

## Superscalar Processors

### Ch 13

Limitations, Hazards  
Instruction Issue Policy  
Register Renaming  
Branch Prediction

24.11.1999 Copyright Teemu Kerola 1999 1

## Superscalar Processing <sup>(5)</sup>

- Basic idea: more than one instruction completion per cycle
- Aimed at speeding up scalar processing
  - use multiple pipelines and not more phases Fig. 13.2
- Many instructions in execution phase simultaneously
  - need parallelism also in earlier & later phases
- Multiple pipelines Fig. 13.1
  - question: when can instruction be executed?
- Fetch many instructions at the same time
  - memory access must not be bottleneck

24.11.1999 Copyright Teemu Kerola 1999 2

### Why couldn't we execute this instruction right now? <sup>(3)</sup>

Fig. 13.3

- (True) Data Dependency (datariippuvuus)

```
load r4, salary(r6)
mul r2, r4, r10
```
- Procedural or Control Dependency (kontrolliriippuvuus)
  - even more costlier than with normal pipeline
  - now may waste more than one instruction!
- Resource Conflict (resurssikonflikti)
  - there is no available circuit right now
  - memory buffer, FP adder, register file port

24.11.1999 Copyright Teemu Kerola 1999 3

### Why couldn't we execute this instruction right now?

- Name dependency (nimiriippuvuus)
  - two instruction use the same data item
    - register or in memory
  - no value passed from one instruction to another
  - instructions have all their correct data available
  - each individual result is the one intended
  - overall result is not the one intended
  - two cases: Output Dependency & Antidependency
    - (outputriippuvuus) (antiriippuvuus)
  - what if there are aliases? (two names, same data)

24.11.1999 Copyright Teemu Kerola 1999 4

### Output Dependency? <sup>(1)</sup>

- Some earlier instruction has not yet finished writing from the same location that we want to write to
- Need to preserve order

```
read r1, sum
add r2, r1, r3
add r1, r4, r5
```

Want to have sum of r4 and r5 in r1

24.11.1999 Copyright Teemu Kerola 1999 5

### Antidependency <sup>(1)</sup>

Some earlier instruction has not yet finished reading from the same location that we want to write to  
Need to preserve order

```
mv r2, r1
add r1, r4, r5
```

Want to have original value of r1 in r2

24.11.1999 Copyright Teemu Kerola 1999 6

### Machine Parallelism (2)

- Instruction-level parallelism
  - How much parallelism is there
  - Theoretical maximum
- Machine parallelism
  - How much parallelism is achieved by any specific machine or architecture?
  - At most as much as instruction-level parallelism
    - dependencies?
    - physical resources?
    - not optimized (stupid) design?

24.11.1999 Copyright Teemu Kerola 1999 7

### Superscalar Processor

Fig. 13.6

- Instruction dispatch
  - get next available executable instruction from instruction stream
- Window of execution
  - all instructions that are considered to be issued
- Instruction issue
  - allow instruction to start execution
  - execution and completion phase should continue now with no stalls
- Instruction reorder and commit (retiring)
  - hopefully all system state changes here!
  - last chance to change order or abandon results

24.11.1999 Copyright Teemu Kerola 1999 8

### Instruction Dispatch

- Whenever there are both
  - available slots in window of execution
  - ready instructions from prefetch or branch prediction buffer

24.11.1999 Copyright Teemu Kerola 1999 9

### Window of Execution

- Bigger is better
  - easier to find a good candidate that can be issued right now
  - more work to figure out all dependencies
  - too small value will limit machine parallelism significantly
    - E.g., 6<sup>th</sup> instruction could be issued, but only 4 next ones are even considered

24.11.1999 Copyright Teemu Kerola 1999 10

### Instruction Issue

- Select next instruction(s) for execution
- Check first everything so that execution can proceed with no stalls (stopping) to the end
  - resource conflicts
  - data dependencies
  - control dependencies
  - output dependencies
  - antidependencies

24.11.1999 Copyright Teemu Kerola 1999 11

### Instruction Issue Policies (3)

- Instruction fetch policy
  - constraints on how many instruction allowed to be considered to be dispatched at a time
    - 2 instructions fetched and decoded at a time
      - ⇒ both must be dispatched before next 2 fetched
- Instruction execution policy
  - constraints on which order dispatched instructions may start execution
- Completion policy
  - constraints of which order completions can occur

24.11.1999 Copyright Teemu Kerola 1999 12

### Example Issue Policy <sup>(6)</sup>

- **In-order issue with in-order completion**
  - same as purely sequential execution Fig. 13.4 (a)
  - no instruction window needed
  - instruction issued only in original order
    - many can be issued at the same time
  - instructions completed only in original order
    - many can be completed at the same time
  - check before issue:
    - resource conflicts, data & control dependencies
    - execution time, so that completions occur in order: wait long enough that earlier instructions will complete first

24.11.1999 Copyright Teemu Kerola 1999 13

### Example Issue Policy <sup>(5)</sup>

- **In-order issue with out-of-order completion**
  - issue in original order Fig. 13.4 (b)
    - many can be issued at the same time
  - no instruction window needed
  - allow executions complete before those of earlier instructions
  - Check before issue:
    - resource conflicts, data & control dependencies
    - output dependencies: wait long enough to solve them

24.11.1999 Copyright Teemu Kerola 1999 14

The real superscalar processor

### Example Issue Policy <sup>(5)</sup>

- **Out-of-order issue with out-of-order completion**
  - issue in any order Fig. 13.4 (c)
    - many can be issued at the same time
  - **instruction window** for dynamic instruction scheduling
  - allow executions complete before those of earlier instructions
  - Check before issue:
    - resource conflicts, data & control dependencies
    - output dependencies
    - antidependencies: must wait for earlier instructions issued later to pick up arguments before overwriting them

