4, r2, r5

4.3 Instruction scheduling

- scheduling or arranging two or more instruction to be executed in parallel
  - Need to detect code dependency (detection)
  - Need to remove false dependency (resolution)
- a means to extract parallelism
  - instruction-level parallelism which is implicit, is made explicit
- Two basic approaches
  - Static: done by compiler
  - Dynamic: done by processor

Instruction Scheduling:

ILP-instruction scheduling

- Static scheduling boosted by parallel code optimization
  - Performed entirely by the compiler
  - The processor receives dependency-free and optimized code for parallel execution
  - Typical for ULPs and a few pipelined processors (e.g., MIPS)
  - Early RISC processors (e.g. DEC VAX, IBM 360/59 etc.)
  - Unusual practice for pipelined and superscalar processors (e.g. ARM, PowerPC, IA-32, Itanium and SPARC)

- Dynamic scheduling without static parallel code optimization
  - Performed entirely by the processor
  - The code is not optimized for parallel execution.
  - The processor detects and resolves dependencies on its own

- Dynamic scheduling boosted by static parallel code optimization
  - Performed by the processor in conjunction with the parallel optimizing compiler
4.4 Preserving sequential consistency

- Care must be taken to maintain the logical integrity of the program execution.
- Parallel execution mimics sequential execution as far as the logical integrity of program execution is concerned.
- E.g.
  - `add r5, r6, r7`
  - `div r1, r2, r3`
  - `jz somewhere`

4.5 The speed-up potential of ILP-processing

- Parallel instruction execution may be restricted by data, control, and resource dependencies.
- Potential speed-up when parallel instruction execution is restricted by true data and control dependencies:
  - General purpose programs: about 2
  - Scientific programs: about 2-4
- Why are the speed-up figures so low?
  - Basic block (a low-efficiency method used to extract parallelism)

Two other methods for speed up

- Potential speed-up embodied in loops
  - Amount to a figure of 50 (on average speed up)
    - Unlimited resources (about 50 processors, and about 400 registers)
    - Ideal schedule
- Appropriate handling of control dependencies
  - Amount to 10 to 100 times speed up
    - Assuming perfect oracles that always pick the right path for conditional branch
- Control dependencies are the real obstacle in utilizing instruction-level parallelism!

What do we do without a perfect oracle?

- Execute all possible paths in conditional branch
  - There are $2^N$ paths for N conditional branches
  - Pursuing an exponential increasing number of paths would be an unrealistic approach.
- Make your Best Guess
  - Branch prediction
  - Pursuing both possible paths but restrict the number of subsequent conditional branches
  - (More more CH. 8)
How close real systems can come to the upper limits of speed-up?

- ambitious processor can expect to achieve speed-up figures of about
  - 4 for general purpose programs
  - 10-20 for scientific programs
- an ambitious processor:
  - predicts conditional branches
  - has 256 integer and 256 FP register
  - eliminates false data dependencies through register renaming,
  - performs a perfect memory disambiguation
  - maintains a gliding instruction window of 64 items