EN164: Design of Computing Systems
Topic 06.b: Superscalar Processor Design

Professor Sherief Reda
http://scale.engin.brown.edu
Electrical Sciences and Computer Engineering
School of Engineering
Brown University
Spring 2012

[ material from Patterson & Hennessy, 4th ed & Inside the Machine by J. Stokes]
Superscalar and VLIW reminder

Superscalar

static VOID _DEFUN(_mor_nu),
  struct _reent *ptr_AND
  register size_t 
  
Normal Compiler

RISC-Level Object Code

IM1 = I-1
IM2 = I-2
IM3 = I-3
T1 = LOAD...
T3 = 2*T1

Scheduling and Operation Independence-
Recognizing Hardware

The Same "Normal"
Source Code in
Both Cases

GFC LGC

Compile-Time

Run-Time

WIDE RISC-Level Object Code

Normal Compiler, PLUS
Scheduling and Operation
Independence-
Recognizing Software

The Same ILP Hardware in
Both Cases

VLIW

static VOID _DEFUN(_mor_nu),
  struct _reent *ptr_AND
  register size_t 
  

Superscalar processors

• CPU decides whether to issue 0, 1, 2, … each cycle
  – Avoiding structural and data hazards
• Avoids the need for compiler scheduling
  – Though it may still help
  – Code semantics ensured by the CPU
  – No need to re-compile existing code
  – Static superscalar (in order execution) and dynamic superscalar (out of order execution)
Challenges in superscalar processor design