24.11.1999 Copyright Teemu Kerola 1999 15

### Get Rid of Name Dependencies <sup>(2)</sup>

- **Problem: independent data stored in locations with the same name**
  - often a storage conflict: same register used for two different purposes
  - results in wait stages (pipeline stalls, “bubbles”)
- **Cure: register renaming**
  - actual registers may be different than named registers
  - actual registers allocated dynamically to named registers
  - allocate them so that name dependencies are avoided

24.11.1999 Copyright Teemu Kerola 1999 16

### Register Renaming <sup>(3)</sup>

Output dependency: I3 can not complete before I1 has completed first:

Antidependency: I3 can not complete before I2 has read value from R3:

Rename registers to hardware registers R3a, R3b, R3c, R4b, R5a, R7b

No name dependencies now:

R3:=R3 + R5; (I1)  
 R4:=R3 + 1; (I2)  
 R3:=R5 + 1; (I3)  
 R7:=R3 + R4; (I4)

R3b:=R3a + R5a (I1)  
 R4b:=R3b + 1 (I2)  
 R3c:=R5a + 1 (I3)  
 R7b:=R3c + R4b (I4)

- Drawback: need more registers
- Why R3a & R3b?

24.11.1999 Copyright Teemu Kerola 1999 17

### Superscalar Implementation <sup>(6)</sup>

- **Fetch strategy** Fig. 13.6
  - prefetch, branch prediction
- **Dependency check logic**
  - forwarding circuits to transfer dependency data directly instead via registers or memory
- Multiple functional units (pipelines)
- Effective memory hierarchy to service many memory accesses simultaneously
- Logic to issue multiple instruction simultaneously
- Logic to commit instruction in correct order

24.11.1999 Copyright Teemu Kerola 1999 18

### Overall Gain from Superscalar Implementation

- Seem the effect of ... Fig. 13.5
  - renaming ⇒ right graph
  - window size ⇒ color of vertical bar
  - out-of-order issue ⇒ “base” machine
  - duplicated
    - data cache access ⇒ “+ld/st”
    - ALU ⇒ “ALU”
    - both ⇒ “both”
- Max speed-up about 4

24.11.1999 Copyright Teemu Kerola 1999 19

24.11.1999 Copyright Teemu Kerola 1999 20

### Example: PowerPC 601 Architecture (2)

- General RISC organization
  - instruction formats Fig. 10.9
  - 3 execution units Fig. 13.10
- Logical view Fig. 13.11
  - 4 instruction window for issue
  - each execution unit picks up next one for it whenever there is room for new instruction
  - integer instructions issued only when 1st in queue

24.11.1999 Copyright Teemu Kerola 1999 21

### PowerPC 601 Pipelines (4)

- Instruction pipelines Fig. 13.12
  - all state changes in final “Write Back” phase
  - up to 3 instruction can be dispatched at the same time, and issued right after that in each pipeline if no dependencies exist
    - dependencies solved by stalls
  - ALU ops place their result in one of 8 condition code field in condition register
    - up to 8 separate conditions active concurrently

24.11.1999 Copyright Teemu Kerola 1999 22

### PowerPC 601 Branches (4)

- Zero cycle branches
  - branch target addresses computed already in lower dispatch buffers
    - before dispatch or issue!
  - Easy: unconditional branches (jumps) or branch on already resolved condition code field
  - otherwise
    - conditional branch backward: guess taken
    - conditional branch forward: guess not taken
    - if speculation ends up wrong, cancel conditional instructions in pipeline before write-back
    - speculate only on one branch at a time

24.11.1999 Copyright Teemu Kerola 1999 23

### PowerPC 601 Example

- Conditional branch example
  - Original C code Fig. 13.13 (a)
  - Assembly code Fig. 13.13 (b)
    - predict branch not taken
  - Correct branch prediction Fig. 13.14 (a)
  - Incorrect branch prediction Fig. 13.14 (b)

24.11.1999 Copyright Teemu Kerola 1999 24

### PowerPC 620 Architecture

- 6 execution units [Fig. 4.25]
- Up to 4 instructions dispatched simultaneously
- Reservation stations to store dispatched instructions and their arguments [HePa96] Fig. 4.49
  - kind of rename registers also!

24.11.1999 Copyright Teemu Kerola 1999 25

### PowerPC 620 Rename Registers (7)

- Rename registers to store results not yet committed [HePa96] Fig. 4.49
  - normal uncompleted and speculative instructions
  - 8 int and 12 FP extra rename registers
    - in same register file as normal registers
  - results copied to normal registers at commit
  - information on what to do at commit is in completion unit in reorder buffers
- Instruction completes (commits) from completion unit reorder buffer once all previous instructions are committed
  - max 4 instructions can commit at a time

24.11.1999 Copyright Teemu Kerola 1999 26

### PowerPC 620 Speculation

- Speculation on branches
  - 256-entry branch target buffer
    - two-way set-associative
  - 2048-entry branch history table
    - used when branch target buffer misses
  - speculation on max 4 unresolved branches

24.11.1999 Copyright Teemu Kerola 1999 27

### -- End of Chapter 13: Superscalar --

[Fig. 4.49]

FIGURE 4.49 The PowerPC 620 uses six different functional units, one to fetch its own reservation stations and a 16-entry reorder buffer, embedded in the hardware compiler cache.

(Hennessy-Patterson, Computer Architecture, 2nd Ed, 1996)

24.11.1999 Copyright Teemu Kerola 1999 28