Dealing with Branches

- Multiple Streams
- Prefetch Branch Target
- Loop buffer
- Branch prediction
- Delayed branching
Multiple Streams

- Have two pipelines
- Prefetch each branch into a separate pipeline
- Use appropriate pipeline

- Leads to bus & register contention
- Multiple branches lead to further pipelines being needed
Prefetch Branch Target

- Target of branch is prefetched in addition to instructions following branch
- Keep target until branch is executed
- Used by IBM 360/91
Loop Buffer

- Very fast memory
- Maintained by fetch stage of pipeline
- Check buffer before fetching from memory
- Very good for small loops or jumps
- Used by CRAY-1
Loop Buffer Diagram

Branch address

8

Loop Buffer (256 bytes)

Instruction to be decoded in case of hit

Most significant address bits compared to determine a hit

Introduction to Computer Architecture and Organization
Lesson 17 – Slide 5/21
Branch Prediction (1)

- Predict never taken
  - Assume that jump will not happen
  - Always fetch next instruction
  - 68020 & VAX 11/780
  - VAX will not prefetch after branch if a page fault would result (O/S v CPU design)

- Predict always taken
  - Assume that jump will happen
  - Always fetch target instruction
Branch Prediction (2)

• Predict by Opcode
  – Some instructions are more likely to result in a jump than others
  – Can get up to 75% success
• Taken/Not taken switch
  – Based on previous history
  – Good for loops
Branch Prediction (3)

- Delayed Branch
  - Do not take jump until you have to
  - Rearrange instructions
Branch Prediction State Diagram

Introduction to Computer Architecture and Organization
Lesson 17 – Slide 10/21
Intel 80486 Pipelining

- Fetch
  - From cache or external memory
  - Put in one of two 16-byte prefetch buffers
  - Fill buffer with new data as soon as old data consumed
  - Average 5 instructions fetched per load
  - Independent of other stages to keep buffers full
- Decode stage 1
  - Opcode & address-mode info
  - At most first 3 bytes of instruction
  - Can direct D2 stage to get rest of instruction
- Decode stage 2
  - Expand opcode into control signals
  - Computation of complex address modes
- Execute
  - ALU operations, cache access, register update
- Writeback
  - Update registers & flags
  - Results sent to cache & bus interface write buffers
(a) No Data Load Delay in the Pipeline

(b) Pointer Load Delay

(c) Branch Instruction Timing
### Pentium 4 Registers

#### (a) Integer Unit

<table>
<thead>
<tr>
<th>Type</th>
<th>Number</th>
<th>Length (bits)</th>
<th>Purpose</th>
</tr>
</thead>
<tbody>
<tr>
<td>General</td>
<td>8</td>
<td>32</td>
<td>General-purpose user registers</td>
</tr>
<tr>
<td>Segment</td>
<td>6</td>
<td>16</td>
<td>Contain segment selectors</td>
</tr>
<tr>
<td>Flags</td>
<td>1</td>
<td>32</td>
<td>Status and control bits</td>
</tr>
<tr>
<td>Instruction Pointer</td>
<td>1</td>
<td>32</td>
<td>Instruction pointer</td>
</tr>
</tbody>
</table>

#### (b) Floating-Point Unit

<table>
<thead>
<tr>
<th>Type</th>
<th>Number</th>
<th>Length (bits)</th>
<th>Purpose</th>
</tr>
</thead>
<tbody>
<tr>
<td>Numeric</td>
<td>8</td>
<td>80</td>
<td>Hold floating-point numbers</td>
</tr>
<tr>
<td>Control</td>
<td>1</td>
<td>16</td>
<td>Control bits</td>
</tr>
<tr>
<td>Status</td>
<td>1</td>
<td>16</td>
<td>Status bits</td>
</tr>
<tr>
<td>Tag Word</td>
<td>1</td>
<td>16</td>
<td>Specifies contents of numeric registers</td>
</tr>
<tr>
<td>Instruction Pointer</td>
<td>1</td>
<td>48</td>
<td>Points to instruction interrupted by exception</td>
</tr>
<tr>
<td>Data Pointer</td>
<td>1</td>
<td>48</td>
<td>Points to operand interrupted by exception</td>
</tr>
</tbody>
</table>
### EFLAGS Register