Challenging code snippets:

```
DIV.F $f0, $f1, $f2
ADD.F $f0, $f1, $f3
DIV.F $f0, $f1, $f2
ADD.F $f2, $f0, $f1
ADD.F $f2, $f4, $f5
DIV.F $f0, $f1, $f2
ADD.F $f2, $f0, $f1
ADD.F $f4, $f4, $f3
DIV.F $f0, $f1, $f2
BEQ $f0, $0, L
... L: ADD $t0, $t1, $t2
...```

![Diagram of superscalar processor design](image-url)
Summary of challenges

1. Structural hazards on RF $\rightarrow$ increase number of ports
   $\rightarrow$ RF size grows squarely with number of port
2. Large stall penalties for RAW hazards because of deep pipelines
3. WAR problems because out-of-order execution
4. WAW name dependency hazards because different instructions take different number of cycles and because of out-of-order execution
5. *Speculation* problems arising from branch prediction
6. Out of order execution problems (e.g., WAW, exceptions / interrupts)
Possible solutions

1. Structural hazards on RF → increase number of ports → RF size grows squarely with number of port → split register file into integer RF and floating point RF

2. Large stall penalties for RAW hazards → out of order execution

3. WAR / WAW problems → register renaming

4. Speculation problems → execute out-of-order but commit in order

5. Exceptions/interrupts → execute out-of-order but commit in order
Superscalar in x86 architecture

- First introduced by Intel in 1976 with the 16-bit processor 8086. Continues with us to this date.
- Instructions variable length (1-17 bytes)
- Small number of registers (mainly 8)
- Instructions kept on expanding with new processor releases
x86 assembly language examples

# move the 4 bytes in memory at the address contained in EBX into EAX
mov eax, [ebx]

# move the 4 bytes of data at address ESI+4*EBX into EDX
mov edx, [esi+4*ebx]

# eax = eax + ebx
add eax, ebx

# add ebx to memory content at address memory_location
add memory_location, ebx

# add 10 the data at address EAX
add [eax], 10

# push content of eax into top of stack
push eax

# compare two registers, register – memory or memory memory
cmp eax, ebx

cmp [eax], [ebx]

# conditional jumps
jl, jeq, jg, jge, jle, jne

Instructions vary in length from 1 to 17 bytes with an average of 3 bytes / instruction
Pentium

Introduced 1993
Technology 0.8 micron
Transistors: 3.1 million
Speed: 66 Mhz
# stages: 5
Issue width: 2-way static

First superscalar (static) design from Intel
Pentium Fetch and Decode

<table>
<thead>
<tr>
<th>IF</th>
<th>ID1</th>
<th>ID2</th>
<th>EX</th>
<th>WB</th>
</tr>
</thead>
</table>

Pentium five-stage pipeline

- **Fetch stage:**
  - Instructions are fetched in a buffer, boundaries detected and marked.
  - Marked instructions are aligned and sent to decode stage.
- **Decode stage 1 (ID1)** does typical decoding and checks if the instruction is a branch (and does static branch prediction for $1^{st}$ encounters and branch prediction using BHT and BTB for $2^{nd}$ encounters)
- **Decode stage 2 (ID2)** does address calculation to support complex memory addressing modes of x86
Pentium backend

- Two single-stage asymmetric integer ALUs
- One three-stage floating point
- Control unit examines pairs of instructions and **statically** issues them simultaneously if there are no hazards and can be supported by both ALUs; otherwise, only one is issued.
- Results of two instructions are put back in order when completed and written back to register file or memory
Pentium processor in perspective

- CISC ISA bloated support (instruction fetch, instruction decode and complex instruction support) almost occupied 30% of die area

- RISC competitors (from IBM, MIPS and Sun) required much less overhead for fetch and decode → used the available area for larger on-chip memory (caches) and more execution units
Pentium Pro

P6 microarchitecture
Introduced 1995
Technology: 0.5 – 0.35 micron
Transistors: 5.5 million
Speed: 150 - 200 Mhz
#stages: 11
Issue width: 5-way dynamic

Intel Core architecture is a direct descendant of P6 microarchitecture.

First Intel x86 superscalar with dynamic scheduler (i.e., out-of-order execution)
## Fetch / decode (in-order)

<table>
<thead>
<tr>
<th>IF1</th>
<th>IF2</th>
<th>IF3</th>
<th>ID1</th>
<th>ID2</th>
<th>RAT</th>
<th>ROB</th>
<th>DIS</th>
<th>EX</th>
<th>RET1</th>
<th>RET2</th>
</tr>
</thead>
</table>

Pentium Pro 11-stage pipeline

1. **IF1**: fetch instruction from memory into a buffer
2. **IF2**: mark boundaries of instruction and dynamic branch prediction
3. **IF3**: Align instructions for decoders
4. **ID1**: Translate complex x86 instructions into internal simpler uop RISC instructions
5. **ID2**: uop RISC instructions enters into queue

![Diagram of Pentium Pro 11-stage pipeline]

- Instruction memory
- 16-byte fetch buffer
- Simple decoder
- Simple decoder
- Complex decoder
- 6-entry uop queue
RAT and ROB stages

<table>
<thead>
<tr>
<th>IF1</th>
<th>IF2</th>
<th>IF3</th>
<th>ID1</th>
<th>ID2</th>
<th>RAT</th>
<th>ROB</th>
<th>DIS</th>
<th>EX</th>
<th>RET1</th>
<th>RET2</th>
</tr>
</thead>
</table>

Pentium Pro 11-stage pipeline

6. RAT: registers are renamed into 40 internal registers to eliminate false dependencies like WAW and WAR. A Register Alias Table (RAT) holds the mapping between architectural registers and hidden registers.

7. The ROB is a circular buffer with 40 entries and holds the RISC uops in strict program order.

6-entry uop queue

RAT
logical $\rightarrow$ physical

ROB
40-entry circular buffer
8. **DIS**: Dynamic scheduler checks instructions in ROB and dispatch (or issue) them to the reservation stations of the execution units if (1) the execution unit is available (no structural hazard) and (2) values of its operands are available (no RAW hazard).

9. **EX**: integer pipelined 1-cycle throughput. Floating pipelined for most instructions. DIV and SQRT take 38-69 cycles to finish.
10. RET1: Results from EX stages are written back to ROB buffer(s)
11. RET2: Results from ROB buffer are committed in order to register file as long as retiring instruction is no longer speculative
Pentium II / III

P6 microarchitecture
Introduced 1997
Technology: 0.35 micron
Transistors: 7.5 million
Speed: 233 - 300 Mhz

- Similar microarchitecture to Pentium Pro
- Added MMX support for Single-Instruction Multiple Data (SIMD) integer ALU
- Increased on chip-memory size

P6 microarchitecture
Introduced: 1999
Technology: 0.25 – 0.18 micron
Transistors: 9.5 million
Speed: 600 Mhz – 1.3 Ghz

- Added SSE support for Single-Instruction Multiple Data (SIMD) single-precision floating point ALU
- Increased on chip-memory size
Limits to superscalar processors

- Programs have real dependencies that limit ILP
- Some dependencies are hard to eliminate
  - e.g., pointer aliasing
- Some parallelism is hard to expose
  - Limited window size during instruction issue
- Memory delays and limited bandwidth
  - Hard to keep pipelines full
- Speculation helps if done well, but heavy penalties for incorrect branch predictions
- HW complexity increases power consumption & die area