<table>
<thead>
<tr>
<th>31</th>
<th>21</th>
<th>16/15</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>D</td>
<td>I</td>
<td>V</td>
<td>Y</td>
</tr>
<tr>
<td>P</td>
<td>I</td>
<td>V</td>
<td>Y</td>
</tr>
<tr>
<td>A</td>
<td>V</td>
<td>C</td>
<td>M</td>
</tr>
<tr>
<td>R</td>
<td>N</td>
<td>O</td>
<td>I</td>
</tr>
<tr>
<td>T</td>
<td>O</td>
<td>P</td>
<td>L</td>
</tr>
<tr>
<td>F</td>
<td>D</td>
<td>F</td>
<td>F</td>
</tr>
<tr>
<td>F</td>
<td>I</td>
<td>F</td>
<td>F</td>
</tr>
<tr>
<td>F</td>
<td>S</td>
<td>Z</td>
<td>F</td>
</tr>
<tr>
<td>Z</td>
<td>A</td>
<td>F</td>
<td>P</td>
</tr>
<tr>
<td>F</td>
<td>P</td>
<td>F</td>
<td>C</td>
</tr>
<tr>
<td>F</td>
<td>C</td>
<td>F</td>
<td>F</td>
</tr>
</tbody>
</table>

**Legend:**

- **ID** = Identification flag
- **VIP** = Virtual interrupt pending
- **VIF** = Virtual interrupt flag
- **AC** = Alignment check
- **VM** = Virtual 8086 mode
- **RF** = Resume flag
- **NT** = Nested task flag
- **IOPL** = I/O privilege level
- **OF** = Overflow flag
- **DF** = Direction flag
- **IF** = Interrupt enable flag
- **TF** = Trap flag
- **SF** = Sign flag
- **ZF** = Zero flag
- **AF** = Auxiliary carry flag
- **PF** = Parity flag
- **CF** = Carry flag
Control Registers

PCE = Performance Counter Enable
PGE = Page Global Enable
MCE = Machine Check Enable
PAE = Physical Address Extension
PSE = Page Size Extensions
DE = Debug Extensions
TSD = Time Stamp Disable
PVI = Protected Mode Virtual Interrupt
VME = Virtual 8086 Mode Extensions
PCD = Page-level Cache Disable
PWT = Page-level Writes Transparent

PG = Paging
CD = Cache Disable
NW = Not Write Through
AM = Alignment Mask
WP = Write Protect
NE = Numeric Error
ET = Extension Type
TS = Task Switched
EM = Emulation
MP = Monitor Coprocessor
PE = Protection Enable

Introduction to Computer Architecture and Organization
Lesson 17 – Slide 15/21
MMX Register Mapping

- MMX uses several 64 bit data types
- Use 3 bit register address fields
  - 8 registers
- No MMX specific registers
  - Aliasing to lower 64 bits (mantissa) of existing floating point registers
MMX Register Mapping Diagram

Introduction to Computer Architecture and Organization
Lesson 17 – Slide 17/21
Pentium Interrupt Processing

- Interrupts
  - Maskable
  - Nonmaskable
- Exceptions
  - Processor detected
  - Programmed
- Interrupt vector table
  - Each interrupt type assigned a number
  - Index to vector table
  - 256 * 32 bit interrupt vectors
- 5 priority classes
PowerPC User Visible Registers

Fixed-Point Unit

<table>
<thead>
<tr>
<th>0</th>
<th>63</th>
</tr>
</thead>
<tbody>
<tr>
<td>R0</td>
<td></td>
</tr>
</tbody>
</table>

R31

0 31

Exception

Branch Processing Unit

<table>
<thead>
<tr>
<th>0</th>
<th>31</th>
</tr>
</thead>
<tbody>
<tr>
<td>Condition</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>0</th>
<th>63</th>
</tr>
</thead>
<tbody>
<tr>
<td>Link</td>
<td></td>
</tr>
<tr>
<td>Count</td>
<td></td>
</tr>
</tbody>
</table>

Floating-Point Unit

<table>
<thead>
<tr>
<th>0</th>
<th>63</th>
</tr>
</thead>
<tbody>
<tr>
<td>FPR0</td>
<td></td>
</tr>
</tbody>
</table>

FPR31

0 31

FPSCR

Introduction to Computer Architecture and Organization

Lesson 17 – Slide 19/21
PowerPC Register Formats

SO = Summary overflow: set to 1 to indicate an overflow occurred during the execution of an instruction; remains 1 until reset by software
OV = Overflow: set to 1 to indicate an overflow occurred during the execution of an instruction; reset to 0 by next instruction if there is no overflow
CA = Carry: set to 1 to indicate carry out of bit 0 during the execution of an instruction
Byte Count = Specifies number of bytes to be transferred by Load/Store String indexed instruction

(a) Fixed-Point Exception Register (XER)

(b) Condition Register

Introduction to Computer Architecture and Organization

Lesson 17 – Slide 20/21
Pipeline, Superpipeline, Superscalar